DISCUSSION – WEEK 1

TA : Mathanky
LA : Hannah

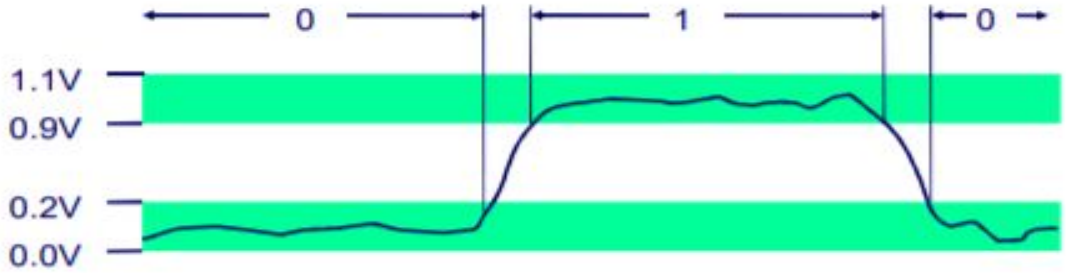
REVIEW OF WEEK 1

- Bits and Bytes
- Integers

BITS AND BYTES

Everything is bits

- **Each bit is 0 or 1**
- **By encoding/interpreting sets of bits in various ways**
  - Computers determine what to do (instructions)
  - … and represent and manipulate numbers, sets, strings, etc…
- **Why bits? Electronic Implementation**
  - Easy to store with bistable elements
  - Reliably transmitted on noisy and inaccurate wires



Encoding Byte Values

- **Byte = 8 bits**
  - Binary 00000000_2 to 11111111_2
  - Decimal: 0_{10} to 255_{10}
  - Hexadecimal 00_{16} to FF_{16}
    - Base 16 number representation
    - Use characters '0' to '9' and 'A' to 'F'
    - Write $FA1D37B_{16}$ in C as
      - 0xFA1D37B
      - 0xfa1d37b

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

One byte is represented by 2 hexadecimal characters

General Boolean Algebra

Truth Tables

AND

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

OR

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

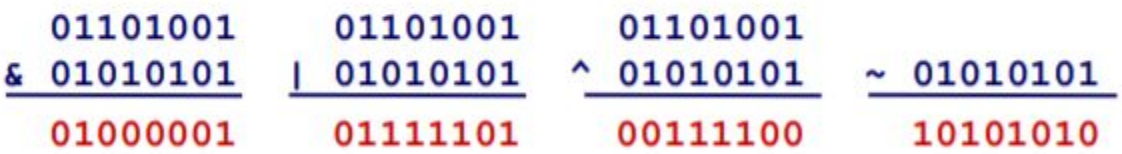
NOT

x	x'
0	1
1	0

XOR?

General Boolean Algebras

- **Operate on Bit Vectors**
  - Operations applied bitwise



- **All of the Properties of Boolean Algebra Apply**

Perform the following BitWise operations :

1. 7 & 8
2. 12 | 11
3. 5 ^ 7
4. ~ 15

Representing and Manipulating Sets

- Consider the set S = {0,1,2,3,4,5,6,7}. Let A = {0,2,3,4,7} be a subset of S. Let B={0,1,4,5,6} be a subset of S.
- Represent A and B using 8 bits (1 byte).
- What is A & B? What does this mean, intuitively?
- What is A | B? What does this mean, intuitively?
- What is A ^ B? What does this mean, intuitively?
- What is ~A? What does this mean, intuitively?

What is the difference between Bitwise Operators and Logical Operators?

Shift Operations

- **Left Shift: `x << y`**
  - Shift bit-vector `x` left `y` positions
    - Throw away extra bits on left
  - Fill with 0's on right
- **Right Shift: `x >> y`**
  - Shift bit-vector `x` right `y` positions
    - Throw away extra bits on right
  - Logical shift
    - Fill with 0's on left
  - Arithmetic shift
    - Replicate most significant bit on left
- **Undefined Behavior**
  - Shift amount < 0 or ≥ word size

Argument x	01100010
<< 3	00010*000*
Log. >> 2	*00*011000
Arith. >> 2	*00*011000

Argument x	10100010
<< 3	00010*000*
Log. >> 2	*00*101000
Arith. >> 2	*11*101000

INTEGERS

Integers can be **signed** or **unsigned**.

