CS 31007 Autumn 2019

COMPUTER ORGANIZATION AND ARCHITECTURE

Instructors

Section I - Rajat Subhra Chakraborty
Class Room NR 423

Section II - Bhargab B. Bhattacharya
Class Room NR 124

Indian Institute of Technology Kharagpur
Computer Science and Engineering

Previous Class

- ❖ Overview of the course
- ❖ Evolution and history of computer design
- ❖ Moore’s law
- ❖ Basic components of a computer
- ❖ Instruction Set Architecture (ISA)
- ❖ Computer organization and computer architecture: Bottom-up and Top-down view

Acknowledgement: Patterson and Hennessy; V. D. Agrawal; S. R. Sarangi

Today’s Agenda

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

Three Challenges

1. How to design efficient hardware (logic)?

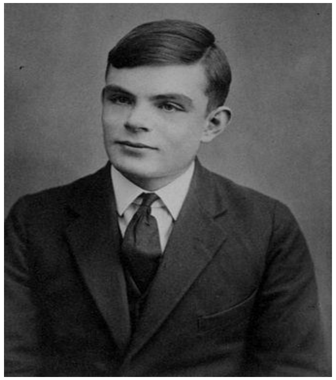
2. What is the simplest yet all powerful computer (computability)?

3. How should basic computer architecture be conceived?

Pioneers who answered these three questions

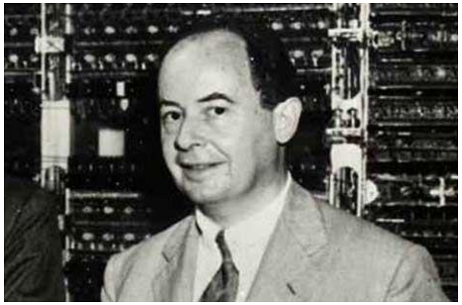
Claude E. Shannon
(1916-2001)

Logic design (basis of computer organization)



Alan Turing
(1912-1954)

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

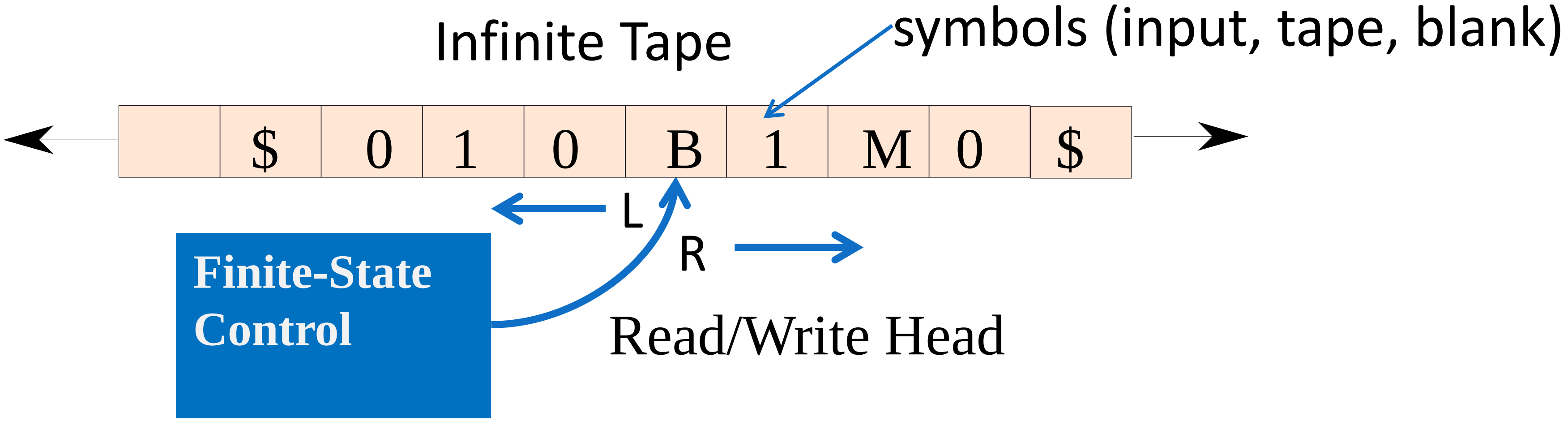


John von Neumann
(1903-1957)

Basic computer architecture

What is the simplest yet all powerful computer?

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



Infinite Tape

symbols (input, tape, blank)

	$	0	1	0	B	1	M	0	$

L

R

Finite-State Control

Read/Write Head

The tape head can only 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

- ❖ 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 Indian Civil Service and worked in Odisha

Turing Machine

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

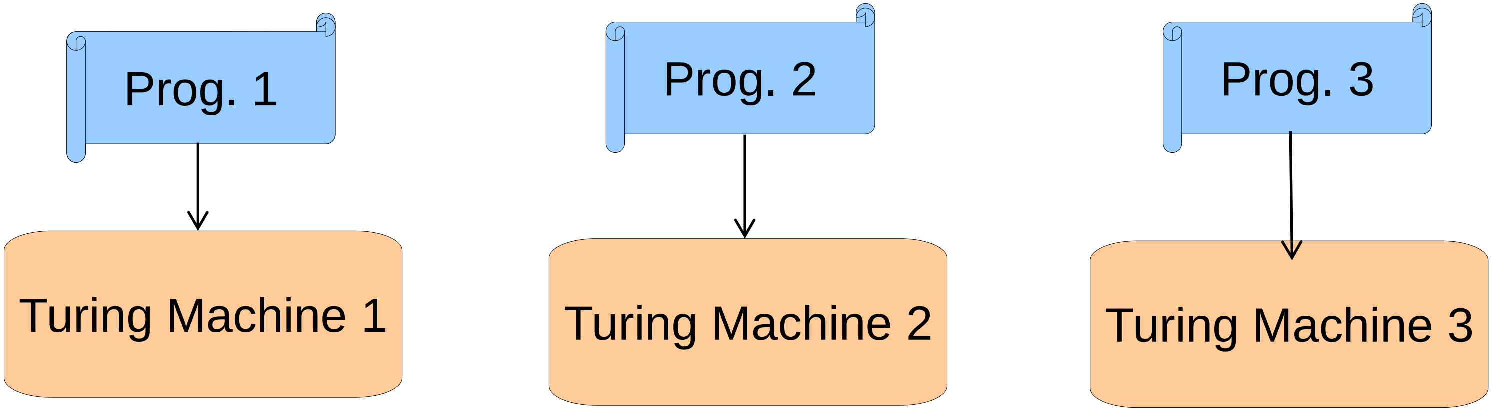
A. M. Turing (1936) , On Computable Numbers with an Application to the Entscheidungsproblem, *Proc. Royal Math. Soc.*, Ser. 2, Vol. 42, pp. 230-265, 1936.

- Extremely powerful

Church-Turing Conjecture (1936): Any procedure that is computable by paper-and-pencil methods (algorithm) can be solved by a Turing machine.

Universal Turing Machine

Church-Turing Conjecture (1936): Any procedure that is computable by paper-and-pencil method (algorithm) can be solved by a Turing machine.

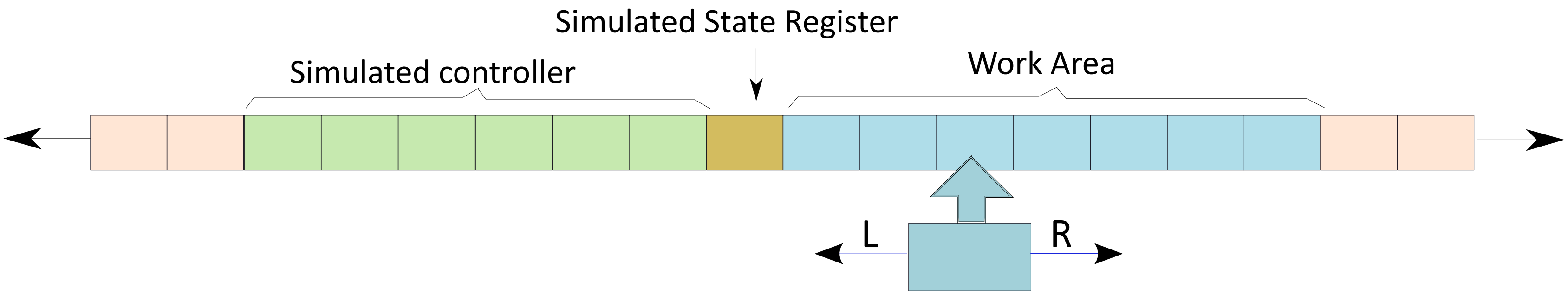


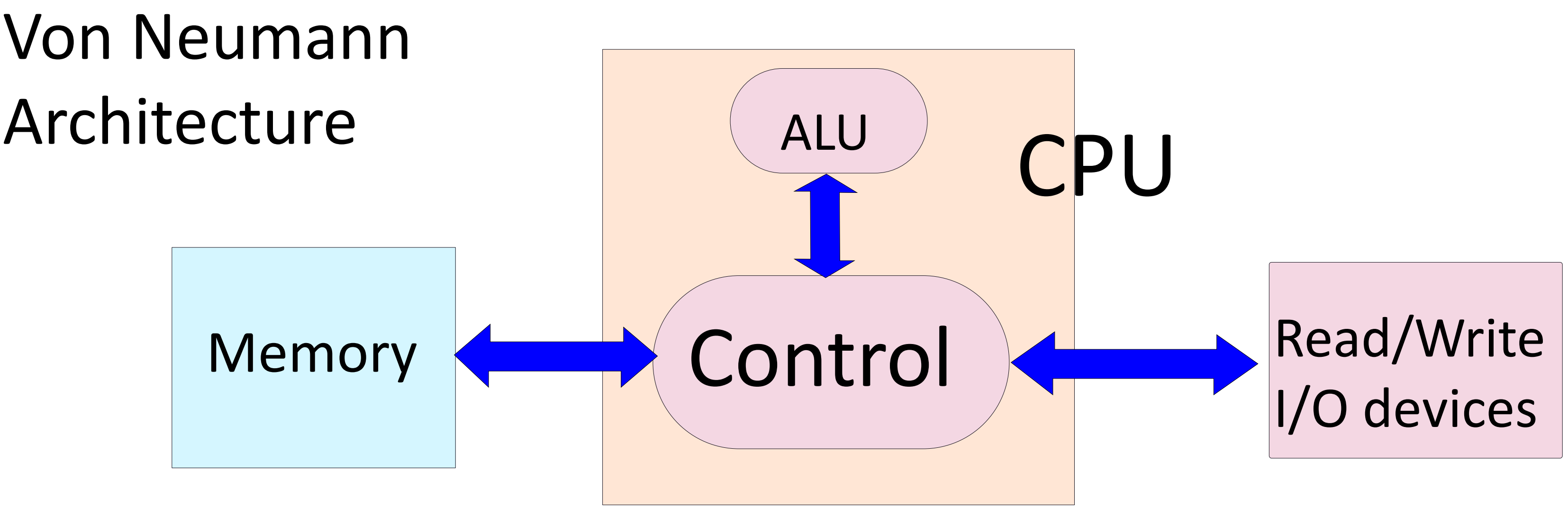
More general question:

Can we design a Universal Turing machine (UTM) that can simulate any Turing machine?

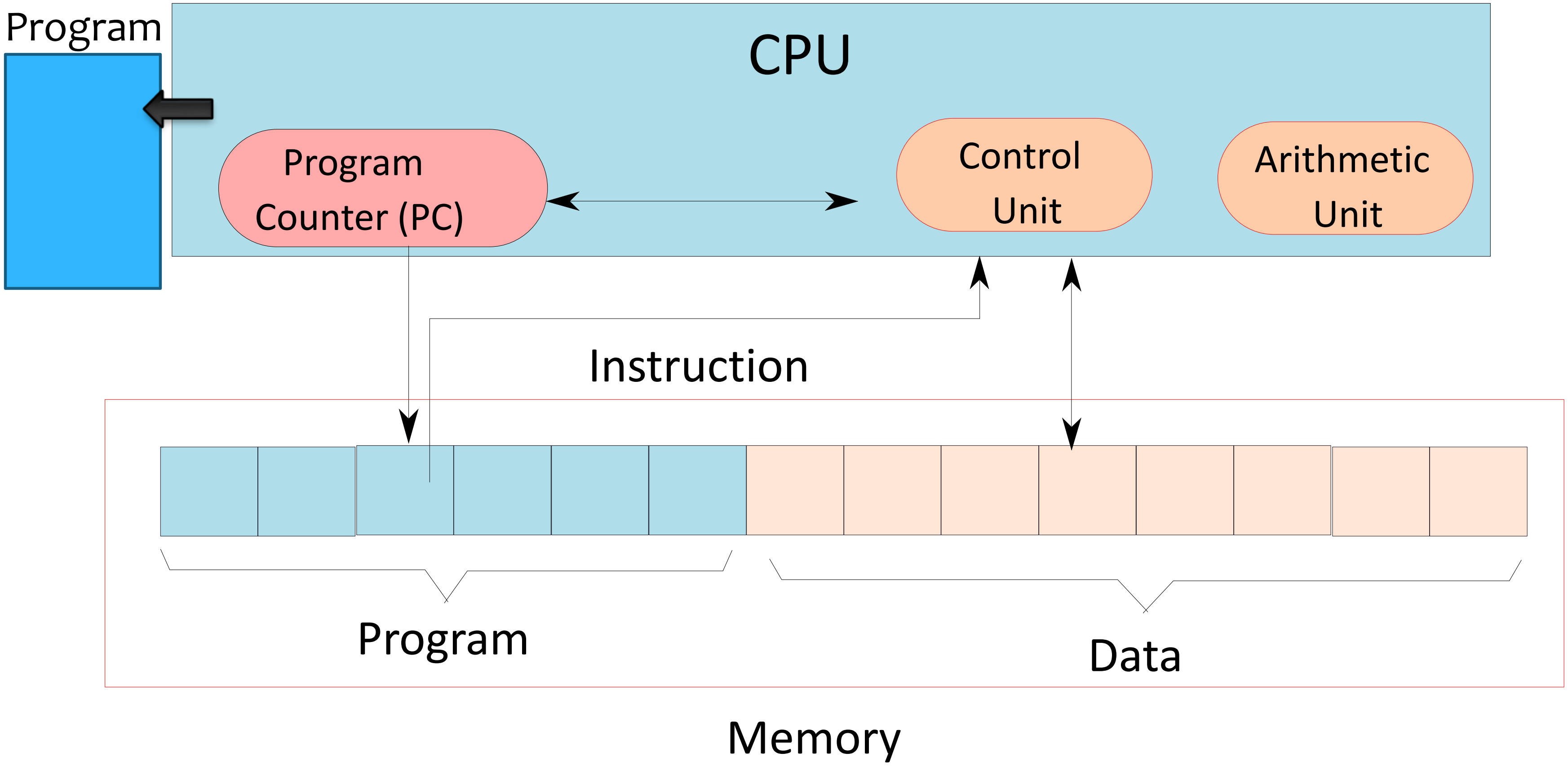
Universal Turing Machine

Controller and states are simulated on tape



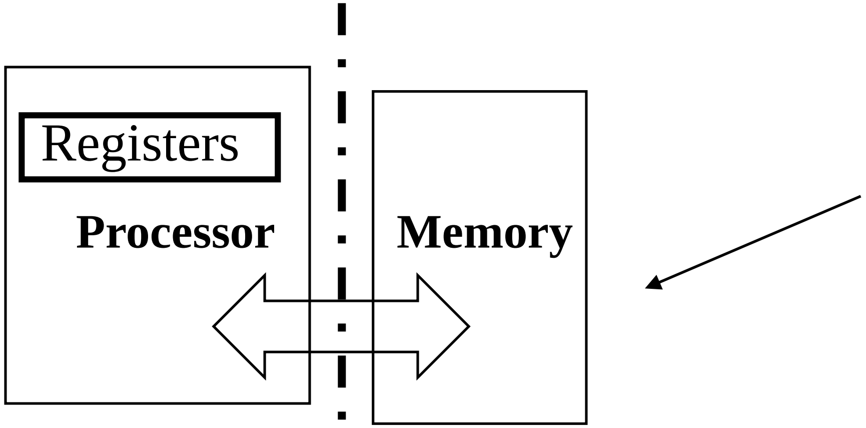


Computer Inspired fromTuring Machine



von Neumann Architecture (1945): Princeton Architecture

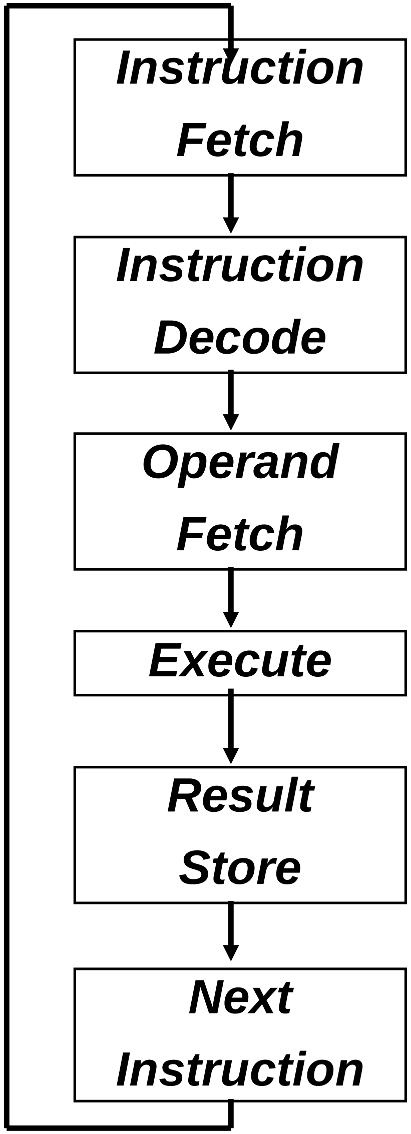
- Stored program concept
- Serves as the basis for almost all modern computers
- Instructions and data are just bits
- Programs (sequence of instructions) are stored in memory to be read or written just like data



memory for data, programs, compilers, systems programs

- **Fetch & Execute Cycle**
  - Program Counter points to the present Instruction to be fetched
  - Bits in the register "control" the subsequent actions
  - Fetch the “next” instruction and continue

Execution Cycle



Instruction Fetch

Instruction Decode

Operand Fetch

Execute

Result Store

Next Instruction

Obtain instruction from memory

Determine required actions and instruction size

Locate and obtain operand data from memory/register

Compute result value or status

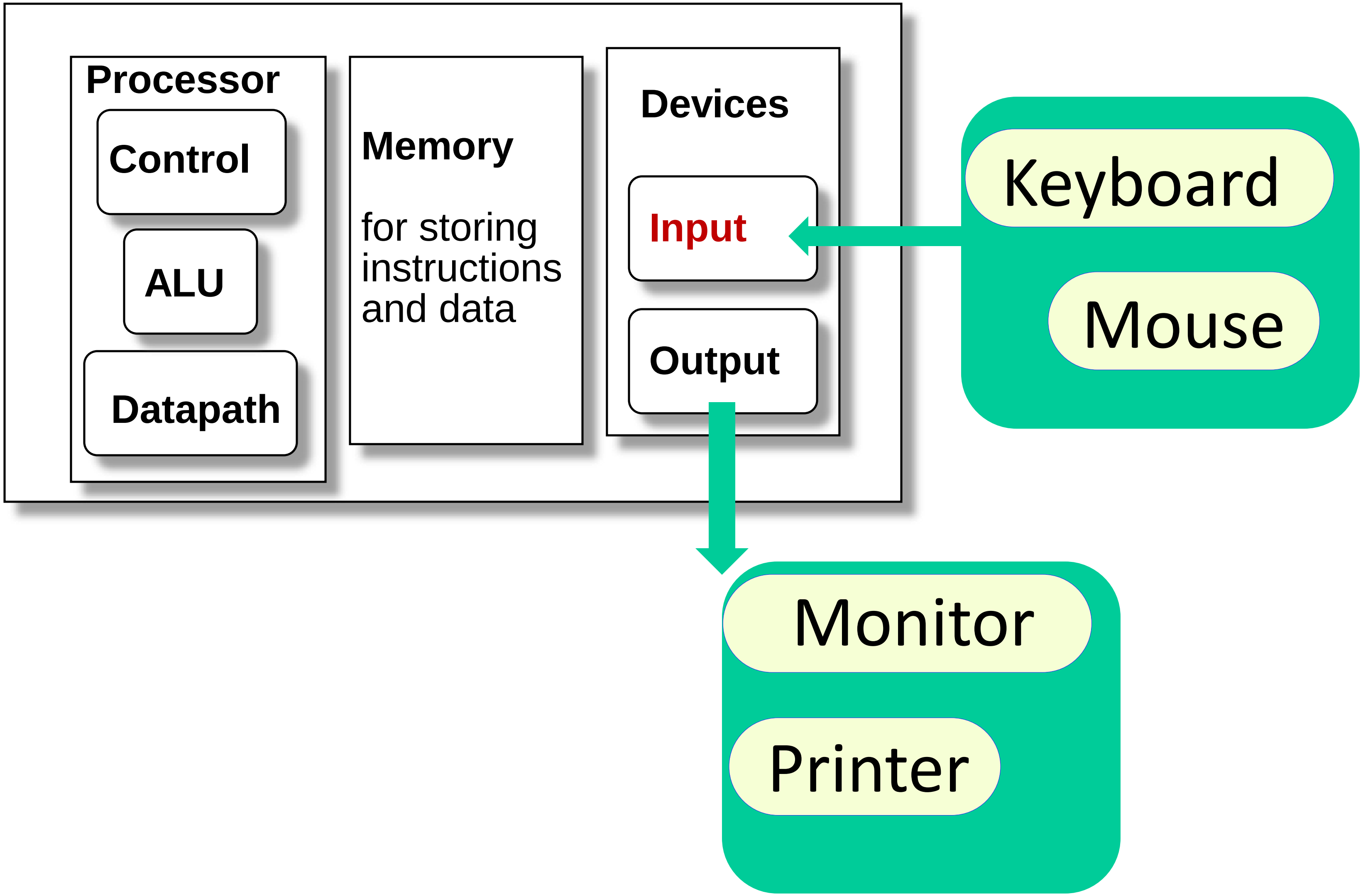
Deposit results in register/memory for later use

Determine successor instruction

Features of von Neumann Architecture

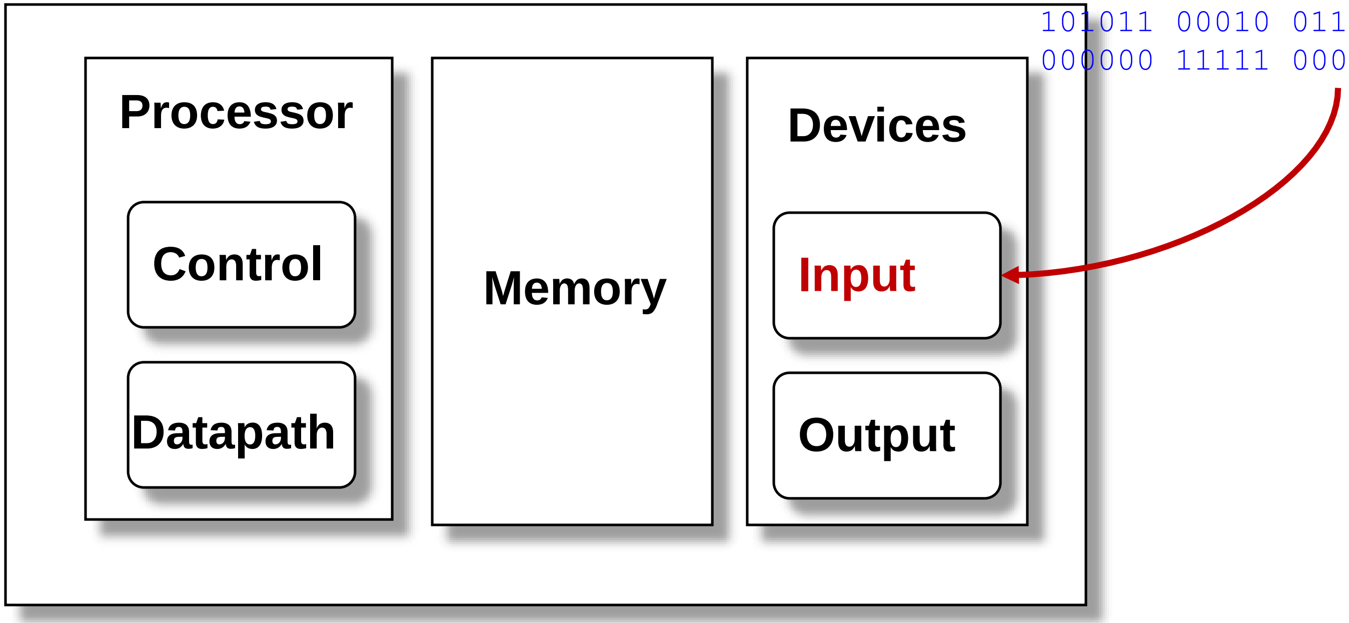
- Same physical memory to save instructions and data
- Instruction fetch and data transfer cannot be done concurrently; they need two clock cycles
- Simple architecture

Full Picture

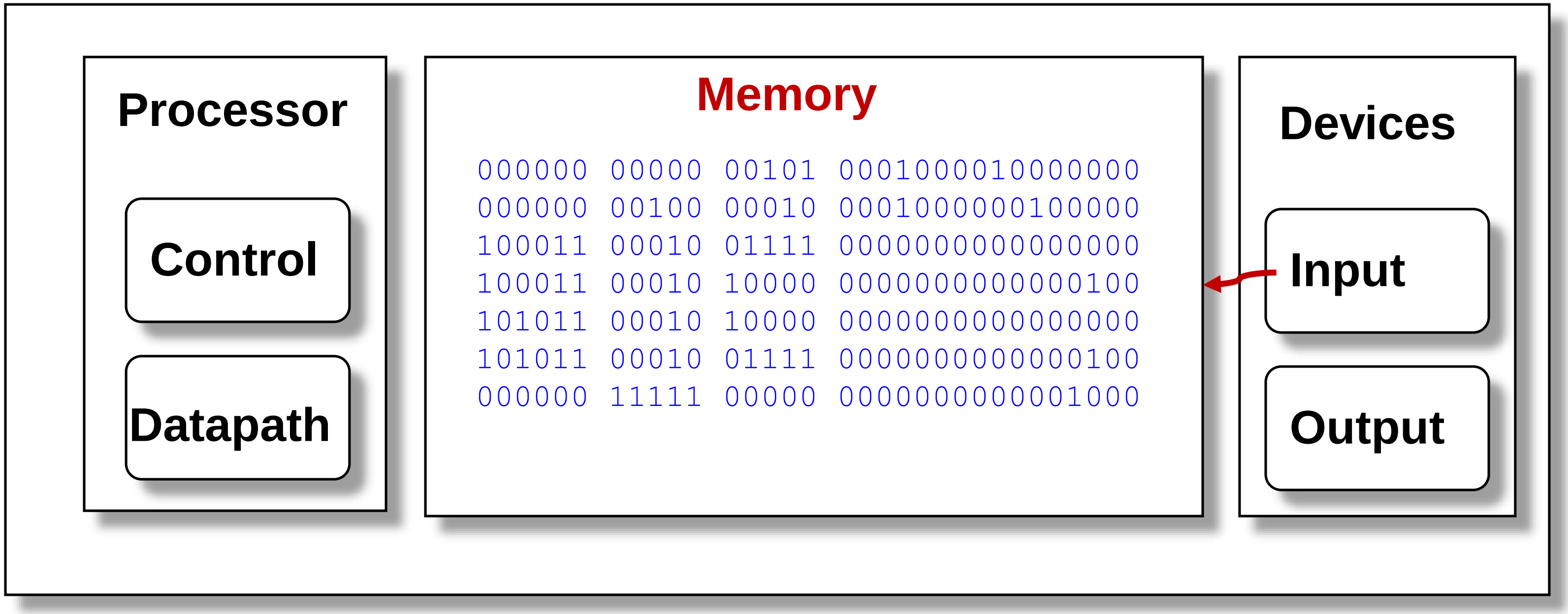


Load Input Binary

```
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
```



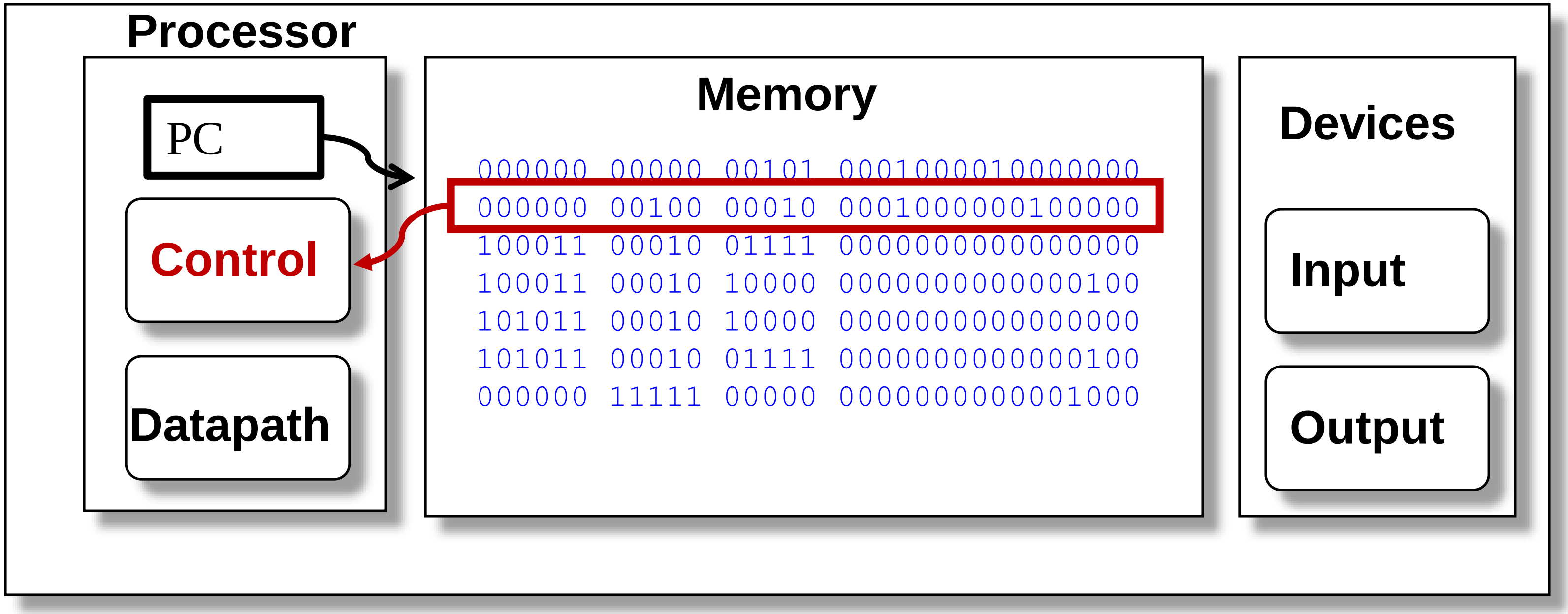
Code Stored in Memory



```
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
```

Processor Fetches an Instruction

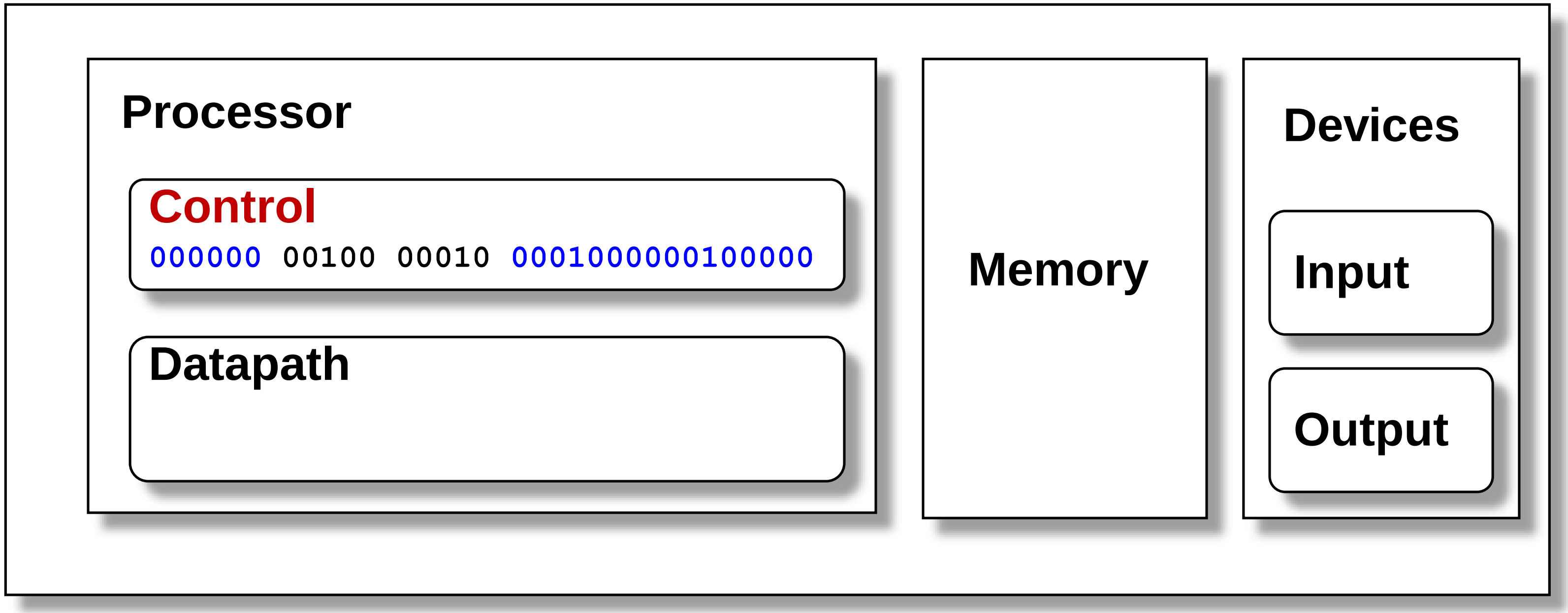
Processor fetches an instruction from memory pointed by Program Counter PC



```
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
```

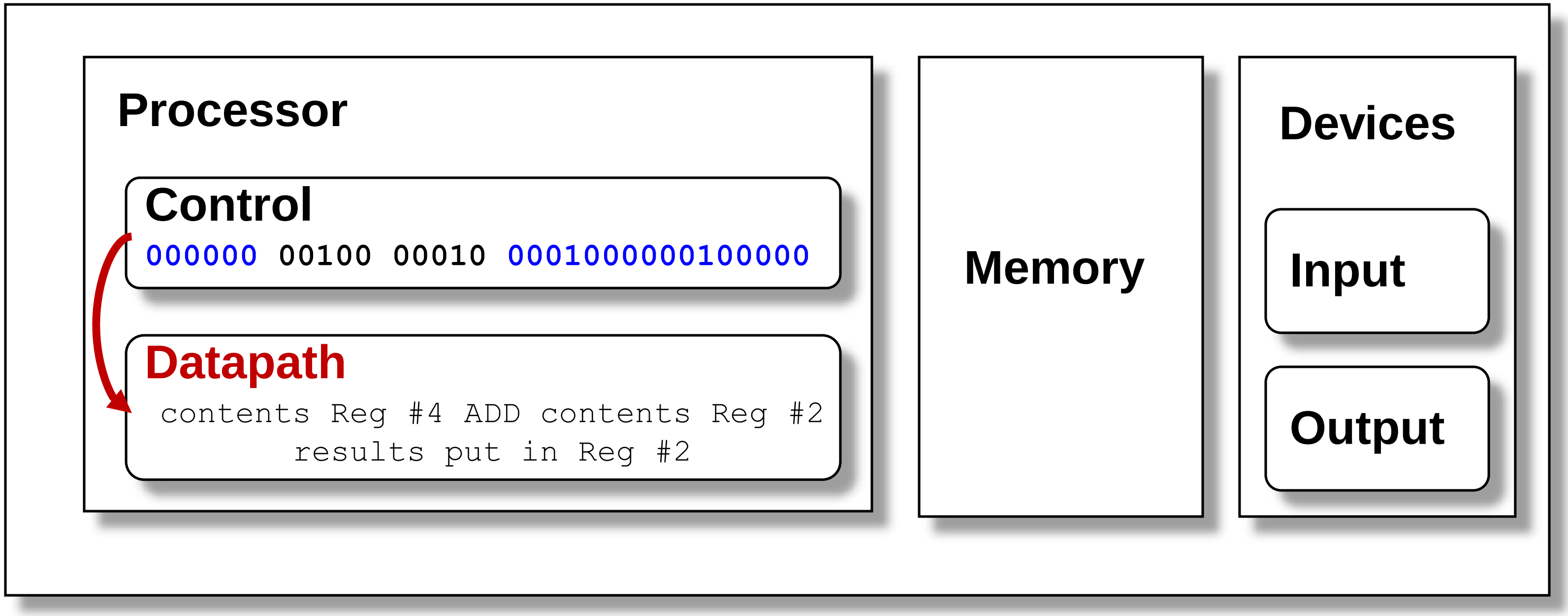
Where does it fetch from?

Control Decodes the Instruction



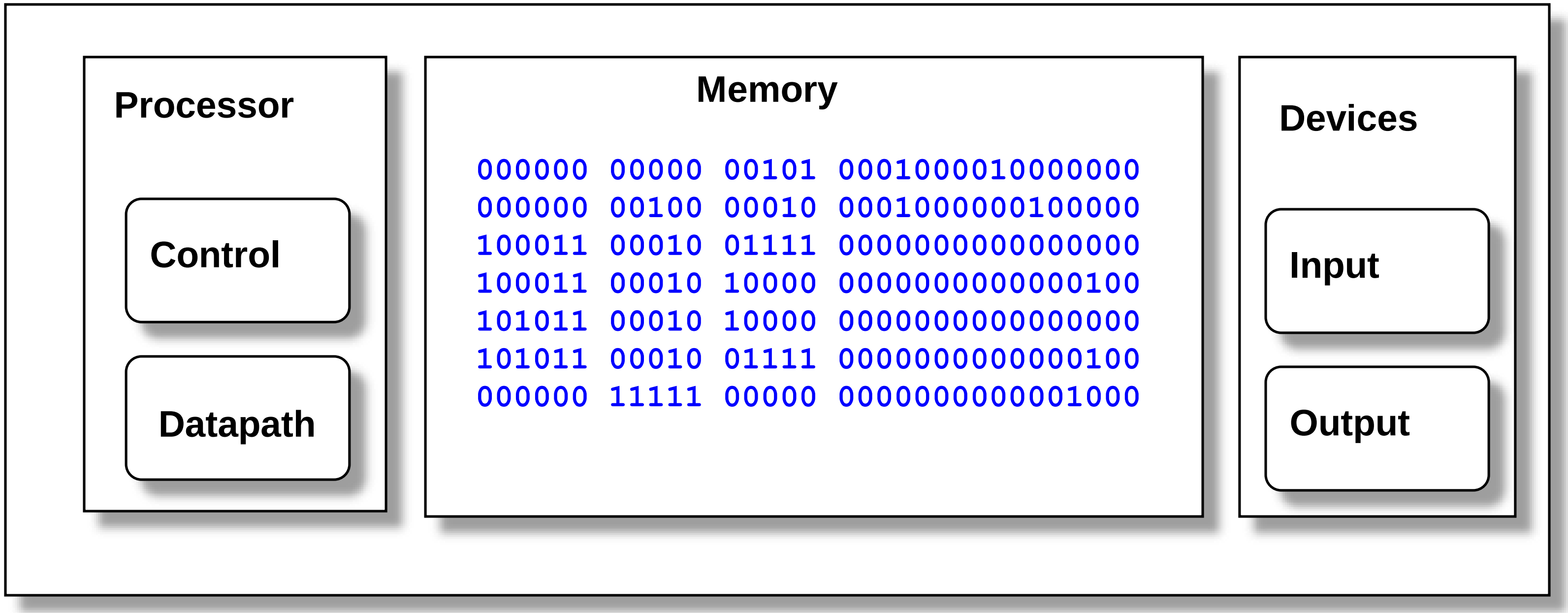
Control decodes the instruction to determine what to execute

Datapath Executes the Instruction

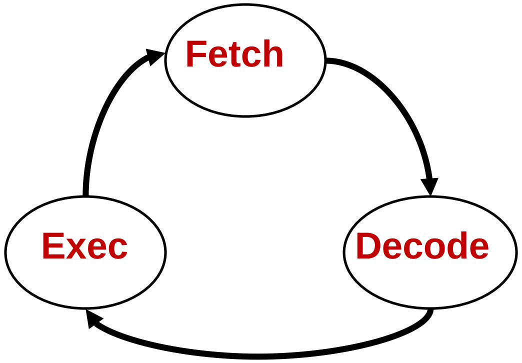


Datapath executes the instruction as directed by control

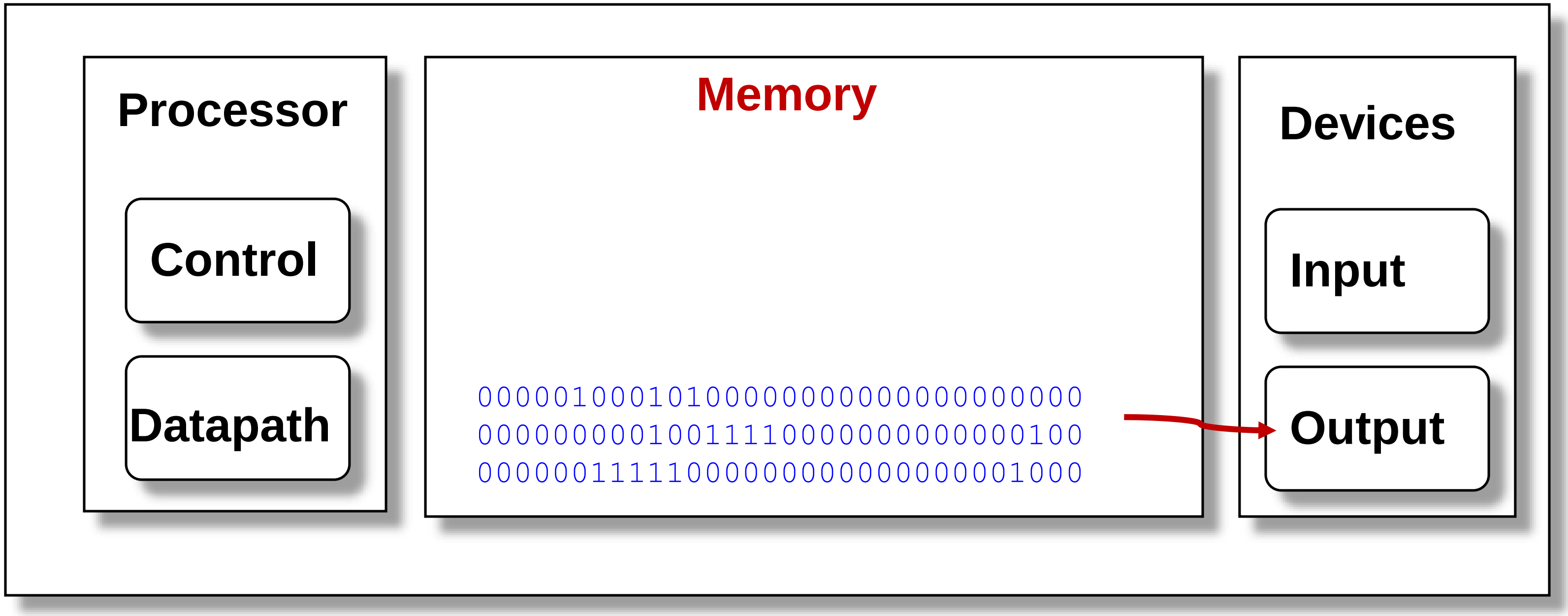
What Happens Next?



```
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
```

