Information Technology Institute

Ministry of Communications and Information Technology

egypt.makes.electronics

Digital IC Design

Mohamed Elsayed Ali Ebrahim Saad

Single-Cycle RISC-V Processor

RISC-V®

Contents

1. Overview

❖ Instruction Formats

7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	
funct7	rs2	rs1	funct3	rd	op	**R-Type**
$imm_{11:0}$		rs1	funct3	rd	op	**I-Type**
$imm_{11:5}$	rs2	rs1	funct3	$imm_{4:0}$	op	**S-Type**
$imm_{12,10:5}$	rs2	rs1	funct3	$imm_{4:1,11}$	op	**B-Type**
$imm_{31:12}$				rd	op	**U-Type**
$imm_{20,10:1,11,19:12}$				rd	op	**J-Type**
20 bits				5 bits	7 bits	

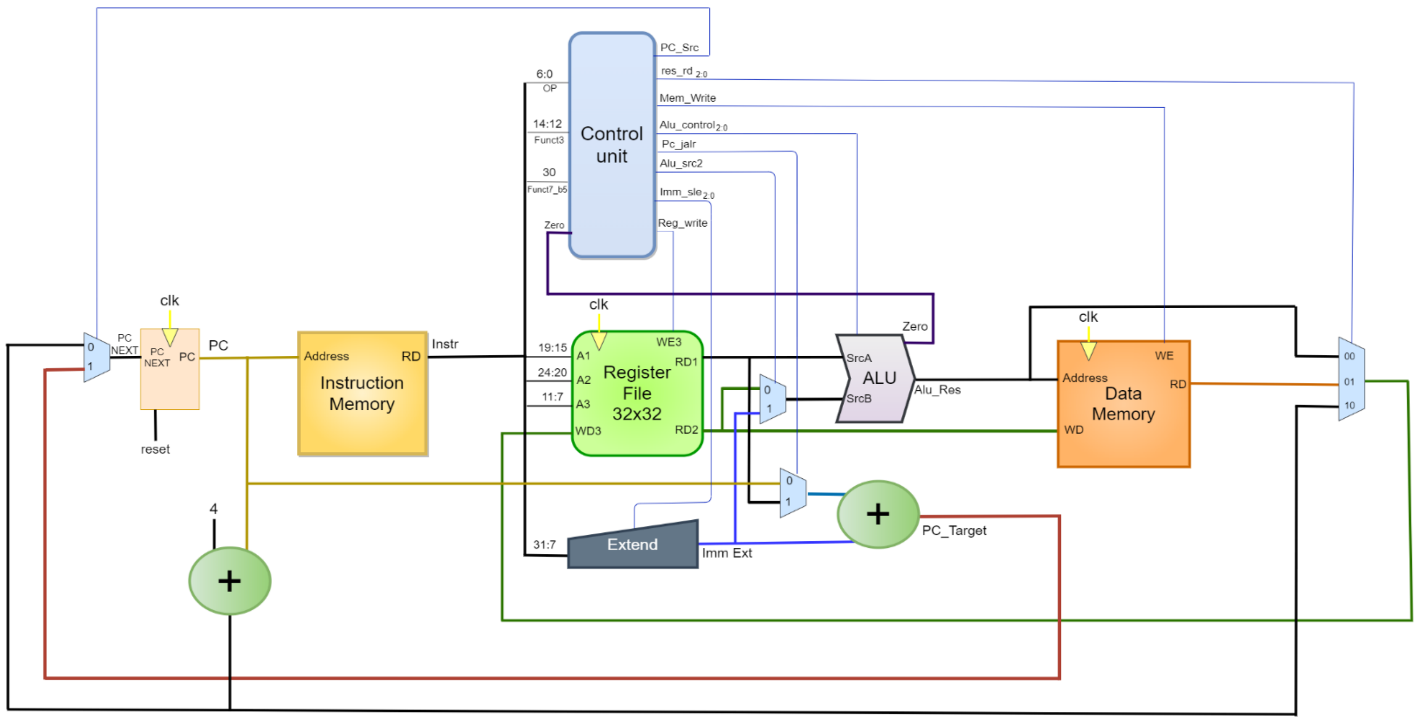
❖ Instruction Set

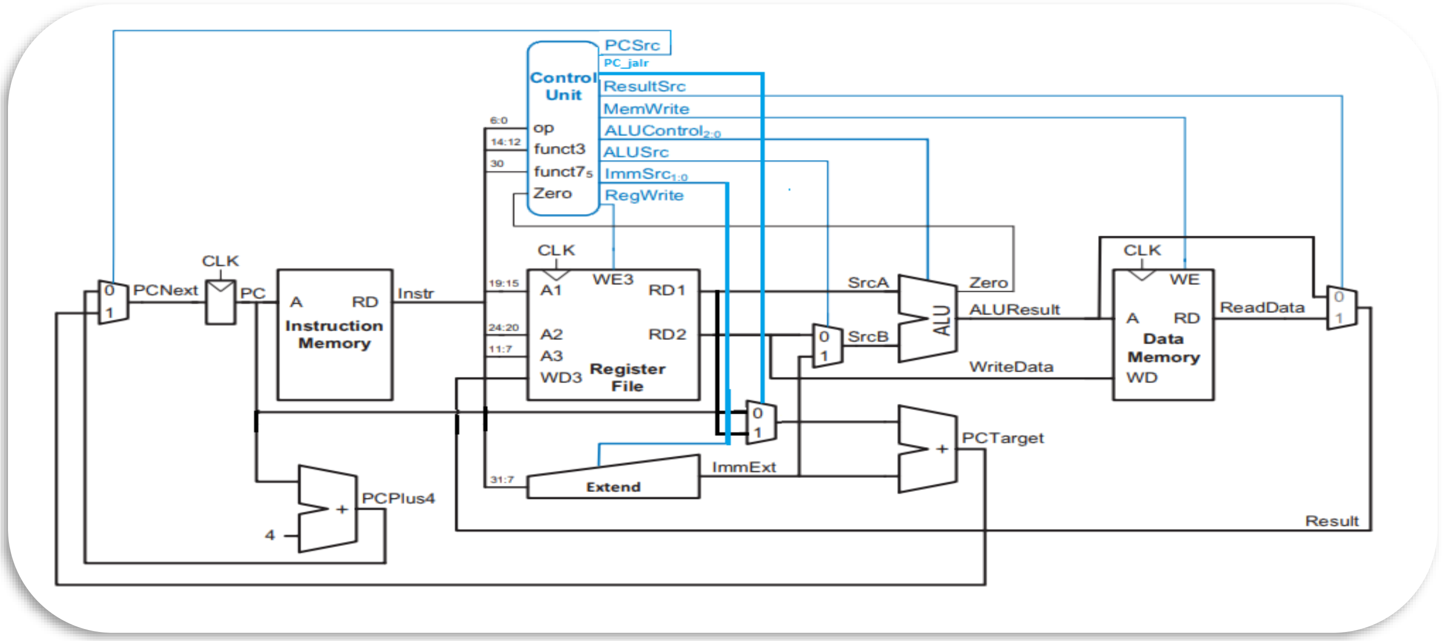
op	funct3	funct7	Type	Instruction	Description	Operation
0000011 (3)	010	–	I	`lw rd, imm(rs1)`	load word	rd = [Address]$_{31:0}$
0010011 (19)	000	–	I	`addi rd, rs1, imm`	add immediate	rd = rs1 + SignExt(imm)
0010011 (19)	001	0000000*	I	`slli rd, rs1, uimm`	shift left logical immediate	rd = rs1 << uimm
0010011 (19)	010	–	I	`slti rd, rs1, imm`	set less than immediate	rd = (rs1 < SignExt(imm))
0010011 (19)	100	–	I	`xori rd, rs1, imm`	xor immediate	rd = rs1 ^ SignExt(imm)
0010011 (19)	101	0000000*	I	`srli rd, rs1, uimm`	shift right logical immediate	rd = rs1 >> uimm
0010011 (19)	110	–	I	`ori rd, rs1, imm`	or immediate	rd = rs1 | SignExt(imm)
0010011 (19)	111	–	I	`andi rd, rs1, imm`	and immediate	rd = rs1 & SignExt(imm)
0100011 (35)	010	–	S	`sw rs2, imm(rs1)`	store word	[Address]$_{31:0}$ = rs2
0110011 (51)	000	0000000	R	`add rd, rs1, rs2`	add	rd = rs1 + rs2
0110011 (51)	000	0100000	R	`sub rd, rs1, rs2`	sub	rd = rs1 – rs2
0110011 (51)	001	0000000	R	`sll rd, rs1, rs2`	shift left logical	rd = rs1 << rs2$_{4:0}$
0110011 (51)	010	0000000	R	`slt rd, rs1, rs2`	set less than	rd = (rs1 < rs2)
0110011 (51)	100	0000000	R	`xor rd, rs1, rs2`	xor	rd = rs1 ^ rs2
0110011 (51)	101	0000000	R	`srl rd, rs1, rs2`	shift right logical	rd = rs1 >> rs2$_{4:0}$
0110011 (51)	110	0000000	R	`or rd, rs1, rs2`	or	rd = rs1 | rs2
0110011 (51)	111	0000000	R	`and rd, rs1, rs2`	and	rd = rs1 & rs2
1100011 (99)	000	–	B	`beq rs1, rs2, label`	branch if =	if (rs1 == rs2) PC = BTA
1100011 (99)	001	–	B	`bne rs1, rs2, label`	branch if ≠	if (rs1 ≠ rs2) PC = BTA
1100111 (103)	000	–	I	`jalr rd, rs1, imm`	jump and link register	PC = rs1 + SignExt(imm), rd = PC + 4
1101111 (111)	–	–	J	`jal rd, label`	jump and link	PC = JTA, rd = PC + 4

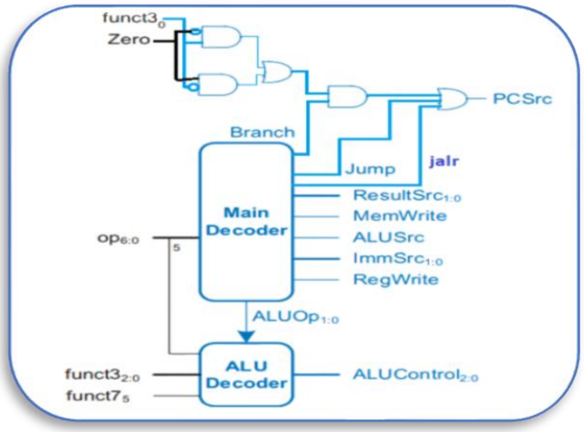
❖ Immediate Encoding Style

ImmSrc	ImmExt	Type	Description
00	{{20{*Instr*[31]}}, *Instr*[31:20]}	I	12-bit signed immediate
01	{{20{*Instr*[31]}}, *Instr*[31:25], *Instr*[11:7]}	S	12-bit signed immediate
10	{{20{*Instr*[31]}}, *Instr*[7], *Instr*[30:25], *Instr*[11:8], 1'b0}	B	13-bit signed immediate
11	{{12{*Instr*[31]}}, *Instr*[19:12], *Instr*[20], *Instr*[30:21], 1'b0}	J	21-bit signed immediate

