Chapter 3

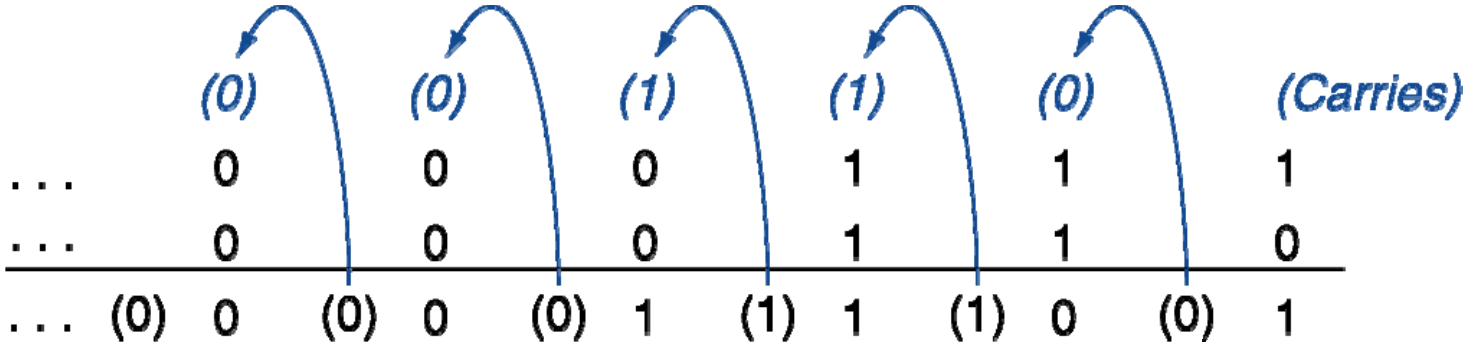
Arithmetic for Computers

Arithmetic for Computers

- Operations on integers
  - Addition and subtraction
  - Multiplication and division
  - Dealing with overflow
- Floating-point real numbers
  - Representation and operations

Integer Addition

- Example: 7 + 6



- Overflow if result out of range
  - Adding +ve and –ve operands, no overflow
  - Adding two +ve operands
    - Overflow if result sign is 1
  - Adding two –ve operands
    - Overflow if result sign is 0

Integer Subtraction

- Add negation of second operand
- Example: 7 – 6 = 7 + (–6)

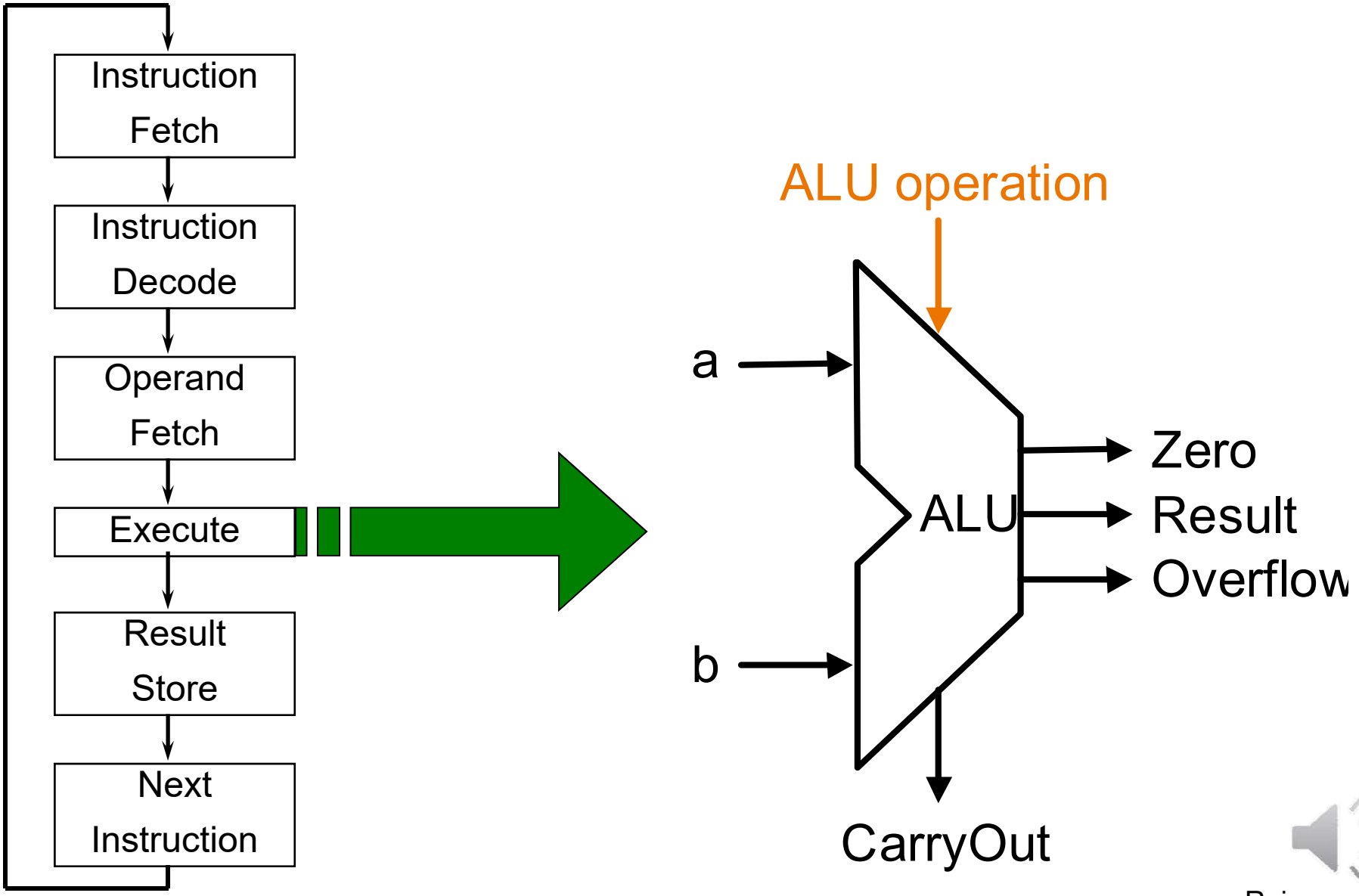
+7:	0000 0000 … 0000 0111
–6:	1111 1111 … 1111 1010
+1:	0000 0000 … 0000 0001

- Overflow if result out of range
  - Subtracting two +ve or two –ve operands, no overflow
  - Subtracting +ve from –ve operand
    - Overflow if result sign is 0
  - Subtracting –ve from +ve operand
    - Overflow if result sign is 1

Dealing with Overflow

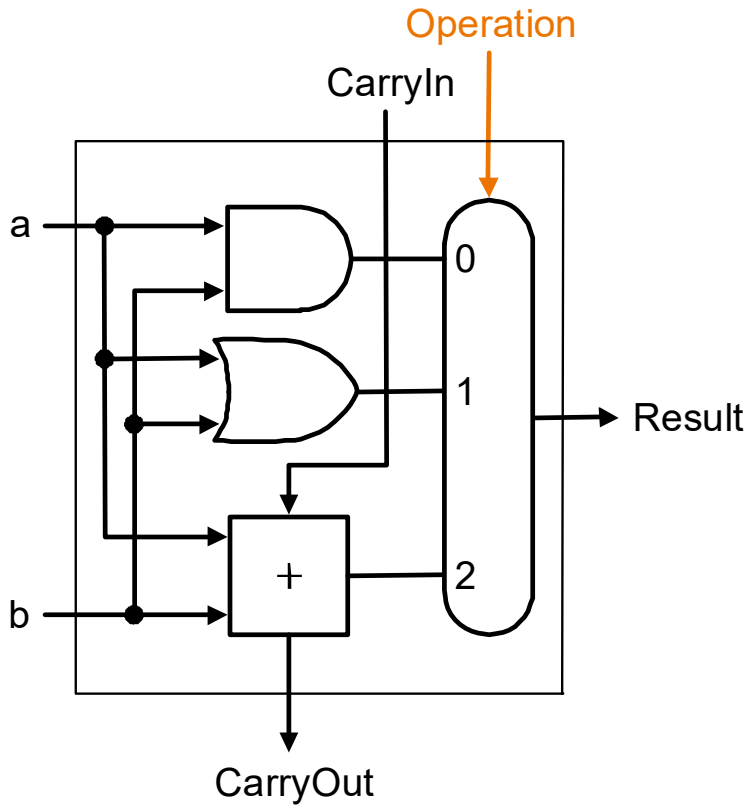
- Some languages (e.g., C) ignore overflow
  - Use MIPS `addu`, `addui`, `subu` instructions
- Other languages (e.g., Ada, Fortran) require raising an exception
  - Use MIPS `add`, `addi`, `sub` instructions
  - On overflow, invoke exception handler
    - Save PC in exception program counter (EPC) register
    - Jump to predefined handler address
    - `mfc0` (move from coprocessor reg) instruction can retrieve EPC value, to return after corrective action

Arithmetic Logic Unit Design



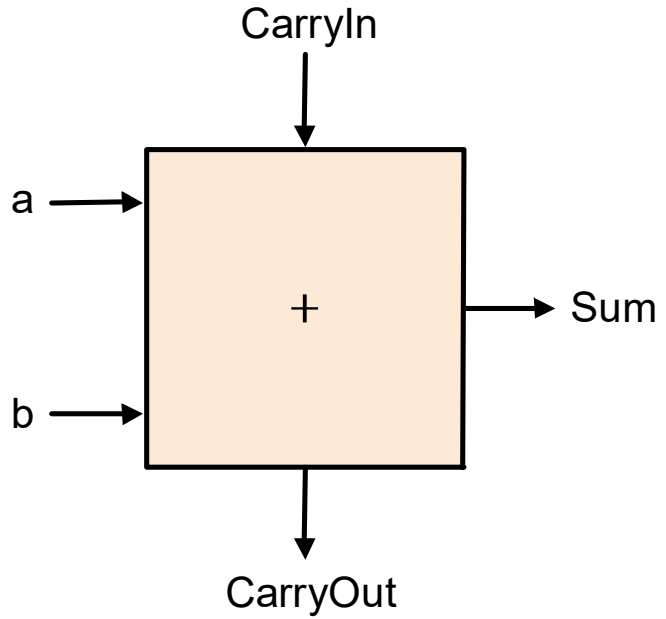
One Bit ALU

- Performs AND, OR, and ADD
  - on 1-bit operands
  - components:
    - AND gate
    - OR gate
    - 1-bit adder
    - Multiplexor



One Bit Full Adder

- Also known as a (3,2) adder
- Half Adder
  - no CarryIn



Inputs			Outputs		Comments
a	b	CarryIn	CarryOut	Sum	
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+0+1=10
1	1	1	1	1	1+1+1=11

CarryOut Logic Equation

- CarryOut = (!a & b & CarryIn) | (a & !b & CarryIn)
  | (a & b & !CarryIn) | (a & b & CarryIn)

- CarryOut = (b & CarryIn) | (a & CarryIn) | (a & b)

Inputs			Outputs		Comments
a	b	CarryIn	CarryOut	Sum	
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+0+1=10
1	1	1	1	1	1+1+1=11

Sum Logic Equation

- Sum = (!a & !b & CarryIn) | (!a & b & !CarryIn)
  | (a & !b & !CarryIn) | (a & b & CarryIn)

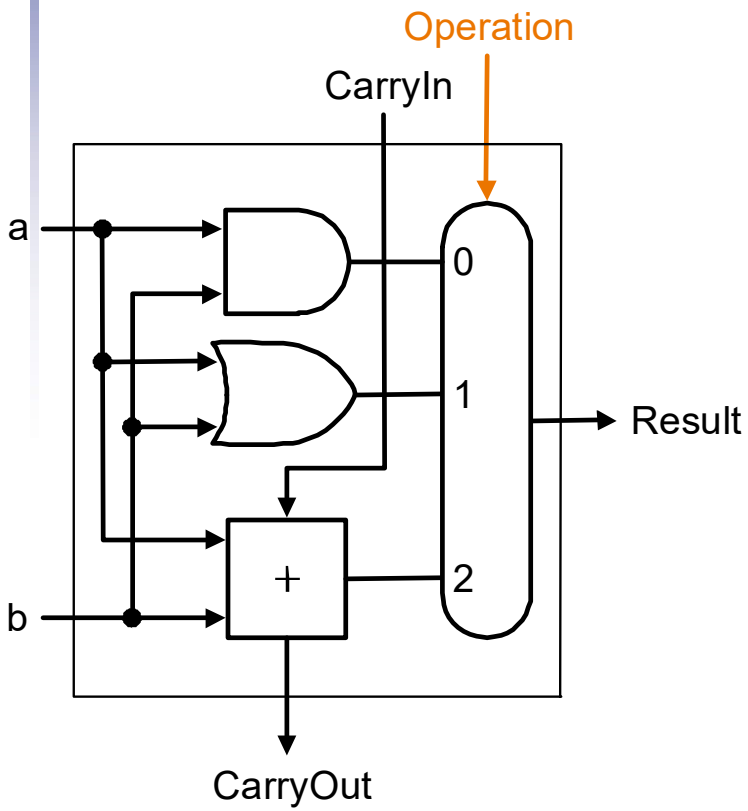
Inputs			Outputs		Comments
a	b	CarryIn	CarryOut	Sum	
0	0	0	0	0	0+0+0=00
0	0	1	0	1	0+0+1=01
0	1	0	0	1	0+1+0=01
0	1	1	1	0	0+1+1=10
1	0	0	0	1	1+0+0=01
1	0	1	1	0	1+0+1=10
1	1	0	1	0	1+0+1=10
1	1	1	1	1	1+1+1=11

Chapter 3

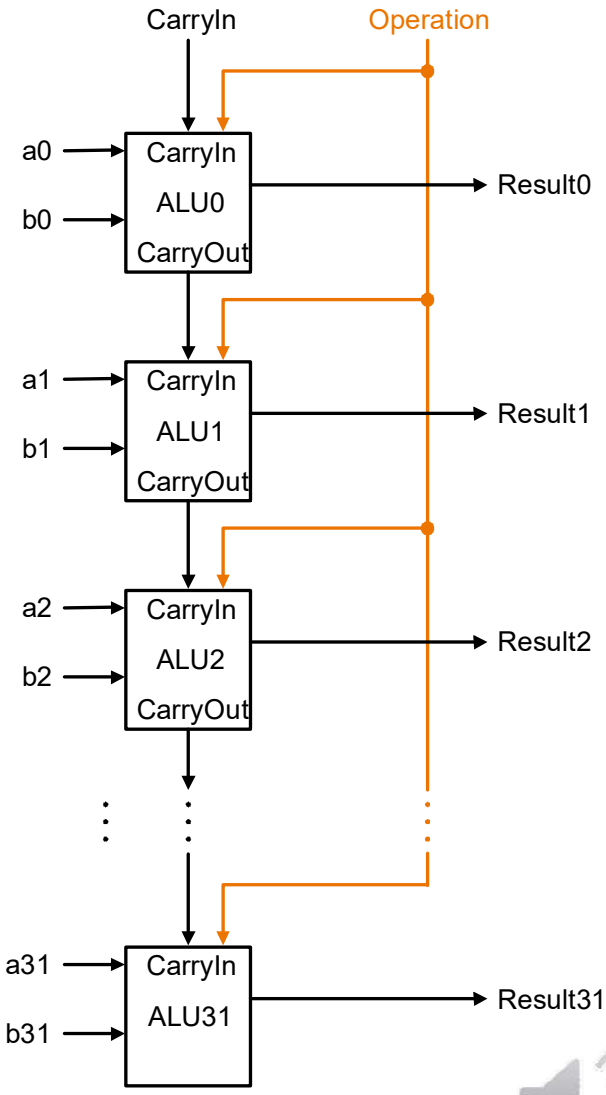
Arithmetic for Computers

32-bit ALU

- Ripple Carry ALU



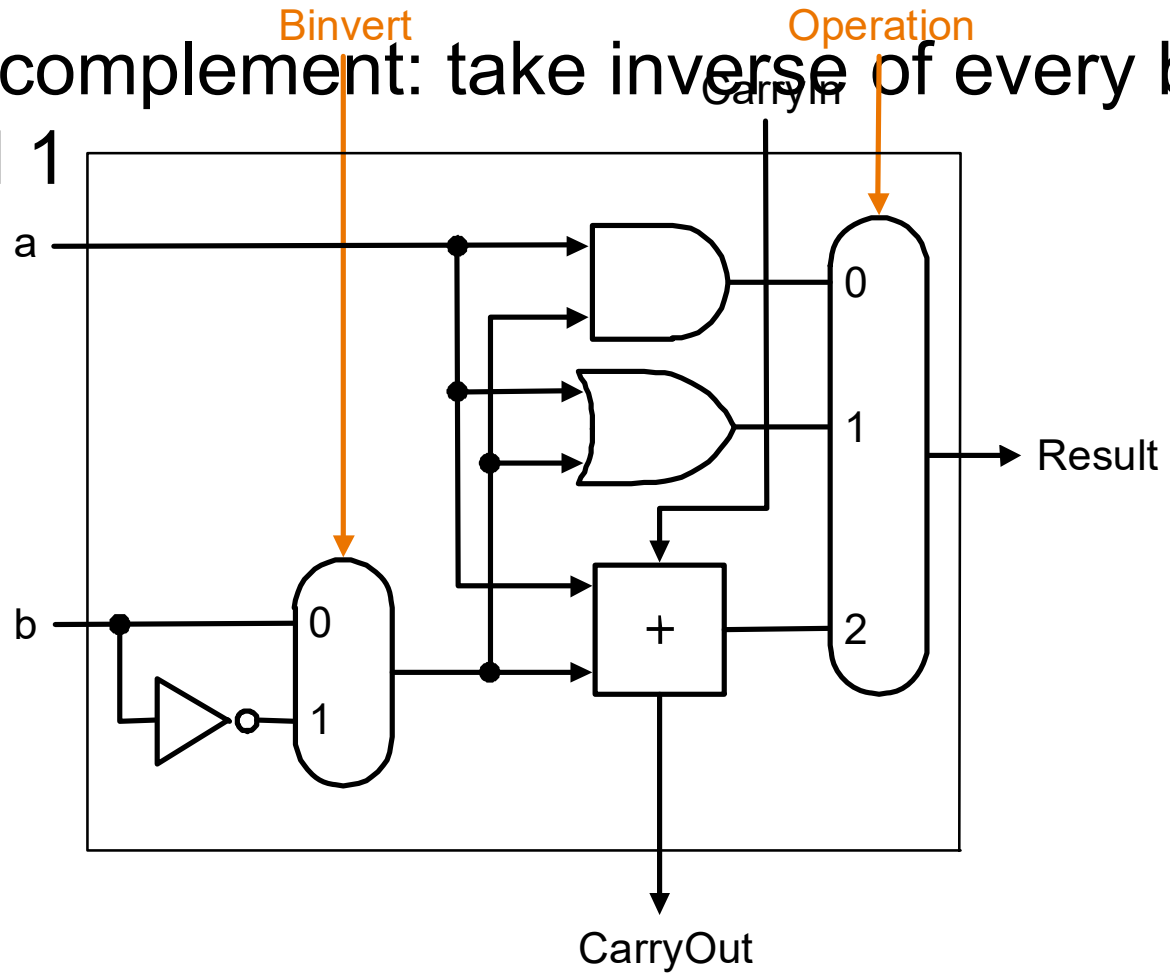
1-bit ALU



32-bit ALU

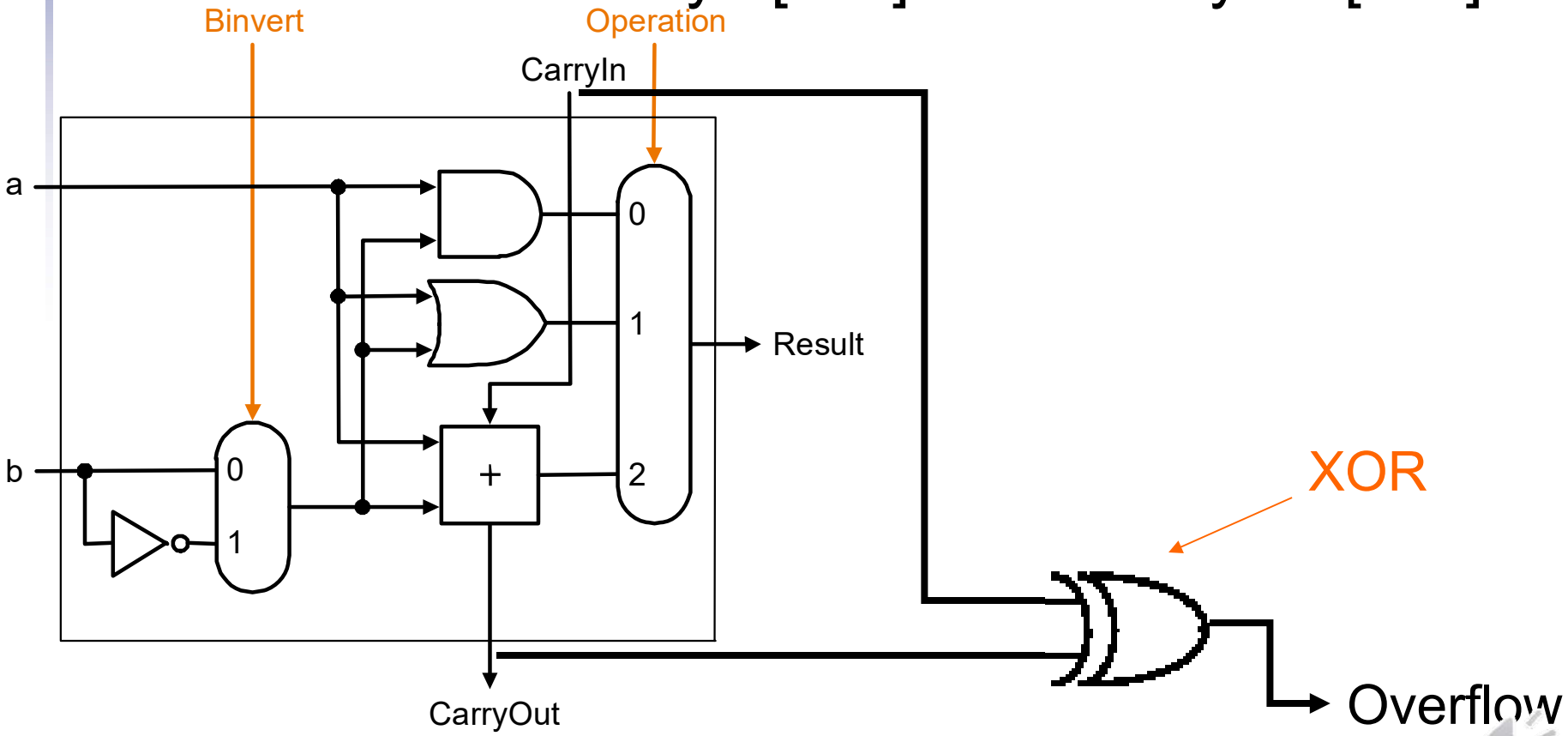
Subtraction?

- Expand our 1-bit ALU to include an inverter
  - 2's complement: take inverse of every bit and add 1



Overflow

- For N-bit ALU
  - Overflow = CarryIn[N-1] XOR CarryOut[N-1]

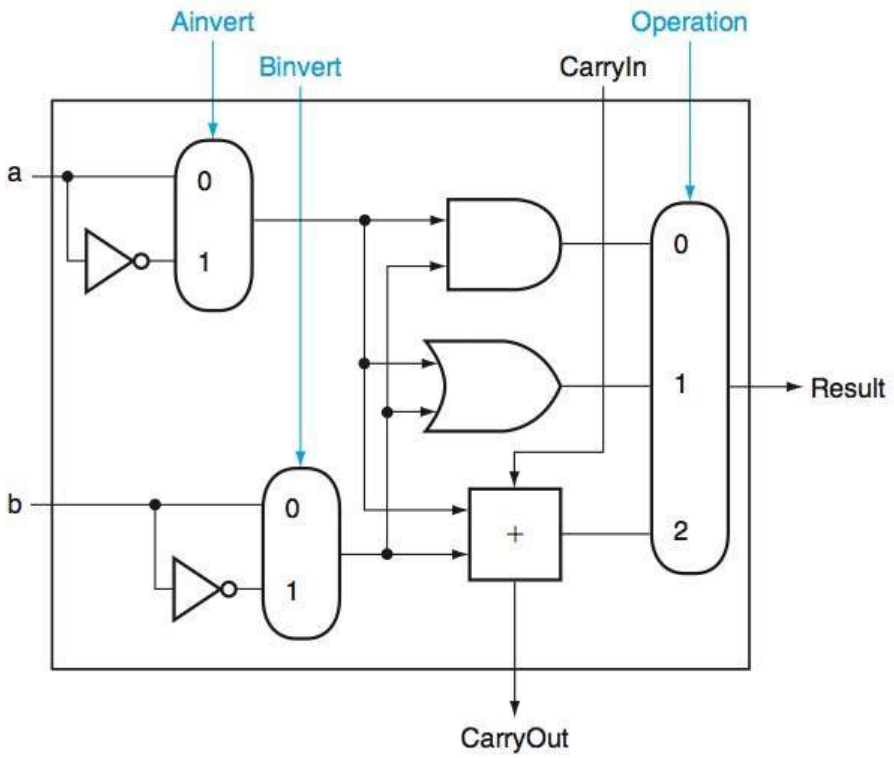


Most significant (N-1) bit ALU

Zero Detection

- Conditional Branches
- One big NOR gate
- Zero = ($Result_{N-1}$+$Result_{N-2}$+.... $Result_1$+$Result_0$)
- Any non-zero result will cause zero detection output to be zero

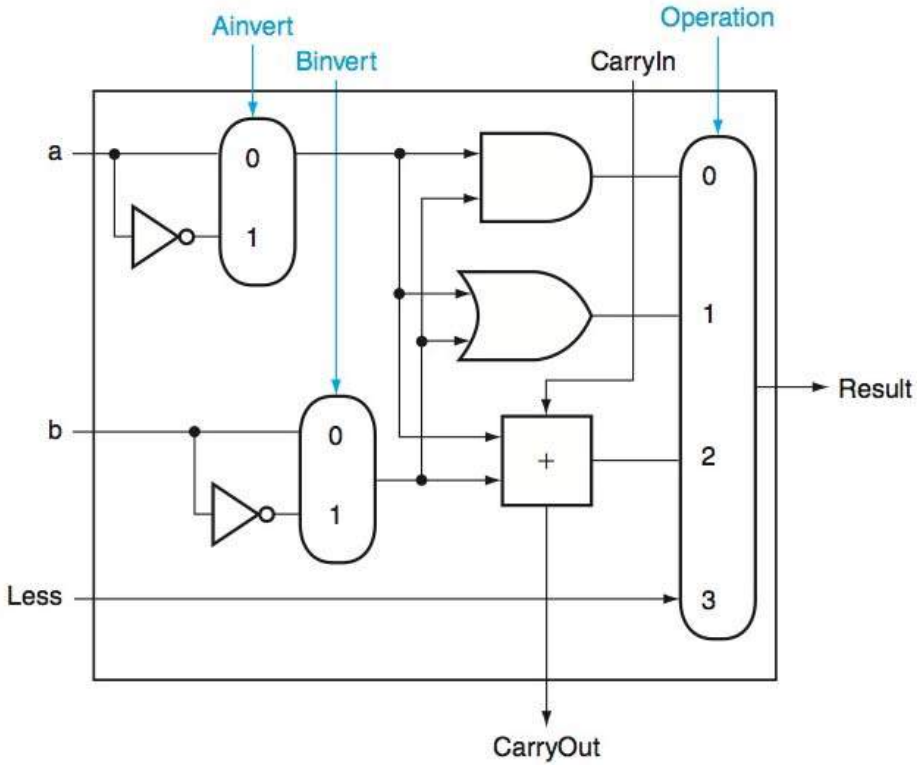
NOR


Ainvert
Binvert
CarryIn
Operation
a
b
0
1
0
1
0
1
2
+
Result
CarryOut

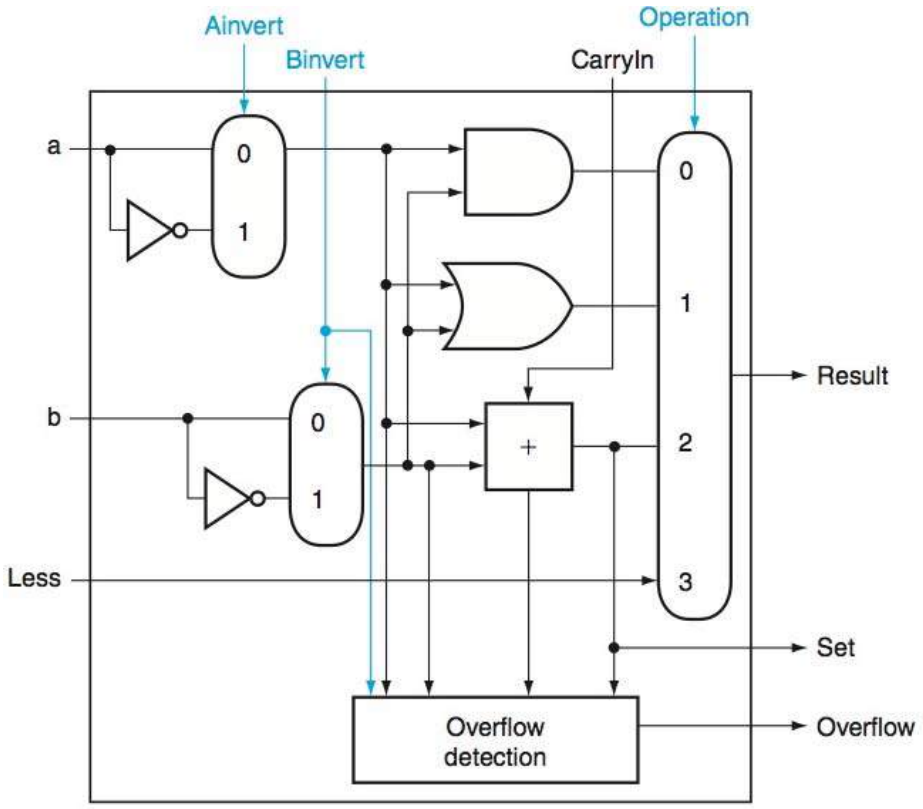
Set-On-Less-Than (SLT)

- SLT produces a 1 if rs < rt, and 0 otherwise
  - all but least significant bit will be 0
  - how do we set the least significant bit?
  - can we use subtraction?
    - rs - rt < 0
  - set the least significant bit to the sign-bit of (rs - rt)
- New input: LESS
- New output: SET

SLT Implementation

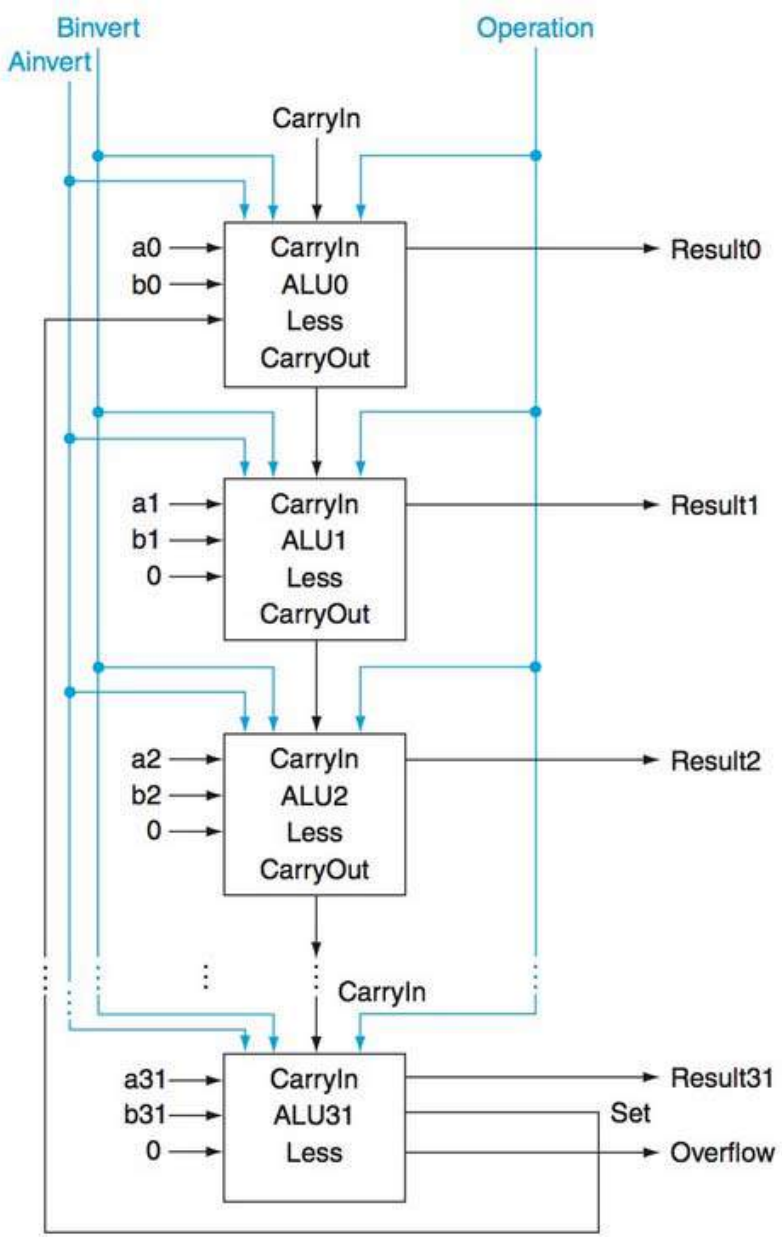


All but MSB



Most Significant Bit

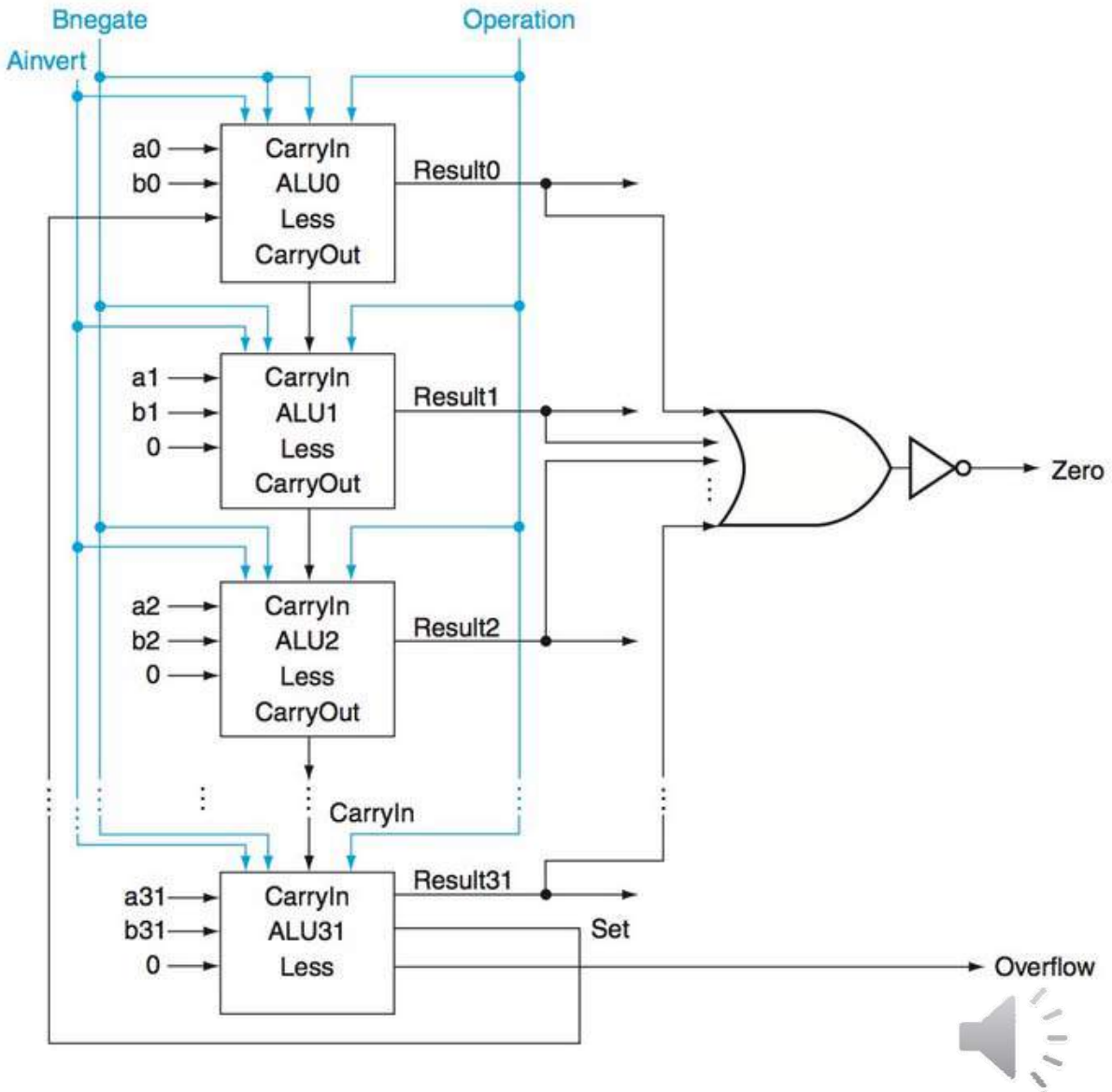
SLT Implementation



- Set of MSB is connected to Less of LSB!

Final ALU

- You should feel comfortable identifying what signals accomplish:
  - add
  - sub
  - and
  - or
  - nor
  - slt

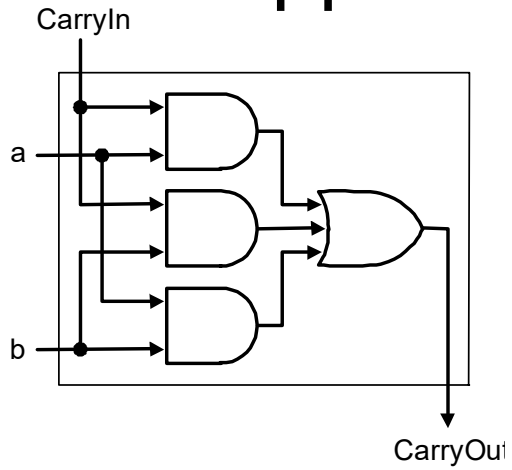


Chapter 3

Arithmetic for Computers

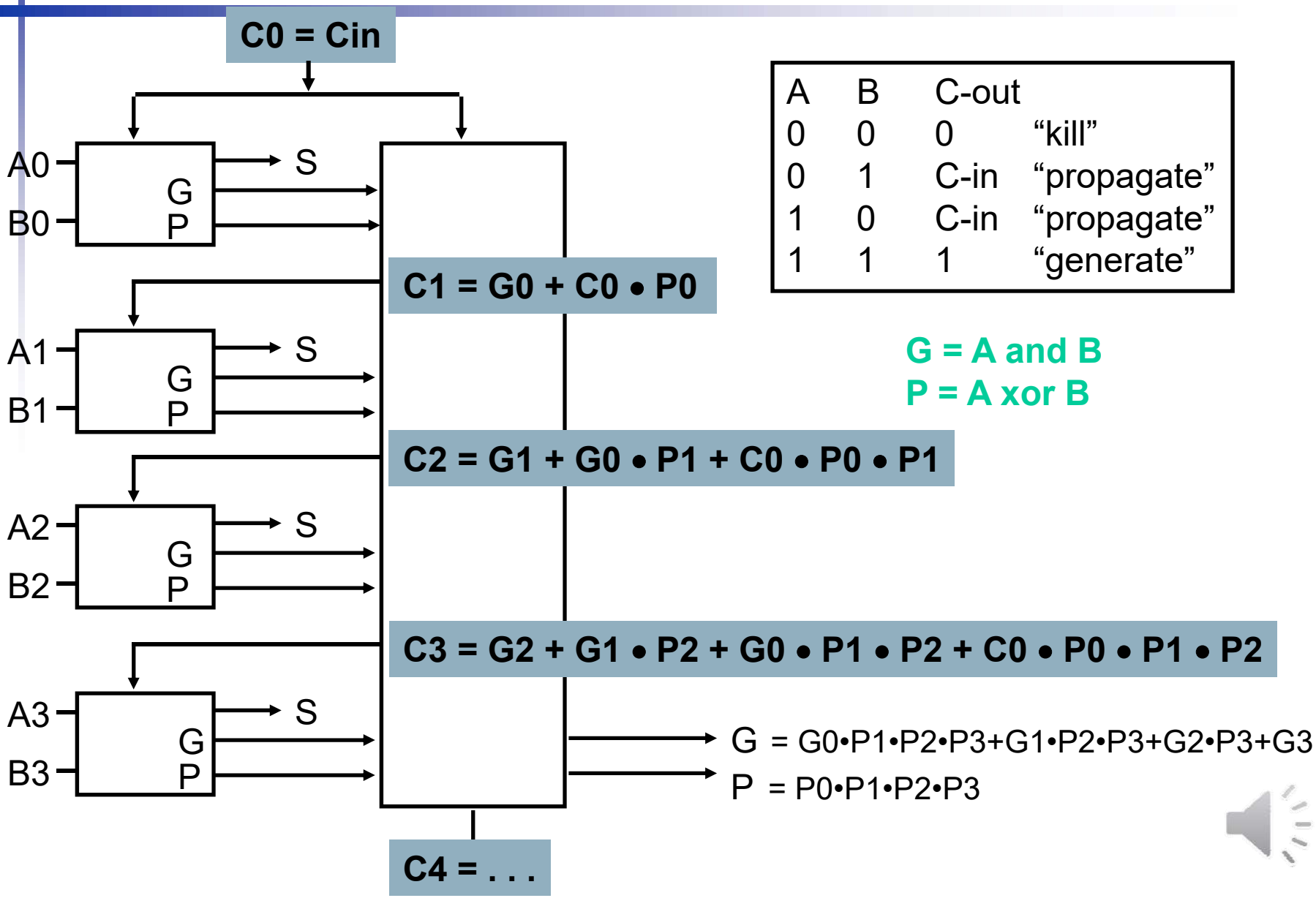
Can We Make a Faster Adder?

- Worst case delay for N-bit Ripple Carry Adder
  - 2N gate delays
  - 2 gates per CarryOut
  - N CarryOuts



- We will explore the Carry Lookahead Adder
  - Generate - Bit i creates new Carry
    - $g_i = A_i \& B_i$

Carry Look Ahead

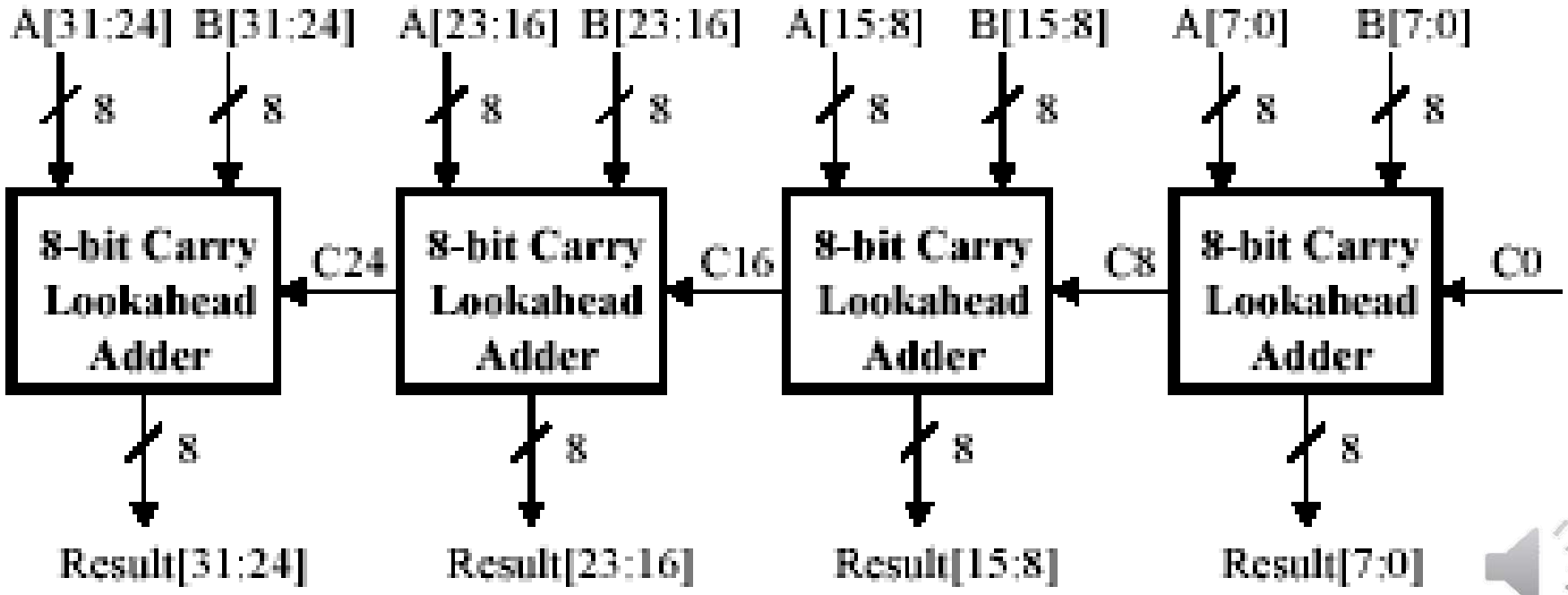


A	B	C-out	
0	0	0	“kill”
0	1	C-in	“propagate”
1	0	C-in	“propagate”
1	1	1	“generate”

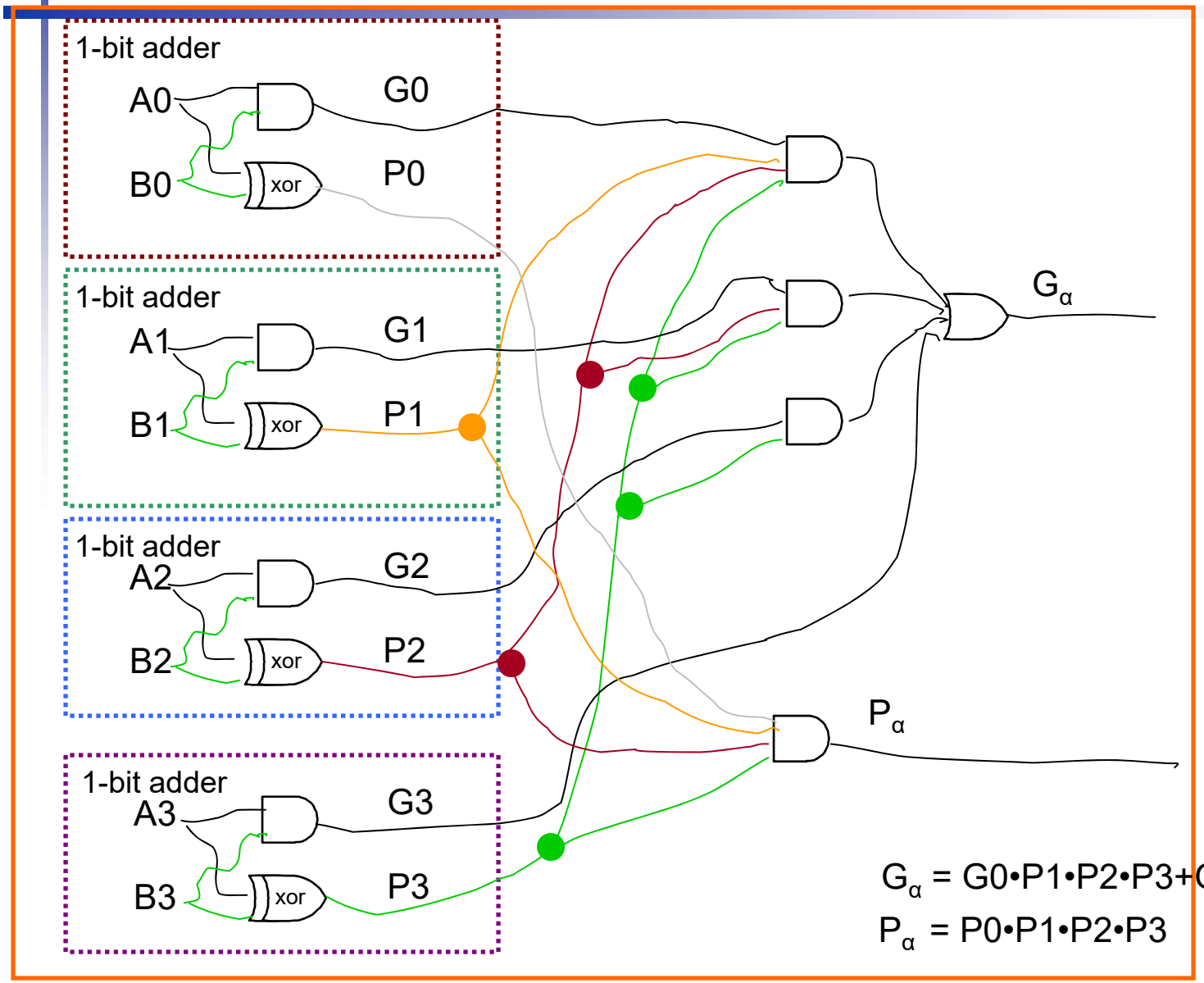
G = A and B
P = A xor B

Partial Carry Lookahead Adder

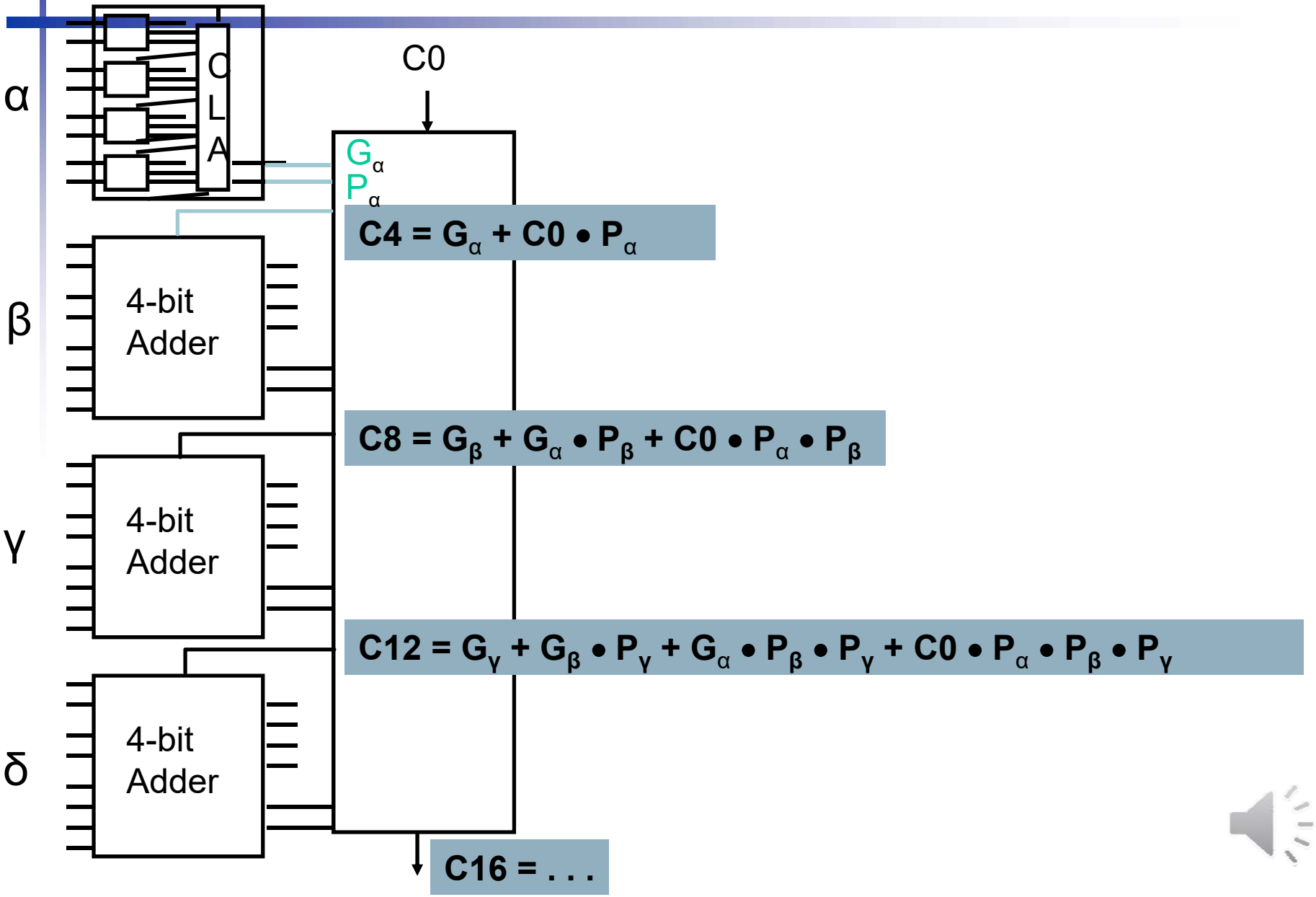
- Connect several N-bit Lookahead Adders together
- Four 8-bit carry lookahead adders can form a 32-bit partial carry lookahead adder



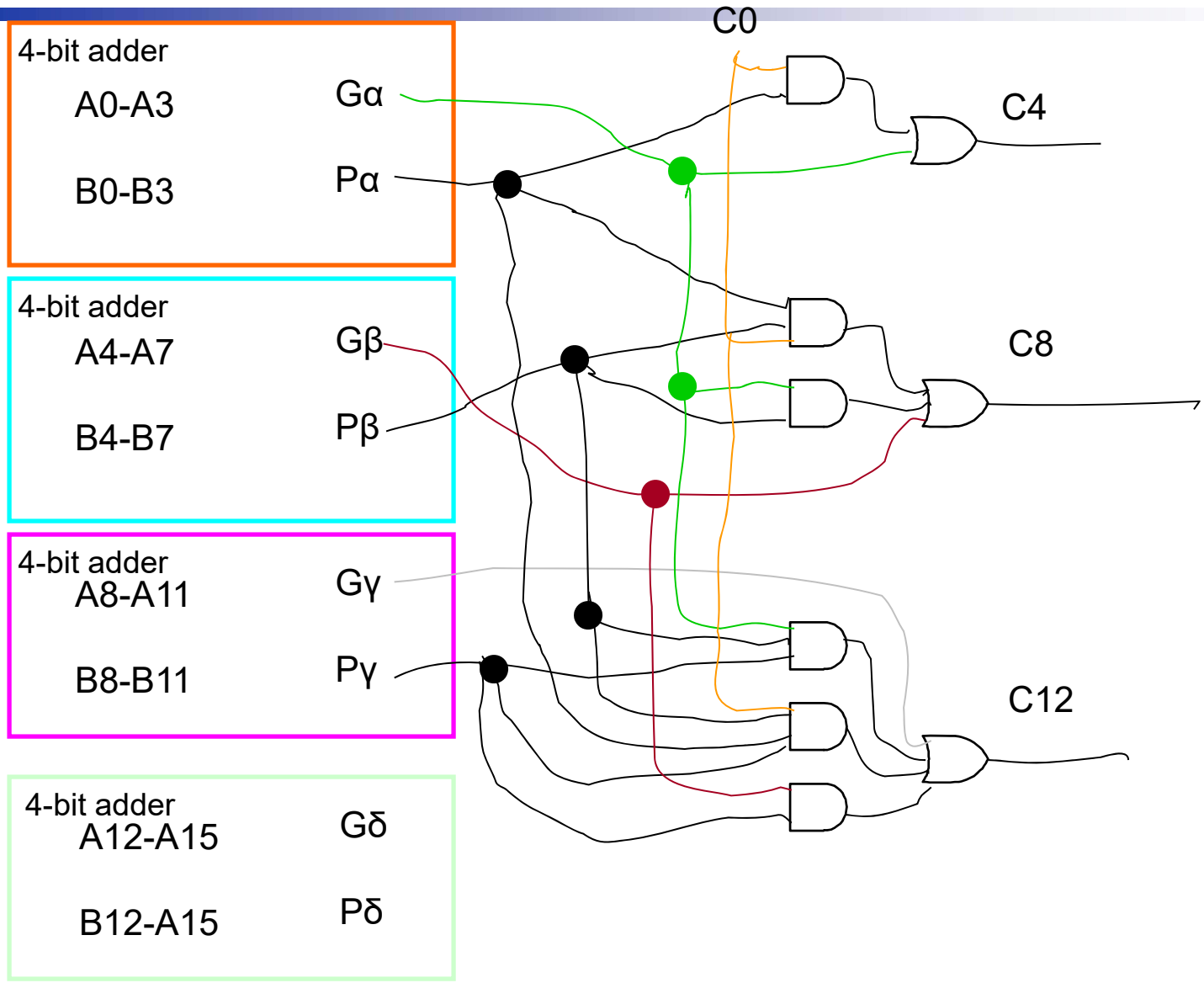
Generate and Propagate



Hierarchical CLA



Generate and Propagate



Carry Select Adder