RISC-V Privileged Architecture

Muhammad Tahir

Lecture 13

Electrical Engineering Department
University of Engineering and Technology Lahore

Contents

CSR Instructions: I Type

- Format: I-type

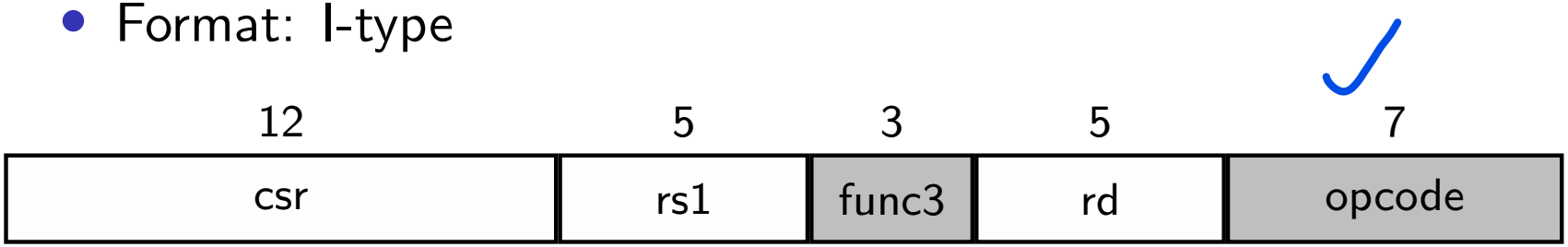
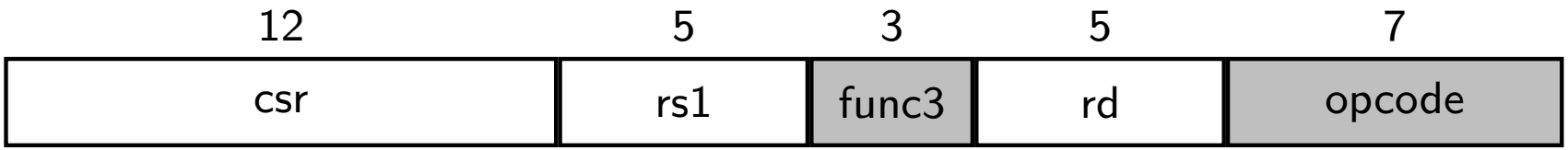


Table 1: I type CSR instructions.

Instruction	Operation	Type	Illustration
CSRRW	Atomic Read and Write	I	CSRRW rd, csr, rs1
CSRRS	Atomic Read and Set bit	I	CSRRS rd, csr, rs1
CSRRC	Atomic Read and Clear bit	I	CSRRC rd, csr, rs1
CSRRWI	Atomic R/W Imm	I	CSRRWI rd, csr, imm
CSRRSI	Atomic Read and Set bit Imm	I	CSRRSI rd, csr, imm
CSRRCI	Atomic Read and Clear bit Imm	I	CSRRCI rd, csr, imm

CSR Instructions: I Type Cont'd

- Format: I-type



- `opcode` = csr $\leftarrow$ rs1, rd $\leftarrow$ csr
  `func3` = CSRRW

- Pseudo-instructions to read and write a CSR
  - `CSRR` csr, rd (funct3 = CSRRS, rs1 = x0): rd $\leftarrow$ csr
  - `CSRW` rs1, csr (funct3 = CSRRW, rd = x0): csr $\leftarrow$ rs1

Special Instructions: Environment Calls

- Format: I-type
- Used to generate an exception to invoke Operating System (ECALL) and debugger (EBREAK)

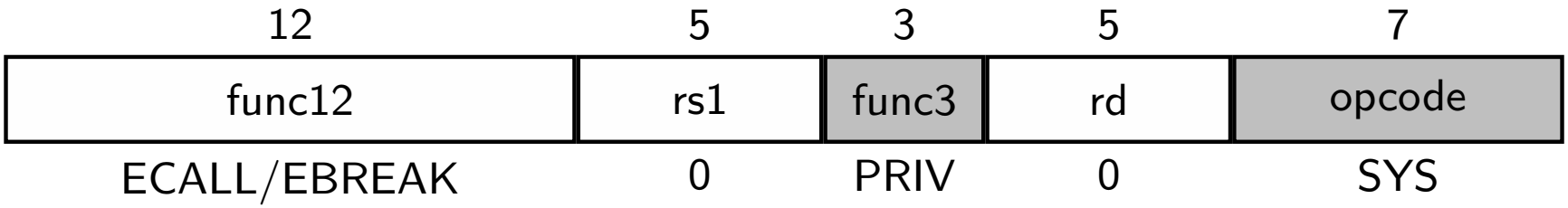


Table 2: Environment Call Instructions.

Instruction	Operation	Type	Illustration
ECALL	Environment Call	I	ECALL
EBREAK	Environment Break	I	EBREAK

Special Instructions: FENCE

- Format: I-type
- Provides (memory) synchronization barriers for data memory (fence) and instruction memory (FENSE.I)