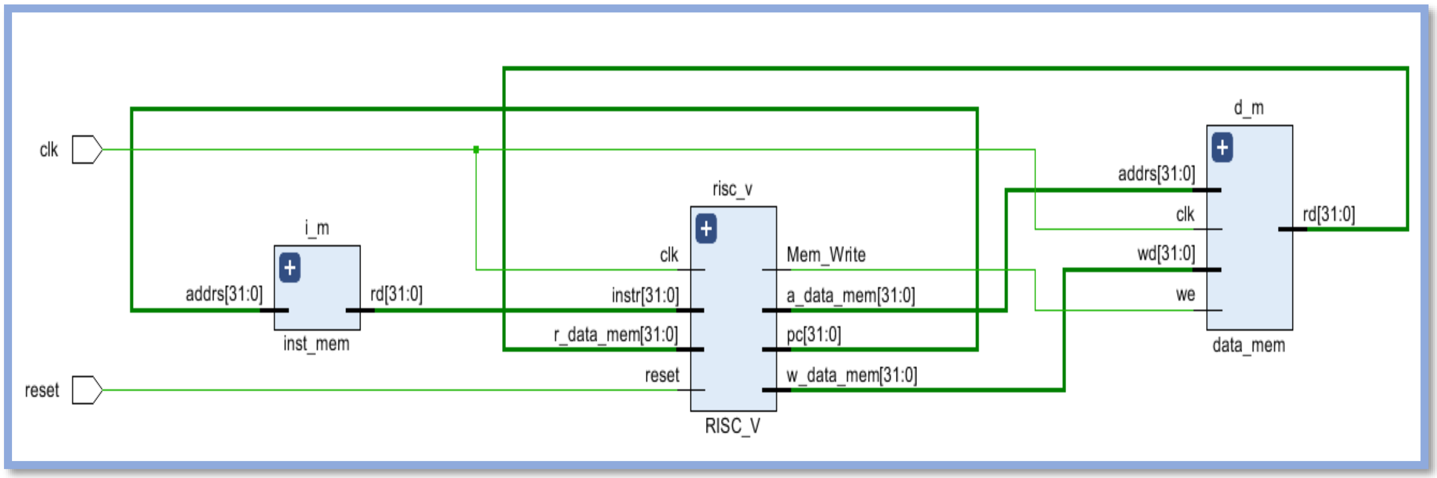
❖ Block Diagram

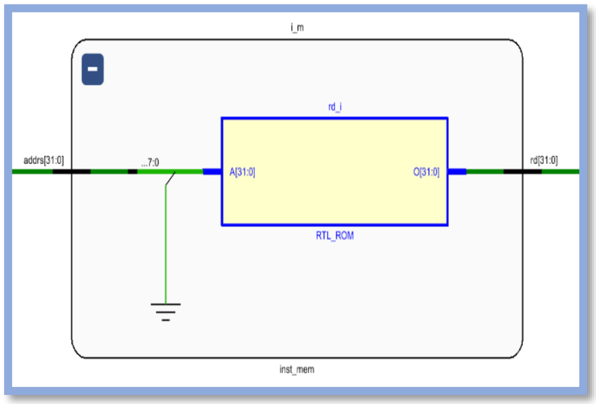


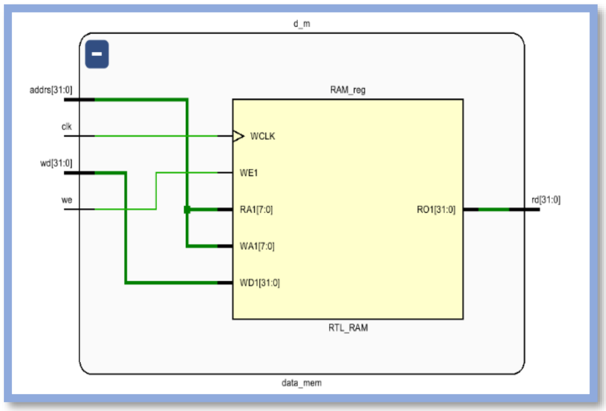


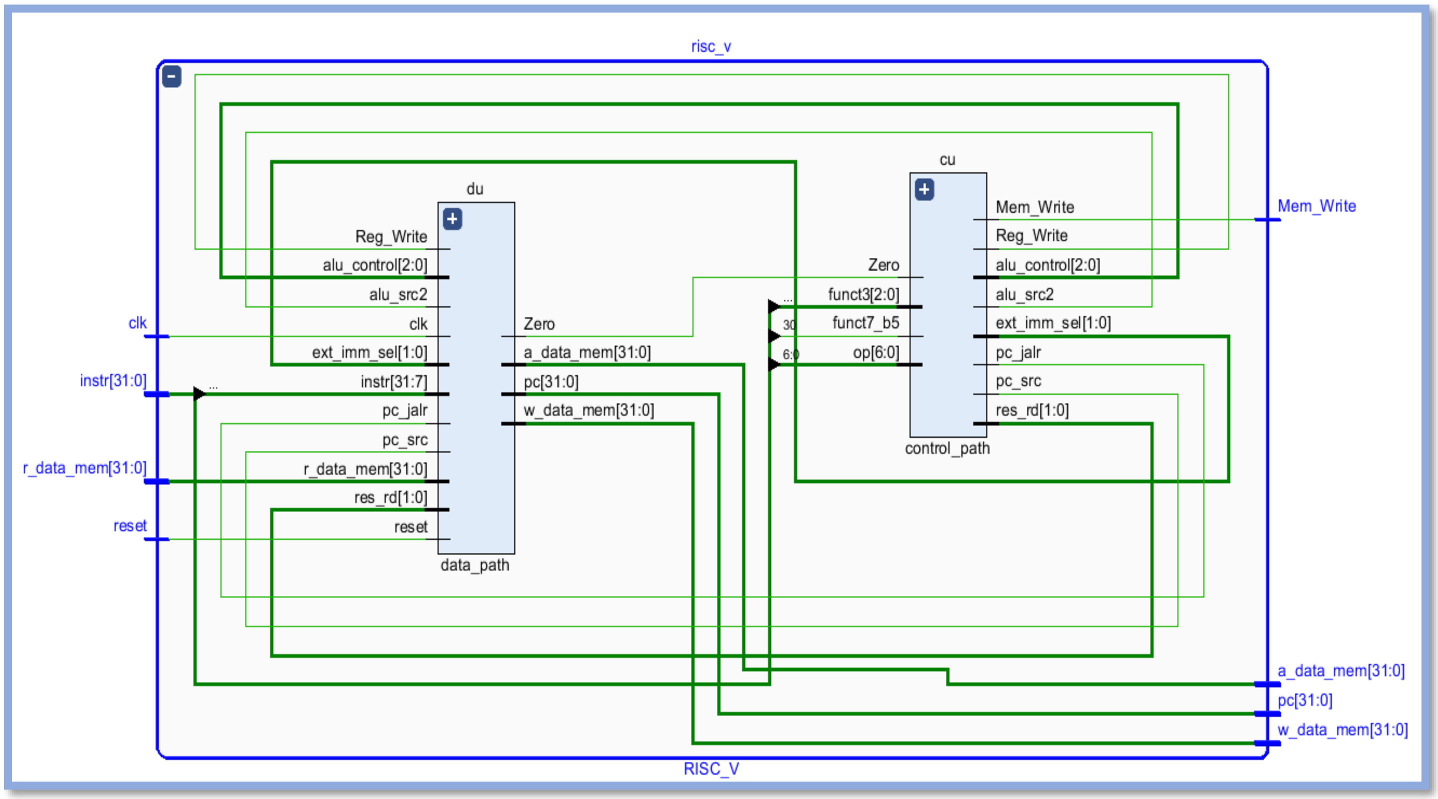


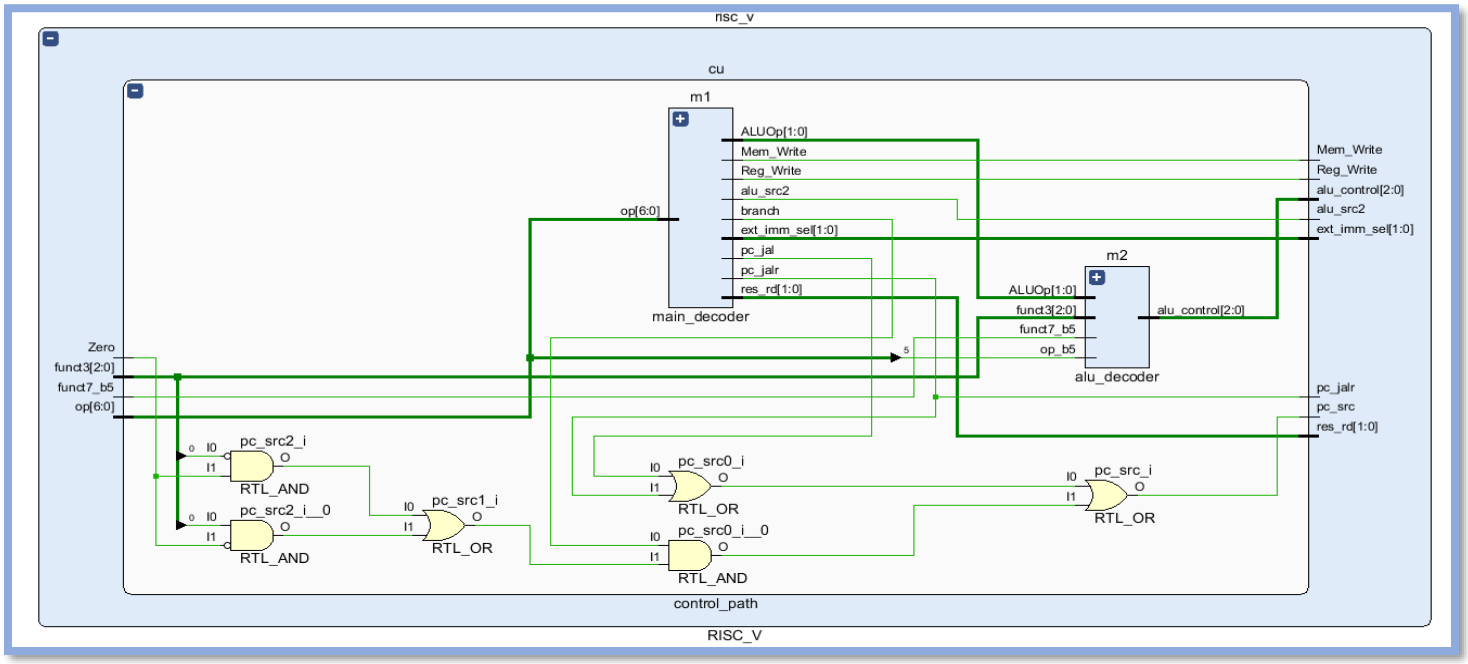
2. RTL Block Diagram

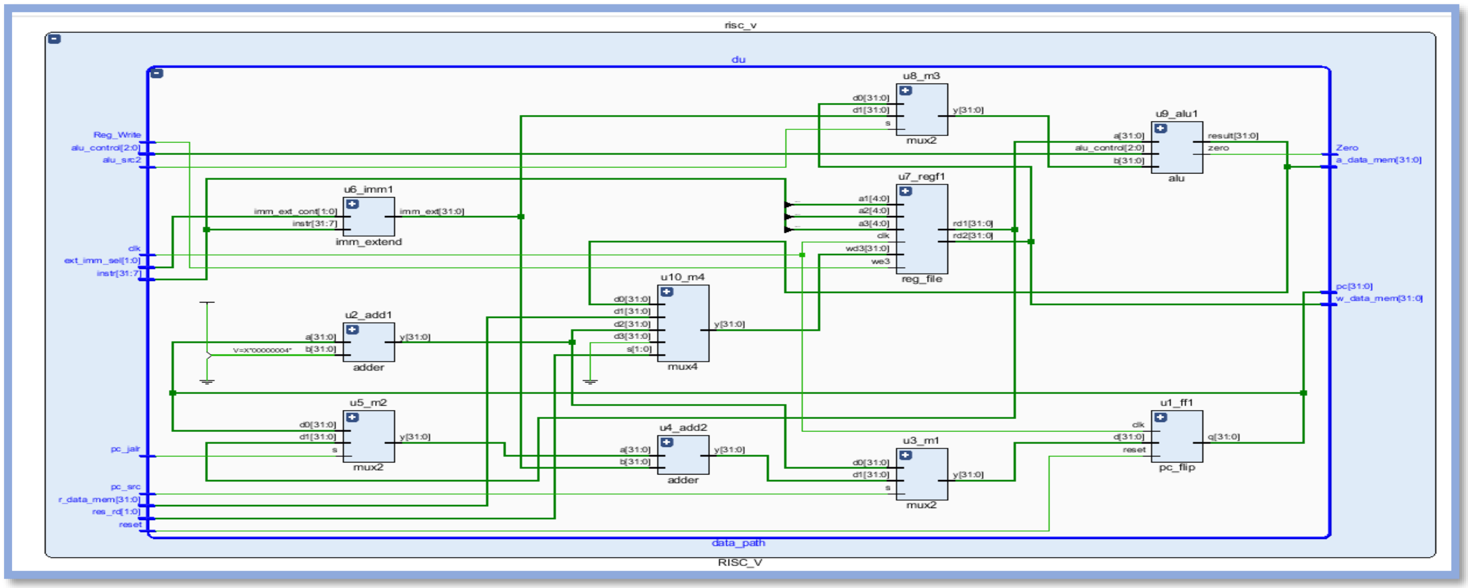


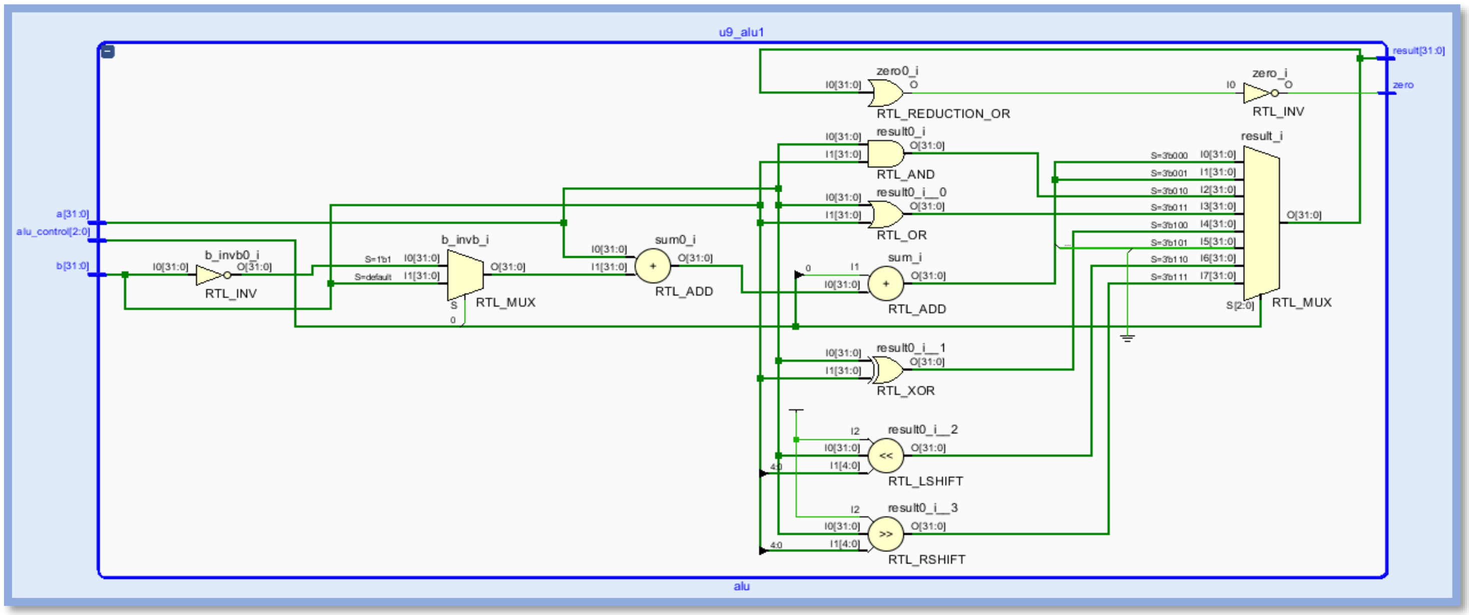






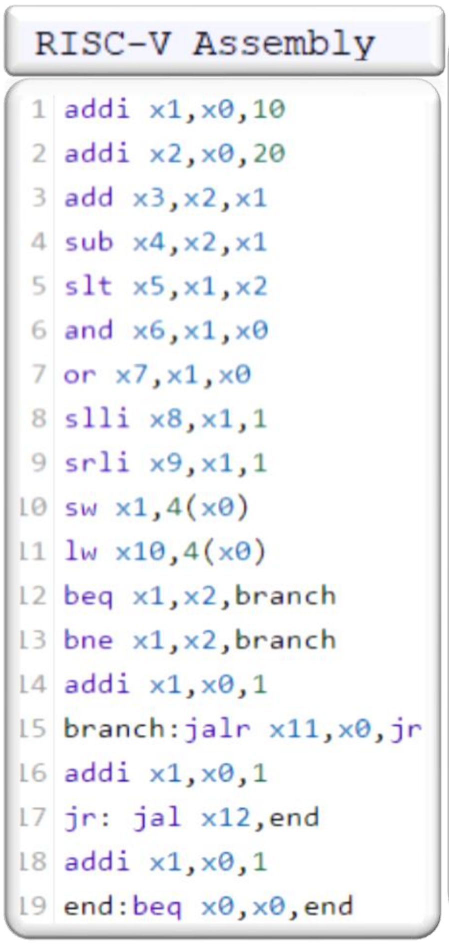

risc_v
cu
m1
ALUOp[1:0]
Mem_Write
Reg_Write
alu_src2
branch
ext_imm_sel[1:0]
pc_jal
pc_jalr
res_rd[1:0]
op[6:0]
main_decoder
m2
ALUOp[1:0]
funct3[2:0]
funct7_b5
op_b5
alu_control[2:0]
alu_decoder
Zero
funct3[2:0]
funct7_b5
op[6:0]
Mem_Write
Reg_Write
alu_control[2:0]
alu_src2
ext_imm_sel[1:0]
pc_jalr
pc_src
res_rd[1:0]
pc_src2_i
RTL_AND
pc_src2_i__0
RTL_AND
pc_src1_i
RTL_OR
pc_src0_i
RTL_OR
pc_src0_i__0
RTL_AND
pc_src_i
RTL_OR
control_path
RISC_V


risc_v
du
u8_m3
mux2
u9_alu1
alu
u6_imm1
imm_extend
u7_regf1
reg_file
u10_m4
mux4
u2_add1
adder
u5_m2
mux2
u4_add2
adder
u3_m1
mux2
u1_ff1
pc_flip
Reg_Write
alu_control[2:0]
alu_src2
clk
ext_imm_sel[1:0]
instr[31:7]
pc_jalr
pc_src
r_data_mem[31:0]
res_rd[1:0]
reset
Zero
a_data_mem[31:0]
pc[31:0]
w_data_mem[31:0]
data_path
RISC_V

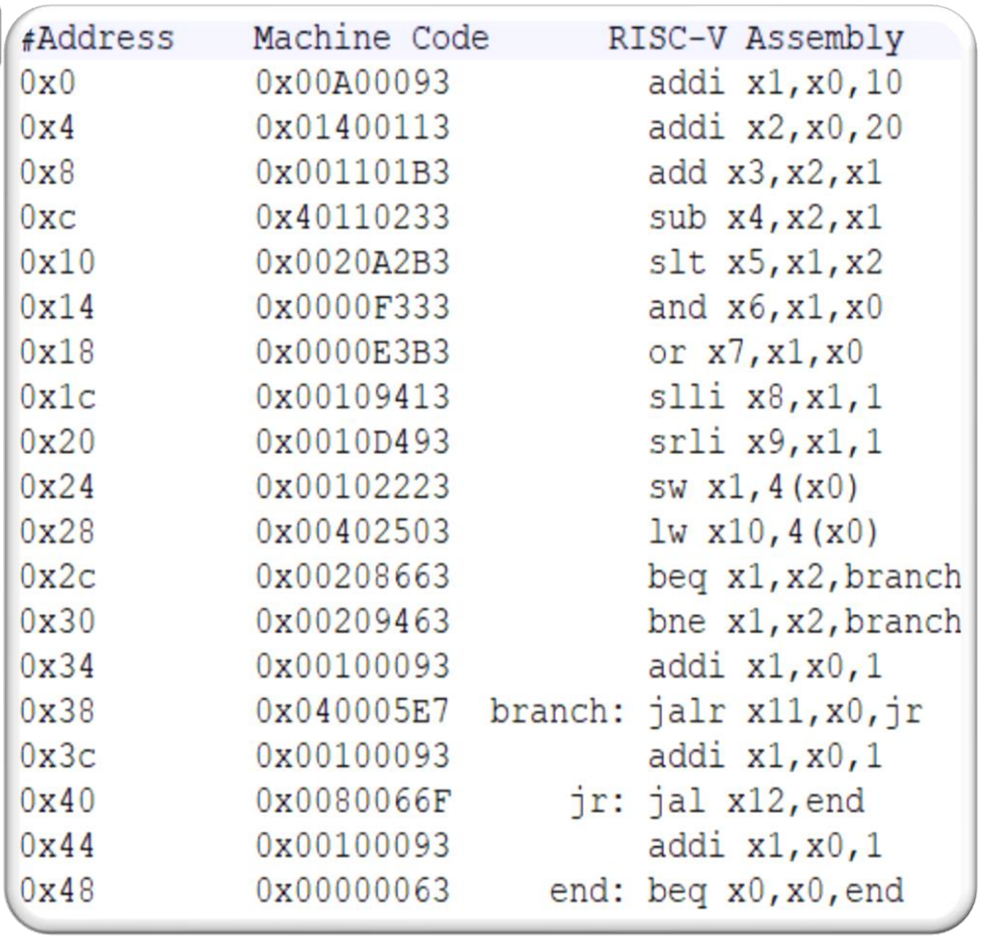

u9_alu1
result[31:0]
zero
a[31:0]
alu_control[2:0]
b[31:0]
zero0_i
RTL_REDUCTION_OR
zero_i
RTL_INV
result0_i
RTL_AND
result0_i__0
RTL_OR
result_i
RTL_MUX
b_invb0_i
RTL_INV
b_invb_i
RTL_MUX
sum0_i
RTL_ADD
sum_i
RTL_ADD
result0_i__1
RTL_XOR
result0_i__2
RTL_LSHIFT
result0_i__3
RTL_RSHIFT
alu

