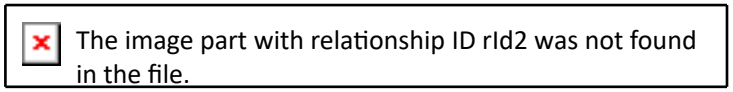


Department of Computer Science

Computer Data Representation (2):

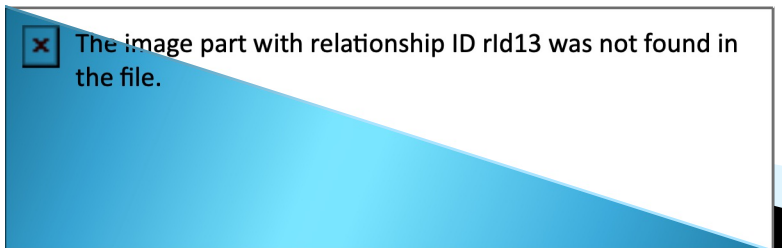
Lecture #3 – part 1

Prof. Soraya Abad-Mota, PhD

Previous lecture

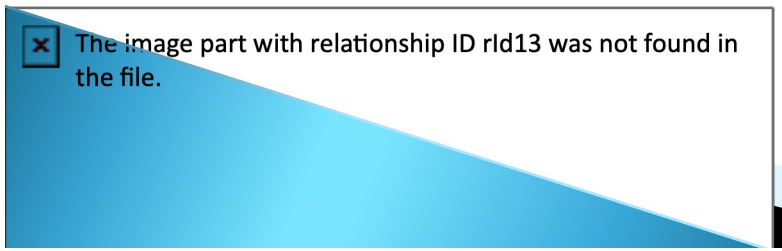
Chapter 2: secs. 2.1, 2.2.1, 2.2.2, 2.2.3

1. Notions of bits and bytes. (1 byte = 8 bits)
2. Integral types
3. Relevant operations in C for this class: bit-level operations (&, |, ^,~), logical operations (&&, ||, !), shift (left and right).



About the index cards

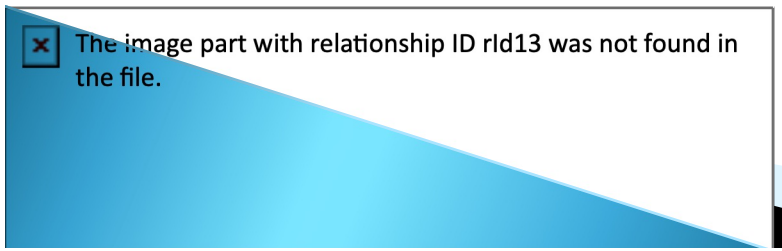
- I saw great questions in them.
- I answered all the questions in the cards.
- Some of the questions apply to material that we cover today, but from the previous material I will highlight a few which are good for reviewing.
- PLEASE acquire a some 3x5 index cards; thin paper, sticky notes, anything other than the 3x5 cards will not count as participation. You only need 27 cards for the rest of the semester.



Good questions from the index cards

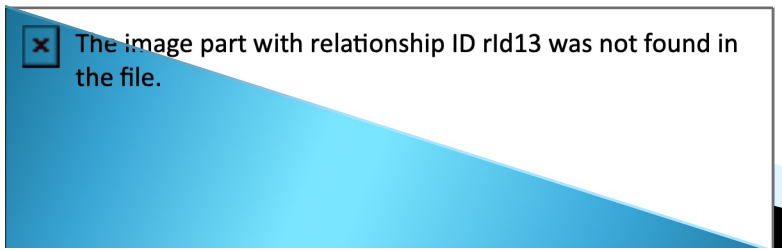
1. *Why is !0x00 0x01 and not 0x11?*
   - `applying ! to 0 is 1`
   - `applying ~ to 0 is FFFFFFFF`
2. *How can you tell if a binary vector is meant to represent a number or a subset?*
3. *How to differentiate between bitwise AND (&) and the operator &* (as give me the address of)*?*

(answers to 2 and 3 are basically the same!)



Today

- Use of shifts and masks
- Encoding integers



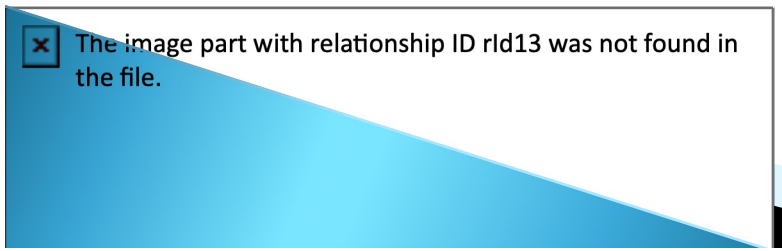
Practice Masking Operations in C

- Problem 2.12 p. 55: Write C expressions for
  A. Least significant byte of x, with all other bits set to 0.

