Student ID: _____

THE HONG KONG UNIVERSITY OF SCIENCE & TECHNOLOGY

Computer Organization (COMP 2611)

Spring Semester, 2013

Final Examination

May 29, 2013

Name: _____ **Student ID:** _____

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

Instructions:

1. **This examination paper consists of 14 pages in total, including 8 questions within 10 pages, 2 appendices and 2 draft pages.**
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 appendix and draft page 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 %	Score
1- MCQ	10	
2- Cache Principles	15	
3- Cache Performance	10	
4- Single Cycle Datapath & Control	14	
5- Multi Cycle Datapath & Control	20	
6- Cache Architecture	10	
7- MIPS Recursion	13	
8- MIPS Coding	8	
TOTAL	100	

Student ID: _____

Question 1: Multiple-choice questions (10 points)

Circle all correct answers and only the correct answers (each incorrect answer reduces your score by one for the question)

A. The "Cache" is:
a) A cheap memory that can be plugged into the mother board to expand the main memory
b) A small, fast memory used by the processor to store copies of recently referred instructions or data from the disk to speed up the loading of the program
c) A reserved portion of the main memory used to save important data and instructions
d) A special area of memory on the chip that is used to save frequently used constants
e) None of the above

B. Desirable characteristic(s) of a memory system is (are) **(note, 1 point only)**
a) Speed and reliability
b) Low power consumption
c) Durability and compactness
d) All of the above
e) None of the above

C. The concept of pipelining is most effective in improving performance if the tasks being performed in different stages
a) Require different amounts of time
b) Require about the same amounts of time
c) All do the same operation
d) All use the same hardware
e) None of the above

D. Which of the following statements are correct?

a) Structural hazards happen when hardware cannot support the combination of instructions to execute in the same clock cycle in a pipeline
b) Control hazards happen when the unconditional jump instruction is executed
c) Forwarding fully solves the data hazard problem
d) We Can always resolve hazards by waiting long enough
e) None of the above

E. Which of the following statements are correct?

a) SRAM needs periodical refresh
b) DRAM needs periodical refresh
c) DRAM mostly used as caches inside the CPU
d) Memory hierarchy exploits the principle of locality
e) None of the above

Student ID: _____

F. Consider executing the following code in a pipelined processor:

```
add  $1,   $2,   $3
add  $4,   $5,   $6
add  $7,   $8,   $9
add  $10,  $11,  $1
add  $13,  $14,  $15
```

At the end of the fifth clock cycle of execution, some registers ($0-$31) have been read and some have been written. Circles the correct answers?

a) Read: $11, Written: $1
b) Read: $14 and $15, Written $1
c) Read: $1, Written: $1
d) Read: $11, Read $1
e) None of the above

Question 2 Cache principles (15 points)

Consider the following MIPS code sequence, and aswer the following questions

```
        li    $t0,  20
loop1:  li    $t1,  2
loop2:  addi  $t1,  $t1,  -1
        beq   $t1,  $zero,  loop2
        addi  $t0,  $t0,  -1
        beq   $t0,  $zero,  loop1
```

a) Does the sequence of code exhibit temporal locality? Explain your answer. (4 points)

No, because the program executes sequentially and the two branches are never taken
So the instructions are never accessed again over time.

b) Does the sequence of code exhibit spatial locality? Explain your answer. (4 points)

Yes, even though the program does not handle any data, the fact that instructions are executed in sequence implies that there is spatial locality

Student ID: _____

c) Consider a 4-way set associative cache with a 4-bit Tag field and the LRU block replacement policy. A sequence of memory references that map, all, to the same cache set is given in the following table. Complete the table to keep track of the current blocks in the set, their usage recency, and the cache hits or misses. (7 points)

	Tag field of the memory accesses generated by CPU (from left to right):								
	0101	0010	0100	0101	1111	1010	0010	0010	0101
Hit/Miss	Miss	Miss	Miss	Hit	Miss	Miss	Miss	Hit	Hit
	Cache Access Tracking:								
MRU	0101	0010	0100	0101	1111	1010	0010	0010	0101
		0101	0010	0100	0101	1111	1010	1010	0010
			0101	0010	0100	0101	1111	1111	1010
LRU					0010	0100	0101	0101	1111

Question 3 Cache Performance (10 points)

