Integers and Integer Operators

\[101010\]

(decimal 34)
Topic 11: Integers and Integer Ops

- Base $b$
- Binary and hexadecimal
- Binary addition
- Overflow
- Signed integers
- Bitwise logical operators
- Shift
Some Jokes...

https://www.youtube.com/watch?v=Fmb3TCvIETk
Decimal Numbers (“base 10”)

- What is this number?
  10

- What does it mean?

- In general, explain how to interpret the any sequence of decimal digits:
  1234
  9999
  2500000
Decimal Numbers ("base 10")

ten thousands  thousands  hundreds  tens  ones

1  2  3  4  5
Decimal Numbers ("base 10")

10^4  10^3  10^2  10^1  10^0
Decimal Numbers ("base 10")

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^4$</td>
<td>$10^3$</td>
<td>$10^2$</td>
<td>$10^1$</td>
<td>$10^0$</td>
<td></td>
</tr>
</tbody>
</table>

$1\times10^4 + 2\times10^3 + 3\times10^2 + 4\times10^1 + 5\times10^0$
### Decimal Numbers ("base 10")

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

\[
1 \times 10^4 + 2 \times 10^3 + 3 \times 10^2 + 4 \times 10^1 + 5 \times 10^0 \\
10000 + 2000 + 300 + 40 + 5
\]
Decimal Numbers ("base 10")

\[
\begin{array}{ccccc}
1 & 2 & 3 & 4 & 5 \\
10^4 & 10^3 & 10^2 & 10^1 & 10^0 \\
1 \times 10^4 & + & 2 \times 10^3 & + & 3 \times 10^2 & + & 4 \times 10^1 & + & 5 \times 10^0 \\
10000 & + & 2000 & + & 300 & + & 40 & + & 5 \\
12345
\end{array}
\]
Decimal Numbers ("base 10")

• What are the valid digits?
  0 . . . 9

• Why?
  9 + 1 = 10
Decimal Numbers (“base 10”)

Adding 1 to any 9 can be expressed as 1 in the next column.

\[
\begin{array}{c}
9 \\
\hline
1 \\
\hline
10
\end{array} + \begin{array}{c}
9 \\
\hline
1 \\
\hline
10
\end{array} + \begin{array}{c}
1 \\
\hline
1 \\
\hline
10
\end{array} + \begin{array}{c}
1 \\
\hline
1 \\
\hline
2
\end{array} = \begin{array}{c}
1 \\
\hline
1 \\
\hline
2
\end{array} \begin{array}{c}
0 \\
\hline
0 \\
\hline
7
\end{array}
Number Capacity

- A 5-digit number has the range:
  \[ 0 - 99,999 \]

- There are \( 100,000 = 10^5 \) possible values.
Octal Numbers ("base 8")

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 \\
8^4 & 8^3 & 8^2 & 8^1 & 8^0 \\
1 \times 8^4 & + & 2 \times 8^3 & + & 3 \times 8^2 & + & 4 \times 8^1 & + & 5 \times 8^0 \\
4096 & + & 1024 & + & 192 & + & 32 & + & 5 \\
5349
\end{array}
\]
Octal Numbers ("base 8")

\[ 12345_{\text{oct}} = 5349_{\text{dec}} \]
Octal Numbers ("base 8")

• Adding 1 to any 7 can be expressed as 1 in the next column.

\[
\begin{array}{c}
\text{7 \_ oct} \\
+ \hspace{1cm} \text{1 \_ oct} \\
\hline
\text{1 \_ oct}
\end{array}
\quad \begin{array}{c}
\text{7 0 \_ oct} \\
+ \hspace{1cm} \text{1 0 \_ oct} \\
\hline
\text{1 0 0 \_ oct}
\end{array}
\quad \begin{array}{c}
\text{1 0 \_ oct} \\
+ \hspace{1cm} \text{1 7 7 \_ oct} \\
\hline
\text{2 0 7 \_ oct}
\end{array}
\]
Topic 11: Integers and Integer Ops

- Base $b$
- Binary and hexadecimal
- Binary addition
- Overflow
- Signed integers
- Bitwise logical operators
- Shift
Binary and Hexadecimal

• **Binary: base 2**
  - It's how a computer thinks!
  - **Symbols:** 0 1

• **Hexadecimal: base 16**
  - Express binary values more compactly
  - **Symbols:** 0 1 2 3 4 5 6 7 8 9 a b c d e f
Binary Numbers

\[
\begin{array}{cccccc}
2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
1 & 0 & 1 & 1 & 1
\end{array}
\]

\[
1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0
\]

\[
16 + 0 + 2 + 1
\]

\[
23
\]
A Full Byte

0 1 1 1 1 1 1 1

\[ 0 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \]

\[ 64 + 32 + 16 + 8 + 4 + 2 + 1 \]

127
Binary ("base 2")

- What are the valid symbols?
  
