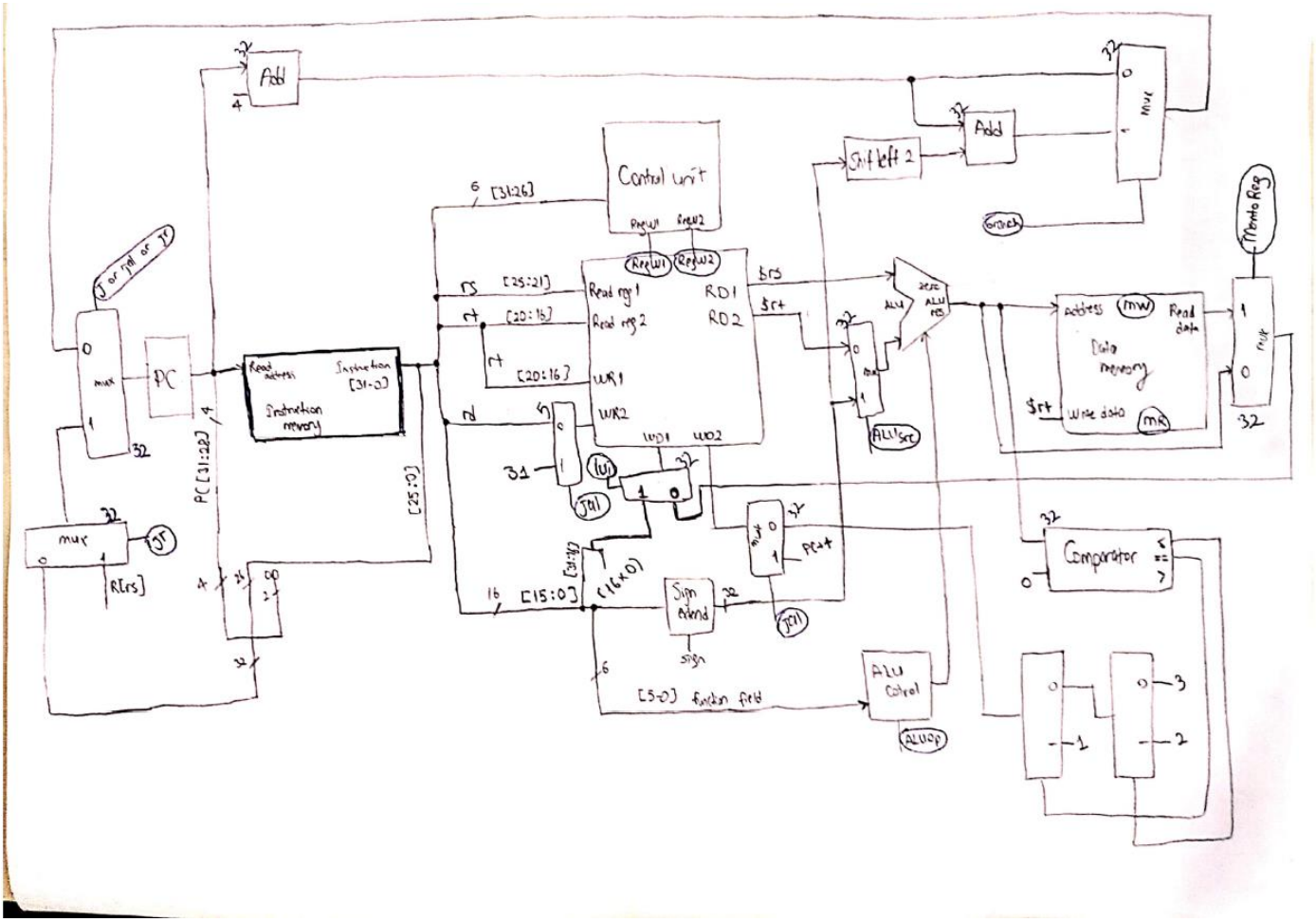
CSE 331 – Computer Organization

Homework #4 Report

Harun Albayrak - 171044014

My Datapath:



Supported Instructions : (14 pieces)

lw,sw,j,jal,jr,beq,bne,addn,subn,xorn,andn,orn,ori,lui

----

lw : 10001100000000000000000000000000
sw : 10101100000000000000000000000000
j : 00001000000000000000000000000000
jal : 00001100000000000000000000000000
jr : 00000100000000000000000000001000
beq : 00010000000000000000000000000000
bne : 00010100000000000000000000000000
ori : 00101100000000000000000000000000
lui : 00111100000000000000000000000000
addn : 00000000000000000000000000000001
subn : 00000000000000000000000000000010
xorn : 00000000000000000000000000000011
andn : 00000000000000000000000000000100
orn : 00000000000000000000000000000101

RTL → Control

Instr.	RegW1	RegW2	j	jal	jr	lui	MemtoReg	MR	MW	Bran.	ALUSrc	ALUop1	ALUOp0
Lw	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**
Sw	**0**	**0**	**0**	**0**	**0**	**0**	**x**	**0**	**1**	**0**	**1**	**0**	**0**
Bran.	**0**	**0**	**0**	**0**	**0**	**0**	**x**	**0**	**0**	**1**	**0**	**x**	**x**
j	**0**	**0**	**1**	**0**	**0**	**0**	**x**	**0**	**0**	**0**	**0**	**x**	**x**
jal	**0**	**1**	**0**	**1**	**0**	**0**	**x**	**0**	**0**	**0**	**0**	**x**	**x**
jr	**0**	**0**	**0**	**0**	**1**	**0**	**x**	**0**	**0**	**0**	**0**	**x**	**x**
ori	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**
lui	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**x**	**x**	**x**
New types(R)	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**

ALU Control Bits

Instruction opcode	ALUOp	Instruction operation	Function field	Desired ALU Action	ALU Control
LW	00	Load word	XXXXXX	Add	010
SW	00	Store word	XXXXXX	Add	010
J,JAL,JR	XX	Jump	XXXXXX	Unnecessary	XXX
BEQ	XX	Branch equal	XXXXXX	Unnecessary	XXX
BNE	XX	Branch not eq.	XXXXXX	Unnecessary	XXX
ORI	01	OR immediate	XXXXXX	Or	001
LUI	XX	Load Upper im.	XXXXXX	Unnecessary	XXX
ADDN	10	Add new	000001	Add	010
SUBN	10	Sub new	000010	Sub	110
XORN	10	Xor new	000011	Xor	111
ANDN	10	And new	000100	And	000
ORN	10	Or new	000101	Or	001

The truth table for the 3 ALU Control bits

ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	OP2	OP1	OP0
0	**0**	**X**	**X**	**X**	**X**	**X**	**X**	**0**	**1**	**0**
0	**1**	**X**	**X**	**X**	**X**	**X**	**X**	**0**	**0**	**1**
1	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**0**
1	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**
1	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**
1	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**
1	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**1**

OP2 = ALUop1*(ALUop0)'*F1

OP1 = (ALUop1)'*(ALUop0)' + (ALUOp1*F2')

OP0 = (ALUop1)'*ALUop0 + ALUop1*(ALUop0)'*(F1*F0 + F2*F0)

The truth table for the Control Unit

Opcode:	100011	101011	000010	000011	000001	000100,000101	001101	001111	000000
	lw	sw	j	jal	jr(R type)	beq,bne	ori	lui	New types(R)
RegW1	1	0	0	0	0	0	1	1	1
RegW2	0	0	0	1	0	0	0	0	1
j	0	0	1	0	0	0	0	0	0
jal	0	0	0	1	0	0	0	0	0
jr	0	0	0	0	1	0	0	0	0
lui	0	x	x	x	x	x	0	1	0
MemtoReg	1	x	x	x	x	x	0	0	0
MR	1	0	0	0	0	0	0	0	0
MW	0	1	0	0	0	0	0	0	0
Branch	0	0	0	0	0	1	0	0	0
ALUSrc	1	1	0	0	0	0	1	0	0
ALUOp(Symbolic)	Add	Add	X	X	X	X	Or	X	R type
ALUOp1	0	0	x	x	x	x	0	x	1
ALUOp0	0	0	x	x	x	x	1	x	0
signExtend	1	1	0	0	0	0	0	0	0

