2020 Fall

A.

Question A

Simplify the following Boolean expressions by finding a minimal sum-of-products expression for each one. These expressions can be reduced into a minimal sum-of-products (SOP) by repeatedly applying the Boolean algebra properties.

1. $\overline{(a + b \cdot \bar{c})} \cdot d + c$

   - A01 = ?

wrong answer

or $\overline{(a+b)} \cdot \bar{c}$

2. $a \cdot \overline{(b + c)}(c + a)$

   - A02 = ?

1. $\overline{(a+b)\cdot\bar{c}} = \bar{a}\cdot\bar{b} + c$, $(\bar{a}\cdot\bar{b}+c)\cdot d = \bar{a}\cdot\bar{b}\cdot d + c\cdot d$

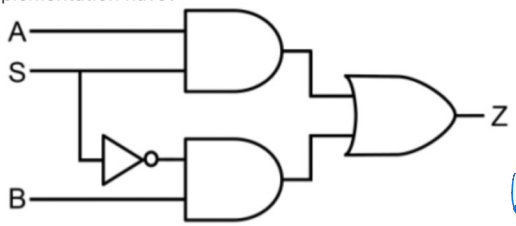
$\bar{a}\bar{b}d + cd + c = \bar{a}\bar{b}d + c(d+1) = \bar{a}\bar{b}d + c$

2. $a\cdot\overline{(b+c)}\,(c+a) = a\cdot(\bar{b}\cdot\bar{c})(c+a) = a\bar{b}\bar{c}\cdot(c+a)$

$= a\bar{b}\bar{c}c + a\bar{b}\bar{c}a = a\bar{b}\bar{c}$

B.

1. Consider the implementation shown below, which uses two AND gates and an OR gate. Because a single CMOS gate cannot implement AND or OR, each AND gate is implemented with a CMOS NAND gate followed by a CMOS inverter, and the OR gate is implemented with a CMOS NOR gate followed by a CMOS inverter. How many transistors does this implementation have?

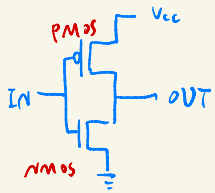


$6 \times 3 + 2 = 20$

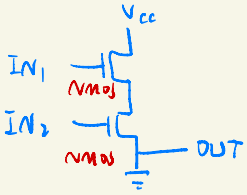
Number of transistors in mux: __ B01 __

   - B01 = ?

CMOS (Inverter)

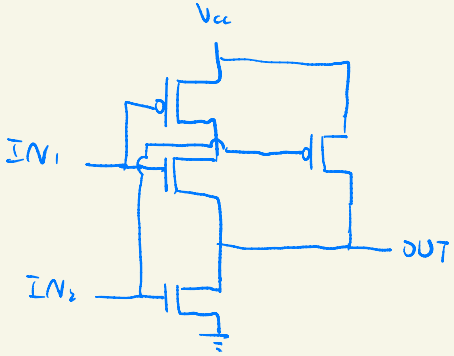
▷ NOT gate : 

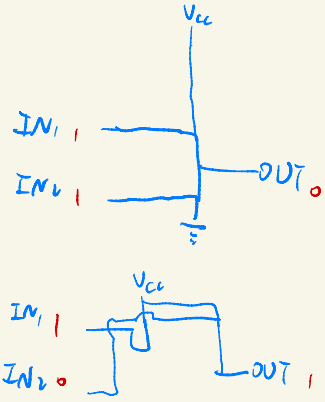
≥ transistors

▷ AND gate : 

(2 transistors)

If CMOS Only

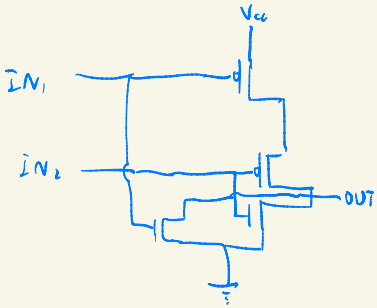
NAND gate : 



Use 4 transistors

Thus, AND gate consists of NAND gate and Inverter,

So we need 6 transistors.

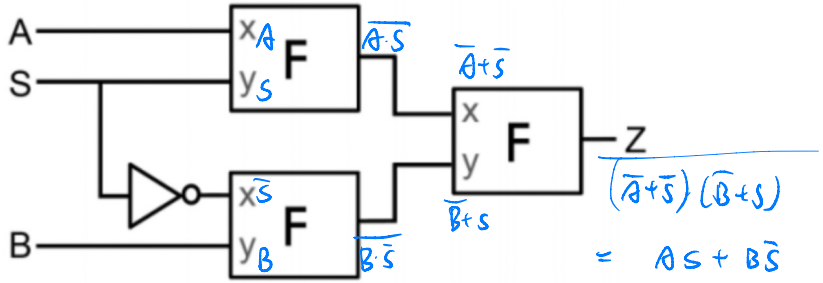
OR gate : 

4 transistors

NOR : 6 transistors

2.

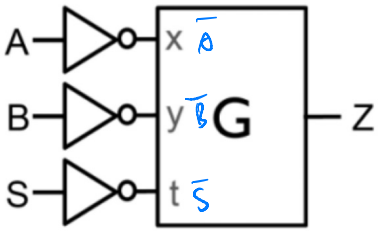
Consider the implementation shown below, which uses three instances of gate `F`. Find the Boolean expression for `F`. If F can be built using a single CMOS gate, say "Yes." Otherwise, give a convincing explanation for why `F` cannot be implemented as a CMOS gate. How many transistors does this implementation have?



Number of transistors in mux (if F can be built as a CMOS gate): __ B02 __

NAND $F(x,y) = \overline{x \cdot y}$ $4 \times 3 + 2 = 14$

3. Consider the implementation shown below, which uses gate `G`. Find the Boolean expression for `G`. If `G` can be built using a single CMOS gate, say "Yes." Otherwise, give a convincing explanation for why G cannot be implemented as a CMOS gate. How many transistors does this implementation have?

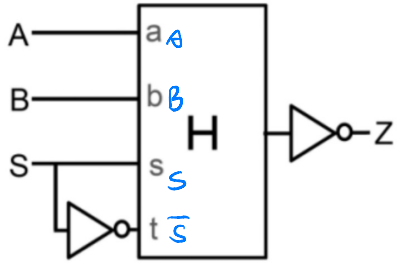


Number of transistors in mux (if G can be built as a CMOS gate): __ B03 __

G 不能被 CMOS gate 組合出來

why?

