Final Review Questions (with Answers)

The final exam will be broken into two components:

- A lab-based portion, on Monday, August 17, 2020. This is during the regularly-scheduled class time. The lab-based exam only is about assembly coding.
- A written (lecture-based) portion, on Tuesday, August 18, 2020. This is during the regularly-scheduled class time.

In lab-based portion will require you to write assembly code on your laptops, which is to be turned in via Canvas by the end of the class.

The written portion will consist of short-answer questions, and will require you to solve various problems.

Comprehensive Exam and Review

Both the lab and the lecture-based exams are intended to be comprehensive. However, since there has not been an exam since after Lab 6, **at least 60%** of the exam will concern material from labs 7 and 8.

Overall, the exam may include material from the following sources:

- This review
- The review for the midterm exam
- **All** your labs (labs 1 through 8).

I will **not** ask any questions which were not somehow covered by one of the above three sources.

It is also recommended to study your midterm exam, particularly any questions which you had difficulty with. I will occasionally repeat questions from the midterm exam on the final, especially if the class overall did poorly on a question (I want to make sure the concept is understood!).

Questions

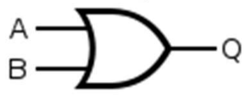
1. Convert the following Java/C-like code into ARM assembly. The names of the variables reflect which registers must be used for the ARM assembly. Non-register variable names indicate a value that should be stored in memory.

```
int[] first = new int[]{0, 5, 27, 98};
int[] second = new int[]{1, 2, 8, 29, 42};
int[] result = new int[9];
int r0 = 0;
int r1 = 0;

while (r0 < 4) {
  result[r0] = first[r0];
  r0++;
}

while (r1 < 5) {
  result[r0] = second[r1];
  r0++;
  r1++;
}

for (r2 = 0; r2 < 9; r2++) {
  print_int(result[r2]);
  print_newline();
}
```

```
  .equ SWI_Print_Int, 0x6B
  .equ SWI_Exit, 0x11
  .equ SWI_Print_Char, 0x00

  .data
first:
  .word 0, 5, 27, 98
first_length:
  .word 4
second:
  .word 1, 2, 8, 29, 42
second_length:
  .word 5
result:
  .word -1, -1, -1, -1, -1, -1, -1, -1, -1
result_length:
  .word 9

  .text
  .global _start
_start:
  ;; r0: from psuedocode
  ;; r1: from pseudocode
  ;; r2: length of first
  ;; r3: length of second
  ;; r4: first base address
  ;; r5: second base address
  ;; r6: result base address

  mov r0, #0
  mov r1, #0
  ldr r2, =first_length
  ldr r2, [r2]
  ldr r3, =second_length
  ldr r3, [r3]
  ldr r4, =first
  ldr r5, =second
  ldr r6, =result

begin_first_while:
  cmp r0, r2
  bpl end_first_while

  ;; r7: first[r0]
  ldr r7, [r4, r0, LSL #2]
  str r7, [r6, r0, LSL #2]
```

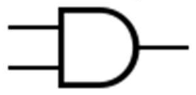
```
    add r0, r0, #1
    b begin_first_while

  end_first_while:

  begin_second_while:
    cmp r1, r3
    bpl end_second_while

    ;; r7: second[r1]
    ldr r7, [r5, r1, LSL #2]
    str r7, [r6, r0, LSL #2]

    add r0, r0, #1
    add r1, r1, #1
    b begin_second_while

  end_second_while:

    ;; r7: length of result
    mov r2, #0
    ldr r7, =result_length
    ldr r7, [r7]
  begin_for:
    cmp r2, r7
    bpl end_for

    mov r0, #1
    ldr r1, [r6, r2, LSL #2]
    swi SWI_Print_Int

    mov r0, #'\n
    swi SWI_Print_Char

    add r2, r2, #1
    b begin_for

  end_for:
    swi SWI_Exit
    .end
```

2. What component is shown below?



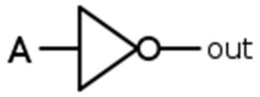
An OR gate.

3. What component is shown below?



An AND gate.

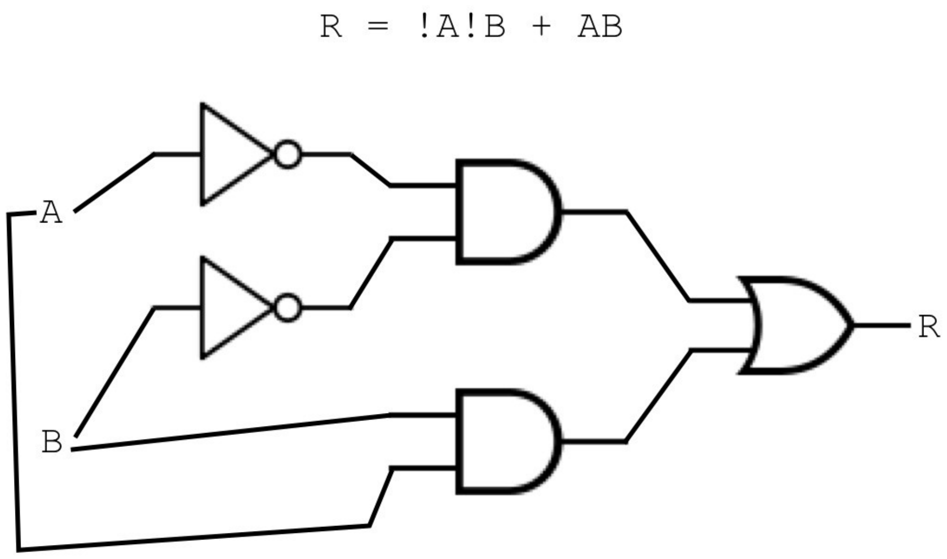
4. What component is shown below?



