1 Boolean Logic

$\boxed{1.1}$ Simplify the following Boolean expressions:

(a) $(A + B)\left(A + \overline{B}\right)C$

(b) $\overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,B\,\overline{C} + A\,B\,\overline{C} + A\,\overline{B}\,\overline{C} + A\,B\,C + A\,\overline{B}\,C$

(c) $\overline{A\left(\overline{B}\,\overline{C} + BC\right)}$

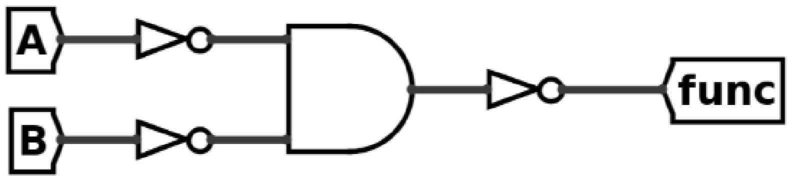
(d) $\overline{A}(A + B) + (B + AA)\left(A + \overline{B}\right)$

2 Digital Logic Simplification

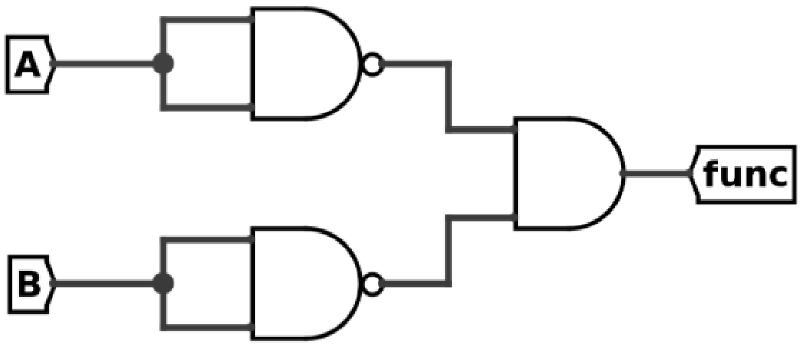
For the following digital logic circuits:

1. Write a boolean algebra expression that corresponds the physical circuit.
2. Simplify the expression and draw the simplified circuit.

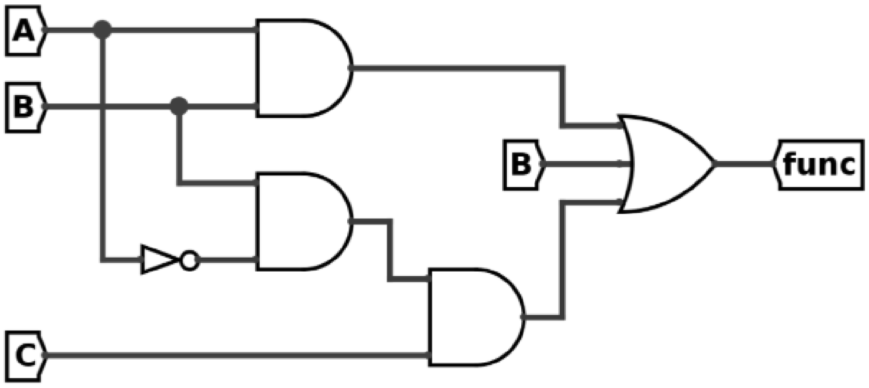
2.1



2.2



2.3



2.4 Why might it be useful to simplify logic circuits?

3 Combinational Logic from Truth Tables

For this question, we have a single 3-bit input and a single 4-bit output. We want to design a combinational logic circuit to achieve the desired output given the appropriate combinations of input bits (`Input=001` $\Longrightarrow$ `Output=0011`, and so on...). Here is the truth table we wish to implement:

Input	**Out**
`000`	`0001`
`001`	`0011`
`010`	`1111`
`011-111`	`xxxx`

The `x`'s for the final entry of the table indicate that any output is valid for the case that `Input` is `011`, `100`, `101`, `110`, and `111`

3.1 Write out and simplify boolean expressions for each of the output bits `Out[3], Out[2], Out[1],` and `Out[0]` in terms of the input bits `In[2], In[1], In[0]`.

3.2 Draw out the boolean circuit based on your simplified expressions above. You may use constants 0 and 1, and the logic gates AND, OR, NOT.

4 Two-Pass Assembly

Consider the following assembly code. Assume that **`printf`** exists in the C standard library and that **`msg`** exists at an unknown address in the **`.data`** section.

```
Address | Assembly
--------|---------------------------
.data   | msg: .string "Hello World"
        |
.text   |
0x0C    |        add  t0, x0, x0
0x10    |        addi t1, x0, 4
0x14    | loop:  beq  t0, t1, end
0x18    |        addi a0, a0, 1
0x1C    |        la   a0, msg      // load address of `msg`
0X20    |        jal  ra, printf
0X24    | n:     addi t0, t0, 1
0X28    |        j    loop
0X2C    | end:   ret
```

4.1 This code is output from the ____________________ (Compiler, Assembler, Linker, or Loader) and ________________ (may / may not) contain pseudoinstructions.

4.2 Assume we are using a two-pass assembler. Fill out the symbol table after the first pass (top-to-bottom) of the assembler. Not all lines may be used.

Symbol Table	
Label	**Address**

4.3 After the first pass of the assembler, which of the instructions do not have their addresses fully resolved?

$\boxed{4.4}$ After the second pass of the assembler but before the linker, which of the instructions do not have their addresses fully resolved?