# Computer System Organization (CS2.201 )

## Lecture # 02-04

## Integer Representation

Avinash Sharma

Center for Visual Information Technology (CVIT),

IIIT Hyderabad

Slide content acknowledgment: Dr. Suresh Purini & other public sources

# Data Representation

- Binary Representation
- Virtual Address Space
- Word size (nominal size of pointer data)
  - 32 bit (4.2^30 ~ 4 GB)
  - 64 bit (16.2^60 ~16 EB)

1 EB = 10^9 GB
1 GB = 10^9 B = 10^6 MB

| C declaration | 32-bit | 64-bit |
|---|---|---|
| `char` | 1 | 1 |
| `short int` | 2 | 2 |
| `int` | 4 | 4 |
| `long int` | 4 | 8 |
| `long long int` | 8 | 8 |
| `char *` | 4 | 8 |
| `float` | 4 | 4 |
| `double` | 8 | 8 |

In Bytes

# Data Representation

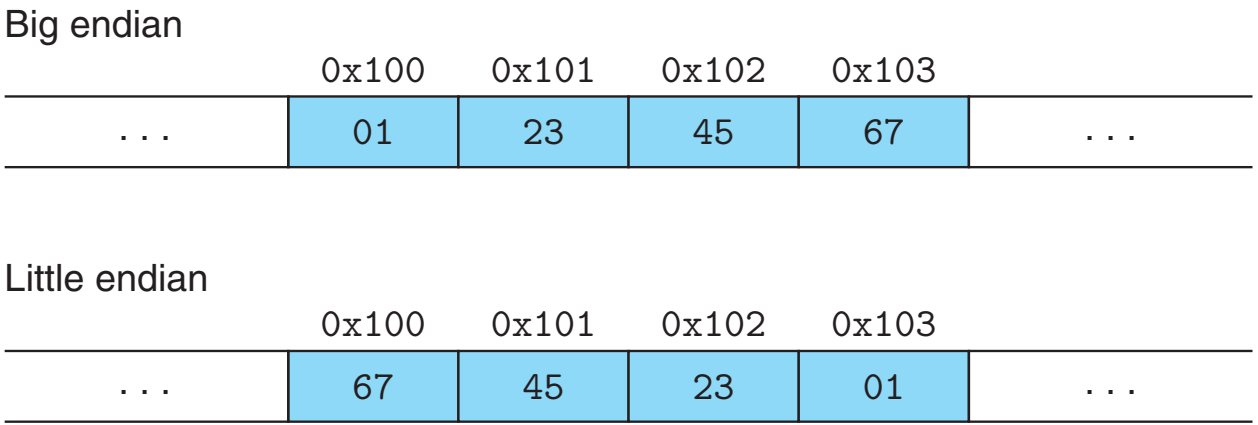
- Binary, Decimal and Hexadecimal representation of integers.

| Hex digit | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Decimal Value | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Binary Value | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 |

| Hex digit | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|
| Decimal Value | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Binary Value | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |

# Storage Format

- Word 0x01234567 stored at address 0x100



# Boolean Algebra

- NOT, AND, OR and EXOR

| ~ | |
|---|---|
| 0 | 1 |
| 1 | 0 |

| & | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

| \| | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 1 |

| ^ | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

- Bitwise vector operations

```
  0110        0110        0110
& 1100      | 1100      ^ 1100      ~ 1100
------      ------      ------      ------
  0100        1110        1010        0011
```

# Logical Operations in C

- NOT, AND, OR

| Expression | Result |
|---|---|
| `!0x41` | `0x00` |
| `!0x00` | `0x01` |
| `!!0x41` | `0x01` |
| `0x69 && 0x55` | `0x01` |
| `0x69 \|\| 0x55` | `0x01` |

# Shift Operations in C

- Left, Right (logical), Right (Arithmetic)

| Operation | Values | |
|---|---|---|
| Argument `x` | [01100011] | [10010101] |
| `x << 4` | [0011*0000*] | [0101*0000*] |
| `x >> 4` (logical) | [*0000*0110] | [*0000*1001] |
| `x >> 4` (arithmetic) | [*0000*0110] | [*1111*1001] |

# Integer Representation

- Range of Signed v/s Unsigned Integer on 32 bit machine

| C data type | Minimum | Maximum |
|---|---|---|
| `char` | −128 | 127 |
| `unsigned char` | 0 | 255 |
| `short [int]` | −32,768 | 32,767 |
| `unsigned short [int]` | 0 | 65,535 |
| `int` | −2,147,483,648 | 2,147,483,647 |
| `unsigned [int]` | 0 | 4,294,967,295 |
| `long [int]` | −2,147,483,648 | 2,147,483,647 |
| `unsigned long [int]` | 0 | 4,294,967,295 |
| `long long [int]` | −9,223,372,036,854,775,808 | 9,223,372,036,854,775,807 |
| `unsigned long long [int]` | 0 | 18,446,744,073,709,551,615 |

