16.08.2018

MICROPROCESSORS

HOMEWORK 4

STUDENTS :

BÜŞRA YAŞAR	**- 150114063**
FEYZANUR KARAKOÇ	**- 150116502**
HALE ŞAHİN	**- 150116841**

ASSISTANT TEACHER :

ETKİ GÜR

INSTRUCTOR :

ALPER ŞİŞMAN

Firstly, we design a microprocessor that can do some work like addition, shifting etc. We only used gates to implement it. We implement by using verilog code and we divide some units into modules to make it easier.
Our processor, at the end, has 7 base units : Debounce, Program Counter, ROM, Instruction Register, ALU, RAM and the Main Control.
We were implemented other 6 units except RAM in the third homework. Now we have RAM unit and we write some data in some address places in RAM and we read them to use them in the operations. We were wanted to execute these instructions :

MOVA 25;	ACC = 25
STORE AA;	<AA> =25
MOVA 45;	ACC = 45
STORE AB;	<AB> =45
LOAD AA;	ACC = <AA> =25
ADDR AB;	ACC = ACC + <AB> = 25+ 45
STORE AC;	<AC> =70
SHFTRR AC;	ACC = <AC> / 2 = 35

Then we converted these assembly code to machine codes, in our machine code our instruction must has 12 bits,1 bit for instruction type 3 for ALU opcode, 8 bit for result. According to this, instructions will be like this;

Instruction 1 = 12'b010000011001;

Immediate type, mov function 100, immediate value is 00011001, Acc will have at the end this number as result output,25.

Instruction 2 = 12'b1111xxxxxx00;

Register type, no function for ALU, used reserved value 111, address number is 00 means for AA register.

Instruction 3 = 12'b010000101101;

Immediate type, mov function 100, immediate value is 00101101, Acc will have at the end this number as result output,45.

Instruction 4 = 12'b1111xxxxxx01;

Register type, no function for ALU, used reserved value 111, address number is 01 means for AB register.

Instruction 5 = 12'b1100xxxxxx00;

Register type, mov(load) function for ALU 100, address number is 00 means for AA register.

Instruction 6 = 12'b1000xxxxxx01;

Register type, add function in ALU 000, It will add the previous acc value to the new input value from RAM address 01, AB register.

Instruction 7 = 12'b1111xxxxxx10;

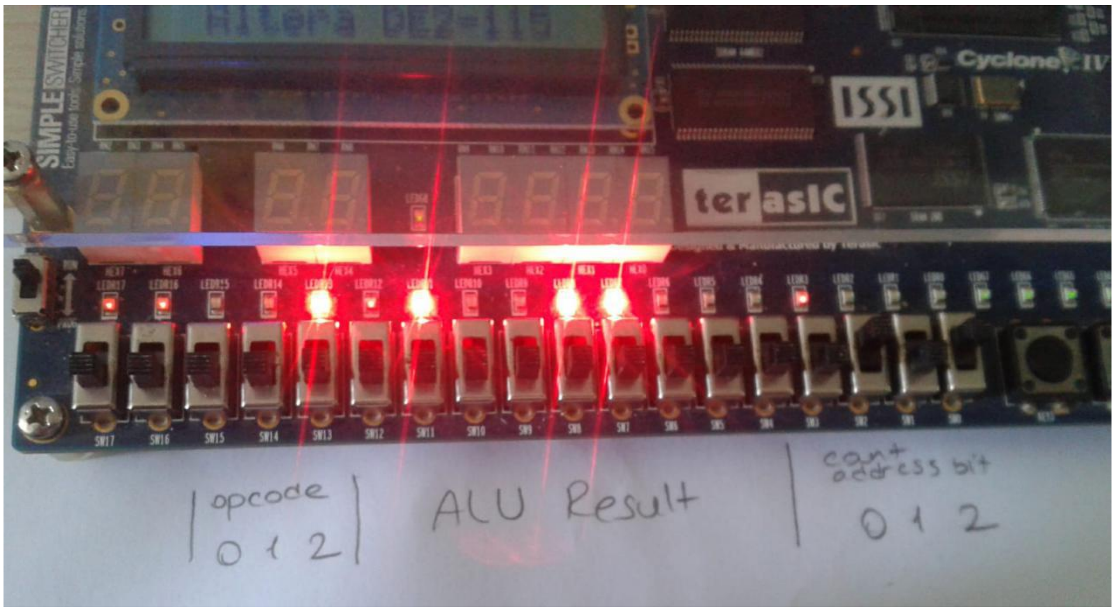
Register type, no function for ALU, used reserved value 111, address number is 10 means for AC register.

