

## Review Questions

- ❖ What is the value (and encoding) of **TMin** for a fictional 6-bit wide integer data type?
- ❖ For unsigned char `uc = 0xA1;`, what are the produced data for the cast **(unsigned short)uc**?
- ❖ What is the result of the following expressions?
  - **(signed char)uc >> 2**
  - **(unsigned char)uc >> 3**

4

## Why Does Two's Complement Work?

- ❖ For all representable positive integers  $x$ , we want:

$$\begin{array}{r} \text{bit representation of } x \\ + \text{bit representation of } -x \\ \hline 0 \end{array} \quad (\text{ignoring the carry-out bit})$$

- What are the 8-bit negative encodings for the following?

$$\begin{array}{r} 00000001 \\ + \text{????????} \\ \hline 00000000 \end{array} \quad \begin{array}{r} 00000010 \\ + \text{????????} \\ \hline 00000000 \end{array} \quad \begin{array}{r} 11000011 \\ + \text{????????} \\ \hline 00000000 \end{array}$$

5

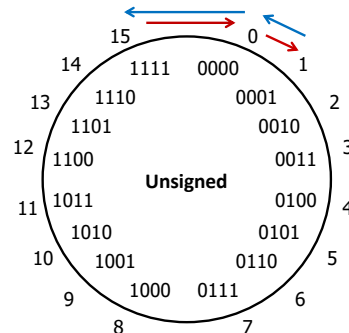
## Overflow: Unsigned

- ❖ **Addition:** drop carry bit ( $-2^N$ )

$$\begin{array}{r} 15 \\ + 2 \\ \hline \text{17} \\ \text{1} \end{array} \quad \begin{array}{r} 1111 \\ + 0010 \\ \hline 10001 \end{array}$$

- ❖ **Subtraction:** borrow ( $+2^N$ )

$$\begin{array}{r} 1 \\ - 2 \\ \hline \text{-1} \\ \text{15} \end{array} \quad \begin{array}{r} 10001 \\ - 0010 \\ \hline 1111 \end{array}$$



$\pm 2^N$  because of modular arithmetic

16

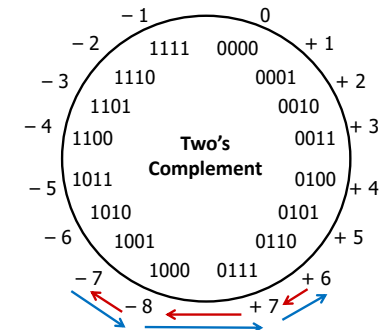
## Overflow: Two's Complement

- ❖ **Addition:**  $(+) + (+) = (-)$  result?

$$\begin{array}{r} 6 \\ + 3 \\ \hline \text{9} \\ \text{-7} \end{array} \quad \begin{array}{r} 0110 \\ + 0011 \\ \hline 1001 \end{array}$$

- ❖ **Subtraction:**  $(-) + (-) = (+)$ ?

$$\begin{array}{r} -7 \\ - 3 \\ \hline \text{-10} \\ \text{6} \end{array} \quad \begin{array}{r} 1001 \\ - 0011 \\ \hline 0110 \end{array}$$



For signed: overflow if operands have same sign and result's sign is different

17

## Shift Operations (Review)

- ❖ Throw away (drop) extra bits that “fall off” the end
- ❖ Left shift ( $x \ll n$ ) bit vector  $x$  by  $n$  positions
  - Fill with 0's on right
- ❖ Right shift ( $x \gg n$ ) bit-vector  $x$  by  $n$  positions
  - Logical shift (for **unsigned** values)
    - Fill with 0's on left
  - Arithmetic shift (for **signed** values)
    - Replicate most significant bit on left (maintains sign of  $x$ )

	x	0010 0010
	$x \ll 3$	0001 0000
logical:	$x \gg 2$	0000 1000
arithmetic:	$x \gg 2$	0000 1000

	x	1010 0010
	$x \ll 3$	0001 0000
logical:	$x \gg 2$	0010 1000
arithmetic:	$x \gg 2$	1110 1000

19

## Left Shifting Arithmetic 8-bit Example

- ❖ No difference in left shift operation for unsigned and signed numbers (just manipulates bits)
  - Difference comes during interpretation:  $x \cdot 2^n$ ?

			Signed	Unsigned
$x = 25;$	00011001	=	25	25
$L1 = x \ll 2;$	0001100100	=	100	100
$L2 = x \ll 3;$	00011001000	=	-56	200
			signed overflow	
$L3 = x \ll 4;$	000110010000	=	-112	144
			unsigned overflow	

21

## Right Shifting Arithmetic 8-bit Examples

- ❖ **Reminder:** C operator  $\gg$  does *logical* shift on **unsigned** values and *arithmetic* shift on **signed** values
  - **Logical** Shift:  $x / 2^n$

$x_u = 240_u;$	11110000	=	240
$R1_u = x_u \gg 3;$	00011110000	=	30
$R2_u = x_u \gg 5;$	0000011110000	=	7
			rounding (down)

22

## Right Shifting Arithmetic 8-bit Examples

- ❖ **Reminder:** C operator  $\gg$  does *logical* shift on **unsigned** values and *arithmetic* shift on **signed** values
  - **Arithmetic** Shift:  $x / 2^n$

$x_s = -16;$	11110000	=	-16
$R1_s = x_s \gg 3;$	11111110000	=	-2
$R2_s = x_s \gg 5;$	111111110000	=	-1
			rounding (down)

23