

---

---

# Integer Representation Error Detection

---

---

# Error Detection

- Errors are possible as the number representation uses a fixed length
- Certain operations will cause the result to exceed
  - the maximum positive value
  - the maximum negative value
- Why do we care?
  - The result of such an operation cannot be expressed in the fixed length

# When Can Error Occur

- Unsigned Numbers
  - Adding two positive numbers
  - Subtracting a larger number from a smaller number
- Complement Numbers
  - Adding two numbers of the same sign (either +ve or -ve)
  - Subtracting a negative number from a positive number
  - Subtracting a positive number from a negative number
- Only the one of the above (orange text) is a guaranteed error → why?

# Error Detection

- There is no way to avoid such errors
  - Why does widening the data size not work?
- There are way(s) to detect when each of the errors occurs
- What the error is and how to detect it are **different**

# Carry Error

- When the result of a math operation exceeds the allowable range, when the values are interpreted as unsigned values
  - Only applies to unsigned values
  - Does not specify a particular math operation
  - Detection is different for different operations
- Textbook **incorrectly** refers to this as an unsigned overflow

# Carry Error - Detection

- During addition, a carry error occurs if and only if a carry out of the MSBit occurs
- During subtraction a carry error occurs if and only if a borrow into the MSBit is required

# Carry Error - Examples

Assuming 4-bit unsigned numbers are being used

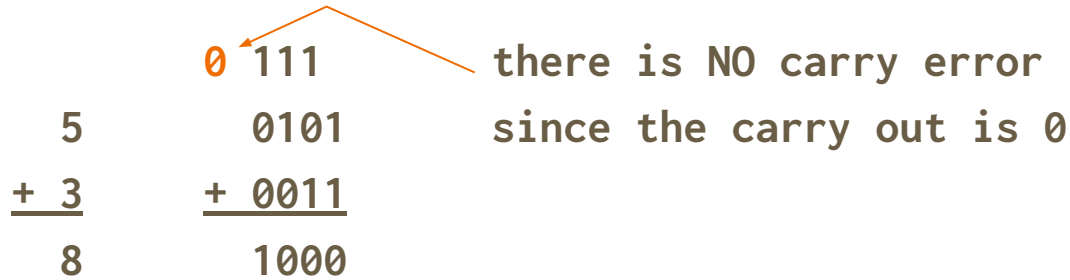


Diagram illustrating a correct 4-bit addition without a carry error. The decimal numbers 5 and 3 are added to get 8. The 4-bit binary representations are shown: 5 is 0101, 3 is 0011, and the result 8 is 1000. An orange arrow points from the text "there is NO carry error since the carry out is 0" to the carry-out bit, which is a 0.

$$\begin{array}{r} 5 \\ + 3 \\ \hline 8 \end{array}$$
$$\begin{array}{r} 0101 \\ + 0011 \\ \hline 1000 \end{array}$$

there is NO carry error  
since the carry out is 0

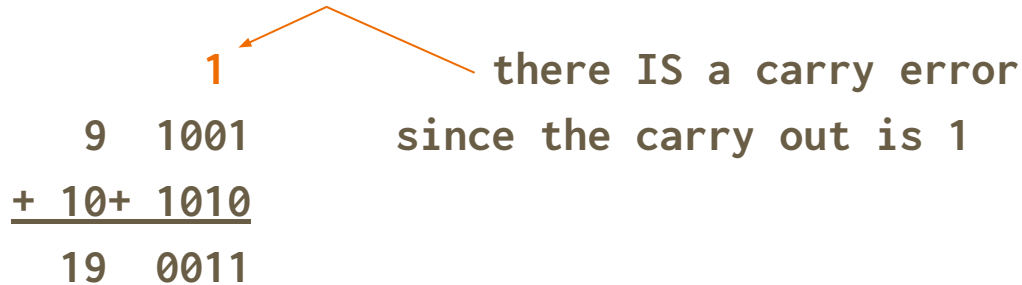


Diagram illustrating a carry error in 4-bit unsigned addition. The decimal numbers 9 and 10 are added to get 19. The 4-bit binary representations are shown: 9 is 1001, 10 is 1010, and the result 19 is 0011. An orange arrow points from the text "there IS a carry error since the carry out is 1" to the carry-out bit, which is a 1.

$$\begin{array}{r} 9 \\ + 10 \\ \hline 19 \end{array}$$
$$\begin{array}{r} 1001 \\ + 1010 \\ \hline 0011 \end{array}$$

there IS a carry error  
since the carry out is 1

# Carry Error - Examples

Assuming 4-bit unsigned numbers are being used

	112	
	+ 2202	there IS a carry error
2	0010	since a borrow <b>T0</b> the
- 3	- 0011	MSB was required
<u>15</u>	<u>1111</u>	



# Overflow Error

- When the result of a math operation exceeds the allowable range, when the values are interpreted as signed values
  - Only applies to signed values
  - Does not specify a particular math operation

# Overflow Error

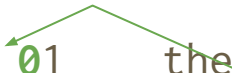

- When the result of a math operation exceeds the allowable range, when the values are interpreted as signed values
  - Only applies to signed values
  - Does not specify a particular math operation

# Overflow Error - Detection

- During addition, an overflow error occurs if and only if carry in to the MSB  $\neq$  carry out from the MSB
  - it is possible to check the signs of the three items but this is far more convoluted and difficult
  - so is not an acceptable method.
- Why is only addition specified and not subtraction?

# Overflow Error - Examples

Assume that 4-bit 2's comp signed numbers are being used

		there is no overflow error
2	0010	since carry in = carry out
<u>+ 3</u>	<u>+ 0011</u>	
5	0101	
		there is an overflow error
5	0101	since carry in (1) != carry out (0)
<u>+ 3</u>	<u>+ 0011</u>	
-8	1000	

# Error Detection - Final Note

## IMPORTANT!

- For a particular operation it is possible to have both errors, one of the errors or neither error
  - 2's complement selected since it has the same addition circuit as unsigned
- ∴ Only the error detection circuitry is different
- i.e. the CPU does not care if the number being added are signed or unsigned, the add operation uses the add circuit and **both** error detection circuits, and the user determines which error to look for