IT Club

ADVANCED C WORKSHOP

Day 4

Overview of the Computer Architecture

Overview

- How the CPU does what it does
  - Understand how the CPU in your computer works
  - Build a good model of your CPU in your mind
- How can we use that to write better code
  - Focus on “hot” code
  - Use matrix multiplication as an example
- Focus on x86_64, Linux

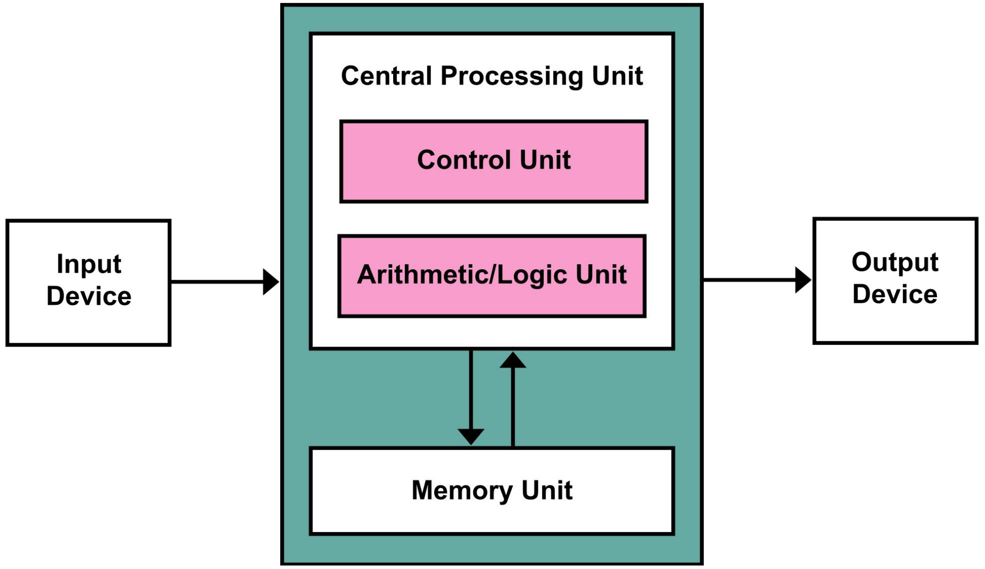
Why study architecture

IT Club

- It is cool and fun!
  - Solve real world problems
  - Solid engineering solutions
  - Very active area of research
- It will help us write better and faster code
  - Moore’s Law and friends are maybe dead (kind of)
    - Atoms are ~0.5nm, transistors <50nm

What the CPU does

- Crunch numbers
- Input, Process, Output



Von Neumann Architecture

Memory

IT Club

Different kinds

- Registers - what CPU needs right now
- RAM - everything else needed to do a job
- Secondary storage - things we might need later

Registers

- Small but very fast storage (~1 cycles)
- Built right in the CPU
- Can be general purpose
  - x86_64 has 16 general purpose 64-bit registers
  - Arm (and other RISCs) have >32
- Can be specialized
  - Store state (PC, SP, offsets etc)
  - SIMD etc.

RAM

- Larger memory, decently fast (~100 cycles)
- Runs independently from the CPU
- Different clock speed

