Q1. Write a MIPS assembly program that is equivalent to the following C program. Assume that x is stored in register $t0, y in $t1, and w in $t3.

```
int32_t x, y;
if (x==y) {
w = w + 50;
}
```

Q1 – Solution:

```
bne $t0, $t1, EXIT

addi $t3, $t3, 50

EXIT: …
```

Q2 - Read the below MIPS program (left) and the memory state (right), and answer the following question. Initial values of the registers: \$s3=0, \$s5=331, \$s6=1000.

What would an equivalent C program be? Finish the code below and briefly explain what the code is doing. Assume that the array Array starts from memory address 1000.

Loop:	sll \$t1, \$s3, 2
	add \$t1, \$t1, \$s6
	lw \$t0, 0(\$t1)
	bne \$t0, \$s5, Exit
	addi \$s3, \$s3, 1
	j Loop
Exit:	…

Address	Value
1000	331
1004	331
1008	431
1012	431

```
Code:

int32_t Array[4]={331, 331, 431, 431};
int32_t i=0;
```

Q2 – Solution:

```
int32_t Array[4]={331, 331, 431, 431};
int32_t i = 0;

While (Array[i] == 331){

i++;

}
```

Q3 - Provide the type, assembly language instruction, and binary representation of the instruction described by the following MIPS fields:

```
op = 0x23, rs = 1, rt = 2, const = 0x4
```

Q3 – Solution:

```
I-type: lw $v0, 4($at),
```

```
0x8C220004
```

Q4 – Write a Verilog code that is supposed to implement the function

z = (x ^ y) & x.

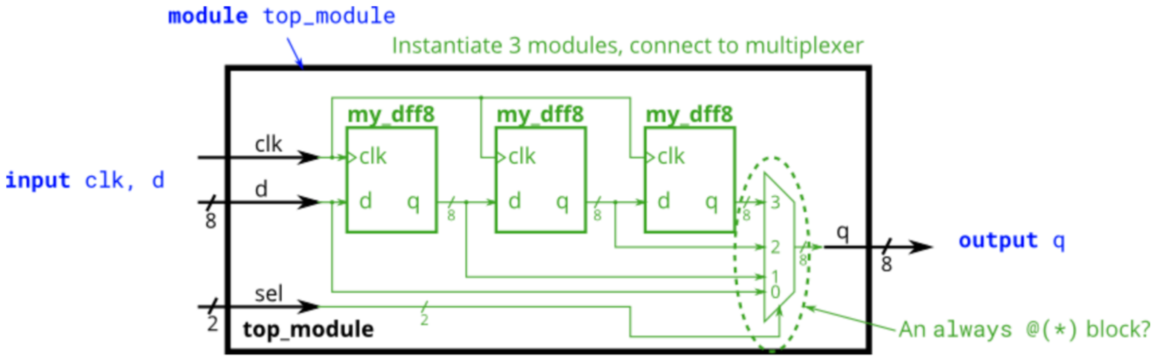
Q3 – Solution:

```
module top_module (input x, input y, output z);
       assign z = (x ^ y) & x;
endmodule
```

Q5 - You are given a module `my_dff8` with two inputs and one output. Instantiate three of them, then chain them together to make a 8-bit wide shift register of length 3. In addition, create a 4-to-1 multiplexer (not provided) that chooses what to output depending on `sel[1:0]`: The value at the input d, after the first, after the second, or after the third `my_dff8` . (Essentially, `sel` selects how many cycles to delay the input, from zero to three clock cycles.)

The module provided to you is: `module my_dff8 (input clk, input [7:0] d, output [7:0] q );`

The multiplexer is not provided. One possible way to write one is inside an `always` block with a `case` statement inside.



Q4- Solution:

```
module top_module (
     input clk,
     input [7:0] d,
     input [1:0] sel,
     output reg [7:0] q
);
     wire [7:0] o1, o2, o3;        // output of each my_dff8

// Instantiate three my_dff8s
     my_dff8 d1 ( clk, d, o1 );
     my_dff8 d2 ( clk, o1, o2 );
     my_dff8 d3 ( clk, o2, o3 );

     // This is one way to make a 4-to-1 multiplexer
     always @(*)         // Combinational always block
          case(sel)
               2'h0: q = d;
               2'h1: q = o1;
               2'h2: q = o2;
               2'h3: q = o3;
          endcase

endmodule
```