NCDC

NUST CHIP DESIGN CENTRE

Computer Architecture

Introduction to Processor Design

Lecture 1

Agenda

- From Transistors to Software
- Memory
- Pipelining
- Register File

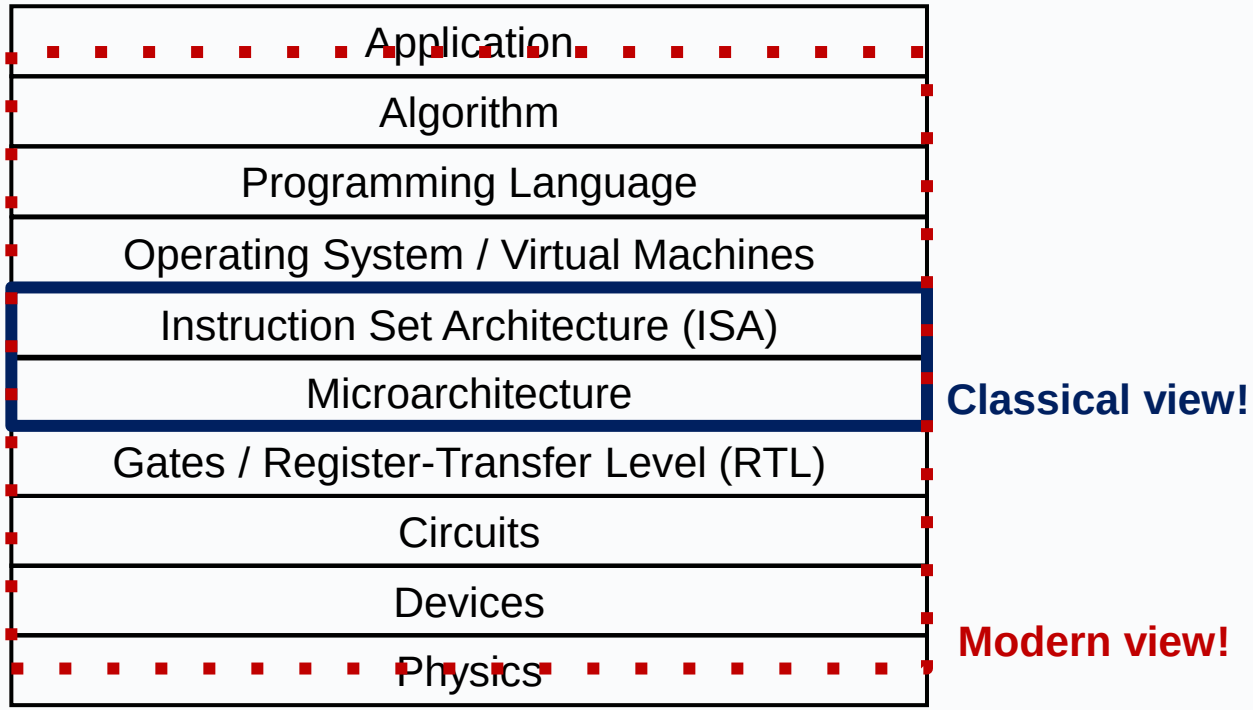
Agenda

- From Transistors to Software
- Memory
- Pipelining
- Register File

Abstraction Layers

- Divide and conqueror approach to designing complex digital systems!

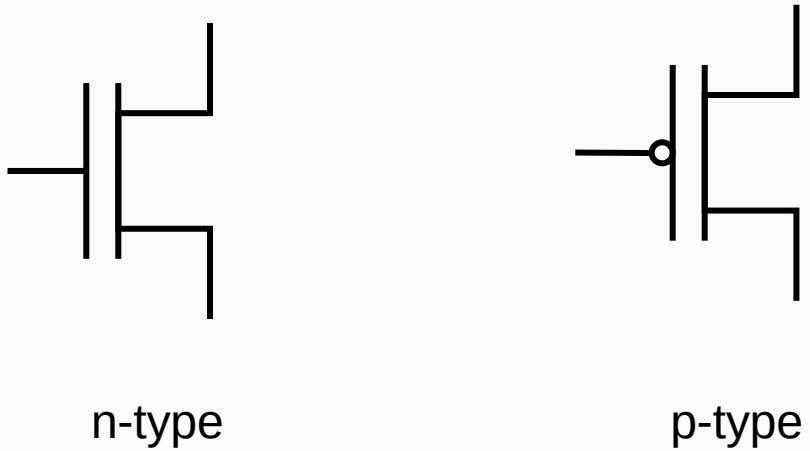
Source: phys.org



Computer Architecture *is the science and art of designing computing platforms*
(hardware, interface, system SW, and programming model)

