TEMPLATE for GRADING

Bilkent University

Computer Engineering Department

CS224: COMPUTER ORGANIZATION

Midterm Date: November 15, 2020 Time: 14:30 – 17:00

Student Name: ID No.: Section:	KEY

GOOD LUCK!

Notes: 1. There are 100 points, 5 questions on 8 pages.

2. Please READ the questions and the notes below.

3. It is an open textbook exam. You can only use the textbook.

4. You can only us four function simple calculators.

5. You cannot share calculators.

6.A copy of the MIPS green card is provided at the end of the exam. You can detach and use it. You can keep it after the exam.

7. There is an empty page after the questions. You may use it for your work.

8. Show your work. You have to provide a legible neat work.

9. You cannot leave the exam room in the first 30 minutes.

10. You may take a picture of your midterm pages when you are allowed by your exam proctor.

11. Please do not write anything in the following table.

Question No.	Q1	Q2	Q3	Q4	Q5	Overall
Points Possible	20	25	25	15	15	100
Your Grade						

Question 1. (20 points): MIPS Programming and Floating Point Numbers

If you think that there is something wrong in a question explain.

a. **(10 points)** In MARS assembler/simulator we want to have another pseudo instruction with the mnemonic jle: jump less than or equal.

For example, "jle $t0, $t1, done". This instruction sets the PC equal to the address of the label done, i.e., it jumps to label done, if $t0 ≤ $t1. Show its implementation below in the provided box. If needed you may also use additional labels such as L1 etc. You are not allowed to use any pseudo instruction. An efficient implementation with minimal number of instructions is essential during grading.

jump far away

```
==>   jle     $t0, $t1, done  # Give the 3 generated instruction below.
    slt $at, $t1, $t0
    bne $at, $zero, L1   # true means        $t0 ≤ $t1 ≡
                         # that $t1<$t0      not($t1 < $t0) ≡
    j done                                   not ($t0>$t1)
L1:
```

```
    slt $at, $t1, $t0                      not($t1 < $t0) ≡
    beq $at, $zero, done  ) accepted         $t1 >= $t0
```

b. **(6 points)** Consider the following single precision IEEE floating point number given in hexadecimal C4 02 D0 00. Give its normalized binary number representation and its decimal equivalent in the following boxes.

Norm. binary:	1.0000 0101 101 × 2^9
Decimal:	-523.25

C4 02 D0 00

1100 0100 0000 0010 1101 0000 0000 0000

s 8 8 → 1.00000101101 × 2^9

88_{16}
$- 7F_{16}$
09 ← exponent

1 00000 1011 . 0100
2 0 B . 4_{16}

⇒ 20B.4_{16}

$2 \times 16^2 + 0 + 11 \times 16^0 + 4/16$

512 + 11 + 0.25 ⇒ 523.25

c. **(4 points)** You are given the following MIPS machine instruction (in hex): 8D 28 FF F4. Disassemble this machine instruction and give its symbolic machine instruction equivalent in the following box.

FFFF
- FFF4
000B
+ 1
C

$8 -12 $9

lw $t0, 0xFFF4($t1)

8D 28 FF F4

1000 1101 0010 1000 1111 1111 1111 0100

23_{16} $9 $8 imm

lw

$9 = $t1

$8 = $t0

I-type
opcode/rs/rt/imm

lw rt, imm(rs)

Question 2. (25 points): MIPS Code Generation and Program Tracing

a. **(9 points)** Consider the following code segment.

```
L1:     j       L1
        la      $t0, sum        # sum is defined in the data segment
        add     $t0, $t0, $t0
next:   add     $t0, $t0, $t0
        beq     $t0, $t1, L1
        add     $t0, $t0, $t0
```

jump: opcode/address 00 40 00 AC 1100 not stored

not stored

00 0010 0000 0100 0000 0000 0000 1010 11

0 8 1 0 0 0 2 B

Assume that L1 is at memory location 0x 00 40 00 AC. Give the MIPS machine instruction that will be generated for "j L1" and " beq $t0, $t1, L1" in the following box in hex.