4. Consider the implementation shown below, which uses gate H. Find the Boolean expression for H. If H can be built using a single CMOS gate, say "Yes." Otherwise, give a convincing explanation for why H cannot be implemented as a CMOS gate. How many transistors does this implementation have?

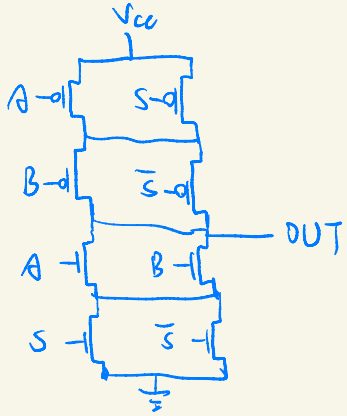


Number of transistors in mux (if H can be built as a CMOS gate): __ B04 __ 12

$Z = AS + B\bar{S}$ $\quad \bar{Z} = \overline{(AS + B\bar{S})} = (\bar{A} + \bar{S})\cdot(\bar{B} + S)$

$\bar{Z} = (\overline{A\cdot S})(\overline{B\bar{S}}) = (\overline{a\cdot s})(\overline{b\bar{t}})$

$H(a, b, s, t) = (\overline{a\cdot s})(\overline{b\cdot t})$



$(\overline{A\cdot S})(\overline{B\bar{S}})$

8 + 4 = 12

CMOS: 可以 implement NOT, NAND, NOR

1個 CMOS 2個 CMOS 2個 CMOS

Consider the C procedure below and its translation to RISC-V assembly code, following the C code.

- C procedure

```c
int f(int a, int b) {
   int c = b - a;
   if (c & C01 == 0) /* c is a multiple of 4 */
       return 1;
   int d = f(a - 1, b + 2);
   return 3 * (d + a);
}
```

- The translated RISC-V assembly code

```
f:    sub a2, a1, a0
      andi a2, a2, __C01__
      bnez a2, ELSE
      li a0, 1
      jr ra
ELSE: addi sp, sp, -8
      sw a0, 0(sp)
      sw ra, 4(sp)
      addi a0, a0, -1
      addi a1, a1, 2
      jal ra, f
A4:   lw a1, 0(sp)
      lw ra, 4(sp)
L1:   add a0, a0, a1
      slli a1, a0, 1
      add a0, a0, a1
      addi sp, sp, 8
      jr ra
```

1. What value should the C01 term in the C code and the assembly be replaced with to make the if statement correctly check if the variable c is a multiple of 4?
   - C01 = ?
2. How many words will be written to the stack before the program makes each recursive call to the function f?
   - C02 = ?

3. The program's initial call to function `f` occurs outside of the function definition via the instruction `jal ra, f`. The program is interrupted at an execution (not necessarily the first) of function f, just prior to the execution of `add a0, a0, a1` at label `L1`. The below diagram on the right shows the contents of a region of memory. All addresses and data values are shown in hex. The current value in the SP register is 0xEB0 and points to the location shown in the diagram.

	Memory Contents	
	Address	Data
	0xEA4	0x0
	0xEA8	0x1
	0xEAC	0xA4
SP→	0xEB0	0x2
	0xEB4	0xA4
	0xEB8	0x3
	0xEBC	0xA4
	0xEC0	0x4
	0xEC4	0x3B8
	0xEC8	0x12
	0xECC	0x44
	0xED0	0xCE8

3
2
1

What were the values of arguments a and b to the initial call to f? Write "UNKNOWN" if the argument does not show up in the stack.
Initial arguments to `f` : a = __ C03 __ ; b = __ C04 __

~~UNKNOWN~~ UNKNOWN
0x4

4. What are the values in the following registers right when the execution of f is interrupted? Write "UNKNOWN" if you cannot tell.
Current value (in hex) of `a1` : __ C05 __
Current value (in hex) of `ra` : __ C06 __

- C05 = ? 0x2
- C06 = ? A4

4
16
2
32
12

5. What is the hex address of the `jal ra, f` instruction that made the initial call to `f` ?
Address (in hex) of instruction that made initial call to `f` : __ C07 __

- C7 = ?

0x3B4
0x3B4 Jal ra f
0x3B8

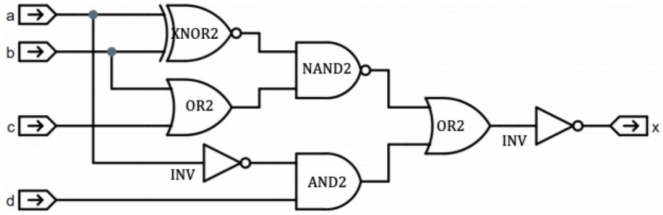
6. What is the hex address of the instruction at label `ELSE` ?
Address of instruction at label `ELSE` : __ C08 __

- C08 = ?

$24_{(10)} = 0x18$
0xA4 - 0x1f = 0xfC

Question D

1. Consider the logic diagram below, which includes `XNOR2` , `OR2` , `NAND2` , `AND2` , and `INV` . Using the t_{PD} information for the gate components shown in the table below, compute the t_{PD} for the circuit.



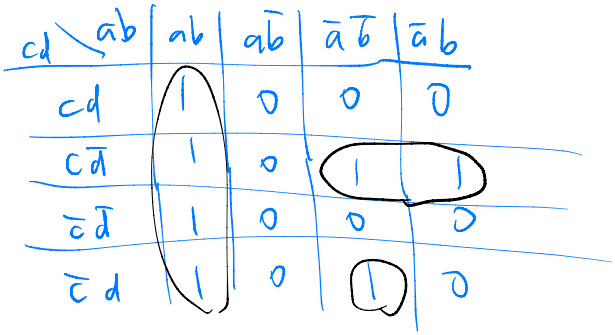
Gate	t_{PD}
XNOR2	7.0 ns
OR2	5.5 ns
NAND2	3.0 ns
AND2	5.0 ns
INV	2.0 ns
Longest path. t_{PD} in ns = D01	17.5ns

7 + 3 + 5.5 + 2 = 17.5

- D01 = ?