# Integer Representation

- Range of Signed v/s Unsigned Integer on 64 bit machine

| C data type | Minimum | Maximum |
|---|---|---|
| `char` | −128 | 127 |
| `unsigned char` | 0 | 255 |
| `short [int]` | −32,768 | 32,767 |
| `unsigned short [int]` | 0 | 65,535 |
| `int` | −2,147,483,648 | 2,147,483,647 |
| `unsigned [int]` | 0 | 4,294,967,295 |
| `long [int]` | −9,223,372,036,854,775,808 | 9,223,372,036,854,775,807 |
| `unsigned long [int]` | 0 | 18,446,744,073,709,551,615 |
| `long long [int]` | −9,223,372,036,854,775,808 | 9,223,372,036,854,775,807 |
| `unsigned long long [int]` | 0 | 18,446,744,073,709,551,615 |

# Unsigned and Signed Integers

Signed or Unsigned Integers – Doesn't Matter! They still need to be represented as a sequence of bits.

- Consider sequences of w-bits. ( Ex: 0101 is a 4-bit sequence)
- There are $2^w$ possible such sequences.
- It is up to us to interpret each of these sequences in the way we would like to.
- In other words we can associate decimal values to these n-bit strings according to our convenience.

# Unsigned Integers

Unsigned Integer Interpretation for n-bit strings:

$$B2U_w : \{ 0, 1 \}^w \rightarrow \{ 0, \ldots , 2^w - 1 \}$$

$$B2U_w(\vec{x}) \doteq \sum_{i=0}^{w-1} x_i 2^i$$

Ex: $B2U_4(0101) = 5$, $B2U_4(1001) = 9$, …

- $U_{w\text{-}max} = B2U_w(111..1) = 2^w - 1$ (Max Integer Value)
- $U_{w\text{-}min} = B2U_w(000….0) = 0$ (Min Integer Value)

# Signed Integers – Sign-Magnitude Representation

Sign-Magnitude Integer Interpretation for w-bit strings:

$$B2S_w : \{ 0, 1 \}^w \rightarrow \{ -2^{w-1}+1, \ldots 0, \ldots , 2^{w-1}-1\}$$

$(w-1)^{th}$ bit is the sign-

$$B2S_w(\vec{x}) \doteq (-1)^{x_{w-1}} \cdot \left( \sum_{i=0}^{w-2} x_i 2^i \right)$$

Ex: $B2S_4(0101) = 5$, $B2S_4(1001) = -1$,

$B2S_4(0000) = 0$, $B2S_4(1000) = -0$

- $S_{w\text{-}max} = B2S_w(011..1) = 2^{w-1} - 1$ (Max Integer Value)
- $S_{w\text{-}min} = B2S_w(111....1) = -2^{w-1} + 1$ (Min Integer Value)

# Signed Integers – Two's Complement Representation

Two's complement Integer Interpretation for w-bit strings:

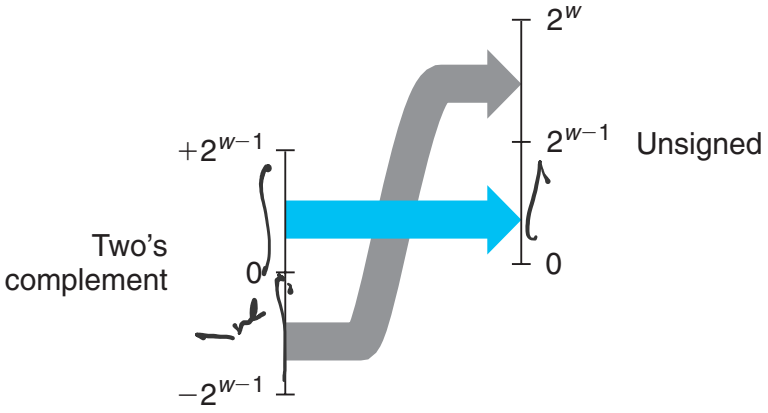
$B2T_w : \{ 0, 1 \}^w \rightarrow \{ -2^{w-1}, \ldots 0, \ldots , 2^{w-1}-1\}$

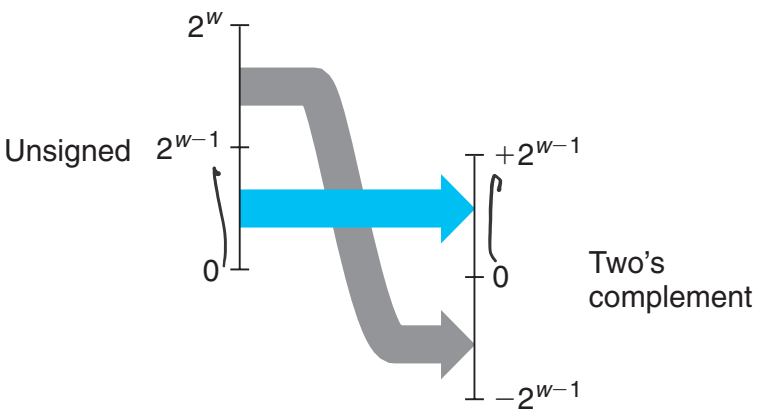
$$B2T_w(\vec{x}) \doteq -x_{w-1}2^{w-1} + \sum_{i=0}^{w-2} x_i 2^i$$