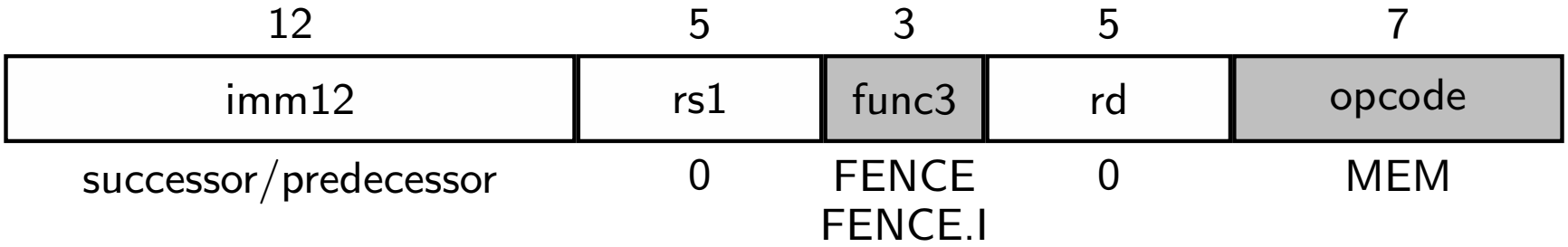
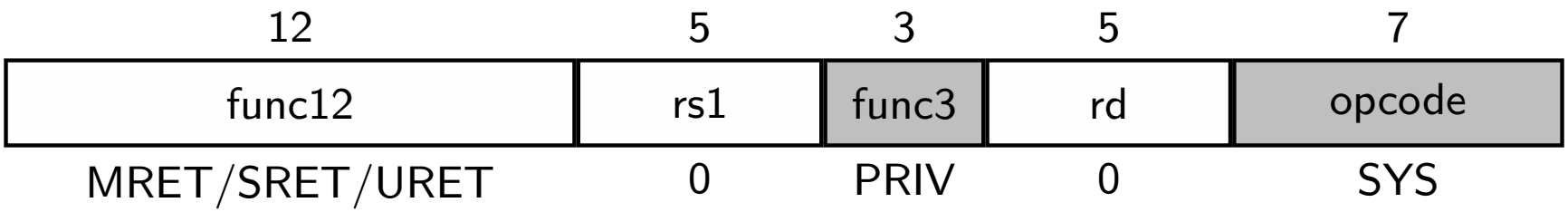


Table 3: Synch Instructions.

Instruction	Operation	Type	Illustration
FENCE	Synchronize data accesses	I	fence
FENCE.I	Synchronize instruction fetches	I	fence.i

Special Instructions: Trap-Return Instructions

- Format: I-type
- `URET` returns from an exception handler in user mode
- `SRET` returns from an exception handler in supervisor mode
- `MRET` returns from an exception handler in machine mode



RISC V Privilege Levels

- Privilege levels are used to provide protection between different components of the software stack
- At a given time, a RISC-V hardware thread (hart) is running at some privilege level
- Privilege level is encoded as a mode in one or more CSRs (control status registers)

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

Table 4: RISC V privilege levels.

RISC V Privilege Levels Cont'd

- Supported modes at different privilege levels

No. of Level	Supported Modes	Usage
1	M	For simple embedded systems
2	M, U	For secure embedded systems
3	M, S, U	For systems running Linux type operating systems

Table 5: Combinations of privilege levels that can be supported.

CSRs Address Allocation

CSR Address [11:10]	CSR Address [9:8]	CSR Address [7:4]	Hex	Use and Accessibility
User CSRs				
00	00	XXXX	0x000-0x0FF	Standard read/write
01	00	XXXX	0x400-0x4FF	Standard read/write
10	00	XXXX	0x800-0x8FF	Custom read/write
11	00	0XXX	0xC00-0xC7F	Standard read-only
11	00	10XX	0xC80-0xCBF	Standard read-only
11	00	11XX	0xCC0-0xCFF	Custom read-only
Supervisor CSRs				
00	01	XXXX	0x100-0x1FF	Standard read/write
01	01	0XXX	0x500-0x57F	Standard read/write
01	01	10XX	0x580-0x5BF	Standard read/write
01	01	11XX	0x5C0-0x5FF	Custom read/write
10	01	0XXX	0x900-0x97F	Standard read/write
10	01	10XX	0x980-0x9BF	Standard read/write
10	01	11XX	0x9C0-0x9FF	Custom read/write
11	01	0XXX	0xD00-0xD7F	Standard read-only
11	01	10XX	0xD80-0xDBF	Standard read-only
11	01	11XX	0xDC0-0xDFF	Custom read-only

⋮

Figure 1: CSR groups with address allocation [Waterman et al., 2016a].

CSRs Address Allocation Cont'd

⋮

Hypervisor CSRs				
00	10	XXXX	0x200-0x2FF	Standard read/write
01	10	0XXX	0x600-0x67F	Standard read/write
01	10	10XX	0x680-0x6BF	Standard read/write
01	10	11XX	0x6C0-0x6FF	Custom read/write
10	10	0XXX	0xA00-0xA7F	Standard read/write
10	10	10XX	0xA80-0xABF	Standard read/write
10	10	11XX	0xAC0-0xAFF	Custom read/write
11	10	0XXX	0xE00-0xE7F	Standard read-only
11	10	10XX	0xE80-0xEBF	Standard read-only
11	10	11XX	0xEC0-0xEFF	Custom read-only
Machine CSRs				
00	11	XXXX	0x300-0x3FF	Standard read/write
01	11	0XXX	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	0XXX	0xB00-0xB7F	Standard read/write
10	11	10XX	0xB80-0xBBF	Standard read/write
10	11	11XX	0xBC0-0xBFF	Custom read/write
11	11	0XXX	0xF00-0xF7F	Standard read-only
11	11	10XX	0xF80-0xFBF	Standard read-only
11	11	11XX	0xFC0-0xFFF	Custom read-only

Figure 2: CSR groups with address allocation [Waterman et al., 2016a].

