Integer Arithmetic

CS 347

Lecture 03

January 20, 1998

Topics

- **Numeric Encodings**
  - Unsigned & Two’s complement
- **Programming Implications**
  - C promotion rules
  - Consequences of overflow
- **Basic operations**
  - Addition, negation, multiplication

Notation

Word Size

- *w*
  - 32 for most machines
  - 64 for Alpha and next generation of high end machines
  - 8 or 16 for many signal processing applications

Integers

- **Lower case**
- **E.g.,** x**,** y**,** z

Bit Vectors

- **Upper Case**
- **E.g.,** X**,** Y**,** Z
- **Write individual bits as integers with value 0 or 1**
- **E.g.,** $X = x_{w-1}, x_{w-2}, \ldots x_0$
  - Big-Endian

Encodings

Unsigned

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

Ranges

- *UMin* = 0

  000…0

  `0x00…0`

- *UMax* = $2^w - 1$

  111…1

  `0xFF…F`

Two's Complement

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

Sign Bit

Ranges

- *TMin* = -2^{w-1}

  100…0

  `0x80…0`

- *TMax* = $2^{w-1} - 1$

  011…1

  `0x7F…F`

Other Values

- Zero

  000…0

- Minus 1

  111…1

  `0xFF…F`

Real-Life Values

W = 32

• UMax	+4,294,967,295
• TMax	+ 2,147,483,647
• TMin	– 2,147,483,648

W = 64

• UMax	+18,446,744,073,709,551,615
• TMax	+9,223,372,036,854,775,807
• TMin	–9,223,372,036,854,775,808

C Programming

- `#include <limits.h>`
- **Declares constants:**
  - `ULONG_MAX`
  - `LONG_MAX`
  - `LONG_MIN`

Encoding Example

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

Observe

- **Same encodings for nonnegative values**
- **Assymetric range**

  TMax ≠ –*Tmin*

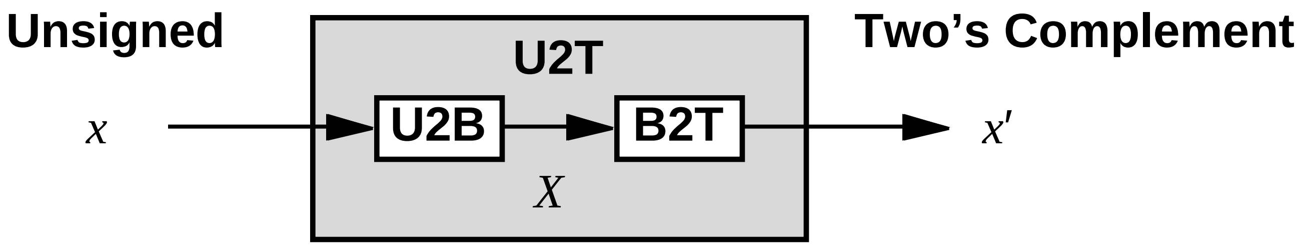
Each Encoding is*Bijection*

- **Every bit pattern represents unique integer value**
- **Each representable integer has unique bit encoding**

==> Can Invert Mappings

- **U2B(*x*) = B2U^{-1}(*x*)**
  - Bit pattern for unsigned integer
- **T2B(*x*) = B2T^{-1}(*x*)**
  - Bit pattern for two's comp integer

Relation Between Unsigned & Two's Comp.



Maintain Same Bit Pattern

$$
\begin{aligned}
B2T(X) &= \sum_{i=0}^{w-2} x_i \; 2^i && - x_{w-1} 2^{w-1} \\
&= \sum_{i=0}^{w-2} x_i \; 2^i && + x_{w-1} 2^{w-1} - x_{w-1} 2^{w} \\
&= \sum_{i=0}^{w-1} x_i \; 2^i && - x_{w-1} 2^{w} \\
&= B2U(X) && - x_{w-1} 2^{w}
\end{aligned}
$$

$$
x' = \begin{cases} x & x < 2^{w-1} \\ x - 2^w & x \geq 2^{w-1} \end{cases}
$$

From Unsigned to Two's Complement

- **U2T(*x*)**

  = B2T(U2B(*x*))