3. Test Bench Results:

❖ First Test Case Assembly code and its output

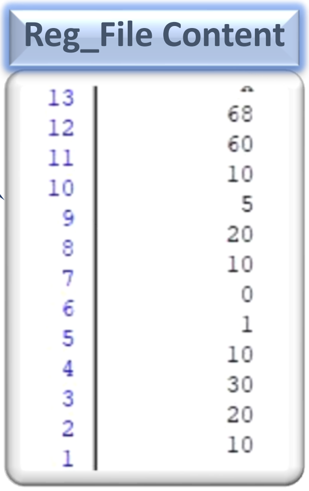


```
RISC-V Assembly
addi x1,x0,10
addi x2,x0,20
add x3,x2,x1
sub x4,x2,x1
slt x5,x1,x2
and x6,x1,x0
or x7,x1,x0
slli x8,x1,1
srli x9,x1,1
sw x1,4(x0)
lw x10,4(x0)
beq x1,x2,branch
bne x1,x2,branch
addi x1,x0,1
branch:jalr x11,x0,jr
addi x1,x0,1
jr: jal x12,end
addi x1,x0,1
end:beq x0,x0,end
```



```
#Address     Machine Code           RISC-V Assembly
0x0          0x00A00093             addi x1,x0,10
0x4          0x01400113             addi x2,x0,20
0x8          0x001101B3             add x3,x2,x1
0xc          0x40110233             sub x4,x2,x1
0x10         0x0020A2B3             slt x5,x1,x2
0x14         0x0000F333             and x6,x1,x0
0x18         0x0000E3B3             or x7,x1,x0
0x1c         0x00109413             slli x8,x1,1
0x20         0x0010D493             srli x9,x1,1
0x24         0x00102223             sw x1,4(x0)
0x28         0x00402503             lw x10,4(x0)
0x2c         0x00208663             beq x1,x2,branch
0x30         0x00209463             bne x1,x2,branch
0x34         0x00100093             addi x1,x0,1
0x38         0x040005E7     branch: jalr x11,x0,jr
0x3c         0x00100093             addi x1,x0,1
0x40         0x0080066F         jr: jal x12,end
0x44         0x00100093             addi x1,x0,1
0x48         0x00000063        end: beq x0,x0,end
```

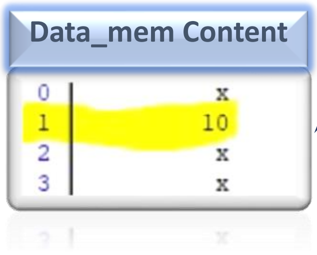
Radix
Address: decimal
data: decimal



Reg_File Content

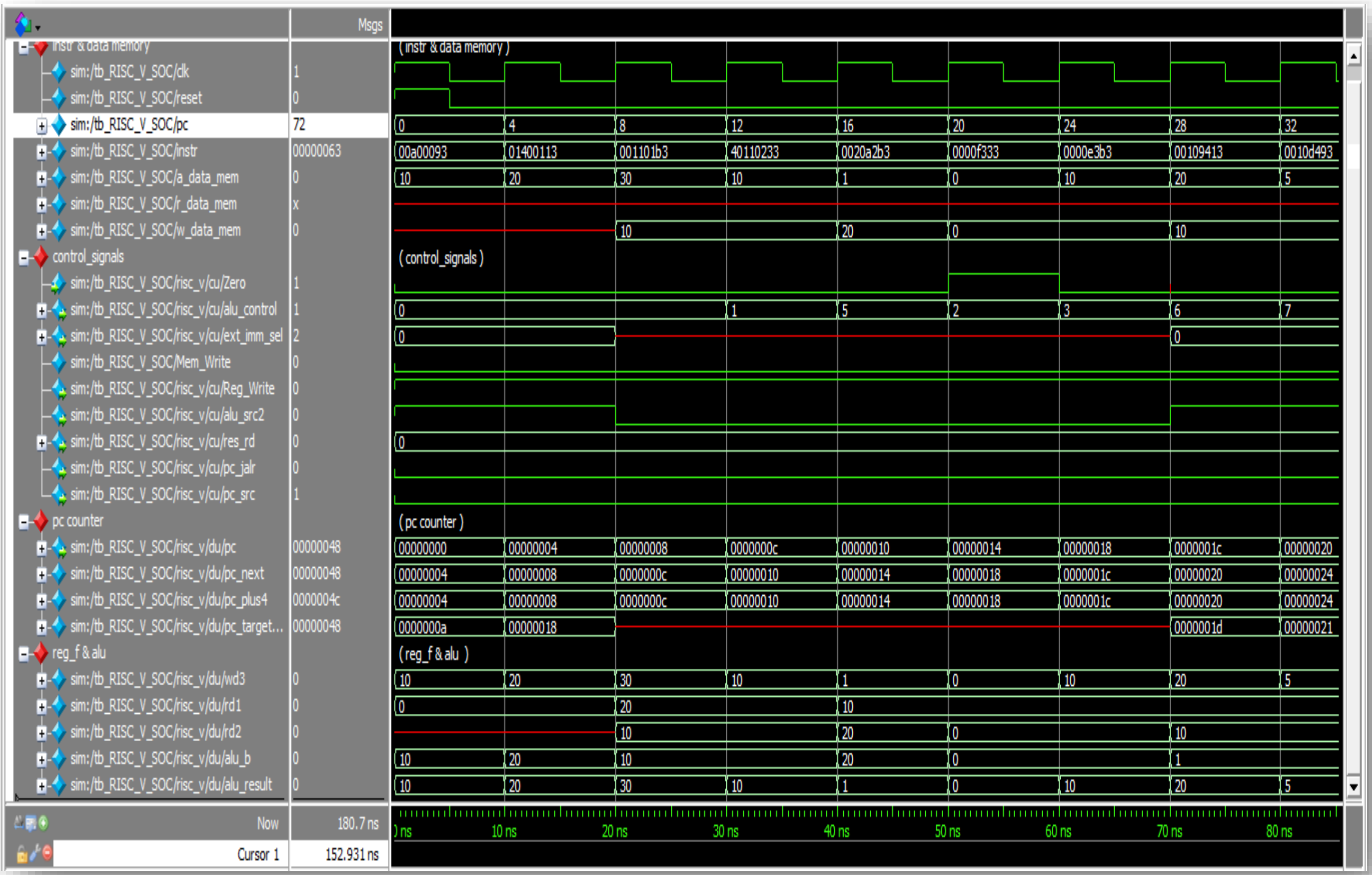
```
13
12   68
11   60
10   10
 9    5
 8   20
 7   10
 6    0
 5    1
 4   10
 3   30
 2   20
 1   10
```

