Computer Systems Design

Lesson 1

General terms of computer systems design. Von Neumann architecture

Alexander Antonov, Assoc. Prof., ITMO University

Hangzhou, 2025

ITMO

Course outline

Goal of the course: learn basic and advanced construction principles of optimized computer systems, including HW, SW, and their interplay

Key focus:

- Basic approaches to optimized computer systems design
- Generic data flow, control flow, communication, and memory optimizations
- Common principles and differences in HW and SW design
- von Neumann (vN) and non-vN architectures: advantages and trade-offs
- Classic computer architecture levels: separation of concerns and interplay
- Relevant CPU architecture (RISC-V), HW microarchitecture, bare-metal C/ASM programming

ITMO

Course structure

- Theoretical part (lectures + 4x intermediate tests):
  - **Section 1. Structure and types of computer architecture**
    - **Lecture 1. Basic terms. Von Neumann architecture**
    - **Lecture 2. Non-vN architectures. DSA and SoCs. SW vs. HW design**
    - **Lecture 3. Abstraction levels of programmable computer systems. Hardware and software implementations**
    - **Lecture 4. Instruction set architectures.**
  - **Section 2. Processor architecture. Low-level software development**
    - **Lecture 5. Software toolchain structure**
    - **Lecture 6. Basics of C programming**
    - **Lecture 7. Basics of ASM programming (RISC-V ISA)**
    - **Lecture 8. Operating systems. Memory virtualization**
  - **Section 3. Processor microarchitecture: optimization of control and data flows**
    - **Lecture 9. Basic microarchitectural templates. Multi-cycle and pipelined processors**
    - **Lecture 10. Data flow optimization**
    - **Lecture 11. Control flow optimization**
    - **Lecture 12. Dynamic branch prediction. Open-source OoO CPU cores.**
  - **Section 4. Processor microarchitecture: optimization of communications and memory**
    - **Lecture 13. Basic memory optimizations**
    - **Lecture 14. Caching**
    - **Lecture 15. Basic communications optimization. Shared memory integration. Cache coherence**
    - **Lecture 16. Bus architectures. Networks-on-chip. Open-source NoCs**
- Practical part: 4 labs (optional, to increase score)
- Final Test

ITMO

Recommended literature

- S. Harris, D. Harris. Digital Design and Computer Architecture, RISC-V Edition. 2021.
- J.L. Hennessy, D.A. Patterson - Computer Architecture. A Quantitative Approach. 6th Ed. Morgan Kaufmann. 2019.
- Shen, J.P. & Lipasti, M.H. Modern processor design: fundamentals of superscalar processors. Waveland Press. 2013.
- B.W. Kernighan, D.M. Ritchie. C Programming Language, 2nd Edition. 1988.
- RISC-V technical specifications. URL: https://riscv.org/technical/specifications/

Lecturer information

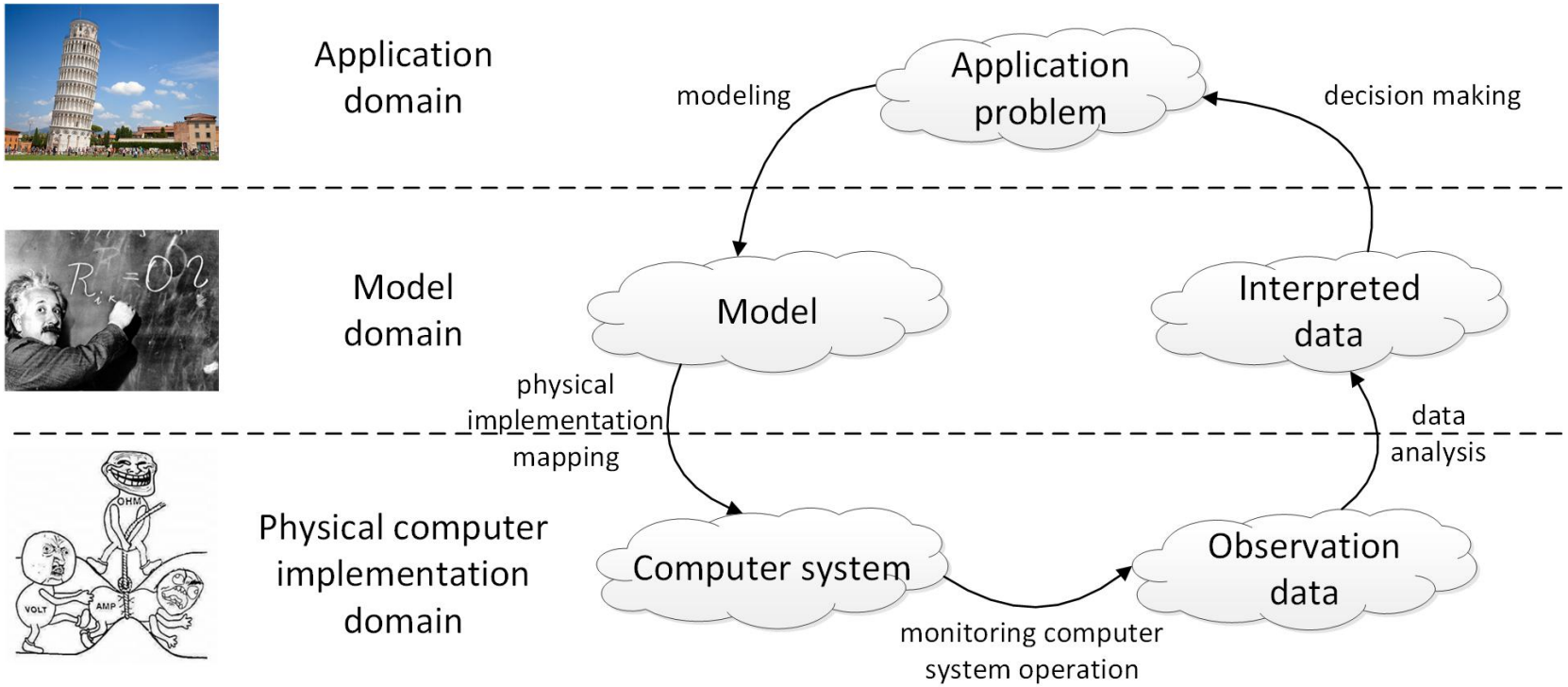
Alexander A. Antonov

- PhD in Informatics and Computer Technologies
- Associate Professor at ITMO University
  mail to antonov@itmo.ru
- Principal FPGA Design Engineer at Huawei

ITMO

General types of computer architectures

Computers as simulation machines



Unique feature of computers:
computer system run is a simulation of some real-life process (usually – of different nature)

Digital electronics for computations

Virtues of digital electronics:

- “cheap” logic abstraction from electronic implementation
- scalability, straightforward assembling of complex systems from simple components

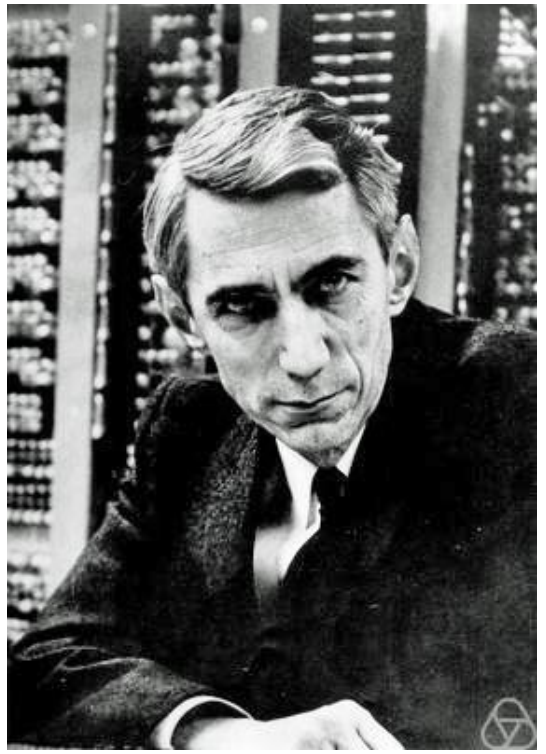
Foundation: Claude Shannon’s digital circuit design theory*

Simulated Boolean algebra with electrical circuits

Enabled “move” from analog electronics to digital

Table I. Analogue Between the Calculus of Propositions and the Symbolic Relay Analysis

Symbol	Interpretation in Relay Circuits	Interpretation in the Calculus of Propositions
X	The circuit X	The proposition X
0	The circuit is closed	The proposition is false
1	The circuit is open	The proposition is true
$X + Y$	The series connection of circuits X and Y	The proposition which is true if either X or Y is true
$X\,Y$	The parallel connection of circuits X and Y	The proposition which is true if both X and Y are true
X'	The circuit which is open when X is closed and closed when X is open	The contradictory of proposition X
=	The circuits open and close simultaneously	Each proposition implies the other



Claude Shannon
1916-2001

C. Shannon. Symbolic Analysis of Relay and Switching Circuits, Master's Thesis, MIT, 1937

Transistor: functional contribution

Invented by W. Shockley, J. Bardeen and W. Brattain (Bell Labs, 1947)

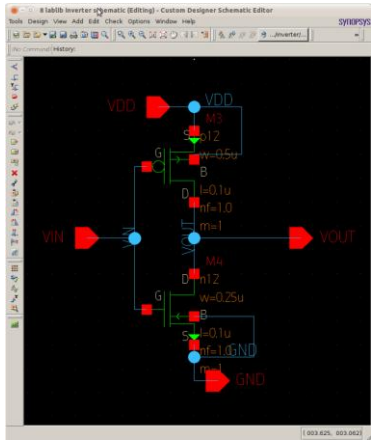
Replaced vacuum tubes (smaller, cheaper, more reliable and efficient)

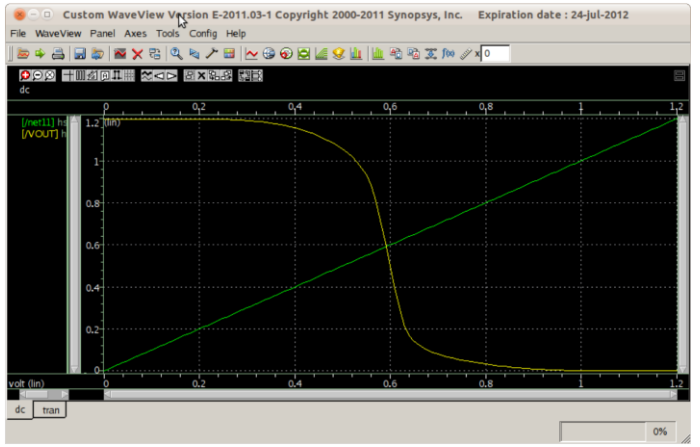
MOSFETs: invented by M. Atalla and D. Kahng (Bell Labs, 1959)