Machine Level CSRs

Number	Privilege	Name	Description
		Machine Information Registers	
0xF11	MRO	mvendorid	Vendor ID.
0xF12	MRO	marchid	Architecture ID.
0xF13	MRO	mimpid	Implementation ID.
0xF14	MRO	mhartid	Hardware thread ID.
		Machine Trap Setup	
0x300	MRW	mstatus	Machine status register.
0x301	MRW	misa	ISA and extensions
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.
		Machine Trap Handling	
0x340	MRW	mscratch	Scratch register for machine trap handlers.
0x341	MRW	mepc	Machine exception program counter.
0x342	MRW	mcause	Machine trap cause.
0x343	MRW	mtval	Machine bad address or instruction.
0x344	MRW	mip	Machine interrupt pending.
		Machine Memory Protection	
0x3A0	MRW	pmpcfg0	Physical memory protection configuration.
0x3A1	MRW	pmpcfg1	Physical memory protection configuration, RV32 only.
0x3A2	MRW	pmpcfg2	Physical memory protection configuration.
0x3A3	MRW	pmpcfg3	Physical memory protection configuration, RV32 only.
0x3B0	MRW	pmpaddr0	Physical memory protection address register.
0x3B1	MRW	pmpaddr1	Physical memory protection address register.
		⋮	
0x3BF	MRW	pmpaddr15	Physical memory protection address register.

ie, ip (globally)

ie (locally)

where pc will go when an interrupt comes

ip (locally)

Figure 3: Machine level CSRs [Waterman et al., 2016a].

Machine Level CSRs Cont'd

Number	Privilege	Name	Description
Machine Counter/Timers			
0xB00	MRW	mcycle	Machine cycle counter.
0xB02	MRW	minstret	Machine instructions-retired counter.
0xB03	MRW	mhpmcounter3	Machine performance-monitoring counter.
0xB04	MRW	mhpmcounter4	Machine performance-monitoring counter.
		⋮	
0xB1F	MRW	mhpmcounter31	Machine performance-monitoring counter.
0xB80	MRW	mcycleh	Upper 32 bits of mcycle, RV32I only.
0xB82	MRW	minstreth	Upper 32 bits of minstret, RV32I only.
0xB83	MRW	mhpmcounter3h	Upper 32 bits of mhpmcounter3, RV32I only.
0xB84	MRW	mhpmcounter4h	Upper 32 bits of mhpmcounter4, RV32I only.
		⋮	
0xB9F	MRW	mhpmcounter31h	Upper 32 bits of mhpmcounter31, RV32I only.
Machine Counter Setup			
0x320	MRW	mcountinhibit	Machine counter-inhibit register.
0x323	MRW	mhpmevent3	Machine performance-monitoring event selector.
0x324	MRW	mhpmevent4	Machine performance-monitoring event selector.
		⋮	
0x33F	MRW	mhpmevent31	Machine performance-monitoring event selector.
Debug/Trace Registers (shared with Debug Mode)			
0x7A0	MRW	tselect	Debug/Trace trigger register select.
0x7A1	MRW	tdata1	First Debug/Trace trigger data register.
0x7A2	MRW	tdata2	Second Debug/Trace trigger data register.
0x7A3	MRW	tdata3	Third Debug/Trace trigger data register.
Debug Mode Registers			
0x7B0	DRW	dcsr	Debug control and status register.
0x7B1	DRW	dpc	Debug PC.
0x7B2	DRW	dscratch0	Debug scratch register 0.
0x7B3	DRW	dscratch1	Debug scratch register 1.

Figure 4: Machine level CSRs [Waterman et al., 2016a].

Trap/Interrupt Setup CSRs

Table 6: 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.

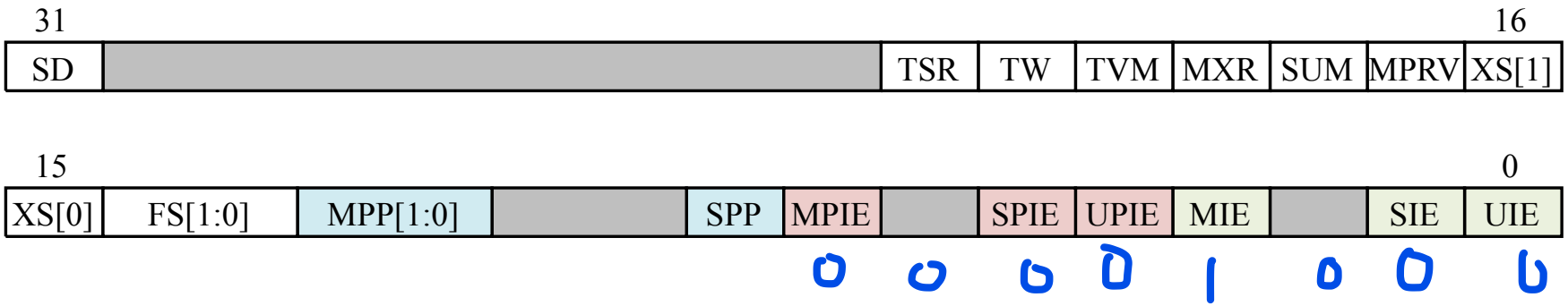
Trap Handling CSRs

Table 7: 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`

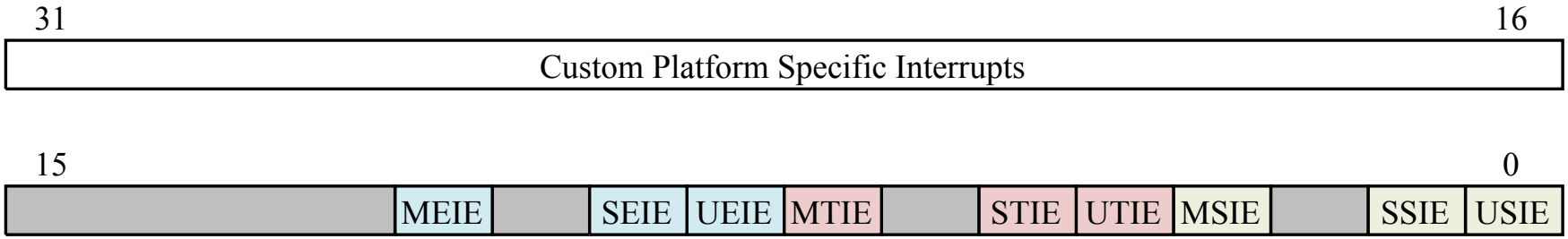
Table 8: Machine status register bit field descriptions.



