Student ID: __

THE HONG KONG UNIVERSITY OF SCIENCE & TECHNOLOGY

COMP2611: Computer Organization

Spring Semester, 2013

Mid-term Examination 2

April 22, 2013

Name: _____ **Student ID:** _____

Email: _____ **Lab Section Number:** _____

Instructions:

1. **This examination paper consists of 15 pages in total, including 7 questions within 12 pages and 3 appendices.**
2. **Please write your name, student ID, email and lab section number on this page.**
3. **Please answer all the questions in the spaces provided on the examination paper.**
4. **Please read each question very carefully, answer clearly and to the point. Make sure that your answers are neatly written.**
5. **Keep all pages stapled together. You can tear off the appendices only.**
6. **Calculator and electronic devices are not allowed.**
7. **The examination period will last for 2 hours.**
8. **Stop writing immediately when the time is up.**

Question	Percentage %	Scores
1	10	
2	13	
3	20	
4	15	
5	15	
6	12	
7	15	
TOTAL	100	

Student ID: __

Question 1 Multiple Choice Questions (10 points)

Please contact ZHANG Shanfeng for grade appealing

Circle all correct answers and only the correct answers as each wrong answer reduces 1 from the question's score (until 0 is reached)

1. Which of the following is/are addressing mode(s) in MIPS:

   A. Immediate addressing
   B. Register addressing
   C. Word addressing
   D. Pseudo-random addressing
   E. None of above

2. If `array1`, an array of words, stores the numbers 0, 1, 2, 3, 4, 5 (in this order). What will be final values in `array1` after executing the following instructions?

```
la   $s0, array1
addi $t0, $s0, 16
lw   $t1, 0($t0)
sw   $t1, -4($t0)
```

   A. 0, 1, 3, 3, 4, 5
   B. 0, 1, 2, 3, 3, 5
   C. 0, 1, 2, 4, 4, 5
   D. 0, 1, 2, 3, 4, 4
   E. None of the above

3. If the first instruction of the following sequence (i.e., `addi $sp, $sp -4`) is stored at memory address 0101 0000 0000 0000 1010 1011 0000 0000. What would be the value in the immediate field of the "`j`" instruction in binary format?

```
      addi  $sp, $sp, -4
      sw    $s0, 0($sp)
loop:
      beq   $s0,$zero, Exit
      addi  $s0,$s0, -1
      j loop
```

   A. 0101 0000 0000 0000 1010 1011 0000 0000
   B. 0000 0000 0000 0010 1010 1100 0100
   C. 0000 0000 0010 1010 1100 0010 00
   D. 0000 0000 0000 1010 1011 0000 10
   E. None of the above

Student ID: __

4. When executing addi $t1, $t1, - 5, which of the following statements are correct?

   A. The immediate field is extended from 16 bit to 32 bit wit zeros
   B. The immediate field is extended from 16 bit to 32 bit wit ones
   C. The ALU doesn't cause an exception if an overflow occurs
   D. The ALU will cause an exception if an overflow occurs
   E. None of the above

5. Recall the refined version of the division algorithm. If we skip the final correction step which of the following statement(s) is/are correct?

   A. The value of Quotient is still correct
   B. The value of Quotient is doubled
   C. The value of Remainder is still correct
   D. The value of Remainder is doubled
   E. None of above

6. What steps are NOT parts of the execution of an instruction?

   A. Fetch the instruction
   B. Encode the instruction
   C. Reset the registers
   D. Perform an ALU operation
   E. None of above

Question 2 MIPS procedure call (13 points)

Please contact WANG Keyu for grade appealing

The following C function `accumulate()` accumulates the elements of integer array `A[]` that satisfy a given condition. `testfunc()` is used to test the validity of such condition, and returns 1 if the condition is fulfilled and 0 otherwise.

```
int accumulate(int A[ ],int n) # n is size of array
{
    int i;
    int sum=0;
    for(i=0; i<n; i++){
        if(testfunc(A[i]) == 1){
            sum += A[i];
        }
    }
    return sum;
}
```

Student ID: __

The MIPS skeleton of `accumulate()` is given hereafter. Registers `$a0` and `$a1` hold the start address of `A[]` and the value of `n` respectively. Register `$v0` holds the return value `sum`. Assume that the MIPS code for `testfunc()` is accessible via label `testfunc`, and that it is supplied with an integer argument in register `$a0` and returns `0` or `1` in register `$v0`.

