Systems Advanced: Linux Containers

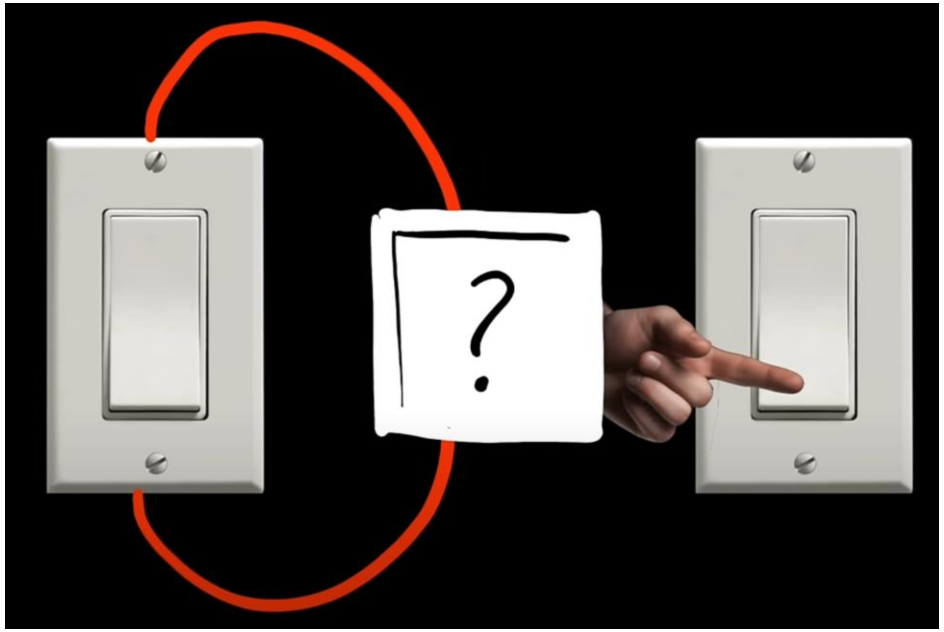
Microprocessors

PXL DE HOGESCHOOL MET HET NETWERK

Elfde-Liniestraat 24, 3500 Hasselt, www.pxl.be

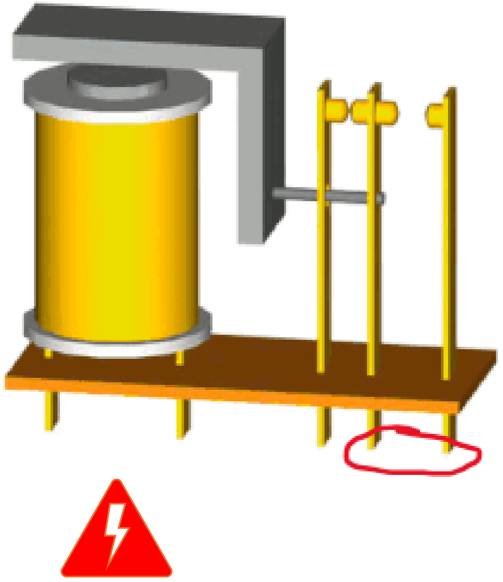
Switches

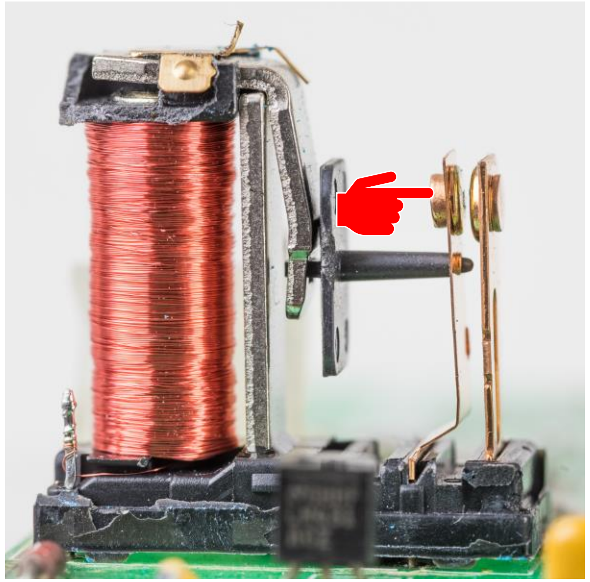
- We can control a switch mechanically, e.g. with a finger.
- What if we want to control a switch using the output of *another* switch?



Relays are electromechanical switches

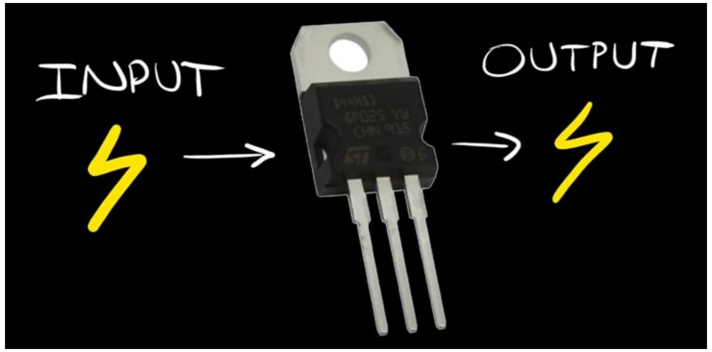
- Relays are electrically operated switches
- use an electromagnet to close or open the contacts and thus flip the switch
- moving parts!
- slow
- expensive
- prone to failure

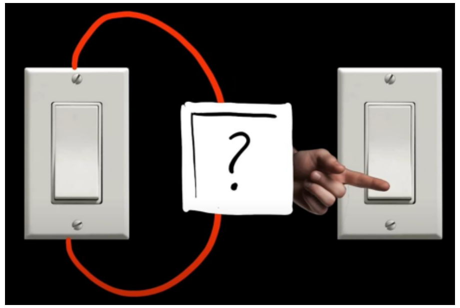


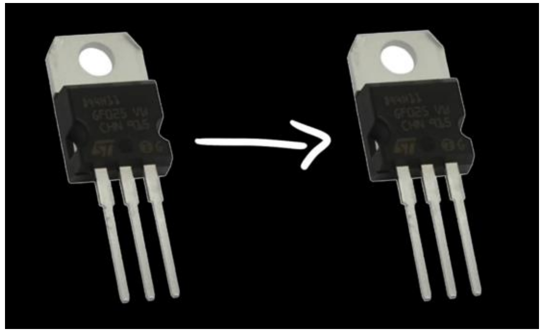


Transistors are electrical switches

- Can be used to switch (or amplify) electrical signals and power using an electrical input signal
- Output of one transistor can be used as input for another transistor
- no moving parts, only electricity
- "semiconductor" device

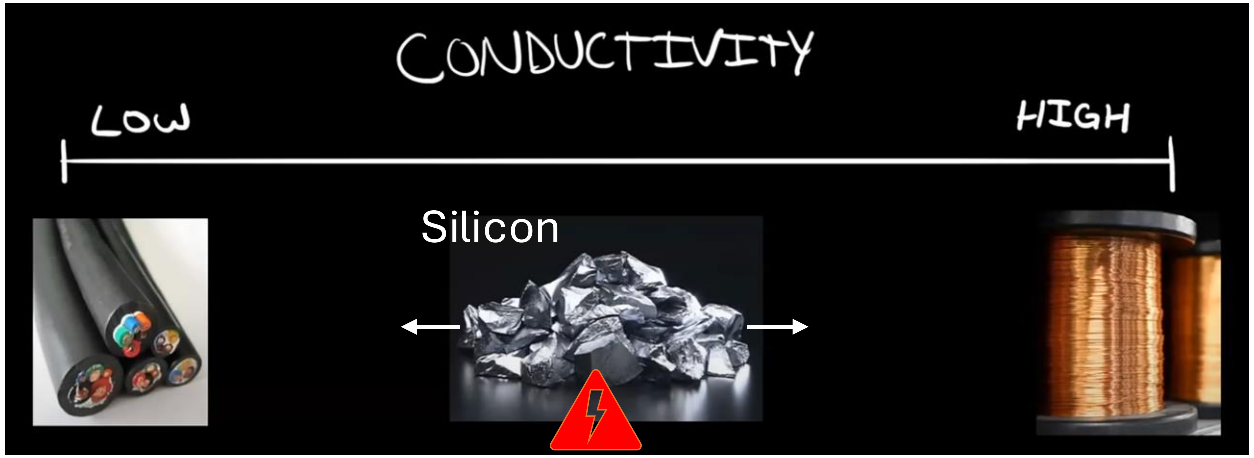


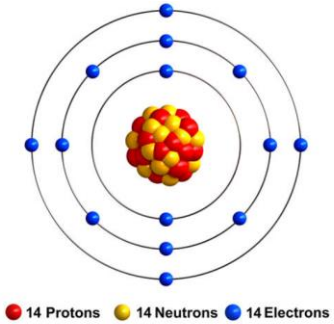




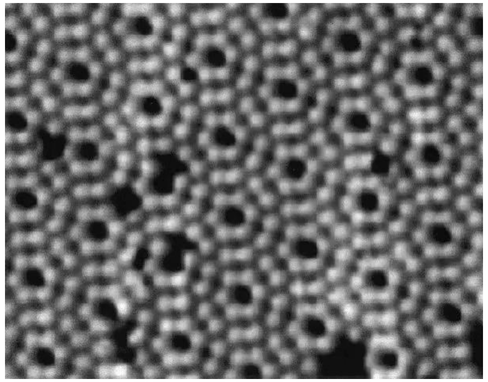
Electrical conductivity

- Electricity (= *flow of electrons*) cannot pass materials with "low conductivity", e.g. rubber. These are also called *insulators*.
- Electricity flows very well in materials with "high conductivity", e.g. copper. These are also called *conductors*.
- Special materials, like `Silicon and Germanium,` can have low conductivity **or** high conductivity depending on input electricity.
- The conductivity of these materials can change and they are called **semiconductors** (Dutch: "halfgeleiders").





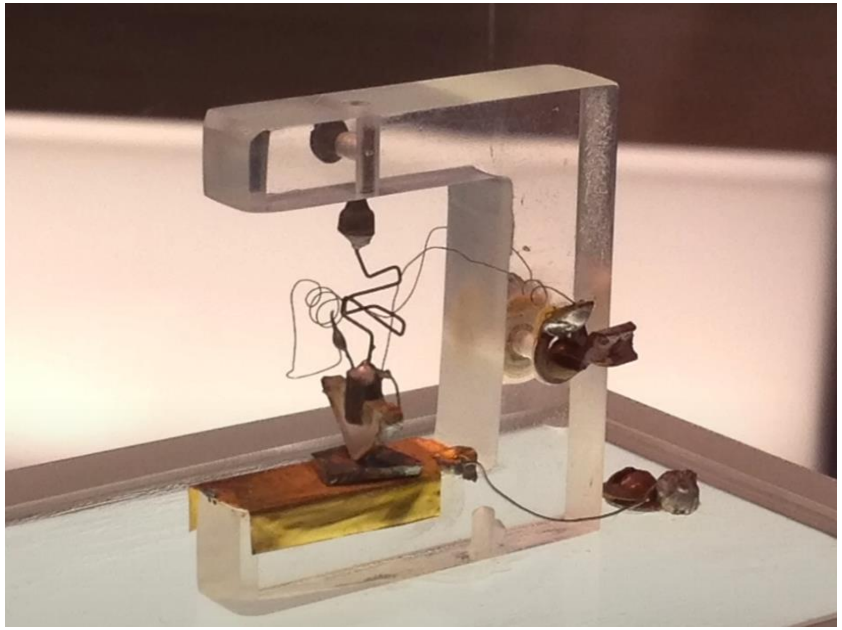
Silicon atom



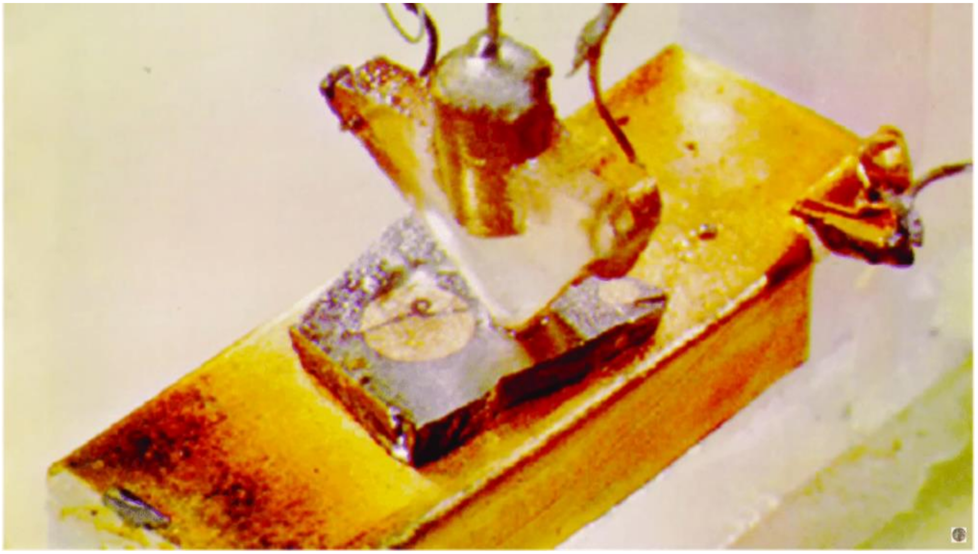
Tunneling Electron Microscope picture of Silicon atoms

Transistors

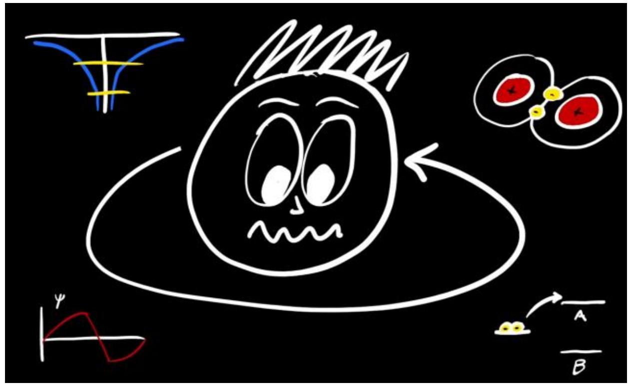
- Transistors use semiconductor materials.
- Transistors are **semiconductor devices**.
- Early transistors were large and unreliable.