Ex: $B2T_4(0101) = 5$, $B2T_4(1001) = -7$,
$B2OT_4(0000) = 0$, $B2S_4(1111) = -1$

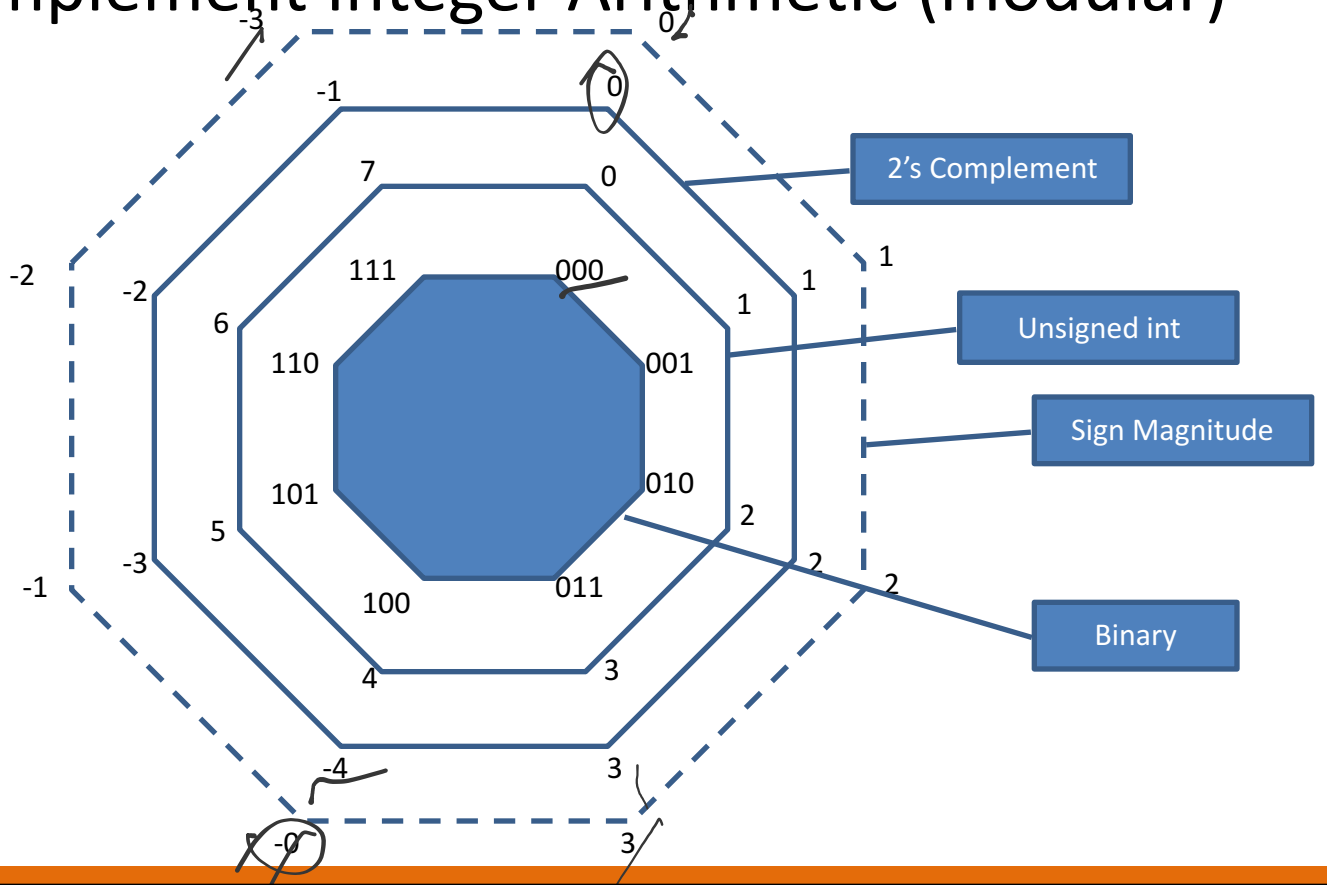
- $T_{w\text{-}max} = B2T_w(011..1) = 2^{w-1} - 1$ (Max Integer Value)
- $T_{w\text{-}min} = B2T_w(100....0) = -2^{w-1}$ (Min Integer Value)

