CSSE 232 – Computer Architecture I
Rose-Hulman Institute of Technology
Computer Science and Software Engineering Department

Exam 1 Solutions

Name:_____ Section: 1 2

This exam is **closed book**. You are allowed to use the reference card from the book. You may not use a computer, calculator, or pda during the examination, except you may use a calculator **only** for the last question.

Write all answers on these pages — use the back if necessary. Be sure to **show all work** and document your code. Do not use instructions that we have not covered (e.g. no `mult` or `div` but you can use `sll`, `srl`).

MIPS code is judged both by its correctness and its efficiency. You may use MIPS pseudoinstructions when writing MIPS code. However, the length of your instruction sequence is judged by the actual instructions to which the sequence expands.

All numbers are expressed in decimal unless specifically stated otherwise.

You are encouraged to read the entire exam before beginning.

	Points available	Your marks
1	16	
2	12	
3	16	
4	20	
5	18	
6	18	
Total	100	

Problem 1 (16 points) Consider the following MIPS assembly code with partial machine language translation to the right.

```
fact:   addi    $sp, $sp, -8
        add     $t0, $t1, $t2
        sw      $ra, 4($sp)
        sw      $a0, 0($sp)
        slti    $t0, $a0, 1
        beq     $t0, $zero, L1
        addi    $v0, $zero, 1
        addi    $sp, $sp, 8
        jr      $ra
L1:     addi    $a0, $a0, -1
        jal     fact
        lw      $a0, 0($sp)
        lw      $ra, 4($sp)
        addi    $sp, $sp, 8
        mul     $v0, $a0, $v0
        jr      $ra
```

0x8	29	29	-8		
0x0	9	10	8	XX	0x20
0x2b	29	31	4		

3
1

0x0	31	XX	XX	XX	0x8

0x3	0x010 0009

This code is loaded into memory starting at address `0x0040 0024`.

(a) (10 points) Fill in the missing values in the instructions above. Give the opcode, function code and JumpAddr in hex notation, and all others in decimal. Mark a field with **XX** if the entry does not matter.

See above.

(b) (3 points) How would you change the code if the branch instruction needed to branch to address `0x004f 0000`? Explain why.

The target address is out of range for a branch instruction (i.e. more than 2^15 words away. A `j` or a `jr` instruction can be used.

(c) (3 points) Convert the instruction below to a 32-bit hex number. All values are decimal except the last which is hex.

op	rs	rt	rd	shamt	funct
0	8	9	10	0	0x20

$\Longrightarrow$ **0x01095020**

Problem 2 (3 x 4 points = 12 points) For each pseudo-instruction in the following table, give a minimal sequence of actual MIPS instructions to accomplish the same thing. You may need to use $at for some of the sequences.

Pseudo-instruction	Description	

`lw $t1, A($t2)`	A is a 32-bit immediate value. Read the value from memory at address (A + contents of `$t2`) and store result in register `$t1`.	`lui $at, UH(A)`
		`ori $at, $at, LH(A)`
		`add $at, $at, $t2`
		`lw $t1, 0($at)`
		
		or
		
		`lui $at, UH(A)`
		`add $at, $at, $t2`
		`lw $t1, LH(A)($at)`

`ble $t1, $t2, Label`	Conditionally branch to instruction at `Label` if value in register `$t1` is less than or equal to the value in register `$t2`. `Label` is less than 2^{15} words away.	`slt $at, $t2, $t1` `beq $at, $zero, label`
`li $t2, 0x12345678`	Load `0x12345678` into register `$t2`.	`lui $t2, 0x1234` `ori $t2, $t2, 0x5678`

Problem 3 (16 points) The following questions examine exceptions on the MIPS processor.

(a) (4 points) What actions does a MIPS processor take when an exception is encountered? Be sure to give details.

Execution jumps to the exception handler (at address `0x8000 0180`). The address of the instruction that caused the exception is stored in the `EPC` register. The cause of the exception is stored in the `cause` register. The exception level is set to 1.

(b) (4 points) How would a systems level software developer determine what caused an exception and where it occurred?

The cause is recorded in the `cause` register. The location of the exception is recorded in the `EPC` register. `mfc0` and `mtc0` can be used to get the value of the `cause` and `EPC` registers.

(c) (4 points) Are there any restrictions or limitations that systems level software developers should be aware of while handling an exception? Explain one of these limitations.

Except for $k0 and $k1, and possibly $at, all registers must be saved before they are used. In general, the stack is not available.

