CS330 - Computer Organization and Assembly Language Programming

Lecture 6

Professor : Mahmut Unan – UAB CS

Agenda

- Integer Representations
  - Integral Data Types
  - Unsigned Encodings
  - Two’s Compliment Encodings
  - Conversions between Signed and Unsigned

Integral Data Types

- C supports several integral data types (ones that represent finite ranges of integers)
- Based on the byte allocations, the different sizes allow different ranges of values to be presented (32-bit vs 64-bit)
  - Note the unsigned modifier
  - Also note the asymmetric ranges

Typical Ranges for C Integral Data Types for 32-bit Programs

C data type (32b)	Size	Minimum	Maximum
char	1	-128	127
unsigned char	1	0	255
short int	2	-32,768	32,767
unsigned short int	2	0	65,535
int	4	-2,147,438,648	2,147,438,647
unsigned int	4	0	4,294,967,295
long int	4	-2,147,438,648	2,147,438,647
unsigned long int	4	0	4,294,967,295
long long int	8	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
unsigned long long int	8	0	18,446,744,073,709,551,615

Typical Ranges for C Integral Data Types for 64-bit Programs

C data type	Minimum	Maximum
[signed] char	−128	127
unsigned char	0	255
short	−32,768	32,767
unsigned short	0	65,535
int	−2,147,483,648	2,147,483,647
unsigned	0	4,294,967,295
long	−9,223,372,036,854,775,808	9,223,372,036,854,775,807
unsigned long	0	18,446,744,073,709,551,615
int32_t	−2,147,483,648	2,147,483,647
uint32_t	0	4,294,967,295
int64_t	−9,223,372,036,854,775,808	9,223,372,036,854,775,807
uint64_t	0	18,446,744,073,709,551,615

Unsigned Encodings

$$B2U(X) = \sum_{i=0}^{w-1} x_i \cdot 2^i$$

(Binary To Unsigned)

e.g. B2U ([1011]) = $1 * 2^3 + 0*2^2 + 1*2^1 + 1*2^0 = 11$

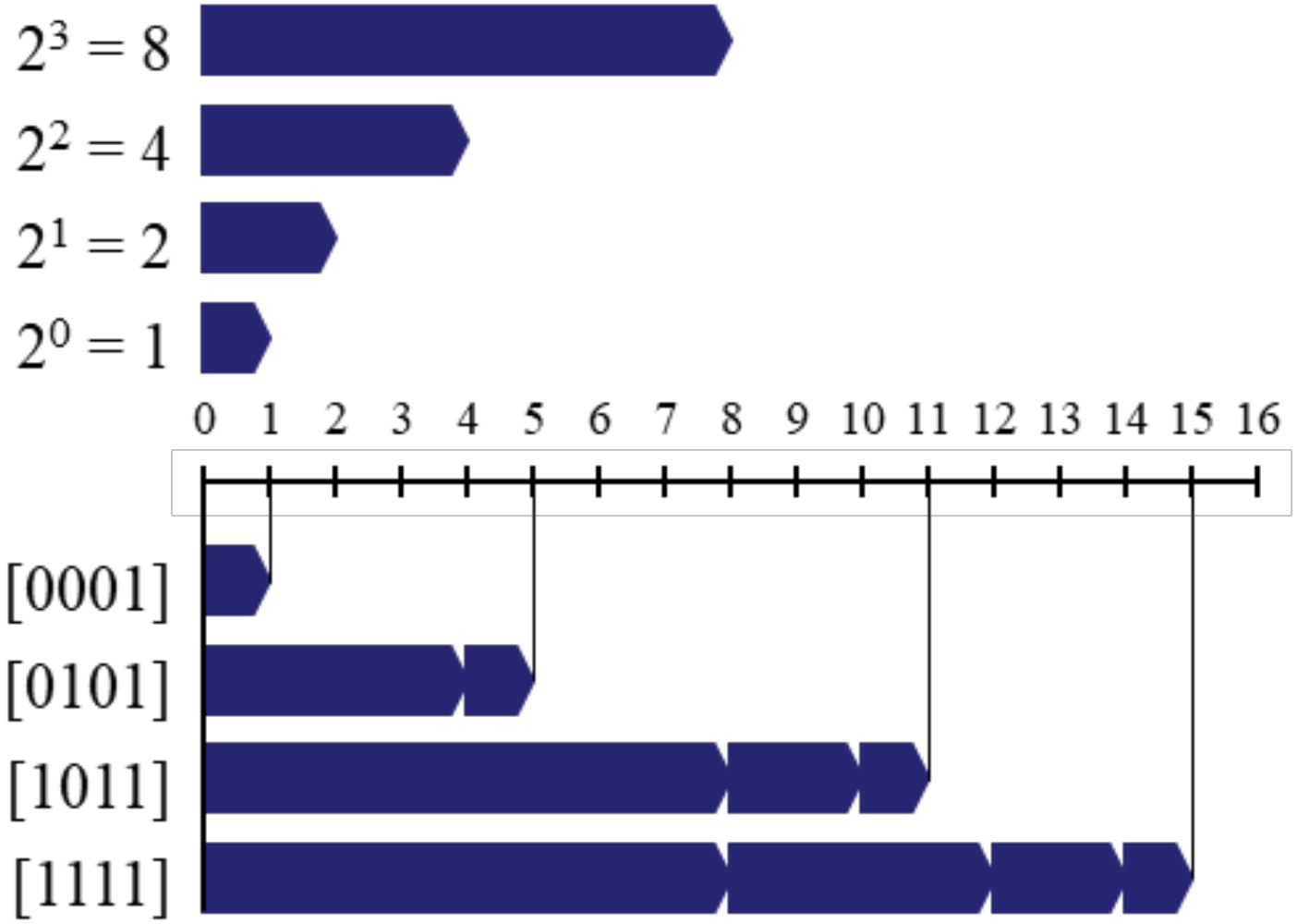
– C `short` 2 bytes long

```
short int x = 15213;
```

