Arch Lab6

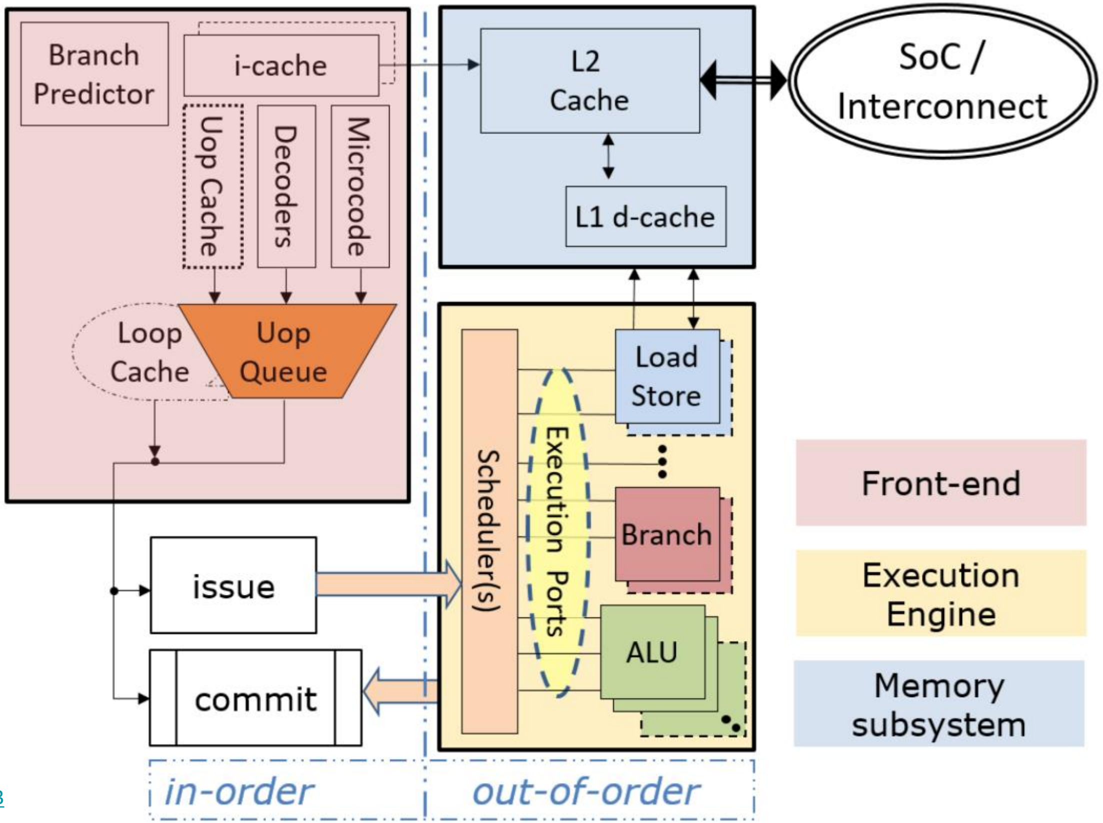
Dynamically Scheduled Pipelines using Scoreboarding

Chenlu Miao 12/2022

Tasks

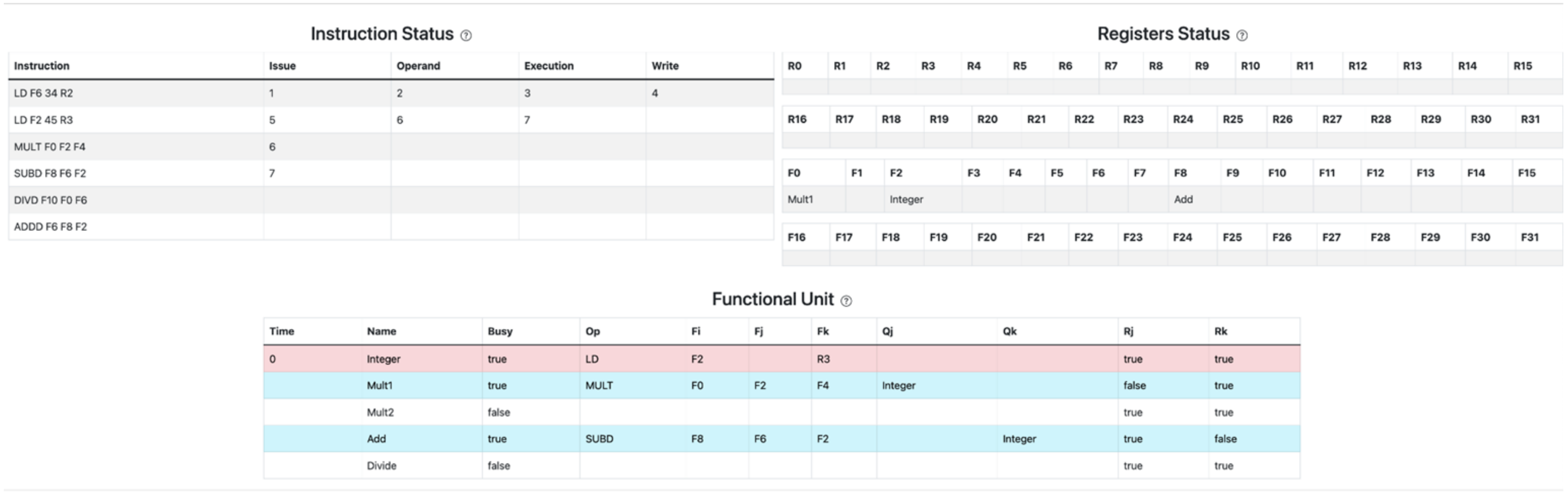
- Redesign the pipelines with IF/IS/RO/FU/WB stages and supporting multicycle operations.
- Design of a scoreboard and integrate it to CPU.

Processor Core Microarchitecture



https://dl.acm.org/doi/pdf/10.1145/3369383

Scoreboard Overview



Instruction Status

Instruction	Issue	Operand	Execution	Write
LD F6 34 R2	1	2	3	4
LD F2 45 R3	5	6	7	
MULT F0 F2 F4	6			
SUBD F8 F6 F2	7			
DIVD F10 F0 F6				
ADDD F6 F8 F2				

Registers Status

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15

R16	R17	R18	R19	R20	R21	R22	R23	R24	R25	R26	R27	R28	R29	R30	R31

F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15
Mult1		Integer						Add							

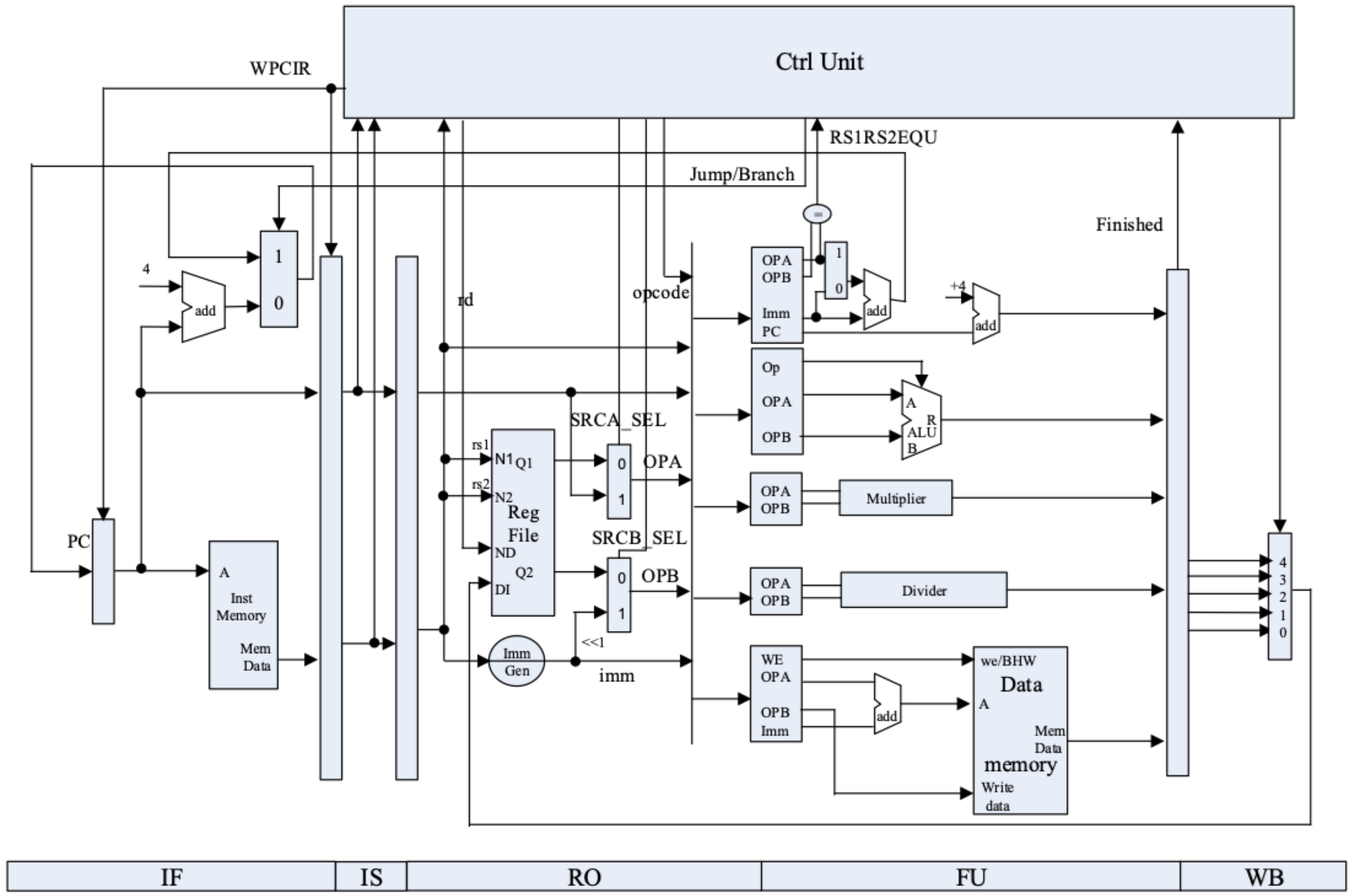
F16	F17	F18	F19	F20	F21	F22	F23	F24	F25	F26	F27	F28	F29	F30	F31

