



# CS 5541 – Computer Systems

"Based on lecture notes developed by Randal E. Bryant and David R. O'Hallaron in conjunction with their textbook "Computer Systems: A Programmer's Perspective"



# Module 1

## Representing Numbers

### Part 3 — Integer Operations

From: Computer Systems, Chapter 2



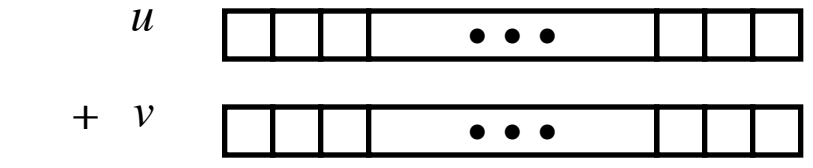
Instructor: James Yang  
<https://cs.wmich.edu/~zjiang>  
[zjiang.yang@wmich.edu](mailto:zjiang.yang@wmich.edu)

# Today: Bits, Bytes, and Integers

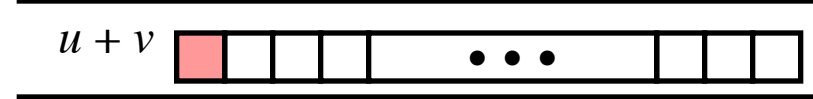
- Representing information as bits
- Bit-level manipulations
- Integers
  - Representation: unsigned and signed
  - Conversion, casting
  - Expanding, truncating
  - **Addition, negation, multiplication, shifting**
- Representations in memory, pointers, strings
- Summary