A NOT gate.

5. Draw the circuit corresponding to the following sum-of-products equation, where `!A` refers to the negation of variable `A`, and so on:

```
R = !A!B + AB
```



6. Consider the following sum-of-products equation:

```
R = !ABC + ABC + A!B!C
```

Using the above equation, do the following:

- Write it as a truth table:

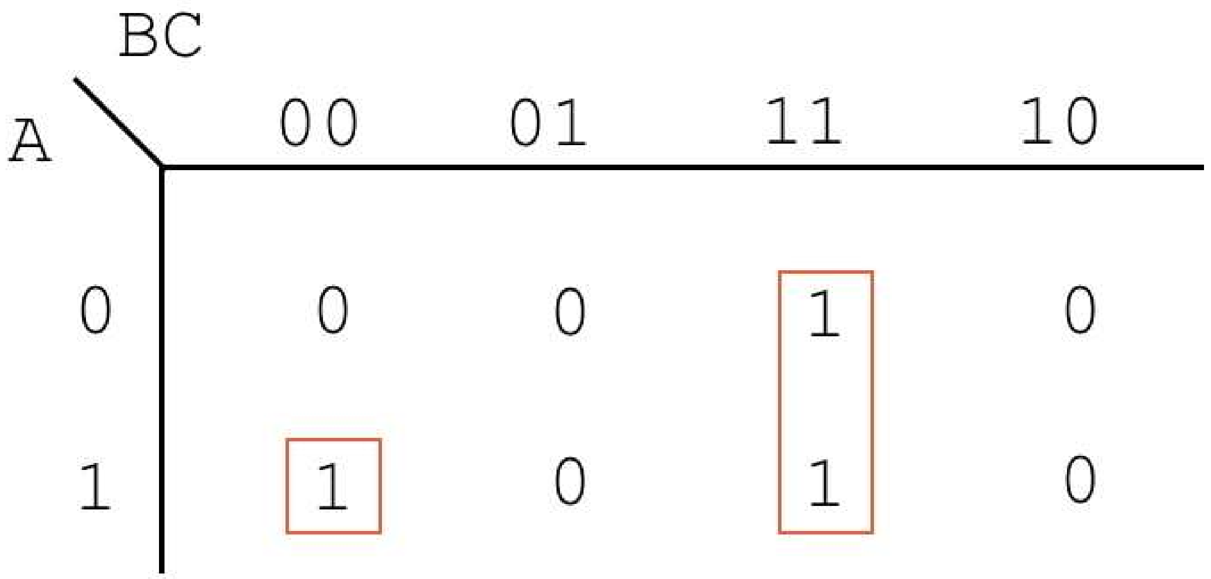
A	B	C	R
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

- Simplify it using boolean algebra:

```
R = !ABC + ABC + A!B!C
R = BC(!A + A)  + A!B!C
R = BC + A!B!C
```

- Simplify it using a Karnaugh map:

  `R = A!B!C + BC`



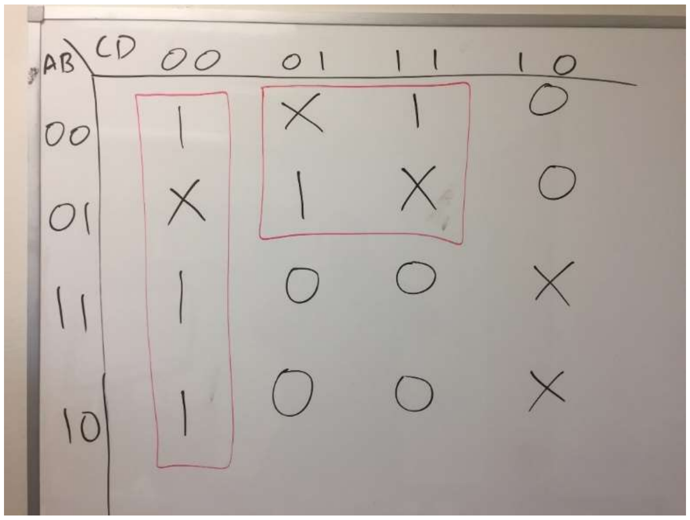
7. Consider the truth table augmented with *don't cares*, shown below:

A	**B**	**C**	**D**	**U**
0	0	0	0	1
0	0	0	1	X
0	0	1	0	0
0	0	1	1	1
0	1	0	0	X
0	1	0	1	1
0	1	1	0	0
0	1	1	1	X
1	0	0	0	1
1	0	0	1	0
1	0	1	0	X
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	X
1	1	1	1	0

8. Using the above truth table, write out the following:
   1. The unoptimized sum-of-products equation, skipping over *don't cares*

      `U = !A!B!C!D + !A!BCD + !AB!CD + A!B!C!D + AB!C!D`

2. A Karnaugh map, along with boxes which exploit *don't cares* where appropriate.



AB \ CD	00	01	11	10
00	1	X	1	0
01	X	1	X	0
11	1	0	0	X
10	1	0	0	X

3. An optimized sum-of-products equation, derived from the Karnaugh map created in the previous step.

```
U = !C!D + !AD
```