

CS250P: Computer Systems Architecture

The Hardware/Software Interface



Sang-Woo Jun

Fall 2023



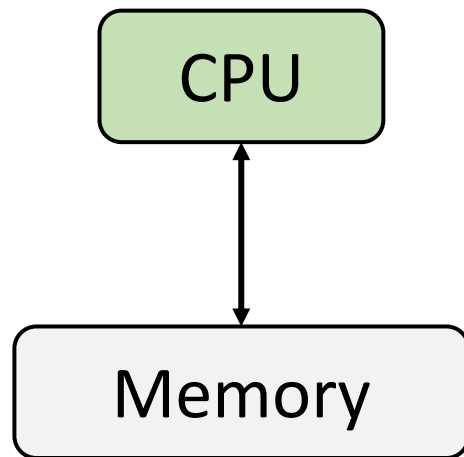
Large amount of material adapted from MIT 6.004, “Computation Structures”,
Morgan Kaufmann “Computer Organization and Design: The Hardware/Software Interface: RISC-V Edition”,
and CS 152 Slides by Isaac Scherson

Course outline

- ❑ Part 1: The Hardware-Software Interface
 - What makes a 'good' processor?
 - Assembly programming and conventions
- ❑ Part 2: Recap of digital design
 - Combinational and sequential circuits
 - How their restrictions influence processor design
- ❑ Part 3: Computer Architecture
 - Simple and pipelined processors
 - Out-of-order and explicitly parallel architectures
 - Caches and the memory hierarchy
- ❑ Part 4: Computer Systems
 - Operating systems, Virtual memory

A simple hardware abstraction

```
for (int i = 0; i < N; i++) {  
    doWork(data[i]);  
}
```



execute "doWork(0x1)"

execute "doWork(0x8)"

request data[0]

response "0x1"

...

request data[1]

response "0x8"

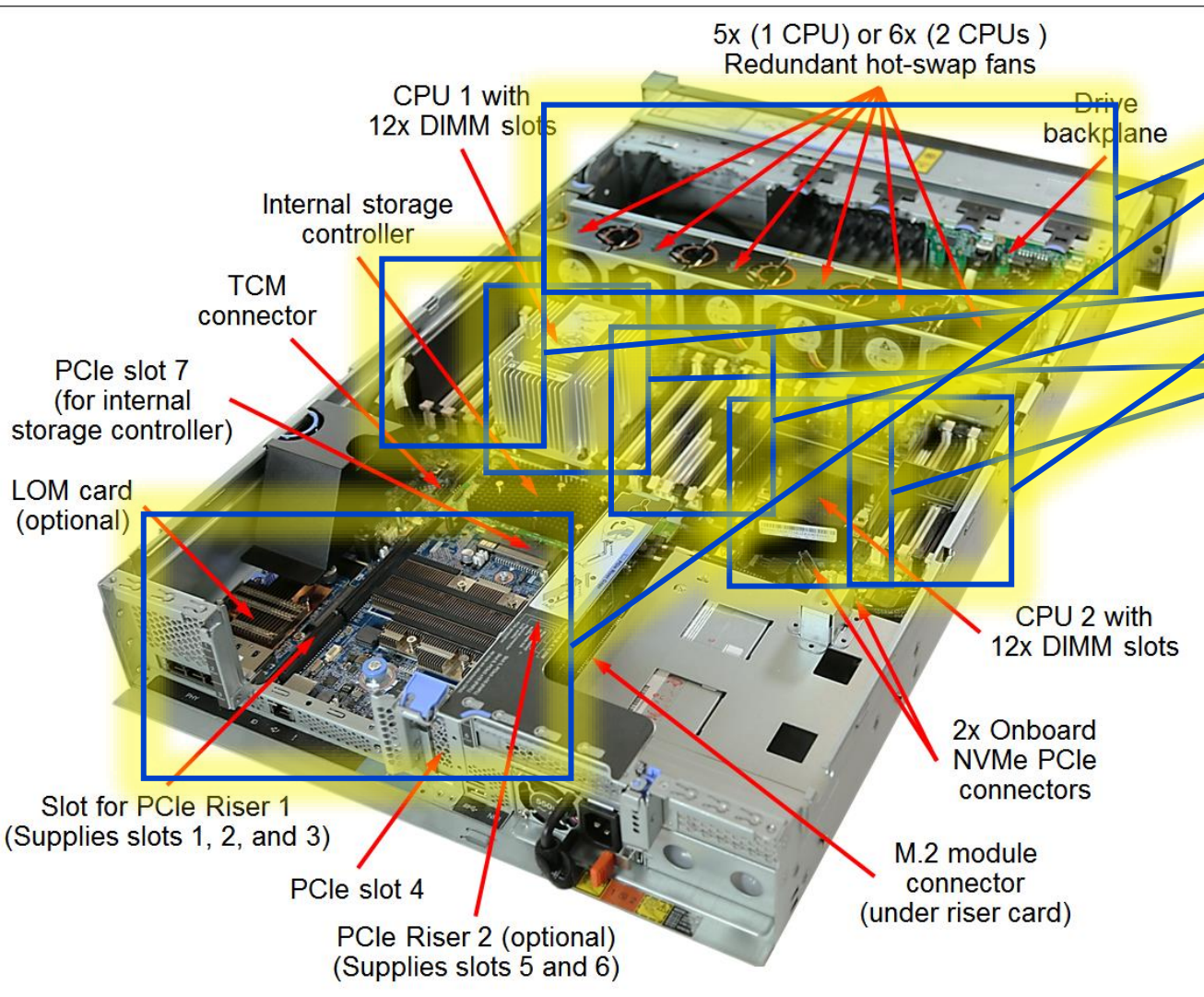
Is this good architecture?