Functional Unit

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
0	Integer	true	LD	F2		R3			true	true
	Mult1	true	MULT	F0	F2	F4	Integer		false	true
	Mult2	false							true	true
	Add	true	SUBD	F8	F6	F2		Integer	true	false
	Divide	false							true	true

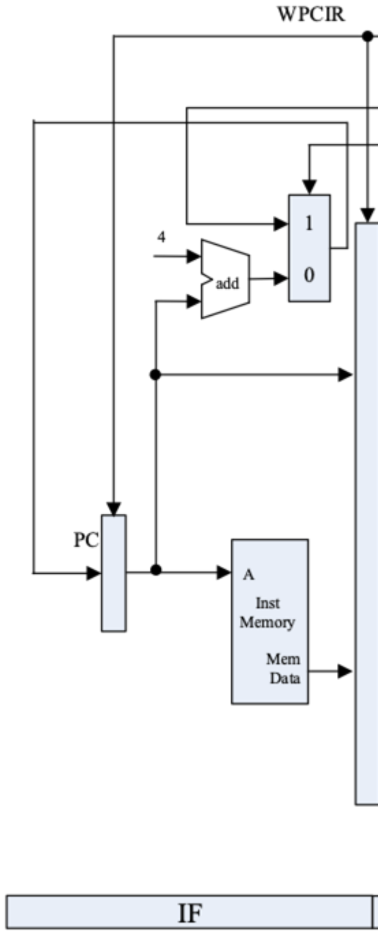
https://jasonren0403.github.io/scoreboard/

Architecture Overview



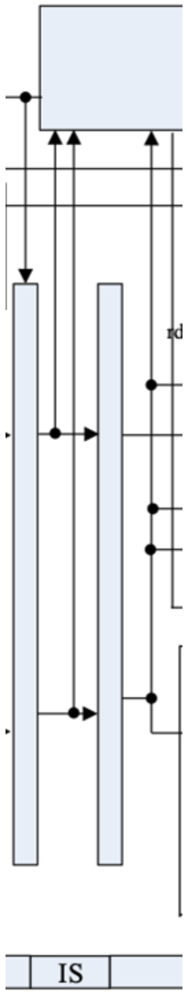
Architecture Overview – IF

```
// IF
assign PC_EN_IF = IS_EN | FU_jump_finish & is_jump_FU;

REG32
REG_PC(.clk(debug_clk),.rst(rst),.CE(PC_EN_IF),.D(next_PC_IF),.Q(PC_IF));

add_32 add_IF(.a(PC_IF),.b(32'd4),.c(PC_4_IF));

MUX2T1_32 mux_IF(.I0(PC_4_IF),.I1(PC_jump_FU),.s(FU_jump_finish &
is_jump_FU),.o(next_PC_IF));

ROM_D inst_rom(.a(PC_IF[8:2]),.spo(inst_IF));
```



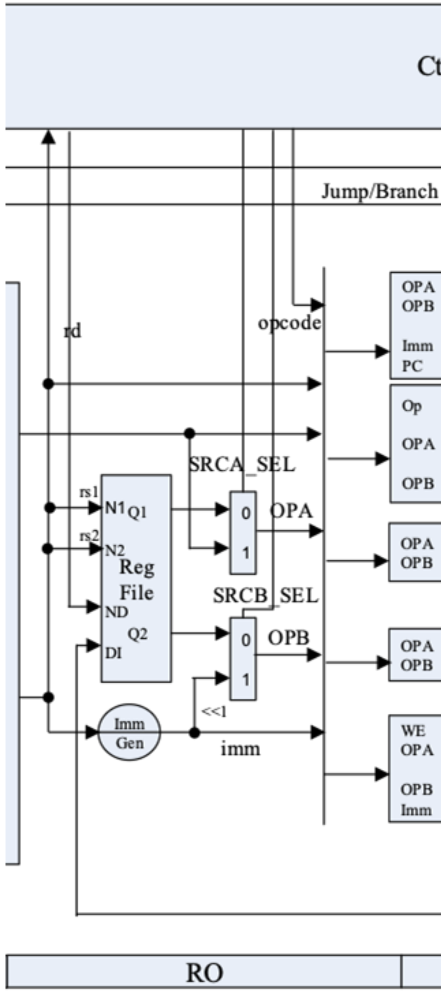
Architecture Overview – IS

```
//Issue
REG_IF_IS reg_IF_IS(.clk(debug_clk),.rst(rst),.EN(IS_EN),
   .flush(1'b0),.PCOUT(PC_IF),.IR(inst_IF),

   .IR_IS(inst_IS),.PCurrent_IS(PC_IS));

ImmGen imm_gen(.ImmSel(ImmSel_ctrl),.inst_field(inst_IS),.Imm_out(Imm_out_IS));

CtrlUnit
ctrl(.clk(debug_clk),.rst(rst),.PC(PC_IS),.inst(inst_IS),.imm(Imm_out_IS), .ALU_done(FU_ALU_finish)
,.MEM_done(FU_mem_finish),.MUL_done(FU_mul_finish),.DIV_done(FU_div_finish),.JUMP_done(FU_jump_fini
sh),.is_jump(is_jump_FU),
   .IS_en(IS_EN),.ImmSel(ImmSel_ctrl),.ALU_en(FU_ALU_EN),
   .MEM_en(FU_mem_EN),.MUL_en(FU_mul_EN),.DIV_en(FU_div_EN),.JUMP_en(FU_jump_EN),
   .PC_ctrl(PC_ctrl),.imm_ctrl(imm_ctrl),.rs1_ctrl(rs1_addr_ctrl),.rs2_ctrl(rs2_addr_ctrl), .JUMP_o
p(Jump_ctrl),.ALU_op(ALUControl_ctrl),.ALU_use_PC(ALUSrcA_ctrl),.ALU_use_imm(ALUSrcB_ctrl),
   .MEM_we(mem_w_ctrl),.MEM_bhw(bhw_ctrl),.MUL_op(),.DIV_op(),
   .write_sel(DatatoReg_ctrl),.reg_write(RegWrite_ctrl),.rd_ctrl(rd_ctrl)
);
```



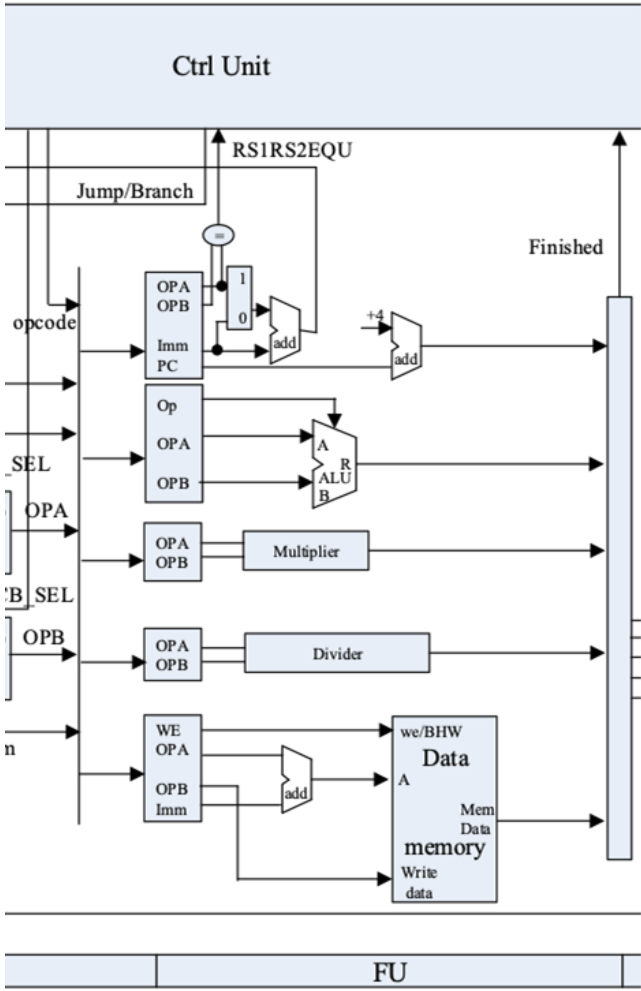
Architecture Overview – RO

```
// RO
Regs register(.clk(debug_clk),.rst(rst),
    .R_addr_A(rs1_addr_ctrl),.rdata_A(rs1_data_RO),
    .R_addr_B(rs2_addr_ctrl),.rdata_B(rs2_data_RO),
    .L_S(RegWrite_ctrl),.Wt_addr(rd_ctrl),.Wt_data(wt_data_WB),
    .Debug_addr(debug_addr[4:0]),.Debug_regs(debug_regs));

MUX2T1_32
mux_imm_ALU_RO_A(.I0(rs1_data_RO),.I1(PC_ctrl),.s(ALUSrcA_ctrl),
.o(ALUA_RO));

MUX2T1_32
mux_imm_ALU_RO_B(.I0(rs2_data_RO),.I1(imm_ctrl),.s(ALUSrcB_ctrl)
,.o(ALUB_RO));
```



