Signed number systems

CSC 236

Java in a Nutshell (Flanagan)

“Integer arithmetic in Java is modular which means that it never produces an overflow when you exceed the range of a given integer type. Instead numbers just wrap around.”

Java in a Nutshell (Flanagan)

“Integer arithmetic in Java is modular which means that it never produces an overflow when you exceed the range of a given integer type. Instead numbers just wrap around.”

Neither the Java compiler nor Java VM warns you in any way when this occurs.

Java in a Nutshell (Flanagan)

"Integer arithmetic in Java is modular which means that it never produces an overflow when you exceed the range of a given integer type. Instead numbers just wrap around."

Neither the Java com
occurs.

When doing integer arithmetic, you simply must ensure that the type you are using has sufficient range for the purpose you intend.

New plan:

New plan:

Don’t put bugs in your Java Code.

Part of a C program

```
unsigned U = 10;

if (U > -1) printf ("U greater than -1");

if (U < -1) printf ("U less than -1 ");
```

Which message is desired?

Part of a C program

```
unsigned U = 10;

if (U > -1) printf ("U greate     1");

if (U < -1) printf ("U     n -1 ");
```

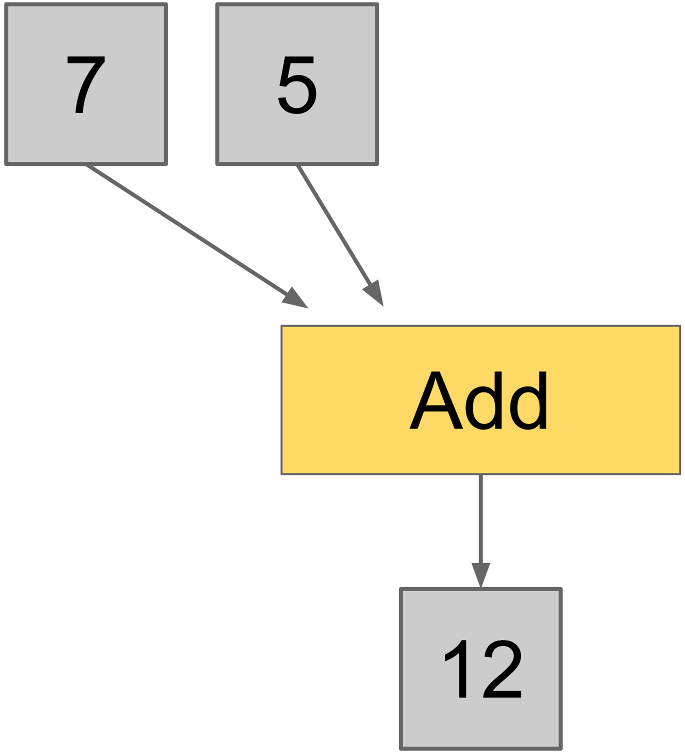
Explained in this lecture

Which message is printed?

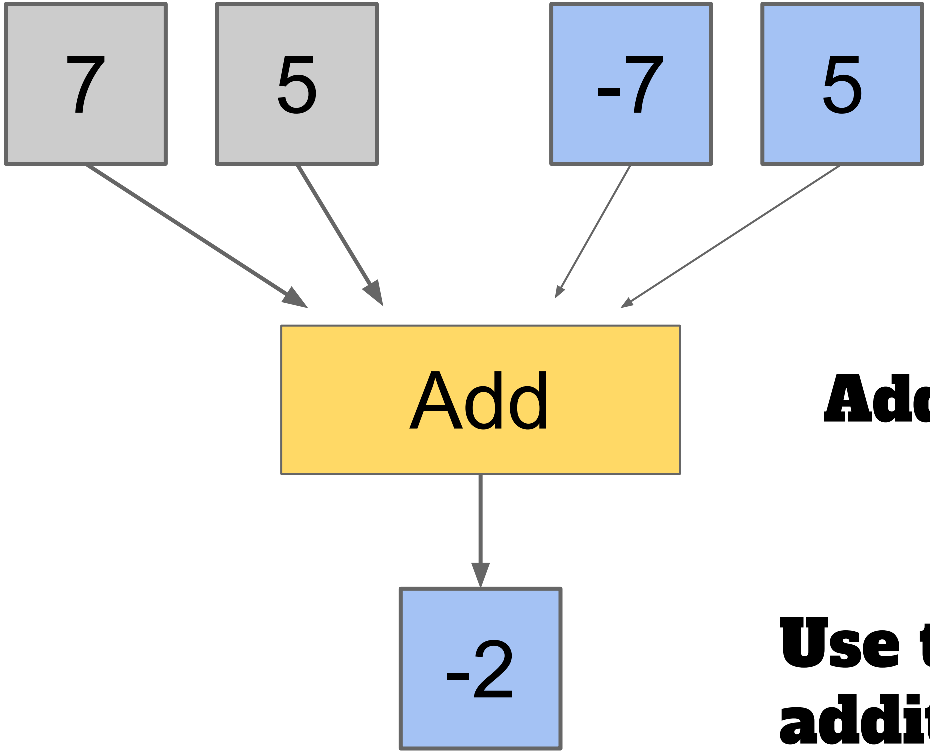
C thinks 10 < -1

Architectural design

- Previously: Showed unsigned positional number (UPN)
- Need to handle signed numbers as well
- Critical architectural question
  - Do signed number system use same addition rule as UPN?