# Unsigned and Two's Complement Conversion



# Isomorphism Between Binary, Unsigned and Two's Complement Integer Arithmetic (modular)

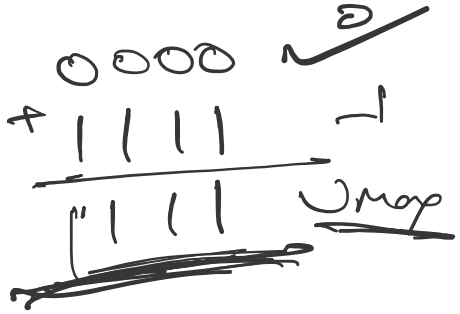


# Isomorphism Between a Binary, Unsigned and Two's Complement Integer Arithmetic (modular)

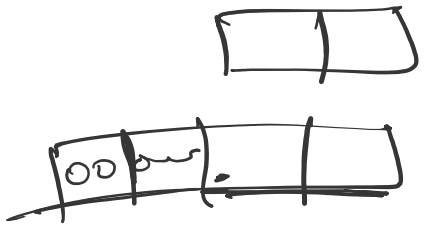
| Row ID | Binary | Unsigned | Signed Magn. | 2's Complement |
|---|---|---|---|---|
| $R_0$ | 000 | 0 | 0 | 0 |
| $R_1$ | 001 | 1 | 1 | 1 |
| $R_2$ | 010 | 2 | 2 | 2 |
| $R_3$ | 011 | 3 | 3 | 3 |
| $R_4$ | 100 | 4 | -0 | -4 |
| $R_5$ | 101 | 5 | -1 | -3 |
| $R_6$ | 110 | 6 | -2 | -2 |
| $R_7$ | 111 | 7 | -3 | -1 |

# Automatic Conversion Risks

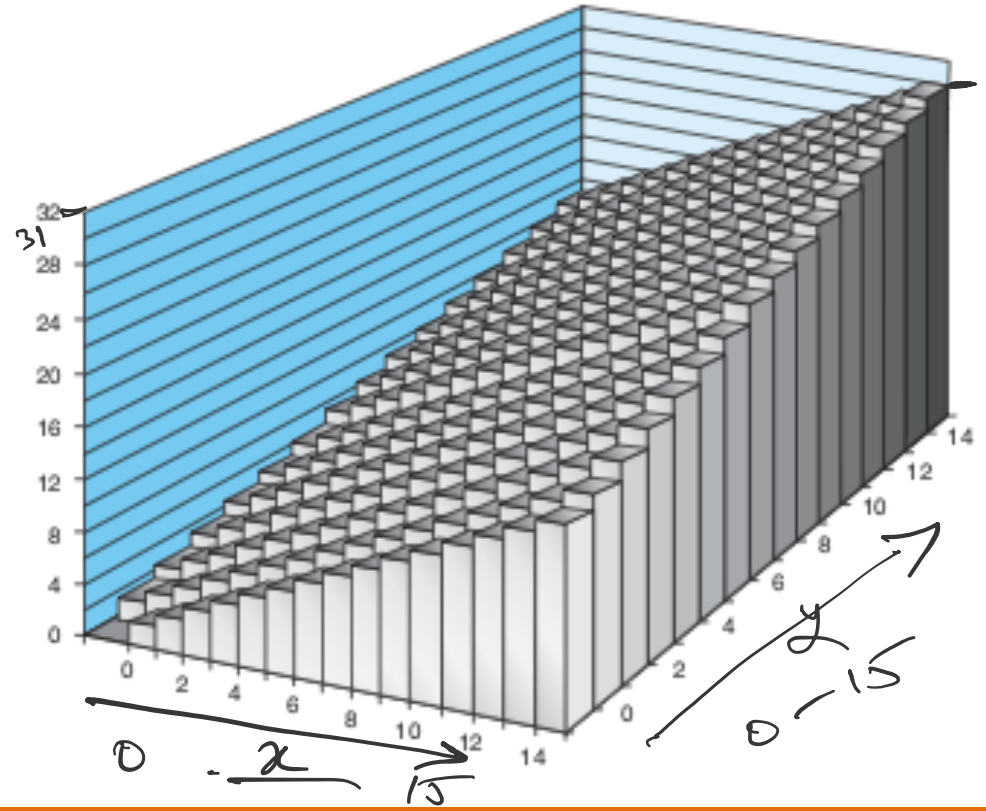
```
float sum_elements(float a[], unsigned length) {
    int i;
    float result = 0;

    for (i = 0; i <= length-1; i++)
        result += a[i];
    return result;
}
```