Radix
Word Address: decimal
data: decimal

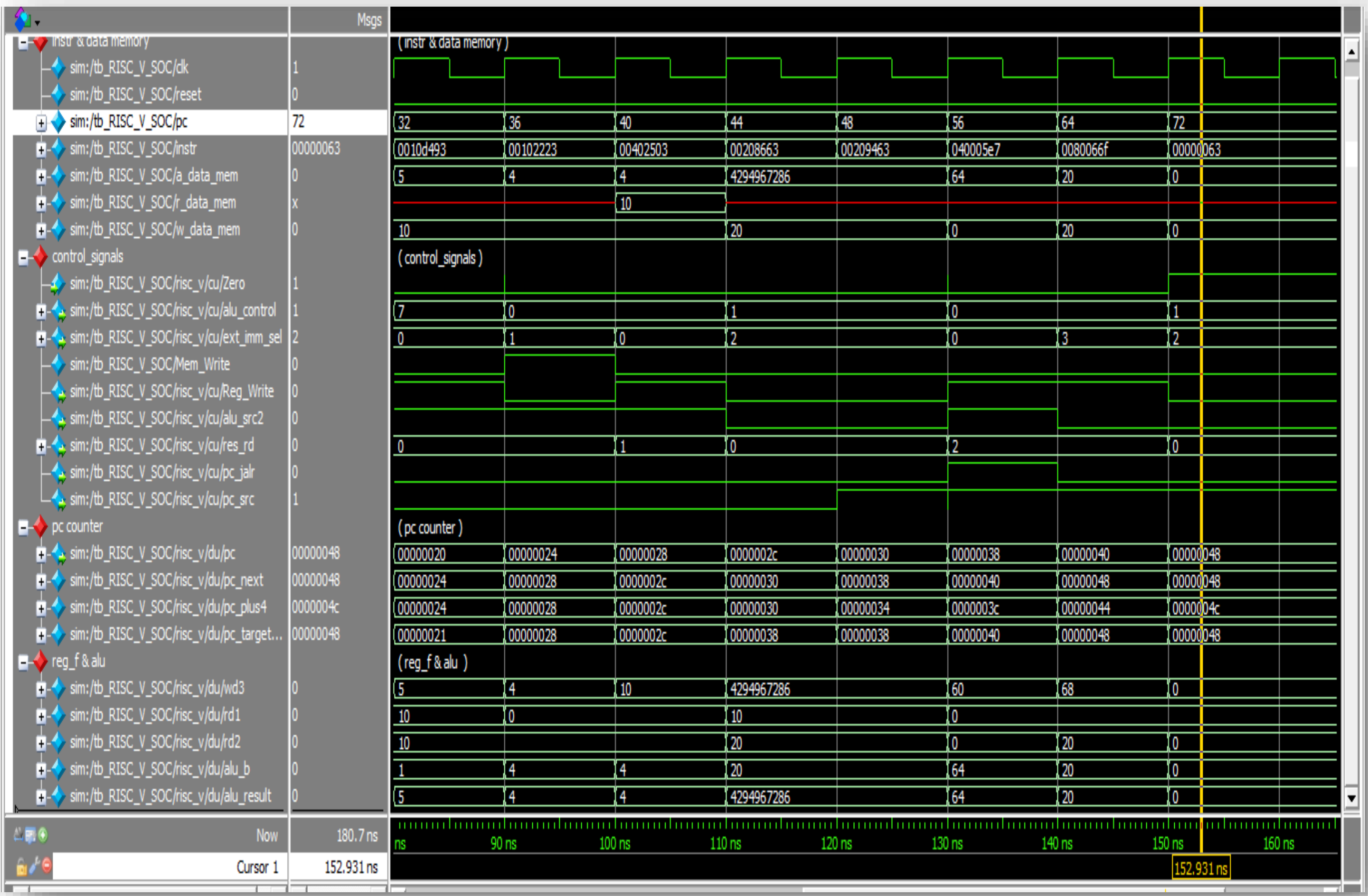


Data_mem Content

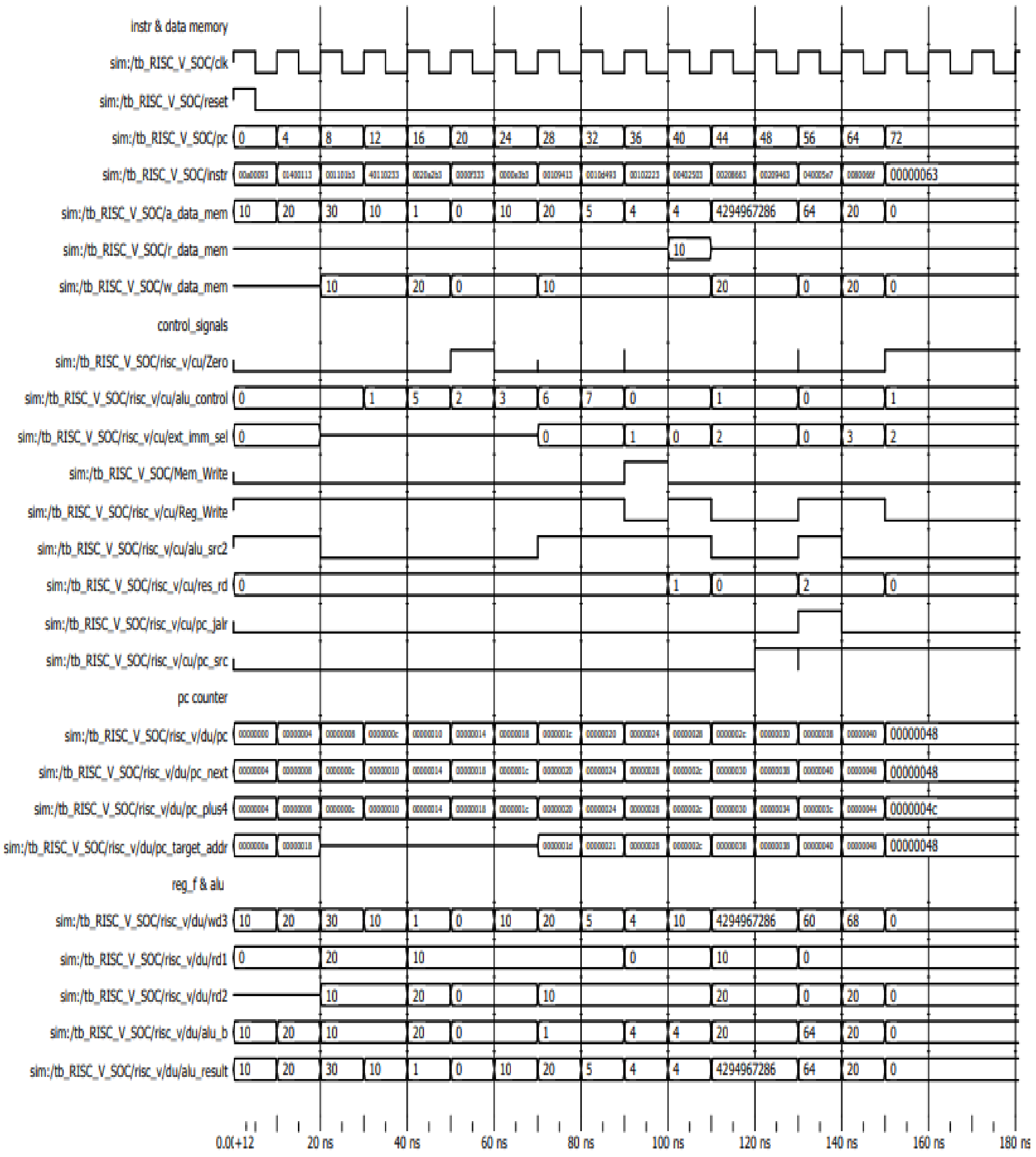
```
0    x
1   10
2    x
3    x
```



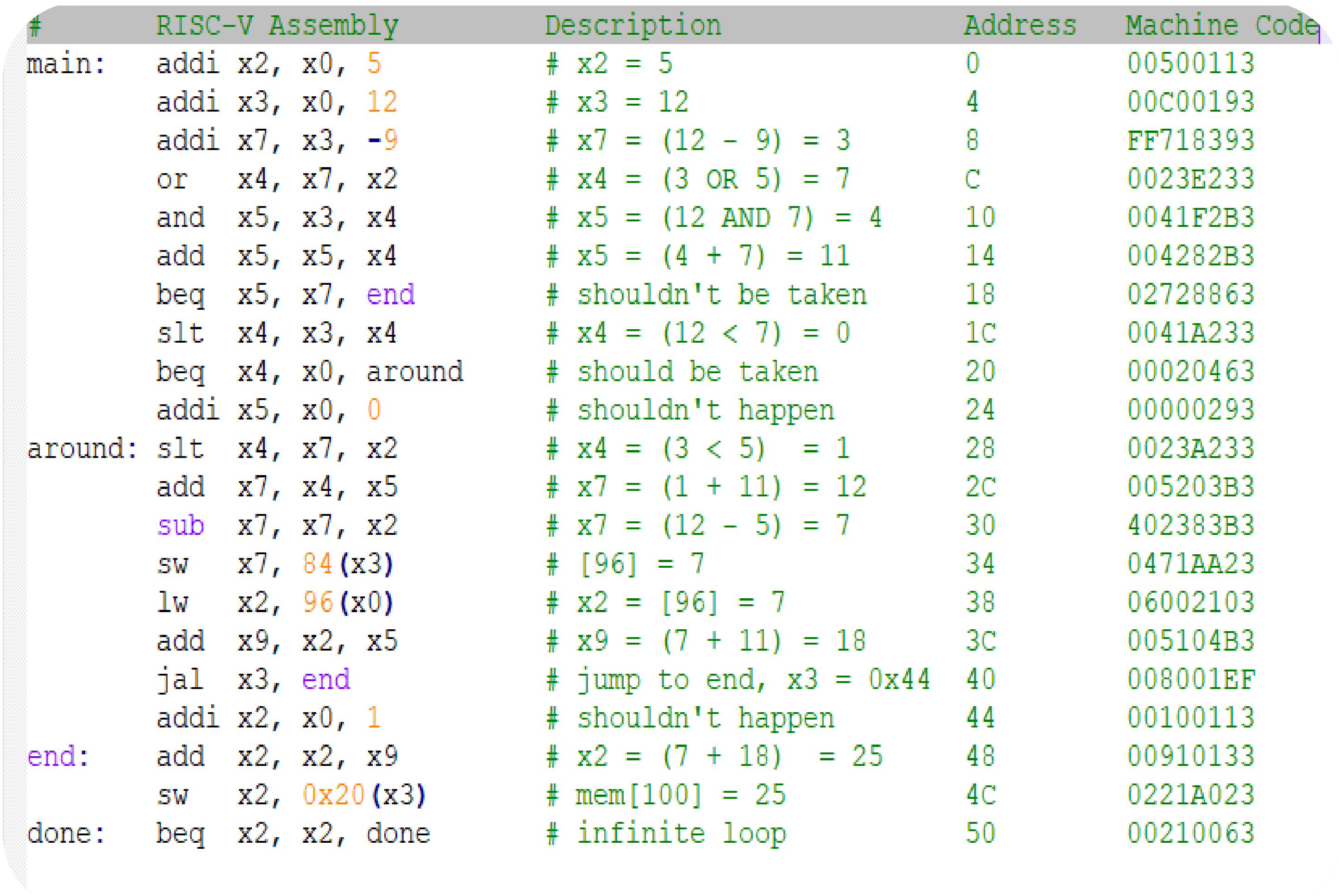
	Msgs
instr & data memory	
sim:/tb_RISC_V_SOC/clk	1
sim:/tb_RISC_V_SOC/reset	0
sim:/tb_RISC_V_SOC/pc	72
sim:/tb_RISC_V_SOC/instr	00000063
sim:/tb_RISC_V_SOC/a_data_mem	0
sim:/tb_RISC_V_SOC/r_data_mem	x
sim:/tb_RISC_V_SOC/w_data_mem	0
control_signals	
sim:/tb_RISC_V_SOC/risc_v/cu/Zero	1
sim:/tb_RISC_V_SOC/risc_v/cu/alu_control	1
sim:/tb_RISC_V_SOC/risc_v/cu/ext_imm_sel	2
sim:/tb_RISC_V_SOC/Mem_Write	0
sim:/tb_RISC_V_SOC/risc_v/cu/Reg_Write	0
sim:/tb_RISC_V_SOC/risc_v/cu/alu_src2	0
sim:/tb_RISC_V_SOC/risc_v/cu/res_rd	0
sim:/tb_RISC_V_SOC/risc_v/cu/pc_jalr	0
sim:/tb_RISC_V_SOC/risc_v/cu/pc_src	1
pc counter	
sim:/tb_RISC_V_SOC/risc_v/du/pc	00000048
sim:/tb_RISC_V_SOC/risc_v/du/pc_next	00000048
sim:/tb_RISC_V_SOC/risc_v/du/pc_plus4	0000004c
sim:/tb_RISC_V_SOC/risc_v/du/pc_target...	00000048
reg_f & alu	
sim:/tb_RISC_V_SOC/risc_v/du/wd3	0
sim:/tb_RISC_V_SOC/risc_v/du/rd1	0
sim:/tb_RISC_V_SOC/risc_v/du/rd2	0
sim:/tb_RISC_V_SOC/risc_v/du/alu_b	0
sim:/tb_RISC_V_SOC/risc_v/du/alu_result	0
Now	180.7 ns
Cursor 1	152.931 ns



	Msgs
instr & data memory	
sim:/tb_RISC_V_SOC/clk	1
sim:/tb_RISC_V_SOC/reset	0
sim:/tb_RISC_V_SOC/pc	72
sim:/tb_RISC_V_SOC/instr	00000063
sim:/tb_RISC_V_SOC/a_data_mem	0
sim:/tb_RISC_V_SOC/r_data_mem	x
sim:/tb_RISC_V_SOC/w_data_mem	0
control_signals	
sim:/tb_RISC_V_SOC/risc_v/cu/Zero	1
sim:/tb_RISC_V_SOC/risc_v/cu/alu_control	1
sim:/tb_RISC_V_SOC/risc_v/cu/ext_imm_sel	2
sim:/tb_RISC_V_SOC/Mem_Write	0
sim:/tb_RISC_V_SOC/risc_v/cu/Reg_Write	0
sim:/tb_RISC_V_SOC/risc_v/cu/alu_src2	0
sim:/tb_RISC_V_SOC/risc_v/cu/res_rd	0
sim:/tb_RISC_V_SOC/risc_v/cu/pc_jalr	0
sim:/tb_RISC_V_SOC/risc_v/cu/pc_src	1
pc counter	
sim:/tb_RISC_V_SOC/risc_v/du/pc	00000048
sim:/tb_RISC_V_SOC/risc_v/du/pc_next	00000048
sim:/tb_RISC_V_SOC/risc_v/du/pc_plus4	0000004c
sim:/tb_RISC_V_SOC/risc_v/du/pc_target...	00000048
reg_f & alu	
sim:/tb_RISC_V_SOC/risc_v/du/wd3	0
sim:/tb_RISC_V_SOC/risc_v/du/rd1	0
sim:/tb_RISC_V_SOC/risc_v/du/rd2	0
sim:/tb_RISC_V_SOC/risc_v/du/alu_b	0
sim:/tb_RISC_V_SOC/risc_v/du/alu_result	0
Now	180.7 ns
Cursor 1	152.931 ns


instr & data memory
sim:/tb_RISC_V_SOC/clk
sim:/tb_RISC_V_SOC/reset
sim:/tb_RISC_V_SOC/pc
0 4 8 12 16 20 24 28 32 36 40 44 48 56 64 72
sim:/tb_RISC_V_SOC/instr
00000063
sim:/tb_RISC_V_SOC/a_data_mem
10 20 30 10 1 0 10 20 5 4 4 4294967286 64 20 0
sim:/tb_RISC_V_SOC/r_data_mem
10
sim:/tb_RISC_V_SOC/w_data_mem
10 20 0 10 20 0 20 0
control_signals
sim:/tb_RISC_V_SOC/risc_v/cu/Zero
sim:/tb_RISC_V_SOC/risc_v/cu/alu_control
0 1 5 2 3 6 7 0 1 0 1
sim:/tb_RISC_V_SOC/risc_v/cu/ext_imm_sel
0 0 1 0 2 0 3 2
sim:/tb_RISC_V_SOC/Mem_Write
sim:/tb_RISC_V_SOC/risc_v/cu/Reg_Write
sim:/tb_RISC_V_SOC/risc_v/cu/alu_src2
sim:/tb_RISC_V_SOC/risc_v/cu/res_rd
0 1 0 2 0
sim:/tb_RISC_V_SOC/risc_v/cu/pc_jalr
sim:/tb_RISC_V_SOC/risc_v/cu/pc_src
pc counter
sim:/tb_RISC_V_SOC/risc_v/du/pc
00000048
sim:/tb_RISC_V_SOC/risc_v/du/pc_next
00000048
sim:/tb_RISC_V_SOC/risc_v/du/pc_plus4
0000004c
sim:/tb_RISC_V_SOC/risc_v/du/pc_target_addr
00000048
reg_f & alu
sim:/tb_RISC_V_SOC/risc_v/du/wd3
10 20 30 10 1 0 10 20 5 4 10 4294967286 60 68 0
sim:/tb_RISC_V_SOC/risc_v/du/rd1
0 20 10 0 10 0
sim:/tb_RISC_V_SOC/risc_v/du/rd2
10 20 0 10 20 0 20 0
sim:/tb_RISC_V_SOC/risc_v/du/alu_b
10 20 10 20 0 1 4 4 20 64 20 0
sim:/tb_RISC_V_SOC/risc_v/du/alu_result
10 20 30 10 1 0 10 20 5 4 4 4294967286 64 20 0
0.0(+12
20 ns
40 ns
60 ns
80 ns
100 ns
120 ns
140 ns
160 ns
180 ns

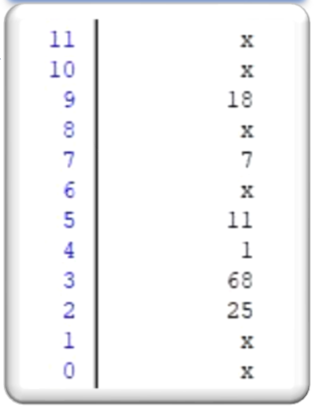
❖ Second Test Case Assembly code and its output



```
#        RISC-V Assembly         Description                Address    Machine Code
main:    addi x2, x0, 5          # x2 = 5                   0          00500113
         addi x3, x0, 12         # x3 = 12                  4          00C00193
         addi x7, x3, -9         # x7 = (12 - 9) = 3        8          FF718393
         or   x4, x7, x2         # x4 = (3 OR 5) = 7        C          0023E233
         and  x5, x3, x4         # x5 = (12 AND 7) = 4      10         0041F2B3
         add  x5, x5, x4         # x5 = (4 + 7) = 11        14         004282B3
         beq  x5, x7, end        # shouldn't be taken       18         02728863
         slt  x4, x3, x4         # x4 = (12 < 7) = 0        1C         0041A233
         beq  x4, x0, around     # should be taken          20         00020463
         addi x5, x0, 0          # shouldn't happen         24         00000293
around:  slt  x4, x7, x2         # x4 = (3 < 5)  = 1        28         0023A233
         add  x7, x4, x5         # x7 = (1 + 11) = 12       2C         005203B3
         sub  x7, x7, x2         # x7 = (12 - 5) = 7        30         402383B3
         sw   x7, 84(x3)         # [96] = 7                 34         0471AA23
         lw   x2, 96(x0)         # x2 = [96] = 7            38         06002103
         add  x9, x2, x5         # x9 = (7 + 11) = 18       3C         005104B3
         jal  x3, end            # jump to end, x3 = 0x44   40         008001EF
         addi x2, x0, 1          # shouldn't happen         44         00100113
end:     add  x2, x2, x9         # x2 = (7 + 18)  = 25      48         00910133
         sw   x2, 0x20(x3)       # mem[100] = 25            4C         0221A023
done:    beq  x2, x2, done       # infinite loop            50         00210063
```

Radix
Address: decimal
data: decimal

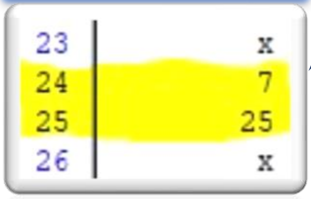
Reg_File Content



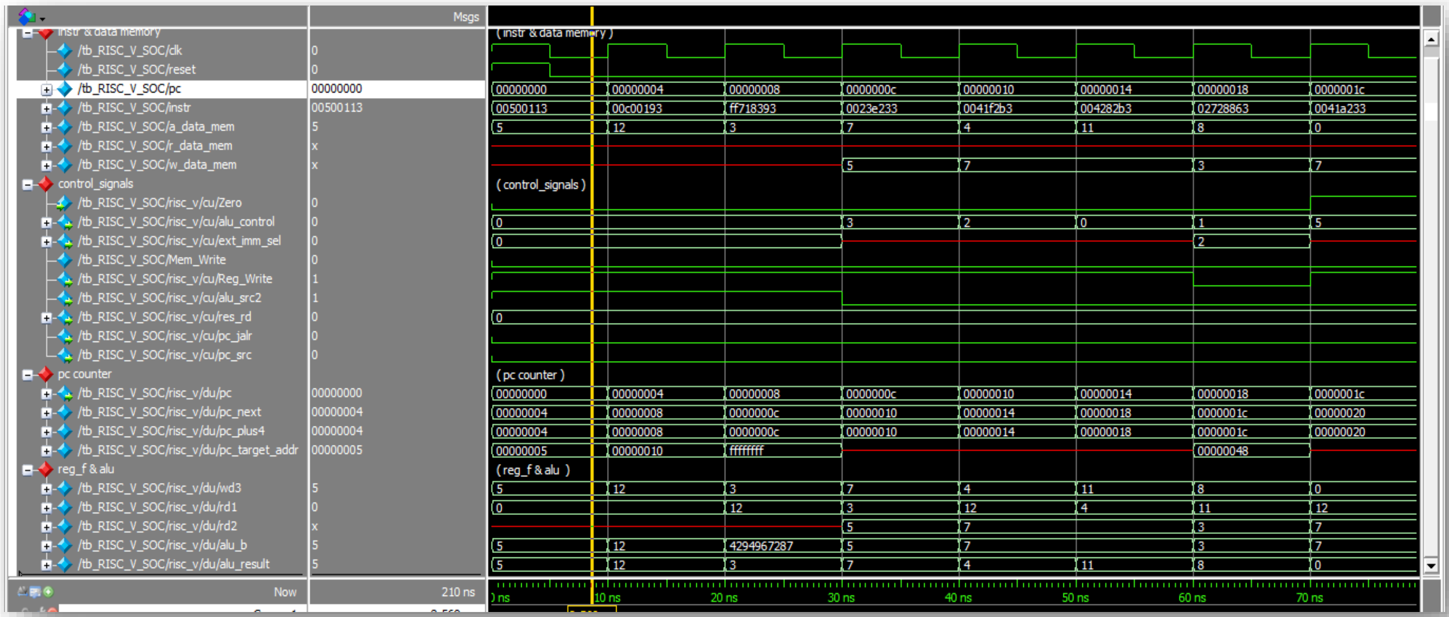
```
11 | x
10 | x
 9 | 18
 8 | x
 7 | 7
 6 | x
 5 | 11
 4 | 1
 3 | 68
 2 | 25
 1 | x
 0 | x
```

