Computer Architecture

Keivan Navi

Cal Poly Pomona University

knavi@cpp.edu

Office hours: Tu/Th 5:25 Pm to 6:55 Pm

Office: 8-49

Computer Architecture

Arithmetic logical Unit

ALU

Arithmetic Logical Unit

- An arithmetic logic unit (ALU) is a digital circuit that performs arithmetic and logic operations on binary numbers in a computer's central processing unit (CPU).
- Arithmetic: Performs operations like addition, subtraction, multiplication, and division.
- Handles logical operations like AND, OR, and XOR, Majority gate, Minority gate,…
- An arithmetic-logic unit is the part of a central processing unit that carries out arithmetic and logic operations on the operands in computer instruction words. In some processors, the ALU is divided into two units: an arithmetic unit (AU) and a logic unit (LU).

ALU (continued)

- ALUs are a fundamental building block of modern CPUs, which contain complex and powerful ALUs. They receive inputs from registers, produce outputs, and store the results back in the registers.
- How it works:
- The control unit tells the ALU what operation to perform on the data.
- 1- The ALU receives inputs from registers.
- 2- The ALU executes calculations using binary numbers and logic gates.
- 3- The ALU generates the desired output.
- 4- The ALU stores the output in an output register.

ALU (continued)

- The Arithmetic Logic Unit (ALU) operates on data stored in registers. ALU operations take one, two or more inputs from registers and store the result back into the register. As an example:

- Or AX,BX AX<= AX +BX Or stands for logical OR
- Add R1,R2 R1<= R1+R2 Add stands for Addition

- Mac R1,R2,R3 R1<= R1+(R2*R3) Mac stands for Multiplication Accumulation
- Mad R1,R2,R3 R1<= R1+(R2*R3) Mac stands for Multiplication Addition
- In computing, especially digital signal processing, the **multiply–accumulate** (MAC) or multiply-add (MAD) operation is a common step that computes the product of two numbers and adds that product to an accumulator.

- Some old fashioned ALU were able to operate directly on data stored in the main memory. Some examples:

- Add R1, R2, Mem1 R1<= R2+Mem1
- And R1, Mem1, Mem2 R1<= Mem1.Mem2
- Sub Mem1, Mem2, Mem3 Mem1<= Mem1+ Mem2

ALU (continued)

- The ALU supports four fundamental data operations described as:
- Arithmetic: Some ALUs supports only addition and subtraction on signed and unsigned operands. Some more sophisticated ones can perform more operations like Multiplications and Divisions. Some of them are even able to do Mac (Mad) and more complex instructions.

- Relational: Some ALUs support the equal, not-equal, greater-than-or-equal, and less-than relational operations (==, != >=, <).

- Logical: The ALUs support logical operations. As examples:
- AND, OR, NOR, and XOR ,…

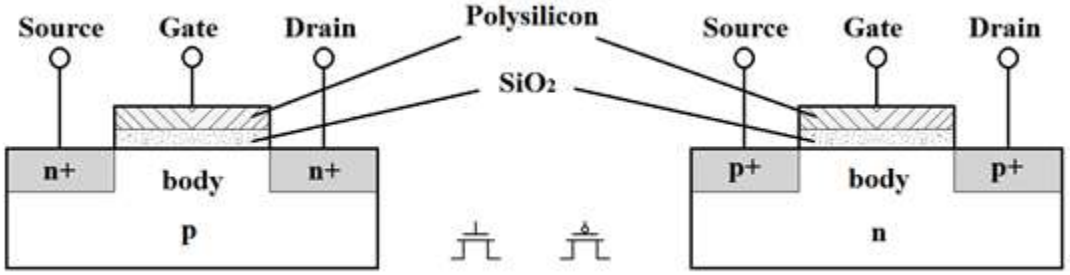
- Shift: Some ALUs support logical and arithmetic shift operations.
- For load/store instructions, some processors use ALU to calculate the memory address. For conditional control transfer instructions some processors use the relational operations in ALU area to decide whether to take the branch or not.