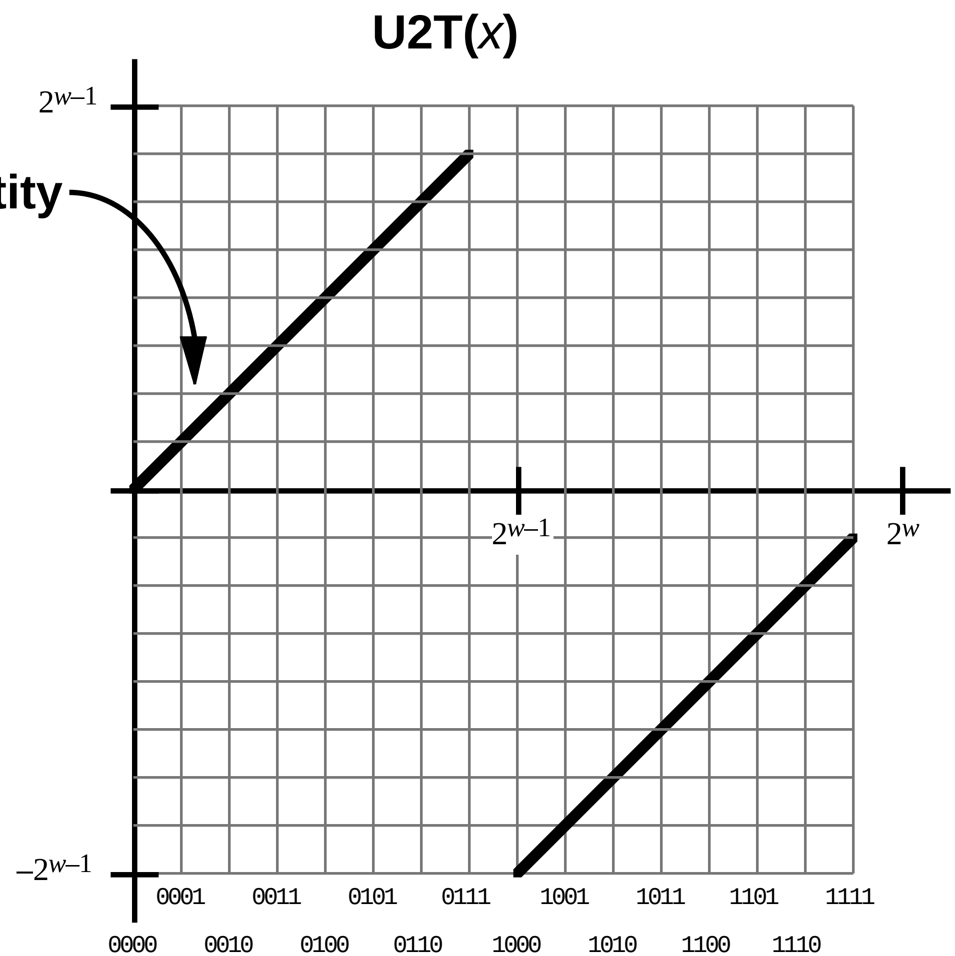
  = $x - x_{w-1} 2^w$

- **What you get in C:**

```
int u2t(unsigned x)
{
  return (int) x;
}
```

X	**B2U(*X*)**	**B2T(*X*)**
1010	10	–6

– 16



From Two's Complement to Unsigned

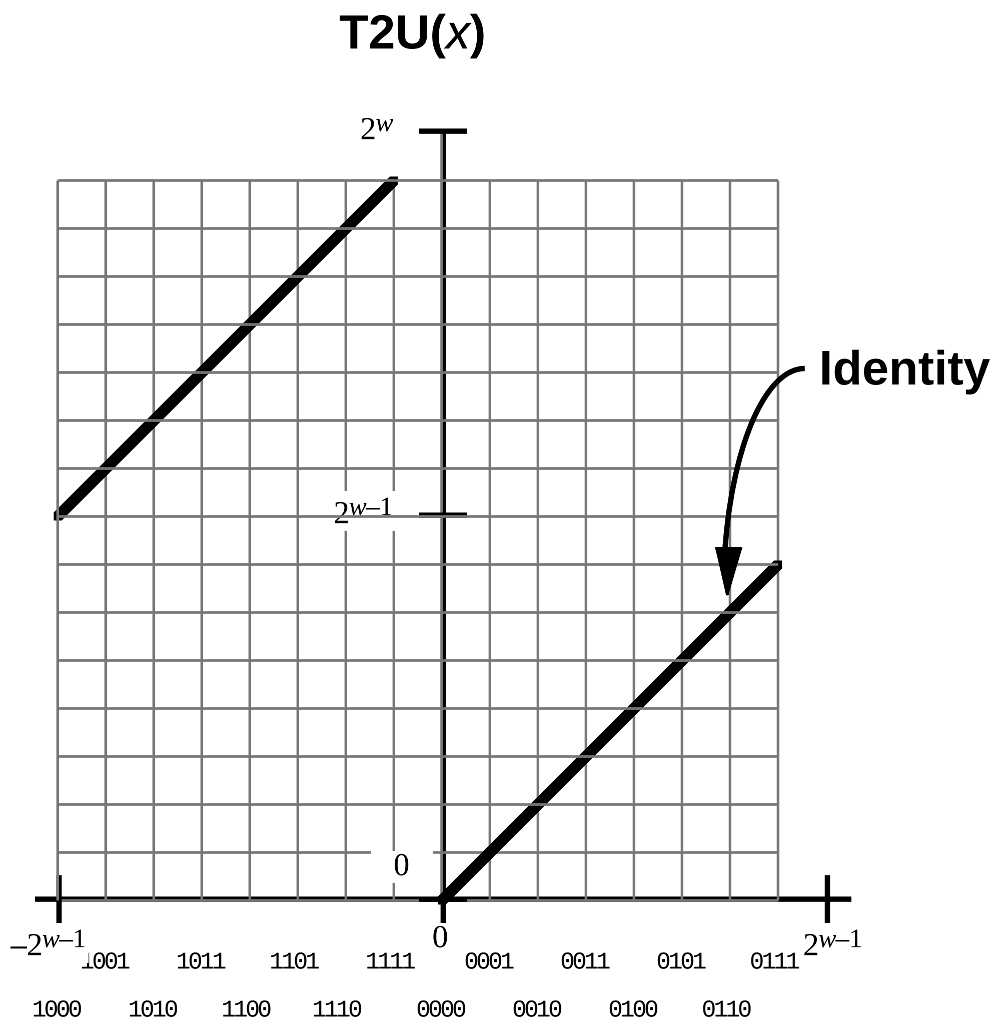
- **T2U(x)**
  = B2U(T2B(x))
  = $x + x_{w-1} 2^w$

- **What you get in C:**

```
int t2u(int x)
{
  return (unsigned) x;
}
```

X	**B2U(X)**	**B2T(X)**
1010	10	–6

+ 16



Signed vs. Unsigned in C

Constants

- By default are considered to be signed integers
- Unsigned if have "U" as suffix

```
0U, 4294967259U
```

Casting

- Explicit casting between signed & unsigned same as U2T and T2U

```
int tx, ty;
unsigned ux, uy;
tx = (int) ux;
uy = (unsigned) ty;
```

- Implicit casting also occurs via assignments and procedure calls

```
tx = ux;
uy = ty;
```

Casting Surprises

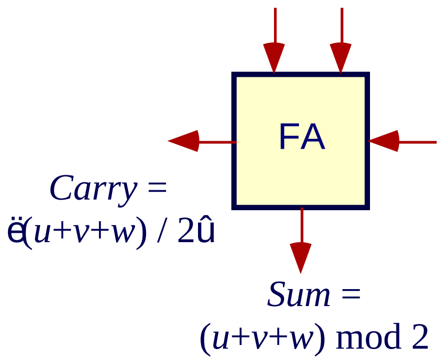
Expression Evaluation

- If mix unsigned & signed in single expression, signed values implicitly cast to unsigned
- Including comparison operations <, >, ==, <=, >=

