C335
Computer Structures

ALU Design (I)

Dr. Liqiang Zhang

Department of Computer and Information Sciences

The Design is to Represent

(1) **Functional Specification**

Inputs: 2 x 16 bit operands- A, B; 1 bit carry input- Cin, 3 bit mode/function.

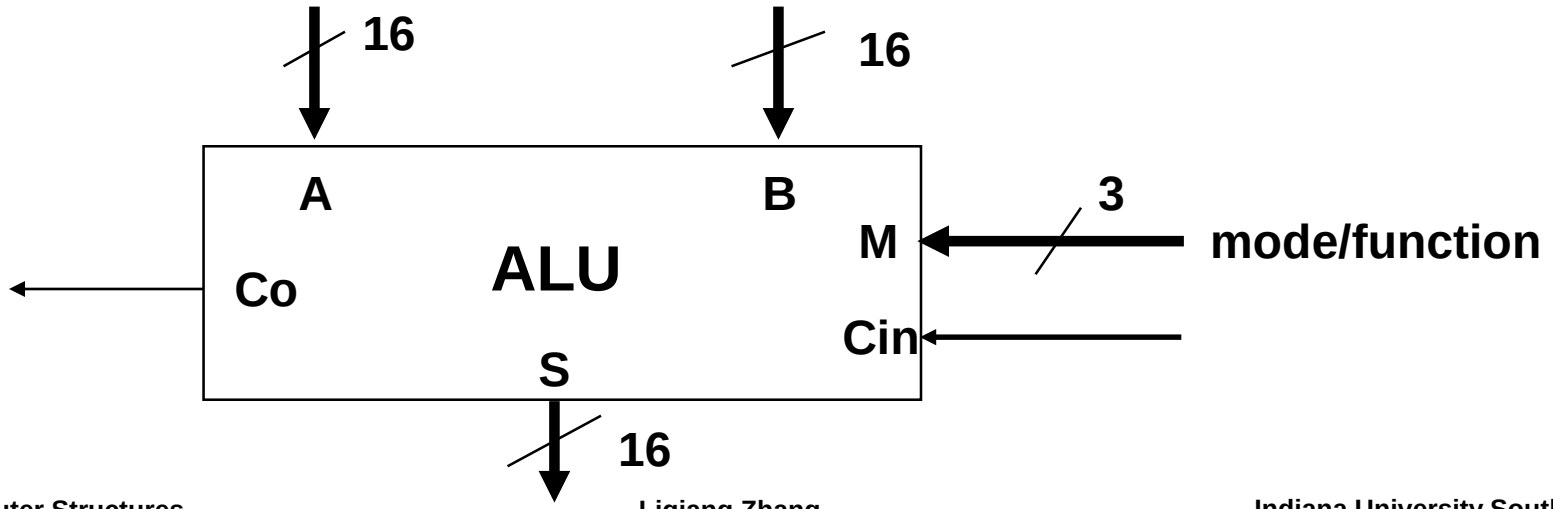
Outputs: 1 x 16 bit result- S; 1 bit carry output- Co.

Operations: SLT, ADD (A plus B plus Cin), SUB (A minus B minus Cin), AND, XOR, OR, COMPARE (equality)

Performance: left unspecified for now!

(2) **Block Diagram**

Understand the data and control flows

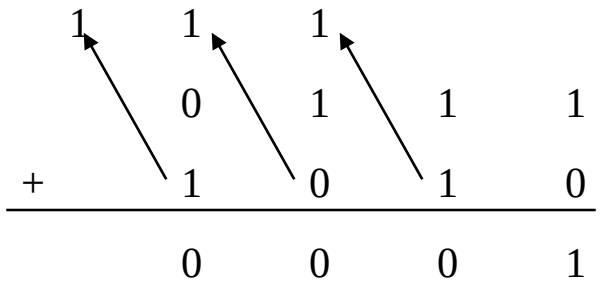


Review: Two's Complement Arithmetic

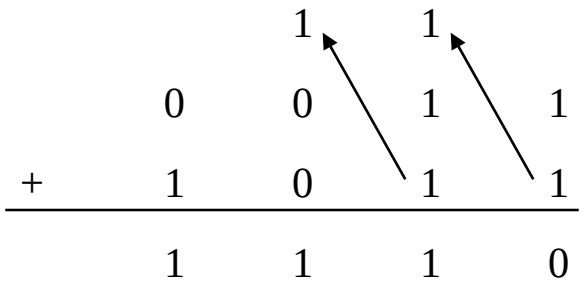
Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Decimal	2's Complement
0	0000
-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001
-8	1000

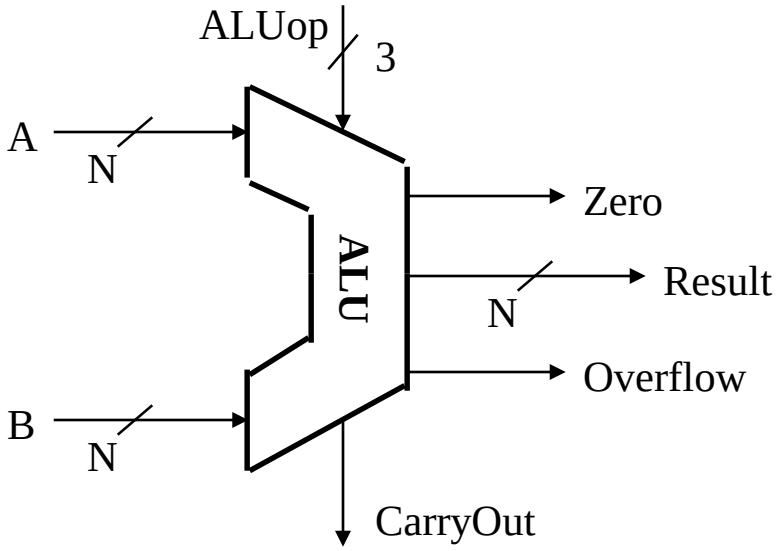
- Examples: 7 - 6 = 7 + (- 6) = 1



3 - 5 = 3 + (- 5) = -2



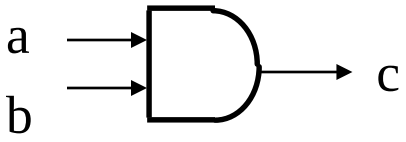
Block Diagram of the ALU



- ALU Control Lines (ALUop) — Function
  - 000 — And
  - 001 — Or
  - 010 — Add
  - 110 — Subtract
  - 111 — Set-on-less-than

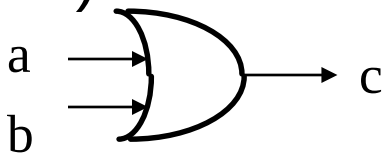
4 Hardware Building Blocks

- AND gate (c = a • b)



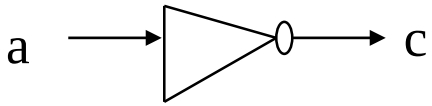
a	b	c = a • b
0	0	0
1	0	0
0	1	0
1	1	1

- OR gate (c = a + b)



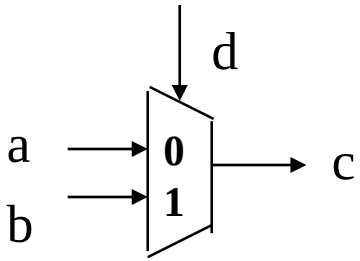
a	b	c = a + b
0	0	0
1	0	1
0	1	1
1	1	1

- Inverter (c = a')



a	c = a'
0	1
1	0

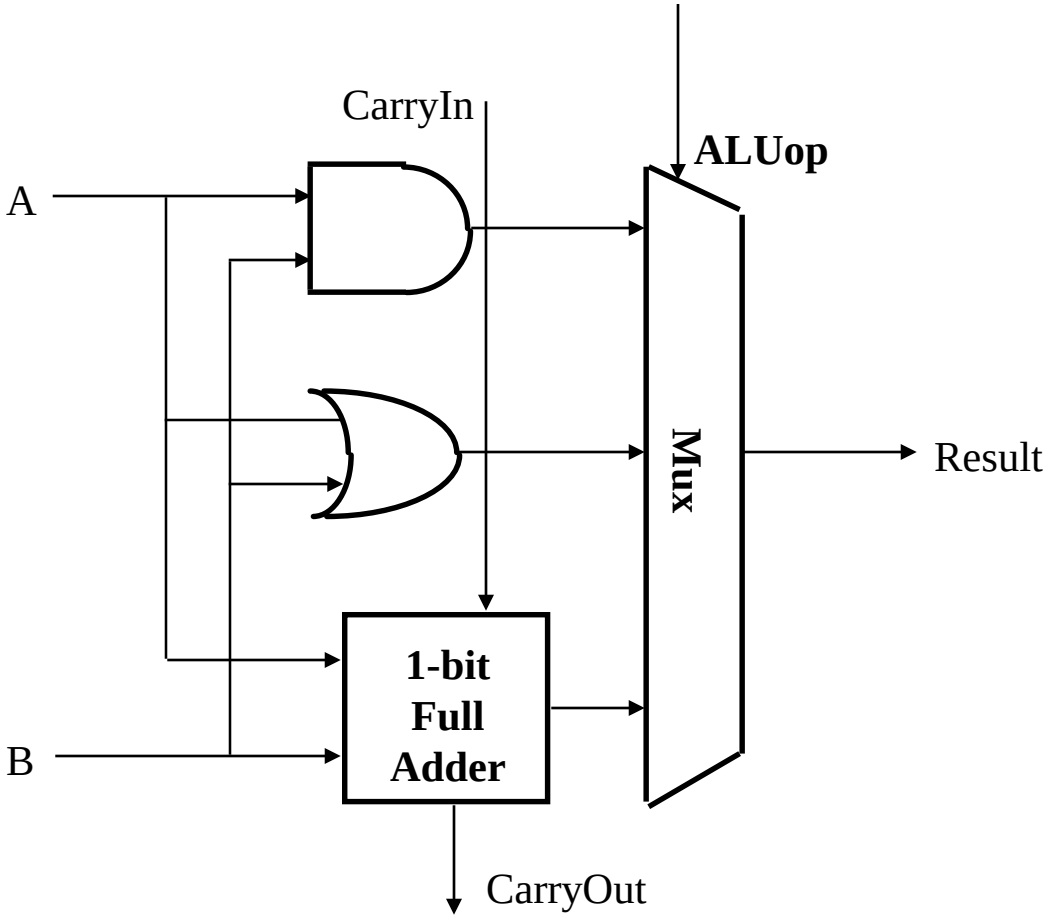
- Multiplexer
  if d==0, c=a;
  otherwise c= b



d	c
0	a
1	b

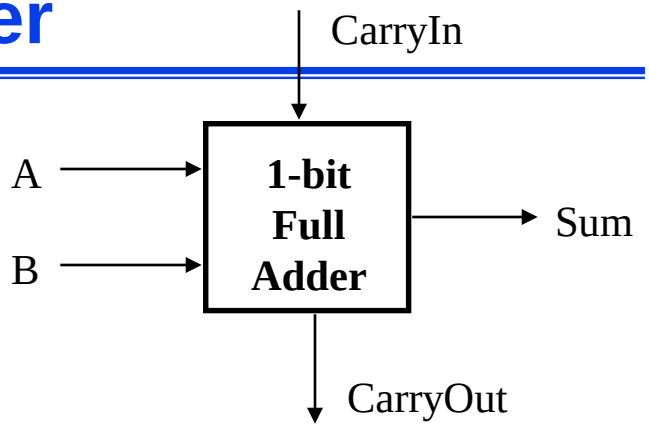
A One Bit ALU

- This 1-bit ALU will perform AND, OR, and ADD



Review: A One-bit Full Adder

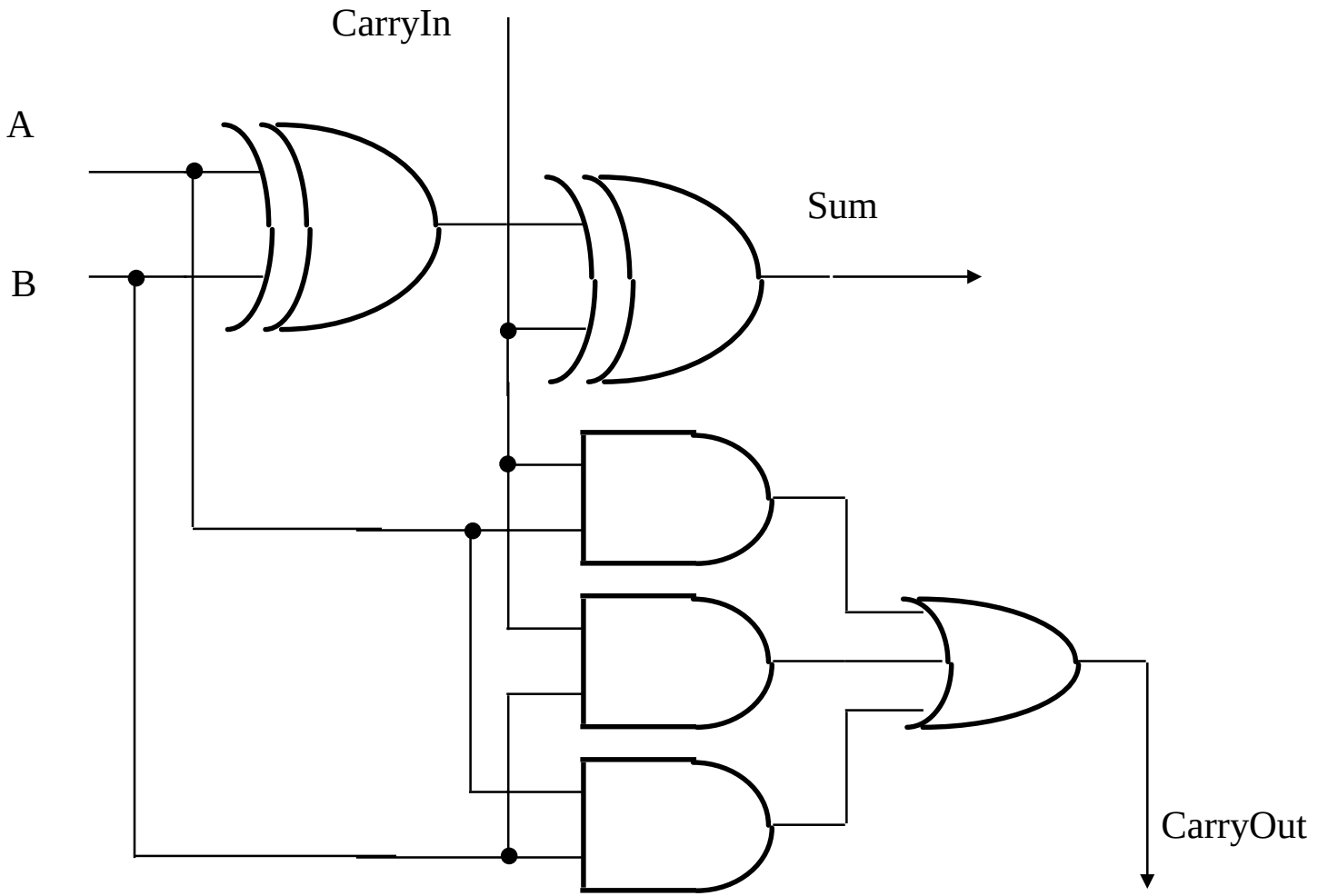
- 1 bit full adder: a switching circuit which add together two binary digits (bits), and a third bit called a *CarryIn* bit which may have come from a previous full adder.



Inputs			Outputs		
A	**B**	**CarryIn**	**CarryOut**	**Sum**	**Comments**
0	0	0	0	0	0 + 0 + 0 = 00
0	0	1	0	1	0 + 0 + 1 = 01
0	1	0	0	1	0 + 1 + 0 = 01
0	1	1	1	0	0 + 1 + 1 = 10
1	0	0	0	1	1 + 0 + 0 = 01
1	0	1	1	0	1 + 0 + 1 = 10
1	1	0	1	0	1 + 1 + 0 = 10
1	1	1	1	1	1 + 1 + 1 = 11

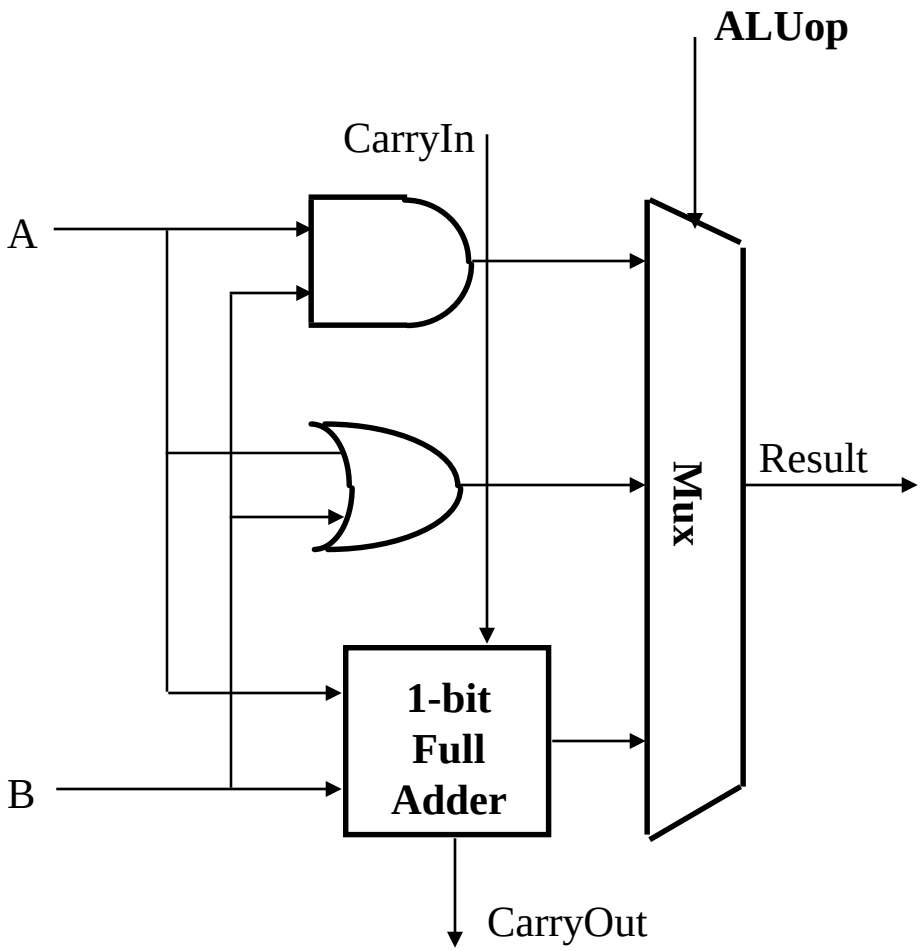
Review: A One-bit Full Adder

- Sum = A ⊕ B ⊕ CarryIn
- CarryOut = B • CarryIn + A • CarryIn + A • B

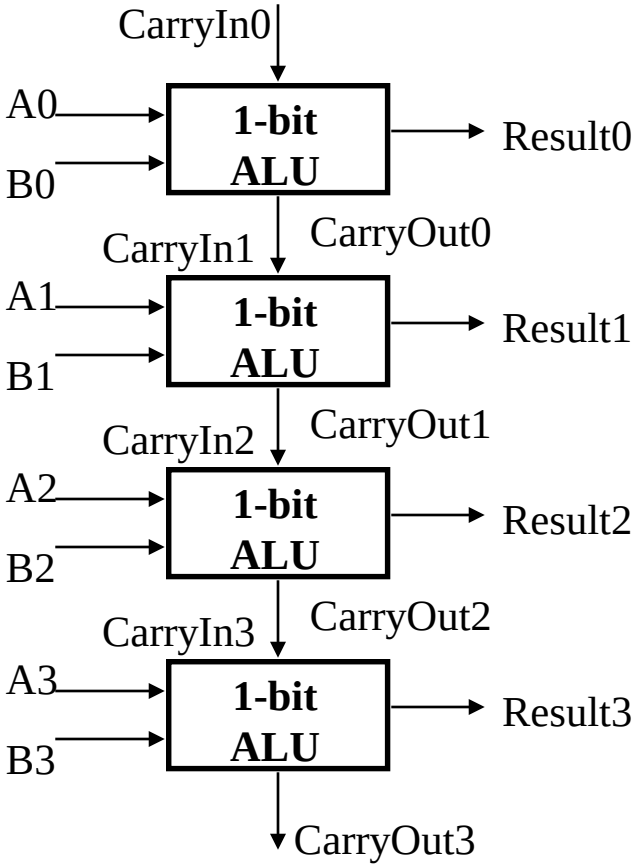


A 4-bit ALU

1-bit ALU

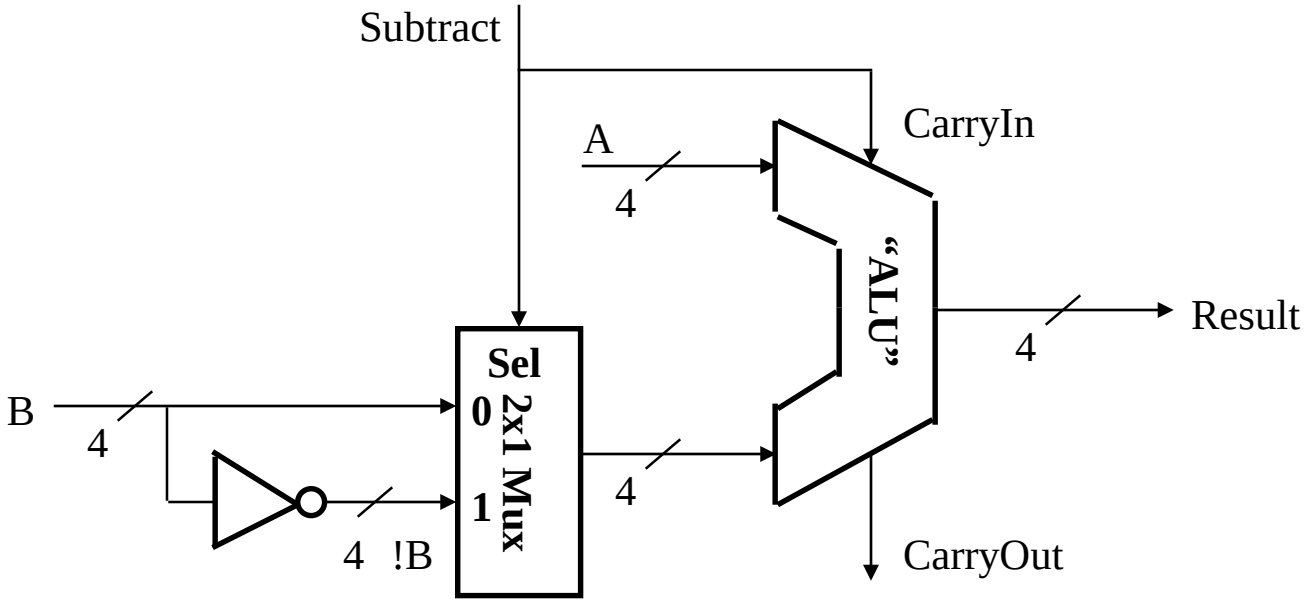


4-bit ALU



How About Subtraction?

- Keep in mind the followings:
  - (A - B) is the that as: A + (-B)
  - 2's Complement: Take the inverse of every bit and add 1
- Bit-wise inverse of B is B':
  - A + B' + 1 = A + (B' + 1) = A + (-B) = A - B

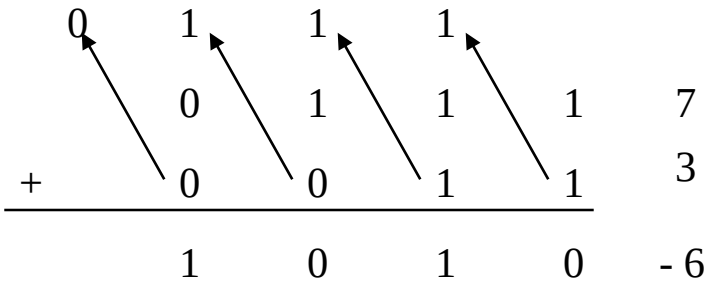


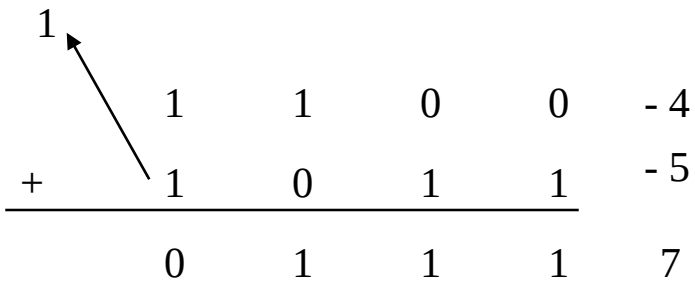
Overflow

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Decimal	2's Complement
0	0000
-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001
-8	1000

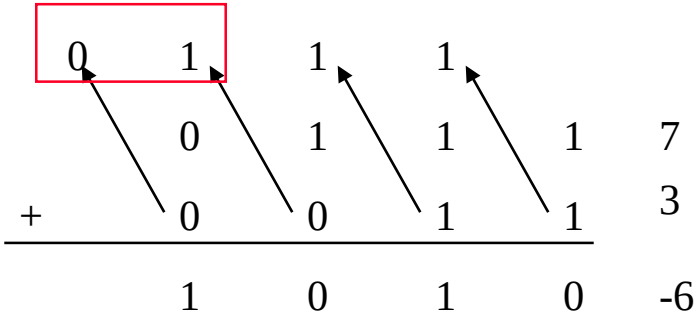
- Examples: 7 + 3 = 10 but ... - 4 - 5 = - 9 but ...

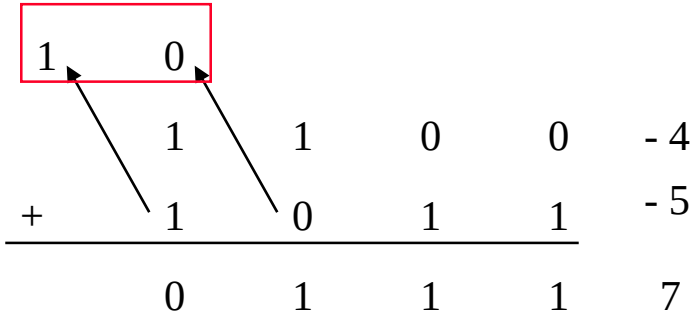




Overflow Detection

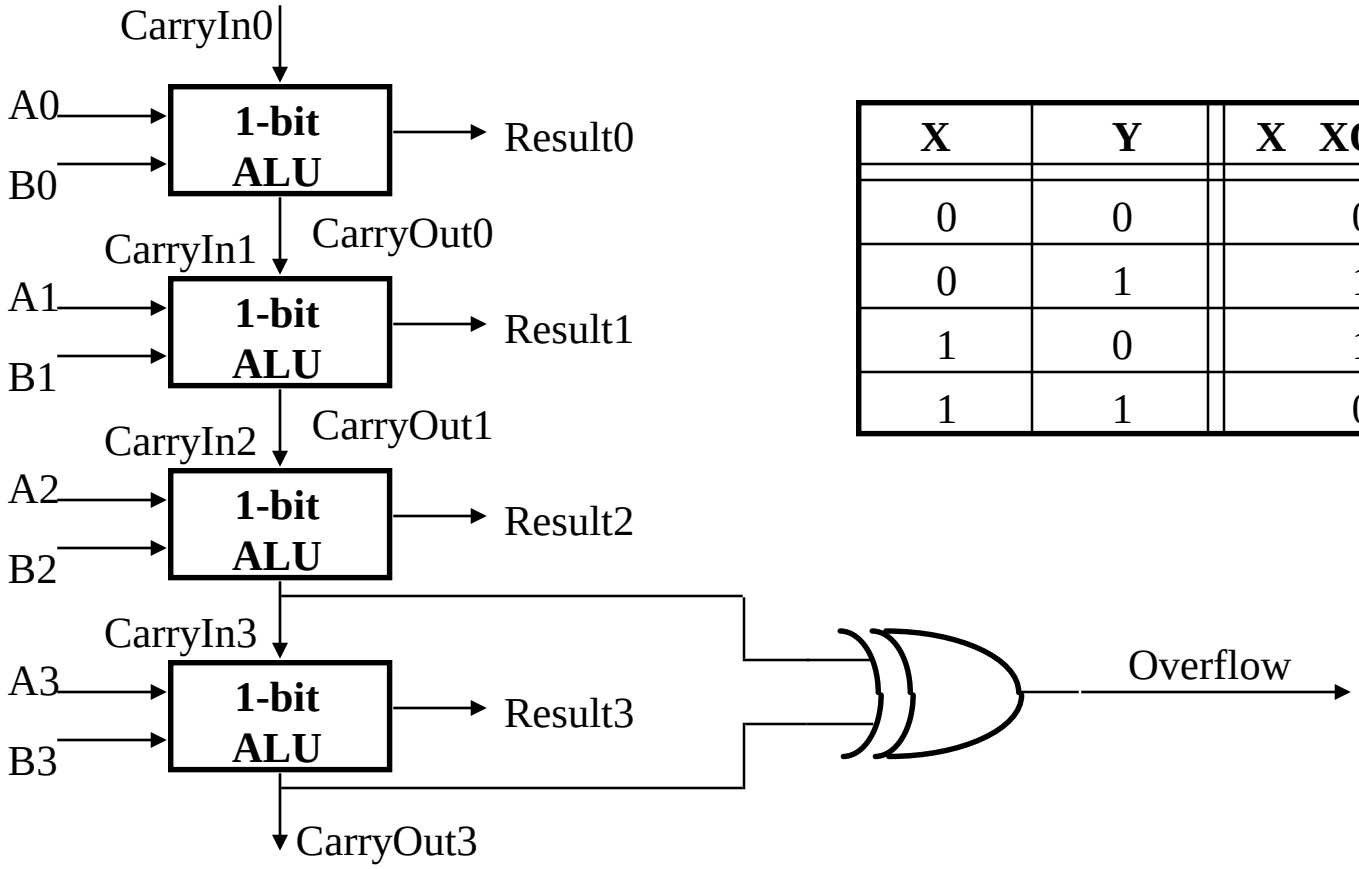
- Overflow: the result is too large (or too small) to represent properly
  - Example: - 8 < = 4-bit binary number <= 7
- Can overflow happen when adding operands with different signs?
- Overflow occurs when adding:
  - 2 positive numbers and the sum is negative
  - 2 negative numbers and the sum is positive
- Exercise: Prove you can detect overflow by:
  - Carry into MSB ! = Carry out of MSB





Overflow Detection Logic

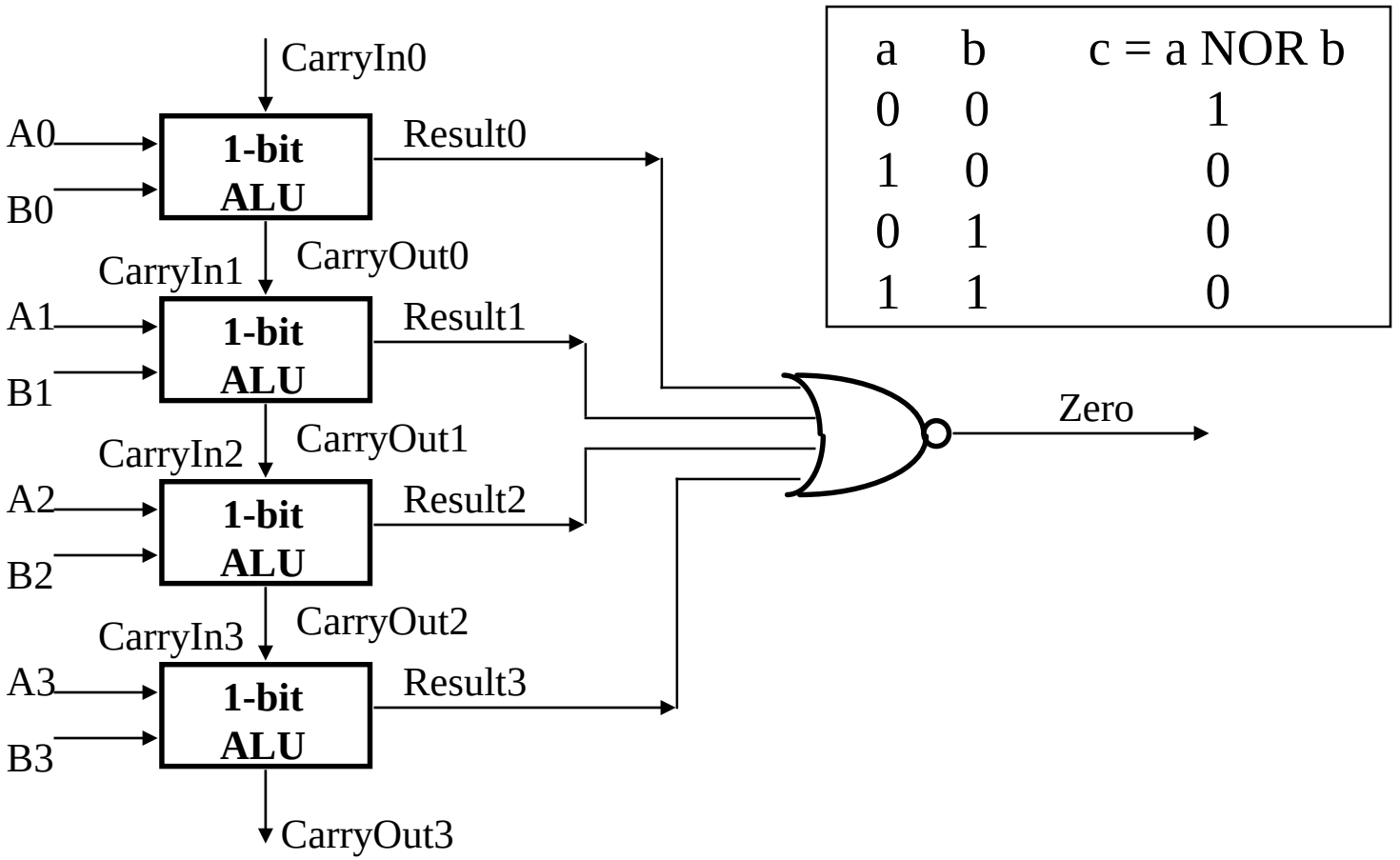
- Carry into MSB ! = Carry out of MSB
  - For a N-bit ALU: Overflow = CarryIn[N - 1] XOR CarryOut[N - 1]



X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

Zero Detection Logic

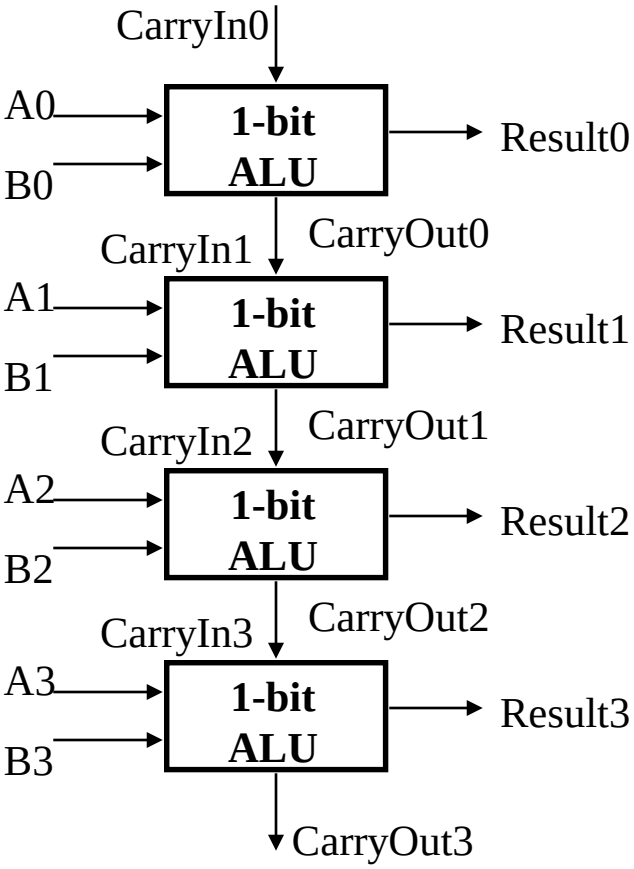
- A = B is the same as A - B = 0
- Zero Detection Logic is just a one BIG NOR gate
  - Any non-zero input to the NOR gate will cause its output to be zero

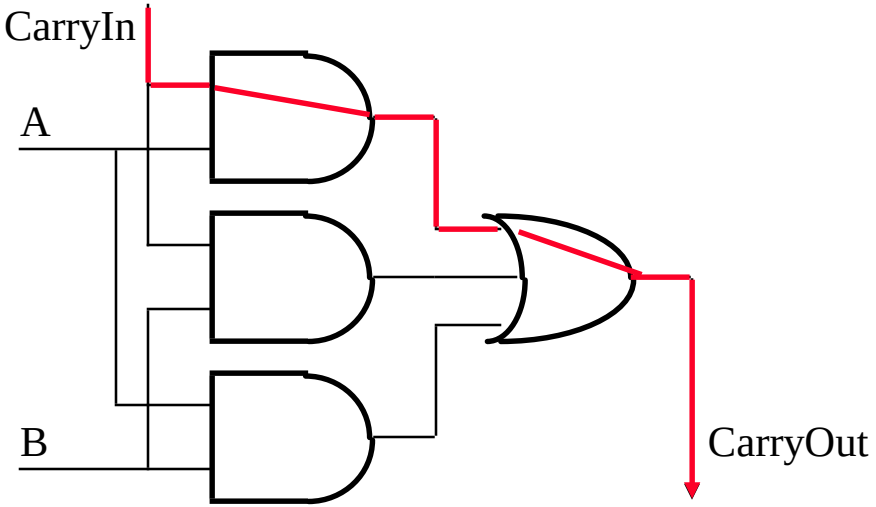


a	b	c = a NOR b
0	0	1
1	0	0
0	1	0
1	1	0

The Disadvantage of Ripple Carry

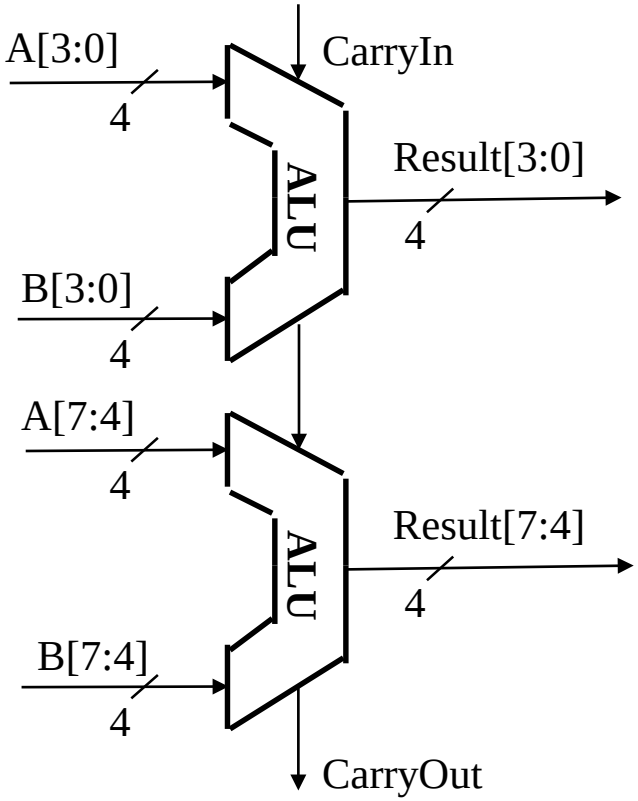
- The adder we just built is called a “Ripple Carry Adder”
  - The carry bit may have to propagate from LSB to MSB
  - Worst case delay for a N-bit adder: 2N-gate delay





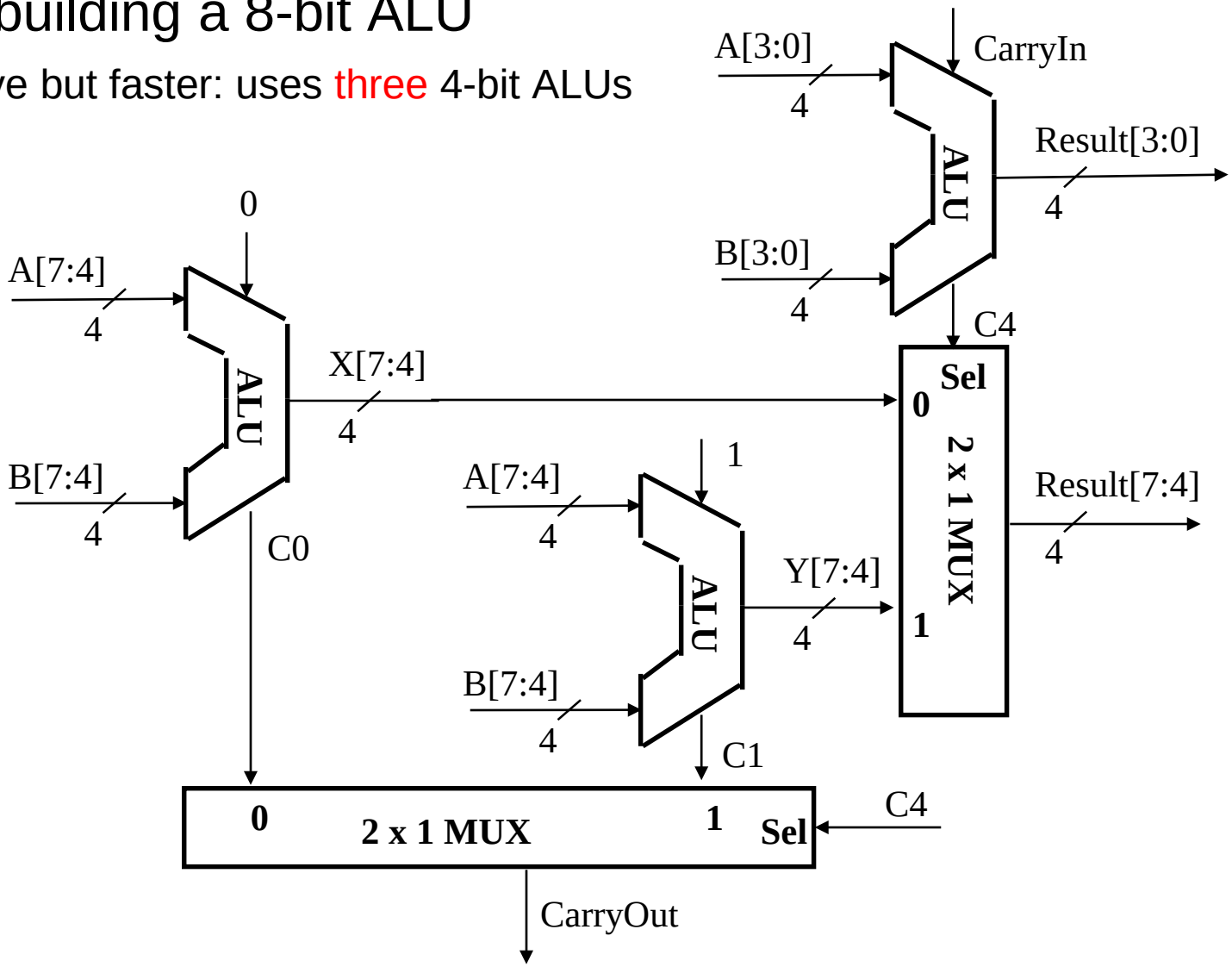
Carry Select Header

- Consider building a 8-bit ALU
  - Simple: connects two 4-bit ALUs in series

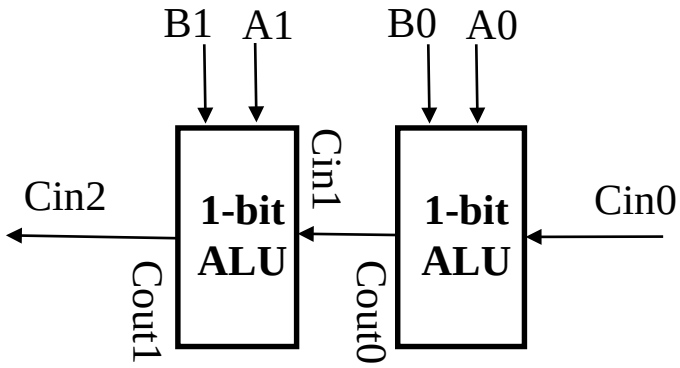


Carry Select Header (Continue)

- Consider building a 8-bit ALU
  - Expensive but faster: uses three 4-bit ALUs



The Theory Behind Carry Lookahead

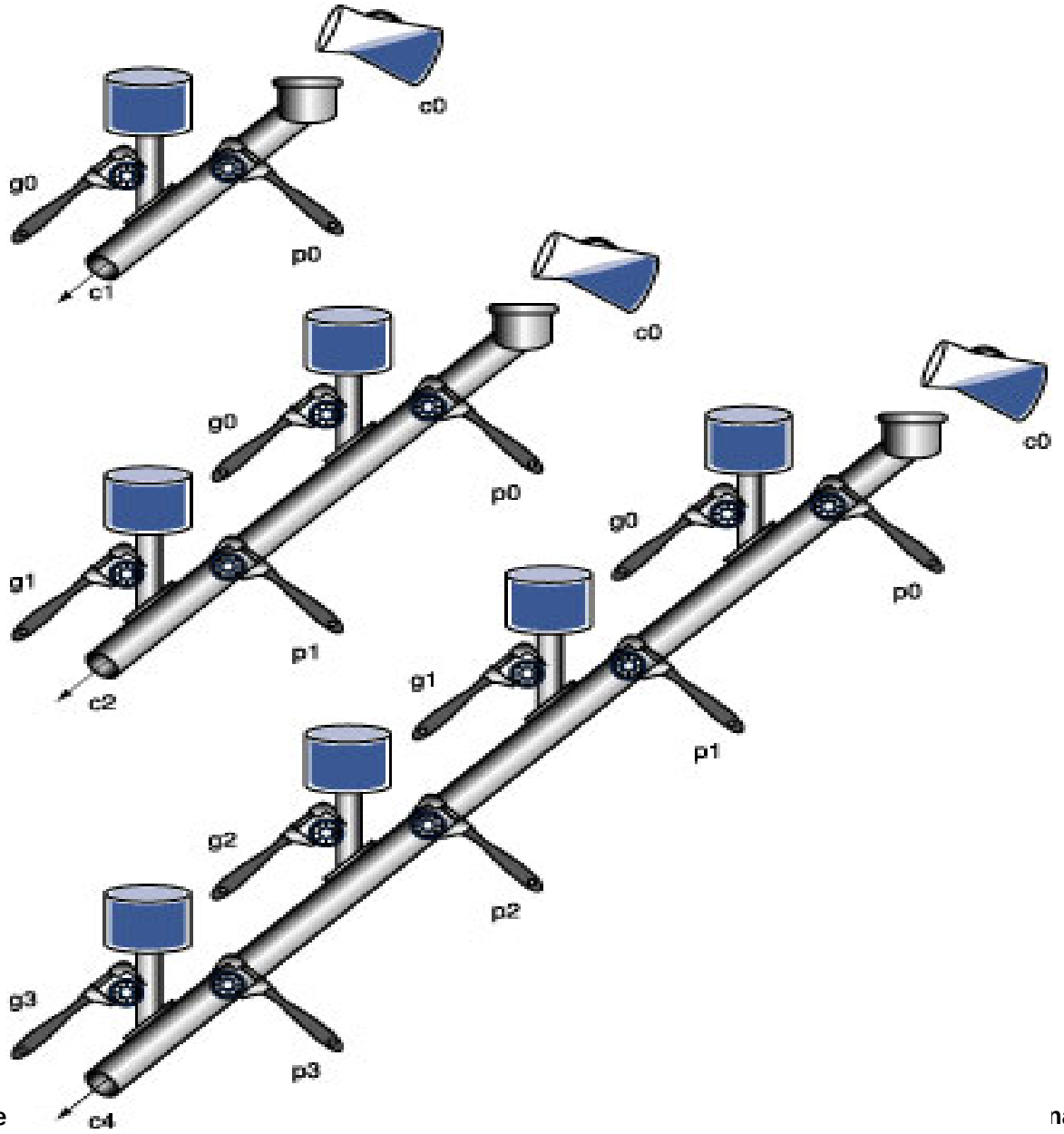


- Recall: CarryOut = (B • CarryIn) + (A • CarryIn) + (A • B)
  - Cin2 = Cout1 = (B1 • Cin1) + (A1 • Cin1) + (A1 • B1)
  - Cin1 = Cout0 = (B0 • Cin0) + (A0 • Cin0) + (A0 • B0)
- Substituting Cin1 into Cin2:
  - Cin2 = (A1 • A0 • B0) + (A1 • A0 • Cin0) + (A1 • B0 • Cin0) + (B1 • A0 • B0) + (B1 • B0 • Cin0) + (B1 • A0 • Cin0) + (A1 • B1)
- Now define two new terms:
  - Generate Carry at Bit i gi = Ai • Bi
  - Propagate Carry via Bit i pi = Ai + Bi

The Theory Behind Carry Lookahead (Continue)

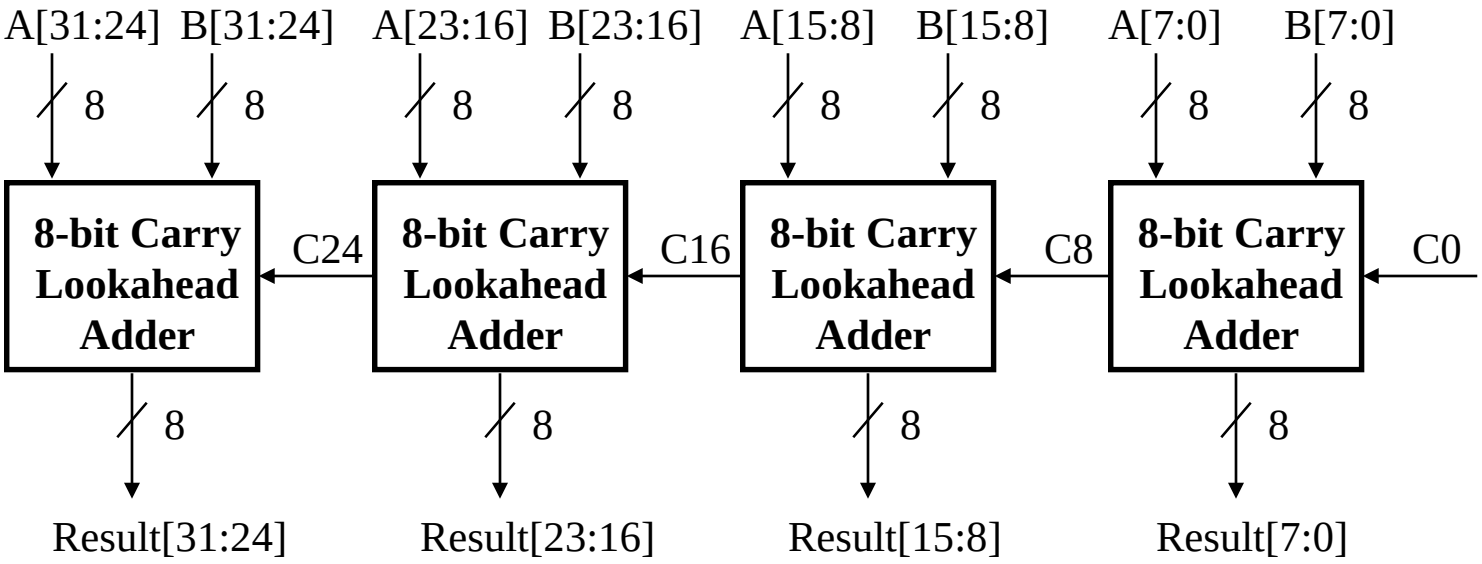
- Using the two new terms we just defined:
  - Generate Carry at Bit i gi = Ai • Bi
  - Propagate Carry via Bit i pi = Ai + Bi
- We can rewrite:
  - Cin1 = g0 + (p0 • Cin0)
  - Cin2 = g1 + (p1 • g0) + (p1 • p0 • Cin0)
  - Cin3 = g2 + (p2 • g1) + (p2 • p1 • g0) + (p2 • p1 • p0 • Cin0)
- Carry going into bit 3 is 1 if
  - We generate a carry at bit 2 (g2)
  - Or we generate a carry at bit 1 (g1) and bit 2 allows it to propagate (p2 & g1)
  - Or we generate a carry at bit 0 (g0) and bit 1 as well as bit 2 allows it to propagate (p2 & p1 & g0)
  - Or we have a carry input at bit 0 (Cin0) and bit 0, 1, and 2 all allow it to propagate (p2 & p1 & p0 & Cin0)

The Theory Behind Carry Lookahead (Continue)



A Partial Carry Lookahead Adder

- It is very expensive to build a “full” carry lookahead adder
  - Just imagine the length of the equation for Cin31
- Common practices:
  - Connects several N-bit Lookahead Adders to form a big adder
  - Example: connects four 8-bit carry lookahead adders to form a 32-bit partial carry lookahead adder



Summary

- An Overview of the Design Process
  - Design is an iterative process-- successive refinement
  - Do NOT wait until you know everything before you start
- Binary Arithmetics
  - If you use 2's complement representation, subtract is easy.
- ALU Design
  - Designing a Simple 4-bit ALU
    - How to implement SLT operation?
  - Other ALU Construction Techniques