COMP2611: Computer Organization

Data Representation

1. Binary numbers and 2's Complement

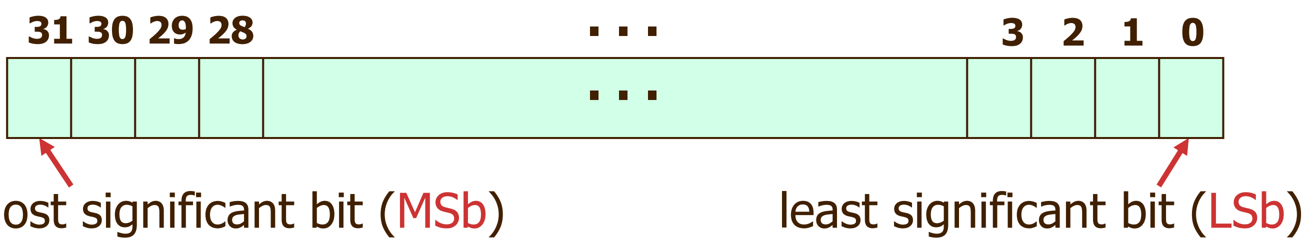
Numbers

- Bits: are the basis for binary number representation in digital computers
- What you will learn here:
  - How to represent negative integer numbers?
  - How to represent fractions and real numbers?
  - What is a representable range of numbers in a computer?
  - How to handle numbers that go beyond the representable range?
- To be covered in Computer Arithmetic:
  - Arithmetic operations: How to add, subtract, multiply, divide binary numbers
  - How to build the hardware that takes care of arithmetic operations

- Numbers can be represented in any base
  - Human: decimal (base 10, has 10 digits 0,1,…,9)
  - Computer: binary (base 2, has 2 digits, 0,1)
- Positional Notation: value of the `i`th digit `d` is `d x Base`^{i}

$1101_2 = (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)_{10} = 13_{10}$

- Bits are grouped and numbered 0, 1, 2, 3 … from right to the left:
  - Byte: a group of 8 bits
  - Word: a group of 32 or 64 bits



- Value of the 32-bit integer binary numbers =

$$(b_{31} \times 2^{31}) + (b_{30} \times 2^{30}) + \ldots + (b_1 \times 2^1) + (b_0 \times 2^0)$$

2's Complement for signed integer numbers

How can we represent negative integer values in binary?

- All computers use 2's complement representation for signed numbers
- The most significant bit is called the sign bit:
  - When it is 0 the number is non-negative
  - When it is 1 the number is negative
  - The positive half uses the same representation as before
  - The negative half uses the conversion from the positive value illustrated below:
  - Ex: What is the representation of -6 in 2's complement on 4 bits?

i) Start from the representation of +6

$$0110_2 = 6_{10}$$

ii) Invert bits to get 1's complement

$$1001_2 = -7_{10}$$

iii) Add 1 to get 2's complement

$$1010_2 = -6_{10}$$

Examples

- Ex: What is the representation of -6 in 2's complement on 8 bits?

```
i)   Representation of +6
ii)  Invert:
iii) Add 1
```

i) $0000\ 0110_2 = 6_{10}$

ii) $1111\ 1001_2 = -7_{10}$

iii) $1111\ 1010_2 = -6_{10}$

- Ex: What is the representation of -6 in 2's complement on 32 bits?

```
i)   Start from the representation of +6
```

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0110_2 = 6_{10}$

```
ii) Invert bits to get 1's complement
```

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1001_2 = -7_{10}$

```
iii) Add 1 to get 2's complement
```

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1010_2 = -6_{10}$

More on 2’s Complement

- **1’s complement**
  - MSb as in sign
  - Invert all the other bits
  - Given a positive number, negate all bits to get negative equivalent

Decimal number	Signed magnitude	1’s complement	2’s complement
3	011	011	011
2	010	010	010
1	001	001	001
0	000	000	000
-0	100	111	---
-1	101	110	111
-2	110	101	110
-3	111	100	101
-4			100

  - We don’t need 2 representations for 0
- **2’s complement = 1’s complement + 1**

Why invert and add 1 works?

