CompE475 Homework Assignment 2

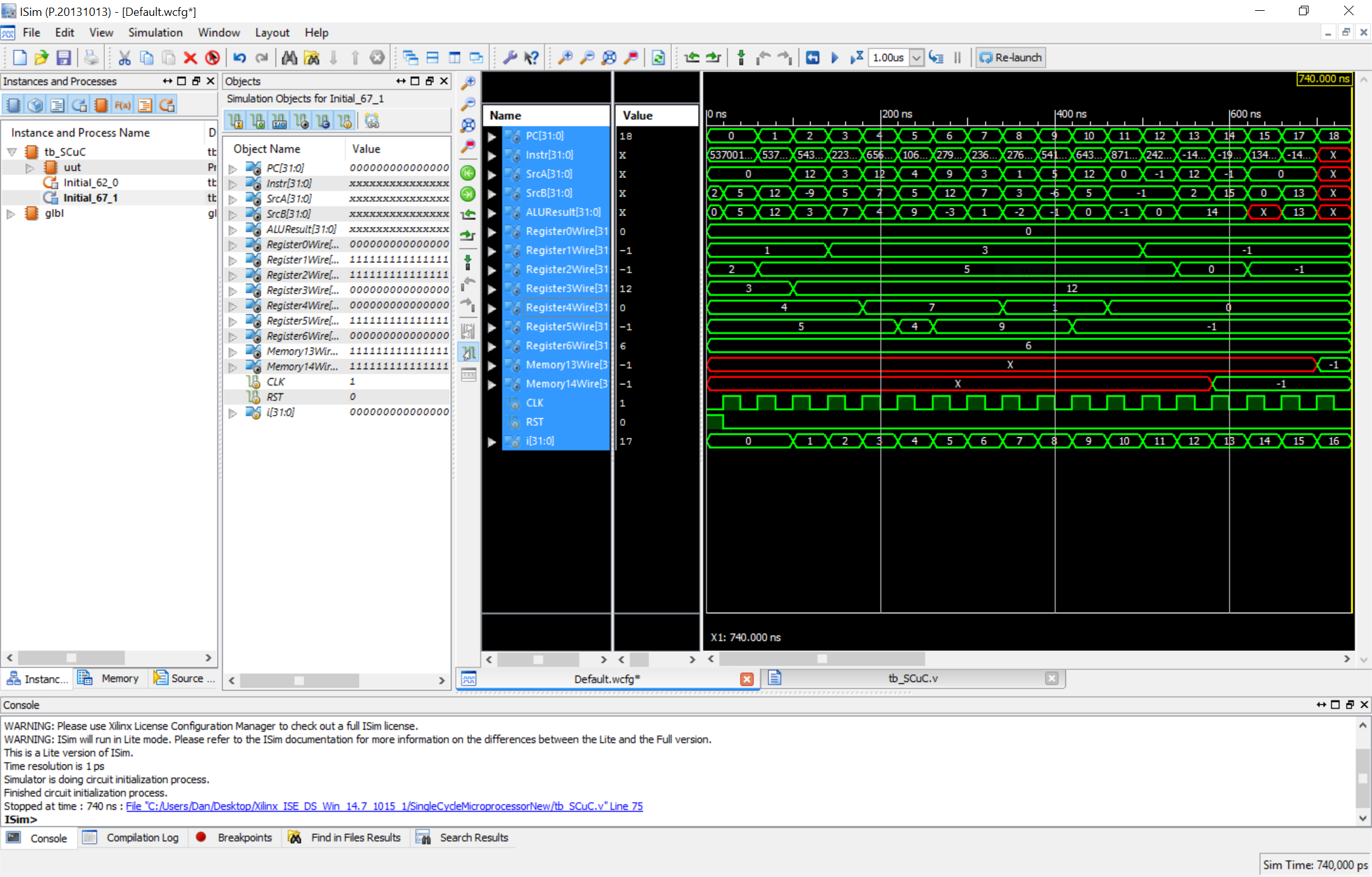
**(Q1)**

1. For the lui instruction, you must shift the immediate value left by 16 bits before storing it in the specified register. First, the immediate value will be sign-extended, then go through the SrcMux 1 input, so ALUSrc will need to be high. Then we can add a ‘left shift by 16’ to the ALU unit to shift SrcB by 16 bits to the left. We can just add this as ALU = 3’b100, for example. The operation will be ALUResult = SrcB << 16. The WriteMux will choose the destination register as rt, so RegDst will be 0. After, the result will be processed as any R-type instruction, as the result is written to the specified register.
2. For the jr instruction, since rs is the location we must jump to, we will add another input on the JumpMux and make the control unit signal Jump 2 bits instead of 1. If 00, we use BranchMuxOut, if 01, we use the traditional jump concatenated value, and if 10, we take SrcA from the register file and bring it around to the input of the JumpMux at 10. Everything else stays pretty much the same as it is similar to the jump instruction.
3. For the sllv instruction, we use basic R-type procedures. We take rs and rt and put them on SrcA and SrcB, respectively. As we have options open with the ALU, we add an option to the ALU that left shifts the SrcB value by the SrcA value. For example, we can make this command 011 on the ALUControl. The result will be written to the destination register as for other R-type instructions.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | RegWrite | RegDst | AluSrc | Branch | MemWrite | MemtoReg | Jump |
| lui | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| jr | 0 | X | X | X | 0 | X | 10 |
| sllv | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

**(Q2)**

Output waveform:



binary format:

00100000000000100000000000000101

00100000000000110000000000001100

00100000011000011111111111110111

00000000001000100010000000100101

00000000011001000010100000100100

00000000101000100010100000100000

00010000101000110000000000000001

00000000001001000010000000101010

00010000100000010000000000001010

00100000010001011111111111111010

00000000011000100010000000101010

00000000100001010000100000100000

00000000001001010001000000100010

10101100011000010000000000000010

10001100101000100000000000001111

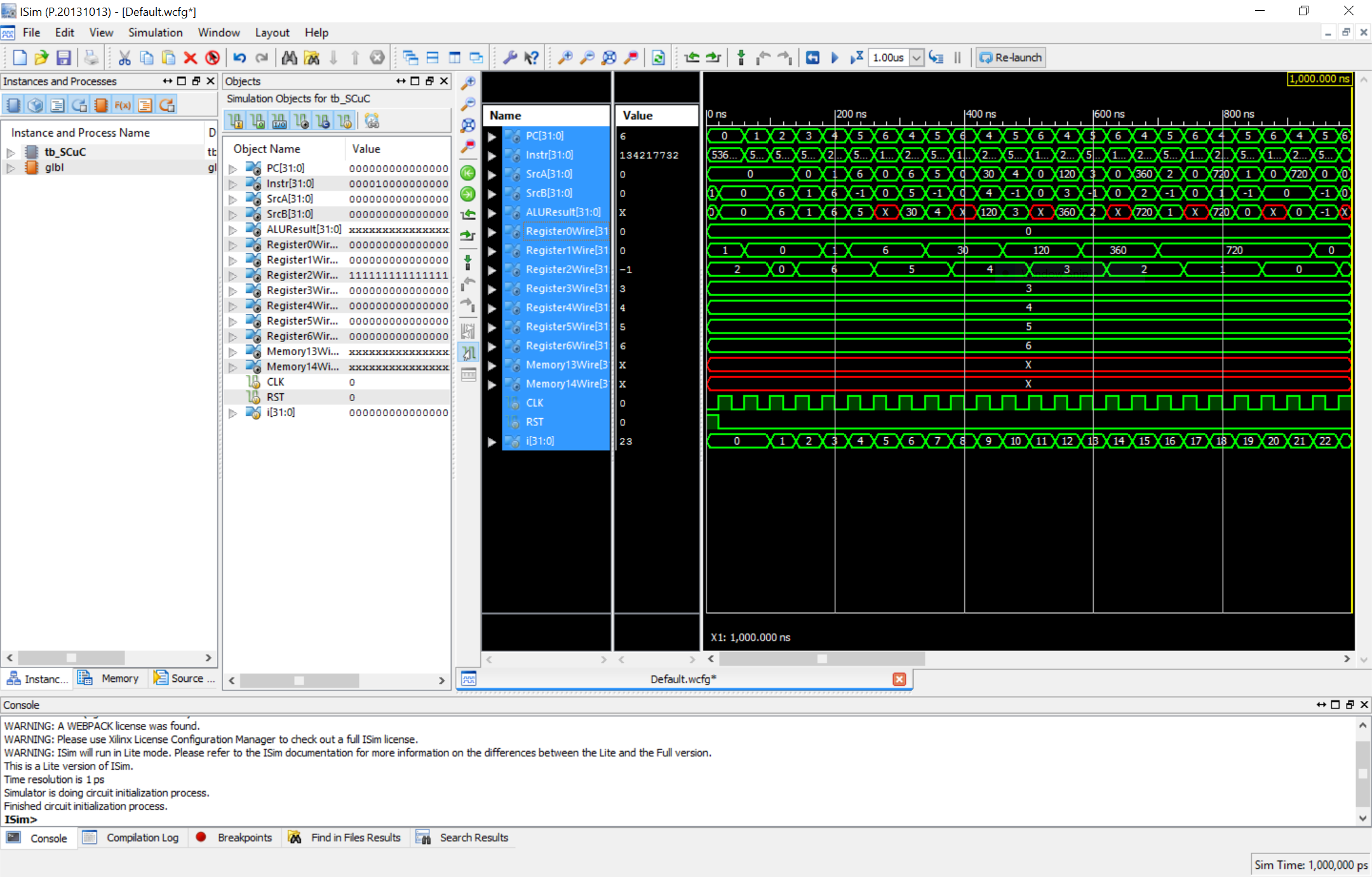
00001000000000000000000000010001

00100000000000100000000000000001

10101100000000100000000000001101

**(Q3)**

Output waveform:



***This is what I was trying to demo, but couldn’t get to work for some reason. I swear everything is right with my processor after double checking 20 times, but I think the following binary MIPS conversion may just be wrong. The jr, jal, and mul(pseudo version) are all additionally implemented, as far as I know, correctly.***

factorial: addi $29, $29, -8

sw $4, 4($29)

sw $31, 0($29)

addi $8, $0, 2

addi $4, $4, 6 // For testing n=6.

slt $8, $4, $8

beq $8, $0, else

addi $2, $0, 1

addi $29, $29, 8

jr $31

else: addi $4, $4, -1

jal factorial

lw $31, 0($29)

lw $4, 4($29)

addi $29, $29, 8

mul $2, $4, $2 (mult $4, $2, mflo $2)

jr $31

binary format:

00100011101111011111111111111110

10101111101001000000000000000001

10101111101111110000000000000000

00100000000010000000000000000010

00100000100001000000000000000110

00000000100010000100000000101010

00010001000000000000000000000011

00100000000000100000000000000001

00100011101111011111111111111110

00000011111000000000000000001000

00100000100001001111111111111111

00001100000000000000000000000000

10001111101111110000000000000000

10001111101001000000000000000001

00100011101111010000000000000010

00000000100000100001000000011000

00000011111000000000000000001000

***The following is what worked for me to display the correct output (should have done it this way much sooner):***

addi $1, $0, 0

addi $2, $0, 0

addi $2, $0, 6

addi $1, $0, 1

loop: mul $1, $1, $2

addi $2, $2, -1

j loop

binary format: (not using function call)

00100000000000010000000000000000

00100000000000100000000000000000

00100000010000100000000000000110

00100000001000010000000000000001

00000000001000100000100000011000

00100000010000101111111111111111

00001000000000000000000000000100