Mathematics 3670: Computer Systems
Bits, Data Types, and Operations

Dr. Andrew Mertz

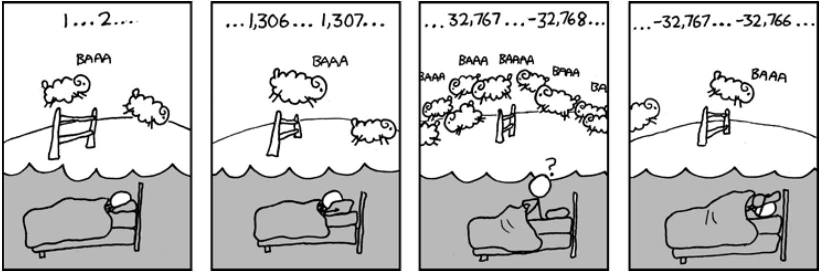
Mathematics and Computer Science Department
Eastern Illinois University

Fall 2012

Week 2: to do

What	**When**
Read Chapter 2	this week
Design Lab 2 TMs	before Thursday
Complete Lab 2	this Thursday
Submit Lab 2 work	by next Thursday

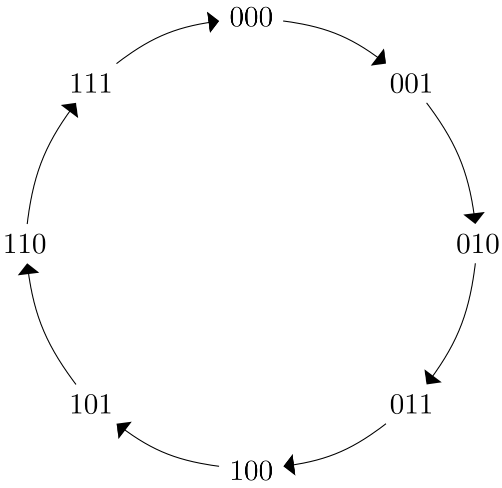
Geek humor



Source: http://xkcd.com/571/

3-bit codes: no assigned meaning

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



3 bits yield $2^3 = 8$ possibilities

Number of bit patterns

number of bits	number of bit patterns
3	$2^3 = 8$
4	$2^4 = 16$
...	...
m	2^m
...	...
16	$2^{16} = 65,536$
...	...
32	$2^{32} = 4,294,967,296$
...	...
64	$2^{64} = 18,446,744,073,709,551,616$
...	...

Representations

What is the meaning of 0011010111110010 ?

- An integer? If so, which representation?
- One or more characters? (ASCII or Unicode)
- A floating point value?
- A value of an enumeration type?
- Something else?

Shorthand notation: hexadecimal

Consider a bit string such as:	0011010111110010
Use 4-bit groups	0011 0101 1111 0010
Use hexadecimal digits:	3 5 F 2

pattern	0000–1001	1010	1011	1100	1101	1110	1111
hexadecimal	0–9	A	B	C	D	E	F

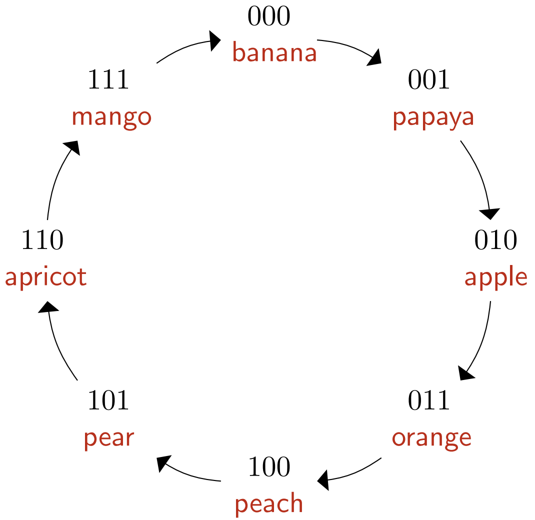
ASCII code

- **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
- ASCII uses a 7-bit code
- 7 bits allows for only $2^7 = 128$ different characters
- See http://highered.mcgraw-hill.com/sites/dl/free/0072467509/104653/PattPatelAppE.pdf