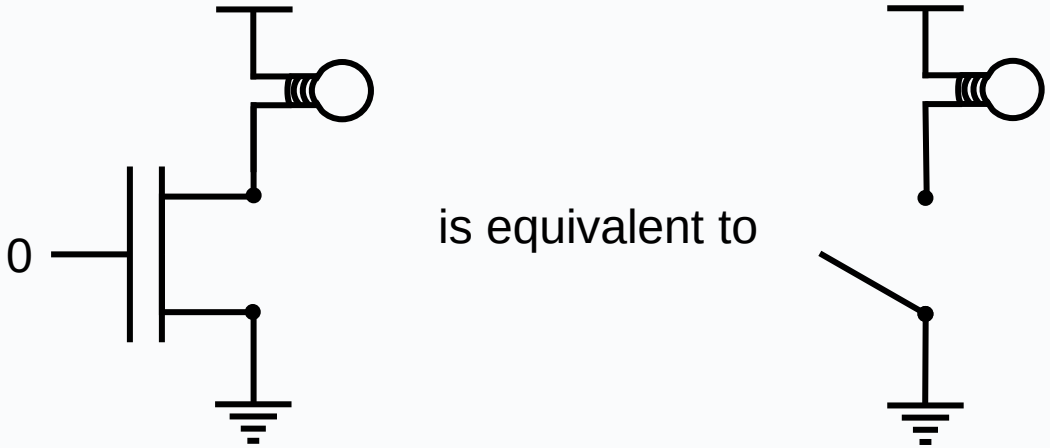
Transistor as a Switch

- MOSFET Transistors



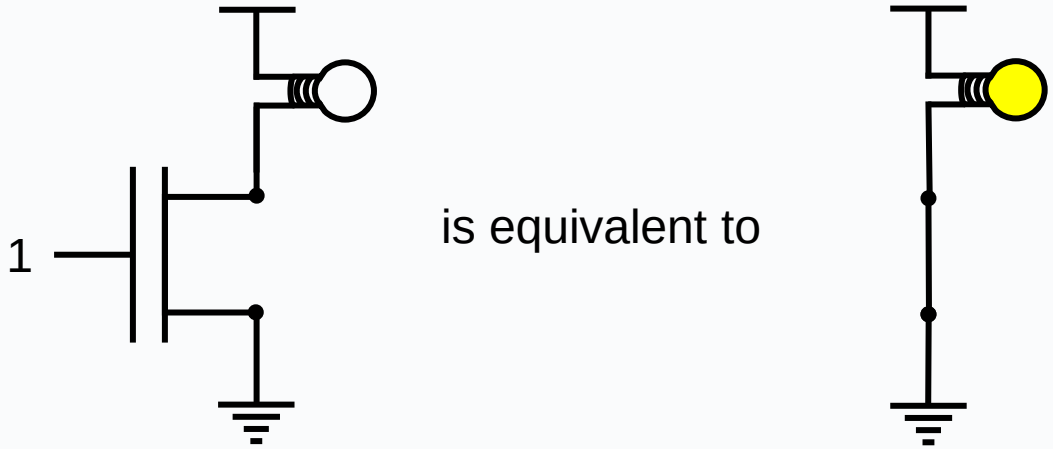
Transistor as a Switch

- MOSFET Switching Behavior
  - n-type



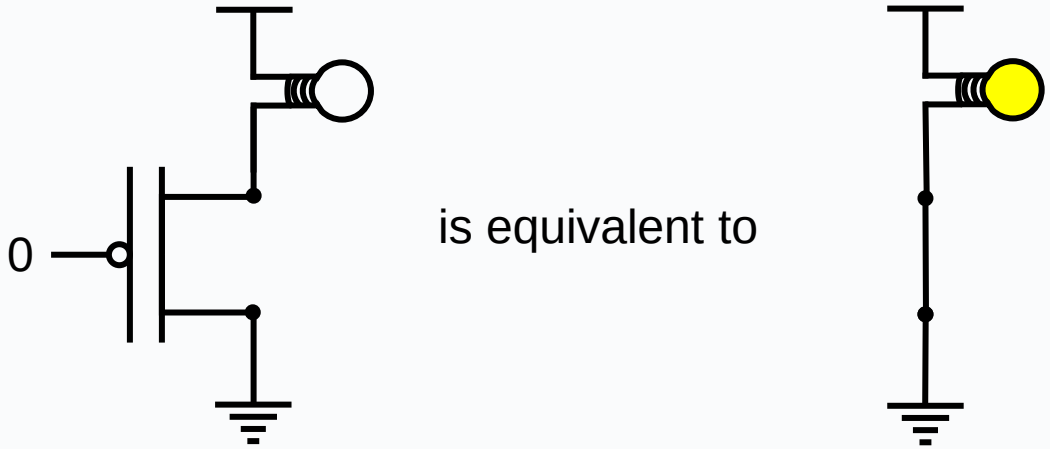
Transistor as a Switch

- MOSFET Switching Behavior
  - n-type



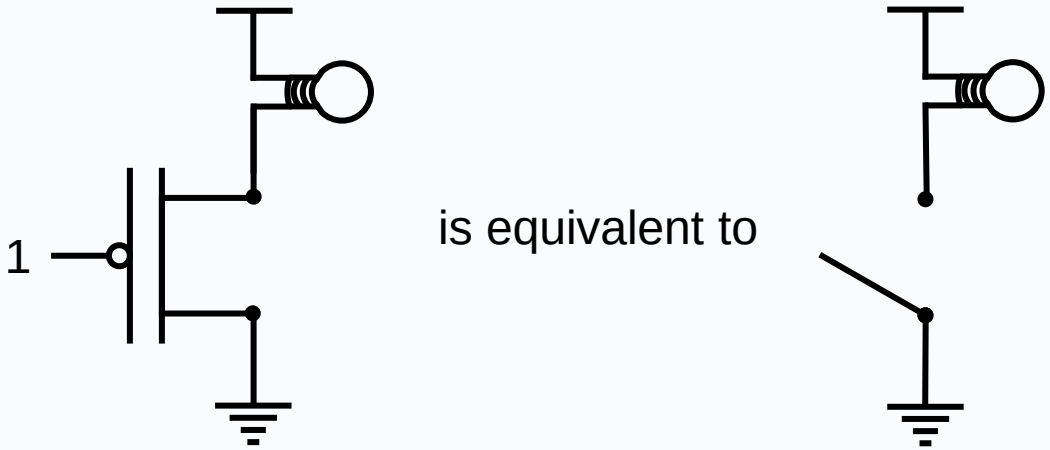
Transistor as a Switch

- MOSFET Switching Behavior
  - p-type



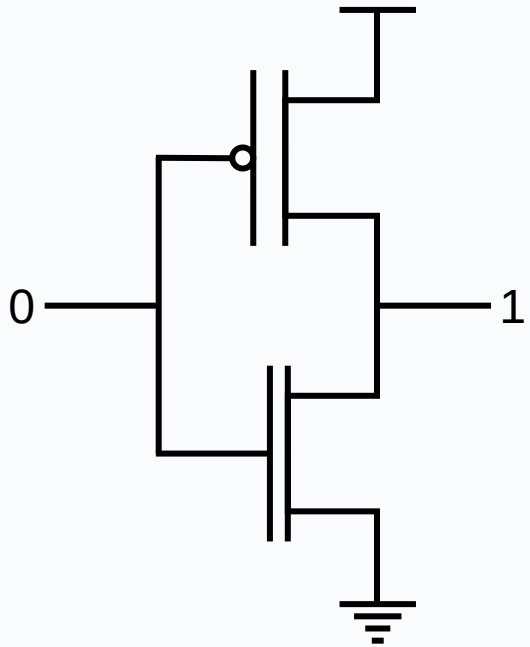
Transistor as a Switch

- MOSFET Switching Behavior
  - p-type



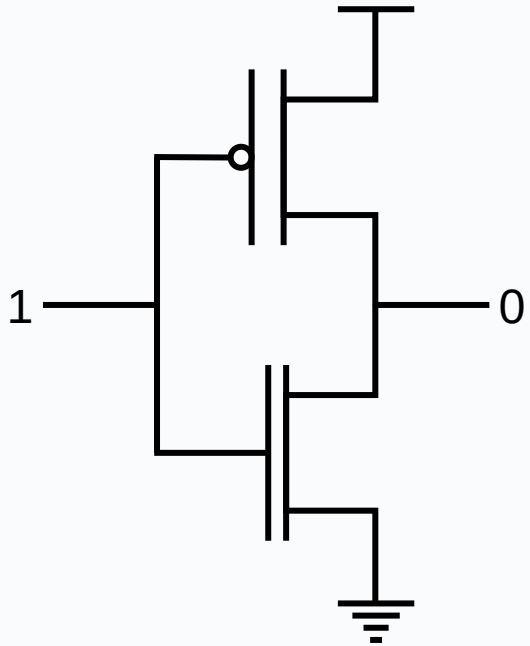
Building Logic Gates

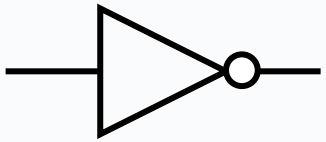
- Inverter or NOT Gate



Building Logic Gates

- Inverter or NOT Gate

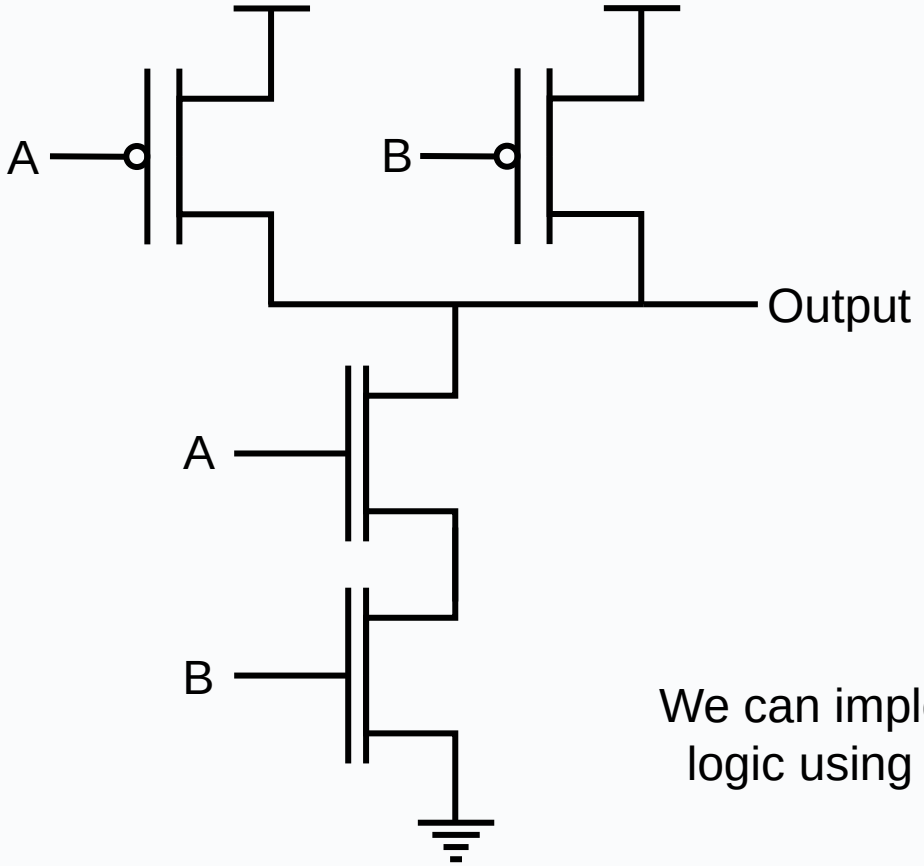


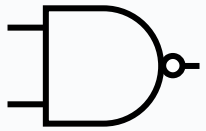


Symbol

Building Logic Gates

- NAND Gate



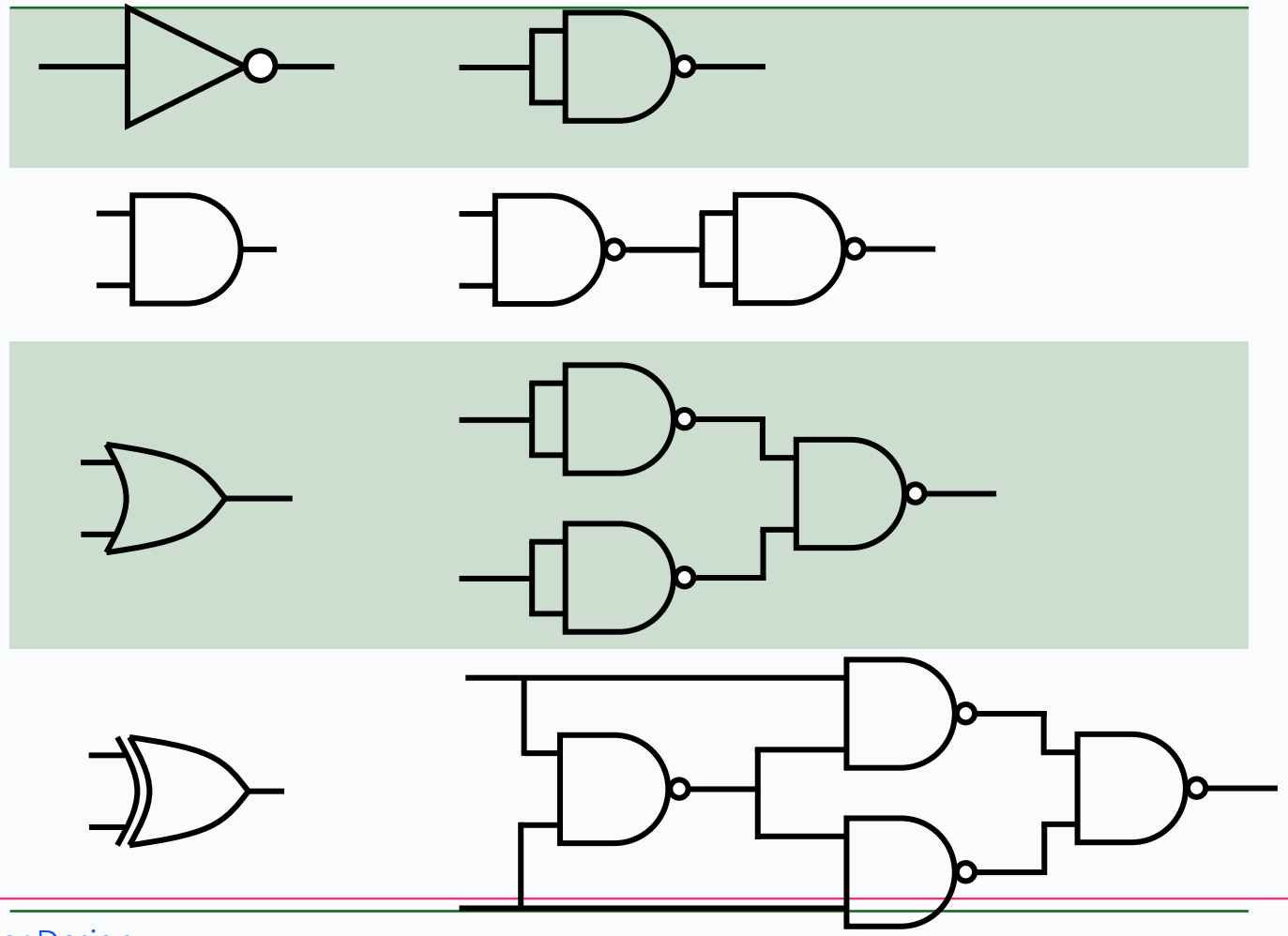


Symbol

We can implement any Boolean logic using only NAND gates!

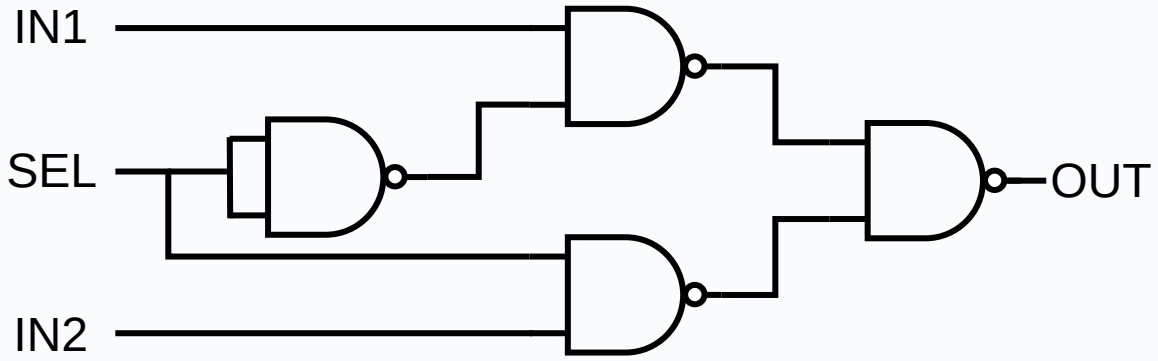
Building Logic Gates

- NAND (or NOR) Gate is universal gate

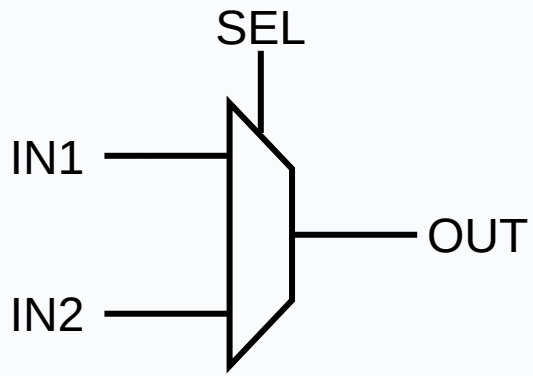


Combinational Logic

- What is this?