no... CPU always idle waiting for memory

1 CPU instruction: ~0.3 ns (@ 3 GHz)

1 DRAM access: 10 - 100 ns

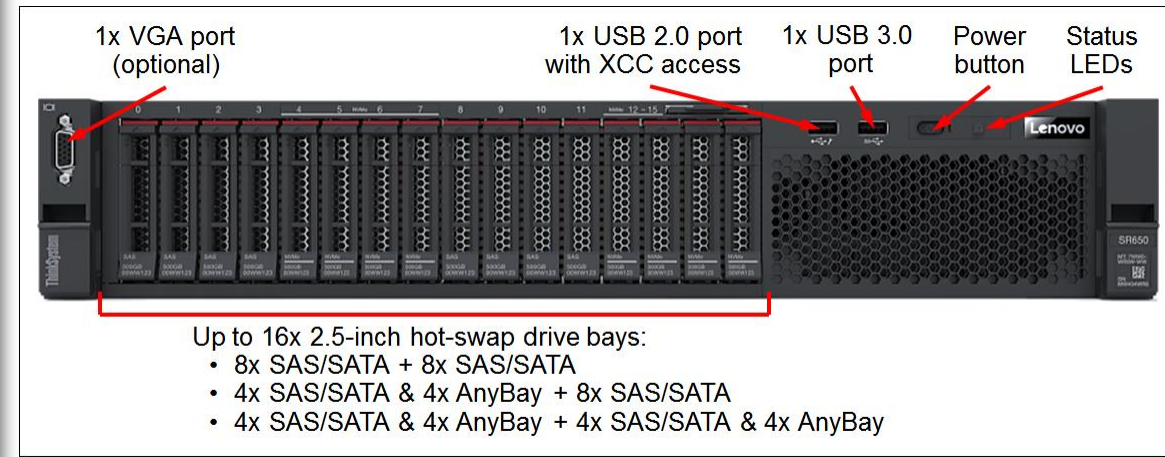
A modern server system hardware



I/O expansion (SATA, PCIe, ...)

