NAME (PRINT): _____

Last / Surname First / Given Name

STUDENT #: _____ **SIGNATURE:** _____

UNIVERSITY OF TORONTO MISSISSAUGA
APRIL 2016 FINAL EXAMINATION
CSC258H5S
Computer Organization
Larry Yueli Zhang
Duration 3 hours
Aids: None

The University of Toronto Mississauga and you, as a student, share a commitment to academic integrity. You are reminded that you may be charged with an academic offence for possessing any unauthorized aids during the writing of an exam. Clear, sealable, plastic bags have been provided for all electronic devices with storage, including but not limited to: cell phones, SMART devices, tablets, laptops, calculators, and MP3 players. Please turn off all devices, seal them in the bag provided, and place the bag under your desk for the duration of the examination. You will not be able to touch the bag or its contents until the exam is over.

If, during an exam, any of these items are found on your person or in the area of your desk other than in the clear, sealable, plastic bag, you may be charged with an academic offence. A typical penalty for an academic offence may cause you to fail the course.

*Please note, once this exam has begun, you **CANNOT** re-write it.*

This final examination consists of a total of 100 marks, for 9 questions on 16 pages (double-sided, including this one, and not including the reference sheets). When you receive the signal to start, please make sure that your copy of the examination is complete. You may tear off the reference sheets (the last two sheets).

Each question is labelled with the suggested amount of time to solve it. If you use anywhere other than the provided space for your solutions, indicate clearly the part of your work that you want to be marked. Precise and concise answers will be given more credit than vague and lengthy ones. Illegible answers will not be given marks.

You must score at least 40% on this exam to pass the course, otherwise you final grade will be set to no higher than 47%.

Everything is figureoutable.

Q1:	_____ /	30
Q2:	_____ /	8
Q3:	_____ /	8
Q4:	_____ /	6
Q5:	_____ /	8
Q6:	_____ /	12
Q7:	_____ /	10
Q8:	_____ /	8
Q9:	_____ /	10
TOTAL:	_____ /	100

Question 1: Short Answers [15x2=30 marks] [20 minutes]
For multiple choices questions, marks will be deducted for both missing choices and wrong choices.

1. What atoms need to be added to silicon to make p-type semiconductors?

2. Suppose an architecture uses 11-bit addresses for byte-addressable memory. What is the maximum size of memory that can be supported? Circle the correct answer.

a. 1KB b. 2KB c. 4KB d. 8KB e. None of above

3. What is the result of right shift 011011 arithmetically by 2 bits?

4. True or False. In a one-hot decoder, only one bit of all input bits can be 1.

a. True b. False

5. What is the third possible output of a tri-state buffer, other than 0 and 1?

6. Which of the following hex values are possible PC values in MIPS architecture? Circle all that apply.

a. ABCD1234 b. DCBA4321 c. A3B2C1D0 d. 98765432

7. Fill in below the correct values of CE' and OE' when reading data from SRAM.

CE': _____ OE': _____

8. Which of the following instruction allow jumping to ***any*** 32-bit address? Circle all that apply.

a. `j` b. `jr` c. `jal` d. `jalr`

9. Which of the following signals should be set to high when performing PC = PC + 4?

a. PCWrite b. MEMRead c. MEMWrite d. IRWrite

10. Suppose we have an architecture where registers store 128-bit values, then **how many bits** do we need to use for **shift amount** in the instruction?

11. What is the logic expression of maxterm **M8** in terms of the 4 inputs A, B, C and D?

12. Below is the K-map of a device that we learned in this course, which device is it?

Answer: _____

	Y'S'	Y'S	YS	YS'
X'	0	0	1	0
X	1	0	1	1

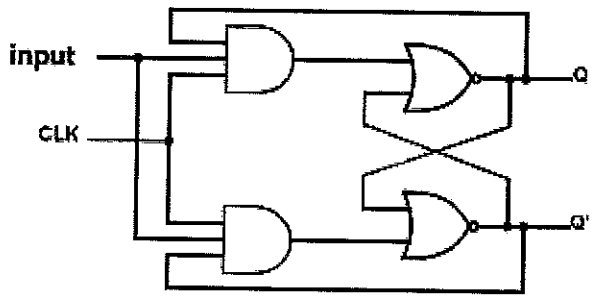
13. Given two 6-bit signed numbers A = -25, B = 18, what is the decimal value of the result of A - B?

