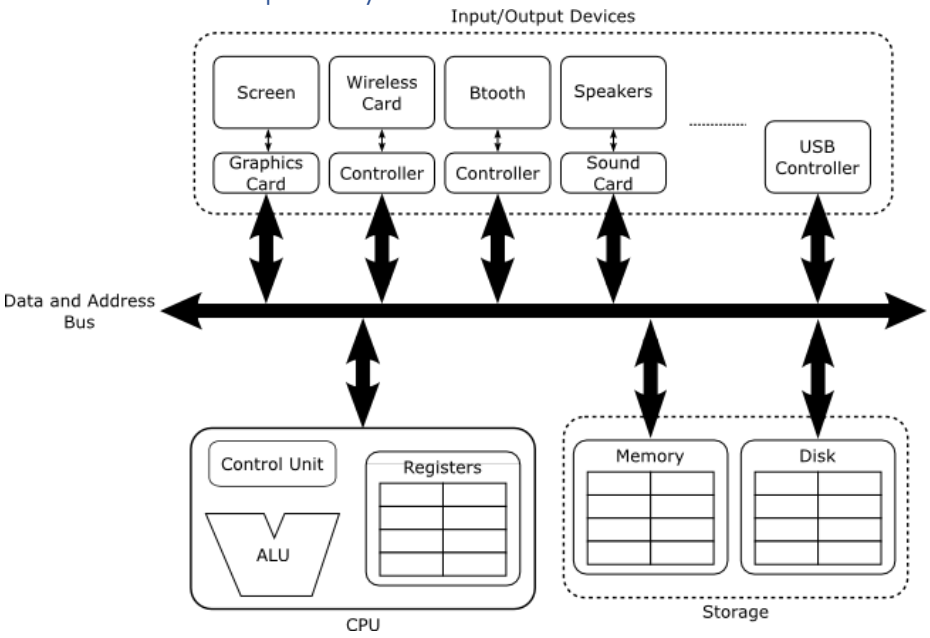
ELEC1601 Notes

Week 1

Structure of a computer system



- Every computer system has at least one **processor or CPU** (central processing unit)
    - The CPU contains elements such as registers (which store some temporary data being processed) and arithmetic-logic unit (ALU) and a control unit in charge of giving the orders to the rest of the digital circuits to execute the sequence of machine instructions.
    - The processor communicates with the rest of the elements in the architecture through the data and address bus
- **Storage devices** are used to store data to be used by the processor.
    - Options to store data: static RAM, dynamic RAM, flash memory, hard drive etc.
    - Some loose their information when powered down (volatile memory) and others still preserve their content without power (non-volatile memory)
- **Input/output devices** are in charge of sending and receiving data for the system.
    - Usually there is a specialised sub-system in charge of one type of operation e.g. screen is handled by a specific sub-system called the graphics card

The assembly language

- Every processor can execute a set of pre-defined instructions called its **machine language**
    - These instructions are encoded in binary digits
    - Processors provide a human readable format to represent the machine language called assembly
- Example of program written in the assembly language of the AVR architecture:

```
        .data                   ; Start the data section
msg:    .asciz "Hello world\n"  ; Message to print through serial port

        .text                   ; Start the code section
        .global main            ; Declare main as global symbol

main:   ldi r26, lo8(msg)       ; Load low 8 bits of address of msg
        ldi r27, hi8(msg)       ; Load high 8 bits of address of msg
        push r26                ; push the 16 bits to the stack
        push r27
        call printf             ; call function to print msg
        pop r27                 ; Remove data from the stack
        pop r26
        ret                     ; Finish the program
```

Week 2

Encoding in base 2, 8 and 16

- **Octal** → base 8
  - **Binary to base 8** → keep removing the last 3 digits of the binary number (last significant digits), then add the powers
  - E.g. 101000110
    = 101000 [110] = $101000\ [1^4 1^2 0^1]$
    = 101 [000] = $101\ [0^4 0^2 0^1]$
    = [101] = $[1^4 0^2 1^1]$
    = $[1^4 + 0^2 + 1^1][0^4 + 0^2 + 0^1][1^4 + 1^2 + 0^1]$
    = [5][0][6] → 506 is base 8 equivalent BUT add 0 to the front so 0506
  - Convert back to base 8 in same way you go binary to base 10, but do it in groups of 3
- **Base 16**
  - Convert to hexadecimal and then add 0x in front of it to distinguish it
- Binary numbers are always written with the prefix '0b'

