RISC-V Pipelined Architecture III

Muhammad Tahir

Lecture 12

Electrical Engineering Department
University of Engineering and Technology Lahore

Contents

Exceptions and Interrupts

- **Exception**: An *internal* event caused by program execution fault (e.g., page fault, misaligned memory access, arithmetic over-/under-flow)
- **Interrupt**: An *external* event caused outside of program execution (e.g. interrupt from a peripheral device UART, timer, Network Interface etc.)
- **Trap**: It is the forced transfer of execution control to supervisor/service-routine caused by exception or interrupt (not all exceptions/interrupts cause traps)

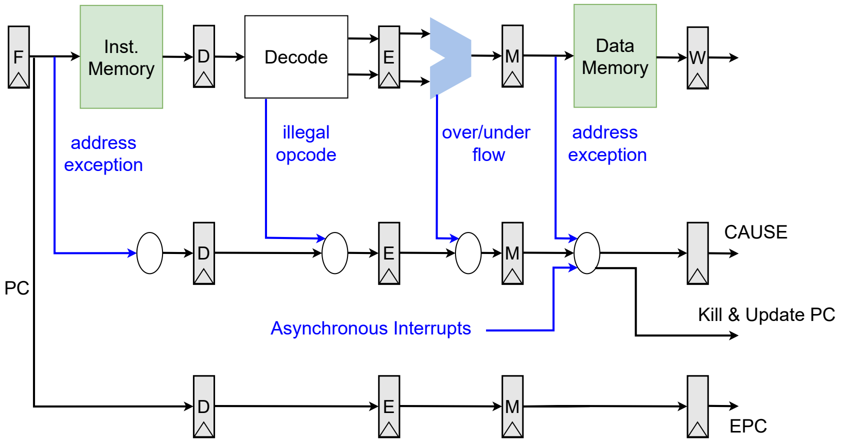
Exception and Interrupt Handling

- Exceptions are *synchronous* events
- Interrupts are *asynchronous* events
- How and where to handle multiple exceptions at different pipeline stages?
- How and where to handle asynchronous external interrupts

Exception and Interrupt Handling

- Exceptions are *synchronous* events
- Interrupts are *asynchronous* events
- How and where to handle multiple exceptions at different pipeline stages?
- How and where to handle asynchronous external interrupts
- Exceptions and interrupts are handled synchronously using trap handlers
- Exceptions and interrupts are handled at memory (or commit) stage

Exception and Interrupt Handling Cont'd



Exception and Interrupt Handling Cont'd

- Exception flags are held in pipeline till memory stage (or commit point)
- Interrupts (external events) are injected directly, overriding others, at memory stage
- At memory stage update Cause and EPC registers, kill all stages, update PC with trap handler address

Trap Handler

- A Trap Handler at the start:
  - Saves EPC before enabling interrupts to allow nested interrupts
  - Reads the Cause register to find the source of exception/interrupt
- A Trap Handler at the end:
  - Uses a special instruction ERET (indirect jump) to resume normal execution
  - Enables the interrupts
  - Restores the processor status

Trap/Interrupt Setup CSRs

Table 1: User, supervisor and machine mode trap/interrupt setup CSRs.

U Mode	S Mode	M Mode	Description
ustatus	sstatus	mstatus	Status register.
uie	sie	mie	Interrupt-enable register.
utvec	stvec	mtvec	Trap-handler base address.
	sedeleg	medeleg	Exception delegation register.
	sideleg	mideleg	Interrupt delegation register.
	scounteren	mcounteren	Counter enable.
		misa	ISA and extensions.

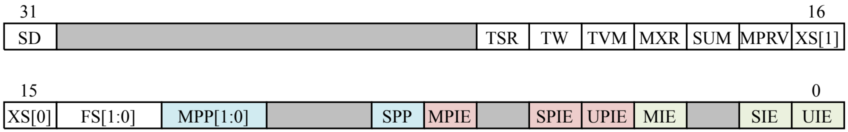
Trap Handling CSRs

Table 2: User, supervisor and machine mode trap handling CSRs.

