Computer Organization

Boolean Arithmetic

Boolean Arithmetic

- As seen in the diagram of the ALU, we need to create additional ways to perform operations on binary numbers
 - subtraction
 - multiplication

Addition is the foundation for all arithmetic operations, so we can
use it to create the others

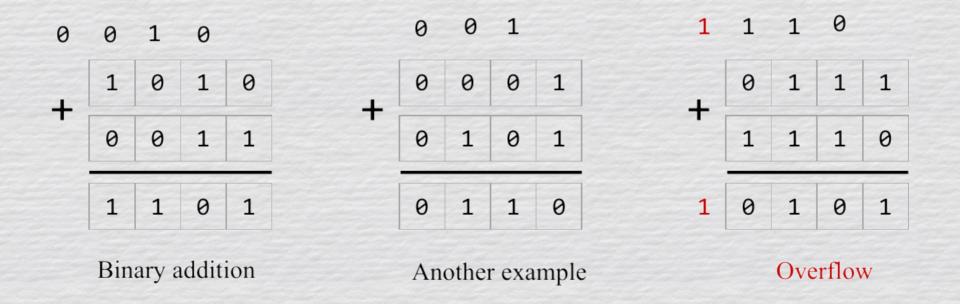
Addition

Binary addition

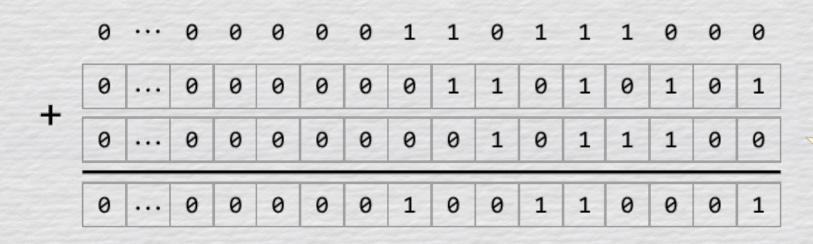
Decimal addition

Addition – cont.

 Computers represent integers using a fixed number of bits, sometimes called "word size"



Addition – cont.



Same addition algorithm for any *n*

Signed & Unsigned Numbers

- Many programming languages initiate numbers using specific data types
 - 16 bit short
 - 32 bit integer
 - 64 bit long

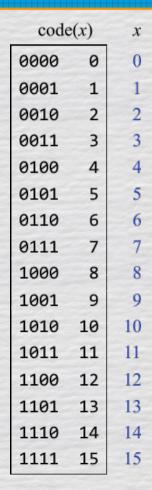
These data types can be set as signed or unsigned

Signed & Unsigned Numbers – cont.

 A data type set to a specific word number (how many bits it represents) has a specific range

- Signed data types have a range of -2^{n-1} , ..., -1, 0, 1, ..., $2^{n-1} 1$
 - Negative, 0, Positive
- Unsigned data types have a range of 0 ... 2ⁿ 1
 - 0, Positive (strictly non-negative)

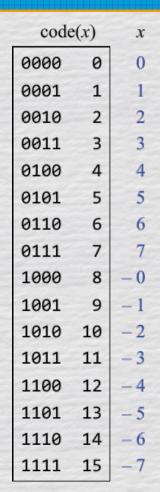
Unsigned Numbers



• Ex. - word size: *n* = 4

- Represents the full capacity of bit combinations (0 ... 15)
- Strictly non-negative values

Signed Numbers – Incorrect

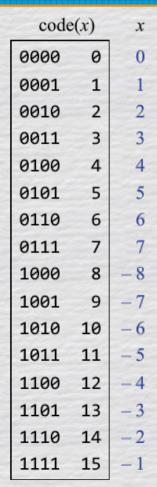


• Ex. - word size: *n* = 4

- Represents half the full capacity of bit combinations
 - (-7, ..., -1, -0, 0, 1, ..., 7)
 - Most Significant Bit (left-most) dictates polarity (sign)

- Contains -0 (we would have to revise our hardware to handle this)
 - Should NOT be used

Signed Numbers – Correct



• Ex. - word size: *n* = 4

- Represents half the full capacity of bit combinations
 - (-8, ..., -1, 0, 1, ..., 7)

- Perfectly compatible with existing hardware that assumes unsigned data
 - Does not contain a -0

Signed Numbers – Correct

```
code(x)
           x
0000
           0
                 The representation
0001
                    Assumption: Word size = n bits
0010
0011
                    The "two's complement" of x is defined to be 2^n - x
0100
                 • The negative of x is coded by the two's complement of x
9191
0110
0111
1000
         -8
                                                 From binary to decimal:
1001
                 From decimal to binary:
1010
         -6
                 if x \ge 0 return binary(x)
                                                if MSB = 0 return decimal(bits)
1011
      11
1100
      12
         -4
                       return binary (2^n - x)
                 else
                                                            return "-" and then (2^n - decimal(bits))
                                                 else
1101
      13
         -3
1110
      14
         -2
1111
         -1
```

