UC Berkeley
Teaching Professor
Dan Garcia

CS61C

Great Ideas in Computer Architecture

(a.k.a. Machine Structures)

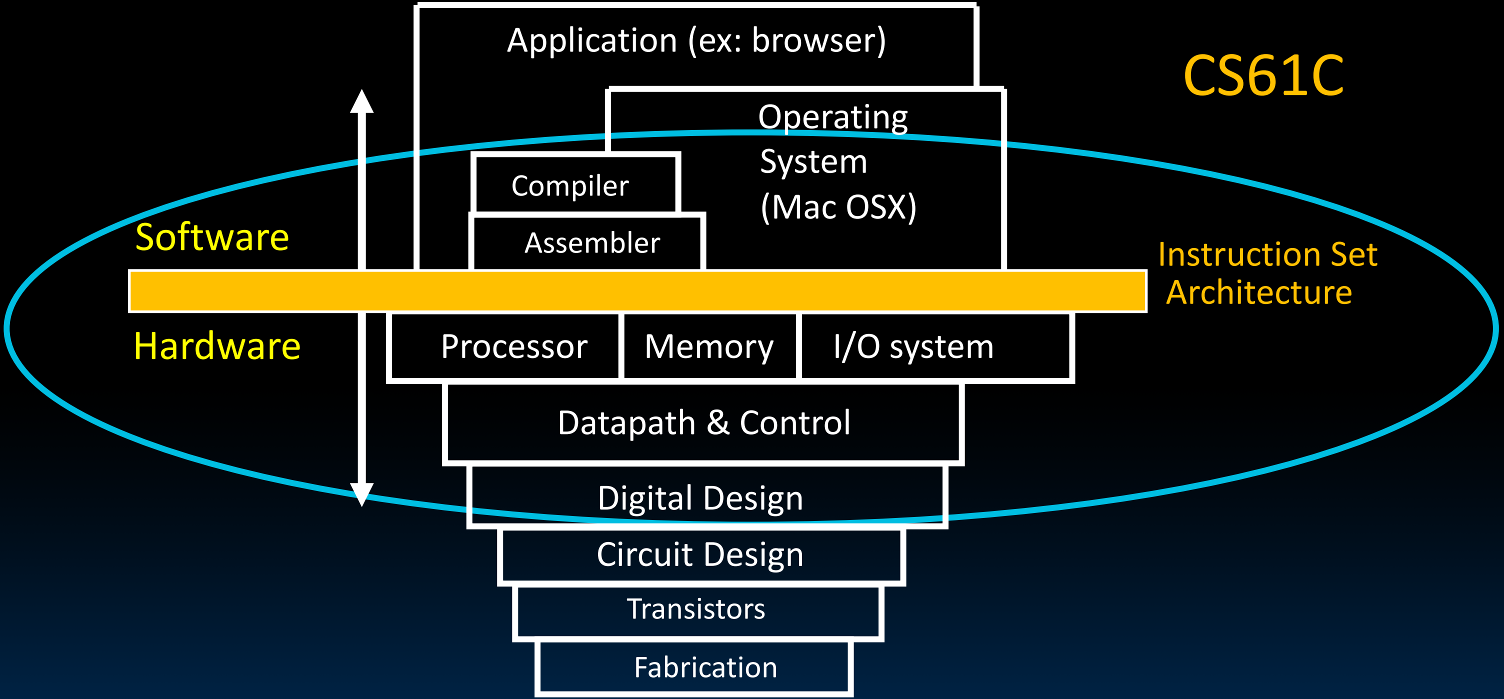
UC Berkeley
Professor
Bora Nikolić

Operating Systems and Virtual Memory

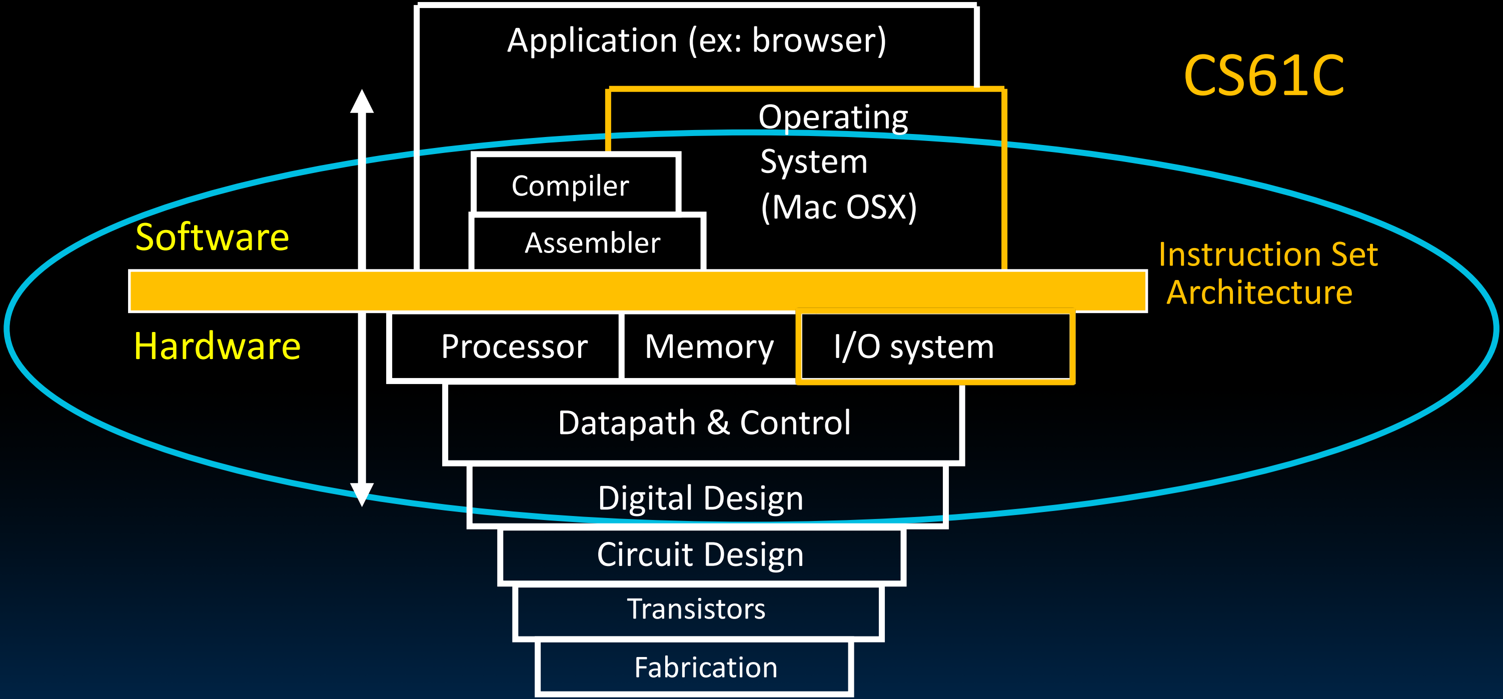