Two Input Multiplexer!

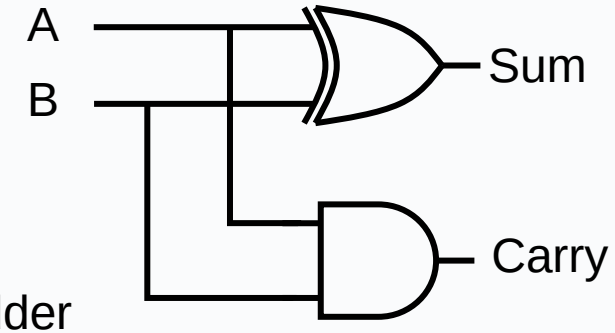


SEL	OUT
0	IN1
1	IN2

Combinational Logic

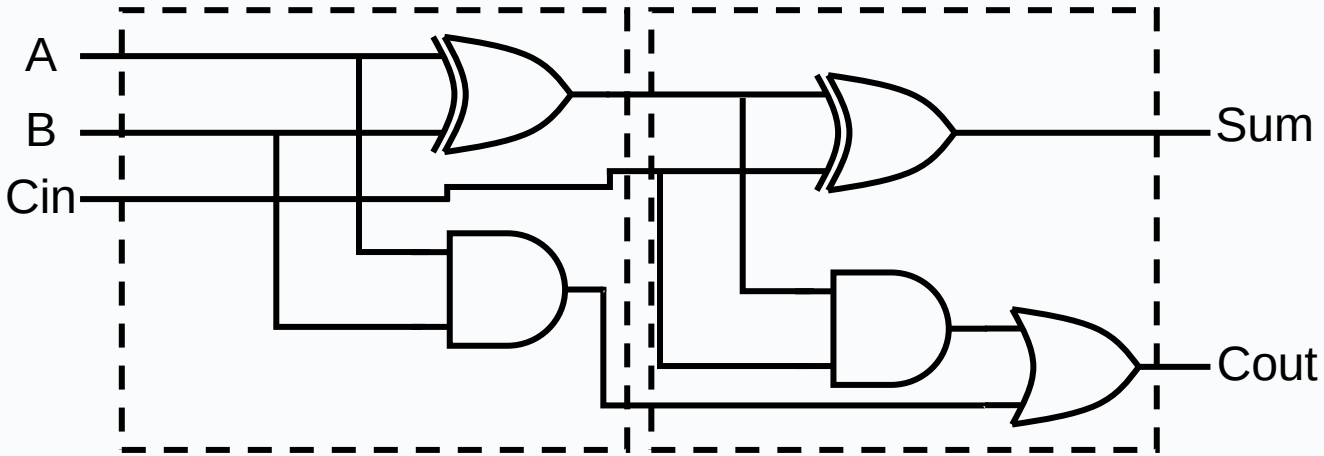
- What is this?

Half Adder!



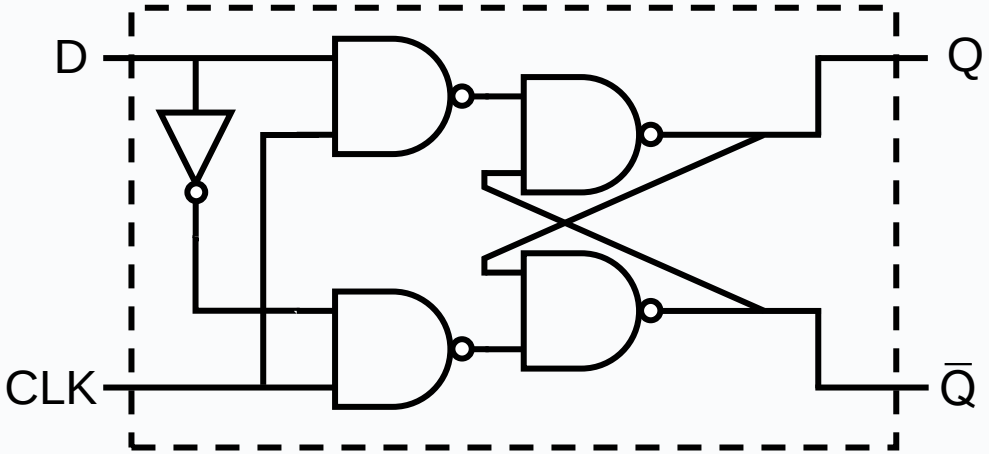
Can we make a Full Adder from two Half Adders?

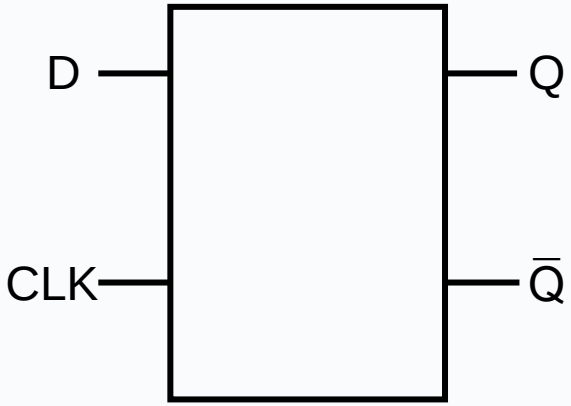
We can also implement using only NAND Gates!



Sequential Logic

- Latch

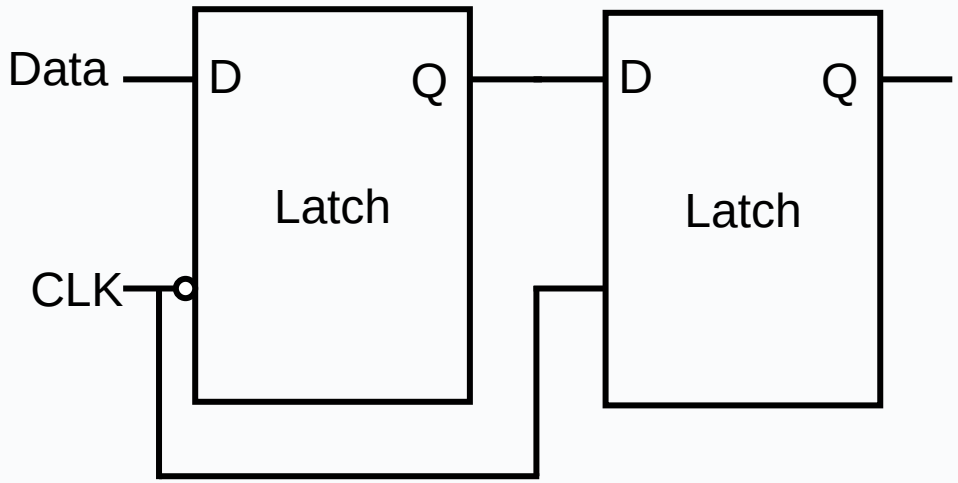




CLK	D	Q
1	0	0
1	1	1
Else	x	Q

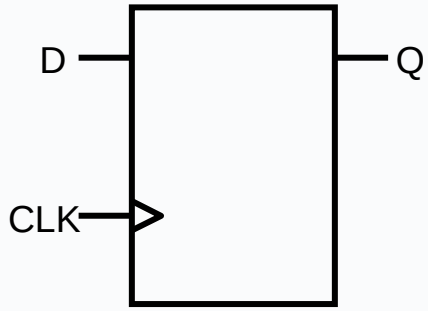
Sequential Logic

- Flip Flop / Register
  - Implemented using latches
    - There are other ways too …



Multiple flip flops can be combined to store multiple bits!

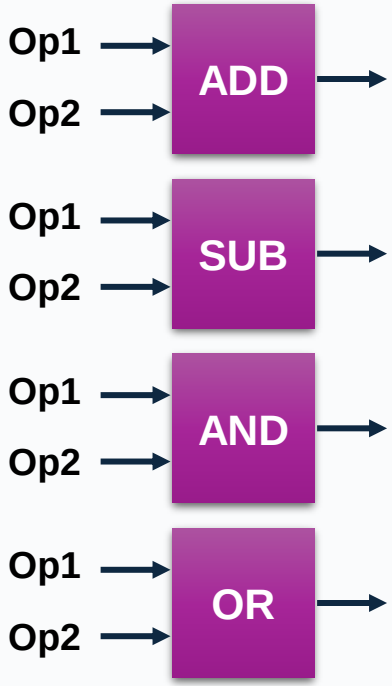
CLK	D	Q
Rising	0	0
Rising	1	1
Else	x	Q



We have got all the ingredients to build a simple ALU or even a simple Microprocessor …

Building a Simple ALU

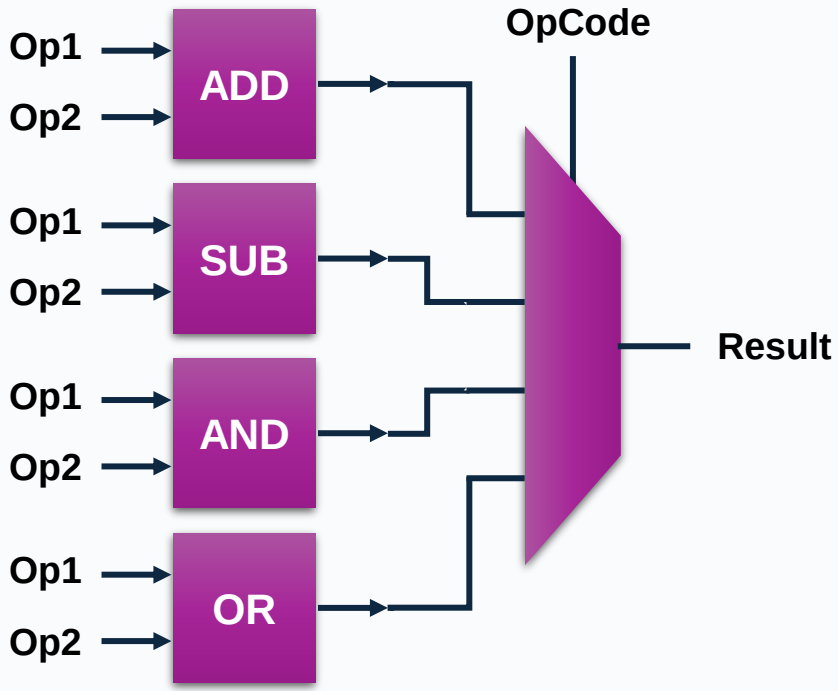
- Let's build an ALU with two operands and four operations!



