Bits, Bytes, and Integers – Part 2

15-213: Introduction to Computer Systems
3rd Lecture, Sept. 6, 2016

Today's Instructor:

Randy Bryant

First Assignment: Data Lab

- **Due: Thursday, Sept. 15th 2016, 11:59:00 pm**
- **Last Possible Time to Turn in: Sunday, Sept. 18th, 11:59PM**
- **Read the instructions carefully: writeup, bits.c, tests.c**
- **Seek help**
  - Office hours already running
  - Recitation, Monday Sept. 12
- **Based on Lecture 2, 3 , and 4 (CS:APP Chapter 2)**
- **After today's lecture you know everything for the integer problems, float problems covered on Tuesday**

Linux Boot Camp

- **Tonight, Tuesday, Sept. 6**
  - 7:30-9:00 pm
  - Rashid Auditorium
- **Bring your laptop**
- **Open to undergrads and masters students**

Summary From Last Lecture

- Representing information as bits
- Bit-level manipulations
- Integers
  - Representation: unsigned and signed
  - Conversion, casting
  - Expanding, truncating
  - Addition, negation, multiplication, shifting
- Representations in memory, pointers, strings
- Summary

Encoding Integers

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

Two's Complement Examples (w = 5)

```
          -16   8    4    2    1
10 =       0    1    0    1    0      8+2 = 10

          -16   8    4    2    1
-10 =      1    0    1    1    0      -16+4+2 = -10
```

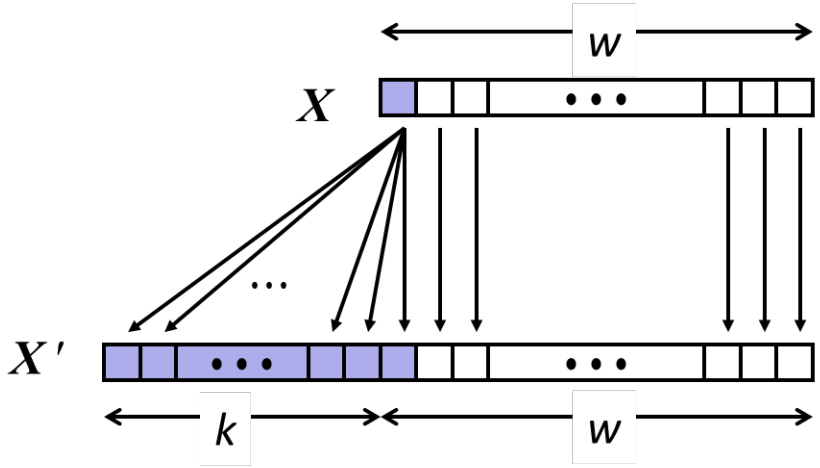
Unsigned & Signed Numeric Values

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

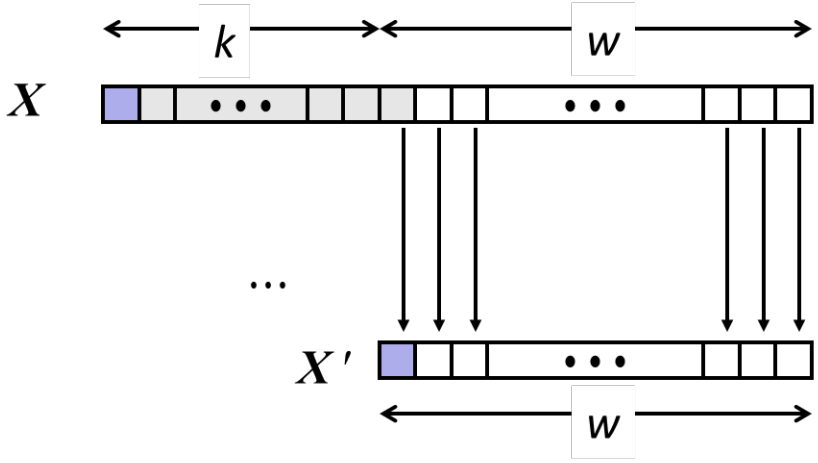
- **Equivalence**
  - Same encodings for nonnegative values
- **Uniqueness**
  - Every bit pattern represents unique integer value
  - Each representable integer has unique bit encoding
- **Expression containing signed and unsigned int:**
  `int` is cast to `unsigned`

