

15-213

“The course that gives CMU its Zip!”

Integer Representations

Aug. 31, 1999

Topics

- **Numeric Encodings**
 - Unsigned & Two’s complement
- **Programming Implications**
 - C promotion rules

Notation

W: Number of Bits in “Word”

C Data Type	Typical 32-bit	Alpha
long int	32	64
int	32	32
short	16	16
char	8	8

Integers

- Lower case
- E.g., x , y , z

Bit Vectors

- Upper Case
- E.g., X , Y , Z
- Write individual bits as integers with value 0 or 1
- E.g., $X = x_{w-1}, x_{w-2}, \dots, x_0$
 - Most significant bit on left

Encoding Integers

Unsigned

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

Two's Complement

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

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

Sign Bit

- C short 2 bytes long

	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

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

Other Encoding Schemes

Other less common encodings

- **One's complement:** Invert bits for negative numbers
- **Sign magnitude:** Invert sign bit for negative numbers

`short int`

15213	Unsigned	00111011 01101101
-15213	Two's complement	11000100 10010011
-15213	One's complement	11000100 10010010
-15213	Sign magnitude	10111011 01101101

ISO C does not define what encoding machines use for signed integers, but 95% (or more) use two's complement.

For truly portable code, don't count on it.

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>`
 - Harbison and Steele, 5.1
- **Declares constants, e.g.,**
 - `ULONG_MAX`
 - `LONG_MAX`
 - `LONG_MIN`
- **Values platform-specific**

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

Example Values

- $W = 4$

Equivalence

- Same encodings for nonnegative values

Uniqueness

- Every bit pattern represents unique integer value
- Each representable integer has unique bit encoding

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

Casting Signed to Unsigned

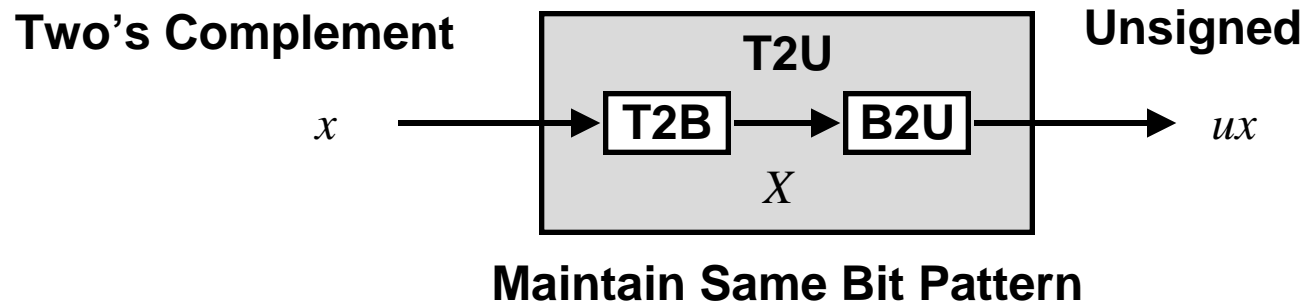
C Allows Conversions from Signed to Unsigned

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

Resulting Value

- No change in bit representation
- Nonnegative values unchanged
 - $ux = 15213$
- Negative values change into (large) positive values
 - $uy = 50323$

Relation Between 2's Comp. & Unsigned



$$\begin{array}{r}
 \begin{array}{c}
 w-1 \qquad \qquad \qquad 0 \\
 ux \quad \boxed{+ \mid + \mid + \mid \dots \mid \dots \mid \dots \mid + \mid + \mid +} \\
 - \quad x \quad \boxed{- \mid + \mid + \mid \dots \mid \dots \mid \dots \mid + \mid + \mid +} \\
 \hline
 +2^{w-1} - -2^{w-1} = 2 * 2^{w-1} = 2^w
 \end{array}
 \end{array}$$

