CENG 5410
Advanced Computer Architecture
Lecture 01: Introduction

Zhengrong Wang
CSE Department, CUHK
zhengrongwang@cuhk.edu.hk

Welcome!

- About Me:
  - Prof. @CUHK since September 2025
  - Research focuses on architectures/accelerators/specialization/heterogeneous computers
  - Love teaching this course – strong ties to research area
- Teaching Style: non-authoritarian?
  - Much of this course contains architecture "wisdom"
  - Applications and Technology change very quickly -- everything is open for debate.
- Let's treat this like a discussion!

- Slides adapted from UW Madison, UPenn, UCLA, and various universities.

Warmup 1

- Which of these is faster?

Version 1

```
void copyij(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}
```

Version 2

```
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```


source: http://www.cs.cmu.edu/afs/cs/academic/class/15213-f15/www/lectures/01-overview.pdf

Warmup 2

- Which of these is faster?

Version 1

```
for (unsigned c = 0; c < n; ++c)
  data[c] = std::rand() % 256;

//std::sort(data, data + n);

// BEGIN TIMER
for (int c = 0; c < n; ++c)
  if (data[c] >= 128)
    sum += data[c]*2;
// END TIMER
```

Version 2

```
for (unsigned c = 0; c < n; ++c)
  data[c] = std::rand() % 256;

std::sort(data, data + n);