# Unsigned Addition

Operands:  $w$  bits



True Sum:  $w+1$  bits



Discard Carry:  $w$  bits

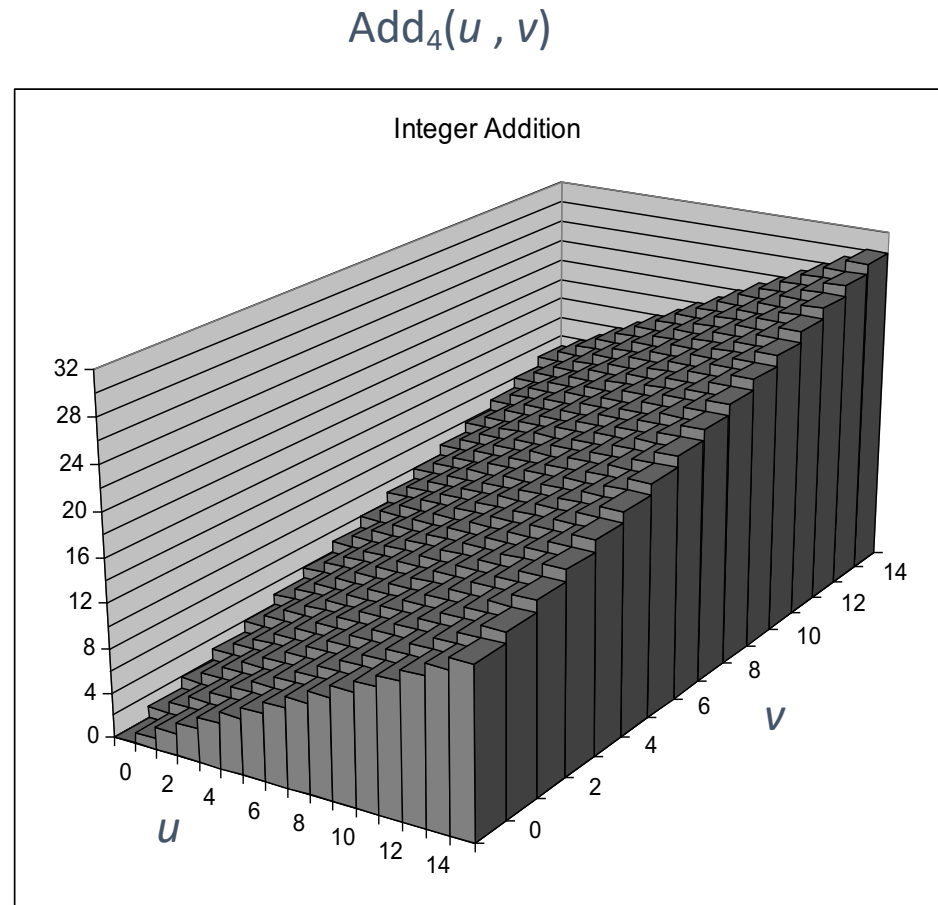


- Standard Addition Function
  - Ignores carry output
- Implements Modular Arithmetic
$$s = \text{UAdd}_w(u, v) = u + v \bmod 2^w$$



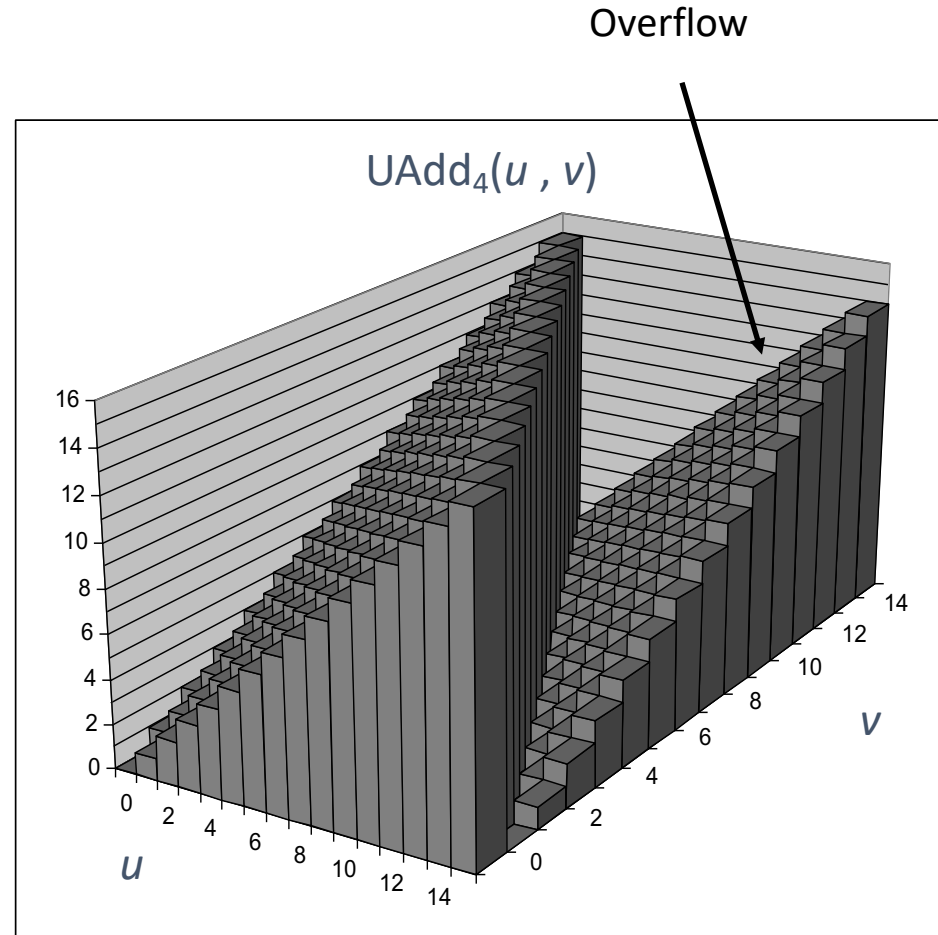
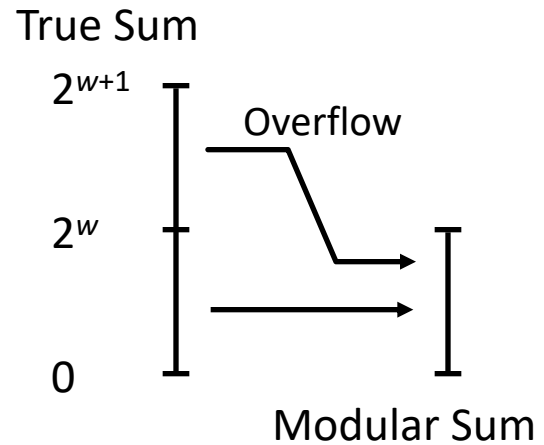
# Visualizing (Mathematical) Integer Addition

