CS 211: Midterm 2: 100 points

Instructor: Prof. Santosh Nagarakatte

Full Name Here:
RUID:

Question	Max Points	Points
1	20	
2	20	
3	25	
4	20	
5	15	

Problem 1: (10+10 points)

Answer "True" or "False" to these questions. If the answer is "False", you need to provide a reason to state why the answer is "False" (you will not get points otherwise).

1. The C switch instruction is converted to a series of jumps in all cases. True or False?

   Answer: False. Switch is generally implemented using jump tables.

2. $A.B + \overline{B}$ is the minimal boolean expression that cannot be simplified to a simpler circuit. True or False?

   Answer: False. $A + \overline{B}$ is the minimal boolean expression for the same.

3. A single D-latch can store 1-bit of information even though it has two outputs. True or False?

   Answer: True

4. 1111 is the largest negative number in signed 4-bit one's complement representation. True or False?

   Answer: True. -0 is the largest negative number.

5. If register *%eax* holds a pointer to an integer, then executing the cmd *p/x $eax* in gdb prints out the address of the integer. True or False?

   Answer: True.

C Programming (10 points)

1. (10 points) Write a C program to reverse a singly linked list? If you write Java code (instead of C), you will not get any points.

   Use the prototype below:

```
struct node{
  int field;
  struct node* link;
};
// complete this function, list points to the head of the linked list
struct node* reverse(struct node* list){
// Fill your code here

 struct node* reversed_list = NULL;
 struct node* temp;
 while(list != NULL){
  temp = list->next;
  list->next = reversed_list;
  reversed_list = list;
  list = temp;
 }
 return reversed_list;
}
```

Problem 2: Assembly Programming 1 (20 Points)

1. (15 points) Write the equivalent C code for the following assembly snippet. **Hint: all functions in this code take one integer input argument and output one integer result**. Show your work next to the assembly statements. No work. No points.

```
.globl bar
        .type   bar, @function
bar:
        pushl   %ebp
        movl    %esp, %ebp
        movl    8(%ebp), %eax
        addl    $10, %eax
        popl    %ebp
        ret

.globl foo
        .type   foo, @function
foo:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $24, %esp
        movl    %ebx, -8(%ebp)
        movl    %esi, -4(%ebp)
        movl    8(%ebp), %ebx
        movl    $1, %eax
        cmpl    $1, %ebx
        jle     .L5
        movl    %ebx, (%esp)
        call    bar
        movl    %eax, %esi
        subl    $1, %ebx
        movl    %ebx, (%esp)
        call    foo
        imull   %esi, %eax
.L5:
        movl    -8(%ebp), %ebx
        movl    -4(%ebp), %esi
        movl    %ebp, %esp
        popl    %ebp
        ret
```

Answer:

```
int bar(int i){
   return i+10;
}

int foo(int i){

  if ( i <= 1)
     return 1;

 int temp = bar(i);
 int temp2 = foo(i-1);

 return temp * temp2;

}
```

2. (5 points) Write the simplified version of the C code to accomplish the same function?

```
int foo(int i){
  int result = 1;
  while (i > 1){
    result = result * (i+10);
    i = i -1;
  }
  return result;
}
```

Problem 3: Diffusing the Assembly Bomb (25 points)

The bomblab designer was upset that a significant portion of the students in his class openly collaborated (and likely cheated) instead of learning the expected skills. To identify the students who learned the expected skills, the bomblab designer has designed the following question that tests the skills learned.

As with the bomblab, you have to devise the inputs to this program. There are multiple inputs that solve this phase named *foo*. **Identify all the inputs that would defuse this phase**. The function *explode_bomb* has the same behavior as in bomblab. The function *sscanf* has the following prototype:

```
int sscanf(const char *str, const char *format, ...);
```

sscanf reads its input from the character string pointed to by str. It returns the number of input items successfully matched to the format and assigned. An example usage is

```
sscanf(ptr, "%d %d %d", &a, &b, &c);
```

The function prototype of the phase is as follows:

```
void foo(char* input);
```

Further, the bomblab designer has ensured that this phase can indeed be diffused without requiring gdb. To help the students the bomblab designer has also annotated the assembly code.

