Maurice V. Wilkes and the Electronic Delay Storage Automatic Calculator

Christian Luijten and Sjoerd Simons

November 16, 2007

Christian Luijten and Sjoerd Simons
History of Computing (2R930)
Department of Mathematics & Computer Science
Technische Universiteit Eindhoven

Contents

1 Introduction

One of the important events for computing in the first years after the Second World War was the development of many different computers.

In the early days of electronic computers there were many developments in parallel, all introducing concepts which we now think belong to the definition of a computer. It can therefore only be said that a certain computer was the first to employ some concept and not that it is in fact 'the first computer'.

This document will discuss an important milestone, the construction of the Electronic Delay Storage Automatic Calculator (EDSAC) in Cambridge, UK; one of the first computers implementing the Von Neumann architecture[1], second only to the Manchester Small-Scale Experimental Machine (SSEM), but the first to be able to solve real problems as the SSEM had only a very limited amount of memory and a small instruction set. EDSAC ran its initial program on May 6, 1949.

We will first introduce the person Maurice Wilkes in Section 2, under whose supervision the EDSAC was built, after which the construction of the computer is discussed in Section 3. Section 4 deals with the way of programming the EDSAC, Section 5 with the preparation and running of programs.

In Section 6 some notable applications are briefly mentioned, Section 7 gives some of the other computers in the time and how they influenced eachother. Section 8 shows the direct influence the EDSAC had on later computer development.

This paper was written for the course History of Computing as teached at the Eindhoven University of Technology in the academic year 2007/2008.

[1] *from Wikipedia:* The von Neumann architecture is a computer design model that uses a processing unit and a single separate storage structure to hold both instructions and data.

2 About Maurice Wilkes

The EDSAC was constructed by a team led by Maurice Vincent Wilkes (now Sir Maurice). To understand why he built the machine and why he built it the way he did, we need to know a bit about his history. We will briefly cover his younger years before the Second World War in which he shows great interest in radio technology. During the War he constructs radar towers for the British Military.

After the war he travels to the United States where he is inspired by researchers such as John von Neumann, something we'll recognize in his work on the EDSAC which he started when he came back from America.

2.1 Early years

Maurice Vincent Wilkes was born on June 26, 1913 in Dudley in Staffordshire county. Already in main school he knew that he "would become a scientist or an engineer"[4].

The family got their first "wireless set" (a radio we would call it today) in 1922, two years after broadcasting had started in the UK. Wilkes, aged nine, "immediately began to make crystal sets and to save up for [his] first valve"[4] to build his own receiver. In 1931 he passed his Morse code test and could start *transmitting* radio, too.

Also in 1931, Wilkes' father decided to send him to Cambridge, against the headmaster's advice. After initial strugglings in his first year he caught on and took various courses on mathematics and engineering. He found a group of radio enthousiasts and took up his old hobby in the University's Wireless Society.

After his B.A. graduation in 1934, he joined the Cavendish radio group and started work on the propagation of very long radio waves. Here he maintained and also built mechanical differential analysers. In 1938 he receives his M.A. degree.

Maurice Wilkes was called for military service in 1939. The group from the Cavendish Laboratory is appointed to construct and work on a chain of radar stations around the south coast of Britain for detecting German U-boots.

2.2 After the War

When the war was over, Wilkes realised from the work he had done during the war the skills and patience needed to operate and the limitations of the then current differential analysers and other analogue calculators. He found that the time had come to "construct a demonstration machine which will perform the operations of arithmetic at reasonable speed and with as great a degree of

simplicity as possible"[4]. He believed that "certain electronic techniques used in radar could be applied to this problem"[4].

Wilkes learned a lot about the state of research in the United States from Professor Douglas R. Hartree who worked in Manchester, but regularly visited Cambridge. He took with him a report by Von Neumann about the state of computing in the United States and his idea of a stored-program machine.

While he was thinking this over, Wilkes was invited to attend a summer course on electronic computers at Moore School in Philadelphia. He decided to join this course, where he meets John Mauchly, who designed the ENIAC (Electronic Numerical Integrator And Computer) together with John Presper Eckert. When the ENIAC was designed, the stored-program machine concept had not been developed and when it was built and operational, it was already outdated by this idea by Von Neumann we still use for modern computers.