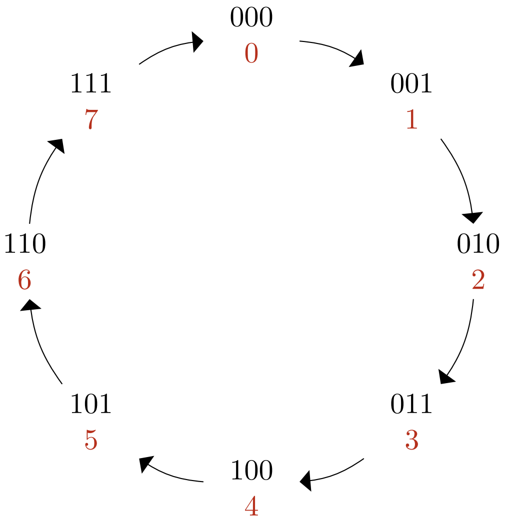
Unicode

- **One** system for all the world's languages
- Unicode uses a muti-byte code
- 2 bytes provides $2^{16} = 65,536$ different characters
- See http://www.unicode.org/charts/

Wheel of 3-bit codes: food choices (enumeration type)

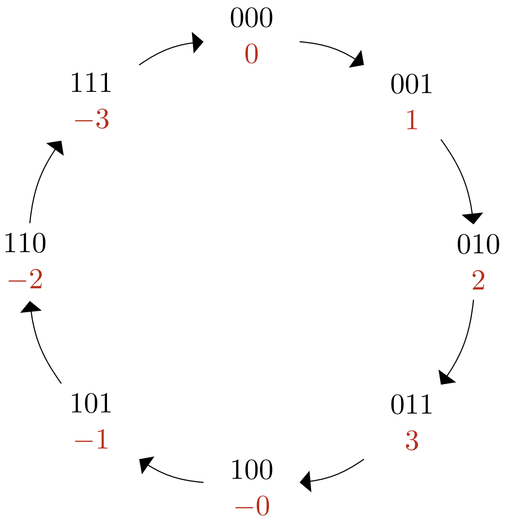


Wheel of 3-bit codes: unsigned integers



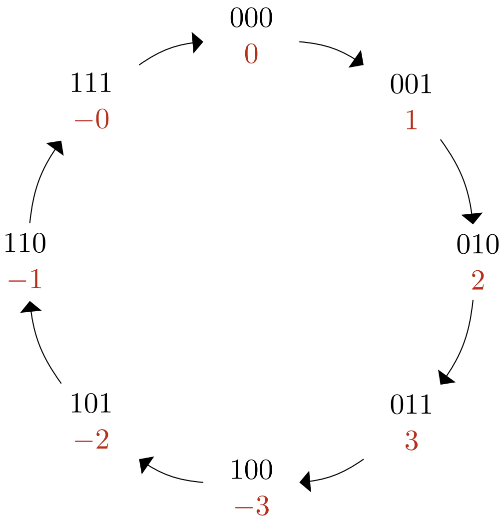
Wheel of 3-bit codes: signed magnitude integers

- leading bit: sign
- $+0$ and -0
- Symmetric range $[-3, +3]$



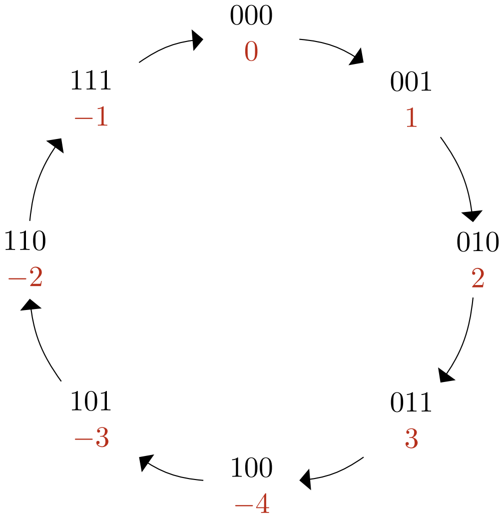
Wheel of 3-bit codes: one's complement integers

- $+0$ and -0
- Symmetric range $[-3, +3]$



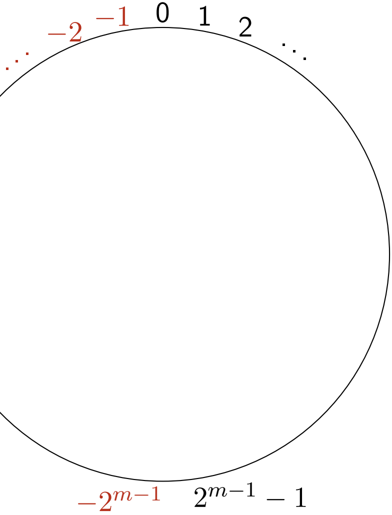
Wheel of 3-bit codes: two's complement, signed integers

- asymmetric range $[-4, +3]$
- economical: subtraction via addition
- explains counting sheep comic



