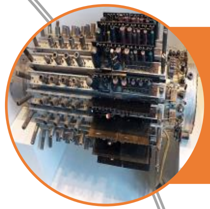
Lecture 3: All about Computer

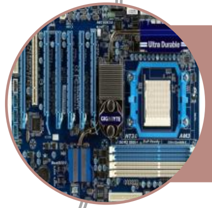
"Everything" about modern computer

Overview



Brief History of Computers

Current Trends

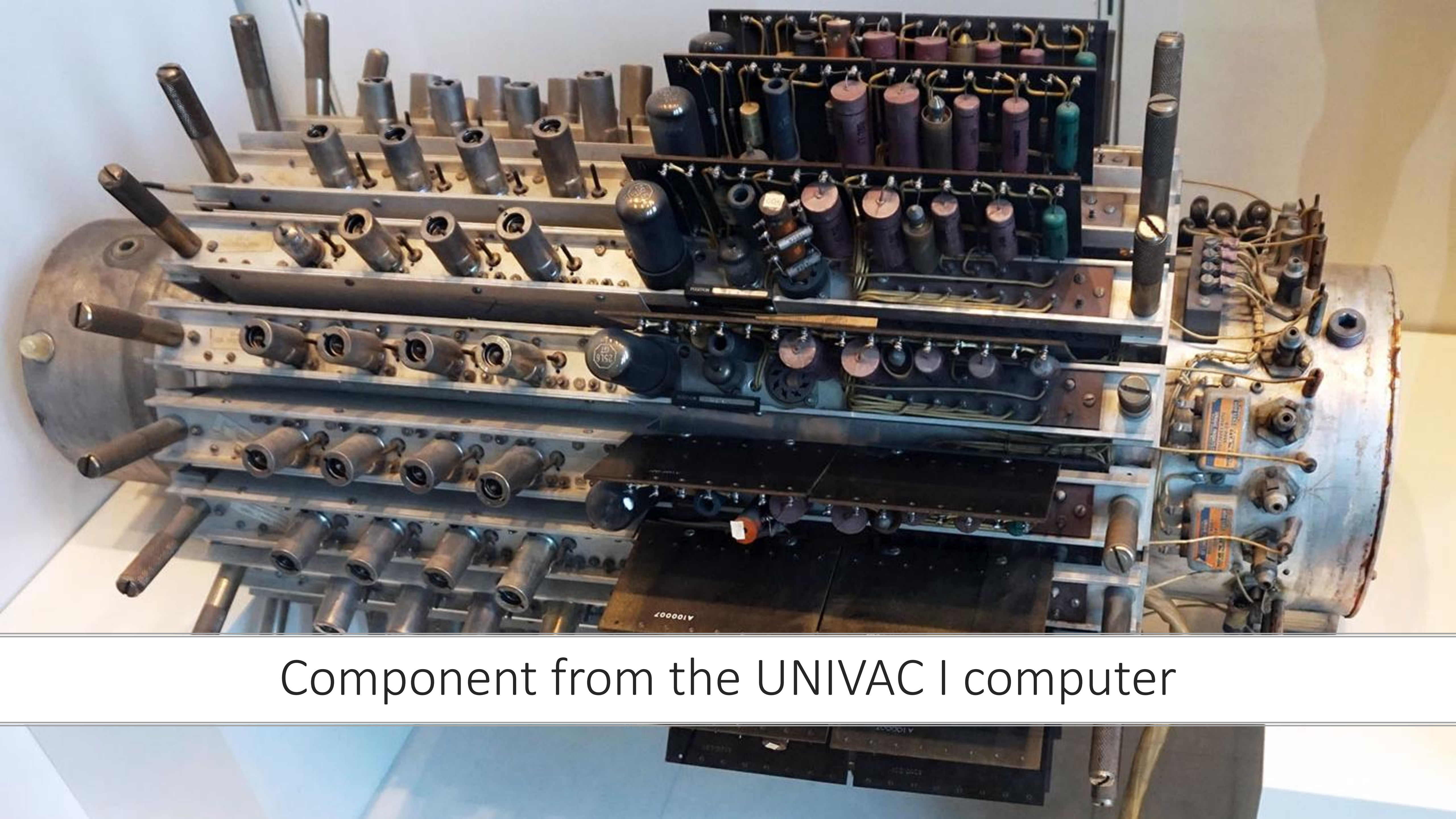


Computer Organization

Code Execution

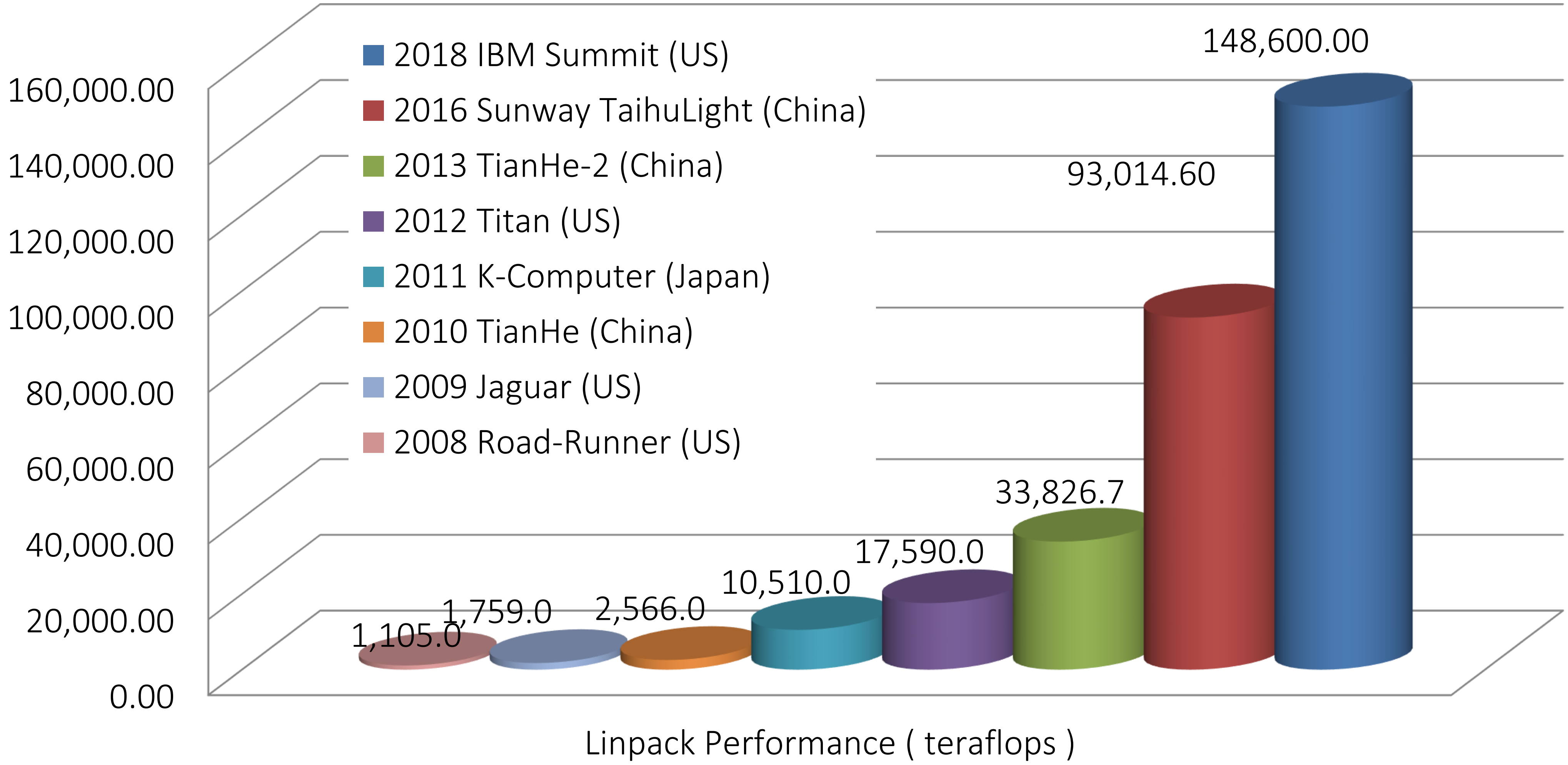
The Brief History: Computer / Processor

Year	Name	Speed	Remarks
1946	ENIAC	~1900 additions/sec	First electronic computer
1951	UNIVAC	~2000 additions/sec	First commercial computer
1964	IBM 360	500k ops/sec	Best known mainframe
1965	PDP-8	330k ops/sec	First minicomputer
1971	Intel 4004	100k ops/sec	First microprocessor
1977	Apple II	200k ops/sec	"First" PC
1981	IBM PC (Intel 8088 + MS-DOS)	240k ops/sec	Dominated market since then
2003	Intel Pentium 4	6G flops	"Last" unicore
2011	Intel Core i7	~120G flops	6 cores
2019	AMD Rizen R7 / Intel Core i9	~1800G flops	16 cores



Component from the UNIVAC I computer

The Brief History: Supercomputer



The Brief History: Embedded

Everywhere

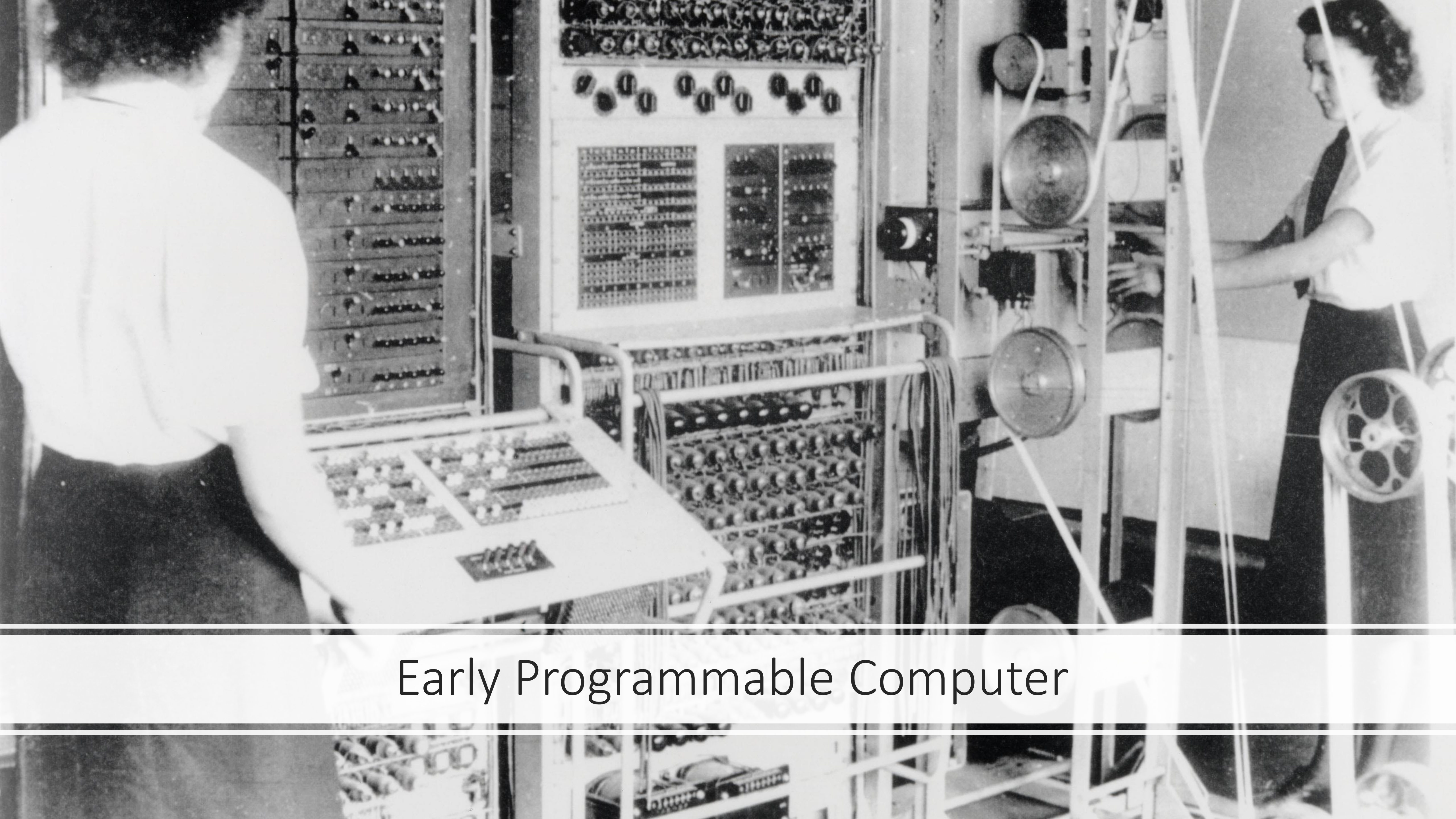
- Smart-phone
- Game consoles
- DVD / Blue-Ray player
- Car, Fridge, Washing Machine……… etc etc

The First Computers

Marble Adding Machine

What are the elements

– Abstraction of state
– Means of mutating state
– Controlling logic



Early Programmable Computer

Electric Computers

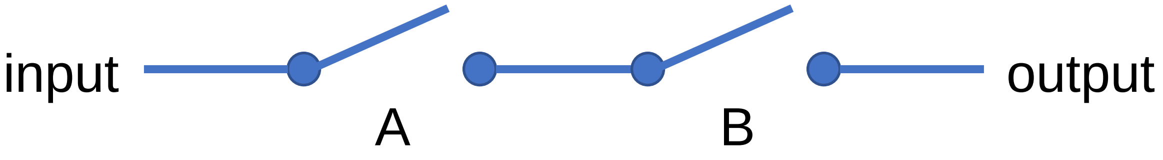
- Use voltage of electricity following through wires to represent 1 and 0

Controlling Logic

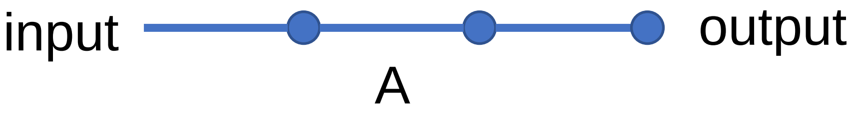
How to control electricity?