Unsigned

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

Two's Complement

$$B2T(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$

Sign Bit

```
short int x =  15213;
short int y = -15213;
```

- **C `short` 2 bytes long**

	Decimal	Hex	Binary
x	15213	3B 6D	00111011 01101101
y	-15213	C4 93	11000100 10010011

- **Sign Bit**
  - For 2's complement, most significant bit indicates sign
    - 0 for nonnegative
    - 1 for negative

Two-complement Encoding Example (Cont.)

```
x =      15213: 00111011 01101101
y =     -15213: 11000100 10010011
```

Weight	15213		-15213	
1	1	1	1	1
2	0	0	1	2
4	1	4	0	0
8	1	8	0	0
16	0	0	1	16
32	1	32	0	0
64	1	64	0	0
128	0	0	1	128
256	1	256	0	0
512	1	512	0	0
1024	0	0	1	1024
2048	1	2048	0	0
4096	1	4096	0	0
8192	1	8192	0	0
16384	0	0	1	16384
-32768	0	0	1	-32768
Sum		**15213**		**-15213**

Numeric Ranges

- **Unsigned Values**
  - *UMin* = 0
    000…0
  - *UMax* = $2^w - 1$
    111…1
- **Two's Complement Values**
  - *TMin* = -2^{w-1}
    100…0
  - *TMax* = $2^{w-1} - 1$
    011…1
- **Other Values**
  - Minus 1
    111…1

Values for *W* = 16

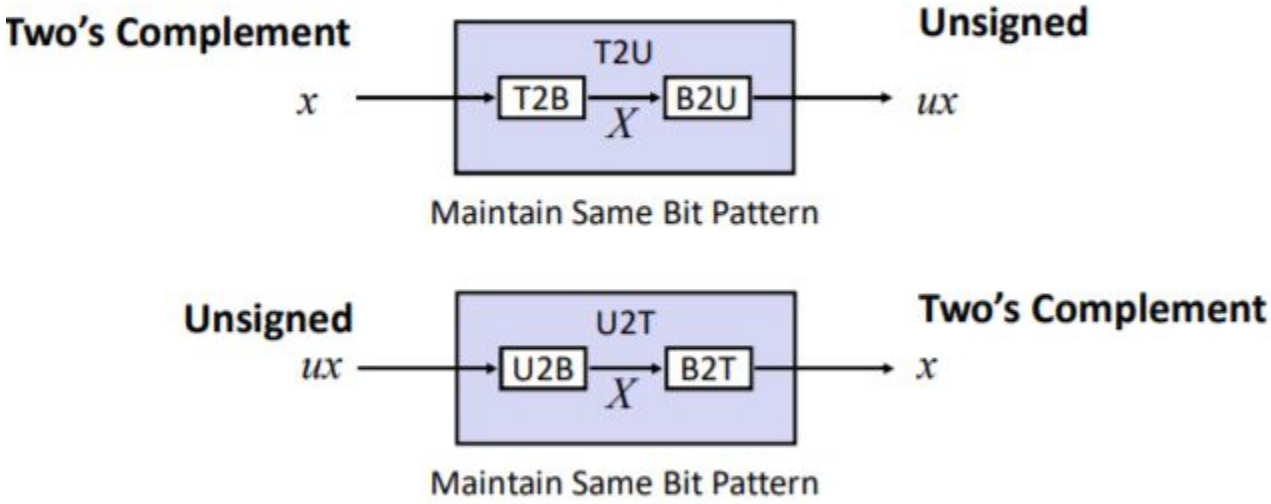
	Decimal	Hex	Binary
UMax	65535	FF FF	11111111 11111111
TMax	32767	7F FF	01111111 11111111
TMin	-32768	80 00	10000000 00000000
-1	-1	FF FF	11111111 11111111
0	0	00 00	00000000 00000000

Unsigned & Signed Numeric Values

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

- **Equivalence**
  - Same encodings for nonnegative values
- **Uniqueness**
  - Every bit pattern represents unique integer value
  - Each representable integer has unique bit encoding
- **⇒ Can Invert Mappings**
  - $U2B(x) = B2U^{-1}(x)$
    - Bit pattern for unsigned integer
  - $T2B(x) = B2T^{-1}(x)$
    - Bit pattern for two's comp integer

