Arch Lab 2

Pipelined CPU supporting exception & interrupt

Chenlu Miao 10/2022

Tasks

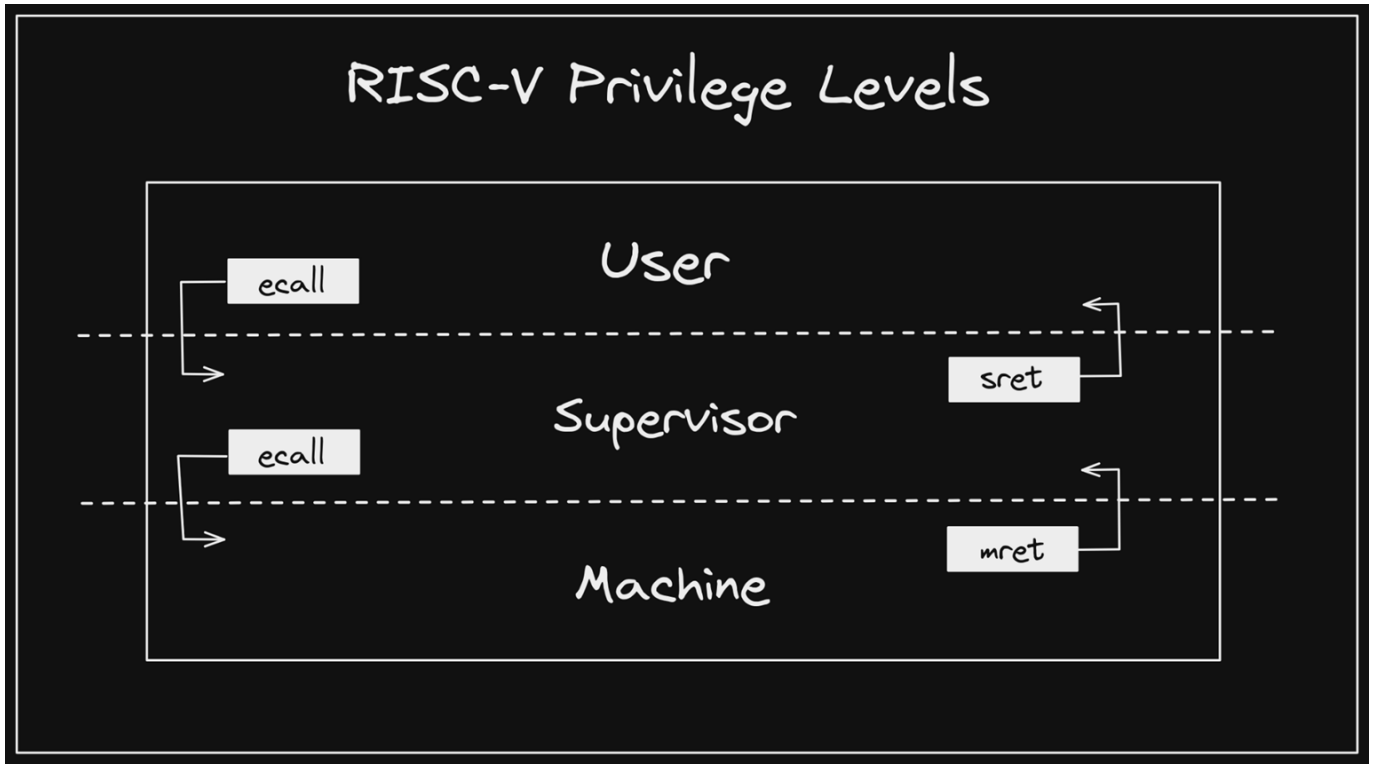
- Master the design methods of pipelined CPU supporting **exception** & **interrupt**.

Overview

- RISC-V Privilege Levels
- Control Status Registers (CSR)
- CSR Instructions
- Machine-Mode Privileged Instructions
- Trap Handling Process
- Pipelined CPU Supporting Exception & Interrupt
- Codes: Exception Unit

RISC-V Privilege Levels

RISC-V Privilege Levels



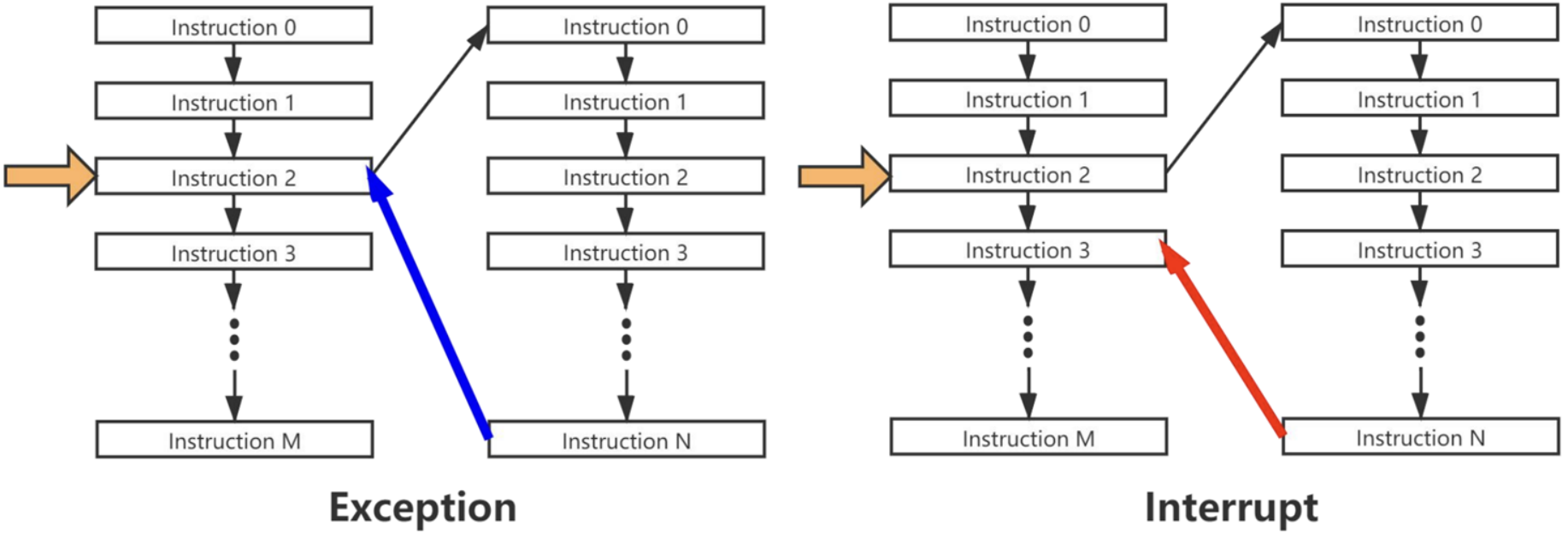
- Machine(M)
- Hypervisor(H)
- Supervisor(S)
- User(U)

RISC-V Privilege Levels

- **Trap**: the transfer of control to a trap handler caused by either an exception or an interrupt
  - **Exception**: an unusual condition occurring at run time associated with an instruction in the current RISC-V hart
  - **Interrupt**: an external asynchronous event that may cause a RISC-V hart to experience an unexpected transfer of control