U Mode	S Mode	M Mode	Description
uscratch	sscratch	mscratch	Scratch register for trap handlers.
uepc	sepc	mepc	Exception program counter.
ucause	scause	mcause	Trap cause.
utval	stval	mtval	Bad address or instruction.
uip	sip	mip	Interrupt pending.

Trap/Interrupt Setup CSRs: `mstatus`

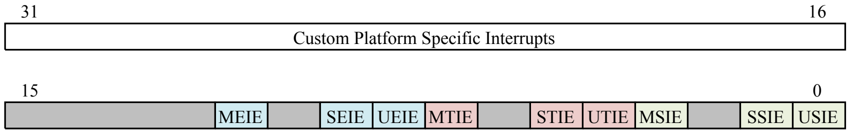
Table 3: Machine status register bit field descriptions.



Bit field	Description
MIE, SIE, UIE	Interrupt-enable bits for each privilege mode. MIE for machine mode, SIE for supervisor mode, and UIE for user mode.
MPIE, SPIE, UPIE	*x*PIE holds the value of the interrupt-enable bit active prior to the trap, where $x \in \{\mathrm{M}, \mathrm{S}, \mathrm{U}\}$.
MPP, SPP	*x*PP holds the previous privilege mode prior to the trap and can only hold privilege modes up to *x*. So MPP is two bits wide, SPP is one bit wide, and UPP is implicitly of bit width zero.

Trap/Interrupt Setup CSRs: `mie`

Table 4: Machine interrupt enable register bit field descriptions.



Bit field	Description
MSIE, SSIE, USIE	These fields enable software interrupts for M-mode, S-mode and U-mode, respectively.
MTIE, STIE, UTIE	These fields enable timer interrupts for M-mode, S-mode, and U-mode, respectively.
MEIE, SEIE, UEIE	These fields enable external interrupts for M-mode, S-mode, and U-mode, respectively.

Trap/Interrupt Setup CSRs: `mtvec`

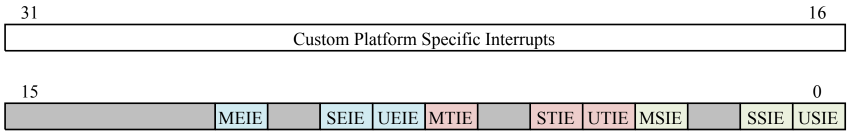
Table 5: Machine trap-handler base address register bit field descriptions.

31 — 2	0
BASE[31:2]	MODE

Bit field	Description
MODE	This two bit field can be configured to following possible values: • A value of `0` corresponds to `Direct` mode. All exceptions set `pc` to `BASE` field. • A value of `1` corresponds to `Vectored` mode. All exceptions set `pc` to (`BASE` + cause $<<$ 2). • Reserved for other values.
BASE[31:2]	This field corresponds to 30 (most significant) bits of the 32-bit (word aligned) `BASE` address.

Trap/Interrupt Handling CSRs: `mip`

Table 6: Machine interrupt pending register bit field descriptions.



Bit field	Description
MSIP, SSIP, USIP	These fields correspond to software pending interrupts for M-mode, S-mode and U-mode, respectively.
MTIP, STIP, UTIP	These fields correspond to timer pending interrupts for M-mode, S-mode, and U-mode, respectively.
MEIP, SEIP, UEIP	These fields correspond to external pending interrupts for M-mode, S-mode, and U-mode, respectively.

Trap/Interrupt Handling CSRs: `mcause`

Table 7: Machine trap cause register bit field descriptions.

31	0
INTR	Exception Code

Bit field	Description
INTR	The INTR (Interrupt) bit is set if the trap was caused by an interrupt and is cleared in case of other traps (e.g. address fault).
Exception Code	This field contains a code identifying the last exception.

Trap/Interrupt Handling CSRs: Exception Codes

Table 8: Machine cause register bit field values after trap.