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	xPIE holds the value of the interrupt-enable bit active prior to the trap, where $x \in \{\mathrm{M}, \mathrm{S}, \mathrm{U}\}$.
MPP, SPP	xPP 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`

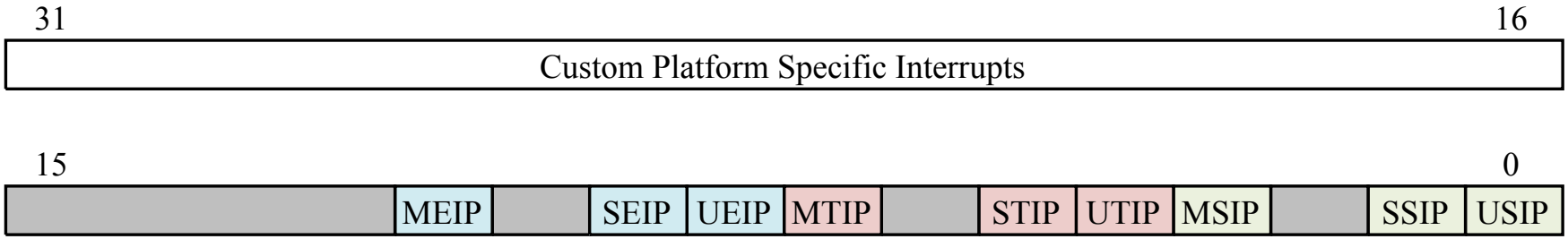
Table 9: Machine interrupt enable register bit field descriptions.



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 Handling CSRs: `mip`

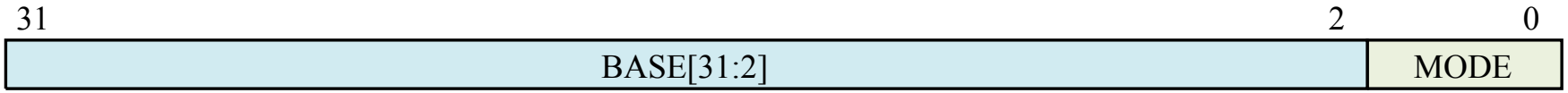
Table 10: Machine interrupt pending register bit field descriptions.



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 Setup CSRs: `mtvec`

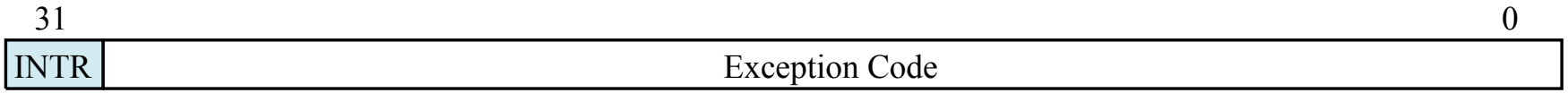
Table 11: Machine trap-handler base address register bit field descriptions.



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: `mcause`

Table 12: Machine trap cause register bit field descriptions.



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 13: 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
⋮	⋮	⋮

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)

