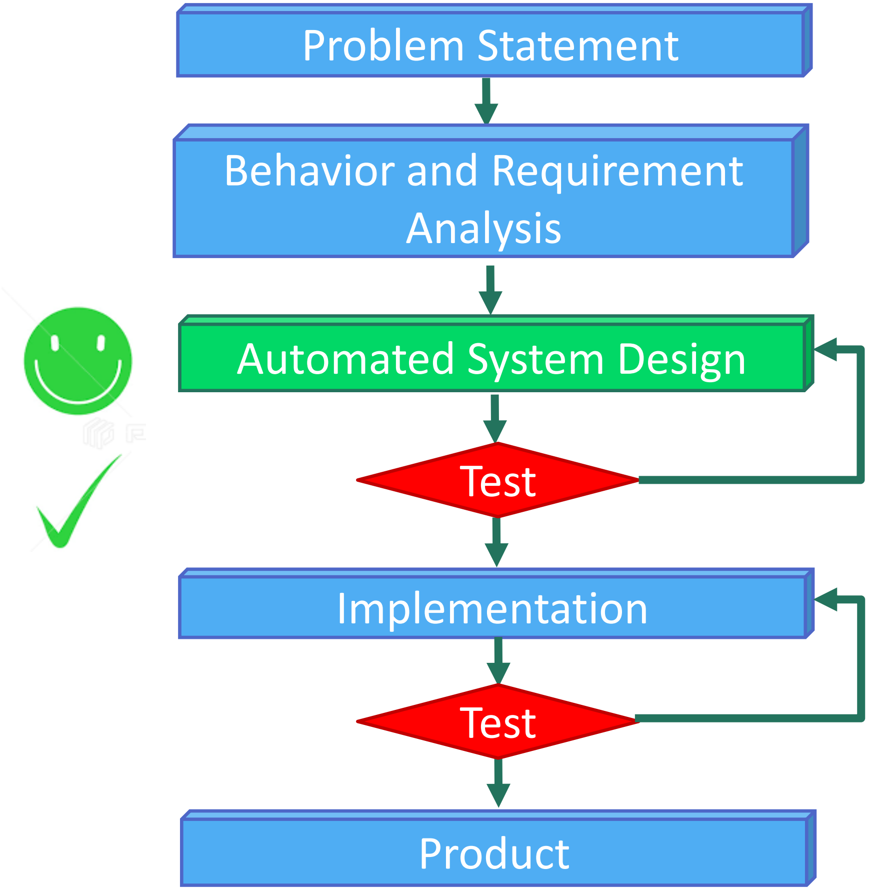
# Digital System Design

Hajar Falahati

hfalahati@ipm.ir
hfalahati@ce.sharif.edu

# Automated Design Flow



# Outline

- Abstraction Level

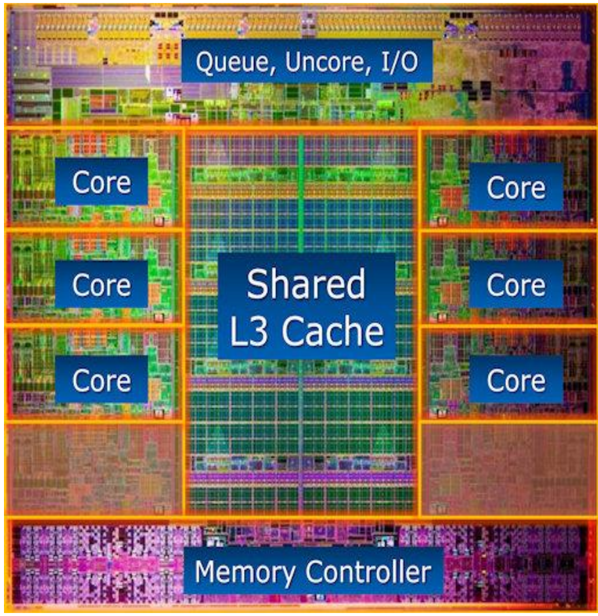
- Design Approach
  - **Top-down**
  - **Buttom-up**

- Modeling
  - **FSM**
  - **ASM**

# Design Complexity

# 2012: Intel Sandy Bridge-E

- 64-bit processor
- 4 cores, 8 threads
- 14-19 pipeline stages
- 3.6 GHz clock
- 2.27B transistor
- 32 nm



https://en.wikichip.org/wiki/intel/microarchitectures/sandy_bridge_(client)

https://techreport.com/review/21987/intel-core-i7-3960x-processor

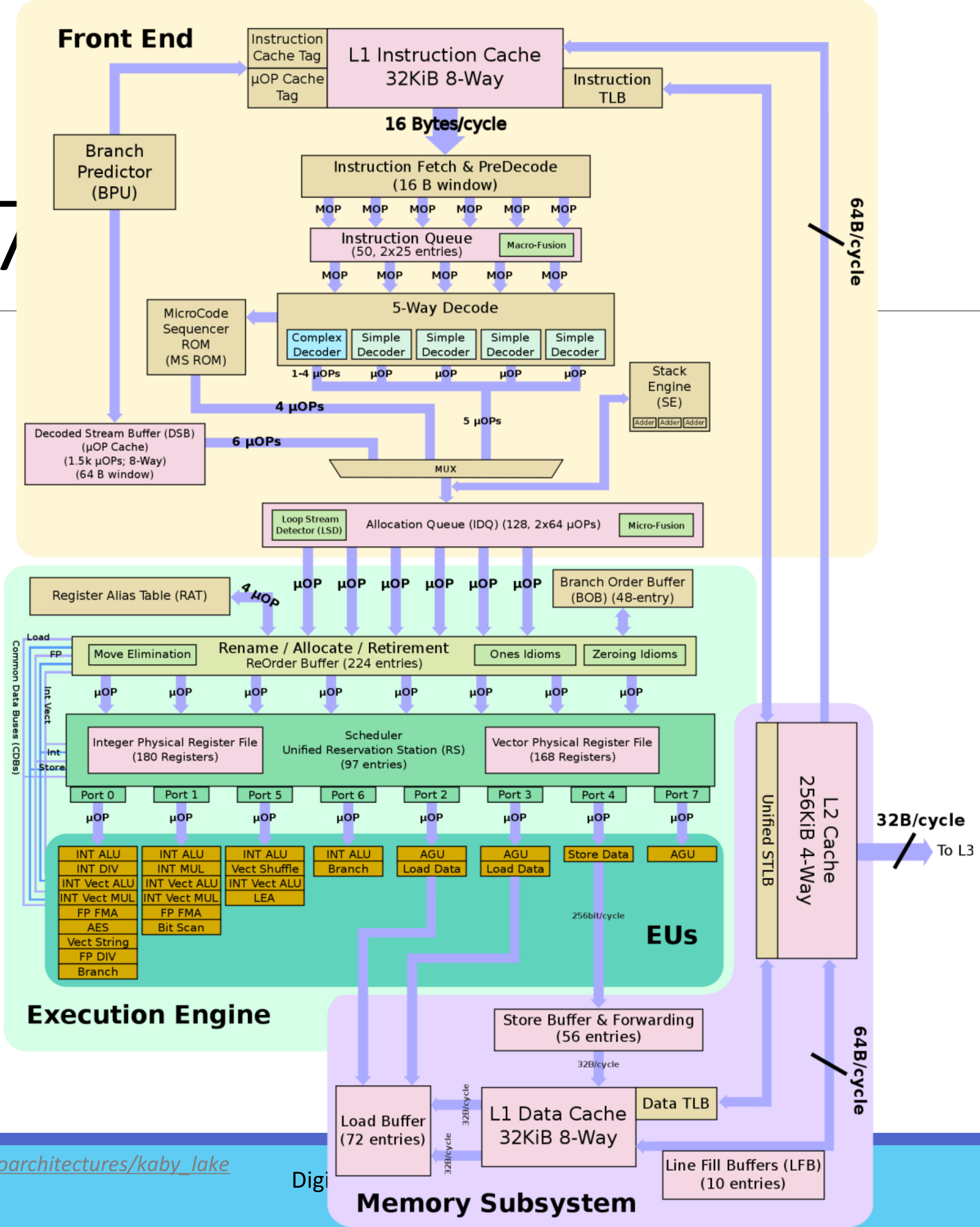
# 2017: Intel Kaby Lake

- 64-bit processor
- 4 cores, 8 threads
- 14-19 pipeline stages
- 3.9 GHz clock
- 1.75B transistor
- 14 nm

Intel Kaby Lake
intel CORE m7
intel CORE m7 vPro
intel CORE m5
intel CORE m5 vPro
intel CORE m3
intel CORE i3
intel CORE i5
intel CORE i5 vPro
intel CORE i7
intel CORE i7 vPro
Y-SERIES
2 in 1 Detachables, Tablets and Compute Stick
U-SERIES
Thin Light Notebooks, Portable AIO, Minis and Conference Room
H-SERIES
Ultimate Mobile Performance, Mobile Workstations
S-SERIES
Desktop Performance to Value, AIO and Minis

https://en.wikichip.org/wiki/intel/microarchitectures/kaby_lake

# 2017

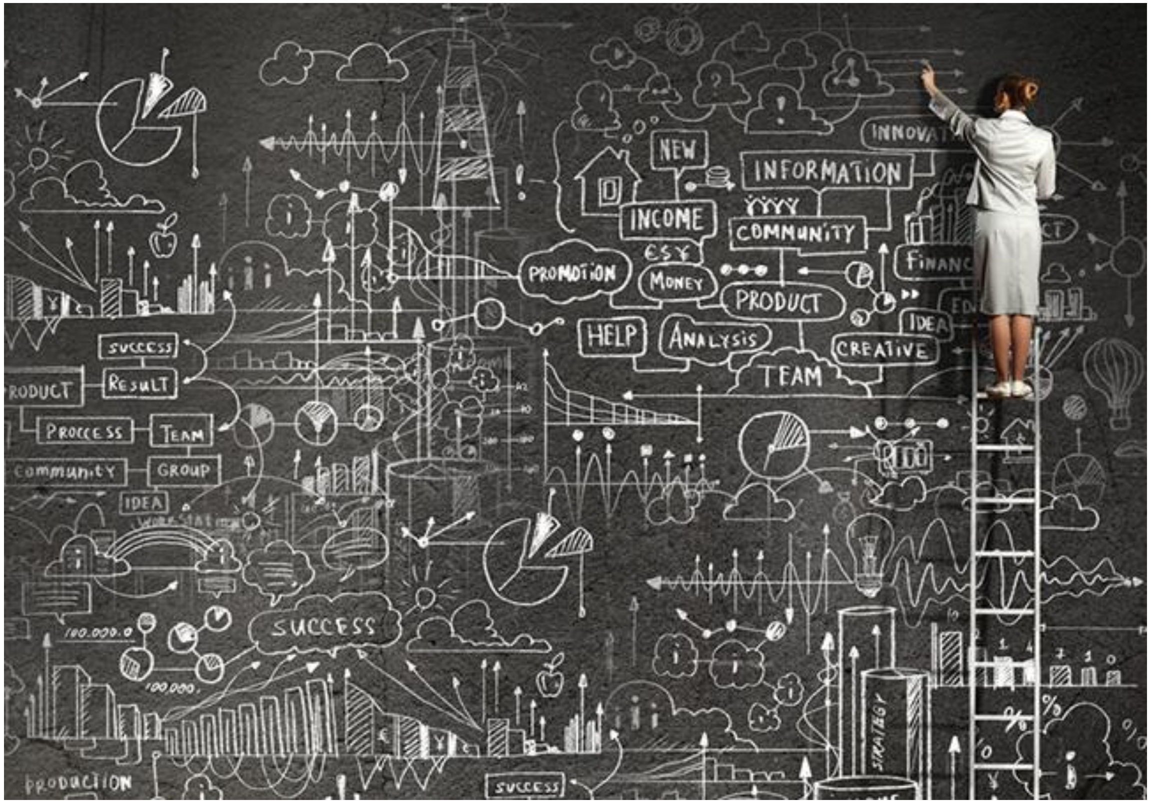


https://en.wikichip.org/wiki/intel/microarchitectures/kaby_lake

Digi

# Design Complexity

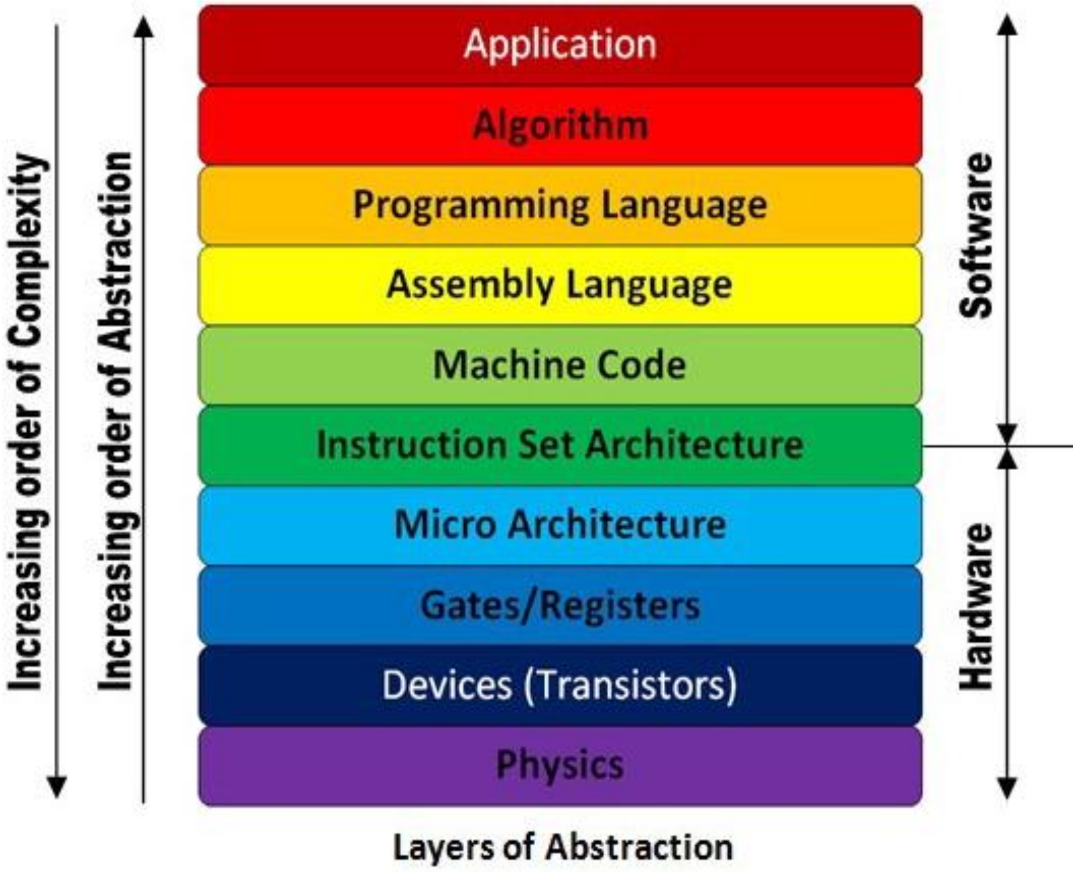
- How to handle this **complexity**?