Virtual Memory

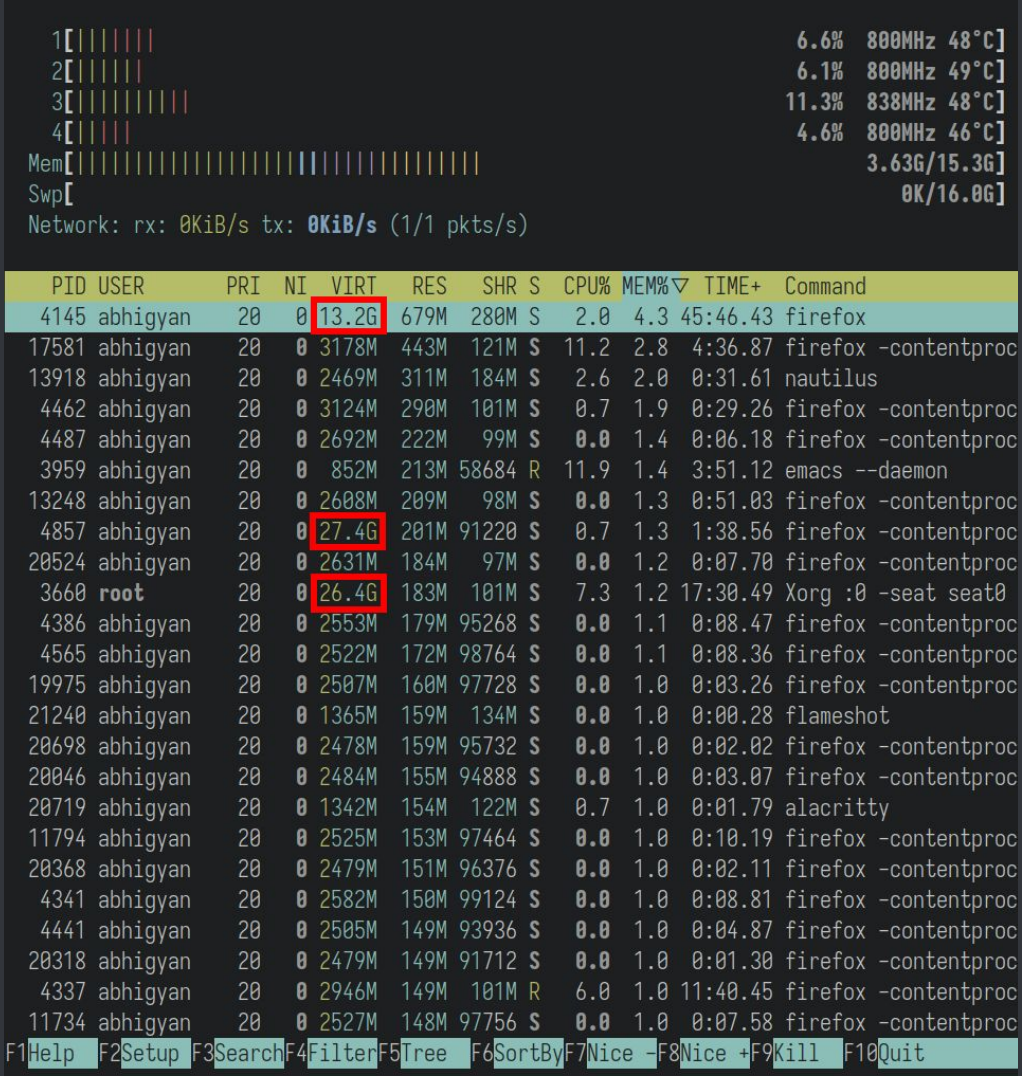
- Makes our life easier
- Idealized abstraction over storage resources
- Illusion
  - RAM is very large
  - Your process is the only thing running
  - Memory is linear and unfragmented
    - You ask for 4GB, you get nice clean 4GB block

Why Virtualize

- Makes our life much easier
- We can move pages between RAM and disk
  - Linux has swaps, Windows has ReadyBoost
  - You can run 10 processes that each need 1 GB even if you only have 4GBs of RAM
  - Performance hit is substantial, but it’s better than giving up