The required ASCII table for alphabets and numbers is provided for reference.

ASCII code	Character	ASCII code	Character
48	'0'	65	'A'
49	'1'	66	'B'
50	'2'	67	'C'
51	'3'	68	'D'
52	'4'	69	'E'
53	'5'	70	'F'
54	'6'	71	'G'
55	'7'	72	'H'
56	'8'	73	'I'
57	'9'	74	'J'

```
.LC0:
     .string "%d %c\n"                        ### some global string
        .text
.globl foo
        .type   foo, @function                ### phase begins here
foo:
        pushl   %ebp                          ### stack frame setup
        movl    %esp, %ebp                    ### stack frame setup
        subl    $40, %esp                     ### reserving stack space
        leal    -13(%ebp), %eax
        movl    %eax, 12(%esp)                ### sscanf has some arguments, store them on the stack
        leal    -12(%ebp), %eax
        movl    %eax, 8(%esp)                 ### sscanf has some arguments, store them on the stack
        movl    $.LC0, 4(%esp)                ### sscanf has some arguments, store them on the stack
        movl    8(%ebp), %eax
        movl    %eax, (%esp)                  ### sscanf has some arguments, store them on the stack
        call    sscanf
        movl    -12(%ebp), %eax
        testl   %eax, %eax
        je      .L3
        cmpl    $1, %eax
        jne     .L7
        jmp     .L8
.L3:
        movl    ptr1, %eax
        movzbl  3(%eax), %eax
        cmpb    -13(%ebp), %al
        je      .L6
        call    explode_bomb
        jmp     .L6
.L8:
        movl    ptr2, %eax
        movzbl  2(%eax), %eax
        cmpb    -13(%ebp), %al
        je      .L6
        call    explode_bomb
        jmp     .L6
.L7:
        call    explode_bomb
.L6:
        leave
        ret

.globl ptr1
        .section        .rodata.str1.1
.LC1:
        .string "cs214"
        .data
        .align 4
        .type   ptr1, @object
        .size   ptr1, 4
ptr1:                                        ###### some global variable here
        .long   .LC1
```

```
.globl ptr2                                    ######## some global variable here
        .section        .rodata.str1.1
LC2:
        .string "ee365"
        .data
        .align 4
        .type   ptr2, @object
        .size   ptr2, 4
ptr2:
        .long   .LC2
```

1. (2 points) How many inputs does the phase take? What are their types?

   Answer: 2 inputs. An integer and a character.

2. (1+1+1 points) How many global pointers are present in this code? What are those? What do they point to?

   Answer: There are two global pointers—*ptr1* and *ptr2*. They point to strings named $''cs214''$ and $''ee365''$.

3. (5 points) How many inputs diffuse this phase? How did you deduce it?

   Answer: 2 inputs. Looking at the two blocks L3 and L8.

4. (15 Points) Enumerate the inputs that diffuse the phase? Answer:

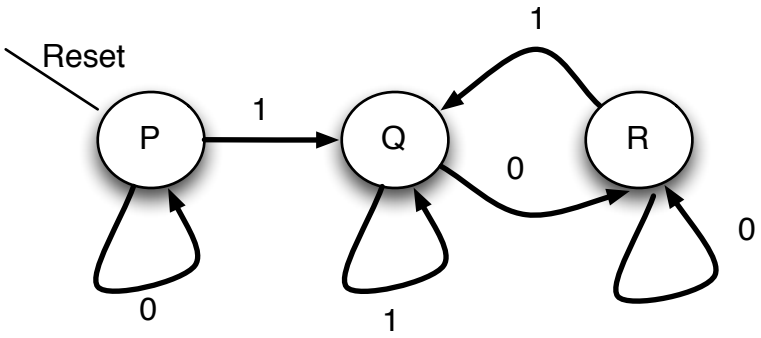
   - First input: Input integer is 0. Deduction from testl instruction in the assembly. If the input integer is 0, the code executes the L3 block. The input character should be same as the *(ptr1+3) which is '1'.
   - Second input: Input integer is 1. Deduction from cmp instruction in the assembly. If the input integer is 1, the code executes the L8 block. The input character should be same as *(ptr2+2), which is character '3'.