– Use switches

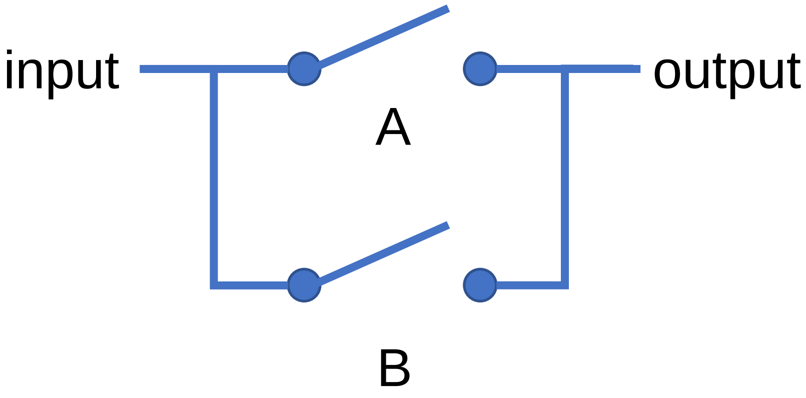
A and B



not A



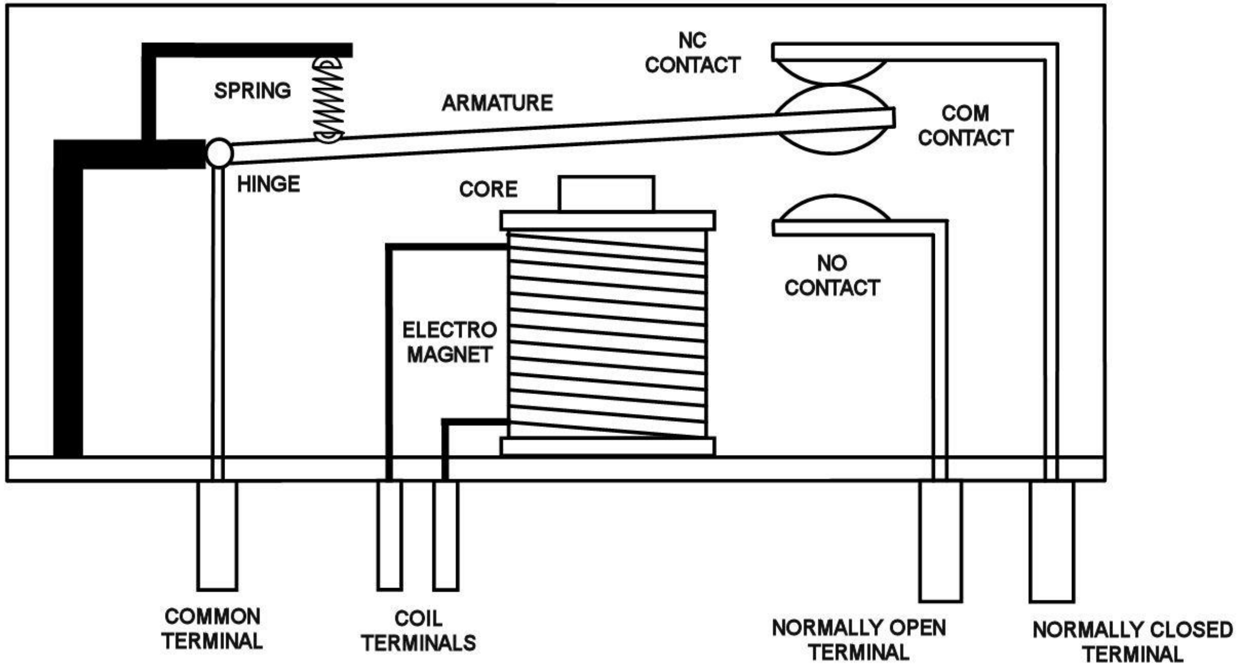
A or B

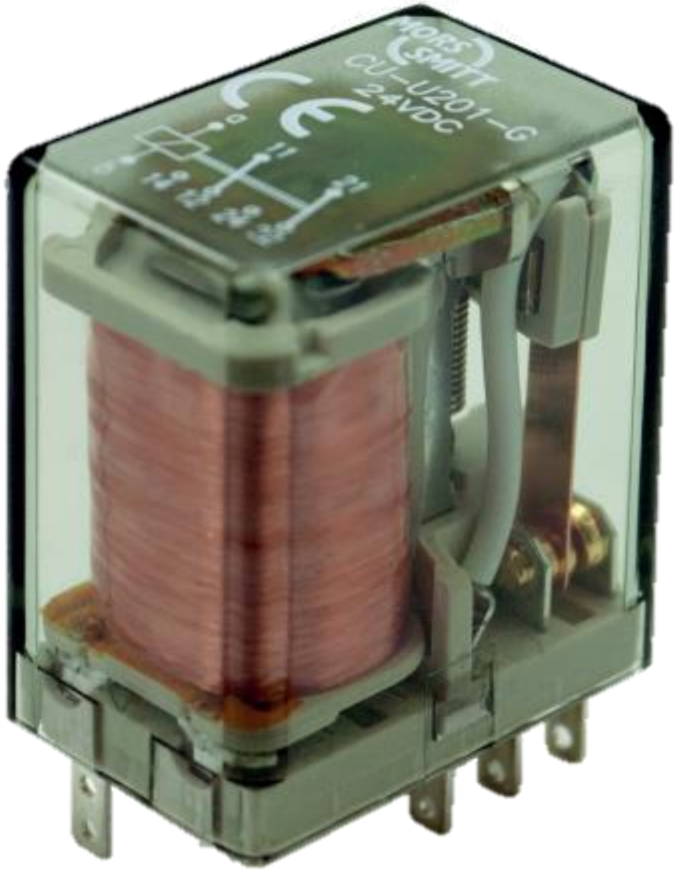


Activating Switches

Using Relays

– Electromagnet attracts the switch

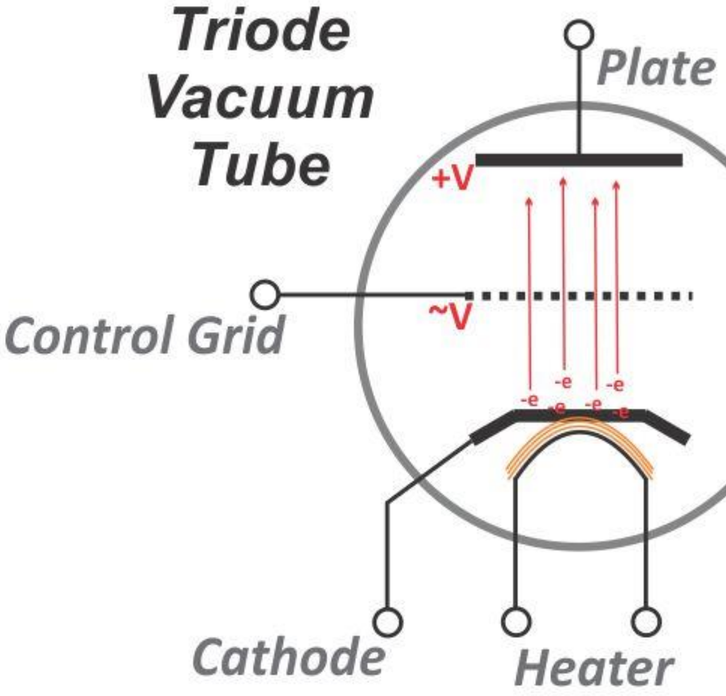


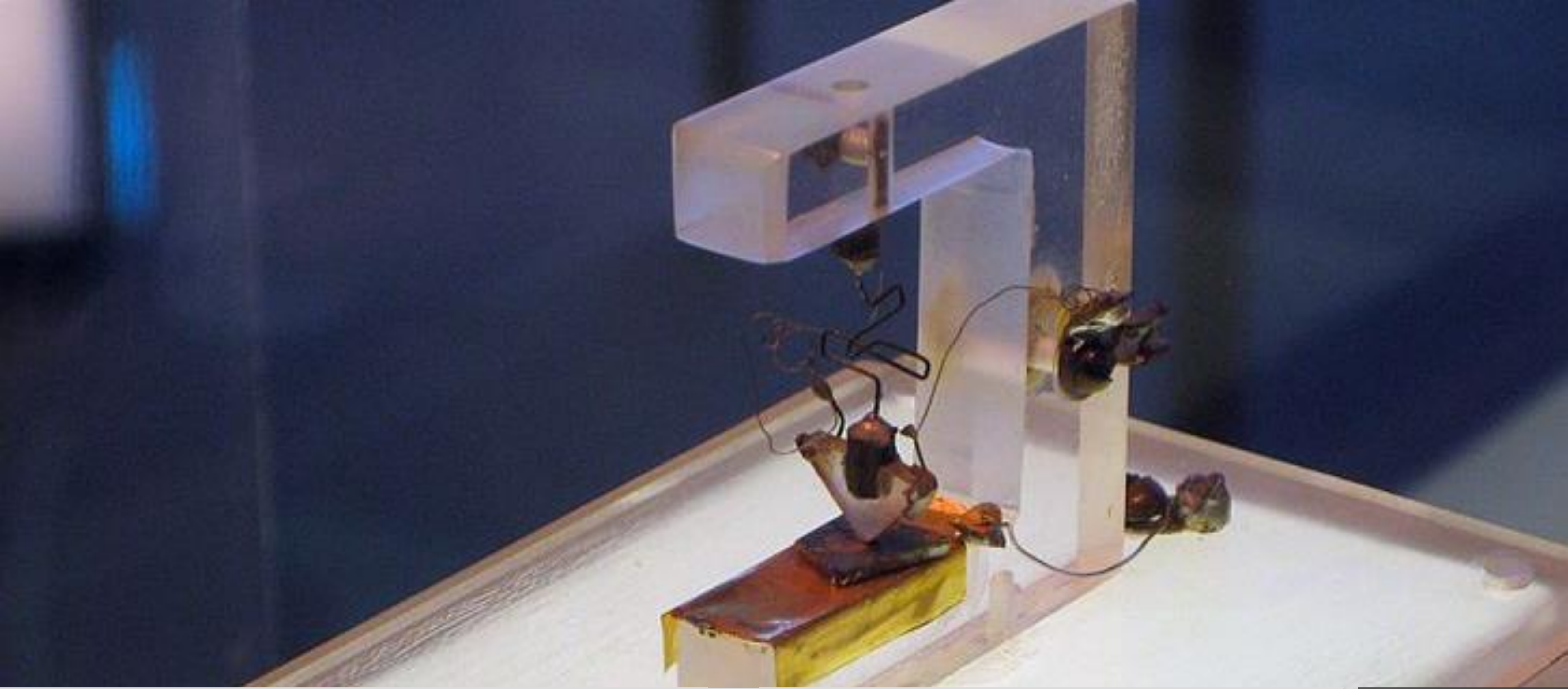


Activating Switches

Using Vacuum Tubes

– Heated metal releases electrons



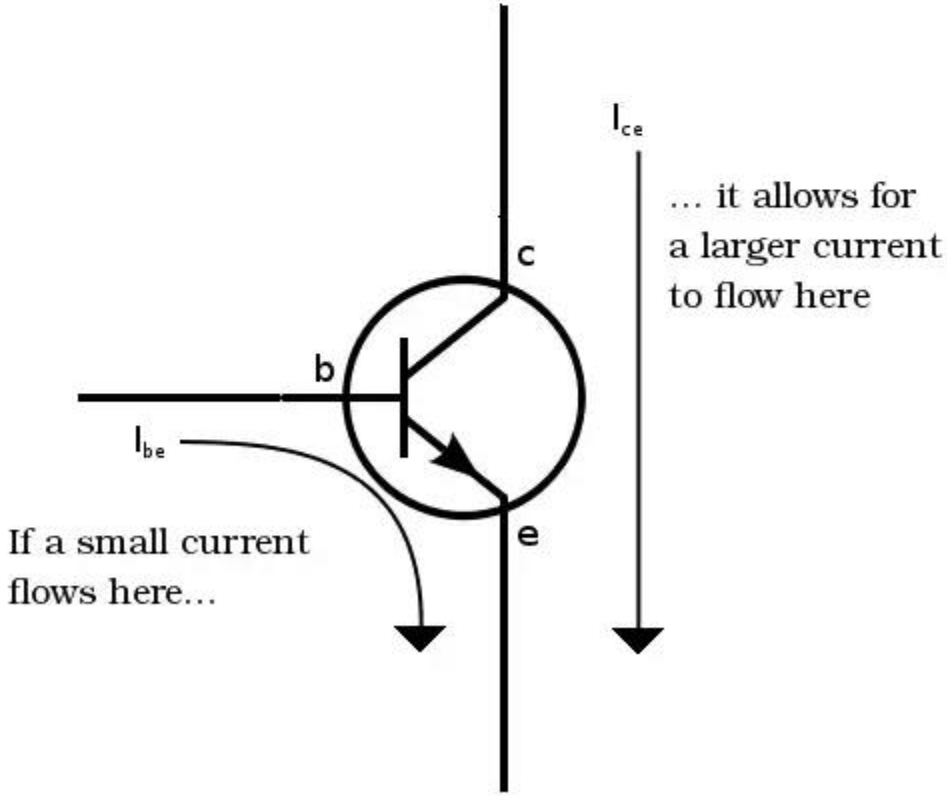


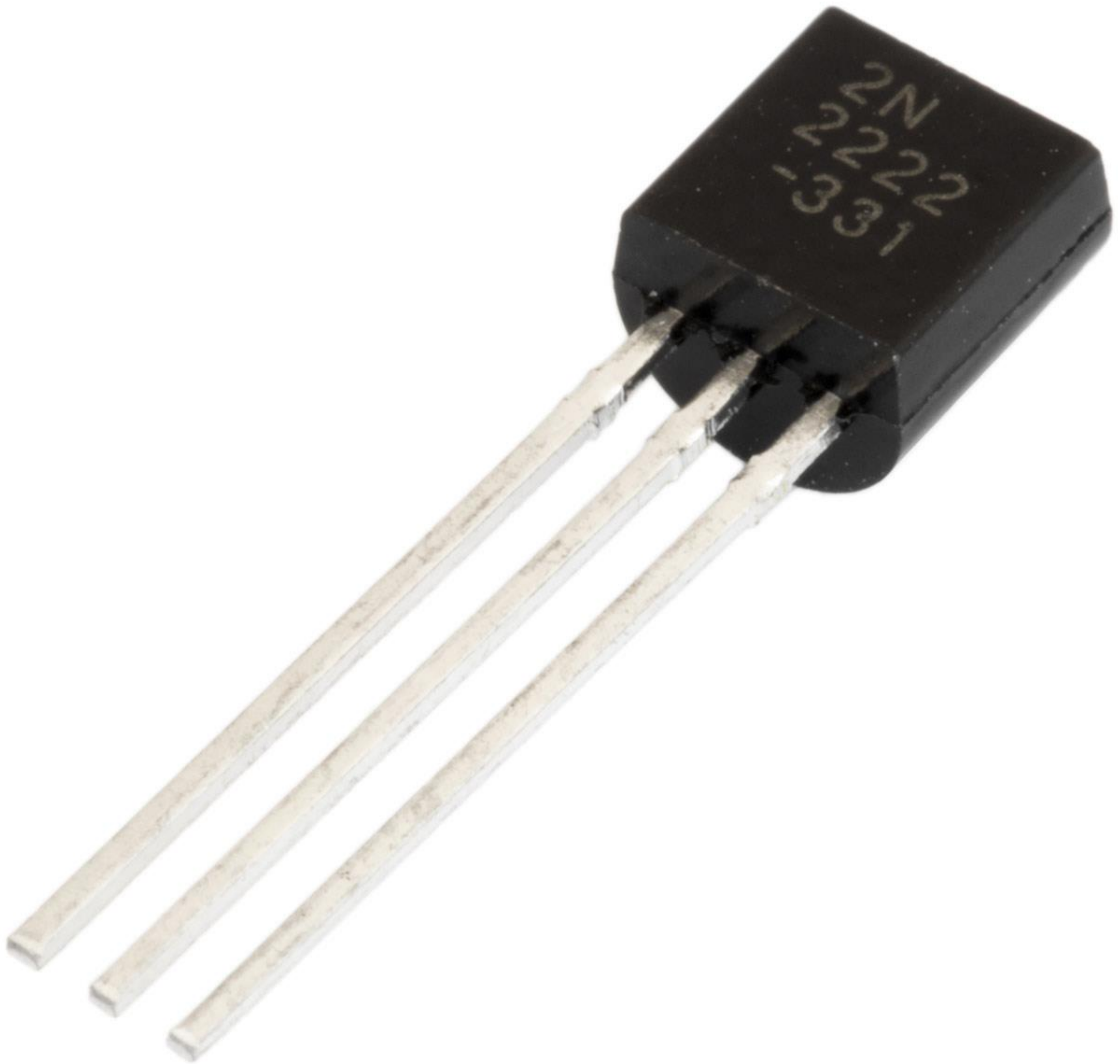
The Invention that Changed the World

Activating Switches

Using Transistors

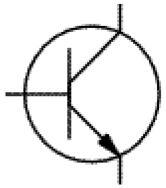
- Doping semiconductors





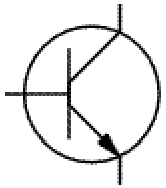
Summary: From a few to many^n

Transistor is the building block of CPU since 1960s



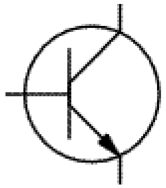
1970 - 1980

2K – 100K



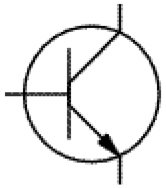
1980 - 1990

100K – 1M



1990 - 2000

1M – 100M



2000 - 2011

100M – 2.2B

Current World Population = 7Billion

about the number of transistors in 3 CPU chips!

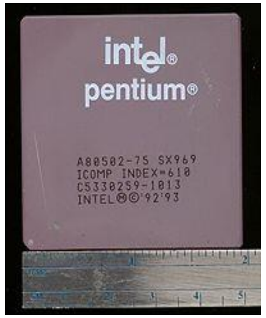
Summary: From BIG to small

Process size = Minimum length of a transistor

80286

1982

1.5 µm



Pentium

1993

0.80 µm
- **0.25 µm**

Pentium 4

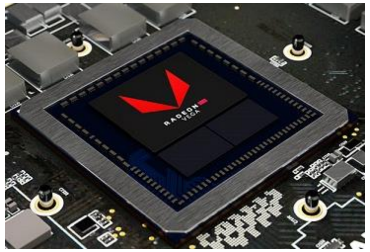
2000

0.180 µm
- **0.065 µm**

Core i7

2010

0.045 µm
- 0.032 µm



Radeon RX 400

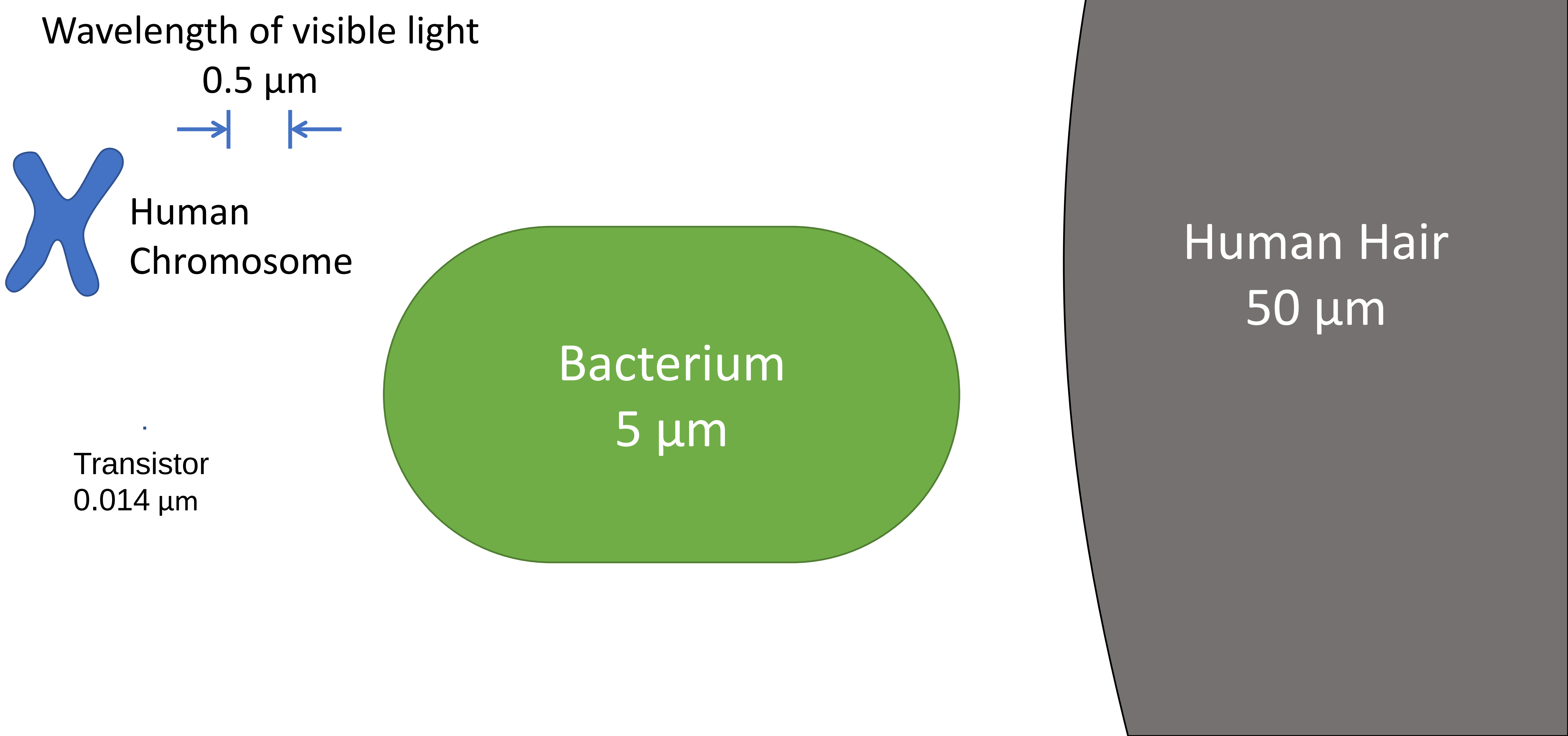
2016

0.014 µm

Wave length of visible light = 350nm (violet) to 780nm (red)

Process size now smaller than wavelength of violet light!

How small is 1 micrometer?



Summary: From S-L-O-W to fast

FLOPS = FLoating-point Operation Per Second

80286

1982

1.8 MIPS*

Pentium

1993

200 MFLOPS#

(5 ns per op)

Pentium 4

2000

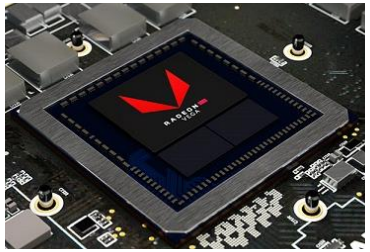
4 GFLOPS#

(0.25 ns per op)

Core i7

2011

120 GFLOPS #



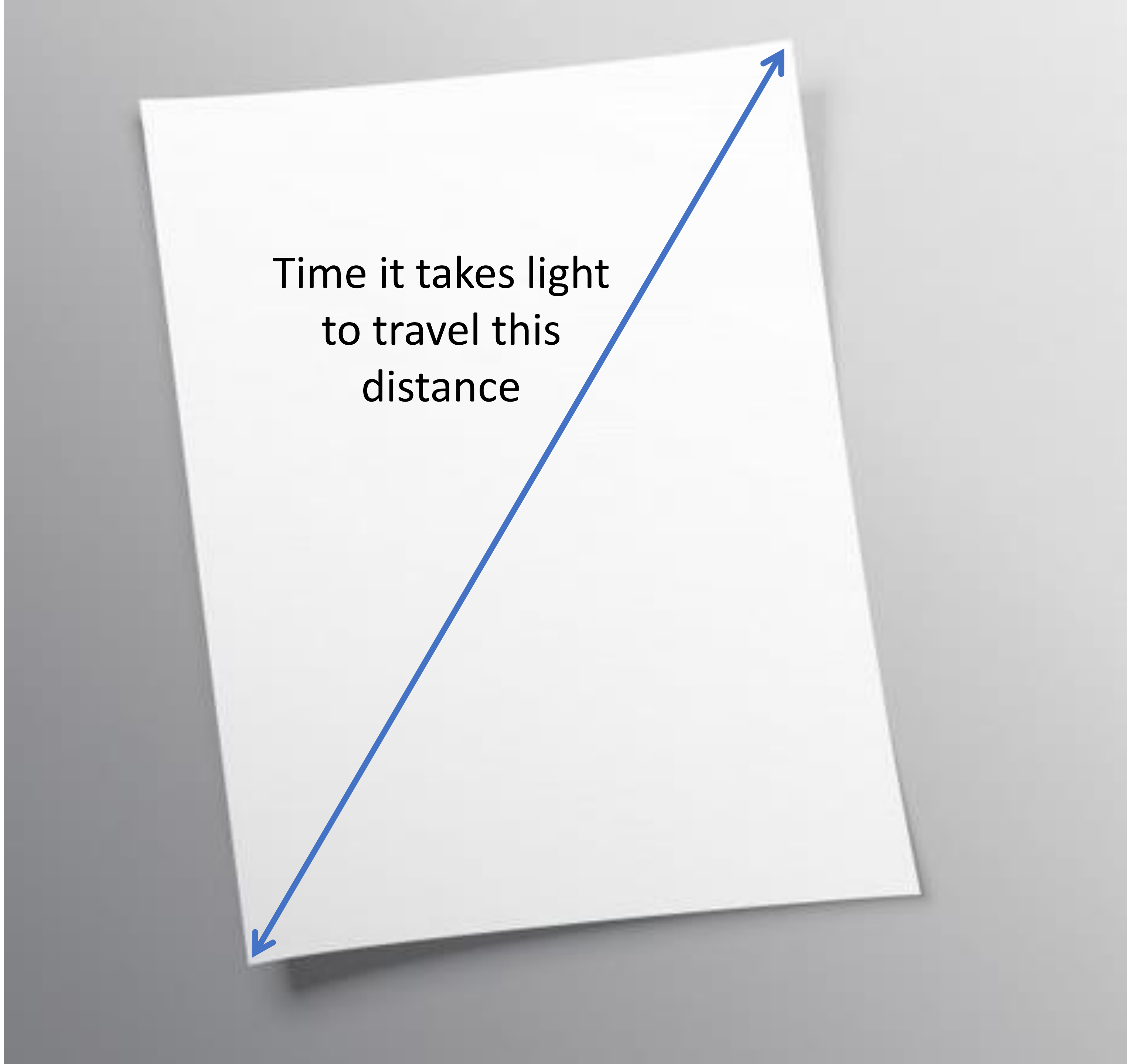
Radeon RX 400

2016

1-5 TFLOPS

How fast is 1 nanosecond?

