Marmara University
Faculty of Engineering

CSE3038

Computer Organization

DATAPATH

Instructor: Haluk Topçuoğlu

Date: 25.05.2024

	Department	**Student Id Number**	**Name & Surname**
1	CSE	150120020	Mustafa Said ÇANAK
2	CSE	150120055	Muhammed Talha KARAGÜL
3	CSE	150121520	Ensar Muhammet YOZGAT
4	CSE	150121076	Abdullah KAN

Overview

The project involved extending the processor to support six new instructions: **brv**, **jmxor**, **nori**, **blezal**, **jalpc**, and **baln**. This required modifications to the control logic, the addition of new multiplexers, and the inclusion of status register flags to handle the specific behaviors of these instructions.

New Instructions

brv: R-type instruction that branches to the address in register $rs if the overflow status flag (V) is set.

jmxor: R-type instruction that jumps to the address found in memory at the address calculated by XORing $rs and $rt, and stores the link address in register $31.

nori: I-type instruction that performs a NOR operation between the value in $rs and a zero-extended immediate, storing the result in $rt.

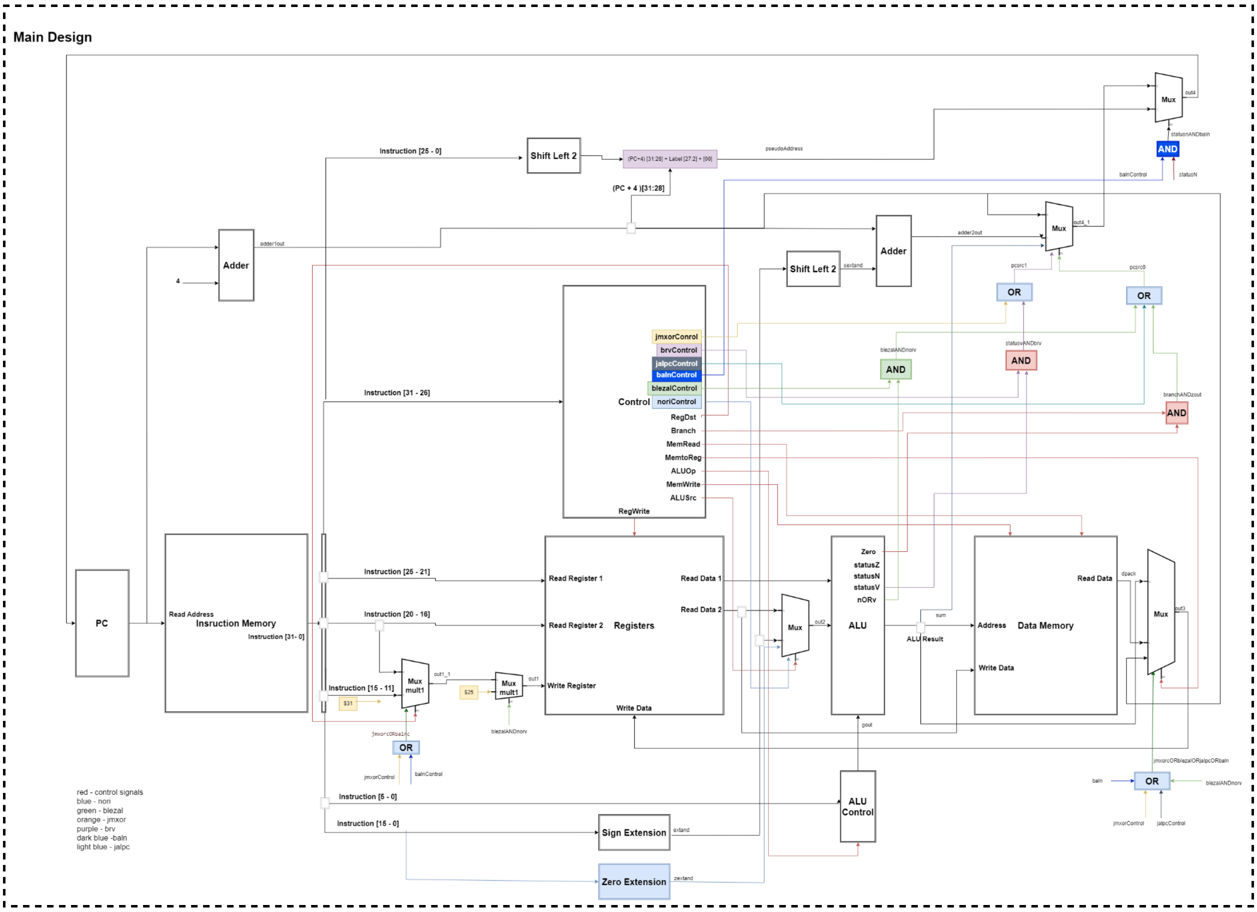
blezal: I-type instruction that branches to a PC-relative address if the value in $rs is less than or equal to zero, and stores the link address in register $25.

jalpc: I-type instruction that jumps to a PC-relative address and stores the link address in $rt.

baln: J-type instruction that branches to a pseudo-direct address if the negative status flag (N) is set, storing the link address in register $31.

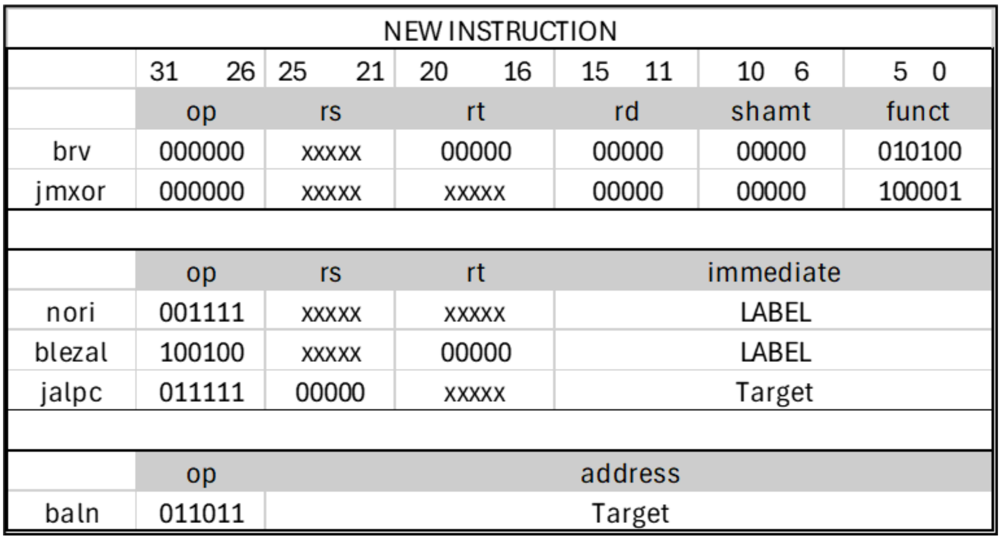
Modifications

Design



The new design is built on the foundation of a single-cycle datapath. Components necessary for Zero extension and pseudo direct addressing have been added to accommodate the new instructions. Status registers have been implemented within the ALU based on the values of the previous instruction. Instructions that use PC-relative addressing are connected to pcsrc0 via an OR gate, while jump instructions are connected to pcsrc1 via another OR gate. Multiplexers have been expanded as needed to ensure the proper operation of all instructions in the design. By providing the function code as an input to the control unit, control signal adjustments have been configured for the new R-type instructions. Additionally, the bit width of the ALUop signal has been increased from 2 to 3 bits.

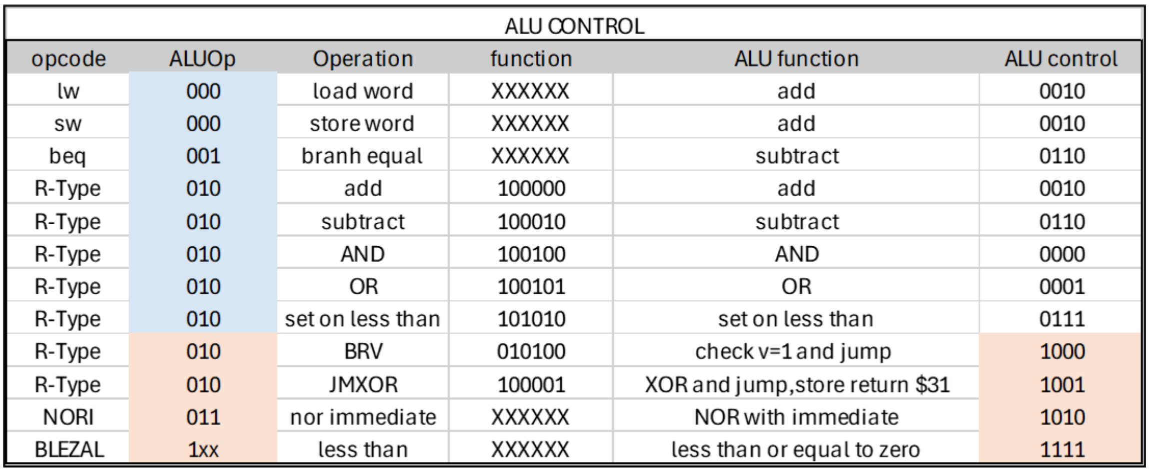
INSTRUCTION FORMAT DETAIL



NEW INSTRUCTION						
	31 26	25 21	20 16	15 11	10 6	5 0
	op	rs	rt	rd	shamt	funct
brv	000000	xxxxx	00000	00000	00000	010100
jmxor	000000	xxxxx	xxxxx	00000	00000	100001
	op	rs	rt	immediate		
nori	001111	xxxxx	xxxxx	LABEL		
blezal	100100	xxxxx	00000	LABEL		
jalpc	011111	00000	xxxxx	Target		
	op	address				
baln	011011	Target				

The parts used by the new instructions were determined according to their instruction formats.

