Name : salma soliman
Id : 900182325
Lab3 Monday 3:30pm

Experiment (1):

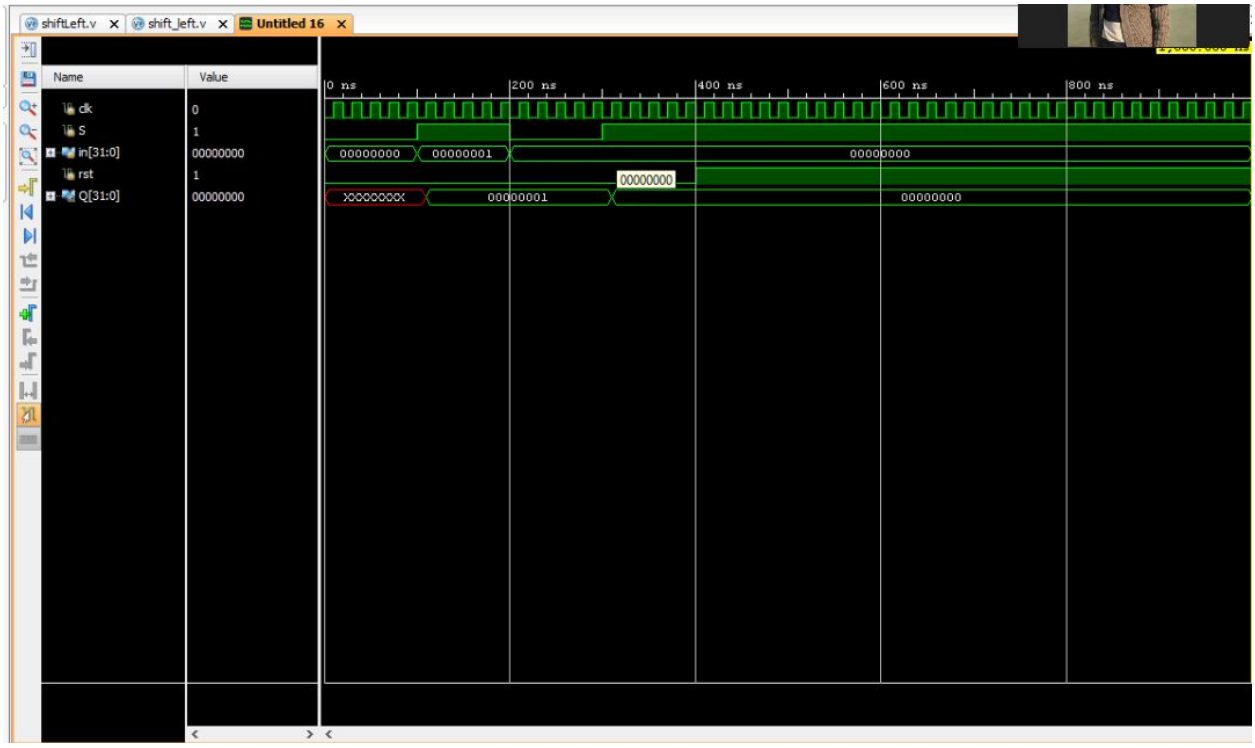
Summary :

In this experiment , we implemented a register module that represents one register. This register is a 32-bit register.

Steps:

1. Created a D flip flop module
2. Created a multiplexer module
3. Created a single bit module in which we are calling the two previous modules
4. Created a 32-bit register in which we are calling the previous module 32 times
5. Developed a testbench for this module

Output:



Interpretation:

Whenever the clock is working and If the selector is 0,the output will be the value of the previous output. If the selector is 1, the output will be the current value in the register.

Proving that on simulation:

1. When s=0 , the previous output was unknown. When s =1 && n = 1 , the output will be 1.
2. When s = 0, the previous output was one so the current output is 1.
3. When s=1 ,the output will be the current input which is zero.