RegW1 = lw + ori + lui + newTypes

RegW2 = jal + newTypes

j = j

jal = jal

jr = jr

lui = lui

MemtoReg = lw

MR = lw

MW = sw

Branch = beq + bne

ALUSrc = lw + sw + ori

ALUOp1 = newTypes

ALUOp0 = ori

signExtend = lw + sw

-

Modules

1) mux5bit : This module select one of two 5 bit number.

```
VSIM 44> run
# Time: 0 _input1:11111, _input2:00000, selectBit:0, _output:11111
VSIM 45> run
# Time:100 _input1:11111, _input2:00000, selectBit:1, _output:00000
```

2) mux32bit : This module select one of two 32 bit number.

```
VSIM 50> run
# Time: 0 _input1:11111111111111111111111111111111, _input2:00000000000000000000000000000000, selectBit:0, _output:11111111111111111111111111111111
VSIM 51> run
# Time:100 _input1:11111111111111111111111111111111, _input2:00000000000000000000000000000000, selectBit:1, _output:00000000000000000000000000000000
```

3) mux32bit_5x1 : This module select one of two 32 bit number. (This is not normal mux, this is special purpose mux). I use the mux for selecting and,or,add,sub,xor in Arithmetic Logic Unit.

```
VSIM 93> run
# Time: 0, selectBit:000,_output:00000000000000000000000000000000
run
# Time:100, selectBit:001,_output:00000000000000000000000000000001
run
# Time:200, selectBit:010,_output:00000000000000000000000000000010
run
# Time:300, selectBit:110,_output:00000000000000000000000000000011
run
# Time:400, selectBit:111,_output:00000000000000000000000000000100
```

4) comparator32bit_with0 : This module compares any 32 bit number with 0. Its outputs are "equal,lessThan and greatherThan".

```
VSIM 63> run
# Time: 0 _input1:11111111111111111111111111111111, _equal: 0, _greaterThan: 0, _lessThan: 1
run
# Time:100 _input1:00000000000000000000000000000000, _equal: 1, _greaterThan: 0, _lessThan: 0
```

5) half_adder : This module is for adder32bit module.

6) full_adder : This module is for adder32bit module.

5) half_subtractor : This module is for subtractor32bit module.

6) full_ subtractor : This module is for subtractor32bit module.

7) adder32bit : This module adds two 32-bit numbers.

```
VSIM 69> run
# Time: 0 _input1:00000000000000001111111111111111, _input2:00000000000000001111111111111111, _output:00000000000000011111111111111110
VSIM 70> run
# Time:100 _input1:00000000000000000000000000000111, _input2:00000000000000000000000000000111, _output:00000000000000000000000000001110
```

8) subtractor32bit : This module subtracts a 32-bit number from a 32-bit number.

```
VSIM 75> run
# Time: 0 _input1:00000000000000001111111111111111, _input2:00000000000000001111111111111111, _output:00000000000000000000000000000000
VSIM 76> run
# Time:100 _input1:00000000000000000000000000000111, _input2:00000000000000000000000000000011, _output:00000000000000000000000000000100
```

9) and32bit : This module ands two 32-bit numbers.

10) or32bit : This module ors two 32-bit numbers.

11) xor32bit : This module xors two 32-bit numbers.