Very simple AIU

- The question is: What is the simplest ALU with less devices? Our simple AIU must be able to perform all standard logical operations as well as Addition, Subtraction, Multiplication and Division.
- We must be looking for a Universal or Complete gate. We consider 3 universal gates and finally chose one of them:
- NOR, NAND and Minority gate (Majority Not)

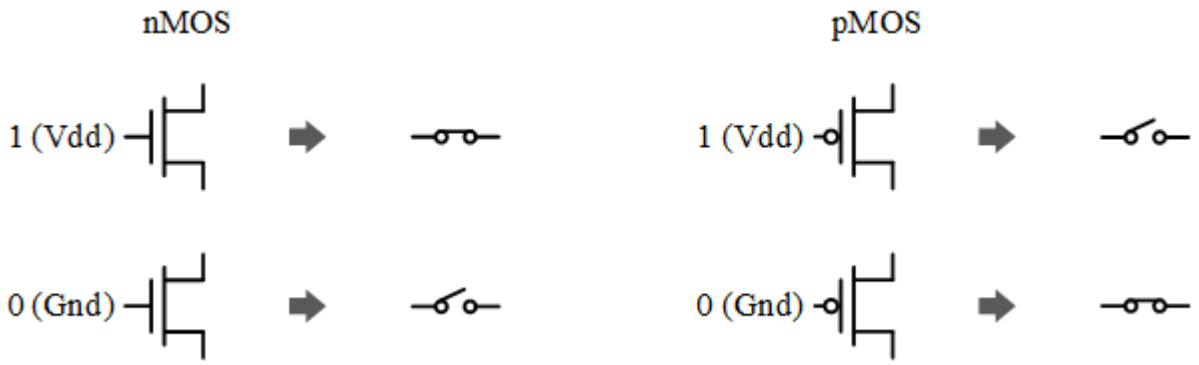
NAND, NOR or Minority gate?

- In order to find out which one is the best choice, we must look at their structures.
- The structure of nMOS and pMOS transistors follows:



Not, NAND, NOR or Minority gate structures

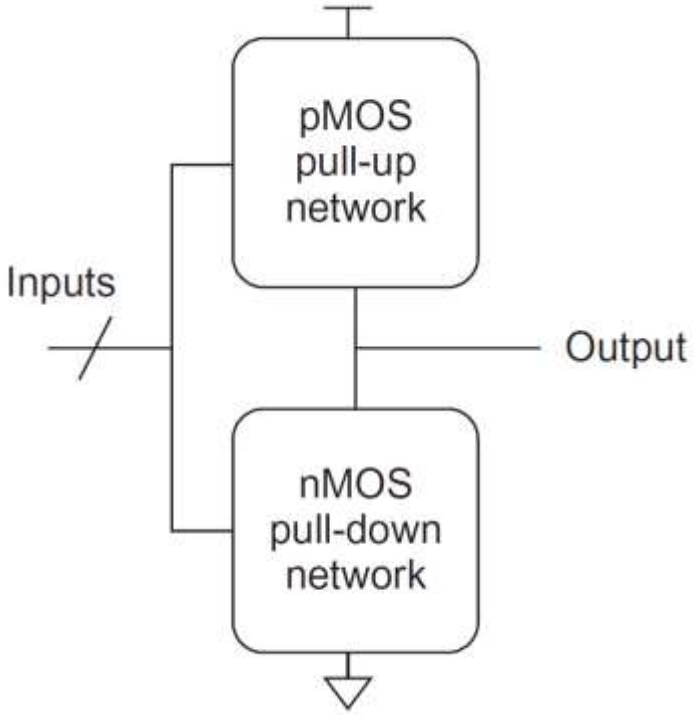
- Applying 1 to the input of an NMOS transistor activates it. In other words it will be ***ON***. On the contrary, applying 0 to the input of an NMOS transistor makes it ***OFF***



- Applying 0 to the input of a PMOS transistor activates it. In other words it will be ***ON***. On the contrary, applying 1 to the input of a PMOS transistor makes it ***OFF.***

Not, NAND, NOR or Minority gate structures

- Logic gates in conventional or complementary CMOS (also simply referred to as CMOS in the sequel) are built from an NMOS pull-down and a dual PMOS pull-up logic network.
- NMOS is faster than PMOS.

