COSC2473

Number Systems

<u>Data representation & Binary Numbers</u>

Other Number Systems and Characters

Units of Measure

- We use all sorts of units to measure things, and in many cases, deal with scale by converting between units.
- So instead of counting the day in seconds (=86400s), we invent minutes and hours as units and define them.

- To convert, we divide by the largest unit that leaves an non-zero integer and a remainder. For example:
  - 25s → 25s (no division needed)
  - 250s → 250/60 minutes = 4 min + 10s rem = 4m19s or 0:4:10
  - 2500s → 2500/(60x60) hours < 1, so 2500/60 min = 41m40s
  - 25000s → 25000/3600 hours = 6h56m40s = 6:56:40

Repeat this using 24, 240, 2400 seconds instead

Units of Measure 2

- How to calculate:
  What is total length of 27 boxes 11’7” (11 feet, 7 inches) wide?
  - Convert size to inches
  - Do the multiplication
  - Convert back to yards, feet, inches.
  - ***Here the factors are different for each scale.***
    - ***3 feet = 1 yard,   12 inches = 1 foot***

- How to calculate
  When is April 22nd + 200 days?
  - One way is to convert April 22nd to a 'day number'
    - 22 + 31 (mar) + 28 (feb) + 31 (jan) = 112
    - target date is 112+200 = 312.
    - Have fun converting this to a date.
  - Another way is to count forward. Same trouble…

  - ***Here the factors don't even add up uniformly <u>within</u> a scale!!***

Units of Measure 3

- All units of measure have two components:
  - a scale factor,
  - a dimension,
  - For now, let's not worry about the dimension part.

- The key to the confusion of the previous slide is that the *<u>factors between all the scales are different</u>* (12 inches → 1 foot, 3 feet → 1 yard, …)
- The French solves this by inventing the metric system, where the scaling factors are factors of 10 and ***<u>are all the same</u>***, which makes them *powers of 10*.
  - 1 km was defined as 1/10000th the circumference of the Earth going through Paris and the North and South poles. (longitude)
  - he French did this in reaction to the English “inventing“ GMT (Greenwich Mean Time)

Roman Number Systems

- The Romans used 7 symbols'
  - I=1, V=5, X=10, L=50, C=100,D=500, M=1000
  - Included scale, but not in a consistent way

- To calculate
  - Eg suppose we add 12+18 in roman numerals
  - Addition: 12=XII, 18=XVIII.
    - We add by appending the symbols: XIIXVIII = XXVIIIII
    - And then simplifying IIIII=V, so XXVV, but VV=X, so XXX
  - Multiplication is just repeated addition: 12*3=XIIXIIXII = XXXIIIIII
    - And again we simplify to XXXVI

- So we do the same thing as with units of measure
  - Collect the symbols' as per the operation (eg add/multiple)
  - Simplify the result

Hindu-Arabic Number System

- We can optimise the simplification process by using a small set of symbols, but then their meanings can differ
- The Hindu Arabic system combines conversion and simplification with the operation
  - Basic operation: 2+3=5
  - Conversion and simplification
    - Use of the carry' for the power conversion from units to tens
    - Eg 3*5 = 5 and a conversion of 1 to 10

- Compare this with units of time (from Phoenicians and Mayans)
  - Consider 5 processes in sequence each taking the time, 00:18:22
    - **<u>Calculation</u>**: 5*0 : 5*18 : 5*41= 00:90:205
    - **<u>Conversion: process</u>** 0:90 :0 = 1:30:0, 0:0:205 = 0:3:25
    - **<u>Simplification</u>** using 'carry' 0+1=0, 30+3=333, so result is 1:33:25
- Can you imagine doing ALL your arithmetic like this?

Number System for Binary Data

- The basic units of data are 1's and 0's which make up binary numbers

- Binary can be:
  - Unipolar: 0 and 1 used for number system
  - Bipolar: -1 and +1 more often used electrically (eg Volts)

  - We will return to this in Data Comms lectures, where the "bit" is the basic unit for the Physical Layer.

- Because these are hard for humans to work with, we often convert them to other number bases such as **octal, decimal** or **hexadecimal** for ease of use

Bits (Binary Digits)

- Computers exist to process information. How can we represent information inside a machine?