Functional feature: **nonlinear input-to-output function**

allows to build "interesting" logic

Example:
inverter





Enables amplification (analog) and control/switching (digital) functions

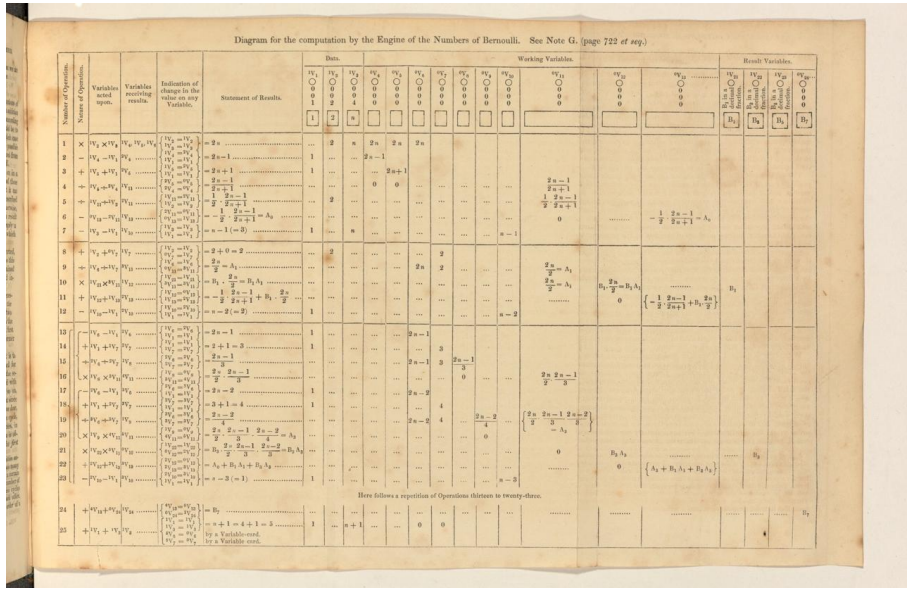
Alternative physical basis: mechanical

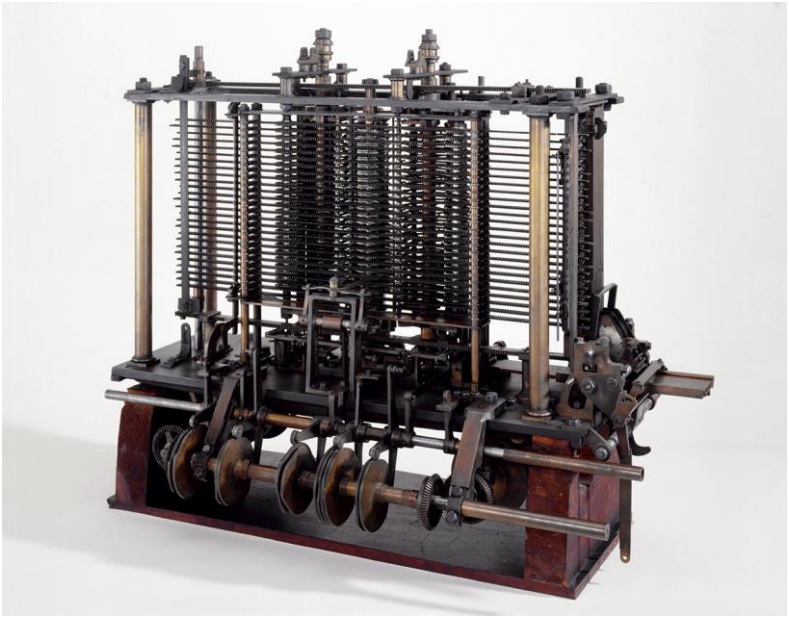
Example: Analytical Engine (Charles Babbage, proposed in 1837)

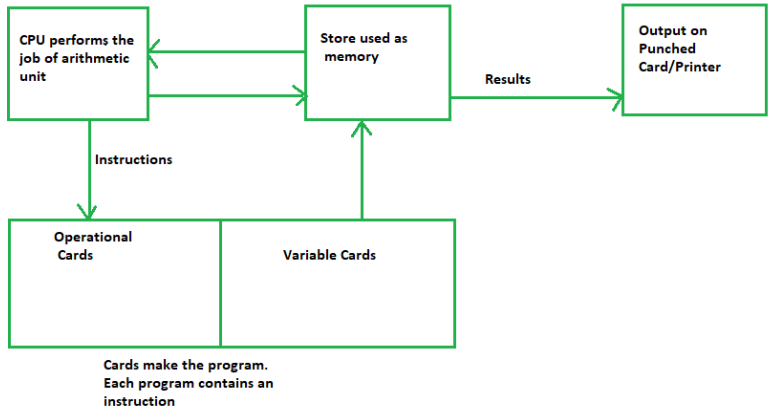
Programmable general-purpose computer

Implemented arithmetic, branches, I/O

First program written by Ada Lovelace (1843)

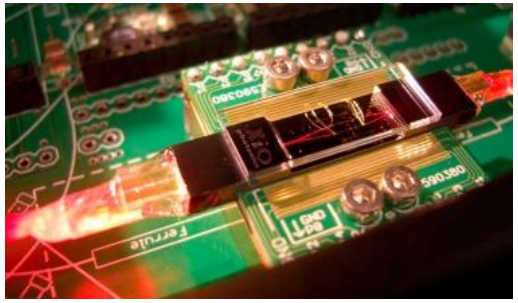






ITMO

Alternative physical basis: optical

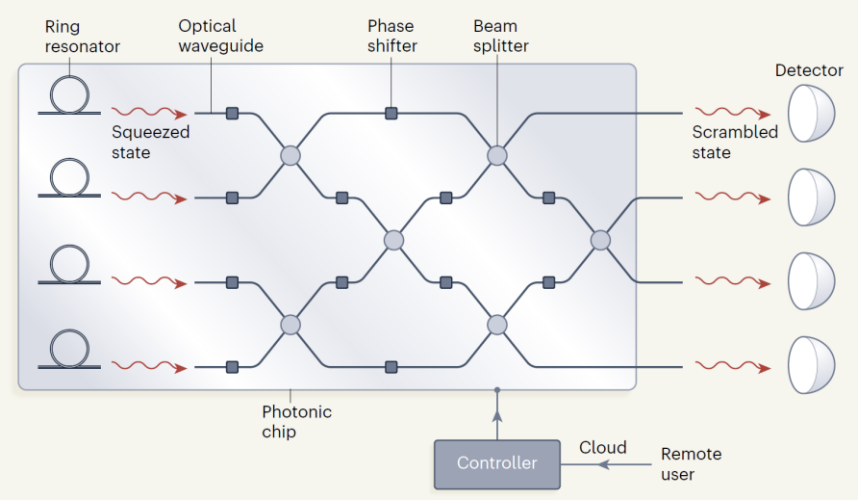


Very fast (speed of light)

Widely used in communications. ***Interest: optical processing***

Basic components:

- Laser: directed light generator (input)
- Waveguide: connectivity
- Detector: output receiver
- Phase shifter: signal change (adjustable)
- Beam splitter: recombines beams of light (adjustable)

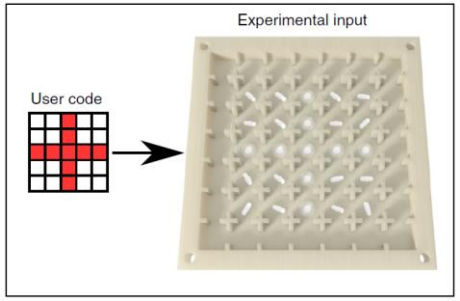


"Control capability":

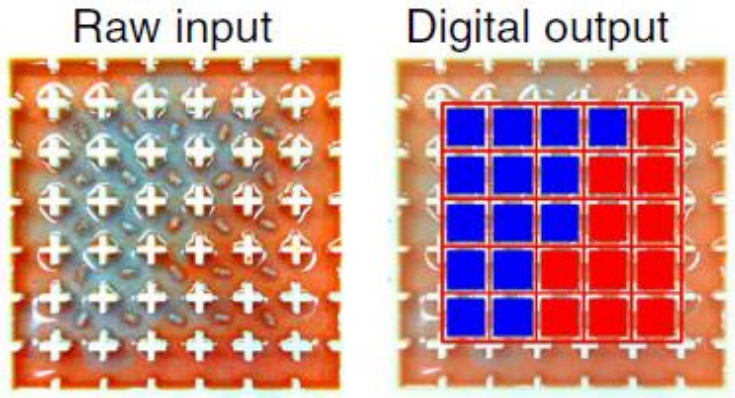
beams in the same phase amplify each other, in opposite phase – suppress

Alternative physical basis: chemical

- Chemical process is going according to ***chemical reactions***
- Data can be modeled e.g. as ***concentration*** of chemicals
- ***Reactions*** model computations
- Promising: ***processing and memory are integrated***, avoiding von Neumann bottleneck
- Issues: imprecise control, statistical nature



Setup



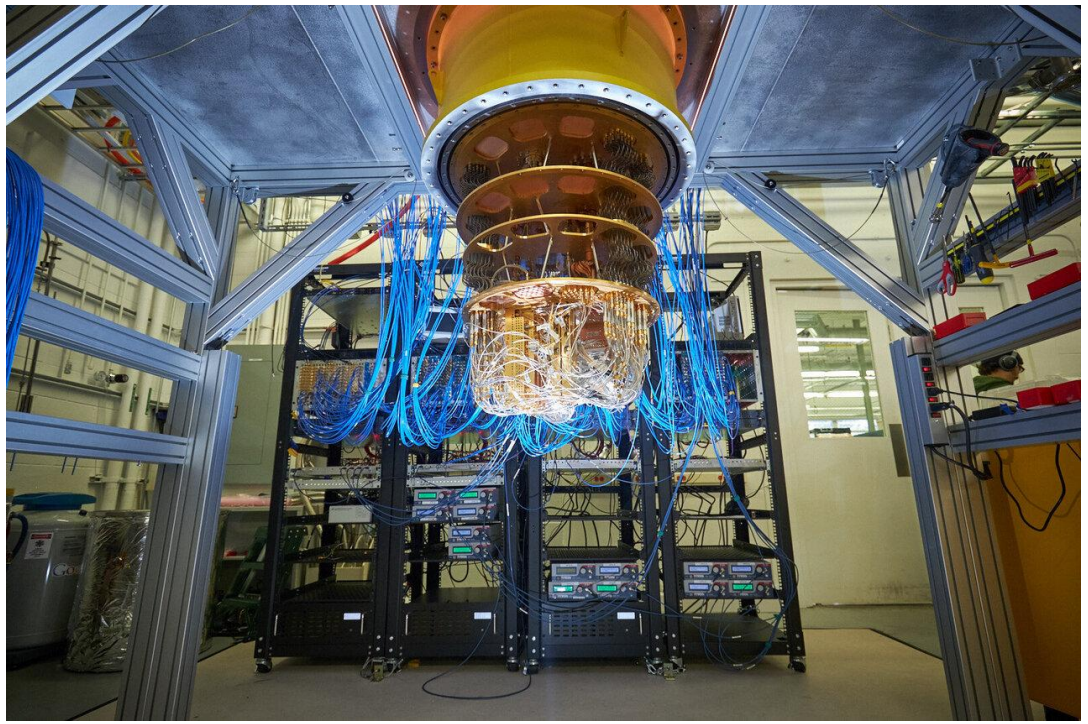
Read-out

Parrilla-Gutierrez, et al. A programmable chemical computer with memory and pattern recognition. Nat. Commun. 11, 1442 (2020).