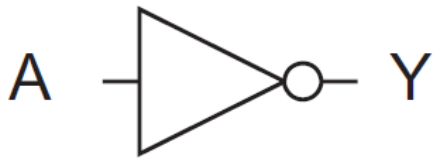


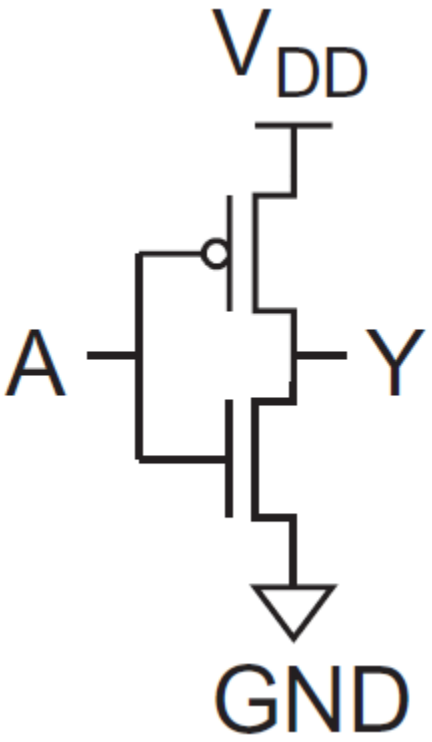
Invertor (Not gate)

- Invertor structure:

Inverter truth table

A	Y
0	1
1	0



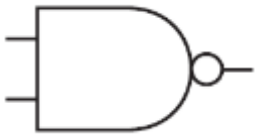


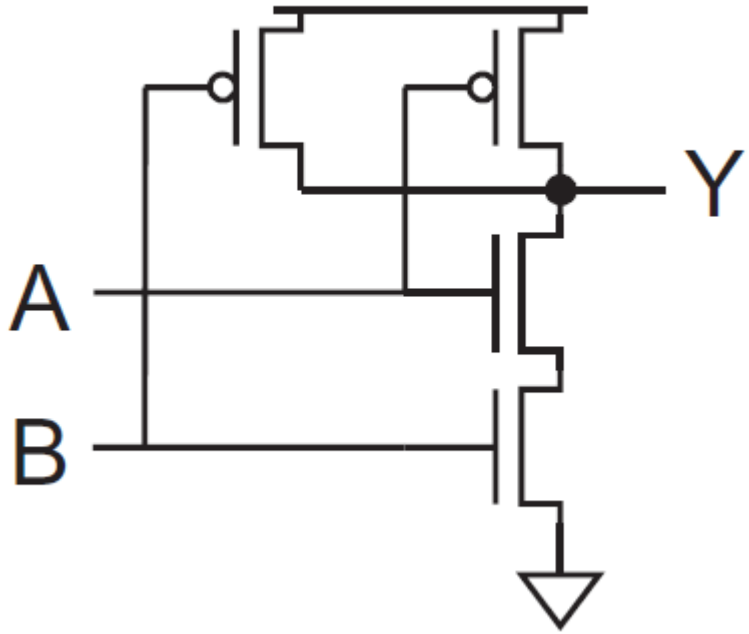
NAND

- NAND structure:

NAND gate truth table

A	B	Pull-Down Network	Pull-Up Network	Y
0	0	OFF	ON	1
0	1	OFF	ON	1
1	0	OFF	ON	1
1	1	ON	OFF	0



