ECS404: Computer Systems and Networks

Prof E.P. Robinson and Dr A. Alomainy

April 5, 2018

Contents

Aims

The aim of this course is to provide students with a basic understanding of how a computer works, how programs are executed by the CPU at the machine level, and how computer networks function. They will gain this firstly by studying the major components of a computer, the interaction between them, and how computers communicate over networks. Secondly, they will learn how data is represented to be processed by computer. Thirdly, students will learn some assembly language and understand how high-level programming concepts are related to their machine language implementation. Finally, they will learn how computer networks function, and will understand how low-level network traffic implements communication between computers.

Summary

The course presents the concepts needed to understand typical computers at the level of their 'machine-code' instruction set, and to understand the basic concepts of computer networks.

The material covered includes

1. the major components of a computer, including CPU, memory, I/O and buses and the role of bandwidth, latency and power dissipation in determining the relationship between them.

2. the use of bits, bytes and data formats to represent numbers, text and programs

3. CPU structure and function: the conventional (von Neumann) computer architecture

4. data types, addressing modes and instruction sets

5. machine-level program structure and its correspondence to higher-level programs

6. the role of wired and wireless networks in modern computer systems

7. a basic understanding of typical network technologies, e.g. ethernet, wifi

8. the role of protocols such as ethernet in the implementation and use of network technology

Part I

Basic Computer Architecture

Chapter 0

Introduction

0.1 Aims

These are the overall aims of the course, with the aims dealt with by this part picked out in bold.

The aim of this course is to provide students with **a basic understanding of how a computer works,** how programs are executed by the CPU at the machine level, and how computer networks function. **They will gain this firstly by studying the major components of a computer, the interaction between them,** and how computers communicate over networks. Secondly, they will learn how data is represented to be processed by computer. Thirdly, students will learn some assembly language and understand how high- level programming concepts are related to their machine language implementation. Finally, they will learn how computer networks function, and will understand how low-level network traffic implements communication between computers.

In more detail: The aim of this part of the course is to familiarise you with the way a computer is built. There are three over-arching themes:

- **taxonomy:** an analysis of the different kinds of devices, so that you know the range of devices that your studies will cover

- **construction:** concentrating on how the devices are built, so that you understand the basic components of a typical computer, something of how the components are made, and a reasonable amount about how they are put together to make a working machine

- **history:** to give you a sense of how rapidly computers have developed and to give you a feel for the way they are likely to develop in the future (this is the least emphasised of the themes).

In addition there is a section on power, covering the energy use of computers and IT generally, with the aim of helping you to understand why the energy use of IT is technically, commercially, and ultimately socially important.

0.2 Summary

This is the overall summary of the course, with the items dealt with by this part picked out in bold.

The course presents the concepts needed to understand typical computers at the level of their 'machine-code' instruction set, and to understand the basic concepts of computer networks.

The material covered includes

1. **the major components of a computer, including CPU, memory, I/O and buses and the role of bandwidth, latency and power dissipation in determining the relationship between them.**

2. the use of bits, bytes and data formats to represent numbers, text and programs

3. **CPU structure and function: the conventional (von Neumann) computer architecture**

4. data types, addressing modes and instruction sets

5. machine-level program structure and its correspondence to higher-level programs

6. the role of wired and wireless networks in modern computer systems

7. a basic understanding of typical network technologies, e.g. ethernet, wifi

8. the role of protocols such as ethernet in the implementation and use of network technology

In more detail:

- **Different types of computer:** (taxonomy, link to history) an analysis of the different kinds of devices,

- **Basic device-level architecture:** looking at how the different kinds of devices are constucted

- **Integrated Circuits:** aka computer chips. An introduction to how they work leading up to the construction of adders and simple memories.

- **Memory:** the various forms of computer memory and how they are used.

- **Internal comms:** how the different parts of a computer talk to each other, the use of buses.

- **IO:** a short section on how IO is handled.

- **Power:** an increasingly important topic for small devices and for large server farms, we provide an introduction.

- **History:** this centres on Moore's Law and the way is has enabled an astonishing development in computer power that not only makes computers faster, but makes different forms of computation feasible to carry out on available devices.

Chapter 1

Different types of computer

Aims

The aim of this part of the course is to introduce the different kinds of computers you might encounter and at some point need to program.

Learning Objectives

By the end of this chapter you should:

☐ understand what different sorts of computers there are

☐ be able to define the concepts of **microprocessor** and **microcontroller** and understand what they are

☐ be able to identify whether an artefact contains a microprocessor or microcontroller with reasonable accuracy

☐ be able to identify when an artefact is not likely to contain either.

Check these off when you are confident you have achieved them.

1.1 What makes something a computer?

There are lots of computers about. Some of them are obvious and others more hidden, either as server farms hidden away in big warehouses, or as small special-purpose devices hidden away inside things you don't think of as computers. Still others are devices that are structured and built as computers but which you might think of as something else. At the top end there are questions about whether a warehouse full of machines is a collection of many different computers or a single massively parallel computational device. At the bottom end, you can ask whether a basic microcontroller is really a computer or just a control device.

The view we take is this: complex concepts have fuzzy definitions. There are things on the boundaries where you can argue either way. Having a precise definition will not be important to us. What will be be important is understanding the rough location of the boundary and what might put a device on one side or the other.

1.1.1 Some basic examples

Here is a list of devices to get things going. We'll expand it later.
Computers obvious to everybody:

- Mainframe servers
- Desktop PC's (personal computers)
- Laptops

Computers marketed as general-purpose devices:

- Smart phones
- Tablets

Computers hiding in plain sight disguised as special-purpose devices:

- Games consoles
- Routers

Anything that contains a microprocessor

- Consumer devices (mircrowaves, ovens, burglar alarms,...)
- Cars

1.1.2 Making things a bit more precise

We're going to focus on **programmmable digital computers.** Unpack this.

Definition [Computer]: A device or machine for performing or facilitating calculation; An electronic device (or system of devices) which is used to store, manipulate, and communicate information, perform complex calculations, or control or regulate other devices or machines, and is capable of receiving information (data) and of processing it in accordance with variable procedural instructions (programs or software) (Oxford English Dictionary)

This definition is in partly in terms of **function**, i.e. what it does:
performing or facilitating calculation and **store, manipulate, and communicate information, perform complex calculations, or control or regulate other devices or machines, and is capable of receiving information (data) and of processing it in accordance with variable procedural instructions (programs or software)**

It is also partly in terms of **implementation**, i.e. how it is constructed: **An electronic device (or system of devices)**.

The distinction between **function** and **implementation** is at the heart of component-based design, hardware and software.

According to the OED, one of the things a computer does is run **programs** (see later).

Definition [Digital]:

1. (of signals or data) expressed as series of the digits 0 and 1, typically represented by values of a physical quantity such as voltage or magnetic polarization. Often contrasted with analogue. (OED)

2. Property of representing values as discrete, usually binary, numbers rather than a continuous spectrum. (Wiktionary)

Digital refers to the representation system used by the computer. All computers have to represent things in the real world in some way. Digital computers use a representation system based on some fixed finite alphabet. We use a digital representation system when we write down numbers: the alphabet is the decimal digits 0..9 plus a few additional signs (-, ., e).

Digital contrasts with **analogue**.

Definition [Analogue]: of a device or system) in which the value of a data item (such as time) is represented by a continuous(ly) variable physical quantity that can be measured (such as the shadow of a sundial) (Wiktionary)

Definition [Analogue Computer]: A computer that performs computations (such as summation, multiplication, integration, and other operations) by manipulating continuous physical variables that are analogues of the quantities being subjected to computation. The most commonly used physical variables are voltage and time. Some analogue computers use mechanical components: the physical variables become, for example, angular rotations and linear displacements. (OED CS)

Example: Before electronic calculators and digital computers, engineers used simple analogue calculators called slide rules.

Definition [Program]: A set of statements that (after translation from programming-language form into executable form ...) can be executed by a computer in order to produce a desired behaviour from the computer. (OED Computer Science)

The point here is that the hardware is multi-purpose. What it does can be changed by giving it different instructions, a different program.

Some other definitions are useful.

Definition [Electronic]: (of a device) having or operating with components such as microchips and transistors that control and direct electric currents. (OED)

Definition [Microprocessor]: A semiconductor chip, or chip set, that implements the central processor of a computer. Microprocessors consist of, at a minimum, an ALU and a control unit.