Complete the following MIPS code for the function `accumulate()` by filling each empty line with one instruction exactly according to the comments provided. Assume values in preserved register (e.g. `$s0` to `$s3`) remain unchanged after calling `testfunc`.

```
accumulate:
      addi $sp, $sp, -4          # store return address
      sw $ra,0($sp)
      li $s0,0                   # $s0 holds the iteration round i
      li $s1,0                   # $s1 holds the intermediate sum
      add $s2, $a0, $zero        # $s2 holds A[] starting address
loop:
      slt $t1, $s0, $a1          # condition check for loop
      beq $t1, $zero, exit
      sll $t2,$s0,2              # load A[i] to $s3
      addu $t2,$s2,$t2
      lw $s3,0($t2)
      add $a0, $s3, $zero        # test A[i] against condition
      jal testfunc

      beq $v0,$zero,next         # add A[i] to sum if fulfilled
      add $s1,$s1,$s3

next:
      addi $s0,$s0,1             # prepare for next iteration
      j loop

exit:
      add $v0,$s1,$zero          # return sum and exit
      lw $ra,0($sp)
```

Student ID: __

```
addi $sp,$sp,4
jr $ra
```

Question 3 Understanding a MIPS Program (20 points)

Please contact SU ZhiYang for grade appealing

Read the following MIPS program and answer questions. Refer to the ASCII code table in the Appendix if necessary.

```
1)  .data
2)  str: .asciiz "Comp2611 mid-term2 is easy!"
3)  .text
4)  .globl main
5)
6)  main:
7)  addi    $sp, $sp, -8
8)  sw      $ra, 4($sp)
9)  sw      $s0, 0($sp)
10) la      $s0, str
11) add     $a0, $s0, $zero
12) jal     function
13) add     $a0, $v0, $zero
14) ori     $v0,$0,1     # syscall to output an integer
15) syscall
16) lw      $s0, 0($sp)
17) lw      $ra, 4($sp)
18) addi    $sp, $sp, 8
19) ori     $v0,$zero,10 # syscall to terminate the program
20) syscall
21)
22) function:
23) add     $v1, $a0, $zero
24) li      $v0, 0
25) li      $t1,0x20
26)
27) function_label1:
28) lb       $t0, 0($v1)
29) beq      $t0, $zero, function_label2
30) addi     $v1, $v1, 1
31) beq      $t0, $t1, function_label3
32)
33) j function_label1
34)
35) function_label2:
36) jr       $ra
37)
38) function_label3:
39) addi     $v0,$v0,1
40) j function_label1
```

Student ID: __

a) State the purpose of lines 7-9 and lines 16-18? (4 points)

Lines 7-9: update the stack pointer to allocate 2 words of space, push $ra, $s0
Lines 16-18, pop $ra and $s0, restore stack pointer
.

b) If we remove lines 7-9 and lines 16-18, what will happen? Explain briefly. (3 points)

Nothing will happen, this program will still run correctly, because there is no nested function calls using "jal", moreover the value stored in the $s0 register is not used after the instruction "jal function".

c) What is the value of `$v0` right after line 28 loaded the 10th Byte from the memory? (4 points)
For reference, `str: .asciiz "Comp2611 mid-term2 is easy!"`

This program is to count the number of spaces,
10th byte => the characters "Comp2611 m" have been loaded and checked, so $v0 will be 1, because one space has been encountered.

d) What is the expected output of this MIPS program? (4 points)

The total number of spaces in the null-terminated string "str".

e) Modify line 31 (only) so that this segment of MIPS code would display the total number of *ASCII characters* in the string "`str`". You can only use *pure* MIPS instructions. (5 points)