Instruction 8 = 12'b1110xxxxxx10;

Register type, shift right operation for ALU 110, shifting value is coming from RAM address 10, AC register. Result is in the accumulator.

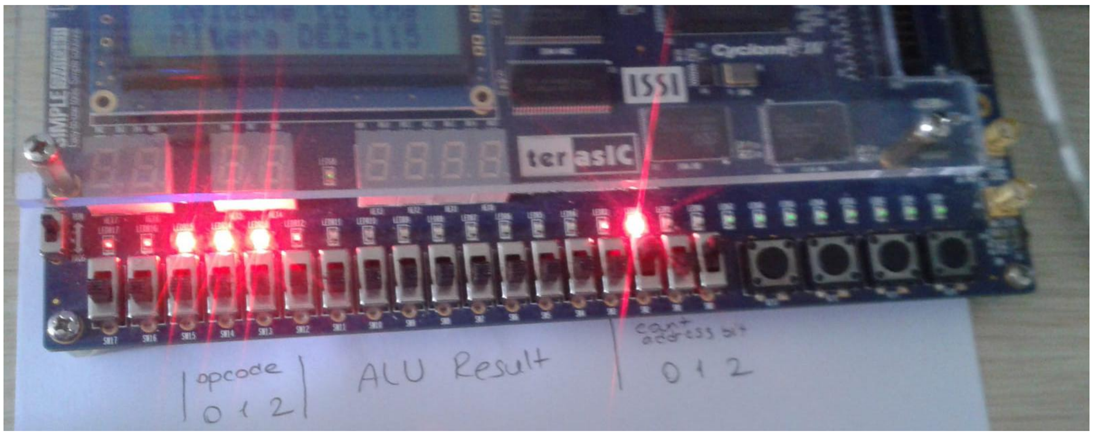
When we run our instructions in the fpga, we use some input buttons. We didnot prefer to implement another contol unit which will has states to execute them consecutive, like first clk is fetch, second clk is decode, third clk is if there is r/w, and so on. Instead we give these signals from outside, to control the flow of the operations by hand. Our inputs were clk, fetch, decode, execute, r/w for RAM, reset for PC, Our outputs were ACC result, to understand which instruction are reading in te ROM, we used ROM address as output, To understand thaht decode is working we used opcode as output. We took pictures for each instruction after all steps executed. We sequentially give the fpga fetch signal with clk, then decode signal, then if read is needed, we give read signal, then we gave execute signal with clk for ALU, at the and if instruction is store, we need write signal as r/w is 0. We run these steps for 8 instructions as we explain below.

Instruction 0:



