



Topic 2. Pipelined CPU supporting exception & interrupt

浙江大学计算机学院

2021年9月

Outline



- Experiment Purpose
- Experiment Task
- Basic Principle
- Operating Procedures
- Checkpoints



Experiment Purpose



- Understand the principle of CPU exception & interrupt and its processing procedure.
- Master the design methods of pipelined CPU supporting exception & interrupt.
- master methods of program verification of Pipelined CPU supporting exception & interrupt.



Experiment Task



- Design of Pipelined CPU supporting exception & interrupt.
 - Design datapath
 - Design Co-processor & Controller

Verify the Pipelined CPU with program and observe the execution of program



Revie 4.6 Interruption & Exception



- What's Interruption & Exception?
- Why need interruption & exception?
- How to deal with interruption & Exception in RISC V ?
 - □ Transfer control to exception handler & return from exception
 - Control status registers
 - CSR instructions
 - How to write an exception handler?



Interruption & Exception



- The cause of changing CPU's work flow:
 - Control instructions in program (bne/beq, jal, etc)
 It is foreseeable in programming flow
 - Something happen suddenly (Exception and Interruption)
 It is unpredictable
 - Call Instructions triggered by hardware

Exception

- Arises within the CPU when execute instruction
- **a** e.g., overflow, undefined opcode, syscall, ...

Type of event	From where?	RISC-V terminology
System reset	External	Exception
I/O device request	External	Interrupt
Invoke the operating system from user program	Internal	Exception
Using an undefined instruction	Internal	Exception
Hardware malfunctions	Either	Either

- Interrupt
 - **□** From an external I/O controller
- Dealing with them without sacrificing performance is hard



Why we need interrupt?



- When you double click the mouse ……
- When a network package arrives ······
- When you want to print a sentence on screen ……

Event external to the running program can interrupt the processor: ex Interruption driven I/O

- polling ----waste a lot of processor time
- Interruption driven I/O
- DMA ---- direct memory access



Why we need exception?



 Processor can be interrupted by exceptional events that occur while the program is running that are caused by the program itself.

Example:

- Page fault:
 - need OS to load the page into the memory from disk, then resume the program
- Memory address fault (segmentation fault)
- Undefined opcode
 - The OS will stop the program and then transfer to other process.



Handling Exceptions



- Save PC of offending (or interrupted) instruction
 - In RISC-V: Supervisor Exception Program Counter (SEPC)
- Save indication of the problem
 - In RISC-V: Supervisor Exception Cause Register (SCAUSE)
 - 64 bits, but most bits unused
 - Exception code field: 2 for undefined opcode, 12 for hardware malfunction, ...
- Jump to handler
 - Assume at 0000 0000 1C09 0000_{hex}
 - Entry address in a special register: Suptervisor Trap Vector (STVEC), which can be loaded by OS.



An Alternate Mechanism



- Vectored Interrupts
 - Handler address determined by the cause
- Exception vector address to be added to a vector table base register:

Undefined opcode

00 0100 0000_{two}

■ Hardware malfunction: 01 1000 0000_{two}

□ ···:

- Instructions either
 - Deal with the interrupt, or
 - Jump to real handler



Handler Actions



- Read cause, and transfer to relevant handler
- Determine action required
- If restartable
 - Take corrective action
 - use SEPC to return to program (mret)
- Otherwise
 - Terminate program
 - Report error using SEPC, SCAUSE, ...
 - OS make the choice to transfer to another ready process



Exceptions in RISC V



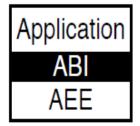
- □ Transfer control to exception handler & return from exception
- Control status registers
- CSR instructions
- How to write an exception handler?

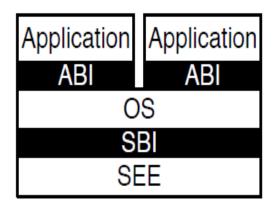


Privileged Architecture



Software stack





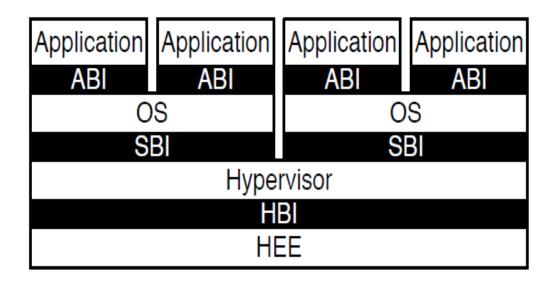


Figure 1.1: Different implementation stacks supporting various forms of privileged execution.



Privileged level



Privilege Level in RISC V

Provide protection between different components of the software stack

Level	Encoding	Name	Abbreviation
0	00	User/Application	U
1	01	Supervisor	S
2	10	Reserved	
3	11	Machine	M

Table 1.1: RISC-V privilege levels.

Number of levels		Supported Modes	Intended Usage	
1 2 3		M	Simple embedded systems	
		M, U	Secure embedded systems	
		M, S, U	Systems running Unix-like operating systems	



Table 1.2: Supported combinations of privilege modes.