Definition [Microcontroller]: A single-chip computer designed specifically for use in device control, communication control, or process-control applications. It typically comprises a microprocessor, memory, and input/output ports. A typical microcontroller might have a relatively short word length, a rich set of bit-manipulation instructions, and lack certain arithmetic and string operations found on general-purpose microprocessors. (OED Computer Science)

1.2 Self-test

Check these questions off when you are confident you have answered them correctly. Be warned, sometimes the questions can be argued either way. A good answer might do that:

☐ 1. Which of these devices use **analogue** representations, and which use **digital**?

- ☐ *a*) an ordinary tape measure
- ☐ *b*) a car speedometer
- ☐ *c*) a car odometer (measures the mileage)
- ☐ *d*) a clock

☐ 2. Which of these devices are **electronic**, and which are simply **electrical**?

- ☐ *a*) a vacuum cleaner
- ☐ *b*) a radio
- ☐ *c*) a cement mixer
- ☐ *d*) a pair of standard headphones

☐ 3. Identify at least two devices from your home that you are confident contain **microcontrollers**, and explain why you believe they do. (See later if you are unsure).

1.3 Different types of computer

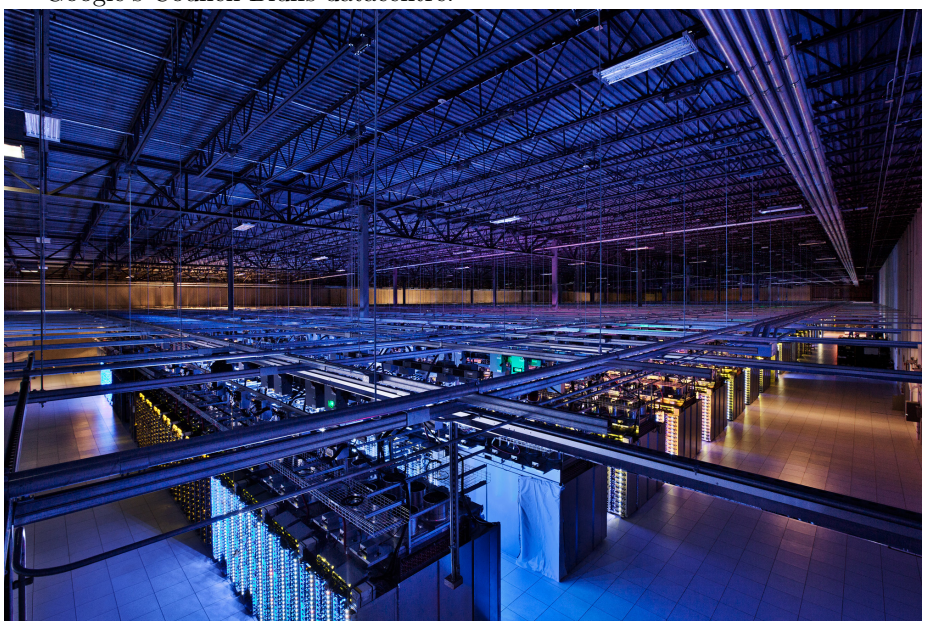
In this section we give a rough taxonomy (or classification) of the different kinds of computing devices around. It is arranged in approximate order of computing power and (generally) price.

1.3.1 Servers and server farms

Companies like Google, Amazon and Facebook (as examples, there are also many many others) use massive amounts of computing power. That computing power takes the form of server farms. A typical server farm consists of a room containing racks of high-end computers all networked together. The size of room can range from being about the size of a normal room in a house to a large shed covering the area of a football pitch. These facilities use a lot of power, and cooling is a major issue. The annual cost of power is a significant fraction of the build cost, and the cost of providing cooling can be about as much as that of the computers themselves.

For more information about how these datacentres are built, and matters such as energy costs, see: https://www.google.com/about/datacenters/ and for some great pictures: https://www.google.com/about/datacenters/gallery/#/

Google's Council Bluffs datacentre:



A standard rack is 19in (480mm) wide, and units are in multiples of 1.75in (44.5mm) high (a bit less to allow removal). Rack-mounted servers are often flat boxes. Sometimes they are vertical boxes intended to be mounted in a chassis that allows them to share things like power supplies and network connections. This is an HP Proliant BL660c (reference https://www.hpe.com/h20195/v2/GetDocument.aspx?docname=c04543743). This particular server can have up to four cpu chips, each with up to about 20 cores, up to 2TB of RAM and four hard disks.

1.3.2 Personal computers and laptops

These are the computers we see at home and in offices. There are quite a few different sorts:

- old-style tower PC's with a separate box for cpu, RAM memory and disk, built out of standardised components
- higher-end laptops that follow standard PC architectures
- all-in-one PC's, basically a laptop (minus battery, keyboard and touchpad) packed into a screen
- low-end laptops, basically a tablet with keyboard and touchpad.

1.3.3 Mobile phones and tablets

The apps on smartphones and tablets are programs. An Android or iOS smartphone or tablet is basically a small computer running a form of the Unix operating system, and an app is a program running on it (technically both Android and IOS are based on Unix kernels, Android on a linux kernel and IOS based on a proprietary Apple kernel). There is a bit more to it than that: apps have to conform to a particular design structure, which is quite complex. Nevertheless, you can write your own apps to run on phones, just as you can write your own programs. Android apps are (mainly) written in Java. IOS apps are written in C++ or Apple's new language, Swift. Both Android and IOS have development environments that will help you generate vanilla apps that you can then extend.

Introductions to app writing for Android are at: `https://developer.android.com/training/basics/firstapp/index.html` and for IOS at: `https://developer.apple.com/library/content/referencelibrary/GettingStarted/DevelopiOSAppsSwift/` We will look at the actual smartphone architectures later.

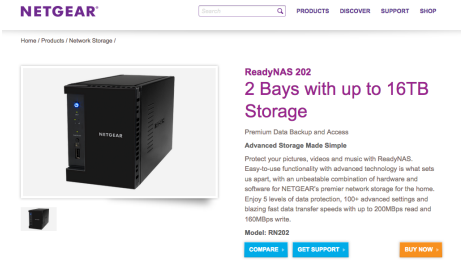
1.3.4 Personal computers disguised as consumer devices

Your house may also contain a number of devices that are basically fully functional computers boxed as something with a special purpose. Many of these run a variety of different apps and/or have software (firmware) that can be updated, and they connect to the internet.

- **(Wireless) router:** a router is a computer whose primary purpose is to direct network traffic. Routers on the internet backbone are powerful computers in their own right. But the router in your own home is much more basic. A common way to construct domestic routers is to build a small basic computer and run a version of linux on it. The advantages of this are that it makes it much easier to re-use existing publicly available code, increasing reliability and security and decreasing cost. Moreover it decouples the control software from the hardware, making it easier to change either independently.

- **Games console:** a games console is basically a desktop PC running slightly different software. Modern games consoles run different apps and connect to the internet for online gaming. Internally they are like PC's but with some different trade-offs made in design, and using the power of having massive production runs to drive construction costs down.

- **Network storage:** lots of homes have network addressed storage that you can use for backups of laptops and PC's as well as centralised storage of music, photos and video. These also run different apps and have firmware that can be upgraded. They are basically a small computer with disks attached.



1.3.5 Compute-heavy consumer devices

We also have many devices that carry out significant amounts of computation, and which themselves may be internet-connected. They may contain microprocessors or high-end microcontroller devices (the difference is disappearing as more computation and control is being placed on individual chips).

1. **Television:** Digital television signals are basically mp4 encodings. Televisions take those encodings and display them on the screen. That takes significant computing power. Many of televisions are now "smart", which

means internet enabled, and they run apps for things like iPlayer and YouTube.

2. **Camera:** Most cameras, whether still or video, do a lot of processing (face recognition, picture encoding and decoding, camera control, ...). They can also be internet enabled.

3. **Printer:** Printers are also networked, and may well do a lot of processing (the simplest ink-jets, though, get the computer you are printing from to do all the work).

4. **Cars:** these use extensive computer-based control covering things like engine-control and monitoring, satnavs,....

In October 2016 there were a number of large distributed denial of service attacks using the massed computing power of internet-enabled devices. The most serious knocked out a number of companies including Twitter. See the BBC report at `http://www.bbc.co.uk/news/technology-37738823` for more detail.