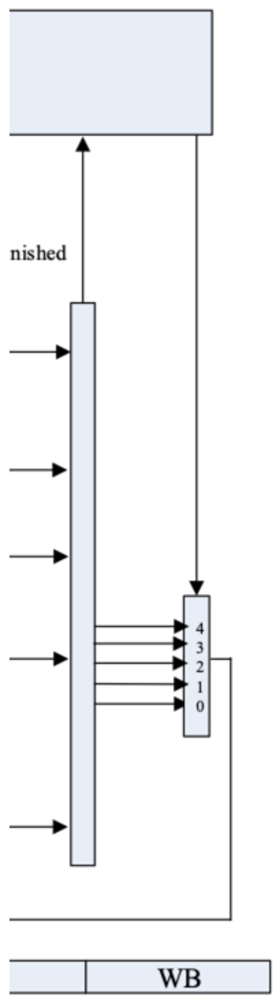
Architecture Overview – FU

```
// FU
FU_ALU alu(...);

FU_mem mem(...);

FU_mul mu(...);

FU_div du(...);

FU_jump ju(...);
```

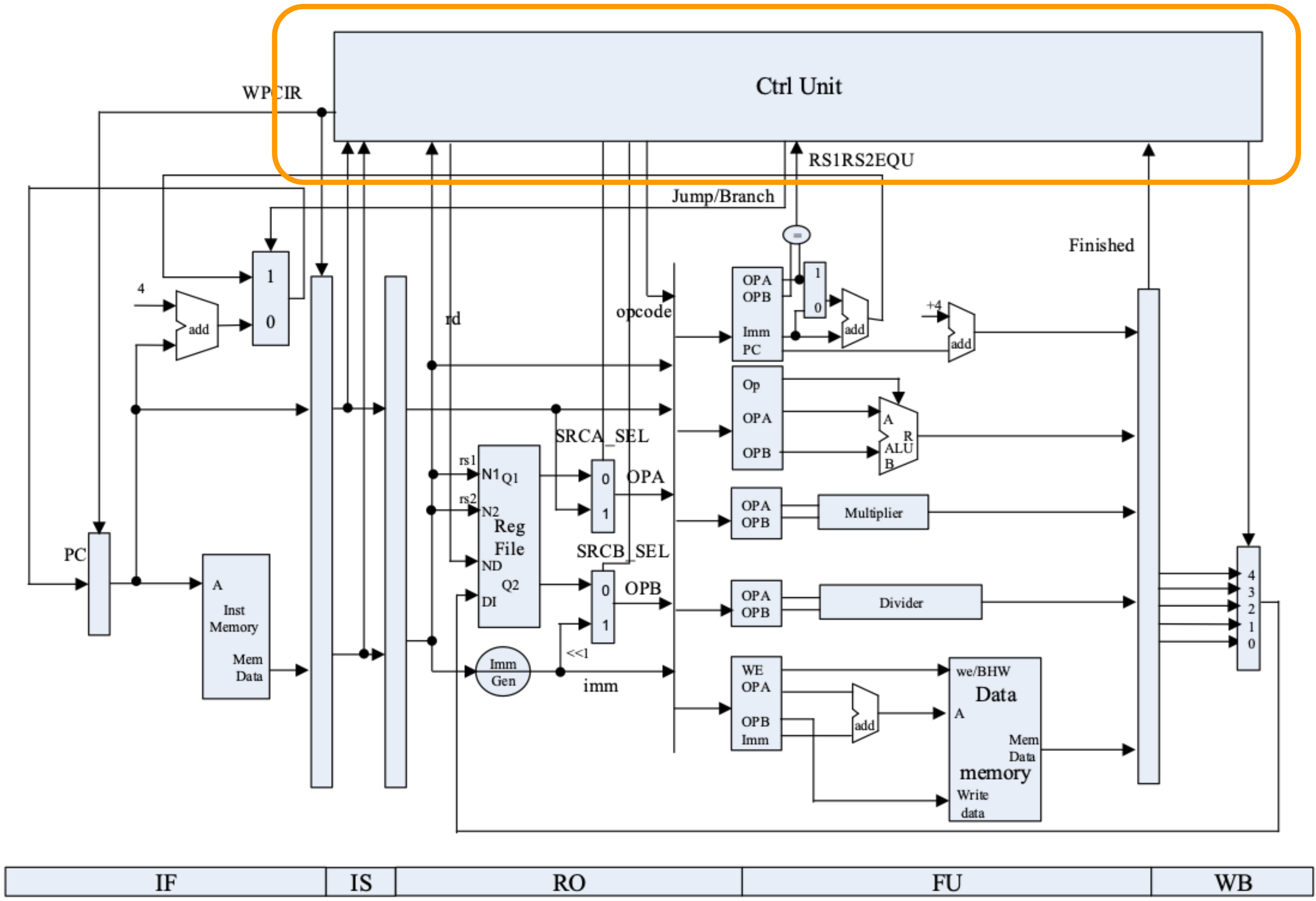


Architecture Overview – WB

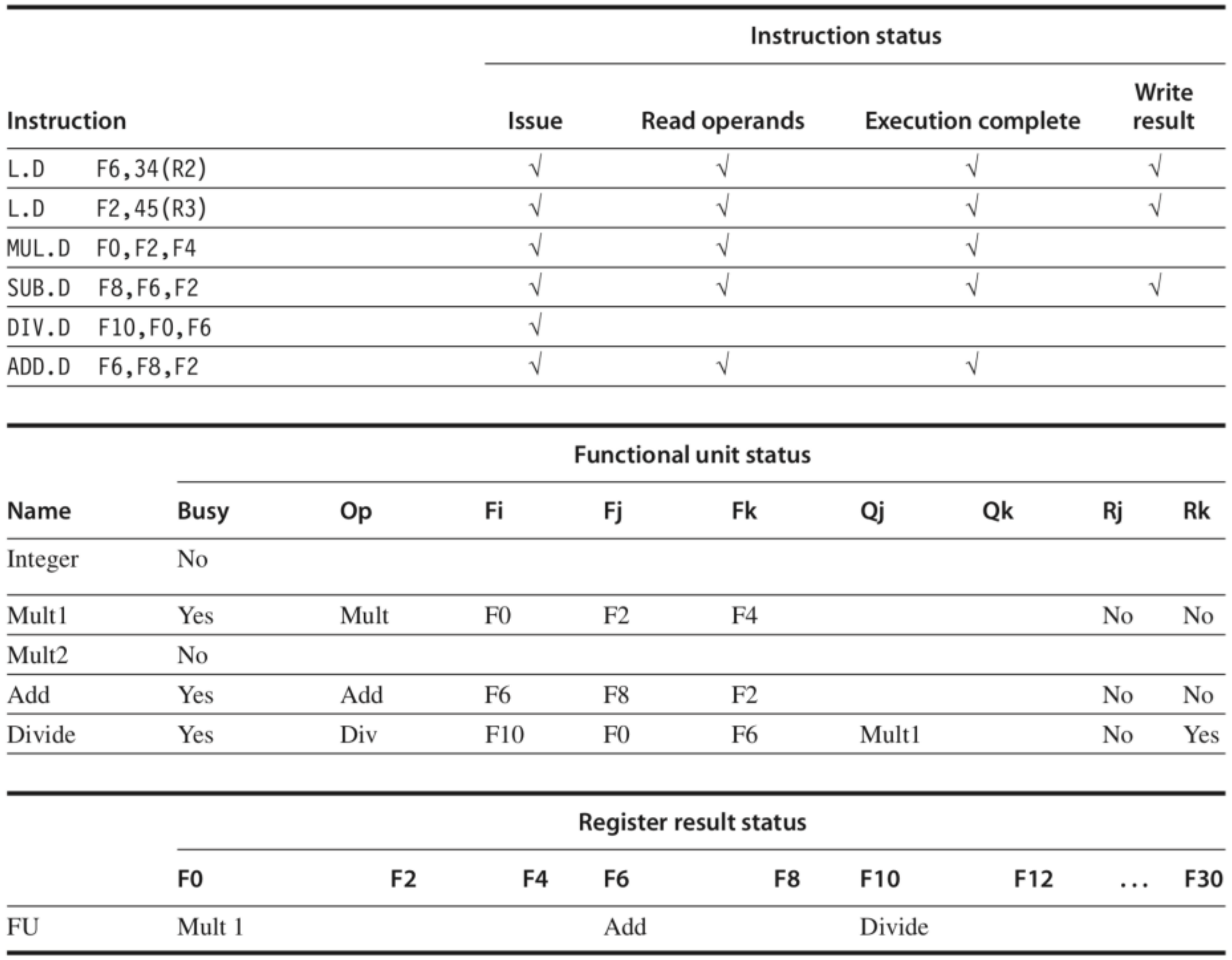
```
// WB
REG32 reg_WB_ALU(.clk(debug_clk),.rst(rst),.CE(FU_ALU_finish),.D(ALUout_FU),.Q(ALUout_WB));

REG32
reg_WB_mem(.clk(debug_clk),.rst(rst),.CE(FU_mem_finish),.D(mem_data_FU),.Q(mem_data_WB));

REG32 reg_WB_mul(.clk(debug_clk),.rst(rst),.CE(FU_mul_finish),.D(mulres_FU),.Q(mulres_WB));

REG32 reg_WB_div(.clk(debug_clk),.rst(rst),.CE(FU_div_finish),.D(divres_FU),.Q(divres_WB));

REG32 reg_WB_jump(.clk(debug_clk),.rst(rst),.CE(FU_jump_finish),.D(PC_wb_FU),.Q(PC_wb_WB));

MUX8T1_32
mux_DtR(.s(DatatoReg_ctrl),.I0(ALUout_WB),.I1(mem_data_WB),.I2(mulres_WB),.I3(divres_WB),
   .I4(PC_wb_WB),.I5(32'd0),.I6(32'd0),.I7(32'd0),.o(wt_data_WB));
```



