CS 240 Lab 3
Basic Digital Circuits

- **Review of Two's Complement and Overflow**
- **Multiplexer**
- **Decoder**
- **Adder**
- **Arithmetic Logic Unit (ALU)**

Two's Complement and Overflow

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

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

Signed representation: $-2^{n-1} \rightarrow 2^{n-1} - 1$ because one bit 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 in Addition

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 carry-in and carry-out 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:

```
   0111
 + 0001
 ------
   1000     overflow        NOTE: there is not a carry-out!
```

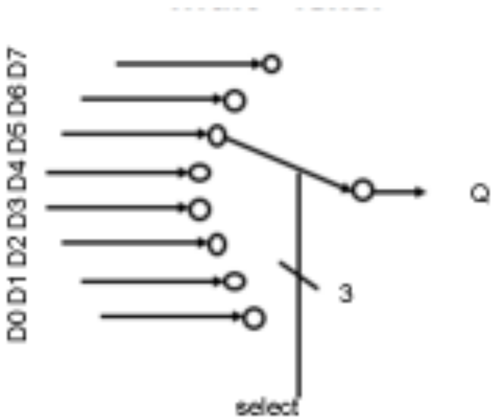
In two's complement representation, a carry-out does not indicate an overflow, as it does in unsigned representation.

Example: given 4 bits,

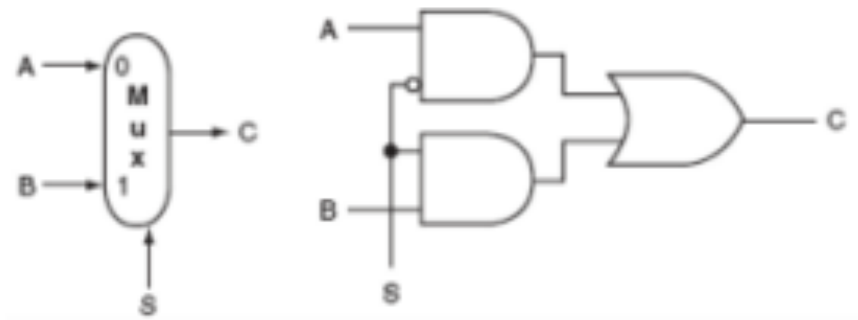
```
   1001 (-7)
 + 1111 (-1)
 -----------
 1 1000 (-8)    no overflow, even though there is a carry-out
```

Multiplexer

Uses n select lines to choose one of the possible 2^n inputs to pass through to the output. Usually used for **selection**, but can also act as code detectors.



8x1 MUX



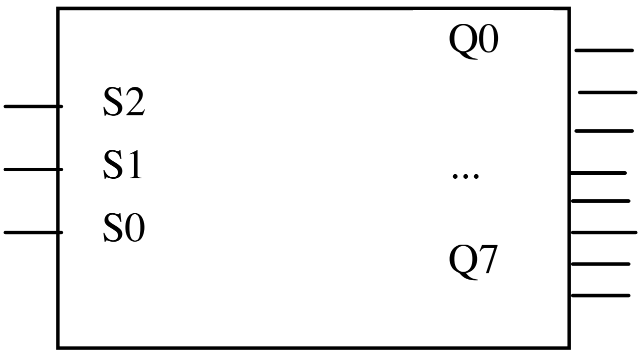
2x1 MUX

Q = S2'S1'S0'D0 +
S2'S1'S0D1 +
S2'S1S0'D2 +
S2'S1S0D3 +
S2S1'S0'D4 +
S2S1'S0D5 +
S2S1S0'D6 +
S2S1S0D7

C = S'A + SB

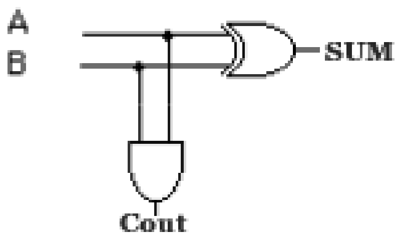
S2 S1 S0	Q
0 0 0	D0
0 0 1	D1
0 1 0	D2
0 1 1	D3
1 0 0	D4
1 0 1	D5
1 1 0	D6
1 1 1	D7

Decoder
Takes an n-digit binary number input and identifies one of 2^n output data lines to activate.



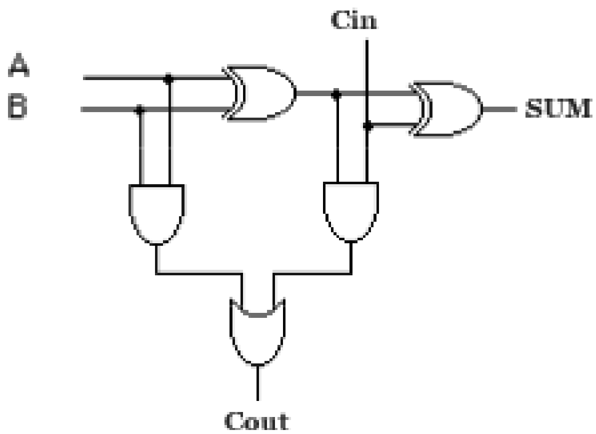
S2	S1	S0	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Half-Adder — adds two one-bit values