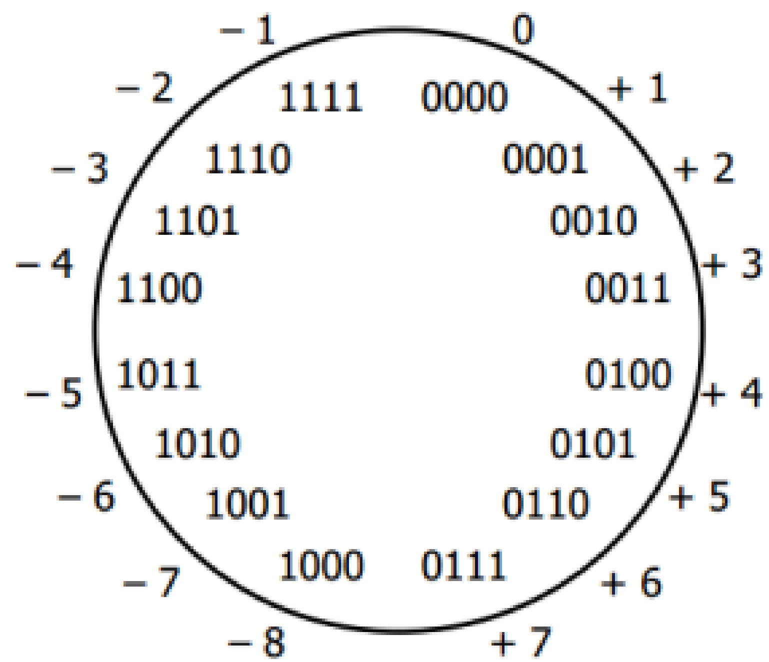
- In One's Complement we have: if $x = 0$ then $\bar{x} = 1$

$$x + \bar{x} = 1111...111_2$$

- In 2's complement $111...111_2$ = -1, therefore

$$x + \bar{x} = 1111...111_2 = -1$$

$$\bar{x} + 1 = -x$$



- Largest integer represented by a 32 bit word:
  $0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = (2^{31} - 1)_{10} = 2,147,483,647_{10}$
- Smallest integer represented by a 32 bit word:
  $1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = -2^{31}{}_{10} = -2,147,483,648_{10}$

- Example: what is largest and smallest integer represented by 8 bits (16 bits)
  - Largest integer

$0111\ 1111_2 = $ 0x7F $= 127 = 128 - 1 = 2^7 - 1$
$0111\ 1111\ 1111\ 1111_2 = $ 0x7FFF $= 32767 = 32768 - 1 = 2^{15} - 1$

  - Smallest integer

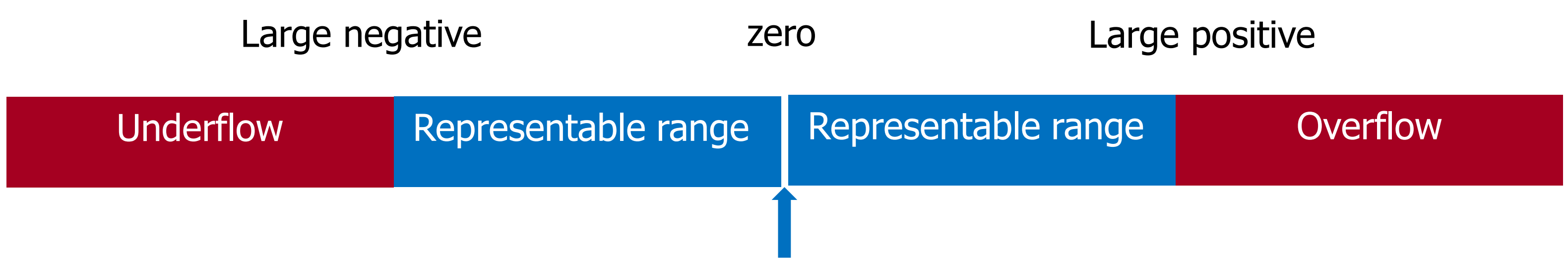
$1000\ 0000_2$
Invert and add 1: $0111\ 1111_2 + 1 = 1000\ 0000_2 = 2^7 = 128$
=> - 128
$1000\ 0000\ 0000\ 0000_2$
Invert – add 1: $0111\ 1111\ 1111\ 1111_2 + 1 = $ 0x8000 $= 2^{15} = 32768$
=> - 32768

Overflow and Underflow in Signed Integer

- Given the number of bits used in representing a signed integer
  - **Overflow** (signed integer)

    The value is bigger than the largest integer that can be represented
  - **Underflow** (signed integer)

    The value is smaller than the smallest integer that can be represented



- **Signed numbers**

  negative or **non-negative** integers, e.g. `int` in C/C++

- **Unsigned numbers**

  non-negative integers, e.g. `unsigned int` in C/C++

- **Ranges for signed and unsigned numbers**

  32 bit words `signed`:

  - from

    $0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = (2^{31} - 1)_{10} = 2{,}147{,}483{,}647_{10}$

  - to

    $1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = -2^{31}{}_{10} = -2{,}147{,}483{,}648_{10}$

  32 bit words `unsigned`:

  - from

    $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = 0_{10}$

  - to

    $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = (2^{32} - 1)_{10} = 4{,}294{,}967{,}295_{10}$

Sign Extension

- Consider using a cast in C/C++ on a 32 bit machine

```
int i;          /* signed integer represented on 32 bits */
char a; /* Character represented on 8 bits */
i = (int) a;
```

- What are the values of upper 24 bits in i?