1.3.6 Low-end microcontroller devices

There are lots of low-end devices coming on the market. These range from smart devices for the home, often network enabled, though regular household appliances to children's toys. It is easy to find a microcontroller for just a few pence (search `mouser.co.uk` for microcontrollers and sort by price to find one starting at 22p - if you buy 100). As a result, in most cases if something has a digital control interface, it is likely to incorporate a microcontroller. Examples include: microwave ovens, thermostats, burglar alarms, ovens, digital radios,

1.3.7 Computer checklist

The more of these you can tick off the more likely something is to have a "computer" inside.

- ☐ Runs apps or programs, particularly if you can install these yourself. (Computers, laptops, phones, tablets)
- ☐ Has software or firmware that can be updated. (Routers, network storage).
- ☐ Carries out tasks you know to be computationally complex (Televisions, cameras)
- ☐ Carries out complex configurable control task. (Household items, toys)

Chapter 2

Basic device-level architecture

Aims

The aim of this part of the course is to look inside the box for the types of computers in the last chapter, show what they have in common and how they differ. It is also to introduce the von Neumann architecture, which serves as a basic reference architecture, and to see how computers still conform to it, and, of course, some of the ways they have moved on.

Learning Objectives

By the end of this chapter you should:

☐ understand that computers contain a cpu, short and long-term memory, io devices, and internal comms,

☐ be able to explain the basic roles of the components above

☐ have seen how these components are built into modern computers, including servers, laptops, tablets and mobile phones

☐ be able to describe the standard von Neumann architecture

☐ be able to explain ways in which modern architectures conform to von Neumann, perhaps with some elaboration, or differ from it.

2.1 Basic Components

In this section we look at the basic components and architecture of these types of computer.

We will see that all computers have five basic sorts of components:

- one or more **central processing units** (**cpu's**) to control the machine and actually carry out computation
- memory to store data and programs, almost always split into
  - short-term memory to store working data and programs which is cleared when the machine is turned off (e.g. RAM).
  - long-term memory that stores data and programs permanently (e.g. disk).
- devices that enable the computer to interact with its users and with the outside world (**Input-Output** or **IO**) (e.g. screen, wifi controller).
- a communication mechanism that enables these devices to talk to each other (often provided by the system board).

These components have been present since very earliest modern computers, when the basic reference architecture was set out by John von Neumann in a paper he wrote for the US government: "First Draft of a Report on the EDVAC" (1945).

2.2 Background: a name you should know - John von Neumann

John von Neumann was a Hungarian mathematician working in the US before and after the second World War. Amongst other things he made fundamental contributions to:

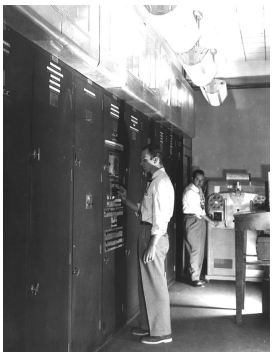
economics: he invented game theory, a basic tool of modern economics in which a system is described as a kind of board game between different players

computer programming: where he did a great deal of work in numeric programming, and in particular invented Monte Carlo methods, one of the basic pillars in machine learning.

During the Second World War he was employed by the US government as a kind of mathematical hitman, and at one point was asked to explain the new electronic computers that were being produced. His report was the result (there is no second draft, and he didn't actually formally publish it).

2.3 The von Neumann Architecture

A diagram of the von Neumann architecture is given in the figure. It includes:



First Draft of a Report
on the EDVAC

by

John von Neumann

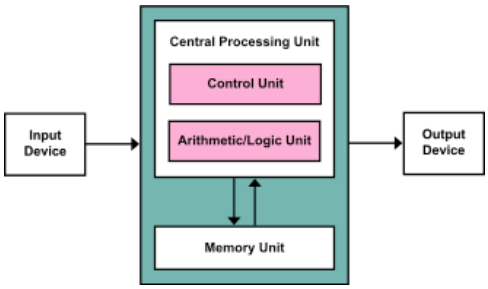
Contract No. W–670–ORD–4926

Between the

United States Army Ordnance Department

and the

University of Pennsylvania



2.0 MAIN SUBDIVISIONS OF THE SYSTEM 1

Figure 2.1: von Neumann, EDVAC and the von Neumann architecture

- a single cpu divided into a control unit and an arithmetic/logic unit (the division is not regarded as an essential feature)
- memory (not divided into short and long-term as disks and other long-term storage had not been invented at the time)
- input and output
- a basic comms structure that includes a single two-way channel between memory and cpu (an essential feature).

2.4 PC architecture

In a standard PC the basic architecture can be seen from the **system board** (or **motherboard**. This board carries the critical components of the PC and contains the basic comms structure linking them.

Figure 2.2 contains a diagram of a motherboard made by the company Gigabyte. This was used in PC's in the ITL in about 2006, and shows a fairly typical structure for tower-based PC's. Compare this to the von Neumann architecture:

1. the cpu (this board uses an AMD processor, not an Intel one).
2. the RAM memory, directly connected to the cpu as in von Neumann.
3. the hard disk(s) would be Serial ATA devices, as would be any optical disks (CD's and DVD's).
4. there are various classes of IO devices including USB and Firewire (IEEE 1394)
5. the communication channels all go via a central comms chip except for the connection to the RAM. There is a single connection from the comms chip to the cpu (the "Hyper Transport Bus").

There are obvious links with the von Neumann architecture (single cpu, single channels from memory to cpu, classification of subdevices into cpu, memory and IO, ...). There are also differences or at least refinements: more than one kind of memory, disks become generic SATA devices, IO devices become generic USB devices, etc.

Here is a physical picture of a similar computer. Note that there are lots of separate components connected by wires. These can be changed by users to either enhance the computer or fix faults. Note also the large fan in the centre that you can just see is covering an equally large piece of finned aluminium. This is the heat sink and fan over the cpu. The cpu typically might use over 100W. This is to get rid of that heat, which would otherwise fry it.

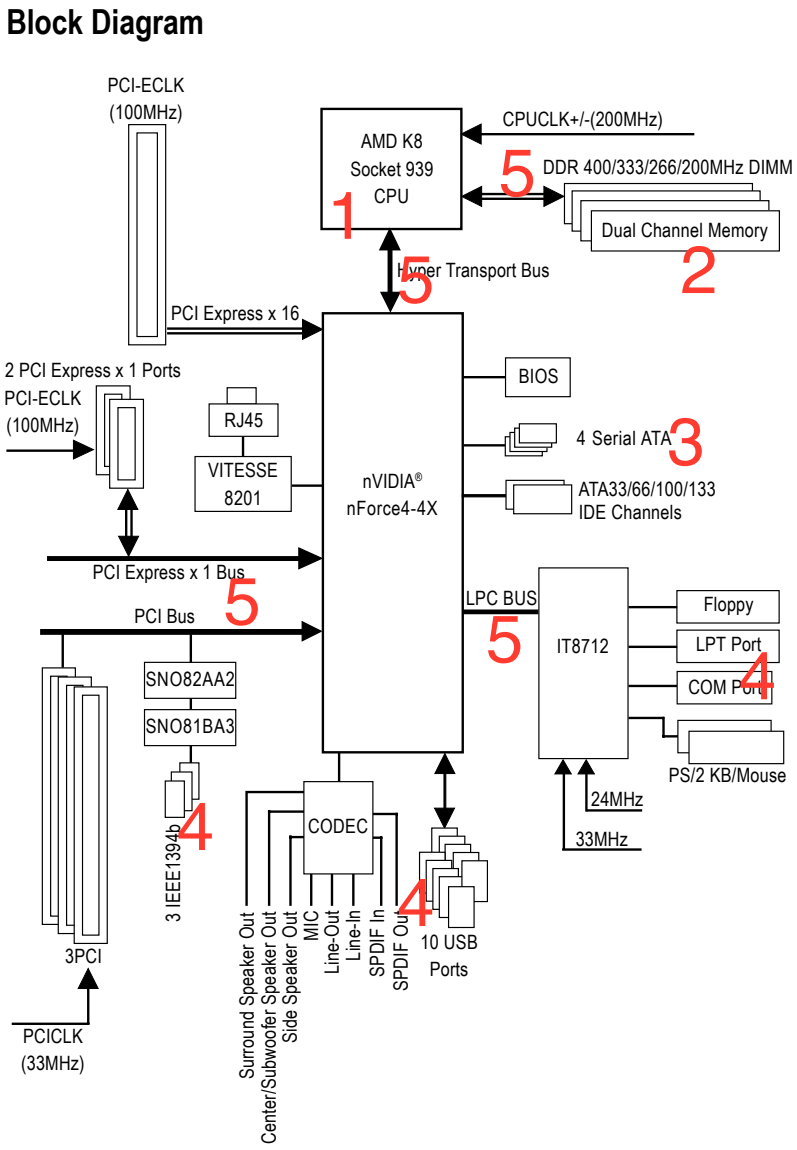
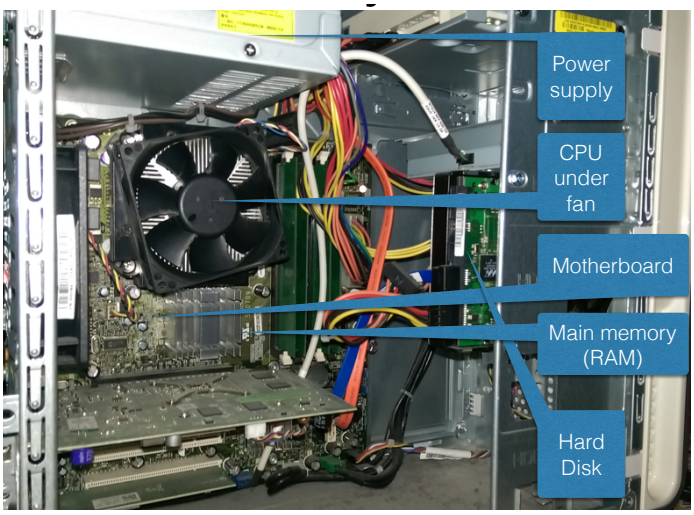


