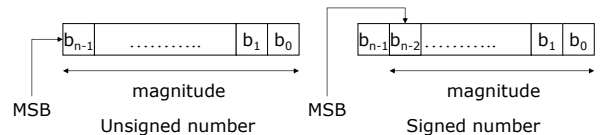


## ECE380 Digital Logic

### Number Representation and Arithmetic Circuits: Signed Numbers, Binary Adders and Subtractors

## Signed numbers

- For signed numbers, in the binary system, the sign of the number is denoted by the left-most bit
  - 0 = positive
  - 1 = negative
- For an  $n$ -bit number, the remaining  $n-1$  bits represent the magnitude



## Negative numbers

- For signed numbers, there are three common formats for representing negative numbers
  - Sign-magnitude
  - 1's complement
  - 2's complement
- Sign-magnitude uses one bit for the sign (0=+, 1=-) and the remaining bits represent the magnitude of the number as in the case of unsigned numbers
- For example, using 4-bit numbers
  - +5=0101 -5=1101
  - +3=0011 -3=1011
  - +7=0111 -7=1111
- Although this is easy to understand, it is not well suited for use in computers

## 1's complement representation

- In the 1's complement scheme, an  $n$ -bit negative number  $K$ , is obtained by subtracting its equivalent positive number,  $P$ , from  $2^n-1$   
 $K=(2^n-1)-P$
- For example, if  $n=4$ , then
  - $K=(2^4-1)-P=(15)_{10}-P=(1111)_2-P$
  - $-5=(15)_{10}-5=(1111)_2-(0101)_2=(1010)_2$
  - $-3=(15)_{10}-3=(1111)_2-(0011)_2=(1100)_2$
- From these examples, clearly the 1's complement can be formed simply by complementing each bit of the number, including the sign bit
- Numbers in the 1's complement form have some drawbacks when used in arithmetic operations

## 2's complement representation

- In the 2's complement scheme, an  $n$ -bit negative number  $K$ , is obtained by subtracting its equivalent positive number,  $P$ , from  $2^n$   
 $K=2^n-P$
- For example, if  $n=4$ , then  
 $K=2^4-P=(16)_{10}-P=(10000)_2-P$   
 $-5=(16)_{10}-5=(10000)_2-(0101)_2=(1011)_2$   
 $-3=(16)_{10}-3=(10000)_2-(0011)_2=(1101)_2$
- A simple way of finding the 2's complement of a number is to add 1 to its 1's complement

## Rule for finding 2's complements

- Given a signed number,  $B=b_{n-1}b_{n-2}\dots b_1b_0$ , its 2's complement,  $K=k_{n-1}k_{n-2}\dots k_1k_0$ , can be found by:
  - examining all the bits of  $B$  from right to left and complementing all the bits after the first '1' is encountered
- For example if  
 $B=00110100$
- Then the 2's complement of  $B$  is  
 $K=11001100$   
 changed bits                      unchanged bits

## Four-bit signed integers

$b_3b_2b_1b_0$	Sign-magnitude	1's complement	2's complement
0111	+7	+7	+7
0110	+6	+6	+6
0101	+5	+5	+5
0100	+4	+4	+4
0011	+3	+3	+3
0010	+2	+2	+2
0001	+1	+1	+1
0000	+0	+0	+0
1000	-0	-7	-8
1001	-1	-6	-7
1010	-2	-5	-6
1011	-3	-4	-5
1100	-4	-3	-4
1101	-5	-2	-3
1110	-6	-1	-2
1111	-7	-0	-1

## Addition and subtraction

- For sign-magnitude numbers, addition is simple, but if the numbers have different signs the task becomes more complicated
  - Logic circuits that compare and subtract numbers are also needed
  - It is possible to perform subtraction without this circuitry
  - For this reason, sign-magnitude is not used in computers
- For 1's complement numbers, adding or subtracting some numbers may require a correction to obtain the actual binary result
- For example,  $(-5)+(-2)=(-7)$ , but when adding the binary equivalents of  $-5$  and  $-2$ , the result is 0111 with an additional carry out of 1 which must be added back to the result to produce the final (correct) result of 1000

## 2's complement operations

- For addition, the result is always correct
- Any carry-out from the sign-bit position is simply ignored

$$\begin{array}{r}
 (+5) \quad 0101 \\
 +(+2) \quad +0010 \\
 \hline
 (+7) \quad 0111
 \end{array}
 \qquad
 \begin{array}{r}
 (-5) \quad 1011 \\
 +(+2) \quad +0010 \\
 \hline
 (-3) \quad 1101
 \end{array}$$
  

$$\begin{array}{r}
 (+5) \quad 0101 \\
 +(-2) \quad +1110 \\
 \hline
 (+3) \quad 10011
 \end{array}
 \qquad
 \begin{array}{r}
 (-5) \quad 1011 \\
 +(-2) \quad +1110 \\
 \hline
 (-7) \quad 11001
 \end{array}$$