Ctrl Unit


Ctrl Unit
WPCIR
RS1RS2EQU
Jump/Branch
Finished
4
add
1
0
PC
A
Inst
Memory
Mem
Data
rd
opcode
rs1
rs2
N1
Q1
N2
Reg
File
ND
Q2
DI
SRCA_SEL
OPA
SRCB_SEL
OPB
<<1
Imm
Gen
imm
OPA
OPB
Imm
PC
=
+4
Op
ALU
R
B
Multiplier
Divider
WE
we/BHW
Data
memory
Write
data
4
3
2
1
0
IF
IS
RO
FU
WB

Ctrl Unit

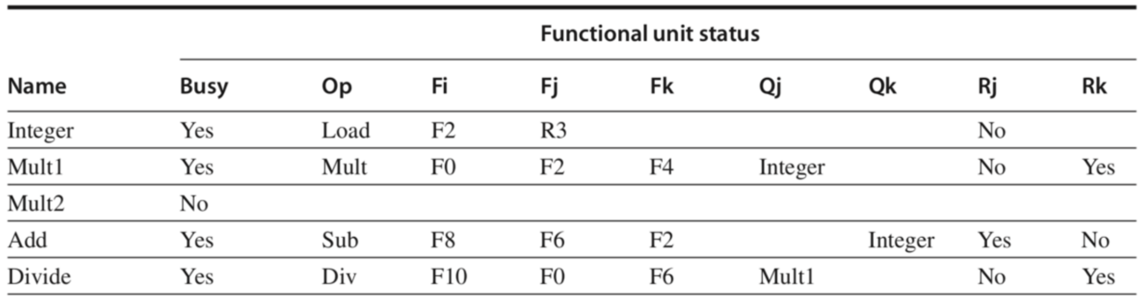


Instruction	Instruction status			
	Issue	Read operands	Execution complete	Write result
`L.D F6,34(R2)`	√	√	√	√
`L.D F2,45(R3)`	√	√	√	√
`MUL.D F0,F2,F4`	√	√	√	
`SUB.D F8,F6,F2`	√	√	√	√
`DIV.D F10,F0,F6`	√			
`ADD.D F6,F8,F2`	√	√	√	

	Functional unit status								
Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	No								
Mult1	Yes	Mult	F0	F2	F4			No	No
Mult2	No								
Add	Yes	Add	F6	F8	F2			No	No
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

	Register result status								
	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mult 1			Add		Divide			

Ctrl Unit – Function Unit Status



	Functional unit status								
Name	**Busy**	**Op**	**Fi**	**Fj**	**Fk**	**Qj**	**Qk**	**Rj**	**Rk**
Integer	Yes	Load	F2	R3				No	
Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
Mult2	No								
Add	Yes	Sub	F8	F6	F2		Integer	Yes	No
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Busy – Indicate whether the unit is busy or not.

Op – Operation to perform in the unit (e.g., addr or subtract).

Fi – Destination register.

Fj, Fk – Source-register numbers.

Qj, Qk – Functional units producing source registers Fj, Fk

Rj, Rk – Flags indicating when Fj, Fk are ready and not yet read. Set to No after operands.

+ FU_DONE

Ctrl Unit – Function Unit Status

Busy – Indicate whether the unit is busy or not.
Op – Operation to perform in the unit
Fi – Destination register.
Fj, Fk – Source-register numbers.
Qj, Qk – Functional units producing source registers Fj, Fk
Rj, Rk – Flags indicating when Fj, Fk are ready and not yet read.
Set to No after operands.
+ FU_DONE

```
reg[31:0] FUS[1:5];
```

```
e.g. FUS[`FU_MEM][`SRC1_H:`SRC1_L]
```

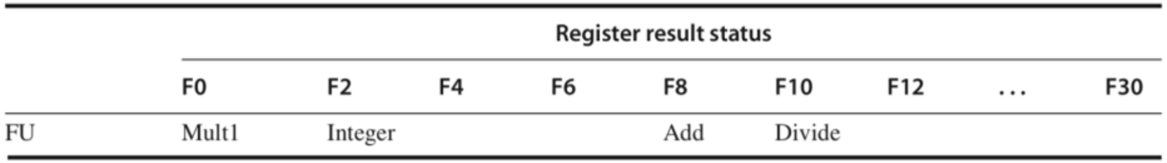
```
// function unit
`define FU_BLANK    3'd0
`define FU_ALU      3'd1
`define FU_MEM      3'd2
`define FU_MUL      3'd3
`define FU_DIV      3'd4
`define FU_JUMP     3'd5
```

```
// bits in FUS
`define BUSY    0
`define OP_L    1
`define OP_H    5
`define DST_L   6
`define DST_H   10
`define SRC1_L  11
`define SRC1_H  15
`define SRC2_L  16
`define SRC2_H  20
`define FU1_L   21
`define FU1_H   23
`define FU2_L   24
`define FU2_H   26
`define RDY1    27
`define RDY2    28
`define FU_DONE 29
```

Ctrl Unit – Register Result Status

```
// function unit
`define FU_BLANK     3'd0
`define FU_ALU       3'd1
`define FU_MEM       3'd2
`define FU_MUL       3'd3
`define FU_DIV       3'd4
`define FU_JUMP      3'd5
```

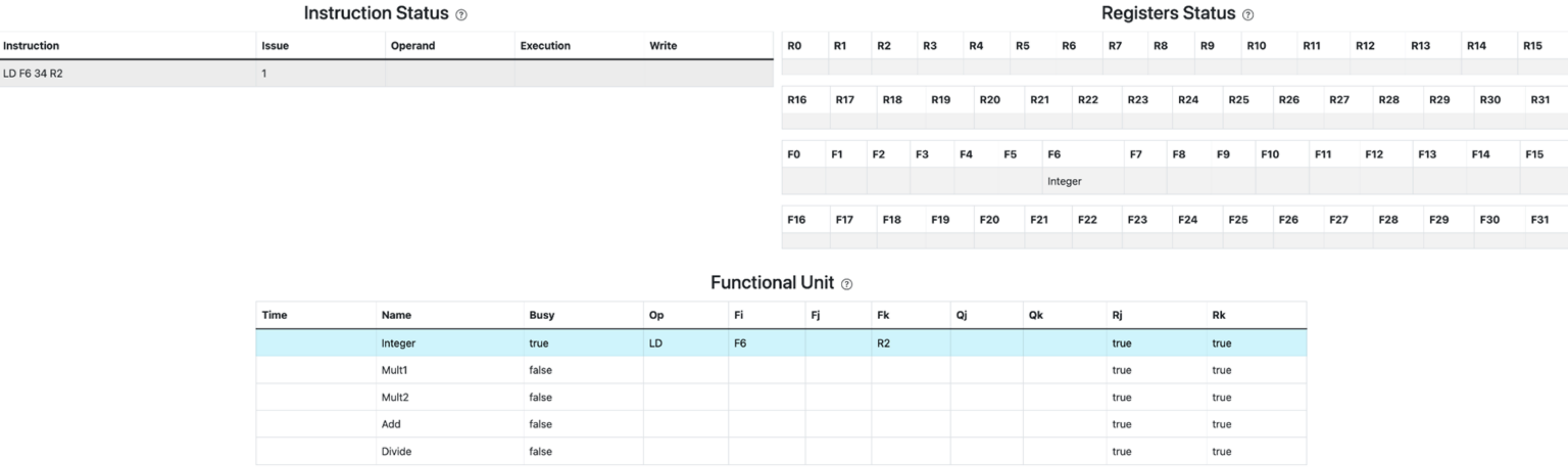
Indicates which functional unit will write each register, if an active instruction has the register as its destination. This field is set to blank whenever there are no pending instructions that will write that register.



	Register result status							
	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mult1	Integer			Add	Divide			

```
// records which FU will write corresponding reg at
WB
reg[2:0] RRS[0:31];
```

ISSUE



Instruction Status

Instruction	Issue	Operand	Execution	Write
LD F6 34 R2	1			

Registers Status

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15

R16	R17	R18	R19	R20	R21	R22	R23	R24	R25	R26	R27	R28	R29	R30	R31

F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15
						Integer									

F16	F17	F18	F19	F20	F21	F22	F23	F24	F25	F26	F27	F28	F29	F30	F31

Functional Unit

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	true	LD	F6		R2			true	true
	Mult1	false							true	true
	Mult2	false							true	true
	Add	false							true	true
	Divide	false							true	true

ISSUE

```
// normal stall: structural hazard or WAW
assign normal_stall = ...;

assign IS_en = IS_flush | ~normal_stall & ~ctrl_stall;
assign RO_en = ~IS_flush & ~normal_stall & ~ctrl_stall;

always @ (posedge clk or posedge rst) begin
   if (rst) begin
       ctrl_stall <= 0;
   end
   else begin
       // IS
       if (RO_en & (use_FU == `FU_JUMP)) begin
           ctrl_stall <= 1;
       end
       else if (JUMP_done) begin
           ctrl_stall <= 0;
       end
   end
end
```

```
// IS
if (RO_en) begin
   // not busy, no WAW, write info to FUS and RRS
   if (|dst)
     RRS[dst] <= use_FU;


   FUS[use_FU][`BUSY] <= 1'b1;
   FUS[use_FU][`OP_H:`OP_L] <= ...
   FUS[use_FU][`DST_H:`DST_L] <= ...
   FUS[use_FU][`SRC1_H:`SRC1_L] <= ...
   FUS[use_FU][`SRC2_H:`SRC2_L] <= ...
   FUS[use_FU][`FU1_H:`FU1_L] <= ...
   FUS[use_FU][`FU2_H:`FU2_L] <= ...
   FUS[use_FU][`RDY1] <= ...
   FUS[use_FU][`RDY2] <= ...
   FUS[use_FU][`FU_DONE] <= ...


   IMM[use_FU] <= imm;
   PCR[use_FU] <= PC;
end
```

