ICE 3003: Computer Architecture
Midterm Exam
April 21st, 2016
Professor Jae W. Lee
SOLUTIONS

Student ID #: ____________________

Name: ____________________

This is a closed book, closed notes exam.
120 Minutes
14 Pages
(+ 4 Appendix Pages)

Total Score: 200 points

Notes:

- Please turn off all of your electronic devices (phones, tablets, notebooks, netbooks, and so on). A clock is available on the lecture screen.
- Please stay in the classroom until the end of the examination.

- You must not discuss the exam's contents with other students during the exam.
- You must not use any notes on papers, electronic devices, desks, or part of your body.

(This page is intentionally left blank. Feel free to use as you like.)

Part A: Short Answers (16 points)

Question 1 (16 points)

Please indicate whether each of the following statements is true or false. You don't have to justify your answer—Just write down true or false.

(1) Typically, dynamically linked programs use less memory than statically linked programs.

TRUE

(2) Increasing clock rate improves *both* throughput and response time of instruction execution.

TRUE

(3) To compare two IEEE 754 floating-point numbers, you can simply interpret them as two sign-magnitude integers and perform an integer comparison to obtain the correct result.

TRUE

(4) When performing multiple integer additions, the order of additions does not affect the final result since addition is commutative.

TRUE

Part B: Iron Law of CPU Performance (24 points)

Question 2 (24 points)

You are the chief architect of *Andromeda*^{TM} *S8*, your company's next-generation smartphone. Based on the customers' feedback for *Andromeda*^{TM} *S7*, you make it the top priority to maximize battery life (i.e., energy efficiency), instead of performance. You are considering two processors: one from S-company (labeled *CPU*$_S$) and the other from Q-company (labeled *CPU*$_Q$). To evaluate energy efficiency of both chips, you use a popular social network app, titled *Instacram*^{TM}, with which you can show off what you ate for dinner. The two processors have the same ISAs but different CPIs and clock rates, and performance analysis results for the app are summarized below:

Instruction Type	**Instr. count (millions)**	**Cycles per Instr. (CPI)**	
		CPU$_S$	**CPU$_Q$**
Arithmetic & Logic	10	1	1
Load & Store	5	4	2
Branch	4	2	3
Miscellaneous (기타)	1	4	4

Clock rate (GHz)	
CPU$_S$	**CPU$_Q$**
2.0	1.5

(1) What are the average CPIs for this app on both processors?

CPU$_S$ = 2.1
CPU$_Q$ = 1.8

(2) What are the CPU times of this app on both processors?
(Note: Be sure to include time units.)

CPU$_S$ = 21 ms
CPU$_Q$ = 24 ms

(3) One way to compare energy efficiency of two processors is to compare their average energy-per-instruction (EPI). Assuming the capacitance (C) and operating voltage (V) are the same for both processors, which one is more energy efficient and by how much? To answer this, calculate the EPI ratio of the two processors.
(*Hint*: Energy [Joule] = Power [Watt] * Time [Sec])

EPI$_S$:EPI$_Q$ = (Freq$_S$ * Time$_S$) : (Freq$_Q$ * Time$_Q$) = 2.0*21 : 1.5*24 = 7 : 6
(i.e., CPU$_Q$ is more energy efficient.)

Question 3 (15 points)

CPU time is affected by three different factors: instruction count (IC), average cycles per instruction (CPI), and clock cycle time (CLK). For each of the following changes, identify **all** factors that are affected. If no factor is affected, just write down "nothing." It's okay to use acronyms (IC, CPI, CLK), and you don't have to justify your answer.

(1) Add a new memory-arithmetic instruction like `addmem`.

```
addmem $s0, 8($s1) # $s0<-$s0+Mem[$s1+8]
```

IC, CPI (+CLK depending on whether this instr increases the critical path)

(2) Switch endianness (e.g., from big endian to little endian).

Nothing

(3) Add more general-purpose registers.

IC, CPI (+CLK depending on whether reg read falls on the critical path)

(4) Increase the width of datapath from 32 bits to 64 bits.

