Q) Convert the following C code to MIPS:

```
void main()
{
        int s0 = 5;
        int s1 = 10;
        firstFun(s0, s1);
}

void firstFun(int v1, int v2)
{
        int s0 = v1 + v2;
        secondFun(s0);
        secondFun(v1);
        secondFun(v2);
}

int secondFun(int val_1)
{
        return val_1 + val_1;
}
```

```
.text
  main:
    addi $s0, $zero, 5   # s0 = 5
    addi $s1, $zero, 10  # s1 = 10

    add $a0, $zero, $s0  # a0 = s0 (1st arg)
    add $a1, $zero, $s1  # a1 = s1 (2nd arg)

    jal firstFun         # Calling firstFun

    addi $v0, $zero, 10
    syscall

  firstFun:
    addi $sp, $sp, -16
    sw $s0, 0($sp)       # Push s0 to stack
    sw $ra, 4($sp)       # Push ra to stack
    sw $a0, 8($sp)       # Push a0 to stack
    sw $a1, 12($sp)      # Push a1 to stack

    add $s0, $a0, $a1    # We can change s0 now

    add $a0, $zero, $s0  # a0 = s0 (1st arg)
    jal secondFun        # Calling secondFun

    lw $a0, 8($sp)       # Bring a0 from stack
    jal secondFun        # Calling secondFun

    lw $a0, 12($sp)      # Bring a1 from stack
    jal secondFun        # Calling secondFun

    lw $s0, 0($sp)       # Bring s0 from stack
    lw $ra, 4($sp)       # Bring ra from stack
    addi $sp, $sp, 16    # Deallocate space
    jr $ra               # Jump back to caller
  secondFun:
    add $v0, $a0, $a0    # ret val_1 + val_1;
    jr $ra               # Jump back to caller
```

Q) Briefly explain the idea behind backward compatibility when a company designs a new CPU with a new set of instruction set?

The idea behind backward compatibility is to build CPUs that can run old programs ("binaries") as well as programs compiled for the new instruction set

Q) Can you tell whether an instruction is an R-format instruction or not just by looking at the opcode? Briefly explain how.

Yes; R-format instructions have opcode 000000

Q) List all the inputs for a half-adder, and all the inputs for the full-adder.

Half-adder: 2 inputs (input0 & input1). Full-adder: 3 inputs (input0, input1, & carry-in)

Q) Provide the type and hexadecimal representation of the following instruction:

sub $s3, $s2, $s1

Type: R-format
0x02519822

Q) Convert the following MIPS code to binary (or machine language):
100 Begin: beq $s0, $s1, IF
104 addi $s1, $s1, -1
108 j Begin
112 IF: addi $s0, $s0, 1

000100 10000 10001 0000000000000010
001000 10001 10001 1111111111111111
000010 00000000000000000000011001
001000 10000 10000 0000000000000001

Q) Provide the type and assembly language instruction for the following binary value:

a)

0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

Type: R-fomat: add $s0, $s0, $s0

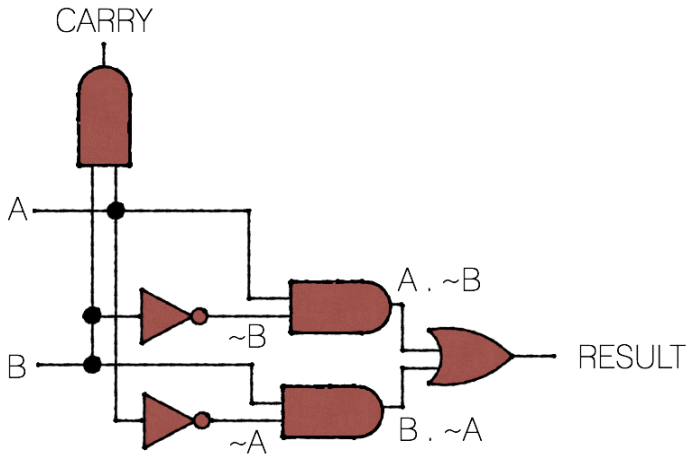
b)

0	0	1	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1

Type: I-format: addi $s0, $t1, 9

Q) Fill the table below for the 'result' and 'Carry Out' outputs of a half adder, then draw the gates diagram to produce 'result' only (no carry out) — You can only use 'and', 'or', and 'not' gates.