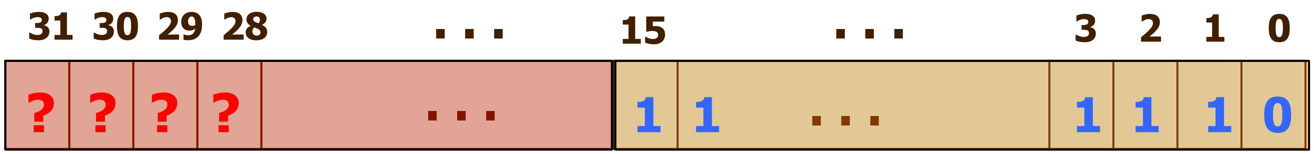
- Similar things happen in hardware when an instruction loads a 16 bit number into a 32 bit register (hardware variable)
- Bits 0~15 of the register will contain the 16-bit value
- What should be put in the remaining 16 bits (16~31) of the register?

- **Zero extension** fills missing bits with 0
  - Bitwise logical operations (e.g. bitwise AND, bitwise OR)
  - Casting unsigned numbers to larger width

- **Sign extension** is a way to **extend signed integer** to more bits

Sign Extension

- Bits 0~15 of **the register** will contain the **16bit value**
- What should be put in the remaining 16 bits (16~31) of the register?



- Depends on the sign of the 16 bit number

  If sign is 0 then fill with 0

  If sign is 1 then fill with 1

- For example:

  2 (16 bits -> 32 bits):

  0000 0000 0000 0010 -> 0000 0000 0000 0000 0000 0000 0000 0010

  -2 (16 bits -> 32 bits):

  1111 1111 1111 1110 -> 1111 1111 1111 1111 1111 1111 1111 1110

- **Does sign extension preserve the same value?**

2. Floating Point Numbers

- In addition to signed and unsigned integers, we also need to represent
  - **Numbers with fractions** (called real numbers in mathematics)
    - e.g. 3.1416
  - **Very small numbers**
    - e.g., 0.00000000001
  - **Very large numbers**
    - e.g., 1.23456×10^{10} (a number a 32-bit integer can't represent)

- In decimal representation, we have **decimal point**
- In binary representation, we call it **binary point**

$101.11_2 = (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1}) + (1 \times 2^{-2})_{10} = 5.75_{10}$

- Such numbers are called **floating point** in computer arithmetic
- Because the binary point is not fixed in the representation

- **Scientific notation**

  A single digit to the left of the decimal point

  e.g. 1.23×10^{-3}, 0.5×10^{5}
- **Normalized scientific notation**

  Scientific notation with no leading 0's

  e.g. 1.23×10^{-3}, 5.0×10^{4}

- Binary numbers can also be represented in scientific notation
- All normalized binary numbers always start with a 1

$$1.xxx \ldots xxx_{two} \times 2^{yyy \ldots yyy_{two}}$$

- Example $101.11_2 = 1.0111_2 \times 2^{10_2}$

Single-Precision Floating-Point Representation

- Single-precision uses 32 bits
- Sign-and-magnitude representation:

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

s	exponent	Significand/Mantissa
1 bit	**8 bits**	**23 bits**

- Interpretation

  S = sign; F = significand; E = exponent

  Value represented = $(-1)^{s} \times F \times 2^{E}$

  Roughly gives 7 decimal digits in precision

  Exponent scale of about 10^{-38} to 10^{+38}

- Compromise between sizes of exponent and significand fields:

  Increase size of **exponent** $\Rightarrow$ increase representable range

  Increase size of **significand** $\Rightarrow$ increase accuracy

Double-Precision Floating-Point Representation

- Double-precision floating-point uses 64 bits
  - In 32 bit architectures like MIPS, each double-precision number requires two MIPS words
  - 11 bits for exponent, 52 bits for significand

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

S	**Exponent**	**Significand/Mantissa**
1 bit	**11 bits**	**20 bits**

Significand/Mantissa (continued)
32 bits

- Provides precision of about 16 decimals
- Exponent scale from 10^{-308} to 10^{+308}