IC, CPI (+CLK depending on whether it affects critical path)

(5) Migrate from 14nm fabrication technology to 10nm technology.

CLK

Part C: MIPS ISA (36 points)

Question 4 (20 points)

Address	Code
	START:
0x20160420	li $s1, 408
0x20160424	li $s2, 1 # TRUE
0x20160428	lw $s6, 20($sp) # p[]
	INIT:
0x2016042c	addi $s1, $s1, -4
0x20160430	add $s3, $s1, $s6 # p[k]
0x20160434	sw $s2, 0($s3) # p[k] = TRUE
0x20160438	**(1)bne $s1, $0, INIT**
0x2016043c	lw $s4, 12($sp) # load i
0x20160440	lw $s5, 16($sp) # load buffer[j]
0x20160444	slti $s7, $s4, 3 # i < TESTNUM
0x20160448	beq $s7, $0, END
0x2016044c	addi $s4, $s4, 1 # i++
0x20160450	li $t0, 32 # $t0 = ' ' (space)
0x20160454	li $t1, 10 # $t1 = '\n' (newline)
0x20160458	li $t2, 10 # $t2 = digit (10)
0x2016045c	move $s0, $0 # n1 = 0
	N1_1:
0x20160460	lb $t3, 0($s5) # buffer[j]
0x20160464	addi $s5, $s5, 1 # j++
0x20160468	**(2)beq $t3, $t0, N1_2** # buffer[j] == '\n'?
0x2016046c	mul $s0, $s0, $t2 # n1 *= 10
0x20160470	add $s0, $s0, $t3
0x20160474	addi $s0, $s0, -48 # n1 += buffer[j] - '0'
0x20160478	**(3)j N1_1**
	N1_2:
0x2016047c	sw $s0, 4($sp) # store n1
0x20160480	move $s1, $0 # n2 = 0

The code above is the part of main function of HW1-1. Encode the three instructions (1), (2), and (3) into 32-bit hexadecimal form.

(1) 0x1620FFFC
(2) 0x11680004
(3) 0x08058118

Question 5 (16 points)

Translate the following pseudo instructions into a *minimum* sequence of native instructions in MIPS ISA. Your pseudo instructions should not require any modifications to the rest of the program.

(1)
```
bgt  $t0,  $s0,  label

slt  $at,  $s0,   $t0
bne  $at,  $zero, label
```

(2)
```
ble  $t0,  $s0,  label

slt  $at,  $s0,   $t0
beq  $at,  $zero, label
```

(3) `li   $t0, 0x34AF14` (constant > 16 bit)

```
lui  $t0,  0x0034
ori  $t0,  $t0,   0xAF14
```

(4) `li   $t0, 0xF14` (constant <= 16 bit)

```
   ori  $t0,  $zero,  0xF14
or addi $t0,  $zero,  0xF14
```

Part D: Computer Arithmetic (39 points)

Question 6 (24 points)

Please answer the following questions.

(1) An IEEE 754 floating point number can be represented in the following scientific notation with base 2: $A = x_{(16)} \times 2^{y_{(10)}}$. To represent the *smallest positive number* in single precision, what should be the value of x and y?
($Note$: 300 (Decimal notation) = 3.0×10^2 (Scientific notation))

$\mathbf{x = 1.0_{(16)}}$
$\mathbf{y = -149_{(10)}}$

(2) How can you represent the *smallest single-precision positive number in the normalized form*? Using the format of A in (1), write the value of x and y.

$\mathbf{x = 1.0_{(16)}}$
$\mathbf{y = -126_{(10)}}$

(3) Fill the snapshot of the floating-point register file after executing following instruction.

```
add.d  $F0, $F2, $F4
```