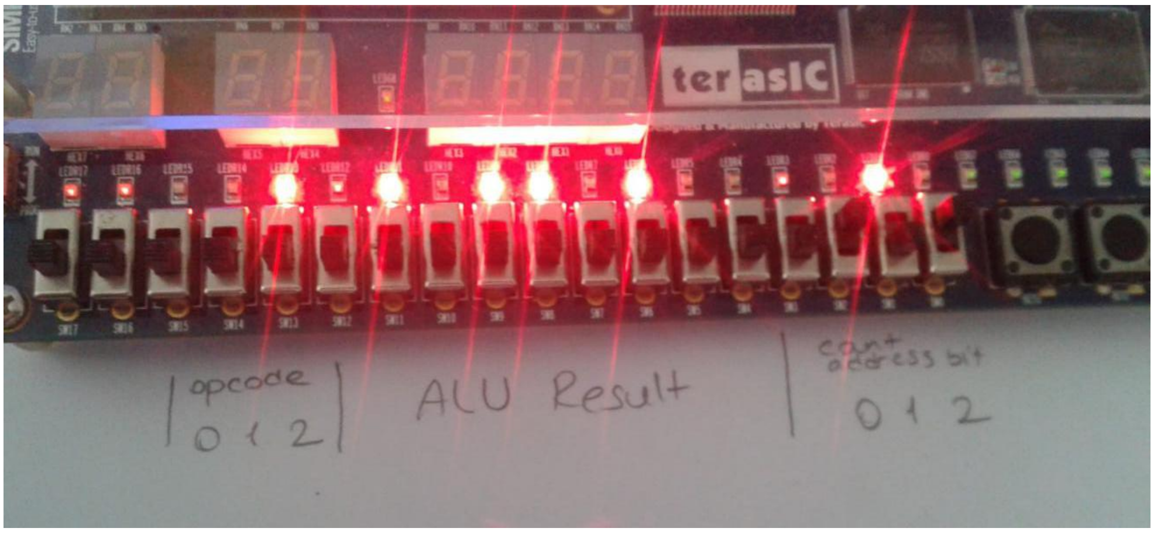
Here we can see after all signals executed, instruction is worked properly, firstly we did fetch, in LED0,LED1,LED2 , which labeled as count says it fetched 000, first instruction, then when we give decode signal signals of opcode which is labeled as opcode are glowing as 100. In the middle we have 8 bit acc result, which is need to be 25, 00011001 which is true we need to read from right to the left as like others.

Instruction 1 :



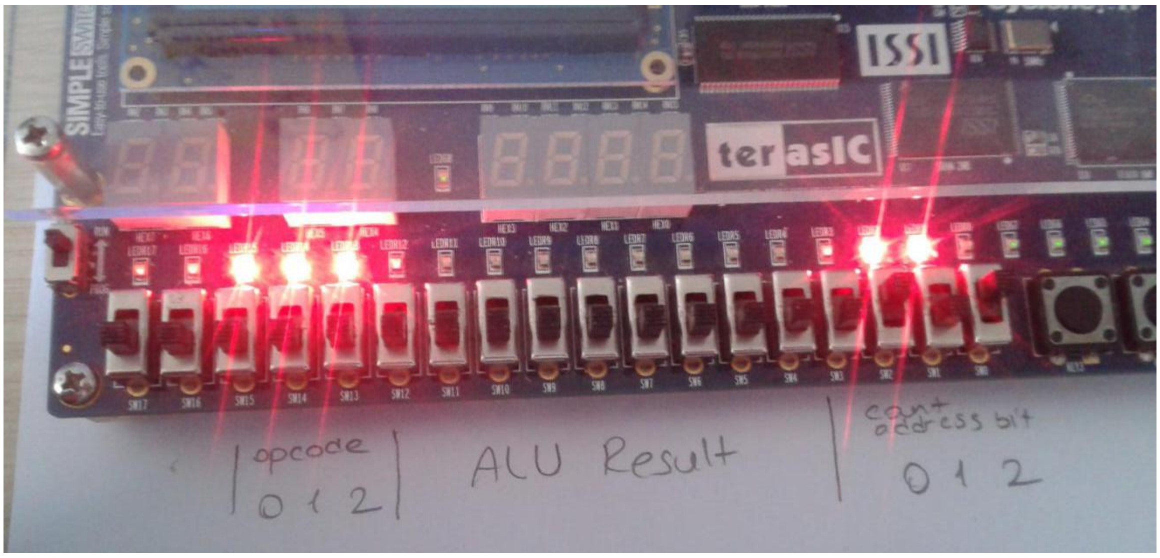
In the second instruction, we can see after all signals executed, instruction is worked properly, firstly we did fetch, it fetched 001, first instruction, then when we give decode signal signals of opcode are glowing as 111. Acc result is not glowing because it is store operation it took acc result and write it to the 00 address in RAM.