https://github.com/riscv-non-isa/riscv-trace-spec/issues/10

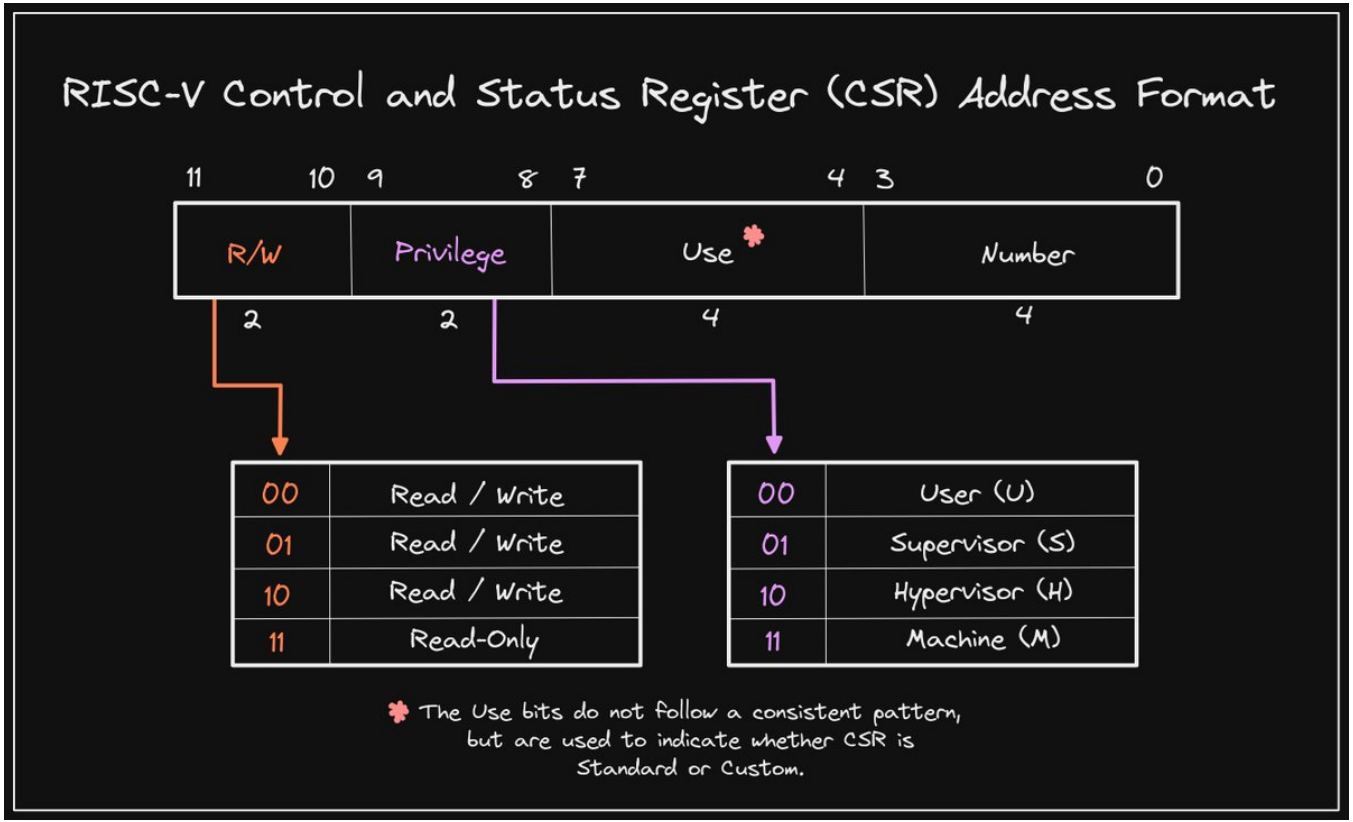
RISC-V Privilege Levels



https://github.com/plctlab/riscv-operating-system-mooc/blob/main/slides/ch02-riscv-isa-introduction.pdf

Control Status Registers

CSR (Control Status Register)

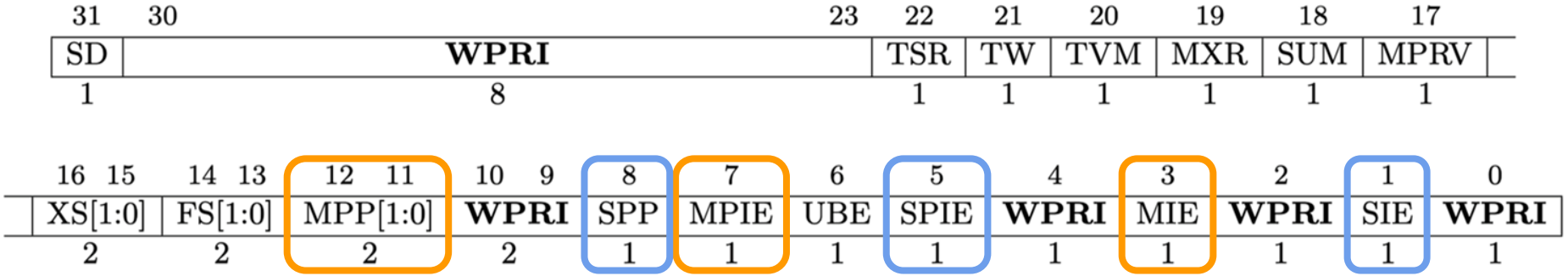


https://danielmangum.com/risc-v-tips/2021-12-12-access-privilege-csr-addresses/

CSR (Control Status Register)

Number	Abbr	Name	Description
0x300	**mstatus**	Machine Status Register	处理器状态，mstatus的MIE域和MPIE域用于反映全局中断使能
0x304	**mie**	Machine Interrupt Enable Registers	用于控制不同类型中断的局部中断使能
0x305	**mtvec**	Machine Trap-Vector Base-Address Register	定义进入异常的程序PC地址
0x341	**mepc**	Machine Exception Program Counter	用于保存异常的返回地址
0x342	**mcause**	Machine Cause Register	反映进入异常的原因
0x343	**mtval**	Machine Trap Value Register	反映进入异常的信息
0x344	**mip**	Machine Interrupt Pending Registers	反映不同类型中断的等待状态

CSR: **mstatus** (Machine Status)

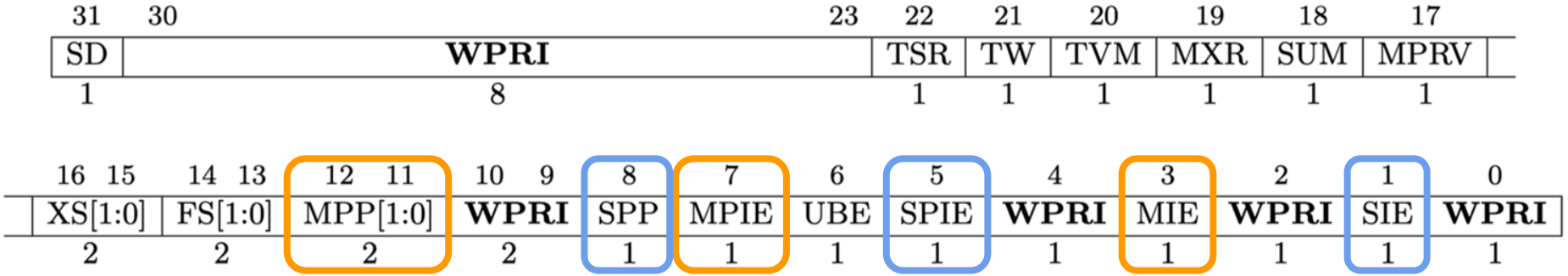


mstatus.*x*IE: Interrupt Enable in x mode

mstatus.*x*PIE: Previous Interrupt Enable in x mode

mstatus.*x*PP: Previous Priviledge mode up to x mode

CSR: **mstatus** (Machine Status)



mstatus.*x*IE: Interrupt Enable in x mode

mstatus.*x*PIE: Previous Interrupt Enable in x mode

mstatus.*x*PP: Previous Priviledge mode up to x mode

👉 enter trap： **mstatus.MPP = priv; mstatus.MPIE = mstatus.MIE; mstatus.MIE = 0;**

👈 exit trap： **mstatus.MIE = mstatus.MPIE; mstatus.MPIE = 1; priv = mstatus.MPP**

CSR: **mtvec** (Machine Trap-Vector Base-Address)

Mode = Direct:

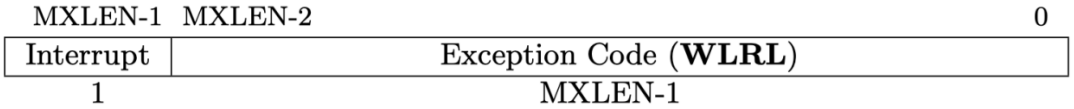
PC ← BASE

Mode = Vectored:

(Exception) PC ← BASE

(Interrupt) PC ← BASE + 4 * Cause

CSR: mcause (Machine Cause)



Interrupt	Exception Code	Description
1	0	*Reserved*
1	1	Supervisor software interrupt
1	2	*Reserved*
1	3	Machine software interrupt
1	4	*Reserved*
1	5	Supervisor timer interrupt
1	6	*Reserved*
1	7	Machine timer interrupt
1	8	*Reserved*
1	9	Supervisor external interrupt
1	10	*Reserved*
1	11	Machine external interrupt
1	12–15	*Reserved*
1	≥16	*Designated for platform use*
0	0	Instruction address misaligned
0	1	Instruction access fault
0	2	Illegal instruction
0	3	Breakpoint
0	4	Load address misaligned
0	5	Load access fault
0	6	Store/AMO address misaligned
0	7	Store/AMO access fault
0	8	Environment call from U-mode
0	9	Environment call from S-mode
0	10	*Reserved*
0	11	Environment call from M-mode
0	12	Instruction page fault
0	13	Load page fault
0	14	*Reserved*
0	15	Store/AMO page fault
0	16–23	*Reserved*
0	24–31	*Designated for custom use*
0	32–47	*Reserved*
0	48–63	*Designated for custom use*
0	≥64	*Reserved*