L1:	j L1 # Give the generated machine instruction below in hex. 0X 08 10 00 2B
	beq $t0, $t1, L1 # Give the generated machine instruction below in hex. 0X 11 09 FF FA

b. **(16 points)** Consider the following code segment. Note that MIPS text segment begins at memory location 0x 00 40 00 00 and the data segment begins at memory location 0x 10 01 00 00.

beq -6 instructions
14 hex

beq $8, $9, -6

-6 ⇒ 0000 0110
1111 1001
1
1111 1010

00 0100 01000 01001, FF FA
1 1 0 9

11 09 FF FA

```
        .text
main:   lw      $t0, a
        la      $t1, array
        lw      $t2, n
        li      $t4,0
L1:     beq     $t2, $0, L2
        lw      $t3, 0($t1)
        addi    $t1, $t1, 4
        addi    $t2, $t2, -1
        bgt     $t3, $t0, L1
        add     $t4, $t3, $t4
        j       L1
L2:     ...
        .data
array:  .word   0, 1, 3, 5, 10, 20
n:      .word   ... # its value is specified below in the following table
a:      .word   ... # its value is specified below in the following table
```

data segment

array
0
1
3
5
10
20
n ← n is here
a ← a is here

What will be the contents of the registers $t1, $t2, $t3, and $t4 (in hex.) when we reach L2 during execution time for the values of the variables a and n as shown in the following table.

a	n	$t1	$t2	$t3	$t4
5_{10}	6_{10}	0X 10 01 00 18	0X 0 - - - 0	0X 00 00 00 14 20_{10}	0X 00 00 00 09
10_{10}	8_{10}	0X 10 01 00 20	0X 0 - - 0	0X 00 00 00 0A 10_{10}	X 00 00 00 25 37_{10}

Data Segment Memory Address

hex	
10 01 00 00	array(0): 0
04	1
08	3
0C	5
10	10
14	20
18	n
1C	a
20	-

For a=5, n=6

$t0	$t1	$t2	$t4
5	0X 00 40 00 00	6	0

Finds the summation of array elements greater than n (6), when exits the loop $t1 points to n

variable	address (hex)
array	0X 00 40 00 00
n	0X 00 40 00 18
a	0X 00 40 00 1C
loc after a	0X 00 40 00 20

The first case (a=5, n=6): find sum of array elements ≤ a (5)

The second case (a=10, n=8): finds sum of array elements and next two consecutive values (10,8) ≤ a (10)

Question 3. (25 points): MIPS Single-Cycle New Instruction Implementation

In this question consider the single-cycle MIPS architecture and the implementation of a new I-type MIPS instruction called waps (opcode is 38_{10}). An example of waps is given below.

waps $t1, 12($t2)

The contents of some registers and some memory locations before and after executing the above instruction is given in the following table. The table shows that before executing the instruction register $8 contains 0x00cc00bc and after executing the instruction it still contains 0x00cc00bc. The memory location with the address 0x00cc00cc, before executing the instruction, contains 0x000000cc; after executing the instruction it still contains 0x000000cc.

	Register Contents			Memory Address			
	$8	$9	$10	0x00cc00cc	0x00cc00d0	0x00cc00d4	0x00cc00d8
Contents Before Execution ==>	0x00cc00bc	0x00000000	0x00cc00cc	0x000000cc	0x000000d0	0x00000000	0x0000000b
Contents After Execution ==>	0x00cc00bc	0x0000000b	0x00cc00cc	0x000000cc	0x000000d0	0x00000000	0x00000000

modified modified

a. **(5 points)** Generate the machine instruction that corresponds to the symbolic instruction

waps $t1, 12($t2)

rt rs

in hex in the following box. Show your work.

0X 99 49 00 0C

waps I-type
opcode/rs/rt/imm
$38_{10} = 26_{16}$ 10 9 12

10 0110 01010 01001 00 ... 1100
9 9 4 9 0 0 0 C

b. **(8 points)** Give the RTL description that defines how the instruction works.