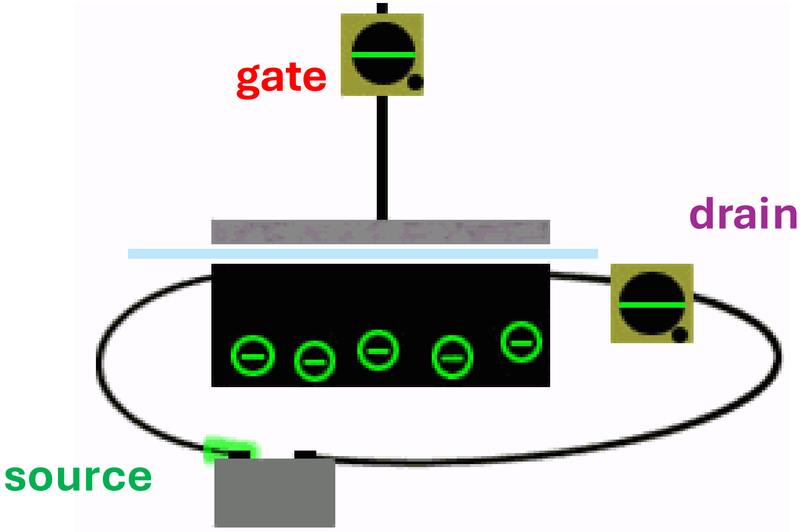
first Point Contact Transistor, 1947



Field Effect Transistors (FETs)



- Field-Effect Transistors (FETs) are transistors that use the *electrical field effect* to control the conductivity of a semiconductor material.
- The **gate** generates an electric field, separated by a thin insulating layer from the semiconductor channel between the **source** (input) and **drain** (output).
- When voltage is applied to the **gate**, it modulates the conductivity of the channel, allowing or blocking current flow between the **source** and **drain**.
- Highly reliable. Ideal for miniaturization and integration.
- E.g. MOSFETs (Metal-Oxide-Semiconductor Field-Effect Transistors) or the newer Gate-All-Around FETs (GAAFETs)



[The Actual Reason Semiconductors Are Different From Conductors and Insulators. - YouTube]

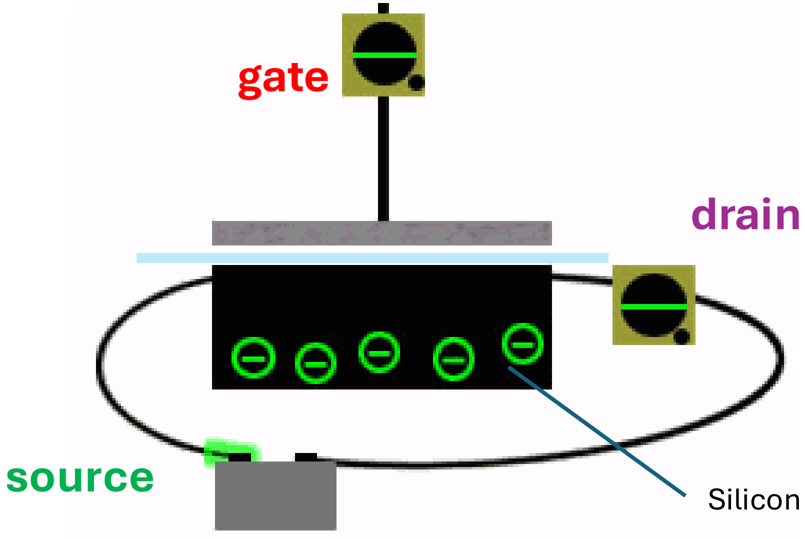
So what?

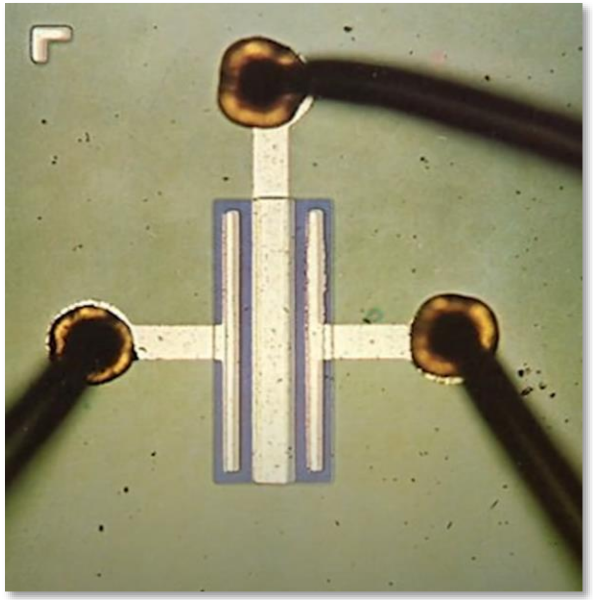
- Using electricical current we can control a switch, so what?
- HIGH/LOW Voltages
  - Transistors in digital circuits work with 2 voltages: HIGH ("1") and LOW ("0").
  - Intel/AMD CPU's: HIGH current is 0.6V-0.9V and LOW current is below 0.3V.
  - *(In the late 1940's HIGH current was 300V and very dangerous.)*
- Let's consider only one transistor
  - gate: LOW (0), source: HIGH/LOW (0 or 1) -> circuit is **OFF** and drain is LOW current (0)
  - gate: HIGH current (1) and source: HIGH current (1) -> circuit is **ON** and drain is HIGH current (1)

<u>*Note*</u>

*Transistors can also be used to **amplify** electrical signals, but that is another use case altogether. We are only focusing here on switching within digital circuits.*

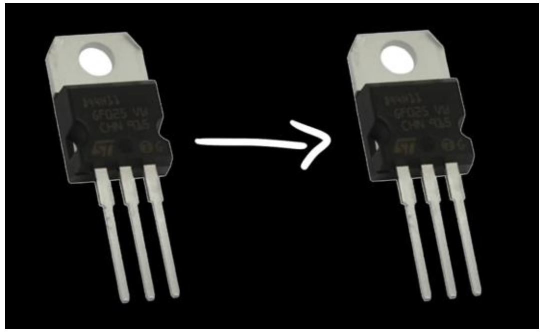
"Digital" implies only 2 precise, discrete values (LOW/HIGH Voltage) instead of an analogue Voltage range.





Logic gates

- We can **connect transistors** (switches) in series and/or in parallel to create **logic gates**.
- If we have logic gates we can build a modern computer that is ***Turing complete***:
  - can write/access to memory storage
  - can do conditional branching. (if-then)
  - => can perform any computation given enough time and storage.

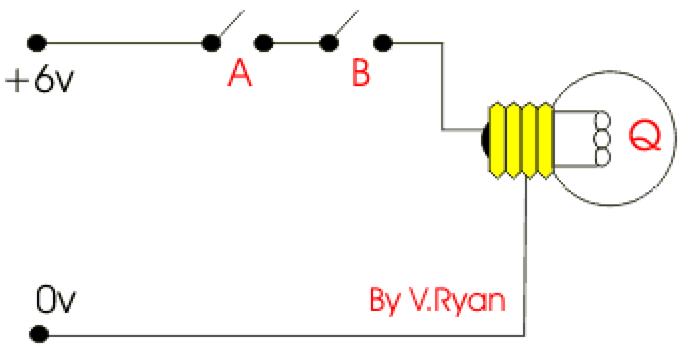


Alan Turing

Logic gates

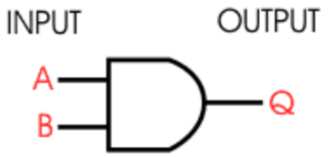
- A logic gate is a device that performs a **Boolean function**, a logical operation performed **on one or more binary inputs** that produces a **single binary output.**
- Let's make an AND and an OR gate with actual transistors A and B and look at the truth tables:

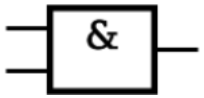
AND GATE



AND gate

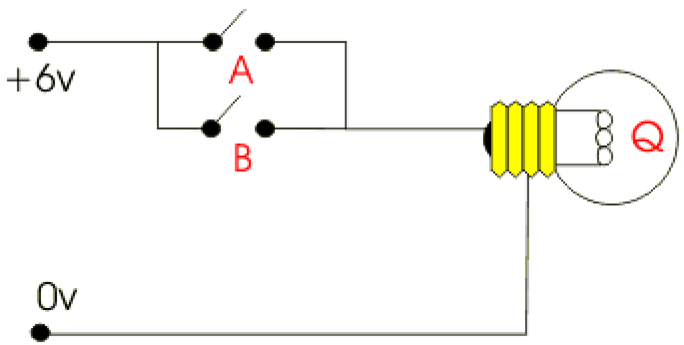
A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1





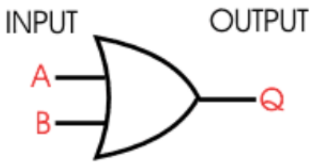
IEC Symbol

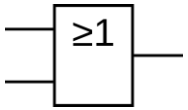
OR GATE



OR gate

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

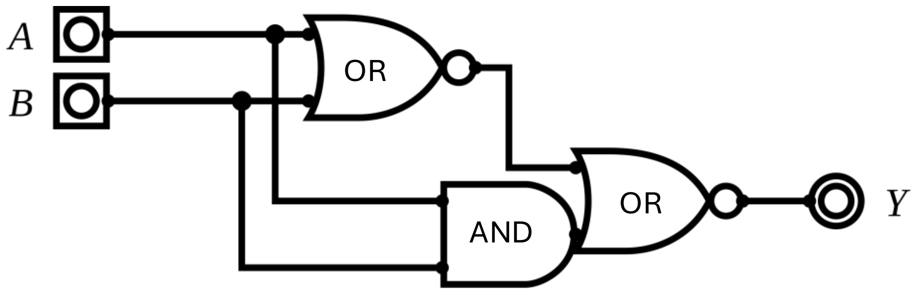




IEC Symbol

Complex logic gates: XOR

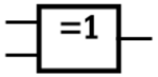
- Use simple logic gates to create more complex logic gates



XOR gate truth table

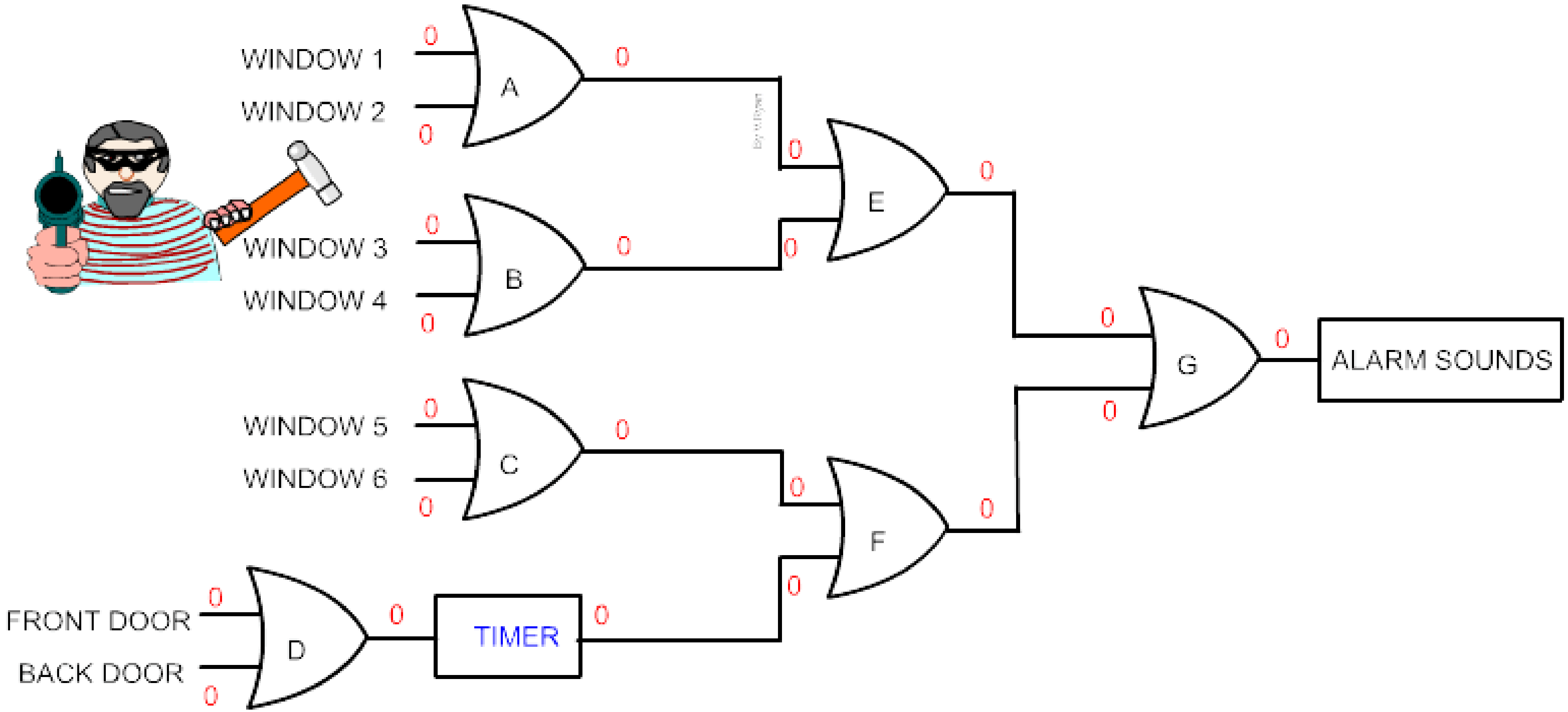
Input		Output
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

