Computer Systems Organization

Topic 2

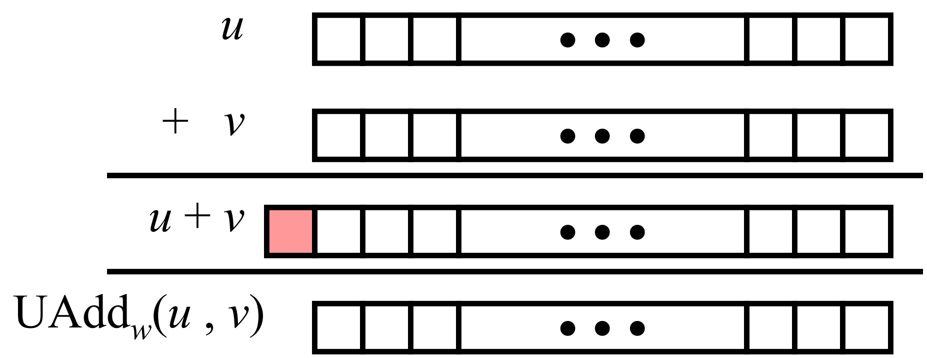
Based on chapter 2 from Computer Systems by Randal E. Bryant and David R. O’Hallaron

Unsigned Addition

Operands: w bits

True Sum: w+1 bits

Discard Carry: w bits



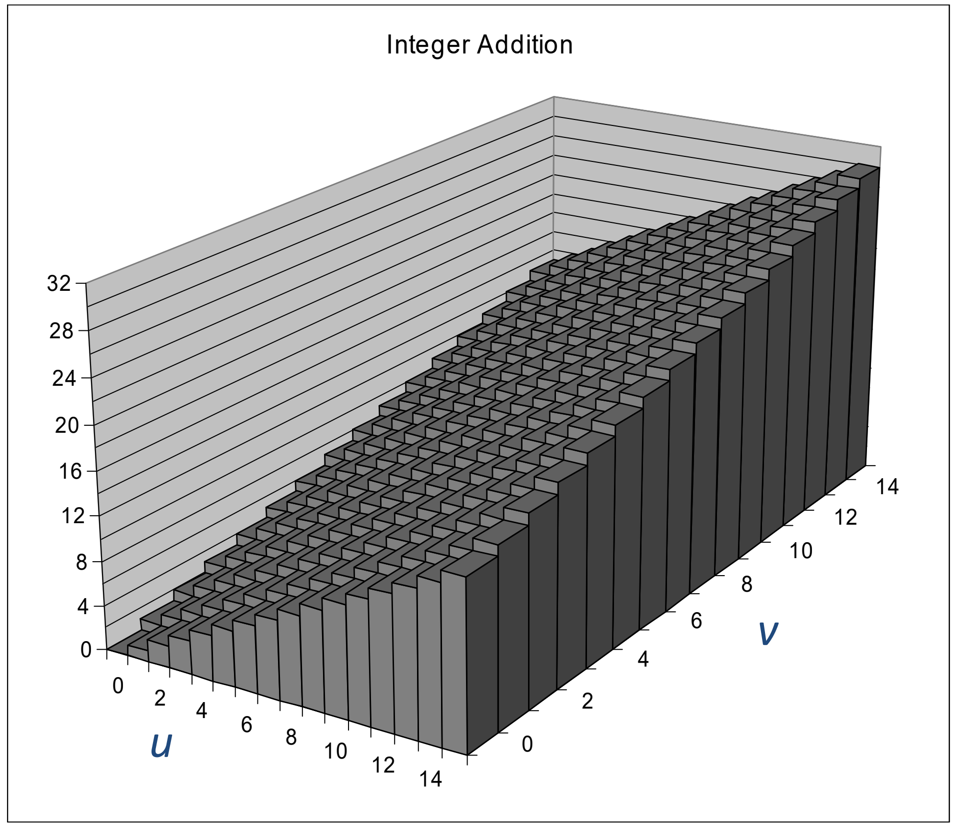
- Standard Addition Function
  - Ignores carry output
- Implements Modular Arithmetic