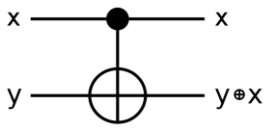
Alternative physical basis: quantum

Mixed discrete/analog architecture:

- works with **qubits** – ***discrete*** elements with "quantum" behavior
- each qubit has ***continuous ("analog")*** probability of being in 0 or 1 state ("superposition")



Example of quantum gate:
Controlled NOT (CNOT)



input		output	
x	y	x	y+x
|0⟩	|0⟩	|0⟩	|0⟩
|0⟩	|1⟩	|0⟩	|1⟩
|1⟩	|0⟩	|1⟩	|1⟩
|1⟩	|1⟩	|1⟩	|0⟩

Performs inversion of target's state probability

Google conducts largest chemical simulation on a quantum computer to date. URL: https://phys.org/news/2020-08-google-largest-chemical-simulation-quantum.html (Aug 28, 2020)

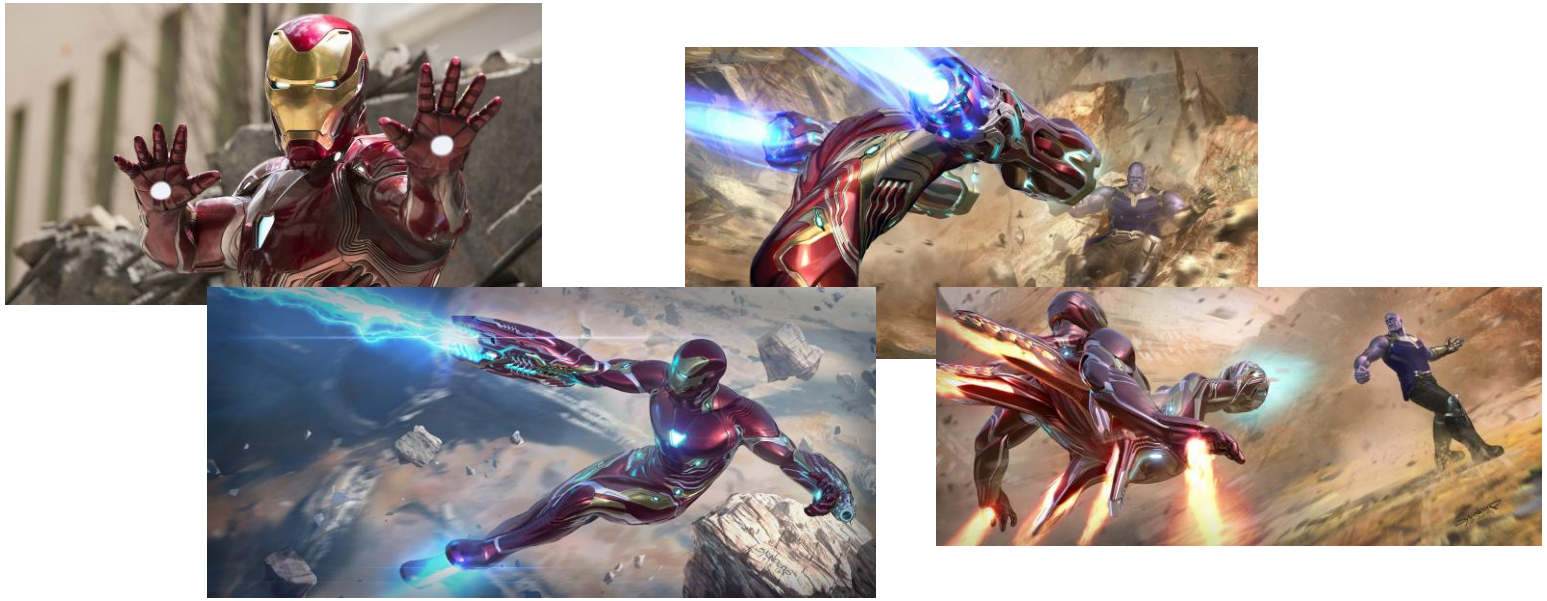
Programmability

Designing unique computer for every single “model” is ***impractical***

economical and complexity reasons

Programmability: re-targeting device to multiple “models”

various software running on fixed hardware



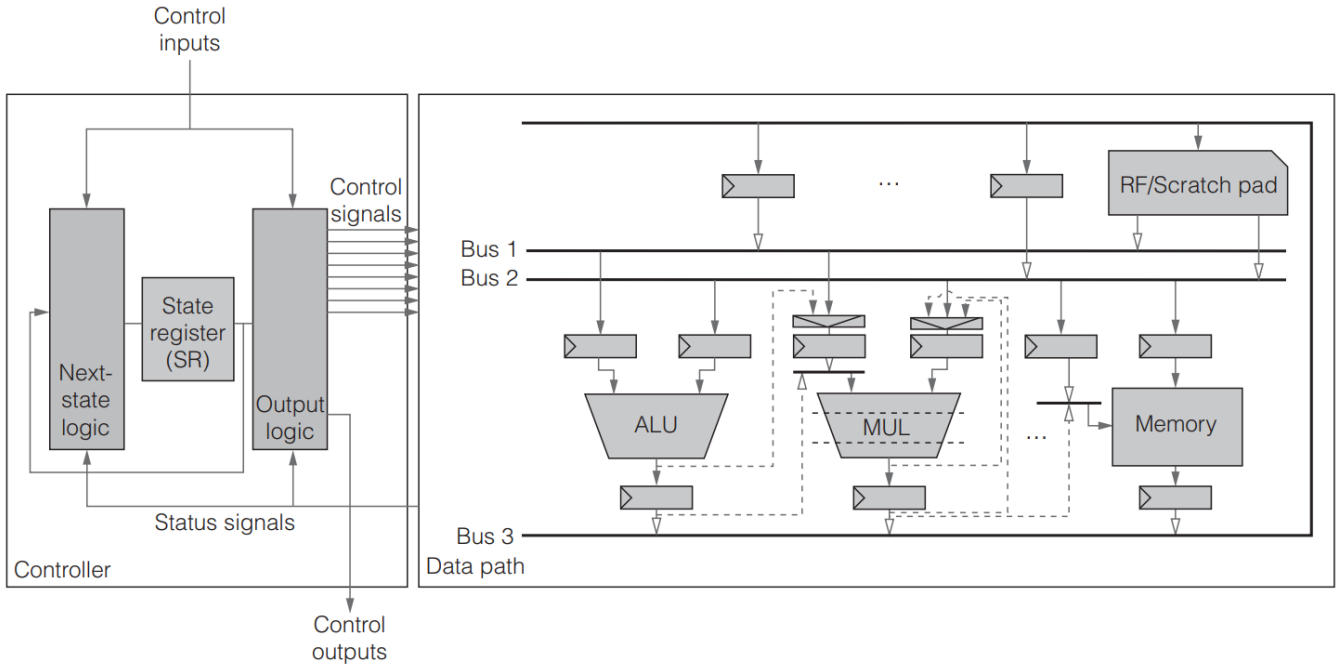
Classes of HW/SW interfaces

- non-programmable («hardwired»)
- programmable instruction flow(s)
  - writing "recipes“ for computations execution
  - General-Purpose and Application-Specific Instruction Processors (GPP/ASIP)
  - *CISC, RISC, VLIW, …: x86, ARM, MIPS, POWER, RISC-V, …*
- programmable data flows
  - coordinating flows of data between processing elements
  - *systolic arrays, dataflow architectures*
- microprogrammable, reconfigurable
  - low-level software management of computations
  - *anti-machine, NISC, TTA, FPGA, CGRA*
- indirectly programmable
  - generalization (training/learning/adapting to) of given examples ("datasets“)
  - *neural networks*

«Hardwired» structures

Base model: **FSMD (FSM+datapath)**

The structure defines execution scheduling on clock cycles (control steps, c-steps)



Almost all hardware structures can be represented as this (on lowest abstraction level)

"Higher-level" architecture is interpretation of structure's operation

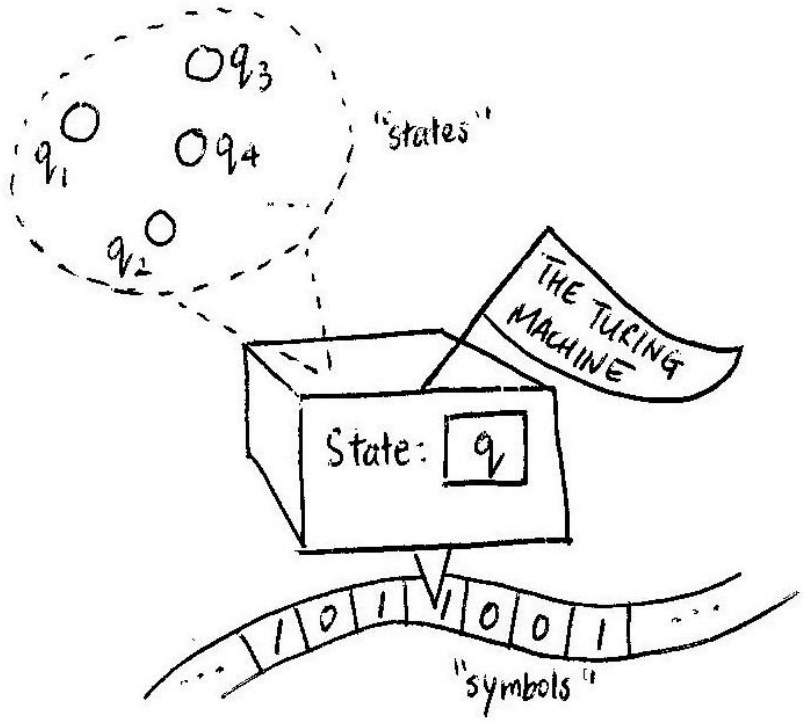
ITMO

Architectures with programmable instruction flows

Turing completeness (1936)

Turing machine - theoretical model of a computer, consisting of

- infinitely extensible tape filled with symbols
- set of finite states
- reader reading one symbol each time
- writer replacing the read symbol on a tape (depending on state)



Proved:

*Given certain (primitive) behaviour of machine **any mathematical function** can be mapped (computed) on the machine (result written on tape)*

*If a machine can imitate Turing machine (**Turing complete**), it can compute all imaginable algorithms*

All further improvements – optimization for efficiency (and convenience)
Limitation: finite memory (not critical)