The largest difference between the ENIAC and a stored-program computer would be the relatively huge amount of memory the stored-program computer would need. Therefore Eckert and Mauchley for their next computer, the EDVAC (Electronic Discrete Variable Automatic Computer), were searching for a suitable memory technology. They adopted the concept of a delay memory. "The existence was assumed of a delay unit into which a large number of pulses, say 500, could be fed before the first began to emerge at the output. Such a device could be made into a storage unit, capable of holding 500 pulses indefinitely, by connecting the output to the input, via suitable circuits for amplifying and regenerating the pulses"[4]. The advantage would be that it was large (capacitywise) and fast enough. This memory concept would later also be used in the EDSAC.

On his way back home to the United Kingdom on the Queen Mary, the design of the EDSAC began taking shape.

3 Building the EDSAC

When work on the EDSAC began, resources in the UK were still pretty scarce. As Director of the Mathematical Laboratory Wilkes gets the responsibility and the funding to build "something significant". He decides to use fairly standard components which had already proven themselves, both because they were actually available and because his job was not to build the best and fastest computer in the world, but to build a practical computer which could actually be used.

Wilkes chose to use mercury tanks for memory and paper tape for input and output. Interesting in this time was the choice for decimal numbers and mnemonics for the input of instructions instead of a purely binary input and output. The finalised EDSAC had 41 predefined, hardwired instructions called 'initial orders' that were loaded before any paper tape was read.

3.1 Materials

EDSAC was mainly built from new components and not from wartime surplus stock, contrary to what people assumed[4]. They bought them either through the normal trading channels or directly from the Government. An exception were the vacuum tubes which they received as a gift from the Ministry of Supply, enough for the life of the machine.

In general, supply was poor, careful planning was therefore necessary in order to keep the build process running. Eventually, a fuel crisis would bring the whole production to a halt. Wilkes decided to take advantage of the situation and began designing the mercury memory tanks.

Although many components in the EDSAC are similar, they aren't mutually interchangeable, simply because small changes were made in the design of a component. It was still too early for standardization.

In his memoirs, Wilkes recollects himself designing a half adder – which nowadays come in fourfold on an integrated circuit no larger than a few square millimeters – takin up a whole chassis with 12 vacuum tubes in that time. It was one of the first times he worked with digital components and it made him realize that "a circuit engineer coming to digital engineering for the first time had a few things to learn"[4].

3.2 Mercury memory

Since the EDSAC was a stored-program machine, it needed a larger memory than earlier machines like the ENIAC. A memory made of vacuum tubes flip-flops was therefore not feasible because of the sheer size it would have occupied.

Another technique in use at that time which was much more compact was the usage of delay lines. As already mentioned in Chapter 2, this type of memory

consists "of a delay unit into which a large number of pulses, say 500, could be fed before the first began to emerge at the output. Such a device could be made into a storage unit, capable of holding 500 pulses indefinitely, by connecting the output to the input, via suitable circuits for amplifying and regenerating the pulses"[4].

Forementioned delay unit would be a tube, called *tanks* to prevent confusion with vacuum tubes[4], filled with mercury. "Sound travels in mercury at about 1400 metres per second so that a tube of mercury one metre long could contain about 1200 pulses, each with a total duration of one micro-second"[4].

The memory of the EDSAC consisted of two batteries of 16 mercury tanks, in which each tank could hold 16 "long" words of 35 bits. This means the store had a size of 512 words (of 10 decimal digits or 35 bits).

4 Programming the EDSAC

The EDSAC had only a small set of instructions. But these were still adequate enough to write useful programs. This chapter will give a small overview of the most commonly used instructions and how they were used to implement more advanced features like loops and subroutines.

This section uses the instruction set as documented in the second edition of the book *The Preparation of Programs for an Electronic Digital Computer* by Wilkes, Wheeler and Gill[6], colloquially called 'WWG', which was the final instructionset of the EDSAC.