	Decimal	Hex	Binary
x	15213	3B 6D	00111011 01101101

http://www.aqualab.cs.northwestern.edu/component/attachments/download/168

Examples


$2^3 = 8$
$2^2 = 4$
$2^1 = 2$
$2^0 = 1$
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
[0001]
[0101]
[1011]
[1111]

Two's Complement

$$B2T(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$

Sign Bit

- e.g. B2T ([1011]) = $-1 * 2^3 + 0*2^2 + 1*2^1 + 1*2^0 = -5$
- C `short` 2 bytes long

```
short int y = -15213;
```

	Decimal	Hex	Binary
x	15213	3B 6D	00111011 01101101
y	-15213	C4 93	11000100 10010011

- Sign bit
  - For 2's complement, most significant bit indicates sign
    - 0 for nonnegative; 1 for negative

http://www.aqualab.cs.northwestern.edu/component/attachments/download/168

Two-complement Encoding Example (Cont.)

```
x =      15213: 00111011 01101101
y =     -15213: 11000100 10010011
```

Weight	15213		-15213	
1	1	1	1	1
2	0	0	1	2
4	1	4	0	0
8	1	8	0	0
16	0	0	1	16
32	1	32	0	0
64	1	64	0	0
128	0	0	1	128
256	1	256	0	0
512	1	512	0	0
1024	0	0	1	1024
2048	1	2048	0	0
4096	1	4096	0	0
8192	1	8192	0	0
16384	0	0	1	16384
-32768	0	0	1	-32768
Sum		**15213**		**-15213**