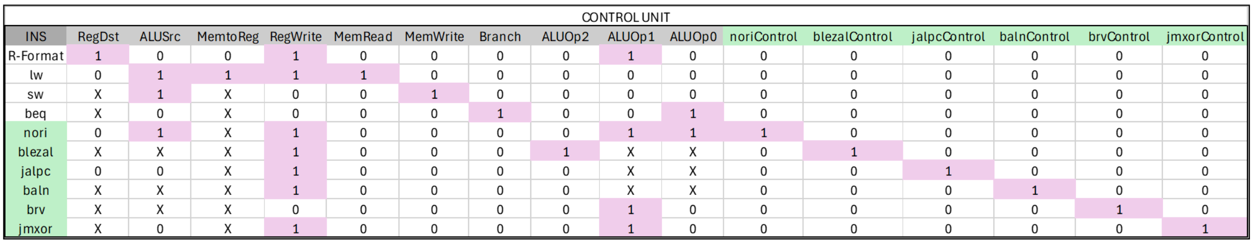
ALU CONTROL DETAILS



ALU CONTROL					
opcode	ALUOp	Operation	function	ALU function	ALU control
lw	000	load word	XXXXXX	add	0010
sw	000	store word	XXXXXX	add	0010
beq	001	branh equal	XXXXXX	subtract	0110
R-Type	010	add	100000	add	0010
R-Type	010	subtract	100010	subtract	0110
R-Type	010	AND	100100	AND	0000
R-Type	010	OR	100101	OR	0001
R-Type	010	set on less than	101010	set on less than	0111
R-Type	010	BRV	010100	check v=1 and jump	1000
R-Type	010	JMXOR	100001	XOR and jump,store return $31	1001
NORI	011	nor immediate	XXXXXX	NOR with immediate	1010
BLEZAL	1xx	less than	XXXXXX	less than or equal to zero	1111

The ALUop bit width was expanded to 3 bits. Accordingly, the signals coming from the ALU CONTROL were customized.

CONTROL UNIT SIGNALS



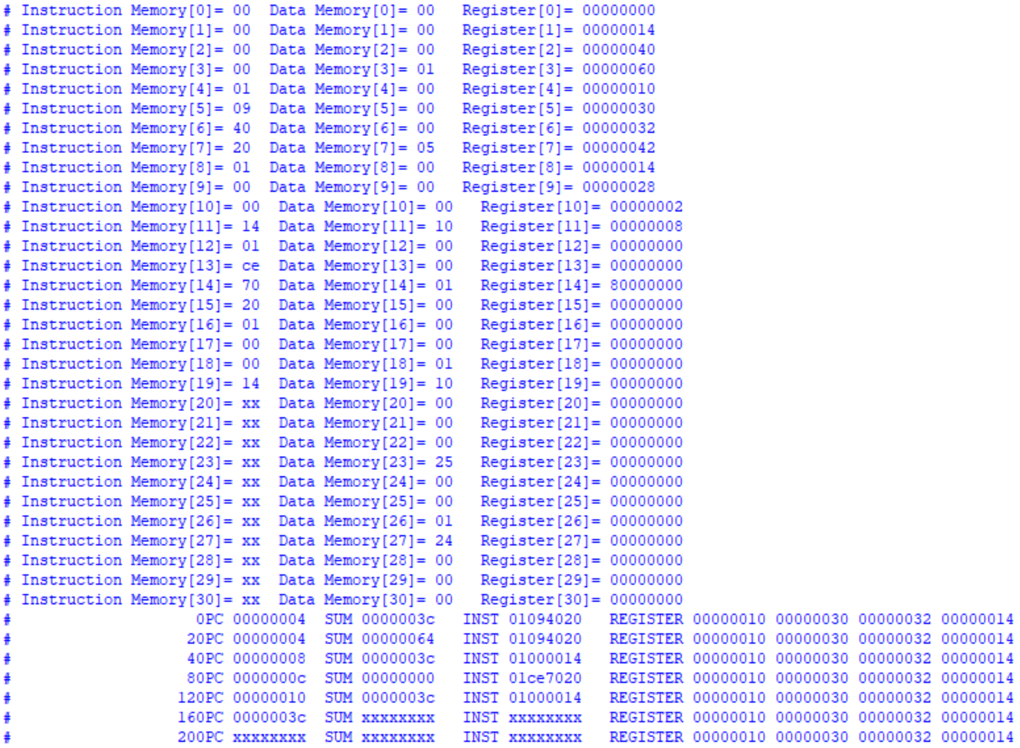
CONTROL UNIT																
INS	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp2	ALUOp1	ALUOp0	noriControl	blezalControl	jalpcControl	balnControl	brvControl	jmxorControl
R-Format	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0
lw	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
sw	X	1	X	0	0	1	0	0	0	0	0	0	0	0	0	0
beq	X	0	X	0	0	0	1	0	0	1	0	0	0	0	0	0
nori	0	1	X	1	0	0	0	0	1	1	1	0	0	0	0	0
blezal	X	X	X	1	0	0	0	1	X	X	0	1	0	0	0	0
jalpc	0	0	X	1	0	0	0	0	X	X	0	0	1	0	0	0
baln	X	X	X	1	0	0	0	0	X	X	0	0	0	1	0	0
brv	X	X	X	0	0	0	0	0	1	0	0	0	0	0	1	0
jmxor	X	0	X	1	0	0	0	0	1	0	0	0	0	0	0	1

The signals coming from the control unit were customized according to the instructions. This made the implementation of the control unit easier.

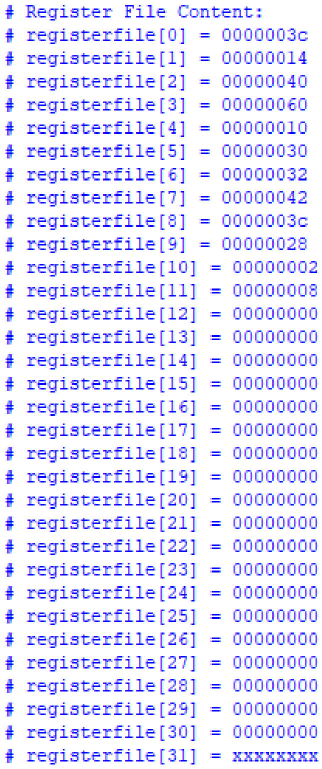
BRV

Insturcion: 01094020 → add $8, $8,$9
Insturcion: 01000014 → brv $8
Insturcion: 01ce7020 → add $14, $14, $14

In this example, our first instruction, add $8, $8, $9, accounts for the condition where there is no overflow. So, the next instruction, which is brv, will not jump to the address in $8, which is 0x0000003C. The next instruction, add $14, $14, $14, causes an overflow. This overflow is written to the statusV register for use in the following instruction. When the brv instruction is encountered again, it will jump to the value in register $8. Therefore, our new PC will be 0x0000003C.



```
# Instruction Memory[0]= 00  Data Memory[0]= 00   Register[0]= 00000000
# Instruction Memory[1]= 00  Data Memory[1]= 00   Register[1]= 00000014
# Instruction Memory[2]= 00  Data Memory[2]= 00   Register[2]= 00000040
# Instruction Memory[3]= 00  Data Memory[3]= 01   Register[3]= 00000060
# Instruction Memory[4]= 01  Data Memory[4]= 00   Register[4]= 00000010
# Instruction Memory[5]= 09  Data Memory[5]= 00   Register[5]= 00000030
# Instruction Memory[6]= 40  Data Memory[6]= 00   Register[6]= 00000032
# Instruction Memory[7]= 20  Data Memory[7]= 05   Register[7]= 00000042
# Instruction Memory[8]= 01  Data Memory[8]= 00   Register[8]= 00000014
# Instruction Memory[9]= 00  Data Memory[9]= 00   Register[9]= 00000028
# Instruction Memory[10]= 00  Data Memory[10]= 00   Register[10]= 00000002
# Instruction Memory[11]= 14  Data Memory[11]= 10   Register[11]= 00000008
# Instruction Memory[12]= 01  Data Memory[12]= 00   Register[12]= 00000000
# Instruction Memory[13]= ce  Data Memory[13]= 00   Register[13]= 00000000
# Instruction Memory[14]= 70  Data Memory[14]= 01   Register[14]= 80000000
# Instruction Memory[15]= 20  Data Memory[15]= 00   Register[15]= 00000000
# Instruction Memory[16]= 01  Data Memory[16]= 00   Register[16]= 00000000
# Instruction Memory[17]= 00  Data Memory[17]= 00   Register[17]= 00000000
# Instruction Memory[18]= 00  Data Memory[18]= 01   Register[18]= 00000000
# Instruction Memory[19]= 14  Data Memory[19]= 10   Register[19]= 00000000
# Instruction Memory[20]= xx  Data Memory[20]= 00   Register[20]= 00000000
# Instruction Memory[21]= xx  Data Memory[21]= 00   Register[21]= 00000000
# Instruction Memory[22]= xx  Data Memory[22]= 00   Register[22]= 00000000
# Instruction Memory[23]= xx  Data Memory[23]= 25   Register[23]= 00000000
# Instruction Memory[24]= xx  Data Memory[24]= 00   Register[24]= 00000000
# Instruction Memory[25]= xx  Data Memory[25]= 00   Register[25]= 00000000
# Instruction Memory[26]= xx  Data Memory[26]= 01   Register[26]= 00000000
# Instruction Memory[27]= xx  Data Memory[27]= 24   Register[27]= 00000000
# Instruction Memory[28]= xx  Data Memory[28]= 00   Register[28]= 00000000
# Instruction Memory[29]= xx  Data Memory[29]= 00   Register[29]= 00000000
# Instruction Memory[30]= xx  Data Memory[30]= 00   Register[30]= 00000000
#                     0PC 00000004  SUM 0000003c   INST 01094020   REGISTER 00000010 00000030 00000032 00000014
#                    20PC 00000004  SUM 00000064   INST 01094020   REGISTER 00000010 00000030 00000032 00000014
#                    40PC 00000008  SUM 0000003c   INST 01000014   REGISTER 00000010 00000030 00000032 00000014
#                    80PC 0000000c  SUM 00000000   INST 01ce7020   REGISTER 00000010 00000030 00000032 00000014
#                   120PC 00000010  SUM 0000003c   INST 01000014   REGISTER 00000010 00000030 00000032 00000014
#                   160PC 0000003c  SUM xxxxxxxx   INST xxxxxxxx   REGISTER 00000010 00000030 00000032 00000014
#                   200PC xxxxxxxx  SUM xxxxxxxx   INST xxxxxxxx   REGISTER 00000010 00000030 00000032 00000014
```



```
# Register File Content:
# registerfile[0] = 0000003c
# registerfile[1] = 00000014
# registerfile[2] = 00000040
# registerfile[3] = 00000060
# registerfile[4] = 00000010
# registerfile[5] = 00000030
# registerfile[6] = 00000032
# registerfile[7] = 00000042
# registerfile[8] = 0000003c
# registerfile[9] = 00000028
# registerfile[10] = 00000002
# registerfile[11] = 00000008
# registerfile[12] = 00000000
# registerfile[13] = 00000000
# registerfile[14] = 00000000
# registerfile[15] = 00000000
# registerfile[16] = 00000000
# registerfile[17] = 00000000
# registerfile[18] = 00000000
# registerfile[19] = 00000000
# registerfile[20] = 00000000
# registerfile[21] = 00000000
# registerfile[22] = 00000000
# registerfile[23] = 00000000
# registerfile[24] = 00000000
# registerfile[25] = 00000000
# registerfile[26] = 00000000
# registerfile[27] = 00000000
# registerfile[28] = 00000000
# registerfile[29] = 00000000
# registerfile[30] = 00000000
# registerfile[31] = xxxxxxxx
```

JMXOR

Insturcion: 014b0022 → jmxor $10, $11

The jmxor instruction jumps to the address obtained by applying the XOR operation to the values in registers $rs and $rt. The next address is stored in register $31.

