Rensselaer IEEE Student Branch

Digital Design & Computer Architecture

Abdul Muizz

GBM Attendance

General Updates

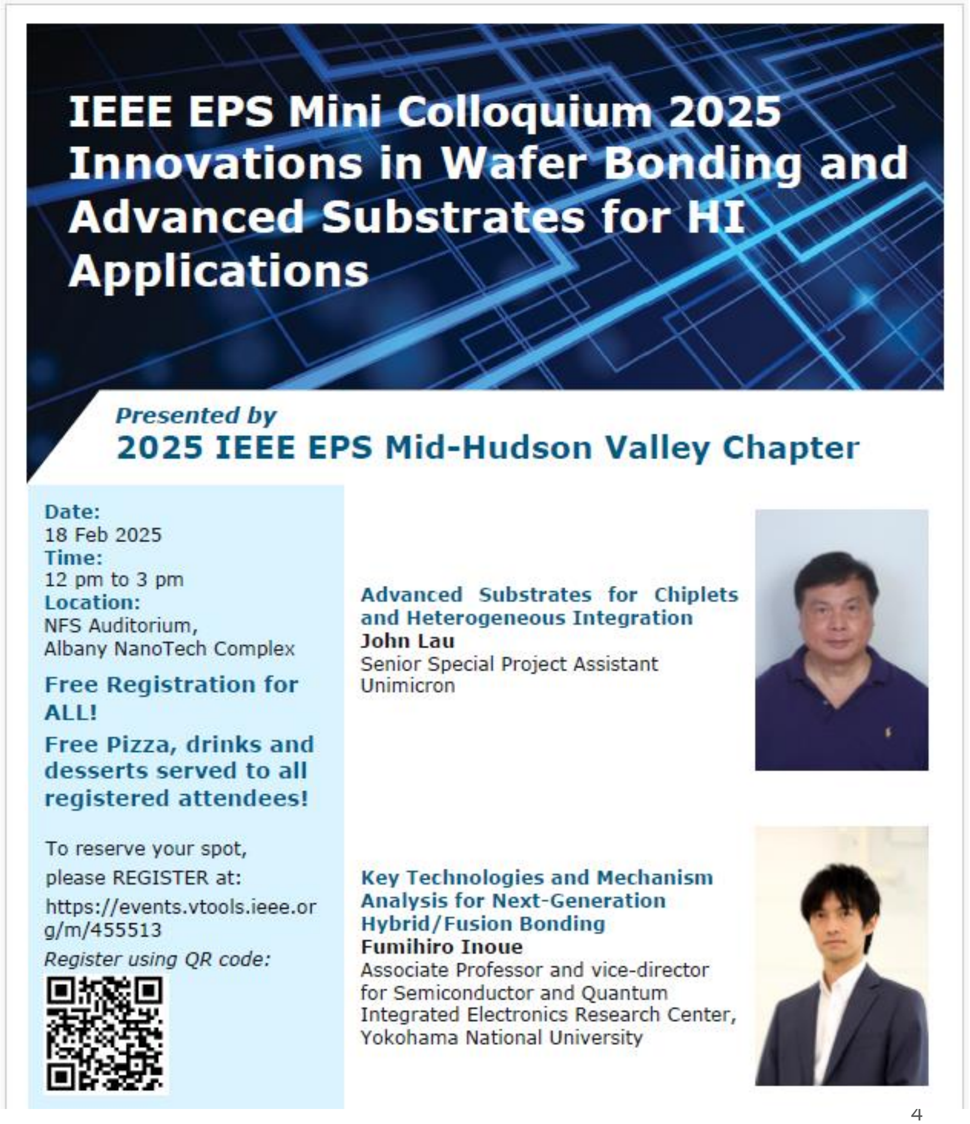
- Albany Nanotech Complex Tours
- Mini-Colloquium on Advanced Packaging and Heterogeneous Integration

Albany Nanotech Complex Tour

- Albany Nanotech Complex
  - A state-of-the-art campus that brings together industry leaders, academia and international partners to develop next-generation chips and chip fabrication processes.
  - Major Companies:
    - IBM, Applied Materials, Tokyo Electron, Wolfspeed
  - Tour Dates:
    - February 26th (Wednesday), 11:30 AM – 1:30 PM
    - March 12th (Wednesday), 11:30 AM – 1:30 PM
- More tours are coming in the future!
- **<u>Tour attendees must be IEEE EPS members!</u>**

Mini-Colloquium on Advanced Packaging and Heterogeneous Integration

- Mini-Colloquium Details
  - Located in Albany Nanotech Complex
  - Feb 18th (Tuesday) from 12pm - 3pm
  - Transportation Provided
  - Open to all students
- Schedule:
  - 12pm – Pizza Lunch
  - 1pm – Welcome/EPS Overview
  - 1:10pm – John Lau, Unimicron Technologies
  - 1:55pm – Break
  - 2:00pm – Prof. Inoue, Yokohama National University
  - 2:45pm – Closing Comments
- Please register on VTools!



IEEE EPS Mini Colloquium 2025 Innovations in Wafer Bonding and Advanced Substrates for HI Applications

Presented by
2025 IEEE EPS Mid-Hudson Valley Chapter

Date:
18 Feb 2025
Time:
12 pm to 3 pm
Location:
NFS Auditorium,
Albany NanoTech Complex

Free Registration for ALL!

Free Pizza, drinks and desserts served to all registered attendees!

To reserve your spot, please REGISTER at:
https://events.vtools.ieee.org/m/455513
Register using QR code:

Advanced Substrates for Chiplets and Heterogeneous Integration
John Lau
Senior Special Project Assistant
Unimicron

Key Technologies and Mechanism Analysis for Next-Generation Hybrid/Fusion Bonding
Fumihiro Inoue
Associate Professor and vice-director for Semiconductor and Quantum Integrated Electronics Research Center, Yokohama National University

What are we covering today? - Overview

- Most of the topics covered in EPS focus on the transistor-level and device packaging but rarely move to more abstract topics, such as computer architecture or digital logic.

- **How do processors build upon transistor-based circuits to execute complex tasks? Where is the connection between device physics and computer science?**

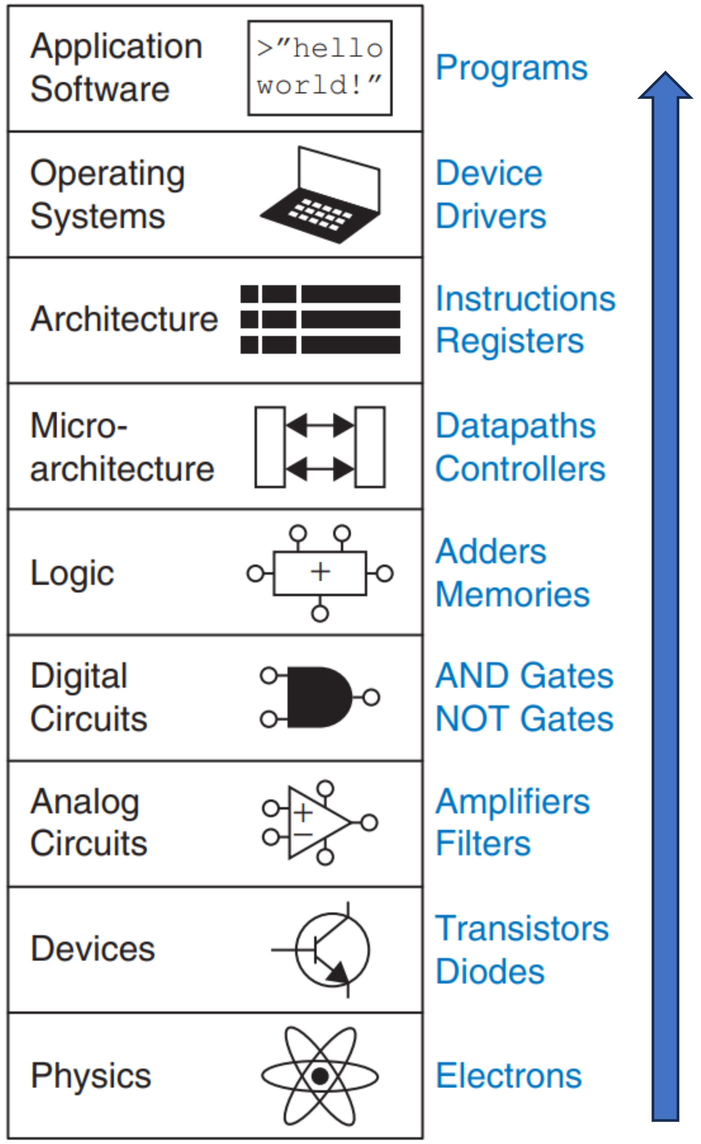
- This lecture will delve into these topics, unveiling the immense complexity of our highly-organized electronic devices.

cpu-564771_1920 Cropped.jpg
77 KB JPG

>get a rock
>melt rock
>turn rock into powder
>turn rock into a crystal
>turn it back into a rock
>inscribe ancient runes with powerful magic onto rock
>trick rock into thinking

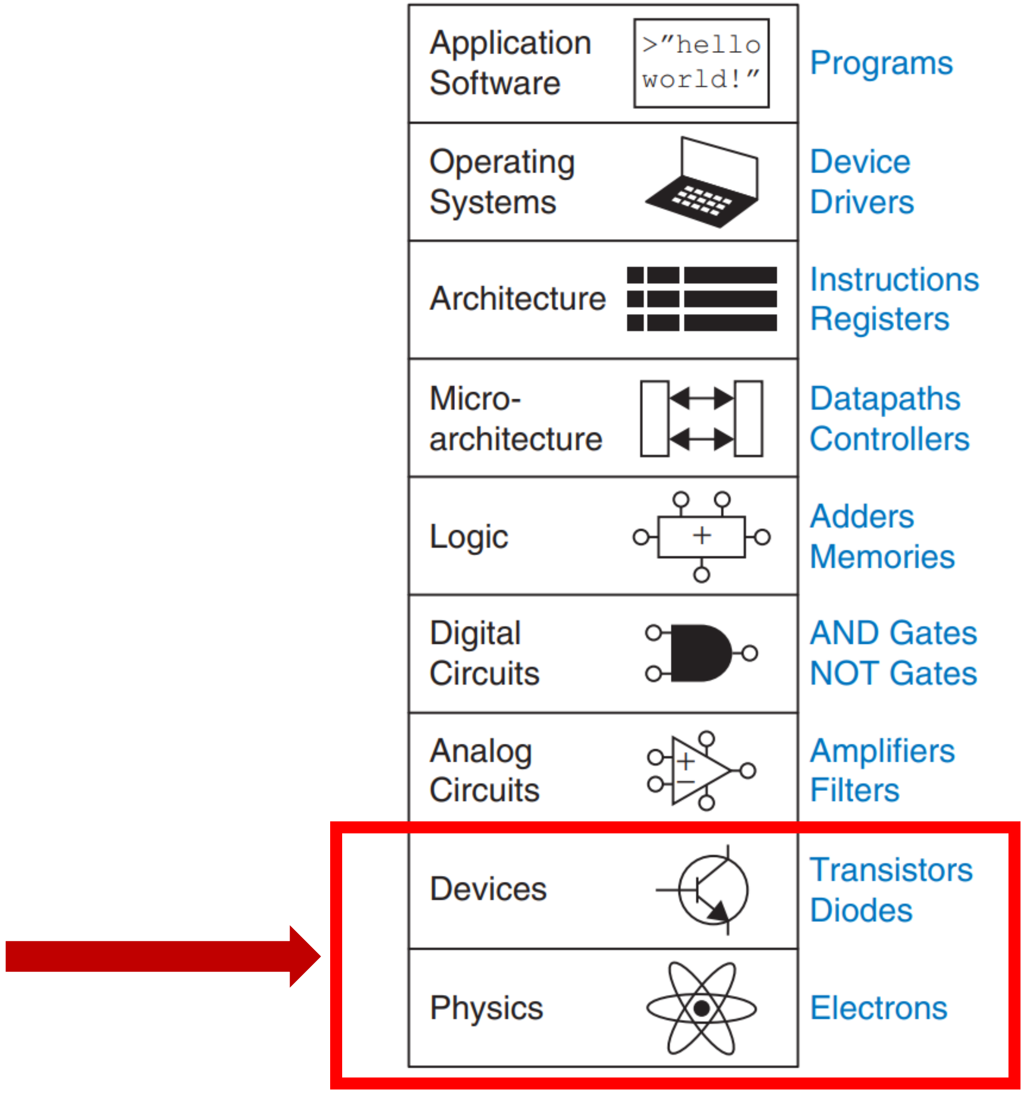
What are we covering today? - Meet the Abstraction Layers

- The computer abstraction layer diagram shows the sophisticated path it takes to get from physics to applicate software.
- Starting from the bottom, we will move up the layers of abstraction.



The higher you go, the more "abstract" each layer becomes, i.e., rooted further away from device physics.

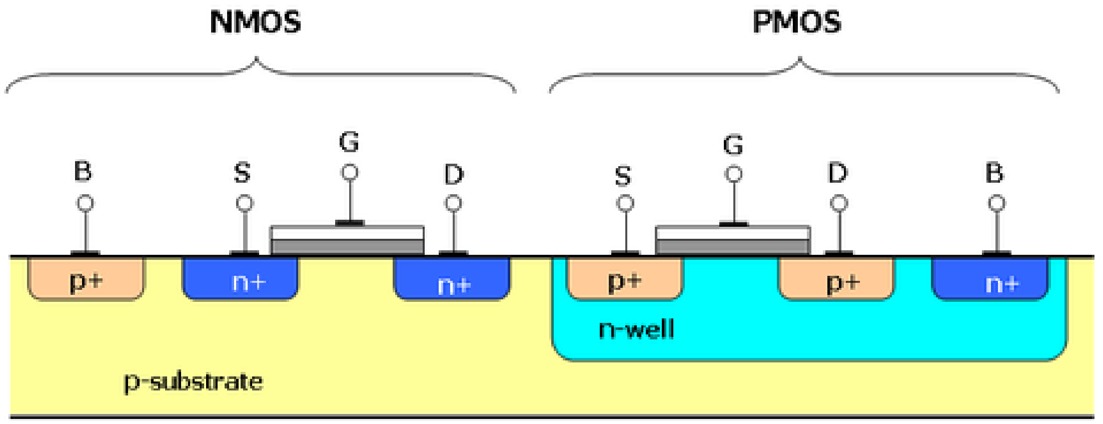
Physics & Devices

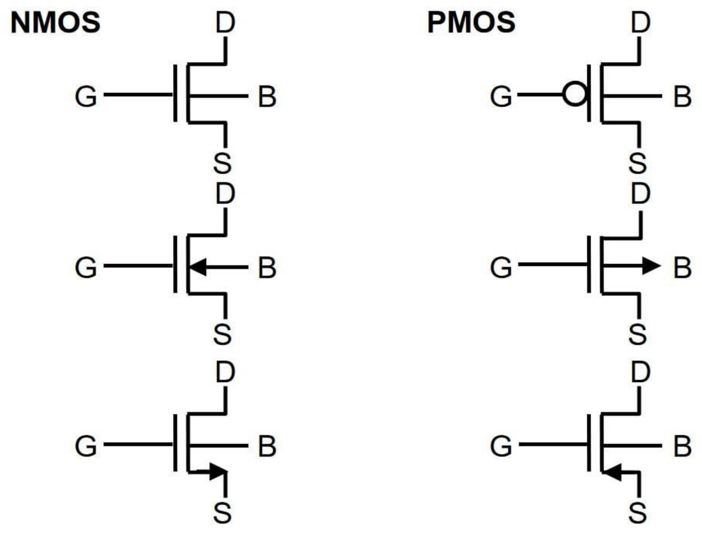


Review – CMOS Transistors

- CMOS (Complementary Metal-Oxide Semiconductor) is a type of MOSFET technology that uses a complementary pair of p-type (PMOS) and n-type (NMOS) MOSFETS.