Consider a memory hierarchy with two levels of caches: the bus and cache lookup takes 0 cycles, a hit in level 1 takes 3 cycles, a hit in level 2 takes 10 cycles and a memory access takes 80 cycles. Of all data accesses, 80% of the times the data is in level 1. Of all data lookups in level 2, 50% of the data is found in level 2. Answer the following questions (briefly explain your answers)

a) Give the general equation for the average latency in such two cache-levels memory hierarchy? (4 points)

Average Latency = Hit_Time1 +
(1-Hit_Rate1) Hit_Time 2 +
(1-Hit_Rate1) (1-Hit_Rate2) miss penalty

b) What is the average memory access time without any caching? (2 points)

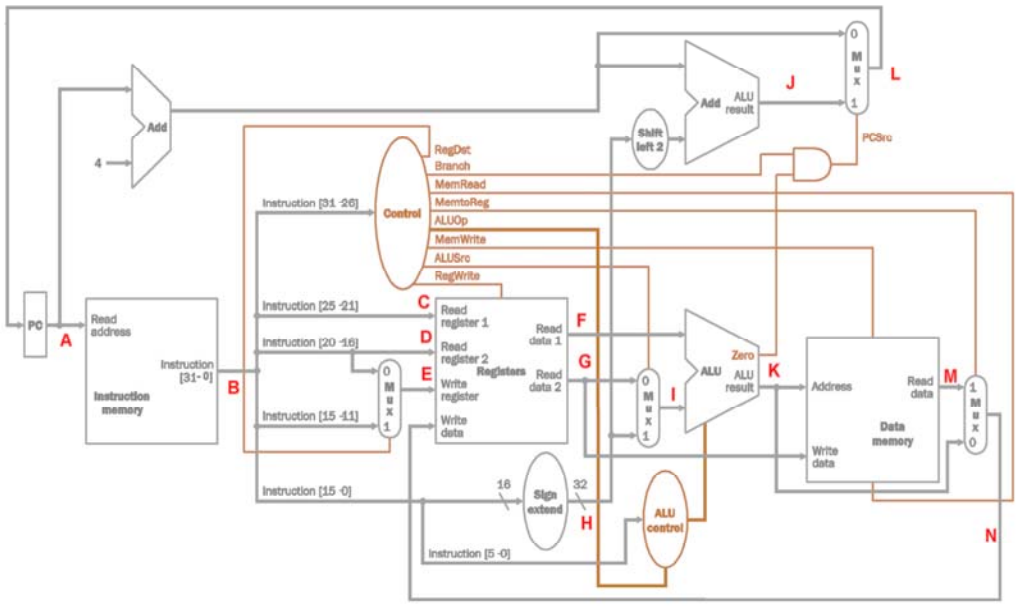
Ans: 80 cycles

c) What is the average memory access time with 2-levels of caching? (4 points)

Ans: 3 + 0.2*(10 + 0.5*80) = 13 cycles

Student ID: _____

Question 4: Single Cycle Datapath & Control (14 points)



a) We want to execute an instruction "SW $t0, 7($t1)" in the above single cycle datapath. Assume the values stored in registers $t0 and $t1 are 5 and 6 respectively. Fill in the table with proper binary numbers. (6 points)

A	Address of the instruction
B	101011 01001 01000 0000 0000 0000 0111
D	01000
E	01000
F	0000 0000 0000 0000 0000 0000 0000 0110
G	0000 0000 0000 0000 0000 0000 0000 0101
I	0000 0000 0000 0000 0000 0000 0000 0111

b) Complete the following table with control signal values to execute the instruction. Don't care signals are represented by 'x'. (8 points)