// BEGIN TIMER
for (int c = 0; c < n; ++c)
  if (data[c] >= 128)
    sum += data[c]*2;
// END TIMER
```


source: stackoverflow.com/questions/11227809

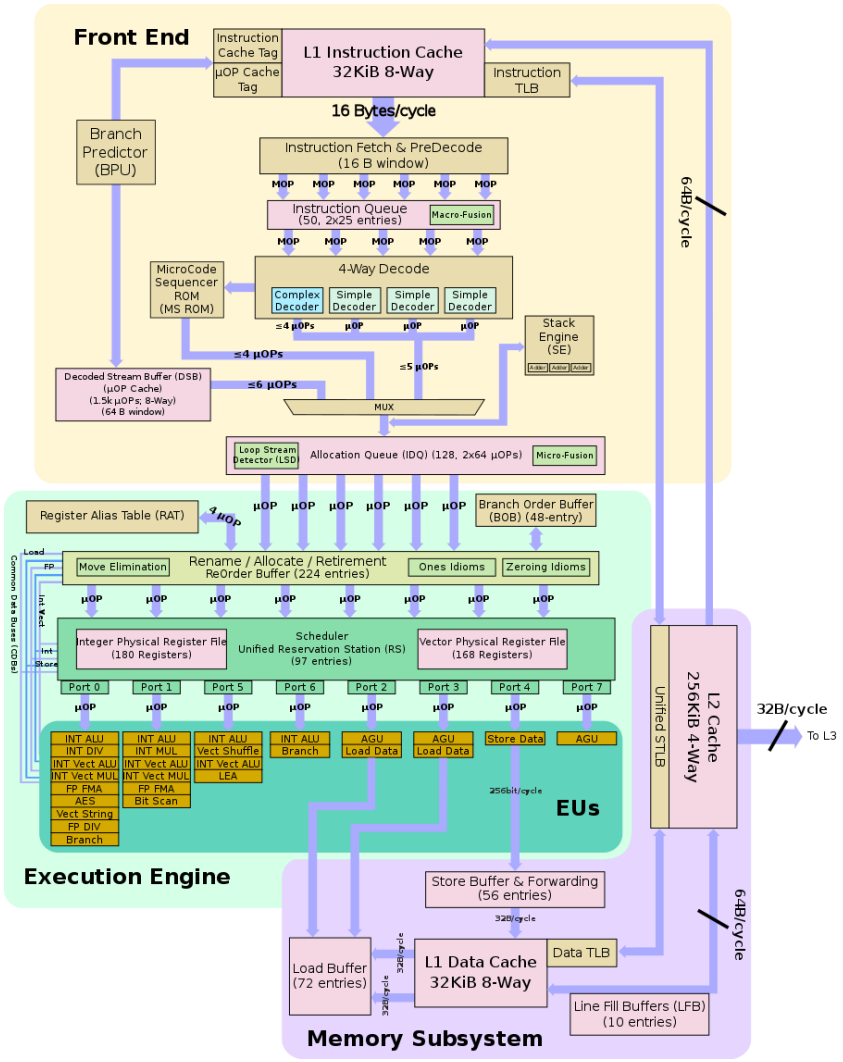
What is computer architecture?

- The intersection between software and hardware.
- Hardware: microarchitecture, performance, power, area.
- ISA: instruction set architecture.
- OS/Runtime/Application: software leveraging the ISA and hardware.

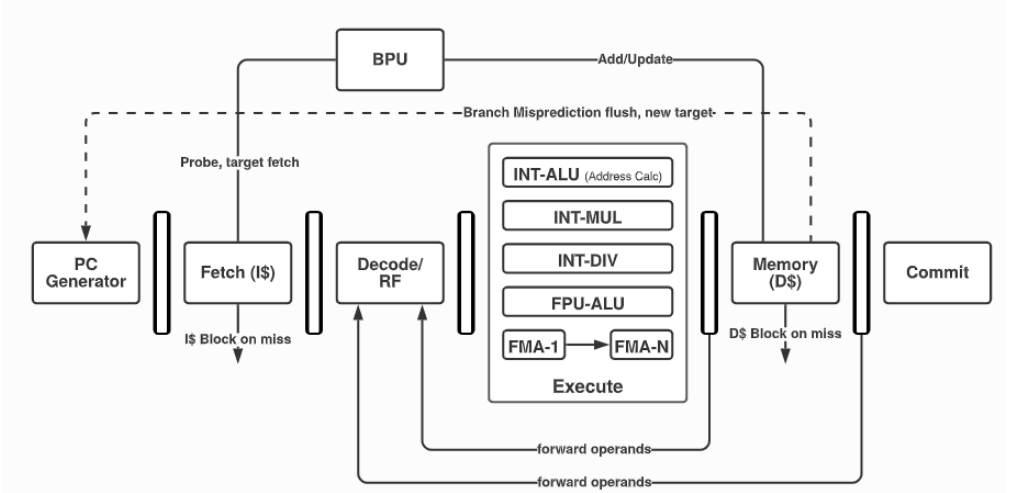
Example Architectures

Example Microarchitectures

Intel Skylake OOO



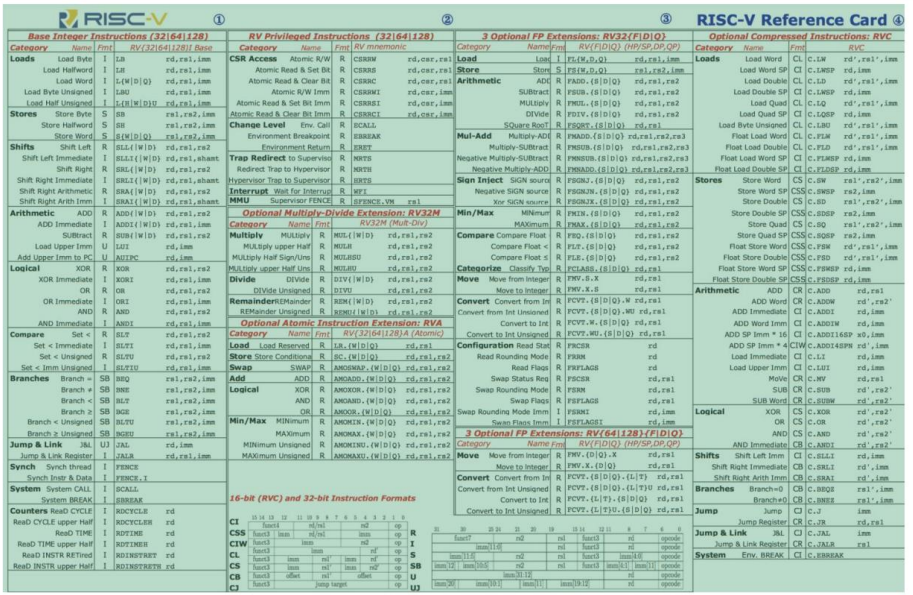
Inorder Core



Example Instruct Set Architectures

X86 Manual

Entire RISC-V Manual



RISC-V ① ② ③ **RISC-V Reference Card ④**

Base Integer Instructions (32|64|128)

Category	Name	Fmt	RV{32|64|128}I Base
Loads	Load Byte	I	LB rd,rs1,imm
	Load Halfword	I	LH rd,rs1,imm
	Load Word	I	L{W|D|Q} rd,rs1,imm
	Load Byte Unsigned	I	LBU rd,rs1,imm
	Load Half Unsigned	I	L{H|W|D}U rd,rs1,imm
Stores	Store Byte	S	SB rs1,rs2,imm
	Store Halfword	S	SH rs1,rs2,imm
	Store Word	S	S{W|D|Q} rs1,rs2,imm
Shifts	Shift Left	R	SLL{W|D} rd,rs1,rs2
	Shift Left Immediate	I	SLLI{W|D} rd,rs1,shamt
	Shift Right	R	SRL{W|D} rd,rs1,rs2
	Shift Right Immediate	I	SRLI{W|D} rd,rs1,shamt
	Shift Right Arithmetic	R	SRA{W|D} rd,rs1,rs2
	Shift Right Arith Imm	I	SRAI{W|D} rd,rs1,shamt
Arithmetic	ADD	R	ADD{W|D} rd,rs1,rs2
	ADD Immediate	I	ADDI{W|D} rd,rs1,imm
	SUBtract	R	SUB{W|D} rd,rs1,rs2
	Load Upper Imm	U	LUI rd,imm
	Add Upper Imm to PC	U	AUIPC rd,imm
Logical	XOR	R	XOR rd,rs1,rs2
	XOR Immediate	I	XORI rd,rs1,imm
	OR	R	OR rd,rs1,rs2
	OR Immediate	I	ORI rd,rs1,imm
	AND	R	AND rd,rs1,rs2
	AND Immediate	I	ANDI rd,rs1,imm
Compare	Set <	R	SLT rd,rs1,rs2
	Set < Immediate	I	SLTI rd,rs1,imm
	Set < Unsigned	R	SLTU rd,rs1,rs2
	Set < Imm Unsigned	I	SLTIU rd,rs1,imm
Branches	Branch =	SB	BEQ rs1,rs2,imm
	Branch ≠	SB	BNE rs1,rs2,imm
	Branch <	SB	BLT rs1,rs2,imm
	Branch ≥	SB	BGE rs1,rs2,imm
	Branch < Unsigned	SB	BLTU rs1,rs2,imm
	Branch ≥ Unsigned	SB	BGEU rs1,rs2,imm
Jump & Link	J&L	UJ	JAL rd,imm
	Jump & Link Register	I	JALR rd,rs1,imm
Synch	Synch thread	I	FENCE
	Synch Instr & Data	I	FENCE.I
System	System CALL	I	SCALL
	System BREAK	I	SBREAK
Counters	ReaD CYCLE	I	RDCYCLE rd
	ReaD CYCLE upper Half	I	RDCYCLEH rd
	ReaD TIME	I	RDTIME rd
	ReaD TIME upper Half	I	RDTIMEH rd
	ReaD INSTR RETired	I	RDINSTRET rd
	ReaD INSTR upper Half	I	RDINSTRETH rd

RV Privileged Instructions (32|64|128)

Category	Name	Fmt	RV mnemonic
CSR Access	Atomic R/W	R	CSRRW rd,csr,rs1
	Atomic Read & Set Bit	R	CSRRS rd,csr,rs1
	Atomic Read & Clear Bit	R	CSRRC rd,csr,rs1
	Atomic R/W Imm	R	CSRRWI rd,csr,imm
	Atomic Read & Set Bit Imm	R	CSRRSI rd,csr,imm
	Atomic Read & Clear Bit Imm	R	CSRRCI rd,csr,imm
Change Level	Env. Call	R	SCALL
	Environment Breakpoint	R	SBREAK
	Environment Return	R	ERET
Trap Redirect	to Supervisor	R	MRTS
	Redirect Trap to Hypervisor	R	MRTH
	Hypervisor Trap to Supervisor	R	HRTS
Interrupt	Wait for Interrupt	R	WFI
MMU	Supervisor FENCE	R	SFENCE.VM rs1

Optional Multiply-Divide Extension: RV32M

Category	Name	Fmt	RV32M (Mult-Div)
Multiply	MULtiply	R	MUL{W|D} rd,rs1,rs2
	MULtiply upper Half	R	MULH rd,rs1,rs2
	MULtiply Half Sign/Uns	R	MULHSU rd,rs1,rs2
	MULtiply upper Half Uns	R	MULHU rd,rs1,rs2
Divide	DIVide	R	DIV{W|D} rd,rs1,rs2
	DIVide Unsigned	R	DIVU rd,rs1,rs2
Remainder	REMainder	R	REM{W|D} rd,rs1,rs2
	REMainder Unsigned	R	REMU{W|D} rd,rs1,rs2

Optional Atomic Instruction Extension: RVA

Category	Name	Fmt	RV{32|64|128}A (Atomic)
Load	Load Reserved	R	LR.{W|D|Q} rd,rs1
Store	Store Conditional	R	SC.{W|D|Q} rd,rs1,rs2