4.1 Architecture

The EDSAC had one arithmetical unit, which contained one result register and one "multiplication register". A store which can be accessed by using 10 bit addresses. When referring to any of these locations the following notation is used

- $C(Acc)$: The value in the result register of the accumulator
- $C(R)$: The value in the multiplication register of the accumulator
- $C(n)$: The value in the store at address n

Words in the store are 17 bits, but the EDSAC had a facility to use an even numbered storage location, the following odd-numbered storage location and one bit sandwiched between those as a 35 bit "long" storage location for long numbers. The sandwiched bits are only ever used for long numbers and ignored for all other purposes[2]. The multiplier register is big enough to hold one long number. The result register is 70 bits, which is big enough to hold the product of two long numbers.

All numbers in the EDSAC are signed and in the range $-1 <= 0 < 1$. The first bit is the sign bit which is immediately followed by the decimal point. Let x be the number stored in the remaining bits (and thus $0 <= x < 1$), if the sign bit is set then the number represents $-(1 - x)$ otherwise x. So for example:

- 0 1100 0000 0000 0000 = 0.75
- 1 1100 0000 0000 0000 = $-(1 - 0.75) = -0.25$

Each instruction is a 17 bit word layed out as follows:

[2] All words were actually intended to be 18 bits, but because of hardware issues the first bit couldn't be used

- 5 bit instruction identifier
- 1 bit being the so-called B-digit
- 10 bit address digits
- 1 special indication bit

The exact meaning of the special indication bit depends on the instruction being used.

After some time the "B-register" or "index register" was added. This register could hold one integer. When the B-digit is set in an instruction, the value in the B register will be added to the address in the instruction. Making it a very useful tool for indexing. The value in the B-register will be referred to as $C(B)$

4.2 Basic instruction set

The basic form of instructions is as follows. First a function letter, then optionally an S indicating that the B-digit needs to be set, then the address and finally optionally an π indicating that the special indication bit needs to be set. So for example $AS100\pi$ means function letter A, address 100, B-digit is set (Thus the contents of the B register is added to 100) and finally the special indication bit is set.

This subsection will introduce various commonly used instructions, which are enough to understand the examples later on. The complete set of instructions can be found in WWG[6].

4.2.1 Arithmetic instructions

All arithmetical instruction take one store address as argument, perform the operation(s) and store the result in the result register. Some of the more often used instruction follow (the complete set of instruction is in [6]):

- An: Add $C(n)$ to $C(Acc)$ and place the result in $C(Acc)$
- Sn: Subtract $C(n)$ from $C(Acc)$ and place the result in $C(Acc)$
- Tn: Transfer $C(Acc)$ to storage location n
- Hn: Put $C(n)$ in the multiplication register
- Vn: Multiple $C(n)$ by $C(R)$ and add the result to $C(Acc)$

The special indication bit in the context of arithmetic instructions indicates the usage of long numbers instead of normal numbers. Thus $An\pi$ adds the long number at location n to $C(Acc)$ (n needs to be even for obvious reasons).

4.2.2 Jump instruction

To write useful programs with loops and (as we shall see) subroutines a way to transfer control to another location is needed. Therefore the EDSAC featured several jump instructions:

- Fn: Jump to n, that is take $C(n)$ as the next order.
- $Fn\pi$: If $C(Acc)! = 0$ then jump to n
- Gn: If $C(Acc) < 0$ then jump to n
- En: If $C(Acc) >= 0$ then jump to n
- Jn: If the B-register is non-zero jump to n

If the special indication bit is set with the G and E instructions, then $C(Acc)$ is cleared if and only if the jump is taken.

4.2.3 B-register instruction

The B-register holds an integer which can be set and changed directly (without the use of the accumulator). The following instructions are used to work with it:

- Bn: Place n in the B-register (Not the content of C(n)!)
- BSn: Add n to the value in the B-register. This is not a special instruction. Merely a normal usage of the B-digit.
- $BSnS$: Subtract n from the value in the B-register. Note that this violates the basic form. How this was exactly handled is unclear.
- Kn : Place the instruction $B(C(B))$ in storage location n

4.3 Implementing programs

With these basic instruction we can start implementing programs. Section 5 will explain how to load a program into the store. In this subsection we will assume the example programs and data are loaded in the store in the correct location. Table 1 shows the notation that will be used for programs in the rest of this section. The first column contains the address in the storage, the second the value initially stored in that store location (which can be either an instruction or a value) and the third column contains some comments if relevant to the instruction. Unless otherwise state, the control begins at the first entry of the table and C(Acc) is clear. The example below which adds x in storage location 200 to y in storage location 201 and saves the result in storage location 202.

