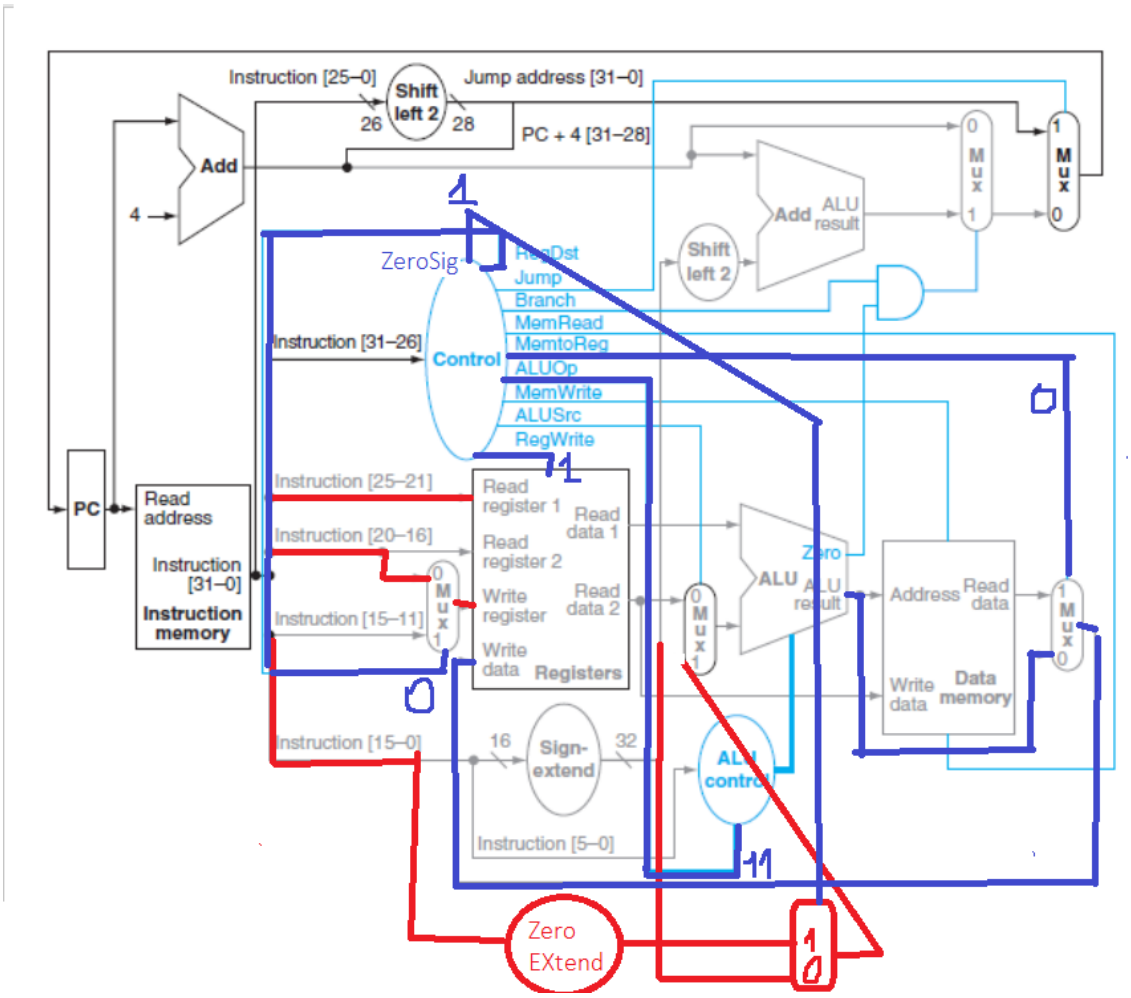
150117509 - Mustafa Abdullah Hakkoz
150115010 – Mahmut Aktaş
14.06.2020

COMPUTER ORGANIZATION PROJECT 2 REPORT

Instruction – 3: ori

I-type, opcode=13
ori $rt, $rs, Label → Put the logical OR of register $rs and the zero- extended immediate into register $qrt.



