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

- **Signed Representation**
- **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.

Example: given 4 bits:
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

When adding two numbers with the same sign which each can be represented with n bits, the result may cause an overflow (not fit in n bits).

An overflow occurs when adding if:

-Two positive numbers added together yield a negative result, or
-Two negative numbers added together yield a positive result, or
-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:

$$\begin{array}{r} 0111_2 \\ \underline{+\ 0001_2} \\ 1000_2 \end{array} \quad = \text{overflow}$$

NOTE: there is not a carry-out!

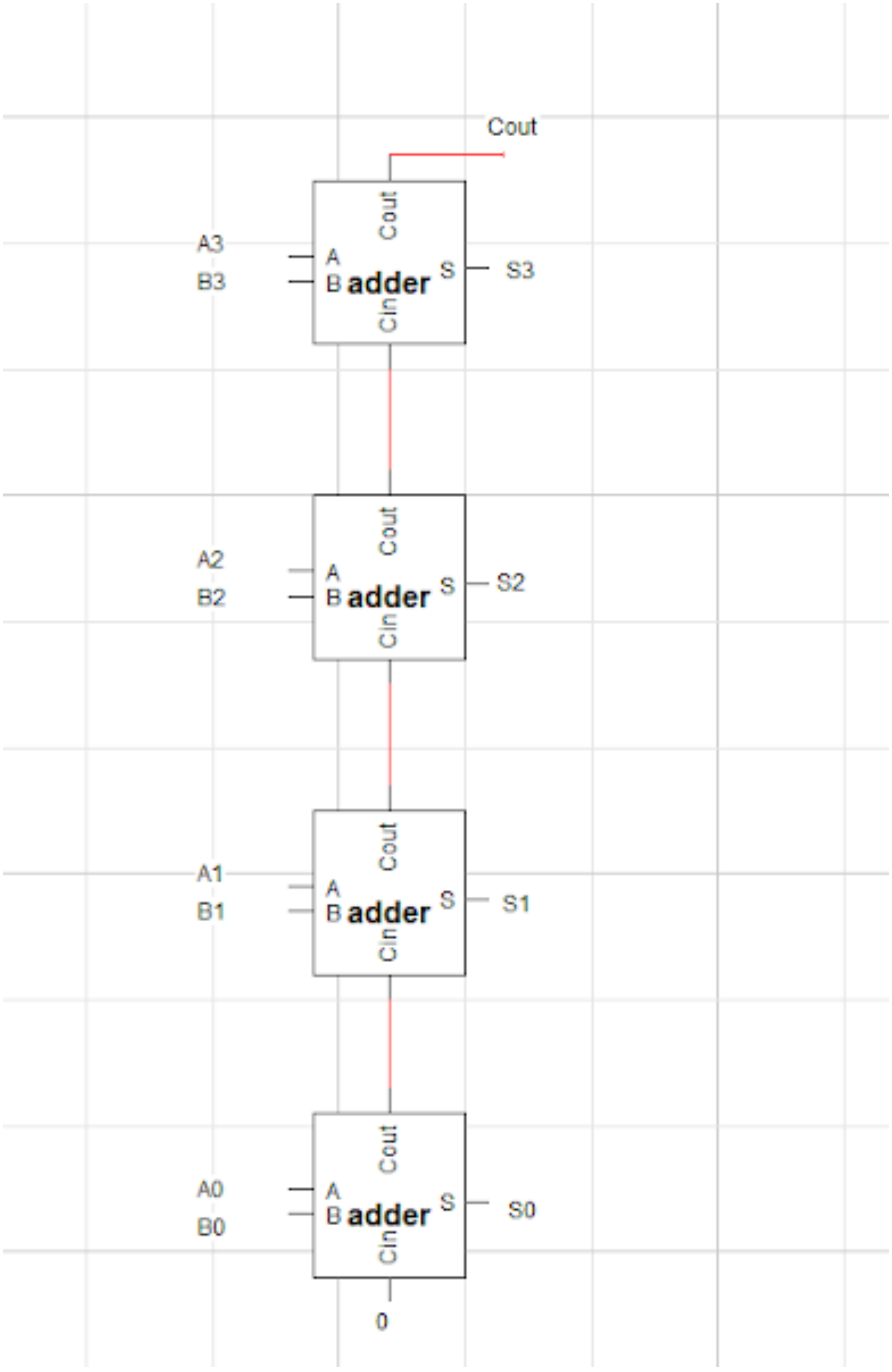
In two's complement representation, a carry-out does not indicate an overflow, as it does in unsigned representation.