Experiment (2):

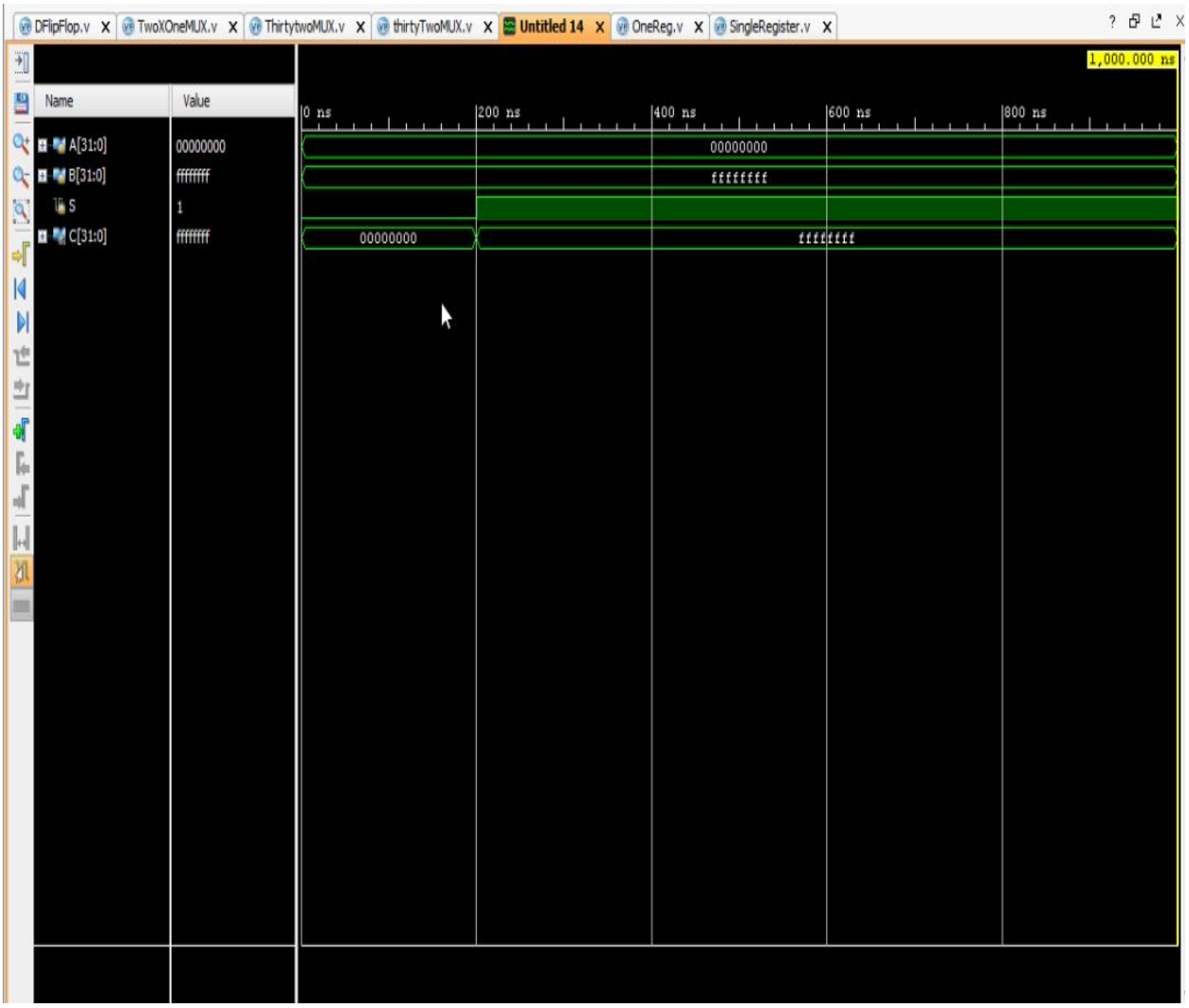
Summary :

We implemented a two to one multiplexer module.