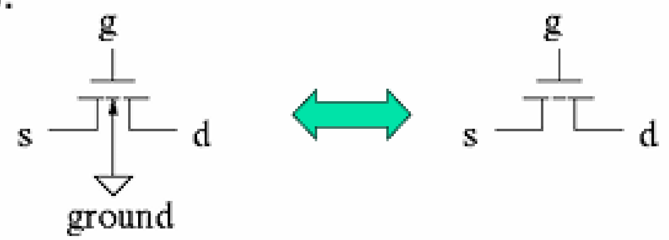
- Transistors work by regulating current (Source-Drain or Drain-Source) by using a "Gate". Based on the received input, current can be turned on or off.





NMOS and PMOS Behavior

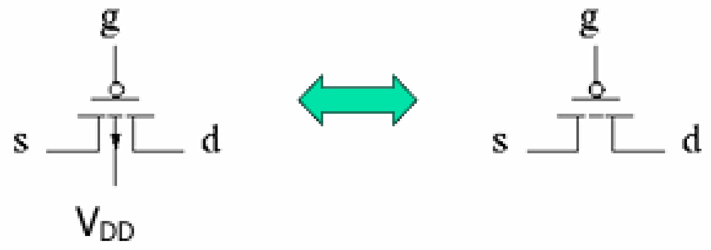
nMOS:



g	switch
0	OFF
1	ON

Logic 0 = 0V
Logic 1 = V_{DD}

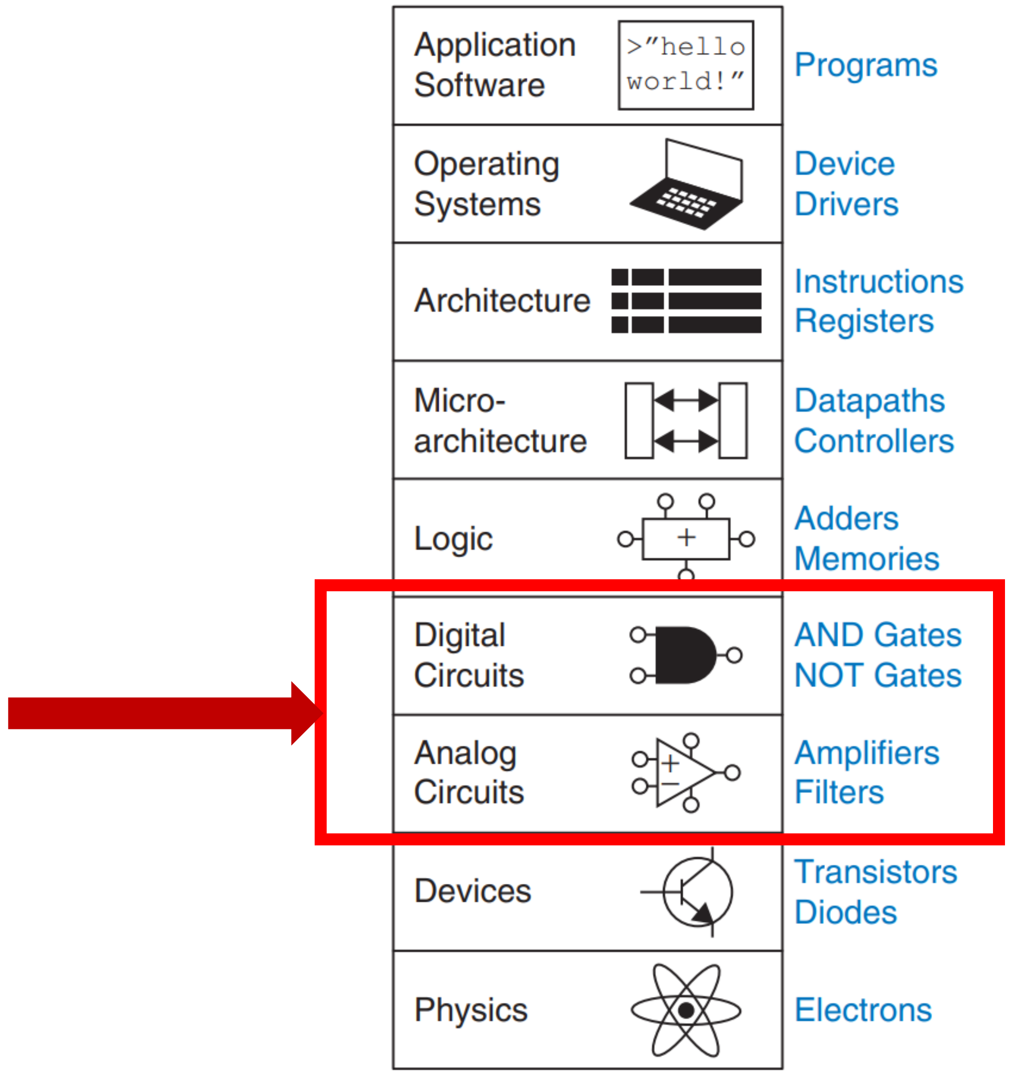
pMOS:



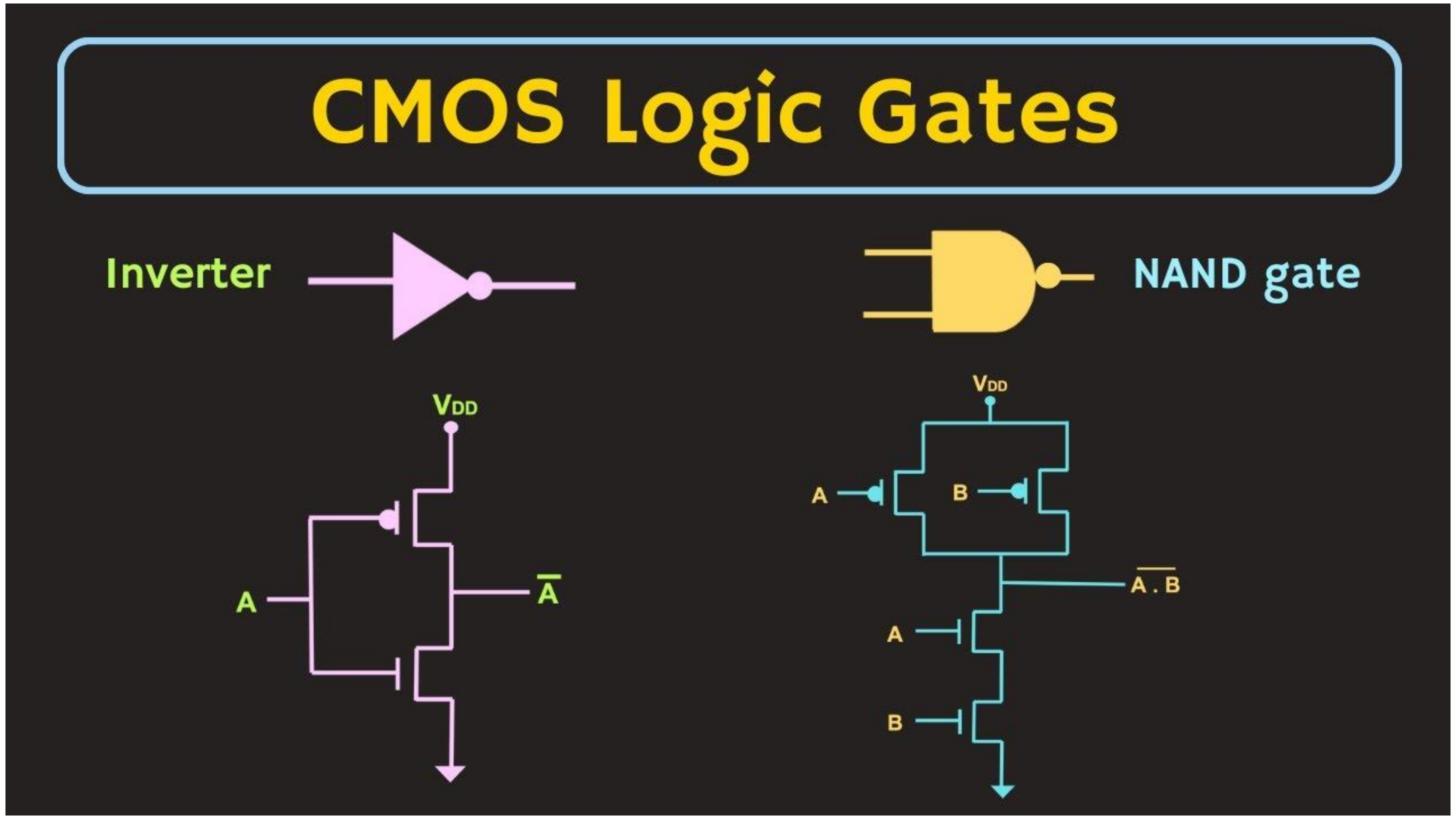
g	switch
0	ON
1	OFF

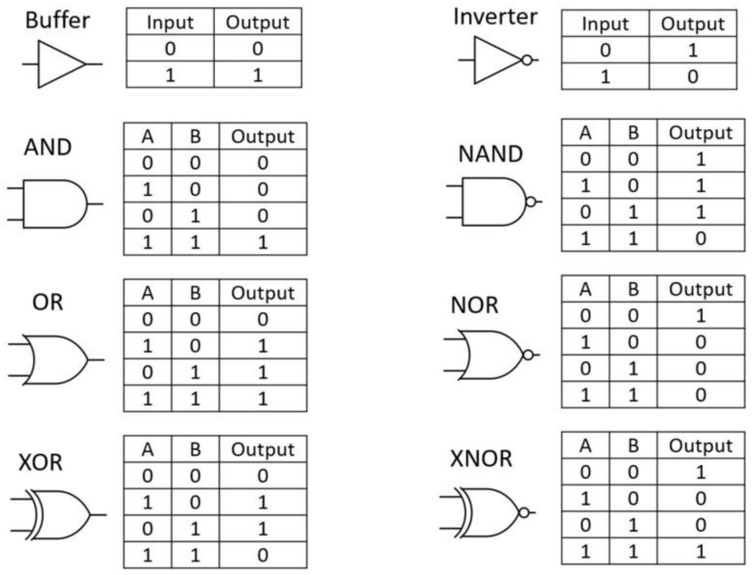
Logic 0 = 0V
Logic 1 = V_{DD}

Analog & Digital Circuits



Linked Transistors can form Digital Logic Gates




Buffer
Inverter
AND
NAND
OR
NOR
XOR
XNOR

Buffer

Input	Output
0	0
1	1

Inverter

Input	Output
0	1
1	0

AND

A	B	Output
0	0	0
1	0	0
0	1	0
1	1	1

NAND

A	B	Output
0	0	1
1	0	1
0	1	1
1	1	0

OR

A	B	Output
0	0	0
1	0	1
0	1	1
1	1	1

NOR

A	B	Output
0	0	1
1	0	0
0	1	0
1	1	0

XOR

A	B	Output
0	0	0
1	0	1
0	1	1
1	1	0

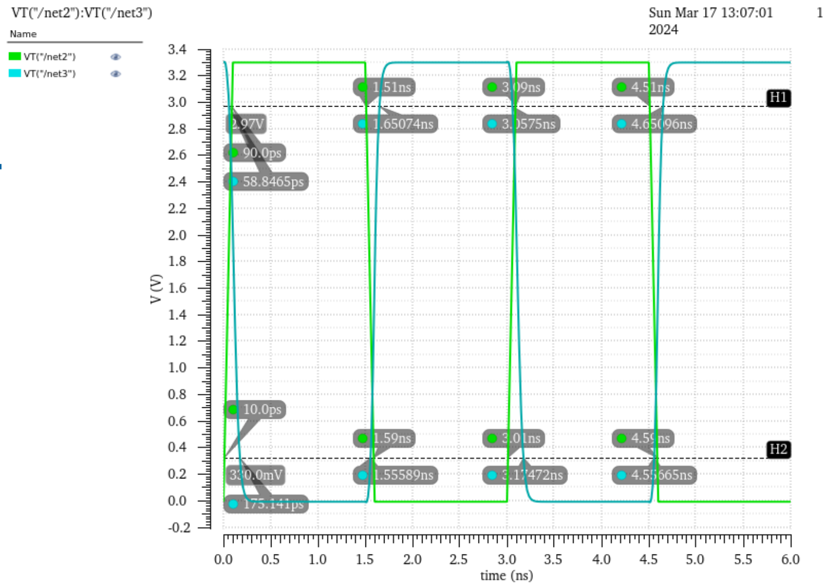
XNOR

A	B	Output
0	0	1
1	0	0
0	1	0
1	1	1

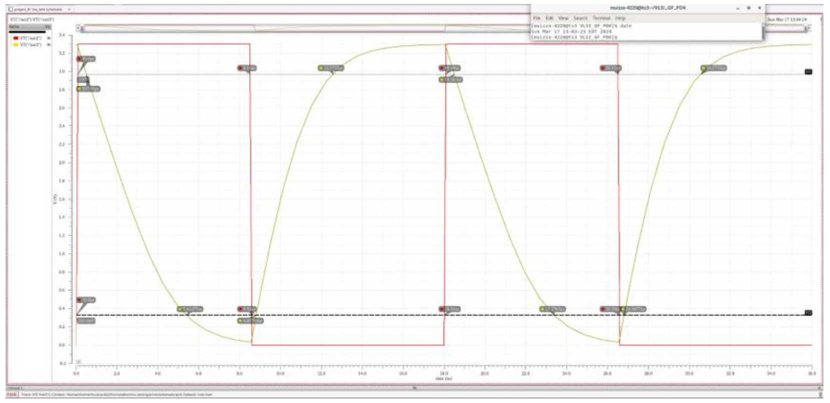
Note the Pull-Up Network (PUN) and Pull-Down Network (PDN)

Understanding the Analog in Digital

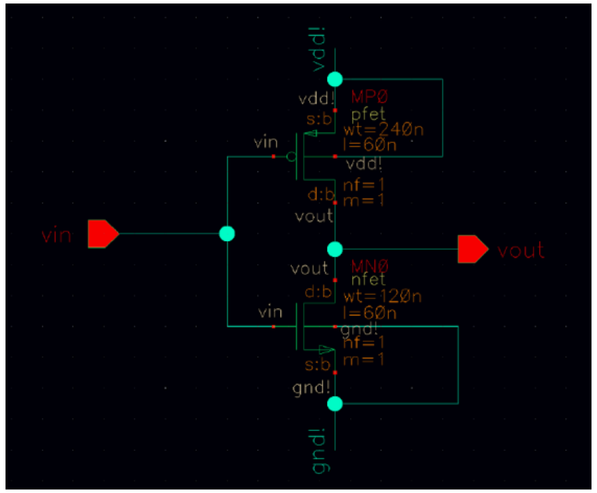
- Moving up the abstraction chart, these gates are viewed simply as digital logic blocks (1's and 0's)
- Under the hood, these are still analog circuits. Rise and fall times are influenced by parasitic capacitances and noise.
- Putting these gates together, along with registers and busses, can create function units, logic blocks, and control systems.

