Lecture 11

Integer Multiplication

CPSC 275
Introduction to Computer Systems

Multiplication

- Computing exact product of *w*-bit numbers, *x* and *y* (either signed or unsigned) gives the following ranges:
  - Unsigned:

$$0 \le x * y \le (2^w - 1)^2 = 2^{2w} - 2^{w+1} + 1$$

  - 2's comp:

min:

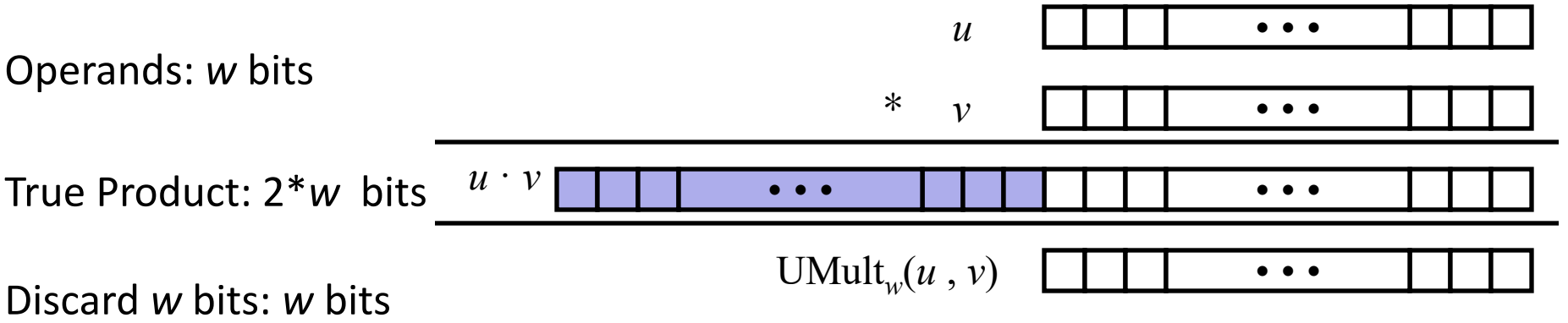
$$x * y \ge (-2^{w-1})*(2^{w-1}-1) = -2^{2w-2} + 2^{w-1}$$

max:

$$x * y \le (-2^{w-1})^2 = 2^{2w-2}$$

  - Require up to 2*w* bits

Unsigned Multiplication



- Implements modular arithmetic:

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

Example

Assume a 4-bit unsigned binary representation.

1010 * 0110 = ?

10 * 6

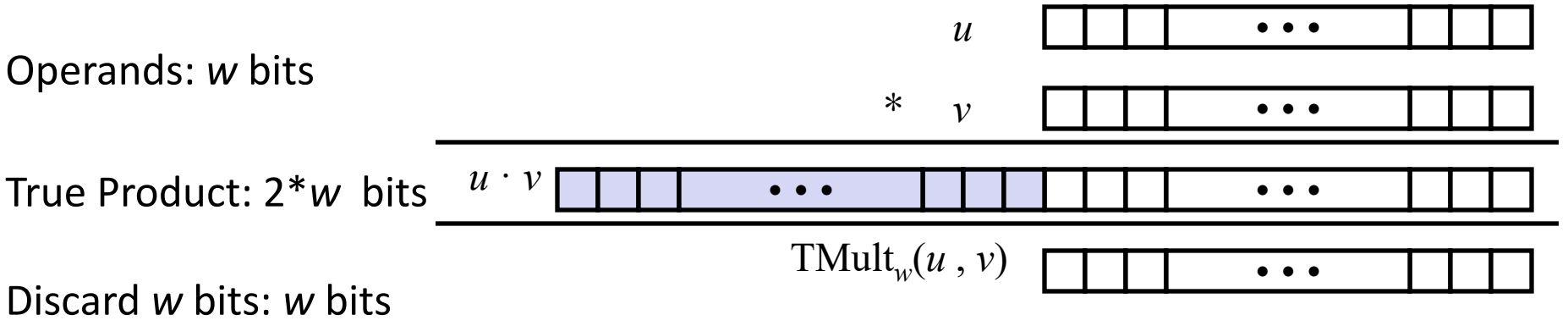
Example

Assume a 4-bit unsigned binary representation.

1010 * 0110 = 00111100 (overflow)

10 * 6 = 12 (?)