Steps:

1. Created a single multiplexer module
2. Created 32 multiplexers module using the generate loop

Output:



Interpretation:

When the selector = 0, the output of a mux will be the first input. When the selector =1 , the output of a mux will be the second input.

Proving that on simulation:

1. When s = 0, the output should be A and C=A=0.
2. When s= 1, the output should be B and C=B.

Experiment (3):

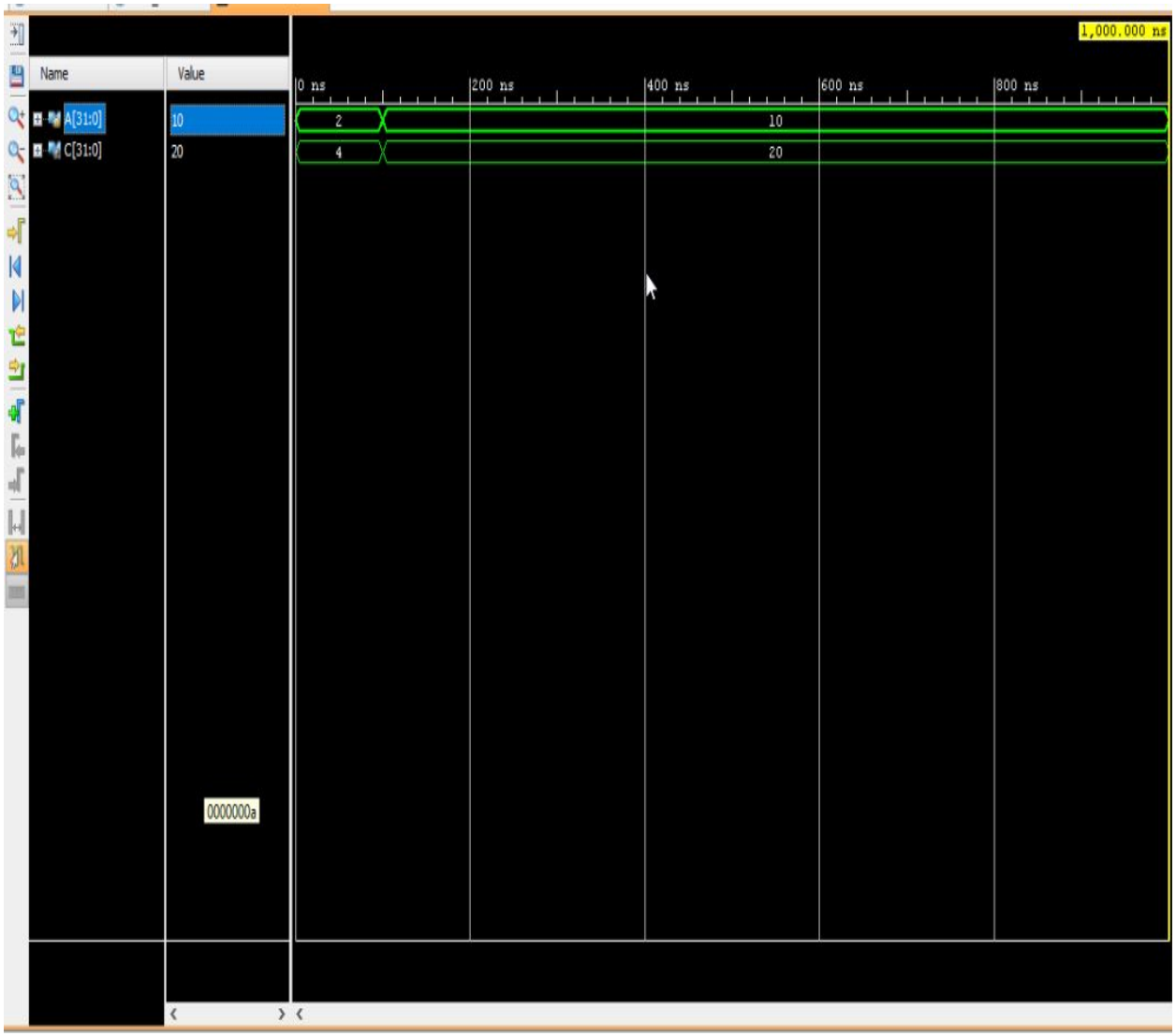
Summary :

We implemented a shift left module.

