CM1101 COMPUTATIONAL THINKING

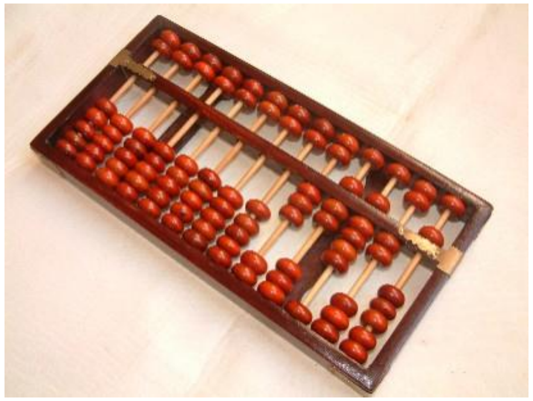
COMPUTER ARCHITECTURE 1

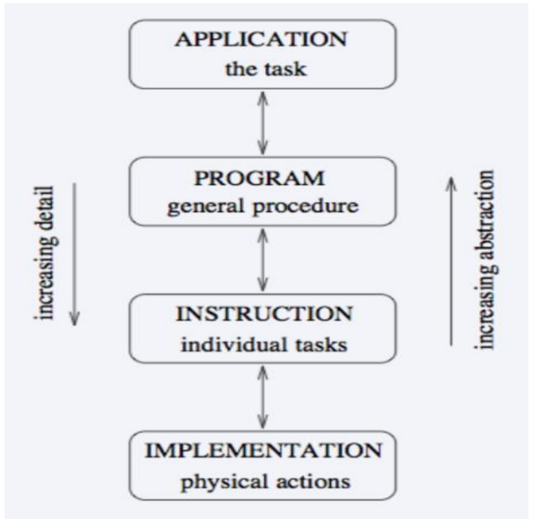
Dr Jing Wu

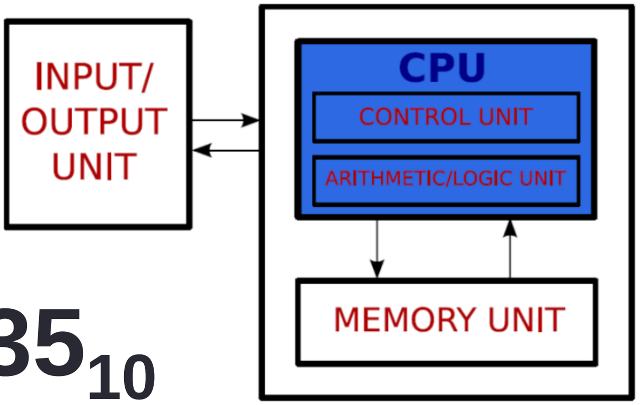
Room WX/2.06B
wuj11@cardiff.ac.uk

Overview

- The history of computers
- Hierarchical abstraction
- The von Neumann architecture
- Data representation







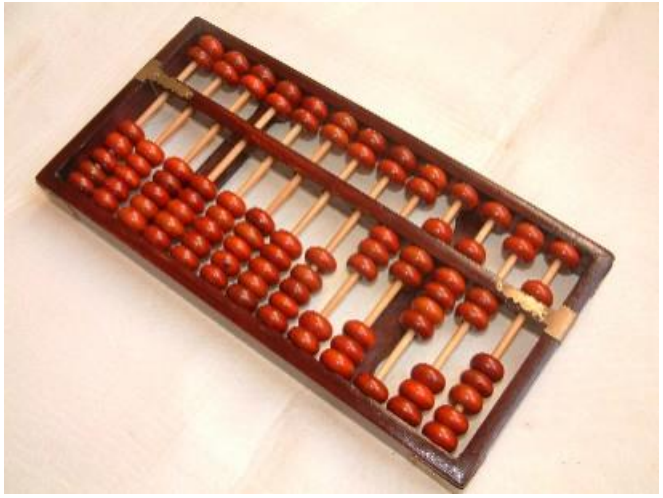
$$00100011_2 = 35_{10}$$

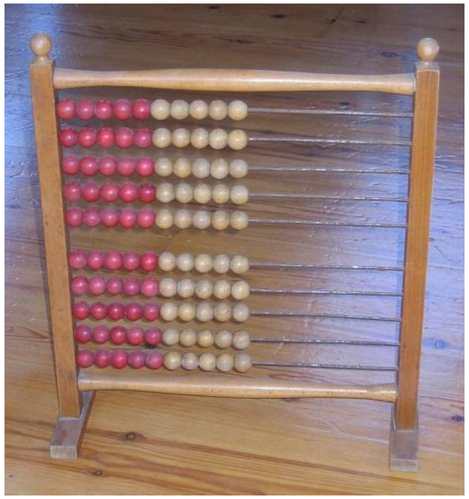
The history of computers

Abacus

- 2700 – 2300 BC
- rods → unit

  beads → digit
- Addition, subtraction, multiplication, division

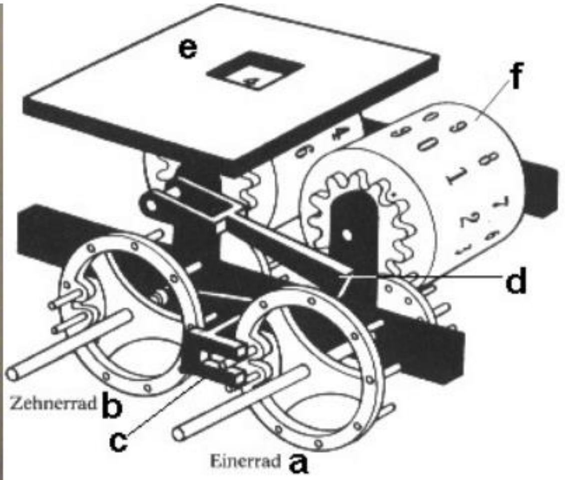




The "Pascaline"

Mechanical Era

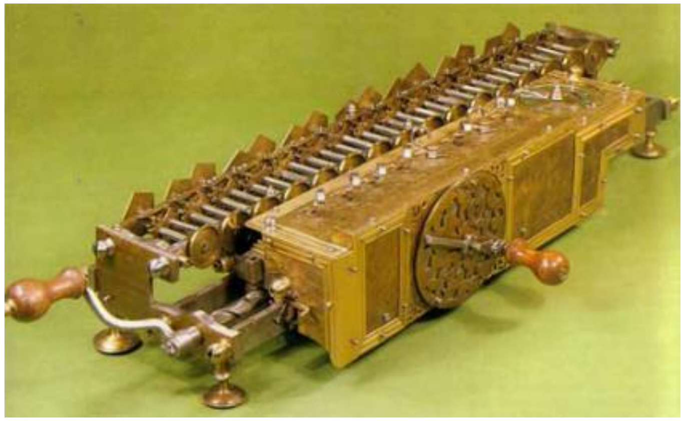
- 1642: Blaise Pascal

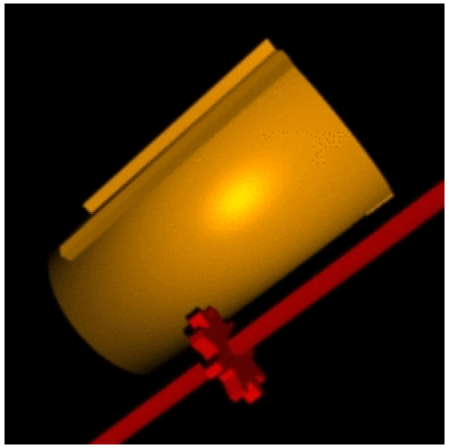


- First mechanical calculator.
- Higher wheel rotates by 1 when a lower dial produces a carry.
  - Can do addition directly on machine.
  - Can do subtraction by method of complements.
  - Short multiplication/division by repeated addition/subtraction

The “Stepped Reckoner”