For instance, consider the instruction 0x014B0021.

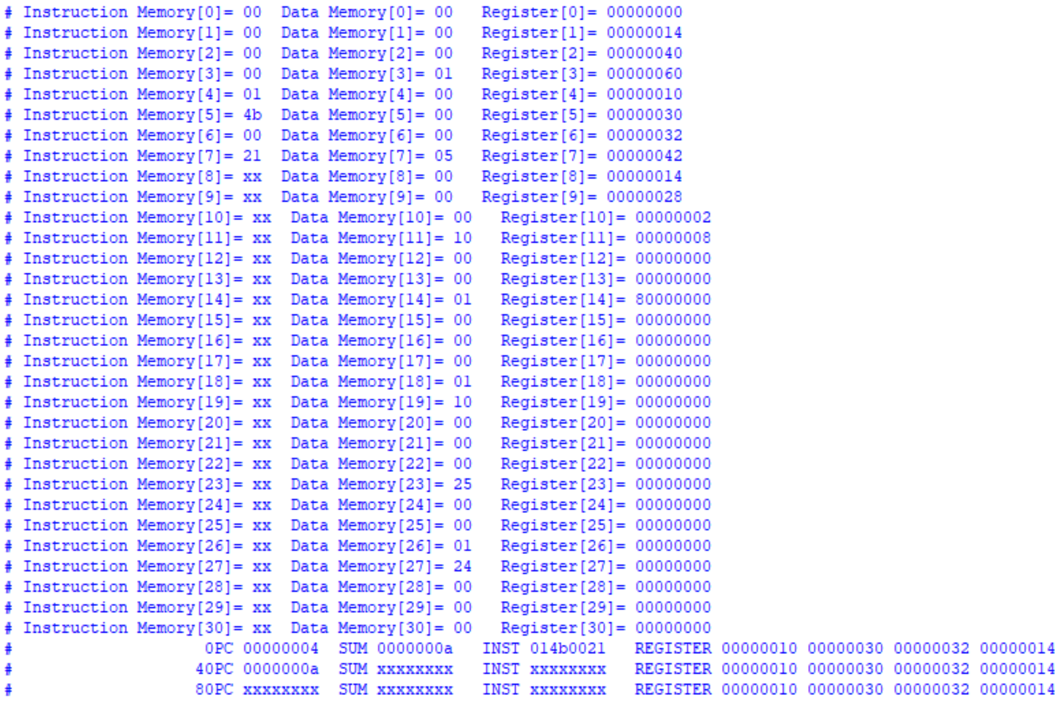
$rs is $10
    $10:   0000 0000 0000 0000 0000 0000 0000 0010
$rt is $11
    $11:   0000 0000 0000 0000 0000 0000 0000 1000

Performing the XOR operation:
    $10 :   0000 0000 0000 0000 0000 0000 0000 0010
    $11 :   0000 0000 0000 0000 0000 0000 0000 1000

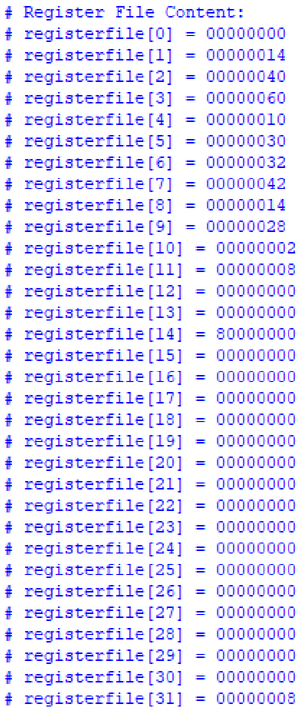
    XOR : 0000 0000 0000 0000 0000 0000 0000 1010 → 0x0000000A

Current PC: 0000 0000 0000 0000 0000 0000 0000 0100
PC + 4: 0000 0000 0000 0000 0000 0000 0000 1000 → 0x00000008 is stored in $31.

The program then jumps to 0x0000000A.



```
# Instruction Memory[0]= 00  Data Memory[0]= 00   Register[0]= 00000000
# Instruction Memory[1]= 00  Data Memory[1]= 00   Register[1]= 00000014
# Instruction Memory[2]= 00  Data Memory[2]= 00   Register[2]= 00000040
# Instruction Memory[3]= 00  Data Memory[3]= 01   Register[3]= 00000060
# Instruction Memory[4]= 01  Data Memory[4]= 00   Register[4]= 00000010
# Instruction Memory[5]= 4b  Data Memory[5]= 00   Register[5]= 00000030
# Instruction Memory[6]= 00  Data Memory[6]= 00   Register[6]= 00000032
# Instruction Memory[7]= 21  Data Memory[7]= 05   Register[7]= 00000042
# Instruction Memory[8]= xx  Data Memory[8]= 00   Register[8]= 00000014
# Instruction Memory[9]= xx  Data Memory[9]= 00   Register[9]= 00000028
# Instruction Memory[10]= xx  Data Memory[10]= 00   Register[10]= 00000002
# Instruction Memory[11]= xx  Data Memory[11]= 10   Register[11]= 00000008
# Instruction Memory[12]= xx  Data Memory[12]= 00   Register[12]= 00000000
# Instruction Memory[13]= xx  Data Memory[13]= 00   Register[13]= 00000000
# Instruction Memory[14]= xx  Data Memory[14]= 01   Register[14]= 80000000
# Instruction Memory[15]= xx  Data Memory[15]= 00   Register[15]= 00000000
# Instruction Memory[16]= xx  Data Memory[16]= 00   Register[16]= 00000000
# Instruction Memory[17]= xx  Data Memory[17]= 00   Register[17]= 00000000
# Instruction Memory[18]= xx  Data Memory[18]= 01   Register[18]= 00000000
# Instruction Memory[19]= xx  Data Memory[19]= 10   Register[19]= 00000000
# Instruction Memory[20]= xx  Data Memory[20]= 00   Register[20]= 00000000
# Instruction Memory[21]= xx  Data Memory[21]= 00   Register[21]= 00000000
# Instruction Memory[22]= xx  Data Memory[22]= 00   Register[22]= 00000000
# Instruction Memory[23]= xx  Data Memory[23]= 25   Register[23]= 00000000
# Instruction Memory[24]= xx  Data Memory[24]= 00   Register[24]= 00000000
# Instruction Memory[25]= xx  Data Memory[25]= 00   Register[25]= 00000000
# Instruction Memory[26]= xx  Data Memory[26]= 01   Register[26]= 00000000
# Instruction Memory[27]= xx  Data Memory[27]= 24   Register[27]= 00000000
# Instruction Memory[28]= xx  Data Memory[28]= 00   Register[28]= 00000000
# Instruction Memory[29]= xx  Data Memory[29]= 00   Register[29]= 00000000
# Instruction Memory[30]= xx  Data Memory[30]= 00   Register[30]= 00000000
#                     0PC 00000004  SUM 0000000a   INST 014b0021   REGISTER 00000010 00000030 00000032 00000014
#                    40PC 0000000a  SUM xxxxxxxx   INST xxxxxxxx   REGISTER 00000010 00000030 00000032 00000014
#                    80PC xxxxxxxx  SUM xxxxxxxx   INST xxxxxxxx   REGISTER 00000010 00000030 00000032 00000014
```



```
# Register File Content:
# registerfile[0] = 00000000
# registerfile[1] = 00000014
# registerfile[2] = 00000040
# registerfile[3] = 00000060
# registerfile[4] = 00000010
# registerfile[5] = 00000030
# registerfile[6] = 00000032
# registerfile[7] = 00000042
# registerfile[8] = 00000014
# registerfile[9] = 00000028
# registerfile[10] = 00000002
# registerfile[11] = 00000008
# registerfile[12] = 00000000
# registerfile[13] = 00000000
# registerfile[14] = 80000000
# registerfile[15] = 00000000
# registerfile[16] = 00000000
# registerfile[17] = 00000000
# registerfile[18] = 00000000
# registerfile[19] = 00000000
# registerfile[20] = 00000000
# registerfile[21] = 00000000
# registerfile[22] = 00000000
# registerfile[23] = 00000000
# registerfile[24] = 00000000
# registerfile[25] = 00000000
# registerfile[26] = 00000000
# registerfile[27] = 00000000
# registerfile[28] = 00000000
# registerfile[29] = 00000000
# registerfile[30] = 00000000
# registerfile[31] = 00000008
```

NORI

Instruction: 3dae894a → nori $14, $13, 0x894a
(001111 01101 011101 1001 1001 0100 1010)

The nori instruction applies the NOR operation to the data of the $rs register and the zero-extended label (0x894A), then writes the result to the $rt register.

