CS 31007 Autumn 2021

COMPUTER ORGANIZATION AND ARCHITECTURE

Instructors

Rajat Subhra Chakraborty (*RSC*)

Bhargab B. Bhattacharya (*BBB*)

Lecture #4:

Hardware-Software Interface: Instruction Set Architecture (ISA)

16 August 2021

Indian Institute of Technology Kharagpur

Computer Science and Engineering

Building a Computer System ……

Technology scaling (Moore's Law)

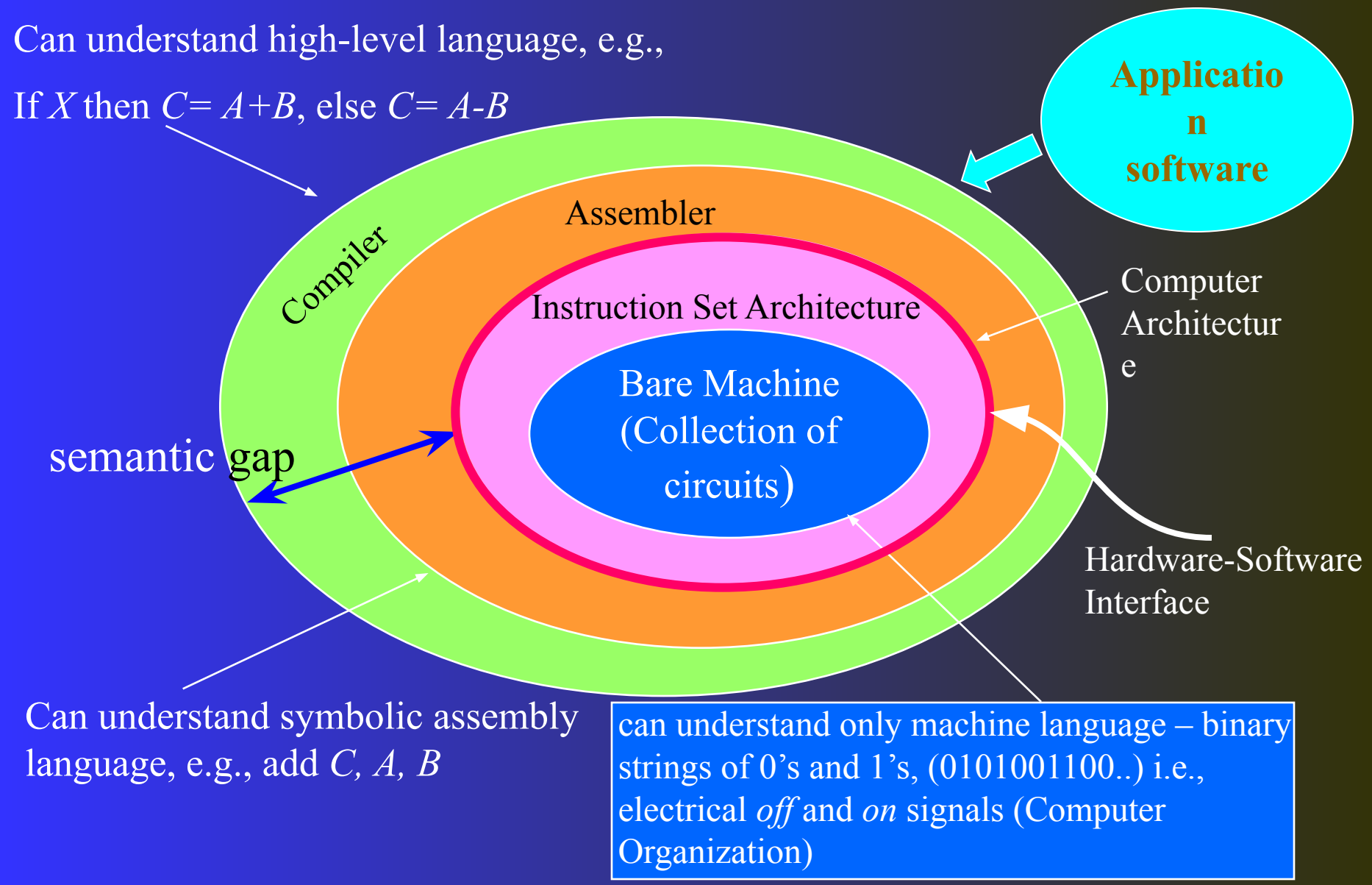
1970: 4004 μP

2300 transistors

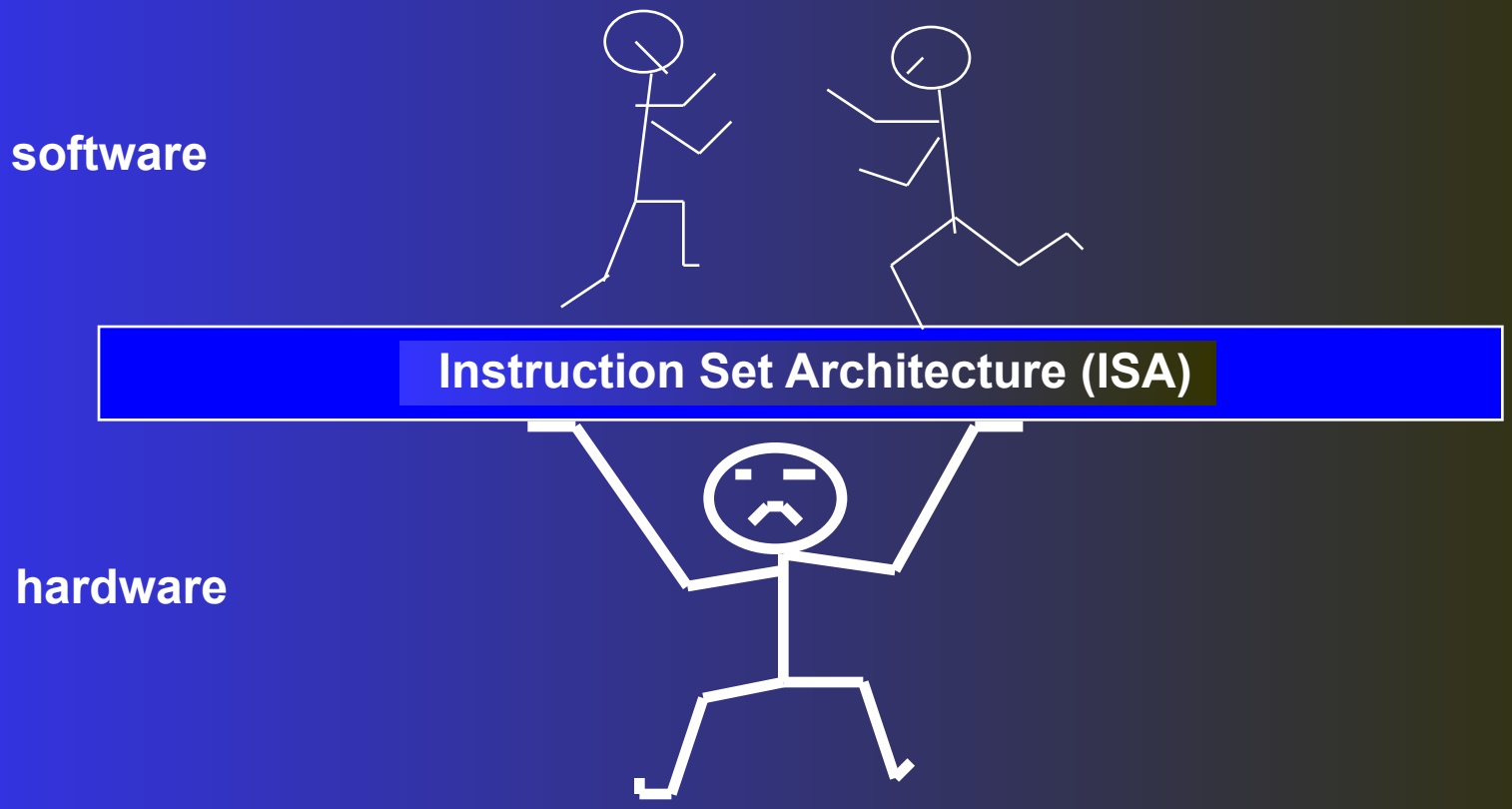
AMD Ryzen 9 5900X processor (2021); ~ 4 billion Transistors

Computing technology has *scaled exponentially* in performance for over half a century. This scaling was mostly the result of a virtuous cycle, where computers based on a new technology is used to create newer technology (Bootstrapping of CAD tools)

Computing System Hierarchy



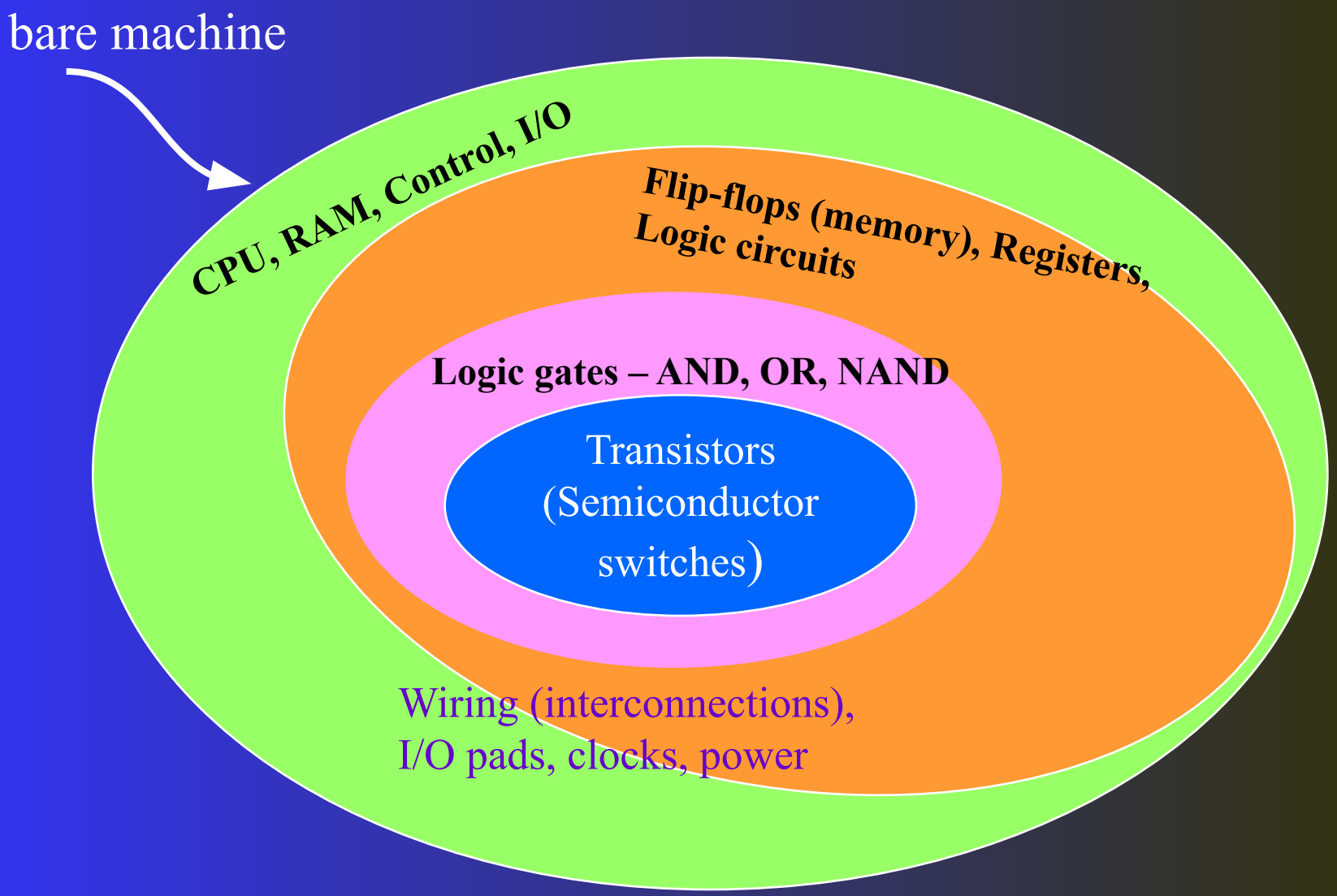
Hardware-Software Interface



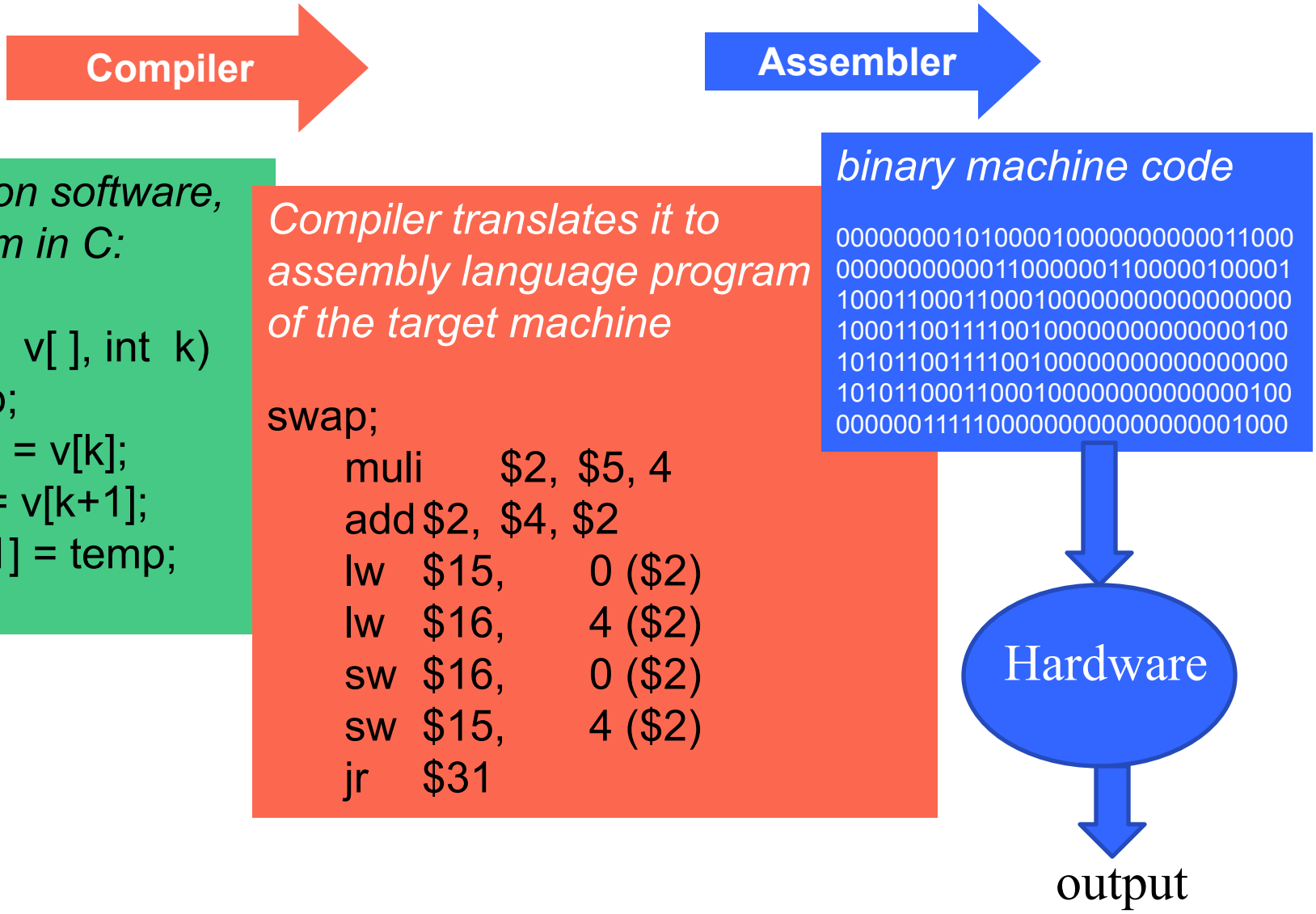
ISA: Collective attributes of the machine-language instruction set; it defines the hardware-software interface

Courtesy: Patterson and Hennessy

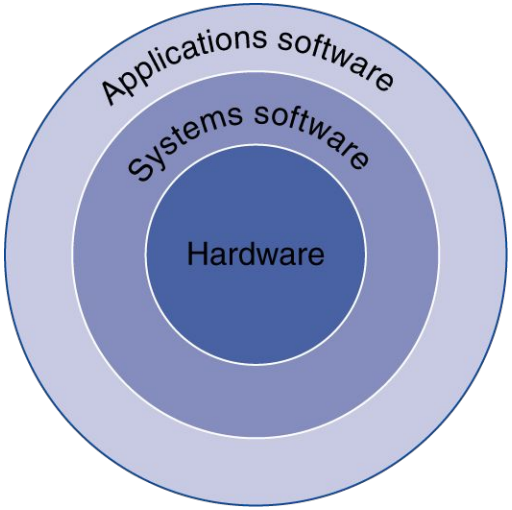
Hardware Hierarchy



How a program is executed by a computer



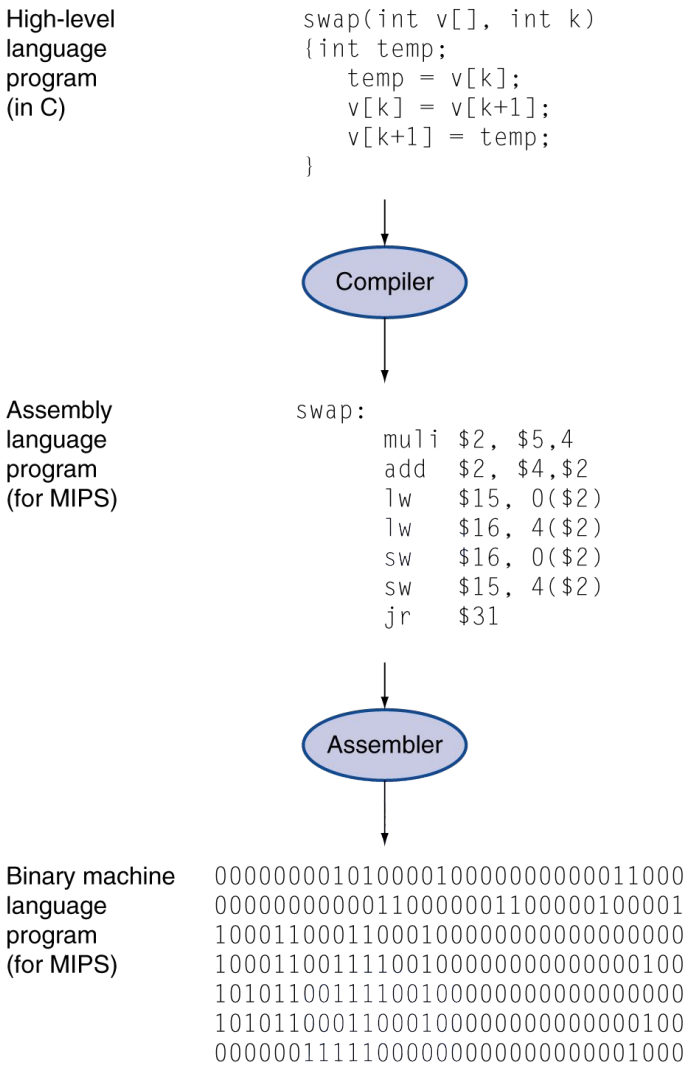
Below Your Program



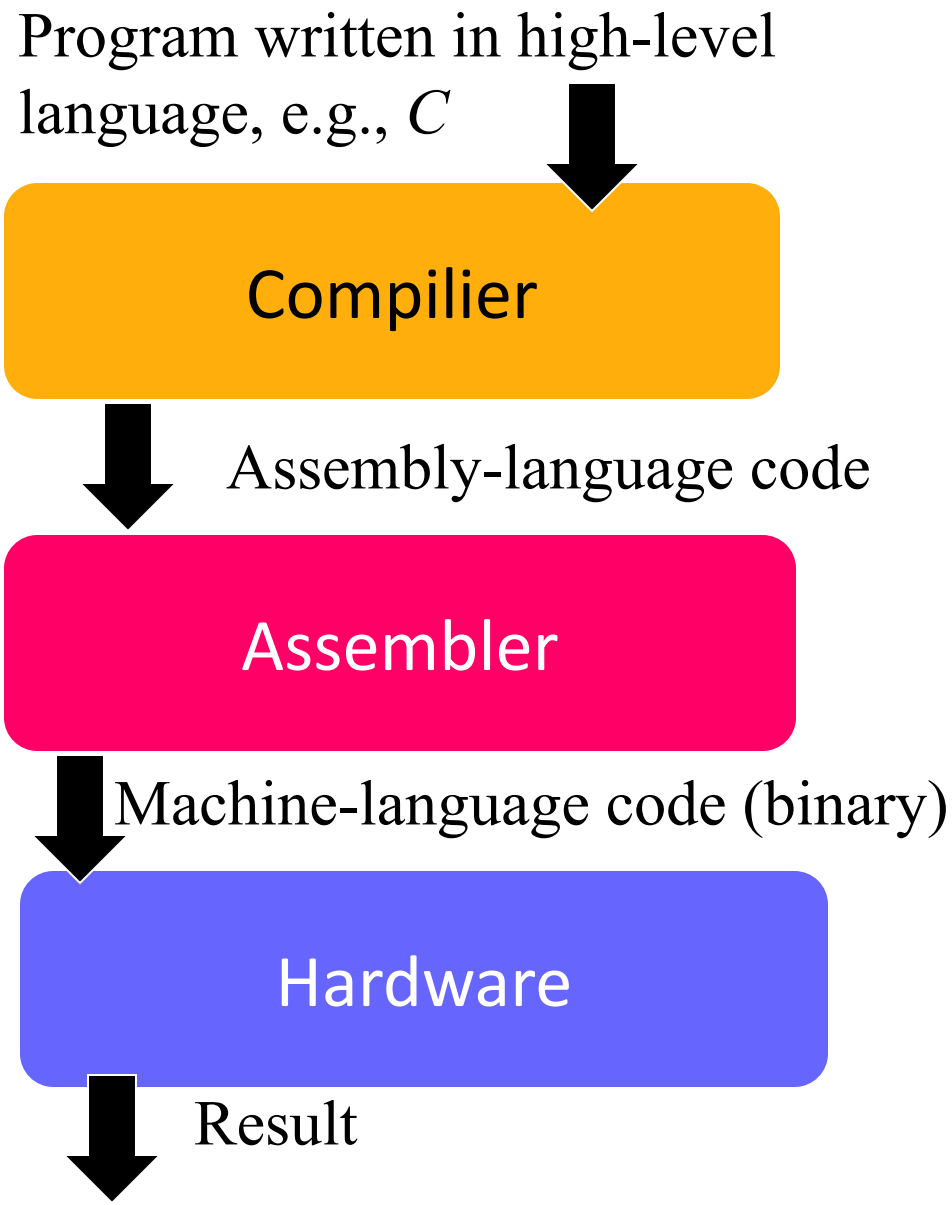
- Application software
  - Written in high-level language
- System software
  - Compiler: translates HLL code to machine code
  - Operating System: service code
    - Handling input/output
    - Managing memory and storage
    - Scheduling tasks & sharing resources
- Hardware
  - Processor, memory, I/O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

Levels of Program Code

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



Program Execution



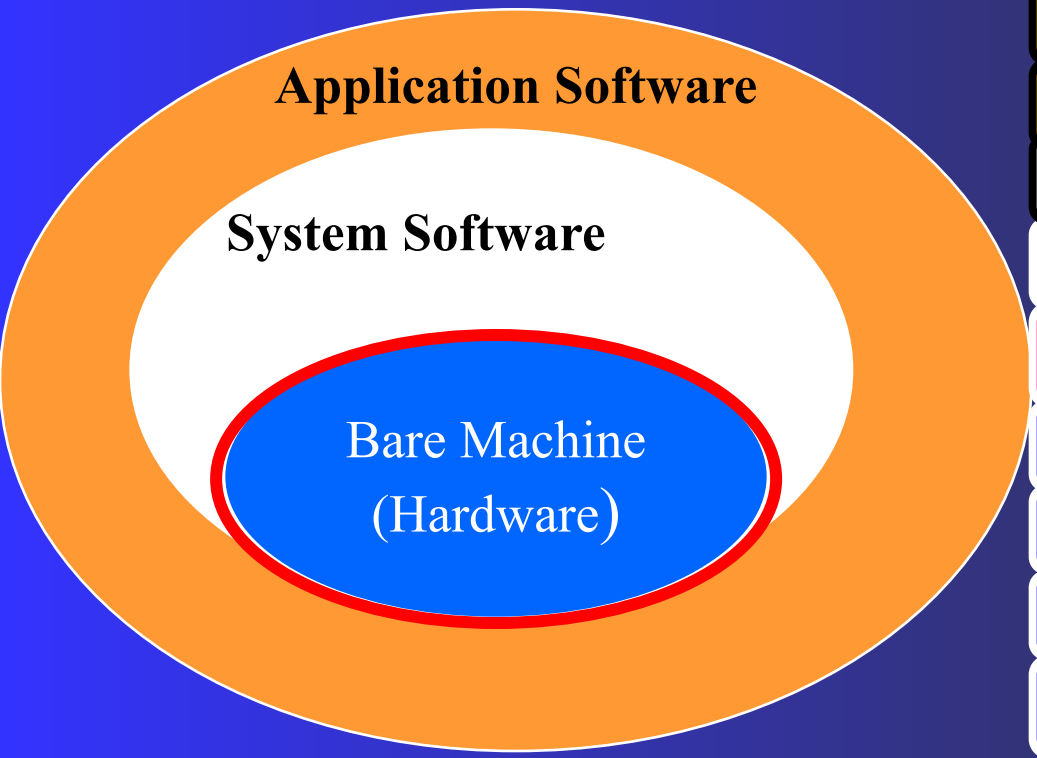
Questions:

Where is the compiler running?

Who compiles the compiler?

How is the Assembler running?

System Hierarchy

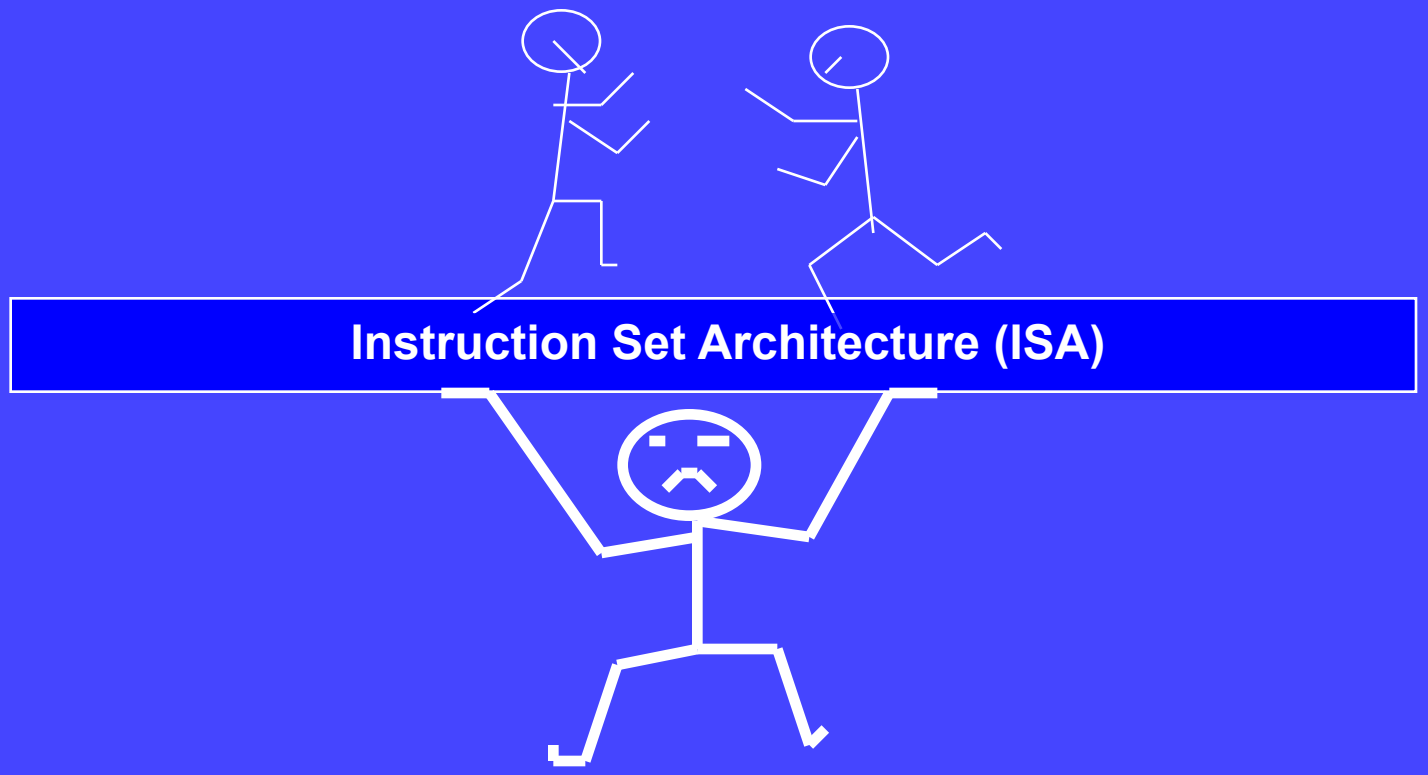


- Applications
- Algorithms
- Programming Language
- Operating Systems, Compilers
- Instruction Set Architecture (ISA)
- Microarchitecture
- Gates/Register-Transfer Level (RTL)
- Circuits – Gates, Flip-flops
- Devices - Transistors
- Physics

Computer Organization *versus* Computer Architecture

Computer Architecture (top-down view)

The view of a computer as perceived by software designers

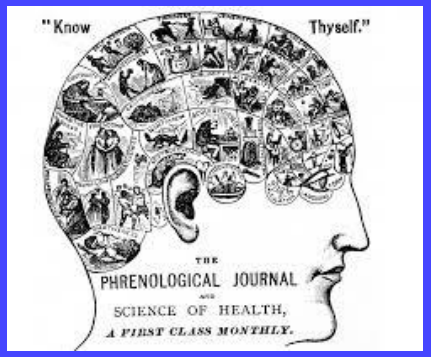


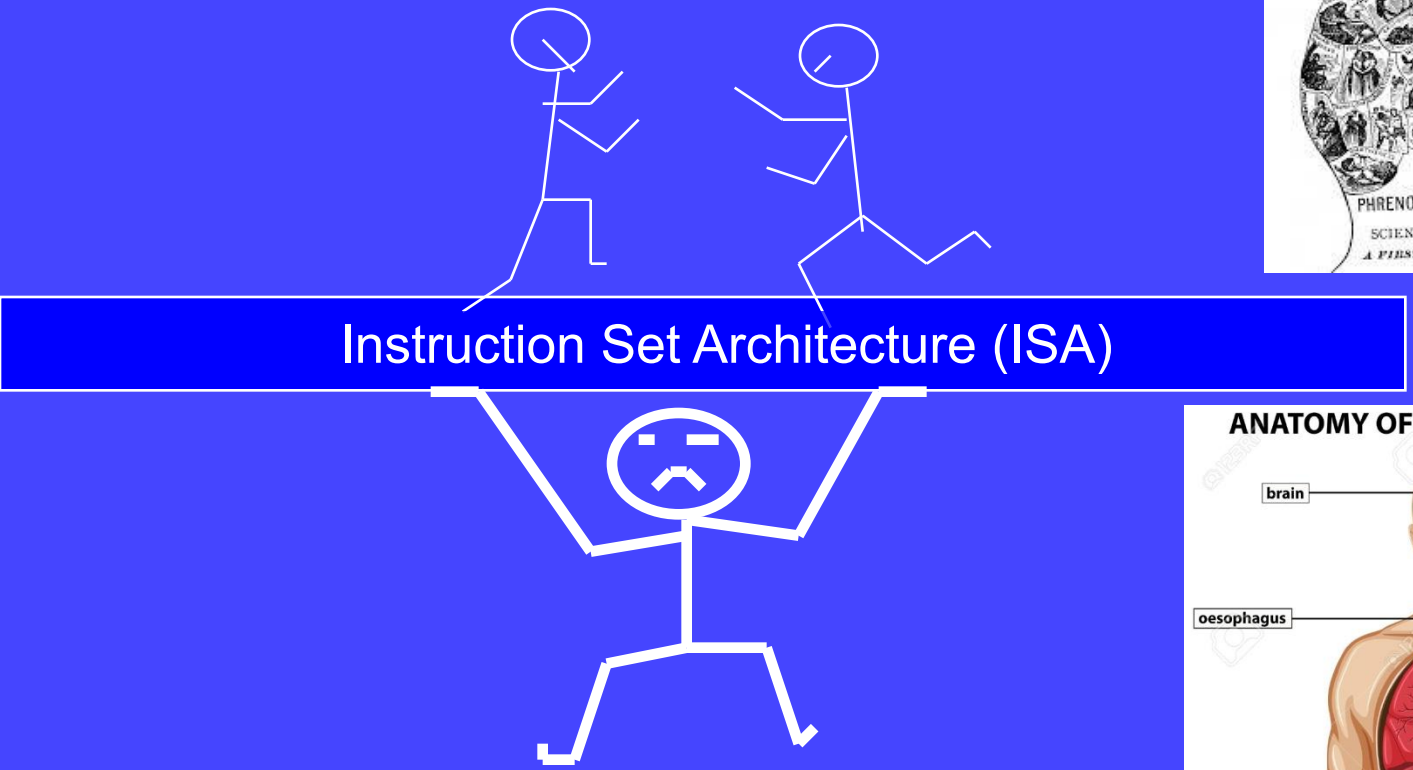
Computer Organization (bottom-up view)

