CASPER

CS305: Computer Architecture

Exceptions in Pipelining

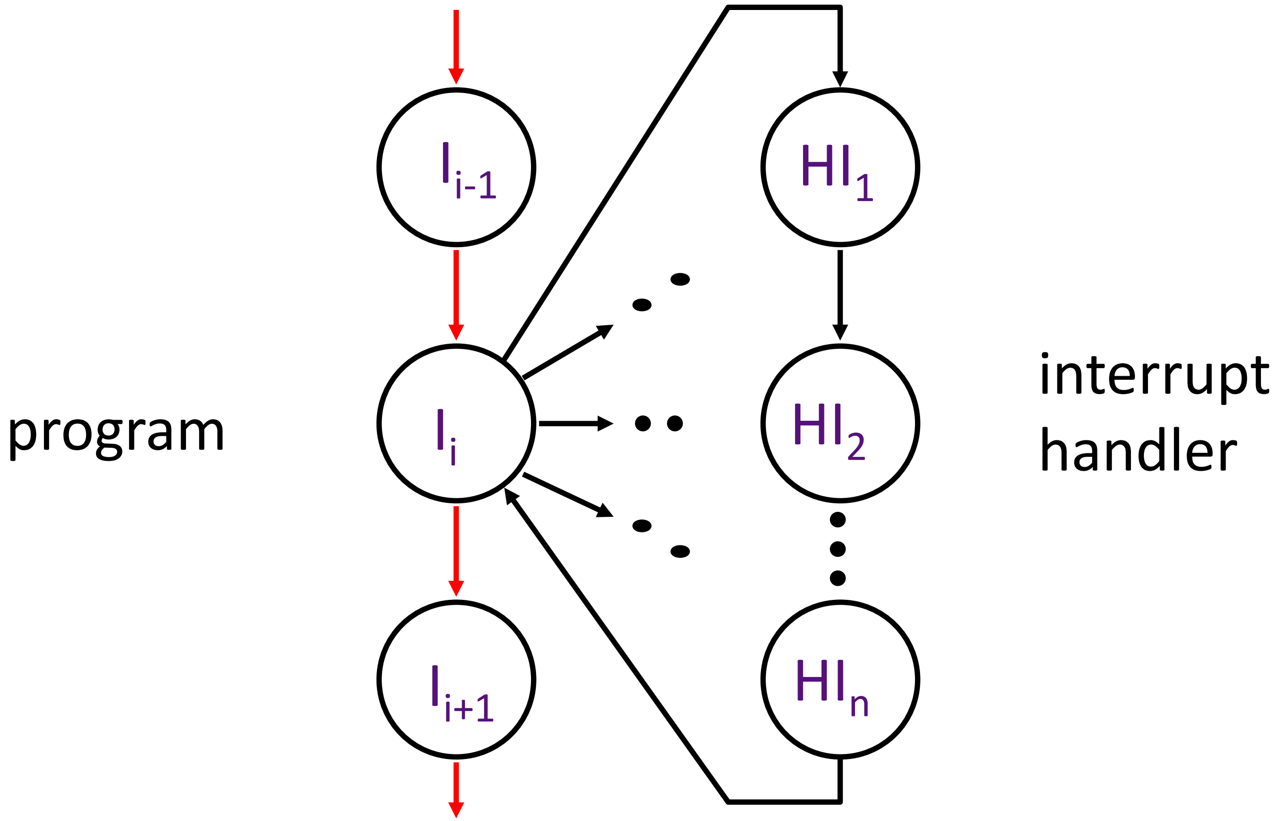
https://www.cse.iitb.ac.in/~biswa/courses/CS305/main.html

https://www.cse.iitb.ac.in/~biswa/

Exception/Interrupt

An unscheduled event that disrupts program (instructions) in action.