# Abstraction Levels

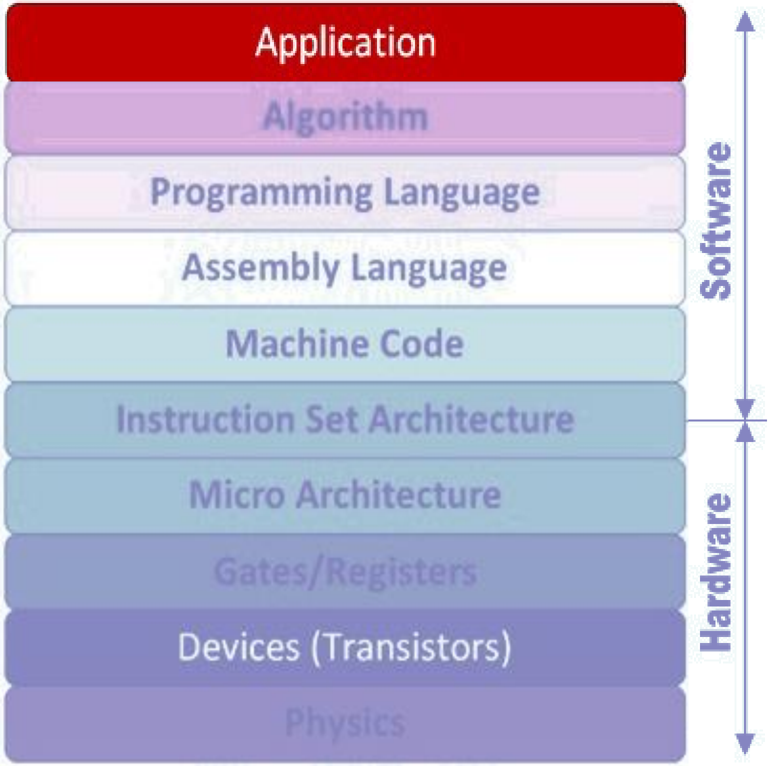
# Abstraction Levels

- To **comprehend** these complicated systems



# Abstraction Levels: 1

- To **comprehend** these complicated systems



Application

Algorithm

Programming Language

Assembly Language

Machine Code

Instruction Set Architecture

Micro Architecture

Gates/Registers

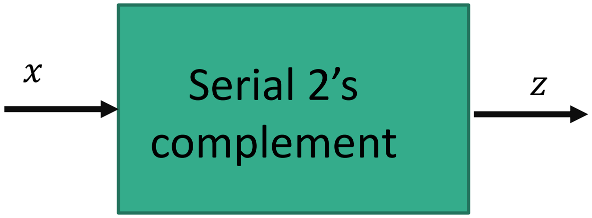
Devices (Transistors)

Physics

Software

Hardware

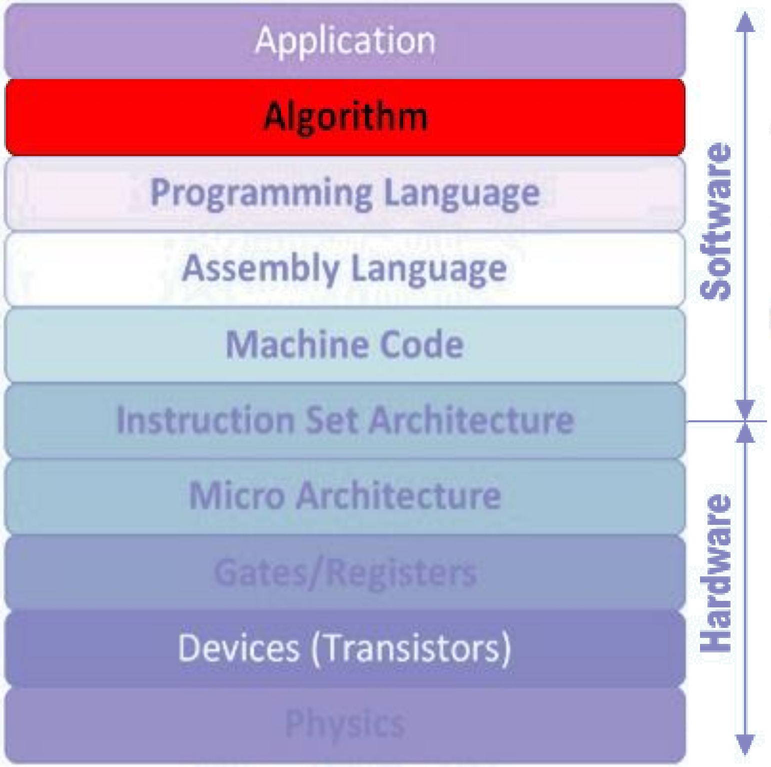
```
>"hello
world!"
```

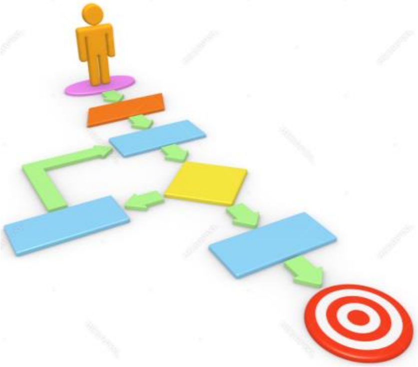


Serial 2’s complement

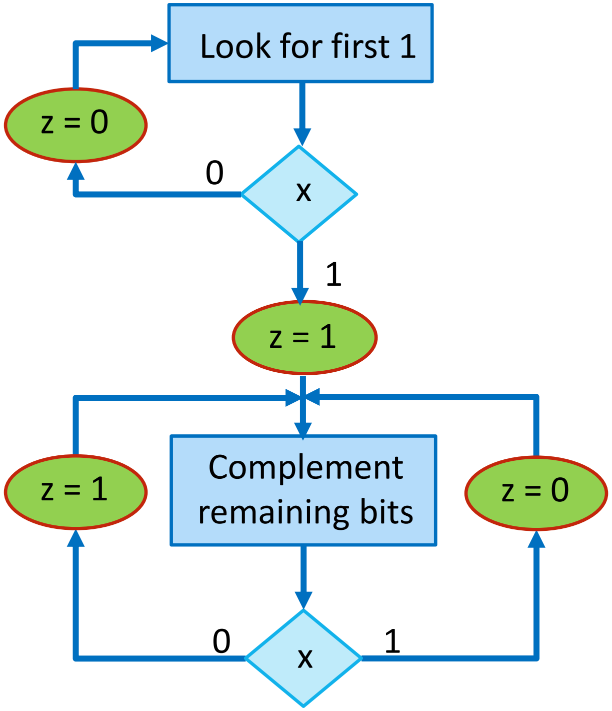
# Abstraction Levels: 2

- To comprehend these complicated systems



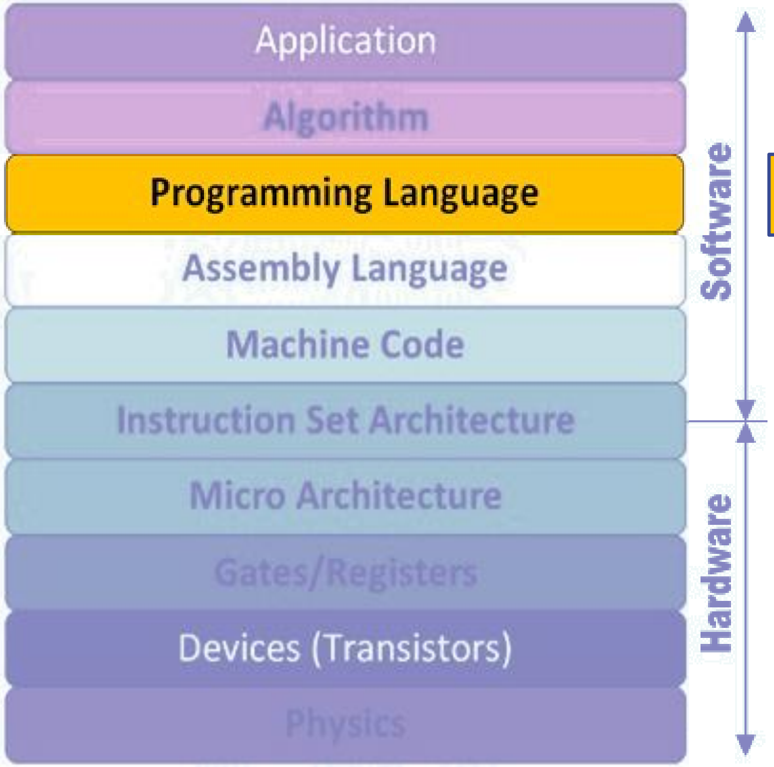


| t | x | z |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 2 | 1 | 1 |
| 3 | 1 | 0 |



# Abstraction Levels: 3

- To comprehend these complicated systems



Application

Algorithm

Programming Language

Assembly Language

Machine Code

Instruction Set Architecture

Micro Architecture

Gates/Registers

Devices (Transistors)

Physics

Software

Hardware

S2c.cpp

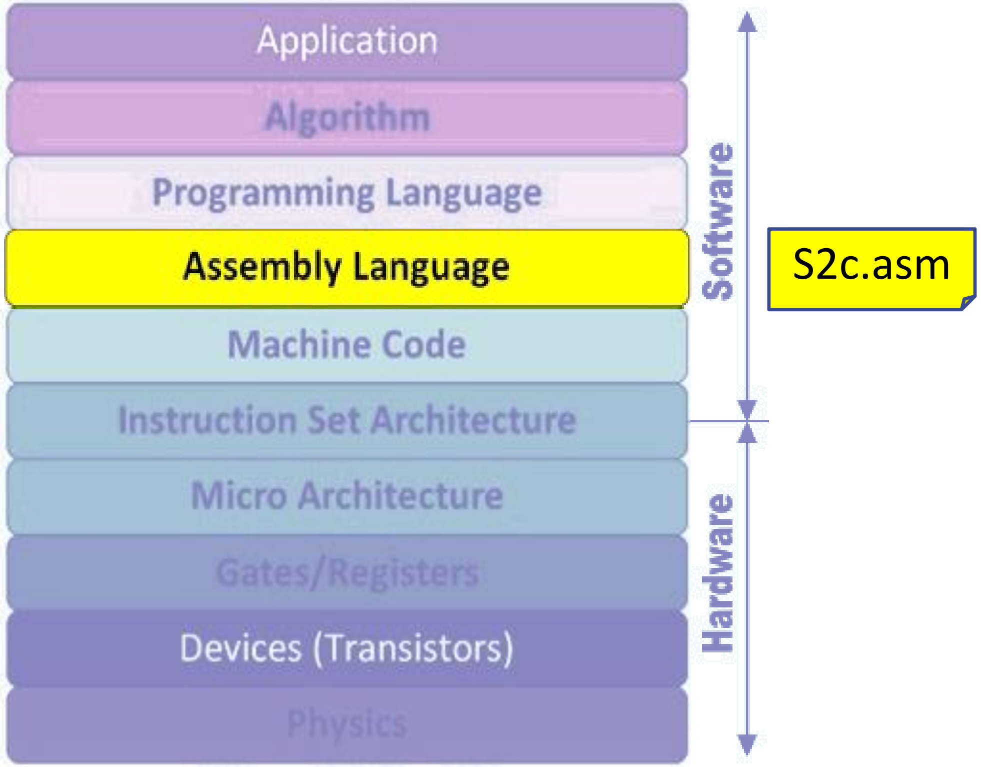
```
int main(){
    bool x,z, flag1;
    z=0;

    cin>>x;
     while(!x){
             z=0;
             cin>>x;
      }

     while(1){
       if(!x)   z =1;
       else     z=0;
      }
}
```

# Abstraction Levels: 4

- To **comprehend** these complicated systems



Application

Algorithm

Programming Language

Assembly Language

Machine Code

Instruction Set Architecture

Micro Architecture

Gates/Registers

Devices (Transistors)