Berkeley
UNIVERSITY OF CALIFORNIA

cs61c.org

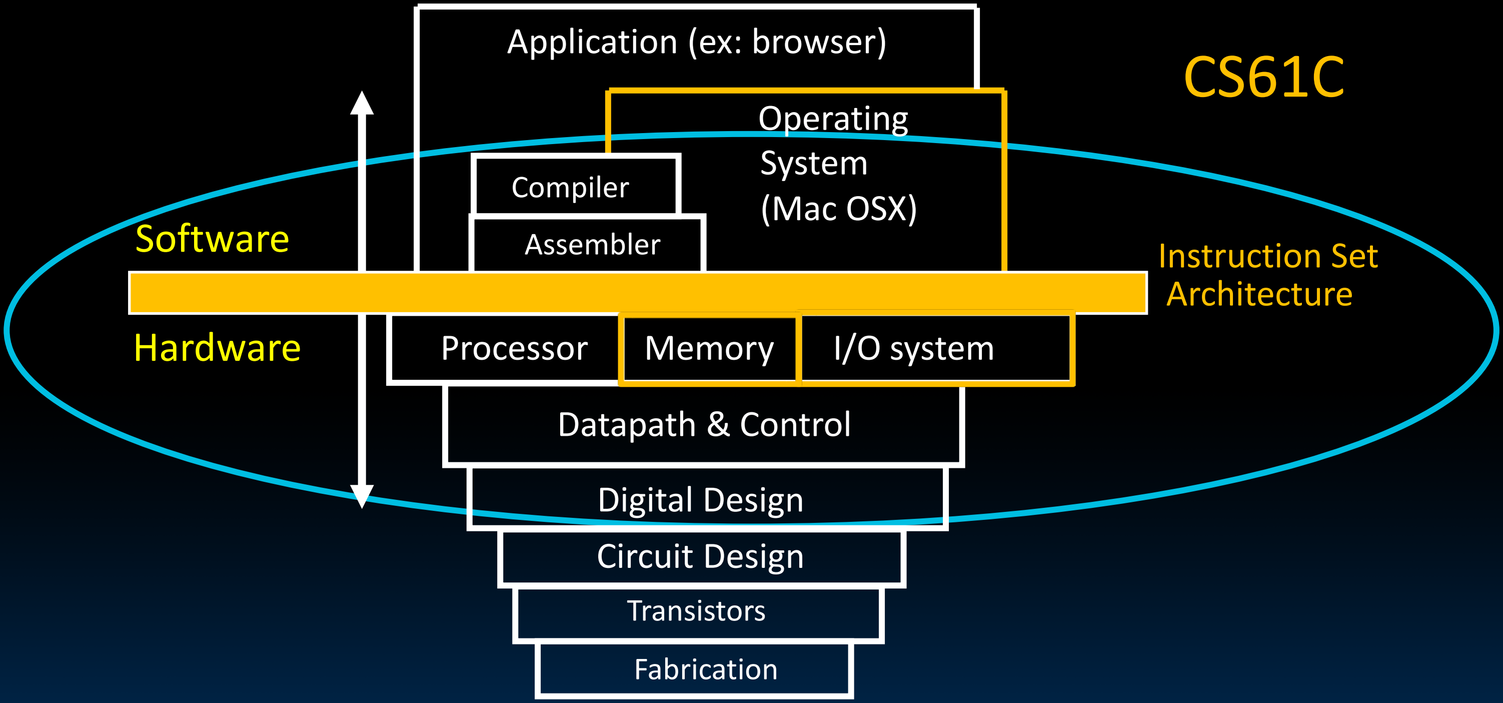
Machine Structures



Machine Structures



Machine Structures



New-School Machine Structures

Software

Parallel Requests

Assigned to computer

e.g., Search “Cats”

