Binary Addition/Subtraction w/Overflow

Last updated 6/4/26

These slides introduce addition with overflow

Binary Addition/Subtraction w/ Overflow

- Overflow
  - In elementary school we did not care how many digits the answer required

$$\begin{array}{r} 745 \\ +\ 589 \\ \hline 1334 \end{array}$$

  - In binary addition – we are generally representing something that ultimately is to be executed in hardware
    - Our hardware cannot change the number of bits (wires) it can hold
    - We must establish a maximum number size (# of bits) and create an error when the result of the addition does not fit in this size
    - The error is called an overflow

© tj

Binary Addition/Subtraction w/ Overflow

- Unsigned, Signed, and Special Binary numbers have a fixed size
  - Unsigned and Signed set by the system
  - Special set by their nature
- The fixed size creates “odd” behaviors in some situations
  - Wrapping
  - Sudden sign changes
  - Erroneous calculations

© tj

Binary Addition/Subtraction w/ Overflow

- Overflow - Unsigned
  - Overflow is defined as:
    - Result does not fit into the allowed # of bits

n bit unsigned addition

3 bit addition

```
  101
+ 011
 1000
```

overflow

5 bit addition

```
 00101
+ 00011
 01000
```

OK

6 bit addition

```
  010101
+ 101011
 1000000
```

overflow

8 bit addition

```
  10011101
+ 00001001
  10100110
```

OK

Our programs will ignore the overflow and just give us the bits that fit

000 01000 000000 10100110

© tj

Binary Addition/Subtraction w/ Overflow

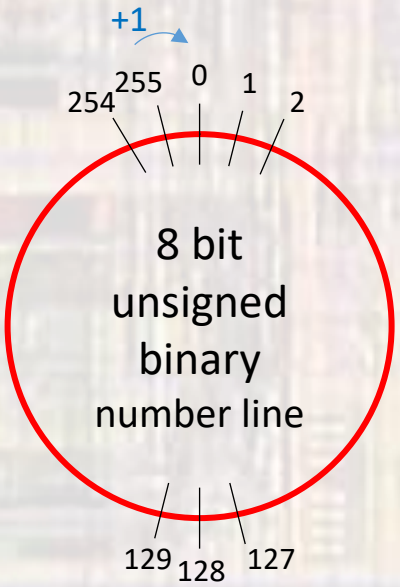
- Overflow – Unsigned – called wrapping
  - We only keep the part of the result that fits in our original number of bits
  - Consider an 8 bit unsigned number (or a uint8_t)
    - Range: 0 to 255
    - What happens if I add: 255 + 1
      - Result is 0

```
  1111 1111
+ 0000 0001
 10000 0000
```

⇨

```
  1111 1111
+ 0000 0001
  0000 0000
```

0



- "wraps" from 255 ↔ 0

© tj

Binary Addition/Subtraction w/ Overflow

- Wrapping – unsigned (overflow/underflow)
  - We only keep the part of the result that fits in our original number of bits
  - Consider an 8 bit unsigned number (or a uint8_t)
    - Range: 0 to 255
    - What happens if I subtract: 4 – 6

      → 4 + (-6)

      - Result is 254

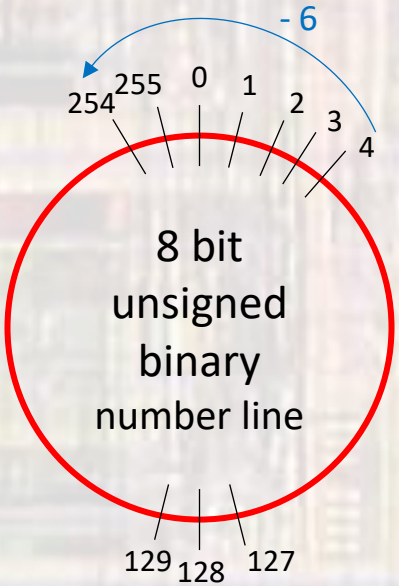
      - "wraps" from 0 ↔ 255

```
  0000 0100
--0000 0110
```

⇨

```
  0000 0100
+ 1111 1010
  1111 1110
```

254



© tj

Binary Addition/Subtraction w/ Overflow

- Overflow – signed(2's complement)
  - Overflow is defined as:
    - carry-in of the msb ≠ carry-out of the msb → overflow
    - Our result may exceed the allowed # of bits and still be OK
      - Extra bits are ignored

8 bit signed addition

	0		1	1	1	1	1		
	0	1	0	0	0	1	1	1	71
+	0	0	0	1	1	1	0	1	+ 29
	0	1	1	0	0	1	0	0	100

	1	1			1	1	1		
	1	1	0	0	0	1	1	1	-57
+	1	1	1	1	0	1	0	1	+ -11
	1	1	0	1	1	1	1	0	0	-68

	0	1	1	1	1	1	1		
	0	1	1	0	0	1	1	1	103
+	0	0	1	1	1	1	0	1	+ 61
	1	0	1	0	0	1	0	0	164

overflow -28

	1				1	1	1		
	1	1	0	0	0	1	1	1	-57
+	1	0	0	0	0	1	0	1	+ -123
	1	0	1	0	0	0	1	0	0	-180

overflow 68

Our programs will ignore the overflow and just give us the bits that fit

© tj

Binary Addition/Subtraction w/ Overflow

- Wrapping and Rapid sign changes – signed
  - We only keep the part of the result that fits in our original number of bits
  - Consider an 8 bit signed number (or an int8_t)
    - Range: -128 to 127
    - What happens if I add: 127 + 1

```
  0111 1111
  0000 0001
+ 1000 0000
-----------
  -128
```

    - Result is -128
    - What happens if I subtract: -127 – 3

-127 + (-3)

    - Result is 126

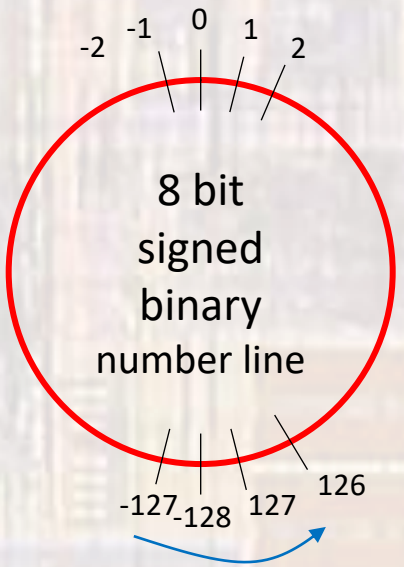
```
  1000 0001
--  0000 0011
```

⇨

```
  1000 0001
+ 1111 1101
-----------
  0111 1110
```

126

    - "wraps" from 127 ↔ -128



© tj

Binary Addition/Subtraction w/ Overflow

- Binary Subtraction
  - Warning
    - Some unsigned numbers cannot be represented as signed in a fixed number of bits

    - 8b unsigned

12 – 7	→	12 + (-7)	5 - OK
12 – 129	→	12 + (-129)	-129 cannot be represented in 8 bits
	→	139	error

```
uint8_t a;
uint8_t b;
uint8_t c;
a = 12;
b = 7;

c = a - b;
printf("%i\n", c);

a = 12;
b = 129;
c = a - b;
printf("%i\n", c);
```

→

```
<terminated> (e
5
139
```


© tj