Physics

Software

Hardware

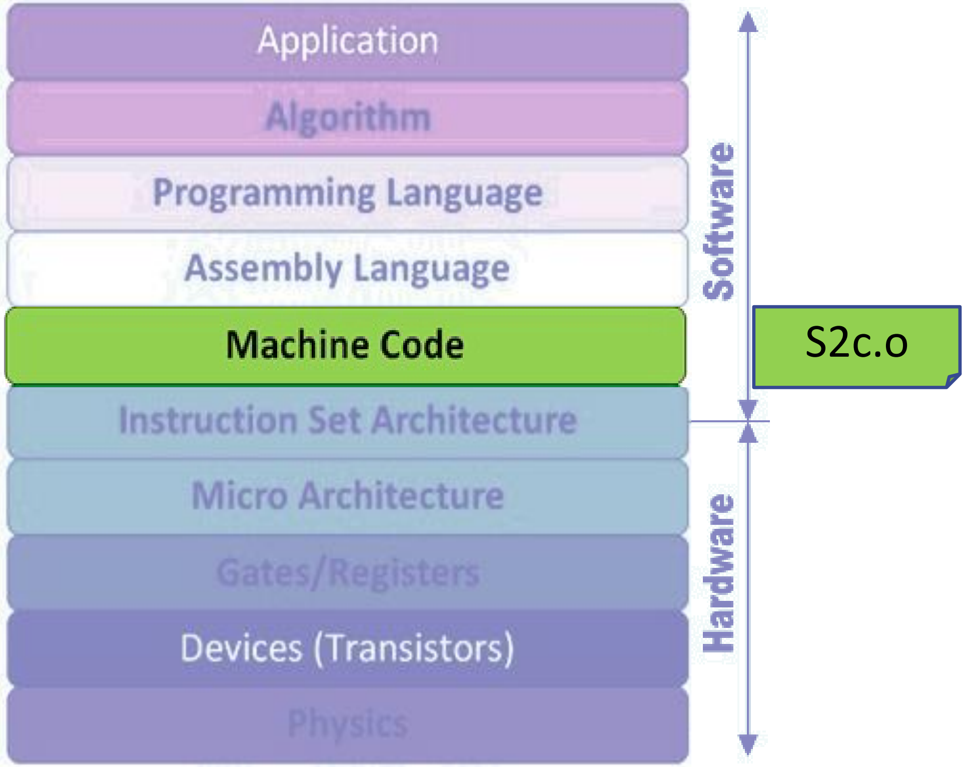
S2c.asm

```
          ORG 0

          add $t1,$zero,$zero    //t1 = z
Look1:
          addi $v0, ,$zero,5
          syscall
          add $t0, $v0,$zero
          beq $t0,$at, Find1
          add $t1,$zero,$zero
          j  Look1
Find1:
          addi $v0, $zero,5
          syscall
          add $t0, $v0,$zero
          beq  $t0,$at, Set0
          add  $t1,$zero,$at
          j   Find1
Set0:
          add $t1,$zero,$zero
          j  Find1
```

# Abstraction Levels: 5

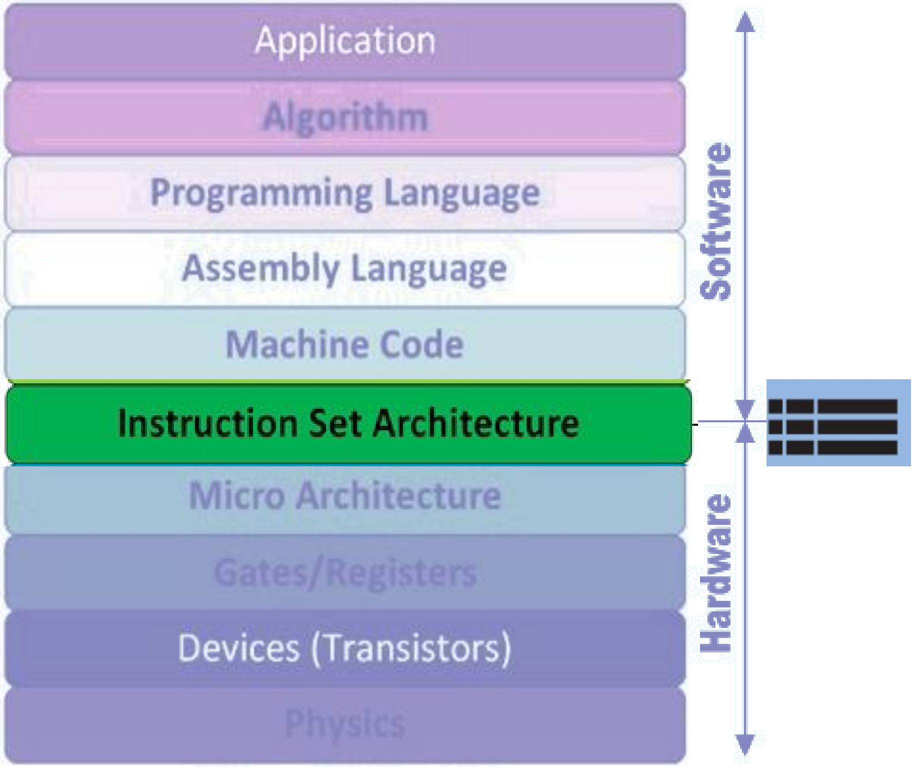
- To **comprehend** these complicated systems



```
00000000000000000100100000000000
00100000000000100000000000000101
00000000000000000000000000001100
00000000010000000100000000000000
00010001000000010000000000000111
00000000000000000100100000000000
00001000000000000000000000000001
00100000000000100000000000000101
00000000000000000000000000001100
00000000010000000100000000000000
00010001000000010000000000001101
00000000000000010100100000000001
00001000000000000000000000000111
00000000000000000100100000000000
00001000000000000000000000000111
```

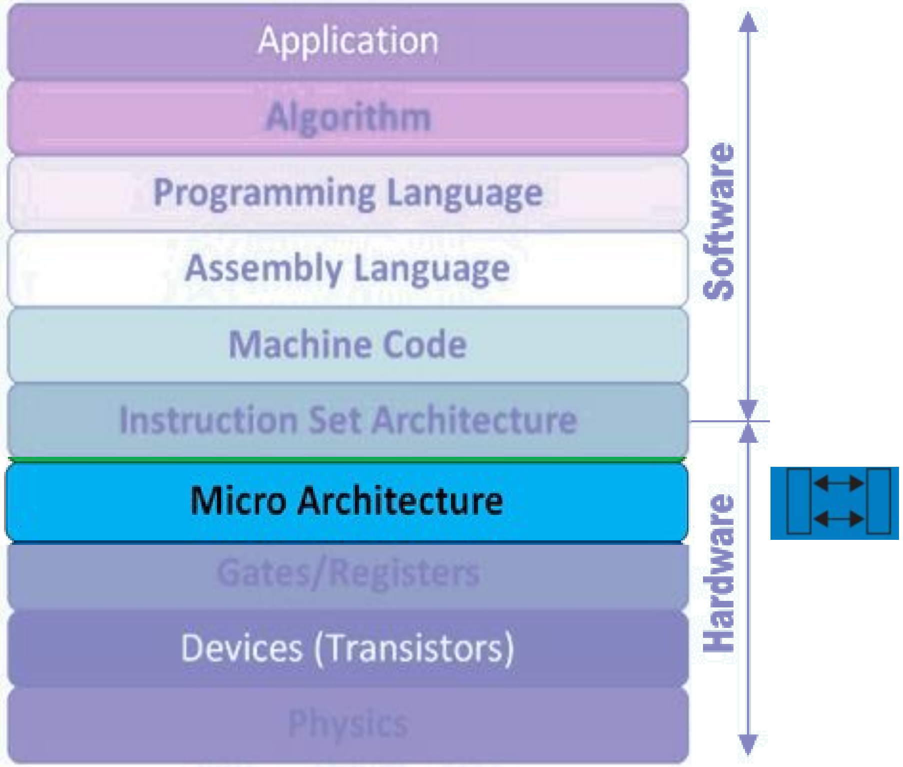
# Abstraction Levels: 6

- To **comprehend** these complicated systems



# Abstraction Levels: 7

- To **comprehend** these complicated systems



Application

Algorithm

Programming Language

Assembly Language

Machine Code

Instruction Set Architecture

**Micro Architecture**

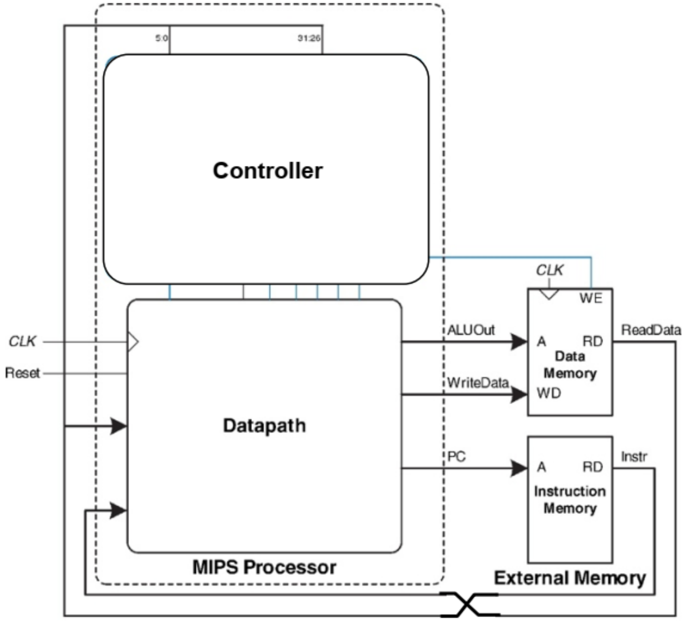
Gates/Registers

Devices (Transistors)