ANSI XOR Schematic Symbol



IEC XOR Schematic Symbol

Using logic gates with transistors in actual digital circuits



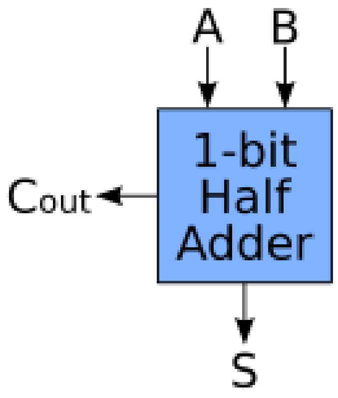
Example: digital circuit with 14 transistors and 1 timer circuit

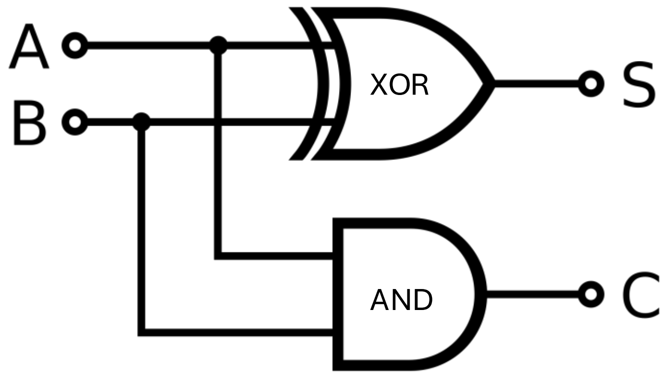
LAB

- check out the logic gates AND, OR and XOR
- [LogiJS: library__0gates]
    - press the "Start" button
    - click on inputs
    - observe logic gates
        - AND
        - OR
        - XOR

Use logic gates to create mathematical circuits

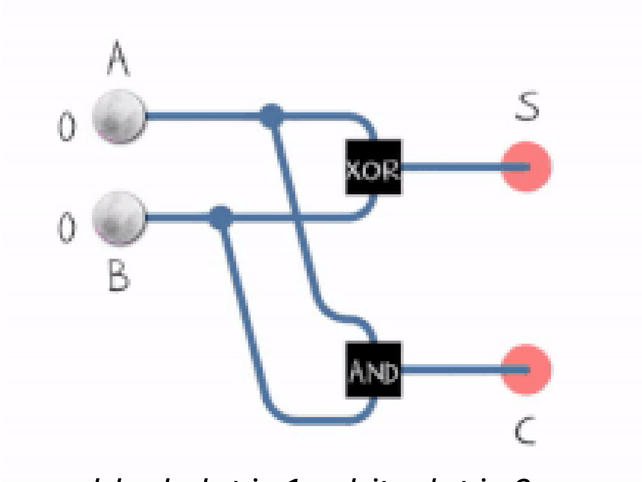
- Let's create a 1-bit "half-adder":
  - The half adder adds two single binary digits A and B.
  - It has two outputs, Sum (S) and Carry (C)
  - The carry signal represents an overflow into the next digit of a multi-digit addition.





The truth table for the half adder is:

Inputs		Outputs	
A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

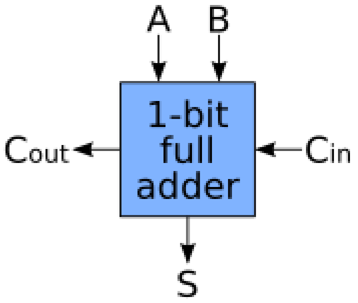


black dot is 1, white dot is 0

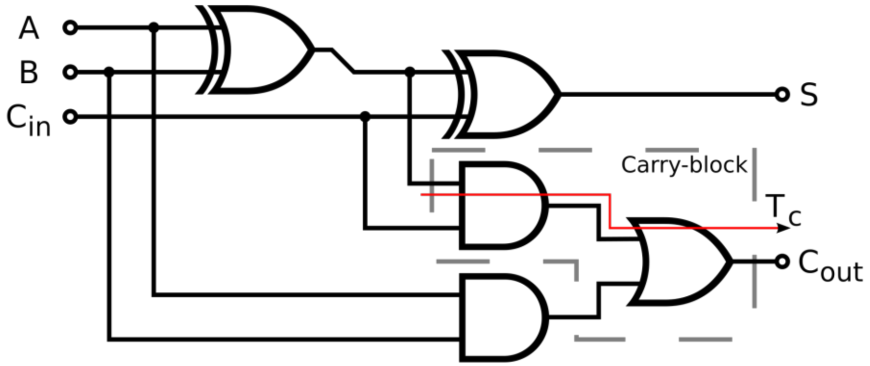
LAB

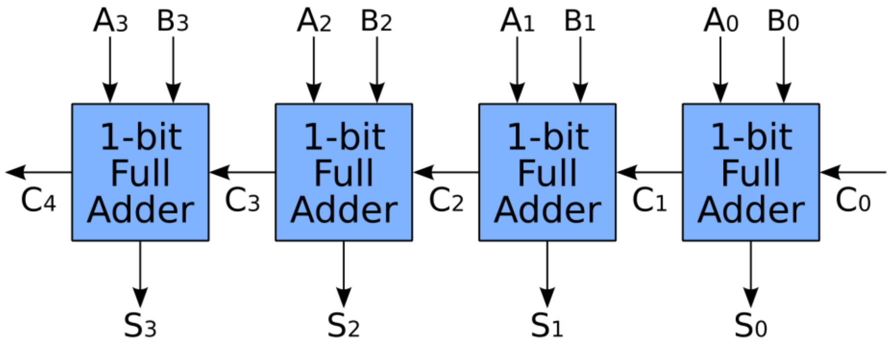
- check out the half adder circuit, built with logical gates
- LogiJS: library__1halfadder
    - press Start
    - click on inputs
    - observe

Etc…



- A full adder takes into account Carry bits as input

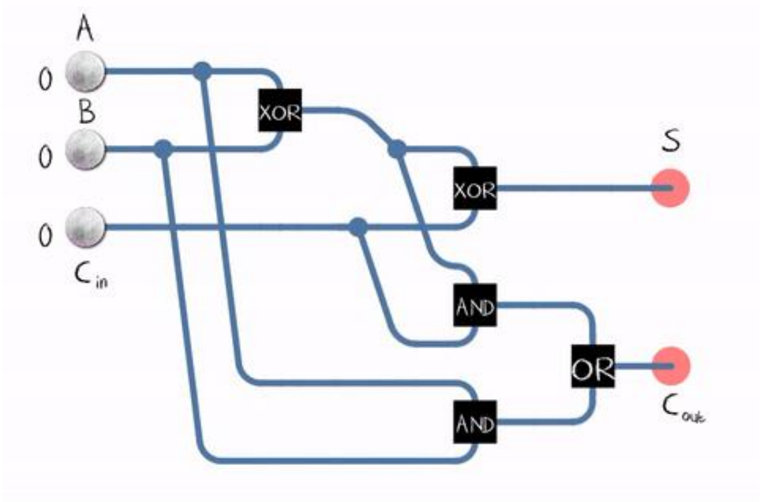




4-bit adder

The truth table for the full adder is:

Inputs			Outputs	
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



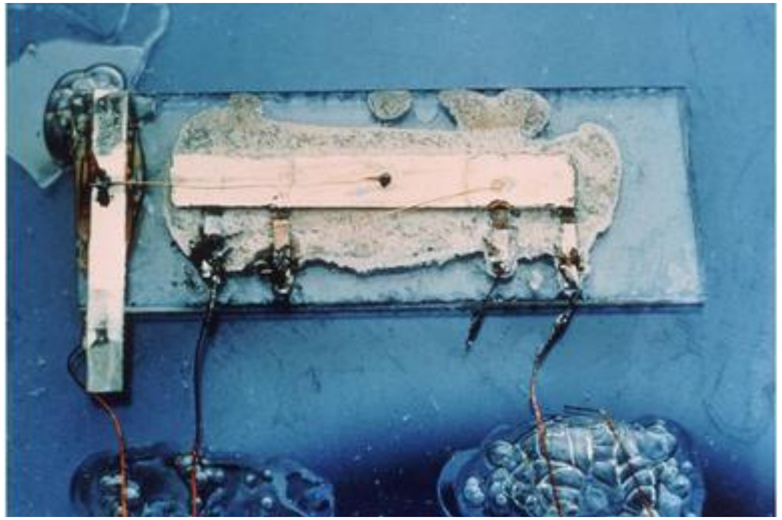
LAB

- check out the full adder circuit, built with logical gates
- LogiJS: library__2fulladder
  - press Start
  - click on inputs
  - observe

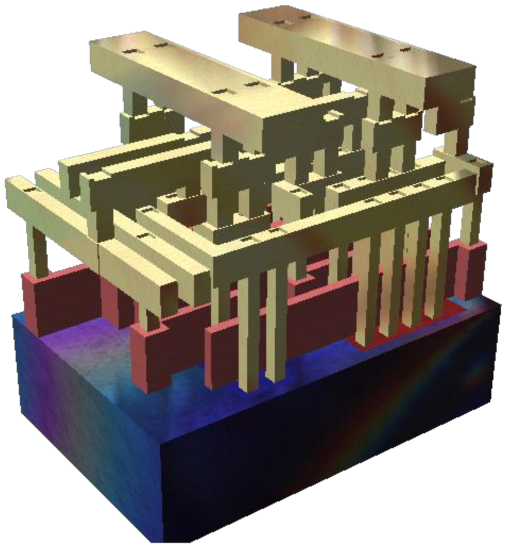
- PS: LogiJS works best with Light Theme

Miniaturization & VLSI

- Basic gates like AND, OR, and NOT are interconnected to create complex functionalities like arithmetic units and control circuits.
- MOSFETs and other FETs are super well suited for miniaturization, using a complex process and enable Very Large Scale Integrastion (VLSI).
- Modern CPUs and GPUs are created using advanced lithography techniques to fabricate billions of transistors on a single chip.
- E.g. NVIDIA's Blackwell GPU packs *208 billion transistors* and is manufactured using a TSMC 5nm process.