Sign Extension and Truncation

- **Sign Extension**



- **Truncation**



Today: Bits, Bytes, and Integers

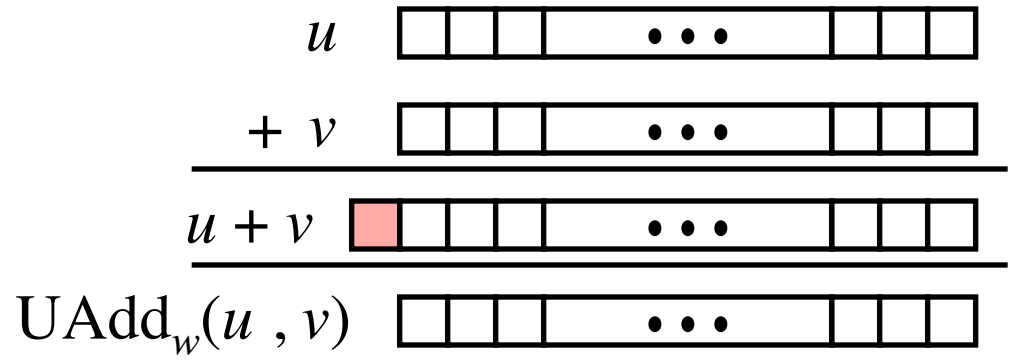
- Representing information as bits
- Bit-level manipulations
- **Integers**
  - Representation: unsigned and signed
  - Conversion, casting
  - Expanding, truncating
  - **Addition, negation, multiplication, shifting**
- Representations in memory, pointers, strings
- Summary

Unsigned Addition

Operands: *w* bits

True Sum: *w*+1 bits

Discard Carry: *w* bits



- **Standard Addition Function**
  - Ignores carry output
- **Implements Modular Arithmetic**

  s = $\text{UAdd}_w(u , v)$ = $u + v \mod 2^w$

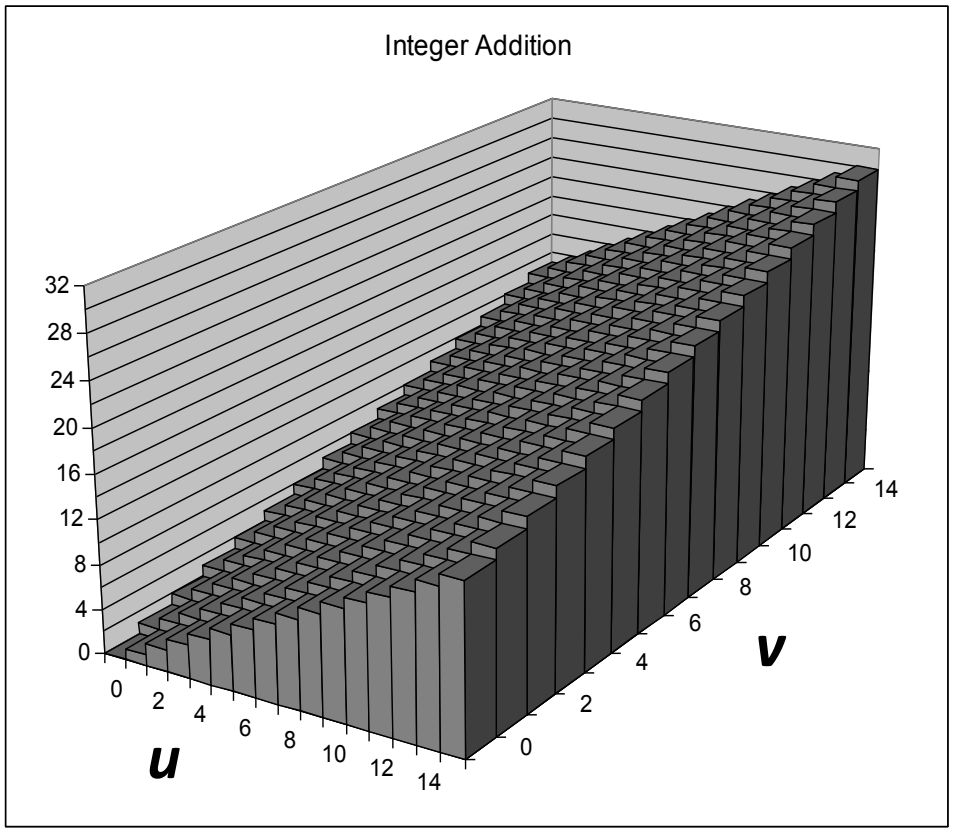
```
unsigned char     1110 1001      E9      223
              +   1101 0101    + D5    + 213
              ------------    ------  ------
               1 1011 1110     1BE     446
              ------------    ------  ------
                  1011 1110      BE     190
```

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Visualizing (Mathematical) Integer Addition