Physics

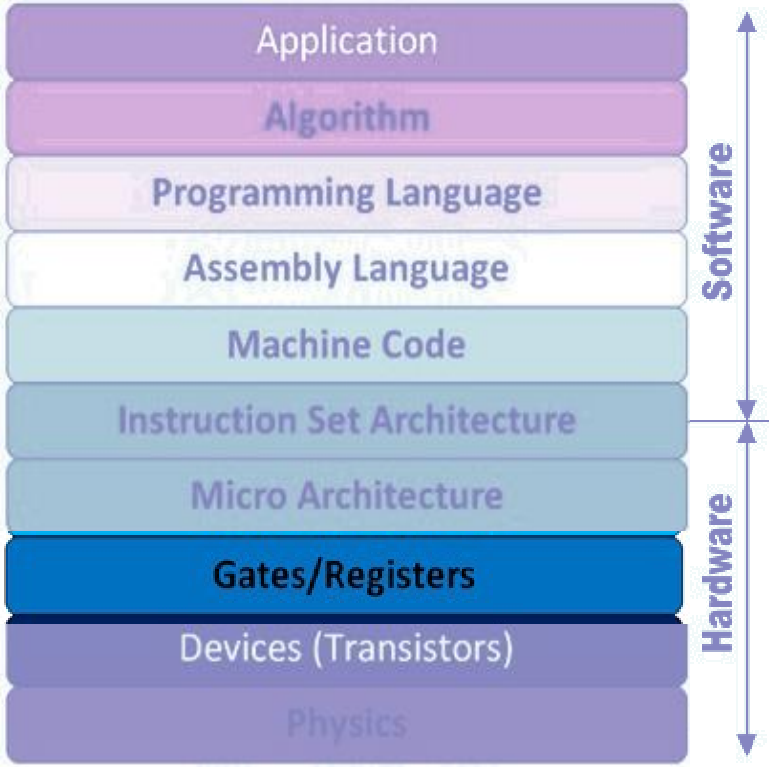
Software

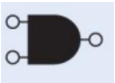
Hardware

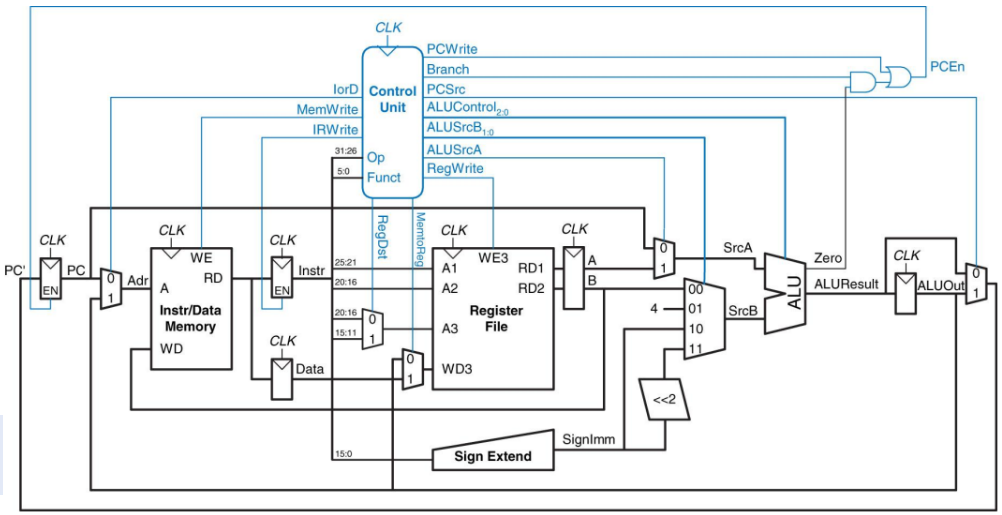


# Abstraction Levels: 8

- To **comprehend** these **complicated** **systems**

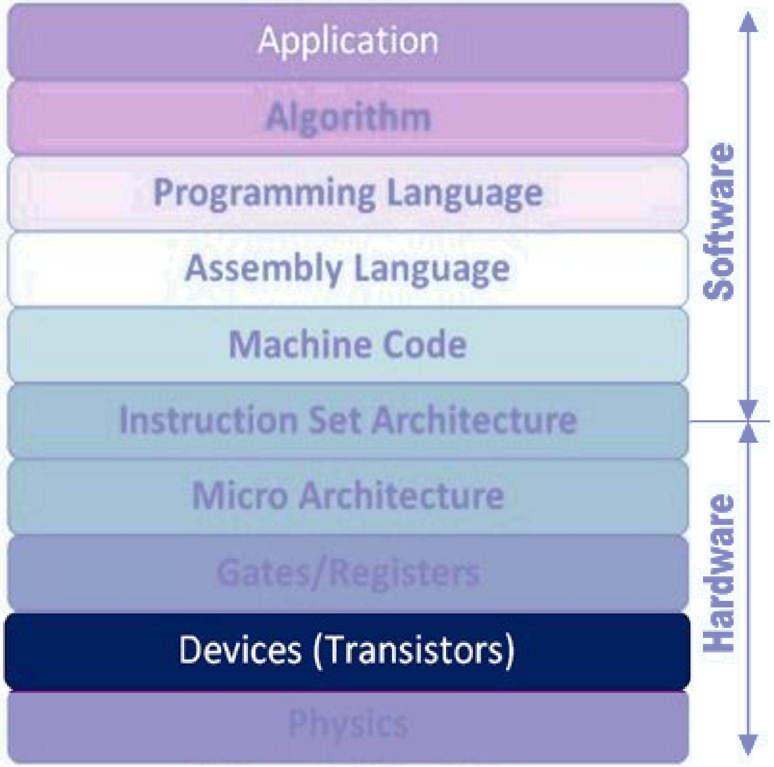


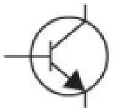


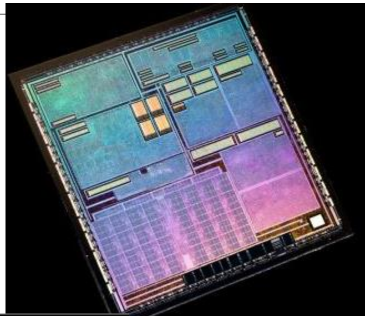


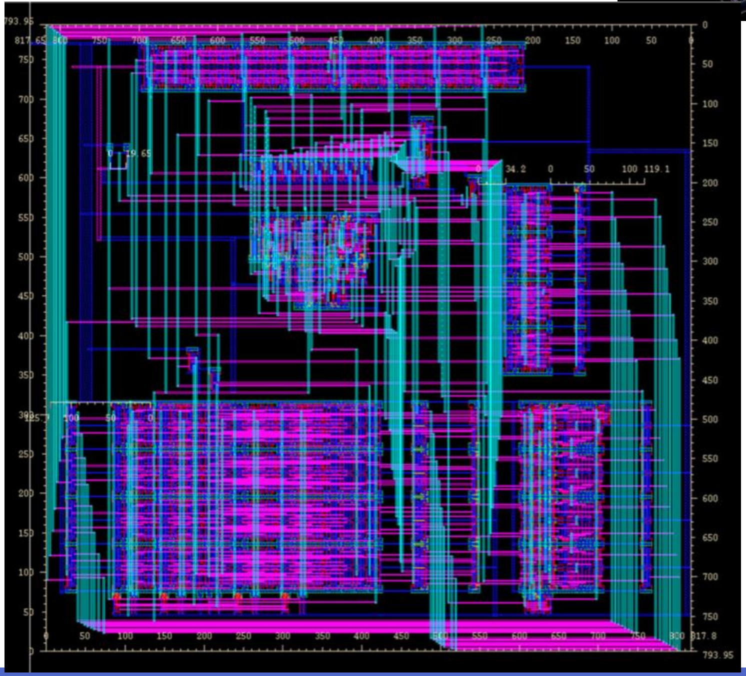
# Abstraction Levels: 9

- To **comprehend** these complicated systems



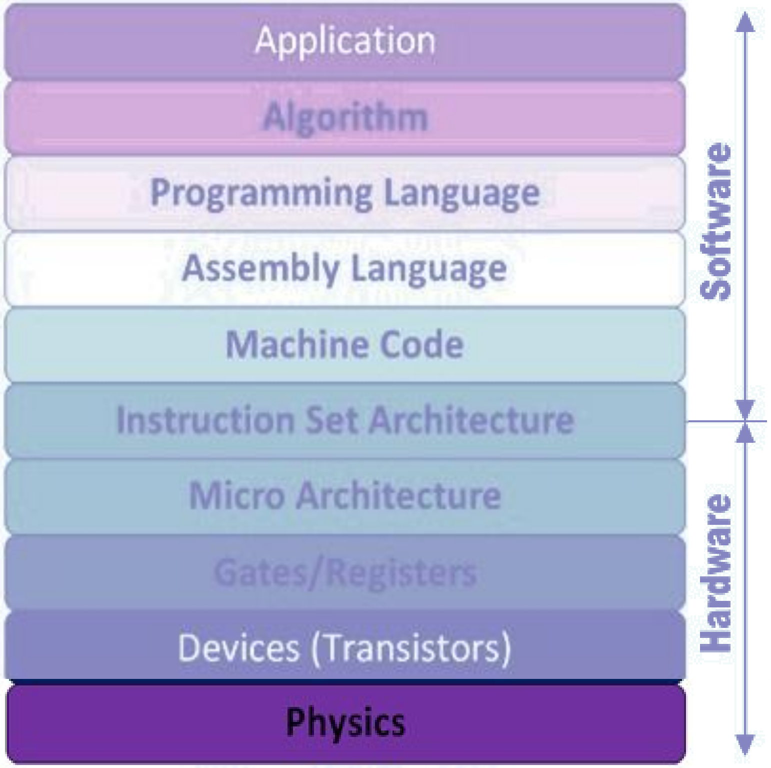






# Abstraction Levels: 10

- To comprehend these complicated systems



Application

Algorithm

Programming Language

Assembly Language

Machine Code

Instruction Set Architecture

Micro Architecture

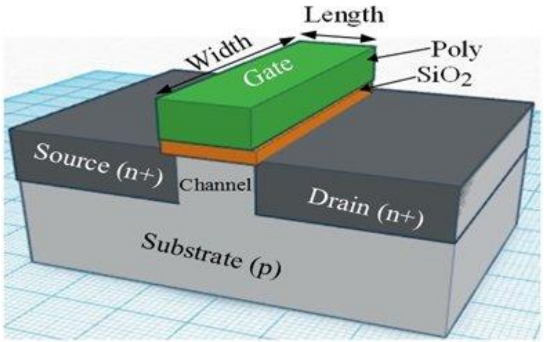
Gates/Registers

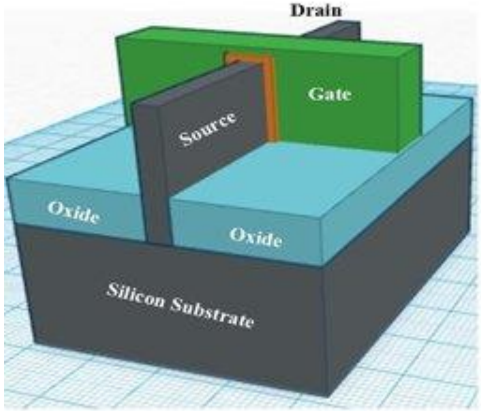
Devices (Transistors)

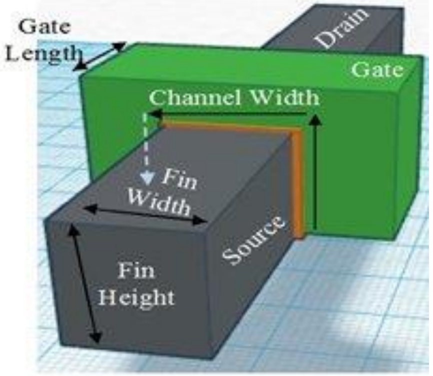
Physics

Software

Hardware





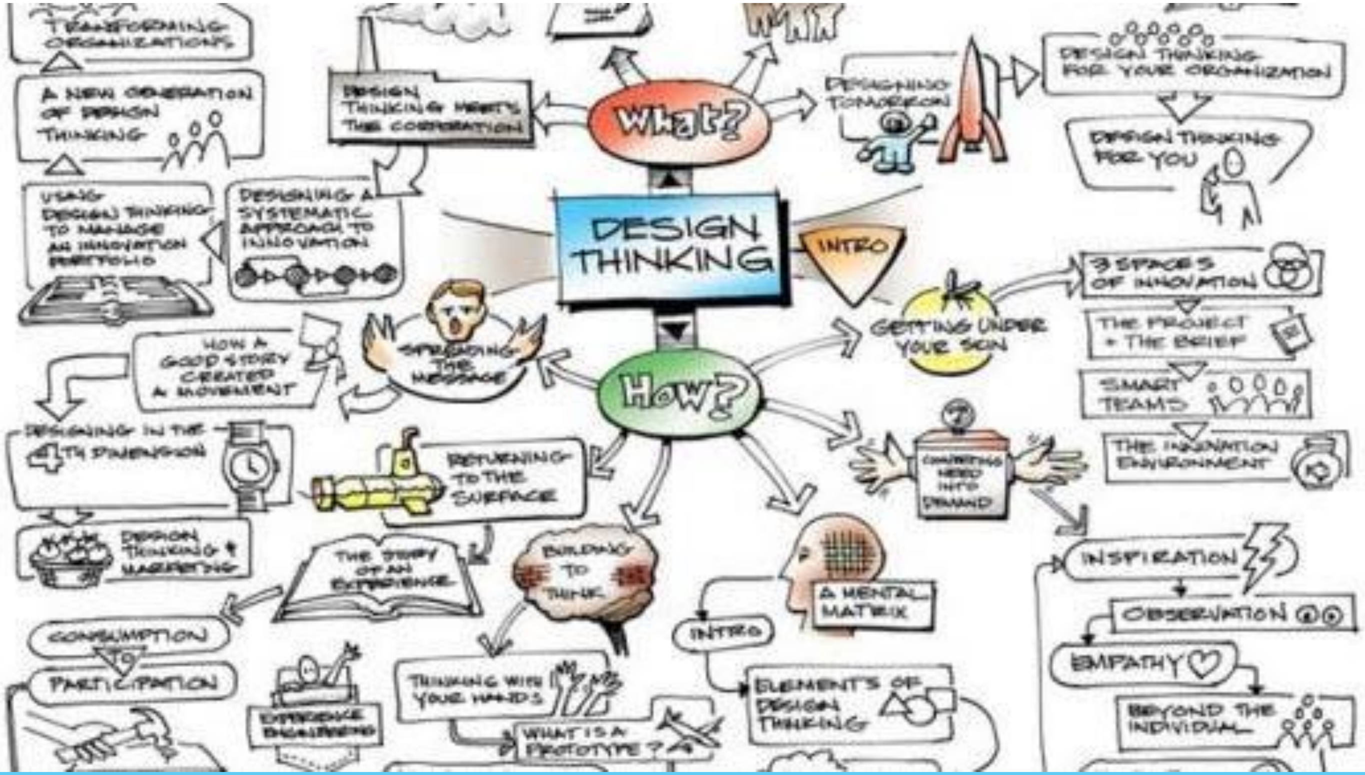


# Design Approach

# Design Approach

- Decide how to organize your project

- Organizational paradigms
  - **Top-Down**
  - **Buttom-Up**


DESIGN THINKING
What?
How?
INTRO
TRANSFORMING ORGANIZATIONS
A NEW GENERATION OF DESIGN THINKING
DESIGN THINKING MEETS THE CORPORATION
USING DESIGN THINKING TO MANAGE AN INNOVATION PORTFOLIO
DESIGNING A SYSTEMATIC APPROACH TO INNOVATION
HOW A GOOD STORY CREATED A MOVEMENT
SPREADING THE MESSAGE
DESIGNING IN THE 4TH DIMENSION
RETURNING TO THE SURFACE
DESIGN THINKING + MARKETING
THE STORY OF AN EXPERIENCE
BUILDING TO THINK
CONSUMPTION TO PARTICIPATION
EXPERIENCE ENGINEERING
THINKING WITH YOUR HANDS
WHAT IS A PROTOTYPE?
DESIGNING TOMORROW
DESIGN THINKING FOR YOUR ORGANIZATION
DESIGN THINKING FOR YOU
GETTING UNDER YOUR SKIN
3 SPACES OF INNOVATION
THE PROJECT + THE BRIEF
SMART TEAMS
THE INNOVATION ENVIRONMENT
CONVERTING NEED INTO DEMAND
A MENTAL MATRIX
INTRO
ELEMENTS OF DESIGN THINKING
INSPIRATION
OBSERVATION
EMPATHY
BEYOND THE INDIVIDUAL

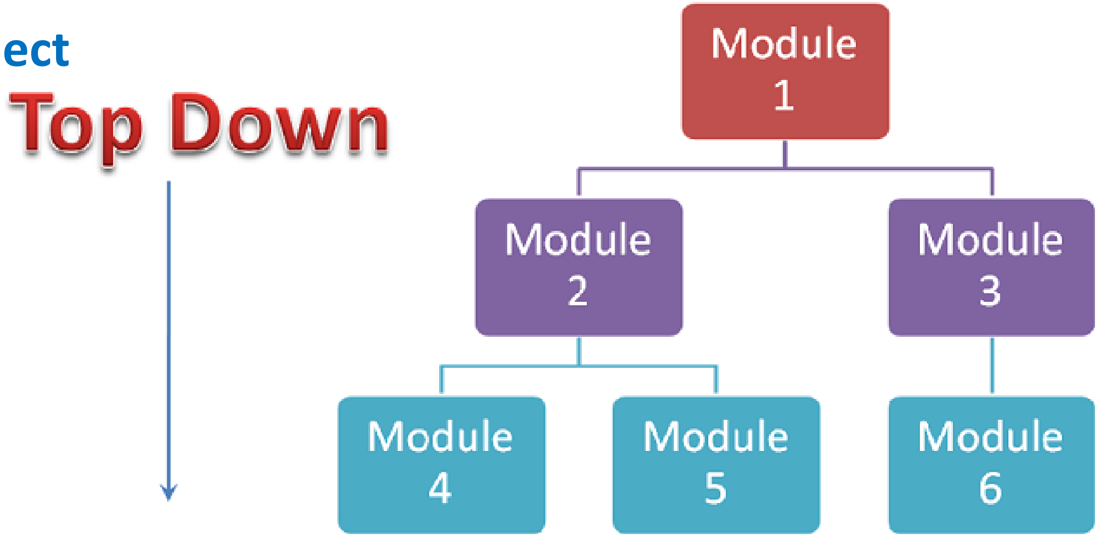
# Top-Down

- To gain **insights** into **its** **compositional sub-systems**
  - Reverse engineering fashion

- **Extensive planning** and **research phase**
  - -> Leads to development of a product

