Computer Science 141
Assignment #6: MIPS and Memory

Due 1PM, November 4th. 2015

Name:

1 PROBLEM 1 (12 PTS):

ROMs can be used to implement combinational circuits. For each of the following circuits, specify the size of the ROM that would be needed to program them. Explain how you arrived at your answer, indicate whether using a ROM is a good design or choice, and justify your choice.

a. A 16-bit adder/subtractor with carry-in and carry-out.
b. An 8x8 multiplier
c. A 16-bit priority encoder

2 PROBLEM 2 (28 PTS):

2.1 COMPARE AND CONTRAST (12 PTS)

We have studied several different types of memory in this class. For the following types of memory, construct a table to compare and contrast them. Order them by relative size of a bitcell, and indicate which components in a computer system are likely to use them and why (e.g. area, power, performance, etc.). You may use resources on the Internet to help you with this, but cite your sources. Please keep your table concise – no need for overwhelming detail.

a. Flip-flop
b. SRAM
c. DRAM
d. Flash memory

2.2 PHASE CHANGE MEMORY (16 PTS)

Researchers are looking at novel memory technologies to overcome the shortcomings of DRAM. One of the most promising new technologies is called phase change memory (PCM). For this problem, you will perform some research on PCM and answer the following questions.

1. (2pts) In a few sentences, explain how PCM stores data and how the data is read to or written from a bitcell.
2. (2 pts) PCM is called a *nonvolatile* memory. Why? How does this differ from SRAM or DRAM?
3. (12 pts) Let's examine the tradeoffs of using PCM in place of DRAM.

	Read	Write	Standby
Power			
Performance			

For each box, fill in which memory technology of the two is better, where better means less power and more performance. If you believe the two technologies are comparable, then write "SAME".

Explain your answers below in a few sentences. Why is (PCM/DRAM) better for read/write/standby? If you answered "SAME", explain this choice.

3 PROBLEM 3 (60 PTS)

You will build an assembler that will translate a subset of the MIPS instruction set into machine code. For this problem, you may collaborate with your lab partner. Submit your solution to harvard.cs141.hw@gmail.com. Remember to follow the electronic submission guidelines.

The input to your assembler will be a `.asm` file. Each line is formatted as:

```
[label:] insn arg0 [arg1] [arg2]  [# comment]
```

Fields in brackets are optional – not every line has a label or comment, and not every instruction has three arguments. Register arguments may be provided either as `$a0`, `$s0`, `$t0` (following MIPS conventions) or `$r0-$r31` (where the register is directly named by number).

The instructions you must support are:

- `add`
- `addi`
- `sub`
- `and`
- `andi`
- `or`
- `ori`
- `xor`
- `xori`
- `nor`
- `sll`
- `sra`
- `srl`
- `slt`
- `slti`
- `beq`
- `bne`
- `j`
- `jal`
- `jr`
- `lw`
- `sw`
- `nop`

Opcodes for each instruction are provided in Appendix B of your textbook. The exception is the nop (no operation) instruction, which is encoded as 32 zeros.

The output of your assembler should be a `.machine` file, which contains machine code in hexadecimal, one instruction per line. **Do not print `0x` at the start of each line.**

Be careful when you are assembling branch and jump instructions, particularly when they contain labels. For simplicity, assume that a label will never exist on its own line. It only refers to the instruction on the same line. We assume that the first instruction of the program is located at address `0x00400000`.

We have provided a small testing assembly file and the correct machine code that you can use to verify your assembler. You are permitted to use any programming language that you prefer, as long as you can demo it and output a correct machine code file. We will use a different assembly file to test your program. We recommend writing your assembler in Python. If you choose to do so, a template of the assembler (in Python) has been provided. All materials should be included in this homework download.

For example, if the input code is:

```
loop:   add  $s0,   $s1,    $s2        # $s0 = $s1 + $s2
        lw   $t4,   0($s0)
        lw   $t3,   0($s1)
        bne  $t3,   $t4,    loop

        addi $s0,   $s0,    1           # $0++
```

The output machine code should be:

```
02328020
8e0c0000
8e2b0000
156cfffc
22100001
```