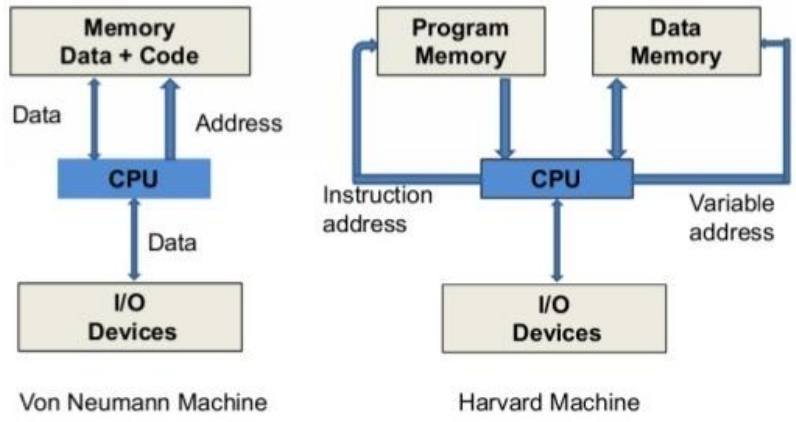
Architecture with programmable instruction flow: Princeton (von Neumann) and Harvard

Key contribution: **decoupling computed algorithm from HW structure**

Added ***Instruction Set Architecture (ISA)*** defining specific computations

Processor works as ***hardware interpreter*** of programs stored in memory

Proposed in 1945 (EDVAC project)*



John von Neumann
1903-1957

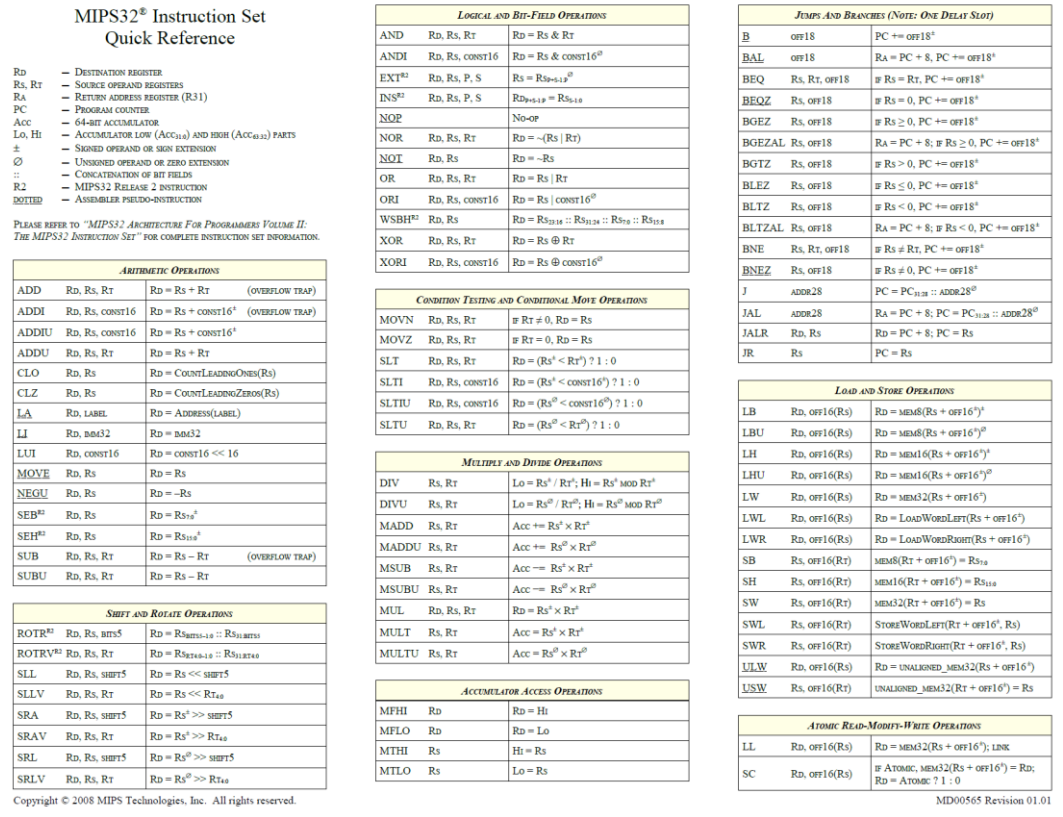
** John von Neumann. First Draft of a Report on the EDVAC. 1945*

Example of vN CPU architecture: MIPS

Executes ***sequential thread*** of instructions controlled by **program counter** (**instruction pointer**)

Typical instructions types:

- arithmetic/logic
- memory load/stores
- branch control



MIPS32® Instruction Set Quick Reference

Rd	– Destination register
Rs, Rt	– Source operand registers
Ra	– Return address register (R31)
PC	– Program counter
Acc	– 64-bit accumulator
Lo, Hi	– Accumulator low ($Acc_{31:0}$) and high ($Acc_{63:32}$) parts
±	– Signed operand or sign extension
∅	– Unsigned operand or zero extension
::	– Concatenation of bit fields
R2	– MIPS32 Release 2 instruction
DOTTED	– Assembler pseudo-instruction

Please refer to *"MIPS32 Architecture For Programmers Volume II: The MIPS32 Instruction Set"* for complete instruction set information.

Arithmetic Operations

ADD	Rd, Rs, Rt	Rd = Rs + Rt (overflow trap)
ADDI	Rd, Rs, const16	Rd = Rs + const16± (overflow trap)
ADDIU	Rd, Rs, const16	Rd = Rs + const16±
ADDU	Rd, Rs, Rt	Rd = Rs + Rt
CLO	Rd, Rs	Rd = CountLeadingOnes(Rs)
CLZ	Rd, Rs	Rd = CountLeadingZeros(Rs)
LA	Rd, label	Rd = Address(label)
LI	Rd, imm32	Rd = imm32
LUI	Rd, const16	Rd = const16 << 16
MOVE	Rd, Rs	Rd = Rs
NEGU	Rd, Rs	Rd = –Rs
SEB[R2]	Rd, Rs	Rd = $Rs_{7:0}$±
SEH[R2]	Rd, Rs	Rd = $Rs_{15:0}$±
SUB	Rd, Rs, Rt	Rd = Rs – Rt (overflow trap)
SUBU	Rd, Rs, Rt	Rd = Rs – Rt

Shift and Rotate Operations

ROTR[R2]	Rd, Rs, bits5	Rd = $Rs_{bits5-1:0}$ :: $Rs_{31:bits5}$
ROTRV[R2]	Rd, Rs, Rt	Rd = $Rs_{Rt4:0-1:0}$ :: $Rs_{31:Rt4:0}$
SLL	Rd, Rs, shift5	Rd = Rs << shift5
SLLV	Rd, Rs, Rt	Rd = Rs << $Rt_{4:0}$
SRA	Rd, Rs, shift5	Rd = Rs± >> shift5
SRAV	Rd, Rs, Rt	Rd = Rs± >> $Rt_{4:0}$
SRL	Rd, Rs, shift5	Rd = Rs∅ >> shift5
SRLV	Rd, Rs, Rt	Rd = Rs∅ >> $Rt_{4:0}$

Logical and Bit-Field Operations

AND	Rd, Rs, Rt	Rd = Rs & Rt
ANDI	Rd, Rs, const16	Rd = Rs & const16∅
EXT[R2]	Rd, Rs, P, S	Rs = $Rs_{p+s-1:p}$∅
INS[R2]	Rd, Rs, P, S	$Rd_{p+s-1:p}$ = $Rs_{s-1:0}$
NOP		No-op
NOR	Rd, Rs, Rt	Rd = ~(Rs | Rt)
NOT	Rd, Rs	Rd = ~Rs
OR	Rd, Rs, Rt	Rd = Rs | Rt
ORI	Rd, Rs, const16	Rd = Rs | const16∅
WSBH[R2]	Rd, Rs	Rd = $Rs_{23:16}$ :: $Rs_{31:24}$ :: $Rs_{7:0}$ :: $Rs_{15:8}$
XOR	Rd, Rs, Rt	Rd = Rs ⊕ Rt
XORI	Rd, Rs, const16	Rd = Rs ⊕ const16∅

Condition Testing and Conditional Move Operations

MOVN	Rd, Rs, Rt	if Rt ≠ 0, Rd = Rs
MOVZ	Rd, Rs, Rt	if Rt = 0, Rd = Rs
SLT	Rd, Rs, Rt	Rd = (Rs± < Rt±) ? 1 : 0
SLTI	Rd, Rs, const16	Rd = (Rs± < const16±) ? 1 : 0
SLTIU	Rd, Rs, const16	Rd = (Rs∅ < const16∅) ? 1 : 0
SLTU	Rd, Rs, Rt	Rd = (Rs∅ < Rt∅) ? 1 : 0

Multiply and Divide Operations

DIV	Rs, Rt	Lo = Rs± / Rt±; Hi = Rs± mod Rt±
DIVU	Rs, Rt	Lo = Rs∅ / Rt∅; Hi = Rs∅ mod Rt∅
MADD	Rs, Rt	Acc += Rs± × Rt±
MADDU	Rs, Rt	Acc += Rs∅ × Rt∅
MSUB	Rs, Rt	Acc –= Rs± × Rt±
MSUBU	Rs, Rt	Acc –= Rs∅ × Rt∅
MUL	Rd, Rs, Rt	Rd = Rs± × Rt±
MULT	Rs, Rt	Acc = Rs± × Rt±
MULTU	Rs, Rt	Acc = Rs∅ × Rt∅

Accumulator Access Operations

MFHI	Rd	Rd = Hi
MFLO	Rd	Rd = Lo
MTHI	Rs	Hi = Rs
MTLO	Rs	Lo = Rs

Jumps And Branches (Note: One Delay Slot)

B	off18	PC += off18±
BAL	off18	Ra = PC + 8; PC += off18±
BEQ	Rs, Rt, off18	if Rs = Rt, PC += off18±
BEQZ	Rs, off18	if Rs = 0, PC += off18±
BGEZ	Rs, off18	if Rs ≥ 0, PC += off18±
BGEZAL	Rs, off18	Ra = PC + 8; if Rs ≥ 0, PC += off18±
BGTZ	Rs, off18	if Rs > 0, PC += off18±
BLEZ	Rs, off18	if Rs ≤ 0, PC += off18±
BLTZ	Rs, off18	if Rs < 0, PC += off18±
BLTZAL	Rs, off18	Ra = PC + 8; if Rs < 0, PC += off18±
BNE	Rs, Rt, off18	if Rs ≠ Rt, PC += off18±
BNEZ	Rs, off18	if Rs ≠ 0, PC += off18±
J	addr28	PC = $PC_{31:28}$ :: addr28∅
JAL	addr28	Ra = PC + 8; PC = $PC_{31:28}$ :: addr28∅
JALR	Rd, Rs	Rd = PC + 8; PC = Rs
JR	Rs	PC = Rs