Instruction 2 :



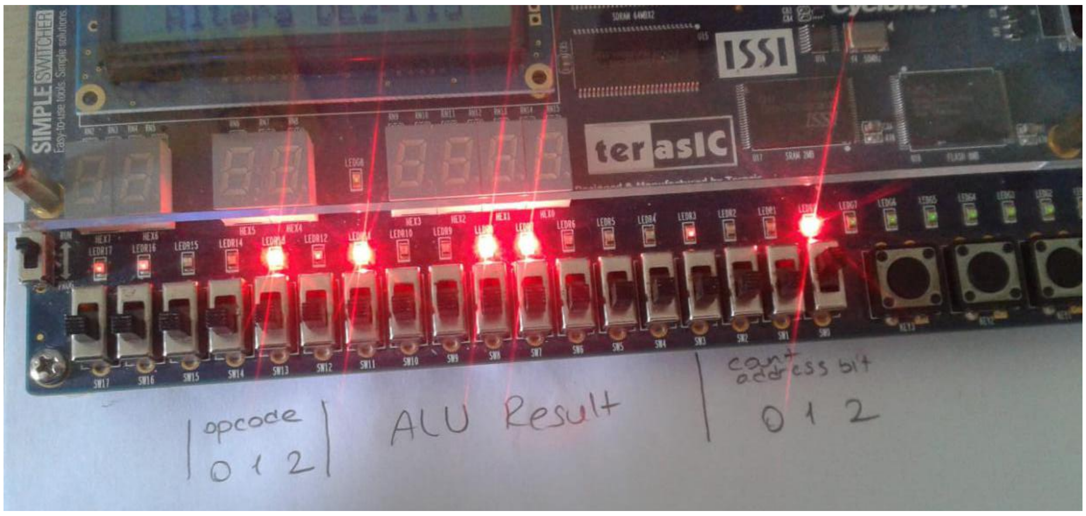
At this instruction we upload accumulator as a new immediate value, its second instruction as address value pointed 010, its opcode is move, and acc is 00101101 as 45.

Instruction 3 :



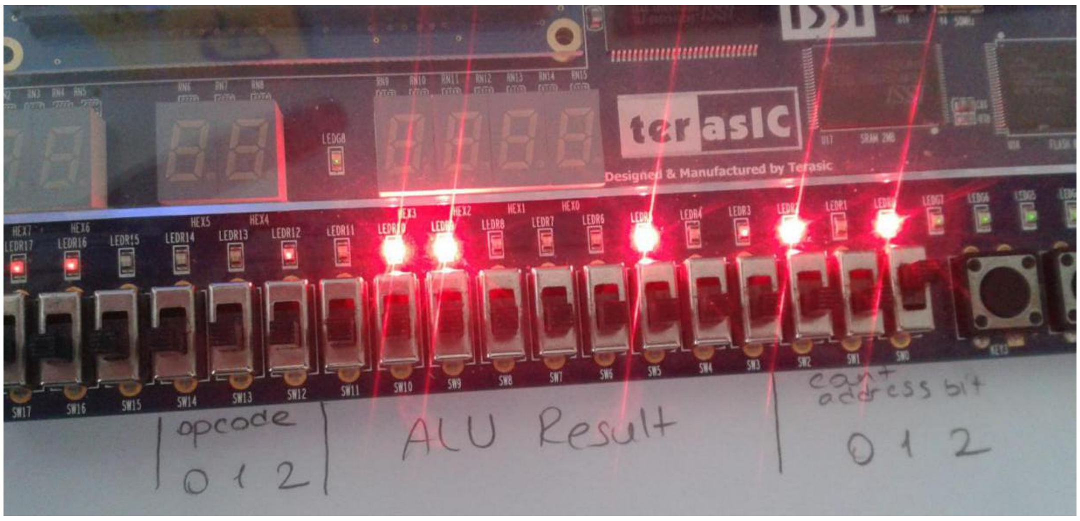
This instruction is for storing the acc result in the 01 RAM address, its address count value is 011, its opcode is 111.

Instruction 4 :