14. True or False. A finite state machine with 31 states must have 5 flip-flops.

a. True b. False

15. Which kind of flip-flop is implemented in the following diagram?

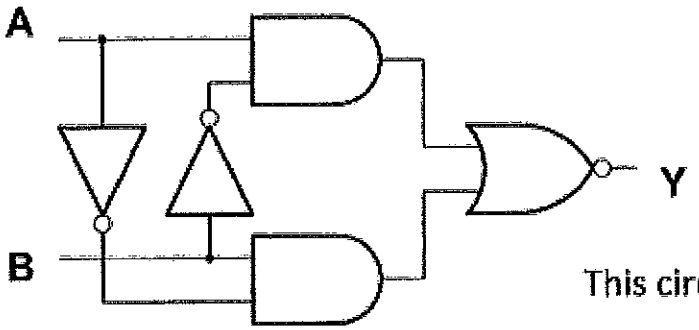
Answer: _____



Question 2: Combinational Circuit Analysis [8 marks] [10 minutes]

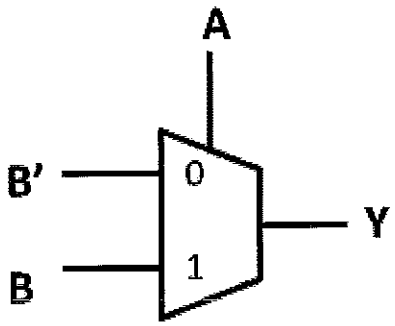
Each of the following circuits implements a functionality that can be described in just a few words. Complete the description for each circuit.

(a) [3 marks]



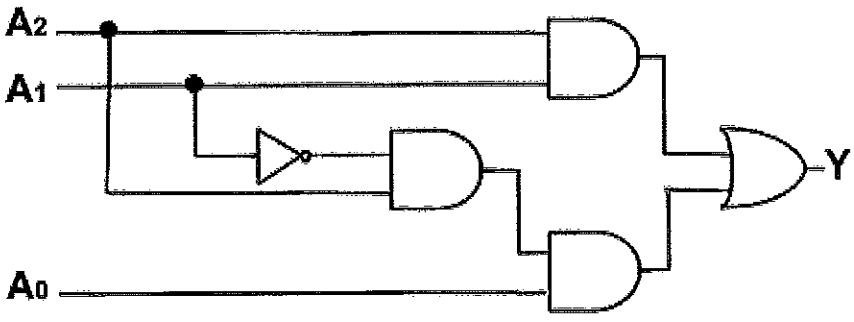
This circuit checks if _____

(b) [3 marks]



This circuit checks if _____

(c) [2 marks]



This circuit checks if the 3-bit number $A_2A_1A_0$ is _____

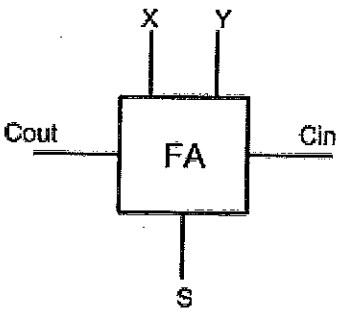
Question 3: Adder and Subtractor [8 marks] [10 minutes]