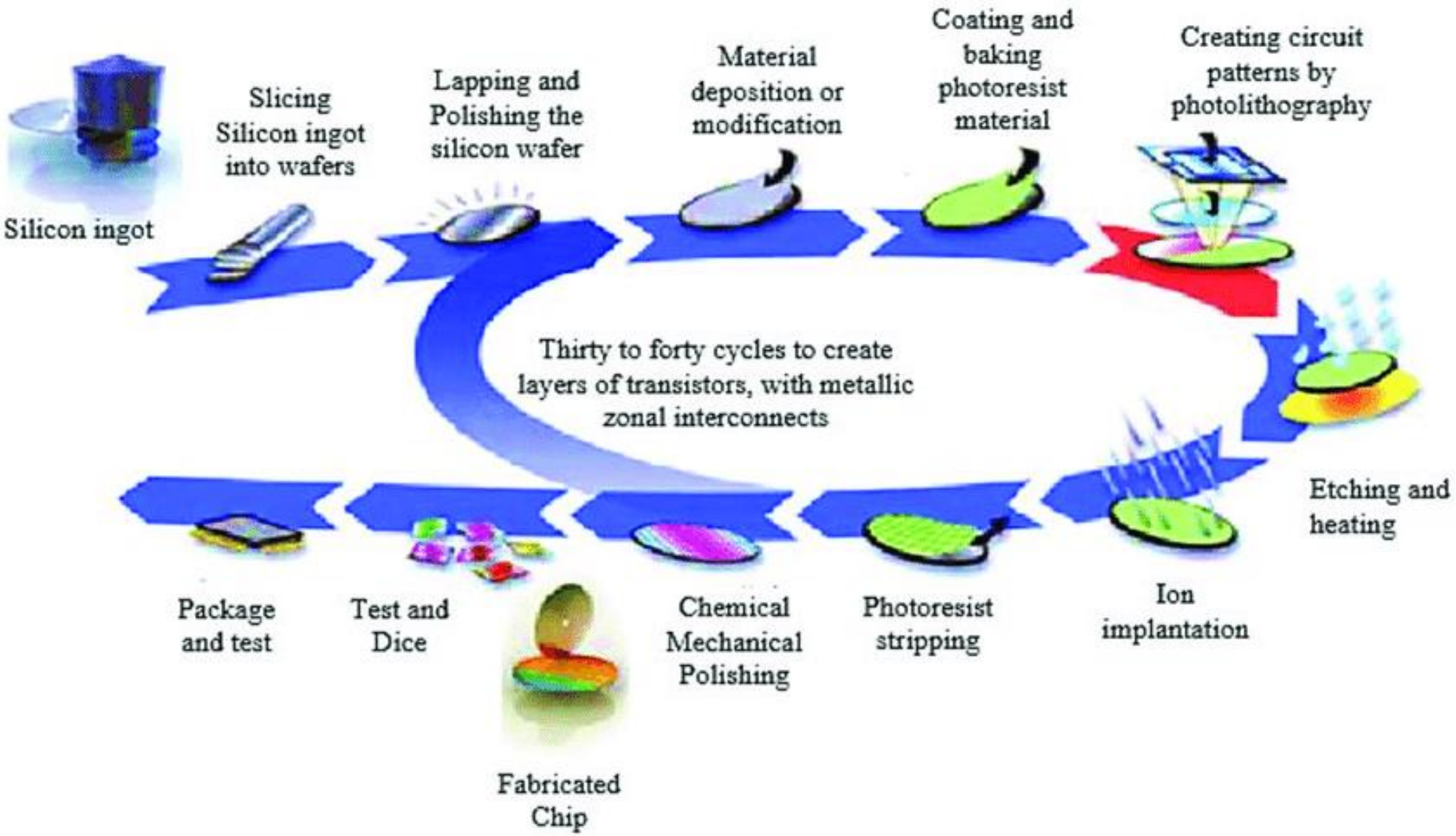


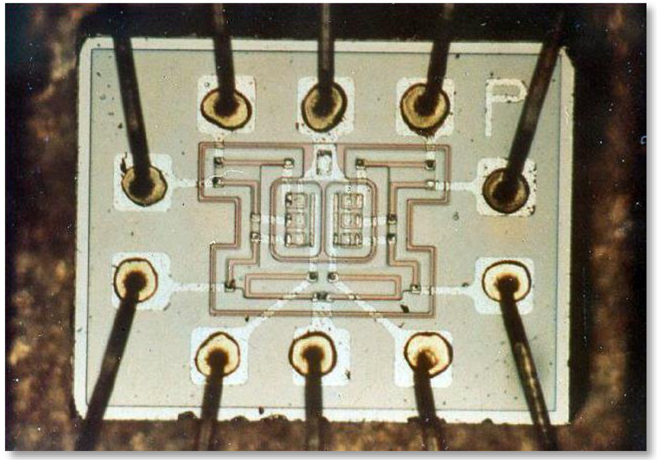
first integrated circuit, Jack Kilby, 1958



design of metal IC interconnects (2000)
it's all about the wiring!

Miniaturization & VLSI





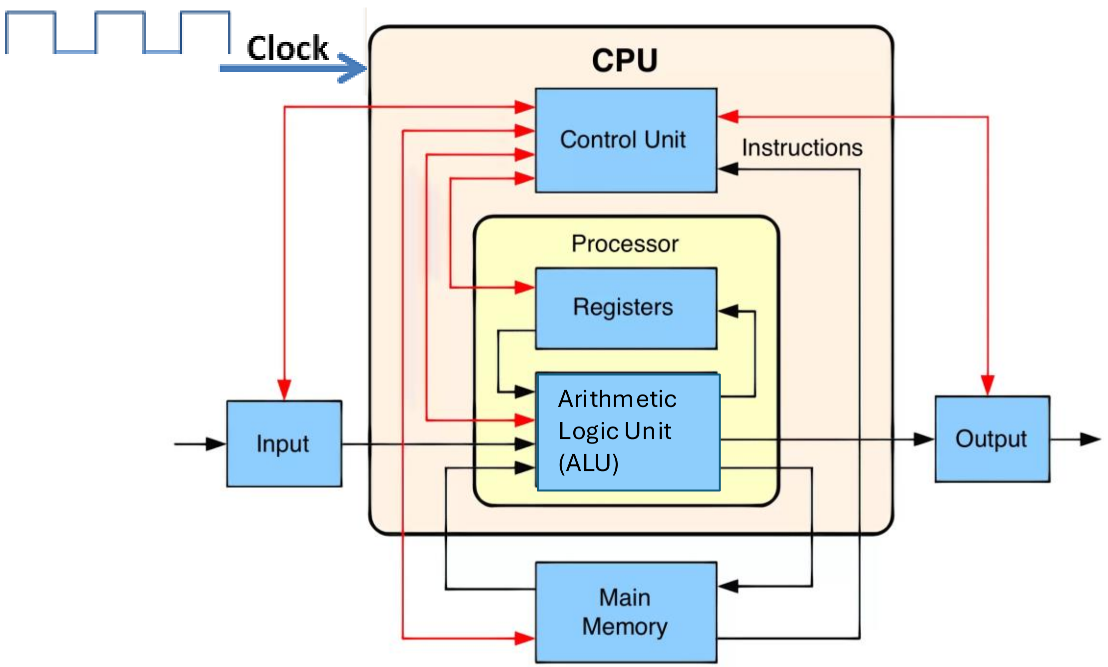
Logical NOR IC from the computer that controlled the Apollo spacecraft

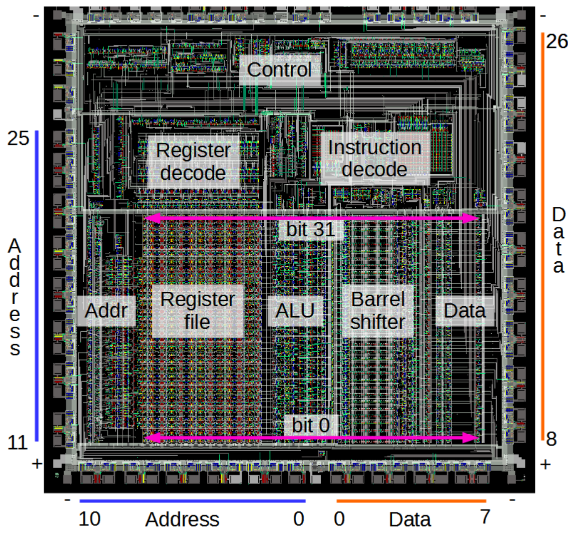
Imec Headquarters in Leuven, Belgium

imec: The Semiconductor Watering Hole

What is a CPU

- A general purpose CPU implements an **Instruction Set Architecture** (ISA) and consists of:
    - **Control Unit** (**CU**): Directs operations within the processor.
    - **Arithmetic Logic Unit** (**ALU**): Handles arithmetic and logical operations.
    - **Registers**: small, high-speed storage locations for immediate data access.
    - *and some fast cache internal on-chip memory*
- The CPU will access slower external **Random-Access Memory** (**RAM**).



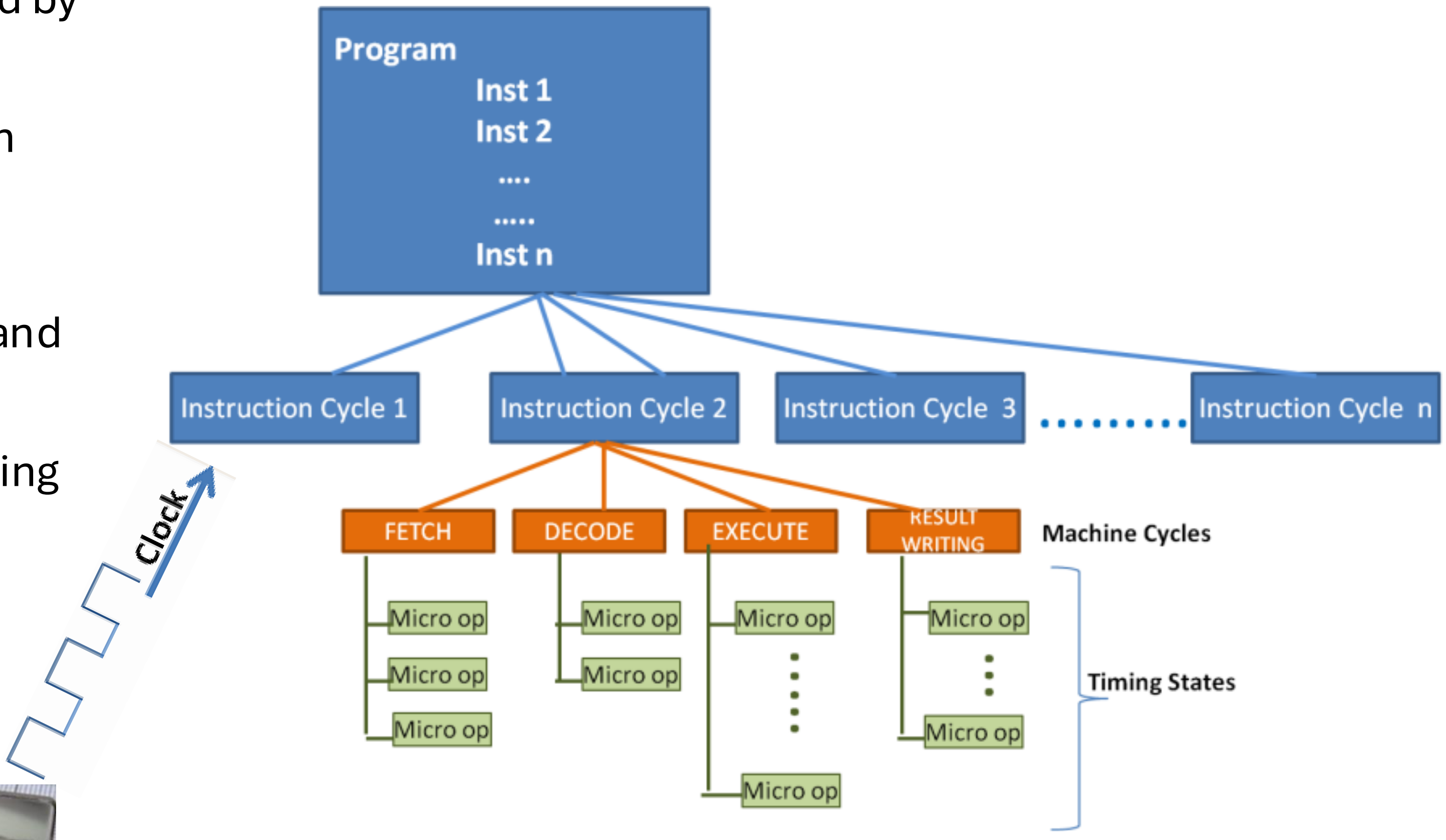


Archimedes Risc Machine 1 (ARM1)

The CPU Instruction Execution Flow

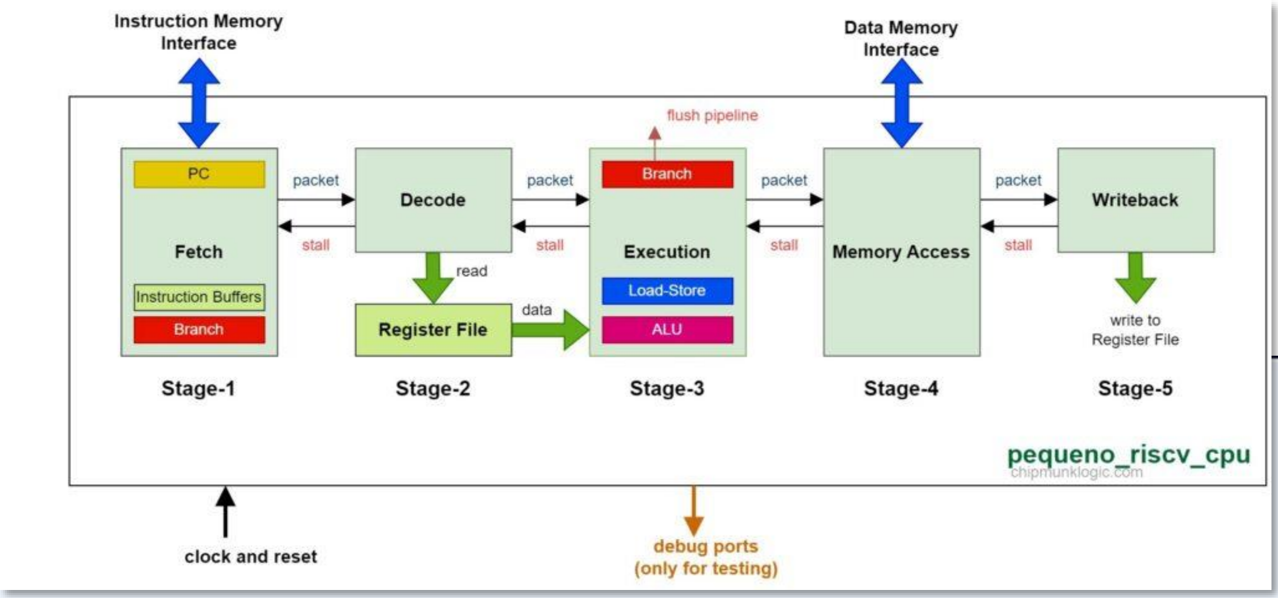
The **Instruction Cycle** is synchronized by the **clock signal**:

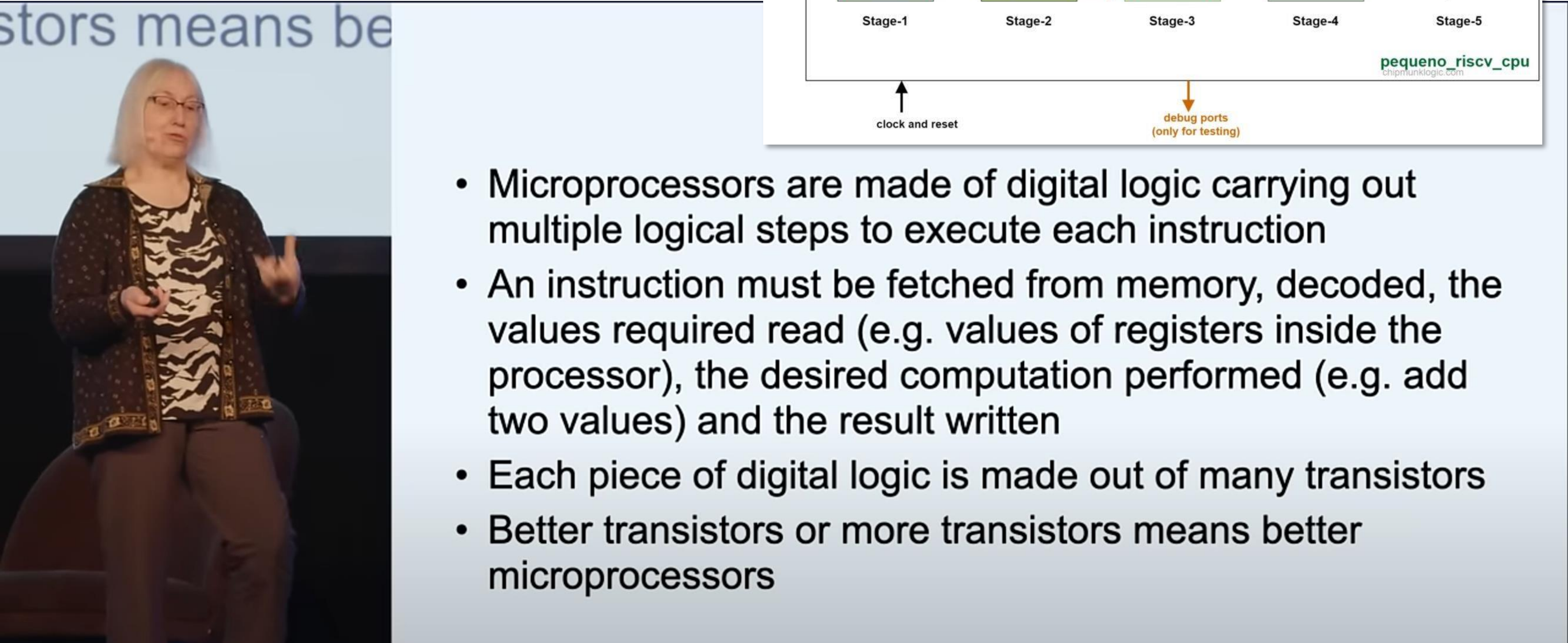
- **Fetch** – Retrieve the next instruction from memory into the Instruction Register (IR).
- **Decode** – Interpret the instruction and determine the required operation.
- **Execute** – Perform the operation using the ALU, registers, or memory.
- **Write Back** – Store the result in a register or memory if needed.



oscillating quartz crystal clock

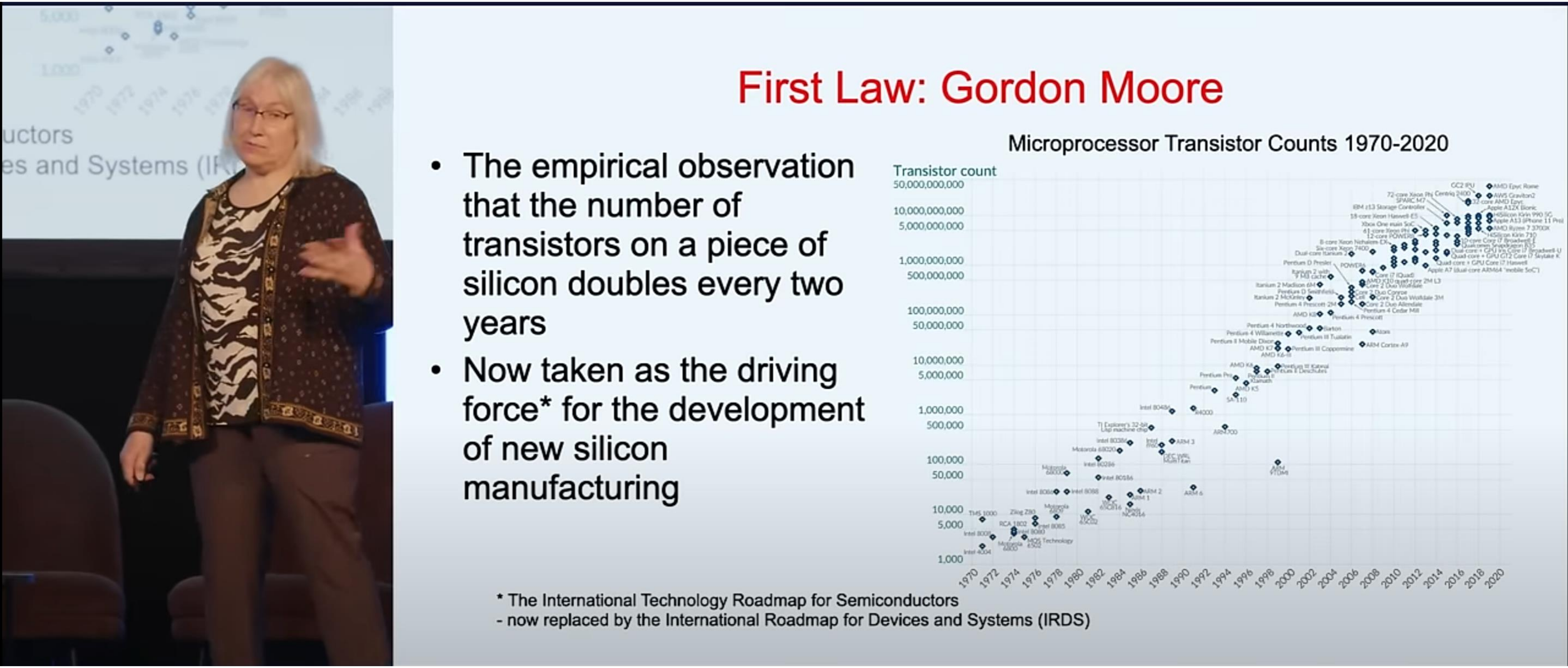
Microprocessors





The Future of Microprocessors • Sophie Wilson • GOTO 2024

Moore's Law



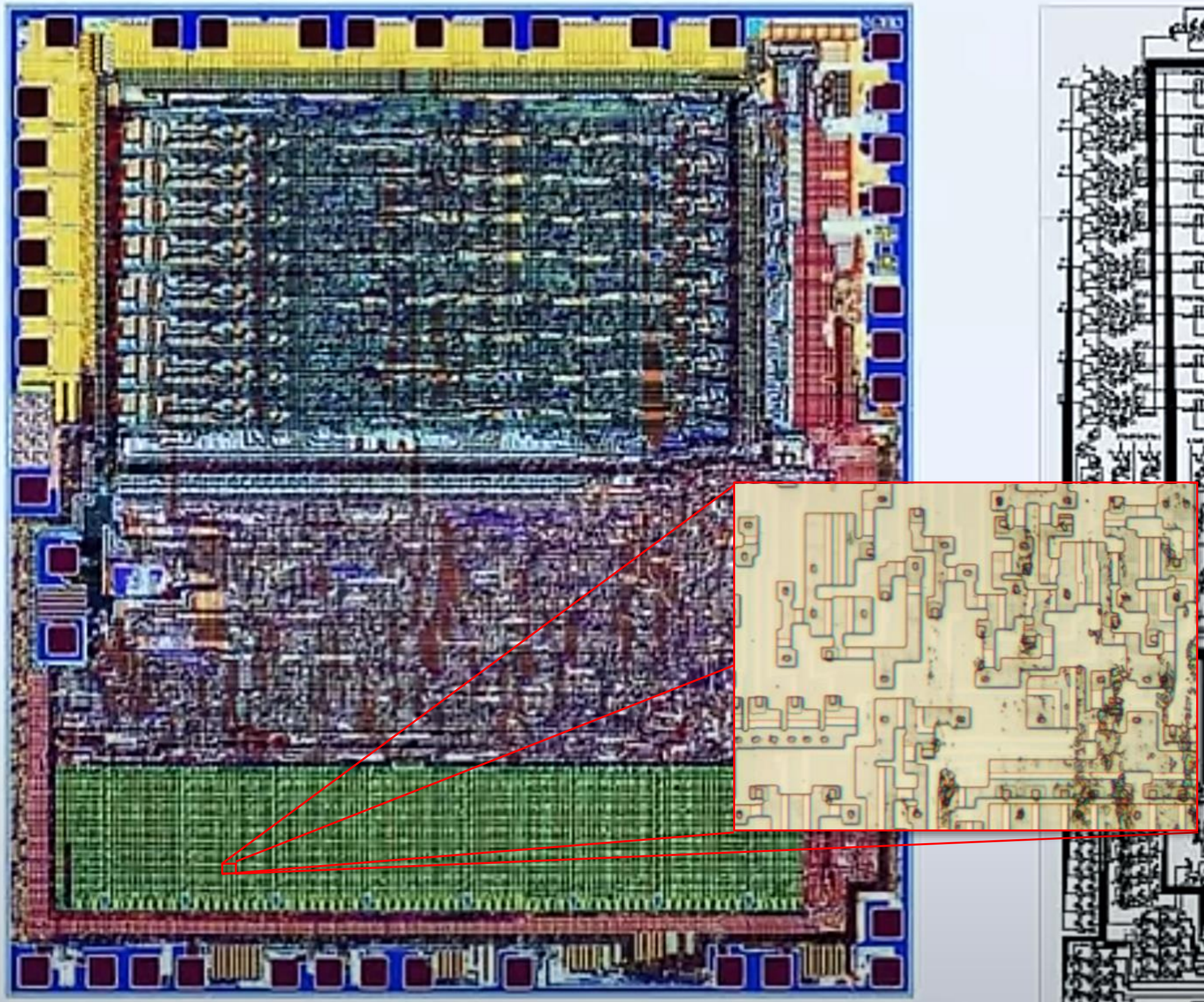
The Future of Microprocessors • Sophie Wilson • GOTO 2024

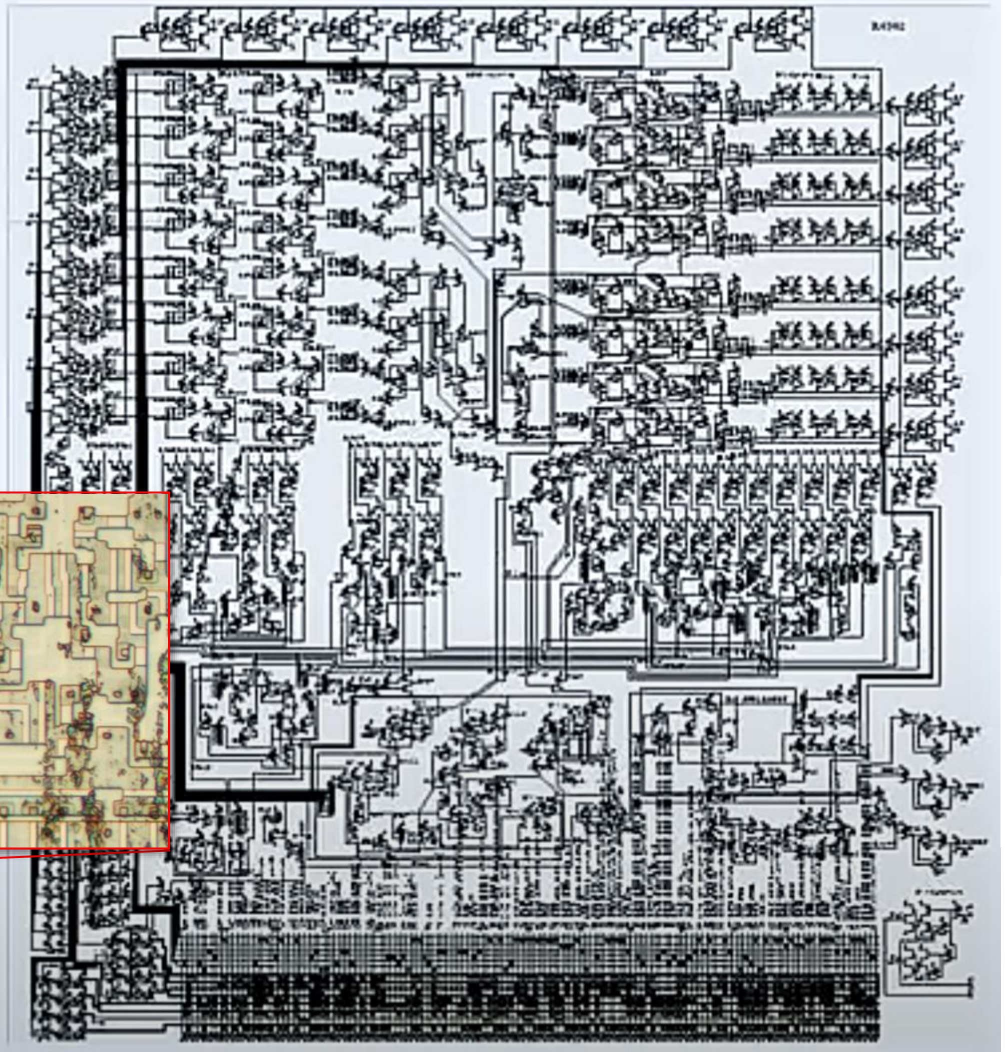
What does this mean?