12) shiftleft2 : This module shifts left any 32 bit numbers.

```
VSIM 79> run
# Time: 0 _input:11111111111111111111111111111111, _output: 11111111111111111111111111111100
```

13) extender : This module extends 16 bit number to 32 bit number by one or zero according to signExtend bit. (sign extend and zero extend)

```
VSIM 84> run
# Time: 0 _input:1111111111111111, _output:11111111111111111111111111111111, signExtend: 1
VSIM 85> run
# Time:100 _input:1111111111111111, _output:00000000000000001111111111111111, signExtend: 0
```

14) alu_control : This module generate the aluControl signal(3 bit) for the ALU. aluControl signal is generated according to function field and aluOp.

```
VSIM 99> run
# Time: 0 _func:00000, _aluop:00, _alucontrol:010
run
# Time:100 _func:00000, _aluop:01, _alucontrol:001
run
# Time:200 _func:00001, _aluop:10, _alucontrol:010
run
# Time:300 _func:00010, _aluop:10, _alucontrol:110
run
# Time:400 _func:00011, _aluop:10, _alucontrol:111
run
# Time:500 _func:00100, _aluop:10, _alucontrol:000
run
# Time:600 _func:00101, _aluop:10, _alucontrol:001
```

15) control_unit : This module generate the signals for muxes' select bits and other things. The signals are generated according to operation code of the instruction.

```
VSIM 116> run
# Time: 0 Opcode:100011, RegWl:1, RegW2:0, j:0, jal:0, jr:0, lui:0, MemtoReg:1, MR:1, MW:0, Branch:0, ALUSrc:1, ALUOp:00
run
# Time:100 Opcode:101011, RegWl:0, RegW2:0, j:0, jal:0, jr:0, lui:0, MemtoReg:0, MR:0, MW:1, Branch:0, ALUSrc:1, ALUOp:00
run
# Time:200 Opcode:000010, RegWl:0, RegW2:0, j:1, jal:0, jr:0, lui:0, MemtoReg:0, MR:0, MW:0, Branch:0, ALUSrc:0, ALUOp:00
run
# Time:300 Opcode:000011, RegWl:0, RegW2:1, j:0, jal:1, jr:0, lui:0, MemtoReg:0, MR:0, MW:0, Branch:0, ALUSrc:0, ALUOp:00
run
# Time:400 Opcode:000001, RegWl:0, RegW2:0, j:0, jal:0, jr:1, lui:0, MemtoReg:0, MR:0, MW:0, Branch:0, ALUSrc:0, ALUOp:00
run
# Time:500 Opcode:000100, RegWl:0, RegW2:0, j:0, jal:0, jr:0, lui:0, MemtoReg:0, MR:0, MW:0, Branch:1, ALUSrc:0, ALUOp:00
run
# Time:600 Opcode:000101, RegWl:0, RegW2:0, j:0, jal:0, jr:0, lui:0, MemtoReg:0, MR:0, MW:0, Branch:1, ALUSrc:0, ALUOp:00
run
# Time:700 Opcode:001101, RegWl:1, RegW2:0, j:0, jal:0, jr:0, lui:0, MemtoReg:0, MR:0, MW:0, Branch:0, ALUSrc:1, ALUOp:01
run
# Time:800 Opcode:001111, RegWl:1, RegW2:0, j:0, jal:0, jr:0, lui:1, MemtoReg:0, MR:0, MW:0, Branch:0, ALUSrc:0, ALUOp:00
run
# Time:900 Opcode:000000, RegWl:1, RegW2:1, j:0, jal:0, jr:0, lui:0, MemtoReg:0, MR:0, MW:0, Branch:0, ALUSrc:0, ALUOp:10
```

16) alu : This module is Arithmetic Logic Unit. The ALU apply the one operate of ‘AND,OR,ADD,SUB,XOR’ .

AND : 000, **OR :** 001, **ADD :** 010, **SUB :** 110, **XOR :** 111

```
VSIM 120> run
# Time: 0 _input1:00000000000000000000000000000111, _input2:00000000000000000000000000000111, _output:00000000000000000000000000000111, alu_control: 000
run
# Time:100 _input1:00000000000000000000000000000111, _input2:00000000000000000000000000000111, _output:00000000000000000000000000000111, alu_control: 001
run
# Time:200 _input1:00000000000000000000000000000111, _input2:00000000000000000000000000000111, _output:00000000000000000000000000001110, alu_control: 010
run
# Time:300 _input1:00000000000000000000000000000111, _input2:00000000000000000000000000000111, _output:00000000000000000000000000000000, alu_control: 110
run
# Time:400 _input1:00000000000000000000000000000111, _input2:00000000000000000000000000000111, _output:00000000000000000000000000000000, alu_control: 111
```

