GTU Department of Computer Engineering CSE 331/503 - Fall 2020
Homework 4 Report

Akif KARTAL
171044098

Problem Definition

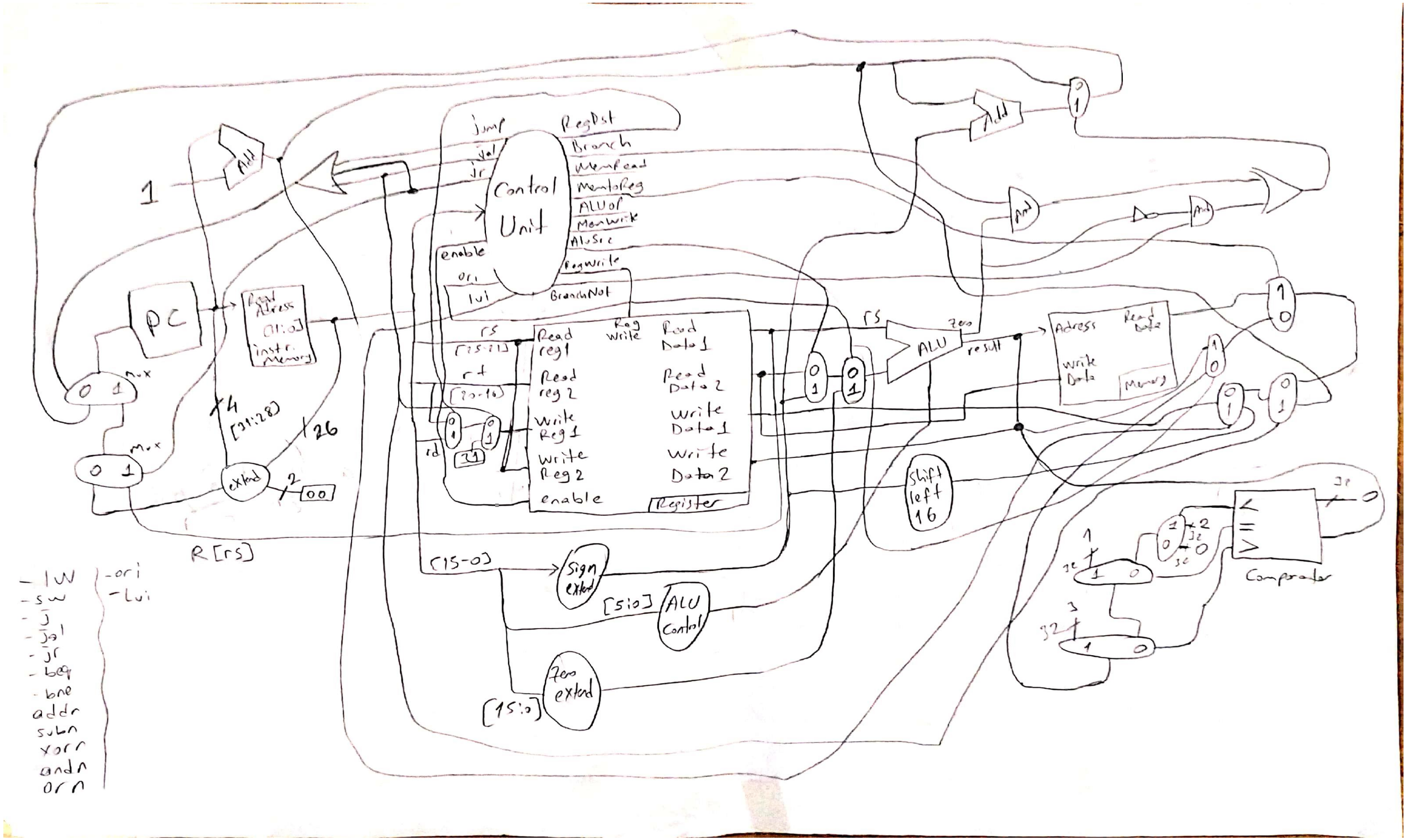
The problem is to implement a different version of 32-bit MIPS processor.

Solution Steps

1) Design new Datapath
2) Desing new Control unit
3) Desing important component of Datapath like register block, ALU block etc.
4) Combine all components.

I will be putting everything step by step in this report.

1) Draw Datapath



2)Design Control Unit

Main Control Unit Truth Table

Instr.	Reg Des	ALUSrc	Memto Reg	Reg Wr	Mem Rd	Mem Wr	Branch	ALUop1	ALUop0	jump	jal	jr	bne	enable	ori	lui
New R-type	1	0	0	1	0	0	0	1	0	0	0	0	0	1	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	1	0	0	0	0	0	0	0
bne	X	0	X	0	0	0	0	0	1	0	0	0	1	0	0	0
jump	X	X	X	0	0	0	0	X	X	1	0	0	0	0	0	0
jal	X	X	X	1	0	0	0	X	X	0	1	0	0	0	0	0
jr	X	X	X	0	0	0	0	X	X	0	0	1	0	0	0	0
ori	0	X	0	1	0	0	0	1	1	0	0	0	0	0	1	0
lui	0	X	X	1	0	0	0	X	X	0	0	0	0	0	0	1

*enable is second signal for register block.

Opcodes

Instr.	Opcode
New R-type (5)	000000
lw	100011
sw	101011
beq	000100
bne	000101
jump	000010
jal	000011
jr	001000
ori	001101
lui	001111

Main Control Unit Test Results

```
VSIM 5> step -current
# time= 0,opcode= 0,reg_dest=1,alu_src=0,mem_to_reg=0,reg_wrt=1,mem_read=0,mem_wrt=0,branch=0,alu_op=10,jump=0,jal=0,jr=0,bne=0,enable=1,ori=0,lui=0  r-type-n
# time=20,opcode=35,reg_dest=0,alu_src=1,mem_to_reg=1,reg_wrt=1,mem_read=1,mem_wrt=0,branch=0,alu_op=00,jump=0,jal=0,jr=0,bne=0,enable=0,ori=0,lui=0  lw
# time=40,opcode=43,reg_dest=0,alu_src=1,mem_to_reg=0,reg_wrt=0,mem_read=0,mem_wrt=1,branch=0,alu_op=00,jump=0,jal=0,jr=0,bne=0,enable=0,ori=0,lui=0  sw
# time=60,opcode= 4,reg_dest=0,alu_src=0,mem_to_reg=0,reg_wrt=0,mem_read=0,mem_wrt=0,branch=1,alu_op=01,jump=0,jal=0,jr=0,bne=0,enable=0,ori=0,lui=0  beq
# time=80,opcode= 5,reg_dest=0,alu_src=0,mem_to_reg=0,reg_wrt=0,mem_read=0,mem_wrt=0,branch=0,alu_op=01,jump=0,jal=0,jr=0,bne=1,enable=0,ori=0,lui=0  bne
# time=100,opcode= 2,reg_dest=0,alu_src=0,mem_to_reg=0,reg_wrt=0,mem_read=0,mem_wrt=0,branch=0,alu_op=00,jump=1,jal=0,jr=0,bne=0,enable=0,ori=0,lui=0  jump
# time=120,opcode= 3,reg_dest=0,alu_src=0,mem_to_reg=0,reg_wrt=1,mem_read=0,mem_wrt=0,branch=0,alu_op=00,jump=0,jal=1,jr=0,bne=0,enable=0,ori=0,lui=0  jal
# time=140,opcode= 8,reg_dest=0,alu_src=0,mem_to_reg=0,reg_wrt=0,mem_read=0,mem_wrt=0,branch=0,alu_op=00,jump=0,jal=0,jr=1,bne=0,enable=0,ori=0,lui=0  jr
# time=160,opcode=13,reg_dest=0,alu_src=0,mem_to_reg=0,reg_wrt=1,mem_read=0,mem_wrt=0,branch=0,alu_op=11,jump=0,jal=0,jr=0,bne=0,enable=0,ori=1,lui=0  ori
# time=180,opcode=15,reg_dest=0,alu_src=0,mem_to_reg=0,reg_wrt=1,mem_read=0,mem_wrt=0,branch=0,alu_op=00,jump=0,jal=0,jr=0,bne=0,enable=0,ori=0,lui=1  lui
```

Opcodes are in decimal format in test results.

