信息科学与技术学院
School of Information Science and Technology

CS 110
Computer Architecture
Operating System & I/O

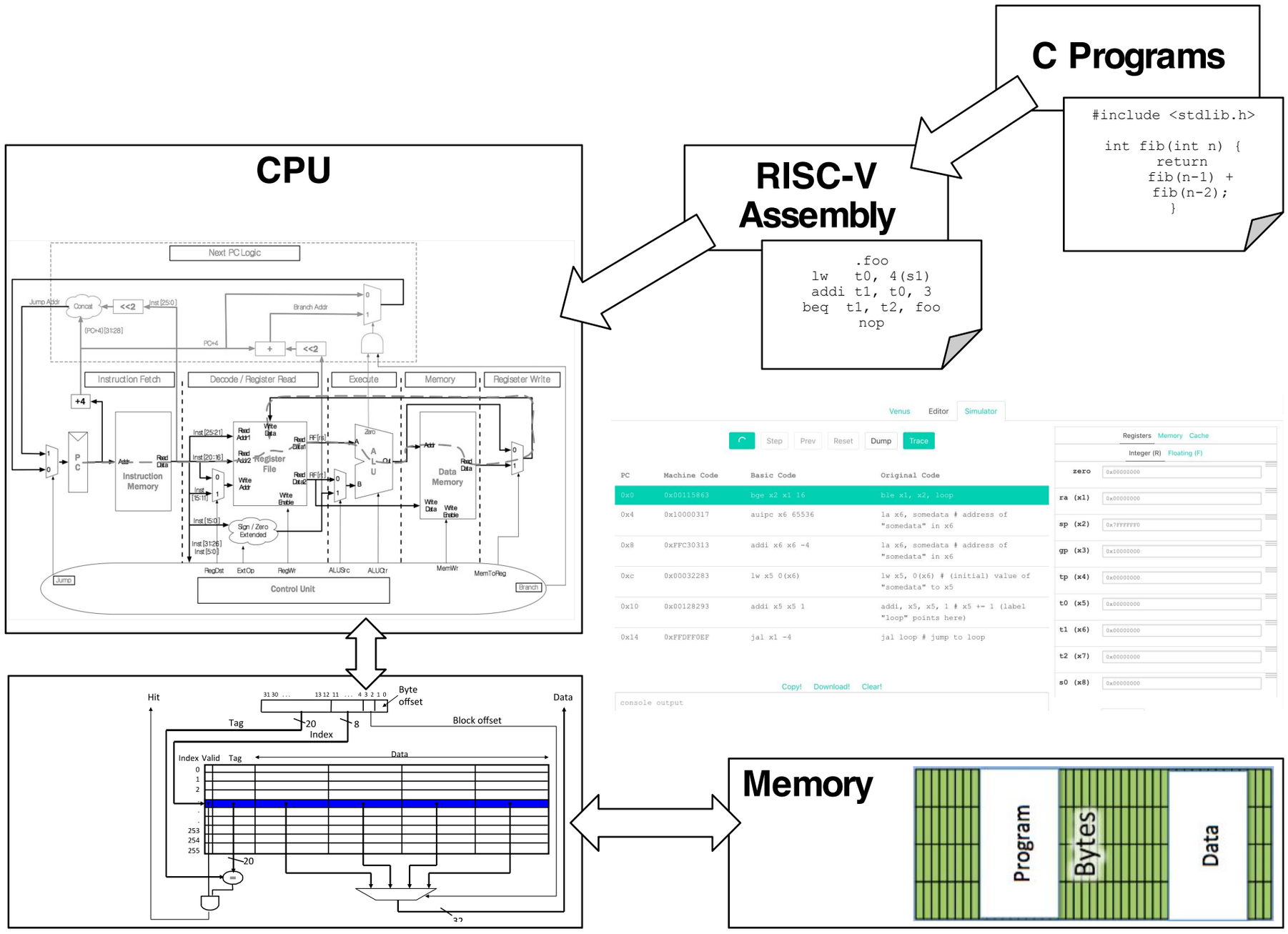
Instructors:
Chundong Wang, Siting Liu & Yuan Xiao
Course website: https://toast-lab.sist.shanghaitech.edu.cn/courses/CS110@ShanghaiTech/Spring-2025/index.html
School of Information Science and Technology (SIST)
ShanghaiTech University

2025/5/22

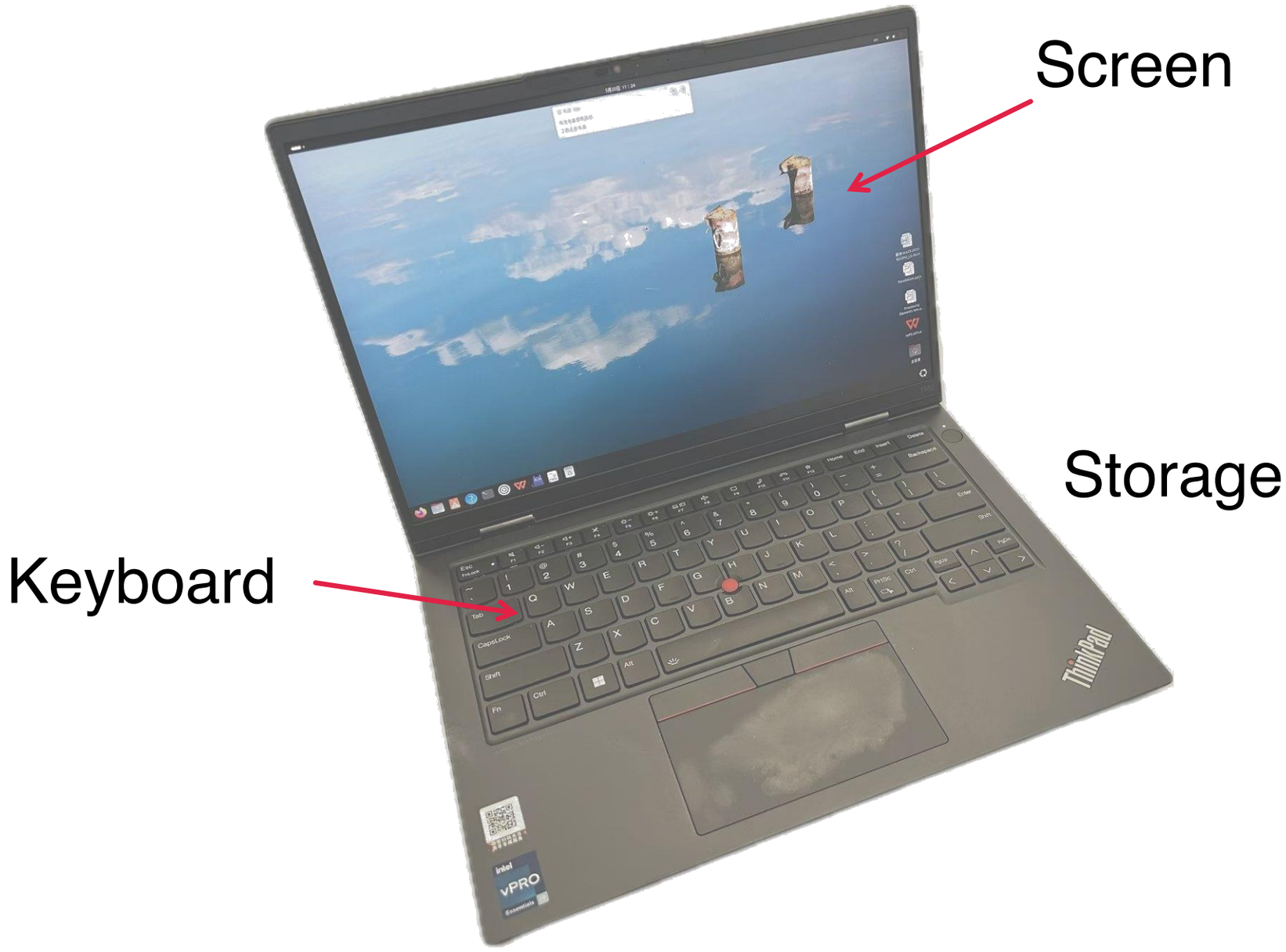
Administratives

- Final exam, June 12th 8am-10am; you can bring **3**-page A4-sized double-sided cheat sheet, **handwritten** only! (Teaching center 201/202/203); the whole course will be covered. No electronic devices (no smart watches, no calculators, etc.)
- Project 3 released. Speed Competition! ddl May 29th.
- Project 4 released, ddl June 3rd. **Will be check the 17th week.**
- HW 7 released, ddl approaching, May 30th.
- **HW 8 will be released soon, ddl June 5th.**
- To check Lab 13 this week, May 20th, 22nd & 26th
- Lab 14 released, to check May 27th, 29nd & June 4th (Lab Session 1 only, 1D104); Prepare in advance!
- Discussion May 23th & 26th on *OS & Virtual memory*.

