Take Exam 1 - Requires Respondus LockDown Browser + Webcam

Due Oct 7 at 3:35pm **Points** 100 **Questions** 26
Available Oct 7 at 1:15pm - Oct 12 at 3:40pm 5 days **Time Limit** 140 Minutes
Requires Respondus LockDown Browser

Instructions

1. Read each question carefully.
2. Show your work on short answer questions as needed. No work, no credit. You can show your work using the Canvas editor, but if you prefer to use paper to show the work, make sure to upload it to Canvas using the assignment link provided in the Exam 1 module. If you use a paper for scratch work, please upload it too.
3. You can use
   1. A simple calculator (Not your phone). There is a calculator provided in the quiz. You may want to use it.
   2. The green sheet and the syscall table provided in the exam.

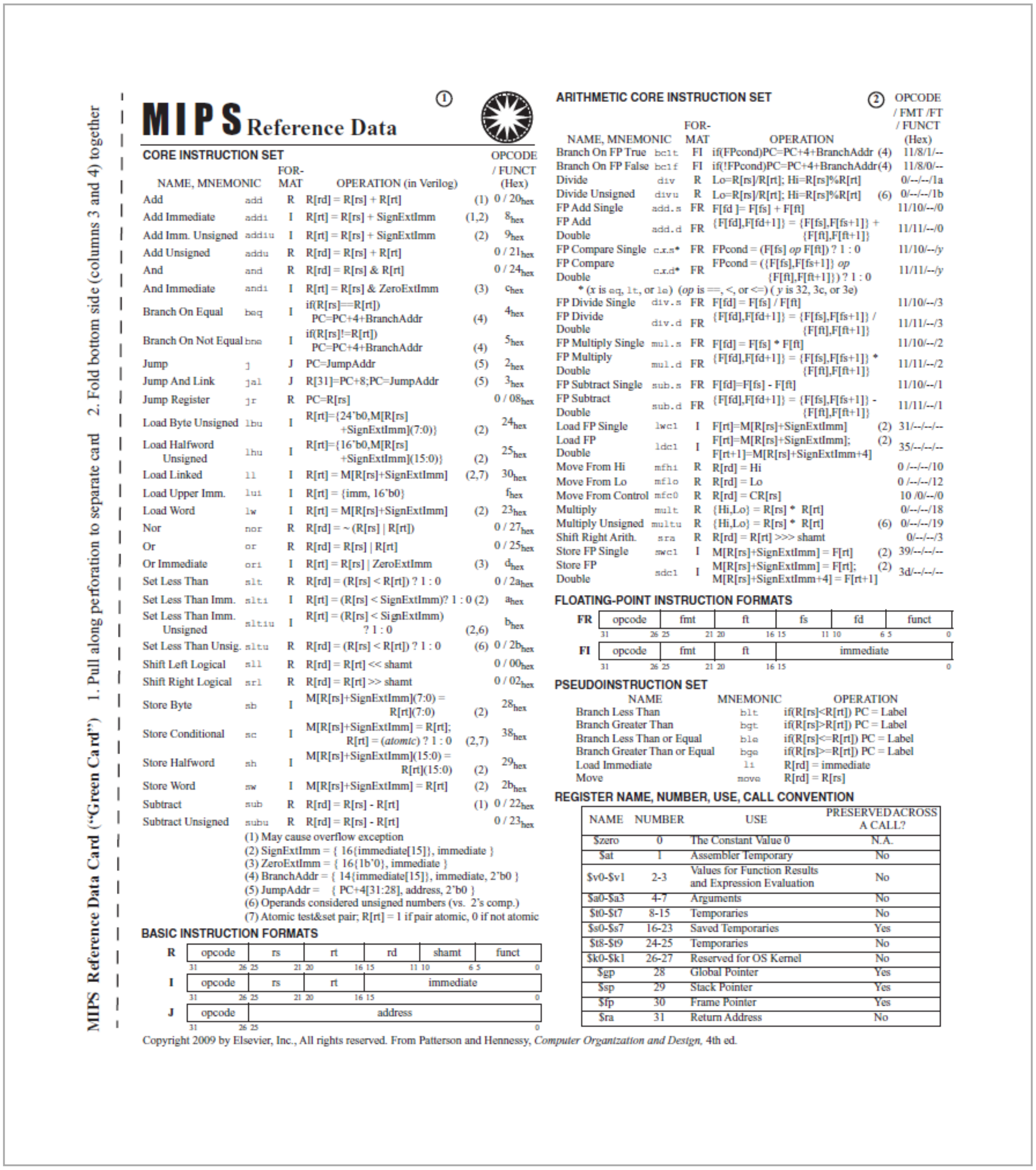
If you are not sure about a question, explain what you understood and the rationale behind your answer.

