Integer Encoding

Karthik Dantu
Ethan Blanton
Computer Science and Engineering
University at Buffalo
kdantu@buffalo.edu

Portions of this lecture are taken from Harvey Mudd College CS 105

University at Buffalo
Department of Computer Science and Engineering
School of Engineering and Applied Sciences

Encoding Integers - Unsigned

- Bijection

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

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

	Decimal	Hex	Binary
x	15213	3B 6D	00111011 01101101

Karthik Dantu

Encoding Integers - Signed

- 2s complement

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

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

	Decimal	Hex	Binary
y	-15213	C4 93	11000100 10010011

Encoding Integers

```
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**

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

Width = 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

Ranges - 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

$$|TMin| \quad = \qquad TMax + 1$$

Asymmetric range

$$UMax \quad = \qquad 2 * TMax + 1$$

```
#include <limits.h>
```

K&R App. B11

Declares constants, e.g.,

```
ULONG_MAX
LONG_MAX
LONG_MIN
```

Values platform-specific

Signed/Unsigned Numeric Values

- Equivalence

  Same encodings for nonnegative values

- Uniqueness

  Every bit pattern represents unique integer value

  Each representable integer has unique bit encoding

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

University at Buffalo
Department of Computer Science and Engineering
School of Engineering and Applied Sciences

Converting 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

Signed and 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**

$uy \quad = \quad y + 2 * 32768 = \quad y + 65536$

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;
```

C 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`		
`-1`	`0`		
`-1`	`0u`		
`2147483647`	`-2147483648`		
`2147483647u`	`-2147483648`		
`-1`	`-2`		
`(unsigned) -1`	`-2`		
`2147483647`	`2147483648u`		
`2147483647`	`(int) 2147483648u`		

University at Buffalo
Department of Computer Science and Engineering
School of Engineering and Applied Sciences

C 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

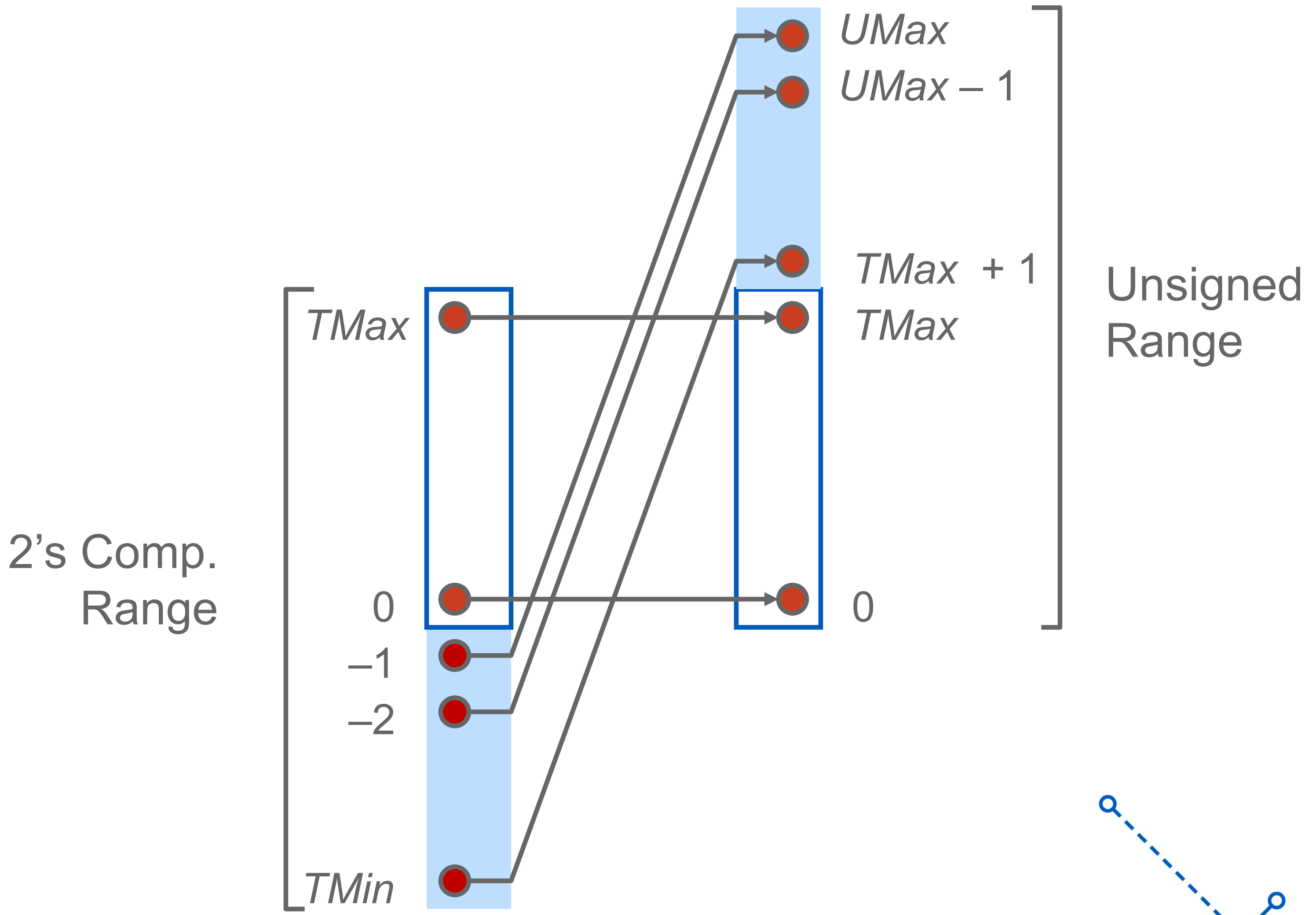
- Constant$_1$ / Constant$_2$ / Relation / Evaluation