Why Virtualize

- Files and I/O devices can be mapped to RAM
  - Use the RAM as a buffer
  - Faster reads and writes
- Programs can be memory position independent
  - Useful for dynamic/shared libraries (plugins)
- Trick OS into using more memory than there is



```
    1[|||||||                                                      6.6%  800MHz 48°C]
    2[||||||                                                       6.1%  800MHz 49°C]
    3[||||||||||                                                  11.3%  838MHz 48°C]
    4[|||||                                                        4.6%  800MHz 46°C]
  Mem[|||||||||||||||||||||||||||||||||||                                3.63G/15.3G]
  Swp[                                                                      0K/16.0G]
  Network: rx: 0KiB/s tx: 0KiB/s (1/1 pkts/s)

    PID USER       PRI  NI  VIRT   RES   SHR S  CPU% MEM%▽  TIME+  Command
   4145 abhigyan    20   0 13.2G  679M  280M S   2.0  4.3 45:46.43 firefox
  17581 abhigyan    20   0 3178M  443M  121M S  11.2  2.8  4:36.87 firefox -contentproc
  13918 abhigyan    20   0 2469M  311M  184M S   2.6  2.0  0:31.61 nautilus
   4462 abhigyan    20   0 3124M  290M  101M S   0.7  1.9  0:29.26 firefox -contentproc
   4487 abhigyan    20   0 2692M  222M   99M S   0.0  1.4  0:06.18 firefox -contentproc
   3959 abhigyan    20   0  852M  213M 58684 R  11.9  1.4  3:51.12 emacs --daemon
  13248 abhigyan    20   0 2608M  209M   98M S   0.0  1.3  0:51.03 firefox -contentproc
   4857 abhigyan    20   0 27.4G  201M 91220 S   0.7  1.3  1:38.56 firefox -contentproc
  20524 abhigyan    20   0 2631M  184M   97M S   0.0  1.2  0:07.70 firefox -contentproc
   3660 root        20   0 26.4G  183M  101M S   7.3  1.2 17:30.49 Xorg :0 -seat seat0
   4386 abhigyan    20   0 2553M  179M 95268 S   0.0  1.1  0:08.47 firefox -contentproc
   4565 abhigyan    20   0 2522M  172M 98764 S   0.0  1.1  0:08.36 firefox -contentproc
  19975 abhigyan    20   0 2507M  160M 97728 S   0.0  1.0  0:03.26 firefox -contentproc
  21240 abhigyan    20   0 1365M  159M  134M S   0.0  1.0  0:00.28 flameshot
  20698 abhigyan    20   0 2478M  159M 95732 S   0.0  1.0  0:02.02 firefox -contentproc
  20046 abhigyan    20   0 2484M  155M 94888 S   0.0  1.0  0:03.07 firefox -contentproc
  20719 abhigyan    20   0 1342M  154M  122M S   0.7  1.0  0:01.79 alacritty
  11794 abhigyan    20   0 2525M  153M 97464 S   0.0  1.0  0:10.19 firefox -contentproc
  20368 abhigyan    20   0 2479M  151M 96376 S   0.0  1.0  0:02.11 firefox -contentproc