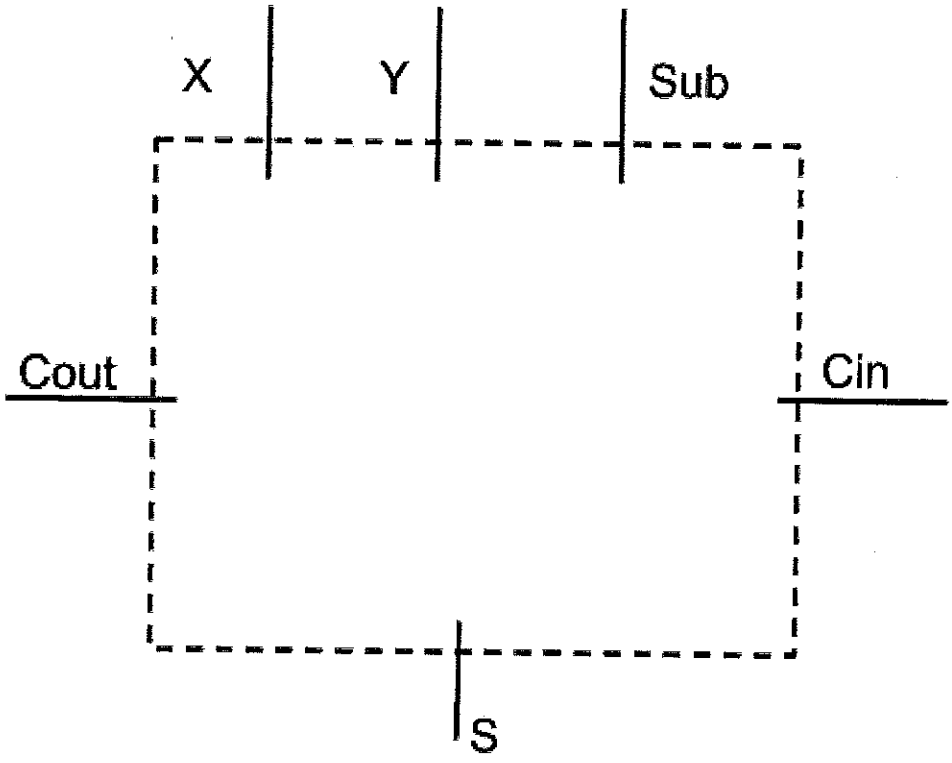
(a) Consider a **one-bit full adder**, which performs one-bit addition with X, Y and carry-in Cin and outputs sum bit S and carry-out bit Cout. Write below the logic expressions of the outputs **S** and **Cout**, in terms of inputs **X, Y** and **Cin**. [4 marks]

S = _____

Cout = _____

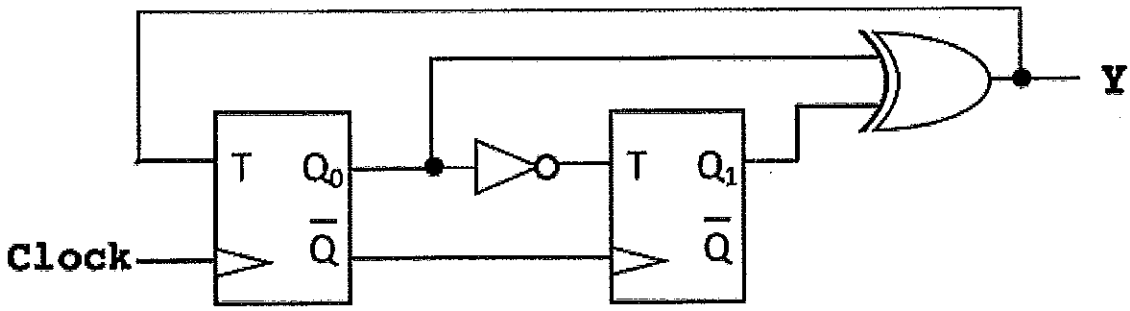
(b) Using the one-bit full adder on the right as a black box, complete the design of a **one-bit adder/subtractor** in the following diagram, i.e., when Sub = 0, the circuit outputs X + Y; when sub = 1, the circuit outputs X - Y. You are allowed to use **one** additional logic gate in your design. [4 marks]

