Lecture 9

Integer Representations

CPSC 275
Introduction to Computer Systems

Encoding Integers

Unsigned

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

where *X* is a *w*-bit binary representation of an integer

Two's Complement

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

Sign Bit

Encoding Integers, cont'd

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

- 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, MSB 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**

Numeric Ranges

- Unsigned Values
  - *UMin*
    - 000…0 = 0
  - *UMax*
    - 111…1 = $2^w - 1$
- Two's Complement Values
  - *TMin*
    - 100…0 = -2^{w-1}
  - *TMax*
    - 011…1 = $2^{w-1} - 1$
- Other Values
  - –1 in 2's complement?
    - 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

Unsigned & Signed Numeric Values

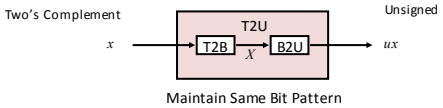
X	B2U(X)	B2T(X)
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	–8
1001	9	–7
1010	10	–6
1011	11	–5
1100	12	–4
1101	13	–3
1110	14	–2
1111	15	–1

- Uniqueness: Each representable integer has unique bit encoding
- Mappings between unsigned and two's complement numbers: Keep bit representations and reinterpret

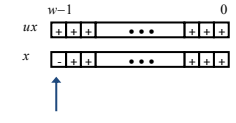
Mapping Signed ↔ Unsigned

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	+ 16	8
1001	–7	+ 16	9
1010	–6	+ 16	10
1011	–5	+ 16	11
1100	–4	+ 16	12
1101	–3	+ 16	13
1110	–2	+ 16	14
1111	–1	+ 16	15

Relation between Signed & Unsigned



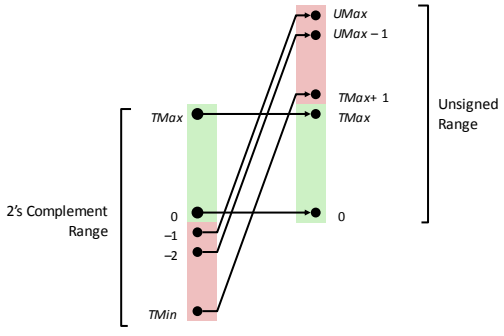
Maintain Same Bit Pattern



Large negative weight *becomes* Large positive weight

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

2's Comp. → Unsigned Conversion (T2U)



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 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

Code Security Example

```
/* Kernel memory region holding user-accessible data */
#define KSIZE 1024
char kbuf[KSIZE];

/* Copy at most maxlen bytes from kernel region to user buffer */
int copy_from_kernel(void *user_dest, int maxlen) {
    int len = KSIZE < maxlen ? KSIZE : maxlen;
    memcpy(user_dest, kbuf, len);
    return len;
}
```

Typical Usage

```
/* Kernel memory region holding user-accessible data */
#define KSIZE 1024
char kbuf[KSIZE];

/* Copy at most maxlen bytes from kernel region to user buffer */
int copy_from_kernel(void *user_dest, int maxlen) {
    int len = KSIZE < maxlen ? KSIZE : maxlen;
    memcpy(user_dest, kbuf, len);
    return len;
}
```

```
#define MSIZE 528

void getstuff() {
    char mybuf[MSIZE];
    copy_from_kernel(mybuf, MSIZE);
    printf("%s\n", mybuf);
}
```

Malicious Usage

```
/* Declaration of library function memcpy */
void *memcpy(void *dest, void *src, size_t n);
```

```
/* Kernel memory region holding user-accessible data */
#define KSIZE 1024
char kbuf[KSIZE];

/* Copy at most maxlen bytes from kernel region to user buffer */
int copy_from_kernel(void *user_dest, int maxlen) {
    int len = KSIZE < maxlen ? KSIZE : maxlen;
    memcpy(user_dest, kbuf, len);
    return len;
}
```

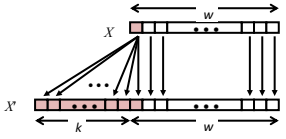
```
#define MSIZE 528

void getstuff() {
  char mybuf[MSIZE];
  copy_from_kernel(mybuf, -MSIZE);
    . . .
}
```

What's happening?

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:



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

Practice Problems

- Read CSaPP Sec. 2.2 and try the following problems:

  2.17, 2.19, 2.22, 2.23