- 1671: Gottfried Wilhelm von Leibniz



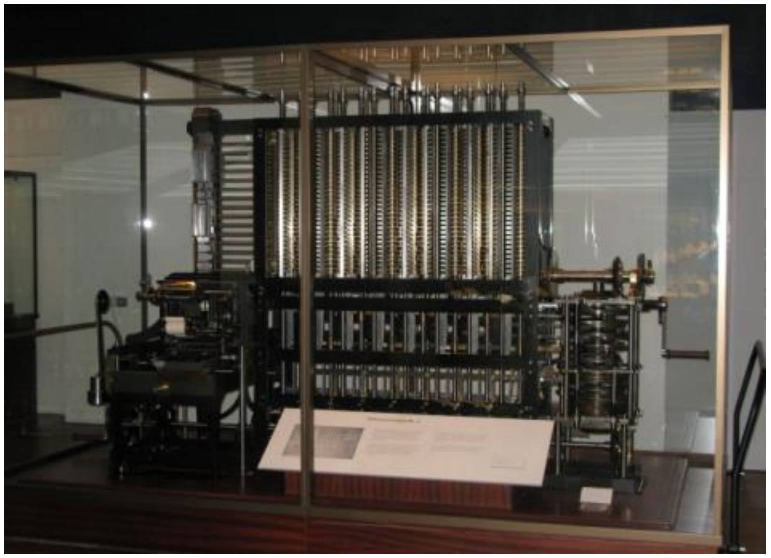


…it is beneath the dignity of excellent men to waste their time in calculation when any peasant could do the work just as accurately with the aid of a machine.-- Gottfried Leibniz

- Can do addition and subtraction as on the Pascaline.
- “Leibniz wheel” allows for long multiplication and division.

The Difference Engine

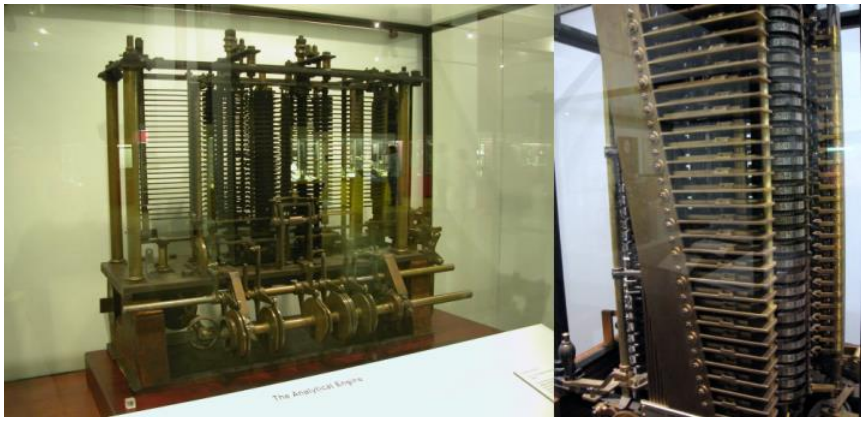
- 1822: Charles Babbage



- Evaluates polynomials. Therefore can approximate trigonometric functions, logarithms, etc. in addition to adding, subtraction, multiplication and division.

The Analytical Engine

- 1837: Charles Babbage

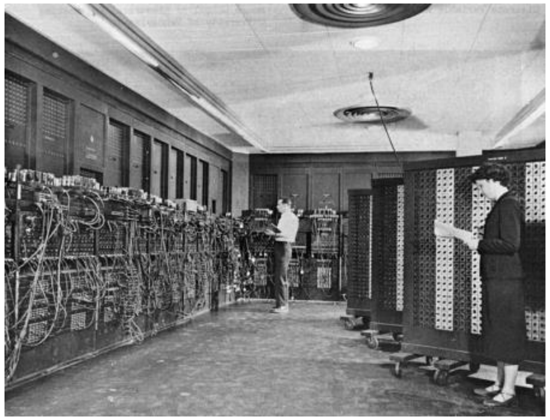


- Programmable
- Basic components found in all modern computers
  - Mill performs arithmetic operations.
  - Operator processes operations specified by punch cards.
  - Store holds data on punch cards.
  - Output prints results.

ENIAC

Electronic Era

- 1943-1946, Electronic Numerical Integrator And Calculator



- Arguably the first all electronic computer.
- Breath-taking speed: 300 multiplications or 5000 additions per second.
- Enormous size: 17,500 vacuum tubes, 500,000 soldered connections, filled a 50-foot long basement room, weighed 30 tons.

Generations of Electronic Computer

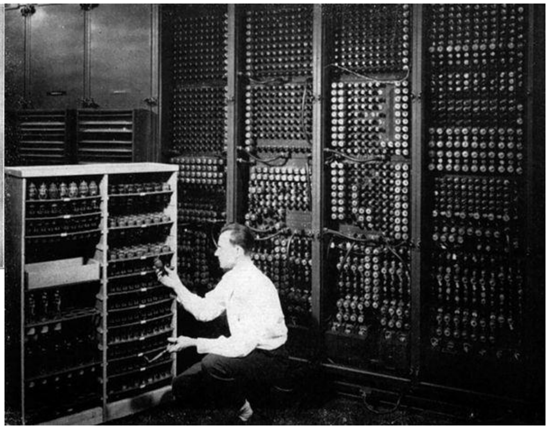
- 1st Generation (1940-1956) -- Vacuum Tubes

  Very large

  Expensive

  Huge amount of electricity

  E.g., ENIAC

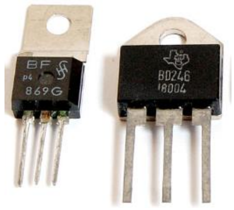


- 2nd Generation (1956-1963) – Transistors

  Smaller, faster, cheaper

- Punched cards for input,

  Print-outs for output.



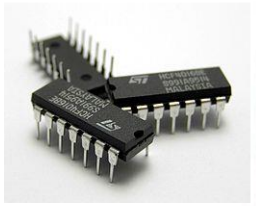
Generations of Electronic Computer

- 3rd Generation (1964-1971) – Integrated Circuits (IC)

  Faster, cheaper, smaller;

  Less electricity, fewer mistakes;

  Keyboards, monitors.



- 4th Generation (1971-present) – Microprocessor

  Small, portable, cheaper;

  Less electricity, less heat;

  More ways to interact.

Generation of Electronic Computers

- 5th Generation (the future) – Artificial Intelligence

  Bridge the gap between computing and thinking

The history and future of computers

- https://www.youtube.com/watch?v=xrUvFJWlYCY

Hierarchical abstraction

What is abstraction?

- By dictionary

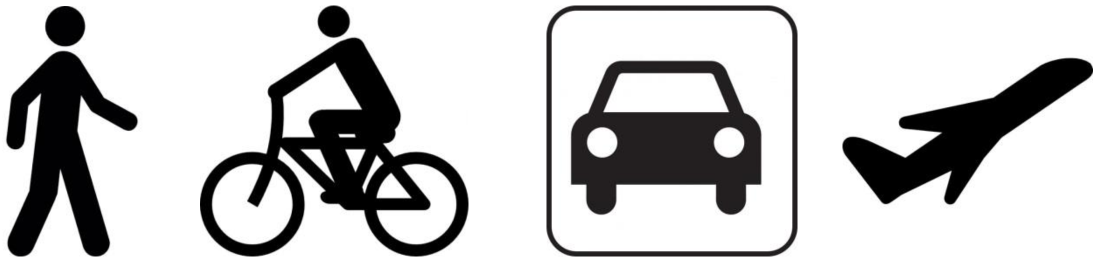
    “a general term or idea”

- In computer science

    “separating meaning from implementation”

“Go”

what we do



how we do it

Hierarchical Abstraction

- 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 yields 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.

Adding Example

- A more detailed view:

  The procedure or program to produce the sum.