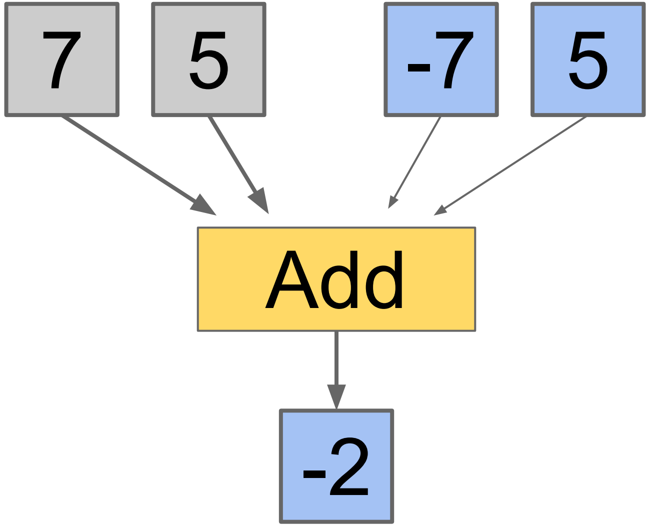


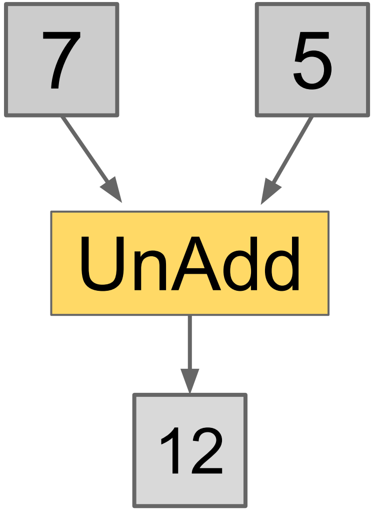
Add hardware



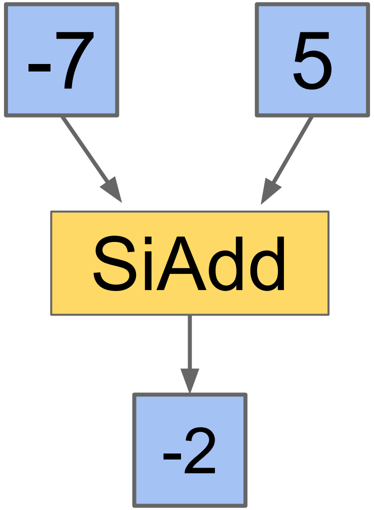
Add hardware

Use the same rule for addition in signed number system as in UPN?





Different hardware



Use the same rule for addition in signed number system as in UPN?
No right or wrong answer

Yes

- 1 instruction
- Less hardware
- Less cost

No

- Can optimize performance of both signed and unsigned instructions

Does our “everyday” signed magnitude follow the addition rule for UPN?

Unsigned	Signed Magnitude
$\begin{array}{r} 10 \\ +\ \underline{05} \\ 5 \end{array}$	$\begin{array}{r} +10 \\ +\ \underline{-05} \end{array}$

Does our "everyday" signed magnitude follow the addition rule for UPN?

Unsigned	Signed Magnitude
10	+10
+ 05	+ -05
15	

Does our "everyday" signed magnitude follow the addition rule for UPN?

Unsigned	Signed Magnitude
10	+10
+ 05	+ -05
15	5

Does our “everyday” signed magnitude follow the addition rule for UPN?

Unsigned	Signed Magnitude
$\begin{array}{r} 10 \\ +\ \underline{05} \\ 15 \end{array}$	$\begin{array}{r} +10 \\ +\ \underline{-05} \\ 15 \end{array}$

Does our “everyday” signed magnitude follow the addition rule for UPN?

Unsigned	Signed Magnitude
10	+10
+ 05	+ -05
15	?15

Does our "everyday" signed magnitude follow the addition rule for UPN?

Unsigned	Signed Magnitude
10	+10
+ 05	+ -05
15	?15

The rule does not handle this column

Can you do arithmetic on Signed Magnitude numbers?

Yes. You extend the UPN rule.