(d) (4 points) Consider the following two chunks of of MIPS assembly code.

```
        .data
mask:   .word   0xffff0000
        .text
start:  lw      $t0, mask
        lw      $s0, branch
        and     $s0, $s0, $t0
        ori     $s0, $s0, 0xffff
        sw      $s0, branch
branch: beq     $s0, $s1, BOGUS
```

```
        la      $t0, main
        addi    $t0, $t0, 0xffff
        jr      $t0
```

Executing the code on the left above does not cause an exception (regardless of whether or not the branch is taken) , while the code on the right does. Explain why?

The immediate value for the branch is in words, thus any immediate value will produce a word aligned address. The register value in the `jr` instruction is an address in bytes. The address is not word aligned and generates a bad address in text read error.

Problem 4 (20 points) You are designing a machine with 16-bit instructions, 16-bit words, a 16-bit address space, and an undetermined number of registers. Your design includes an `L`-type instruction used for loading an immediate value into a register and an `I`-type which allows an immediate value to be used with arithmetic and logical operations. Your design should allow a 16-bit value to be loaded into a register using no more than two instructions. In answering the following questions, make the best design compromises possible.

(a) (4 points) Draw the format for the `L`-type instruction. Be sure to label each field, any unused bits and their sizes.

Reasonable answers accepted. One reasonable possibility is:

op	rd	imm
4	4	8

(b) (4 points) Draw the format for the `I`-type instruction. Be sure to label each field, any unused bits and their sizes.

Reasonable answers accepted. One reasonable possibility is:

op	rs	rd	imm
4	4	4	4

(c) (4 points) How many registers will your design have?

Reasonable answers accepted. The design presented above has 16 registers.

(d) (4 points) Describe how your instruction set architecture will load a 16-bit value into a register.

Reasonable answers accepted. One possibility is to use 2 L-type instructions. The first loads the upper 8 bits into the top half of the register and the second loads the lower 8 bits into the bottom half..

(e) (4 points) The I-type instruction format is used for branch instructions. What is the range of the branch instruction?

Reasonable answers accepted. The design presented above allows branches from -2^3 to $+2^3-1$ words.

Problem 5 (18 points) A simple car rental calculation program is provided below. Fragments of the assembly language implementation of the program are also provided to you. You must complete the program by supplying the code required in the provided spaces. The program must strictly follow MIPS register conventions. Comments are provided to assist you in understanding the purpose and arguments of each procedure. The implementations of CalcSubtotal and CalcTax are not provided for the sake of brevity. You are not required to implement these two functions.

```
//High-level language implementation
//Main program that determines the total due.
int main( )
{
        int numDays, dailyRate, insuranceFee, totalDue;
        .
        .
        totalDue = CalcTotalDue(numDays, dailyRate, insuranceFee);
        .
        .
        .
 }

//Procedure to calculate the total due.
int CalcTotalDue( int numDays, int dailyRate, int insuranceFee )
{
        int subtotal, tax, total;

        subtotal = CalcSubtotal(numDays, dailyRate);
        tax =   CalcTax(subtotal);

        total = subtotal + tax + insuranceFee;

        return total;
}
```

Procedures `CalcSubtotal` and `CalcTax` will calculate the rental subtotal and taxes respectively.

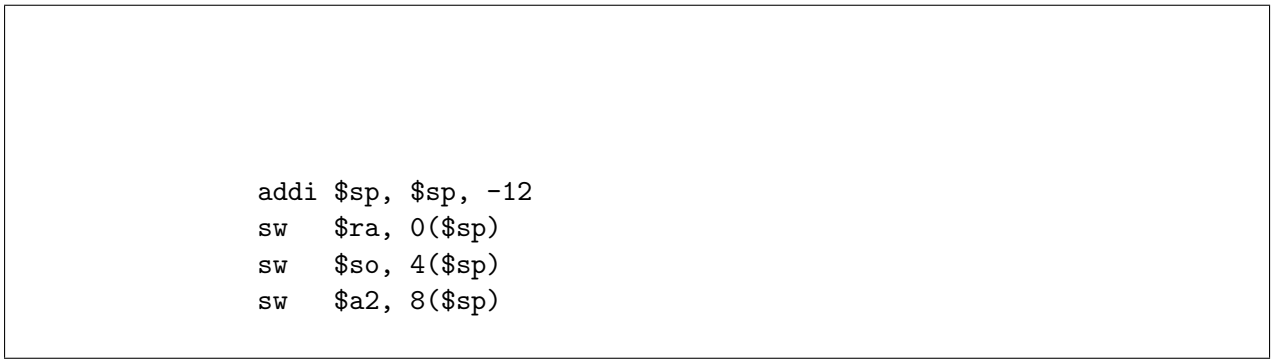
```
#
#       Assembly language implementation of the above program.
#
                .data  # data segment of the program

totalDue:       .word 0
numDays:        .word 3
dailyRate:      .word 13
insuranceFee:   .word 7

                .text  # code segment of the program

# Main program that calculates the total due. The inputs are numDays,
# dailyRate, and insuranceFee, all are available in memory. The result
# totalIncurred is stored back in memory.

main:
                la      $t0, numDays       # read the value of numDays
                lw      $t0, 0($t0)        # from memory
                la      $t1, dailyRate     # read the value of dailyRate
                lw      $t1, 0($t1)        # from memory
                la      $t2, insuranceFee  # read the value of insuranceFee
                lw      $t2, 0($t2)        # from memory
```

```
                move $a0, $t0
                move $a1, $t1
                move $a2, $t2

                #optional setup stack
```

```
                                           # Procedure calculates total
                                           # due. Takes three parameters -
                                           # numDays, dailyRate, and
                                           # insuranceFee.
                jal        CalcTotalDue
                                           # Procedure returns total due.
```