The operands can be single bit or multi-bit!

From ALU to Microprocessor

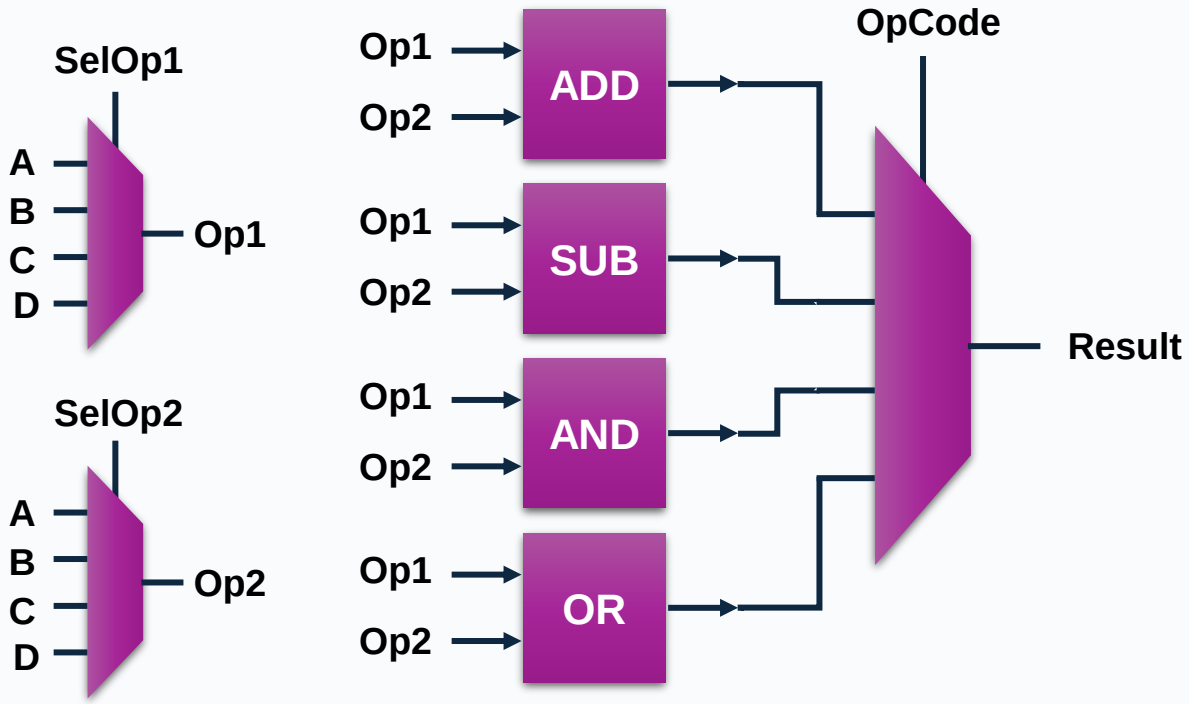
- Let's add logic to select result



2-bit OpCode selects one of the four operations!

From ALU to Microprocessor

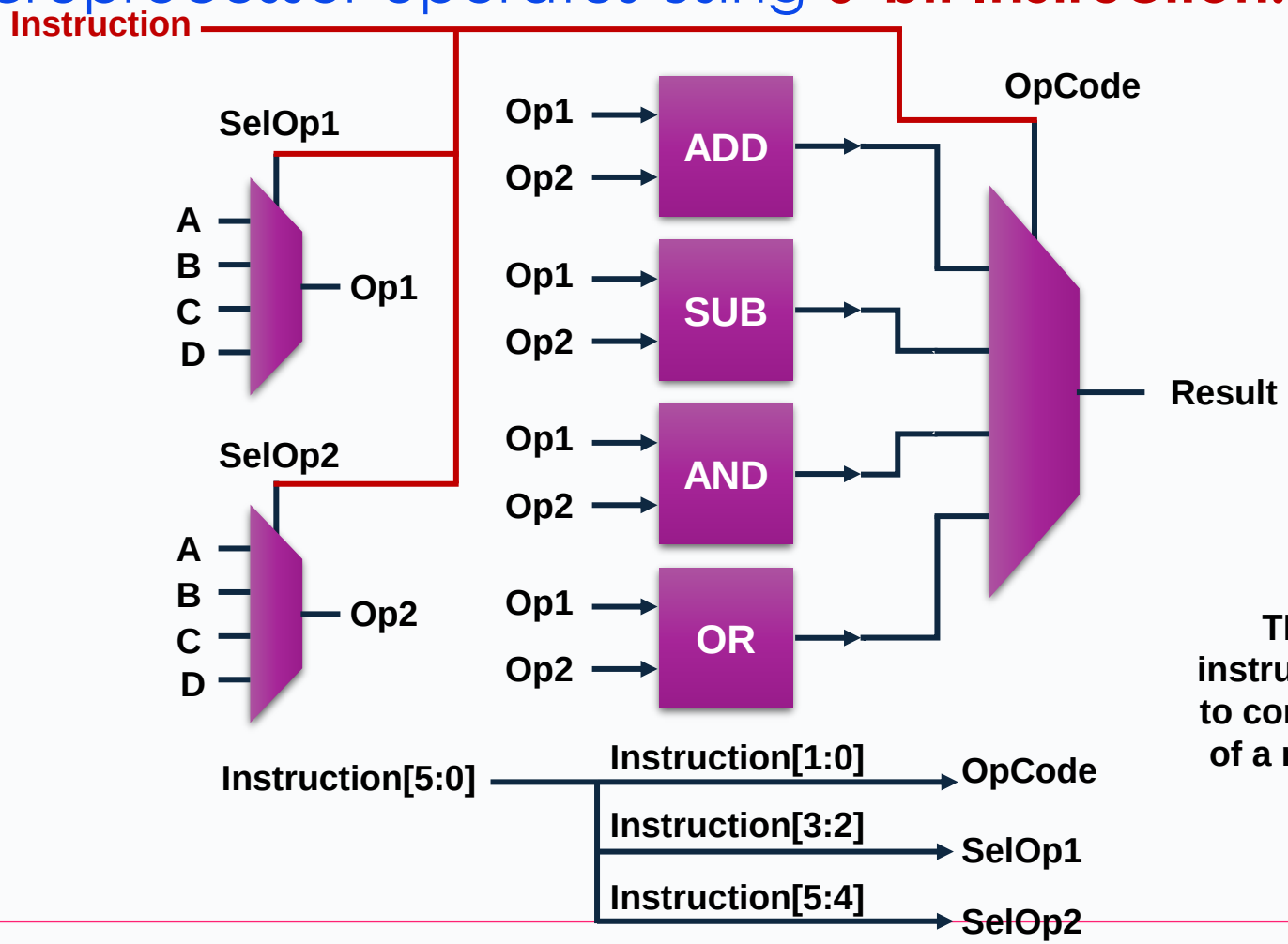
- Selecting two operands from four possible ones!



Together SelOp1, SelOp2 and OpCode form 6 bits that control the working of this simple Microprocessor!

A Simple Microprocessor

- This Microprocessor operates using **6-bit Instruction!**



That's how the instruction is decoded to control the working of a microprocessor!

Programming Microprocessor

- To program simple microprocessor, we can use 6-bit instruction
  - Suppose we want to add B and C
    - OpCode = 00 for ADD
    - SelOp1 = 01 to select B
    - SelOp2 = 10 to select C
    - Corresponding Instruction is 100100 i.e., {SelOp2, SelOp1, OpCode}!
  - 100100 is **machine code** or **machine language**
    - That's the lowest level programming language understood directly by the hardware!
    - That's not quite readable or easier to use for humans, isn't it?
      - So, we use high-level languages!

Programming Microprocessor

- **ADD B, C** could be a possible syntax of assembly language instruction for our simple microprocessor!
  - Similarly, we can have **AND A, D**
  - **OR C, A** and so on …
  - All the possible instructions form the **instruction set** for our microprocessor!

Programming Microprocessor

- High-level programming languages were made to simplify our task beyond assembly language!
  - For instance, a statement such **Result = A+B+C+D** in a high-level language may replace following assembly language instructions
    - ADD A, B # Store result in Result
    - MOV X, Result
    - ADD C, D
    - MOV Y, Result
    - ADD X, Y
      - We will need to add some more logic and temporary registers in our microprocessor to support all the above instructions!

Programming Microprocessor

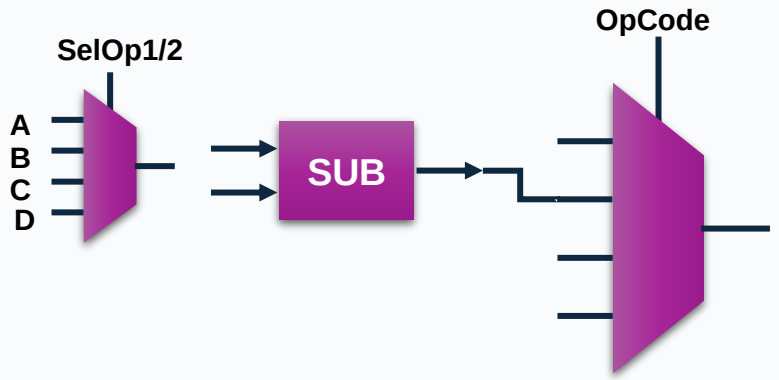
- A Compiler translates software from a high-level language to assembly language!
  - There can be different assembly language codes for the same high-level language code!
- An Assembler translates the assembly language to machine language!
  - There is exactly one possible machine instruction for every assembly language instruction!
  - Compilers can directly generate machine code as well!
- Microarchitecture is the logic that executes the machine language!

