CS2002 Logic Lecture 1

Data Representation in Binary

Ian Gent

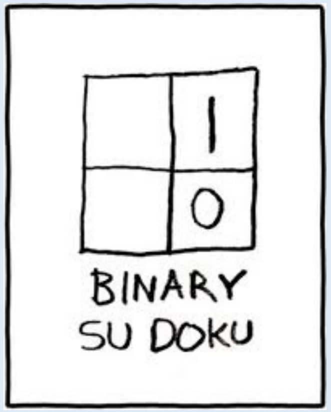
This Time …

- Representing numbers and text using binary
- Some of this is revision from CS1003
  - Will move quite fast to get through this material today

The Bit

1 or 0
true or false

- Binary digIT
- Two possibilities
  - high/low voltage
  - current on or off
  - magnetised up/down
  - hole present/absent
  - electrons there or not
  - light/dark in the optical fibre
- Basic unit of almost all modern information storage and processing.



Comic XKCD
https://xkcd.com/74/

Binary Numbers

- Positional system like decimal
- Reminder: right-most number is n×base^0 = n×1
- *k* bits can represent 2^k numbers, e.g., those from 0 to 2^k - 1

	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
3 = 2+1	0	0	1	1
6 = 4+2	0	1	1	0
10 = 8+2	1	0	1	0
15 = 8+4+2+1	1	1	1	1

Positive Numbers

- The atomic number of Tungsten : 74_{10}
  - $74_{10} = 2^6+2^3+2^1 = 1001010_2 = 112_8 = 4A_{16}$
- Population of UK in 2011: $63{,}181{,}775_{10}$
  - $=11110001000001001111001111_2$
  - $=361011717_8 = 3C413CF_{16}$
- Population of the world in 2010: $6{,}898{,}455{,}709_{10}$
  - $=110011011001011100001010010011101_2$
  - $=63313412235_8 = 19B2E149D_{16}$

Bytes and Bigger

- Group bits together to represent more than two possibilities
- n bits, 2^n possibilities
- A byte is traditionally the number of bits used to represent a text character, *usually* 8 bits.
- An octet is *always* 8 bits
- A word is the number of bits usually manipulated as a group on a particular computer
    - now almost always a multiple of 8 bits (16, 32, 64)
    - Usually 64 on modern computers
    - Hence "64 bit"
    - (x86 is confusing about what a word is)

Number of Bits	Number of Possibilities
5	$2^5 = 32$
6	$2^6 = 64$
8	$2^8 = 256$
16	$2^{16} = 65\ 536$
32	$2^{32} = 4\ 294\ 967\ 296$
64	$2^{64} = 18\ 446\ 744\ 073\ 709\ 551\ 616$

0	1	1	0	1	1	0	0	1	1	1	0	1	1	1	1
8 bit BYTE 01101100								8 bit BYTE 11101111							
16 bit WORD 0110110011101111															

Interpreting groups of bits

- Different ways of interpreting groups of bits:
  - as logical trues and falses
  - as integers, signed or unsigned
  - as characters and strings
  - as instructions telling the computer what to do
  - as “addresses” identifying a particular piece of memory in a store

Octal and Hexadecimal

<table>
<tr><th>Decimal</th><th colspan="16">61272</th></tr>
<tr><td>Binary</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td rowspan="2">Octal</td><td>1</td><td colspan="3">110</td><td colspan="3">111</td><td colspan="3">101</td><td colspan="3">011</td><td colspan="3">000</td></tr>
<tr><td>1</td><td colspan="3">6</td><td colspan="3">7</td><td colspan="3">5</td><td colspan="3">3</td><td colspan="3">0</td></tr>
<tr><td rowspan="2">Hex</td><td colspan="4">1110</td><td colspan="4">1111</td><td colspan="4">0101</td><td colspan="4">1000</td></tr>
<tr><td colspan="4">E</td><td colspan="4">F</td><td colspan="4">5</td><td colspan="4">8</td></tr>
</table>

Shorter notations for sequences of bits, easy to convert to/from binary

- Octal: group in 3s
- Hexadecimal: group in 4s
  - 10->A, 11->B ... 15->F
- Use prefixes or suffixes to disambiguate,
  - e.g., in C, 0x10 is in hexadecimal, = 16_{10}

Arithmetic and Overflow

- Adding two 8 bit numbers:
- Result is 9 bits
- Modern hardware discards the extra bit
  - Equivalent to reducing modulo 2^8 =256
  - Or modulo 2^{16} 2^{32} or 2^{64} for 16, 32, 64 bits
  - Note: in x86 extra bit used to set carry flag
- Similarly for subtraction or multiplication

```
  01100100
+11001000
----------
 100101100
```

Negative Numbers

- Idea: use the combinations of bits that start with 1 to represent negative values
- There are a whole range of possible ways of doing this:
  - Sign and magnitude
  - One’s complement
  - Two’s complement

