2. Hierarchical Abstraction

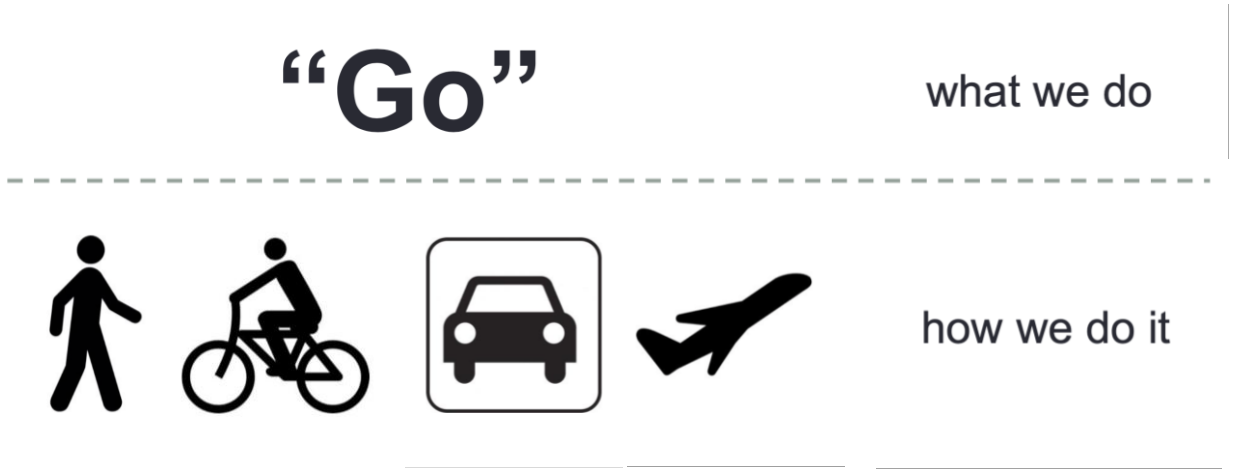
What is abstraction?

- By dictionary

  “a general term or idea”

- In computer science

  “separating meaning from implementation”



Main Characteristics

- Applying abstraction level by level
- Abstraction at some level will hide all implementation details at a lower level
- Enable us to consider a task at different levels of detail
- Adding Example:

  Consider the task of adding together a group of N numbers, e.g., 1, 2, 3, ... , 98, 99, 100.

Adding Example

- At its simplest / highest level:

  An exercise that generates the sum of adding 1 – 100

  - Input: 1, 2, 3, ... , 98, 99, 100
  - Output: the sum of adding 1 – 100

- At this level of abstraction, we consider only the task or **application** being performed.

- At the application level we only consider **what** we want to do

Adding Example

- A more detailed view (**how** to get the sum):

  The procedure or **program** to produce the sum

  - One alternative is:

    1 + 2 + 3 + 4 + … + 98 + 99 + 100 = 5050

  - Another alternative is:

    1 + 100 = 101

    2 + 99 = 101

    …

    49 + 52 = 101

    50 + 51 = 101

    so, 101 × 50 = 5050

- Different ways (procedures) of accomplishing the same task

Adding Example

- In CS, program: express the procedure formally, precisely, in some language, known as programming language (C/C++, Python, Java, etc.)

- "Adding the number in order" in C:

```
int sum(int a[5])
{
  int i,s ;
  s = 0;
  for(i=0;i<5;i++) s = s + a[i];
  return s;
}
```

Adding Example

- Programs in programming languages are not directly understandable for a computer
- Assembly language: **instructions** understandable by the computer.
- The above C program translated into instructions for an x86 computer:

Assembly Language Version:

```
        MOV AX,0
        MOV CX,05H
LOOP:   ADD AX,[BX]
        INC BX
        INC BX
        DEC CX
        JNZ LOOP
```

Machine Code Version:

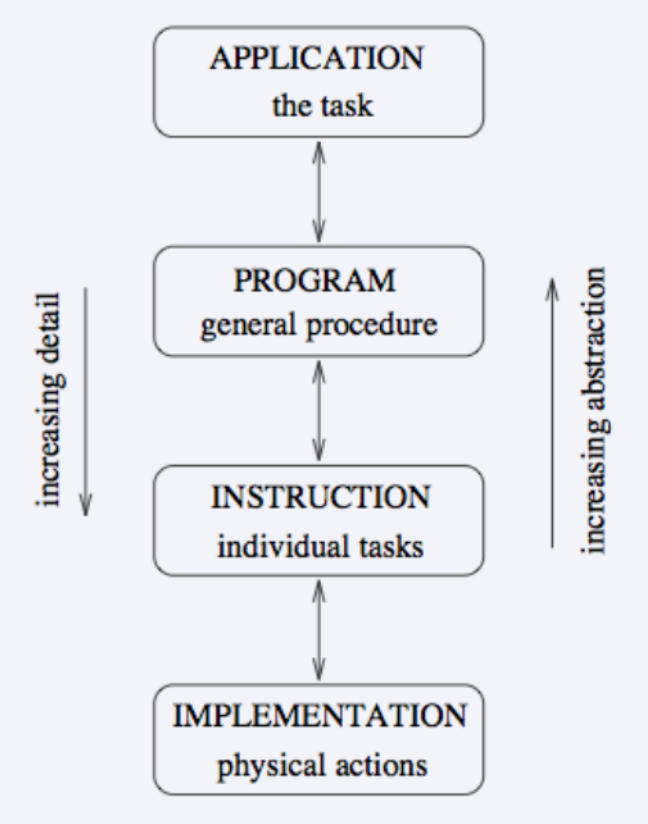
```
10111000000000000000000
10111001000000000000101
0000000000000111
01000011
01000011
01001001
0111010100000111
```

- The same program may be translated to different lists of instructions, depends on the computer's instruction sets.