Load and Store Operations

LB	Rd, off16(Rs)	Rd = mem8(Rs + off16±)±
LBU	Rd, off16(Rs)	Rd = mem8(Rs + off16±)∅
LH	Rd, off16(Rs)	Rd = mem16(Rs + off16±)±
LHU	Rd, off16(Rs)	Rd = mem16(Rs + off16±)∅
LW	Rd, off16(Rs)	Rd = mem32(Rs + off16±)
LWL	Rd, off16(Rs)	Rd = LoadWordLeft(Rs + off16±)
LWR	Rd, off16(Rs)	Rd = LoadWordRight(Rs + off16±)
SB	Rs, off16(Rt)	mem8(Rt + off16±) = $Rs_{7:0}$
SH	Rs, off16(Rt)	mem16(Rt + off16±) = $Rs_{15:0}$
SW	Rs, off16(Rt)	mem32(Rt + off16±) = Rs
SWL	Rs, off16(Rt)	StoreWordLeft(Rt + off16±, Rs)
SWR	Rs, off16(Rt)	StoreWordRight(Rt + off16±, Rs)
ULW	Rd, off16(Rs)	Rd = unaligned_mem32(Rs + off16±)
USW	Rs, off16(Rt)	unaligned_mem32(Rt + off16±) = Rs

Atomic Read-Modify-Write Operations

LL	Rd, off16(Rs)	Rd = mem32(Rs + off16±); link
SC	Rd, off16(Rs)	if Atomic, mem32(Rs + off16±) = Rd; Rd = Atomic ? 1 : 0

MD00565 Revision 01.01

Architecture/microarchitecture decoupling

Model 30

Model 50

Model 65

Introduction

The design philosophies of the new general-purpose machine organization for the IBM System/360 are discussed in this paper.† In addition to showing the architecture* of the new family of data processing systems, we point out the various engineering problems encountered in attempts to make the system design compatible, at the program bit level, for large and small models. The compatibility was to extend not only to models of any size but also to their various applications—scientific, commercial, real-time, and so on.

*The term *architecture* is used here to describe the attributes of a system as seen by the programmer, i.e., the conceptual structure and functional behavior, as distinct from the organization of the data flow and controls, the logical design, and the physical implementation.

† Additional details concerning the architecture, engineering design, programming, and application of the IBM System/360 will appear in a series of articles in the *IBM Systems Journal*.

The
the new
technol
efficient
The re
problen
sions m
storage

Design

The ne
signs th
of the c
improve
develop

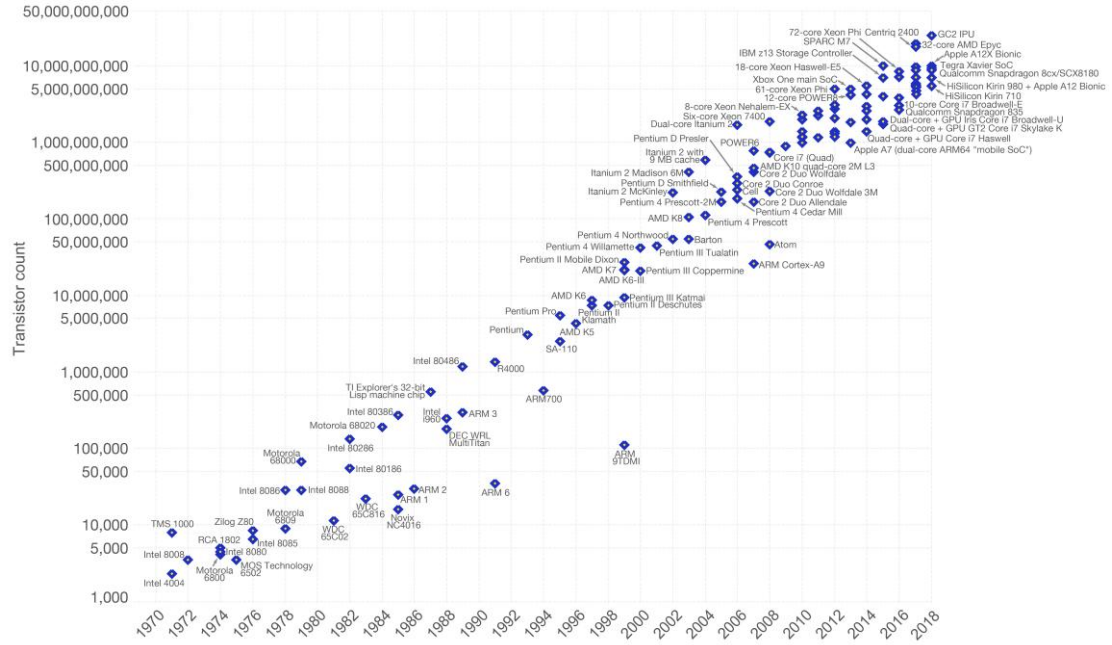
- Term "***architecture***" for CPUs is commonly used to define **programmer's view**
  instruction encoding, registers, modes, …
- "***Microarchitecture***" defines **hardware internals**
- First well-known explicit decoupling: **IBM System/360** (1965-1978)
- Offered multiple models with **binary compatibility**
  From low-end to high-end:
  Models 30, 40, 50, 60, 62, 70

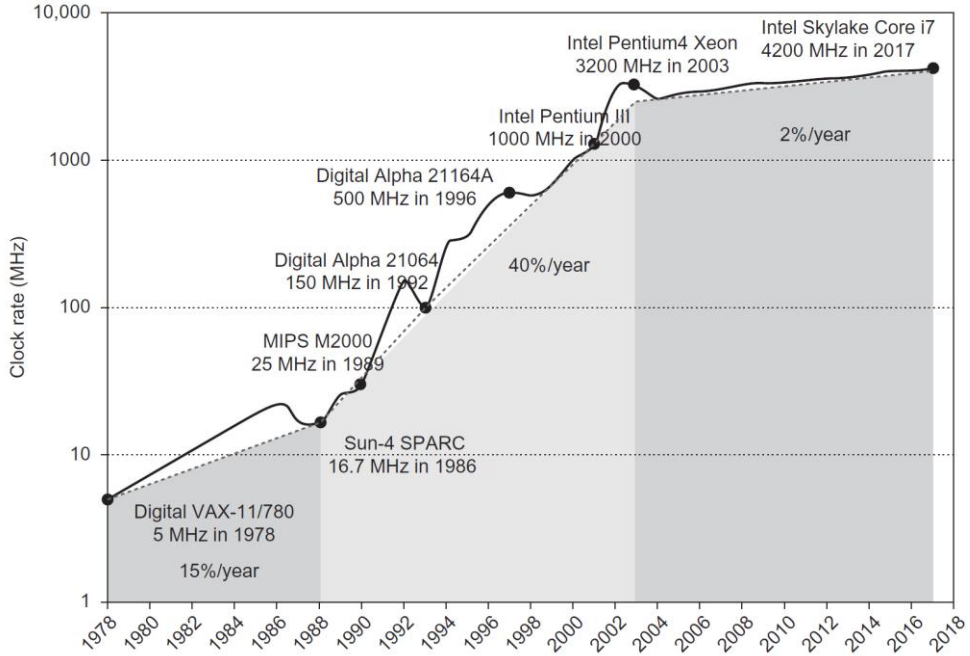
G.M. Amdahl, G.A. Blaauw, F.P. Brooks, Architecture of the IBM System/360, IBM Journal of Research and Development, Volume 8, Issue 2, 1964

Motivation for computer architecture research

Modern technology offers enormous amount of fast transistors:

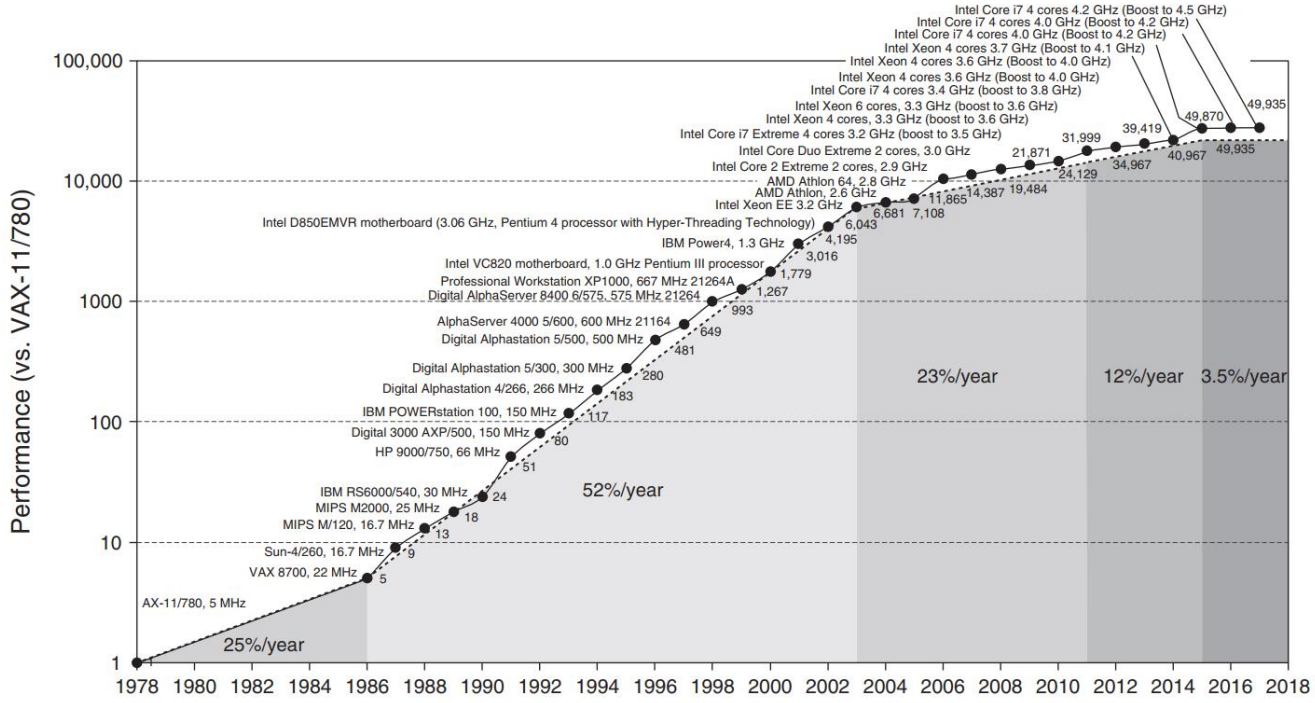
- ***ten of billions*** on chip, ***~1,000,000x (since 1978)***
- ***multi-GHz*** frequencies, ***~1,000x***





J. Hennessy and D. Patterson, Computer Architecture: A Quantitative Approach, 6th edition, Morgan Kauffmann, 2019

Motivation for computer architecture research



CPU performance: ~50,000x (only)

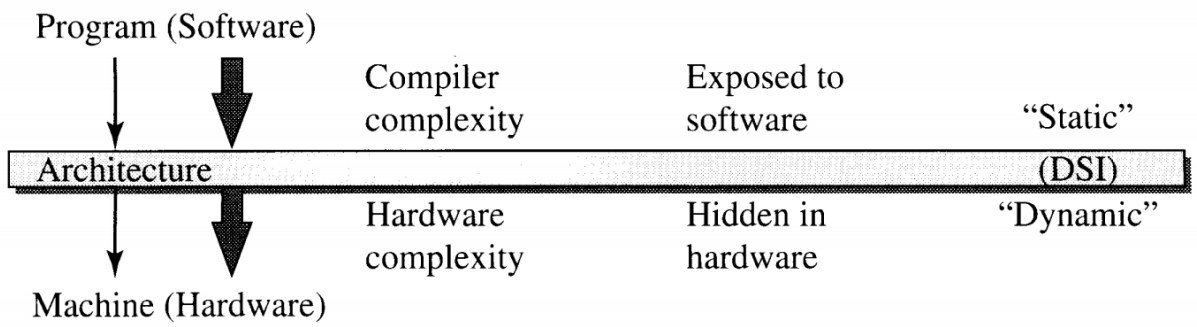
CPU performance per transistor per Hz: 4-5 orders of magnitude lost!