Steps:

1. Using the assignment operator and the concatenation, we dropped the M.S.B and added a zero in the L.S.B.

Output:



Interpretation:

Shifting left means multiplying the input by 2.

Proving that on simulation:

1. When the input is 2 , the output is 4
2. When the input is 10 , the output is 20

Experiment (4):

Summary :

In the assembly language, a single instruction is represented by a single 32 bit binary number. Some instructions have an immediate number which will be represented in a 12-bit from the 32-bit binary number that represents this particular instruction.However, this immediate number can not be found in a certain arrangement of bits. It is found in different places in each instruction type. We created an immediate generator module that can extract this 12 bit from each instruction according to its type.

Steps:

1. We differentiated the instruction according to bit 6 and 5. If bit 6 and 5 ==0 , then this is a load LW instruction. If bit 6 =0 and bit 5 =1 , then this is a SW instruction. If bit 6 == 1 , then it is a BEQ instruction.
2. Created the extension of the immediate number by checking whether the bit 31 is one or zero. if it is one we sign extend it with one. If it is zero we sign extend it with zeros. The extension is done with 20 bits as we need the number to be a 32 bit and we already have a 12 bit.
3. Used concatenation to get the whole signed immediate number

Output:

Interpretation:

We used the three provided instructions in the lab manual and checked the output.

Proving that on simulation:

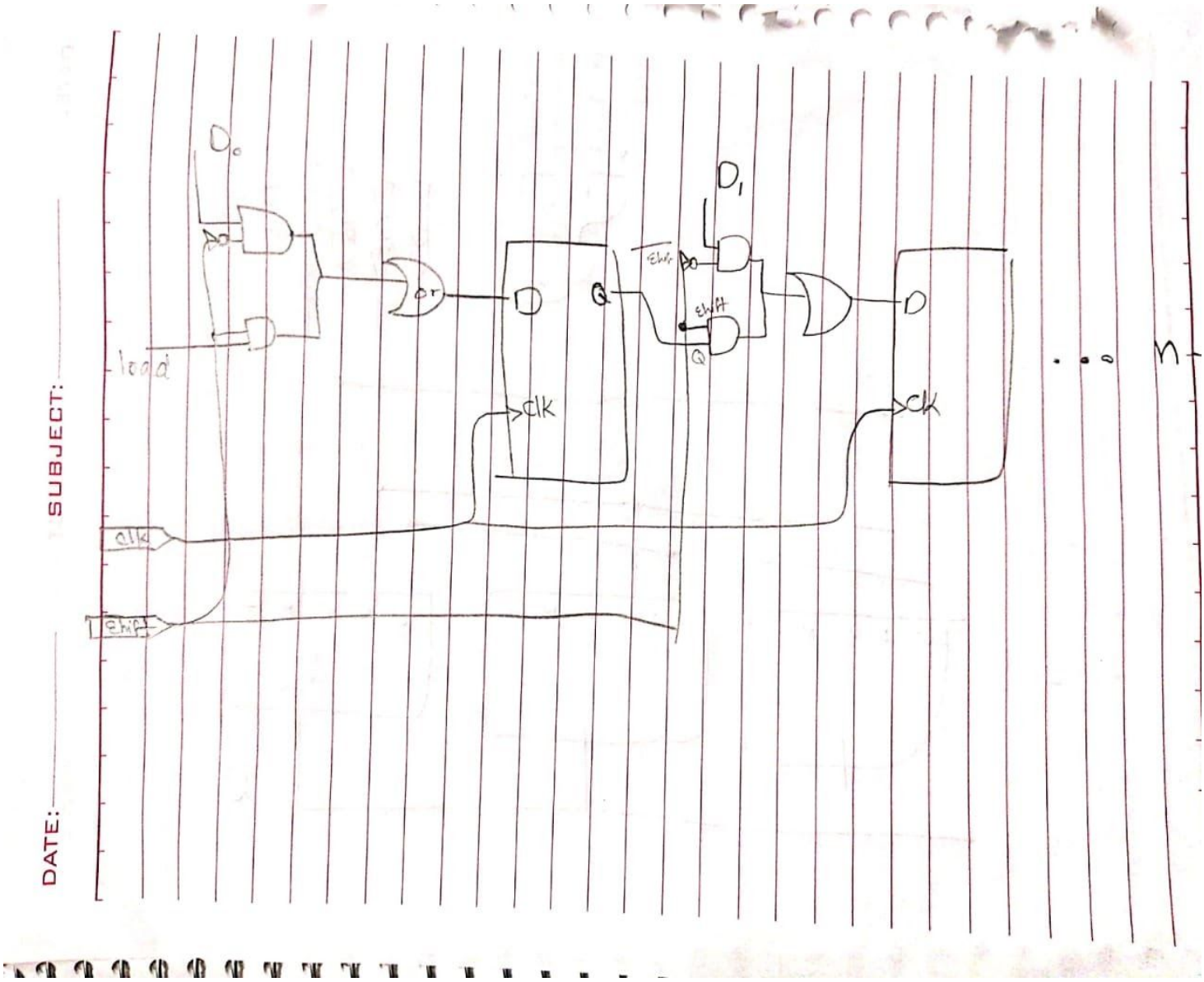
1. If the instruction is 32'h4D62A303 , the immediate should be 1238
2. If the instruction is 32'hDCA7AF23 , the immediate should be -546
3. If the instruction is 32'h18E18F63 , the immediate should be 207

Experiment (5)