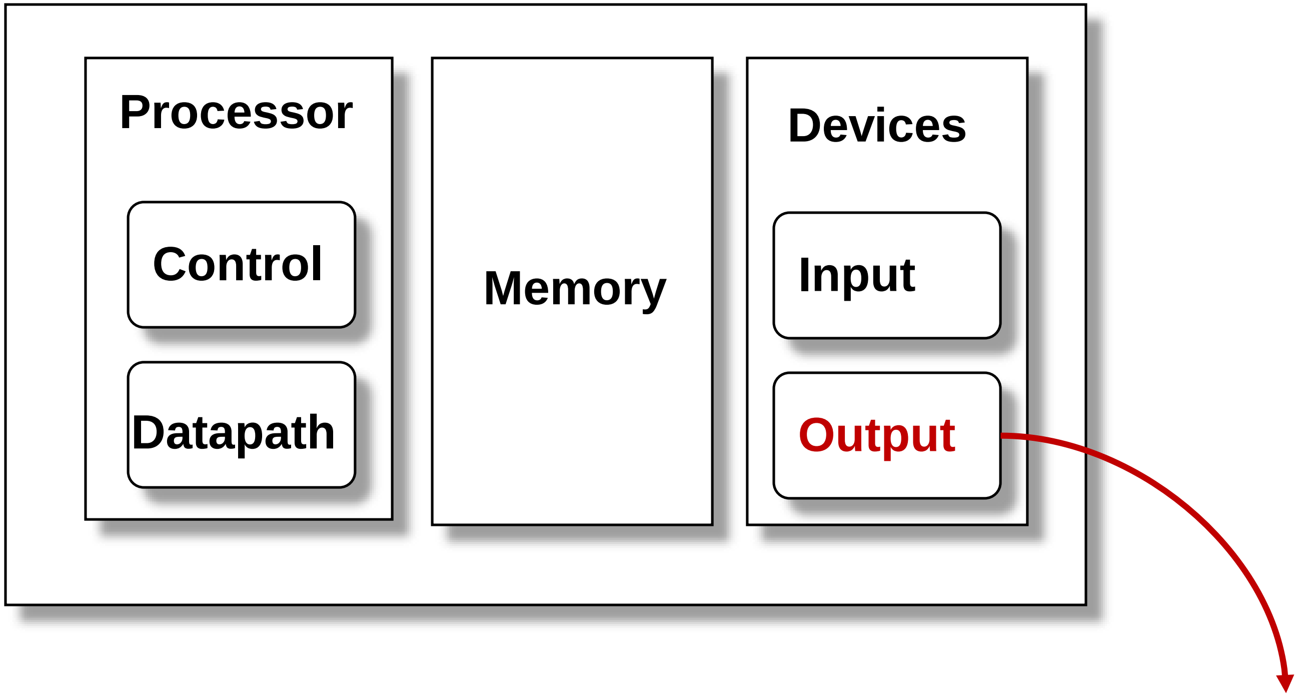


Output Data Stored in Memory



On program completion, results reside in memory

Output Device Outputs Data

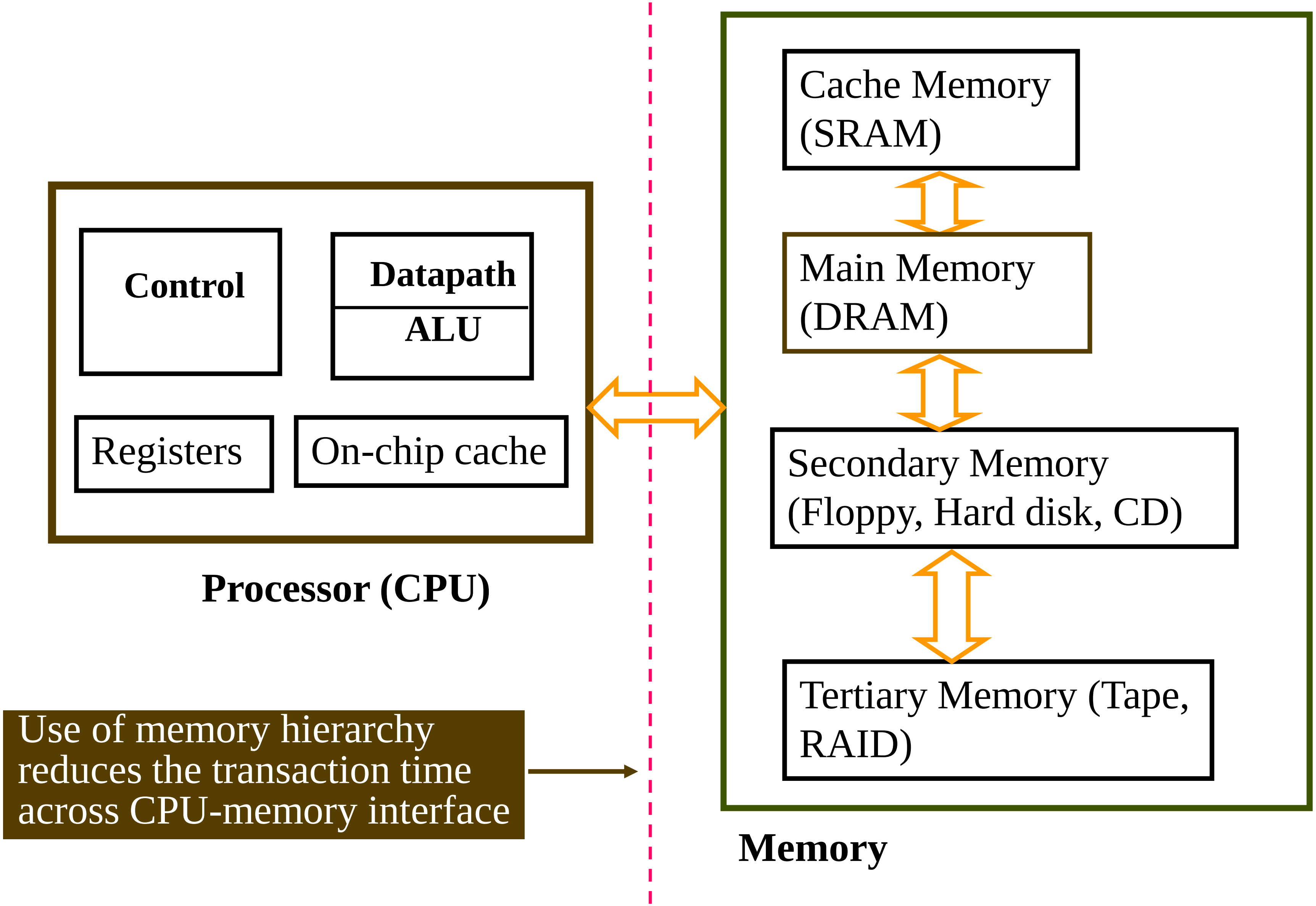


```
00000100010100000000000000000000
00000000010011110000000000000100
00000011111000000000000000001000
```

von Neumann Bottleneck

- von Neumann architecture uses the same memory for instructions (program) and data
- 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 restrict operands to registers (temporary storage in processor).

Complete View of Computer Architecture



Control

Datapath

ALU

Registers

On-chip cache

Processor (CPU)

Cache Memory (SRAM)

Main Memory (DRAM)

Secondary Memory (Floppy, Hard disk, CD)

Tertiary Memory (Tape, RAID)

