

# Integer arithmetic - Multiplication

Computer Systems  
5<sup>th</sup> Lecture, Sep 10 2017

Michael Kirkedal Thomsen

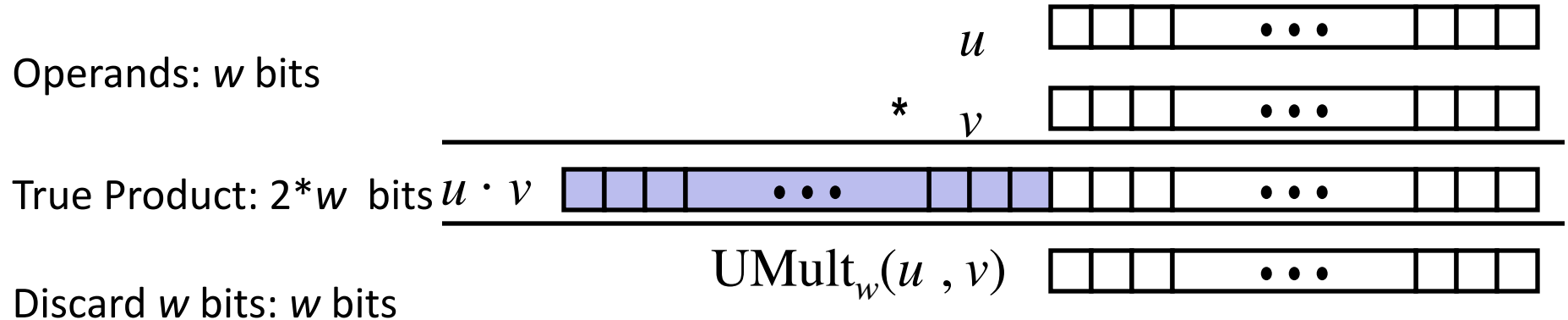
**Based on slides by:**

Randal E. Bryant and David R. O'Hallaron

# 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



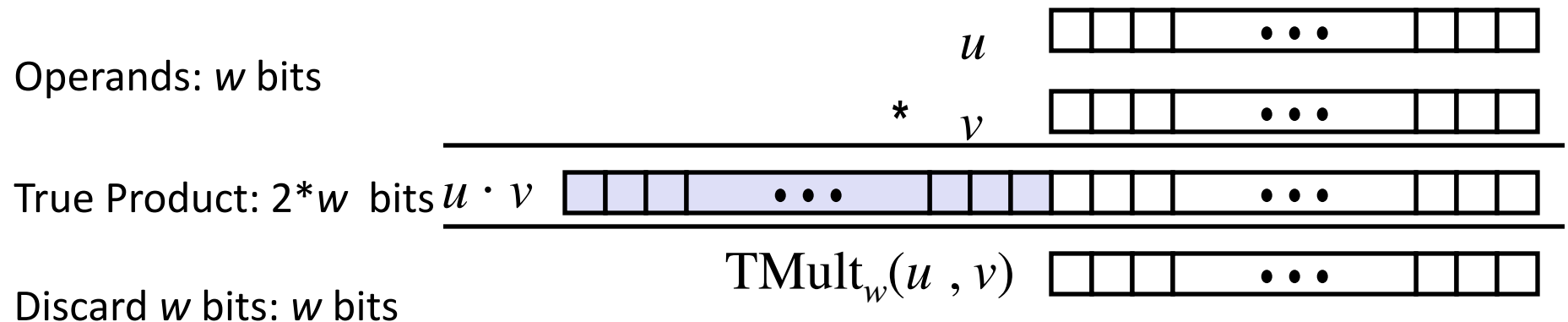
## ■ Standard Multiplication Function

- 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

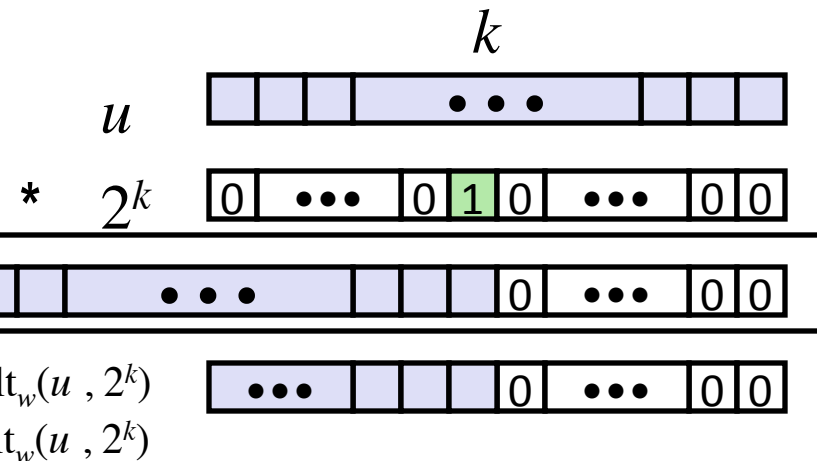
## ■ Operation

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

Operands:  $w$  bits

True Product:  $w+k$  bits

Discard  $k$  bits:  $w$  bits



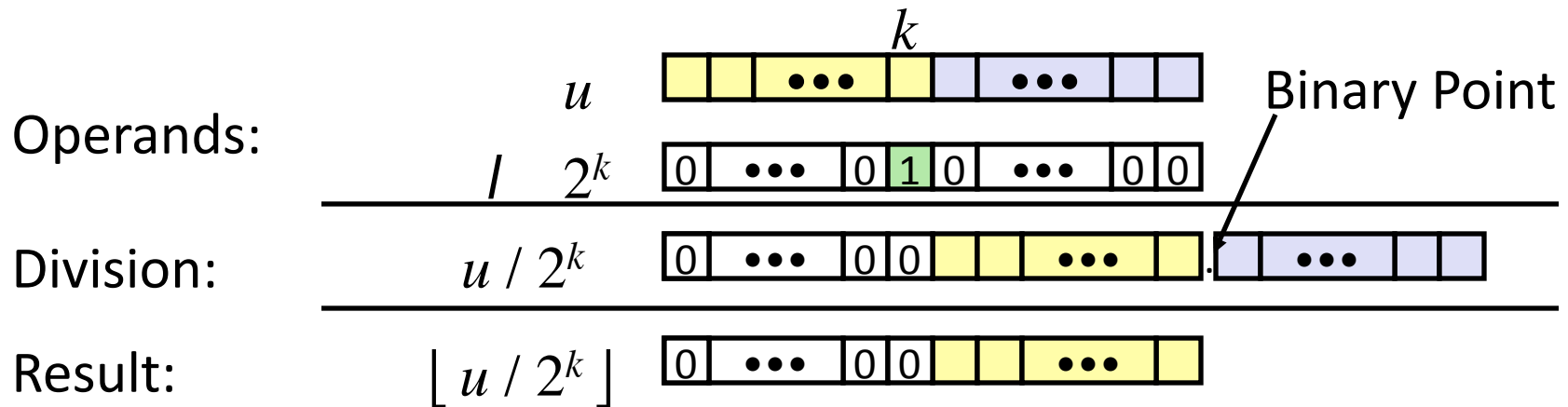
## ■ Examples

- $u \ll 3 == u * 8$
- $(u \ll 5) - (u \ll 3) == u * 24$
- 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>	<b>3B 6D</b>	<b>00111011 01101101</b>
<b>x &gt;&gt; 1</b>	<b>7606.5</b>	<b>7606</b>	<b>1D B6</b>	<b>00011101 10110110</b>
<b>x &gt;&gt; 4</b>	<b>950.8125</b>	<b>950</b>	<b>03 B6</b>	<b>00000011 10110110</b>
<b>x &gt;&gt; 8</b>	<b>59.4257813</b>	<b>59</b>	<b>00 3B</b>	<b>00000000 00111011</b>

# Unsigned Division C

Operands:  $w$  bits

True Product:  $2 \cdot w$  bits

Discard  $w$  bits:  $w$  bits

- **Standard Multiplication Function**

- Ignores high order  $w$  bits

- **Implements Modular Arithmetic**

$$\text{UMult}_w(u, v) = u \cdot v \bmod 2^w$$

# Today: Integer arithmetic and floating point

- **Recap**