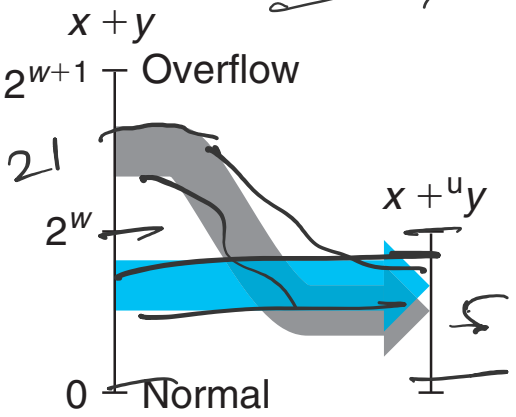


# Copying to Extended size datatype

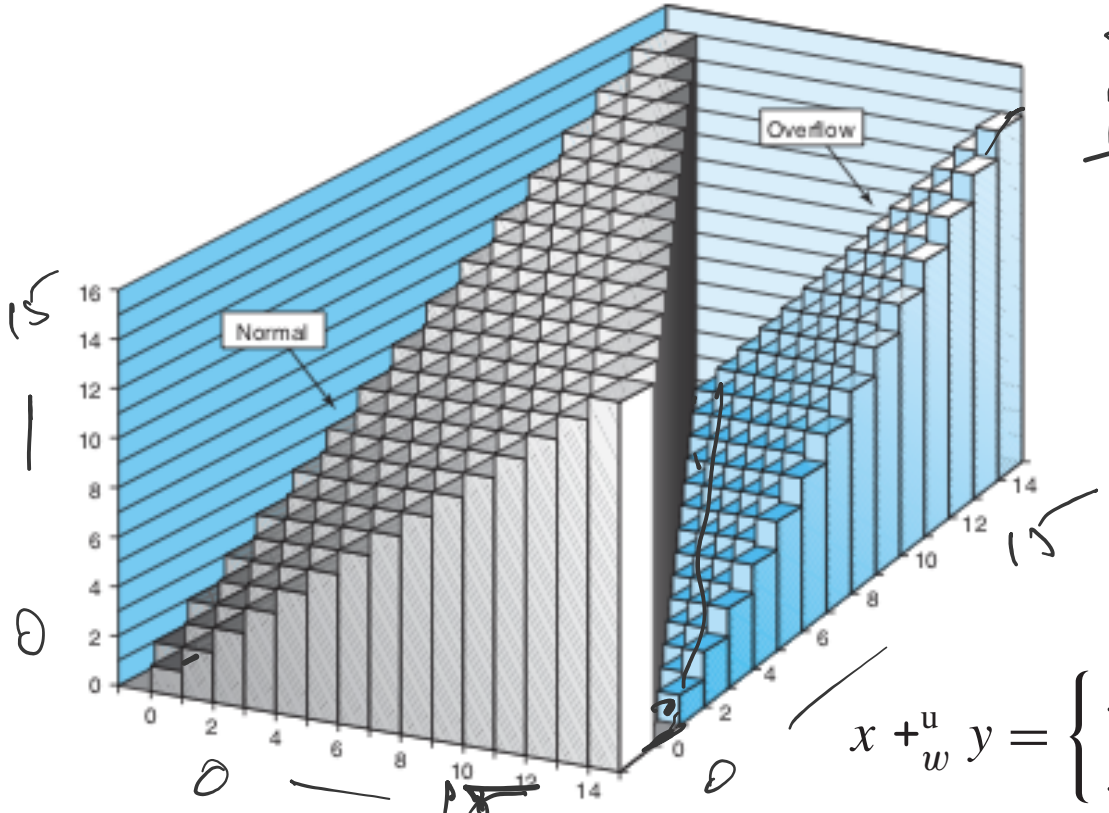


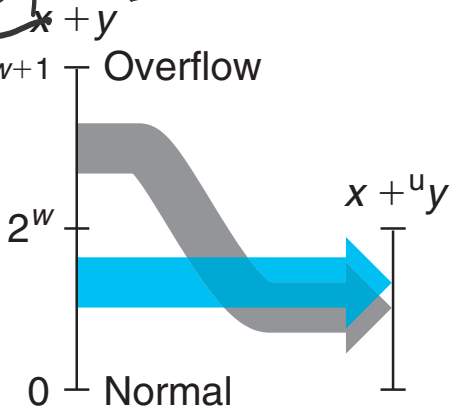
# Integer Addition



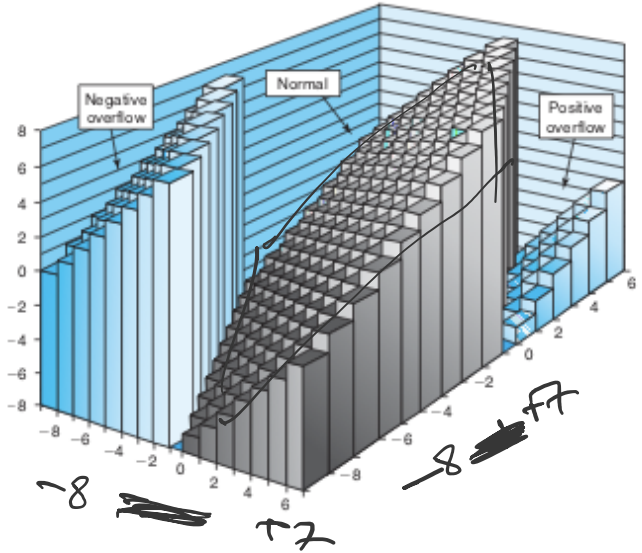
# Unsigned Integer Addition

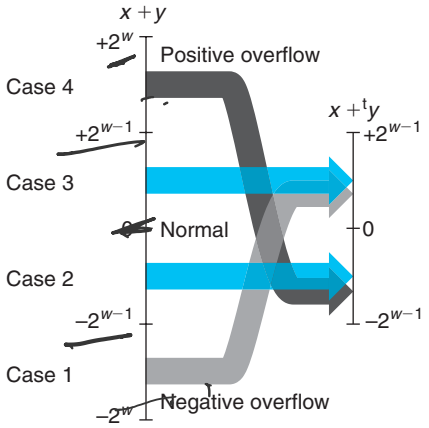



$$x +_w^{\mathrm{u}} y = \begin{cases} x + y, & x + y < 2^w \\ x + y - 2^w, & 2^w \le x + y < 2^{w+1} \end{cases}$$

# Signed Integer (2's Compl.) Addition




$$x +_w^t y = \begin{cases} x + y - 2^w, & 2^{w-1} \le x + y \quad \text{Positive overflow} \\ x + y, & -2^{w-1} \le x + y < 2^{w-1} \quad \text{Normal} \\ x + y + 2^w, & x + y < -2^{w-1} \quad \text{Negative overflow} \end{cases}$$