Swap	SWAP	R	AMOSWAP.{W|D|Q} rd,rs1,rs2
Add	ADD	R	AMOADD.{W|D|Q} rd,rs1,rs2
Logical	XOR	R	AMOXOR.{W|D|Q} rd,rs1,rs2
	AND	R	AMOAND.{W|D|Q} rd,rs1,rs2
	OR	R	AMOOR.{W|D|Q} rd,rs1,rs2
Min/Max	MINimum	R	AMOMIN.{W|D|Q} rd,rs1,rs2
	MAXimum	R	AMOMAX.{W|D|Q} rd,rs1,rs2
	MINimum Unsigned	R	AMOMINU.{W|D|Q} rd,rs1,rs2
	MAXimum Unsigned	R	AMOMAXU.{W|D|Q} rd,rs1,rs2

16-bit (RVC) and 32-bit Instruction Formats

3 Optional FP Extensions: RV32{F|D|Q}

Category	Name	Fmt	RV{F|D|Q} (HP/SP,DP,QP)
Load	Load	I	FL{W,D,Q} rd,rs1,imm
Store	Store	S	FS{W,D,Q} rs1,rs2,imm
Arithmetic	ADD	R	FADD.{S|D|Q} rd,rs1,rs2
	SUBtract	R	FSUB.{S|D|Q} rd,rs1,rs2
	MULtiply	R	FMUL.{S|D|Q} rd,rs1,rs2
	DIVide	R	FDIV.{S|D|Q} rd,rs1,rs2
	SQuare RooT	R	FSQRT.{S|D|Q} rd,rs1
Mul-Add	Multiply-ADd	R	FMADD.{S|D|Q} rd,rs1,rs2,rs3
	Multiply-SUBtract	R	FMSUB.{S|D|Q} rd,rs1,rs2,rs3
	Negative Multiply-SUBtract	R	FNMSUB.{S|D|Q} rd,rs1,rs2,rs3
	Negative Multiply-ADD	R	FNMADD.{S|D|Q} rd,rs1,rs2,rs3
Sign Inject	SIGN source	R	FSGNJ.{S|D|Q} rd,rs1,rs2
	Negative SIGN source	R	FSGNJN.{S|D|Q} rd,rs1,rs2
	Xor SIGN source	R	FSGNJX.{S|D|Q} rd,rs1,rs2
Min/Max	MINimum	R	FMIN.{S|D|Q} rd,rs1,rs2
	MAXimum	R	FMAX.{S|D|Q} rd,rs1,rs2
Compare	Compare Float	R	FEQ.{S|D|Q} rd,rs1,rs2
	Compare Float <	R	FLT.{S|D|Q} rd,rs1,rs2
	Compare Float ≤	R	FLE.{S|D|Q} rd,rs1,rs2
Categorize	Classify Typ	R	FCLASS.{S|D|Q} rd,rs1
Move	Move from Integer	R	FMV.S.X rd,rs1
	Move to Integer	R	FMV.X.S rd,rs1
Convert	Convert from Int	R	FCVT.{S|D|Q}.W rd,rs1
	Convert from Int Unsigned	R	FCVT.{S|D|Q}.WU rd,rs1
	Convert to Int	R	FCVT.W.{S|D|Q} rd,rs1
	Convert to Int Unsigned	R	FCVT.WU.{S|D|Q} rd,rs1
Configuration	Read Stat	I	FRCSR rd
	Read Rounding Mode	I	FRRM rd
	Read Flags	I	FRFLAGS rd
	Swap Status Reg	R	FSCSR rd,rs1
	Swap Rounding Mode	R	FSRM rd,rs1
	Swap Flags	R	FSFLAGS rd,rs1
Swap Rounding Mode Imm		I	FSRMI rd,imm
	Swap Flags Imm	I	FSFLAGSI rd,imm

3 Optional FP Extensions: RV{64\|128}{F|D|Q}

Category	Name	Fmt	RV{F|D|Q} (HP/SP,DP,QP)
Move	Move from Integer	R	FMV.{D|Q}.X rd,rs1
	Move to Integer	R	FMV.X.{D|Q} rd,rs1
Convert	Convert from Int	R	FCVT.{S|D|Q}.{L|T} rd,rs1
	Convert from Int Unsigned	R	FCVT.{S|D|Q}.{L|T}U rd,rs1
	Convert to Int	R	FCVT.{L|T}.{S|D|Q} rd,rs1
	Convert to Int Unsigned	R	FCVT.{L|T}U.{S|D|Q} rd,rs1

RISC-V Reference Card ④

Optional Compressed Instructions: RVC

Category	Name	Fmt	RVC
Loads	Load Word	CL	C.LW rd',rs1',imm
	Load Word SP	CI	C.LWSP rd,imm
	Load Double	CL	C.LD rd',rs1',imm
	Load Double SP	CI	C.LWSP rd,imm
	Load Quad	CL	C.LQ rd',rs1',imm
	Load Quad SP	CI	C.LQSP rd,imm
	Load Byte Unsigned	CL	C.LBU rd',rs1',imm
	Float Load Word	CL	C.FLW rd',rs1',imm
	Float Load Double	CL	C.FLD rd',rs1',imm
	Float Load Word SP	CI	C.FLWSP rd,imm
	Float Load Double SP	CI	C.FLDSP rd,imm
Stores	Store Word	CS	C.SW rs1',rs2',imm
	Store Word SP	CSS	C.SWSP rs2,imm
	Store Double	CS	C.SD rs1',rs2',imm
	Store Double SP	CSS	C.SDSP rs2,imm
	Store Quad	CS	C.SQ rs1',rs2',imm
	Store Quad SP	CSS	C.SQSP rs2,imm
	Float Store Word	CSS	C.FSW rd',rs1',imm
	Float Store Double	CSS	C.FSD rd',rs1',imm
	Float Store Word SP	CSS	C.FSWSP rd,imm
	Float Store Double SP	CSS	C.FSDSP rd,imm
Arithmetic	ADD	CR	C.ADD rd,rs1
	ADD Word	CR	C.ADDW rd',rs2'
	ADD Immediate	CI	C.ADDI rd,imm
	ADD Word Imm	CI	C.ADDIW rd,imm
	ADD SP Imm * 16	CI	C.ADDI16SP x0,imm
	ADD SP Imm * 4	CIW	C.ADDI4SPN rd',imm
	Load Immediate	CI	C.LI rd,imm
	Load Upper Imm	CI	C.LUI rd,imm
	MoVe	CR	C.MV rd,rs1
	SUB	CR	C.SUB rd',rs2'
	SUB Word	CR	C.SUBW rd',rs2'
Logical	XOR	CS	C.XOR rd',rs2'
	OR	CS	C.OR rd',rs2'
	AND	CS	C.AND rd',rs2'
	AND Immediate	CB	C.ANDI rd',rs2'
Shifts	Shift Left Imm	CI	C.SLLI rd,imm