IEEE 754 Floating-Point Standard

- Most computers use this standard for both single and double precision

- Why use a standard floating-point representation?
  - Simplify porting floating-point programs across different computers

- To pack even more bits into the significand

  This standard makes the leading 1 bit (in 1.xx … xxx) implicit

  Interpretation: $(-1)^{S} \times (1 + 0.\text{significand}) \times 2^{E}$

  Effective number of bits used for representing the significand:
  - 24 (i.e., 23 + 1) – for single precision
  - 53 (i.e., 52 + 1) – for double precision

  Special case:
  - Since 0 has no leading 1, it is given the reserved exponent value 0 so that the hardware does not attach a leading 1 to it

- **Computation of significand**:

$$\text{Significand} = s_1 \times 2^{-1} + s_2 \times 2^{-2} + s_3 \times 2^{-3} + \ldots$$

The significand bits are denoted as $\boxed{s_1, s_2, s_3, \ldots,}$ from left to right

- **To allow quick comparisons in hardware implementation**:

The sign is in the most significant bit

The exponent is placed before the significand

(Comparisons mean "less than", "greater than", "equal to zero")

- **How to represent a NEGATIVE exponent?**

Biased exponent: a bias is implicitly added to the exponent

$$(-1)^{s} \times (1 + 0.\text{significand}) \times 2^{(E\text{-bias})}$$

bias = 127 for single precision, bias = 1023 for double precision

The most negative exponent = 0_2, the most positive = $11\ldots11_2$

- What decimal number is represented by this word (single precision)?

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

- **Answer**:

$$(-1)^s \times (1 + Significand) \times 2^{(E-Bias)}$$

$$= (-1)^1 \times (1 + 0.25) \times 2^{(129-127)}$$

$$= -1 \times 1.25 \times 2^2$$

$$= -1.25 \times 4$$

$$= -5.0$$

- Give the binary representation of -0.75_{10} in single & double precisions

- **Answer**

  $-0.75_{10} = -0.11_2$

  $0.75 * 2 = \mathbf{1.50}$, S1 = **1**

  $0.50 * 2 = \mathbf{1.00}$, S2 = **1**; stop because the fraction is 0.00

  Scientific notation: $-0.11_2 \times 2^0$

  Normalized scientific notation: $-1.1_2 \times 2^{-1}$

  Sign = 1 (negative), exponent = -1

  Single precision:

  S = 1, E = 01111110, significand = 100...00 (23 bits)

  = -1+127, (127 is the bias)

  Double precision:

  S = 1, E = 01111111110, significand = 100...00 (52 bits)

  = -1+1023, (1023 is the bias)

IEEE 754 Standard for Floating Point Arithmetic

Single precision:

Exponent / Significand	0 (Denormalized)	**1 - 254** (Normalized)	255
0	0	$(-1)^S \times (1.F) \times (2)^{E-127}$	$(-1)^S \times (\infty)$
≠ 0	$(-1)^S \times (0.F) \times (2)^{-126}$		non-numbers e.g. 0/0 , $\sqrt{-1}$

Double precision:

Exponent / Significand	0	**1 - 2046**	2047
0	0	$(-1)^S \times (1.F) \times (2)^{E-1023}$	$(-1)^S \times (\infty)$
≠ 0	$(-1)^S \times (0.F) \times (2)^{-1022}$		non-numbers e.g. 0/0 , $\sqrt{-1}$

