03.01.2021

CENG 311

HOMEWORK 3

REPORT

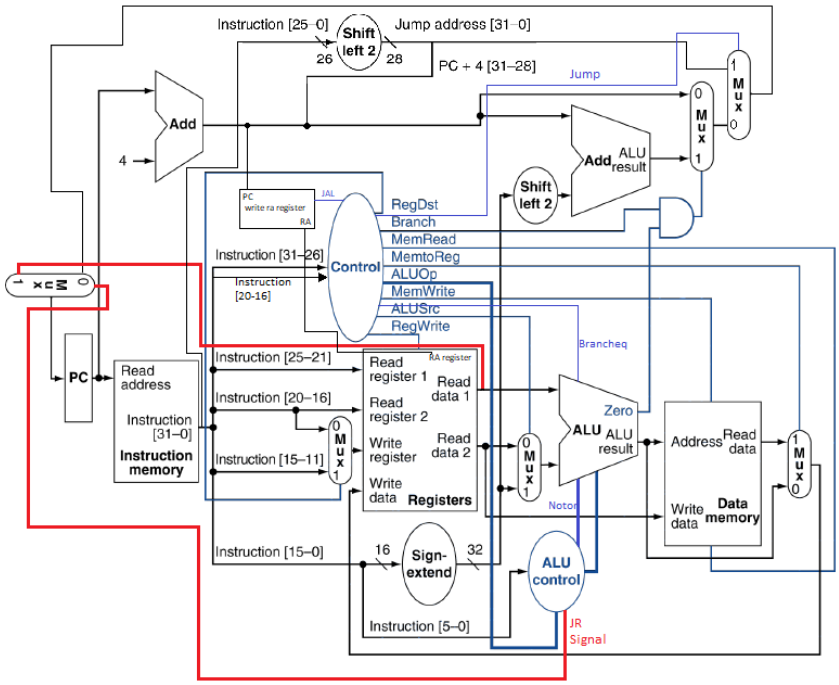
Arif Burak Demiray

Table Of Contents

Implementation Details

The implementation is very basic and easy to understand. I did not add more things to the datapath. I updated existing control unit, alu control unit, alu and processor modules.

Datapath



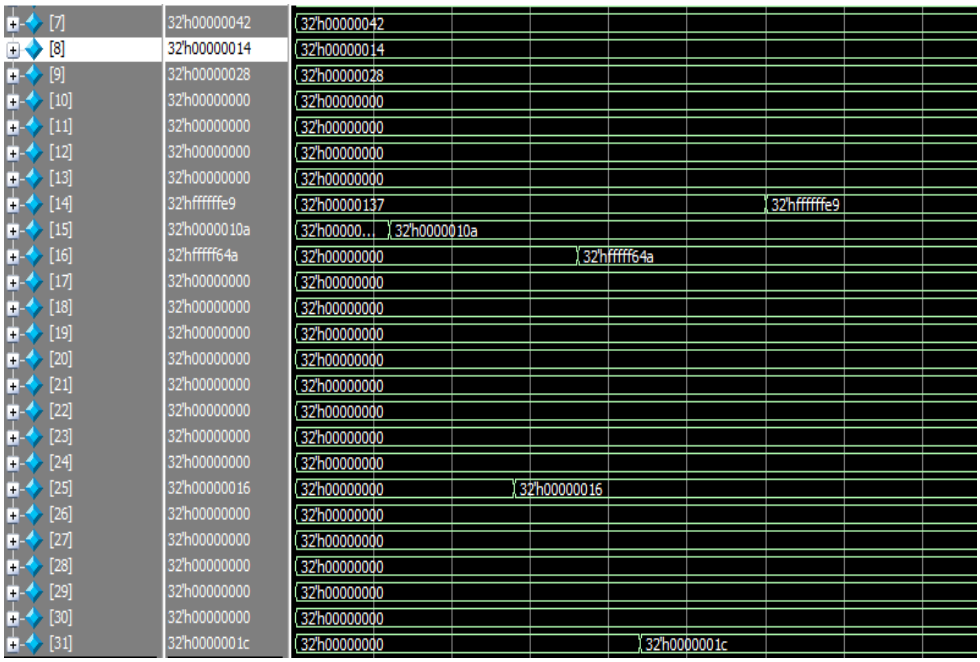
Here is the datapath. It is not complicated. I just added a copy to ra register unit. And there is a brancheq signal to alu, and from alu control jr signal to last mux. And not or signal to alu. Jump logic is same as in the last lesson pdf.

