CSE 331 Computer Organization Project 3 – R-type Single cycle MIPS with Structural Verilog Report

Name: Muaz Kurt
Number: 151044062

Schematic Design for added modules:

Register can write data in positive clock pulse. If writing address is 0th register or write enable signal is 0, then there will be no write operation. Registers always can be accessible for reading. Read data 1 (Rs = instruction [25:21]) and Read data 2(Rt = instruction [20:16]) are the reading addresses. In reading mode, R[Rs] and R[Rt] will be output. There are 32 registers, Rs and Rt must be 5 bits binary inputs. Output will be 32 bits. Writing data will be 32 bits input, writing address (Rd = instruction [15:11]) will be 5 bits input.

Module definition:

Registers(R[Rs], R[Rt], Rs, Rt, Rd, Write Data, Write Enable, Clock);

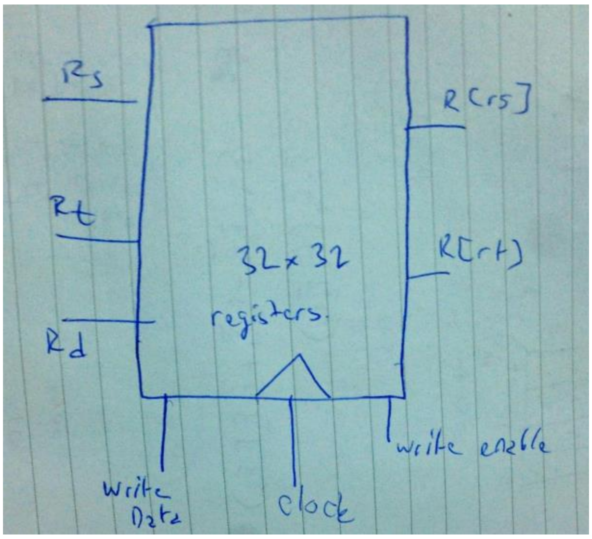


Figure 1: Register

Control Unit (module control_unit):

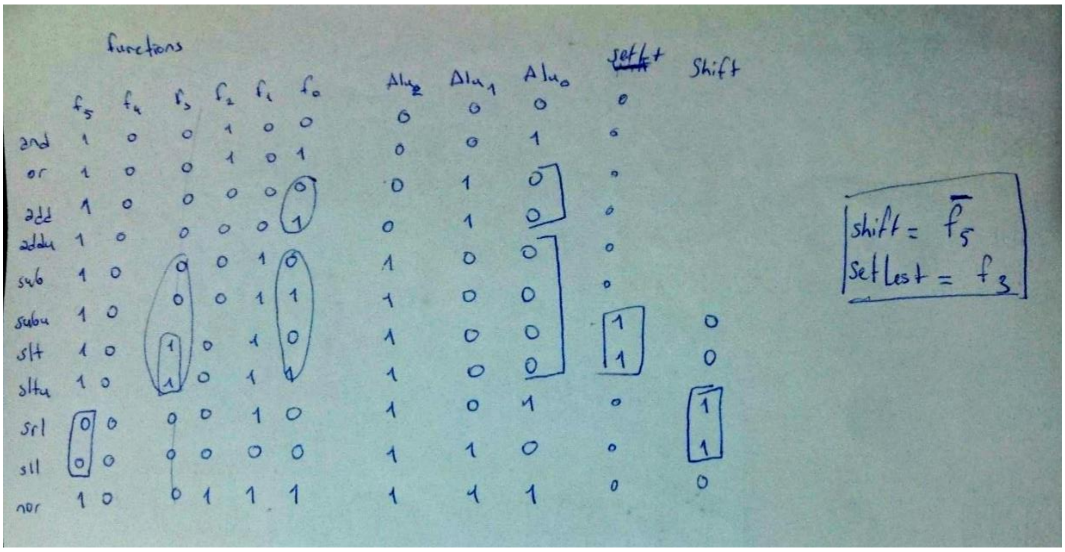
Control unit gets function code (instruction [5:0]) as input. Generates output signal as, ALU control [2:0], write enable, set less than select, shifter select.

Write enable checks the function code is in the set of known R type instruction. If so, it will be 1; otherwise, 0.

Set less than, checks the function code is set less than or set less than unsigned. If so, it will be 1; otherwise, 0.

Shifter select does the same thing as set less than for shift instruction.

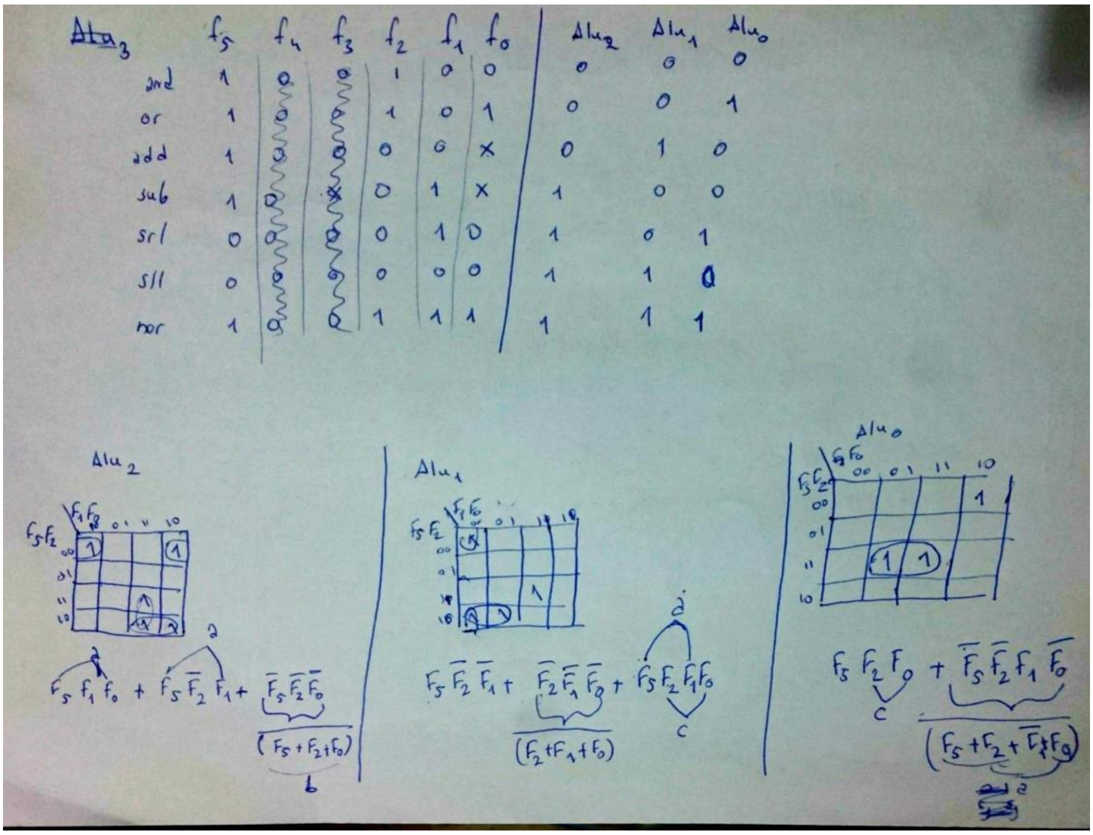
ALU control checks the function code for generating known ALU select. All 3 of ALU control bits (ALU [2], ALU [1], ALU [0]) will be generated separately. Details are in the photos below.