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

Visualizing (Mathematical) Integer Addition

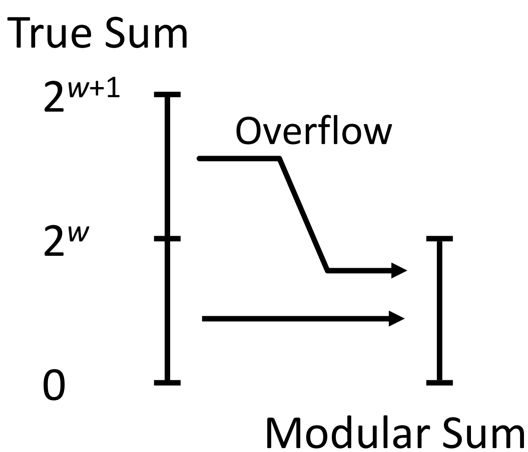
- Integer Addition
  - 4-bit integers *u*, *v*
  - Compute true sum $\text{Add}_4(u\ ,\ v)$
  - Values increase linearly with *u* and *v*
  - Forms planar surface

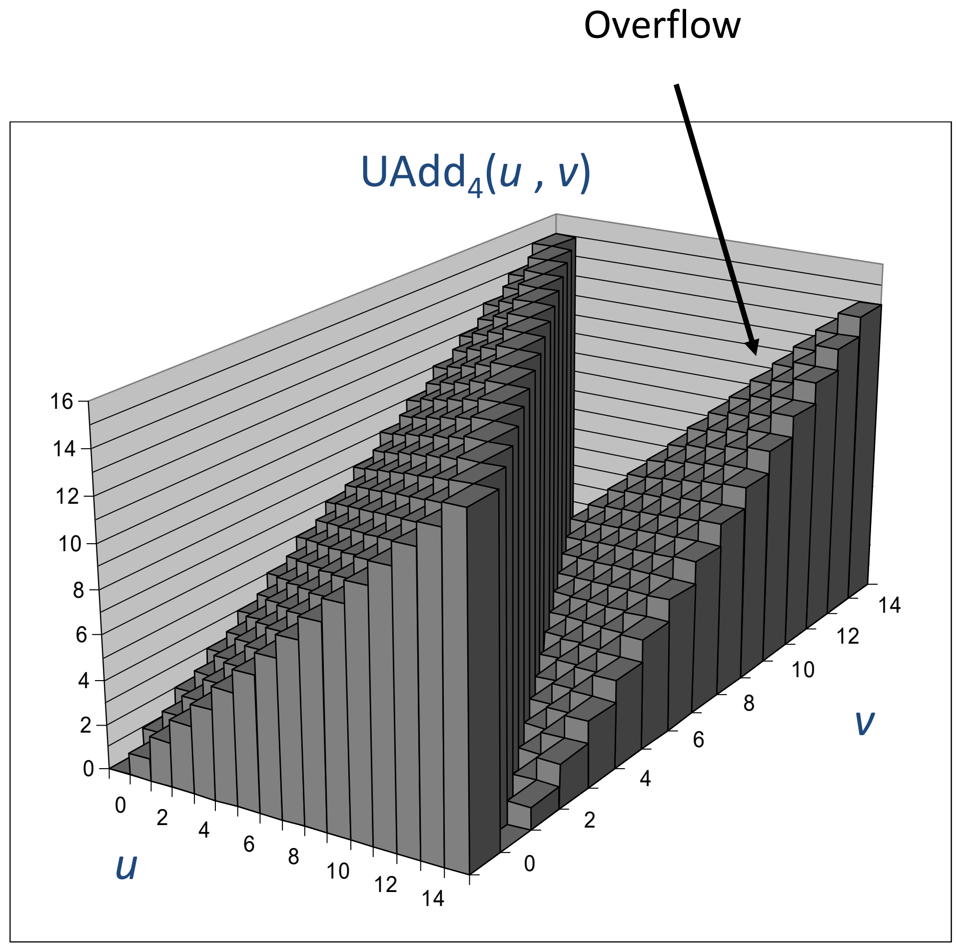
$\text{Add}_4(u\ ,\ v)$



Visualizing Unsigned Addition

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



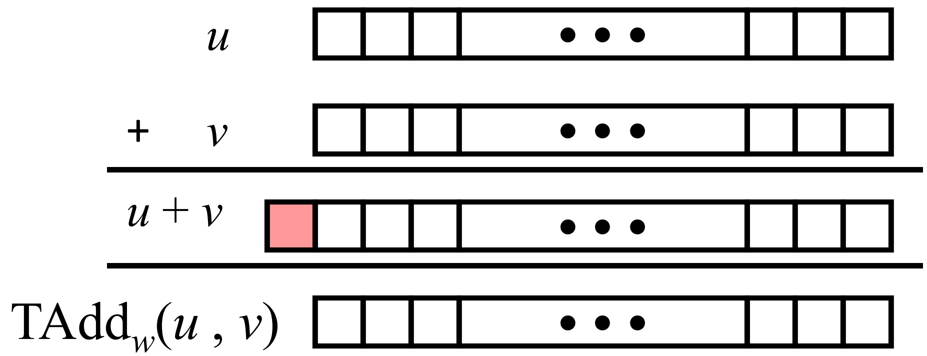


Two’s Complement Addition

Operands: *w* bits

True Sum: *w*+1 bits

Discard Carry: *w* bits



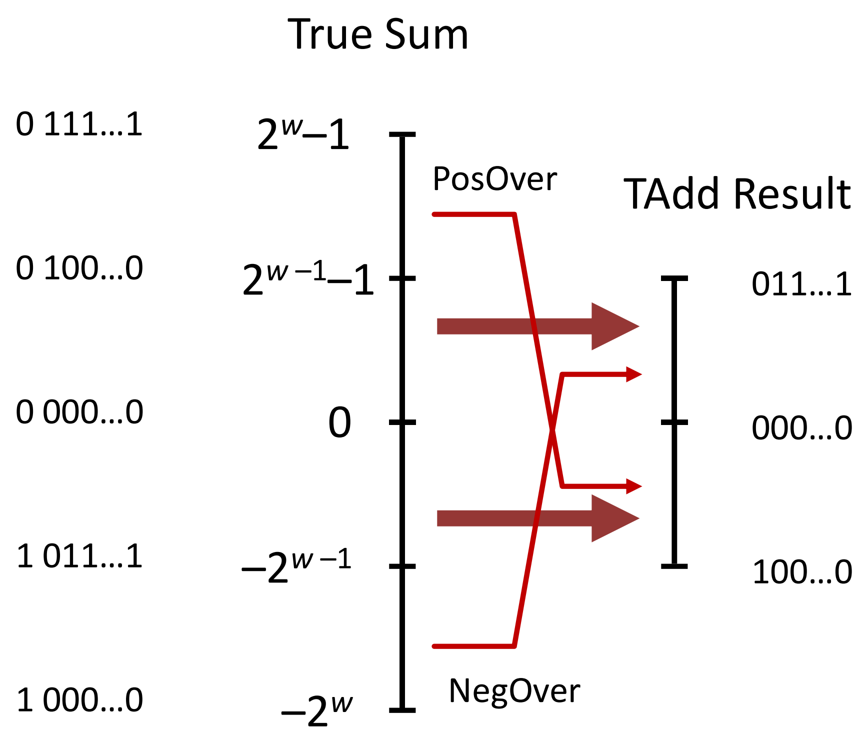