ALU Control Truth Table

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**
beq	01	branch equal	XXXXXX	subtract	**11 0**
bne	01	branch not equal	XXXXXX	subtract	**11 0**
j	X	jump	XXXXXX	X	**XXX**
jal	X	jump and link	XXXXXX	X	**XXX**
jr	X	jump register	XXXXXX	X	**XXX**
ori	11	or immediate	XXXXXX	or	**001**
lui	X	load upper imm.	XXXXXX	X	**XXX**
R-type-n	10	addn	100000	add	**010**
R-type-n	10	subtractn	100010	subtract	**110**
R-type-n	10	andn	100100	and	**000**
R-type-n	10	orn	100101	or	**001**
R-type-n	10	xorn	101010	xor	**111**

Note: **Since xor instr. don't have function code, I used slt instr. function code(0x2A).**

Boolean expressions from table;

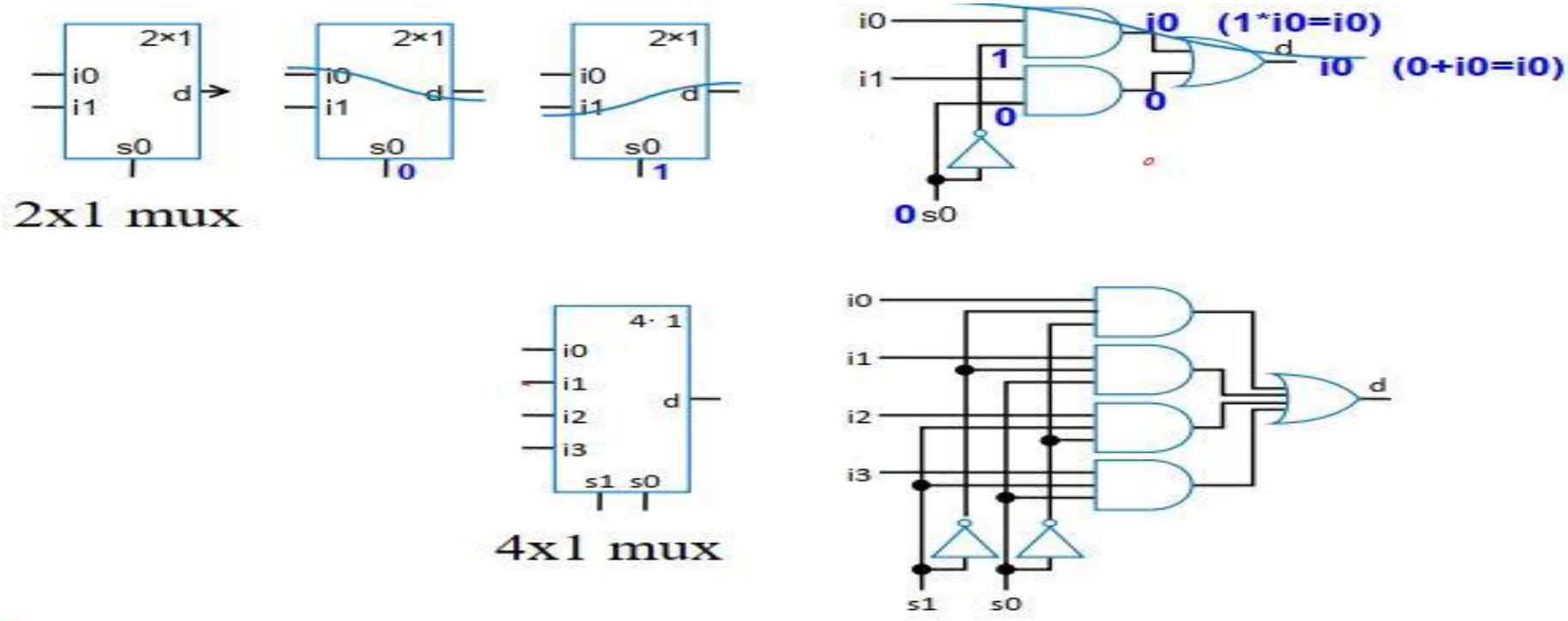
- ALUctr<2> = (ALUop0 +(ALUop1 & func<1>)) & ! (ALUop0 & ALUop1)
- ALUctr<1> = (!ALUop1 + !func<2>) & ! (ALUop0 & ALUop1)
- ALUctr<0> = (ALUop1 & (func<3> + func<0>)) + (ALUop0 & ALUop1)

ALU Control Test Results

```
# time= 0,alu_op=00,function=000000,alu_ctr=010  lw-sw
# time=20,alu_op=01,function=000000,alu_ctr=110  beq-bne
# time=40,alu_op=11,function=000000,alu_ctr=001  ori
# time=60,alu_op=10,function=100000,alu_ctr=010  addn
# time=80,alu_op=10,function=100010,alu_ctr=110  subn
# time=100,alu_op=10,function=100100,alu_ctr=000 andn
# time=120,alu_op=10,function=100101,alu_ctr=001 orn
# time=140,alu_op=10,function=101010,alu_ctr=111 xorn
```

MUX Design

- ✓ Mux design is taken from last year CSE 232 lecture slides(Book: Digital Design, Frank Vahid).



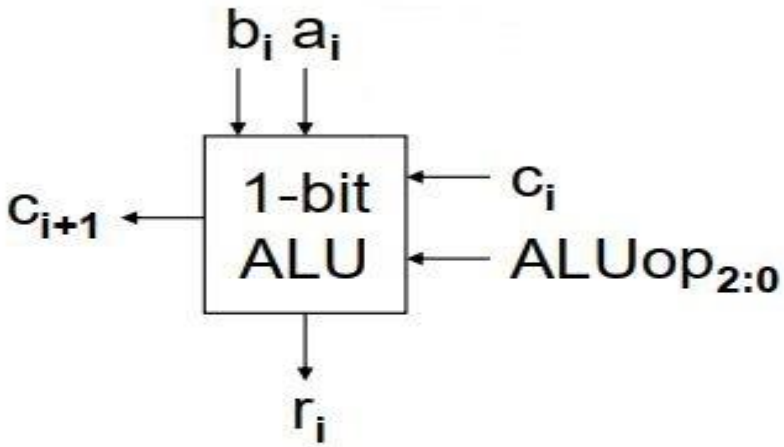
32-bit Mux Test Results

```
# time= 0,a= 0,b=63,s0=1,result=63
# time=20,a= 0,b=63,s0=0,result= 0
# time=40,a=16914,b=63,s0=0,result=16914
```

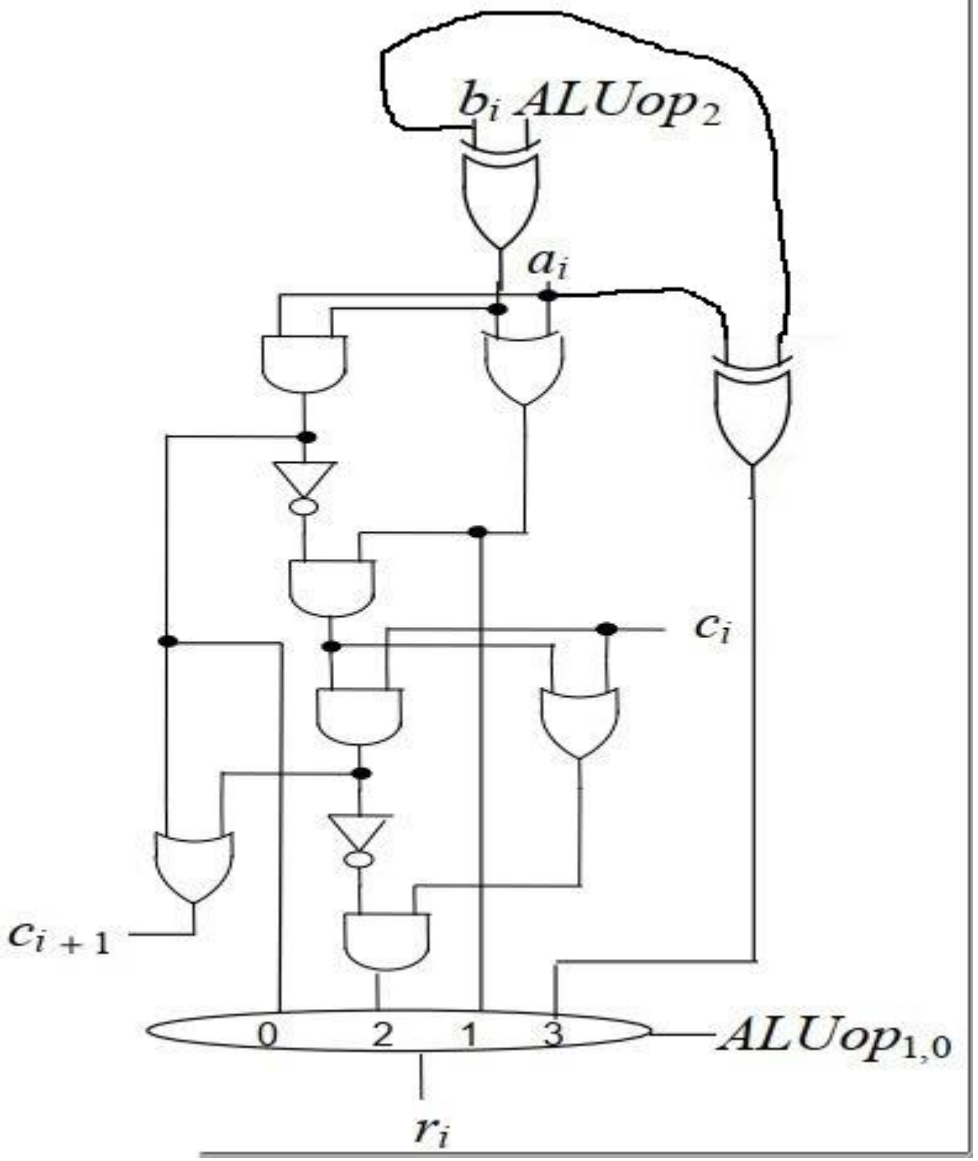
a, b and result numbers were printed in decimal format to gain space. In test they are in binary format.

