Mechanical Aids to Computation and the Development of Algorithms

Summary: Mechanical aids to computation developed out of a need to reduce the effort involved in lengthy and tedious mathematical calculations such as those arising in making astronomical predictions, navigation, bookkeeping, etc. Linked with the the idea of such devices is the concept of an algorithm — a precise description of the steps to be carried out in order to calculate some quantity. Early calculating devices could only be applied to very specific tasks, i.e. these could not be programmed. In the 20th century the first general-purpose computing devices were built. The effect of these has been to extend the range of fields in which automatic computation can be applied, e.g. in areas such as Artificial Intelligence, Databases etc.

Syllabus

1. Present concerns in Computer Science: popular views of computers; common applications and their effects. Scientific concerns: Artificial Intelligence, theory of computation, non-Von Neumann machines
2. Introduction; Counting systems to represent numbers: Babylonian, Mayan, Roman numerals, Arabic/Indian. Early calculating devices: clay tablet (Babylonia, ca. 2000 B.C); abacus (Babylonian, before 500 B.C). Concept of algorithm (ruler and compass constructions, Euclid; Persian work)
3. 16-18th Century developments: Galileo's geometric compass (1597); Napier's and Briggs' development of Logarithms (1614); Oughtred's Slide rule (1621); Mechanical calculators of Pascal (1644) and Leibniz (1673). Jacquard's invention of punched cards (1753)
4. 19th Century: Babbage: Development of Difference Engine (1820-1830s); Analytical Engine first attempt at general-purpose programmable computer (1840s); Boole - formulation of logical algebra underpinning operation of digital computers (1880s); Hollerith's machine for tabulating census statistics (1887).
5. Development of first digital computers: 1920-1948: Stibitz, Eckert, Mauchly, Atanasoff, von Neumann (US); Zuse (Germany); Turing, Wilkes (UK).
6. 1950-1960s: Spread of large-scale mainframe computers: contributions of IBM, ICL etc.; First (high-level) programming languages; FORTRAN, COBOL, ALGOL 60.

Mechanical Aids to Computation and the Development of Algorithms

If this is the best of all possible worlds, what can the rest be like?

Voltaire
Candide

1. Introduction - Current Concerns in Computer Science

One of the more hackneyed clichés touted around at present is that "nowadays, computers are everywhere". It is widely believed that computers influence significant aspects of our lives; computers are seen, by some, as a social threat; by others, as a panacea for social problems; and, frequently, computers are accepted as a valid scapegoat for human incompetence. Yet, as little as 30 years ago, such views would not have been found outside the pages of science-fiction stories. It is clearly the case that, in order for these perceptions to have gained ground, a tremendous increase in the public awareness of computers must have occurred. In this, introductory, lecture we shall examine what the principal current concerns in computer science are and discuss how these have contributed to popular misconceptions about the rôle and functionality of computational systems. In particular we will address the following set of questions:

1. Why are the attitudes, summarised in the opening paragraph, misrepresentations of the use and potential of computers?
2. What has happened, in the recent development of computational aids, to cause a widespread acceptance of such views and what are the actual areas on which the present state of computer technology has an impact?
3. What, in practise, are the present concerns of Computer Science as a scientific discipline?

1.1. Misconceptions concerning computational systems

The views described above can be reduced to two assumptions:

A. Computers can do anything.

B. Computers are ‘responsible’ for their behaviour.

(A) accounts for the perception of computer potential as threatening or beneficial; (B) for the proffering of computers as causes of errors.

In order to refute (B) it is enough to observe that a computer is, merely, a *tool* that is used under *human* control to specific ends. To reason that the quadrupling of a pensioner’s electricity bill was caused as a result of ‘the computer making a mistake’ is no more logical than propounding ‘the engine made a mistake’ as a defence to a motoring offence. Thus, the only individuals who are ‘responsible’ for the behaviour of machines are their users and designers.[1]

The assertion ‘computers can do anything’ is rather more difficult to deal with. As a precisely formulated mathematical abstraction, however, it has been known to be false since

1) There are numerous examples of the (often wilful) failure to understand this obvious fact: the excuses made by television companies who failed to come close to predicting the outcome of the last two General Elections; the near nuclear accident at 3-Mile Island; the actual nuclear disaster at Chernobyl.

1936.[2]

It is the case, however, that this technical refutation of the claim ‘computers can do anything’ is not universally accepted: Turing’s premises are questioned, even by some computer scientists. Nevertheless, underlying Turing’s proof is the appreciation that all any computational system does is to manipulate finite sequences of symbols, such manipulations being carried out in accordance with a given *program*[3] of instructions. Given this, rather stark, description of the capabilities of computational mechanisms, it should seem immediately apparent that the claim ‘computers can do anything’ is fallacious. There are those who would dispute this, however, and we will return to this issue at the end of the lecture.

1.2. Cultural and social causes of misconceptions concerning computers

One of the most noticeable aspects of how important scientific developments become widely known is in their (frequently misleading) portrayal in popular culture. As examples one may cite the rôles played by: electrical power in Mary Shelley’s *Frankenstein*; new psychological theories of personality in Stevenson’s *Dr. Jekyll and Mr. Hyde*; the exploitation of Freud’s psychoanalytic theories by Surrealist art; the effects of nuclear radiation and atomic power in films such as *Them!* and *The Incredible Shrinking Man*; etc. Progress in Computer Science has been no exception to this phenomenon, even though much of its history is concerned with the development of artefacts rather than abstractions. Of course the association of anthropomorphic attributes with machines and the view of these as threatening forces are very old ideas, cf. the Luddite riots in the 18th century. In this century the same theme may be found in works ranging from early instances such as Duchamp’s great sculpture *La mariée mise à nu par ses célibitaires, même (Le grand verre)*[4] through musical compositions like Varèse’ *Déserts* to its specific computer representations in such films as Russell’s *Billion Dollar Brain*[5], Tarkovsky’s *Solaris*, Kubrick’s *2001*, etc. Thus in the specific case of computer development the popular dissemination of the capabilities of these machines was, from the 1950s onward through the media of cinematic fiction and less than accurate newsreel and journalistic coverage. This, of course, during a period (post-war to mid 1960s) when a general appreciation of new technology and ideas would largely be acquired through cinemas rather than through television.

Despite the fact that the image of machines as threatening/beneficial has long been fashionable it is not one that can be seriously sustained without some underlying foundation — no matter how weak such a foundation may be in reality, e.g. the Luddite reaction to new technology, mirrors the real threat that individuals felt concerning their employment and welfare. If we examine what this foundation is in the case of progress in computer science then some understanding of how it has been distorted may be gained.

When we observe an individual performing, with great proficiency, a task that is regarded as intellectually demanding we have a natural tendency to view such a person as possessing above average ‘intelligence’. This is especially the case when the activity at which

2) *vide* ‘On computable numbers with an application to the *Entscheidungsproblem*’ (*Proc. London Mathl. Soc.*, Series 2 (42), pp. 230-265 (1936); corrections: *ibid*, (43) pp. 544-546 (1937)). An achievement of the great English mathematician and computer pioneer Alan M. Turing (1913-54) whose contribution will be discussed later in this course.

3) The U.S. spelling ‘program’ (as opposed to the British ‘programme’) has become a standard usage when the word is used in the particular sense implied here.

4) Although gynomorphic would be a more appropriate description of this.

5) A particularly laughable misrepresentation of computers: the eponymous machine is capable of analysing material of enormous complexity in order to make forecasts concerning military strategy; it can produce output in graphical, spoken and printed form, but throughout the film information is supplied to it in the form of paper tape and punched cards, techniques that had already been improved upon by the time the film was made.

they excel is one which we, ourselves, find beyond our abilities. Thus, a gifted writer or artist, a successful entrepreneur, a prominent scientist or mathematician, a persuasive orator, or a skilled chess player, may often be regarded as people of 'great intelligence', despite whatever evidence they may exhibit to the contrary. This tendency accounts in part for certain misconceptions that are widely held about computers and, unfortunately, is one which too many computer scientists have encouraged and too few have tried to correct.

Traditional applications of computers have included numerical calculations, deciphering codes, storage and processing of recorded information, and recently decision-support and advice giving. Consider the first of these. Involved arithmetic calculation is an activity that many, if not most, people find difficult to carry out, particularly if it is to be done solely in one's head. Computers, however, can be programmed to perform such calculations extremely quickly and in a manner which always gives the correct answer. Similarly deciphering encrypted text (the process of translating a piece of text given in an encoded form back to its original plain text format) is something which requires a skilled human agent but which can be accomplished, fairly easily, by appropriate computer systems.[6] Now both of these activities — arithmetic calculation and code decryption — are ones which would be regarded as indicating 'intelligence' in an individual who performs them 'well'. Hence we have one source of fallacies concerning computational mechanisms: since computers can (be programmed to) perform extremely well tasks that are normally found difficult, e.g. arithmetic, and since a human with some competence in such tasks would be considered 'intelligent', it follows that computers are 'intelligent' and therefore could solve other problems which we find hard to deal with. This is a rather casual logic but one which was (is, even) widely accepted.[7].

One might argue, however, that there are many examples of mechanical devices that perform tasks well beyond the ability of a human agent — e.g. compact disc players, television sets, etc — and no (sane) individual would regard such as 'intelligent'; why, then, should computers be considered differently? Such an argument is, of course, perfectly sound, there is no reason: the misconception that computers are capable of 'intelligent' action arises not only from their facility at tasks such as arithmetic but also from the fact that they can be configured (i.e. *programmed*) to carry out other, unrelated, tasks as well. In addition a computer executing a program appears, in some sense, to be independent of human control.[8]

The reasoning "computers can perform well some tasks that require 'intelligence' and thus may be able to carry out other functions needing a similar ability" was one of the historical motivations behind a study that, at present, forms one of the principal fields in Computer Science: the area of Artificial Intelligence (A.I.). We shall return to this topic when discussing current issues in Computer Science in the concluding section of these notes. For the moment, it should be noted that the misrepresentation and exaggerated claims concerning the potential of A.I. systems has been, to a significant degree, a factor in the view of computers as threatening/beneficial objects.

Aside from the distorted perception of A.I., it is, undoubtedly, the application of computer systems as record maintenance tools that has contributed most to public disquiet about computers. Here is a selection of areas in which personal records are stored, processed and

6) One of the first British computers — COLOSSUS — was constructed during the Second World War to assist with precisely this task of decoding intercepted messages.

7) For which fact newsreel coverage of early computer systems bears some responsibility: a famous newsreel of the Whirlwind computer, built at M.I.T in the late 1940s, involves an over-excited reporter awe-stricken by the machine's 10 second calculation of a table of square roots and enthusing over the potential other uses of this machine.

8) This (illusory) appearance of detachment is used to considerable effect in at least three films: *2001*, *The Forbin Project*, and the execrable *Demon Seed*.

maintained on computer *databases*: Personal taxation records (by the Inland Revenue); student exam marks, course registrations, and other details (by the University of Liverpool, among others); Vehicle Licencing records (by the D.V.L.C. at Swansea); Social Security information (by the D.S.S. at Newcastle); car theft and motoring offences (by the Police National Computer); credit status and personal financial records (by banks, building societies, credit card agencies); employee salary details and other personal information (by most large employers).

It can be seen that data processing and handling is one of the areas of computer activity which affects almost everybody. The last 20-25 years have seen, in this sphere, an enormous increase in the number of applications which are dealt with by computer-based record systems rather than by the more traditional paper filing techniques. This expansion in the amount of confidential and personal information held on computer systems has been viewed with trepidation by many individuals, largely on account of the following reasons:

i. Civil liberties groups feel concerned about the uses to which personal details of individual citizens are put, and have expressed fears about the extent of (and reasons for holding) information about certain categories of people. Examples cited as grounds for such concern include: the use of employment blacklists by certain companies; details about members of political parties/pressure groups etc; the fact that the storage capacity of the Police National Computer database considerably exceeds that which would be needed to record all instances of car thefts.

ii. Clearly, since a lot of information is confidential, appropriate safeguards must be adopted to ensure that only authorised individuals have access to this. A number of database systems have failed to address this requirement.

iii. Finally, there is concern about detecting and amending errors in stored records, e.g. if a credit card company has mistakenly identified an individual as a bad credit risk then this can have serious consequences for the person affected: they may find themselves unable to obtain a bank loan or mortgage, or face the embarrassment of having their credit card confiscated when they attempt to use it.

Some indication of the seriousness with which the last two points have been treated may be discerned in actions taken by Parliament within the last 10 years. In 1984 the Data Protection Act became law. This created the post of Data Protection Registrar by whom all instances of computer recording of personal information had to approved[9]. The Act also gave individuals the power to inspect computer record systems where they had reason to believe information relating to them was held[10] and to have such information corrected if it was in error. The other legislative action occurred in 1990 when The Computer Misuse Act, a Private Member's bill sponsored by the Conservative M.P., Emma Nicholson, became law. This criminalised the activity of obtaining unauthorised access to computer systems — so called 'hacking'. Regrettably its success has been rather mixed: the only prosecutions brought to date have resulted in one highly publicised acquittal and custodial sentences for two individuals who pleaded guilty at trial.

In summary, we can see that two of the main causes of the public awareness of computer development have been the popular reporting of A.I. potential — thus the fallacious extrapolation from facility at arithmetic to capability for intelligent independent action — and

9) This provision did not, in fact, apply universally: certain classes of system were exempted, specifically those dealing with 'national security' issues.

10) The reason why the University of Liverpool now informs students of the actual marks they obtained on exam papers is in order to comply with the provisions of the Data Protection Act (although the Act only requires that such marks be released *on request*; voluntary disclosure is not, in fact, a legal requirement). Ten years ago such action would have been considerable unthinkable in many quarters.

the growth of computer information storage systems with the attendant problems these create. We conclude this sub-section by, briefly, examining a few other areas in which computer technology has become influential and the effect of these developments on the general perception of computers.

Some applications which have noticeably been affected by technological developments are the areas of: process control; graphical displays; and transaction systems.

Process control concerns the use of computer systems to regulate activities such as the running of processes ranging from chemical production plants and nuclear power stations to washing machines and video recorders. In these the fact that computer mechanisms are being applied is often not obvious, part of the reason for this is the increased sophistication of the devices that carry out the control actions so that the physical space occupied by the controller is very small.

Similar technological progress accounts for the advances and wider applications of computer graphics. Unfortunately much of the exploitation of this capability has been in somewhat trivial areas: special effects in films, arcade and computer games. The approbation the latter have attracted is an interesting sociological phenomenon, *viz.* reports about children addicted to such games, claims that these may trigger epileptic fits or cause psychological illnesses; assertions that they encourage sociopathic behaviour; and the belief that these games deprive children of time that could be spent in more ‘worthwhile’ pursuits[11].

Finally, transaction handling via computer systems has increased greatly since the early 1970s. Common examples of such systems are: automatic cash dispensers used by banks and building societies; airline booking and other seat reservation applications; library information services, e.g. the LIBIS system used at Liverpool University. In the first area — cash dispensers — some concerns have been expressed about security, specifically the occurrence of ‘phantom withdrawals’ from accounts. At present there has been no resolution of the dispute between banks (who maintain that all questioned transactions occur because customers have released their identification code to a third party) and consumer organisations (who claim that unauthorised withdrawals from accounts are possible without such a code being disclosed). It seems probable, however, that this argument will be resolved in favour of the consumer interests.

1.3. Current Scientific Concerns in Computer Science

In the preceding sub-section we discussed those developments in computer technology, over the last quarter of a century, that have had a significant impact on the public awareness of computers. We conclude this opening section by examining a few of the research areas which are currently of importance in the development of Computer Science as a scientific discipline. The topics discussed, briefly, below are not intended to provide an exhaustive survey, but merely to give an overview of some areas which are being pursued.

We shall concentrate on three general areas in which there have been significant interest over the last few years:

i. Artificial Intelligence

ii. Non-traditional approaches to computation.

iii. Mathematical Theory of Computation.

11) The last is a recurring objection to new entertainment media: contemporary sources may be found inveighing against comics and cheap thrillers, gramophone records, cinema, radio, television, etc. Undoubtedly the 15th century invention of the printing press attracted similar approbation from those claiming to be solicitous of children’s well-being.

1.3.1. Artificial Intelligence (A.I.)

We have already mentioned this area in connection with misconceptions about computer potential. There are a number of competing factions working in this field who disagree over what the exact aims of A.I. should be. It is to be regretted that the strength of this disagreement is such that opposing camps often comment on ideas whose validity they dispute, in a style which might be considered inappropriate to the conduct of a rational, scientific discourse. The two predominant opinions may be summarised as:

> *"The aim of A.I. should be to understand the processes of thought and intelligence as computational phenomena."*
>
> *"The aim of A.I. should be to build systems that can perform some activities at least as well as human agents."*

The latter approach has, without question, been the more successful. One of its most important contributions has been the development of, what are called, *Expert Systems*. Such systems are attempts to replicate the advisory and decision-making processes used by experts in specialist domains of knowledge. Fields in which successful systems have been built include: medical diagnosis (for particular classes of ailment); legal systems (for interpreting specific items of legislation); and mineral prospecting. Research into extending the range and developing the capabilities of such systems is, at present, one of the central areas of computer science.