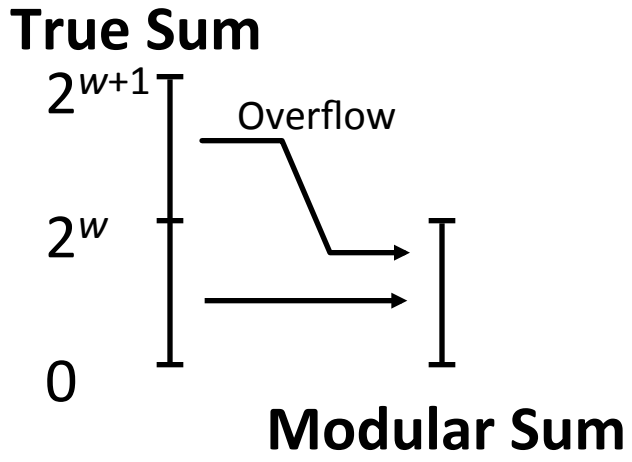
- **Integer Addition**
  - 4-bit integers *u*, *v*
  - Compute true sum $\text{Add}_4(u , v)$
  - Values increase linearly with *u* and *v*
  - Forms planar surface

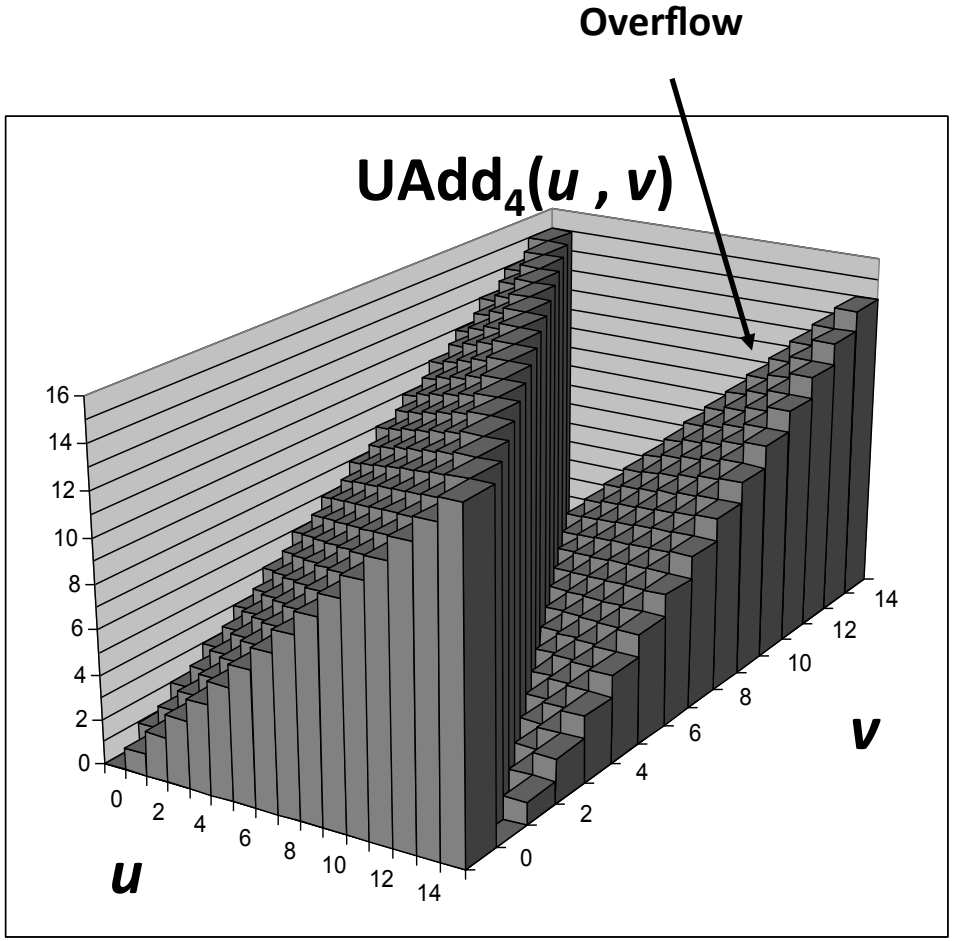
$\text{Add}_4(u , v)$



Visualizing Unsigned Addition

- **Wraps Around**
  - If true sum ≥ 2^w
  - At most once



