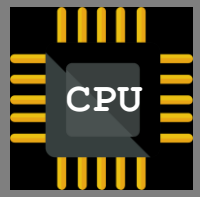


Topic 2

Quantitative Design and Analysis #1

Xinfei Guo
xinfei.guo@sjtu.edu.cn

May 11th, 2022

T2 learning goals

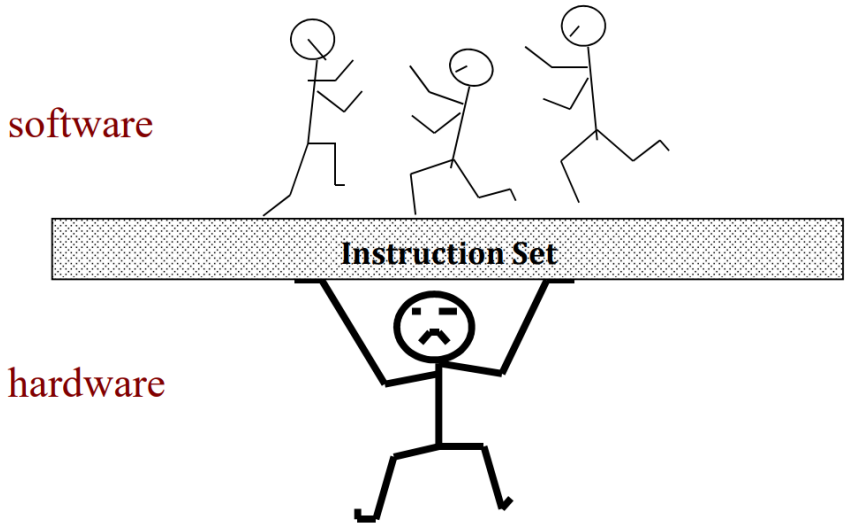
- **Quantitative** Design and Analysis
  - Section #1
    - ISAs
    - Performance
  - Section #2
    - Power and Energy
    - Reliability
    - Cost

Instruction Set

- or ISA, a “set of commands/instruction” that a computer understands
- Different computers have different instruction sets
  - But with many common aspects
- Examples:
  - Reduced Instruction Set Computer – RISC (Main stream)
  - Complex Instruction Set Computer – CISC

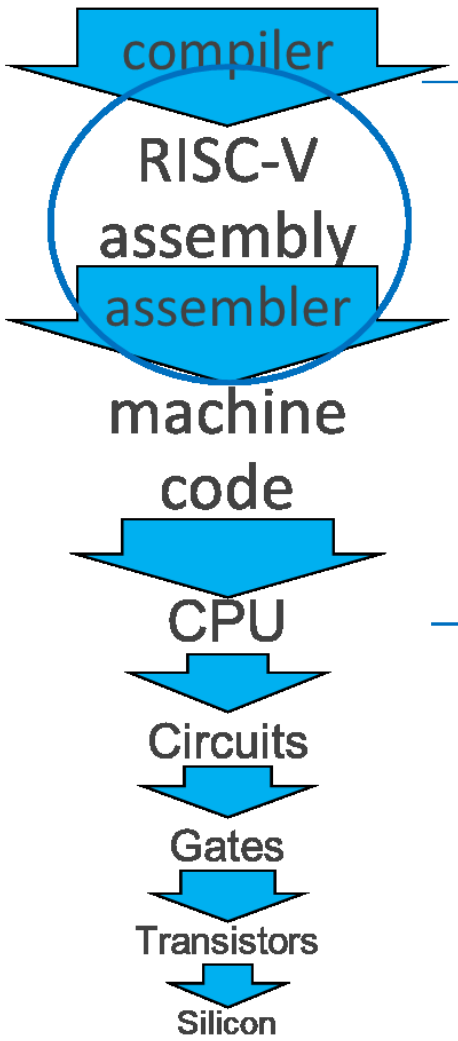
Instruction set architecture (ISA)

- Interface between hardware & low-level software
- Typically described by giving all the programmer-visible state (registers + memory) plus the semantics of the instructions that operate on that state
- Standardizes instructions and machine language bit patterns
- An abstraction omits unnecessary details
- Main Goals for ISA Design - Maximize performance, minimize cost and reduce design time



Source & Image: Dr. Gul N. Khan COE608

ISAs

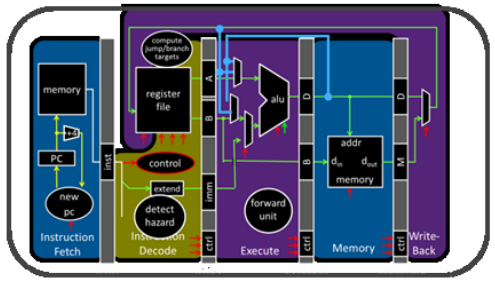



```
addi x5, x0, 10
muli x5, x5, 2
addi x5, x5, 15
```

```
00000000101000000000001010010011
00000000001000101000001010000000
00000000111100101000001010010011
```