The alternative school has concentrated on areas such as understanding natural language, and philosophical debate about the nature of what constitutes an ‘intelligent machine’. In recent years there has been a growing interaction between this approach and work carried out in cognitive psychology, neurophysiology, and linguistics. In this respect some interesting work is being performed. Arguably, the both approaches are still suffering the consequences of inflated claims made in the 1960s and 1970s about what could be delivered.[12]

1.3.2. Non-traditional approaches to computation

Non-traditional, or more properly *non-von Neumann*[13] computer models, have become established as a core research area in computer science in the last 10 years. Four sub-topics within this field are currently the subject of much attention:

A. Neural networks

B. Genetic Algorithms

C. Quantum Computers

D. Parallel Computers

A. Neural Networks

The theory behind these developed from biological models of the working of the human brain. Such networks have been successfully applied to deal with categories of pattern recognition problems, e.g. voice recognition. One of the computationally interesting aspects of the neural network approach is the fact that neural networks are customised by being ‘trained’ with examples of the objects to be recognised. Thus the internal characteristics of the

12) There are numerous stories about failed A.I. systems, e.g. in the 1970s the U.S. Intelligence agencies funded a project for determining geo-political strategies; on being asked to draw a conclusion from the three statements: ‘Russia is a communist dictatorship’; ‘Russia is hostile to the U.S.A.’ and ‘Cuba is a communist dictatorship’, the system responded ‘Cuba is in Russia’.

13) After the Hungarian born, U.S. computer scientist John von Neumann (1903-57) who is credited with the first formal description of the structure of general-purpose computers.

network are modified, sometimes by the network itself, until a satisfactory performance level is attained. Investigation of learning and training approaches forms an important facet of work on neural networks. In addition advances in the physical construction of computer systems have created the possibility of building very large neural networks inexpensively.

B. Genetic algorithms

Genetic algorithms provide another example of ideas from biology being translated into computational analogues. The idea underlying these is to emulate the evolution of DNA sequences as a means of solving computational problems. Thus the processes by which DNA strings combine and evolve are mirrored by symbol manipulation operations on sequences of symbols. Some promising approaches have been developed for some difficult optimisation problems. Research in this area, at present, is concerned with developing the theoretical basis of these methods (which is currently not well understood) and with extending its applicability.

C. Quantum Computation

Whereas neural networks and genetic algorithms have come about through importing ideas from biology, quantum computation has its origins in developments in physics — specifically exploiting quantum level effects as a computational device. The theoretical properties of quantum computers were first described by the physicist David Deutsch[14] by whom the possibility that these may be considerably faster than classical machines was raised. At present much of the current research on this model is of a highly theoretical nature (hindered by the fact that a number of publications mis-report Deutsch's conclusions). There is, however, some work in progress concerning the feasibility of constructing quantum computers.

D. Parallel Computers

The methods described above may be seen as particular special cases of a more general class of non-von Neumann computers: parallel computers. In these a (potentially very large) number of individual computers are 'linked' together in order to provide a more powerful machine. There are many significant research problems being addressed in this field at present: developing parallel machines for a specific applications; designing methodologies for programming on parallel computers; analysing the efficiency of parallel solutions; etc. This is an area that has only really begun to develop within the last ten years and is likely to pose important questions in Computer Science for a number of years to come.

1.3.3. Mathematical Theory of Computation

This group of subjects forms one of the oldest traditional research concerns in Computer Science. Its origin (arguably in 1900[15]) predates the appearance of the first modern computer systems by almost fifty years. The general aim of this field is to address questions in the design of computer systems and programs from a formal mathematical perspective. Within this area fall three studies of principal interest:

14) In, Deutsch, D: 'Quantum Theory, the Church-Turing principle and the universal quantum computer; *Proc. Royal Soc. of London, Series A*, (400), pp. 97-117 (1985). The concluding section of this paper gives a novel, if somewhat technical, discussion of the relationship between computer science and modern theoretical physics.

15) I have chosen this date from the presentation of David Hilbert's lecture '*Mathematical Problems*', given during the International Congress of Mathematicians at Paris in 1900, the text of which was published in *Archiv der Mathematik und Physik*, (1), pp. 44-63, 213-237 (1901). Hilbert, in this lecture, presented a collection of 23 open problems in mathematics, the second of which — *Die Entscheidungsproblem* — poses the challenge of constructing a specific algorithm (i.e. program).

A. Semantics of programming languages.

B. Computability Theory.

C. Computational Complexity Theory.

A. Semantics of programming languages

Informally this area is concerned with how to attach precise ‘meanings’ to constructs that occur in computer programs. The motivation behind this is twofold: if the actual behaviour of a program can be defined in a rigorous enough manner then the problem of showing that the program fulfills a particular function reduces to that of mathematically proving that the *semantics* of the program accord with the intended functionality. This objective is the goal of the Formal Verification theory. Formal verification was first mooted in the early 1970s by, among others, the Dutch mathematician Dijskstra, the English computer scientist Hoare, and the U.S. computer scientist Floyd. The other motivation for the theory of program semantics is that if precise meanings can be attached to programs then it becomes possible to *specify* formally the functionality a program should achieve. Formal specification concerns the construction of methods by which the requirements of a system can be described precisely and unambiguously. While some success has been achieved with the design of specification systems (most notably at Manchester and Oxford) the existing formal verification techniques have yet to achieve any great level of sophistication. Programming semantics is one of the internationally recognised areas of expertise in British computer science.

Computability Theory

Computability theory is concerned with the classification of which problems can and cannot be solved by computer programs. The first significant work in this field dates back to the 1930’s when concepts of what constitutes a valid computational system were proposed by Emil Post, Alan Turing, Stephen Kleene, and many others. There has been an increased revival of work in this field, partly as a consequence of the developments in non-von Neumann models discussed earlier. Some of these models challenge the premises which are used in the proofs that certain problems cannot be solved. It is, currently, unclear whether the models with the strongest claims in this area would actually be constructible in practice.

Computational Complexity Theory

While computability theory is concerned with the question of which problems can be solved by computers, computational complexity is concerned with which problems can be solved *efficiently*. The term ‘complexity theory’ was first coined in the mid-1960s by Hennie and Stearns, but related work in this field has been in progress since the 1930s, specifically the work of Shannon, Shestakov, and Lupanov concerning building efficient ‘hardware’. Several of the most important open questions in Computer Science and mathematics are ones that have been raised as a consequence of work in this area, the most widely studied of which has been the $P = ?\, NP$ question first formulated by Steven Cook in 1973. Briefly this asks whether a specific class of ‘decision problems’ can be solved by ‘efficient’ programs; a positive answer (which is not expected by most experts in the field) would have considerable implications for a very large number of applications areas. This field is another area of Computer Science in which Britain has a considerable international reputation.

1.4. Summary

Computer technology has developed to a considerable degree over the last 20 years. This has resulted in certain applications of computers impinging on various aspects of individual’s day-to-day lives. As a scientific concern research interests in Computer Science have led to an interaction between older scientific disciplines such as Biology, Psychology, and Physics. Unfortunately, it is still the case that the advent of increased computer application has met with either public concern and suspicion or with over-optimistic beliefs about what computers can do.

Mechanical Aids to Computation and the Development of Algorithms

2. Introduction - Early History

Although the computer and its widespread application in our society, are phenomena that have become predominant in only the last 30 years, many of the concepts underlying these developments have their origins in concerns dating back to the earliest cultures. Computers manipulate *data* (Latin, plural of *datum*, neut. p.p. *dare*, cognate Sanskrit, *datta*: those things which have been given), i.e. process and transform given representations of information in order to obtain a desired result. Within this basic description of computer behaviour we can discern two fundamental ideas:

1. *Representation:* A concrete, symbolic encoding of information, e.g. numbers, words, names.
2. *Transformation:* The steps (recipe, *program*, *algorithm*) used to calculate a specific result.

A symbolic encoding of information provides a vehicle for communication — information can be passed on in a commonly understood form. A record of the process by which the representation is transformed allows the calculation process to be carried out repeatedly on different sets of data, e.g. we have all learned the steps needed to determine the result of multiplying any two large numbers.

One might ask, however, why, given a system for encoding information and the sequence of steps needed to manipulate this to a specific end, it should be necessary to seek *mechanical* assistance with the task? The answer to this question lies in the fact that the calculations required to carry out these tasks are often laborious. This fact has two consequences if no mechanical aid is employed:

1. The computation will take a long time to complete.
2. The answers may be incorrect, because of human error, and so the *same* computation may have to be carried out several times.

(Consider, which would you prefer: to multiply two 5 figure numbers by hand on paper or to use an electronic calculator? Which answer would you have greater confidence in?)

The calculations involved in predicting celestial phenomena from previously observed data; in assessing the rates of taxation to levy in order to raise a required sum; in analysing census statistics; in determining the path of a projectile; all of these are examples where lengthy, tedious and (if done by hand) error-prone computations arise.

Thus the historical development of the topic we are considering can be seen to be based on three related processes:

1. The development of symbolic representations of information *that are amenable* to manipulation.
2. The formulation of algorithms by which such representations may be processed to solve computational problems.
3. The construction of mechanical aids that allow such algorithms to be implemented in an efficient and a reliable manner.

It ought to be clear that methods for representing information must been have developed first, so it is appropriate to examine one aspect of such representations — number systems —

before going on to consider the algorithms and mechanical aids that utilise them.

The simplest method of representing numbers is to use a sequence of identical marks to denote quantities, e.g. Figure 1 below

Modern form	Tally System
1	|
2	||
3	|||
4	||||
5	---𝍸
10	---𝍸-𝍸
25	---𝍸-𝍸-𝍸-𝍸-𝍸

Figure 1: Simple Counting System

Archaeological discoveries have established that the ‘*tally system*’ was independently developed by many early cultures, e.g. bones with notches denoting quantities have been found by anthropologists in Czechoslovakia, similar fragments dating from around 8500 B.C. have also been discovered in Africa. Despite its rudimentary nature, this system does exhibit important features that were to be preserved in later systems. The most important of these is the concept of counting in multiples of some *basic* number (5 in the example above). While most societies adopted 5 or 10 as the typical base (from the practice of counting on fingers) these were by no means universally chosen. The South American Mayan culture employed a system based around the number 360 (from their estimate of the number of days in a year); the Babylonians used 60 as a base. It is interesting to note that the influence of the Mayan and Babylonian systems continues in the present day (a circle contains 360 degrees, a degree 60 subparts called minutes, a minute 60 subparts called seconds; similarly we denote time units in multiples of 60 — 1 hour is 60 minutes is 60×60 seconds)

Although the tally system has several advantages — it is easy to understand, simple arithmetic operations such as addition, subtraction and multiplication can be performed without great difficulty — it is extremely cumbersome when used to represent large numbers, such as might arise in recording population sizes, and it is not suitable for more complicated computational tasks, e.g. division. Attempts to address the first problem can be discerned in the notational systems used by Greek and Roman societies (from ca. 1000 B.C for Greek, 700 B.C, Roman). These, though, were really only slightly more sophisticated versions of the tally system: instead of using a single denotational symbol to construct numbers, a set of different symbols is employed, each representing a different quantity, Figure 2.

Modern	Greek	Roman Numeral	Modern	Greek	Roman Numeral
1	α'	**I**	25	α''	**XXV**
2	β'	**II**	50	β'''	**L**
3	γ'	**III**	99	γ''''	**XCIX** (*not* **IC**!)
4	δ'	**IV**	100	δ''''	**C**
5	ε'	**V**	500	-	**D**
6	ζ'	**VI**	900	-	**CM**
9	ι'	**IX**	1000	-	**M**
10	λ'	**X**	1948	-	**MCMXLVIII**
20	υ'	**XX**	10000	$\mu\upsilon\rho\iota o\iota$[1]	$\underline{\bar{\mathbf{M}}}$
24	ω'	**XXIV**	100000	-	$\overline{\bar{\mathbf{M}}}$

Figure 2: Greek and Roman Counting Systems

Notice that both of these systems eventually become no better (and in some respects considerably worse) than the simple tally system. The method of Roman numerals does, however, introduce an important idea, although failing to exploit it fully: the concept of *positional notation*. Thus in the representation of 1948 the opening **M** has a different meaning from the **M** occurring as the third character. Both of these systems indicate the importance of the qualification '*amenable to manipulation*' that we stressed earlier. The major deficiency of the Roman system is that it is decidedly unsuitable as a basis for computation. In the tally system the calculation of $494+506$ is a (notationally) lengthy but very easy computation; the computation of **CDXCIV**+**DVI** in Roman numerals is not (the answer, **M**, bears no symbolic relation to the summands and although the result is almost twice as large as the individual contributions, it is expressed using one symbol instead of nine). A task such as multiplication, conceivable in the tally system, present major difficulties in Roman numerals. So inflexible is this system that a suitable topic for doctoral research in the early European universities concerned algorithms for multiplication and division using Roman numerals. Despite all of these drawbacks, Roman numerals remained the predominant means of representing quantities in European culture well into the 14th century.

They were ultimately replaced by a system which contributed what was one of the most important discoveries of early science: a fully positional notation *with a representation for the number zero*. There is evidence that the Mayan civilisation employed a symbol for zero in their number system. Its arrival in European science came via knowledge of Arabic mathematics. Arab scholars had themselves learned of this system from Indian civilisation. The Indian discovery dates from, at the latest, 200 B.C (the earliest recorded use, in a textbook, by Bakhshali). The word 'zero' in English, itself originates from India (Sanskrit *sunya* — empty or blank — translated as *zifr* by Arab writers, hence Latin *zephirum* and English *zero* and *cipher*). The Arabic system employed 10 different symbols representing the numbers $0,1,2,\ldots,9$ and formed the basis of the *decimal system* that is used today. First popularised in Europe by Leonardo of Pisa in his *Liber Abaci* (The Book of Computation) of 1228, despite attempts to suppress it during the reaction against Islamic scholarship, the obvious advantages of the system over Roman numerals eventually led to its being universally adopted.

We now turn to the development of the earliest algorithms and mechanical computing devices.

1) $\mu\upsilon\rho\iota o\iota$ is typically translated as $10,000$ but is more accurately rendered as 'countless'. Herodotus' estimate of the Persian forces at the Battle of Marathon and the source of the English word 'myriad'.

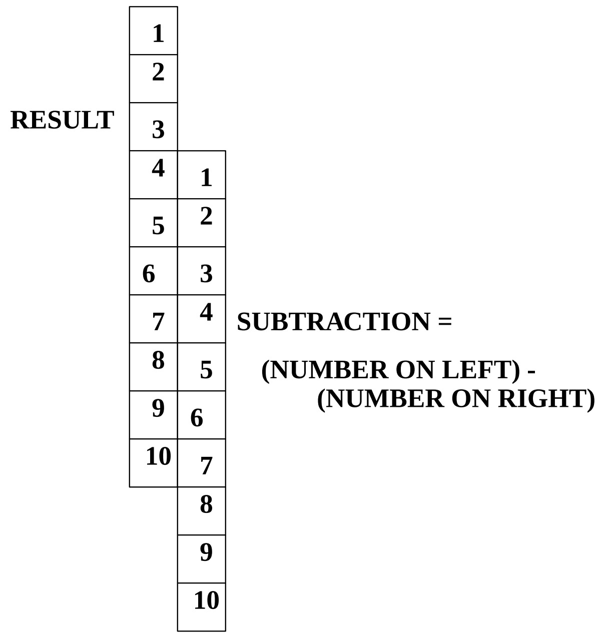

Figure 3: Simple Addition and Subtraction Device

Figure 3 shows the structure of one of the earliest forms of primitive computing device that could be used for addition and subtraction. Units were marked out on two lengths of wood. Addition tables and the result of subtractions could be constructed by lining up appropriate units. Other than such basic mechanisms, *look-up* tables were constructed. At Senkereh in what was Babylonia a clay tablet dating from between 2300-1600 B.C has been discovered which contains the squares of the first 24 numbers. This is now in the British Museum. Such tables obviate the need to recompute particular values and were probably constructed using stones as counters, e.g. Figure 4.

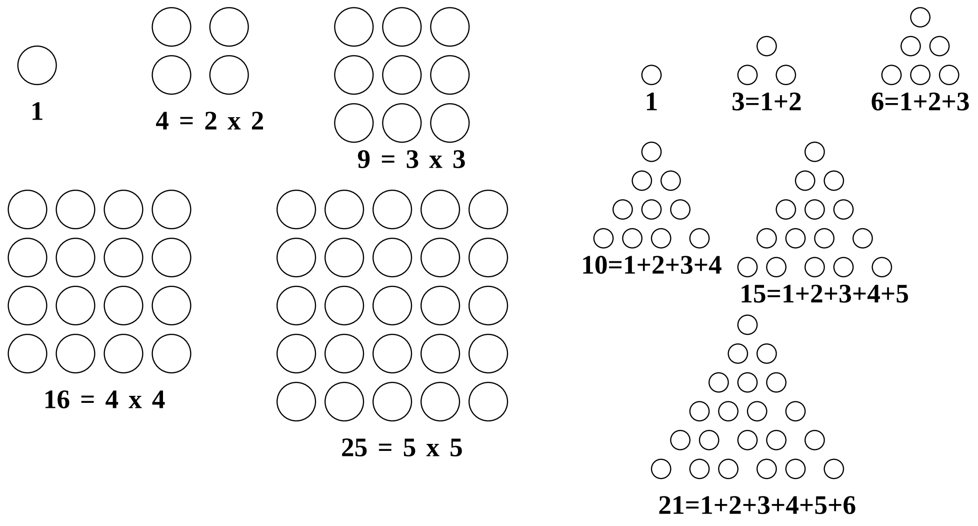

Figure 4: Computing Squares and Sums By Arranging Counters

The use of pebbles and arrangements of these as aids to computing was common to many cultures. Its influence is apparent in many terms still used today, e.g *square* (via arrangements in Figure 4), *triangular* numbers which represent the sums of the first *n* numbers; *calculus* and *calculation* both deriving from the Latin for pebble (hence the different usages of the word *calculus* in mathematical sciences and medicine); *cheque* and *exchequer* which derive from the mediaeval English custom of calculating tax levies by piling stones on a board marked out in black and white squares (the word is combination of Latin *ex* meaning ‘from’ and Norman French *cheque* for the checked pattern).