Extend the UPN rule to handle Signed Magnitude

+10	+10	-10	-10	+10
+<u>05</u>	-<u>05</u>	+<u>05</u>	-<u>05</u>	<u>-10</u>

Extend the UPN rule to handle Signed Magnitude

+10	+10	-10	-10	+10
+05	-05	+05	-05	-10
+15				

Extend the UPN rule to handle Signed Magnitude

+10	+10	-10	-10	+10
+05	-05	+05	-05	-10
15	+05			

Extend the UPN rule to handle Signed Magnitude

+10	+10	-10	-10	+10
+05	-05	+05	-05	-10
15	+05	-05		

Extend the UPN rule to handle Signed Magnitude

+10	+10	-10	-10	+10
+05	-05	+05	-05	-10
15	+05	-05	-15	

Extend the UPN rule to handle Signed Magnitude

+10	+10	-10	-10	+10
+<u>05</u>	-<u>05</u>	+<u>05</u>	-<u>05</u>	<u>-10</u>
15	+05	-05	-15	±00

Extend the UPN rule to handle Signed Magnitude

+10	+10	-10	-10	+10
+05	-05	+05	-05	-10
+15	+05	-05	-15	±00

Extend the UPN rule to handle Signed Magnitude

+10	+10	-10	-10	+10
+05	-05	+05	-05	-10
+15	+05	-05	-15	±00

If signs are the same

- **Add digits using UPN rule**
- **Result has sign of input numbers**

Extend the UPN rule to handle Signed Magnitude

+10	+10	−10	−10	+10
+05	−05	+05	−05	−10
+15	+05	−05	−15	±00

Else if magnitudes are different

- **Subtract smaller from larger**
- **Result has sign of larger**

Extend the UPN rule to handle Signed Magnitude

+10	+10	-10	-10	+10
+<u>05</u>	-<u>05</u>	+<u>05</u>	-<u>05</u>	<u>-10</u>
+15	+05	-05	-15	±00

Else

- **Result is zero**
- **Sign is plus or minus**

Extend the UPN rule to handle Signed Magnitude

+10	+10	-10	-10	+10
+05	-05	+05	-05	-10
+15	+05	-05	-15	±00

Do you want to build such a complicated rule into hardware?

Signed numbers

Use 4 bits d d d d

leftmost bit is sign
0 = pos 1 = neg

other bits provide magnitude

We will see ...

- **Positive numbers look the same but**
- **Negative number differ**

Question

$$+0 \stackrel{?}{=} -0$$

+0 & -0 dollars in you wallet is the same.

Told that launch authorization code is: zero.

The program checks:

```
if input == code
```

There are multiple ways to represent signed numbers

We will look at three different ways:

- Signed magnitude
- One’s complement
- Two’s complement

Characteristics of interest

1. Rule for addition
2. How many ways can you represent zero
3. Relative quantity of positive and negative values

Signed magnitude

bin	dec	dec	bin
0000	+ 0		
0001	+ 1		
0010	+ 2		
0011	+ 3		
0100	+ 4		

Signed magnitude

bin	dec	dec	bin
0000	+ 0	- 0	1000
0001	+ 1	- 1	1001
0010	+ 2	- 2	1010
0011	+ 3	- 3	1011
0100	+ 4	- 4	1100

Negate obtained by inverting sign bit

Signed magnitude

bin	dec	dec	bin
0000	+ 0	- 0	1000
0001	+ 1	- 1	1001
0010	+ 2	- 2	1010
0011	+ 3	- 3	1011
0100	+ 4	- 4	1100

Add numbers

$$\begin{array}{rcl} +1 & = & 0001 \\ -\underline{1} & = & \underline{1001} \\ +/-\ 0 & & \end{array}$$

Signed magnitude

bin	dec	dec	bin
0000	+ 0	- 0	1000
0001	+ 1	- 1	1001
0010	+ 2	- 2	1010
0011	+ 3	- 3	1011
0100	+ 4	- 4	1100

Add numbers

$$\begin{array}{rcr} & & 1 \\ +1 & = & 0001 \\ -\underline{1} & = & \underline{1001} \\ +/-\,0 & & 1010 \end{array}$$

To add binary signed magnitude numbers, must extend UPN rule (as for decimal)

Characteristics of interest

	Signed magnitude		
Same rule for addition	No		
How many ways can you represent zero	2		
Relative quantity of positive and negative values	Same		

Has computer been built using Signed Magnitude?

Has a computer been built using Signed Magnitude?