RegDst	Branch	MemRead	MemtoReg	ALUOp	MemWrite	ALUSrc	RegWrite
x	0	0	x	00	1	1	0

Student ID: _____

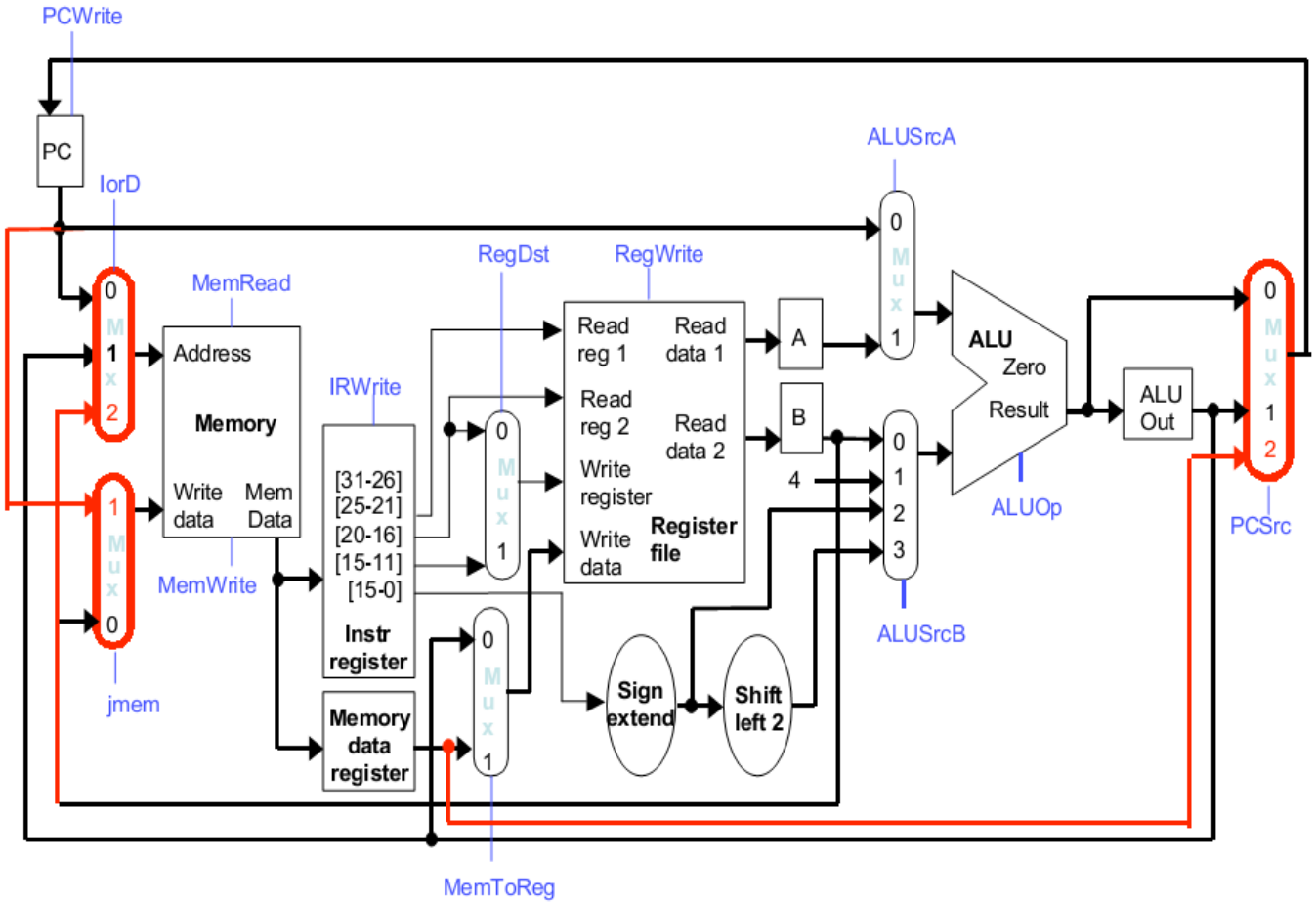
Question 5: Multi-Cycle Data Path (20 points)

Consider implementing an imaginary instruction in MIPS: *jump memory* (jmem), similar to the *jump-and-link* (jal) instruction, except that: i) the target address is loaded from memory and ii) the return address is saved to memory. *jmem* is an I-type instruction, as shown below.

Field	op	rs	rt	imm
Bits	31-26	25-21	20-16	15-0

And the syntax of *jmem* is explained below:
jmem rt, offset(rs) **# Mem[Reg[rs]+offset] = PC + 4; PC = Mem[Reg[rt]]**
Assume that Reg[rt] and (Reg[rs] + offset) are distinct (non-overlapping) addresses.

a) Show the changes that are needed to support *jmem*, without any modification to the main functional units. (Hint: You should only add wires, and add or expand multiplexers)
Explain your design by specifying the operations that need to be done in the table below. Note: *jmem* can be implemented in 4 cycles. (10 points)