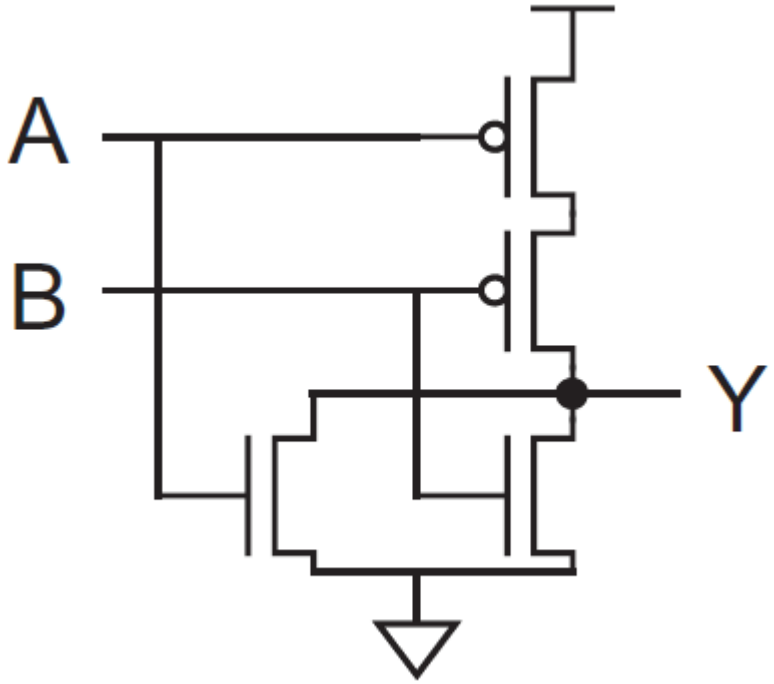


NOR

- NOR structure:

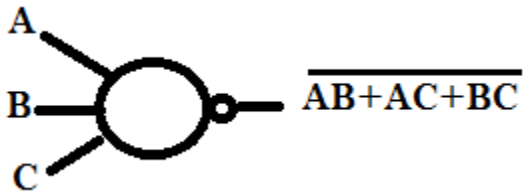
NOR gate truth table

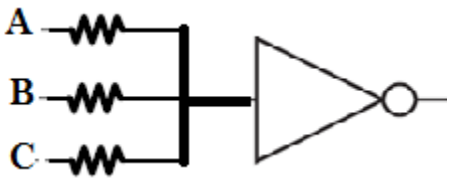
A	B	Pull-Down Network	Pull-Up Network	Y
0	0	OFF	ON	1
0	1	ON	OFF	0
1	0	ON	OFF	0
1	1	ON	OFF	0

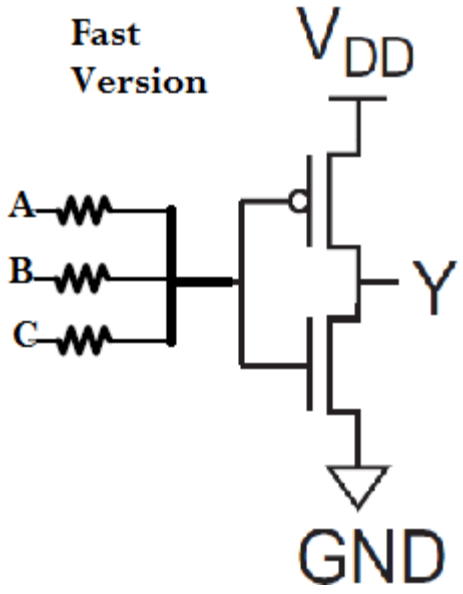


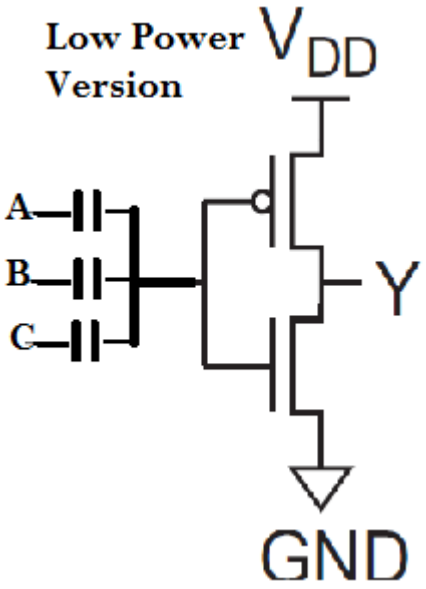
Minority Gate

- Minority gate Structure:



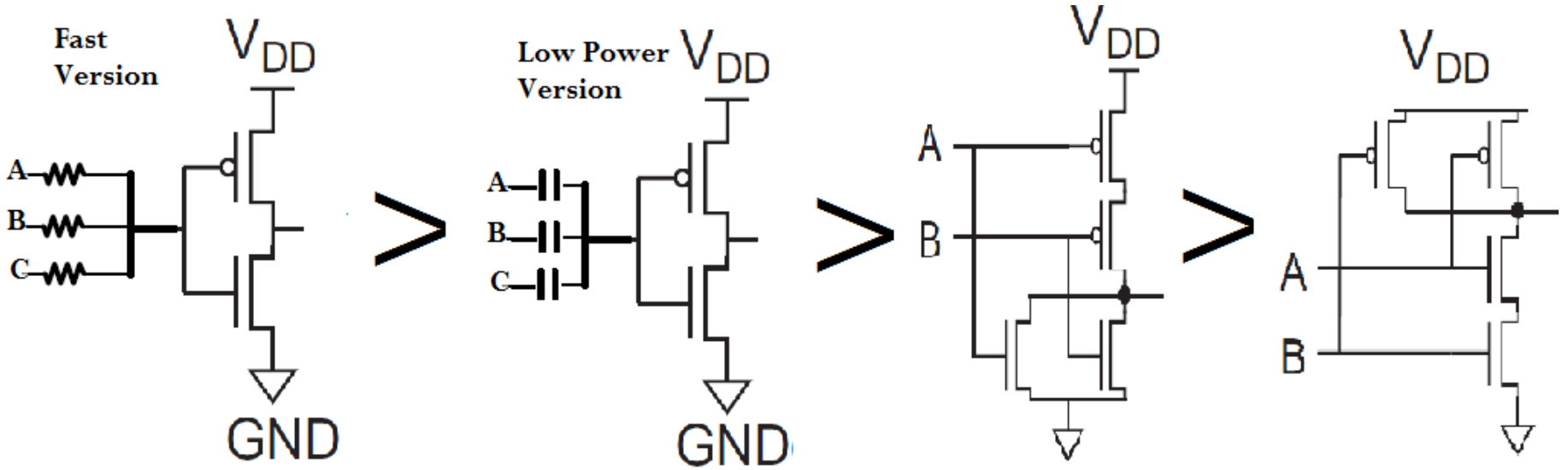






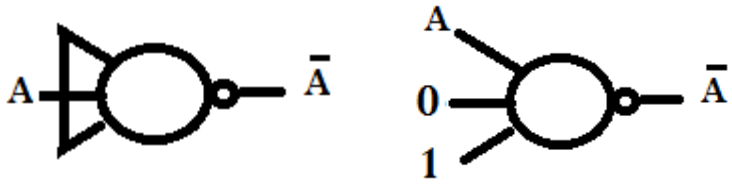
Which one is the Fastest

- Minority is faster than NAND and NAND is faster than Nor. Resistor based Minority gate is faster than Capacitor based one.

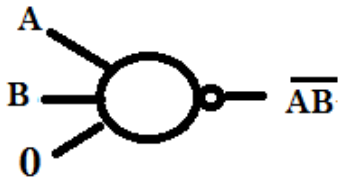


The Minority gate is a Universal gate

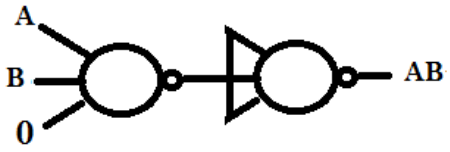
- Making NOT, NAND, NOR, NOT, AND…:
- NOT:



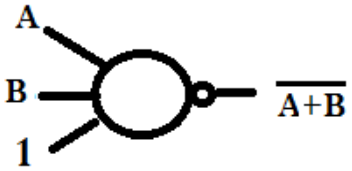
- NAND:



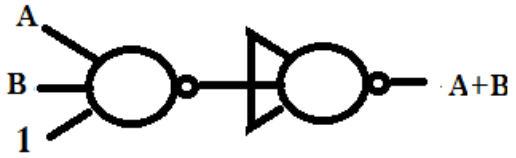
AND:



- NOR:



OR:

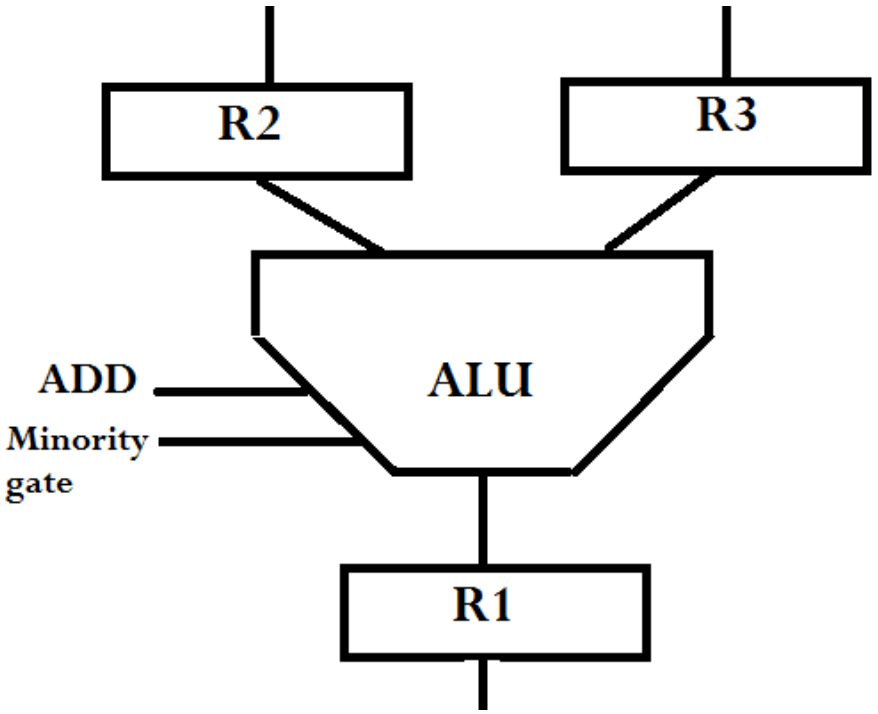


Adder, Subtractor, Multiplier or Divider?

- What is the best Arithmetic operator?
- Using Adder we can make a 2's complement subtraction: A+$\bar{B} + 1$
- Using Adder we can make a Multiplier by repeated addition.
- Using Adder we can make a Divider by repeated subtraction.
- Adder is the simplest and fastest arithmetic operator. Then the choice is "ADDER".