INTR	Exception Code	Description
1	0	User software interrupt
1	1	Supervisor software interrupt
1	2	Reserved
1	3	Machine software interrupt
1	4	User timer interrupt
1	5	Supervisor timer interrupt
1	6	Reserved
1	7	Machine timer 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
⋮	⋮	⋮

Example Startup File

```
.equ CSR_MSTATUS, 0x300
.equ MSTATUS_MIE, 0x00000008
.equ CSR_MTVEC,   0x305

 # Main interrupt vector table entries
.global vtable
.type vtable, %object
.section .text.vector_table,"a",%progbits

# this entry is to align reset_handler at address 0x04
  .word     0x00000013
  j         reset_handler
  .align    2
vtable:
  j         default_interrupt_handler
  .word     0
  .word     0
  j         msip_handler
  .word     0
  .word     0
```

Example Startup File Cont'd

```
    .word     0
    j         mtip_handler
    .word     0
    .word     0
    .word     0
    .word     0
    .word     0
    .word     0
    .word     0
    .word     0
    j         user_handler
    .word     0
    .word     0
```

Example Startup File Cont'd

```
# Weak aliases to point each exception handler to the
# 'default_interrupt_handler', unless the application defines
# a function with the same name to override the reference.

  .weak msip_handler
  .set  msip_handler, default_interrupt_handler
  .weak mtip_handler
  .set  mtip_handler, default_interrupt_handler
  .weak user_handler
  .set  user_handler, default_interrupt_handler

# Assembly 'reset handler' function to initialize core CPU registers
    .
.section .text.default_interrupt_handler,"ax",%progbits

.global reset_handler
.type reset_handler,@function

reset_handler:
# Set mstatus bit MIE = 1 (enable M mode interrupts)
  li      t0, 8
  csrrs   zero, CSR_MSTATUS, t0
```

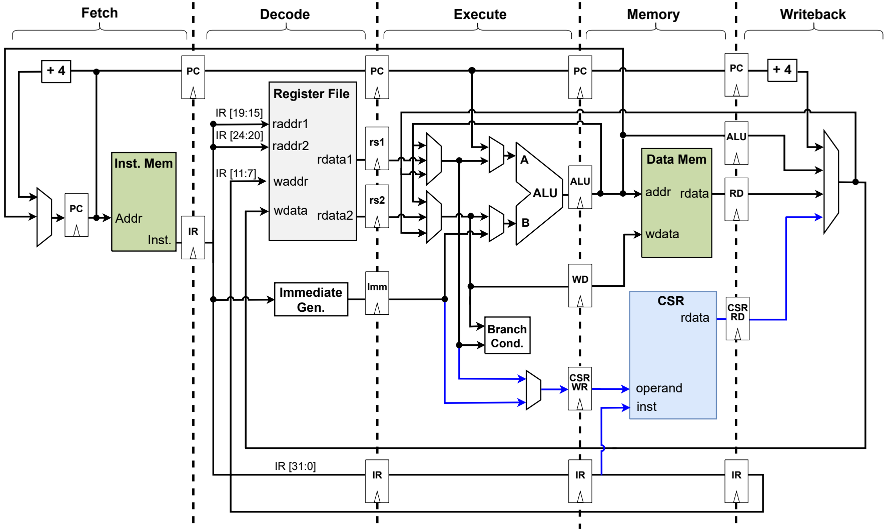
Example Startup File Cont'd

```
# Load the initial stack pointer value.
  la   sp, _sp

# Set the vector table's base address.
  la   a0, vtable
  addi a0, a0, 1
  csrw CSR_MTVEC, a0

# Call user 'main(0,0)' (.data/.bss sections initialized there)
  li   a0, 0
  li   a1, 0
  call main

# A 'default' handler, in case an interrupt triggers without its
    handler defined
default_interrupt_handler:
  j default_interrupt_handler
```