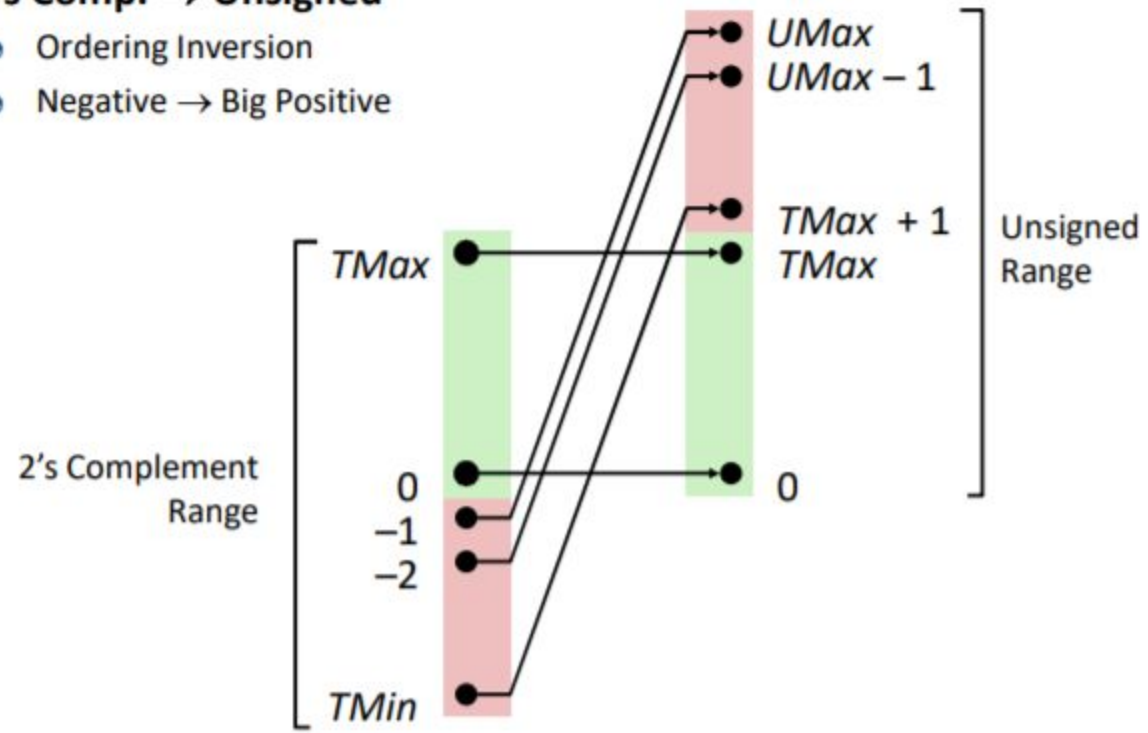
Mapping Between Signed & Unsigned



- **Mappings between unsigned and two's complement numbers:**
  Keep bit representations and reinterpret

Conversion Visualized

- **2's Comp. → Unsigned**
  - Ordering Inversion
  - Negative → Big Positive



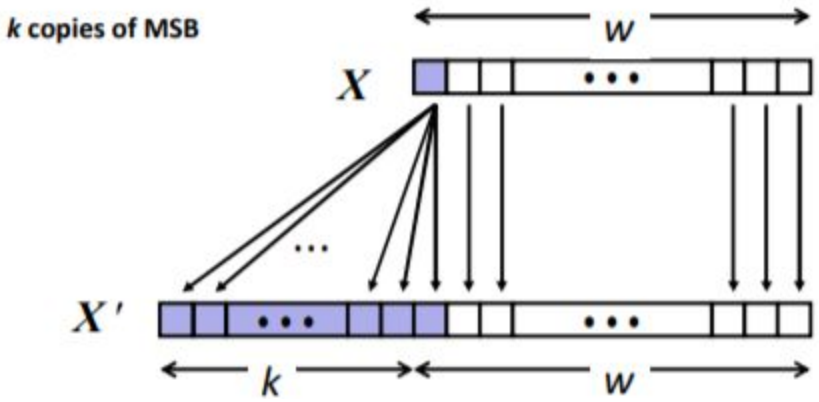
Casting Surprises

- **Expression Evaluation**
  - If there is a mix of unsigned and signed in single expression, ***signed values implicitly cast to unsigned***
  - Including comparison operations **<, >, ==, <=, >=**
  - Examples for *W* = 32: **TMIN = -2,147,483,648 , TMAX = 2,147,483,647**

Constant$_1$	**Constant$_2$**	**Relation**	**Evaluation**
0	0U	==	**unsigned**
-1	0	<	**signed**
-1	0U	>	**unsigned**
2147483647	-2147483647-1	>	**signed**
2147483647U	-2147483647-1	<	**unsigned**
-1	-2	>	**signed**
(unsigned)-1	-2	>	**unsigned**
2147483647	2147483648U	<	**unsigned**
2147483647	(int) 2147483648U	>	**signed**

Sign Extension

- **Task:**
  - Given *w*-bit signed integer *x*
  - Convert it to *w*+*k*-bit integer with same value
- **Rule:**
  - Make *k* copies of sign bit:
  - $X' = x_{w-1}, \ldots, x_{w-1}, x_{w-1}, x_{w-2}, \ldots, x_0$



Expanding, Truncating: Basic Rules

- **Expanding (e.g., short int to int)**
  - Unsigned: zeros added
  - Signed: sign extension
  - Both yield expected result

- **Truncating (e.g., unsigned to unsigned short)**
  - Unsigned/signed: bits are truncated
  - Result reinterpreted
  - Unsigned: mod operation
  - Signed: similar to mod
  - For small numbers yields expected behavior

Unsigned Addition

Operands: w bits

True Sum: w+1 bits

Discard Carry: w bits

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

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

ADDITION OF BINARY NUMBERS

- How do we add 10 + 5 using binary representation?
- What about 10 - 5?

LEFT SHIFT : MULTIPLY BY POWER OF 2

RIGHT SHIFT : DIVIDE BY POWER OF 2

- How do we multiply an integer x with 5?
- How do we divide an integer x by 8?
- What about unsigned integers? How do you round them towards zero?

Problems to think about :

1. Check if a given number is odd or even, by using only bitwise operators.
2. Check if an integer x is divisible by 4.
3. Check if an integer x is a power of 2.