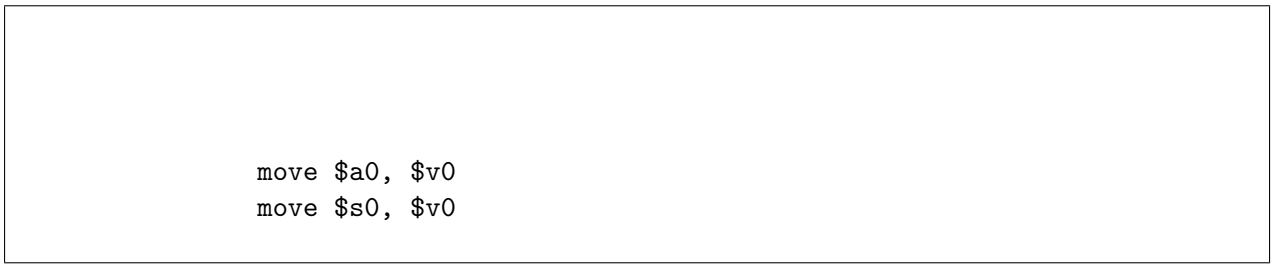
```
                move  $t0, $v0

                #optional undo stack
```

```
                la $t1, totalDue          # store the result in memory
                sw $t0, 0($t1)

                li $v0, 10                # Done - so exit
                syscall

# Procedure CalcTotalDue calculates total due.
# Takes three parameters - numDays, dailyRate, and insuranceFee.
# Returns the total due.
#
# Calls procedure CalcSubtotal to determine the subtotal.
# Saves the value of subtotal in register $s0.
#
# Calls procedure to CalcTax to determine the tax.
# Saves the value of tax in $t0.

CalcTotalDue:
```



```
                addi $sp, $sp, -12
                sw   $ra, 0($sp)
                sw   $so, 4($sp)
                sw   $a2, 8($sp)
```

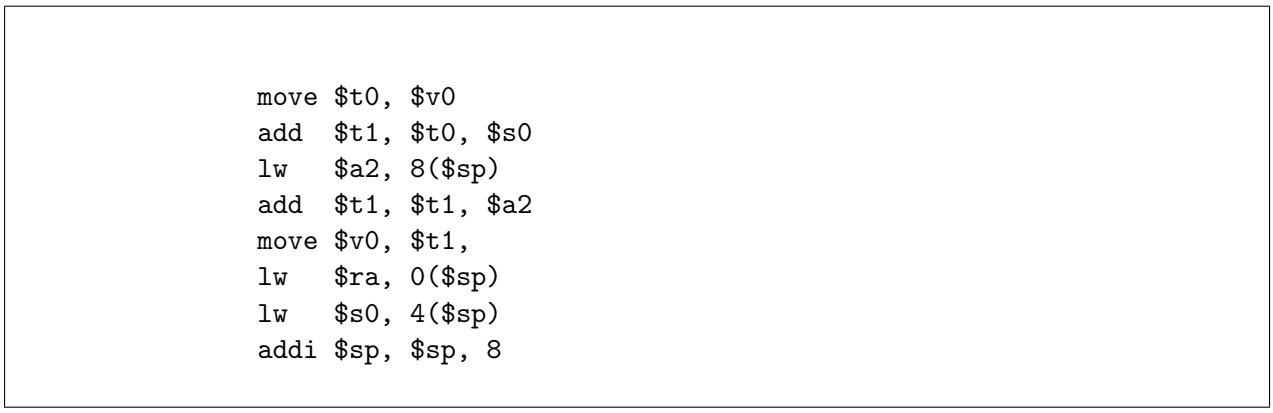
```
                                  # Procedure takes 2 parameters
                                  # i.e. numDays and dailyRate
                jal CalcSubtotal
                                  # subtotal value is stored in $s0.
```



```
                move $a0, $v0
                move $s0, $v0
```

```
                                  # Procedure takes 1 parameter i.e.
                                  # subtotal.
                jal CalcTax       # Procedure determines and
                                  # returns the amount of tax.
                                  # Store tax in $t0.
```

```
                                        # Calculate the return value of
                                        # totalDue using subtotal, tax, and
                                        # insurance fee.
```



```
                move $t0, $v0
                add  $t1, $t0, $s0
                lw   $a2, 8($sp)
                add  $t1, $t1, $a2
                move $v0, $t1,
                lw   $ra, 0($sp)
                lw   $s0, 4($sp)
                addi $sp, $sp, 8

                jr $ra                  # Return to main.

# The implementations of CalcSubtotal and CalcTax should follow.
# They are not provided for the sake of brevity.
```