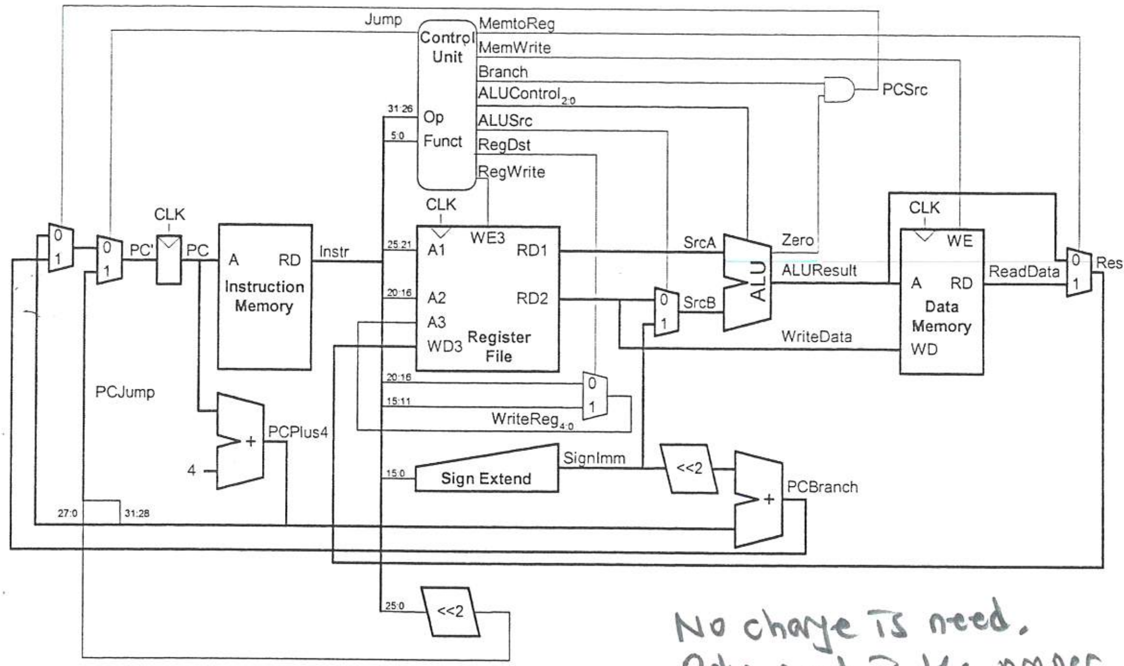
The table shows that the content of the 1st and 2nd operands are exchanged (swapped)!

IM [PC]
RF [rt] ← DM [RF [rs] + signExt (imm)]
DM [RF [rs] + signExt (imm)] ← RF [rt]
PC ← PC + 4

Note:
rt = cannot be $zero
the assembler would give an error message for a case like that,
$0 = 0 always

See the next page for the continuation of the question.

c. **(12 points)** Consider the following MIPS single-cycle architecture given in the textbook. Provide the necessary changes to implement the new instruction waps.



No change Is need.
Only need Is the proper settings of the control signals.

During grading an implementation with the minimal number of changes is essential. For the new control lines and new hardware units explain their purpose.

State the values of the following control signals for the waps instruction. If you have new control signal(s) state their name(s), and value. If needed add additional rows to the following line.

(Bin.)

	Control Line	Value
1	Jump	0
2	MemtoReg	1
3	MemWrite	1
4	Branch	0
5	ALUControl	010
6	ALUSrc	1
7	RegDst	0
8	RegWrite	1

1. Jump=0, not a jump instruction
2. MemtoReg=1, send contents of memory to RF
3. MemWrite=1, write to memory
4. Branch=0, not a branch instruction
5. ALUControl= , perform an addition
6. ALUSrc=1, use SignImm as SrcB
7. RegDst= 0, write to rt register
8. RegWrite= 1, write to reg. file

Question 4. (15 points): MIPS Programming for Linked Lists: Completing Missing Parts

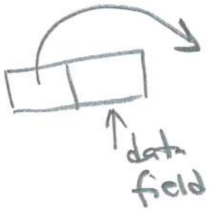
Consider a singly linked list structure with nodes containing two fields: First the link field, then followed by a data field of size one word. Complete the subprogram replaceWithSum given below which replaces the data field of each node with the summation of the data fields starting with that node. For example, if the linked list data fields contain the following numbers 1, 2, 3, 4 (the data field of the first node contains 1 etc.); after executing the subprogram the updated data fields contain the following values 10, 9, 7, 4. This means that the data field of the first node now contains 10 after executing replaceWithSum, etc.

For implementing replaceWithSum assume that a subprogram called findSum is available as a library routine: It returns a value which is equal to the summation of data fields of the linked list starting from the node pointed by the only parameter of the subprogram. For example, if linked list data fields contain the following numbers 1, 2, 3, 4 and if we pass the address of the second node to findSum it returns 9.