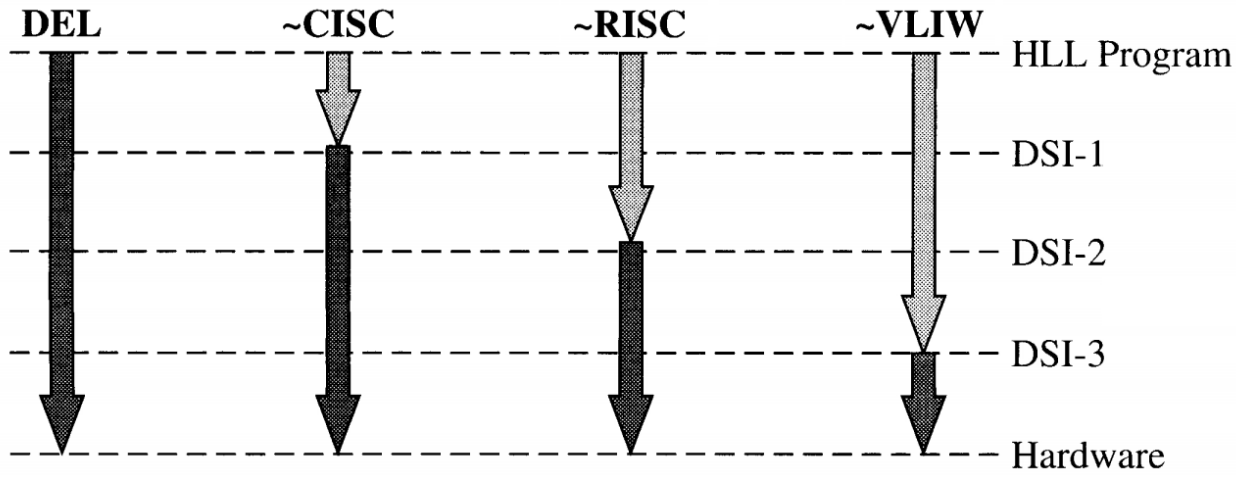
Actual problem: how to make HW resources efficiently collaborate on application problems?

J. Hennessy and D. Patterson, Computer Architecture: A Quantitative Approach, 6th edition, Morgan Kauffmann, 2019

Level of Dynamic/Static Interface (DSI)

Defines distribution of complexity of computational process management between HW and SW

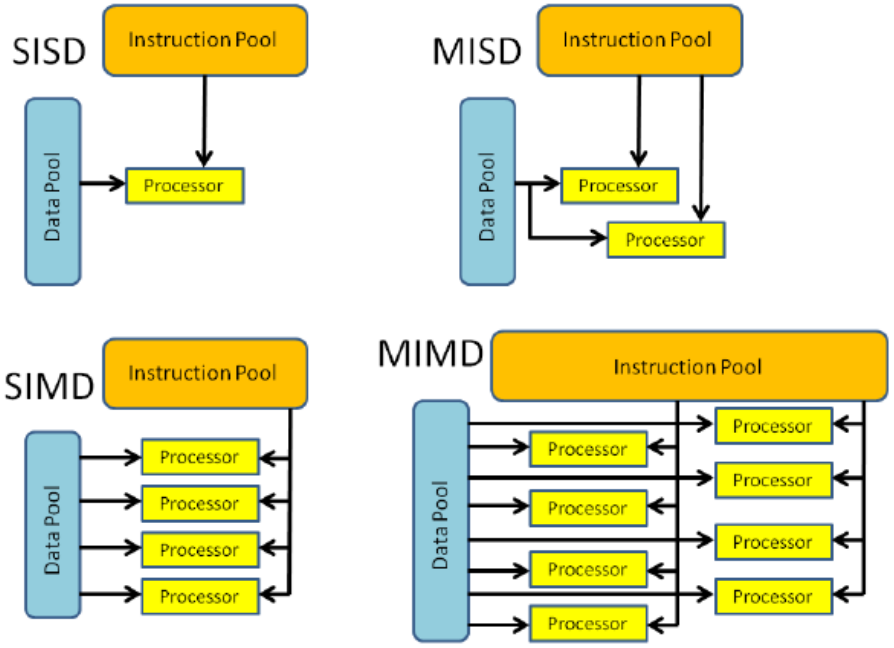


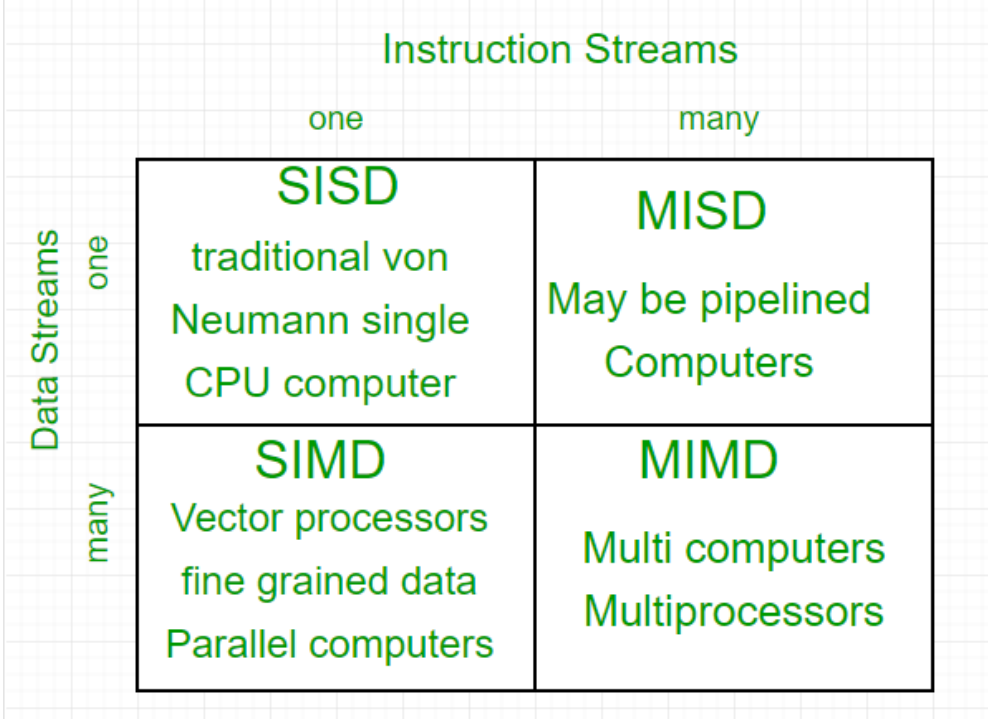


Shen, J. P., & Lipasti, M. H. Modern processor design: fundamentals of superscalar processors. Waveland Press, 2013

Instruction-level parallelism: Flynn's taxonomy (1966)

M.J. Flynn: computer scientist (Stanford University)





CISC vs. RISC: design priorities

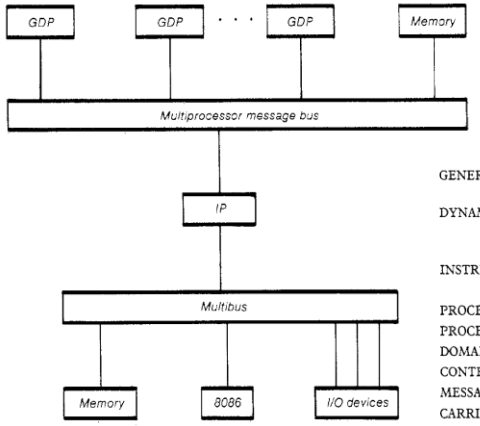
CISC

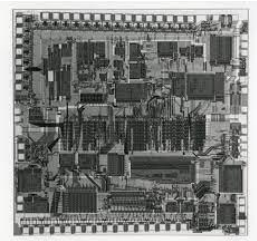
- ***making processor "smart"***
- instructions for high-level processing
- lots of formats, addressing schemes
- few registers (with specialization)
- difficult ILP extraction
- convenient manual programming

RISC

- ***making processor "sprint"***
- instructions do elementary operations
- unified formats and addressing schemes
- many general-purpose registers
- convenient ILP extraction
- compiler automates programming

“Capability/object-based” computer system: Intel iAPX 432 Micromainframe (1981)





GENERIC SEGMENT	basic storage for data and access descriptors (capabilities)
DYNAMIC SEGMENT	storage segment created by a programmer-defined type manager
INSTRUCTION SEGMENT	segment containing executable code
PROCESS	basic unit of scheduling
PROCESSOR	432 GDP or IP
DOMAIN	module or package
CONTEXT	dynamic state for a procedure invocation
MESSAGE PORT	interprocess communication object
CARRIER	extension of a message used to queue it to a port
TYPE DEFINITION	object containing information about a specific object type
TYPE CONTROL	object permitting creation of specific object types
STORAGE RESOURCE	source of primary memory for object storage allocation
OBJECT TABLE	mapping table of object descriptors

Table 9-1: Intel 432 System Object Types

- Planned to be mainline Intel architecture
- First 32-bit Intel architecture
- First hardware multitasking and memory management
- Architectural (hardware) support for high-level functions
  process scheduling, interprocess communication, storage allocation, fault tolerance

Fail: took 6 years of development (8086: 2 years)
Fail: several times slower than competition

D. Henry M. Levy. Capability-Based Computer Systems. Digital Press, 1984.

RISC concept contribution

1970s: RISC concept emerged (Stanford, Berkeley, IBM)

Key observation: ***most used instructions are simple***

Key design principle: ***optimize for the most common case***

simple instructions that can be easily pipelined

1991: research published comparing RISC (MIPS M2000) and CISC (VAX 8700) based on (approx.) similar tech and gate count *