2. Find a minimal sum-of-products expression for output X of the circuit described by the truth table shown below.

a	b	c	d	X
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1



$ab + \bar{a}c\bar{d} + \bar{a}\bar{b}\bar{c}d$

Minimal sum of products for X = __ D02 __

- D02 = ?

Question E

1. What is the hexadecimal encoding of the RISC-V instruction `sw t1, -4(t1)`? You can use the table below to help you with the encoding.

[31:25]	[24:20]	[19:15]	[14:12]	[11:7]	[6:0]
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode

   - opcode = 0100011_2
   - funct3 = 010_2
   - `t1` = `x6` (ABI register name)
   - `rs1` = 00110, `rs2` = 00110

`sw t1, -4(t1)` instruction encoding in HEX: __ E01 __

   - E01 = ?

2. For the following code snippet, provide the value left in each register after executing the entire code snippet (i.e., when the processor reaches the instruction at the end label), or specify "UNKNOWN" if it is impossible to tell the value of a particular register.

```
. = 0x100
    li x4, 0x6
    addi x5, zero, 0xC00
    slli x4, x4, 8
    or x6, x4, x5
end:
```

   - `x4` = __ E02 __
   - `x5` = __ E03 __
   - `x6` = __ E04 __
   - `pc` = __ E05 __
   - E02 = ?
   - E03 = ?
   - E04 = ?
   - E05 = ?

F,

Consider the following program that computes the Fibonacci sequence recursively. The C code is shown on the below, and its translation to RISC-V assembly is provided as well. You are told that the execution has been halted just prior to executing the ret instruction.

- C code

```c
int fib(int n) {
    if (n <= 1) return n;
    return fib(n - 1) + fib(n - 2);
}
```

- The translated RISC-V assembly

```
fib:  addi sp, sp, -12
      sw ra, 0(sp)
      sw a0, 4(sp) # a0 is n
      sw s0, 8(sp)
      li s0, 0
      li a7, 1
if:   ble __F01__      ble a0, a7, done
sum:  addi a0, a0, -1
      call fib
      add s0, s0, a0
      lw a0, 4(sp)
      addi a0, a0, -2
      call fib
      mv t0, a0
      add a0, s0, t0
done: lw ra, 0(sp)
      lw s0, 8(sp)
L1:   addi sp, sp, 12
      ret
```

Complete the missing portion of the `ble` instruction to make the assembly implementation match the C code.

- F01 = ?

How many distinct words will be allocated and pushed into the stack each time the function `fib` is called? Number of words pushed onto stack per call to fib: __ F02 __

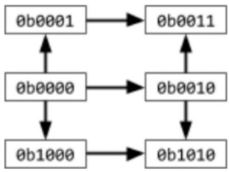
- F02 = ? 3

2021 Fall

Problem A

We are given an array of n unique `uint32_t` that represent nodes in a directed graph. We say there is an edge between A and B if `A < B` and the Hamming distance between A and B is exactly `1`. A Hamming distance of `1` means that the bits differ in `1` (and only 1) place. As an example, if the array were `{0b0000, 0b0001, 0b0010, 0b0011, 0b1000, 0b1010}`, we would have the edges shown as following:

A	B
0b0000	0b0001
0b0000	0b0010
0b0000	0b0100
0b0001	0b0011
0b0010	0b0011
0b0010	0b1010
0b1000	0b1010



> See also: LeetCode 461. Hamming Distance

Construct an `edgelist_t` (specified below) that contains all of the edges in this graph.

```
typedef struct { uint32_t A, B; } edge_t;
typedef struct {
    edge_t *edges;
    int len;
} edgelist_t;
```

Our solution used every line provided, but if you need more lines, just write them to the right of the line they are supposed to go after and put semicolons between them. All of the necessary `#include` statements are omitted for brevity; do not worry about checking for `malloc`, `calloc`, or `realloc` returning `NULL`. Make sure `L->edges` has no unused space when `L` is eventually returned.

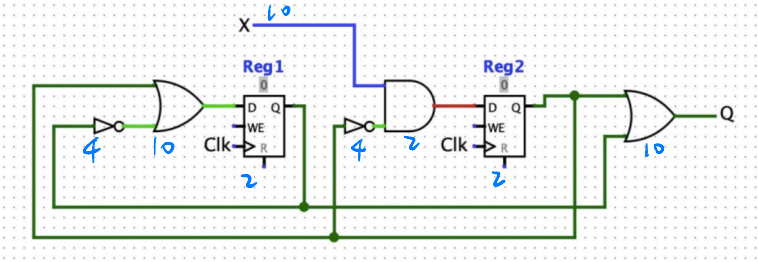
```
edgelist_t *build_edgelist(uint32_t *nodes, int n)
{
    edgelist_t *L = malloc(sizeof(edgelist_t));
    L->len = 0;

    L->edges = malloc(n * n * sizeof(edge_t));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            uint32_t tmp = A01;
            if ((nodes[i] < nodes[j]) && !(A02)) {
                A03;
                A04;
                L->len++;
            }
        }
    }
    L->edges = realloc(L->edges, sizeof(edge_t) * L->len);
    return L;
}
```

- A01 = ?
- A02 = ?
- A03 = ?
- A04 = ?

Problem B

Consider the following circuit:



You are given the following information:

- `Clk` has a frequency of 50 MHz
- AND gates have a propagation delay of 2 ns
- NOT gates have a propagation delay of 4 ns
- OR gates have a propagation delay of 10 ns
- `X` changes 10ns after the rising edge of `Clk`
- `Reg1` and `Reg2` have a clock-to-Q delay of 2 ns

The clock period is $\frac{1}{50\times10^6}s = 20ns$. This means that if `X` changes, it changes 10 ns after the clock positive edge.

1. What is the longest possible setup time such that there are no setup time violations? (Please include ns in your answer.)

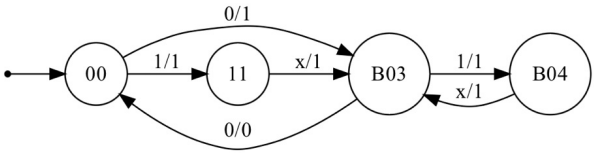
   B01 = ? $\min(20-(4+10+2),\ 20-(10+2)) = 4$

2. What is the longest possible hold time such that there are no hold time violations? (Please include ns in your answer.)