Two's Compliment

499
2

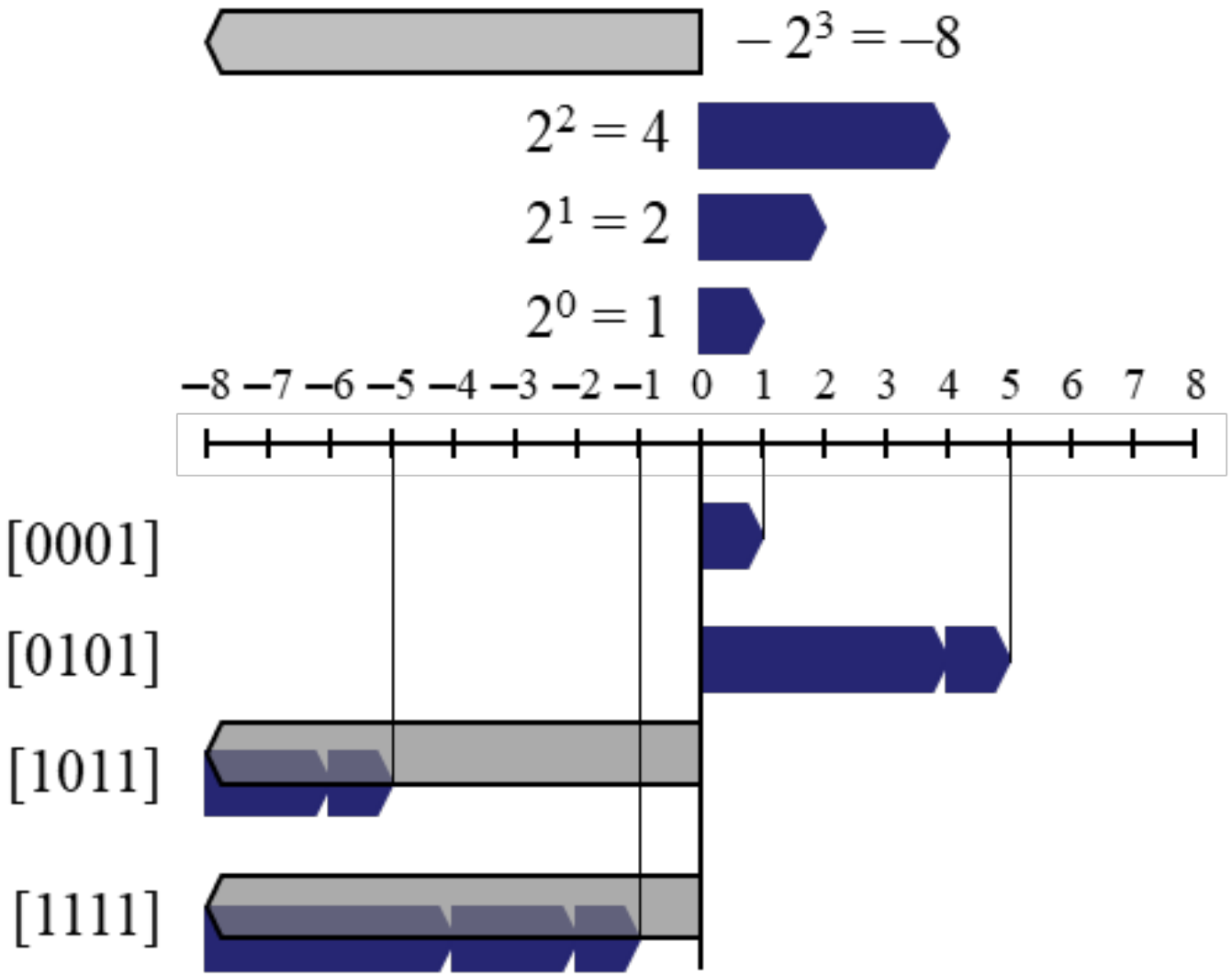
- Invert and add **one**

Suppose we're working with 8 bit quantities and suppose we want to find how **-28** would be expressed in two's complement notation.

- First we write out 28 in **binary form.**
  00011100
- Then we **invert the digits**. 0 becomes 1, 1 becomes 0.
  11100011
- Then we **add 1.**
  11100100

That is how one would write -28 in 8 bit binary.

11100011
1


$-2^3 = -8$
$2^2 = 4$
$2^1 = 2$
$2^0 = 1$
−8 −7 −6 −5 −4 −3 −2 −1 0 1 2 3 4 5 6 7 8
[0001]
[0101]
[1011]
[1111]

11111111111

1111111111111

01111111

10000000

Characteristics of Twos Complement Representation and Arithmetic

Range	-2_{n-1} through $2_{n-1} - 1$
Number of Representations of Zero	One
Negation	Take the Boolean complement of each bit of the corresponding positive number, then add 1 to the resulting bit pattern viewed as an unsigned integer.
Expansion of Bit Length	Add additional bit positions to the left and fill in with the value of the original sign bit.
Overflow Rule	If two numbers with the same sign (both positive or both negative) are added, then overflow occurs if and only if the result has the opposite sign.
Subtraction Rule	To subtract *B* from *A*, take the twos complement of *B* and add it to *A*.

–128	64	32	16	8	4	2	1

(a) An eight-position two's complement value box

–128	64	32	16	8	4	2	1
1	0	0	0	0	0	1	1

–128 +2 +1 = –125

(b) Convert binary 10000011 to decimal

–128	64	32	16	8	4	2	1
1	0	0	0	1	0	0	0

–120 = –128 +8

(c) Convert decimal –120 to binary

Numeric Ranges

- **Unsigned Values**
  - *UMin* = 0

    000…0
  - *UMax* = $2^w - 1$

    111…1
- **Two's Complement Values**
  - *TMin* = -2^{w-1}

    100…0
  - *TMax* = $2^{w-1} - 1$

    011…1
- **Other Values**
  - Minus 1

    111…1

Values for *W* = 16

	Decimal	Hex	Binary
UMax	65535	FF FF	11111111 11111111
TMax	32767	7F FF	01111111 11111111
TMin	-32768	80 00	10000000 00000000
-1	-1	FF FF	11111111 11111111
0	0	00 00	00000000 00000000