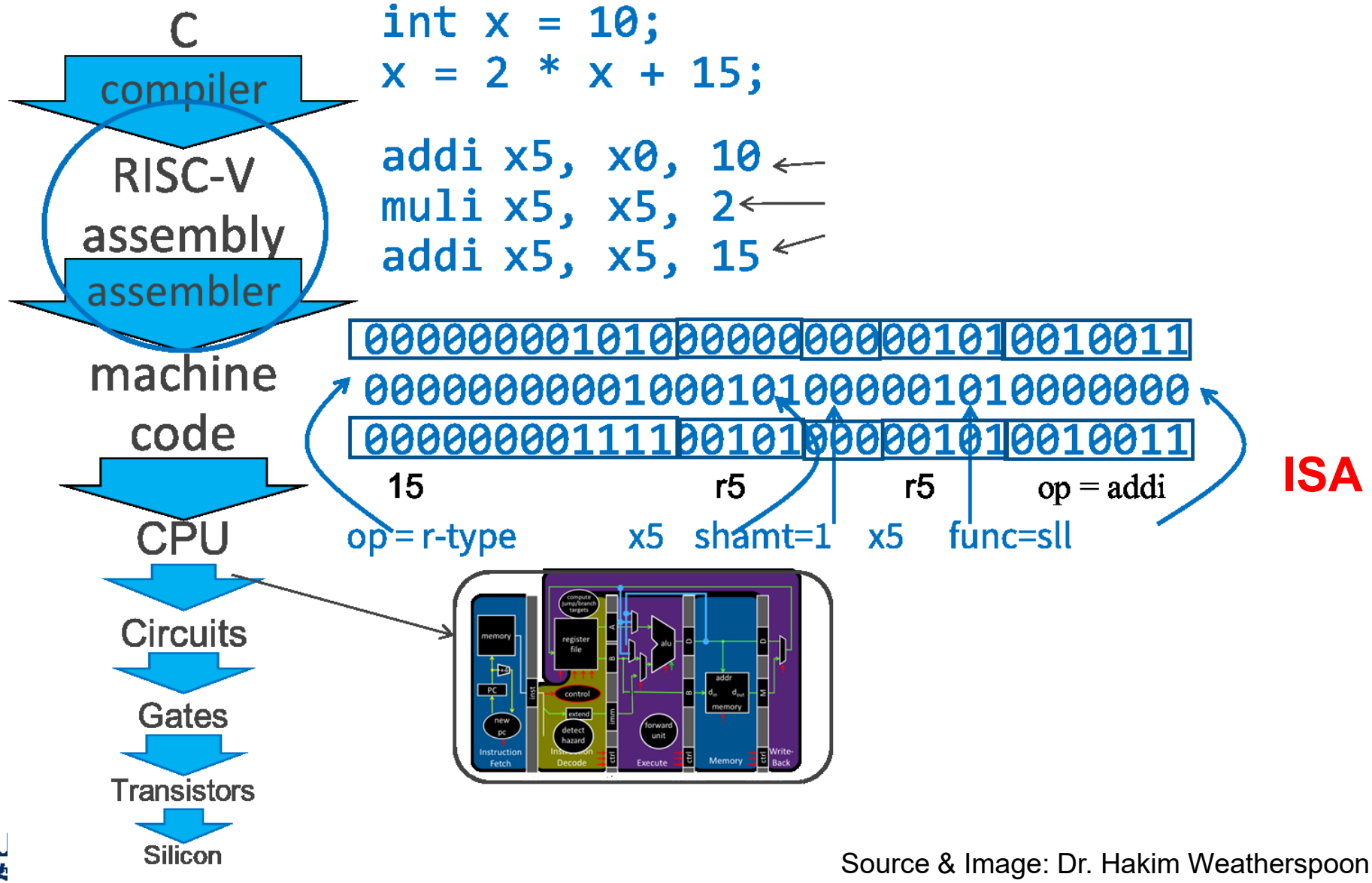
High Level Languages

Instruction Set Architecture (ISA)



Source & Image: Dr. Hakim Weatherspoon CS 3410

ISAs



Source & Image: Dr. Hakim Weatherspoon CS 3410

Major ISAs

ISA defines the permissible instructions

- RISC-V: load/store, arithmetic, control flow, …
- ARMv7: similar to RISC-V, but more shift, memory, & conditional ops
- ARMv8 (64-bit): even closer to RISC-V, no conditional ops
- VAX: arithmetic on memory or registers, strings, polynomial evaluation, stacks/queues, …
- Cray: vector operations, …
- x86: a little of everything

Why not CISC?

CISC	RISC
The original microprocessor ISA	Redesigned ISA that emerged in the early 1980s
Instructions can take several clock cycles	Single-cycle instructions
Hardware-centric design – the ISA does as much as possible using hardware circuitry	Software-centric design – High-level compilers take on most of the burden of coding many software steps from the programmer
More efficient use of RAM than RISC	Heavy use of RAM (can cause bottlenecks if RAM is limited)
Complex and variable length instructions	Simple, standardized instructions
May support microcode (micro-programming where instructions are treated like small programs)	Only one layer of instructions
Large number of instructions	Small number of fixed-length instructions
Compound addressing modes	Limited addressing modes

- Example: *Multiplying Two Numbers in Memory*

  one stored in location 2:3 and another stored in location 5:2 - and then store the product back in the location 2:3

  - CISC approach (x86)

    ```
    MULT 2:3, 5:2
    ```

    one instruction does all, compiler happy

  - RISC approach (RISC-V, ARM)

    ```
    LOAD A, 2:3
    LOAD B, 5:2
    PROD A, B
    STORE 2:3, A
    ```

    simple instructions, one cycle each, pipeline is possible, less logic, load and store can preserve values…

Table: https://www.seeedstudio.com/blog/2020/06/01/10-things-you-should-know-about-risc-v-in-2020-m/

RISC vs CISC

RISC Philosophy

- Regularity & simplicity
- Leaner means faster
- Optimize the common case

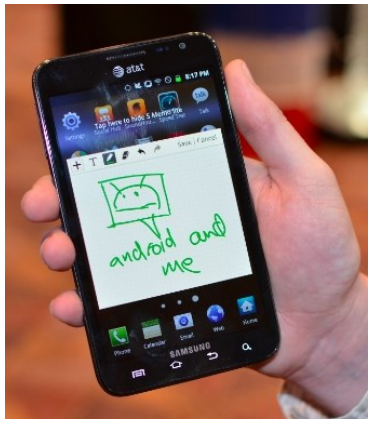
CISC Rebuttal

- Compilers can be smart
- Transistors are plentiful
- Legacy is important
- Code size counts
- Micro-code!

Energy efficiency

Embedded Systems

Phones/Tablets



Desktops/Servers

source: Prof. Hakim Weatherspoon, CS 3410

The RISC-V Instruction Set

- Used as the example throughout the book/lectures
- Fifth generation of RISC
- Developed at UC Berkeley as **open ISA**
- Now managed by the RISC-V Foundation (riscv.org)
- Typical of many modern ISAs
  - See RISC-V Reference Data tear-out card
- Similar ISAs have a large share of embedded core market
  - Applications in consumer electronics, network/storage equipment, cameras, printers, …
- Easy to extend!

Why RISC-V?

- “This is an exciting opportunity for the computing industry as well as for education, and thus at the time of this writing more than 40 companies have joined the RISC-V foundation. This sponsor list includes virtually all the major players except for ARM and Intel, including AMD, Google, Hewlett Packard Enterprise, IBM, Microsoft, NVIDIA, Oracle, and Qualcomm.”