- 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



# Visualizing Unsigned Addition

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

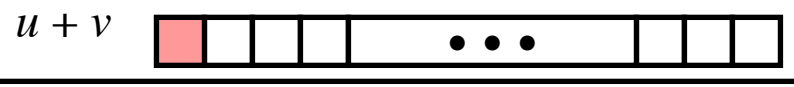


# Two's Complement Addition

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);
```

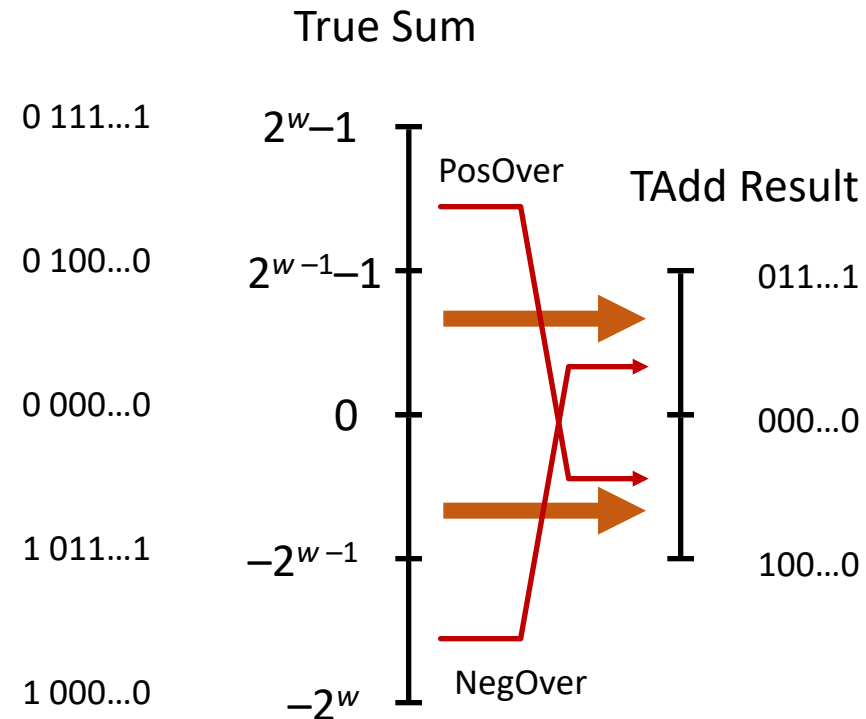
```
t = u + v
```

- Will give `s == t`

# TAdd Overflow

- Functionality

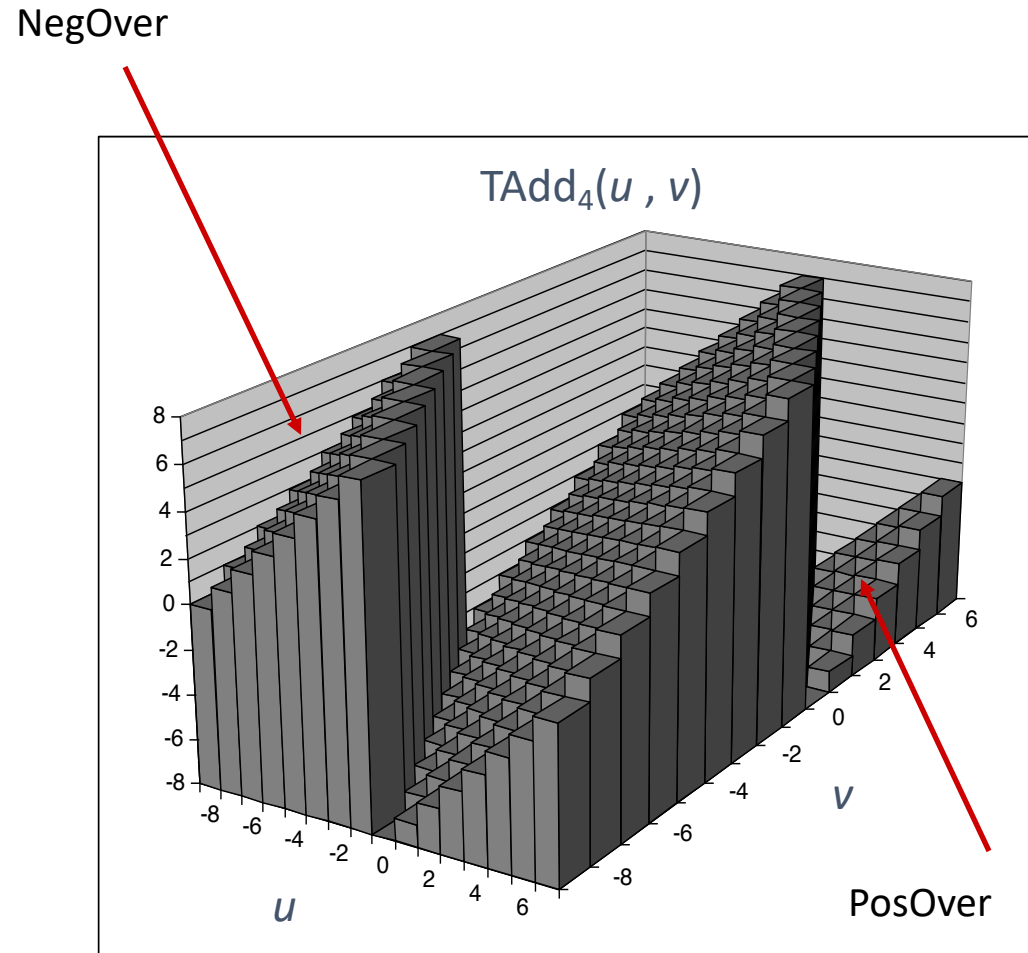
- True sum requires  $w+1$  bits
- Drop off MSB
- Treat remaining bits as 2's comp. integer





# Visualizing 2's Complement Addition

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



# Multiplication

- Goal: Computing Product of  $w$ -bit numbers  $x, y$ 
  - Either signed or unsigned
- But, exact results can be bigger than  $w$  bits
  - Unsigned: up to  $2w$  bits
    - Result range:  $0 \leq x * y \leq (2^w - 1)^2 = 2^{2w} - 2^{w+1} + 1$