Parallel Threads

Assigned to core e.g., Lookup, Ads

Parallel Instructions

>1 instruction @ one time

e.g., 5 pipelined instructions

Parallel Data

>1 data item @ one time

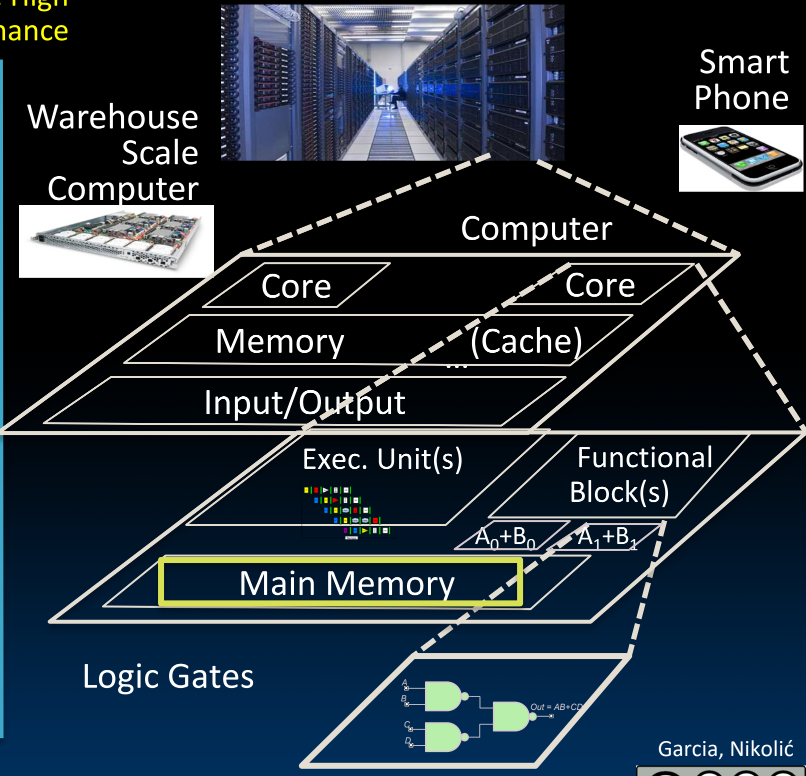
e.g., Add of 4 pairs of words

Hardware descriptions

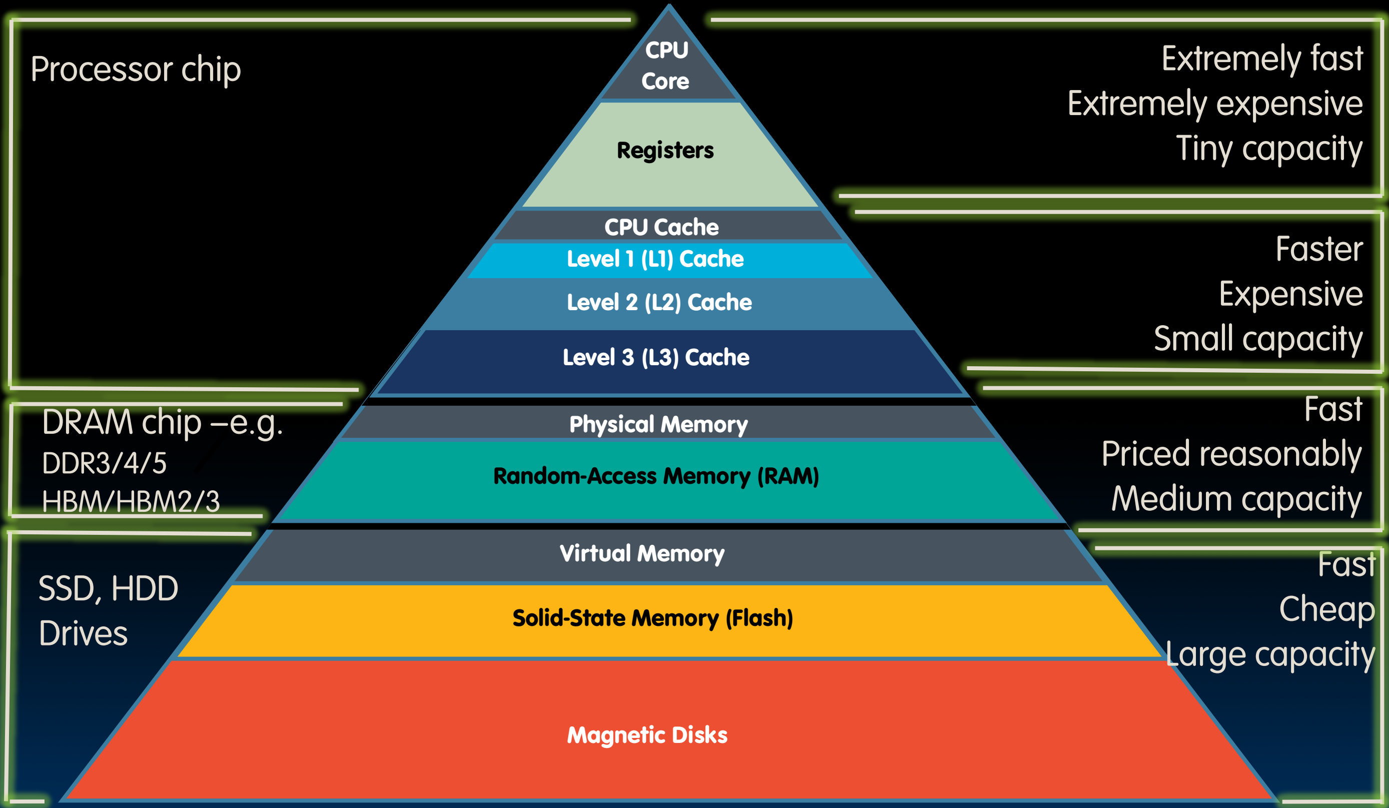
All gates work in parallel at same time

Harness Parallelism & Achieve High Performance

Hardware



Great Idea #3: Principle of Locality / Memory Hierarchy



CS61C so far…

C Programs

```
#include <stdlib.h>

int fib(int n) {
  return
    fib(n-1) +
    fib(n-2);
}
```

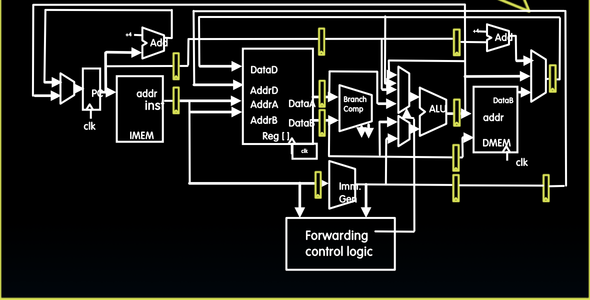
Project 1

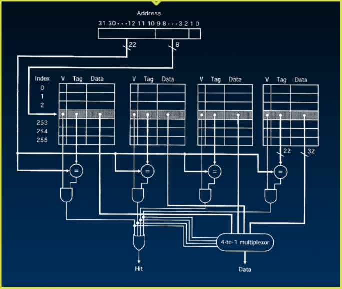
RISC-V Assembly

```
.foo
lw t0, 4(a0)
addi t1, t0, 3
beq t1, t2, foo
nop
```

