UNCA CSCI 255
Final Exam Fall 2016
15 November, 2016 – 3:00 PM to 5:30 PM

This is a closed book and closed notes exam. Communication with anyone other than the instructor is not allowed during the exam. Furthermore, calculators, cell phones, and any other electronic or communication devices may not be used during this exam. Anyone needing a break during the exam must leave their exam with the instructor. Cell phones or computers may not be used during breaks.

If you want partial credit for imperfect answers, explain the reason for your answer!

Name:_____

Problem 1 (3 points) Range
What are the smallest **and** largest integers that can be represented as 7-bit unsigned numbers?

What are the smallest **and** largest integers that can be represented as 7-bit signed two's complement numbers?

Problem 2 (5 points) C expressions
In the left column, there are some tricky, and some not-so tricky, C expressions. (Except for the first two, these are also Java expressions.) Write their values in the right column. Express your answers in base 10. Assume two's complement representation.

`! (15 > 2)`	
`6 || (143 * 66 / 33)`	
`25 << 3`	
`25 >> 3`	
`~25`	
`25 + 15`	
`25 | 15`	
`25 ^ 15`	
`3 * 2 / 4`	
`3 + 2 / 4`	

Problem 3 (4 points) Two's complement to decimal conversion
Convert the following four five-bit *two's complement* numbers into signed decimal representation.

10000	**01001**
10101	**01010**

Problem 4 (4 points) Decimal to two's complement conversion
Convert the following four signed decimal numbers into five-bit *two's complement* representation. Some of these numbers may be outside the range of representation for five-bit two's complement numbers. Write "out-of-range" for those cases.

16	**5**
-5	**-16**

Problem 5 (3 points) Binary arithmetic
Perform the following operations and express the results as they should be for CSCI 255.

64k * 32
128M / 16
$\log_2$(32k)

Problem 6 (4 points) Adding signed numbers
Add the following pairs of five-bit *two's complement* numbers **and indicate which additions result in an overflow by writing one of "overflow" or "no overflow" in each box**. You must write either "overflow" or "no overflow" in each box in addition to the result of the addition.

01011 + 01111	10111 + 11101
10101 + 10101	11100 + 01000

Problem 7 (3 points) Fixed point decoding
In the left column, are five-bit twos-complement ***fixed point*** numbers with two fractional bits (and consequently three integer bits). Express those numbers in standard decimal point notation in the right column.

01011	
01111	
10001	

Problem 8 (3 points) Fixed point encoding
In the left column are decimal numbers. Express these numbers as five-bit twos-complement fixed point numbers with two fractional bits (and consequently three integer bits) in the right column. *You may not be able to get an exact representation for all of them, but get as close as you can.*

2.71828	
-2.75	

Problem 9 (4 points) Memory cache
Suppose a computer has 16M bytes of memory and 64k bytes of cache structured as an 8-way set associative cache with blocks (lines) of 32 bytes.

After figuring out the address size for the computer, draw a box for the typical address and divide it into tag, index, and offset fields. I have left you room to show your arithmetic. Use it!

Problem 10 (3 points)
Describe how the following have been used in the CSCI 255 labs. Your three best descriptions will be graded. The other two will be ignored. Keep it short! Use a phrase, not a sentence.