  0, 1

- How to calculate the value?
  
  - Each column is worth a power of 2

- How many numbers (including zero) with $d$ bits?
  
  $2^d$

  2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, ...
Max Value in 7 bits

We're not going to use the highest bit until a little later!
Half a Byte is a Nibble

Upper Nibble

Lower Nibble

0 1 1 1 1 1 1 1 1 1 1 1
Half a Byte is a Nibble

4 bits
$2^4 = 16$ possible values

Same as 1 hexadecimal symbol!
Hex = Easy Binary

Upper Nibble

0111 \_bin = 7 \_dec = 7 \_hex

Lower Nibble

1111 \_bin = 15 \_dec = f \_hex
Hex = Easy Binary

Prefixes tell Java how to interpret literals

Java Integer Literals:
- binary: \( \text{0b}0111\_1111 \)
- hexadecimal: \( \text{0x}7f \)
- decimal: 127
- octal: 0177
Hex = Easy Binary

Java Integer Literals:

<table>
<thead>
<tr>
<th>Binary</th>
<th>Hexadecimal</th>
<th>Decimal</th>
<th>Octal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0111_1111</td>
<td>0x7f</td>
<td>127</td>
<td>0177</td>
</tr>
</tbody>
</table>

Added in Java 7
Hex = Easy Binary

1 hex digit = 4 bits

<table>
<thead>
<tr>
<th>Type</th>
<th>Hex Digits</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>2</td>
</tr>
<tr>
<td>short</td>
<td>4</td>
</tr>
<tr>
<td>int</td>
<td>8</td>
</tr>
<tr>
<td>long</td>
<td>16</td>
</tr>
<tr>
<td>char</td>
<td>4</td>
</tr>
</tbody>
</table>
Topic 11: Integers and Integer Ops

- Base $b$
- Binary and hexadecimal
- Binary addition
- Overflow
- Signed integers
- Bitwise logical operators
- Shift
How Does Addition Work?

- Add each column
- If the sum is more than the max symbol, carry
  - Carries can repeat, domino-style
Decimal Addition

Let's look at decimal addition first.

When we understand how it works, it will be easier to learn binary.
Decimal Addition

We add each column. Some of the columns end up with results that are too large.
Decimal Addition

We solve too-large columns by using a carry.

To carry, you subtract 10 from the column, and add 1 to the next column.
We solve too-large columns by using a carry.

To carry, you subtract 10 from the column, and add 1 to the next column.
Decimal Addition

3 9 9 9 9

+ 0 0 0 0 1

Carrying can ripple through a number, affecting many columns.
Decimal Addition

Carrying can ripple through a number, affecting many columns.
Decimal Addition

Carrying can ripple through a number, affecting many columns.
Decimal Addition

Carrying can ripple through a number, affecting many columns.
Decimal Addition

Carrying can ripple through a number, affecting many columns.
Binary Addition

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Binary addition works by the exact same principles. The only difference is that each column has a max value of 1, instead of 9.
Binary Addition

A column is too large; carrying is required.
Binary Addition

\[ \begin{array}{cccccc}
1 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 \\
\end{array} \]

\[ + \]

\[ \begin{array}{cccccc}
1 & 1 & 0 & 1 & 1 \\
\end{array} \]

A column is too large; carrying is required.
Binary Addition

Does binary addition really work?

Let's check by converting the three numbers to decimal.
Binary Addition

\[
\begin{array}{cccc}
1 & 0 & 1 & 0 \\
+ & 0 & 0 & 1 & 1 \\
\hline
1 & 1 & 0 & 1 \\
\end{array}
\]

\[
8 + 2 = 10 \\
2 + 1 = 3 \\
8 + 4 + 1 = 13
\]
Topic 11: Integers and Integer Ops

- Base $b$
- Binary and hexadecimal
- Binary addition
- **Overflow**
- Signed integers
- Bitwise logical operators
- Shift
Each binary number has a fixed number of digits.