Input 1	Input 2	Result	Carry Out
1	1	0	1
1	0	1	0
0	1	1	0
0	0	0	0



Q) Fill the table below for a 4-1 multiplexer, mark 'X' for the "Don't cares" cells;

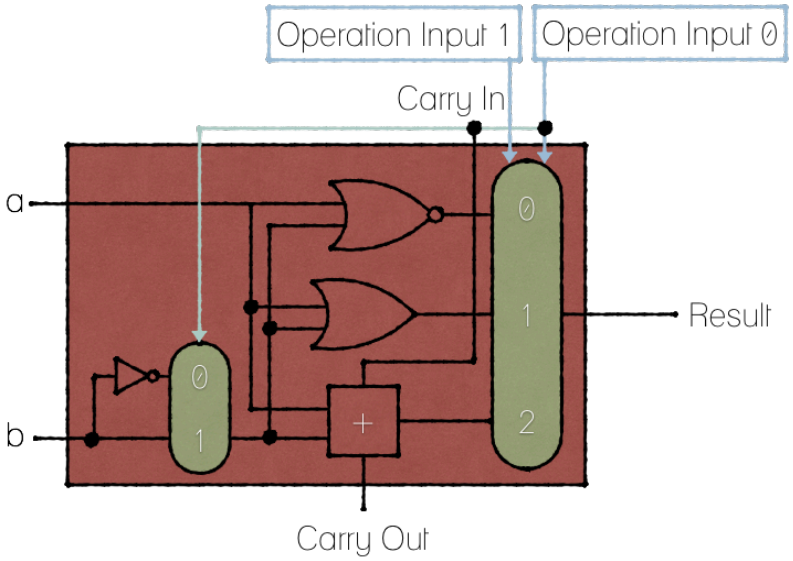
Input 0	Input 1	Input 2	Input 3	Control Signal 0	Control Signal 1	Output
1	X	X	X	0	0	1
1	1	0	1	0	1	0
0	0	0	1	1	1	1
0	X	X	X	0	0	0
X	1	X	0	1	1	0

Q) What's the boolean expression that matches the behavior of a 4-1 Multiplexer — Truth table will be too long, think of another way (use descriptive names, e.g. input0)

Input0 . ~Control0 . ~Control1 +
Input1 . Control0 . ~Control1 +
Input2 . ~Control0 . Control1 +
Input3 . Control0 . Control1

Q) Fill the table below according to the input and the circuit below;

a	b	Operation Input 0	Operation Input 1	Result
1	1	0	0	0
1	1	1	0	1
1	1	0	1	1



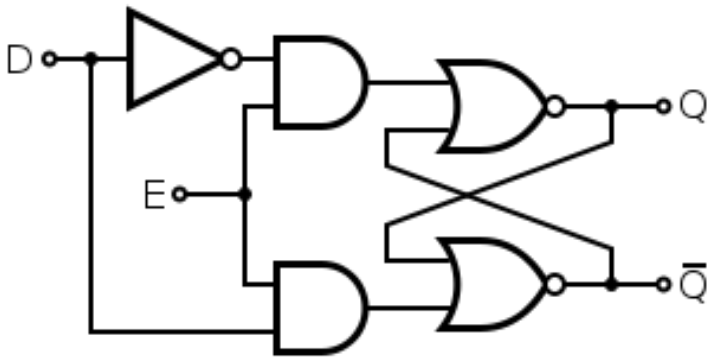
Q) Simplify the following expression: ~(~A + B + ~C) . ~(A + ~B + C)

~(~A + B + ~C) . ~(A + ~B + C)
= (A . ~B . C) . (~A . B . ~C)
= A . ~B . C . ~A . B . ~C
= False

Q) List the five stages (in hardware) for MIPS instruction execution.