   4341 abhigyan    20   0 2582M  150M 99124 S   0.0  1.0  0:08.81 firefox -contentproc
   4441 abhigyan    20   0 2505M  149M 93936 S   0.0  1.0  0:04.87 firefox -contentproc
  20318 abhigyan    20   0 2479M  149M 91712 S   0.0  1.0  0:01.30 firefox -contentproc
   4337 abhigyan    20   0 2946M  149M  101M R   6.0  1.0 11:40.45 firefox -contentproc
  11734 abhigyan    20   0 2527M  148M 97756 S   0.0  1.0  0:07.58 firefox -contentproc
F1Help  F2Setup  F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

Memory Management Unit

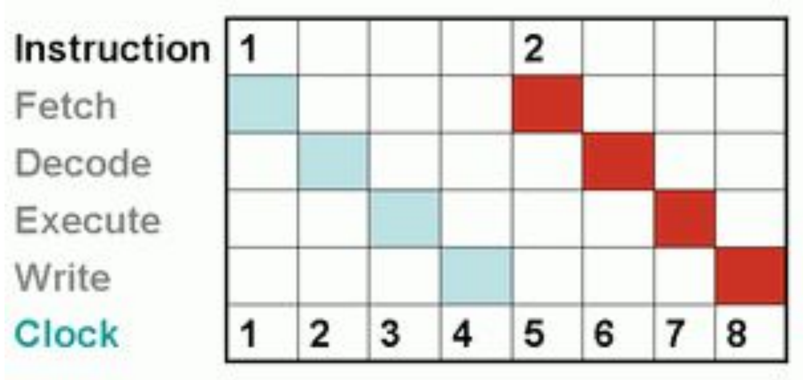
- Address translation has overheads
- MMU does this to free the CPU
- MMU is a hardware unit in the CPU
- Kernel populates tables in the MMU
- MMU translates on the fly
  - We can cache!

Translation Lookaside Buffer

- Caches virtual address to physical address translation
- Is usually hierarchical
- May have separate sections for small and large pages
- Typical TLBs have ~1000 entries, <1% miss rate
- Miss penalty is ~100 cycles
- TLB thrashing - lots of TLB misses

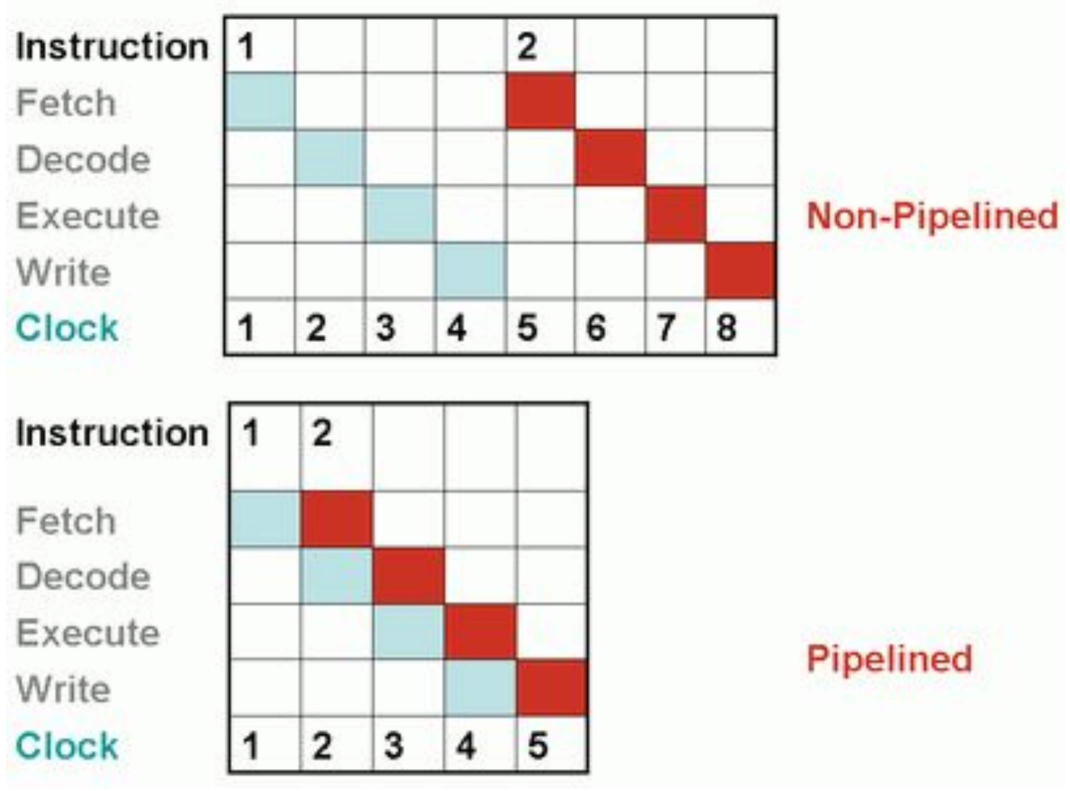
Instruction Cycle

- Cycle that a CPU follows until powered off
- Fetch - read next instruction from RAM[PC]
- Decode - Interpret the instruction
- Execute - Do the actual work
- Write - store the results
- Repeat



Instruction	1				2			
Fetch	🟦				🟥			
Decode		🟦				🟥		
Execute			🟦				🟥	
Write				🟦				🟥
Clock	1	2	3	4	5	6	7	8

Pipelining - Instruction Level Parallelism



Instruction	1				2			
Fetch	🟦				🟥			
Decode		🟦				🟥		
Execute			🟦				🟥	
Write				🟦				🟥
Clock	1	2	3	4	5	6	7	8

Non-Pipelined

Instruction	1	2			
Fetch	🟦	🟥			
Decode		🟦	🟥		
Execute			🟦	🟥	
Write				🟦	🟥
Clock	1	2	3	4	5

Pipelined

Branches

IT Club

- Sometimes we don’t know which instruction will come next
- Conditional Jumps / Branches
- Wait until we know exactly what will happen?
  - Bad, we have to flush the pipeline and start over
- Choose one randomly?
  - Great only when we are right

Branch Prediction

- Use some strategy to predict outcomes
- Simple strategies work well
    - Branches are always taken
    - Backward branches are taken, forward branches are not
- We can go further
    - Cache the paths taken previously to help BP
    - Use Neural Networks (because why not?)

Helping Branch Prediction

- We cannot override BP, but we can help
- Modern processors guess that backward branches are taken
- We can ask the compiler to lay code out to help with this