ALU Design

1 Bit Updated ALU



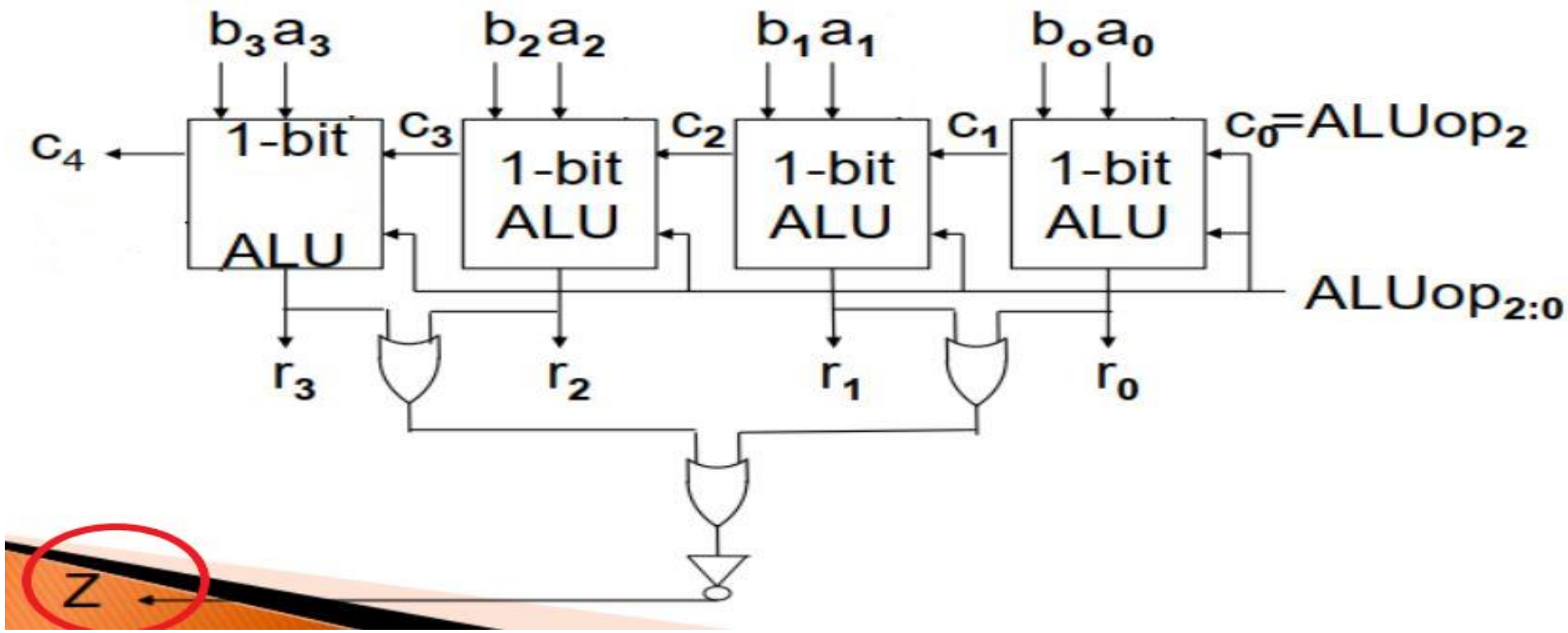
ALUOp	Function
000	AND
001	OR
010	ADD
110	SUBTRACT
111	XOR



1 bit ALU Test Results

```
# time= 0,a=0,b=0,alu_ctr=000,cin=0,r=0,cout=0  and
# time=20,a=1,b=0,alu_ctr=001,cin=0,r=1,cout=0  or
# time=40,a=1,b=1,alu_ctr=010,cin=1,r=1,cout=1  add
# time=60,a=1,b=1,alu_ctr=110,cin=1,r=0,cout=1  sub
# time=80,a=0,b=1,alu_ctr=111,cin=0,r=1,cout=0  xor
```

32 bit Updated ALU(4 bit version)



32 bit Updated ALU Test Results

```
# time= 0,a=10,b=30,alu_ctr=000,r=10,cout=0,z=0      and
# time=20,a= 0,b=20,alu_ctr=000,r= 0,cout=0,z=1      and
# time=40,a=10,b=30,alu_ctr=001,r=30,cout=0,z=0      or
# time=60,a= 0,b=20,alu_ctr=001,r=20,cout=0,z=0      or
# time=80,a=10,b=30,alu_ctr=010,r=40,cout=0,z=0      add
# time=100,a= 0,b=20,alu_ctr=010,r=20,cout=0,z=0     add
# time=120,a=10,b=30,alu_ctr=110,r=-20,cout=0,z=0    sub
# time=140,a= 0,b=20,alu_ctr=110,r=-20,cout=0,z=0    sub
# time=160,a=100,b=99,alu_ctr=110,r= 1,cout=1,z=0    sub
# time=180,a=10,b=30,alu_ctr=111,r=20,cout=0,z=0     xor
# time=200,a= 0,b=20,alu_ctr=111,r=20,cout=0,z=0     xor
# time=220,a=20,b=20,alu_ctr=111,r= 0,cout=1,z=1     xor
```

I didn’t consider V and set outputs of ALU, since we don’t need them in these 14 instructions.

Comparator Design

In zero comparator design first, I checked sign bit of number if it is 1 then number is less than 0, otherwise I checked zero equality of number by using xnor gate.

Zero Comparator results (number is in decimal format)

```
# time= 0,number= 0,equal=1,bigger=0,less=0
# time=20,number=270851,equal=0,bigger=1,less=0
# time=40,number=-266845678,equal=0,bigger=0,less=1
```

Sign extender Design Test Results(numbers are in decimal format)

```
# time= 0,number= 0,result= 0          zero
# time=20,number=1157,result=1157      positive
# time=40,number=-3559,result=-3559    negative
```

- I designed sign extender as 16 to 32 bits. Because my ALU block 32 bit and I need to make add operation by using ALU therefore my extender result should be 32 bits. Since, our data memory requires 18 bits instead of 32 bits I will only use first 18 bit of ALU result(we don't need rest of it). **Note that our CPU is 32 bits** therefore, extender must be 16 to 32.

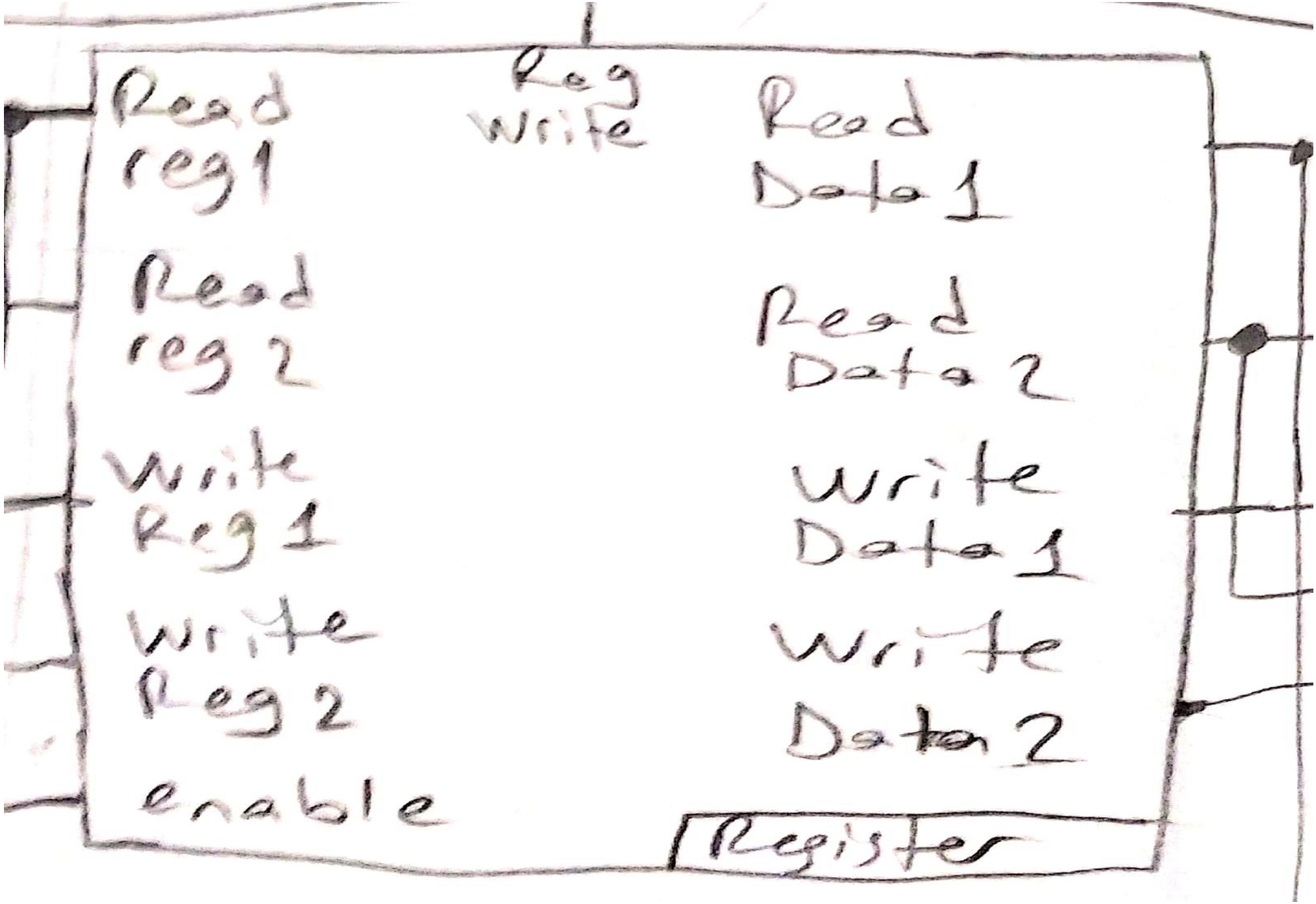
Zero extender Design Test Results(numbers are in binary format)

```
# time= 0,number=0000000000000000,result=0000000000000000000000000000000 zero
# time=20,number=0000010010000101,result=0000000000000000000010010000101 positive
# time=40,number=1111001000011001,result=0000000000000000111100100001100 negative
```

Shift Left Logical 16 bit Design Test Results(will be used in lui instr.)

```
# time= 0,number=00000000000000000000000000000000,result=00000000000000000000000000000000
# time=20,number=01100000000001001010001001111111,result=10100010011111110000000000000000
# time=40,number=11110001100110000111111000000010,result=01111110000000100000000000000000
```

Register Block (Writes two different register addresses in one cycle)



Register Block Test Results

register_data_in file before testing.

```
00000000000000000000000000001101
00000000000000000000000000010001
00000000000000000000000000111010
00000000000000000000010001010001
11111111111111111111110111101011 11
11111111111111111111111111111011
00000000000010100000011100010110 1
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
.
.
.
```

Only first 7 of them was used for read operation.