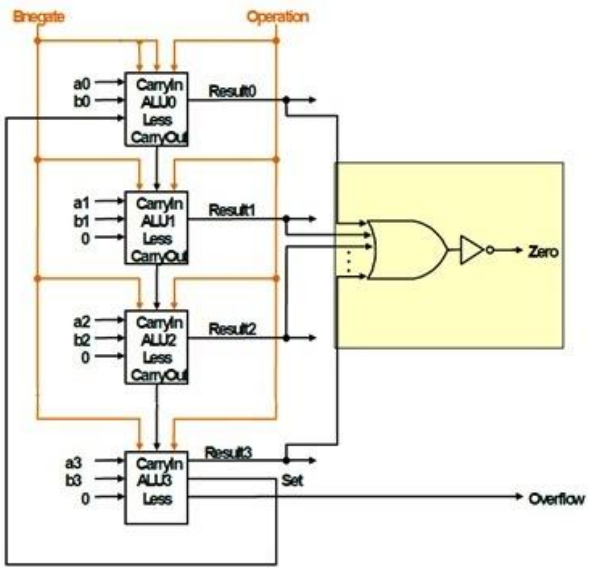
Replace "beq" by the unconditional jump "j" or replace the line with "addi $v0, $v0, 1", etc.

Question 4 ALU design (15 points)

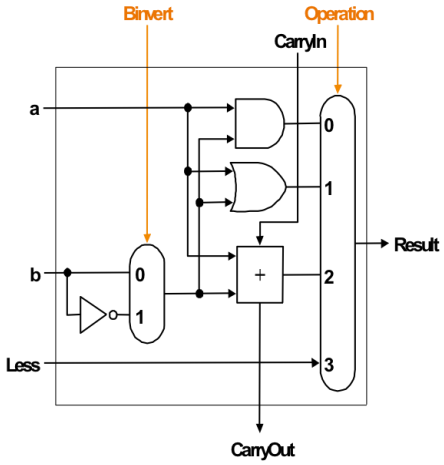
Please contact SU ZhiYang for grade appealing

Consider the 4-bit ALU shown below. Operations "00", "01", "10", and "11" correspond respectively to "AND", "OR", "Result of the adder", and "Less".

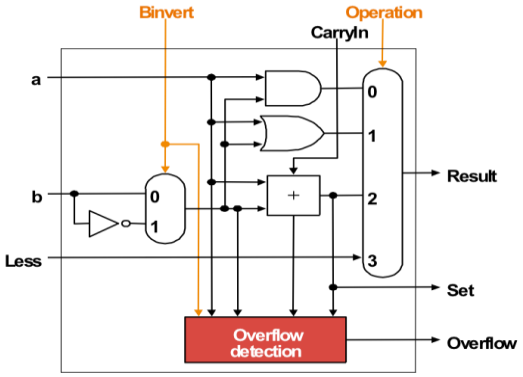
Student ID: _



a) Draw the diagram of the circuit for the 1-bit ALU of bit-0. For simplification, use a black box to represent the "adder". (5 points)



b) Draw the diagram of the circuit for the 1-bit ALU of bit-3. For simplification, use black boxes to represent the "adder" and the "overflow detection" unit. (5 points)

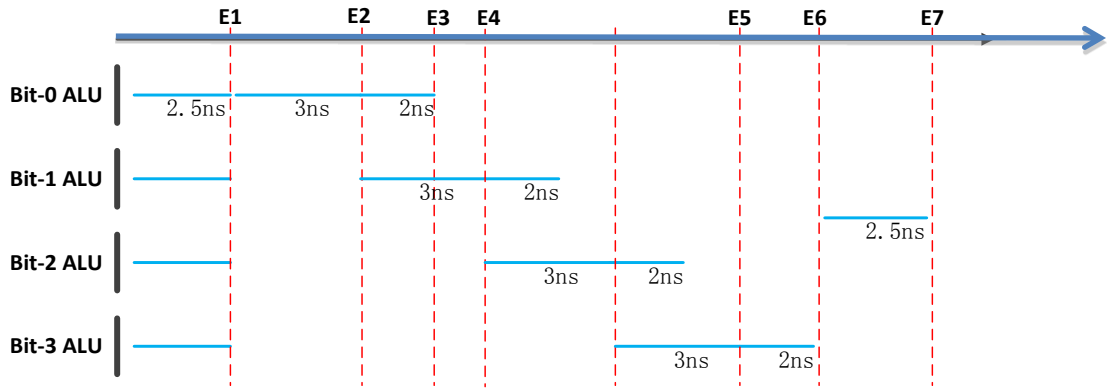


Student ID: __

c) Now assume that it takes certain time for the different logic components to generate the correct outputs based on the inputs as follows:

- 0.5 ns (nano second) for an inverter (NOT gate);
- 1 ns for a 2-input AND or OR gate;
- 2 ns for a 4-input AND or OR gate;
- 2 ns for any N-to-1 multiplexer;
- 3 ns for the adder to output both the result and the carryout;
- 5 ns for the overflow detection unit;
- Finally you may assume that the propagation of a signal on wires takes no time (i.e., 0ns).