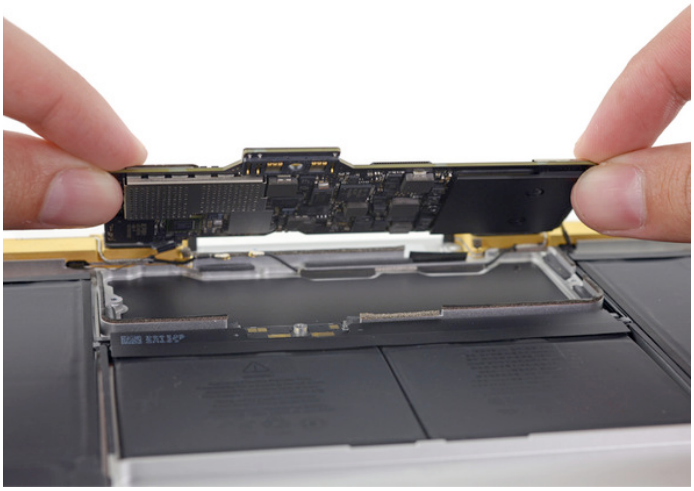
Figure 2.2: A Gigabyte PC motherboard

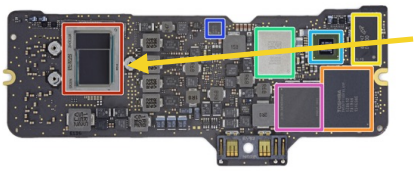


A laptop tends to have fewer user-replaceable devices, with many of its components being soldered onto the motherboard. However the logical architecture is similar to the conventional PC's. Modern all- in-one PC's are much like laptops, as can be seen in this photograph:

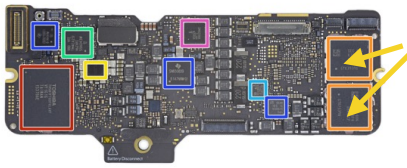
2.5 Laptop architecture

Modern laptops have gone even further. Here is a series of pictures of a Macbook Air from 2015. The first shows the motherboard in the context of the computer itself. The remaining pictures show what is on the motherboard. This is now a consumer commodity device, mass-produced with little or nothing that can be done in the way of customisation or replacement of parts.

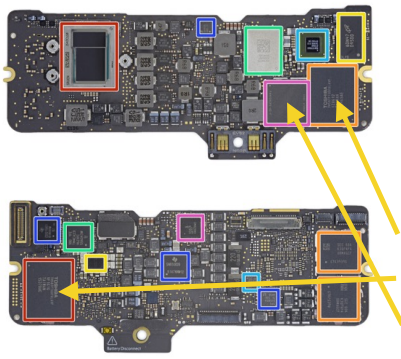




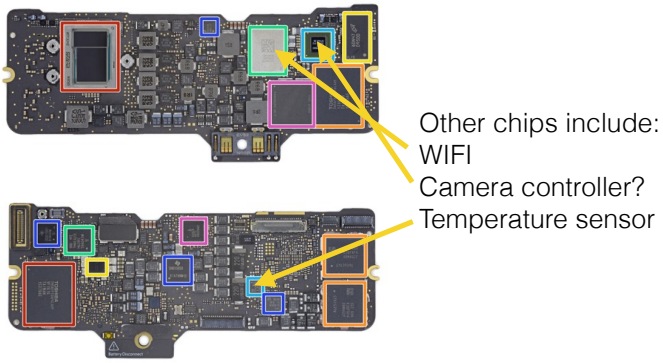
CPU: Intel SR2EN
Intel Core m3-6Y30
Processor



Main Memory: 2 x
Samsung K3QF4F4
4 GB LPDDR3 RAM
(total 8 GB)
just behind the cpu



Long-term memory:
solid state disk
consisting
of 2 x Toshiba
TH58TFT0DFKLAVF
128 GB MLC NAND
Flash mounted
front
and back
with
controller (beneath).

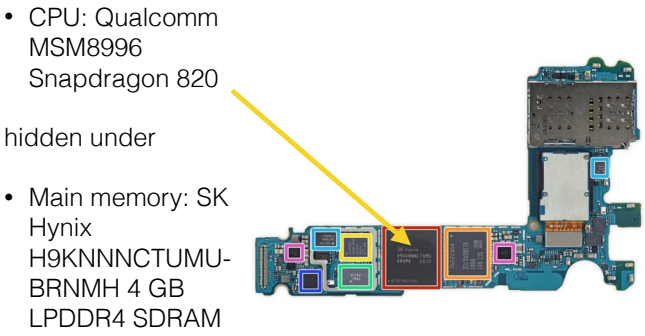


This shows that the Macbook has the components we expect, even if we cannot see the communication channels between them.

If the insides of a Macbook Air look vaguely as if they might be inside an iPad or a mobile phone, this is not a coincidence. The designs have been converging and the fact that high-end laptops like the Macbooks have a very long battery life is down to the use of low-power components like those used in mobile phones, along with the replacement of power-hungry screens and mechanical disks with led screens and solid-state disks. Note that the Macbook has no fan. It can get away with this because the Intel cpu used draws only 5W.

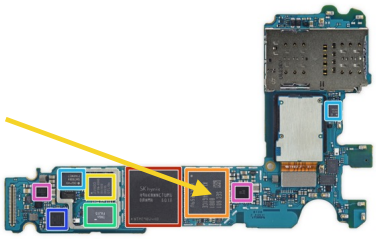
2.6 Mobile phones and tablets

Shown here is the motherboard of a Samsung Galaxy S7. It looks very like the motherboard for the Macbook Air. Chips are soldered on to both sides. The components are broadly similar. The influence of the von Neumann architecture is still apparent.