   B02 = ? $\min(2,8) = 8$

   when F, reg value change so should be hold

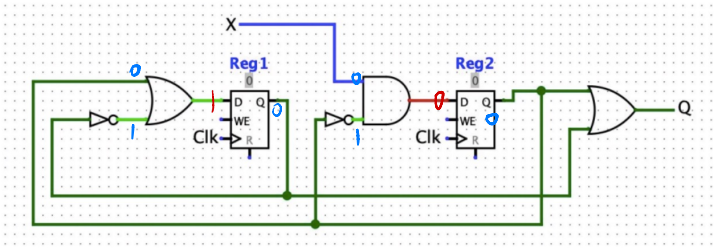
3. Represent the circuit above using an equivalent FSM, shown in the following, where X is the input and Q is the output, with the state labels encoding Reg1Reg2 (e.g., `01` means `Reg1 = 0` and `Reg2 = 1`). We did one transition already.



- B03 = ?
- B04 = ?

B03

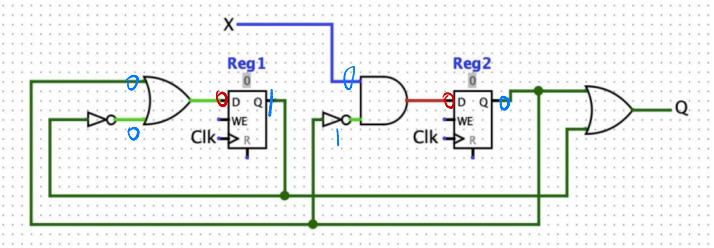
Consider the following circuit:



from state 00 to B03
10

B04

Consider the following circuit:



from state B03 (10) to B04
00

Problem C

What is the FULLY SIMPLIFIED (fewest primitive gates) circuit for the equation below? You may use the following primitive gates: AND, NAND, OR, NOR, XOR, XNOR, and NOT. (You can use the LaTeX syntax `\overline A` to represent $\overline{A}$)

$$\overline{\overline{(C + AB\overline{C} + \overline{B}\overline{C}D)}} + \overline{\overline{(C + \overline{B + D})}} = C01$$

- C01 = ?

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

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

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

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

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

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

$$= \overline{C}(BD + \overline{A}\,\overline{D} + 1) = \overline{C}$$

D.

```
1   .text
2           mv s1, a0
3           addi s2, s2, 4
4   Start:  beq s1, x0, End
5           lw a0, 0(s1)
6           jal ra, printf
7           add s1, s2, s1
8           lw s1, 0(s1)
9           jal x0, Start
10  End:    jalr x0, ra, 0
```

Recall that immediate values are generated from instructions with the following table:

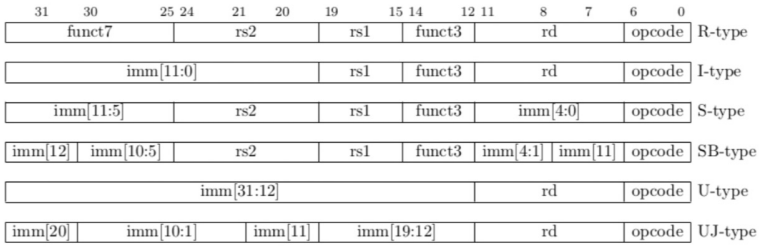


Figure 2.3: RISC-V base instruction formats showing immediate variants.

We will refer to the number produced after this process is completed as the "immediate value."

What are the fields for the machine code generated for `beq s1, x0, End` (line 4)?

Immediate value

- D01 = ? 24

funct3

- D02 = ? $000_{(2)} = 0_{(10)}$

opcode

- D03 = ? $1100011 = 64+32+2+1 = 99$

rs1

- D04 = ? 9

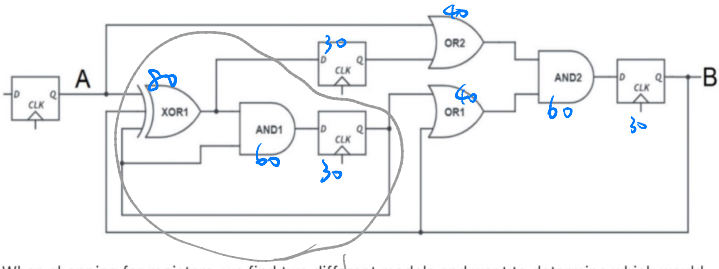
rs2

- D05 = ? 0

E.

Consider the following pipelined circuit. Assume all registers have their clock inputs correctly connected to a global clock signal and that logic gates have the following parameters:

- XOR gate delay: 80 ps
- AND gate delay: 60 ps
- OR gate delay: 40 ps



When shopping for registers, we find two different models and want to determine which would be best for our circuit.

critical path

Register Type λ

- Setup Time: 40 ps
- Hold Time: 20 ps
- Clock-to-Q Delay: 30 ps

Register Type τ

- Setup Time: 10 ps
- Hold Time: 10 ps
- Clock-to-Q Delay: 80 ps

1 What is the minimum latency for the circuit from A to B if we use register type λ ? (Please include ps in your answer.)

$(80+60+30+40)\times 2 = 420$ ps

- E01 = ?

2 What is the minimum latency for the circuit from A to B if we use register type τ ? (Please include ps in your answer.)

- E02 = ?

$(80+60+80+60)\times 2 = 460$

F.

Consider the following RISC-V code:

```
Loop:   andi t2, t1, 1
        srli t3, t1, 1
        bltu t1, a0, Loop
        jalr s0, s1, MAX_POS_IMM
        ...
```

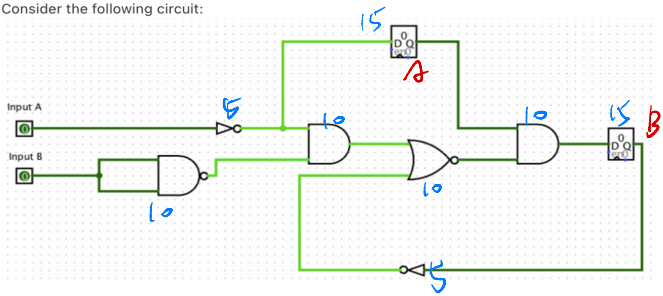
1. What is the value of the byte offset that would be stored in the immediate field of the `bltu` instruction?
   - F01 = ?