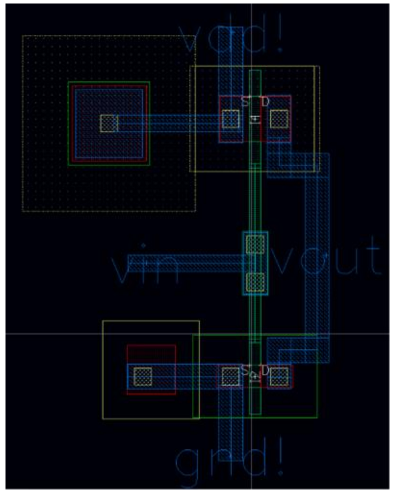


Inverter Output with 0.01 pF Load Parasitic

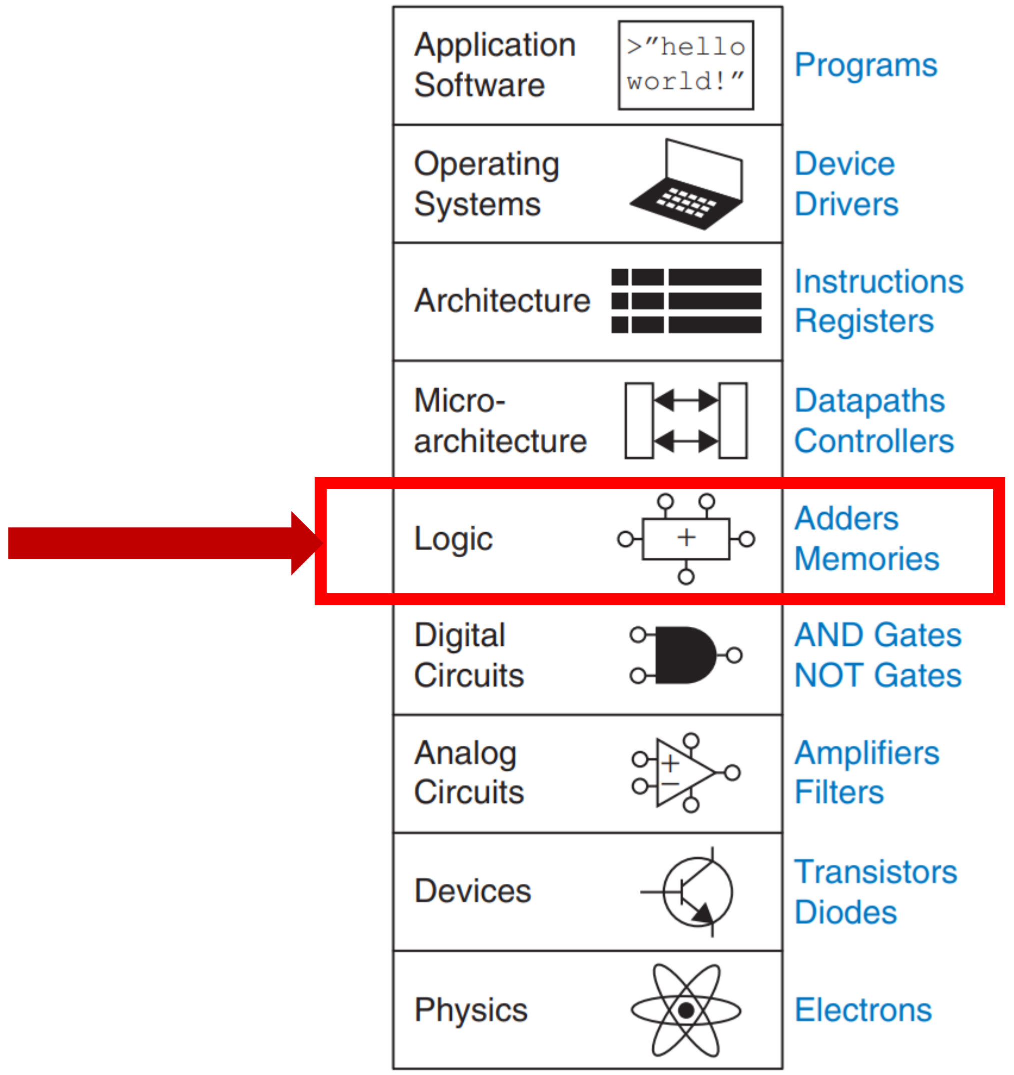


Inverter Output with 0.5 pF Load Parasitic



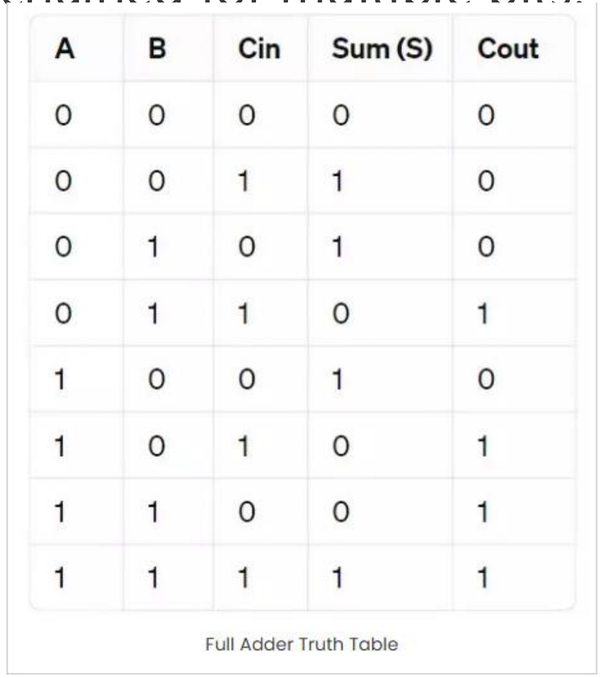


Logic


Application Software
>"hello world!"
Programs
Operating Systems
Device Drivers
Architecture
Instructions Registers
Micro-architecture
Datapaths Controllers
Logic
+
Adders Memories
Digital Circuits
AND Gates NOT Gates
Analog Circuits
+
−
Amplifiers Filters
Devices
Transistors Diodes
Physics
Electrons

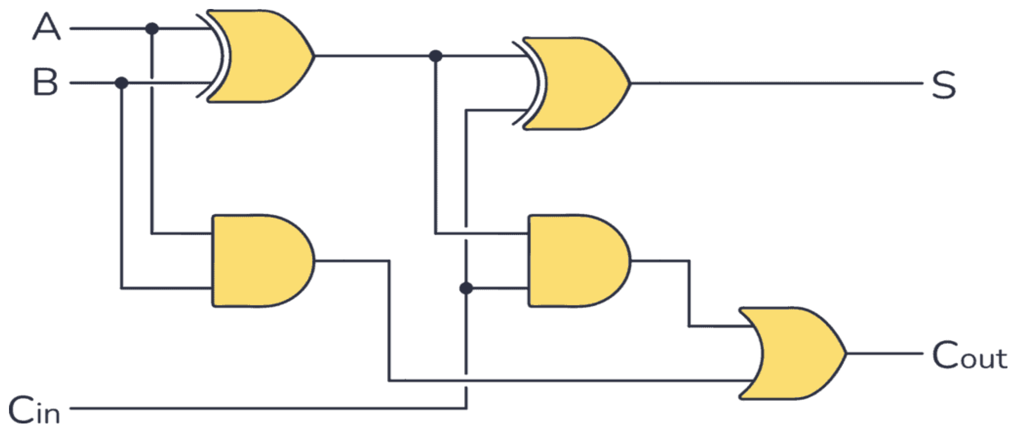
Full Adder

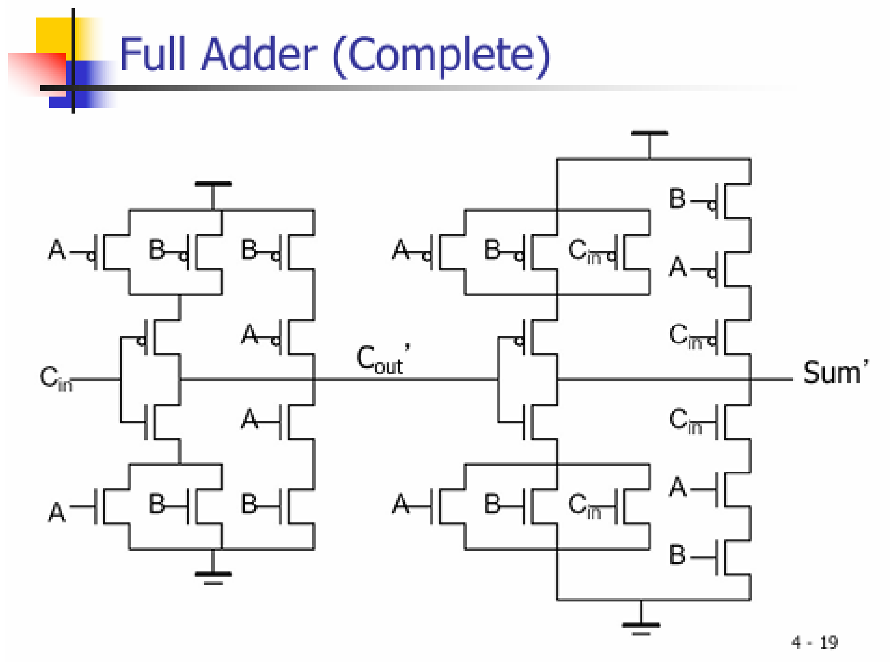
- A full adder can take three inputs, A, B, and Cin (carry-in) and produce two outputs, S and carry-out.
- Extends on from the half-adder (2 inputs, 1 output), and can be chained for multiple bits.



A	B	Cin	Sum (S)	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

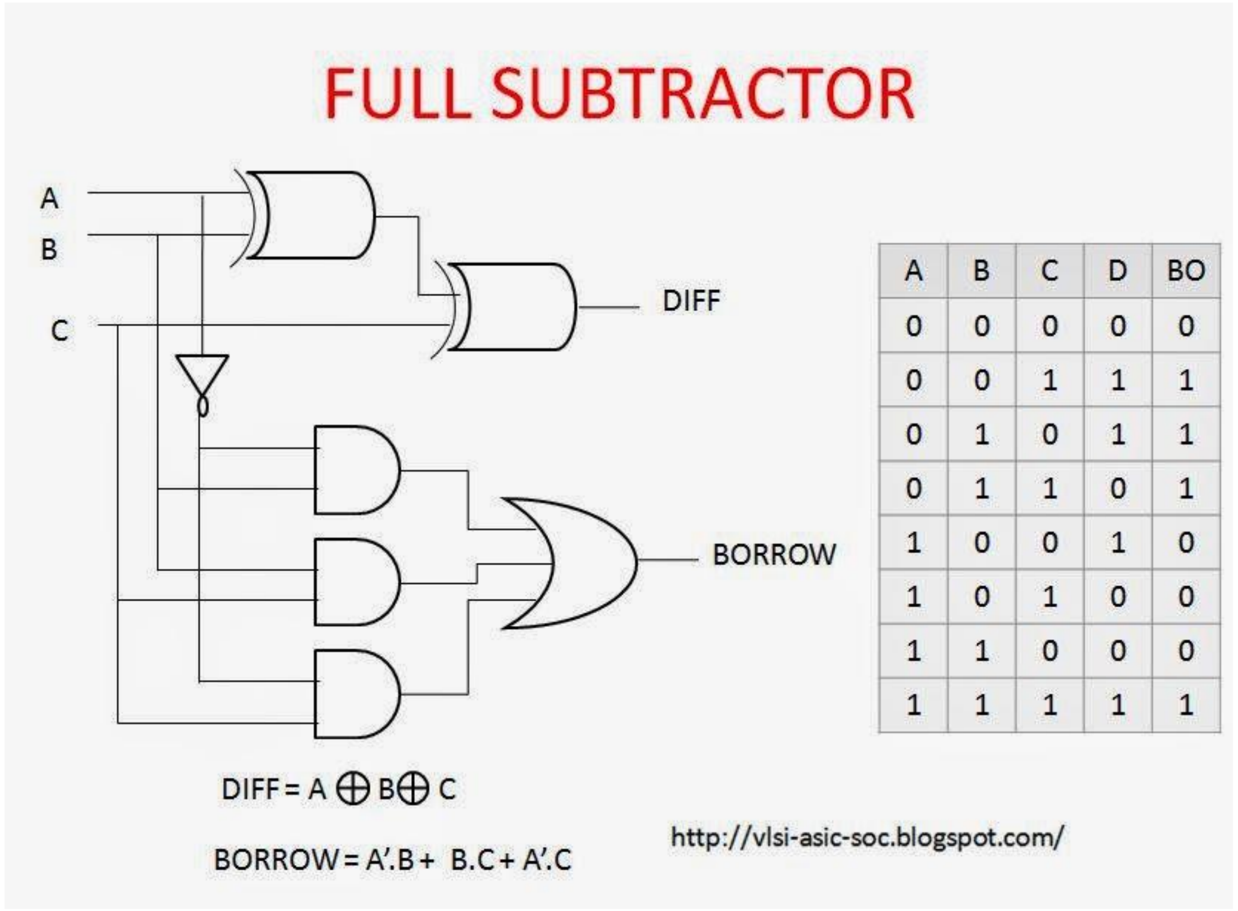
Full Adder Truth Table





Full Subtractor

- Similarly, there is a full subtractor for subtracting bit values.