IBM 7090 -- circa 1958

Intended for the scientific community and NASA's Mercury and Gemini space missions

One's complement

bin	dec	dec	bin
0000	+ 0		
0001	+ 1		
0010	+ 2		
0011	+ 3		
0100	+ 4		

One's complement

bin	dec	dec	bin
0000	+ 0	- 0	1111
0001	+ 1	- 1	1110
0010	+ 2	- 2	1101
0011	+ 3	- 3	1100
0100	+ 4	- 4	1011

Negate obtained by inverting all bits

One's complement

bin	dec	dec	bin
0000	+ 0	- 0	1111
0001	+ 1	- 1	1110
0010	+ 2	- 2	1101
0011	+ 3	- 3	1100
0100	+ 4	- 4	1011

Add numbers

+ 1 = 0001
- 1 = 1110

+/- 0

One's complement

bin	dec	dec	bin
0000	+ 0	- 0	1111
0001	+ 1	- 1	1110
0010	+ 2	- 2	1101
0011	+ 3	- 3	1100
0100	+ 4	- 4	1011

Add numbers

+ 1	=	0001
- 1	=	1110
+/- 0		1111

Equals -0, Looks okay

Another example

$$
\begin{array}{rcl}
-1 & = & 1\ 1\ 1\ 0 \\
-1 & = & \underline{1\ 1\ 1\ 0} \\
-2 & &
\end{array}
$$

Another example

```
      1 1 1
- 1 =  1 1 1 0
- 1 =  1 1 1 0
- 2    1 1 0 0
```

Does this equal -2?

Another example

$$
\begin{array}{rrcrrrr}
 & & & 1 & 1 & 1 & \\
-1 & = & & 1 & 1 & 1 & 0 \\
-1 & = & & \underline{1} & \underline{1} & \underline{1} & \underline{0} \\
-2 & & & 1 & 1 & 0 & 0
\end{array}
\Rightarrow \; 0\ 0\ 1\ 1 = +3
$$

Flip bits

Another example

```
          1 1 1
- 1 =   1 1 1 0
- 1 =   1 1 1 0
        -------
- 2     1 1 0 0        0 0 1 1 = +3
```

-3

One's complement

Need to extend the UPN rule for one's complement

"End around carry"

```
          1 1 1
- 1 =   1 1 1 0
- 1 =   1 1 1 0
- 2     1 1 0 0
              1
        1 1 0 1
```

A carry out of the left most digit is added to the right most digit

Characteristics of interest

	Signed magnitude	One's complement	
Same rule for addition	No	No (simple rule)	
How many ways can you represent zero	2	2	
Relative quantity of positive and negative values	Same	Same	

Has computer been built using One's complement?

Has a computer been built using One's complement?

DEC PDP-1 - 1960

First mini-computer

Has a computer been built using One's complement?

Univac 1100 - 1962

Main frame

Two's complement

Definition

$$N + TC(N) = 2^d$$

N	(any number)
TC(N)	(negation of N)
d	(number of digits used)

$N + (-N) = 0$

Two's complement

$N + TC(N) = 2^d$

$N + (-N) = 0$

$1\ \boxed{0\ 0\ 0\ 0\ 0\ 0\ 0}$

d zeros

carry + zero

- $2^0 = 1$
- $2^1 = 10$
- $2^2 = 100$
- $2^3 = 1000$
- …
- $2^d = 10…0$
- $2^d = 1$ followed by d 0s

How to calculate two's complement

$$N + TC(N) = 2^d$$

Rewrite

$$TC(N) = 2^d - N$$

For d = 4

$$TC(N) = 10000 - N$$

Two's complement

bin	dec	dec	bin
0000	+ 0		
0001	+ 1		
0010	+ 2		
0011	+ 3		
0100	+ 4		

Two's complement

TC(N) = `10000` **- N**

bin	dec		dec	bin
0000	+ 0	- 0		
0001	+ 1	- 1		
0010	+ 2	- 2		
0011	+ 3	- 3		
0100	+ 4	- 4		

Two's complement

$TC(N) = 10000 - N$

bin	dec		dec	bin
0000	+ 0	- 0		
0001	+ 1	- 1		
0010	+ 2	- 2		
0011	+ 3	- 3		
0100	+ 4	- 4		

$$\begin{array}{r} 10000 \\ -\ \underline{\ 0001} \end{array}$$

Two's complement

TC(N) = 10000 - **N**

bin	dec		dec	bin
0000	+ 0	- 0		
0001	+ 1	- 1		
0010	+ 2	- 2		
0011	+ 3	- 3		
0100	+ 4	- 4		

redistribute the data

$$
\begin{array}{r}
1 \\
01111 \\
\cancel{10000} \\
-\quad \underline{0001}
\end{array}
$$

Two's complement

$\mathbf{TC(N)} = 10000 - \mathbf{N}$

bin	dec	dec	bin
0000	+ 0	- 0	
0001	+ 1	- 1	
0010	+ 2	- 2	
0011	+ 3	- 3	
0100	+ 4	- 4	

$$\begin{array}{r} 1 \\ 01111 \\ \cancel{10000} \\ -\ \underline{\ \ 0001} \\ 1111 \end{array}$$

Two's complement

$TC(N) = 10000 - N$

bin	dec		dec	bin
0000	+ 0	- 0		
0001	+ 1	- 1	1111	
0010	+ 2	- 2		
0011	+ 3	- 3		
0100	+ 4	- 4		

$$\begin{array}{r} 1 \\ 01111 \\ \cancel{10000} \\ -\ \underline{\ \ 0001} \\ 1111 \end{array}$$