  - Two's complement min (negative): Up to  $2w-1$  bits
    - Result range:  $x * y \geq (-2^{w-1}) * (2^{w-1} - 1) = -2^{2w-2} + 2^{w-1}$
  - Two's complement max (positive): Up to  $2w$  bits, but only for  $(TMin_w)^2$ 
    - Result range:  $x * y \leq (-2^{w-1})^2 = 2^{2w-2}$
- So, maintaining exact results...
  - would need to keep expanding word size with each product computed
  - is done in software, if needed
    - e.g., by “arbitrary precision” arithmetic packages

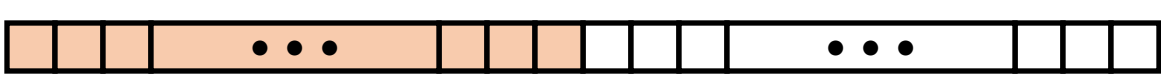
# Unsigned Multiplication in C

Operands:  $w$  bits

$u$  

$*$   $v$  

True Product:  $2*w$  bits

$u \cdot v$  

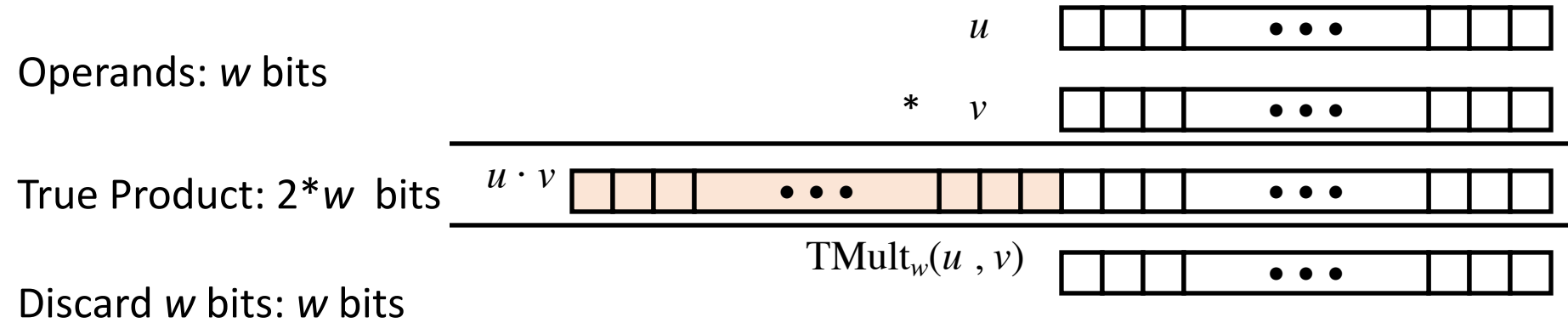
$\text{UMult}_w(u, v)$



Discard  $w$  bits:  $w$  bits

- Standard Multiplication Function
  - Ignores high order  $w$  bits
- Implements Modular Arithmetic
$$\text{UMult}_w(u, v) = u \cdot v \bmod 2^w$$

# Signed Multiplication in C

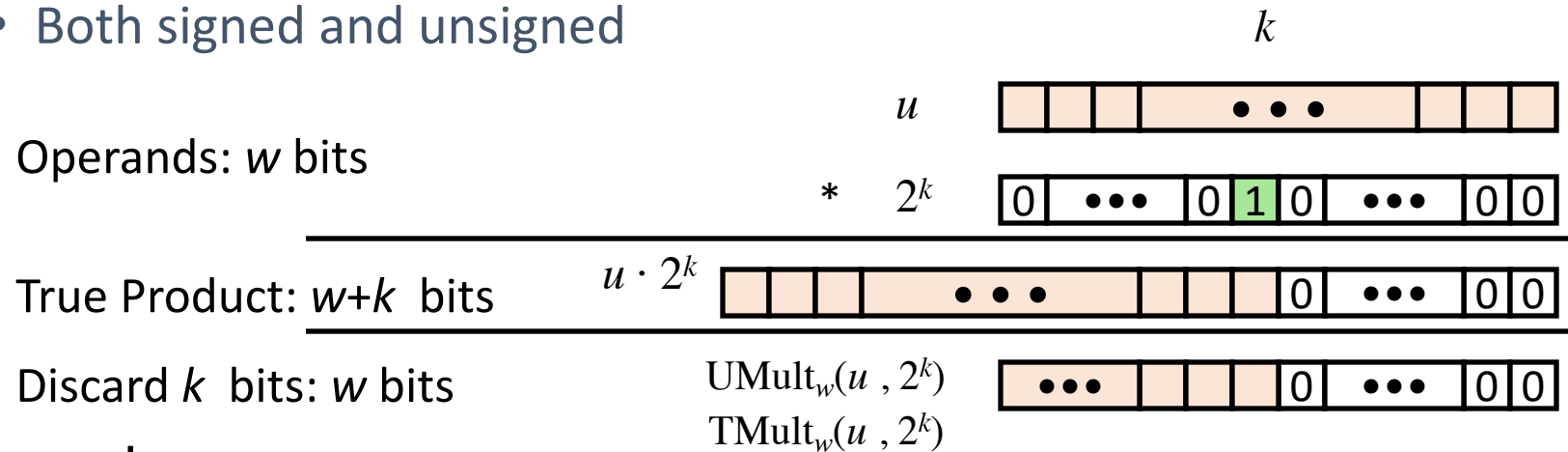


