Computer Science Fundamentals II

Soultions to the Practice Final Exam

Fall 2020

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) The range of values this register could hold is from 0 up to $2^{k/2} - 1$. For $k = 8$ it's from 0 to 15.

   (b) The range of values are from $-2^{k/2}$ to -1. For $k = 8$ this is -16 to -1.

   (c) This means that the leftmost 6 bits are all either all `1`s or all `0`s. The expression for the condition of the bits is then

   $$r_7 \cdot r_6 \cdot r_5 \cdot r_4 \cdot r_3 \cdot r_2 + \overline{r}_7 \cdot \overline{r}_6 \cdot \overline{r}_5 \cdot \overline{r}_4 \cdot \overline{r}_3 \cdot \overline{r}_2.$$

2. The 3-MUX logic is expressed as

   $$\overline{s}_1 \cdot s_0 \cdot \ell_1 + s_1 \cdot \overline{s}_0 \cdot \ell_2 + s_1 \cdot s_0 \cdot \ell_3.$$

3. The truth table for the behavior of this machine is

b	s_1	s_0	s'_1	s'_0
0	0	0	0	1
0	0	1	0	0
0	1	0	1	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	1
1	1	0	0	0
1	1	1	0	1

   The truth table above gives us this logic for each "next state" bit:

   $$\begin{aligned} s'_1 &:= b \cdot \overline{s}_1 + \overline{b} \cdot s_1 \\ s'_0 &:= \overline{b} \cdot \overline{s}_0 + b \cdot s_0 \end{aligned}$$

4. Here is that snippet of MIPS32 code:

```
begin_snippet:
        li   $t0, 0
loop:
        lw   $t1,($a0)      # Load the next value in the array.
        addi $a0,$a0,4      # Bump the pointer within the array.
        addi $a1,$a1,-1     # Count down how many items remain to be read.
        beqz $a1,compare    # If we've read the last item... (see * below)
        add  $t0,$t0,$t1    # If not the last, update the sum.
        b    loop
compare:
        beq  $t0,$t1,same   # (*) ...see if the last item matches the sum.
different:
        li   $v0,0          # Set v0 to 0 if they differ.
        b    end_snippet
same:
        li   $v0,1          # Set v0 to 1 if they match.
end_snippet:
```

5. The following MIPS32 program behaves like that Python program:

```
        .text
        .globl main
main:
        sw   $ra,-4($sp)
        li   $v0,5          # Get an input value.
        syscall
        move $a0,$v0        #
        jal  collatzLength  # Compute length of its Collatz sequence.
        move $a0,$v0        #
        li   $v0,1
        syscall
        li   $v0,0          # Print the length.
        lw   $ra,-4($sp)
        jr   $ra

collatzLength:
        li   $v0, 0
        li   $t1, 1
loop:
        addi $v0,$v0,1      # Increment the count.
        beq  $a0,$t1,done   # See if we've hit 1 yet.
        .
        andi $t0,$a0,1      #
        beqz $t0,div2       # If we haven't, check a0 even-/odd-ness.

times3p1:                   # => ODD CASE:
        sll  $t2,$a0,1      # Multiply a0 by 3 using:
        add  $a0,$a0,$t2    #    a0 := 2*a0 + a0
        addi $a0,$a0,1      # Then add 1.
        b    loop
div2:                       # => EVEN CASE:
        sra  $a0,$a0,1      # Divide a0 by 2.
        b    loop
done:
        jr   $ra
```

6. Here is the class definition followed by the method definitions:

```
class Vehicle {
private:
  double positionX;
  double positionY;
  double tankSize;
  double mpg;
  double fuel;
  double distanceTo(double x, double y) const;
protected:
  void setPosition(double x, double y);
public:
  Vehicle(double x, double y, double tank, double eff);
  double gasRemaining(void) const;
  bool driveTo(double x, double y);
};

Vehicle::Vehicle(double x, double y, double tank, double eff) :
  positionX {x}, positionY {y},
  tankSize {tank}, mpg {eff}, fuel {tank} { }

void Vehicle::setPosition(double x, double y) {
  positionX = x;
  positionY = y;
}

double Vehicle::gasRemaining(void) const {
  return fuel;
}

double Vehicle::distanceTo(double x, double y) const {
  double dx = x - positionX;
  double dy = y - positionY;
  return std::sqrt(dx*dx + dy*dy);
}

bool Vehicle::driveTo(double x, double y) {
  double range = mpg * gasRemaining();
  double distance = distanceTo(x,y);
  if (range >= distance) {
    fuel -= distance/mpg;
    setPosition(x,y);
    return true;
  } else{
    return false;
  }
}
```

7. Here is the code for the `DigitSequence` methods:

```
DigitSequence::DigitSequence(void) :
  digits {new DigitNode {0,nullptr}} { }

void DigitSequence::increment(void) {
  DigitNode* current = digits;
  while (current->digit == 9) {   // Flip all the 9s to 0s, maybe
    current->digit = 0;           // adding a new digit.
    if (current->next == nullptr) {
      current->next = new DigitNode {0,nullptr};
    }
    current = current->next;
  }
  current->digit++;               // Increment the first non-9 digit.
}

void DigitSequence::decrement(void) {
  if (digits->digit == 0 && digits->next == nullptr) {
    return;                       // Return with no change if 0.
  }
  DigitNode* previous = nullptr;
  DigitNode* current = digits;
  while (current->digit == 0) {   // Flip all the 0s to 9s.
    current->digit = 9;
    previous = current;
    current = current->next;
  }
  current->digit--;
  if (current->digit == 0         // If we just created a 0...
      && current->next == nullptr // ...and there aren't other digits
      && previous != nullptr) {   // ...and we're not a single digit
    previous->next = nullptr;     // erase digit if 100..0 => 099..9.
    delete current;
  }
}

DigitSequence::~DigitSequence(void) {
  while (digits != nullptr) {
    DigitNode* toDelete = digits;
    digits = digits->next;
    delete toDelete;
  }
}
```

8. The program outputs the lines below and then crashes with the deletion of `pa2` because it has the same pointer as the just deleted `pa`.

```
Nick:108
Nick:108
Nick:108
Erika:108
Erika:108
Nick:108
Erika:3
Erika:3
Nick:108
Erika:21
Erika:3
Erika:3
Nick:108
```

9. Each answer below is a pair consisting of the `sort3` and its call. They all use the following function:

```
void reorder(int& x, int& y) {
  if (x > y) {
    int tmp = y;
    y = x;
    x = tmp;
  }
}
```

(a) Function:

```
void sort3(int& x, int& y, int &z) {
  reorder(y,z);
  reorder(x,y);
  reorder(y,z);
}
```

Call:

```
sort3(i,j,k);
```

(b) Function:

```
void sort3(int* px, int* py, int* pz) {
  reorder(*py,*pz);
  reorder(*px,*py);
  reorder(*py,*pz);
}
```

Call:

```
sort3(&i,&j,&k);
```

(c) Function:

```
void sort3(Triple t) {
  reorder(*(t.s),*(t.t));
  reorder(*(t.f),*(t.s));
  reorder(*(t.s),*(t.t));
}
```

Call:

```
sort3(Triple {&i,&j,&k});
```

This last answer relies on the struct definition:

```
struct Triple {
  int* f;
  int* s;
  int* t;
};
```