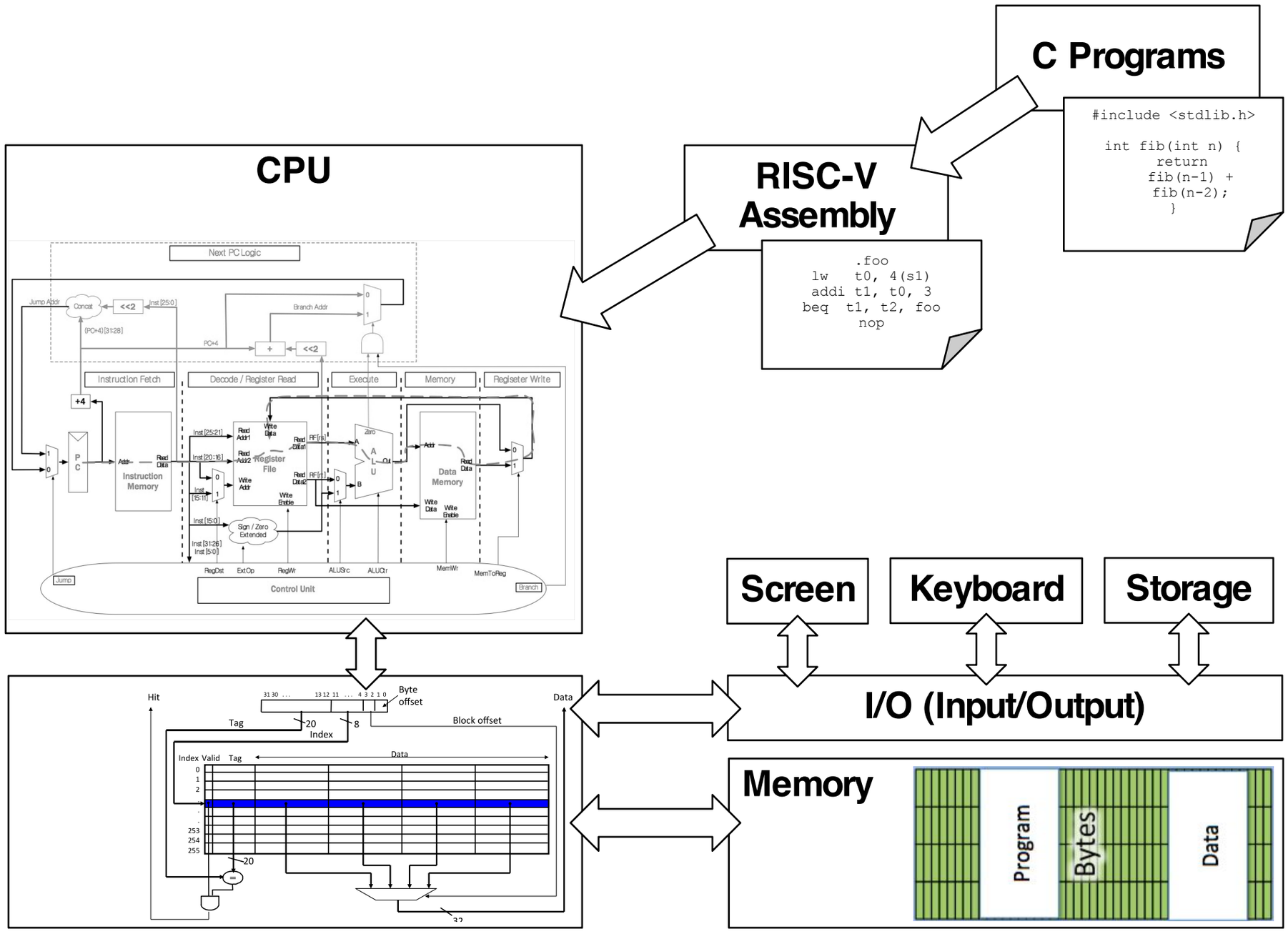
CA so far ...


C Programs
#include <stdlib.h>
int fib(int n) {
return
fib(n-1) +
fib(n-2);
}
RISC-V Assembly
.foo
lw t0, 4(s1)
addi t1, t0, 3
beq t1, t2, foo
nop
CPU
Next PC Logic
Jump Addr
Concat
<<2
Inst [25:0]
(PC+4)[31:28]
Branch Addr
PC+4
+
<<2
Instruction Fetch
Decode / Register Read
Execute
Memory
Regiseter Write
+4
P C
Instruction Memory
Register File
Data Memory
A L U
Sign / Zero Extended
Control Unit
RegDst
ExtOp
RegWr
ALUSrc
ALUCtr
MemWr
MemToReg
Jump
Branch
Venus
Editor
Simulator
Step
Prev
Reset
Dump
Trace
PC
Machine Code
Basic Code
Original Code
0x0 0x00115863 bge x2 x1 16 ble x1, x2, loop
0x4 0x10000317 auipc x6 65536 la x6, somedata # address of "somedata" in x6
0x8 0xFFC30313 addi x6 x6 -4 la x6, somedata # address of "somedata" in x6
0xc 0x00032283 lw x5 0(x6) lw x5, 0(x6) # (initial) value of "somedata" to x5
0x10 0x00128293 addi x5 x5 1 addi, x5, x5, 1 # x5 += 1 (label "loop" points here)
0x14 0xFFDFF0EF jal x1 -4 jal loop # jump to loop
Copy!
Download!
Clear!
console output
Registers
Memory
Cache
Integer (R)
Floating (F)
zero
ra (x1)
sp (x2)
gp (x3)
tp (x4)
t0 (x5)
t1 (x6)
t2 (x7)
s0 (x8)
Hit
Data
Tag
Index
Byte offset
Block offset
Index Valid Tag
Data
Memory
Program
Bytes
Data