Problem 4: Logic Design (20 points)

1. (20 points) Design an FSM that detects the input stream encountered till now is divisible by 2 with a non-zero quotient. Implement it using a combination of sequential and combinational logic. Clearly show your work. Draw the truth table for inputs, outputs and your states. Draw the K-map for everything. Clearly indicate how many flip flops you are going to use.

```
INPUT : 0 1 1 0 1 1 1 0 1 0 ....

OUTPUT: 0 0 0 1 0 0 0 1 0 1 ....
```



R is the state that outputs 1, any other state outputs 0

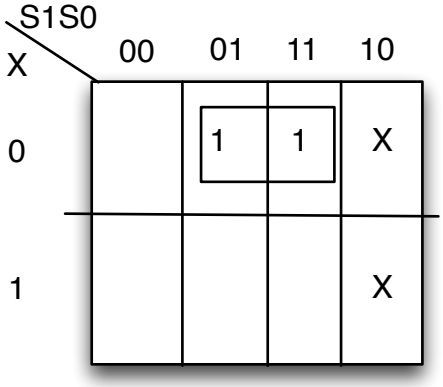
Encoding for the state

	S0	S1
P	0	0
Q	0	1
R	1	1

Truth Table for Next State (S1N and S0N are next states)

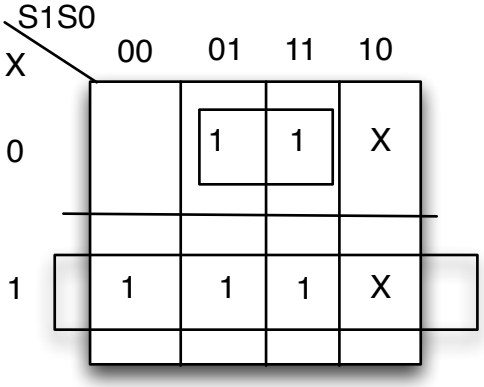
S1	S0	X	S1N	S0N
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	0	1
1	1	1	0	1
1	1	0	1	1
1	0	X	X	X