- On the wall is a plot of the ARM1 in the 3 micron process it was designed in
- To the same scale, in my hand is a plot of the ARM Cortex M0+ in a 20nm process – its the small black dot

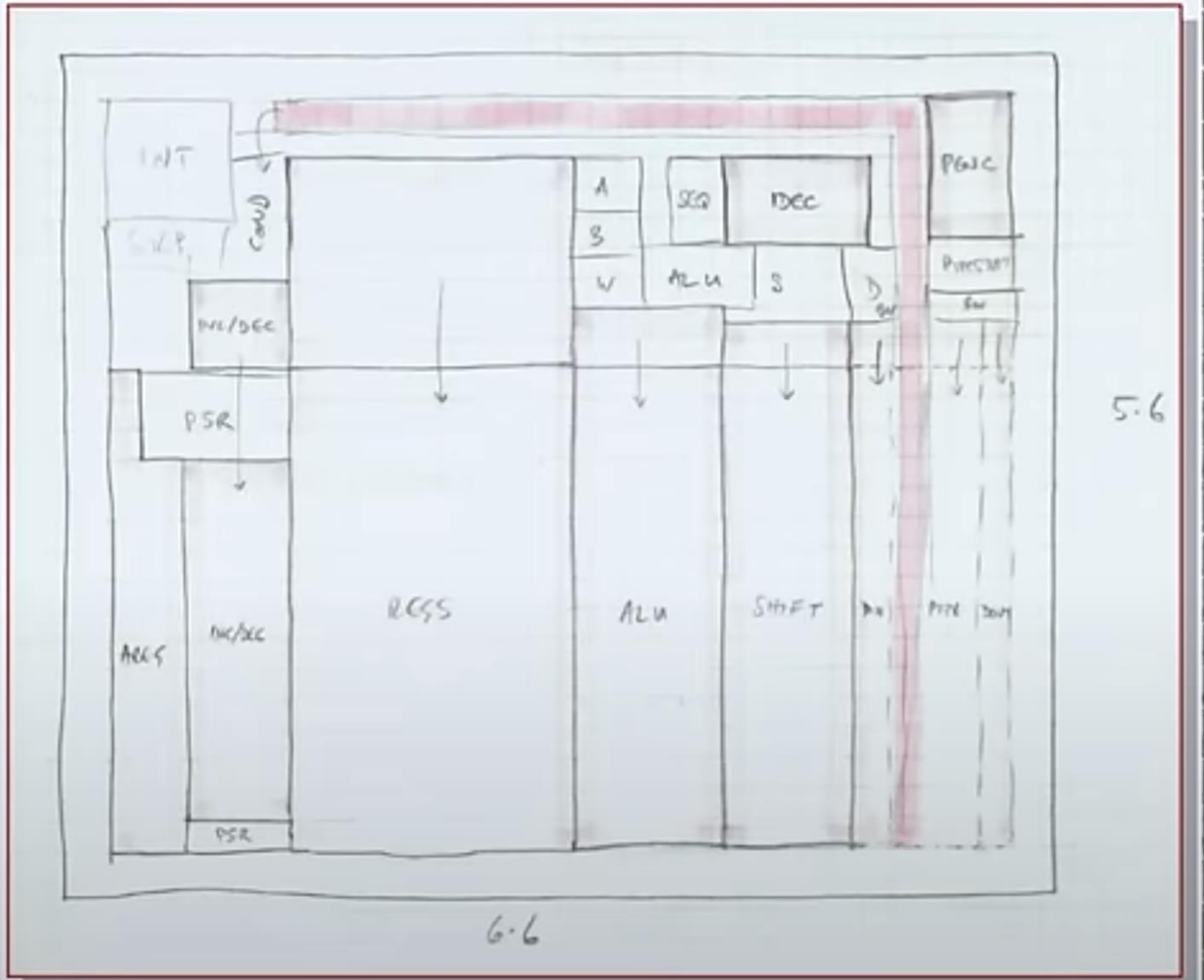
Sophie Wilson, 2013

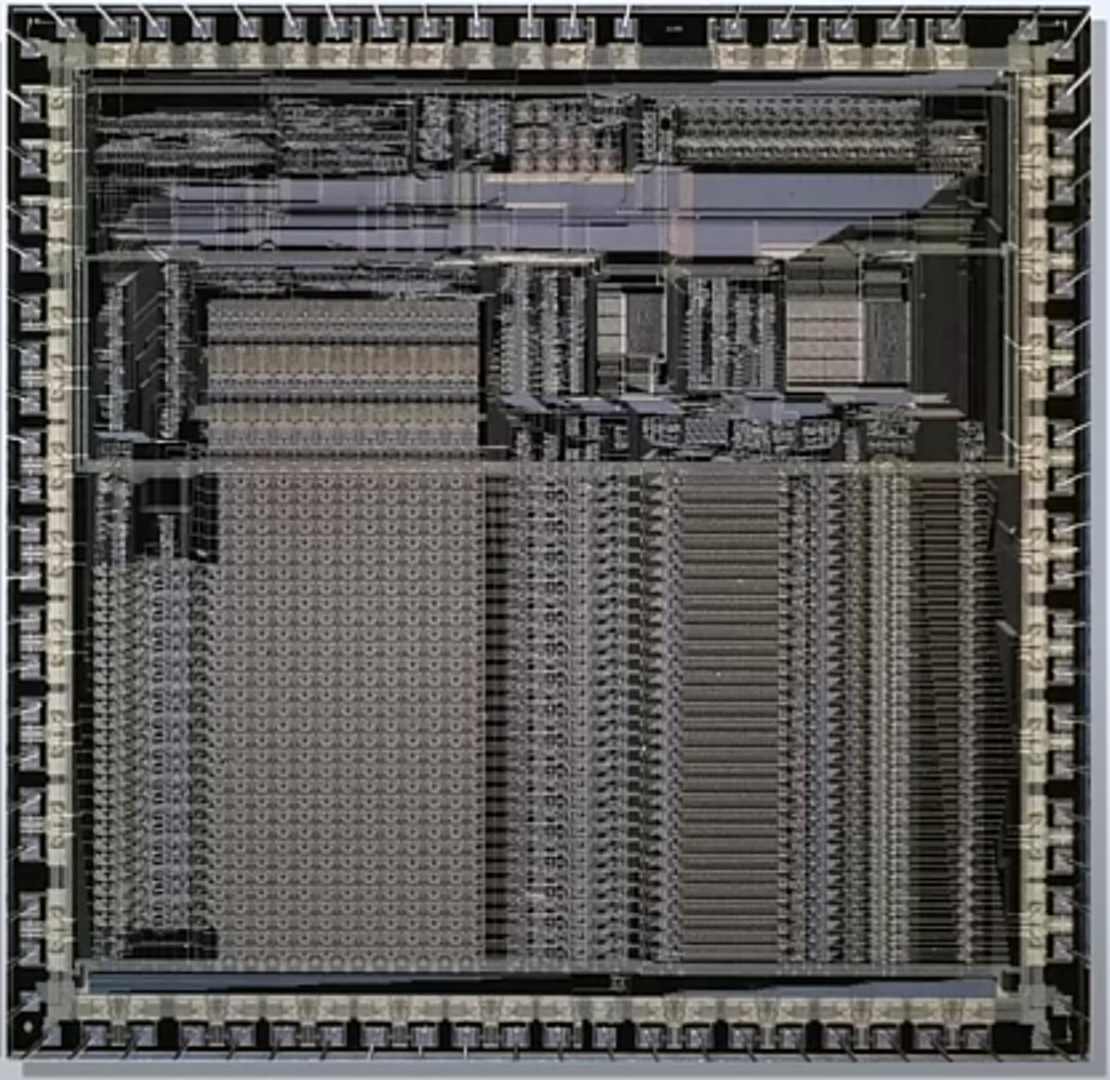
6502 – 4 thousand transistors - 1975





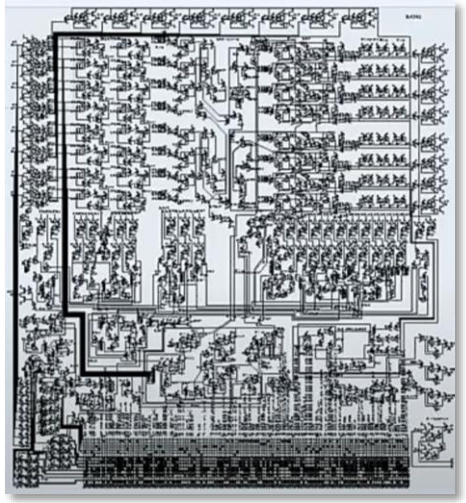
ARM1 – 25 thousand transistors 1985

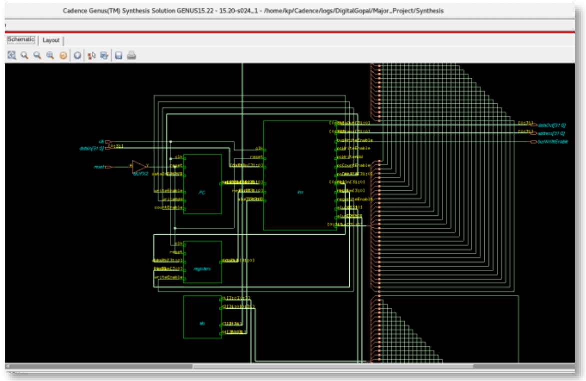




Faster CPUs Enable Better CPU Design Tools

- Early IC and CPU design was done manually, with engineers drawing layouts by hand and physically placing components on silicon.
- As Computer-Aided Design (CAD) workstations emerged, they enabled more efficient CPU designs. This led to:
  - More complex circuits and wiring optimizations.
  - Faster CPUs, which in turn made even more powerful CAD tools possible.

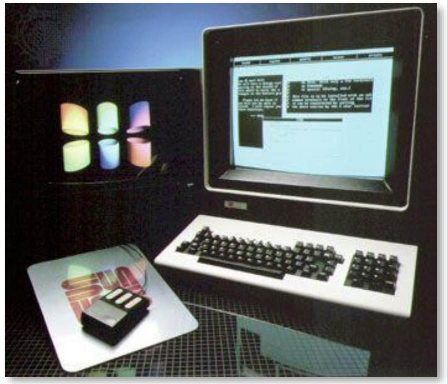




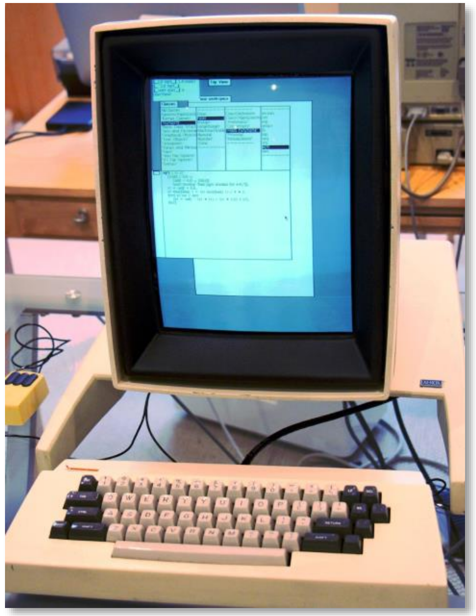
Apollo ND100 CAD Workstation, 1981

Faster CPUs Enable Better CPU Design Tools

- These CAD systems ran on powerful workstations that required:
  - Networked, time-sharing operating systems (e.g., UNIX variants on Sun, Apollo, and SGI systems).
  - Advanced graphical capabilities to visualize chip layouts and simulations.
- Innovations from workstation environments influenced graphical user interfaces (GUI) in mainstream computing, from the Xerox Alto to later commercial systems like MacOS, Windows, and UNIX/Linux/X11.