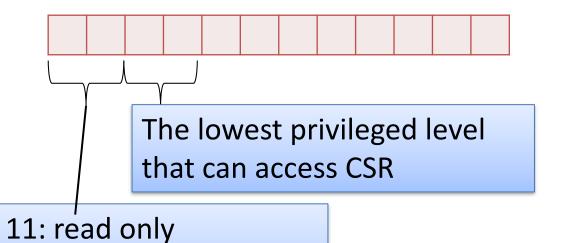
CSR Address Mapping Convention



• CSR[11..0] 4096

00/01/10: read/write

12 bit encoding space



	Machine CSRs									
00	11	XXXX	0x300-0x3FF	Standard read/write						
01	11	OXXX	0x700-0x77F	Standard read/write						
01	11	100X	0x780-0x79F	Standard read/write						
01	11	1010	0x7A0-0x7AF	Standard read/write debug CSRs						
01	11	1011	0x7B0-0x7BF	Debug-mode-only CSRs						
01	11	11XX	0x7C0-0x7FF	Custom read/write						
10	11	OXXX	0xB00-0xB7F	Standard read/write						
10	11	10XX	0xB80-0xBBF	Standard read/write						
10	11	11XX	0xBC0-0xBFF	Custom read/write						
11	11	OXXX	0xF00-0xF7F	Standard read-only						
11	11	10XX	0xF80-0xFBF	Standard read-only						
11	11	11XX	0xFC0-0xFFF	Custom read-only						



8 important CSR for exception handling



■ 8 CSR:

- mtvec (Machine Trap Vector): jump to this address when exception
- mepc (Machine Exception PC): the instruction raise the exception
- mcause (Machine exception Cause): which kind of exception(cause)
- > mie (Machine Interrupt Enable): which exception can be handled or negleted
- mip (Machine interrupt pending): pending interruptions (read only register)
- mtval (Mahine trap value): error address, illegal instruction, or 0
- mscratch (Machine Scratch):
- mstatus (Machine status): processor status





Pipelined CPU supporting exception & interrupt

Number	Privilege	Name	Description		
		Mach	ine Information Registers		
0xF11	MRO	mvendorid	Vendor ID.		
0xF12	MRO	marchid	Architecture ID.		
0xF13	MRO	mimpid	Implementation ID.		
0xF14	MRO	mhartid	Hardware thread ID.		
		1	Machine Trap Setup		
0x300	MRW	mstatus	Machine status register.		
0x301	0x301 MRW misa		ISA and extensions		
0x302	0x302 MRW medeleg		Machine exception delegation register.		
0x303	MRW	mideleg	Machine interrupt delegation register.		
0x304	MRW	mie	Machine interrupt-enable register.		
0x305	MRW	mtvec	Machine trap-handler base address.		
0x306	MRW	mcounteren	Machine counter enable.		
		M	achine Trap Handling		
0x340	MRW	mscratch	Scratch register for machine trap handlers.		
0x341	MRW	mepc	Machine exception program counter.		
0x342 MRW mcause		mcause	Machine trap cause.		
0x343 MRW mtval			Machine bad address or instruction.		
0x344	MRW	mip	Machine interrupt pending.		



mtvec



base[31..2] mode

- Store the interruption handler entrance address
- The base can be explained according to mode code

□ 00: PC ←base

□ x1: vector mode, PC \leftarrow mtval -1 + 4x (not required)



mepc



- Save the instruction address when exception raised or interruption happens.
 - the PC indicate the instruction that raise the exception
 - □ the instruction need to be executed after back from interruption
 - next instruction



mcause



_	31	30	
	interruption	exception code	

- If mcause[31] == 1 then the trap was caused by an interruption.
- Exception code field: a code identifying the last exception.



Exception Causes



Asynchronization exception

- Software interrupt
- Timer interrupt
- External interrupt

Synchronization exception

- Address misaligned
- Access fault
- illegal instruction
- Breakpoint
- Environment call

		ON UNIT
Interrupt / Exception	Exception Code	Description
mcause[XLEN-1]	mcause[XLEN-2:0]	Description
1	1	Supervisor software interrupt
1	3	Machine software interrupt
1	5	Supervisor timer interrupt
1	7	Machine timer interrupt
1	9	Supervisor external interrupt
1	11	Machine external interrupt
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 address misaligned
0	7	Store access fault
0	8	Environment call from U-mode
0	9	Environment call from S-mode
0	11	Environment call from M-mode
0	12	Instruction page fault
0	13	Load page fault
0	15	Store page fault

图 10.3: RISC-V 异常和中断的原因。中断时 mcause 的最高有效位置 1, 同步异常时置 0, 且低有效标识了中断或异常的具体原因。只有在实现了监管者模式时才能处理监管者模式中断和页面错误异



Machine Status Register



31	3	0							23	22	21	20	19	18	1	7
SD					WPI	RI				TSR	TW	TVM	MX	R SUM	MP	RV
1					8					1	1	1	1	1	1	
16	15	14 13	12	11	10	9	8	7	6	5	4	4	3	2	1	0
XS[1	[0:	FS[1:0]	MPF	P[1:0]	WP	RI	SPP	MPIE	UBE	SPIE	WI	PRI	MIE	WPRI	SIE	WPRI
2		2	-	2	2		1	1	1	1		1	1	1	1	1

Figure 3.7: Machine-mode status register (mstatus) for RV32.

