Computer Science Fundamentals II

Final Exam

Fall 2019

YOUR NAME:____

You have 3 hours to complete an exam like this, and my expectation is that it should take you only 2 hours to complete most of it. **There are seven problems on nine pages,** including this cover page.

Here is a summary of the MIPS32 assembly instructions that you can use in your programs:

`li $RD,` *value*	–loads an immediate value into a register.
`la $RD,` *label*	–loads the address of a label into a register.
`lw $RD,` *label*	–loads a word at an labelled address into a register.
`lw $RD, ($RS)`	–loads a register from memory at an address specified in a register.
`lw $RD,` *offset*`($RS)`	–loads from an address specified in a register, but at an offset.
`sw $RS, ($RD)`	–stores a register into memory at an address specified in a register.
`sw $RS,` *offset*`($RD)`	–stores to an address specified in a register, but at an offset.
`addu $RD, $RS1, $RS2,`	–add two registers, storing the sum in another.
`subu $RD, $RS1, $RS2,`	–subtract two registers, storing the difference in a third.
`addiu $RD, $RS,` *value*	–add a value to a register, storing the sum in another.
`move $RD, $RS`	–copy a register's value to another.
`negu $RD, $RS`	–copy the negated value of a register's value to another.
`b` *label*	–jump to a labelled line.
`blt $RS1, $RS2,` *label*	–jump to a labelled line if one register's value is less than another.
`bltz $RS,` *label*	–jump to a labelled line if a register's value is less than zero.
`gt, le, ge, eq, ne`	–other conditions than `lt`

Some of the registers you can access are named `$v0-v1`, `$a0-a3`, `$t0-t9`, `$s0-s7`, `$sp`, `$fp`, and `$ra`.

1. *Using only the MIPS32 instructions on the prior page*, **write a snippet of MIPS32 code** that takes two pieces of information:

   - an array of integers whose start address is stored in register `a0`
   - the number of integer items in that array, stored in register `a1`

   The code should perform a *right rotation* of the items in that array. This means that every item in the array should be shifted right excepting the last item, which moves to the front.

   For example, if the array has five items sequenced as 3, 1, 4, 7, 5 then, after your code has completed execution, it should hold them instead as the sequence 5, 3, 1, 4, 7.

2. In MIPS32, a linked list node can be represented by eight consecutive bytes in memory. The first four bytes would hold an integer data value, and the last four bytes could contain the address of some other linked list node. That second part could instead contain the address 0 to represent the null pointer, and thus represent a node at the end of a linked list.

   Using only the MIPS32 instructions on the prior page, **write a snippet of MIPS32 code** that takes two pieces of information:

   - the address of a linked list node `a0` that is the first item in a linked list
   - the address of another linked list node `a1` that is not yet in the linked list, whose second four bytes contain the null pointer 0

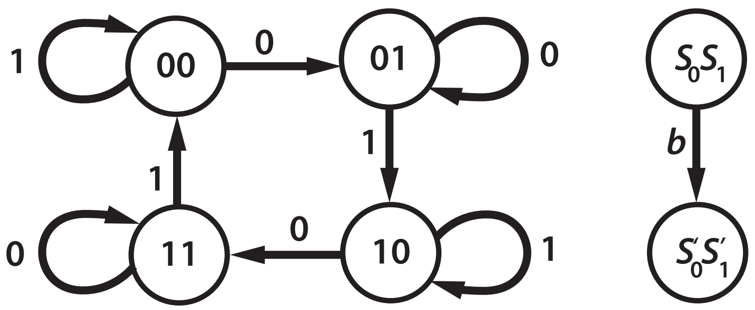
   The code should do the work of inserting the linked list node pointed at by `a1` **onto the end** of the linked list starting at `a0`. In other words, it should traverse the linked list starting at `a0` until it finds the last node in the list. It should then link that last node to the node referenced by `a1`.

   You can assume that both `a0` and `a1` point to two valid places in memory with 8 consecutive bytes as each described.