  - $1 + 2 + 3 + 4 + \ldots + 98 + 99 + 100 = 5050$
  - Alternatively, $1 + 100 = 101$ $\quad 101 \times 50 = 5050$

    $2 + 99 = 101$

    $\ldots$

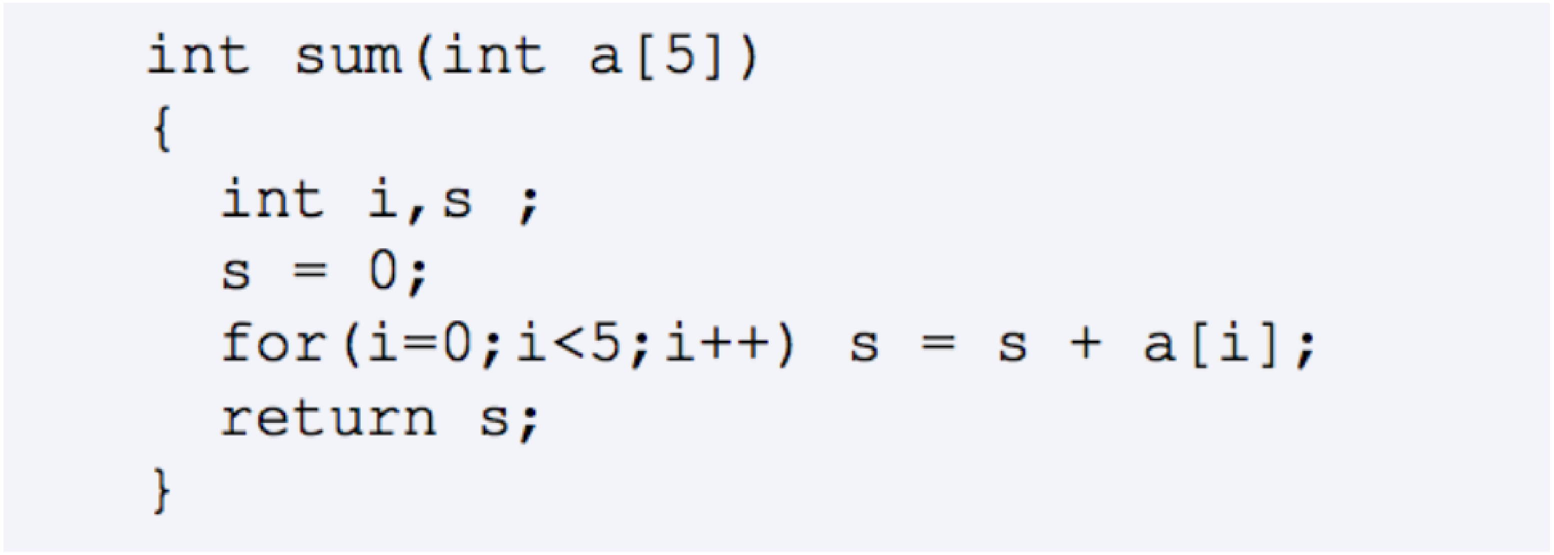
    $49 + 52 = 101$

    $50 + 51 = 101$

- Different ways of accomplishing the same task.

Adding Example

- Program: express the procedure formally, precisely, in some language, known as programming language.
- “Adding the number in order” in C:

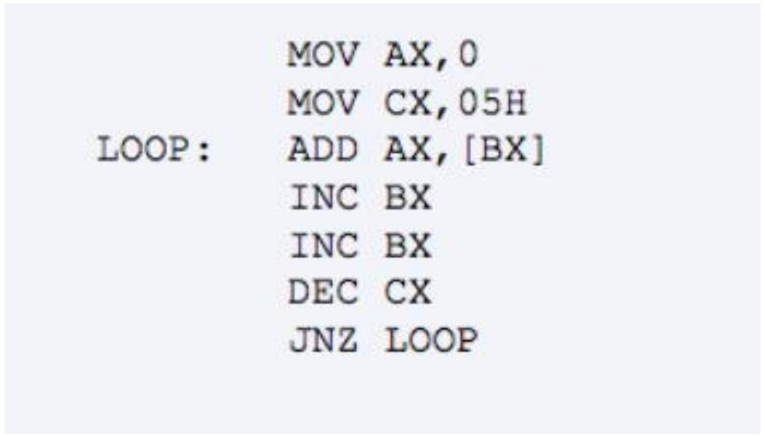


```
int sum(int a[5])
{
   int i,s ;
   s = 0;
   for(i=0;i<5;i++) s = s + a[i];
   return s;
}
```

Adding Example

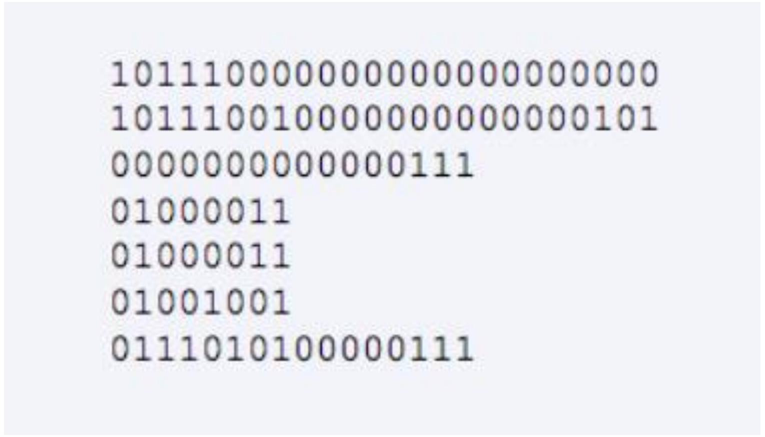
- Assembly language: instructions understandable by the computer.
- The above C program translated into instructions for an x86 computer:

Assembly Language Version:



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

Machine Code Version:



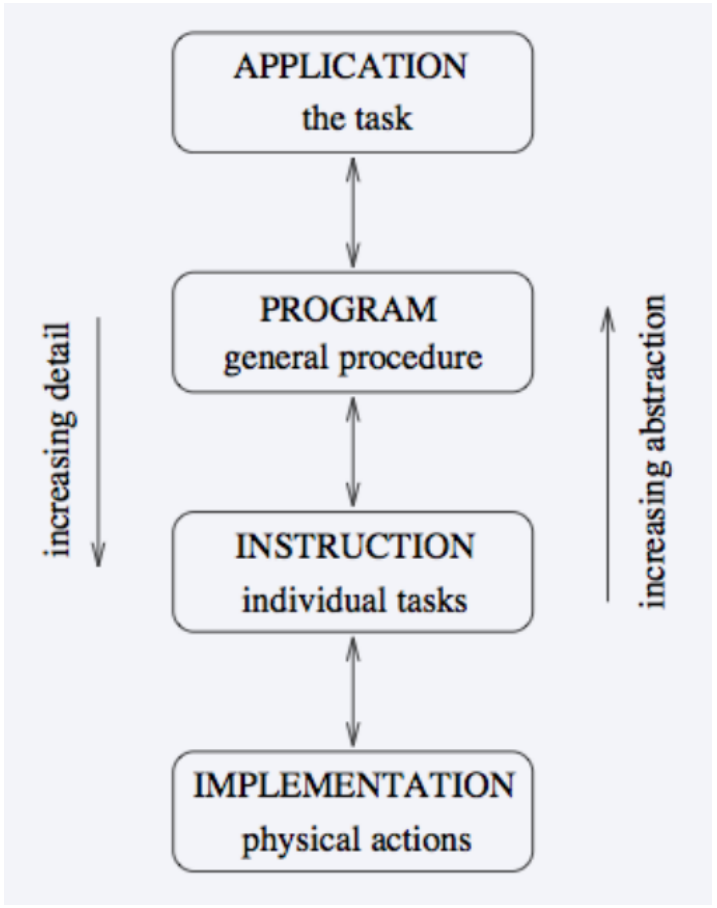
```
101110000000000000000000
101110010000000000000101
0000000000000111
01000011
01000011
01001001
0111010100000111
```

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

Adding Example

- Finally, the physical implementation.

- Abstraction hierarchy 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.

Application Level

- Remember that at the application level we are concerned with the task that the computer can perform.
- It might be a 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.
- Moreover, the basic features are the same: input, output, a means of communication, and a place to record the data.