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

Relation Between Signed & Unsigned

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

- $uy = y + 2 * 32768 = y + 65536$

From Two's Complement to Unsigned

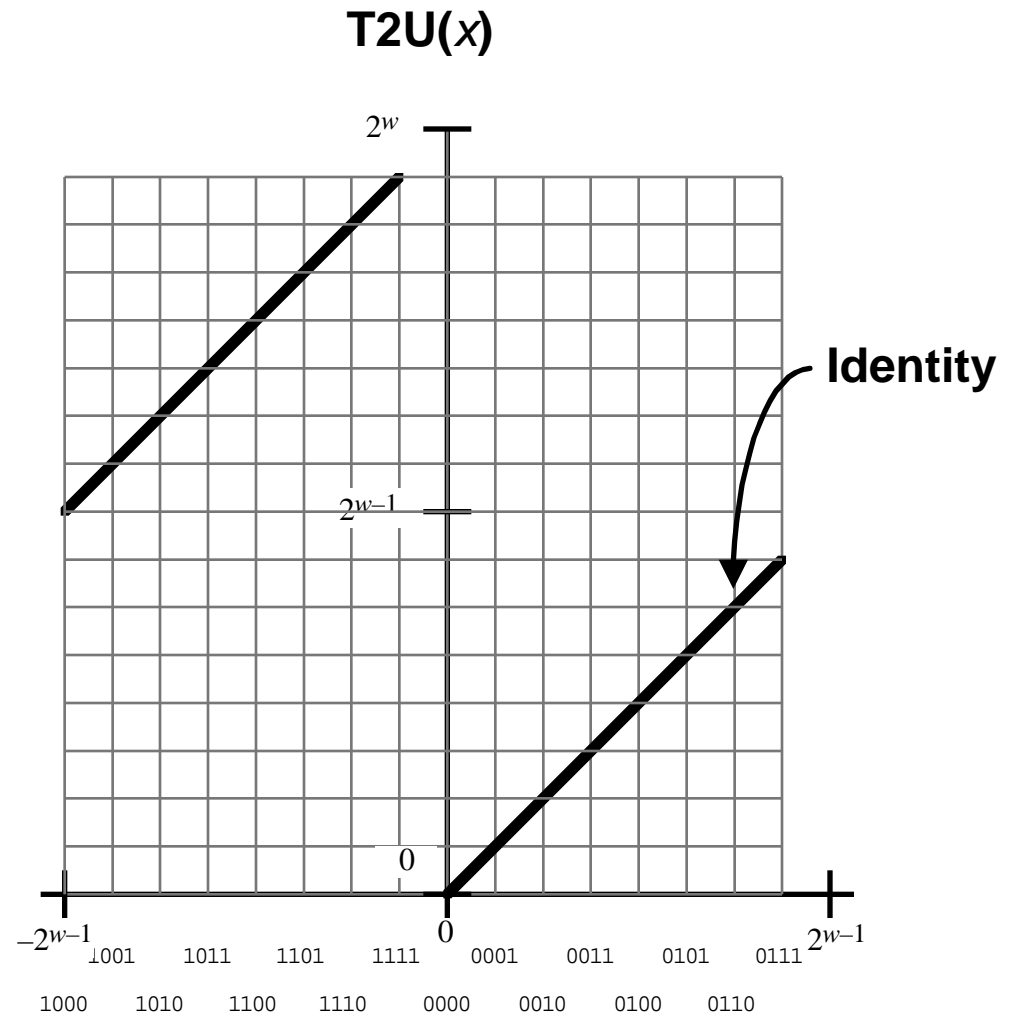
- $T2U(x)$
 $= B2U(T2B(x))$
 $= x + x_{w-1} 2^w$