Adding Example

- Finally, the physical **implementation** (0s and 1s, voltage levels)

- Hierarchical abstraction for the adding example



The Importance of Hierarchical Abstraction

- It enables us to consider any task or machine at a level of detail which is appropriate for our needs, whilst hiding any irrelevant detail or complexity

- We can gain a better understanding of the purpose and operation of something at an appropriate level without being impeded by unnecessary detail

- A given abstraction level is essentially independent of those below it, i.e., it can remain unchanged even though the levels below may be altered (e.g., a C program can be used in different machines)

Application Level

- We are concerned with the task that the computer can perform (e.g., word processor, spreadsheet package, a library database, or some other information system)
- There will be a procedure for using the application and a means of interaction via the screen, keyboard, mouse or other input device
- These applications are analogous to our summation task, albeit sometimes rather more complicated
- The basic features are the same: input, output, a means of communication, and a place to record the data

Program Level

- Deals with the procedure used to carry out the task.
- This is known to the computer as the program and the language that it is specified in is a programming language (e.g., C/C++, Java, Python, Pascal, Fortran)
- Programming languages look nothing like a spoken language, they are based on words and numbers which, once learnt, makes them readable and in some sense intelligible
- They are termed high level languages, which reflect their position in the abstraction hierarchy
- A computer can thus be “told” what procedure it should use to complete a task by giving it a program written in one of these languages

Instruction Level

- The computer generates a set of machine instructions using a compiler
- The compiler translates the program into a series of instructions, which the electronic circuits of the computer can recognise and directly execute
- These basic instructions are rarely more complicated than: add two numbers, check if a number is equal to zero, or move a piece of data from one part of the computer's memory to another

Instruction Level I & II

- There are two instruction levels – the difference being the language in which the instructions are specified
- **Level II** consists of assembly language instructions, which form a very primitive language
- Although it still consists of words and numbers, it is much less readable than C or Java etc. at the program level
- **Level I** consists of machine code instructions
- These form a completely unreadable language consisting entirely of 0's and 1's. This is the native language of the computer that we will return to later in the module

Instruction Level I & II

- From the initial program, the computer generates a set of detailed instructions, first in assembly language and then in machine code (which is generated from the assembly language instructions using an assembler)

- The reason for the two levels is in fact very simple: when designing computers, humans need to work at the instruction level and find it difficult, if not impossible, to work with 0's and 1's

- Note, however, that the instructions at the two levels are usually equivalent – each assembly language instruction being assembled into an equivalent machine code instruction.

C Program example

- Program to add 5 numbers together:

```
int sum(int a[5])
{
  int i,s ;
  s = 0;
  for(i=0;i<5;i++) s = s + a[i];
  return s;
}
```

Assembly Language vs Machine Code

- Assembly Language Version:

```
        MOV AX,0
        MOV CX,05H
LOOP:   ADD AX,[BX]
        INC BX
        INC BX
        DEC CX
        JNZ LOOP
```

- Machine Code Version:

```
10111000000000000000000
10111001000000000000101
0000000000000111
01000011
01000011
01001001
0111010100000111
```

Implementation Level

- At the bottom of the hierarchy, we are concerned with how the computer actually carries out the set of machine instructions
- The interest is in the hardware – the physical computer itself, from the electronic components inside, such as integrated circuits, to the screen and keyboard
- Moreover, we are interested in how these work together, how information is communicated between them, and how the machine communicates with the user, e.g., the generation of characters on the screen or storing data on some form of storage media

Implementation Level

- Note that the form or structure of the hardware is not determined from the higher levels of the hierarchy – it only has to be able to carry out the operations that are used at the instruction level
- This is an example of the independence of the abstraction levels and is why there are so many different types of computer, all doing the same thing right down to the instruction level
- The difference is that they are implementing the tasks differently; either because of efficiency, performance, or cost, or in some cases all three
- In addition, there are computers that perform a different set of instructions, in which case their instruction levels also look slightly different
- However, in all cases, the program level and those above it should remain the same

Implementation Level

- Hardware and software and the interface between them
- By software we mean everything above the implementation level
- A conceptual difficulty arises when we consider the question: when does software become hardware and vice versa?
- In other words, what is the difference between a program and the integrated circuitry that carry out the program?

Program vs Integrated Circuitry

- In one sense, there is no difference – hardware and software are equivalent, they are just different forms of the same thing
- The best way to view it is again via abstraction: software represents an abstract representation of the tasks carried out by the hardware in a form which is in some sense understandable by humans
- This makes it easy to work with and to alter what the computer should be doing
- However, it is important to bear in mind that everything specified in software could also be directly implemented in hardware, it is just too difficult, cumbersome and expensive to do so.

Abstraction Hierarchy for a Computer