17) register

18) instruction_memory

19) data_memory

20) mips32_processor_171044014 : The main module

TESTBENCH

My testbench file: mips32_processor_171044014_test.v

Note : I use PC+1 instead of PC+4 because i’ve designed the processor. Therefore, I had to use dummy instructions between some jump instructions.

```
10001100001000100000000000000000
10101100011000100000000000000000
00001000000000000000000000000001
00000000000000000000000000000000
00001100000000000000000000000010
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
10001100100111110000000000000000
00000111111000000000000000001000
00010000101001100000000000000000
00010100110001110000000000000000
00110101000010010000000000000000
00111101010000000000000000000000
00000001100011010101100000000001
00000001111100000111000000000010
00000010010100111000100000000011
00000010101101101010000000000100
00000011000110011011100000000101
```

respectively,

lw $2, 0($1) -- instruction : 10001100001000100000000000000000

sw $2, 0($3) -- instruction : 10101100011000100000000000000000

j Label -- instruction : 00001000000000000000000000000001

Dummy instruction -- instruction : 00000000000000000000000000000000

jal Label -- instruction : 00001100000000000000000000000010

Dummy instruction -- instruction : 00000000000000000000000000000000

Dummy instruction -- instruction : 00000000000000000000000000000000

Dummy instruction -- instruction : 00000000000000000000000000000000

lw $31, 0($4) -- instruction : 10001100100111110000000000000000

jr $ra -- instruction : 00000111111000000000000000001000

beq $5,$6,Label -- instruction : 00010000101001100000000000000000

bne $6,$7,Label -- instruction : 00010100110001110000000000000000

ori $8,$9,3 -- instruction : 00110101000010010000000000000000

lui $10, 16'b1 -- instruction : 00111100000010101111111111111111

addn $11,$12,$13 -- instruction : 00000001100011010101100000000001

subn $14,$15,$16 -- instruction : 00000001111100000111000000000010

xorn $17,$18,$19 -- instruction : 00000010010100111000100000000011

andn $20,$21,$22 -- instruction : 00000010101101101010000000000100

orn $23,$24,$25 -- instruction : 00000011000110011011100000000101

lui $26, 0101010101010101 -- instruction : 00111100001101000101010101010101

Register:

```
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000001000000000000000
00000000000000000100000000000000
00000000000000000000000000000001
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000001111100
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000101
00000000000000000000000000000101
00000000000000000000000000000000
00000000000000000000000000010100
00000000000000000000000000000101
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000011111
00000000000000000000000000000111
00000000000000000000000000000000
00000000000000000000000000010101
00000000000000000000000000001010
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000001010
```

Memory:

```
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
```

$1 content = 00000000000000000000000000000000 (before)

$2 content = 00000000000000000000000000000000 (before)

Memory[0] = 11111111111111111111111111111111 (before)

lw $2, 0($1)

$1 content = 00000000000000000000000000000000 (after)

$2 content = 11111111111111111111111111111111 (after)

Memory[0] = 00000000000000000000000000000000 (after)

$3 content = 00000000000000001000000000000000 (before)

$2 content = 11111111111111111111111111111111 (before)

Memory[2] = 00000000000000000000000000000000 (before)

sw $2, 0($3)

$3 content = 00000000000000000000000000000001 (after)

$2 content = 00000000000000000000000000000000 (after)

Memory[2] = 11111111111111111111111111111111 (after)

PC = 00000000000000000000000000000011 (before)

j Label

PC = 00000000000000000000000000000100 (after)

PC = 00000000000000000000000000000100 (before)

jal Label

PC = 00000000000000000000000000001000 (after)

$31 content = 00000000000000000000000000000100 (after)

$4 content = 00000000000000001000000000000000 (before)

$31 content = 00000000000000000000000000000101 (before)

Memory[1] = 00000000000000000000000000001010 (before)

lw $31, 0($4)