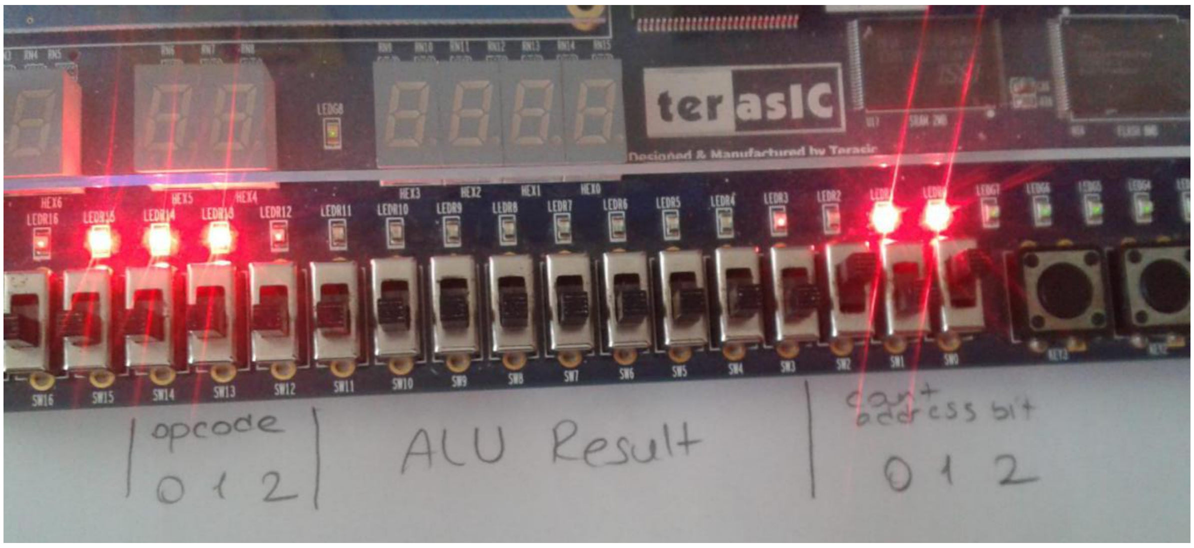
Here we have the 4th instruction 100, its opcode is 100, move function, but its R type move that means it gets ALU input value from RAM data out, as we give read signal. It loaded the value inside RAM address 00, means AA register, it was 25, and at the result pins we can see it, 00011001.

Instruction 5 :



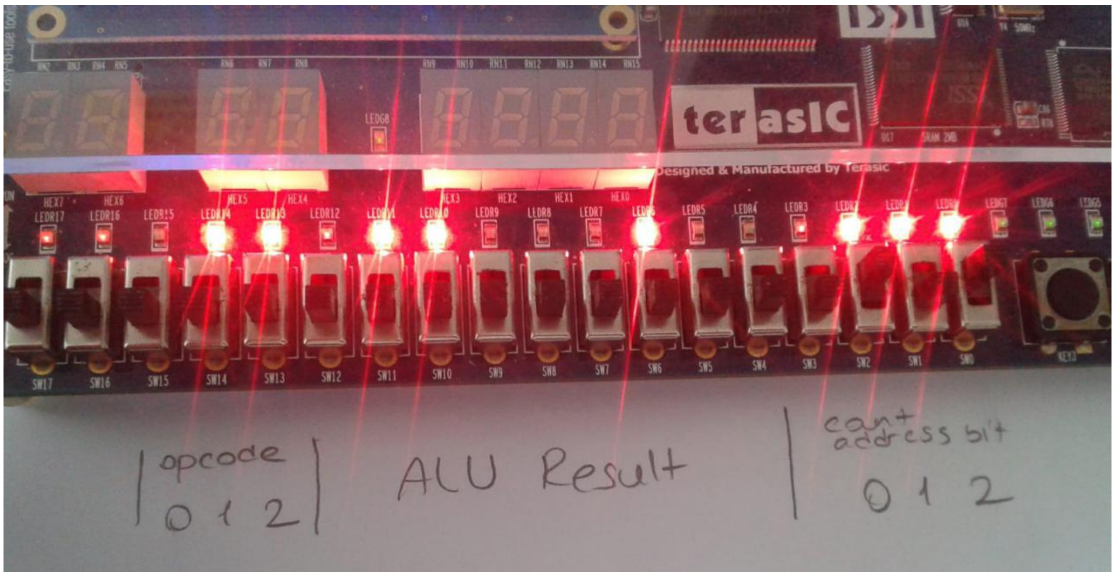
In the 5th instruction we can see it is fetched 5th 101 in address pins, we can see it decodes properly in opcode is 000 Add for ALU, we can see the result as add function is true, 01000110, as 70.

