Computer Science 61
Lecture Notes for Lecture 3

Julian Debus, Eric Lu, and Jesse Chen

September 11, 2012

1 Introduction

Announcements: Thursday 9/13/2011: Room G115: Section on using Git and the CS 50 Appliance

2 Computer Arithmetic

C data type	largest representable value	bytes
unsigned char	$2^8 = 255$	1
unsigned short	$2^16 - 1 = 65535$	2
unsigned [int]	$2^32 - 1 = 42949672$	4
unsigned long	$2^32 - 1 = 42949672$	4
unsigned long long	$2^{64} - 1 = \ldots$	8

Unsigned arithmetic is the same as "normal" arithmetic mod 2^w.
(w=#bits in a value)

2.1 Addition

Examples of addition in binary

```
 011  (3)
+100  (4)
=111  (7)

  100  (4)
+ 110  (6)
=1010  (10)
```

2.2 Subtraction

We are still only using unsigned arithmetic and thus can't represent negative values. Nevertheless, subtraction should be possible (we can subtract 3 from 4, afterall, using only positive numbers).

$$a - b = a + (-b)$$

What is (-b)? Let's say $(-b) = c$ so that $b + c = 0$.

$$\begin{aligned} b + c &== 0 \mod 2^w \\ b + c &== 2^w \mod 2^w \\ c &== 2^w - b \end{aligned}$$

We can subtract a number b from a number a, $0 \leq b \leq 2^w$ by doing:

$$a + (2^w - b)$$

Example:
For $w = 1$: $1 - 1 = 1 + 2^1 - 1 = 1 + 2 - 1 = 1 + 1 = 0$
For $w = 3$: $7 - 3 = 7 + (2^3 - 3) = 7 + 5$

```
 111  (7)
+101  (5)
=100  (4)
```

3 C Operators

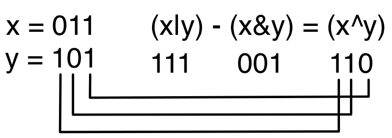
Operator	Explanation
Arithmetic Operators	
$+$	Addition
$-$	Subtraction
$/$	Division
$*$	Multiplication
$\%$	Modulus
Bitwise Operators	
$<<$	Bitwise shift left
$>>$	Bitwise shift right
$\&$	Bitwise AND
$\vert$	Bitwise OR
$\hat{}$	Bitwise XOR
Logical Operators	
$\&\&$	Logical AND
$\vert\vert$	Logical OR
Comparison Operators	
$==$	Equal to
$!=$	Does not equal
$<$	Less than
$<=$	Less than/equal to
$>$	Greater than
$>=$	Greater than/equal to

$x|y >= max(x,y) >= x, y >= min(x,y) >= x\&y$

Let:
$x = 3 \rightarrow 011$
$y = 5 \rightarrow 101$

x|y $= 7 \rightarrow 111$. In every bit position in x|y, the value is greater or equal to the value of the corresponding position in both x and y.
x&y $= 1 \rightarrow 001$: In every bit position in $x\&y$, the value is lower or equal to the value of the corresponding position in both x and y.

This reveals the identity:



Another interesting identity:
(x & (x-1)) == 0 **iff** $x = 0$ or $x = 2^k$.
This ensures that upon subtracting 1, no two bits are 1 and therefore evaluate to 0.

Example:

$x = 2^{10}$
100000000
&011111111
=000000000

$x = 0$
000000000
&111111111
=000000000

If $x \neq 2^k \rightarrow x = 2^k + y$ where $0 < y < 2^k$, then $x - 1$ is always greater than or equal to 2^k. As a result, we know that at least the k bits will both be 1. This means that when we do the bitwise and of x and x-1, it cannot be 0.

~**x** $= 4 \rightarrow 100$: Every bit in x is flipped.
$x|{\sim}x = -1$, since ${\sim}x + 1 = -x$
$x + {\sim}x == x|{\sim}x$
This is due to the fact that there can't be two 1s in the same position when adding x and ~x and therefore no carries (which would make a difference).

What's a good representation for a computer for a set of letters?