Program Level

- Recall that the 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.
- There are many programming languages e.g. C, Java, Python, Pascal, Fortran, and although they 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

- At the 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

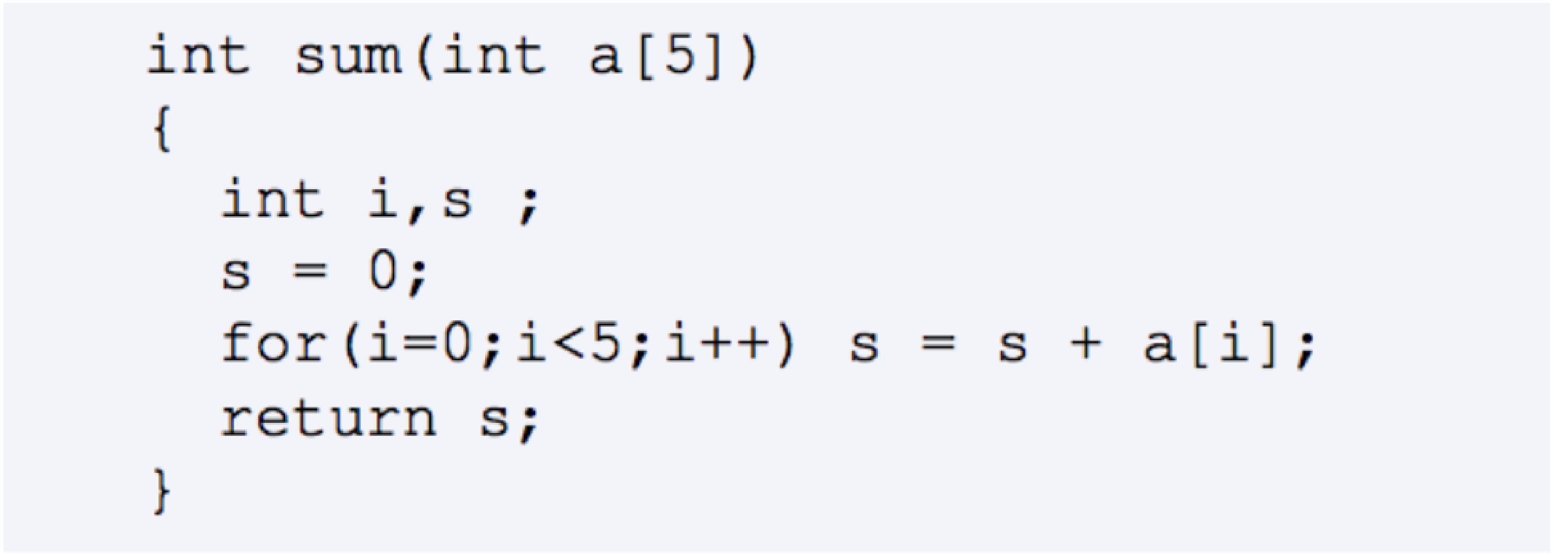
- There are two instruction levels – the difference being the language in which the instructions are specified.
- Level II consists of assembly language instructions, which form a very primitive language.
- Although it still consists of words and numbers, it is much less readable than C or Java etc. 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

- Thus, in going 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:



```c
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:

```
101110000000000000000000
101110010000000000000101
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

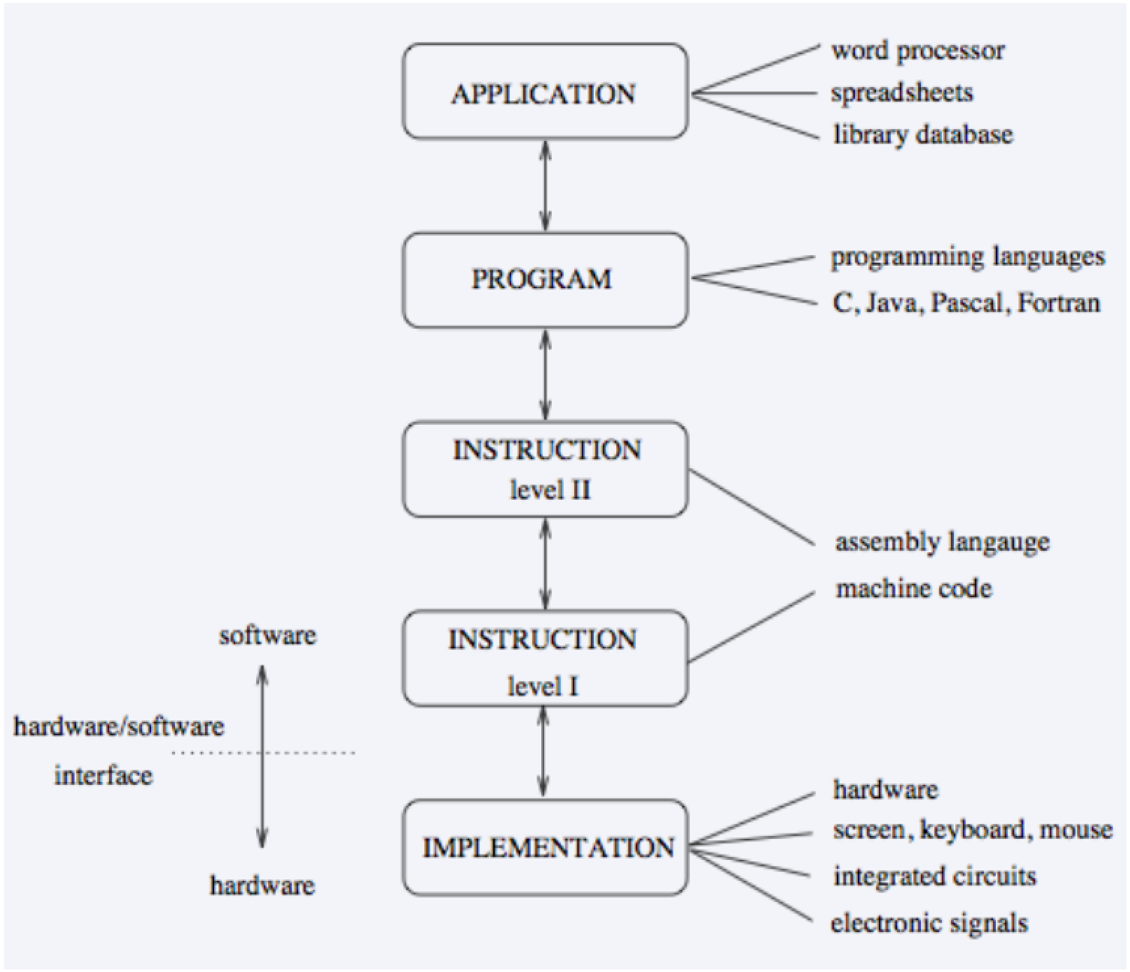
- We need to say something about hardware and software, and in particular the interface between them.
- By software we mean everything above the implementation level.
- The term “soft” derives partly from the fact that it is “ware” that does not physically exist – it is the program or set of instructions that define the operation of the computer; it has no physical form.
- 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