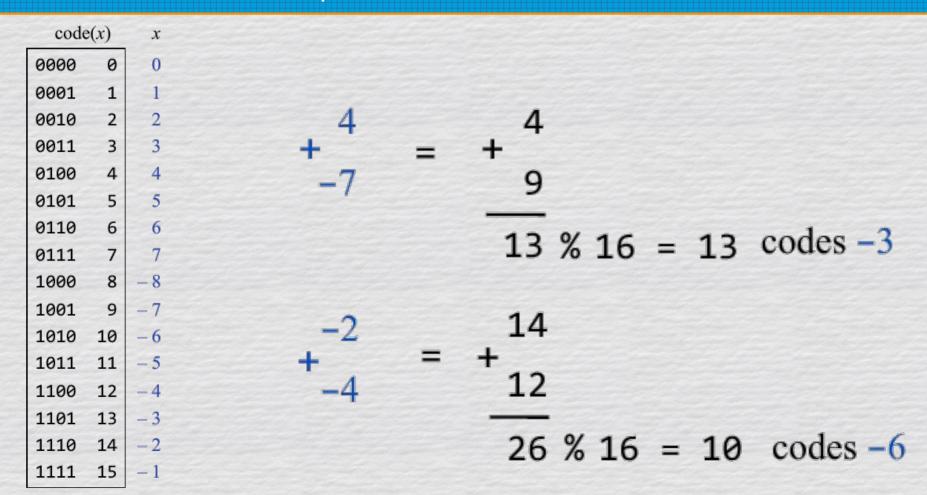
Two's Complement - Addition

```
code(x)
                       Algorithm: Regular addition, modulo 2^n
0000
9991
0010
9911
0100
                                        20 \% 16 = 4 \text{ codes } 4
9191
0110
0111
1000
                             -5
1001
                                          14 \% 16 = 14 \text{ codes } -2
1010
1011
       11
           -5
                                          14
1100
      12
           -4
1101
       13
           -3
1110
       14
                                          25 \% 16 = 9 \text{ codes } -7
1111
```

Two's Complement – Addition

code(x)		x			
0000	0	0			
0001	1	1			
0010	2	2	4		_
0011	3	3	+	=	7
0100	4	4	_7		•
0101	5	5	-,		
0110	6	6			
0111	7	7			
1000	8	-8			
1001	9	-7			
1010	10	-6	-2		0
1011	11	-5	+	=	1
1100	12	-4	-4		
1101	13	-3			
1110	14	-2			
1111	15	1			

Two's Complement - Addition



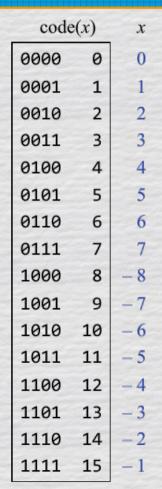
Two's Complement – Addition

code	e(x)	x						
0000	0	0	At the binary level (same algorithm):					
0001	1	1		2110	Q			
0010	2	2	+ 6 = +	9110		Ignoring the overflow bit		
0011	3	3		1110		is the binary equivalent of		
0100	4	4	1/0	0100	codes 4	modulo 2 ⁿ		
0101	5	5						
0110	6	6	3	0011				
0111	7	7	+ = +	1011				
1000	8	-8	- 5	1011				
1001	9	-7		1110	codes -2			
1010	10	-6						
1011	11	-5	the state of the s	1110				
1100	12	-4	+_5 = +	1011				
1101	13	-3		1011				
1110	14	-2	1	1001	codes -7			
1111	15	-1						

Two's Complement – Addition

```
code(x)
0000
                  At the binary level (same algorithm):
0001
                                0110
0010
                                1110
0011
0100
                               10100 codes 4
9191
0110
                   More examples:
0111
                                0101
1000
                            0111
1001
1010
                                       codes -4 5+7=-4 ???
                                1100
1011
     11
         -5
                                 1001
1100
     12
         -4
1101
     13
         -3
                                 1101
1110
     14
                                       codes 6 -7 + -3 = 6 ???
                              10110
1111
```

Two's Complement – Subtraction



• Simply consider that $x - y \equiv x + (-y)$

We just need to convert our second number and do simple addition

But, how do we convert the polarity of a number?

Two's Complement - Conversion

```
code(x)
0000
                   Insight: code(-x) = (2^n - x) = 1 + (2^n - 1) - x
0001
                                                  = 1 + (1111) - x
9919
9911
                                                  = 1 + flippedBits(x)
0100
9191
                   Algorithm: To convert bbb...b:
0110
0111
                                Flip all the bits and add 1 to the result
1000
1001
1010
         -6
                   Example: Convert 0010 (2)
1011
         -5
                                       1101 (flipped)
1100
     12
         -4
1101
     13
         -3
1110
     14 -2
                                       1110(-2)
1111
```

Two's Complement - Conversion

```
code(x)
0000
                Insight: code(-x) = (2^n - x) = 1 + (2^n - 1) - x
0001 1
0010
                                                 = 1 + (1111) - x
9911
0100
                                                 = 1 + flippedBits(x)
9191
0110
0111
                Algorithm: To convert bbb...b:
1000
                              Flip all the bits and add 1 to the result
1001
1010
1011
     11
        -5
1100
        -4
                Practice: Convert 1010 (-6)
1101
     13
1110
     14 | -2
1111
```

Two's Complement - Conversion

```
code(x)
0000
                Insight: code(-x) = (2^n - x) = 1 + (2^n - 1) - x
0001 1
0010
                                                = 1 + (1111) - x
9911
0100
                                                = 1 + flippedBits(x)
9191
0110
0111
                Algorithm: To convert bbb...b:
1000
                             Flip all the bits and add 1 to the result
1001
1010
        -6
1011
        -5
                                                    0101 (flipped)
1100
        -4
                Practice: Convert 1010 (-6) + 1
1101
     13
       -3
1110
     14 | -2
                                                    0110 (6)
1111
```