- **Integer arithmetic**
  - Addition, negation, multiplication, shifting
  - Summary
- **Floating Points**



# 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  $2^w$
- Fast execution (usually 1 clock cycle)

## ■ 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)
- Slower execution (usually multiple clock cycles)

# Why Should I Use Unsigned?

- ***Don't*** use without understanding implications

- Easy to make mistakes

```
unsigned i;  
for (i = cnt-2; i >= 0; i--)  
    a[i] += a[i+1];
```

- Can be very subtle

```
#define DELTA sizeof(int)  
int i;  
for (i = CNT; i-DELTA >= 0; i-= DELTA)  
    . . .
```

# Counting Down with Unsigned

## ■ Proper way to use unsigned as loop index

```
unsigned i;  
for (i = cnt-2; i < cnt; i--)  
    a[i] += a[i+1];
```

## ■ See Robert Seacord, *Secure Coding in C and C++*

- C Standard guarantees that unsigned addition will behave like modular arithmetic
  - $0 - 1 \rightarrow UMax$

## ■ Even better

```
size_t i;  
for (i = cnt-2; i < cnt; i--)  
    a[i] += a[i+1];
```

- Data type **size\_t** defined as unsigned value with length = word size
- Code will work even if **cnt** = *UMax*
- What if **cnt** is signed and  $< 0$ ?

# Why Should I Use Unsigned? (cont.)

- **Do Use When Performing Modular Arithmetic**
  - Multiprecision arithmetic
- **Do Use When Using Bits to Represent Sets**
  - Logical right shift, no sign extension