- TAdd and UAdd have Identical Bit-Level Behavior
  - Signed vs. unsigned addition in C:

```
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

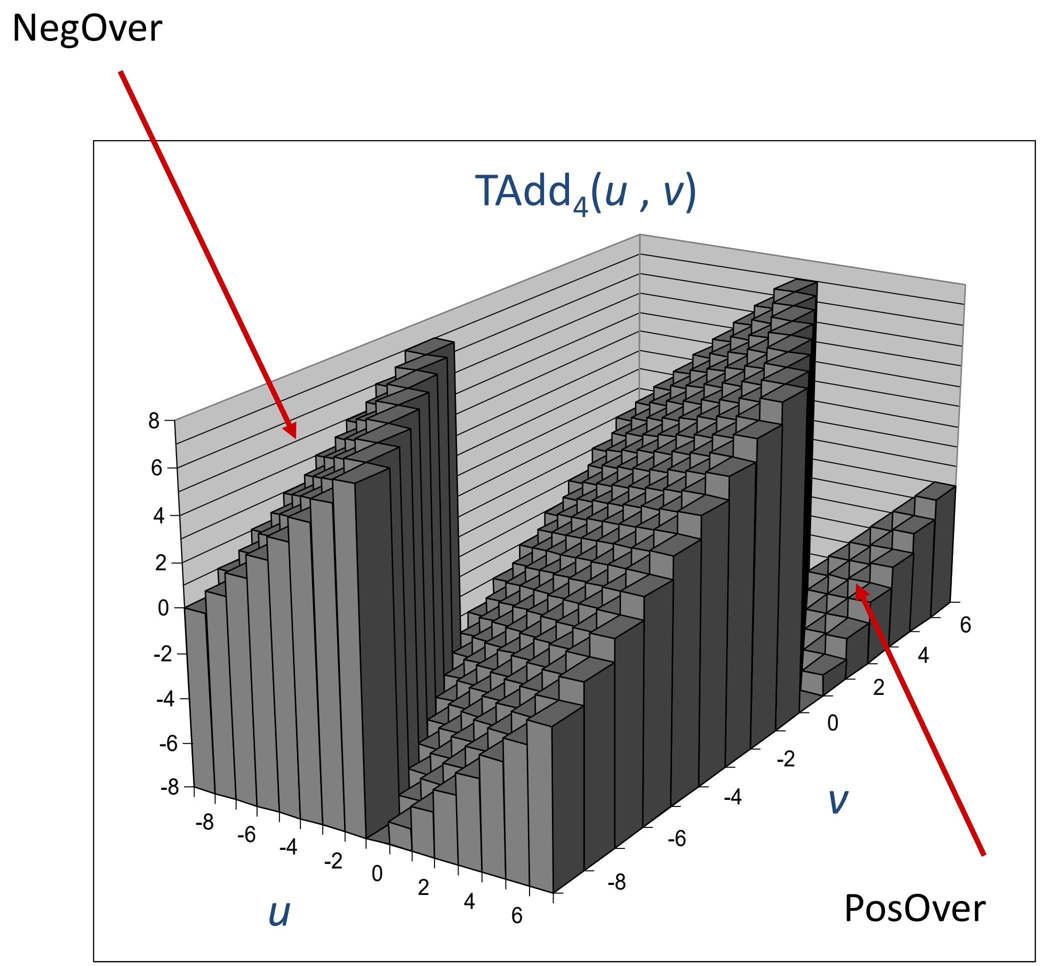


Two’s Complement Addition

- In summary, subtract 2^w if positive overflow
- Add 2^w if negative overflow
- No changes if 2^(w-1) <= sum < 2^(w-1)
- For w = 4 bits,
- -8 [1000] + -5 [1011] = -13 [10011] = 3 [0011]
- 5 [0101] + 5 [0101] = 10 [01010] = -6 [1010]

Visualizing 2's Complement Addition

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



Negation

- Complement the bits, increment the result by 1
- 0101 [5] → 1010 [-6] → 1011 [-5]