  - Patterson and Hennessy

- Dominant ISAs (x86 and ARM) are too complex to use for teaching or research

ARM Users

source: https://www.arm.com/zh-TW/architecture/system-architectures/systemready-certification-program/partners

RISC-V users

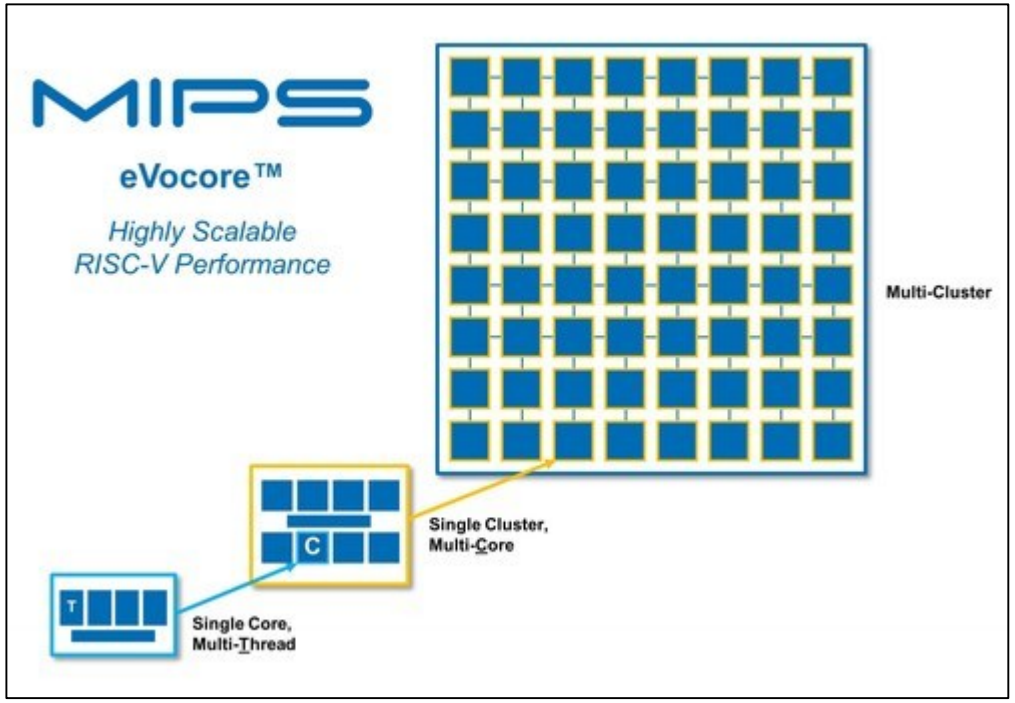
Image: https://circuitdigest.com/article/understanding-risc-v-architecture-and-why-it-could-be-a-replacement-for-arm

Even MIPS Pivots to RISC-V

- MIPS Tech switched to RISC-V
- Better Performance and Scalability

About MIPS

MIPS is a leading developer of highly scalable RISC processor IP for high-end automotive, computing and communications applications. With its deep engineering expertise built over 35 years and billions of MIPS-based chips shipped to-date, today the company is accelerating RISC-V innovation for a new era of heterogeneous processing. The company's proven solutions are uniquely configurable, enabling semiconductor companies to hit exacting performance and power requirements and differentiate their devices.
Visit: www.mips.com.



Released on May 10, 2022

MIPS TECHNOLOGIES

March 8, 2021

Wait, What? MIPS Becomes RISC-V

Classic CPU Company Exits Bankruptcy, Throws in the Towel

by Jim Turley

What a long, strange trip it's been. MIPS Technologies no longer designs MIPS processors. Instead, it's joined the RISC-V camp, abandoning its eponymous architecture for one that has strong historical and technical ties. The move apparently heralds the end of the road for MIPS as a CPU family, and a further (slight) diminution in the variety of processors available. It's the final arc of an architecture.

Sources:

- https://www.eejournal.com/article/wait-what-mips-becomes-risc-v/
- https://www.hpcwire.com/off-the-wire/mips-pivots-to-risc-v-with-performance-and-scalability/

RISC-V Encoding Cheat Sheet

Format	Instruction	Opcode	Funct3	Funct6/7
R-type	add	0110011	000	0000000
	sub	0110011	000	0100000
	sll	0110011	001	0000000
	xor	0110011	100	0000000
	srl	0110011	101	0000000
	sra	0110011	101	0000000
	or	0110011	110	0000000
	and	0110011	111	0000000
	lr.d	0110011	011	0001000
	sc.d	0110011	011	0001100
I-type	lb	0000011	000	n.a.
	lh	0000011	001	n.a.
	lw	0000011	010	n.a.
	ld	0000011	011	n.a.
	lbu	0000011	100	n.a.
	lhu	0000011	101	n.a.
	lwu	0000011	110	n.a.
	addi	0010011	000	n.a.
	slli	0010011	001	000000
	xori	0010011	100	n.a.
	srli	0010011	101	000000
	srai	0010011	101	010000
	ori	0010011	110	n.a.
	andi	0010011	111	n.a.
	jalr	1100111	000	n.a.
S-type	sb	0100011	000	n.a.
	sh	0100011	001	n.a.
	sw	0100011	010	n.a.
	sd	0100011	111	n.a.
SB-type	beq	1100111	000	n.a.
	bne	1100111	001	n.a.
	blt	1100111	100	n.a.
	bge	1100111	101	n.a.
	bltu	1100111	110	n.a.
	bgeu	1100111	111	n.a.
U-type	lui	0110111	n.a.	n.a.
UJ-type	jal	1101111	n.a.	n.a.

RISC-V

- Memory addressing
  - RISC-V: byte addressed, aligned accesses faster
- Addressing modes
  - RISC-V: Register, immediate, displacement (base+offset)
  - Other examples: autoincrement, indexed, PC-relative
- Types and size of operands
  - RISC-V: 8-bit, 32-bit, 64-bit

RISC-V

- Operations
  - RISC-V: data transfer, arithmetic, logical, control, floating point
  - See Fig. 1.5 in text
- Control flow instructions
  - Use content of registers (RISC-V) vs. status bits (x86, ARMv7, ARMv8)
  - Return address in register (RISC-V, ARMv7, ARMv8) vs. on stack (x86)
- Encoding
  - Fixed (RISC-V, ARMv7/v8 except compact instruction set) vs. variable length (x86)

RISC-V ISA Variance

- RV32: the x registers are 32 bits wide (word)
- RV64: the x registers are 64 bits wide (doubleword)
- RV128: integer registers extended to 128 bits

RISC-V Extensions

Name	Description	Version	Status[a]
Base			
RVWMO	Weak Memory Ordering	2.0	Ratified
RV32I	Base Integer Instruction Set, 32-bit	2.1	Ratified
RV32E	Base Integer Instruction Set (embedded), 32-bit, 16 registers	1.9	Open
RV64I	Base Integer Instruction Set, 64-bit	2.1	Ratified
RV128I	Base Integer Instruction Set, 128-bit	1.7	Open
Extension			
M	Standard Extension for Integer Multiplication and Division	2.0	Ratified
A	Standard Extension for Atomic Instructions	2.1	Ratified
F	Standard Extension for Single-Precision Floating-Point	2.2	Ratified
D	Standard Extension for Double-Precision Floating-Point	2.2	Ratified
G	Shorthand for the base integer set (I) and above extensions (MAFD)	N/A	N/A
Q	Standard Extension for Quad-Precision Floating-Point	2.2	Ratified
L	Standard Extension for Decimal Floating-Point	0.0	Open
C	Standard Extension for Compressed Instructions	2.0	Ratified
B	Standard Extension for Bit Manipulation	0.92	Open
J	Standard Extension for Dynamically Translated Languages	0.0	Open
T	Standard Extension for Transactional Memory	0.0	Open
P	Standard Extension for Packed-SIMD Instructions	0.2	Open
V	Standard Extension for Vector Operations	0.9	Open
N	Standard Extension for User-Level Interrupts	1.1	Open
H	Standard Extension for Hypervisor	0.4	Open
ZiCSR	Control and Status Register (CSR)	2.0	Ratified
Zifencei	Instruction-Fetch Fence	2.0	Ratified
Zam	Misaligned Atomics	0.1	Open
Ztso	Total Store Ordering	0.1	Frozen