Example: given 4 bits,

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

no overflow, even though there is a carry-out

4-bit Ripple-Carry Adder



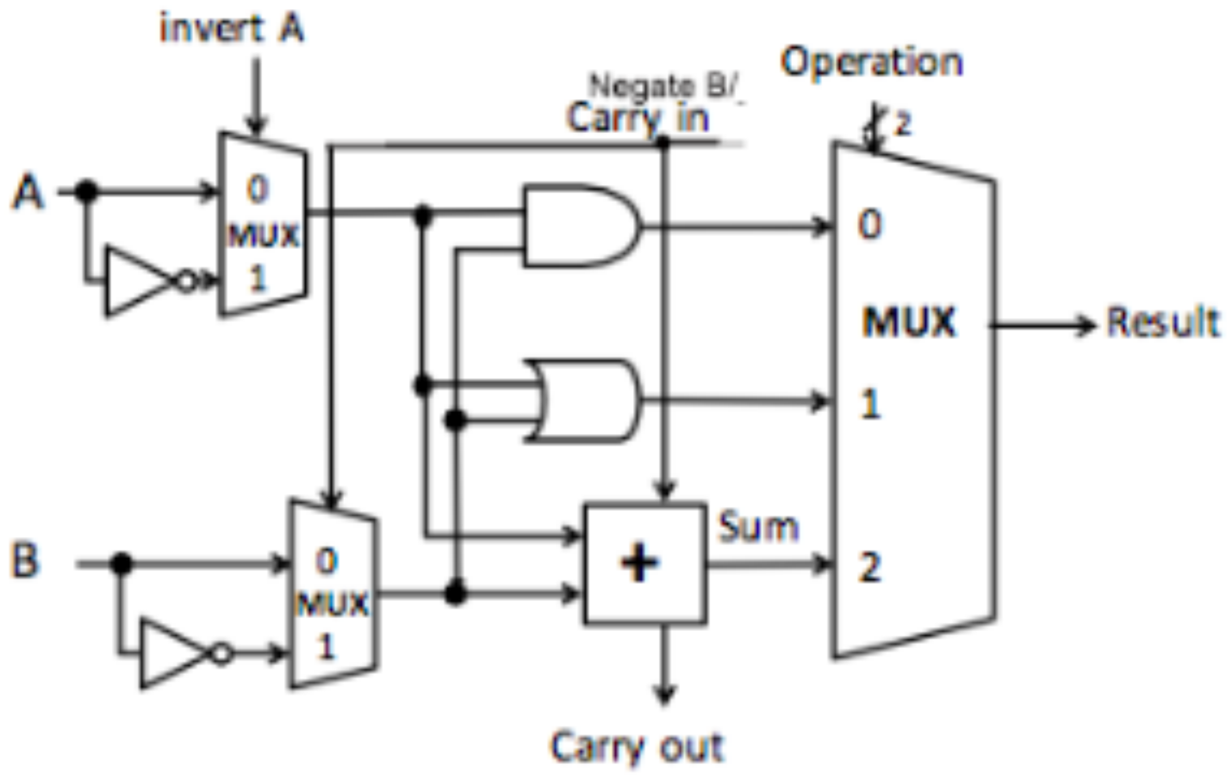
Arithmetic Logic Unit
Combinational circuit used to perform all the arithmetic and logical operations for a computer processor

Can be built using basic components

Invert A 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.

Operation (2 bits) selects which logical or arithmetic operation to select as output from the ALU