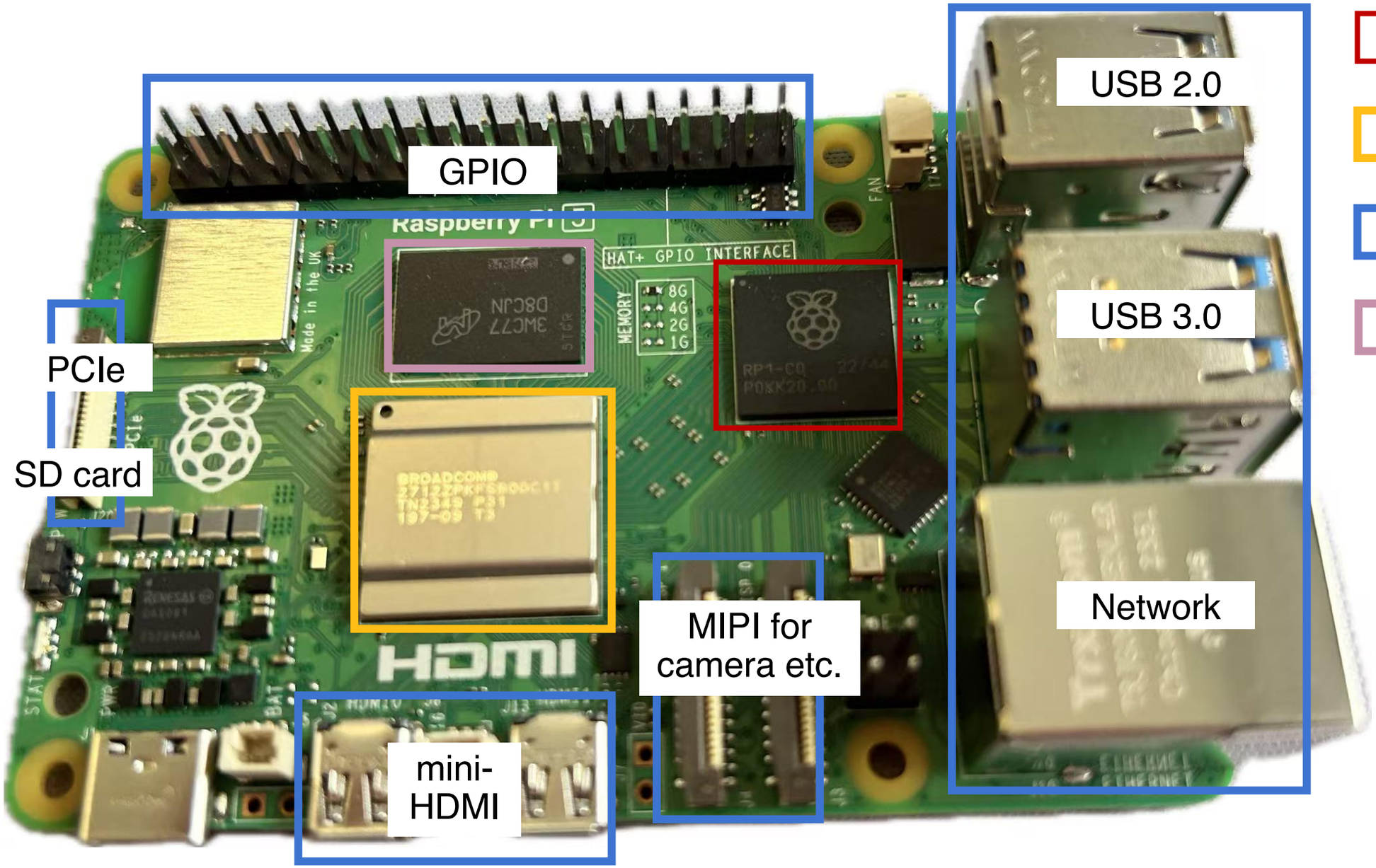
Different from Real Staff



Add I/O


C Programs
#include <stdlib.h>
int fib(int n) {
return
fib(n-1) +
fib(n-2);
}
RISC-V Assembly
.foo
lw t0, 4(s1)
addi t1, t0, 3
beq t1, t2, foo
nop
CPU
Next PC Logic
Instruction Fetch
Decode / Register Read
Execute
Memory
Regiseter Write
Instruction Memory
Register File
Data Memory
Control Unit
Screen
Keyboard
Storage
I/O (Input/Output)
Memory
Program
Bytes
Data
Hit
Data
Tag
Index
Byte offset
Block offset

Respberry Pi



It is a real computer!

Just Software?

- The biggest piece of software on your machine?
- How many lines of code? These are guesstimates (in millions of lines of code)

Linux Kernel Over Time

- OS ≈ kernel
  - User interface and other components not covered

Year	Kernel Version	Size of zipped file
1994	linux-1.0.tar.gz	1MB
1996	linux-2.0.tar.gz	6MB
2001	linux-2.4.0.tar.gz	23MB
2003	linux-2.6.0.tar.gz	40MB
2011	linux-3.0.tar.gz	92MB
2015	linux-4.0.tar.gz	118MB
2019	linux-5.0.tar.gz	155MB
Apr 2020	linux-5.6.8.tar.gz	166MB
May 2022	linux-5.17.5.tar.gz	189MB
May 2024	Linux-6.9.tar.gz	222MB
May 2025	Linux-6.14.7.tar.gz	231MB

All 7 fictions in txt format zipped to be 2.5MB

Besides the Hardware … …

- That's still not the same. CS 110 experience isn't like the real world.
- When switching on a computer, get this

Yes, that is just the software!

The operating system (OS)

What does the OS do?

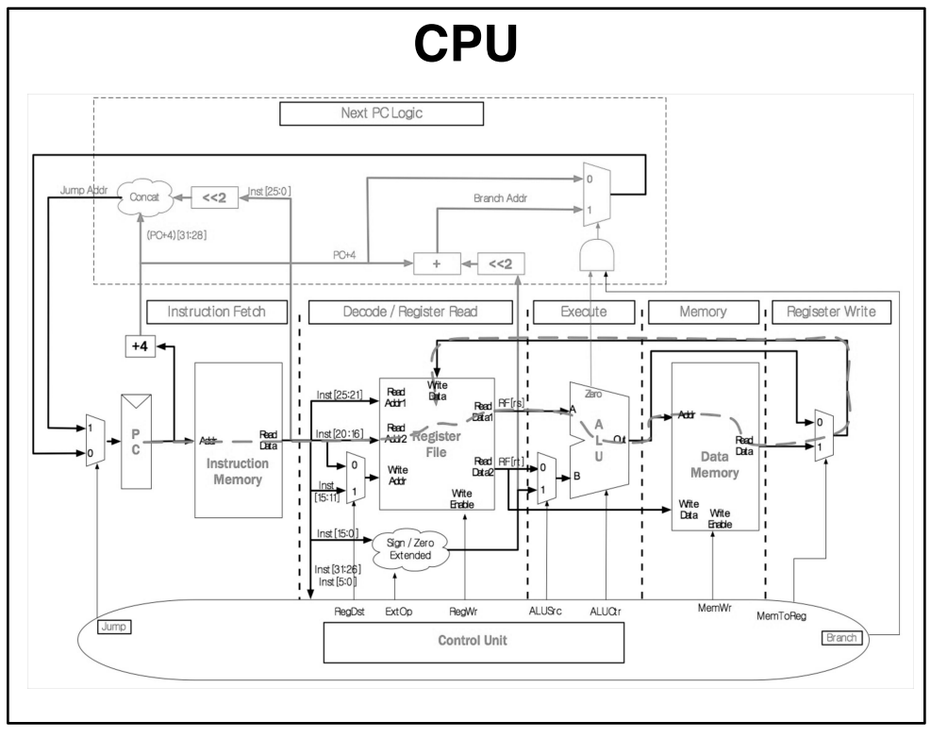
- One of the first things that runs when your computer starts (right after firmware/bootloader)
  - Provides services (100+): File System, Network stack, printer, etc.
- Loads, runs and manages programs (recall CALL):
  - Isolate programs from each other (isolation), each program/process runs (appears to run) in its own little world; (later, virtual memory)
  - Multiplex resources between applications (e.g., hardware threads memory, I/O devices: disk, keyboard, diaplay, network, etc.)
- I/O with the rest of computer
  - Provide interaction with the outside world;
  - Finds and controls all the devices in the machine in a general way (using “device drivers”)

What does the OS do?

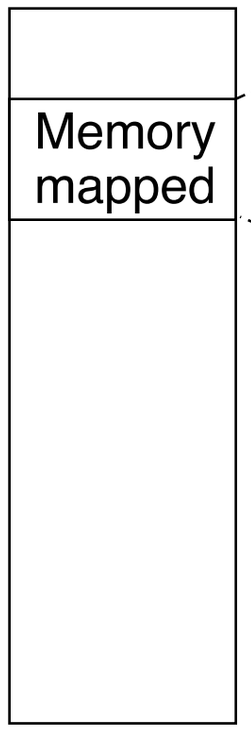
- One of the first things that runs when your computer starts (right after firmware/bootloader)
- Loads, runs and manages programs
- I/O with the rest of computer

What happens at Boot?