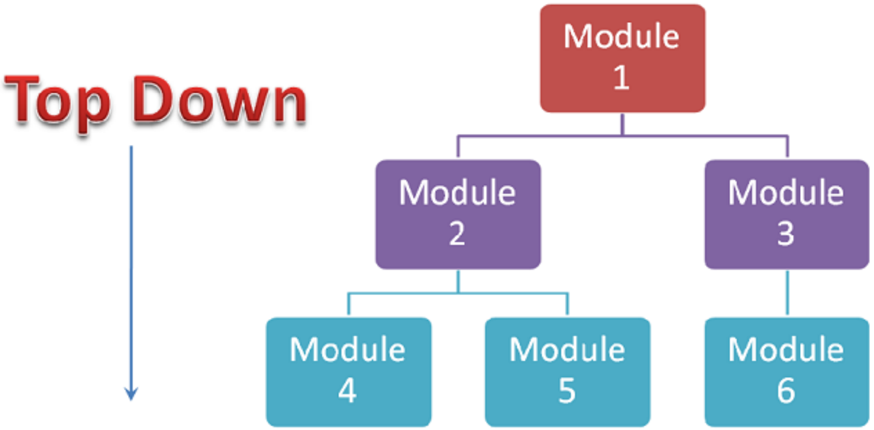
- **Structural control of a project**

- **More straightforward**



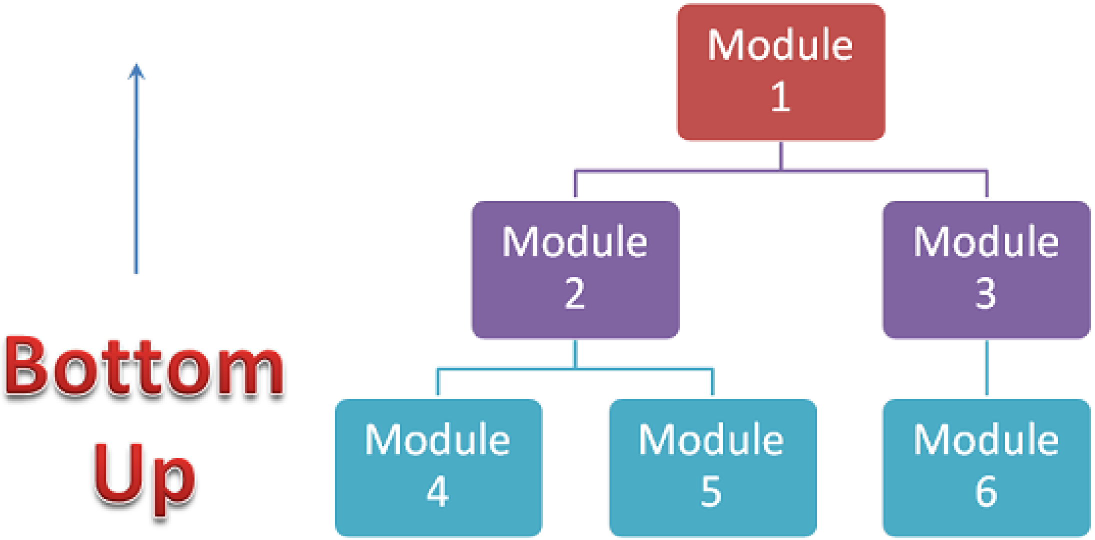
# Top-Down: Steps

1. Breaking down of a system
2. Formulating an overview of the system
3. Specifying any first-level subsystem
   - Not in detail!
4. Refining any subsystem in greater detail
   - Sometimes in many additional subsystem levels
5. Go to step 1, until the entire specification is reduced to base elements.



# Buttom-Up

- Piecing together of systems to give rise to more complex systems
- More **optimized**
- More **realistic**



# Hybrid Approach

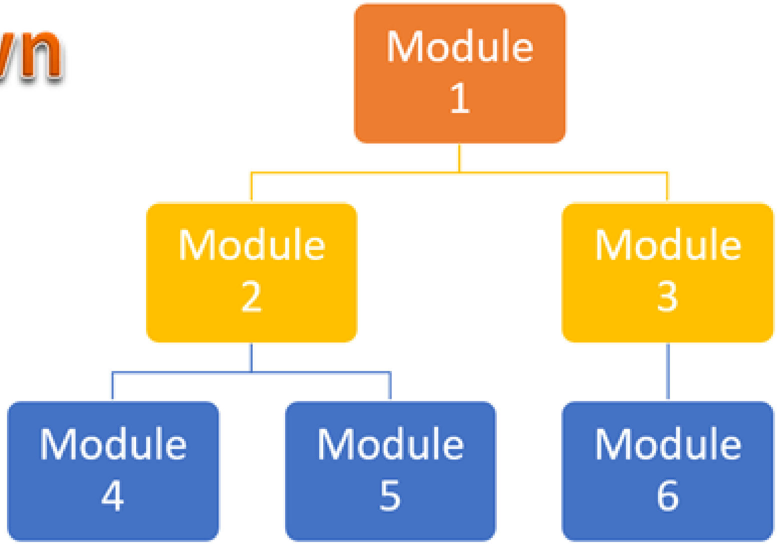
- Design architecture
  - Top level block specification
  - More straightforward

Top Down

Hybrid

Bottom Up

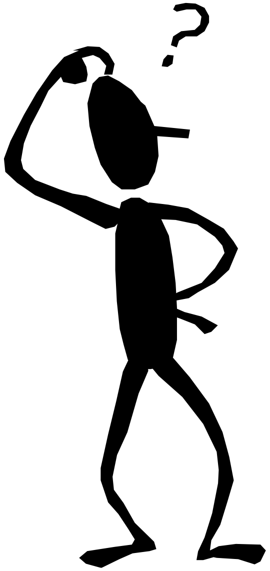
- Logic designer
  - Structured design
  - Breaks up the functionality
    - Blocks and sub blocks



- Circuit designer
  - At the same time
  - Optimizes leaf-level cells

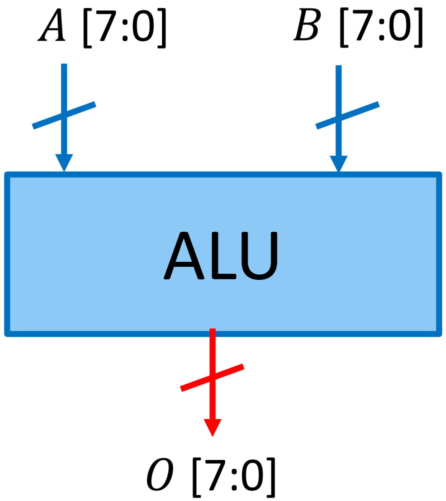
# Simple ALU

- Design an 8-bit ALU?!



# Step1: Specification: User

- Input
  - Two 8-bit data
- Output
  - A 8-bit data
- Behavior
  - Arithmetic (addition, subtract, maximum, minimum)
  - Logical (AND, OR, XOR, NOT A, NOT B)



# Step1: Specification: More Detail

- Input
  - Two 8-bit data; A, B
  - A 3-bit function code; F
  - A 1-bit invertor; Inv
- Output
  - A 8-bit data; O
  - 1-bit Carry, C
- Behavior
  - Arithmetic (addition, subtract, maximum, minimum)
  - Logical (AND, OR, XOR, NOT A, NOT B)

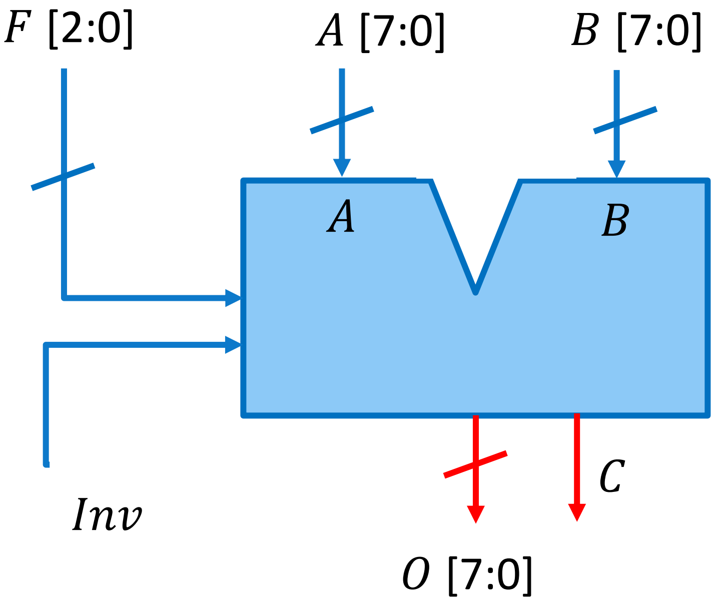
| Operation | F[2:0] |
|---|---|
| Addition (ADD) | 000 |
| Subtractor (SUB) | 001 |
| Maximum (MAX) | 010 |
| Minimum (MIN) | 011 |
| Bitwise AND (AND) | 100 |
| Bitwise OR (OR) | 101 |
| Bitwise XOR (XOR) | 110 |
| Bitwise NOT (NOT) | 111 |

| Invertor Input | Operation |
|---|---|
| 0 | NOT A |
| 1 | NOT B |

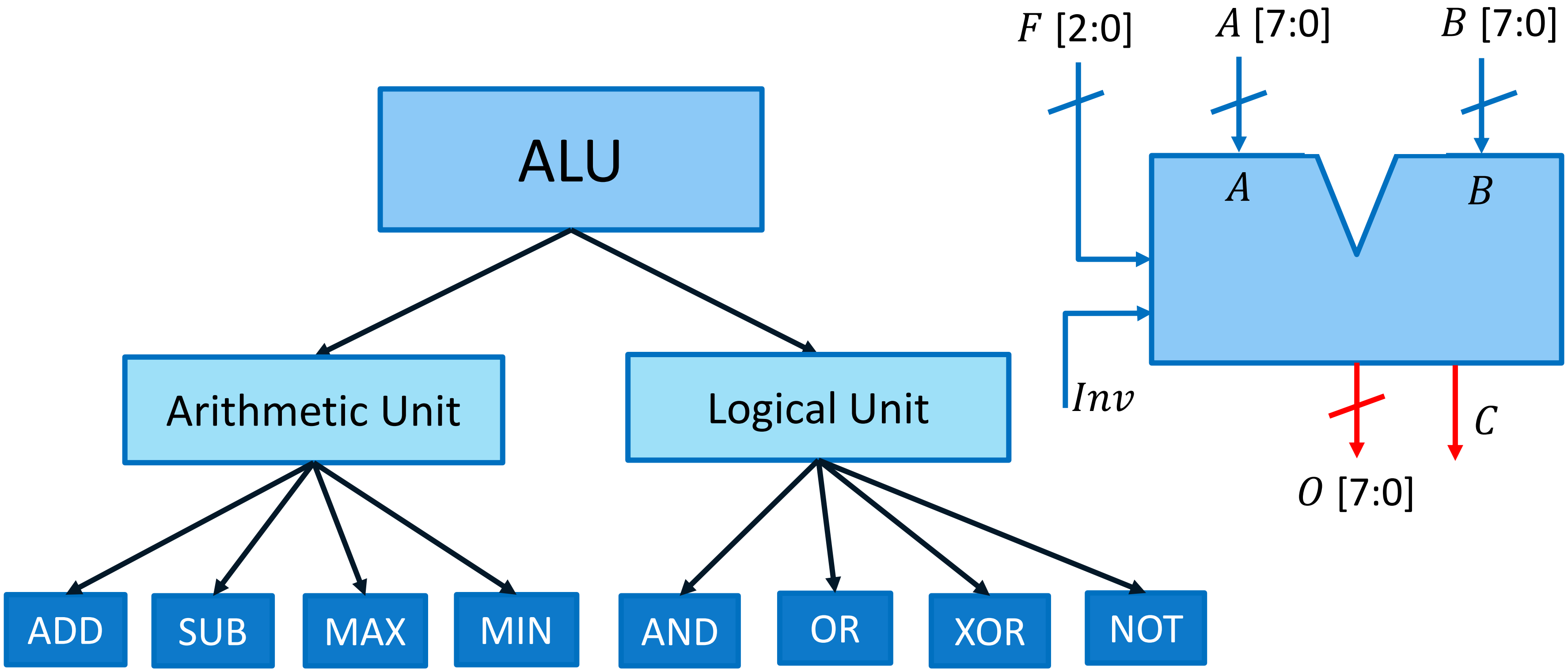
# Step2: Schematic

| Operation | F[2:0] |
|---|---|
| Addition (ADD) | 000 |
| Subtractor (SUB) | 001 |
| Maximum (MAX) | 010 |
| Minimum (MIN) | 011 |
| Bitwise AND (AND) | 100 |
| Bitwise OR (OR) | 101 |
| Bitwise XOR (XOR) | 110 |
| Bitwise NOT (NOT) | 111 |

