What is Software?

Recall the first Computers...

• Wish to automate this process
• Software is the controller for the hardware (switches)
• At the heart of software is algorithms...

Algorithms

• Wikipedia defines an algorithm as:
  “A formula or set of steps for solving a particular problem. To be an algorithm, a set of rules must be unambiguous and have a clear stopping point. Algorithms can be expressed in any language from natural languages like English or French, to programming languages like FORTAN.”

• Named after the ninth century scholar Abu Ja’far Muhammad Ibn Musu Al-Khowarizmi (780-840 AD)

• In fact, algorithms go back further...

The First Integer Relation Algorithm

• Euclid’s Algorithm
  – Calculating the greatest common divisor of two numbers.
    \[
    \begin{align*}
    \frac{a}{b} & \text{ gives a remainder of } r \\
    \frac{b}{r} & \text{ gives a remainder of } s \\
    \frac{r}{s} & \text{ gives a remainder of } t \\
    & \text{ ...} \\
    \frac{w}{x} & \text{ gives a remainder of } y \\
    \frac{x}{y} & \text{ gives no remainder}
    \end{align*}
    \]

    \[\text{GCD of } a \text{ and } b \text{ is } y\]
The First Integer Relation Algorithm

- Euclid did his proof of the algorithm geometrically (!!!) as algebra had not been invented yet
  – appeared in Euclid’s *Elements* (300BC)
- One of the best examples of an efficient algorithm.

Algorithms

- Challenge in programming – developing *elegant* algorithms that are simple and require the fewest steps possible.

Searching for Efficient Algorithms

- Efficiency of algorithms is crucial
- For example, searching for all occurrences of a letter in a list, assuming there are N letters randomly stored in a long array.

  ```
  A L O N G L I S T O F N A M E S
  ```

  - You will have to test every entry to ensure you find all occurrences of a letter.
  - You need to perform N tests.

Searching for Efficient Algorithms

- If names are stored alphabetically, a sequential search for a letter can be stopped as soon as you find a letter that comes after the one you want.
  - e.g. Look for ‘I’
    ```
    A A E F G J I L L M N N O O S S T
    ```
    - On average, you will need N/2 tests – twice as fast
Searching for Efficient Algorithms

- A better solution – binary search to look for 'I'
  - Check the list ¼ way down the list
  - If it comes after the one you want, ignore the right half of the list
  - Repeat the process with the left half
  - Repeat until you find your desired letter
- Each step, we divide the list in half – in the worst case, you need $\log_2(N)$ tests to find the letter.

• Time proportional to $\log_2(N)$ vs $N$

• Speed gain for using an efficient algorithm is enormous especially for large $N$.
  - e.g. population of Australia is about 20,000,000 – you will need no more than 25 tests to find a name!
  - Sorting is essential but not necessarily easy.

• Let’s see a comparison of sorting algorithms:

Implementing Algorithms

- Restricted by available hardware
- Need to
  - supply data (the algorithm “inputs”)
  - control operations
  - read resulting data (the algorithm “outputs”)

Step 1: The Data

- If I could talk to the computers...
Integer Arithmetic

- We (humans) usually represent integers using a base-10 notation.
  The number 99120 means
  \[9 \times 10^4 + 9 \times 10^3 + 1 \times 10^2 + 2 \times 10^1 + 0 \times 10^0\]
- Computers represent integers using a base-2 notation – binary numbers.
  The number 10001 means
  \[1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0\]
  \[= 16 + 0 + 0 + 0 + 1 = 17\] in base-10

Binary Numbers

Question: Why do computers use binary?
One answer: Only need to know the one times table!
- We normally represent numbers in base 10 – need to know times tables up to 10.
- If the human race only had one hand with five fingers, we might count in base 5 like:
  1, 2, 3, 4, 10, 11, 12, 13, 14, 20, 21, 22…
  - only need to learn up to the 4x table but the table will be different (e.g. 4x4=31 in base 5)

Binary Numbers

- Working in binary, one will count like this:
  1, 10, 11, 100, 101, 110, 111, 1000, 1001, …
- To do arithmetic we only need to be able to add one and one and know the one times table 
  - eg. 6x3
- This is readily implemented in hardware using logic gates
  - and, or, nand, nor, etc.