Control unit gives appropriate signals to execute instruction. Looks opcodes of instruction in there. Which is jal,j,bne,bgtz,bltz,blez,bgez,andi and ori. Addi signal is same as lw and sw signals. We just need addition for it. Nor and Jr signals are looked in alu control unit. Because, they are R-Type, and are looked easily in alu control unit because there are function code input. In control unit, there is additional alu op signal and rt field signal which is required for derived branch operations because in derived branch operation, rt field is continuation of op code. Control unit gives additional brancheq, jump and jal signals. Brancheq signal goes to alu unit to execute bgez and blez operations. Jump signal can execute jal and j. Goes to right top multiplexer. Jal signal goes to copy to ra unit to store pc in ra register which is 32nd register. In alu control there is additional output jr which jumps to given source register's value that is dataa in processor.v. It jumps it at left most multiplexer(In here I do not used multiplexer. I used a basic ternary operation to control it. I draw a mux to visualize it). Also, not or signal to alu to make not or operation in there. Not or just not version of or. It goes to same path with or. Control gives brancheq signal to alu mentioned above, bgez and bgzt have same alu op signals. This branch eq signal can control directly this situation. When there is brancheq signal it executes bgez. This logic is not differs for blez and bltz. Finally, there is not much different things from base code. There is additions for specific instructions. The code works for 8 instructions. And, there are three initIM.dat files to control different situations. There are screen shots of these files' executions. And there is assembly codes in the additional file for that instructions.

Sample Executions

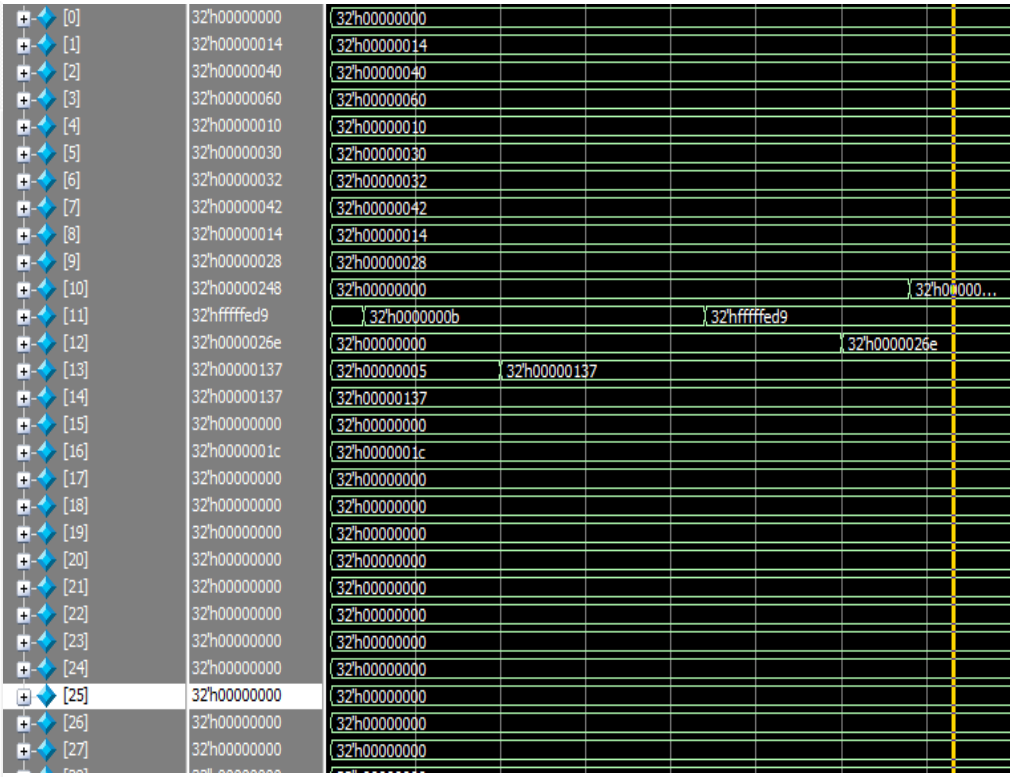
First execution is test1.asm I compiled it in qtspim to check results.

```
R0   [r0] = 0
R1   [at] = 0
R2   [v0] = a
R3   [v1] = 0
R4   [a0] = 10
R5   [a1] = 7ffff7f8
R6   [a2] = 7ffff800
R7   [a3] = 0
R8   [t0] = 0
R9   [t1] = 0
R10  [t2] = 0
R11  [t3] = 0
R12  [t4] = 0
R13  [t5] = 0
R14  [t6] = fffffe9
R15  [t7] = 10a
R16  [s0] = fffff64a
R17  [s1] = 0
R18  [s2] = 0
R19  [s3] = 0
R20  [s4] = 0
R21  [s5] = 0
R22  [s6] = 0
R23  [s7] = 0
R24  [t8] = 0
R25  [t9] = 16
R26  [k0] = 0
R27  [k1] = 0
R28  [gp] = 10008000
R29  [sp] = 7ffff7f4
R30  [s8] = 0
R31  [ra] = 400044
```



