NUMERIC COMPUTATION

Yujing M. Jiang
The University of Melbourne, 2018

Integer representations

Bit pattern	Integer representation		
	unsigned	sign-magn.	twos-comp.
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	8	−0	−8
1001	9	−1	−7
1010	10	−2	−6
1011	11	−3	−5
1100	12	−4	−4
1101	13	−5	−3
1110	14	−6	−2
1111	15	−7	−1

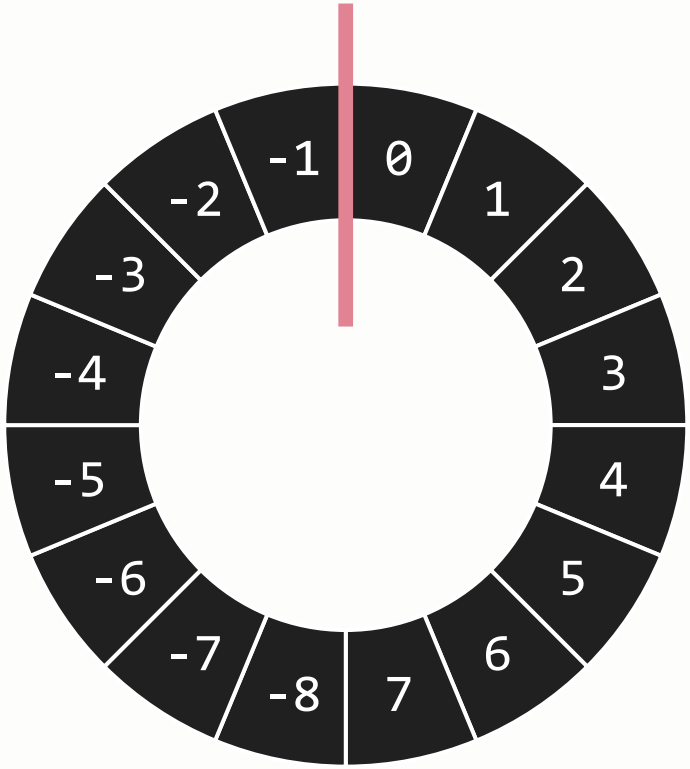
Binary: Sign-magnitude

$Sign$	2^2	2^1	2^0	$Decimal$
0	1	0	1	$+5$
1	1	0	1	-5
0	0	0	0	$+0$
1	0	0	0	-0
0	0	1	0	$+2$
1	0	1	0	-2

$Sign$: $0+$ $1-$

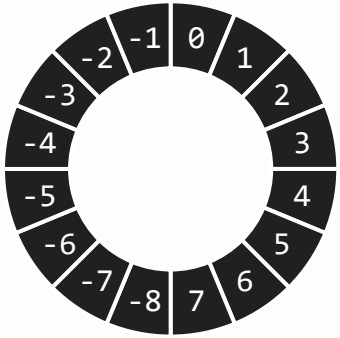
Bit pattern	representation sign-magn.
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	−0
1001	−1
1010	−2
1011	−3
1100	−4
1101	−5
1110	−6
1111	−7

Binary: Two's complement



Bit pattern	twos-comp.
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	−8
1001	−7
1010	−6
1011	−5
1100	−4
1101	−3
1110	−2
1111	−1

Binary: Two's complement



How to represent a negative number?

1. Flip all bits of the positive binary.
2. Plus 1.

Bit pattern	twos-comp.
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	−8
1001	−7
1010	−6
1011	−5
1100	−4
1101	−3
1110	−2
1111	−1

	$Sign$	2^2	2^1	2^0	$Decimal$
	0	1	0	1	$+5$
$Flip:$	1	0	1	0	
$+1:$	1	0	1	1	-5
	0	0	1	0	$+2$
$Flip:$	1	1	0	1	
$+1:$	1	1	1	0	-2

Overflow

In C Programming:

Storage: ***Two's comp***.

Operation: ***Unsigned***

```
int4 i = 7;
i = i + 1;
printf("%d ", i);
// Output: -8

int4 i = 7;
i = i + 2;
printf("%d ", i);
// Output: -7
```

Bit pattern	Integer	
	unsigned	twos-comp.
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

Number Representation: Binary 16-bit

- Similar to ***Scientific Notation***: $+0.101 \times 2^3$
- Stored as 3 parts:
- 1. Sign ($+$ or $-$)
- 2. Fraction (aka. Mantissa, 1.01)
- 3. Exponential offset (3 in $\times 2^3$, using two's complement)
- However, in fraction (mantissa), the standard form is 0.xxxx.

Number (decimal)	Number (binary)	Exponent (decimal)	Mantissa (binary)	Representation (bits)
0.5	0.1	0	.100000000000	0 000 1000 0000 0000
0.375	0.011	−1	.110000000000	0 111 1100 0000 0000
3.1415	11.001001000011 · · ·	2	.110010010000	0 010 1100 1001 0000
−0.1	−0.0001100110011 · · ·	−3	.110011001100	1 101 1100 1100 1100

Number Representation: Binary 32-bit

- Similar to ***Scientific Notation***: $+1.000 \times 2^0$
- Stored as 3 parts:
- 1. Sign (+ or −)
- 2. Fraction (aka. Mantissa, 1.01)
- 3. Exponential offset, + 127 (0+127 = 127, **biased**)
- ****However, in fraction (mantissa), the standard form is **1**.xxxx.

0 01111111 00000000000000000000000

Number Representation: 16-bit vs 32-bit

	Sign	Exponential	Mantissa
16-bit	1	3	12
32-bit	1	8	23

	Sign	Exponential	Mantissa
16-bit	`0:+, 1:-`	Two's comp.	0.xxxx
32-bit	`0:+, 1:-`	Biased +127	1.xxxx