1 GHz =
1 tick per ns



Summary: The Age of Computer

Unprecedented progress since late 1940s

Performance doubling ~2 years (1971-2005):

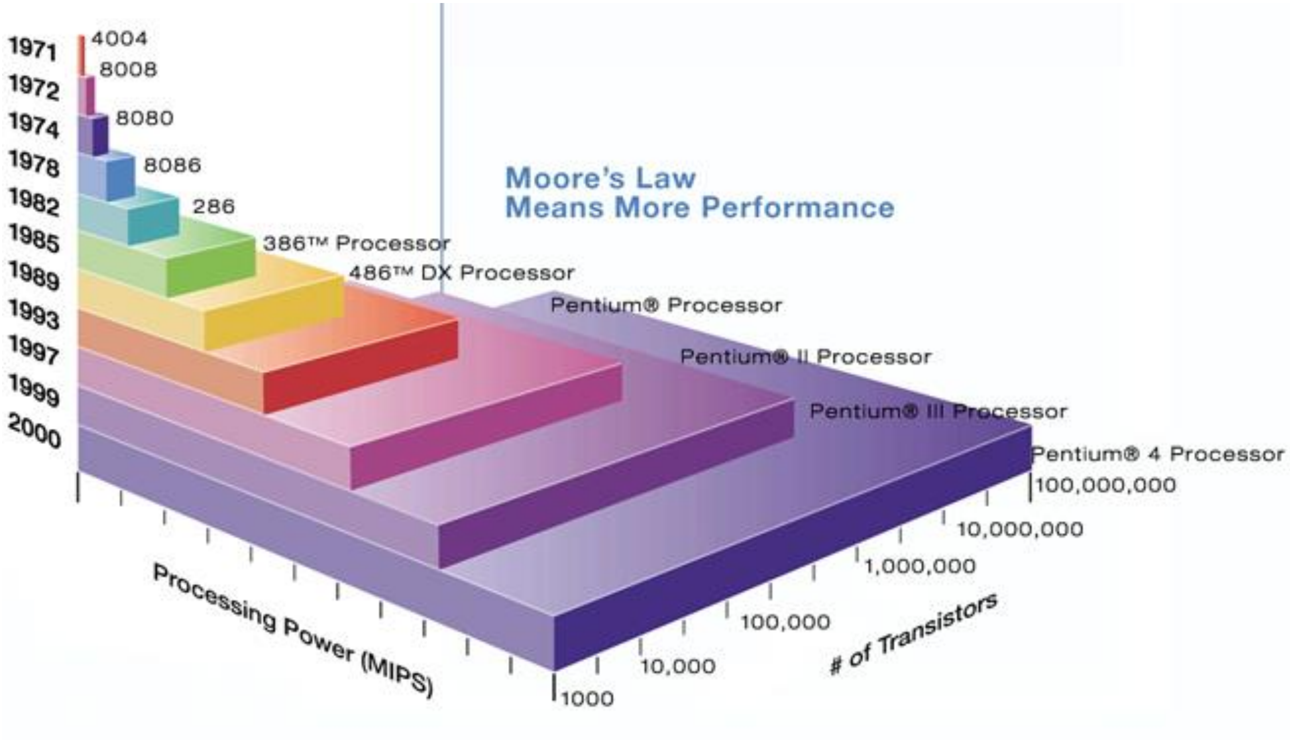
- Total of 36,000X improvement!

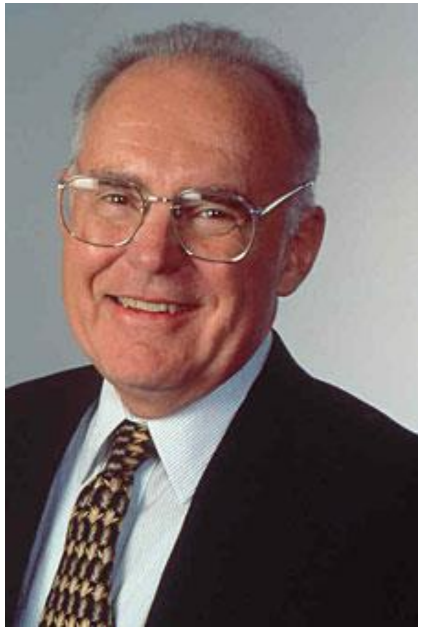
Incredible amount of innovations to revolutionize the computing industry again and again

Current Trends

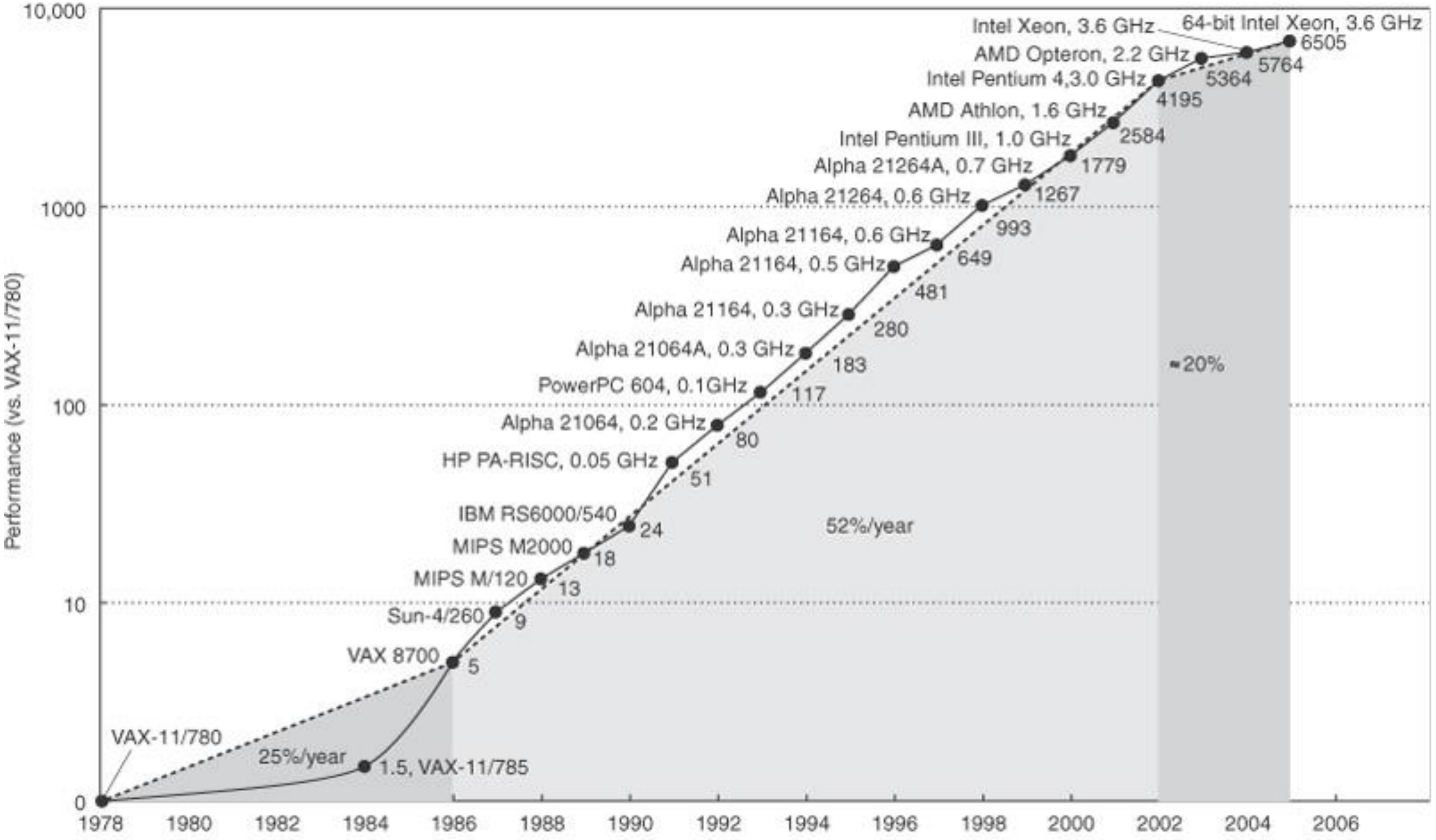
Moore's Law

Intel co-founder Gordon Moore "predicted" in 1965 that transistor density will double every 18 months





Processor Performance Increases



The Three Walls

Three major reasons for the unsustainable growth in uniprocessor performance

The Memory Wall:

– Increasing gap between CPU and Main memory speed

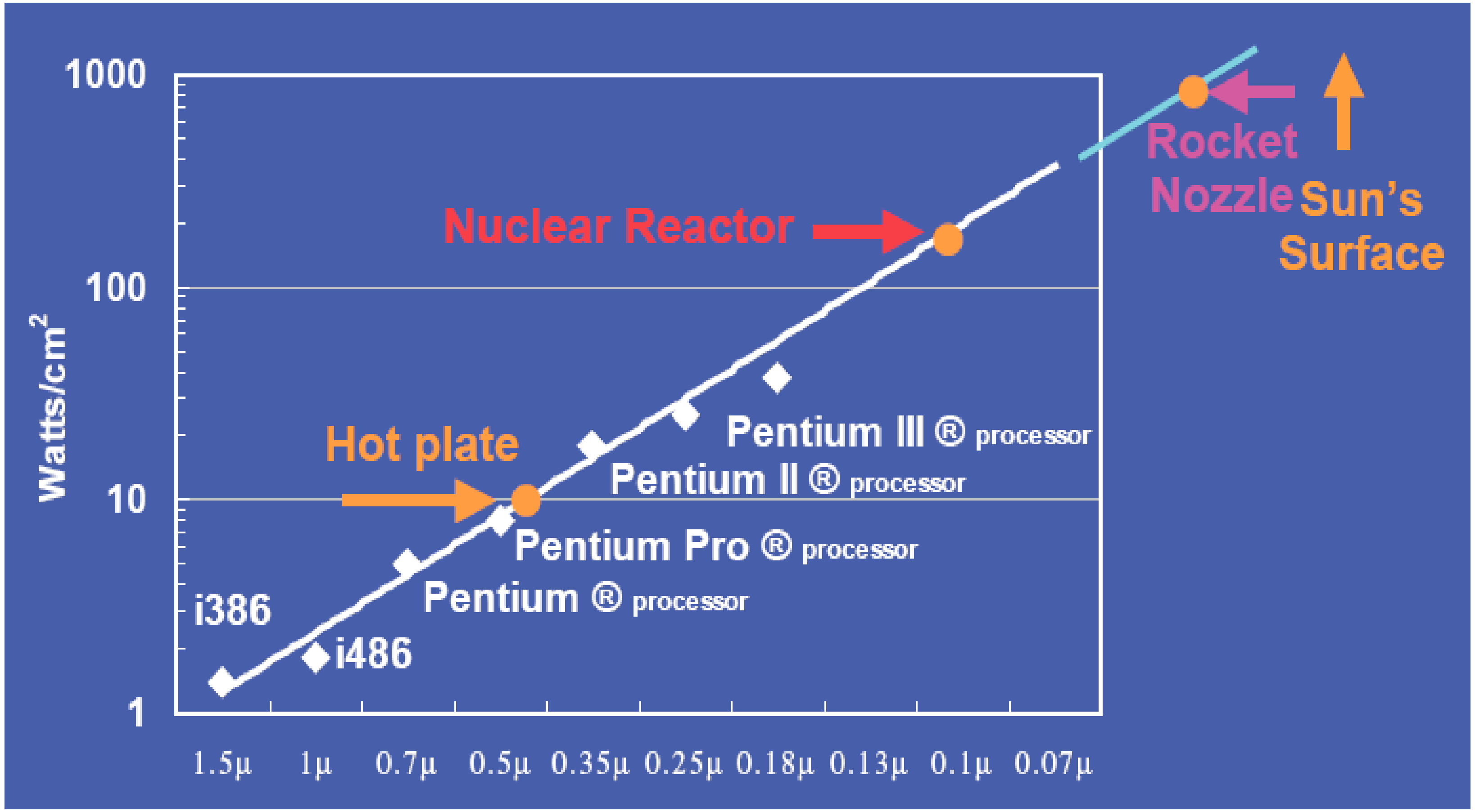
The ILP Wall:

– Decreasing amount of "work" (instruction level parallelism) for processor

The Power Wall:

– Increasing power consumption of processor

The Power Wall



We can now cramp more transistor into a chip than the ability (power) to turn them on!

Current State of Computer

Multicore is irreversible:

- All PC chip manufacturers have abandoned unicore development
- Expect to have more cores in a single chip
- Parallel programming is more important than ever

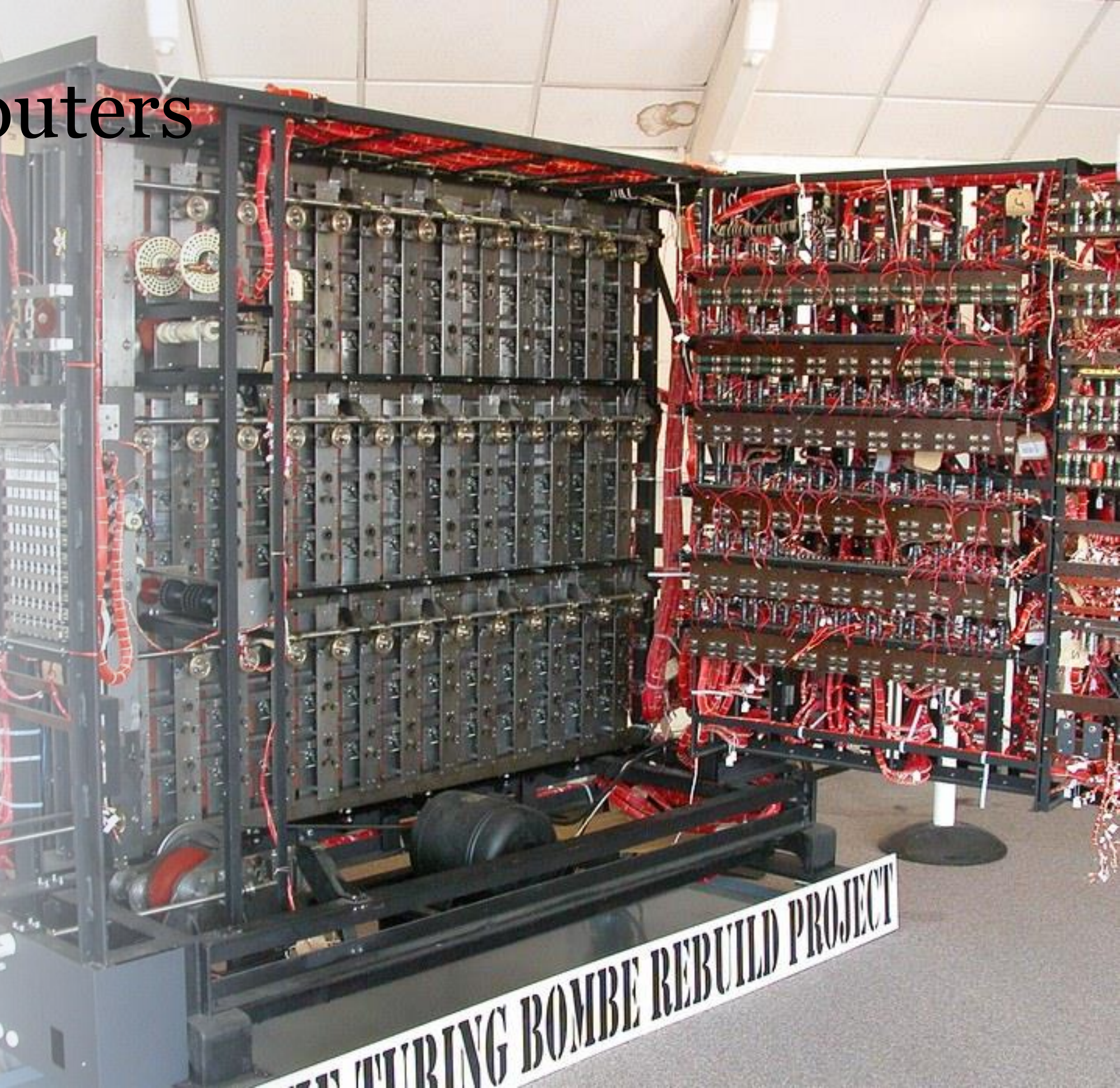
Great opportunity for computing professional

- New programming model is required
- Parallelising existing software
- Innovative ways to tap into the computing power