The actual implementation of components in hardware

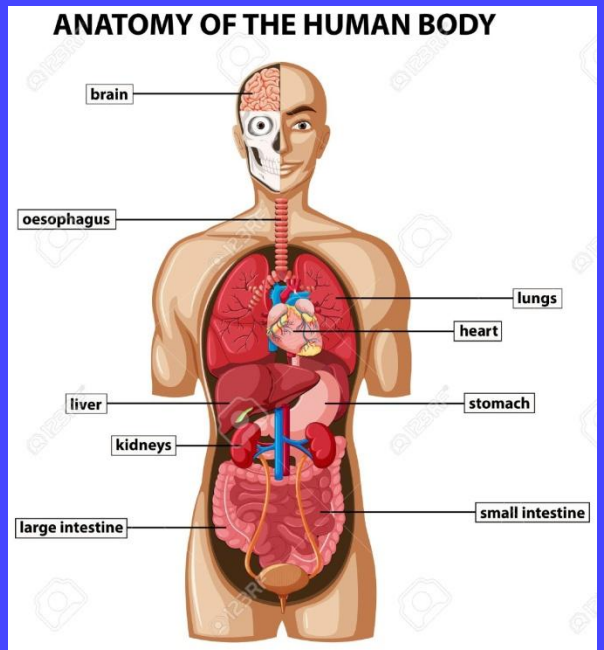
Computer Organization *versus* Computer Architecture

Computer Architecture: The personality of a machine
(behavioral view)





Computer Organization:
The anatomy of a machine
(structural view)



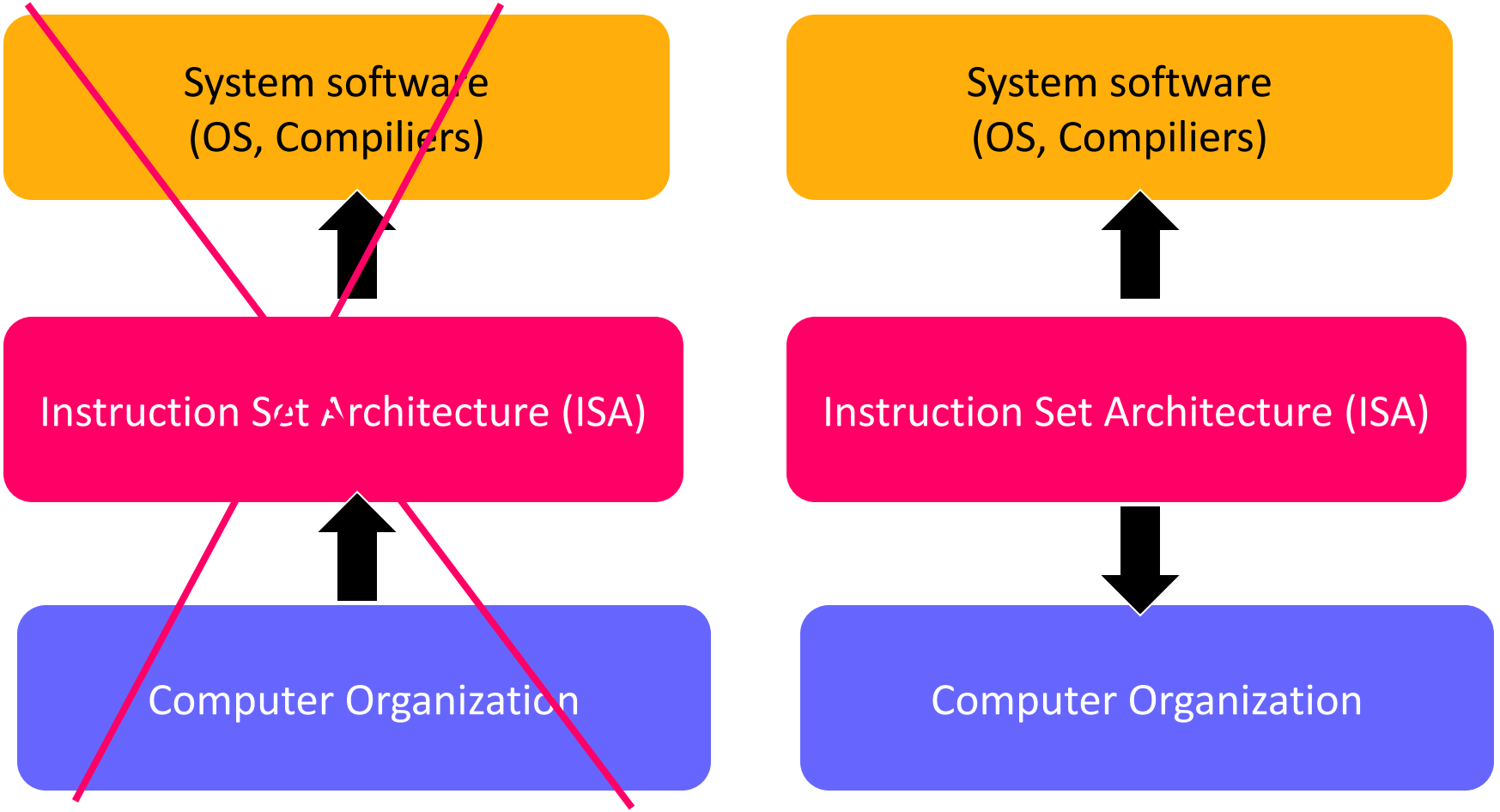
Computer Architecture

- Architecture is visible to a programmer
  - Instruction set
  - Registers
  - Data representation, addressing modes
  - I/O mechanisms
  - Virtual view of memory

Computer Organization

- Organization is visible to hardware designer
  - Hardware implementation of an instruction
  - Registers, program counter
  - Arithmetic and logical units
  - Control logic, internal states
  - Pipelining hardware
  - Cache and main memory

Computer Design Approach



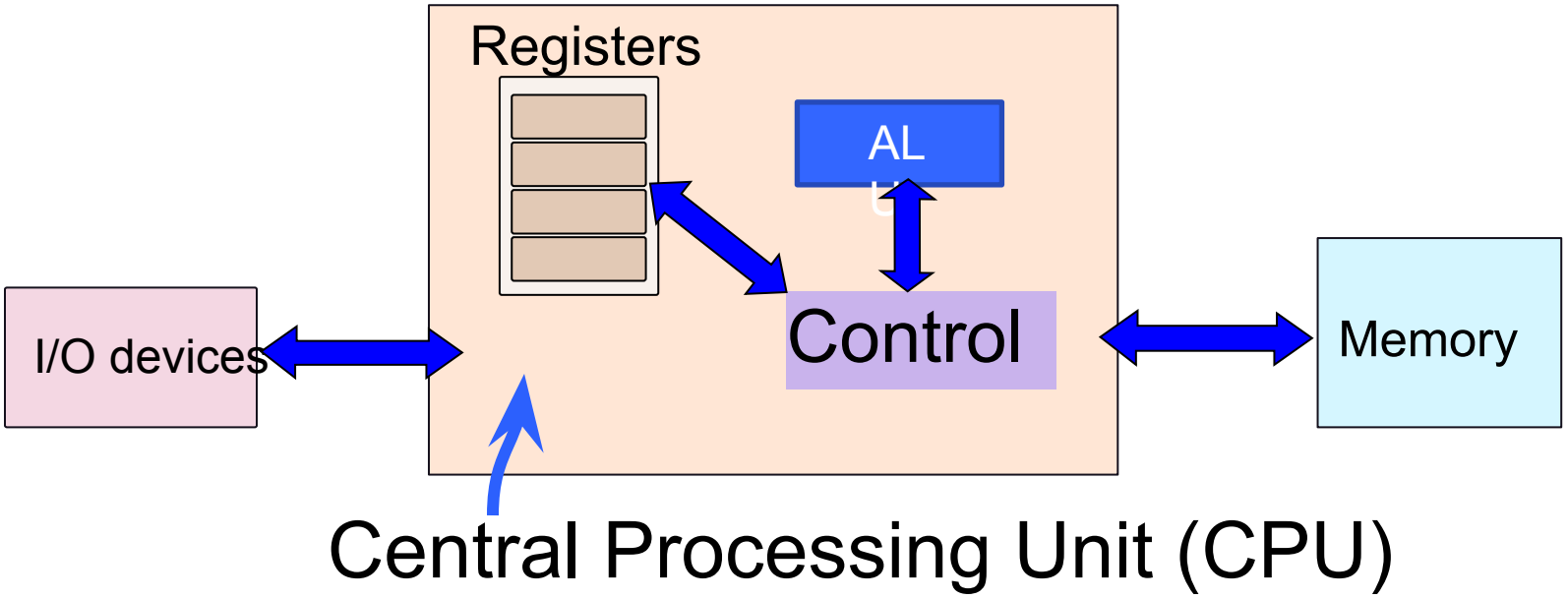
Bottom-up approach:
Hardware first, ISA later

Top-down approach:
ISA first, hardware later

Instruction Set Architecture (ISA)

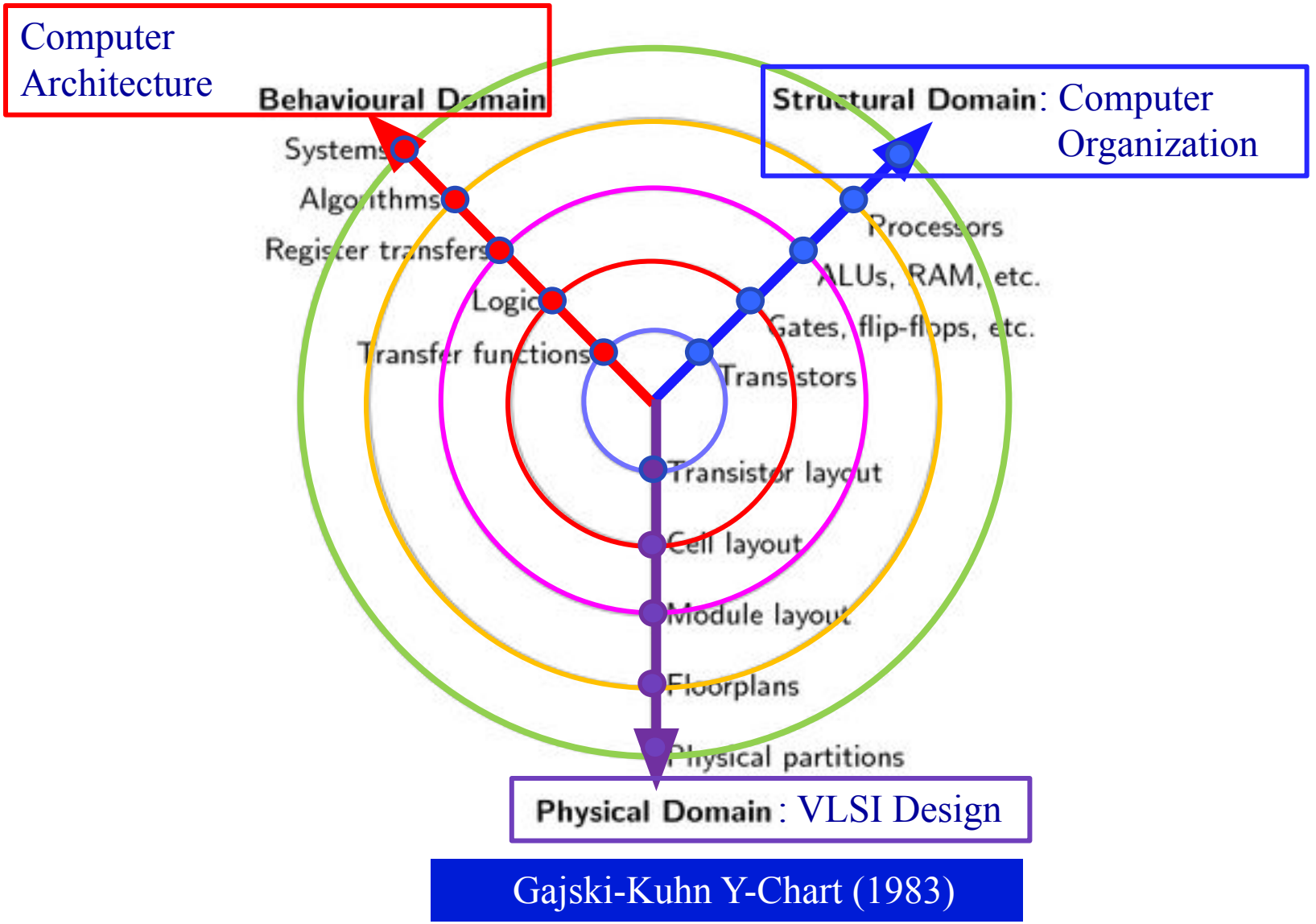
- A set of assembly language instructions that separates the interface between software and hardware
- In top-down design, given an ISA, an appropriate hardware platform is built to support it
- Based on ISA, OS and compilers are to be developed accordingly further going up
- Once ISA is fixed, software and hardware engineers can work independently (start from the middle!)
- ISA is designed to optimize the performance supported by the available hardware technology

The Hardware of a Computer



Five easy pieces:
Control, Arithmetic Logic Unit (ALU), Memory, Input/Output, Datapath (Bus)

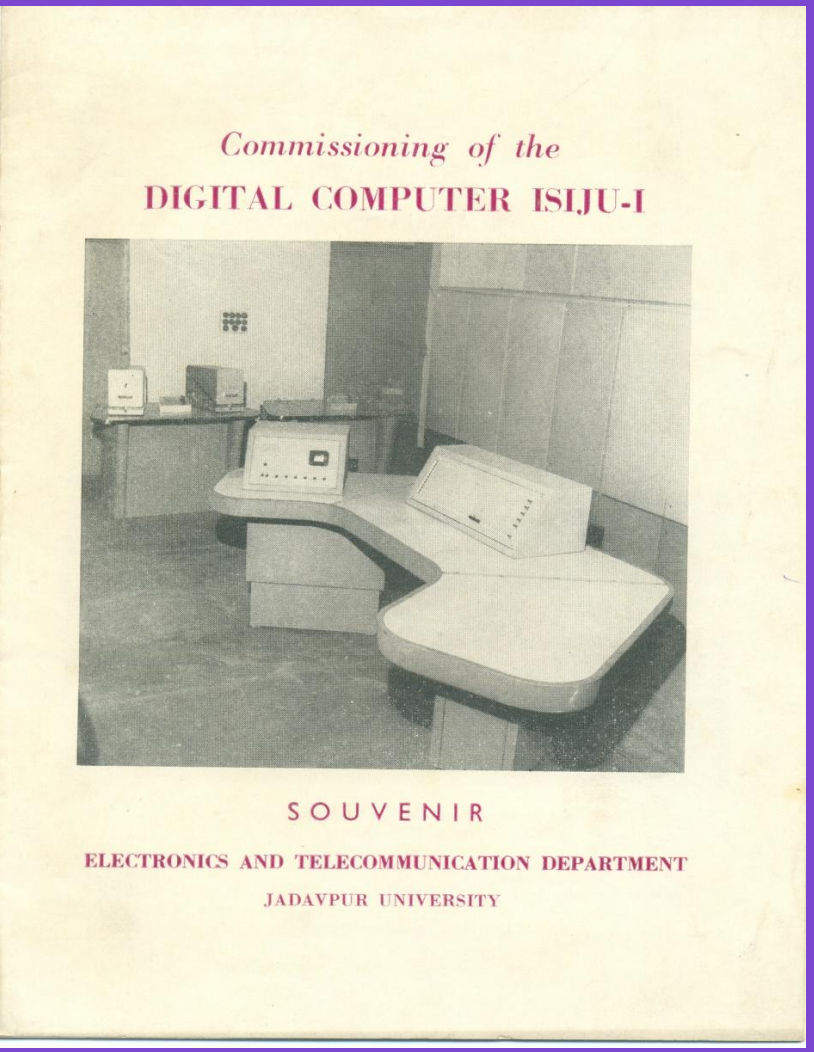
Digital System Design from Three Perspectives



Factors behind the exponential growth of computing paradigms

- Evolution of computer architecture
- Growth of semiconductor technology and IC design processes
- Memory technologies
- Algorithms and data structures
- Programming languages, compilers, OS
- Test, verification, error management techniques
- Bootstrapping: using present computers to design the next generation of computers (design automation)
- Parallel and distributed computing
- Networking, Cloud, IoT

First Computer Built in India



In 1966, a fully-transistorized, digital computer named ISIJU was designed and commissioned, with joint collaboration between Indian Statistical Institute (ISI) and Jadavpur University (JU), Kolkata

Commemoration plaque at the Computer History Museum, Mountain View, California, USA

In 1953, S K. Mitra designed and constructed the first (analog) computer built in India for solving linear equations with ten variables and related problems

CS 31007 Autumn 2021

COMPUTER ORGANIZATION AND ARCHITECTURE

Instructors

Rajat Subhra Chakraborty (*RSC*)

Bhargab B. Bhattacharya (*BBB*)

Lecture #5, #6:

17 August 2021

Indian Institute of Technology Kharagpur

Computer Science and Engineering

Next Agenda

- Model for computation and Turing Machine
- von Neumann Architecture
- Basic Features of Instruction Set Architecture (ISA)
- CPU Performance Equation
- Amdahl's Law; Gustavson-Barsis Law
- RISC *versus* CISC
- Die yield

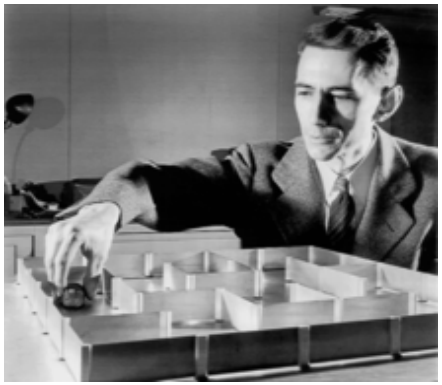
Three Challenges

1. How to design compact, efficient, and reliable hardware (logic)?

2. What is the simplest, yet all powerful computer (computability)?

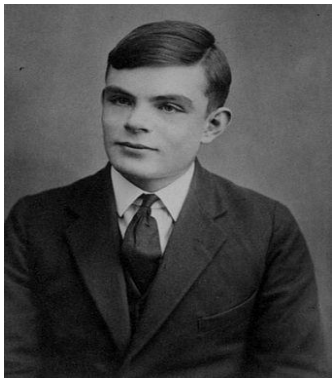
3. How should the basic computer architecture be conceived (mechanism for executing instructions)?

Pioneers who answered these three questions



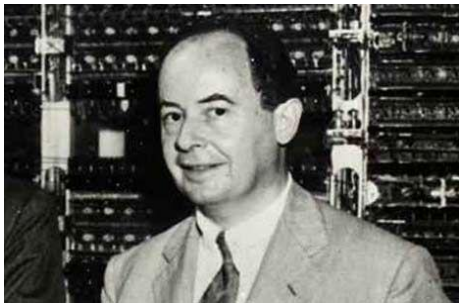
Claude E. Shannon
(1916-2001)

Logic design (basis of computer organization; hardware cost/ circuit delay/power optimization)



Alan Turing
(1912-1954)

Theory of computability (basis for the fundamental requirement in computation)



John von Neumann
(1903-1957)

Blueprint for basic computer architecture

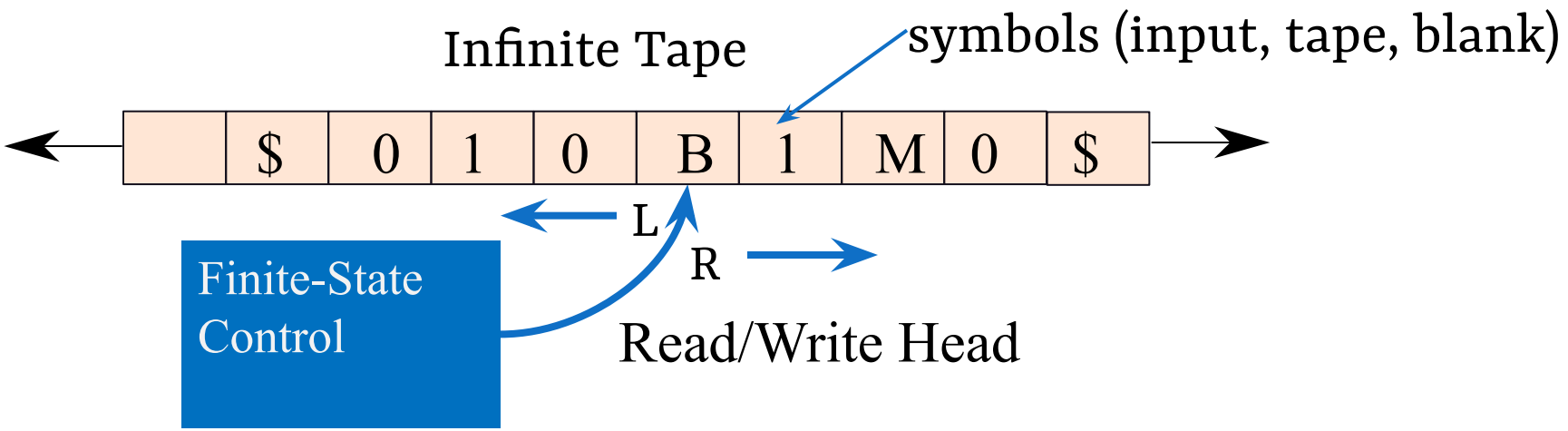
Claude E. Shannon: A Symbolic Analysis of Relay and Switching Circuits, Master's Thesis, MIT, 1937

Turing Machine

- Alan Turing, who gave the fundamental abstraction of a computing machine, was an excellent long distance runner
- Pioneer of theoretical computer science and artificial intelligence
- His father served in Indian Civil Service and worked in Odisha

What is the simplest yet all powerful computer?

Alan Turing (1936): Conceived a machine that introduces a model for computation (Turing Machine)



The tape head can move left or right

Actions: (present state, current symbol) → (new state, write symbol, move one cell left/right);

-- The machine halts when an "accept"/"reject" state is reached

Turing Machine

- Very simple mechanism:
  -- memory, control, read/write, shift, accept/reject

A. M. Turing (1936) , On Computable Numbers with an Application to the Entscheidungsproblem, *Proc. Royal Math. Soc.*, Ser. 2, Vol. 42, pp. 230-265, 1936.

- Extremely powerful

Church-Turing Conjecture (1936): A function on natural numbers can be calculated by an effective method *if and only if* it is computable by a Turing machine

Any procedure that is computable by paper-and-pencil methods (algorithm) can be solved by a Turing machine

Some Fundamental Limitations

Albert Einstein (1905):
Special theory of relativity → elimination of the notion of absolute time (postulate: the velocity of light is constant)

Kurt Gödel (1931):
Incompleteness theorem → questioned the notion of absolute mathematical truth

Alan Turing (1936):
Halting problem of Turing Machine → algorithmic inability to identify the presence of an infinite loop in arbitrary programs

Turing Undecidability?

Halting Problem: Can we write a program *Q*, which given any arbitrary program *P* as input, will decide whether on not *P* terminates or falls into an infinite loop on some input data?

Halting problem is Turing undecidable

Gödel's Incompleteness Theorem

Kurt Gödel
1906-1978

- In 1931, Gödel demonstrated that within any given branch of mathematics, there would always be some propositions that cannot be proven either true or false using the rules and axioms

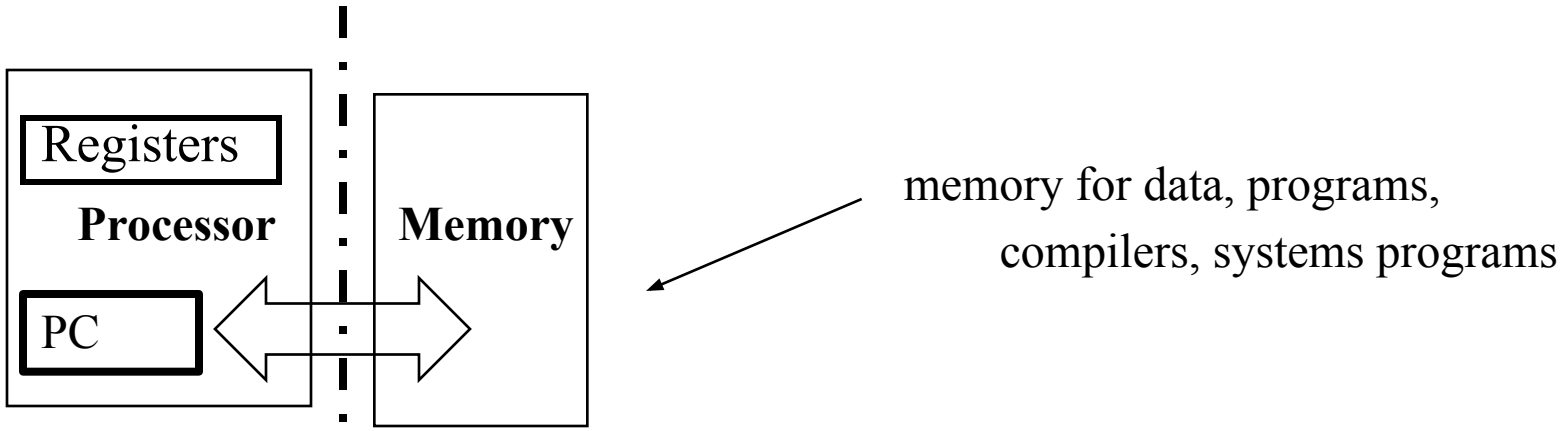
Father of AI/Machine Learning?

"What we want is a machine that can learn from experience"

- Alan Turing

von Neumann Architecture (1945): Princeton Architecture

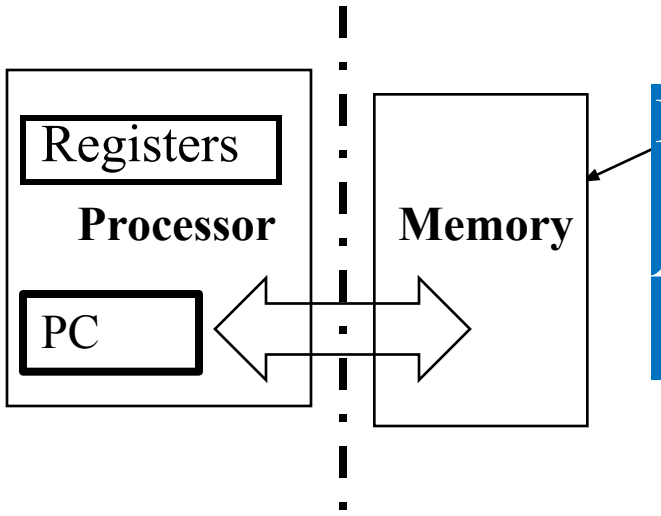
- Stored program concept
- Serves as the basis for almost all modern computers
- Instructions and data are just bits
- Programs (sequence of machine instructions) are stored in memory to be read or written just like data



- Fetch & Execute Cycle (Instruction Cycle)
  - Program Counter (PC) points to the next Instruction to be fetched
  - Specific bits (opcode field) in the Instruction "control" subsequent actions
  - Fetch the “next” instruction and continue

Binary String Stored in Memory: Who am I?

- X = - 1,880,113,152 (if X is a 2's complement binary number)
- X = 2,414,854,144 (when X is an unsigned binary number)
- X = - 1.873 × 2^{-96} (when X is a floating-point number)
- X ⟶ lw $t7, -16384 ($ra) (when X is a MIPS instruction)

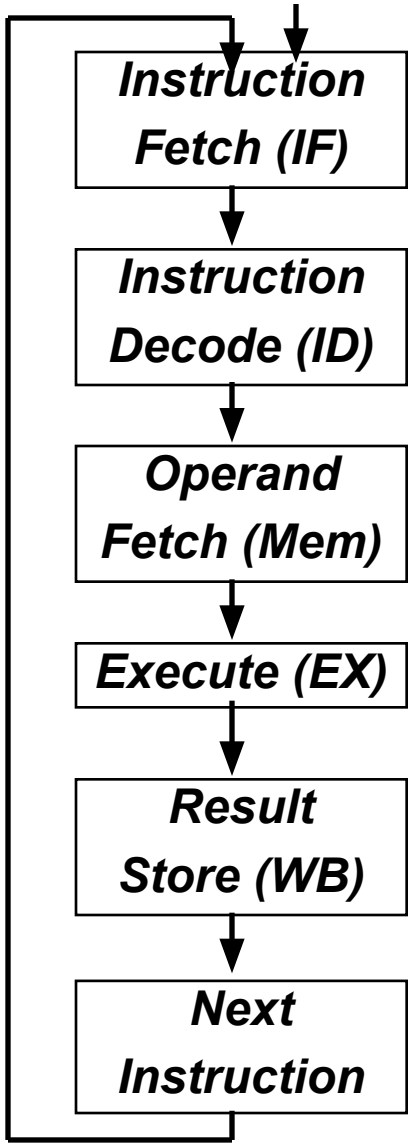


Let the content (X) of a memory location be:

X: 0x 8FEFC000

- Need execution cycles to interpret the binary strings properly – whether it is instruction or data, and if data, what type?

Execution Cycle *a.k.a.* Instruction Cycle



Obtain instruction from memory

Determine required actions and instruction size

Locate and obtain operand data from memory/register (Mem)

Compute result value or status (Execute)

Deposit results in register/memory for later use (Write Back)

Determine successor instruction

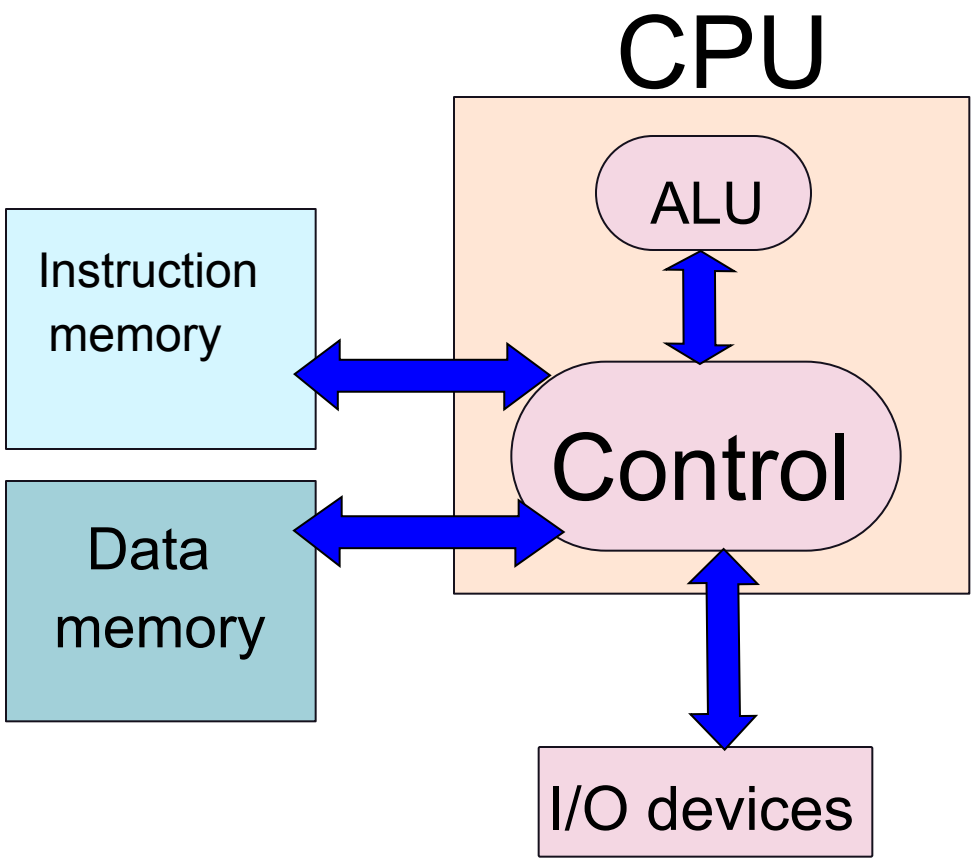
(Program Counter points to next instruction; next sequential memory address or branch targets)

Features of von Neumann Architecture

- Same physical memory to save instructions and data
- Instruction fetch and data transfer cannot be done concurrently; they need separate clock cycles
- Simple architecture

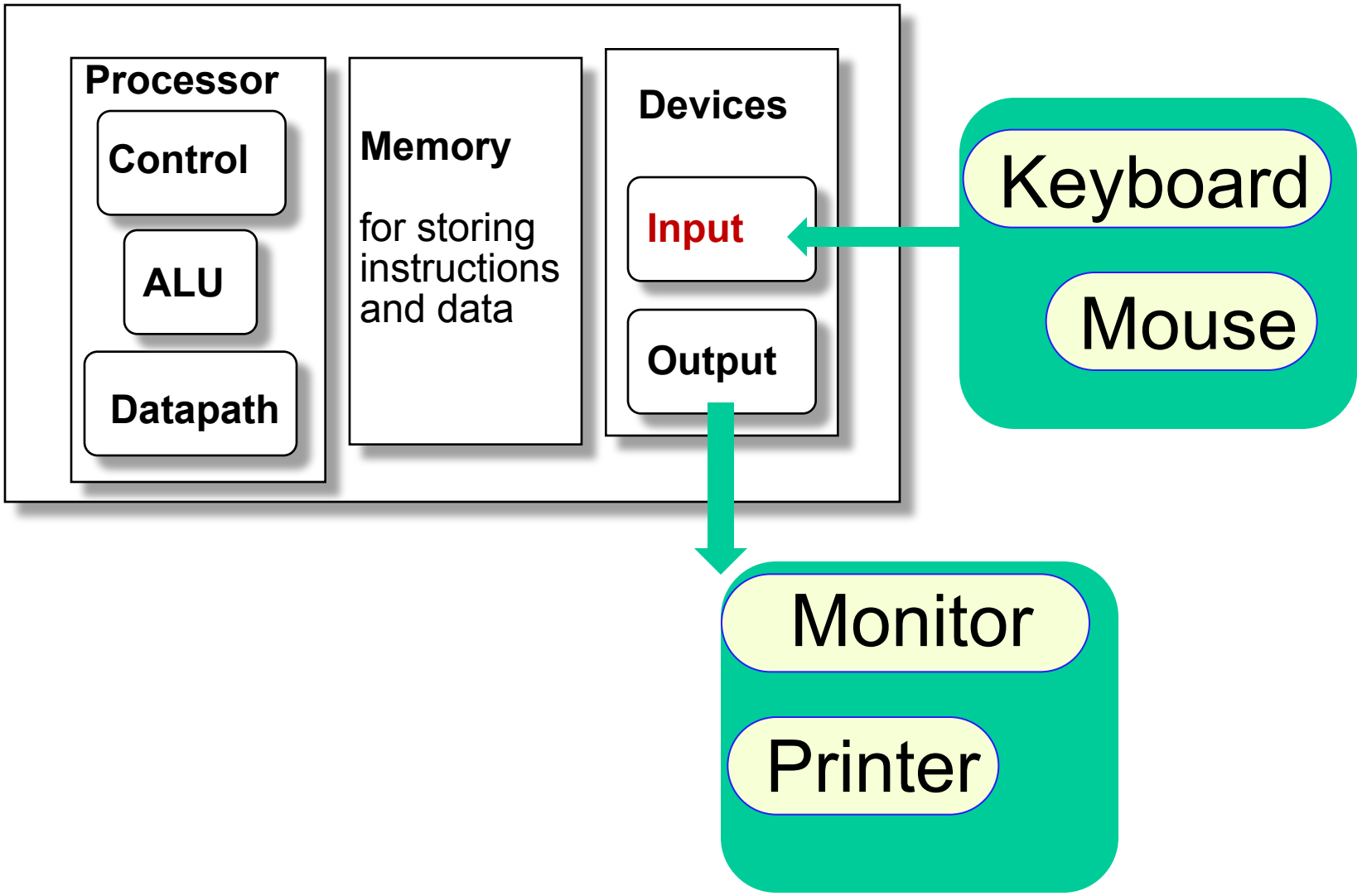
- Harvard Architecture: Separate instruction and data memory

Harvard Architecture



- Based on Harvard Mark I relay-based computer model
- Separate signal pathways for instruction and data; can be accessed concurrently

Full Picture



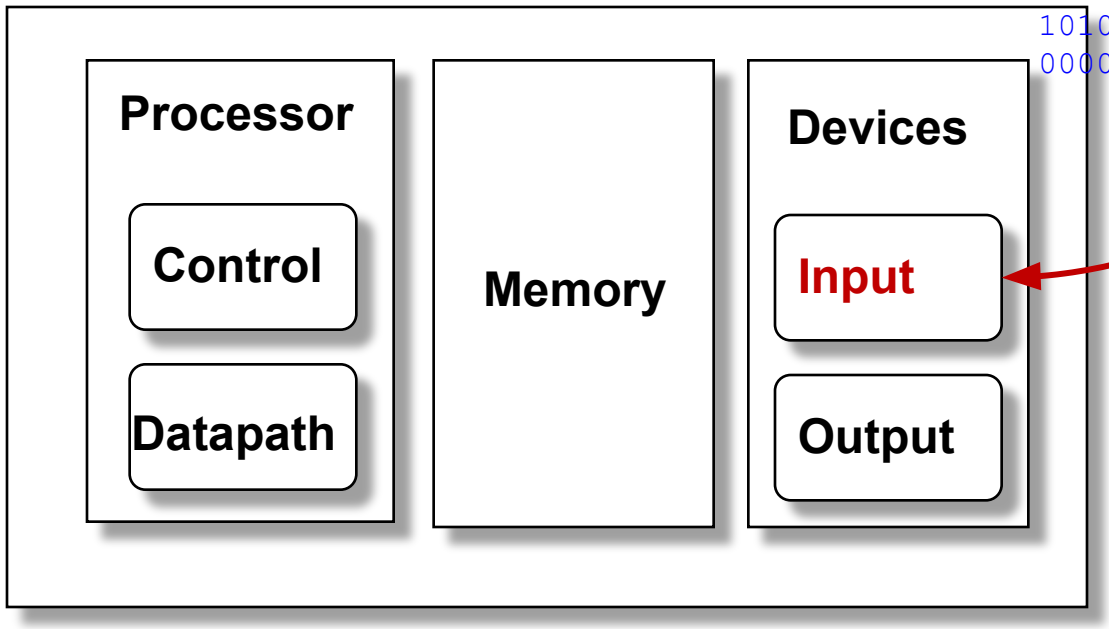
Next ..

- How programs are translated into the machine language
  - And how the hardware executes them
- The hardware/software interface
- What determines program performance
  - And how it can be improved
- How hardware designers improve performance
- How parallel processing helps

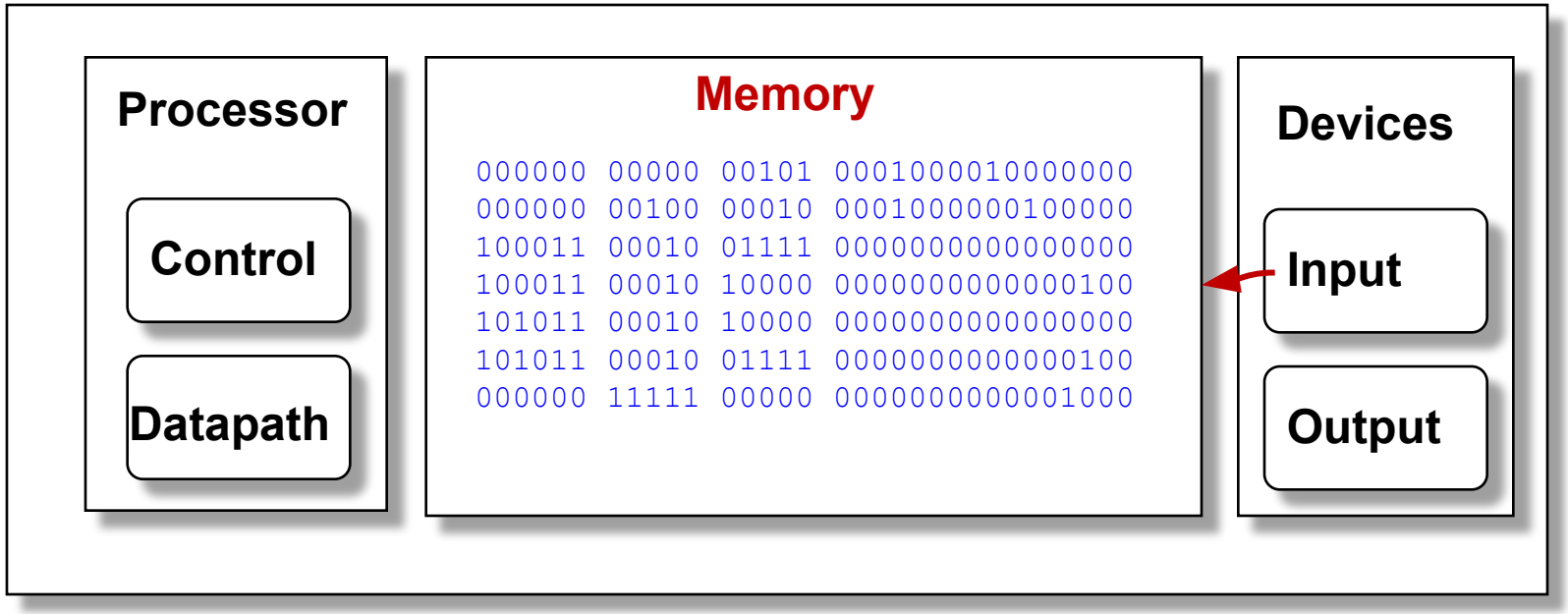
How machine-language programs are executed?

Load Input Binary

```
000000 00000 00101 0001000010000000
000000 00100 00010 0001000000100000
100011 00010 01111 0000000000000000
100011 00010 10000 0000000000000100
101011 00010 10000 0000000000000000
101011 00010 01111 0000000000000100
000000 11111 00000 0000000000001000
```

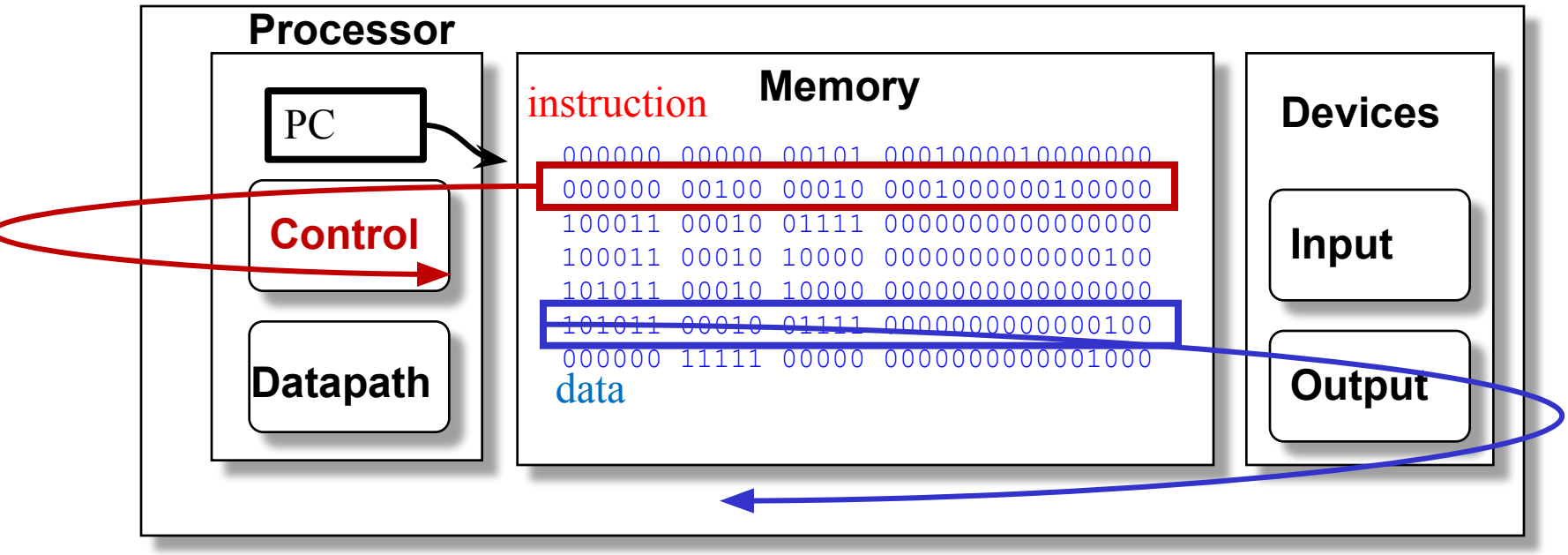


Code Stored in Memory



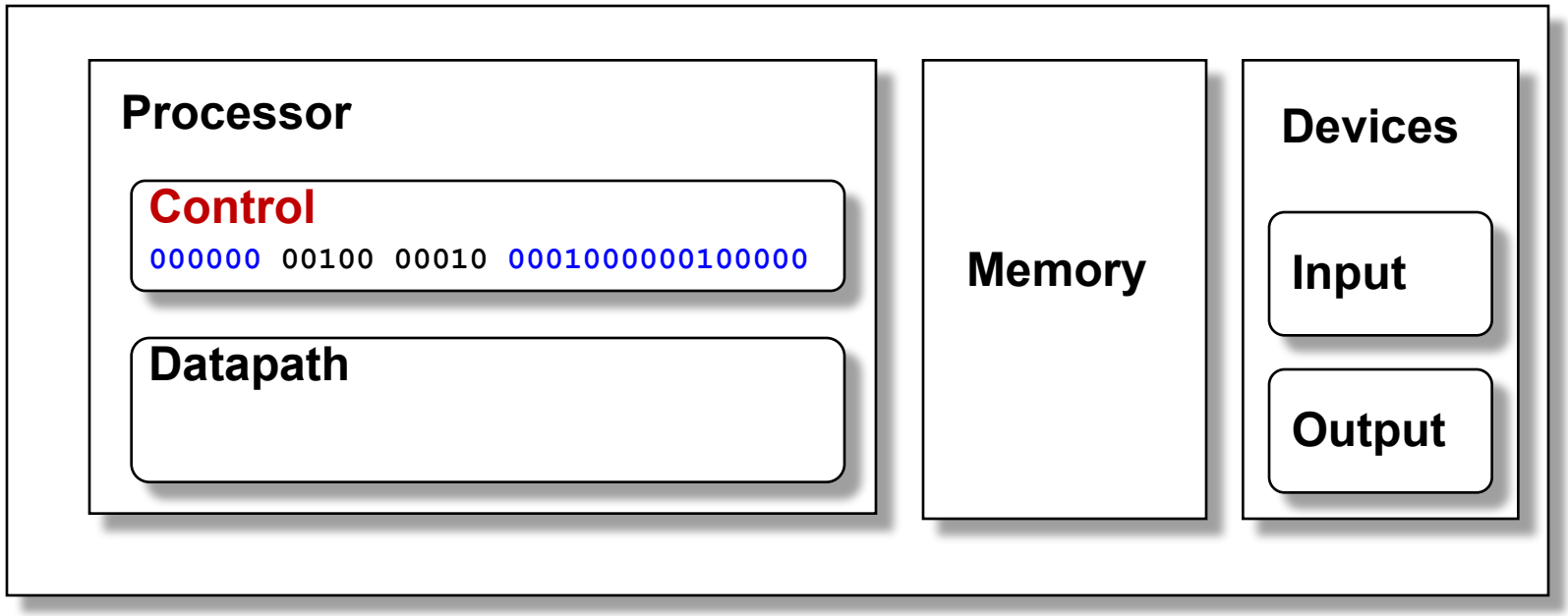
Processor Fetches an Instruction

Processor fetches an instruction from memory pointed by Program Counter PC



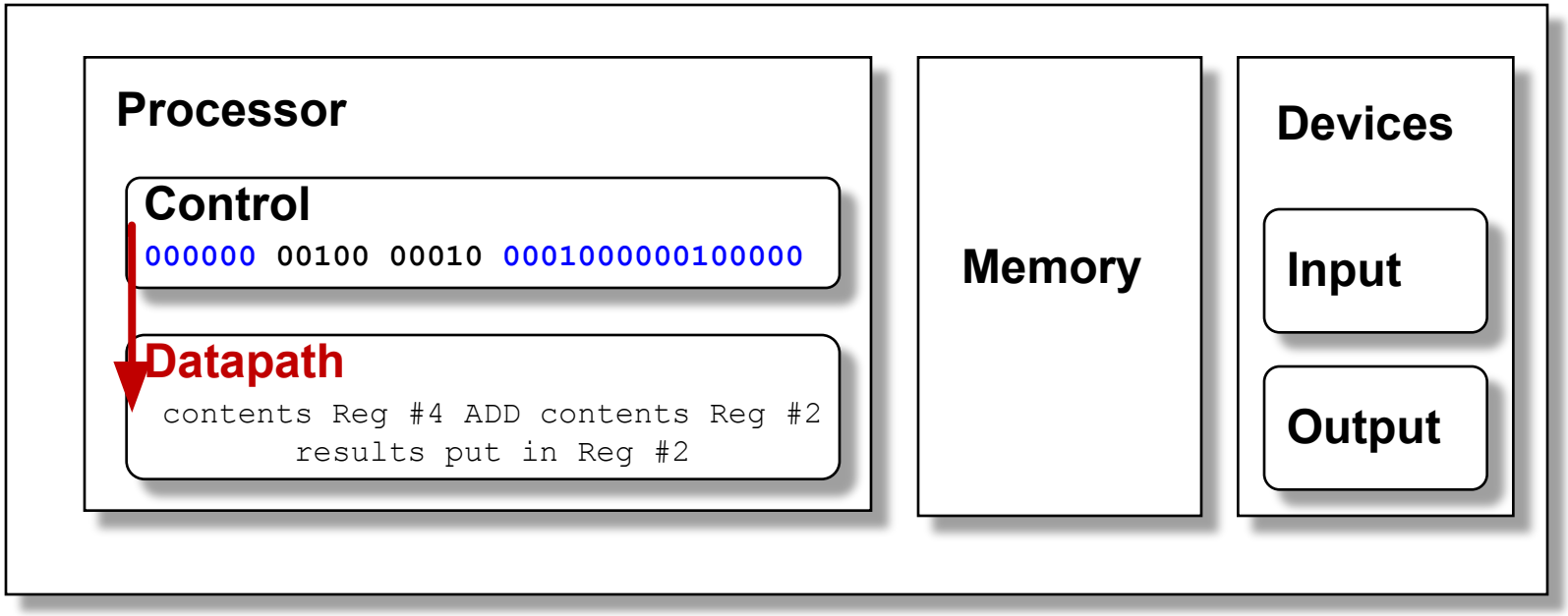
Where does it fetch from?

Control Decodes the Instruction



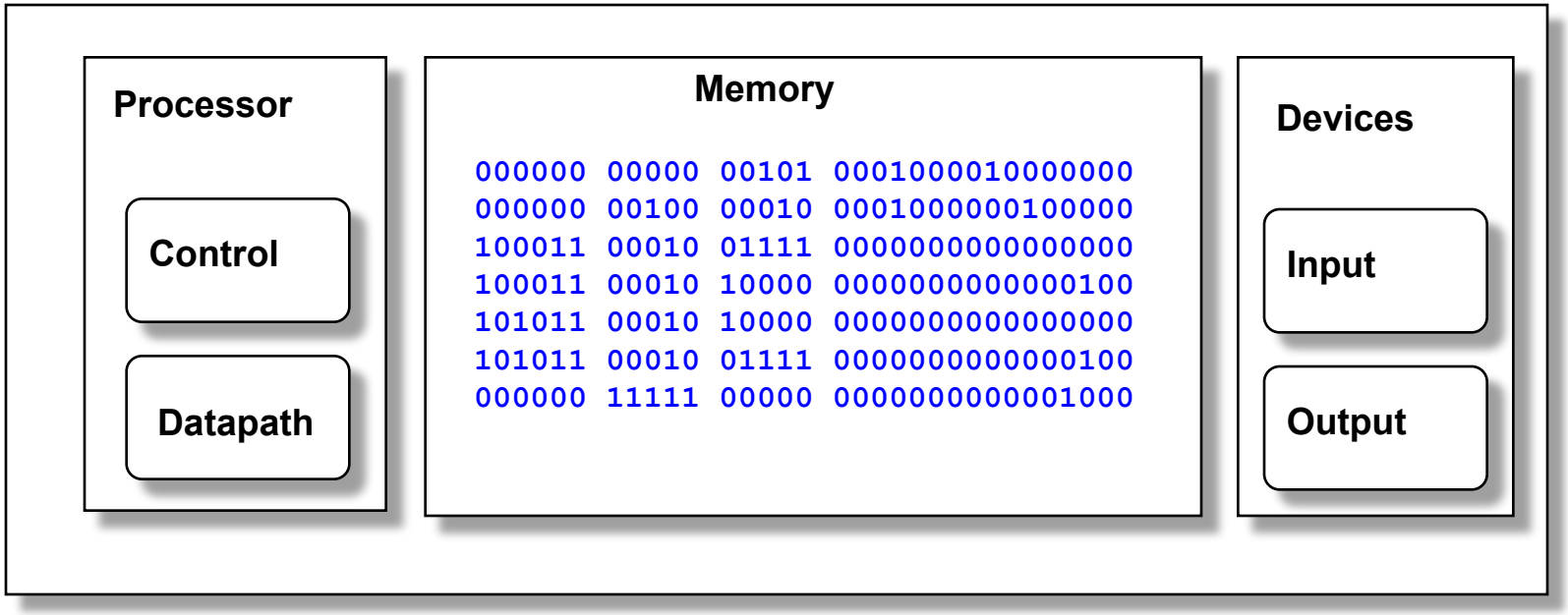
Control decodes the instruction to determine what to execute

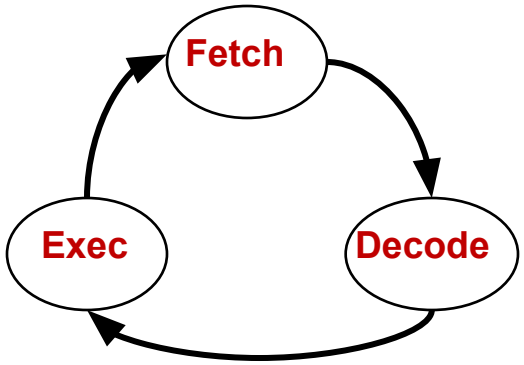
Datapath Executes the Instruction



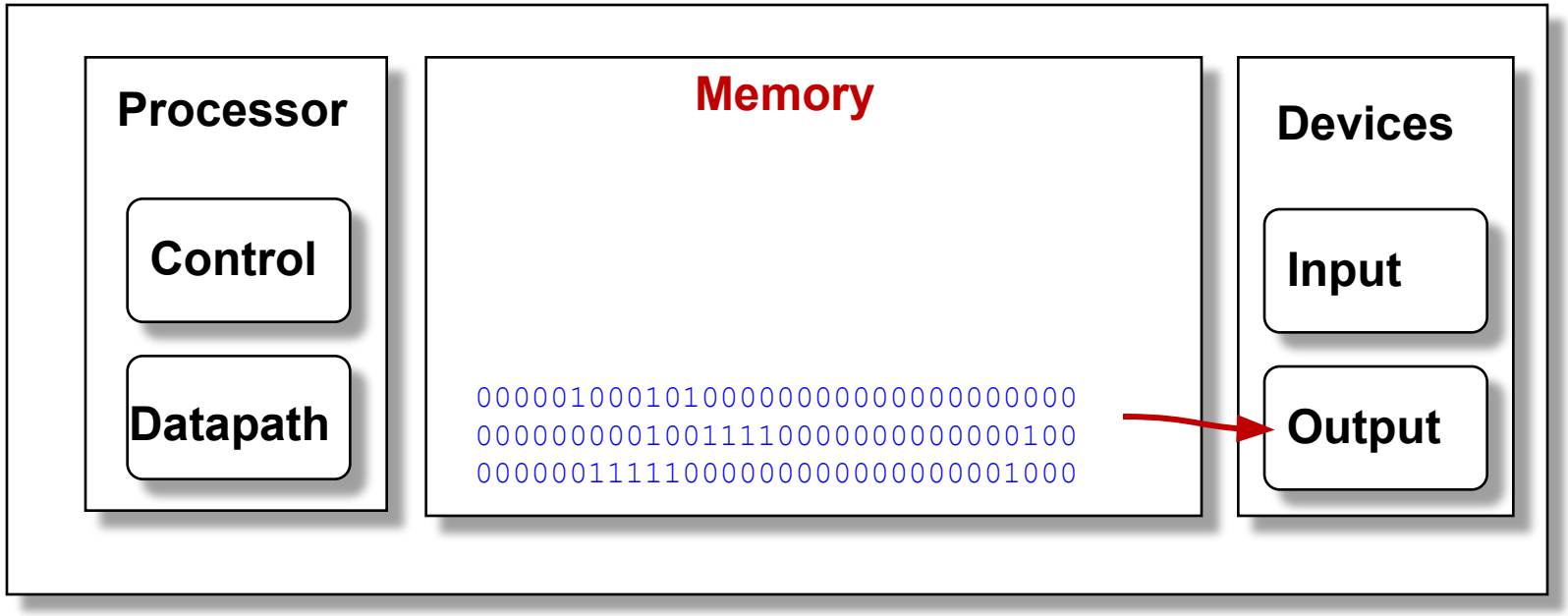
Datapath executes the instruction as directed by control

What Happens Next?



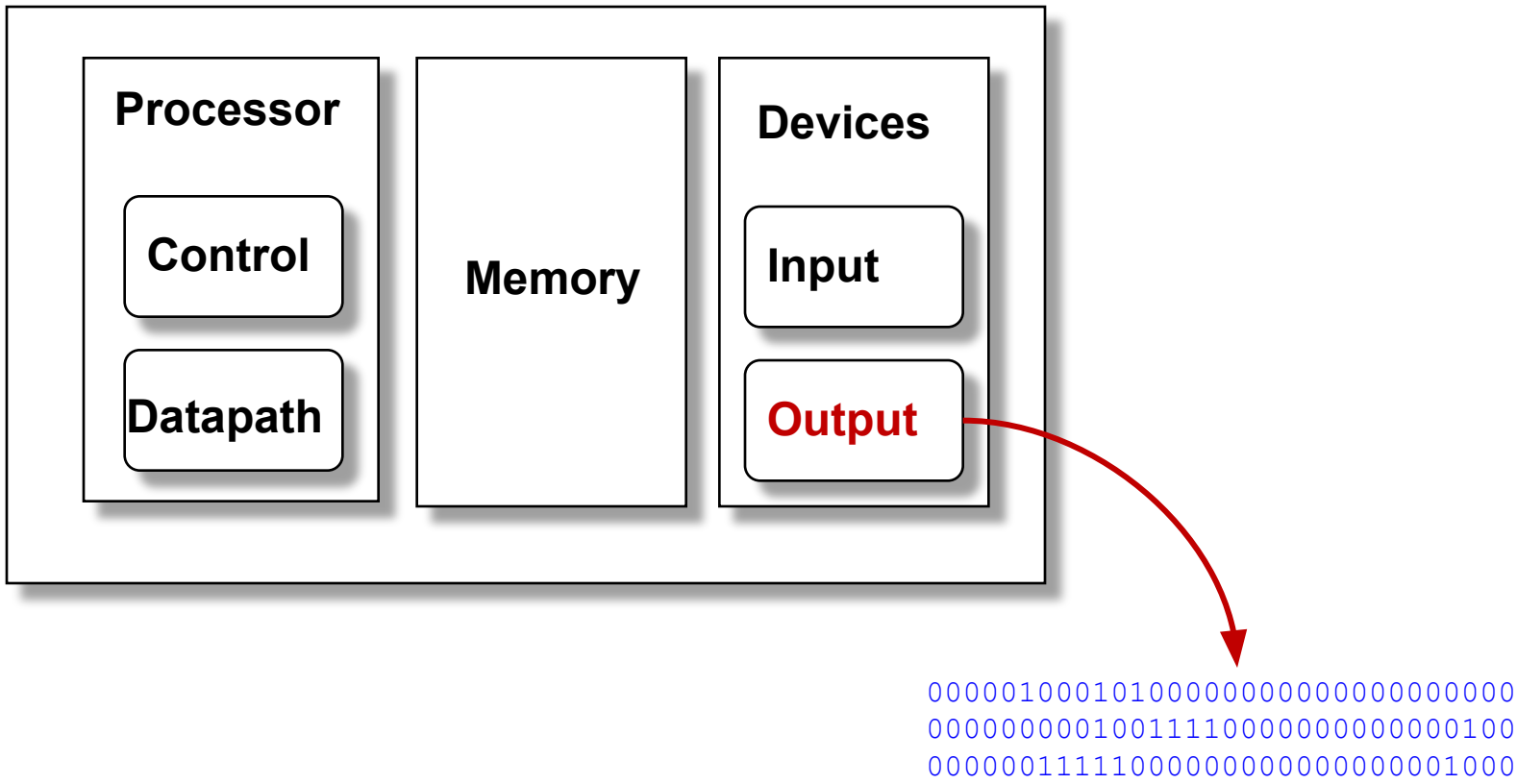


Output Data Stored in Memory



On program completion, results reside in memory

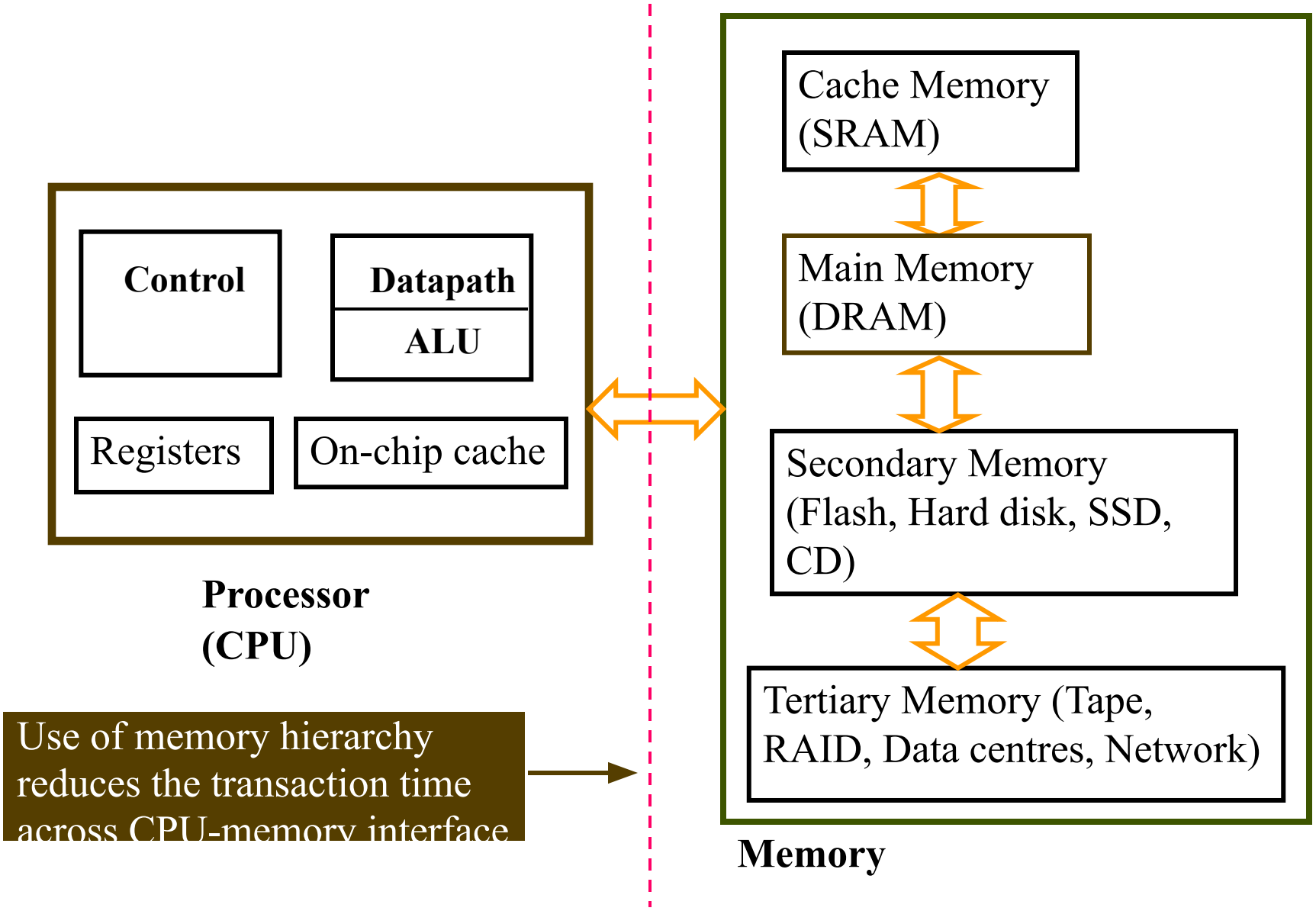
Output Device Outputs Data



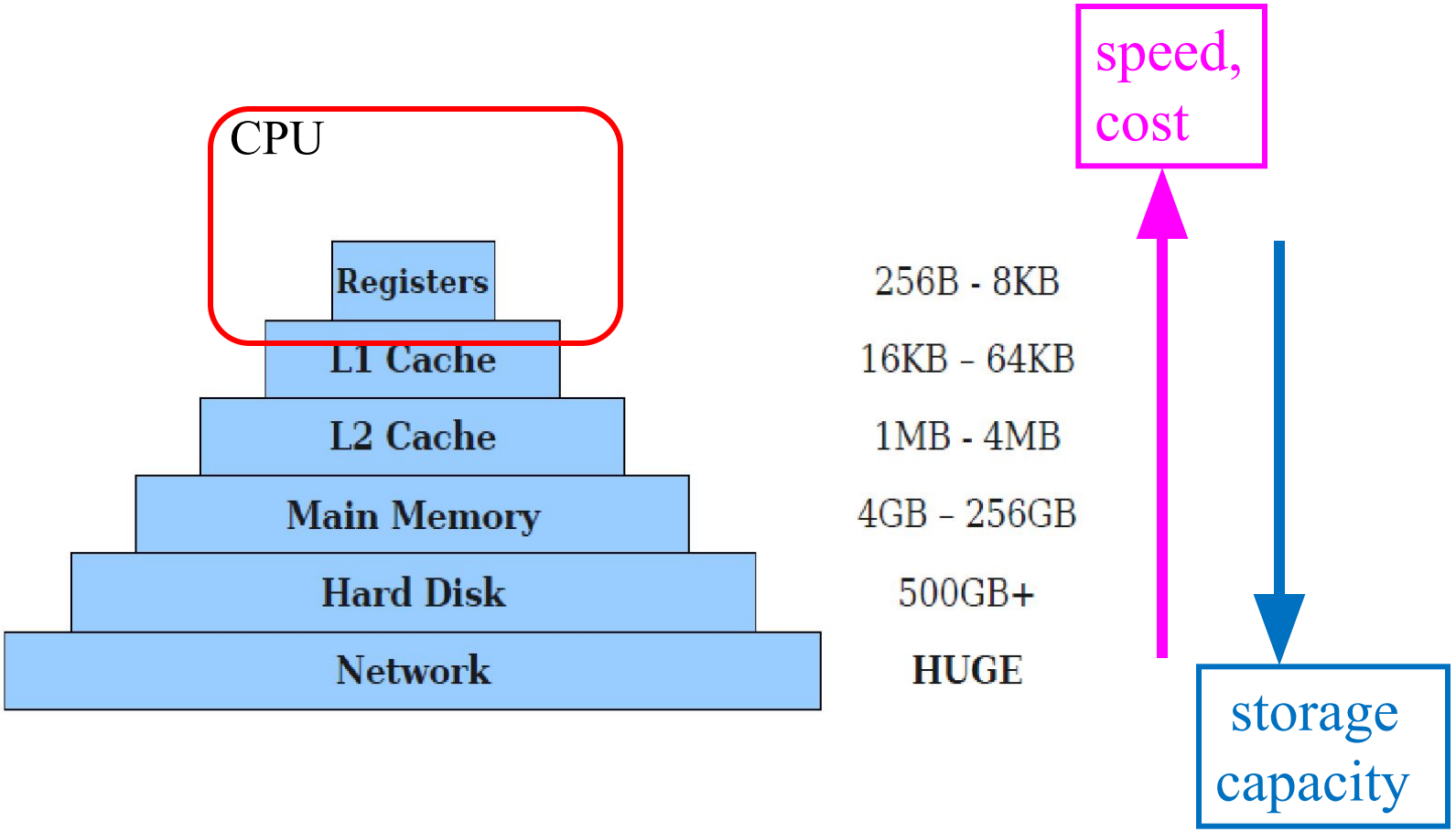
Von Neumann Bottleneck

- Von Neumann architecture uses the same memory for instructions (program) and data.
- Memory is inherently slower than logic. The time spent in memory accesses can limit the performance. This phenomenon is referred to as *von Neumann bottleneck*.
- To avoid the bottleneck, later architectures allow frequently used operands to reside in on-chip registers, or use cache.

Complete View of Computer Architecture



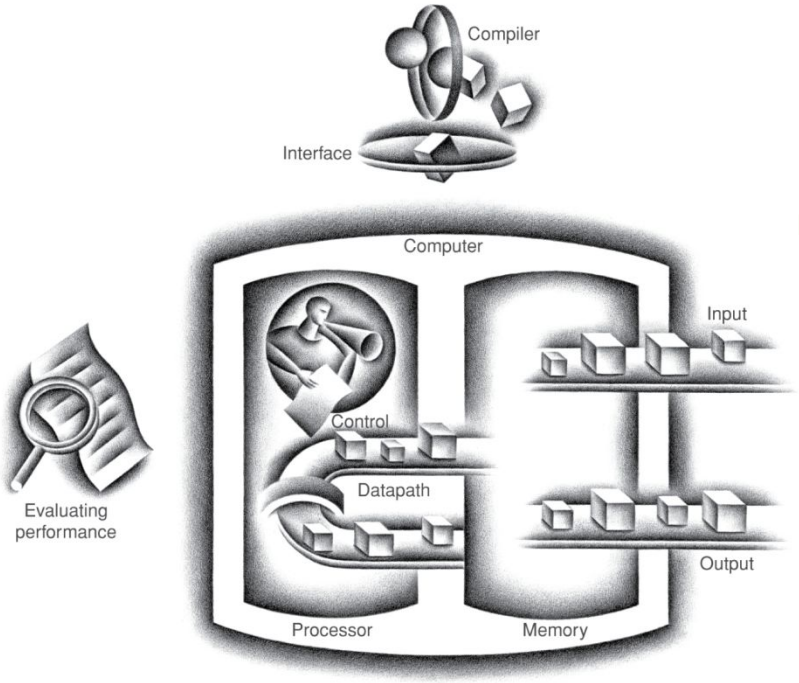
Memory Hierarchy (MH)



MH reduces average CPU-Memory transaction time;
need additional hardware, OS (memory management)

Components of a Computer

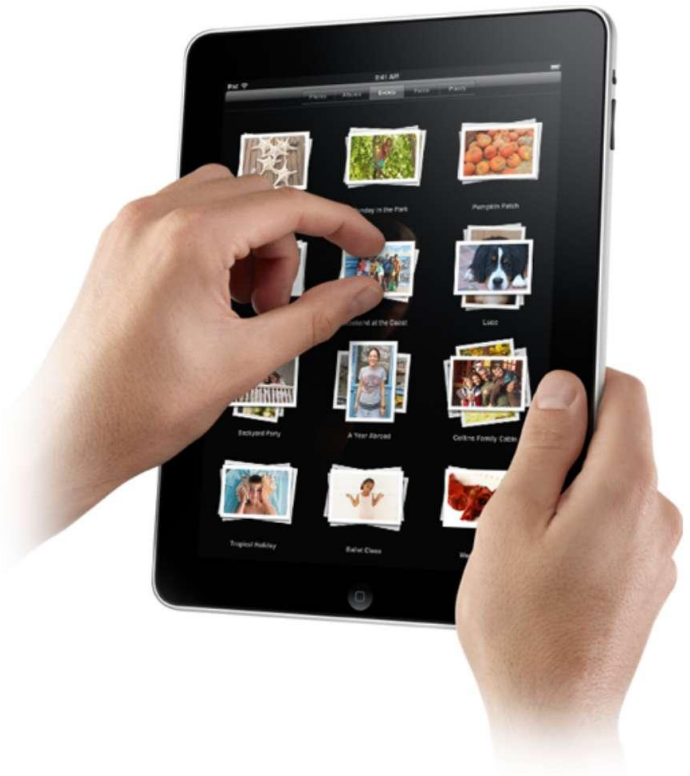
The BIG Picture



- Same components for all kinds of computer
  - Desktop, server, embedded
- Input/output includes
  - User-interface devices
    - Display, keyboard, mouse
  - Storage devices
    - Hard disk, CD/DVD, flash
  - Network adapters
    - For communicating with other computers

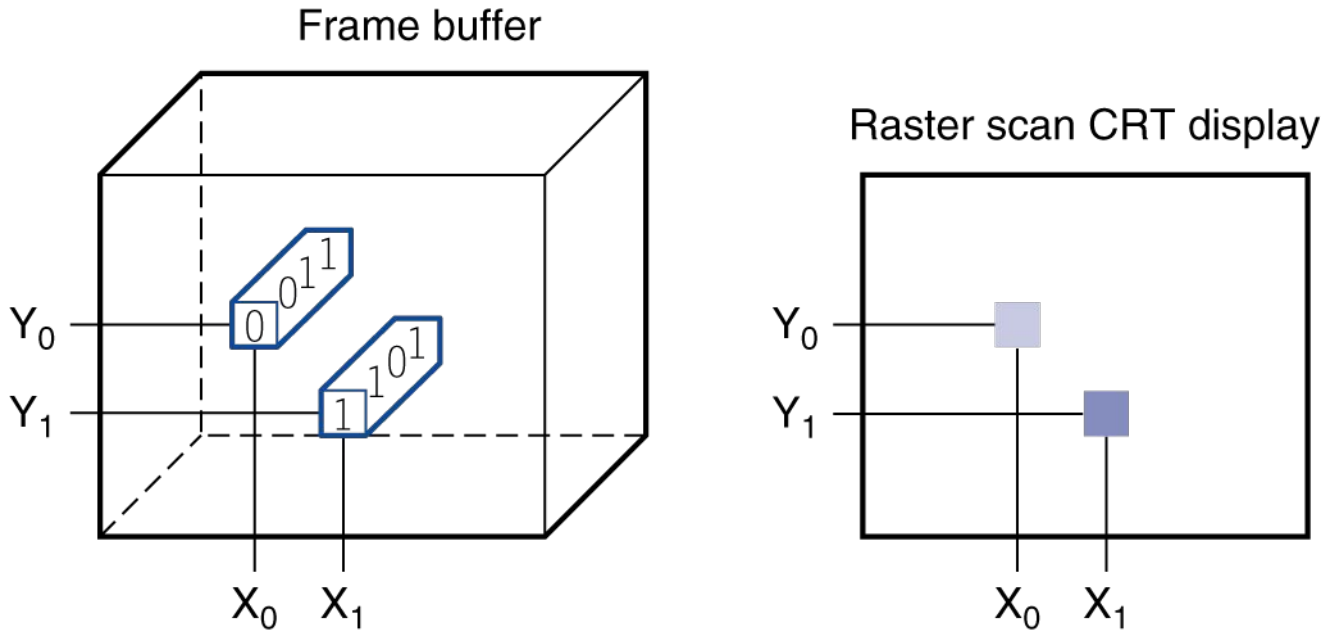
Touchscreen

- PostPC device
- Supersedes keyboard and mouse
- Resistive and Capacitive types
  - Most tablets, smart phones use capacitive
  - Capacitive allows multiple touches simultaneously

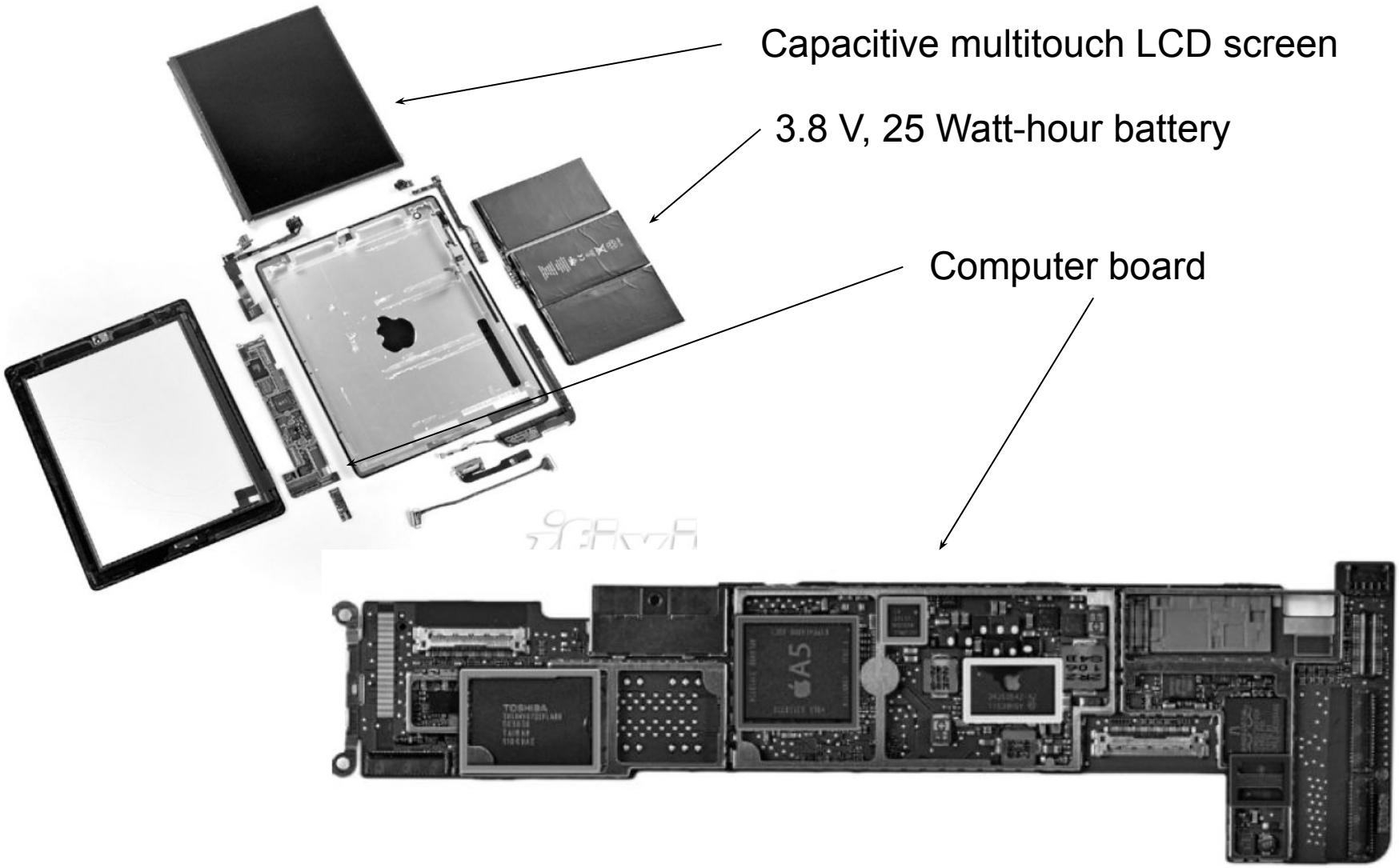


Through the Looking Glass

- LCD screen: picture elements (pixels)
  - Mirrors content of frame buffer memory



Opening the Box

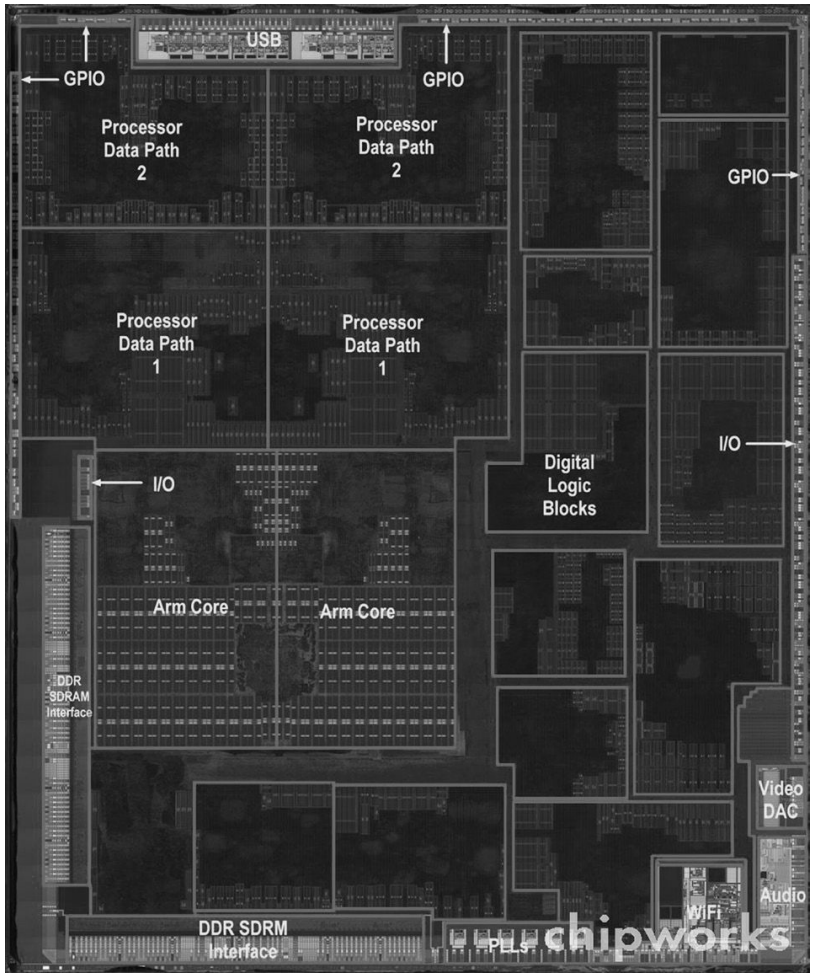


Inside the Processor (CPU)

- Datapath: performs operations on data
- Control: sequences datapath, memory, ...
- Cache memory
  - Small fast SRAM memory for immediate access to data

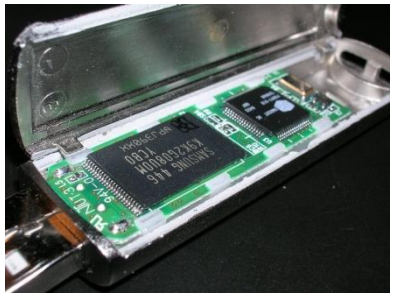
Inside the Processor

- Apple A5



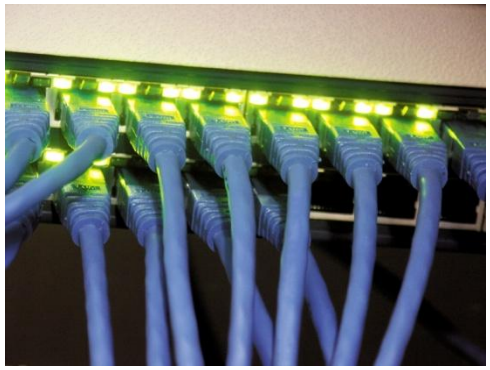
A Safe Place for Data

- Volatile main memory
  - Loses instructions and data when power off
- Non-volatile secondary memory
  - Magnetic disk
  - Flash memory, SSD
  - Optical disk (CDROM, DVD)



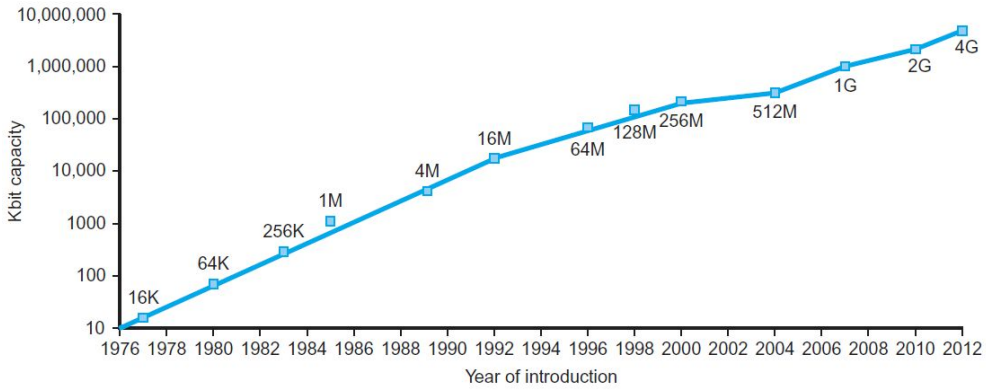
Networks

- Communication, resource sharing, nonlocal access
- Local area network (LAN): Ethernet
- Wide area network (WAN): the Internet
- Wireless network: WiFi, Bluetooth



Technology Trends

- Electronics technology continues to evolve
  - Increased capacity and performance
  - Reduced cost



DRAM capacity

Instruction Set Architecture (ISA)

- The set of machine-level instructions in a particular CPU implementation is called *Instruction Set*

- Goals of Instruction Set:
  - Software must be able to compute anything in a reasonable number of steps using the instructions in the instruction set
- Different CPUs implement different sets of instructions

Features of Instruction Set Architecture (ISA)

Instruction format: length (how many bits – fixed or variable?), format, fields, opcodes, register specifications, how many operands/memory addresses specified?
Size of the logical address space? **Addressability** (byte/word level)?
Number of instructions? – determines #bits in op-code.
Instruction types? – arithmetic (integer/floating point), logical, data transfer, branch, procedure calls, bit-shifting
Addressing modes: immediate, direct, register, displacement, scaled, indirect; addressing granularity (word-level, byte-level?)
Others: Orthogonality, Completeness, Alignment (Big-Endian/Little-Endian)

ISA governs both hardware implementation (downward) and compiler design (upward)

Example: MIPS ISA

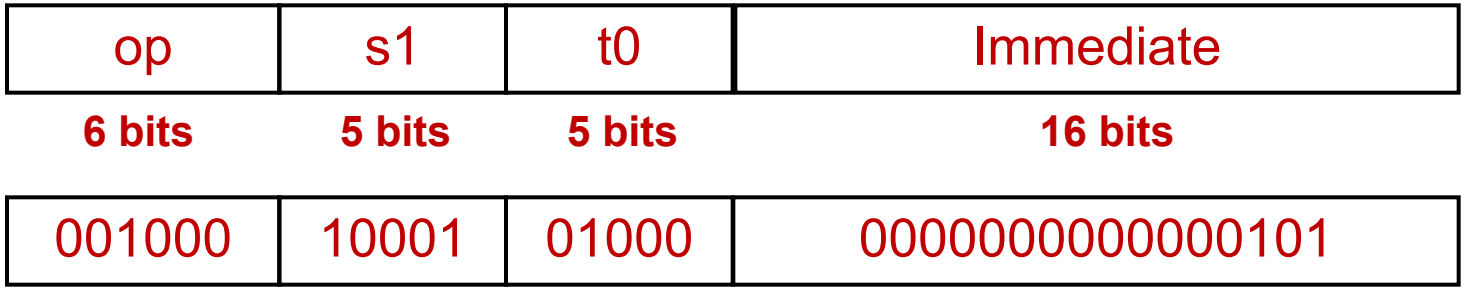
- The length of every instruction = 32 bits
- Memory is byte-addressable, word-organized, each word comprising 4 bytes = 32 bits
- Width of the address bus: 32 bits; size of the logical address space = 2^32 bytes =2^30 words
- 32 on-chip general purpose registers, each 32-bit wide; register-ID needs 5 bits each
- Integers, FP-numbers: 32 bits
- Op-code in each instruction: 6 bits

MIPS: Microprocessor without Interlocked Pipeline Stages (RISC Computer, 1986)
MIPS: Million Instructions Per Second (Performance)

Example: MIPS Instruction

Collection of Assembly-level or Machine-level (M/L) instructions, which are executable by the hardware

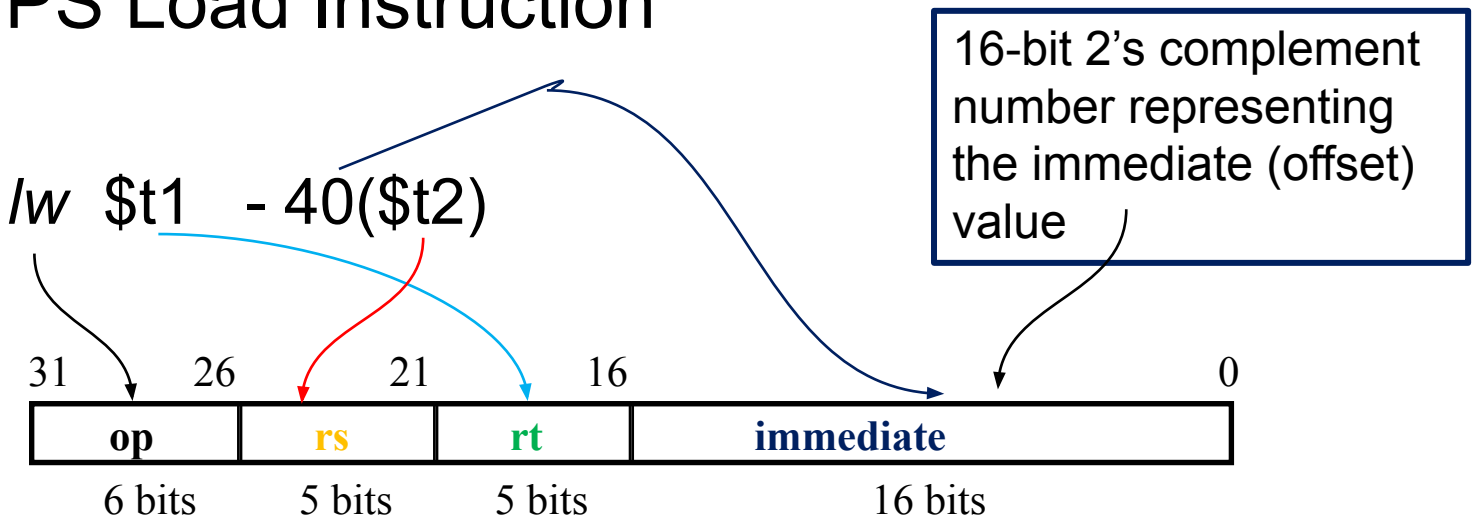
An example of MIPS M/L instruction: `addi $t0, $s1, 5`

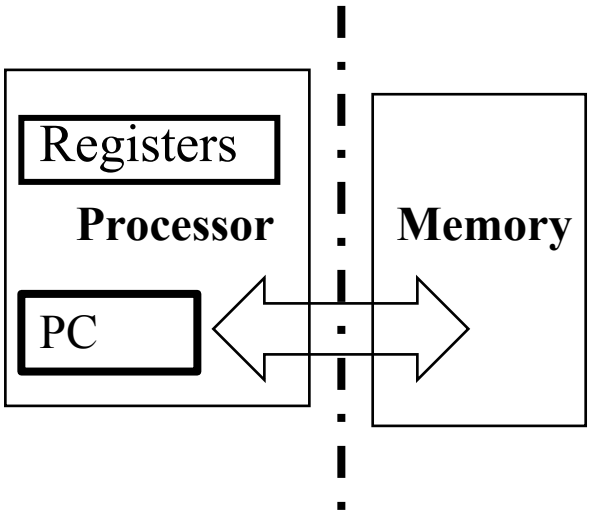


Total length of the instruction – 32 bits; *opcode*: 6 bits; $t0, $s1 -> 32-bit CPU *registers* (ID: 5-bit each, as there are a total of 32 such registers); *immediate value* is provided as a 16-bit 2's complement integer;

Add 5 to the content of $s1 and save the result in $t0

MIPS Load Instruction





Meaning of *lw* $rt imm($rs):

R[rt] ← Mem{R[rs] + Sign_Ext(imm16)}

MIPS is based on *load-store architecture*: instructions are broadly classified as two types - memory access (load and store between external memory and registers) and ALU operations (occurring only among internal registers)

Immediate Operands

- Constant data specified in an instruction

```
addi $s3, $s3, 4
```

- No subtract immediate instruction
  - Just use a negative constant

```
addi $s2, $s1, -1
```

- *Design Principle 3:* Make the common case fast
  - Small constants are common
  - Immediate operand avoids a *load* instruction, which memory access

Arithmetic Operations

- Add and subtract, three operands
  - Two sources and one destination

```
add a, b, c   # a gets b + c
```

- All arithmetic operations have this form
- *Design Principle 1:* Simplicity favours regularity
  - Regularity makes implementation simpler
  - Simplicity enables higher performance at lower cost

Register Operands

- Arithmetic instructions use register operands
- MIPS has a 32 × 32-bit register file
  - Use for frequently accessed data
  - Numbered 0 to 31
  - 32-bit data called a “word”
- Assembler names
  - $t0, $t1, …, $t9 for temporary values
  - $s0, $s1, …, $s7 for saved variables
- *Design Principle 2:* Smaller is faster

  Compare to: main memory millions of locations

Register Operand Example

- C code:

```
f = (g + h) - (i + j);
```

  - f, …, j in $s0, …, $s4

- Compiled MIPS code:

```
add $t0, $s1, $s2
add $t1, $s3, $s4
sub $s0, $t0, $t1
```

Memory Operands

- Main memory used for composite data
  - Arrays, structures, dynamic data
- To apply arithmetic operations
  - Load values from memory into registers
  - Store result from register to memory
- Memory is byte addressed
  - Each address identifies an 8-bit byte
- Words are aligned in memory
  - Address must be a multiple of 4; PC <- PC +4
- MIPS is Big Endian
  - Most-significant byte at least address of a word
  - *c.f.* Little Endian: least-significant byte at least address

Registers vs. Memory

- Registers are faster to access than memory
- Operating on memory data requires loads and stores
  - More instructions to be executed
- Compiler must use registers for variables as much as possible
  - Only spill to memory for less frequently used variables
  - Register optimization is important!

Performance Issues

-- CPU-Performance Equation
-- Amdahl's Law

What Affects Performance?

- Algorithm
  - Determines number of operations executed
- Programming language, compiler, architecture
  - Determine number of machine instructions executed per operation □depends on ISA
- Processor and memory system
  - Determine how fast instructions are executed
- I/O system (including OS)
  - Determines how fast I/O operations are executed

Response Time and Throughput

- Response time
  - How long it takes to do a task
- Throughput
  - Total work done per unit time
    - e.g., tasks/transactions/… per hour
- How are response time and throughput affected by
  - Replacing the processor with a faster version?
  - Adding more processors?
- We'll study their estimation

- **Response time:** time between submission of a job (program P) and its completion (depends on overall system load)

This includes

--- I/O

--- Operating system time for managing programs, compile time, etc.

--- **CPU-time** includes time for executing the machine code for P, memory access time, procedure calls, and system time spent on P.

- Performance is proportional to the *inverse* of the CPU time.

What determines the execution time of a machine/assembly-level program *P* when it is run on a machine *M*?

- *P* consists of a number of machine-level instructions (IC: *instruction count*);
- Each machine instruction requires several clock cycles to complete (CPI: average number of *clock cycles per instruction*);
- Each clock cycle has certain time period (CCT: *clock cycle time*)

Thus, CPU-time = IC × CPI × CCT

(CPU Performance Equation)

Example (Compute CPI & CPU-time)

Problem Statement

A program P has 1000 M/L instructions and is being executed on a machine M running at 1 GHz clock speed.

ALU-op: 40%, Clock-cycles needed = 1
load-op: 20%, Clock-cycles needed = 2
store-op: 10%, Clock-cycles needed = 2
branch-op: 30%, Clock-cycles needed = 2

CPI = 1.60 | # IC = 1000
CCT = 1 GHz

Hence, CPU_time

$= IC * CPI * CCT$

$= 1000 * 1.60 * \frac{1}{10^9}$ sec

$= 1600 \times 10^{-9}$ sec $=$ **1600 ns**