I have read these instructions and am ready to begin the test

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	85 minutes	95.5 out of 100

Score for this quiz: **95.5** out of 100
Submitted Oct 7 at 2:44pm
This attempt took 85 minutes.



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				

Copyright 2009 by Elsevier, Inc., All rights reserved. From Patterson and Hennessy, *Computer Organization and Design*, 4th ed.

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)

NAME, MNEMONIC		FOR- MAT	OPERATION		OPCODE / FMT /FT / FUNCT (Hex)
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

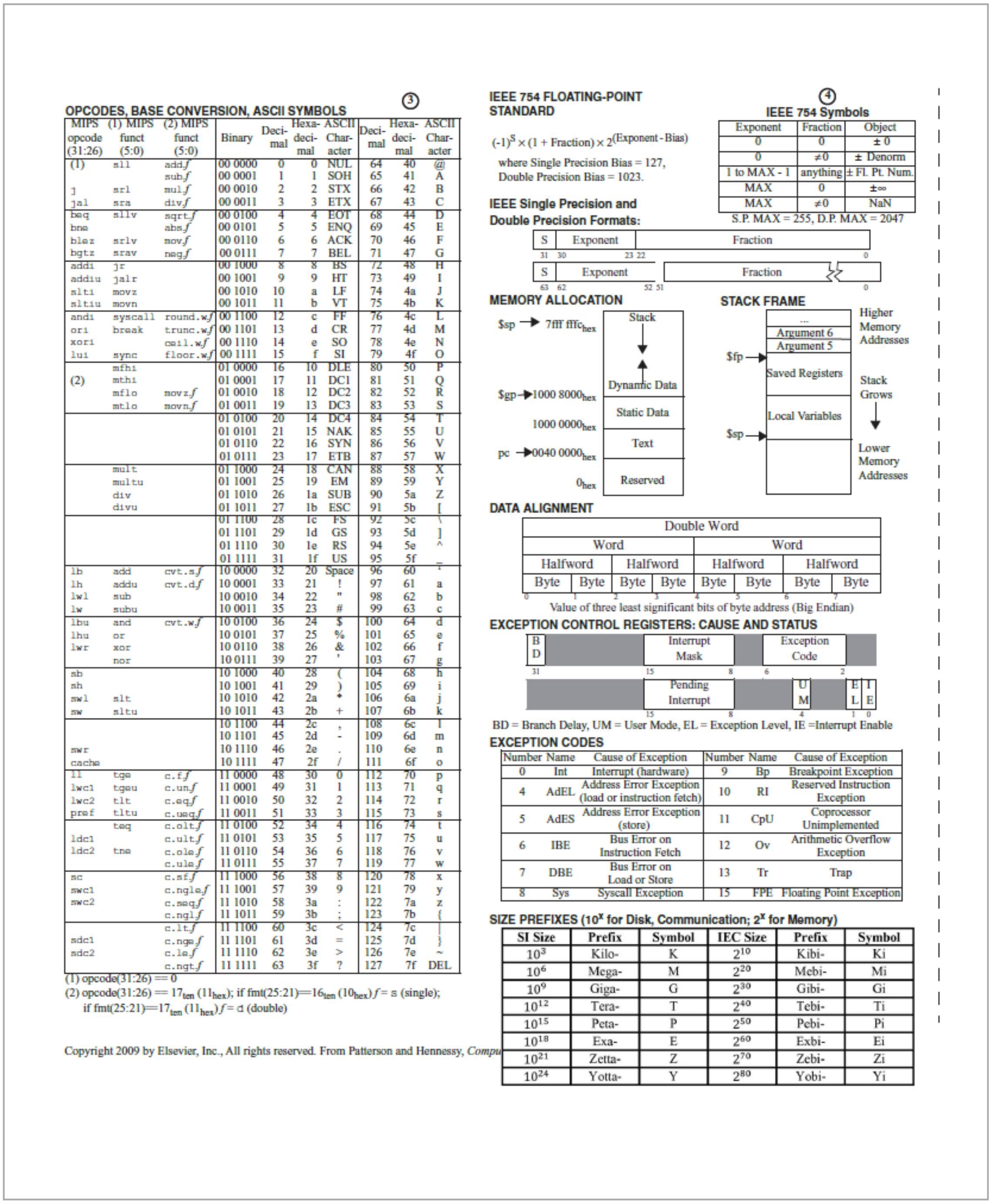
FR	opcode	fmt	ft	fs	fd	funct
	31 26	25 21	20 16	15 11	10 6	5 0
FI	opcode	fmt	ft	immediate		
	31 26	25 21	20 16	15 0		