	functions										
	f_5	f_4	f_3	f_2	f_1	f_0	Alu_2	Alu_1	Alu_0	setLt	Shift
and	1	0	0	1	0	0	0	0	0	0	
or	1	0	0	1	0	1	0	0	1	0	
add	1	0	0	0	0	0	0	1	0	0	
addu	1	0	0	0	0	1	0	1	0	0	
sub	1	0	0	0	1	0	1	0	0	0	
subu	1	0	0	0	1	1	1	0	0	0	
slt	1	0	1	0	1	0	1	0	0	1	0
sltu	1	0	1	0	1	1	1	0	0	1	0
srl	0	0	0	0	1	0	1	0	1	0	1
sll	0	0	0	0	0	0	1	1	0	0	1
nor	1	0	0	1	1	1	1	1	1	0	0

$$shift = \overline{f_5}$$
$$SetLest = f_3$$

Figure 2: Select and Shift signals selecting



Alu_3	f_5	f_4	f_3	f_2	f_1	f_0	Alu_2	Alu_1	Alu_0
and	1	0	0	1	0	0	0	0	0
or	1	0	0	1	0	1	0	0	1
add	1	0	0	0	0	X	0	1	0
sub	1	0	X	0	1	X	1	0	0
srl	0	0	0	0	1	0	1	0	1
sll	0	0	0	0	0	0	1	1	0
nor	1	0	0	1	1	1	1	1	1

Alu_2

$F_5F_2 \backslash F_1F_0$	00	01	11	10
00	1			1
01				
11			1	
10			1	1

$$\overset{a}{F_5 F_1 F_0} + \overset{a}{F_5 \overline{F_2} F_1} + \underbrace{\overline{F_5}\,\overline{F_2}\,\overline{F_0}}_{\underbrace{\overline{(F_5+F_2+F_0)}}_{b}}$$

Alu_1

$F_5F_2 \backslash F_1F_0$	00	01	11	10
00	1			
01				
11			1	
10	1	1		

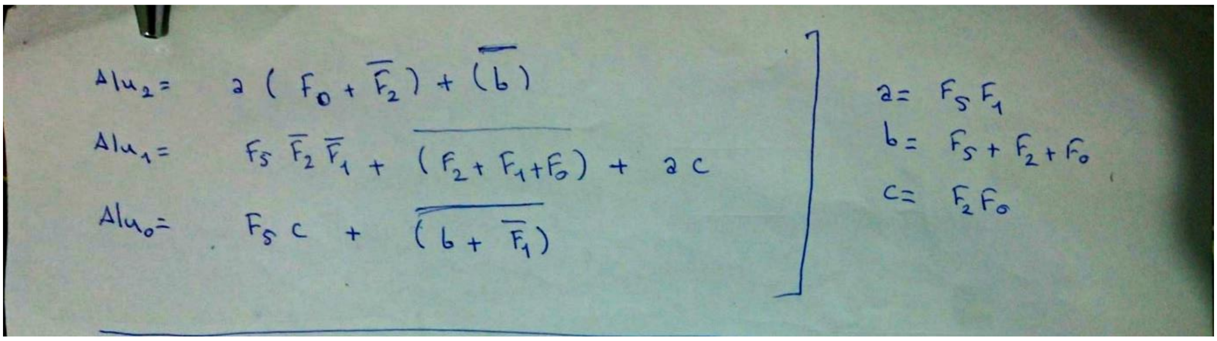
$$F_5\overline{F_2}\,\overline{F_1} + \underbrace{\overline{F_2}\,\overline{F_1}\,\overline{F_0}}_{\overline{(F_2+F_1+F_0)}} + \overset{a}{\underbrace{F_5 F_2 F_1 F_0}_{c}}$$

Alu_0

$F_5F_2 \backslash F_1F_0$	00	01	11	10
00				1
01				
11		1	1	
10				

$$\underbrace{F_5 F_2 F_0}_{c} + \underbrace{\overline{F_5}\,\overline{F_2}\,F_1\,\overline{F_0}}_{\underbrace{\overline{(F_5+F_2+\overline{F_1}+F_0)}}_{a}}$$

