Lecture 2: Representing Integers

CS 105 January 27, 2020

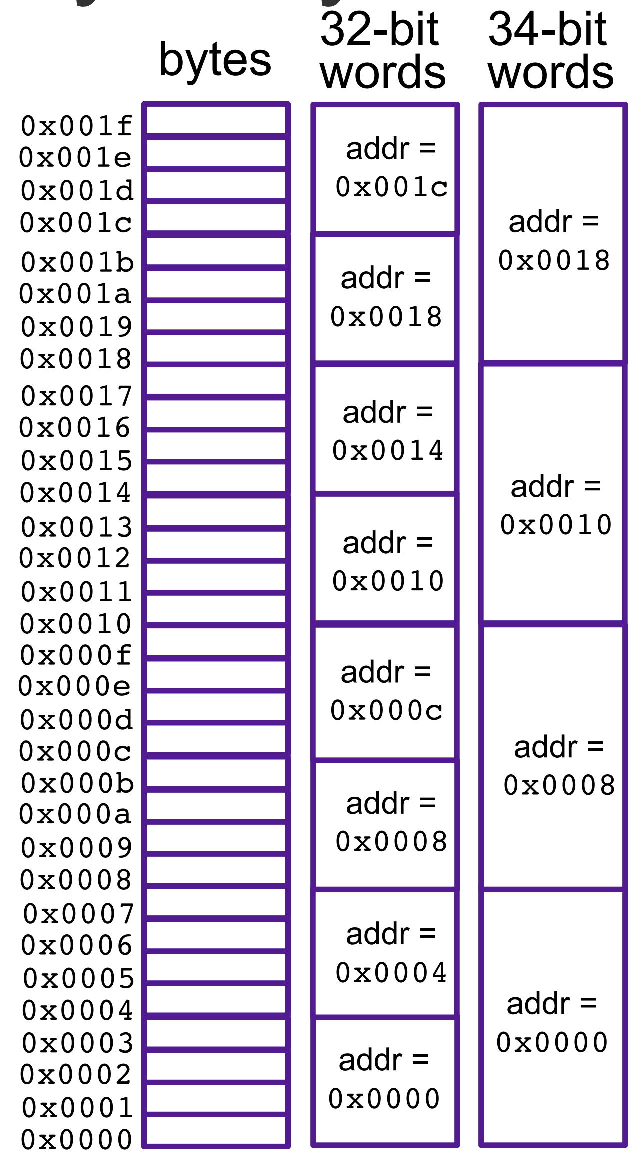
Abstraction

The C Language

- Syntax like Java: declarations, **`if, while, return`**
- Data and execution model are “closer to the machine”
  - More power and flexibility
  - More ways to make mistakes
  - Sometimes confusing relationships
  - Pointers!!

Memory: A (very large) array of bytes

- An index into the array is an *address*, *location*, or *pointer*
  - Often expressed in hexadecimal
- We speak of the *value* in memory at an address
  - The value may be a single byte …
  - … or a multi-byte quantity starting at that address
- Larger **words** (32- or 64-bit) are stored in contiguous bytes
  - The address of a word is the address of its first byte
  - Successive addresses differ by word size