Signed Multiplication



- Ignores high order *w* bits
- Different interpretation for signed vs. unsigned multiplication
- Lower bits are the same

Signed Multiplication in Practice

- Method 1: Sign-Magnitude (indirect)
  - Find the magnitude of the two multiplicands
  - Multiply them together
  - Determine the sign
    - Same sign → positive
    - Different sign → negative

- Method 2: Sign-Extension (direct)
  - Sign-extend the two multiplicands to the product width
  - Multiply them together
  - Extract low *w* bits

Example

Assume a 3-bit signed binary representation.

111 × 011 = ?

-1 × 3

Example

Assume a 3-bit signed binary representation.

111 × 011 = 111101

-1 × 3 = -3

Example

Assume a 3-bit signed binary representation.

111 × 100 = ?

-1 × -4

Example

Assume a 3-bit signed binary representation.

111 × 100 = 000100 (overflow)

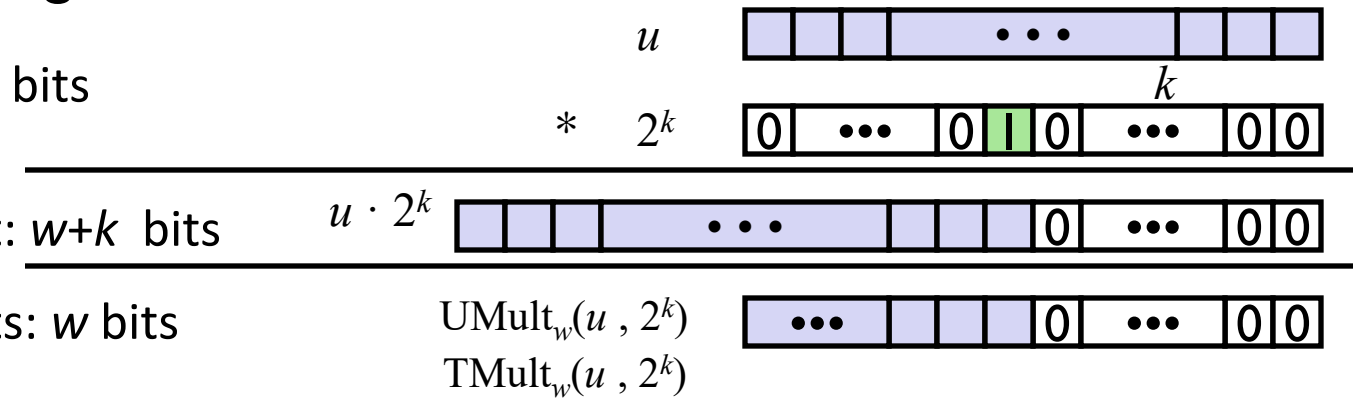
-1 × -4 = -4 (?)

Power-of-2 Multiply with Shift

- Operation

  `u << k` gives `u * `2^k

  for both signed and unsigned



- Most machines shift and add faster than multiply
  - Compiler generates this code automatically

- Examples

```
u << 3 == u * 8
u * 12 == ?
```

Dividing by Powers of 2

- Integer division is much slower than multiplication
- Dividing by 2^k can be done by a right shift by k.
  - logical right shift
  - arithmetic right shift
- Integer division always *rounds toward zero*,
  e.g., 7/3 = 2, -7/3 = -2

Unsigned Power-of-2 Divide with Shift

- Quotient of unsigned by power of 2

  `u >> k` gives $\lfloor$ `u / 2`^{k} $\rfloor$

  - Uses logical shift

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

Signed Power-of-2 Divide with Shift

- Quotient of signed by power of 2

  `u >> k` gives $\lfloor$ `u / 2^k` $\rfloor$

  - Uses arithmetic shift
  - Rounds wrong direction when `u < 0` (round down!)

	Division	Computed	Hex	Binary
`y`	-15213	-15213	C4 93	11000100 10010011
`y >> 1`	-7606.5	-7607	E2 49	11100010 01001001
`y >> 4`	-950.8125	-951	FC 49	11111100 01001001
`y >> 8`	-59.4257813	-60	FF C4	11111111 11000100

Correct Power-of-2 Divide

- Quotient of negative number by power of 2
  - Want $\lceil$ `x / 2^k` $\rceil$ (round toward 0)

- Use the property (from CPSC 203)

$$\lceil a / b \rceil = \lfloor (a + b - 1) / b \rfloor$$ (adding a *bias*)

- Compute $\lceil$ `x / 2^k` $\rceil$ as

$$\lfloor (x + 2^k - 1) / 2^k \rfloor$$

- In C:

```
(x + (1 << k) - 1) >> k
```