Instruction 6 :



Here we can see that fetch operation is true when we gave posedge clk, it is 110, 6th operation, after that we can see the opcode is true, then we understand decode worked properly, 111 reserved value for store operation, we see the result not glowing because its making write operation in RAM.

Instruction 7 :



At the last instruction we fetched 111 address from ROM, and when we decode it, machine understands its a shift right operation, 110, and it does the operation to the given value, and this value is coming from RAM address 10, because this is register type instruction. The mux before ALU input decides it, according to the instruction type bit. We give read signal before execute signal and then we read the result as 35 in the ACC.

What we understand from all this?

We understand the connection of the ALU, RAM and ROM units and what are they for, and how they work. In ALU we take one input from outside its determined by a mux which uses instruction type bit as selection bit, its inputs are 8 bit data from RAM, and 8 bit data from Instruction register as immediate data. If it is I type instruction then its gonna choose the immediate value, otherwise it chooses RAM data. The second input of ALU coming from Accumulator register which keeps ALU result in the memory when execute signal is on. If the operation is add, and, sub , or then it will take one input and make the operation between that input and the previous accumulator result. We make accumulator register as another module in verilog, cause it needs to write the ALU output when execute signal is set and clock is on posedge. In ALU there are seven operations adder, subtractor, and, or, mov, shift left,s shift right. We designed all of them with gates. We send inputs to all and take outputs, these outputs are going to 8 to 1 with selection input from Instruction register, opcode 3 bits. Only one 8 bit output coming out of ALU. Then we determine some flags as zero flag, negativity flag, carry and overflow flags.

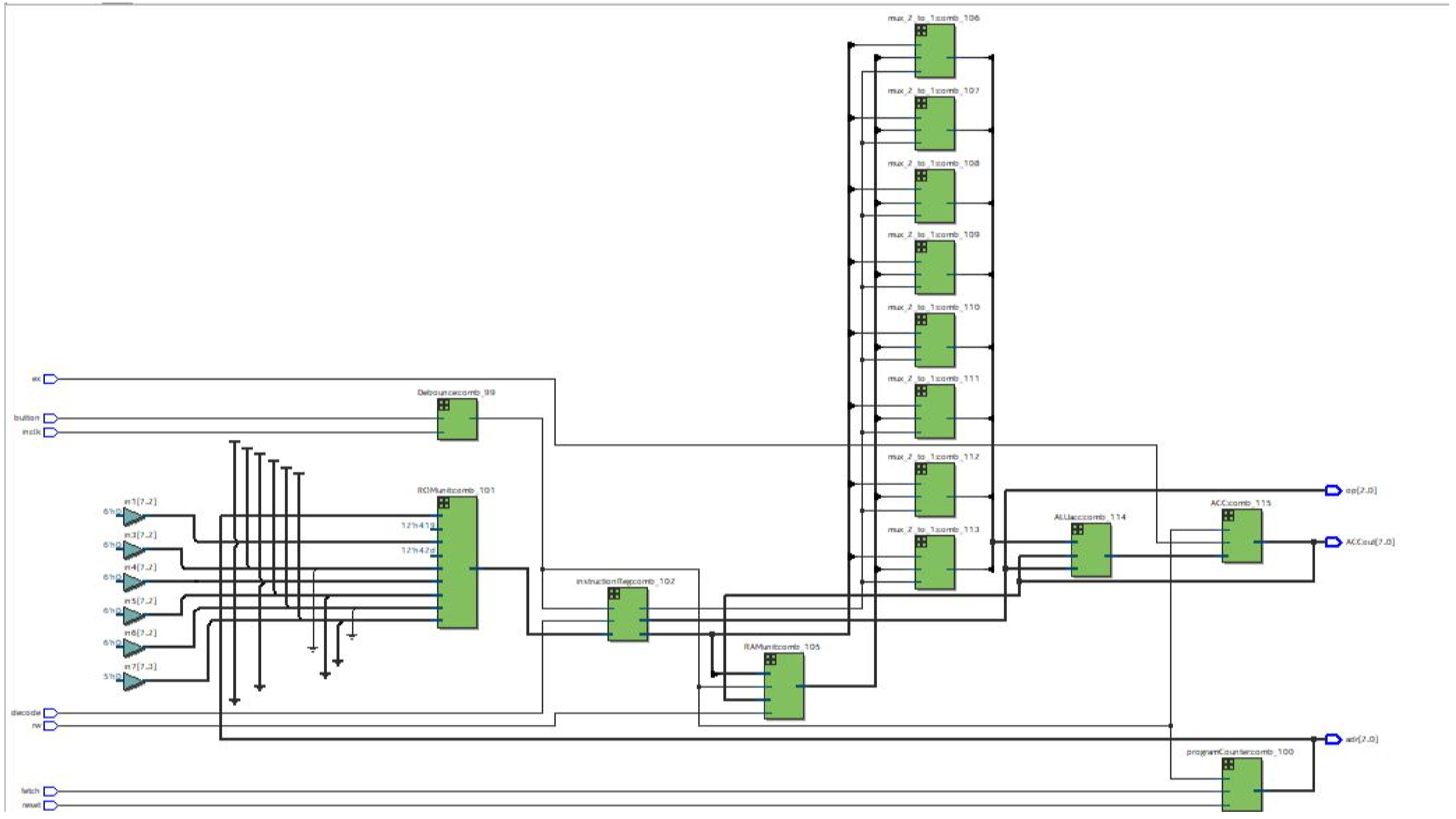
The program starts from program counter. It is counter which counts three bit number consecutive. It works when fetch signal and posedge clk are set. It produces 3 bit number which enters ROM as address place number.