2. We would like to propose a revision to the standard 32-bit RISC-V instruction formats where each instruction has a unique opcode (which still is 7 bits). This justifies taking out the `funct3` field from the R, I, S, and SB instructions, allowing you to allocate bits to other instruction fields except the opcode field. Assume register `s0 = 0x1000 0000`, `s1 = 0x4000 0000`, `PC = 0xA000 0000`. Let's analyze the instruction: `jalr s0, s1, MAX_POS_IMM` where `MAX_POS_IMM` is the maximum possible positive immediate for `jalr`. After the instruction executes, what are the values in the following registers? (Answer in HEX)
   - `s0` = F02
   - `s1` = F03
   - `PC` = F04
   - F02 = ?
   - F03 = ?
   - F04 = ?

Full: https://inst.eecs.berkeley.edu/~cs61c/sp21/pdfs/exams/Su18_MT1_Solutions.pdf

G.

Consider the following circuit:



Assume input A and input B come from registers. Assume all 2-input logical gates have a 10 ns propagation delay. The `NOT` gate has a 5 ns delay. All registers have a clk-to-q of 15 ns and setup time of 20 ns.

1. Find the minimum clock period to ensure the validity of the circuit. (Please include ns in your answer)
   - G01 = ?
2. Find the maximum hold time such that there are no hold time violations. (Please include ns in your answer)
   - G02 = ?

H.

We wish to implement a function, `reverse`, that will take in a pointer to a string, its length, and reverse it. Assume that the argument registers, `a0` and `a1`, hold the pointer to and length of the string, respectively. Complete the following code skeleton to implement this function.

```
reverse:
    # This part saves all the required registers you will use.
    mv s0, a0 # memory address
    mv s1, a1 # strlen
    addi t0, x0, 0 # iteration  t0=0
Loop:
    # retrieve left and right letters
    add t1, s0, t0 # t1 is moving pointer from left (base + offset/iteration)
    lb t2, 0(t1) # t2 contains char from left
    sub t3, s1, t0 # imm needs to be s1 - t0
    H01 # since strlen indexes out of string
    add t4, s0, t3 # t4 is moving pointer from right (base + strlen - offset/iteration - 1)
    lb t5, 0(t4) # t5 contains char from right
    # switch chars
    sb t2, 0(t4)
    H02
    # iterate if necessary
    addi t0, t0, 1 # update iter
    H03
    H04
    mv a0, s0 # not necessary
    # This part restores all of the registers which were used.
    ret
```

- H01 = ?
- H02 = ?
- H03 = ?
- H04 = ?

H01: addi t3, t3, -1 要減 1

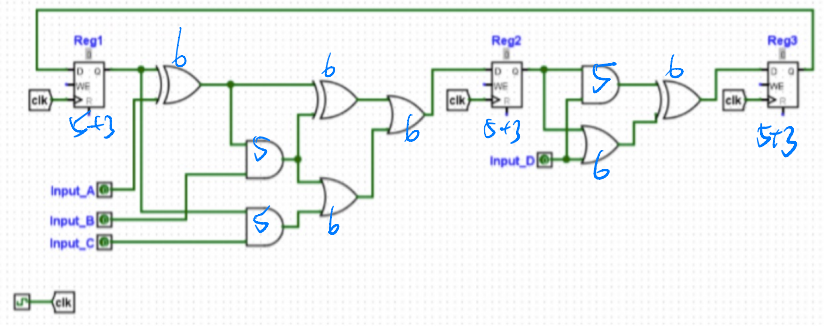
H02: sb t5, 0(t1)

H03: sr|; (s8), s1, 1 why s8

H04: bne t0, s8, Loop

J.

Take a look at the following circuit:



We have a register clk-to-Q time of 5ps, a hold time of 2ps, and a setup time of 3ps. AND and NAND gates have a delay of 5ps, OR and XOR gates have a delay of 6ps, and NOT gates have a delay of 1ps. Assume that our inputs A, B, C, and D arrive on the rising edge of the clock.

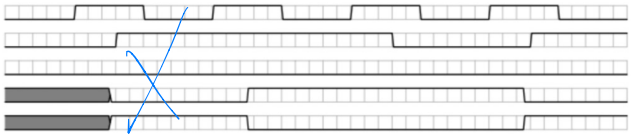
1. Which gates make up the critical path in the circuit above? Your answer should be correctly ordered from left to right, e.g. NOT → OR → NAND.
   - J01 = ? Input_A → XOR → AND → XOR → OR → Reg ✓

2. What is the critical path delay in the circuit?
   - J02 = ? 6+5+6+6+8 = 31ps

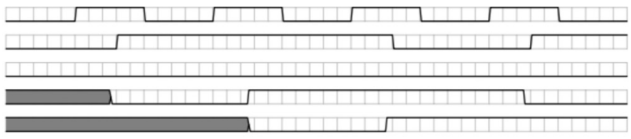
3. Let us now consider only the portion of the circuit between `Reg2` and `Reg3`. Assume that the clock period (rising edge to rising edge) is 100 ps, registers have a clk-to-Q delay of 25ps and a setup and hold time of 20ps, and all gates have a delay of 5ps. Choose the waveform with the correct outputs for `Reg2` and `Reg3`.

B

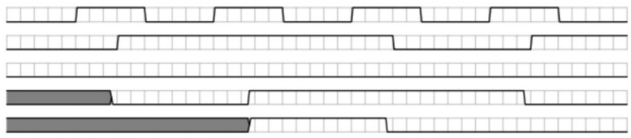
Option A



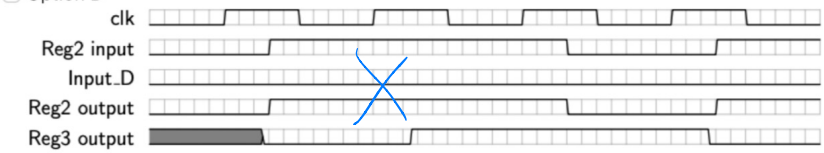
Option B



Option C



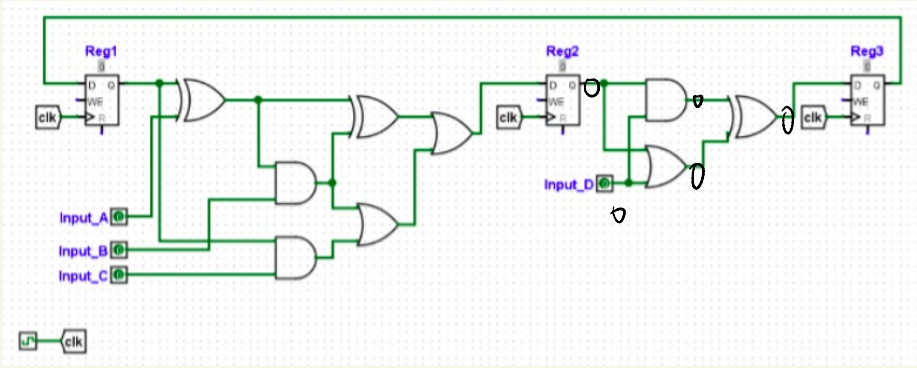
Option D



Notation: For reference, in the diagram below, the first region indicates an “undefined” signal, the second region indicates a signal of “high” or 1, and the third region indicates a signal of “low” or 0.