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**

Karthik Dantu

Casting Visualization

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

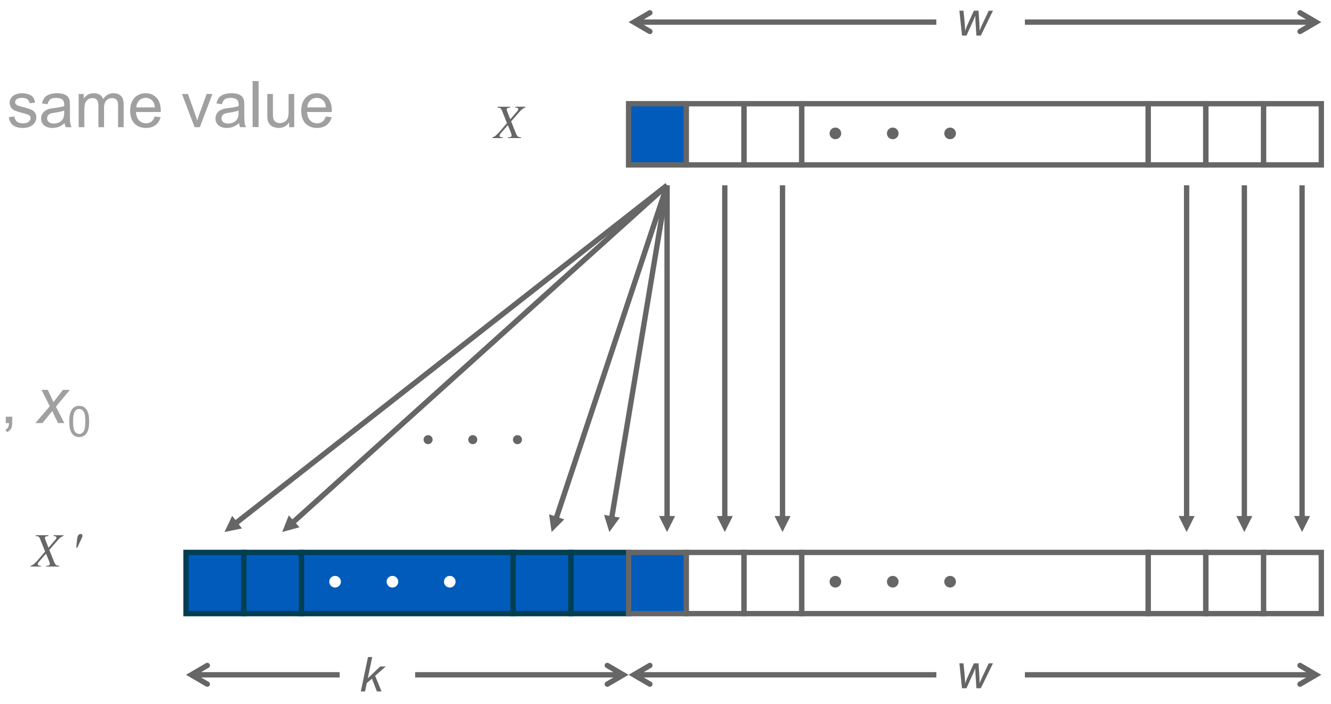


University at Buffalo
Department of Computer Science and Engineering
School of Engineering and Applied Sciences

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:

$$X' = \underbrace{x_{w-1}, \ldots, x_{w-1}}_{k \text{ copies of MSB}}, x_{w-1}, x_{w-2}, \ldots, x_0$$



Karthik Dantu

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

Homework (Optional)

For each of the following C expressions, either:

- Argue that it is true for all argument values
- Give example where it is not true

Initialization

```
int x = foo();
int y = bar();
unsigned ux = x;
unsigned uy = y;
```

- `x < 0` ⇒ `((x*2) < 0)`
- `ux >= 0`
- `x & 7 == 7` ⇒ `(x<<30) < 0`
- `ux > -1`
- `x > y` ⇒ `-x < -y`
- `x * x >= 0`
- `x > 0 && y > 0` ⇒ `x + y > 0`
- `x >= 0` ⇒ `-x <= 0`
- `x <= 0` ⇒ `-x >= 0`

Homework 2 (Optional)

- Write your own `dump_mem()` method to read memory byte-by-byte from a given pointer and display it in hex
- Syntax: `void dump_mem(void *ptr, int size);`
- Hint: `char` is a data type of size 1 byte. Casting to `char` allows you to read any data byte-by-byte

Required readings

- B&O 2.2