Team #: Name:_____

Absent:

0) Compare zero-, one-, two-, and three-address machines by writing programs to compute

$$X = (A + B) / (D - E * B)$$

for each of the four machines. The instructions available for use are as follows:

3 Address	2 Address	1 Address (Accumulator machine)	0 Address (Stack machine)
MOVE (X ← Y)	MOVE (X ← Y)	LOAD M	PUSH M
		STORE M	POP M
ADD (X ← Y + Z)	ADD (X ← X + Y)	ADD M	ADD
SUB (X ← Y - Z)	SUB (X ← X - Y)	SUB M	SUB
MUL (X ← Y * Z)	MUL (X ← X * Y)	MUL M	MUL
DIV (X ← Y / Z)	DIV (X ← X / Y)	DIV M	DIV

Team #: Name:_____

Absent:

1) How relocatable (i.e., can it be moved in memory) is the code in memory if **direct addressing** is used?

2) How many bits are needed to represent a direct address on a 32-bit machine?

3) From your programming experience, what range of Integer values would cover 90% of the constant Integer values used in your programs?

4) How many binary bits would you need to represent this range of values?

5) What determines how many bits are needed to represent a register in a machine language instruction?

6) Which of the following programming language constructs would be good candidates for using PC-relative addressing?

a) conditional branch used when implementing loops

b) calling a subprogram

c) accessing a global variable

d) accessing a local variable

Team #: Name:_____

Absent:

Type of Instruction	MIPS Assembly Language	Register Transfer Language Description
Memory Access (Load and Store)	lw $4, Mem	$4← [Mem]
	sw $4, Mem	Mem←$4
	lw $4, 16($3)	$4← [Mem at address in $3 + 16]
	sw $4, Mem	[Mem at address in $3 + 16]← $4
Move	move $4, $2	$4← $2
	li $4, 100	$4← 100
Load Address	la $5, mem	$4← load address of mem
Arithmetic Instruction (reg. operands only)	add $4, $2, $3	$4← $2 + $3
	mul $10, $12, $8	$10← $12 * $8 (32-bit product)
	sub $4, $2, $3	$4← $2 - $3
Arithmetic with Immediates (last operand must be an integer)	addi $4, $2, 100	$4← $2 + 100
	mul $4, $2, 100	$4← $2 * 100 (32-bit product)
Conditional Branch	bgt $4, $2, LABEL (bge, blt, ble, beq, bne)	Branch to LABEL if $4 > $2
Unconditional Branch	j LABEL	Always Branch to LABEL

MIPS Logical Instructions	
and $4, $5, $6	$4←$5 (bit-wise AND) $6
andi $4, $5, 0x5f	$4←$5 (bit-wise AND) $5f_{16}$
or $4, $5, $6	$4←$5 (bit-wise OR) $6
ori $4, $5, 0x5f	$4←$5 (bit-wise OR) $5f_{16}$
xor $4, $5, $6	$4←$5 (bit-wise Exclusive-OR) $6
xori $4, $5, 0x5f	$4←$5 (bit-wise Exclusive-OR) $5f_{16}$
nor $4, $5, $6	$4←$5 (bit-wise NOR) $6
not $4, $5	$4←NOT $5 #inverts all the bits

MIPS Shift and Rotate Instructions	
sll $4, $5, 3	$4←shift left $5 by 3 positions. Shift in zeros (only least significant 5-bits of immediate value are used to shift)
sllv $4, $5, $6	Similar to sll, but least significant 5-bits of $6 determine the amount to shift.
srl $4, $5, 3	$4← shift right $5 by 3 positions. Shift in zeros
srlv $4, $5, $6	Similar to srl, but least significant 5-bits of $6 determine the amount to shift.
sra $4, $5, 3	$4←shift right $5 by 3 positions. Sign-extend (shift in sign bit)
srav $4, $5, $6	Similar to sra, but least significant 5-bits of $6 determine the amount to shift.
rol $4, $5, 3	$4←rotate left $5 by 3 positions
rol $4, $5, $6	Similar to above, but least significant 5-bits of $6 determine the amount to rotate.
ror $4, $5, 3	$4←rotate right $5 by 3 positions
ror $4, $5, $6	Similar to above, but least significant 5-bits of $6 determine the amount to rotate.

Team #: Name:_____

Absent:

MIPS Example

Fibonacci Sequence:	0	1	1	2	3	5	8	13	21
Position in Sequence:	0	1	2	3	4	5	6	7	8

A high-level language program to calculate the nth fibonacci number would be:

```
temp2 = 0
temp3 = 1

for i = 2 to n do
    temp4 = temp2 + temp3
    temp2 = temp3
    temp3 = temp4
end for

result = temp4
```

Trace of Program (time →)

HLL variables								MIPS registers
temp2	0	1	1	2	3	5	8	$2
temp3	1	1	2	3	5	8	13	$3
temp4	1	2	3	5	8	13	21	$4
i	2	3	4	5	6	7	8	$6

A complete assembly language MIPS program to calculate the nth fibonacci number.

```
                .data
n:              .word  8          # variable in memory
result:         .word  0          # variable in memory

                      .text
                      .globl main
main:       li    $2, 0
            li    $3, 1
            lw    $5, n # load "n" into $5
            li    $6,
for_loop:   bgt   $6, $5, end_for
            add   $4, $2, $3
            move  $2, $3
            move  $3, $4
            addi  $6, $6, 1
            j     for_loop
end_for:
            sw    $4, result

            li    $v0, 10         # system code for exit
            syscall
```