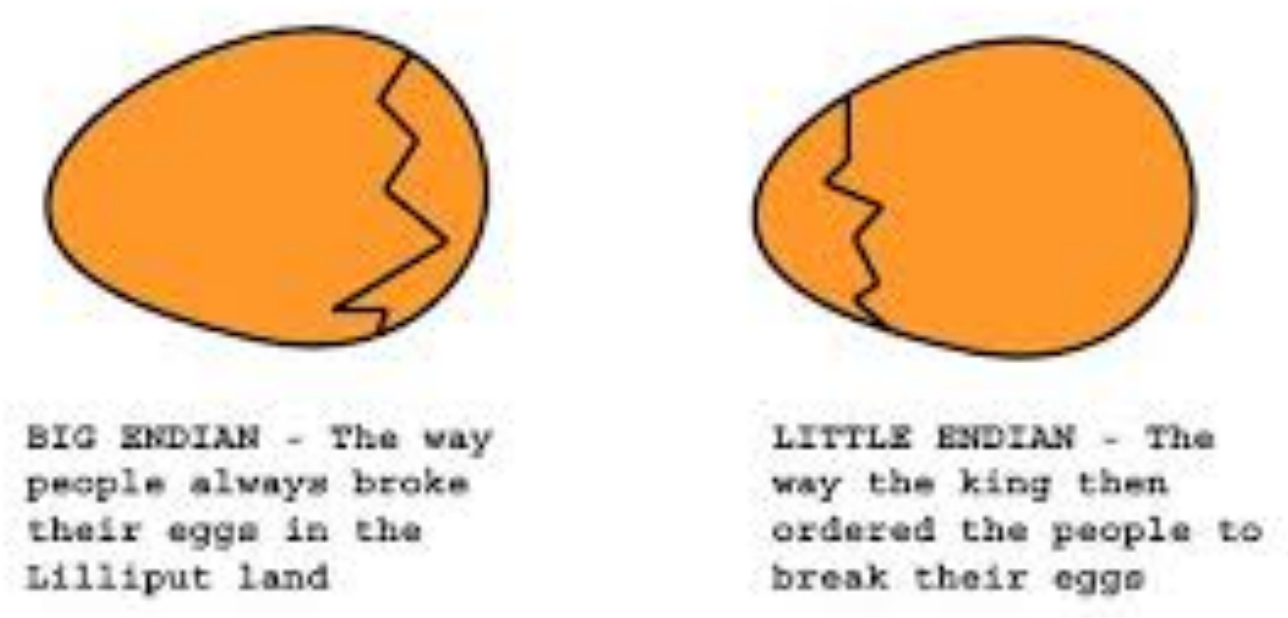
Representing Unsigned Integers

- Think of bits as the binary representation

$$\text{UnsignedValue}(x) = \sum_{j=0}^{w-1} x_j \cdot 2^j$$

- If you have w bits, what is the range?

Endianness


BIG ENDIAN - The way people always broke their eggs in the Lilliput land
LITTLE ENDIAN - The way the king then ordered the people to break their eggs

Unsigned Integers in C

C Data Type	Size (bytes)
unsigned short	2
unsigned int	4
unsigned long	8

- What about casting?
  - Casting from shorter to longer types preserves the value
  - Casting from longer to shorter types truncates the bits
- What about negative numbers?

Representing Signed Integers

- Option 1: sign-magnitude
  - One bit for sign; interpret rest as magnitude
- Option 2: excess-K
  - Choose a positive K in the middle of the unsigned range
  - SignedValue(w) = UnsignedValue(w) – K
- Option 3: one’s complement
  - Flip every bit to get the negation

Representing Signed Integers

- Option 4: two's complement
  - Most commonly used
  - Like unsigned, except the high-order contribution is *negative*

$$\text{SignedValue}(x) = -x_{w-1} \cdot 2^{w-1} + \sum_{j=0}^{w-2} x_j \cdot 2^j$$

- Exercise: Assume C short (2 bytes)
  - What is the binary representation for 47?
  - What is the hex representation for 47?
  - What is the binary representation for -47?
  - What is the hex representation for -47

Example: Three-bit integers

unsigned		signed
111	7	
110	6	
101	5	
100	4	
011	3	011
010	2	010
001	1	001
000	0	000
	-1	111
	-2	110
	-3	101
	-4	100

- The high-order bit is the *sign bit.*
- The largest unsigned value is $11\ldots1$, UMax.
- The signed value for -1 is always $11\ldots1$.
- Signed values range between `TMin` and TMax.

This representation of signed values is called *two's complement.*

Two's Complement Signed Integers

- "Signed" does not mean "negative"
- High order bit is the *sign bit*
  - To negate, complement all the bits and add 1
- Arithmetic is the same as unsigned—same circuitry
- Error conditions and comparisons are different

Important Signed Numbers

	8	16	32	64
TMax	0x7F	0x7FFF	0x7FFFFFFF	0x7FFFFFFFFFFFFFFF
TMin	0x80	0x8000	0x80000000	0x8000000000000000
0	0x00	0x0000	0x00000000	0x0000000000000000
-1	0xFF	0xFFFF	0xFFFFFFFF	0xFFFFFFFFFFFFFFFF

Unsigned and Signed Integers

- Use w-bit words; w can be 8, 16, 32, or 64
- The bit sequence $b_{w-1} \ldots b_1 b_0$ represents an integer

	unsigned	signed
value	$\sum_{i=0}^{w-1} b_i 2^i$	$-b_{w-1}2^{w-1} + \sum_{i=0}^{w-2} b_i 2^i$
smallest	0	-2^{w-1}
largest	$2^w - 1$	$2^{w-1} - 1$

Casting between Numeric Types

- Casting from shorter to longer types preserves the value
- Casting from longer to shorter types truncates the bits

- Casting between signed/unsigned types preserves the bits (it just changes the interpretation)

Exercise: Numeric Data Representations

- Assume you have a machine with 6-bit integers/3-bit shorts
- Assume variables: `int x = -17; short sy = -3;`
- Complete the following table

Expression	Decimal	Binary
	-6	
		101010
(unsigned int) x		
(int) sy		
TMax		
TMin		