

CS152: Computer Systems Architecture

The Hardware/Software Interface



Sang-Woo Jun

Winter 2021

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
 - Computer Arithmetic
 - Simple and pipelined processors
 - Caches and the memory hierarchy
- ❑ Part 4: Computer Systems
 - Operating systems, Virtual memory

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



COMMON CASE FAST



PARALLELISM



PIPELINING



PREDICTION



HIERARCHY



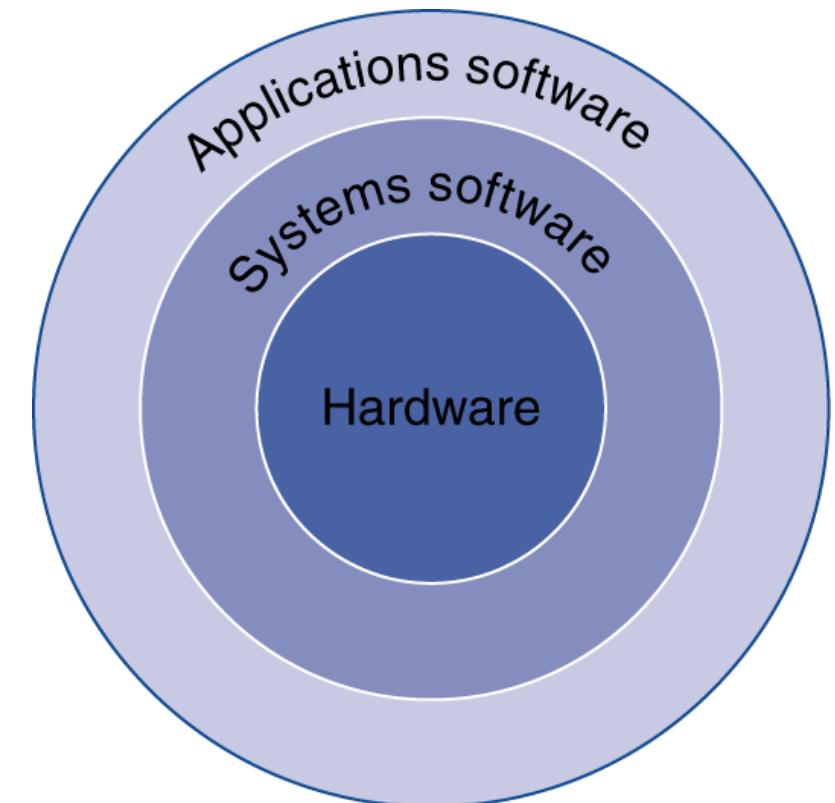
DEPENDABILITY

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

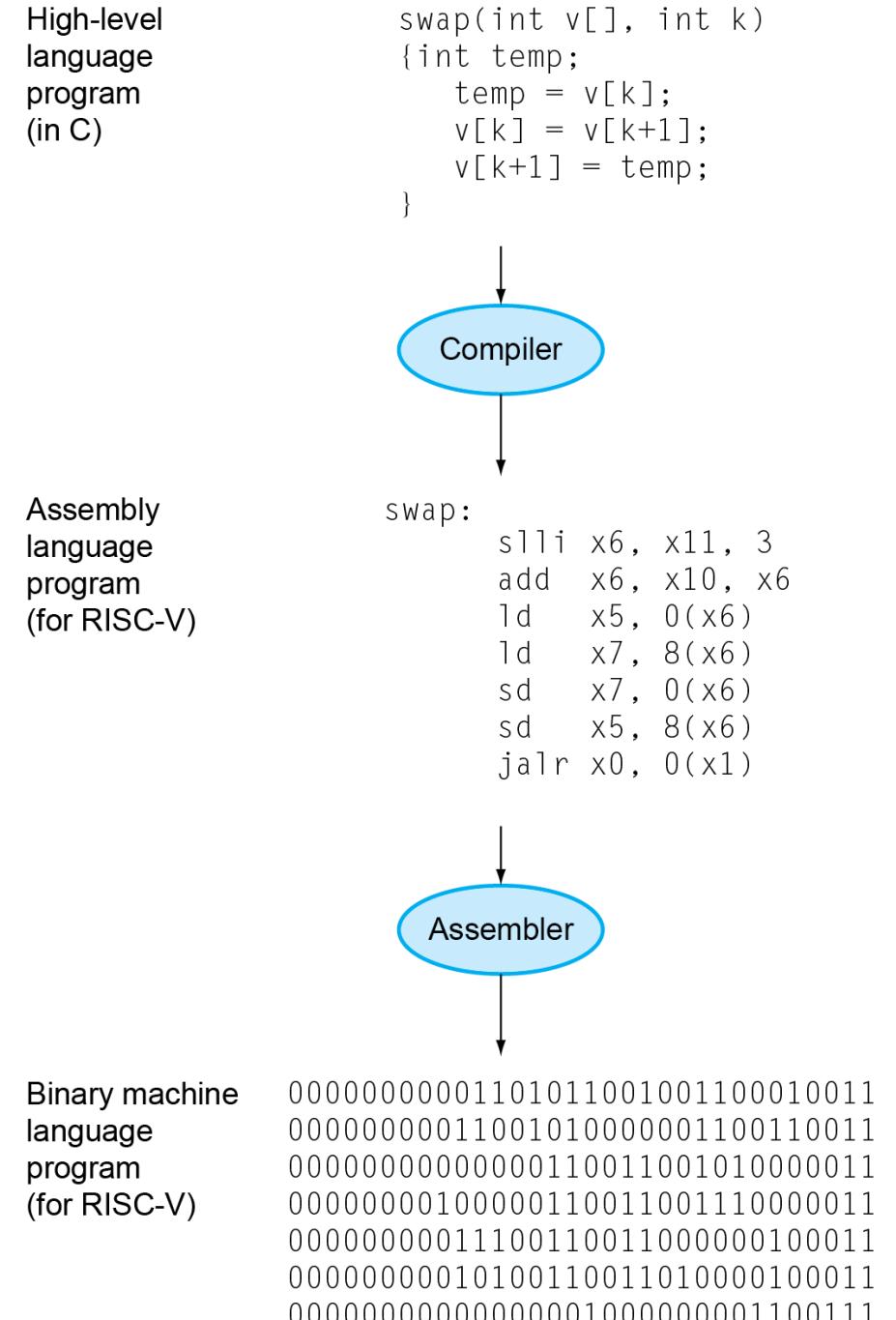
- An Instruction-Set Architecture (ISA) is the abstraction between the software and processor hardware
 - The ‘Hardware/Software Interface’
 - Different from ‘Microarchitecture’, which is how the ISA is implemented

- 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



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

add x18,x19,x10					
31	25 24	20 19	15 14	12 11	7 6
funct7	rs2	rs1	funct3	rd	opcode
7	5	5	3	5	7
0000000	01010	10011	000	10010	0110011

ADD rs2=10 rs1=19 ADD rd=18 Reg-Reg OP

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



CS152: Computer Systems Architecture

What Makes a “Good” ISA?



Sang-Woo Jun

Winter 2021

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
- ❑ Let's define performance = 1/execution time
- ❑ Example: time taken to run a program
 - 10s on A, 15s on B
 - Execution Time_B / Execution Time_A
 $= 15s / 10s = 1.5$
 - So A is 1.5 times faster than B

$$\frac{\text{Performance}_x}{\text{Performance}_y} = \frac{\text{Execution time}_y}{\text{Execution time}_x} = n$$