- 1000 [-8] → 0111 [7] → 1000 [-8]
- …

Multiplication

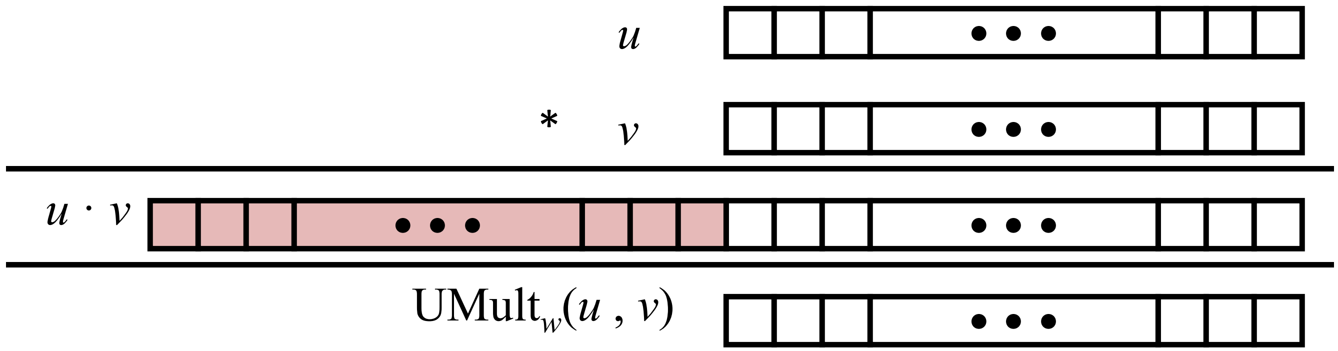
- Goal: Computing Product of w-bit numbers x, y
  - Either signed or unsigned
- But, exact results can be bigger than w bits
  - Unsigned: up to $2w$ bits
    - Result range: $0 \le x * y \le (2^w - 1)^2 = 2^{2w} - 2^{w+1} + 1$
  - Two's complement min (negative): Up to $2w$-1 bits
    - Result range: $x * y \ge (-2^{w-1})*(2^{w-1}-1) = -2^{2w-2} + 2^{w-1}$
  - Two's complement max (positive): Up to $2w$ bits, but only for $(TMin_w)^2$
    - Result range: $x * y \le (-2^{w-1})^2 = 2^{2w-2}$
- 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