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	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	No



③

OPCODES, BASE CONVERSION, ASCII SYMBOLS

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)

Copyright 2009 by Elsevier, Inc., All rights reserved. From Patterson and Hennessy, *Compu*

④

IEEE 754 FLOATING-POINT STANDARD

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

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

IEEE 754 Symbols

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

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

IEEE Single Precision and Double Precision Formats:

S	Exponent	Fraction
31	30 23	22 0

S	Exponent	Fraction
63	62 52	51 0

MEMORY ALLOCATION

$sp → 7fff fffc_{hex} — Stack

Dynamic Data

$gp → 1000 8000_{hex}

Static Data

1000 0000_{hex}

Text

pc → 0040 0000_{hex}

Reserved

0_{hex}

STACK FRAME

Higher Memory Addresses

...

Argument 6

Argument 5

$fp →

Saved Registers

Local Variables

$sp →

Stack Grows

Lower Memory Addresses

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

B D		Interrupt Mask		Exception Code	
31		15 8		6 2	

	Pending Interrupt		U M		E L	I E
	15 8		4		1	0

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)

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

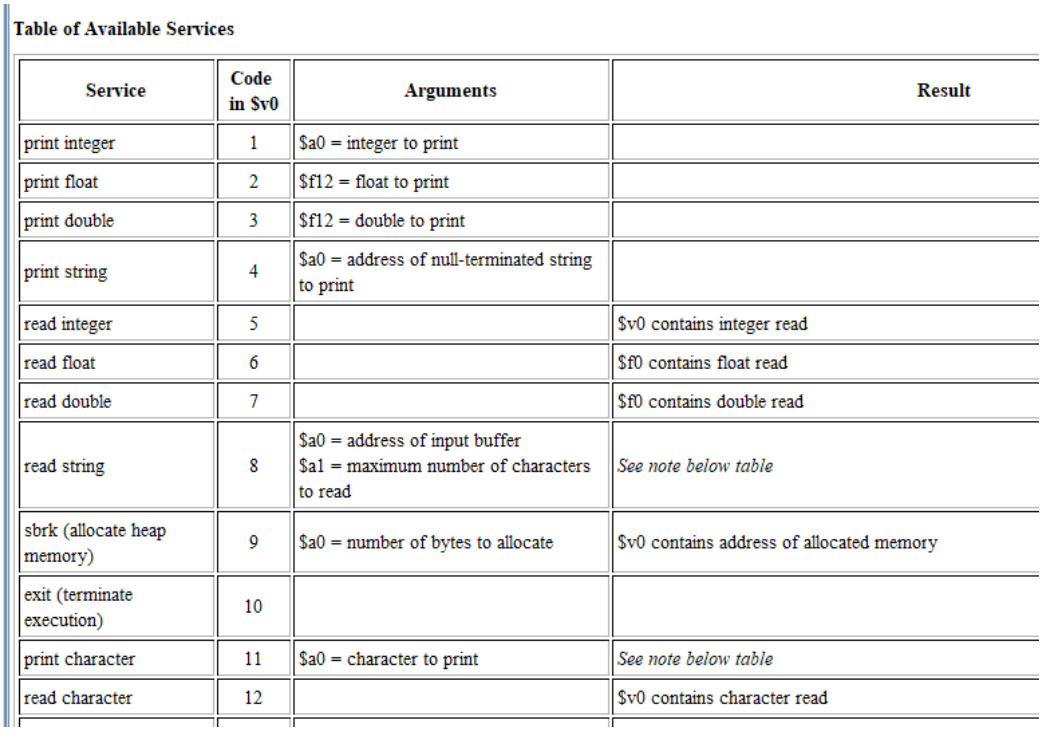


Table of Available Services

Service	Code in $v0	Arguments	Result
print integer	1	$a0 = integer to print	
print float	2	$f12 = float to print	
print double	3	$f12 = double to print	
print string	4	$a0 = address of null-terminated string to print	
read integer	5		$v0 contains integer read
read float	6		$f0 contains float read
read double	7		$f0 contains double read
read string	8	$a0 = address of input buffer $a1 = maximum number of characters to read	*See note below table*
sbrk (allocate heap memory)	9	$a0 = number of bytes to allocate	$v0 contains address of allocated memory
exit (terminate execution)	10		
print character	11	$a0 = character to print	*See note below table*
read character	12		$v0 contains character read

Question 1

3 / 3 pts

The 32-bit number 0xFF00AA11 will be stored as follows using the Big-Endian Method