Measuring Performance

- The implementation is simple!
  - Pure combinational circuit!
    - Output changes when input instruction changes!
- Performance
  - If we add a register at input (for instruction) and one at output (for result), each instruction takes one cycle to execute!
  - CPI (cycles per instruction) = 1
  - IPC (instructions per cycle) = 1

Measuring Performance

- How fast is this processor?
  - Depends on the critical path!
  - Critical path consists of
    - Two multiplexers
    - Subtractor (longest path in the ALU blocks!)
  - $Max.Clock\ Rate = \frac{1}{2 \times (Muliplexer\ Delay) + Subtractor\ Delay + t_S + t_{CLK2Q}}$



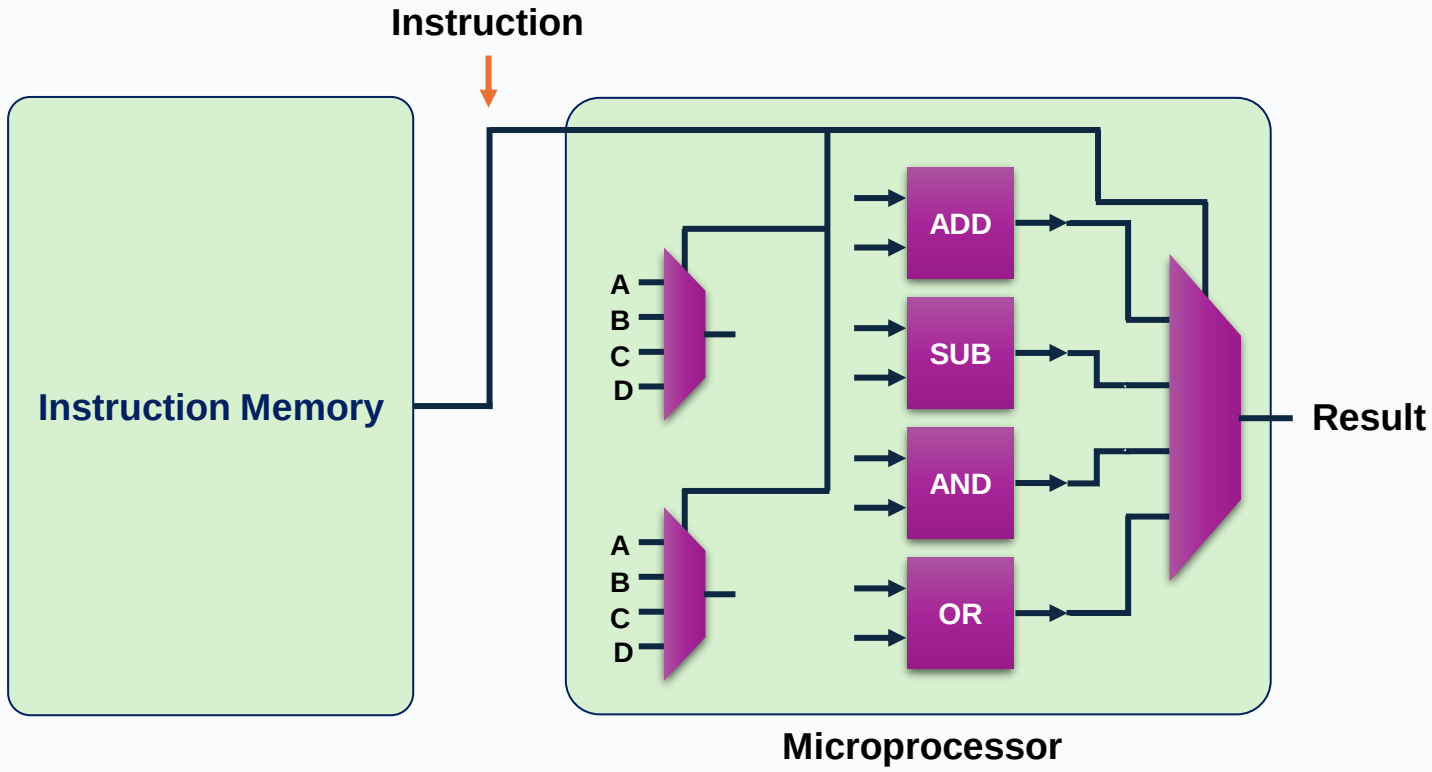
Agenda

- From Transistors to Software
- **Memory**
- Pipelining
- Register File

Let’s add memory for multiple instructions …

Adding Instruction Memory

- Instruction Memory can hold multiple instruction!

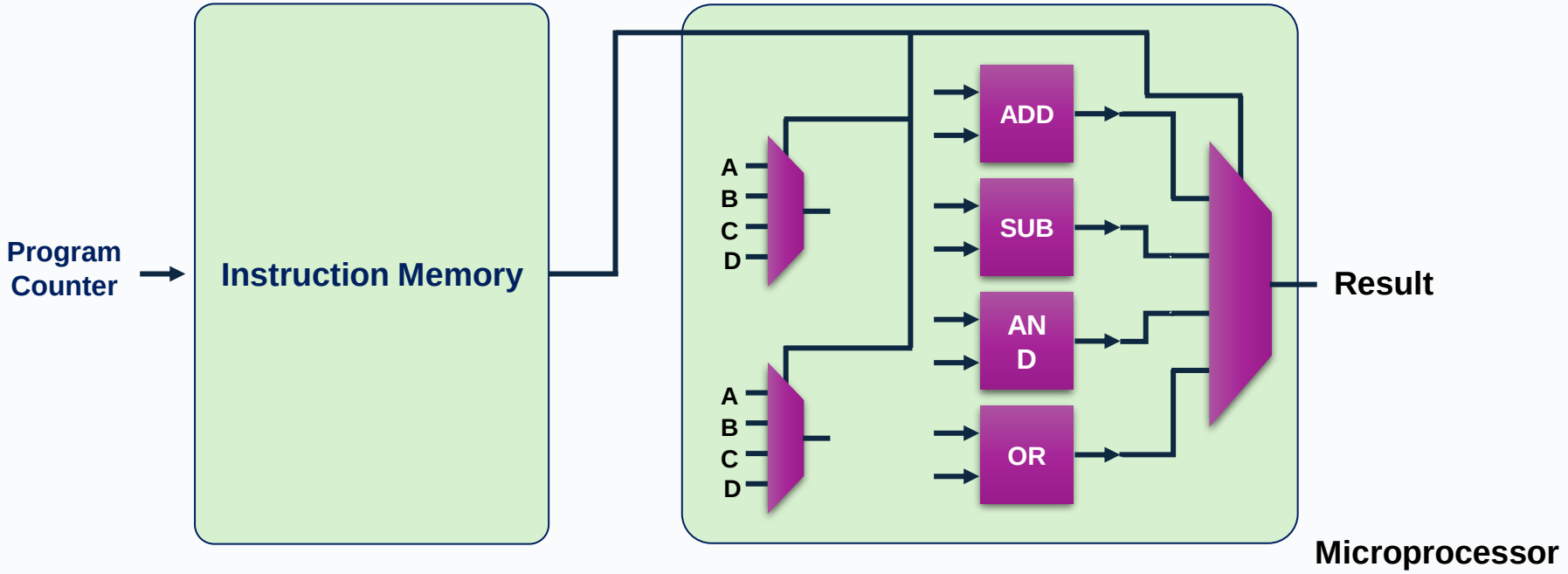


Adding Instruction Memory

- 6-bit Instructions require a simple memory where each location may be 6-bit
  - Practical memories are sized in bytes (8 bits) and usually an address corresponds to a byte address!
- Let's add an instruction memory of size 8x6 bits
  - 8 locations
  - Each location is 6 bits in size!
  - How many bits do we need for address?
    - 3 bits
  - Instruction Cache is addressed using "Program Counter"
    - Also called "Instruction Pointer Register"!
    - Usually a *register* in actual microprocessors!

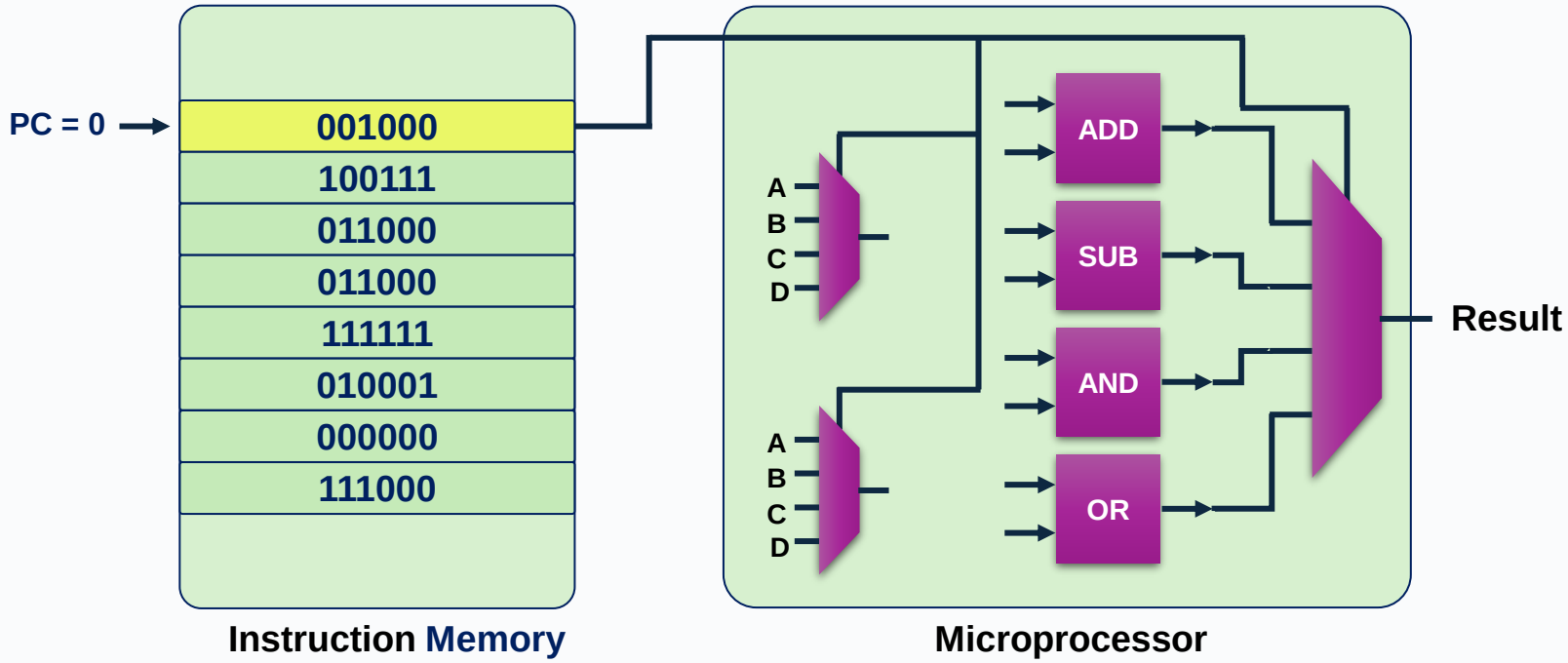
Program Counter / Instruction Pointer

- Program Counter (PC)
  - To address instruction cache!
  - For 8x6 bit instruction cache, PC is 3-bit!
    - PC can go from 000 to 111 to select 8 instructions!