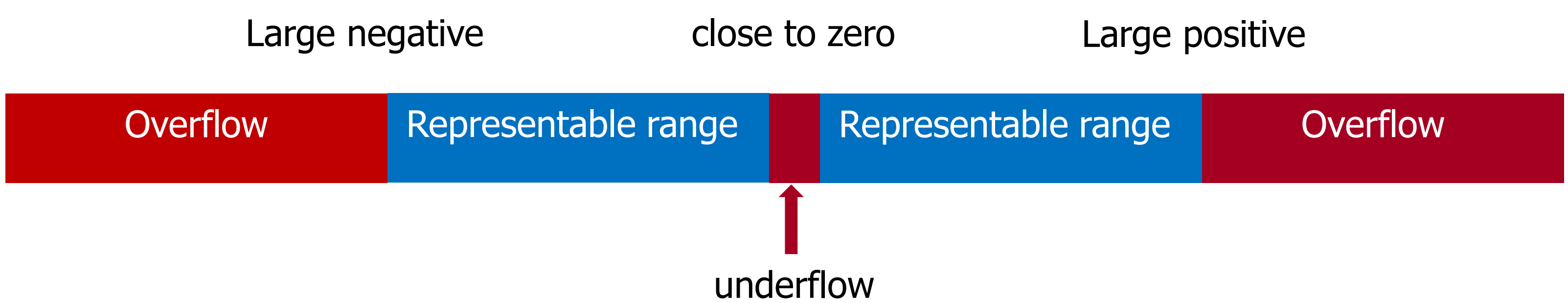
Example

0	00000000	00000000000000000000000	=	0
1	00000000	00000000000000000000000	=	-0
0	11111111	00000000000000000000000	=	+ infinity
1	11111111	00000000000000000000000	=	- infinity
0	11111111	01001100010001000001000	=	**NaN** (**Not a Number**)
1	11111111	01001100010001000001000	=	NaN
0	10000000	00000000000000000000000	=	$+(1.0_2)\times(2)^{128-127}=2$
0	10000001	10100000000000000000000	=	$+(1.101_2)\times(2)^{129-127}=6.5$
1	10000001	10100000000000000000000	=	$-(1.101_2)\times(2)^{129-127}=-6.5$
0	00000001	00000000000000000000000	=	$+(1.0_2)\times(2)^{1-127}=(2)^{-126}$
0	00000000	10000000000000000000000	=	$+(0.1_2)\times(2)^{-126}=(2)^{-127}$
0	00000000	00000000000000000000001	=	$+(2)^{-23}\times(2)^{-126}=(2)^{-149}$

Overflow and Underflow in Floating Point

- **Overflow** (floating-point)
  A positive exponent becomes too large to fit in the exponent field

- **Underflow** (floating-point)
  A negative exponent becomes too large to fit in the exponent field



Communicating with People

❑ How to represent Characters

Characters are unsigned bytes e.g., in C++ `Char`

Usually follow the ASCII standard

Uses 8 bits unsigned to represent a character

b_7 →					0	0	0	0	1	1	1	1
b_6 →					0	0	1	1	0	0	1	1
b_5 →					0	1	0	1	0	1	0	1
Bits	b_4 ↓	b_3 ↓	b_2 ↓	b_1 ↓	Column → / Row ↓	0	1	2	3	4	5	6	7
	0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
	0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
	0	0	1	0	2	STX	DC2	"	2	B	R	b	r
	0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
	0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
	0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
	0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
	0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
	1	0	0	0	8	BS	CAN	(	8	H	X	h	x
	1	0	0	1	9	HT	EM	)	9	I	Y	i	y
	1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
	1	0	1	1	11	VT	ESC	+	;	K	[	k	{
	1	1	0	0	12	FF	FC	,	<	L	\	l	|
	1	1	0	1	13	CR	GS	-	=	M	]	m	}
	1	1	1	0	14	SO	RS	.	>	N	^	n	~
	1	1	1	1	15	SI	US	/	?	O	_	o	DEL

Exercises

- **What does the following 32 bit pattern represent: 0x32363131**
  - If it were a 2's complement integer

    `the MSb is 0 therefore this is a positive number`

    `evaluation left as an exercise`
  - An unsigned number

    `Same value as above`
  - A sequence of ASCII encoded bytes: `2611`

    `Checking the ascii table gives:`

    `0x32 = code for character '2'`

    `0x36 = code for character '6'`

    `0x32 = code for character '1'`

    `0x32 = code for character '1'`
  - A 32 bit IEEE 754 floating point number

    `s= 0, E = 01100100, S = 01101100011000100110001`

    `This is a normalized number so E is biased.`

- Consider building a floating point number system like the IEEE754 standard on 8 bit only, with 3 bits being reserved for the exponent.
  - What is the value of the bias?
    - **3**
  - What is the representation of 0?
    - **0 000 0000**
  - What is the representation of -4?
    - **$-4 = -1.0 \times 2^2$**
    - **S=1, F= 0 and the biased exponent must be**
    - **E – 3 = 2 or E = +5**
    - **So - 4 = 1 101 0000**
  - What is the next value representable after – 4?
    - **1 101 0001 = - 4.25 so we can see that 4 bits for the significand is not accurate enough**
  - What does the byte 1 111 1011 represent? **- NAN**
  - What is the representation of -∞? **1 111 0000**

Key Concepts to Remember

- **2's complement representation** for signed numbers
- **Floating-point numbers**
  - Representation follows closely the **scientific notation**
  - Almost all computers, including MIPS, follow **IEEE 754 standard**
- **Single-precision** floating-point representation takes 32 bits
- **Double-precision** floating-point representation takes 64 bits
- **Overflow** and **underflow** in signed integer and floating number