File operations were made in testbench. Not in register block as denoted in homework.

Test Results of register Block

```
# time= 0, read_reg1= 0,  read_reg2= 1,  write_reg1=10,  write_reg2=11,  write_data1=4294967295,   write_data2= 0,  enable=0,  reg_write=1,   read_data1=13,   read_data2=17 ,  clk= 1
# time=20, read_reg1= 0,  read_reg2= 1,  write_reg1=10,  write_reg2=11,  write_data1=4294967295,   write_data2= 0,  enable=0,  reg_write=1,   read_data1=13,   read_data2=17 ,  clk= 0
# time=40, read_reg1= 2,  read_reg2= 3,  write_reg1=11,  write_reg2=12,  write_data1=29368965,   write_data2=4027057152,  enable=1,  reg_write=1,   read_data1=58,   read_data2=1105 ,  clk= 1
# time=60, read_reg1= 2,  read_reg2= 3,  write_reg1=11,  write_reg2=12,  write_data1=29368965,   write_data2=4027057152,  enable=1,  reg_write=1,   read_data1=58,   read_data2=1105 ,  clk= 0
# time=80, read_reg1= 4,  read_reg2= 5,  write_reg1=13,  write_reg2=14,  write_data1=100,   write_data2=3354644,  enable=1,  reg_write=0,   read_data1=4294966191,   read_data2=4294967291 ,  clk= 1
# time=100, read_reg1= 4,  read_reg2= 5,  write_reg1=13,  write_reg2=14,  write_data1=100,   write_data2=3354644,  enable=1,  reg_write=0,   read_data1=4294966191,   read_data2=4294967291 ,  clk= 0
# time=120, read_reg1= 6,  read_reg2= 7,  write_reg1=15,  write_reg2=16,  write_data1=100,   write_data2=20,  enable=0,  reg_write=0,   read_data1=658989,   read_data2= 0 ,  clk= 1
# time=140, read_reg1= 6,  read_reg2= 7,  write_reg1=15,  write_reg2=16,  write_data1=100,   write_data2=20,  enable=0,  reg_write=0,   read_data1=658989,   read_data2= 0 ,  clk= 0
# time=160, read_reg1= 6,  read_reg2= 7,  write_reg1=15,  write_reg2=16,  write_data1=100,   write_data2=20,  enable=0,  reg_write=0,   read_data1=658989,   read_data2= 0 ,  clk= 1
```

Test values are in decimal format to gain space.

register_data_out file after testing.

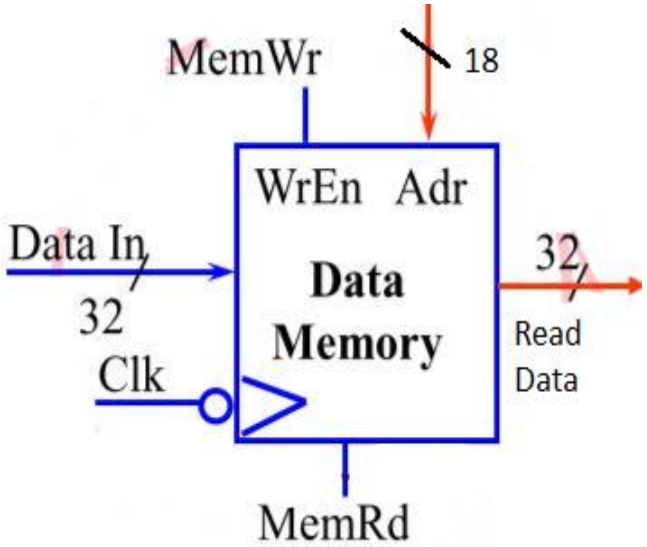
As you can see 13th register didn't change and 10, 11, 12, 14th registers were written from outside.

```
00000000000000000000000000001101
00000000000000000000000000010001
00000000000000000000000000111010
00000000000000000000010001010001
11111111111111111111101110101111
11111111111111111111111111111011
00000000000101000001110001011101
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
11111111111111111111111111111111 10
00000001110000000001000101000101 11
11110000000010000000010000000000 12
00000000000000000000000000000000 13
00000000001100110011000000010100 14
00000000000000000000000000000000
00000000000000000000000000000000
.
.
.
```

Before Testing

```
1   00000000000000000000000000001101
2   00000000000000000000000000010001
3   00000000000000000000000000111010
4   00000000000000000000010001010001
5   11111111111111111111101110101111
6   11111111111111111111111111111011
7   00000000000101000001110001011101
8   00000000000000000000000000000000
9   00000000000000000000000000000000
10  00000000000000000000000000000000
.
.
.
```

Data Memory Block



Data Memory file before test

```
00000000000000000000000000000000
00000000000000000000010011000111
00000000000000000000000000000000
00000000000000000000000000000000
```

Memory Block Test Results(read data results)

read_data=00000000000000000000010011000111 read_data=00000000000000000000010011000111	1
read_data=00000000000000000000010011000111 read_data=00000000000000000000010011000111	2
read_data=00000000000000000000010011000111 read_data=00000000000000000000010011000111	3

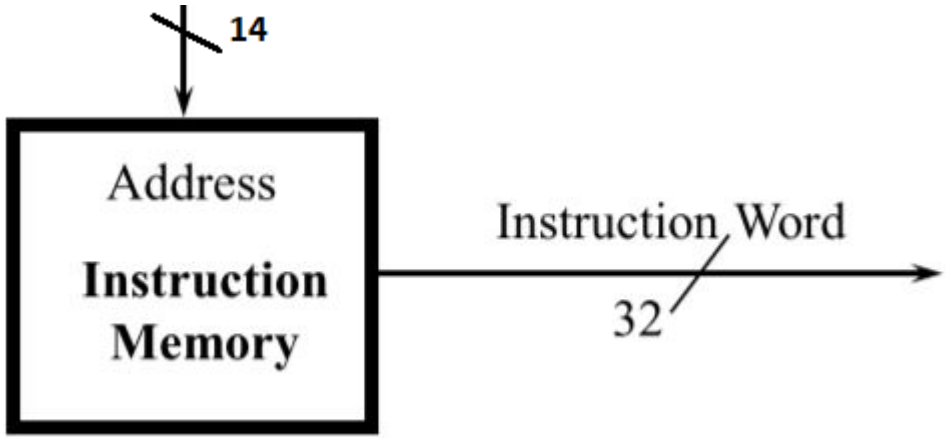
# time= 0,	write_data=11111111111111111111111111111111,	adress= 1,	mem_read=1,	mem_write=0,	clk=1,	1
# time=20,	write_data=11111111111111111111111111111111,	adress= 1,	mem_read=1,	mem_write=0,	clk=0,	
# time=40,	write_data=11111111111111111111111111111111,	adress= 0,	mem_read=0,	mem_write=1,	clk=1,	2
# time=60,	write_data=11111111111111111111111111111111,	adress= 0,	mem_read=0,	mem_write=1,	clk=0,	
# time=80,	write_data=00000111111111111111111111111111,	adress= 0,	mem_read=0,	mem_write=0,	clk=1,	3
# time=100,	write_data=00000111111111111111111111111111,	adress= 0,	mem_read=0,	mem_write=0,	clk=0,	

***1st row read from data memory file in test.**

Data memory out file after testing(saved with different name)(0th address data was changed after test)

```
11111111111111111111111111111111
00000000000000000000010011000111
00000000000000000000000000000000
00000000000000000000000000000000
.
.
.
```

Instruction Memory Block



Instruction Memory File

```
1   00000000000000000000000000000000
2   00000001010010110100100000100000
3   00000000000000000000000000000000
4   00000000000000000000000000000000
5   00000000000000000000000000000000
6   10001101000010000000000000001100
7   00000000000000000000000000000000
8   00000000000000000000000000000000
9   00000000000000000000000000000000
10  10101101000010000000000000001100
11  00000000000000000000000000000000
12  00000000000000000000000000000000
.
.
.
```

Instruction Memory Block Test Results

```
# time= 0, read_adress= 1,  clk=1,  instruction=00000001010010110100100000100000
# time=20, read_adress= 1,  clk=0,  instruction=00000001010010110100100000100000
# time=40, read_adress= 5,  clk=1,  instruction=10001101000010000000000000001100
# time=60, read_adress= 5,  clk=0,  instruction=10001101000010000000000000001100
# time=80, read_adress= 9,  clk=1,  instruction=10101101000010000000000000001100
# time=100, read_adress= 9,  clk=0,  instruction=10101101000010000000000000001100
```

Instruction Splitter (Parser)

Splitter Test Results (We don't need to shamt field in datapath.)(Instructions codes are garbage in this test.)

```
# time= 0, instruction=01100010100110001000010000010010,  opcode=011000,  rs=10100,  rt=11000,  rd=10000, 1
# time=20, instruction=10100001000001000010001000000011,  opcode=101000,  rs=01000,  rt=00100,  rd=00100, 2
# time=40, instruction=11110000000110000100001000010010,  opcode=111100,  rs=00000,  rt=11000,  rd=01000, 3
```

```
function=010010,  immediate=1000010000010010,  adress=10100110001000010000010010, 1
function=000011,  immediate=0010001000000011,  adress=01000001000010001000000011, 2
function=010010,  immediate=0100001000010010,  adress=00000110000100001000010010, 3
```

32 Bit Full Adder Test Results

```
# time= 0, a=37, b=63, carry_in=0, carry_out=0, result=100
# time=20, a=152, b=63, carry_in=0, carry_out=0, result=215
# time=40, a=-10, b=-23, carry_in=0, carry_out=1, result=-33
# time=60, a=-3565, b=63, carry_in=0, carry_out=0, result=-3502
```

Note: I take 1 bit full adder design **directly from PS_1 files on moodle** because of **time constraint**. Instead of I could use my ALU block to make addition but I didn't want to do that. I used this adder in program counter block.