- When the computer switches on, the CPU executes instructions from some start address (stored in Flash ROM)



PC = 0x2000 (some default value)



Address Space

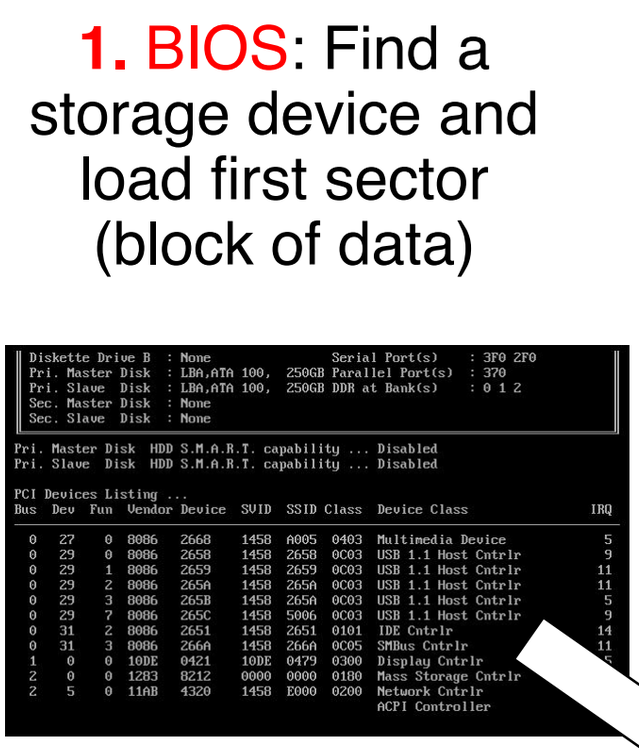

```
0x2000:
addi t0, zero, 0x1000
lw t0, 4(t0)
…
(Code to copy firmware
into regular memory
and jump into it)
```

- Bootstrapping: https://en.wikipedia.org/wiki/Bootstrapping

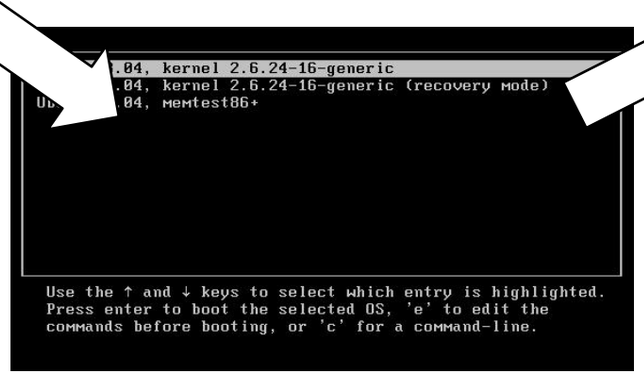
What happens at Boot?

- When the computer switches on, the CPU executes instructions from some start address (stored in Flash ROM)

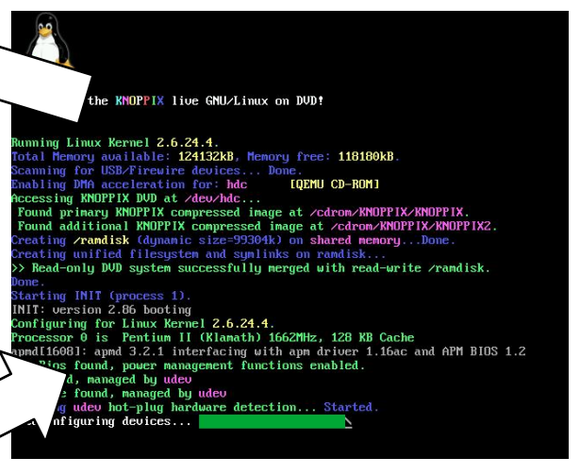
1. BIOS: Find a storage device and load first sector (block of data)



2. Bootloader (stored on, e.g., disk): Load the OS *kernel* from disk into a location in memory and jump into it.



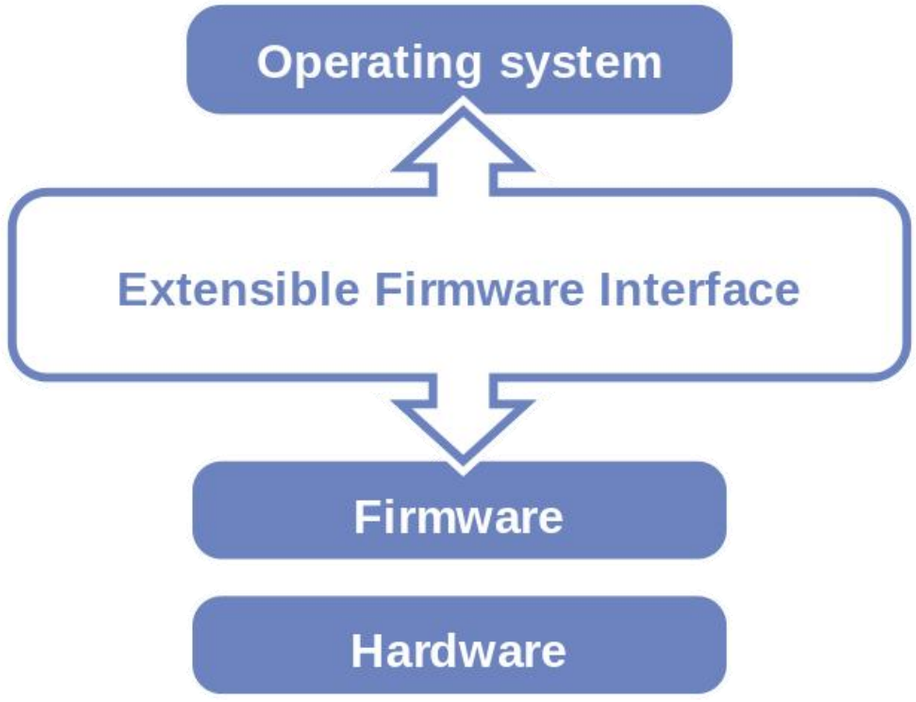
4. Init: Launch an application that waits for input in loop (e.g., Terminal/Desktop/...



3. OS Boot: Initialize services, drivers, etc.

UEFI: Unified Extensible Firmware Interface

- Successor of BIOS
- Much more powerful and complex
- E.g.graphics menu; networking; browsers
- All modern Intel & AMD-based computer use UEFI