Two's complement

$TC(N) = 10000 - N$

bin	dec		dec		bin
0000	+ 0	- 0			
0001	+ 1	- 1		1111	
0010	+ 2	- 2			
0011	+ 3	- 3			
0100	+ 4	- 4			

$$\begin{array}{r} 10000 \\ - \underline{\quad 0010} \end{array}$$

Two's complement

TC(N) = 10000 - N

bin	dec	dec	bin
0000	+ 0	- 0	
0001	+ 1	- 1	1111
0010	+ 2	- 2	
0011	+ 3	- 3	
0100	+ 4	- 4	

$$\begin{array}{rr} & 10000 \\ - & \underline{\ \ 0010} \\ & 0 \end{array}$$

Two's complement

TC(N) = 10000 - **N**

bin	dec		dec	bin
0000	+ 0	- 0		
0001	+ 1	- 1	1111	
0010	+ 2	- 2		
0011	+ 3	- 3		
0100	+ 4	- 4		

```
    1
 0111
 10000
- 0010
------
     0
```

Two's complement

TC(N) = 10000 - N

bin	dec	dec	bin
0000	+ 0	- 0	
0001	+ 1	- 1	1111
0010	+ 2	- 2	
0011	+ 3	- 3	
0100	+ 4	- 4	

$$\begin{array}{r} 1 \\ 0111 \\ \cancel{1000}0 \\ -\quad 0010 \\ \hline 1110 \end{array}$$

Two's complement

TC(N) = 10000 - N

bin	dec	dec	bin
0000	+ 0	- 0	
0001	+ 1	- 1	1111
0010	+ 2	- 2	1110
0011	+ 3	- 3	
0100	+ 4	- 4	

$$\begin{array}{r} 1 \\ 0111 \\ \cancel{1000}0 \\ -\quad 0010 \\ \hline 1110 \end{array}$$

Is there an easier way to calculate two’s complement?

Yes. There is a *shortcut*

Two's complement shortcut

TC(N) = flip-bits + 1

Two's complement

TC(N) = flip-bits + 1

bin	dec	dec	bin
0000	+ 0	- 0	
0001	+ 1	- 1	1111
0010	+ 2	- 2	1110
0011	+ 3	- 3	
0100	+ 4	- 4	

```
+3 = 0011

-3 = 1100
   +    1
     ----
     1101
```

Two's complement

TC(N) = flip-bits + 1

bin	dec	dec	bin
0000	+ 0	- 0	
0001	+ 1	- 1	1111
0010	+ 2	- 2	1110
0011	+ 3	- 3	1101
0100	+ 4	- 4	

```
+3 = 0011

-3 = 1100
    +   1
     1101
```

Two's complement

TC(N) = flip=bits + 1

bin	dec	dec	bin
0000	+ 0	- 0	
0001	+ 1	- 1	1111
0010	+ 2	- 2	1110
0011	+ 3	- 3	1101
0100	+ 4	- 4	1100

$$
\begin{array}{rcr}
+4 & = & 0100 \\
-4 & = & 1011 \\
 & + & \underline{\quad 1} \\
 & & 1100
\end{array}
$$

Two's complement

bin	dec	dec	bin
0000	+ 0	- 0	
0001	+ 1	- 1	1111
0010	+ 2	- 2	1110
0011	+ 3	- 3	1101
0100	+ 4	- 4	1100

Can we add using the UPN rule?

```
+1  =  0001
-1  =  1111
+0  =
```

Two's complement

bin	dec	dec	bin
0000	+ 0	- 0	
0001	+ 1	- 1	1111
0010	+ 2	- 2	1110
0011	+ 3	- 3	1101
0100	+ 4	- 4	1100

Can we add using the UPN rule?

```
       1111
+1  =  0001
-1  =  1111
+0  =  0000
```

Two's complement

bin	dec	dec	bin
0000	+ 0	- 0	
0001	+ 1	- 1	1111
0010	+ 2	- 2	1110
0011	+ 3	- 3	1101
0100	+ 4	- 4	1100

```
      1111
-1 = 1111
-1 = 1111
-2 = 1110
```

Remember, -1 + (-1) didn't work for one's complement

Two's complement

bin	dec	dec	bin
0000	+ 0	- 0	????
0001	+ 1	- 1	1111
0010	+ 2	- 2	1110
0011	+ 3	- 3	1101
0100	+ 4	- 4	1100

```
+0    = 0000

TC(0)= 1111
     +    1
```

What about zero? How many representations?