```
if (__builtin_expect(foo==bar)) {...}
```

Cache Memory

- Each trip to the RAM takes ~100 cycles
- For context, integer addition is ~1 cycle (pipelined)
- Let’s cache data that we might need
    - If you access x, you might access x’s neighbors too
        - Spatial locality
    - If you access x 10 times, you might access it again
        - Temporal locality

Cache Levels

IT Club

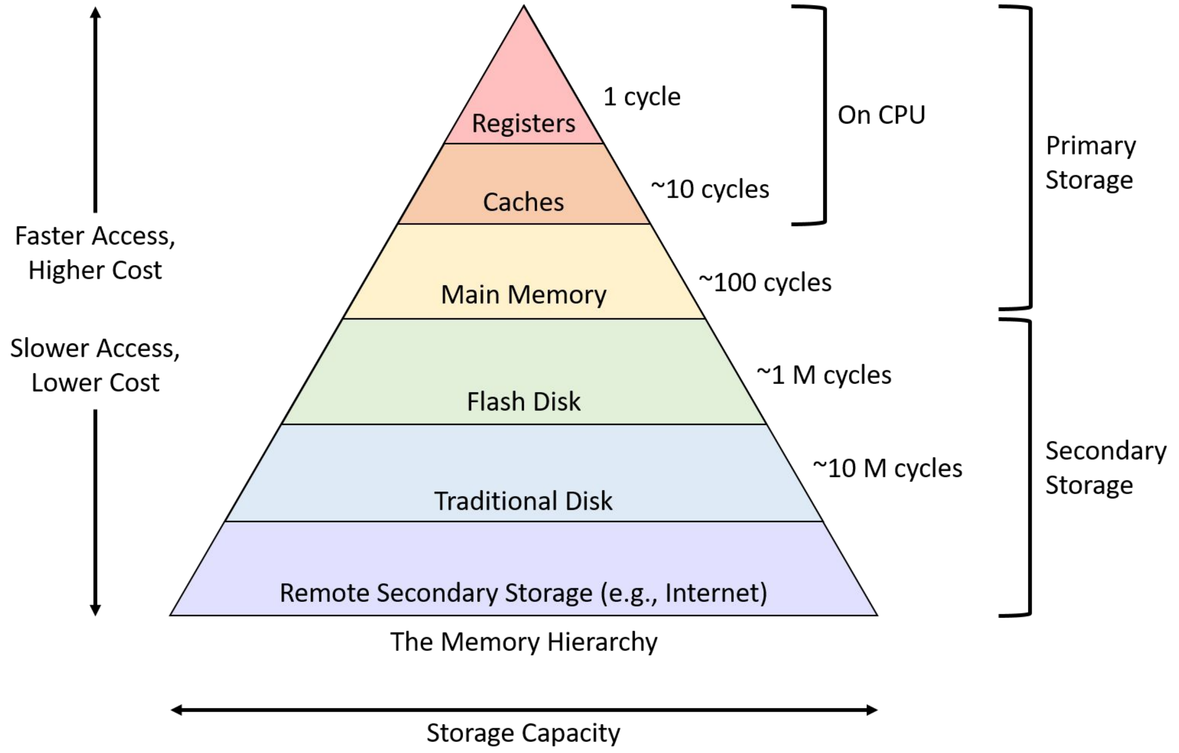
- Practical CPUs have multiple levels of cache
    - L1 cache - 32KB, 5 cycles
    - L2 cache - 1024KB, 70 cycles
    - L3 cache - 10MB, 80 cycles

Cache Eviction

- Not everything fits in the cache
- To add, we need to evict
- Algorithms choose pages to evict
  - Least Frequently Used
  - Least Recently Used
- L1 cache has simple eviction strategy
- Higher cache levels are more complicated

Memory Hierarchy

IT Club



The Memory Hierarchy

Same Instruction, Multiple Data

IT Club

- Find the sum of a large array (databases)
- Multiply a lot of number pairs (matrix multiplication)
- Invert, XOR a large stream of data (hash functions)

Same Instruction, Multiple Data

- Modern processors have SIMD instructions
- Intel and AMD have SSE, AVX1, AVX2 etc
- Arm has Neon
- SIMD enables data parallelism
  - You can do the same thing on multiple pieces of data
- SIMD is not concurrency
  - You cannot do multiple kinds of things at the same time

Superscalar CPUs

IT Club

- CPU has different circuits and units for different things
- Idea: do multiple things at once
- Only possible with careful data dependency checks
- Enables execution of multiple instructions per clock cycle

Simultaneous Multi-Threading (SMT)

IT Club

- Multiple execution units in the same CPU
- With careful pipelining, CPUs can ~double the performance
- Eg: Hyperthreading, AMD SMT
- (X cores, 2*X threads)

Multiprocessing

- Everything so far happens in one core
- Most CPUs have multiple cores
- Some problems are embarrassingly parallel
  - Use N CPUs, get close to Nx boost

Speculative Execution

IT Club

- Do work before you need to do it
- Needs careful planning and dependency checks
- Eg: Branch Prediction
- Thread Level Speculation - SE in multithreading systems
- Has caused problems in the past
    - Look into Spectre, Meltdown etc

IT Club

ADVANCED C WORKSHOP

THANK YOU!