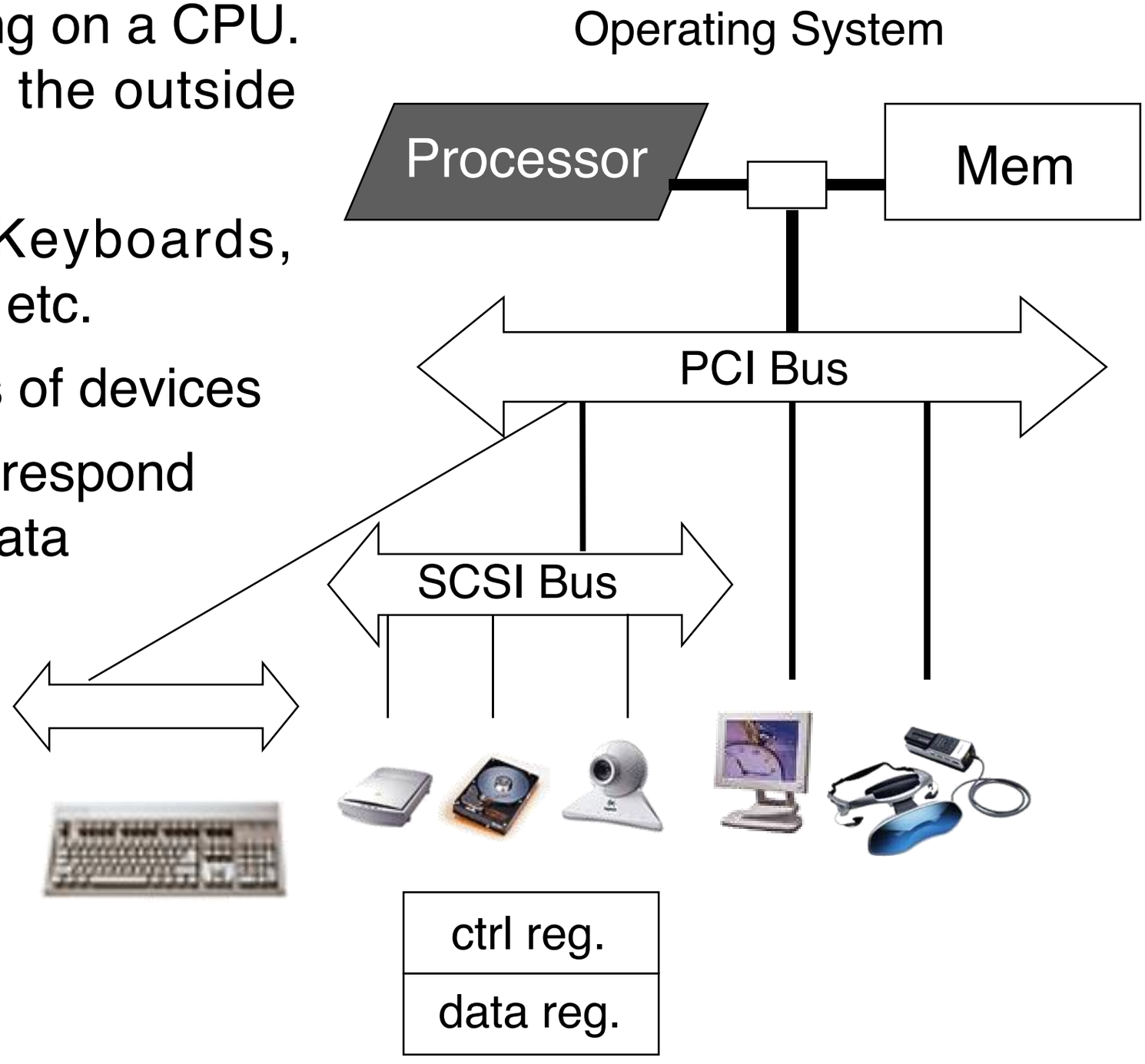


What does the OS do?

- One of the first things that runs when your computer starts (right after firmware/bootloader)
- Loads, runs and manages programs
- I/O with the rest of computer

How to interact with devices?

- Assume a program running on a CPU. How does it interact with the outside world?
- Need I/O interface for Keyboards, Network, Mouse, Screen, etc.
  - Connect to many types of devices
  - Control these devices, respond to them, and transfer data
  - Present them to user programs so they are useful



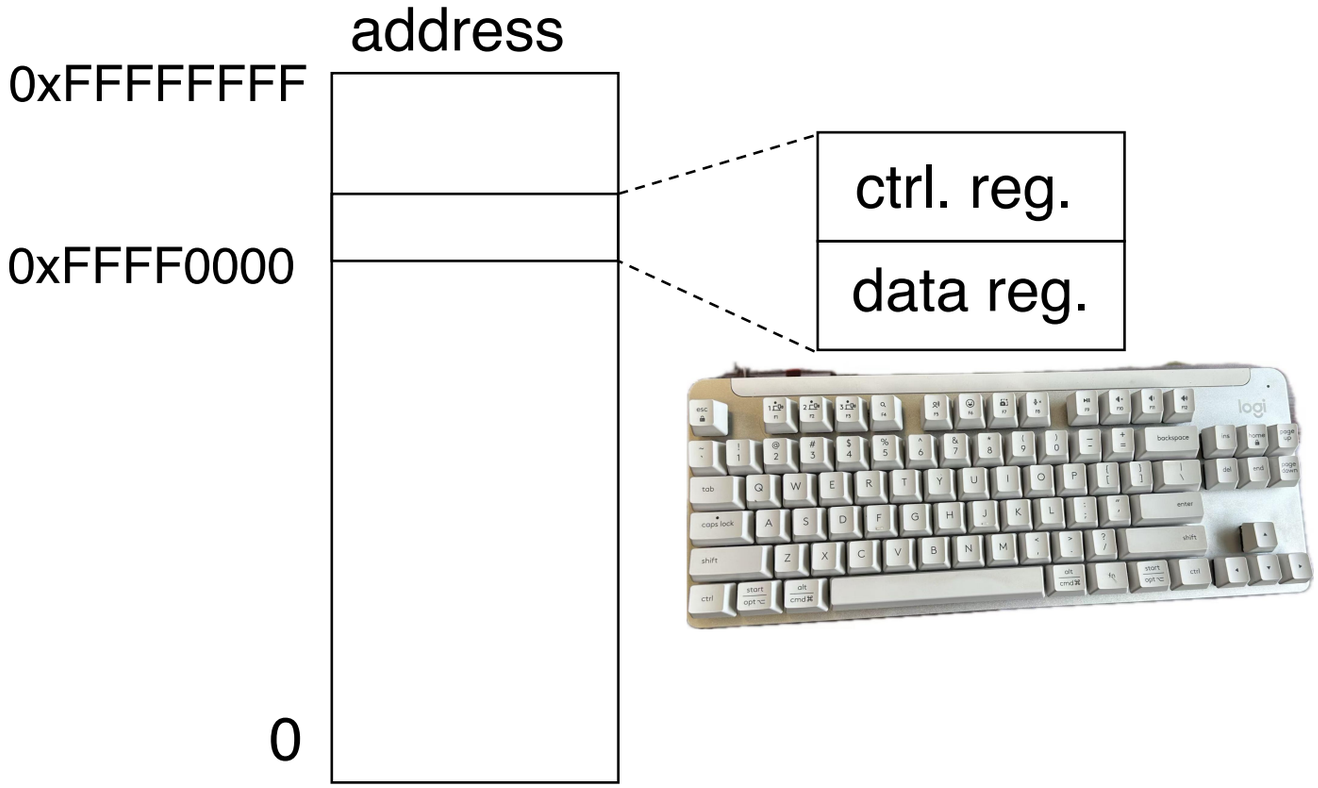
Instruction Set Architecture for I/O

- Could define a separate scheme to handle each different I/O device …
  - …or we could standardize the interface and let the I/O device handle any complications.
- What must the processor do for I/O?
  - Input: reads a sequence of bytes
  - Output: writes a sequence of bytes
- Interface options
  - Some processors have special input/output instructions
  - Memory Mapped Input/Output (MMIO, used by RISC-V):
    - Use normal load/store instructions, e.g., `lw`/`sw`, for input/output
      - In small pieces
    - A portion of the address space dedicated to I/O
    - I/O device registers there (no memory there)

Memory Mapped I/O

- Certain addresses are not regular memory, instead, they correspond to registers in I/O devices
- Shared abstraction: I/O devices read/write bytestreams.
- Shared interface is therefore memory mapped I/O

- This is a portion of address space dedicated to I/O that does not contain “regular” memory; rather, it corresponds to registers in I/O devices!
- Makes it easy to use normal load/store instructions, e.g. `lw`/`sw`
  - (Very common, used by RISC-V)



Caveat: Speed Mismatch

- In theory, simple. In practice, extremely difficult to standardize over 9 orders of magnitude of data rate!
- 1 GHz microprocessor I/O throughput: 4 GiB/s (`lw`/`sw`)
- Some I/O peak data rates:
  - 10 B/s (keyboard)
  - 3 MiB/s (Bluetooth 3.0)
  - 0.06-1.25 GiB/s (USB 2/3.1)
  - 7-250 MiB/s (WIFI, depends on standard)
  - 125 MiB/s (G-bit Ethernet)
  - 480 MiB/s (SATA3 HDD)
  - 560 MiB/s (cutting edge SSD)
  - 5 GiB/s (Thunderbolt 3)
  - 32 GiB/s (High-end DDR4 DRAM)
  - 64 GiB/s (HBM2 DRAM)
- Input may be waiting for human to act
- Output device may not be ready to accept data (as fast as processor stores it)