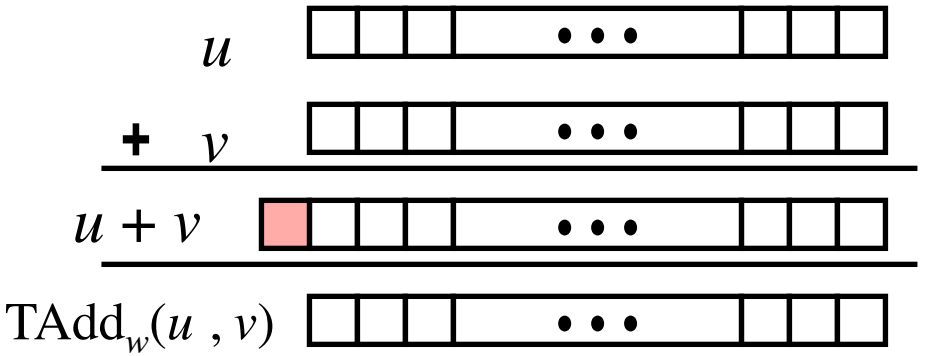


Two's Complement Addition

Operands: *w* bits

True Sum: *w*+1 bits

Discard Carry: *w* bits



- **TAdd and UAdd have Identical Bit-Level Behavior**
  - Signed vs. unsigned addition in C:

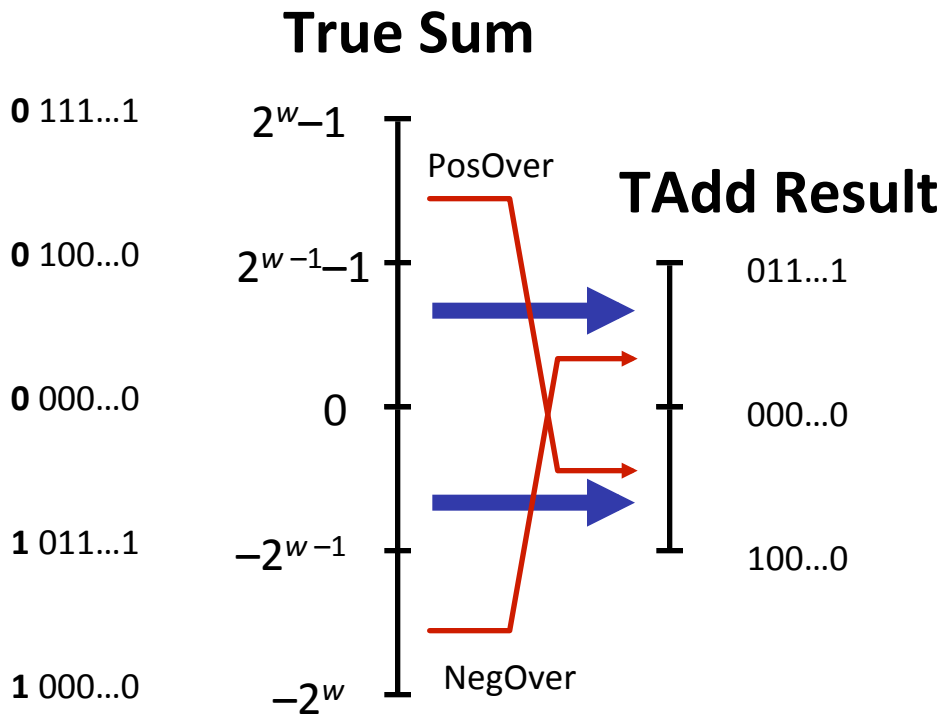
```
int s, t, u, v;
s = (int) ((unsigned) u + (unsigned) v);
t = u + v
```