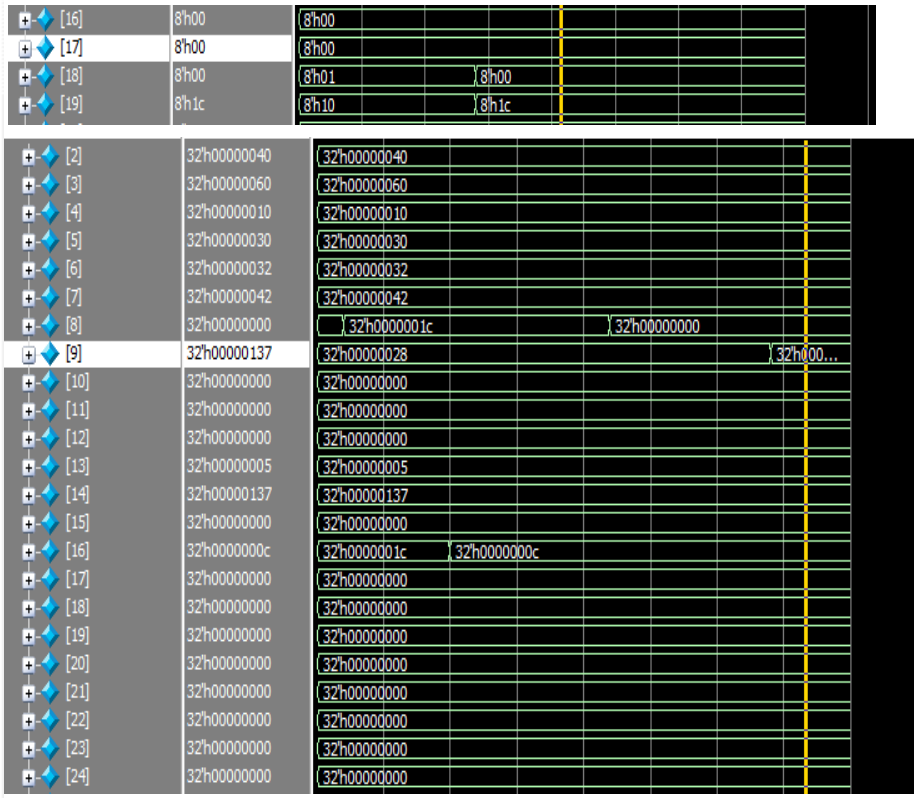
Here is the results for registers. RA register differs because PC numbers are different. But it stores true result because it jumps back to last instruction which is nor to t6. And its place seventh line which is true place to jump back. $a0 is defaul 10hex.

```
R0   [r0] = 0
R1   [at] = 0
R2   [v0] = a
R3   [v1] = 0
R4   [a0] = 10
R5   [a1] = 7ffff7f8
R6   [a2] = 7ffff800
R7   [a3] = 0
R8   [t0] = 0
R9   [t1] = 0
R10  [t2] = 248
R11  [t3] = fffffed9
R12  [t4] = 26e
R13  [t5] = 137
R14  [t6] = 137
R15  [t7] = 0
R16  [s0] = 0
R17  [s1] = 0
R18  [s2] = 0
R19  [s3] = 0
R20  [s4] = 0
R21  [s5] = 0
R22  [s6] = 0
R23  [s7] = 0
R24  [t8] = 0
R25  [t9] = 0
R26  [k0] = 0
R27  [k1] = 0
R28  [gp] = 10008000
R29  [sp] = 7ffff7f4
R30  [s8] = 0
R31  [ra] = 400018
```



Here is test2.asm results are same again. In DM[8] there is 00000137 , in reg file s0 28 and t5 5.

```
R0   [r0] = 0
R1   [at] = 0
R2   [v0] = a
R3   [v1] = 0
R4   [a0] = 1
R5   [a1] = 7ffff7f8
R6   [a2] = 7ffff800
R7   [a3] = 0
R8   [t0] = 0
R9   [t1] = 137
R10  [t2] = 0
R11  [t3] = 0
R12  [t4] = 0
R13  [t5] = 0
R14  [t6] = 0
R15  [t7] = 0
R16  [s0] = c
R17  [s1] = 1c
R18  [s2] = 0
R19  [s3] = 0
R20  [s4] = 0
R21  [s5] = 0
R22  [s6] = 0
R23  [s7] = 0
R24  [t8] = 0
R25  [t9] = 0
R26  [k0] = 0
R27  [k1] = 0
R28  [gp] = 10008000
R29  [sp] = 7ffff7f4
R30  [s8] = 0
R31  [ra] = 400018
```



Lastly this is the test3.asm file's results. Again they are same. The upper is data memory it writes t0 to 4 + s0. Why t0's value in s1. I wrote the cause of it to the test3.asm

References

https://www.eg.bucknell.edu/~csci320/mips_web/

https://www.cs.cmu.edu/afs/cs/academic/class/15740-f97/public/doc/mips-isa.pdf

https://lms.iyte.edu.tr/mod/resource/view.php?id=54692