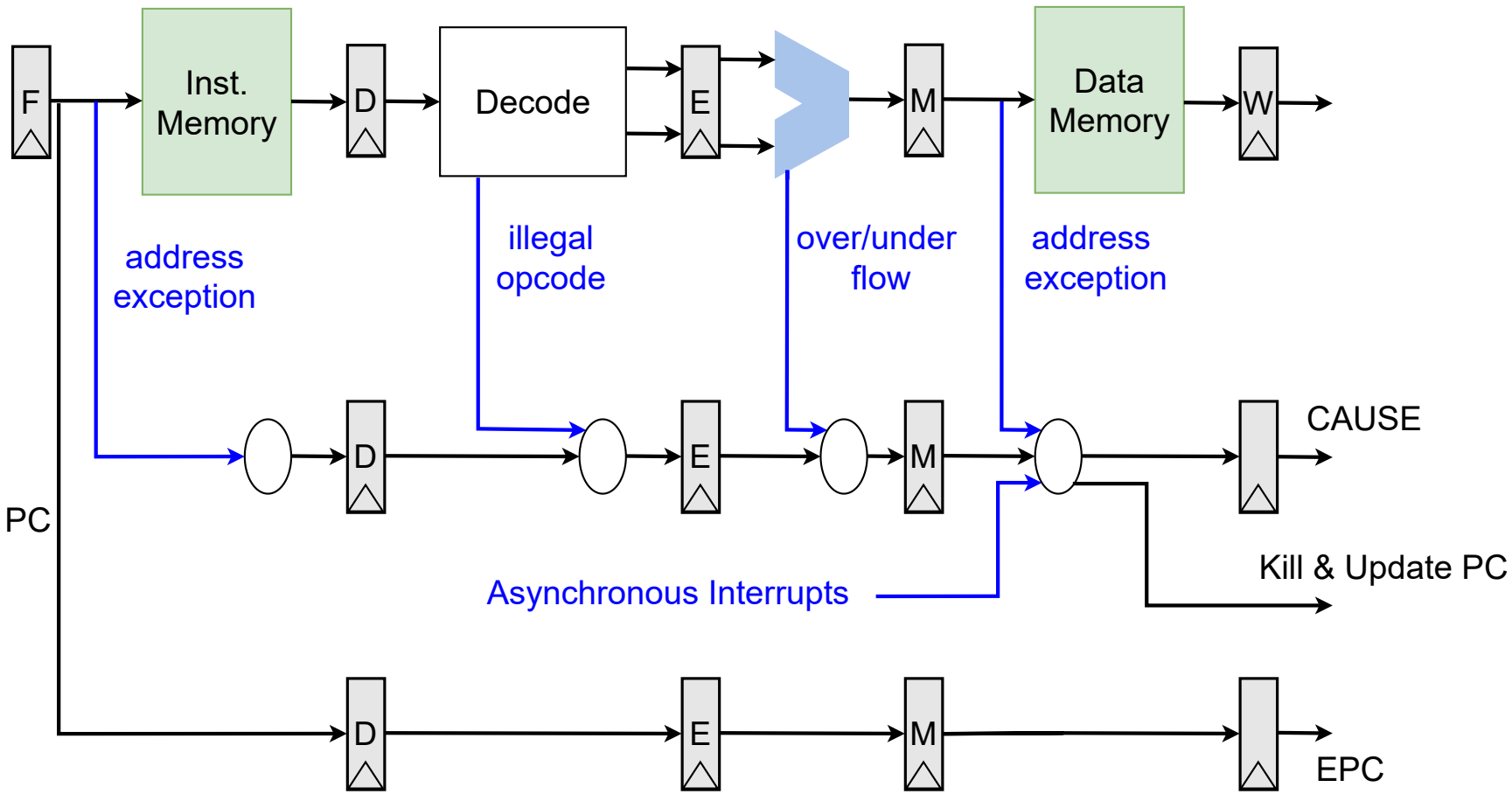
Exception and Interrupt Handling

- 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



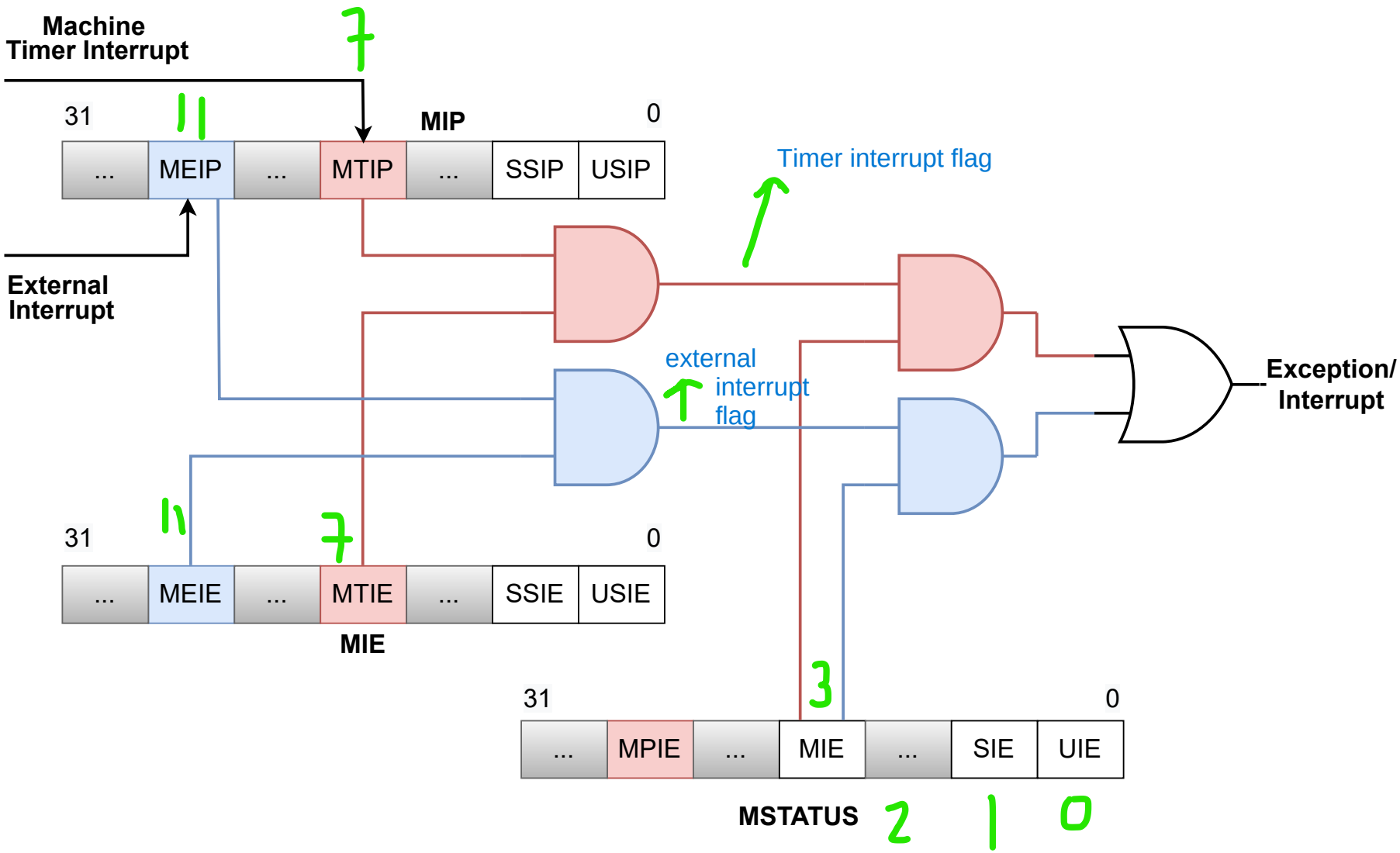
Exception and Interrupt Handling Cont'd