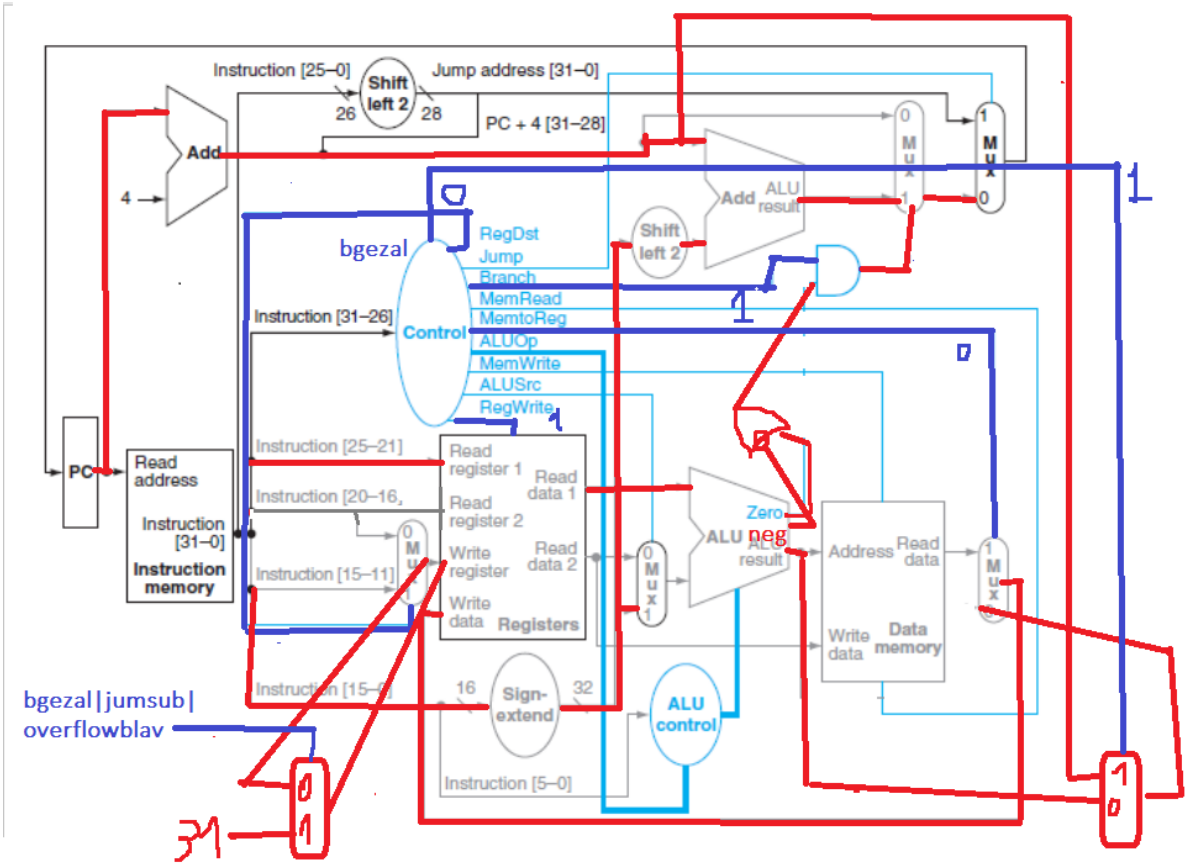
We start by reading two registers (rs and rt) along with a label code. An additional **zero extender** is implemented to extend 16 bits label to 32 bits. Output of zero extended will be headed to a **multiplexer** which is controlled by **ZeroSig.** After that it will go in to the multiplexer between Register File and ALU, instead of original sign extender. Rest of the path stays without changing. The result of ALU goes to the multiplexer directly without visiting **Data Memory** and finally ends in Write data input of the register file.

RegDst	Jump	Branch	MemRead	MemtoReg	ALUOp	MemWrite	ALUSrc	RegWrite	**ZeroSig**
0	x	x	x	0	11	x	x	1	1

Instruction – 6: bgezal

I-type, opcode=35

bgezal $rs, Label → if R[rs] >= 0, branch to PC-relative address (formed as beq & bne do), link address is stored in register 31.



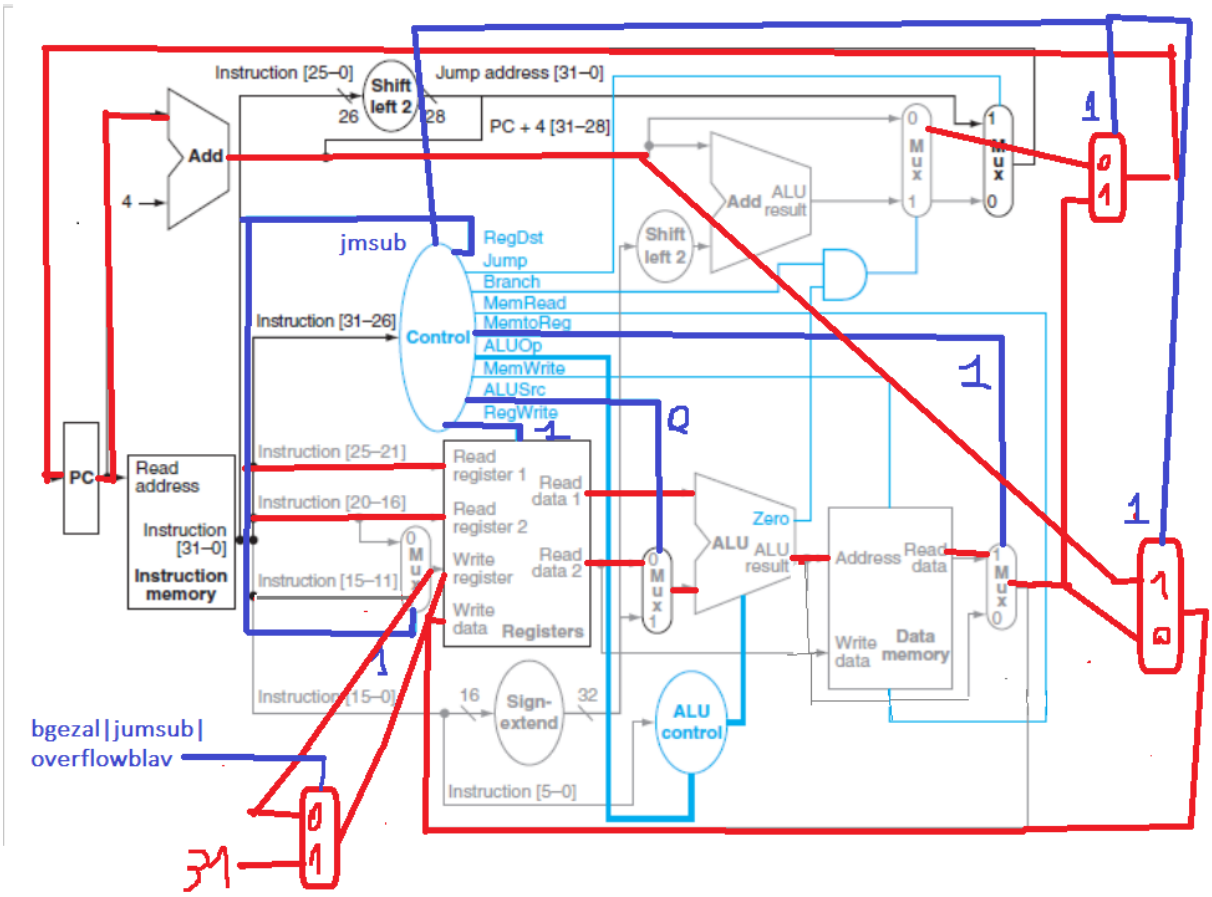
We start by reading rs and label from instruction code. Label is handed to **sign extender** first, then **shift left 2**. Resulting address is added to **pc+4**. This is passed to the Pc adress wire, if **rs is zero or positive**. This condition is controlled by **zero signal** and an additional **negative signal** of ALU. This path was for branch part of implementation.