Table 2: RISC factors

benchmark	instruc. ratio	CPI MIPS	CPI VAX	CPI ratio	RISC factor
spice2g6	2.48	1.80	8.02	4.44	1.79
matrix300	2.37	3.06	13.81	4.51	1.90
nasa7	2.10	3.01	14.95	4.97	2.37
fpppp	3.88	1.45	15.16	10.45	2.70
tomcatv	2.86	2.13	17.45	8.18	2.86
doduc	2.65	1.67	13.16	7.85	2.96
espresso	1.70	1.06	5.40	5.09	2.99
eqntott	1.08	1.25	4.38	3.51	3.25
li	1.62	1.10	6.53	5.97	3.69
geo. mean	2.17	1.71	9.87	5.77	2.66

	min	geo. mean	max
VAX CPI	5.4	9.9	17.4
MIPS CPI	1.1	1.7	3.1
CPI ratio (VAX/MIPS)	3.5	5.8	10.4
Inst. ratio (MIPS/VAX)	1.1	2.2	3.9
RISC factor	1.8	2.7	3.7

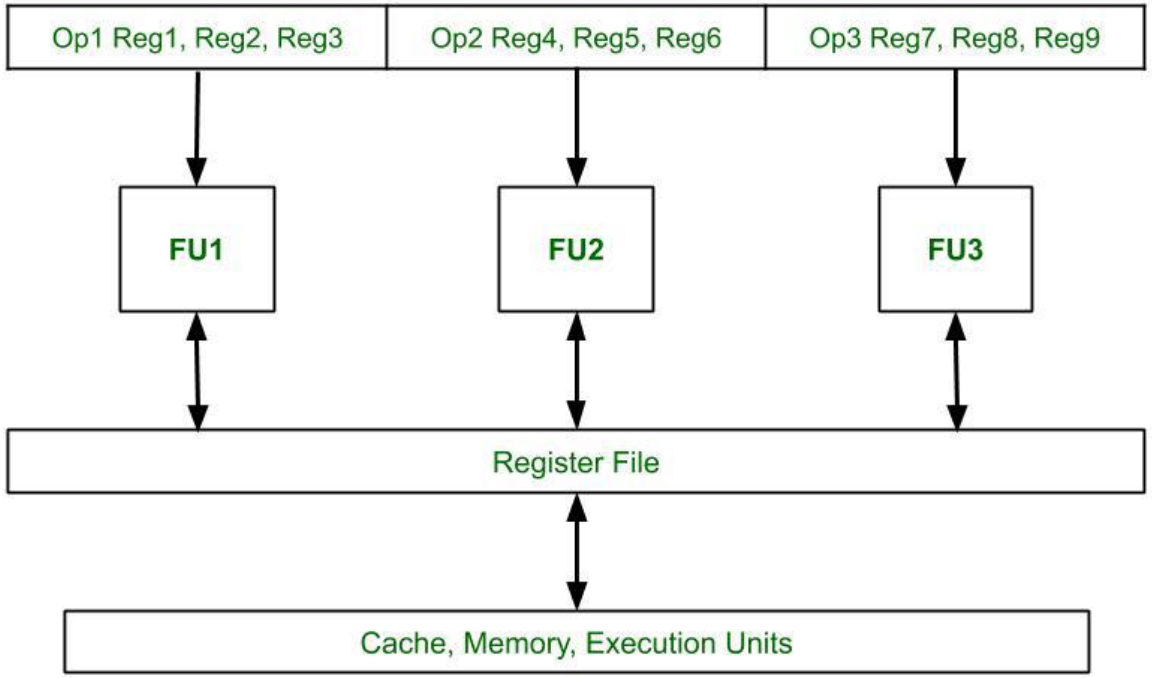
"Refined" change of ISA design principle -> 2.7x acceleration

** D. Bhandarkar and D. W. Clark. Performance from architecture: comparing a RISC and a CISC with similar hardware organization. SIGARCH Comput. Archit. News 19, 2 (Apr. 1991), 310–319.*

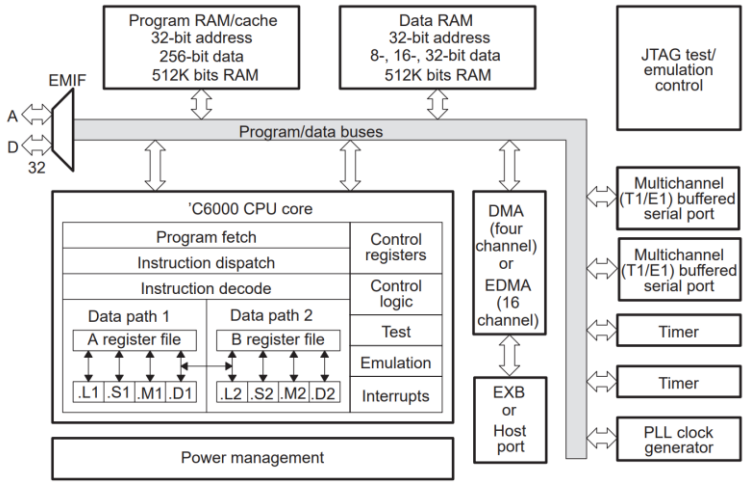
ITMO

Very Long Instruction Word (VLIW) architectures

- Wide instruction contains several slots (basic operations) that are executed in parallel
- All operations in slots synchronously traverse through processor execution core
- Implements statically (software) scheduled parallelism
- VLIW: can be considered as further “adapting ISA to actual hardware”
  similar to CISC -> RISC



VLIW example: Texas Instruments TMS320C62x/C67x DSP



- Datapath: 8 FUs (operate in parallel), 2x similar sets with 4x FUs
- No data dependency checking within instruction word done in hardware during run time

```
Inst1
||  Inst2
||  Inst3
||  Inst4
||  Inst5
||  Inst6
Inst7
```

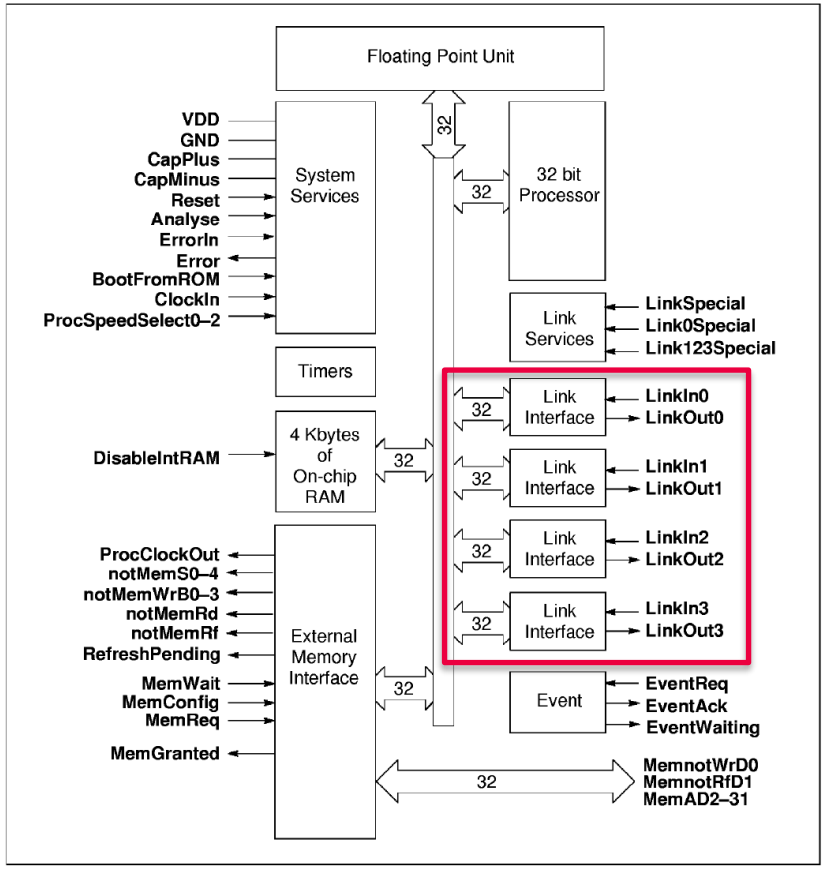
These five instructions run in parallel with the first instruction.

```
;*************************************
;* FUNCTION NAME: myfunc *
;*************************************
myfunc:
;** -----------------------------------
      B .S1          $C$L10
||    SUB .L2X    A4,10,B5
||    STW .D2T2   B3,*SP--(16)
      CMPGTU .L2  B5,7,B0
||    STW .D2T1   A4,*+SP(12)
||    MV .S2X     A4,B4
  [ B0] BNOP .S1  $C$L9,3
      ; BRANCH OCCURS {$C$L10} ; |6|
;** -----------------------------------
```

TI TMS320 DSP series – commonly used in embedded DSP applications

TMS320C6000 Technical Brief (SPRU197D), Texas Instruments, Feb. 1999

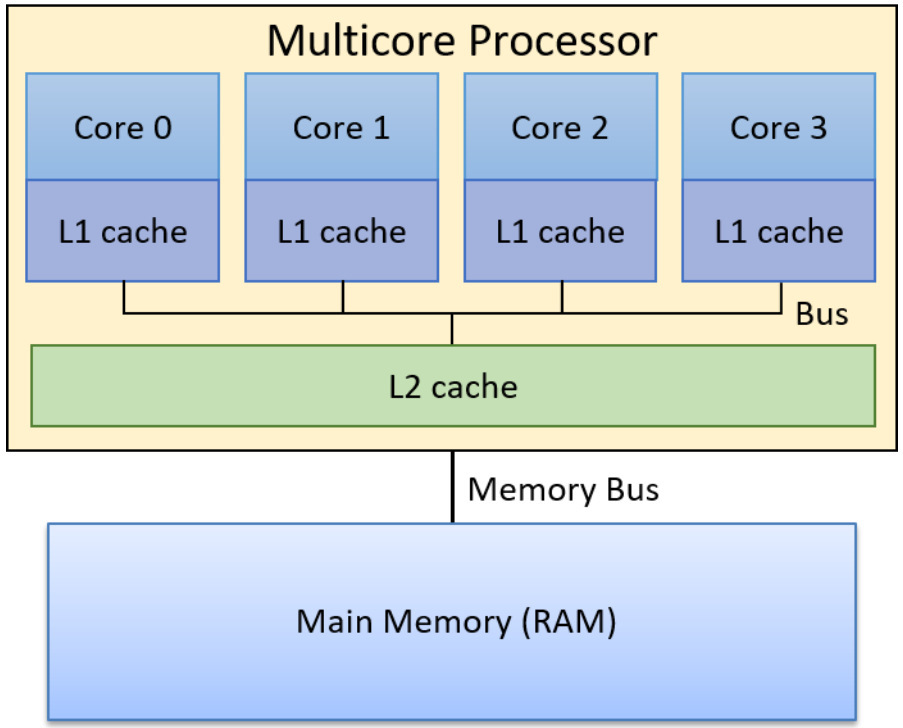
Transputers (1980s)