Two's complement

bin	dec	dec	bin
0000	+ 0	- 0	~~????~~
0001	+ 1	- 1	1111
0010	+ 2	- 2	1110
0011	+ 3	- 3	1101
0100	+ 4	- 4	1100

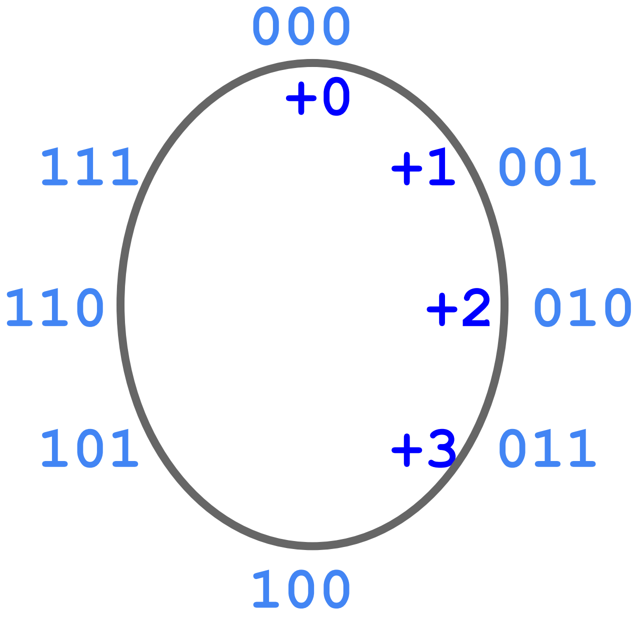
```
+0     = 0000

TC(0)= 1111
     +    1
     ------
       0000
```

What about zero? How many representations?

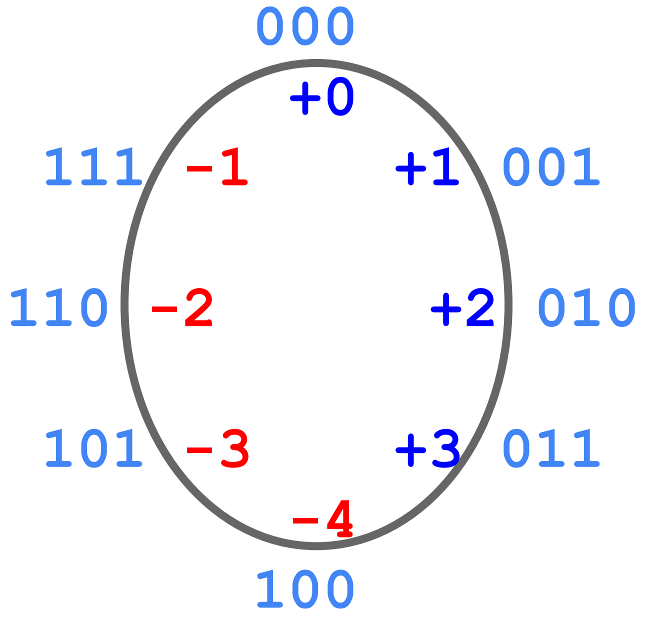
Two's complement

Only one zero leads to quirk.



Two's complement

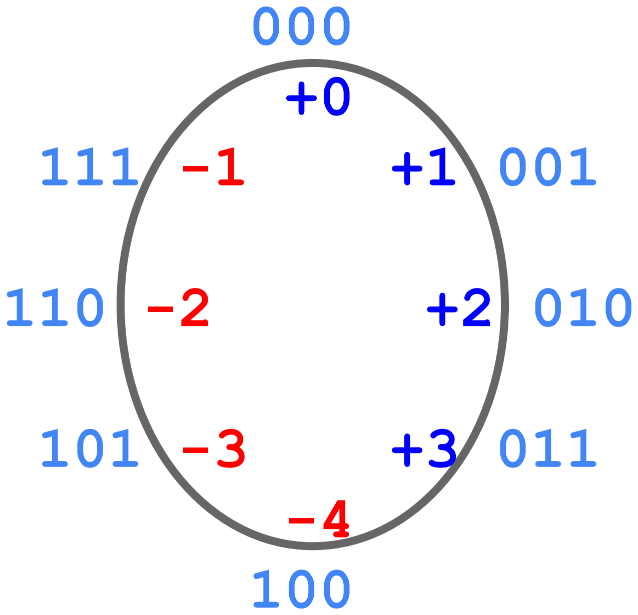
Only one zero leads to quirk.



Two's complement

Only one zero leads to quirk.