For link part, we added a **multiplexer** to the output ot **ALU result** which is controlled by bgzeal signal. **First input** is coming from the **PC+4**, and the **second input** is coming from **ALU Result**. The output of this multiplexer will be going to the original multiplexer that comes after Data Memory and the resulting address will be written in to **register 31**. To implement it, we used another **extra multiplexer** controlled by relevant signal **bgzeal**.

RegDst	Jump	Branch	MemRead	MemtoReg	ALUOp	MemWrite	ALUSrc	RegWrite	**bgzeal**
0	x	1	x	0	xx	x	x	1	1

Instruction – 15: jmsub

R-type, funct=34
jmsub $rs,$rt → jumps to address found in memory [$rs-$rt], link address is stored in $31.



We start by reading **rs** and **rt** from **register file**. After that, they will go in to **ALU for substruction**. The resulting number will go in to **Data Memory**, relevant adress will be fetched from it and then handed to **two additional multiplexers** which are controlled by **jmsub signal**. **First multiplexer** will control the **link** implementation, **second multiplexer** will control the **jump** implementation.

Fort he first mux, inputs are **current PC+4 adress** and the **original output of Data memory**. Output of the mux will be headed to register file to be written **in register 31**. This behaviour is also controlled by a multiplexer same as **bgzeal** and **balv.**

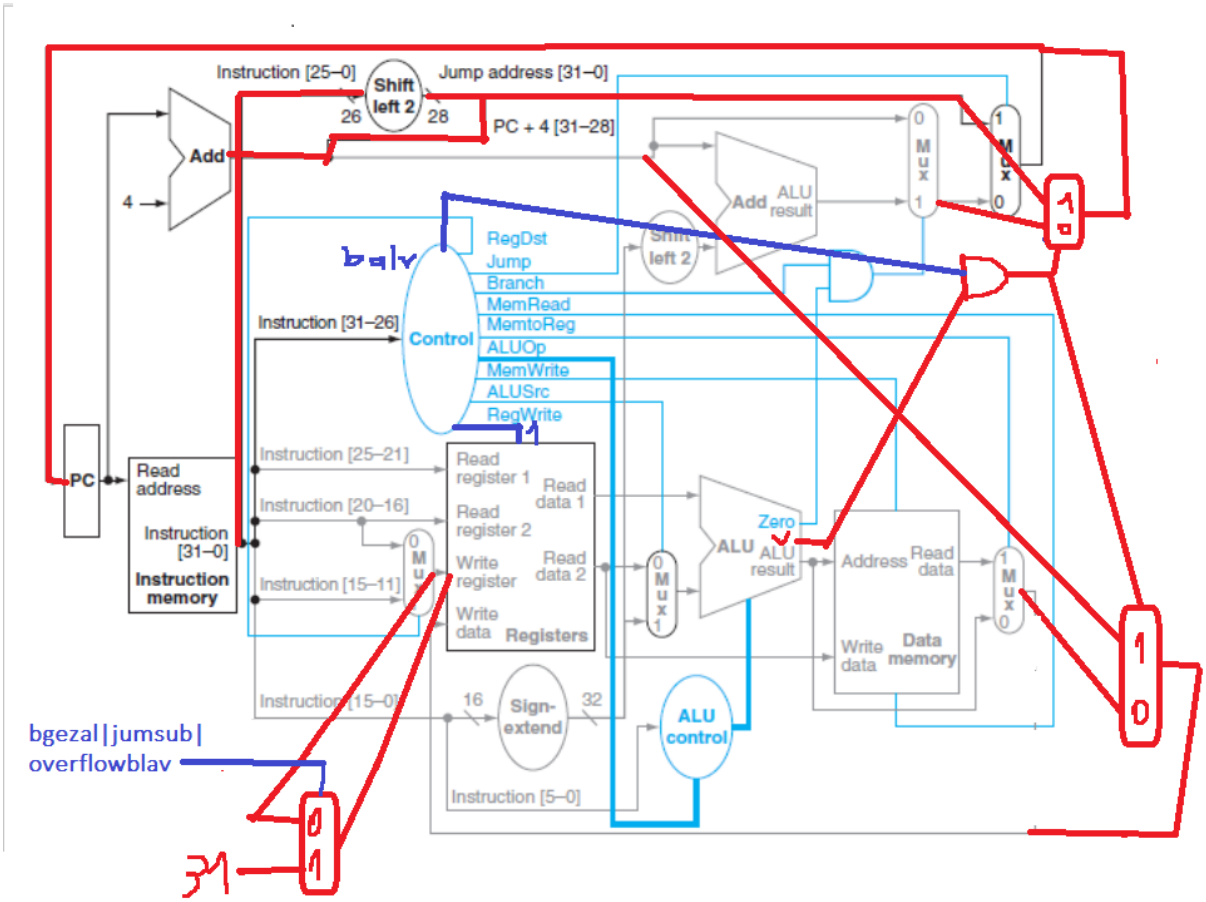
Fort he **second mux** which is controlling the jump part of the implementation, inputs are **branch path** and output of the **ALU-DataMem** path. Output of the multiplexer will be going to PC to become the adress to jump.