The use of counters as an aid to computation reaches its greatest level of sophistication and power in a device which is still very extensively used today in the Far East: the *abacus*. Figure 5

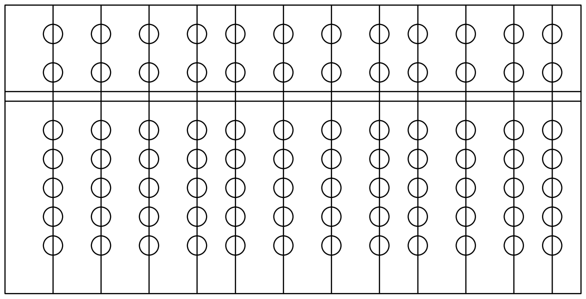

Figure 5: Structure Of Japanese Abacus

The abacus represents the state of a calculation by the position of beads strung into columns on wires. The earliest invention of the abacus is now credited to the Babylonians (the word abacus derives from a Phoenician word *abak*). By the sixth century B.C. they were widely used in Greek society: the historian Herodotus (*ca.* 484-424 B.C) mentions them in this period and left records of complex calculations performed on them, e.g. the accrued interest on a loan of 766 talents, 1095 drachmae and 5 obols over a period of 1464 days at a rate of one drachma per day for every 5 talents. Other incidences of the use of abaci in late Greek culture are Eutocius of Ascalon’s computation of $(30133/4)^2$; references in the surviving speeches of the orator Demosthenes (385-322 B.C); and the writings of the Cynic philosopher Diogenes (d. *ca.* 320 B.C). It was in Oriental cultures, principally China and Japan, however, that the abacus reached its highest level of development. The Japanese contributed the concept of dividing the abacus frame horizontally into two zones (the so-called *Heaven* zone with 2 beads and *Earth* Zone with 5)[2] With this device calculations of great complexity could be performed at great speed. In 1946, in a meeting between the fastest mechanical calculator operator in the U.S. Army, one Private. T.N. Wood equipped with a contemporary state-of-the-art calculator, was defeated in 4 out 5 speed contests by Kiyoshi Matsuzaki, who used an abacus. This device is still widely used in banking and financial calculations in Japan today.

It should be noted that even such crude devices as the notched sticks of Figure 4, presume some understanding of the *algorithmic process* determining their use. It is likely that the first employers of simple computational aids were unaware that the reason *why* they produced the correct answer was because they exploited simple algorithms relating the data operated upon to the results obtained. The Greek mathematician, Euclid (4th Century B.C) is the earliest known identifier of specific algorithms (although it is clear that the calculations, such

2) Although the modern Japanese abacus has reduced the heaven zone to a single bead.

as those documented by Herodotus on abaci — compute the interest due on amount x at a rate of y per z for each w — presuppose methods stating how to proceed). Euclid’s work *The Elements* is of collection of results, mainly in Geometry, spread over 7 books. It is important, for our purposes, for two reasons:

i. It is the earliest known attempt to formalise the concept of algorithm, i.e. to separate ‘admissible’ computational processes from ‘inadmissible’ ones.

ii. Book VII, dedicated to properties of numbers, describes a number of important algorithms at least one of which is still taught today.

As regards the first point, Euclid addressed the issue of what algorithms could be said to be valid if one was concerned with constructing various geometrical objects, e.g. squares with a certain area; lines and angles with particular properties; regular polygons, etc. *The Elements* admits such an object if and only if one can show how to construct it in a *finite* number of steps using only a ruler, i.e. straight edge to draw lines, and a pair of compasses (to draw circles and arcs). See Figure 6.

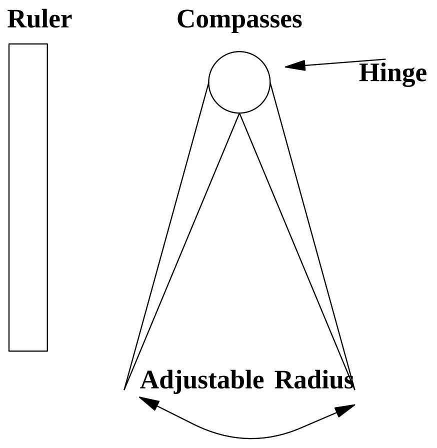

Figure 6: Ruler and Compasses

For example one may easily check that the algorithm below constructs a regular hexagon — a six sided figure in which all sides have equal length and all internal angles are equal:

1. Draw a circle C of radius r (uses compasses)
2. Choose a point ϕ on C and draw a circle with centre ϕ and radius r. The new circle cuts C at two points α and β.
3. Draw a circle with centre α and radius r. This cuts C at ϕ and a new point γ.
4. Draw a circle with centre γ and radius r. This cuts C at α and a new point δ.
5. Draw a circle with centre δ and radius r. This cuts C at γ and a new point ε.
6. Draw a circle with centre ε. This cuts C at δ and β.
7. Use a straight line to connect $\phi \rightarrow \alpha \rightarrow \gamma \rightarrow \delta \rightarrow \varepsilon \rightarrow \beta \rightarrow \phi$ The resulting polygon is a regular hexagon.

Ruler and Compass Construction of Regular Hexagon

Stages of this process are illustrated in Figure 7.

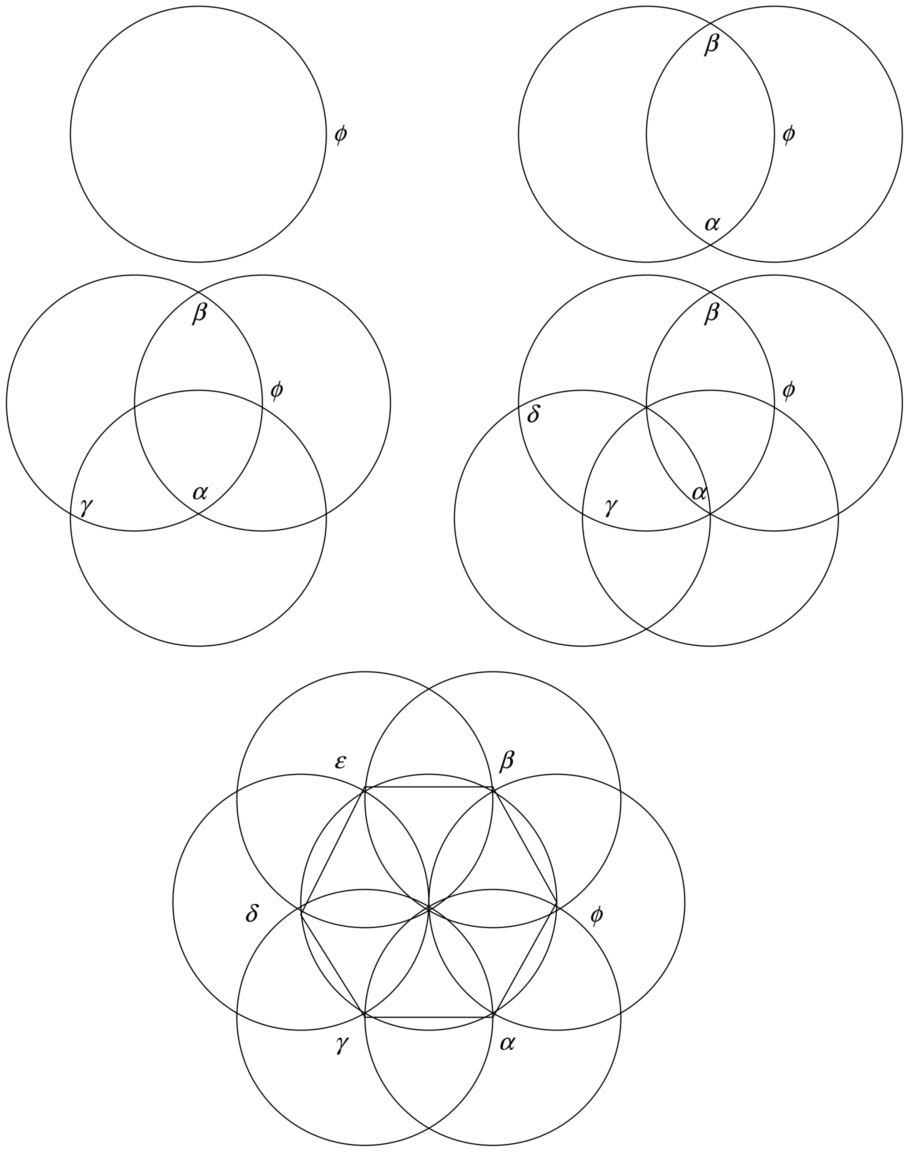

Figure 7: Ruler and Compass Construction of Regular Hexagon

Although this may seem a very primitive class of algorithm, problems of great subtlety arise in it. In particular, the problem of squaring the circle (i.e. construct a square whose area is equal to that of a given circle) was first raised by Euclid. This was not shown to be impossible using ruler and compasses until the end of the 19th Century[3] — over 2300 years after Euclid's death.

The second important contribution of *The Elements* is the algorithm to calculate the *greatest common divisor* of two numbers. Given two numbers — m and n — the greatest common divisor of m and n, denoted $gcd(m,n)$, is the largest number that divides both m and n without leaving any remainder, e.g. $gcd(12,40)=4$, $gcd(7,15)=1$, etc. *Euclid's Algorithm*, as it is now known, is shown below:

3) By Lindemann in 1882. Of the other important constructions left open by Euclid — trisection of a given angle and duplication of a given cube — the former was proved impossible by Descartes (1637), the latter was known to be impossible by Arabic mathematicians.

```
greatest common divisor of $m$ and $n$

    1. If $m$ is smaller than $n$ then swop the values of $m$ and $n$.
    2. Set $m$ equal to the remainder of $m$ divided by $n$.
    3. If $m$ is not equal to 0 then go back to step 1, with the
           new values of $n$ and $m$.
    4. Return $n$ as the answer.
```

Euclid's Algorithm

Euclid's work was not the only early investigation of algorithms. Another important Greek contribution concerns a problem which is still very much of interest today — the generation of *prime numbers*. A prime number is a whole number, greater than 1, which is divisible without remainder by only itself and by 1. Methods of finding prime numbers have existed for over 2000 years and, since the only known algorithms are computationally very demanding, finding large primes is a standard performance test carried out on new powerful computer systems: in the last 40 years the size of the largest known prime has advanced from a 40 digit number (found in 1886) to a 15,000 digit number (identified in 1991). The first algorithm to identify prime numbers is the famous *Sieve of Eratosthenes* (fl. 3rd Century B.C). This works as follows: suppose on wishes to know all the prime numbers less than some number, n say. Write down all the numbers from 2 to n and then carry out the following steps:

```
1. Let $k := 2$
2. For each number, $m$, between $k + 1$ and $n$ if $m$ is an exact
       multiple of $k$ then cross $m$ of the list of numbers.
3. Set $k$ to be the smallest uncrossed off number left.
4. If $k$ is less than $n$ then repeat process from step (2).
5. Any number which has not been crossed off is a prime number.
```

The Sieve of Eratosthenes

The process is illustrated below:

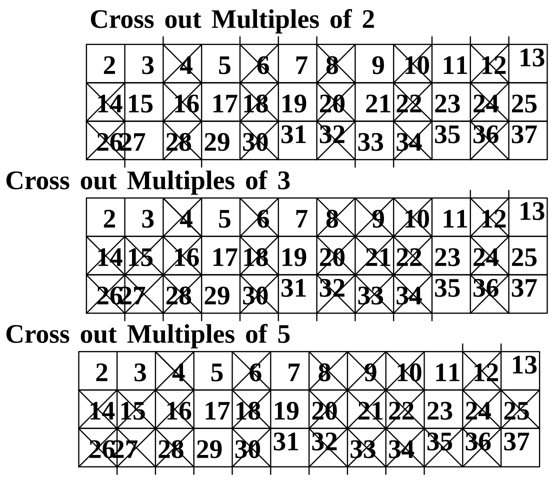

Figure 8: Sieve of Eratosthenes

The word *algorithm* in English comes from the Arab mathematician *al-Khowarizmi* who flourished at the end of the 8th Century A.D. Al-Khowarizmi had reported the Indian discovery of the decimal number system with zero and introduced a number of important concepts in his book *Al-jabr wa'l muqabala* whose title gives us the word *algebra* in English. Another Arab mathematician, *al-Khashi* (1393-1449), devised a method of computing the decimal expansion of π (the ratio between a circle's circumference and its diameter)[4], obtaining a result accurate to 16 places. Al-Khashi also devised the first mechanical computing devices which could be used to predict the occurrence of important celestial phenomena such as solar and lunar eclipses.

In summary, by the middle of the 15th Century there had evolved a flexible and powerful system for representing numerical quantities (both whole numbers and decimal fractions); some basic algorithmic techniques had been developed for manipulating these representations to deal with various calculations; and the first moderately sophisticated aids to such calculations had been developed.

One of the important trends in the history of this topic becomes apparent around this period: that as the level of scientific and engineering knowledge increases, so in parallel does the complexity of the mathematical calculations associated with their development. Thus, to date, there has never been a point reached where the contemporary state of computational aids has exceeded the most demanding requirements of contemporary sciences. We can cite two examples of this at the end of the 15th century. First, increasing knowledge about matters such as planetary motion had resulted in data from observations that was increasingly difficult to process and interpret with the methods available. Second, there was little in the way of devices to help with navigation at a time when there was a considerable increase in exploration and commercial trade. In the next section of these notes we examine the continuing development of computational mechanisms following this period.

4) A method of memorising this is the phrase: 'How I need a drink, alcoholic of course, after all those lectures involving tedious mnemonics for pi'. The number of letters in each word gives the sequence of digits in the expansion of π.

Mechanical Aids to Computation and the Development of Algorithms

3. Mechanical Calculators prior to the 19th Century

3.1. Introduction

It was observed, at the conclusion of the last chapter, that the general adoption of a notational formalism sophisticated enough in which to express some complex calculations, coincided with a greatly increased requirement for techniques and tools that could assist in difficult numerical analyses. Consider, for example, the following fields in which such computational problems became of great importance from the middle of the 15th century onwards:

A. Navigation

By the middle of the 16th century the first major European exploration of the Americas was well advanced, trade-routes by sea had been established with parts of the Indian sub-continent and the Far East, and the first circumnavigation of the globe had been completed. The growth of merchant trading houses, in Italy and other European sea powers, coupled with the demand for imports from distant parts of the world, created a need for more and more sophisticated navigational instruments. Thus, in the mid-16th century there were no suitable mechanisms for constructing detailed maps, for accurately measuring distances at sea, and thus for precisely determining how long a particular voyage would take to complete. As a result, merchants financing trading expeditions would be faced with ruin if a ship bringing back goods arrived too early (so that the available market was already saturated with the imported goods from a rival trader or earlier journey) or too late: if the financial backing for the voyage had been raised by a loan secured on the projected profit, the date set for repayment might have passed by the time a ship had returned.[1] While computational mechanisms to help in navigation would not in themselves be sufficient to alleviate all potential difficulties, such tools would be of some assistance in planning expeditions, estimating how long they would take, and scheduling departures in order to avoid adverse weather conditions *en route*, e.g. if a particular sea area is known, from previous experience, to be prone to violent storms during certain parts of the year, then a ship that had to travel through such an region could have its departure dates fixed to try and avoid that period. A primitive and crude measuring device — the geometric compass — was produced by Galileo in 1597: this was of some assistance in translating distances on maps to distances at sea.

B. Financial assessments

Even with the vastly more sophisticated technology available today, the calculation and analysis of financial data is an extremely complex process, e.g. business concerns must keep record of transactions carried out in order to: make legal tax and V.A.T returns, pay their work-force appropriately, and set the price of goods and services competitively; similarly, at the level of Government — local, national, and, to an increasing extent, supra-national — accurate assessment of finance is critical in determining taxation policy, limiting government spending plans, and predicting the 'likely' trend of important economic indicators. Although the scale of similar pecuniary activities was considerably less in the 16th century, nevertheless this was more

1) A scenario used in Shakespeare's *The Merchant of Venice* (*ca.* 1596): Antonio's troubles with Shylock arise as a result of the former being unable to repay the loan advanced to him due to the failure of his merchant ships to arrive within a month of their scheduled date.

than offset by the absence of any powerful tools for assisting with the relevant calculations. We noted above the growing importance of merchant traders, particularly in Italy, as regards navigation. In city states, such as Florence and Venice following the Renaissance, the major merchant families exercised enormous financial, and thereby political, power. In order to maintain such influence it was important to such groups that their mercantile concerns[2] were as successful as possible: errors in business calculations might result in too little or too much of a specific commodity being available and/or an uneconomic price being charged.

In the same way, while the contemporary social organisations did not lead to the tax regime which is common today, when revenue was required by a state for some purpose, e.g. financing a military campaign, the minimum amount to raise had to be assessed and a mechanism by which this amount could be realised, determined[3]. Thus, as with navigation, in the spheres of finance and commerce there was little in the way of tools and methods to assist in the calculations required, at a time when these were becoming increasingly more complicated activities.

C. The study of mechanics and planetary motion

The Italian astronomer and mathematician Galileo (1564-1642) in work carried out around 1600, had noted that the behaviour of certain natural phenomena could be described through the use of mathematical models, e.g. the path that might be taken by a projectile. By the end of the century such ideas had been developed into a detailed mathematical theory of mechanics and motion by, principally, Newton (1642-1727). One of the fields in which such developments were of notable scientific and practical importance was the study of planetary motion. Historically this had always been an important activity in European culture: the position of planets as seen from Earth relative to the fixed stars, formed the basis for astrological prognostications. Copernicus (1473-1543) had developed his heliocentric theory by mathematical analyses of ancient observations of planetary position. This work involved extremely cumbersome arithmetic calculations. Thus, as the Copernican theory became more widely accepted[4] attempts began to produce more accurate theories from new observational data[5], Once again, however, the problem of carrying long and cumbersome calculations in order to verify experimental hypotheses arose. In summary, applying and verifying the correctness of mathematical models of motion often involved, what were at the time, extremely complex operations, such as the calculation of square roots or trigonometric functions.