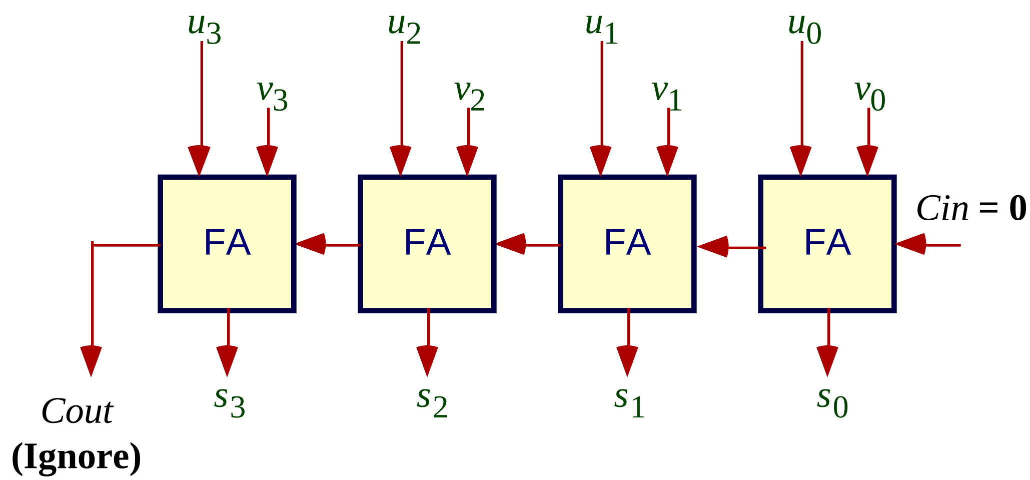
Constant$_1$	Constant$_2$	Relation	Evaluation
`0`	`0U`	`==`	`unsigned`
`-1`	`0`	`<`	`signed`
`-1`	`0U`	`?`	
`2147483647`	`-2147483648`	`?`	
`2147483647U`	`-2147483648`	`?`	
`-1`	`-2`	`?`	
`(unsigned) -1`	`-2`	`?`	
`2147483647`	`2147483648U`	`?`	
`2147483647`	`(int) 2147483648U`	`?`	

Adder Circuit

Full Adder



Adder Structure



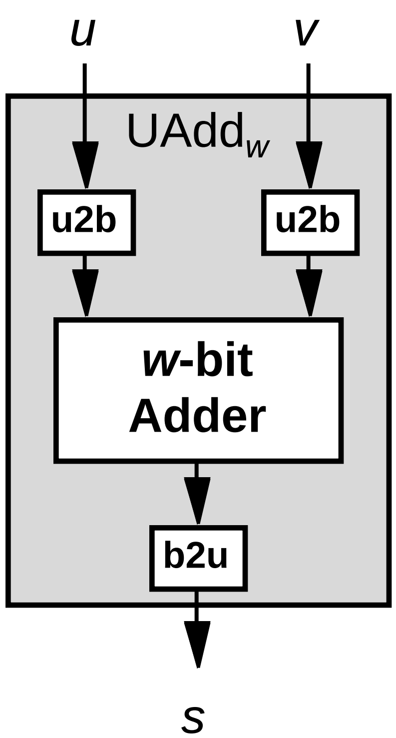
Assume

- **Carry Input = 0**
- **Ignore Carry Output**

Observation

- **Output at bit position *i* depends only on inputs at positions 0 to *i***

Unsigned Addition



Standard Addition Function

- **Ignores carry output**

Implements Modular Arithmetic

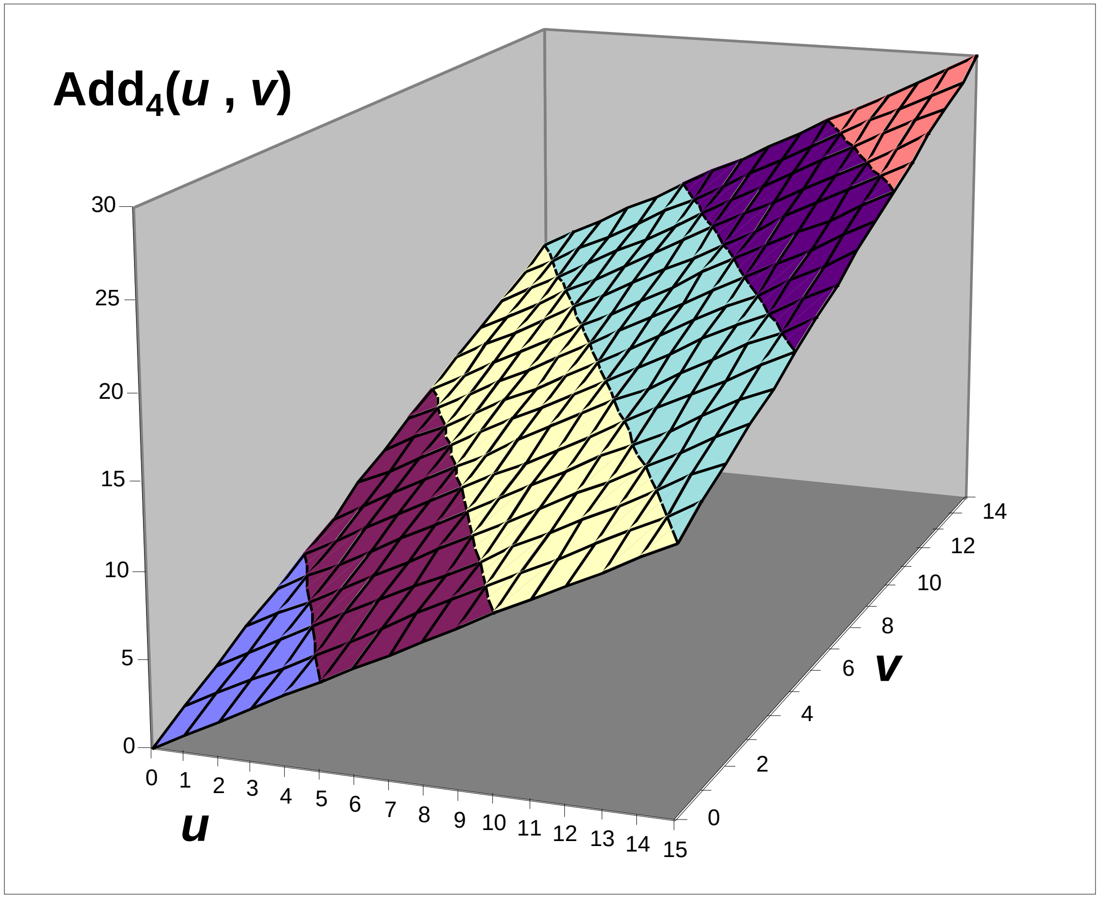
$s \quad = \quad \text{UAdd}_w(u\,,\,v) \quad = \quad u + v \mod 2^w$

$$UAdd_w(u,v) = \begin{cases} u+v & u+v < 2^w \\ u+v-2^w & u+v \geq 2^w \end{cases}$$

