Bits, Bytes, and Integers

CSCI 237: Computer Organization
5th Lecture, Monday, Sept. 16

Kelly Shaw

1

1

Administrative Details

- Lab #1 due Tuesday at 11pm
  - Any questions?
  - Lab video on Glow – using gdb for bit puzzles
- Read CSAPP 2.2
- In lecture practice problems for last week posted on Glow
- WICS Info Session
  - Today at 5:30pm in CS Common Area
  - Snacks included
- SnackAndGab
  - Wednesday 4:10-4:30 CS Common Room

2

2

Last Time: Bits, Bytes, and Integers

- Number representation
  - Conversion between decimal and binary
  - Conversion between hexadecimal, decimal, and binary
- Relationship between digits and range of unique values
- Bit-level manipulations
  - **Any questions from 15 minute video on Glow?**

3

3

Today: Bits, Bytes, and Integers

- Integers (Ch 2.2)
  - Representation: unsigned and signed
  - Conversion, casting
  - Expanding, truncating
  - Addition, negation, multiplication, shifting (Ch 2.3)

4

4

Encoding Integers

Unsigned

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

Binary To Unsigned

Two's Complement

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

Binary To Two's Complement

Sign Bit

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

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

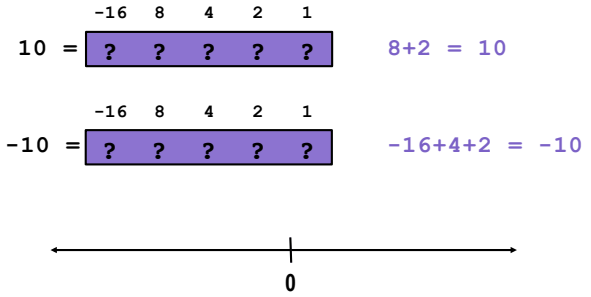
(Recall that In C, a `short` is 2 bytes long)

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

5

5

Two's-complement Example with 5 bit numbers



How to represent the smallest signed value using **5** bits?
How to represent the largest signed value using **5** bits?
Can you generalize to **w** bits?

6

6

Two's-complement Encoding Example (Cont.)

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

	Weight	15213		-15213	
2^i	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
2^{-i}	-32768	0	0	1	-32768
	Sum		**15213**		**-15213**

7

7

Nice Feature: Flip all the Bits and Add 1

10:	**01010**	
Flip bits:	**10101**	
Add 1:	**10110**	**-16+4+2 = -10**
Flip bits:	**01001**	
Add 1:	**01010**	**8+2 = 10**

8

8

Numeric Ranges & Important Values

- 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
- Negative 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

9

9

Decimal Values for Different Word Sizes

	W (in bits)			
	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
    - One extra negative number
  - *UMax* = (2 * *TMax*) + 1
- **C Programming**
  - #include <limits.h>
  - Declares constants, e.g.,
    - ULONG_MAX
    - LONG_MAX
    - LONG_MIN
  - Values are platform specific

10

10

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

11

11

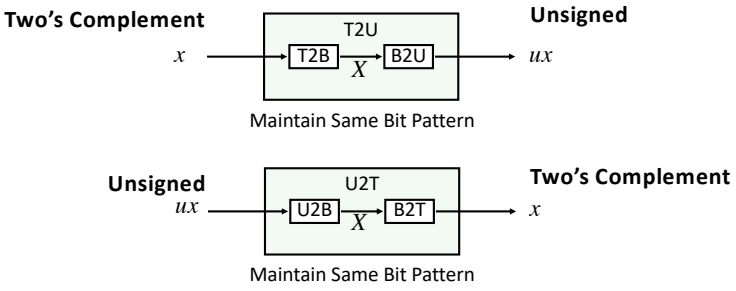
Today: Bits, Bytes, and Integers

- Integers (Ch 2.2)
  - Representation: unsigned and signed
  - Conversion, casting
  - Expanding, truncating
  - Addition, negation, multiplication, shifting (Ch 2.3)

12

12

Mapping Between Signed & Unsigned



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

13

13

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

T2U

U2T

14

14

Signed ↔ Unsigned: Same bits, different interp.