Radix
Word Address: decimal
data: decimal

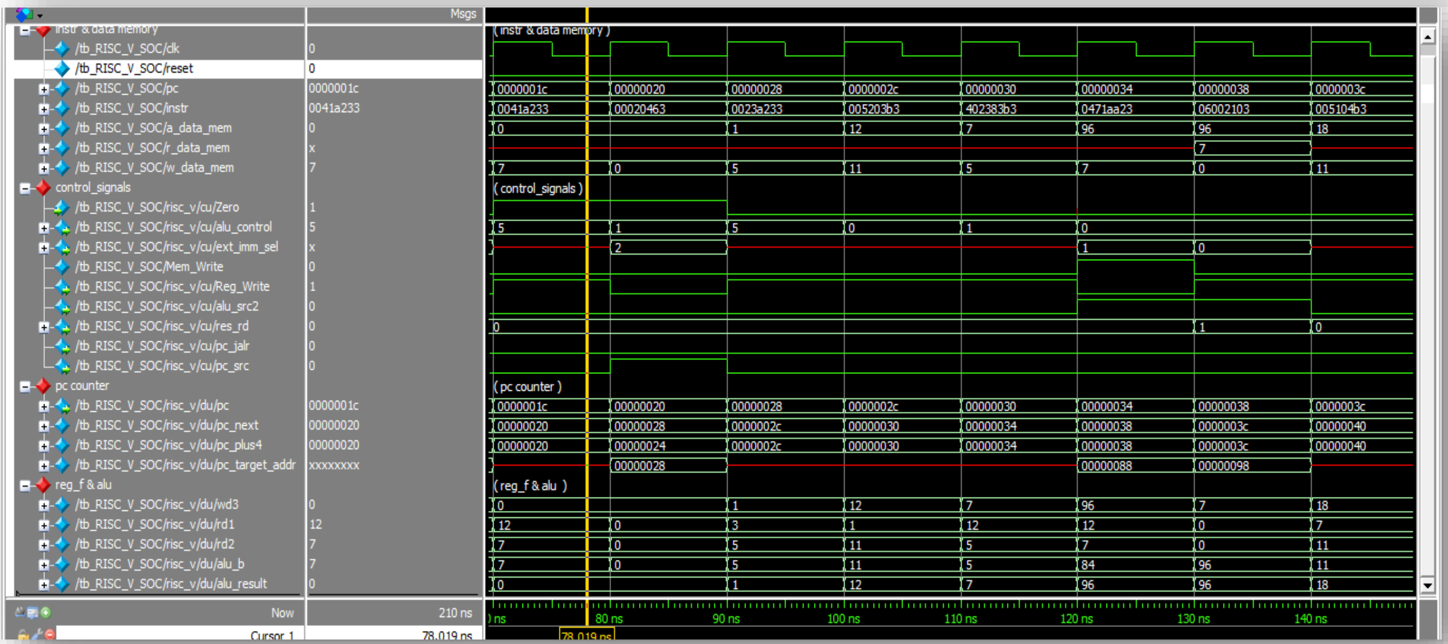
Data_mem Content



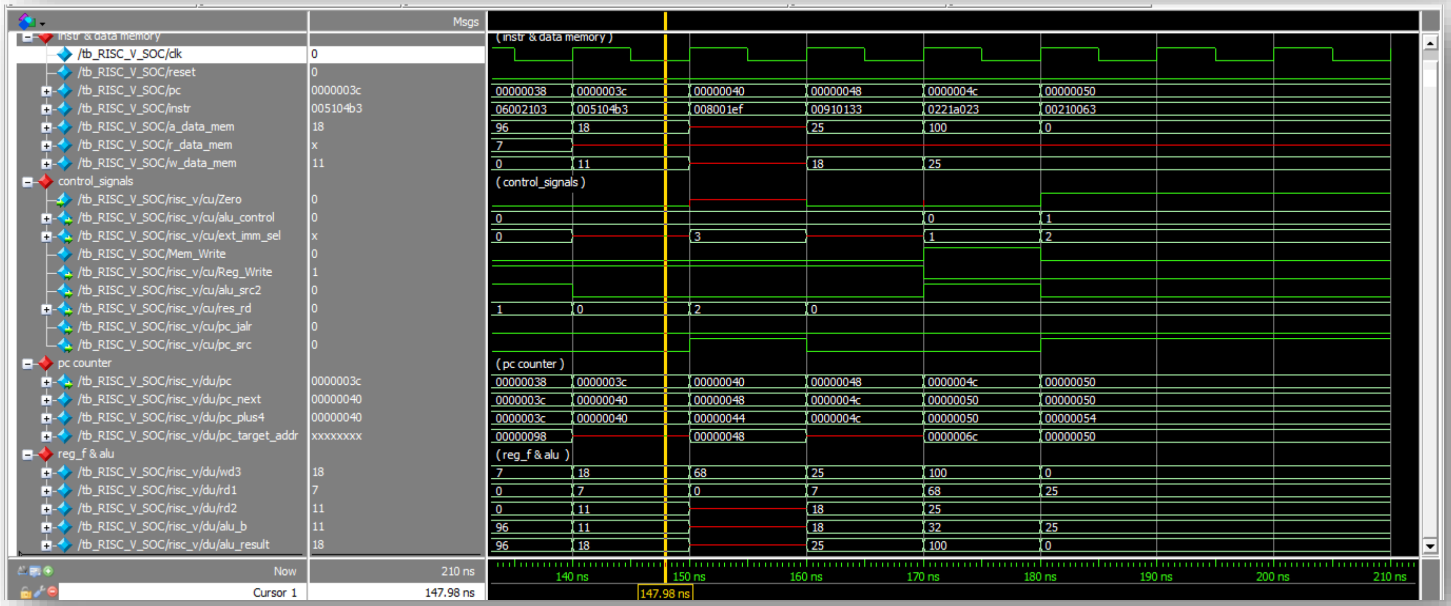
```
23 | x
24 | 7
25 | 25
26 | x
```



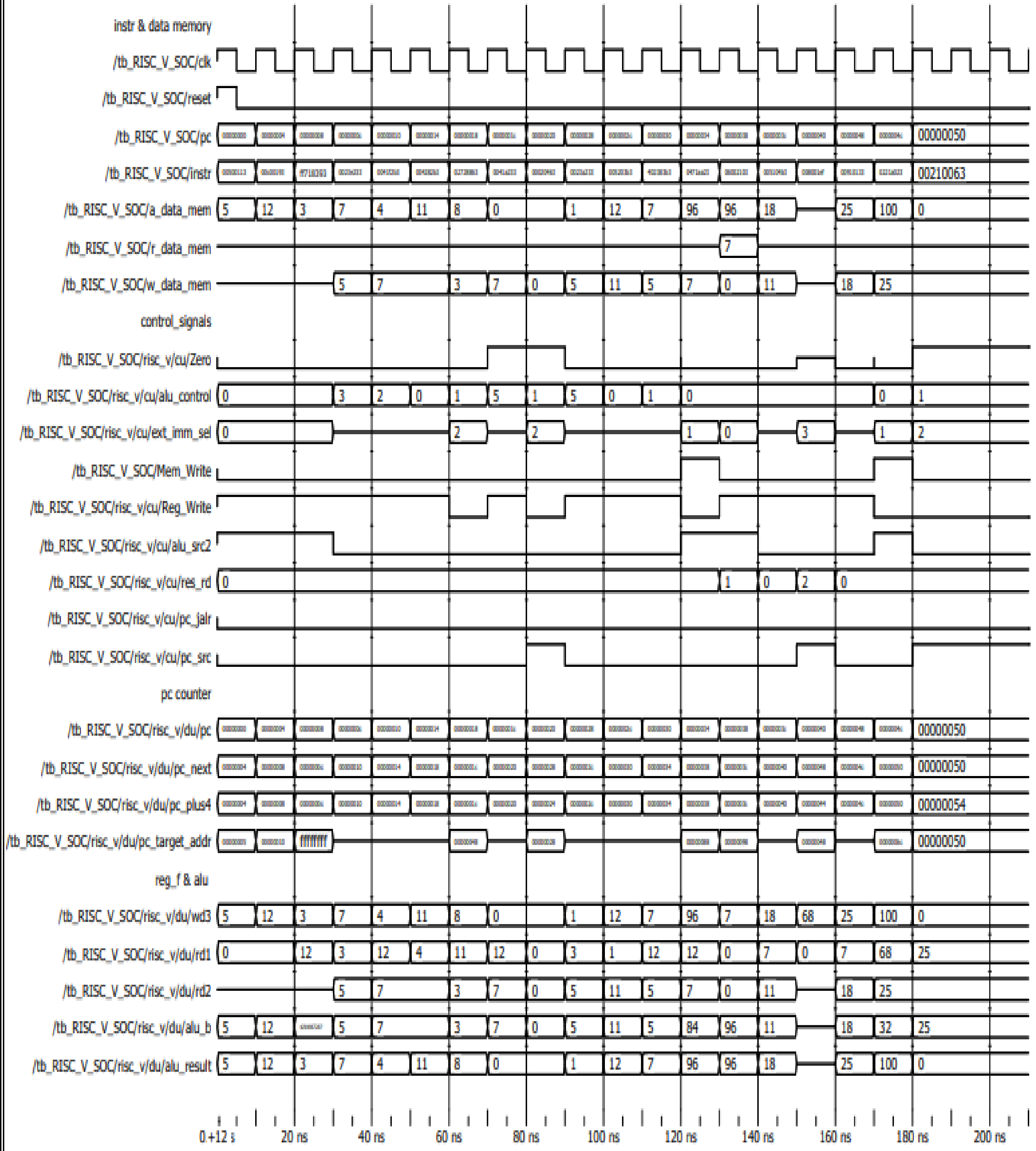
```
instr & data memory
/tb_RISC_V_SOC/clk                        0
/tb_RISC_V_SOC/reset                      0
/tb_RISC_V_SOC/pc                         00000000
/tb_RISC_V_SOC/instr                      00500113
/tb_RISC_V_SOC/a_data_mem                 5
/tb_RISC_V_SOC/r_data_mem                 x
/tb_RISC_V_SOC/w_data_mem                 x
control_signals
/tb_RISC_V_SOC/risc_v/cu/Zero             0
/tb_RISC_V_SOC/risc_v/cu/alu_control      0
/tb_RISC_V_SOC/risc_v/cu/ext_imm_sel      0
/tb_RISC_V_SOC/Mem_Write                  0
/tb_RISC_V_SOC/risc_v/cu/Reg_Write        1
/tb_RISC_V_SOC/risc_v/cu/alu_src2         1
/tb_RISC_V_SOC/risc_v/cu/res_rd           0
/tb_RISC_V_SOC/risc_v/cu/pc_jalr          0
/tb_RISC_V_SOC/risc_v/cu/pc_src           0
pc counter
/tb_RISC_V_SOC/risc_v/du/pc               00000000
/tb_RISC_V_SOC/risc_v/du/pc_next          00000004
/tb_RISC_V_SOC/risc_v/du/pc_plus4         00000004
/tb_RISC_V_SOC/risc_v/du/pc_target_addr   00000005
reg_f & alu
/tb_RISC_V_SOC/risc_v/du/wd3              5
/tb_RISC_V_SOC/risc_v/du/rd1              0
/tb_RISC_V_SOC/risc_v/du/rd2              x
/tb_RISC_V_SOC/risc_v/du/alu_b            5
/tb_RISC_V_SOC/risc_v/du/alu_result       5
Now                                       210 ns
```



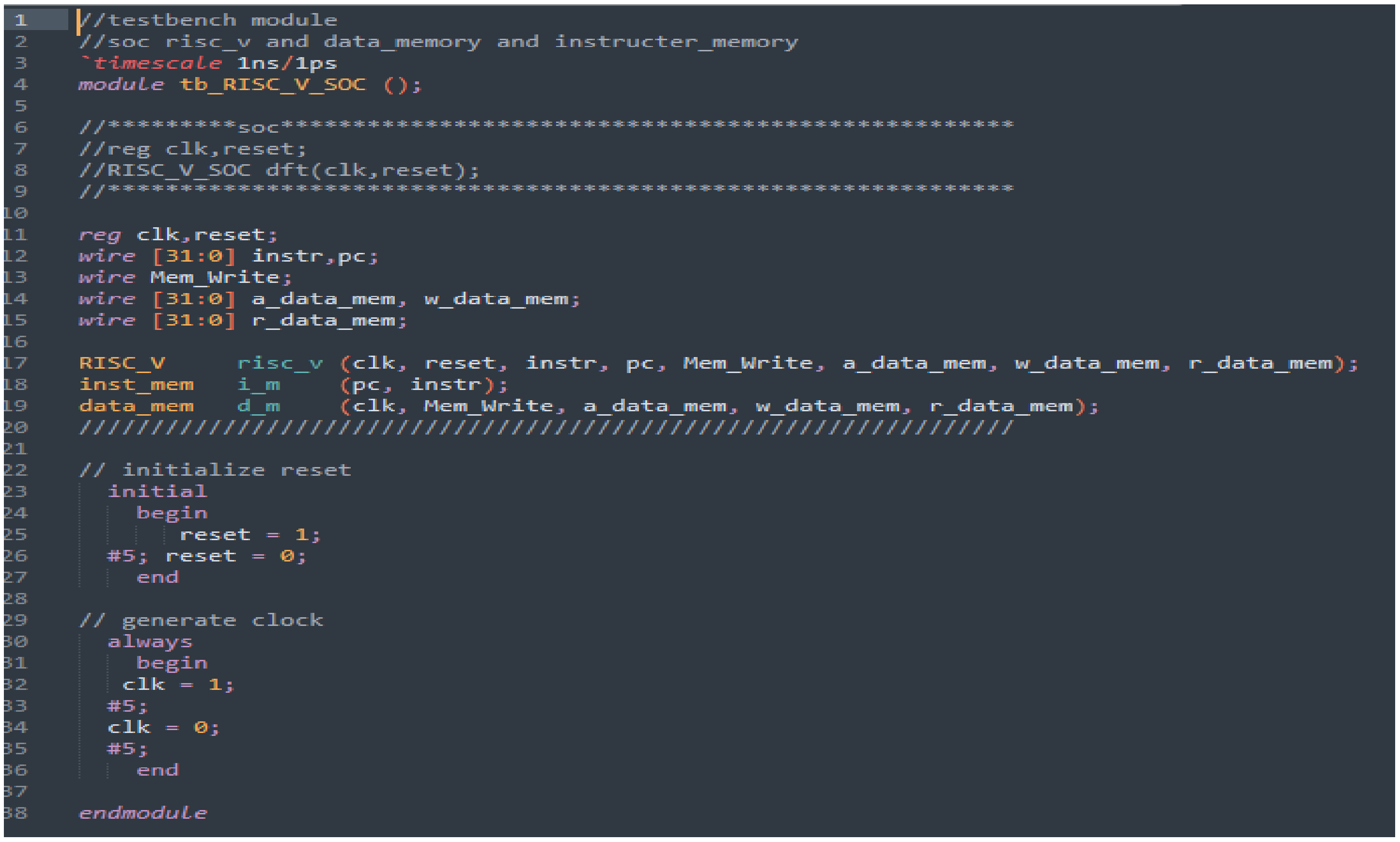
```
instr & data memory
/tb_RISC_V_SOC/clk                        0
/tb_RISC_V_SOC/reset                      0
/tb_RISC_V_SOC/pc                         0000001c
/tb_RISC_V_SOC/instr                      0041a233
/tb_RISC_V_SOC/a_data_mem                 0
/tb_RISC_V_SOC/r_data_mem                 x
/tb_RISC_V_SOC/w_data_mem                 7
control_signals
/tb_RISC_V_SOC/risc_v/cu/Zero             1
/tb_RISC_V_SOC/risc_v/cu/alu_control      5
/tb_RISC_V_SOC/risc_v/cu/ext_imm_sel      x
/tb_RISC_V_SOC/Mem_Write                  0
/tb_RISC_V_SOC/risc_v/cu/Reg_Write        1
/tb_RISC_V_SOC/risc_v/cu/alu_src2         0
/tb_RISC_V_SOC/risc_v/cu/res_rd           0
/tb_RISC_V_SOC/risc_v/cu/pc_jalr          0
/tb_RISC_V_SOC/risc_v/cu/pc_src           0
pc counter
/tb_RISC_V_SOC/risc_v/du/pc               0000001c
/tb_RISC_V_SOC/risc_v/du/pc_next          00000020
/tb_RISC_V_SOC/risc_v/du/pc_plus4         00000020
/tb_RISC_V_SOC/risc_v/du/pc_target_addr   xxxxxxxx
reg_f & alu
/tb_RISC_V_SOC/risc_v/du/wd3              0
/tb_RISC_V_SOC/risc_v/du/rd1              12
/tb_RISC_V_SOC/risc_v/du/rd2              7
/tb_RISC_V_SOC/risc_v/du/alu_b            7
/tb_RISC_V_SOC/risc_v/du/alu_result       0
Now                                       210 ns
Cursor 1                                  78.019 ns
```



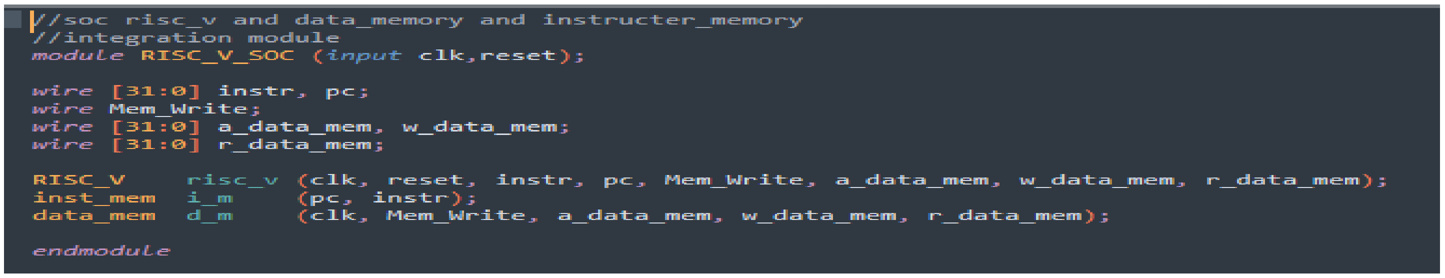
```
instr & data memory
/tb_RISC_V_SOC/clk                        0
/tb_RISC_V_SOC/reset                      0
/tb_RISC_V_SOC/pc                         0000003c
/tb_RISC_V_SOC/instr                      005104b3
/tb_RISC_V_SOC/a_data_mem                 18
/tb_RISC_V_SOC/r_data_mem                 x
/tb_RISC_V_SOC/w_data_mem                 11
control_signals
/tb_RISC_V_SOC/risc_v/cu/Zero             0
/tb_RISC_V_SOC/risc_v/cu/alu_control      0
/tb_RISC_V_SOC/risc_v/cu/ext_imm_sel      x
/tb_RISC_V_SOC/Mem_Write                  0
/tb_RISC_V_SOC/risc_v/cu/Reg_Write        1
/tb_RISC_V_SOC/risc_v/cu/alu_src2         0
/tb_RISC_V_SOC/risc_v/cu/res_rd           0
/tb_RISC_V_SOC/risc_v/cu/pc_jalr          0
/tb_RISC_V_SOC/risc_v/cu/pc_src           0
pc counter
/tb_RISC_V_SOC/risc_v/du/pc               0000003c
/tb_RISC_V_SOC/risc_v/du/pc_next          00000040
/tb_RISC_V_SOC/risc_v/du/pc_plus4         00000040
/tb_RISC_V_SOC/risc_v/du/pc_target_addr   xxxxxxxx
reg_f & alu
/tb_RISC_V_SOC/risc_v/du/wd3              18
/tb_RISC_V_SOC/risc_v/du/rd1              7
/tb_RISC_V_SOC/risc_v/du/rd2              11
/tb_RISC_V_SOC/risc_v/du/alu_b            11
/tb_RISC_V_SOC/risc_v/du/alu_result       18
Now                                       210 ns
Cursor 1                                  147.98 ns
```


instr & data memory
/tb_RISC_V_SOC/clk
/tb_RISC_V_SOC/reset
/tb_RISC_V_SOC/pc
/tb_RISC_V_SOC/instr
/tb_RISC_V_SOC/a_data_mem
/tb_RISC_V_SOC/r_data_mem
/tb_RISC_V_SOC/w_data_mem
control_signals
/tb_RISC_V_SOC/risc_v/cu/Zero
/tb_RISC_V_SOC/risc_v/cu/alu_control
/tb_RISC_V_SOC/risc_v/cu/ext_imm_sel
/tb_RISC_V_SOC/Mem_Write
/tb_RISC_V_SOC/risc_v/cu/Reg_Write
/tb_RISC_V_SOC/risc_v/cu/alu_src2
/tb_RISC_V_SOC/risc_v/cu/res_rd
/tb_RISC_V_SOC/risc_v/cu/pc_jalr
/tb_RISC_V_SOC/risc_v/cu/pc_src
pc counter
/tb_RISC_V_SOC/risc_v/du/pc
/tb_RISC_V_SOC/risc_v/du/pc_next
/tb_RISC_V_SOC/risc_v/du/pc_plus4
/tb_RISC_V_SOC/risc_v/du/pc_target_addr
reg_f & alu
/tb_RISC_V_SOC/risc_v/du/wd3
/tb_RISC_V_SOC/risc_v/du/rd1
/tb_RISC_V_SOC/risc_v/du/rd2
/tb_RISC_V_SOC/risc_v/du/alu_b
/tb_RISC_V_SOC/risc_v/du/alu_result
00000050
00210063
00000054
0 +12 s
20 ns
40 ns
60 ns
80 ns
100 ns
120 ns
140 ns
160 ns
180 ns
200 ns