RegDst	Jump	Branch	MemRead	MemtoReg	ALUOp	MemWrite	ALUSrc	RegWrite	**jmsub**
1	x	x	x	1	xx	x	0	1	1

Instruction – 18: balv

J-type, opcode=33

balv Target → if Status [V] = 1, branches to pseudo-direct address (formed as jal does), link address is stored in register 31.



We start by reading 26-bits target adress. After **shifting** and concatenating to **PC+4**, it will headed to PC, if the **overflow status is 1**. To implement to calculation of overflow, we inserted necessary calculations for the cases of substruction and summation in to **ALU**.

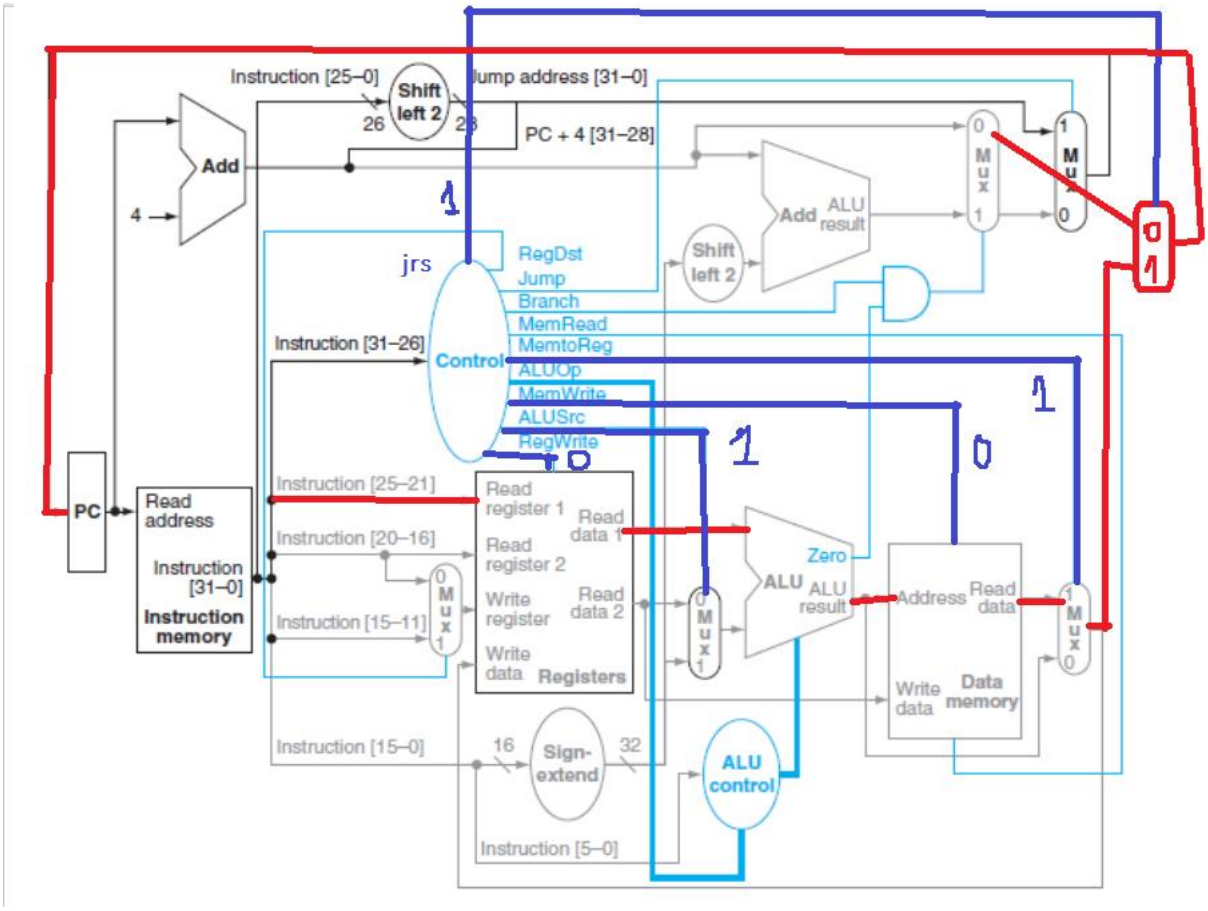
There's also **link** part of the datapath. The implementation of it is the same as **bgzeal** and **jmsub**. There will be an extra mux where the output of the original mux which controls ALU-DataMem path and the output of extra mux will be headed to register file to be written in register 31. This behaviour is also controlled by another multiplexer same as **bgzeal** and **balv.**

RegDst	Jump	Branch	MemRead	MemtoReg	ALUOp	MemWrite	ALUSrc	RegWrite	**balv**
x	x	x	x	x	xx	x	x	1	1

Instruction – 23: jrs

I-type, opcode=18

jrs $rs → jumps to address found in memory where the memory address is written in register $rs.