- MIE is an interruption enable (global) (different from the MIE register)
- When an interruption/ exception raised: mstatus[7] ← mstatus[3]
- When MRET is executed: mstatus[3] ← mstatus[7]



CSR instructions



31	25 24	20	19 15	14 12	,11 7	76 0	4
	csr		rs1	001	rd	1110011	I csrrw
	csr		rs1	010	rd	1110011	I csrrs
	csr		rs1	011	rd	1110011	I estre
	csr		zimm	101	rd	1110011	I csrrwi
	csr		zimm	110	rd	1110011	I cssrrsi
	csr		zimm	111	rd	1110011	I esrrei



CSR instructions



- csrrw rd, csr, rs1: t←CSRs[csr], CSRs[csr]←x[rs1], x[rd]←t
- csrrs rd, csr, rs1: t←CSRs[csr], CSRs[csr] ←t | x[rs1], x[rd]←t
- csrrc rd, csr, rs1: t←CSRs[csr], CSRs[csr] ←t & ~x[rs1],
 x[rd]←t
- csrrwi rd, csr, zimm[4..0]: x[rd] ←CSRs[csr], CSRs[csr]←zimm
- cssrrsi rd,csr,zimm[4..0]: t←CSRs[csr], CSRs[csr]←t | zimm;

x[rd]←t

How to Checks for Exceptions(Hardware)



Add test logic

illegal instruction, load address misaligned, store address misaligned

add control signal

- CauseWrite for mcause
- EPCWrite for mepc
- TVALWrite for mtval

process of control

- mepc ← PC(exception) or PC + 4 (interruption)
- mcause ← set correspondent bit
- mtval ← memory address or illegal instruction
- mstatus.mpie ←Mstatus.mie; mstatus.mie ←0; mstatus.mpp←mp; mp←11
- PC←address of process routine (mtvec, ex. c0000000)







Jump to handler

■ Assume at 0000 0000 1C09 0000_{hex}

jump when

mstatus.MIE = 1 && mie[i] = 1 && mip [i] = 1







```
# save registers
csrrw a0, mscratch, a0 # save a0; set a0 = &temp storage
                     # save a1
sw a1, 0(a0)
sw a2, 4(a0)
                    # save a2
sw a3, 8(a0)
                    # save a3
sw a4, 12(a0)
                   # save a4
# decode interrupt cause
csrr a1, mcause
                      # read exception cause
bgez a1, exception
                      # branch if not an interrupt
andi a1, a1, 0x3f
                      # isolate interrupt cause
li a2, 7
                      # a2 = timer interrupt cause
bne a1, a2, otherInt
                      # branch if not a timer interrupt
```





Ex. Exception handler (cont.)

```
# handle timer interrupt by incrementing time comparator
                       # a1 = &time comparator
la a1, mtimecmp
lw a2, 0(a1)
                       # load lower 32 bits of comparator
lw a3, 4(a1)
                       # load upper 32 bits of comparator
addi a4, a2, 1000
                       # increment lower bits by 1000 cycles
sltu a2, a4, a2
                       # generate carry-out
add a3, a3, a2
                       # increment upper bits
sw a3, 4(a1)
                       # store upper 32 bits
sw a4, 0(a1)
                       # store lower 32 bits
# restore registers and return
lw a4, 12(a0)
                    # restore a4
lw a3, 4(a0)
                     # restore a3
lw a2, 4(a0)
               # restore a2
lw a1, 0(a0)
               # restore a1
csrrw a0, mscratch, a0 # restore a0; mscratch = &temp storage
                       # return from handler
mret
```





Pipelined CPU supporting exception & interrupt

Environment Call and Breakpoint

31	20 1	9	15 14	12	11	7 6	0
funct12		rs1	fun	ct3	$_{\mathrm{rd}}$	opcode	
12		5	3		5	7	•
ECALL		0	PR	IV	0	SYSTEM	
EBREAK		0	PR	IV	0	SYSTEM	

Trap-Return Instructions

31		20 19	1	5 14 12	11	7 6	0
	funct12		rs1	funct3	rd	opcode	
	12		5	3	5	7	
MRE	ET/SRET/URET		0	PRIV	0	SYSTEM	





How to back to the exception breakpoint?

mret

- □ PC ← CSRs[mepc]
- mstatus.MIE ←mstatus.MPIE
- □ mp← mstatus.MPP

31	2:	5 24 20	19 15	14 12	11 7	6 0
0	011000	00010	00000	000	00000	1110011





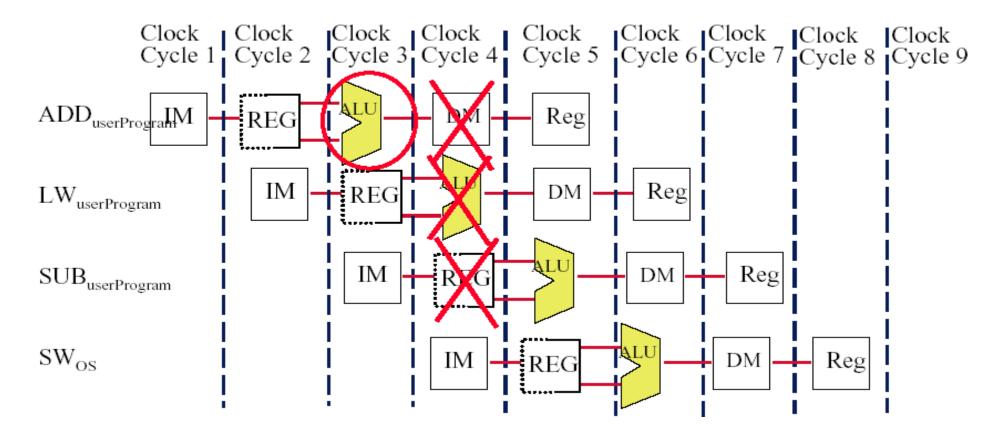
How to support exception in pipelined CPU?