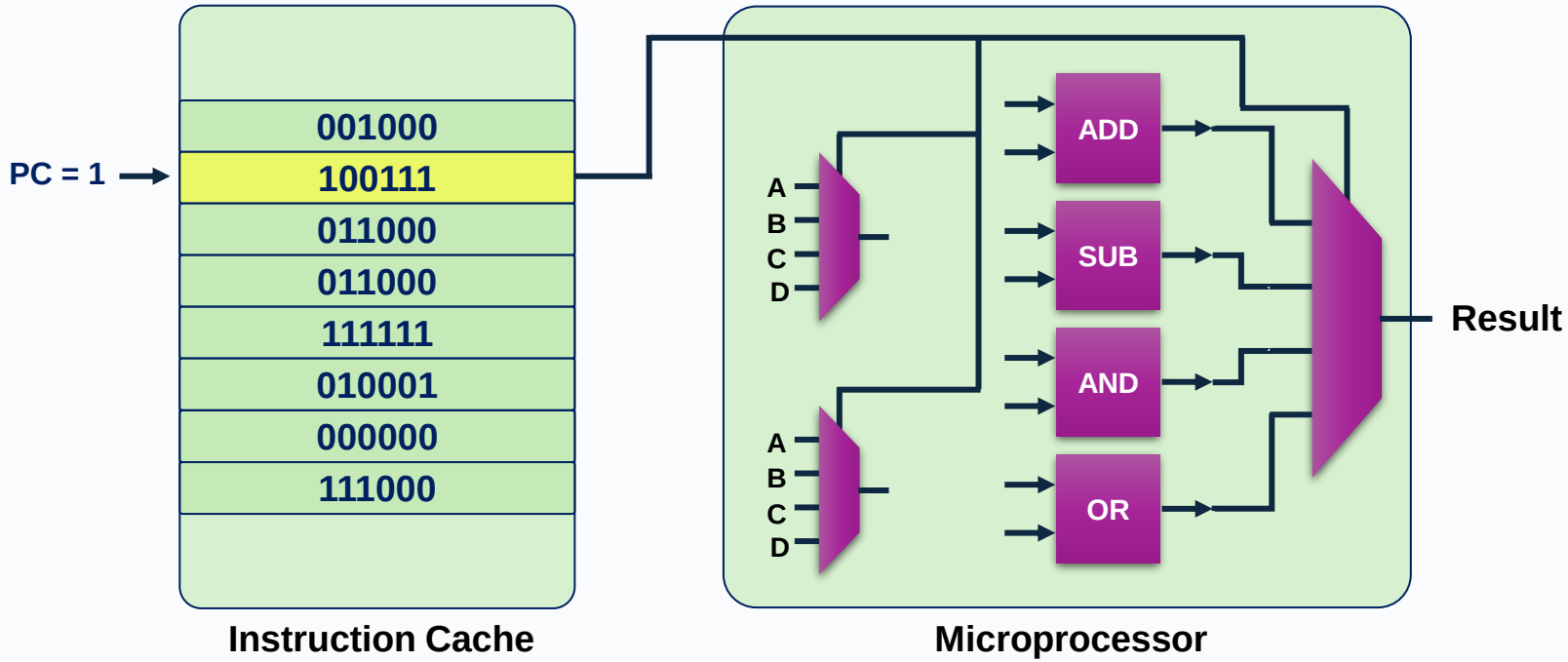
Executing Multiple Instructions

- PC is incremented to select successive instructions!



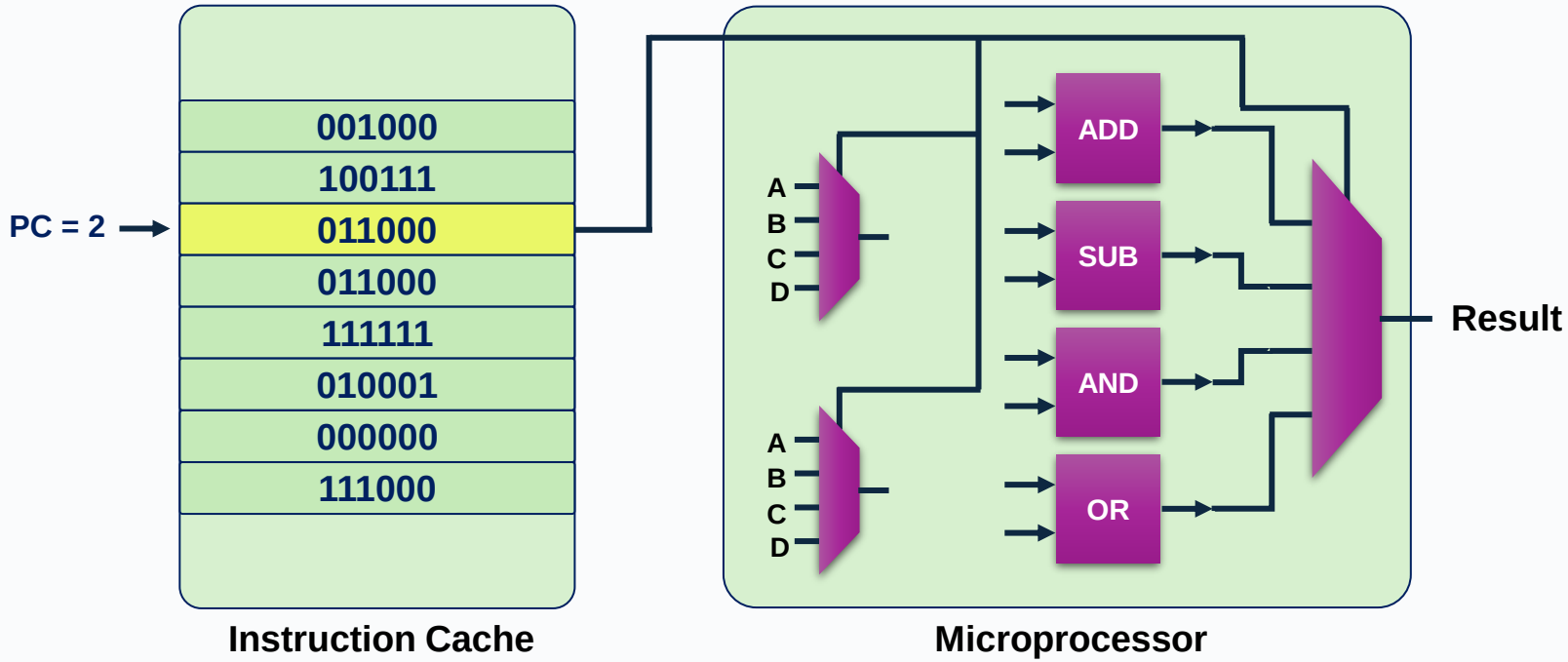
Executing Multiple Instructions

- PC is incremented to select successive instructions!



Executing Multiple Instructions

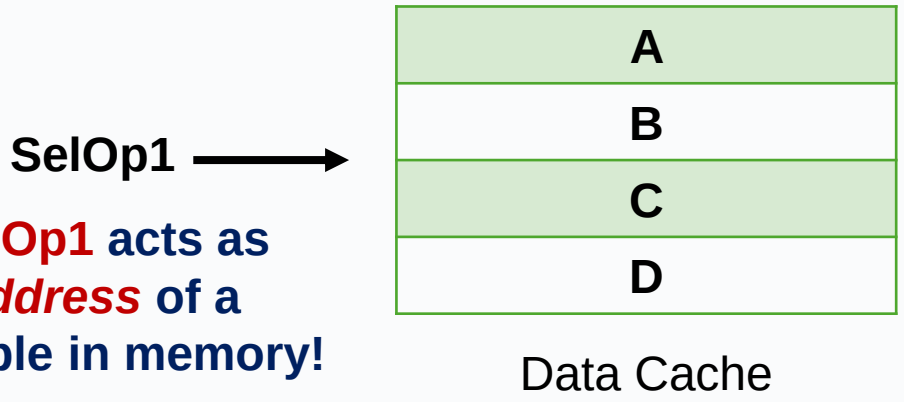
- PC is incremented to select successive instructions!



Just like Instruction Memory, we have Data Memory to hold operands to be processed…

Adding Data Memory

- Can add two simple data memories to load operands! (multiplexers will be removed)
  - Each data memory can be (4 x n) bits
    - 4 locations
    - Each having an n-bit operand!
    - 2-bit address to select one of the four operands!



SelOp1 acts as *address* of a variable in memory!

Usually, microprocessors load one operand at a time from memory instead of two!

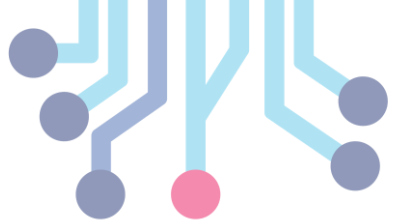
Operands are loaded into registers inside CPU!

It is also possible to have a single memory for instructions as well as data!

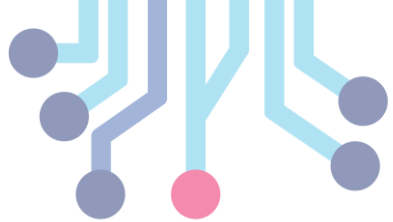
With one memory for I or D, instruction is loaded in a different clock cycle while data is accessed in a different cycle!

Agenda

- From Transistors to Software
- Memory
- **Pipelining**
- Register File

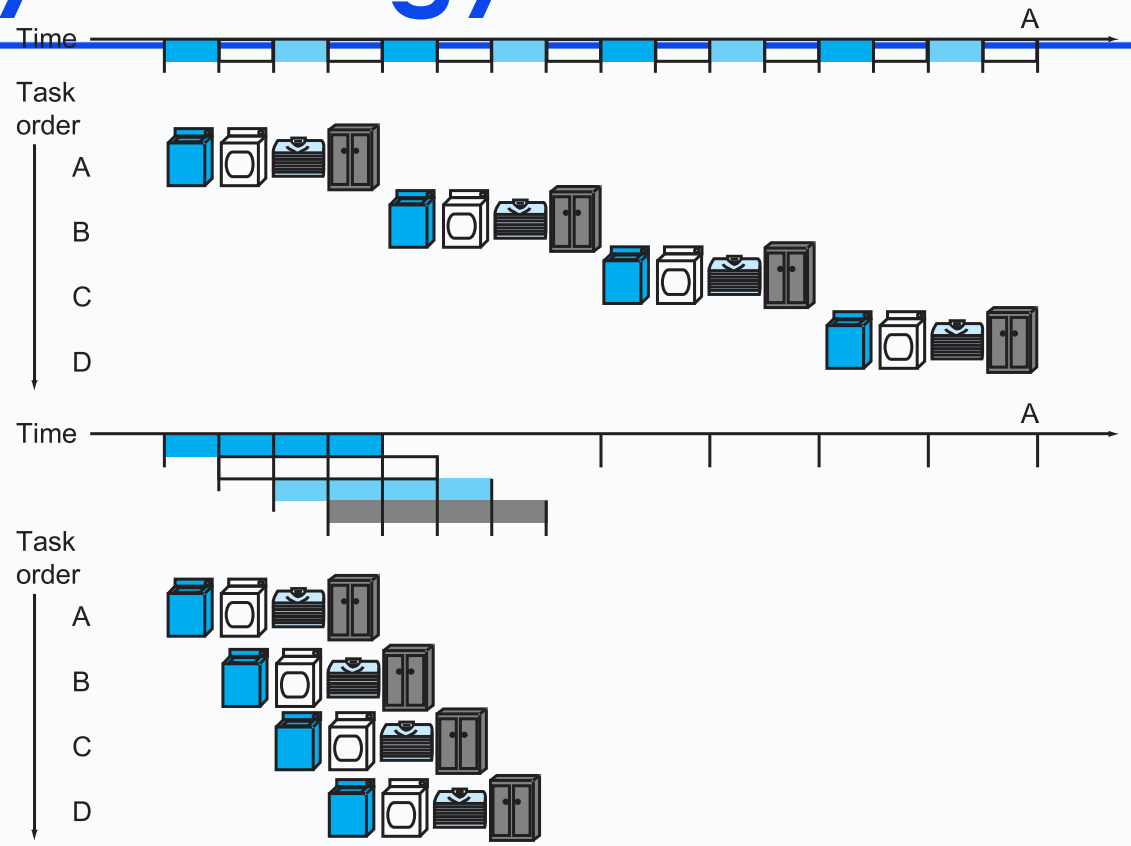


Can we improve our processor's performance?