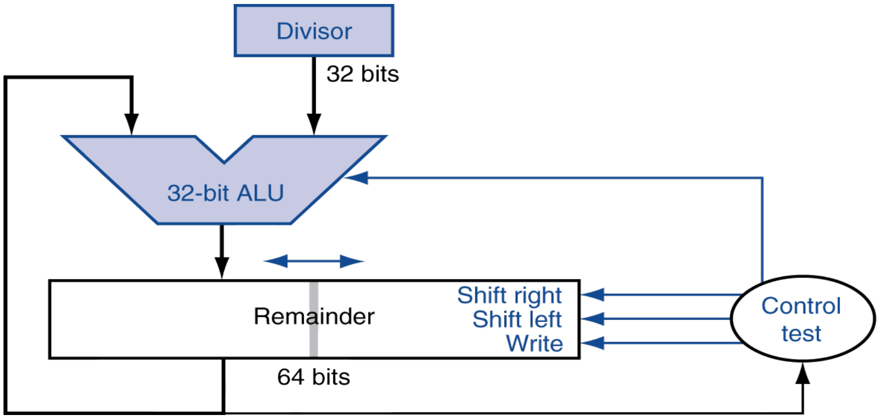
F0	0x3f800000
F1	0x40500000
F2	0x40500000
F3	0x00000000
F4	0x40080000
F5	0x00000000
…	…

⇒

F0	**0x4050C000**
F1	**0x00000000**
F2	**0x40500000**
F3	**0x00000000**
F4	**0x40080000**
F5	**0x00000000**
…	…

Question 7 (15 points)

Here is "an improved version of the division hardware" from the textbook. When the divisor is $0421_{(10)}$ and dividend is $2016_{(10)}$, what will be the value of Remainder register (64 bits) after 33 cycles? Write the answer in hexadecimal form.