If Bnegate = 1, Operation = 10, A = 0110, and B = 0101. How long would it take for the `zero` output to settle (show your steps by completing the time diagram below and adding the explanation as given below)? (5 points)



E1: a0, (-b0), and CarryIn0 are ready for the adder of Bit-0 ALU. a1, (-b1) are ready for the adder of bit-1, …
E2: SumOut0 and CarryOut0 are ready;

E1-E2: Adder0
E2-E3: Mux
E3-E4: Adder1
E4-E-missing1: Mux
E4-Elabelmissing: Adder2
Elabelmissing-Emissing2: Mux
ElabelMissing-E5: Adder3
E5-E6: Mux
E6-E7: 4-1 OR + Not

Student ID: __

Question 5 Multiplication (15 points)

Please contact `WANG Keyu for grade appealing`

a) Consider the refined (last) version of the multiplication hardware is used for a 6-bit multiplication. Given unsigned multiplicand 011110, and multiplier 010110, fill the blanks below to produce the multiplication result. (Write down the operations in the Remark column) (12 points)

Verify the solution from the blue text on

Iteration	Multiplicand (M)	Product (P)	Remark
0	011110	000000 010110	Initial State
1		000000 010110 000000 001011	No operation P = P >> 1
2		011110 001011 001111 000101	Left(P) = Left(P) + M P = P >> 1
3		**101101 000101** **010110 100010**	Left(P) = Left(P) + M P = P >> 1
4		010110 100010 001011 010001	No operation P = P >> 1
5		101001 010001 010100 101000	Left(P) = Left(P) + M P = P >> 1
6		010100 101000 001010 010100	No operation P = P >> 1

b) Suggest a simple approach to multiply signed numbers with the same hardware. (3 points)

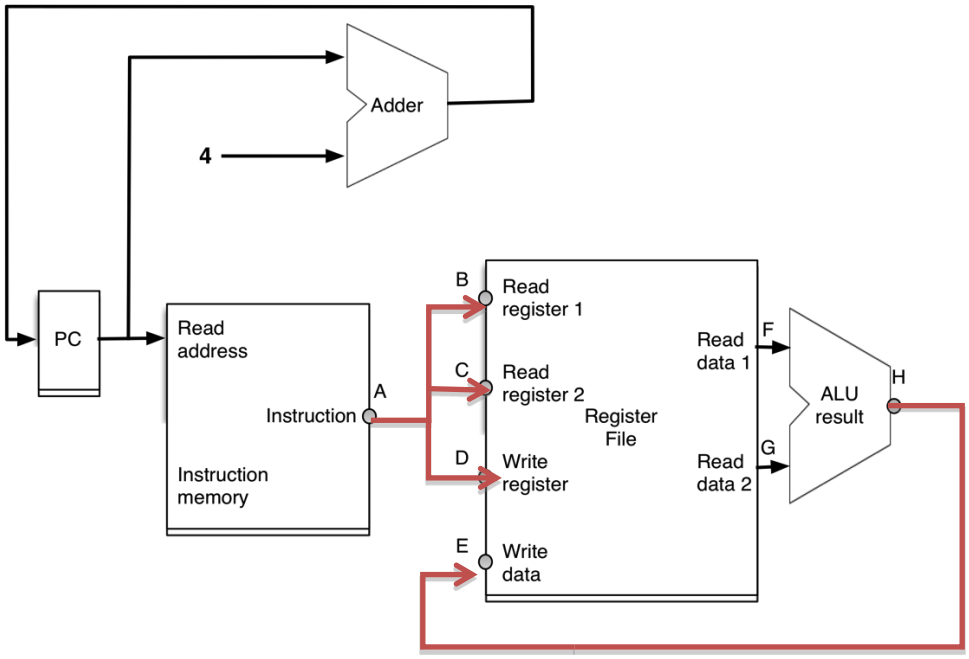
We can add simple hardware to negate the multiplicand and the multiplier if they are negative then negate the result if the signs disagree.