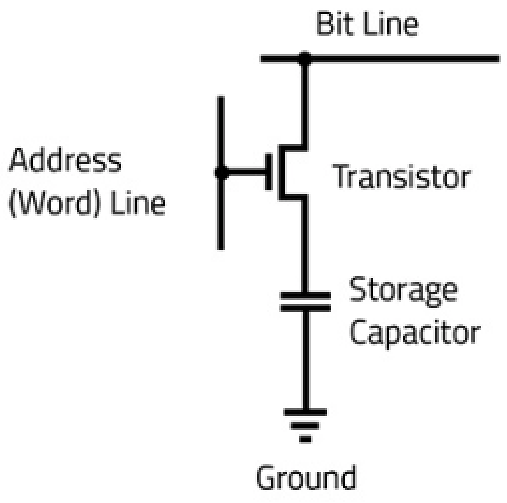


A	B	C	D	BO
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

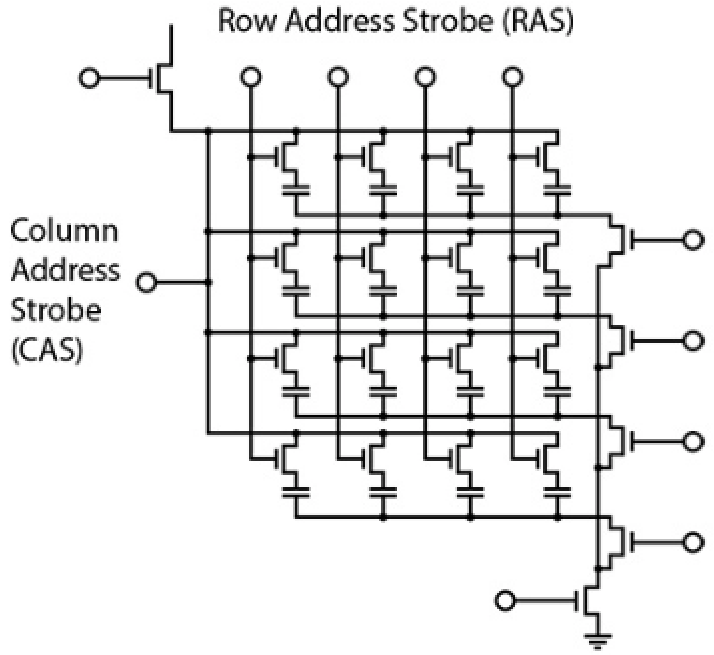
$DIFF = A \oplus B \oplus C$

$BORROW = A'.B + B.C + A'.C$

Dynamic Random Access Memory - DRAM

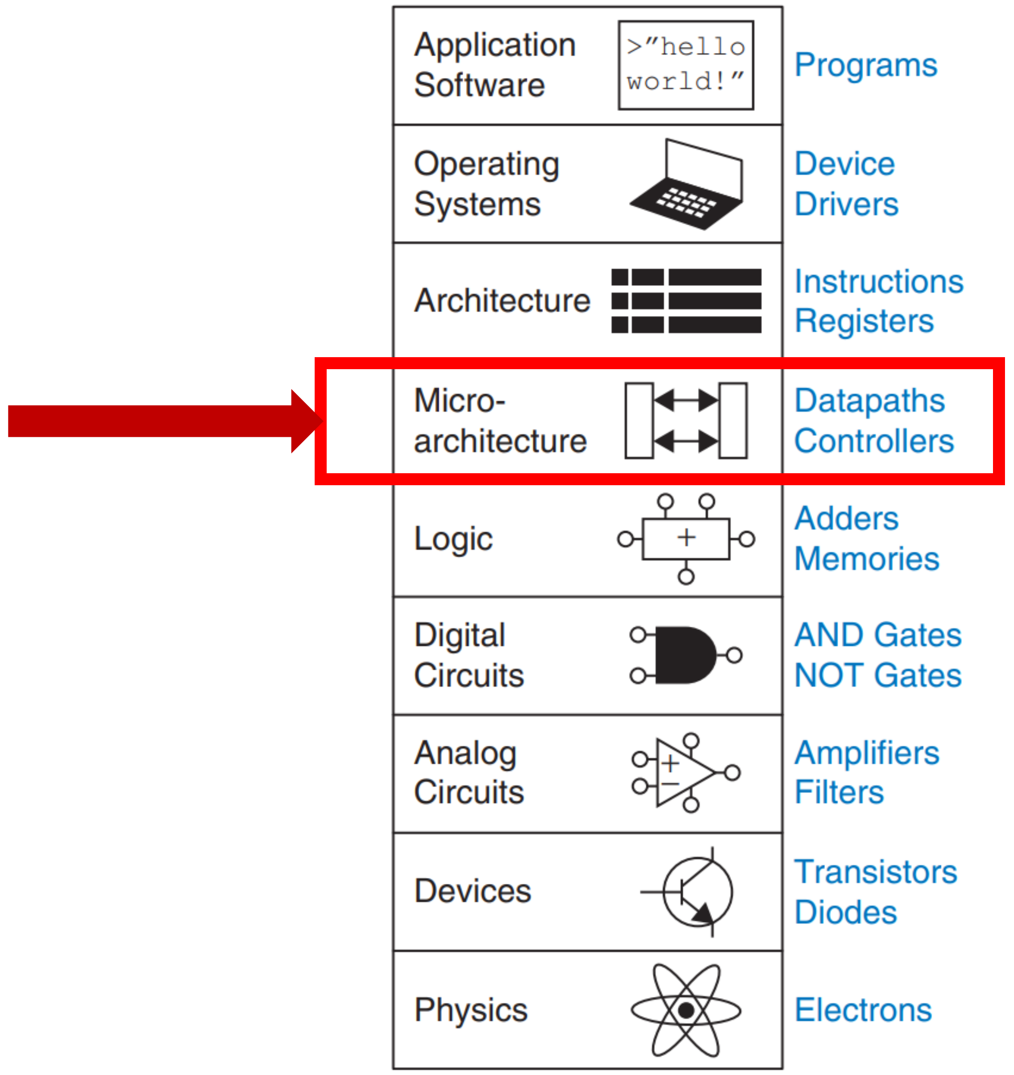


Single Memory Cell



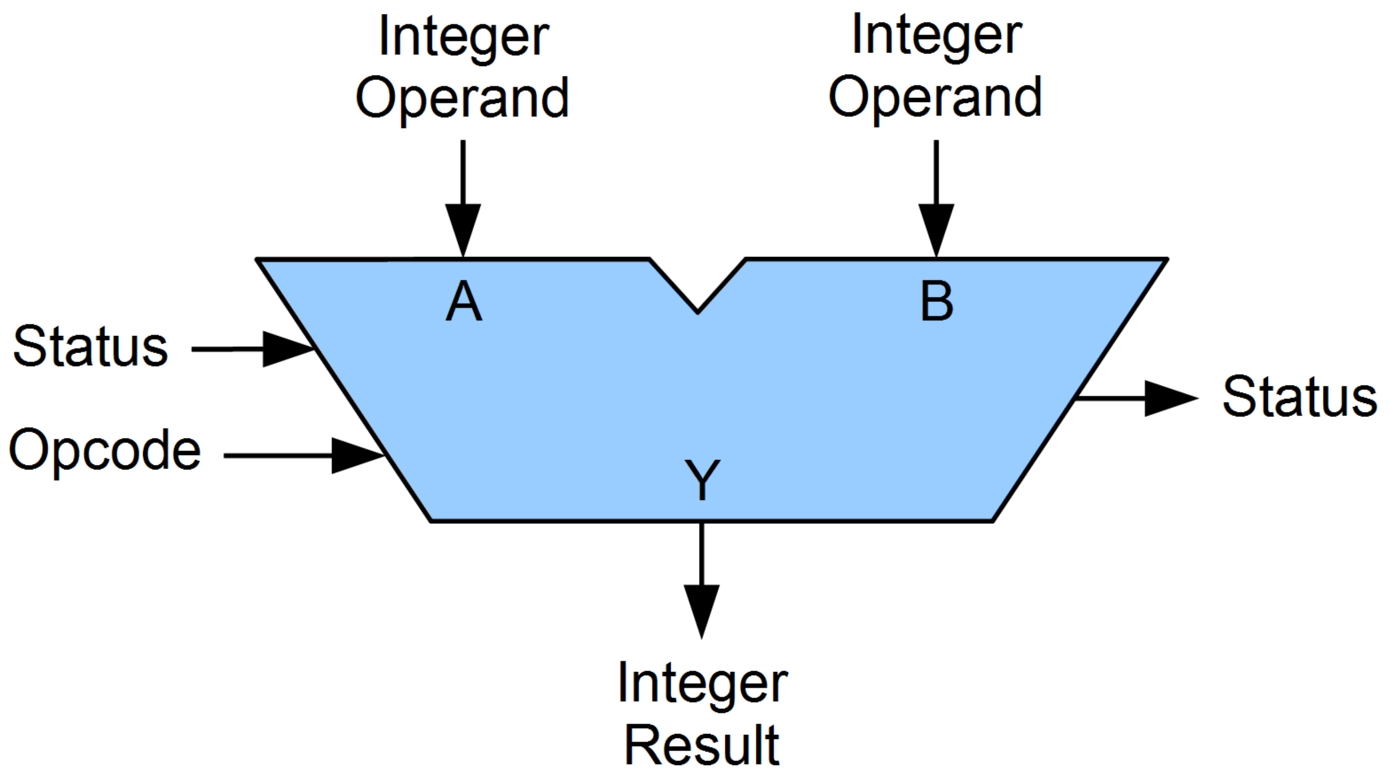
Memory Cell Array

Micro-Architecture



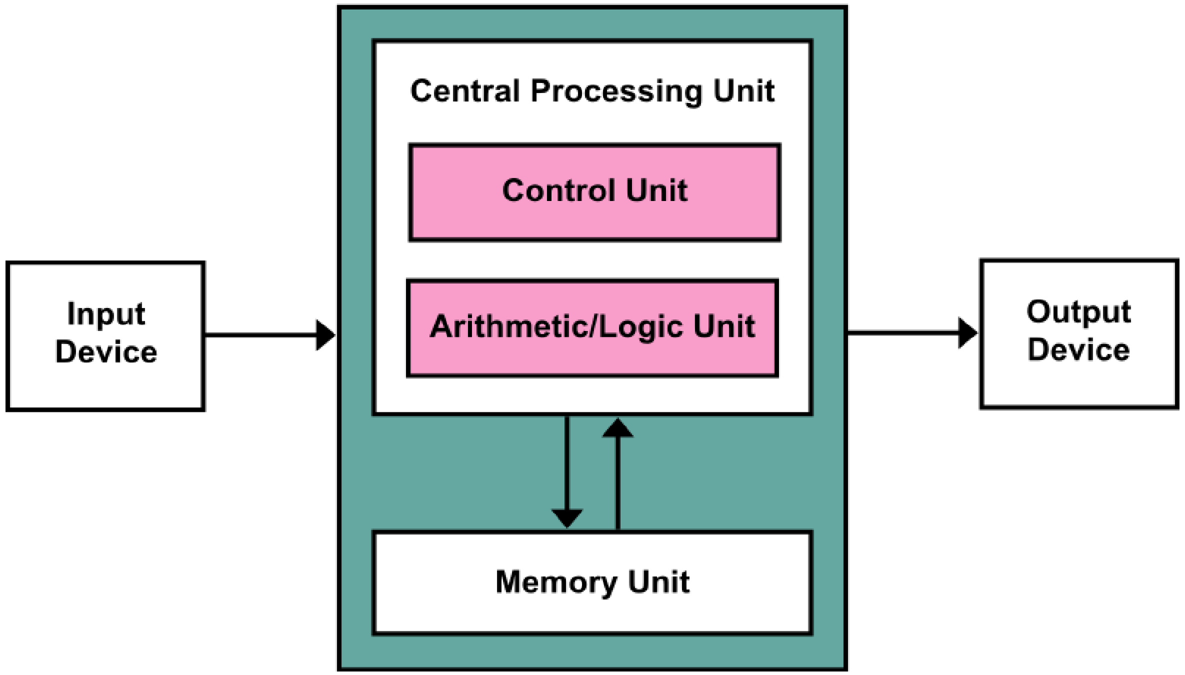
Arithmetic Logic Unit (ALU)

- By combining several arithmetic circuits (such as adders and subtractors), logic units, an accumulator, several registers, busses and control systems, you can create an ALU.
- This is the heart of a CPU and performs mathematical and logical operations.
- **How do we interface with the ALU to execute a set of instructions?**



Von Neumann Architecture

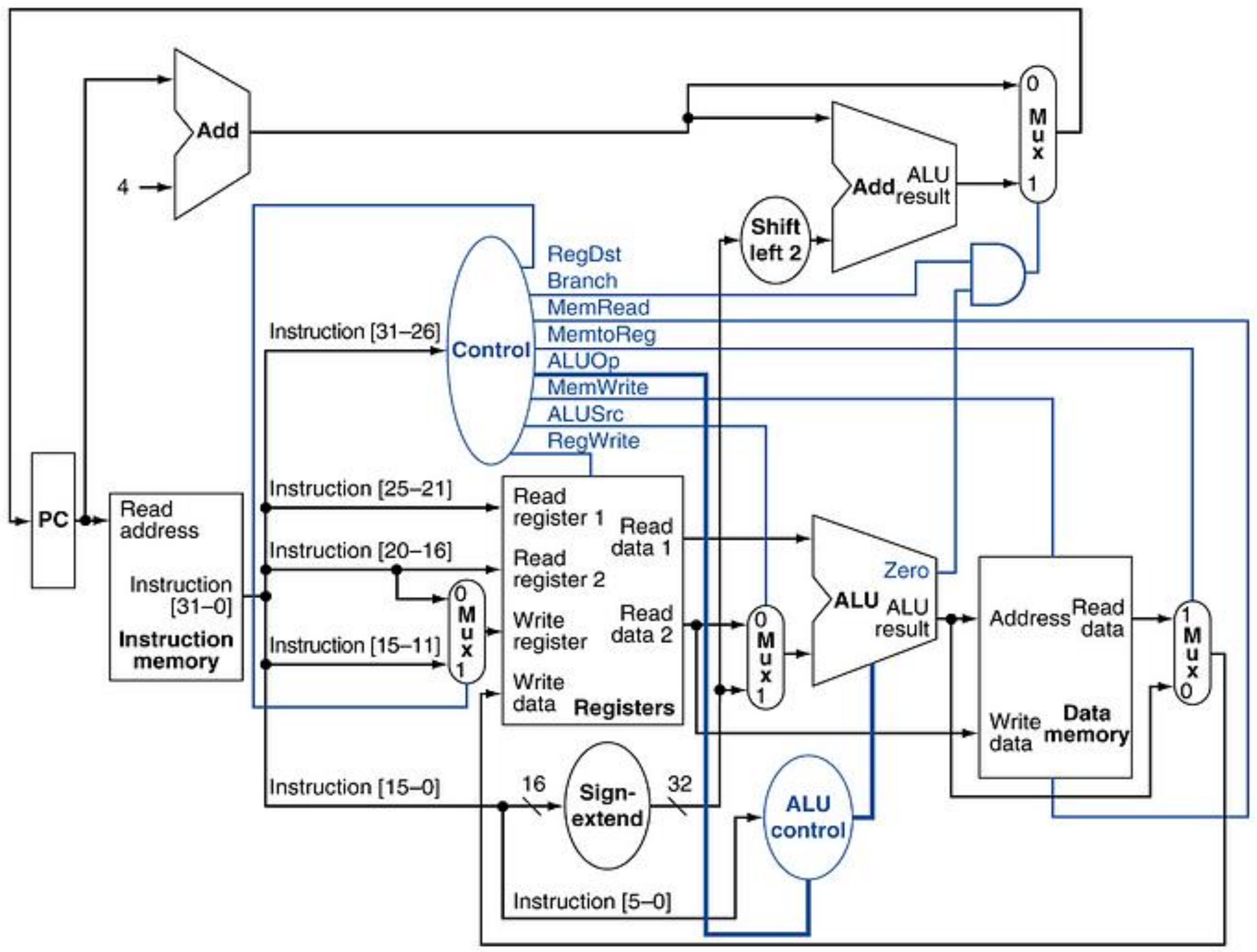
- Computer design philosophy that calls for storing instructions and data in memory and execute them sequentially.
- This design would include components, such as an arithmetic processing unit, a control unit, a memory for instructions, external mass storage, and I/O mechanisms
- Proposed in 1945 by Hungarian-American physicist and mathematician John von Neumann.



The RISC-V Single Cycle Processor – Data path

- RISC-V is an open-source instruction set architecture (ISA) that's used to design processors.