1. Available letters: {'a'-'z'}
2. With the following operations
   a) Lookup: Is a letter in the set?
   b) Set union
   c) Set intersection
   d) Set difference

$\rightarrow$ An array of bits where 1 means "letter in set" and 0 means "letter not in set".
Empty Set: 0
We can represent {z} where $z \in$ {'a'-'z'} as:

```
1 << (z - 97) == 1 << (z - 'a')
```

x $<<$ **1** $= 6 \rightarrow 110$: Left shift x by one bit position.
More Examples:
$1 << 1 = 0010$
$1 << 2 = 0100$

z == 'a': 1«('a' - 'a') = 1«0 $\rightarrow$ 1
z == 'b': 1«('b' - 'a') = 1«1 $\rightarrow$ 2
z == 'c': 1«('c' - 'a') = 1«2 $\rightarrow$ 4

a) Lookup: Is a letter in the set?
A letter z is in the set S if:
$\{z\} \cap S \neq \emptyset \Leftrightarrow$ (1«(z-97))&S!=0

b) Set union
Union of set S_1 and S_2 represented by Integers s_1 and s_2:
$S_1 \cup S_2 \Leftrightarrow s_1$ & s_2

c) Set intersection
Intersection of set S_1 and S_2 represented by Integers s_1 and s_2:
$S_1 \cap S_2 \Leftrightarrow s_1$ & s_2

d) Set difference
Difference of set S_1 and S_2 represented by Integers s_1 and s_2:
$S_1 - S_2 \Leftrightarrow s_1$ - (s_1 & s_2)
or $S_1 - S_2 \Leftrightarrow s_1$ - (($s_1|s_2$) - s_2)

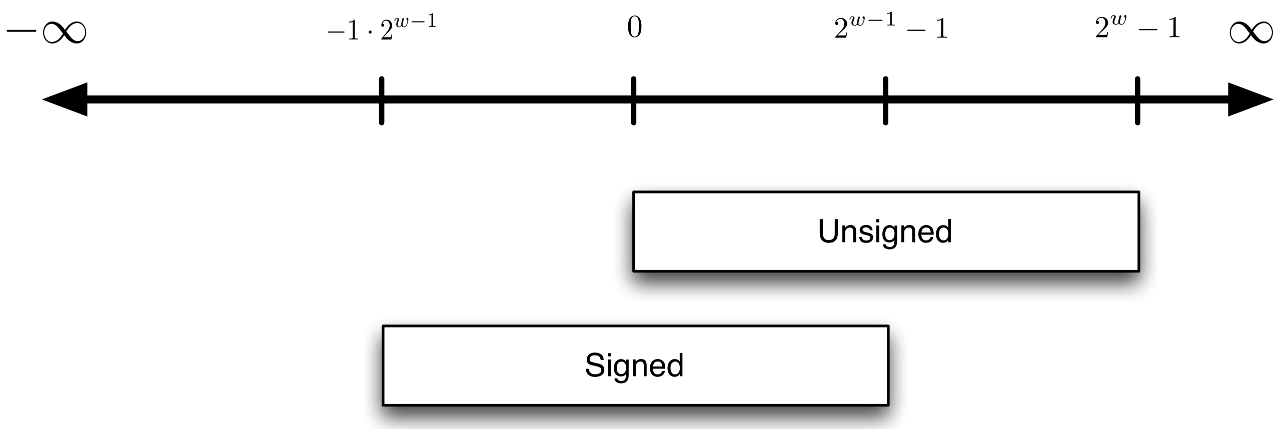
e) Bonus: Check for Singleton
x!=0 && (x&(x-1))==0

4 Signed arithmetic

Until now, we have only used unsigned arithmetic.
What if implementation of signed + and - was the same as unsigned?
→ The bit pattern for signed -1 must be the same as the bit pattern for unsigned -1.
Bit pattern for x (in unsigned representation): $2^w + x$. This is called **Two's Complement**.

1000	-8	0000	0
1001	-7	0001	1
1010	-6	0010	2
1011	-5	0011	3
1100	-4	0100	4
1101	-3	0101	5
1110	-2	0110	6
1111	-1	0111	7

There is no way to represent 2^{w-1}!



As can be seen in the table above, the first bit denotes whether the value is negative or not. Therefore, bit w is called the **sign bit**.