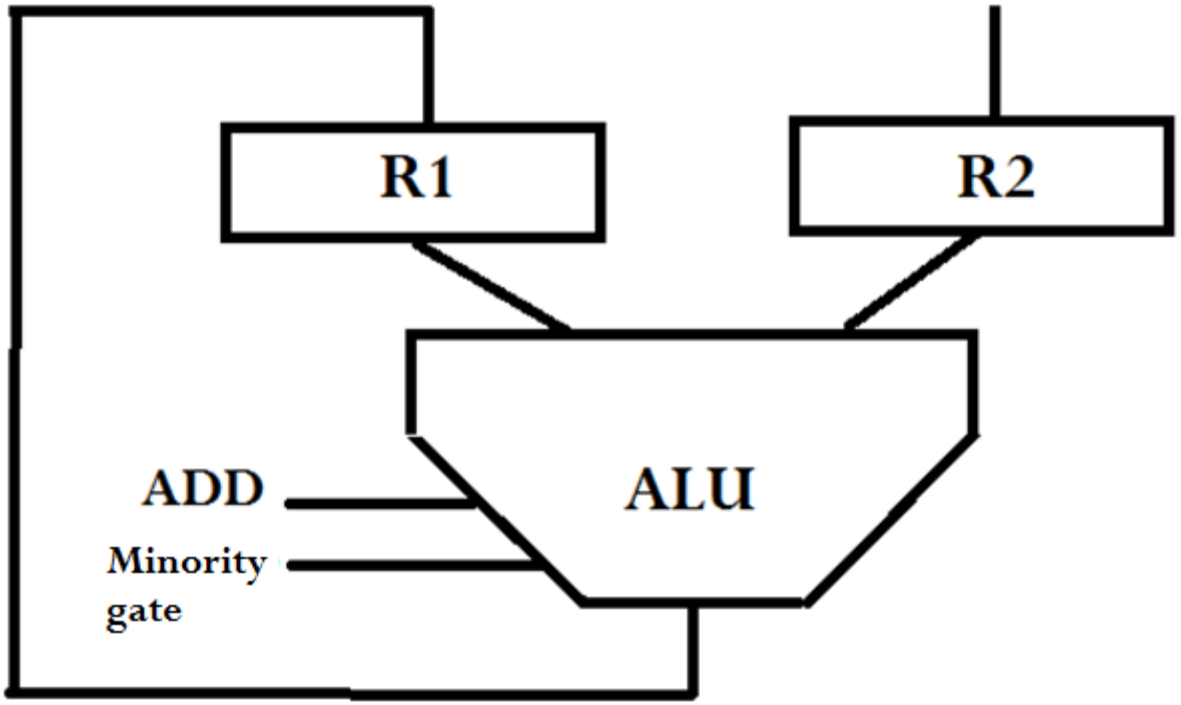
A simple ALU

- A simple ALU follows:
- Can we make it simpler?
- Operation R1,R2,R3
- Mul R1,R2,R3
- R1<= R2*R3