CSR: **mepc** (Machine Exception Program Counter)

Exception: mepc ← Current PC

Interrupt: mepc ← Next PC

Ret: PC ← mepc

Why Current PC ? Why Next PC

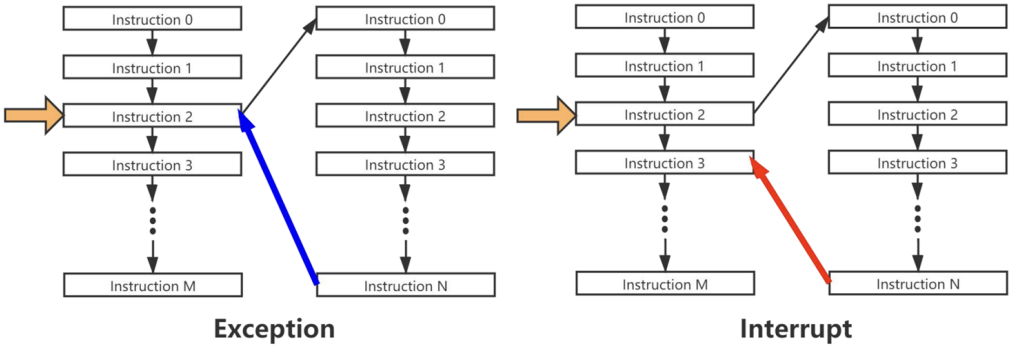
🥴 ecall ? → software set to Next PC

CSR: **mepc** (Machine Exception Program Counter)

Exception: mepc ← Current PC

Interrupt: mepc ← Next PC

Ret: PC ← me



Why Current PC ? Why Next PC

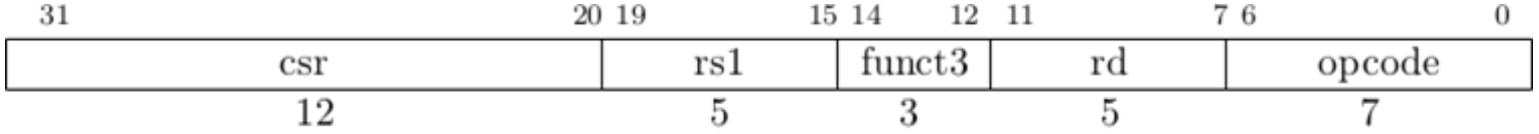
 ecall ? → software set to Next PC

CSR Instructions

CSR Instructions

csrrw rd, csr, rs1	t ← CSRs[csr], CSRs[csr] ← x[rs1], x[rd] ← t	读取一个 CSRs[csr] 的值到rd，然后把rs1写入该 CSR
csrrs rd, csr, rs1	t ← CSRs[csr], CSRs[csr] ← t|x[rs1], x[rd] ← t	读取一个 CSR 的值到rd，然后把该 CSR 中rs1指定的 bit 置 1
csrrc rd, csr, rs1	t ← CSRs[csr], CSRs[csr] ← t&~x[rs1], x[rd] ← t	读取一个 CSR 的值到rd，然后把该 CSR 中rs1指定的 bit 置 0
csrrwi rd, csr, zimm[4:0]	x[rd] ← CSRs[csr], CSRs[csr] ← zimm	
csrrsi rd, csr, zimm[4:0]	x[rd] ← CSRs[csr], CSRs[csr] ← t|zimm	
csrrci rd, csr, zimm[4:0]	x[rd] ← CSRs[csr], CSRs[csr] ← t&~zimm	

CSR Instructions

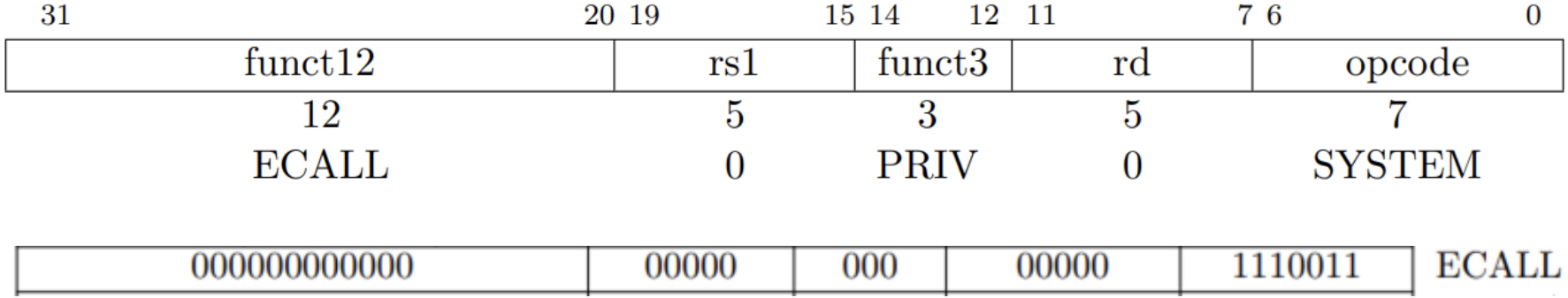


RV32/RV64 *Zicsr* Standard Extension

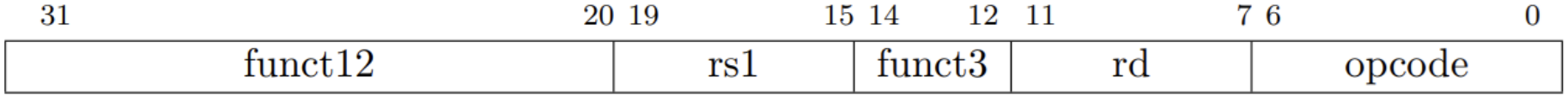
csr	rs1	001	rd	1110011	CSRRW
csr	rs1	010	rd	1110011	CSRRS
csr	rs1	011	rd	1110011	CSRRC
csr	uimm	101	rd	1110011	CSRRWI
csr	uimm	110	rd	1110011	CSRRSI
csr	uimm	111	rd	1110011	CSRRCI

Machine-Mode Privileged Instructions

Machine-Mode Privileged Instructions - ECALL



Machine-Mode Privileged Instructions - MRET



Trap-Return Instructions

0001000	00010	00000	000	00000	1110011	SRET
0011000	00010	00000	000	00000	1110011	MRET

👉 exit trap： **mstatus.MIE = mstatus.MPIE; mstatus.MPIE = 1; priv = mstatus.MPP**

Trap Handling Process

Trap Handling Process – Enter Trap

- Stop the execution of the current program
- Start from the PC address defined by the CSR mtvec.
- Update the CSR registers: mcause, mepc, and mstatus
  - mstatus (mstatus[7]=mstatus[3], mstatus[3]= 0)
  - mepc (interrupt: Next PC, exception: Cur PC)
  - mcause (interrupt? ecall ? illegal inst? load fault? store fault?)

Trap Handling Process – Trap Handler

NO.	Instruction	Addr.	Label	ASM	Comment
30	34102cf3	78	trap:	csrr x25, 0×341	# mepc
31	34202df3	7C		csrr x27, 0×342	# mcause
32	30002e73	80		csrr x28, 0×300	# mstatus
33	30402ef3	84		csrr x29, 0×304	# mie
34	34402f73	88		csrr x30, 0×344	# mip
35	004c8113	8C		addi x2, x25, 4	
36	34111073	90		csrw 0×341, x2	
37	30200073	94		mret	# 30200073 mret

Trap Handling Process – Exit Trap

- Stop the execution of the current program
- Start from the PC address defined by the CSR mepc.
- Update the CSR mstatus.
  - mstatus (mstatus[3] = mstatus[7], mstatus[7] = 1)

Pipelined CPU Supporting Exception & Interrupt

Pipelined CPU Supporting Exception & Interrupt

Precise Exceptions:

- All instructions **before** the faulting instruction **complete**.
- Instructions **following** the faulting instruction, including the faulting instruction, **do not change the state of the machine**.