Visualizing Integer Addition

Integer Addition

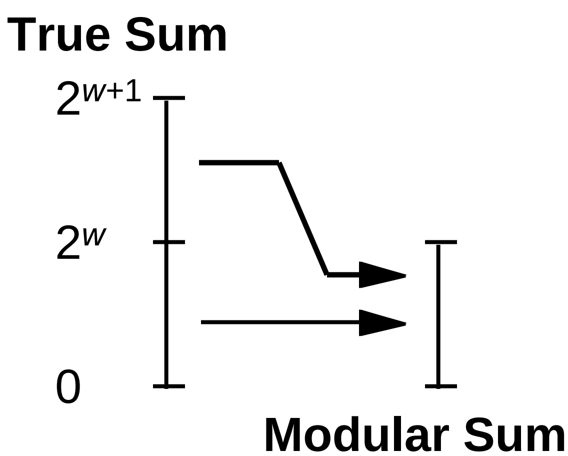
- **4-bit integers *u* and *v***
- **Compute sum Add$_4$(*u* , *v*)**
- **Values increase linearly with *u* and *v***
- **Forms planar surface**

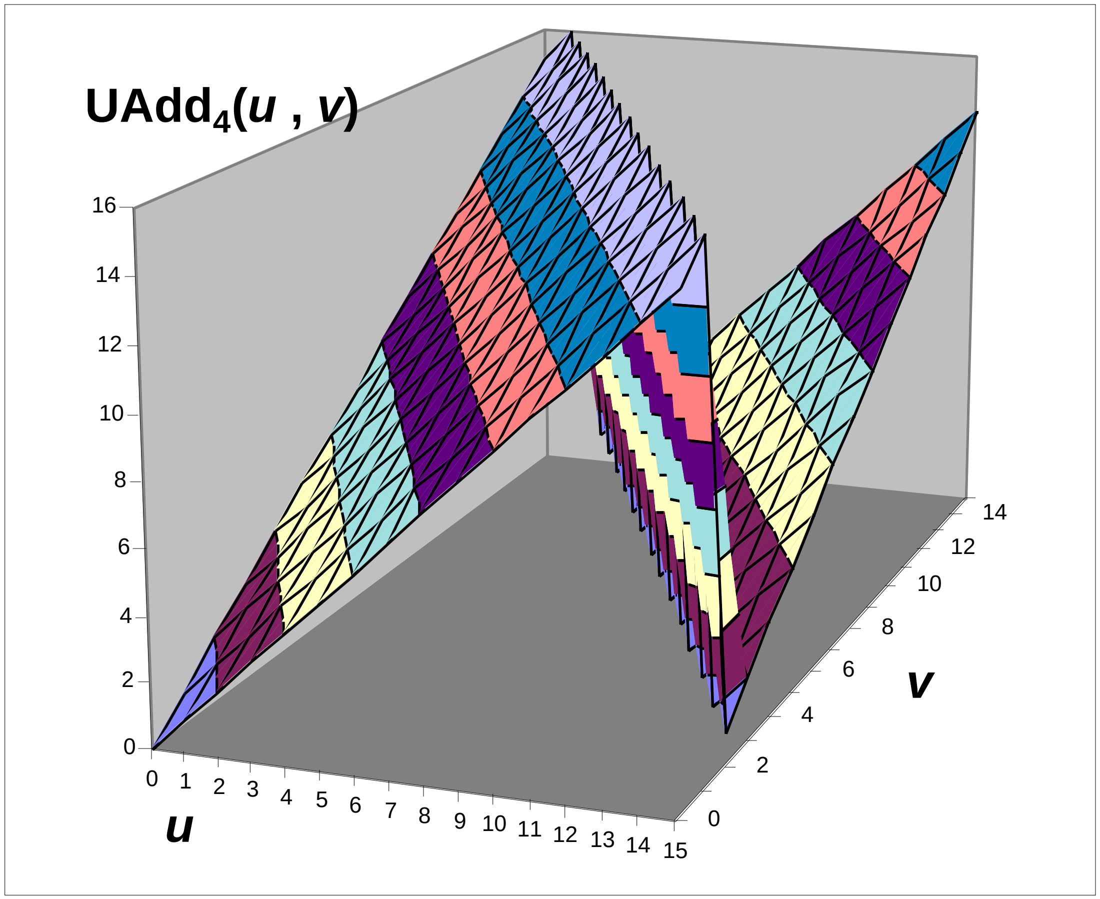


Visualizing Unsigned Addition

Wraps Around

- **If true sum $\geq 2^w$**
- **At most once**





Mathematical Properties

Modular Addition Forms an *Abelian Group*

- **Closed under addition**

  $0 \leq \mathrm{UAdd}_w(u, v) \leq 2^w - 1$
- **Commutative**

  $\mathrm{UAdd}_w(u, v) = \mathrm{UAdd}_w(v, u)$
- **Associative**

  $\mathrm{UAdd}_w(t, \mathrm{UAdd}_w(u, v)) = \mathrm{UAdd}_w(\mathrm{UAdd}_w(t, u), v)$
- **0 is additive identity**

  $\mathrm{UAdd}_w(u, 0) = u$
- **Every element has additive inverse**
  - Let $\mathrm{UComp}_w(u) = 2^w - u$