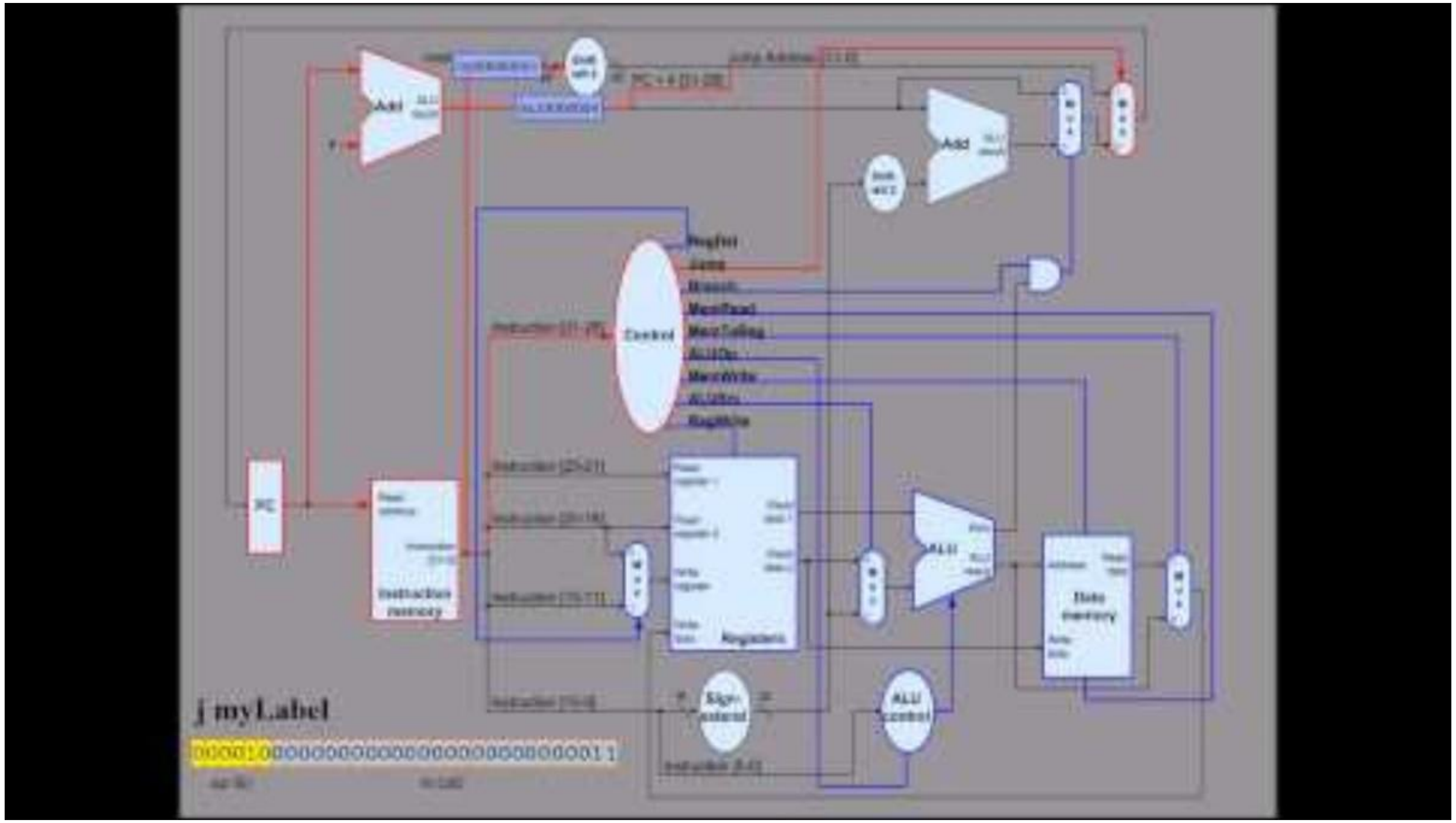
- The data path illustrates how an instruction (32-bit) can be fetched from memory, be broken down into control signals, pass through ALU and Write module to modify data, and then call the next instruction.



Instruction Types within the RISC-V ISA

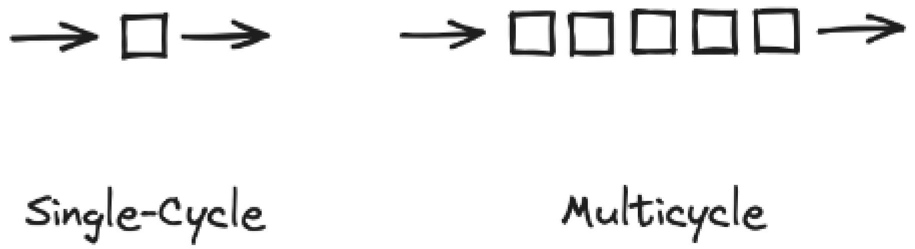
Type	Description
R-type (Register)	Perform arithmetic and logical operations that work entirely on registers (Ex: add, sub, and, or)
I-type (Immediate)	Handle operations that use an immediate (constant) value along with a register. They are also used for load instructions. (Ex: addi, lb, lw)
S-type (Store)	Used for storing data from a register into memory. (Ex: sb, sh, sw)
B-type (Branch)	Enable conditional branching based on comparison between registers. (Ex: beq, bne)
U-type (Upper Immediate)	Load a 20-bit immediate into the upper 20 bits of a register. This is useful for constructing larger constants or addresses (Ex: lui, auipc)
J-type (Jump)	Facilitate jump operations, where a larger immediate value is needed to compute a jump target relative to the current program counter (ex: jal)

Short Clip following the data path for an R-Type Instruction (5:19-10:00)



Single Cycle is slow, can we speed it up? - Multicycle

- Breaks an instruction down into multiple steps for execution.
- Useful when different stages of an instruction have different latencies.
- Beneficial for shortening the clock period and performing instructions incrementally across multiple cycles.
- Provides some level of perfromance increase, but not typically used today.

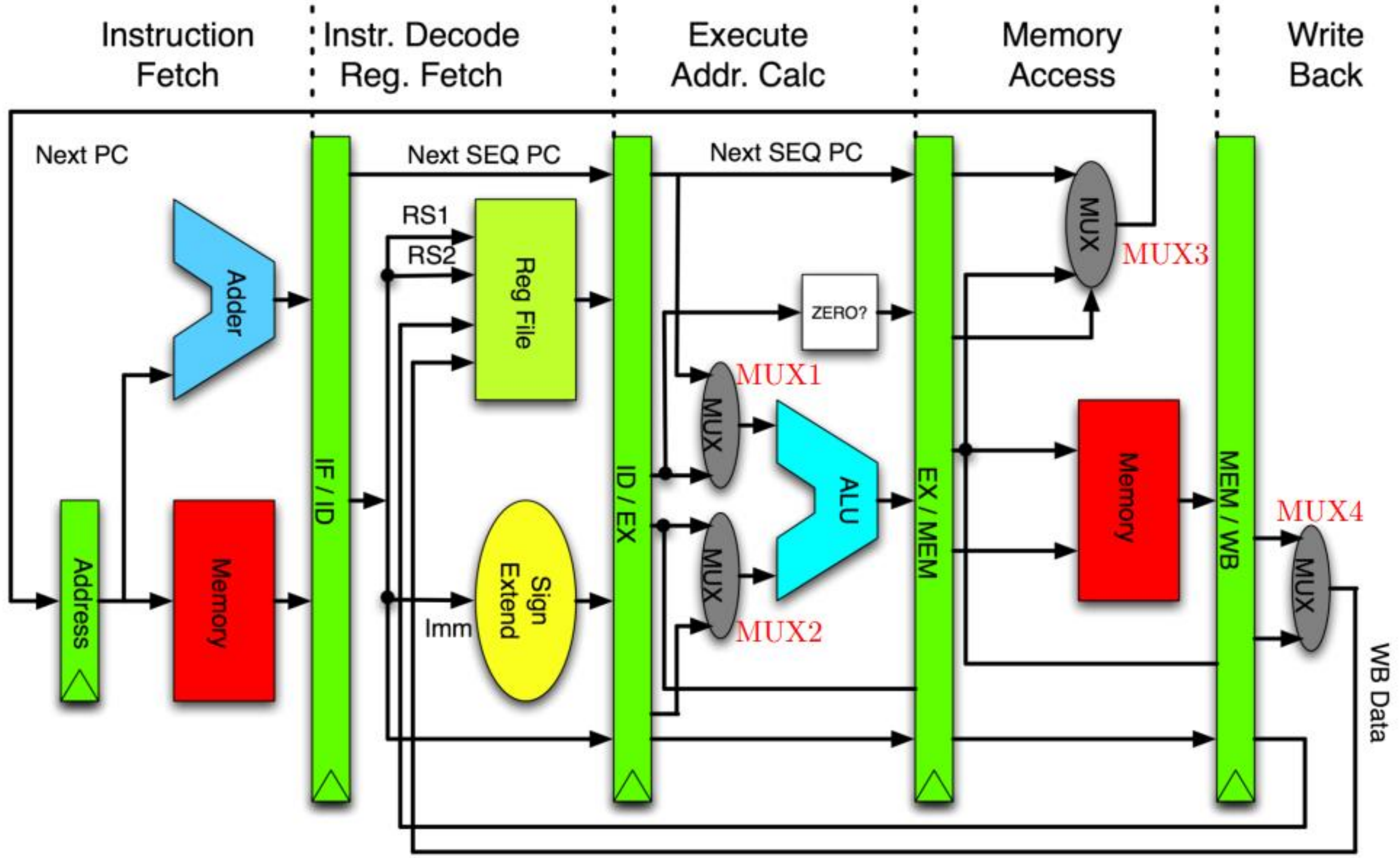


Single Cycle is slow, can we speed it up? - Pipelining

- Yes! Through pipelining!
- Laundry analogy, rather than performing all laundry steps for one load at a time (Sort, Wash, Dry, Fold), we can accelerate the process by starting the next load while the previous load is still drying.

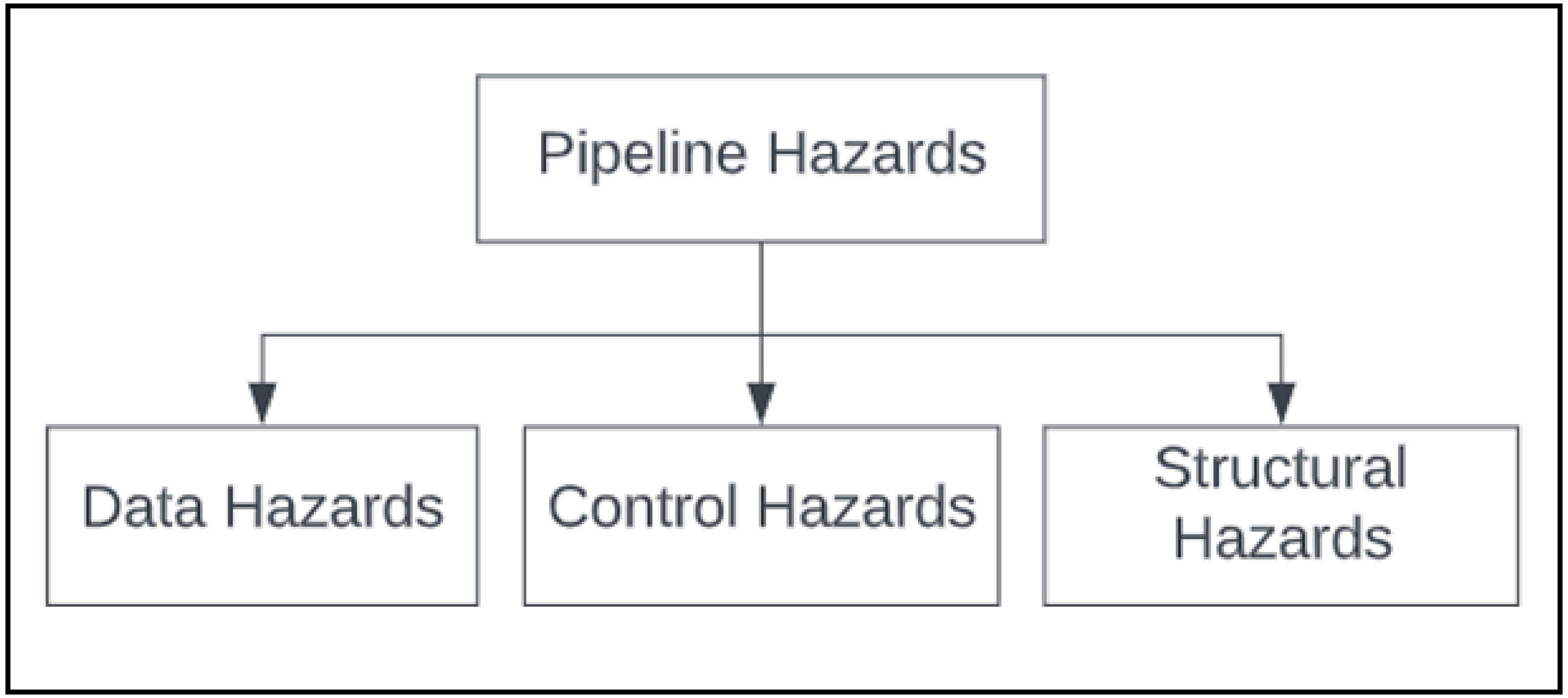
LAUNDRY TIME
STEP 1: SORT
Separate the clothes into:
Whites ~~ Light Colours ~~ Dark Colours
STEP 2: WASH
Load machine, Add soap & softener
WHITES (hot water)
LIGHT COLOURS (warm water)
DARK COLOURS (cold water)
STEP 3: HANG / DRY
STEP 4: FOLD / IRON
Sort & fold garments according to type, (eg. shirts, pants) & owner

Pipelined RISC-V Datapath



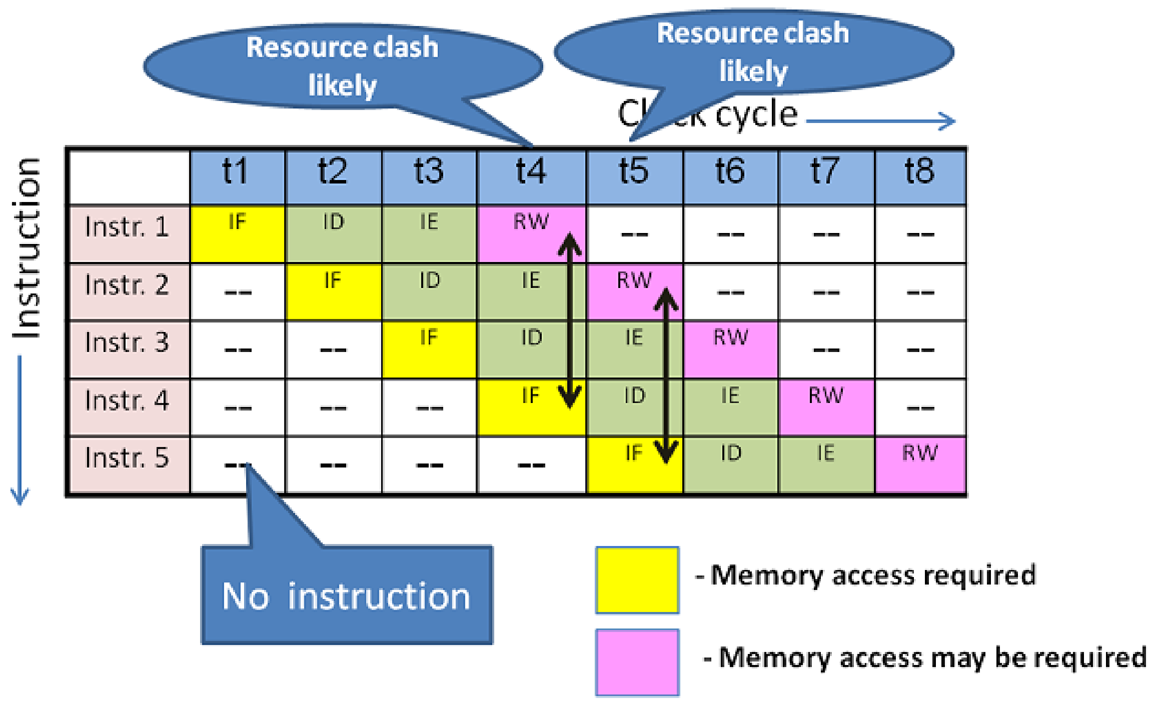
Each pipeline stall segment requires some form of control logic to prevent pipelining hazards