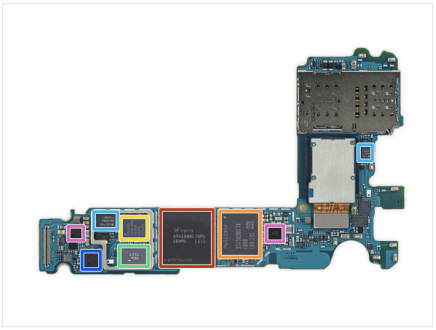


Smartphone construction (Samsung Galaxy S7)

- Long-term memory: Samsung KLUBG4G1CE 32 GB MLC Universal Flash Storage 2.0

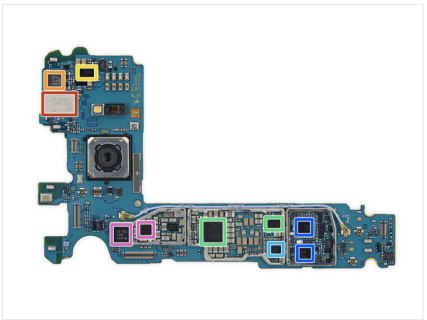


Details from teardown on iFixit



Step 7 Edit 5

- With that, it's time to digitally convey some chip ID. On the front side of the motherboard, we note:
  - SK Hynix H9KNNNCTUMU-BRNMH 4 GB LPDDR4 SDRAM layered over the Qualcomm MSM8996 Snapdragon 820
  - Samsung KLUBG4G1CE 32 GB MLC Universal Flash Storage 2.0
  - Avago AFEM-9040 Multiband Multimode Module
  - Murata FAJ15 Front End Module
  - Qorvo QM78064 high band RF Fusion Module and QM63001A diversity receive module
  - Qualcomm WCD9335 Audio Codec
  - Maxim MAX77854 PMIC and MAX98506BEWV audio amplifier



Step 8 Edit

- With so many similarities to the standard S7's chipset, it almost feels like we're repeating the computer:
  - Murata KM5D17074 Wi-Fi module
  - NXP 67T05 NFC Controller
  - IDT P9221 Wireless Power Reciever (likely an iteration of IDT P9220)
  - Qualcomm PM8996 and PM8004 PMICs
  - Qualcomm QFE3100 Envelope Tracker
  - Qualcomm WTR4905 and WTR3925 RF Transceivers
  - Samsung C3 image processor and Samsung S2MPB02 PMIC

2.7 Rack-mounted computers and servers

Rack-mounted computers are descended from desktop PC's in a somewhat different way. The basic design imperative is to put as much computing power into a box as you can manage for a given price. The way to do that is to put some standard architecture cpu's and as much standard memory as you can get in, with them sharing components such as power supplies, and communication infrastructure.

To illustrate this see the HP Proliant BL680c, below. This has up to four cpu's, and 2.0TB of RAM (that is about 500 times the amount on a basic laptop, or 1000 times the amount on a standard mobile phone).

The first picture gives a block diagram of the basic components and the way in which they are connected.

Differences from von Neumann architecture:

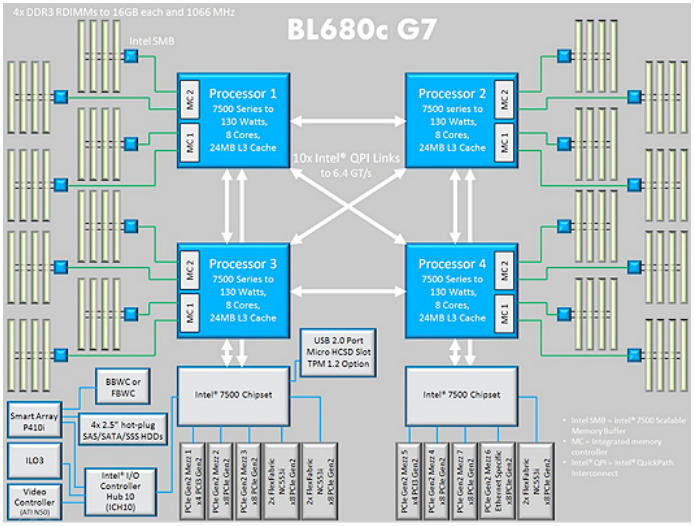
- four cpu's all interconnected (instead of one single one)
- memory split into four bits each connected to single cpu (instead of single memory connected to central processing component)

Similarities with von Neumann architecture:

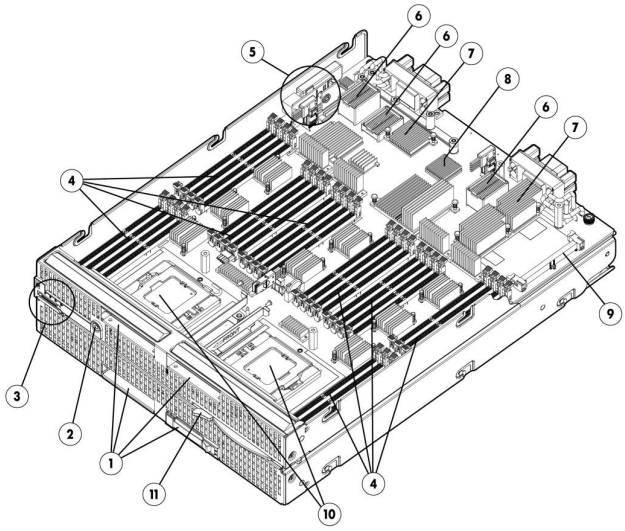
- same general classification of components
- each individual cpu plus its allocated memory is a von Neumann machine

The second picture gives a diagram showing the physical layout. Notice

- there is no power supply because this is intended to go into a chassis that supplies power
- ditto networking
- ditto fan for cooling
- note that the components are aligned front to back. This is so that cooling air can pass over them more easily (see chapter 7 on power). This is not shown in the block diagram.
- notice the rails on the side so that the whole server can be removed from its chassis.



HP ProLiant BL680c Gen7 Server Blade



HP ProLiant BL680c Gen7 Server Blade
Front view image showing side "A"

1. Four hot-plug SAS/SATA/SSD drive bays
2. Power on/standby button
3. UID, health, and network adapter LEDs
4. 64 RDIMM slots supporting up to 2.0TB of DDR3-1333MHz memory (operating up to 1066 MHz) (32 RDIMMs per side)
5. One USB 2.0 port, one MicroSD high capacity (SDHC) port, and one TPM 1.2 connector
6. Seven PCIe Gen2 I/O expansion mezzanine slots (3 on one side, 4 on the other)
7. Six NC553i 10Gb FlexFabric adapter ports (4 one on side, 2 on the other)
8. iLO 3 Management adapter port
9. HP P410i Smart Array flash cache connector
10. Two, three or four Intel® Xeon® 4800 family processors (below the hard drives) (two per side)
11. Server release lever

2.8 Data Centres

This extends the picture beyond what would normally be regarded as an individual machine. But an entire data centre can be regarded as a computational device (see for example the case made by two Google engineers in Barroso, Luiz Andr, Jimmy Clidaras, and Urs Hlzle. "The datacenter as a computer: An introduction to the design of warehouse-scale machines." Synthesis lectures on computer architecture 8.3 (2013): 1-154.

Modern cpu's contain a number of cores, microprocessors in their own right. We have seen that rack-mounted computers can contain several communicating cpu's. In a data centre the computers in a rack can often be linked, and a number of racks are often linked. So the same structure of communicating computational devices is replicated at a number of levels. This kind of structure is called **fractal**.

Google gives a good introduction to their data centres in `https://www.google.com/about/datacenters/`, including detail of their construction.

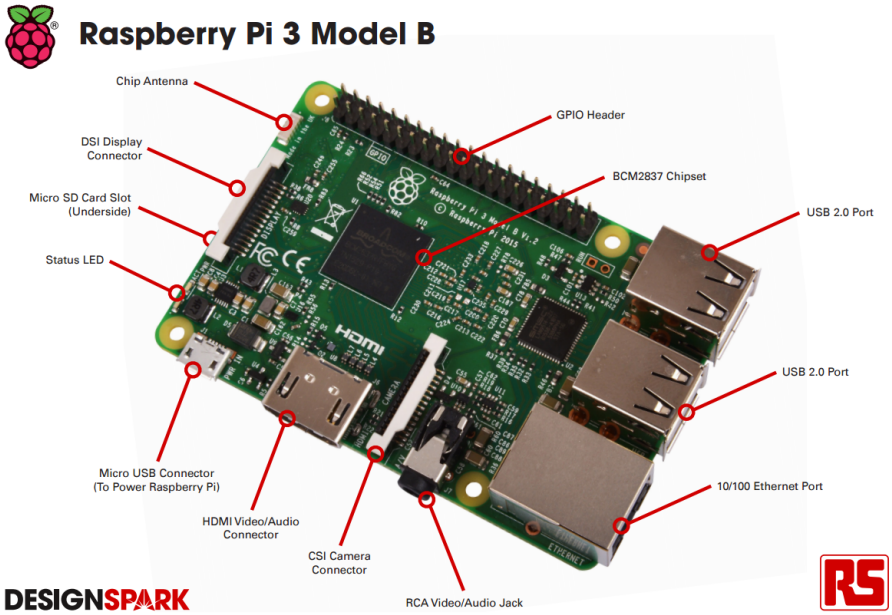
There's an introduction to Microsoft's data centres at `https://www.microsoft.com/en-gb/cloud-platform/global-datacenters`. There's also an introduction to them, including novel forms of construction in an article on ZDet: `http://www.zdnet.com/article/photos-a-tour-inside-one-of-microsofts-cloud-data-centers/`.