Values for Different Word Sizes

	W			
	8	**16**	**32**	**64**
UMax	255	65,535	4,294,967,295	18,446,744,073,709,551,615
TMax	127	32,767	2,147,483,647	9,223,372,036,854,775,807
TMin	-128	-32,768	-2,147,483,648	-9,223,372,036,854,775,808

- **Observations**
  - | *TMin* | = *TMax* + 1
    - Asymmetric range
  - *UMax* = 2 * *TMax* + 1
- **C Programming**
  - #include <limits.h>
  - Declares constants, e.g.,
    - ULONG_MAX
    - LONG_MAX
    - LONG_MIN
  - Values platform specific

Casting signed to unsigned

- C allows conversions from signed to unsigned

```
short int          x =  15213;
unsigned short int ux = (unsigned short) x;
short int          y  = -15213;
unsigned short int uy = (unsigned short) y;
```

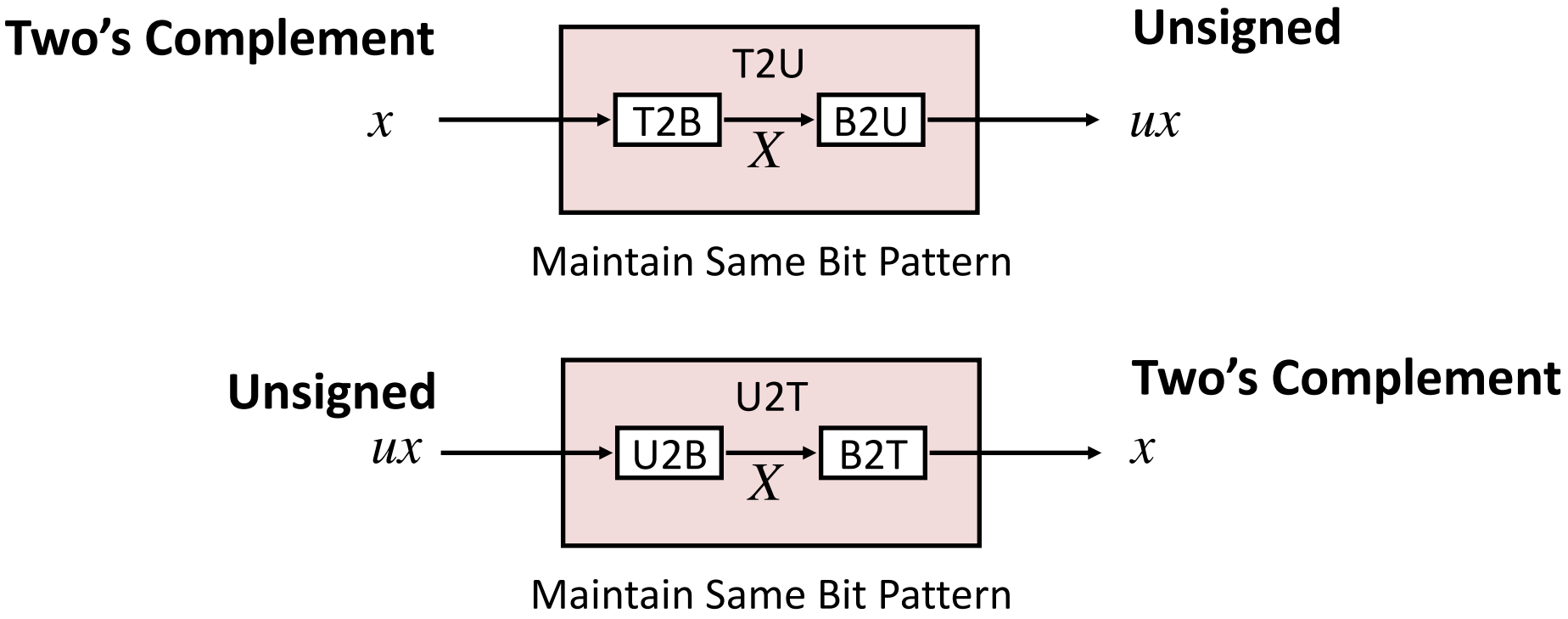
- Resulting value
  - Not based on a numeric perspective
  - No change in bit representation
  - Non-negative values unchanged
    - ux = 15213
  - Negative values change into (large) positive values
    - uy = 50323