| Invertor Input | Operation |
|---|---|
| 0 | NOT A |
| 1 | NOT B |

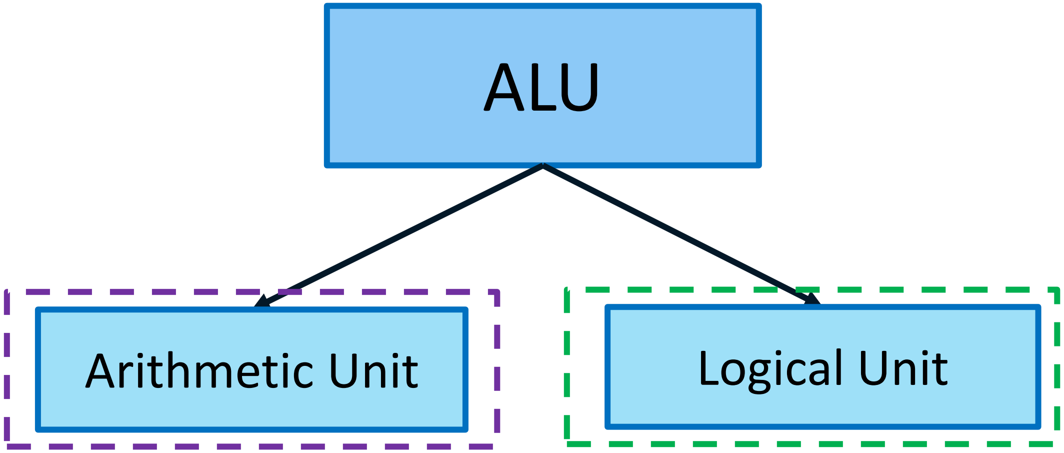


# Step3: Top-Down Design Approach

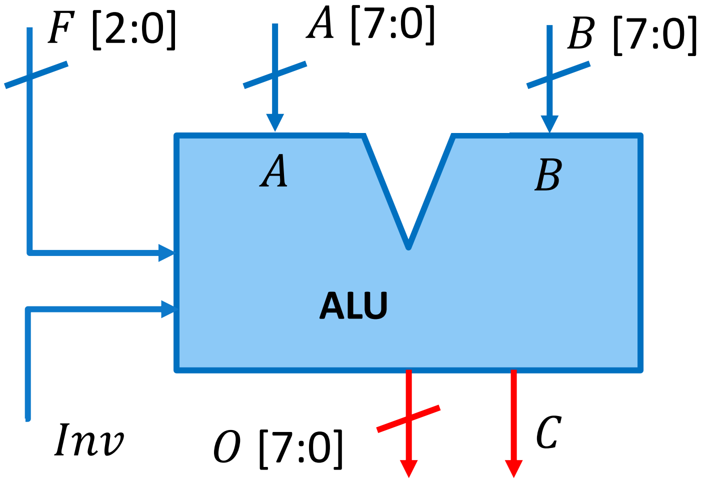


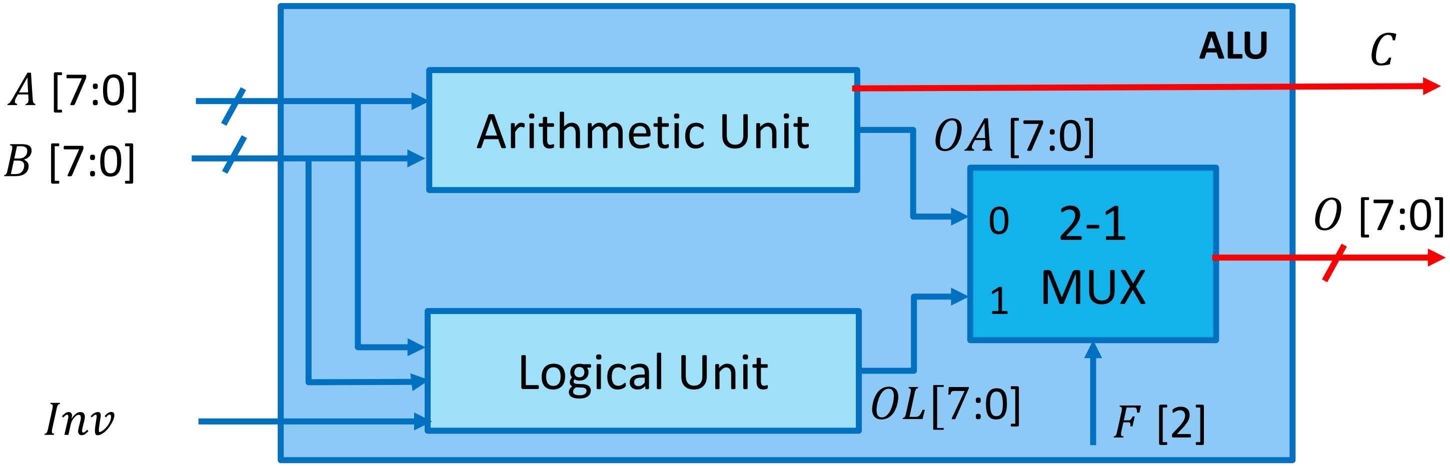
# Step4: Design: Level1

| Operation | Ctrl[2:0] |
|---|---|
| Addition (ADD) | 000 |
| Subtractor (SUB) | 001 |
| Maximum (MAX) | 010 |
| Minimum (MIN) | 011 |
| Bitwise AND (AND) | 100 |
| Bitwise OR (OR) | 101 |
| Bitwise XOR (XOR) | 110 |
| Bitwise NOT (NOT) | 111 |

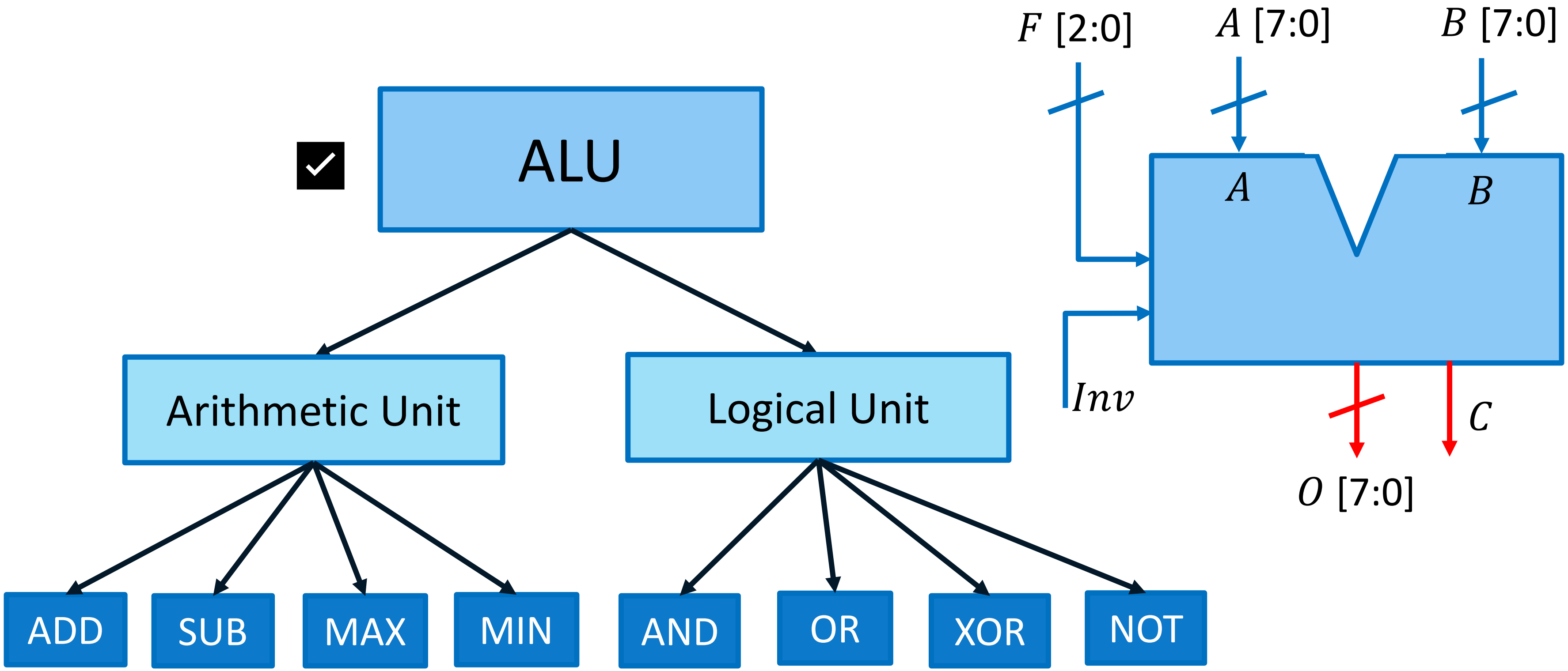


# Step4: Design: Level 1





# Step4: Level 1: DONE



☑ ALU

Arithmetic Unit

Logical Unit

ADD

SUB

MAX

MIN

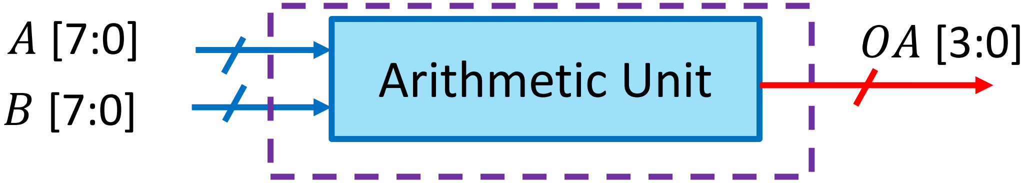
AND

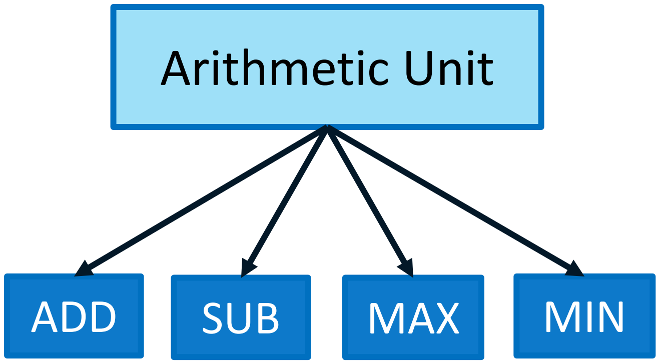
OR

XOR

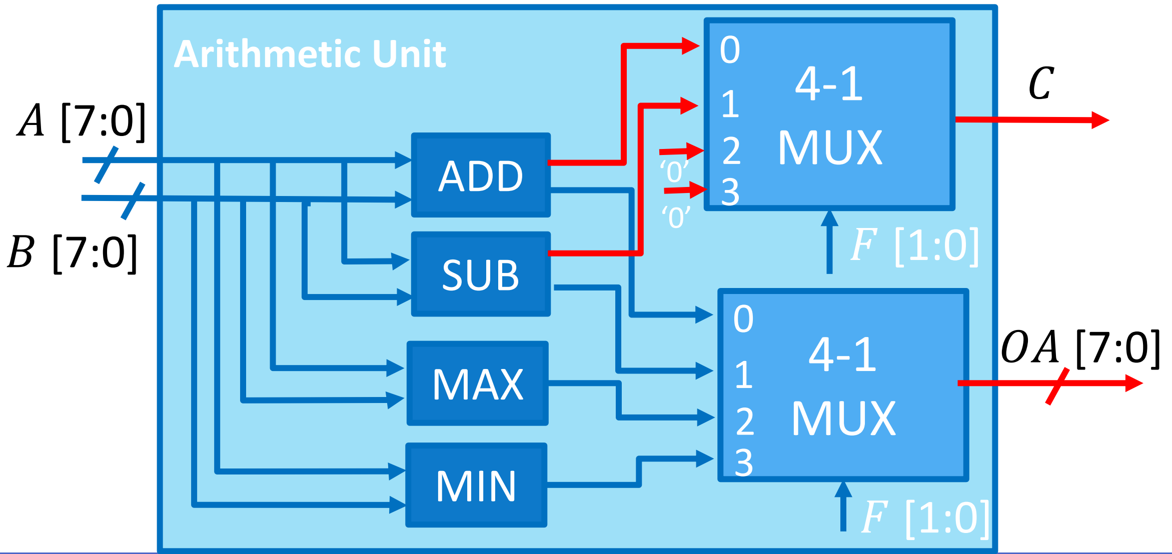
NOT

# Step4: Design: Level 2

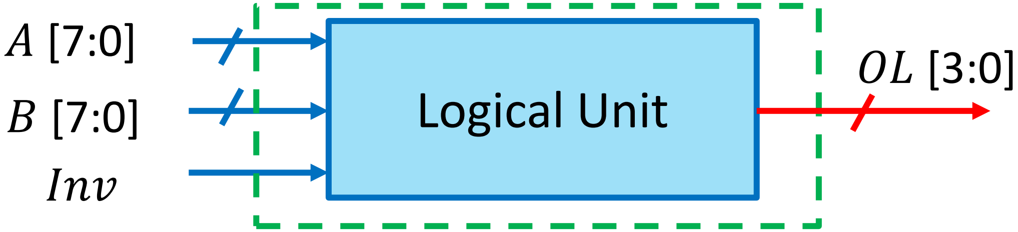




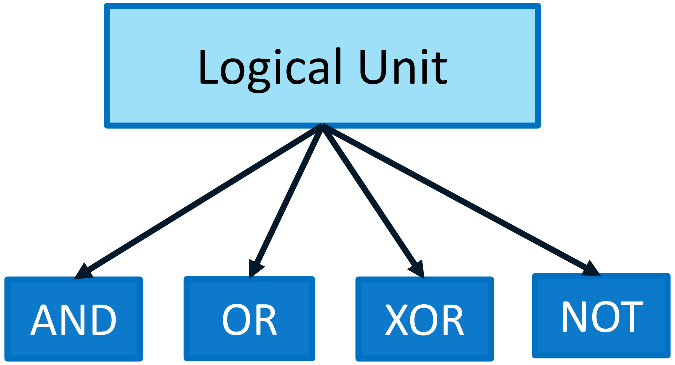
| Operation | Ctrl[2:0] |
|---|---|
| Addition (ADD) | 000 |
| Subtractor (SUB) | 001 |
| Maximum (MAX) | 010 |
| Minimum (MIN) | 011 |
| Bitwise AND (AND) | 100 |
| Bitwise OR (OR) | 101 |
| Bitwise XOR (XOR) | 110 |
| Bitwise NOT (NOT) | 111 |