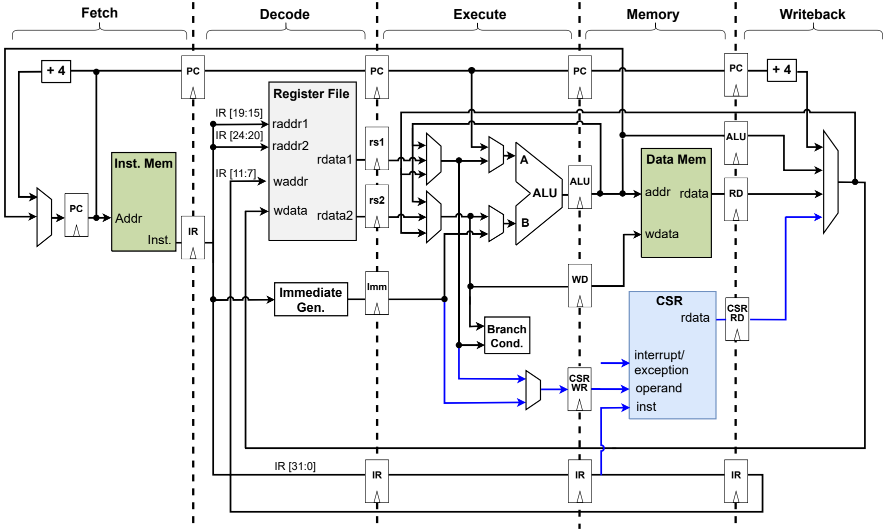
Example Interrupt Service Routine

```
# RISC-V Interrupt Service Routines (ISRs)
# ALL supported ISRs should be put here

.section .text.isr

# User interrupt handler
.globl user_handler
user_handler:
  nop
  # you can call user ISR here and then return using 'mret'
  mret
```

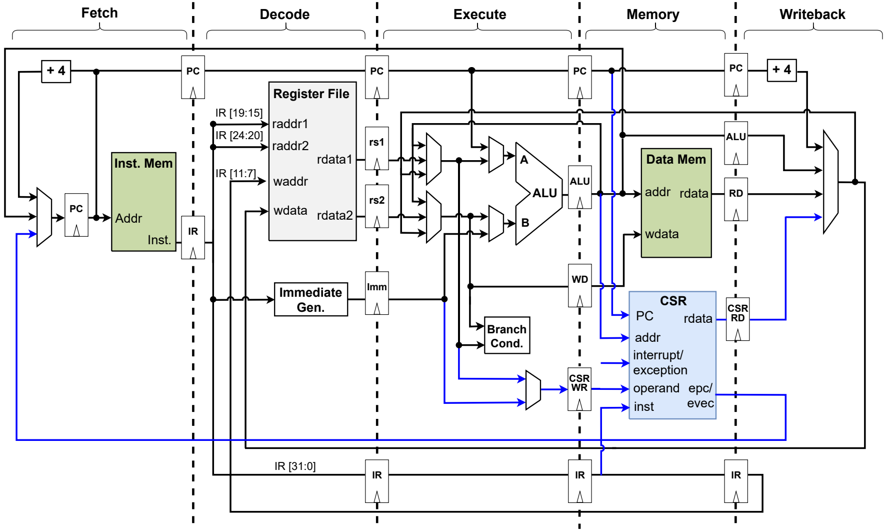
Pipelined Datapath with Exception/Interrupt Handling



Pipelined Datapath with Exception/Interrupt Handling



Pipelined Datapath with Exception/Interrupt Handling



Suggested Reading

- Read relevant sections of Chapter 4 of [Patterson and Hennessy, 2021].
- For Control and Status register description consult Privileged architecture manual [Waterman et al., 2016].
- Read Section 5.14 of [Patterson and Hennessy, 2021].

Acknowledgment

- Preparation of this material was partly supported by Lampro Mellon Pakistan.

References

Patterson, D. and Hennessy, J. (2021).
Computer Organization and Design RISC-V Edition: The Hardware Software Interface, 2nd Edition.
Morgan Kaufmann.

Waterman, A., Lee, Y., Avizienis, R., Patterson, D. A., and Asanovic, K. (2016).
The risc-v instruction set manual volume ii: Privileged architecture version 1.9.
EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-129.