WASHINGTON STATE UNIVERSITY

School of Electrical Engineering and Computer Science

CptS260: Introduction to Computer Architecture - Fall 2016

Final Exam, Duration: 120 Minutes, December 13, 2016

NAME: **ID:**

Notes:

- You may bring a double-side letter-size cheat-sheet and a calculator to the test. No other resources are allowed! In particular, NO textbooks, lecture notes, internet access, smartphone usage, etc. are allowed!
- MIPS reference data sheet is provided.
- The test includes 5 implicit bonus points.
- Make sure to write your name and WSU ID down on all pages.
- Show your work for each question. Even if you don't know exact answer to a question, show your work to get partial credit.
- You are strongly encouraged to complete the course evaluation before the deadline of 12/16/2016. Your feedback is very much appreciated!

Problem	1	2	3	4	5	6	7	8
Total Points	10	10	10	15	10	20	15	15
Points Earned								

Total Points Earned: .....

1. **Concepts (10 points).** For each sentence below, indicate if it is a *True* or *False* statement.

   a. Non-negative numbers have the same unsigned and 2's-complement representation.
   True

   b. In MIPS assembly, instruction 'beq $1, $2,1' will advance the program counter (PC) by one (i.e., PC = PC + 1) if [$1]=[$2]. (note: [] indicates content of the register)
   False, $PC_{new} = PC_{old} + 4$

   c. In MIPS assembly, instruction 'jr' will update register '$ra' by copying content of the program counter (PC) into '$ra'.
   False, PC = $ra, PC is updated by $ra.

   d. MIPS is a byte-addressable computer.
   True

   e. Compared to an un-pipelined processor, a pipelined architecture usually has a shorter clock cycle.
   True

   f. Control hazards in a pipelined architecture are due to data dependency of the instructions.
   False, Control hazards are due to branch instruction.

   g. DRAM (Dynamic Random Access Memory) is as fast as SRAM (Static Random Access Memory)
   False, DRAM is slower than SRAM.

   h. A fully associative cache memory has a higher hit time (i.e., it is slower) compare to a direct mapped cache memory of the same size.
   True, FA is slower than direct mapped

   i. For primary (L1) cache in multilevel cache systems, focus is on minimizing hit time.
   True

   j. Conflict misses are not affected by cache size because they arise from blocks from main memory mapping to the same position in the cache memory.
   True, the number of blocks from main memory mapping remains the same.

2. **Computer Performance (10 points).** Assume a processor has a D-Cache (Data Cache Memory) with 10% miss rate and an I-Cache (Instruction Cache Memory) with 15% miss rate. Also, assume that access to these L1 cache memories takes 1ns while access to the main memory takes 60ns.

   a. (*5 points*) Assume that 30% of the instructions are load/store. Calculate the effective CPI assuming an ideal/base CPI of 2 and a clock cycle time of 1ns.
   b. (*5 points*) If we add an L2 cache memory with 2% miss rate and 5ns access time, what would be the new effective/actual CPI?

a - $CPI = CPI_{ideal} + Stalls_{I\text{-}cache} + Stalls_{D\text{-}cache}$

$CPI = 2 + 15\% \times \frac{60\,ns}{1\,ns} + 30\% \times 10\% \times \frac{60\,ns}{1\,ns}$

$CPI = 2 + 9 + 1.8 = \boxed{12.8}$

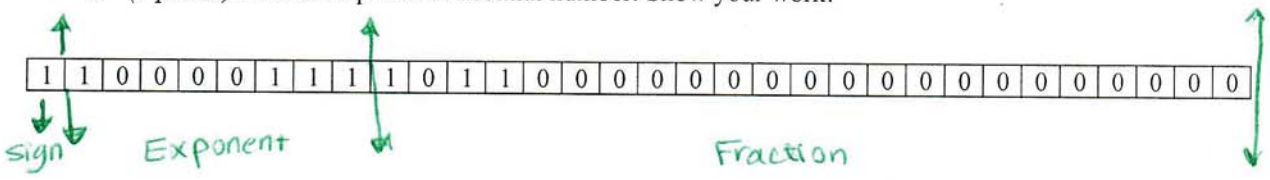
b - Miss penalty of $(L1 + L2) = 5ns + 2\% \times 60ns = 6.2ns$