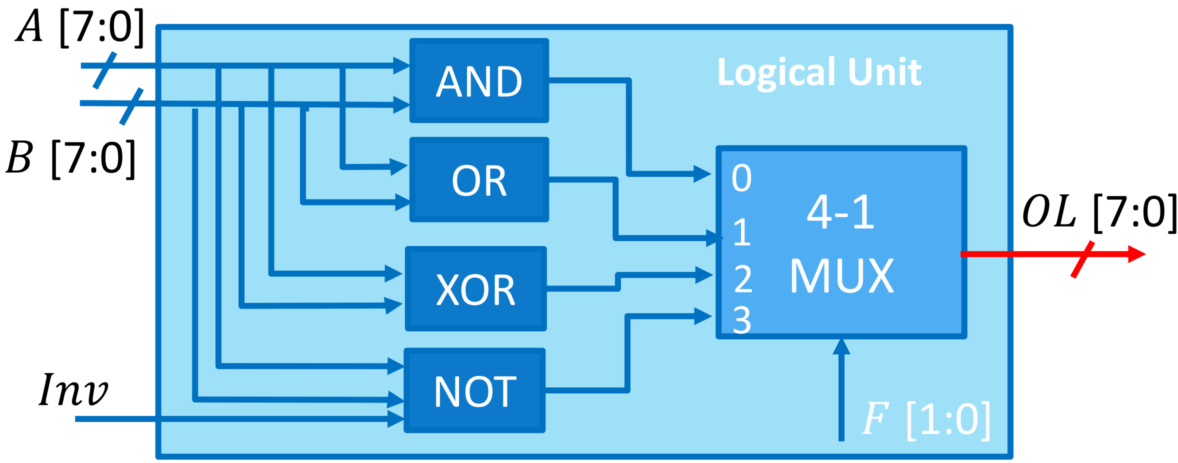


# Step4: Design: Level 2

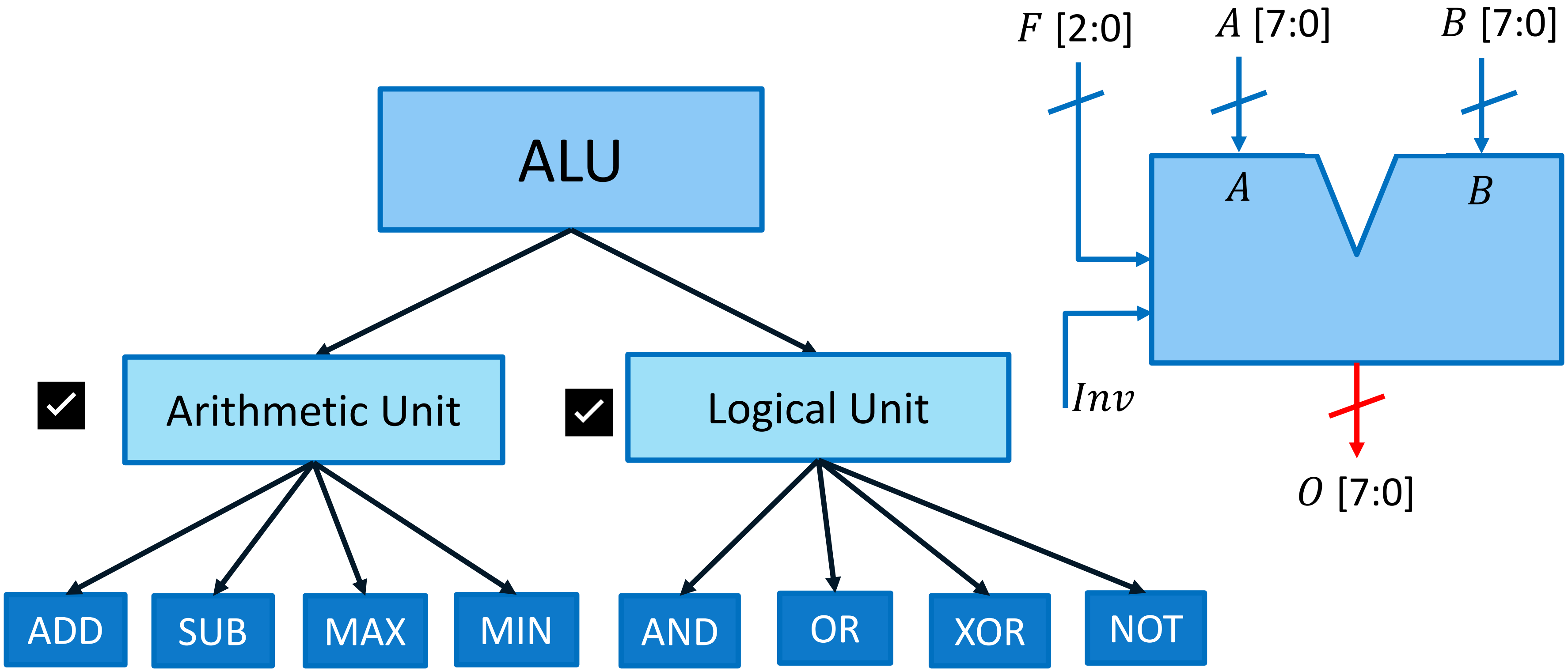


| Operation | Ctrl[2:0] |
|---|---|
| Addition (ADD) | 000 |
| Subtractor (SUB) | 001 |
| Maximum (MAX) | 010 |
| Minimum (MIN) | 011 |
| Bitwise AND (AND) | 100 |
| Bitwise OR (OR) | 101 |
| Bitwise XOR (XOR) | 110 |
| Bitwise NOT (NOT) | 111 |





# Step4: Level 2: DONE



# Step4: Design: Level 3

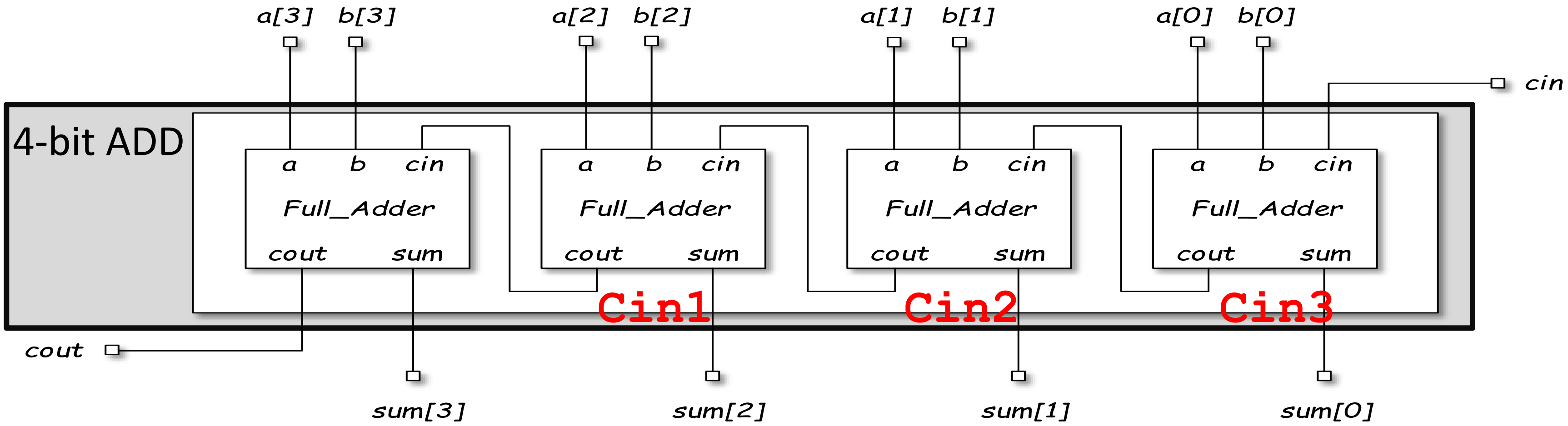
- Design an 8-bit adder

8-bit ADD

# Step4: Design: Level 3

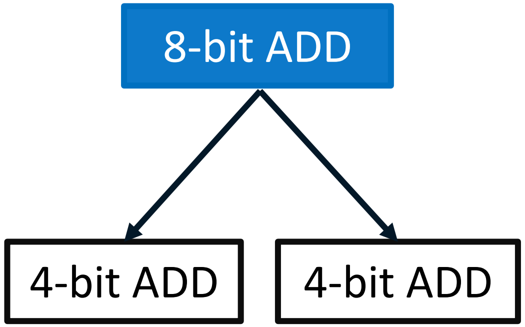
- We have already designed a 4-bit adder 🤓
- Do you remember? 🤔

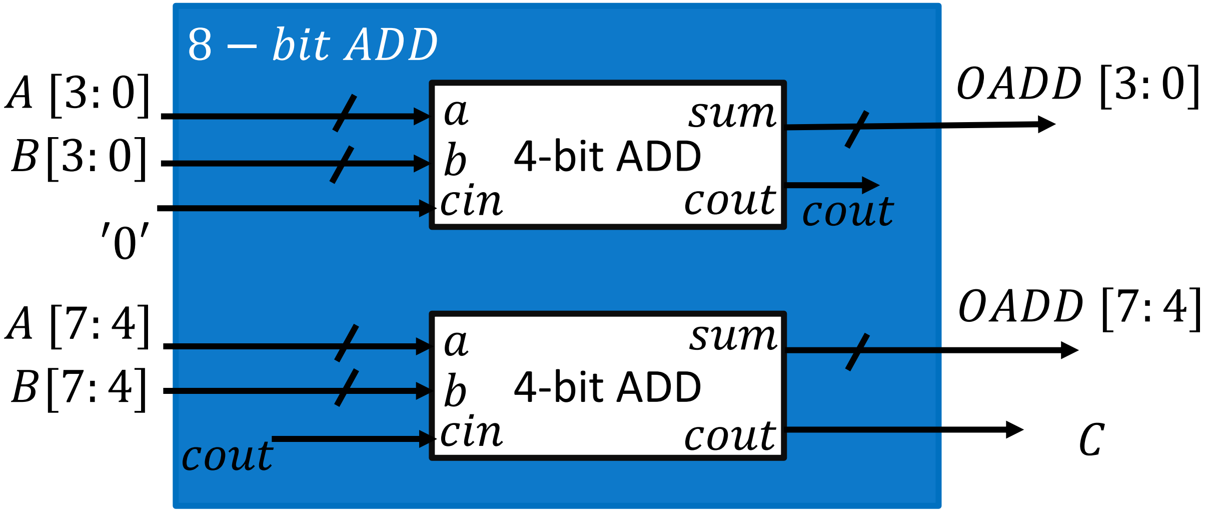
8-bit ADD



# Step4: Design: Level 3

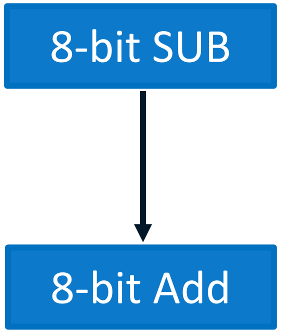
- We have already designed a 4-bit adder 🤓
- Do you remember? 🤔