Flow of Instructions During Exception



■ Example: Add instruction overflows in clock cycle 3





Precise Exceptions



- If the pipeline can be stopped so that the instructions issued before the faulting instruction complete, then the pipeline is said to implement precise exceptions
 - All instructions before the faulting instruction complete
 - And instructions following the faulting instruction, including the faulting instruction, do not change the state of the machine.
- Under this model, restarting is easy:
 - Simply re-execute the original faulting instruction.
 - Or, if it is not a resumable instruction, i.e. an integer overflow, start with the next instruction.



Imprecise Exceptions



- Difficult to do when some instructions take multiple cycles to complete
 - Some instructions may complete before an exception is detected
 - Example

Multiply r1, r2, r3; multiply takes 10 cycles

Add r10,r11,r12; add takes 5 cycles

- Add will complete before multiply is done. If multiply overflows, thenan exception will be raised AFTER the add has updated the value in R10.
- □ This is an imprecise exception.



Which stage can exceptions occur in?



Stage Problem exceptions occurring

IF page fault on instruction fetch;

misaligned memory access;

memory protection violation

ID undefined or illegal opcode

EX arithmetic exception

MEM page fault on data fetch;

misaligned memory access;

memory-protection violation

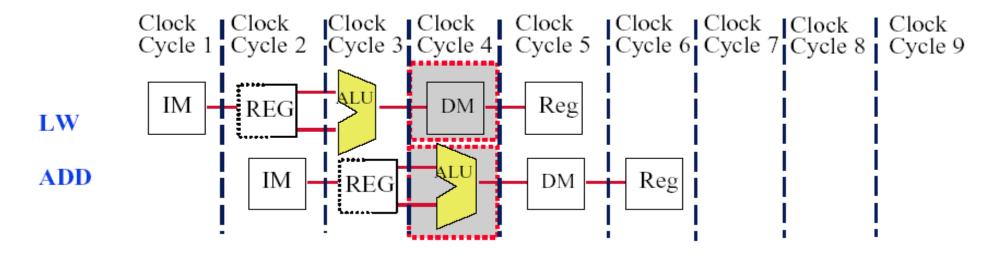
WB none



Multiple Exceptions in one clock cycle



- In Clock Cycle 4, LW can have a data page fault while the ADD has an arithmetic exception
- Handled by servicing the page fault and then restarting the LW instruction
- The ADD's arithmetic exception will occur again because the ADD instruction is restarted after the exception is handled

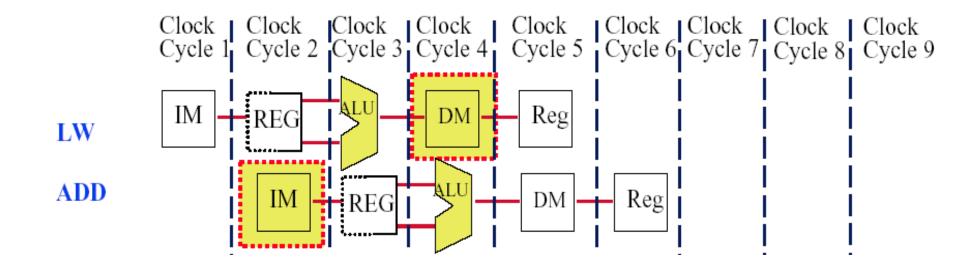




Multiple Exceptions out-of-order



- ADD causes an exception in the instruction fetch stage while LW causes an exception in the memory access stage
- If we implement precise exceptions, LW exception must be handled first
- This is done by having hardware post exceptions by order of instruction





Exception ordering



- When the instruction is about to exit the pipeline (MEM/WB), any pending exceptions for the instruction are examined.
- If an instruction generates multiple exceptions, the exception occurring in the earliest stage takes precedence.
- This is done by keeping an exception vector for each instruction:
 - ☐ If an exception is posted, it is added to the vector and all writes that affect system state are disabled.



About Exceptions



- One of the single messiest parts of designing a modern CPU
 - □ It isn't pretty, it's easy to get wrong
 - □ It's often not too elegant
 - □ It usually takes huge wads of special logic
- Further complicated by modern CPU mechanisms
 - □ Deep pipes
 - □ Superscalar --lots of instructions in flight in parallel
 - Out-of-order execution
 - time order of exceptions ≠ program order of the instructions on which the exceptions happened
 - Maintaining illusion of "sequential instruction execution" gets really complicated.