	Shift Right Immediate	CB	C.SRLI rd',imm
	Shift Right Arith Imm	CB	C.SRAI rd',imm
Branches	Branch=0	CB	C.BEQZ rs1',imm
	Branch≠0	CB	C.BNEZ rs1',imm
Jump	Jump	CJ	C.J imm
	Jump Register	CR	C.JR rd,rs1
Jump & Link	J&L	CJ	C.JAL imm
	Jump & Link Register	CR	C.JALR rs1
System	Env. BREAK	CI	C.EBREAK

Role of a Computer Architect

- Starting from technology and demands
  - Manufacturing, e.g. 7nm.
  - Packaging, e.g. chiplet, wafer-scale, EMIB.
  - Application requirements, e.g. AI.
- Explore the design space of various components
  - In-order vs OOO
  - SRAM vs. DRAM
  - Cache vs. scratchpad
  - Coherence and consistency
- Deliver products with strong PPA metrics
  - Performance, power, area

Analogy Breakdown

We care less about a particular instance of a design...

- **Fungibility**
  - No intrinsic value to a particular instance -- Don't care where my program lives
- **Durability**
  - Every two years you throw away your personal out-of-order multicore chip (smartphone) and buy a new one.
  - Compute devices will anyways become obsolete due to technology
- **Manufacturing Tradeoffs**
  - Nth+1 chip costs ~$0

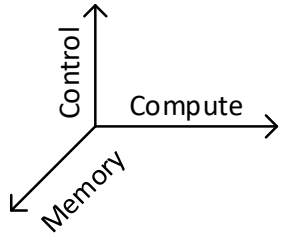
Design Goals / Constraints

- **Functional**
  - Needs to be correct
    - And unlike software, difficult to update once deployed
  - Security: Should provide guarantees to software
- **Reliable**
  - Does it ***continue*** to perform correctly?
  - Hard fault vs transient fault
  - Space satellites vs desktop vs server reliability
- **High performance**
  - Many factors to performance, Iron Law
- **Generality**
  - "Fast" is only meaningful in the context of a set of important tasks
  - Impossible goal: fastest possible design for all programs

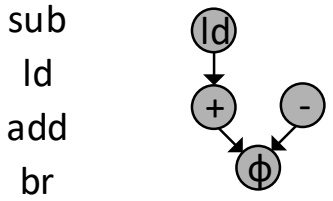
Design Goals / Constraints ... continued

- **Low cost**
  - Design/verification cost
  - Cost of making first chip after design (mask cost)
  - Per unit manufacturing cost (wafer cost)
- **Low power/energy**
  - Energy in (battery life, cost of electricity)
  - Energy out (cooling and related costs)
  - Cyclic problem, very much a problem today
- **Low carbon contribution?**
- Challenge: balancing the relative importance of these goals
  - And the balance is constantly changing
  - No goal is absolutely important at expense of all others

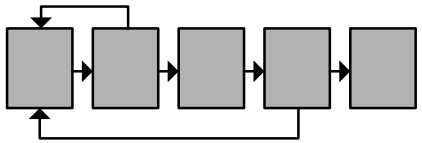
Layers of Abstraction



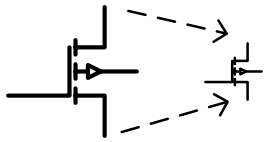
Applications



ISA: Hardware/
Software Interface



Microarchitecture



Technology

Architects' Domain

Layers of Abstraction

Software World

Algorithm

Application

Programming Language

Compiler

Operating System

Instruction Set Architecture (ISA)

Hardware Organization

Component Design

Circuit Design

Devices (Transistors)

Physics/Manufacturing

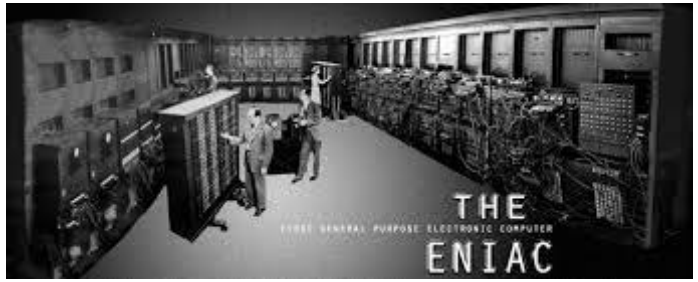
Hardware World

Early Computers: Had No "Architecture"!

- Each Computer Required Tremendous Effort
  - Hardware: Stuff that has mass
  - Software: Stuff that has no mass (instructions)

Betty Jean Jennings and Fran Bilas

- Problem: Boundary between hardware/software was fuzzy
  - No standard interface to specify programs!
  - New programs require fiddling with hardware, yuk!
  - New hardware required fiddling with programs, gross!
  - Software Lagged Hardware



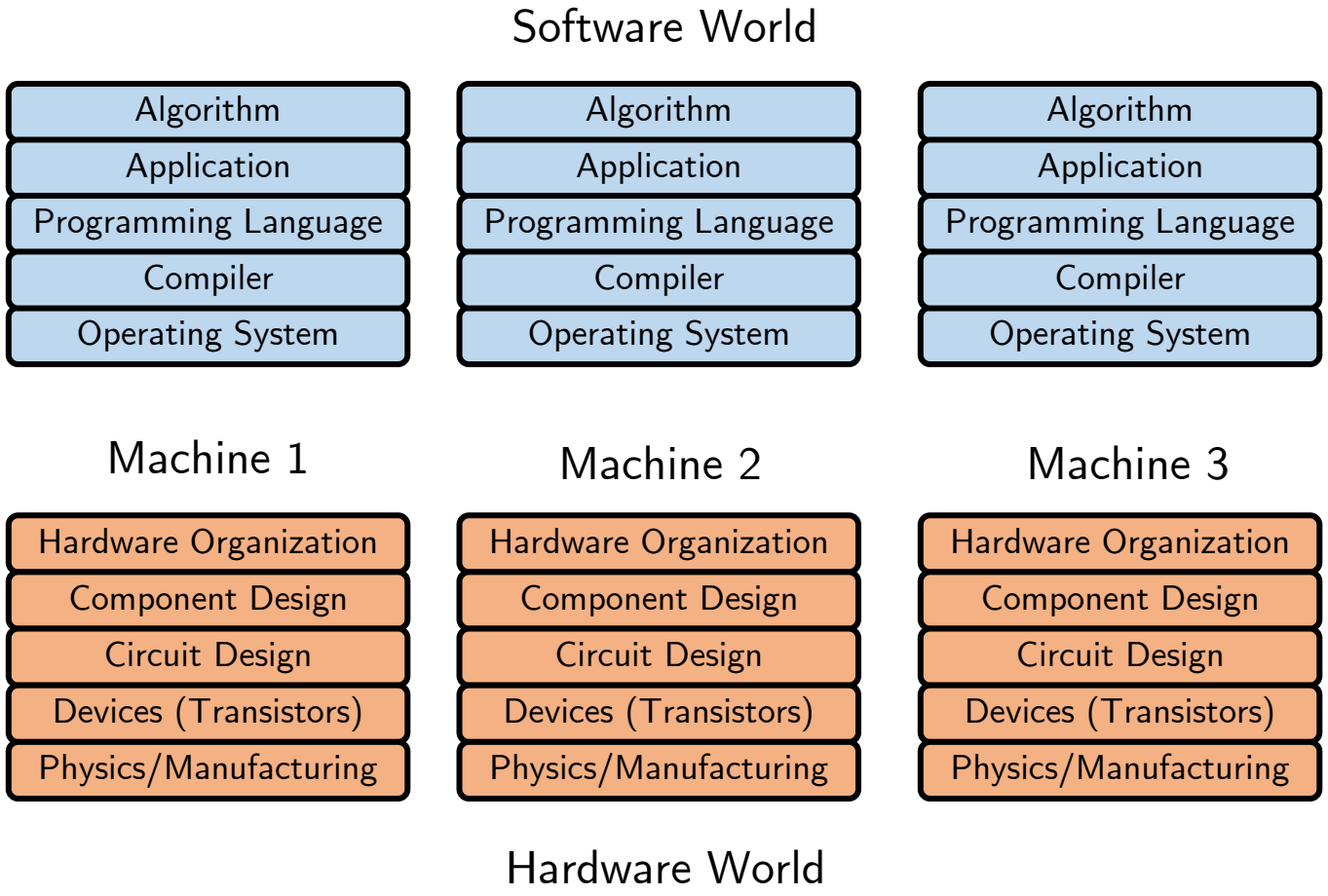
Creators: John Mauchly, J. Presper Eckert

- Unimaginable today
  - Going forward: Need common interface b/t HW & SW – instruction set architecture
  - Question: Is the ISA the right way to do that?

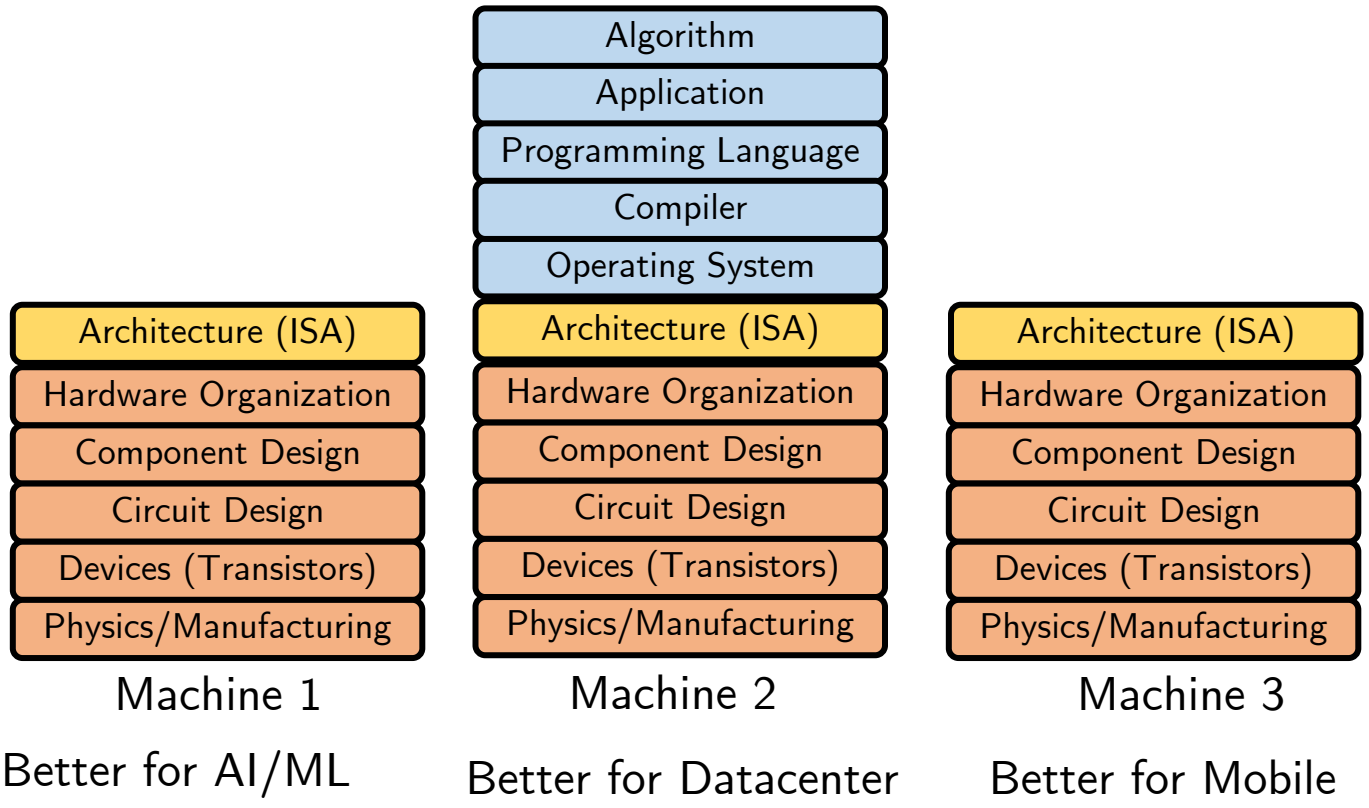
ENIAC: First machine with abstractions, kindof