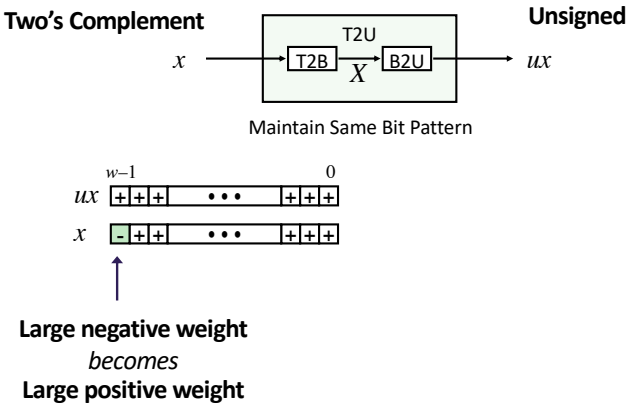
Bits	Signed		Unsigned	Bits
0000	0	=	0	0000
0001	1	=	1	0001
0010	2	=	2	0010
0011	3	=	3	0011
0100	4	=	4	0100
0101	5	=	5	0101
0110	6	=	6	0110
0111	7	=	7	0111
1000	-8	+/- 16	8	1000
1001	-7	+/- 16	9	1001
1010	-6	+/- 16	10	1010
1011	-5	+/- 16	11	1011
1100	-4	+/- 16	12	1100
1101	-3	+/- 16	13	1101
1110	-2	+/- 16	14	1110
1111	-1	+/- 16	15	1111

Why 16?

15

15

Relation between Signed & Unsigned

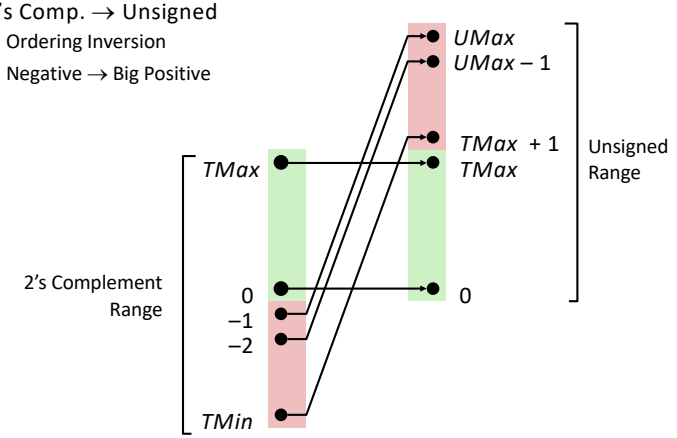


16

16

Conversion Visualized

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



17

17

C Signed vs. Unsigned: A series of surprises?

- Constants
  - By default are considered to be **signed** integers
  - Literals unsigned if have "U" as suffix (kind of like 0x for hex)

```
0U, 4294967259U
```

- Casting
  - Explicit casting between signed & unsigned same as "U2T" and "T2U"
    - The bits don't change, but the interpretation does

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

  - Implicit casting also occurs via assignments and procedure calls

```
tx = ux;                    int fun(unsigned u);
uy = ty;                    uy = fun(tx);
```

18

18

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**

19

19

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`!!

20

20

Practice on Your Own

- Between signed and unsigned

```
unsigned ux;
int x = -10;
ux = x;
printf("x %x ux %x\n", x, ux);
printf("x %d ux %u\n", x, ux);
```

- Between different types (of different sizes)

```
short sx = 0, sy = 0xfedc;
int x = 0x80004000;
sx = x;
x = sy;
printf("sx: %x x: %x\n", sx, x);
printf("sx: %d x: %d\n", sx, x);
```

21

21

Revisiting Right Shift Operations

- Right Shift: `x >> y`
  - Logical shift
    - Fill with 0's on left
  - Arithmetic shift
    - Replicate MSB on left

Argument x	01100010	98_{10}
Log. >> 2	00011000	24_{10}
Arith. >> 2	00011000	24_{10}

Argument x	10100010	-94_{10}
Log. >> 2	00101000	40_{10}
Arith. >> 2	11101000	-24_{10}

Arithmetic right shift maintains negativity

22

22

Today: Bits, Bytes, and Integers

- Integers (Ch 2.2)
  - Representation: unsigned and signed
  - Conversion, casting
  - Expanding, truncating
  - Addition, negation, multiplication, shifting (Ch 2.3)