Processor: Polls vs. Interrupts

- Polling
  - e.g., “30 times per second”
- Processor reads from control register in loop
  - Wait for device to set ready bit in control reg. (0→1) indicates “data available” (for input device) or “ready to accept data” (for output device);
  - Then loads from/writes to data reg.
  - I/O device resets control reg. (1→0)

- Interrupts
  - Avoid wasting processor resources for low data rate devices (e.g., mouse, key-board)
- Processor runs as usual
  - Occurs when I/O is ready
  - Interrupt current program
  - Transfer control to the trap handler in the operating system

I/O Polling Example

- Input: Read from keyboard into **a0**

```
                li   t0, 0xffff0000 #ffff0000
Waitloop:       lw   t1, 0(t0)      #control
                andi t1, t1,0x1
                beq  t1, zero, Waitloop
                lw   a0, 4(t0)      #data
```

- Output: Write to display from **a0**

```
                li   t0, 0xffff0000 #ffff0000
Waitloop:       lw   t1, 8(t0)      #control
                andi t1, t1,0x1
                beq  t1, zero, Waitloop
                sw   a0, 12(t0)     #data
```

"Ready" bit is from processor's point of view!

Cost of Polling

- Assume for a processor with a 1GHz clock it takes 400 clock cycles for a polling operation (call polling routine, accessing the device, and returning). Determine % of processor time for polling
  - Mouse: polled 30 times/sec so as not to miss user movement

Cost of Polling

- Assume for a processor with a 1GHz clock it takes 400 clock cycles for a polling operation (call polling routine, accessing the device, and returning). Determine % of processor time for polling
  - Mouse: polled 30 times/sec so as not to miss user movement
- Mouse Polling [clocks/sec]

  = 30 [polls/s] * 400 [clocks/poll] = 12K [clocks/s]
- % Processor for polling:

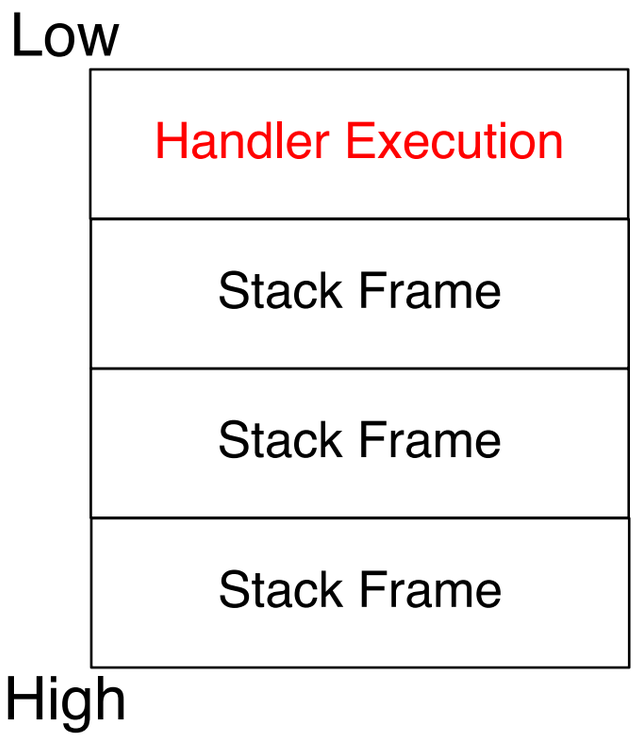
  $12*10^3$ [clocks/s] / $1*10^9$ [clocks/s] = 0.0012%

  => Polling mouse little impact on processor

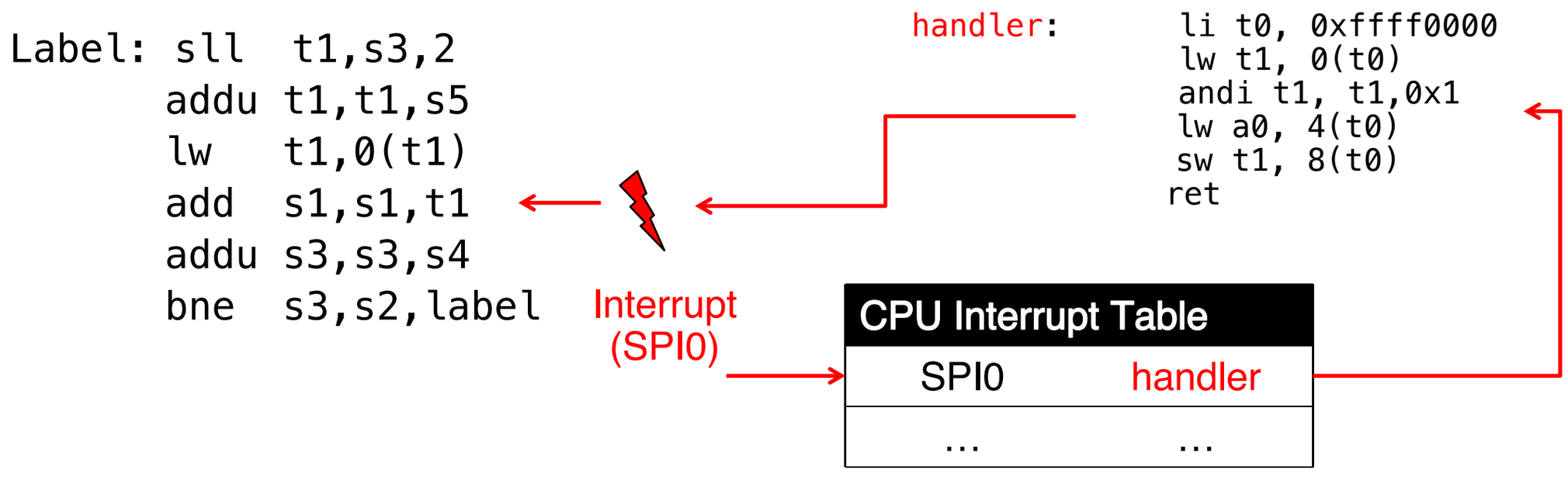
Alternative to Polling: Interrupts

- Wasteful to have processor spend most of its time “spin-waiting” for I/O to be ready
- Would like an unplanned procedure call that would be invoked only when I/O device is ready
- Solution: use interrupts to help I/O.
  - Interrupt program when I/O ready, return when done with data transfer
- Allow to register (post) interrupt handlers: functions that are called when an interrupt is triggered

Interrupt-driven I/O



1. Incoming interrupt suspends instruction stream
2. Looks up the vector (function address) of a handler in an interrupt vector table stored within the CPU
3. Perform a `jal` to the handler (needs to store any state)
4. Handler run on current stack and returns on finish (thread doesn't notice that a handler was run)



