Representing Numerical Data Integers

Reading: Chapter 5.1-5.2
(except binary coded decimal
and 10's complement)

Representing Numbers

- If we attempt to add numbers stored as characters, simple binary addition does not provide the correct result

Character representation of 3 → first row

Character representation of 0 → second row

	0	1	1	0	0	1	1
+	0	1	1	0	0	0	0
	1	1	0	0	0	1	1

Binary addition results in 63_{16} which is c

3 is represented as 33_{16} and 0 is represented as 30_{16}

Copyright 2010-2011 John Wiley & Sons, Inc. & Robert F. Kelly

Number Representation

- We need to use a non-character representation to enable mathematical operations (e.g., addition)
- Numbers can be integers (e.g., 3) or real (floating point) numbers (e.g., 2.75)
- Numbers can be represented as a combination of
  - Value or magnitude
  - Sign (plus or minus)

We cover floating point numbers in the next session

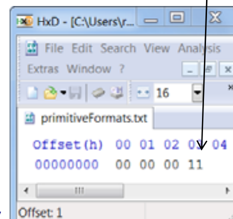
Copyright 2010-2011 John Wiley & Sons, Inc. & Robert F. Kelly 5-3

Example

```
package lectures;
import java.io.*;
import java.lang.Object.*;
public class GeneralBinaryCreator {
    public static void main(String[] args) throws IOException {
        FileOutputStream out = null;
        File f = new File("primitiveFormats.txt");
        boolean result = f.createNewFile();
        out = new FileOutputStream("primitiveFormats.txt");
        ByteArrayOutputStream byte_out = new ByteArrayOutputStream();
        DataOutputStream data_out = new DataOutputStream(byte_out);
        int c = 17;
data_out.writeInt(c);
        byte[] b = byte_out.toByteArray();
        out.write(b);
        out.close();
    }
}
```

Program writes the Java int 17 as a binary value (not a string)

Note the hex representation of 17

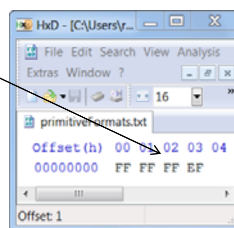


DataOutputStream does not always write the exact binary representation of a Java primitive

Copyright 2010-2011 John Wiley & Sons, Inc. & Robert F. Kelly 5-4

Example (continued)

- If we change the value of c in the program to -17, notice the new hex value



What is FF FF FF EF + 11 in hex arithmetic?

Is it the same as (-17+17)

Copyright 2010-2011 John Wiley & Sons, Inc. & Robert F. Kelly

Unsigned Numbers: Integers

- Unsigned whole number or *integer*
- Direct *binary* equivalent of decimal integer
  - 8 bits: 0 to 99
  - 16 bits: 0 to 9,999
  - 32 bits: 0 to 99,999,999

Decimal	Binary
68	= 0100 0100 = $2^6 + 2^2 = 64 + 4 =$ ***68***
255 (largest 8-bit binary)	= 1111 1111 = $2^8 - 1 =$ ***255***

Copyright 2010-2011 John Wiley & Sons, Inc. & Robert F. Kelly

Signed-Integer Representation

- No obvious direct way to represent the sign in binary notation
- Options:
  - Sign-and-magnitude representation
  - 1's complement
  - 2's complement (most common)

Copyright 2010-2011 John Wiley & Sons, Inc. & Robert F. Kelly

Sign-and-Magnitude

- Use left-most bit for sign
  - 0 = plus; 1 = minus

Not used in practice (difficult to implement in HW)

- Total range of integers the same
  - Half of integers positive; half negative
  - Magnitude of largest integer half as large
- Example using 8 bits:
  - Unsigned: 1111 1111 = +255
  - Signed: 0111 1111 = +127
    1111 1111 = -127
  - Note: 2 values for 0:
    +0 (0000 0000) and -0 (1000 0000)

Copyright 2010-2011 John Wiley & Sons, Inc. & Robert F. Kelly

What Should Integers Do

- Leading bit tells us the sign of the integer
- The negative of a negative integer is the original integer (i.e., -(-17) is 17)
- x – y gives the same result as x + -y (i.e., 5 - 3 result is the same as 5 + -3)
- Negative and positive numbers are treated the same in integer operations (e.g., multiplication)

Copyright 2010-2011 John Wiley & Sons, Inc. & Robert F. Kelly 5-9

Complement Representation

- Sign of the number does not have to be handled separately
- Consistent for all different signed combinations of input numbers
- Two methods
  - 1's complement - overflow bits are carried around back into the sum
  - 2's complement - overflow bits are discarded