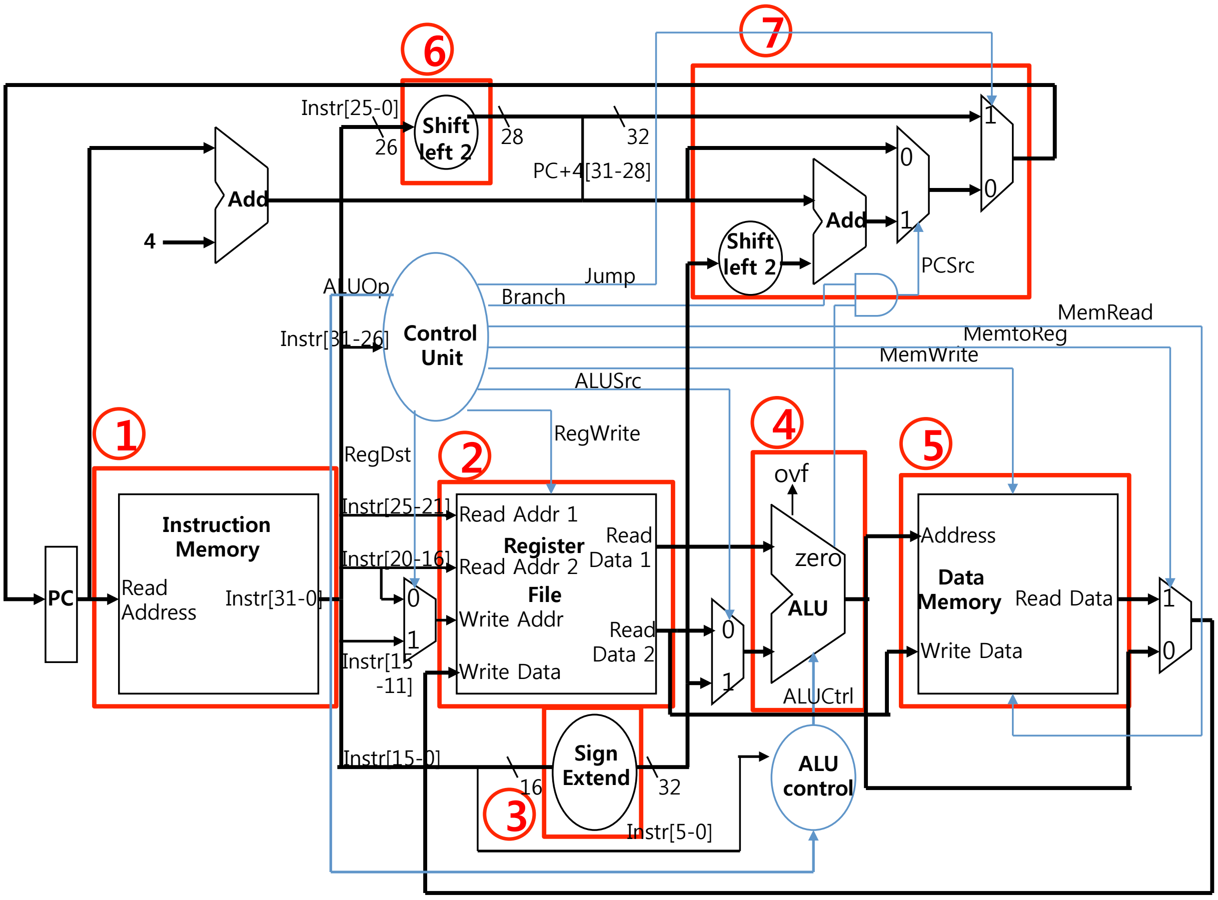


0x14C00000004

Part E: MIPS Datapath and Control Logic (70 points)

Question 8 (30 points)

The following figure shows the datapath and control of a single-cycle MIPS processor as we have covered in the lectures. Answer the following questions.



(1) Identify datapath components (among ①~⑦) that are used when you fetch the following instructions: (a) LW, (b) ADD, (c) BEQ. For each instruction, write down those component numbers and explain their functions.

LW : 1, 2, 3, 4, 5

- **1 : Instruction fetch**
- **2 : Read register operands, update register**
- **3 : 16bit off set -> 32bit**
- **4 : calculate address (using 16-bit offset)**
- **5 : Read memory**

ADD : 1, 2, 4

1 : Instruction fetch
2 : Read 2 register operands, Write register result
4 : perform arithmetic operation

BEQ : 1, 2, 3, 4, 7
1 : Instruction fetch
2 : Read register operands
3 : Sign-extend displacement
4 : compare operands
7 : word displacement(shift left 2 places), add to PC+4

(2) Fill in the control signals in the table below for the **addi** instruction using 1, 0, and X (don't care). If you want, you can write the answer to the helper sheet and submit it together. Also, find datapath components that are *not* used for **addi** (Write down the number(s).)

Instruction opcode	**ALU ctrl**
Add	0010
Sub	0110
AND	0000
OR	0001
Set on less than	0111

Table. ALU control signal bit

addi : 5, 6, and 7 are not used

Instruction	**RegDst (1 bit)**	**ALUSrc (1 bit)**	**MemtoReg (1 bit)**	**RegWrite (1 bit)**	**MemRead (1 bit)**	**MemWrite (1 bit)**	**Branch (1 bit)**	**ALUctrl[*] (4 bits)**
addi	**0**	**1**	**0**	**1**	**0** **(X)**	**0**	**0**	**0010**

Question 9 (40 points)

The following code is a simple insertion sort program written in MIPS assembly, which will be run on a single-cycle MIPS processor shown in Question 8. The input array has 4 integers with {6, 9, 7, 4}. Registers are initialized to 0.

```
.data
input: .word 6, 9, 7, 4
.text
main:
             la $s0, input          # $s0 = Base address of A[]
             li $s1, 4              # Length of A
             li $s2, 1
Loop1:
             add $t0, $s2, $s2
             add $t0, $t0, $t0
             add $t0, $t0, $s0
             lw $s4, 0($t0)
        ①    addi $s3, $s2, -1
             add $t1, $s3, $s3
             add $t1, $t1, $t1
             add $t1, $t1, $s0
Loop2:
             lw $t3, 0($t1)
        ②    slt $t6, $t3, $s4
             bne $t6, $zero, Break

             sw $t3, 4($t1)
             addi $t1, $t1, -4
        ③    addi $s3, $s3, -1
             slt $t7, $s3, $zero
             beq $t7, $zero, Loop2
Break:
             sw $s4, 4($t1)
             addi $s2, $s2, 1
        ④    slt $t8, $s2, $s1
             bne $t8, $zero, Loop1
```

(1) The table below summarizes the delay of each datapath component. (a) Which instruction determines the cycle time? (b) What will the minimum cycle time?