Inmos (STMicroelectronics) IMS T850

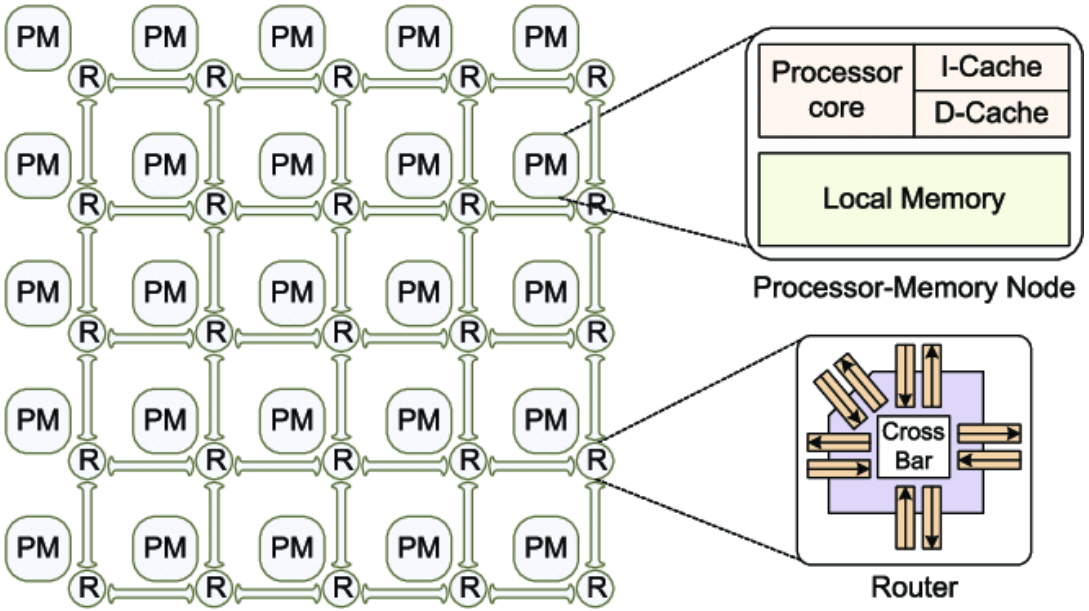
- "**Trans**istor/com**puter**": basic block to construct high-performance parallel systems
- Each transputer has CPU core, integrated memory, and several links for connection to other transputers
- Small and cheap (planned)
- Intended to perform diverse tasks (CPU, coprocessor, I/O controller, etc.)
- Focused on (potentially) infinite scaling capabilities
- Demonstrated useful principles commonly used till nowadays

Multicore processors



- Nonlinear performance scaling with increasing CPU complexity -> implement multiple CPU cores
- Usually integrated by memory subsystem with cache coherency (see further lectures)
- Can provide near-linear performance scaling, but not for all workloads (Amdahl's Law, see further lectures)

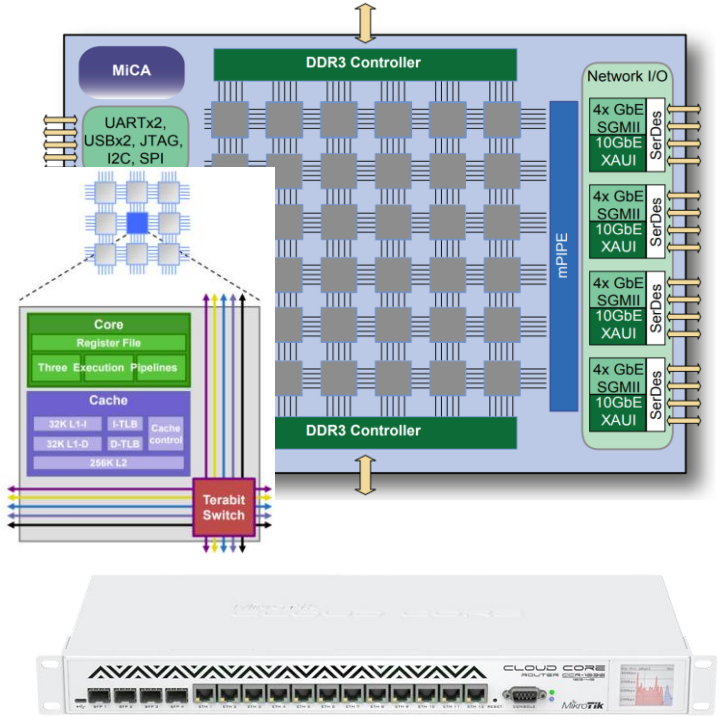
Manycore processors



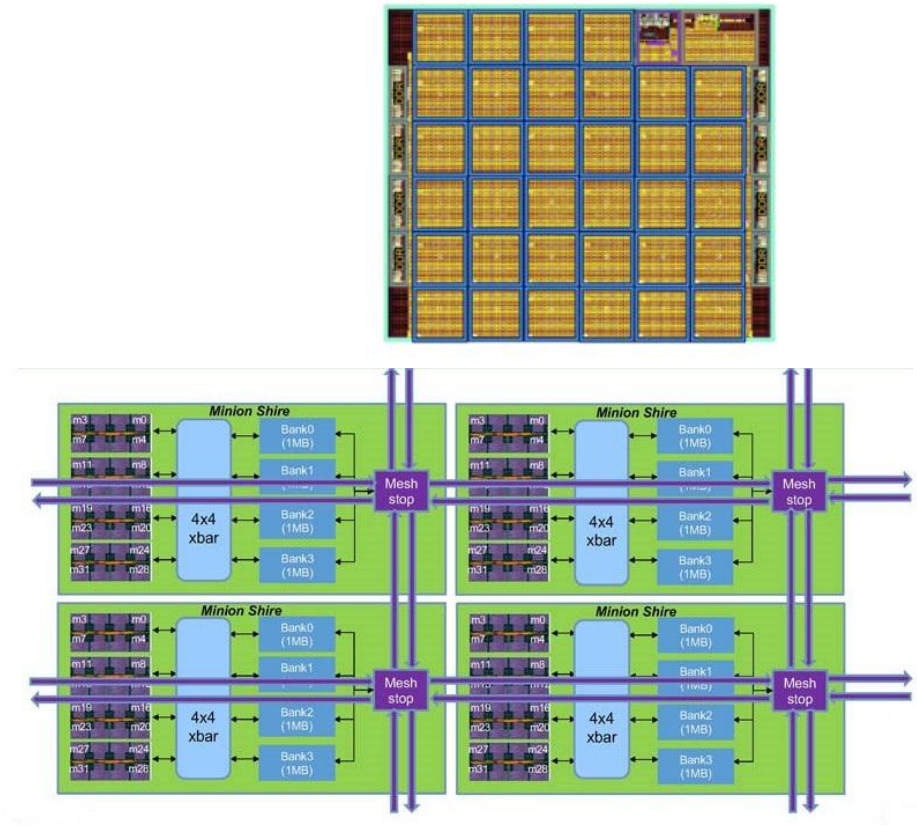
- Too many cores -> too much complexity for shared memory subsystem with cache coherence
- Typical design: mesh of CPU cores (usually – RISC) with local memory and NoC interface
- Direct communications (cache coherence usually not implemented)

Yang Li et al. Round-trip latency prediction for memory access fairness in mesh-based many-core architectures. IEICE Electronics Express 11, Dec. 2014

Manycore examples



Tilera TILE-Gx8036 (2011)
32 CPU cores
Target applications:
network and multimedia



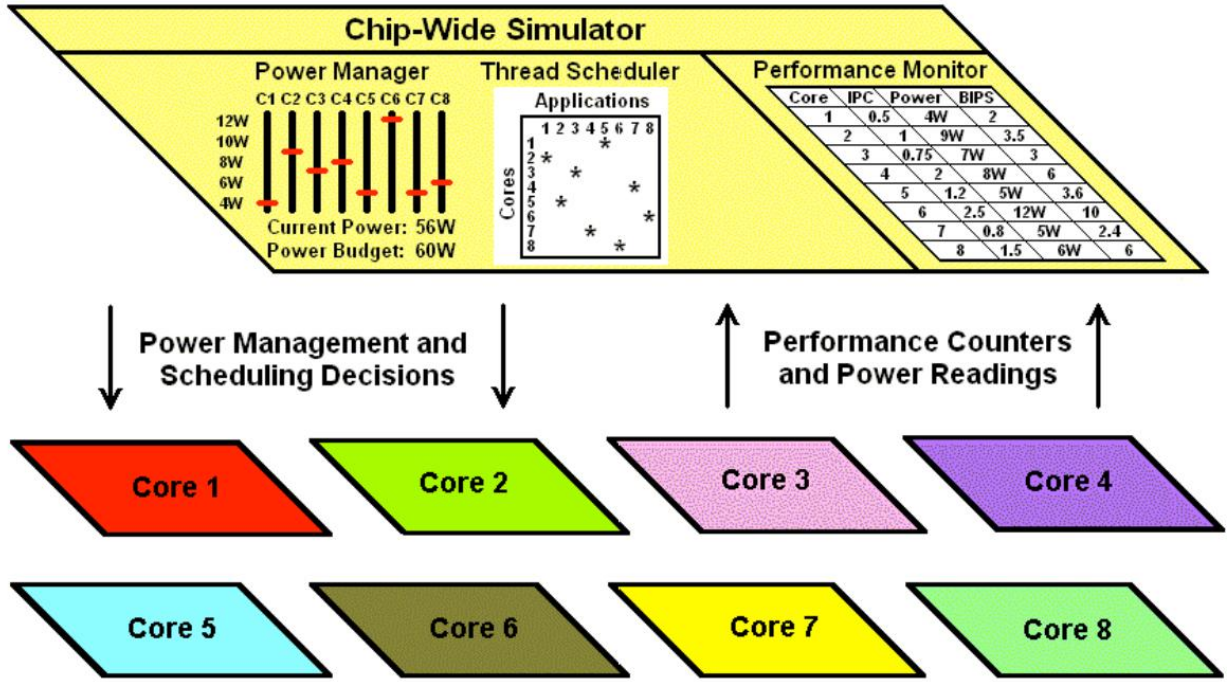
Esperanto Technologies ET-SoC-1 (2021)
~1100 RISC-V CPU cores
Target application:
neural networks

ITMO

Limitations of manycore architectures

- Poor single-thread performance
- Communication overhead
- Software-exposed spatial planning
- Specialized tooling

  schedulers, libraries, simulators, profilers, etc.



J.A. Winter et al. Scalable thread scheduling and global power management for heterogeneous many-core architectures, 19th International Conference on Parallel Architectures and Compilation Techniques (PACT), 2010

Thank you for the lesson!

Alexander Antonov, Assoc. Prof., antonov@itmo.ru

Hangzhou, 2025