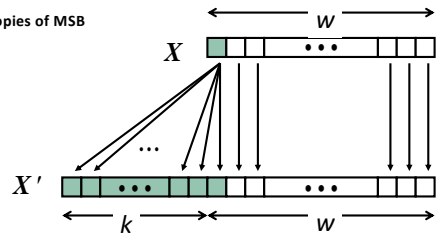
23

23

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:
  - $\boldsymbol{X} = x_{w-1}, \ldots, x_{w-1}, x_{w-1}, x_{w-2}, \ldots, x_0$

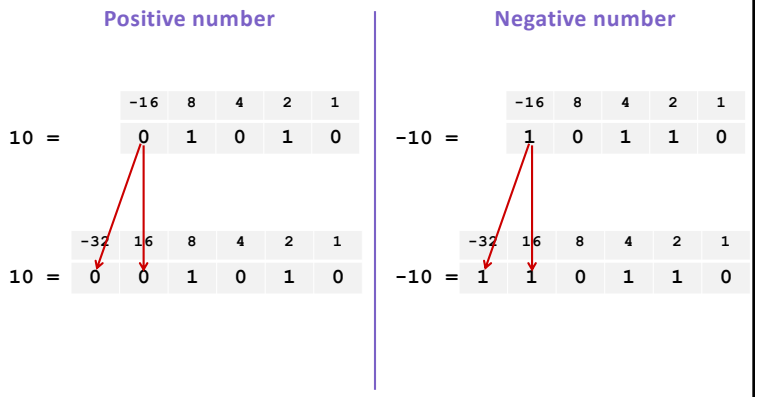
k copies of MSB



24

24

Sign Extension: Simple Example



25

25

Larger 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

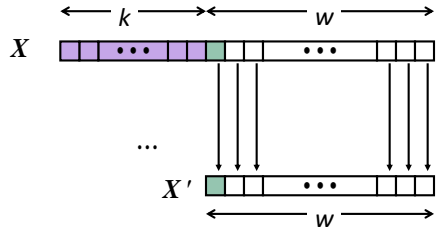
- When converting from smaller to larger integer data type, C automatically performs sign extension

26

26

Truncation

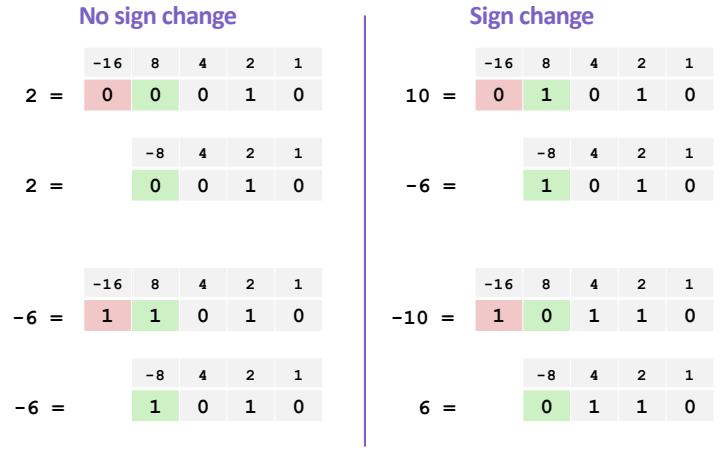
- Task:
  - Given k+*w*-bit signed or unsigned integer *X*
  - Convert it to *w*-bit integer X' with same value for "small enough" X
- Rule:
  - Drop top *k* bits:
  - $X = x_{w-1}, x_{w-2}, \ldots, x_0$



27

27

Truncation Example: 5 bits -> 4 bits



28

28

Summary:
Expanding, Truncating: Basic Rules

- Expanding (e.g., short int to int)
  - Unsigned: zeros added
  - Signed: sign extension
  - Both yield expected result: value is preserved w.r.t. signedness
- Truncating (e.g., unsigned to unsigned short)
  - Unsigned/signed: bits are truncated, result reinterpreted
  - For small numbers yields expected behavior
    - Value is preserved
  - For numbers that are too large to fit into smaller number of bits
    - Value is changed

29