Student ID: __

Question 6 Single Cycle Datapath (12 points)

Please contact ZHANG Shanfeng for grade appealing

a) The following datapath is used to implement the instruction '`or $t0, $s1, $s2`' in a single cycle. Connect the grey bullets A, B, C, D, E, and H appropriately. (2 points)



b) Using the reference card provided in appendices 1 and 2, write the binary format of the instruction `or $t0, $s1, $s2` (2 points)

opcode	$s1	$s2	$t0	shamt	funct
000000	10001	10010	01000	00000	100101

c) Assume the values stored in registers $s1 and $s2 are 4 and 12 respectively. Fill the table below with binary values that correspond to the values available at points A, B, C, D, E, F, G and H when the Datapath above is used to execute the instruction. (8 points)

A	**0000 0010 0011 0010 0100 0000 0010 0101**
B	**10001**
C	**10010**

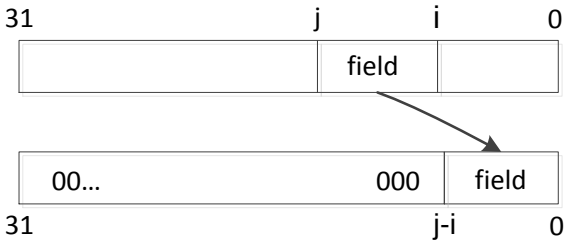
Student ID: __

D	**01000**
E	**0000 0000 0000 0000 0000 0000 0000 1100**
F	**0000 0000 0000 0000 0000 0000 0000 0100**
G	**0000 0000 0000 0000 0000 0000 0000 1100**
H	**0000 0000 0000 0000 0000 0000 0000 1100**

Question 7: General Questions (15 Points)

Please contact ZHANG Shanfeng for grade appealing

Assume we want to design a pseudo-instruction "`extc $s0, $s1, 5, 22`" that takes two registers (e.g., $s0 and $s1) and two scalars (e.g., 5, and 22) and extracts the bits from 5 to 22 (inclusive) from $s1 and stores them in the least significant bits of $s0, filling the remaining bits with 0, as illustrated below.



a) Give the **shortest** sequences of MIPS instructions (two) that replaces `extc $s0, $s1, 5, 22`. (5 points for 2 instructions, 3 for 3 instructions, 0 for more)

```
sll  $s0,  $s1, 9
srl  $s0,  $s0, 14
```

b) Given your understanding of PC-relative addressing, explain why the assembler might have problems directly implementing the branch instruction in the following code sequence (5 points)

```
Here:   beq  $s0,  $s2, There
...
There:  add  $s0,  $s0,   $s0
```

Student ID: __

Since label There fits on 16 bits only and is an offset with respect to 'Here', if the resulting address is too far away (16-bit is not able to describe it), the assembler might have problems replacing label There with an appropriate scalar when translating the program into binary. In other words, if there is more than 32KWords between Here+4 and There then the assembler will not be able to represent it

Show how the assembler might rewrite this code sequence to solve the problem and explain why. (5 points)

```
Here:   bne  $s0,  $s2, Next
        j There
Next: …

There:  add  $s0,  $s0,  $s0
```

This will work as long as Next → There) is less than 256MB or 64KWords. The loader will be in charge also of positioning the program in memory such that both Next and There have the same upper 4 bits.

Student ID: _

Appendix 1 : MIPS instructions 1

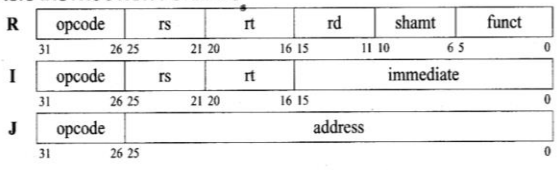
MIPS Reference Data ①

CORE INSTRUCTION SET

NAME	MNEMONIC	FORMAT	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+4;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)	$0 / 24_{hex}$
Load Halfword Unsigned	lhu	I	R[rt]={16'b0,M[R[rs]+SignExtImm](15:0)}	(2)	$0 / 25_{hex}$
Load Upper Imm.	lui	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]	(2)	$0 / 23_{hex}$
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 Unsigned	sltu	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6)	$0 / 2b_{hex}$
Shift Left Logical	sll	R	R[rd] = R[rs] << shamt		$0 / 00_{hex}$
Shift Right Logical	srl	R	R[rd] = R[rs] >> shamt		$0 / 02_{hex}$
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2)	28_{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[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2 s comp.)