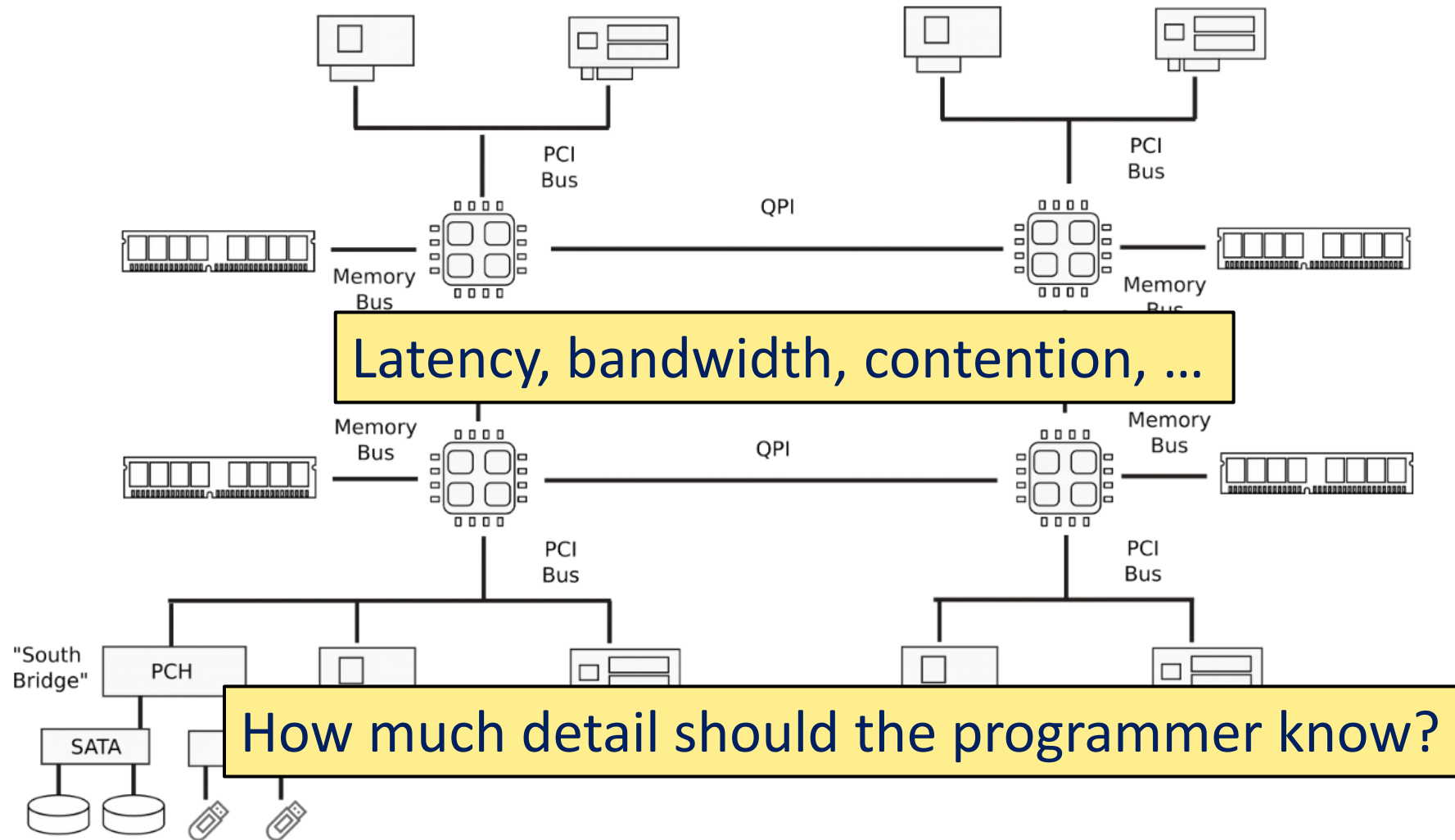
Memory DIMMs per CPU socket

1+ CPU sockets



Lenovo ThinkSystem SR650 Server

A modern server system hardware



Eight great ideas

- Design for Moore's Law
- Use abstraction to simplify design ^{today}
- Make the common case fast
- Performance via parallelism
- Performance via pipelining
- Performance via prediction
- Hierarchy of memories
- Dependability via redundancy