Figure 3: ALU Select(3bits) signal select



$$Alu_2 = a(F_0 + \bar{F}_2) + \overline{(b)}$$

$$Alu_1 = F_5 \bar{F}_2 \bar{F}_1 + \overline{(F_2 + F_1 + F_0)} + ac$$

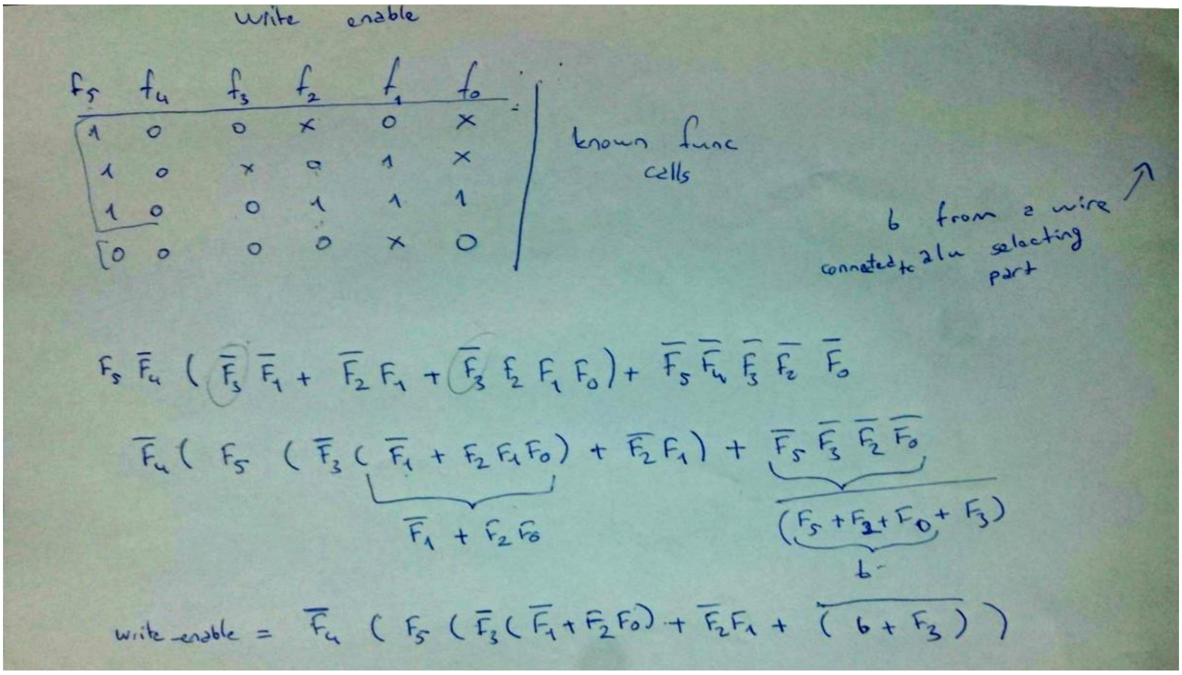
$$Alu_0 = F_5 c + \overline{(b + \bar{F}_1)}$$

$a = F_5 F_1$

$b = F_5 + F_2 + F_0$

$c = F_2 F_0$

Figure 4: ALU select results



write enable

f_5	f_4	f_3	f_2	f_1	f_0
1	0	0	x	0	x
1	0	x	0	1	x
1	0	0	1	1	1
0	0	0	0	x	0

known func cells

b from a wire connected to alu selecting part

$$F_5 \bar{F}_4 (\bar{F}_3 \bar{F}_1 + \bar{F}_2 F_1 + \bar{F}_3 F_2 F_1 F_0) + \bar{F}_5 \bar{F}_4 \bar{F}_3 \bar{F}_2 \bar{F}_0$$

$$\bar{F}_4 (F_5 (\bar{F}_3 (\underbrace{\bar{F}_1 + F_2 F_1 F_0}_{\bar{F}_1 + F_2 F_0}) + \bar{F}_2 F_1) + \underbrace{\bar{F}_5 \bar{F}_3 \bar{F}_2 \bar{F}_0}_{\underbrace{\overline{(F_5 + F_2 + F_0 + F_3)}}_{b}}$$

$$write\_enable = \bar{F}_4 (F_5 (\bar{F}_3 (\bar{F}_1 + F_2 F_0) + \bar{F}_2 F_1 + \overline{(b + F_3)}))$$

Figure 5: Write enable signal selecting

Alu32 (module alu32):

Gets an instruction (32 bits) and a clock signal (1 bit) as input, extends 5 bits shift count (instruction [10:6]) to 32 bits with adding 0s to its most significant bits (concatenate (27’b0, instruction [10:6]).

Generates 2 ALU result from 1 output. ALU output will be 32 bits operation result. For set less than operation result, result must be 0 or 1 in 32 bits. For making this, output’s most significant bit will be extended by 31 bits of 0s. Finally, there is 2 outputs.

Uses:

1. Control_unit module with giving function code (instruction [5:0]) as an input to generate necessary signals.
2. Registers module with giving Rs (instruction [25:21]), Rt (instruction [20:16]), Rd (instruction [15:10]), write enable from control unit output, write data (32 bits) from function operation output and clock which is also an input for itself as inputs to read register contents.
3. ALU module with two 32 bits variables to calculate and 3 bits ALU select signa to specify the operation to generate specific action of function code.

ALU’s two 32 bits inputs are changeable depend on the control unit’s shift select signal.

1. If the signal is 0, then ALU’s first input is R[Rs] (Rs positioned register data) and second is R[Rt] (Rt positioned register data).
2. Otherwise, ALU’s first input is R[Rt] and second is extended shift bits.

Write data is changeable depend on the control unit’s set less than signal.

1. If signal is 0, then write data will be original result of ALU.
2. Otherwise, generated ALU output will be write data (concatenate (31’b0, ALU_OUTPUT[31])).