$CPI = 2 + 15\% \times \frac{6.2\,ns}{1\,ns} + 30\% \times 10\% \times \frac{6.2\,ns}{1\,ns}$

$CPI = 2 + 0.93 + 0.186 = \boxed{3.1116}$ $\frac{cycles}{instructions}$

3. **Number Representation (10 points).** Assume the following 32-bit binary represents a floating point number using single precision IEEE 754 standard.

   a. *(3 points)* Specify fraction field, exponent field, and sign bit of this number in binary.
   b. *(7 points)* Find the equivalent decimal number. Show your work!



a- sign = 1

Exponent = $(10000111)_2$ = 135

Fraction = $(10110\ldots0)_2$

b- Decimal = $(-1)^{Sign} \times (1.\text{Fraction}) \times 2^{Exponent - 127}$

$(-1)^1 \times (1.1011) \times 2^{135-127} = -(1.1011) \times 2^8 = -(110110000)_2$

$= -432$

4. **MIPS Assembly Programming (15 points)**. Trace the following assembly program. Assume that the registers' initial values and memory contents are as shown in the following tables. The program starts from location 10000d in the instruction memory. Show the final state of the processor including registers' content/data, and memory contents.

Address	Initial Value	Final Value
100d	0d	40d
104d	1d	44d
108d	2d	48d
112d	3d	52d
116d	4d	56d
…	…	…
200d	10d	40d
204d	11d	45d
208d	12d	52d
212d	13d	61d
216d	14d	72d

Register	Initial Value	Final Value
$t0	0	120d
$t1	0	220d
$t2	0	5d
$t3	0	5d
$s0	0	4d
$s1	0	56d
$s2	0	72d

tmp = 4yi

yi = xi² + tmp ⟶ Function of code

xi = tmp

```
        li      $t3, 5d
        li      $t2, 0d
        la      $t0, 100d
        la      $t1, 200d
loop:   lw      $s0, 0($t0)
        lw      $s1, 0($t1)
        mult    $s0, $s0
        mflo    $s2
        sll     $s1, $s1, 2d
        add     $s2, $s2, $s1
        sw      $s2, 0($t1)
        sw      $s1, 0($t0)
        addi    $t0, $t0, 4
        addi    $t1, $t1, 4
        addi    $t2, $t2, 1
        bne     $t2, $t3, loop
exit:
```

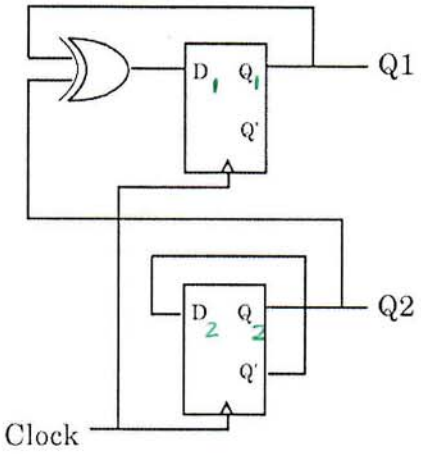
5. **Sequential Logic (10 points).** The following figure shows a sequential circuit with two D Flip Flops with a common input clock. Assume that the flip flops are initialized at '0'. That is, $Q_1 = Q_2 = 0$ during Cycle 0. This means the output of this circuit is initially Q_2Q_1 = '00'.

   a. *(5 points)* Compute the value of each output signal (Q_2 and Q_1) for 8 cycles using the table below.

   b. *(5 points)* Convert the 3-bit binary Q_2Q_1 to its equivalent decimal in the table. How many unique output states (i.e., Q_2Q_1) does this circuit produce?