- 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

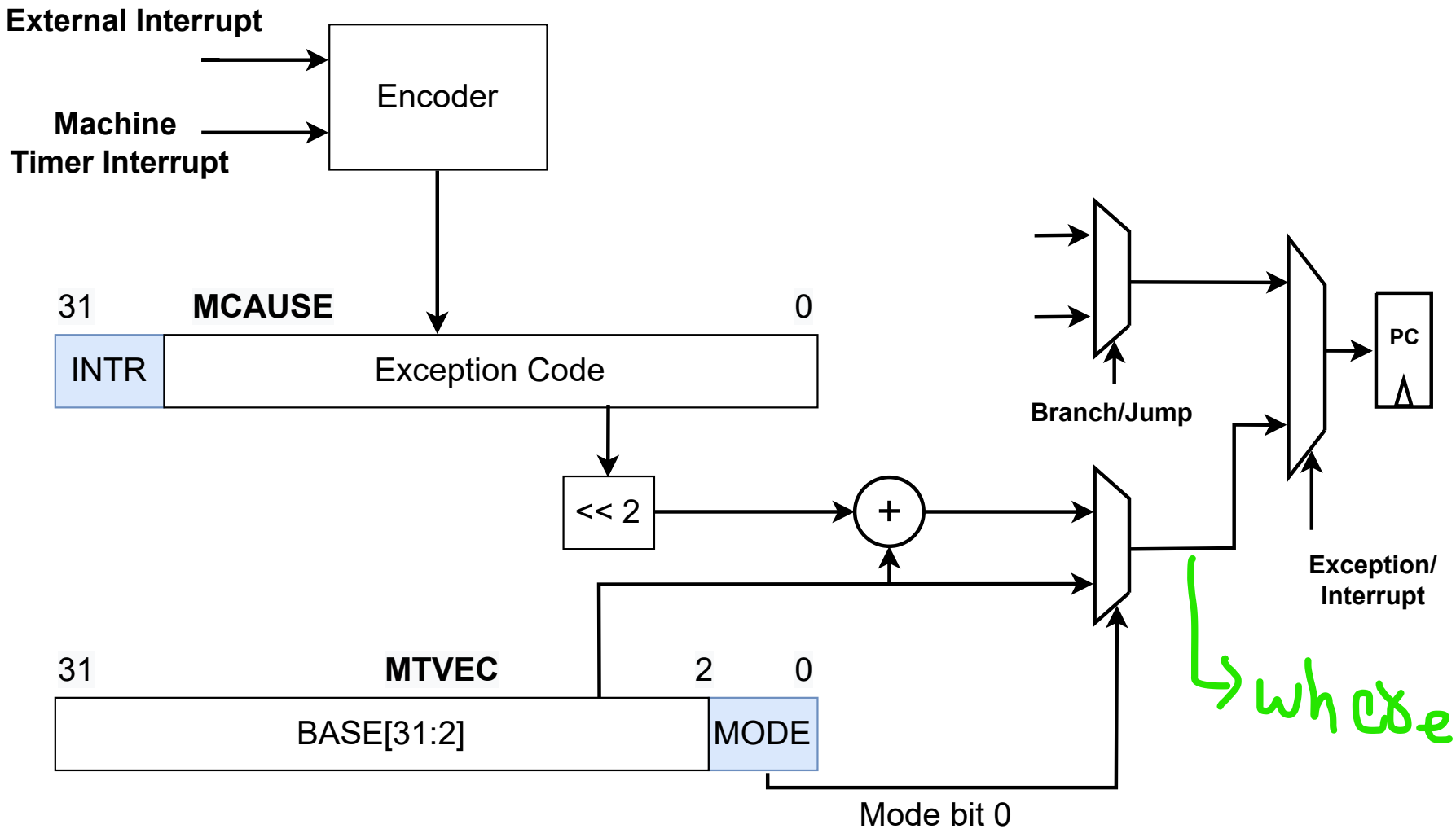
Trap Handler

- 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 MRET/SRET to resume normal execution
  - Enables the interrupts
  - Restores the processor status