Instruction Fetch	Register read	ALU Operation	Memory Access	Register Write
100 ps	50 ps	100 ps	100 ps	50 ps

(a) lw
(b) 400 ps

(2) Write down the execution sequence of the loop in the dotted box (e.g., ① → ② → …) and total instruction count. (*Note*: Do not count the three instructions outside the box.)

Loop sequence : 1 - 2 – 4 – 1 – 2 – 3 – 2 – 4 – 1 – 2 – 3 – 2 – 3 – 2 – 3 – 4 (Helper sheet)

	①	②	③	④	**Total**
# of iteration	**3**	**6**	**4**	**3**	**16**
IC	**8**	**3**	**5**	**4**	
Total IC	**24**	**18**	**20**	**12**	**74**

(3) Ben Bitdiddle is the chief designer of the MIPS processor product line at SKK Electronics, and he proposes two new instructions to optimize this program: `sltmem` (`slt` with **mem**ory operand), `swpd` (`sw` with **p**ost-**d**ecrement). If we apply these instructions whenever applicable, what will be the new instruction count?

```
sltmem $t1, $s0, $t0, $t2   # $t1 <- ($s0 < Mem[$t0]) ? 1 : 0,
                            # $t2 <- Mem[$s0]

swpd   $s0, 8($t0)          # Mem[$t0+8] <- $s0, $t0 <- $t0 - 4
```

	①	②	③	④	**Total**
# of iteration	**3**	**6**	**4**	**3**	**16**
IC	**8**	**2**	**4**	**4**	
Total IC	**24**	**12**	**16**	**12**	**64**

(4) What will the CPU time for the original and modified programs? Assume CPI is 1 for both processors, and the CPU time is determined only by instruction count and clock cycle time. Be sure to calculate the cycle time of the modified processor (with `sltmem` and `swpd`) if it is changed. (*Note*: Be sure to include the time unit.)