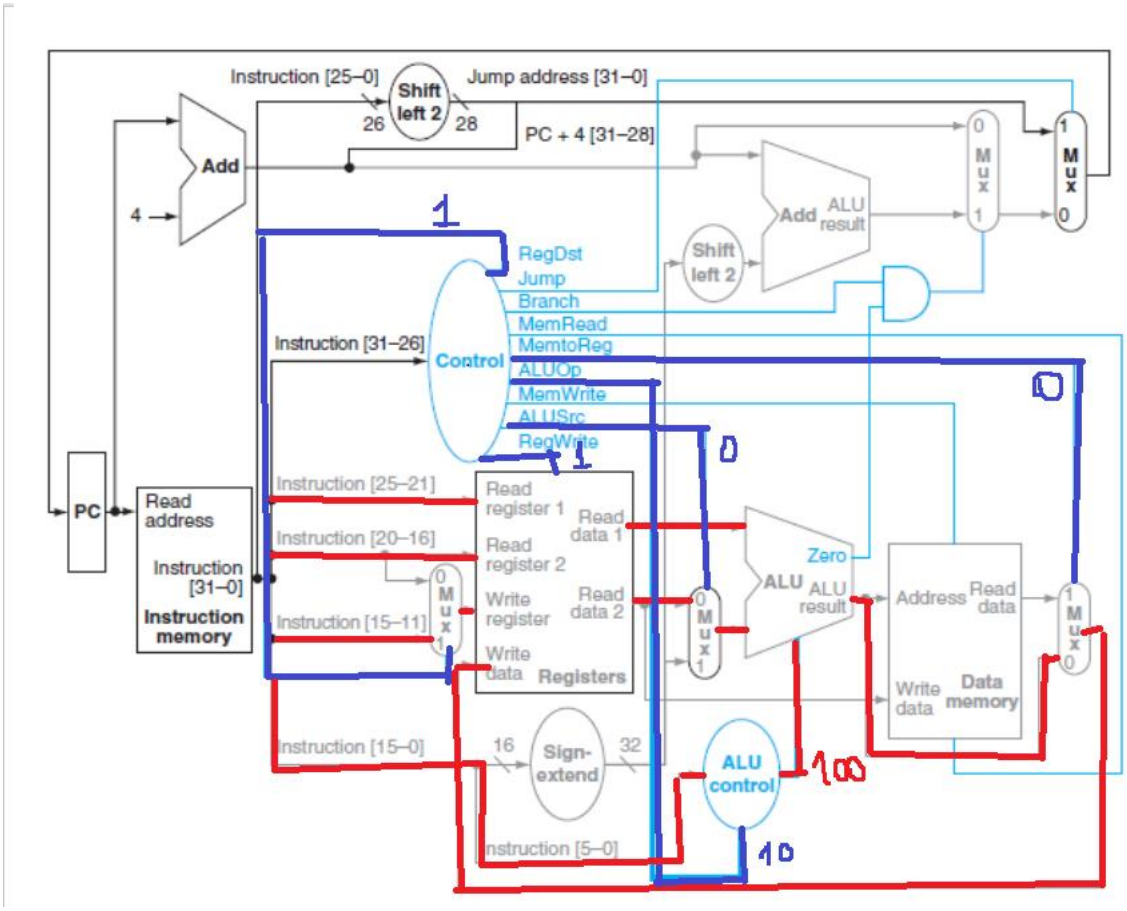
We start by reading **rs** from **registerFile**. The output value is the **Data Memory adress**. After reading the value in the Data Memory adress, we will jump to it.

RegDst	Jump	Branch	MemRead	MemtoReg	ALUOp	MemWrite	ALUSrc	RegWrite	**jrs**
x	x	x	x	1	xx	0	1	0	1

Instruction – 27: sllv

R-type, func=4

sllv $rd, $rt, $rs → shift register $rt to left by the value in register $rs, and store the result in register $rd.



We start by reading three registers, rt, rs and rd. Shifting will be occur in ALU, and the result will be written in register rt. We didn't implement any extra module or signal for this instruction.

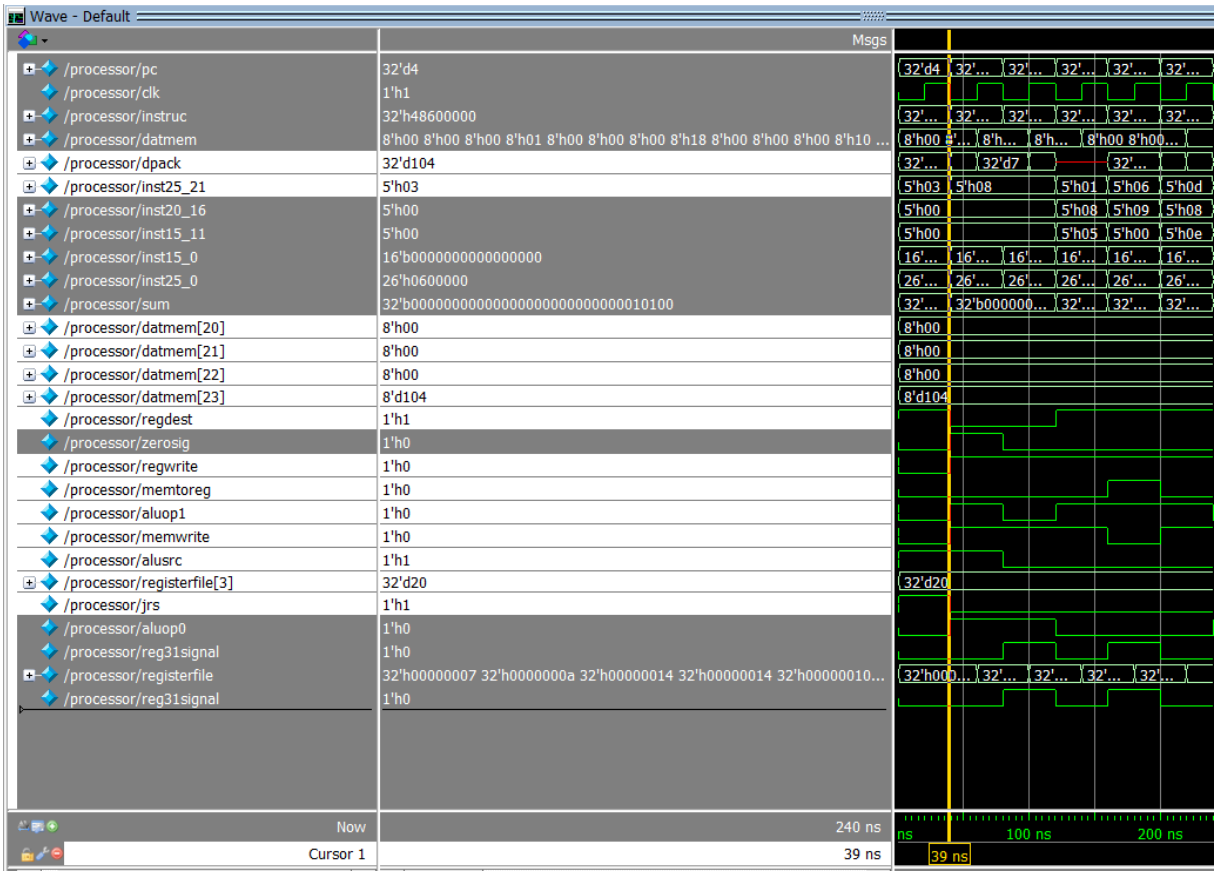
RegDst	Jump	Branch	MemRead	MemtoReg	ALUOp	MemWrite	ALUSrc	RegWrite
1	x	x	x	0	10	x	0	1