Pipelined CPU Supporting Exception & Interrupt

IF： Memory Fault (Illegal Memory Address)

ID: Illegal Instruction

EX: Arithmetic Exception

MEM: Memory Fault (Illegal Memory Address)

WB

Pipelined CPU Supporting Exception & Interrupt

	1	2	3	4
Inst 1	IF	ID	EX	**MEM**
Inst 2			IF	**ID**
				EX

Pipelined CPU Supporting Exception & Interrupt

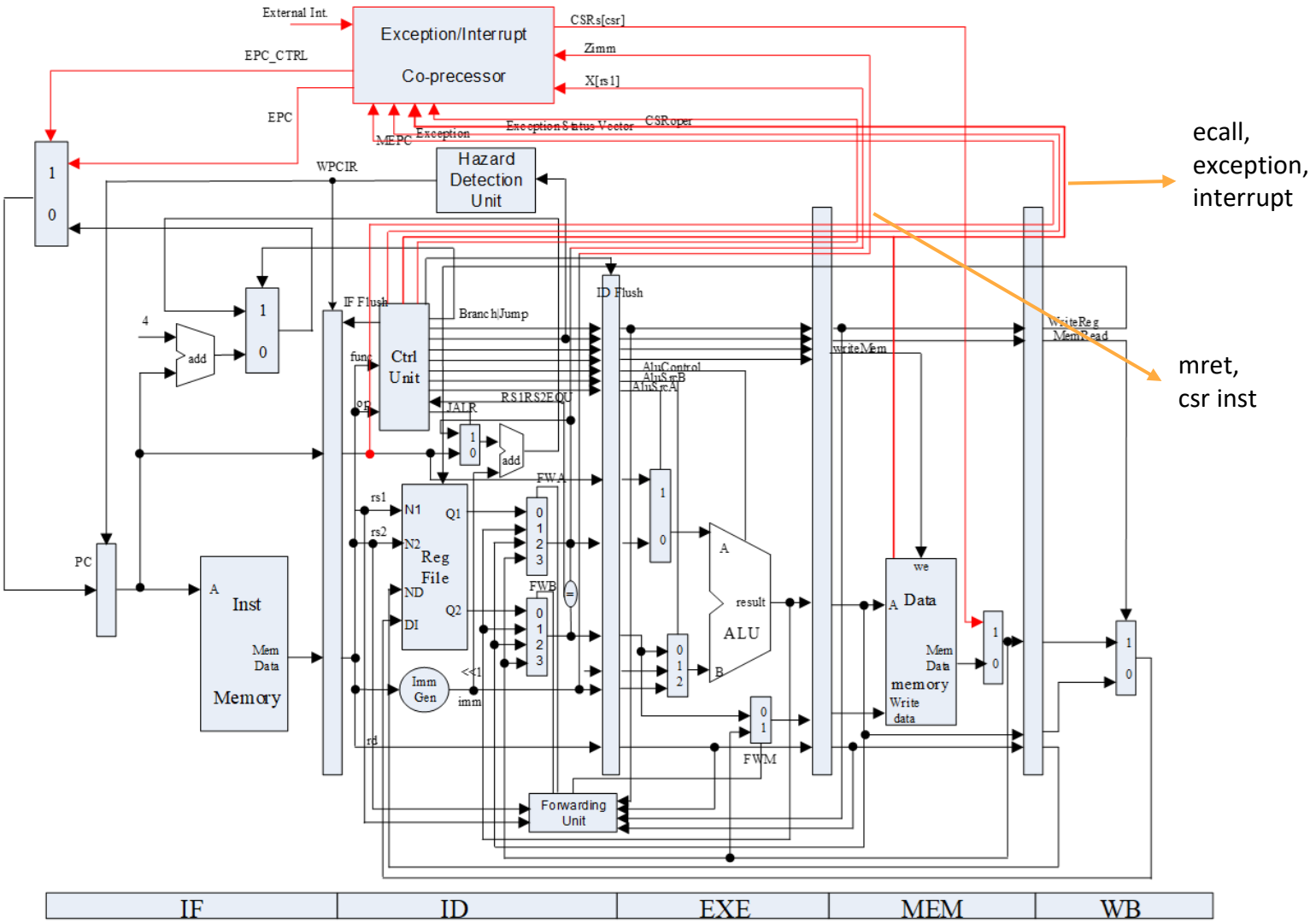
IF: Memory Fault (Illegal Memory Address)

ID: Illegal Instruction

EX: Arithmetic Exception

MEM: Memory Fault (Illegal Memory Address)

WB


External Int.
Exception/Interrupt
Co-precessor
CSRs[csr]
Zimm
X[rs1]
EPC_CTRL
EPC
MEPC
Exception
Exception Status Vector
CSRoper
ecall,
exception,
interrupt
mret,
csr inst
WPCIR
Hazard
Detection
Unit
IF Flush
ID Flush
Branch/Jump
4
add
func
Ctrl
Unit
op
JALR
RS1RS2EQU
AluControl
AluSrcB
AluSrcA
MemWrite
WriteReg
MemRead
FWA
FWB
FWM
rs1
rs2
rd
N1
N2
ND
DI
Q1
Q2
Reg
File
Imm
Gen
imm
<<1
PC
A
Inst
Mem
Data
Memory
ALU
result
we
Data
memory
Write
data
Forwarding
Unit
IF
ID
EXE
MEM
WB

Codes: Exception Unit

Exception Unit

```
// MEM
module ExceptionUnit(
    input clk, rst,
    input csr_rw_in,
    input[1:0] csr_wsc_mode_in,
    input csr_w_imm_mux,
    input[11:0] csr_rw_addr_in,
    input[31:0] csr_w_data_reg,
    input[4:0] csr_w_data_imm,
    output[31:0] csr_r_data_out,

    input interrupt,
    input illegal_inst,
    input l_access_fault,
    input s_access_fault,
    input ecall_m,
    input mret,

    input[31:0] epc_cur,
    input[31:0] epc_next,
    output[31:0] PC_redirect,
    output redirect_mux,

    output reg_FD_flush, reg_DE_flush,
reg_EM_flush, reg_MW_flush,
    output RegWrite_cancel,
);
```

```
CSRRegs
csr(.clk(clk),.rst(rst),.csr_w(csr_w),.raddr(csr_raddr),.waddr(csr_waddr),
.wdata(csr_wdata),.rdata(csr_r_data_out),.mstatus(mstatus),.csr_wsc_mode(cs
r_wsc));
```

Exception Unit

```
ExceptionUnit exp_unit(.clk(debug_clk),.rst(rst),.csr_rw_in(csr_rw_MEM),.csr_wsc_mode_in(inst_MEM[13:12]),
    .csr_w_imm_mux(csr_w_imm_mux_MEM),.csr_rw_addr_in(inst_MEM[31:20]),
    .csr_w_data_reg(rs1_data_MEM),.csr_w_data_imm(rs1_MEM),
    .csr_r_data_out(CSRout_MEM),
```

csr operations

```
    .interrupt(interrupter),
    .illegal_inst(~isFlushed_WB & exp_vector_WB[3]),
    .ecall_m(~isFlushed_WB & exp_vector_WB[2]),
    .l_access_fault(~isFlushed_WB & exp_vector_WB[1]),
    .s_access_fault(~isFlushed_WB & exp_vector_WB[0]),
    .mret(mret_MEM),

    .epc_cur(PC_WB),
    .epc_next(~isFlushed_MEM ? PC_MEM : ~isFlushed_EXE ? PC_EXE :
    ~isFlushed_ID ? PC_ID : PC_IF),
    .PC_redirect(PC_redirect_exp),.redirect_mux(redirect_mux_exp),
    .reg_FD_flush(reg_FD_flush_exp),.reg_DE_flush(reg_DE_flush_exp),
    .reg_EM_flush(reg_EM_flush_exp),.reg_MW_flush(reg_MW_flush_exp),
    .RegWrite_cancel(RegWrite_cancel_exp));
```