Unsigned & Signed Numeric Values

X	B2U(*X*)	B2T(*X*)
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	–8
1001	9	–7
1010	10	–6
1011	11	–5
1100	12	–4
1101	13	–3
1110	14	–2
1111	15	–1

- **Equivalence**
  - Same encodings for nonnegative values
- **Uniqueness**
  - Every bit pattern represents unique integer value
  - Each representable integer has unique bit encoding
- **$\Rightarrow$ Can Invert Mappings**
  - $U2B(x) = B2U^{-1}(x)$
    - Bit pattern for unsigned integer
  - $T2B(x) = B2T^{-1}(x)$
    - Bit pattern for two's comp integer

Mapping Between Signed & Unsigned



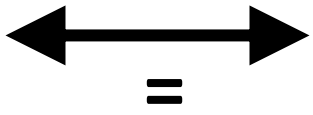
- **Mappings between unsigned and two's complement numbers:**
  Keep bit representations and reinterpret

Mapping Signed ↔ Unsigned

Bits	Signed	Unsigned
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	-8	8
1001	-7	9
1010	-6	10
1011	-5	11
1100	-4	12
1101	-3	13
1110	-2	14
1111	-1	15

Mapping Signed ↔ Unsigned

Bits	Signed	Unsigned
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	-8	8
1001	-7	9
1010	-6	10
1011	-5	11
1100	-4	12
1101	-3	13
1110	-2	14
1111	-1	15



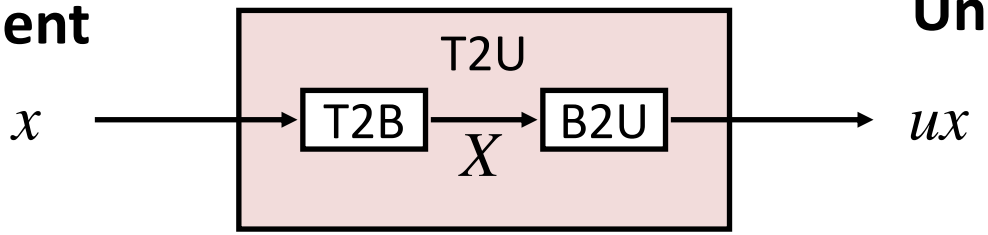
=

+/- 16

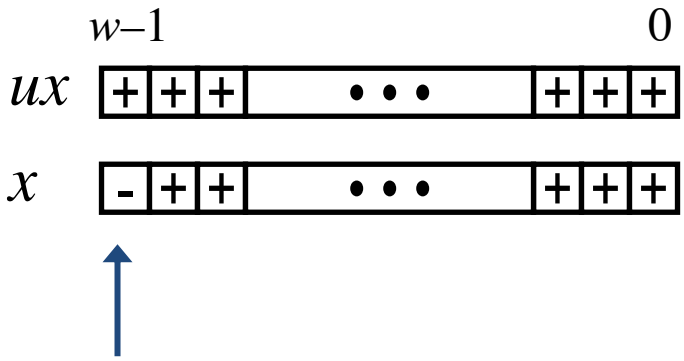
Relation between Signed & Unsigned

Two's Complement

Unsigned



Maintain Same Bit Pattern



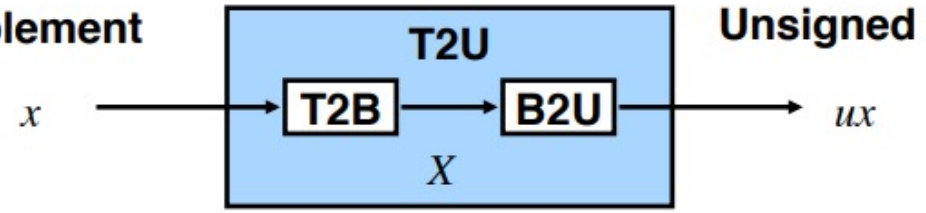
Large negative weight

becomes

Large positive weight

Relation between Signed & Unsigned / 2

Casting from signed to unsigned



Maintain same bit pattern

Consider B2U and B2T equations

$$B2U(X) = \sum_{i=0}^{w-1} x_i \cdot 2^i \qquad B2T(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$

and a bit pattern X; compute $B2U(X) - B2T(X)$
weighted sum of for bits from 0 to w – 2 cancel each other

$$B2U(X) - B2T(X) = x_{w-1}(2^{w-1} - -2^{w-1}) = x_{w-1}2^w$$
$$B2U(X) = x_{w-1}2^w + B2T(X)$$

