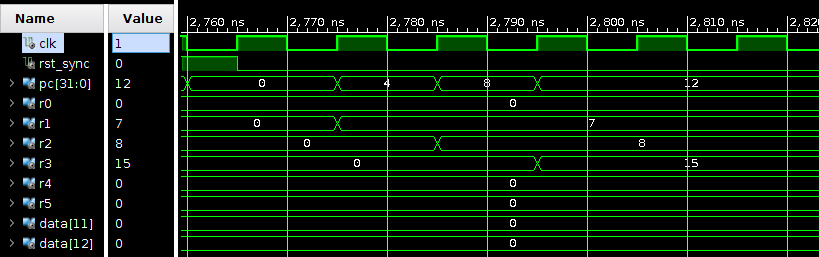
2.3.2 Correctness of the Processor Design

The processor is tested with the sample code provided on the project description. The results are as follows:

Sample code 1:

00000100000000010000000000000111 --ADDI R1, R0, 7 // R1 = 7 00000100000000100000000000001000 --ADDI R2, R0, 8 // R2 = 8 00000000010000010001100000010000 --ADD R3, R1, R2 // R3 = R1 + R2 =15 11111100000000000000000000000000 --HAL // HALT



Here, the instruction is to add the immediate value 7 and R0 and store it in R1. As we can see from the above screenshot, R1 is 7. Similarly, R2 is 8. Then we add the value of R1 and R2 and store the result in R3. It’s easy to get that R3 is 15. The result we get is the same as the hand calculation.

Sample code 2:

000001 00000 00001 0000000000000010 --ADDI R1, R0, 2 //R1=R0+2(decimal)

000001 00000 00011 0000000000001010 --ADDI R3, R0, 10 //R3=R0+10(decimal)

000001 00000 00100 0000000000001110 --ADDI R4, R0, 14 //R4=R0+14(decimal)

000001 00000 00101 0000000000000010 --ADDI R5, R0, 2 //R5=R0+2

001000 00011 00100 0000000000000010 --SW R4, 2(R3) //Mem[R3+2]=R4

001000 00011 00011 0000000000000001 --SW R3, 1(R3) //Mem[R3+1]=R3

000000 00100 00011 00100 00000 010001 --SUB R4, R4, R3 //R4=R4-R3

000010 00000 00100 0000000000000001 --SUBI R4, R0, 1 //R4=R0-1(decimal)

000000 00011 00010 00100 00000 010010 --AND R4, R2, R3 //R4=R2 and R3

000011 00010 00100 0000000000001010 --ANDI R4, R2, 10 //R4=R2 and 10(decimal)

000000 00011 00010 00100 00000 010011 --OR R4, R2, R3 //R4= R2 or R3

000111 00011 00010 0000000000000001 --LW R2, 1(R3) //R2=Mem[1+R3]

000100 00010 00100 0000000000001010 --ORI R4, R2, 10 //R4=R2 or 10(decimal)

000000 00011 00010 00100 00000 010100 --NOR R4, R2, R3 //R4= R2 nor R3

000101 00010 00100 0000000000001010 --SHL R4, R2, 10 //R4= R2 << 10(decimal)

000110 00010 00100 0000000000001010 --SHR R4, R2, 10 //R4=R2 >> 10(decimal)

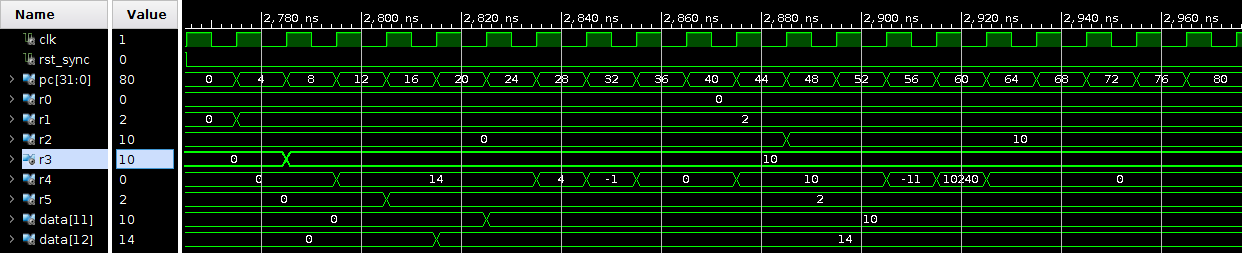
001010 00000 00101 1111111111111110 --BEQ R5, R0, -2

001001 00100 00101 0000000000000000 --BLT R5, R4, 0

001011 00100 00101 0000000000000000 --BNE R5, R4, 0

001100 00000000000000000000010100 --JMP 20

111111 00000000000000000000000000 --HAL



(1) ADDI R1, R0, 2 //R1=R0+2(decimal)

The instruction is to add the immediate value 2 and R0 and store it in R1. So as seen from the above screenshot, R1 is 2 after the instruction.