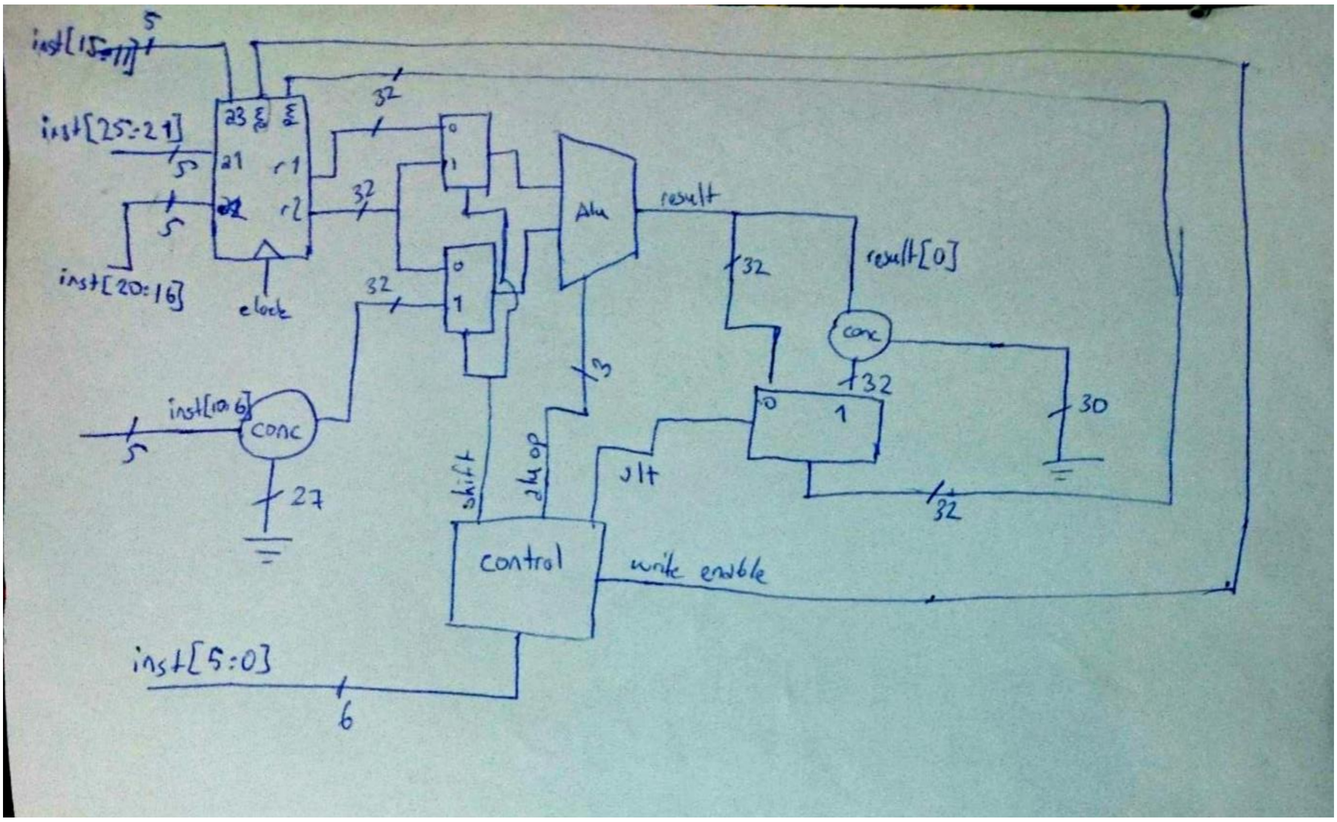
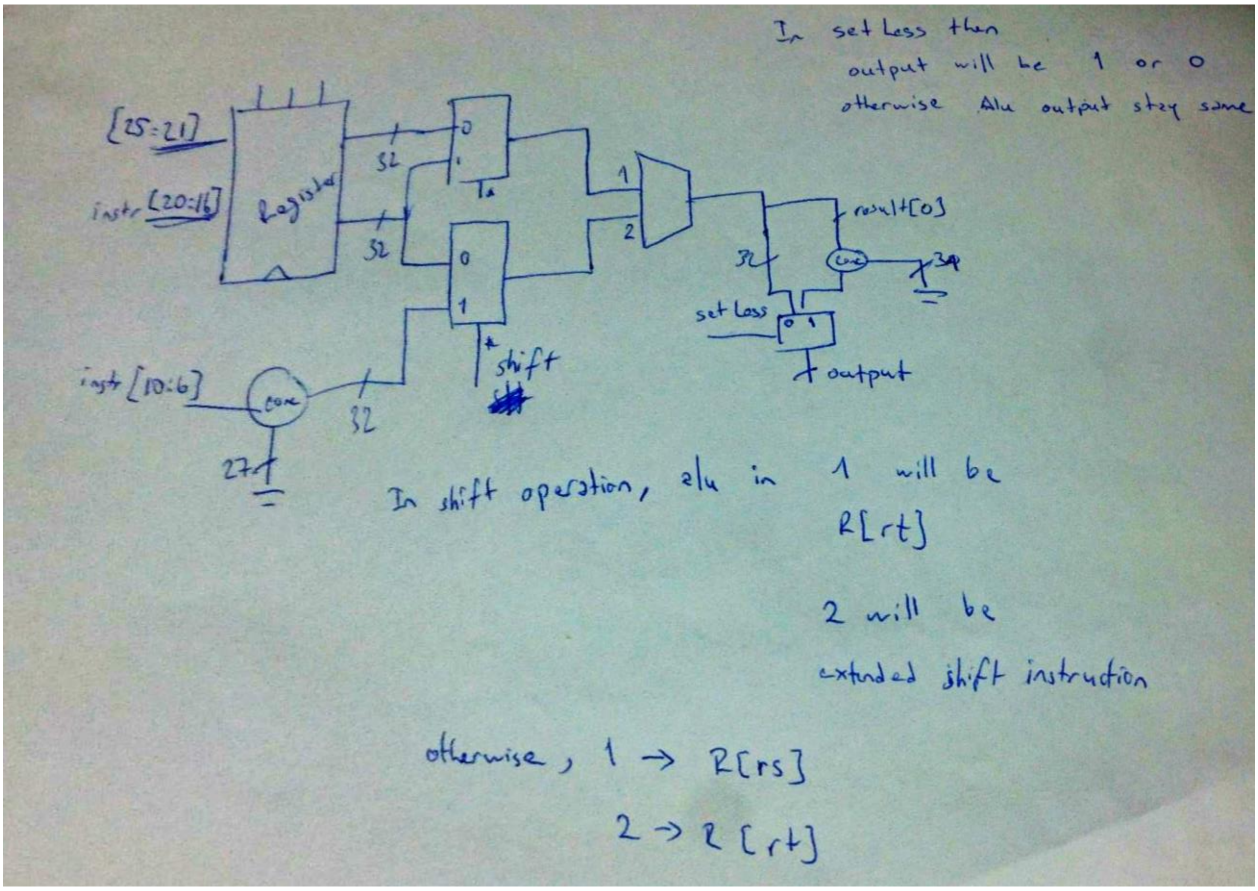