3.2. Tabular methods of making calculation easier — Logarithms

In 1614 the Scots mathematician John Napier of Merchiston (1550-1617) published a paper entitled *Mirifici logarithmorum canonis descriptio*[6] in which he demonstrated how the difficult processes of multiplication, division and root extraction could, given suitable information,

2) Despite its present-day importance, banking was not a powerful (political) influence: the Roman Catholic and Protestant churches prohibited their adherents from charging interest on loans of money, the practise which forms the main source of income for banks.

3) A formal system of Government set Income Tax is a comparatively recent development (early 19th century in the U.K).

4) It must be recalled that at this time, publicising arguments contrary to classical (i.e Aristotelian) philosophy was frowned upon: Copernicus' *De Revolutionibus Orbitum Coelestium* was immediately placed on the Roman Catholic *Index Librorum Prohibitorum* upon its publication in 1543 (and was not removed from this list until 1837). Opposition to these ideas continued for over 50 years, e.g. after an ecclesiastical trial held in Venice and lasting seven years, on 17th February 1600, the philosopher Giordano Bruno of Nola (1548-1600) was burnt at the stake in Rome for, among other offences, promoting the validity of Copernicus' ideas.

5) cf. the work of Kepler and Brahe as described elsewhere in this course.

6) 'A description of the miraculous working of the rules of logarithms'

be reduced to the relatively easy processes of addition and subtraction. Napier's method is based on a very simple idea. Suppose we take any number, x which is greater than zero. Then for any two numbers p and q the relationship 'x raised to the power of p' multiplied by 'x raised to the power of q' is equal to 'x raised to the power of the *sum* of p and q', i.e. $x^p \times x^q = x^{p+q}$. Similarly 'x raised to the power p' divided by 'x raised to the power q' is equal to 'x raised to the power of the difference between p and q', i.e. $x^p \div x^q = x^{p-q}$. How does these relationships assist in performing multiplication and division? Suppose we wish to multiply two numbers v and w. If we can find two numbers c and d say such that $v = x^c$ and $w = x^d$, then $v \times w = x^{c+d}$ and so the result of multiplying v and w is the *unique* number y with the property $y = x^{c+d}$. Here c (resp. d) is called the *logarithm* (to the *base* x) of p (resp. q); the answer (y) is the *anti-logarithm* (to the base x) of $c + d$. For example suppose $x = 2$ and we wish to multiply $p = 16$ and $q = 128$. We have $p = 16 = 2^4$ and $q = 128 = 2^7$, hence $c = 4$ and $d = 7$, thus $16 \times 128 = 2^{11} = 2048$. In this 4 is the logarithm (to the base 2) of 16; 7 is the logarithm to the base 2 of 128 and 2048 is the anti-logarithm to the base 2 of 11.

Of course there is an obvious, immediate problem with this technique: once we have fixed the base x (2 in the example above) we need to know the logarithms with respect to this base of any numbers to be multiplied and the antilogarithm of the result of adding or subtracting these. Since this calculation is itself likely to be extremely cumbersome, ideally one needs a *table* of logarithms and antilogarithms that have already been constructed. Thus, suppose we have the following information available:

- A list of the logarithms (to the base 10, say) of all numbers (to some precision) between 0 and 100.
- A list of the antilogarithms (to the base 10) of all numbers (to some precision) between 0 and 4.

Then with such tables we can multiply and divide *any* two numbers (with a reasonable degree of accuracy depending on the precision of the tables). Notice that, if some small degree of error is acceptable — and the extent of the tables used will make such errors inevitable anyway — we can indeed multiply or divide any two numbers. If a given number is too small or too large then there is a simple transformation that can be applied to make the calculation possible. The method of calculating using *log tables* was commonly taught in schools in the U.K well into the mid 1970s (when electronic calculators obviated the need for them) and this provided a standard method for involved numerical calculations arising in science and technical applications until the advent of reliable electronic mechanisms. The technique is applied by the following algorithm:

```
Input: 2 numbers $p$ and $q$
Output: $p \times q$ (or $p \div q$)
Method:
  1. Find the logarithm of $p$ in the table given (or of the number closest to $p$);
     call this number $c$
  2. Find the logarithm of $q$ in the table given (or of the number closest to $q$);
     call this number $d$
  3. Add $c$ and $d$; (or subtract $d$ from $c$ if division is wanted);
     call the result $w$.
  4. Find the antilogarithm of the number nearest to $w$ in the table of antilogarithms
     and return this as the result of $p \times q$ (or $p \div q$).
```

Algorithm for calculating using logarithms

Napier's original work announced the *method* of calculation by logarithms but his tables were not very easy to use. Napier's tables used the constant $1/e$ as the base[7] of the logarithm, i.e. the value x in our description above.

The production of the first detailed 'practical' table of logarithms was undertaken by Napier's contemporary, the English mathematician Henry Briggs (1561-1631). Soon after the publication of Napier's paper, Briggs' recognised the importance of using 10 as the base. This discovery was of crucial importance in simplifying and extending the applicability of logarithms, *viz.* suppose one wishes to know the log to the base 10 of 9.82 and one has available only the logs of the whole numbers between 1 and 1000: since $9.82 = 982 \div 100$; and $\log_{10} 100 = 2$, i.e. $100 = 10^2$; therefore $\log_{10} 9.82 = \log_{10} 982 - 2$. Briggs communicated this idea to Napier and in 1617 they met at Napier's house in Merchiston (a suburb of Edinburgh). The 17th century writer William Lilly relates the following account of their first meeting:

> *When Merchiston [Napier] first published his Logarithms Mr Briggs $\cdots$ was so surprised with admiration of them that he could have no quietness in himself until he had seen that notable person whose only invention they were $\cdots$ Mr Briggs appoints of a certain day when to meet at Edinburgh; but, failing thereof, Merchiston was fearful he would not come. It happened one day as John Marr and the Lord Napier were speaking of Mr. Briggs $,\ldots,$ saith Merchiston, "Mr. Briggs will not come now"; at the very instant one knocks at the gate, John Marr hasted down and it proved to be Mr. Briggs $\cdots$ He brings Mr. Briggs into my Lord's chamber, where almost one quarter of an hour was spent, each beholding the other with admiration, before one word was spoken.*
>
> **William Lilly**
> *Autobiography*

Subsequently, Briggs published a table of logarithms to the base 10 for the whole numbers between 1 and 1000. He spent the remainder of his life producing tables for all numbers between 2000 and 29000 and 90000 and 100000. Briggs' tables were accurate to 14 decimal places, an astonishing feat of calculation given the absence of any mechanism to assist in its generation. In the years following Briggs' death the gaps in his tables were filled in and tables of the logarithms of trigonometric functions — such as *sine* and *tangent* also calculated[8].

Napier's and Briggs' development of logarithms represents one of the most important scientific achievements of the 17th century. As a result of this breakthrough, what were once enormously difficult computations could be performed with great accuracy by anyone who had mastered addition and subtraction. As we observed earlier, the use of logarithms as an aid to calculation continued well into the late 1970s when the method was still being taught as part of school mathematical courses.

An important by-product of logarithms was the *slide rule*, another computing aid that was very widely used in scientific and technical calculations until the appearance of electronic calculators. In the previous chapter we saw that an early form of calculating device was provided by using two lengths of wood, marked with equidistant symbols, in order to carry out addition. In 1620 the English mathematician William Gunter (1581-1621)

7) The fact that Napier chose this constant suggests that he was, probably, unaware of the general concept of 'base of a logarithm'. e is a constant that, like π, arises in many mathematical analyses. Unlike π, defined as the ratio between the circumference and diameter of a circle, there is no simple physical definition of e. While there are several precise characterisations of the value of e using ideas from advanced mathematics — e.g. in calculus e is the unique constant such that the function $f(x) = e^x$ is its own derivative — Napier would not have known of these.

8) Logarithms to the base 10 are now known as *common logarithms*, whereas a minor modification of Napier's system (using the base e, instead of $1/e$) is given the name *natural* or *Napierian logarithms*. The latter often arise in mathematical analysis. Base 2 occurs in several Computer Science applications.

recognised that the same principle, coupled with the ideas underlying logarithms, could be exploited to construct a device with which rough estimates of division and multiplication calculations could be made. Thus instead of using marks which were placed equidistantly, successive marks were placed at (appropriately) decreasing distances, e.g. the distance from the mark representing 1 and the mark representing 2 would be same as the distance between the mark representing 2 and that representing 4, etc. Gunter's calculating aid consisted of a grid on which numbers could be multiplied and divided by adding and subtracting lengths with the assistance of a compass. The slide rule in its modern form, however, was the invention of another English mathematician — William Oughtred (1574-1660). The form of modern slide rules is outlined in Figure 9 below:

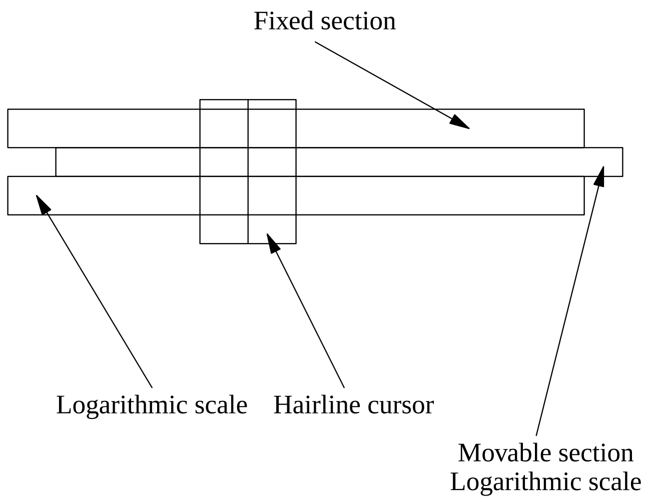

Figure 9: Structure of Modern Slide Rule

The fixed and movable sections are marked off using a *logarithmic scale*, i.e. one in which the distances between quantities varies according to the regime described above. The fixed section may also contain scales corresponding to other mathematical functions, e.g. trigonometric functions, square roots etc. The movable section is aligned with values on the fixed section during a calculation with the cursor being used to assist in reading off results. High-quality slide-rules are capable, in the hands of an experienced user, of giving answers to the precision of 4 or 5 places. The method by which quantities are multiplied using a slide rule is described in the algorithm below:

Input: p and q, numbers between 1 and 10
Output: The result of multiplying p and q
Method:
1. Adjust the movable section until the place marked 1 on this is aligned with the place marked p on the fixed section.
2. Find the place marked q on the movable section.
3. The number on the fixed section which is aligned with q is the result of multiplying p and q.

Procedure for multiplication using a slide rule

Notice that the restriction forcing p and q to be between 1 and 10 is not serious: if either is outside this range then it is easy to adjust the values to be multiplied, e.g. 982×0.15 is the same calculation as $9.82 \times 1.5 \times 10$.

Towards the end of his life, Napier invented a device which, for many years, was more highly regarded than his researches concerning logarithms: a mechanism for simplifying the task of multiplying numbers that has since become known as *Napier's bones*[9]. Napier's bones were, in effect, a clever representation of multiplication tables: Figure 10, below, depicts the rods used for the numbers 1 to 8. Each rod contains 9 squares: the first is inscribed with the number associated with the particular rod; the remaining 8 are each bisected by a diagonal running from the lower left to the upper right; the nth square contains the result of multiplying the rod-number by n so that the upper triangle of the square contains the most significant figure and the lower triangle the least significant figure, e.g. in the rod numbered 8, the sixth square (numbering from 2), contains the number 48 written in this form.

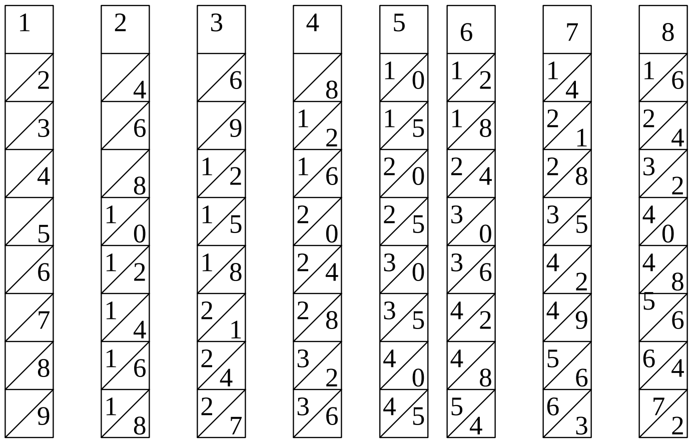

Figure 10: Napier's Bones

Napier's bones could be used to set up a 2 to 9 times multiplication table for any number. Given a particular number one selected the rods corresponding to the digits in the number and placed them together in a rack whose side was labelled from 2 to 9. To multiply this by 6, say, one proceeded along the row marked 6 going from right to left adding the numbers in each parallelogram to give the next digit. Figure 11 shows how the rods would be set up to

9) It is indicative of the religious strife during the times that Napier lived in that he considered neither this invention nor his discoveries concerning logarithms to be his most important work. Napier was certain that he would mainly be remembered for his lengthy anti-Catholic tract entitled *Plaine Discovery of the whole Revelation of Saint John*; an item of work which is now almost forgotten.

multiply by the number 132,577.

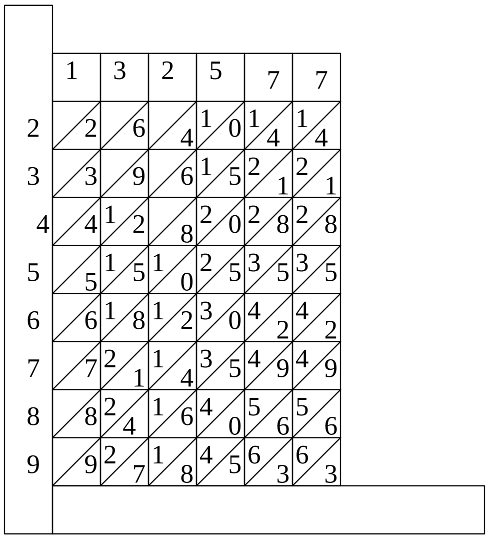

Figure 11: Multiplication Table for 132,577

To find the result of multiplying 132,577 by 9 one has the following squares in the 9th row:

0/9 2/7 1/8 4/5 6/3 6/3

With these we have: $132,577 \times 9 = 1,193,193$, i.e. the rightmost 3 is the rightmost 3 in the row; then going from right to left: $9 = 3 + 6$, $1 = 5 + 6$ (with a carry-over of 1); $3 = 8 + 4 + 1$ (again with a carry-over); $9 = 7 + 1 + 1$; $1 = 9 + 2$ (with another carry-over of 1); and $1 = 0 + 1$.

Napier's invention was extremely successful and was very widely used. Many different versions were manufactured and employed by accountants, bookkeepers, and others whose work routinely involved computing products of numbers. The sets of rods came in a number of different sizes and were normally engraved on wood, however, in rare cases ivory was sometimes employed. As late as the mid-1960s, Napier's bones were still being used in primary schools in Britain to assist in teaching multiplication.

3.3. The first mechanical calculators — Schickard, Pascal, and Leibniz

Tables of logarithms, slide rules, and Napier's rods reduced the complexities of multiplication and division to the comparatively easy processes of addition and subtraction. In our description of these devices above, we also outlined the methods by which specific calculations were performed with them, i.e. the *algorithms* which someone would employ. It should be clear that these algorithms were quite simple. The next development was the invention of *mechanical* systems that went some way to *implement* such algorithms.

In the present day, the idea of taking some routine, repetitive, and methodical task and automating it, is a commonplace; something which rarely strikes one as novel or surprising. To some extent, however, this is because we have become accustomed to the concept of automation, e.g. in factory assembly lines, or the various dealings with computer database systems such as those described in the opening lecture. Thus, since we are aware that *some* processes can be automated, new applications tend to go unreported, unless they involve some significant technical development. With such an attitude the construction of the first 'semi-automatic' calculating machines may appear to be an unremarkable development. We can advance two reasons as to why this is not the case. Firstly, as we observed above, new spheres of automation are found less surprising today because we are aware of precedents: in the early 17th century, when the first mechanical calculators were thought of, the concept of performing calculations by machine was a radically new idea. Secondly, the technological resources available today render many tasks much simpler to automate, e.g. very rapid computer systems rely on electrical power and developments in device physics: in the 17th century no such resource was available, thus in order to build calculating machines one only had recourse to mechanical ingenuity.

The invention of the first mechanical calculator is now credited to the German polymath Wilhelm Schickard (1592-1635). Schickard's contribution has only recently been recognised, largely on account of the researches of the historian Franz Hammer and the mathematician Bruno von Freytag Löringhoff. Hammer, an authority on Kepler's writings, discovered notes and correspondence from Schickard while preparing an edition of Kepler's complete works. In 1935, Hammer found a letter from Schickard to Kepler containing a rough drawing of and detailed description of Schickard's calculating machine.[10] Unfortunately, Schickard's letter referred to a more detailed sketch of the machine, which was not among the correspondence examined by Hammer. In 1956, however, twenty-one years after his initial discovery, Hammer came across a more detailed diagram among an archive of Schickard's papers in Stuttgart. This diagram also provided instructions as to how to build the machine. Hammer was unable to determine precisely how Schickard's machine operated, however, von Freytag, working from the rediscovered documents and a knowledge of contemporary mathematical techniques, was able to build a replica of the machine. The working version was finally completed in 1960.