# Unsigned and Signed (2's Complement) Integer Addition

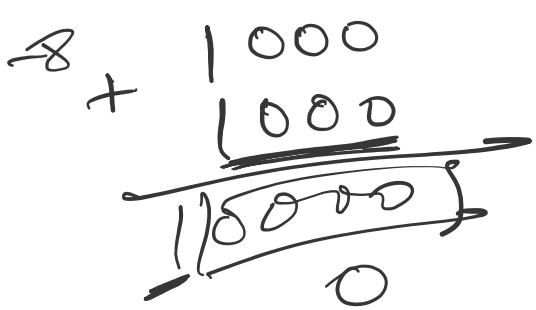
1. $R_2 + R_4 = R_6$
   1. 010 + 100 = 110
   2. 2 + 4 = 6
   3. 2 + (-4) = -2
2. $R_6 + R_7 = R_5$
   1. 110 + 111 = 1101
   2. 6 + 7 = 5 (13 mod 8)
   3. -2 + (-1) = -3
3. $R_2 + R_3 = R_5$
   1. 010 + 011 = 101
   2. 2 + 3 = 5
   3. 2 + 3 = -3

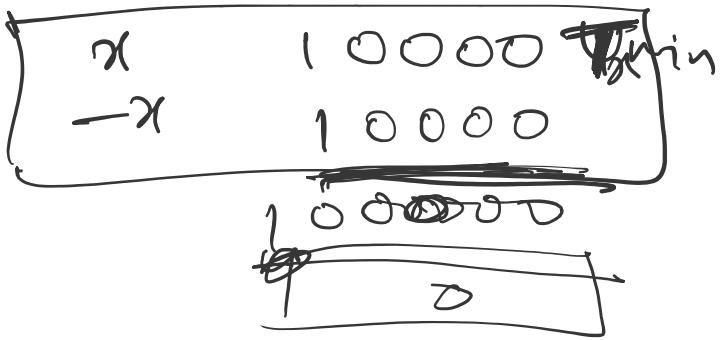
| Row ID | Binary | Unsigned | 2's Complement |
|---|---|---|---|
| $R_0$ | 000 | 0 | 0 |
| $R_1$ | 001 | 1 | 1 |
| $R_2$ | 010 | 2 | 2 |
| $R_3$ | 011 | 3 | 3 |
| $R_4$ | 100 | 4 | -4 |
| $R_5$ | 101 | 5 | -3 |
| $R_6$ | 110 | 6 | -2 |
| $R_7$ | 111 | 7 | -1 |