Read Operands

```
// JUMP
if (FUS[`FU_JUMP][`RDY1] & FUS[`FU_JUMP][`RDY2])
begin
   ALU_en = 1'b0;
   MEM_en = 1'b0;
   MUL_en = 1'b0;
   DIV_en = 1'b0;
   JUMP_en = 1'b1;

   JUMP_op = FUS[`FU_JUMP][`OP_H:`OP_L];
   rs1_ctrl = FUS[`FU_JUMP][`SRC1_H:`SRC1_L];
   rs2_ctrl = FUS[`FU_JUMP][`SRC2_H:`SRC2_L];
   PC_ctrl = PCR[`FU_JUMP];
   imm_ctrl = IMM[`FU_JUMP];
end
```

```
// ALU
...
// MEM
...
// MUL
…
// DIV
```

Read Operands

```
// RO
if (FUS[`FU_JUMP][`RDY1] & FUS[`FU_JUMP][`RDY2]) begin
   // JUMP
   FUS[`FU_JUMP][`RDY1] <= 1'b0;
   FUS[`FU_JUMP][`RDY2] <= 1'b0;
end
else if (...) begin                  //fill sth. here.
   // ALU
   ...                              //fill sth. here.
end
else if (..) begin                  //fill sth. here.
   // MEM
   ...                              //fill sth. here.
end
else if (...) begin                  //fill sth. here.
   // MUL
   ...                              //fill sth. here.
end
else if (..) begin                  //fill sth. here.
   // DIV
   ...                              //fill sth. here.
end
```

Execute

```
// EX
FUS[`FU_ALU][`FU_DONE] <= ...    //fill sth. here
...                              //fill sth. here
```

Write Back

```
// WB
always @ (*) begin
   write_sel = 0;
   reg_write = 0;
   rd_ctrl = 0;

   if (FUS[`FU_JUMP][`FU_DONE] & JUMP_WAR) begin
       write_sel = 3'd4;
       reg_write = 1'b1;
       rd_ctrl = FUS[`FU_JUMP][`DST_H:`DST_L];
   end
   else if (FUS[`FU_ALU][`FU_DONE] & ALU_WAR) begin
       write_sel = 3'd0;
       reg_write = 1'b1;
       rd_ctrl = FUS[`FU_ALU][`DST_H:`DST_L];
   end
   ...
end
```

```
// WB
if (FUS[`FU_JUMP][`FU_DONE] & JUMP_WAR) begin
   FUS[`FU_JUMP] <= 32'b0;
   RRS[FUS[`FU_JUMP][`DST_H:`DST_L]] <= 3'b0;

   // ensure RAW
   ...
end
// ALU
   ...;            //fill sth. here
// MEM
   ...;            //fill sth. here
// MUL
   ...;            //fill sth. here
// DIV
   ...;            //fill sth. here
```

R
O
M

NO.	Instruction	Addr.	Label	ASM	Comment
0	00000013	0	__start:	addi x0, x0, 0	
1	00402103	4		lw x2, 4(x0)	
2	00802203	8		lw x4, 8(x0)	Structural Hazard
3	004100b3	C		add x1, x2, x4	
4	fff08093	10		addi x1, x1, -1	WAW
5	00c02283	14		lw x5, 12(x0)	
6	01002303	18		lw x6, 16(x0)	
7	01402383	1C		lw x7, 20(x0)	
8	402200b3	20		sub x1,x4,x2	
9	ffd50093	24		addi x1,x10,-3	
10	00520c63	28		beq x4,x5,label0	
11	00420a63	2C		beq x4,x4,label0	
12	00000013	30		addi x0,x0,0	
13	00000013	34		addi x0,x0,0	
14	00000013	38		addi x0,x0,0	

R
O
M

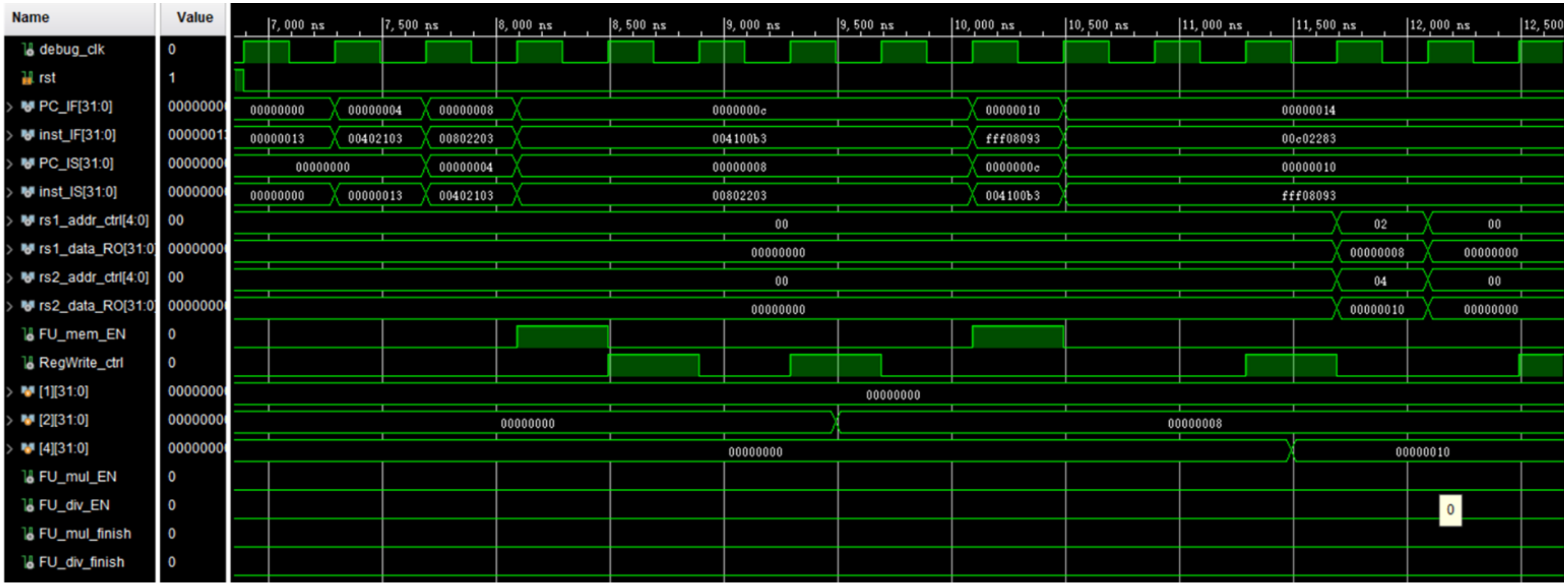
NO.	Instruction	Addr.	Label	ASM	Comment
15	00000013	3C		addi x0,x0,0	
16	000040b7	40	label0:	lui x1,4	
17	00c000ef	44		jal x1,12	
18	00000013	48		addi x0,x0,0	
19	00000013	4C		addi x0,x0,0	
20	ffff0097	50		auipc x1, 0xffff0	
21	0223c433	54		div x8, x7, x2	
22	025204b3	58		mul x9, x4, x5	St. Ha./RAW/WAW
23	022404b3	5C		mul x9, x8, x2	WAR
24	00400113	60		addi x2, x0, 4	
25	000000e7	64		jalr x1,0(x0)	
26	00000013	68		addi x0,x0,0	
27	00000013	6C		addi x0,x0,0	

R A M

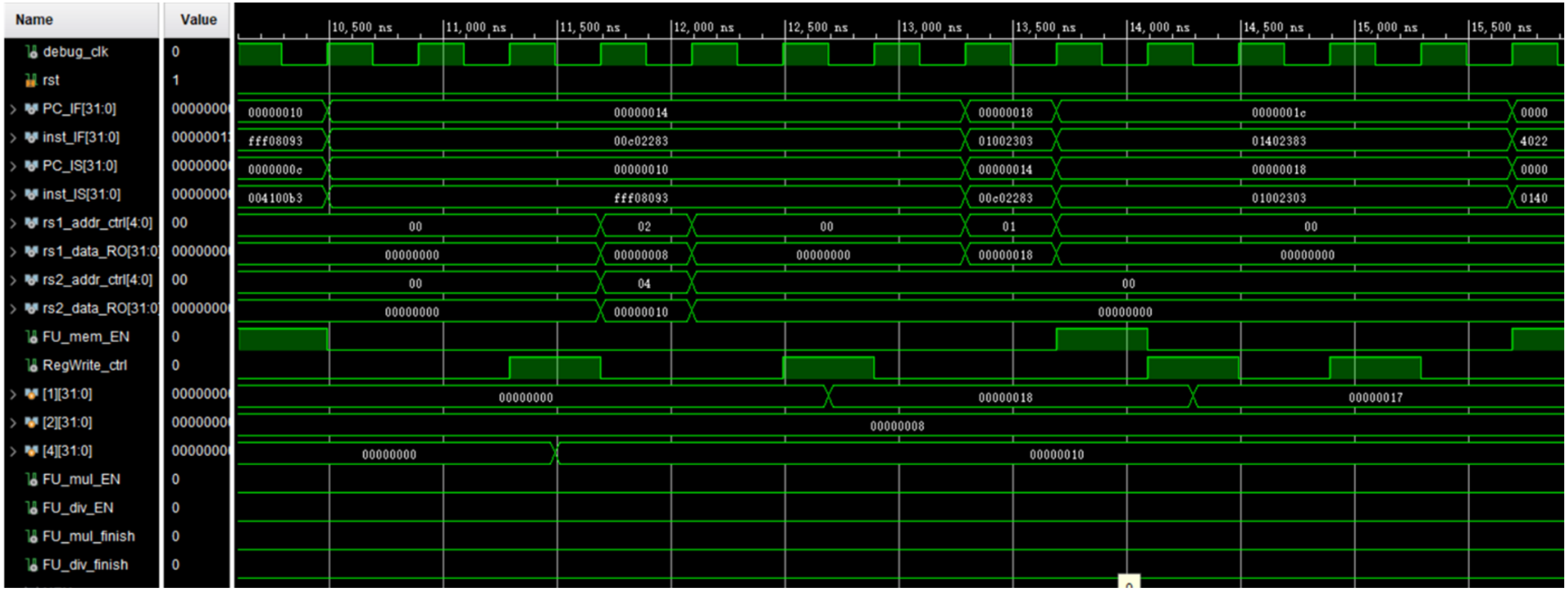
NO.	Data	Addr.
0	000080BF	0
1	00000008	4
2	00000010	8
3	00000014	C
4	FFFF0000	10
5	0FFF0000	14
6	FF000F0F	18
7	F0F0F0F0	1C
8	00000000	20
9	00000000	24
10	00000000	28
11	00000000	2C
12	00000000	30
13	00000000	34
14	00000000	38
15	00000000	3C