a) current limiting resistor

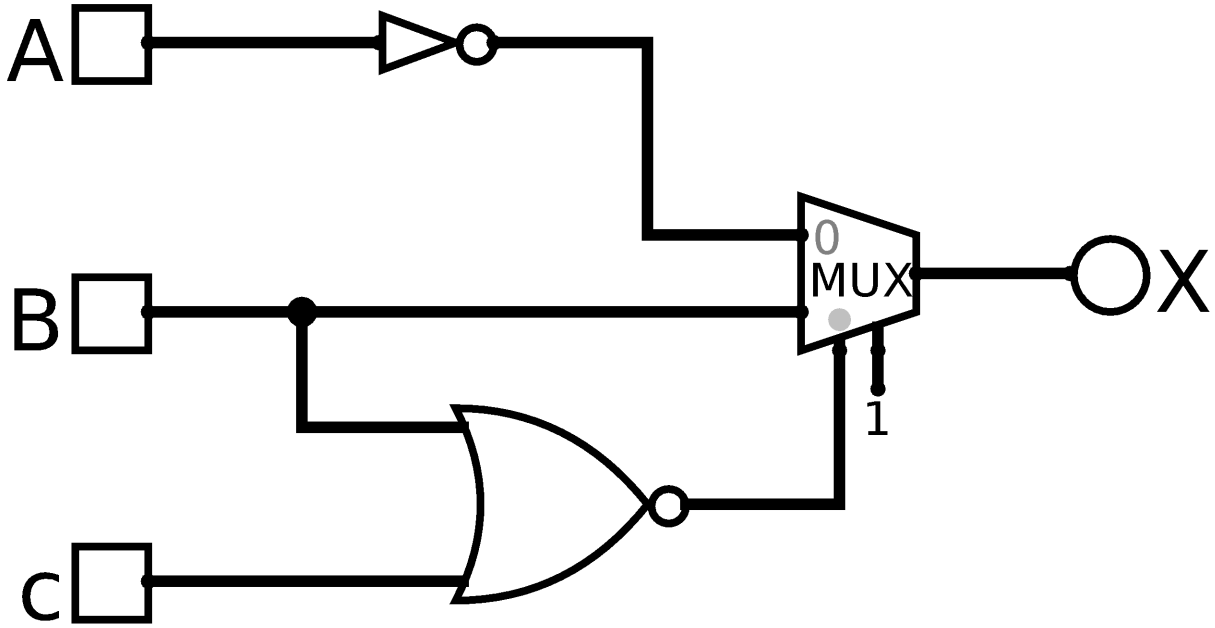
b) **`DigitalWrite()`**

c) I^2C (inter-integrated circuit) bus

d) **`PORTB`** – a special function register

e) PWM – pulse wave modulation

Problem 11 (7 points) Circuit to Boolean table and expression
Shown below is a digital circuit with inputs **A**, **B** and **C** and a single output **X**. The box labeled **MUX** is a multiplexer with two data inputs and one select input.



First, complete the truth table for the circuit.

A	B	C	X
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Second, write a boolean equation for the circuit.

Problem 12 (7 points) Boolean expression to truth table and circuit

First, fill in the truth table on the right below so that it corresponds to the following Boolean equation

$$X = A\,B + \overline{B + C}$$

If you prefer that your negations be primes, you can think of the equation as

$$X = A\,B + (B + C)'$$

Or, if you really like Java and C expressions, you can go with

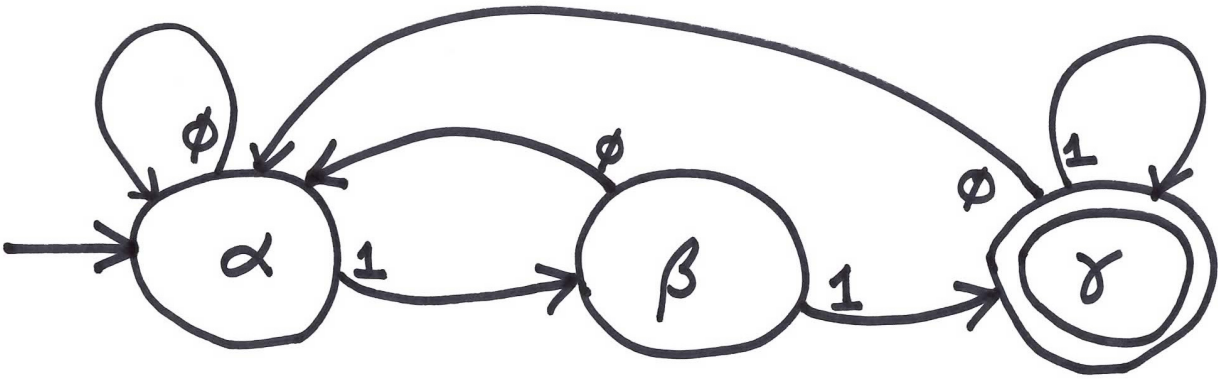
```
X = A && B || !(B || C)
```

A	B	C	X
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Second, draw a logic circuit to implement the boolean equation and truth table.

Problem 13 (8 points) Finite State Machines
This question is a bit like Problem 4, but without the `logisim` implementation.

Start with the FSM specified with the following fine diagram.