If we let $B2T(X) = x$

$$B2U(T2B(x)) = T2U(x) = x_{w-1}2^w + x$$

Sign bit

$$ux = \begin{cases} x & x \geq 0 \\ x + 2^w & x < 0 \end{cases}$$

Relation between Signed & Unsigned / 3

$$T2U(x) = x_{w-1}2^w + x$$

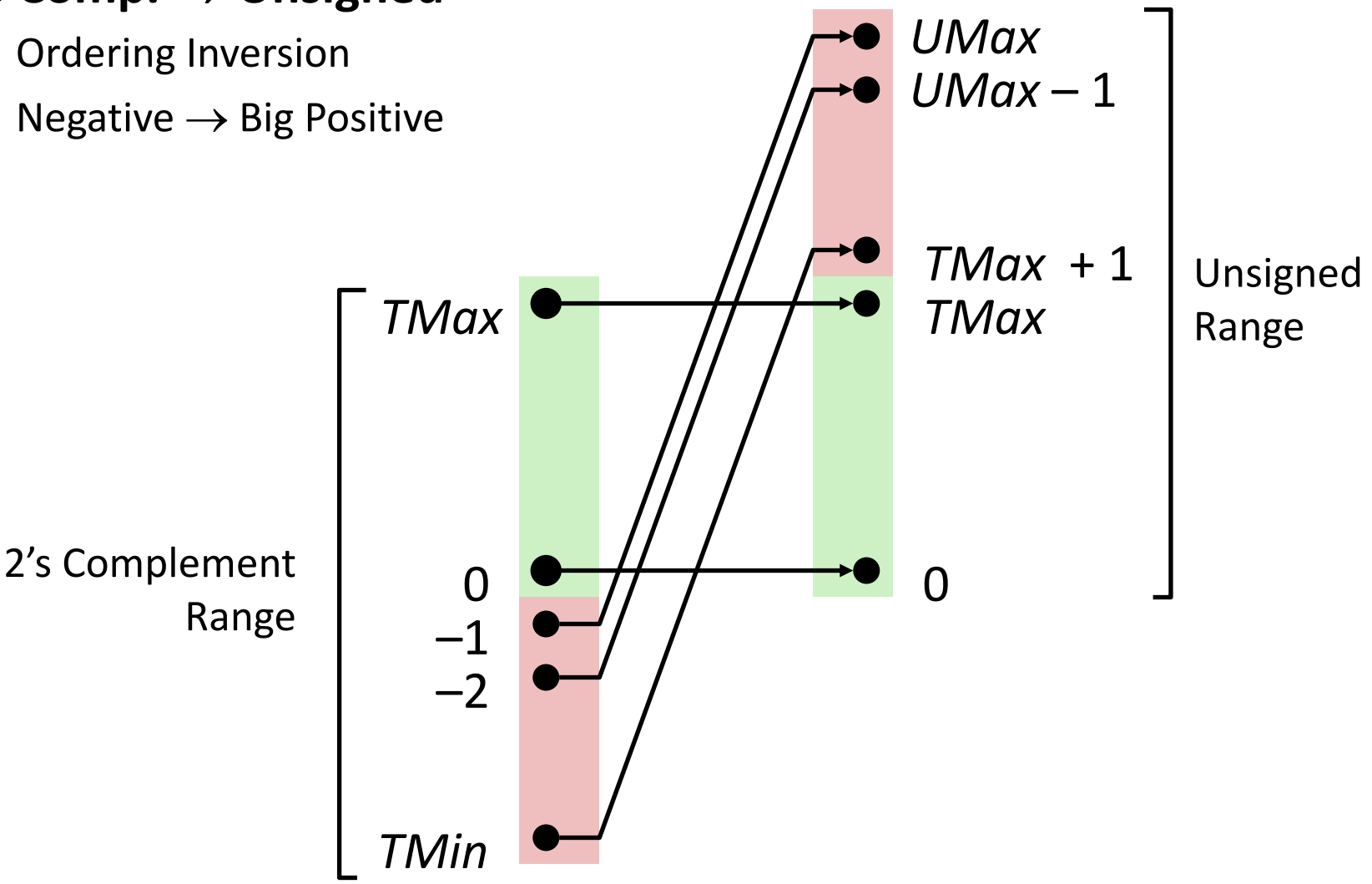
Weight	15213		-15213		50323	
1	1	1	1	1	1	1
2	0	0	1	2	1	2
4	1	4	0	0	0	0
8	1	8	0	0	0	0
16	0	0	1	16	1	16
32	1	32	0	0	0	0
64	1	64	0	0	0	0
128	0	0	1	128	1	128
256	1	256	0	0	0	0
512	1	512	0	0	0	0
1024	0	0	1	1024	1	1024
2048	1	2048	0	0	0	0
4096	1	4096	0	0	0	0
8192	1	8192	0	0	0	0
16384	0	0	1	16384	1	16384
-/+32768	0	0	1	-32768	1	32768
Sum		15213		-15213		50323