  $\mathrm{UAdd}_w(u, \mathrm{UComp}_w(u)) = 0$

Detecting Unsigned Overflow

Task

- **Given** $s = \text{UAdd}_w(u, v)$
- **Determine if** $s = u + v$

Application

```
unsigned s, u, v;
s = u + v;
```

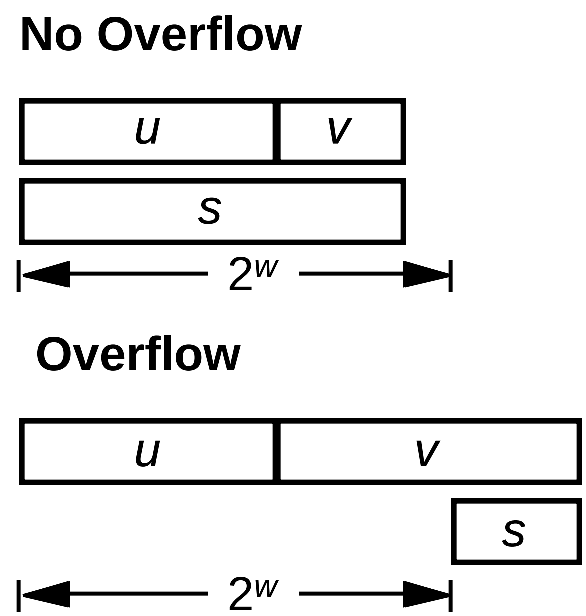
- **Did addition overflow?**

Claim

- **Overflow iff** $s < u$

```
ovf = (s < u)
```

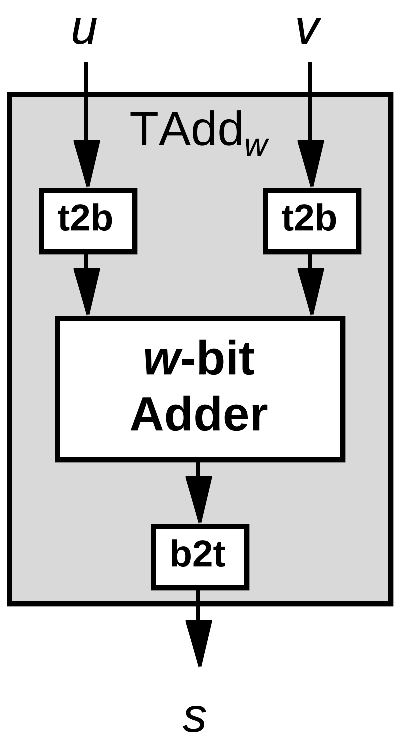
- **Or symmetrically iff** $s < v$



Proof

- **Know that** $0 \leq v < 2^w$
- **No overflow** **==>** $s = u + v \geq u + 0 = u$
- **Overflow** **==>** $s = u + v - 2^w < u + 0 = u$

Two's Complement Addition



Add Two's Complement Numbers with Conventional Adder

- E.g., signed addition in C:

```
int s, u, v;
s = u + v;
```

- Ignores carry output

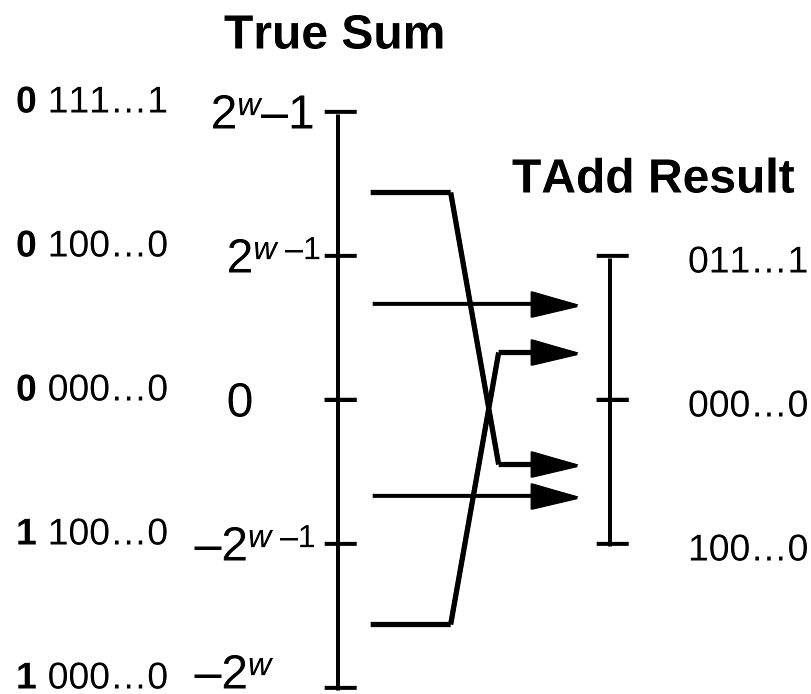
$$s = \mathrm{TAdd}_w(u, v)$$

- What is the behavior of this operation?

Characterizing TAdd

Functionality

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



$$TAdd_w(u,v) = \begin{cases} u+v+2^{w-1} & u+v < TMin_w \\ u+v & TMin_w \le u+v \le TMax_w \\ u+v-2^{w-1} & TMax_w < u+v \end{cases}$$