Answer the following questions. (If you have no idea, make a guess, so you'll have something to follow for the rest of this problem.)

Subquestion 13A: What is the starting state for this FSM?

Subquestion 13B: What are the accepting state(s) for this FSM?

Subquestion 13C: Fill in the underscored blanks below to illustrate the state transitions and the outputs of the FSM as it reads the input string **010110**.

inputs	**0**	**1**	**0**	**1**	**1**	**0**
states	___	___	___	___	___	___
outputs	___	___	___	___	___	___

Subquestion 13D: In the table below, specify an output function for the logical states of the finite machine.

state	output
α	
β	
γ	

Subquestion 13E: In the table below, specify a next-state function for the logical states of the finite machine.

present state	input	next state
α	0	
α	1	
β	0	
β	1	
γ	0	
γ	1	

Subquestion 13F: Complete the following table to specify an encoding of your logical states using two state bits called **X** and **Y**.

state	encoding	
	X	Y
α		
β		
γ		

Subquestion 13G: Draw a table that represents the FSM output function you specified in Subquestion 13D using the binary encoding you specified in Subquestion 13F. You can start with the following empty table.

X	Y	output

Subquestion 13H: Write a Boolean equation for the table you completed for Subquestion 13G. Don't worry about don't cares.

Problem 14 (6 points) Assembly to Machine
Below is a section of five MIPS32 instructions written in MIP32 assembly language. This isn't a useful program, but with tweaking the code might be able to count the number of elements in a linked-list.

```
              beq    $zero,$zero,testOut
              addi   $v0,$zero,-1
tryAgain:     lw     $a0,12($a0)
testOut:      bne    $a0,$zero,tryAgain:
              addi   $v0,$v0,1
```

These five MIPS32 instructions are inserted in the following table. Write the 32 bits needed to encode each of these instructions, at the binary level, in the MIPS32 instruction set architecture.

Do the opcode and function fields first!

`beq $zero,$zero,testOut`
`addi $v0,$zero,-1`
`tryAgain: lw $a0,12($a0)`
`testOut: bne $a0,$zero,tryAgain:`
`addi $v0,$v0,1`

Problem 15 (6 points) Disassembly required
This is similar to the instruction decoding exercise of Homework 8. For each of the seven 32-bit words shown below, translate the 32-bit word into the appropriate MIPS32 instruction. Use labels for instructions that are targets of branches. **Start by drawing vertical lines!**

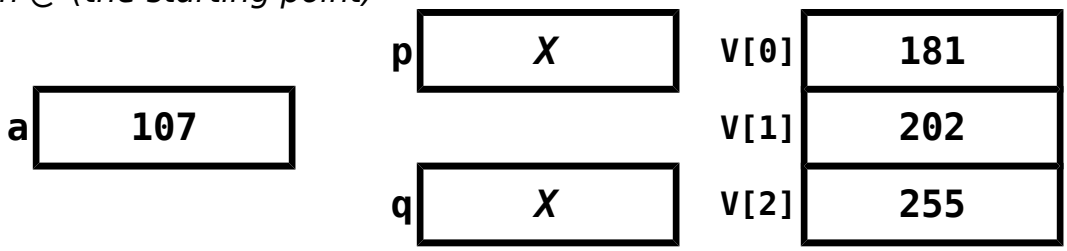
00100000000011100000000000100000
00000000100010000101000000100000
10000001010010100000000000000000
00010001010011100000000000000001
00100001000010000000000000000001
00100000100001000000000000000001
00010101000001001111111111111010

Problem 16 (8 points)
In this question, you are to fill in boxes representing the following C integer or pointer variables to show their values after each of seven sections of C code are executed. **You should consider all the sections as being independently executed after the following declaration and initialization statements**:

```
int    a = 107 ;
int    V[3] = {181, 202, 255} ;
int    *p = NULL ;
int    *q = NULL ;
```

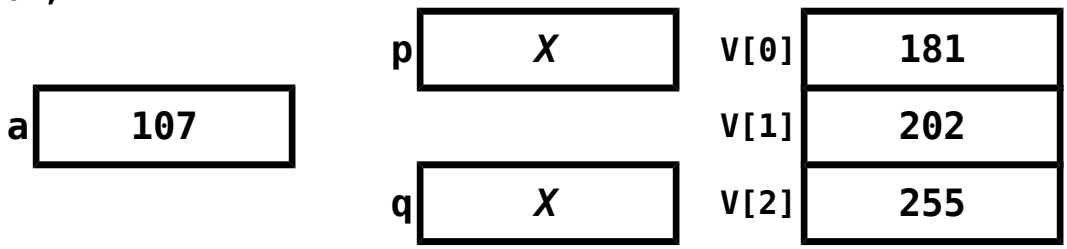
As you might guess, **`null`** in Java is similar to **`NULL`** in C. Draw the value **`NULL`** with a little ***X***. Don't ever just leave the pointer variable boxes empty.