The 3 Hazards with Pipelining



1. Structural Hazard

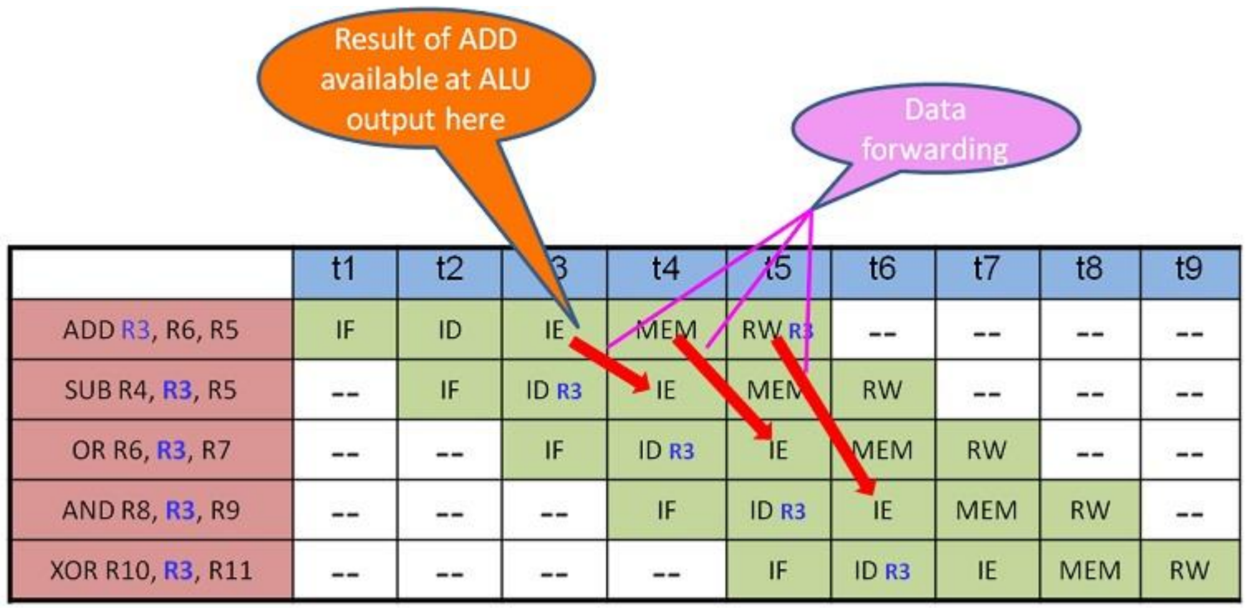
- Structural hazards – Occurs when two or more instructions in different pipeline stages simultaneously require the same hardware resource, but the resource is not available for them at the same time. Can be addressed with scheduling.
    - Ex: Fetch and Data Memory stages may need to access memory concurrently.



	t1	t2	t3	t4	t5	t6	t7	t8
Instr. 1	IF	ID	IE	RW	--	--	--	--
Instr. 2	--	IF	ID	IE	RW	--	--	--
Instr. 3	--	--	IF	ID	IE	RW	--	--
Instr. 4	--	--	--	IF	ID	IE	RW	--
Instr. 5	--	--	--	--	IF	ID	IE	RW

2. Data Hazard

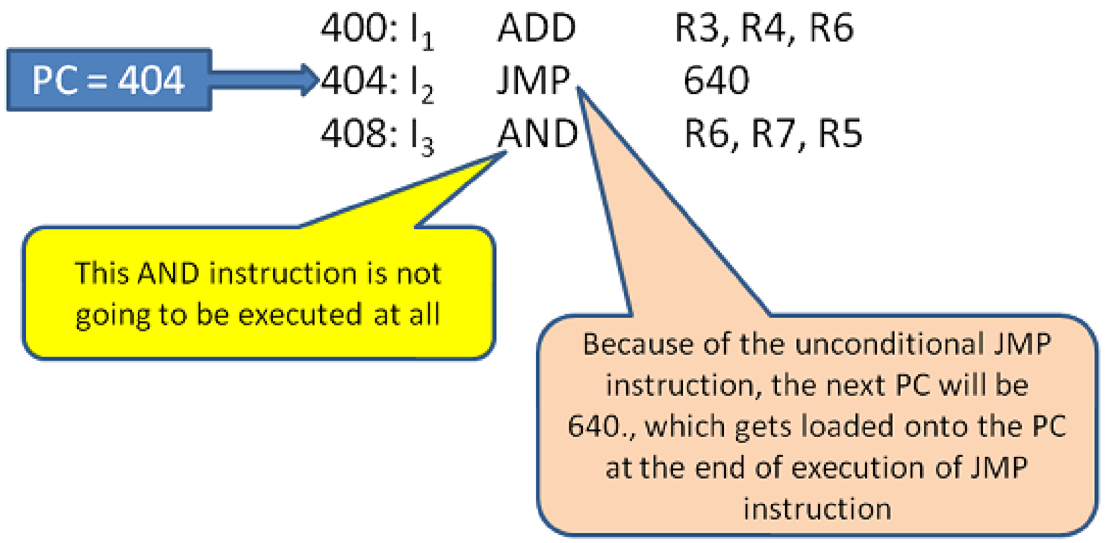
- Data hazards – Occur when an instruction depends on the result of the previous instruction that has not yet completed its execution in the pipeline. Can be addressed with stalling.
  - Ex: Read after write, write after read, or write after write.



	t1	t2	t3	t4	t5	t6	t7	t8	t9
ADD R3, R6, R5	IF	ID	IE	MEM	RW R3	--	--	--	--
SUB R4, R3, R5	--	IF	ID R3	IE	MEM	RW	--	--	--
OR R6, R3, R7	--	--	IF	ID R3	IE	MEM	RW	--	--
AND R8, R3, R9	--	--	--	IF	ID R3	IE	MEM	RW	--
XOR R10, R3, R11	--	--	--	--	IF	ID R3	IE	MEM	RW

3. Control Hazard (aka Branch Hazard)

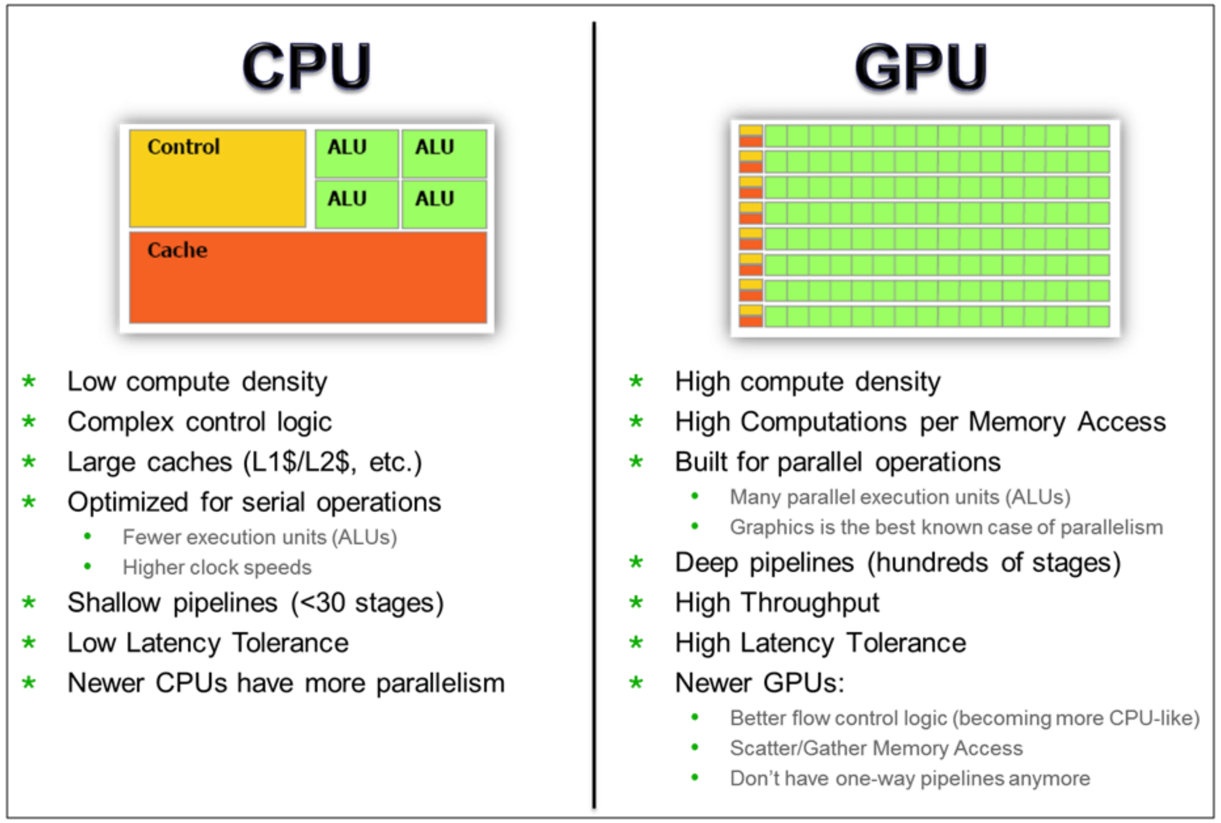
- Control Hazards – aka Branch Hazards, occur when the pipeline makes wrong assumptions about the path of a branch or jump instruction. Until the branch outcome is determined, the pipeline may have already fetched incorrect instructions. Can be addressed by discarding whatever is in the pipeline (flushing).
  - Ex: When reaching a conditional branch (if, for, while), a wrong prediction could lead to incorrect execution.


PC = 404
400: I_1 ADD R3, R4, R6
404: I_2 JMP 640
408: I_3 AND R6, R7, R5
This AND instruction is not going to be executed at all
Because of the unconditional JMP instruction, the next PC will be 640., which gets loaded onto the PC at the end of execution of JMP instruction

Why are GPUs better at Pipelining than CPUs?

- GPUs have
  - many more cores and threads,
  - can split instructions onto multiple threads,
  - have simplified execution units and specialized pipelines,
  - are optimized for throughput over latency.

Other ways to improve performance outside of pipelining include out-of-order execution, forwarding, and branch prediction.



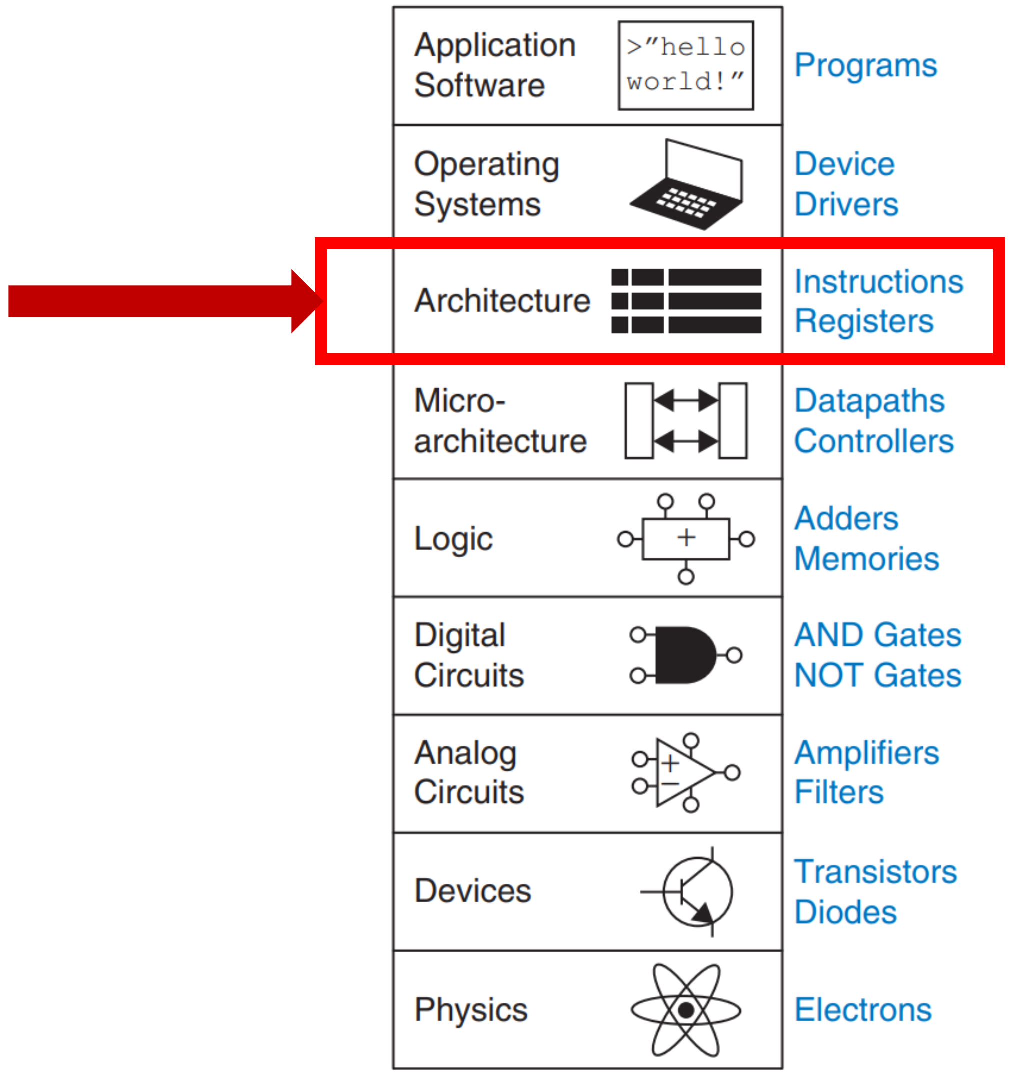
CPU

* Low compute density
* Complex control logic
* Large caches (L1$/L2$, etc.)
* Optimized for serial operations
  - Fewer execution units (ALUs)
  - Higher clock speeds
* Shallow pipelines (<30 stages)
* Low Latency Tolerance
* Newer CPUs have more parallelism

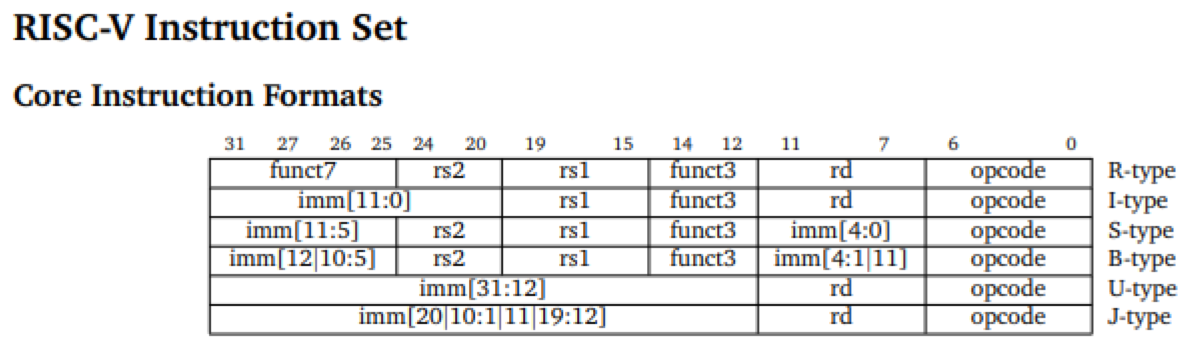
GPU

* High compute density
* High Computations per Memory Access
* Built for parallel operations
  - Many parallel execution units (ALUs)
  - Graphics is the best known case of parallelism
* Deep pipelines (hundreds of stages)
* High Throughput
* High Latency Tolerance
* Newer GPUs:
  - Better flow control logic (becoming more CPU-like)
  - Scatter/Gather Memory Access
  - Don't have one-way pipelines anymore