Terminologies

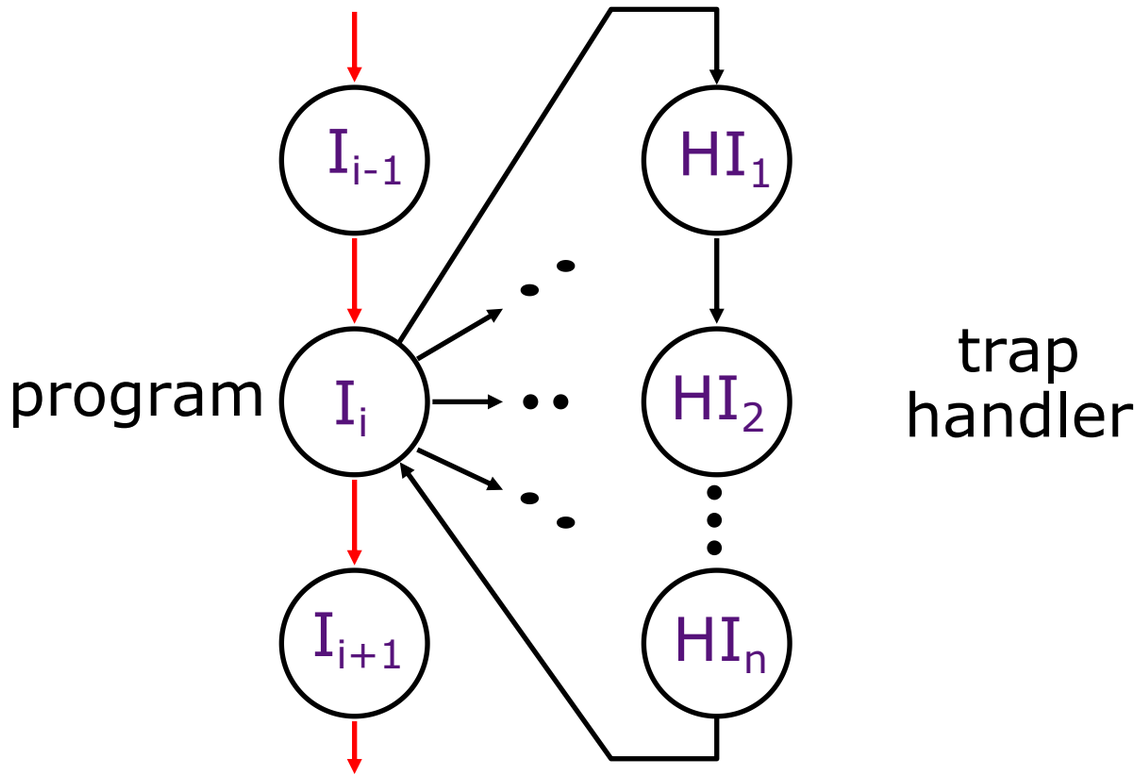
In CA (you'll see other definitions in use elsewhere):

- Interrupt – caused by an event *external* to current running program (e.g. key press, mouse activity)
  - Asynchronous to current program, can handle interrupt on any convenient instruction
- Exception – caused by some event during execution of one instruction of current running program (e.g., page fault, bus error, illegal instruction)
  - Synchronous, must handle exception on instruction that causes exception
- Trap – action of servicing interrupt or exception by hardware jump to "trap handler" code

Traps, Interrupts & Exceptions

- Altering the normal flow of control

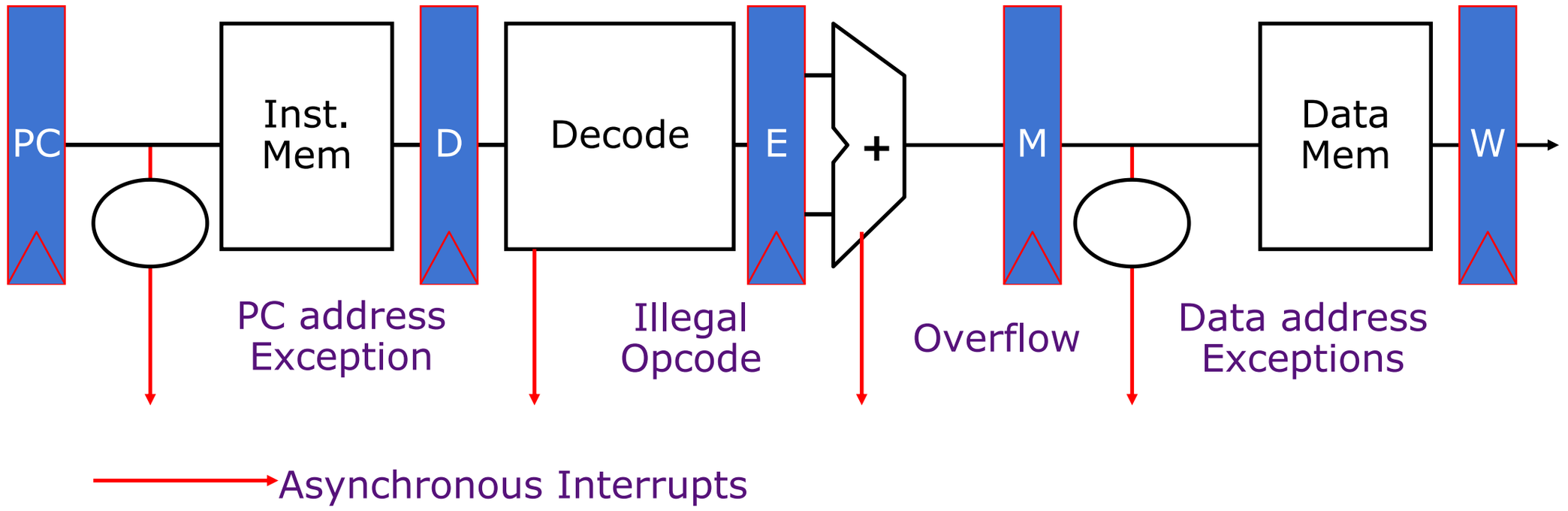
An external or internal event that needs to be processed-by another program-the OS. The event is often unexpected from original program's point of view.



Precise Traps

- Trap handler's view of machine state is that every instruction prior to the trapped one has completed, and no instruction after the trap has executed.
- Implies that handler can return from an interrupt by restoring user registers and jumping back to interrupted instruction (Supervisor exception program counter (SEPC) register will hold the instruction address)
  - Interrupt handler software doesn't need to understand the pipeline of the machine, or what program was doing!
  - More complex to handle trap caused by an exception than interrupt
- Providing precise traps is tricky in a pipelined superscalar out-of-order processor!
  - But handling imprecise interrupts in software is even worse.