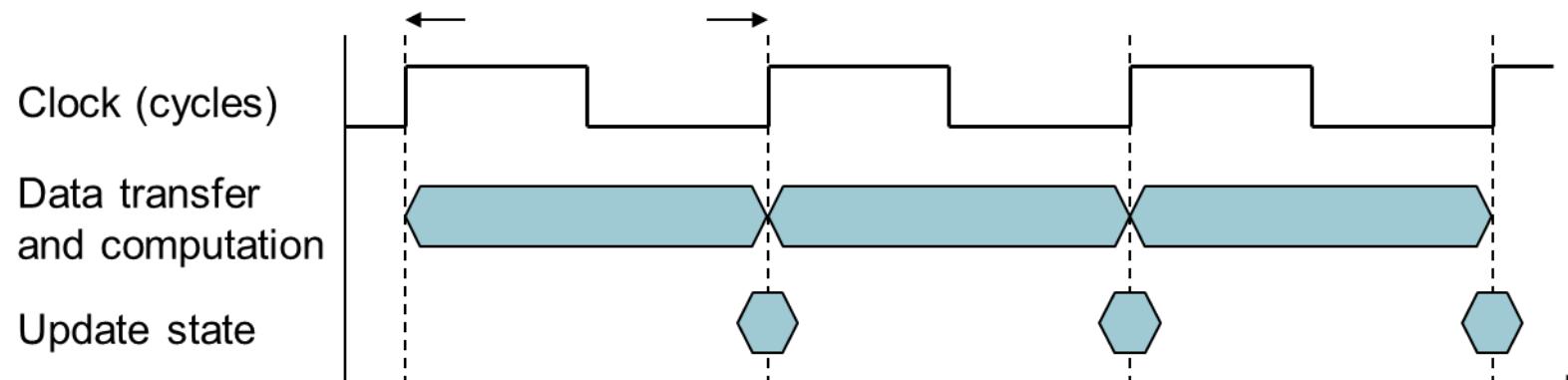
Measuring execution time

- ❑ 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 processing a given job
 - Discounts I/O time, other jobs' shares
 - Comprises user CPU time and system 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

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

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

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 \times Cycles per Instruction

CPU Time = Instruction Count \times CPI \times 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

$$\begin{aligned}\text{CPU Time}_A &= \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A \\ &= I \times 2.0 \times 250\text{ps} = I \times 500\text{ps} \quad \text{A is faster...}\end{aligned}$$

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

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{I \times 600\text{ps}}{I \times 500\text{ps}} = 1.2 \quad \text{...by this much}$$

CPI in more detail

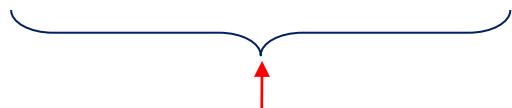
- If different instruction classes take different numbers of cycles

