Lecture 5:
Integers: Sign and Size

CSE 29: Systems Programming and Software Tools

Aaron Schulman (Shalev)

Today's Lecture

- Review Integer Arithmetic
  - UTF-8 Code Point Analysis
- How signed integers work in computers

Demo: UTF-8 Code Point Analysis

- **Big Ideas:**
  - We need to mask (and shift) bits to do code point analysis
  - We need large size integers to store large code points (int32_t)
  - We may run into issues with signed integers resulting in neg. nums!

Negative Integers in Computers

- Integers are stored as binary numbers; binary has no sign (+/-)!
  - *e.g.*, **1 0 1 0** = $1x2^3 + 0x2^2 + 1x2^1 + 0x2^0 = 10$
- Must encode somehow in the binary digits that a number is neg
- We need a way that is efficient (doesn't waste bits) and simple to implement in the hardware of a CPU

Sign Magnitude – Simple but inefficient

- The simplest way to do this would be to reserve a bit for the sign

S	**0**	**0**	
0	0	1	**= 1**
1	0	1	**= -1**

- *Inefficient* both in terms of storage and hardware:
  - Two ways to represent zero (**0** and **-0**)
  - Math hardware in CPU needs to handle positive and negative differently
    - » Adding a positive number to a negative number needs to read sign bit
    - » Adding positive to positive does not

Two's Complement

- What if we make the MSB equal to -2^{MSB}?
  - In other words, if the MSB is set, the number becomes negative with that magnitude

MSB							LSB
-128	64	32	16	8	4	2	1
-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

- Minimum will have higher magnitude than the maximum (by -1)
  - Min (-128)
  - Max (127 = 64 + 32 +16 +8 + 4 + 2 + 1)

- Only one zero, and hardware is the same as an unsigned int!

Two's Complement compared to Unisgned

Two's Complement has a lower max compared to unsigned (power of 2)

MSB							LSB
-128	64	32	16	8	4	2	1
-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Max = 127

128	64	32	16	8	4	2	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Max = 255

8 bits size (1 byte)

Examples of Two's Complement

- Work with your neighbor to figure out what the 2's complement value is

1	0	0	1
-2^3	2^2	2^1	2^0

= -8 + 1 = **-7**

1	1	1	1	1
-2^4	2^3	2^2	2^1	2^0

= -16 + 8 + 4 + 2 + 1 = **-1**

Copying small int into larger size int

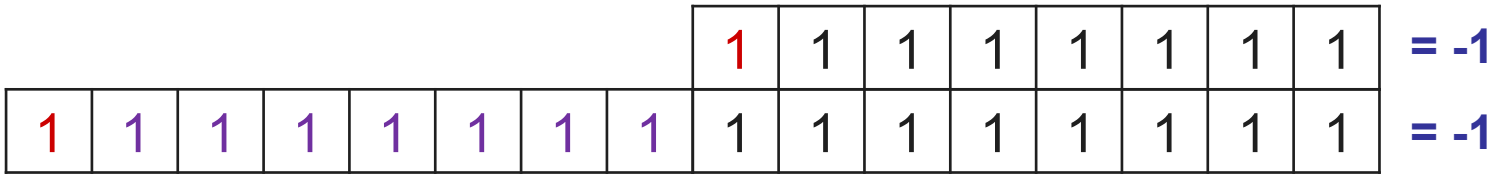
- Need to handle copying a smaller 2's complement int into a larger one

```
short int a_s = -1;
int a = a_s;
printf("a=%d a_s=%d\n", a_s, a);
a=-1 a_s=-1
```

- Just copying the bits into the LSBs of the larger size won't work:

								1	1	1	1	1	1	1	1	= -1
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	= 255 ☹

Sign extension in 2's complement



[When copying, extend the sign bit]

Data types in typically used in C

- Integer data types
  - **char** = ‘A’ (1 byte – max 127) - *Signed*
  - **int/int32_t** = 42 (4 bytes – max 2 billion) - *Signed*
  - **unsigned char** = (1 byte – max 255) - *Unsigned*
  - **unsigned int/uint32_t** = (4 bytes - max 4 billion)