shift	load	Output (D)
0	0	0 , no change
0	1	1 , load
1	0	1 , shift
1	1	1 , load

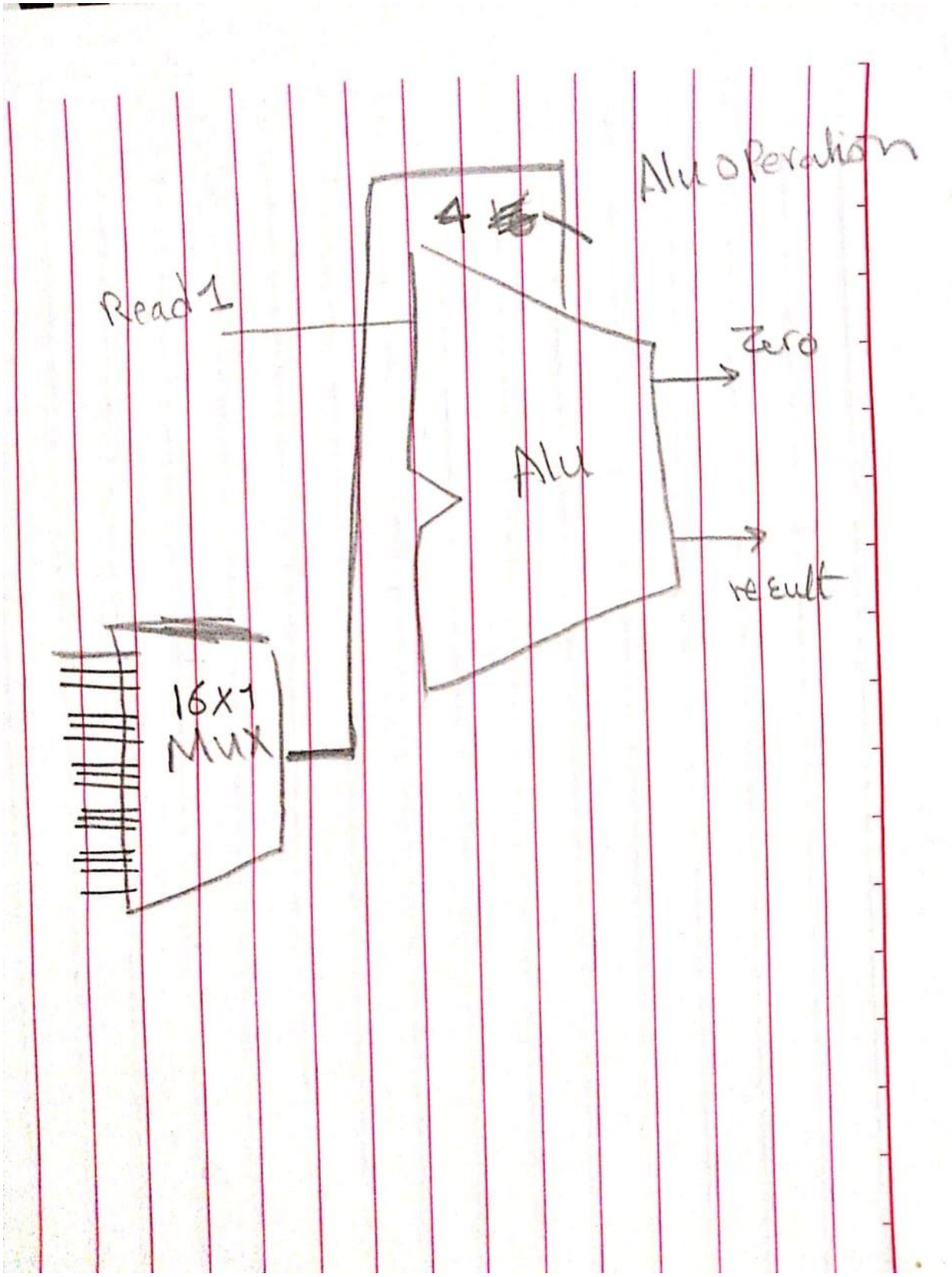
I tried several times to run this code and other versions of it on vivado but it was so slow and i restarted it. When i did that , it didn't allow me to open a new project or add any source files.I did not have enough time to investigate that so the coming code is not tested but based on what I understood.the same for the schematic.(the full code is provided in lab3 file experiment 5).

```
module shiftReg (input CLK, rst, shift, load, [n-1:0] D, nbit,output reg [n-1:0] Q);


genvar i;
generate
for (i=0;i<=31;i=i+1)
begin
        onebitReg mod1 (   in[i],  S, clk,  rst,   Q[i]);
        end
  endgenerate


always @(posedge CLK)
begin
if (rst)
Q <= 0;
else
 if (load | (load==1 && shift==1))
Q <= D;
else if (shift && load==0)
Q <= { Q[n-2:0], nbit };                    //nbit = the number of shifting operation
Endendmodule
```



Here, I modified the components of the MUX whose output is the input of the Dflipflop. I added the load signal in the first FF and the shift signal in the following FFs. To Trace it:

1. Case (shift=1;load=0): Q=0 → and the Q shifted to the next FF
2. Case (shift=0;load=1): D0 entered → Q=D0;Q isn't transferred to the net FF → D1 entered as well → Q2=D1, so loading occurred.
3. Case (shift=0;load=0): Q1=D0 → Q2=D1 → remains the same so NO change status occurred
4. Case (shift=1;load=1): Q1=1 →So loading occurred



Here , I used 16 to 1 MUX to select one number of 4 bits to send it to the ALU.