- 1. When an exception is raised, there are several instructions flying in the pipeline, how to stop the pipeline at the right place and safely shutdown the pipeline?
 - > If an exception is posted, it is added to the vector and all writes that affect system state are disabled
 - > Record the correspond cause and information (such as memory address)

- 2. How to keep the right exception ordering?
 - > Trap when there is exception or interruption at the stage of WB



Pipelined CPU supporting exception & interrupt

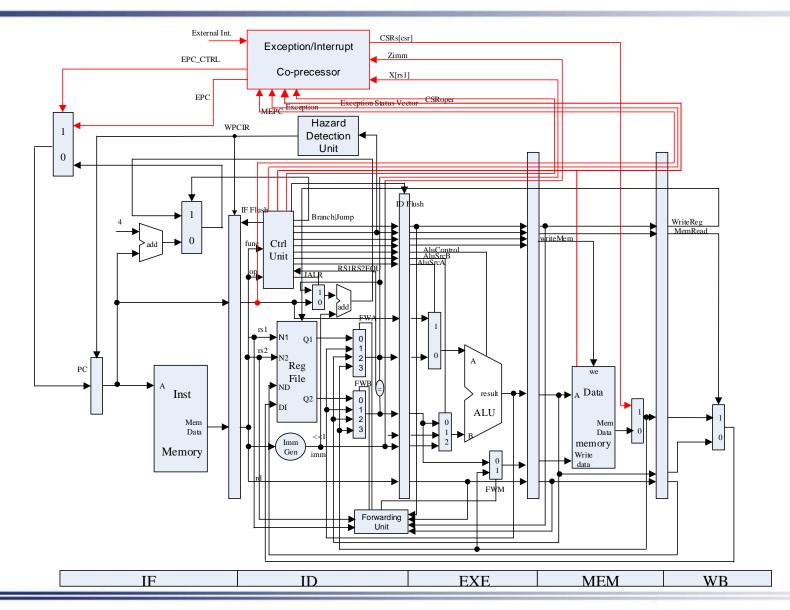


发生异常/中断时,硬件自动经历如下的状态转换:

- 异常指令的PC被保存在mepc中,PC被设置为mtvec。mepc指向导致异常的指令;对于中断,它指向中断处理后应该恢复执行的位置。
- 根据异常来源设置mcause,并将mtval设置为出错的地址或者其它适用于特定异常的信息字。
- 把控制状态寄存器mstatus中的MIE位置零以禁用中断,并把先前的MIE值保留到MPIE中。
- 发生异常之前的权限模式保留在mstatus的MPP域中,再把权限模式更改为M。



Pipelined CPU supporting exception & interrupt





Instr. Mem.(1)

NO.	Instruction	Addr.	Label	ASM	Comment
0	0000013	0	start:	addi x0, x0, 0	
1	00402103	4		lw x2, 4(x0)	
2	00802203	8		lw x4, 8(x0)	
3	00c02283	С		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	
8	306310f3	20		csrrw x1, 0x306, x6	
9	306020f3	24		csrr x1, 0x306	
10	0000013	28		addi x0, x0, 0	
11	07800093	2C		addi x1, x0, 120	
12	30509073	30		csrw 0x305, x1	
13	0000013	34		addi x0, x0, 0	
 14	0000073	38		ecall	





Instr. Mem.(2)

	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, <mark>128</mark> (x0)	# I access fault
	20	00000013	50		addi x0, x0, 0	
	21	08102023	54		sw x1, <mark>128</mark> (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	0000013	68		addi x0, x0, 0	
	27	0000013	6C		addi x0, x0, 0	
	28	00000013	70		addi x0, x0, 0	
4	29	00000067	74		jr x0	





Instr. Mem.(3)

	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	
	37	30200073	94		mret	# 30200073 mret
	38	0000013	98		addi x0, x0, 0	
	39	0000013	9C		addi x0, x0, 0	
	40	0000013	A0		addi x0, x0, 0	
	41	0000013	A4		addi x0, x0, 0	
-						



Data Mem.

NO.	Data	Addr.	Comment	NO.	Instruction	Addr.	Comment
0	000080BF	0		16	00000000	40	
1	8000000	4		17	00000000	44	
2	0000010	8		18	00000000	48	
3	0000014	С		19	00000000	4C	
4	FFFF0000	10		20	A3000000	50	
5	0FFF0000	14		21	27000000	54	
6	FF000F0F	18		22	79000000	58	
7	F0F0F0F0	1C		23	15100000	5C	
8	0000000	20		24	00000000	60	
9	0000000	24		25	00000000	64	
10	0000000	28		26	00000000	68	
11	0000000	2C		27	00000000	6C	
12	0000000	30		28	00000000	70	
13	0000000	34		29	00000000	74	
14	00000000	38		30	00000000	78	
3 , 15	0000000	3C		31	0000000	7C	