- One of the simplest machines is the **binary** (i.e. two-state) switch.
  - such a two-state switch can represent True/False, Off/On etc.
  - by convention, the two states are represented as the binary digits (**bits**) 0 (off) and 1 (on)

- Size in bits: accepted but unofficial usage
  - 8 bits = 1 byte (4 bits = 1 nybble)
  - 1024 bytes = 1 Kilobyte = 1 KiB
  - 1024 KiB = 1 Megabyte = 1MiB
  - 2^{10} MiB = 1 Gigabyte = 1 GiB
  - 2^{10} GiB = 1 Terabyte = 1 TiB = 2^{40} B
  - 2^{10} TiB = 1 Petabyte = 1 PiB
  - 2^{10} PiB = 1 Exabyte = 1 EiB = 2^{60} B

- NOT the same as the standard metric
  - where Kilo = 1000 , Mega=10^6, … Exa = 10^{18}

The Importance of Zero

- What comes after 9?

- 10 of course, we don't need to think about it do we?
- Zero allows digits to represent different things depending on <u>place value</u>

- 50 is different from 500 and 5
- If we didn't have zero, we would run out of numbers after 9, just like Roman numerals…
  - VIII, IX, XL, C, D, M largest "digit" is M
  - Consider multiplication: Method of Replication + Substitution
    - 2 x VIII = VV IIIIII (replication) = XVI (Substitution)

Data representation

- A digital computer represents all information as strings of bits
  - numbers: integer, floating point (i.e. 'real') numbers etc
  - characters
  - program instructions

- How can we represent integer (whole) numbers using bits?

- Consider decimal 105

1 0 5

$1 * 10^2 + 0 * 10^1 + 5 * 10^0$

  - In decimal each place indicates a power of 10
  - It is a **base-10** number system
  - Notice how the zero has a place value

Number Systems

- If
  - scale factors between different sized units are the same, and
  - these scale factors are integer powers of a base
- then we have a power-law based ***<u>number system</u>***
  - and we can assume the base, and simply append the digits.
- This notation is our familiar number system

- We can have number systems where each place indicates a power of some base number
- Consider base 5 number: 410

$$4 * 5^2 \quad + \quad 1 * 5^1 \quad + \quad 0 * 5^0$$

- In base 5, each place indicates a power of 5
- 410_5 in base 5 = 105_{10} in base 10

Binary numbers

- Decimal is a **base-10** number system
- Binary is a **base-2** number system
  - each position represents a power of 2

- Consider binary 1101001:

1 1 0 1 0 0 1

$1 * 2^6 + 1 * 2^5 + 0 * 2^4 + 1 * 2^3 + 0 * 2^2 + 0 * 2^1 + 1 * 2^0$

64 + 32 + 0 + 8 + 0 + 0 + 1

which is 105 (decimal)

- To avoid confusion, we can indicate a number's base as a subscript or some other mechanism
- e.g. 105 dec = 105_{10}

Binary numbers

- Each bit position (from right to left) has a weight that is $2^{\text{bit number}}$
  - e.g. position 0 (right-most) has weight 2^0, position 1 has weight 2^1 etc
- With n bits, we can represent values 0 to $2^n - 1$
  - e.g. with 3 bits we can have 0 to $2^3 - 1 = 7$
- The left-most bit is the most-significant bit (MSB)
- The right-most bit is the least-significant bit (LSB)

Binary numbers

- With 4 bits we can represent:

- $0000_2 = 0_{10}$
- $0001_2 = 1_{10}$
- $0010_2 = 2_{10}$
- $0011_2 = 3_{10}$
- $0100_2 = 4_{10}$
- $0101_2 = 5_{10}$
- $0110_2 = 6_{10}$
- $0111_2 = 7_{10}$

- $1000_2 = 8_{10}$
- $1001_2 = 9_{10}$
- $1010_2 = 10_{10}$
- $1011_2 = 11_{10}$
- $1100_2 = 12_{10}$
- $1101_2 = 13_{10}$
- $1110_2 = 14_{10}$
- $1111_2 = 15_{10}$

Conversion of Decimal to Binary (#1)

- Successive halving can be used to convert base-10 numbers to base-2

	remainder	
178/2	0 (LSB i.e. rightmost bit)	
89/2	1	
44/2	0	
22/2	0	Result
11/2	1	Reading up from bottom
5/2	1	$178_{10} = 10110010_2$
2/2	0	
1/2	1 (MSB)	

Conversion of Binary to Decimal (#1)

- **<u>Successive doubling</u>** can be used to convert base-2 to base-10 numbers

1 0 1 1 0 0 1 0

0 + 2 * 89 = 178

1 + 2 * 44 = 89

0 + 2 * 22 = 44

0 + 2 * 11 = 22

1 + 2 * 5 = 11

1 + 2 * 2 = 5

0 + 2 * 1 = 2

1 + 2 * 0 = 1

Conversion of Binary to Decimal (#2)

- Alternatively, just add the relevant powers of 2

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
1	0	1	1	0	0	1	0

128 +32+16 +2 = 178

MSB=1…………………LSB=0

Binary Addition

$0 + 0 = 0$ $\quad$ $0 + 1 = 1$

$1 + 0 = 1$ $\quad$ $1 + 1 = 10$ (?)

