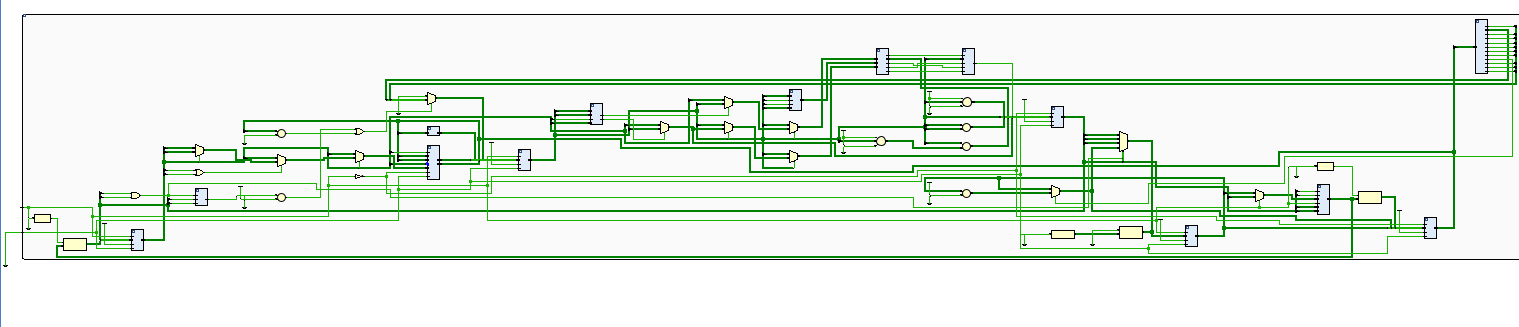
Project Report Computer Architecture

FemtoRV32 Project

Names: Omar Saqr:900223343-Aabed Elghadban: 900223106- Noor Emam: 900222081

RV32 Schematic:



The FemRV32 processor has three stages, the IF/ID stage, the EX/MEM stage, and the WB stage. Unlike traditional processors, the FemRV32 fully utilizes all periods of the clock cycle in order to allow for faster execution. However this comes at the expense of extra control logic in order to control which instruction/dataout is coming out from the data memory, gating the inputs to the registers, and so much more. However for the sake of simplicity, we will not go over this, this report will center around showing that our program works as expected.

1- 00000000000000000010000010000011 // lw x1, 0(x0)



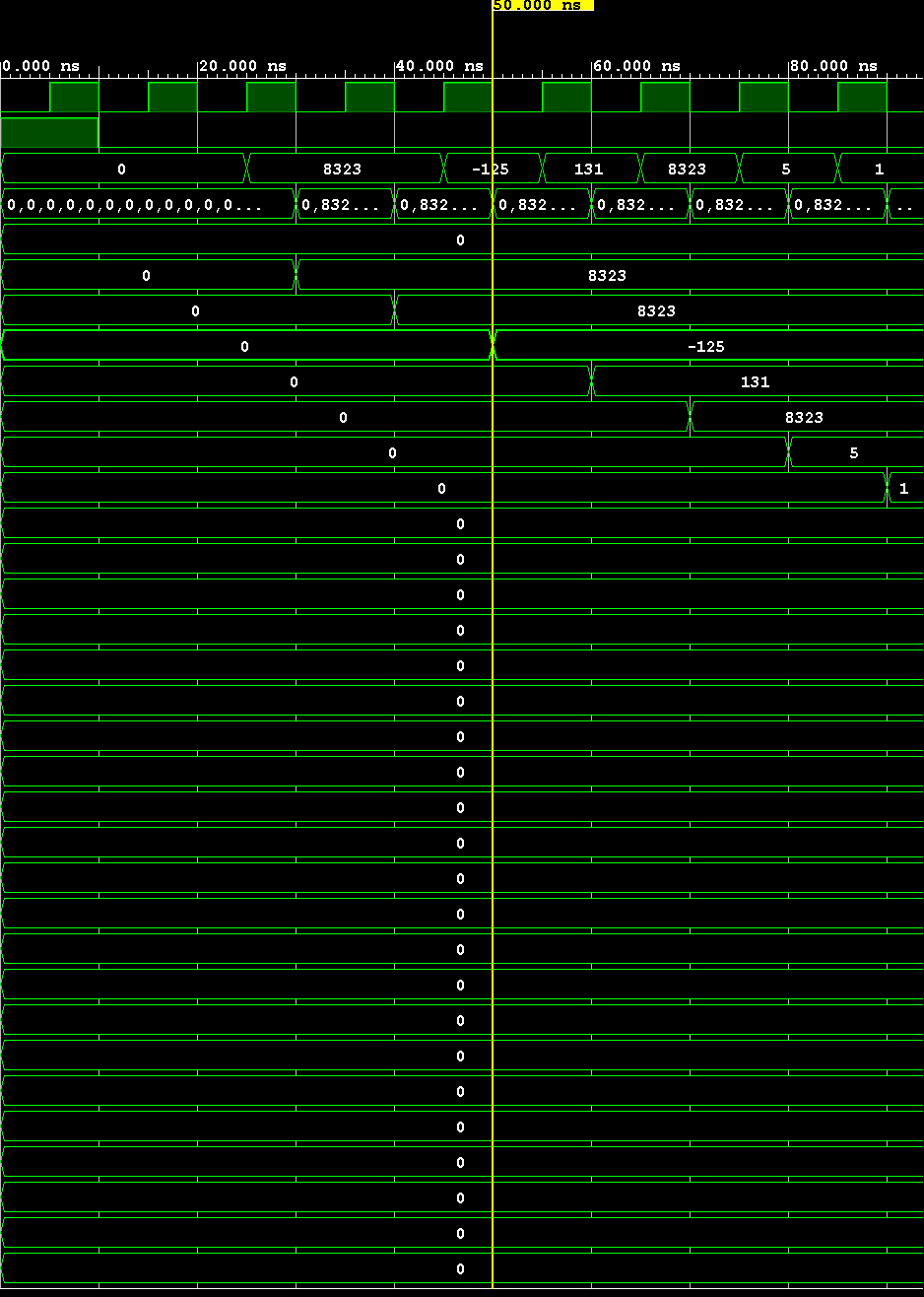
This will load the instruction stored, which is 8323 in the decimal into the register, which is shown correctly in the simulation picture above. Please note that this delay in clock cycle is due to the initial warmup required for it to be fully pipelined.

2- lh x2, 0(x0)



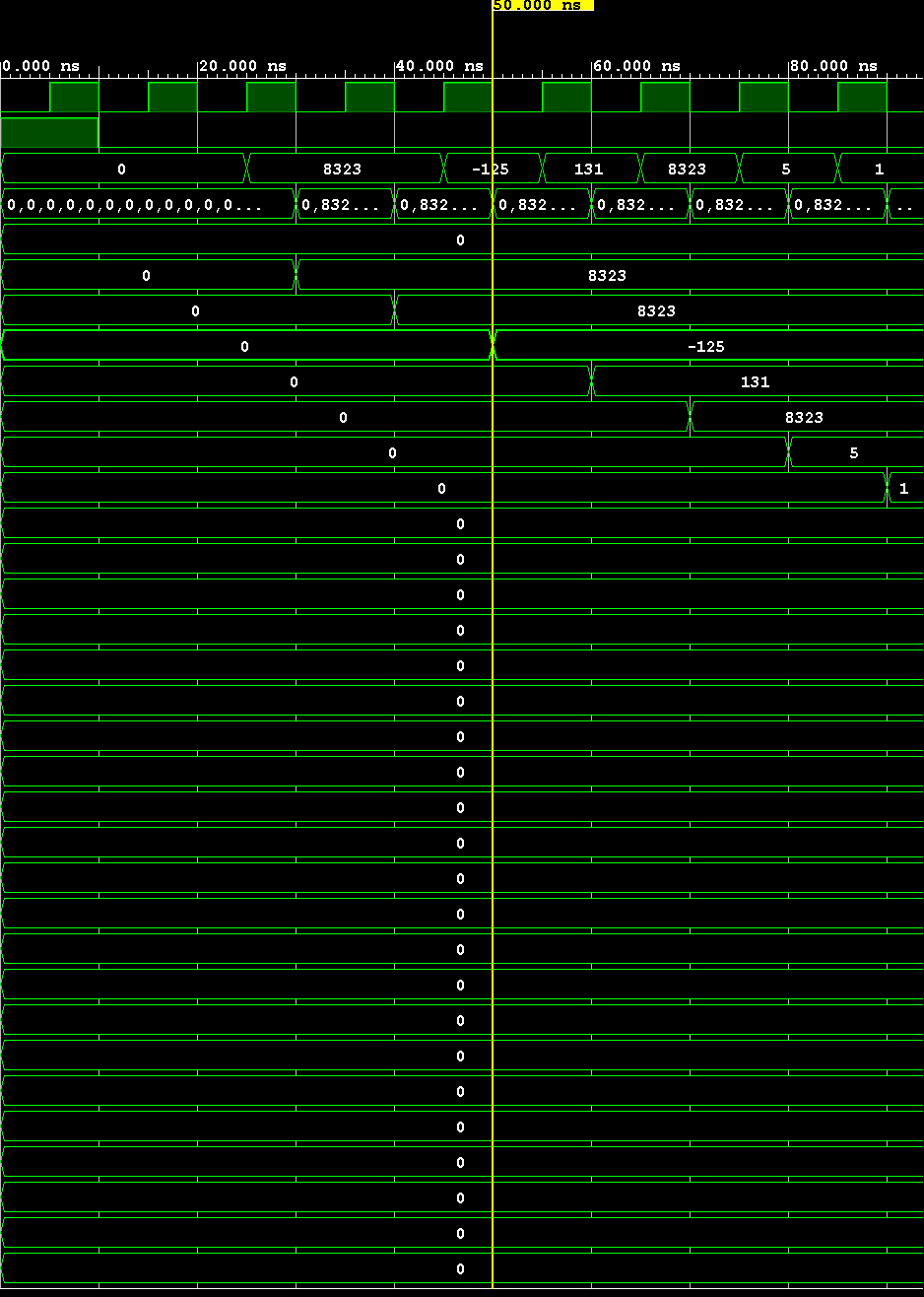
This will also load the half word, which will be in our case 001000001000001, which is 8323 in decimal.

3- lb x3, 0(x0):



This will load 131, not -125, as it will interpret it as unsigned as shown above. This will be loaded to x4.

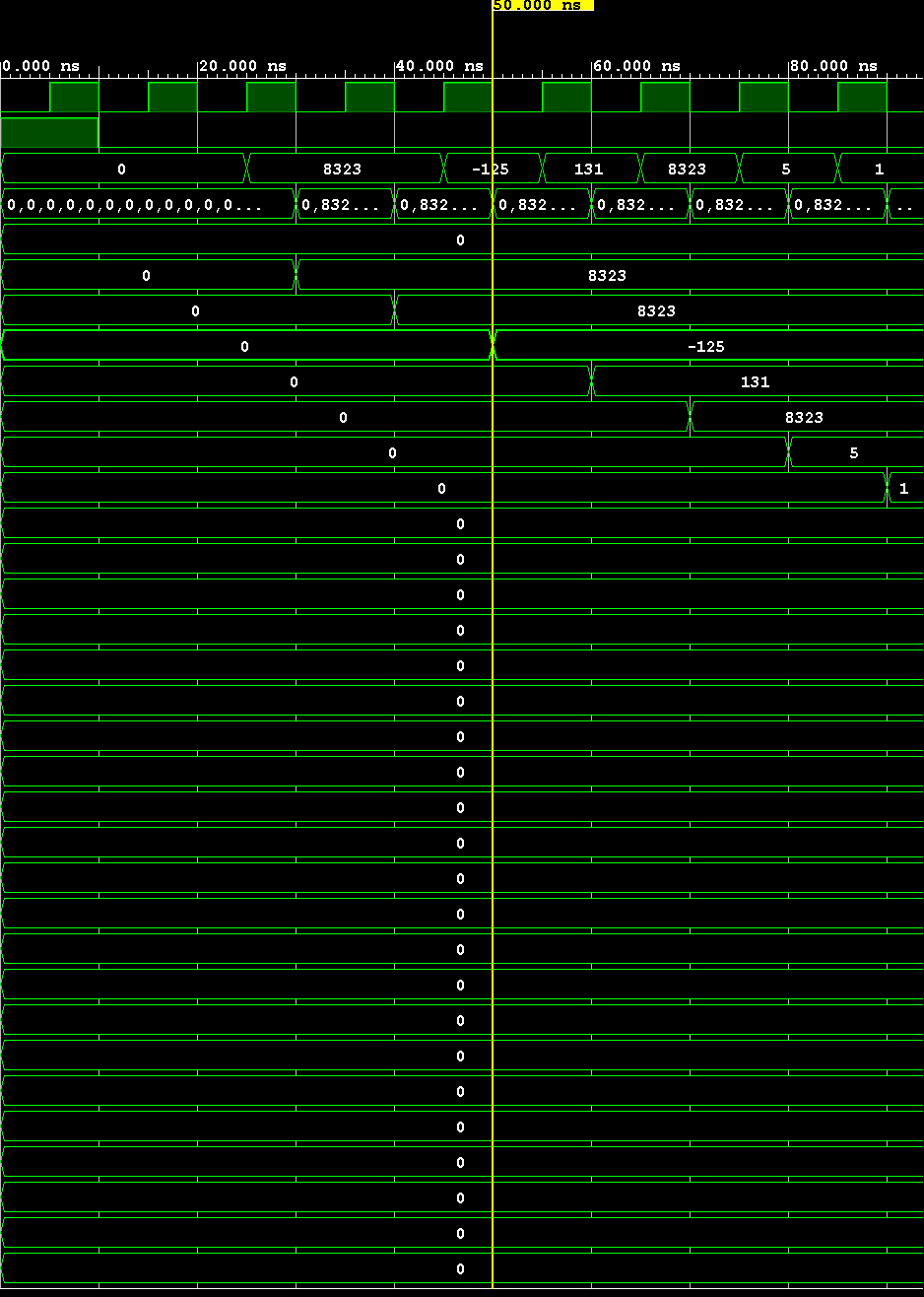
4- lhu x5, 0(x0)



This will load half word unsigned which is stored in the memory into x5. Hence, we will take the first 16 bits of the memory, so mem[0] and mem[1] as well. The value will be 8323.

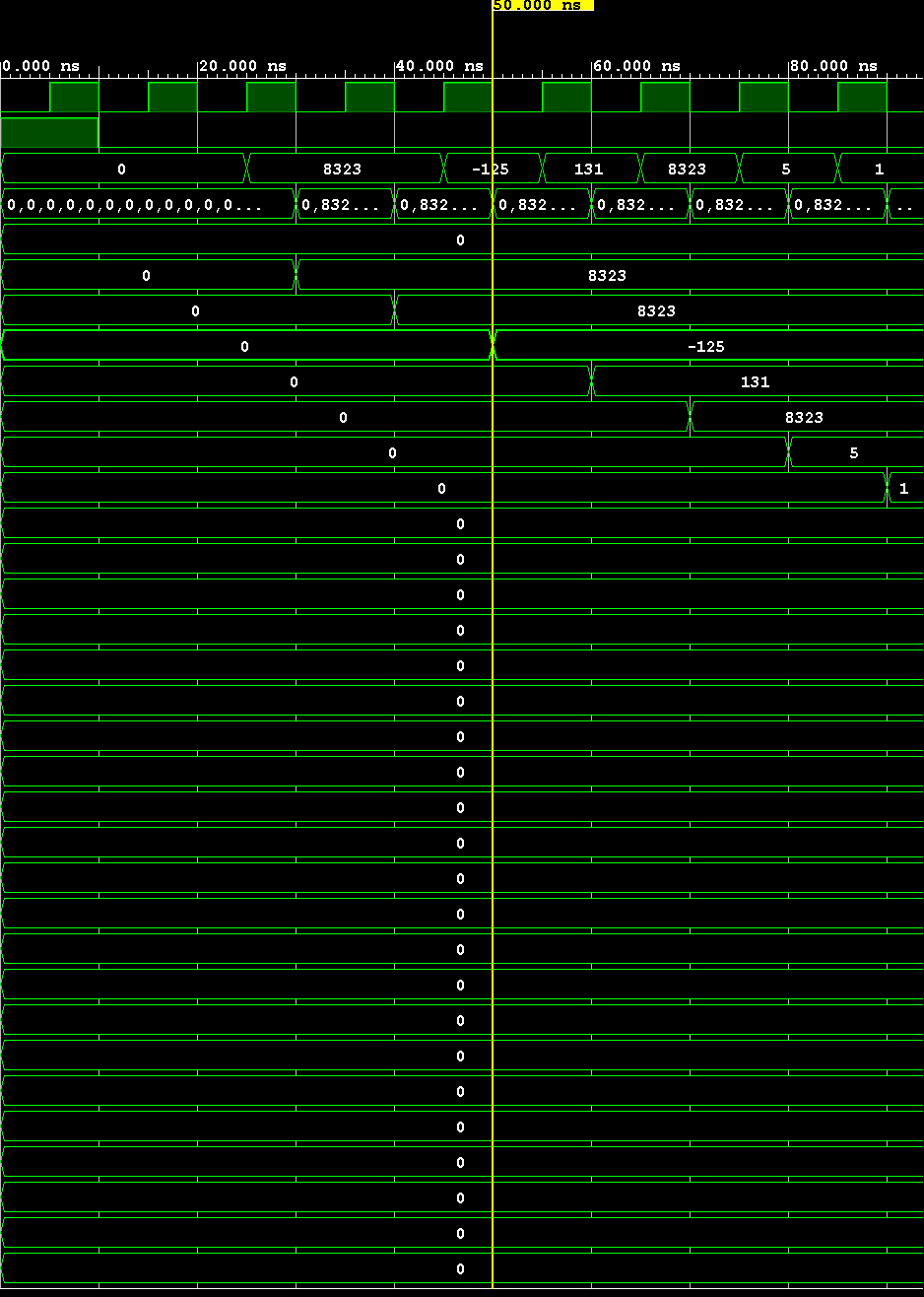
Of course, we are on a pc of 16.

5- //addi x6, x0, 5 // x6: 0 -> 5, PC: 32 -> 36, since all registers are initially set to 9



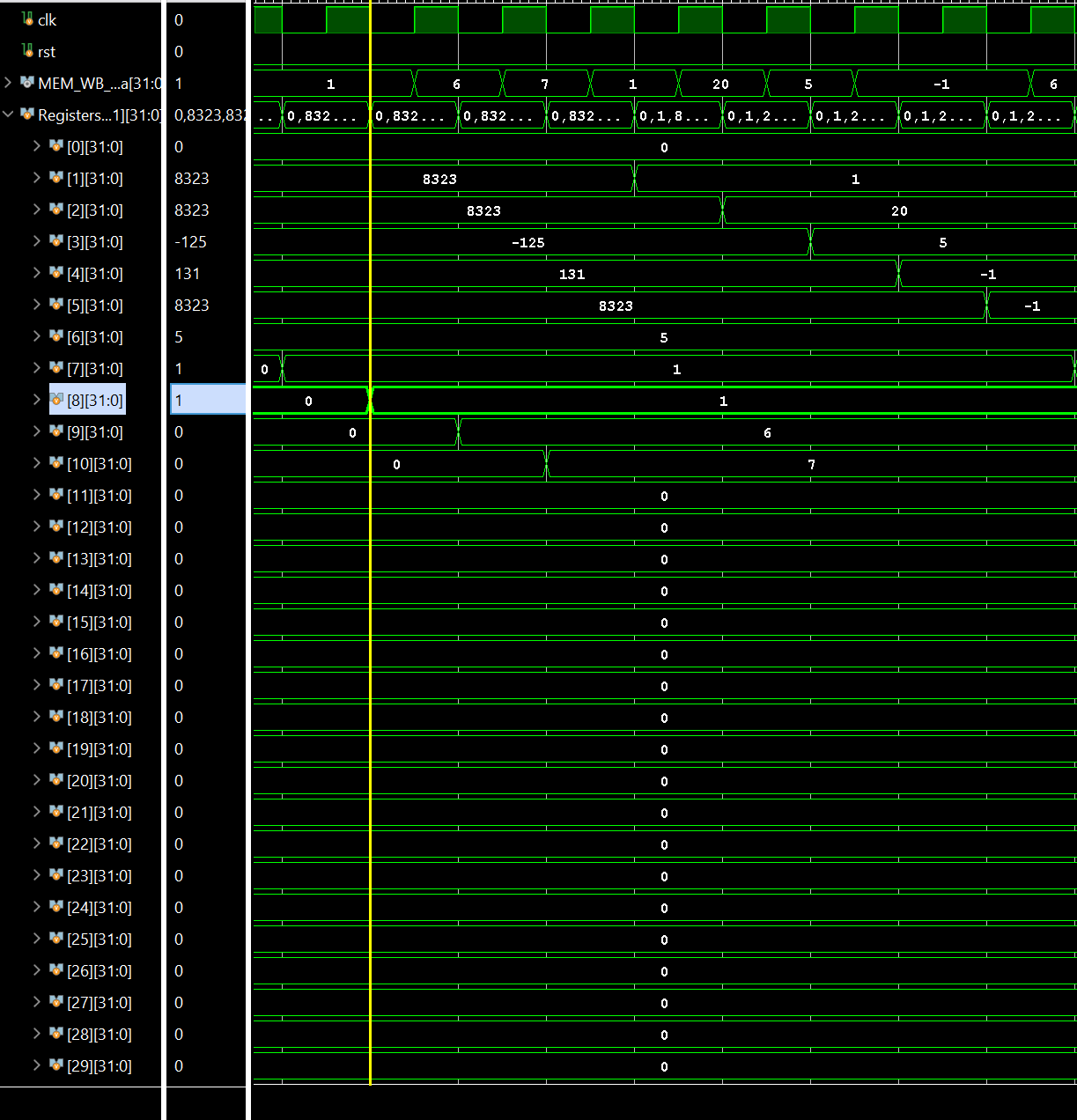
The addi will add the value of x0 with the value of 5. X6 will contain the value of 5, which is shown in the picture above.

6- slti x7,x6,10



X6 is less than 10, hence x7 should contain 1, which is also shown in the figure above.

//sltiu x8, x6, 10



X6 is also less than 10, so x8 contains 1

//xori x9,x6,3



This will take x6 which contains 5 and xor it with 3. 0101 and 0011 -> 0110, which is 6. Hence, the picture above.

//ori x10, x6, 3



We or the x6 and 3. So, 0101 | 0011, which is 0111, which is 7

//andi x1, x6, 3



Anding the value stored in x6 and 3, giving us 1.

//slli x2,x6,2



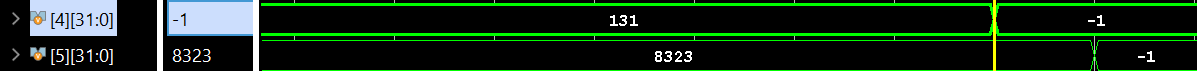
X6 initially contained 5, and we will shift the 5 by 2 bits. Hence becoming x2 = 20.

//srli x3,x2,2



Same procedure as the previous instruction. X2 initially contains 20, and we will shift to the right by 2, hence it will be 5.

//addi x4, x0, -1



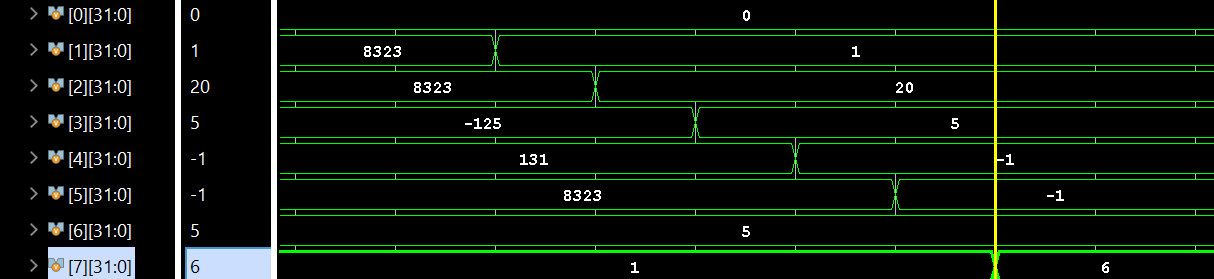
Adds 0 with -1 which will make x4 = -1

// srai x5, x4, 2



X4 initially contained -1, which will be shifted 2 bits to the right. Hence preserving the sign. However when it shifts to the right by 2, however it is all one’s.

// add x7, x6, x1



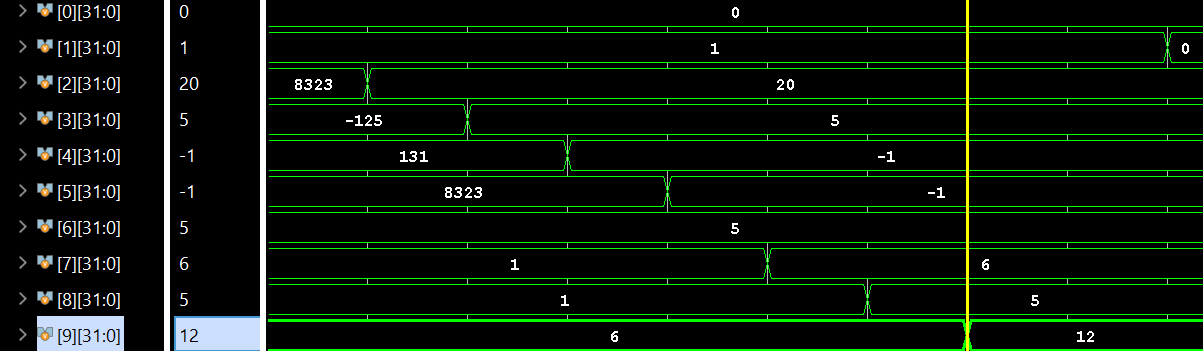
X6 contains 5 and x1 contains 1. so , x7 = 5 + 1 = 6

// sub x8, x7, x1



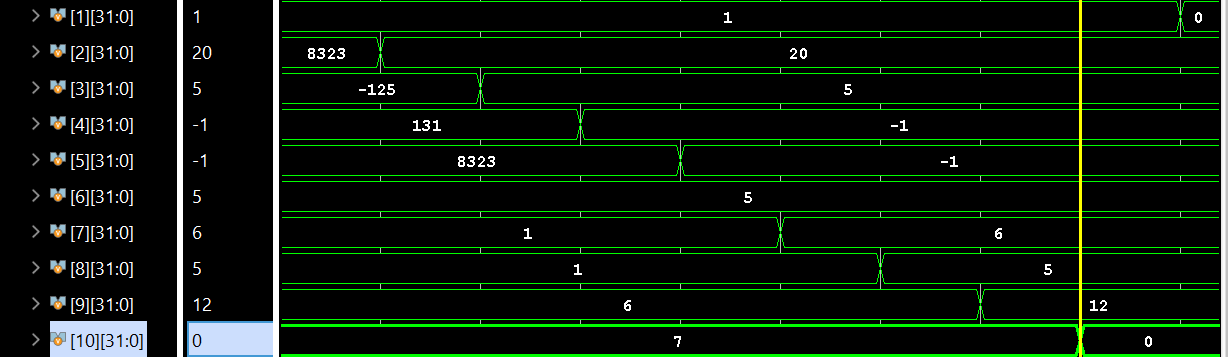
X8 = value in x7 - value in x1 which is 6-1 = 5. Of course this was handled by the forwarding unit.

// sll x9, x7, x1



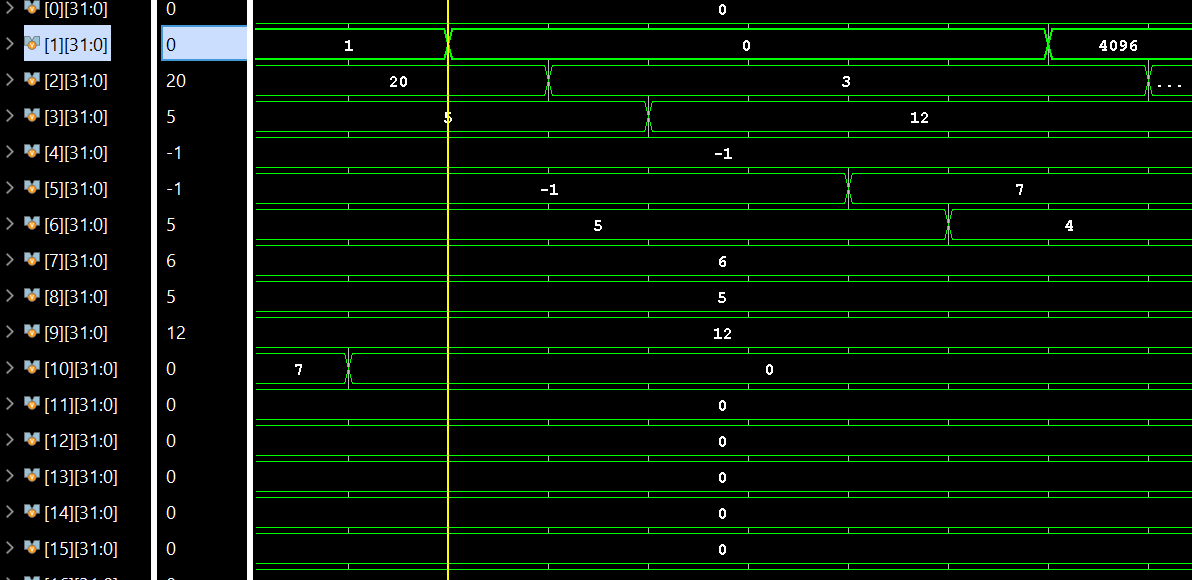
X9 = 6 << 1, which is 6 times 2 = 12.

// slt x10, x7, x8



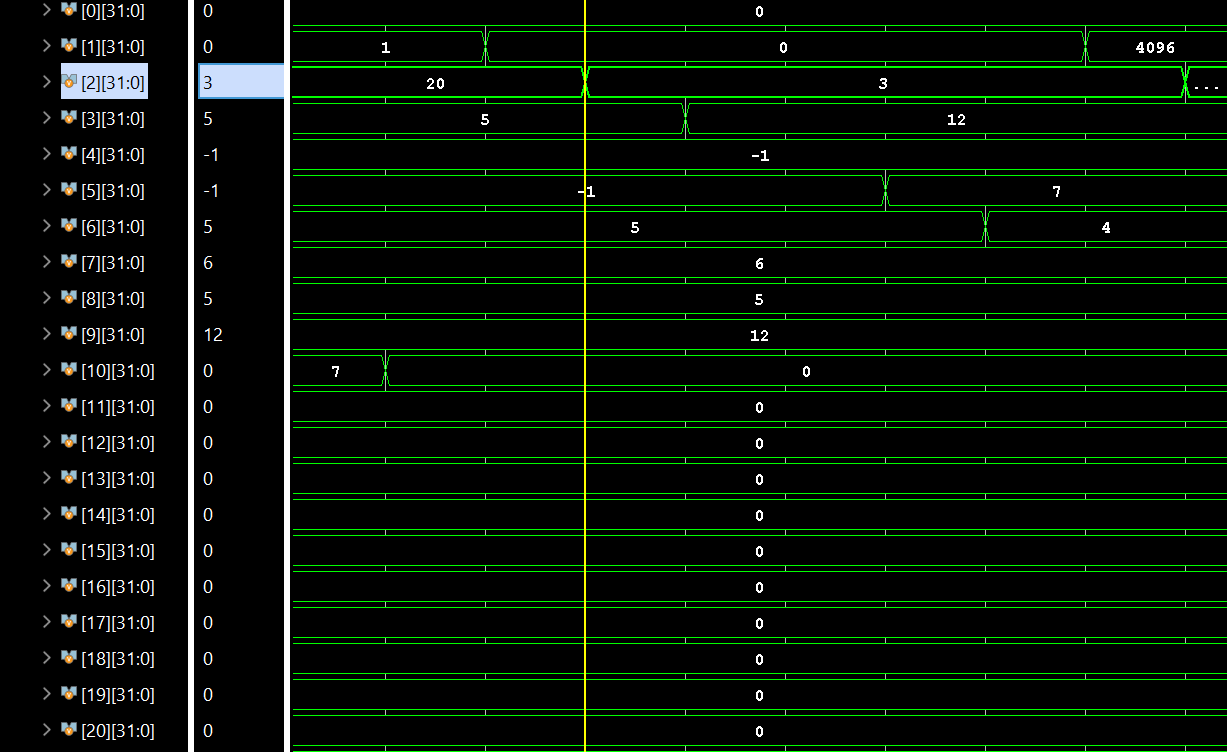
X7 is not less than x8, so x10 will contain 0.

// sltu x1, x7, x1



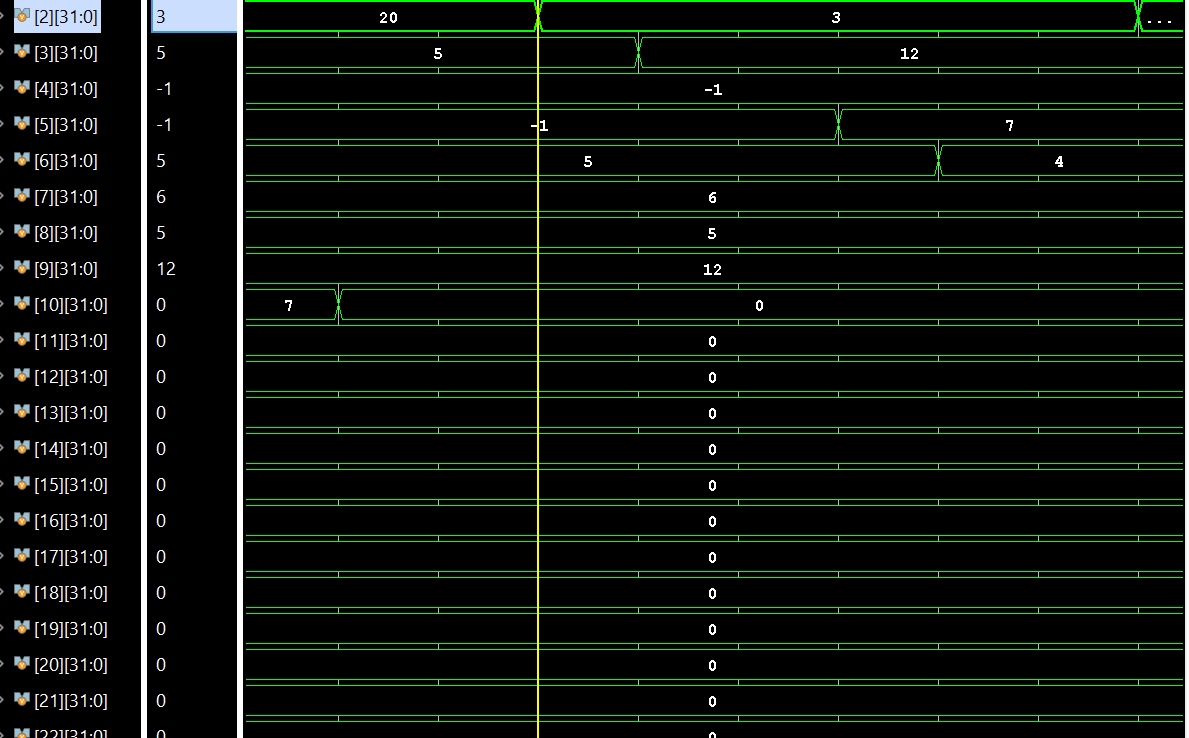
X7 is not less than x1, hence it will be set to 0 (the x1).

// xor x2, x7, x8



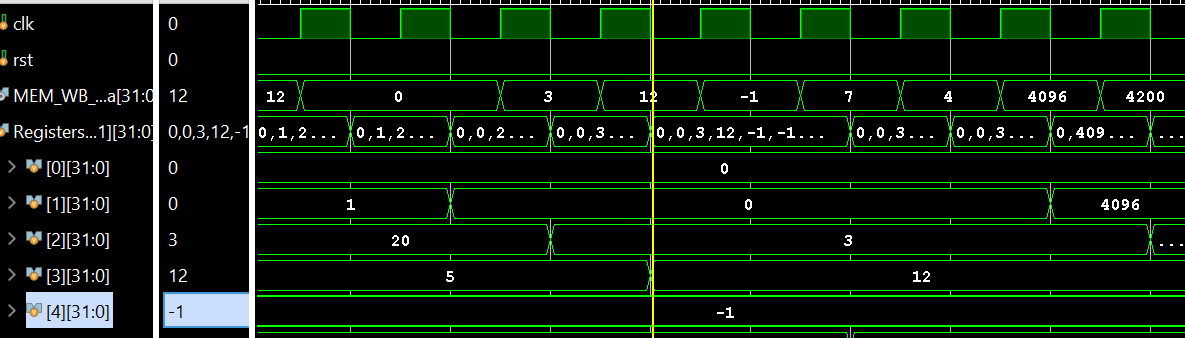
take the xor of (6 ^ 5) 110 101 which is 011

//srl x3, x9,x1



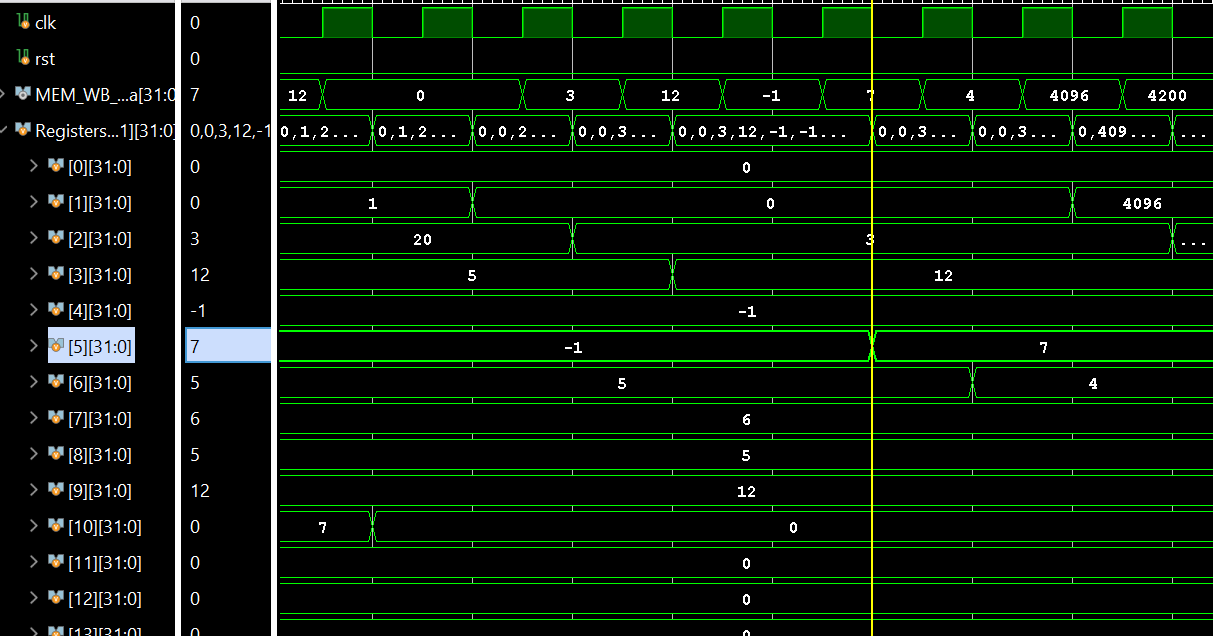
shift to right by 0 which is equivalent to dividing by 1 so we have (12 >> 0)=12

// sra x4, x4, x1



doesn't change since sign was preserved

// or x5, x7, x8



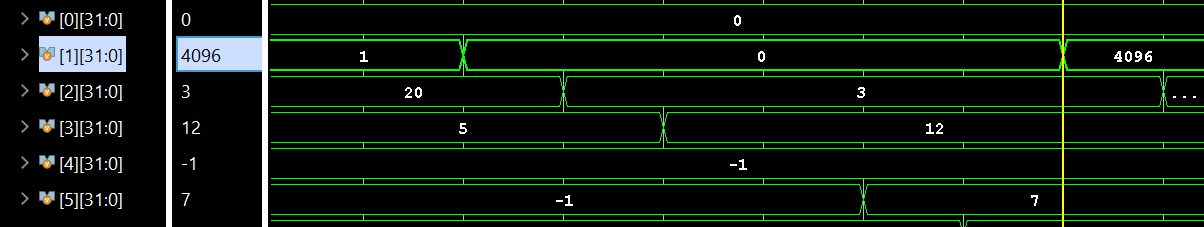
X5 = x7 || x8 which is the oring of 110 101 -> 111, which is 7

// and x6, x7, x8



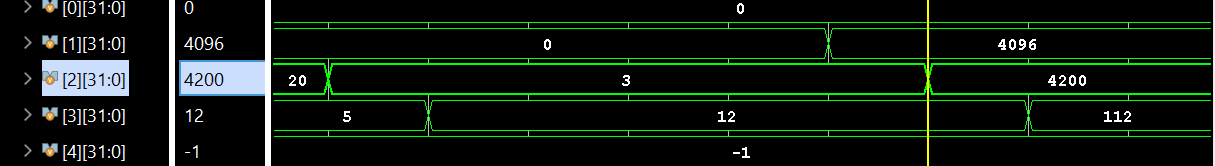
110 & 101 -> 100 which is 4

// lui x1, 1



X1 = 4096

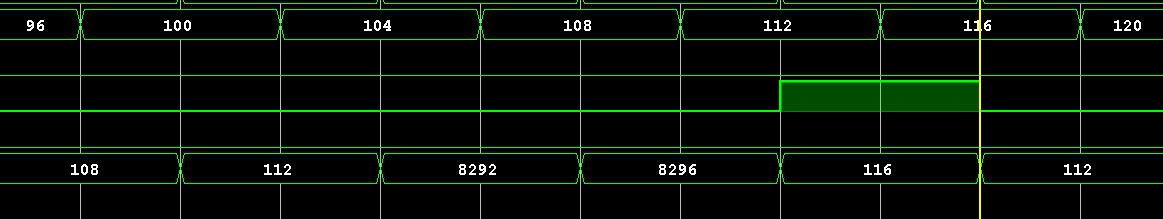
//auipc x2, 1



Pc + (1 << 12), the 104 + 4096 = 4200

// jal x3, 4

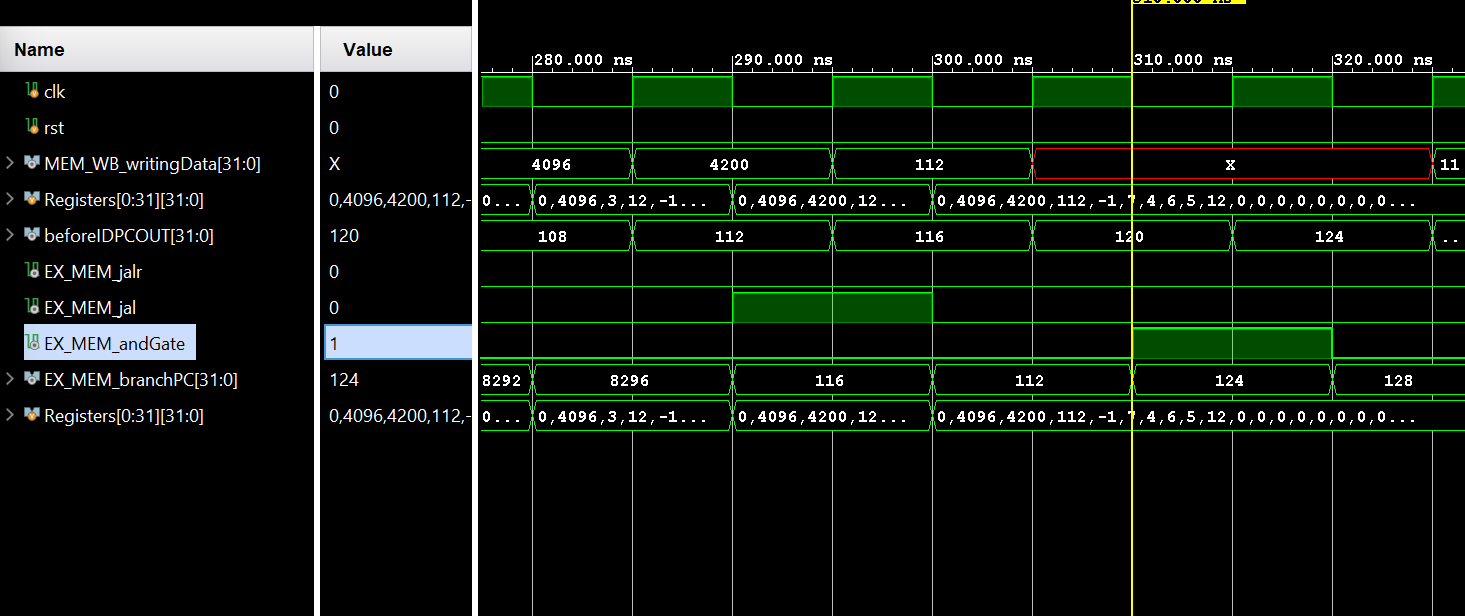
The pc will jump with an immediate of 8



And pcPlus4 will be stored in x3, which is 112 because 108 will be in the MEM

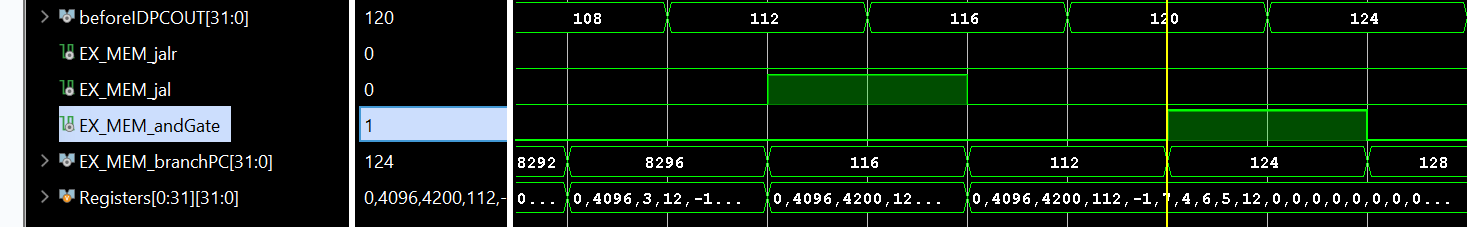


// beq x1, x1, 4



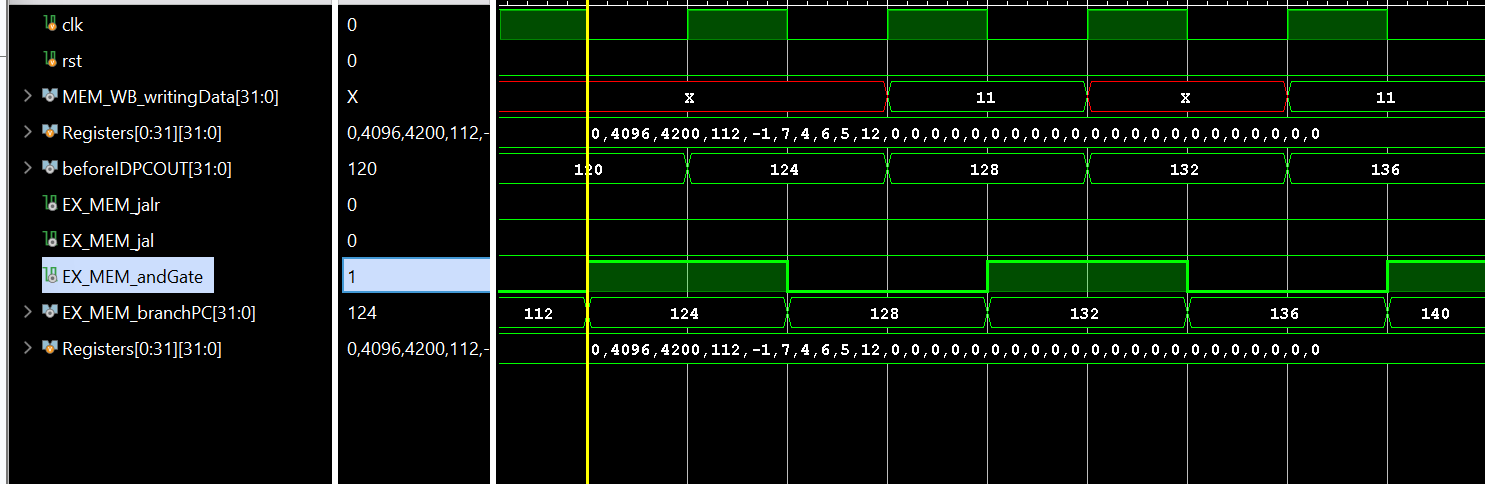
This will also branch because x1 = x1.

// bne x7, x8, 4



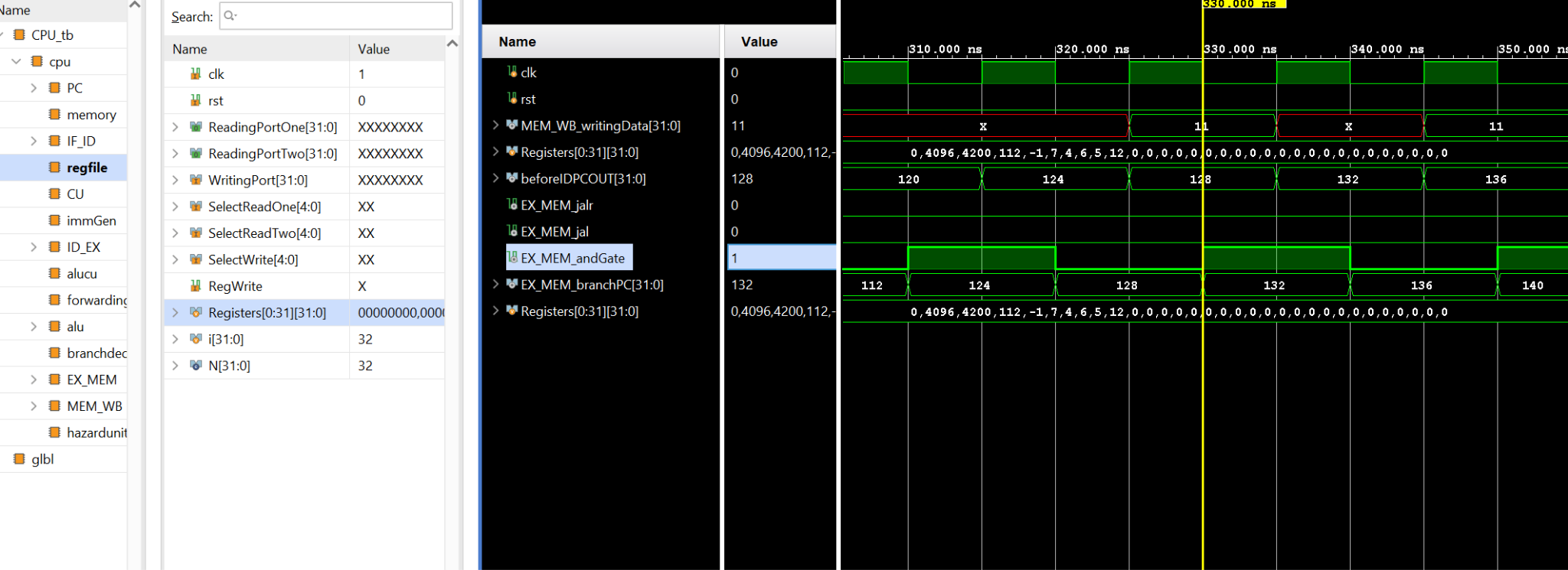
X7 ! = x8 , so it will branch to next instruction

// blt x8, x7, 4



Will not branch because x8 is less than x7

// bge x7, x8, 4



It will branch because x7 > x8

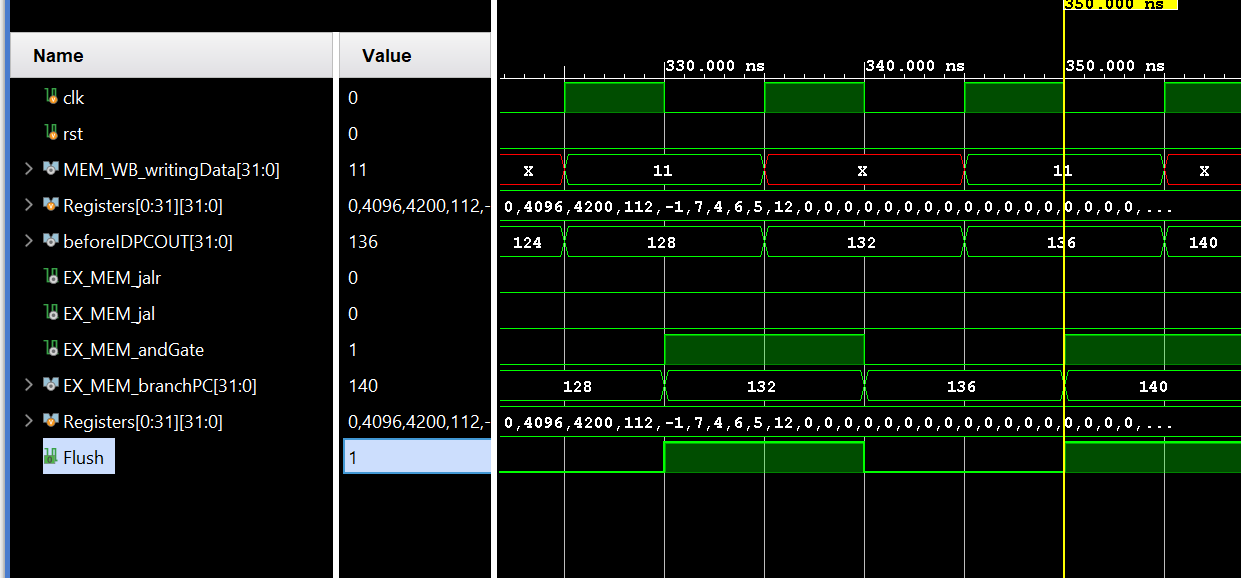
// bltu x8, x7, 4

It will not work because I did not implement a branch prediction mechanism here.

The hazard detection unit always give a nop and as a result the instruction below it, which is the

Bltu, will be flushed out, as we always assume that we will not predict correctly.

//bgeu x7,x8, 4



It will branch because x7 unsigned is less than x8 unsigned (carry flag is 1)

// sw x1, 4(x0)

It will not do so because of the flush

//sh x2, 8(x0)





It stores part of x2 into the memory, as shown in the picture above

sb x3, 12(x0)



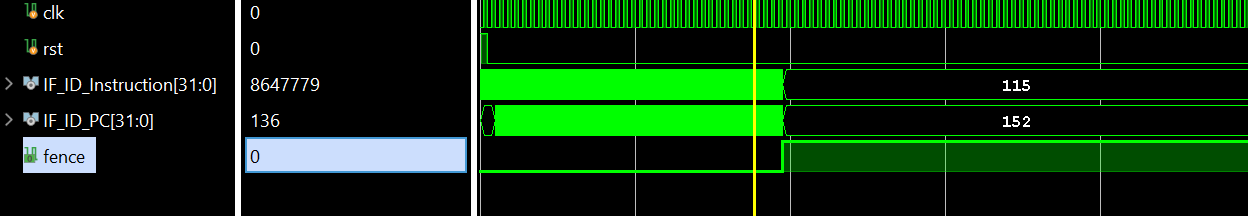
This is the register x3 contents



This is what happened to the memory location number 12. It changes from 03 to 70 because we stored the contents of x3 into it.

Fence Instructions:

Ecall, ebreak, pause, fence.tso, fence



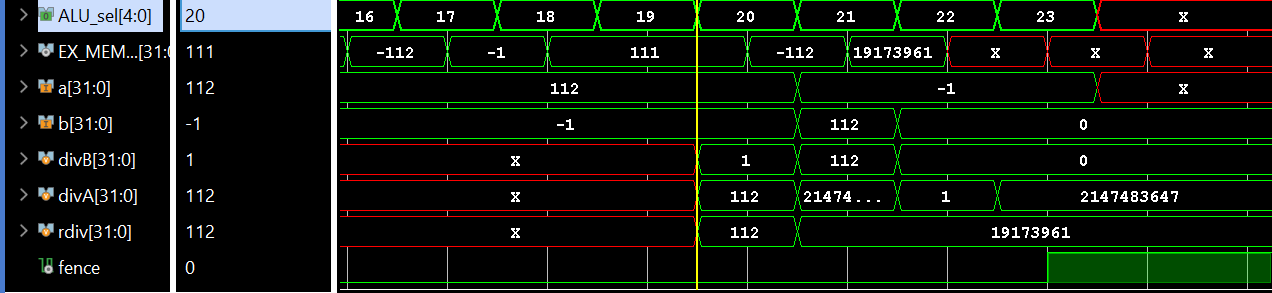
The fence instructions work, note that this applies to the other ending instructions as well.

Bonuses Implemented:

1- Random Testcase Generator:

This will be provided alongside the other files

2- RV32IM support:



Note that a is x3, and b is x4.

Note that i'm using the following instructions

Mul x24, x3, x4

Mulh x24, x3, x4

Mulhsu x24, x3, x4

Mulhu x24, x3, x4

Div x24 , x3 ,x4

Divu x24, x4, x3

Rem x24, x4, x3

Remu x24, x4, x3

Where x3 = 112, and x4 = -1

Mul x24, x3, x4 will store the result of 112 times -1 which is -112 which is stored in x24.



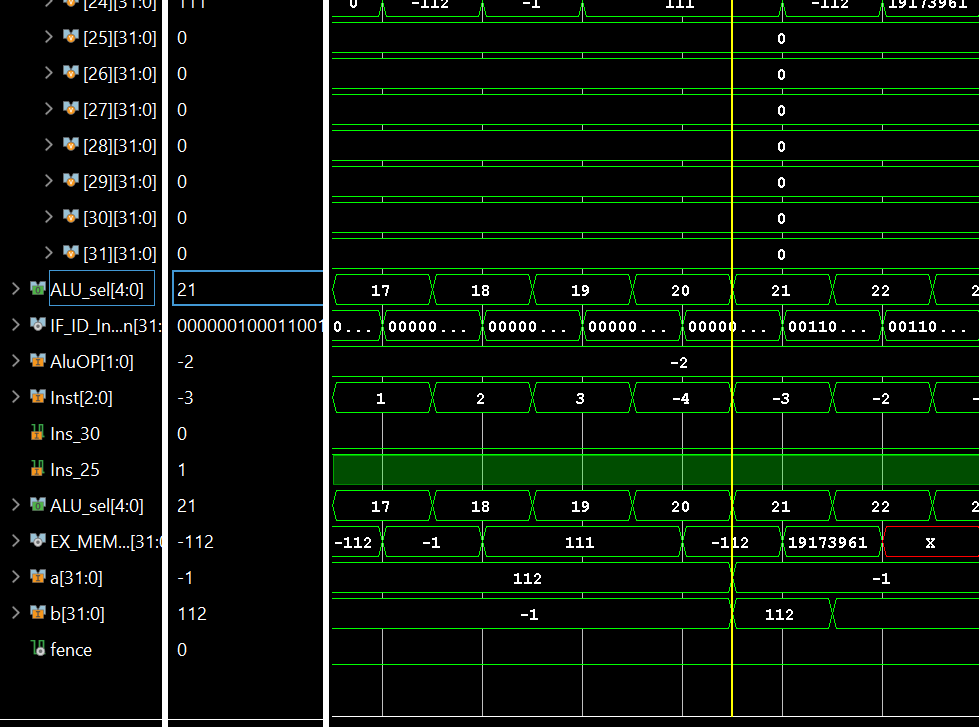
Mulhu x24, x3, x4 will store the upper result which is -1, which can be shown by the figure above as well

Mulhsu x24, x3, x4 will be 111, as we are doing a multiplication of signed x3 by unsigned x4, which will contain the result. The final result will be the upper 32 bits which is 111.

Mulhu x24, x3,x4 will also be 111, as we are doing an unsigned multiplication of signed x3 by unsigned x4.

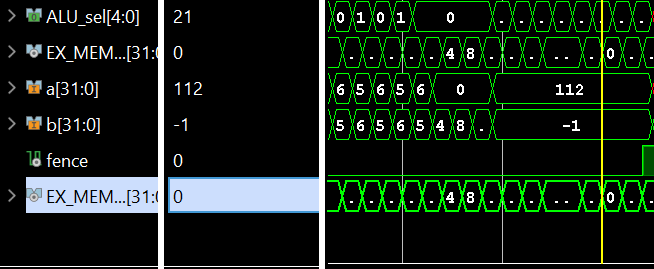
Div x24, x3, x4

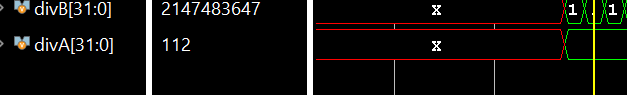
So signed 112 divided by signed 01. The result will be -112 which will be in the figure below.



Divu x24, x3, x4

We are dividing x3 by x4 in an unsigned manner. Hence unsigned x3 divided by unsigned x4.

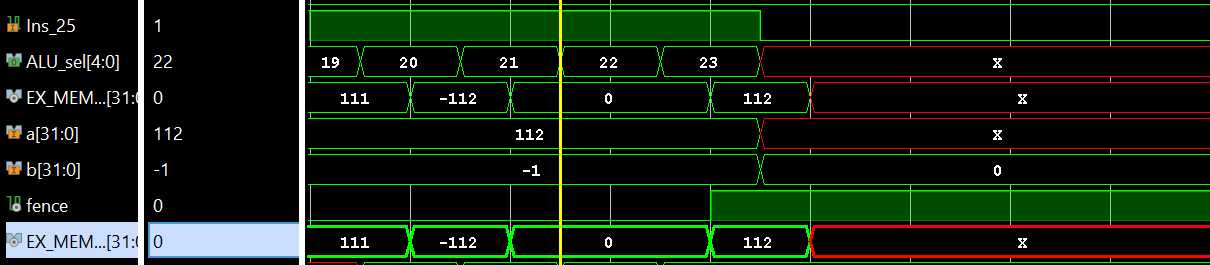


112 will be divided by -1, however the unsigned version of -1. The result will be a very big number of -1, the unsigned version of it is the following number: 

The -1 will be 2147483647

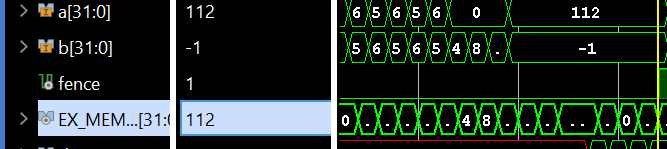
So, 112/ 2147483247 will be 0, which is shown in the image above it.

REM x24, x3 , x4



112 % -1 should be 0 , as there is a number multiplied by -1 that will give us 112, which is -112.

REMU x24, x3, x4



As before, x4 is represented as unsigned, which will turn it into a huge number, which is mentioned before ( 2147483647). Hence, the remainder will be 112, since there’s no integer number multiplied by 2147483647 that will give us 112.

Link to Draw.io file to see the diagram:

<https://drive.google.com/file/d/1CHdNiYKKWFL2lcJN_PO5w8Dmp5oNMjYb/view?usp=sharing>