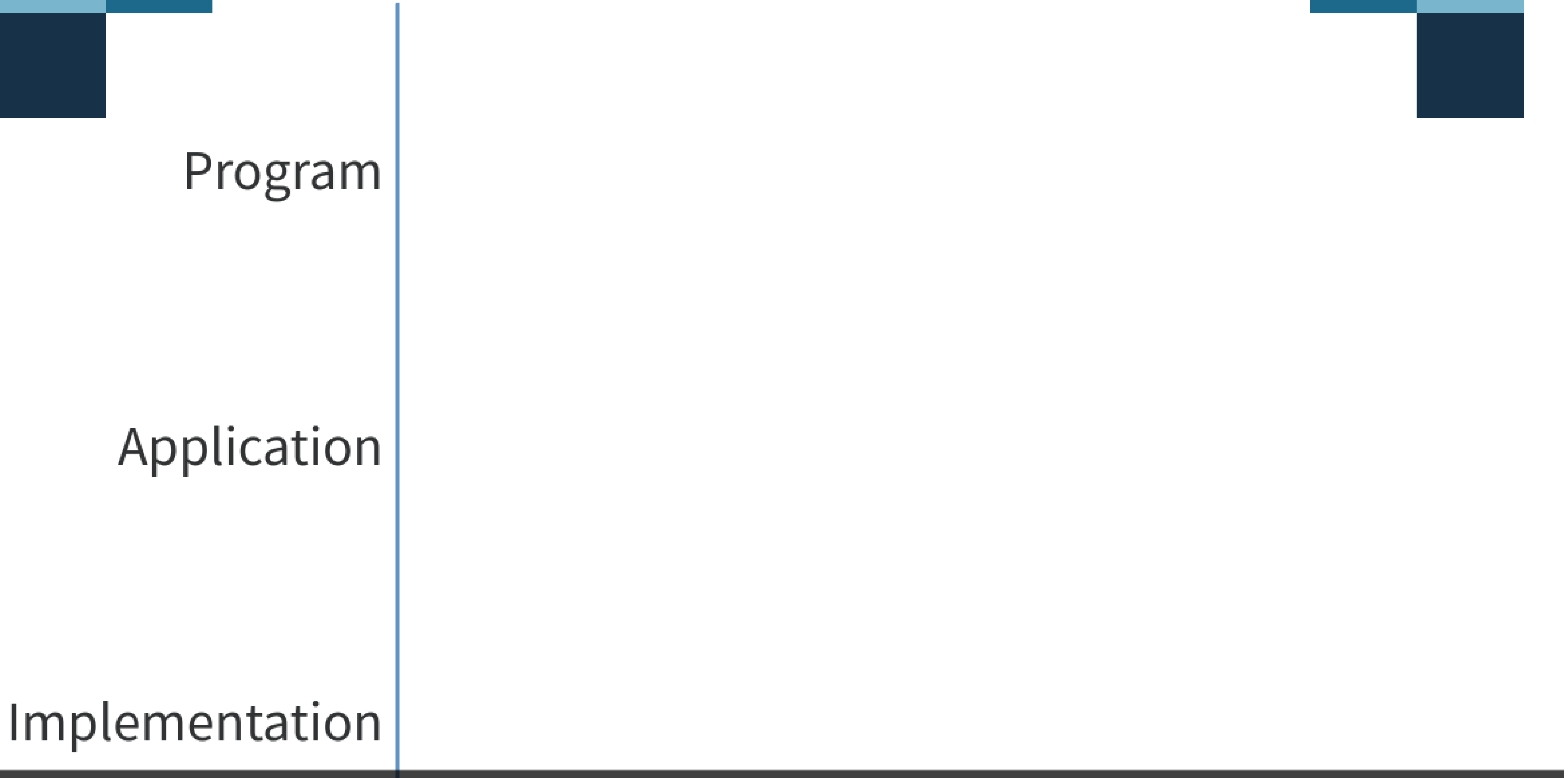
An Abstraction hierarchy for a modern computer



___ level, a program is translated to machine

program	**A**
instruction	**B**
implementation	**C**

_ level deals with the procedure to carry out a



Start the presentation to activate live content

If you see this message in presentation mode, install the add-in or get help at PollEv.com/app

The von Neumann architecture

History

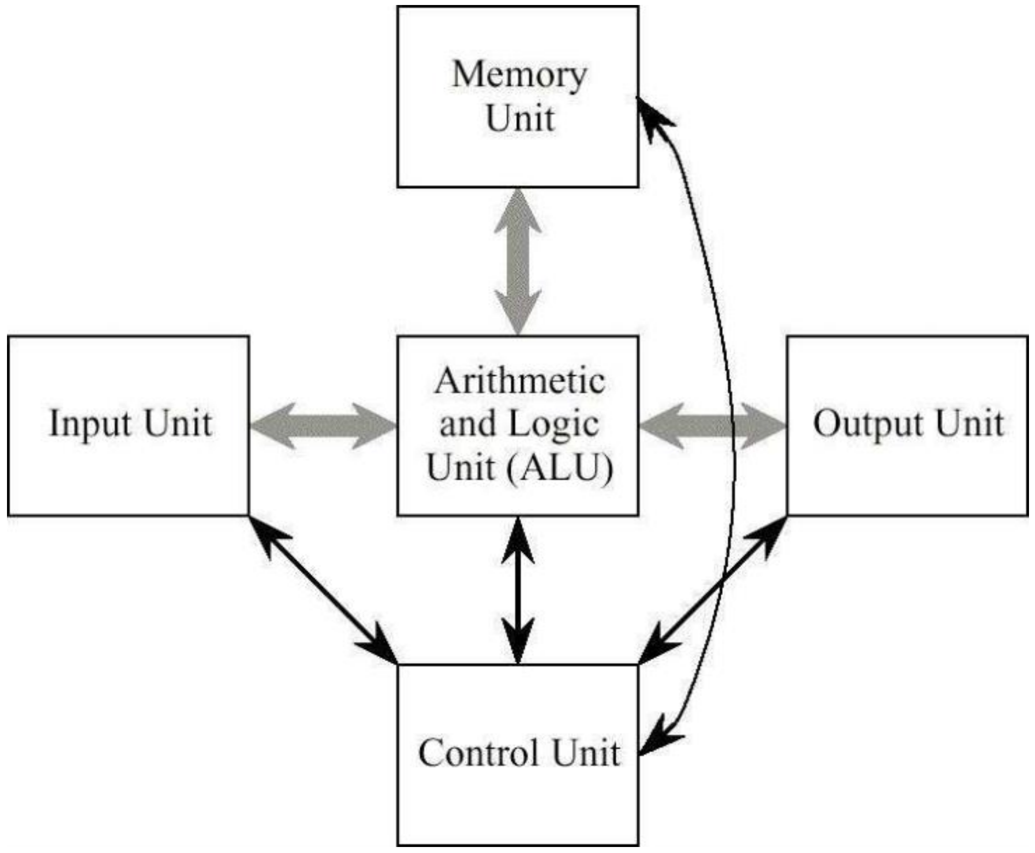
- Early computers were not (re)programmable. To change the instructions to be performed, you would need to rebuild the entire machine.
- An example is ENIAC. It took three weeks to rewire in order to do a different calculation.
- In 1945, just after the World War, Jon Von Neumann proposed to build a more flexible computer.
- Not only the data should be stored in memory, but also the program should be stored in the same memory.
- In 1948, Manchester Small-Scale Experimental Machine (SSEM) made the first successful run of a stored-program.

The von Neumann architecture

- A stored program computer in which the program and the data are kept in the same memory is said to have the von Neumann architecture.
- Effectively the program by itself is treated as data.
- A computer built with this architecture would be much easier to re-program.
- Virtually every general purpose modern day computer is based on the von Neumann architecture.
- Non von Neumann architectures (e.g. Harvard architecture) are only used in systems with very specialist requirements.

The von Neumann architecture

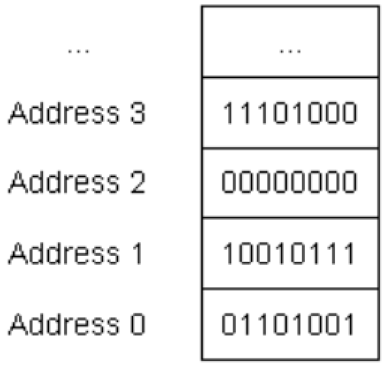
- Five key components:
  - Control unit
  - Arithmetic-logic unit
  - Memory unit
  - Input unit
  - Output unit



(Thick arrows show data paths and thin arrows represent control paths.)

Memory unit

- Memory is a collection of cells, each having an address.
- An address uniquely identifies a memory cell. No two different memory cells can have the same address.



...	...
Address 3	11101000
Address 2	00000000
Address 1	10010111
Address 0	01101001

- The number of bits in each addressable location is known as the memory's addressability.
- Example: the x86 is byte-addressable, i.e. there is one address for every 8 bits (byte).