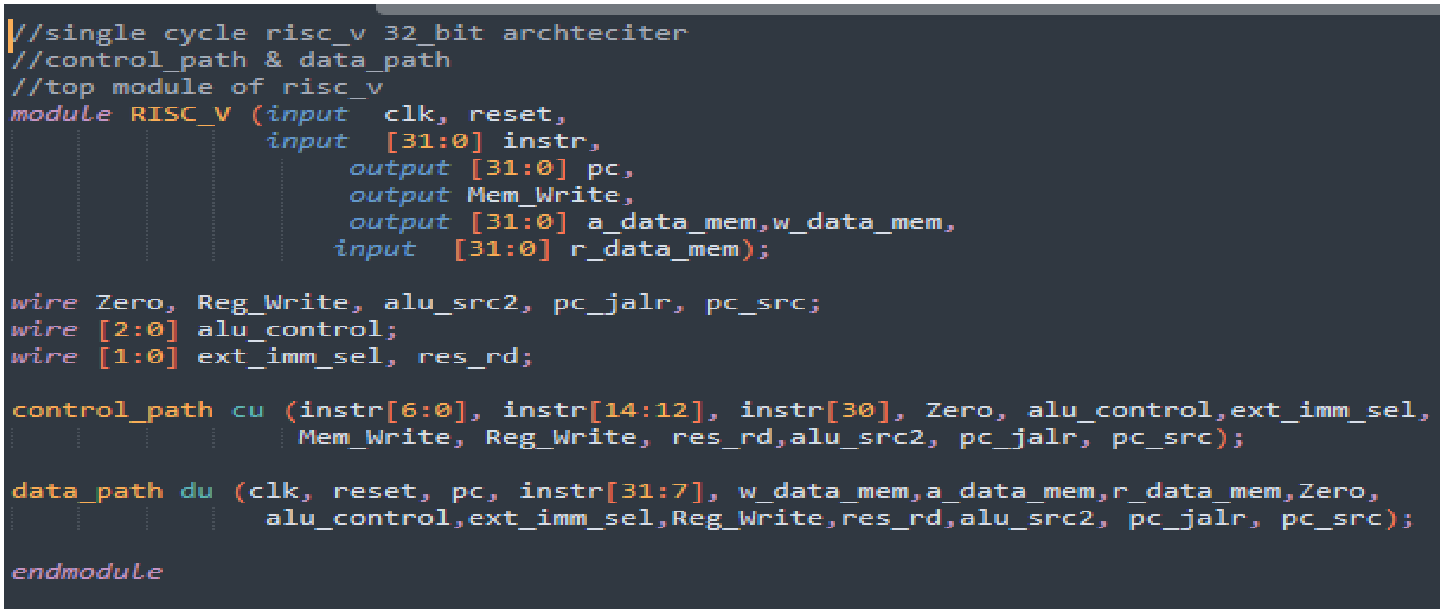
4. Verilog Codes:



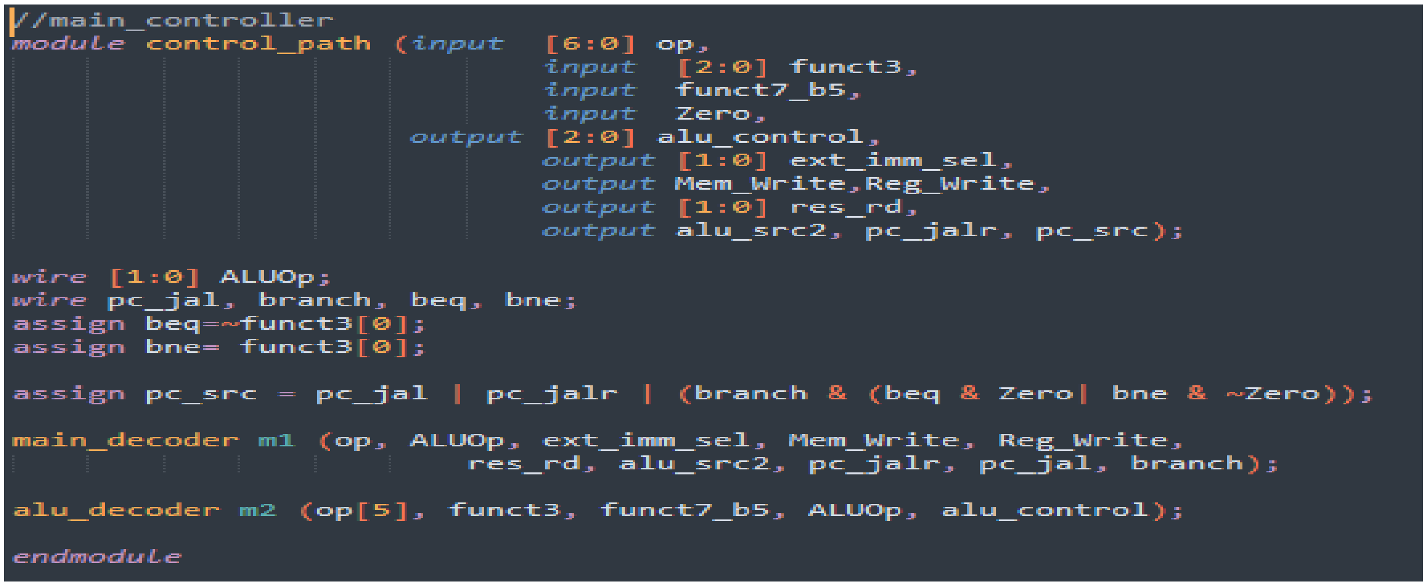
```verilog
//testbench module
//soc risc_v and data_memory and instructer_memory
`timescale 1ns/1ps
module tb_RISC_V_SOC ();

//*********soc*****************************************************
//reg clk,reset;
//RISC_V_SOC dft(clk,reset);
//****************************************************************

reg clk,reset;
wire [31:0] instr,pc;
wire Mem_Write;
wire [31:0] a_data_mem, w_data_mem;
wire [31:0] r_data_mem;

RISC_V     risc_v (clk, reset, instr, pc, Mem_Write, a_data_mem, w_data_mem, r_data_mem);
inst_mem   i_m    (pc, instr);
data_mem   d_m    (clk, Mem_Write, a_data_mem, w_data_mem, r_data_mem);
//////////////////////////////////////////////////////////////////

// initialize reset
  initial
    begin
       reset = 1;
  #5; reset = 0;
    end

// generate clock
  always
    begin
   clk = 1;
  #5;
  clk = 0;
  #5;
    end

endmodule
```



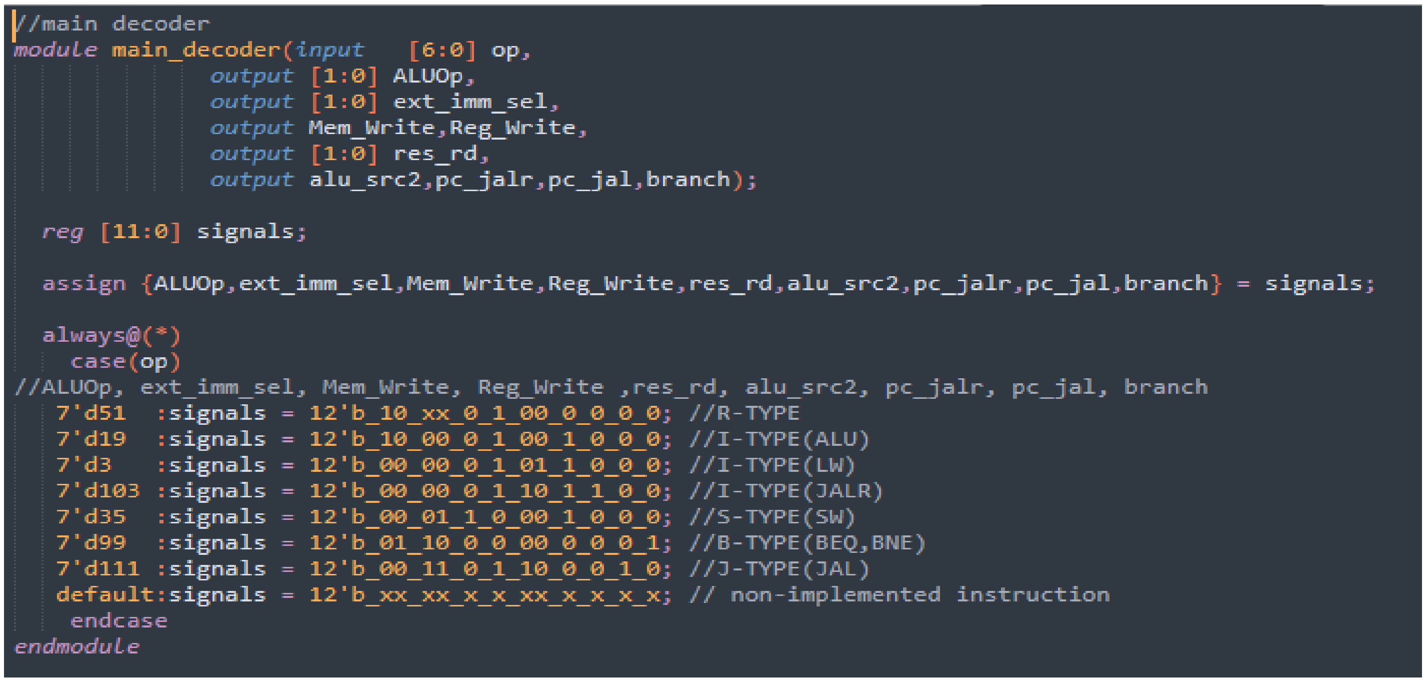
```verilog
//soc risc_v and data_memory and instructer_memory
//integration module
module RISC_V_SOC (input clk,reset);

wire [31:0] instr, pc;
wire Mem_Write;
wire [31:0] a_data_mem, w_data_mem;
wire [31:0] r_data_mem;

RISC_V     risc_v (clk, reset, instr, pc, Mem_Write, a_data_mem, w_data_mem, r_data_mem);
inst_mem   i_m    (pc, instr);
data_mem   d_m    (clk, Mem_Write, a_data_mem, w_data_mem, r_data_mem);

endmodule
```



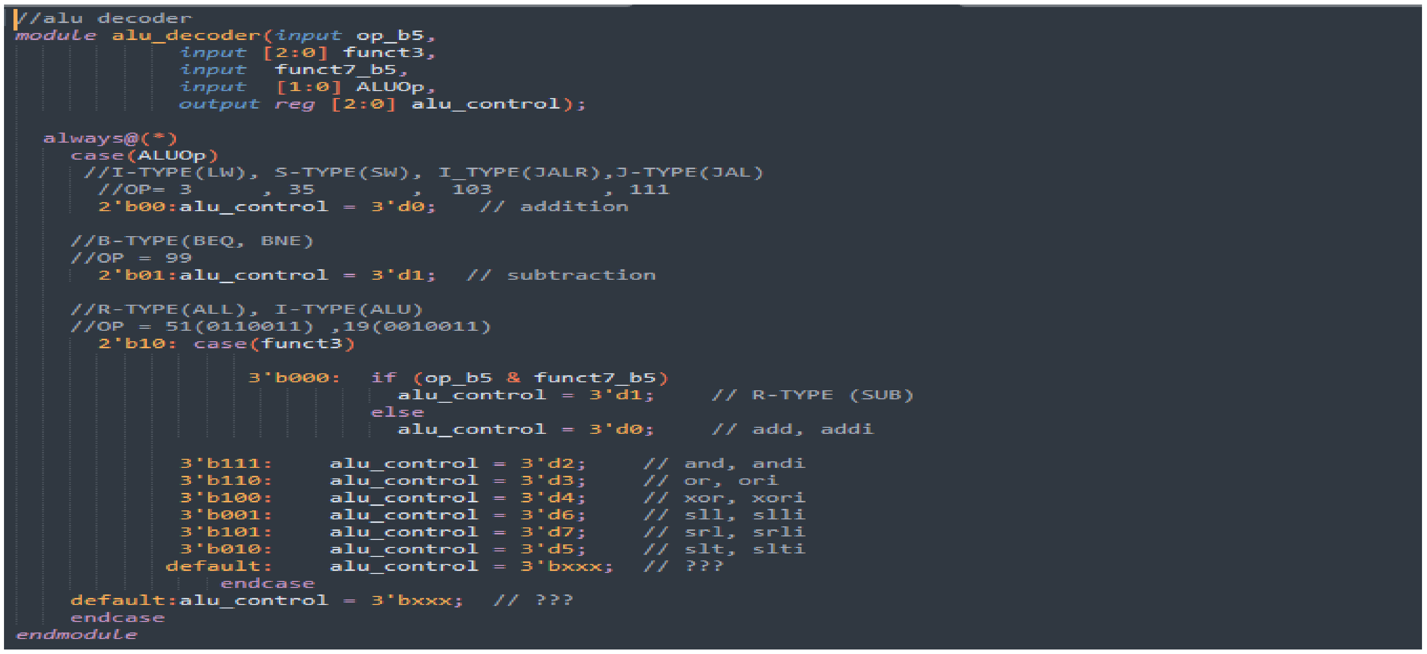
```verilog
//single cycle risc_v 32_bit archteciter
//control_path & data_path
//top module of risc_v
module RISC_V (input  clk, reset,
               input   [31:0] instr,
                   output [31:0] pc,
                   output Mem_Write,
                   output [31:0] a_data_mem,w_data_mem,
                  input   [31:0] r_data_mem);

wire Zero, Reg_Write, alu_src2, pc_jalr, pc_src;
wire [2:0] alu_control;
wire [1:0] ext_imm_sel, res_rd;

control_path cu (instr[6:0], instr[14:12], instr[30], Zero, alu_control,ext_imm_sel,
                 Mem_Write, Reg_Write, res_rd,alu_src2, pc_jalr, pc_src);

data_path du (clk, reset, pc, instr[31:7], w_data_mem,a_data_mem,r_data_mem,Zero,
               alu_control,ext_imm_sel,Reg_Write,res_rd,alu_src2, pc_jalr, pc_src);

endmodule
```