CPUtime_original = 74 * 1 * 400ps

CPUtime_modified = 64 * 1 * 400ps

# Reference

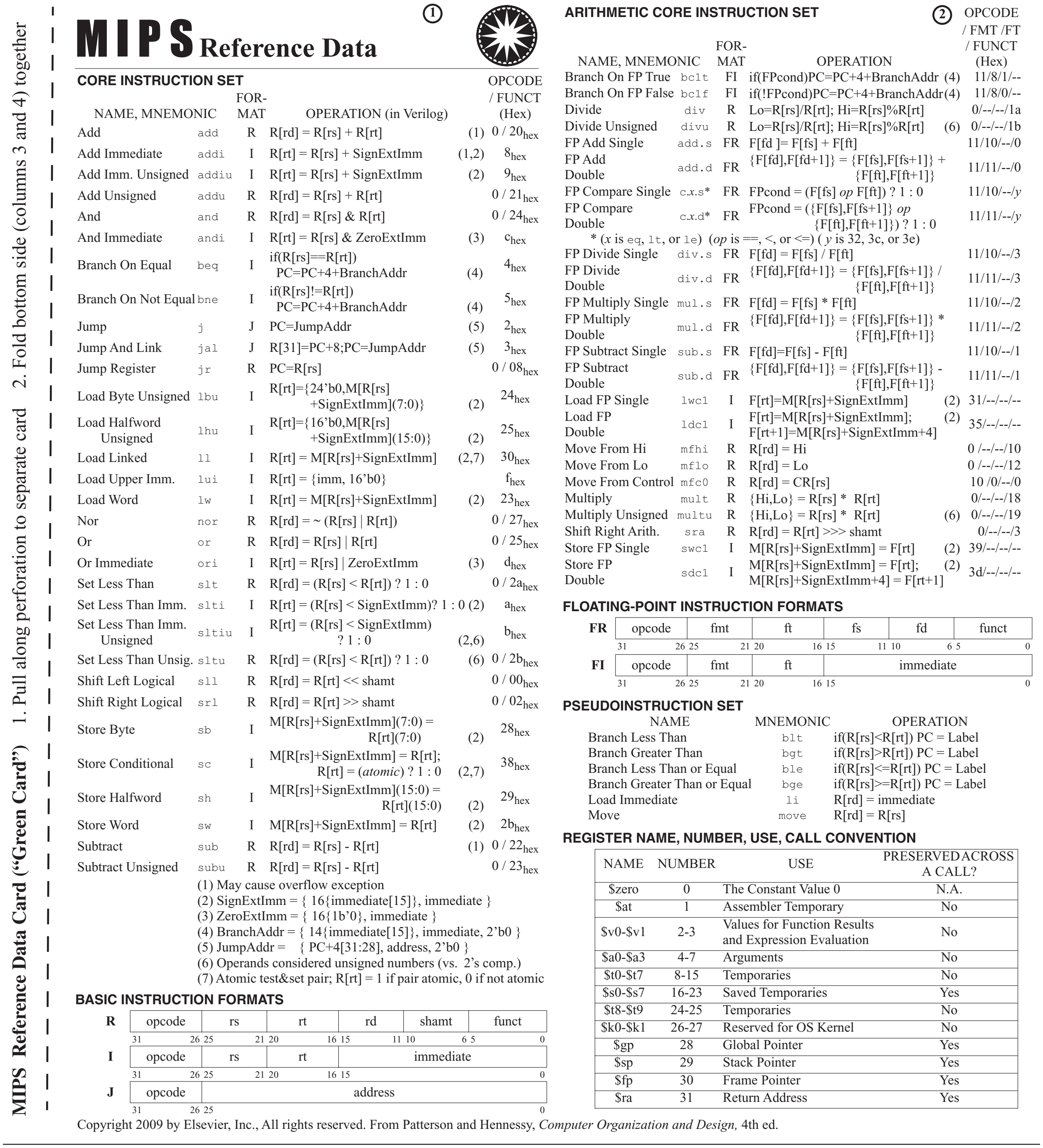
Code No.	$s0	$s1	$s2	$s3	$s4	$t0	$t1	$t3	$t6	$t7	$t8	A[]			
												[0]	[1]	[2]	[3]
1	&A[]	4	1	0	9	&A[1]	&A[0]	0	0	0	0	6	9	7	4
2								6	1			6	9	7	4
4			2								1	6	9	7	4
1				1	7	&A[2]	&A[1]					6	9	7	4
2								9	0			6	9	7	4
3				0			&A[0]			0		6	9	9	4
2								6	1			6	9	9	4
4			3								1	6	7	9	4
1				2	4	&A[3]	&A[2]					6	7	9	4
2								9	0			6	7	9	4
3				1			&A[1]			0		6	7	9	9
2								7	0			6	7	9	9
3				0			&A[0]			0		6	7	7	9
2								6	0			6	7	7	9
3				-1			X &A[-1]			1		6	6	7	9
4			4								0	4	6	7	9

# Reference code (insertion sort)

```
int main() {
        int i, j, v;
        int A[4] = {6, 9, 7, 4};

        for (i = 1; i < 4; ++i) {
                v = A[i];
                for (j = i - 1; j >= 0 && A[j] >= v; --j) {
                        A[j+1] = A[j];
                }
        A[j+ 1] = v;
        }
        return 0;
}
```

Appendix A: MIPS Green Card (Page 1)



MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (columns 3 and 4) together

MIPS Reference Data ①

CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Add	add	R	R[rd] = R[rs] + R[rt]	(1)	0 / 20$_{hex}$
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1,2)	8$_{hex}$
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm	(2)	9$_{hex}$
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]		0 / 21$_{hex}$
And	and	R	R[rd] = R[rs] & R[rt]		0 / 24$_{hex}$
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm	(3)	c$_{hex}$
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4$_{hex}$
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5$_{hex}$
Jump	j	J	PC=JumpAddr	(5)	2$_{hex}$
Jump And Link	jal	J	R[31]=PC+8;PC=JumpAddr	(5)	3$_{hex}$
Jump Register	jr	R	PC=R[rs]		0 / 08$_{hex}$
Load Byte Unsigned	lbu	I	R[rt]={24'b0,M[R[rs] +SignExtImm](7:0)}	(2)	24$_{hex}$
Load Halfword Unsigned	lhu	I	R[rt]={16'b0,M[R[rs] +SignExtImm](15:0)}	(2)	25$_{hex}$
Load Linked	ll	I	R[rt] = M[R[rs]+SignExtImm]	(2,7)	30$_{hex}$
Load Upper Imm.	lui	I	R[rt] = {imm, 16'b0}		f$_{hex}$
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23$_{hex}$
Nor	nor	R	R[rd] = ~ (R[rs] | R[rt])		0 / 27$_{hex}$
Or	or	R	R[rd] = R[rs] | R[rt]		0 / 25$_{hex}$
Or Immediate	ori	I	R[rt] = R[rs] | ZeroExtImm	(3)	d$_{hex}$
Set Less Than	slt	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0		0 / 2a$_{hex}$
Set Less Than Imm.	slti	I	R[rt] = (R[rs] < SignExtImm)? 1 : 0	(2)	a$_{hex}$
Set Less Than Imm. Unsigned	sltiu	I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6)	b$_{hex}$
Set Less Than Unsig.	sltu	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6)	0 / 2b$_{hex}$
Shift Left Logical	sll	R	R[rd] = R[rt] << shamt		0 / 00$_{hex}$
Shift Right Logical	srl	R	R[rd] = R[rt] >> shamt		0 / 02$_{hex}$
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2)	28$_{hex}$
Store Conditional	sc	I	M[R[rs]+SignExtImm] = R[rt]; R[rt] = (*atomic*) ? 1 : 0	(2,7)	38$_{hex}$
Store Halfword	sh	I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2)	29$_{hex}$
Store Word	sw	I	M[R[rs]+SignExtImm] = R[rt]	(2)	2b$_{hex}$
Subtract	sub	R	R[rd] = R[rs] - R[rt]	(1)	0 / 22$_{hex}$
Subtract Unsigned	subu	R	R[rd] = R[rs] - R[rt]		0 / 23$_{hex}$

(1) May cause overflow exception
(2) SignExtImm = { 16{immediate[15]}, immediate }
(3) ZeroExtImm = { 16{1b'0}, immediate }
(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
(5) JumpAddr = { PC+4[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2's comp.)
(7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct
	31 26	25 21	20 16	15 11	10 6	5 0
I	opcode	rs	rt	immediate		
	31 26	25 21	20 16	15		0
J	opcode	address				
	31 26	25				0

ARITHMETIC CORE INSTRUCTION SET ②

NAME, MNEMONIC		FOR-MAT	OPERATION		OPCODE / FMT /FT / FUNCT (Hex)
Branch On FP True	bc1t	FI	if(FPcond)PC=PC+4+BranchAddr	(4)	11/8/1/--
Branch On FP False	bc1f	FI	if(!FPcond)PC=PC+4+BranchAddr	(4)	11/8/0/--
Divide	div	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]		0/--/--/1a
Divide Unsigned	divu	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	(6)	0/--/--/1b
FP Add Single	add.s	FR	F[fd]= F[fs] + F[ft]		11/10/--/0
FP Add Double	add.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}		11/11/--/0
FP Compare Single	c.*x*.s*	FR	FPcond = (F[fs] *op* F[ft]) ? 1 : 0		11/10/--/*y*
FP Compare Double	c.*x*.d*	FR	FPcond = ({F[fs],F[fs+1]} *op* {F[ft],F[ft+1]}) ? 1 : 0		11/11/--/*y*
* (*x* is eq, lt, or le) (*op* is ==, <, or <=) (*y* is 32, 3c, or 3e)					
FP Divide Single	div.s	FR	F[fd] = F[fs] / F[ft]		11/10/--/3
FP Divide Double	div.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}		11/11/--/3
FP Multiply Single	mul.s	FR	F[fd] = F[fs] * F[ft]		11/10/--/2
FP Multiply Double	mul.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} * {F[ft],F[ft+1]}		11/11/--/2
FP Subtract Single	sub.s	FR	F[fd]=F[fs] - F[ft]		11/10/--/1
FP Subtract Double	sub.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}		11/11/--/1
Load FP Single	lwc1	I	F[rt]=M[R[rs]+SignExtImm]	(2)	31/--/--/--
Load FP Double	ldc1	I	F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4]	(2)	35/--/--/--
Move From Hi	mfhi	R	R[rd] = Hi		0 /--/--/10
Move From Lo	mflo	R	R[rd] = Lo		0 /--/--/12
Move From Control	mfc0	R	R[rd] = CR[rs]		10 /0/--/0
Multiply	mult	R	{Hi,Lo} = R[rs] * R[rt]		0/--/--/18
Multiply Unsigned	multu	R	{Hi,Lo} = R[rs] * R[rt]	(6)	0/--/--/19
Shift Right Arith.	sra	R	R[rd] = R[rt] >>> shamt		0/--/--/3
Store FP Single	swc1	I	M[R[rs]+SignExtImm] = F[rt]	(2)	39/--/--/--
Store FP Double	sdc1	I	M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1]	(2)	3d/--/--/--