Example run:

We run the instructions in a order of:

1. **Jrs:**

   Instruction: 010010 00011 00000 0000000000000000
   Details: Jrs looks at the register3 and relevant data memory then jumps to the address written in it.
   register3 = 20, dm20 = 104, before_pc = 4, after_pc =104

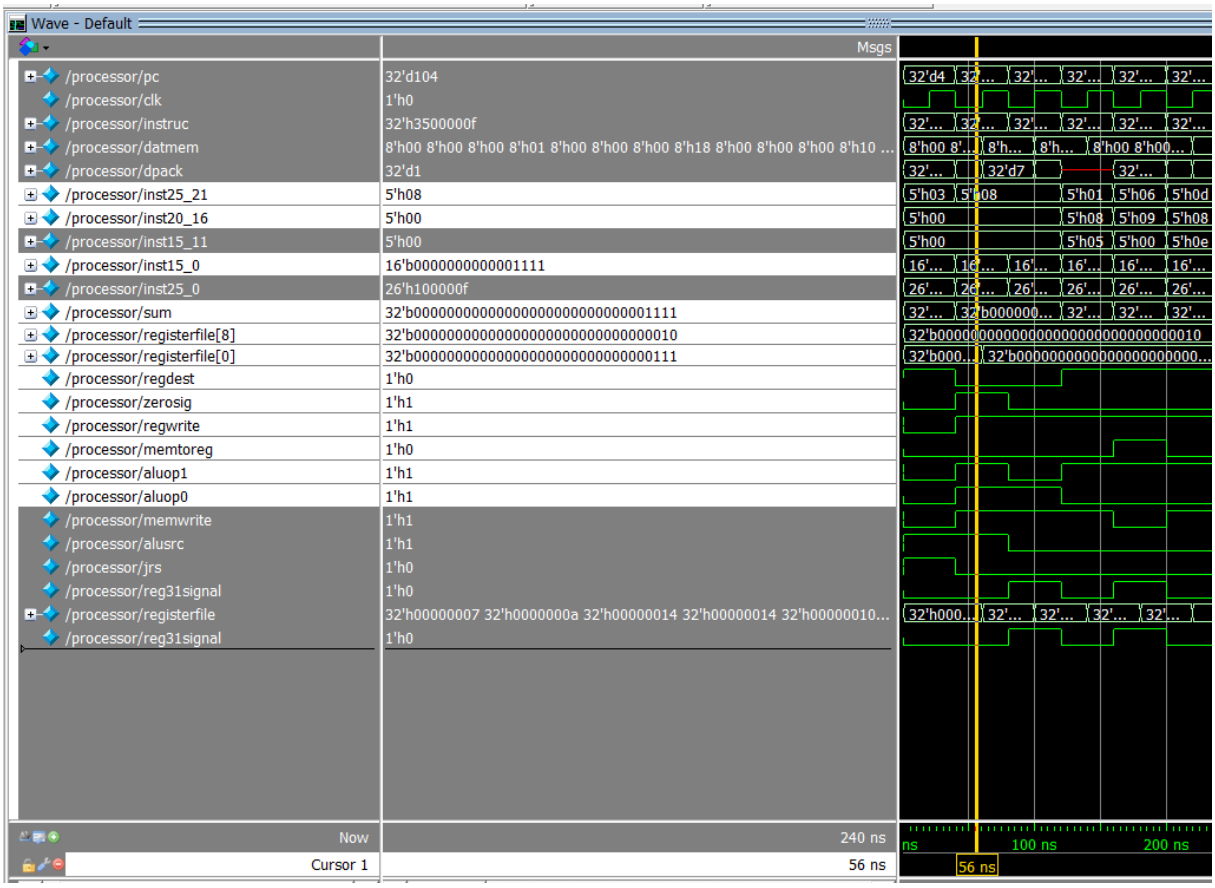


2. Ori:

Instruction: 001101 01000 00000 0000000000001111

Details: Ori looks at register8 and makes or calculation with the immediate value 15 and writes to register0.

