Bits, Bytes and Integers Review

Recitation 002
September 15th, 2017

Seonghak Kim (ksh931102@gmail.com)
Architecture and Code Optimization (ARC) Lab
Seoul National University

Contents

- **Review (concentrating on some tricky points)**
- **Practice**
- **Lab demonstration**

Obtaining Hexadecimal Numbers

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

- **Binary to Hex**
  - Convert every consecutive 4 bits into a hex digit
- **Hex to Binary**
  - Convert each hex digit into 4 bits

Applications of Logical Operators

- **Converting into Boolean**
  - If all bits are 0, return 0. Otherwise, 1.
  - `!!x`
- **Saturating Information**
  - Check whether the third bit is zero
  - `!!(x & ` 100_2`)`
- **Aggregating bits**
  - Data of 2 bytes consisted with two independent bytes
  - `x << 8 | y`

Early Termination Examples

- `p && *p`
  - Check the availability of the pointer
- `1 && x++`
  - Unpredictable behaviors
  - Bad coding style!
- `(1 && 0) && 1`
  - Early terminations are executed recursively

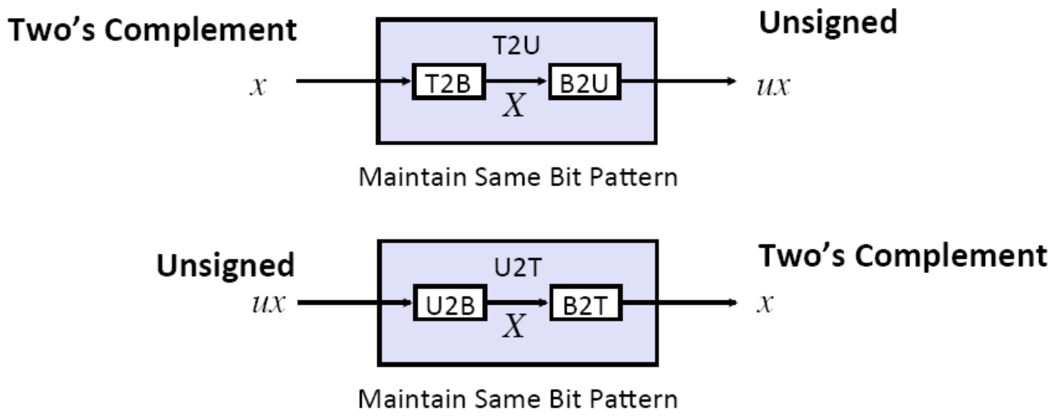
Shift Operators and Value

- **Is left shifting equivalent to multiplying power of 2?**
  - Yes. But it also suffers from overflows
  - 1<<31 is negative when treated as a signed number.

- **Is arithmetic shifting equivalent to divid-ing by power of 2?**
  - Yes. In detail:
  - Rounds down for positive integers
  - And rounds up for negative integers

Type Casting

- **Bit representations can be changed**
  - Considering type casting from int to float …



- **Mappings between unsigned and two's complement numbers:**
  Keep bit representations and reinterpret

- **Note that the correspondence of bit representation was intended**

Type Casting Puzzles

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

- **1. Decide the type of operands**
- **2. Decide the casted type of operands**
- **3. Calculate the casted value**
- **4. Calculate the result**