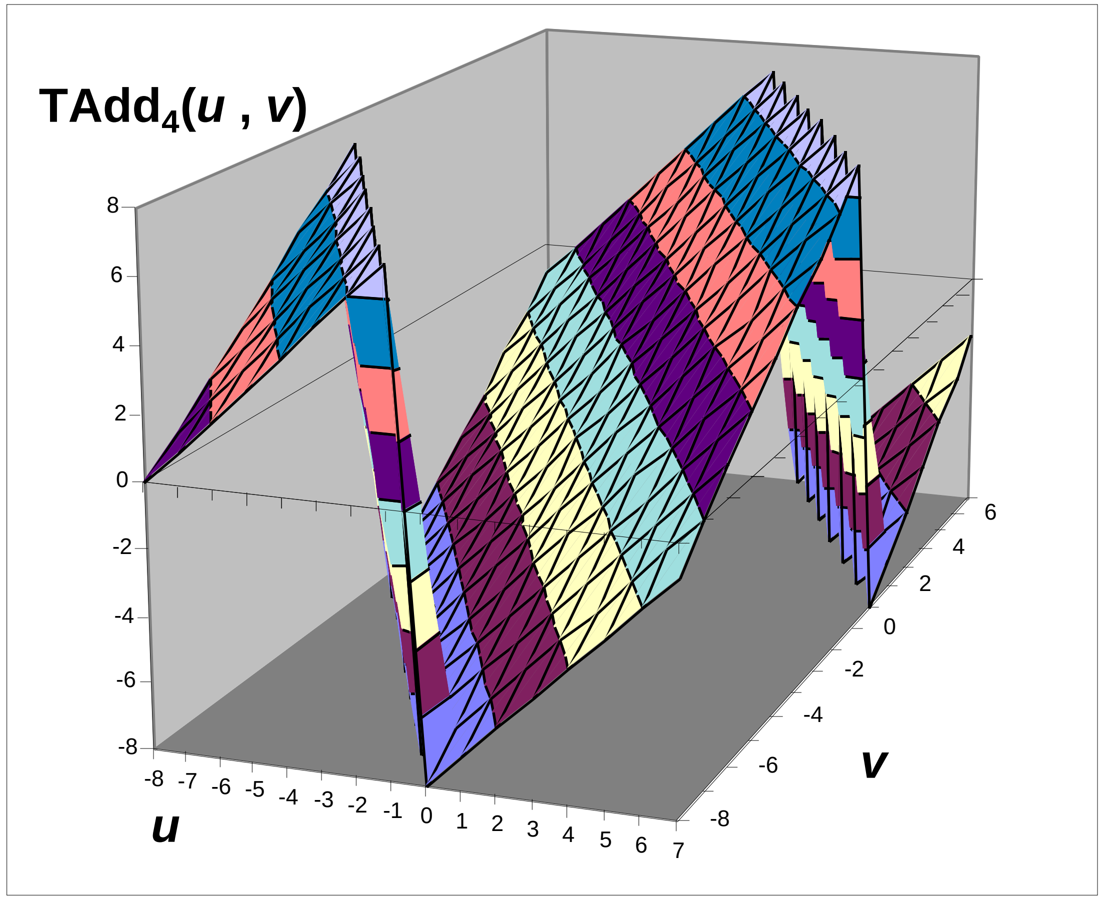
Visualizing 2's Comp. Addition

Values

- **4-bit two's comp.**
- **Range from -8 to +7**

Wraps Around

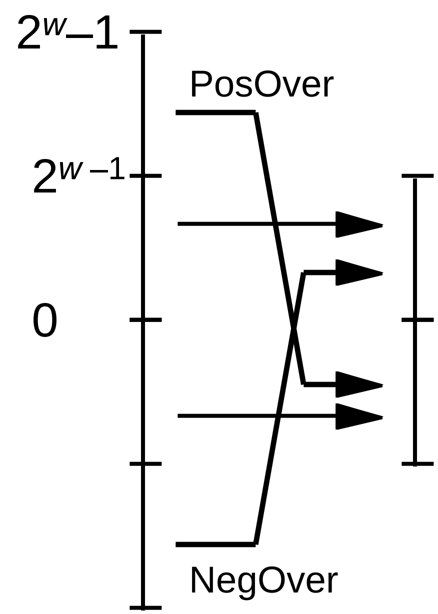
- **If sum ≥ 2^{w-1}**
  - At most once
  - Becomes negative
- **If sum < -2^{w-1}**
  - At most once
  - Becomes positive



Detecting 2's Comp. Overflow

Task

- **Given** $s = \mathrm{TAdd}_w(u, v)$
- **Determine if** $s = u + v$



Application

```
int s, u, v;
s = u + v;
```

Claim

- **Overflow iff either:**
  - $u, v < 0, s \geq 0$ (NegOver)
  - $u, v \geq 0, s < 0$ (PosOver)

```
ovf = (u<0 == v<0) && (u<0 != s<0);
```

Proof

- **Easy to see that if** $u \geq 0$ **and** $v < 0$**, then** $TMin_w \leq u + v \leq TMax_w$
- **Symmetrically if** $u < 0$ **and** $v \geq 0$
- **Other cases from analysis of** TAdd

Mathematical Properties of TAdd

Isomorphic Algebra to UAdd

- $TAdd_w(u, v) = U2T(UAdd_w(T2U(u), T2U(v)))$
  - Since both have identical bit patterns

Two's Complement Under TAdd Forms a Group

- **Closed, Commutative, Associative, 0 is additive identity**
- **Every element has additive inverse**
  - Let $TComp_w(u) = U2T(UComp_w(T2U(u))$

  $TAdd_w(u, TComp_w(u)) = 0$

$$\boxed{TComp_w(u) \quad = \quad \begin{cases} -u & u \neq TMin_w \\ ?? & u = TMin_w \end{cases}}$$