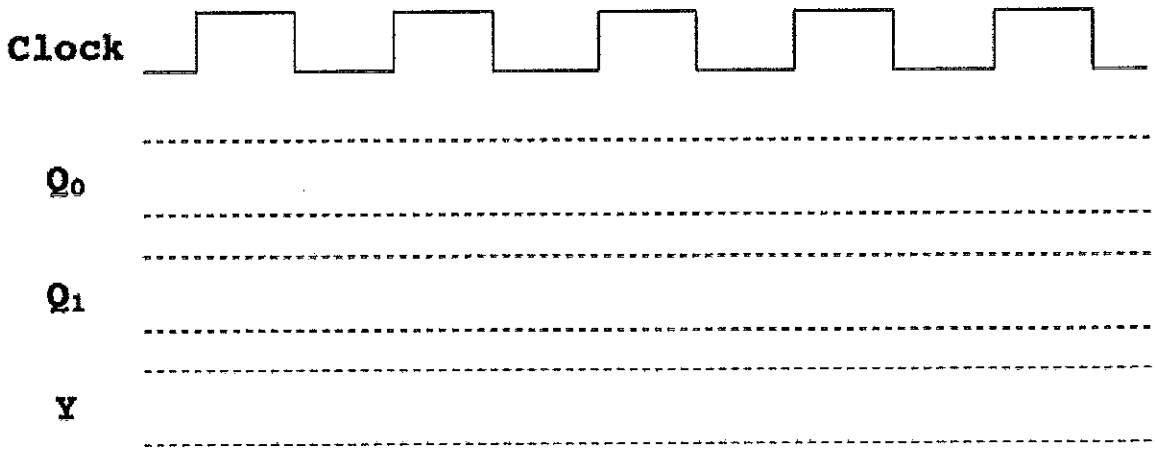




Question 4: Sequential Circuit Analysis [6 marks] [8 minutes]

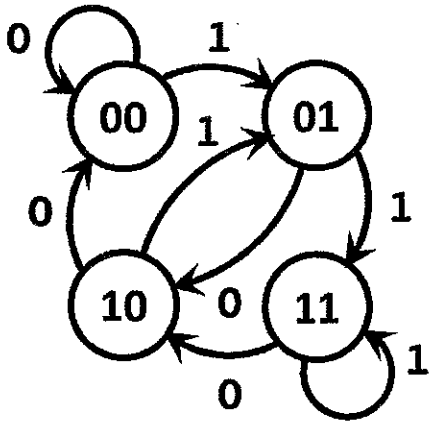
Given the following circuit diagram, and the waveform of the clock signal, complete the waveforms for Q_0, Q_1 and Y. Assume that Q_0 and Q_1 start with initial values of zero.





Question 5: FSM Design [8 marks] [10 minutes]

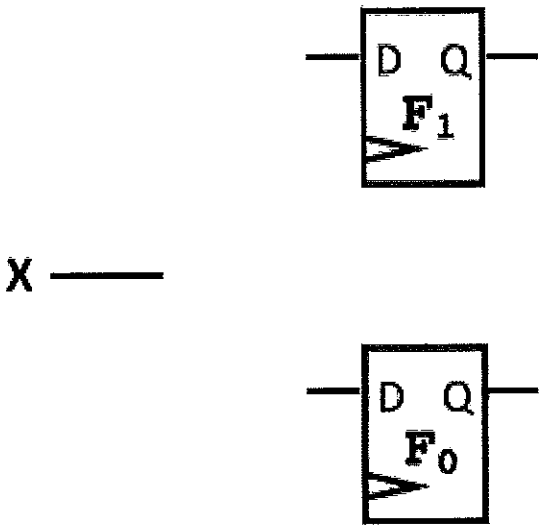
Consider the finite state diagram on the right, which has two flip-flops and a single input called X. In the space below, first fill in the state table of the FSM; and then complete the circuit diagram for this finite state machine. Assume that the two digits in each state (from left to right) are the values for flip-flops **F1 and F0**, respectively.



State table:

F1	F0	X	F1	F0
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Circuit diagram (You do NOT need to draw the clock signal)



Question 6: Assembly One Liners [2x6=12 marks] [20 minutes]

For each of the following task, write an assembly instruction that accomplishes the task. Your answer must be in **one line.**

(a) Increase the value stored in register $t1 by 3.

(b) Multiply the value in $t1 by 8 and store the result in $t2.

(c) Set the value in $t1 to 0 if the value in $t2 has lowest bit 0, otherwise set the value in $t1 to 1.

(d) Invert all bits of the value stored in $t1.

(e) Invert only the lowest 16 bits of the value stored in $t1.

(f) Make PC jump to a function labelled by FUNC and store the return address in $ra.

Question 7: Assembly: Control Flow [10 marks] [10 minutes]

Below is a piece of C code and the partially completed assembly translation of it. Complete the assembly code by filling in the blanks.

```
int i = 0, x = 10, y =20, n = 100;
for (i = 0; i < n; i++) {
    if (i != x) {
        x = x + 2;
    }
    else if (i <= y) {
        y = y - 2;
    }
    else {
        y = y + x;
    }
}
x = y;
```

```
main:
        li $t1, 0       # $t1 stores i
        li $t2, 10      # $t2 stores x
        li $t3, 20      # $t3 stores y
        li $t4, 100     # $t4 stores n
START:
        bge $t1, $t4, _________          # (a) [2 marks]
        beq $t1, $t2, ELIF
IF:
        addi $t2, $t2, 2
        ____________________________     # (b) [2 marks]
ELIF:
        _________ $t1, $t3, ELSE         # (c) [2 marks]
        addi $t2, $t2, -2
        ____________________________     # (d) [2 marks]
ELSE:
        add $t3, $t3, $t2
UPDATE:
        ____________________________     # (e) [2 marks]
        j START
END:
        move $t2, $t3
```