3. A bit sequence is *non-decreasing* if a `1` bit never occurs or is never followed by a `0`. For example, the bit sequences `000000`, `00011`, and `111` are all non-decreasing.

   Build a circuit that takes a 3-bits of input, say $b_0 : b_1 : b_2$, and outputs a `1` only if their bit sequence is non-decreasing. **Give the truth table** for the circuit's behavior, **give its** boolean expression, and then **draw** the circuit.

4. Consider building the next state logic for a finite state machine that behaves like the one depicted in the diagram below. Its input bit is named b, and its state bits are named s_1 and s_0. For example, when the state bits $s_1 : s_0$ are `01` and the machine processes a bit b of `1`, the next state bits $s_1' : s_0'$ are `10`.



Devise the truth table for the next state logic for this machine. When faced with the next input bit b when in state $s_1 : s_0$, the machine transitions to state $s_1' : s_0'$. **Give the boolean expression** for s_1' and s_0' in terms of b, s_1, and s_0.

5. Consider writing C++ code that uses a vector of integers to represent a large positive number as a sequence of its decimal digits. The integer at index 0 contains the ones digit, the integer at index 1 contains the tens digit, the integer at index 2 contains the hundreds digit, and so on. For example, the number 257 would be represented by a vector with items 7, 5, 2. You can assume that the most significant digit (the last item in the vector) is not zero.

   (a) **Write a C++ function** `increment` that takes such a vector representing an integer, and modifies it so that it equals the successor of that integer. For example, if the vector contains the 3-digit sequence 9, 2, 1, then `increment` should modify it to contain the sequence 0, 3, 1, because 130 is the successor of 129.

   (b) **Write a C++ function** `add` that takes two vectors of digits and gives back a third. That third vector should represent the sum of those two vectors' integers. For example, if the first contained 0, 3, 1 and the second contained 7, 5, then `add` would return the vector containing the sequence 7, 8, 1 because the sum of 130 and 57 is 187. The returned vector should not have a last item of 0 (the most significant digit of the sum should not be 0).

6. Consider the following struct definition for a linked list node

```
struct Node {
    int data;
    struct Node *next;
};
```

Write a C function with the following signature:

```
     Node *copy_list(Node *);
```

It takes a pointer to the first node of a linked list, possibly null, and returns a pointer to a linked list that is an exact copy of the given linked list. So, if the given linked list has n nodes, then the code should ultimately create n new linked nodes on the heap, and give back a pointer to the first node (or a null pointer if the given list was empty).

7. A `CountDown` is an object that tracks an integer state. It gets constructed by providing the constructor a maximum integer value, and that puts the `CountDown` into that state. The constructor takes a second piece of information, a message string. When its method `tick` is called, and its state value is not 0, its integer state value decreases by one and it outputs a line with that new integer value to `cout`. If it is instead at 0, `tick` simply outputs a line with the message string of the object. When its method `reset` is called, its integer state value gets reinitialized to that maximum integer value, but nothing gets printed.

   You can assume that the maximum value provided to the constructor is an integer greater than 1.

   (a) **Write the C++ code** for the `CountDown` class, both the specification and the implementation. You should write it assuming that it has classes that are derived from it, tagging the methods as `virtual`

   (b) **Give the C++ code** for a class `CountUpDown` that derives from `CountDown`, again by giving both its specification and its implementation. It should have an additional method `uptick` that increments the state value, but only if it is not at its maximum value, and does not output anything regardless.

(c) Now **give the C++ code** for a class `CountMod` that derives from `CountDown`. It should instead be initialized so that its state is one less than the maximum value given to the constructor. When its method `reset` is called its integer state value gets reinitialized to that same value, one minus the value provided by the constructor. Nothing gets printed. When its method `tick` is called, and its state value is not 0, its integer state value decreases by one and it outputs a line with that new integer value to `cout`. If it is instead at 0, `tick` simply outputs a line with the message string of the object, but then the state value is reset. Use as little C++ code for these methods, relying on inheritance.