Undefined → High → Low 

> J03 = ?



K.

Consider the following program that computes the Fibonacci sequence recursively. The C code is shown on the left, and its translation to RISC-V assembly is provided on the right. You are told that the execution has been halted just prior to executing the ret instruction. The SP label on the stack frame (part 3) shows where the stack pointer is pointing to when execution halted.

- C code

```c
int fib(int n)
{
    if (n <= 1) return n;
    return fib(n - 1) + fib(n - 2);
}
```

- RISC-V Assembly (incomplete)

```
fib:  addi sp, sp, -12
      sw ra, 0(sp)
      sw a0, 4(sp) # a0 is n
      sw s0, 8(sp)
      li s0, 0
      li a7, 1
if:   ble __K01__
sum:  addi a0, a0, -1
      call fib
      add s0, s0, a0
      lw a0, 4(sp)
      addi a0, a0, -2
      call fib
      mv t0, a0
      add a0, s0, t0
done: lw ra, 0(sp)
      lw s0, 8(sp)
L1:   addi sp, sp, 12
      ret
```

1. Complete the missing portion of the `ble` instruction to make the assembly implementation match the C code.
   - K01 = ? ble a0, a7, done
2. How many distinct words will be allocated and pushed into the stack each time the function `fib` is called?
   - K02 = ? 3

3. Please fill in the values for the blank locations in the stack trace below. Please express the values in HEX.

Notation	address
Smaller address	0x280
	0x1
	K03
SP →	K04
	K05
	0x0
	0x280
	0x3
	0x0
	0x2108
	0x4
	0x6
Larger address	0x1

- K03 = ?
- K04 = ?
- K05 = ?

What is the hex address of the `done` label? (Answer in HEX)

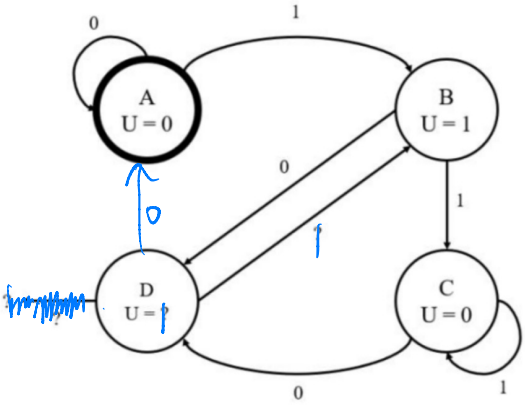
- K06 = ?

What was the address of the original function call to `fib`? (Answer in HEX)

- K07 = ?

Suppose we want to create a system that decides if the concatenation of its previous 2 single-bit inputs is a power of 2 (where the MSB is the input from 2 cycles ago and the LSB is from 1 cycle ago). If the previous 2 bits (prior to the current input) are a power-of-two the system outputs a 1, otherwise it outputs 0. Before any input is sent, assume the initial previous 2 bits are 2'b00.

A partial finite state machine diagram of this circuit is shown below:



> Before receiving any inputs the FSM is in state A.

1. For this FSM to provide the correct answer, to what existing states must D transition to (A, B, C, or D), and what output does D give (0 or 1)?
    - Current State = D, Input = 0, Next State = __ L01 __
    - Current State = D, Input = 1, Next State = __ L02 __
    - Current State = D, Output = __ L03 __
        - L01 = ?
        - L02 = ?
        - L03 = ?

2022 Fall

A.

You may use the strlen function to count the length of a string in the C programming language. For this function to be implemented in RISC-V, please fill out the following.

Pseduoinstructions MUST NOT be used.

```
    addi s1, x0, 1
    jal strlen
    add a0, x0, s1
    ecall
strlen:
    A01 # Fill your answer
    sw s1, 0(sp)
    mv a1, x0
    la s1, str
loop:
    A02 # Fill your answer
    A03 # Fill your answer
    addi a1, a1, 1
    addi s1, s1, 1
    j loop
epilogue:
    lw s1, 0(sp)
    A04 # Fill your answer
    jr ra
```

- A01 = ?
- A02 = ?
- A03 = ?
- A04 = ?

B.

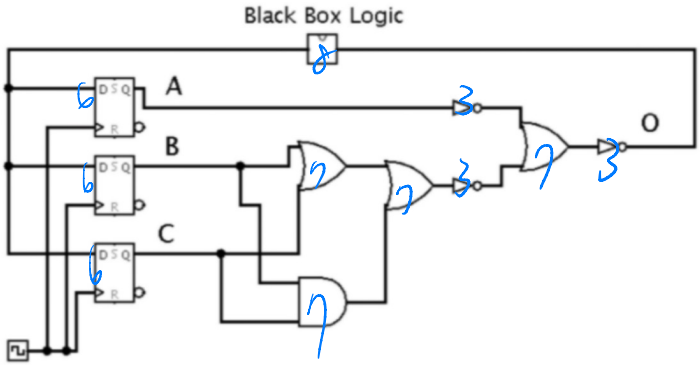
Write a RISC-V function that returns `0` if the 32-bit float input is an infinite value and a non-zero value otherwise. As usual, `a0` will be used to hold the input and output. Pseduoinstructions MUST NOT be used.

```
is_not_inf:
    B01 # Fill your answer
    addi a1, a1, -1
    and a0, a0, a1
    B02 # Fill your answer
    B03 # Fill your answer
    ret # Return instruction
```

- B01 = ?
- B02 = ?
- B03 = ?