exception & interrupt

CSR Regs

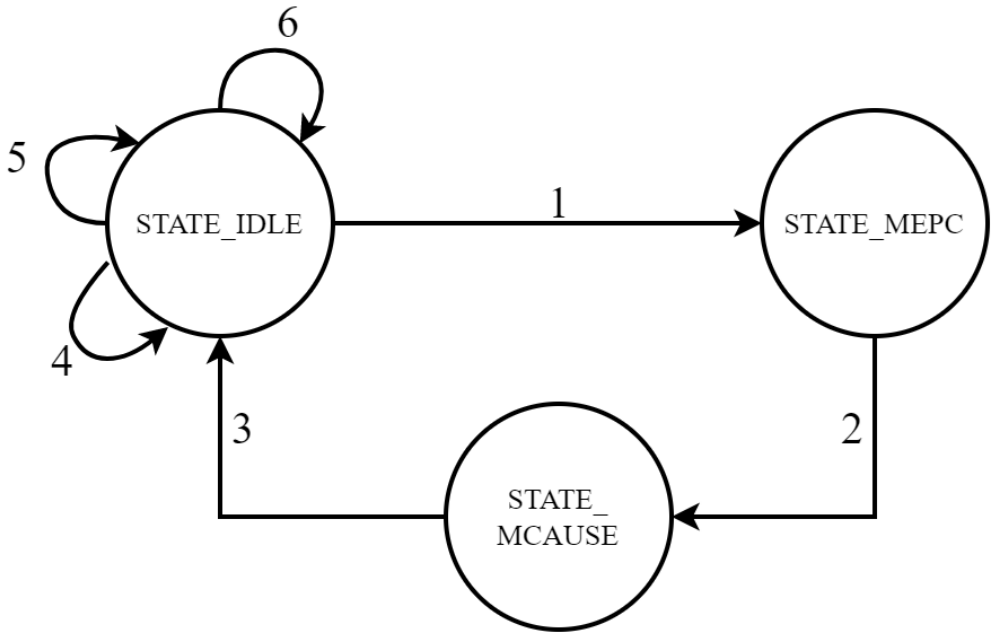
```
module CSRRegs(
    input clk, rst,
    input[11:0] raddr, waddr,
    input[31:0] wdata,
    input csr_w,
    input[1:0] csr_wsc_mode,
    output[31:0] rdata,
    output[31:0] mstatus
);
```

one read port / write port

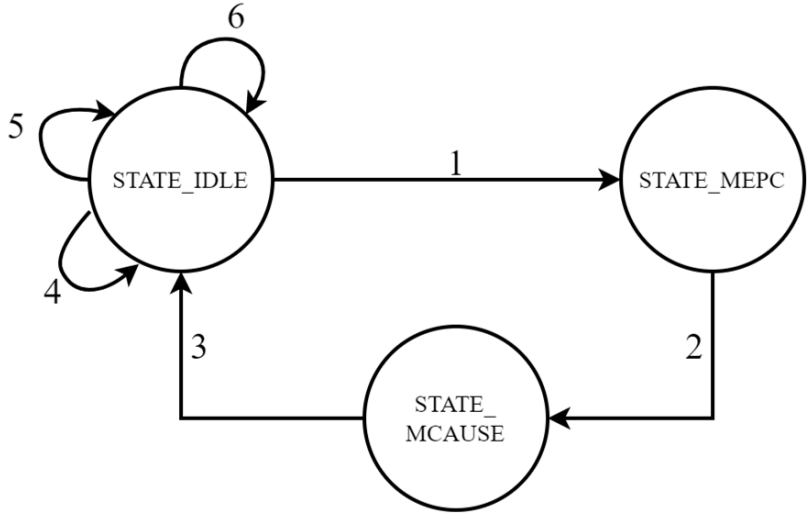
read addr / write addr

write/set/clear

Exception Unit

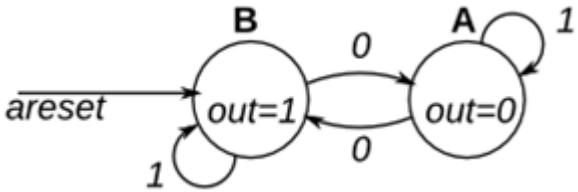


Exception Unit



1. STATE_IDLE →(exception or interruption) STATE_MEPC
   - write mstatus
   - flush all the pipeline registers
   - if exception (not interrupt), cancal regwrite
   - record epc and cause
2. STATE_MEPC → STATE_MCAUSE
   - write epc to mepc
   - read mtvec
   - flush pipeline register (FD)
   - set redirect pc mux (next cycle pc → mtvec)
3. STATE_MCAUSE → STATE_IDLE
   - write cause to mcause
4. STATE_IDLE → (mret) STATE_IDLE
   - write mstatus
   - read mepc
   - set redirect pc mux (next cycle pc → mepc)
   - flush pipeline registers (EM, DE, FD)
5. STATE_IDLE → (csr insts) STATE_IDLE
   - csr operations
6. STATE_IDLE → (other) STATE_IDLE

Verilog Finite State Machine




```
module top_module(
    input clk,
    input reset,
    input in,
    output out);

    parameter A=0, B=1;
    reg state, next_state;
```

```
    always @(*) begin    // This is a combinational always block
        // State transition logic
        case(state)
            A: begin
                if(in) next_state = A;
                else next_state = B;
            end
            B: begin
                if(in) next_state = B;
                else next_state = A;
            end
            default:
                next_state = B;
        endcase
    end
```

```
    always @(posedge clk) begin
    // This is a sequential always block
    // State flip-flops
        if(reset) state <= B;
        else state <= next_state;
    end

    // Output logic
    // assign out = (state == ...);
    assign out = ((state == A) & 0)
               |((state == B) & 1);
endmodule
```

https://hdlbits.01xz.net/wiki/Fsm1

Exception Unit

```
localparam STATE_IDLE = 2'b00;
localparam STATE_MEPC = 2'b01;
localparam STATE_MCAUSE = 2'b10;

reg[1:0] cur_state, next_state;
```

```
always @(posedge clk) begin
    cur_state <= next_state;
end
```

⚠️ Save epc and cause in `always @(posedge clk)`

Exception Unit

```
CSRRegs
csr(.clk(clk),.rst(rst),.csr_w(csr_w),.raddr(csr_raddr),.waddr(csr_waddr), .wdata(c
sr_wdata),.rdata(csr_r_data_out),.mstatus(mstatus),.csr_wsc_mode(csr_wsc));
```

```
always @* begin
    case(cur_state)
        STATE_IDLE:begin
            if(trap_in) begin
                csr ...
                next_state = ...
            end
            else if(mret) begin
                csr ...
                next_state = ...
            end
            else if(csr_rw_in) begin
                csr ...
                next_state = ...
            end
            else begin
                csr_w = 0;
                next_state = STATE_IDLE;
            end
        end
        STATE_MEPC:begin
                csr ...
                next_state = ...
        end
        STATE_MCAUSE:begin
                csr ...
                next_state = ...
        end
    endcase
end
```

Exception Unit

```
assign PC_redirect = csr_r_data_out;
assign redirect_mux = ...

assign reg_MW_flush = ...
assign reg_EM_flush = ...
assign reg_DE_flush = ...
assign reg_FD_flush = ...
assign RegWrite_cancel = ...
```

ROM

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)	
3	00c02283	C		lw x5, 12(x0)	
4	01002303	10		lw x6, 16(x0)	
5	01402383	14		lw x7, 20(x0)	
6	306850f3	18		csrrwi x1, 0x306, 16	
7	306020f3	1C		csrr x1, 0x306	csrrs x1, 0x306, x0
8	306310f3	20		csrrw x1, 0x306, x6	
9	306020f3	24		csrr x1, 0x306	
10	00000013	28		addi x0, x0, 0	
11	07800093	2C		addi x1, x0, 120	
12	30509073	30		csrw 0x305, x1	set mtvec=0x78
13	00000013	34		addi x0, x0, 0	
14	00000073	38		ecall	

test interrupt after setting mtvec

ROM

NO.	Instruction	Addr.	Label	ASM	Comment
15	00000013	3C		addi x0, x0, 0	
16	00000012	40		addi x0, x0, 0	# change to illegal
17	00000013	44		addi x0, x0, 0	
18	07f02083	48		lw x1, 127(x0)	
19	08002083	4C		lw x1, 128(x0)	# l access fault
20	00000013	50		addi x0, x0, 0	
21	08102023	54		sw x1, 128(x0)	# s access fault
22	00000013	58		addi x0, x0, 0	
23	00000013	5C		addi x0, x0, 0	
24	00000013	60		addi x0, x0, 0	
25	00000013	64		addi x0, x0, 0	
26	00000013	68		addi x0, x0, 0	
27	00000013	6C		addi x0, x0, 0	
28	00000013	70		addi x0, x0, 0	
29	00000067	74		jr x0	