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

*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

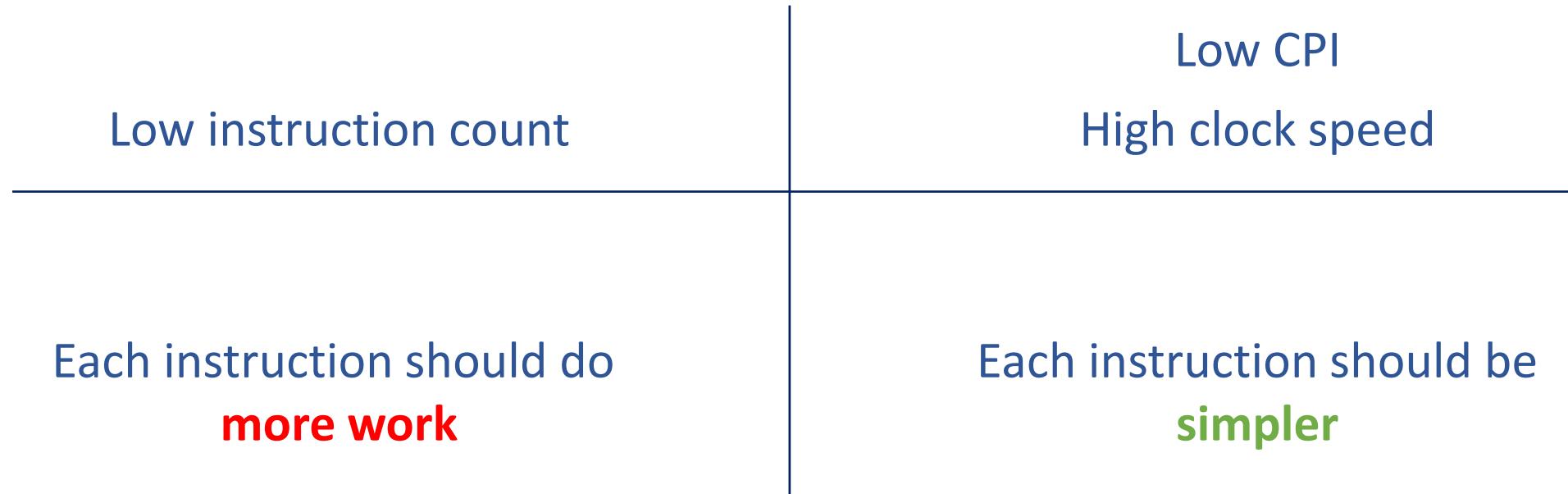
Performance summary

- ❑ 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

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

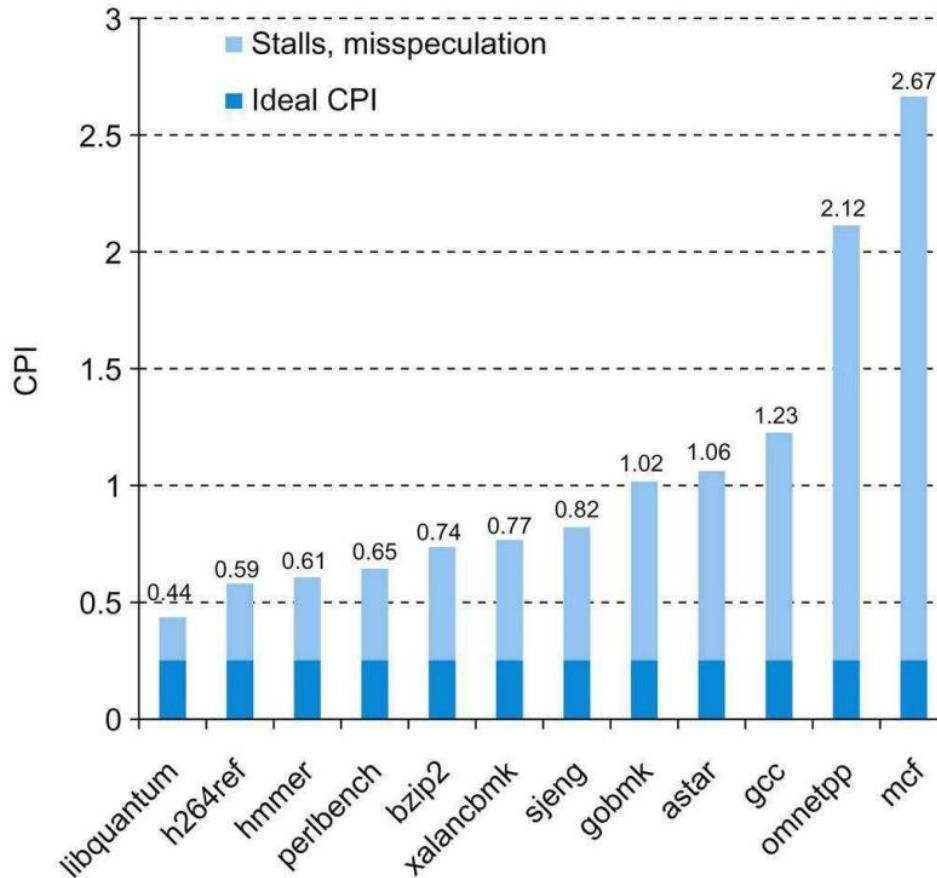
A good ISA: Low instruction count, Low CPI, High clock speed