- Standard Multiplication Function
  - Ignores high order  $w$  bits
  - Some of which are different for signed vs. unsigned multiplication
  - Lower bits are the same

# Power-of-2 Multiply with Shift

- Operation

- $u \ll k$  gives  $u * 2^k$
- Both signed and unsigned

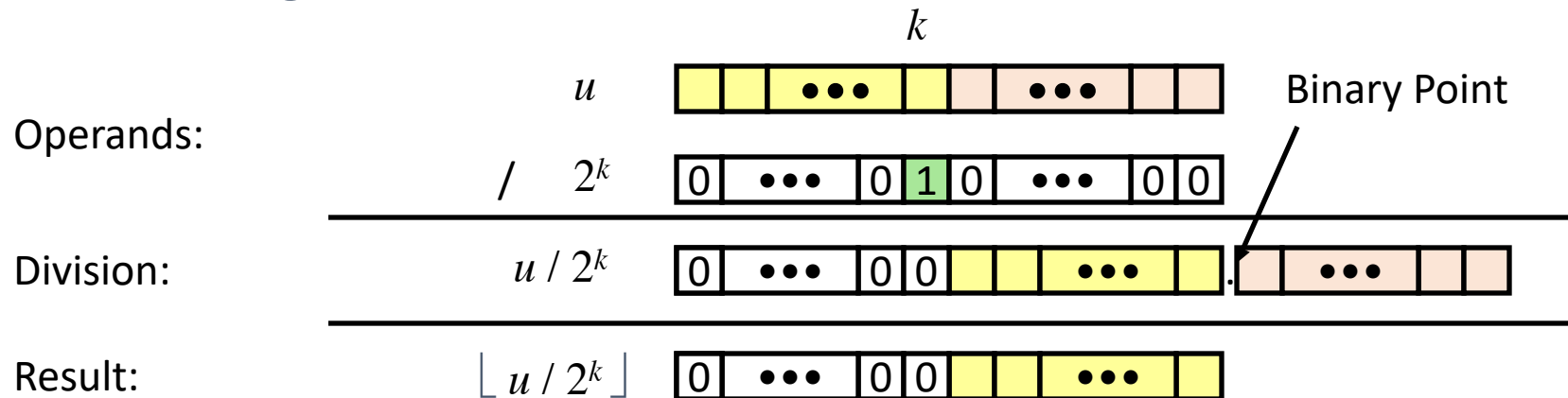


- Examples

- $u \ll 3 \quad == \quad u * 8$
- $(u \ll 5) - (u \ll 3) \quad == \quad u * 24$
- Most machines shift and add faster than multiply
  - Compiler generates this code automatically

# Unsigned Power-of-2 Divide with Shift

- Quotient of Unsigned by Power of 2
  - $u \gg k$  gives  $\lfloor u / 2^k \rfloor$
  - Uses logical shift



	Division	Computed	Hex	Binary
<b>x</b>	<b>15213</b>	<b>15213</b>	3B 6D	00111011 01101101
<b>x &gt;&gt; 1</b>	<b>7606.5</b>	<b>7606</b>	1D B6	00011101 10110110
<b>x &gt;&gt; 4</b>	<b>950.8125</b>	<b>950</b>	03 B6	00000011 10110110
<b>x &gt;&gt; 8</b>	<b>59.4257813</b>	<b>59</b>	00 3B	00000000 00111011



# 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
  - **Summary**
- Representations in memory, pointers, strings

# Arithmetic: Basic Rules

- Addition:

- Unsigned/signed: Normal addition followed by truncate, same operation on bit level
- Unsigned: addition mod  $2^w$ 
  - Mathematical addition + possible subtraction of  $2^w$
- Signed: modified addition mod  $2^w$  (result in proper range)
  - Mathematical addition + possible addition or subtraction of  $2^w$

- Multiplication:

- Unsigned/signed: Normal multiplication followed by truncate, same operation on bit level
- Unsigned: multiplication mod  $2^w$
- Signed: modified multiplication mod  $2^w$  (result in proper range)

# Module 1 (Part 3)

## Summary

- Implement signed and unsigned integer addition
- Implement multiply and divide by shift