COS10004 Computer Systems

Lecture 1.4 Bits and Number Systems

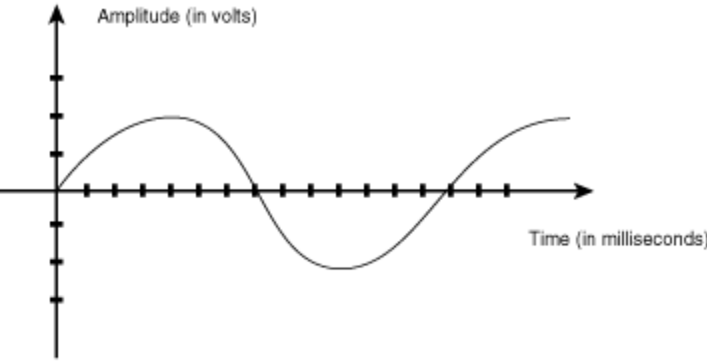
CRICOS provider 00111D

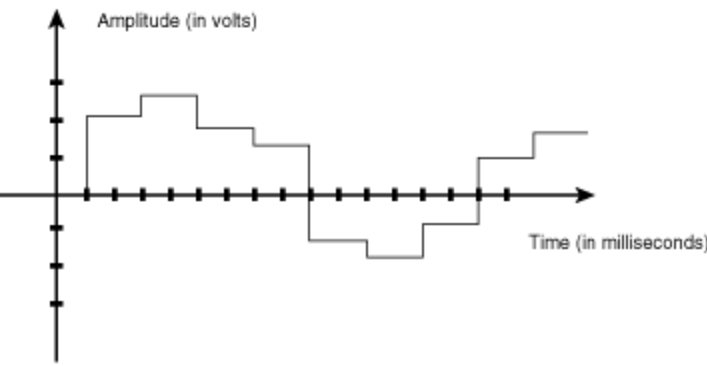
Dr Chris McCarthy

Information and Computers

- Multiple bit representation of information:
  - numbers (32 bits) -> double/float; int
  - Characters (8 bits) -> ASCII chars
- Numerical equivalence of multiple bits:
  - The computer doesn't know/care what the bits are supposed to be used for, it just sees bits
  - numbers/chars can be manipulated by same instructions.

ANALOGUE AND DIGITAL





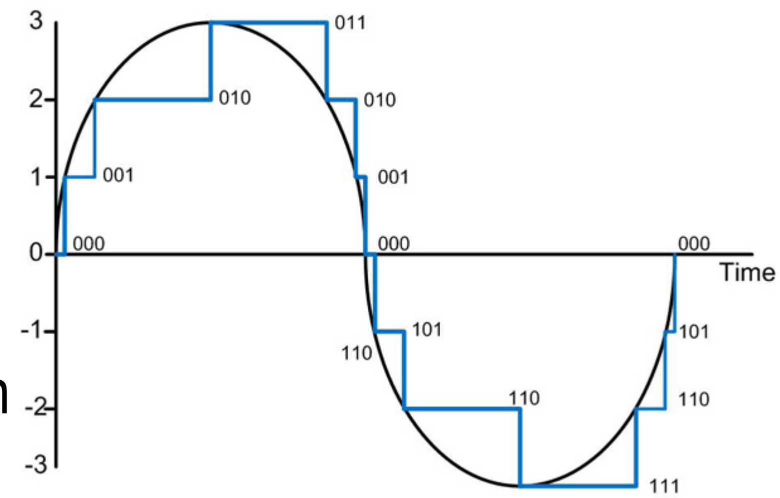
Analogue - Data that represented on a continuous scale (real numbers)

- e.g. voltage, height, distance, amount of magnetisation on audio tapes.
- Good for real world measurements, transients (spikes), transistors.

Digital - Data that is represented as whole numbers (count)

- e.g. class size, coins in your pocket, sound samples on CD recordings,
- Good for noise rejection, computers.

DIGITAL ADVANTAGES



- Digital systems are easier to:
  - design and modify
  - store data
  - maintain accuracy and precision
  - program operations
  - to protect from noise
  - to create on an IC chip

- However the real world is mostly analogue
- **Translation is inaccurate.**
- **The more bits the more precise.**

DIGITAL CODE - 1

- Data in computer is stored digitally as 0s and 1s in a binary code.
- The 1s and 0s are represented in many ways:
  - by a voltage e.g. 0 or 5 volts (actually we don't use exact voltages) eg < 0.8 volts == 0 > 2.4 volts == 1 (TTL)
  - by the absence or presence of a pit (CD)
  - by the absence or presence of a magnetic field (disks)
- Why binary?
  - Simple, robust, universal laws (Shannon-Hartley), flexible
  - c.f. sending decimal using voltages…. long cables… interference.)