Copyright 2010-2011 John Wiley & Sons, Inc. & Robert F. Kelly 5-10

Overflow

- Fixed word size has a fixed range size
- Overflow: combination of numbers that adds to result outside the range
- End-around carry in modular arithmetic avoids problem
- Complementary arithmetic: numbers *out of range* have the opposite sign
  - Test: If both inputs to an addition have the same sign and the output sign is different, an overflow occurred

Copyright 2010-2011 John Wiley & Sons, Inc. & Robert F. Kelly

Overflow in Java

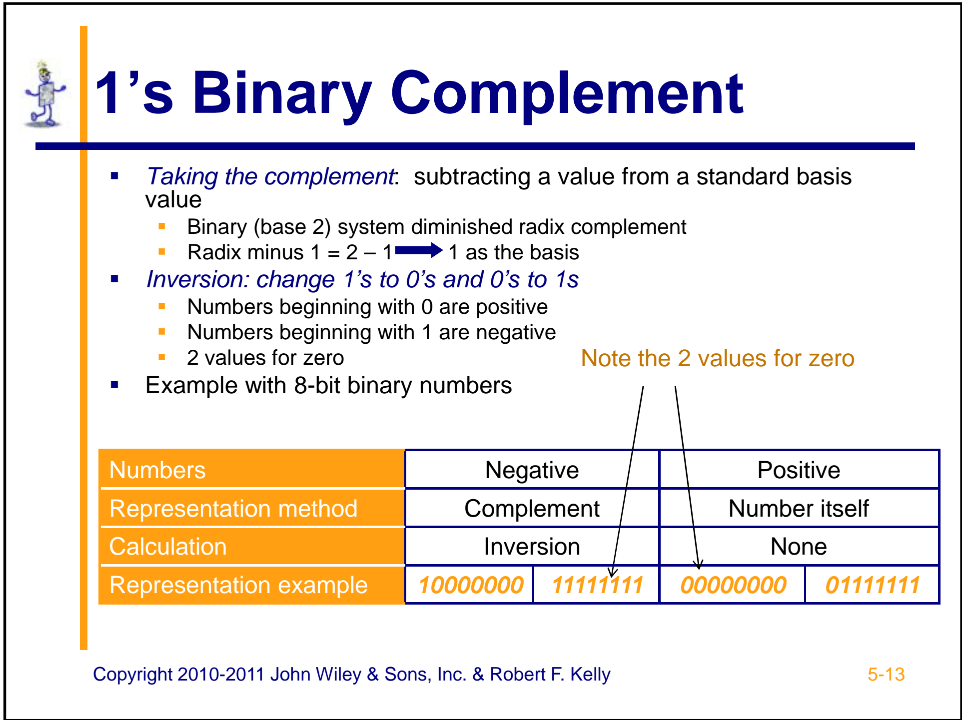
- Example causes an overflow of a short
- Notice that the overflow causes a wrap

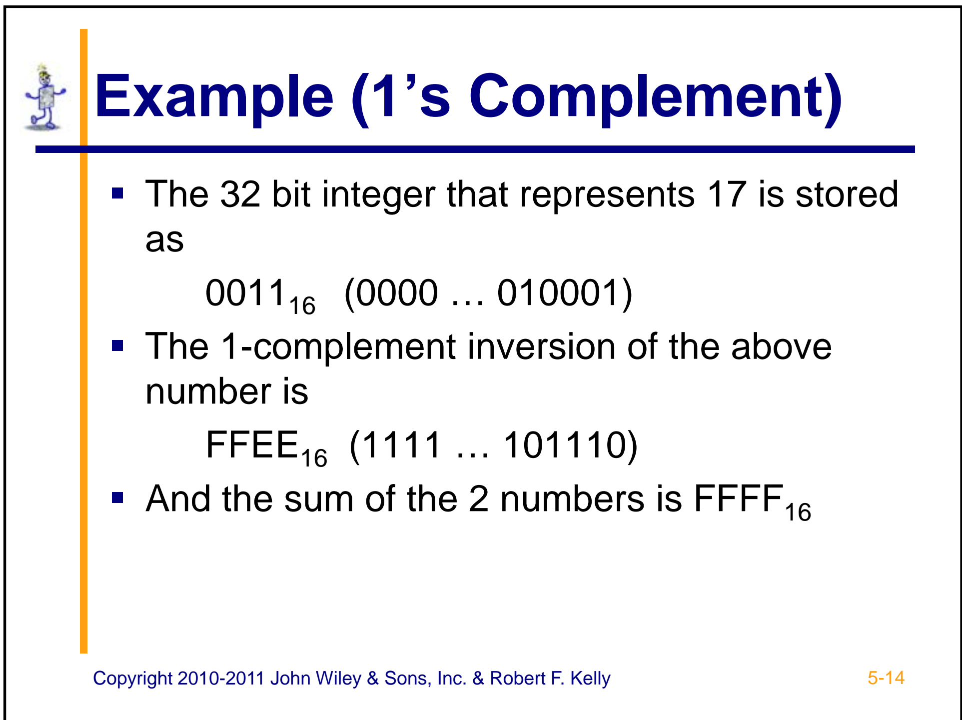
```
1   package lectures;
2
3   import java.io.*;
4
5   public class IntOverflow {
6
7       public static void main(String[] args)
8           short s = 32763;
9           for (short i = 0; i < 10; i++) {
10              s++;
11              System.out.println(s);
12          }
13      }
14  }
```