Computer Organization

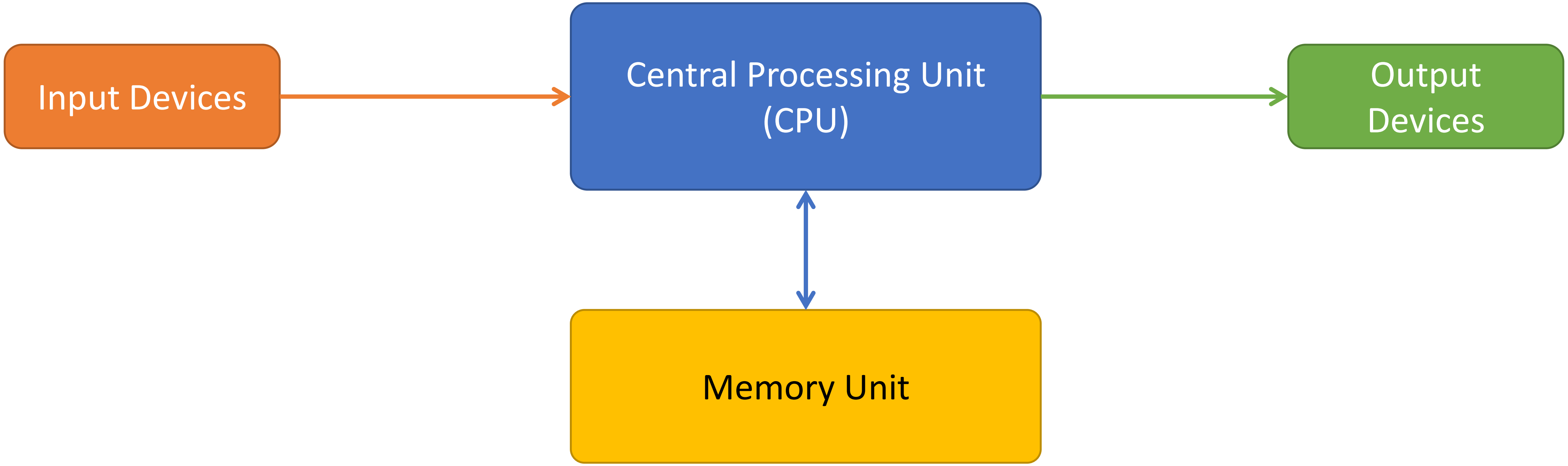
Two types of Computers

- Fixed Program Computers
  - Specific function by design
  - Cannot be programmed
- Stored Program Computers
  - Can be programmed
  - Applications can be stored and executed



Von Neumann Architecture

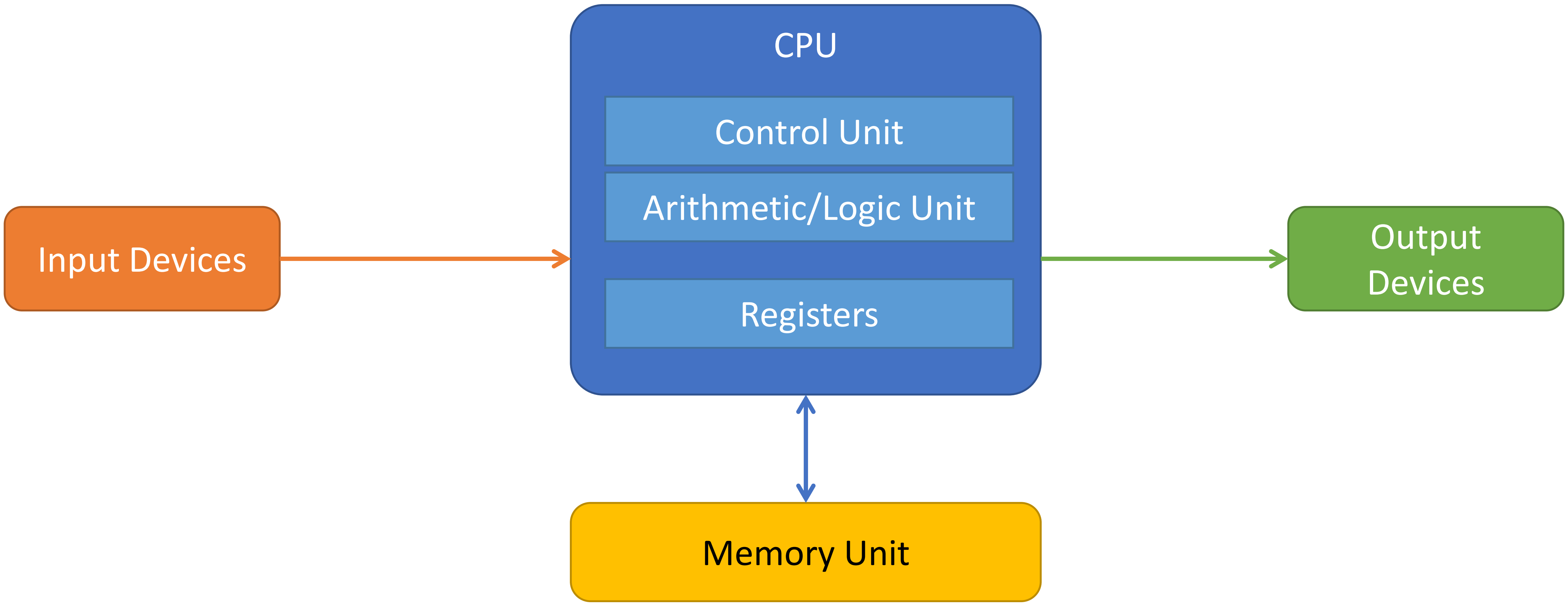
Proposed by John Von Neumann et al, 1945



Stored-Memory Concept:

– Data and program are stored in memory

Von Neumann Architecture



Components of a Computer

ALU
- Performs arithmetic and comparison logic

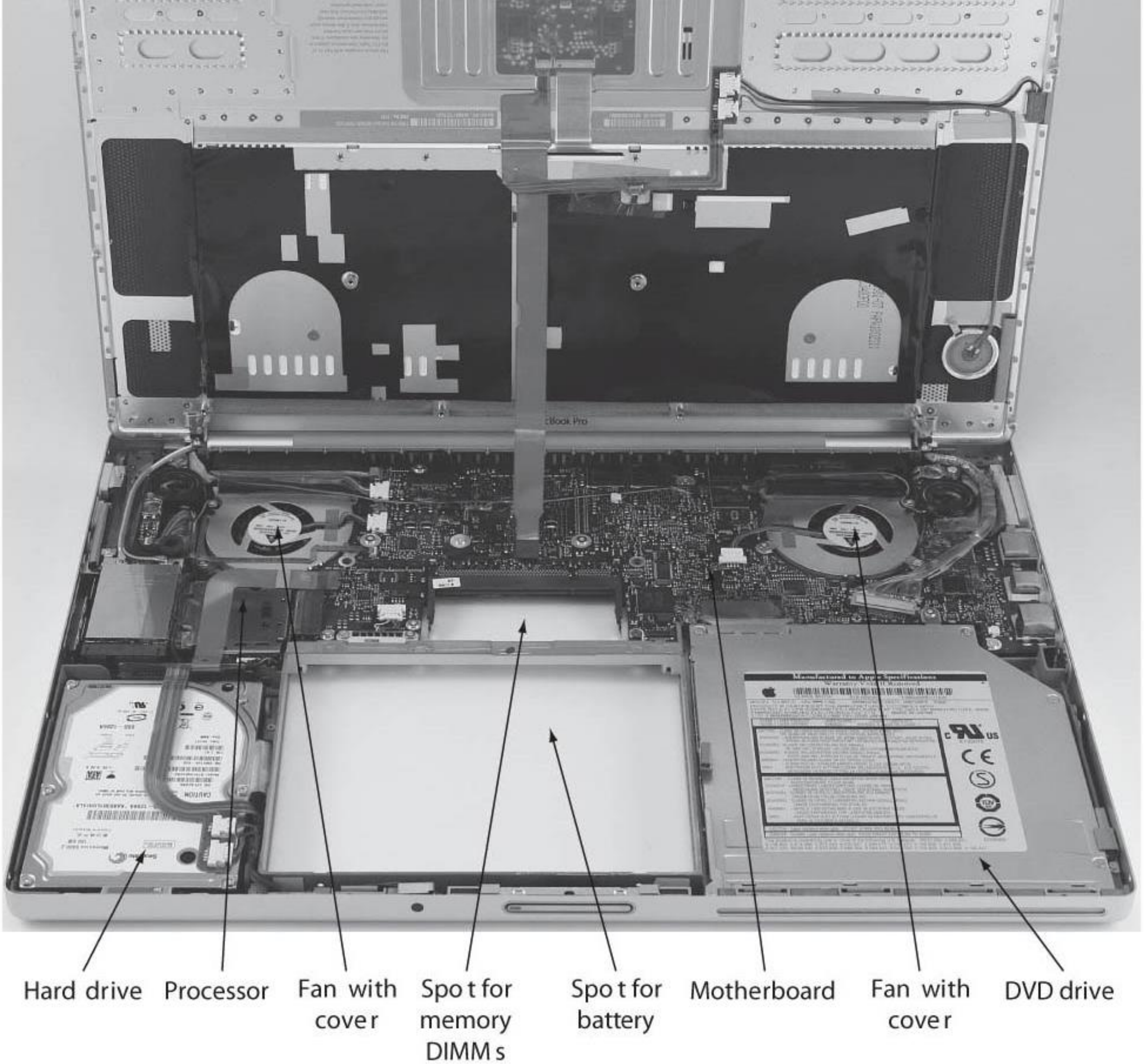
Registers
- Intermediate storage for ALU
- Very fast access speed

Memory
- Store programs and data

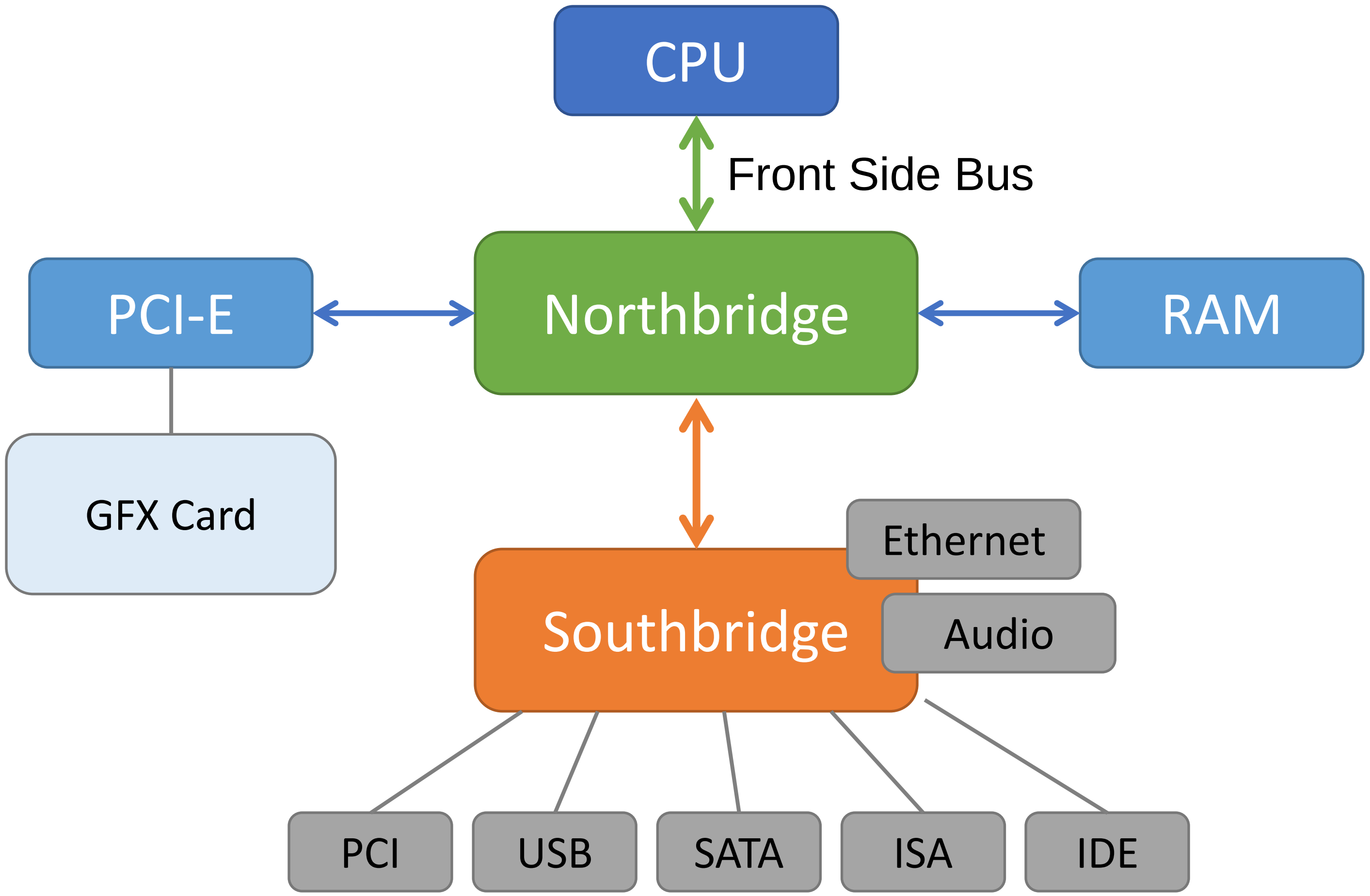
Input
- Feeds data, e.g. keyboard, mouse

Output
- Displays data, e.g. printer, monitor

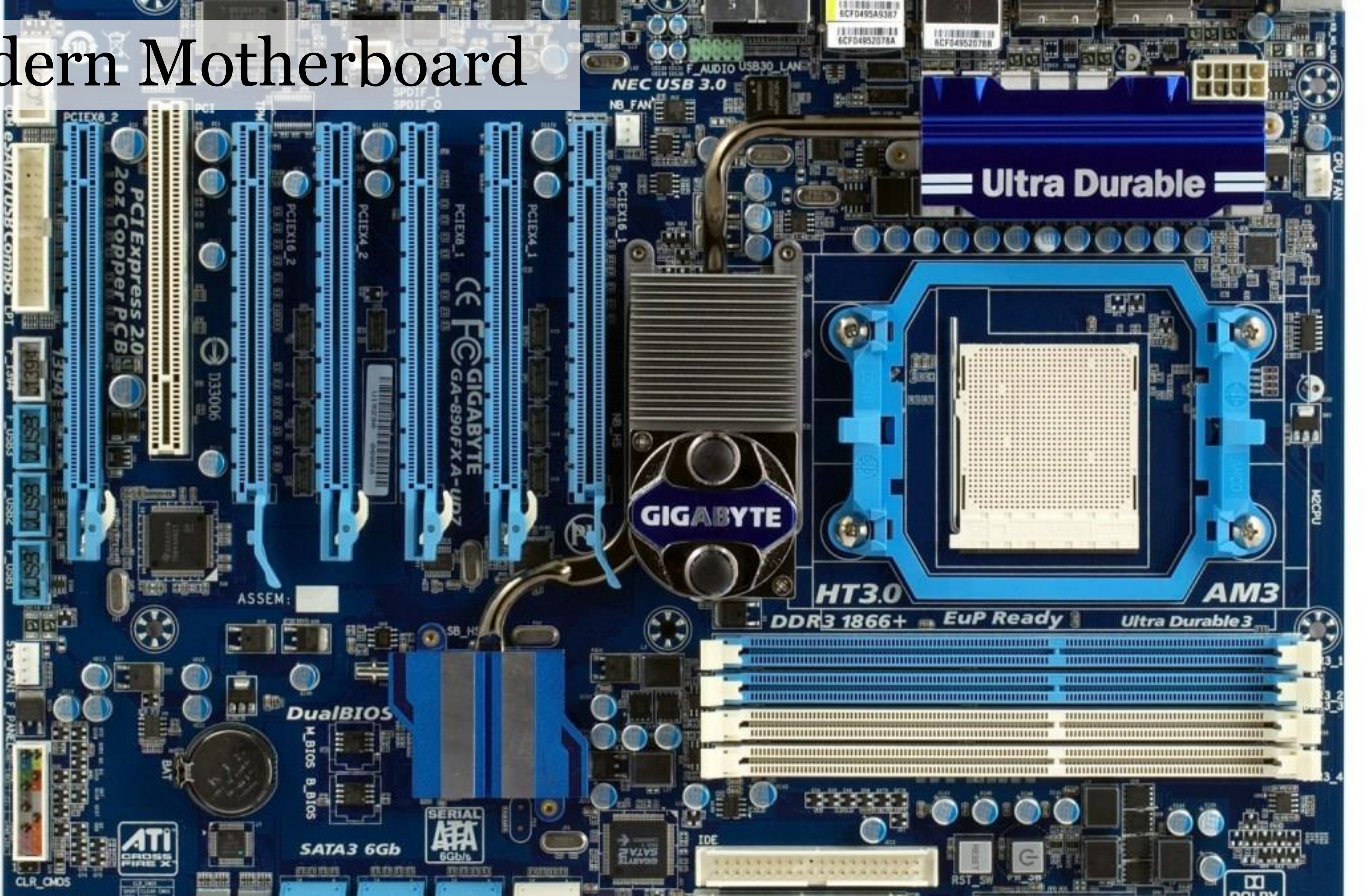
Example: Inside Your Laptop



Modern Computers



A Modern Motherboard



“Controlling” the Hardware

You write programs in high level programming languages,

– e.g., C/C++, Java:

A + B

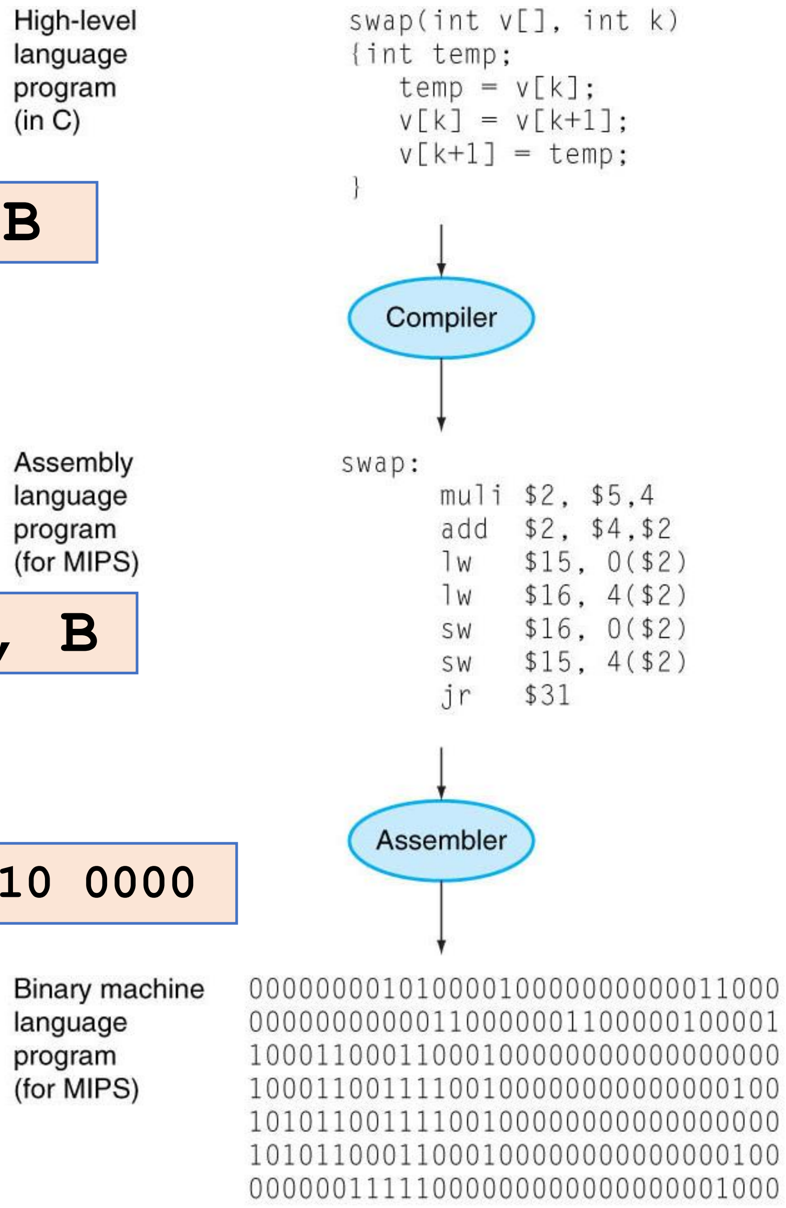
Compiler translates into assembly language