- What you get in C:

```
unsigned t2u(int x)
{
    return (unsigned) x;
}
```

X	$B2U(X)$	$B2T(X)$
1010	10	-6

← +16 —



From Unsigned to Two's Complement

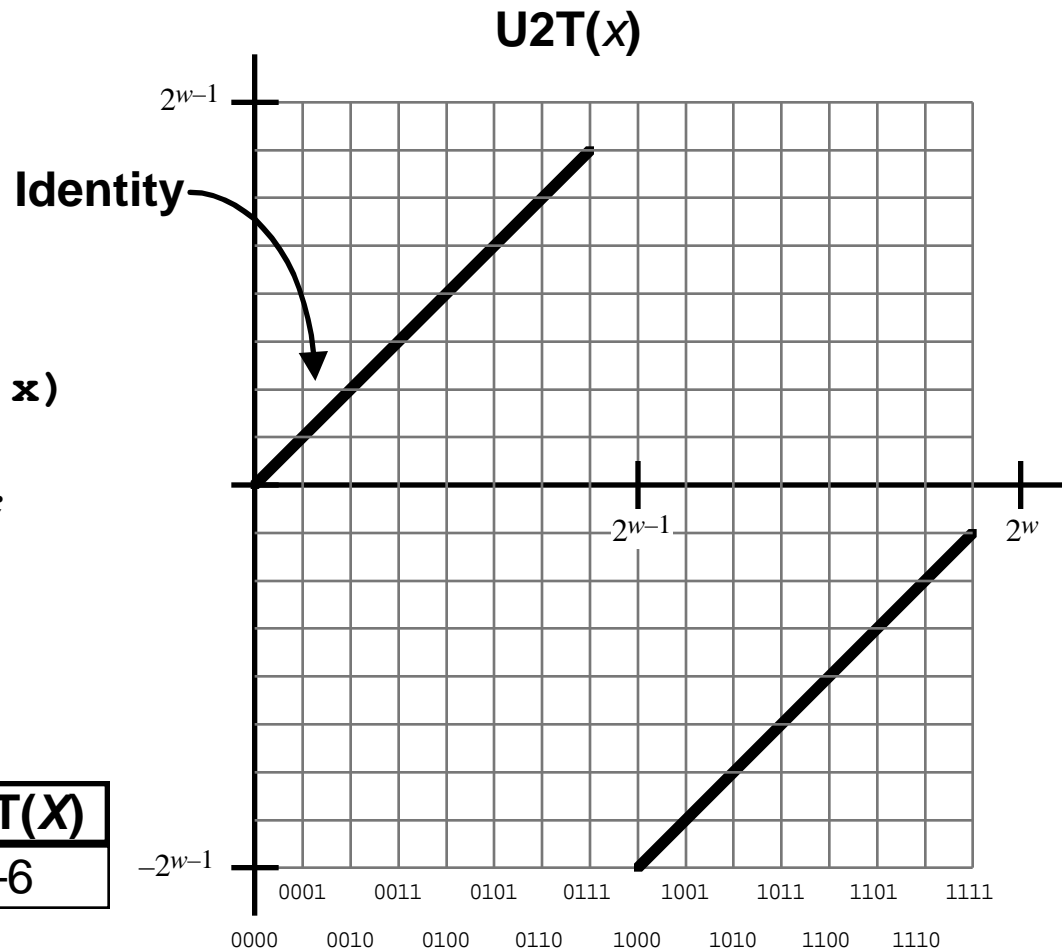
- $U2T(x)$
 $= B2T(U2B(x))$
 $= x - x_{w-1} 2^w$

- What you get in C:

```
int u2t(unsigned x)
{
    return (int) x;
}
```

X	$B2U(X)$	$B2T(X)$
1010	10	-6

— -16 →



Signed vs. Unsigned in C

Constants

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

0U, 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

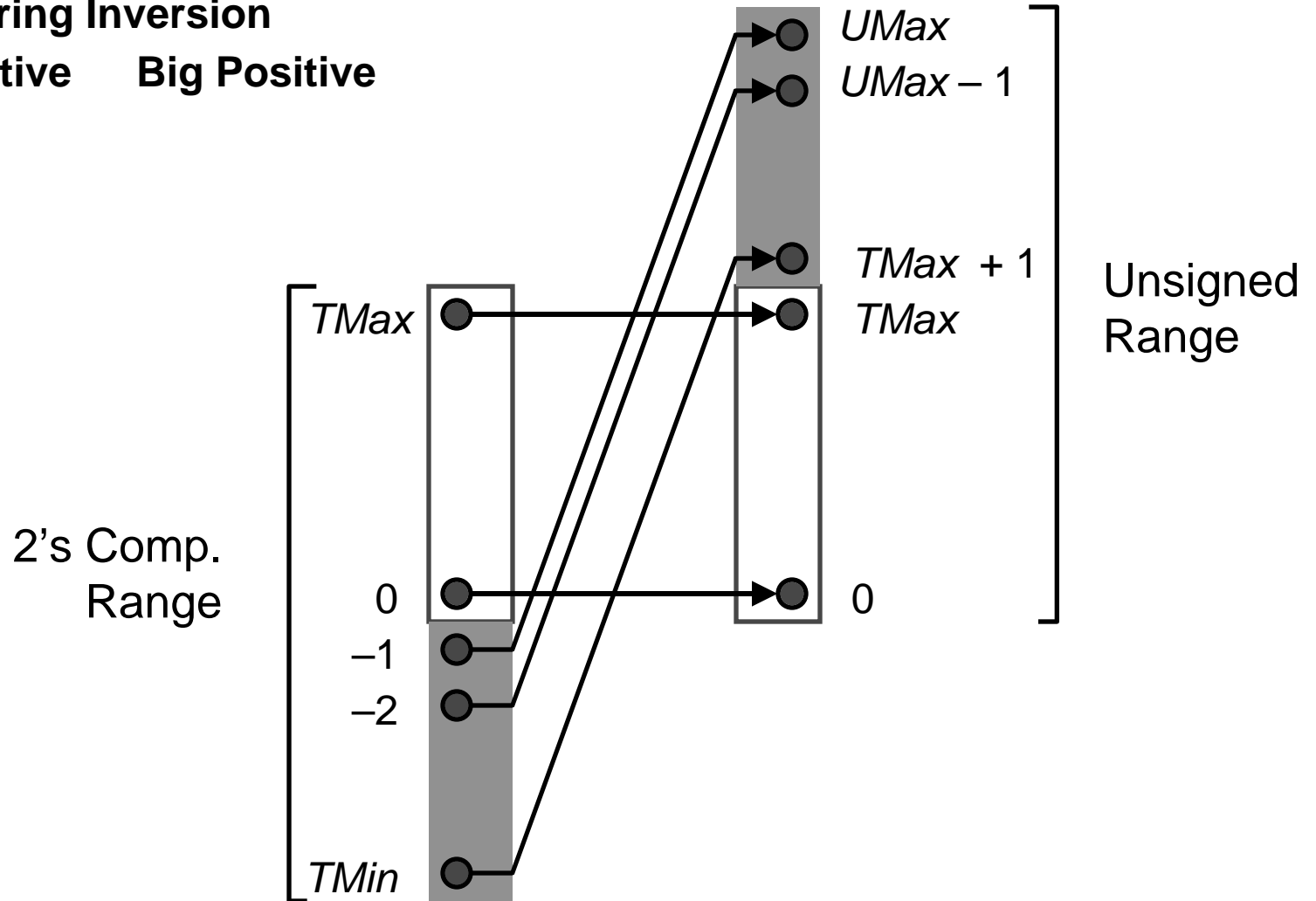
- If mix unsigned and signed in single expression, signed values implicitly cast to unsigned
- Including comparison operations $<$, $>$, $==$, $<=$, $>=$
- Examples for $W = 32$