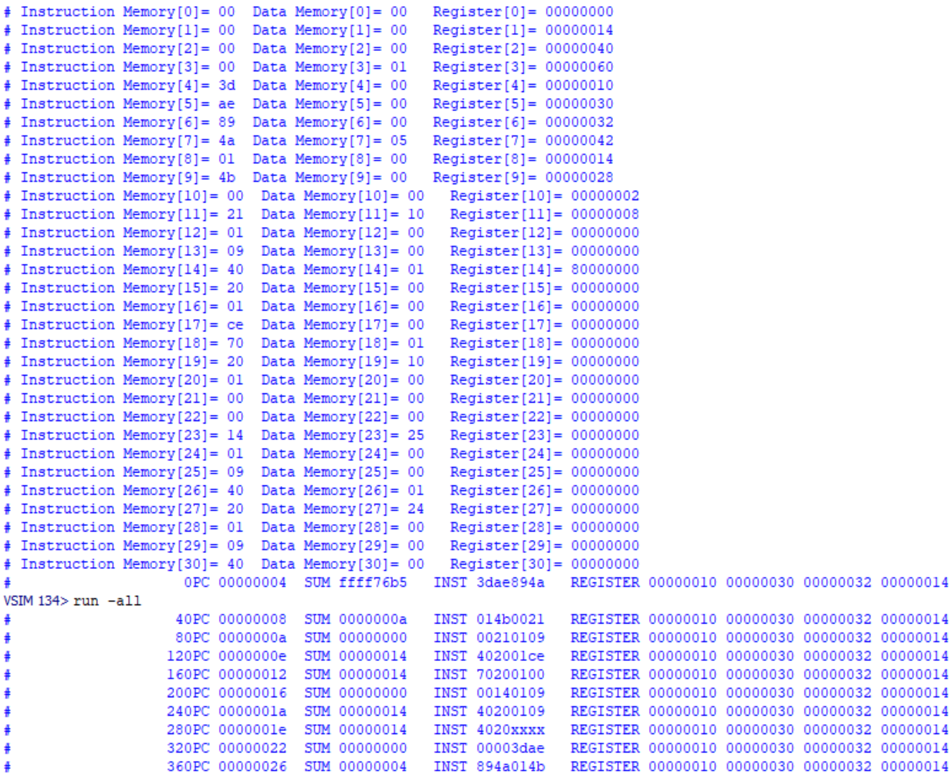
For instance, let's consider the instruction with the hex code 0x3DAE894A.

Label: 0x894A
$13 (rs): 0x00000000
$14 (rt): (result destination register)

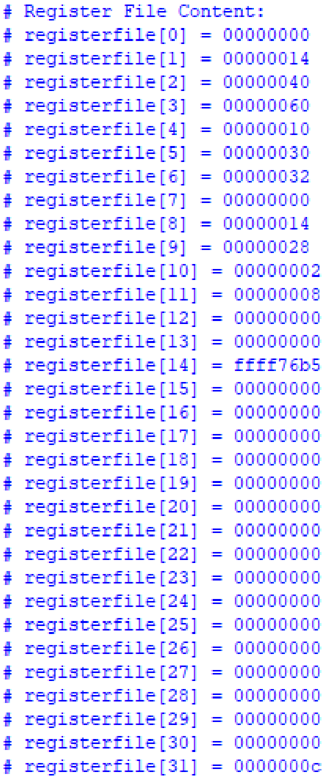
Performing the NOR operation:

```
            Label:    0000 0000 0000 0000 1000 1001 0100 1010
            $13 (rs): 0000 0000 0000 0000 0000 0000 0000 0000
-----------------------------------------------------------------------------------------
            NORI:     1111 1111 1111 1111 0111 0110 1011 0101  → 0xFFFF76B5
```

So, 0xFFFF76B5 is stored in $14 (rt).



```
# Instruction Memory[0]= 00  Data Memory[0]= 00   Register[0]= 00000000
# Instruction Memory[1]= 00  Data Memory[1]= 00   Register[1]= 00000014
# Instruction Memory[2]= 00  Data Memory[2]= 00   Register[2]= 00000040
# Instruction Memory[3]= 00  Data Memory[3]= 01   Register[3]= 00000060
# Instruction Memory[4]= 3d  Data Memory[4]= 00   Register[4]= 00000010
# Instruction Memory[5]= ae  Data Memory[5]= 00   Register[5]= 00000030
# Instruction Memory[6]= 89  Data Memory[6]= 00   Register[6]= 00000032
# Instruction Memory[7]= 4a  Data Memory[7]= 05   Register[7]= 00000042
# Instruction Memory[8]= 01  Data Memory[8]= 00   Register[8]= 00000014
# Instruction Memory[9]= 4b  Data Memory[9]= 00   Register[9]= 00000028
# Instruction Memory[10]= 00  Data Memory[10]= 00   Register[10]= 00000002
# Instruction Memory[11]= 21  Data Memory[11]= 10   Register[11]= 00000008
# Instruction Memory[12]= 01  Data Memory[12]= 00   Register[12]= 00000000
# Instruction Memory[13]= 09  Data Memory[13]= 00   Register[13]= 00000000
# Instruction Memory[14]= 40  Data Memory[14]= 01   Register[14]= 80000000
# Instruction Memory[15]= 20  Data Memory[15]= 00   Register[15]= 00000000
# Instruction Memory[16]= 01  Data Memory[16]= 00   Register[16]= 00000000
# Instruction Memory[17]= ce  Data Memory[17]= 00   Register[17]= 00000000
# Instruction Memory[18]= 70  Data Memory[18]= 01   Register[18]= 00000000
# Instruction Memory[19]= 20  Data Memory[19]= 10   Register[19]= 00000000
# Instruction Memory[20]= 01  Data Memory[20]= 00   Register[20]= 00000000
# Instruction Memory[21]= 00  Data Memory[21]= 00   Register[21]= 00000000
# Instruction Memory[22]= 00  Data Memory[22]= 00   Register[22]= 00000000
# Instruction Memory[23]= 14  Data Memory[23]= 25   Register[23]= 00000000
# Instruction Memory[24]= 01  Data Memory[24]= 00   Register[24]= 00000000
# Instruction Memory[25]= 09  Data Memory[25]= 00   Register[25]= 00000000
# Instruction Memory[26]= 40  Data Memory[26]= 01   Register[26]= 00000000
# Instruction Memory[27]= 20  Data Memory[27]= 24   Register[27]= 00000000
# Instruction Memory[28]= 01  Data Memory[28]= 00   Register[28]= 00000000
# Instruction Memory[29]= 09  Data Memory[29]= 00   Register[29]= 00000000
# Instruction Memory[30]= 40  Data Memory[30]= 00   Register[30]= 00000000
#                    0PC 00000004  SUM ffff76b5   INST 3dae894a   REGISTER 00000010 00000030 00000032 00000014
VSIM 134> run -all
#                   40PC 00000008  SUM 0000000a   INST 014b0021   REGISTER 00000010 00000030 00000032 00000014
#                   80PC 0000000a  SUM 00000000   INST 00210109   REGISTER 00000010 00000030 00000032 00000014
#                  120PC 0000000e  SUM 00000014   INST 402001ce   REGISTER 00000010 00000030 00000032 00000014
#                  160PC 00000012  SUM 00000014   INST 70200100   REGISTER 00000010 00000030 00000032 00000014
#                  200PC 00000016  SUM 00000000   INST 00140109   REGISTER 00000010 00000030 00000032 00000014
#                  240PC 0000001a  SUM 00000014   INST 40200109   REGISTER 00000010 00000030 00000032 00000014
#                  280PC 0000001e  SUM 00000014   INST 4020xxxx   REGISTER 00000010 00000030 00000032 00000014
#                  320PC 00000022  SUM 00000000   INST 00003dae   REGISTER 00000010 00000030 00000032 00000014
#                  360PC 00000026  SUM 00000004   INST 894a014b   REGISTER 00000010 00000030 00000032 00000014
```



```
# Register File Content:
# registerfile[0] = 00000000
# registerfile[1] = 00000014
# registerfile[2] = 00000040
# registerfile[3] = 00000060
# registerfile[4] = 00000010
# registerfile[5] = 00000030
# registerfile[6] = 00000032
# registerfile[7] = 00000000
# registerfile[8] = 00000014
# registerfile[9] = 00000028
# registerfile[10] = 00000002
# registerfile[11] = 00000008
# registerfile[12] = 00000000
# registerfile[13] = 00000000
# registerfile[14] = ffff76b5
# registerfile[15] = 00000000
# registerfile[16] = 00000000
# registerfile[17] = 00000000
# registerfile[18] = 00000000
# registerfile[19] = 00000000
# registerfile[20] = 00000000
# registerfile[21] = 00000000
# registerfile[22] = 00000000
# registerfile[23] = 00000000
# registerfile[24] = 00000000
# registerfile[25] = 00000000
# registerfile[26] = 00000000
# registerfile[27] = 00000000
# registerfile[28] = 00000000
# registerfile[29] = 00000000
# registerfile[30] = 00000000
# registerfile[31] = 0000000c
```

JALPC

Instruction: 7c0f0002 → jalpc $15, 0x0002

The Jalpc instruction jumps to a PC-relative address (BTA) and links the address to $rt.

For instance, let's consider the instruction with the hex code 0x7C0F0002.