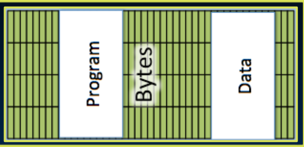
Project 2

Project 3



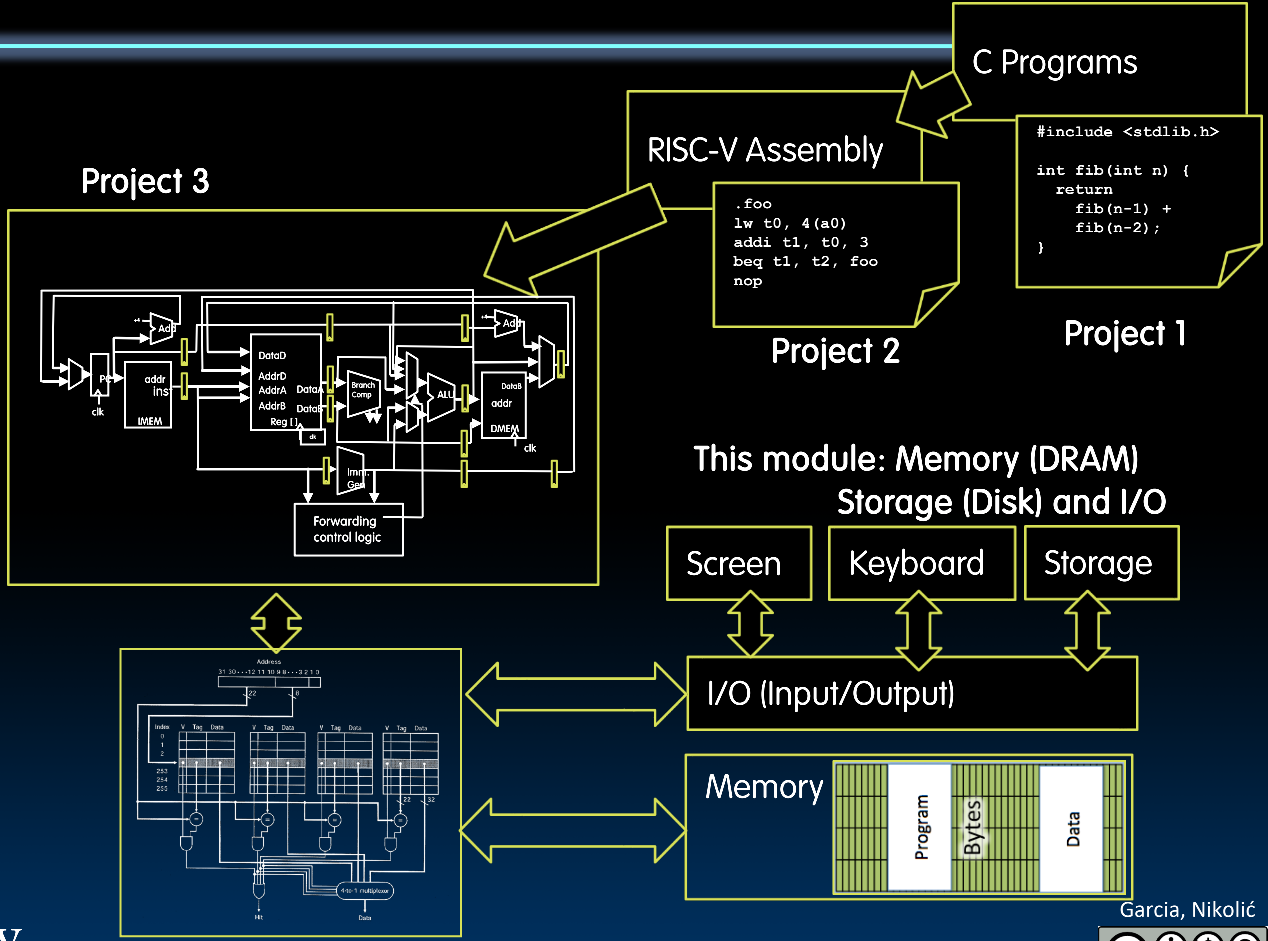


Memory



So How is a Laptop Any Different?

Adding I/O



C Programs

```
#include <stdlib.h>

int fib(int n) {
  return
    fib(n-1) +
    fib(n-2);
}
```

Project 1

RISC-V Assembly

```
.foo
lw t0, 4(a0)
addi t1, t0, 3
beq t1, t2, foo
nop
```

Project 2

Project 3

This module: Memory (DRAM) Storage (Disk) and I/O

Screen

Keyboard

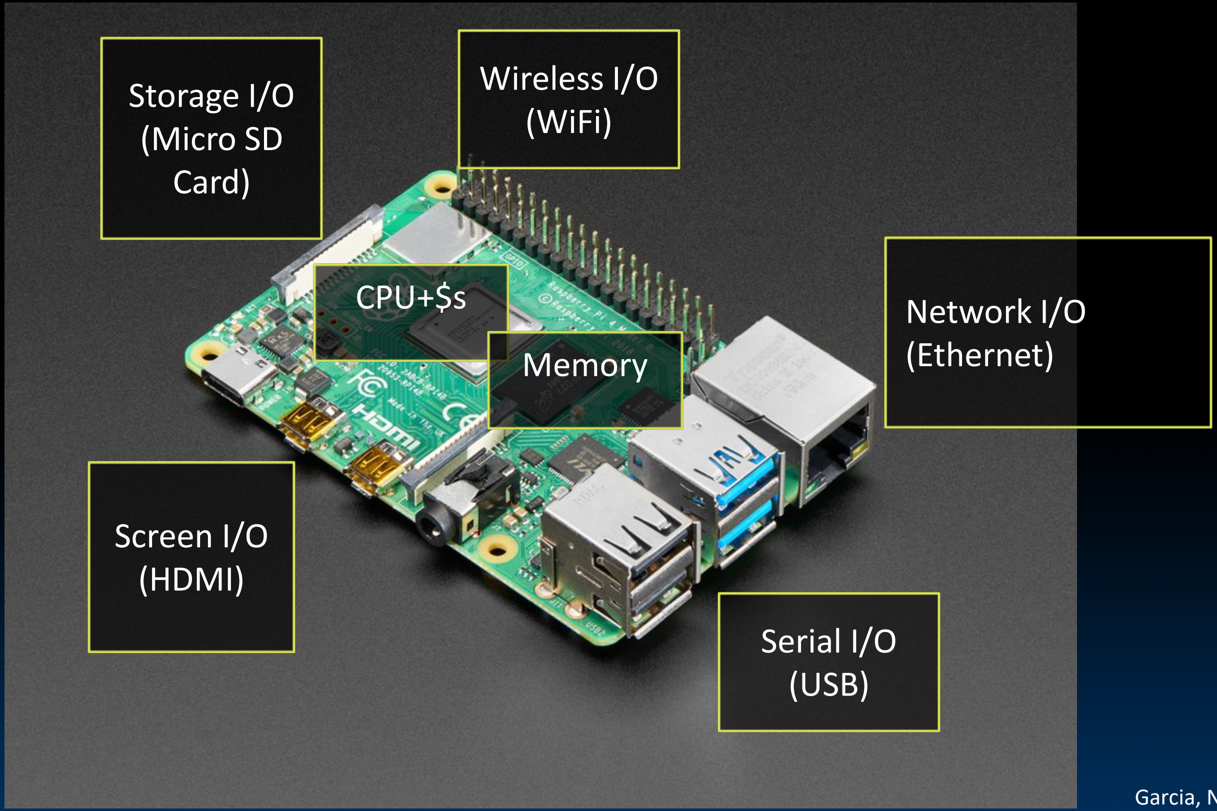
Storage

I/O (Input/Output)

Memory

Garcia, Nikolić

Raspberry Pi ($35)



It's a Real Computer!