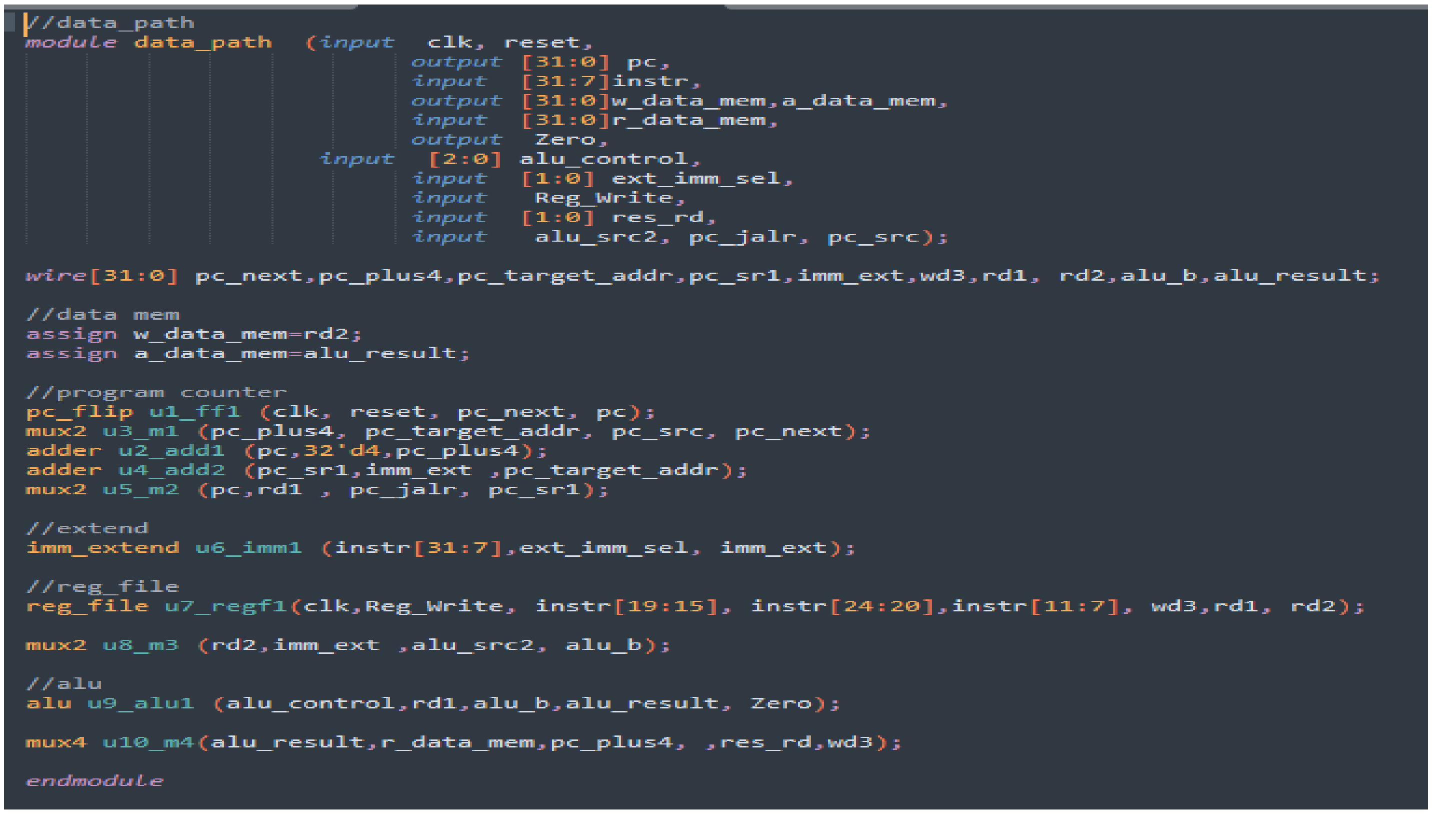
```verilog
//main_controller
module control_path (input   [6:0] op,
                             input   [2:0] funct3,
                             input   funct7_b5,
                             input   Zero,
                      output [2:0] alu_control,
                             output  [1:0] ext_imm_sel,
                             output Mem_Write,Reg_Write,
                             output  [1:0] res_rd,
                             output alu_src2, pc_jalr, pc_src);

wire [1:0] ALUOp;
wire pc_jal, branch, beq, bne;
assign beq=~funct3[0];
assign bne= funct3[0];

assign pc_src = pc_jal | pc_jalr | (branch & (beq & Zero| bne & ~Zero));

main_decoder m1 (op, ALUOp, ext_imm_sel, Mem_Write, Reg_Write,
                        res_rd, alu_src2, pc_jalr, pc_jal, branch);

alu_decoder m2 (op[5], funct3, funct7_b5, ALUOp, alu_control);

endmodule
```



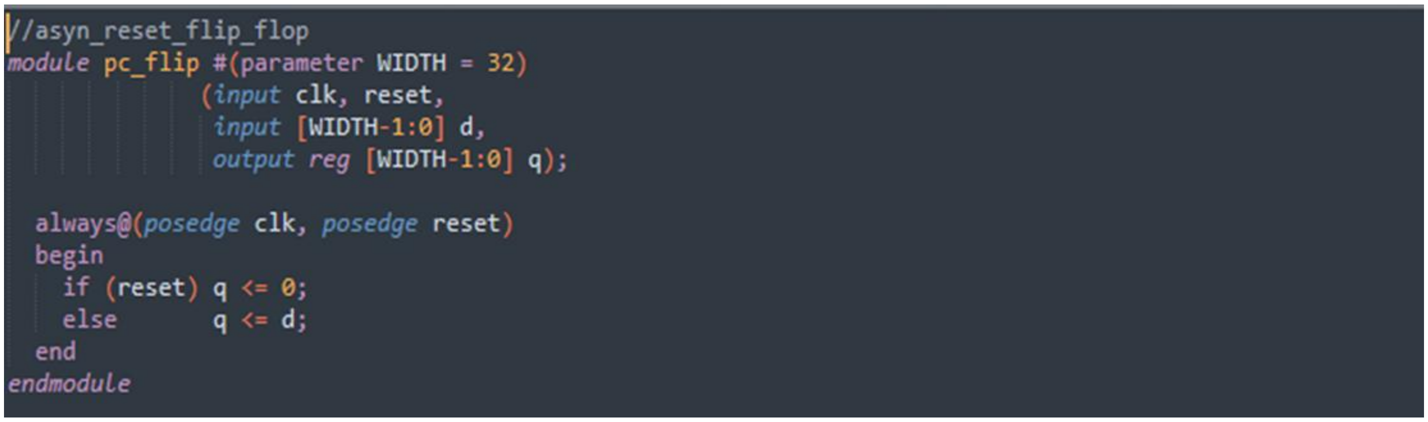
```verilog
//main decoder
module main_decoder(input   [6:0] op,
              output [1:0] ALUOp,
              output [1:0] ext_imm_sel,
              output Mem_Write,Reg_Write,
              output [1:0] res_rd,
              output alu_src2,pc_jalr,pc_jal,branch);

  reg [11:0] signals;

  assign {ALUOp,ext_imm_sel,Mem_Write,Reg_Write,res_rd,alu_src2,pc_jalr,pc_jal,branch} = signals;

  always@(*)
    case(op)
//ALUOp, ext_imm_sel, Mem_Write, Reg_Write ,res_rd, alu_src2, pc_jalr, pc_jal, branch
   7'd51  :signals = 12'b_10_xx_0_1_00_0_0_0_0; //R-TYPE
   7'd19  :signals = 12'b_10_00_0_1_00_1_0_0_0; //I-TYPE(ALU)
   7'd3   :signals = 12'b_00_00_0_1_01_1_0_0_0; //I-TYPE(LW)
   7'd103 :signals = 12'b_00_00_0_1_10_1_1_0_0; //I-TYPE(JALR)
   7'd35  :signals = 12'b_00_01_1_0_00_1_0_0_0; //S-TYPE(SW)
   7'd99  :signals = 12'b_01_10_0_0_00_0_0_0_1; //B-TYPE(BEQ,BNE)
   7'd111 :signals = 12'b_00_11_0_1_10_0_0_1_0; //J-TYPE(JAL)
   default:signals = 12'b_xx_xx_x_x_xx_x_x_x_x; // non-implemented instruction
    endcase
endmodule
```



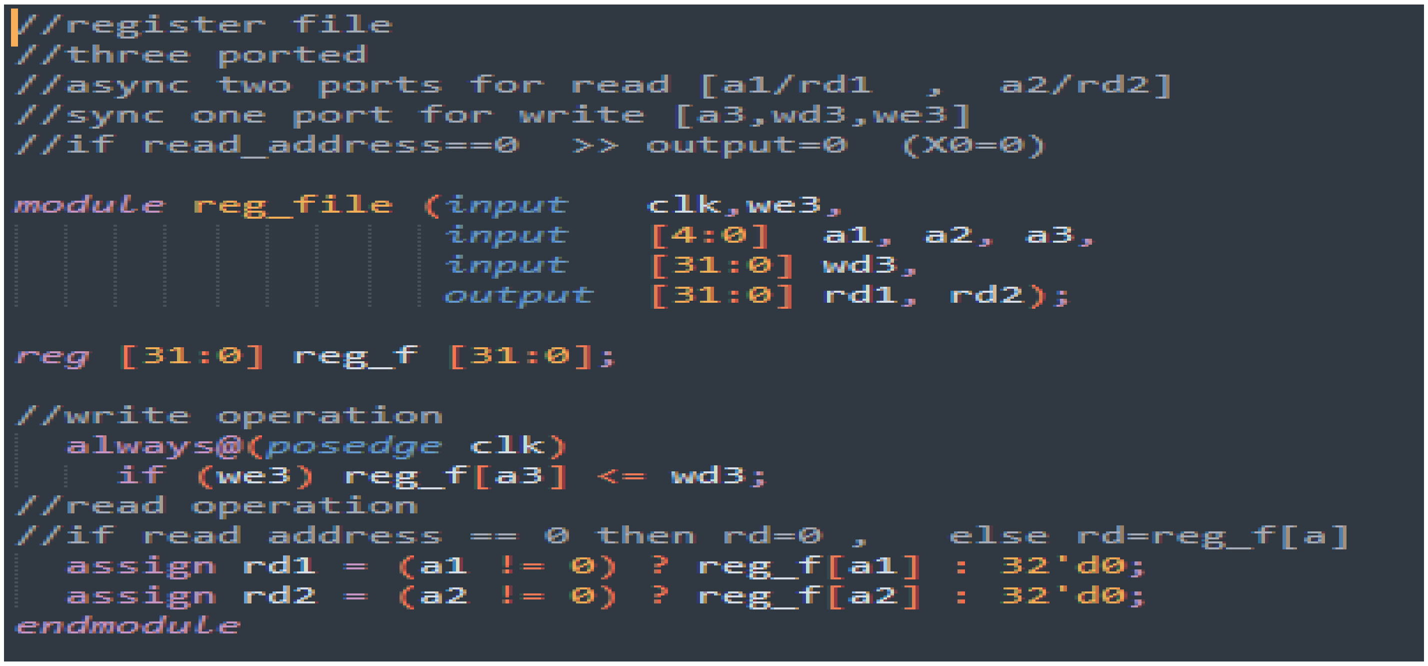
```verilog
//alu decoder
module alu_decoder(input op_b5,
            input [2:0] funct3,
            input  funct7_b5,
            input  [1:0] ALUOp,
            output reg [2:0] alu_control);

  always@(*)
    case(ALUOp)
     //I-TYPE(LW), S-TYPE(SW), I_TYPE(JALR),J-TYPE(JAL)
      //OP= 3     , 35      ,  103        , 111
      2'b00:alu_control = 3'd0;    // addition

    //B-TYPE(BEQ, BNE)
    //OP = 99
      2'b01:alu_control = 3'd1;  // subtraction

    //R-TYPE(ALL), I-TYPE(ALU)
    //OP = 51(0110011) ,19(0010011)
      2'b10: case(funct3)

                 3'b000:  if (op_b5 & funct7_b5)
                            alu_control = 3'd1;    // R-TYPE (SUB)
                          else
                            alu_control = 3'd0;    // add, addi

            3'b111:    alu_control = 3'd2;    // and, andi
            3'b110:    alu_control = 3'd3;    // or, ori
            3'b100:    alu_control = 3'd4;    // xor, xori
            3'b001:    alu_control = 3'd6;    // sll, slli
            3'b101:    alu_control = 3'd7;    // srl, srli
            3'b010:    alu_control = 3'd5;    // slt, slti
           default:    alu_control = 3'bxxx;  // ???
               endcase
    default:alu_control = 3'bxxx;  // ???
    endcase
endmodule
```



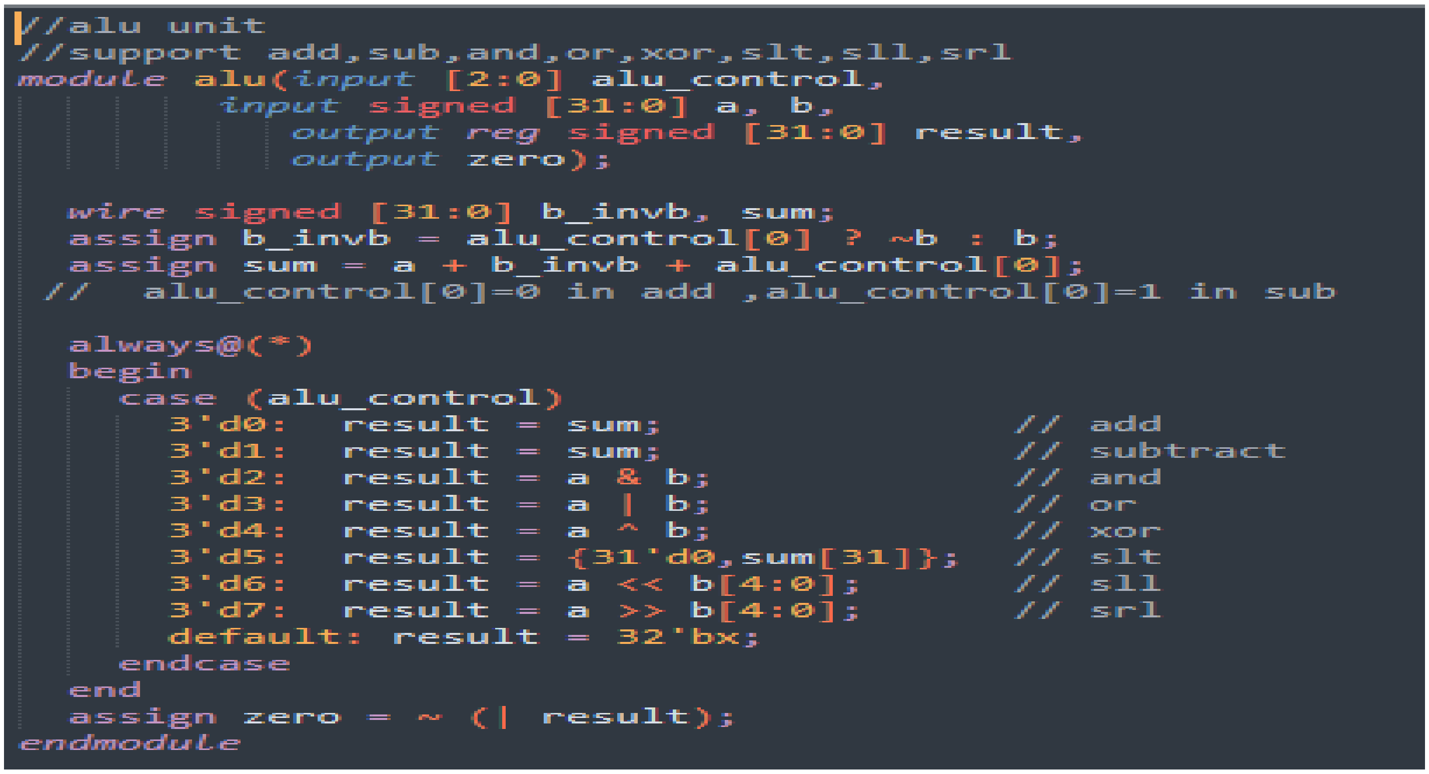
```verilog
//data_path
module data_path  (input  clk, reset,
                          output [31:0] pc,
                          input  [31:7]instr,
                          output [31:0]w_data_mem,a_data_mem,
                          input  [31:0]r_data_mem,
                          output  Zero,
                     input  [2:0] alu_control,
                          input  [1:0] ext_imm_sel,
                          input   Reg_Write,
                          input  [1:0] res_rd,
                          input   alu_src2, pc_jalr, pc_src);

wire[31:0] pc_next,pc_plus4,pc_target_addr,pc_sr1,imm_ext,wd3,rd1, rd2,alu_b,alu_result;

//data mem
assign w_data_mem=rd2;
assign a_data_mem=alu_result;

//program counter
pc_flip u1_ff1 (clk, reset, pc_next, pc);
mux2 u3_m1 (pc_plus4, pc_target_addr, pc_src, pc_next);
adder u2_add1 (pc,32'd4,pc_plus4);
adder u4_add2 (pc_sr1,imm_ext ,pc_target_addr);
mux2 u5_m2 (pc,rd1 , pc_jalr, pc_sr1);

//extend
imm_extend u6_imm1 (instr[31:7],ext_imm_sel, imm_ext);

//reg_file
reg_file u7_regf1(clk,Reg_Write, instr[19:15], instr[24:20],instr[11:7], wd3,rd1, rd2);

mux2 u8_m3 (rd2,imm_ext ,alu_src2, alu_b);

//alu
alu u9_alu1 (alu_control,rd1,alu_b,alu_result, Zero);

mux4 u10_m4(alu_result,r_data_mem,pc_plus4, ,res_rd,wd3);

endmodule
```