True Product: 2**w* bits

Discard *w* bits: *w* bits



- Standard Multiplication Function
  - Ignores high order *w* bits
- Implements Modular Arithmetic

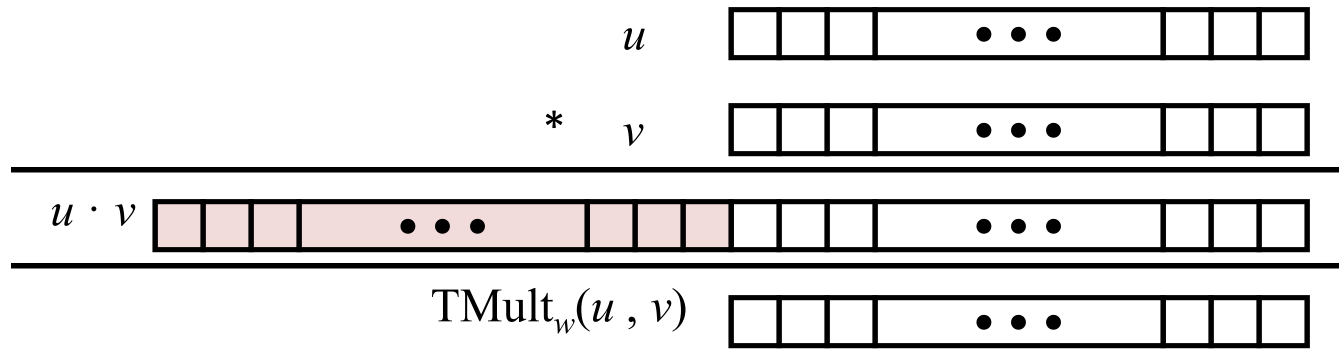
$\mathrm{UMult}_w(u, v) \quad = \quad u \cdot v \mod 2^w$

Signed Multiplication in C

Operands: *w* bits

True Product: 2**w* bits

Discard *w* bits: *w* bits



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

Example

- Unsigned: 5 [101] * 3 [011] = 15 [01111] → 7 [111] Truncated
- 101
- 011
- 101
- 101
- 000

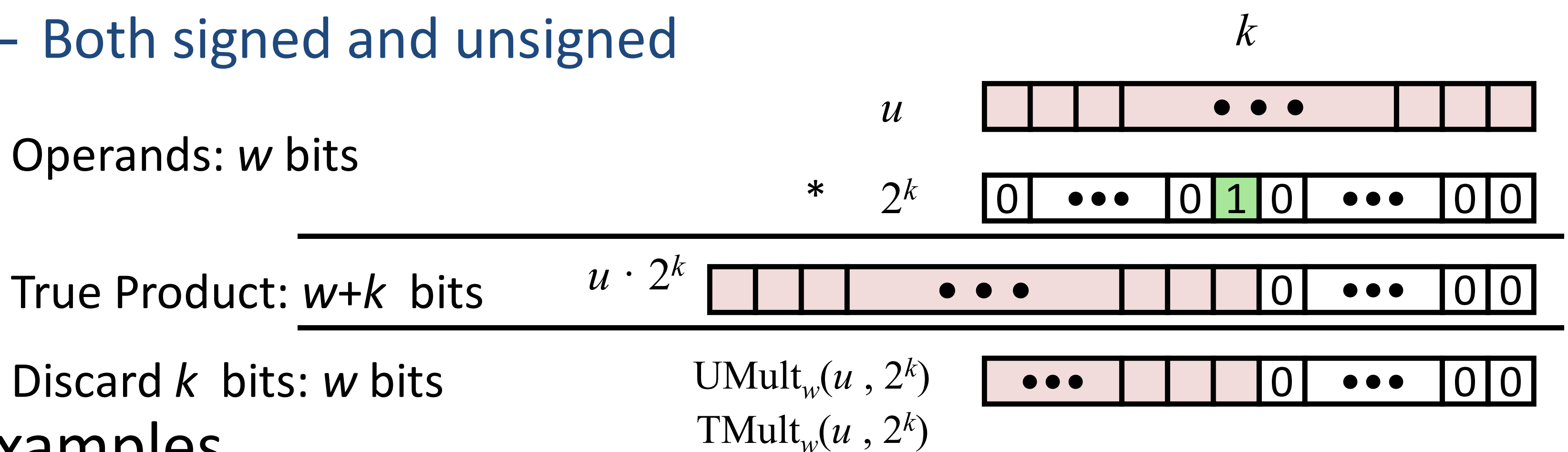
- 01111

Example

- Two's C: -3 [101] * 3 [011] = -9 [110111] → -1 [111] Truncated