Table: Wikipedia

Example: specialized custom instruction

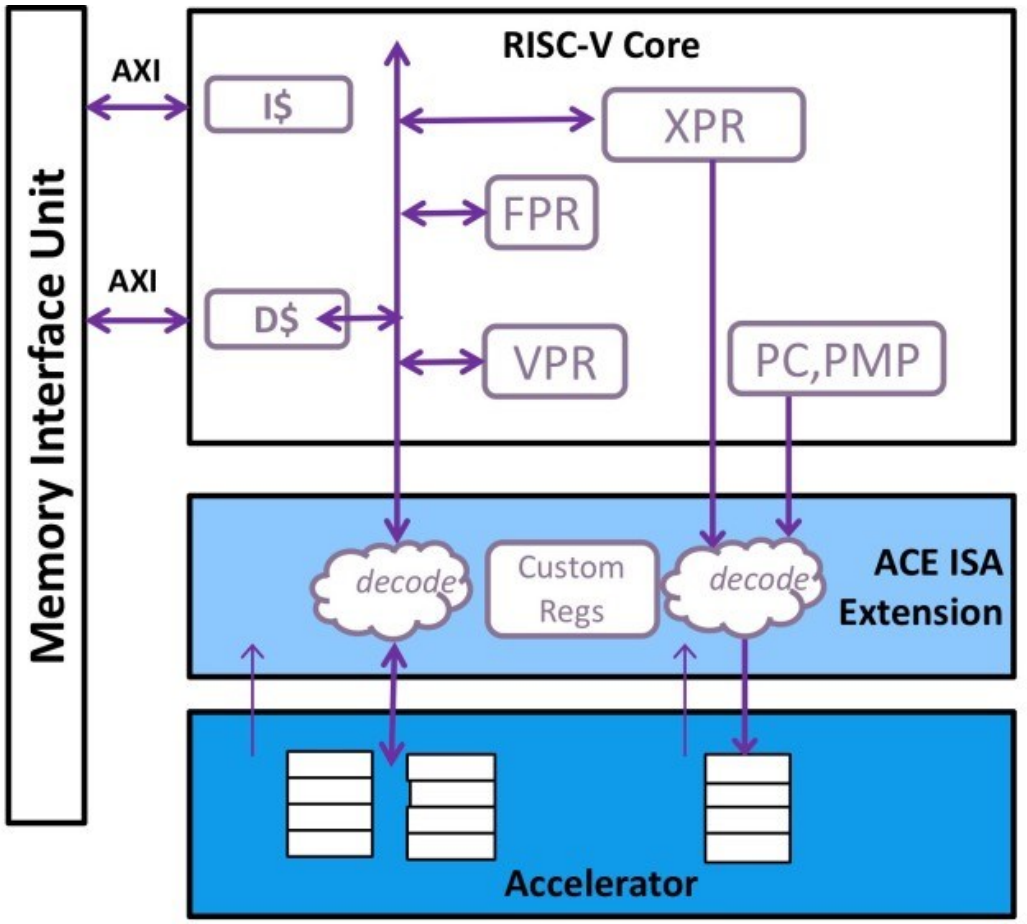


Image: Andes Technology

ISA Takeaways

- The number of available registers greatly influenced the instruction set architecture (ISA)
- Complex Instruction Set Computers were very complex
  - + Small # of insns necessary to fit program into memory.
  - - greatly increased the complexity of the ISA as well.
- RISC ISA's today such as ARM have an ever increasing market share (of our everyday life!).
- RISC-V is getting popular due to its open nature

PROCESSOR PERFORMANCE

CPU performance

- Definitions
  - Latency (execution time): Time to finish a fixed task
  - Throughput (bandwidth): number of tasks in fixed time
  - Throughput can exploit parallelism, latency can't
  - Need to pick the right metric to meet different goals
- Speedup of X relative to Y
  - Execution time_Y / Execution time_X
- Execution time
  - Wall clock time: includes all system overheads
  - CPU time: only computation time
- Benchmarks
  - Kernels (e.g. matrix multiply)
  - Toy programs (e.g. sorting)
  - Synthetic benchmarks (e.g. Dhrystone)
  - Benchmark suites (e.g. SPEC06fp, TPC-C)

Example

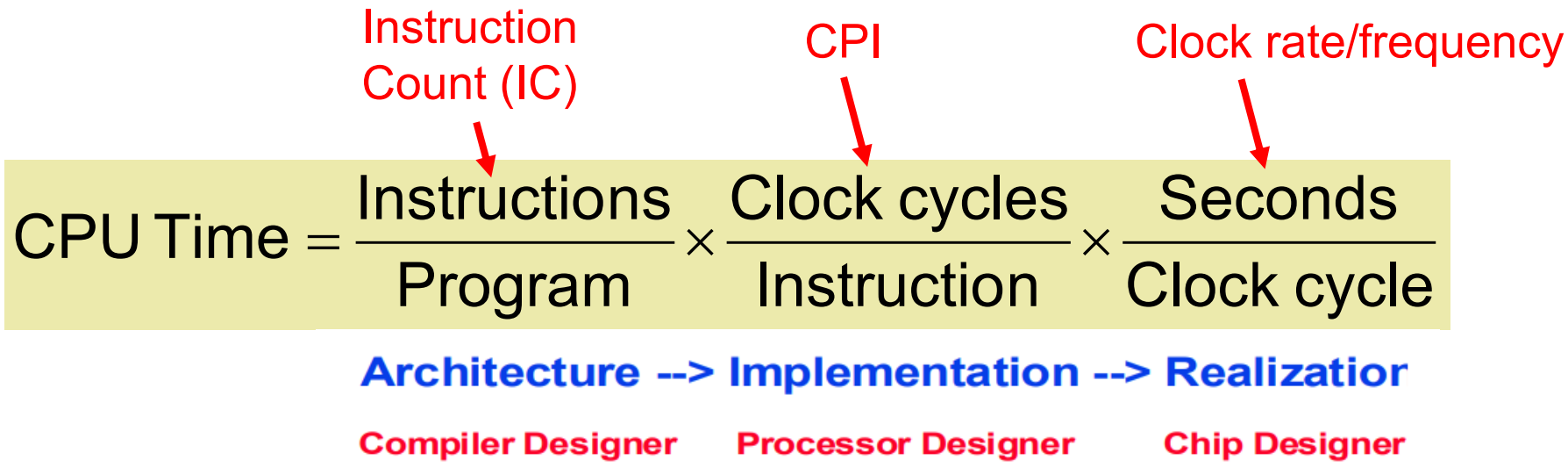
- Move people from A to B
  - Car: capacity = 5, speed = 60 miles/hour
  - Bus: capacity = 60, speed = 20 miles/hour
  - Latency: car = 10 min, bus = 30 min
  - Throughput: car = 15 PPH (count return trip), bus = 60 PPH
- Latency? Car is 3 Times (and 200%) faster than bus
- Throughput? Bus is 4 Tmes (and 300%) faster than car