Code section @ (the starting point)



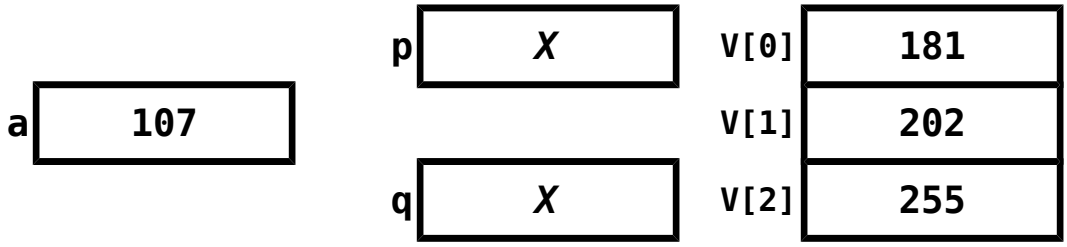
Code section A

```
p = &V[1] ;
q = &V[2] ;
*p = 200 ;
*q = 300 ;
```



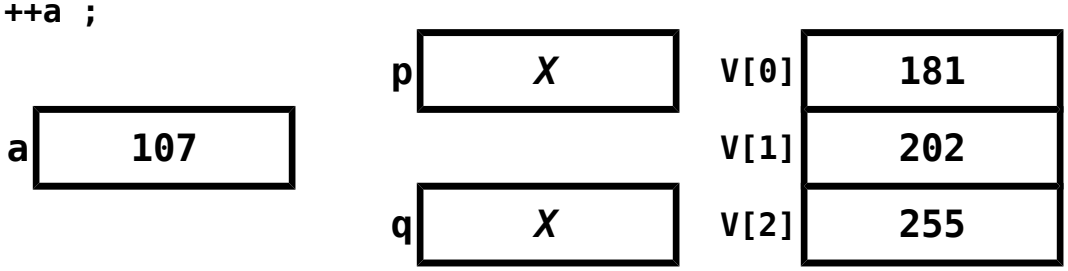
Code section B

```
p = V ;
q = p + 1 ;
*p = *q ;
```



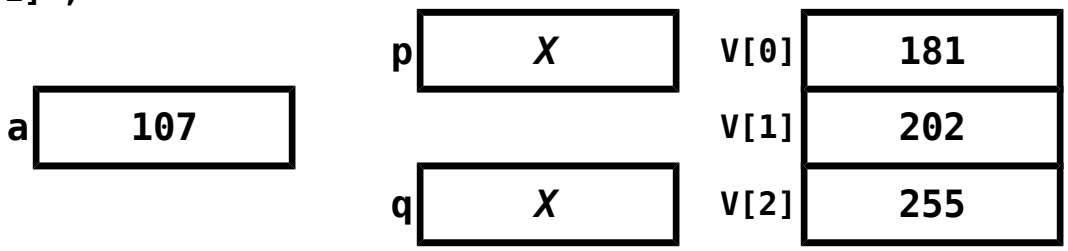
Code section C

```
V[0] = a++ ;
V[1] = a++ ;
V[2] = ++a ;
```