	11	AA	00	FF
Address	4000	4001	4002	4003

○ True

Correct! ● False

Question 2

3 / 3 pts

Which list of instructions computes 3x+7, where x starts out in register $8 and the result is put in $9?

Correct!

●

```
ori $3,$0,3
mult $8,$3
mflo $9
addiu $9,$9,7
```

○

```
ori $3,$0,3
mult $8,$3
mfhi $9
addiu $9,$9,7
```

○

```
ori $3,$0,3
mult $8,$3
addiu $9,$8,7
```

○

```
mult $8,3
mflo $9
addiu $9,$9,7
```

Question 3

0 / 3 pts

What is the main difference between the section of memory that holds instructions and the section of memory that holds data?

○ Data uses virtual memory; instructions use physical memory.

ou Answered

● Data memory is connected to the data bus. Instruction memory is connected to the instruction bus.

○ Data memory is arranged into bytes with addresses; instruction memory holds words without addresses.

orrect Answer

○ There is no difference. All memory does is hold bit patterns. It is up to the rest of the computer system to determine what those patterns mean.

Question 4

3 / 3 pts

What is the correct number of steps in the machine cycle?

○ increment, fetch, execute, increment

Correct!

● fetch instruction, decode, fetch operand, execute, and store result

○ load, compile, run

○ wash, rinse, spin dry"

Question 5

3 / 3 pts

In a right _______________, the sign bit (the MSB in two's complement) is shifted in on the left, thus preserving the sign of the operand.

○ logical shift.

○ R-Type Shift

○ binary shift.

Correct! ● arithmetic shift

Question 6

3 / 3 pts

What are the Five Classic Components of the Computer?

○ D. Input, Output, Memory, ALU, Registers

○ C. CPU, ALU, Registers, Memory, PC

○ B. ALU, Registers, Memory, PC, Instruction Register

Correct! ● A. Input, Output, Memory, Datapath and Control.

Question 7

3 / 3 pts

Which of the following assembler directives reserves 3 words of memory?

Correct! ● .space 12

○ .block 3

○ .word 3

○ .byte 12

Question 8

3 / 3 pts

A model for modern computer organization where both data & instructions are stored in the same place (shared memory) is called

○ RISC

Correct!