Program Counter Block

Program counter block produces new program counter value according to given instruction.

Program Counter Block Test Results

```
# time= 0,jump=0,jal=0,jr=0,beq=0,bne=0,zero=0,adress= 5,rs=10,ext_immed= 8,pc= 0,pc_artil= 1,new_pc= 1
# time=20,jump=1,jal=0,jr=0,beq=0,bne=0,zero=0,adress= 5,rs=10,ext_immed= 8,pc= 0,pc_artil= 1,new_pc= 5
# time=40,jump=0,jal=1,jr=0,beq=0,bne=0,zero=0,adress= 5,rs=10,ext_immed= 8,pc= 0,pc_artil= 1,new_pc= 5
# time=60,jump=0,jal=0,jr=1,beq=0,bne=0,zero=0,adress= 5,rs=10,ext_immed= 8,pc= 0,pc_artil= 1,new_pc=10
# time=80,jump=0,jal=0,jr=0,beq=1,bne=0,zero=1,adress= 5,rs=10,ext_immed= 8,pc= 3,pc_artil= 4,new_pc=12
# time=100,jump=0,jal=0,jr=0,beq=0,bne=1,zero=0,adress= 5,rs=10,ext_immed= 8,pc= 2,pc_artil= 3,new_pc=11
```

Our helper modules are done step by step.

Let's combine them in mips32.

First, I will show 14 instructions separately then I will test them in same program.

Let's start with R-type-n instructions because we need to these instructions in jump, beg etc. to pass and see they really didn't work.

For reading I am using between 4-8th registers, results will be between 10-19th registers(rd).

addn

Instruction Memory File (I didn't put screenshot since there will be ambiguity, you can check files after tests)

Machine Code	Instruction	Adress
00000000100001010101000000100000	addn $t2,$a0,$a1	0
00000000101001100101100000100000	addn $t3,$a1,$a2	1

Register_data_in File

Adress	Data	Value
4 (a0)	00000000000000000000000000001101	13
5 (a1)	00000000000000000000000000010001	17
6 (a2)	00000000000000000000000000011001	25
7 (a3)	11111111111111111111111111111011	-5
8 (t0)	00000000000000000000010111011100	1500
9 (t1)	00000000000000000000000000000000	0
10 (t2)	00000000000000000000000000000000	0
. . .	. . .	. . .

Register_data_out file(data was taken directly from file)

Adress	Data	Value
4 (a0)	00000000000000000000000000011110	30
5 (a1)	00000000000000000000000000101010	42
6 (a2)	00000000000000000000000000011001	25
7 (a3)	11111111111111111111111111111011	-5
8 (t0)	00000000000000000000010111011100	1500
9 (t1)	00000000000000000000000000000000	0
10 (t2)	00000000000000000000000000000011	3
11(t3)	00000000000000000000000000000011	3

subn

Instruction Memory File

Machine Code	Instruction	Adress
00000000111010000110000000100010	subn $t4,$a3,$t0	0
00000000101001100110100000100010	subnn $t5,$a1,$a2	1

Register_data_in File

Adress	Data	Value
4 (a0)	00000000000000000000000000001101	13
5 (a1)	00000000000000000000000000010001	17
6 (a2)	00000000000000000000000000011001	25
7 (a3)	11111111111111111111111111111011	-5
8 (t0)	00000000000000000000010111011100	1500
9 (t1)	00000000000000000000000000000000	0
10 (t2)	00000000000000000000000000000000	0
. . .	. . .	. . .

Register_data_out file

Adress	Data	Value
4 (a0)	00000000000000000000000000001101	13
5 (a1)	11111111111111111111111111111000	-8
6 (a2)	00000000000000000000000000011001	25
7 (a3)	11111111111111111111101000011111	-1505
8 (t0)	00000000000000000000010111011100	1500
9 (t1)	00000000000000000000000000000000	0
12 (t4)	00000000000000000000000000000010	2
13(t5)	00000000000000000000000000000010	2

andn

Instruction Memory File

Machine Code	Instruction	Adress
00000000101001100111000000100100	andn $t6, $a1, $a2	0
00000001000001000111100000100100	andn $t7, $t0, $a0	1

Register_data_in File

Adress	Data	Value
4 (a0)	00000000000000000000000000001101	13
5 (a1)	00000000000000000000000000010001	17
6 (a2)	00000000000000000000000000011001	25
7 (a3)	11111111111111111111111111111011	-5
8 (t0)	00000000000000000000010111011100	1500
9 (t1)	00000000000000000000000000000000	0
10 (t2)	00000000000000000000000000000000	0
. . .	. . .	. . .

Register_data_out file

Adress	Data	Value
4 (a0)	00000000000000000000000000001101	13
5 (a1)	00000000000000000000000000010001	17
6 (a2)	00000000000000000000000000011001	25
7 (a3)	11111111111111111111111111111011	-5
8 (t0)	00000000000000000000000000001100	12
9 (t1)	00000000000000000000000000000000	0
14(t6)	00000000000000000000000000000011	3
15(t7)	00000000000000000000000000000011	3

orn

Instruction Memory File

Machine Code	Instruction	Adress
00000000110001111000000000100101	orn $s0, $a2,$a3	0
00000001000001011000100000100101	orn $s1, $t0,$a1	1

Register_data_in File

Adress	Data	Value
4 (a0)	00000000000000000000000000001101	13
5 (a1)	00000000000000000000000000010001	17
6 (a2)	00000000000000000000000000011001	25
7 (a3)	11111111111111111111111111111011	-5
8 (t0)	00000000000000000000010111011100	1500
9 (t1)	00000000000000000000000000000000	0
10 (t2)	00000000000000000000000000000000	0
. . .	. . .	. . .

Register_data_out file

Adress	Data	Value
4 (a0)	00000000000000000000000000001101	30
5 (a1)	00000000000000000000000000010001	42
6 (a2)	11111111111111111111111111111011	-5
7 (a3)	11111111111111111111111111111011	-5
8 (t0)	00000000000000000000010111011101	1501
9 (t1)	00000000000000000000000000000000	0
16 (s0)	00000000000000000000000000000010	2
17(s1)	00000000000000000000000000000011	3

xorn

Instruction Memory File

Machine Code	Instruction	Adress
00000000111010001001000000101010	xorn $s2,$a3,$t0	0
00000001001010101001100000101010	xorn $s3,$t1,$t2	1

Register_data_in File

Adress	Data	Value
4 (a0)	00000000000000000000000000001101	13
5 (a1)	00000000000000000000000000010001	17
6 (a2)	00000000000000000000000000011001	25
7 (a3)	11111111111111111111111111111011	-5
8 (t0)	00000000000000000000010111011100	1500
9 (t1)	00000000000000000000000000000000	0
10 (t2)	00000000000000000000000000000000	0
. . .	. . .	. . .

Register_data_out file

Adress	Data	Value
4 (a0)	00000000000000000000000000001101	13
5 (a1)	00000000000000000000000000010001	17
6 (a2)	00000000000000000000000000011001	25
7 (a3)	11111111111111111111101000100111	-1497
8 (t0)	00000000000000000000010111011100	1500
9 (t1)	00000000000000000000000000000000	0
18 (s2)	00000000000000000000000000000010	2
19(s3)	00000000000000000000000000000001	1

Register_data_out file screenshot after xor operation

register_data_out.txt - Notepad

File Edit Format View Help

```
// memory data file (do not edit the following line - required for mem load use)
// instance=/mips32_testbench/my_processor/regs/registers
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000001101
00000000000000000000000000010001
00000000000000000000000000011001
11111111111111111111101000100111
00000000000000000000101110111100
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000010
00000000000000000000000000000001
00000000000000000000000000000000
00000000000000000000000000000000
```

***Understanding is not easy here without my tables.**

lw

Instruction Memory File

Machine Code	Instruction	Adress
10001101001010100000000000000000	lw $t2, 0($t1)	0
10001101001010110000000000000001	lw $t3, 1($t1)	1

Register_data_in File

Adress	Data	Value
4 (a0)	00000000000000000000000000001101	13
5 (a1)	00000000000000000000000000010001	17
6 (a2)	00000000000000000000000000011001	25
7 (a3)	11111111111111111111111111111011	-5
8 (t0)	00000000000000000000010111011100	1500
9 (t1)	00000000000000000000000000000000	0
10 (t2)	00000000000000000000000000000000	0
. . .	. . .	. . .

Register_data_out file

Adress	Data	Value
4 (a0)	00000000000000000000000000001101	13
5 (a1)	00000000000000000000000000010001	17
6 (a2)	00000000000000000000000000011001	25
7 (a3)	11111111111111111111111111111011	-5
8 (t0)	00000000000000000000010111011100	1500
9 (t1)	00000000000000000000000000000000	0
10 (t2)	00000000000000000000010011000111	1223
11(t3)	11111111111111111111110100010010	-750

Memory_data file

Adress	Data	Value
0	00000000000000000000010011000111	1223
1	11111111111111111111110100010010	-750
2	00000000000000000000000000000000	0
. .	. .	. .