CPU performance

- You can add latencies, but not throughput
  - Latency(P1+P2, A) = Latency(P1,A) + Latency(P2,A)
  - Throughput(P1+P2,A) != Throughput(P1,A) + Throughput(P2,A)

- Sequential vs. Parallel

The "Iron Law" of Processor Performance



- Performance depends on
  - Algorithm: affects IC, possibly CPI
  - Programming language: affects IC, CPI
  - Compiler: affects IC, CPI
  - Instruction set architecture: affects IC, CPI, T_c

Average CPI

- If different instruction classes take different numbers of cycles

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

- Weighted average CPI

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

More about CPI

- Early machines were limited by transistor count. As a result, they often required multiple clock cycles to execute each instruction (CPI >> 1).
- As transistor budgets improved, we could aim to get closer to a CPI of 1.
- This is easy if we don’t care at all about clock frequency.
- Designing a high-frequency design with a good CPI is much harder. We need to keep our high-performance processor busy and avoid it stalling, which would increase our CPI. This requires many different techniques and costs transistors (area) and power.

More about CPI

- Eventually, the industry was also able to fetch and execute multiple instructions per clock cycle. This reduced CPI to below 1.
- When we fetch and execute multiple instructions together, we often refer to **Instructions Per Cycle (IPC),** which is 1/CPI.
- For instructions to be executed at the same time, they must be independent.
- Growing transistor budgets were exploited to help find and exploit this **Instruction-Level Parallelism (ILP).**

Danger: Partial Performance Metrics

- Micro-architects often ignore dynamic instruction count
  - Typically work in one ISA/one compiler → treat it as fixed
- Micro-architects often ignore instructions/program
- General public (mostly) also ignores CPI
  - Equates clock frequency with performance!!
- Example: which processor would you buy?
  - A: CPI=2, f=500MHz, B: CPI=1, f=300MHz (assume the same ISA/compiler)
  - A? But B is faster!

Source: Prof. Ron Dreslinski @ UMich, EECS 470

MIPS as a Performance Metric

- **MIPS**: Millions of Instructions Per Second (recall IPC)

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$
$$= \frac{\text{Instruction count}}{\frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$

  - Doesn't account for
    - Differences in ISAs between computers
    - Differences in complexity between instructions
  - Example:
    - Compiler removes instructions, program faster
    - However, "MIPS" goes down (misleading)

Computer Technology

- Performance improvements:
  - Improvements in semiconductor technology
    - Feature size, clock speed
  - Improvements in computer architectures
    - Enabled by HLL compilers, UNIX
    - Lead to RISC architectures
  - Together have enabled:
    - Lightweight computers
    - Productivity-based managed/interpreted programming languages

Historical Performance Gains

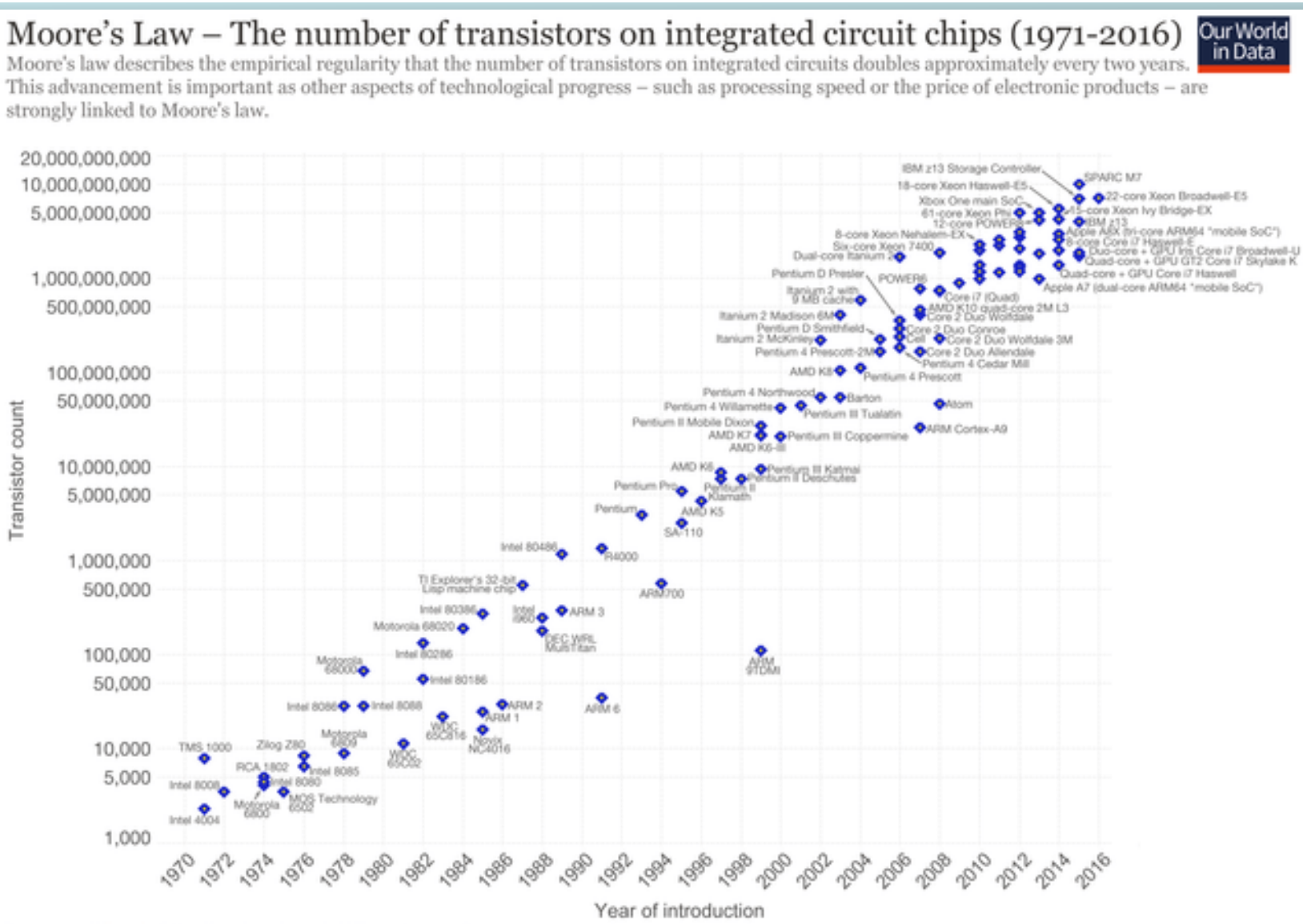
- By 1985, it was possible to integrate a complete microprocessor onto a single die or “chip.”
- As fabrication technology improved, and transistors got smaller, the performance of a single core improved quickly.
- Performance improved at the rate of 52% per year for nearly 20 years (measured using SPEC benchmark data).
- Note: the data are for desktop/server processors