Trap Handling: Configuration



Trap Handling: Runtime



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
```

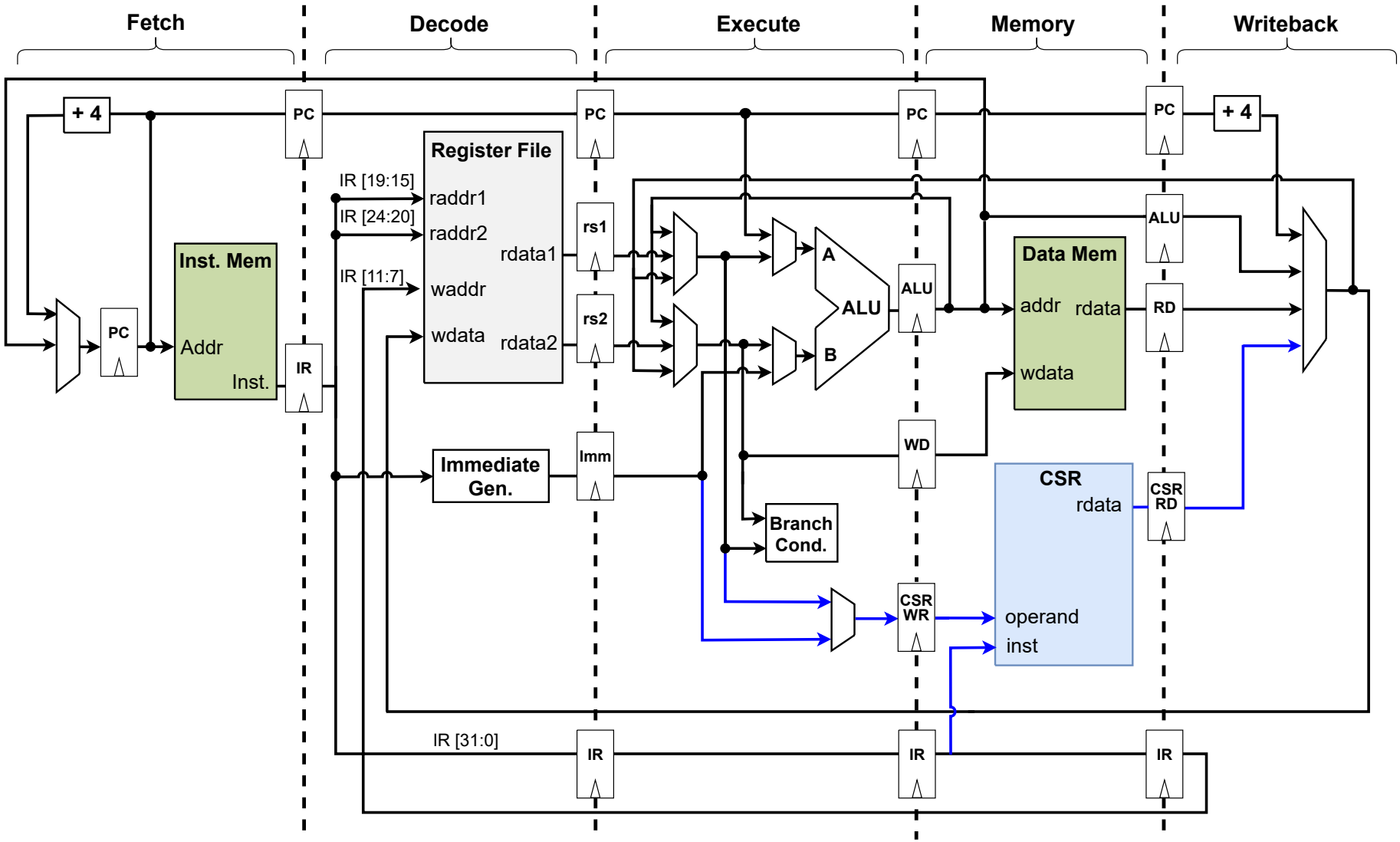
Example Startup File Cont'd

```
# 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
```

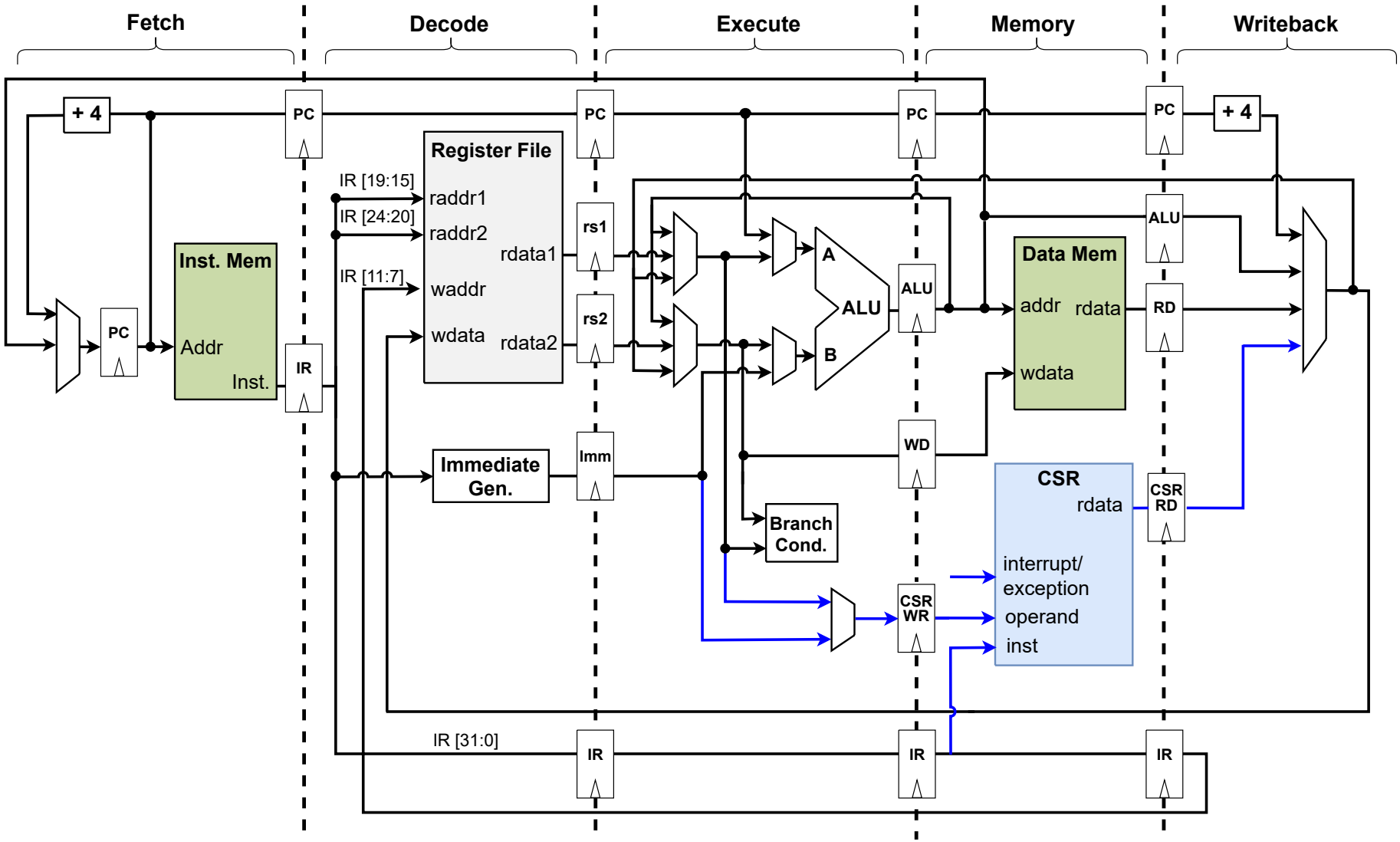
Example Interrupt Service Routine