Pipelining is an important technique for improving performance …

A Laundry Analogy

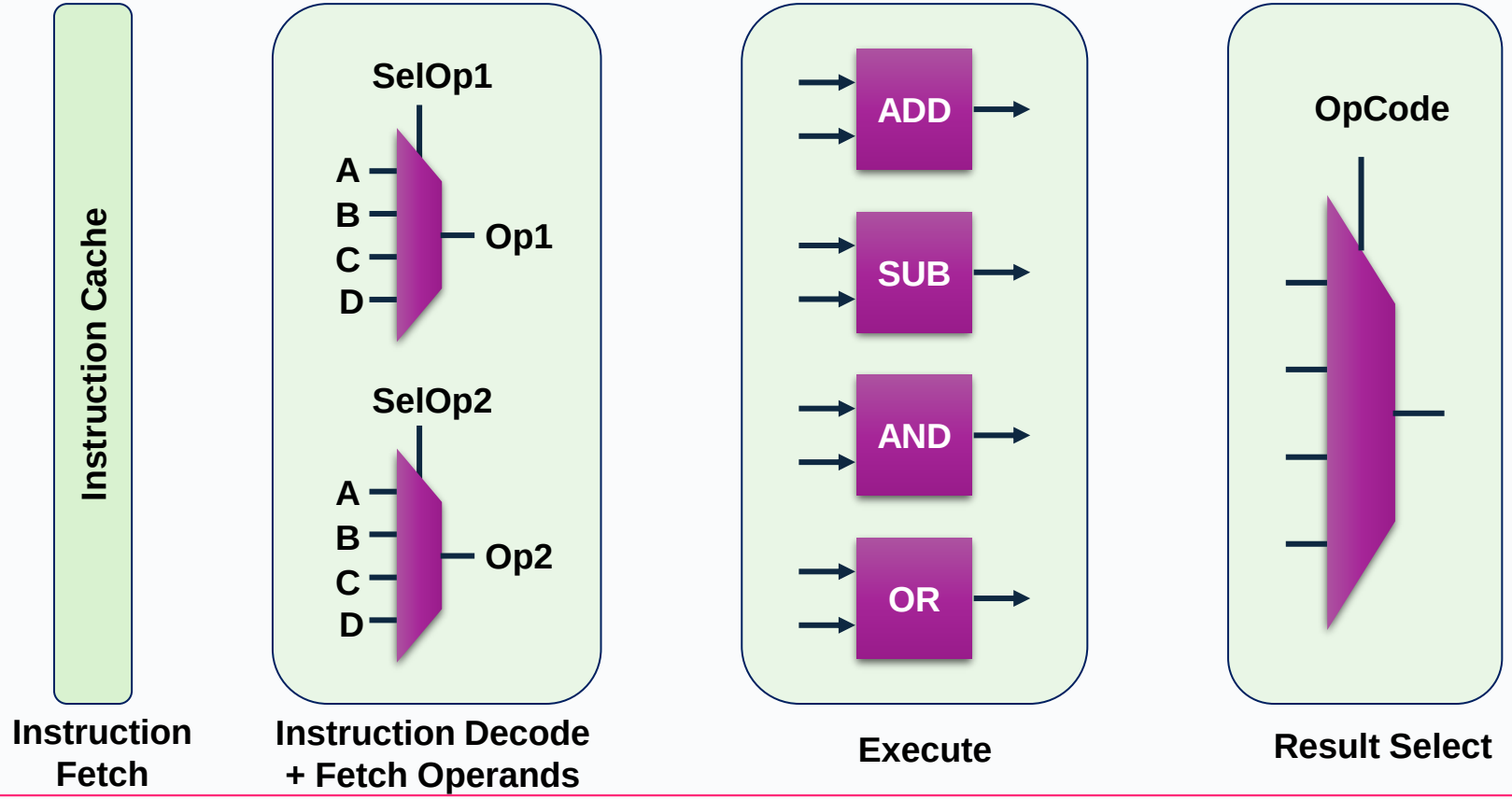


- Pipelining helps execute multiple tasks in parallel
- Improves throughput …

Source: Computer Organization and Design (RISC-V Edition), Patterson and Hennessy