Test Bench

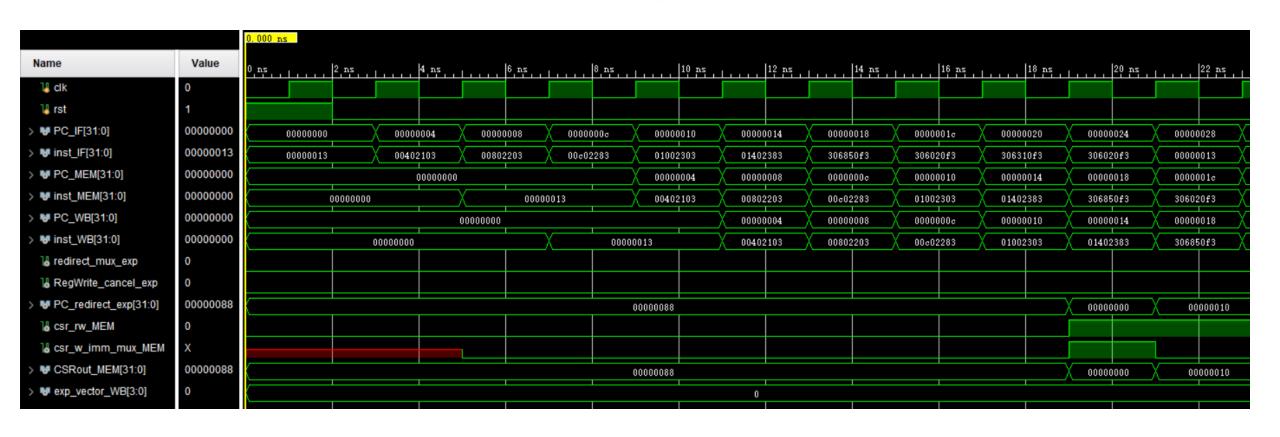


```
RV32core core(
    .debug_en(1'b0),
    .debug_step(1'b0),
    .debug_addr(7'b0),
    .debug_data(),
    .clk(clk),
    .rst(rst),
    .interrupter(1'b0)
 initial begin
    clk = 0;
    rst = 1;
    #2 rst = 0;
 end
 always #1 clk = ^{\sim}clk;
```



Simulation (1)







Simulation (2)



Name	Value	20 ns	22 ns	24 ns	26 n	s 28 m	s 30 ns	32 ns	34 ns	36 ns	38 ns	40 ns	42 ns
¼ clk	0												
¼ rst	1												
▶ ₩ PC_IF[31:0]	00000000	00000024	00000028	0000002c	00000030	00000034	00000038	0000003e	00000040	00000044	00000048	0000004c	00000078
₩ inst_IF[31:0]	00000013	306020 f 3	00000013	07800093	30509073	00000013	00000073	00000013	00000012	00000013	07f02083	08002083	34102cf3
₩ PC_MEM[31:0]	00000000	00000018	0000001c	00000020	00000024	00000028	00000020	00000030	00000034	00000038	0000003e	00000040	00000044
▼ inst_MEM[31:0]	00000000	306850f3	306020f3	306310f3	306020f3	00000013	07800093	30509073	00000013	00000073	00000013	X 00	000000
▶ PC_WB[31:0]	00000000	00000014	00000018	0000001c	00000020	00000024	00000028	00000020	00000030	00000034	00000038	00000030	00000040
▼ inst_WB[31:0]	00000000	01402383	306850f3	306020f3	306310f3	306020f3	00000013	07800093	30509073	00000013	00000073	X 00	000000
<pre>redirect_mux_exp</pre>	0												
RegWrite_cancel_exp	0												
♥ PC_redirect_exp[31:0]	00000088	00000000	00000	0010	X	ffff0000		00000000	Χ		00000078		
d csr_rw_MEM	0												
dia csr_w_imm_mux_MEM	X												
♥ CSRout_MEM[31:0]	00000088	00000000	00000	0010	Χ	ffff0000		00000000	X		00000078		
<pre>exp_vector_WB[3:0]</pre>	0					0					X 4	χ	0



Simulation (3)



Name	Value	40 ns 4	42 ns 44 ns	46 ns	48 ns	50 ns	52 ns	54 ns	56 ns	58 ns	60 ns	62 ns
¹ ₄ clk	0											
¼ rst	1											
> W PC_IF[31:0]	00000000	0000004c 000000	0000007c	00000080	00000084	00000088	0000008c	00000090	00000094	00000098	0000009c	0000000
> W inst_IF[31:0]	00000013	08002083 34102c	f3 34202df3	30002e73	30402ef3	34402f73	004c8113	34111073	30200073	(<u> </u>	00000013	
> W PC_MEM[31:0]	00000000	00000040	00000044		00000078	0000007c	00000080	00000084	00000088	00000086	00000090	00000094
> W inst_MEM[31:0]	00000000		00000000		34102cf3	34202df3	30002e73	30402ef3	34402f73	004c8113	34111073	30200073
> W PC_WB[31:0]	00000000	00000036 000000	040	00000044		00000078	0000007c	00000080	00000084	00000088	0000008c	00000090
> W inst_WB[31:0]	00000000		00000000			34102cf3	34202df3	30002e73	30402ef3	34402f73	004c8113	34111073
redirect_mux_exp	0											
RegWrite_cancel_exp	0											
> W PC_redirect_exp[31:0]	88000000		00000078		00000038	0000000Ъ	00000080	00000fff	0000	0000	00000038	00000036
¼ csr_rw_MEM	0											
d csr_w_imm_mux_MEM	X											
> W CSRout_MEM[31:0]	88000000		00000078		00000038	0000000ь	00000080	00000fff	0000	0000	00000038	00000030
> W exp_vector_WB[3:0]	0						0					



Simulation (4)