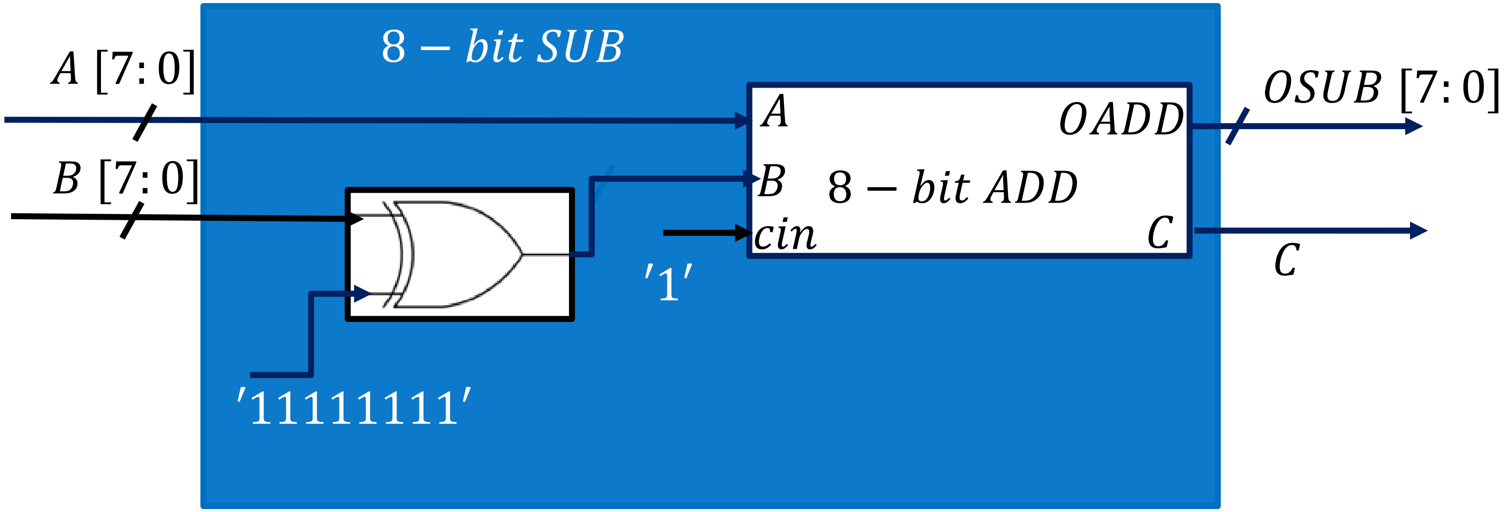




# Step4: Design: Level 3

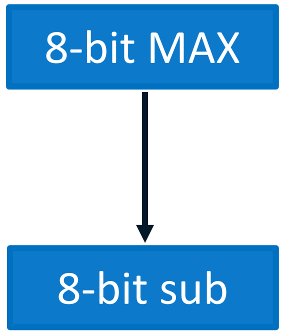
- Design a 8-bit subtractor

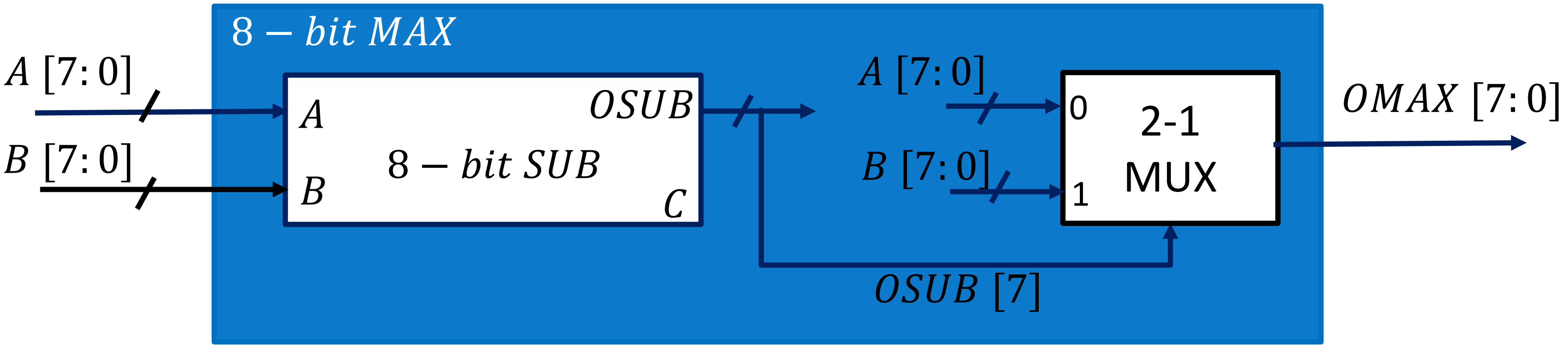




# Step4: Design: Level 3

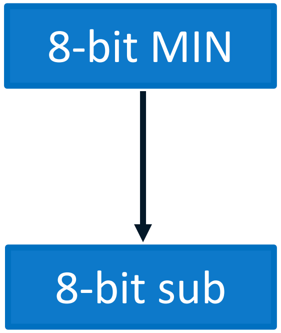
- Design a 8-bit maximum

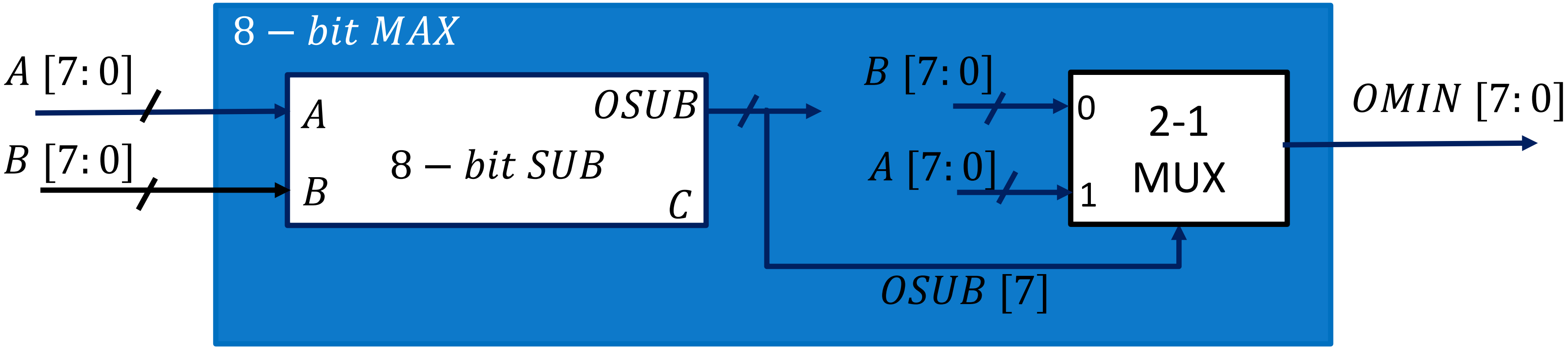




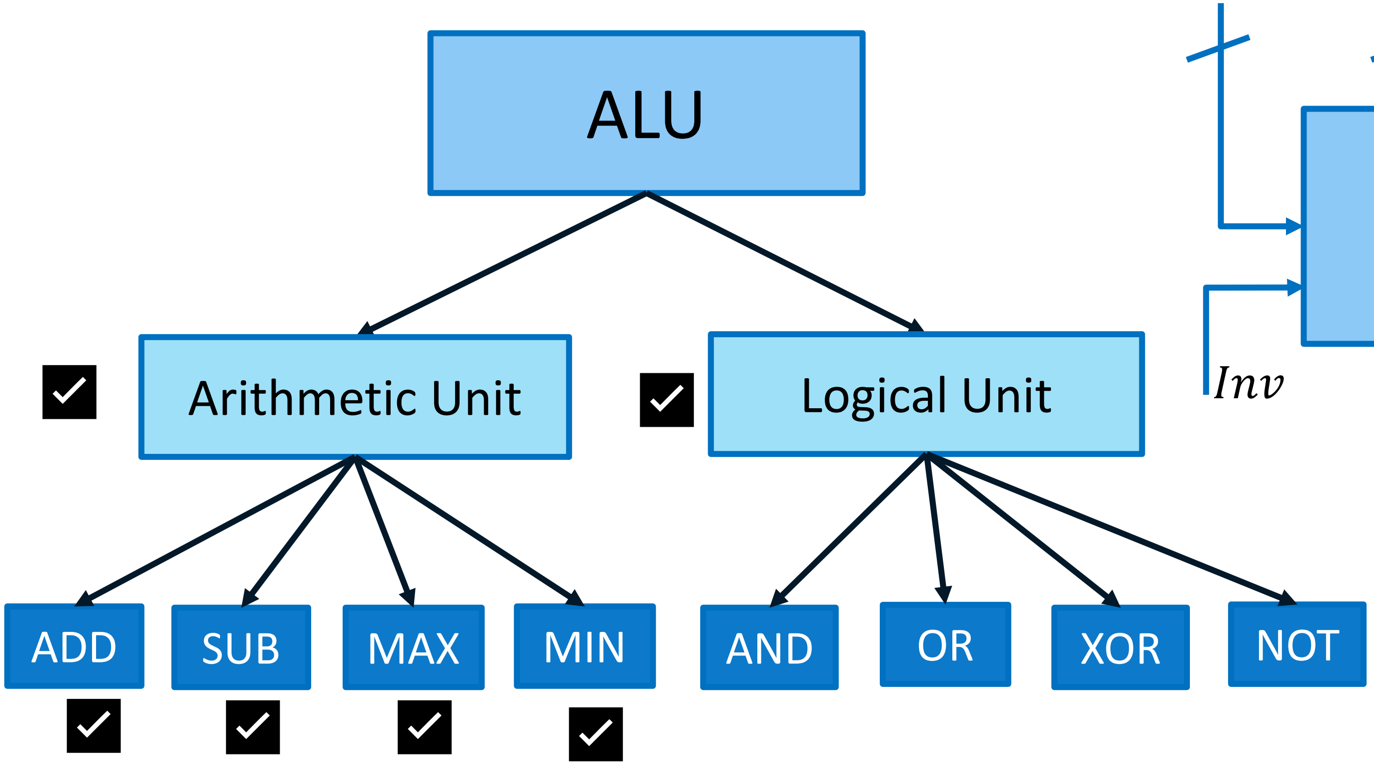
# Step4: Design: Level 3

- Design a 8-bit minimum





# Step4: Level 3: Semi-DONE!



ALU

☑ Arithmetic Unit

☑ Logical Unit

ADD ☑

SUB ☑

MAX ☑

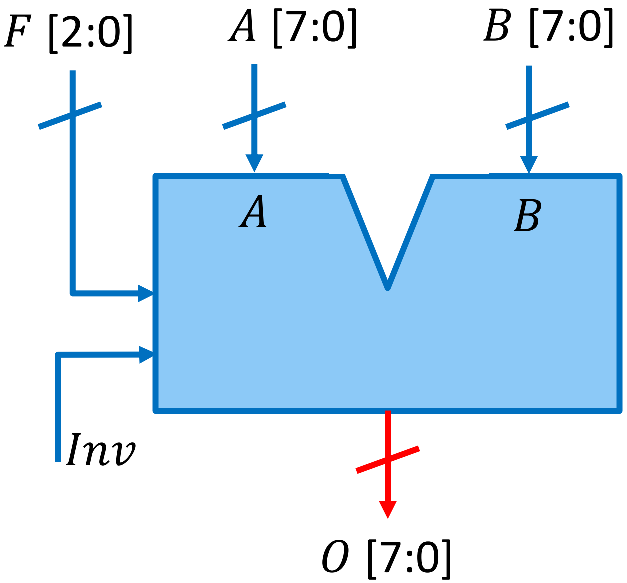
MIN ☑

AND

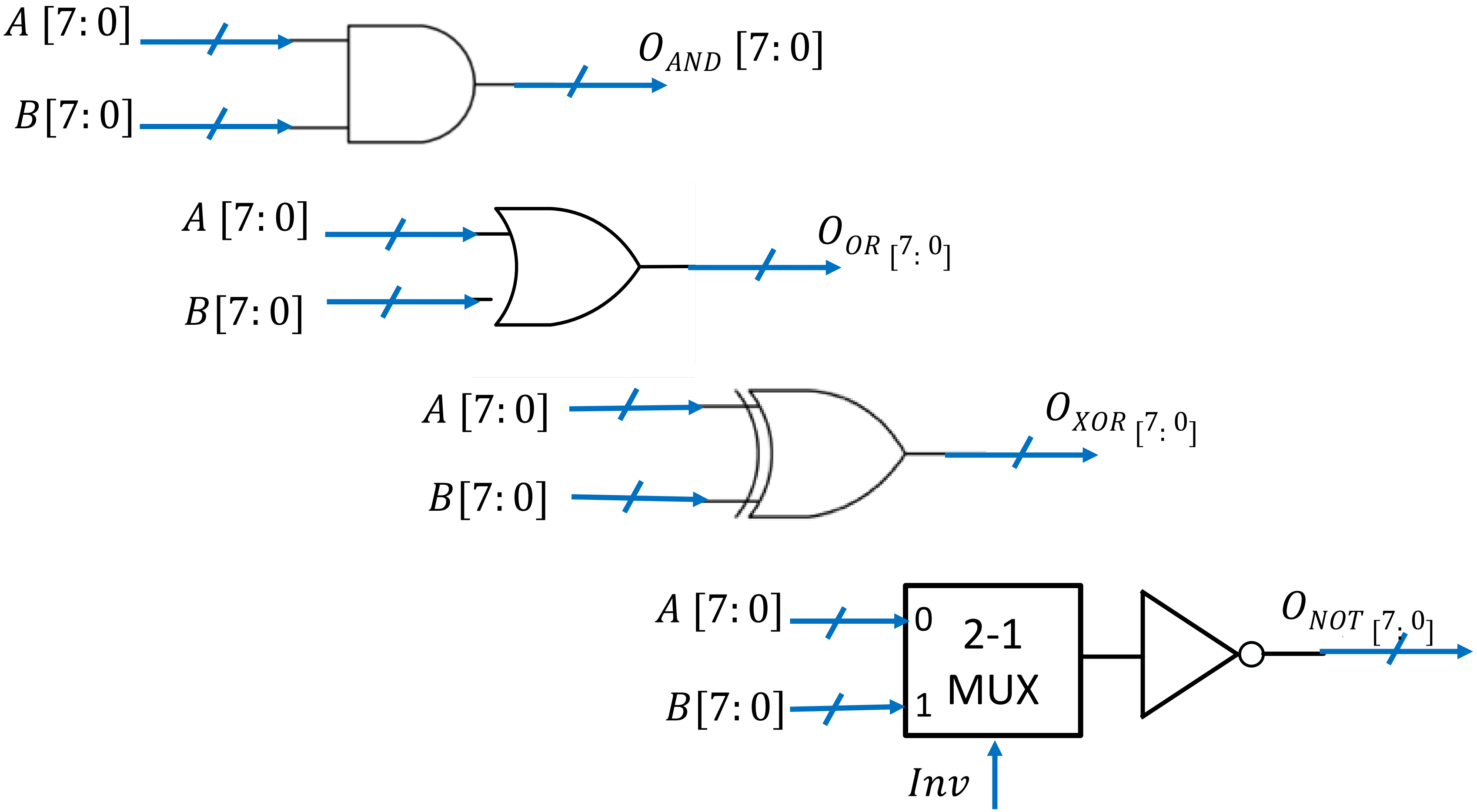
OR

XOR

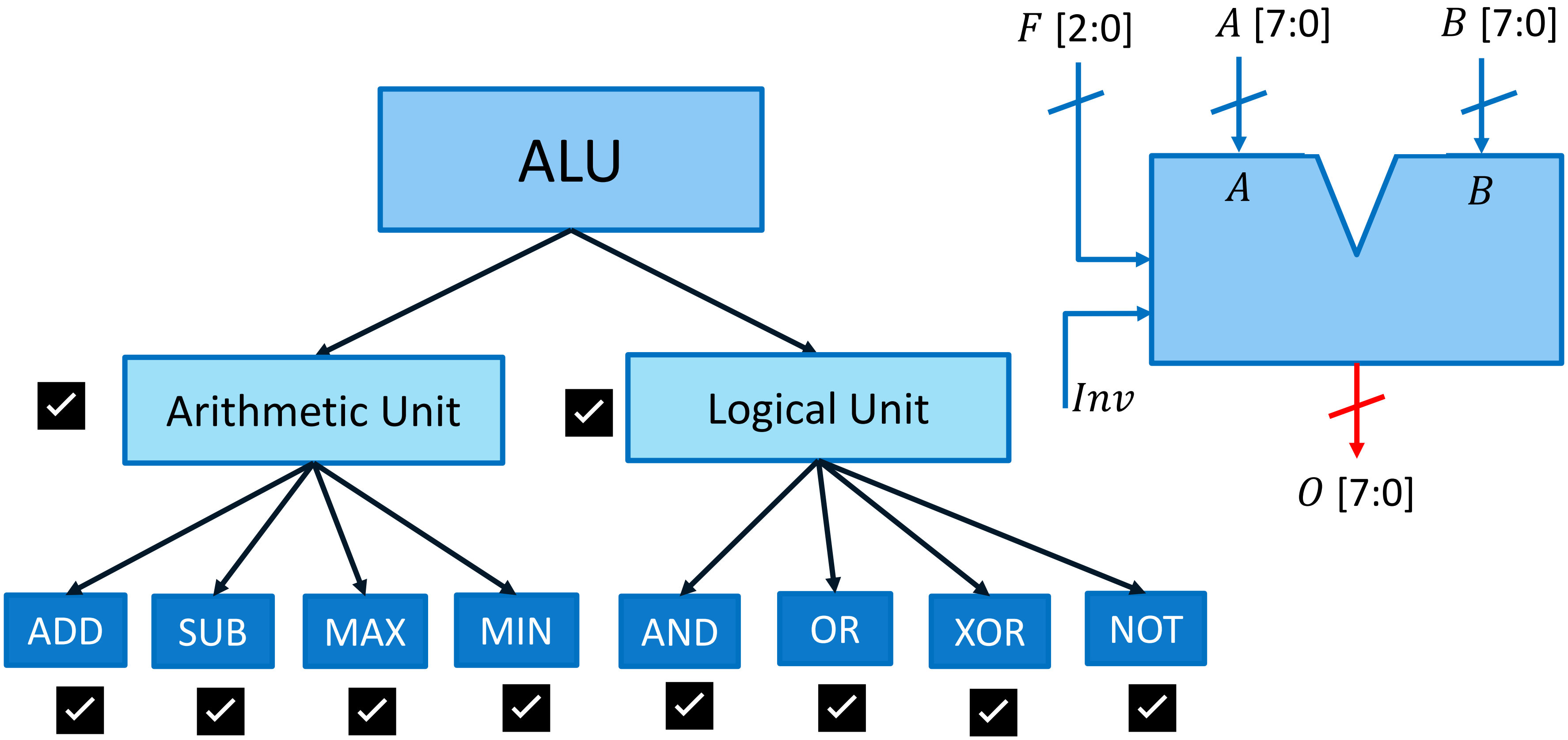
NOT



# Step4: Design: Level 3



# Step4: Level 3: DONE!



# Thank You