↑ ignore
↑ ignore

## 2's complement subtraction

- The easiest way of performing subtraction is to negate the subtrahend and add it to the minuend
  - Find the 2's complement of the subtrahend and then perform addition

$$\begin{array}{r}
 (+5) \quad 0101 \\
 -(+2) \quad -0010 \\
 \hline
 (+3) \quad 0111
 \end{array}
 \Rightarrow
 \begin{array}{r}
 0101 \\
 +1110 \\
 \hline
 10011
 \end{array}$$

↑ ignore

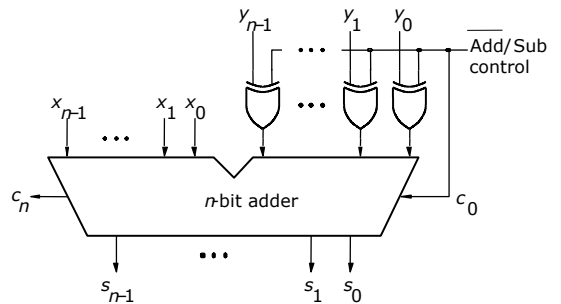
  

$$\begin{array}{r}
 (-5) \quad 1011 \\
 -(-2) \quad -1110 \\
 \hline
 (-3) \quad 1101
 \end{array}
 \Rightarrow
 \begin{array}{r}
 1011 \\
 +0010 \\
 \hline
 1101
 \end{array}$$

## Adder and subtractor unit

- The subtraction operation can be realized as the addition operation, using a 2's complement of the subtrahend, regardless of the signs of the two operands
  - It is possible to use the same adder circuit to perform both addition and subtraction
- Recall that the 2's complement can be formed from the 1's complement simply by adding 1
- We can use the XOR operation to perform a 1's complement
  - Recall  $x \oplus 1 = x'$  and  $x \oplus 0 = x$
  - If we are performing a subtract operation, 1's complement the subtrahend by XORing each bit with 1

## Adder and subtractor unit



## Arithmetic overflow

- The result of addition or subtraction is supposed to fit within the significant bits used to represent the numbers
- If  $n$  bits are used to represent signed numbers, then the result must be in the range  $-2^{n-1}$  to  $+2^{n-1}-1$
- If the result does not fit in this range, we say that **arithmetic overflow** has occurred
- To insure correct operation of an arithmetic circuit, it is important to be able to detect the occurrence of overflow

## Examples for determining arithmetic overflow

For 4-bit numbers, there are 3 significant bits and the sign bit

$$\begin{array}{r} (+7) \quad 0 \ 1 \ 1 \ 1 \\ + (+2) \quad + 0 \ 0 \ 1 \ 0 \\ \hline (+9) \quad 1 \ 0 \ 0 \ 1 \\ c_4 = 0 \\ c_3 = 1 \end{array} \qquad \begin{array}{r} (-7) \quad 1 \ 0 \ 0 \ 1 \\ + (+2) \quad + 0 \ 0 \ 1 \ 0 \\ \hline (-5) \quad 1 \ 0 \ 1 \ 1 \\ c_4 = 0 \\ c_3 = 0 \end{array}$$

$$\begin{array}{r} (+7) \quad 0 \ 1 \ 1 \ 1 \\ + (-2) \quad + 1 \ 1 \ 1 \ 0 \\ \hline (+5) \quad 1 \ 0 \ 1 \ 0 \ 1 \\ c_4 = 1 \\ c_3 = 1 \end{array} \qquad \begin{array}{r} (-7) \quad 1 \ 0 \ 0 \ 1 \\ + (-2) \quad + 1 \ 1 \ 1 \ 0 \\ \hline (-9) \quad 1 \ 0 \ 1 \ 1 \ 1 \\ c_4 = 1 \\ c_3 = 0 \end{array}$$

If the numbers have different signs, no overflow can occur

## Arithmetic overflow

- In the previous examples, overflow was detected by

$$\begin{aligned} \text{overflow} &= c_3 c_4' + c_3' c_4 \\ \text{overflow} &= c_3 \oplus c_4 \end{aligned}$$

- For  $n$ -bit numbers we have

$$\text{overflow} = c_{n-1} \oplus c_n$$

- The adder/subtractor circuit introduced can be modified to include overflow checking with the addition of one XOR gate