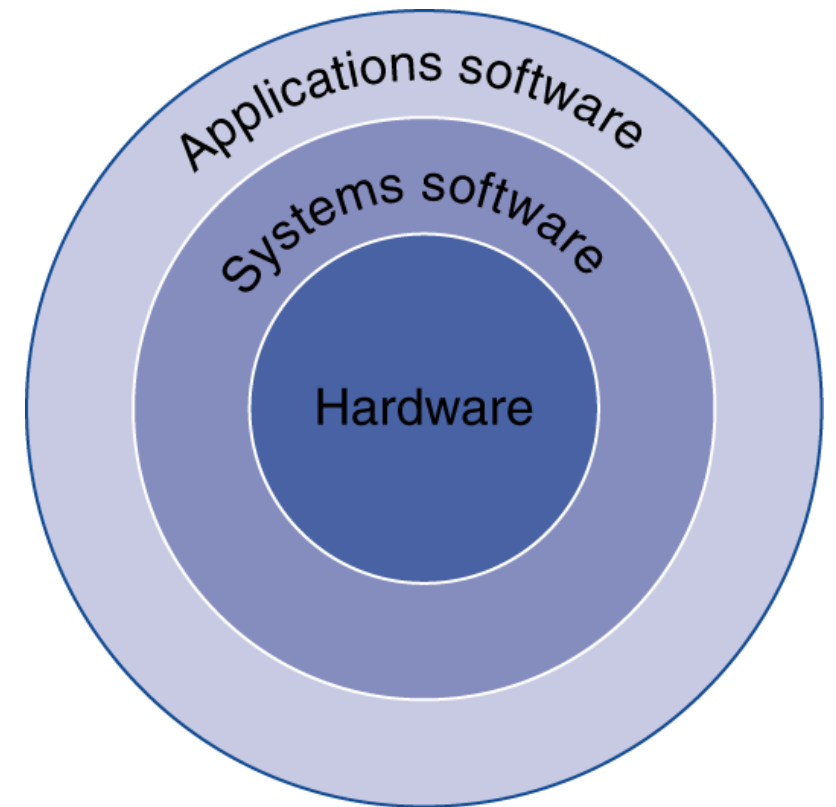


Great idea: Use abstraction to simplify design

- Abstraction helps us deal with complexity by hiding lower-level detail
 - One of the most fundamental tools in computer science!
 - Examples:
 - Application Programming Interface (API),
 - System calls,
 - Application Binary Interface (ABI),
 - Instruction-Set Architecture

Below your program

- ❑ Application software
 - Written in high-level language (typically)
- ❑ System software
 - Compiler: translates HLL code to machine code
 - Operating System: service code
 - Handling input/output
 - Managing memory and storage
 - Scheduling tasks & sharing resources
- ❑ Hardware
 - Processor, memory, I/O controllers



The Instruction Set Architecture

- ❑ An Instruction-Set Architecture (ISA) is the abstraction between the software and processor hardware
 - The 'Hardware/Software Interface'
 - Different from 'Microarchitecture', which is how the ISA is implemented
- ❑ A consistent ISA allows software to run on different machines of the same architecture
 - e.g., x86 across Intel, AMD, and various speed and power ratings

Levels of program code

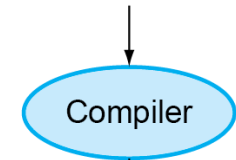
- ❑ High-level language
 - Level of abstraction closer to problem domain
 - Provides for productivity and portability
- ❑ Assembly language
 - Textual representation of instructions
- ❑ Hardware representation
 - Binary digits (bits)
 - Encoded instructions and data

Instruction Set Architecture (ISA) is the agreement on what this will do



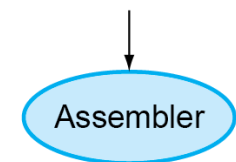
High-level
language
program
(in C)

```
swap(int v[], int k)
{int temp;
  temp = v[k];
  v[k] = v[k+1];
  v[k+1] = temp;
}
```



Assembly
language
program
(for RISC-V)

```
swap:
  slli x6, x11, 3
  add  x6, x10, x6
  ld   x5, 0(x6)
  ld   x7, 8(x6)
  sd   x7, 0(x6)
  sd   x5, 8(x6)
  jalr x0, 0(x1)
```



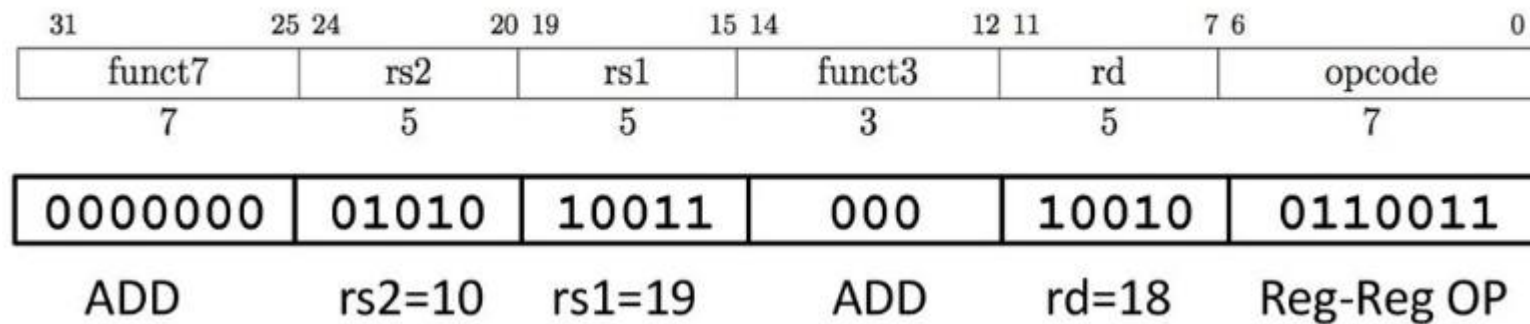
Binary machine
language
program
(for RISC-V)

```
00000000001101011001001100010011
00000000011001010000001100110011
00000000000000110011001010000011
00000000100000110011001110000011
00000000011100110011000000100011
00000000010100110011010000100011
00000000000000001000000011001111
```