John Von Neumann -> popularized the concept of ISA

Without ISA


Software World
Algorithm
Application
Programming Language
Compiler
Operating System
Algorithm
Application
Programming Language
Compiler
Operating System
Algorithm
Application
Programming Language
Compiler
Operating System
Machine 1
Machine 2
Machine 3
Hardware Organization
Component Design
Circuit Design
Devices (Transistors)
Physics/Manufacturing
Hardware Organization
Component Design
Circuit Design
Devices (Transistors)
Physics/Manufacturing
Hardware Organization
Component Design
Circuit Design
Devices (Transistors)
Physics/Manufacturing
Hardware World

ISAs: Language for HW/SW Interaction



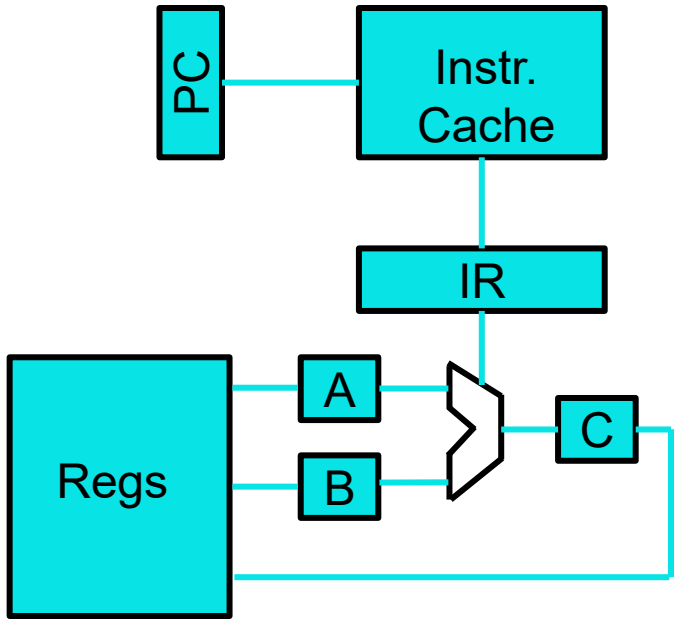
Instruction Set Architecture

- Challenge: Enable efficient expression of algorithms
- Expose information that hardware can use... (ok, vague)
- Expose Parallelism
  - Instruction Parallelism
  - Lack false serialization/dependences
  - What else?
- A good compiler target
  - Simple & orthogonal
- Dense
  - Reduce cost of reading/storing instructions
- Permits Control/Virtualization
  - Interruptible/Preemptable
  - Virtual memory

Microarchitecture

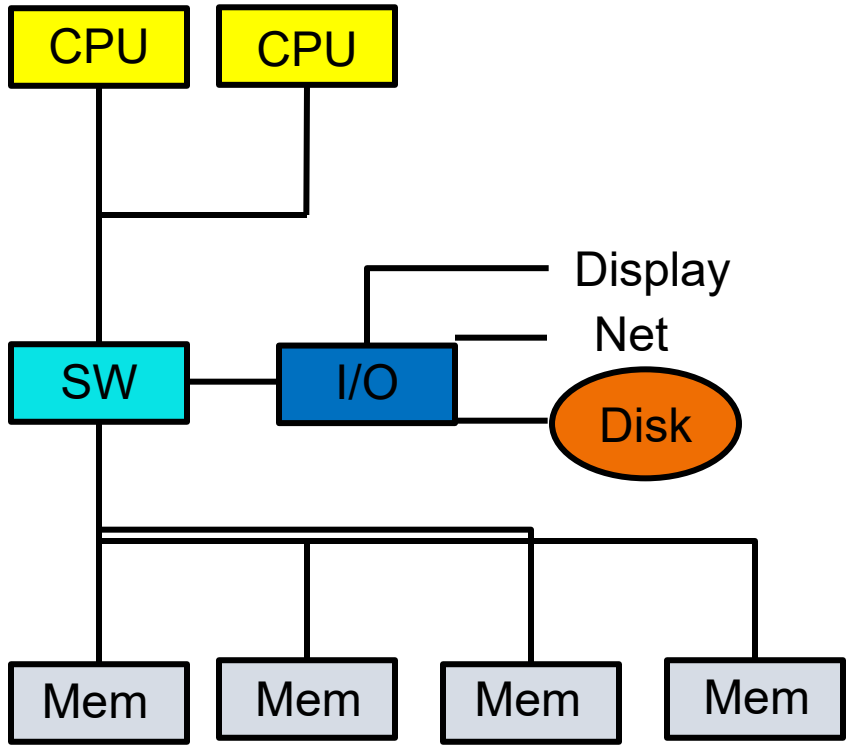
- Challenge: overcome sequential nature of programs

- Example Techniques
  - Deep pipelining
  - Multiple issue
  - Dynamic scheduling： Branch prediction/speculation
- Constraints Up-the-stack
  - Constrained by the ISA
  - Constrained by the behaviors present in applications
- Constraints Down-the-stack
  - Constrained by technology/hardware costs.
  - Physical constraints (power, area, communication)

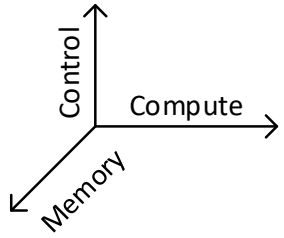


System-Level Design

- Challenge: Balancing possible bottlenecks
  - processors, memories, and interconnect
- Feeds and speeds
  - Constrained by IC pin count, module pin count, and signaling rates
- System balance
  - For a particular application
- Driven by
  - Performance/cost gains
  - Available components (cost/perf)
  - Technology constraints (I/O, network tech, manufacturing, chiplets, etc.)

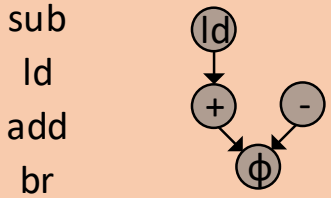


Layers of Abstraction



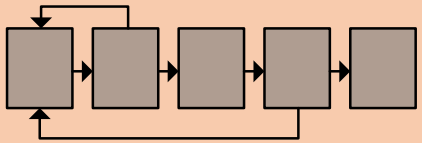
Applications

Major Driver
Going forward?

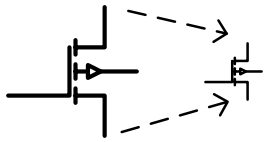


ISA: Hardwae/
Software Interface

Architects'
Domain



Microarchitecture



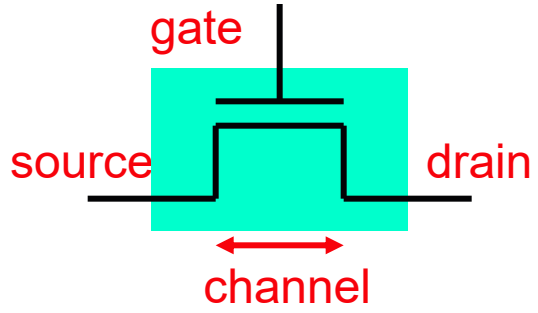
Technology

Major Driver
for 50+ years

Co-evolution of Technology and Microarchitecture

"Technology"

- Basic element
  - Solid-state **transistor** (i.e., electrical switch)
  - Building block of **integrated circuits (ICs)**

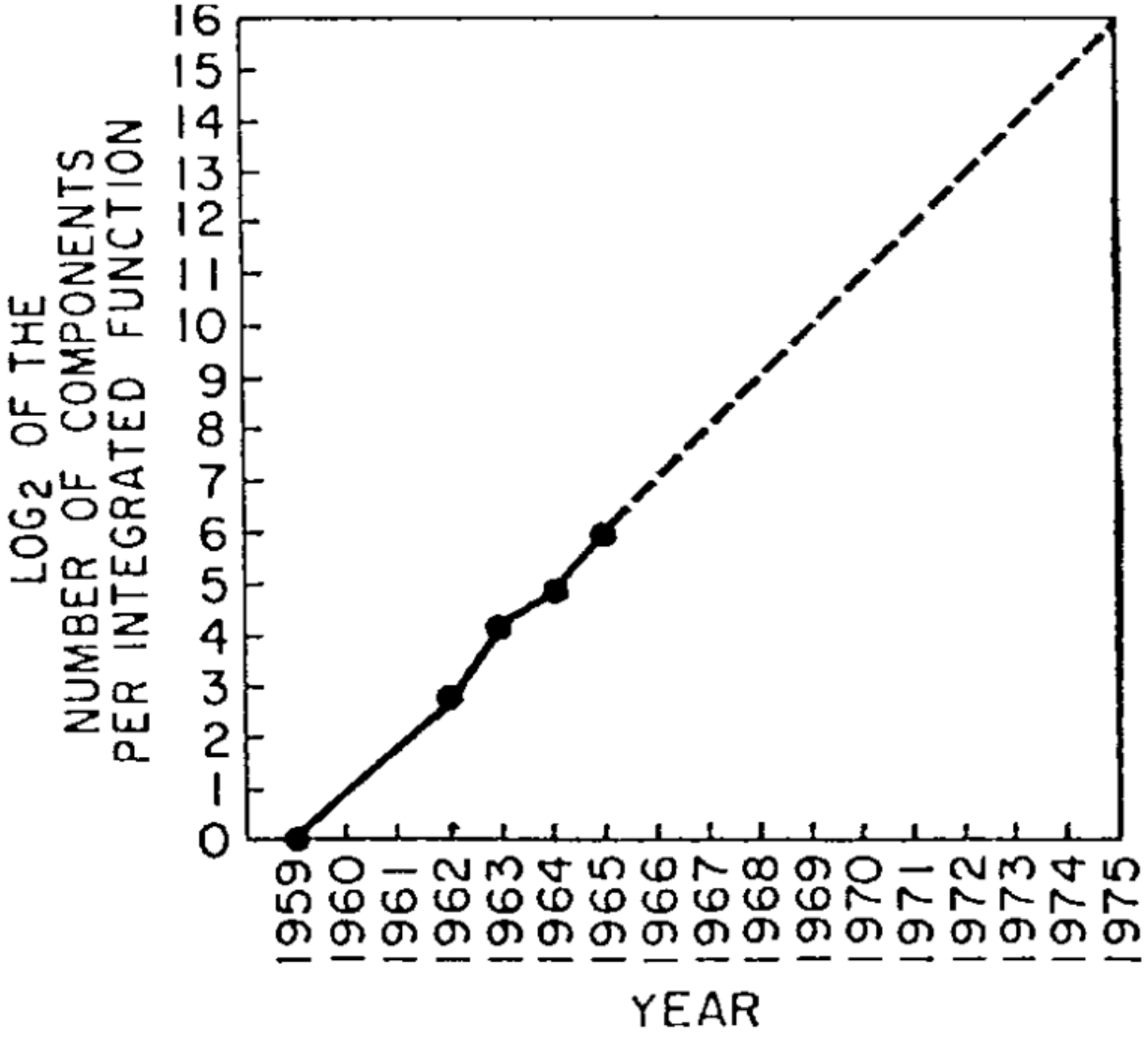


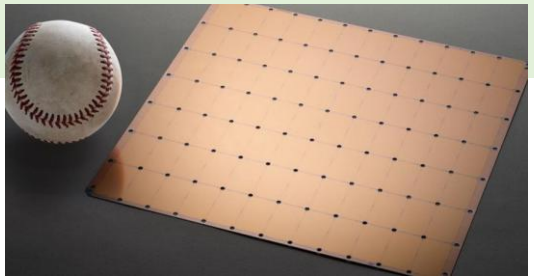
- Why ICs changed everything
  - High performance, high reliability, low cost, low power
  - Economies of scale via fabrication

- Several kinds of IC families
  - **SRAM/logic**: optimized for speed (used for processors)
  - **DRAM**: optimized for density, cost (used for memory)
  - **Flash/NVM**: optimized for non-volatility, density, cost (storage)
  - Increasing opportunities for integrating multiple technologies
    - Chiplets and Die Stacking

- Non-CMOS storage and interconnect technologies
  - Ethernet, fiber optics, wireless, Disks

Moore's Law -- 1965

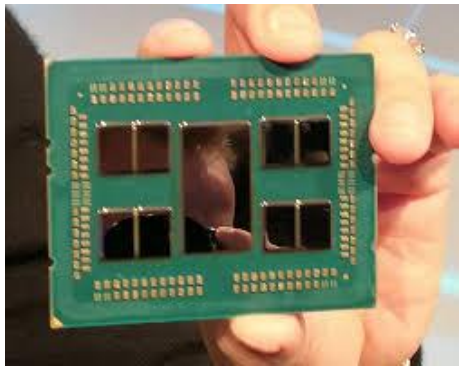




Cerebras
WS1

WS3

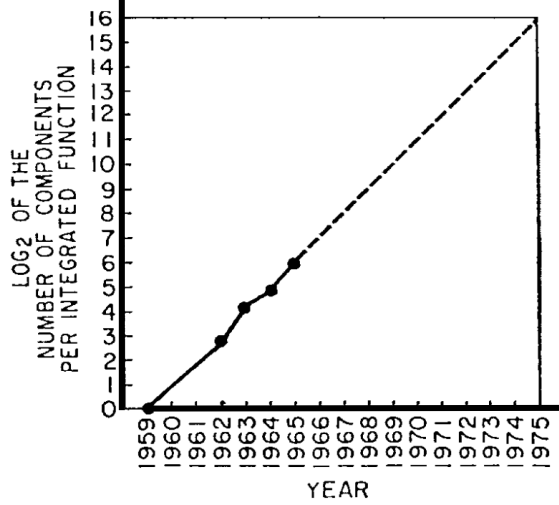
AMD EPYC
(chiplet)



2^{45}

2^{40}

2^{35}



2020

2025

Technology Change Drives Everything

- Historic/Optimistic View: Computers get 10x faster & more energy efficient every 10 years!
  - A 10x quantitative change is qualitative change