2.9 The Raspberry Pi

The Raspberry Pi illustrates the modern architecture for a small machine. It uses a **System on a Chip (SOC)** architecture, where the main chip carries most of the system as well as just the cpu.

Notice that there is a single chip (a Broadcom BCM2835 in the first version, and a Broadcom BCM2837 on the Raspberry Pi 3 version B). This has the cpu and the RAM memory and the various IO devices on it. The CPU uses an ARM architecture, a 1.2GHz 64-bit quad-core ARMv8 CPU.

This illustrates a trend in the production of large-volume consumer systems, where designs for components are bought in and then packaged together on a chip intended for a specific device.


Raspberry Pi 3 Model B
Chip Antenna
GPIO Header
DSI Display Connector
BCM2837 Chipset
Micro SD Card Slot (Underside)
USB 2.0 Port
Status LED
USB 2.0 Port
Micro USB Connector (To Power Raspberry Pi)
10/100 Ethernet Port
HDMI Video/Audio Connector
CSI Camera Connector
RCA Video/Audio Jack
DESIGNSPARK

Part II

Data Representation

Chapter 9

Introduction

9.1 Aims

These are the overall aims of the course, with the aims dealt with by this part picked out in bold.

The aim of this course is to provide students with a basic understanding of how a computer works, how programs are executed by the CPU at the machine level, and how computer networks function. They will gain this firstly by studying the major components of a computer, the interaction between them, and how computers communicate over networks. **Secondly, they will learn how data is represented to be processed by computer.** Thirdly, students will learn some assembly language and understand how high- level programming concepts are related to their machine language implementation. Finally, they will learn how computer networks function, and will understand how low-level network traffic implements communication between computers.

In more detail: The aim of this part of the course is to familiarise you with the way computers represent the data that they are working with. We will cover how numbers and characters are represented in some detail, aiming for you to understand exactly what bit sequences are used. We will then move on briefly to different forms of media, aiming to convince you that the kinds of representation used before can be extended to allow the representation of sound and video. In this context we will also discuss the compression techniques used and their importance.

9.2 Summary

This is the overall summary of the course, with the items dealt with by this part picked out in bold.

The course presents the concepts needed to understand typical computers at the level of their 'machine-code' instruction set, and to understand the basic concepts of computer networks.

The material covered includes

1. the major components of a computer, including CPU, memory, I/O and buses and the role of bandwidth, latency and power dissipation in determining the relationship between them.

2. **the use of bits, bytes and data formats to represent numbers, text and programs**

3. CPU structure and function: the conventional (von Neumann) computer architecture

4. data types, addressing modes and instruction sets

5. machine-level program structure and its correspondence to higher-level programs

6. the role of wired and wireless networks in modern computer systems

7. a basic understanding of typical network technologies, e.g. ethernet, wifi

8. the role of protocols such as ethernet in the implementation and use of network technology

In more detail:

We will cover:

- **Unsigned integers**

- **Signed integers**

- **Floating point numbers**

- **Character sets and text**

- **Media: sound, pictures and video**

Chapter 10

Unsigned integers

Aims

To cover the the basic use of bits to represent data, concentrating on the representation of unsigned whole numbers.

Learning Objectives

By the end of this chapter you should:

- ☐ 1. understand that a bit is a single binary digit, and understand the difference between bits and bytes;
- ☐ 2. understand the correspondence between bit sequences and unsigned integers and in particular:
  - ☐ *a*) be able to translate numbers from decimal to binary and vice versa
  - ☐ *b*) be able to carry out long addition and long multiplication in binary
- ☐ 3. understand the use of hexadecimal to represent bit sequences, and in particular:
  - ☐ *a*) be able to translate between hexadecimal and binary

10.1 Introduction

Computers use binary representations, and at the hardware level these representations are almost always fixed width (i.e. they have a fixed number of binary digits, usually 8, 32 or 64). There are two ways of looking at these representations:

1. as a sequence of binary digits

2. as the binary representation of a number

Example:

1. An IP address is given as a sequence of four numbers: 192.168.0.1 Each number is between 0 and 255, and can be represented as an 8-bit binary sequence. The IP address itself therefore has $4 * 8 = 32$ bits, and can be thought of as a sequence of binary digits in which each group of 8 represents some level of the address.

2. A memory address is given by a fixed length bit sequence, but we think of the memory as being one long sequence of memory words. So it makes sense to think of the address as a number.

This chapter is primarily about how sequences of binary digits represent numbers, how we can convert between binary and decimal and other forms of representation, and how we can adapt familiar algorithms to carry out arithmetic on these binary numbers.

10.2 Types of Numbers

You first meet numbers at primary school, and there you are introduced to different types of numbers in a particular order. First you meet small positive whole numbers. You learn about addition, subtraction and multiplication, and you start to learn addition and multiplication tables. Then you learn about larger numbers and positional notation for numbers bigger than ten. You learn long addition and multiplication. At about the same time you find out about fractions (which we won't study) and negative numbers, and you learn how to add, subtract, multiply and divide. Then you learn about decimals. You learn (again) how to add, subtract, multiply and divide. And you learn that some numbers ($\pi = 3.1415\ldots$) have infinite decimal expansions, and that this is true even of many simple fractions ($1/3 = 0.3333\ldots$). Finally (by which time you'll be at secondary school), you learn that some numbers are simply too large be easily written down as simple decimals, and that you need scientific notation for them (the speed of light is $2.9978 * 10^8 ms^{-1}$ and the mass of an electron is $\ldots$). And you may learn about the square root of -1 and complex numbers.

The numbers used by computers follow a similar pattern, but don't have all the same stages. You may meet some of the same stages in maths courses.

Numbers	Maths name	CS name	Examples
Positive whole numbers	Natural numbers	Unsigned integers	0,1,2,3,...
Positive or negative whole numbers	Integers	Integers	-3,-2,-1,0,1,2,3,...
Fractions	Rationals	no equivalent	$2\frac{3}{4}$
Decimals (scientific notation)	Reals	Floating point	$2.9979 * 10^8$
Complex numbers	Complex numbers	no equivalent	$3 + 4i$

Computers typically do not have all of these supported by hardware and programming languages do not have them as standard types. The ones that are missing are fractions and complex numbers.

Class	Examples	Computer Representation	Java types
unsigned	$0, 1, 2, 3, \ldots$	unsigned binary	
signed	$\ldots -3, -2, -1,$ $0, 1, 2, 3 \ldots$	2's complement	byte, short, int, long
floating point	$-2, 80,\ 1.00,\ 3.14$	IEEE floating point	float, double

10.3 Number bases

We count in tens. There is an obvious reason for this: we have ten digits (digit is latin for finger) on our hands (including thumbs). That is a biological, not a mathematical, explanation.

Thought experiment Suppose we had a different number of fingers and thumbs (say eight, which would be one less on each hand, or twelve, which

would be one more). Would we then count in 8's or 12's (answer: yes), and would this make a real difference to us (answer: no).

Mathematicians have studied this idea under the name of **number bases**.

When we write numbers down we are trained to use positional notation: the position that a digit is in controls how much it is actually worth. A digit one place to the left is worth ten times what it was in its original position. This idea is incredibly important. It is critical to our being able to do calculations with large numbers. And it explains why the invention of 0 is so important. Without 0 we would not be able to see all the positions.

1984

1	9	8	4
10^3	10^2	10^1	10^0
1000	100	10	1
1000	900	80	4

In the number 1984 the digit 8 is second from the right, so it is multiplied by 10, and is worth 80. The digit 9 is third from the right, so it is multiplied by 100 and worth 900.

$$1984 = 1000 + 900 + 80 + 4 = 1 * 10^3 + 9 * 10^2 + 8 * 10^1 + 4 * 10^0$$

We can use the same kind of representation mechanism with any number we like, not just 10. This number is called the **base** of the representation.