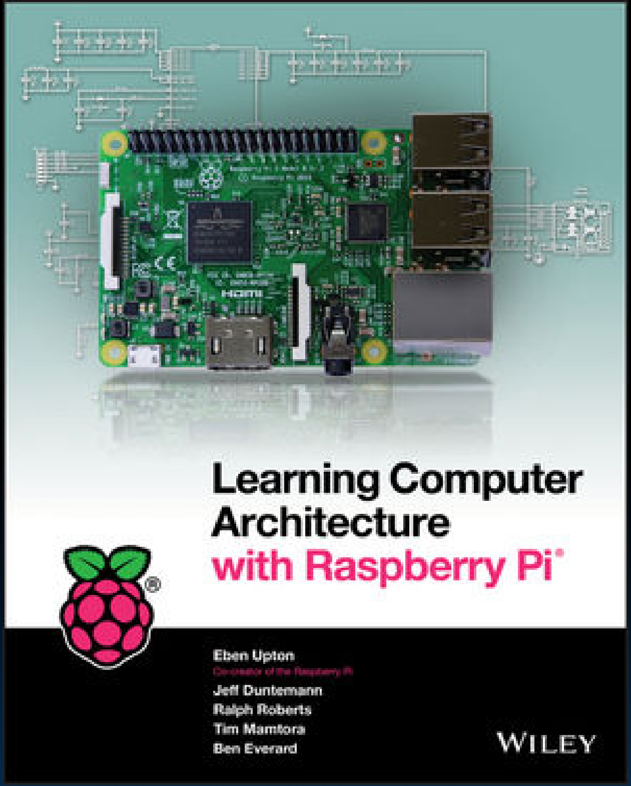
thepihut.com

CS61C with Raspberry PI?

- Lot’s of concepts from 61C covered in a book, with Raspberry Pi exercises

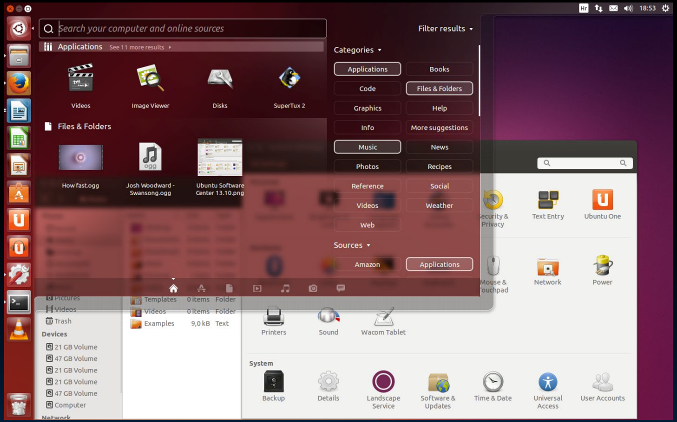
(and it is free to download if you are a Cal student:

https://onlinelibrary.wiley.com/doi/book/10.1002/9781119415534)



But Wait…

- That’s not the same! When we run VENUS, it only executes one program and then stops.
- When I switch on my computer, I get this:

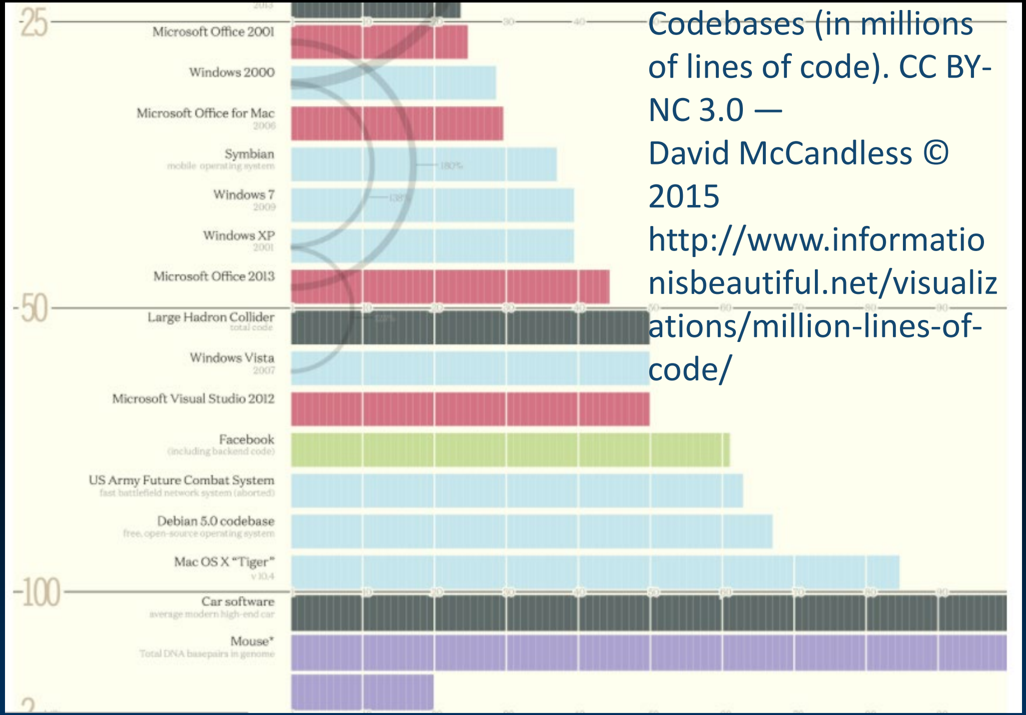


Yes, but that’s *just* software! The Operating System (OS)

Operating System Basics

Well, "Just Software"