1) Instruction Fetch 2) Instruction Decode / Register Read 3) ALU 4) Access Memory 5) Write to Register

Q) Fill the table (below) according to the circuit provided:



	D	E	Q
1st Set of Values	1	1	1
2nd Set of Values	1	0	1 (Repeat)
3rd Set of Values	0	0	1 (Repeat)
4th Set of Values	0	1	0
5th Set of Values	1	1	1

Q) List the stages that are active in each following instruction:

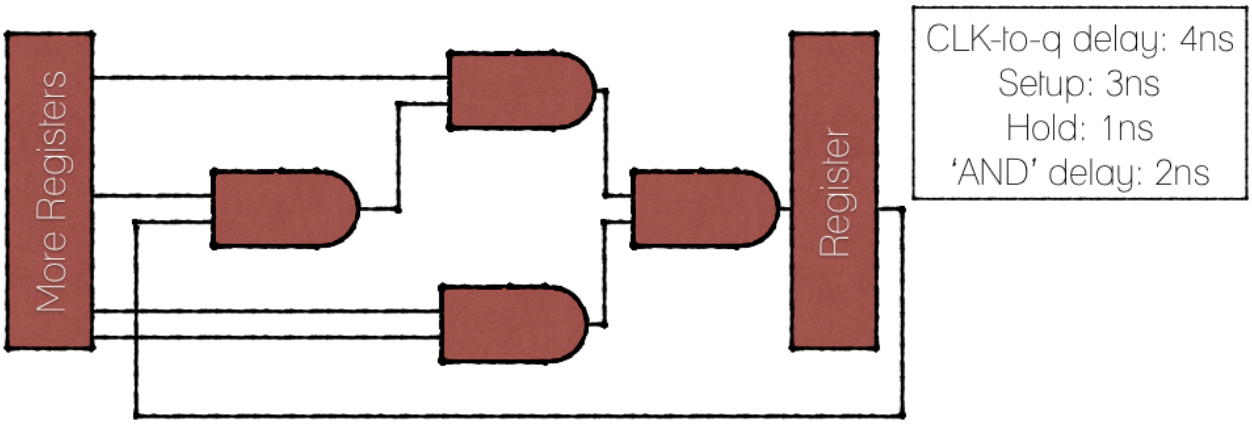
a) Store word
1) Instruction Fetch 2) Decode & Register Read 3) ALU 4) Memory Write

b) Load word
1) Instruction Fetch 2) Decode & Register Read 3) ALU 4) Memory Read 5) Register Write

c) Add Register
1) Instruction Fetch 2) Decode & Register Read 3) ALU 5) Register Write

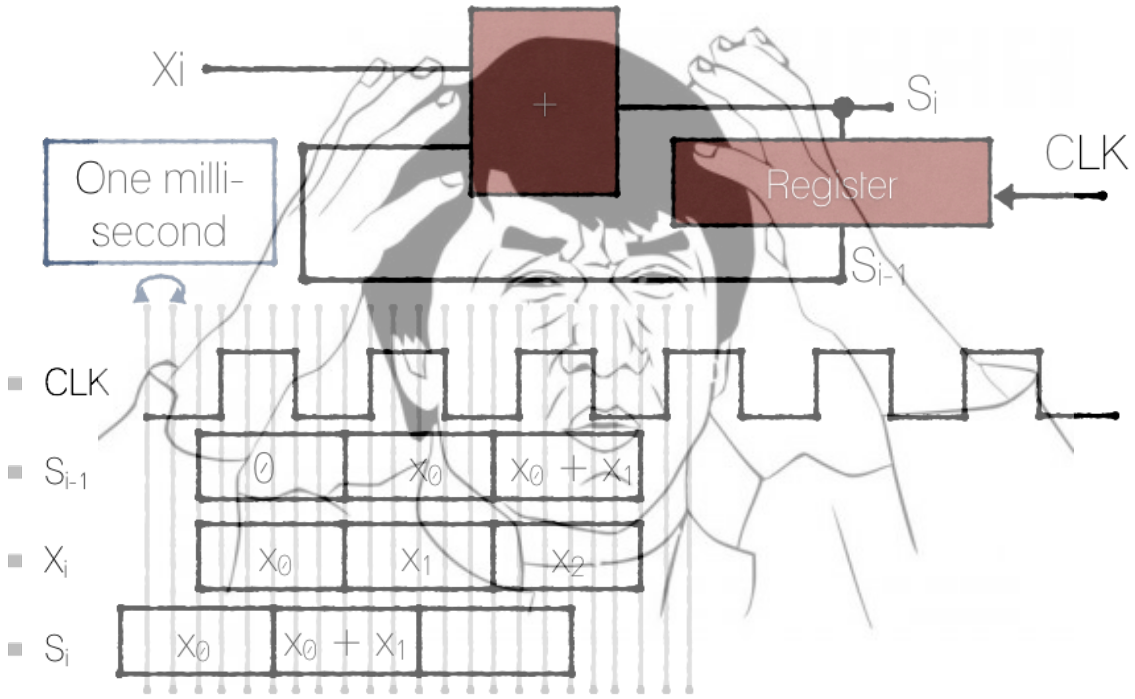
d) Set Less than
1) Instruction Fetch 2) Decode & Register Read 3) ALU 5) Register Write