Interrupt Handling

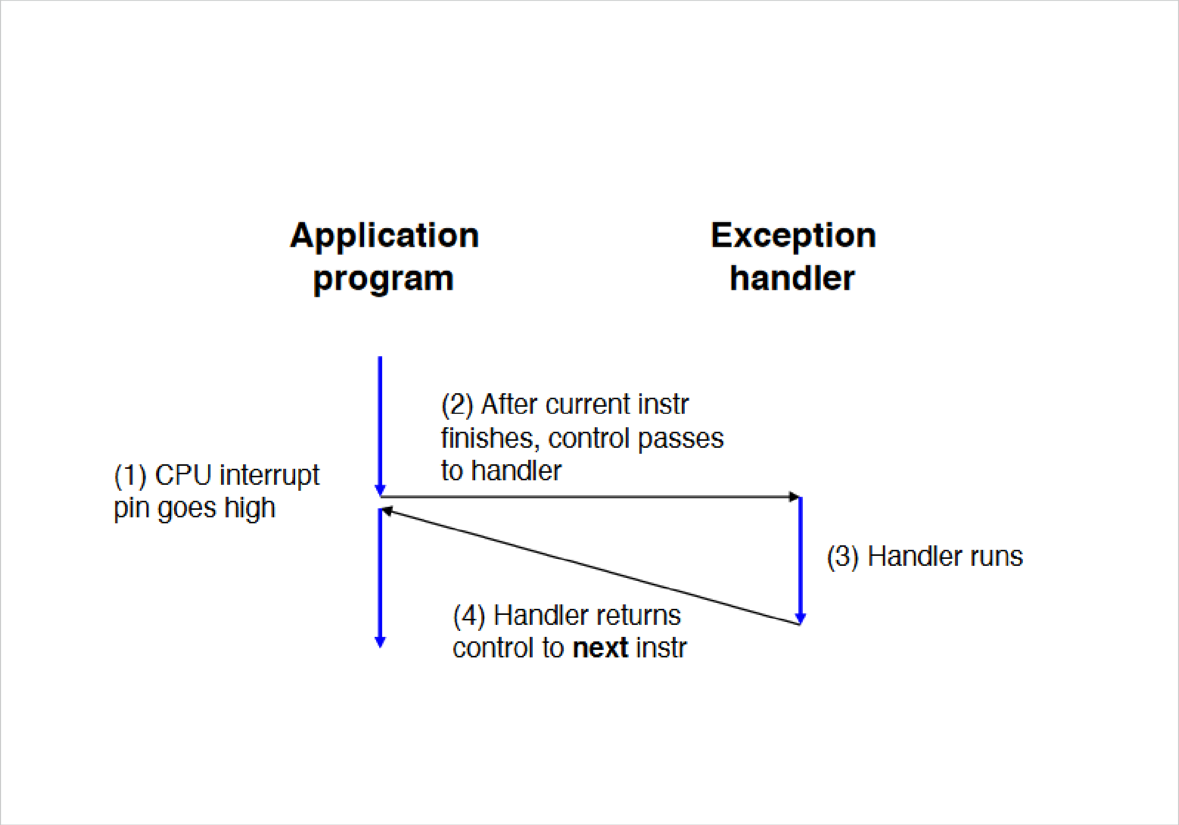


An *external or internal* *event* that needs to be processed. The event is usually unexpected or rare from program's point of view.

Causes

- Asynchronous: an *external event*
  - input/output device service-request
  - timer expiration
  - power disruptions, hardware failure
- Synchronous: an *internal event (a.k.a. traps or exceptions)*
  - undefined opcode, privileged instruction
  - arithmetic overflow, FPU exception, misaligned memory access
  - *virtual memory exceptions:* page faults,
    TLB misses, protection violations
  - system calls, e.g., jumps into kernel