Quantity of positive and negative values differ



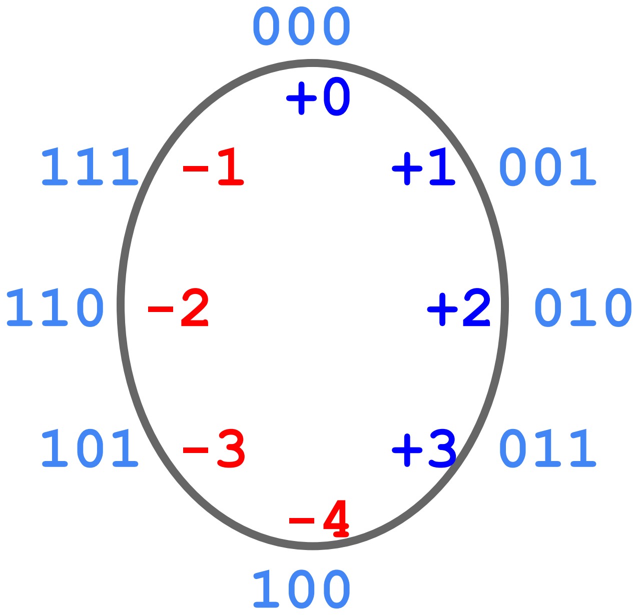
Two's complement

Can we verify the number wheel?

`0 - 1` should be `-1`

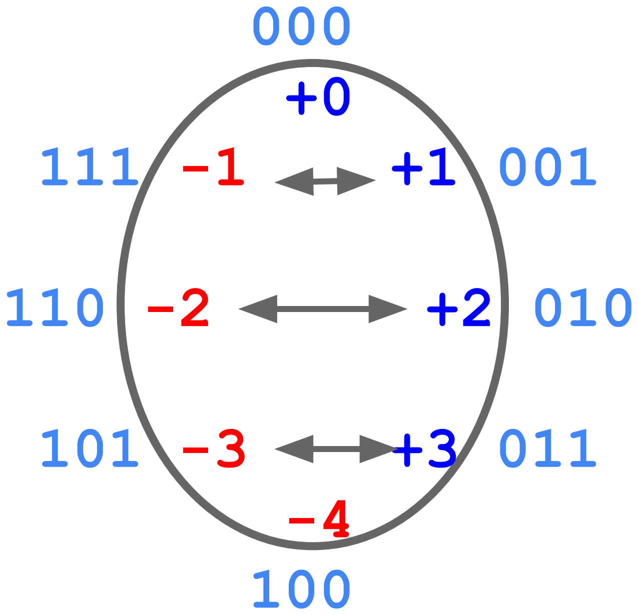
`Logically`

```
  000
- 001
  111
```



Two's complement

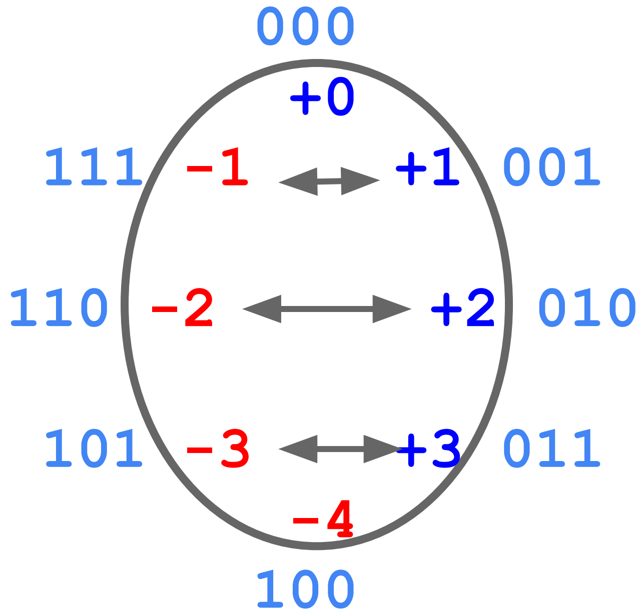
For 1, 2, 3 we can complement the value to show its negative



Two's complement

For -4, we need a different strategy

```
  -2 =   110
+ -2 = +110
  -4     100
```



Recall a byte Java ranges from
-128 to +127

Characteristics of interest

	Signed magnitude	One's complement	Two's complement
Same rule for addition	No	No (simple rule)	Yes
How many ways can you represent zero	2	2	1
Relative quantity of positive and negative values	Same	Same	One extra negative value

Has a computer been built using two’s complement?

Has a computer been built using two's complement?

Most modern computers use two's complement

Overflow

Two’s complement is

finite precision arithmetic

Therefore it can overflow

Two ways to detect

1. Intuitive
2. Mathematical

Overflow

1. Two numbers with like signs and a different sign results

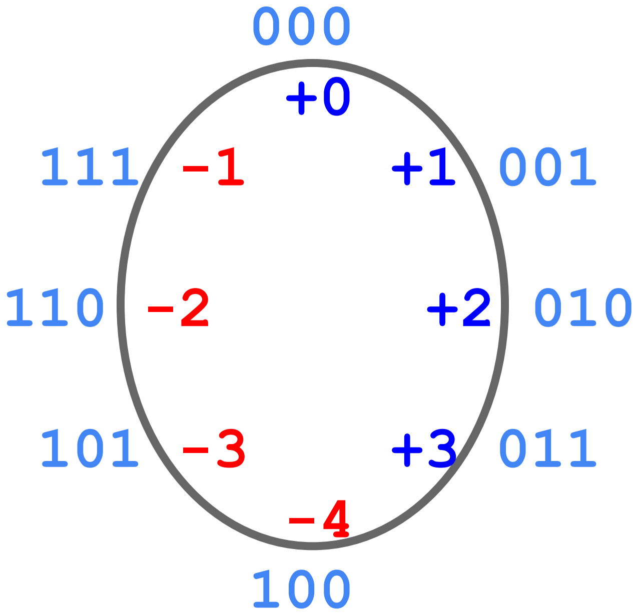
The sum of 2 positive numbers should be positive.