Wheel of m-bit codes: two's complement, signed integers

0	000...000
1	000...001
2	000...010
...	...
$2^{m-1}-1$	011...111
-2^{m-1}	100...000
...	...
-2	111...110
-1	111...111



Lab 2 exercise: complement and add one

n	input	0 1 1 0 0 1 0 0 0
		$\Downarrow$
	complement	1 0 0 1 1 0 1 1 1
		$\Downarrow$
$TC(n)$	add one	1 0 0 1 1 1 0 0 0

Consider $n + TC(n) \ldots$

$$
\begin{array}{ll}
n & 0\ 1\ 1\ 0\ 0\ 1\ 0\ 0\ 0 \\
TC(n) & \underline{1\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0} \\
 & 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0
\end{array}
$$

$TC(n)$ is the additive inverse of n: i.e., $n + TC(n) = 0$

Two's complement addition

Let $m = b_{n-1}b_{n-2}\ldots b_2b_1b_0$ be an arbitrary n-bit pattern

Complement each bit: $C(m) = \overline{b}_{n-1}\overline{b}_{n-2}\ldots\overline{b}_2\overline{b}_1\overline{b}_0$

$$
\begin{aligned}
m + TC(m) &= m + (C(m) + 1) \\
&= (m + C(m)) + 1 \\
&= (b_{n-1}b_{n-2}\ldots b_2b_1b_0 + \overline{b}_{n-1}\overline{b}_{n-2}\ldots\overline{b}_2\overline{b}_1\overline{b}_0) + 1 \\
&= (11\ldots111) + 1 \\
&= 00\ldots000
\end{aligned}
$$

Conclusion

If m represents an integer k, then $TC(m)$ represents $-k$.

Two's complement: example

Using an 8-bit register, what is the two's complement representation of -20?

$$\begin{array}{rcl} 20 & = & 16 + 4 \\ & = & 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0 \\ \text{complement} & \rightarrow & 1\ 1\ 1\ 0\ 1\ 0\ 1\ 1 \\ \text{add one} & \rightarrow & 1\ 1\ 1\ 0\ 1\ 1\ 0\ 0 \end{array}$$

Verify. . .

$$\begin{array}{ll} n & 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0 \\ TC(n) & \underline{1\ 1\ 1\ 0\ 1\ 1\ 0\ 0} \end{array}$$

Two's complement: example

Using an 8-bit register, what is $TC(-20)$?

$$
\begin{array}{rcl}
-20 & \rightarrow & 1\ 1\ 1\ 0\ 1\ 1\ 0\ 0 \\
\text{complement} & \rightarrow & 0\ 0\ 0\ 1\ 0\ 0\ 1\ 1 \\
\text{add one} & \rightarrow & 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0 \\
 & = & 16 + 4 \\
TC(-20) & = & 20
\end{array}
$$

Two's complement: binary to decimal conversion

Given a bit string $n = b_{w-1}b_{w-2}\ldots b_2b_1b_0$, what value is represented?

MSB?	Conclusion	What to do
$b_{w-1} = 0$	value is non-negative	evaluate n as a binary value
$b_{w-1} = 1$	value is negative	find $TC(n)$, evaluate, affix sign

0 1 1 0 0 1 0 0 0 $=$?

1 0 0 1 1 1 0 0 0 $=$?

Two's complement: decimal to binary conversion

Give a decimal value n and a word length w, what bit string $b_{w-1}b_{w-2}\ldots b_2b_1b_0$ represents n?

Sign?	What to do
$n \geq 0$	convert(n)
$n < 0$	TC(convert($\lvert n \rvert$))

To convert a non-negative value. . .

Greedy "brute force" algorithm	identify highest powers of two
Successive division by two	identify bits from LSB to MSB

Examples: Using an 8-bit word. . .

$$57 = ?$$
$$-57 = ?$$

Decimal to binary conversion (non-negative value)

convert(n)

Successive division by two generates the bits in reverse order, from LSB to MSB. For example, take $n = 57$:

$$\begin{aligned} 57 \div 2 &= 28 \times 2 + 1 \\ 28 \div 2 &= 14 \times 2 + 0 \\ 14 \div 2 &= 7 \times 2 + 0 \\ 7 \div 2 &= 3 \times 2 + 1 \\ 3 \div 2 &= 1 \times 2 + 1 \\ 1 \div 2 &= 0 \times 2 + 1 \end{aligned}$$

Conclusion: $(57)_{10} = (111001)_2$. For an 8-bit register, we fill with leading zeros: $(57)_{10} = (00111001)_2$.