NO.	Instruction	Addr.
16	00000000	40
17	00000000	44
18	00000000	48
19	00000000	4C
20	A3000000	50
21	27000000	54
22	79000000	58
23	15100000	5C
24	00000000	60
25	00000000	64
26	00000000	68
27	00000000	6C
28	00000000	70
29	00000000	74
30	00000000	78
31	00000000	7C

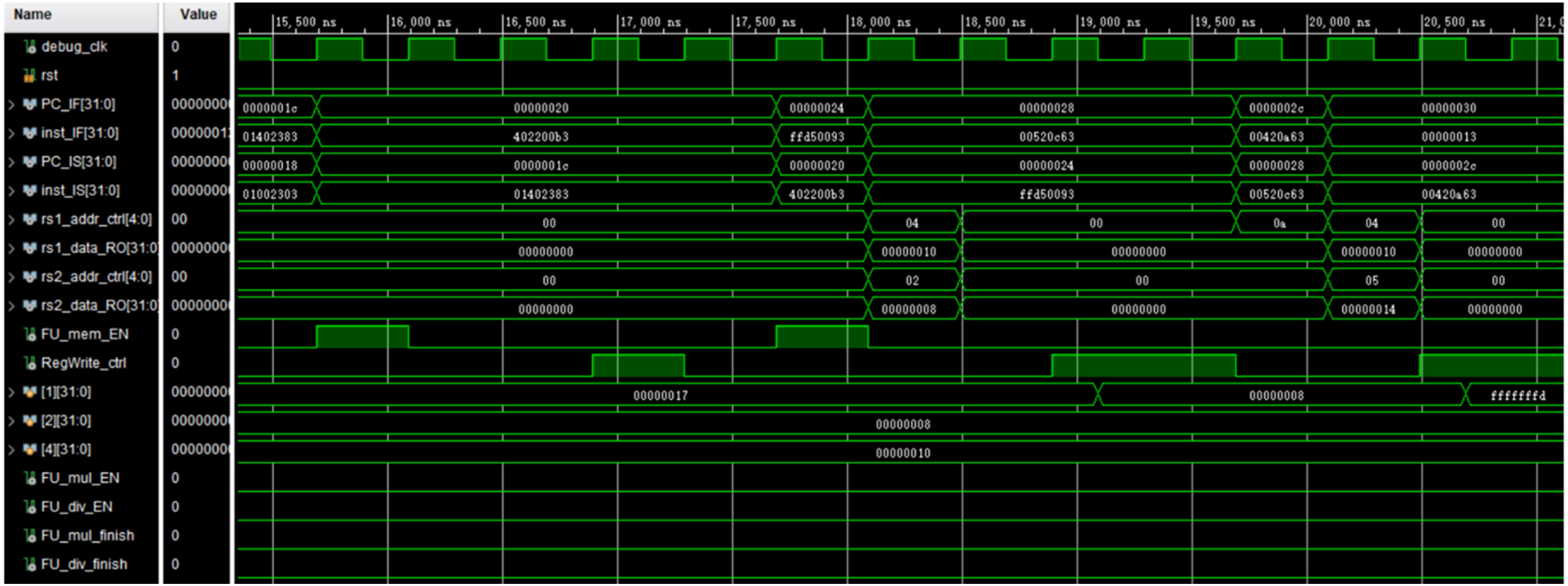
Simulation


Name
Value
7,000 ns
7,500 ns
8,000 ns
8,500 ns
9,000 ns
9,500 ns
10,000 ns
10,500 ns
11,000 ns
11,500 ns
12,000 ns
12,500
debug_clk 0
rst 1
PC_IF[31:0] 0000000 | 00000000 | 00000004 | 00000008 | 0000000c | 00000010 | 00000014
inst_IF[31:0] 0000001 | 00000013 | 00402103 | 00802203 | 004100b3 | fff08093 | 00c02283
PC_IS[31:0] 0000000 | 00000000 | 00000004 | 00000008 | 0000000c | 00000010
inst_IS[31:0] 0000000 | 00000000 | 00000013 | 00402103 | 00802203 | 004100b3 | fff08093
rs1_addr_ctrl[4:0] 00 | 00 | 02 | 00
rs1_data_RO[31:0] 0000000 | 00000000 | 00000008 | 00000000
rs2_addr_ctrl[4:0] 00 | 00 | 04 | 00
rs2_data_RO[31:0] 0000000 | 00000000 | 00000010 | 00000000
FU_mem_EN 0
RegWrite_ctrl 0
[1][31:0] 0000000 | 00000000
[2][31:0] 0000000 | 00000000 | 00000008
[4][31:0] 0000000 | 00000000 | 00000010
FU_mul_EN 0
FU_div_EN 0
FU_mul_finish 0
FU_div_finish 0
0

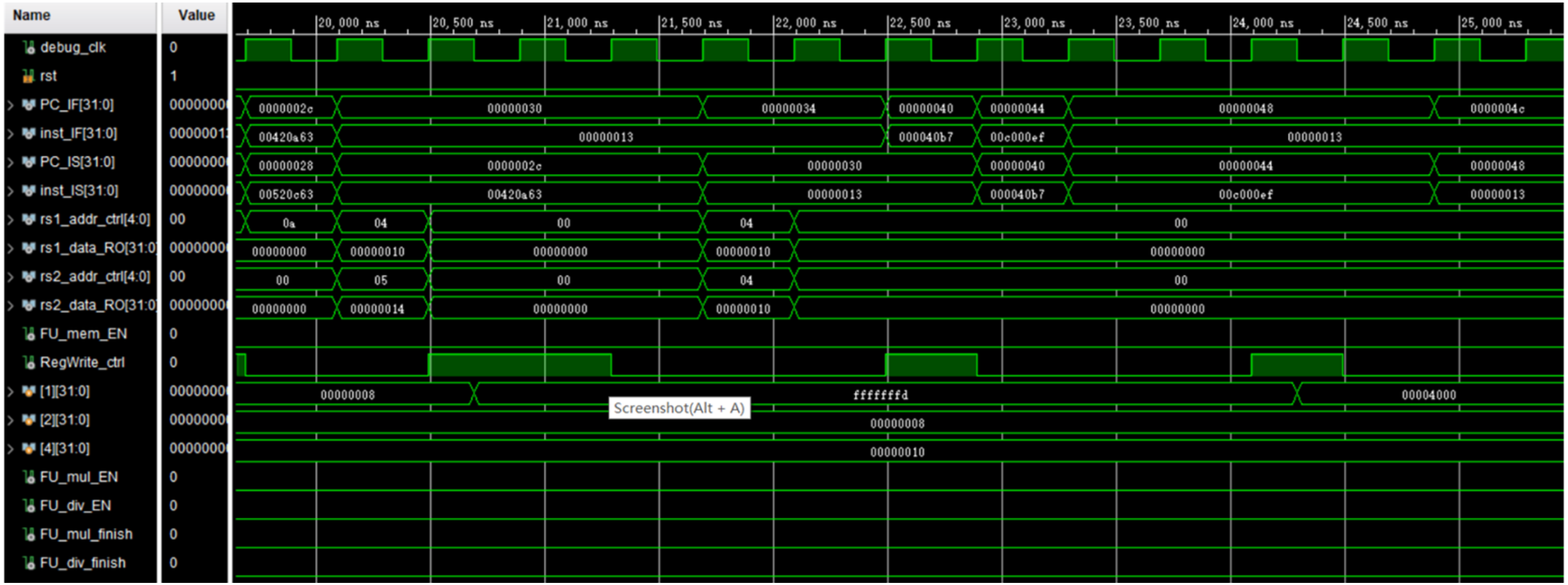
Simulation


Name
Value
10, 500 ns
11, 000 ns
11, 500 ns
12, 000 ns
12, 500 ns
13, 000 ns
13, 500 ns
14, 000 ns
14, 500 ns
15, 000 ns
15, 500 ns
debug_clk 0
rst 1
PC_IF[31:0] 0000000 00000010 00000014 00000018 0000001c 0000
inst_IF[31:0] 0000001 fff08093 00c02283 01002303 01402383 4022
PC_IS[31:0] 0000000 0000000c 00000010 00000014 00000018 0000
inst_IS[31:0] 0000000 004100b3 fff08093 00c02283 01002303 0140
rs1_addr_ctrl[4:0] 00 00 02 00 01 00
rs1_data_RO[31:0] 0000000 00000000 00000008 00000000 00000018 00000000
rs2_addr_ctrl[4:0] 00 00 04 00
rs2_data_RO[31:0] 0000000 00000000 00000010 00000000
FU_mem_EN 0
RegWrite_ctrl 0
[1][31:0] 0000000 00000000 00000018 00000017
[2][31:0] 0000000 00000008
[4][31:0] 0000000 00000000 00000010
FU_mul_EN 0
FU_div_EN 0
FU_mul_finish 0
FU_div_finish 0