ROM

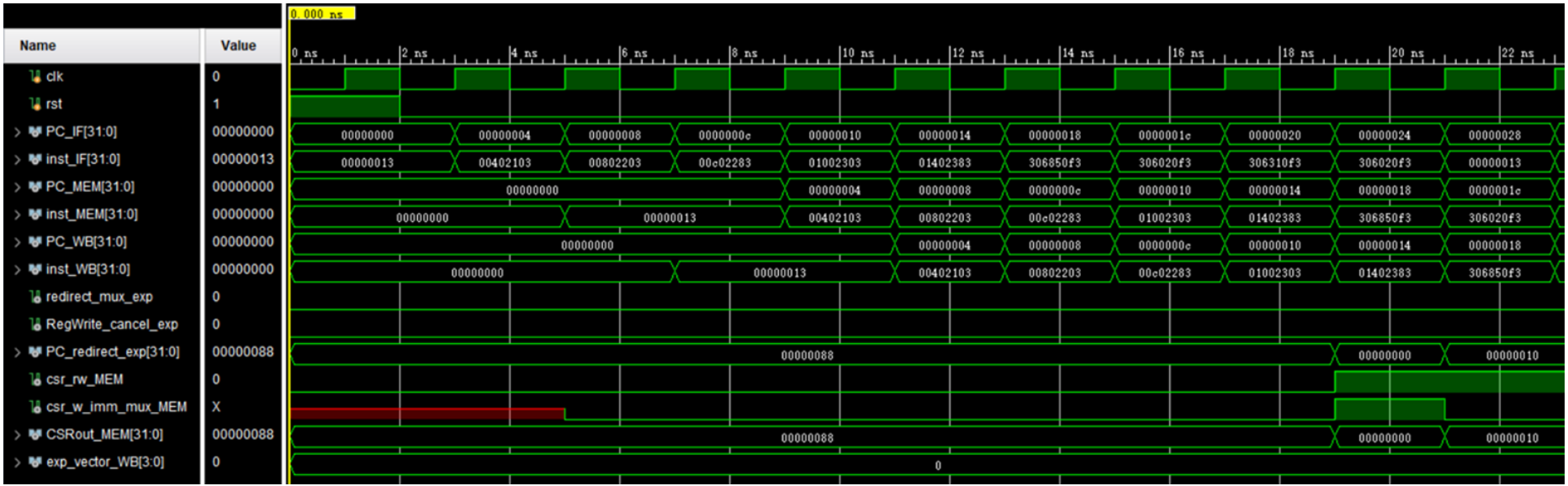
NO.	Instruction	Addr.	Label	ASM	Comment
30	34102cf3	78	trap:	csrr x25, 0x341	# mepc
31	34202df3	7C		csrr x27, 0x342	# mcause
32	30002e73	80		csrr x28, 0x300	# mstatus
33	30402ef3	84		csrr x29, 0x304	# mie
34	34402f73	88		csrr x30, 0x344	# mip
35	004c8113	8C		addi x2, x25, 4	
36	34111073	90		csrw 0x341, x2	# mepc = mepc + 4
37	30200073	94		mret	# 30200073 mret
38	00000013	98		addi x0, x0, 0	
39	00000013	9C		addi x0, x0, 0	
40	00000013	A0		addi x0, x0, 0	
41	00000013	A4		addi x0, x0, 0	

RAM

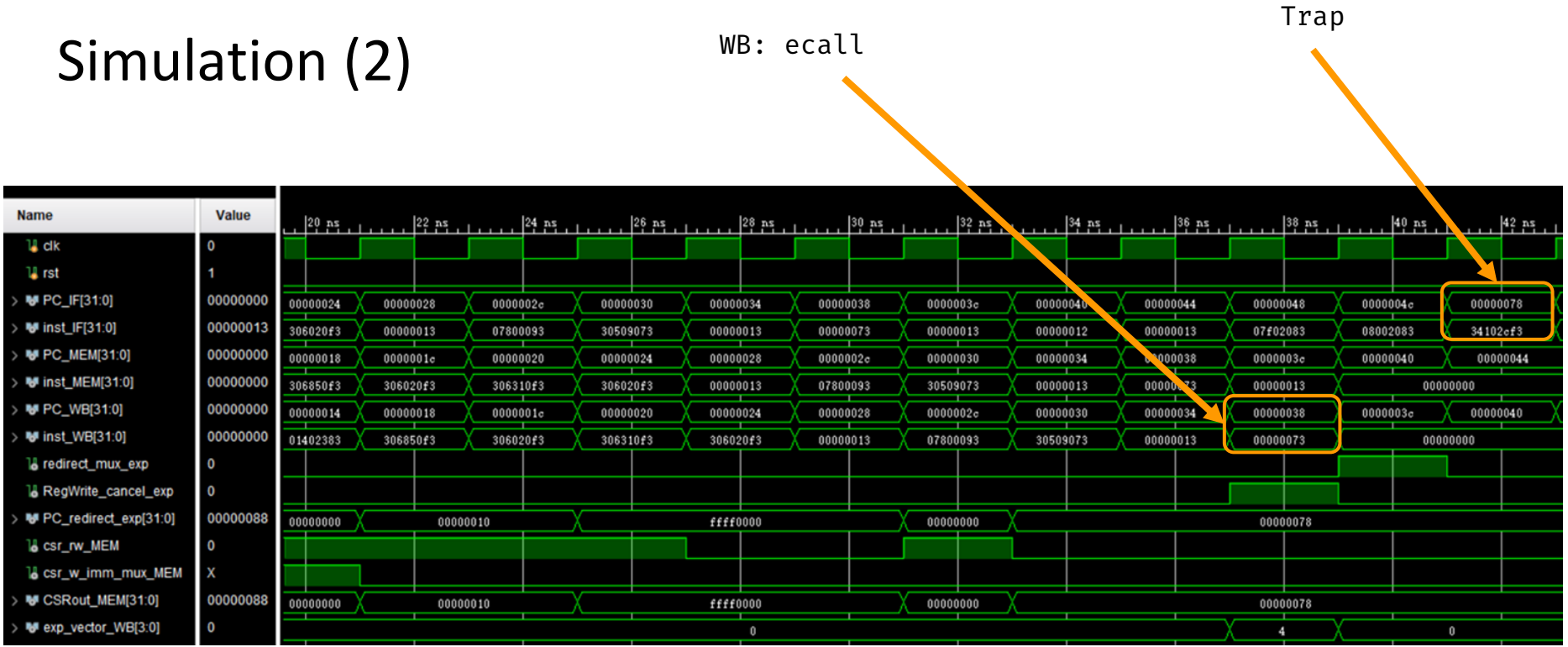
NO.	Data	Addr.
0	000080BF	0
1	00000008	4
2	00000010	8
3	00000014	C
4	FFFF0000	10
5	0FFF0000	14
6	FF000F0F	18
7	F0F0F0F0	1C
8	00000000	20
9	00000000	24
10	00000000	28
11	00000000	2C
12	00000000	30
13	00000000	34
14	00000000	38
15	00000000	3C