0x0002 is our target address.
Target : 0000 0000 0000 0000 0000 0000 0000 0010

To go to PC-related address we must shift by two the target and add to the PC+4

PC+4 : 0000 0000 0000 0000 0000 0000 0000 0100
Target *4 : 0000 0000 0000 0000 0000 0000 0000 1000

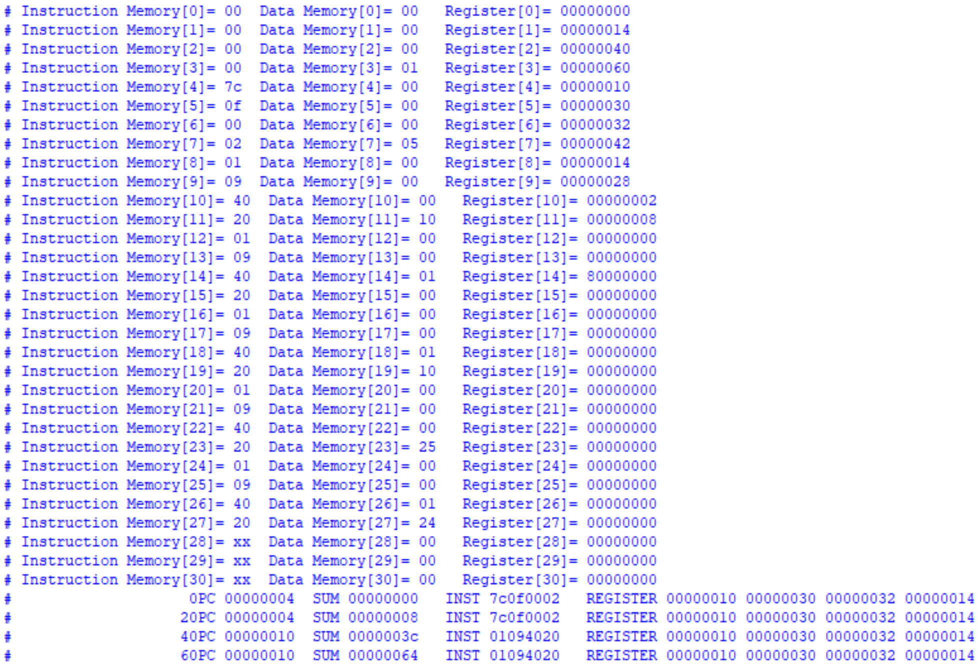
Total : 0000 0000 0000 0000 0000 0000 0000 1100

After determining the destination address we go there by adding 4:

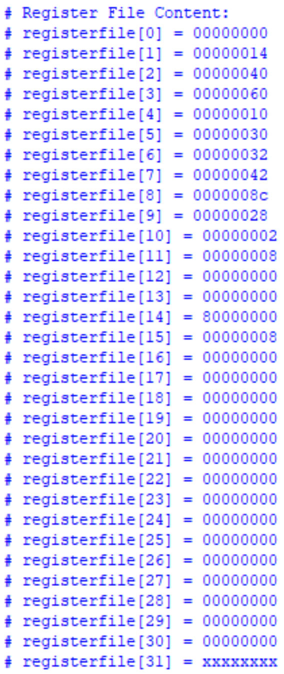
Next PC: 0000 0000 0000 0000 0000 0000 0001 0000 → 0x00000010

And the previous address is stored in $rt register which in our case it's $15 register and it's content is

PC+4 : 0000 0000 0000 0000 0000 0000 0000 1000 → 0x00000008



```
# Instruction Memory[0]= 00  Data Memory[0]= 00   Register[0]= 00000000
# Instruction Memory[1]= 00  Data Memory[1]= 00   Register[1]= 00000014
# Instruction Memory[2]= 00  Data Memory[2]= 00   Register[2]= 00000040
# Instruction Memory[3]= 00  Data Memory[3]= 01   Register[3]= 00000060
# Instruction Memory[4]= 7c  Data Memory[4]= 00   Register[4]= 00000010
# Instruction Memory[5]= 0f  Data Memory[5]= 00   Register[5]= 00000030
# Instruction Memory[6]= 00  Data Memory[6]= 00   Register[6]= 00000032
# Instruction Memory[7]= 02  Data Memory[7]= 05   Register[7]= 00000042
# Instruction Memory[8]= 01  Data Memory[8]= 00   Register[8]= 00000014
# Instruction Memory[9]= 09  Data Memory[9]= 00   Register[9]= 00000028
# Instruction Memory[10]= 40  Data Memory[10]= 00   Register[10]= 00000002
# Instruction Memory[11]= 20  Data Memory[11]= 10   Register[11]= 00000008
# Instruction Memory[12]= 01  Data Memory[12]= 00   Register[12]= 00000000
# Instruction Memory[13]= 09  Data Memory[13]= 00   Register[13]= 00000000
# Instruction Memory[14]= 40  Data Memory[14]= 01   Register[14]= 80000000
# Instruction Memory[15]= 20  Data Memory[15]= 00   Register[15]= 00000000
# Instruction Memory[16]= 01  Data Memory[16]= 00   Register[16]= 00000000
# Instruction Memory[17]= 09  Data Memory[17]= 00   Register[17]= 00000000
# Instruction Memory[18]= 40  Data Memory[18]= 01   Register[18]= 00000000
# Instruction Memory[19]= 20  Data Memory[19]= 10   Register[19]= 00000000
# Instruction Memory[20]= 01  Data Memory[20]= 00   Register[20]= 00000000
# Instruction Memory[21]= 09  Data Memory[21]= 00   Register[21]= 00000000
# Instruction Memory[22]= 40  Data Memory[22]= 00   Register[22]= 00000000
# Instruction Memory[23]= 20  Data Memory[23]= 25   Register[23]= 00000000
# Instruction Memory[24]= 01  Data Memory[24]= 00   Register[24]= 00000000
# Instruction Memory[25]= 09  Data Memory[25]= 00   Register[25]= 00000000
# Instruction Memory[26]= 40  Data Memory[26]= 01   Register[26]= 00000000
# Instruction Memory[27]= 20  Data Memory[27]= 24   Register[27]= 00000000
# Instruction Memory[28]= xx  Data Memory[28]= 00   Register[28]= 00000000
# Instruction Memory[29]= xx  Data Memory[29]= 00   Register[29]= 00000000
# Instruction Memory[30]= xx  Data Memory[30]= 00   Register[30]= 00000000
#                     0PC 00000004  SUM 00000000   INST 7c0f0002   REGISTER 00000010 00000030 00000032 00000014
#                    20PC 00000004  SUM 00000008   INST 7c0f0002   REGISTER 00000010 00000030 00000032 00000014
#                    40PC 00000010  SUM 0000003c   INST 01094020   REGISTER 00000010 00000030 00000032 00000014
#                    60PC 00000010  SUM 00000064   INST 01094020   REGISTER 00000010 00000030 00000032 00000014
```



```
# Register File Content:
# registerfile[0] = 00000000
# registerfile[1] = 00000014
# registerfile[2] = 00000040
# registerfile[3] = 00000060
# registerfile[4] = 00000010
# registerfile[5] = 00000030
# registerfile[6] = 00000032
# registerfile[7] = 00000042
# registerfile[8] = 0000008c
# registerfile[9] = 00000028
# registerfile[10] = 00000002
# registerfile[11] = 00000008
# registerfile[12] = 00000000
# registerfile[13] = 00000000
# registerfile[14] = 80000000
# registerfile[15] = 00000008
# registerfile[16] = 00000000
# registerfile[17] = 00000000
# registerfile[18] = 00000000
# registerfile[19] = 00000000
# registerfile[20] = 00000000
# registerfile[21] = 00000000
# registerfile[22] = 00000000
# registerfile[23] = 00000000
# registerfile[24] = 00000000
# registerfile[25] = 00000000
# registerfile[26] = 00000000
# registerfile[27] = 00000000
# registerfile[28] = 00000000
# registerfile[29] = 00000000
# registerfile[30] = 00000000
# registerfile[31] = xxxxxxxx
```

BLEZAL

Instruction: 0x91c00004 → blezal $14, 0x0004
Instruction: 0x91400004 → blezal $10, 0x0004
Instruction: 0x91800004 → blezal $12, 0x0004

Blezal instruction checks if R[rs] is smaller or equal to 0 if it is it jumps to do PC-relative address. The link address is stored $25 register.

For instance,

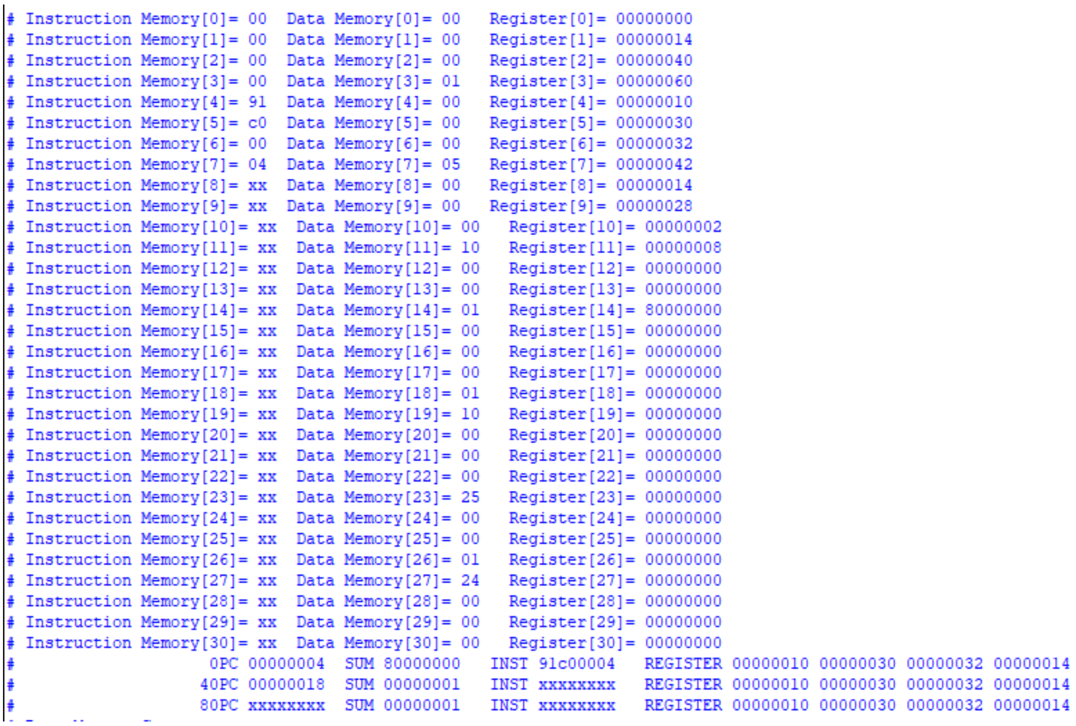
Case 1

Let's consider the instruction with the hex code 0x91C00004.