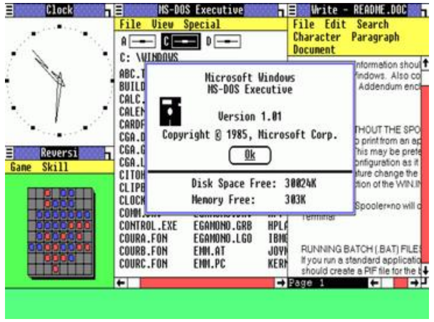


Sun-1 UNIX graphical workstation, 1982



Xerox Alto, 1973
first GUI, keyboard, mouse

Apple Macintosh, 1984

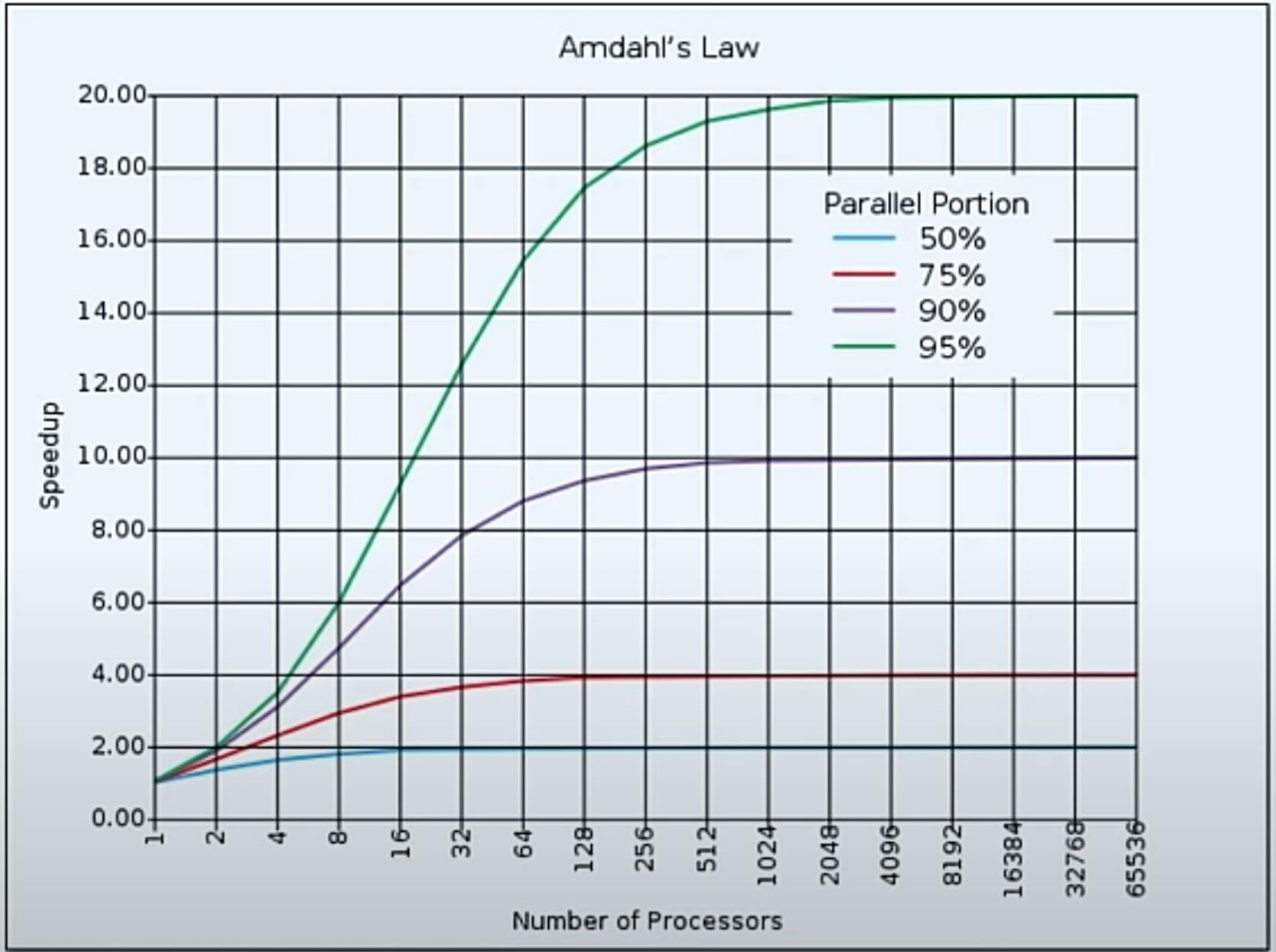


Microsoft Windows 1.0, 1985

Second Law: Gene Amdahl

- Speedup of multiple processors is limited by the sequential part of the programme
- Speedup for N processors is

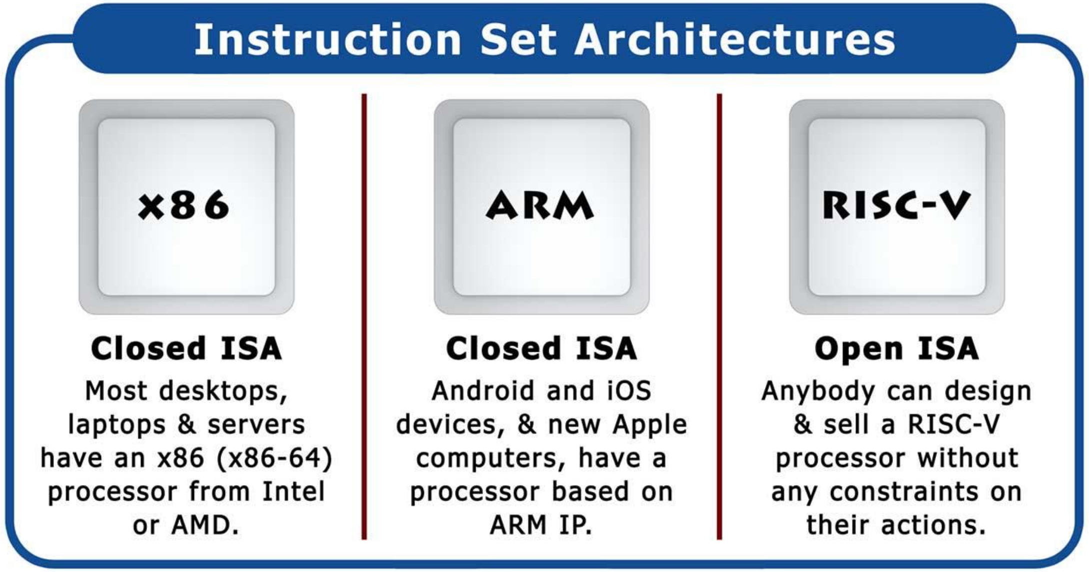
$$\frac{1}{(1-P)+\frac{P}{N}}.$$

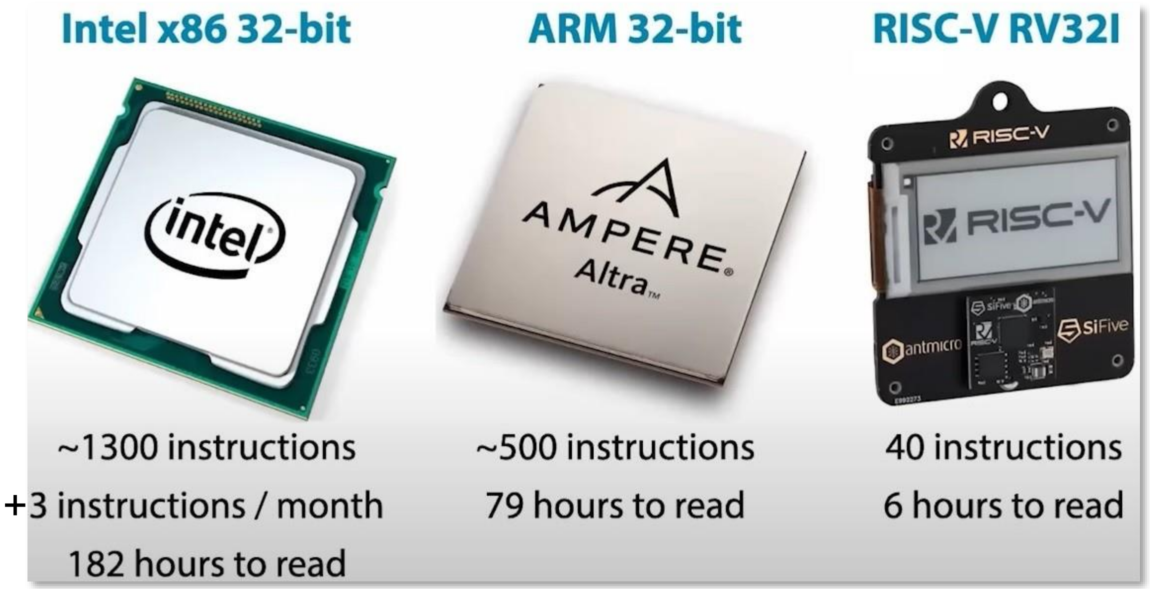


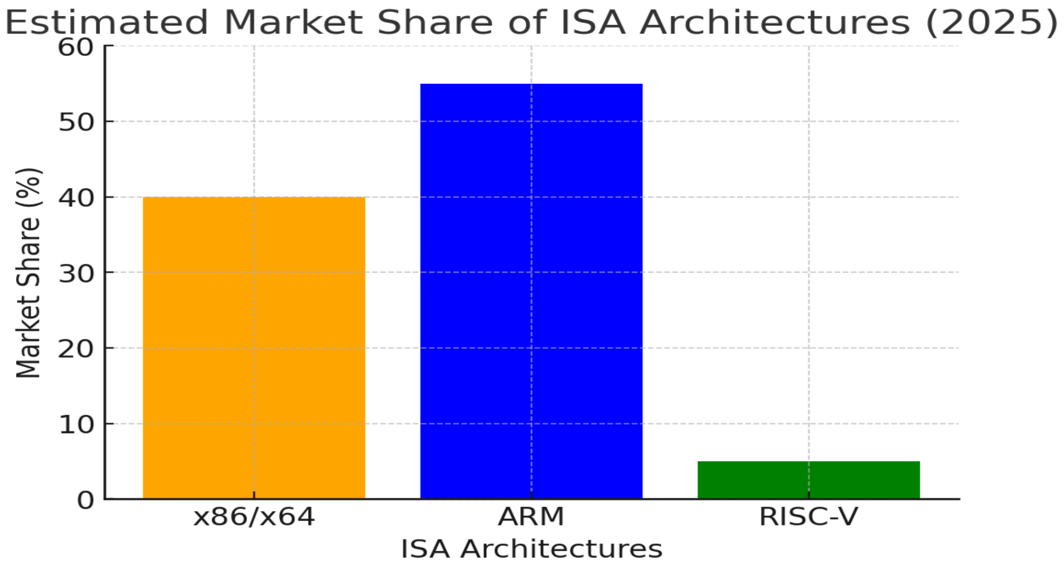
War of the CPU architectures

- The **Instruction Set Architecture** (**ISA**) specifies the set of instructions that a processor can execute, including data types, registers, addressing modes, …
  - enables software developers to write machine-level code (using *assembler* code) that the CPU can interpret and execute.
- **Complex Instruction Set Computing** (**CISC**): a large set of instructions, some of which can execute complex tasks in a single instruction.
- **Reduced Instruction Set Computing** (**RISC**): smaller set of simple instructions, designed for efficient execution. *Usually more power-efficient*.

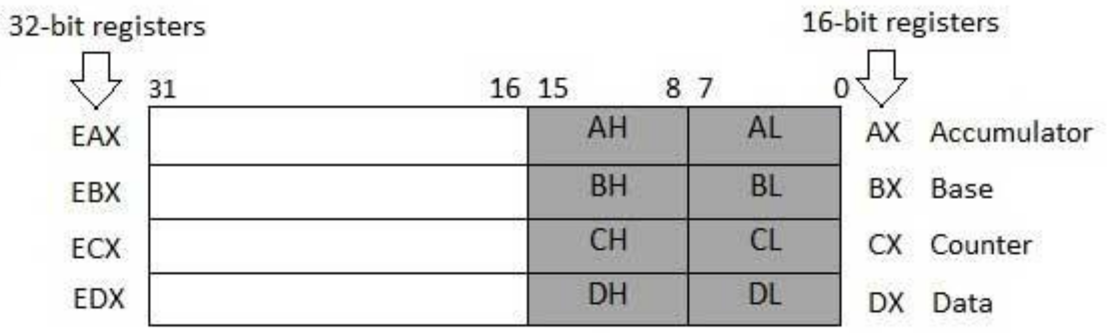
War of the CPU architectures



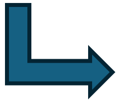




Processor Registers



- Small, quickly accessible memory location.
- Accessing main memory is much slower/costly.
- Used heavily in the Instruction Set Architecture.
- Instruction examples in **Assembler**
  - `ADD AH, BH        ; ADD the content of the BH register into the AH register`
  - `AND AH, 128       ; Perform AND operation on the variable AH and value 128`
  - `ADD AH, 10        ; ADD 10 to the register AH`
  - `MOV AL, 10        ; MOVe the value 10 to the AL register`
  - ```
    INC EDX           ; INCrement EDX register with 1
    CMP EDX, 10       ; CoMPare the EDX counter to 10
    JLE L7            ; Jump to memory address at label "L7"
                      ; if it is Less than or Equal to 10
    ```



`100101110100111010101110001011010 11...`
    ```

Instruction Set examples

x86-64 Assembly (64-bit, Linux):

```
section .data
    msg db 'Hello, PXL!', 0xA       ; end with a newline
    len equ $ - msg                 ; string length

section .text
    global _start

_start:
    ; Write the message to stdout
    mov rax, 1          ; syscall number for sys_write
    mov rdi, 1          ; file descriptor 1 is stdout
    mov rsi, msg        ; pointer to the message
    mov rdx, len        ; length of the message
    syscall             ; call kernel