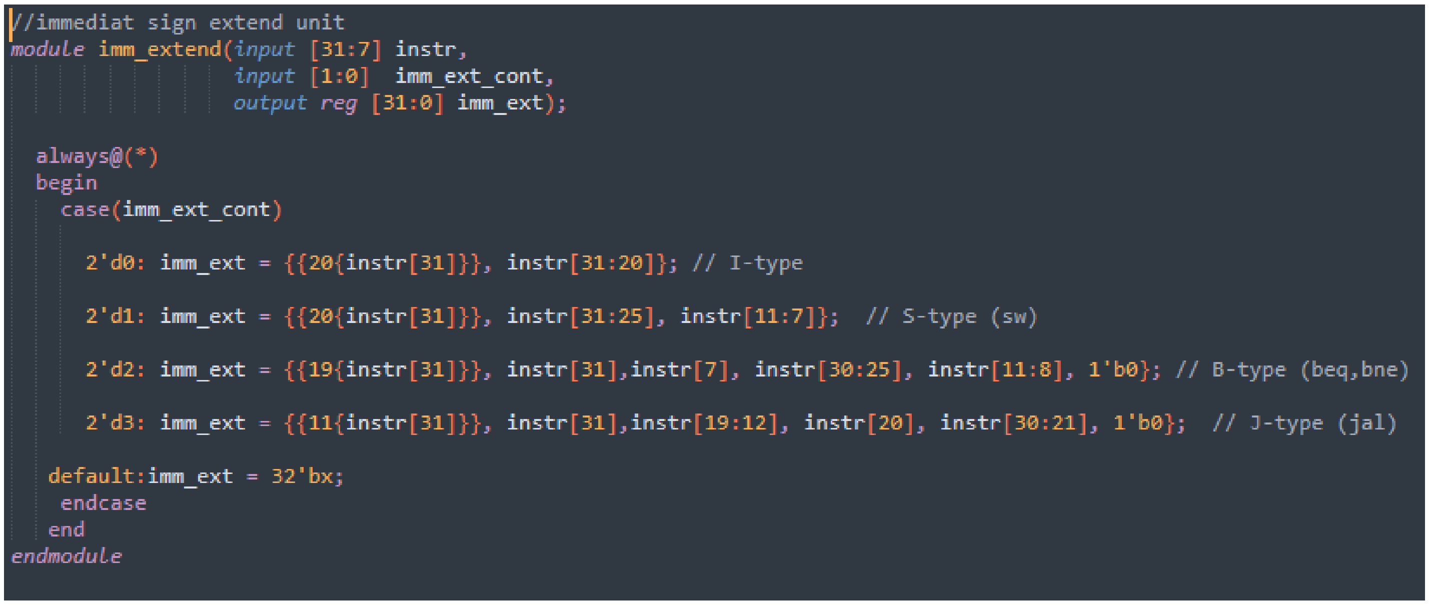
```verilog
//asyn_reset_flip_flop
module pc_flip #(parameter WIDTH = 32)
             (input clk, reset,
              input [WIDTH-1:0] d,
              output reg [WIDTH-1:0] q);

  always@(posedge clk, posedge reset)
  begin
    if (reset) q <= 0;
    else       q <= d;
  end
endmodule
```



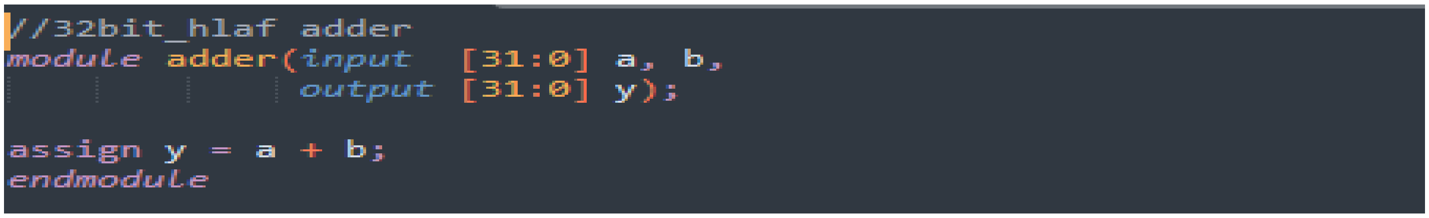
```verilog
//register file
//three ported
//async two ports for read [a1/rd1  ,  a2/rd2]
//sync one port for write [a3,wd3,we3]
//if read_address==0  >> output=0  (X0=0)

module reg_file (input    clk,we3,
                 input    [4:0]  a1, a2, a3,
                 input    [31:0] wd3,
                 output   [31:0] rd1, rd2);

reg [31:0] reg_f [31:0];

//write operation
  always@(posedge clk)
    if (we3) reg_f[a3] <= wd3;
//read operation
//if read address == 0 then rd=0 ,   else rd=reg_f[a]
  assign rd1 = (a1 != 0) ? reg_f[a1] : 32'd0;
  assign rd2 = (a2 != 0) ? reg_f[a2] : 32'd0;
endmodule
```



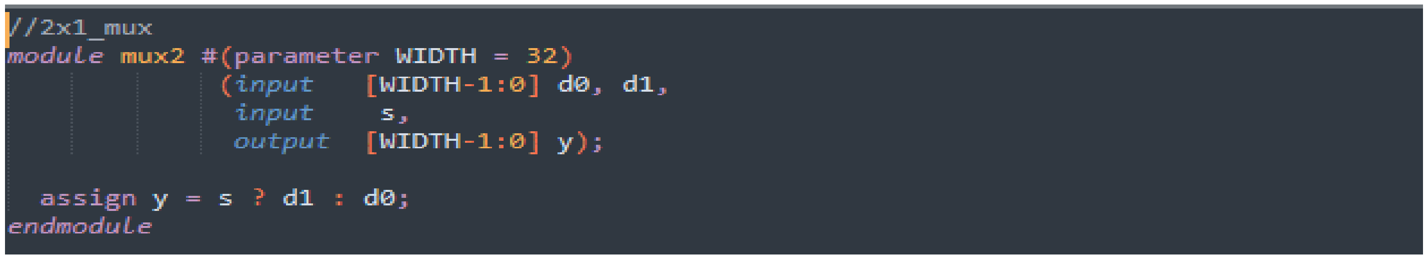
```
//alu unit
//support add,sub,and,or,xor,slt,sll,srl
module alu(input [2:0] alu_control,
           input signed [31:0] a, b,
             output reg signed [31:0] result,
             output zero);

  wire signed [31:0] b_invb, sum;
  assign b_invb = alu_control[0] ? ~b : b;
  assign sum = a + b_invb + alu_control[0];
 //  alu_control[0]=0 in add ,alu_control[0]=1 in sub

  always@(*)
  begin
    case (alu_control)
      3'd0:  result = sum;                   // add
      3'd1:  result = sum;                   // subtract
      3'd2:  result = a & b;                 // and
      3'd3:  result = a | b;                 // or
      3'd4:  result = a ^ b;                 // xor
      3'd5:  result = {31'd0,sum[31]};       // slt
      3'd6:  result = a << b[4:0];           // sll
      3'd7:  result = a >> b[4:0];           // srl
      default: result = 32'bx;
    endcase
  end
  assign zero = ~ (| result);
endmodule
```



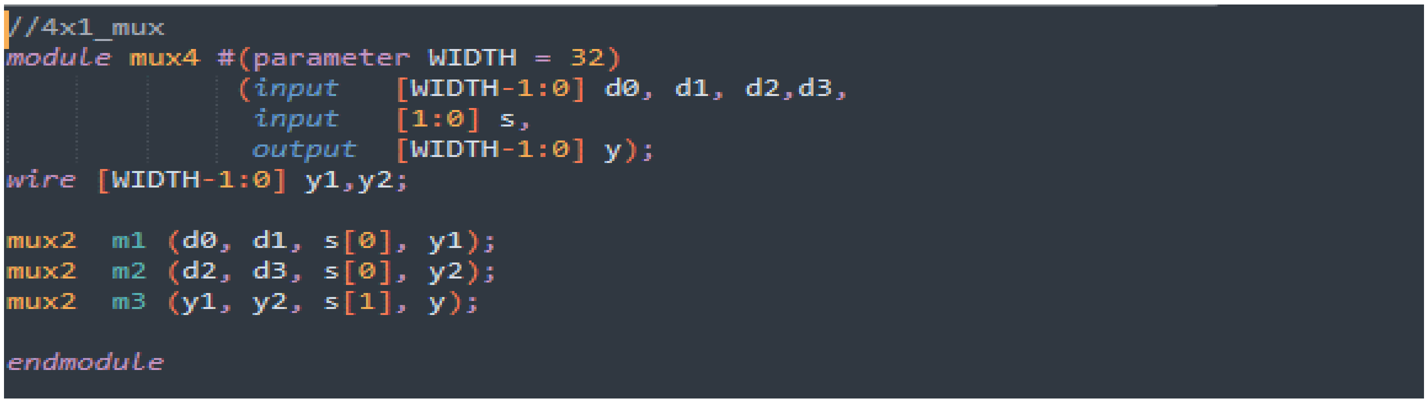
```
//immediat sign extend unit
module imm_extend(input [31:7] instr,
                  input [1:0]  imm_ext_cont,
                  output reg [31:0] imm_ext);

  always@(*)
  begin
    case(imm_ext_cont)

      2'd0: imm_ext = {{20{instr[31]}}, instr[31:20]}; // I-type

      2'd1: imm_ext = {{20{instr[31]}}, instr[31:25], instr[11:7]};  // S-type (sw)

      2'd2: imm_ext = {{19{instr[31]}}, instr[31],instr[7], instr[30:25], instr[11:8], 1'b0}; // B-type (beq,bne)

      2'd3: imm_ext = {{11{instr[31]}}, instr[31],instr[19:12], instr[20], instr[30:21], 1'b0};  // J-type (jal)

   default:imm_ext = 32'bx;
    endcase
   end
endmodule
```



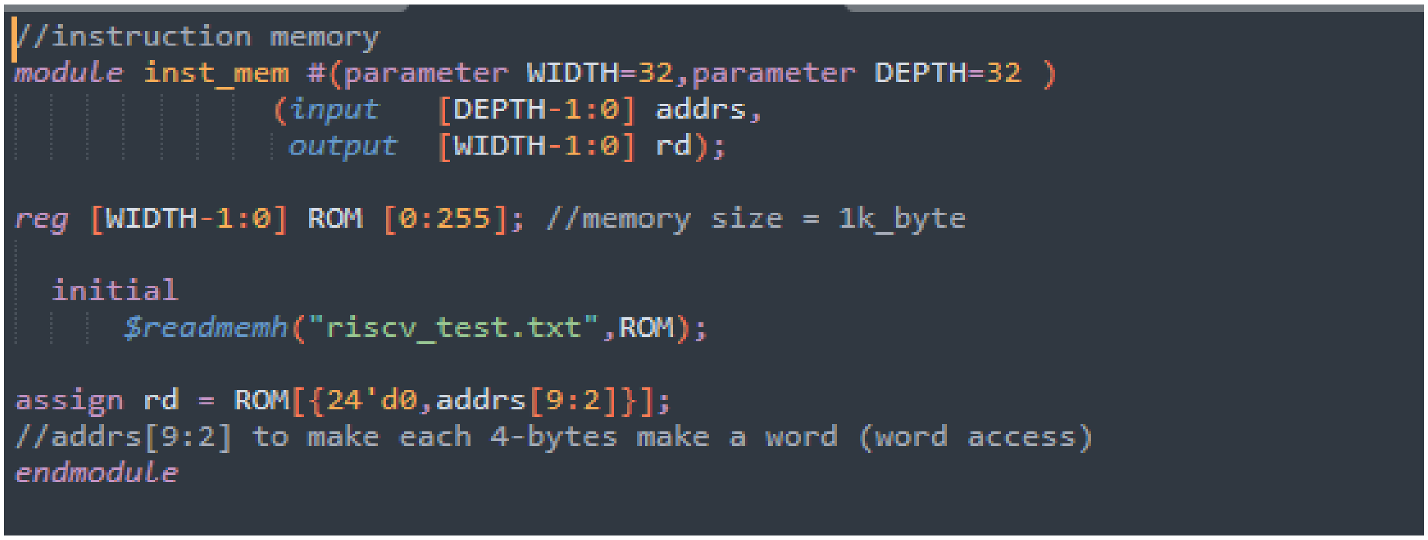
```
//32bit_hlaf adder
module adder(input  [31:0] a, b,
             output [31:0] y);

assign y = a + b;
endmodule
```



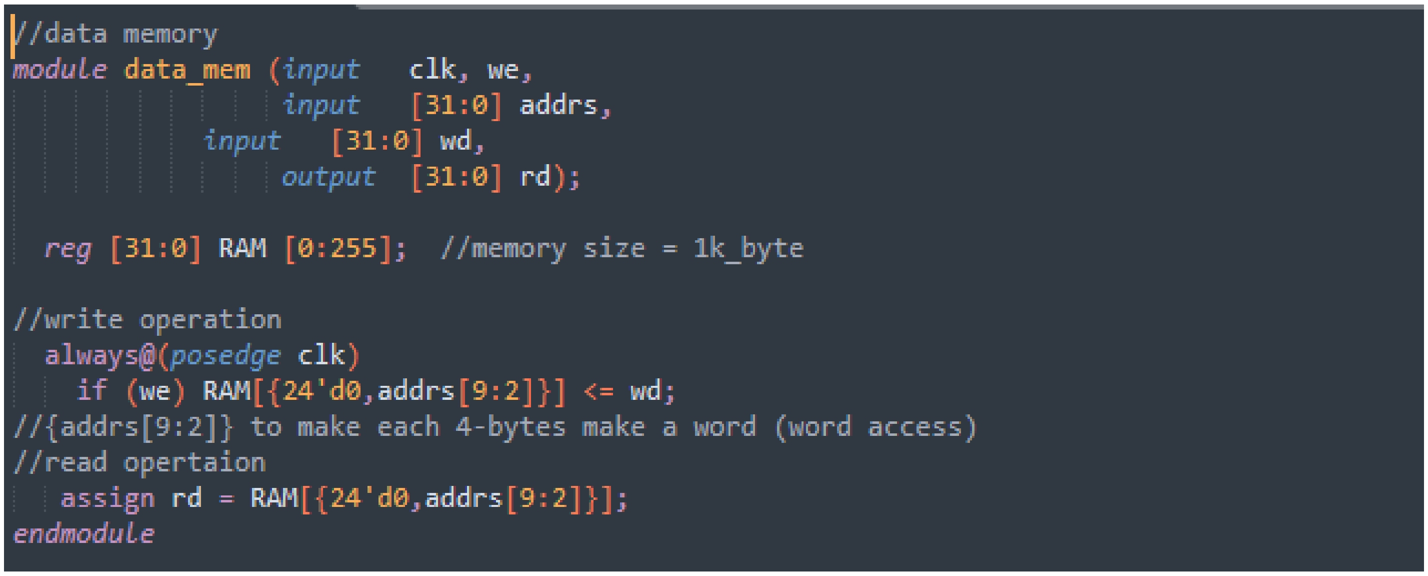
```
//2x1_mux
module mux2 #(parameter WIDTH = 32)
             (input   [WIDTH-1:0] d0, d1,
              input    s,
              output  [WIDTH-1:0] y);

  assign y = s ? d1 : d0;
endmodule
```



```verilog
//4x1_mux
module mux4 #(parameter WIDTH = 32)
             (input    [WIDTH-1:0] d0, d1, d2,d3,
              input    [1:0] s,
              output   [WIDTH-1:0] y);
wire [WIDTH-1:0] y1,y2;

mux2  m1 (d0, d1, s[0], y1);
mux2  m2 (d2, d3, s[0], y2);
mux2  m3 (y1, y2, s[1], y);

endmodule
```



```verilog
//instruction memory
module inst_mem #(parameter WIDTH=32,parameter DEPTH=32 )
              (input   [DEPTH-1:0] addrs,
               output  [WIDTH-1:0] rd);

reg [WIDTH-1:0] ROM [0:255]; //memory size = 1k_byte

  initial
     $readmemh("riscv_test.txt",ROM);

assign rd = ROM[{24'd0,addrs[9:2]}];
//addrs[9:2] to make each 4-bytes make a word (word access)
endmodule
```



```verilog
//data memory
module data_mem (input   clk, we,
                 input   [31:0] addrs,
            input   [31:0] wd,
                 output  [31:0] rd);

  reg [31:0] RAM [0:255];  //memory size = 1k_byte

//write operation
  always@(posedge clk)
    if (we) RAM[{24'd0,addrs[9:2]}] <= wd;
//{addrs[9:2]} to make each 4-bytes make a word (word access)
//read opertaion
   assign rd = RAM[{24'd0,addrs[9:2]}];
endmodule
```