A RISC-V Example (“00A9 8933”)

- This four-byte binary value will instruct a RISC-V CPU to perform
 - add values in registers x19 x10, and store it in x18
 - regardless of processor speed, internal implementation, or chip designer
 - Various “microarchitectures” adhere to same ISA, with cost/performance/etc tradeoffs

`add x18,x19,x10`



Some history of ISA

- ❑ Early mainframes did not have a concept of ISAs (early 1960s)
 - Each new system had different hardware-software interfaces
 - Software for each machine needed to be re-built
- ❑ IBM System/360 (1964) introduced the concept of ISAs
 - Same ISA shared across five different processor designs (various cost!)
 - Same OS, software can be run on all
 - Extremely successful!
- ❑ Aside: Intel x86 architecture introduced in 1978
 - Strict backwards compatibility maintained even now (The A20 line... 😞)
 - Attempted clean-slate redesign multiple times but failed (iAPX 432, EPIC, ...)

IBM System/360 Model 20 CPU



CS250P: Computer Systems Architecture

What Makes a “Good” ISA?



Sang-Woo Jun

Fall 2023

What makes a 'good' ISA?

- ❑ Computer architecture is a complicated art...
 - No one design method leads to a 'best' computer
 - Subject to workloads, use patterns, criterion, operation environment, ...
- ❑ Important criteria: Given the same restrictions,
 - High performance!
 - Power efficiency
 - Low cost
 - ...
- ❑ May depend on target applications
 - E.g., Apple knows (and cares) more about its software than Intel

What does it mean to be high-performance?

- ❑ In the 90s, CPUs used to compete with clock speed
 - “My 166 MHz processor was faster than your 100 MHz processor!”
 - Not very representative between different architectures
 - 2 GHz processor may require 5 instructions to do what 1 GHz one needs only 2
 - (Or not!)

- ❑ Sometimes ISA designers make trade-offs
 - E.g., Capability of each instruction vs. Circuit simplicity (=> Faster clock)
 - Which choice is better?

What does it mean to be high-performance?

- ❑ Let's define performance = $1/\text{execution time}$
- ❑ Example: time taken to run a program
 - 10s on A, 15s on B
 - $\text{Execution Time}_B / \text{Execution Time}_A$
 $= 15s / 10s = 1.5$
 - So A is 1.5 times faster than B

$$\begin{aligned} & \text{Performance}_x / \text{Performance}_y \\ &= \text{Execution time}_y / \text{Execution time}_x = n \end{aligned}$$

Measuring execution time

❑ Elapsed time

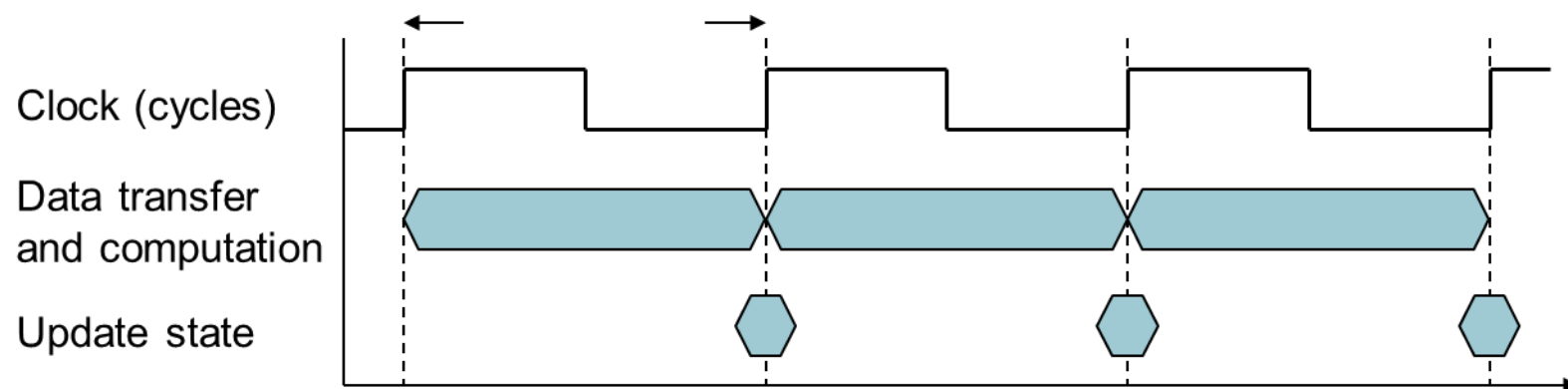
- Total response time, including all aspects
 - Processing, I/O, OS overhead, idle time
- Determines system performance