add A, B

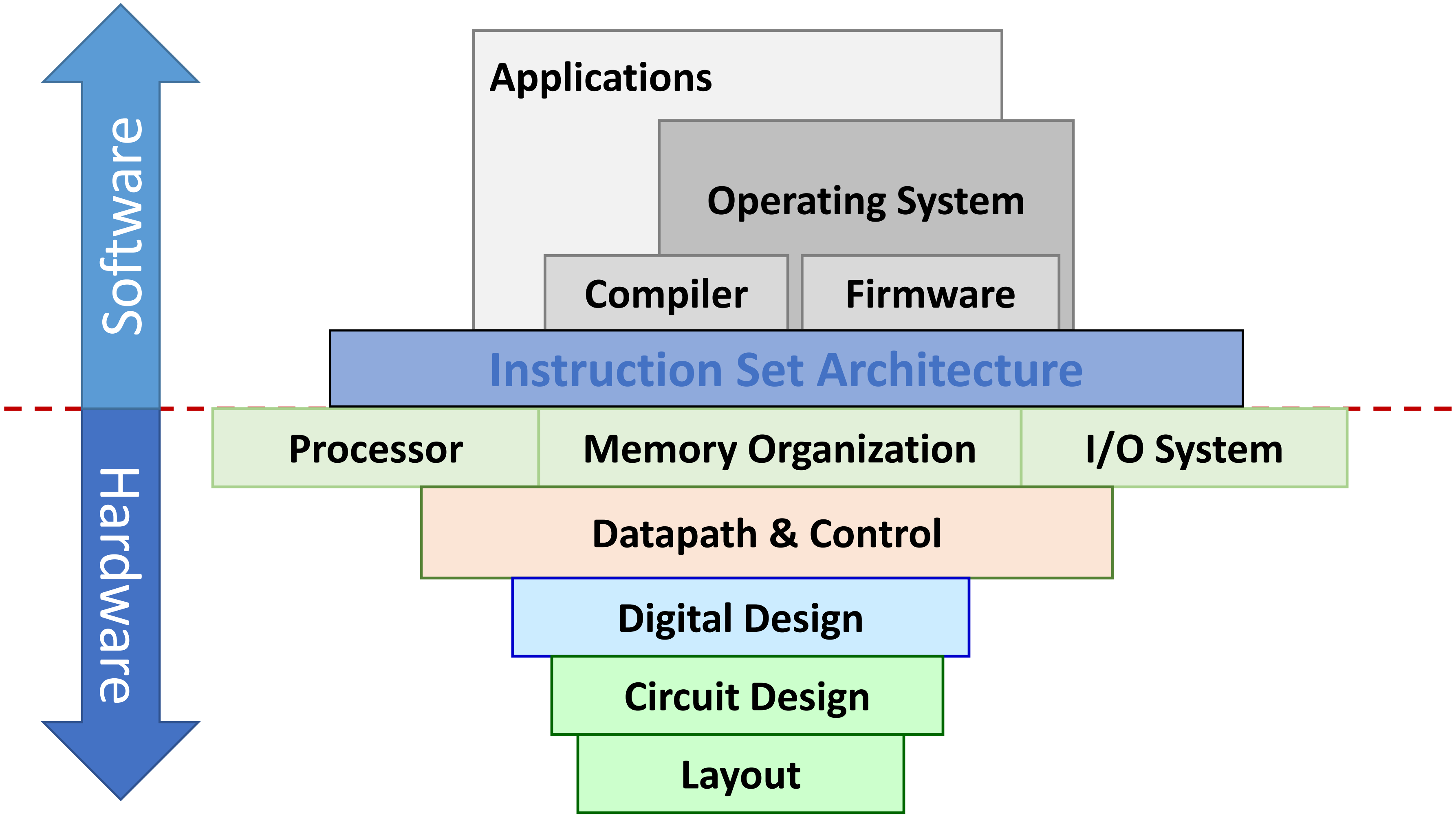
Assembler translates into machine language

– instructions that the processor can execute

1000 1100 1010 0000



ISA – Interface between HW & SW



Instruction Set Architecture (ISA)

Instruction Set Architecture (ISA)

- A subpart of computer architecture that is related to programming, as seen by the programmer and compiler

ISA exposes the capabilities of the underlying processor as a set of well-defined instructions

- Serves as the interface between hardware and software
- Serves as an abstraction which allow freedom in hardware implementations

Example: Instruction Set Architecture

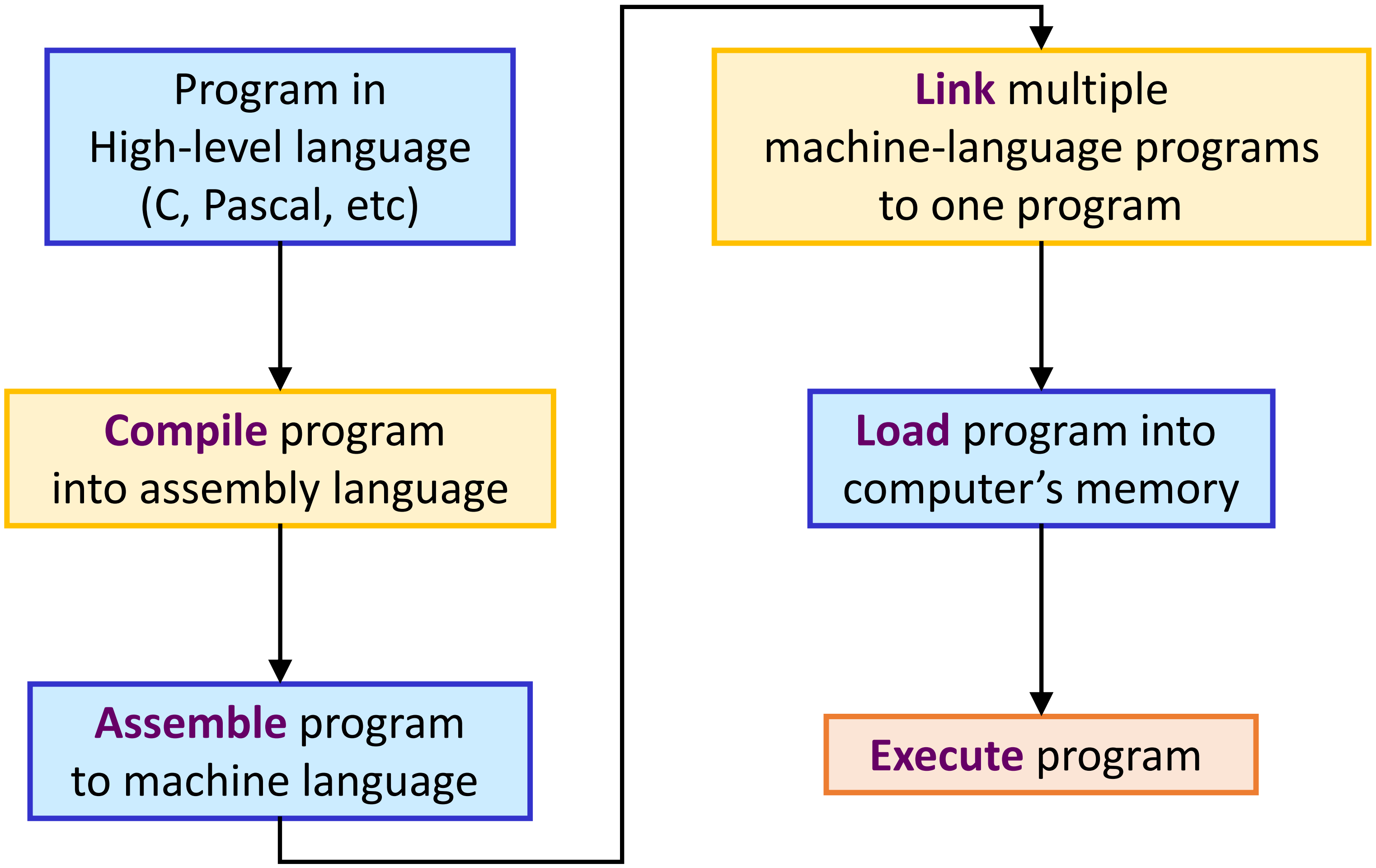
x86-32 (IA32)	MIPS	ARM
Intel 80486, Pentium (2, 3, 4), Core i3, i5, i7	R2000, R3000, …, R10000	Generations of chips: ARMv1, v2, …., v7
AMD K5, K6, Athlon, Duron, Sempron		StrongARM, ARM Cortex
Dominates the PC market	Widely used in classroom for teaching	Most popular embedded system chip

Each ISA has a family of chips

➔ Multiple hardware implementations

Code execution

The *Life* of a program



Code Execution

Instruction Execution Cycle in the Processor:

Fetch:

- **Fetch** next instruction from memory into processor

Decode:

- **Decode** the instruction

Execute:

- Get operands
- **Execute** instruction
- Store the execution result

Walkthrough: The code example

1. Discover the typical computer components
2. Learn the different types of instruction

Heavily Simplified! ☺

```
//assume "res is 0" initially
for ( i = 1; i < 10; i++ ) {
    res = res + i;
}
```

C-like code fragment

compiles to

```
i   ← 1
res ← res + i
i   ← i + 1
if i < 10, jump -2
```

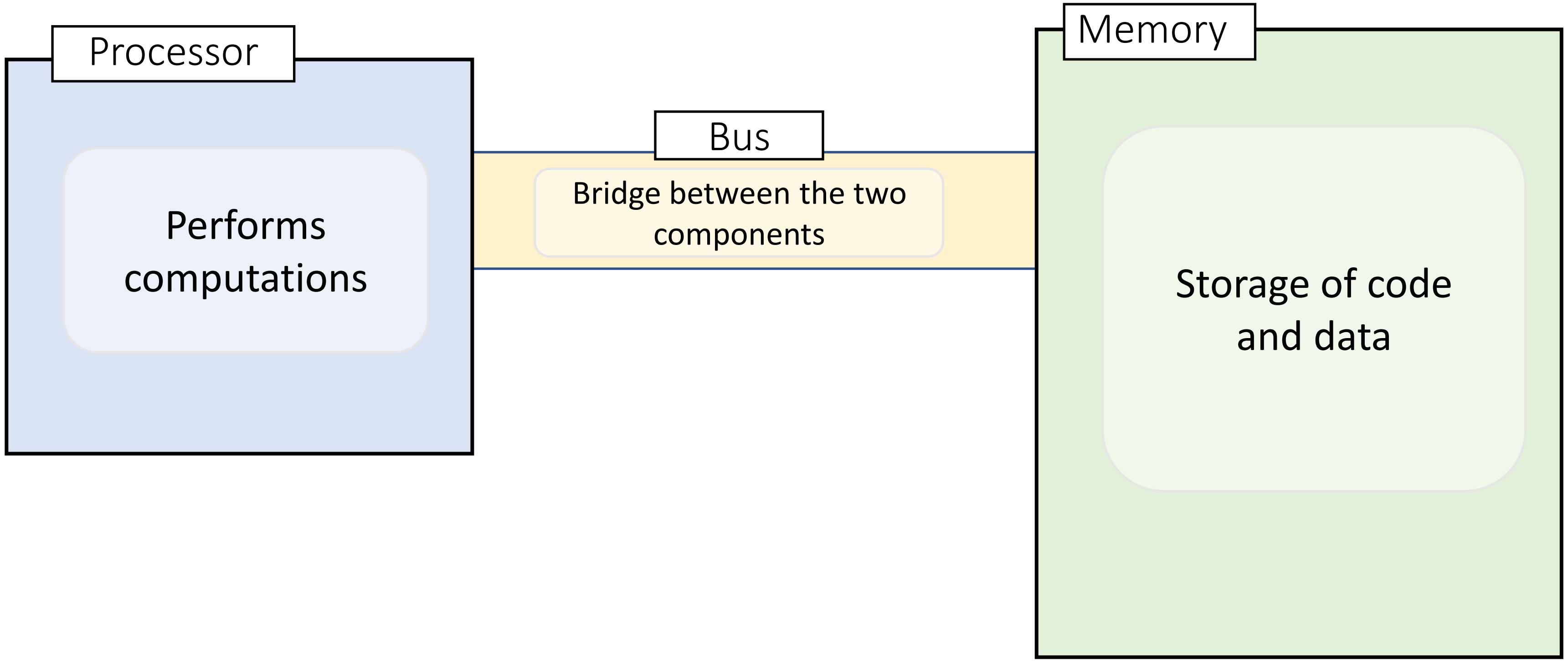
"Assembly" Code

Go back 2 instructions

Walkthrough: The Components

The two major components in a computer

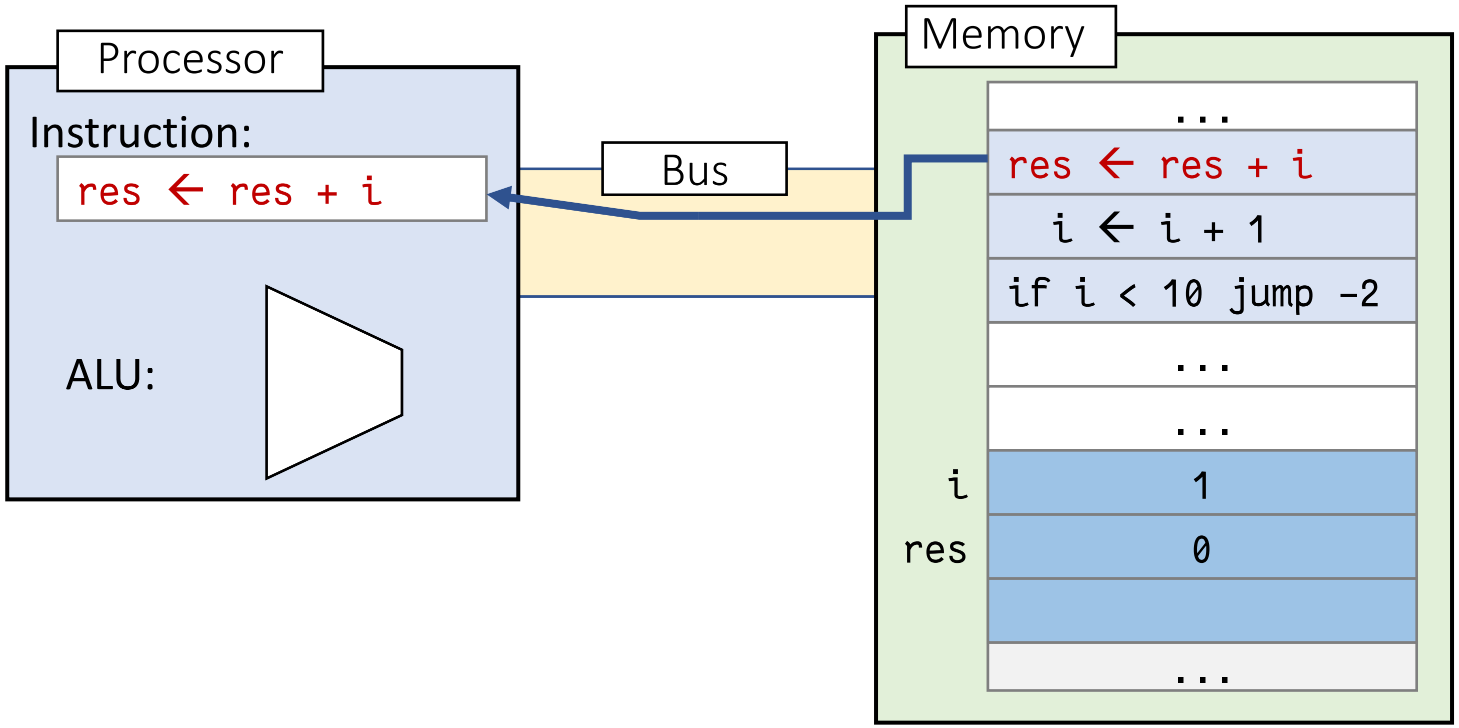
- Processor and Memory
- Input/Output devices omitted in this example