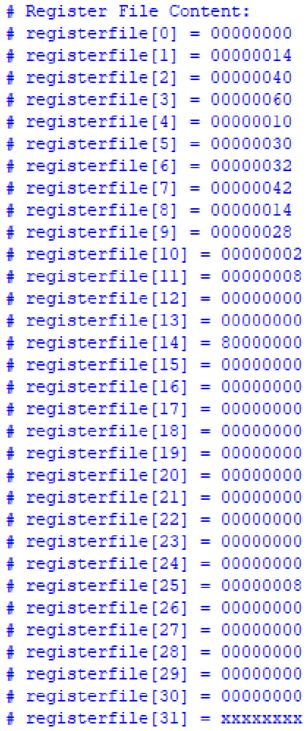
$14 (rs) : 1000 0000 0000 0000 0000 0000 0000 → negative so instruction will jump to new address
LABEL: 0000 0000 0000 0000 0000 0000 0100

PC+4 : 0000 0000 0000 0000 0000 0000 1000 → 0x00000008 stored in $25 register
LABEL*4: 0000 0000 0000 0000 0000 0001 0000

New PC : 0000 0000 0000 0000 0000 0001 1000 → 0x00000018



```
# Instruction Memory[0]= 00  Data Memory[0]= 00   Register[0]= 00000000
# Instruction Memory[1]= 00  Data Memory[1]= 00   Register[1]= 00000014
# Instruction Memory[2]= 00  Data Memory[2]= 00   Register[2]= 00000040
# Instruction Memory[3]= 00  Data Memory[3]= 01   Register[3]= 00000060
# Instruction Memory[4]= 91  Data Memory[4]= 00   Register[4]= 00000010
# Instruction Memory[5]= c0  Data Memory[5]= 00   Register[5]= 00000030
# Instruction Memory[6]= 00  Data Memory[6]= 00   Register[6]= 00000032
# Instruction Memory[7]= 04  Data Memory[7]= 05   Register[7]= 00000042
# Instruction Memory[8]= xx  Data Memory[8]= 00   Register[8]= 00000014
# Instruction Memory[9]= xx  Data Memory[9]= 00   Register[9]= 00000028
# Instruction Memory[10]= xx  Data Memory[10]= 00   Register[10]= 00000002
# Instruction Memory[11]= xx  Data Memory[11]= 10   Register[11]= 00000008
# Instruction Memory[12]= xx  Data Memory[12]= 00   Register[12]= 00000000
# Instruction Memory[13]= xx  Data Memory[13]= 00   Register[13]= 00000000
# Instruction Memory[14]= xx  Data Memory[14]= 01   Register[14]= 80000000
# Instruction Memory[15]= xx  Data Memory[15]= 00   Register[15]= 00000000
# Instruction Memory[16]= xx  Data Memory[16]= 00   Register[16]= 00000000
# Instruction Memory[17]= xx  Data Memory[17]= 00   Register[17]= 00000000
# Instruction Memory[18]= xx  Data Memory[18]= 01   Register[18]= 00000000
# Instruction Memory[19]= xx  Data Memory[19]= 10   Register[19]= 00000000
# Instruction Memory[20]= xx  Data Memory[20]= 00   Register[20]= 00000000
# Instruction Memory[21]= xx  Data Memory[21]= 00   Register[21]= 00000000
# Instruction Memory[22]= xx  Data Memory[22]= 00   Register[22]= 00000000
# Instruction Memory[23]= xx  Data Memory[23]= 25   Register[23]= 00000000
# Instruction Memory[24]= xx  Data Memory[24]= 00   Register[24]= 00000000
# Instruction Memory[25]= xx  Data Memory[25]= 00   Register[25]= 00000000
# Instruction Memory[26]= xx  Data Memory[26]= 01   Register[26]= 00000000
# Instruction Memory[27]= xx  Data Memory[27]= 24   Register[27]= 00000000
# Instruction Memory[28]= xx  Data Memory[28]= 00   Register[28]= 00000000
# Instruction Memory[29]= xx  Data Memory[29]= 00   Register[29]= 00000000
# Instruction Memory[30]= xx  Data Memory[30]= 00   Register[30]= 00000000
#                     0PC 00000004  SUM 80000000   INST 91c00004   REGISTER 00000010 00000030 00000032 00000014
#                    40PC 00000018  SUM 00000001   INST xxxxxxxx   REGISTER 00000010 00000030 00000032 00000014
#                    80PC xxxxxxxx  SUM 00000001   INST xxxxxxxx   REGISTER 00000010 00000030 00000032 00000014
```



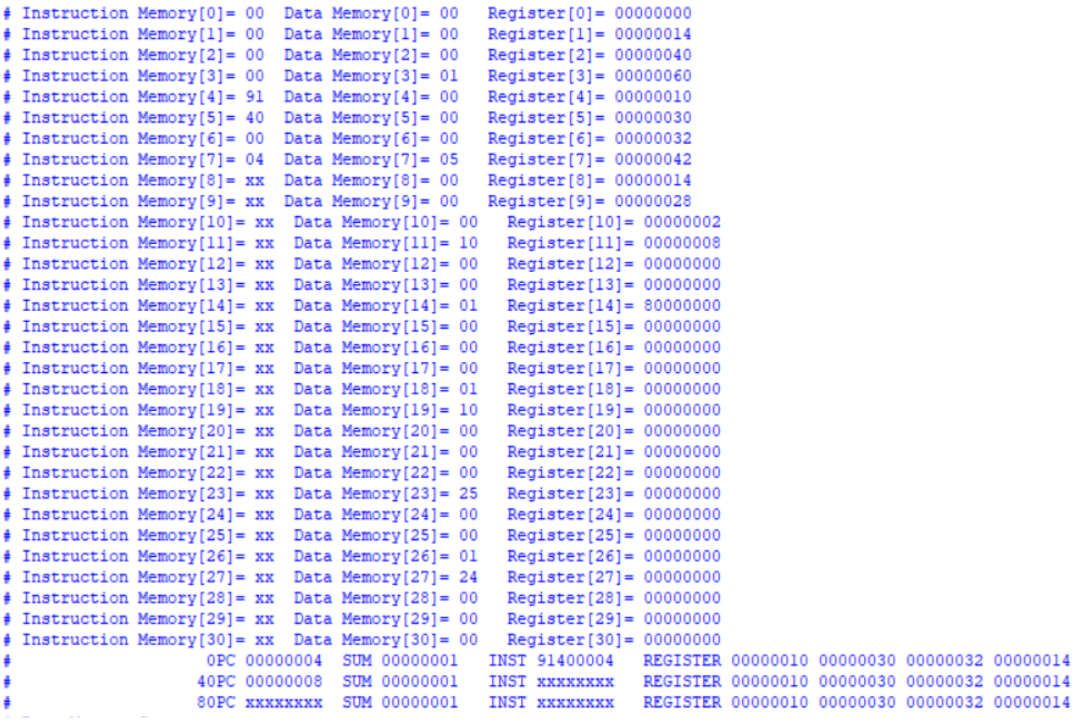
```
# Register File Content:
# registerfile[0] = 00000000
# registerfile[1] = 00000014
# registerfile[2] = 00000040
# registerfile[3] = 00000060
# registerfile[4] = 00000010
# registerfile[5] = 00000030
# registerfile[6] = 00000032
# registerfile[7] = 00000042
# registerfile[8] = 00000014
# registerfile[9] = 00000028
# registerfile[10] = 00000002
# registerfile[11] = 00000008
# registerfile[12] = 00000000
# registerfile[13] = 00000000
# registerfile[14] = 80000000
# registerfile[15] = 00000000
# registerfile[16] = 00000000
# registerfile[17] = 00000000
# registerfile[18] = 00000000
# registerfile[19] = 00000000
# registerfile[20] = 00000000
# registerfile[21] = 00000000
# registerfile[22] = 00000000
# registerfile[23] = 00000000
# registerfile[24] = 00000000
# registerfile[25] = 00000008
# registerfile[26] = 00000000
# registerfile[27] = 00000000
# registerfile[28] = 00000000
# registerfile[29] = 00000000
# registerfile[30] = 00000000
# registerfile[31] = xxxxxxxx
```

Case 2

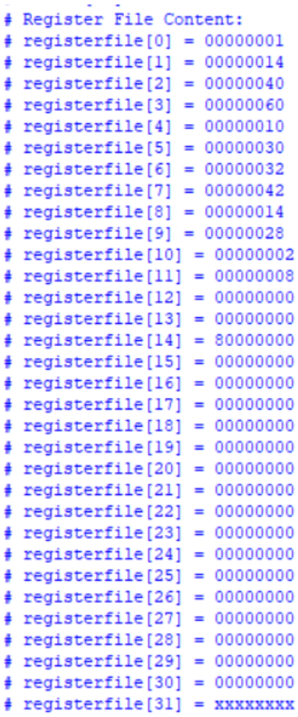
Let's consider the instruction with the hex code 0x91400004.

$10 (rs) : 0000 0000 0000 0000 0000 0000 0002 → positive so instruction will not jump to new address
LABEL : 0000 0000 0000 0000 0000 0000 0100