Addition Rules for 1 bit numbers

- Binary addition:

01100101 +	101_{10}
00001010	10_{10}

01101111	111_{10}

. . . . . 1 . . . . Carry

01100101 +

00010110

01101011

Carry condition

- 0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1, 1 + 1 = 10
- Notice how in the last case, we needed an extra bit to represent the sum?
  - This is because the addition produced a ‘carry’ and we needed a place to put it.
- More complex example:

```
.11..1..        .1.        Carry
10110010 +      178₁₀
00010010         18₁₀
--------
11000100        196₁₀
```

this bit is ‘lost’, this error is known as a ‘carry condition’ or ‘Overflow’

Carry condition

- Computers can only allocate a finite number of bits to represent a number
- Assume only **8** bits are used for each integer

	11111111 +	255_{10}
	00000001	1_{10}

1	00000000	256_{10}

– this bit is usually 'lost', this error is known as a 'carry condition' or 'Overflow'

– The carry can be temporarily stored in the CPU for further use. It is often called the "carry bit" and can be tested for in programs

Negative Numbers (1's complement)

- A simple way to represent negative numbers is to take the **complement** of the positive number
- To get the complement each 0 becomes 1 and each 1 becomes 0

- Example +5 in binary is 0101 and +2 is 0010

0101 (+5)	0010 (+2)
1010	1101
So -5 is 1010 ,	and -2 is 1101

Negative Numbers (1’s complement)

- Why does this work?
- Consider the number line

-3	-2	-1	0	+1	+2	+3	Decimal
00	01	10	11				1’s Compl
			00	01	10	11	Normal
			|	Position of mirror			

- Notice:
  - Numbers have shift symmetry, so addition moves to the right
  - The pattern of 00 01 10 11 is the same on the negative side as the positive side, so binary addition works the same way
- But also notice:
  - -0 = +0, so binary 11 is the same as 00
  - On top of that, we don’t know how to represent -3 without confusion

Problem with 1's complement

- Consider 0

  0000
- We can complement this code and get

  1111,
- which in ones complement is also a valid representation of 0
- Having two different ways to represent the same number creates problems:

  If (x == 0)…..

  While (n != 0), n = n-1
- We always need to test for both 0000 and 1111, which is inefficient

Negative Numbers (2's complement)

- The solution is to use two's complement
- To get the two's complement negative representation of a number we complement it and add 1:
- Examples

  0101 (5)

  1010 ones complement

  1 add 1

  1011 is the twos complement representation

- We can check this by taking the complement again

Negative Numbers (2's complement)

- Example 2

  1011 (-5)

  0100 ones complement

  +1 add 1

  0101 We get back the original number

- Now consider 0

  0000

  1111 complement

  +1 add 1

  0000

- So there is only 1 representation of 0

- How has the number line changed for 2's complement?

Sign Condition

- Using 2’s complement has another advantage
- Negative numbers all have a ‘1’ in the MSB
- The sign condition and carry condition behave identically
- CPU’s can use this fact to simplify their design
  - Typically the MSB is copied into a ‘sign bit’. In the CPU, just as a carry bit is.

Overflow

- The sign / carry bit can also be used to signify overflow.
- Suppose we enter the number $173_{10} = 10101101_2$ into an 8 bit byte without a sign bit. This is an ***unsigned byte***
  - It fits.

Unsigned Byte = 8 bits

1	0	1	0	1	1	0	1

- Now use this number in a 7 bit byte with sign bit
  - Number: 0 0101101 = 45
  - Complement + 1: 1 1010010 + 1 = 82+1 = 83

Signed Byte = 7 bits + sign bit

1	0	1	0	1	1	0	1

Quiz

- What is the largest unsigned base-10 integer that can be stored in 6 bits?

- Assuming an 8 bit integer representation, convert 107_{10} to binary. Then convert it to Octal, and Hexadecimal

- Convert 11111011110_2 to Octal and decimal

- Convert $9A9_{16}$ and then $DEAD_{16}$ to decimal