Some goals for a good ISA

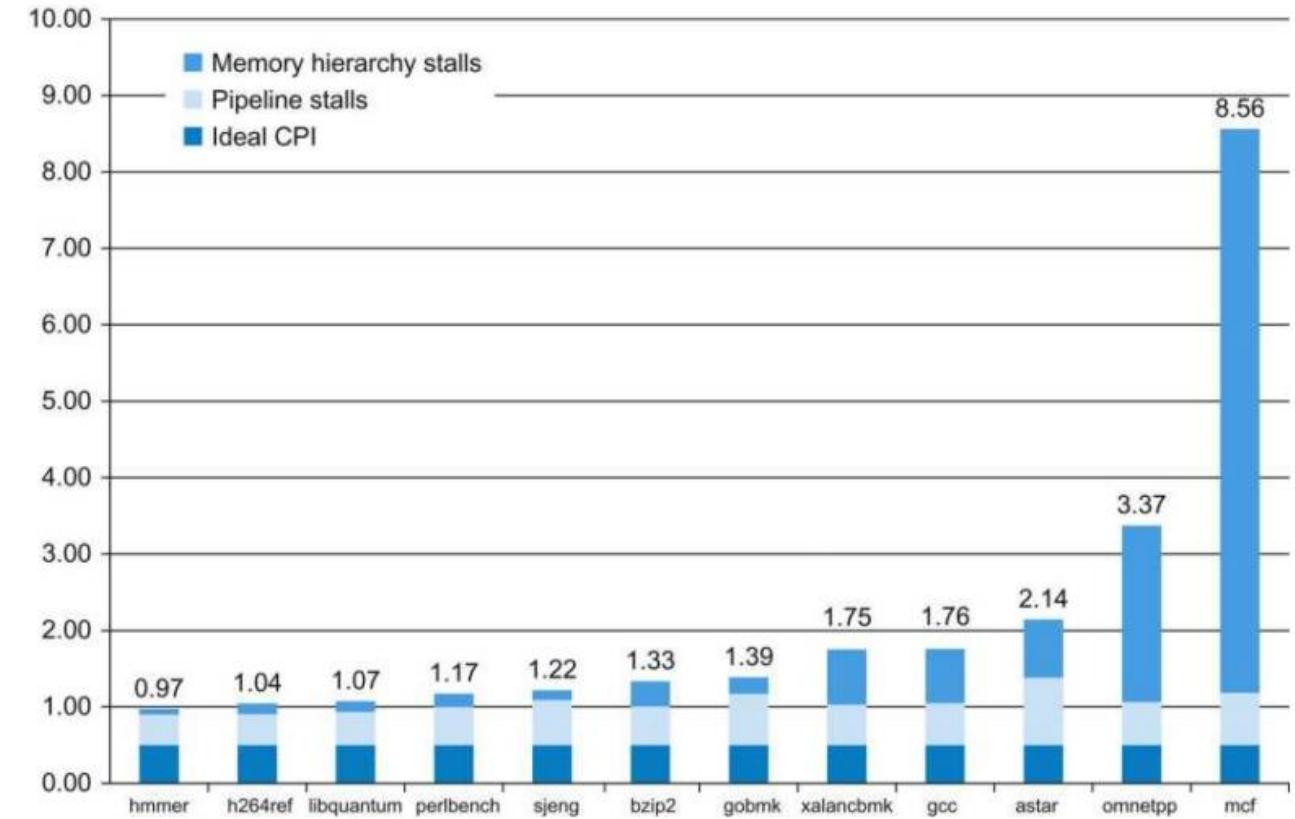


How do we reconcile?

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

CS152: Computer Systems Architecture

Some ISA Classifications



Sang-Woo Jun

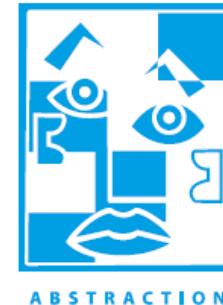
Winter 2021

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

Eight great ideas

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

today



COMMON CASE FAST



PARALLELISM



PIPELINING



PREDICTION



HIERARCHY



DEPENDABILITY

The RISC/CISC Classification

- ❑ Reduced Instruction-Set Computer (RISC)
 - Precise definition is debated
 - Small number of more general instructions
 - RISC-V base instruction set has only dozens of instructions
 - **Memory load/stores not mixed with computation operations**
(Different instructions for load from memory, perform computation in register)
 - Often fixed-width encoding (4 bytes for base RISC-V)
 - Complex operations implemented by composing general ones
 - Compilers try their best!
 - RISC-V, ARM (Advanced RISC Machines),
MIPS (Microprocessor without Interlocked Pipelined Stages),
SPARC, ...

The RISC/CISC Classification

- ❑ Complex Instruction-Set Computer (CISC)
 - Precise definition is debated (Not RISC?)
 - Many, complex instructions
 - Various memory access modes per instruction (load from memory? register? etc)
 - Typically variable-length encoding per instruction
 - Modern x86 has thousands!
 - Intel x86,
IBM z/Architecture,
 - ...

The RISC/CISC Classification

- ❑ RISC paradigm is winning out
 - Simpler design allows faster clock
 - Simpler design allows efficient microarchitectural techniques
 - Superscalar, Out-of-order, ...
 - Compilers very good at optimizing software
- ❑ Most modern CISC processors have RISC internals
 - CISC instructions translated on-the-fly to RISC by the front-end hardware
 - Added overhead from translation (silicon, power, performance, ...)