Rose-Hulman Institute of Technology
Computer Science and Software Engineering
Prof. Laurence D. Merkle

Exam 2

Name: _____ Section: MWR or TWF

Instructions:

- Write all answers on these pages. Use the back as necessary.
- Clearly indicate your final answer.
- For full credit, show your work, and document your code.
- This exam is long, so read the entire examination before starting and budget your time.

Authorized resources:

- Reference materials provided by the instructor at the time of the exam.
- Both sides of one 8½"x11" sheet of paper containing only handwritten material.

Unauthorized resources:

You are NOT permitted to use any resources other than those identified above. In particular, you may NOT use books, notes, electronic files, calculators, PDAs, or computers.

Problem 1a	/15 points
Problem 1b	/18 points
Problem 2a	/18 points
Problem 2b	/18 points
Problem 2c	/18 points
Problem 2d	/18 points
Problem 2e	/18 points
Problem 3a	/18 points
Problem 3b	/18 points
Problem 3c	/18 points
Total	/180 points

Problems

Problem 1 – In this problem, you will design the machine language for the hypothetical MULDER processor, which is based on a stack model of execution. As described by Patterson & Hennessy, this means that operands are pushed on the stack from memory or popped off the stack into memory. The `push` instruction and the `pop` instruction each have one operand, consisting of the address of a word in memory:

```
push a # $sp = $sp – 4; Mem[$sp] = Mem[a]
pop a  # Mem[a] = Mem[$sp]; $sp = $sp + 4
```

All other instructions take their operands from the stack and then place the result back onto the stack, so no operands need to be specified in the instruction. For example:

```
add    # Mem[$sp+4] = Mem[$sp] + Mem[$sp+4]; $sp = $sp + 4
```

The following facts may be useful: $2^6 = 64$, $2^7 = 128$, and $2^8 = 256$.

Problem 1a - [18 points] Assume that the MULDER processor has a 6-bit address space, that there are 67 different instructions including `push` and `pop`, and that all instructions are exactly 8 bits long. Propose machine language instruction formats for the MULDER instruction set. Clearly indicate which bits are assigned to which fields, and state which instructions would be assigned to each instruction type.

Problem 1b - [18 points] Assume that the MULDER processor has a 6-bit address space and that all instructions are exactly 8 bits long. Not including `push` and `pop`, how many instructions can be represented in the machine language? Explain your answer.

Problem 2 – In this problem you will design an implementation of the hypothetical SCULLY processor, which has only one instruction: subtract and branch if negative, or `sbn` for short. The `sbn` instruction has three operands, each consisting of the address of a word in memory:

```
sbn a, b, c # Mem[a] = Mem[a] – Mem[b]; if (Mem[a]<0) go to c
```

The instruction will subtract the number in memory location `b` from the number in location `a` and place the result back in `a`, overwriting the previous value. If the result is greater than or equal to 0, the computer will take its next instruction from the memory location just after the current instruction. If the result is less than 0, the next instruction is taken from memory location `c`. SCULLY has no programmer-visible registers and no instructions other than `sbn`.

Assume that SCULLY has

- An 8-bit address space and an 8-bit address bus
- 24-bit instructions, 24-bit numbers, and a 24-bit data bus
- Byte-addressable memory

Problem 2a - [18 points] Complete the RTL description for the `sbn` instruction. The instruction format and the RTL description of the first cycle are given, and comments are provided for each cycle.

Bits:	23-16	18-8	7-0
Field:	A	B	C

1. IR = Mem[PC] PC = PC + 3	Instruction Fetch Increment PC
2.	Fetch A
3.	Fetch B
4.	Subtract
5.	Store A Conditionally branch

Problem 2b - [18 points] List the components required to support your RTL description, along with their inputs, outputs, and control signals, as well as the widths of those signals. The components used in the first cycle have been completed for you. Add additional rows to the table if necessary.

Component	Inputs	Outputs	Control Signals
PC	PCDataIn (8)	PCDataOut (8)	PCWrite (1)
Mem	Address (8) WriteData (24)	MemData (24)	MemWrite (1)
IR	IRDataIn (24)	IRDataOut (24)	IRWrite (1)
ALU	ALUDataInA (24) ALUDataInB (24)	ALUDataOut (24)	ALUOp (1)

Problem 2c – [18 points] Draw a diagram of the datapath to implement the SCULLY architecture. Ensure that it is consistent with the RTL description that you produced for Problem 2a. Clearly indicate any necessary control signals (English descriptions are not necessary). Be as neat as reasonably possible.

Problem 2d – [18 points] Draw the state transition diagram specifying a finite state machine that will control the datapath you specified in Problem 2c in such a way that it supports the SCULLY architecture. Specify the values of all control signals in each state.

Problem 2e – [18 points] Assume that SCULLY uses the 2's-complement representation for numbers. Draw a 1-bit ALU to support the SCULLY architecture. You may use AND gates, OR gates, inverters, multiplexers, and 1-bit full adders.

Problem 3a – [18 points] Write a MIPS program to add the number in memory location `a` and the number in memory location `b`, leaving the result in location `a`, and leaving the number in memory location `b` unmodified. Do not use pseudoinstructions. You may modify the contents of registers `$at`, `$t0`, and `$t1`, but not the contents of any other registers.

Problem 3b - [18 points] Write a MULDER program to add `a` and `b`, leaving the result in `a`, and leaving `b` unmodified.

Problem 3c – [18 points] Write a SCULLY program to add `a` and `b`, leaving the result in `a`, and leaving `b` unmodified. Use the notation `.+1` to mean "the instruction after this one," and assume that `temp` is the address of a spare memory word that can be used for temporary results.