  B. All but the least significant byte of x complemented, with least significant byte left unchanged.

  C. Least significant byte set to all ones, and all other bytes of x left unchanged.

Do for x = 0x789ABC21 and w = 32 (any w>=8)



Extracting and Assembling Integral Types – Shift and Mask

- Shift/AND commonly used to extract subset of bits from integral types

```
int x, y, z;
// Pull bits 16-23 out of X
y = (x >> 16) & 0xff;
```

- OR and shift used to assemble multiple values into a single larger value

```
// Construct z out of those bits and another value;
z = (y & 0xff) | ((0x18 << 8) & 0x3f00);
```

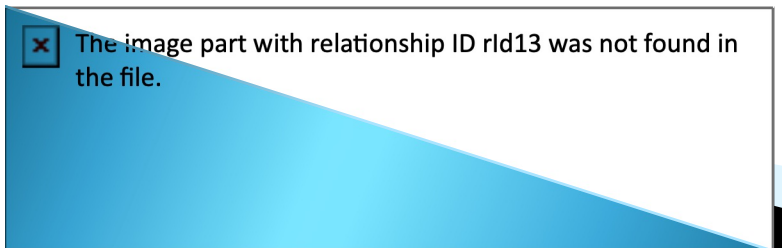
- Used in places where you pack multiple values into a single word (e.g. device drivers)
- C unions, structs, and bitfields are another way to do this (more later)

Good problems to do

- Problems: (no answer in the book)
  - 2.59,
  - 2.61,
  - 2.68

With coding rules on pp. 128-129

Homework #1 will be posted tonight, includes some of these and some others.



Learning Objectives for this portion

After this session + practice you should be able to:

1. Describe how are integral types represented in the computer: unsigned and signed.
2. Define and perform conversion, casting, expanding and truncating operations on integers.
3. Define and perform integer arithmetic operations + Explain and analyze the errors derived from these operations.

The image part with relationship ID rId13 was not found in the file.

Encoding Integers (fig. 2.8 terminology)

Covered in CS241L

Unsigned

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

Two's Complement

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

Sign Bit

```
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, most significant bit indicates sign
    - 0 for nonnegative
    - 1 for negative

Two's-complement Encoding Example

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

Pwr	Weight	15213		-15213	
0	1	1	1	1	1
1	2	0	0	1	2
2	4	1	4	0	0
3	8	1	8	0	0
4	16	0	0	1	16
5	32	1	32	0	0
6	64	1	64	0	0
7	128	0	0	1	128
8	256	1	256	0	0
9	512	1	512	0	0
10	1024	0	0	1	1024
11	2048	1	2048	0	0
12	4096	1	4096	0	0
13	8192	1	8192	0	0
14	16384	0	0	1	16384
15	-32768	0	0	1	-32768
	Sum		**15213**		**-15213**

Two's-complement Encoding Example

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

What we did was:

1. Take 15213 in binary.
2. Complement it
3. Add 1

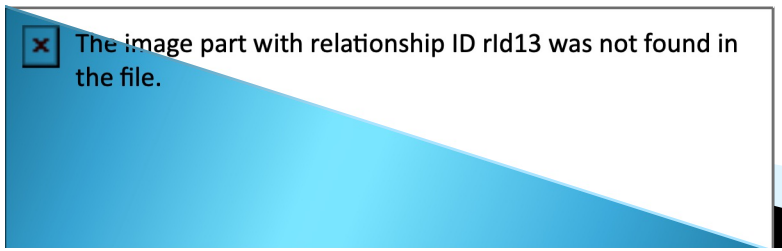
```
00111011 01101101
11000100 10010010
                 1
-------------------
11000100 10010011
```

Apply the B2T formula to verify that this last sequence is indeed -15213

Example

- How to represent 13 as unsigned integer.

- How to represent -13 in the two's complement notation.



Example

- How to represent 13_{10} as unsigned integer.
  - $00001101 = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^0 = 13_{10}$

- How to represent -13 in the two's complement notation.
  A. is it 10001101?
  B. is it 11110011?

Procedure to compute the decimal value from the binary form.

Example

- From 13_{10} to find -13
  1. 13 is 0000 1101
  2. Complement gives 1111 0010
  3. Add 1 1
  4. 1111 0011
  5. 1111 0011 is -13

Verify:

$-1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^1 + 1 \times 2^0 =$

-128 + 64 + 32 + 16 + 2 + 1 =

-128 + 115 = -13

Practice

- How to represent 20_{10} as unsigned integer.

- How to represent -20 in the two's complement notation. (Applying the B2T(x) formula.)

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

- good to visualize all the values in each of the encodings

Numeric Ranges

- Unsigned Values (2^w *values > 0)*
  - *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

w = 4 ➔ TMax =
TMin =

- Other Values
  - Minus 1 (-1)
    111…1

Values for *w* = 16 (word size)

	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

Go to Lecture #3 – part 2

The image part with relationship ID rId13 was not found in the file.