Q) What's the total delay and maximum clock frequency for the circuit below?

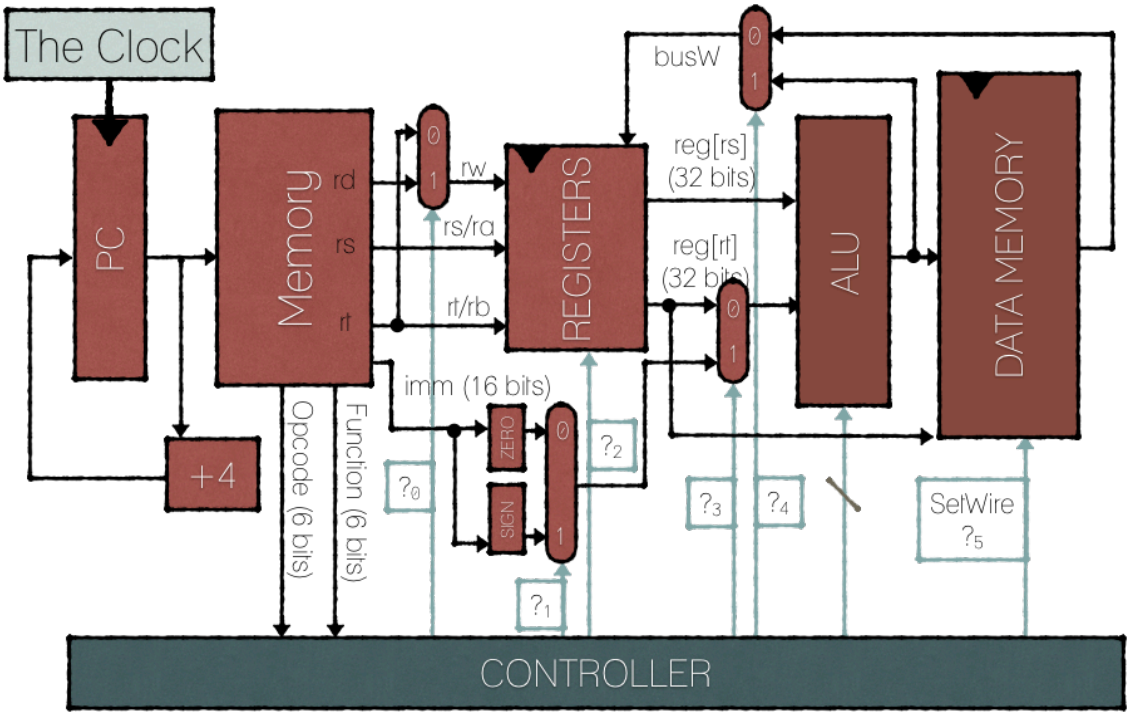


Total delay is 6 + 3 + 4 = 13 nanoseconds
Maximum clock frequency = 1/13 Gigahertz

Q) What's the combinational logic delay and the CLK-to-Q Delay for the diagram below?



Q) Based on the following data-path (MIPS data-path), fill the table below (use 'X' for "don't cares"):



	$?_0$	$?_1$	$?_2$	$?_3$	$?_4$	$?_5$
add	1	X	1	0	1	0
ori	0	0	1	1	1	0
lw	0	1	1	1	0	0
sub	1	X	1	0	1	0
sw	X	1	0	1	X	1