Name	Value	60 ns	62 ns	64 ns	66 ns	68 ns	70 ns	72 ns	74 ns	76 ns	78 ns	80 ns	82 ns
¼ clk	0												
¼ rst	1												
> W PC_IF[31:0]	00000000	0000009c	000000a0	0000003c	00000040	00000044	00000048	0000004c	00000050	00000054	00000078	0000007c	00000080
> W inst_IF[31:0]	00000013		00000013		00000012	00000013	07f02083	08002083	00000013	08102023	34102cf3	34202df3	30002e73
> W PC_MEM[31:0]	00000000	00000090	00000094	00000098	00000)09c	00000036	00000040	00000044	00000048	Х	0000004c	X
> W inst_MEM[31:0]	00000000	34111073	30200073		00000000		00000013	00000012	00000013	X	0000	0000	X
> W PC_WB[31:0]	00000000	0000008c	00000090	00000094	00000098	0(100009c	0000003c	00000040	00000044	00000048	00	00004c
> W inst_WB[31:0]	00000000	004c8113	34111073	30200073		00000000		00000013	00000012	XX	000	00000	
<pre>Tedirect_mux_exp</pre>	0												
RegWrite_cancel_exp	0												
> W PC_redirect_exp[31:0]	00000088	00000038	X			0000003c				X	0000	0078	X
la csr_rw_MEM	0												
lacsr_w_imm_mux_MEM	X												
> W CSRout_MEM[31:0]	00000088	00000038	X			0000003c				Χ	0000	0078	X
> W exp_vector_WB[3:0]	0				0				8	X		0	



Simulation (5)



Name	Value	80 ns	1	82 ns	84.	ns	86 ns	8	8 ns	90 ns	1	92 ns	94	ns	96 ns		98 ns	100	ns 1	102 ns
¹ dk	0																			
¼ rst	1																			
> W PC_IF[31:0]	00000000	0000007c	00000	080	00000084	X 0	0000088	000000	8c X	00000090	00000	094	00000098	X 00	00009c	00000	0a0 X	00000044	000000	048
> W inst_IF[31:0]	00000013	34202df3	30002	e73	30402ef3	3-	1402f73	004c81	13	34111073	30200	73			000	00013			07f020	083
> W PC_MEM[31:0]	00000000	00	00004c		00000078	\	000007c	000000	80	00000084	00000	088	0000008c	X 00	0000090	00000	094	00000098	00000	009c
> W inst_MEM[31:0]	00000000		000000		34102cf3	3-	1202df3	30002e	73	30402ef3	34402	£73	004c8113	34	111073	30200	073		00000000	
> W PC_WB[31:0]	00000000		000	0004c		\	0000078	000000	7c	00000080	00000	084	00000088	X 00	000008c	00000	090	00000094	000000	098
> W inst_WB[31:0]	00000000		000	00000		3-	1102cf3	342026	f3	30002e73	30402	ef3	34402f73	X 00	4c8113	34111	073	30200073	00000	0000
aredirect_mux_exp	0																			
RegWrite_cancel_exp	0																			
> W PC_redirect_exp[31:0]	00000088	00	0000078		00000040	X 0	0000002	000000	80	00000fff	Х	0000000	0	X 00	0000040	X		00000044		
d csr_rw_MEM	0																			
d csr_w_imm_mux_MEM	X																			
> W CSRout_MEM[31:0]	88000000	00	000078		00000040	\	0000002	000000	80	00000fff	X	0000000	0	X 00	0000040	X		00000044		
> W exp_vector_WB[3:0]	0										0									



Simulation (6)



Name	Value	100 ns	102 ns	10	4 ns.	106 ns	1	108 ns		110 ns		112 ns		114 ns		116 ns		118 ns	1	120 ns		122 ns. 1
¹ dk	0																					
¼ rst	1																					
> W PC_IF[31:0]	00000000	00000044	00000048	0000004	c 00	000050	00000	054	00000	058	00000	05c	0000	0060	00000	0078	0000	007c	00000	080	00000	084
> W inst_IF[31:0]	00000013	00000013	07f02083	0800208	3 00	000013	08102	2023			00000	013			34102	2cf3	3420	2df3	30002	e73	30402	ef3
> W PC_MEM[31:0]	00000000	00000098	0000	009c	00	000044	00000	048	00000	04c	00000	050	0000	0054			0000	0058			00000	078
> W inst_MEM[31:0]	00000000		00000000		00	000013	07f0	2083	08002	083	00000	013				00000	000				34102	cf3
> W PC_WB[31:0]	00000000	00000094	00000098	X	0000009c		00000	044	00000	048	00000	04c	0000	0050	00000	0054			00000	058		\square X
> W inst_WB[31:0]	00000000	30200073	Χ	0000000	0		00000	013	07f02	083	08002	2083					0000	0000				X
<pre>redirect_mux_exp</pre>	0																					
RegWrite_cancel_exp	0																					
> W PC_redirect_exp[31:0]	00000088					00000044										00000	078				00000	04c
d csr_rw_MEM	0																					
csr_w_imm_mux_MEM	Х																					
> W CSRout_MEM[31:0]	00000088					00000044										00000	078				00000	04c
> W exp_vector_WB[3:0]	0				0						[2							0				



Simulation (7)