source: ARM

Trends in Technology

- Integrated circuit technology (Moore's Law)
  - Transistor density: 35%/year
  - Die size: 10-20%/year
  - Integration overall: 40-55%/year
- DRAM capacity: 25-40%/year (slowing)
  - 8 Gb (2014), 16 Gb (2019), possibly no 32 Gb
- Flash capacity: 50-60%/year
  - 8-10X cheaper/bit than DRAM
- Magnetic disk capacity: recently slowed to 5%/year
  - Density increases may no longer be possible, maybe increase from 7 to 9 platters
  - 8-10X cheaper/bit then Flash
  - 200-300X cheaper/bit than DRAM

Transistor count increases significantly



Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

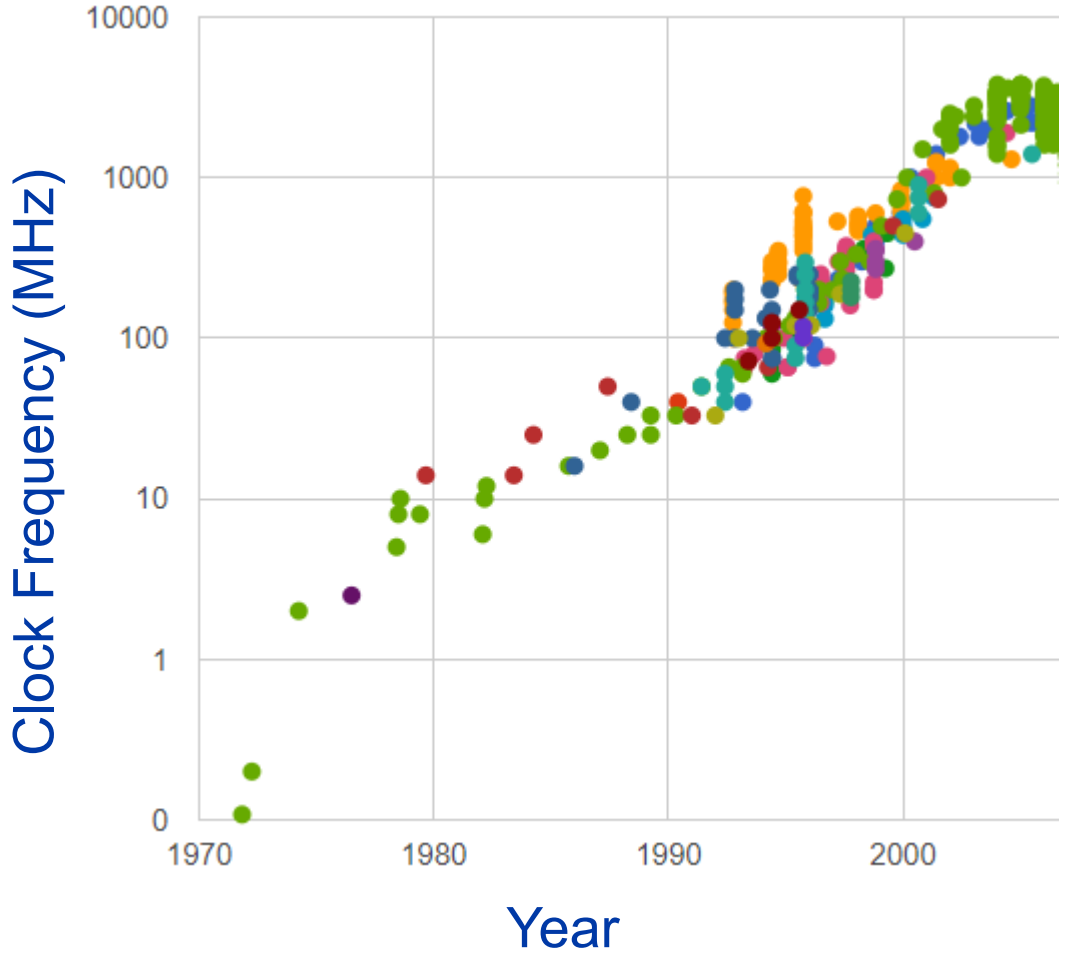
Clock frequency improved!

Clock period

- Clock frequency improved quickly between 1985 and 2002:
  - ~10x from faster transistors, and
  - ~10x from pipelining and circuit-level advances.
- So overall, **~100X** of the total 800X gains came from reduced clock periods.