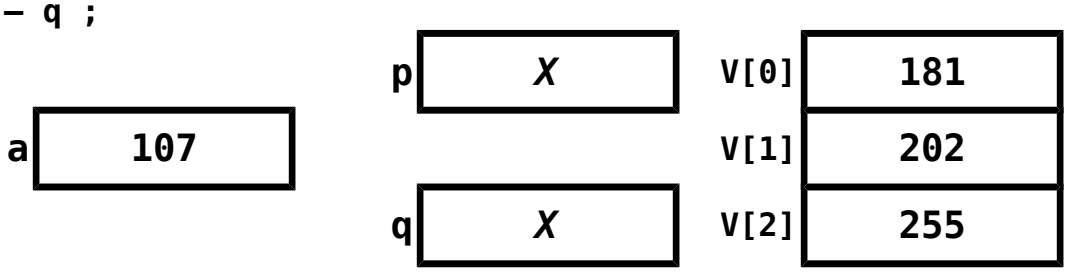
Code section D

```
p = &a ;
q = &V[0] ;
*p = q[1] ;
```



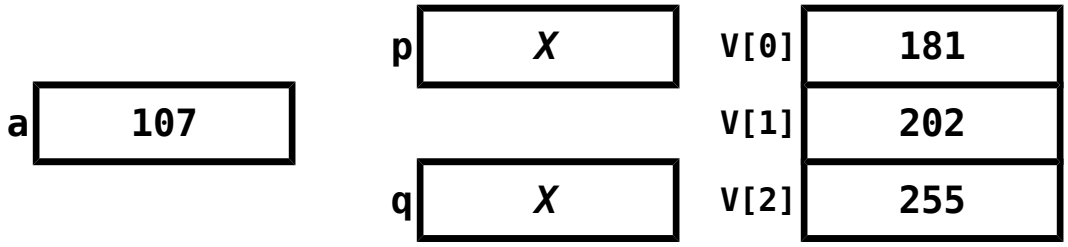
Code section E

```
p = &V[2] ;
q = &V[0] ;
*p = *p – *q ;
*q = p – q ;
```



Code section F

```
p = &V[0] ;
*p++ = 320 ;        // same as *(p++) = 320 ;
a = (*p)++ ;
```



The last two problems are based on the following C function.

```
int positiveSum(int V[], int size) {
  int sum = 0 ;
  for (int i=0; i<size; ++i) {
    if (V[i] >= 0) {
      sum = sum + V[i] ;
    } else {
      V[i] = 0 ;
    }
  }
  return sum ;
}
```

You will write your answers on the next three pages where you will have lots of space.

You should include appropriate comments in your code.

Problem 17 (10 points)
If the style of the sixth homework, implement the `postiveSum` function as a C function that only uses two control structures:

goto *label* ;
if (*expression*) goto *label* ;

Do not use the `?:` operator of C (and Java) to simulate an `if-then-else`.

This specifically means that you can't use the `for`, `while`, `switch`, `break`, `continue`, or even the statement block delimiters `{` and `}`. You can use the `if`, but only when the conditional expression is immediately followed by a `goto` statement.

Problem 18 (12 points)
Implement the `positiveSum` function as a MIPS32 assembly language routine.

Keep in mind the following aspects of the MIPS32 calling convention:

- The first argument, `V`, will be passed in register `$a0`.
- The second argument, `size`, will be passed in register `$a1`.
- The return value must be returned in register `$v0`.
- Registers `$t0` to `$t9` may be used as temporary registers.
- Pay attention to delay slots!

Problem 17, C with goto, answer goes here

Your answer starts here. Some code has been provided for you.

```
int positiveSum(int V[], int size) {

  int sum = 0 ;
```

```
  return sum ;
}
```

Problem 18, MIPS32, answer starts on this page

```
int positiveSum(int V[], int size) {
  int sum = 0 ;
  for (int i=0; i<size; ++i) {
    if (V[i] >= 0) {
      sum = sum + V[i] ;
    } else {
      V[i] = 0 ;
    }
  }
  return sum ;
}
```

Your answer starts here. Some code has been provided for you.

```
          .global  positiveSum
          .ent     positiveSum
          .text
positiveSum:
#  Argument V:     $a0
#  Argument size:  $a1
#  Return value:   $v0
```

Problem 18, MIPS32, answer ends on this page

```
int positiveSum(int V[], int size) {
  int sum = 0 ;
  for (int i=0; i<size; ++i) {
    if (V[i] >= 0) {
      sum = sum + V[i] ;
    } else {
      V[i] = 0 ;
    }
  }
  return sum ;
}
```

Your answer continues here. .

```
#  Argument V:     $a0
#  Argument size:  $a1
#  Return value:   $v0
```

```
        jr      $ra
        nop
        .end    positiveSum
```