In Computer Science, the most important base is 2 (**binary**). But we also use base 16 (**hexadecimal** or **hex**).

10.3.1 Converting from binary to decimal

in binary, the powers of 2 play the same role as the powers of 10 in decimal, and the only digits are 0 and 1. This means that a 1 in the fourth position from the right in a binary number is worth $2^3 = 8$ (not $10^3 = 1000$).

position	...	7	6	5	4	3	2	1	0
worth	...	128	64	32	16	8	4	2	1

Conversion method: binary to decimal Example: to convert 1110 0101 to decimal

Write down the digits in a table:

digits	1	1	1	0	0	1	0	1

Insert the powers of 2:

digits	1	1	1	0	0	1	0	1
powers	128	64	32	16	8	4	2	1

Multiply the columns:

digits	1	1	1	0	0	1	0	1
powers	128	64	32	16	8	4	2	1
worth	128	64	32	0	0	4	0	1

Add the results:

digits	1	1	1	0	0	1	0	1
powers	128	64	32	16	8	4	2	1
worth	128	64	32	0	0	4	0	1
sums	229	101	37	5	5	5	1	1

So 1110 0101_2 (the subscript indicates binary), converted to base 10 is 229 (229_{10}).

10.3.2 Converting from decimal to binary

The process can almost be reversed. Taking the same example -
Write out powers of 2 till you reach the largest possible power of 2 that is contained in the number:

sums		229							
powers	256	128	64	32	16	8	4	2	1

Then subtract that power and find the largest power that is contained in the result:

sums		229	101						
powers	256	128	64	32	16	8	4	2	1
sub		101							

And keep on repeating until you get 0:

sums		229	101	37			5		1
powers	256	128	64	32	16	8	4	2	1
diff		101	37	5			1		0

Then put 1 where you have used that power of 2 and 0 otherwise:

sums		229	101	37			5		1
powers	256	128	64	32	16	8	4	2	1
diff		101	37	5			1		0
binary		1	1	1	0	0	1	0	1

The bottom line gives the binary expansion: 1110 0101_2.

Of course you can do this with larger numbers. For example: 1348_{10}

sums		1348		324		68			4			
powers	2048	1024	512	256	128	64	32	16	8	4	2	1
diff		324		68		4				0		
binary		1	0	1	0	1	0	0	0	1	0	0

So $1348_{10} = 101\ 0100\ 0100_2$.

10.4 Other bases

Let b be any positive number greater than 1, then any positive number n can be written uniquely as:

$$n = a_m b^m + a_{m-1} b^{m-1} + \ldots + a_2 b^2 + a_1 b^1 + a_0 b^0$$

where each a_i is between 0 and $m-1$ ($0 \leq a_i \leq m-1$) and $a_m \neq 0$. b is called the **base** of the representation.

Example: In this example the subscript is used to indicate that the expression is written in base 8.

$$100 = 64 + 32 + 4 = 1 * 8^2 + 4 * 8^1 + 4 * 8^0 = 144_8$$

10.4.1 Converting from base 10 to base b

Method 1: Successive division

Construct the base b representation from the right. To calculate the base b representation of the number n:

```
repeat
   divide n by b and let q be the quotient and r the remainder
   write r as current digit
   move left
   replace n by q

   until n = 0
```

Examples:

1. **Convert** 100_{10} **to base** 8**:**
   $n = 100$, $100/8 = 12$ remainder 4, write 4, move left, $n = 12$: current output 4
   $n = 12$, $12/8 = 1$ remainder 4, write 4, move left, $n = 1$: current output 44
   $n = 1$, $1/8 = 0$ remainder 1, write 1, move left, $n = 0$: current output 144
   $n = 0$, stop, output 144. So $100_{10} = 144_8$, as above.

2. **Convert** 100_{10} **to base** 3**:**
   $n = 100$, $100/3 = 33$ remainder 1, write 1, move left, $n = 33$: current output 1
   $n = 33$, $33/3 = 11$ remainder 0, write 0, move left, $n = 11$: current output 01
   $n = 11$, $11/3 = 3$ remainder 2, write 2, move left, $n = 3$: current output 201
   $n = 3$, $3/3 = 1$ remainder 0, write 0, move left, $n = 1$: current output

0201
$n = 1$, $1/3 = 0$ remainder 1, write 1, move left, $n = 0$: current output 10201
$n = 0$, stop, output 10201. So $100_{10} = 1201_3$.
We can check this: $10201_3 = 1 * 3^4 + 2 * 3^2 + 1 * 3^0 = 1 * 81 + 2 * 9 + 1 * 1 = 81 + 18 + 1 = 100$.

This is not a very efficient way of writing this down, and you might want to use:

0	1	3	11	33	100
	1	0	2	0	1

Answer: $100_{10} = 10201_3$.

Examples:

1. **Convert** 123_{10} **to base** 5**:**

0	4	24	123
	4	4	3

Answer: $123_{10} = 443_5$.

2. **Convert** 246_{10} **to base** 7**:**

0	5	35	246
	5	0	1

Answer: $246_{10} = 501_7$.

10.4.2 Converting from base b to base 10

You could, in principle, use the same algorithm as before. But that would involved dividing a number written in base b by 10_{10}, and this is not something we are practised at. So another way is to invert the algorithm above.

Method 2: Successive multiplication

Decompose the base b representation from the left. To calculate the base 10 representation of a number given in base b:
`proceed from left to right: take the first digit of the number, multiply by` b `and add the second digit, multiply again by` b `and add the third, proceed until the end of the number.`

Examples:

1. **Convert** 144_8 **in base** 8 **to base** 10**:** Start with 1,
multiply by 8 and add 4 to get 12,
multiply by 8 and add 4 to get $96 + 4 = 100$

2. **Convert** 10201_3 **in base** 3 **to base** 10**:** Start with 1,
multiply by 3 and add 0 to get 3,
multiply by 3 and add 2 to get 11
multiply by 3 and add 0 to get 33
multiply by 3 and add 1 to get $99 + 1 = 100$

3. **Convert** 443_5 **in base** 5 **to base** 10**:** Start with 4,
   multiply by 5 and add 4 to get 24,
   multiply by 5 and add 3 to get $120 + 3 = 123$

4. **Convert** 501_7 **in base** 7 **to base** 10**:** Start with 5,
   multiply by 7 and add 0 to get 35,
   multiply by 7 and add 1 to get $245 + 1 = 246$

You could also use the arrays we used previously to write these calculations down:

0	4	24	123
	4	4	3

0	5	35	246
	5	0	1

10.5 Hexadecimal (Hex)

From the Computer Science point of view you need to know about two number systems:

binary: base 2

hexadecimal (or **hex** for short): which is base 16.

It is also useful to at least know about a third: **octal**, which is base 8.
Each binary digit is called a **bit**.

10.5.1 Hexadecimal

Hex digits need to run from 0 to 15, so we use a...f for 10...15:

hex	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Since $16 = 2^4$, each hex digit corresponds exactly to four bits. This means that hex is a compact way of representing bit patterns that is both close to the actual pattern and that people can read.