Memory unit

- Most computers have a characteristic unit quantity of data, which can be handled most efficiently/naturally by the computer – the machine word. Usually:
  - Instructions operate on a word at once
  - Data is fetched one word at time.
- Most machines are byte-addressable (1 byte = 8 bits)
- Typically words are an integer number of bytes, and often are 2 bytes (16 bits), 4 bytes (32 bits), and 8 bytes (64 bits).

Memory unit

Example:

A computer has 9-bit words, in the sense that memory cells store words (word-addressable computer). The size of the addresses is 11 bits. What is the max memory capacity of this computer, in bytes?

Solution:

There are 2^{11}=2048 possible addresses, which is the maximum number of (addressable) memory cells. Each cell stores one 9-bit word, which yields $9\times2048=18432$ bits. There are 8 bits in a byte, therefore the final answer is 18432/8=2304 bytes.

Input/output unit

- The input unit:
  - Allows data and program to be entered into the computer.
  - The very first input units received data via punched cards.
  - Modern computers receive input into them from a range of sources, e.g. keyboard, mouse, scanner *etc.*
- The output unit:
  - Is a device through which results stored in the computer memory are made available to the outside world.
  - Modern computers provide output to a range of devices, e.g. screen, printer *etc.*

Arithmetic-Logic Unit (ALU)

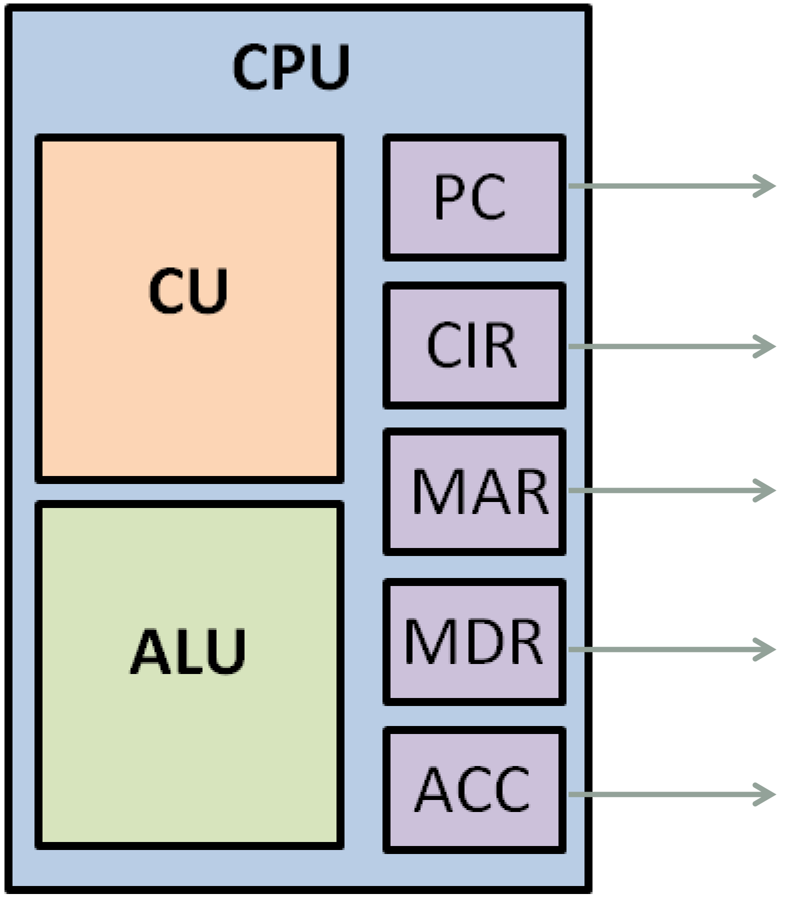
- Capable of performing basic arithmetic operations:
  - $+,\ -,\ \times,\ \div$
- It is also capable of performing logic operations:
  - AND, OR, NOT, $<,\ =,\ >$
- Most modern ALUs have a small amount of special storage known as registers, which are used to store information that will be needed again immediately.
- Typically, each register can hold one word of data.

Control Unit (CU)

- This is the organising force within a computer system.
- It controls the fetch-execute cycle (will be explained shortly).
- The control unit contains two very important registers:
  - The Current Instruction Register (CIR) – at any point in time this register holds the instruction that is currently being executed by the computer.
  - The Program Counter (PC) – at any point in time this register holds the location of the next instruction to be executed by the computer.

Central processing unit (CPU)

- CU and ALU are combined into Central Processing Unit (CPU)



Program Counter

Current Instruction Register

Memory Address Register

Memory Data Register

Accumulator

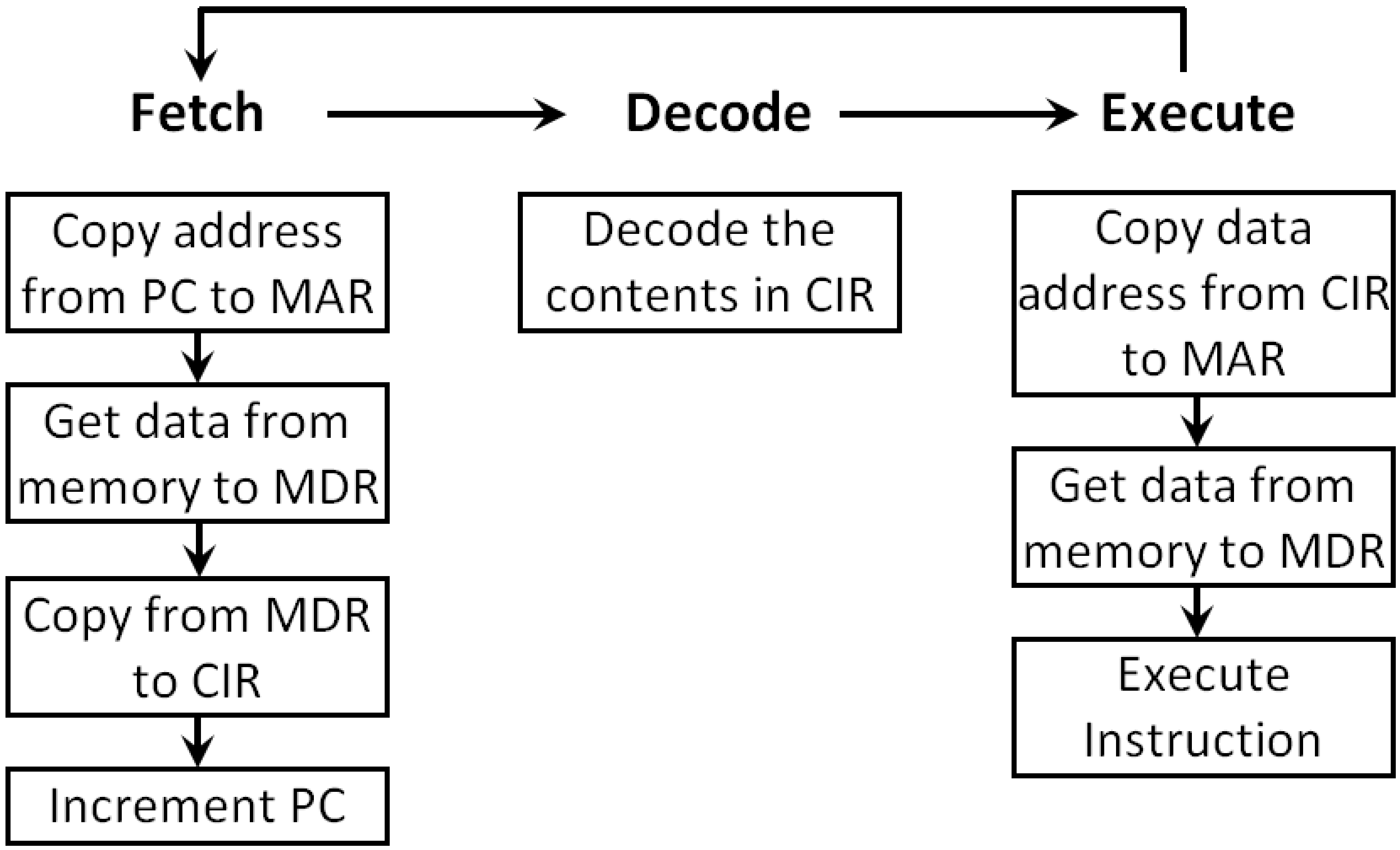
Execution of instructions

- Let us assume that a program has been converted into a set of machine instructions, and the instructions have been loaded into the main memory.
- The CPU will carry out the sequence of instructions.
- It first records the address of the first instruction of the program in its PC, and then proceeds to carry out the following steps:

Fetch-execute cycle

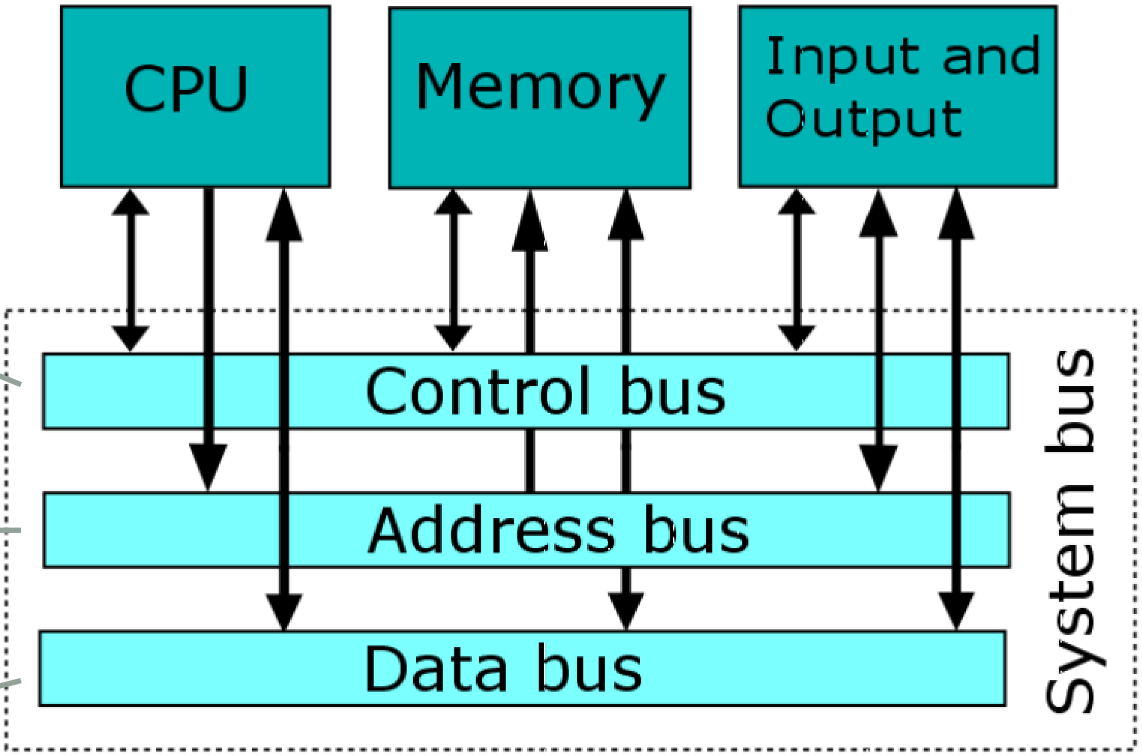
1. Fetch the next instruction (from the address stored in the PC) from main memory into the CIR.
2. Write the address of the next instruction to the Program Counter (usually by incrementing it).
3. Determine the type of instruction just fetched.
4. If the instruction requires data, find out its address in main memory.
5. Fetch the data from its address into MDR.
6. Execute the instruction.
7. Store the result at the specified address in main memory.
8. Repeat from step 1.

Fetch-execute cycle


Fetch
Decode
Execute
Copy address from PC to MAR
Get data from memory to MDR
Copy from MDR to CIR
Increment PC
Decode the contents in CIR
Copy data address from CIR to MAR
Get data from memory to MDR
Execute Instruction

The system bus model

The system bus model is a streamlined von Neumann model with a set of communication lines, known as a bus.



Determine the operation

Determine where to send the data

Carries data

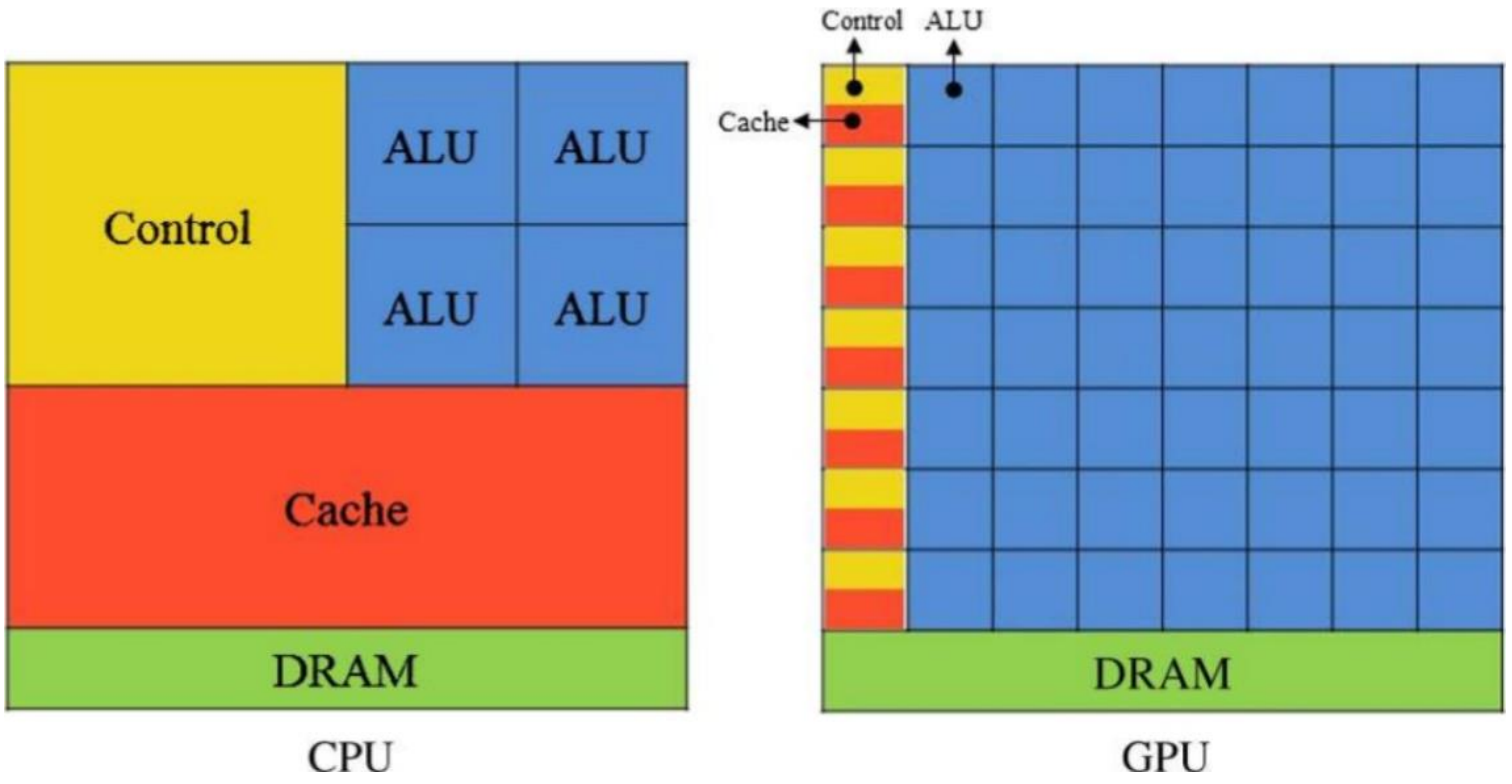
The von Neumann bottleneck

- The separation of the CPU and memory results in the von Neumann bottleneck.
- Data and instructions (held in main memory) are transmitted, one word at a time, to the CPU.
- There is a limited throughput (transfer rate) between the CPU and main memory.
    - The shared bus is busy when fetching/writing data from/to memory, so the CPU has to wait before fetching the next instruction, and vice versa.
    - Memory is physically far away from the CPU and the signal propagation speed is finite.