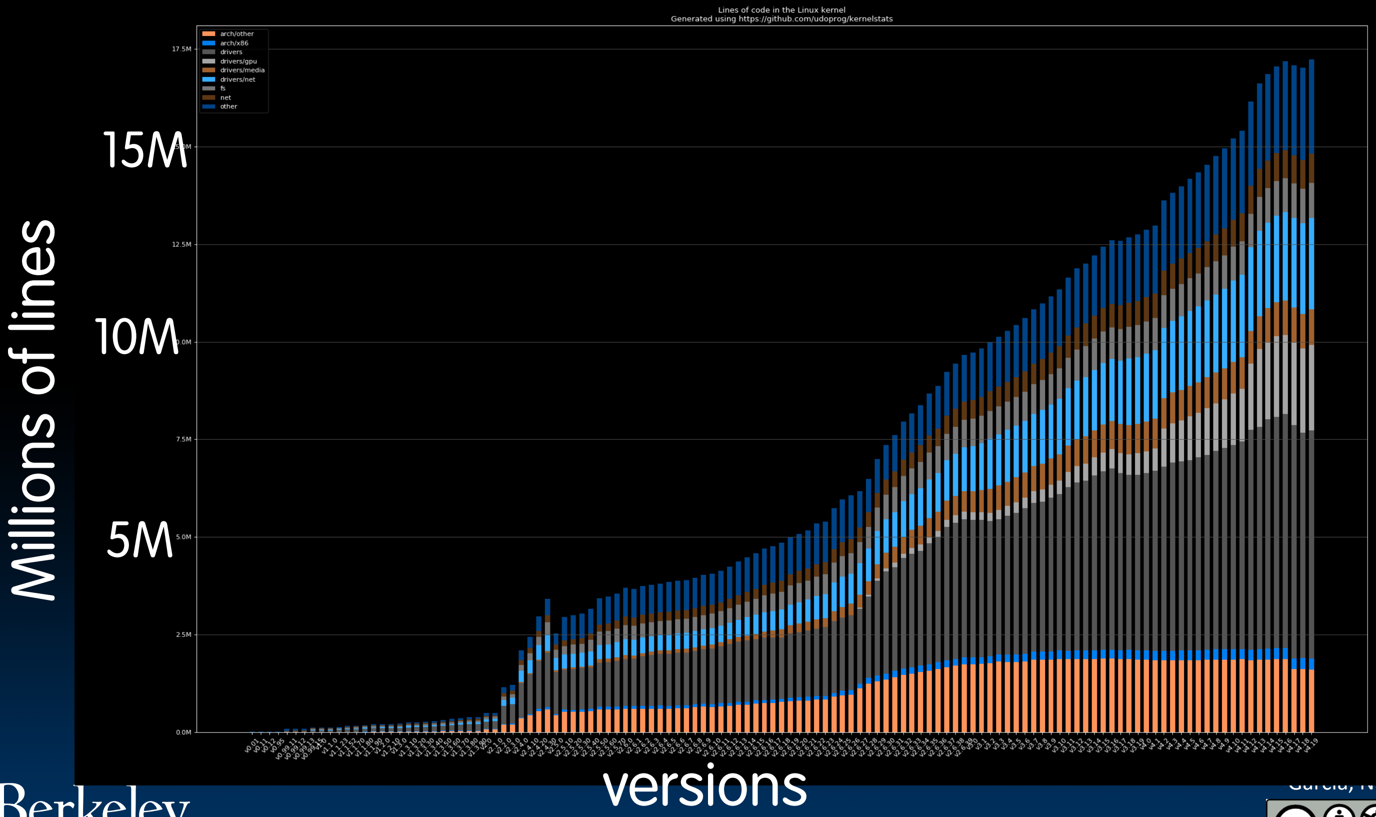
- The biggest piece of software on your machine?
- How many lines of code? These are guesstimates:



Codebases (in millions of lines of code). CC BY-NC 3.0 — David McCandless © 2015 http://www.informationisbeautiful.net/visualizations/million-lines-of-code/

Operating System

Lines of code in Linux kernel



What Does the OS do?

- OS is the (first) thing that runs when computer starts
- Finds and controls all devices in the machine in a general way
  - Relying on hardware specific "device drivers"
- Starts services (100+)
  - File system,
  - Network stack (Ethernet, WiFi, Bluetooth, …),
  - TTY (keyboard),
  - …
- Loads, runs and manages programs:
  - Multiple programs at the same time (time-sharing)
  - Isolate programs from each other (isolation)
  - Multiplex resources between applications (e.g., devices)

What Does the Core of the OS Do?

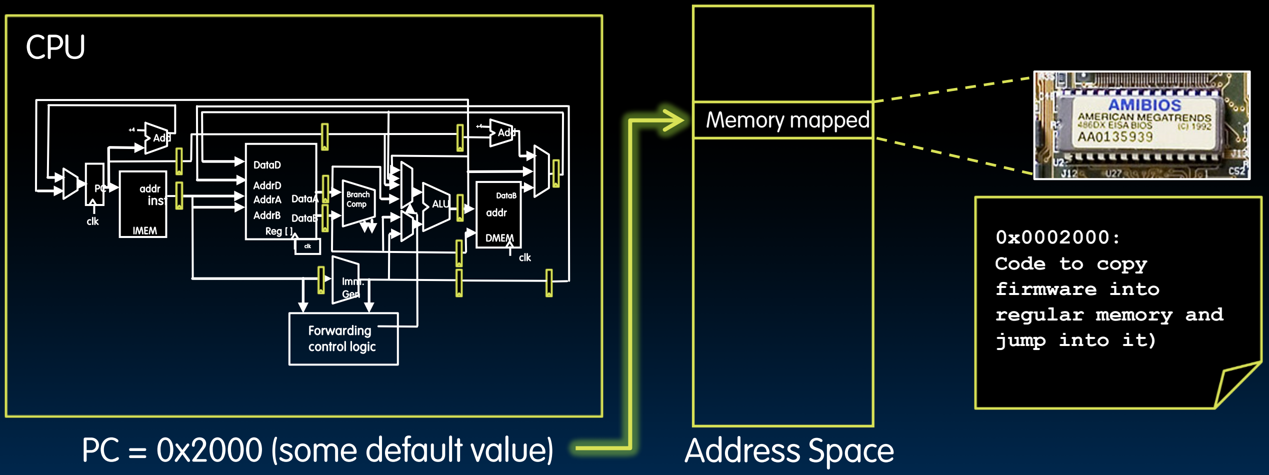
- Provide ***isolation*** between running processes
  - Each program runs in its own little world •
- Provide ***interaction*** with the outside world
  - Interact with "devices": Disk, display, network, etc... 11

What Does OS Need from Hardware?

- Memory translation
  - Each running process has a mapping from "virtual" to "physical" addresses that are different for each process
  - When you do a load or a store, the program issues a virtual address... But the actual memory accessed is a physical address
- Protection and privilege
  - Split the processor into at least two running modes: "User" and "Supervisor"
    - RISC-V also has "Machine" below "Supervisor"
  - Lesser privilege ***can not*** change its memory mapping
    - But "Supervisor" ***can*** change the mapping for any given program
    - And supervisor has its own set of mapping of virtual->physical
- Traps & Interrupts
  - A way of going into Supervisor mode on demand

What Happens at Boot?

- When the computer switches on, it does the same as VENUS: the CPU executes instructions from some start address (stored in Flash ROM)



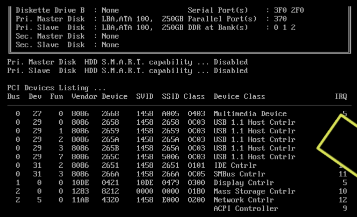
```
0x0002000:
Code to copy
firmware into
regular memory and
jump into it)
```

PC = 0x2000 (some default value)

Address Space

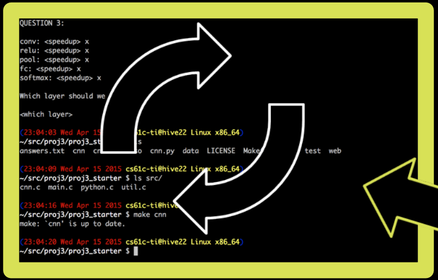
What Happens at Boot?