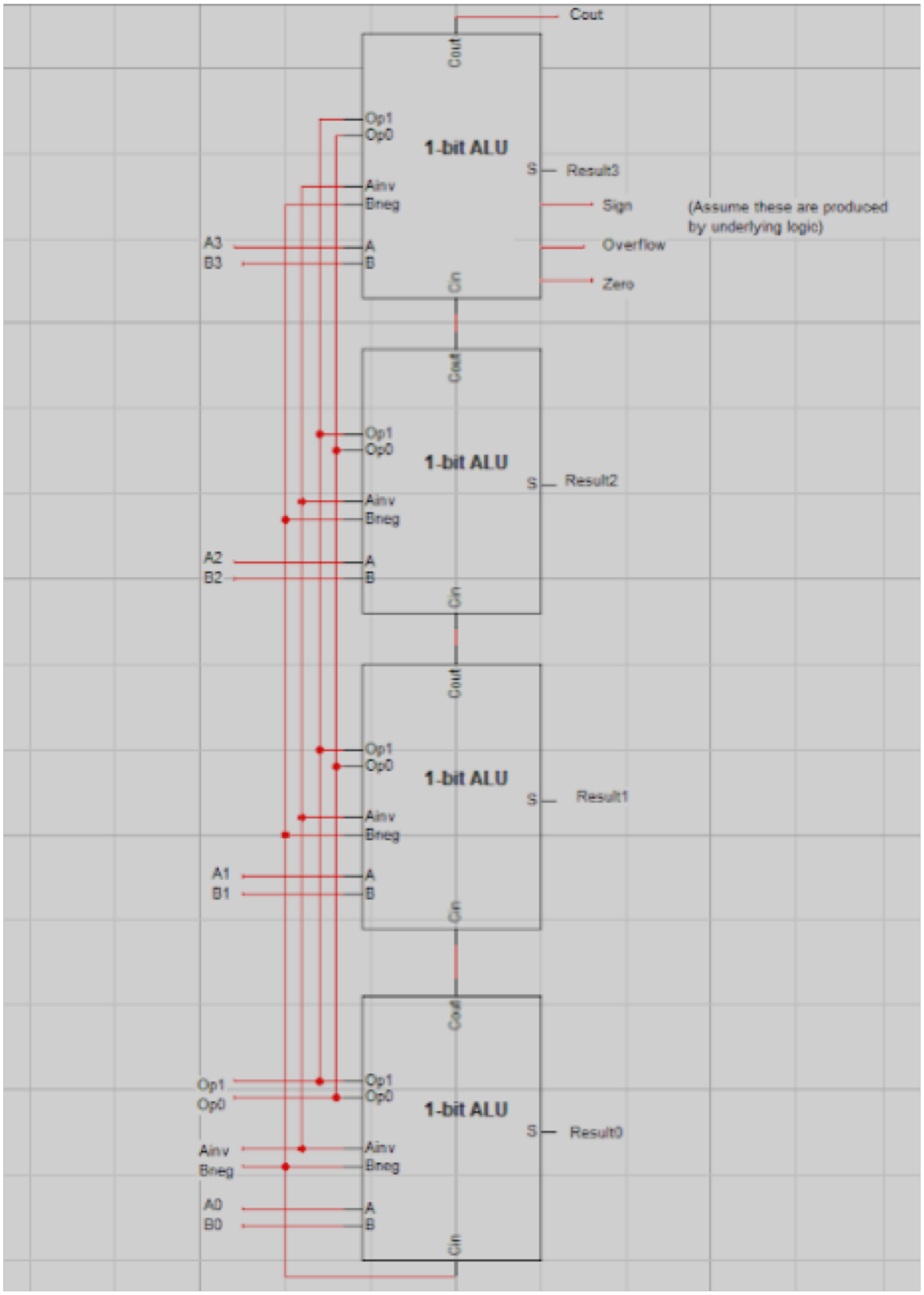


Basic Operations

- **add** (A + B + Carry in)
- **sub** (negate B/Carry in = 1, A+B)
- **AND (**A AND B**)**
- **OR (**A OR B**)**
- **NOR (**invert A=1, negate B=1, A AND B**)**

invert A	negate B/Carry in	Op1	Op0	Result
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:

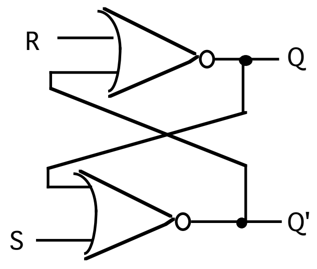


How do you produce the Sign, Overflow, and Zero bits?

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		reset (clear)
1	0	1		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

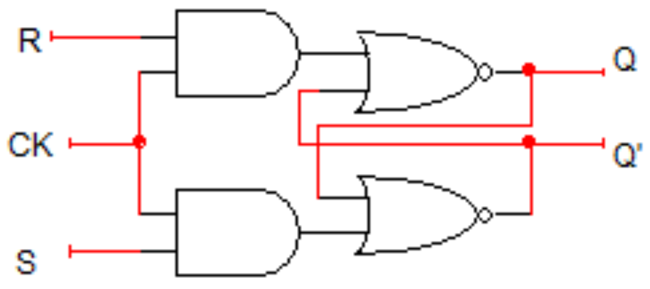
So, although you wouldn’t usually try to *set* and *reset* at the same time (it doesn’t make sense), if you did, Q and Q′ will both be 0 (which is not unpredictable).

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.