Schickard's machine employed a simple mechanical device that continued to form the basis of calculating machines right up to the appearance of the first electronic computers: addition and subtraction are performed by the movement of geared wheels linked to a numeric display. Thus the effect of adding one to a displayed number would be accomplished by rotating the appropriate wheel so that the next digit was displayed[11]. A rough outline of such a device is shown in Figure 12, below.

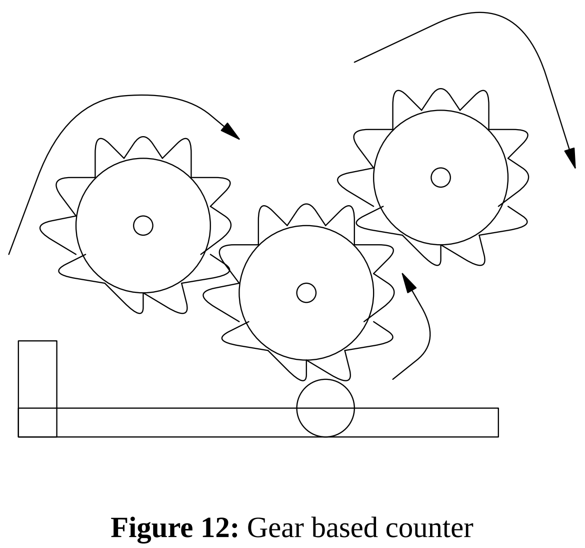

Figure 12: Gear based counter

In devices like this the main problem to be solved is that of recording carries and borrows resulting from additions and subtractions. Schickard solved this by employing a complex

10) A facsimile of this letter appeared in the edition *Litterae ad Kepplerum*, prepared by Hammer in the 1930s.

11) Much the same principle is used in odometers in cars.

system of mutilated and auxiliary gears. Schickard informed Kepler of his invention in letters written between 20th September 1623 and 25th February 1624:

> What you have done by calculation I have just tried to do by way of mechanics. I have constructed a machine consisting of eleven complete and six incomplete sprocket wheels which can calculate.
>
> Letter to Kepler, 20th Sept. 1623

> I had placed an order with a local man, Johan Pfister, for the construction of a machine for you: but when half-finished, this machine, together with some other things of mine, especially several metal plates, fell victim to a fire which broke out unseen during the night $\cdots$ I take the loss very hard, now especially, since there is no time to produce a replacement soon.
>
> Letter to Kepler, 25th Feb. 1624

The above letter of 1624 is the last, known, extant correspondence of Schickard, concerning his invention of what he dubbed the *Calculating Clock*. Schickard died from bubonic plague on the 24th of October 1635. It is probable that his surviving papers were lost or destroyed during the Thirty Years War, that raged through central Europe in the mid-17th Century: Schickard's home town, Tübingen, was particularly badly affected by the course of this war.

Prior to the discoveries of Hammer and von Freytag Löringhoff concerning Schickard's researches, the construction of the first mechanical calculator was generally attributed to the French mathematician, scientist and philosopher Blaise Pascal (1623-1662).

Pascal was born in 1623 in the Auvergne region of France, and first came to prominence as a precociously gifted mathematician at the age of 16. Although he died at the relatively young age of 39 during his lifetime he made significant contributions to mathematics (being, with Fermat, one of the founders of modern probability theory), and hydraulics: Pascal was the first to demonstrate the existence of air-pressure and vacuums; the importance of his work in the latter field is recognised in the adoption of his name as the S.I. unit of pressure. Pascal's researches into mechanical calculators were motivated by the problems faced by his father Etienne Pascal. Etienne Pascal had been a successful lawyer and presiding judge who had fallen out of favour with the authorities, but thanks to the intercession of powerful friends, was eventually appointed as a tax commissioner based in Rouen. Unfortunately, his appointment, in 1639, coincided with an enormous revision of the tax levying system in France, as Cardinal Richelieu sought to raise money to conduct a war against Spain. Even with the assistance of his son, Etienne Pascal frequently found himself working into the early hours of the morning as he recalculated levies and rates to be raised. The only 'mechanical' tools available were the mediaeval counting boards (or *exchequers*) that were briefly alluded to in the last lecture. Almost all of the computation involved repeatedly adding and subtracting various totals. In 1642, at the age of 19, Blaise Pascal suggested to his father that it could be possible to design a machine that was capable of dealing with these simple calculations automatically. Pascal's machine — the *Pascaline* as it became known — was completed in 1644 after almost three years of experimentation. A schematic of the front panel layout of the first Pascaline is depicted in Figure 13 below.

The prototype Pascaline could add and subtract five digit numbers — not large enough for practical purposes, but Pascal subsequently had six and eight digit variants constructed. Subtraction with the machine was not an easy operation. As with Schickard's calculating clock, the arithmetic operations were performed by rotating gearing mechanisms which affected the digits shown in the upper section. The user set up the required calculation by rotating the dials in the lower half. Unlike Schickard's machine, however, the method by which the Pascaline dealt with carry-overs was extremely clumsy. This had two undesirable side-effects: the

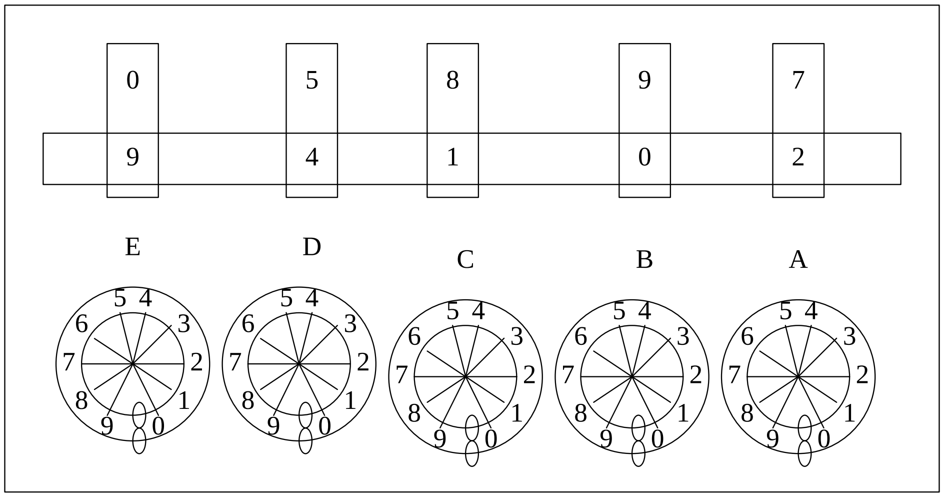

Figure 13: Front Panel of Pascal's Calculator

gearing mechanism was prone to jam; and the numerical dials (hence gears) could only rotate in one direction. This latter drawback meant that subtraction became a far more complicated process. The numeric display consisted of two distinct sections, a sliding bar being used to uncover whichever section was relevant to the operation being performed. Subtraction was carried out by exploiting a notational trick called "9's complement representation" the use of which allows subtraction to be reduced to addition.[12]

Overall, despite its ambitious design, the Pascaline was a rather inelegant and difficult to use machine. It seems probable that had Schickard's ideas and construction found a wide audience[13] then Pascal's machine would not have been invented.

Nevertheless, the Pascaline enjoyed an enormous vogue, to such an extent that Pascal applied for a patent to protect his invention. Pascal had the misfortune for his patent (or privilege as it was then called) application to need the approval of the Chancellor Peter Seguier, one of the people that Etienne Pascal had offended in 1638. Seguier received the application in 1645 but delayed approving it until late in 1649. Contemporary documentation indicates the anger that Pascal felt upon seeing his invention being imitated by others:

> I have seen with my own eyes one of these false products of my own idea constructed by a workman of the City of Rouen, a clockmaker ⋯ After being given a simple account of my first model, which I had constructed several months previously, he was bold enough to attempt another, and what is more, with a different kind of movement; but since the fellow has no aptitude for anything ⋯ and does not even know whether there is such a thing as geometry or mechanics, the result was that he simply turned out a useless object ⋯ so imperfect inside that it was no good for anything; but owing simply to its novelty it aroused a certain admiration among *people who know nothing at all about such things* ⋯ The sight of this little abortion was extremely distasteful to me and so chilled the enthusiasm with which I was working at the time that I dismissed all my workmen, fully intending to abandon the enterprise owing to the fear that others might set to work with the same boldness and that the spurious objects they might

12) In modern digital computers a variant of this trick, called "1's complement" is used to represent negative numbers and hence perform subtraction in the same way as addition. While Pascal discovered neither method, his calculator is of historical interest as the earliest known device to employ it.

13) It should be noted that Pascal, and indeed almost every contemporary of his, was completely unaware of Schickard's work. Kepler was probably the only significant scientific figure to know of this and there is little evidence to suggest that that he had seen Schickard's machine in operation or could have reconstructed it. Thus, the Pascaline was an independent development.

produce from my original thought would undermine both public confidence and the use that the Public might derive from it.

Publicity statement by Pascal
Rouen, *ca.* 1645

Once the initial novelty of the Pascaline had disappeared, however, the machine failed to be widely sold. Partly this was because of its lack of robustness and habit of becoming jammed during calculations and partly this was because of its expense: almost a year's salary for a middle-income worker. Even those in a position easily to afford this cost, royalty and aristocrats, took no interest in it: this group tended to regard arithmetic as a task to be carried out by servants and underlings. A final reason was the distrust people felt towards machines: a suspicion that since mechanical objects can be made to give incorrect answers (e.g. weighing machines) the so could calculators. It is likely that fewer than 20 machines were sold during its time of manufacture.[14] Six years after the patent was approved, in 1655, Pascal entered a Jansensist convent outside Paris and for the remainder of his life did no further scientific or mathematical work, concentrating instead on philosophical writings until his death in 1669. The famous *Pensées sur la religion et sur quelques autres sujets* of 1660 was a product of this period and is work that is as highly regarded as his mathematical achievements. In recognition of his contribution to mathematics and computer science, a (now extensively used) programming language was given, by Niklaus Wirth its designer, the name PASCAL.

Finally, in this chapter, we come to one of the great figures of 17th century culture: Gottfried Wilhelm von Leibniz — still noted today for his contributions to mathematics, logic, philosophy and calculation. Leibniz was the inventor of differential calculus in the form that it is known and used today: although his development was 20 years after Newton's work, the notation constructed by Leibniz was vastly superior to the crude symbolism employed by Newton. Leibniz was a child prodigy: learning Latin at the age of eight and Greek after a few more years (largely self-taught). He had completed a doctorate in Law by the age of 19 at the University of Altdorf in Nürnberg[15] and subsequently became employed by the Archbishop of Mainz.

For our purposes, Leibniz is of interest because of his invention of a calculator known as the *Stepped Reckoner*. This was an ambitious attempt to build a machine that could not only add and subtract automatically but could also multiply and divide automatically. Neither Schickard's nor Pascal's calculators attempted seriously to address the latter problems, thus Leibniz' machine is the first known calculator that found a mechanism for dealing with these tasks. In this way it is much closer to the mechanical calculators of the late 19th to mid 20th centuries.

The idea of constructing such a machine occurred to Leibniz while he was on a diplomatic mission to Paris in 1672. In a document written in 1685, Leibniz recalled that the initial inspiration was caused by learning of a pedometer that had recently been built. This discovery had prompted the idea that it might be possible to build a machine to perform all the basic arithmetic operations. At first, Leibniz was unaware of Pascal's work but later found out about it from a passing reference to the Pascaline in Pascal's posthumously published *Pensées*. Once he had uncovered the principles underlying Pascal's machine, Leibniz concentrated on applying the same ideas to solving the problems of multiplication and division. By

14) Surviving examples of the Pascaline are extremely rare and considerably sought after by collectors of scientific instruments. The current (September 1993) record auction price for a scientific instrument is the seven figure sum realised for an eight digit Pascaline in 1989.

15) This was in fact Leibniz' second doctorate: he had earlier been refused permission to graduate at the University of Liepzig on account of his age. His Liepzig dissertation — *De Arte Combinatoria* is one of the first attempts to systematise logic as a mathematical tool.

1674 he had progressed sufficiently far to commission a working model of his design: the resulting machine became known as the Stepped Reckoner. Leibniz solved the problem of multiplication by inventing a special type of gear, now called the *Leibniz Wheel*. This consisted of a cylinder in which gearing teeth were set at varying lengths along the cylinder: there were nine rows in total, the row corresponding to the digit k running k-tenths of the distance along the cylinder. By combining a system of these together it was possible to amend a numeric display in a manner consistent with multiplying by a single digit. Although not fully automatic — the machine required user intervention to sort out carry-overs — the Stepped Reckoner was undoubtedly the most conceptually ambitious automatic calculating device that had been attempted. Only one Stepped Reckoner is known to have been built (which is now in a museum in Hannover). There are two reasons for this lack of development: the calculator was a highly intricate work requiring great mechanical expertise to construct; and the machine had one disadvantage — it gave the wrong answers. In 1893, over 175 years after Leibniz' death, the reason for this was discovered: a design error in the carrying mechanism meant that the machine failed to carry tens correctly when the multiplier was a two or three digit number. It is unknown whether Leibniz was aware of this design fault, but in any event, it is probable that a corrected design would not have been constructed.

3.4. Summary

The 16th and 17th centuries saw enormous breakthroughs in the domain of methods and machines for simplifying calculations. Some — as in the development of logarithms — came from increased mathematical understanding; others from the insight of a few individuals that arithmetic tasks could be translated into mechanical analogues. We have seen that two of the important motivations behind these developments were the need to analyse data from astronomical observations; and the desire to reduce the labour intensiveness of calculating taxation levies in the increasingly complicated economic systems that had grown up. It is important to realise that the advances made during this period are of lasting importance. We have noted that many of the calculating techniques — logarithms, slide rules etc. — continued to be extensively used and taught until as little as twenty years ago. Furthermore, the first mechanical calculators, crude though they may seem to us today, did have two significant consequences: the mechanisms developed formed the basis of *all* automatic calculators until the middle of the twentieth century; and, far more importantly, these provided the first real indication that complex arithmetic tasks *could* be solved on machines. An awareness of the last consequence can be seen as central to the ultimate development of electronic digital computers in the form that we know them today.

Mechanical Aids to Computation and the Development of Algorithms

4. 19th Century Contributions and their Impact on Elements of Modern Computers

4.1. Introduction

The prototype calculators of the late seventeenth century demonstrated the feasibility of performing lengthy calculations by mechanical methods. Ultimately this demonstration would result in the construction of the modern computer in the form that is common today. We can at this point, however, note a number of elements — both conceptual and concrete — that are missing from the devices considered previously.

- There is no concept of ‘program’, so that in order to repeat a calculation the same steps must again be performed by hand and separately instantiated for differing input data.
- Computation is, for the most part, memory less: except in some special cases, partial results must be written down and re-entered when they are to used in completing a calculation.
- Each step of a calculation requires some manual intervention, thus the computation does not proceed independently of human control.
- The technology employed is mechanical not electronic.

The history of calculating machines post-Leibniz can — admittedly with hindsight — be seen as a series of ideas and technological advances that progressively dealt with these lacunae. With the sole exception of the final point, ideas relating to all of these aspects were developed in the 19th century. In this lecture we shall examine the work of, principally, three people — Joseph-Marie Jacquard (1752-1834); Charles Babbage (1791-1871); and George Boole (1815-64) — and their contribution to the development of computational devices.

4.2. ‘Stored Programs’ and Punched Cards — Jacquard’s Loom and its consequences.

If the impetus behind much of the development of calculating machines discussed so far had arisen from numerical computation, the motivation that led to the earliest form of ‘stored program’ was to come from a very different source: the textile industry. We have seen earlier that one of the fundamental aspects of computational systems is the concept of representing information and, although we have not done so explicitly, the application of this idea can be discerned in all of the artefacts that we have examined up to now: in the development of written representations for numeric values and the mechanical parallels that sprung from these. Thus, the alignment of pebbles on an abacus frame, the juxtaposition of moving scales on a slide-rule, and the configuration of cogged gears on the devices of Schickard, Pascal and Leibniz, are all examples of representational techniques that seek to simplify the complex processes underlying arithmetic tasks. There are, however, categories of information, and representations thereof, other than number upon which computational processes can be performed. The weaving technology developed by Joseph-Marie Jacquard in 1801 illustrates one example of such a category.

In consequence of the Industrial Revolution, the late 18th century had witnessed a considerable expansion in the automation of processes that had once been the preserve of small groups of highly skilled workers employed in so-called ‘cottage industries’. The textile industry was one sphere were industrialisation had rendered obsolete such skills. Whereas, prior to the development of mechanical looms and weaving machines, lengths of fabric had to be woven slowly by hand, the advent of powered tools for carrying out this task meant that quantities of fabric could be mass-produced at a far quicker rate than previously, thereby reducing its expense. There was one area, however, where the new machines could not compete with skilled manual workers: in the generation of cloth containing anything other than a plain (or at best extremely simple) woven pattern. The Jacquard Loom provided a solution to this problem so that, with it in use, extremely intricate patterns and pictures could be automatically woven into cloth at much the same rate as a plain length of fabric could be generated. The key idea behind Jacquard's loom was to control the action of the weaving process by interfacing the behaviour of the loom to an encoding of the pattern to be reproduced. In order to do this Jacquard arranged for the pattern to be depicted as a groups of holes 'punched' into a sequence of pasteboard card. Each card contained the same number of rows and columns, the presence or absence of a hole was detected mechanically and used to determine the actions of the loom. By combining a 'tape' of cards together the Jacquard loom was able to weave (and reproduce) patterns of great complexity, e.g. a surviving example is a black and white silk portrait of Jacquard woven under the control of a 10,000 card 'program'.