Question 8: Assembly: Arrays [8 marks] [15 minutes]

Below are two pieces of assembly code that manipulate the arrays declared in the data section. For each piece of code, write the final content of the array after the code is executed.

(a) [4 marks]

```
.data
len:        .word       4
array:      .word       -1, 0, 1, 2

.text
main:       la $s0, array
            la $s1, len
            lw $t1, 0($s1)
            lw $t0, 0($s0)
            add $t0, $t0, $t0
            sw $t0, 4($s0)
            addi $s0, $s0, 4
            lw $t0, 4($s0)
            sw $t0, 8($s0)
END:
```

Final content of array:

_____, _____, _____, _____

(b) [4 marks]

```
.data
len:        .word       4
array:      .word       -1, 0, 1, 2

.text
main:       la $s0, array
            la $s1, len
            lw $t1, 0($s1)
START:      lw $t0, 0($s0)
            add $t0, $t0, $t1
            sw $t0, 0($s0)
            addi $t1, $t1, -1
            addi $s0, $s0, 4
            bne $t1, $zero, START
END:
```

Final content of array:

_____, _____, _____, _____

Question 9: Machine Code [2x5=10 marks] [20 minutes]

Finish the following machine code - assembly translation. For (a), (b) and (c) fill in the missing bits in the machine code according to the given assembly code; **fill in the space with "X" if the value doesn't matter.** For (d) and (e) write the equivalent assembly code according to the given machine code.

(a) `and $t7, $t0, $t1`

`_ _ _ _ 0001 0000 1001 0111 _ _ _ _ _ _ _ _ 0100`

(b) `lw $t1, 49($s0)`

`_ _ _ _ _ _ _ _ 0000 1000 0000 0000 0011 _ _ _ _`

(c) `sll $s3, $t4, 4`

`0000 _ _ _ _ _ _ _ _ 1100 0011 _ _ _ _ 0000 0000`

(d) `0010 0110 0101 0100 0000 0000 0001 0001`

(e) `0000 0011 1110 0011 0011 1010 0000 1000`

MIPS/SPIM Reference Card

CORE INSTRUCTION SET (INCLUDING PSEUDO INSTRUCTIONS)