❑ CPU time *(Focus here for now)*

- Time spent by the processor on a given job
 - Ignores I/O time, other peoples' jobs' shares
- Consists of user CPU time and system (OS) CPU time
- Different programs are affected differently by CPU and system performance

CPU clocking

- ❑ Operation of digital hardware governed by a constant-rate clock



- ❑ Clock period: duration of a clock cycle
 - e.g., $250\text{ps} = 0.25\text{ns} = 250 \times 10^{-12}\text{s}$
- ❑ Clock frequency (rate): cycles per second
 - e.g., $4.0\text{GHz} = 4000\text{MHz} = 4.0 \times 10^9\text{Hz}$

CPU time

- ❑ CPU time = Clock cycle * clock cycle time
- ❑ Performance improved by
 - Reducing number of clock cycles
 - Increasing clock rate
- ❑ Hardware designer must often trade off clock rate against cycle count

$$\begin{aligned}\text{CPU Time} &= \text{CPU Clock Cycles} \times \text{Clock Cycle Time} \\ &= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}\end{aligned}$$

Instruction count and CPI

- ❑ Instruction Count for a program
 - Determined by program, ISA and compiler
- ❑ Average cycles per instruction
 - Determined by CPU hardware
 - If different instructions have different CPI
 - Average CPI affected by instruction mix

Clock Cycles = Instruction Count × Cycles per Instruction

CPU Time = Instruction Count × CPI × Clock Cycle Time

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

CPI example

- ❑ Computer A: Cycle Time = 250ps, CPI = 2.0
- ❑ Computer B: Cycle Time = 500ps, CPI = 1.2
- ❑ Same ISA

$$\begin{aligned}\text{CPU Time}_A &= \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A \\ &= 1 \times 2.0 \times 250\text{ps} = 1 \times 500\text{ps}\end{aligned}$$

A is faster...

$$\begin{aligned}\text{CPU Time}_B &= \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B \\ &= 1 \times 1.2 \times 500\text{ps} = 1 \times 600\text{ps}\end{aligned}$$

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{1 \times 600\text{ps}}{1 \times 500\text{ps}} = 1.2$$

...by this much

CPI in more detail

- ❑ If different instruction classes take different numbers of cycles

$$\text{Clock Cycles} = \sum_{i=1}^n (\text{CPI}_i \times \text{Instruction Count}_i)$$

*Not always true with microarchitectural tricks
(Pipelining, superscalar, ...)

- ❑ Weighted average CPI

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^n \left(\text{CPI}_i \times \frac{\text{Instruction Count}_i}{\text{Instruction Count}} \right)$$

Dynamic profiling!