SW

Instruction Memory File

Machine Code	Instruction	Adress
10101101001010000000000000000010	sw $t0, 2($t1)	0
10101101001001010000000000000011	sw $a1, 3($t1)	1

Register_data_in File

Adress	Data	Value
4 (a0)	00000000000000000000000000001101	13
5 (a1)	00000000000000000000000000010001	17
6 (a2)	00000000000000000000000000011001	25
7 (a3)	11111111111111111111111111111011	-5
8 (t0)	00000000000000000000010111011100	1500
9 (t1)	00000000000000000000000000000000	0
10 (t2)	00000000000000000000000000000000	0
. . .	. . .	. . .

Memory_data_out file

Adress	Data	Value
0	00000000000000000000010011000111	1223
1	11111111111111111111110100010010	-750
2	00000000000000000000010111011100	1500
3	00000000000000000000000000010001	17
4	00000000000000000000000000000000	0
5	00000000000000000000000000000000	0
6	00000000000000000000000000000000	0
. . .	. . .	. . .

Memory_data file(before test)

Adress	Data	Value
0	00000000000000000000010011000111	1223
1	11111111111111111111110100010010	-750
2	00000000000000000000000000000000	0
. .	. .	. .

ori

Instruction Memory File

Machine Code	Instruction	Adress
00110100100010100000000000000111	ori $t2,$a0,0x7	0
00110101000010110000000000001001	ori $t3,$t0,0x9	1

Register_data_in File

Adress	Data	Value
4 (a0)	00000000000000000000000000001101	13
5 (a1)	00000000000000000000000000010001	17
6 (a2)	00000000000000000000000000011001	25
7 (a3)	11111111111111111111111111111011	-5
8 (t0)	00000000000000000000010111011100	1500
9 (t1)	00000000000000000000000000000000	0
10 (t2)	00000000000000000000000000000000	0
. . .	. . .	. . .

Register_data_out file

Adress	Data	Value
4 (a0)	00000000000000000000000000001101	13
5 (a1)	00000000000000000000000000010001	17
6 (a2)	00000000000000000000000000011001	25
7 (a3)	11111111111111111111111111111011	-5
8 (t0)	00000000000000000000010111011100	1500
9 (t1)	00000000000000000000000000000000	0
10 (t2)	00000000000000000000000000001111	15
11(t3)	00000000000000000000010111011101	1501

lui

Instruction Memory File

Machine Code	Instruction	Adress
00111100000010100000000011111111	lui $t2,255	0
00111100000010110000000000001111	lui $t3,15	1

Register_data_in File

Adress	Data	Value
4 (a0)	00000000000000000000000000001101	13
5 (a1)	00000000000000000000000000010001	17
6 (a2)	00000000000000000000000000011001	25
7 (a3)	11111111111111111111111111111011	-5
8 (t0)	00000000000000000000010111011100	1500
9 (t1)	00000000000000000000000000000000	0
10 (t2)	00000000000000000000000000000000	0
. . .	. . .	. . .

Register_data_out file

Adress	Data	Value
4 (a0)	00000000000000000000000000001101	13
5 (a1)	00000000000000000000000000010001	17
6 (a2)	00000000000000000000000000011001	25
7 (a3)	11111111111111111111111111111011	-5
8 (t0)	00000000000000000000010111011100	1500
9 (t1)	00000000000000000000000000000000	0
10 (t2)	00000000111111110000000000000000	16711680
11(t3)	00000000000011110000000000000000	983040

j, beq, bne

Instruction Memory File (red's will not evaluated only green and yellows will work)

Machine Code	Instruction	Adress
00010001010010010000000000000010	beq $t2, $t1, 2	0
00000000111010000101000000100000	addn $t2, $a3, $t0	1
00000000100010000101100000100101	orn $t3, $a0,$t0	2
00001000000000000000000000000110	j 6	3
00000000110001010110000000100100	andn $t4, $a2, $a1	4
00000000100001110110100000100010	subn $t5,$a0,$a3	5
00010100100001010000000000000010	bne $a0, $a1, 2	6
10001101001010100000000000000000	lw $t2, 0($t1)	7
10101101001001010000000000000011	sw $a1, 3($t1)	8
00000000110001110111000000100000	addn $t6,$a2,$a3	9

Register_data_in File

Adress	Data	Value
4 (a0)	00000000000000000000000000001101	13
5 (a1)	00000000000000000000000000010001	17
6 (a2)	00000000000000000000000000011001	25
7 (a3)	11111111111111111111111111111011	-5
8 (t0)	00000000000000000000010111011100	1500
9 (t1)	00000000000000000000000000000000	0
10 (t2)	00000000000000000000000000000000	0
. . .	. . .	. . .

register_data_out file

Adress	Data	Value
4 (a0)	00000000000000000000000000001101	13
5 (a1)	00000000000000000000000000010001	17
6 (a2)	00000000000000000000000000010100	20
7 (a3)	11111111111111111111111111111011	-5
8 (t0)	00000000000000000000010111011100	1500
9 (t1)	00000000000000000000000000000000	0
10 (t2)	00000000000000000000000000000000	0
14(t6)	00000000000000000000000000000011	3

Memory_data_out file(nothing changed)

Adress	Data	Value
0	00000000000000000000010011000111	1223
1	11111111111111111111110100010010	-750
2	00000000000000000000000000000000	0
. . .	. . .	. . .

Change of Program Counter in previous example

```
# clk= 1,  pc= 0, rs= 4, opcode=10, rt= 9, alu_res= 0,  write_data1= 1, write_data2= 0, instr=00010001010010010000000000000010,
m_wrt=0, branch=1  alu_op=01, jump=0, jal=0,jr=0, bne=0, enable=0, ori=0, lui=0
# clk= 0,  pc= 3, rs= 4, opcode=10, rt= 9, alu_res= 0,  write_data1= 1, write_data2= 0, instr=00010001010010010000000000000010,
m_wrt=0, branch=1  alu_op=01, jump=0, jal=0,jr=0, bne=0, enable=0, ori=0, lui=0
# clk= 1,  pc= 3, rs= 2, opcode= 0, rt= 0, alu_res= 0,  write_data1= 1, write_data2= 0, instr=00001000000000000000000000000110,
m_wrt=0, branch=0  alu_op=00, jump=1, jal=0,jr=0, bne=0, enable=0, ori=0, lui=0
# clk= 0,  pc= 6, rs= 2, opcode= 0, rt= 0, alu_res= 0,  write_data1= 1, write_data2= 0, instr=00001000000000000000000000000110,
m_wrt=0, branch=0  alu_op=00, jump=1, jal=0,jr=0, bne=0, enable=0, ori=0, lui=0
# clk= 1,  pc= 6, rs= 5, opcode= 4, rt= 5, alu_res=4294967292,  write_data1= 2, write_data2=4294967292, instr=00010100100001010000000000000010,
, mem_read=0, mem_wrt=0, branch=0, alu_op=01, jump=0, jal=0,jr=0, bne=1, enable=0, ori=0, lui=0
# clk= 0,  pc= 9, rs= 5, opcode= 4, rt= 5, alu_res=4294967292,  write_data1= 2, write_data2=4294967292, instr=00010100100001010000000000000010,
, mem_read=0, mem_wrt=0, branch=0, alu_op=01, jump=0, jal=0,jr=0, bne=1, enable=0, ori=0, lui=0
# clk= 1,  pc= 9, rs= 0, opcode= 6, rt= 7, alu_res=20,  write_data1= 3, write_data2=20, instr=00000000110001110111000000100000,
m_wrt=0, branch=0  alu_op=10, jump=0, jal=0,jr=0, bne=0, enable=1, ori=0, lui=0
```

jal, jr

Instruction Memory File (red's will not evaluated only green and yellows will work) (beq will not jump here)

Machine Code	Instruction	Adress
00001100000000000000000000000100	jal 4	0
00000000111010000101000000100000	addn $t2, $a3, $t0	1
00010000100001010000000000000010	beq $a0, $a1, 2	2
00001000000000000000000000001001	j 9	3
00100011111000000000000000000000	jr $ra	4
00000000100001110110100000100010	subn $t5,$a0,$a3	5
00010100100001010000000000000010	bne $a0, $a1, 2	6
10001101001010100000000000000000	lw $t2, 0($t1)	7
10101101001001010000000000000011	sw $a1, 3($t1)	8
00000001000001010111000000100000	addn $t6,$t0,$a1	9

Register_data_in File

Adress	Data	Value
4 (a0)	00000000000000000000000000001101	13
5 (a1)	00000000000000000000000000010001	17
6 (a2)	00000000000000000000000000011001	25
7 (a3)	11111111111111111111111111111011	-5
8 (t0)	00000000000000000000010111011100	1500
9 (t1)	00000000000000000000000000000000	0
10 (t2)	00000000000000000000000000000000	0
.	.	.
.	.	.

Memory_data_out file

Adress	Data	Value
4 (a0)	00000000000000000000000000001101	13
5 (a1)	00000000000000000000000000010001	17
6 (a2)	00000000000000000000000000011001	25
7 (a3)	00000000000000000000010111010111	1495
8 (t0)	00000000000000000000010111101101	1517
9 (t1)	00000000000000000000000000000000	0
10 (t2)	00000000000000000000000000000011	3
14(t6)	00000000000000000000000000000011	3
31(ra)	00000000000000000000000000000001	1

Memory_data_out file(nothing changed)

Adress	Data	Value
0	00000000000000000000010011000111	1223
1	11111111111111111111110100010010	-750
2	00000000000000000000000000000000	0
. . .	. . .	. . .