FLOATING-POINT INSTRUCTION FORMATS

FR	opcode	fmt	ft	fs	fd	funct
	31 26	25 21	20 16	15 11	10 6	5 0
FI	opcode	fmt	ft	immediate		
	31 26	25 21	20 16	15		0

PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	if(R[rs]<=R[rt]) PC = Label
Branch Greater Than or Equal	bge	if(R[rs]>=R[rt]) PC = Label
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVEDACROSS A CALL?
\$zero	0	The Constant Value 0	N.A.
\$at	1	Assembler Temporary	No
\$v0-\$v1	2-3	Values for Function Results and Expression Evaluation	No
\$a0-\$a3	4-7	Arguments	No
\$t0-\$t7	8-15	Temporaries	No
\$s0-\$s7	16-23	Saved Temporaries	Yes
\$t8-\$t9	24-25	Temporaries	No
\$k0-\$k1	26-27	Reserved for OS Kernel	No
\$gp	28	Global Pointer	Yes
\$sp	29	Stack Pointer	Yes
\$fp	30	Frame Pointer	Yes
\$ra	31	Return Address	Yes

Appendix A: MIPS Green Card (Page 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 ④

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

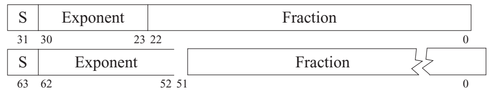
where Single Precision Bias = 127,
Double Precision Bias = 1023.

IEEE 754 Symbols

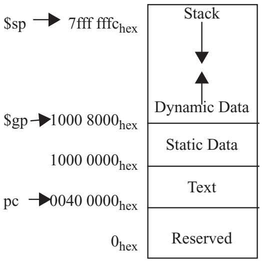
Exponent	Fraction	Object
0	0	± 0
0	≠0	± Denorm
1 to MAX - 1	anything	± Fl. Pt. Num.
MAX	0	±∞
MAX	≠0	NaN

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

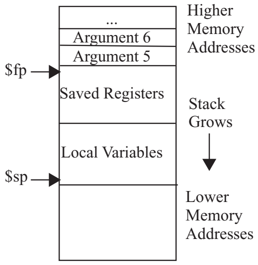
IEEE Single Precision and Double Precision Formats:



MEMORY ALLOCATION



STACK FRAME

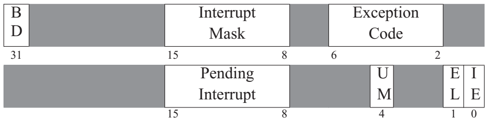


DATA ALIGNMENT

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 (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.

MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (columns 3 and 4) together

Appendix B: Helper sheet for Question 8

Question 8: (Control signals for `addi`)

Instruction	RegDst (1 bit)	ALUSrc (1 bit)	MemtoReg (1 bit)	RegWrite (1 bit)	MemRead (1 bit)	MemWrite (1 bit)	Branch (1 bit)	ALUctrl* (4 bits)
addi								

(This page is intentionally left blank. Feel free to use as you like.)