  - Plane is 10x faster than car, and fundamentally different travel mode
- New applications become self-sustaining market segments
  - Examples: laptops, mobile phones, virtual/augmented reality, autonomous vehicles, brain-computer interfaces, energy-harvesting intermittent devices etc.
- Low-level improvements appear as discrete high-level jumps
  - Capabilities cross thresholds, enabling new applications and uses

Revolution I: The Microprocessor

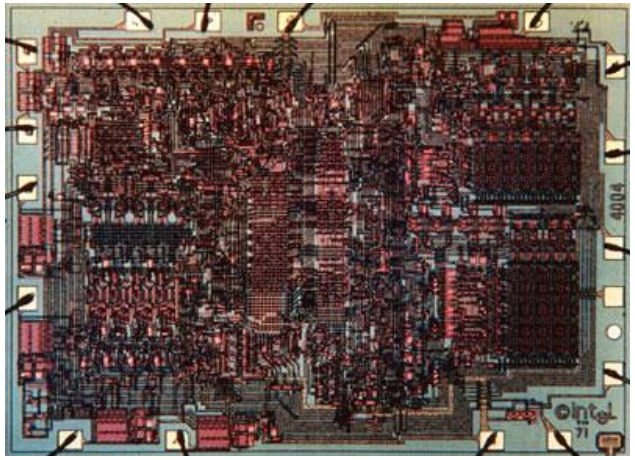
- **Microprocessor revolution**
  - One significant technology threshold was crossed in 1970s
  - Enough transistors (~25K) to put a 16-bit processor on one chip
  - Huge performance advantages: fewer slow chip-crossings
  - Even bigger cost advantages: one "stamped-out" component

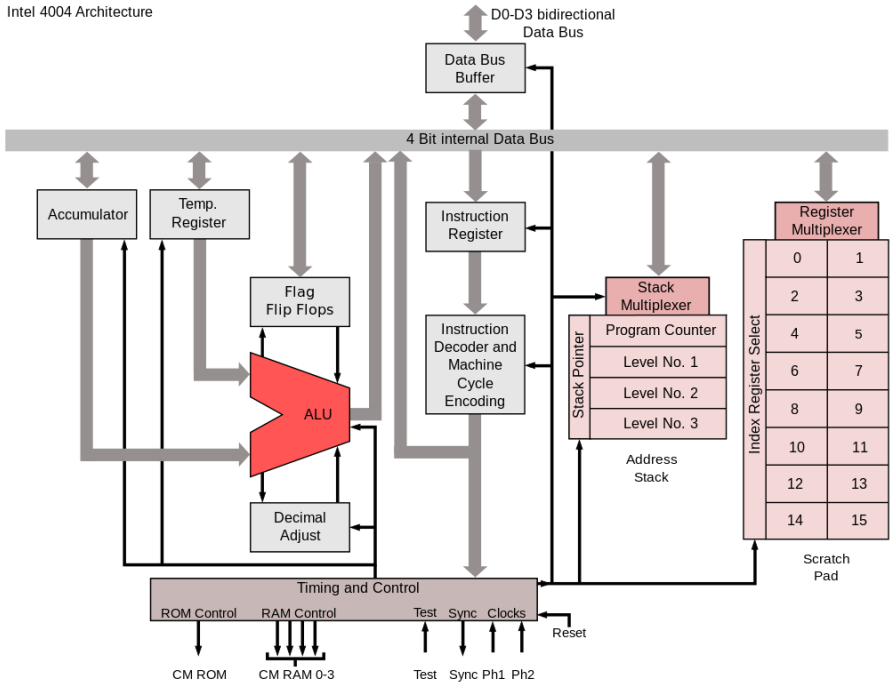
- Microprocessors created new market segments
  - Desktops, CD/DVD players, laptops, game consoles, set-top boxes, digital camera, mp3 players, GPS, mobile phones

- And replaced incumbents in existing segments
  - Microprocessor-based system replaced "mainframes", "minicomputers", etc.

First Microprocessor

- Intel 4004 (1971)
  - Application: calculators
  - Technology: 10000 nm
  - 2300 transistors
  - 13 mm^2
  - 108 KHz
  - 12 Volts
  - 4-bit data
  - Single-cycle datapath





Revolution II: Implicit Parallelism

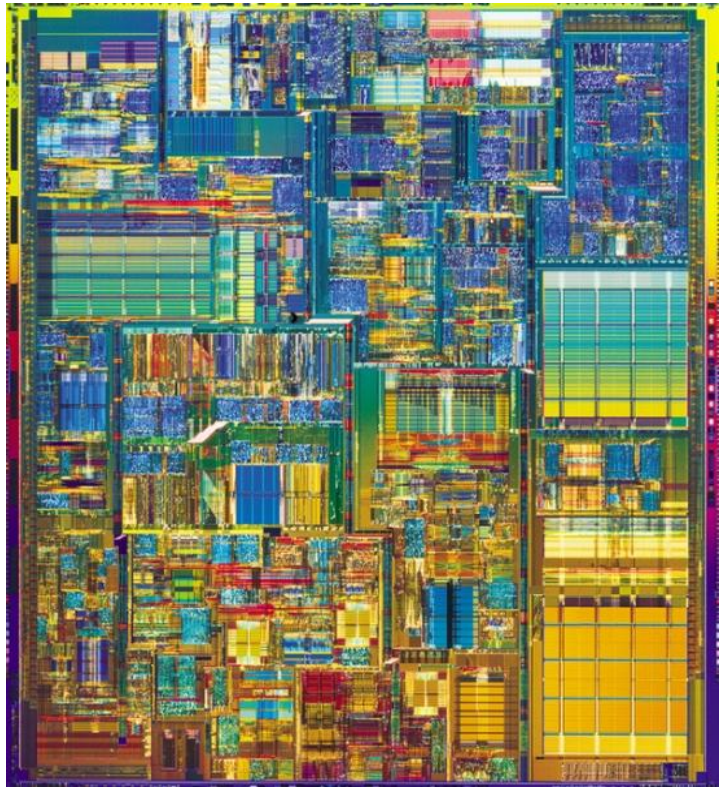
- Then to **extract implicit instruction-level parallelism**
  - Hardware provides parallel resources, figures out how to use them
  - Software is oblivious

- Initially using pipelining ...
  - Which also enabled increased clock frequency
- ... caches ...
  - Which became necessary as processor clock frequency increased
- ... and integrated floating-point
- Then deeper pipelines and branch speculation
- Then multiple instructions per cycle (superscalar)
- Then dynamic scheduling (out-of-order execution)

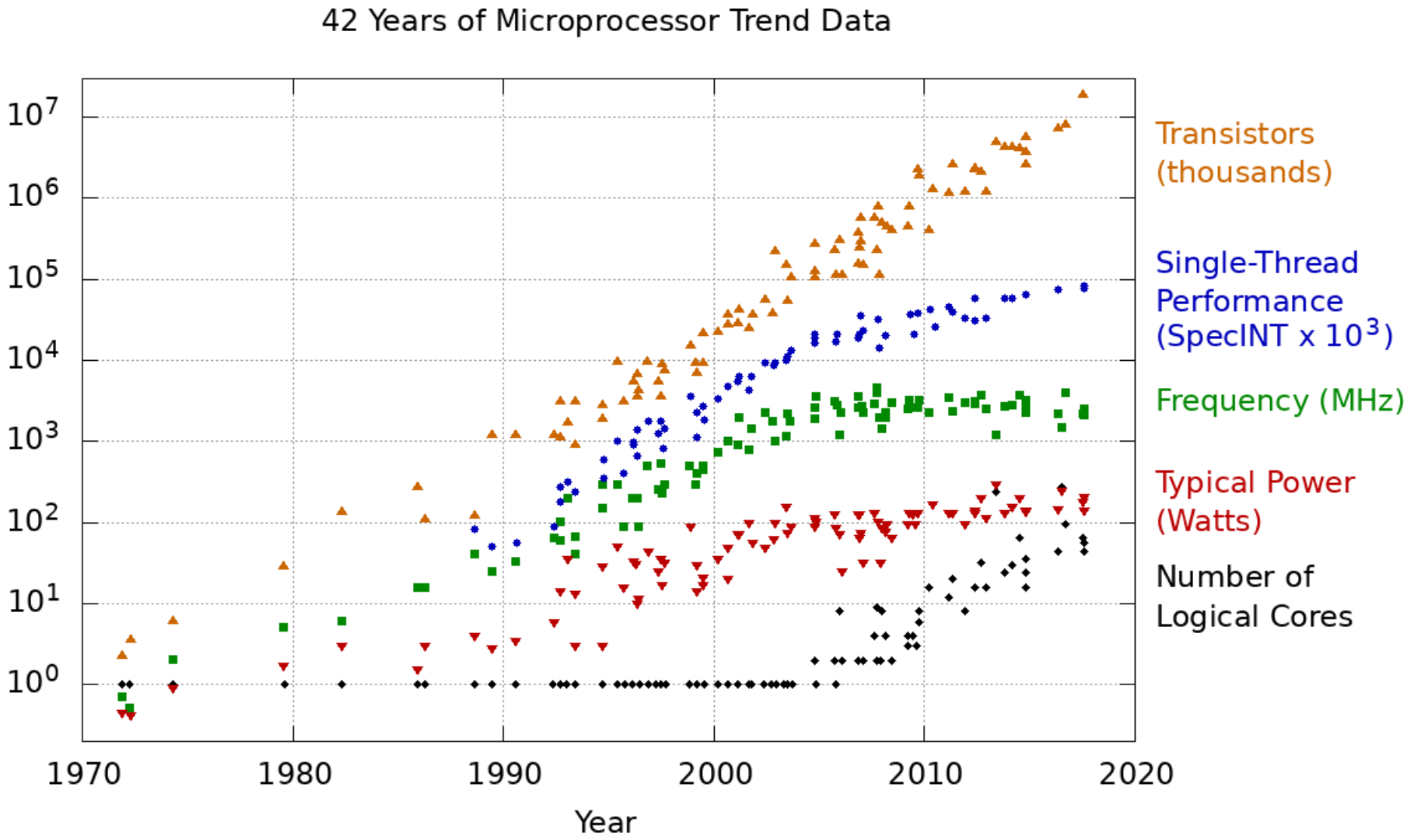
- We will talk about these things

Pinnacle of single-core microprocessors

- Intel Pentium4 (2003)
  - Market: Desktop/server
  - Technology: 90nm (1/100x)
  - 55M transistors (20,000x)
  - 101 mm^2 (10x)
  - 3.4 GHz (10,000x)
  - 1.2 Volts (1/10x)
  - 32/64-bit data (16x)
  - 22-stage pipelined datapath (22x)
  - 3 instructions per cycle (superscalar)
  - Two levels of on-chip cache
  - data-parallel vector (SIMD) instructions, hyperthreading



Continue the Scaling



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten

New plot and data collected for 2010-2017 by K. Rupp

Revolution III: Explicit Parallelism

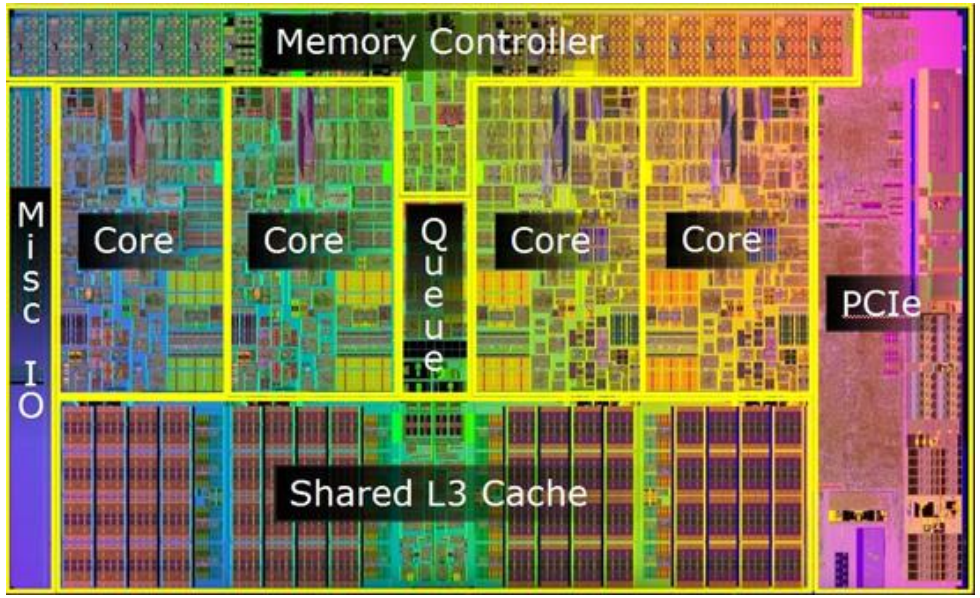
- Support **explicit data & thread level parallelism**
  - Hardware provides parallel resources, software specifies usage
  - Why? diminishing returns on instruction-level-parallelism
- First using (subword) vector instructions..., Intel's SSE
  - One instruction does four parallel multiplies
- ... and general support for multi-threaded programs
  - Coherent caches, hardware synchronization primitives
- Then using support for multiple concurrent threads on chip
  - First with single-core multi-threading, now with multi-core

Multicore Processor

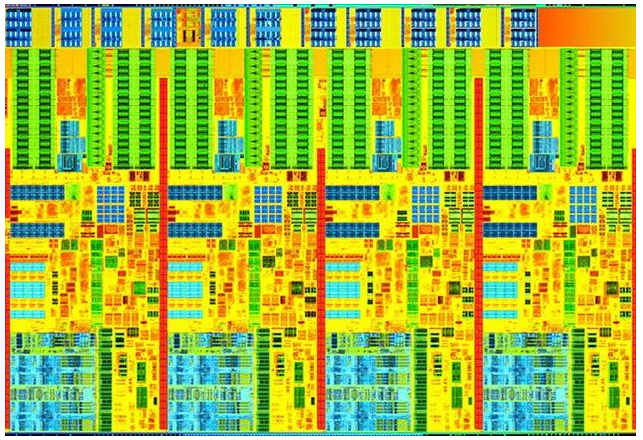
- Intel Core i7 (2009)
  - Application: desktop/server
  - Technology: 45nm (1/2x)

  - 774M transistors (12x)
  - 296 mm^2 (3x)
  - 3.2 GHz to 3.6 Ghz (~1x)
  - 0.7 to 1.4 Volts (~1x)

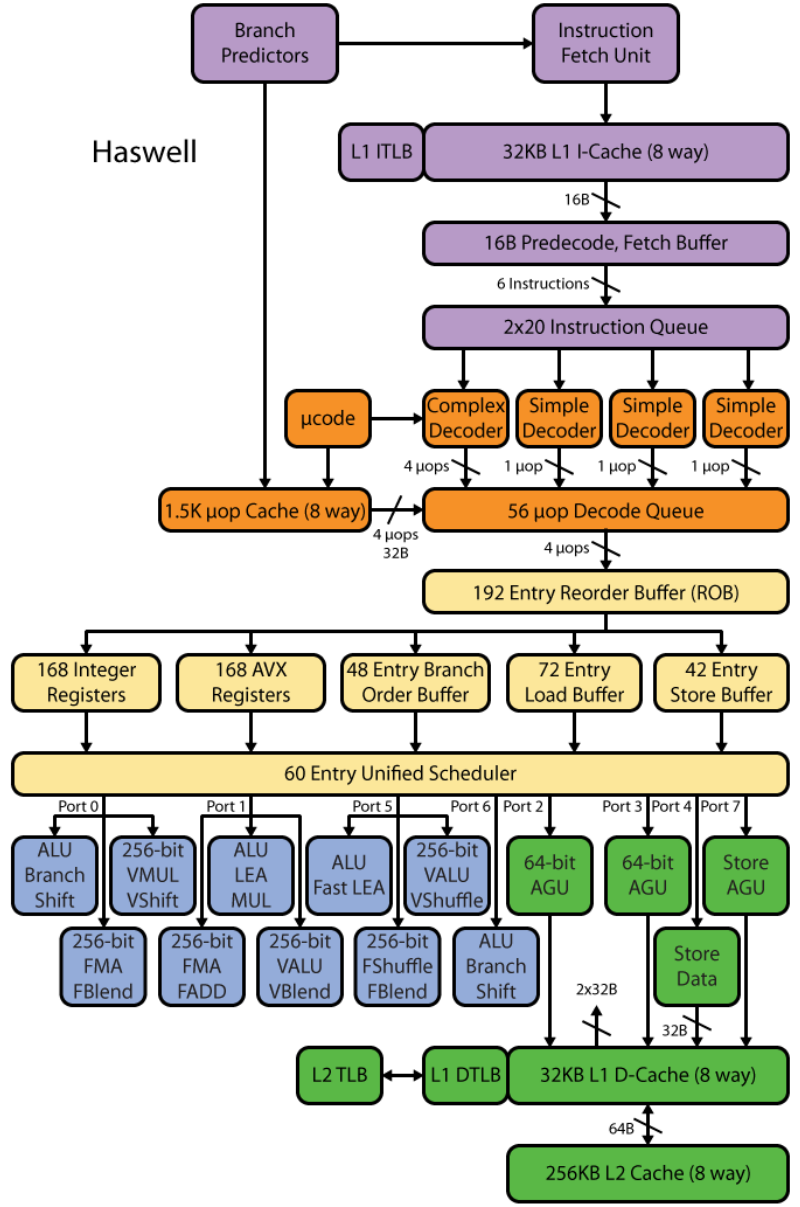
  - 128-bit data (2x)
  - 14-stage pipelined datapath (0.5x)
  - 4 instructions per cycle (~1x)
  - *Three levels of on-chip cache*
