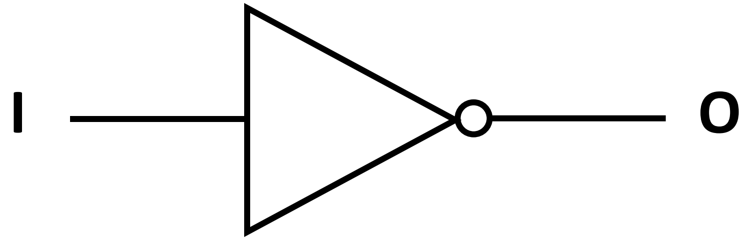
Digital Logic

Abstractions in CS (gates)

- Basic Gate: Inverter

Truth Table

I	O
0	**1**
1	**0**

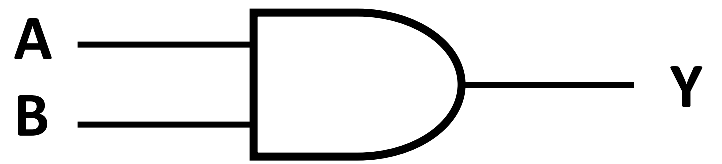


Abstractions in CS (gates)

- Basic Gate: AND

Truth Table

A	B	Y
0	**0**	**0**
0	**1**	**0**
1	**0**	**0**
1	**1**	**1**

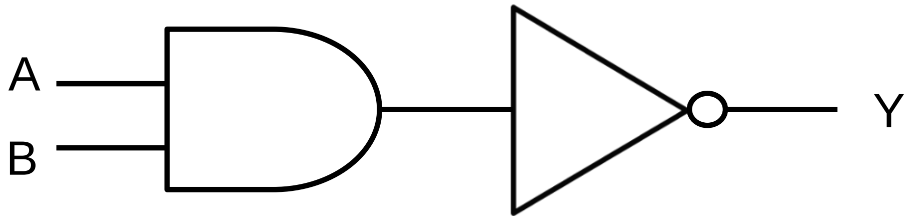


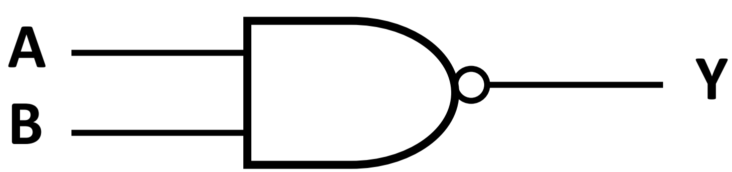
Abstractions in CS (gates)

- Basic Gate: NAND (Negated AND)

Truth Table

A	B	Y
0	**0**	**1**
0	**1**	**1**
1	**0**	**1**
1	**1**	**0**



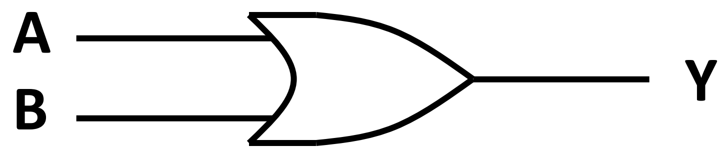


Abstractions in CS (gates)

- Other Basic Gates: OR gate

Truth Table

A	B	Y
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**1**

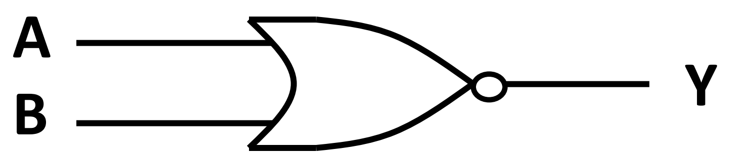


Abstractions in CS (gates)

- Other Basic Gates: NOR (Negated OR) gate

Truth Table

A	B	Y
0	**0**	**1**
0	**1**	**0**
1	**0**	**0**
1	**1**	**0**

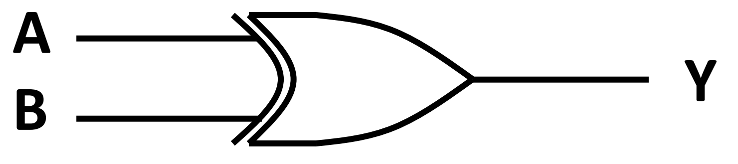


Abstractions in CS (gates)

- Other Basic Gates: XOR gate

Truth Table

A	B	Y
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**0**



Logic Blocks

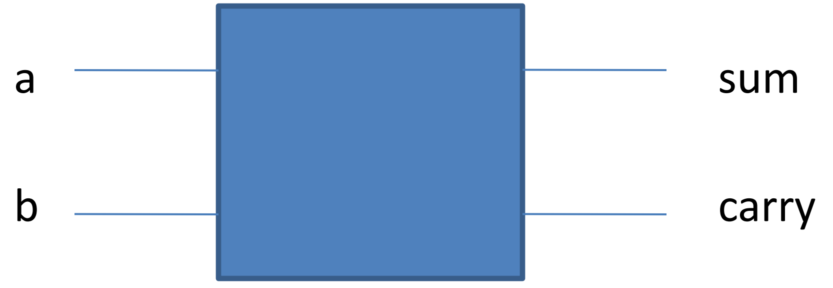
- Logic blocks are built from gates that implement basic logic functions
  - Any logical function can be constructed using AND gates, OR gates, and inversion.

Adder

- In computers, the most common task is add.
- In MIPS, we write "`add $t0, $t1, $t2`." The hardware will get the values of `$t1` and `$t2`, feed them to an adder, and store the result back to `$t0`.
- So how the adder is implemented?

Half-adder

- How to implement a one-bit half-adder with logic gates?
- A half adder takes two inputs, a and b, and generates two outputs, sum and carry. The inputs and outputs are all one-bit values.



Half-adder

- First, how many possible combinations of inputs?

Half-adder

- Four combinations.

a	b	sum	carry
0	0		
0	1		
1	0		
1	1		

Half-adder

- Four combinations.

Index = ∑inputs	a	b	sum	carry
0	0	0		
1	0	1		
2	1	0		
3	1	1		

Half-adder

- The value of sum should be? Carry?

a	b	sum	carry
0	0		
0	1		
1	0		
1	1		

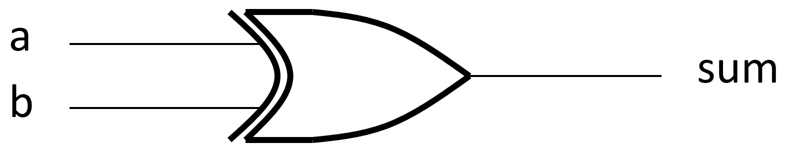
Half-adder

- Okay. We have two outputs. But let’s implement them one by one.
- First, how to get sum? Hint: look at the truth table.

a	b	sum
0	0	0
0	1	1
1	0	1
1	1	0

Half-adder

- Sum



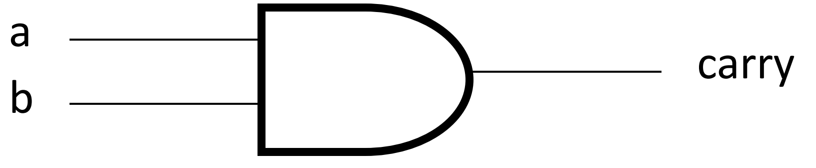
How about carry?

- The truth table is

a	b	carry
0	0	0
0	1	0
1	0	0
1	1	1

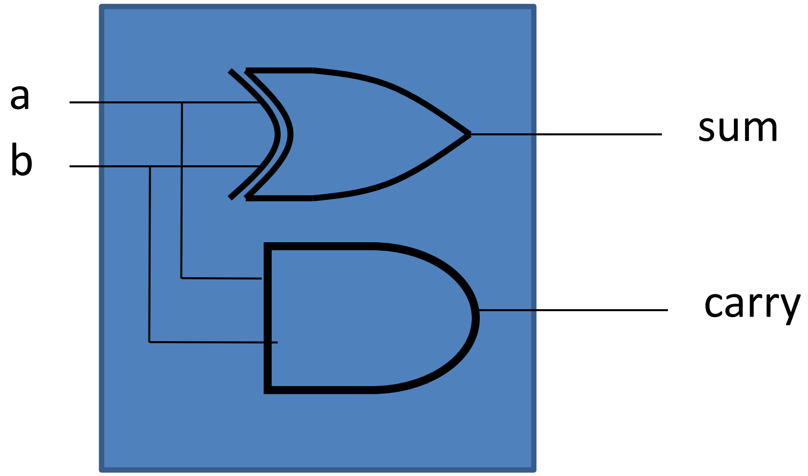
Carry

- So, the circuit for carry is



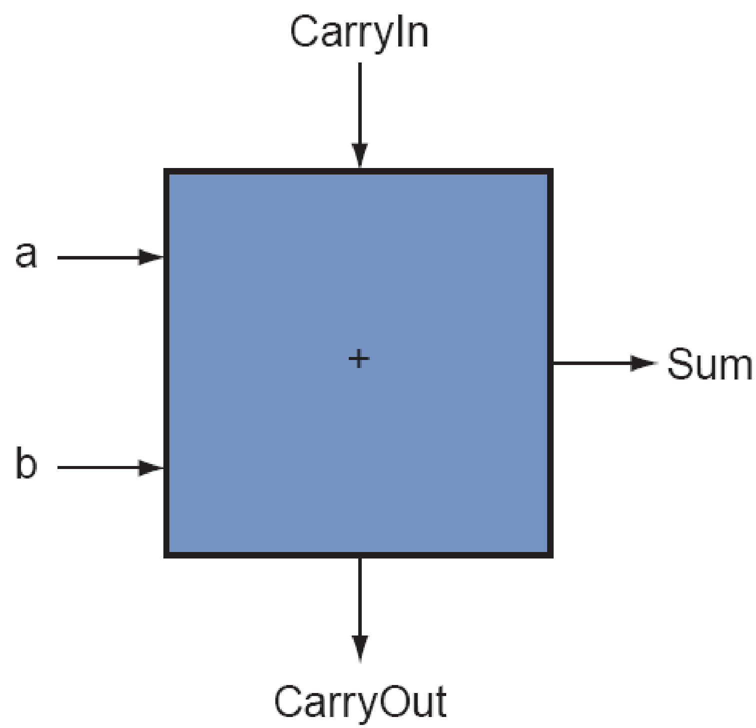
Half-adder

- Put them together, we get



1-Bit Adder

- 1-bit full adder
  - Also called a (3, 2) adder



Constructing Truth Table for 1-Bit Adder

Inputs			Outputs	
a	b	CarryIn	CarryOut	Sum
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Truth Table for a 1-Bit Adder

Inputs			Outputs		
a	b	CarryIn	CarryOut	Sum	Comments
0	0	0	0	0	$0 + 0 + 0 = 00_{two}$
0	0	1	0	1	$0 + 0 + 1 = 01_{two}$
0	1	0	0	1	$0 + 1 + 0 = 01_{two}$
0	1	1	1	0	$0 + 1 + 1 = 10_{two}$
1	0	0	0	1	$1 + 0 + 0 = 01_{two}$
1	0	1	1	0	$1 + 0 + 1 = 10_{two}$
1	1	0	1	0	$1 + 1 + 0 = 10_{two}$
1	1	1	1	1	$1 + 1 + 1 = 11_{two}$

Sum?

- Sum is `1’ when one of the following four cases is true:
  - a=1, b=0, c=0
  - a=0, b=1, c=0
  - a=0, b=0, c=1
  - a=1, b=1, c=1

Sum

- The idea is that we will build a circuit made of **and** gates and an **or** gate **faithfully** according to the truth table.
  - Each and gate corresponds to one ``true'' row in the truth table. The and gate should output a ```1'' if and only if the input combination is the same as this row. If all other cases, the output is ``0.''
  - So, whenever the input combination falls in one of the ``true'' rows, one of the and gates is ``1'', so the output of the or gate is 1.
  - If the input combination does not fall into any of the ``true'' rows, none of the and gates will output a ``1'', so the output of the or gate is 0.

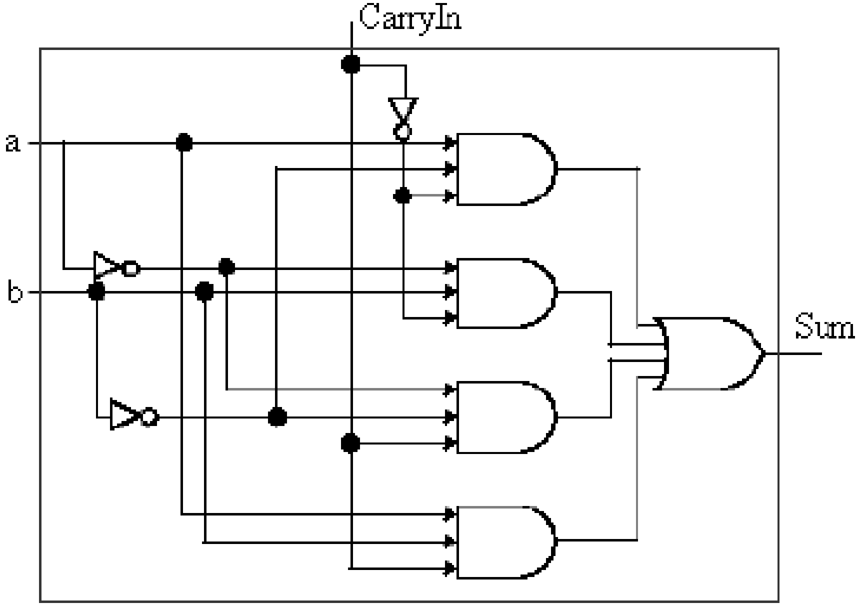
Boolean Algebra

- We express logic functions using logic equations using Boolean algebra
  - The OR operator is written as +, as in A + B.
  - The AND operator is written as ·, as A · B.
  - The unary operator NOT is written as $\overline{A}$ or A'.
- **Remember:** This is not the binary field. Here 0+0=0, 0+1=1+0=1, **1+1=1.**

Sum

a=1, b=0, c=0
a=0, b=1, c=0
a=0, b=0, c=1
a=1, b=1, c=1

$$Sum = (a \cdot \bar{b} \cdot \overline{CarryIn}) + (\bar{a} \cdot b \cdot \overline{CarryIn}) + (\bar{a} \cdot \bar{b} \cdot CarryIn) + (a \cdot b \cdot CarryIn)$$



Carryout bit?

- Carryout bit is also ‘1’ in four cases. When a, b and carryin are 110, 101, 011, 111.

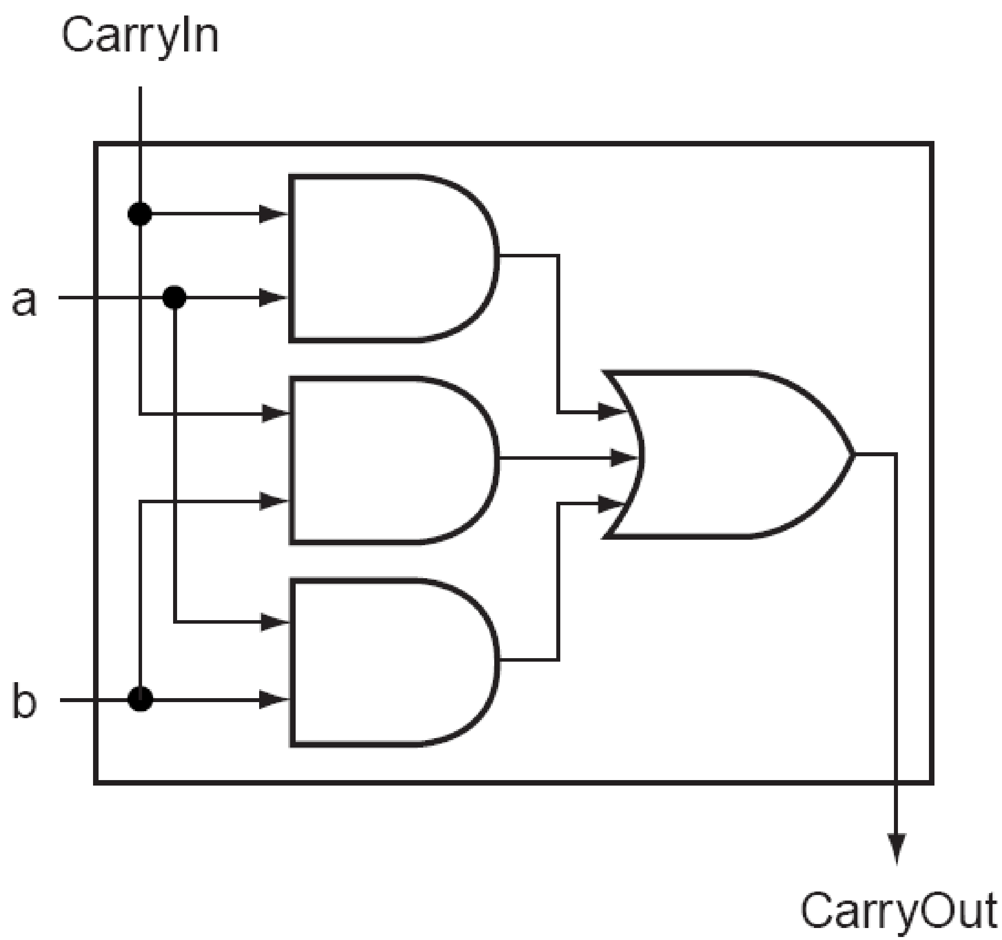
$$CO = (a \cdot b \cdot \bar{c}) + (a \cdot \bar{b} \cdot c) + (\bar{a} \cdot b \cdot c) + (a \cdot b \cdot c)$$

- Does it mean that we need a similar circuit as sum?

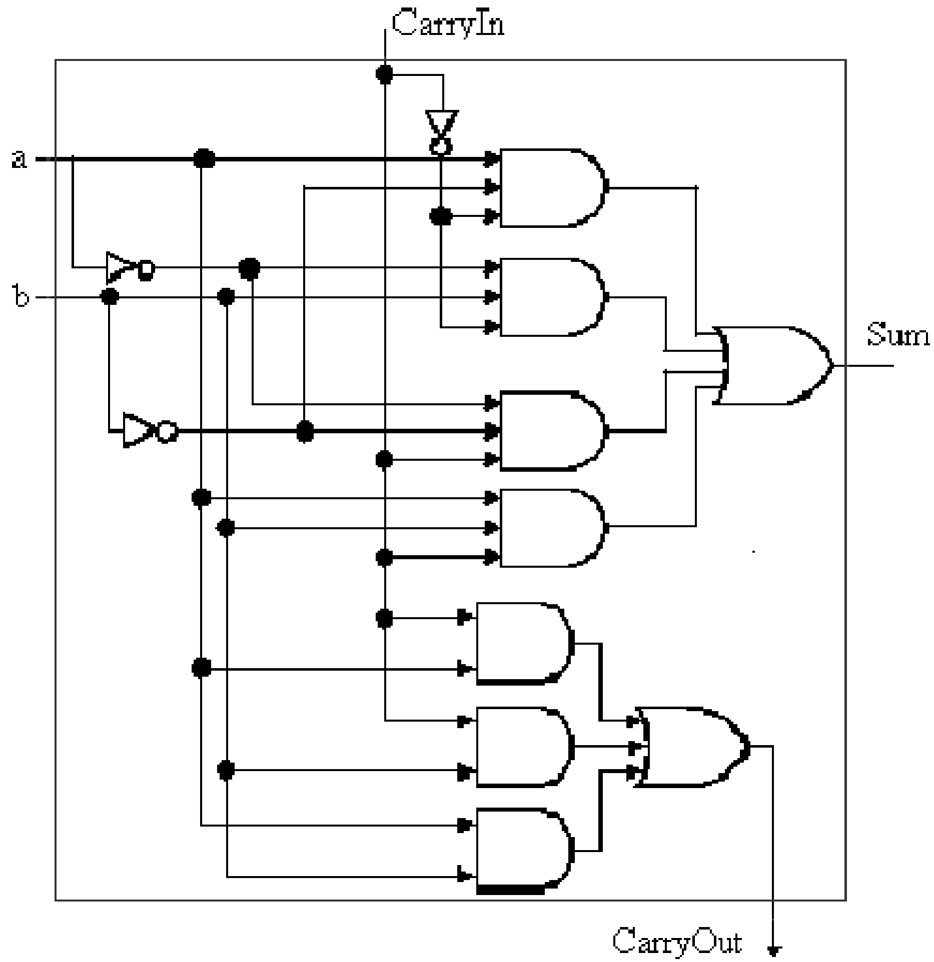
Carryout bit

$$CO = (a \cdot b \cdot \bar{c}) + (a \cdot \bar{b} \cdot c) + (\bar{a} \cdot b \cdot c) + (a \cdot b \cdot c)$$

- Actually, it can be simpler $CO = (a \cdot b) + (b \cdot c) + (c \cdot a)$

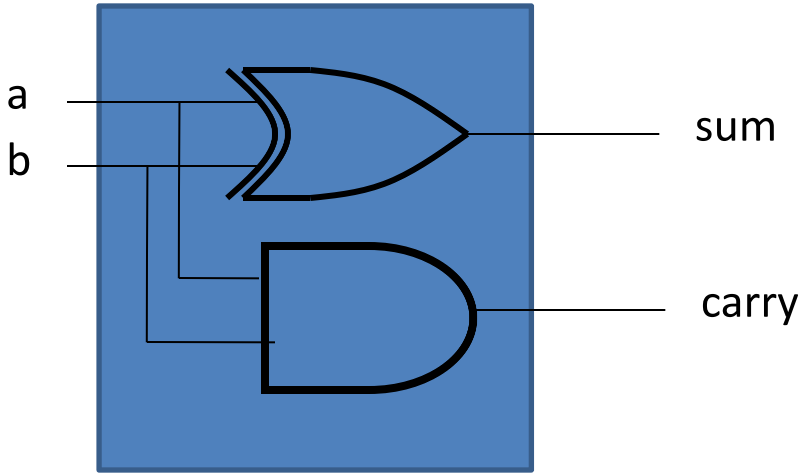


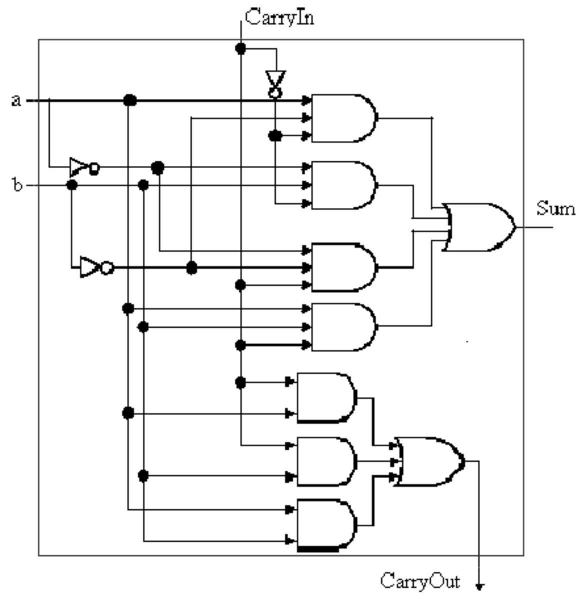
1-Bit Adder



Delay

- Hardware has delays.
- Delay is defined as the time since the input is stable to the time when the output is stable.
- How much more delay does the one-bit full adder take, when compared to the one-bit half adder?

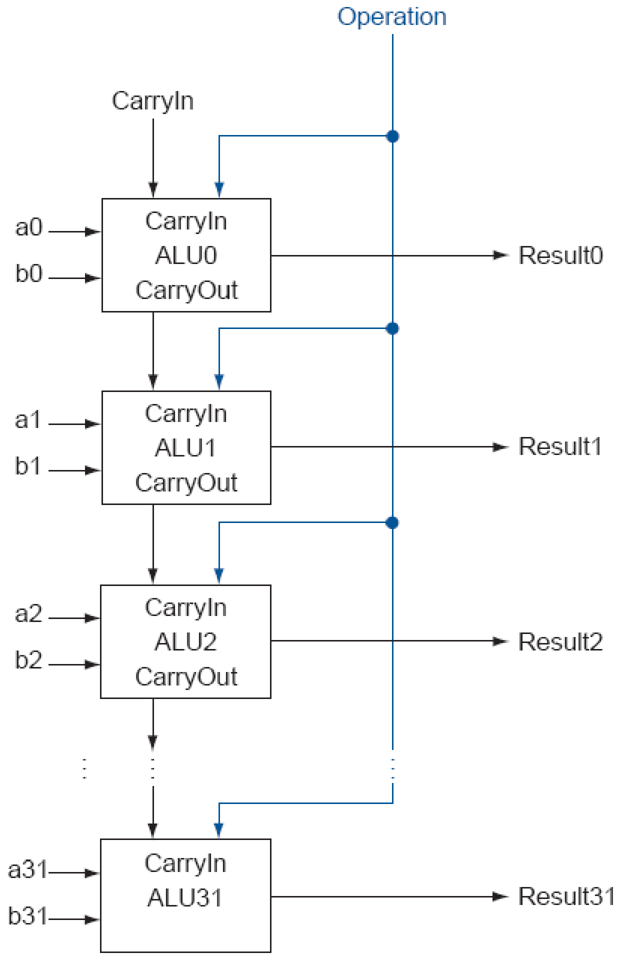




32-bit adder

- How to get the 32-bit adder used in MIPS?

32-bit adder


Operation
CarryIn
a0
b0
CarryIn
ALU0
CarryOut
Result0
a1
b1
CarryIn
ALU1
CarryOut
Result1
a2
b2
CarryIn
ALU2
CarryOut
Result2
a31
b31
CarryIn
ALU31
Result31