Computer Science Fundamentals II

Practice Final Exam

Fall 2020

This document contains an example of the number, kind, and mix of problems that you should expect on this semester's final exam held on December 15, 1-5pm.

Several problems have you write MIPS assembly code. Here is a summary of relevant parts of its instruction set:

MIPS32 coding guide:

`li` $RD, *value*	–loads an immediate value into a register.
`lw` $RD, ($RS)	–loads a word from memory at an address specified in a register.
`sw` $RS, ($RD)	–stores register's word into memory at an address specified in a register.
`add` $RD, $RS1, $RS2,	–add two registers, storing the sum in another.
`sub` $RD, $RS1, $RS2,	–subtract two registers, storing the difference in a third.
`addi` $RD, $RS, *value*	–add a value to a register, storing the sum in another.
`andi` $RD, $RS, *value*	–compute the bitwise AND of a register with a value.
`sll` $RD, $RS, *positions*	–shift a register's bits left, storing the result in another.
`srl` $RD, $RS, *positions*	–shift a register's bits right, storing the result in another.
`sra` $RD, $RS, *positions*	–same, but preserves the sign bit.
`move` $RD, $RS	–copy a register's value to another.
`jal` *label*	–jump to a labelled line, saving the return address.
`jr` $RT	–jump to a line according to a register.
`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`

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

1. Let $r_{k-1} : \cdots : r_2 : r_1 : r_0$ be the bits of a k-bit register that stores the two's complement encoding of some integer. Assume that k is an even number.

   (a) Suppose I tell you that the first $k/2$ bits are `0`. What does that tell you about the integer value held by that register? **What's the range** of possible values it could hold? If it helps, tell me your answer assuming k is 8. This would mean that the leftmost 4 bits are `0000`.

   (b) Suppose I tell you that the first $k/2$ bits are `1`. What does that tell you about the integer value held by that register? **What's the range** of possible values it could hold? If it helps, tell me your answer assuming k is 8. This would mean that the leftmost 4 bits are `1111`.

   (c) Suppose $k = 8$ and suppose that the register holds a value in the range -4 to 3, inclusive. (That is, it's not smaller than -4 and it's not larger than 3.) **Give a boolean expression** on the bits r_i for the condition that the register's value is in that range. That expression should use only AND, OR, and NOT.

2. Consider the circuit for a "3-MUX". This circuit has three input "lines" ℓ_1, ℓ_2, and ℓ_3 along with a two "select" inputs s_1 and s_0. If $s_1 : s_0$ are `01`, then the circuit should output ℓ_1. If $s_1 : s_0$ are `10`, then the circuit should output ℓ_2. If $s_1 : s_0$ are `11`, then the circuit should output ℓ_3. If the select lines are both 0, it should just output a 0.

   Give the boolean expression for a 3-MUX circuit's output behavior. In devising that formula ask yourself the question "What are the conditions that make the circuit output a 1?" **Do not** build a 32-row truth table.

3. Consider building the next state logic for a finite state machine that processes a sequence of 1s and 0s. Its input bit is named b, and its state bits are named s_1 and s_0. State bit s_1 is set to 0 if the machine has seen a sequence with an even number of 1s. s_1 is set to 1 if the machine has seen a sequence with an odd number of 1s. State bit s_0 is set to 0 if the machine has seen a sequence with an even number of 0s. s_0 is set to 1 if the machine has seen a sequence with an odd number of 0s.

   Thus, initially, the machine is in the state `00` because it has seen no 1s and no 0s. And, after the machine has seen the sequence of bits `01101`, then the machine is in state `10` because it has seen three 1s (an odd number) and two 0s (an even number).. Processing `01101` it goes through the state transitions

   $$\texttt{00} \xrightarrow{0} \texttt{01} \xrightarrow{1} \texttt{11} \xrightarrow{1} \texttt{01} \xrightarrow{0} \texttt{00} \xrightarrow{1} \texttt{10}$$

   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.

4. **Write a snippet** of MIPS32 assembly that scans through an array of 32-bit integers that start at the memory address stored in register `a0`. It can assume that the length of the array, that is, the number of items in that array, is stored in register `a1`. By the end of the code, the register `v0` should be set to 1 if the last item in the array equals the sum of all the other items in the array. It should be set to 0 otherwise.

5. The *Collatz sequence* for $n > 0$ is defined as follows: when n is even, the next number in the sequence is $n/2$. When n is odd, the next number in the sequence is $3n + 1$. The sequence ends when the number is 1. The numbers 13, 40, 20, 10, 5, 16, 8, 4, 2, 1 demonstrate that the Collatz sequence for 13 is length 10. Here is Python code that computes the length of an integer's Collatz sequence:

```
def collatzLength(n):
    length = 1
    while n > 1:
        if n%2 == 0:
            n = n//2
        else:
            n = 3*n+1
        length = length + 1
    return length
v = int(input())
vlen = collatzLength(v)
print(vlen)
```

Write MIPS code that behaves the same as this Python program. This should include a section of MIPS code labelled as `collatzLength` that is a function that takes an integer argument and returns an integer result. Have `main` call that function with an input and print the result. Recall that the calling conventions have you pass the argument in register `a0` and return the result in `v0`. Since the call is a leaf call, and nothing from `main` needs to be preserved, you need not save anything onto the stack frame.

Recall that getting an integer input is system call 1 and printing an integer is system call 5. (You do not need to print a newline character after output of the Collatz sequence length.)

6. Below we invent a C++ class `Vehicle`. Vehicles drive around on a flat surface with a series of straight-line paths. They have a position that's specified by x and y coordinates, a size of gas tank (in gallons), a current amount of gas, and a fuel efficiency (in miles per gallon). Their constructor takes these four pieces of information. The **instance variables should not be accessible** to any clients of the class or to any methods of its derived subclasses.

   Vehicles support four methods:

   - `setPosition`: this puts a vehicle at a new location by setting its x and y coordinates to ones specified as arguments to the method. This method **should not be accessible** to clients but **should be accessible** to the methods of derived subclasses.
   - `distanceTo`: This computes and returns the Euclidean distance from the Vehicle's position to a position given as coordinates to the method. This method **should not be be accessible** to any client code or to methods of derived subclasses. Recall that the distance between two positions (x_1, y_1) and (x_2, y_2) is given by the formula:
     $$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$
     You can call the C++ function `std::sqrt` to compute the square root here.
   - `gasRemaining`: Returns the amount of gas remaining in the fuel tank. This **should be accessible** to clients.
   - `driveTo`: this places a vehicle at a new location by setting its x and y coordinates to ones specified to the method. But it should only do that if the amount of gas in the gas tank allows it to make that trip. It should return `true` if it makes the trip and `false` if it does not. This **should be accessible** to clients and methods of derived subclasses.

   (a) **Give the class definition** of `Vehicle`. In that definition, mark any instance variables as `const` and mark any methods as `const` appropriately. You need not mark any method arguments as `const`.

   (b) **Give the implementation of each method**, separately from the class definition. They should employ other methods when appropriate.

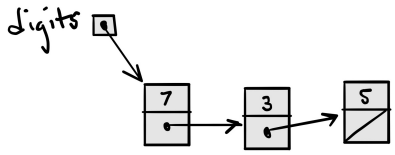
7. Consider the following C++ definition of a struct that serves as a linked list node for a sequence of digits:

```
struct DigitNode {
  int digit;
  struct DigitNode* next;
};
```

Consider the following class specification:

```
class DigitSequence {
private:
  DigitNode* digits;
public:
  DigitSequence(void);
  void increment(void);
  void decrement(void);
  ~DigitSequence(void);
};
```

It is meant to represent a sequence of digits of a non-negative integer as a linked list. The first node referenced by the pointer `digits` contains the least significant digit. The last node contains the most significant digit. The number 537 would be represented as a linked list with three nodes, the first containing 7, the second containing 3, and the last containing 5. Here is a picture of the representation of 537:



Note that the single-digit integers 0 through 9 have just one linked list node with that single digit.

Write each of the four methods according to the specs below.

- The first method `DigitSequence` is the constructor. It should build a linked list containing the single digit 0.
- The second method `increment` is a method that adds one to the number coded by the digit sequence. If that number happens to contain a sequence of 9 digits, for example 9999, then `increment` will have to change all the 9s to 0s and then allocate and link a new node containing the digit 1 to the end of the linked list.
- The third method `decrement` subtracts one from the number coded by the digit sequence, if the number is greater than 0. If it is 0, nothing is changed. If the digit sequence encodes a positive power of 10, for example 10000, then `decrement` will have to change all the 0s to 9s and remove the node at the end containing the digit 1. It should `delete` that node.
- The fourth method `~DigitSequence` is the destructor. It should give all the nodes that it has allocated back to the heap using `delete`.

8. For the code below, **give the output of the program** when it runs. The code may have bugs that make it crash. When that happens, indicate what caused the crash.

```
#include <iostream>
#include <string>

class A {
public:
    int num;
    std::string name;
    void print() { std::cout << name << ":" << num << std::endl; }
};

void func1(A* px) {
    px->num = 3;
}

void func2(A x) {
    x.num = 21;
    x.print();
}

int main(void) {
    A* pa  =  new A{ 108, "Nick" };
    A* pa2 =  pa;
    A  a   = *pa;
    pa->print();
    pa2->print();
    a.print();

    pa2->name = "Erika";
    pa->print();
    pa2->print();
    a.print();

    func1(pa);
    pa->print();
    pa2->print();
    a.print();

    func2(*pa);
    pa->print();
    pa2->print();
    a.print();

    delete pa;
    delete pa2;
}
```

9. For this problem, you will write several C++ functions named `sort3` that each have access to three values, rearranging them to be the minimum, the median, and maximum. For example of their use, suppose we have the start of a body of C++ code with these three declarations:

```
int i = 13;
int j = 5;
int k = 8;
```

If after these lines `sort3` is called appropriately in a single line of code, the resulting values of `i`, `j`, and `k` should be 5, 8, and 13, respectively.

(a) **Write a first version** of `sort3` that uses the C++ feature of passing arguments by reference to do its work. **Give the code** for calling this `sort3` to modify `i`, `j`, and `k`.

(b) **Write a second version** of `sort3` that expects the addresses of the variables `i`, `j`, and `k` and then does the appropriate work on the contents of memory referenced by these three pointers to sort them in order. **Give the code** for calling this `sort3` to modify `i`, `j`, and `k`.

(c) For a third version of `sort3`, we invent a struct type called `Triple`. **Define** the function `sort3` that takes a `Triple` and rearranges the three values. **Give the call** to `sort3`, passing it an appropriately `Triple` to do that work on `i`, `j`, and `k`. When it returns they should be 5, 8, and 13, respectively. **Give the definition** of the struct `Triple` required to make this all work.