Jacquard's invention of the punched card is now recognised as important largely because of the influence it had on other developers of computing machinery. One of these, Charles Babbage, will be discussed later in this section. Another significant offshoot resulting from Jacquard's idea is found in the work of Herman Hollerith (1860-1929)

4.2.1. Hollerith's Census Collator

Societies, or more correctly the wielders of power in societies, have long been concerned with accurately assessing statistical data relating to population, i.e. the activity that is called *census taking*. Many ancient sources contain records of contemporary censuses, ranging from Biblical reports (e.g. 2 Samuel 2:1-9, Luke 2:2) to surviving accounts of Roman census taking: in classical Roman government the rôle of Censor[1] was an important public office. There were a number of reasons for carrying out such counts: exact information on the size of the current population giving details of the proportions of men, women and children; numbers in different occupations, and income could be used in assessing tax levies, determining entitlement to representatives in legislative bodies, and in planning future policy.

It is immediately apparent that conducting a full census of even a moderate size population, entails collecting, preparing, and collating an enormous quantity of information. The U.S. Constitution (Article 1, Sect. 3) set in motion the constitutional requirement to hold a decennial census, the first of which commenced on August 2nd 1790. Censuses have been held at the required interval ever since: Britain and other European countries adopted similar practises over a century later. The 1790 census took over nine months to gather and process all of the information involved. Even at this early stage and dealing with a population of about 3.8 million people a problem is apparent: processing the information gathered is costing a considerable amount of time and expense. By the time of the 1860 census the U.S. population had increased to 31.4 million people and it had become necessary to place a limit of 100 questions on the census form. Even with this restriction it was becoming apparent that without any kind of mechanical assistance a point would be reached where the results of the previous census would not have been processed before the statutory requirement to hold the next one came into effect. For the 1870 census, a crude device invented by Charles Seaton was available to assist with the processing task. This, however, was little more than a convenient data entry and output tool.

1) The noun 'censor' in English carries two meanings: that of 'person responsible for conducting a census' and that of 'person responsible for delineating changes to texts, films, etc so that these are in accordance with what is deemed acceptable for public consumption'. This dual meaning arises from the fact that the office of Public Censor in Roman society held both responsibilities.

Hollerith became involved with this problem following the 1880 census, having started work with the Census Office in 1881. At this time he met John Shaw Billings, who had been involved in statistical analysis of the 1880 returns. Hollerith's biographer records that the idea for an improved method for processing census forms came to Billings and Hollerith during dinner at the former's house[2] Hollerith, himself, attributed the idea to Billings reporting their discussion as:

> He said to me there ought to be a machine for doing the purely mechanical work of tabulating population and similar statistics ··· he thought of using cards with the description of the individual shown by notches punched in the edge of the card ···

Billings himself, credited the idea to Hollerith. In any event, for the 1890 census Hollerith had perfected a system for encoding census returns onto punched cards and designed machinery which could process these to tally the totals corresponding to various statistics. He had earlier demonstrated the efficacy of his approach by reorganising record keeping systems in various large institutions. The success of Hollerith's systems led to his ideas being copied by other companies keen to make money from the lucrative contract for census automation. By the time of the 1910 census this erupted in an acrimonious Patent Dispute between Hollerith's company (Tabulating Machine) and a rival organisation controlled by Edward Durand. Hollerith eventually lost the lawsuit after the case had been appealed to the Supreme Court. Nevertheless, Hollerith's contributions to and application of punched cards was a significant step in the development of automatic computing machinery. The format he developed for storing information continued to be used extensively well into the 1960s. Equally significant was the rôle eventually played by his company. After merging or taking over rival concerns Tabulating Machine became the Computing-Tabulating-Recording Company. In 1914 CTR acquired a salesman from NCR — Thomas J. Watson. Watson had taken overall control of CTR within five years of joining them. The last name change took place in 1924 when CTR became International Business Machines or IBM.

4.3. Difference Engines and Analytic Engines — Charles Babbage

Our description of Hollerith's development of automated punched card analyses for the purpose of collating census data has taken this historical review into the early twentieth century. Hollerith's work represents one of the earliest large-scale applications of mechanical methods in a domain where computer systems are now commonplace: that of *data processing*, i.e. the storage, maintenance, and analysis of recorded information. Computer technology has, however, traditionally been associated with another, rather different field: the rapid evaluation of intricate algebraic or arithmetic formulae. Arguably the first serious attempt to realise such a facility, with devices conceptually similar to modern computer components, can be found in the efforts of Charles Babbage (1791-1871) and his co-researcher Augusta Ada Byron, Countess of Lovelace (1816-52). Between them, Babbage and Countess Lovelace greatly extended the functionality and sophistication of calculating devices based on mechanical gears. The engineering concepts and designs underlying this work were principally the contribution of Babbage. These, important though they were, represented no tremendous *conceptual* advance, however; we have already seen that the exploitation of systems of mechanical gears in order to emulate arithmetic processes, was central to the basic calculating devices of Schickard, Pascal, and Leibniz dating from over one hundred years earlier. True, Babbage determined that such mechanisms could be configured to perform much more complicated tasks than the four basic arithmetic operations and, moreover, that the engineering technology of the time

2) Hollerith's biographer, G.D. Austrian, claims that Hollerith wished to make the acquaintance of Billings' daughter: on first meeting her he had attempted to impress her by buying all but one of the lottery tickets that she was selling — naturally the one ticket left unpurchased was the winning one.

allowed an effective demonstration of this potential — from a present day perspective, however, the real significance of Babbage's and Lovelace's work lies other than in mere engineering legerdemain. Thus, it is perhaps rather the case that two innovations, due in no small degree to Countess Lovelace, should be seen as the most important legacy to modern computing left by these two. The innovations in question? Principally the concept of a *stored program*; and, secondly, the suggestion that there is a direct analogue between semiotic manipulation as a facet of computational artifice and semiotic manipulation in the classical mathematical context of algebra. This latter concept is fundamental to the bases on which the mathematical theories of computation are constructed. Before considering Babbage's early work and his joint endeavours with Countess Lovelace in more detail, it is worth briefly reviewing the background of both.

Babbage first became noted as a prolific inventor whose work naturally led him to consider the possibility of building better mechanical computing devices. While his first major idea — the so-called Difference Engine — could only be partially built[3], for almost the last twenty years of his life, Babbage was obsessed with the design and construction of a much more powerful machine — the Analytic Engine. The ambitious scope of this was far beyond the engineering capability of the time[4]. Nevertheless, unlike the Difference Engine, work on which Babbage ceased when the concept of the Analytic Engine occurred to him (since the latter would have encompassed the former) — the Analytic Engine is, arguably, the first real attempt to construct something resembling a modern computer. By the end of his life Babbage had become extremely enbittered by the lack of public and (more importantly as regards financing) government recognition of the significance of the Analytic Engine, a full working design of which is not known to have been completed. In his seventies, Babbage had turned into that rather tiresome character: the 'English Victorian eccentric'. Among his proclivities at this stage was a healthy and fierce dislike of street musicians: upon hearing these he used to race into the road to chase them away from his house. An almost equal intolerance of children was created as a result of local children taunting him by imitating the noise and cries of street-corner musicians outside his home.

The background of Ada, Countess of Lovelace was entirely different from that of Babbage whose family ran a prospering banking concern. Had it not been as a result of the success of one of his earliest calculators, the two might never have met. Augusta Ada Byron was the only daughter of the great English poet Lord Byron, born from a marriage undertaken by Byron in order to quieten the contemporary scandal surrounding his, then considered to be, vigorous and unorthodox love life. Lord Byron died in 1829 (in Greece) when she was only 14 and as the only (legitimate) daughter of an English aristocrat, her life would normally have consisted of the then usual round of so-called upper class society events, followed by marriage to someone of a similar background, followed by subsequent obscurity caring for husband and children (in that order). Countess Lovelace was, however (and fittingly for a daughter of Byron!) an extremely gifted, intelligent and independent-minded woman. She first encountered Babbage at one of the, then popular, society events dedicated to showing-off interesting and amusing new inventions. Here she came across a prototype of the Difference Engine upon which Babbage was working at the time. Fascinated by the device she became intent on meeting its inventor and discussing it with him. From the time they met until her death Babbage and Countess Lovelace were in constant touch with each other, she, in her letters, suggesting and describing, a number of significant ideas for enhancing such artefacts. It is only recently that her contribution to the development of modern

3) Independently a complete, working, though less ambitious machine based on much the same ideas was realised by the Swedish partnership of Georg Scheutz (1785-1873) and his son Edvard (1821-1881).

4) Whether the machine could ever have been built, prior to the advent to electronic devices, is open to question.

computer principles has received the recognition it deserves. It was as a tribute to her work that a newly designed and now widely used programming language was named, in her honour, ADA.

Babbage's interest in computing machines arose while he was checking tables produced for the Royal Astronomical Society, in 1820[5] These tables, containing astronomical data, values of logarithms, trigonometric functions, and various physical constants, formed the basis for the analysis of scientific experiments and for navigation. They had been produced by hand, in some cases the measures given had been compiled over two centuries previously, and thus due to human errors in the calculations and copyists mistakes in transcribing the tables for publication, the standard tables were rife with errors. By the 1820s the standard Government tables for navigational purposes contained well over 1000 known errors and the corrigenda ran to 7 volumes. Even the corrigenda required further corrections. While engaged in the tedious task of checking such tables Babbage realised that much of the calculation required was of a routine and mechanical nature. Thus producing a mechanical device which could generate the tables automatically would have significant benefits: calculating and transcription errors would be eliminated.

Babbage's first important idea in this area was the so-called *Difference Engine*. This was intended to evaluate *polynomials*[6] The term 'difference' came about from the principle used to evaluate such expressions: a mathematical technique called the *Method of Differences,* on which Babbage had carried out important work. By this technique all polynomial evaluation could, in principle, be reduced to addition. In 1822 Babbage presented a paper to the Royal Astronomical Society in which he proposed constructing a machine to generate and print scientific tables. The process of carrying out addition mechanically was well understood by this time and so Babbage was able to demonstrate a machine he had constructed to print tables of squares, cubes, and a single more complicated polynomial. Impressed, the Society supported Babbage's proposal to build a machine that would work to the accuracy of 20 decimal places. Babbage now faced a problem that has bedevilled the development of science and engineering: money. Babbage attempted to obtain Government funding to construct his proposed machine. In 1823, after some debate the Exchequer approved the award of 1500 to meet the costs of developing a large Difference Engine. Babbage planned to take 2-3 years designing and building the machine and estimated that about 5000[7] would be needed. It had been understood by Babbage that such funds as would required in addition to the initial 1500 grant would be forthcoming. Unfortunately for the subsequent history of the Difference Engine the Exchequer's understanding differed from Babbage's.

The Difference Engine was never finished. After 10 years work, involving acrimonious disputes with the Government over financing, arguments with influential scientific figures of the time, allegations of fraud, and the expenditure of 34,000[8] (at least half of which came from Babbage's own resources out of the estate inherited from his father) Babbage ceased work on the Difference Engine. What remains of it is now on display in the Science Museum in London.

5) Babbage's autobiography gives a different version, but this is considered to be unreliable.

6) A polynomial is a function, $f(x)$ which may be written in the form $f(x) = \sum_{k=0}^{n} a_k x^k$ for some choice of constants $a_0, \ldots, a_n$, e.g. $41 + x + x^2$.

7) In 1823 an income of 100 per annum would allow a comfortable standard of living. In modern terms the grant awarded to Babbage was of the order of 750,000; the amount he actually believed was needed would be around 2.5 million at today's costs.

8) About 17 million at current prices.

Undoubtedly the main reason why Babbage did not continue work on the Difference Engine was because he had conceived the idea of a machine which would render it obsolete: the *Analytic Engine*. Babbage's description of the components and function of this machine display extraordinary prescience when considered in the light of modern digital computers:

> The Analytical Engine consists of two parts:
>
> > 1st. The *store* in which all the *variables* to be operated upon, as well as those quantities which have arisen from the result of other operations, are placed.
> >
> > 2nd. The mill in which the quantities to be operated upon are always brought.
>
> algebraical operations to be performed upon given letters, and of certain other modifications depending on the numerical values assigned to those letters.
>
> There are two sets of cards, the first to direct the nature of the operations to be performed — these are called operation cards; the other to direct the particular variables on which these cards are required to operate — these latter are called variable cards.
>
> required to develop, the law of its development must be communicated to it by two sets of cards. When these have been placed, the engine is special for that particular formula. The numerical value of its constants must then be put on the columns of wheels below them, and on setting the Engine in motion it will calculate and print the numerical results of that formula.

What Babbage describes in these excerpts was something remarkably close the the modern computer: a device utilising the elements of Memory, Processing Unit, and program (or formula as Babbage called it). Babbage envisaged the actual power of the machine being provided by steam with its realisation being by mechanical gears. Ada Lovelace was one of the only people to understand and appreciate the significance of Babbage's ideas. She expanded considerable effort on developing the first programs for the planned machine and on documenting its design and logic. Her detailed descriptions are the only clear records left of the proposed mechanics of the machine. Babbage had to develop the machine at his own expense and quickly ran into problems with funding, since the Government refused to support the new machine. Babbage and Lovelace worked extensively on the principles underlying the machine. In 1852, however, Ada Lovelace died, in considerable pain, from cancer of the uterus at the age of 36. Babbage effectively worked alone on the machine until his own death in 1871.

The range and ambitious scope of the Analytical Engine was far beyond the capabilities of the technology available at the time. Mechanical gears and steam power might have been adequate for the limited needs of earlier machines, however, these could not have been manufactured to the precision required to make the Analytical Engine a reality. Nevertheless, as a theoretical concept, the idea of the Analytical Engine and its logical design are of enormous significance. This is the first realisation that, by mechanical means, it might be possible to *program* complicated algorithms. The Analytic Engine had, in principle, all of the important components (Memory, Processor and Input/Output protocol) that are present in modern-day computer systems. For this reason Babbage has a strong claim to be the inventor (even if not the first builder) of the modern computer.

4.4. George Boole (1815-64)

The achievement of the autodidactic English mathematician George Boole may not, at first sight, seem as significant as the concepts moted by Babbage and amplified by Ada Lovelace. Nevertheless, he was responsible for formalising and developing a system which underpins the operation of *every* modern digital computer: the system, named in his honour, of *Boolean algebra*.

Boole was not primarily concerned with the automation of calculating activities. His motivation for developing the system was to assist in expressing and evaluating the soundness of logical propositions. For example, suppose we have a number of assertions *P*, *Q*, *R*, *S*, etc. Boole considered the question of what could be said of assertions made by combining these in various ways, e.g. *P AND Q AND R*, (*P OR S*) *AND* (*Q OR R*). By considering assertions to by **true** or **false**, Boole developed an algebraic calculus to interpret whether composite assertions were **true** or **false** in terms of how they composition was formed (i.e. the combination of AND, OR, etc) and the truth or falsity of the atomic assertions. Boole published his work in the mid 19th Century[9] Its importance with respect to computer design was realised in the 20th century when approaches to constructing digital computer were being investigated. Boole's calculus was instrumental in breaking the tradition of reducing computation to addition: the route that had, in effect, dominated the design of automatic calculating machines from Schickard through to Babbage, since by exploiting Boolean algebra and electrical (or electro-mechanical) *switching components*[10] it became possible to build reasonably reliable systems capable of carrying out complex computing tasks.

4.5. Conclusion

The nineteenth century developments in automatic computation form a bridge between the mechanical methods of early calculating machines and the advent of digital computer systems in the twentieth century. Babbage and Lovelace conceived the idea of constructing a machine that in its composition was similar to the machines present today. That they failed to realise their conception was largely due to the inadequacies of the contemporary engineering technology. Their work, however, delineated the components that were to be adopted in subsequent machines. These, in combination with Boole's formalisation of an algebraic system that was to prove subsequently to be of immense value, were ultimately to lead to the development of electronic computing systems in the following century.

9) *An Investigation of the Laws of Thought*, Dover Publications, Inc., 1854

10) A switching component is one which is always in one of two 'states' (on or off) and can be moved from one to the other on receipt of a suitable stimulus.

Mechanical Aids to Computation and the Development of Algorithms

5. Digital Electronic Computers prior to 1950

5.1. Introduction

We argued, at the conclusion of the previous chapter, that the failure of Babbage to realise a functioning prototype of the Analytic Engine — arguably the first proposal for a programmable 'computer' — was not due to any conceptual lack of insight but was, rather, an inevitable consequence of the inadequacies of contemporary engineering technology. Babbage had sought to construct a programmable calculating tool using the interaction of mechanical gear wheels meshed together as a basis. Undoubtedly the most significant technological insight in the genesis of modern digital computers was the realisation that *binary switching components* provided not only a sufficient basis for the implementation of automated calculators but, indeed, would also be a *necessary* foundation of any machine as ambitious in concept as Babbage's Analytic Engine, i.e. a machine which could:

- remember the results of intermediate calculations;
- automatically repeat a sequence of calculations using *new* data;
- be reconfigured (programmed) to carry out an entirely different sequence of computations.

How do 'binary switching devices' aid in the construction of such machines. Recall that such a switching device can viewed as in Figure 14 below.

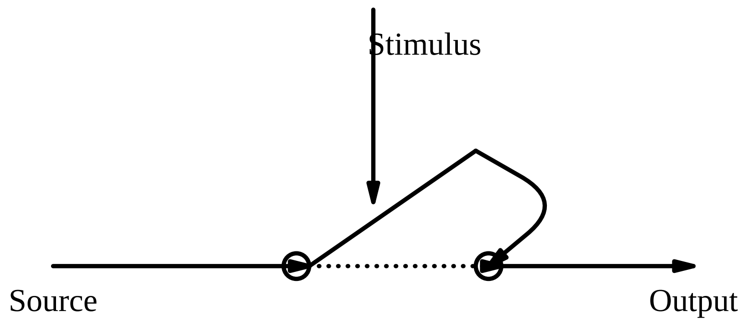

Figure 14: Switching Component

Under a sufficiently large 'stimulus' the switch is closed and (while the stimulus remains sufficiently 'high') a signal can pass from the source to the output. If the stimulus and signal are both electrical and the concept of a 'high enough stimulus' is that of carrying a large enough voltage then such a configuration gives a convention for representing 0 (the voltage on the output is too small to close a switch) and 1 (the voltage on the output is large enough to close a switch). We can thereby represent arbitrary number using a notational system called *binary*, e.g. the number 20 is represented by 5 switches set (in order) to 10100. We can, however, do much more than merely represent numbers: we can also manipulate such representations by using an appropriately powered (e.g. electrical) system of switches. Thus consider the configurations in Figure 16 below.