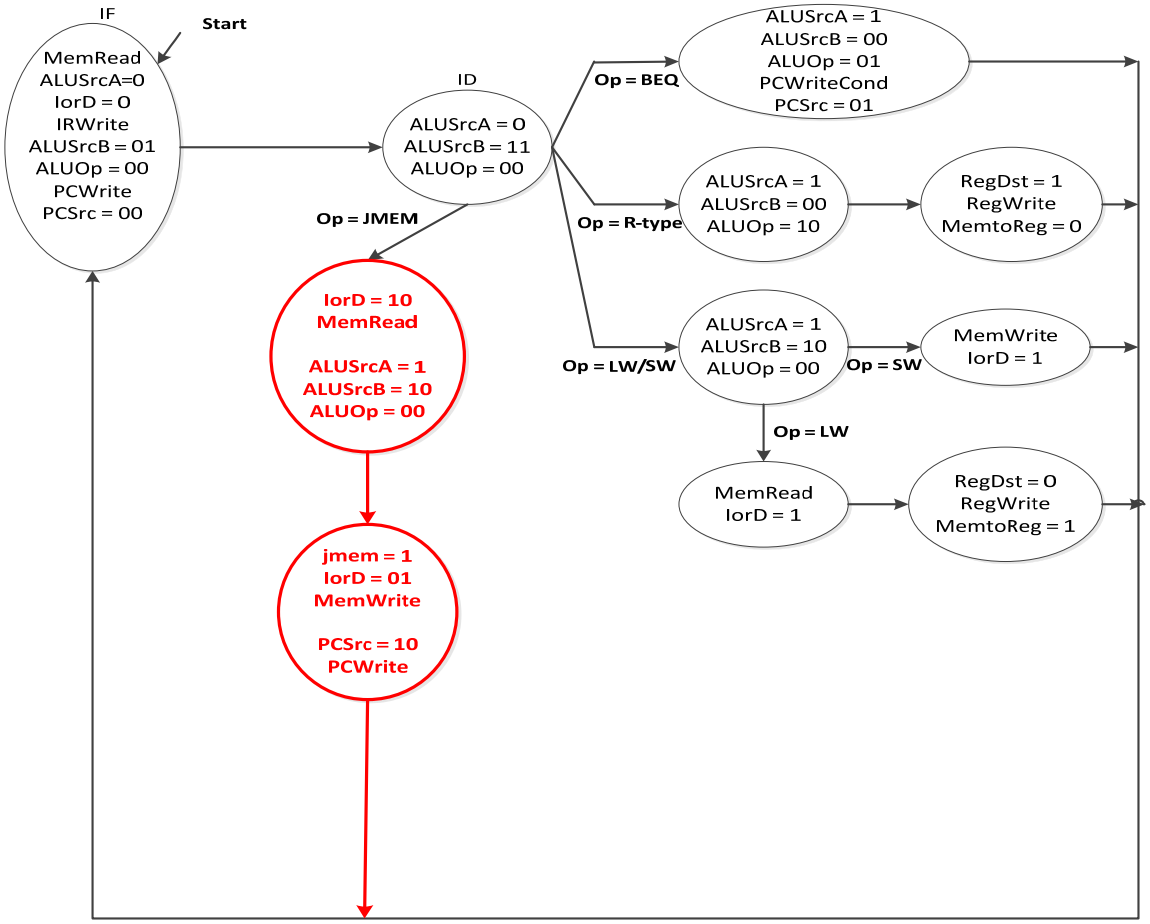


Cycle	Operations

Cycle 1	- **Fetch** - **PC = PC + 4**
Cycle 2	- **Decode** - **Read rs and rt** - **Calculate the branch target (for possible beq)**
Cycle 3	- **Calculate the store address Reg[rs] + SignExtend[imm]** - **Do a memory read: MDR = Mem[Reg[rt]]**
Cycle 4	- **Send AluOut to the Memory Address and PC to the data port: Mem[AluOut]=PC+4** - **Send MDR to the PC: PC = MDR**

b) Complete the finite state machine diagram to support the *jmem* instruction. Remember to account for any control signals that you added or modified in part (a) (10 points).

Student ID: _____


IF
Start
MemRead
ALUSrcA=0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSrc = 00
ID
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
Op = BEQ
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSrc = 01
Op = R-type
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10
RegDst = 1
RegWrite
MemtoReg = 0
Op = LW/SW
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
Op = SW
MemWrite
IorD = 1
Op = LW
MemRead
IorD = 1
RegDst = 0
RegWrite
MemtoReg = 1
Op = JMEM
IorD = 10
MemRead
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
jmem = 1
IorD = 01
MemWrite
PCSrc = 10
PCWrite

Student ID: _____

Question 6: Cache Architecture (10 points)

a) Consider an 8-way set associative cache with a cache size of 16Kbytes, 32 Bytes block size and 32 bit memory addresses. Answer the following questions (show briefly your steps). (4 points)