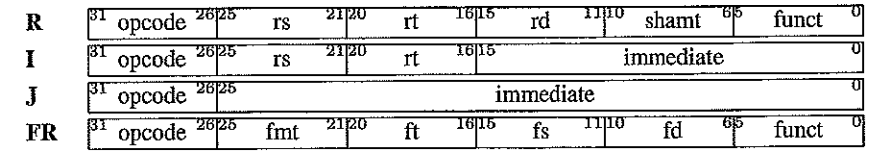
NAME	MNE-MON-IC	FOR-MAT	OPERATION (in Verilog)		OPCODE/ FUNCT (Hex)
Add	add	R	R[rd]=R[rs]+R[rt]	(1)	0/20
Add Immediate	addi	I	R[rt]=R[rs]+SignExtImm	(1)(2)	8
Add Imm. Unsigned	addiu	I	R[rt]=R[rs]+SignExtImm	(2)	9
Add Unsigned	addu	R	R[rd]=R[rs]+R[rt]	(2)	0/21
Subtract	sub	R	R[rd]=R[rs]-R[rt]	(1)	0/22
Subtract Unsigned	subu	R	R[rd]=R[rs]-R[rt]		0/23
And	and	R	R[rd]=R[rs]&R[rt]		0/24
And Immediate	andi	I	R[rt]=R[rs]&ZeroExtImm	(3)	c
Nor	nor	R	R[rd]=~(R[rs]|R[rt])		0/27
Or	or	R	R[rd]=R[rs]|R[rt]		0/25
Or Immediate	ori	I	R[rt]=R[rs]|ZeroExtImm	(3)	d
Xor	xor	R	R[rd]=R[rs]^R[rt]		0/26
Xor Immediate	xori	I	R[rt]=R[rs]^ZeroExtImm		e
Shift Left Logical	sll	R	R[rd]=R[rs]<<shamt		0/00
Shift Right Logical	srl	R	R[rd]=R[rs]>>shamt		0/02
Shift Right Arithmetic	sra	R	R[rd]=R[rs]>>>shamt		0/03
Shift Left Logical Var.	sllv	R	R[rd]=R[rs]<<R[rt]		0/04
Shift Right Logical Var.	srlv	R	R[rd]=R[rs]>>R[rt]		0/06
Shift Right Arithmetic Var.	srav	R	R[rd]=R[rs]>>>R[rt]		0/07
Set Less Than	slt	R	R[rd]=(R[rs]<R[rt])?1:0		0/2a
Set Less Than Imm.	slti	I	R[rt]=(R[rs]<SignExtImm)?1:0	(2)	a
Set Less Than Imm. Unsign.	sltiu	I	R[rt]=(R[rs]<SignExtImm)?1:0	(2)(6)	b
Set Less Than Unsigned	sltu	R	R[rd]=(R[rs]<R[rt])?1:0	(6)	0/2b
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5
Branch Less Than	blt	P	if(R[rs]<R[rt]) PC=PC+4+BranchAddr		
Branch Greater Than	bgt	P	if(R[rs]>R[rt]) PC=PC+4+BranchAddr		
Branch Less Than Or Equal	ble	P	if(R[rs]<=R[rt]) PC=PC+4+BranchAddr		
Branch Greater Than Or Equal	bge	P	if(R[rs]>=R[rt]) PC=PC+4+BranchAddr		
Jump	j	J	PC=JumpAddr	(5)	2
Jump And Link	jal	J	R[31]=PC+4; PC=JumpAddr	(5)	2
Jump Register	jr	R	PC=R[rs]		0/08
Jump And Link Register	jalr	R	R[31]=PC+4; PC=R[rs]		0/09
Move	move	P	R[rd]=R[rs]		
Load Byte	lb	I	R[rt]={24'b0, M[R[rs]+ZeroExtImm](7:0)}	(3)	20
Load Byte Unsigned	lbu	I	R[rt]={24'b0, M[R[rs]+SignExtImm](7:0)}	(2)	24
Load Halfword	lh	I	R[rt]={16'b0, M[R[rs]+ZeroExtImm](15:0)}	(3)	25
Load Halfword Unsigned	lhu	I	R[rt]={16'b0, M[R[rs]+SignExtImm](15:0)}	(2)	25
Load Upper Imm.	lui	I	R[rt]={imm,16'b0}		f
Load Word	lw	I	R[rt]=M[R[rs]+SignExtImm]	(2)	23
Load Immediate	li	P	R[rd]=immediate		
Load Address	la	P	R[rd]=immediate		
Store Byte	sb	I	M[R[rs]+SignExtImm] (7:0)=R[rt](7:0)	(2)	28
Store Halfword	sh	I	M[R[rs]+SignExtImm] (15:0)=R[rt](15:0)	(2)	29
Store Word	sw	I	M[R[rs]+SignExtImm]=R[rt]	(2)	2b

REGISTERS

NAME	NMBR	USE	STORE?
$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

(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 }
(4) JumpAddr = {PC[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2 s comp.)

BASIC INSTRUCTION FORMATS, FLOATING POINT INSTRUCTION FORMATS



ARITHMETIC CORE INSTRUCTION SET

NAME	MNE-MON-IC	FOR-MAT	OPERATION (in Verilog)		OPCODE/ FMT/FT/ FUNCT
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
Multiply	`mult`	R	{Hi,Lo}=R[rs]*R[rt]		0/–/–/18
Multiply Unsigned	`multu`	R	{Hi,Lo}=R[rs]*R[rt]	(6)	0/–/–/19
Branch On FP True	`bc1t`	FI	if(FPCond) PC=PC+4+BranchAddr	(4)	11/8/1/–
Branch On FP False	`bc1f`	FR	if(!FPCond) PC=PC+4+BranchAddr	(4)	11/8/0/–
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 Add Single	`add.s`	FR	F[fd]=F[fs]+F[ft]		11/10/–/0
FP Divide Single	`div.s`	FR	F[fd]=F[fs]/F[ft]		11/10/–/3
FP Multiply Single	`mul.s`	FR	F[fd]=F[fs]*F[ft]		11/10/–/2
FP Subtract Single	`sub.s`	FR	F[fd]=F[fs]-F[ft]		11/10/–/1
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 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 Double	`mul.d`	FR	{F[fd],F[fd+1]}={F[fs],F[fs+1]}*{F[ft],F[ft+1]}		11/11/–/2
FP Subtract Double	`sub.d`	FR	{F[fd],F[fd+1]}={F[fs],F[fs+1]}-{F[ft],F[ft+1]}		11/11/–/1
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]		16/0/–/0
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/–/–/–
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/–/–/–