$$ux = x + 2^{16} = -15213 + 65536$$

http://www.aqualab.cs.northwestern.edu/component/attachments/download/168

Conversion Visualized

- **2's Comp. → Unsigned**
  - Ordering Inversion
  - Negative → Big Positive



Exercises

- Unsigned Numbers

1001 0010 = 2+16+128 =136

010101= 1+4+16 = 21

- Signed Numbers

10100 = 4+ -16 =-12

11100 =4+8+ -16 = -4

01111= 1+2+4+8 = 15

Convert Unsigned to Two's Complement - 16bit

- 9 =0000 0000 0000 1001
- 1 = 0000 0000 0000 0001
- 14= 0000 0000 0000 1110

- -127 = 1111 1111 1000 0001

Signed vs. Unsigned in C

- Constants
  - By default are considered to be signed integers
  - Unsigned if have “U” as suffix

    ```
    5U, 4294967259U
    ```
- Casting
  - Explicit casting between signed & unsigned same as U2T and T2U

    ```
    int tx, ty;
    unsigned ux, uy;
    tx = (int) ux;
    uy = (unsigned) ty;
    ```

  - Implicit casting also occurs via assignments and procedure calls

    ```
    tx = ux;
    uy = ty;
    ```

Casting Surprises

- Expression Evaluation
  - If there is a mix of unsigned and signed in single expression, ***signed values implicitly cast to unsigned***
  - Including comparison operations **<, >, ==, <=, >=**
  - Examples for *W* = 32: **TMIN = -2,147,483,648 , TMAX = 2,147,483,647**

Constant$_1$	**Constant$_2$**	**Relation**	**Evaluation**
0	0U	==	unsigned
-1	0	<	signed
-1	0U	>	unsigned
2147483647	-2147483647-1	>	signed
2147483647U	-2147483647-1	<	unsigned
-1	-2	>	signed
(unsigned)-1	-2	>	unsigned
2147483647	2147483648U	<	unsigned
2147483647	(int) 2147483648U	>	signed

Question

- How many times the code prints “Hello” ?

```
unsigned i;

for (i=50; i>=0; i--){
   printf(“Hello”);
}
```

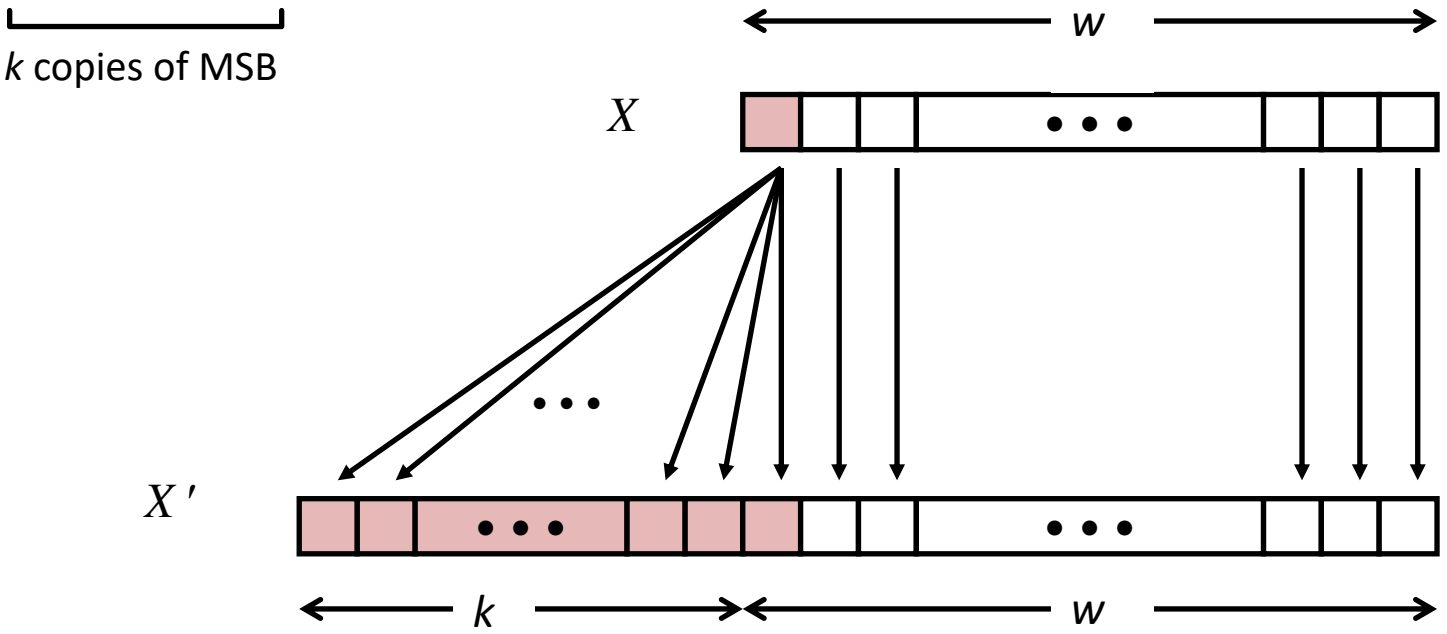
Summary
Casting Signed ↔ Unsigned: Basic Rules