Number of blocks in the cache = 16K / 32 = 512 = 2^9

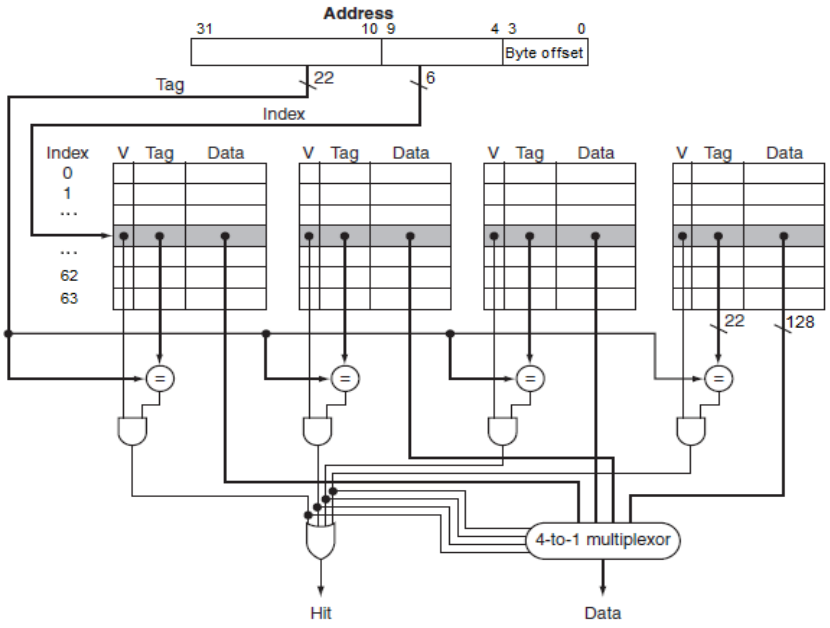
Number of sets in the cache = 512 / 8 = 64 = 2^6

Number of bits for the Byte offset = 5

Number of bits for the Index field = 6

Number of bits for the Tag field = 32 - 6 - 5 = 21

b) Consider the following 4-way set associative cache.



Complete the last two columns of the following table for the given sequence of memory accesses: (6 points)

Address of the memory access generated by CPU	Assigned cache set	Hit or miss
0000 1111 0101 0010 0011 0111 0100 0110	110100	Miss
0000 1111 0101 0010 0011 0111 0110 0110	110110	Miss
0000 1111 0101 0010 0011 0111 0100 1101	110100	Hit
0000 1111 0101 1010 0011 0111 0100 1101	110100	Miss
0000 1111 0101 1010 0011 0111 1111 0110	111111	Miss
0000 1111 0101 0010 0011 0111 0100 1101	110100	Hit

Student ID: _____

Question 7: MIPS Recursion (13 points)

The following C/C++ function `factorial()` computes the factorial value for an unsigned integer *n*.

```
unsigned int factorial(unsigned int n)/*n is a positive integer*/
{
     if (n<=1)  return 1;              /*Base case              *
     else return n*factorial(n-1);     /*Non-base case          */
}
```

Implement the function in pure MIPS instructions using *recursion*. Upon return, the function should have the integer value *n* stored in register `$v0,` and the *factorial(n)* value stored in `$a0`. Both *n* and the result of *factorial(n)* are **unsigned**. You do not need to check for arithmetic overflows in the function. You could refer to the appendix for the MIPS instruction set. The lines below are given just to keep your code neat they may be more than you need (we could write it in about 18 lines including all labels!)

```
factorial:


base_case:               # Base Case
sltiu $t0, $v0, 2         # n<2 (if true n<=1)
bne $t0,1,non_base_case  #
addi $a0, $zero, 1       #
jr $ra                   # return

non_base_case:           # Non Base Case
addi $sp, $sp, -8        # Preserve the register values using the stack
sw $v0, 4($sp)           #
sw $ra, 0($sp)           #

addiu $v0, $v0, -1        # reduce n before calling factorial
jal factorial            # recursively call factorial

lw $v0, 4($sp)           # retrieve the register values from the stack
lw $ra, 0($sp)           #
addi $sp, $sp, 8         #

multu $v0, $a0           # calculate n*f(n-1) and store in $a0
mflo $a0
jr $ra                   # return back to the caller
```

1 mark for base_case testing,
1 mark for base_case return,
2 marks for preserving the registers $ra and $v0 in non_base_case,
2 marks for retrieving the registers $ra and $v0 in non_base_case,
2 marks for making the correct recursive call,
2 marks for making the correct unsigned multu and retrieving the results to $a0
3 marks for the correct whole function

Minor mistakes (ie store values in the wrong registers) -1 to -3 marks
ZERO MARK for **non-recursive** function without the jal instruction.