Name	Value	120 ns	122 ns	124 ns	126 ns	128 ns	130 ns	132 ns	134 ns	136 ns	138 ns	140 ns	142 ns
¼ dk	0												
¼ rst	1												
> W PC_IF[31:0]	00000000	00000080	00000084	00000088	0000008c	00000090	00000094	00000098	0000009c	000000a0	00000050	00000054	00000058
> W inst_IF[31:0]	00000013	30002e73	30402ef3	34402f73	004c8113	34111073	30200073		00000	013		08102023	00000013
> W PC_MEM[31:0]	00000000	00000058	00000078	0000007c	00000080	00000084	00000088	0000008c	00000090	00000094	00000098	00000	09c
> W inst_MEM[31:0]	00000000	00000000	34102cf3	34202df3	30002e73	30402ef3	34402f73	004c8113	34111073	30200073	X	00000000	X
> W PC_WB[31:0]	00000000	00	000058	00000078	0000007c	00000080	00000084	00000088	0000008e	00000090	00000094	00000098	0000009e
> W inst_WB[31:0]	00000000	00	000000	34102cf3	34202df3	30002e73	30402ef3	34402f73	004c8113	34111073	30200073	0000	0000
<pre>redirect_mux_exp</pre>	0												
RegWrite_cancel_exp	0												
> W PC_redirect_exp[31:0]	8800000	00000078	0000004c	00000005	00000080	00000fff	00000	000	0000004c		000	00050	
I csr_rw_MEM	0												
d csr_w_imm_mux_MEM	X												
> W CSRout_MEM[31:0]	88000000	00000078	0000004c	00000005	00000080	00000fff	00000	000	0000004c		000	00050	
> W exp_vector_WB[3:0]	0							0					



Simulation (8)



Name	Value	140	ns. I .	1	142 ns		144 ns		146	ns. I		148 ns		150 ns		152 ns		15	4 ns. 1		156 ns	1	158 ns	1	160 ns		162 ns
¹ ₄ dk	0																										
¼ rst	1																										
> W PC_IF[31:0]	00000000	0000005	4	00000	058	0000	005c	X 00	000060	X	0000	0064	000	0068	000	00078	χ ,	000007	·	0000	0800	0000	0084	0000	10088	0000	008c
> 😽 inst_IF[31:0]	00000013	0810202	3					00	000013	}					341	02cf3	3	4202df	3	3000	12e73	3040	2ef3	3440	2f73	004c	8113
> W PC_MEM[31:0]	00000000		00000	009c		0000	0050	X 00	000054	X	0000	0058	000	005c	X		0	000006	0			0000	0078	0000	1007c	0000	0080
> W inst_MEM[31:0]	00000000		00000	0000		0000	00013	08	102023	; X	0000	0013	X			000	00000					3410	2cf3	3420	2df3	3000	2e73
> W PC_WB[31:0]	00000000	0000009	8		0000	009c		00	000050	\mathbf{x}	0000	0054	000	0058	000	0005c	X			0000	10060			0000	10078	0000	007c
> W inst_WB[31:0]	00000000			000	00000			X 00	000013	X	0810	2023	X				0	000000	10					3410	2cf3	3420	2df3
redirect_mux_exp	0																										
RegWrite_cancel_exp	0																										
> W PC_redirect_exp[31:0]	8800000					0	0000050						X			000	00078					0000	0054	0000	0007	0000	0080
I csr_rw_MEM	0																										
csr_w_imm_mux_MEM	X																										
> W CSRout_MEM[31:0]	8800000					0	0000050						X			000	00078					0000	0054	0000	0007	0000	0080
> W exp_vector_WB[3:0]	0					0				$=$ χ		1	X								0						



Simulation (9)



Name	Value	160 ns	162 ns	164 ns	166 ns	168 ns	170 ns	172 ns	174 ns	176 ns	178 ns	is 182 ns
¹ ₄ dk	0											
¼ rst	1											
> W PC_IF[31:0]	00000000	00000088	0000008c	00000090	00000094	00000098	0000009c	000000a0	00000058	0000005c 00000	0000064	00000068
> W inst_IF[31:0]	00000013	34402f73	004c8113	34111073	30200073	Χ			0000	00013		
> W PC_MEM[31:0]	00000000	0000007c	00000080	00000084	00000088	0000008c	00000090	00000094	00000098	0000009c	00000058	00000050
> W inst_MEM[31:0]	00000000	34202df3	30002e73	30402ef3	34402f73	004c8113	34111073	30200073	X	00000000	X	00000013
> W PC_WB[31:0]	00000000	00000078	0000007c	00000080	00000084	00000088	0000008c	00000090	00000094	00000098	0000009c	00000058
> W inst_WB[31:0]	00000000	34102cf3	34202df3	30002e73	30402ef3	34402f73	004c8113	34111073	30200073	00000	0000	00000013
Tedirect_mux_exp Tedirect_mux_exp	0											
RegWrite_cancel_exp	0											
> W PC_redirect_exp[31:0]	88000000	00000007	00000080	00000fff	0000	0000	00000054			00000058		
d csr_rw_MEM	0											
¹⊌ csr_w_imm_mux_MEM	X											
> W CSRout_MEM[31:0]	88000000	00000007	00000080	00000fff	0000	0000	00000054			00000058		
> W exp_vector_WB[3:0]	0							0				



Checkpoints



• CP 1:

Waveform Simulation of the Pipelined CPU with the verification program

• CP 2:

FPGA Implementation of the Pipelined CPU with the verification program





Thanksl