What happens if we add and carries happen – but we run out of digits?
Overflow

Each binary number has a fixed number of digits.

What happens if we add and carries happen – but we run out of digits?
Overflow

Each binary number has a fixed number of digits.

What happens if we add and carries happen – but we run out of digits?
Overflow

Each binary number has a fixed number of digits.

What happens if we add and carries happen – but we run out of digits?
Overflow

We imagine that there is a temporary bit that we can carry into...

```
1 1 1 1 1
+ 0 0 0 1
```

```
0 2 0 0 0 0
```
We imagine that there is a temporary bit that we can carry into...
Overflow

...but then we discard this extra bit.

This is called overflow.
Topic 11: Integers and Integer Ops

- Base \( b \)
- Binary and hexadecimal
- Binary addition
- Overflow
- Signed integers
- Bitwise logical operators
- Shift
Remember this?

| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

This is our **SIGN BIT**.

\[
0111_{\text{bin}} - 1111_{\text{bin}} = 127_{\text{dec}}
\]
A Signed Byte

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
</table>

- $2^7 = -128$

This bit is worth -128!
A Signed Byte

\[-2^7 \quad 2^6 \quad 2^5 \quad 2^4 \quad 2^3 \quad 2^2 \quad 2^1 \quad 2^0\]

\[-128 \quad 64 \quad 32 \quad 16 \quad 8 \quad 4 \quad 2 \quad 1\]

\[-1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0\]

\[-128 + 64 + 32 + 16 + 8 + 4 + 2 + 1\]

\[-1\]
A Signed Byte

1 1 1 1 1 1 1 1 1 1

-1
When Overflow is Good

\[
\begin{array}{cccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}
\]
When Overflow is Good

$$\begin{array}{cccccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\hline
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 \\
\end{array}$$
When Overflow is Good

\[
\begin{array}{cccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{cccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{cccccccccc}
0 & \leftarrow 1 & \leftarrow 1 & \leftarrow 1 & \leftarrow 1 & \leftarrow 1 & \leftarrow 1 & \leftarrow 1 & \leftarrow 1 & \leftarrow 2 \\
\end{array}
\]
When Overflow is Good

\[\begin{array}{cccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
+ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}\]
When Overflow is Good

Because of overflow,

\[-1 + 1 = 0\]
When Overflow is Bad

\[\begin{array}{cccccccccc}
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\hline \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{array}\]
When Overflow is Bad

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The result shows an overflow, indicated by the carry-out of the 12th bit.
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

When Overflow is Bad
When Overflow is Bad

\[ \text{In byte arithmetic,} \]

\[ 127 + 1 = -128 \]
Negative Numbers

\[
\begin{array}{cccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

-128

a seven-bit unsigned integer

\[-128 + 0 = -128\]
Negative Numbers

-128 + 1 = -127

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0

-128 a seven-bit unsigned integer
Negative Numbers

-128 + 2 = -126

-128

a seven-bit unsigned integer
Negative Numbers

-128

1 0 0 0 0 0 0 1 1

a seven-bit unsigned integer

-128 + 3 = -125
Negative Numbers

-128 + 127 = -1

a seven-bit unsigned integer
### Smallest Number:
- **-128**
  - Binary: \( 10000000 \)
- **-1**
  - Binary: \( 1111111 \)
- **0**
  - Binary: \( 0000000 \)

### Largest Number:
- **127**
  - Binary: \( 1111111 \)
short

Smallest Number: -32768

-1

0

Largest Number: 32767

15 positive bits

-32768

0 0 0 0

1 1 1 1

0 0 0 0

0 1 1 1

0 0 0 0

0 1 1 1
int

-2^31

Smallest Number: -2^31

0

Largest Number: 2^31-1

31 positive bits

1 0 0 0 0 ...

1 1 1 1 1 ...

0 0 0 0 0 ...

0 1 1 1 1 ...

0 0 0 0 0 ...