Example: For example, if you look at a Mac address or a WIFI base station ID, you will see something like:

00:24:a8:b7:e2:60

This is a way of writing down a bit pattern using hex. Each hex digit corresponds to four bits as follows:

hex	0	1	2	3	4	5	6	7
decimal	0	1	2	3	4	5	6	7
binary	0000	0001	0010	0011	0100	0101	0110	0111

hex	8	9	a	b	c	d	e	f
decimal	8	9	10	11	12	13	14	15
binary	1000	1001	1010	1011	1100	1101	1110	1111

The hex expression gives a bit pattern in which each hex digit is replaced by the corresponding four bits:

00	:	24	:	a8	:
0000 0000		0010 0100		1010 1000	
b7	:	e2	:	60	
1011 0111		1110 0010		0110 0000	

Similarly, colours are sometimes represented by hex string. In Word, choose the text colour "dark purple", and then open "More Colors" to get a dialogue box. Choose RGB sliders and you will see that colours have a hex representation. Dark purple is 660066. This represents a 24-bit pattern, with 8 bits for each of the red green and blue components:

6	6	0	0	6	6
0110	0110	0000	0000	0110	0110

Hexadecimal key learning points

- the 15 hexadecimal digits are $0 \ldots 9$ and $a \ldots f$
- because each hex digit represents four bits, hexadecimal is mainly used as a compact human-readable way of writing down binary
- there is a very simple digit by digit conversion algorithm for hex to binary conversion as above
- it can easily be reversed to give a binary to hex conversion algorithm

Self Test Exercises:

☐ 1. Convert the following from hex to binary:

☐ *a*) a9b

☐ *b*) c08

- ☐ *c*) 174
- ☐ *d*) 1100 1101

☐ 2. Convert from binary to hex:

- ☐ *a*) 1110 1001
- ☐ *b*) 0101 1101 0000 1100

Binary Binary is often used as a number representation. Where our standard decimal representation uses powers of ten, binary uses powers of two. You should know and recognise the first of these:

n	0	1	2	3	4	5	6	7	8	9	10
2^n	1	2	4	8	16	32	64	128	256	512	1024

Notice that $2^{10} = 1024$ is very close to 1000 (it is 2.4% out, which is a good approximation).

n	10	20	30	40
2^n	1024	1048576	1073741824	1099511627776

Similarly 2^{20} is close to a million, and 2^{30} is close to a billion, though the errors are increasing (nearly 10% for 2^{40}). Nevertheless this means that decimal prefixes (kilo, mega, giga, and tera) are also used for binary multiples, though incorrectly. There are strict binary analogues: kibi, mebi, gibi, tebi, see `https://en.wikipedia.org/wiki/Binary_prefix` for details.

In decimal, using 3 digits we can write down the numbers from 0 to $999 = 1000 - 1$. Using four it is 0 to $9999 = 10000 - 1$, and using n it is 0 to $99\ldots9 = 10^n - 1$. Similarly in binary, using 3 bits we can write down the numbers 0 to $7 = 8 - 1$, using four bits the numbers 0 to $15 = 16 - 1$, using eight bits the numbers 0 to $255 = 256 - 1$, and using n bits we can write down numbers 0 to $2^n - 1$. This means that with n bits we can produce 2^n different bit-strings to act as labels, and so label 2^n different objects.

Examples:

1. There are 26 characters in the standard English alphabet, each of which can be either upper or lower case (52). There are also 10 digits $0\ldots9$, and perhaps 20 other common characters used in text and for punctuation `".,;:$ ()[]&..."`. This makes a total of something like $52+10+20 = 82$ separate characters. 7 bits would give us a total of 128 bits strings, so these characters can be encoded using 7 bits.

2. The standard IPv4 address is (effectively) a 32-bit bit sequence. This means that there are $2^{32} = 4,294,967,296$ possible IP addresses. Since there are about 8 billion people in the world, only one person in every two can have their own individual IPv4 address.

Logarithms

Logarithms are the inverse operation to exponentiation:

$$m = b^n \iff n = \log_b m$$

in particular:

$$m = 2^n \iff n = \log_2 m$$

For example

$$\begin{array}{rcr} 128 = 2^7 & \text{so} & 7 = log_2 128 \\ 4,294,967,296 = 2^{32} & \text{so} & 32 = log_2(4,294,967,296) \end{array}$$

If you have m objects, then $\log_2 m$ is the number of bits you need to give each of them an individual representation.

10.6 Long addition

Back in primary school you learned how to do long addition.

First you learned your basic addition tables: how to add small numbers together (where here small basically means less than 10, so single digits). Long addition lets you add larger numbers together. Viewed from an algorithmic perspective it is a method for extending single digit addition to the addition of larger numbers represented using place notation. So it works for numbers written in any base. In particular it works for binary.
Recap: $482 + 364 = 846$:

4	8	2		
3	6	4	+	
8	4	6		
1			carries	

The same method works for binary:

1	0	1	0	1	0	1	0	
		1	1	0	0	1	1	+
1	1	0	1	1	1	0	1	
	1				1			carries

A key difference is that the addition table you need is much smaller (two digits given so that carries are explicit):

+	0	1
0	00	01
1	01	10

as opposed to:

+	0	1	2	3	4	5	6	7	8	9
0	00	01	02	03	04	05	06	07	08	09
1	01	02	03	04	05	06	07	08	09	10
2	02	03	04	05	06	07	08	09	10	11
3	03	04	05	06	07	08	09	10	11	12
4	04	05	06	07	08	09	10	11	12	13
5	05	06	07	08	09	10	11	12	13	14
6	06	07	08	09	10	11	12	13	14	15
7	07	08	09	10	11	12	13	14	15	16
8	08	09	10	11	12	13	14	15	16	17
9	09	10	11	12	13	14	15	16	17	18

This makes the implementation on the computer simpler.

10.7 Long multiplication

The same holds true for long multiplication. The same algorithm works, with a simplification in binary.

Once again, you should have learned the method for decimal in primary school:

		2	5	6	
			2	1	*
		2	5	6	
	5	1	2	0	
	5	3	7	6	

The method for binary is similar:

				1	0	1	0	1	0	1	0	
								1	1	0	1	*
				1	0	1	0	1	0	1	0	
		1	0	1	0	1	0	1	0	0	0	
	1	0	1	0	1	0	1	0	0	0	0	
1	0	0	0	1	0	1	0	0	0	1	0	
1	1	1	1	1	1	1	1					carries

Notice that we only ever multiply by:

0: we can simply ignore this

1: from the point of view of the bit pattern, this is just a shift.

So long multiplication in binary can be carried out using the operations of shift left and addition. We do not need to learn a multiplication table.

The example above was fairly simple because we never had a number larger than 1 to carry. But, that can often happen:

12	11	10	9	8	7	6	5	4	3	2	1	columns
				1	1	1	0	1	0	1	1	
								1	1	1	1	*
				1	1	1	0	1	0	1	1	
			1	1	1	0	1	0	1	1	0	
		1	1	1	0	1	0	1	1	0	0	
	1	1	1	0	1	0	1	1	0	0	0	
1	1	0	1	1	1	0	0	0	1	0	1	
		1		1		1	0		1			carries
		1	0			1		1				carries
1	0			1	0							carries

In this example,
The first column contains a single 1, so the sum is 1 carry 0.
The second column contains two 1's and no carries, so the result is 10, written as 0 carry 1. That 1 is in the first row of carries.
The third column contains two 1's and a carry 1, so the result is 11, written as 1 carry 1. That carry 1 is in the fourth column in the second row of carries.
The fourth column contains three 1's and a carry 1, so the result is 100, written as 0 carry 10. That carry is written in the first row of carries, with 0 in the fifth column and 1 in the sixth.
The fifth column contains two 1's and a carry 0, so the result is 10, written as 0 carry 1, with the carry 1 in the sixth column of the second row of carries.
The sixth column contains two 1's and two carry 1's, so the result is 100, written as 0 carry 10, with the carry now written in the seventh and eighth columns of a third row (so that we can distinguish where we are getting carries from). 1 in the sixth column of the second row of carries.
The seventh column contains three 1's and a carry 0, so the result is 11, written as 1 carry 1, with the carry 1 in the eighth column of the first row of carries.
The eighth column contains three 1's and two carry 1's, so the result is 101, written as 1 carry 10, with the carry 10 in the second row of carries.
The ninth column contains three 1's and a carry 0, so the result is 11, written as 1 carry 1, with the carry 1 in the first row of carries.
The tenth column contains two 1's and two carry 1's, so the result is 100, written as 0 carry 10, with the carry 10 in the third row of carries.
The eleventh column contains a single 1 and a carry 0, so the result is 1.
Finally the twelfth column contains no 1's and a single carry 1, so the result is 1.

10.8 Self-test solutions

Self Test Exercises:

☐ 1. Convert the following from hex to binary:

- ☐ *a*) a9b: 1010 1001 1011: straightforward using table
- ☐ *b*) c08: 1100 0000 1000: note the zeroes in the middle
- ☐ *c*) 174: 0001 0111 0100: these are bit patterns so leading zeroes are included
- ☐ *d*) 1100 1101: if you had cd, then you were deceived into converting this from binary to hex. The correct solution is: 0001 0001 0000 0000 0001 0001 0000 0001

☐ 2. Convert from binary to hex:

- ☐ *a*) 1110 1001 : e9: this is conveniently split into groups of four bits, so just look them up in the table.
- ☐ *b*) 0101 1101 0000 1100: 5b0a