The von Neumann bottleneck

- Since CPU speed and memory size have increased much faster than the throughput between them, the bottleneck has become more of a problem on modern fast computers.
- The von Neumann bottleneck limits the effective processing speed of the CPU.
- It is particularly pronounced in scenarios when it is required to carry out minimal processing on large quantities of data. The CPU is continually forced to wait for data transfer.
- The bottleneck is a serious drawback for the computers we use today. Possible solutions:
  - Data/instructions cache
  - Parallel computing

CPU vs GPU


Control
ALU
ALU
ALU
ALU
Cache
DRAM
CPU
Control
ALU
Cache
DRAM
GPU

CPU vs GPU

“GPU computing is the use of a GPU (graphics processing unit) together with a CPU to accelerate general-purpose scientific and engineering applications. CPU + GPU is a powerful combination because CPUs consist of a few cores optimized for serial processing, while GPUs consists of thousands of smaller, more efficient cores designed for parallel performance. Serial portions of the code run on the CPU while parallel portions run on the GPU.”

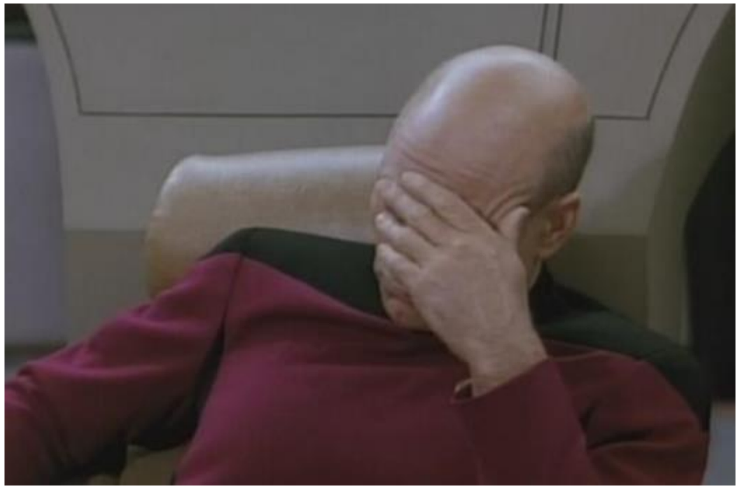
(Source: nvidia.com)

Can fast computers be big?

- Before the invention of ICs, more components meant physically larger computers.
- Even sci-fi writers of the 1960s imagined powerful computers of the future to be big:

- But can fast computers really be big?

Size vs. speed



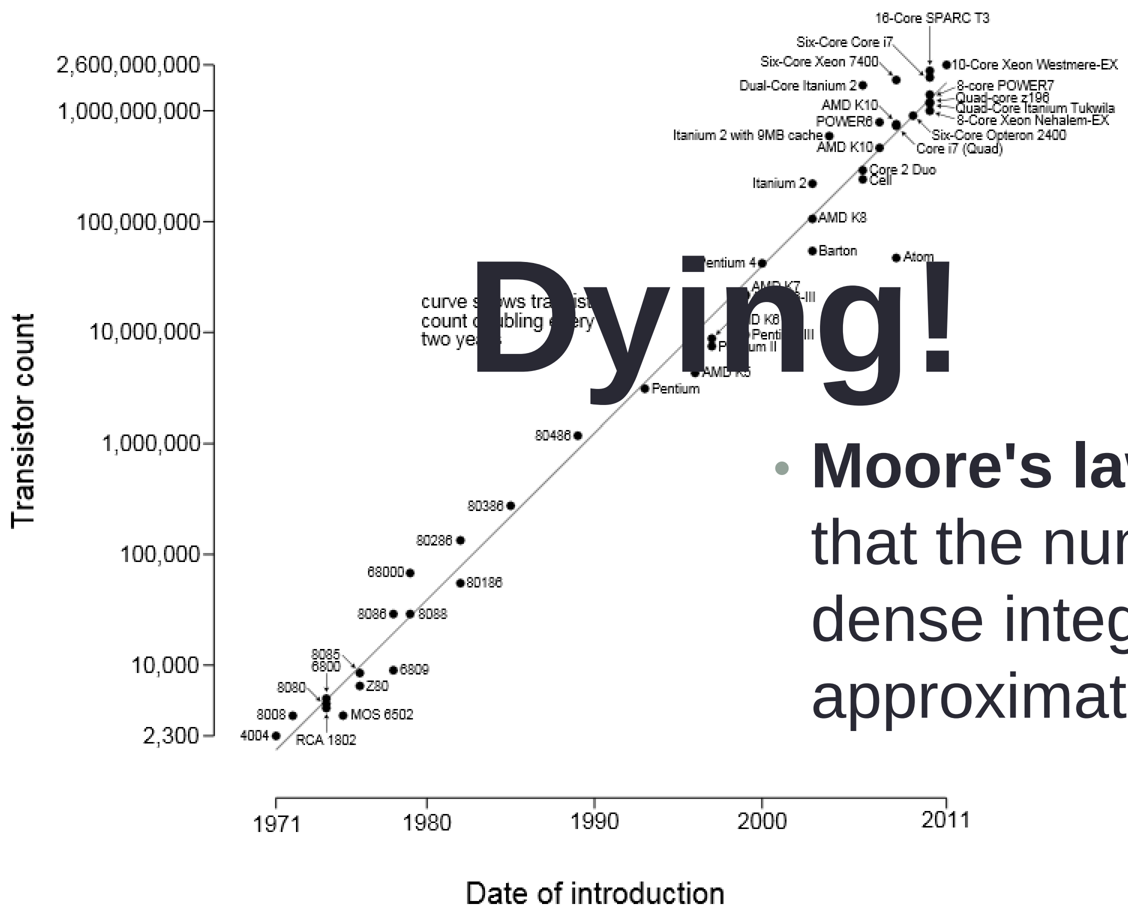
- Signals (e.g. between parts of a computer) cannot travel faster than the speed of light, $c = 299{,}792{,}458\ m/s$.
- The larger the computer the more the time it takes to send information from one part to another!

Example

- Assume a signal has to be sent between terminals (3.5 cm apart) at 3 GHz.
- $s = v \times t$. $t = 1/(3 \times 10^9)$ sec.
- Light travels only 10cm during this time!
- Noticed how CPU speeds got stuck at around 3-4 GHz?
- Main reason for miniaturisation.

Moore's Law

Microprocessor Transistor Counts 1971-2011 & Moore's Law



By Wgsimon - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=15193542

- Attributed to an observation by Intel co-founder Gordon **Moore** in 1965.

- **Moore's law** is the observation that the number of transistors in a dense integrated circuit doubles approximately every two years.

Miniaturisation

Size of a silicon atom = 111 pm = 1.11×10^{-10} m.

Current transistor size on ICs ≈ 10nm = 1.00×10^{-8} m.

So, the size of a transistor on ICs is only about 100 silicon atoms across!

…in terms of size of transistors you can see that we're approaching the size of atoms which is a fundamental barrier…

– G. Moore

So, fast computers must be small, but cannot be infinitely small. Solution? Probably parallel computing.

you call the characteristic quantity of data, as a unit, most efficiently by a computer arch

Bit **A**

Byte **B**

Word **C**

Start the presentation to activate live content

If you see this message in presentation mode, install the add-in or get help at PollEv.com/app

on Neumann architecture, data and instructi
parately, in physically distinct memory bank
false?

any different values can be stored in a 6-bit m cell?

6

36

64

Start the presentation to activate live content

If you see this message in presentation mode, install the add-in or get help at PollEv.com/app