ASSEMBLER DIRECTIVES

`.data` [*addr*]*	Subsequent items are stored in the data segment
`.kdata` [*addr*]*	Subsequent items are stored in the kernel data segment
`.ktext` [*addr*]*	Subsequent items are stored in the kernel text segment
`.text` [*addr*]*	Subsequent items are stored in the text * starting at [*addr*] if specified
`.ascii` *str*	Store string *str* in memory, but do not null-terminate it
`.asciiz` *str*	Store string *str* in memory and null-terminate it
`.byte` $b_1, \ldots, b_n$	Store the n values in successive bytes of memory
`.double` $d_1, \ldots, d_n$	Store the n floating-point double precision numbers in successive memory locations
`.float` $f_1, \ldots, f_1$	Store the n floating-point single precision numbers in successive memory locations
`.half` $h_1, \ldots, h_n$	Store the n 16-bit quantities in successive memory halfwords
`.word` $w_1, \ldots, w_n$	Store the n 32-bit quantities in successive memory words
`.space` n	Allocate n bytes of space in the current segment
`.extern` *symsize*	Declare that the datum stored at *sym* is *size* bytes large and is a global label
`.globl` *sym*	Declare that label *sym* is global and can be referenced from other files
`.align` n	Align the next datum on a 2^n byte boundary, until the next `.data` or `.kdata` directive
`.set at`	Tells SPIM to complain if subsequent instructions use `$at`
`.set noat`	prevents SPIM from complaining if subsequent instructions use `$at`

SYSCALLS

SERVICE	$v0	ARGS	RESULT
`print_int`	1	integer `$a0`	
`print_float`	2	float `$f12`	
`print_double`	3	double `$f12/$f13`	
`print_string`	4	string `$a0`	
`read_int`	5		integer (in `$v0`)
`read_float`	6		float (in `$f0`)
`read_double`	7		double (in `$f0`)
`read_string`	8	buf `$a0`, buflen `$a1`	
`sbrk`	9	amount `$a`	address (in `$v0`)
`exit`	10		

EXCEPTION CODES

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

MIPS Reference

Machine Encoding Aids

Key

o/f	instruction/function opcodes
s/t/d	first/second/third register
a/i	shift amount/immediate

Instruction Encoding Formats

Register	000000ss ssstttttt dddddaaa aafffffff
Immediate	oooooooss ssstttttt iiiiiiii iiiiiiii
Jump	oooooooii iiiiiiii iiiiiiii iiiiiiii

Instruction Syntax

Encoding	Syntax	Template
Register	ArithLog	f $d, $s, $t
	DivMult	f $s, $t
	Shift	f $d, $t, a
	ShiftV	f $d, $t, $s
	JumpR	f $s
	MoveFrom	f $d
	MoveTo	f $s
Immediate	ArithLogI	o $t, $s, i
	LoadI	o $t, immed32
	Branch	o $s, $t, label
	BranchZ	o $s, label
	LoadStore	o $t, i($s)
Jump	Jump	o label
	Trap	o i