Interrupt and Exception



Application program

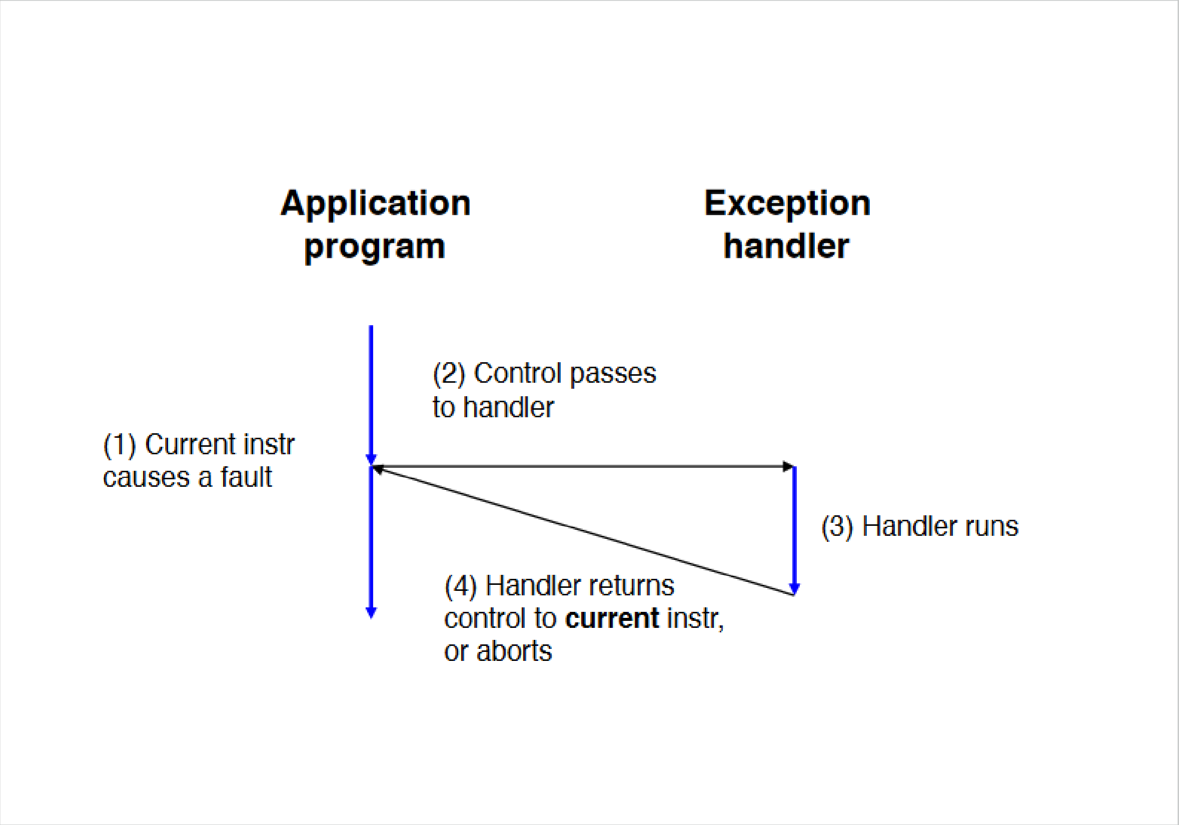
Exception handler

(1) CPU interrupt pin goes high

(2) After current instr finishes, control passes to handler

(3) Handler runs

(4) Handler returns control to **next** instr



Application program

Exception handler

(1) Current instr causes a fault

(2) Control passes to handler

(3) Handler runs

(4) Handler returns control to **current** instr, or aborts

Interrupt Handler

Exception program counter (EPC): address of the offending instruction,

Saves EPC before enabling interrupts to allow nested interrupts

Need to mask further interrupts at least until EPC can be saved

Need to read a *status register* that indicates the cause of the interrupt

Handshake between processor and the OS

Processor:

- stops the offending instruction,
- makes sure all prior instructions complete,
- flushes all the future instructions (in the pipeline)
- Sets a register to show the cause
- Saves EPC
- Disables further interrupts
- Jumps to pre-decided address (cause register or vectored)

Handshake between processor and the OS

OS:

Looks at the cause of the exception

Interrupt handler saves the GPRs

Handles the interrupt/exception

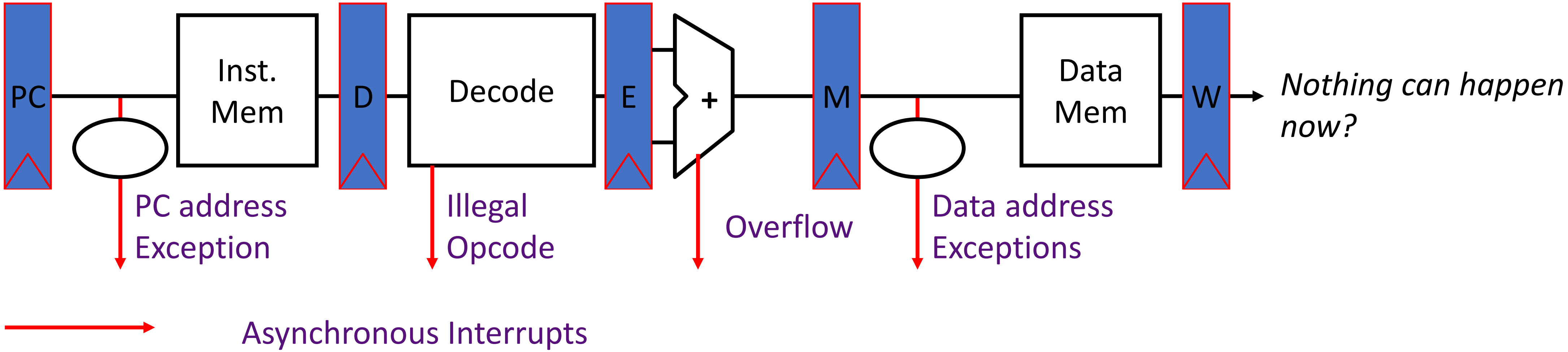
Calls RFE

Contd.