The sum of 2 negative numbers should be negative.

Overflow

1. Two numbers with like signs and a different sign results

```
+2  =  010
+2  =  010
```

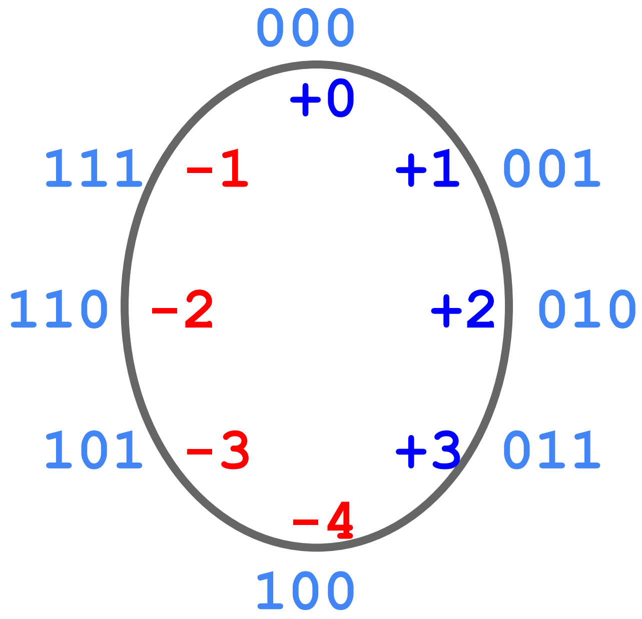


Overflow

1. Two numbers with like signs and a different sign results

```
        1
+2  =  010
+2  =  010
       ---
       100
```

sign change



Overflow

1. Two numbers with like signs and a different sign results

The sum of positive and negative has a magnitude smaller than the largest ⇒ magnitude decreases

Overflow

1. Two numbers with like signs and a different sign results

The sum of positive and negative has a magnitude smaller than the largest ⇒ magnitude decreases

So, overflow can never happen here.

Overflow

1. Two numbers with like signs and a different sign results

The sum of positive and negative has a magnitude smaller than the largest ⇒ magnitude decreases

Can be expressed mathematically. Look at sign bit column

2. If carry-out of sign ≠ carry-in an overflow occurred

Overflow

We can check this both ways.

1. Two numbers with like signs and a different sign results
2. If carry-out of sign ≠ carry-in an overflow occurred

```
        0 1
+2 =      0 1 0
+2 =      0 1 0
          -----
          1 0 0
```

Two's complement in Hex

```
 05
+FF
```

Two's complement in Hex

```
11
 05
+FF
 04
```

Is this correct?

Two's complement in Hex

```
 11
  05
+FF
  04
```

hex	binary	sign
0-7	0000-0111	pos
8-F	1000-1111	neg

High-order digit of hex tells us the sign.

If the number of bits is a multiple of 4.

Which it usually is.

Two's complement in Hex

```
 11
  05 = +5
+FF = -1
  04   +4
```

This is 1111 1111
That's an 8-bit
value for -1.

hex	binary	sign
0-7	0000-0111	pos
8-F	1000-1111	neg

What's the big idea?

Two's Complement does not need separate hardware for subtraction!

$$A - B = A + B + 1$$

What's the big idea?

Two's Complement does not need separate hardware for subtraction!

A - B = A + B + 1

```
  0100
 -0101
  1111
```

Say A == 4 and
B == 5

A subtract gives you a
two's complement -1

What's the big idea?

Two's Complement does not need separate hardware for subtraction!

A - B = A + B + 1

```
 0100           0100
-0101          +1010
 1111              1
                ----
                1111
```

Adding a two's complement –B gives the same result.

With Two's Complement the same hardware add instruction works for

- **unsigned** or
- **signed** numbers

Part of a C program

```
unsigned U = 10;

if (U > -1) printf ("U greater than -1");

if (U < -1) printf ("U less than -1 ");
```

C chooses to treat the test as unsigned.

```
U less than -1
```

$10_{10} = 000A_{16}$ **and** $-1_{10} = FFFF_{16}$

Program compared an unsigned number (10) to a signed number.
No hardware support for that ... so it converted -1 to unsigned.