IPC

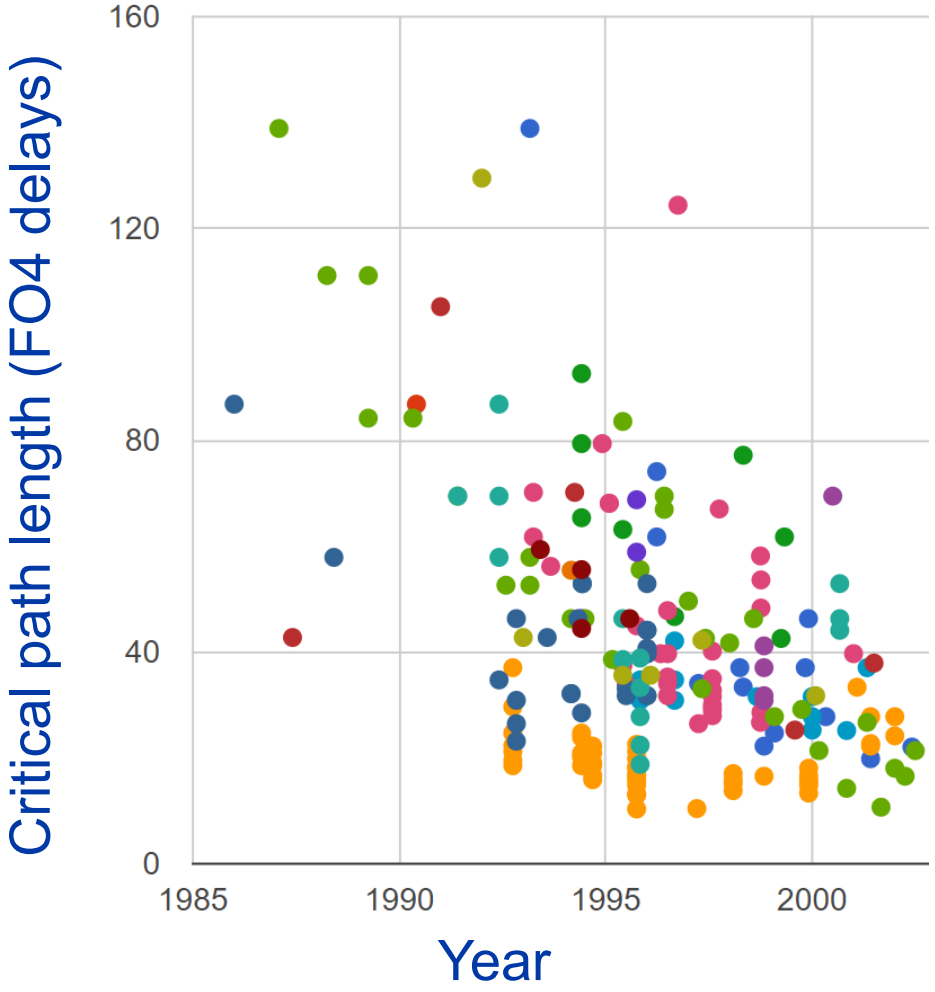
- The remaining gains (~8x) were from a reduction in instruction count, better compiler optimizations, and improvements in IPC.



A. Danowitz, K. Kelley, J. Mao, J. P. Stevenson, and M. Horowitz. *Clock Frequency*, Stanford CPU DB. Accessed on Nov. 5, 2019. [Online]. Available: http://cpudb.stanford.edu/visualize/clock_frequency

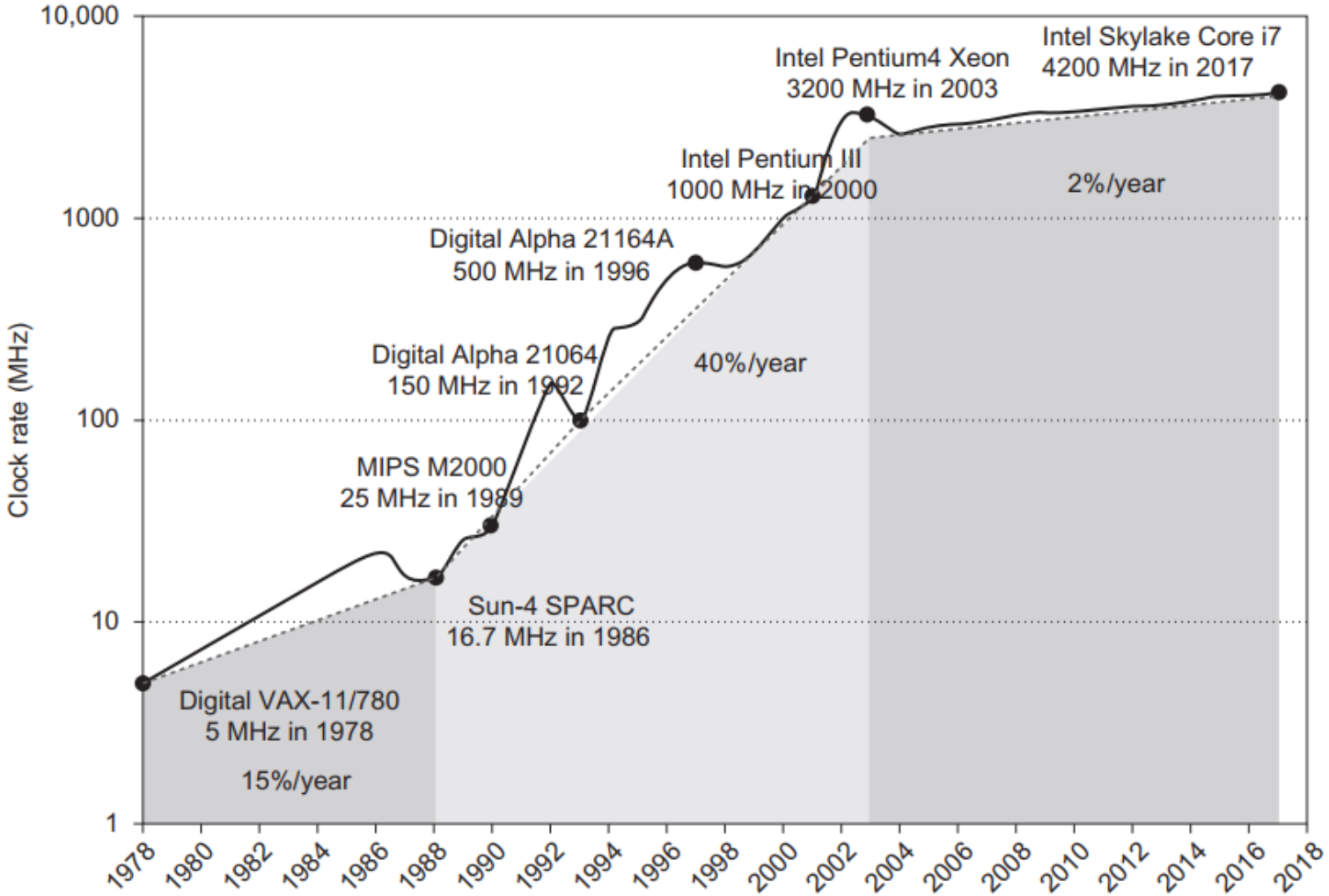
A Shorter Critical Path

- We can also try to reduce the number of gates on our critical path.
- This can be done by inserting additional registers to break complex logic into different “pipeline” stages.
- Advances were also made that improved circuit-level design techniques.
- The length of our critical paths reduced by **~10x** (1985-2002).



A. Danowitz, K. Kelley, J. Mao, J. P. Stevenson, and M. Horowitz. Stanford CPU DB. Accessed on Nov. 5, 2019. [Online]. Available: http://cpudb.stanford.edu

What happened after 2003?