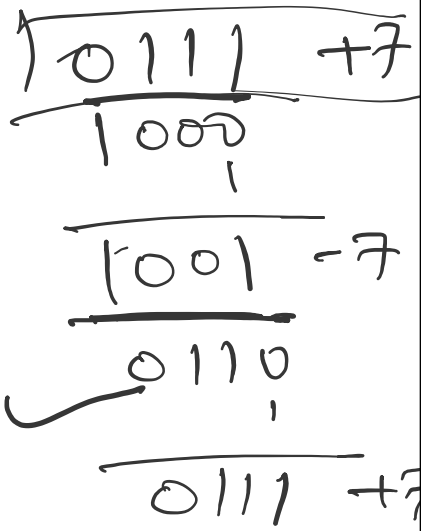
# Signed Integer (2's Compl.) Addition

| $x$ | $y$ | $x + y$ | $x +_4^t y$ | Case |
|---|---|---|---|---|
| −8 | −5 | −13 | 3 | 1 |
| [1000] | [1011] | [10011] | [0011] | |
| −8 | −8 | −16 | 0 | 1 |
| [1000] | [1000] | [10000] | [0000] | |
| −8 | 5 | −3 | −3 | 2 |
| [1000] | [0101] | [11101] | [1101] | |
| 2 | 5 | 7 | 7 | 3 |
| [0010] | [0101] | [00111] | [0111] | |
| 5 | 5 | 10 | −6 | 4 |
| [0101] | [0101] | [01010] | [1010] | |

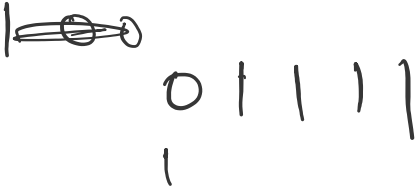
# Signed Integer (2's Compl.) Negation







$$-_w^t x = \begin{cases} -2^{w-1}, & x = -2^{w-1} \\ -x, & x > -2^{w-1} \end{cases}$$



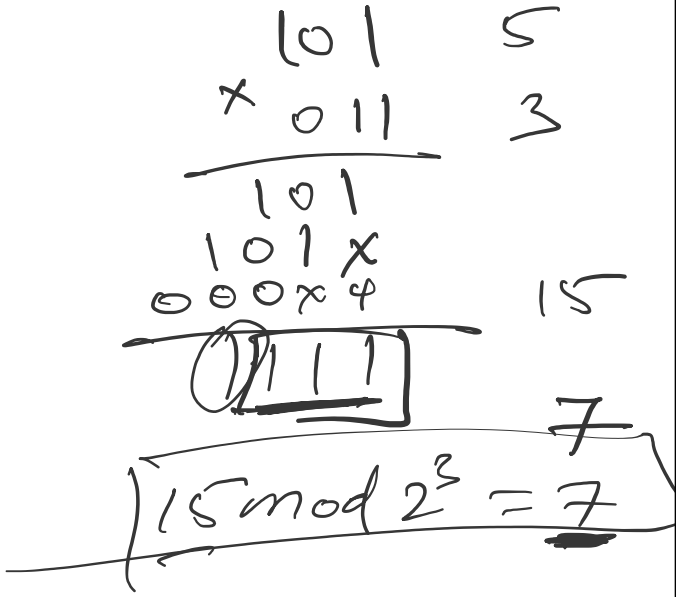
# Unsigned Integer Multiplication

Input: $0 \leq x, y \leq 2^w - 1$

Max Output: $(2^w - 1)^2 = 2^{2w} - 2^{w+1} + 1$

$0 \leq x, y, \leq (2^w - 1)$

$x *_w^u y = (x \cdot y) \bmod 2^w$



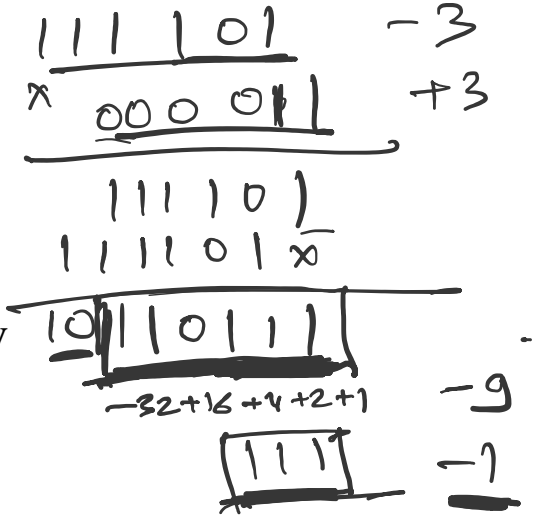
# Signed Int (2's Compl.) Multiplication

Input: $-2^{w-1} \le x, y \le 2^{w-1} - 1$

Max Output Range: $-2^{w-1} \cdot (2^{w-1} - 1) = -2^{2w-2} + 2^{w-1}$ and $-2^{w-1} \cdot -2^{w-1} = 2^{2w-2}$

$$x *_w^t y = U2T_w((x \cdot y) \bmod 2^w)$$

This implies that the machine can use a single type of multiply instruction to multiply both signed and unsigned integers.