- Bit pattern is maintained
- But reinterpreted
- Can have unexpected effects: adding or subtracting 2^w

- Expression containing signed and unsigned int
  - `int` is cast to `unsigned`!!

Sign Extension

- **Task:**
  - Given *w*-bit signed integer *x*
  - Convert it to *w*+*k*-bit integer with same value
- **Rule:**
  - Make *k* copies of sign bit:
  - $X' = x_{w-1}, \ldots, x_{w-1}, x_{w-1}, x_{w-2}, \ldots, x_0$



Sign Extension Example

```
short int x =  15213;
int      ix = (int) x;
short int y = -15213;
int      iy = (int) y;
```

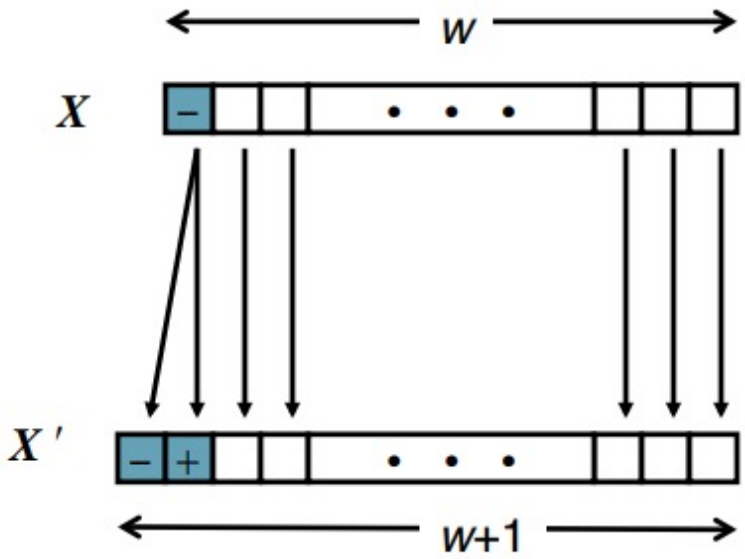
	Decimal	Hex	Binary
x	15213	3B 6D	00111011 01101101
ix	15213	00 00 3B 6D	00000000 00000000 00111011 01101101
y	-15213	C4 93	11000100 10010011
iy	-15213	FF FF C4 93	11111111 11111111 11000100 10010011

- Converting from smaller to larger integer data type
- C automatically performs sign extension

Justification for sign extension

- Prove correctness by induction on k
  - Induction Step: extending by single bit maintains value

$$B2T(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$



  - Key observation: $2^w - 2^{w-1} = 2^{w-1}$
  - Look at weight of upper bits:
    - X $-2^{w-1} x_{w-1}$
    - X' $-2^w x_{w-1} + 2^{w-1} x_{w-1} = -2^{w-1} x_{w-1}$

- 0001
- 0000 0001
- 1111
- 1111 1111

Next Class

- We will continue **Chapter 2**
- Please read **2.2.5 – 2.3.3**