(2) ADDI R3, R0, 10 //R3=R0+10(decimal)

The instruction is to add the immediate value 10 and R0 and store it in R3. So as seen from the above screenshot, R3 is 10 after the first instruction.

(3) ADDI R4, R0, 14 //R4=R0+14(decimal)

The instruction is to add the immediate value 14 and R0 and store it in R4. So as seen from the above screenshot, R4 is 14 after the instruction.

(4) ADDI R5, R0, 2 //R5=R0+2

The instruction is to add the immediate value 2 and R0 and store it in R5. So as seen from the above screenshot, R5 is 2 after the instruction.

(5) SW R4, 2(R3) //Mem[R3+2]=R4

The instruction is to store the word from the memory location R3+2. So as seen from the above screen shot, R3 is 10, Mem[12] = 14, so R4 is 14 after the instruction.

(6) SW R3, 1(R3) //Mem[R3+1]=R3

This instruction is to store the word from the memory location R3+1. R3 is 10, Mem[11] is 10. So after the instruction, the value of R3 is still 10.

(7) SUB R4, R4, R3 //R4=R4-R3

The instruction is to subtract R3 from R4 and store in R4. R4 is 14, R3 is 10, so R4 will be 4 after this instruction, which matches the result in the screenshot.

(8) SUBI R4, R0, 1 //R4=R0-1(decimal)

The instruction is to subtract R0 from the immediate value 1 and store in R4. R0 is 0, so after the instruction, R4 will change to -1.

(9) AND R4, R2, R3 //R4=R2 and R3

The instruction is to AND R2 and R3 and store in R4. R2 is 0, R3 is 10, after this instruction, R4 is 0.

(10) ANDI R4, R2, 10 //R4=R2 and 10(decimal)

The instruction is to AND Immediate R2 and the immediate value 10 and store it in R4. As R2 is 0, the result in R4 is also 0.

(11) OR R4, R2, R3 //R4= R2 or R3

The instruction is to OR R2 and R3 and store the result in R4. R2 is 0, but R3 is 10, so the result is 10. As we can see from the screenshot, the result in R4 is 10.

(12) LW R2, 1(R3) //R2=Mem[1+R3]

The instruction is to Load the Data from the memory location [1+R3] and store it in the R2. R3 is 10, Mem[11] is 10, so R2 change to 10 after this instruction.

(13) ORI R4, R2, 10 //R4=R2 or 10(decimal)

The instruction is to OR R2 and the immediate value 10 and store in the register R4. R2 is 10, so R4 is still 10, the value doesn’t change.

(14) NOR R4, R2, R3 //R4= R2 nor R3

The instruction is to implement NOR on R2 and R3 and store the result in R4. R2 is 10 and R3 is 10, R4 should be -11. (01010 nor 01010 = 10101)

(15) SHL R4, R2, 10 //R4= R2 << 10(decimal)

The instruction performs a logical left shift, left shift R2 by 10 bits, R2 is 10, after the instruction, the result should be 10240.

(16) SHR R4, R2, 10 //R4=R2 >> 10(decimal)

The instruction performs a logical left shift, left shift R2 by 10 bits, R2 is 10, after the instruction, the result should be 0.

(17) BEQ R5, R0, -2

The instruction “BEQ” compares the branch for equality, if R5 equals to R0, it will go backward 2 instructions. In the given instruction, it checks if R0 is equal to R5, which is false.

(18) BLT R5, R4, 0

The instruction “BLT” checks if R5 is less than R4. R5 is 2 and R4 is 0. So it R5 is not less than R4.

(19) BNE R5, R4, 0

The instruction “BNE” compares the branch for equality. R5 is 2 and R4 is 0, which is not equal. But as the offset is 0, it will do the next instruction.

(20) JMP 20

Jump instruction is used to jump to the given address. In this case, it will jump to itself.

(21) HAL

HAL is used to stop the program.

As we can see from the screenshot and the hand calculation, our processor can correctly do the job. The results of the screenshot and the hand calculation matches.