  - *data-parallel vector (SIMD) instructions*, hyperthreading
  - **Four-core multicore** (4x)



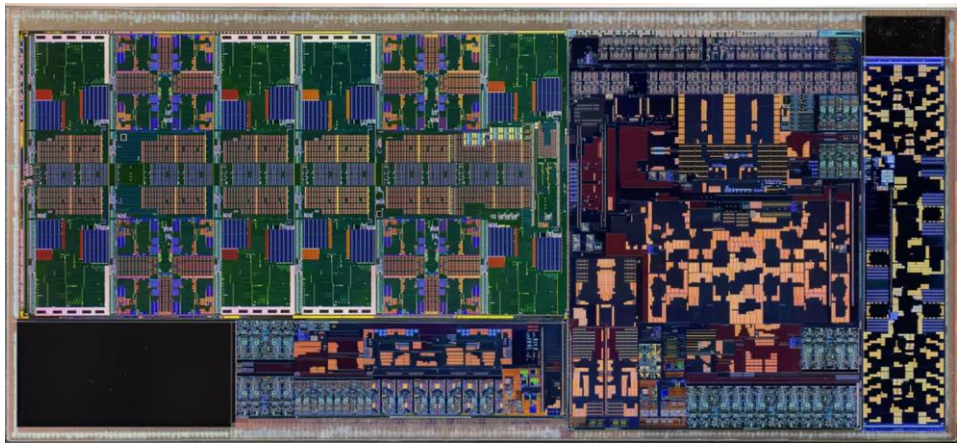
General Purpose 12 Years Ago 2014



Intel Haswell



General Purpose in 2026



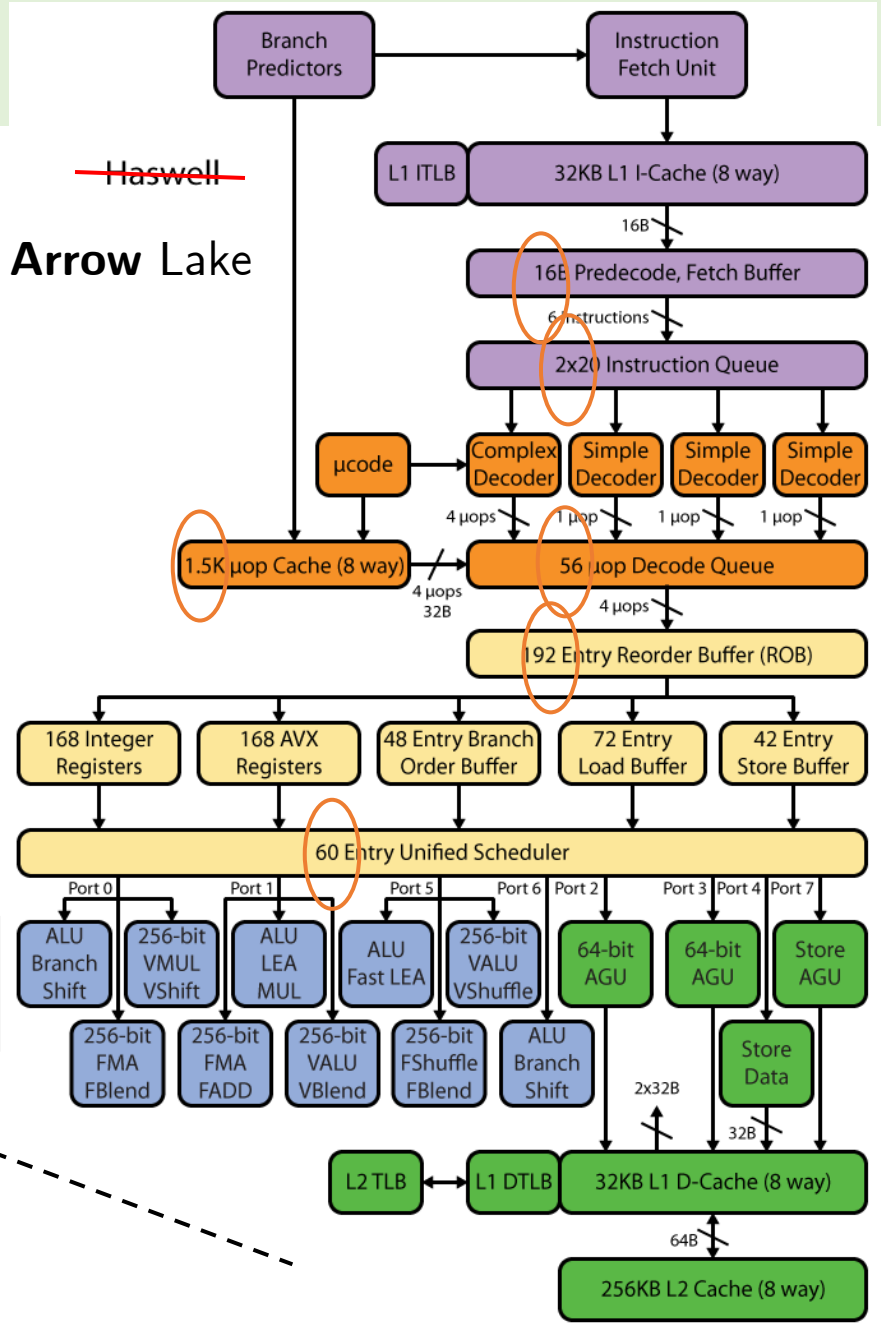
Matrix Multiply

Arrow Lake

~2x Single Core Speedup

~~Haswell~~

Arrow Lake



Oh, Intel (Well Thanks to Trump!)

-12.28 (-23.78%) ↓ past 5 years

Closed: Jan 6, 12:17 AM EST • Disclaimer
After hours 39.44 +0.069 (0.18%)

1D | 5D | 1M | 6M | YTD | 1Y | 5Y | Max

Open	41.59	Mkt cap	187.79B	Dividend	-
High	42.10	P/E ratio	3,710.65	Qtrly Div Amt	-
Low	39.27	52-wk high	44.01	52-wk low	17.66

NVIDIA Mkt cap 4.57T

(~25x)

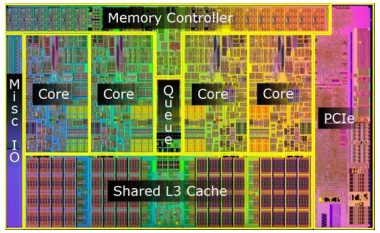
Revolution IV: Specialization

- Combine implicit/explicit parallelism with a focus on a particular domain
  - Scope can be very different: GPGPUs are quite broad, while TPUs are not...
- Tradeoff the overheads of supporting "general purpose" workloads for efficiency on a smaller set of workloads.
- But why is this happening now?
  - Because its hard and there wasn't a reason to do it before.
    (tech scaling was enough to satisfy exponential growth in speed/efficiency)
  - Energy and Area are now big limiting factors

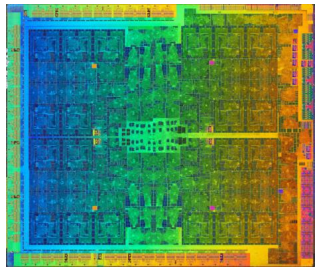
Specialization Spectrum

General Purpose ⟷ Application Specific

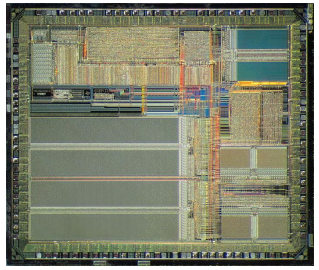
CPU
("Ordinary" Apps)



Graphics Proc. Unit (GPU)

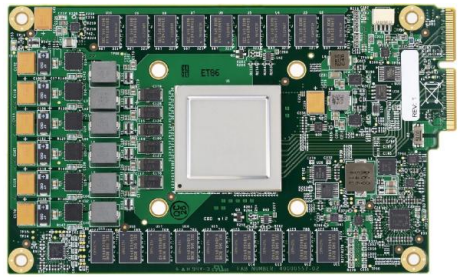


Digital Signal Proc. (DSP)

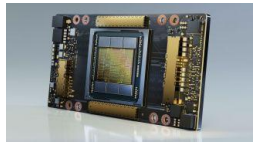


Field Prog. Gate Array (FPGA)

Google TPU:
(Deep Learning)



Machine learning in Industry



NVIDIA A100

Google TPU

Microsoft Brainwave

Amazon Inferentia

Baidu Kunlun

+Apple & Meta

Cambricon MLU-100

GraphCore Colossus

Groq TSP

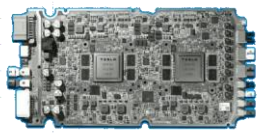
Cerebras WSE

Habana Gaudi

Mythic IPU

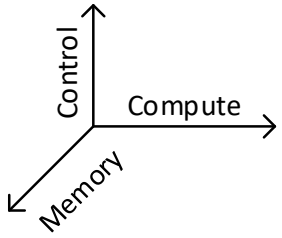
HAILO HAILO-8

Tenstorrent Grayskull



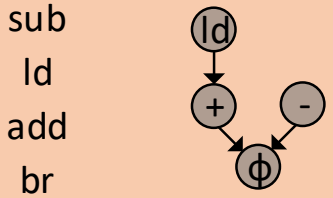
Tesla D1

Layers of Abstraction



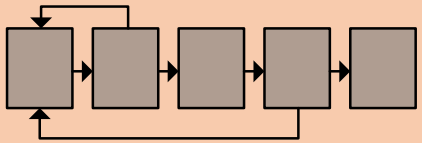
Applications

Major Driver Going forward?

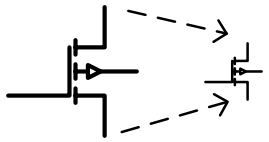


ISA: Hardwae/ Software Interface

Architects' Domain



Microarchitecture



Technology

Major Driver for 50+ years

Applications as a Driver

Applications Views

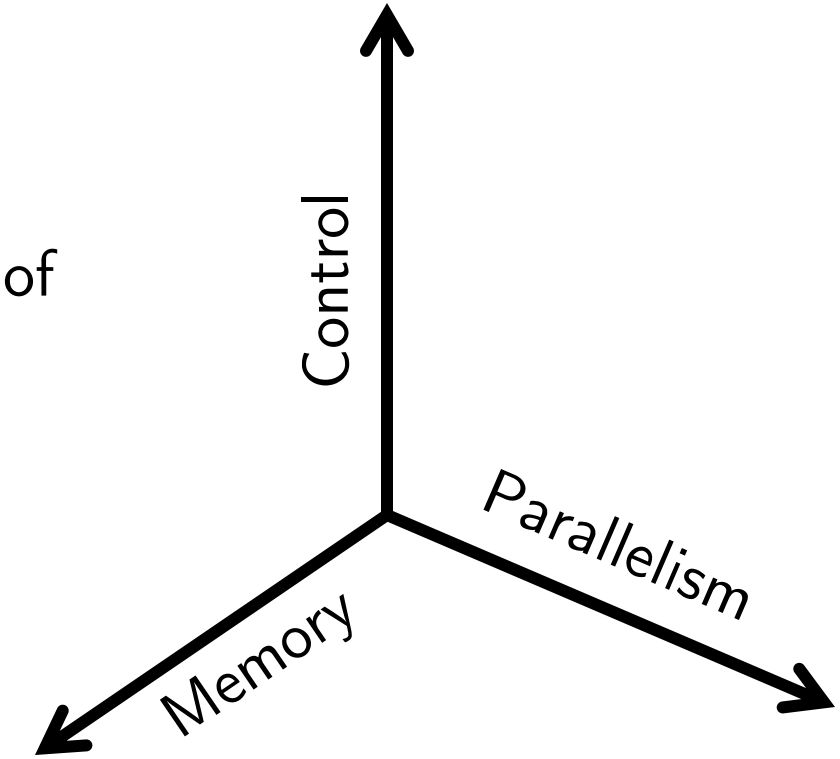
- Many ways to view distinction between application settings:
  - **Deployment-centric view:**
    - Where the machine is deployed affects the set of applications
    - Desktop, Server, Mobile, IoT
  - **Domain-centric view:**
    - Set applications from a similar background
  - **Property-centric view:**
    - Set of applications with different properties

Domain-centric View

- Old Dichotomy:
  - **Scientific**: weather prediction, physical simulation, genome sequencing
    - First computing application domain: naval ballistics firing table
    - Need: large memory, heavy-duty floating point
    - Examples: CRAY T3E, IBM BlueGene
  - **Commercial**: database/web serving, e-commerce
    - Need: data movement, high memory + I/O bandwidth
    - Examples: Sun Enterprise Server, AMD Opteron, Intel Xeon, IBM Power 7
- **Recently – finer grain domains:**
  - Deep learning, Digital Signal Processing, Graphics, Genomics, Database Processing, Compression/Decompression, Molecular Dynamics (anton by DE shaw)

Property-centric View

Dimensions of Application "Regularity"



More regularity → Less data dependences

Less dependences → Easier to run efficiently

Control Regularity

Increasing "Irregularity"

- No Control (or non critical)

```
for i
  ... = a[i]
```

- Data-Independent

```
for i
  if(i%2)
    ... = a[i]
```

- Data-Dependent, Predictable

```
for i
 if(age[i]>2)
    ... = a[i]
```

- Data-Dependent, Unpredictable

```
for i
  if(age[i]>22)
    ... = a[i]
```

(also, indirect branches)

Memory Regularity

Increasing "Irregularity"

- Data dependence

```
for i=0 to n
  ... = a[i]
```

```
while(node)
  ... = *node
  node = node->next
```

- Alias freedom

```
for i=0 to n
  ... = a[i]
```

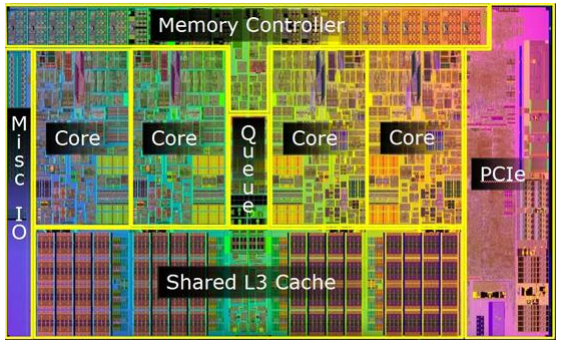
```
for i=0 to n ... =
  a[index[i]]++
```

Parallelism Regularity

- Types of Parallelism
  - Instruction-level Parallelism (ILP): Nearby instructions running together
  - Thread-level (task-level) Parallelism (TLP): Independent threads (at least to some extent) running simultaneously.
  - Data-level Parallelism (DLP): Do same thing to many pieces of data.
- Which is the most regular: (one perspective)
  - DLP: more regular
  - TLP: less regular (dependences infrequent)
  - ILP: least regular (more frequent dependences)
- Dimensions of Regularity
  - Granularity: Fine vs Coarse Grain
  - Data-dependence: Static vs Dynamic
- Complexity Ahead:
  - DLP implies ILP... (but not other way around)
  - DLP implies TLP ... (but not other way around)

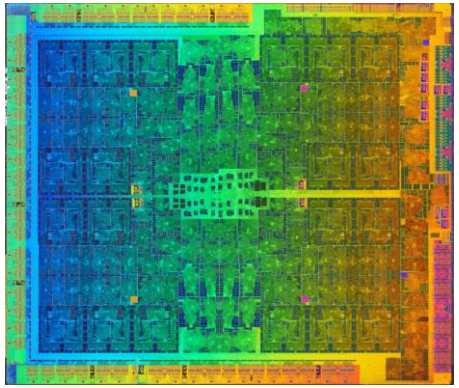
A naïve property-based classification

CPU
("Ordinary" Apps)



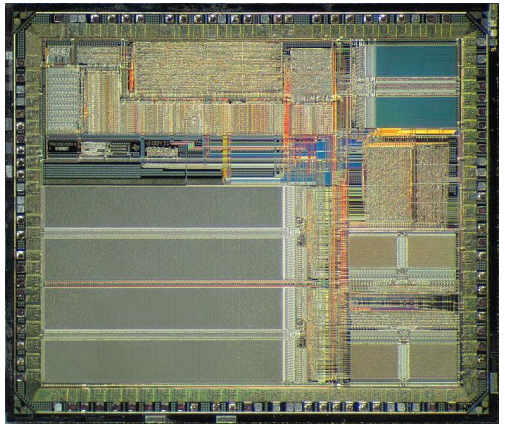
+ Irregular Control/Memory
+ Fine-grain Instruction Level Parallelsim

GPU
(Graphics, Data-proc.)



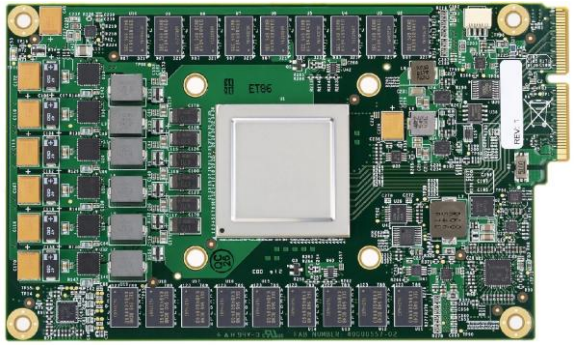
+Thread-level and Data-level Parallelism
+Medium Control/Memory

DSP (signal proc.)



+Fine-grain ILP
+Highly-regular Memory

Google TPU:
(Deep Learning)



+Extremely Regular Data Parallelism
+Extreme Control/Memory Regularity

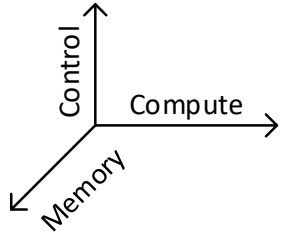
Other properties

- Locality
  - Temporal:
    - Do instructions/data repeat within some time?
    - Loopy vs function-call code
  - Spatial:
    - Reuse related data/instructions in sequence?

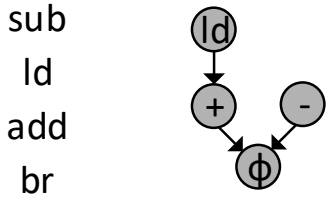
- Data and instruction types
  - Integer vs Floating Point
  - Small vs Large Bitwidth
  - (Ratio of resources + conversion overheads)

- Probably many other important properties, depending on the context and architecture...

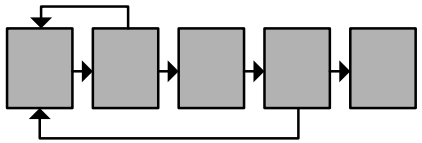
Course Theme: Thinking Across Layers



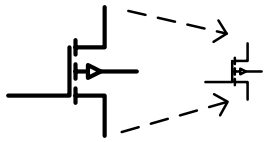
Applications



ISA: Hardwae/ Software Interface



Microarchitecture



Technology

CENG 5410 Course Overview

Some Course Goals

- Exposure to "big ideas" in computer architecture
  - Pipelining, parallelism, caching, locality, abstraction, etc.
  - Understand the "why" behind architecture/systems tradeoffs
  - Exposure to examples of good (and some bad) engineering
- "Practical" Skills
  - Simulators/C++
  - Empirical evaluation/analysis/experiment design
  - Programming for performance
- Research exposure (through papers/project)
  - Evaluating your own idea is rewarding!
  - Skepticism without Cynicism
- Understand where computers are going
  - Future capabilities drive the computing world
  - Forced to think 5+ years into the future

Course Prerequisites

- Basic Computer Organization
  - Logic: gates, Boolean functions, latches, memories
  - Datapath: ALU, register file, muxes
  - Control: single-cycle control, micro-code
  - Caches & pipelining (will go into these in more detail here)
  - Some familiarity with assembly language
  - Hennessy & Patterson's "Computer Organization and Design"
  - Operating Systems (processes, threads, & virtual memory)

- Significant programming experience
  - Why? assignments require writing code to simulate hardware
  - Not difficult if competent programmer; extremely difficult if not

- **This class will have a gentle ramp, so don't worry.**

Course Components

- Discussions:
  - We will read research papers from literature, including classic and modern works. Participate in discussion on Blackboard one or two papers each week.
- Midterm Exam:
  - One in-class midterm.
  - There will be no final exam.
- Project:
  - Option 1: Hacking the gem5/FlashGPU-sim simulator based on a suggested idea (evaluate new/existing architecture idea using simulation).
  - Option 2: Open ended project of your choice.

Paper Readings/Reviews

- You should participate in class if possible.
  - Bonus points for participation (up to 5% of class grade)
- Online Discussion
  - Meaningfully participate in the discussion of 8 papers during the quarter.
  - Post early and build on other students' participation (most post before corresponding class starts)
- What is meaningful participation (in order):
  - **Novel Idea / Augmentation** – Original idea, no need to be good
  - **Non-trivial Question** (only if you are 24 hours before deadline)
  - **Critique** -- a significant flaw/shortcoming (no methodology)
  - **Impact** -- Impact beyond stated contributions
- Reserve right to take off for redundant answers. :)

Required Texts

- No required *traditional* textbook for this course.
- Required reading will include:
  - Springer Synthesis Lectures – link on website
  - Published papers (available on campus network through ACM and IEEE libraries).

- Optional Textbooks:
  - John Shen and Mikko Lipasti, Modern Processor Design: Fundamentals of Superscalar Processors, McGraw-Hill, 2005.
  - John L. Hennessy and David A. Patterson, Computer Architecture: A Quantitative Approach Morgan Kaufmann Publishers, Sixth Edition.

Midterm Exam

- One exam (no final), testing 2/3 of material each
- Exam Format: In-class exam. Open book, but no internet.
- Exam Content:
  - 3-5 short analysis questions (focusing on concepts)
  - Topic fair game: anything discussed in class or in readings
  - Questions may be about a new aspect of a relevant subject

Project

- In lieu of final exam, we will have a ~4 week project.
  - Start before the midterm exam!
  - Work in teams of 2-3. (preferably not 1 or 4...)
- Why Project?
  - Give you a chance to put into practice some of the ideas
  - Give you freedom to work on something you like
  - Learn tools/approach that can be useful later
- What is a project? Options:
  - Hacksim: Hack an architecture simulator to implement a new architecture or microarchitecture idea, and evaluate it.
  - Open-ended: Propose a research idea and evaluate it using any means (okay to combine with ongoing/concurrent work)
- Deliverables: report (+ source code if applicable)
  - Report should be similar to research papers (but shorter)
  - Bonus if code merged into upstream code base.

Grading Scheme

- Reviews: 30%
- Midterm exam: 30%
- Final project: 40%

Grading philosophy for graduate courses:

- Grade is only an incentive to meet minimal requirements
- Intended curve of the class is from B+ to A.
  - B+ means you tried on everything, but didn't do that well on the exams or project
- Your own motivation will determine whether you get anything meaningful out of this course.

Logistics

- Blackboard:
  - Turning things in and reporting grades.
  - Also used for review discussion (easier for us to grade than piazza).
- Webpage: https://seanzw.github.io/courses/ceng-5410/2026-fall/
  - Post lectures, course schedule, project description, etc...
  - Lecture PDFs are on the course schedule, won't be 100% up to date until just before class.
- Piazza: (https://piazza.com/cuhk.edu.hk/fall2026/ceng5410)
  - This link is on the course webpage
  - Discussions & announcements

Contact Me

- Email:
  - zhengrongwang@cuhk.edu.hk
  - Put [CENG5410] in subject line

- Office Hours: SHB931,
  - Wednesday (3-5pm)