Question 8: General MIPS Questions (8 points)

a) We want to be able to compare two 64 bit integers represented on two registers each. Write the shortest sequence of pure MIPS instructions to perform a 64-bit slt operation. The first operand is passed in registers $s0 and $s1 and the second operand is passed in registers $s2 and $s3. The most significant words of the operands are in $s1 and $s3 respectively. The comparison result is stored in register $s4. (Hint: It can be done in 3 instructions. If needed, you can use as many labels as you want) (4 points)

64_SLT:
_____slt $s4, $s0, $s2
_____beq $s1, $s3, Done
_____slt $s4, $s1, $s3
Done:

b) We want to be able to add two 64-bit integers represented on two registers each. Write the shortest sequence of pure MIPS instructions to perform a 64-bit ADD operation. The operands are passed in registers $s0 and $s1 for operand 1 and $s2 and $s3 for operand 2. The addition result is to be stored in $s4 and $s5. The most significant words of the two operands and result are in $s1, $s3, and $s5 respectively. (Hint: it can be done in four instructions. If needed, you can use as many labels as you want) (4 points)

64_ADD:
_____addu $s4, $s0, $s2

_____sltu $t0, $s4, $s0

_____add $s5, $s1, $s3

_____add $s5, $s5, $t0

Student ID: _____

Appendix 1 : MIPS instructions 1

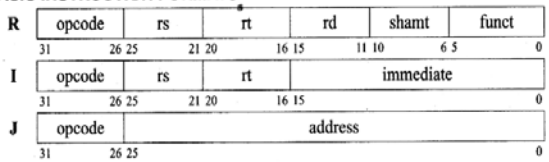
MIPS Reference Data ①

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.)

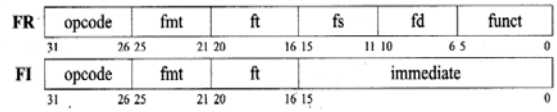
BASIC INSTRUCTION FORMATS



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*
* (*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]		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	Decimal	Hexadecimal	ASCII Character	Decimal	Hexadecimal	ASCII Character
(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

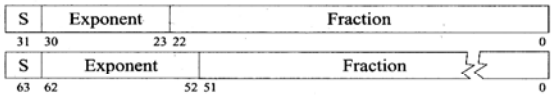
(2) opcode(31:26) == 17_{ten} (11_{hex}); if fmt(25:21)==16_{ten} (10_{hex}) *f* = s (single); if fmt(25:21)==17_{ten} (11_{hex}) *f* = d (double)

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:

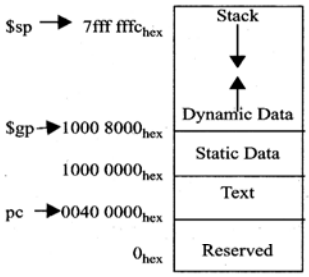


IEEE 754 Symbols ④

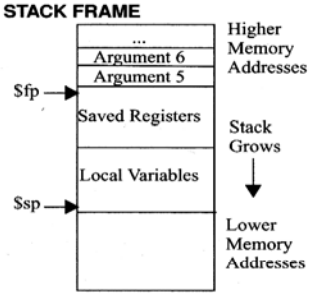
Exponent	Fraction	Object
0	0	± 0
0	≠0	± Denorm
1 to MAX - 1	anything	± Fl. Pt. Num.
MAX	0	±∞
MAX	≠0	NaN

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

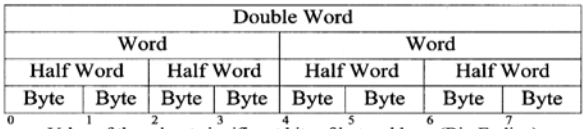
MEMORY ALLOCATION



STACK FRAME

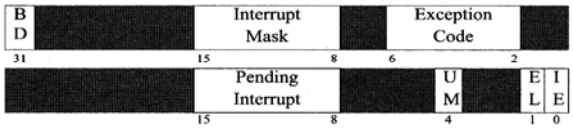


DATA ALIGNMENT



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	PREFIX	SIZE	PREFIX	SIZE	PREFIX	SIZE	PREFIX
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: _____

- Draft paper -

Student ID: _____

- Draft paper -