Extension and Truncation

W = 3

100	-4
101	-3
110	-2
111	-1
000	0
001	1
010	2
011	3

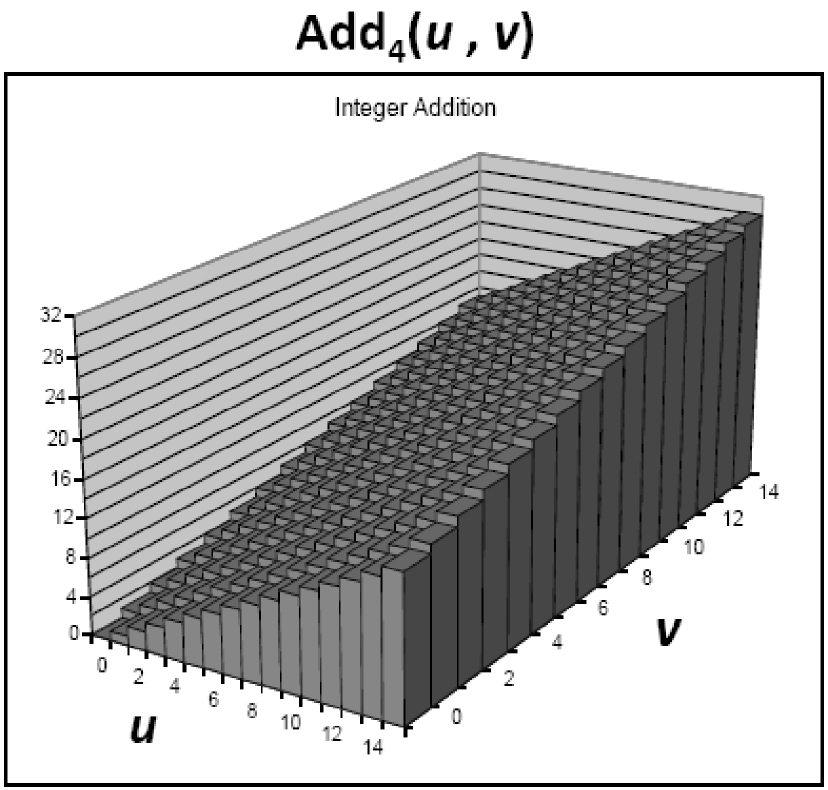
W = 2

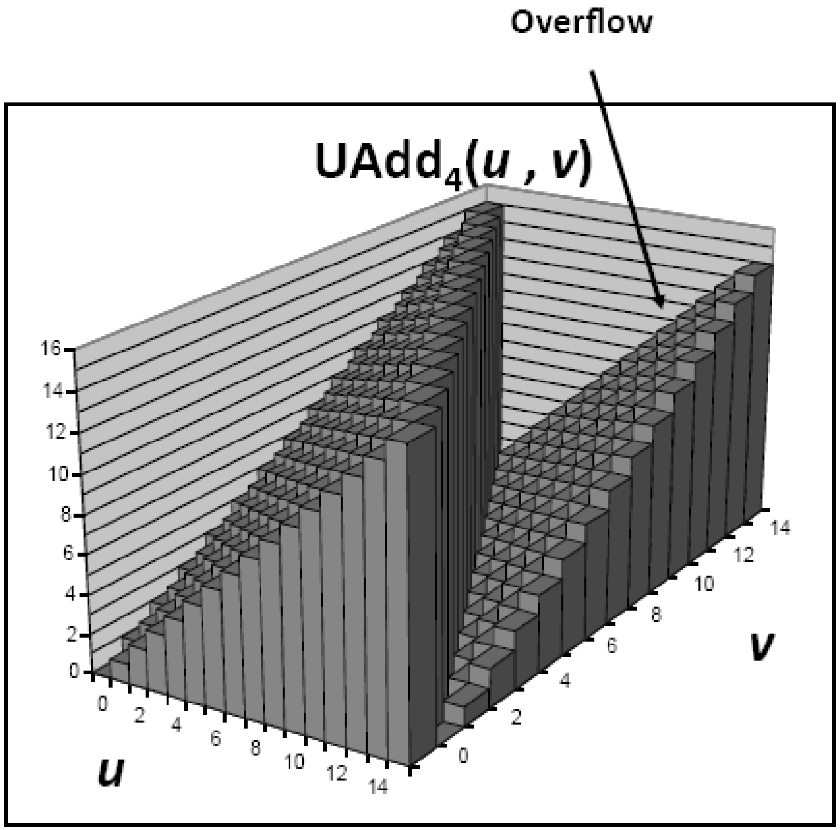
-2	10
-1	11
0	00
1	01

- **'Similar to mod'**
  - Preserves the remainder of x divided by 2^w

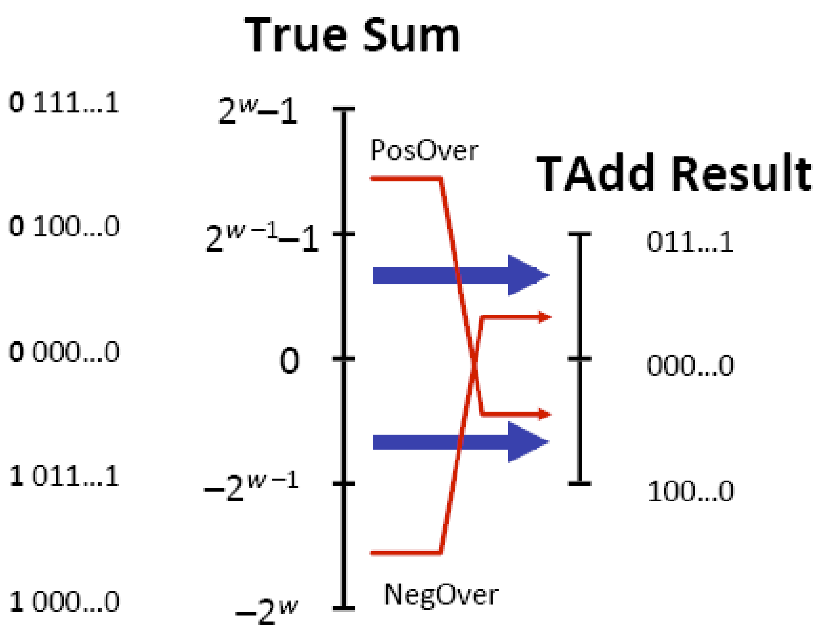
Unsigned Addition

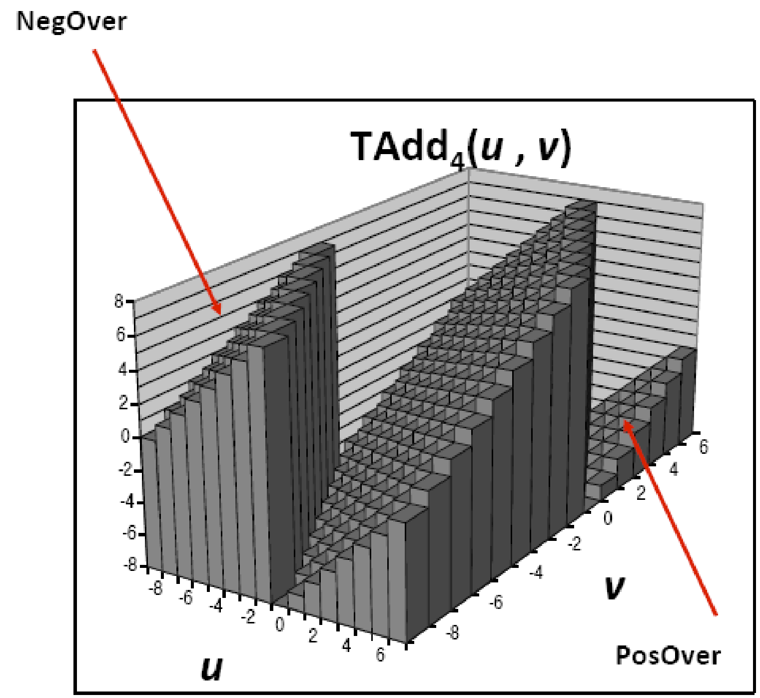
- **UAddw(u,v) = u + v mod 2^w**
  - = addition + truncation!





Signed Addition





- **TAdd and UAdd have identical bit-level behavior: why?**
  - Both are addition + truncation
  - ‘similar to mod’

Practice

- **Practice Problem 2.15: Using only bit-level and logical operations, write a C expression that is equivalent to `x == y`.**

- **Specify the bit-level behavior of the function**
- **Making zero is important when it comes to Boolean**

Practice

- **Practice Problem 2.21: Assuming the expressions are evaluated when executing a 32-bit program on a machine that uses two's-complement arithmetic, evaluate the following.**

```
-2147483647-1 == 2147483648U
 -2147483647-1 < 2147483647
-2147483647-1U < 2147483647
 -2147483647-1 < -2147483647
-2147483647-1U < -2147483647
```

Practice

- **Practice Problem 2.27: Write a function with the following prototype**

```
/* Determine whether arguments can be
   added without overflow */

int uadd_ok(unsigned x, unsigned y);
```

Lab Demonstration

Q&A

- **Thank you for paying attention.**