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



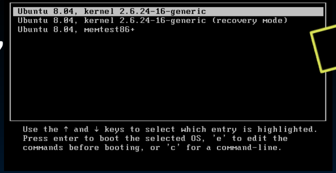
```
 Diskette Drive B  : None                  Serial Port(s)    : 3F0 2F0
 Pri. Master Disk  : LBA,ATA 100,  250GB Parallel Port(s)  : 370
 Pri. Slave  Disk  : LBA,ATA 100,  250GB DDR at Bank(s)    : 0 1 2
 Sec. Master Disk  : None
 Sec. Slave  Disk  : None

Pri. Master Disk  HDD S.M.A.R.T. capability ... Disabled
Pri. Slave  Disk  HDD S.M.A.R.T. capability ... Disabled

PCI Devices Listing ...
Bus  Dev  Fun  Vendor Device  SVID  SSID Class  Device Class              IRQ
-----------------------------------------------------------------------------
  0   27    0   8086   2668   1458  A005  0403  Multimedia Device            5
  0   29    0   8086   2658   1458  2658  0C03  USB 1.1 Host Cntrlr
  0   29    1   8086   2659   1458  2659  0C03  USB 1.1 Host Cntrlr
  0   29    2   8086   265A   1458  265A  0C03  USB 1.1 Host Cntrlr
  0   29    3   8086   265B   1458  265A  0C03  USB 1.1 Host Cntrlr
  0   29    7   8086   265C   1458  5006  0C03  USB 1.1 Host Cntrlr
  0   31    2   8086   2651   1458  2651  0101  IDE Cntrlr
  0   31    3   8086   266A   1458  266A  0C05  SMBus Cntrlr                11
  1    0    0   10DE   0421   10DE  0479  0300  Display Cntrlr               5
  2    0    0   1283   8212   0000  0000  0180  Mass Storage Cntrlr         10
  2    5    0   11AB   4320   1458  E000  0200  Network Cntrlr              12
                                                ACPI Controller              9
```

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

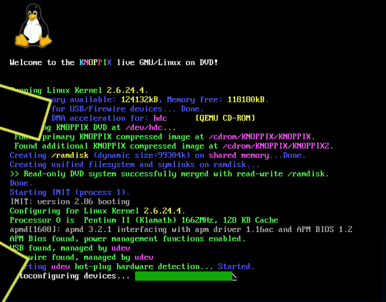


```
QUESTION 3:

conv: <speedup> x
relu: <speedup> x
pool: <speedup> x
fc: <speedup> x
softmax: <speedup> x

Which layer should we

<which layer>

(23:04:03 Wed Apr 15          1c-ti@hive22 Linux x86_64)
~/src/proj3/proj3_st          s
answers.txt  cnn  cr          o  cnn.py  data  LICENSE  Make          test  web

(23:04:09 Wed Apr 15 2015 cs61c-ti@hive22 Linux x86_64)
~/src/proj3/proj3_starter $ ls src/
cnn.c  main.c  python.c  util.c

(23:04:16 Wed Apr 15 2015 cs61c-ti@hive
~/src/proj3/proj3_starter $ make cnn
make: `cnn' is up to date.

(23:04:20 Wed Apr 15 2015 cs61c-ti@hive22 Linux x86_64)
~/src/proj3/proj3_starter $ █
```

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



```
 Ubuntu 8.04, kernel 2.6.24-16-generic
 Ubuntu 8.04, kernel 2.6.24-16-generic (recovery mode)
 Ubuntu 8.04, memtest86+

    Use the ↑ and ↓ keys to select which entry is highlighted.
    Press enter to boot the selected OS, 'e' to edit the
    commands before booting, or 'c' for a command-line.
```



```
Welcome to the KNOPPIX live GNU/Linux on DVD!

  nning Linux Kernel 2.6.24.4.
        ory available: 124132kB, Memory free: 118180kB.
        for USB/Firewire devices... Done.
        DMA acceleration for: hdc      [QEMU CD-ROM]
        ng KNOPPIX DVD at /dev/hdc...
 Fou    primary KNOPPIX compressed image at /cdrom/KNOPPIX/KNOPPIX.
 Found additional KNOPPIX compressed image at /cdrom/KNOPPIX/KNOPPIX2.
Creating /ramdisk (dynamic size=99304k) on shared memory...Done.
Creating unified filesystem and symlinks on ramdisk...
>> Read-only DVD system successfully merged with read-write /ramdisk.
Done.
Starting INIT (process 1).
INIT: version 2.86 booting
Configuring for Linux Kernel 2.6.24.4.
Processor 0 is  Pentium II (Klamath) 1662MHz, 128 KB Cache
apmd[1608]: apmd 3.2.1 interfacing with apm driver 1.16ac and APM BIOS 1.2
APM Bios found, power management functions enabled.
USB found, managed by udev
  wire found, managed by udev
  rting udev hot-plug hardware detection... Started.
  toconfiguring devices...
```

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

*BIOS: Basic Input Output System

Operating System Functions

Launching Applications

- Applications are called “processes” in most OSs
  - Thread: shared memory
  - Process: separate memory
  - Both threads and processes run (pseudo) simultaneously
- Apps are started by another process (e.g., shell) calling an OS routine (using a “syscall”)
  - Depends on OS; Linux uses fork to create a new process, and execve (execute file command) to load application
- Loads executable file from disk (using the file system service) and puts instructions & data into memory (.text, .data sections), prepares stack and heap
- Set argc and argv, jump to start of main
- Shell waits for main to return (join)

Supervisor Mode

- If something goes wrong in an application, it could crash the entire machine. And what about malware, etc.?
- The OS enforces resource constraints to applications (e.g., access to memory, devices)
- To help protect the OS from the application, CPUs have a supervisor mode (e.g., set by a status bit in a special register)
  - A process can only access a subset of instructions and (physical) memory when not in supervisor mode (user mode)
  - Process can change out of supervisor mode using a special instruction, but not into it directly – only using an interrupt
  - Supervisory mode is a bit like “superuser”
    - But used much more sparingly (most of OS code does *not* run in supervisory mode)
    - Errors in supervisory mode often catastrophic (blue “screen of death”, or “I just corrupted your disk”)

Garcia, Nikolić

Syscalls

- What if we want to call an OS routine? E.g.,
  - to read a file,
  - launch a new process,
  - ask for more memory (malloc),
  - send data, etc.
- Need to perform a syscall:
  - Set up function arguments in registers,
  - Raise software interrupt (with special assembly instruction)
- OS will perform the operation and return to user mode
- This way, the OS can mediate access to all resources, and devices