Figure 6: Datapath



In set Less then output will be 1 or 0 otherwise Alu output stay same

In shift operation, alu in 1 will be R[rt]

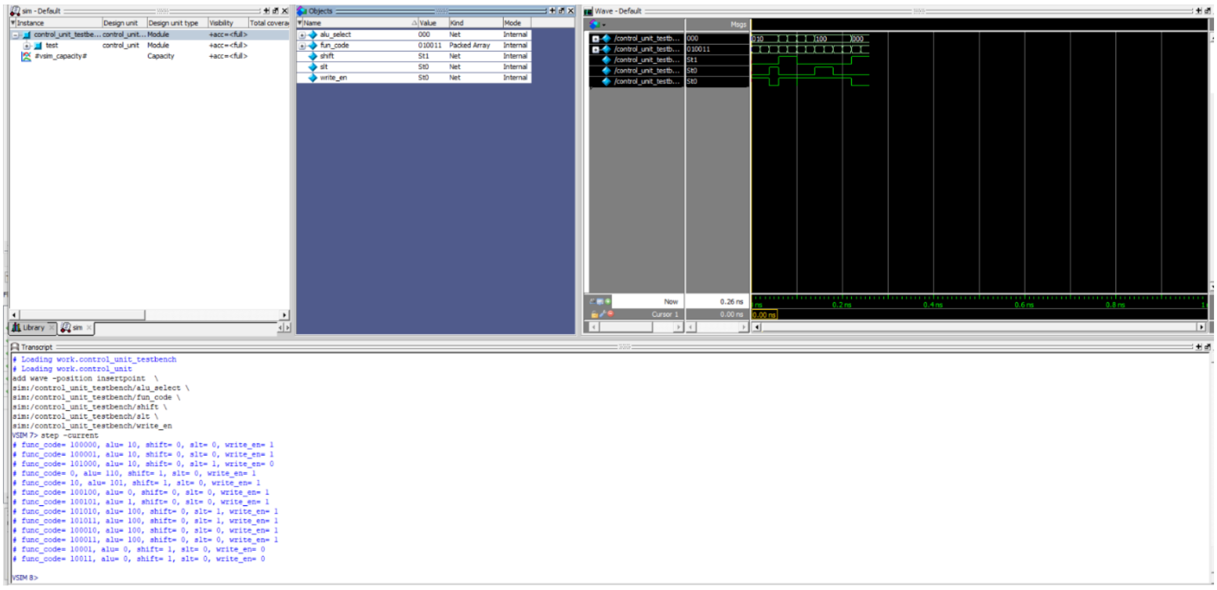
2 will be extended shift instruction

otherwise, 1 → R[rs]

2 → R[rt]

Figure 7: Datapath, signals and descriptions

Test Results



Instance	Design unit	Design unit type	Visibility	Total covera
control_unit_testbe...	control_unit...	Module	+acc=<full>	
test	control_unit	Module	+acc=<full>	
#vsim_capacity#		Capacity	+acc=<full>	

