integer arithmetic

and, xor, shifting → bit-wise

how to interpret as integer?

$B2U = \sum_{i=0}^{w-1} x_i \cdot 2^i$ unsigned

$B2T = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$ signed (non-negative & negative values)

most significant bit (leftmost) $w \rightarrow$ width

$B2T(1100_2) = -1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = -8 + 4 = -4$

if most significant is 1, it's negative

X	B2U(X)	B2T(X)
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	−8
1001	9	−7
1010	10	−6
1011	11	−5
1100	12	−4
1101	13	−3
1110	14	−2
1111	15	−1

16

only 1 value

0 has same representation

T_{max} $(2^{w-1} - 1)$ 011...1 max is when leading bit is 0

T_{min} $(-2^{w-1} - 1)$ 100...0

U_{max} $(2^w - 1)$

Range is asymmetric: $|T_{min}| = T_{max} + 1$

$U_{max} = |T_{min}| + T_{max} = 2T_{max} + 1$

both have same values OR are seperated by 2^w 16

same bit vector can be interpreted as either B2U or B2T

-5 → 5

1011 —not→ 0100 —+1→ 0101

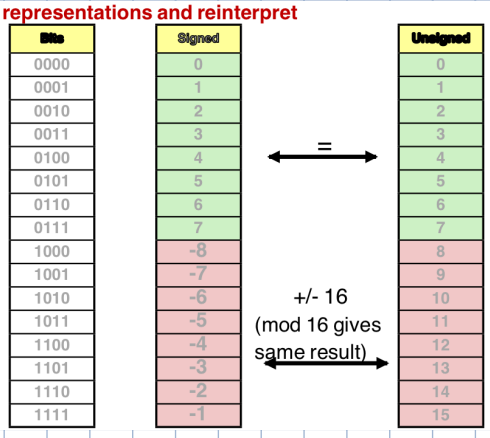
-5 → 5

$X + (\text{not } X) = 1$

Conversion & Casting

mapping doesn't change bit

Keep bit representations and reinterpret



casting

constants integers are signed, unless there's a U @ end

```
int tx, ty
unsigned ux, uy
tx = (int) ux        tx = ux      implicitly
uy = (unsigned) ty   uy = ty
```

simply copy bit to new variable, interpret differently

signed implicitly casted to unsigned when manipulating the 2

constant,	constant2	Relation	evaluation	
0 (0000)→0	0U (0000)→0	0 == 0	unsigned	
-1 (1111)→-1	0 (0000)→0	-1 < 0	signed	both are signed
-1 (1111)→15	0U (0000)→0	15 > 0	unsigned	

bit pattern is maintained, only reinterpreted

Expanding & truncating

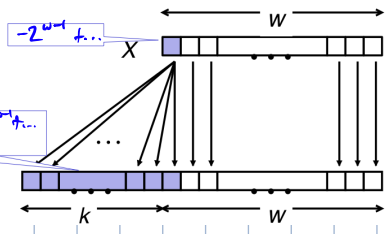
given w-bit signed integer X
convert to k+w bit integer X' w/same value

Rule:

Make *k* copies of sign bit:

$X' = x_{w-1}, \ldots, x_{w-1}, x_{w-1}, x_{w-2}, \ldots, x_0$

k copies of the most significant bit



Why they are the same value?

$-2^{w+k-1} + 2^{w+k-2} + \ldots + 2^{w-1} + \ldots$
$= -2^{w-1} + \ldots$

s154

unsigned? add 0's

2 byte → 4byte
16 bit → 32 bit

```
short int x =  15213;
int      ix = (int) x;
short int y = -15213;
int      iy = (int) y;
```

	Decimal	Hex	Binary
x	15213	3B 6D	00111011 01101101
ix	15213	00 00 3B 6D	00000000 00000000 00111011 01101101
y	-15213	C4 93	11000100 10010011
iy	-15213	FF FF C4 93	11111111 11111111 11000100 10010011

copy 0 16 times

copy 1 16 times

kinda like right shift

value of variable doesn't change

casting keeps bit vector, change value
extension keeps value, changes bit vector

truncating. get rid of k bits, redundant. keep leftmost bits

unsigned
unsigned int X to k bits equivalent to X mod (2^k)

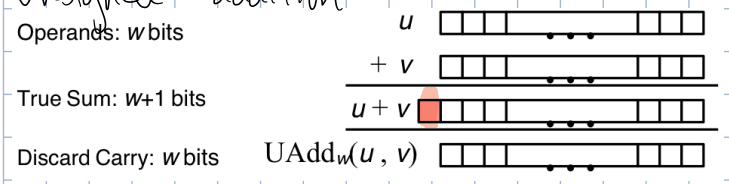
signed
same, X mod (2^k), then change to signed

Arithmetic

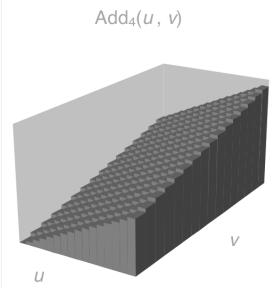
Negative #'s: 2's complement is cool

have fixed width bit, might cause overflow

Unsigned addition



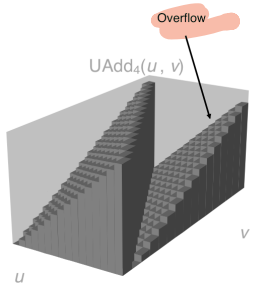
get rid of carry bit similar to truncation

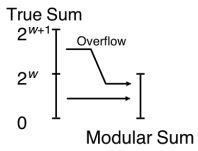


"True"

need 5 bit output

bound to 4 bits





Signed addition

same bit behavior, different interpretation

(-8) + (-5)

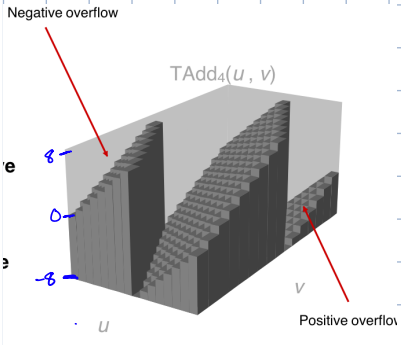
```
 1000   (-8)
 1011   (-5)
10011   (3)
```

negative overflow, value is bumped up
should be -13, but -8 is min

(5) + (5)

```
 0101   5
 0101   5
 1010   -6
```

positive overflow, value is bumped down
should be 10, but 7 is max



range -8 to 7

if sum $\geq 2^{w-1}$ → becomes negative

if sum $< -2^{w-1}$ → becomes positive