Interrupts, Exceptions

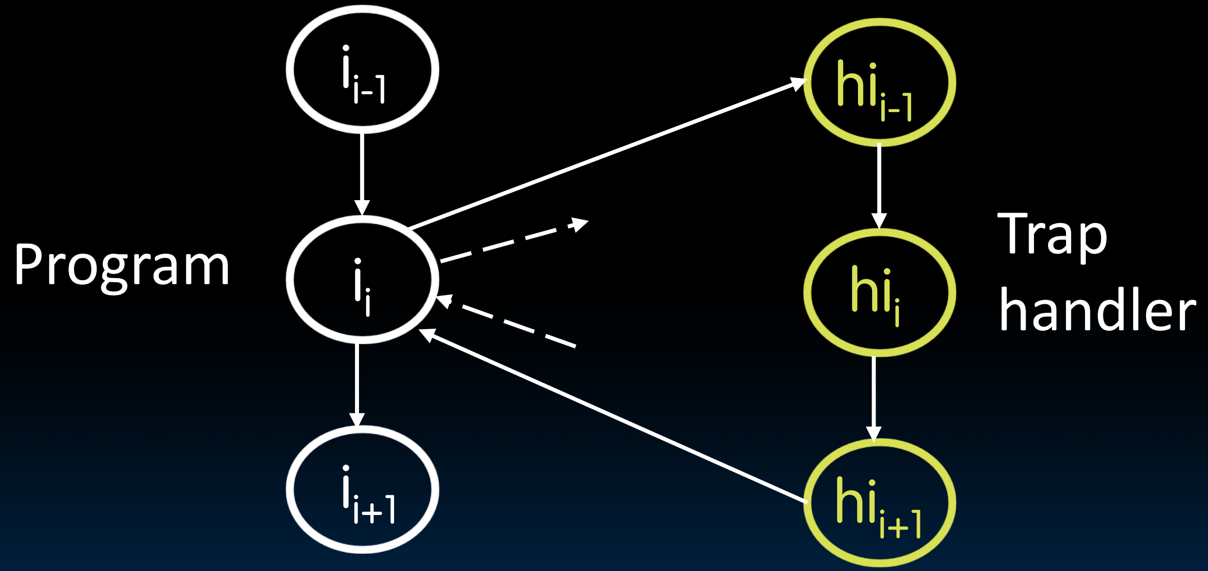
- We need to transition into Supervisor mode when "something" happens
- Interrupt: Something external to the running program
  - Something happens from the outside world
- Exception: Something done by the running program
  - Accessing memory it isn't "supposed" to, executing an illegal instruction, reading a csr not supposed at that privilege
- **ECALL**: Trigger an exception to the higher privilege
  - How you communicate with the operating system: Used to implement "syscalls"
- **EBREAK**: Trigger an exception within the current privilege

Terminology (In 61C)

- Interrupt – caused by an event *external* to current running program
  - E.g., key press, disk I/O
  - Asynchronous to current program
    - Can handle interrupt on any convenient instruction
    - “Whenever it’s convenient, just don’t wait too long”
- Exception – caused by some event *during* execution of one instruction of current running program
  - E.g., memory error, bus error, illegal instruction, raised exception
  - Synchronous
    - Must handle exception *precisely* on instruction that causes exception
    - “Drop whatever you are doing and act now”
- Trap – action of servicing interrupt or exception by hardware jump to “interrupt or trap handler” code

Trap Handling

- Altering the regular execution flow

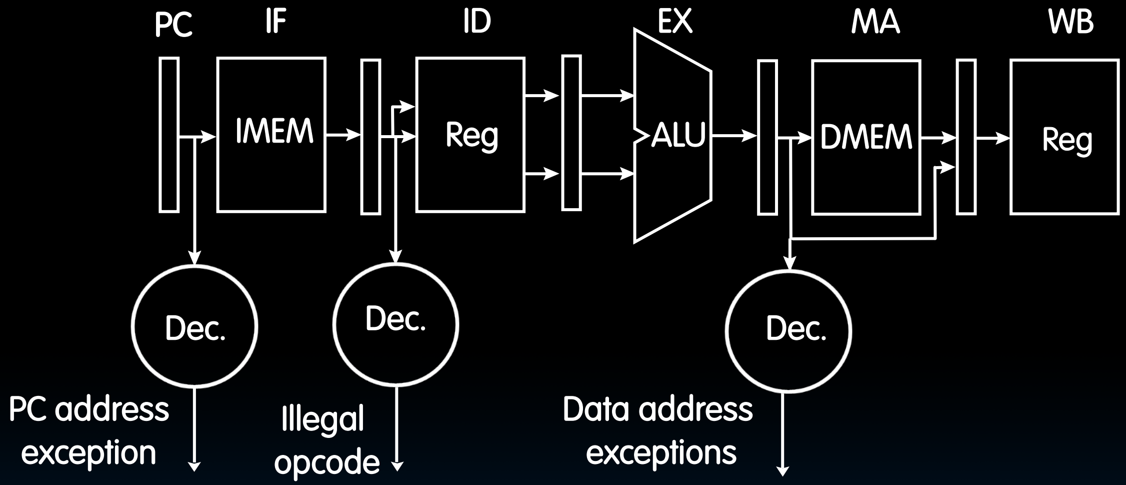


An *external or internal event* that needs to be processed - by another program; often handled by OS. The event is often unexpected from original program's point of view.

Precise Traps

- *Trap handler's view of machine state is that every instruction prior to the trapped one (e.g., memory error) 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
  - 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 a requirement for things to actually work right!

Exceptions in a 5-Stage Pipeline



Trap Handling

- Exceptions are handled *like pipeline hazards*
  1) Complete execution of instructions before exception occurred
  2) Flush instructions currently in pipeline (i.e., convert to **`nops`** or "bubbles")
  3) Optionally store exception cause in status register
    - Indicate type of exception
- 4) Transfer execution to trap handler
- 5) Optionally, return to original program and re-execute instruction

Multiprogramming

- The OS runs multiple applications at the same time
- But not really (unless you have a core per process)
- Switches between processes very quickly (on human time scale) – this is called a “context switch”
- When jumping into process, set timer (we will call this ‘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
- Deciding what process to run is called scheduling

Protection, Translation, Paging

- Supervisor mode alone is not sufficient to fully isolate applications from each other or from the OS
  - Application could overwrite another application’s memory.
  - Typically programs start at some fixed address, e.g. 0x8FFFFFFF
    - How can 100’s of programs share memory at location 0x8FFFFFFF?
  - Also, may want to address more memory than we actually have (e.g., for sparse data structures)
- Solution: Virtual Memory
  - Gives each process the *illusion* of a full memory address space that it has completely for itself