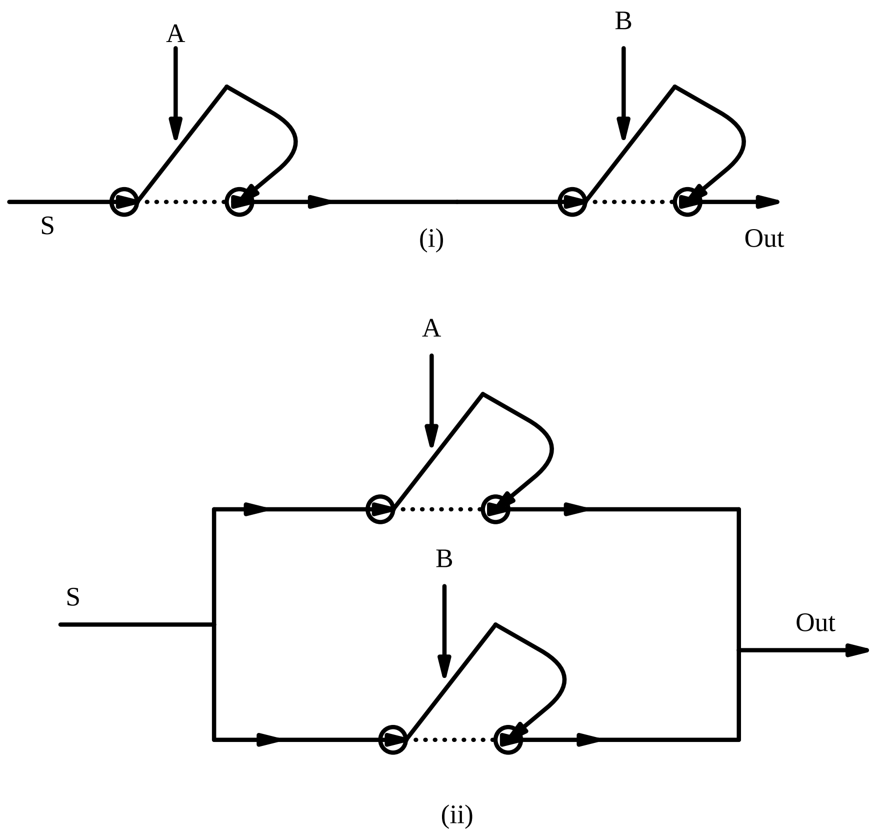

Figure 16: Logical AND and Logical OR circuits

In (i) the output equals ‘1’ if *both A and* equal 1; in (ii) the output equals ‘1’ if *A or B* equal 1. The algebra developed by George Boole and introduced at the end of Chapter 4 provides a mechanism for reducing complex arithmetic operations to networks of switching components representing the operations logical AND, OR, and negation which manipulate binary representations of numbers. Thus, in principle, one can build systems to carry out arithmetic (and other operations) automatically. Using electrically driven switches we can

- represent numbers (using binary notation)
- perform calculations on these (e.g. addition, multiplication and more complex functions) by providing an appropriate network of switching components.
- Store and execute programs by expressing programming tasks as a sequence of basic operations on data and relaising these operations by switching networks.

One might argue that all of these could (in principle) be done mechanically, however, the use of gearing mechanisms does not enjoy many of the advantages of systems built as outlined above:

- Binary systems have only two values to represent; so such systems would be less likely to suffer from failures.
- Using electricity as the source of power allows for much faster systems.

In summary the realisation that binary notation and Boolean algebra were better foundations for building automatic calculating machines than mechanical gearing systems proved to be crucial in rendering programmable systems feasible: the computational representation of a quantity is more flexible; and the processing of such representations can be described in a concise algebra and therefore replicated. All modern computing systems can, at the most basic level of operations, be described in terms of complex networks of binary switching components.

The first such ‘switching components’ were electro-mechanical *relay-contact* switches. These comprised two metal contacts connected by a hinge; on receipt of a sufficiently high voltage the two metal contact closed together to all a current to pass through them.

5.2. Development of First Electrical Machines in Germany (1936-44) — Zuse and Schreyer

Probably the first person to appreciate the importance of electrical components and Boolean algebra, as concepts that could be used to build a general-purpose computing tool, was the German engineer Konrad Zuse. Zuse had studied engineering in Berlin and in the course of his training had to learn about solutions to simultaneous linear equations. In such equations one is given a set of k identities involving k variables, e.g.

$$\begin{aligned} 3x + 2y + 3z &= 18 \\ 5x + 3y + 2z &= 17 \\ 4x + 6y + 7z &= 37 \end{aligned}$$

and one is required to find (usually, unique) values for the k variables which simultaneously satisfy the k identities. In the example, the (unique) solution is $x = 1$, $y = 2$, and $z = 3$. Such systems commonly arise in engineering applications as a model for representing trade-offs between different criteria in building structures, e.g. weight, strength, rigidity etc of materials used. Such systems are not mathematically difficult to solve, however, considerable amounts of calculation are involved. The traditional approach (known as Gaussian elimination) becomes extremely laborious when more then four identities are to be dealt with. In engineering applications such as architecture systems involving several hundred equations often arose and constructing feasible solutions to such systems would require months of work by a number of teams. Zuse conceived the idea of carrying out the required calculations by a machine while he was a student in 1934. Zuse recognised that such a machine would need to meet three basic criteria.

- The machine would have to be as general as possible, i.e. not only deal with basic arithmetic or even simultaneous equations but with any suitably formulated series of calculations. To this end Zuse designed the machine to have a very similar structure to that proposed for the Analytic Engine by Babbage:[1] thus there was to be a memory for recording data, an arithmetic unit for carrying out calculations, a *control* unit governing which operations were to be performed and the flow of data upon which they operated, a program unit for entering instructions and data, and finally, an output unit for printing results.
- Binary representations of data would be used.
- The operations of the machine were described and implemented using a formulation of Boolean algebra.

In 1936, while working at the Henschel Aircraft Company in Berlin, Zuse started to build his first machine: the Z1. This was undertaken entirely privately at his own expense. This was completed in 1938 but was not really satisfactory, since the arithmetic unit did not function correctly. Zuse was still using a basically mechanical system to carry out all of the calculations although the machine represented a significant advance on Babbage since the use of

1) This design was proposed without any knowledge of Babbage’s work, which Zuse did not encounter until 1939.

binary avoided many of the traditional engineering problems. The important breakthrough came with his second machine, the Z2, in which Zuse dispensed with the mechanical plates of his prototype machine (except for the memory design which had worked successfully in the Z1) and substituted electromechanical relays instead. This had two effects: the machine worked; and it operated (for the time) extremely quickly. The suggestion that electro-mechanical relay switches be used had come from Zuse's colleague, Helmut Schreyer, an electrical engineer. Schreyer, in fact, proposed a far more radical solution to the problems with the mechanical aspects of the arithmetic unit: that of using a purely electronic device called a thermionic valve (similar to the valves found in old wireless sets). This could be used both as a memory device and as a switching mechanism. Zuse, however, felt more comfortable using the electromechanical relay devices. Schreyer's idea was eventually taken up in the early 1950s by prototype British and U.S. computers. The final version of the Z2 was completed in 1939. Zuse was conscripted at the outbreak of the Second World War eventually ending up working at the Aerodynamics Research Institute. Schreyer, who was not conscripted, continued on his own idea of building a machine using purely electronic components, in this case thermionic valve. Both Zuse and Schreyer were attempting to build a general-purpose computer. Both faced the same problem: a total lack of official interest in their work. In 1942, Schreyer submitted a research proposal describing his machine and its potential to the German High Command: it was rejected as being of no importance. Zuse was only able to work on his next machine, the Z3, by presenting it as a *special-purpose* calculator to solve a problem that was hindering progress on aircraft design. In fact Zuse and the team working with him had always intended to build a general-purpose machine. The Z3, which was the first ever working general-purpose programmable computer was completed in December 1941. By modern standards it was not a large machine: its memory could hold 64 numbers each containing 22 binary digits (or *bits*). Multiplication could be performed in under 5 seconds and in addition to the basic arithmetic functions the machine could be programmed to carry out more complex tasks such as the calculation of square roots. The machine was programmed by entering the instructions to be executed into the memory. This input was done by manually setting the memory content. The machine which comprised a tape reader, operators console and functional units containing about two and a half thousand relays was built for very little cost, $\cap$ about 1000[2] One of the important innovations of the Z3 was its practice of treating data in fixed length *words* (of 22 bits long) so that all input numbers were translated into this form for processing. The practise of designing machines in terms of some fixed word size continues in machines built today were 32 bit and 64 bits are common sizes.

Zuse went on to built a larger machine, the Z4, which after being relocated a number of times during its construction to avoid bombing attacks, was completed in the late 1940s. It was subsequently placed in a technical institute in Zurich in 1950 were for a number of years it was the only available powerful calculating tool in mainland Europe. The Z3 no longer exists: it was destroyed by an air-raid on Berlin in April 1945.

5.3. U.S. Research — Aiken, Atanasoff, Eckert, Mauchly and Von Neumann

The work of Zuse and Schreyer was unknown outside Germany at the time. Most of the innovations made by the two were independently developed in the United States (and Britain) in the 1940s.

The history of electronic digital computer development in the United States over this period is extremely complex and still regarded as a matter of some controversy. The results of this confusion were the patent disputes involving the company representing two of the significant pioneers in this work — J. Presper Eckert (1929-) and John Mauchly (1907-1980).

2) Approximately 20,000 at current prices.

The disputes started in 1952: Eckert and Mauchly finally lost their patent case (under which they claimed to have ‘invented’ the first ‘modern electronic computer’) in the Supreme Court on October 19th 1973.

In this section we only have space to quickly review the main contributors to the development of the important early machines in the U.S. and to outline the work of the individuals responsible for them. As well as Eckert and Mauchly two other figures are significant: John Atanasoff (1904-)and John von Neumann (1903-1957). That Eckert and Mauchly were not granted legal recognition of ‘their’ invention may, in some part, be attributed to claims made on behalf of these two.

Independently of these four, however, Howard Aiken had in 1937 at Harvard University proposed some ideas concerning the structure of what he considered an ideal computer: one that could represent negative as well as positive numbers, perform all standard arithmetic operations, carry out sequences of operations. Aiken’s idea combined elements of Babbage’s work but with the additional concept of using the punched-card representation for data as delineated by Hollerith. Aiken was able to interest I.B.M. (which Hollerith’s original company had become) into supporting his proposals. The machine, subsequently called the Mark 1, was built between 1939 and 1944. As with Zuse’s Z machines it employed electromechanical relays, and comprised three-quarter of a million parts. Its speed at individual arithmetic functions was similar to that of the Z3, and when finished it was capable of completing six months manual calculation in less than a single day’s working.

The Mark 1 was already out-of-date by the time it became operational in 1944. It had been superseded by the machine being developed at the Moore School of Pennsylvania State University. Before this project started an earlier machine had been designed by John Atanasoff and Clifford Berry. Their machine was to be based on valve devices, which although favoured by Schreyer had been ignored by Zuse who preferred to use relays. Atanasoff had succeeded in constructing an electronic calculator that could perform simple arithmetic. He and Berry then set about scaling this simple machine into a more complex computer. Their design provided for a novel storage medium on magnetic drums, 50 bit binary arithmetic, and input on punched cards. A number of components, including arithmetic units were built and later studies of the full design (made in the 1960s) concluded that the proposed machine would have worked as a special-purpose calculating machine. The machine, however, was never completed as both Berry and Atanasoff abandoned it to work for the U.S. military at the onset of the Second World War. The subsequent recognition of Atanasoff’s work (over 20 years after it was done) has led to the Atanasoff-Berry Computer (or ABC) as justifying claims being made for Atanasoff having invented the first general-purpose electronic computer. These claims are defended by Iowa State University were the two worked and were accepted in the Supreme Court ruling invalidating the patent suit filed on behalf of Eckert and Mauchly. While it is certainly the case that the design of the ABC was technically sound, nevertheless the machine was never built, and on this ground Eckert and Mauchly’s claim to be the first U.S. developers has some foundation.

Like Shreyer and Atanasoff, Mauchly was interested in employing valves as the basic switching component for constructing an electronic calculating machine. He was interested in the problem of weather forecasting and the possibility of doing this by a machine. Mauchly met Atanasoff after presenting a paper on approach to tackling this problem. Mauchly’s mechanism for doing this, however, performed its analyses on analogue electronic signals. Atanasoff and Mauchly disagree about the outcome of their discussions. Nevertheless, Mauchly subsequently became concerned with working on a digital electronic machine, based on valves. Eckert and Mauchly came together while the former was studying at the Moore School. Eventually, in May 1943, they succeeded in persuading the U.S. Government to finance the construction of a electronic computer. This was to become known as ENIAC: Electronic Numerical Integrator and Calculator. The initial budget for the machine was $61,700. It was completed in May 1944 and has a strong claim to be the first ever general-purpose electronic machine. The total cost of the project by the time a demonstration was held in February 1945 was almost half a million dollars. The final machine had two noticeable features: it was extremely large (the dimensions were $100 \times 10 \times 3$ feet, weight of 30 tons, over 100,000 components, and occupying an area of 1800 square feet); it was also far faster than anything that has been built previously, being able to multiply in under 3/1000 second. The publicity demonstration in 1945 involved calculations of tables of trigonometric functions and powering of large numbers; all of these calculations were performed in a matter of seconds.

The final figure of importance in our review of U.S. developments is John von Neumann. Although von Neumann made a number of technical improvements to the initial design of ENIAC and was involved in setting up the successor project: EDVAC which would be a programmable machine, he is principally of importance for his theoretical work on computer structures (or *architecture*) and his r:ˆole in convincing sceptical authorities of the significance of the potential provided. He has (mistakenly) been credited as the inventor of the digital computer as a result of circulating the first draft description of its operation. Among his many important contributions to Computer Science is his development of a structural description of how computers can be organised: this is now known as the von Neumann model and in effect describes the relationship between memory, program, and control units. In addition, von Neumann became involved with the design and construction of one machine — the IAS — which could be used to demonstrate his ideas, conceived in 1946 this machine was not completed until 1950. Nevertheless, von Neumann came to be regarded in the U.S. as the principal authority on computer design.

5.4. Developments in the U.K. — COLOSSUS to EDSAC

Zuse and Schreyer in trying to develop their ideas for a general-purpose computer met with apathy when seeking support from the German High Command. Some military historians[3) have claimed that the failure of the German military intelligence to appreciate the potential uses of such a machine, and thereby divert resources from other research, was a major strategic error. Clearly, it is impossible to say for certain if this was indeed the case. Nevertheless if neglecting to pursue computer development was a technical error it was not one that was also made by British and U.S. research strategists. Much of the impetus behind developments in the U.S. arose from the need for the army to be able to calculate trajectories of projectiles quickly and accurately. In the U.K. the main motivating factor in the development of computers for military use came about through another application: that of deciphering coded intercepted messages.

In 1938 the British Intelligence service managed to obtain a complete working description of the German *ENIGMA* machine from Richard Lewinski, a Polish Jew who had been employed at the factory where these machines were built. Lewinski had been dismissed from his post on account of his religion. The ENIGMA machine was a coding device: by setting up a secret key, messages typed on it would appear in an encrypted form; the receiver of the message who knew the key could then decipher its actual content. ENIGMA was the mechanism by which all command orders and strategic decisions were communicated through the High Command to field officers. As such the encryption device (and, of course, the rota for setting keys) was carefully guarded. Lewinski had been able to pass on a precise description of the machine since, possessing a formidable memory, he had remembered exact details of

3) e.g. Mark Arnold-Forster

its construction and operation while employed where it was built. In principle, once the British Intelligence service understood the workings of ENIGMA they would have complete knowledge of German military operations.

The machine, however, was not enough in itself. Having intercepted a message it was still necessary to know the keyword that had been used to encode it and codes were changed three times a day. Knowledge of the internal working of the machine greatly reduced the range of possible keys consistent with a particular intercepted text, but typically there would be an enormous number of possible keys left to test. The frequency of code changes and the labour involved in testing a possible key meant that the task of decoding messages clearly needed some degree of non-human intervention. It was to assist in this that early in the Second World War Project ULTRA was started at Bletchley Park. The aim of this project was to construct a machine that would quickly identify code keys so that messages could be decrypted. The most important figure involved with the research at Bletchley Park was the English mathematician Alan Mathieson Turing (1912-1954). Turing was to become a significant pioneer in computer development and popularisation. In 1936 he had addressed the question of whether it was possible to specify algorithms (i.e. programs) for every conceivable function and demonstrated that a particular problem — that of deciding whether an algorithm came to halt on a given input[4] could not, in general, be solved. Turing was also one of the first people to consider the question of whether machines were capable of ‘intelligent’ behaviour and so a founder of the Computer Science discipline of *Artificial Intelligence*. He proposed a method, now called *The Turing Test*, for determining if a machine was acting in an ‘intelligent’ way: namely, that a human observer monitoring the responses to questions from the machine and another human participant would be unable to distinguish which set of responses came from the machine and which from the human.