$4 content = 00000000000000000000000000000000 (after)

$31 content = 00000000000000000000000000001010 (after)

Memory[0] = 00000000000000000000000000000000 (after)

PC = 00000000000000000000000000001001 (before)

jr $ra

PC = 00000000000000000000000000001010 (after)

$5 content = 00000000000000000000000000000001 (before)

$6 content = 00000000000000000000000000000000 (before)

beq $5,$6,Label

$5 content = 00000000000000000000000000000001 (after)

$6 content = 00000000000000000000000000000000 (after)

$6 content = 00000000000000000000000000000000 (before)

$7 content = 00000000000000000000000000000000 (before)

bne $6,$7,Label

$6 content = 00000000000000000000000000000000 (after)

$7 content = 00000000000000000000000000000000 (after)

$8 content = 00000000000000000000000000000000 (before)

$9 content = 00000000000000000000000001111100 (before)

ori $8,$9,3

$8 content = 00000000000000000000000000000000 (after)

$9 content = 00000000000000000000000001111111 (after)

$8 content = 00000000000000000000000000000000 (before)

lui $10, 16'b1

$8 content = 11111111111111110000000000000000 (after)

$11 (rd content) = 00000000000000000000000000000000 (before)

$12 (rs content) = 00000000000000000000000000000101 (before)

$13 (rt content) = 00000000000000000000000000000101 (before)

addn $11,$12,$13

$11 (rd content) = 00000000000000000000000000000011 (after)

$12 (rs content) = 00000000000000000000000000001010 (after)

$13 (rt content) = 00000000000000000000000000000101 (after)

$14 (rd content) = 00000000000000000000000000000000 (before)

$15 (rs content) = 00000000000000000000000000010100 (before)

$16 (rt content) = 00000000000000000000000000000101 (before)

subn $14,$15,$16

$14 (rd content) = 00000000000000000000000000000011 (after)

$15 (rs content) = 00000000000000000000000000001111 (after)

$16 (rt content) = 00000000000000000000000000000101 (after)

$17 (rd content) = 00000000000000000000000000000000 (before)

$18 (rs content) = 00000000000000000000000000000000 (before)

$19 (rt content) = 00000000000000000000000000000000 (before)

xorn $17,$18,$19

$17 (rd content) = 00000000000000000000000000000001 (after)

$18 (rs content) = 00000000000000000000000000000000 (after)

$19 (rt content) = 00000000000000000000000000000000 (after)

$20 (rd content) = 00000000000000000000000000000000 (before)

$21 (rs content) = 00000000000000000000000000011111 (before)

$22 (rt content) = 00000000000000000000000000000111 (before)

andn $20,$21,$22

$20 (rd content) = 00000000000000000000000000000011 (after)

$21 (rs content) = 00000000000000000000000000000111 (after)

$22 (rt content) = 00000000000000000000000000000111 (after)

$23 (rd content) = 00000000000000000000000000000000 (before)

$24 (rs content) = 00000000000000000000000000010101 (before)

$25 (rt content) = 00000000000000000000000000001010 (before)

orn $23,$24,$25

$23 (rd content) = 00000000000000000000000000000011 (after)

$24 (rs content) = 00000000000000000000000000011111 (after)

$25 (rt content) = 00000000000000000000000000001010 (after)

Registers:

```
00000000000000000000000000000000
00000000000000000000000000000000
11111111111111111111111111111111
00000000000000001000000000000000
00000000000000000100000000000000
00000000000000000000000000000001
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
11111111111111110000000000000000
00000000000000000000000000000011
00000000000000000000000000001010
00000000000000000000000000000101
00000000000000000000000000000011
00000000000000000000000000001111
00000000000000000000000000000101
00000000000000000000000000000001
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000011
00000000000000000000000000000111
00000000000000000000000000000111
00000000000000000000000000000011
00000000000000000000000000011111
00000000000000000000000000001010
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000001010
```

Memory:

```
00000000000000000000000000000000
00000000000000000000000000001010
11111111111111111111111111111111
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
```