Output - CodeISE218 (run)

```
run:
32764
32765
32766
32767
-32768
-32767
-32766
-32765
-32764
-32763
BUILD SUCCESSFUL (total time: 1 second)
```

Copyright 2010-2011 John Wiley & Sons, Inc. & Robert F. Kelly


1's Binary Complement
Taking the complement: subtracting a value from a standard basis value
Binary (base 2) system diminished radix complement
Radix minus 1 = 2 – 1 1 as the basis
Inversion: change 1's to 0's and 0's to 1s
Numbers beginning with 0 are positive
Numbers beginning with 1 are negative
2 values for zero
Note the 2 values for zero
Example with 8-bit binary numbers
Numbers
Negative
Positive
Representation method
Complement
Number itself
Calculation
Inversion
None
Representation example
10000000
11111111
00000000
01111111
Copyright 2010-2011 John Wiley & Sons, Inc. & Robert F. Kelly
5-13


Example (1's Complement)
The 32 bit integer that represents 17 is stored as
0011₁₆ (0000 … 010001)
The 1-complement inversion of the above number is
FFEE₁₆ (1111 … 101110)
And the sum of the 2 numbers is FFFF₁₆
Copyright 2010-2011 John Wiley & Sons, Inc. & Robert F. Kelly
5-14

Addition

- Add 2 positive 8-bit numbers
- Add 2 8-bit numbers with different signs
  - Take the 1's complement of 58 (i.e., invert)
    0011 1010
    1100 0101