Next state for S1N



S0N = `X . S0

Next state for S0N

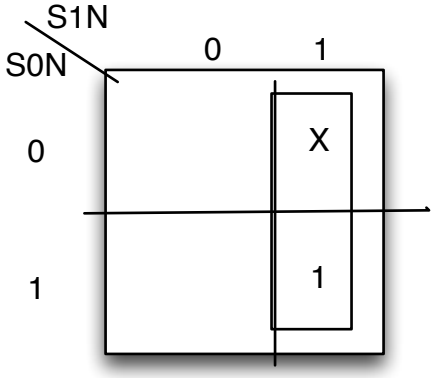


S0N = X + S0

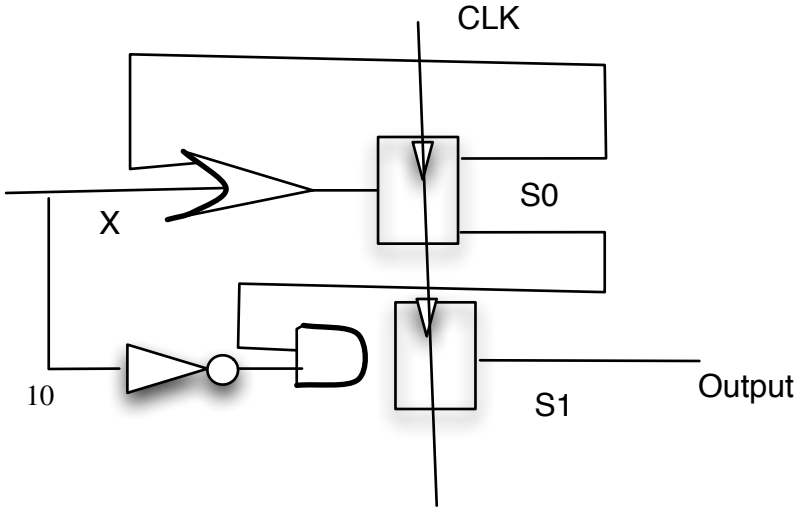
Truth Table for Output

S1N	S0N	Y
0	0	0
0	1	0
1	1	1
1	0	X

K-map for output

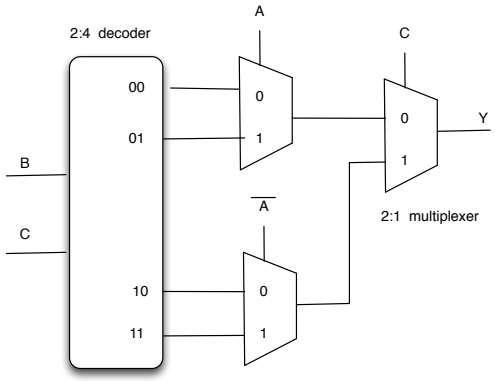


Y = S1



Problem 5: Logic Design (15 points)

1. (10 points) Consider the circuit given below with decoders and multiplexers. Identify the boolean expression performed by the circuit. Draw the resultant circuit with simple gates (AND, OR and NOT). Show work for your result.



Input to first mux.

$\bar{B}\bar{C}$ and $\bar{B}C$

o/p of first mux.

$\bar{A} \cdot \bar{B} \cdot \bar{C} + A \cdot \bar{B} C$

Input to second mux; $\bar{B}C$ and $B\bar{C}$

o/p of second mux; $\bar{A} \cdot B\bar{C} + \bar{A} \cdot BC$

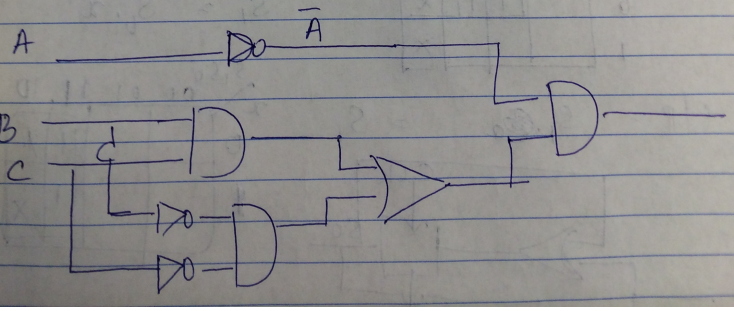
o/p of third mux.

$\bar{C} \cdot (\bar{A} \cdot \bar{B} \cdot \bar{C} + A \cdot \bar{B} C) +$

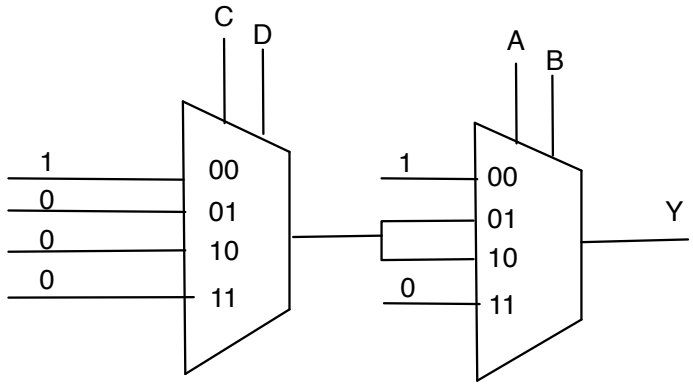
$C (A \cdot B\bar{C} + \bar{A} \cdot BC)$

$= \bar{A} \cdot \bar{B} \cdot \bar{C} + 0 + A \cdot 0 + \bar{A} \cdot BC$

$= \bar{A} \cdot (\bar{B}\bar{C} + BC)$



2. (5 points) Calculate the boolean expression performed by the circuit given below.



Answer:

$\overline{C}.\overline{D}.(\overline{A}.B + A.\overline{B}) + \overline{A}.\overline{B}$