```
# 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
```

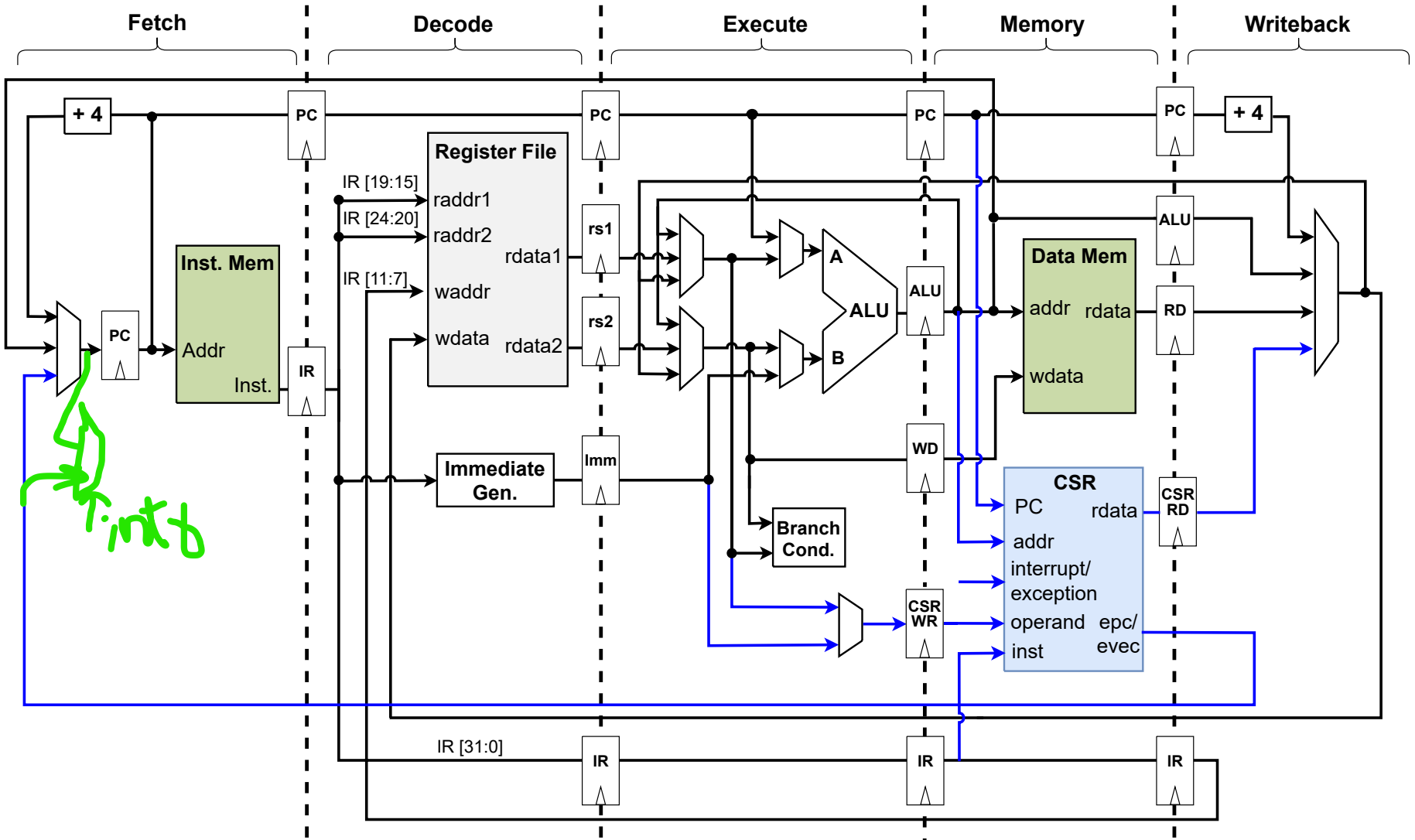
Pipelined Datapath with Exception/Interrupt Handling



Pipelined Datapath with Exception/Interrupt Handling



Pipelined Datapath with Exception/Interrupt Handling



csrrw x9, mcause, x10

Suggested Reading

- Read User Manual for the instruction set and its architecture [Waterman et al., 2016b].
- For Control and Status register description consult Privileged architecture manual [Waterman et al., 2016a].
- 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. (2016a).

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.

Waterman, A., Lee, Y., Patterson, D. A., and Asanović, K. (2016b).

The risc-v instruction set manual, volume i: User-level isa, version 2.1.

EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-129.