    ; Exit the program
    mov rax, 60         ; syscall number for sys_exit
    xor rdi, rdi        ; exit code 0
    syscall             ; call kernel
```

```
.section .data
msg:
    .ascii "Hello, PXL!\n"  ; string to print
    len = . - msg           ; length of the string

.section .text
.global _start

_start:
    ; Write the message to stdout
    mov x0, #1          ; file descriptor 1 is stdout
    ldr x1, =msg        ; pointer to the message
    mov x2, len         ; length of the message
    mov x8, #64         ; syscall number for sys_write
    svc #0              ; call kernel

    ; Exit the program
    mov x0, #0          ; exit code 0
    mov x8, #93         ; syscall number for sys_exit
    svc #0              ; call kernel
```

ARM Assembly (64-bit, Linux)

```
.section .data
msg:
    .ascii "Hello, PXL!\n"      ; string to print
    len = . - msg               ; length of the string

.section .text
.global _start

_start:
    ; Write the message to stdout
    li a0, 1            ; file descriptor 1 is stdout
    la a1, msg          ; pointer to the message
    li a2, len          ; length of the message
    li a7, 64           ; syscall number for sys_write
    ecall               ; call kernel

    ; Exit the program
    li a0, 0            ; exit code 0
    li a7, 93           ; syscall number for sys_exit
    ecall               ; call kernel
```

RISC-V Assembly (64-bit, Linux)

```
tomc :: DESKTOP-TOMC :: 00:43:39 :: ~/asm_test
❯ nasm -f elf64 hello_x64.asm -o hello_x64.o

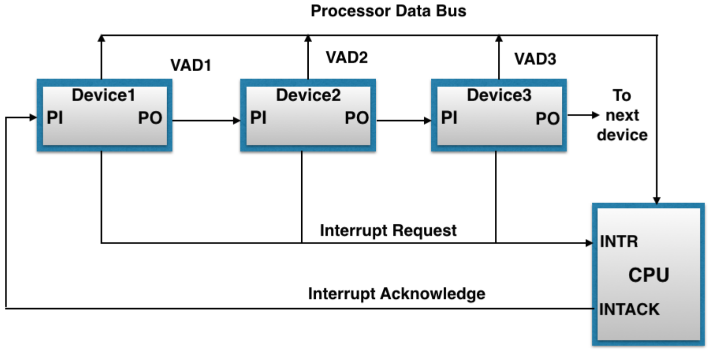
tomc :: DESKTOP-TOMC :: 00:43:49 :: ~/asm_test
❯ ld hello_x64.o -o hello_x64

tomc :: DESKTOP-TOMC :: 00:43:55 :: ~/asm_test
❯ ls -la
.rwxr-xr-x 8.9k tomc 20 Feb 00:43 hello_x64
.rwxr-xr-x  575 tomc 20 Feb 00:41 hello_x64.asm
.rw-r--r--  880 tomc 20 Feb 00:43 hello_x64.o

tomc :: DESKTOP-TOMC :: 00:43:57 :: ~/asm_test
❯ ./hello_x64
Hello, PXL!

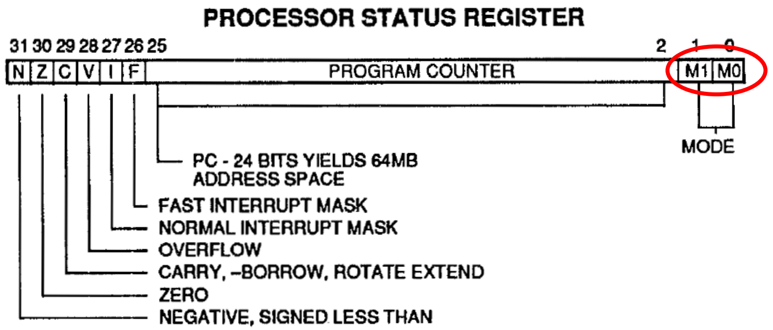
tomc :: DESKTOP-TOMC :: 00:44:01 :: ~/asm_test
❯
```

CPU Feature: Hardware Interrupts



- External devices (USB, GPU, Network Interface Card, NVME, …) send an electrical signal on a dedicated interrupt line to the CPU.
- The interrupt controller prioritizes and forwards the highest-priority interrupt.
- Handling
  - The CPU acknowledges the request and retrieves the interrupt handler address.
  - Execution state (registers, program counter) is temporarily saved, and the control jumps to the interrupt handler address.
  - After handling, the CPU restores the previous state and resumes execution.
- *more on interrupts later*

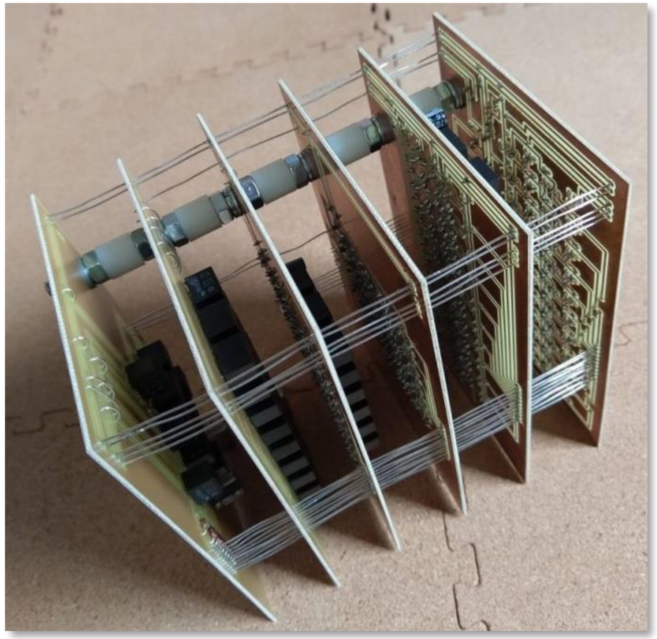
CPU features: protected FLAG



ARM-1 CPU status register

- A special bit in the CPU's control register enforces privilege levels.
- When set, the CPU operates in protected mode, restricting direct hardware access.
- Prevents user-mode programs from modifying system memory or executing privileged instructions.
- Used in architectures like x86 to separate kernel mode (Ring 0) from user mode (Ring 3). *more on this later*
- *For example, on the x86-64:*
  - *Privileged instructions like `MOV CR3, reg` and `INVLPG` control paging and memory access, while `LGDT`, `LIDT`, and `WRMSR` protect system tables and prevent unauthorized memory modifications.*

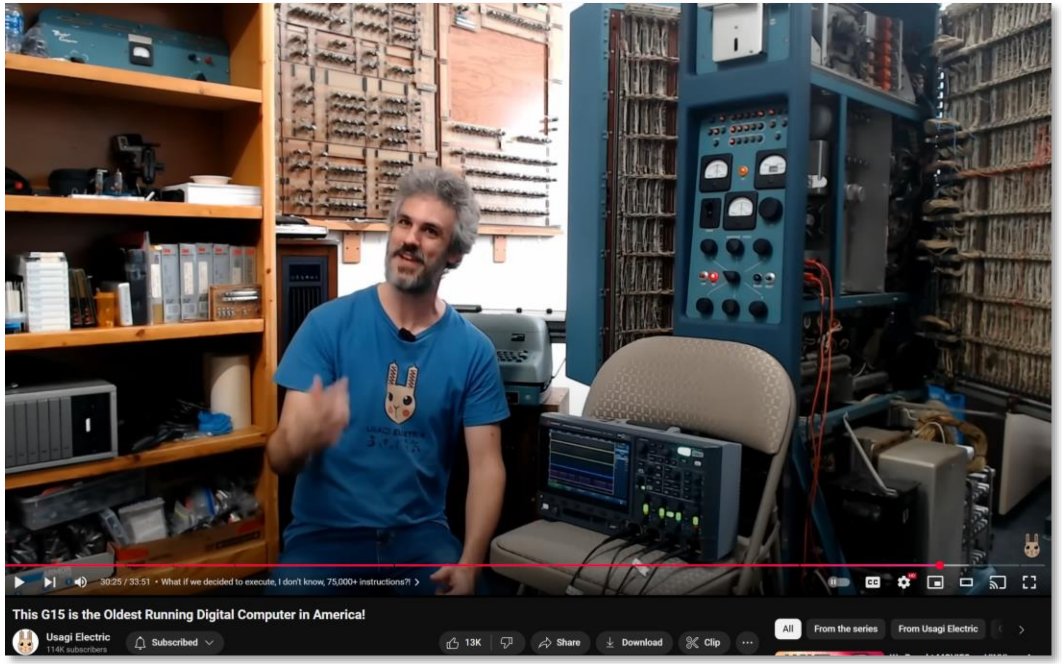
Relays real-world example



10-bit adder logic circuit built using electro-magnetic relays

Vacuum-tube computers

Bendix G-15, 1956, first "personal computer"



A **vacuum tube** is an electronic switch that controls the flow of current without moving parts.

- Essential for early computers *before transistors*
- Large, power-hungry, fragile, limited lifespan, expensive, and slower than transistors.

CPU examples

Minecraft RISC CPU

- 8 bit data, 16 bit fixed size instruction length
- 1Hz clock speed, 4 stage instruction pipeline (fetch - decode - execute - writeback)
- 64 byte automatic 8-way associative data cache and 256 bytes RAM
- Up to 256 addressable I/O ports
- 7 general purpose registers
- Over 40 ALU functions, including a hardware barrel shifter, multiplier, divider and square rooter
- 32x128 byte program pages for a total of 4KiB program storage

Advancements in semiconductor technology are the true driving force behind all IT Innovation.

- ***1947*** *- Bell Labs invents the first transistor, replacing bulky relays and vacuum tubes and enabling miniaturization.*
- ***1958*** *- Jack Kilby (Texas Instruments) and Robert Noyce (Fairchild) develop the IC, paving the way for modern electronics and microcontrollers.*
- ***1971*** *- Intel launches the 4004 for a Japanese calculator, making it possible to have an entire CPU on a single microprocessor chip.*
- ***1970's*** *- Home computers like the Commode PET, Apple II, and Tandy TRS-80 launch the rise of Personal Computers. Hundreds of 8-bit home computer models are all powered by Intel 8080/Zilog 80/MOS Technology 6502 CPU's.*

Altair 8800 home computer with 8-bit Intel 8080 CPU, 1974

Busicom 141-PF with 4-bit Intel 4004 CPU, 1971

Byte Magazine "The 1977 Trinity"

Commodore PET,
MOS Technologies 6502 8-bit CPU

Apple II,
MOS Technologies 6502 8-bit CPU

Tandy TRS-80,
Zilog-80 8-bit CPU
(Intel 8080 compatible)

Advancements in semiconductor technology are the enabler behind all IT Innovation.

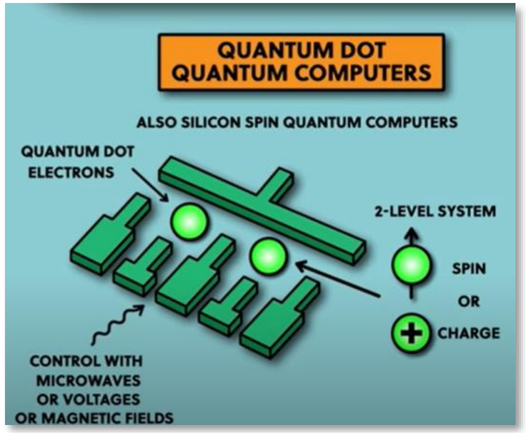
- ***1980s*** *- IBM PCs (Intel 8088, 80286) and Apple Macintosh/Commodore Amiga (Motorola 68000) revolutionize computing.*
- ***1990s*** *- ARM-based chips power mobile devices. Semiconductor improvements enable high-speed internet.*
- ***2000s*** *- Apple (A-series chips) and Qualcomm (Snapdragon) ARM CPUs push mobile computing.*
- ***2010s*** *- NVIDIA gaming GPUs drive deep learning breakthroughs. Cloud computing scales due to custom chips.*
- ***2020s*** *- Custom AI chips (NVIDIA H100, Google TPU) make generative AI breakthrough possible. Quantum computing sees significant progress using Quantum Dots with semiconductors. RISC-V gains traction.*

Apple iPhone (Internet Phone) with Samsung S5L8900 SoC, featuring a single core ARMv11 32-bit CPU, 2007

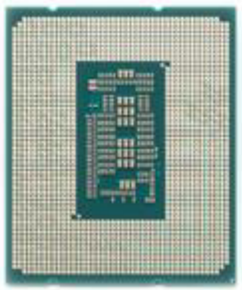
IBM PC with Intel 16-bit 8088 CPU, 1981

Commodore Amiga with a Motorola 32-bit 68000, 1985

Nintendo Switch 2 with NVIDIA Tegra T239 SoC, featuring an octa-core ARM Cortex-A78C 64-bit CPU, 2025.



Recap from a Linux perspective



- Complex logical circuits can be built from electrical switches (transistors).
- Extreme miniaturization allows for the design of complex microprocessors with advanced features that are used by modern operating systems such as Linux
  - Hardware Interrupts *(Linux interrupts and signals)*
  - Protected mode *(Linux kernel mode)*
  - Virtualization *(Linux Hypervisors)*
- Semiconductor advancements drive industry innovation.

end