Change of Program Counter in previous example

```
# clk= 1,  pc= 0,  rs= 3, opcode= 0, rt= 0, alu_res= 0,  write_data1= 1, write_data2= 0, instr=00001100000000000000000000000100,
m_wrt=0, branch=0, alu_op=00, jump=0, jal=1,jr=0, bne=0, enable=0, ori=0, lui=0
# clk= 0,  pc= 4,  rs= 3, opcode= 0, rt= 0, alu_res= 0,  write_data1= 5, write_data2= 0, instr=00001100000000000000000000000100,
m_wrt=0, branch=0, alu_op=00, jump=0, jal=1,jr=0, bne=0, enable=0, ori=0, lui=0
# clk= 1,  pc= 4,  rs= 8, opcode=31, rt= 0, alu_res= 1,  write_data1= 3, write_data2= 1, instr=00100011111000000000000000000000,
m_wrt=0, branch=0, alu_op=00, jump=0, jal=0,jr=1, bne=0, enable=0, ori=0, lui=0
# clk= 0,  pc= 1,  rs= 8, opcode=31, rt= 0, alu_res= 1,  write_data1= 3, write_data2= 1, instr=00100011111000000000000000000000,
m_wrt=0, branch=0, alu_op=00, jump=0, jal=0,jr=1, bne=0, enable=0, ori=0, lui=0
# clk= 1,  pc= 1,  rs= 0, opcode= 7, rt= 8, alu_res=1495,  write_data1= 3, write_data2=1495, instr=00000000111010000101000000100000,
, mem_wrt=0, branch=0, alu_op=10, jump=0, jal=0,jr=0, bne=0, enable=1, ori=0, lui=0
# clk= 0,  pc= 2,  rs= 0, opcode= 7, rt= 8, alu_res=2995,  write_data1= 3, write_data2=2995, instr=00000000111010000101000000100000,
, mem_wrt=0, branch=0, alu_op=10, jump=0, jal=0,jr=0, bne=0, enable=1, ori=0, lui=0
# clk= 1,  pc= 2,  rs= 4, opcode= 4, rt= 5, alu_res=4294967292,  write_data1= 2, write_data2=4294967292, instr=00010000100001010000000000000010,
, mem_read=0, mem_wrt=0, branch=1, alu_op=01, jump=0, jal=0,jr=0, bne=0, enable=0, ori=0, lui=0
# clk= 0,  pc= 3,  rs= 4, opcode= 4, rt= 5, alu_res=4294967292,  write_data1= 2, write_data2=4294967292, instr=00010000100001010000000000000010,
, mem_read=0, mem_wrt=0, branch=1, alu_op=01, jump=0, jal=0,jr=0, bne=0, enable=0, ori=0, lui=0
# clk= 1,  pc= 3,  rs= 2, opcode= 0, rt= 0, alu_res= 0,  write_data1= 1, write_data2= 0, instr=00001000000000000000000000001001,
m_wrt=0, branch=0, alu_op=00, jump=1, jal=0,jr=0, bne=0, enable=0, ori=0, lui=0
# clk= 0,  pc= 9,  rs= 2, opcode= 0, rt= 0, alu_res= 0,  write_data1= 1, write_data2= 0, instr=00001000000000000000000000001001,
m_wrt=0, branch=0, alu_op=00, jump=1, jal=0,jr=0, bne=0, enable=0, ori=0, lui=0
# clk= 1,  pc= 9,  rs= 0, opcode= 8, rt= 5, alu_res=1517,  write_data1= 3, write_data2=1517, instr=00000001000001010111000000100000,
, mem_wrt=0, branch=0, alu_op=10, jump=0, jal=0,jr=0, bne=0, enable=1, ori=0, lui=0
```

Lastly , Test all of them in same program.

Instruction Memory File

Machine Code	Instruction	Adress
00000000100001010101000000100000	addn $t2,$a0,$a1	1
00000000111010000110000000100010	subn $t4,$a3,$t0	2
00001000000000000000000000000110	j 6	3
00000001000001011001100000100000	addn $s3,$t0,$a1	4
00000000110001011010000000100010	subn $s4,$a2,$a1	5
00000000101001100111000000100100	andn $t6, $a1, $a2	6
00000000110001111000000000100101	orn $s0, $a2,$a3	7
00001100000000000000000000001101	jal 13	8
00010100101001100000000000000011	bne $a1, $a2, 3	9
00000001000001011010100000100100	andn $s5,$t0,$a1	10
00000000110001011011000000100101	orn $s6,$a2,$a1	11
00000001000001011100000000100010	subn $t8,$t0,$a1	12
00010001000010010000000000000011	beq $t0, $t1, 3	13
00111100000100010000011011011011	lui $s1,1755	14
10001110111010000000000000001100	lw $t0, 12($s7)	15
00100011111000000000000000000000	jr ra	16
00000001000001111001000000101010	xorn $s2,$t0,$a3	17
10001101001010110000000000000001	lw $t3, 1($t1)	18
10101101001001110000000000000010	sw $a3, 2($t1)	19
00110101000011110000000000001011	ori $t7,$t0,0xB	20
00001000000000000000000000010111	j 23	21
10001101001110010000000000000001	lw $t9, 1($t1)	22

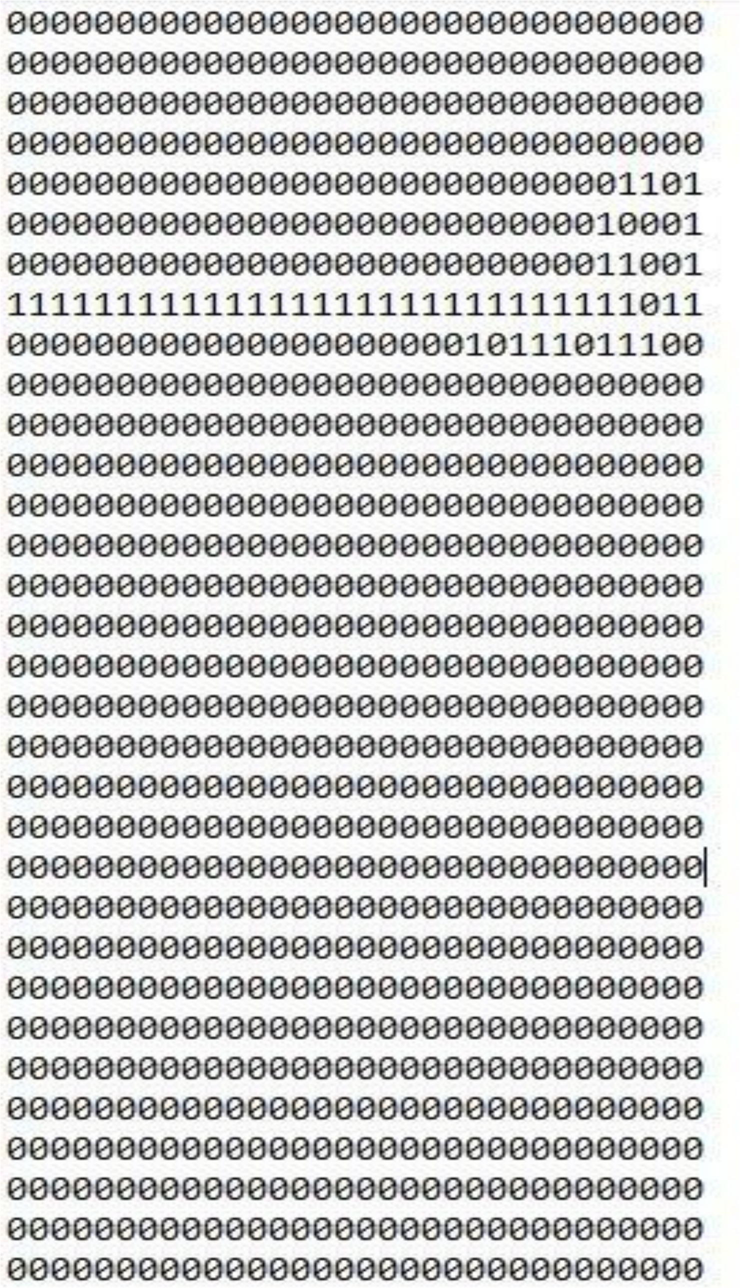
memory_data_file(before test)

```
00000000000000000000010011000111
11111111111111111111110100010010
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
```

memory_data_out_file(after test)