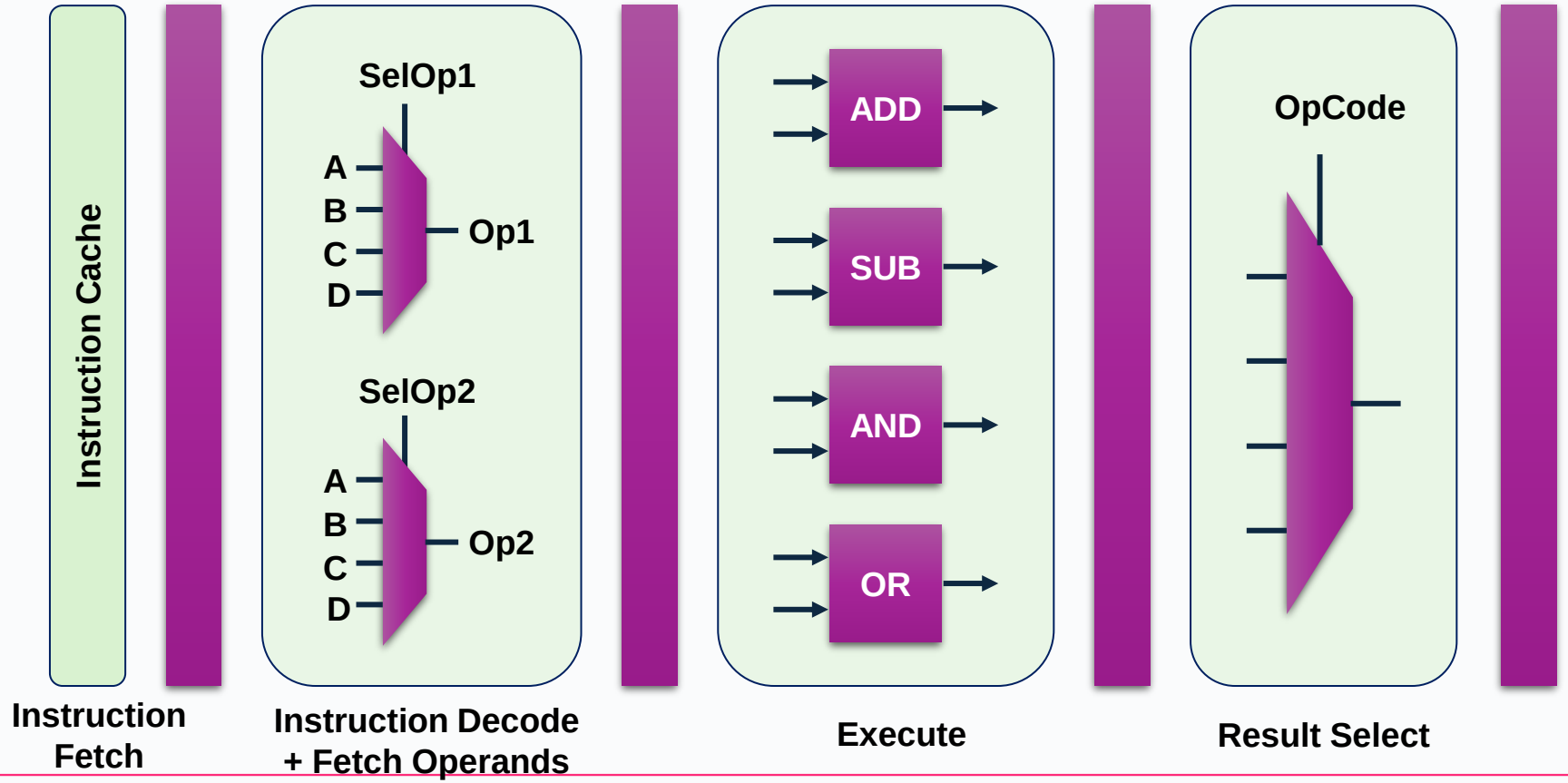
Pipelining Processor

- Without Pipelining



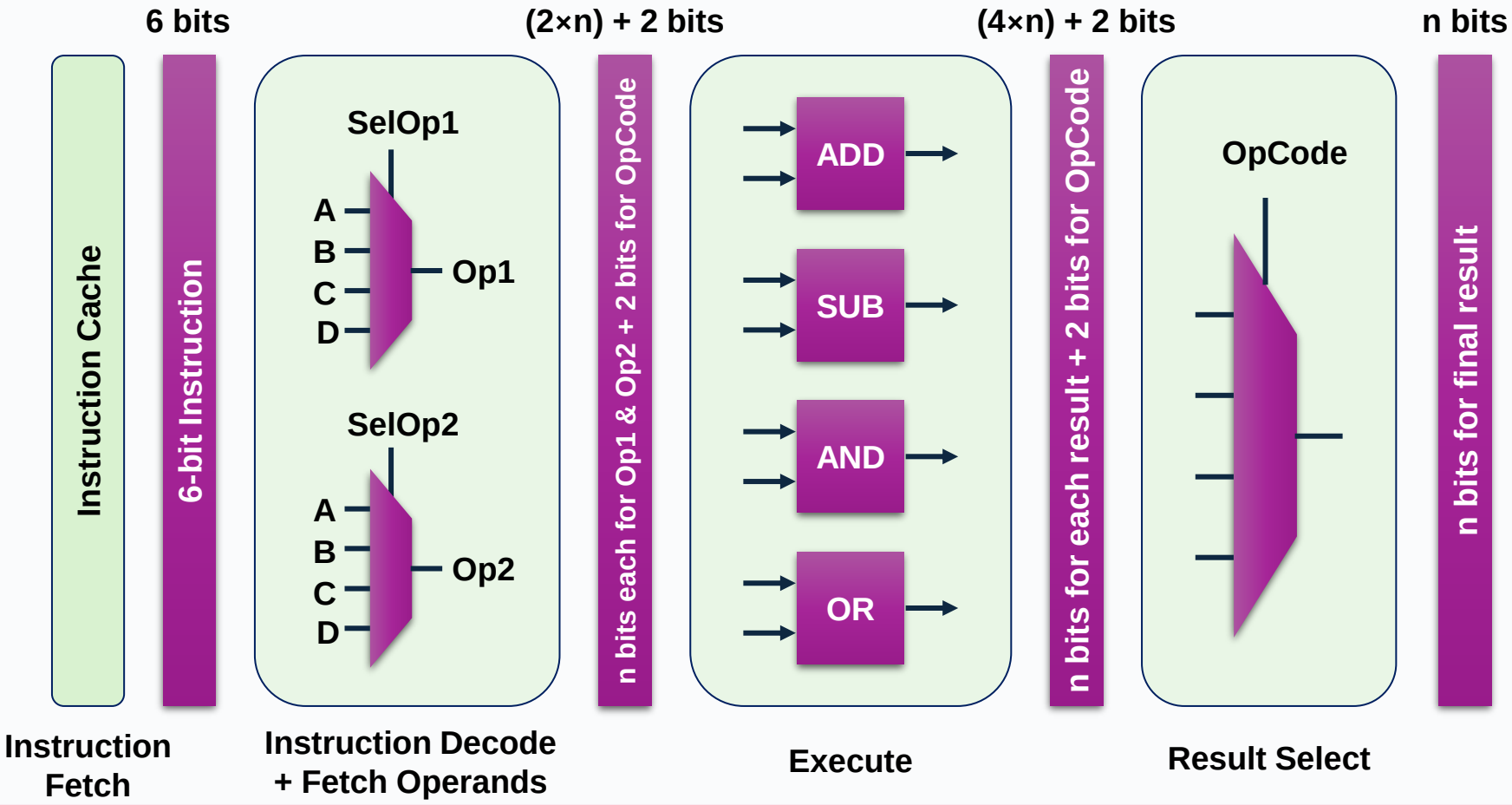
Pipelining Processor

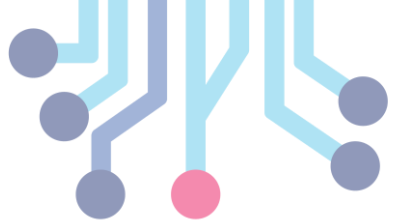
- Adding Pipeline Registers



Pipelining Processor

- Register Sizes





What's the critical path in pipelined implementation?

Critical Path in Pipelined Design

- It's reduced to that of subtractor
  - Assuming subtractor path is longer than that of multiplexer!
  - $Max\ Clock = \frac{1}{Subtractor\ Delay + t_S + t_{CLK2Q}}$
  - Much faster than the previous design!
  - As first instruction result (OpCode+Operands) goes to second pipelined register
    - A new instruction can be loaded from instruction cache to the first pipelined register!

Agenda

- From Transistors to Software
- Memory
- Pipelining
- **Register File**

In load-store architectures (all recent ones), data is loaded into registers inside CPU, which act as temporary storage, and after processing it is stored back in data memory…

Let’s add a Register File to our processor and modify the instruction set a bit …

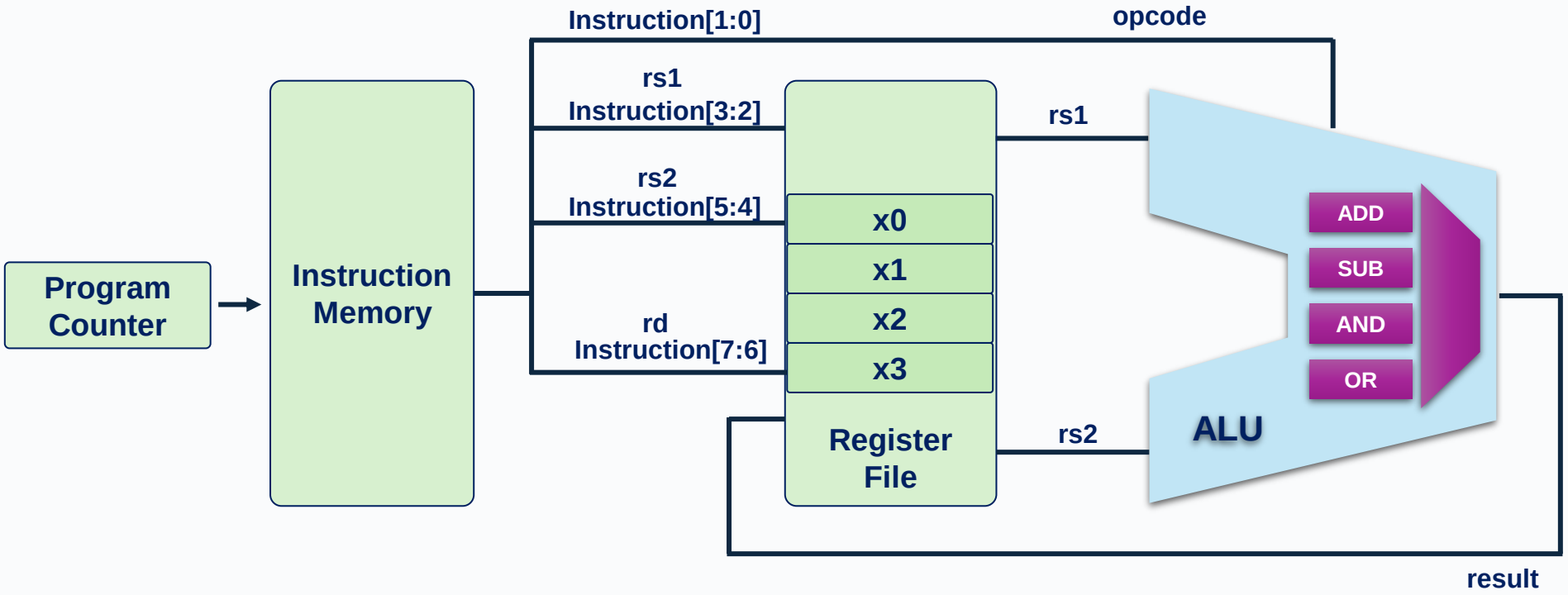
Architecture with Register File

- Let's assume there are four n-bit registers in processor named x0, x1, x2 and x3!
  - Let's assume x0 is hardwired to 0 value!
  - Two instruction bits select source operand 1 (rs1)
  - Two instruction bits select source operand 2 (rs2)
  - Two instruction bits select destination operand (rd) where the result form ALU is stored!
- Our instruction is now 8-bit!
  - Instruction[1:0] → opcode
  - Instruction [3:2] → rs1
  - Instruction [5:4] → rs2
  - Instruction [7:6] → rd

Architecture with Register File

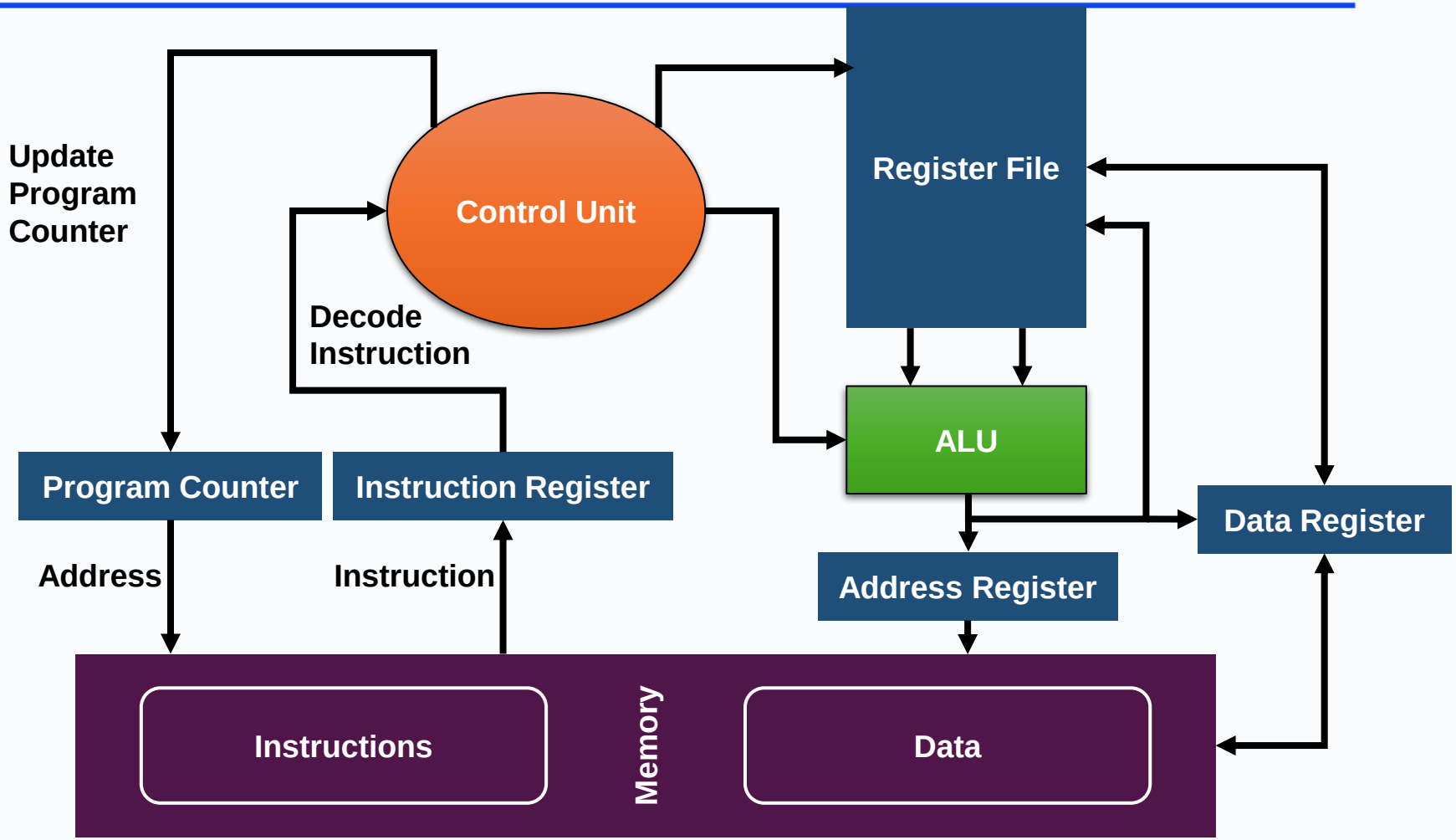
- Example
  - add x2, x1, x3
    - opcode = 00
    - rs1 = 01
    - rs2 = 11
    - rd = 10
    - Instruction = {10 11 01 00}
  - Similarly, we can have sub, and, or etc.

Microarchitecture with Register File



We will see how more complex processors are build based on same fundamental principles!

Main Components in a Processor



Thank You

NCDC
NUST
CHIP
DESIGN
CENTRE