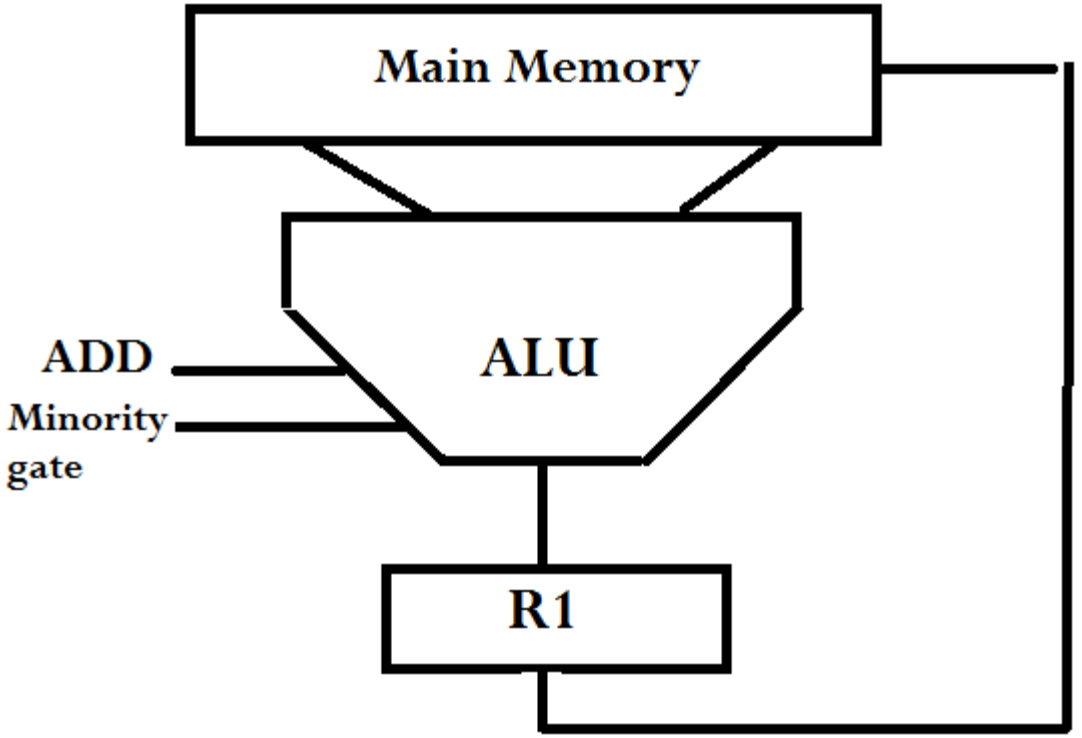
A simpler ALU

- A simpler ALU follows:
- Operation R1, R2 Add R1,R2 R1<=R1+R2



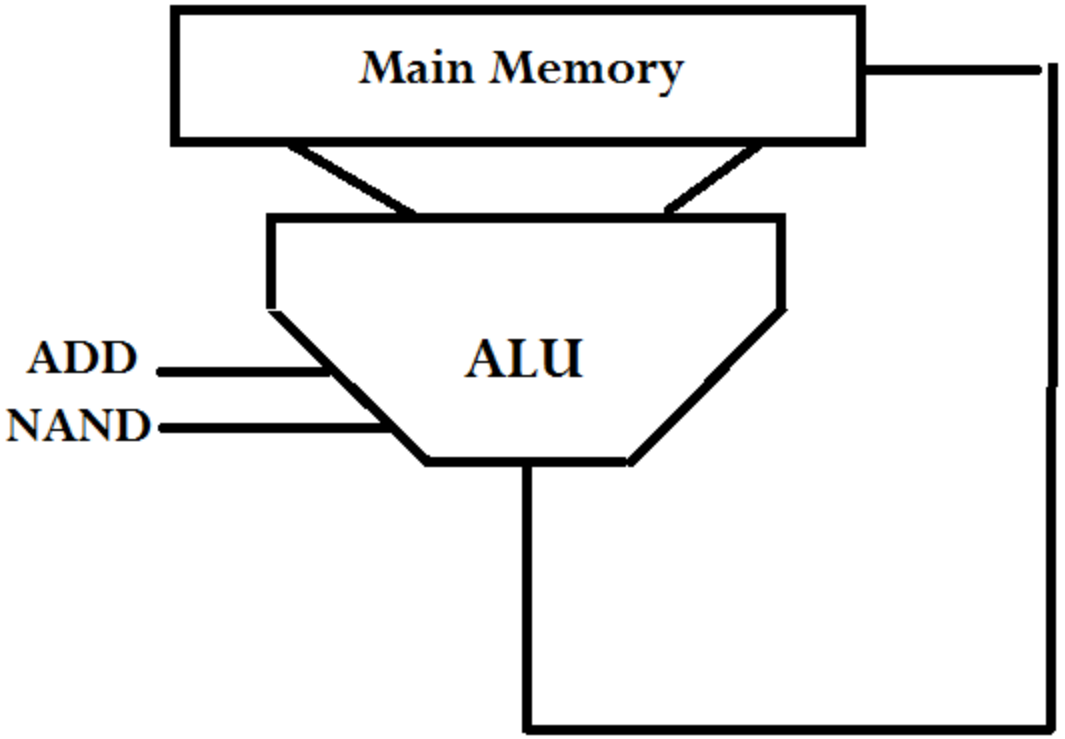
Another example of ALU

- Operation R1, Mem1, Mem2
- Sub R1, 100,105 R1<= (100) - (105)



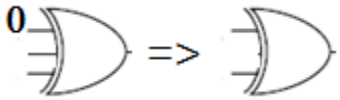
Another ALU

- Operation Meme1,Mem2,Mem3
- NAND 100,101,104 100<= (101) + (104)

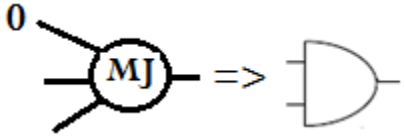


Only Add instruction

- How can we do: Xor R1,R2,R3?
- Full Adder Structure:
- SUM = a XOR b XOR c Carry= ab + ac+ bc
- 2 inputs XOR using 3 inputs XOR:



- 2 inputs AND using Majority gate:



- NOT gate using 3 inputs XOR: