CS240 Laboratory 3
Arithmetic Logic Unit (ALU)
and
Introduction to Memory

- **Signed Representation Review**
- **4-bit Addition with Ripple-Carry**
- **Arithmetic Logic Unit**
- **1-bit Memory Circuit (Latch)**

Signed Representation
Given n bits, the range of binary values which can be represented using:

Unsigned representation: $0 \rightarrow 2^{n} - 1$

Signed representation: $-2^{n-1} \rightarrow 2^{n-1} - 1$, MSB is used for sign

Two's Complement (signed representation):

Most significant /leftmost bit (0/positive, 1/negative)

Example: given a fixed number of 4 bits:
1000_2 is negative.
0111_2 is positive.

Overflow
Given a fixed number of n available bits, overflow occurs if a value cannot fit in n bits.

For example, given 4 bit representation:
The largest negative value we can represent is -8_{10} (1000_2)
The largest positive value we can represent is $+7_{10}$ (0111_2)

Overflow when Adding

An overflow occurs when adding two n-bit numbers if the result will not fit in n bits.

An overflow can be detected when:
-Two positive numbers added together yield a negative result, or
-Two negative numbers added together yield a positive result.

Overflow can also be detected when:
-The Cin and Cout bits to the most significant pair of bits being added are not the same.

An overflow cannot result if a positive and negative number are added.

Example: given 4 bits:

$$\begin{array}{r} 0111_2 \\ +\ 0001_2 \\ \hline 1000_2 \end{array}$$

= overflow **NOTE:** there is not a carry-out!

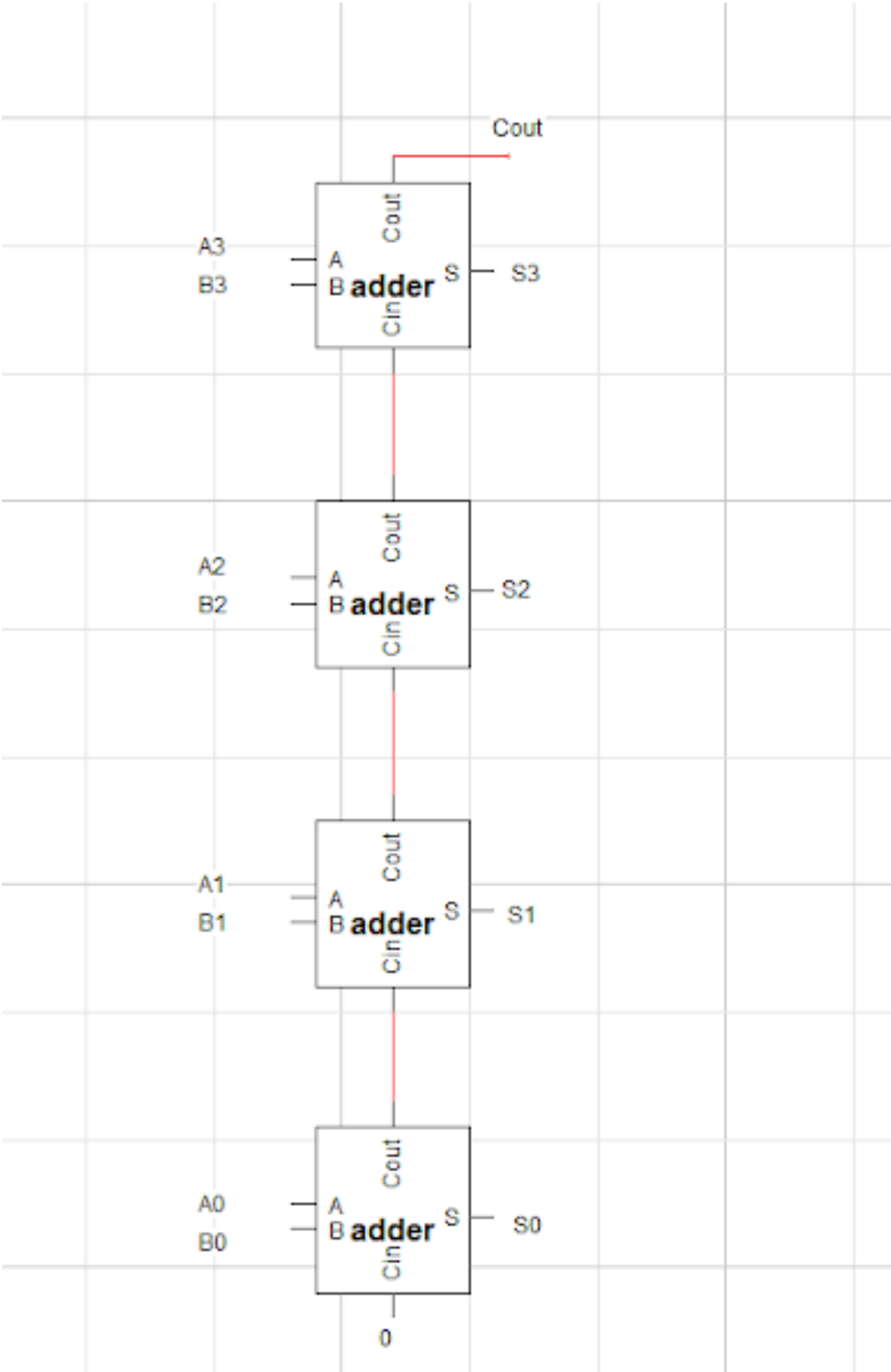
In two's complement representation, a carry-out does not indicate an overflow, as it does in unsigned representation.

Example: given 4 bits,

$$\begin{array}{r} 1001_2\ (-7_{10}) \\ +\ 1111_2\ (-1_{10}) \\ \hline 1\ 1000_2\ (-8_{10}) \end{array}$$

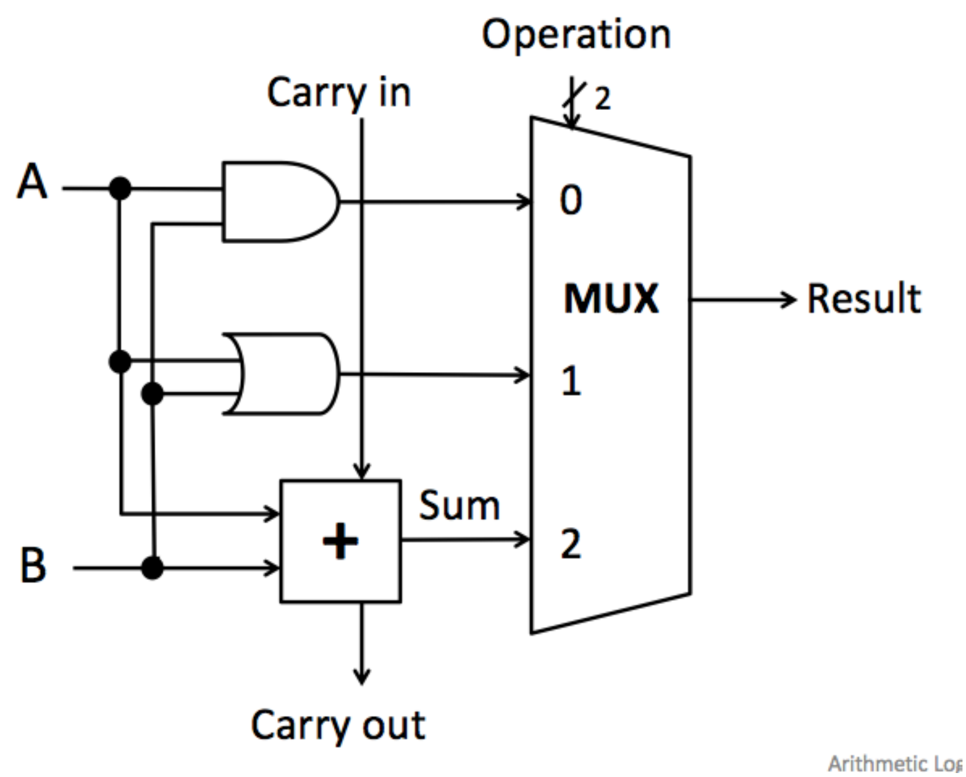
no overflow, even though there is a carry-out

4-bit Ripple-Carry Adder



Arithmetic Logic Unit
An **ALU** is a combinational circuit used to perform all the arithmetic and logical operations for a computer processor.

A simple 1-bit ALU can be built using the basic components you have learned about: **AND** gate, **OR** gate, **1-bit adder**, and **multiplexer**:



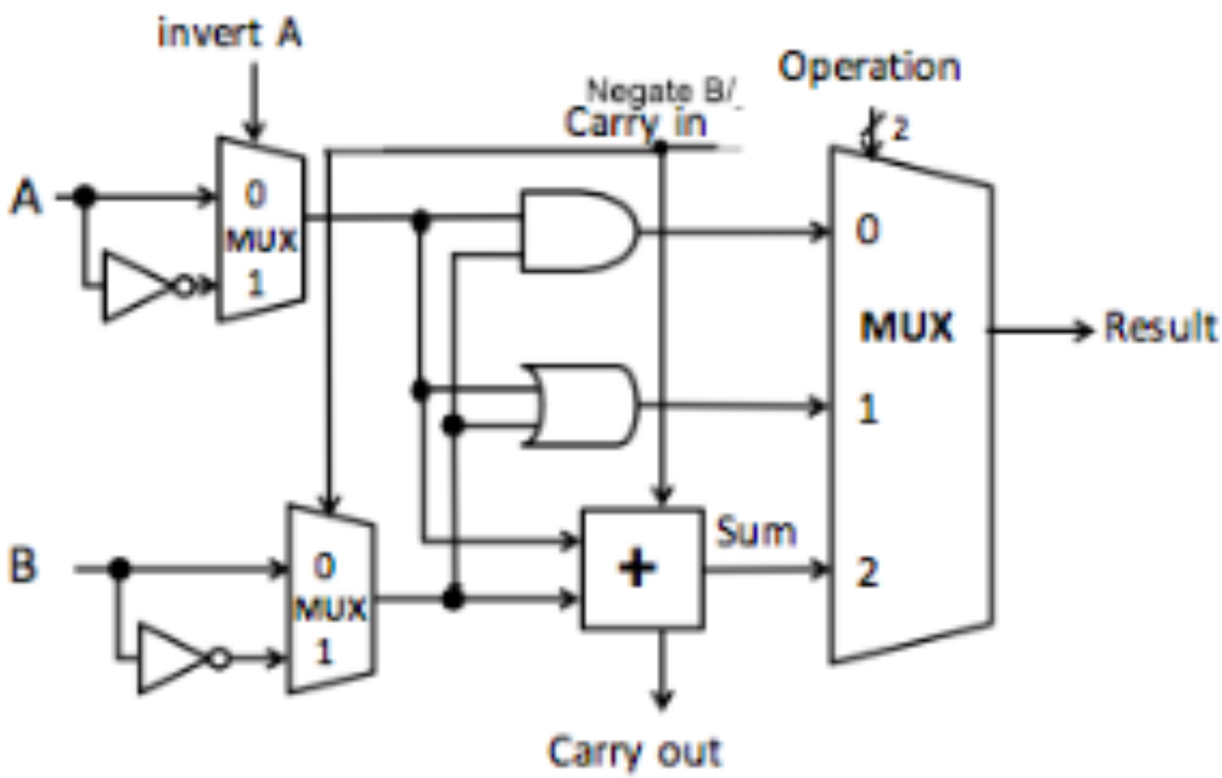
The **Operation (**a 2-bit value**)**, selects which operation should be produced as the **Result**:

Operation bits	Result
0 0	A AND B
0 1	A OR B
1 0	A + B

By adding some additional control inputs, it is possible to get produce additional functions with the ALU.

Invert A is used to complement the input A.

Negate B/Carry in used to complement input B for logical operations, and as a carry-in when addition is performed.

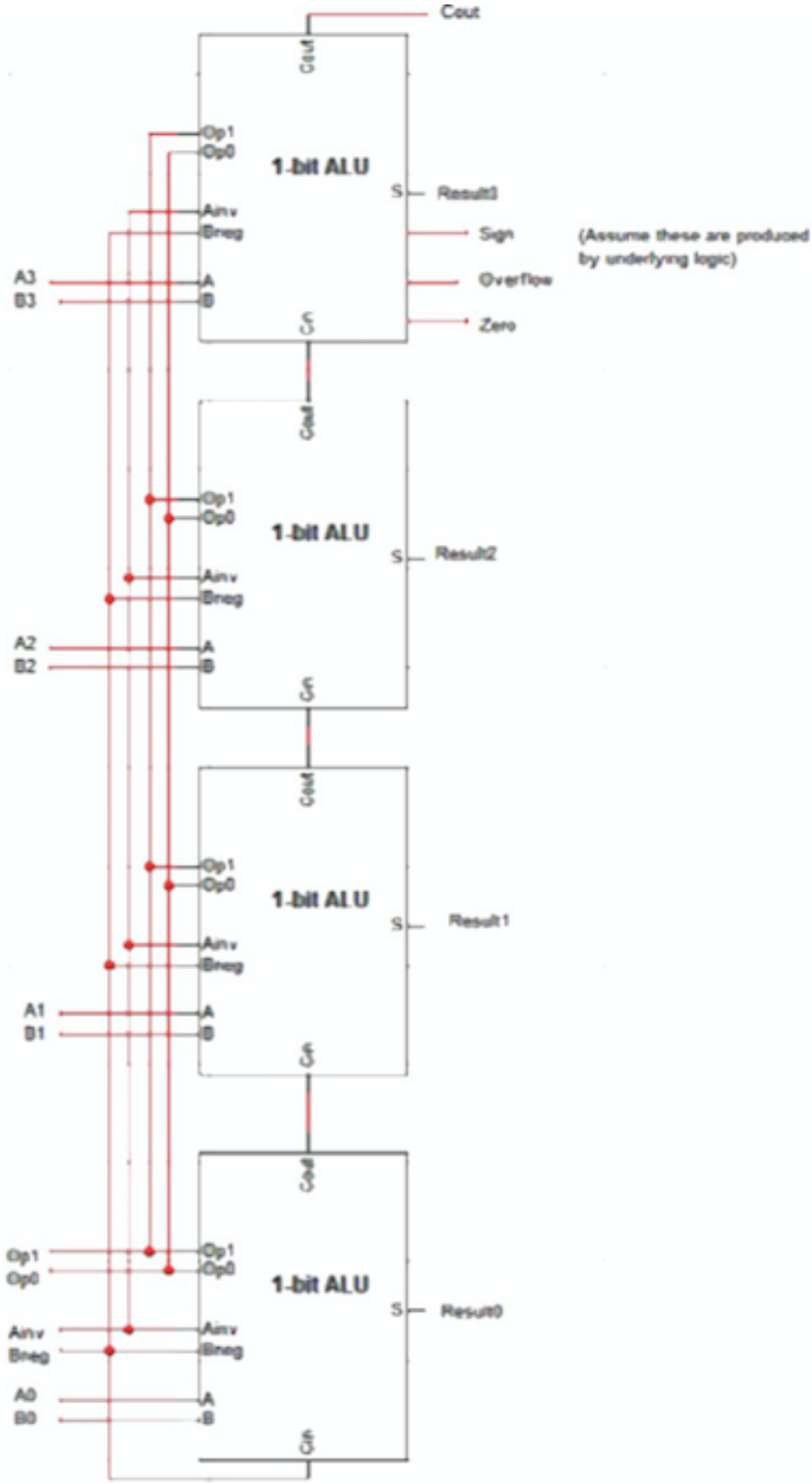


Basic Operations

- **add** (operation = add)
- **sub** (negate B/Carry in = 1, operation = ADD)
- **AND (**operation = AND**)**
- **OR (**operation = OR**)**
- **NOR (**invertA=1, negateB=1, operation = AND**)**

Control Inputs				Result
invertA	**negateB**	**Operation**		
0	0	0	0	a AND b
0	0	0	1	a OR b
0	0	1	0	a + b
0	1	1	0	a – b
1	1	0	0	a NOR b

A 4-bit ALU can be built from 4 1-bit ALUs in the same way that a 4-bit adder can be built from 1-bit adders:



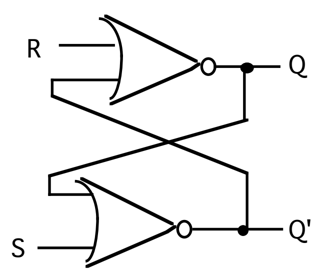
Cout
Cout
Op1
Op0
1-bit ALU
S
Result3
Ainv
Bneg
Sign
(Assume these are produced by underlying logic)
A3
B3
A
B
Overflow
Cin
Zero
Cout
Op1
Op0
1-bit ALU
S
Result2
Ainv
Bneg
A2
B2
A
B
Cin
Cout
Op1
Op0
1-bit ALU
S
Result1
Ainv
Bneg
A1
B1
A
B
Cin
Cout
Op1
Op0
Op1
Op0
1-bit ALU
S
Result0
Ainv
Bneg
Ainv
Bneg
A0
B0
A
B
Cin

How do you produce the Sign, Overflow, and Zero bits?

Basic Memory Circuit

Latch Single-bit memory, level-triggered

SR (Set Reset) Latch



S	R	Q	Q′	
0	0	Qp	Qp′	remember
0	1	0	1	reset/clear
1	0	1	0	set
1	1	unpredictable		

What does **unpredictable** mean? Notice in a NOR gate, if either input = 1 to a gate, its output = 0 (**1** is a deterministic input):

A	B	(A+B)′
0	0	1
0	1	0
1	0	0
1	1	0

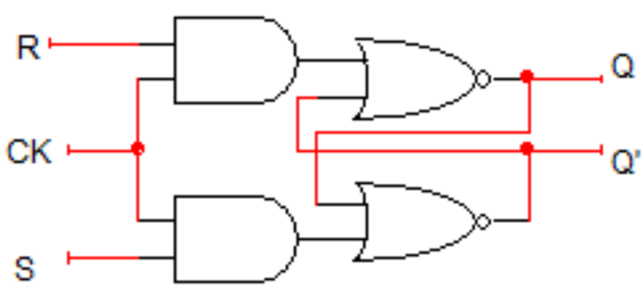
So, although you wouldn’t usually try to *set* and *reset* at the same time (it doesn’t make sense), if you did, Q and Q′ will both be 0 (which is not unpredictable).

However, when you go back to the *remember* state (S=R=0), Q and Q′ will not stay at 0 0. The circuit passes through one of either the *set* or *reset* state on its way back to the *remember* state, and Q and Q' change to the complement of one another.

Since the final state depends on which transitional state was sensed on the way back to *remember*, you cannot predict whether the final state of Q will be 1 or 0.

Clocked SR Latch

Incorporates a clock input.



Output Q can change in response to S and R whenever the CK input is asserted.