Executing an Instruction

1. Fetch

Obtain instruction from memory



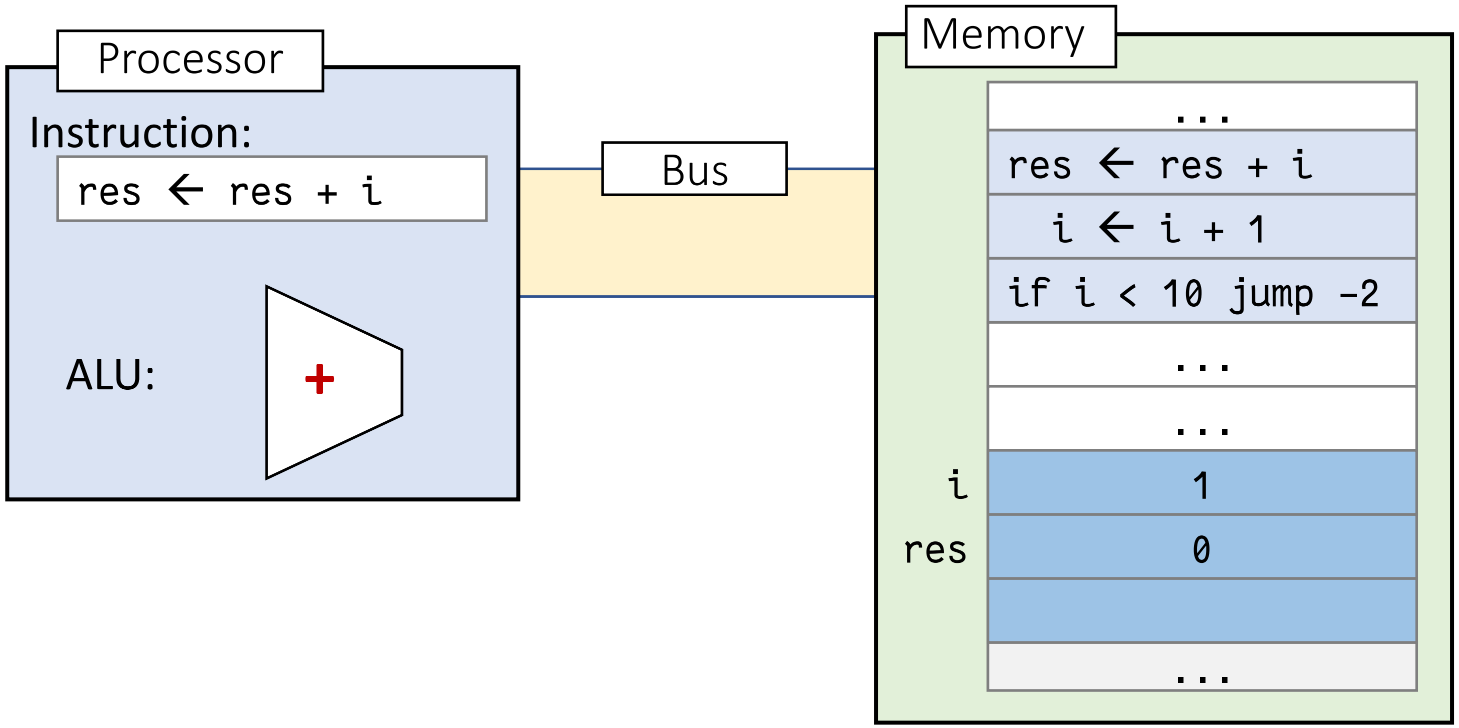
Executing an Instruction

1. Fetch

   Obtain instruction from memory

2. Decode

   Interpret the instruction



Executing an Instruction

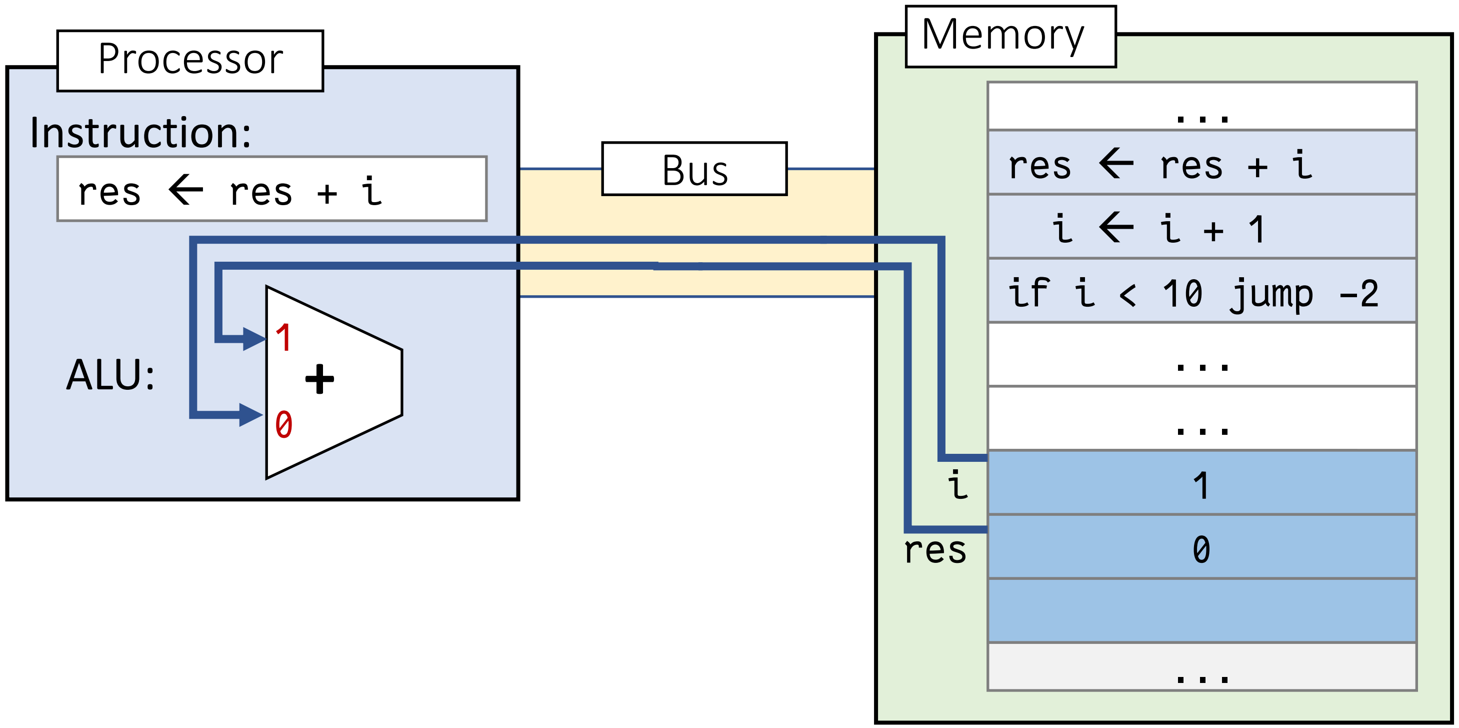
1. Fetch

   Obtain instruction from memory

2. Decode

   Interpret the instruction

3. Get operands



Executing an Instruction

1. Fetch

   Obtain instruction from memory

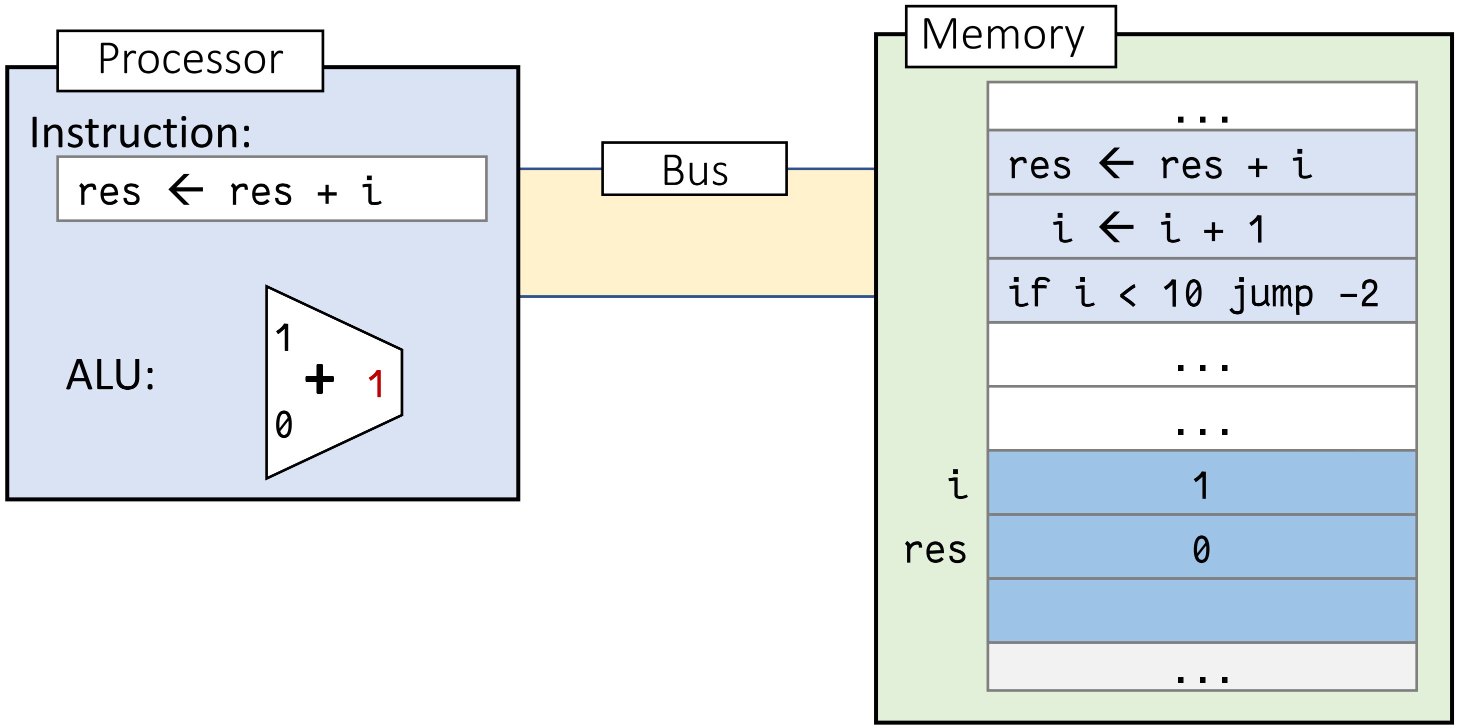
2. Decode

   Interpret the instruction

3. Get operands

4. Execute

   Perform the operation



Executing an Instruction

1. Fetch

   Obtain instruction from memory

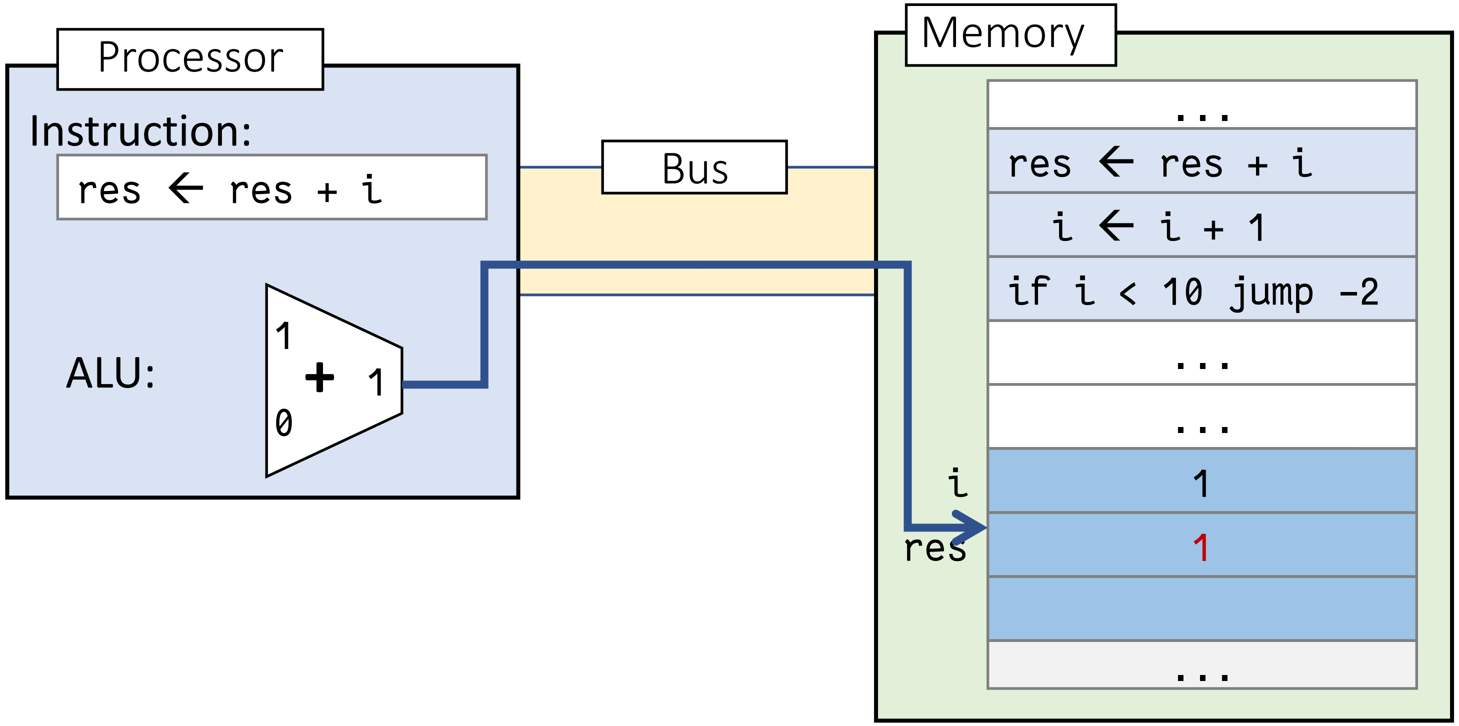
2. Decode

   Interpret the instruction

3. Get operands
4. Execute

   Perform the operation

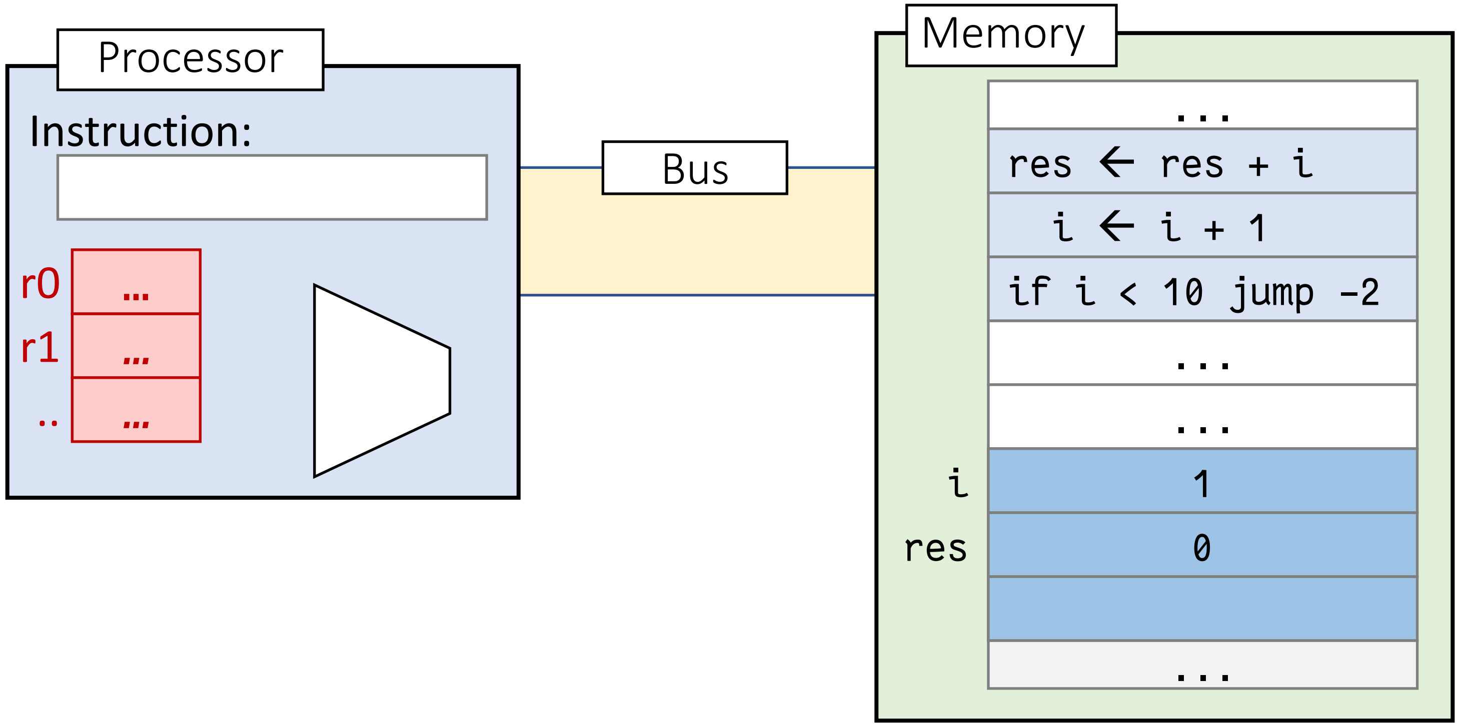
5. Store result



Problem: Memory access is slow!

To avoid frequent access of memory

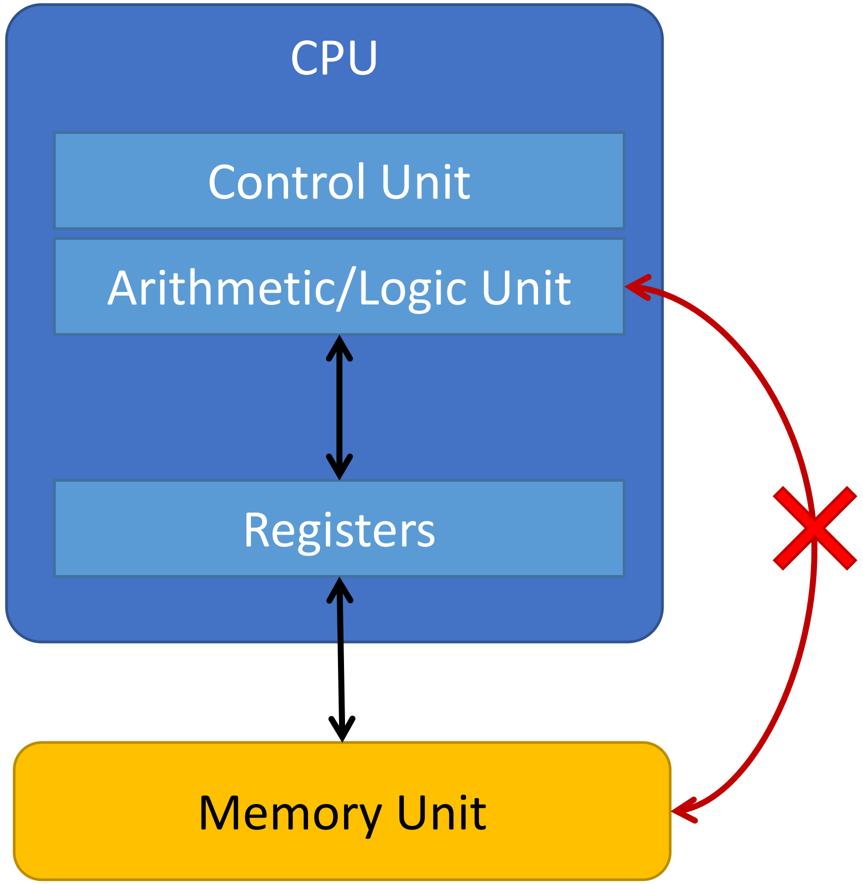
– Provide temporary storage in the processor (called registers)



Load-Store Architecture

Instructions belong to one of two categories

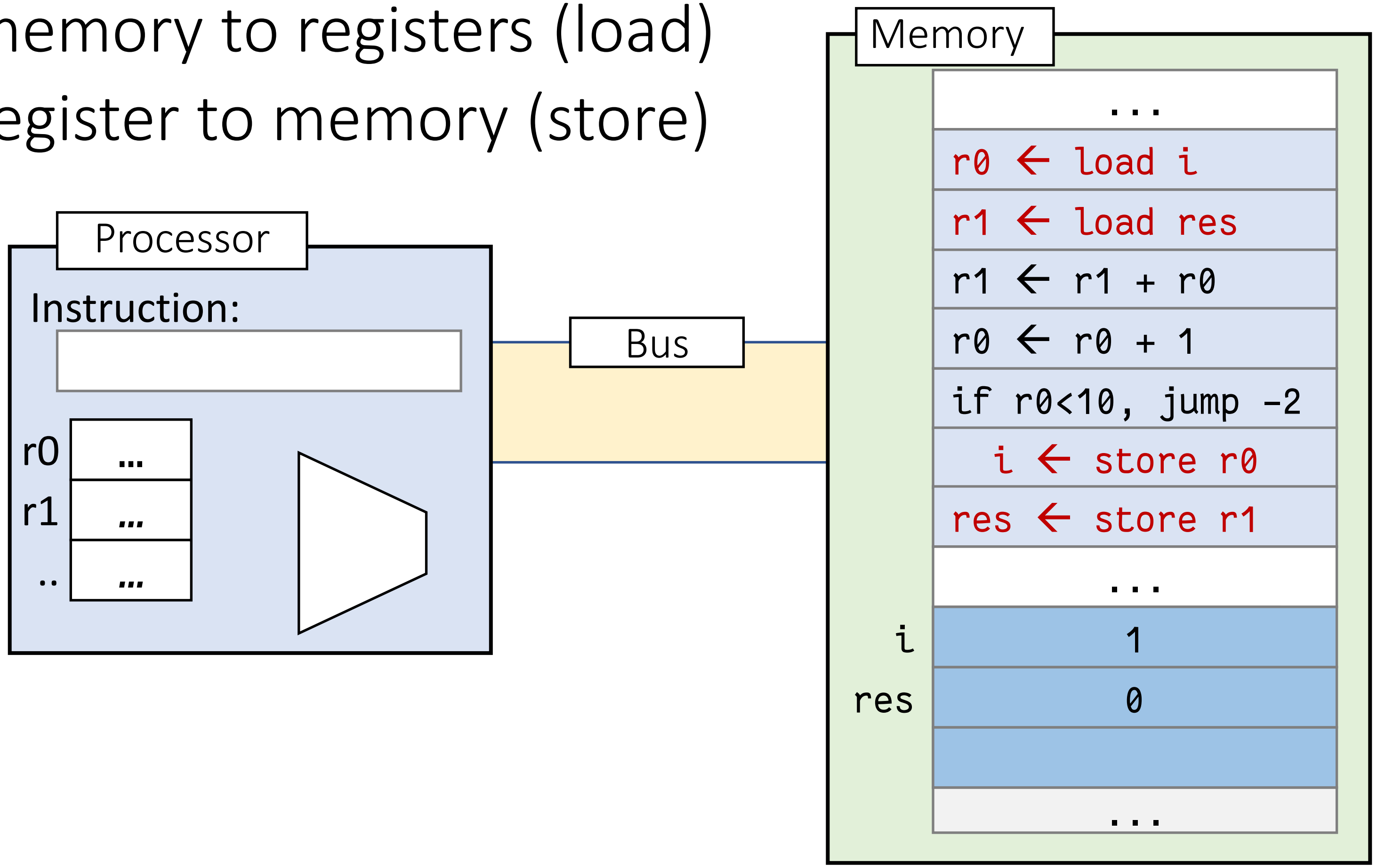
1. memory access
   – Load and store between memory and registers
2. ALU operations
   – Operations between registers