Simulation

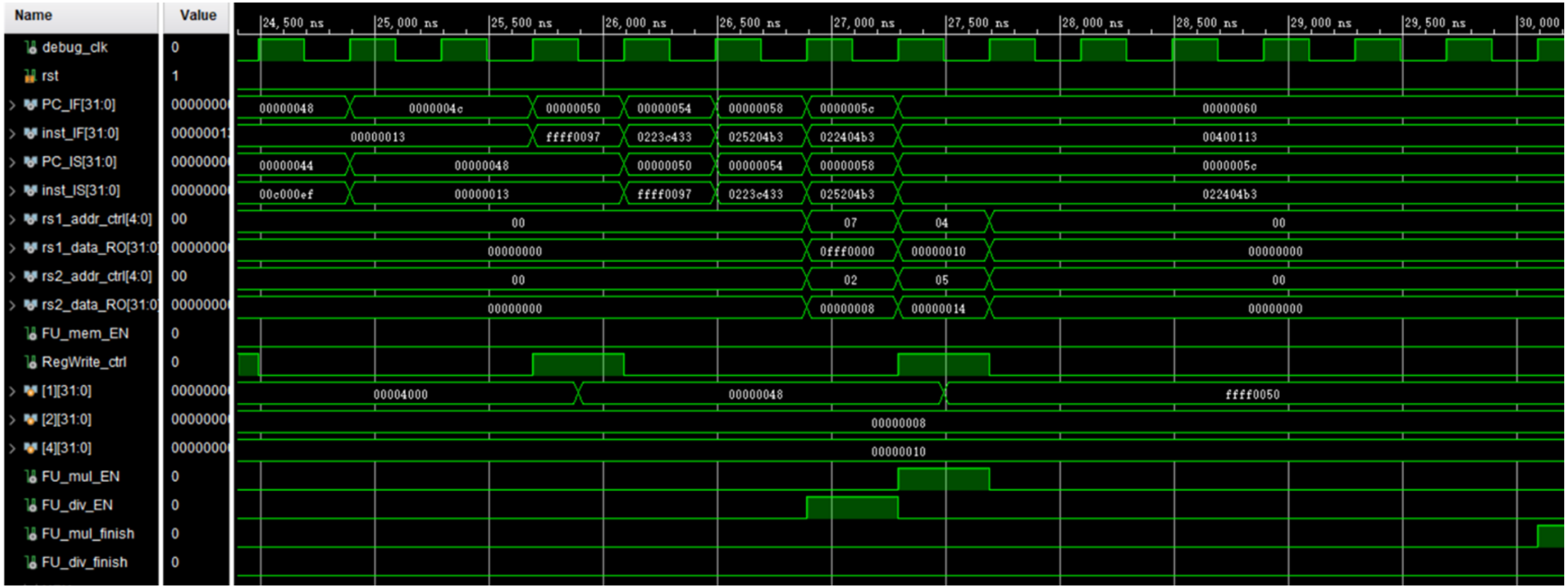

Name
Value
15, 500 ns
16, 000 ns
16, 500 ns
17, 000 ns
17, 500 ns
18, 000 ns
18, 500 ns
19, 000 ns
19, 500 ns
20, 000 ns
20, 500 ns
21,
debug_clk 0
rst 1
PC_IF[31:0] 0000000 0000001c 00000020 00000024 00000028 0000002c 00000030
inst_IF[31:0] 0000001 01402383 402200b3 ffd50093 00520c63 00420a63 00000013
PC_IS[31:0] 0000000 00000018 0000001c 00000020 00000024 00000028 0000002c
inst_IS[31:0] 0000000 01002303 01402383 402200b3 ffd50093 00520c63 00420a63
rs1_addr_ctrl[4:0] 00 00 04 00 0a 04 00
rs1_data_RO[31:0] 0000000 00000000 00000010 00000000 00000010 00000000
rs2_addr_ctrl[4:0] 00 00 02 00 05 00
rs2_data_RO[31:0] 0000000 00000000 00000008 00000000 00000014 00000000
FU_mem_EN 0
RegWrite_ctrl 0
[1][31:0] 0000000 00000017 00000008 fffffffd
[2][31:0] 0000000 00000008
[4][31:0] 0000000 00000010
FU_mul_EN 0
FU_div_EN 0
FU_mul_finish 0
FU_div_finish 0

Simulation



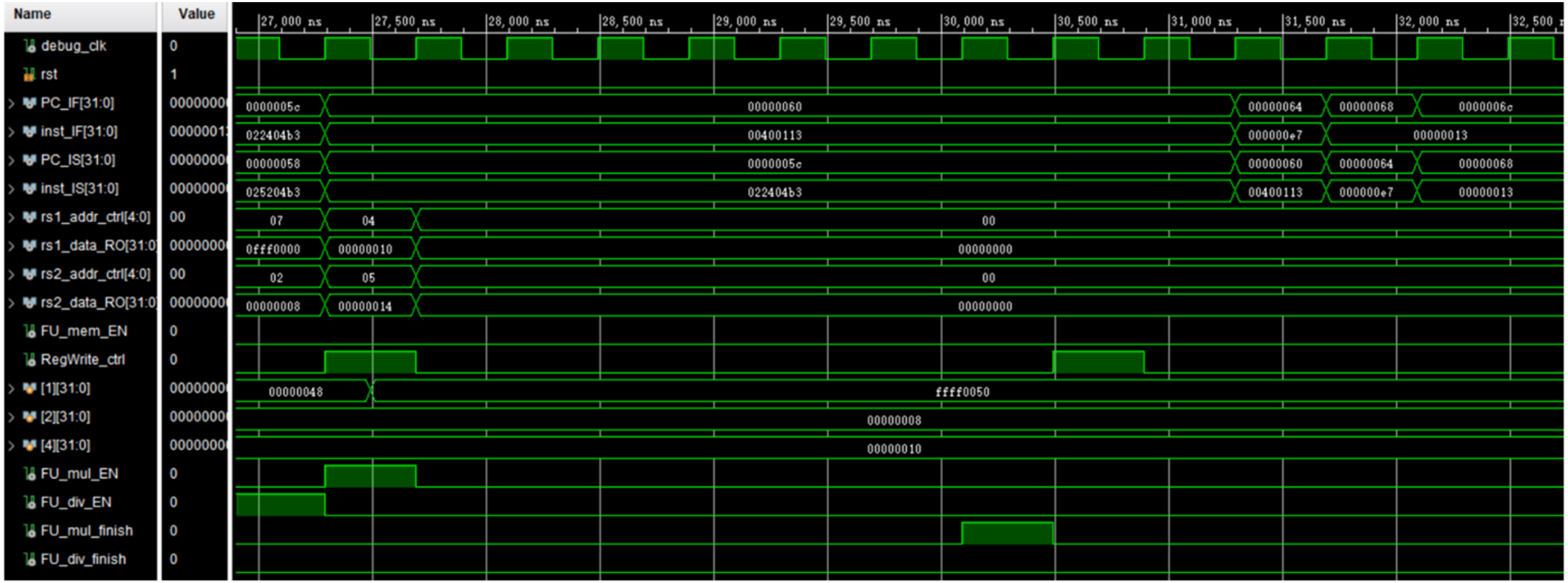
Name
Value
20,000 ns
20,500 ns
21,000 ns
21,500 ns
22,000 ns
22,500 ns
23,000 ns
23,500 ns
24,000 ns
24,500 ns
25,000 ns
debug_clk 0
rst 1
PC_IF[31:0] 0000000 0000002c 00000030 00000034 00000040 00000044 00000048 0000004c
inst_IF[31:0] 0000001 00420a63 00000013 000040b7 00c000ef 00000013
PC_IS[31:0] 0000000 00000028 0000002c 00000030 00000040 00000044 00000048
inst_IS[31:0] 0000000 00520c63 00420a63 00000013 000040b7 00c000ef 00000013
rs1_addr_ctrl[4:0] 00 0a 04 00 04 00
rs1_data_RO[31:0] 0000000 00000000 00000010 00000000 00000010 00000000
rs2_addr_ctrl[4:0] 00 00 05 00 04 00
rs2_data_RO[31:0] 0000000 00000000 00000014 00000000 00000010 00000000
FU_mem_EN 0
RegWrite_ctrl 0
[1][31:0] 0000000 00000008 fffffffd 00004000
[2][31:0] 0000000 00000008
[4][31:0] 0000000 00000010
FU_mul_EN 0
FU_div_EN 0
FU_mul_finish 0
FU_div_finish 0
Screenshot(Alt + A)