Two's Complement Negation

Mostly like Integer Negation

- TComp(*u*) = –*u*

TMin is Special Case

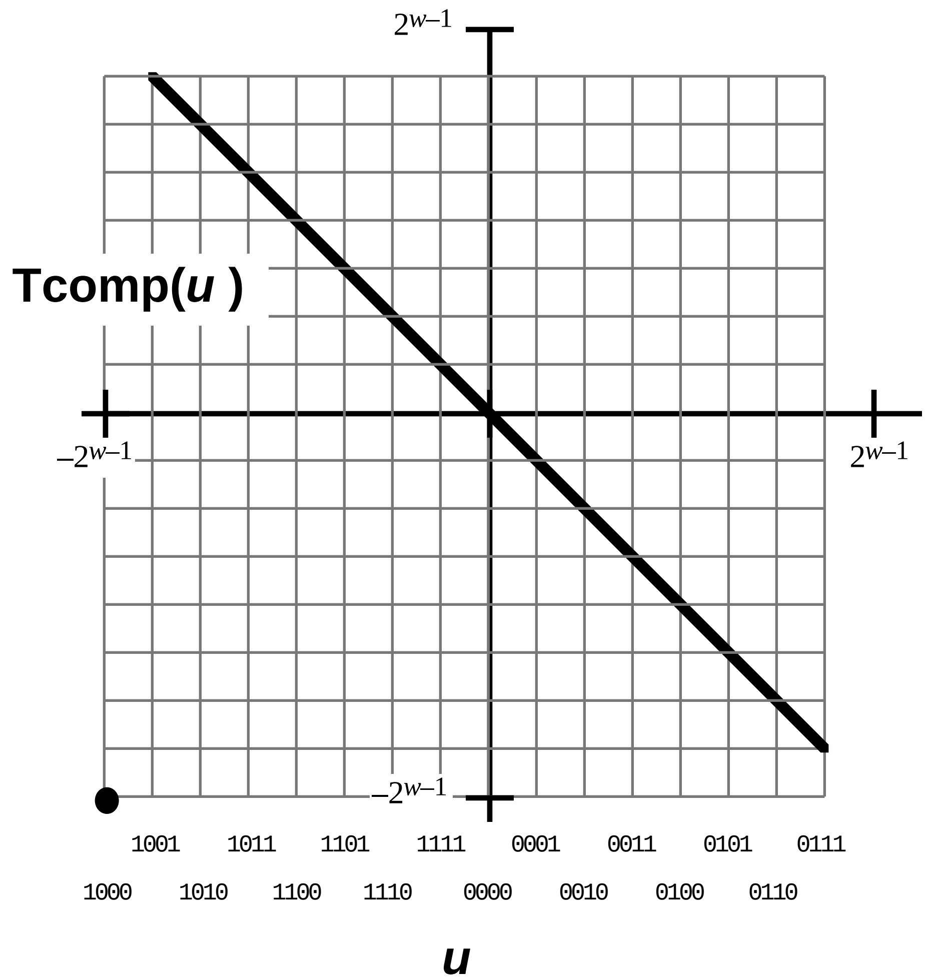
- TComp(*TMin*) = *TMin*

Negation in C is *NOT* True Negation

```
mx = -x
```

- **Really** *mx* = TComp(*x*)
- **But is a complement**

```
x + -x == 0
```



Negating with Complement & Increment

In C

```
~x + 1 == -x
```

Complement

- **Derive using property that for bit value *x* :** $\bar{x} = 1 – x$

$$
\begin{aligned}
B2T_w\left(\bar{X}\right) &= \sum_{i=0}^{w-2} (1 - x_i)\ 2^i - (1 - x_{w-1})\ 2^{w-1} \\
&= \left(2^{w-1} - 1\right) + 2^{w-1} - B2T_w(X) \\
&= -B2T_w(X) - 1
\end{aligned}
$$

Increment

$$
\begin{aligned}
TAdd_w\left(B2T_w\left(\bar{X}\right), 1\right) &= \begin{cases} -B2T_w(X) & X \neq 100 \quad 0 \\ TMin_w & X = 100 \quad 0 \end{cases} \\
&= TComp_w\left(B2T_w(X)\right)
\end{aligned}
$$

Comp. & Incr. Example

X	**B2T(X)**	$\overline{X}$	**B2T($\overline{X}$)**	Add($\overline{X}$, 1)	**B2T(•)**
0000	0	1111	–1	0000	0
0001	1	1110	–2	1111	–1
0010	2	1101	–3	1110	–2
0011	3	1100	–4	1101	–3
0100	4	1011	–5	1100	–4
0101	5	1010	–6	1011	–5
0110	6	1001	–7	1010	–6
0111	7	1000	–8	1001	–7
1000	–8	0111	7	1000	–8
1001	–7	0110	6	0111	7
1010	–6	0101	5	0110	6
1011	–5	0100	4	0101	5
1100	–4	0011	3	0100	4
1101	–3	0010	2	0011	3
1110	–2	0001	1	0010	2
1111	–1	0000	0	0001	1

Complement Upper Bits

Task

- **Given bit vector** X
- **Negate it without using addition**

Rule

- **Let** k **be minimum value such that** $x_k = 1$
- **I.e.,** X **=** x_{w-1} **,** x_{w-2} **,…** x_{k+1} ,1, 0,…,0
- **Compute** Negate(X) **=** $\overline{x_{w-1}}$ **,** $\overline{x_{w-2}}$ **,…** $\overline{x_{k+1}}$,1, 0,…,0

Justification

- **Effect of complement:**

$$\overline{X} = \overline{x_{w-1}}, \overline{x_{w-2}}, \ldots \overline{x_{k+1}}, 0,1,\ldots,1$$

- **Effect of increment:**

$$\text{Add}(\overline{X}, 1) = \overline{x_{w-1}}, \overline{x_{w-2}}, \ldots \overline{x_{k+1}}, 1,0,\ldots,0$$

Observation

- **Low order bit of *X* and –*X* identical**

Upper Bits Complement Example

X	**B2T(*X*)**	UC(*X*)	**B2T(•)**
0000	0	0000	0
000*1*	1	111*1*	–1
00*10*	2	11*10*	–2
001*1*	3	110*1*	–3
0*100*	4	1*100*	–4
010*1*	5	101*1*	–5
01*10*	6	10*10*	–6
011*1*	7	100*1*	–7
1000	–8	*1000*	–8
100*1*	–7	011*1*	7
10*10*	–6	01*10*	6
101*1*	–5	010*1*	5
1*100*	–4	0*100*	4
110*1*	–3	001*1*	3
11*10*	–2	00*10*	2
111*1*	–1	000*1*	1

Sign Extension

Task:

- **Given *w*-bit signed integer *x***
- **Convert it to *w*+*k*-bit integer with same value**

Rule:

- **Make *k* copies of sign bit:**
- $X' = \underbrace{x_{w-1}, \ldots, x_{w-1}}_{k \text{ copies of MSB}}, x_{w-1}, x_{w-2}, \ldots, x_0$

Correctness Argument

- **Induct on *k*, i.e., that extending by single bit maintains value**
- **Key observation:** $-2^{w-1} = -2^{w} + 2^{w-1}$

$$
\begin{aligned}
B2T(X') &= -x_{w-1}\,2^{w} + x_{w-1}\,2^{w-1} + \sum_{i=0}^{w-2} x_i\,2^i \\
&= x_{w-1}\left(-2^{w} + 2^{w-1}\right) + \sum_{i=0}^{w-2} x_i\,2^i \\
&= x_{w-1}\left(-2^{w-1}\right) + \sum_{i=0}^{w-2} x_i\,2^i \\
&= B2T(X)
\end{aligned}
$$

Multiplication

Computing Exact Product of w-bit numbers x, y

- **Either signed or unsigned**

Ranges

- **Unsigned:** $0 \leq x * y \leq (2^w - 1)^2 = 2^{2w} - 2^{w+1} + 1$
  - Up to $2w$ bits
- **Two's complement min:** $x * y \geq (-2^{w-1})*(2^{w-1}-1) = -2^{2w-2} + 2^{w-1}$
  - Up to $2w-1$ bits
- **Two's complement max:** $x * y \leq (-2^{w-1})^2 = 2^{2w-2}$
  - Up to $2w$ bits, but only for $TMin_w^2$

Maintaining Exact Results

- **Would need to keep expanding word size with each product computed**
- **Done in software by "arbitrary precision" arithmetic packages**
- **Also implemented in Lisp, ML, and other "advanced" languages**

Multiplication In C

Operation

```
int x, y;
int p = x * y;
```

- **Compute exact product of two *w*-bit numbers *x*, *y***
- **Truncate result to *w*-bit number *p* = $\text{TMult}_w(x, y)$**

Unsigned Counterpart

```
unsigned ux = (unsigned) x;
unsigned uy = (unsigned) y;
unsigned up = ux * uy
```

- **Truncates product to *w*-bit number *up* = $\text{UMult}_w(ux, uy)$**
- **Simply modular arithmetic**