Instruction Reference

Arithmetic and Logical Instructions

Instruction	Operation	Opcode or Function	Syntax	Comments
add $d, $s, $t	$d = $s + $t	100000	ArithLog	
addu $d, $s, $t	$d = $s + $t	100001	ArithLog	
addi $t, $s, i	$t = $s + i	001000	ArithLogI	i is sign-extended
addiu $t, $s, i	$t = $s + i	001001	ArithLogI	i is sign-extended
and $d, $s, $t	$d = $s & $t	100100	ArithLog	
andi $t, $s, i	$t = $s & i	001100	ArithLogI	i is zero-extended
div $s, $t	lo = $s / $t; hi = $s % $t	011010	DivMult	
divu $s, $t	lo = $s / $t; hi = $s % $t	011011	DivMult	
mult $s, $t	hi:lo = $s * $t	011000	DivMult	
multu $s, $t	hi:lo = $s * $t	011001	DivMult	
nor $d, $s, $t	$d = ~($s | $t)	100111	ArithLog	
or $d, $s, $t	$d = $s | $t	100101	ArithLog	
ori $t, $s, i	$t = $s | i	001101	ArithLogI	i is zero-extended
sll $d, $t, a	$d = $t << a	000000	Shift	Zero is shifted in
sllv $d, $t, $s	$d = $t << $s	000100	ShiftV	Zero is shifted in
sra $d, $t, a	$d = $t >> a	000011	Shift	Sign bit is shifted in
srav $d, $t, $s	$d = $t >> $s	000111	ShiftV	Sign bit is shifted in
srl $d, $t, a	$d = $t >> a	000010	Shift	Zero is shifted in
srlv $d, $t, $s	$d = $t >> $s	000110	ShiftV	Zero is shifted in
sub $d, $s, $t	$d = $s - $t	100010	ArithLog	
subu $d, $s, $t	$d = $s - $t	100011	ArithLog	
xor $d, $s, $t	$d = $s ^ $t	100110	ArithLog	
xori $d, $s, i	$d = $s ^ i	001110	ArithLogI	i is zero-extended

Movement Instructions

Instruction	Operation	Opcode or Function	Syntax	Comments
lhi $t, i	$t = i << 16	011001	LoadI	i is zero-extended
llo $t, i	$t = i	011000	LoadI	i is zero-extended
mfhi $d	$d = hi	010000	MoveFrom	
mflo $d	$d = lo	010010	MoveFrom	
mthi $s	hi = $s	010001	MoveTo	
mtlo $s	lo = $s	010011	MoveTo	

Comparison Instructions

Instruction	Operation	Opcode or Function	Syntax	Comments
slt $d, $s, $t	$d = $s < $t	101010	ArithLog	
sltu $d, $s, $t	$d = $s < $t	101001	ArithLog	
slti $t, $s, i	$d = $s < i	001010	ArithLogI	i is sign-extended
sltiu $t, $s, i	$d = $s < i	001001	ArithLogI	i is sign-extended

Branch and Jump Instructions

Instruction	Operation	Opcode or Function	Syntax	Comments
beq $s, $t, label	if ($s == $t) pc += i << 2	000100	Branch	label is a line reference in the code
bgtz $s, label	if ($s > 0) pc += i << 2	000111	BranchZ	label is a line reference in the code
blez $s, label	if ($s <= 0) pc += i << 2	000110	BranchZ	label is a line reference in the code
bne $s, $t, label	if ($s != $t) pc += i << 2	000101	Branch	label is a line reference in the code
j label	pc += i << 2	000010	Jump	label is a line reference in the code
jal label	$ra = pc; pc += i << 2	000011	Jump	label is a line reference in the code
jalr $s	$ra = pc; pc = $s	001001	JumpR	
jr $s	pc = $s	001000	JumpR	

Memory Instructions

Instruction	Operation	Opcode or Function	Syntax	Comments
lb $t, i($s)	$t = MEM[$s + i]	100000	LoadStore	Sign-extends the loaded byte
lbu $t, i($s)	$t = MEM[$s + i]	100100	LoadStore	Zero-extends the loaded byte
lh $t, i($s)	$t = MEM[$s + i]	100001	LoadStore	Sign-extends the loaded bytes
lhu $t, i($s)	$t = MEM[$s + i]	100101	LoadStore	Zero-extends the loaded bytes
lw $t, i($s)	$t = MEM[$s + i]	100011	LoadStore	
sb $t, i($s)	MEM[$s + i] = $t	101000	LoadStore	Lowest order byte is stored
sh $t, i($s)	MEM[$s + i] = $t	101001	LoadStore	2 lowest order bytes are stored
sw $t, i($s)	MEM[$s + i] = $t	101011	LoadStore	

Exception and Interrupt Instructions

Instruction	Operation	Opcode or Function	Syntax	Comments
trap i	Exception	0011010	Trap	i is a trap code; implements syscall