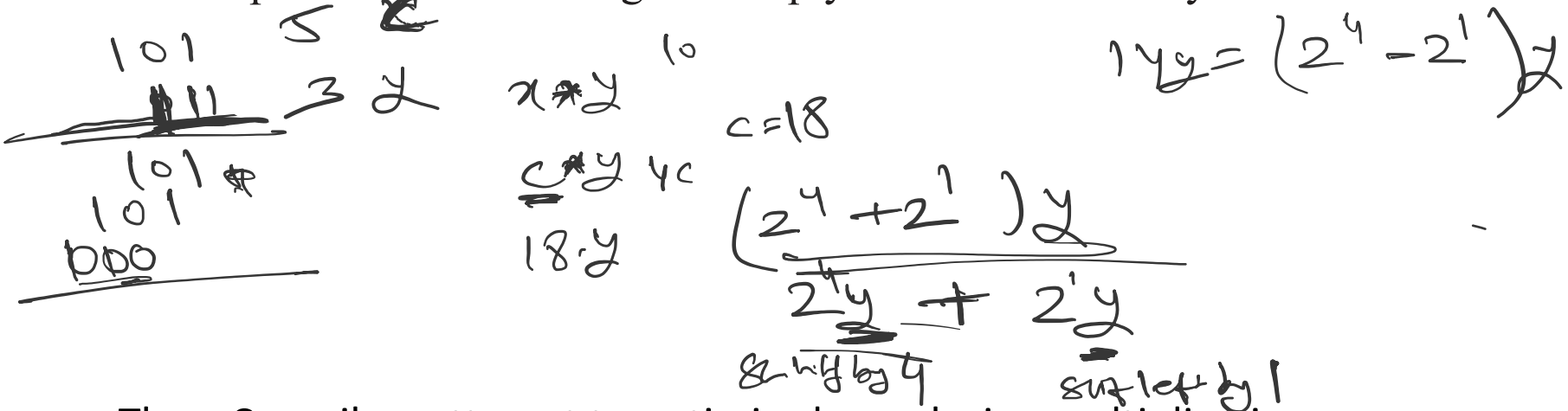


# Signed Int (2's Compl.) Multiplication

| Mode | $x$ | | $y$ | | $x \cdot y$ | | Truncated $x \cdot y$ | |
|---|---|---|---|---|---|---|---|---|
| Unsigned | 5 | [101] | 3 | [011] | 15 | [001111] | 7 | [111] |
| Two's comp. | −3 | [101] | 3 | [011] | −9 | [110111] | −1 | [111] |
| Unsigned | 4 | [100] | 7 | [111] | 28 | [011100] | 4 | [100] |
| Two's comp. | −4 | [100] | −1 | [111] | 4 | [000100] | −4 | [100] |
| Unsigned | 3 | [011] | 3 | [011] | 9 | [001001] | 1 | [001] |
| Two's comp. | 3 | [011] | 3 | [011] | 9 | [001001] | 1 | [001] |

# Multiplication by Constant

- The implementation of integer multiply instruction is fairly slow.




- Thus, Compilers attempt to optimize by replacing multiplications by constant factors with combinations of shift and addition operations.

# Signed Integers – One's Complement Representation

One's complement Integer Interpretation for w-bit strings:

$B2O_w : \{0, 1\}^w \rightarrow \{(-2^{w-1})+1, \ldots 0, \ldots, (2^{w-1}-1)\}$

$$B2O_w(\vec{x}) \doteq -x_{w-1}(2^{w-1}-1) + \sum_{i=0}^{w-2} x_i 2^i$$

Ex: $B2O_4(0101) = 5$, $B2S_4(1001) = -6$,
$B2O_4(0000) = 0$, $B2S_4(1111) = -0$

- $O_{w\text{-}max} = B2O_w(011..1) = 2^{w-1} - 1$ (Max Integer Value)
- $O_{w\text{-}min} = B2O_w(100....0) = -2^{w-1} + 1$ (Min Integer Value)

0 0101 +5
1's Com 1010 -5
2's Com 1011 -5

2's 1's
7 7
-8 -7

# Isomorphism Between ABinary, Unsigned and Two's Complement Integer Arithmetic (modular)

| Row ID | Binary | Unsigned | Signed Magn. | 1's Complement | 2's Complement |
|---|---|---|---|---|---|
| $R_0$ | 000 | 0 | 0 | 0 | 0 |
| $R_1$ | 001 | 1 | 1 | 1 | 1 |
| $R_2$ | 010 | 2 | 2 | 2 | 2 |
| $R_3$ | 011 | 3 | 3 | 3 | 3 |
| $R_4$ | 100 | 4 | -0 | -3 | -4 |
| $R_5$ | 101 | 5 | -1 | -2 | -3 |
| $R_6$ | 110 | 6 | -2 | -1 | -2 |
| $R_7$ | 111 | 7 | -3 | -0 | -1 |

# Converting unsigned to 1’s and 2’s complement