C.

The following circuit has a delay of 3 ns for NOT gates, 7 ns for AND and OR gates, and 8 ns for the "Black Box" logic component. The registers have setup times of 5 ns and a clk-to-q latency of 6ns.



1. What is the maximum allowable hold time of the registers? __ C01 __ ns
   - C01 = ?

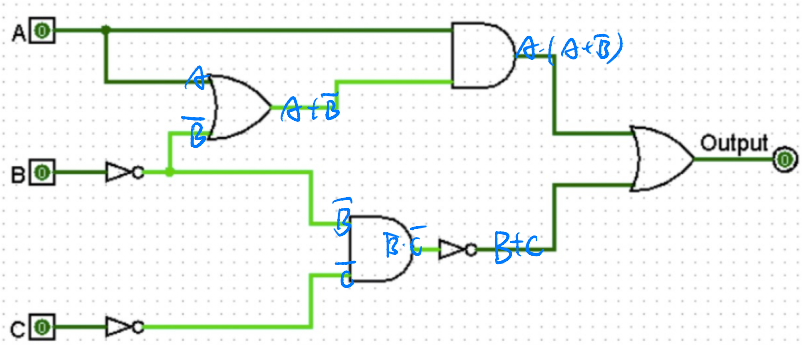
   min(11, 33, 33) = 11

2. What is the minimum acceptable clock period for this circuit? __ C02 __ ns
   - C02 = ?

   7+7+3+7+3+8+6+5 = 46

Problem D

The circuit seen below can be made simpler. Please write the circuit's origin boolean expression and then simplify it using the fewest possible two-input logic gates. __ D01 __



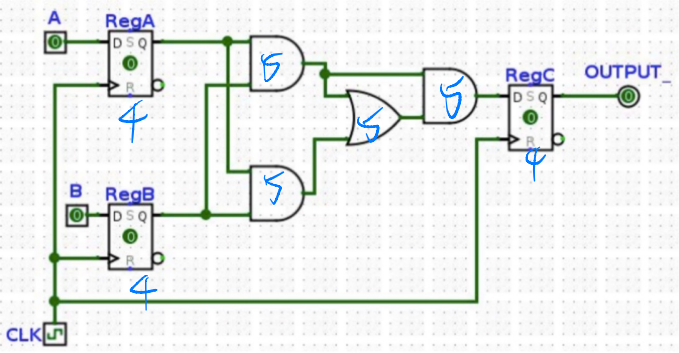
- D01 = ?

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

$= A + B + C$

Problem E

All logic gates in the circuit below have a delay of 5 ns, RegC has a setup time of 6 ns, and RegA and RegB have setup, hold, and clk-to-q durations of 4 ns.



1. What is the maximum allowable hold time for RegC? __ E01 __ ns
   - E01 = ?

   $10+4=14$
2. What is the minimum acceptable clock cycle time for this circuit (E02 ns), and clock frequency does it correspond to (E03 MHz)?
   - E02 = ?
   - E03 = ?

   $5+5+5+6+4 = 25$ ns

   $\frac{1}{25} \times 10^9 = \frac{100}{25} \times 10^7 = 40$ MHz

F.

Look at the RISC-V function below, whose start address is in a0 and length is in a1, which sorts a word array in-place.

```
sort:
    # prologue
    mv s0, a0
    0x00259493 # p1
    addi s1, s1, -4
    add s1, s1, s0
    mv t0, s0
outer_loop:
    mv t1, t0
    0x00032383 # p2
    mv t3, t1
inner_loop:
    addi t1, t1, 4
    lw t4, 0(t1)
    ble t4, t2, mystery_label
    mv t2, t4
    mv t3, t1
mystery_label:
    blt t1, s1, inner_loop
    lw t5, 0(t0)
    sw t2, 0(t0)
    sw t5, 0(t3)
    addi t0, t0, 4
    blt t0, s1, outer_loop
    # epilogue
    ret
```

0000 0000 0010 | 0101 | 1001 | 0100 | 1001 0011
imm rs1 funct3 rd slli
11 9

0000 0000 0000 | 0011 | 0010 | 0011 | 1000 0011
imm rs1 funct3 rd lw
0 6 7

1 Please disassemble machine code marked #p1 and #p2 above to RISC-V instructions. (Please use register names, e.g., s0, s1, etc., NOT x8, x9, etc.)

- 0x00259493 # p1: __ F01 __
- F01 = ? slli s1, a1, 2

2

- 0x00032383 # p2: __ F02 __
- F02 = ? lw t2, 0(t1)

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5	t0	Temporary/alternate link register	Caller
x6–7	t1–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller
f0–7	ft0–7	FP temporaries	Caller
f8–9	fs0–1	FP saved registers	Callee
f10–11	fa0–1	FP arguments/return values	Caller
f12–17	fa2–7	FP arguments	Caller
f18–27	fs2–11	FP saved registers	Callee
f28–31	ft8–11	FP temporaries	Caller

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

RV32I Base Instruction Set

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]			rs1	000	rd	1100111	JALR
imm[12|10:5]		rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]		rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]		rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]		rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]		rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]		rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]			rs1	000	rd	0000011	LB
imm[11:0]			rs1	001	rd	0000011	LH
imm[11:0]			rs1	010	rd	0000011	LW
imm[11:0]			rs1	100	rd	0000011	LBU
imm[11:0]			rs1	101	rd	0000011	LHU
imm[11:5]		rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]		rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]		rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]			rs1	000	rd	0010011	ADDI
imm[11:0]			rs1	010	rd	0010011	SLTI
imm[11:0]			rs1	011	rd	0010011	SLTIU
imm[11:0]			rs1	100	rd	0010011	XORI
imm[11:0]			rs1	110	rd	0010011	ORI
imm[11:0]			rs1	111	rd	0010011	ANDI
0000000		shamt	rs1	001	rd	0010011	SLLI
0000000		shamt	rs1	101	rd	0010011	SRLI
0100000		shamt	rs1	101	rd	0010011	SRAI
0000000		rs2	rs1	000	rd	0110011	ADD
0100000		rs2	rs1	000	rd	0110011	SUB
0000000		rs2	rs1	001	rd	0110011	SLL
0000000		rs2	rs1	010	rd	0110011	SLT
0000000		rs2	rs1	011	rd	0110011	SLTU
0000000		rs2	rs1	100	rd	0110011	XOR
0000000		rs2	rs1	101	rd	0110011	SRL
0100000		rs2	rs1	101	rd	0110011	SRA
0000000		rs2	rs1	110	rd	0110011	OR
0000000		rs2	rs1	111	rd	0110011	AND
fm	pred	succ	rs1	000	rd	0001111	FENCE
1000	0011	0011	00000	000	00000	0001111	FENCE.TSO
0000	0001	0000	00000	000	00000	0001111	PAUSE
000000000000			00000	000	00000	1110011	ECALL
000000000001			00000	000	00000	1110011	EBREAK

F.