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

Explanation of Casting Surprises

2's Comp. Unsigned

- Ordering Inversion
- Negative Big Positive



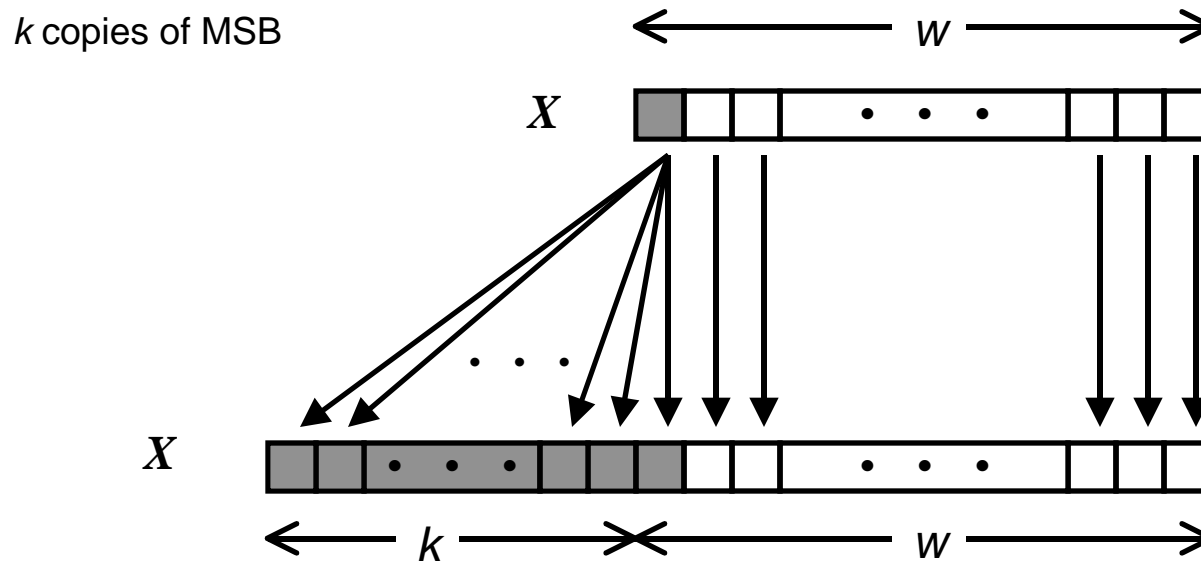
Sign Extension

Task:

- Given w -bit signed integer x
- Convert it to $w+k$ -bit integer with same value

Rule:

- Make k copies of sign bit:
- $X = \underbrace{X_{w-1}, \dots, X_{w-1}}_{k \text{ copies of MSB}}, X_{w-1}, X_{w-2}, \dots, X_0$