  - Will give `s == t`

	1110 1001	E9	-23
+	1101 0101	+ D5	+ -43
1	1011 1110	1BE	446
	1011 1110	BE	-66

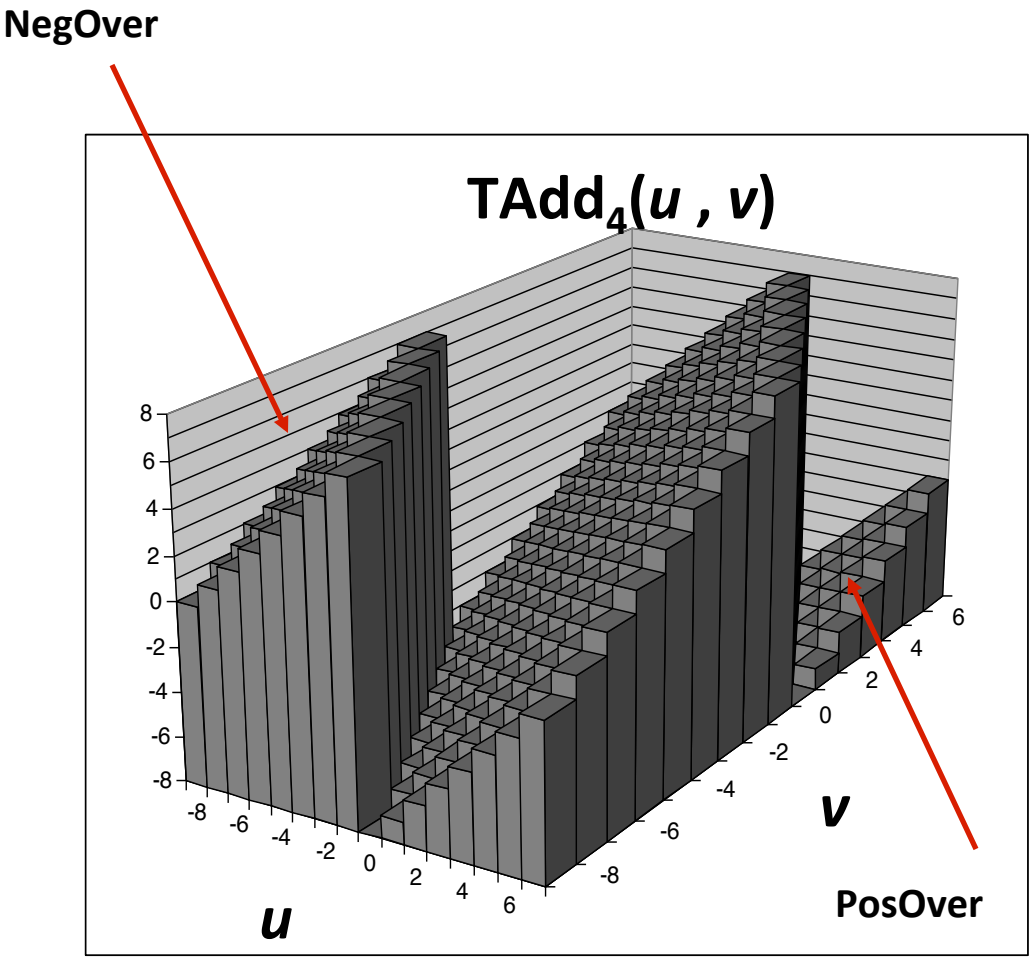
TAdd Overflow

- **Functionality**
  - True sum requires *w*+1 bits
  - Drop off MSB
  - Treat remaining bits as 2's comp. integer



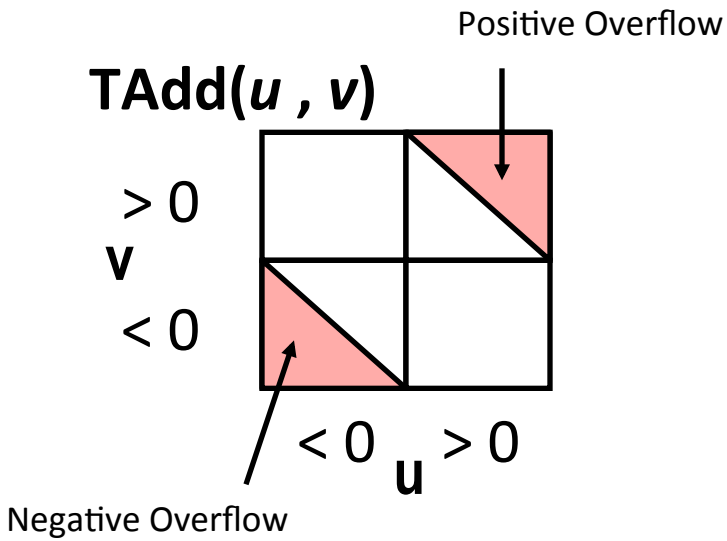
Visualizing 2's Complement Addition

- **Values**
  - 4-bit two's comp.
  - Range from -8 to +7
- **Wraps Around**
  - If sum ≥ 2^{w-1}
    - Becomes negative
    - At most once
  - If sum < -2^{w-1}
    - Becomes positive
    - At most once



Characterizing TAdd

- **Functionality**
  - True sum requires *w*+1 bits
  - Drop off MSB
  - Treat remaining bits as 2's comp. integer



$$TAdd_w(u,v) = \begin{cases} u+v+2^w & u+v < TMin_w \quad \textbf{(NegOver)} \\ u+v & TMin_w \le u+v \le TMax_w \\ u+v-2^w & TMax_w < u+v \quad \textbf{(PosOver)} \end{cases}$$