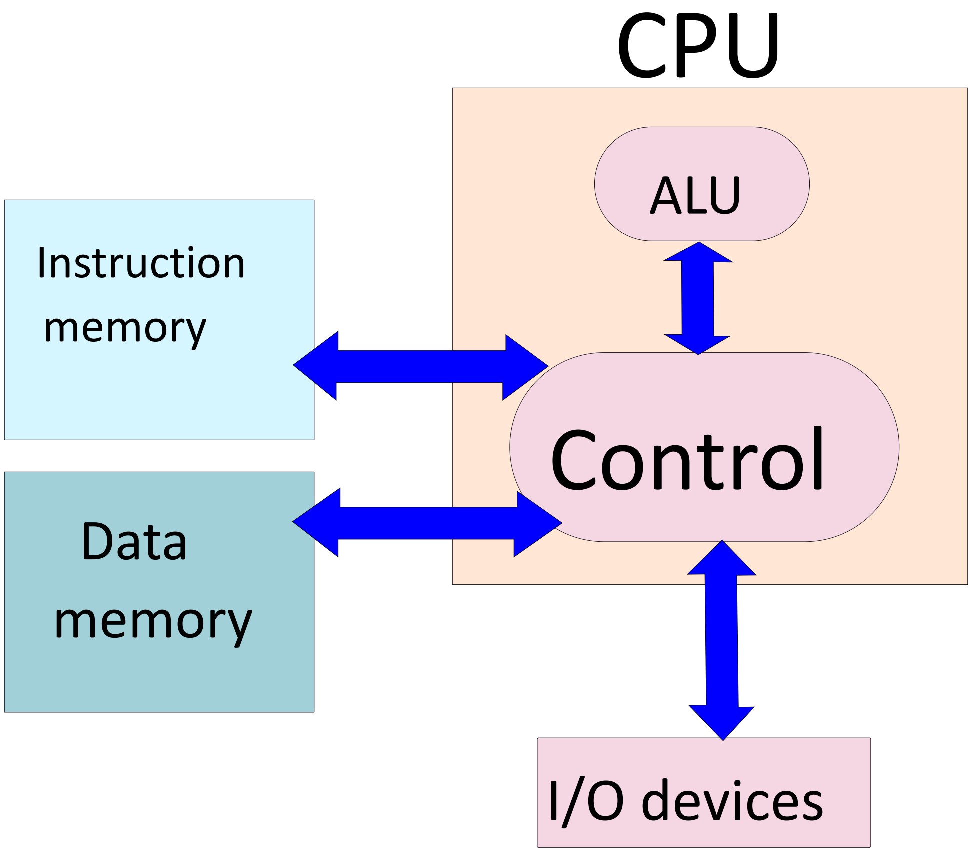
Memory

Use of memory hierarchy reduces the transaction time across CPU-memory interface

Inside the Processor (CPU)

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

Harvard Architecture



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

Summary: Elements of a Computer

- Memory (array of bytes) contains
  - The program, which is a sequence of instructions
  - The program data → variables and constants
- The program counter (PC) points to an instruction in a program
  - After executing an instruction, it points to the next instruction by default
  - A branch instruction makes the PC point to another instruction (not in sequence)
- CPU (Central Processing Unit) contains
  - Program counter, instruction execution units, arithmetic logic unit (ALU)

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 Archiecture (ISA)

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

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

op	s1	t0	Immediate
6 bits	5 bits	5 bits	16 bits
001000	10001	01000	0000000000000101

Total length of the instruction – 16 bits; opcode: 6 bits; $t0, $s1 -> CPU registers;

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

Features of Instruction Set Archiecture (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?

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 (below) and compiler design (up)

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** that 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.

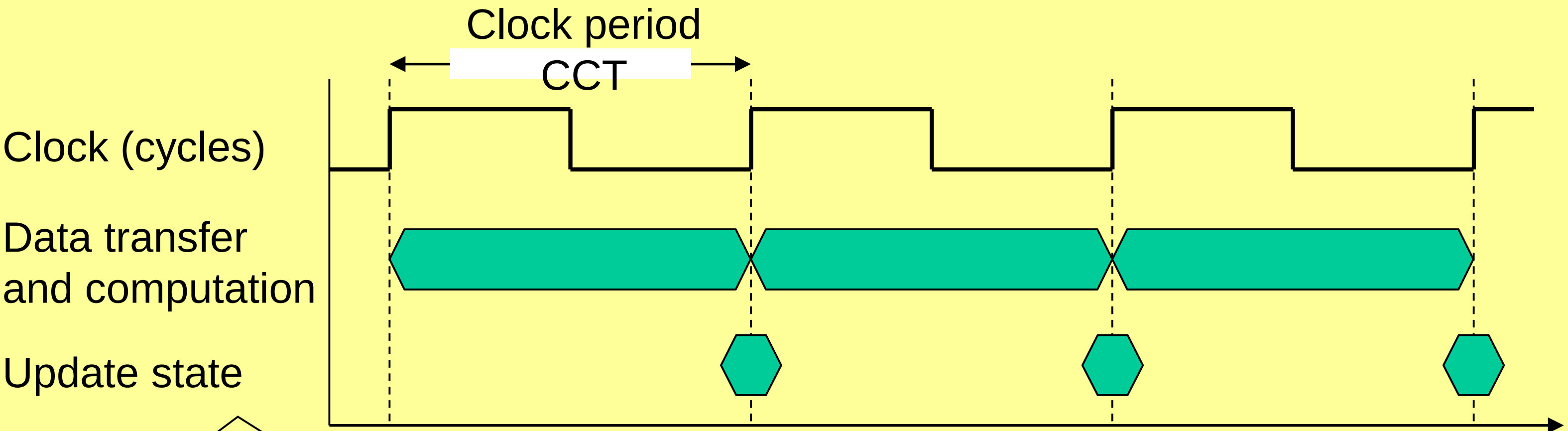
What determines the execution time of a machine/assembly-level program P when run on a machine M?

- 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 Clocking

- CPUs are driven by constant-rate system clocks:
  - 100 MHz clock frequency (*f*) means the system clock ticks 100 million times every second:



One Clock Cycle Time (“Tick”), i.e., CCT
= 1/100,000,000 *sec*
=1/100 *microsec* = 10 *nanosec*

CCT = 1/*f*

Board Work

Examples of Performance Evaluation

Amdahl’s Law

CPU Time

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

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

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

CPU Time Example

- Computer A: 2GHz clock, 10s CPU time
- Designing Computer B
  - Aim for 6s CPU time
  - Can do faster clock, but causes 1.2 × clock cycles
- How fast must Computer B clock be?

$$\text{Clock Rate}_B = \frac{\text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times \text{Clock Cycles}_A}{6s}$$

$$\text{Clock Cycles}_A = \text{CPU Time}_A \times \text{Clock Rate}_A$$

$$= 10s \times 2\text{GHz} = 20 \times 10^9$$

$$\text{Clock Rate}_B = \frac{1.2 \times 20 \times 10^9}{6s} = \frac{24 \times 10^9}{6s} = 4\text{GHz}$$

Instruction Count and CPI

$$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}$$

$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

- Instruction Count for a program
  - Determined by program, ISA and compiler
- Average cycles per instruction
  - Determined by CPU hardware
  - If different instructions have different CPI
    - Average CPI affected by instruction mix

CPI Example

- Computer A: Cycle Time = 250ps, CPI = 2.0
- Computer B: Cycle Time = 500ps, CPI = 1.2
- Same ISA
- Which is faster, and by how much?

$$\text{CPU Time}_A = \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A$$

$$= \text{I} \times 2.0 \times 250\text{ps} = \text{I} \times 500\text{ps}$$

A is faster…

$$\text{CPU Time}_B = \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B$$

$$= \text{I} \times 1.2 \times 500\text{ps} = \text{I} \times 600\text{ps}$$

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{\text{I} \times 600\text{ps}}{\text{I} \times 500\text{ps}} = 1.2$$

…by this much

CPI in More Detail

- If different instruction classes take different numbers of cycles

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

  - Weighted average CPI

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

CPI Example

- Alternative compiled code sequences using instructions in classes A, B, C

Class	A	B	C
CPI for class	1	2	3
IC in sequence 1	2	1	2
IC in sequence 2	4	1	1

- Sequence 1: IC = 5
  - Clock Cycles
= 2×1 + 1×2 + 2×3
= 10
  - Avg. CPI = 10/5 = 2.0

- Sequence 2: IC = 6
  - Clock Cycles
= 4×1 + 1×2 + 1×3
= 9
  - Avg. CPI = 9/6 = 1.5

Performance Summary

The BIG Picture: CPU Performance Equation

$$\textbf{CPU-time} = \textbf{IC} \times \textbf{CPI} \times \textbf{CCT}$$

- Performance depends on
  - Algorithm: affects IC, possibly CPI
  - Programming language: affects IC, CPI
  - Compiler: affects IC, CPI
  - Instruction set architecture: affects IC, CPI, CCT
  - CPI is also affected by memory hierarchy, pipelining; CCT is affected by logic design, technology

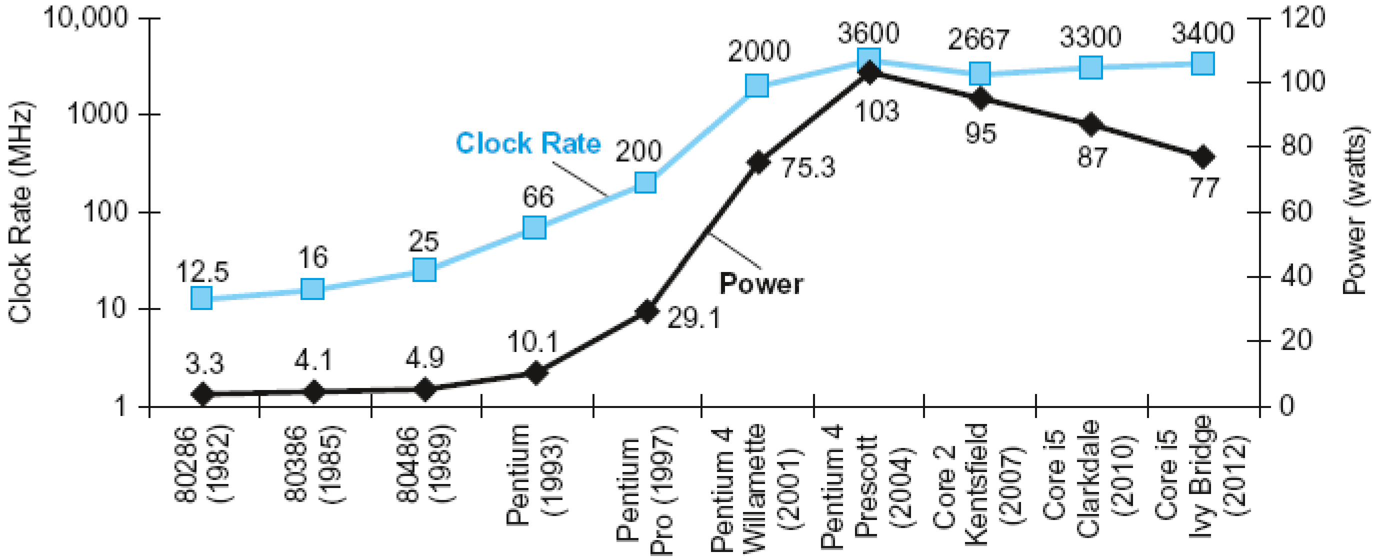
MIPS as Performance Measure

MIPS = Millions of Instructions per Second

$$= \frac{\text{Instruction Count (IC) of a program P}}{\text{Execution time of P in seconds} \times 10^{6}}$$

Power Trends

§1.7 The Power Wall



- In CMOS IC technology

$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

×30

5V → 1V

×1000

Reducing Power

- Suppose a new CPU has
  - 85% of capacitive load of old CPU
  - 15% voltage and 15% frequency reduction

$$\frac{P_{new}}{P_{old}} = \frac{C_{old} \times 0.85 \times (V_{old} \times 0.85)^2 \times F_{old} \times 0.85}{C_{old} \times V_{old}{}^2 \times F_{old}} = 0.85^4 = 0.52$$

- The power wall
  - We can’t reduce voltage further
  - We can’t remove more heat
- How else can we improve performance?

Multiprocessors

- Multicore microprocessors
  - More than one processor per chip
  - Clock frequency limited
- Requires explicitly parallel programming
  - Compare with instruction level parallelism
    - Hardware executes multiple instructions at once
    - Hidden from the programmer
  - Hard to do
    - Programming for performance
    - Load balancing
    - Optimizing communication and synchronization

Concluding Remarks

- Cost/performance is improving
  - Due to underlying technology development
- Hierarchical layers of abstraction
  - In both hardware and software
- Instruction set architecture
  - The hardware/software interface
- Execution time: the best performance measure
- Power is a limiting factor
  - Use parallelism to improve performance

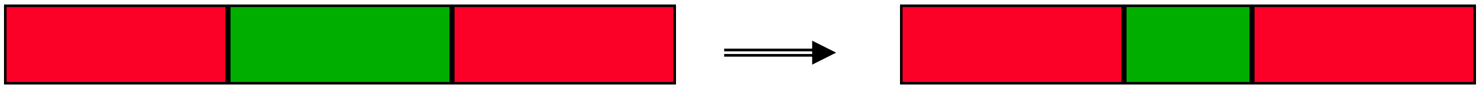
How to enhance performance of a machine by “Enhancement”?

→ Amdahl’s Law

Amdahl's Law

Speedup due to enhancement E:

$$\text{Speedup(E)} = \frac{\text{ExTime w/o E}}{\text{ExTime w/ E}} = \frac{\text{Performance w/E}}{\text{Performance w/o E}}$$



Suppose that enhancement E accelerates a fraction F of the task by a factor S, and the remainder of the task is unaffected

Amdahl's Law

$$\text{ExTime}_{\text{new}} = \text{ExTime}_{\text{old}} \times \left[(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right]$$

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

Amdahl's Law

- Floating point instructions improved to run 2X; but only 10% of actual instructions are FP

$\text{ExTime}_{\text{new}} =$

$\text{Speedup}_{\text{overall}} =$

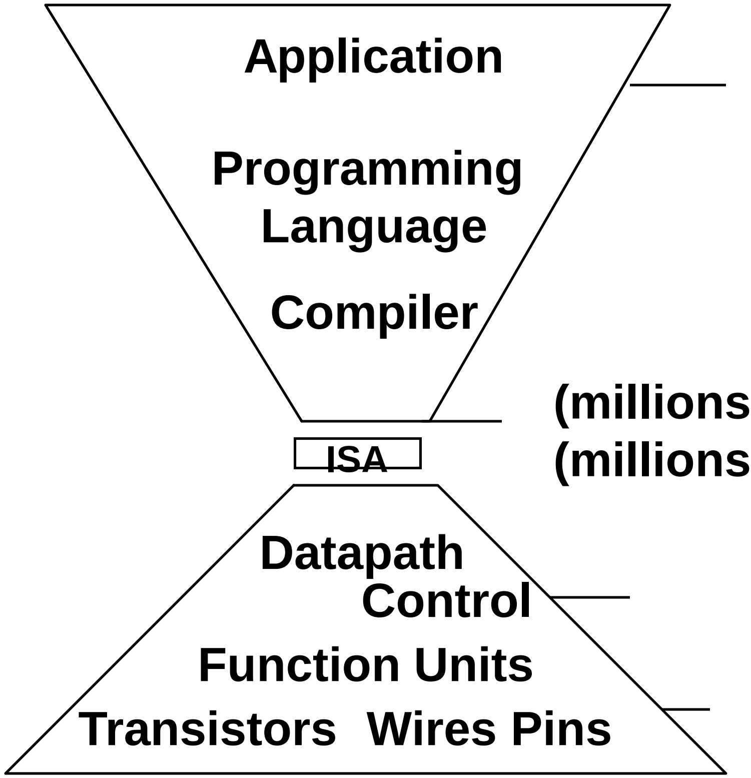
Amdahl's Law

- **Floating point instructions improved to run 2X; but only 10% of actual instructions are FP**

$$\text{ExTime}_{new} = \text{ExTime}_{old} \text{ x } (0.9 + .1/2) = 0.95 \text{ x } \text{ExTime}_{old}$$

$$\text{Speedup}_{overall} = \frac{1}{0.95} = 1.053$$

Metrics of Performance



Application

Programming Language

Compiler

ISA

Datapath

Control

Function Units

Transistors Wires Pins

Throughput per month
Operations per second

(millions) of Instructions per second: MIPS
(millions) of (FP) operations per second: MFLOP/s

Megabytes per second

Cycles per second (clock rate)

ISA: RISC *versus* CISC

- RISC (Reduced Instruction Set Computing)
  - Keep the instruction set small and simple
  - Fixed instruction lengths
  - Load-store instruction sets
  - Limited addressing modes
  - Limited operations
  - CPI low

Advantage: makes the hardware simple and fast; decoding simple; pipelining easy

Performance is optimized focused on software

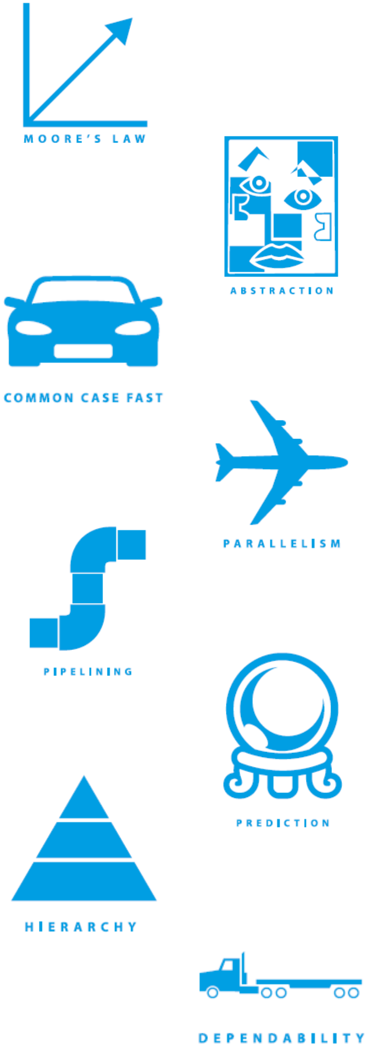
RISC Example: MIPS, Sun SPARC, HP PA-RISC, IBM PowerPC, Alpha, RISC-V, ARM

- CISC - Complex Instruction Set Computer
- Examples of CISC processors are Intel x86 CPUs, System/360, VAX, AMD.

Eight Great Ideas

§1.2 Eight Great Ideas in Computer Architecture

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



Next Class

- Introducing MIPS Assembly Language