Sign Extension Example

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

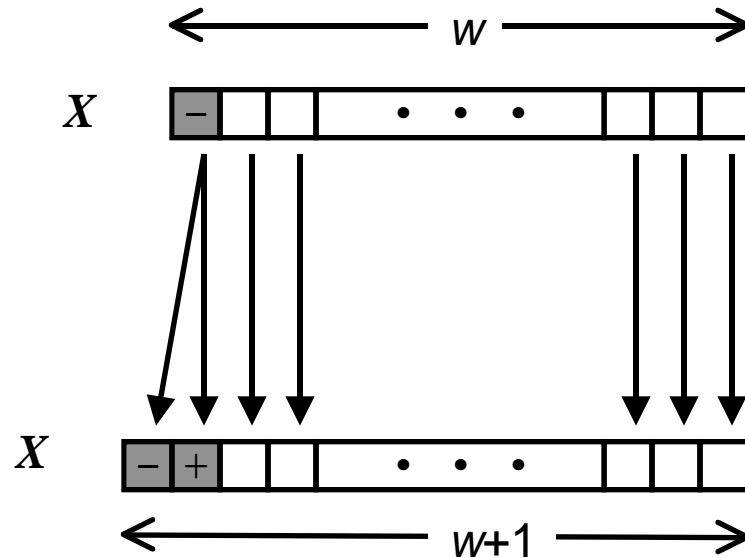
	Decimal	Hex	Binary
x	15213	3B 6D	00111011 01101101
ix	15213	00 00 C4 92	00000000 00000000 00111011 01101101
y	-15213	C4 93	11000100 10010011
iy	-15213	FF FF C4 93	11111111 11111111 11000100 10010011

- 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



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

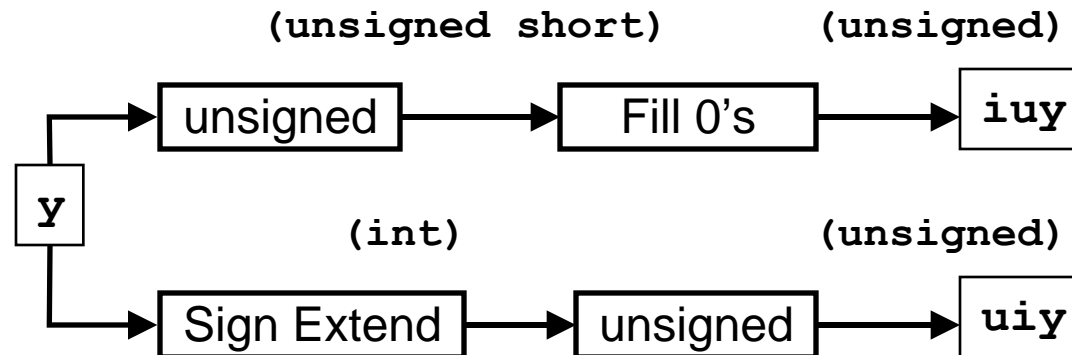
$$X \quad -2^{w-1} X_{w-1}$$

$$X \quad -2^w X_{w-1} + 2^{w-1} X_{w-1} = -2^{w-1} X_{w-1}$$

Casting Order Dependencies

```

short int x = 15213;
short int y = -15213;
unsigned iux = (unsigned) (unsigned short) x;
unsigned iuy = (unsigned) (unsigned short) y;
unsigned uix = (unsigned) (int) x;
unsigned uiy = (unsigned) (int) y;
unsigned uuy = y;
    
```



```

iux =      15213: 00000000 00000000 00111011 01101101
iuy =      50323: 00000000 00000000 11000100 10010011
uix =      15213: 00000000 00000000 00111011 01101101
uiy = 4294952083: 11111111 11111111 11000100 10010011
uuy = 4294952083: 11111111 11111111 11000100 10010011
    
```

Why Should I Use Unsigned?

Don't Use Just Because Number Nonzero

- C compiler on Alpha generates less efficient code
 - Comparable code on Intel/Linux

```
unsigned i;  
for (i = 1; i < cnt; i++)  
    a[i] += a[i-1];
```

- Easy to make mistakes

```
for (i = cnt-2; i >= 0; i--)  
    a[i] += a[i+1];
```

Do Use When Performing Modular Arithmetic

- Multiprecision arithmetic
- Other esoteric stuff

Do Use When Need Extra Bit's Worth of Range

- Working right up to limit of word size