Relative frequency

Performance summary

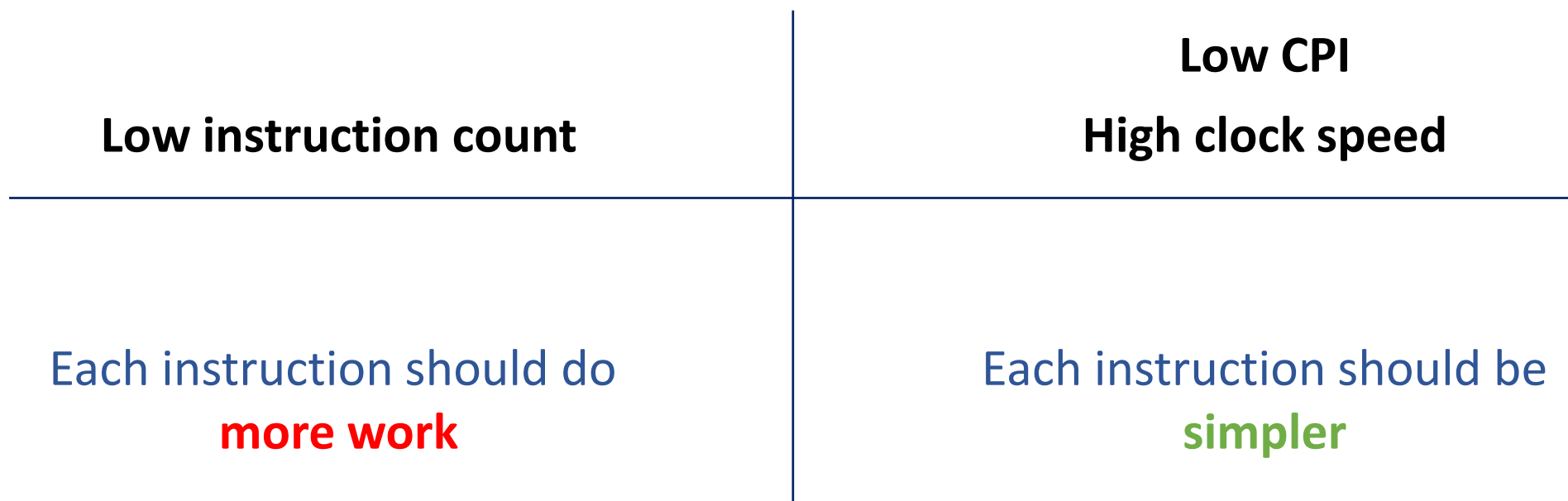
□ Performance depends on

- Algorithm: affects Instruction count, (possibly CPI)
- Programming language: affects Instruction count, (possibly CPI)
- Compiler: affects Instruction count, CPI
- Instruction set architecture: affects Instruction count, CPI, Clock speed

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

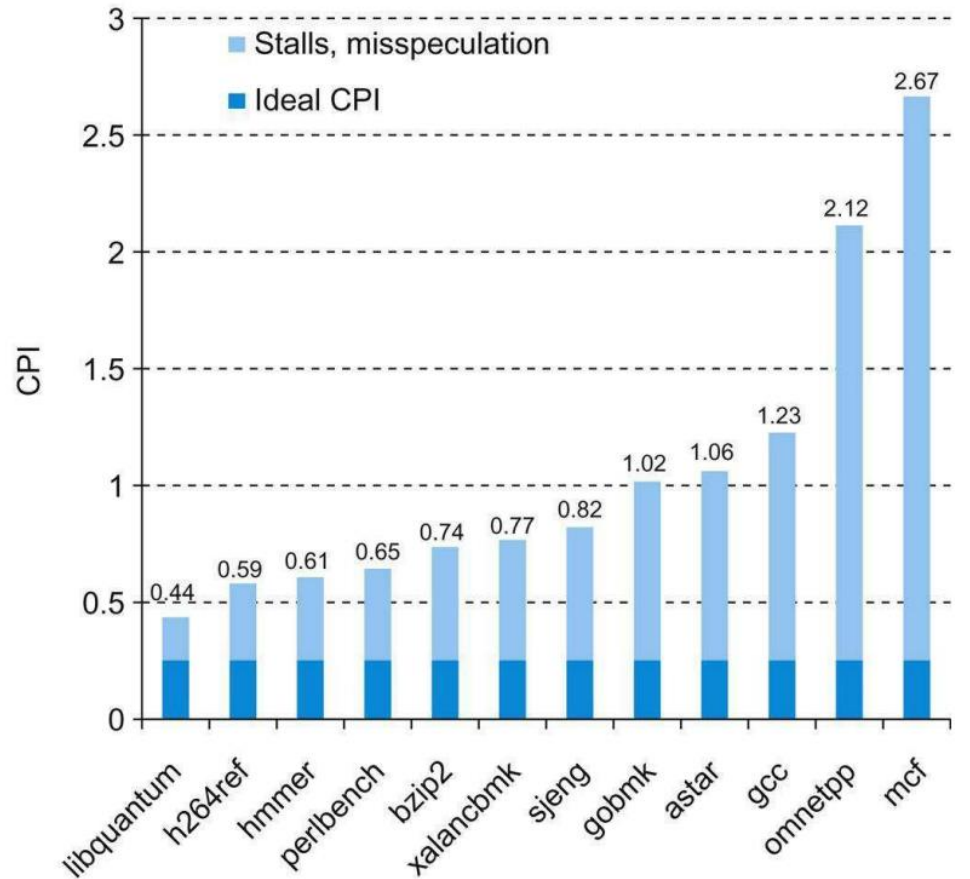
A good ISA: Low instruction count, Low CPI, High clock speed