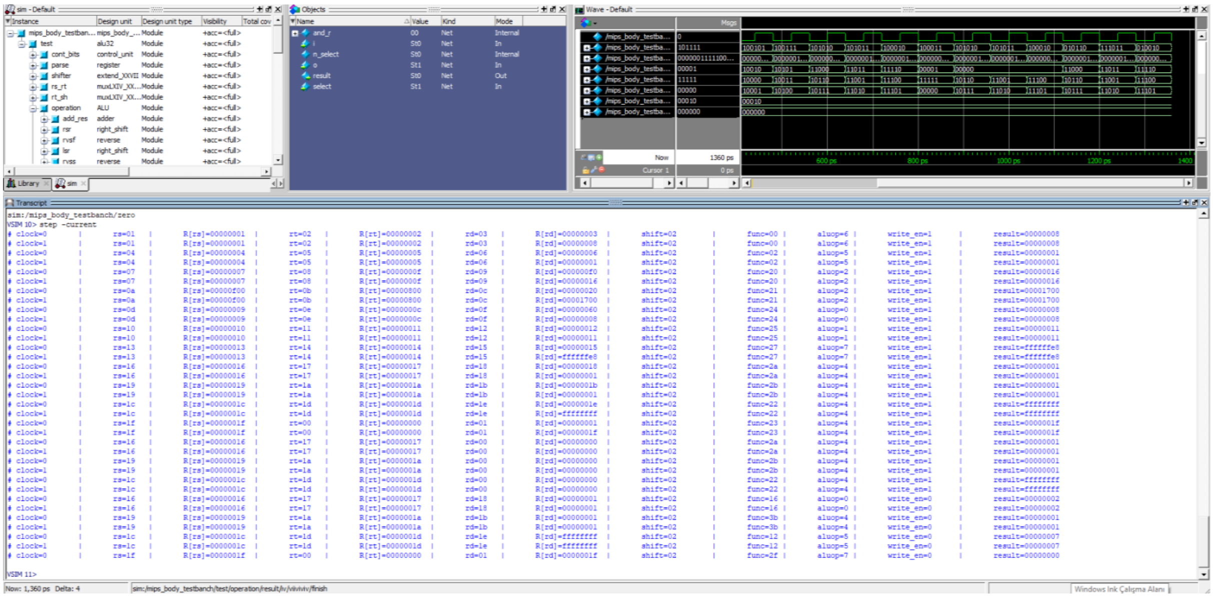
Name	Value	Kind	Mode
alu_select	000	Net	Internal
fun_code	010011	Packed Array	Internal
shift	St1	Net	Internal
slt	St0	Net	Internal
write_en	St0	Net	Internal

```
/control_unit_testb...   000
/control_unit_testb...   010011
/control_unit_testb...   St1
/control_unit_testb...   St0
/control_unit_testb...   St0
```

```
# Loading work.control_unit_testbench
# Loading work.control_unit
add wave -position insertpoint  \
sim:/control_unit_testbench/alu_select \
sim:/control_unit_testbench/fun_code \
sim:/control_unit_testbench/shift \
sim:/control_unit_testbench/slt \
sim:/control_unit_testbench/write_en
VSIM 7> step -current
# func_code= 100000, alu= 10, shift= 0, slt= 0, write_en= 1
# func_code= 100001, alu= 10, shift= 0, slt= 0, write_en= 1
# func_code= 101000, alu= 10, shift= 0, slt= 1, write_en= 0
# func_code= 0, alu= 110, shift= 1, slt= 0, write_en= 1
# func_code= 10, alu= 101, shift= 1, slt= 0, write_en= 1
# func_code= 100100, alu= 0, shift= 0, slt= 0, write_en= 1
# func_code= 100101, alu= 1, shift= 0, slt= 0, write_en= 1
# func_code= 101010, alu= 100, shift= 0, slt= 1, write_en= 1
# func_code= 101011, alu= 100, shift= 0, slt= 1, write_en= 1
# func_code= 100010, alu= 100, shift= 0, slt= 0, write_en= 1
# func_code= 100011, alu= 100, shift= 0, slt= 0, write_en= 1
# func_code= 10001, alu= 0, shift= 1, slt= 0, write_en= 0
# func_code= 10011, alu= 0, shift= 1, slt= 0, write_en= 0

VSIM 8>
```