  $up = ux \quad uy \mod 2^w$

Relation

- **Claim that up `== (unsigned) p`**

Multiplication Analysis

$$B2T(X) = \underbrace{\sum_{i=0}^{w-2} x_i \; 2^i}_{ax} - \underbrace{x_{w-1}}_{sx} 2^{w-1}$$

$x = ax - 2^{w-1}\, sx$

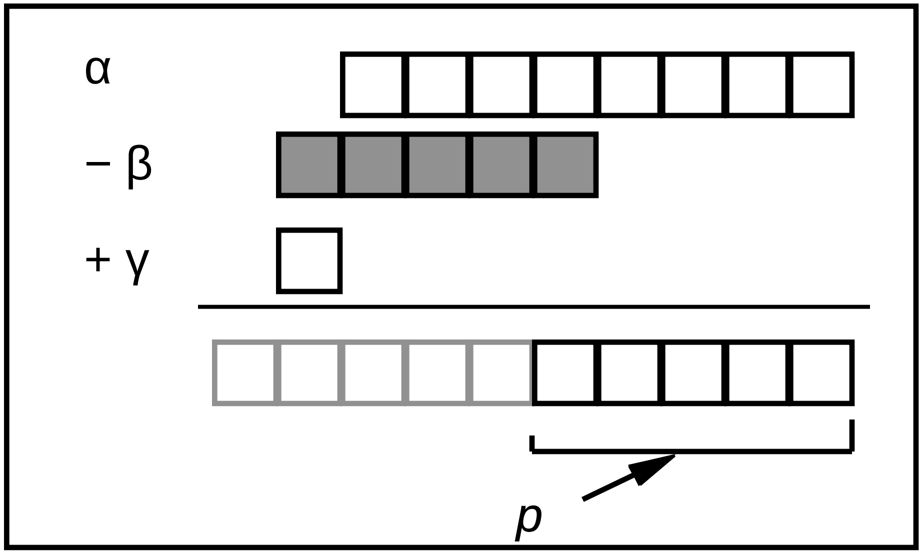
$ux = ax + 2^{w-1}\, sx$

$$x\;\; y \;=\; ax\;\; ay \;-\; 2^{w-1}\,(sx\;\; ay + sy\;\; ax) + 2^{2w-2}\, sx\;\; sy$$

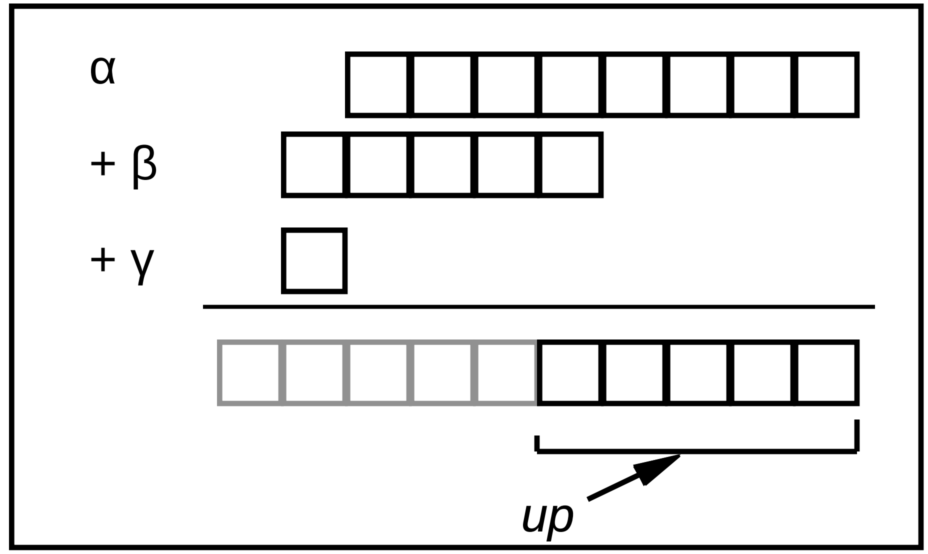
$$ux\;\; uy \;=\; \underbrace{ax\;\; ay}_{\alpha\ (2w-2\text{ bits})} \;+\; 2^{w-1}\,\underbrace{(sx\;\; ay + sy\;\; ax)}_{\beta\ (w\text{ bits})} + 2^{2w-2}\, \underbrace{sx\;\; sy}_{\gamma\ (1\text{ bit})}$$

Multiplication Analysis (cont.)

Signed (w = 5)



Unsigned



Observe

- Only LSB of β affects values of *p* and *up*
- LSB of β and −β are the same
- *p* and *up* have matching bit patterns!

Algebraic Properties of Unsigned

Unsigned Multiplication with Addition Forms Commutative Ring

- **Addition is commutative group**
- **Closed under multiplication**

  $0 \le \mathrm{UMult}_w(u, v) \le 2^w - 1$
- **Multiplication Commutative**

  $\mathrm{UMult}_w(u, v) = \mathrm{UMult}_w(v, u)$
- **Multiplication is Associative**

  $\mathrm{UMult}_w(t, \mathrm{UMult}_w(u, v)) = \mathrm{UMult}_w(\mathrm{UMult}_w(t, u), v)$
- **1 is multiplicative identity**

  $\mathrm{UMult}_w(u, 1) = u$
- **Multiplication distributes over addtion**

  $\mathrm{UMult}_w(t, \mathrm{UAdd}_w(u, v)) = \mathrm{UAdd}_w(\mathrm{UMult}_w(t, u), \mathrm{UMult}_w(t, v))$

Algebraic Properties of Two's Comp.

Isomorphic Algebras

- **Unsigned multiplication and addition**
  - Truncating to *w* bits
- **Two's complement multiplication and addition**
  - Truncating to *w* bits

Both Form Rings

- **Isomorphic to ring of integers mod** 2^w
- **NOT isomorphic to ring of integers**

C Puzzles

- **Taken from Exam #2, CS 347, Spring '97**
- **Assume machine with 32 bit word size, two's complement integers**
- **For each of the following C expressions, either:**
  - Argue that is true for all argument values
  - Give example where not true

Initialization

```
int x = foo();

int y = bar();

unsigned ux = x;

unsigned uy = y;
```

- `x < 0` ⇒ `((x*2) < 0)`
- `ux >= 0`
- `x & 7 == 7` ⇒ `(x<<30) < 0`
- `ux > -1`
- `x > y` ⇒ `-x < -y`
- `x * x >= 0`
- `x > 0 && y > 0` ⇒ `x + y > 0`
- `x >= 0` ⇒ `-x <= 0`