ROM unit is Read Only Memory, is used for Instruction Memory, we embedded inside its addresses some instruction we write above. It will read them when it take its address number. It does not use any clock or signal, it directly reads. It gives an output for the Instruction register, which will decode and divide instruction. Its output is 12 bit for our project. The reading process is done by choosing the address space. We first determined addresses as s1,s2,s3.. for example we make s1 equal to address bit 000, like ~address[2] & ~address[1] & ~address[0], when we take not, it means zero, and when we add it means s1 will be set only if all bits in address is 0. And the embedded instructions are selected by these s places first instruction s1, second s2, ...etc.

When the right instruction comes into the Instruction register, It will make decode if decode signal is set and the clk is on posedge. Decode works with flip flops it writes output opcode value the value from instruction input bits 11 to 9, the instruction type bit is 12th bit, and to 8 to 1 bits are for immediate value for ALU or address value for RAM. It assigned these outputs from the input instruction.

If we talk about RAM, it takes 8 bit data input, 2 bit address input, clk and r/w signal, it give output 8 bit data. Inside of RAM we used binary cells which is composed of flip flops and controls by r/w signal and select bit. We used different modules for binary cell and flip flops. RAM is used for two mission, to write down and keep data, and to read data to use in operations. When we used for RAM to write operation, We first do fetch, decode, execute, and then execute signal will make the result in accumulator, RAM will take the result from accumulator as data in, and when it call binary cells, cells choose write operation as we gave in signal, and cell choose where to write according to the select bit. Select bits are assigned just like in ROM, if address bits are 00 choose s1, if 01 choose s2, and goes on like that. Then flip flops in the binary cell objects makes the write operation to the those places. Then if were going to do read operation, these flip flops will keep the data of course. When we send r/w signal as read, binary cells choose to read the data inside flip flops. And make the RAM output is that data. We used 2 address bit for RAM, we had 8 bit for the addressing in the instruction, thats why we only used 2 bits of them as 0 and 1 bit of the 8 bit value.

Here our RTL design;


ex
button
mclk
Debouncecomb_99
ROMunitcomb_101
in1[7..2]
in3[7..2]
in4[7..2]
in5[7..2]
in6[7..2]
in7[7..3]
6'h0
5'h0
12'h419
12'h42d
instructionRegcomb_102
RAMunitcomb_105
mux_2_to_1:comb_106
mux_2_to_1:comb_107
mux_2_to_1:comb_108
mux_2_to_1:comb_109
mux_2_to_1:comb_110
mux_2_to_1:comb_111
mux_2_to_1:comb_112
mux_2_to_1:comb_113
ALUacccomb_114
ACC:comb_115
programCountercomb_100
decode
rw
fetch
reset
op[2..0]
ACCout[7..0]
adr[2..0]