Simulation

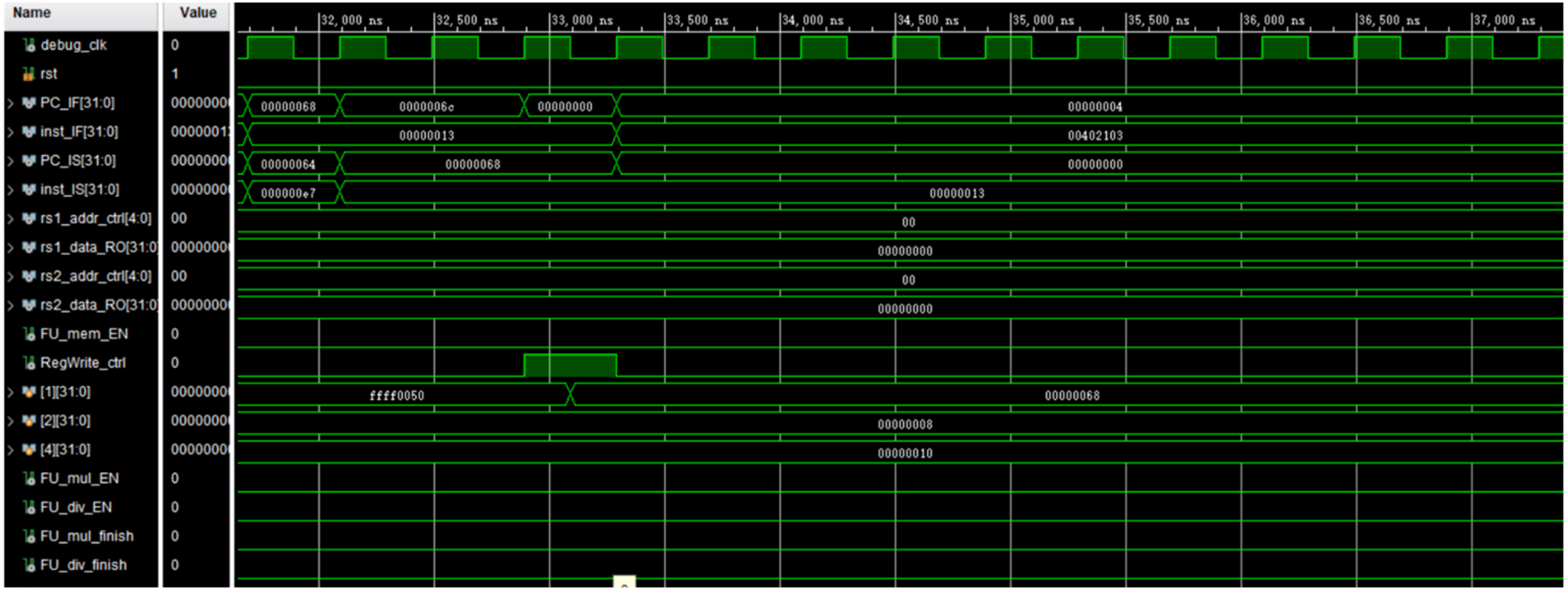


Name
Value
24, 500 ns
25, 000 ns
25, 500 ns
26, 000 ns
26, 500 ns
27, 000 ns
27, 500 ns
28, 000 ns
28, 500 ns
29, 000 ns
29, 500 ns
30, 000
debug_clk 0
rst 1
PC_IF[31:0] 0000000 | 00000048 | 0000004c | 00000050 | 00000054 | 00000058 | 0000005c | 00000060
inst_IF[31:0] 0000001 | 00000013 | ffff0097 | 0223c433 | 025204b3 | 022404b3 | 00400113
PC_IS[31:0] 0000000 | 00000044 | 00000048 | 00000050 | 00000054 | 00000058 | 0000005c
inst_IS[31:0] 0000000 | 00c000ef | 00000013 | ffff0097 | 0223c433 | 025204b3 | 022404b3
rs1_addr_ctrl[4:0] 00 | 00 | 07 | 04 | 00
rs1_data_RO[31:0] 0000000 | 00000000 | 0fff0000 | 00000010 | 00000000
rs2_addr_ctrl[4:0] 00 | 00 | 02 | 05 | 00
rs2_data_RO[31:0] 0000000 | 00000000 | 00000008 | 00000014 | 00000000
FU_mem_EN 0
RegWrite_ctrl 0
[1][31:0] 0000000 | 00004000 | 00000048 | ffff0050
[2][31:0] 0000000 | 00000008
[4][31:0] 0000000 | 00000010
FU_mul_EN 0
FU_div_EN 0
FU_mul_finish 0
FU_div_finish 0

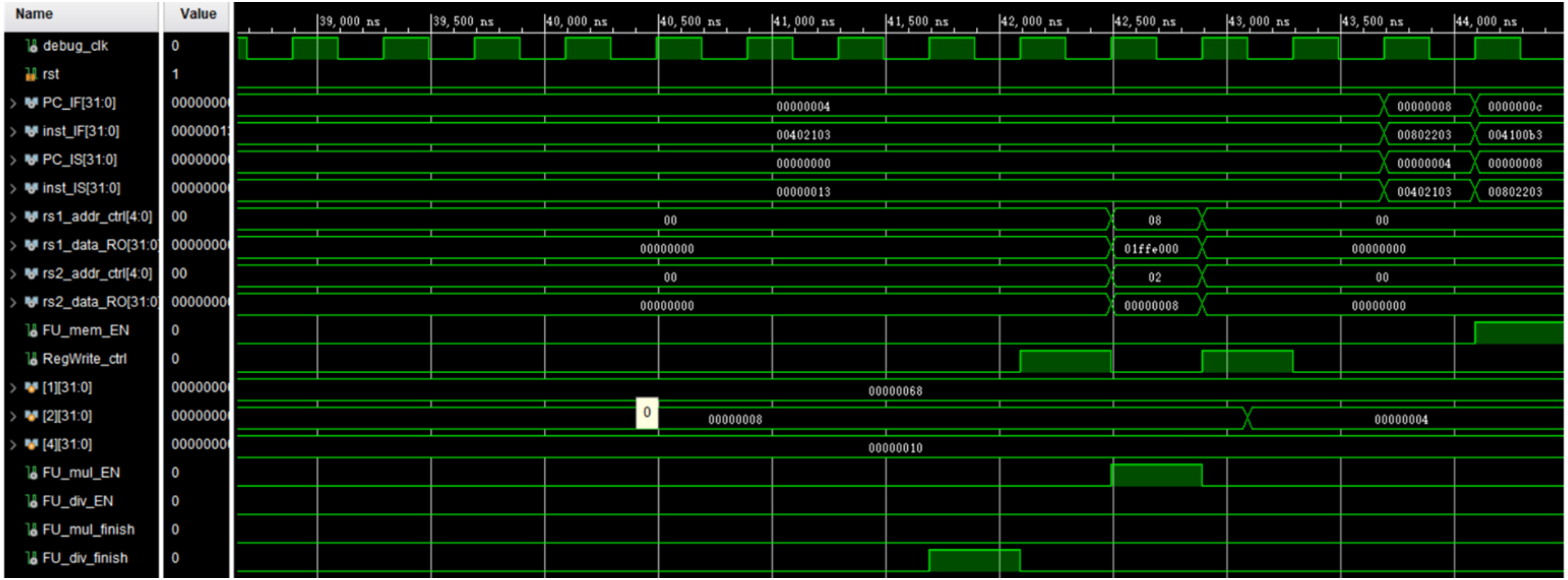
Simulation


Name
Value
27,000 ns
27,500 ns
28,000 ns
28,500 ns
29,000 ns
29,500 ns
30,000 ns
30,500 ns
31,000 ns
31,500 ns
32,000 ns
32,500
debug_clk 0
rst 1
PC_IF[31:0] 0000000 0000005c 00000060 00000064 00000068 0000006c
inst_IF[31:0] 0000001 022404b3 00400113 000000e7 00000013
PC_IS[31:0] 0000000 00000058 0000005c 00000060 00000064 00000068
inst_IS[31:0] 0000000 025204b3 022404b3 00400113 000000e7 00000013
rs1_addr_ctrl[4:0] 00 07 04 00
rs1_data_RO[31:0] 0000000 0fff0000 00000010 00000000
rs2_addr_ctrl[4:0] 00 02 05 00
rs2_data_RO[31:0] 0000000 00000008 00000014 00000000
FU_mem_EN 0
RegWrite_ctrl 0
[1][31:0] 0000000 00000048 ffff0050
[2][31:0] 0000000 00000008
[4][31:0] 0000000 00000010
FU_mul_EN 0
FU_div_EN 0
FU_mul_finish 0
FU_div_finish 0

Simulation


Name
Value
32,000 ns
32,500 ns
33,000 ns
33,500 ns
34,000 ns
34,500 ns
35,000 ns
35,500 ns
36,000 ns
36,500 ns
37,000 ns
debug_clk 0
rst 1
PC_IF[31:0] 0000000
00000068
0000006c
00000000
00000004
inst_IF[31:0] 0000001
00000013
00402103
PC_IS[31:0] 0000000
00000064
00000068
00000000
inst_IS[31:0] 0000000
000000e7
00000013
rs1_addr_ctrl[4:0] 00
00
rs1_data_RO[31:0] 0000000
00000000
rs2_addr_ctrl[4:0] 00
00
rs2_data_RO[31:0] 0000000
00000000
FU_mem_EN 0
RegWrite_ctrl 0
[1][31:0] 0000000
ffff0050
00000068
[2][31:0] 0000000
00000008
[4][31:0] 0000000
00000010
FU_mul_EN 0
FU_div_EN 0
FU_mul_finish 0
FU_div_finish 0

Simulation


Name
Value
39,000 ns
39,500 ns
40,000 ns
40,500 ns
41,000 ns
41,500 ns
42,000 ns
42,500 ns
43,000 ns
43,500 ns
44,000 ns
debug_clk 0
rst 1
PC_IF[31:0] 0000000
00000004
00000008
0000000c
inst_IF[31:0] 0000001
00402103
00802203
004100b3
PC_IS[31:0] 0000000
00000000
00000004
00000008
inst_IS[31:0] 0000000
00000013
00402103
00802203
rs1_addr_ctrl[4:0] 00
00
08
00
rs1_data_RO[31:0] 0000000
00000000
01ffe000
00000000
rs2_addr_ctrl[4:0] 00
00
02
00
rs2_data_RO[31:0] 0000000
00000000
00000008
00000000
FU_mem_EN 0
RegWrite_ctrl 0
[1][31:0] 0000000
00000068
[2][31:0] 0000000
0
00000008
00000004
[4][31:0] 0000000
00000010
FU_mul_EN 0
FU_div_EN 0
FU_mul_finish 0
FU_div_finish 0

References

- https://dl.acm.org/doi/pdf/10.1145/3369383
- https://zhuanlan.zhihu.com/p/496078836
- https://jasonren0403.github.io/scoreboard/