Architecture



The RISC-V ISA Reference Card



RISC-V Instruction Set

Core Instruction Formats

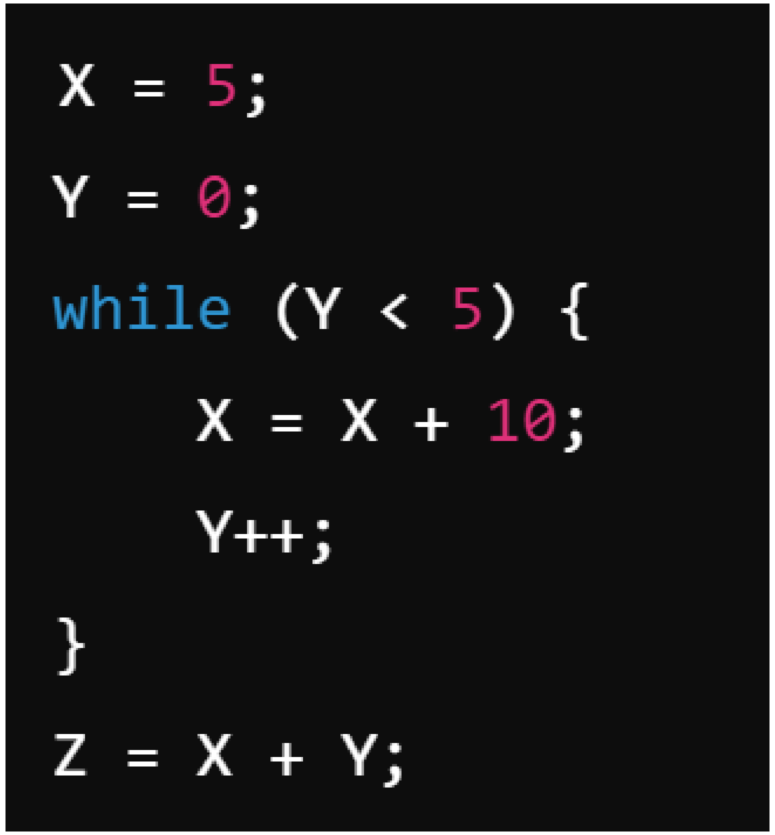
31 27 26 25	24 20	19 15	14 12	11 7	6 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]		rs1	funct3	rd	opcode	I-type
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]				rd	opcode	U-type
imm[20|10:1|11|19:12]				rd	opcode	J-type

https://www.cs.sfu.ca/~ashriram/Courses/CS295/assets/notebooks/RISCV/RISCV_CARD.pdf <- Full reference card for RISC-V ISA

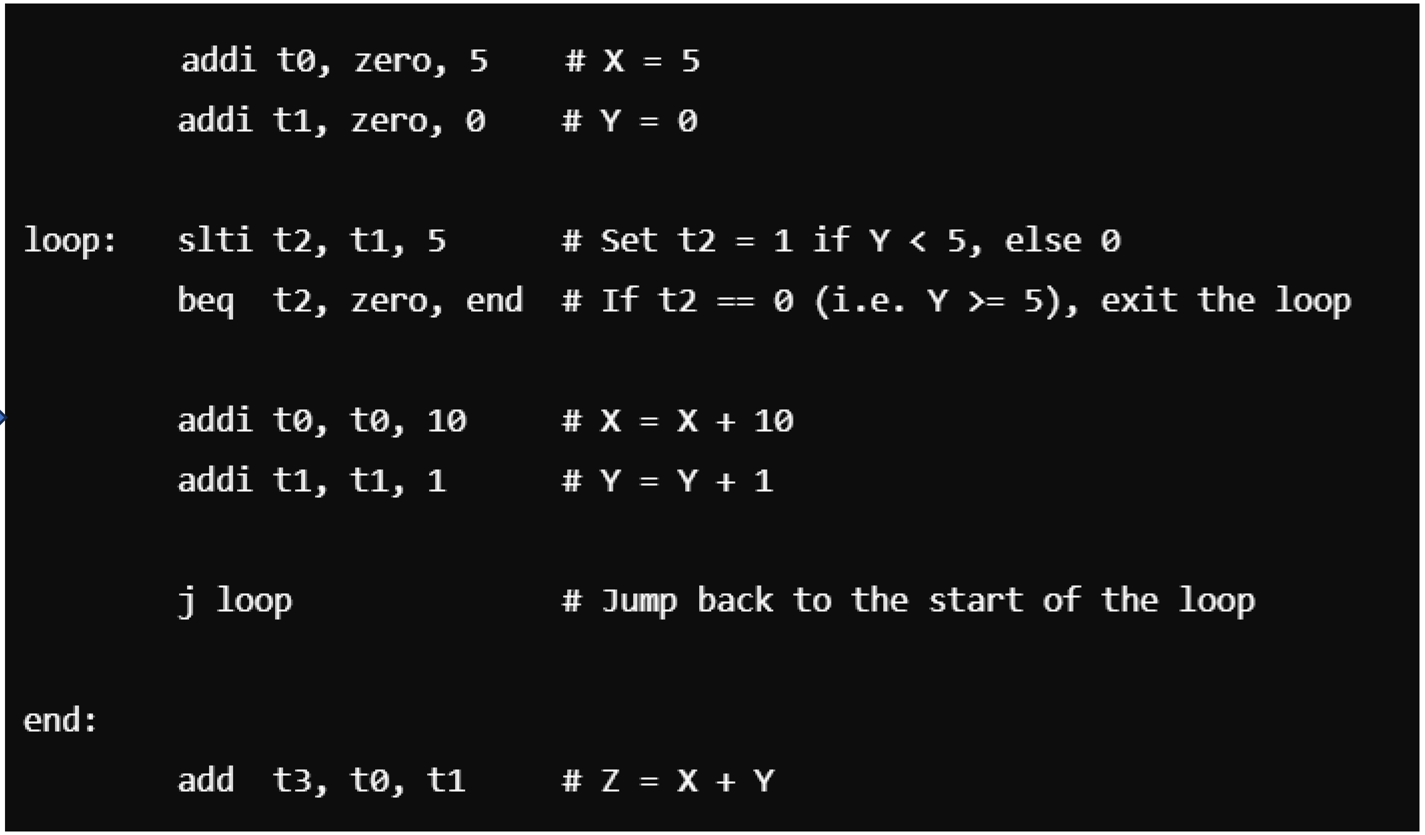
- ISA needs to be consistent and organized in order to support a wide variety of instructions.
- The 32-bit instruction can be broken down into several critical pieces (Ex: for an R-type Instruction).

funct7 **7 bits, [31:25]**	**rs2 (Src. Reg. 2)** **5 bits, [24:20]**	**rs1 (Src. Reg. 1)** **5 bits, [19:15]**	**funct3** **3 bits, [14:12]**	**rd (Dest. Reg.)** **5 bits, [11:7]**	**Opcode** **7 bits, [6:0]**
This field is used to further distinguish between variants of an instruction.	This field indicates the second source register operand for operations that require two registers (like many R-type operations).	The value from this register is typically one of the inputs to the operation.	This secondary opcode field further refines the operation defined by the primary opcode.	In instructions that write results to a register (like R-type and I-type instructions), these bits specify the destination register (rd).	This field identifies the broad class of the instruction (for example, whether it's an arithmetic operation, a load/store, or a branch).

Let's convert a simple program into a set of instructions (machine code)



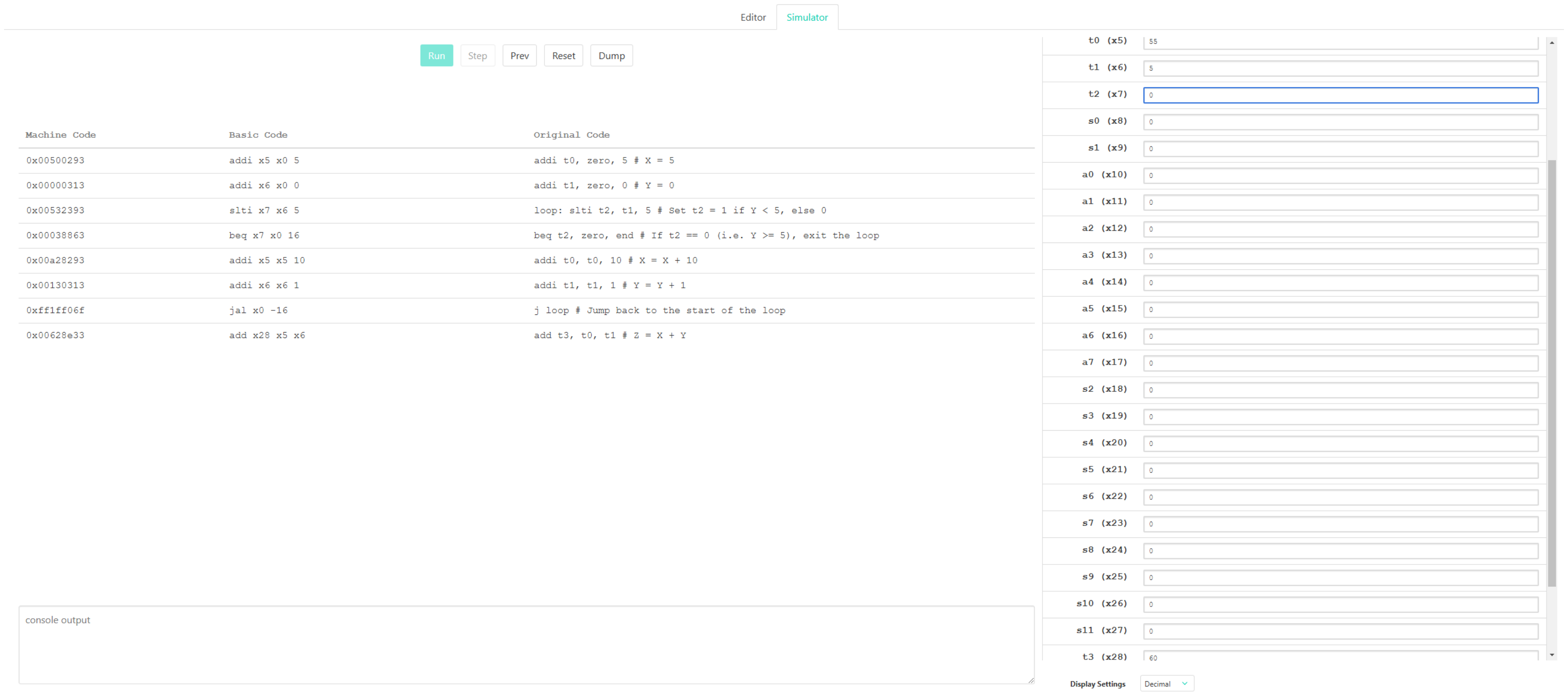
```
X = 5;
Y = 0;
while (Y < 5) {
    X = X + 10;
    Y++;
}
Z = X + Y;
```



```
        addi t0, zero, 5    # X = 5
        addi t1, zero, 0    # Y = 0

loop:   slti t2, t1, 5      # Set t2 = 1 if Y < 5, else 0
        beq  t2, zero, end  # If t2 == 0 (i.e. Y >= 5), exit the loop

        addi t0, t0, 10     # X = X + 10
        addi t1, t1, 1      # Y = Y + 1

        j loop              # Jump back to the start of the loop

end:
        add  t3, t0, t1     # Z = X + Y
```

A good start, but a processor still can't read this. Let's visit: https://venus.kvakil.me/

Let's convert a simple program into a set of instructions (machine code)



Machine Code	Basic Code	Original Code
0x00500293	addi x5 x0 5	addi t0, zero, 5 # X = 5
0x00000313	addi x6 x0 0	addi t1, zero, 0 # Y = 0
0x00532393	slti x7 x6 5	loop: slti t2, t1, 5 # Set t2 = 1 if Y < 5, else 0
0x00038863	beq x7 x0 16	beq t2, zero, end # If t2 == 0 (i.e. Y >= 5), exit the loop
0x00a28293	addi x5 x5 10	addi t0, t0, 10 # X = X + 10
0x00130313	addi x6 x6 1	addi t1, t1, 1 # Y = Y + 1
0xff1ff06f	jal x0 -16	j loop # Jump back to the start of the loop
0x00628e33	add x28 x5 x6	add t3, t0, t1 # Z = X + Y

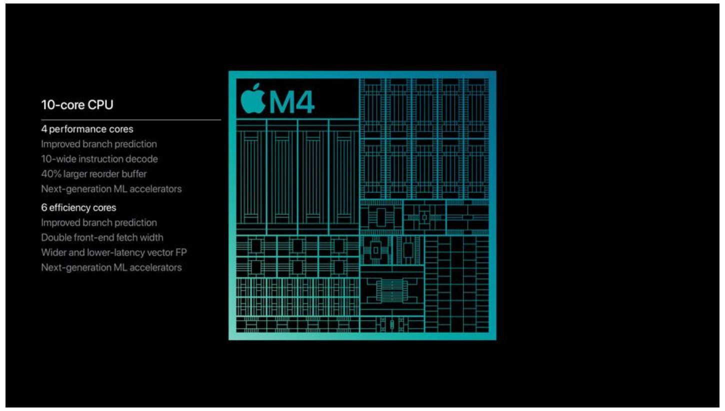
console output

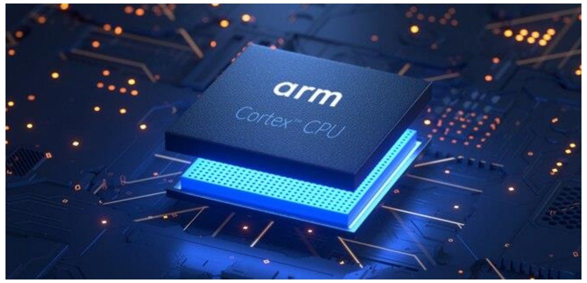
t0 (x5)	55
t1 (x6)	5
t2 (x7)	0
s0 (x8)	0
s1 (x9)	0
a0 (x10)	0
a1 (x11)	0
a2 (x12)	0
a3 (x13)	0
a4 (x14)	0
a5 (x15)	0
a6 (x16)	0
a7 (x17)	0
s2 (x18)	0
s3 (x19)	0
s4 (x20)	0
s5 (x21)	0
s6 (x22)	0
s7 (x23)	0
s8 (x24)	0
s9 (x25)	0
s10 (x26)	0
s11 (x27)	0
t3 (x28)	60

Display Settings: Decimal

Other ISAs

- X86, incredibly widespread platform used on almost all Intel and AMD builds
- ARM, another "reduced instruction set" ISA, but not open source like RISC-V
- Apple switched to an ARM based ISA in 2020 (the switch from Intel to Apple Silicon in Macs)


10-core CPU
4 performance cores
Improved branch prediction
10-wide instruction decode
40% larger reorder buffer
Next-generation ML accelerators
6 efficiency cores
Improved branch prediction
Double front-end fetch width
Wider and lower-latency vector FP
Next-generation ML accelerators
M4