$D_1 = Q_1 \oplus Q_2$

$D_2 = \overline{Q_2}$

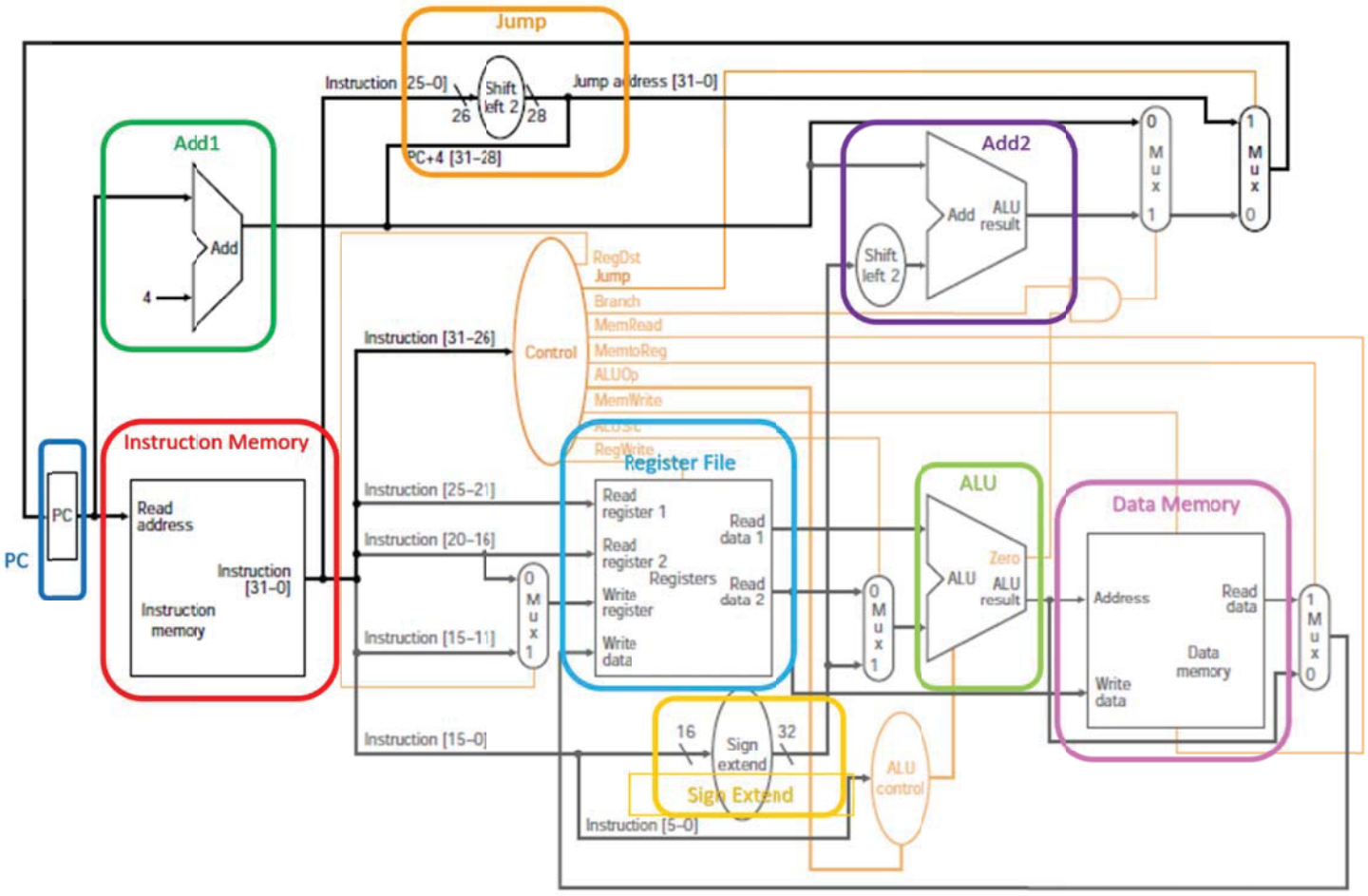


$Q_1 \times (2)^0 + Q_2 \times (2)^1$

Clock Cycle	Q_2	Q_1	Decimal ($Q = Q_2Q_1$)
0	0	0	0
1	1	0	2
2	0	1	1
3	1	1	3
4	0	0	0
5	1	0	2
6	0	1	1
7	1	1	3
8	0	0	0

6. **Simple MIPS Architecture (20 points)**. Consider the following 3 instructions are to be executed though the MIPS architecture shown below. Assume that $t2 = 24d prior to execution of this short program.

```
li      $t1, 12d
sub     $s0, $t2, $t1
bne     $0, $s0, 1048d
```



a. Fill out the table below to show which of the hardware blocks are used (i.e., are active) during execution of each instruction. Each hardware block is shown by a box in the next figure.