0010 1101	=	45
0011 1010	=	58
0110 0111	=	103

0010 1101	=	45
1100 0101	=	–58
1111 0010	=	**–13**

Invert to get magnitude

0000 1101

8 + 4 + 1 = 13

Copyright 2010-2011 John Wiley & Sons, Inc. & Robert F. Kelly

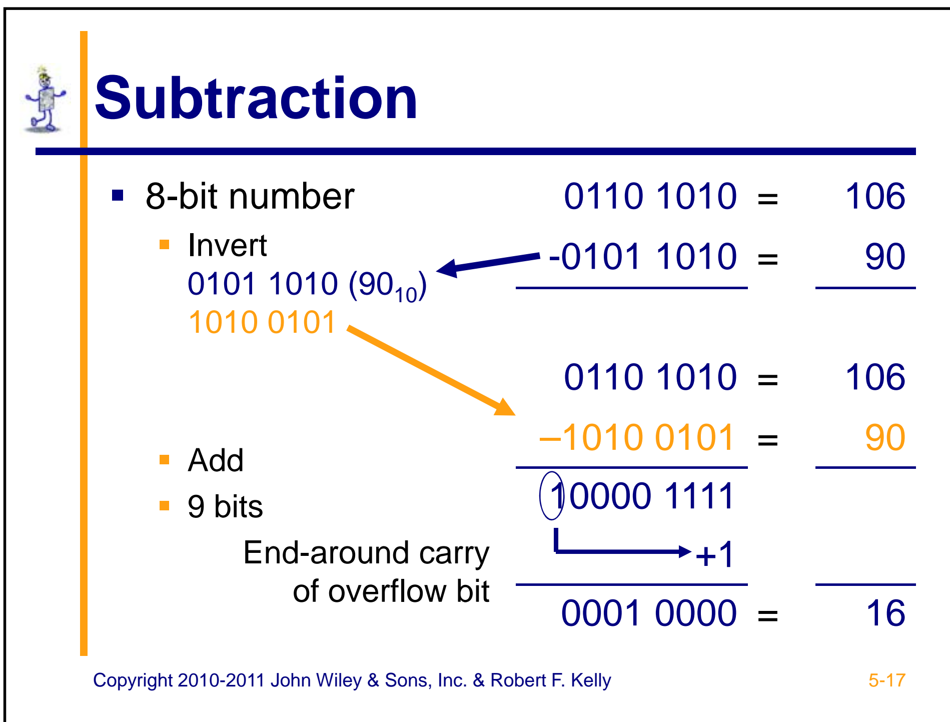
Addition with Carry

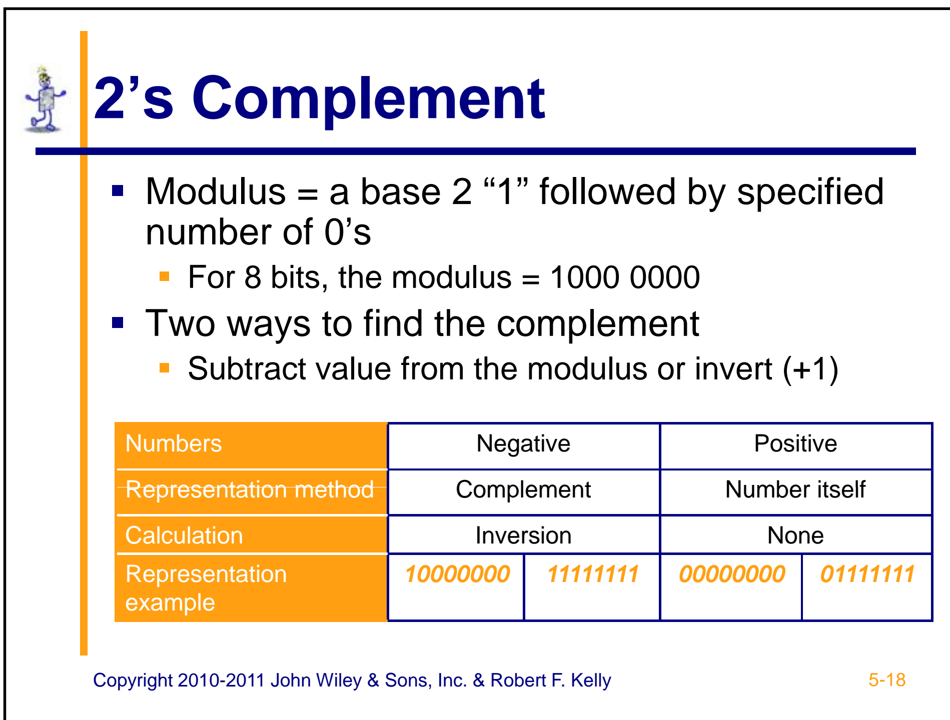
- 8-bit number
  - Invert
    0000 0010 (2_{10})
    1111 1101
  - Add
  - 9 bits

0110 1010	=	106
1111 1101	=	–2
10110 0111		
+1		
0110 1000	=	104

End-around carry of overflow bit

Copyright 2010-2011 John Wiley & Sons, Inc. & Robert F. Kelly


Subtraction
8-bit number
Invert
0101 1010 (90₁₀)
1010 0101
Add
9 bits
End-around carry
of overflow bit
0110 1010 = 106
-0101 1010 = 90
0110 1010 = 106
–1010 0101 = 90
10000 1111
+1
0001 0000 = 16
Copyright 2010-2011 John Wiley & Sons, Inc. & Robert F. Kelly
5-17


2's Complement
Modulus = a base 2 "1" followed by specified number of 0's
For 8 bits, the modulus = 1000 0000
Two ways to find the complement
Subtract value from the modulus or invert (+1)
Numbers
Negative
Positive
Representation method
Complement
Number itself
Calculation
Inversion
None
Representation example
10000000
11111111
00000000
01111111
Copyright 2010-2011 John Wiley & Sons, Inc. & Robert F. Kelly
5-18

Example (1's Complement)

- The 32 bit integer that represents 17 is stored as

  0011_{16} (0000 … 010001)
- The 2's-complement of the above number is

  $FFEE_{16}$ (1111 … 101111)
- And the sum of the 2 numbers is 0000_{16}

2's complement formed by inverting original number and adding 1

Copyright 2010-2011 John Wiley & Sons, Inc. & Robert F. Kelly