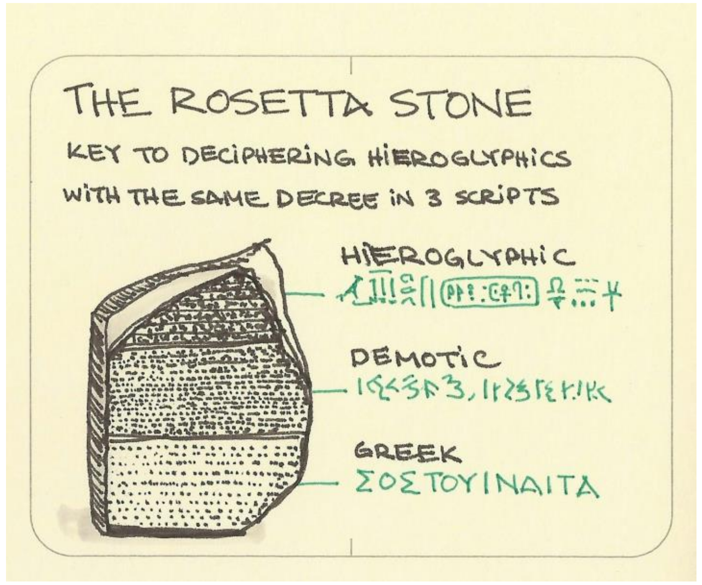


x86 vs ARM, Pros for both

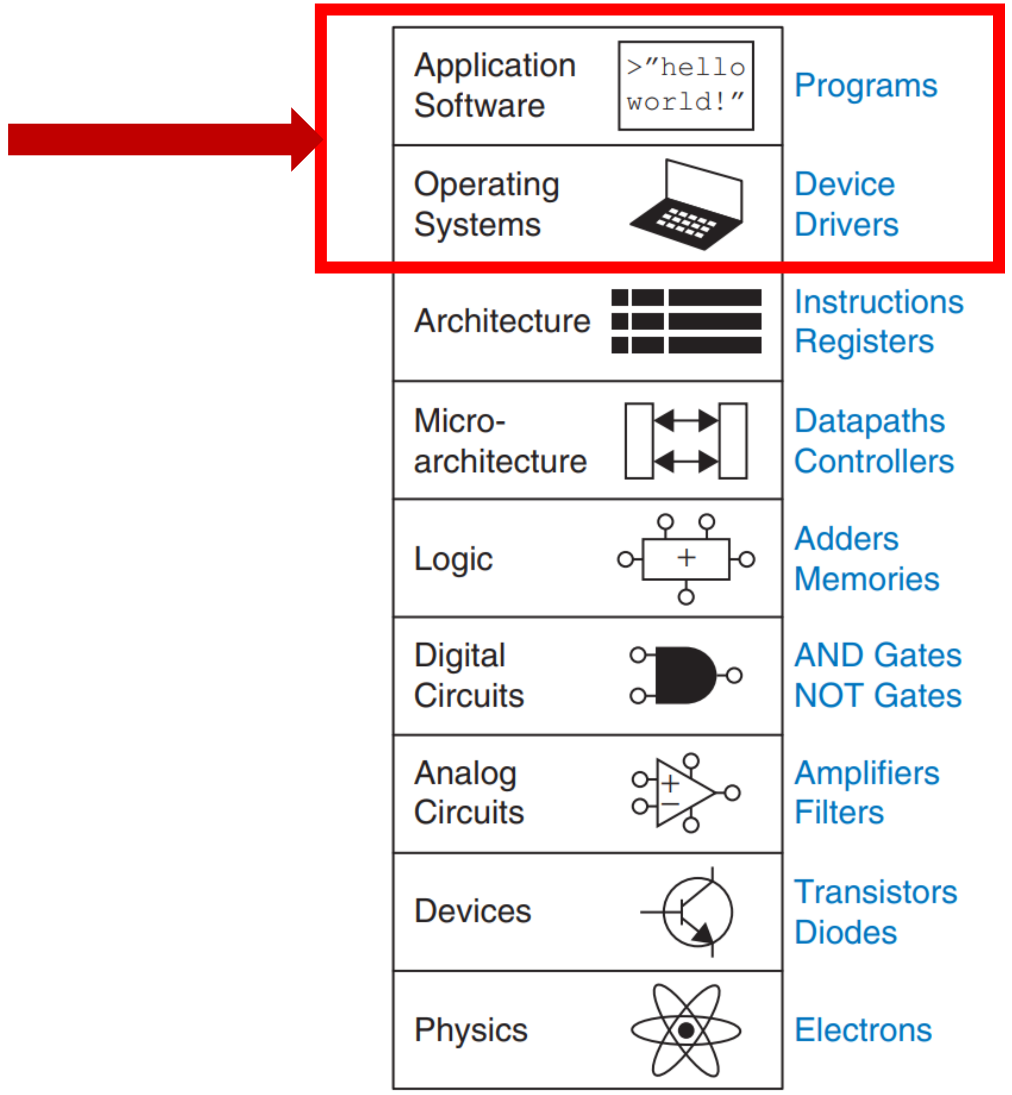
x86	ARM
• Higher raw performance for intensive tasks (video editing, gaming, data analysis) • Wide software compatibility (large software ecosystem) • Wide and flexible instruction sets, allowing for greater customization and optimization	• Increased power efficiency, due to low power consumption (ideal for mobile devices and embedded systems with limited battery life). • Cost-effective

Apple Rosetta

- A dynamic binary translator for macOS.
- Released in 2006 with new Intel Macs to allow applications to run from previous Macs using PowerPC processors (Dropped in 2011).
- Rosetta 2 released in 2020 with Apple Silicon, allowing Intel applications to run on Apple silicon-based Macs.
- Uses Ahead-of-Time (AOT) Translation to pre-translate parts of the application into Arm code (Cost of time). Also uses Just-In-time (JIT) Dynamic Translation on the fly (Efficient enough to not notice performance degradation).
- Low-level or Kernal level features might still encounter issues and still comes with a performance cost.
- "Whisky" is similar in concept and allows games to run on macOS.

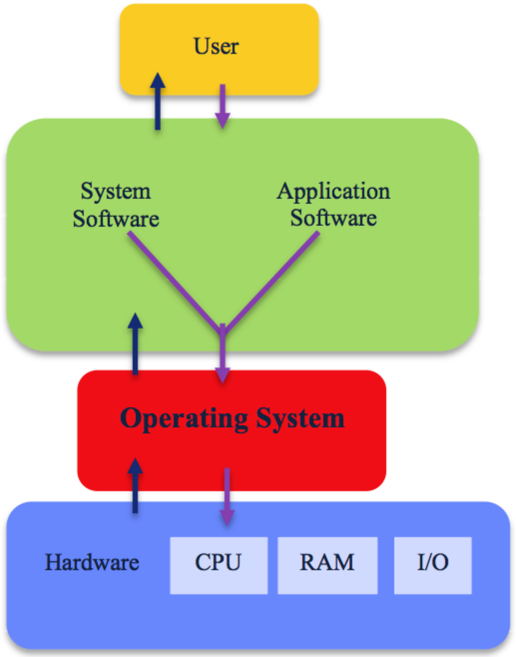

THE ROSETTA STONE
KEY TO DECIPHERING HIEROGLYPHICS
WITH THE SAME DECREE IN 3 SCRIPTS
HIEROGLYPHIC
DEMOTIC
GREEK
ΣΟΣΤΟΥΙΝΑΙΤΑ

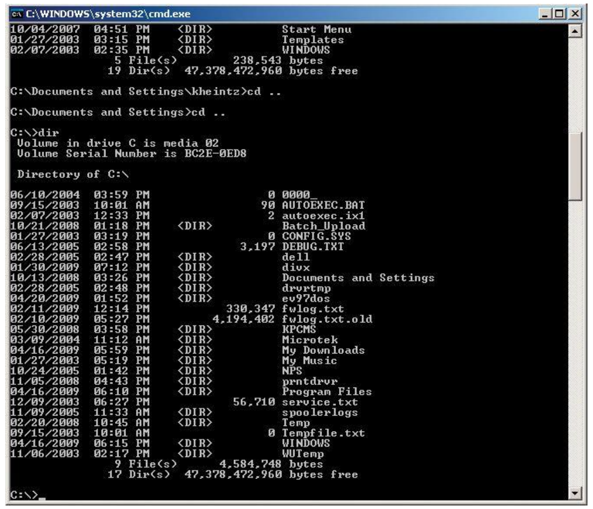
Operating Systems & Application Software



Operating Systems

- Operating Systems exist to hide the complexities of underlying hardware (e.g., registers, memory management, I/O operations)
- Provide a simpler interface for applications.
- Operating systems will look at process and thread management, memory management, contain a file system and storage management, and manage I/O devices
- Modern OS will also consider Networking protocols, security and user management, and virtualization (multiple instances on one machine)
- Current OS include Windows (evolved from MS-DOS), Linux, MacOS, ChromeOS. Mobile devices include Android and iOS.

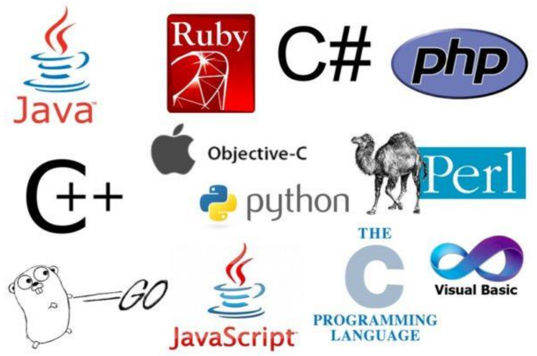


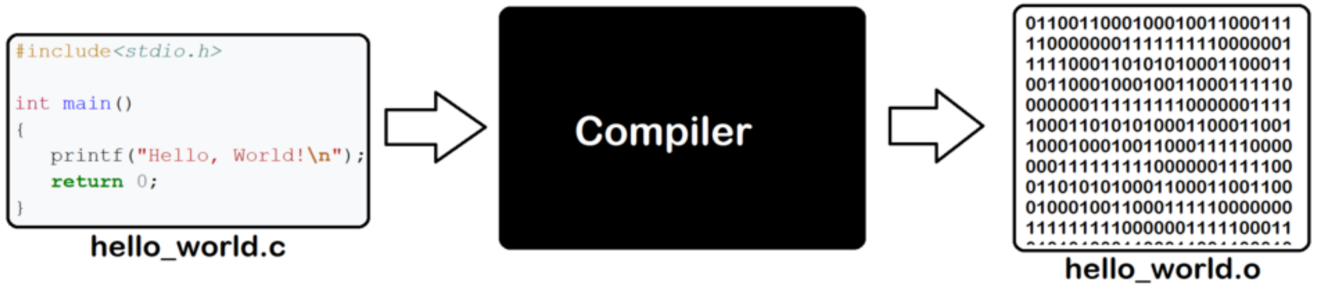


```
10/04/2007  04:51 PM    <DIR>          Start Menu
01/27/2003  03:15 PM    <DIR>          Templates
02/07/2003  02:35 PM    <DIR>          WINDOWS
               5 File(s)        238,543 bytes
              19 Dir(s)  47,378,472,960 bytes free

C:\Documents and Settings\kheintz>cd ..

C:\Documents and Settings>cd ..

C:\>dir
 Volume in drive C is media 02
 Volume Serial Number is BC2E-0ED8

 Directory of C:\

06/10/2004  03:59 PM                 0 0000_
09/15/2003  10:01 AM                90 AUTOEXEC.BAT
02/07/2003  12:33 PM                 2 autoexec.ix1
10/21/2008  01:18 PM    <DIR>          Batch_Upload
01/27/2003  03:19 PM                 0 CONFIG.SYS
06/13/2005  02:58 PM             3,197 DEBUG.TXT
02/28/2005  02:47 PM    <DIR>          dell
01/30/2009  07:12 PM    <DIR>          divx
10/13/2008  03:26 PM    <DIR>          Documents and Settings
02/28/2005  02:48 PM    <DIR>          drvrtmp
04/20/2009  01:52 PM    <DIR>          ev97dos
02/11/2009  12:14 PM           330,347 fwlog.txt
02/10/2009  05:27 PM         4,194,402 fwlog.txt.old
05/30/2008  03:58 PM    <DIR>          KPCMS
03/09/2004  11:12 AM    <DIR>          Microtek
04/16/2009  05:59 PM    <DIR>          My Downloads
01/27/2003  05:19 PM    <DIR>          My Music
10/24/2005  01:42 PM    <DIR>          NPS
11/05/2008  04:43 PM    <DIR>          prntdrvr
04/16/2009  06:10 PM    <DIR>          Program Files
12/09/2003  06:27 PM            56,710 service.txt
11/09/2005  11:33 AM    <DIR>          spoolerlogs
02/20/2008  10:45 AM    <DIR>          Temp
09/15/2003  10:01 AM                 0 Tempfile.txt
04/16/2009  06:15 PM    <DIR>          WINDOWS
11/06/2003  02:17 PM    <DIR>          WUTemp
               9 File(s)      4,584,748 bytes
              17 Dir(s)  47,378,472,960 bytes free

C:\>_
```

Application Software

- Written in programming languages that vary in complexity and features
- High level languages (C, Java, Python, Rust) can allow developers to write code without worrying about the intricate details of the underlying hardware. Features include structured programming, object-oriented programming, garbage collection, standard libraries.
- A compiler will translate high level code (C, Java, Rust) into low level code (assembly), which the computer can understand. An assembler can than translate assembly into machine code (binary).
- Interpreted languages (Python) use an interpreter rather than following the compiling chain, meaning source code is interpreted at runtime rather than being pre-compiled.
- Modern Java environments may use Just-In-Time (JIT) which compiles code on the fly during execution.



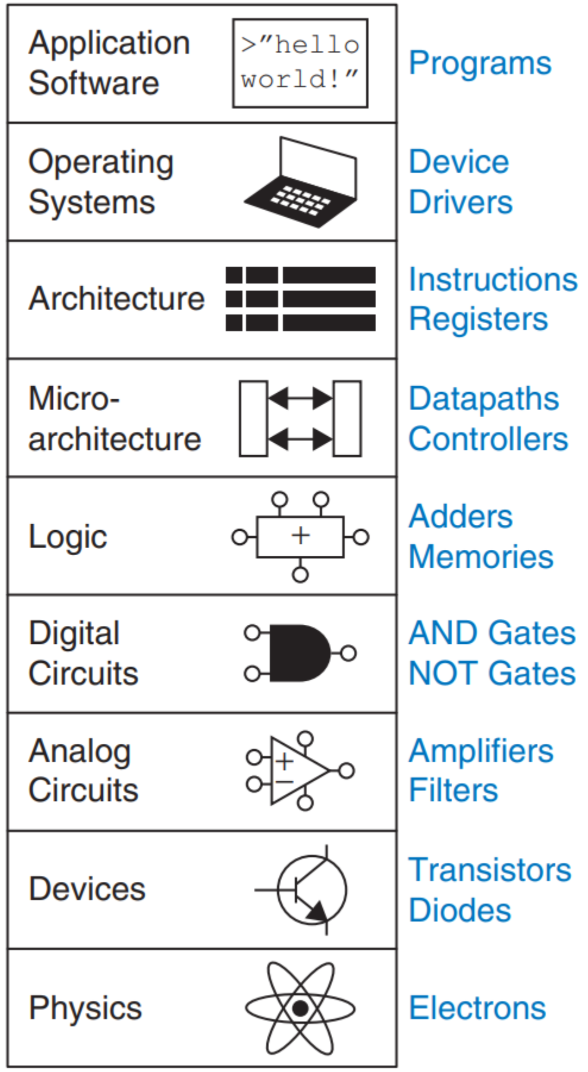


```c
#include<stdio.h>

int main()
{
   printf("Hello, World!\n");
   return 0;
}
```

```
0110011000100010011000111
1100000000111111111000000 1
1111000110101010001100011
00110001000100110001111110
0000001111111110000001111
1000110101010001100011001
10001000100110001111110000
00011111111110000001111100
01101010100011000110011 00
01000100110001111110000000
1111111110000000111110001 1
```

The End

Thanks for listening!



GBM Attendance