PC+4   :   0000 0000 0000 0000 0000 0000 1000 → 0x00000008 (nothing change)



```
# Instruction Memory[0]= 00  Data Memory[0]= 00   Register[0]= 00000000
# Instruction Memory[1]= 00  Data Memory[1]= 00   Register[1]= 00000014
# Instruction Memory[2]= 00  Data Memory[2]= 00   Register[2]= 00000040
# Instruction Memory[3]= 00  Data Memory[3]= 01   Register[3]= 00000060
# Instruction Memory[4]= 91  Data Memory[4]= 00   Register[4]= 00000010
# Instruction Memory[5]= 40  Data Memory[5]= 00   Register[5]= 00000030
# Instruction Memory[6]= 00  Data Memory[6]= 00   Register[6]= 00000032
# Instruction Memory[7]= 04  Data Memory[7]= 05   Register[7]= 00000042
# Instruction Memory[8]= xx  Data Memory[8]= 00   Register[8]= 00000014
# Instruction Memory[9]= xx  Data Memory[9]= 00   Register[9]= 00000028
# Instruction Memory[10]= xx  Data Memory[10]= 00   Register[10]= 00000002
# Instruction Memory[11]= xx  Data Memory[11]= 10   Register[11]= 00000008
# Instruction Memory[12]= xx  Data Memory[12]= 00   Register[12]= 00000000
# Instruction Memory[13]= xx  Data Memory[13]= 00   Register[13]= 00000000
# Instruction Memory[14]= xx  Data Memory[14]= 01   Register[14]= 80000000
# Instruction Memory[15]= xx  Data Memory[15]= 00   Register[15]= 00000000
# Instruction Memory[16]= xx  Data Memory[16]= 00   Register[16]= 00000000
# Instruction Memory[17]= xx  Data Memory[17]= 00   Register[17]= 00000000
# Instruction Memory[18]= xx  Data Memory[18]= 01   Register[18]= 00000000
# Instruction Memory[19]= xx  Data Memory[19]= 10   Register[19]= 00000000
# Instruction Memory[20]= xx  Data Memory[20]= 00   Register[20]= 00000000
# Instruction Memory[21]= xx  Data Memory[21]= 00   Register[21]= 00000000
# Instruction Memory[22]= xx  Data Memory[22]= 00   Register[22]= 00000000
# Instruction Memory[23]= xx  Data Memory[23]= 25   Register[23]= 00000000
# Instruction Memory[24]= xx  Data Memory[24]= 00   Register[24]= 00000000
# Instruction Memory[25]= xx  Data Memory[25]= 00   Register[25]= 00000000
# Instruction Memory[26]= xx  Data Memory[26]= 01   Register[26]= 00000000
# Instruction Memory[27]= xx  Data Memory[27]= 24   Register[27]= 00000000
# Instruction Memory[28]= xx  Data Memory[28]= 00   Register[28]= 00000000
# Instruction Memory[29]= xx  Data Memory[29]= 00   Register[29]= 00000000
# Instruction Memory[30]= xx  Data Memory[30]= 00   Register[30]= 00000000
#                    0PC 00000004  SUM 00000001   INST 91400004   REGISTER 00000010 00000030 00000032 00000014
#                   40PC 00000008  SUM 00000001   INST xxxxxxxx   REGISTER 00000010 00000030 00000032 00000014
#                   80PC xxxxxxxx  SUM 00000001   INST xxxxxxxx   REGISTER 00000010 00000030 00000032 00000014
```



```
# Register File Content:
# registerfile[0] = 00000001
# registerfile[1] = 00000014
# registerfile[2] = 00000040
# registerfile[3] = 00000060
# registerfile[4] = 00000010
# registerfile[5] = 00000030
# registerfile[6] = 00000032
# registerfile[7] = 00000042
# registerfile[8] = 00000014
# registerfile[9] = 00000028
# registerfile[10] = 00000002
# registerfile[11] = 00000008
# registerfile[12] = 00000000
# registerfile[13] = 00000000
# registerfile[14] = 80000000
# registerfile[15] = 00000000
# registerfile[16] = 00000000
# registerfile[17] = 00000000
# registerfile[18] = 00000000
# registerfile[19] = 00000000
# registerfile[20] = 00000000
# registerfile[21] = 00000000
# registerfile[22] = 00000000
# registerfile[23] = 00000000
# registerfile[24] = 00000000
# registerfile[25] = 00000000
# registerfile[26] = 00000000
# registerfile[27] = 00000000
# registerfile[28] = 00000000
# registerfile[29] = 00000000
# registerfile[30] = 00000000
# registerfile[31] = xxxxxxxx
```

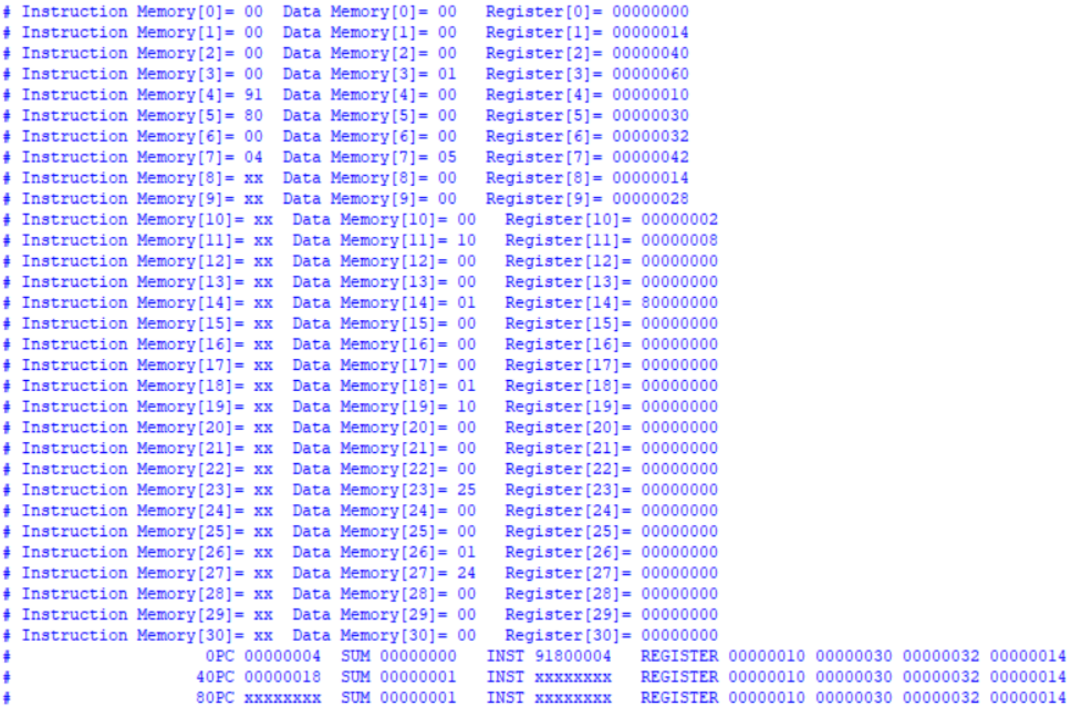
Case 3

Let's consider the instruction with the hex code 0x91800004.

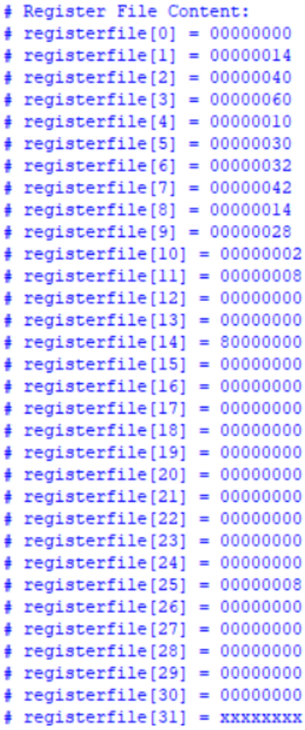
$12 (rs) : 0000 0000 0000 0000 0000 0000 0000 → zero so instruction will jump to new address
LABEL : 0000 0000 0000 0000 0000 0000 0100

PC+4 : 0000 0000 0000 0000 0000 0000 1000 → 0x00000008 stored in $25 register
LABEL*4: 0000 0000 0000 0000 0000 0001 0000

New PC : 0000 0000 0000 0000 0000 0001 1000 → 0x00000018



```
# Instruction Memory[0]= 00  Data Memory[0]= 00   Register[0]= 00000000
# Instruction Memory[1]= 00  Data Memory[1]= 00   Register[1]= 00000014
# Instruction Memory[2]= 00  Data Memory[2]= 00   Register[2]= 00000040
# Instruction Memory[3]= 00  Data Memory[3]= 01   Register[3]= 00000060
# Instruction Memory[4]= 91  Data Memory[4]= 00   Register[4]= 00000010
# Instruction Memory[5]= 80  Data Memory[5]= 00   Register[5]= 00000030
# Instruction Memory[6]= 00  Data Memory[6]= 00   Register[6]= 00000032
# Instruction Memory[7]= 04  Data Memory[7]= 05   Register[7]= 00000042
# Instruction Memory[8]= xx  Data Memory[8]= 00   Register[8]= 00000014
# Instruction Memory[9]= xx  Data Memory[9]= 00   Register[9]= 00000028
# Instruction Memory[10]= xx  Data Memory[10]= 00   Register[10]= 00000002
# Instruction Memory[11]= xx  Data Memory[11]= 10   Register[11]= 00000008
# Instruction Memory[12]= xx  Data Memory[12]= 00   Register[12]= 00000000
# Instruction Memory[13]= xx  Data Memory[13]= 00   Register[13]= 00000000
# Instruction Memory[14]= xx  Data Memory[14]= 01   Register[14]= 80000000
# Instruction Memory[15]= xx  Data Memory[15]= 00   Register[15]= 00000000
# Instruction Memory[16]= xx  Data Memory[16]= 00   Register[16]= 00000000
# Instruction Memory[17]= xx  Data Memory[17]= 00   Register[17]= 00000000
# Instruction Memory[18]= xx  Data Memory[18]= 01   Register[18]= 00000000
# Instruction Memory[19]= xx  Data Memory[19]= 10   Register[19]= 00000000
# Instruction Memory[20]= xx  Data Memory[20]= 00   Register[20]= 00000000
# Instruction Memory[21]= xx  Data Memory[21]= 00   Register[21]= 00000000
# Instruction Memory[22]= xx  Data Memory[22]= 00   Register[22]= 00000000
# Instruction Memory[23]= xx  Data Memory[23]= 25   Register[23]= 00000000
# Instruction Memory[24]= xx  Data Memory[24]= 00   Register[24]= 00000000
# Instruction Memory[25]= xx  Data Memory[25]= 00   Register[25]= 00000000
# Instruction Memory[26]= xx  Data Memory[26]= 01   Register[26]= 00000000
# Instruction Memory[27]= xx  Data Memory[27]= 24   Register[27]= 00000000
# Instruction Memory[28]= xx  Data Memory[28]= 00   Register[28]= 00000000
# Instruction Memory[29]= xx  Data Memory[29]= 00   Register[29]= 00000000
# Instruction Memory[30]= xx  Data Memory[30]= 00   Register[30]= 00000000
#                    0PC 00000004  SUM 00000000   INST 91800004   REGISTER 00000010 00000030 00000032 00000014
#                   40PC 00000018  SUM 00000001   INST xxxxxxxx   REGISTER 00000010 00000030 00000032 00000014
#                   80PC xxxxxxxx  SUM 00000001   INST xxxxxxxx   REGISTER 00000010 00000030 00000032 00000014
```



```
# Register File Content:
# registerfile[0] = 00000000
# registerfile[1] = 00000014
# registerfile[2] = 00000040
# registerfile[3] = 00000060
# registerfile[4] = 00000010
# registerfile[5] = 00000030
# registerfile[6] = 00000032
# registerfile[7] = 00000042
# registerfile[8] = 00000014
# registerfile[9] = 00000028
# registerfile[10] = 00000002
# registerfile[11] = 00000008
# registerfile[12] = 00000000
# registerfile[13] = 00000000
# registerfile[14] = 80000000
# registerfile[15] = 00000000
# registerfile[16] = 00000000
# registerfile[17] = 00000000
# registerfile[18] = 00000000
# registerfile[19] = 00000000
# registerfile[20] = 00000000
# registerfile[21] = 00000000
# registerfile[22] = 00000000
# registerfile[23] = 00000000
# registerfile[24] = 00000000
# registerfile[25] = 00000008
# registerfile[26] = 00000000
# registerfile[27] = 00000000
# registerfile[28] = 00000000
# registerfile[29] = 00000000
# registerfile[30] = 00000000
# registerfile[31] = xxxxxxxx
```

BALN

First Instruction: 01c07020 → add $14, $14, $0
Second Instruction: 6c000005 → baln 0x000005

The "baln" instruction functions similarly to the "jal" instruction but with a condition. If the status flag [N] is set to 1, indicating a negative result from a previous operation, the program branches to a pseudo-direct address, similar to how "jal" operates. Additionally, the address of the instruction following the branch is stored in register 31.