Signal-to-noise ratio. http://www.linfo.org/shannon-hartley_theorem.html

DIGITAL CODE - 2

> groups of bits can represent numbers, characters, computer instructions.

> How we interpret a 32-bit word depends on what we expect! (Exercise)

What data can you represent in 4 bytes?

- Hardware has to be able to manipulate groups of bits differently depending on what they represent.
- Amazingly, we can use combinations of Gates to do this.

8-bit example

128	64	32	16	8	4	2	1
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1	0	1	0	1	0	1	1

- Possible interpretations:
  - Character '1/2' assuming extended ASCII/ISO
  - The number 171
  - The number -85

```c
#include <stdio.h>
int main()
{
    unsigned char dec = 171;
    printf("integer:%d  char %c   signed %d\n", (int)dec, (char)dec, (signed char)dec);
    return 0;
}
```

E:\Apps\quincy\bin\quincy.exe

```
integer:171   char ½   signed -85

Press Enter to return to Quincy...
```

NUMBER SYSTEMS

- We use a “positional number system” #big-endian
  - in decimal numbers, the further left a digit is, the greater the power of 10 it is multiplied by.
  - e,g, $348 = 3 \times 10^2 + 4 \times 10^1 + 8 \times 10^0$
- This system applied to other Radix (or Bases)

– eg.	Decimal	radix = 10 digits 0..9
–	Binary	radix = 2 digits 0..1
–	Octal	radix = 8 digits 0..7
–	Hexadecimal	radix = 16 digits 0..F

BINARY NUMBERS

> Computers work in binary numbers (2 values)

e,g, binary number 01001110_2 is

$=0 * 2^7 + 1 * 2^6 + 0 * 2^5 + 0 * 2^4 + 1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 0 * 2^0$

$=0 * 128 + 1 * 64 + 0 * 32 + 0 * 16 + 1 * 8 + 1 * 4 + 1 * 2 + 0 * 1$

$= 1 * 64 + 1 * 8 + 1 * 4 + 1 * 2$

$= 78$

> Exercise :- convert to decimal

- $01110_2 =$
- $10011_2 =$
- notation: in maths use 01110_2 to indicate a binary number
- some systems use other ways such as %01110 or **b01110**

Binary Counting

You can fill in the rest.

Decimal	Binary	Decimal	Binary
0	0000	11	1011
1	0001	12	1100
2	0010	13	1101
3	0011	14	1110
4	0100	15	1111
5	0101	16	10000
6	0110	17	
7	0111	...	
8		31	
9		32	
10		65	

HEXADECIMAL NUMBERS

- Hex is another abbreviation for binary numbers.
  - Each byte represented by 2 hex digits; each hex digit represents 4 bits in an 8-bit byte.

  - Hex to binary can be done one digit at a time:

e.g. what is $4E_{16}$ in binary?

4	E
0100	1110
4*16	14*1

$=4 \times 16^1 + 14 \times 16^0 = 64+14 = 78_{10}$

Hexadecimal Numbers

- 0=0, 1=1, 2=2, 3=3, 4=4, 5=5, 6=6, 7=7, 8=8, 9=9
- Memorise: A=10, B=11, C=12, D=13, E=14, F=15
  - mathematically use 1230_{16} or 1230_{H} to indicate a hexadecimal number
  - some systems use other ways such as 0x1110 or 01110h
    - Or sometimes hex is default (not decimal!)

leading 0x means hex

HEX – BINARY (4 BITS TO A HEX DIGIT)

Hex	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Hex	Binary
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

BIT-WISE OPERATIONS

- Can base decisions on a single bit #button state
  - e.g. if (bit represents True) then …
  - Usually 1 means True, 0 False.
- Unary operation: NOT
  - Complement ("invert", "flip") the value
- Binary* operations: AND, NAND, OR, NOR, XOR
- N-bit operations: e.g. 4-input AND, OR
  - All can be implemented using binary building blocks ("2-input gates")

SUMMARY

- Modern computers operate with digital representations of information:
  - Easier to work with but has Implications
- Number systems and conversions to know:
  - Binary ← → decimal
  - Hex ← → Binary
  - Hex ← → Decimal
  - And the formula in general!
- Next Lecture: Gates