Your task is completing the missing parts of the subprogram replaceWithSum. Note that it does not use any $t registers or pseudo instructions. You cannot add new instructions. The parameter passing and returning results from methods follow the traditions of MIPS programming.

```
replaceWithSum:
            addi   $sp, $sp, -8
            sw     $s0, 4($sp)
            sw     $ra, 0($sp)
            add    $s0, $zero, $a0
L1:         beq    $s0, $zero, L2
            add    $a0, $s0, $zero
            jal    findSum
            sw     $v0, 4($s0) / 4($a0)←
            lw     $s0, 0($s0) / 0($a0)←
            j      L1              # jump ...
L2:         lw     $ra, 0($sp)
            lw     $s0, 4($sp)
            addi   $sp, $sp, 8
            jr     $ra             # Program end here
```

OK but not recommended since $a0 can be changed by findSum



Question 5. (15 points): MIPS Recursive Program Tracing

Consider the following program segment.

```
             $a0
        lw   $a, m    # Variable m is defined in the data segment
        lw   $a1, n   # Variable n is defined in the data segment
        jal  subProgram
        ...
#-----------------------------------------
subProgram:
        bge   $a0, $a1, next
        move  $v1, $a0
        jr    $ra                    # <== Instruction no. 1
# Give the values of the cells at this point. (Read the rest to understand this.)
next:
        sub   $sp, $sp,4
        sw    $ra, 0($sp)
        addi  $v0, $v0,1
        sub   $a0, $a0,$a1
        blt   $a0, $a1, done
        jal   subProgram             # <== Instruction no. 2
done:
        move  $v1, $a0
        lw    $ra, 0($sp)
        add   $sp, $sp,4
        jr    $ra                    # <== Instruction no. 3
```

saves $ra 3 times for below line
= 3 times done = 3 times inst 3

Trace the code given above and fill in the empty cells of the following table when we execute the program with m and n values given in the first two columns of the table. Under $v0 and $v1 give the contents of these registers after completing the execution. Under the column "Inst. No. 1" give the number of times the instruction given on that line is executed when subProgram is executed for the m and n given in the first two columns. Do the same for the last two columns for "Inst. No. 2" and "Inst. No. 3".

m	n	$v0	$v1	Inst. No. 1	Inst. No. 2	Inst. No. 3
4	3	1	1	0	0	1
15	4	3	3	0	2	3
38	7	5	3	0	4	5

If you think that we have an infinite loop explain why and state how to fix it.

Page number 8 is intentionally left blank.

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

(1)

MIPS Reference Data

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	6 opcode	rs 5	rt 5	16 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)

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	Yes

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)

④

IEEE 754 FLOATING-POINT STANDARD

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

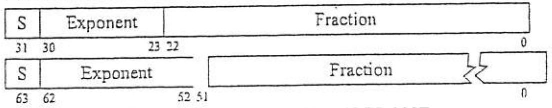
where Single Precision Bias = 127,
Double Precision Bias = 1023.

IEEE 754 Symbols

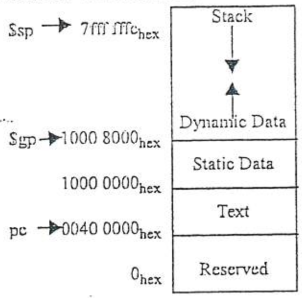
Exponent	Fraction	Object
0	0	± 0
0	≠0	± Denorm
1 to MAX - 1	anything	± Fl. Pt. Num.
MAX	0	±∞
MAX	≠0	NaN

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

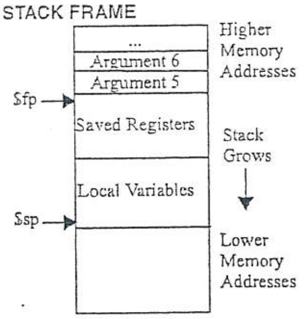
IEEE Single Precision and Double Precision Formats:



MEMORY ALLOCATION



STACK FRAME



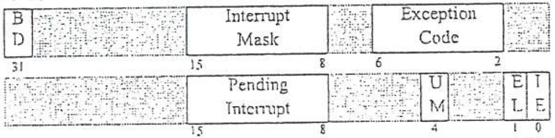
DATA ALIGNMENT

<table>
<tr><td colspan="8">Double Word</td></tr>
<tr><td colspan="4">Word</td><td colspan="4">Word</td></tr>
<tr><td colspan="2">Halfword</td><td colspan="2">Halfword</td><td colspan="2">Halfword</td><td colspan="2">Halfword</td></tr>
<tr><td>Byte</td><td>Byte</td><td>Byte</td><td>Byte</td><td>Byte</td><td>Byte</td><td>Byte</td><td>Byte</td></tr>
</table>

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