Figure 8: Control Unit Test Results



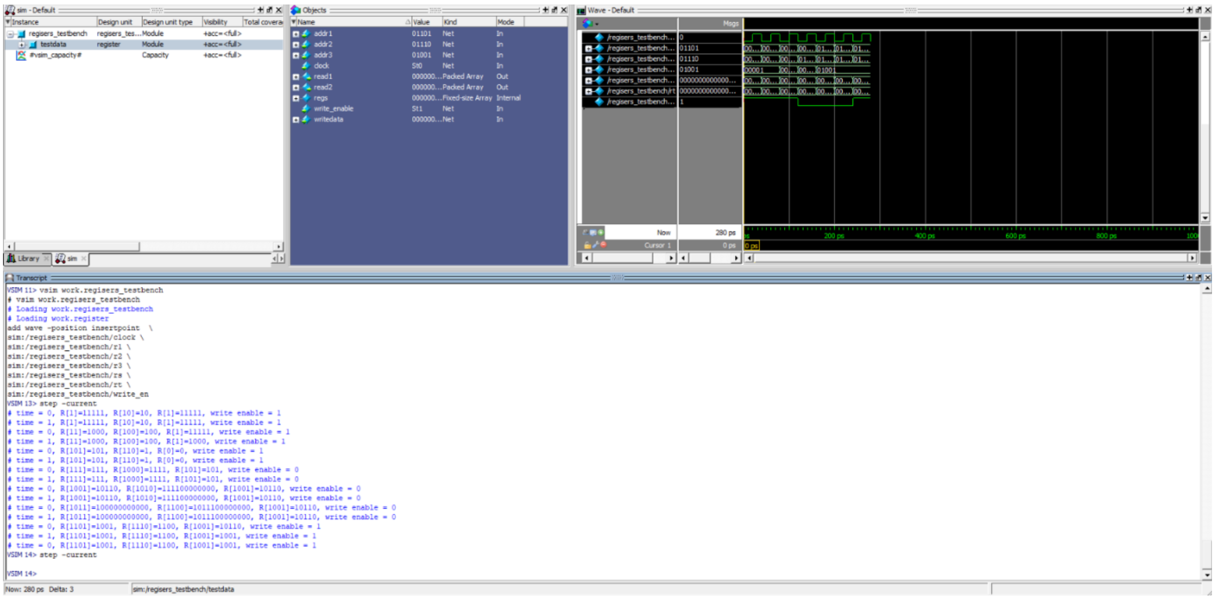
```
Instance                Design unit        Design unit type   Visibility
mips_body_testban...    mips_body_...      Module             +acc=<full>
  test                  alu32              Module             +acc=<full>
    cont_bits           control_unit       Module             +acc=<full>
    parse               register           Module             +acc=<full>
    shifter             extend_XXVII       Module             +acc=<full>
    rs_rt               muxLXIV_XX...      Module             +acc=<full>
    rt_sh               muxLXIV_XX...      Module             +acc=<full>
    operation           ALU                Module             +acc=<full>
      add_res           adder              Module             +acc=<full>
      rsr               right_shift        Module             +acc=<full>
      rvsf              reverse            Module             +acc=<full>
      lsr               right_shift        Module             +acc=<full>
      rvss              reverse            Module             +acc=<full>
```

```
Name        Value   Kind   Mode
and_r       00      Net    Internal
i           St0     Net    In
n_select    St0     Net    Internal
o           St1     Net    In
result      St0     Net    Out
select      St1     Net    In
```

```
sim:/mips_body_testbanch/zero
VSIM 10> step -current
# clock=0  |  rs=01  |  R[rs]=00000001  |  rt=02  |  R[rt]=00000002  |  rd=03  |  R[rd]=00000003  |  shift=02  |  func=00 |  aluop=6 |  write_en=1  |  result=00000008
# clock=1  |  rs=01  |  R[rs]=00000001  |  rt=02  |  R[rt]=00000002  |  rd=03  |  R[rd]=00000008  |  shift=02  |  func=00 |  aluop=6 |  write_en=1  |  result=00000008
# clock=0  |  rs=04  |  R[rs]=00000004  |  rt=05  |  R[rt]=00000005  |  rd=06  |  R[rd]=00000006  |  shift=02  |  func=02 |  aluop=5 |  write_en=1  |  result=00000001
# clock=1  |  rs=04  |  R[rs]=00000004  |  rt=05  |  R[rt]=00000005  |  rd=06  |  R[rd]=00000001  |  shift=02  |  func=02 |  aluop=5 |  write_en=1  |  result=00000001
# clock=0  |  rs=07  |  R[rs]=00000007  |  rt=08  |  R[rt]=0000000f  |  rd=09  |  R[rd]=000000f0  |  shift=02  |  func=20 |  aluop=2 |  write_en=1  |  result=00000016
# clock=1  |  rs=07  |  R[rs]=00000007  |  rt=08  |  R[rt]=0000000f  |  rd=09  |  R[rd]=00000016  |  shift=02  |  func=20 |  aluop=2 |  write_en=1  |  result=00000016
# clock=0  |  rs=0a  |  R[rs]=00000f00  |  rt=0b  |  R[rt]=00000800  |  rd=0c  |  R[rd]=00000020  |  shift=02  |  func=21 |  aluop=2 |  write_en=1  |  result=00001700
# clock=1  |  rs=0a  |  R[rs]=00000f00  |  rt=0b  |  R[rt]=00000800  |  rd=0c  |  R[rd]=00001700  |  shift=02  |  func=21 |  aluop=2 |  write_en=1  |  result=00001700
# clock=0  |  rs=0d  |  R[rs]=00000009  |  rt=0e  |  R[rt]=0000000c  |  rd=0f  |  R[rd]=00000060  |  shift=02  |  func=24 |  aluop=0 |  write_en=1  |  result=00000008
# clock=1  |  rs=0d  |  R[rs]=00000009  |  rt=0e  |  R[rt]=0000000c  |  rd=0f  |  R[rd]=00000008  |  shift=02  |  func=24 |  aluop=0 |  write_en=1  |  result=00000008
# clock=0  |  rs=10  |  R[rs]=00000010  |  rt=11  |  R[rt]=00000011  |  rd=12  |  R[rd]=00000012  |  shift=02  |  func=25 |  aluop=1 |  write_en=1  |  result=00000011
# clock=1  |  rs=10  |  R[rs]=00000010  |  rt=11  |  R[rt]=00000011  |  rd=12  |  R[rd]=00000011  |  shift=02  |  func=25 |  aluop=1 |  write_en=1  |  result=00000011
# clock=0  |  rs=13  |  R[rs]=00000013  |  rt=14  |  R[rt]=00000014  |  rd=15  |  R[rd]=00000015  |  shift=02  |  func=27 |  aluop=7 |  write_en=1  |  result=ffffffe8
# clock=1  |  rs=13  |  R[rs]=00000013  |  rt=14  |  R[rt]=00000014  |  rd=15  |  R[rd]=ffffffe8  |  shift=02  |  func=27 |  aluop=7 |  write_en=1  |  result=ffffffe8
# clock=0  |  rs=16  |  R[rs]=00000016  |  rt=17  |  R[rt]=00000017  |  rd=18  |  R[rd]=00000018  |  shift=02  |  func=2a |  aluop=4 |  write_en=1  |  result=00000001
# clock=1  |  rs=16  |  R[rs]=00000016  |  rt=17  |  R[rt]=00000017  |  rd=18  |  R[rd]=00000001  |  shift=02  |  func=2a |  aluop=4 |  write_en=1  |  result=00000001
# clock=0  |  rs=19  |  R[rs]=00000019  |  rt=1a  |  R[rt]=0000001a  |  rd=1b  |  R[rd]=0000001b  |  shift=02  |  func=2b |  aluop=4 |  write_en=1  |  result=00000001
# clock=1  |  rs=19  |  R[rs]=00000019  |  rt=1a  |  R[rt]=0000001a  |  rd=1b  |  R[rd]=00000001  |  shift=02  |  func=2b |  aluop=4 |  write_en=1  |  result=00000001
# clock=0  |  rs=1c  |  R[rs]=0000001c  |  rt=1d  |  R[rt]=0000001d  |  rd=1e  |  R[rd]=0000001e  |  shift=02  |  func=22 |  aluop=4 |  write_en=1  |  result=ffffffff
# clock=1  |  rs=1c  |  R[rs]=0000001c  |  rt=1d  |  R[rt]=0000001d  |  rd=1e  |  R[rd]=ffffffff  |  shift=02  |  func=22 |  aluop=4 |  write_en=1  |  result=ffffffff
# clock=0  |  rs=1f  |  R[rs]=0000001f  |  rt=00  |  R[rt]=00000000  |  rd=01  |  R[rd]=00000001  |  shift=02  |  func=23 |  aluop=4 |  write_en=1  |  result=0000001f
# clock=1  |  rs=1f  |  R[rs]=0000001f  |  rt=00  |  R[rt]=00000000  |  rd=01  |  R[rd]=0000001f  |  shift=02  |  func=23 |  aluop=4 |  write_en=1  |  result=0000001f
# clock=0  |  rs=16  |  R[rs]=00000016  |  rt=17  |  R[rt]=00000017  |  rd=00  |  R[rd]=00000000  |  shift=02  |  func=2a |  aluop=4 |  write_en=1  |  result=00000001
# clock=1  |  rs=16  |  R[rs]=00000016  |  rt=17  |  R[rt]=00000017  |  rd=00  |  R[rd]=00000000  |  shift=02  |  func=2a |  aluop=4 |  write_en=1  |  result=00000001
# clock=0  |  rs=19  |  R[rs]=00000019  |  rt=1a  |  R[rt]=0000001a  |  rd=00  |  R[rd]=00000000  |  shift=02  |  func=2b |  aluop=4 |  write_en=1  |  result=00000001
# clock=1  |  rs=19  |  R[rs]=00000019  |  rt=1a  |  R[rt]=0000001a  |  rd=00  |  R[rd]=00000000  |  shift=02  |  func=2b |  aluop=4 |  write_en=1  |  result=00000001
# clock=0  |  rs=1c  |  R[rs]=0000001c  |  rt=1d  |  R[rt]=0000001d  |  rd=00  |  R[rd]=00000000  |  shift=02  |  func=22 |  aluop=4 |  write_en=1  |  result=ffffffff
# clock=1  |  rs=1c  |  R[rs]=0000001c  |  rt=1d  |  R[rt]=0000001d  |  rd=00  |  R[rd]=00000000  |  shift=02  |  func=22 |  aluop=4 |  write_en=1  |  result=ffffffff
# clock=0  |  rs=16  |  R[rs]=00000016  |  rt=17  |  R[rt]=00000017  |  rd=18  |  R[rd]=00000001  |  shift=02  |  func=16 |  aluop=0 |  write_en=0  |  result=00000002
# clock=1  |  rs=16  |  R[rs]=00000016  |  rt=17  |  R[rt]=00000017  |  rd=18  |  R[rd]=00000001  |  shift=02  |  func=16 |  aluop=0 |  write_en=0  |  result=00000002
# clock=0  |  rs=19  |  R[rs]=00000019  |  rt=1a  |  R[rt]=0000001a  |  rd=1b  |  R[rd]=00000001  |  shift=02  |  func=3b |  aluop=4 |  write_en=0  |  result=00000001
# clock=1  |  rs=19  |  R[rs]=00000019  |  rt=1a  |  R[rt]=0000001a  |  rd=1b  |  R[rd]=00000001  |  shift=02  |  func=3b |  aluop=4 |  write_en=0  |  result=00000001
# clock=0  |  rs=1c  |  R[rs]=0000001c  |  rt=1d  |  R[rt]=0000001d  |  rd=1e  |  R[rd]=ffffffff  |  shift=02  |  func=12 |  aluop=5 |  write_en=0  |  result=00000007
# clock=1  |  rs=1c  |  R[rs]=0000001c  |  rt=1d  |  R[rt]=0000001d  |  rd=1e  |  R[rd]=ffffffff  |  shift=02  |  func=12 |  aluop=5 |  write_en=0  |  result=00000007
# clock=0  |  rs=1f  |  R[rs]=0000001f  |  rt=00  |  R[rt]=00000000  |  rd=01  |  R[rd]=0000001f  |  shift=02  |  func=2f |  aluop=7 |  write_en=0  |  result=00000000

VSIM 11>
Now: 1,360 ps  Delta: 4    sim:/mips_body_testbanch/test/operation/result/iv/viiviviv/finish
```