Problem 6 (18 points) You are presented with two machines, Machine 1 and Machine 2 with clock rates of 2.33GHz and 2GHz, respectively. The instruction mixes, CPI, and clock rate are shown in the table below.

Machine 1	CPI	Freq	Inst	Cycles
Floating pnt ops	7	7	7	49
Integer ops	4	40	40	160
Load/Store	5	33	33	165
Branch	3	20	20	60
Total			100	434

Machine 2	CPI	Freq	Inst	Cycles
Floating pnt ops	8	5	5	40
Integer ops	5	55	55	275
Load/Store	5	17	17	85
Branch	3	23	23	69
Total			100	469

(a) (5 points) Calculate the average CPI for the Machine 1.

See calculation above. CPI = 4.34

(b) (5 points) Calculate the average CPI for the Machine 2.

See calculation above. CPI = 4.69

(c) (5 points) Which machine is faster? List any assumptions.

Without the number of instructions in each case the performance cannot be determined. If the number of instructions is assumed to be the same.

$$\begin{aligned}
\frac{\text{Perf}_{\text{Machine1}}}{\text{Perf}_{\text{Machine2}}} &= \frac{\text{Exec.Time}_{\text{Machine1}}}{\text{Exec.Time}_{\text{Machine2}}} \\
&= \frac{\text{CPI}_{\text{Machine1}} \times \#\ \text{INST}_{\text{Machine1}} \times 1/\text{clkfreq}_{\text{Machine1}}}{\text{CPI}_{\text{Machine2}} \times \#\ \text{INST}_{\text{Machine2}} \times 1/\text{clkfreq}_{\text{Machine2}}} \\
&= \frac{4.34 \times \#\ \text{INST}_{\text{Machine1}} \times 1/2.33\ \text{Ghz}}{4.69 \times \#\ \text{INST}_{\text{Machine1}} \times 1/2\ \text{Ghz}} \\
&= 0.794
\end{aligned}$$

Machine 1 is 20.6% faster.

(d) (3 points) Suppose the clock rate of the slower machine was increased to 3GHz and the CPI of its integer instructions were reduced by 25% would it still be slower? List any assumptions.

Machine 2	CPI	Freq	Inst	Cycles
Floating pnt ops	8	5	5	40
Integer ops	3.75	55	55	206.25
Load/Store	5	17	17	85
Branch	3	23	23	69
Total			100	400.25

New CPI for Machine 2 = 4.00

Without the number of instructions in each case the performance cannot be determined. If the number of instructions is assumed to be the same.

$$\begin{aligned}
\frac{\text{Perf}_{\text{Machine1}}}{\text{Perf}_{\text{Machine2}}} &= \frac{\text{Exec.Time}_{\text{Machine1}}}{\text{Exec.Time}_{\text{Machine2}}} \\
&= \frac{\text{CPI}_{\text{Machine1}} \times \#\ \text{INST}_{\text{Machine1}} \times 1/\text{clkfreq}_{\text{Machine1}}}{\text{CPI}_{\text{Machine2}} \times \#\ \text{INST}_{\text{Machine2}} \times 1/\text{clkfreq}_{\text{Machine2}}} \\
&= \frac{4.34 \times \#\ \text{INST}_{\text{Machine1}} \times 1/2.33\ \text{Ghz}}{4.00 \times \#\ \text{INST}_{\text{Machine1}} \times 1/3\ \text{Ghz}} \\
&= 1.397
\end{aligned}$$

Machine 2 is now 39.7% faster.