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

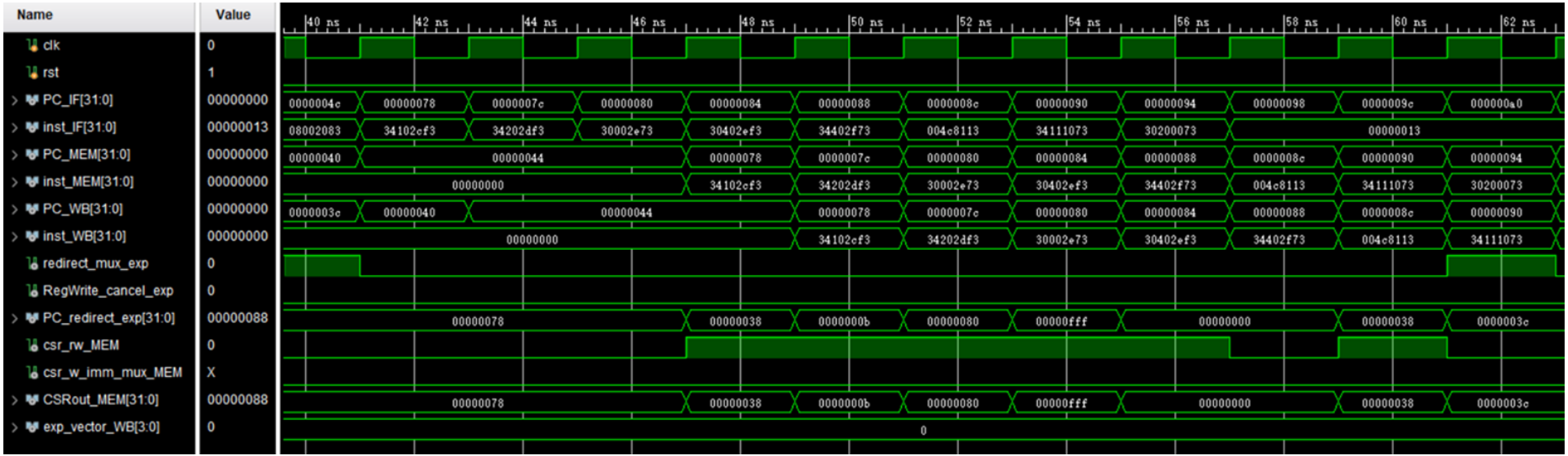
Simulation (1)



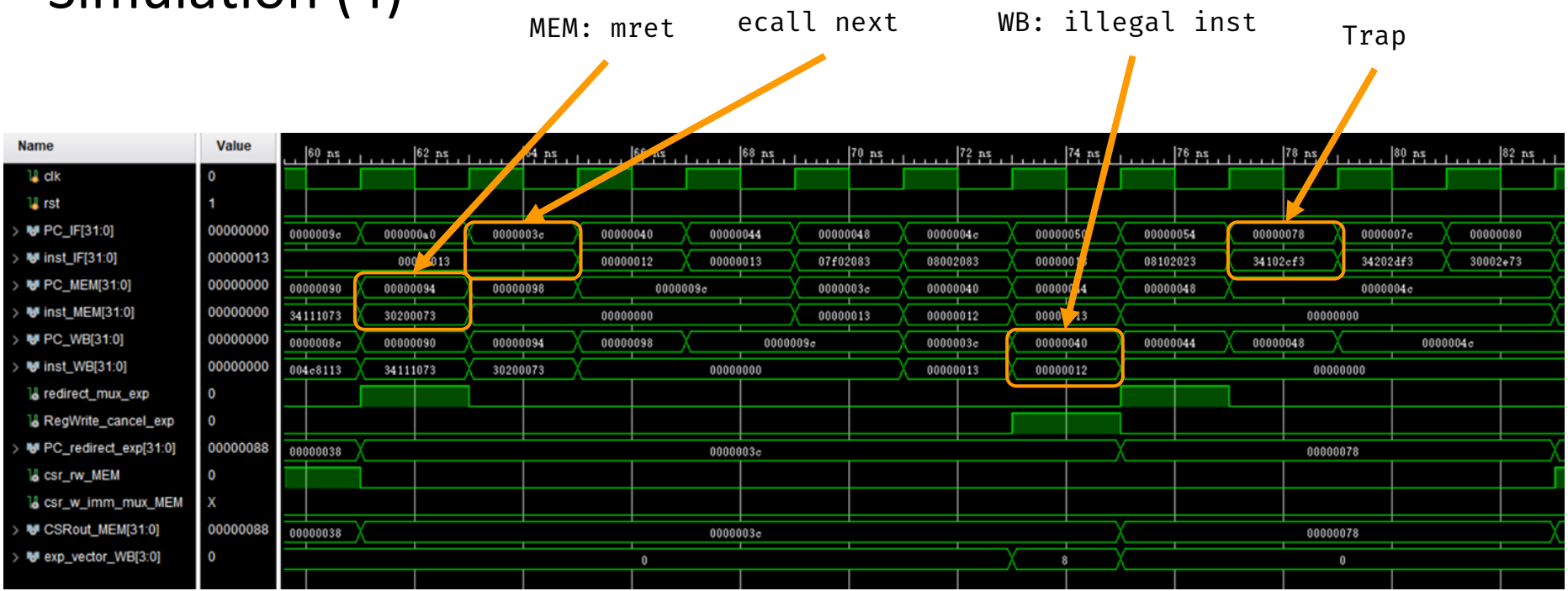
Simulation (2)



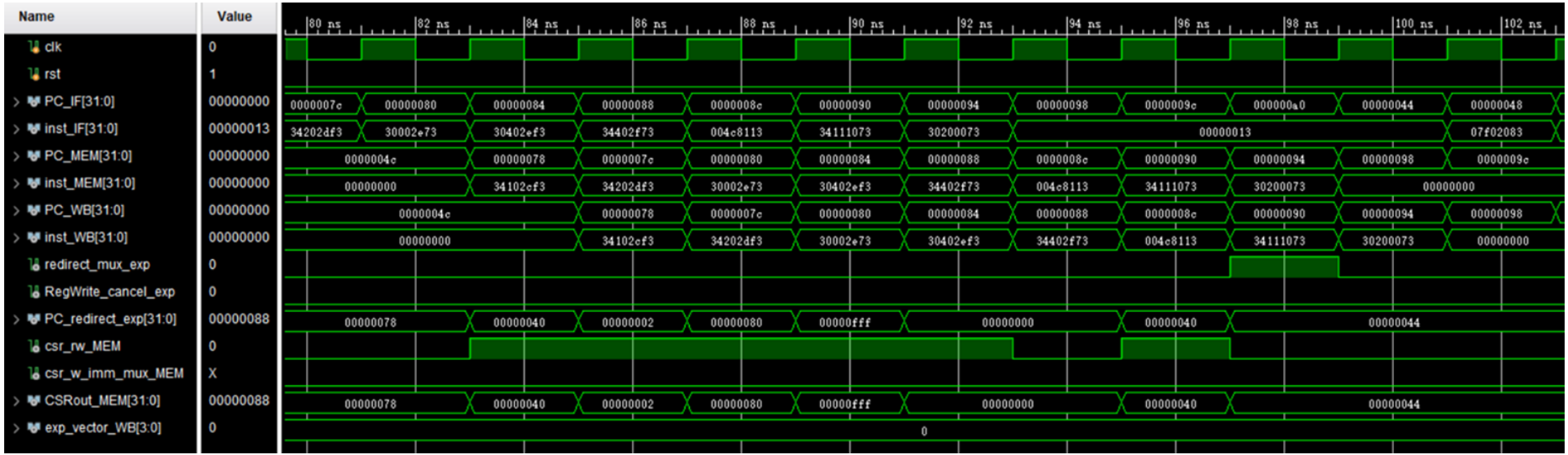
Simulation (3)



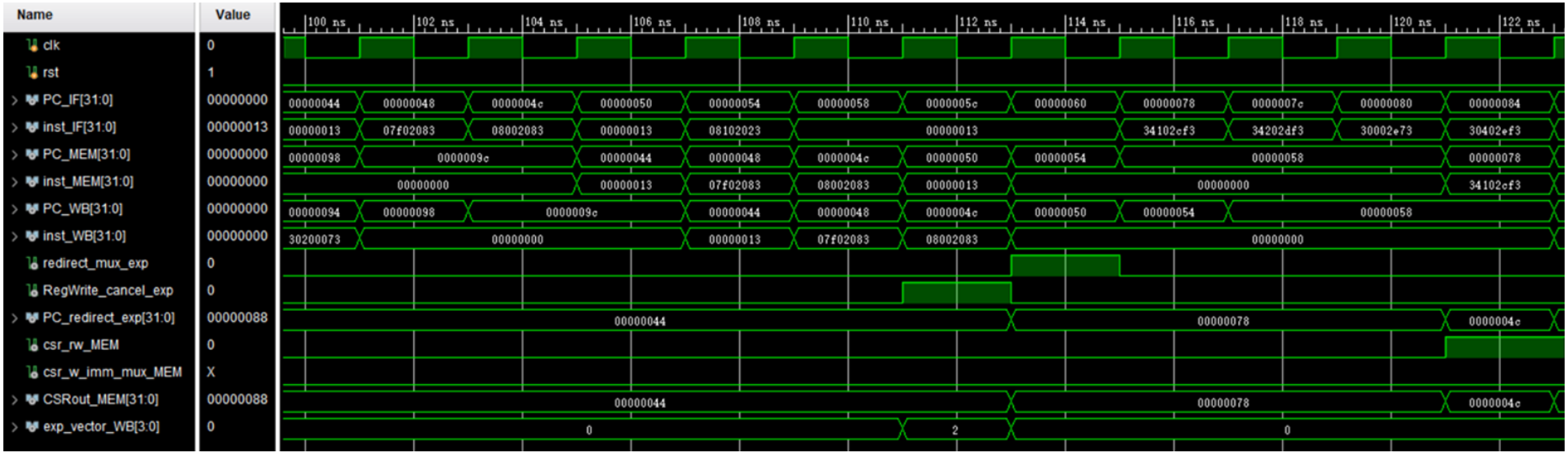
Simulation (4)