Register8 = 2 before_register0=7, after_register0=15

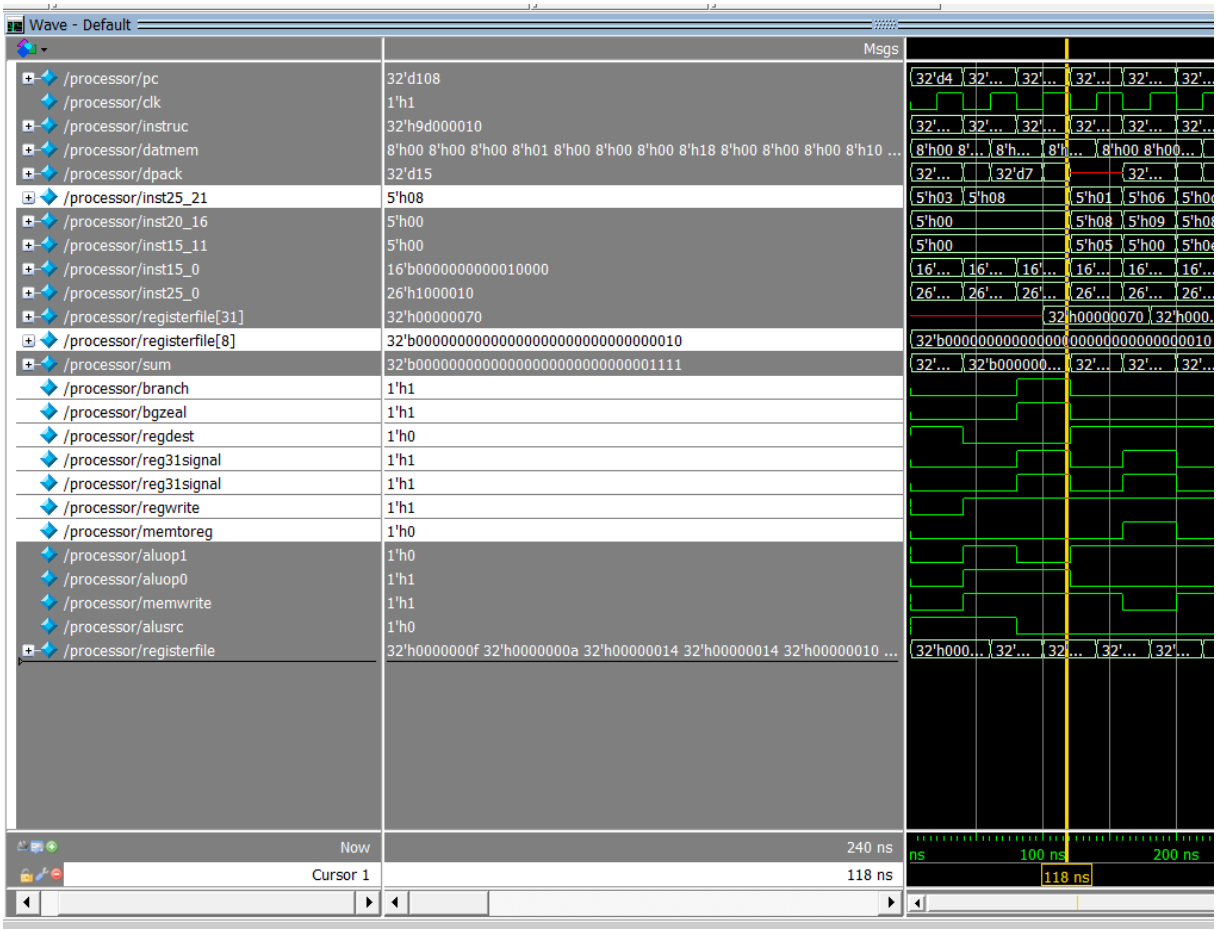


3. BGZEAL

Instruction: 100111 01000 00000 0000000000010000

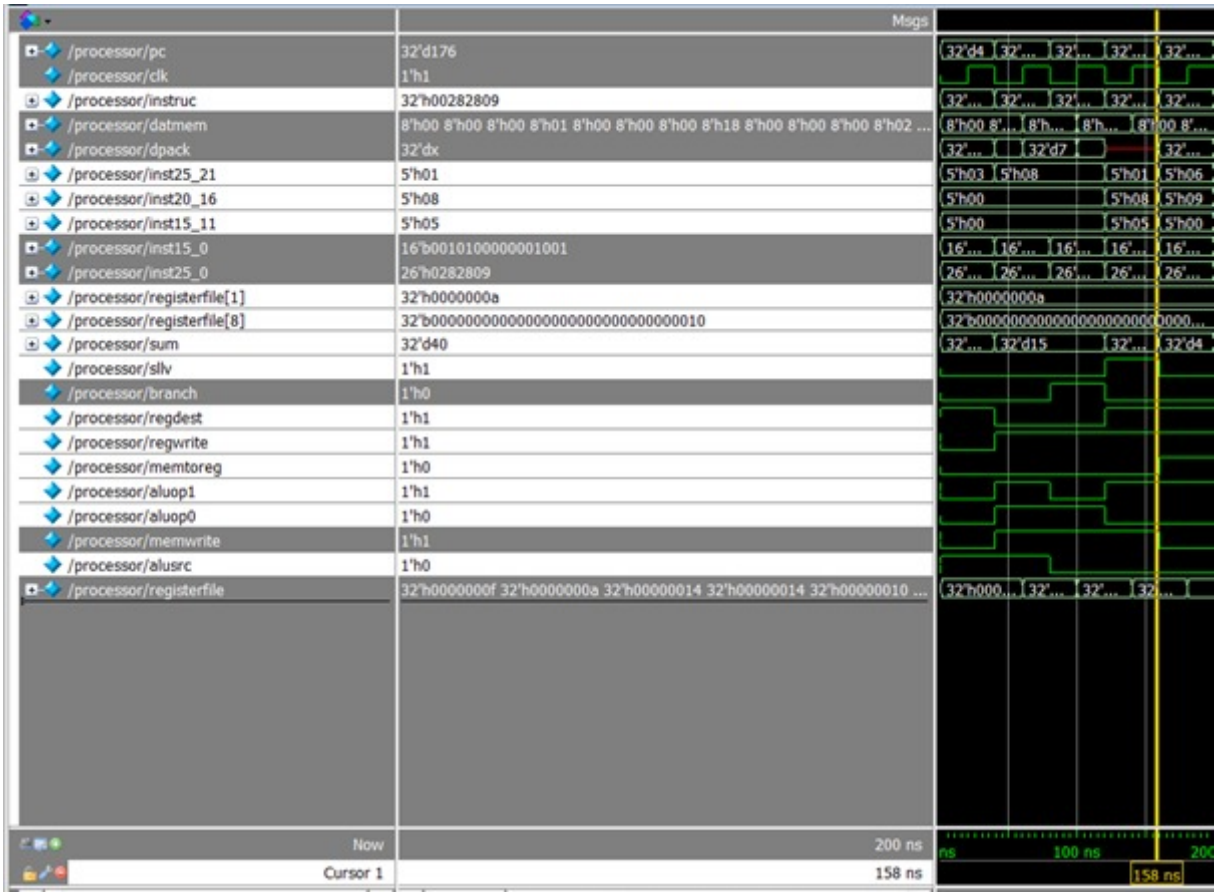
Details: Bgzeal looks at register 8, if its equal or greater than zero, branches to pc relative address and link to register31.