Numbers in bases 2,8,10 and 16

- Given a number written in base 10, the least significant digit which is written in the right most position corresponds with the units.
- Any number represented in a base *b* satisfies the following conditions:
  - *b* digits are used to represent numbers. The digits go from 0 to *b*-1
  - The number represented by *b* – 1 is followed by the number 10
  - The maximum number with *d* digits is followed by one with *d* + 1 in which the most significant is 1 and the rest are all zeros
- The value of a sequence of digits is obtained by multiplying each of them by the base raised to the exponent denoting its position starting by the least significant one being zero as shown in the following example
  - $3214 = 3 * 1000 + 2 * 100 + 1 * 10 + 4$
  - Can be written as: $3214 = 3 * 10^3 + 2 * 10^2 + 1 * 10^1 + 4 * 10^0$
  - The general formula to obtain the value of any number on any base can be re-written as:
    - $3214 = \sum_{i=0}^{3}(d_i * base^i) = (d_0 * 10^0) + (d_1 * 10^1) + (d_2 * 10^2) + (d_3 * 10^3)$
  - Thus, the value in base 10 of any n-digit number in base b is obtained by the following equation:
    - $\sum_{i=0}^{n-1}(d_i * b^i)$

$$
\begin{aligned}
6342_7 &= \sum_{i=0}^{3}(d_i * 7^i) \\
&= (2 * 7^0) + (4 * 7^1) + (3 * 7^2) + (6 * 7^3) \\
&= 2 + 28 + 147 + 2058 \\
&= 2235_{10}.
\end{aligned}
$$

- The opposite process, that is, given a number in base 10 obtain its equivalent in a different base requires repeated divisions by the base to obtain the digits of the new number starting by the lease significant.
  - E.g. 8675_{10} to base 7

$= \frac{8675}{7} = 1239\ r\ 2$
= 1239/7 = 177 r 0
= 177/7 = 25 r 2
= 25/7 = 3 r 4
= 3/7 = 0 r 3
SO answer is 34202_7

Integers encoded as sign and magnitude

- Any integer can be considered as a pair formed by a sign and a natural number (representing its absolute value). E.g. -345 can be represented by the pair (-,345)
    - SO the natural number can be encoded normally, and we use one bit to represent the sign. 0 for the positive sign and 1 for the negative sign.
- Integers encoded with sign and magnitude using a 10 bit representation:

Number in decimal	Sign	Absolute Value	Sign and magnitude
-342	- = 1	342 = 101010110	0b1101010110
342	+ = 0	342 = 101010110	0b0101010110
-23	- = 1	23 = 10111	0b1000010111

Integers encoded in 2s complement and addition

- A sign and magnitude representation of size n bits allows to encode the integers in the interval $[-(2^{n-1}-1), 2^{n-1}-1]$ → highest number starts with a zero (for positive), followed by all ones. Lowest value has a 1 in its left-most bit (negative) followed by all ones.
    - The issue with this is zero has two representations. All zeros and a 1 as the most significant bit and the rest zeros → means an opportunity is wasted to encode on more integer in that interval.
- The **2s complement encoding scheme** allows 2^n consecutive integers around the value zero to be encoded with n bits. The range represented by n bits is: $[-(2^{n-1}), 2^{n-1}-1]$
- The translation of an integer encoded in base 10 to a 2s complement with n bits is done with the following steps:
    1. If the number is larger or equal to zero, simply translate it to base 2
    2. If the number is negative, apply these 3 steps:
        a. Obtain the base 2 encoding of the absolute value of the number
        b. In that representation, replace every 0 by a 1 and every 1 by a 0
        c. Add 1 to the resulting number
- The translation of an integer in 2s complement with n bits to its representation in base 10 is done with the following steps:
    1. If the number is positive (most significant bit is zero), the number in base 10 is obtained by usual process of binary to decimal
    2. If the number is negative (most significant bit is one), the base 10 value can be obtained by either of the following 2 methods:
        a. Apply the equation: $ABS(N) - 2^n$ where ABS(N) is the value obtained when translating the n bits directly into a base 10 number
        b. Apply the following 3 steps:

i. Replace every 0 by a 1 and every 1 by a 0 in the n bits
ii. Add 1 to the resulting number
iii. Translate the resulting number to base 10 and take its value as a negative number

The floating point format

- Real numbers are encoded in binary logic with the technique known as **floating point**. Every real number is represented as a set of digits to the left of the point and another set of digits to the right of the point (the part to the right of the point is called mantissa or significand).
    - Multiplying & dividing by the base allows the point to be moved to the left & right respectively. E.g.

$$\begin{aligned} 14345.342 &= 1434.5342 * 10 \\ &= 143.45342 * 10^2 \\ &= 14.345342 * 10^3 \\ &= 1.4345342 * 10^4 \\ &= 0.14345342 * 10^5 \end{aligned}$$

    - Base 2 example:

$$\begin{aligned} 101.111 &= 2^2 + 2^0 + 2^{-1} + 2^{-2} + 2^{-3} \\ &= 4 + 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} \\ &= 5.875 \end{aligned}$$

- An n-bit 2's complement signed integer can represent integers from -2^(n-1) to +2^(n-1)-1
- The structure of the floating point representation of a real number:

Sign	Mantissa	Exponent