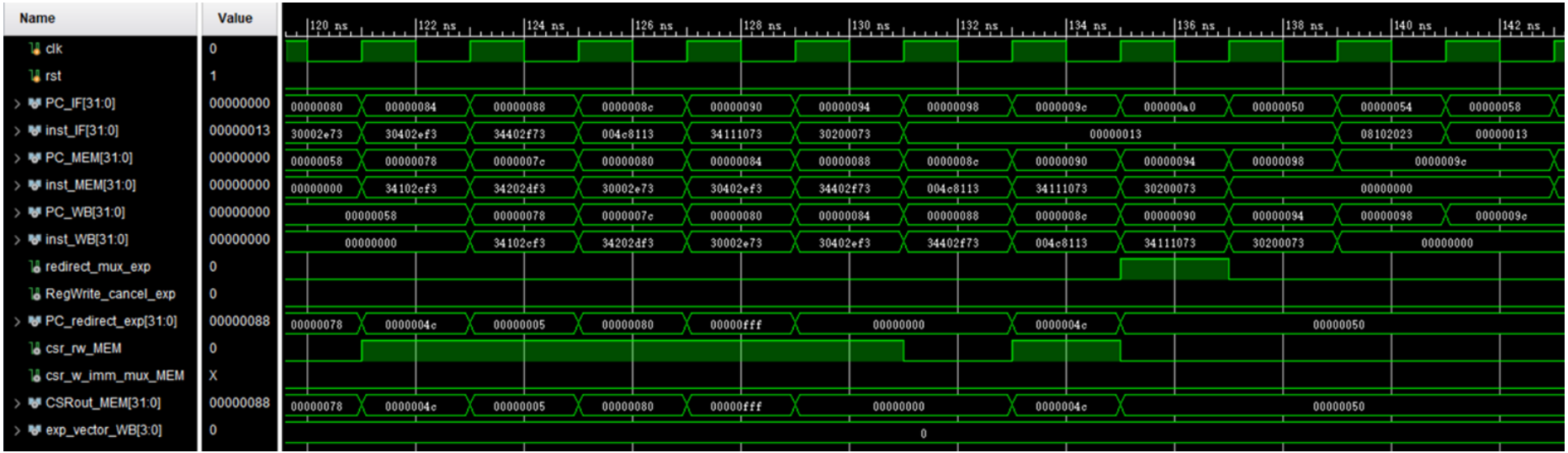
Simulation (5)



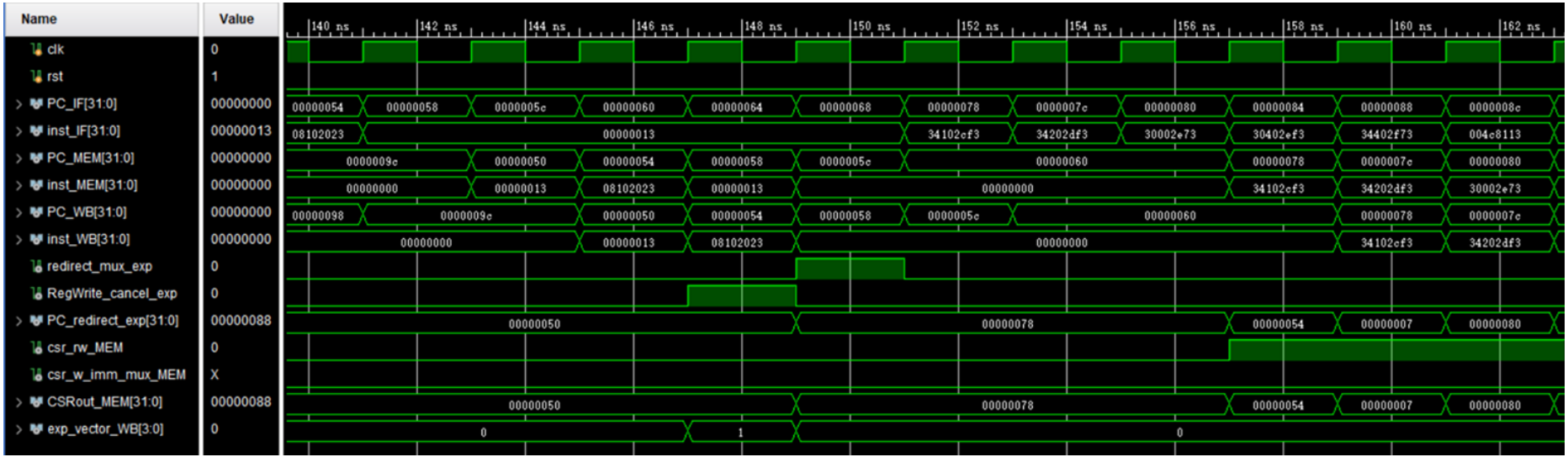
Simulation (6)



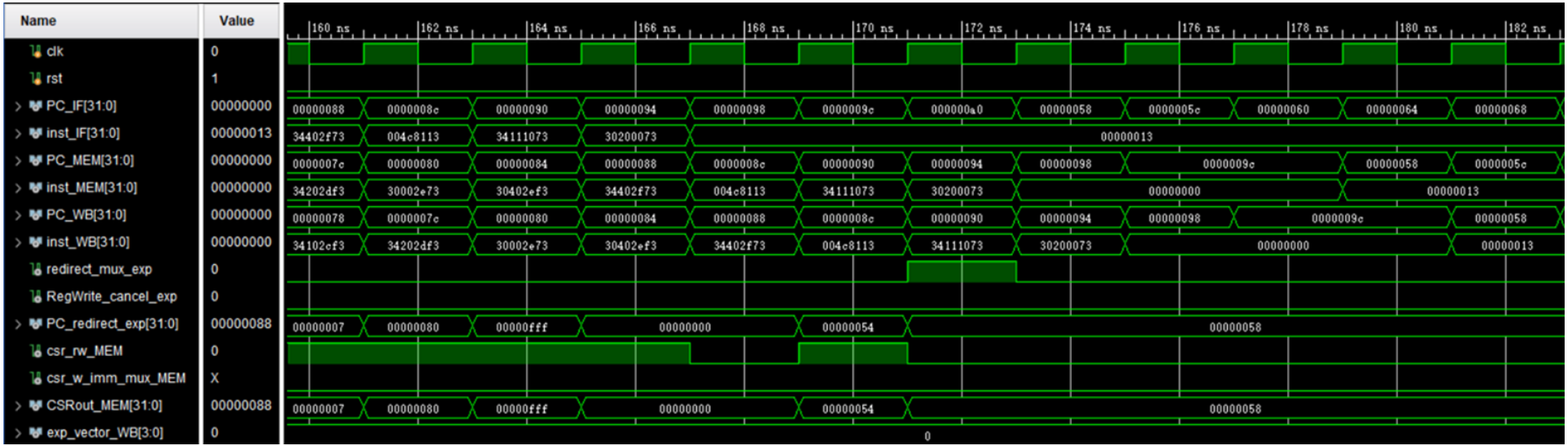
Simulation (7)



Simulation (8)



Simulation (9)



SW[12] Interrupt

Code2Inst.v CSR instructions

References

- https://danielmangum.com/posts/risc-v-bytes-privilege-levels/
- https://www.notion.so/MODE-e78a2091e1fd4de199a63262508d15e5#0cb123be2cc3471d9990288c5ba1cd6c
- https://doc.nucleisys.com/nuclei_spec/isa/exception.html
- https://github.com/plctlab/riscv-operating-system-mooc/blob/main/slides/ch02-riscv-isa-introduction.pdf
- https://groups.google.com/g/comp.lang.verilog/c/2X9f9ds9XnE