Negative Numbers: Sign and Magnitude

- Leftmost bit is sign! 0 ⇒ +, 1 ⇒ -
- Rest are numerical value of number
- In 16 bits:
  - $+1_{10} = 0000\ 0000\ 0000\ 0001_2$
  - $-1_{10} = 1000\ 0000\ 0000\ 0001_2$
  - $? \quad = 1000\ 0000\ 0000\ 0000_2$

Negative Numbers: One's Complement

- Also suffers from the -0 problem
- Does not make best use of the binary patterns
- Difficult to implement subtraction in hardware

Binary	Decimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-7
1001	-6
1010	-5
1011	-4
1100	-3
1101	-2
1110	-1
1111	-0

Negative Numbers: Two’s Complement

- Subtract a bigger number from a smaller one
- E.g. in 8 bit how do we do 50-100?
- Well let’s add $2^8 = 256$
- We can do 306 – 100 easily
- gives ‘206’
- Idea: use 206 to represent -50
- n bits: represent -x by 2^n-x
- Two’s complement solves the problems of -0 and hardware implementation of subtraction.

```
(1)00110010

  -01100100
 ___________
   11011110
```

Negative Numbers: Two’s Complement (II)

- Avoids -0 problem
- Extends range of negative numbers: n bits: $-2^{n-1} \ldots 2^{n-1}-1$
  - E.g. -8 to 7 for 4 bits.
- Subtraction is easy to implement
  - For y – x just add y and (-x)
- Given x get –x by:
  - Flipping bits
  - Adding one

Binary	One’s Complement	Two’s Complement
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	-7	-8
1001	-6	-7
1010	-5	-6
1011	-4	-5
1100	-3	-4
1101	-2	-3
1110	-1	-2
1111	-0	-1

Two's Complement Arithmetic

```
 5 = 0101
-5 = 1010
(1s complement)
-5 = 1011
(2s complement)

 7 = 0111
-7 = 1000
(1s complement)
-7 = 1001
(2s complement)
```

```
-5 + 7 = 2
      1011
   +  0111
   = (1)0010

4 - 7 = -3
      0100
   +  1001
   =  1101
```

Binary	**Two's Complement**
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

Floating Point

- Bigger and smaller numbers, fractions
- Scientific notation:
  - 2.99792458×10^{8}
  - 6.626068×10^{-34}
  - 6.0221415×10^{23}
- Do the same in binary:
  - $1.0011 \times 2^{11} = ?$
  - $-1.0011 \times 2^{1101} = ?$

Floating Point Ctd

- $1.0011 \times 2^{11} = ?$
  - $1.0011 = 1+2^{-3}+2^{-4} = 1+1/8+1/16 = 1.1875_{10}$
  - $2^{11} = 2^{3} = 8_{10}$
  - $1.0011 \times 2^{11} = 1.1875 \times 8 = \mathbf{9.5}_{10}$
- $-1.0011 \times 2^{1101} = ?$
  - $2^{1101} = 2^{13} = 8192_{10}$
  - $-1.0011 \times 2^{1101} = -1.1875 \times 8192 = \mathbf{-9728_{10}}$

Sign, Mantissa, Exponent

$$-1.0011 \times 2^{1101}$$

- Sign is what you expect
  - negative
- Exponent is power of 2
  - 1101
- Mantissa is binary fraction
  - 1.0011
- Divide word into sign (1bit), mantissa and exponent.
- No need to record the .
- Can skip first bit of mantissa - why?

Example Floating Point Formats (IEEE 754 names)			
Single 32 bits	Sign 1 bit	Exponent 8 bits	Mantissa 23 bits
Double 64 bits	Sign 1 bit	Exponent 11 bits	Mantissa 52 bits
Quadruple 128 bits	Sign 1 bit	Exponent 15 bits	Mantissa 112 bits

It gets complicated

- IEEE 754 is floating point standard
  - 2008 version is 70 page pdf
- Many many issues to cope with, e.g....
    - +0 and -0, NaN, +Infinity and –Infinity
    - Five kinds of rounding
    - Underflow and subnormal numbers
- Software Carpentry Video about FP
  - https://www.youtube.com/watch?v=Qoam964M_6Y

Representing Text 1

- ASCII -- 7 bits per character
  - 33 control, 94 visible, plus space
- 8 bit hardware -- high bit 0, or for parity
- extended ASCII -- multiple versions
- Strings:
  - In C: one octet per character, terminated by 0

Representing Text 2: Unicode

- 109,000 20 bit numbers for characters
- First 256 agree with most common extended ASCII
- Two common representations:
  - UTF-16, all common characters in one 16 bit value, rest in 2.
  - UTF-8: ASCII characters as octets 0-127
    - other characters 2-4 octets in range 128-255

UTF-8

Bits	Max code	Octet 1	Octet 2	Octet 3	Octet 4
7	7F	0xxxxxxx			
11	7FF	110xxxxx	10xxxxxx		
16	FFFF	1110xxxx	10xxxxxx	10xxxxxx	
21	1FFFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

Next Time …

- Why cover logic in a course called “Computer Systems”?
- Some key logical connectives
- Modus Ponens
- “A puff of logic”