● Von Neumann architecture

○ .data and .text

○ MIPS

Question 9

3 / 3 pts

Which of the following are **not** one of the Four ISA Design Principles

○ Make the common case fast.

Correct!

● Go big or go home.

○ Good design demands good compromises.

○ Simplicity favors regularity.

Question 10

3 / 3 pts

How many bits are in a nibble?

Correct!

● 4

○ 8

○ 2

○ 16

Question 11

3 / 3 pts

Which of the following statements are correctly following the MIPS register usage convention when writing subroutines

Correct!

☑ Subroutine parameters should be stored in registers $a0, ...$a3

□ Subroutine parameters should be stored in registers $s0, ...$s3

□ subroutine should return results in registers $t0.....$t3

Correct!

☑ subroutine should return results in registers $v0 and /or $v1.

Question 12

3 / 3 pts

What is it called when bit 15 of a 16-bit immediate operand is copied to the 16 bits to its left to form a 32-bit operand?

○ Bit extention

○ Zero extention

○ Immediate extention

Correct! ● Sign extention

Question 13

4 / 4 pts

What is the bit-wise XOR of the following pattern?

1110 1101

0101 0110

Your Answer:

xor: exclusive or

XOR table:

A	B	
0	0	0
0	1	1
1	0	1
1	1	0

ANSWER: 1011 1011

Question 14

5 / 5 pts

Add the following two **bytes (8 bits)** together using **2’s complement arithmetic**. You can show your work here in this editor. Check your answer.

–12 + (-32)

Your Answer:

WORK ON PAPER

ANSWER: 11010100 = -44

Question 15

6 / 6 pts

Do the following multiplication operation using binary shift operations. Show each step.

8 * 13

Your Answer:

DONE ON PAPER.

ANSWER: 01101000 = 104

Question 16

6 / 6 pts

Write an instruction that would **set** bits 8 through 11 in register $8 and leave the remaining bits unchanged.

Your Answer:

0000 0000 0000 0000 0000 0000 0000 0000 BITS START AT 0

0000 0000 0000 0000 0000 1111 0000 0000 = 0x00000F00

Setting bits uses or. clearing bits uses AND

ori $t1, 0xF00 # this will extend F00 to a 32 bit by using sign extension. giving 0x00000F00

ANSWER: or $t0, $t0, $t1

Question 17

5 / 5 pts

The move command is a Pseudo Instruction. Convert the Pseudo Instruction into an actual instruction that will accomplish the same thing.

(Do **NOT** convert it into Machine Code)

move $t0, $s1

Your Answer:

ANSWER: or $t0, $s1, $zero

Question 18

4.5 / 5 pts

Convert the following Machine Code into Assembly language instruction. Show your work.

0x012a402b

Your Answer:

DONE ON PAPER.

ANSWER: sltu $t0, $t1, $t3

Question 19

4 / 6 pts

The following hexadecimal number represents a floating-point number in EEE 32 Bit Floating Point Format. Convert that number to binary and then convert it to base 10. Show each step.

0xC2090000

Your Answer:

DONE ON PAPER.

ANSWER: -66.03515625

Question 20

2 / 2 pts

One of the five components of a computer is "Datapath". What is it? Explain.

Your Answer:

Datapath is the route the data takes through memory and the registers, and eventually to the ALU, back into registers, and then to memory.

Question 21

5 / 5 pts

Given the data values shown below in the .data segment, write MIPS instructions that store the sum of the values labeled mary, jim, and sue into the location labeled result. It then prints the result as an output.

```
.data
    mary:     .word   17
    jim:      .word  -99
    sue:      .word   10
    result:   .word    0
```

Your Answer:

```
.text

# Loads all data into registers

lw $s0, mary

lw $s1, jim

lw $s2, sue


# Adds the values in the registers together
```

```
add $s3, $s0, $s1

add $s3, $s3, $s2


# Stores into the result label

sw $s3, result
```

Question 22

5 / 5 pts

The following Load Word instruction is used to move a word from memory to a register. Explain what each element of the instruction is used for. i.e. the $t0, 4 and ($t1)

```
lw $t0, 4($t1)
```

It is similar to the assignment statement

```
int x = myInts[4];
```

Your Answer:

lw: the pseudoinstruction to load a word, or 32 bits, into a register

$t0, the register $8, is a temporary register, which is the destination register in which the next value will be stored.