- Need to sign extend and then multiply
- 111101
- 000011
- 111101
- 111101
- 000000
- 000000
- 000000
- 000000
- ----------------
- 000101**110111**

Power-of-2 Multiply with Shift

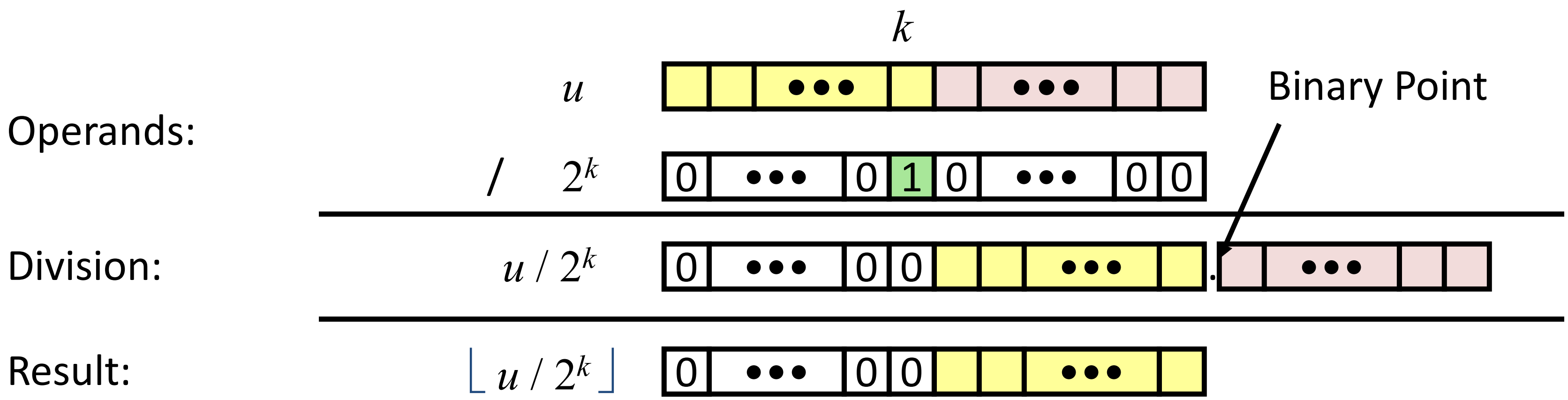
- Operation
  - **`u << k`** gives **`u * `** ***2^k***
  - Both signed and unsigned



- Examples
  - **`u << 3          ==  u * 8`**
  - **`(u << 5) - (u << 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 >> 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`

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)

Using 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];
```