1 1 1 1 1 ...
long

-2^{63}

Smallest Number: -2^{63}

-1

0

Largest Number: 2^{63}-1

63 positive bits

-1

0

0

0
Topic 11: Integers and Integer Ops

- Base \( b \)
- Binary and hexadecimal
- Binary addition
- Overflow
- Signed integers
- Bitwise logical operators
- Shift
Bitwise Logical Operators

• Remember the boolean logical operators?
  &&  ||  !

• Bitwise logical operators do the same, but with bits:
  &  |  ~
& - the Bitwise And Operator

Every set of bits is treated as its own boolean equation.

0 represents false
1 represents true
\& - the Bitwise And Operator

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

false AND true
result: false
### & - the Bitwise And Operator

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

true AND false

result: false
**& - the Bitwise And Operator**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**true AND true**

**result: true**
- the Bitwise Or Operator

Vertical bar \(|\) is the bitwise OR operator.
~ - the Bitwise Not Operator

Tilde ~ is the bitwise NOT operator.

<table>
<thead>
<tr>
<th>~</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
A Number and its Bitwise Negation

Doing bitwise NOT on zero gives an interesting result. What is it?

0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0

1 1 1 1 1 1 1 1 1 1 1 1
A Number and its Bitwise Negation

Bitwise NOT of zero equals negative one!

0 0 0 0 0 0 0 0 0 0 0 0

0

1 1 1 1 1 1 1 1 1 1 1 1

-1
A Number and its Bitwise Negation

Bitwise NOT of four equals negative five...
A Number and its Bitwise Negation

In general, any number, plus its bitwise negation, equals:

-1

Amazing.
A Number and its Bitwise Negation

\[ x + \sim x = -1 \]

\[ -x = \sim x + 1 \]

\[ a - x = a + (\sim x) + 1 \]

This allows us to figure out how to negate a number:
- Do bitwise negation
- Add one

This also tells us how we can do subtraction in binary.
Topic 11: Integers and Integer Ops

- Base $b$
- Binary and hexadecimal
- Binary addition
- Overflow
- Signed integers
- Bitwise logical operators
- Shift
Bit Shifting

- Bit shift operators move the bits around inside an integer.
  - `<<` Left shift
  - `>>` Right shift (sign extend)
  - `>>>` Right shift (zero extend)
Doubling a Number

What happens when you add a number to itself in binary?

```
  0 0 1 1
+ 0 0 1 1
```

```
  0 0 2 2
```
Doubling a Number

What happens when you add a number to itself in binary?

It ends up just shifting every bit one space to the left.
Left Shift

• Left Shift does quick multiplication
  - Shift by 1 bit = multiply by 2
  - Shift by 2 bits = multiply by 4
  - Shift by 8 bits = multiply by 256
<< - the Left Shift Operator

0 0 0 0 1 0 0 1

<< 2

0 0 1 0 0 1 1 0 0 0
<< - the Left Shift Operator

0 0 0 0 1 0 0 1

9

* 4

0 0 1 0 0 1 0 0 0

36
Right Shift

0 0 0 0 1 0 0 1

0 0 0 0 0 0 0 1 1 0 0
Right Shift

0 0 0 0 1 0 0 1

9 / 4

0 0 0 0 0 0 0 1 0 0
This is one of the reasons why computers round integer division **down**.
Right Shift of Negative Numbers?

This number is negative.
>> - the Signed Right Shift Operator

\[
\begin{array}{cccccccc}
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1
\end{array}
\]

\[
\begin{array}{cccccccc}
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}
\]
>> - the Signed Right Shift Operator

Note that this is rounding down again. The precise answer is -29.75
>>> - the Unsigned Right Shift Operator

1 0 0 0 1 0 0 0

0 0 1 0 0 0 0 1 0

>>>  2
In this slide, we're interpreting the numbers as unsigned integers.

### >>>> - the Unsigned Right Shift Operator

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
</table>

137

/ 4

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
</table>

34
int val = ... 
int low = val & 0x0f;
int rest = val >>> 4;
System.out.println("val="+val);
System.out.println("low="+low);
System.out.println("rest="+rest);
Topic 11: Integers and Integer Ops

- Base $b$
- Binary and hexadecimal
- Binary addition
- Overflow
- Signed integers
- Bitwise logical operators
- Shift

Summary