Researchers at Bletchley Park initially built a number of small relay-based machines to process potential keys. While these devices were of considerable help in decoding it was eventually decided to build a valve based machine to automate fully the decryption process. Construction of this machine, called COLOSSUS, started in January 1943 and was completed by December of the same year. There is a strong case for regarding COLOSSUS as the first working electronic computer[5] despite the fact that it was limited to carrying out a special-purpose task. COLOSSUS comprised 1800 valves and could read 5000 characters per second via a paper tape reader. By the end of 1944 a larger machine, called Mark II, had been built in addition.

British engineers were among the first to develop and enrich the ideas that had produced the ENIAC computer in the U.S. In July 1945, Douglas Hartree had visited and talked with a number of people who had worked on this machine (despite the fact that its operation was technically classified information owned by the U.S. Military). Hartree, on his return to Britain, tried to encourage British development of similar machines. As a consequence a number of institutions started work on such projects. At Cambridge University, a team coordinated by Maurice Wilkes built the EDSAC computer. This was the first stored-program computer and is the closest of machines considered so far to modern machines. The EDSAC led to two significant innovations: it was the basis of the world’s first user Computing Service; and user programs were coded in an *assembler language*. Since machines operated on binary codes a description of the program instructions has to be supplied in this form. This, however, is difficult to do since it is very easy to enter part of an instruction wrongly. In order

4) Now known as, *The Halting Program*.

5) The British work at Bletchley Park was not a factor considered in the Eckert-Mauchly patent dispute in the U.S. since the existence of COLOSSUS and documents relating to it were classified under the Official Secrets Act in the U.K. Details about this work were not publicly released until 1976.

to overcome this problem, simple English language mnemonics are used to write the program and this is then translated into the equivalent binary patterns understood by the machine.

Other British computers followed from the work at Cambridge: Wilkes assisted the Lyons Company in designing a machine (LEO or Lyons Electronic Office) which handled their account and office transaction work. LEO was handling clerical tasks by the end of 1951. At Manchester University the first of a long series of machines, the Manchester Mark I, was built between 1947 and 1949 and followed by the Mark II in 1951. In February 1946, Alan Turing, had put forward a complete design of a stored program computer — the Automatic Computing Engine or ACE — which it was envisaged would provide a national computing facility. Unable to interest Wilkes at Cambridge or Williams at Manchester (both of whom disliked Turing), Turing developed his design at the National Physical Laboratory in Teddington. When Turing left the NPL in 1947 a new design based on Turing's but containing a number of fundamental new ideas was developed under the name of Pilot ACE. Although the Pilot ACE was working by the middle of 1950 it did not become fully operational until 1952. Turing died in 1954[6] without seeing all of his ideas for the ACE realised.

5.5. Conclusion

The first half of the twentieth century was to see the development of computer systems in the form that they are common today. Dispensing with solely mechanical emulations of arithmetic processes and moving to electrically based switching components and the formalisms of binary and Boolean algebra, allowed fast and fairly reliable calculating machines to be built. All of the major *technological* improvements in computer construction since this innovation have come about as a result of the engineering of better (smaller, faster, and more reliable) binary switching devices. By the middle 1950s large machines of unprecedented size and speed were operational at many U.K. institutions and several U.S. sites. Despite this progress, however, some significant problems remained concerning the ease of using these machines. Although Wilkes' team had originated the idea of assembly languages, coding of programs was still a time-consuming and error prone activity. In the final chapter of these notes we shall examine what developments took place from the late 1950s onward which would render the task of programming computers a far simpler task.

6) Of cyanide poisoning. The inquest subsequently ruled that Turing had committed suicide, citing as motive Turing's impending prosecution for homosexual activity. Turing's relatives, however, have always maintained that his death was accidental: given Turing's eccentric habit of mixing lethal chemicals together to see what the outcome would be there is some degree of justification for their claim.

Mechanical Aids to Computation and the Development of Algorithms

6. Making computing easier: Programming Languages

6.1. Introduction

In the early 1950s even if one had access to a computer facility, actually using this to carry out a specific calculation would often be an extremely frustrating activity. There were two reasons for this: technology; and the problems in describing the task to be performed in a manner that the computer system could act upon. The technological problem arose because the machines were built using several thousand valves. Valves require significant amounts of electrical power, when they are operating they generate heat, when they become too hot they burn-out and have to be replaced (much as an electric bulb does when supplied with too strong a current). Given the size and expense of contemporary systems and the fact that only one calculation series at a time could be run, in order to perform some calculation it was necessary to reserve a time slot in which one had sole access to the facility. It was often the case, however, that having set up the 'program' the computer system immediately ceased to function because some of its valves burnt out. This technological drawback was overcome within a few years of the discovery of the transistor. The TRADIC, built in 1954, was the first computer to employ transistors as replacements for valves. Although over-heating still posed (and indeed continue to pose) a problem, transistors were much more reliable. They also led to far faster machines being possible. Almost all of the subsequent development of more reliable 'hardware' and faster computer systems came about as a consequence of improved transistor technology and device physics: from the early transistor machines of the mid 1950s, through to machines using LSI components (one which a logical AND gate could be placed) in the 1960s, up to the VLSI devices of the present-day. The last are of such complexity that a computer processor, of far greater power than any of the 1950s machines, can be built on a silicon chip no larger than a fingernail (as opposed to the space of several gymnasia taken by machines like the Whirlwind 1 at M.I.T.).

Improvements in fabricating switching components made large computer systems less prone to hardware failure, but they did not render the task of implementing a series of calculations any less difficult. Recall that a key development in the realisation of automatic calculating machines was the introduction of binary representations as the fundamental items operated upon. John von Neumann, in formulating the structural organisation of stored program computers, had shown how a string of binary digits could be used to encode both instructions and data. Following this approach a machine that operated on, say 16 bit *words*, the memory locations that held the program would have the instructions interpreted as follows: the first few bits (4 for example) would indicate a particular operation (ADD, STORE, LOAD etc) and the remaining bits (12 in this case) would indicate where the data for the operation was stored in memory. For a such a program to be executable by the computer, however, the binary pattern corresponding to each individual instruction would have to be entered into the memory. A typical application program for a complex scientific calculation might break down into 200 or more such instructions and so to carry out the calculation 3200 0s and 1s would have to be produced and loaded into memory. A single error would render the program useless. Even though entry can be facilitated by paper tape or punched cards, the task of actually *generating* the correct code offers massive potential for error when it is done manually.

Wilkes' invention of *assembly language* coding, as described at the end of the last chapter, removed some of the possibility last chapter, removed some of the possibility for transcription error, even this was at such a low level then coding complex applications was a demanding and time-consuming task. It was in this context that the first *high-level programming language* were developed. The difference between the assembly level and high-level languages were that in the former a single line of the program would typically translate to a single machine instruction, whereas a single statement in a high-level language might require several machine instructions. Of course such programs would have to be transformed (*compiled*) into a representation that could be executed by a computer. Thus a major problem faced by the developers of the first such languages was the construction of *compilers* to carry out this task automatically. In the final part of the course we shall, briefly, review the main contributions made within the field of high-level language development.

6.2. Scientific Computing Applications — FORTRAN

A, perhaps surprising, property of the several of the early programming languages has been their durability. The language FORTRAN (an acronym for *FORmula TRANslation*) is one of the best examples of this phenomenon. Developed in the mid-1950s, FORTRAN was intended for use in scientific and numerical computing applications. As such the analysis of experimental data on computer systems could, in principle, be done by programs designed by the scientists and engineers who had gathered the data. The designers of FORTRAN were among the first to seriously address the principle objection to high-level languages that had been raised: although producing programs that were less error-prone became easier, this fact was counterbalanced by the inefficiency of early compilers. Thus a skilled assembly level programmer could produce code for an application that was more compact and faster than that generated by a compiler. A number of technical decisions were taken in the provision of FORTRAN capabilities that would make it possible to construct FORTRAN compilers which produced machine executable code comparable to that of the best human programmers. The most important of these was that the amount of memory to be used during the running of a program could be exactly determined in advance, using the description of the program alone[1]. As a result of this, various *optimisations* could be made to the code produced to enhance the running of the program.

FORTRAN was promoted by I.B.M. and continues to be extensively used today. The original version of the language has undergone several revisions (on average a new version appears about once every seven years). A key criterion that has to be satisfied by revised versions of the language explains, in part, why FORTRAN has continued to survive as a language today: no revisions to the language are accepted if as a result older FORTRAN programs would cease to compile. AS a consequence of this policy, in principle, FORTRAN programs from 1955 can still be compiled using the latest version of the language. The concept of *upward compatibility*, as this policy is called, is (commercially) important in ensuring that users of a system can still continue to run old programs when new versions of the language are released.

6.3. Commercial Data Processing — COBOL

We saw, in the opening lecture, that one of the areas in which computer systems have become predominant is that of record maintenance. The field of processing records stored on computer systems, e.g. computing payroll statistics and figures for corporations employing large numbers of people, is known as (commercial) *data processing*. The Lyons Electronic

1) This feature of the FORTRAN language continued for almost 25 years, through successive revisions of the language.

Office (LEO), mentioned at the end of the last chapter, and the various census collation devices discussed, are both examples of data processing applications. FORTRAN went some way to addressing the requirements of scientific and engineering applications. Although it could have been used as a method of writing typical data processing applications programs it was not really suitable for such tasks. Various primitives desirable for data processing tasks were missing (since they were not needed in numerical applications), and the style of FORTRAN programs (which read as a series of mathematical formulae) was alien to those working in the fields of record maintenance and processing.

COBOL (*COmmon Business Oriented Language*) was designed at the end of the 1950s by Admiral Grace Hopper in response to a commission from the U.S. Admiralty. Like FORTRAN, it has proved extremely durable and most large-scale data processing systems today are still realised in COBOL. One of the design aims underlying COBOL is that programs written in it should be readable, even by non-computer professionals. So whereas FORTRAN, in describing program instructions, employs standard mathematical symbols and operators, in contrast, COBOL programs attempt to mimic natural language descriptions. In the earliest versions of the language no mathematical symbols were used and arithmetic operators were specified using the equivalent English word. This, amongst other features of the language, made COBOL programs appear to be extremely verbose[2]. Thus the simple statement $A = C + D$ in FORTRAN becomes *ADD C TO D GIVING A* in COBOL. The belief that by substituting English for mathematical symbols programs would become more understandable and thereby easier to write turned out, however, to be fallacious. The reason for this, which is obvious with hindsight but was not apparent at the time, is that what makes designing a computer program for a specific task difficult to do is the process of specifying, structuring, and ordering the activities to be carried out at a fine enough level of description, i.e. the difficulty in programming is constructing the appropriate *algorithmic* process. Once this has been done it is usually a relatively easy task to translate the algorithm steps into any programming formalism. Once this fact had been recognised, an important consequence was that more research effort was concentrated on methodologies for designing programs and algorithms rather than on superficial attempts to make such programs ‘readable’ or ‘comprehensible’. The study of programming methodologies ultimately led to the theoretical disciplines of Programming Language Semantics, Formal Specification, and Formal Verification that were mentioned, briefly in the opening lecture.

As with FORTRAN, COBOL has undergone several revisions since its initial design. In the case of non-academic activity, COBOL has been since the early 1960s probably the most widely used language in commercial applications. This is likely to continue to be the case for the foreseeable future despite the growing use of other languages.

6.4. Artificial Intelligence Applications — LISP

LISP (*List Processing*) was developed by John McCarthy and a team of research students at M.I.T. in 1960, originally to provide a realisation of what is known known as the paradigm of ‘functional programming’. It subsequently became extremely popular and widely used (in academic environments) as a language for programming Artificial Intelligence systems. While it is still used today for such problems it has been replaced in some specific areas of this field by subsequent developments in programming language theory: most notably by the logic programming language PROLOG and by so-called ‘object-orientated’ languages such as

2) There is an apocryphal story to the effect that COBOL was designed in this way so that directors and company chairmen could understand and modify the payroll programs run by their corporations and that English was necessary since such people would be incapable of understanding complex mathematical symbols such as $+$.

SMALLTALK. If readability of programs played a significant rôle in the design of COBOL, and was at least superficially addressed by the designers of FORTRAN, anyone with experience of LISP programming will be aware that no such considerations bothered McCarthy and his team in designing LISP. Underlying LISP is an algebraic formalism for describing effective algorithms developed by the great U.S. logician Alonzo Church (1903-) and called the λ-calculus. Thus LISP in its most basic form[3] has a precisely defined mathematical semantics and as a consequence. it is, in principle, possible to prove precise properties of such programs. LISP is also of historical interest as being one of the first languages to use extensively an algorithm control structure that was (deliberately) not provided in FORTRAN or COBOL: that of *recursion*. In simple terms this is the action of describing an algorithm in terms of itself, e.g. the mathematical function '*n factorial*' (denoted $n!$) defined as the result of multiplying the first n natural numbers can be defined recursively as $n! = 1$ (if $n = 1$) $n! = n \times (n-1)!$ (if $n > 1$). This facility is useful in describing approaches to various problems in Artificial Intelligence applications such as automated theorem-proving and game-playing systems. A number of early, apparently successful A.I. systems, were built in the 1960s and 1970s using LISP including Samuel's 'checker playing program'; the 'conversational' system ELIZA, and an early natural language system *SHRDLU* built by Winograd in 1972. LISP was also used in robotics and computer vision systems.

6.5. Algorithmic Languages — ALGOL60 and ALGOL68

The three languages we have examined above were all developed in the U.S. and are all still very much in use today. The reasons for this vary: FORTRAN and COBOL were the first languages of their kind available and thus were taken up quickly so that, even when technically superior methods arrived, there was a reluctance to move away from what had become a familiar idiom. A second very important reason for the continuing survival of these two is the fact that they were heavily promoted by U.S. computer companies. In particular, I.B.M. has played a significant part not only in developing new versions of FORTRAN but also in promoting it to run on their systems[4]. In the same way COBOL has survived largely because of the considerable investment put into building commercial data processing systems that use COBOL. There is little point in promoting or building a 'better' language since, in order for it to be adopted, organisations would be faced with the task of reprogramming their complete system.[5] LISP, although largely restricted to academic and research environments, continued to be used partly because of its theoretical interest but largely because of the applications in A.I. that were developed from it. In the latter field there was no serious rival to it as a general-purpose A.I. applications language until the development of PROLOG.

ALGOL60 and its successor ALGOL68 are now (effectively) extinct languages. Nevertheless ALGOL60 was a landmark in the development of programming language theory. Its disappearance can be entirely attributed to the failure of industrial concerns to adopt it, despite the fact that it was extensively taught at European and U.S. universities in the 1960s.

3) All widely used implementations of this language greatly extend the functionality of the language defined by McCarthy, which is now known as 'pure' LISP

4) It should be noted that I.B.M. FORTRAN compilers were until the demise of large-scale 'mainframe computers' far ahead of alternative versions in terms of their speed of operation and the efficiency of the machine code generated. The FORTRAN 'H' compiler developed by I.B.M. for their 370 series computers is still regarded as one of the best optimising compilers ever designed.

5) This, of course, would be a problem when an old computer system was replaced. One of the primary reasons for I.B.M's dominant position in the computing industry, until very recently, was the fact that when a concern had acquired I.B.M. equipment it was easiest to replace it with a new I.B.M. machine. Moving to a rival company would have entailed extensive rewriting of systems programs. The widespread industrial usage of COBOL and FORTRAN also meant that at least one (often both) of these languages would be covered in undergraduate Computer Science degree programmes.

Its chief importance now is in the innovations that continue to influence the design of programming languages. The effects of these are apparent in languages such as Niklaus Wirth's PASCAL (which now occupies the position formerly held by ALGOL60 as the language used in degree programmes); the subsequent revisions of FORTRAN; and the U.S. Department of Defense language ADA.

ALGOL60 was designed by an international committee working between 1958 and 1963. It was intended to be a general-purpose language but with a particular emphasis on scientific and numerical applications. In this respect it improved upon FORTRAN in two ways: firstly, ALGOL60 provided facilities to represent and manipulate complex structures inside a program; and, secondly, like LISP, it provided recursion as a control mechanism. Until the advent of PASCAL, the notational style of ALGOL60 was accepted as a standard way of describing complex algorithms and a number of textbooks on the subject of algorithm design employed an ALGOL derived notation for illustrating programs. An important theoretical innovation was the manner in which valid statements in an ALGOL program were described: a system called Backus-Naur Form (or BNF) after two of the committee who worked on the design. In simple terms this described how *syntactically correct* statements could be recognised by prescribing 'grammatical rules'. This formalism was subsequently adopted in describing new programming languages as well as revisions to existing ones.

ALGOL68, the successor to ALGOL60, was also designed by an international committee and was intended to remove some of the weaknesses that had been identified with the original ALGOL60 language and to extend its functionality. The final language posed considerable problems: although it was no more difficult to construct programs in it, the design and implementation of compilers to translate such programs into a machine understandable form was a major task. The first two implementations of such compilers were both carried out in the U.K: one at the R.S.R.E (Royal Signals and Radar Establishment); and the second at the University of Liverpool. ALGOL68 continued to be taught at Liverpool up until 1987. The investment of effort required to construct compilers was one of the factors in ALGOL68's failure to become established, despite its considerable merits as a language.

6.6. Conclusion

From 1954 onwards the significant difficulty facing users of computer systems was not the unreliability of the electrical devices but the task of actually describing what operations were to be performed in a manner that could be executed by the machine. While assembly and low-level languages went some way to addressing this, such methods were mainly in the provenance of computer specialists rather than the individuals who wished to use computers to solve applications problems. As a result a number of 'high-level' programming languages began to appear from 1956 onwards. Some, like FORTRAN, were specifically tailored to applications in science and engineering; others, such as COBOL, were intended for the needs of data processing and record maintenance systems. The effect of such languages was to make computers accessible to individuals who did not have a detailed technical knowledge of the actual computer operation.