Figure 9: alu32 Test Results



```
Instance               Design unit       Design unit type   Visibility
regisers_testbench     regisers_tes...   Module             +acc=<full>
  testdata             register          Module             +acc=<full>
#vsim_capacity#                          Capacity           +acc=<full>
```

```
Name           Value       Kind               Mode
addr1          01101       Net                In
addr2          01110       Net                In
addr3          01001       Net                In
clock          St0         Net                In
read1          000000...   Packed Array       Out
read2          000000...   Packed Array       Out
regs           000000...   Fixed-size Array   Internal
write_enable   St1         Net                In
writedata      000000...   Net                In
```

```
VSIM 11> vsim work.regisers_testbench
# vsim work.regisers_testbench
# Loading work.regisers_testbench
# Loading work.register
add wave -position insertpoint  \
sim:/regisers_testbench/clock \
sim:/regisers_testbench/r1 \
sim:/regisers_testbench/r2 \
sim:/regisers_testbench/r3 \
sim:/regisers_testbench/rs \
sim:/regisers_testbench/rt \
sim:/regisers_testbench/write_en
VSIM 13> step -current
# time = 0, R[1]=11111, R[10]=10, R[1]=11111, write enable = 1
# time = 1, R[1]=11111, R[10]=10, R[1]=11111, write enable = 1
# time = 0, R[11]=1000, R[100]=100, R[1]=11111, write enable = 1
# time = 1, R[11]=1000, R[100]=100, R[1]=1000, write enable = 1
# time = 0, R[101]=101, R[110]=1, R[0]=0, write enable = 1
# time = 1, R[101]=101, R[110]=1, R[0]=0, write enable = 1
# time = 0, R[111]=111, R[1000]=1111, R[101]=101, write enable = 0
# time = 1, R[111]=111, R[1000]=1111, R[101]=101, write enable = 0
# time = 0, R[1001]=10110, R[1010]=111100000000, R[1001]=10110, write enable = 0
# time = 1, R[1001]=10110, R[1010]=111100000000, R[1001]=10110, write enable = 0
# time = 0, R[1011]=100000000000, R[1100]=1011100000000, R[1001]=10110, write enable = 0
# time = 1, R[1011]=100000000000, R[1100]=1011100000000, R[1001]=10110, write enable = 0
# time = 0, R[1101]=1001, R[1110]=1100, R[1001]=10110, write enable = 1
# time = 1, R[1101]=1001, R[1110]=1100, R[1001]=1001, write enable = 1
# time = 0, R[1101]=1001, R[1110]=1100, R[1001]=1001, write enable = 1
VSIM 14> step -current

VSIM 14>
Now: 280 ps  Delta: 3    sim:/regisers_testbench/testdata
```

Figure 10: Registers Test Result

NOT:

There is no check for opcode, because only R type instructions are supported. If opcode must be checked, then this is the 'partial working' issue of the project.

Thanks for reading 😊