Hexadecimal Numbers

- Easier for humans to group blocks of 4 bits 
  - base 16 → Hexadecimal numbers

<table>
<thead>
<tr>
<th>decimal</th>
<th>binary</th>
<th>hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>A</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>B</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td>C</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td>D</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>E</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td>F</td>
</tr>
<tr>
<td>16</td>
<td>10000</td>
<td>10</td>
</tr>
</tbody>
</table>
Hexadecimal Numbers

- Why work in hexadecimal?
  It can be convenient when specifying binary numbers
  e.g.
  \[1001101 = 2^7 + 2^4 + 2^3 + 2^2 + 2^0 = 128 + 16 + 8 + 4 + 1 = 157\]
  Working in hexadecimal, we simply group the bits in fours and look up the appropriate digit:
  \[1001 \ 1101\]
  \[9 \ D\]

Signed Integers

- How can we express negative numbers?
- Three options...
  - **Sign Magnitude**
    - The most significant bit is 0 for positive numbers and 1 for negative numbers.
    - Advantage: easy for humans
    - Disadvantages
      - Contains zero twice
      - Arithmetic logic hard for computers
    - e.g adding two positive numbers different from +ve and -ve

One's complement

- The negative of a number is represented by flipping the number's bits one at a time
  e.g.  \[00011101 \text{ (29)}\]
  \[11100010 \text{ (-29)}\]
  - Leftmost bit still gives sign
  - Useful properties:
    - negating a -ve number gives the original number
    - addition two numbers works whether they are positive or negative!
      - subtraction becomes addition!
  - Disadvantages
    - still two 0s (+0 and -0)
    - still difficult for multiplication (different for +ve and -ve)
Two’s complement

• In two’s complement representation, negative of a number is represented by:
  1. Invert all the bits in the number
  2. Add 1.
  e.g. \(3_{10} = 0011_{2c}\)
      \(-3_{10} = 1100 + 1 = 1101_{2c}\)
      \((-3)_{10} = 0010 + 1 = 0011_{2c}\)

• For N-bit numbers, this system represents the values in the range of
  \([-2^{N-1}, 2^{N-1}-1]\)
  e.g. for 8-bit numbers, \([-128, 127]\)
      for 4-bit numbers, \([-8, 7]\)

• Adding two’s complement numbers
  e.g. \(-75 + 13 = -01001011 + 00001101\)

Two’s complement

• Advantages
  - Negation of -ve number is original number
  - Adding two’s complement numbers works for both +ve and -ve numbers
    e.g. \(-75 + 13 = -01001011 + 00001101\)
  - Even works for multiplying!
    \(-12: 11110100\)
    \(8: 00001000\)
    \(--\)
    Result: 11110100000010100000
    \(-96!\)

More Data

• We will see representations later for
  - character and string data
  - floating point (decimal) numbers
  - structured data (eg matrices)

Step 2: Controlling the Operations

Hard Disc  ← Central Processing Unit
Memory management unit
Arithmetic logic unit
Floating point unit
Memory
(RAM)

Input & Output Devices
Display monitor
Keyboard
Mouse
Sound card etc.
Central Processing Unit (CPU)

- The chip that runs the computer
- Consists of a number of components, some of which include:
  - Memory management unit
  - Arithmetic logic unit – operations on numbers represented by integers (…-1, 0, 1, 2, …)
  - Floating point unit – allows efficient operations on floating point numbers (e.g. 1.5695)

Random Access Memory (RAM)

- A set of locations where data may be stored and read. (Why is it called random access?)
- The smallest unit of storage is the bit, a single binary digit (0 or 1).
- A byte is the smallest individually accessible unit of memory – consists of 8 bits. Operations may use 8, 16, 32 or 64 bit words.
- Memory can be accessed very quickly, but when the machine is switched off, loses its value (“volatile”).
- Secondary memory – eg. hard disc can be used for long term (“persistent”) storage.
- Accessing data on a hard disc is much slower than accessing RAM (Why can adding more RAM make your computer faster?)

What Happens when you Run a Program?

- When you start a program (e.g. a word processor), the program is read from the hard disc (if not already resident) and loaded into memory
- The program consists of machine instructions expressed as binary patterns of bits
- Instructions include
  - control signals (eg tell ALU to add)
  - addresses (eg where to get or put the data)
What Happens when you Run a Program?

- The CPU operates in a cycle
  - read (fetch) next program instruction from memory
  - decode it
    - eg separate control signals from addresses
  - and execute it (usually moving or operating on data)
- *Fetch-execute cycle*

What Happens when you Run a Program?

- When you load (or open) a document (or file) for editing, this file is read from the disc and loaded into memory too.
- As you edit the file, it is the copy sitting in memory that is being changed.
  (What if the file is too large for the memory?)
- When you save the document, the edited version sitting in memory is written back to disc.
  - The old version on disc is lost.
  - The copy in memory remains for further editing.
- When you exit the program, the copy sitting in memory, and any loaded documents, are removed – the space in memory is marked as being free for overwriting.

Controlling the Operations

- Computer thus requires sets of wires for
  - sending control signals (eg telling ALU to add)
  - transferring data
  - passing addresses (eg telling CPU where to find or put the data)
- Called *busses*
Computer Architecture

Operating Systems...