Some goals for a good ISA

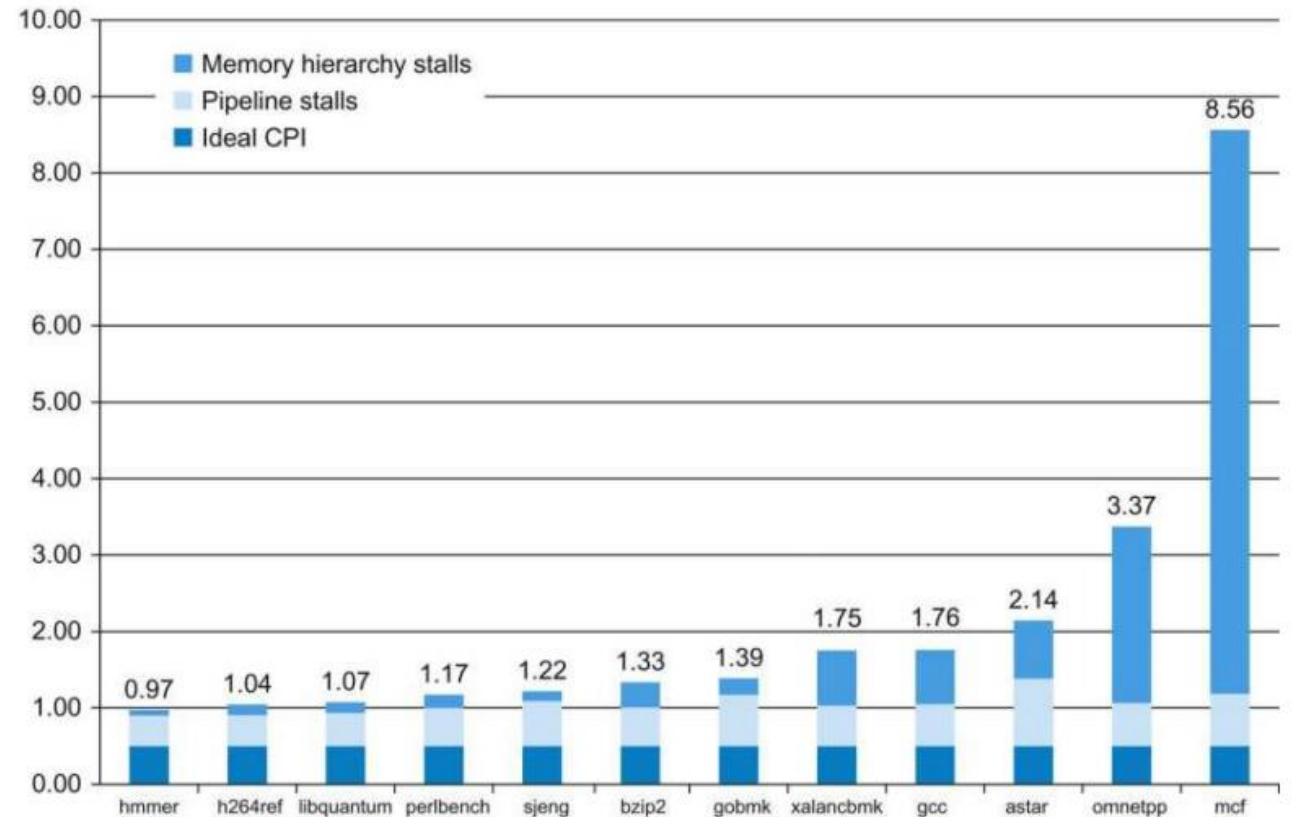


How do we reconcile?

Real-world examples: Intel i7 and ARM Cortex-A53



CPI of Intel i7 920 on SPEC2006 Benchmarks



CPI of ARM Cortex-A53 on SPEC2006 Benchmarks

(High CPI is not ARM-inherent. Newer A78 has ideal 6 CPI)