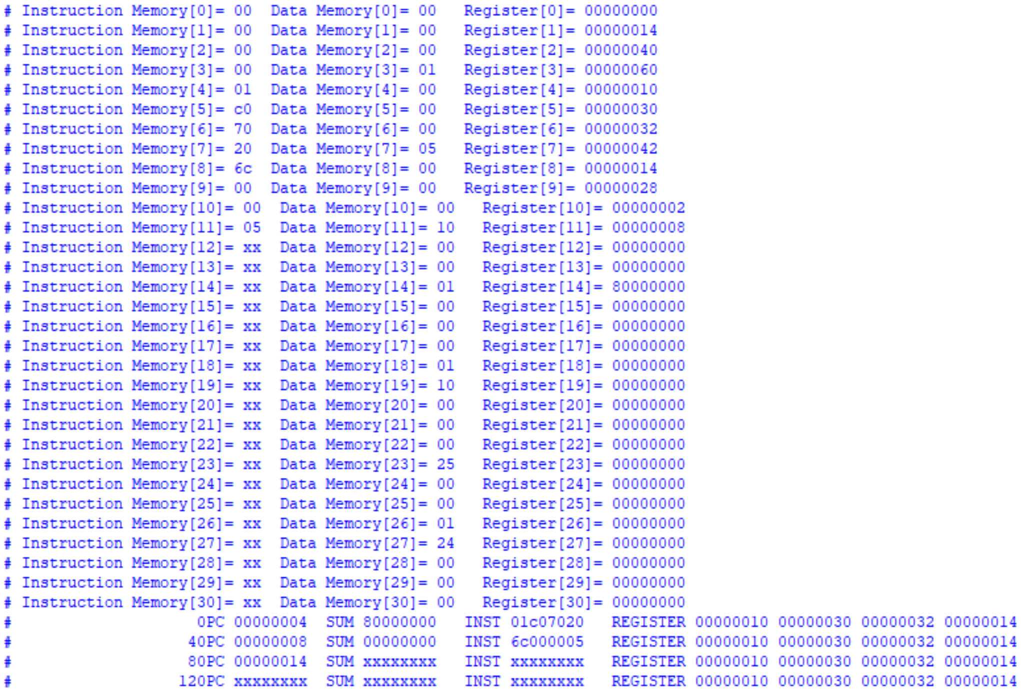
The first instruction is add $14, $14, $0
Register $14 contains a value of 0x80000000, which is a negative value. Since $0 is a zero register with a value of 0, the result will indeed be negative.

The program then branches to a pseudo-direct address calculated by concatenating the PC incremented by 4 and the extended label, resulting in a new PC value:

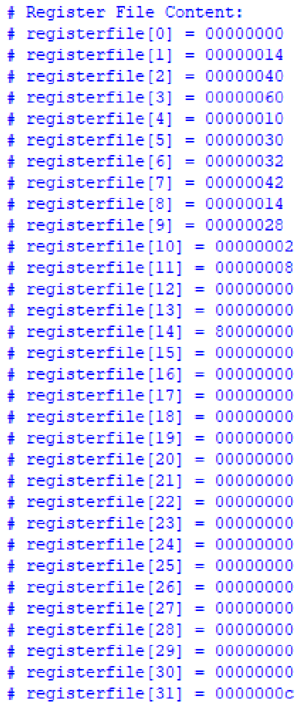
PC+4 : 0000 0000 0000 0000 0000 0000 1000 → first 4 bit ⇒ 0000
(0x00000008 stored in $31 register)

Label : 0000 0000 0000 0000 0101
Extended label: 0000 0000 0000 0000 0101 00

New PC : 0000 0000 0000 0000 0001 0100 → 000014



```
# Instruction Memory[0]= 00  Data Memory[0]= 00   Register[0]= 00000000
# Instruction Memory[1]= 00  Data Memory[1]= 00   Register[1]= 00000014
# Instruction Memory[2]= 00  Data Memory[2]= 00   Register[2]= 00000040
# Instruction Memory[3]= 00  Data Memory[3]= 01   Register[3]= 00000060
# Instruction Memory[4]= 01  Data Memory[4]= 00   Register[4]= 00000010
# Instruction Memory[5]= c0  Data Memory[5]= 00   Register[5]= 00000030
# Instruction Memory[6]= 70  Data Memory[6]= 00   Register[6]= 00000032
# Instruction Memory[7]= 20  Data Memory[7]= 05   Register[7]= 00000042
# Instruction Memory[8]= 6c  Data Memory[8]= 00   Register[8]= 00000014
# Instruction Memory[9]= 00  Data Memory[9]= 00   Register[9]= 00000028
# Instruction Memory[10]= 00  Data Memory[10]= 00   Register[10]= 00000002
# Instruction Memory[11]= 05  Data Memory[11]= 10   Register[11]= 00000008
# Instruction Memory[12]= xx  Data Memory[12]= 00   Register[12]= 00000000
# Instruction Memory[13]= xx  Data Memory[13]= 00   Register[13]= 00000000
# Instruction Memory[14]= xx  Data Memory[14]= 01   Register[14]= 80000000
# Instruction Memory[15]= xx  Data Memory[15]= 00   Register[15]= 00000000
# Instruction Memory[16]= xx  Data Memory[16]= 00   Register[16]= 00000000
# Instruction Memory[17]= xx  Data Memory[17]= 00   Register[17]= 00000000
# Instruction Memory[18]= xx  Data Memory[18]= 01   Register[18]= 00000000
# Instruction Memory[19]= xx  Data Memory[19]= 10   Register[19]= 00000000
# Instruction Memory[20]= xx  Data Memory[20]= 00   Register[20]= 00000000
# Instruction Memory[21]= xx  Data Memory[21]= 00   Register[21]= 00000000
# Instruction Memory[22]= xx  Data Memory[22]= 00   Register[22]= 00000000
# Instruction Memory[23]= xx  Data Memory[23]= 25   Register[23]= 00000000
# Instruction Memory[24]= xx  Data Memory[24]= 00   Register[24]= 00000000
# Instruction Memory[25]= xx  Data Memory[25]= 00   Register[25]= 00000000
# Instruction Memory[26]= xx  Data Memory[26]= 01   Register[26]= 00000000
# Instruction Memory[27]= xx  Data Memory[27]= 24   Register[27]= 00000000
# Instruction Memory[28]= xx  Data Memory[28]= 00   Register[28]= 00000000
# Instruction Memory[29]= xx  Data Memory[29]= 00   Register[29]= 00000000
# Instruction Memory[30]= xx  Data Memory[30]= 00   Register[30]= 00000000
#                    0PC 00000004  SUM 80000000   INST 01c07020   REGISTER 00000010 00000030 00000032 00000014
#                   40PC 00000008  SUM 00000000   INST 6c000005   REGISTER 00000010 00000030 00000032 00000014
#                   80PC 00000014  SUM xxxxxxxx   INST xxxxxxxx   REGISTER 00000010 00000030 00000032 00000014
#                  120PC xxxxxxxx  SUM xxxxxxxx   INST xxxxxxxx   REGISTER 00000010 00000030 00000032 00000014
```



```
# Register File Content:
# registerfile[0] = 00000000
# registerfile[1] = 00000014
# registerfile[2] = 00000040
# registerfile[3] = 00000060
# registerfile[4] = 00000010
# registerfile[5] = 00000030
# registerfile[6] = 00000032
# registerfile[7] = 00000042
# registerfile[8] = 00000014
# registerfile[9] = 00000028
# registerfile[10] = 00000002
# registerfile[11] = 00000008
# registerfile[12] = 00000000
# registerfile[13] = 00000000
# registerfile[14] = 80000000
# registerfile[15] = 00000000
# registerfile[16] = 00000000
# registerfile[17] = 00000000
# registerfile[18] = 00000000
# registerfile[19] = 00000000
# registerfile[20] = 00000000
# registerfile[21] = 00000000
# registerfile[22] = 00000000
# registerfile[23] = 00000000
# registerfile[24] = 00000000
# registerfile[25] = 00000000
# registerfile[26] = 00000000
# registerfile[27] = 00000000
# registerfile[28] = 00000000
# registerfile[29] = 00000000
# registerfile[30] = 00000000
# registerfile[31] = 0000000c
```

TEST CASES

BRV	JMXOR	NORI	BLEZAL	JALPC	BALN
@00 00 00 00 00 01 09 40 20 01 00 00 14 01 ce 70 20 01 00 00 14	@00 00 00 00 00 01 4b 00 22	@00 00 00 00 00 3d ae 89 4a	@00 00 00 00 00 91 C0 00 04 *91* *40* *00* *04* *91* *80* *00* *04*	@00 00 00 00 00 7c 0f 00 02	@00 00 00 00 00 01 c0 70 20 6c 00 00 05