Memory Instructions

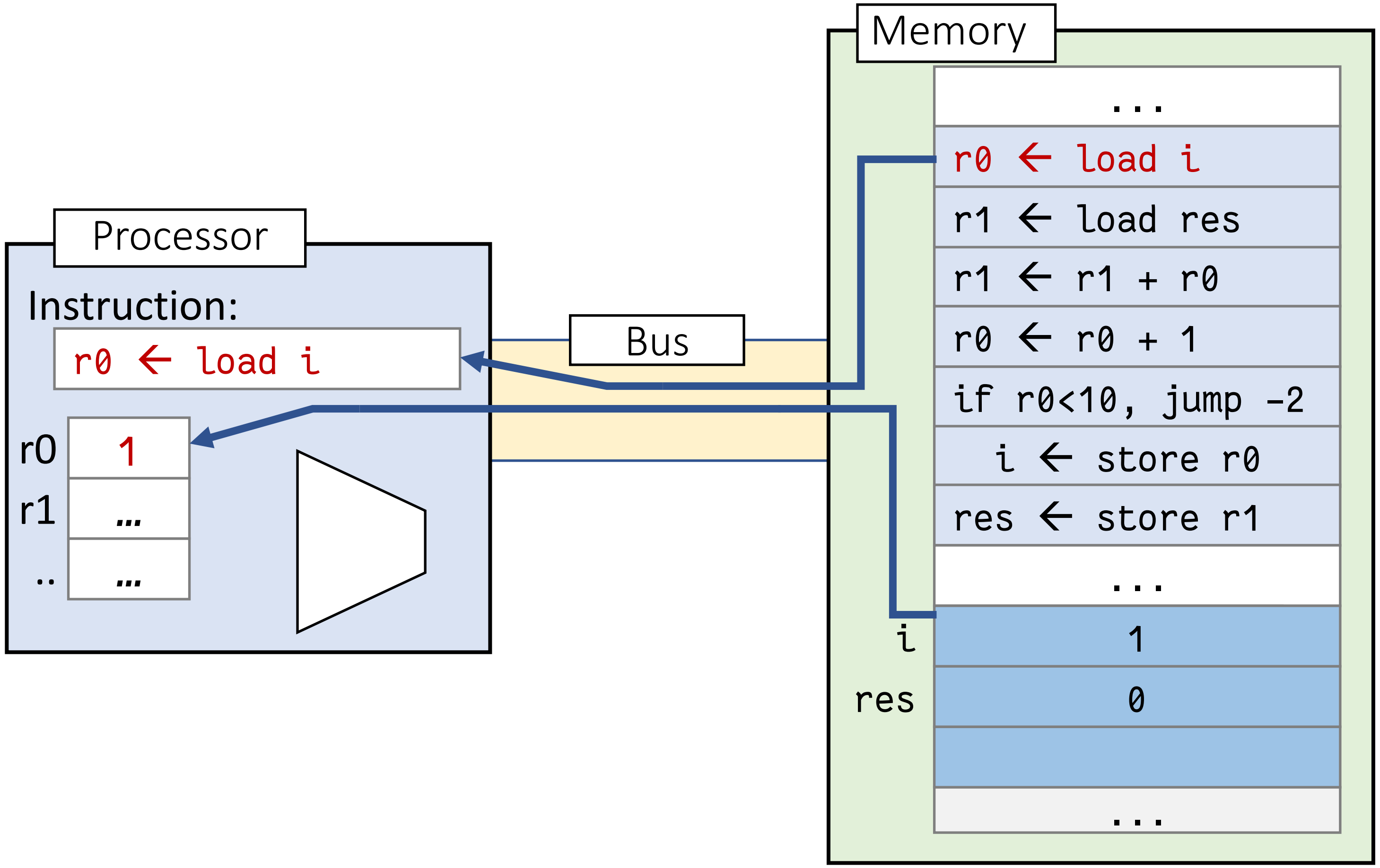
Need instruction to move data

- from memory to registers (load)
- from register to memory (store)



Load Operation

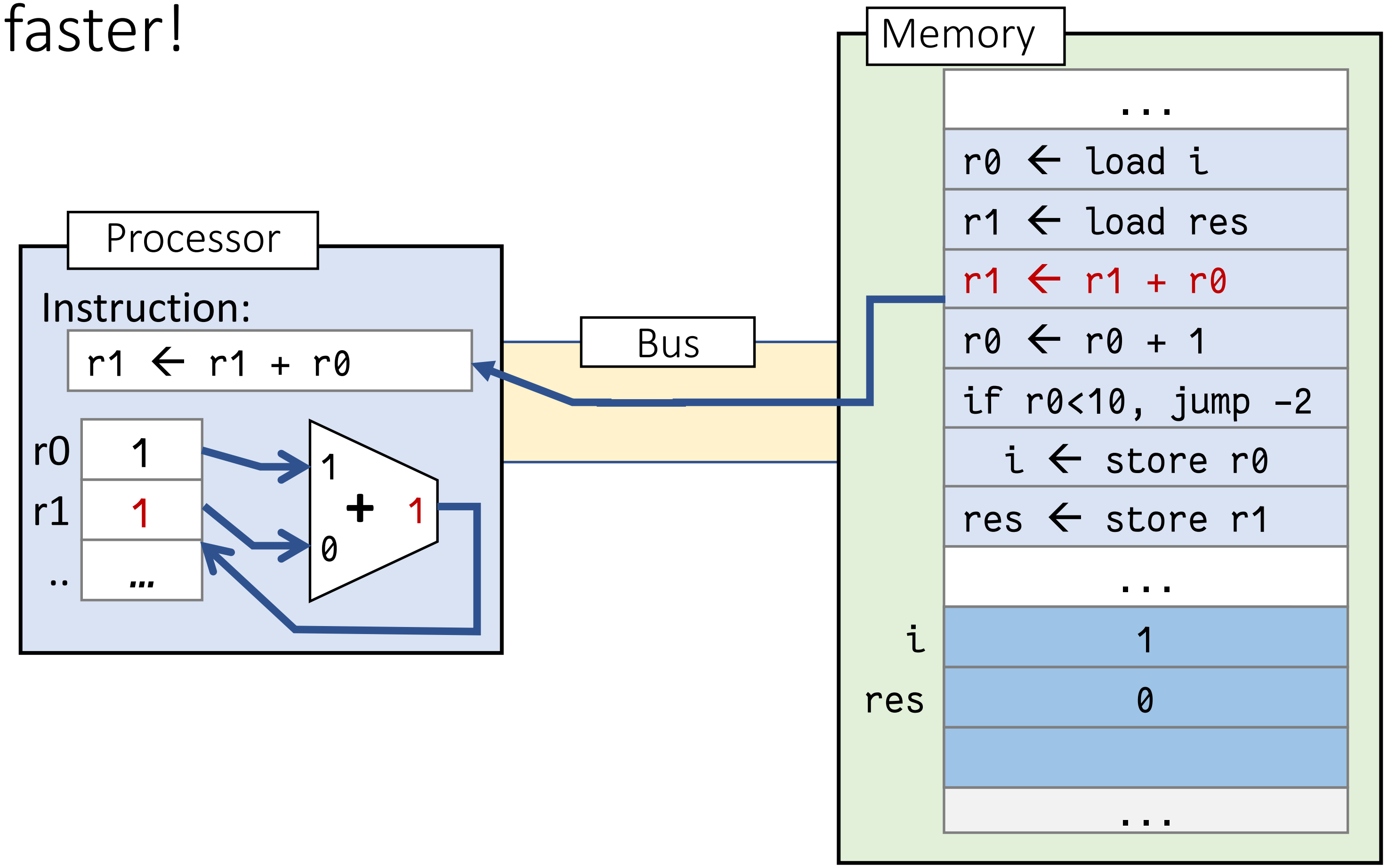
1. Fetch
2. Decode
3. Execute



Register-to-Register Instructions

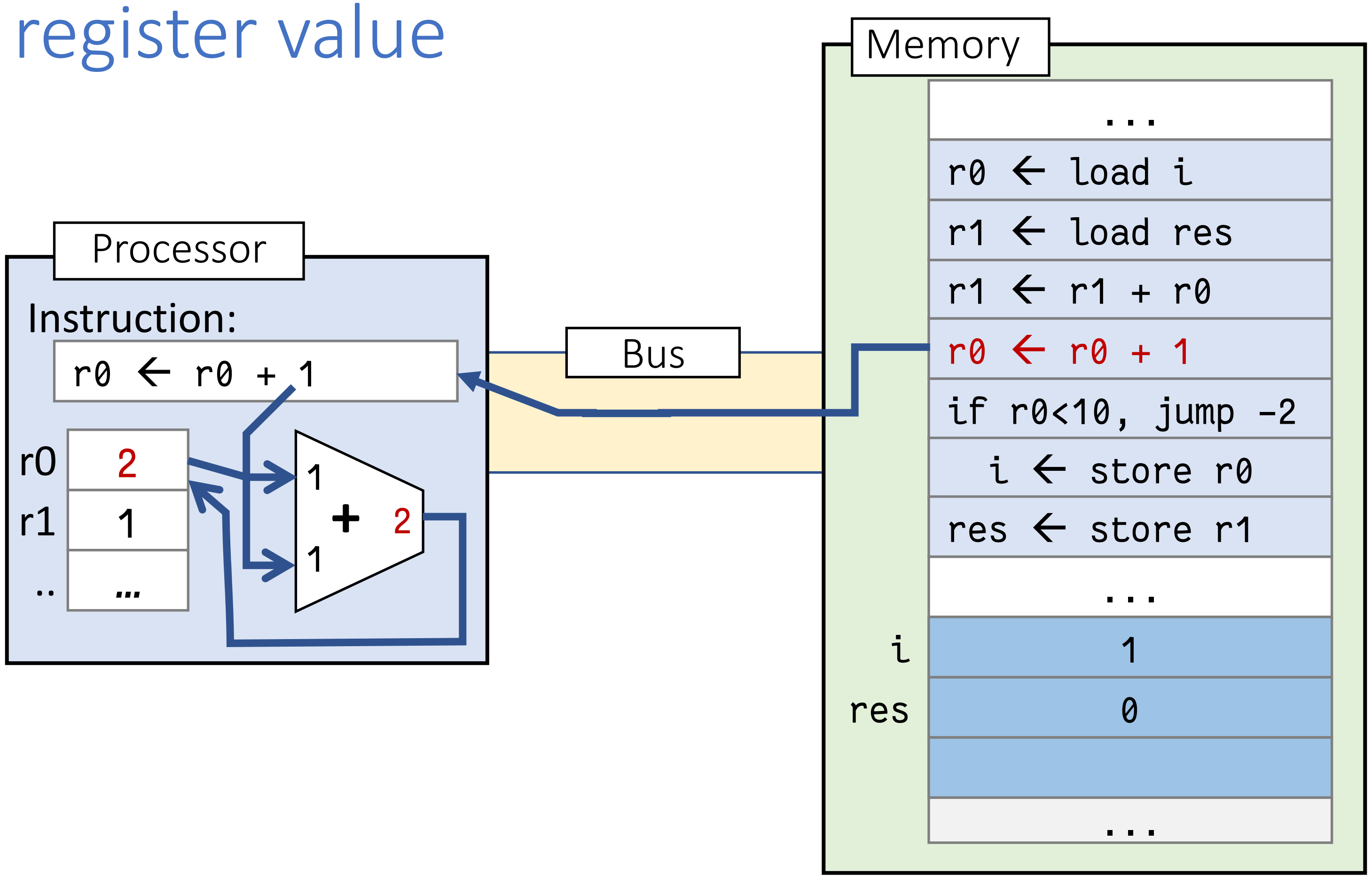
Arithmetic/Logic operations now work on registers only

- Much faster!



Register-to-Register Instructions

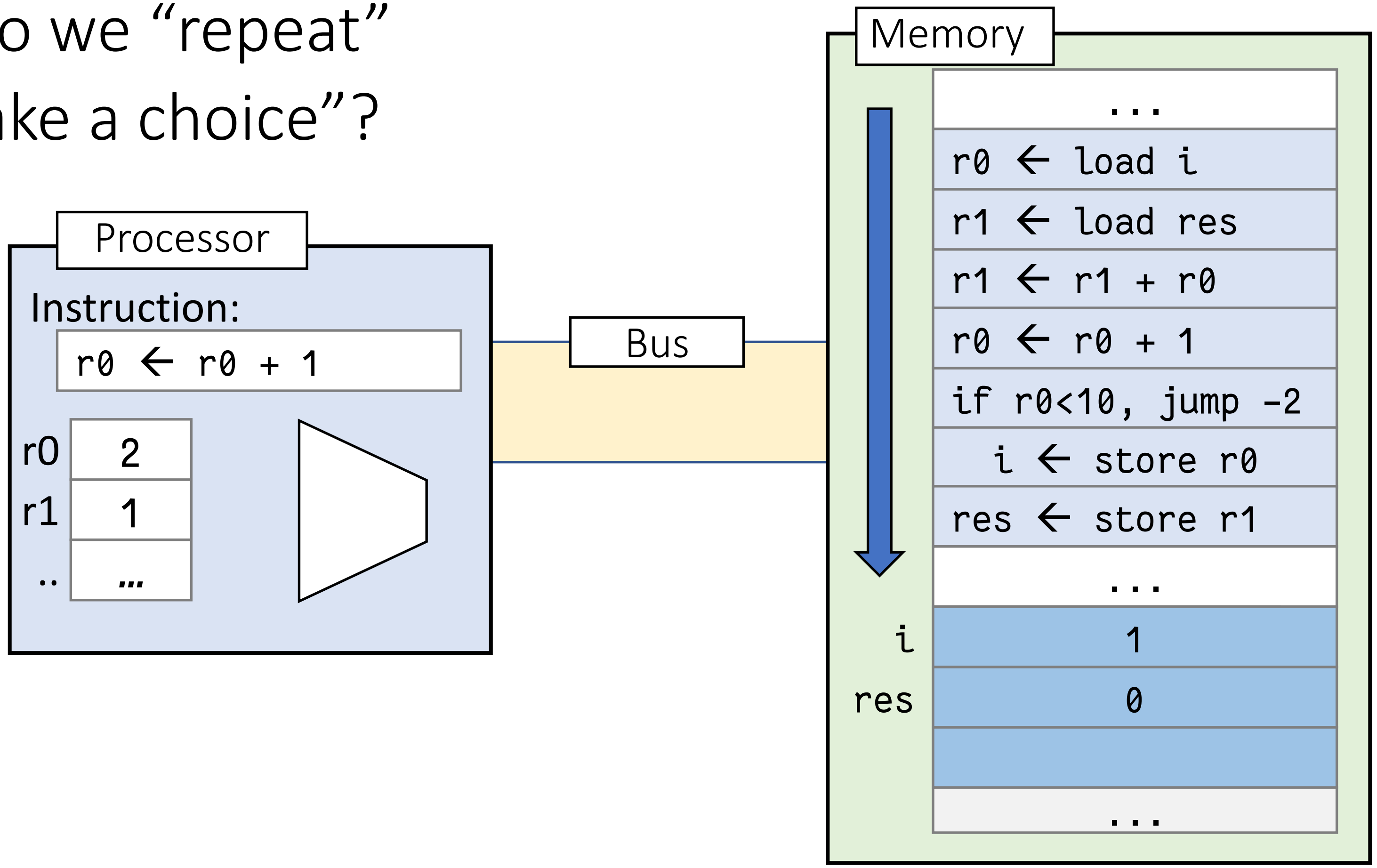
Sometimes, arithmetic operation uses a constant value instead of register value



Execution Sequence

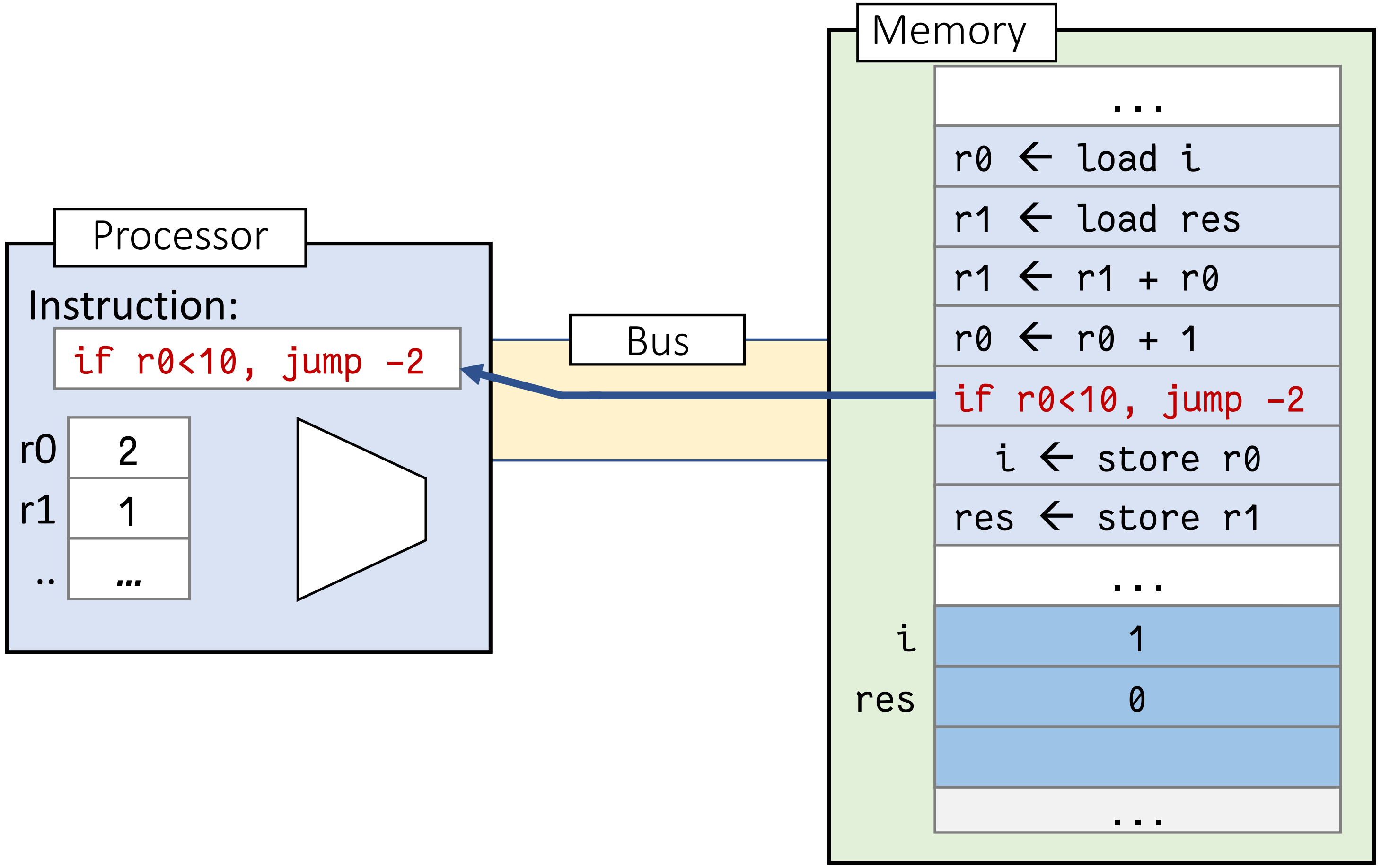
Instruction is executed sequentially by default

- How do we “repeat”
- or “make a choice”?