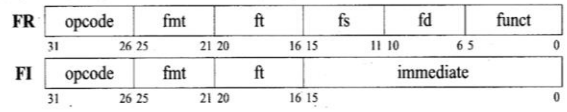
BASIC INSTRUCTION FORMATS



ARITHMETIC CORE INSTRUCTION SET ②

NAME	MNEMONIC	FORMAT	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]		16 /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
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



PSEUDO INSTRUCTION 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

Student ID: _

Appendix 2: MIPS instructions 2

OPCODES, BASE CONVERSION, ASCII SYMBOLS ③

MIPS opcode (31:26)	(1) MIPS funct (5:0)	(2) MIPS funct (5:0)	Binary	Deci-mal	Hexa-deci-mal	ASCII Char-acter	Deci-mal	Hexa-deci-mal	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

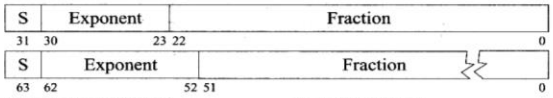
(2) opcode(31:26) == 17_{ten} (11_{hex}); if fmt(25:21)==16_{ten} (10_{hex}) *f* = s (single); if fmt(25:21)==17_{ten} (11_{hex}) *f* = d (double)

IEEE 754 FLOATING POINT STANDARD

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

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

IEEE Single Precision and Double Precision Formats:



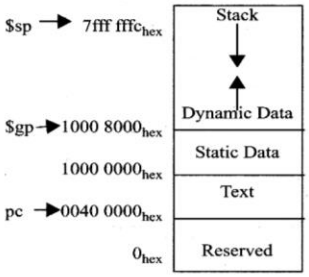
④

IEEE 754 Symbols

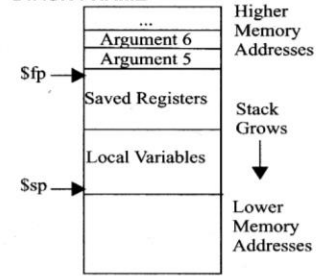
Exponent	Fraction	Object
0	0	± 0
0	≠0	± Denorm
1 to MAX - 1	anything	± Fl. Pt. Num.
MAX	0	±∞
MAX	≠0	NaN

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

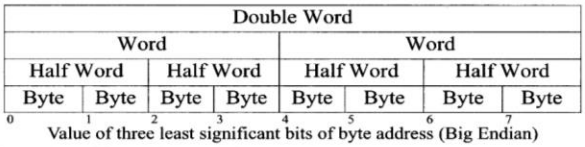
MEMORY ALLOCATION



STACK FRAME

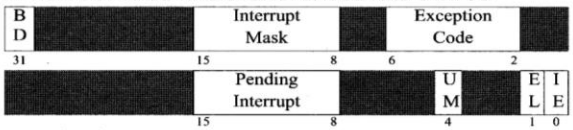


DATA ALIGNMENT



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 (10^x for Disk, Communication; 2^x for Memory)

SIZE	PRE-FIX	SIZE	PRE-FIX	SIZE	PRE-FIX	SIZE	PRE-FIX
10^3, 2^{10}	Kilo-	10^{15}, 2^{50}	Peta-	10^{-3}	milli-	10^{-15}	femto-
10^6, 2^{20}	Mega-	10^{18}, 2^{60}	Exa-	10^{-6}	micro-	10^{-18}	atto-
10^9, 2^{30}	Giga-	10^{21}, 2^{70}	Zetta-	10^{-9}	nano-	10^{-21}	zepto-
10^{12}, 2^{40}	Tera-	10^{24}, 2^{80}	Yotta-	10^{-12}	pico-	10^{-24}	yocto-

The symbol for each prefix is just its first letter, except μ is used for micro.

Student ID: _

Appendix 3: ASCII Code Table

Dec = Decimal; Hex = Hexadecimal; Char = Character

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□