Decimal to binary conversion (negative value)

TC(convert($|n|$))

What is the 8-bit, two's complement representation of $n = -57$?

$$\begin{aligned} \text{convert}(|n|) &= \text{convert}(57) \\ &= (00111001)_2 \end{aligned}$$

Now, find the two's complement...

$$\begin{array}{rcl} 57 & \rightarrow & 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1 \\ \text{complement} & \rightarrow & 1\ 1\ 0\ 0\ 0\ 1\ 1\ 0 \\ \text{add one} & \rightarrow & 1\ 1\ 0\ 0\ 0\ 1\ 1\ 1 \end{array}$$

Addition on binary quantities

$$\begin{array}{cccc} a_3 & a_2 & a_1 & a_0 \\ b_3 & b_2 & b_1 & b_0 \\ \hline \end{array}$$

Addition on binary quantities

$$
\begin{array}{cccc}
 & & c_1 & \\
a_3 & a_2 & a_1 & a_0 \\
b_3 & b_2 & b_1 & b_0 \\
\hline
 & & & s_0
\end{array}
$$

Addition on binary quantities

$$\begin{array}{cccc} & c_2 & c_1 & \\ a_3 & a_2 & a_1 & a_0 \\ b_3 & b_2 & b_1 & b_0 \\ \hline & & s_1 & s_0 \end{array}$$

Addition on binary quantities

$$
\begin{array}{cccc}
{\color{red}c_3} & {\color{red}c_2} & {\color{red}c_1} & \\
a_3 & a_2 & a_1 & a_0 \\
b_3 & b_2 & b_1 & b_0 \\
\hline
 & {\color{blue}s_2} & {\color{blue}s_1} & {\color{blue}s_0}
\end{array}
$$

Addition on binary quantities

$$\begin{array}{ccccc} c_4 & c_3 & c_2 & c_1 & \\ & a_3 & a_2 & a_1 & a_0 \\ & b_3 & b_2 & b_1 & b_0 \\ \hline & s_3 & s_2 & s_1 & s_0 \end{array}$$

Addition on binary quantities

$$
\begin{array}{ccccc}
c_4 & c_3 & c_2 & c_1 & \\
 & a_3 & a_2 & a_1 & a_0 \\
 & b_3 & b_2 & b_1 & b_0 \\
\hline
 & s_3 & s_2 & s_1 & s_0
\end{array}
$$

We ignore the “carry out” c_4 generated in the leftmost column

Addition on binary quantities: example 1

$$\begin{array}{cccc} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ \hline \end{array}$$

Addition on binary quantities: example 1

$$\begin{array}{cccc} & & 0 & \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ \hline & & & 1 \end{array}$$

Addition on binary quantities: example 1

$$
\begin{array}{cccc}
 & 1 & 0 & \\
0 & 0 & 1 & 1 \\
0 & 0 & 1 & 0 \\
\hline
 & & 0 & 1
\end{array}
$$

Addition on binary quantities: example 1

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

Addition on binary quantities: example 1

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

Addition on binary quantities: example 1

$$\begin{array}{ccccc} 0 & 0 & 1 & 0 & \\ & 0 & 0 & 1 & 1 \\ & 0 & 0 & 1 & 0 \\ \hline & 0 & 1 & 0 & 1 \end{array}$$

This shows $3 + 2 = 5$ in a 4-bit system

Addition on binary quantities: example 2

$$\begin{array}{cccc} 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ \hline \end{array}$$

Addition on binary quantities: example 2

$$\begin{array}{cccc} & & 1 & \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ \hline & & & 0 \end{array}$$

Addition on binary quantities: example 2

$$
\begin{array}{cccc}
 & 1 & 1 & \\
0 & 0 & 1 & 1 \\
1 & 0 & 1 & 1 \\
\hline
 & & 1 & 0
\end{array}
$$

Addition on binary quantities: example 2

$$
\begin{array}{cccc}
\color{red}{0} & \color{red}{1} & \color{red}{1} & \\
0 & 0 & 1 & 1 \\
1 & 0 & 1 & 1 \\
\hline
 & \color{blue}{1} & \color{blue}{1} & \color{blue}{0}
\end{array}
$$

Addition on binary quantities: example 2

$$
\begin{array}{ccccc}
\color{red}{0} & \color{red}{0} & \color{red}{1} & \color{red}{1} & \\
 & 0 & 0 & 1 & 1 \\
 & 1 & 0 & 1 & 1 \\
\hline
 & \color{blue}{1} & \color{blue}{1} & \color{blue}{1} & \color{blue}{0}
\end{array}
$$