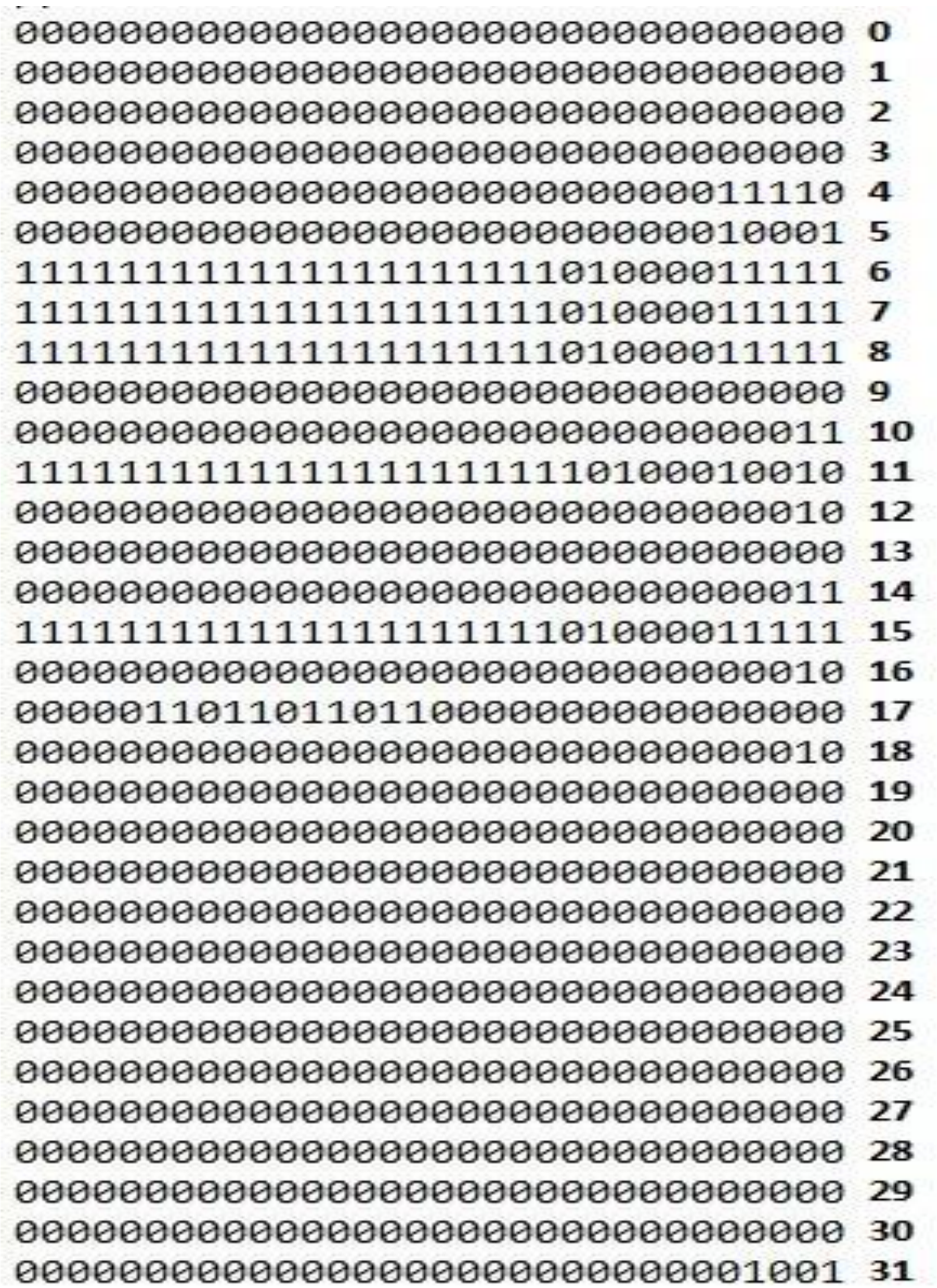
```
00000000000000000000010011000111
11111111111111111111110100010010
11111111111111111111101000011111
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
```

register_data_in file



```
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000001101
00000000000000000000000000010001
00000000000000000000000000011001
11111111111111111111111111111011
00000000000000000000010111011100
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
```

register_data_out_file



```
00000000000000000000000000000000 0
00000000000000000000000000000000 1
00000000000000000000000000000000 2
00000000000000000000000000000000 3
00000000000000000000000000011110 4
00000000000000000000000000010001 5
11111111111111111111101000011111 6
11111111111111111111101000011111 7
11111111111111111111101000011111 8
00000000000000000000000000000000 9
00000000000000000000000000000011 10
11111111111111111111101000010010 11
00000000000000000000000000000010 12
00000000000000000000000000000000 13
00000000000000000000000000000011 14
11111111111111111111101000011111 15
00000000000000000000000000000010 16
00000110110110110000000000000000 17
00000000000000000000000000000010 18
00000000000000000000000000000000 19
00000000000000000000000000000000 20
00000000000000000000000000000000 21
00000000000000000000000000000000 22
00000000000000000000000000000000 23
00000000000000000000000000000000 24
00000000000000000000000000000000 25
00000000000000000000000000000000 26
00000000000000000000000000000000 27
00000000000000000000000000000000 28
00000000000000000000000000000000 29
00000000000000000000000000000000 30
00000000000000000000000000001001 31
```

Change of program counter

```
# clk= 1,  pc= 0, rs= 0, opcode= 0, rt
m_wrt=0, branch=0, alu_op=10, jump=0,
# clk= 0,  pc= 1, rs= 0, opcode= 0, rt
m_wrt=0, branch=0, alu_op=10, jump=0,
# clk= 1,  pc= 1, rs= 0, opcode= 4, rt
m_wrt=0, branch=0, alu_op=10, jump=0,
# clk= 0,  pc= 2, rs= 0, opcode= 4, rt
m_wrt=0, branch=0, alu_op=10, jump=0,
# clk= 1,  pc= 2, rs= 0, opcode= 7, rt
, mem_read=0, mem_wrt=0, branch=0, alu
# clk= 0,  pc= 3, rs= 0, opcode= 7, rt
, mem_read=0, mem_wrt=0, branch=0, alu
# clk= 1,  pc= 3, rs= 2, opcode= 0, rt
m_wrt=0, branch=0, alu_op=00, jump=1,
# clk= 0,  pc= 6, rs= 2, opcode= 0, rt
m_wrt=0, branch=0, alu_op=00, jump=1,
# clk= 1,  pc= 6, rs= 0, opcode= 5, rt
m_wrt=0, branch=0, alu_op=10, jump=0,
# clk= 0,  pc= 7, rs= 0, opcode= 5, rt
m_wrt=0, branch=0, alu_op=10, jump=0,
# clk= 1,  pc= 7, rs= 0, opcode= 6, rt
, mem_read=0, mem_wrt=0, branch=0, alu
# clk= 0,  pc= 8, rs= 0, opcode= 6, rt
, mem_read=0, mem_wrt=0, branch=0, alu
# clk= 1,  pc= 8, rs= 3, opcode= 0, rt
m_wrt=0, branch=0, alu_op=00, jump=0,
# clk= 0,  pc=13, rs= 3, opcode= 0, rt
m_wrt=0, branch=0, alu_op=00, jump=0,
# clk= 1,  pc=13, rs= 4, opcode= 8, rt
, mem_wrt=0, branch=1, alu_op=01, jump
# clk= 0,  pc=14, rs= 4, opcode= 8, rt
, mem_wrt=0, branch=1, alu_op=01, jump
# clk= 1,  pc=14, rs=15, opcode= 0, rt
d=0, mem_wrt=0, branch=0, alu_op=00, j
# clk= 0,  pc=15, rs=15, opcode= 0, rt
wrt=1, mem_read=0, mem_wrt=0, branch=0
# clk= 1,  pc=15, rs=35, opcode=23, rt
m_wrt=0, branch=0, alu_op=00, jump=0,
```

```
# clk= 0,  pc=16, rs=35, opcode=23, r
m_wrt=0, branch=0, alu_op=00, jump=0,
# clk= 1,  pc=16, rs= 8, opcode=31, r
m_wrt=0, branch=0, alu_op=00, jump=0,
# clk= 0,  pc= 9, rs= 8, opcode=31, r
m_wrt=0, branch=0, alu_op=00, jump=0,
# clk= 1,  pc= 9, rs= 5, opcode= 5, r
, mem_wrt=0, branch=0, alu_op=01, jum
# clk= 0,  pc=13, rs= 5, opcode= 5, r
, mem_wrt=0, branch=0, alu_op=01, jum
# clk= 1,  pc=13, rs= 4, opcode= 8, r
m_wrt=0, branch=1, alu_op=01, jump=0,
# clk= 0,  pc=17, rs= 4, opcode= 8, r
m_wrt=0, branch=1, alu_op=01, jump=0,
# clk= 1,  pc=17, rs= 0, opcode= 8, r
, mem_read=0, mem_wrt=0, branch=0, al
# clk= 0,  pc=18, rs= 0, opcode= 8, r
m_wrt=0, branch=0, alu_op=10, jump=0,
# clk= 1,  pc=18, rs=35, opcode= 9, r
ad=1, mem_wrt=0, branch=0, alu_op=00,
# clk= 0,  pc=19, rs=35, opcode= 9, r
ad=1, mem_wrt=0, branch=0, alu_op=00,
# clk= 1,  pc=19, rs=43, opcode= 9, r
m_wrt=1, branch=0, alu_op=00, jump=0,
# clk= 0,  pc=20, rs=43, opcode= 9, r
m_wrt=1, branch=0, alu_op=00, jump=0,
# clk= 1,  pc=20, rs=13, opcode= 8, r
eg_wrt=1, mem_read=0, mem_wrt=0, bran
# clk= 0,  pc=21, rs=13, opcode= 8, r
eg_wrt=1, mem_read=0, mem_wrt=0, bran
# clk= 1,  pc=21, rs= 2, opcode= 0, r
m_wrt=0, branch=0, alu_op=00, jump=1,
# clk= 0,  pc=23, rs= 2, opcode= 0, r
m_wrt=0, branch=0, alu_op=00, jump=1,
# clk= 1,  pc=23, rs= 0, opcode= 0, r
m_wrt=0, branch=0, alu_op=10, jump=0,
```

Last test was successful but I don't know how can I show but you can check results.

Some Notes:

- **In jal inst. instead of PC+8, I did PC+1 in Verilog.**
- **In 256KB memory my address input is 18bits and I keep each data block as 32bit by extending address.**
- **In mips 32 testbench when you do test change number of clock according to your number of instruction.**
- **In data memory and register block there was a problem about clock, to solve it I didn't use negedge or posedgde clock instead I used "*" symbol as a solution from internet.**

The homework is done step by step.