4: the offset amount from the base address of the register within parentheses. Gives the value at that address in the array.

($t1): presumably an array. This register is being offset by 4, and the value at that offset address is being stored in $t0.

Question 23

5 / 5 pts

Write the following code snippet given in C++ in Assembly. You can assume that the variable *input* is assigned to $t0 register.

```
if (input % 7 == 0 && input > 0)
    printf("Good Rabbit");
else
   printf("Bad Panda");
```

Your Answer:

```
#Puts 7 in the t1 register, divides, then moves the remainder (from the hi register) into $s0.

ori $t1, $zero, 7

div $t0, $t1

mfhi $s0


# Sets s3 to 1 if s0 (input % 7) is equal to 0. Then sets s1 to 1 if t0 (input) is greater than 0.

seq $s3, $s0, $zero

sgt $s1, $t0, $zero


#performs an and operation on s1 and s3, which hold either 1 or 0 from the condition checks, then stores in s4

and $s4, $s1, $s3


# branch if s4, the result of the and operation is 0. START IF

beqz $s4, ENDIF

   #prints Good Rabbit

   ori $v0, $zero, 4

   la $a0, true

   syscall
```

```
#prints Bad Panda

ENDIF:

    ori $v0, $zero, 4

    la $a0, false

    syscall

#ENDIF


#exit

ori $v0, $zero, 10

syscall



.data

true: .asciiz "Good Rabbit"

false: .asciiz "Bad Panda"
```

Question 24

5 / 5 pts

Write the following method as a simple MIPS subprogram. Make sure to document it well.

/* Returns the first parameter plus second parameter multiplied by 16 */

```
int meth (int x, int y)
{
   return  x +  y * 16  ;
}
```

Your Answer:

#Author: Josh Hutchinson

#Subroutine: meth

#Purpose: receive two values through $a0 and $a1, then return $a0 + ($a1 * 16) through v0

#side effects: $a0, $a1, and $v0 will be used and altered. otherwise none

#Multiply $a1, or y, by the immediate value of 16.

multi $a1, 16

#move the product from the lo register into $a1.

mflo $a1

#add $a0 and $a1, and store into v0. then return.

add $v0, $a0, $a1

jr $ra

Question 25

5 / 5 pts

Write an assembly language program that is equivalent to the following code snippet. Hint: You can use these registers as is in your code.

```
if($t0 > 6)
{
   $v0 = 0;
   $a1++;
}
else
{
  $v0 = $s0;
  $a1 = 0;
}
```

Your Answer:

#Author: Josh Hutchinson

#Date: 10/07/2020

#Purpose: Basic if else statement in assembly

.text

.globl main

main:

#check if t0 is greater than 6. store 1 in s1 if it is, otherwise store 0 in s1

sgti $s1, $t0, 6

#START IF BLOCK

#Checks if $s1 = 0, branches to ELSE if it is.

beqz $s1, ELSE

#sets $v0 to 0 using and. (could also use ori $v0, $zero, 0). increments $a1 by 1.

andi $v0, $v0, 0

addi $a1, $a1, 1

# Branches if t0 was not greater than 6

```
ELSE:

#Sets $v0 to $s0

or $v0, $s0, $zero


#and used for clearing bits. could also do ori $a1, $zero, 0. and seems cleaner though.

andi $a1, $a1, 0


#END IF


#exit

ori $v0, $zero, 10

syscall
```

Question 26

0 / 0 pts

1 Point Extra Credit.

Assume that the address of a particular word in memory is word-aligned. Is bit 0 (i.e, the LSB) of the address 0 or 1? Why?

Your Answer:

The address is 0, as the bits start at 0. When memory is word-aligned, the addresses start at the beginning of the next word addres.

Quiz Score: **95.5** out of 100

This quiz score has been manually adjusted by +1.0 points.