Addition on binary quantities: example 2

$$\begin{array}{ccccc} 0 & 0 & 1 & 1 & \\ & 0 & 0 & 1 & 1 \\ & 1 & 0 & 1 & 1 \\ \hline & 1 & 1 & 1 & 0 \end{array}$$

This shows $3 + (-5) = -2$ in a 4-bit system

Addition on binary quantities: example 3

$$\begin{array}{cccc} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ \hline \end{array}$$

Addition on binary quantities: example 3

$$
\begin{array}{cccc}
 & & 0 & \\
0 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 \\
\hline
 & & & 1
\end{array}
$$

Addition on binary quantities: example 3

$$\begin{array}{cccc} & 0 & 0 & \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ \hline & & 0 & 1 \end{array}$$

Addition on binary quantities: example 3

$$\begin{array}{cccc} \color{red}{1} & \color{red}{0} & \color{red}{0} & \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ \hline & \color{blue}{0} & \color{blue}{0} & \color{blue}{1} \end{array}$$

Addition on binary quantities: example 3

$$\begin{array}{ccccc} 0 & 1 & 0 & 0 & \\ & 0 & 1 & 0 & 1 \\ & 0 & 1 & 0 & 0 \\ \hline & 1 & 0 & 0 & 1 \end{array}$$

Addition on binary quantities: example 3

$$
\begin{array}{ccccc}
\color{red}{0} & \color{red}{1} & \color{red}{0} & \color{red}{0} & \\
 & 0 & 1 & 0 & 1 \\
 & 0 & 1 & 0 & 0 \\
\hline
 & \color{blue}{1} & \color{blue}{0} & \color{blue}{0} & \color{blue}{1}
\end{array}
$$

This shows $5 + 4 = -7$ in a 4-bit system

Oops: **arithmetic overflow**

Overflow summary for $A + B$

A	B	Outcome
positive	negative	correct result
negative	positive	correct result
negative	negative	possible overflow
positive	positive	possible overflow

Informal justification: two's complement wheel

Bit fiddling: arithmetic left shift

Various low-level operations on bit strings are often useful

Arithmetic left shift

$$b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0 \quad \text{becomes} \quad b_6 b_5 b_4 b_3 b_2 b_1 b_0 0$$

If there is no overflow...

- an arithmetic left shift operation computes $2k$, given k
- n successive arithmetic left shifts computes $2^n k$, given k

Bit fiddling: sign extension

- We use **sign extension** when we increase the number of bits
- For example, we may convert an 4-bit value to a 8-bit value
- Simply replicate the MSB
- $b_3b_2b_1b_0$ becomes $b_3b_3b_3b_3b_3b_2b_1b_0$

$$\begin{array}{ccc} 0101 & 00000101 & +5 \\ 1101 & 11111101 & -3 \end{array}$$

Why does it work?

Bit fiddling: bitwise AND

a	b	a AND b
0	0	0
0	1	0
1	0	0
1	1	1

Summary

- 1 AND $b = b$
- 0 AND $b = 0$

Bit fiddling: bitwise OR — inclusive or

a	b	a OR b
0	0	0
0	1	1
1	0	1
1	1	1

Summary

- 1 OR $b = 1$
- 0 OR $b = b$

Bit fiddling: bitwise XOR — exclusive or

a	b	a XOR b
0	0	0
0	1	1
1	0	1
1	1	0

Summary

- $a = b$ yields 0
- $a \neq b$ yields 1

Bit fiddling: masking operations

AND is useful for isolating specific bits

	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
AND	0	0	0	0	0	1	1	1
	0	0	0	0	0	b_2	b_1	b_0

OR is useful for inserting ones

	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
OR	1	1	1	1	1	0	0	0
	1	1	1	1	1	b_2	b_1	b_0

Bit fiddling: XOR application — testing for equality

$$
\begin{array}{rcccccccc}
 & a_7 & a_6 & a_5 & a_4 & a_3 & a_2 & a_1 & a_0 \\
\text{XOR} & b_7 & b_6 & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 \\
\hline
 & ? & ? & ? & ? & ? & ? & ? & ?
\end{array}
$$

Two bit patterns match if and only if all result bits are 0

Floating point representation

- Use a fixed number of bits, e.g., 32 bits
- Subdivide bits into fields: sign, exponent, fraction
- IEEE floating point standard (including Java's `float`):
  - 1 sign bit
  - 8 exponent bits, using "excess 127"
  - 23 fraction bits plus "hidden bit"
- Example 1: How can we represent $-6\frac{5}{8}$ as a 32-bit `float`?
- Example 2: What `float` value is represented by 3D800000 ?