Trap Handling in 5-stage Pipeline

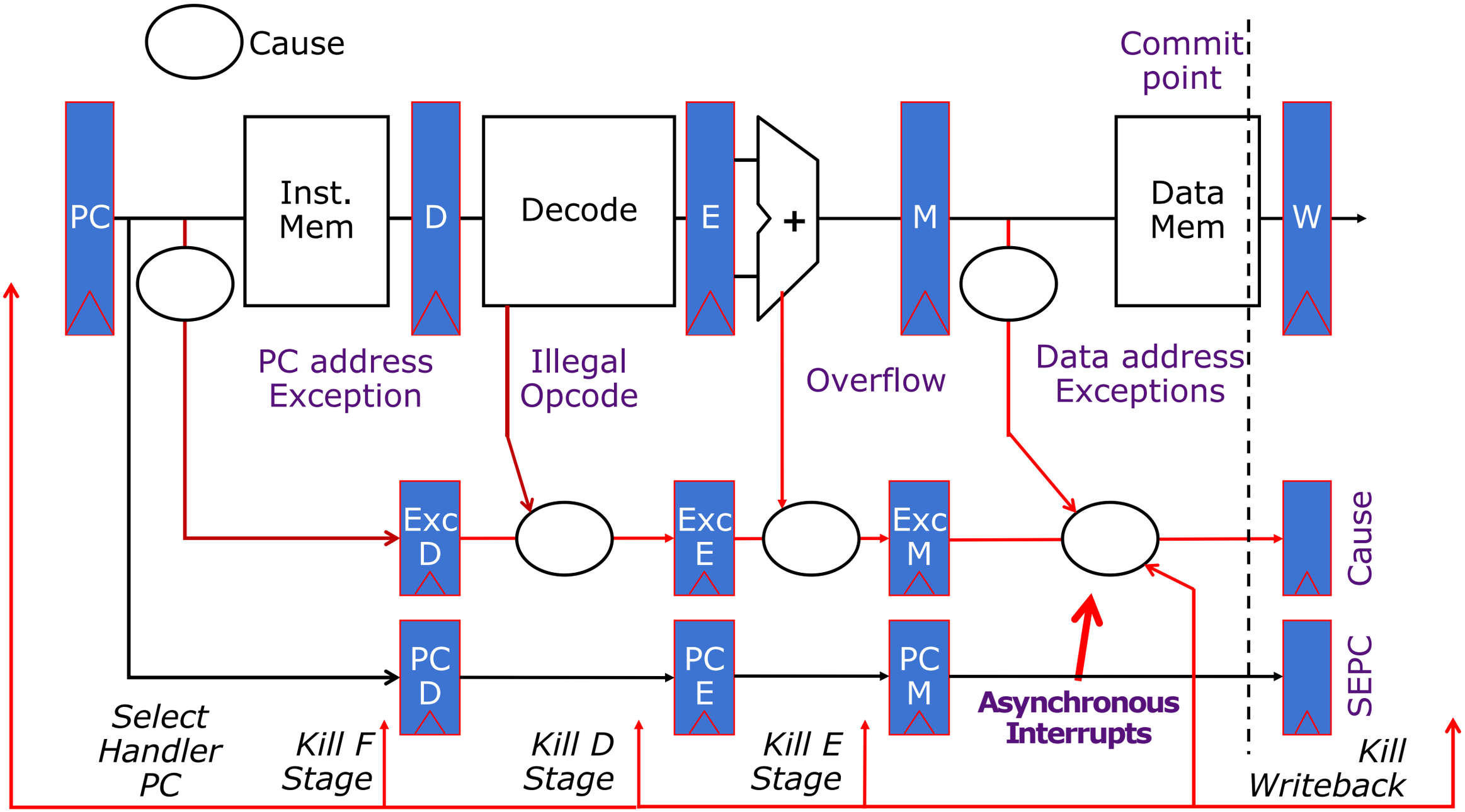


- How to handle multiple simultaneous exceptions in different pipeline stages?
- How and where to handle external asynchronous interrupts?

Handling Traps in In-Order Pipeline

- Hold exception flags in pipeline until commit point (M stage)
- Exceptions in earlier instructions override exceptions in later instructions
- Exceptions in earlier pipe stages override later exceptions *for one given instruction*
- Inject external interrupts at commit point
- If exception/interrupt at commit: update Cause and SEPC registers, kill all stages, inject handler PC into fetch stage

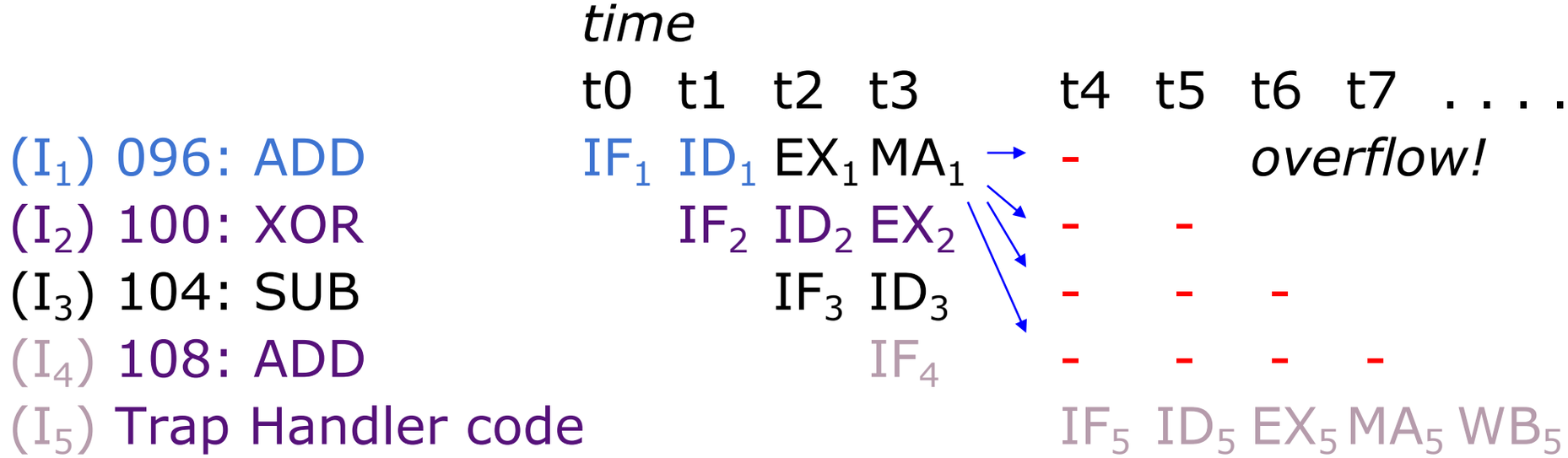
Save Exceptions until Commit



Handling Traps in In-Order Pipeline

- Hold exception flags in pipeline until commit point (M stage)
- Exceptions in earlier instructions override exceptions in later instructions
- Exceptions in earlier pipe stages override later exceptions *for a given instruction*
- Inject external interrupts at commit point
- If exception/interrupt at commit: update Cause and SEPC registers, kill all stages, inject handler PC into fetch stage

Trap Pipeline Diagram



What does the OS do?

- One of the first things that runs when your computer starts (right after firmware/bootloader)
- Loads, runs and manages programs
- I/O with the rest of computer

Launching Applications

- Applications are called “processes” in most OSes.
  - Process: separate memory;
  - Thread: shared memory
- Created by another process calling into an OS routine (using a “syscall”, more details later).
  - Depends on OS, but Linux uses fork to create a new process, and execve to load application.
- Loads executable file from disk (using the file system service) and puts instructions & data into memory (.text, .data sections), prepare stack and heap.
- Set `argc` and `argv`, jump into the main function.

Processes Management

- The OS manages multiprogramming.
  - Multiplexing (i.e., running “simultaneously”) multiple applications (processes) on one CPU.
  - This is achieved via process-level context switches, i.e., switches between processes very quickly (on the human time scale)

- The OS also manages multiprocessing.
  - Running processes simultaneously on different CPUs.

- The OS also manages processes calling other programs.
  - Current process requests this from the OS via a syscall.

System Call: Request OS Service (1/2)

- A system call (syscall) is a "software interrupt" that allows a (user) process to request a service from the operating system.
  - Similar to a function call, except now executed by the kernel.
- Example service requests:
  - Creating and deleting files; reading/writing files;
  - Accessing external devices (e.g., scanner);
  - `printf`, `malloc`, etc. (`ecalls` in RISC-V); etc.
  - Launch a new process

System Call: Request OS Service (2/2)

- A system call (syscall) is a "software interrupt" that allows a (user) process to request a service from the operating system.
  - Similar to a function call, except now executed by the kernel.
- Example service requests:
  - Creating and deleting files; reading/writing files;
  - Accessing external devices (e.g., scanner);
  - printf, malloc, etc. (ecalls in RISC-V); etc.
  - Launch a new process
- Suppose a user process (e.g., Linux shell) wants to launch a new app:
  - Shell forks: a syscall that traps into the OS kernel process
  - OS (supervisor mode): Loads program (see CALL); jumps to start of new app's main. Returns to user mode.
  - Shell: "waits" for new app's main to return (joins)

Multiprogramming

- The OS runs multiple applications at the same time.
- But not really (unless you have a core per process)
  - Time-sharing processor
- When jumping into process, set timer interrupt.
  - When it expires, store PC, registers, etc. (process state).
  - Pick a different process to run and load its state.
  - Set timer, change to user mode, jump to the new PC.
- Switches between processes very quickly. This is called a “context switch”.
- Deciding what process to run is called **scheduling**.

Protection, Translation, Paging

- Supervisor mode does not fully isolate applications from each other or from the OS.
  - Application could overwrite another application's memory.
  - Also, may want to address more memory than we actually have (e.g., for sparse data structures).
- Solution: **Virtual Memory** (Next lecture). Gives each process the illusion of a full memory address space that it has completely for itself.

Summary

- Once we have a basic machine, it’s mostly up to the OS to use it and define application interfaces.
- Main functions of an OS
  - Startup the computer (boot)
  - I/O by polling or interrupt
    - Exception, interrupt, trap
    - Precise trap
- Resources management, etc.
- More to explore in CS 130 ... ...