Uses a special indirect jump instruction RFE (*return-from-exception*) which

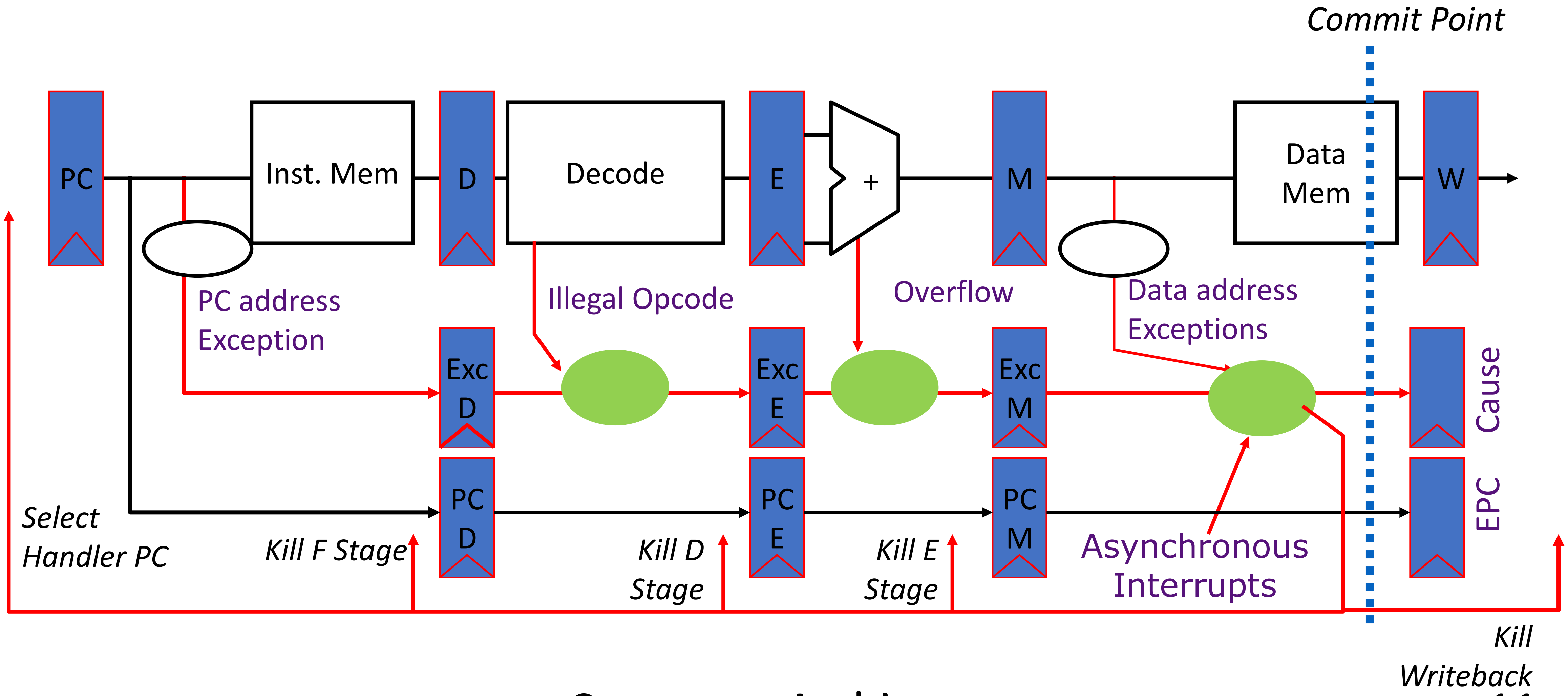
- enables interrupts
- restores the processor to the user mode

Exception handling and Pipelining



- When do we stop the pipeline for *precise* interrupts or exceptions?
- How to handle multiple simultaneous exceptions in different pipeline stages?
- How and where to handle external asynchronous interrupts?

Contd.



Commit Point
PC
Inst. Mem
D
Decode
E
+
M
Data Mem
W
PC address Exception
Illegal Opcode
Overflow
Data address Exceptions
Exc D
Exc E
Exc M
Cause
PC D
PC E
PC M
EPC
Select Handler PC
Kill F Stage
Kill D Stage
Kill E Stage
Asynchronous Interrupts
Kill Writeback

Contd.

- Hold exception flags in pipeline until commit point for instructions that will be killed

- Exceptions in earlier pipe stages override later exceptions *for a given instruction*

- If exception at commit: update cause and EPC registers, kill all stages, inject handler PC into fetch stage

Grazie