	PC	Instruction Memory	Register File	Sing Extend	ALU	Data Memory	ADD1	ADD2	Jump
li	✓	✓	✓	✓	✓		✓		
sub	✓	✓	✓		✓		✓		
bne	✓	✓	✓	✓	✓		✓	✓	

b. Fill out the table below to show which of the control signals in the figure should be activated (set to 1) during execution of each instruction. You do not need to specify inactive or unknown (i.e. '0' and 'x' values) signals in this part.

	RegDst	Jump	Branch	MemRead	MemtoReg	MemWrite	RegWrite
li	0	0	0	0	0	0	1
sub	1	0	0	0	0	0	1
bne	X	0	1	0	X	0	0

c. Assume the program starts from address 1032d (i.e., address 1032 decimal) in the instruction memory. Compute the value of PC after each instruction is executed.

	Program Counter (PC) Content
li	1032d + 4d = 1036d
sub	1036d + 4d = 1040d
bne	(1040d + 4d) + 4 x 1048d = 5236d

$s0 is not equal to zero, so branch is taken

$PC_{new} = (PC_{old} + 4) + \text{sign-extend}(\text{branch address}) \ll 2$

7. **Pipelining (15 points).** Consider the following instruction sequence executing on the 5-stage MIPS pipeline of form IF, ID, EX, MEM, WB.

```
lw    $t2, 60($t1)
lw    $t1, 40($t2)
slt   $t1, $t1, $t2
sw    $t1, 20($t2)
```

a. Assume that the pipeline does not use any forwarding. Complete the following pipeline execution diagram for the above instruction sequence, showing the flow of instructions through the pipeline during each clock cycle. Indicate any necessary stalls on the pipeline diagram. How many cycles does it take to complete execution of the instruction sequence? What is the overall CPI?

No forwarding, dependency needs 2 stalls

Instruction / Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14
lw $t2, 60($t1)	IF	ID	EX	MEM	WB									
lw $t1, 40($t2)		•	•	IF	ID	EX	MEM	WB						
slt $t1, $t1, $t2					•	•	IF	ID	EX	MEM	WB			
sw $t1, 20($t2)								•	•	IF	ID	EX	MEM	WB

14 cycles, $CPI = \frac{\text{cycles \#}}{\text{instruction \#}} = \frac{14}{4} = 3.5 \frac{\text{cycles}}{\text{instruction}}$

b. Now, assume that all possible forwarding paths have been added to the pipeline. Redraw the pipeline execution diagram below. In the diagram show forwarding by arrows. How many cycles does it take to finish the instruction sequence? What is the overall CPI?

Forwarding, 1 stall for lw dependency and none for the rest

Instruction / Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14
lw $t2, 60($t1)	IF	ID	EX	MEM	WB									
lw $t1, 40($t2)		•	IF	ID	EX	MEM	WB							
slt $t1, $t1, $t2				•	IF	ID	EX	MEM	WB					
sw $t1, 20($t2)						IF	ID	EX	MEM	WB				

10 cycles, $CPI = \frac{10}{4} = 2.5 \frac{\text{cycles}}{\text{instruction}}$

8. **Cache Memory (15 points).** Consider a 2-way associative cache memory with a total capacity of 32 words. The block size is 2 words and the processor has an 8-bits address bus. Assume that the processor is only word-addressable meaning that the offset field of the address needs to distinguish between the two words within a block. The following table is intended to show the cache content (in columns labeled as 'Way-1' and 'Way-2') as well as the line/set address (in column labeled as 'Index').