2. `li x1, 0xDCBAABCD` is also a pseudo instruction. Below it is expanded to 2 normal instructions. Please fill in the blanks and translate each of them to machine code.

Assembly	Machine code
lui x1, 0x __ F03 __	0x __ F04 __
addi x1, __ F05 __ , __ F06 __	0x __ F07 __

- F03 = ? DCBAB
- F04 = ? DCBAABCD
- F05 = ? x1
- F06 = ? -1015
- F07 = ? 0xBCD3093

00001 011011 = 0x 0 B 7

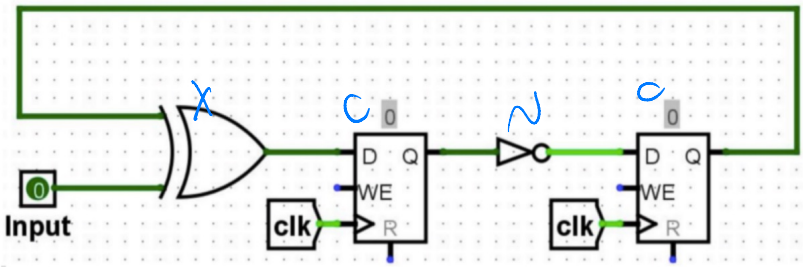
00001 000 00001 001001 = 0x 0 8 0 9 3

G.

Assume that all input originates from a register and that no hold time laws have been broken. What is the highest frequency at which you can run this circuit's clock such that it operates properly? __ G01 __

Using these variables, formulate your response as a mathematical expression (you may also use `min()`, `max()`, `abs()`, and other simple operations if necessary):

- X = XOR delay
- N = NOT delay
- C = $t_{clk-to-Q}$
- S = t_{setup}
- H = t_{hold}



- G01 = ?

$$\frac{1}{\max(X+C+S,\ N+C+S)}$$

Problem `H`

The `fabs` instruction in the x86 architecture returns a floating-point number's absolute value. Compared to utilizing branches, this command is quicker.

1. Complete the following C code that mimics this instruction without ternary operators or `if-else` statements. Assume that int and float are of equal size. (The x86 architecture makes use of the IEEE 754).

```c
float fabs(float x) {
    int tmp = *(int *) &x;
    tmp &= H01 /* Your code here */;
    return H02 /* Your code here */;
}
```

   - H01 = ? 0x7FFFFFFF
   - H02 = ? *(float *)&tmp

2. Can we use the same method to get the absolute value of a single-precision 32-bit HFP number? __ H03 __ (Yes or No) Why? __ H04 __

   - H03 = ? Yes
   - H04 = ? Sign bit is the same position

J

We wish to develop a new ISA while we work on a new processor for an embedded application. We choose to include only one, the X-type instruction, because we are tired of the several RISC-V instruction types. Let's say we want to include the instructions below:

```
add rd1, rs1, rs2
and rd1, rs1, rs2
lw rd1, offset1 (rs1)
sw rs2, offset1 (rs1)
addi rd1, rs1, imm1
beq rs1, rs2, offset1
lui rd1, offset1
jal rd1, imm
stw rs3, offset1, offset2 (rs1)
```

The new stw instruction stores the contents of rs3 into both `rs1 + offset1` and `rs1 + offset2`.

1 We want to do away with the `funct3` and `funct7` fields and only use an opcode. If we only wish to support the instructions listed above, what is the minimum number of bits the opcode field can be? __ J01 __ bit(s)

- J01 = ? 4

2 We want to be able to jump up to 64 KiB in either direction with a single instruction. How many bits are necessary to encode an immediate that would allow us to do this? __ J02 __ bit(s) Assume that, just like RV32, the least significant bit is an implicit 0 and is not stored in the instruction.

- J02 = ? 16

3 Then, we switch to a 32-bit machine, and finalize our instruction format to have 4 bits for each of the immediate fields, 4 bits for each register, and 4 bits for the opcode. Convert the instruction `stw x8, 0, 4 (x5)` into machine code. Leave your answer in binary (don't forget the prefix!). If a field is not used, fill in the field with xs. __ J03 __

- J03 = ?

A bloom filter is a very clever data structure for effectively and probabilistically storing a set. It does two functions: insert and check. The fundamental principle of checking is to hash your search term many times. Each hash identifies a specific bit that has to be set or checked. So, to verify, you check to see if the bit is set. You do this repeatedly, with each iteration's count of repetitions being included in the hash (so each hash is different). The element does not exist in the bloom filter if not all bits are set. The element PROBABLY occurs in the bloom filter if all bits are set. Similar to that, you just set an element as present when using a bloom filter.

Similar to that, you just set all those bits to 1 to indicate that an element is present in a bloom filter.
A flexible and portable bloom filter design is what we are aiming for. Therefore, we create the structure below.

```
struct BloomFilter {
4B  uint32_t size; /* Size is # of bits, NOT BYTES, in the bloom filter */
2B  uint16_t itercount;
    uint64_t (*)(void *data, uint16t iter) hash;
4B  uint8_t *data;
};
```

On a 32-bit architecture that requires word alignment for 32-bit integers and pointers, what is `sizeof(struct BloomFilter)`? __ K01 __ bytes

- K01 = ?

And now we have the `insert` function. For this we need to set the appropriate bit for each iteration.

```
void insert(struct BloomFilter *b, void *element)

{
    uint64_t bitnum; /* which bit we need to set */
    for (int i = 0; i < (K02 /* Your code here */); ++i) {
        bitnum = b->hash(element, (uint16 t) i) % b->size;
        b->data[bitnum >> 3] |= 1 << (K03 /* Your code here */);
    }
}
```

- K02 = ?
- K03 = ?