Register8 = 2, before_pc=108, after_pc= 176, before_register31=112, after_register31=108



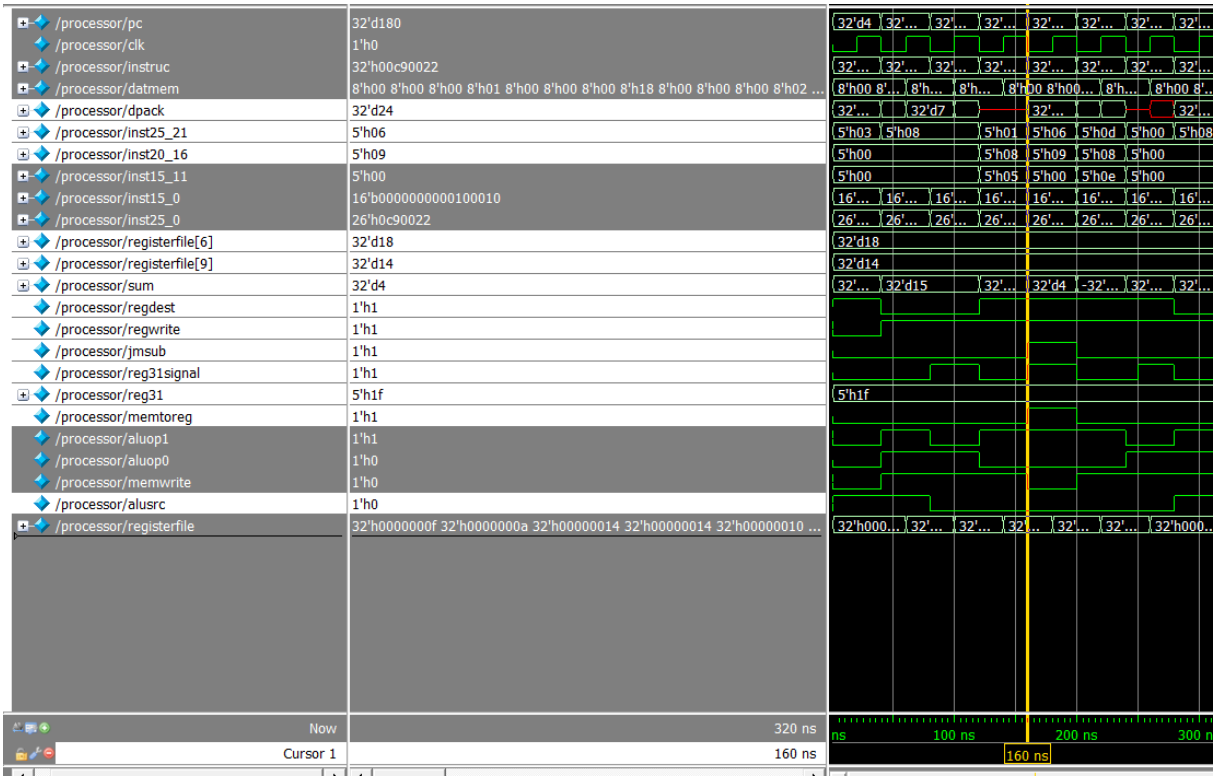
4 . SLLV

Instruction: 000000 00001 01000 00101 00000 001001
Details: Sllv shift register 1, with the value of register 8, and stores it in register 5. Register1 = 10, Register8 = 2, after calculation register5 = 40



5. JMSUB

Instruction: 000000 00110 01001 00000 00000 100010
Details: Jmsub reads values of register6 and register9, substract the values inside them, then fetches the value of datamemory, and finaly jumps to the relevant address.
register6 = 18, register9 = 14, datamemory4(dpack) = 24



6. ADD and BALV

Instruction: 000000 01101 01000 01110 00000 000000
Details: Add reads values of register13 and register8, sum the values inside them, then writes in register14.
register13 = 32h'7ffffff, register8 = 2, register14=overflow

Instruction: 100001 00000000000000000000001010
Details: Balv jumps to the label if previous instruction overflows. And links to register31.
overflowreg2 = 1, before_pc = 28, after_pc= 40, after_register31=32