Index	Way-1			Way-2		
	Valid	Tag	Data	Valid	Tag	Data
000	1	0010	MEM[32] MEM[33]			
001						
010						
011						
100	1	0000	MEM[8] MEM[9]			
101						
110	1	0001 ~~0010~~	MEM[28] ~~MEM[44]~~ MEM[29] ~~MEM[45]~~	1	0000	MEM[12] MEM[13]
111		0000	MEM[14] MEM[15]			

a. *(5 points)* Assume that the cache contains data for two memory references with addresses 8 and 44. Update the above table for these addresses. (*Hint:* when you reference a memory location, you will need to move a block that contains this memory address).

$8 = (\underbrace{0000}_{tag}\ \overbrace{1000}^{index})_2$

$44 = (\underbrace{0010}_{tag}\ \overbrace{1100}^{index})_2$

b. *(10 points)* Now assume that the CPU accesses the following memory references (i.e, numbers from left to right represent memory addresses)

8, 14, 33, 12, 33, 32, 13, 14, 29, 9

Update the table from part (a) with valid bit, tag, and data fields in appropriate rows/places to show the cache content. Also, fill out the following table and mention if a reference to each location/address is a cache hit or cache miss. Assume an LRU (Least Recently Used) replacement policy for this cache.

Word Address	Binary Address	Tag	Index	Hit/Miss
8	00001000	0000	100	*Hit*
14	00001110	0000	111	*Miss*
33	00100001	0010	000	*Miss*
12	00001100	0000	110	*Miss*
33	00100001	0010	000	*Hit*
32	00100000	0010	000	*Hit*
13	00001101	0000	110	*Hit*
14	00001110	0000	111	*Hit*
29	00011101	0001	110	*Miss*
9	00001001	0000	100	*Hit*

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.

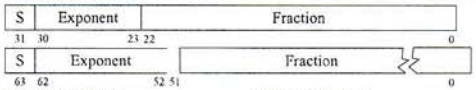
④

IEEE 754 Symbols

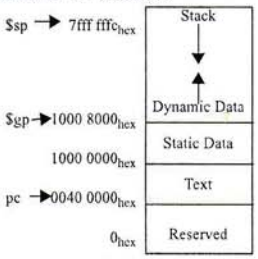
Exponent	Fraction	Object
0	0	± 0
0	≠0	± Denorm
1 to MAX - 1	anything	± Fl. Pt. Num.
MAX	0	±∞
MAX	≠0	NaN

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

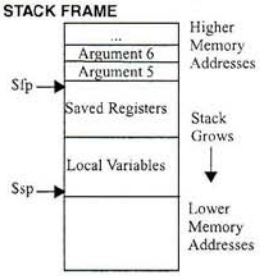
IEEE Single Precision and Double Precision Formats:



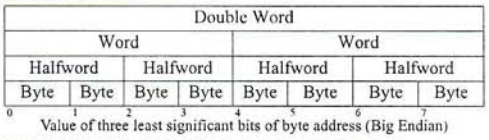
MEMORY ALLOCATION



STACK FRAME

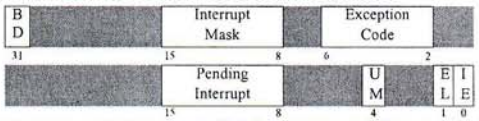


DATA ALIGNMENT



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.

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

MIPS Reference Data

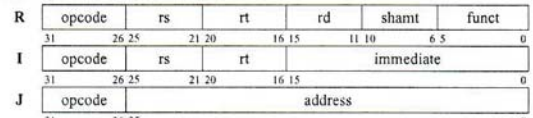
①

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

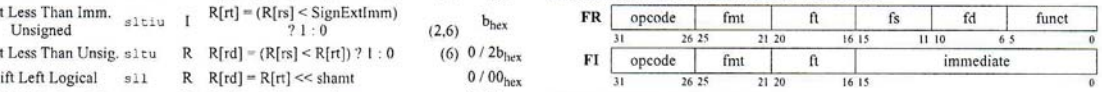


ARITHMETIC CORE INSTRUCTION SET

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



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	Yes