A	B	Cout	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Full Adder — incorporates a carry-in



Cin	A	B	Cout	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

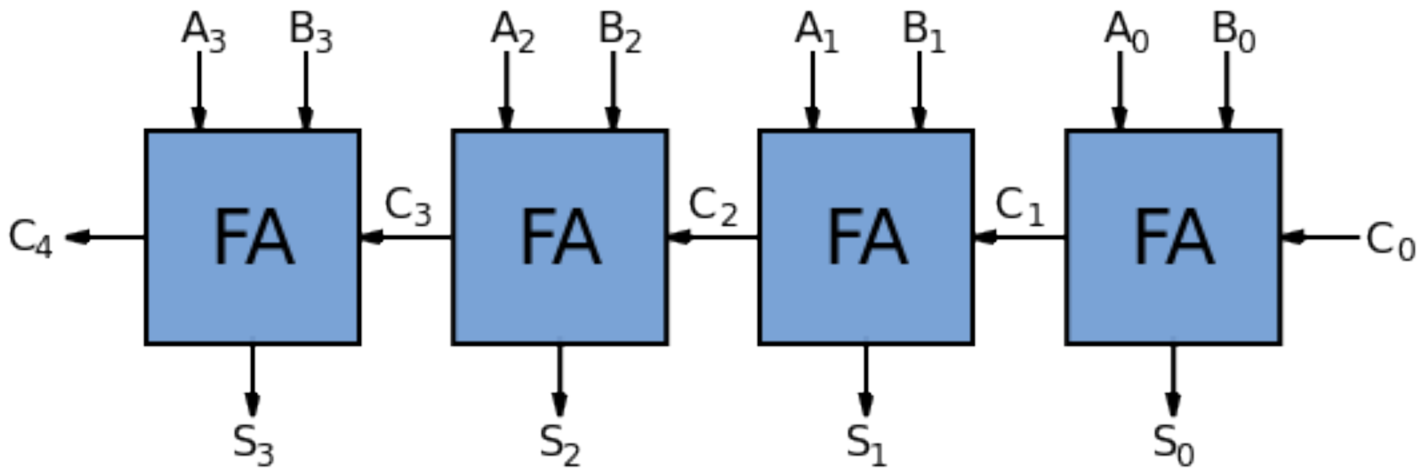
$Sum = A \oplus B \oplus Cin$

$Cout = AB + (A \oplus B)Cin$

n-bit adder = n 1-bit adders

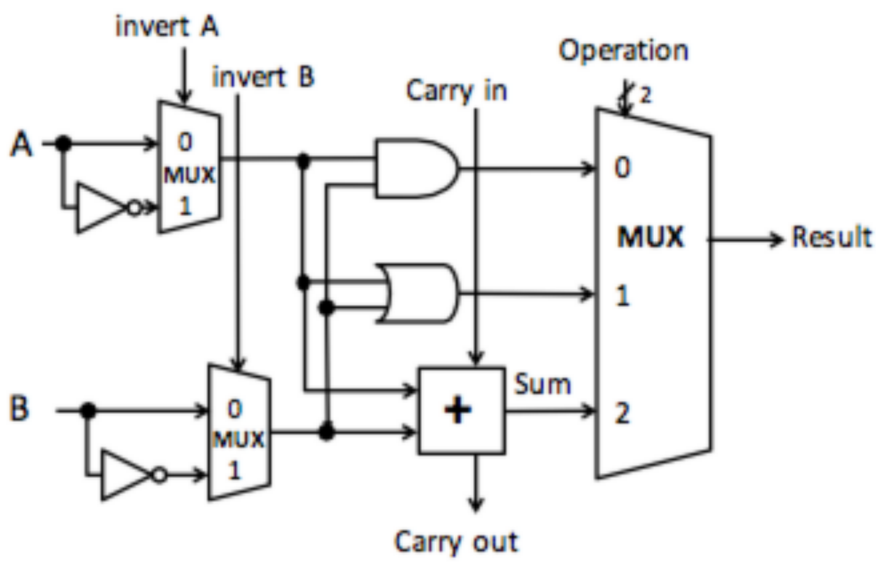
Carry-out of each adder = Carry-in for next two most significant bits being added

Carry-in to least significant adder is normally = 0



ALU

Want to be able to select whether the ALU will produce the bitwise AND, OR, and sum as a result.



The basic operations and results are:

add (a + b + Cin),

AND (a AND b),

OR (a OR b),

Adding the ability to choose whether to invert A or B provides additional operations:

sub (invert b, Cin = 1, a + b + Cin)

NOR (invert a, invert b, a AND b)

invA	invB	Cin	Op1	Op0	Result
0	0	X	0	0	a AND b
0	0	X	0	1	a OR b
0	0	0/1	1	0	a + b
0	1	1	1	0	a – b
1	1	X	0	0	a NOR b