Walkthrough: Control flow instruction

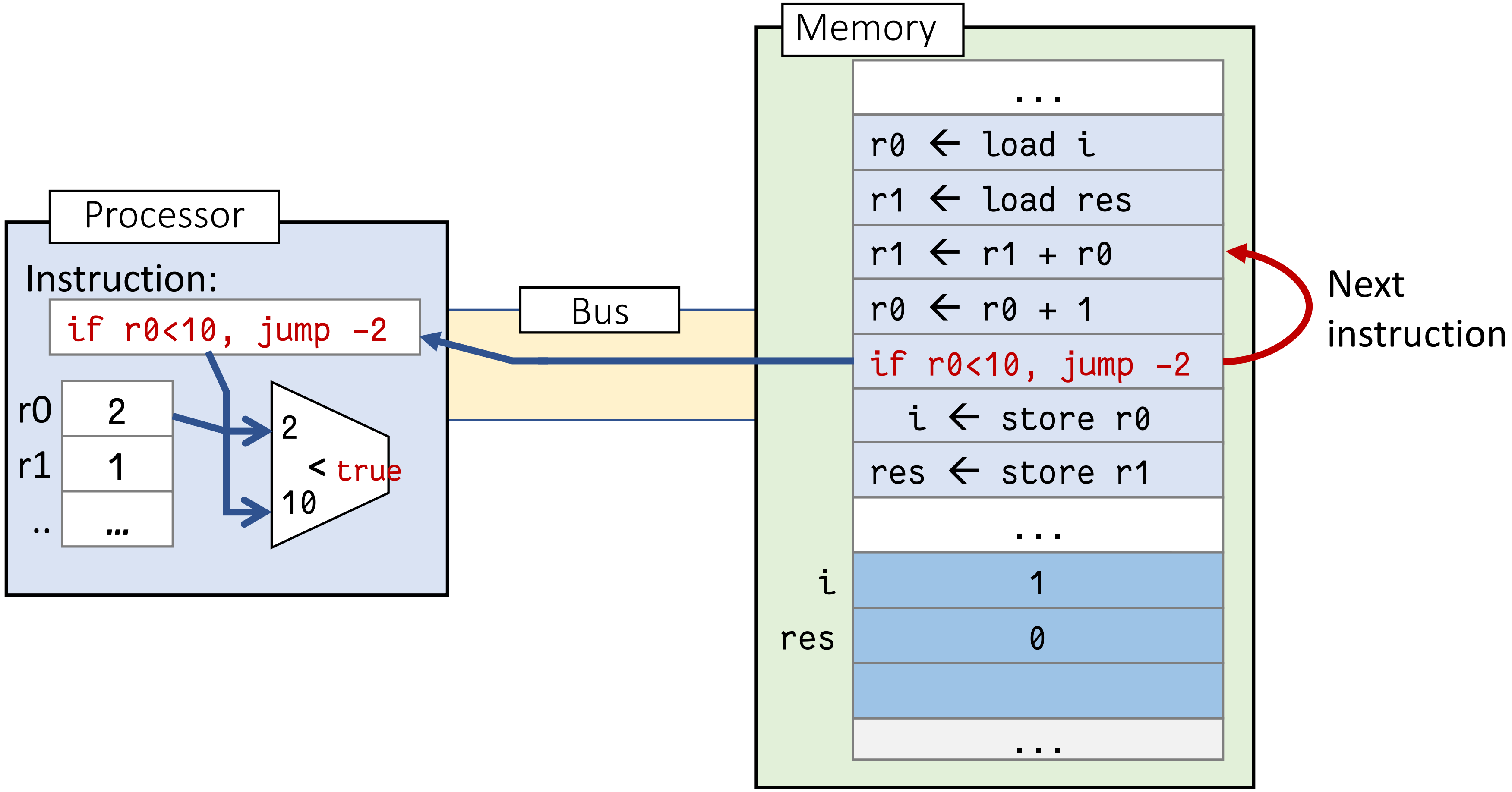
- Jump to instruction based on condition
  - Repetition (loop) and Selection (if-else) can both be supported



Walkthrough: Looping!

Since the condition is true

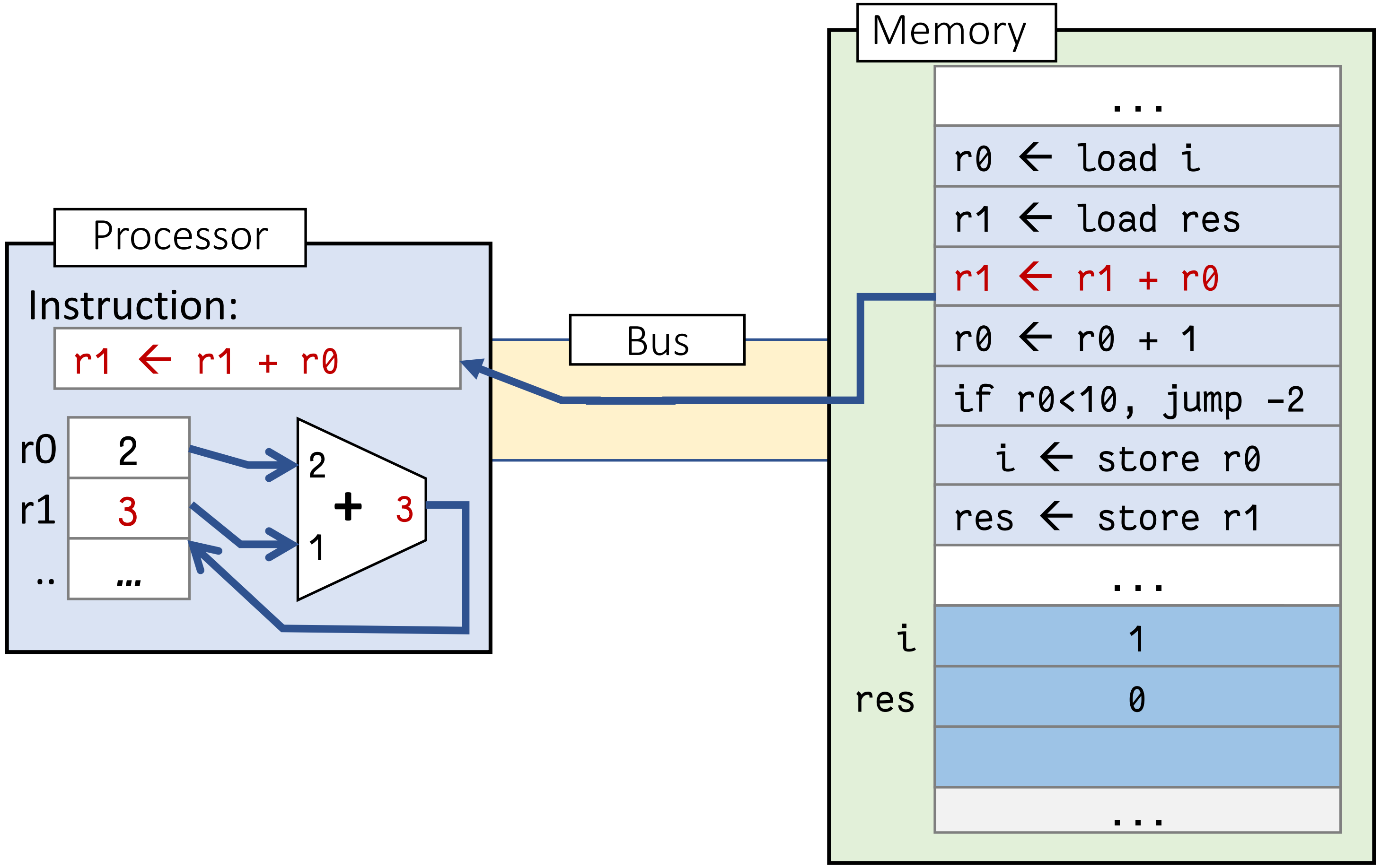
- Next instruction will “jump” to indicated position



Walkthrough: Looping!

Execution will continue sequentially:

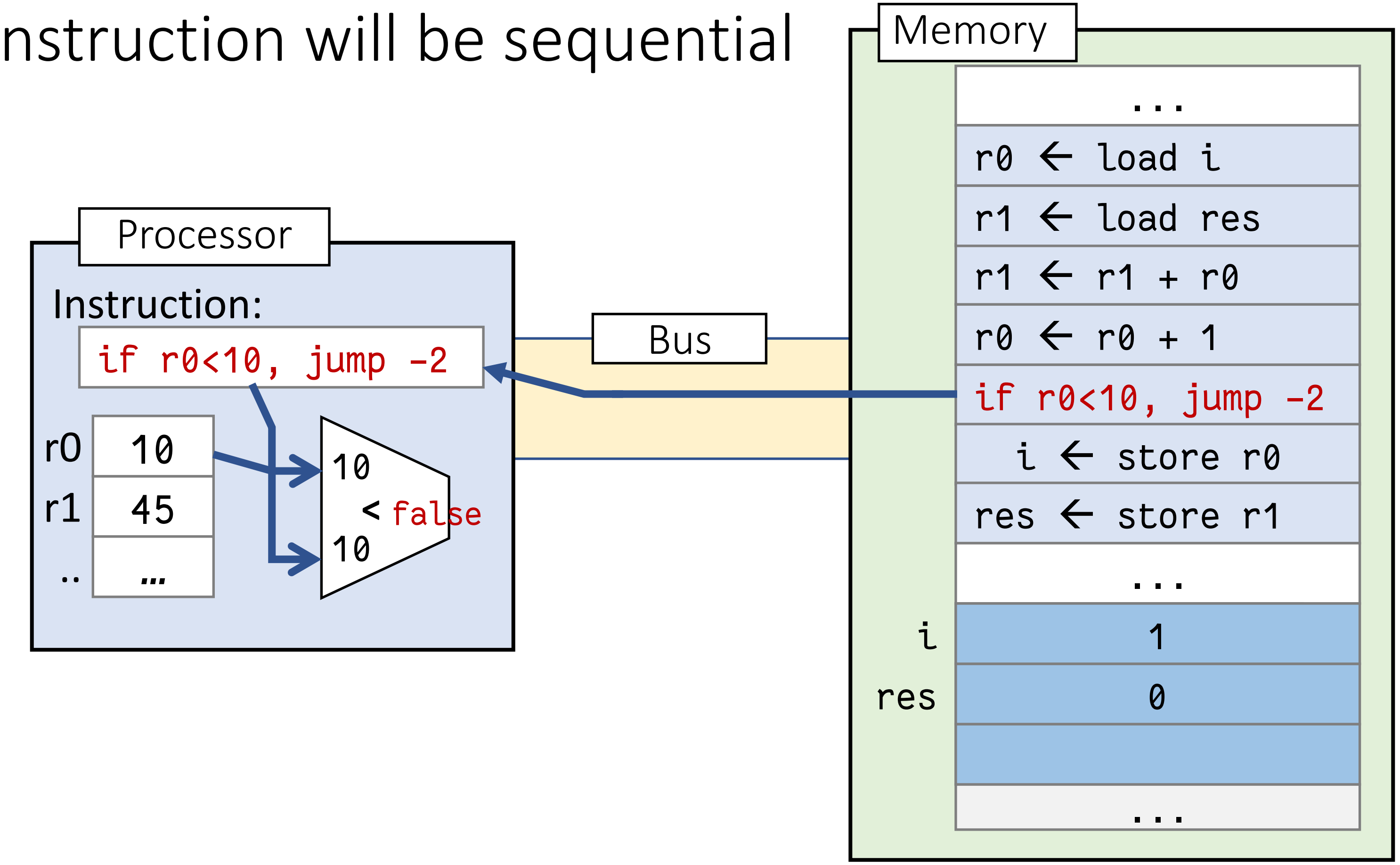
- Until we see another control flow instruction!



Walkthrough: Control flow instruction

The three instructions will be repeated until the condition fails

- No jump
- Next instruction will be sequential

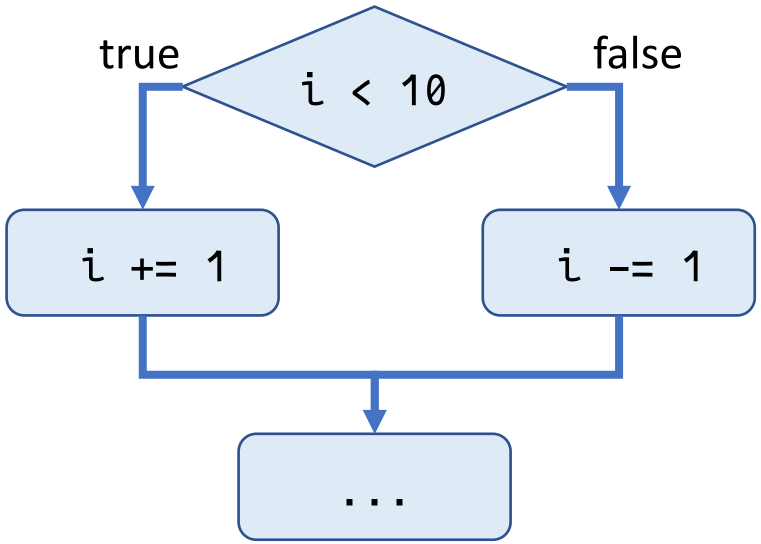


Question: How to translate if-else?

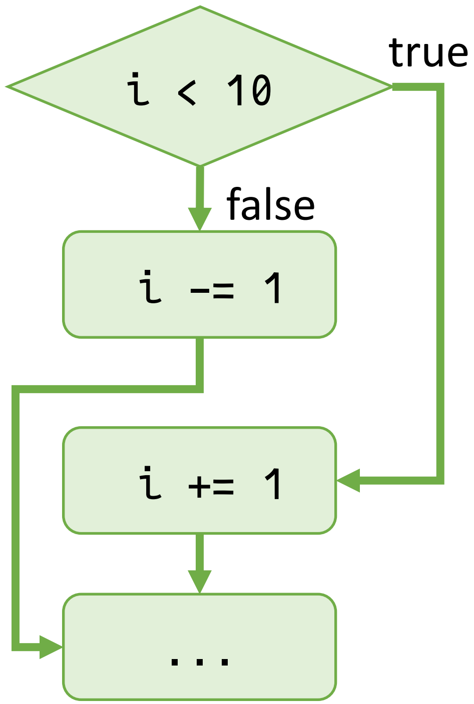
Using only conditional jumps?

```
if (i < 10) {
    i += 1;
} else {
    i -= 1;
}
...
```

How to split flow into two branches?



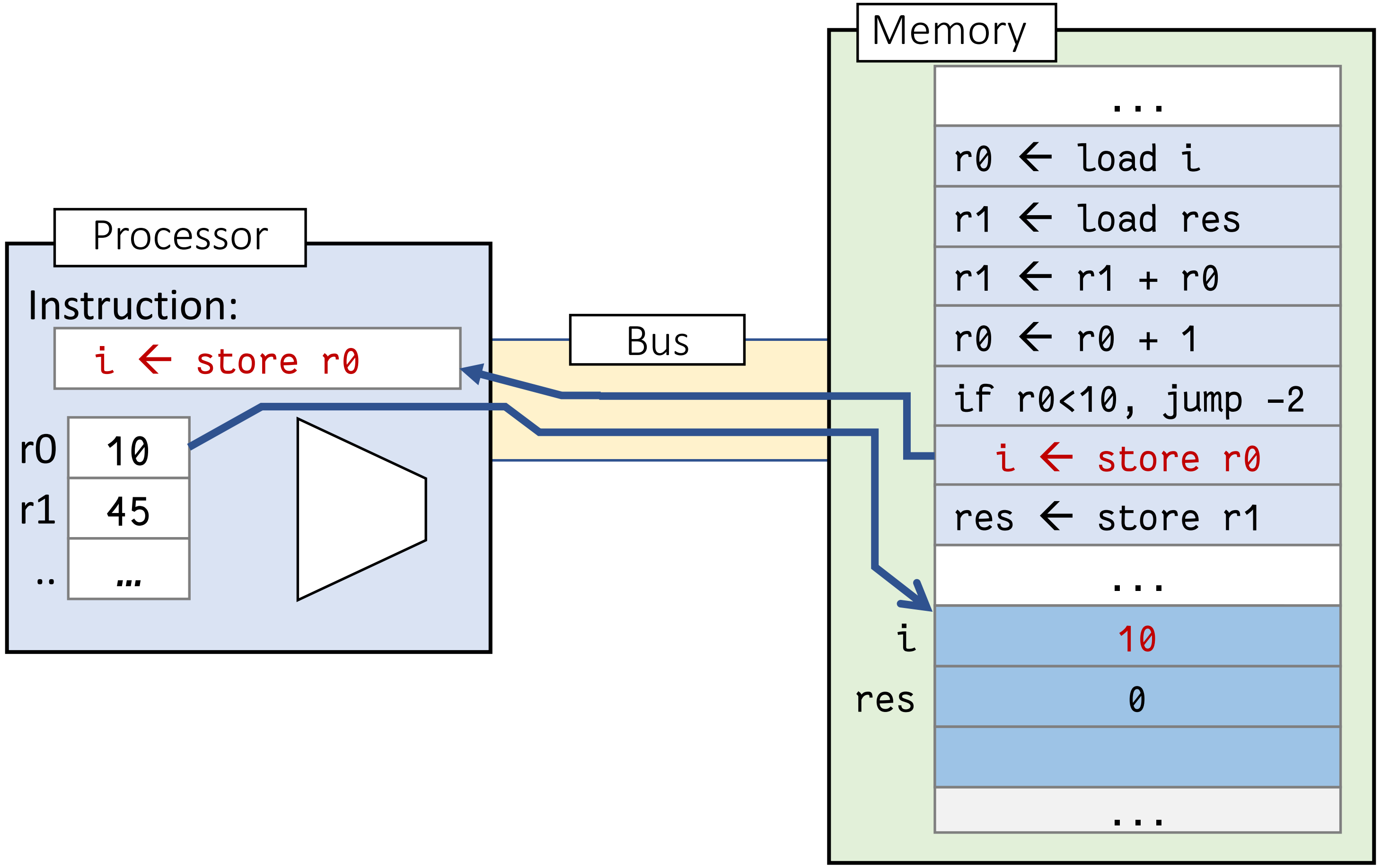
Hint: rearrange flow in one column



Walkthrough: Memory instruction

Move the register values back to their "home" in memory

- Similarly store r1 to res



Summary of observations

The stored-memory concept:
- Both instruction and data are stored in memory

The load-store architecture:
- Limit memory operations and only relies on registers for execution

The major types of instructions:
- Memory: Move values between memory and register
- Calculation: Arithmetic and Logic operations
- Control flow: Changes the sequential execution (jump to another instruction)

END