Heat a wall due to power issues (will be covered soon)…

What is next then?

- Cannot continue to leverage Instruction-Level parallelism (ILP)
  - Single processor performance improvement ended in 2003
- New models for performance:
  - Data-level parallelism (DLP)
  - Thread-level parallelism (TLP)
  - Request-level parallelism (RLP)
- These require explicit restructuring of the application

Quantitative Principles of Computer Design

- Take Advantage of Parallelism
  - e.g. multiple processors, disks, memory banks, pipelining, multiple functional units
- Principle of Locality
  - Reuse of data and instructions
- Speculation
  - Guess you are right/wrong all the time
- Memorization
  - Programs do the same thing over and over again.
- Focus on the Common Case
  - … but speedup ultimately limited by the uncommon case
  - Amdahl's Law

$$T_{improved} = \frac{T_{affected}}{\text{improvement factor}} + T_{unaffected}$$

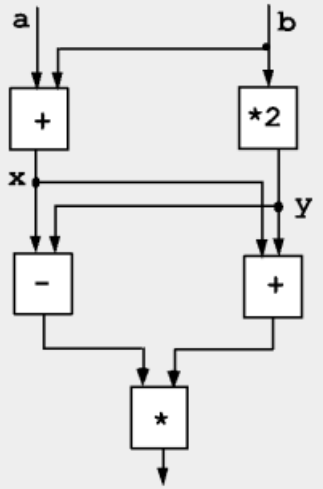
Parallelism

- Parallelism - the amount of **independent** sub-tasks available

- For a p wide system, average parallelism p_avg is

  p_avg ~ T_sequential/p

```
x = a + b;
y = b * 2
z =(x-y) * (x+y)
```



Locality

- Assumption: Past is always a good proxy of future
  - Temporal locality (you look up something before, you will look it up again)
  - Spatial Locality (If you look up something somewhere, you will look up something else nearby)
- If you haven't done something for a very long time, it is very likely you won't do it in the near future either.
- Overhead?
  - What if it is completely wrong…

Locality Example:

- **Data**
  - Reference array elements in succession (stride-1 reference pattern): **Spatial locality**
  - Reference `sum` each iteration: **Temporal locality**
- **Instructions**
  - Reference instructions in sequence: **Spatial locality**
  - Cycle through loop repeatedly: **Temporal locality**

```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

https://slideplayer.com/slide/5950187/

Speculation

- Do something before you know it is needed.
- Make “educated” guesses to avoid expensive operations if you can be right most of the time…
- Examples
  - Single processor: branch prediction, data value prediction, prefetching
  - Multiprocessor: thread-level parallelism, transactional memory, helper thread

Memorization

- If something is expensive to compute, you might want to remember the answer for a while, just in case you will need the same answer again.
- Examples
  - Branch prediction
  - Trace cache

    storing traces of instructions that have already been fetched and decoded

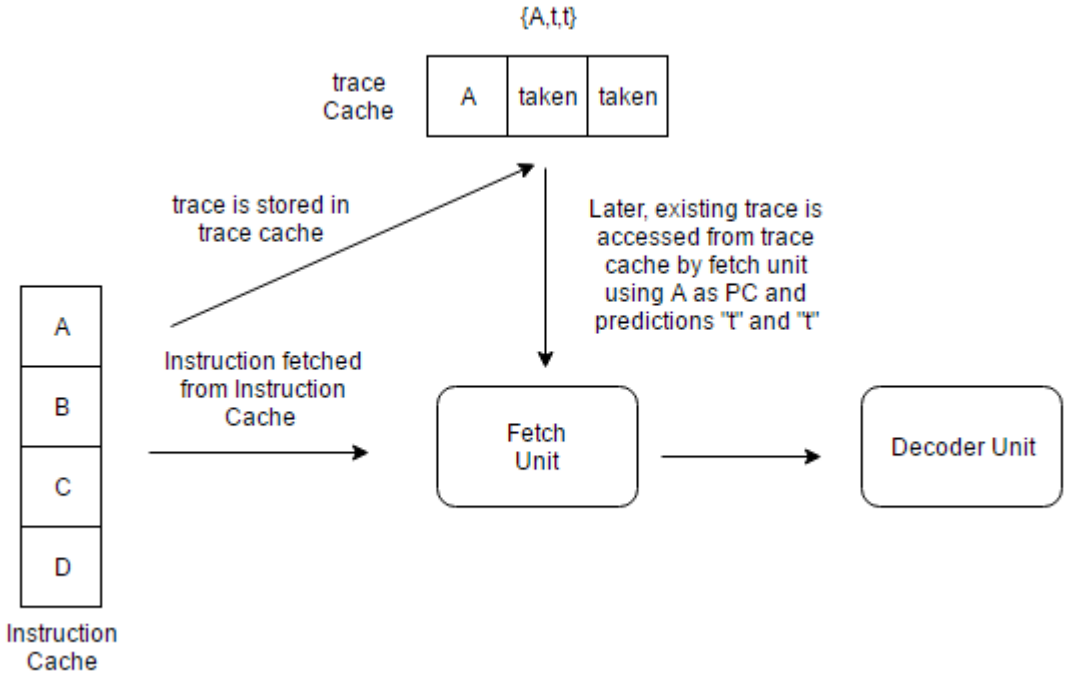


Figure: Wikipedia

Make the common case fast!

- Simpler and can be done faster
- The instruction fetch and decode unit of a processor may be used much more frequently than a multiplier, so optimize it first.
- Need to decide what the frequent case is and how much performance can be improved by making that case faster.

Amdahl's Law

$$\text{Speedup}_{\text{overall}} = \frac{\text{Execution time}_{\text{old}}}{\text{Execution time}_{\text{new}}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

- Example: multiply accounts for 80s/100s
  - How much improvement in multiply performance to get 5× overall?

$$20 = \frac{80}{n} + 20$$

  - Can't be done!

Performance in summary

- CPU time Iron law

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

- Amdahl's law

$$T_{\text{improved}} = \frac{T_{\text{affected}}}{\text{improvement factor}} + T_{\text{unaffected}}$$

- Quantitative Principles of Computer Design

Where are we Heading?

- T2: Quantitative Design and Analysis
  - Section 2: Power and energy, Cost and Reliability

Acknowledgement

Slides in this topic are inspired in part by material developed and copyright by:

- ARM Courseware
- Prof. Ron Dreslinski @ UMich, EECS 470
- Prof. Krste Asanović @ UCB, CS152/252
- Xinfei Guo@JI, VE370 2021 SU

Action Items

- Join Feishu group
- Join Piazza group
- Finish the course survey!!
- Reading Materials
  - Ch. 1.1-1.4, 1.8, 1.9