100	A 200	C(Acc) = x
101	A 201	C(Acc) = x + y
102	T 202	C(Acc) = 0; C(202) = x + y
200	x	
201	y	
202	...	

Table 1: Tabular notation example

4.3.1 Implementing loops

Most useful programs contain loops. A simple example would be to calculate the product of 10 numbers. On the EDSAC with the use of the B register, this can be done by using the following code (assuming the 10 numbers are in address 200 to 209):

100	B 10	
102	BS 1S	Subtract 1 from the B-register
101	AS 200	Add C(200 + C(B)) to the accumulator
103	J 104	Jump if the C(B) is not zero
104	T 210	Save the result in C(210)

Unfortunately the B-register was only added after the EDSAC was in use for a few years. Programs implementing loops before needed an another solution and also even after the introduction of the B-register, some programs might want to use B register for different purposes.

The way this was solved is by recognising that each instruction is also just a normal number. In this example the loop need to execute $A200$ to $A209$. Which means the loop can change the $A..$ instruction until it has reached $A210$. To increase the address in the instruction by one we can add $P1$ to it (The P function letter equals 0). As this is an often-used technique, $P1$ is placed in storage location 2 at the start of the machine and it should not be changed by any program. The loop can now be written as follows:

100	A 210	The accumulator and C(210) are assumed to be clear at the start
101	A 200	This instruction will be replaced by $A201...$ $A209$
102	T 210	Save preliminary result in 210 and clear the accumulator
103	A 101	Put the Add instruction into the accumulator
104	A 2	Increase the address with one
105	U 101	Change the Add instruction for the next iteration.
106	S 100	
107	G 100 π	If $C(101) < A210$ then start the next iteration with

| | the accumulator cleared otherwise continue

Note that after the loop is finished the instruction in location 101 is $A210..$ If it is needed to execute the loop again some code needs to be added to reset location 101 to $A200$. Either before or after the execution of the loop. Also note that it is not generally possible to use one of the useful instruction in the program to check if the end is reached (In this case the instruction at location 100 is used for this purpose).. In which case it is needed to use some known storage location to store the value to compare to.

The way a simple iteration is implemented using the B register can be compared to how it would be done on a modern machine. The second variant which modifies the code in place is, like all self-modifying code, frowned upon these days though. While it was a good way to do things on the EDSAC, it has been recognized that self-modifying code is error-prone and hard to maintain. Also for security reasons modern operating systems try to make memory containing code read-only, on such a system the technique is simply impossible.

4.3.2 Implementing subroutine calls

The EDSAC can perform only very basic operations, thus for more extensive calculations a large sequence of orders is needed. It was recognised very early on that it was very convenient to break up orders into self-contained groups, referred to as subroutines. Their was even a complete library of subroutines available to users of the EDSAC, how to use these is covered in the next section.

The simplest way to use subroutines is to simply insert their code at the point control should be transferred to them. These were called “open” subroutines. From a modern perspective we would say those subroutines were inlined. The other variant called “closed” subroutines could be placed anywhere in the store and control is transferred by means of a jump. Where today most if not all systems have a calling convention, that is a standard way how arguments are passed to functions and how results are communicated back, for the EDSAC each subroutine specified in which location it expected its arguments and where it would save its results. It was the programmers responsibility to setup the environment correctly before calling a subroutine. The next issue is to ensure that after the subroutine the control is transferred back to the correct location. There were two standard conventions for this, so-called “Closed A” subroutines and “Closed B” subroutines.

Closed B subroutines use the B register and were thus only used after it was fitted to the EDSAC. The calling code would place the address of the instruction before the actual jump in the B register and expected the subroutine to transfer control to the instruction after the jump when done. This was done as follows, first the setup code:

200	B 200	Put the address of this instruction in the B register
201	F 150	Jump to the subroutine at location 150
202	...	Control is transferred here after the subroutine has finished

At the time of the jump $C(B)$ equals 200. Below is the boilerplate code for the subroutine. Note that it is safe for the subroutine to use the B register.

150	K 150 + r	place the B C(B) instruction in location 150 + r
151	...	Start of the actual subroutine code
...	...	Subroutine implementation
$150 + r - 1$	..	End of the actual subroutine code
$150 + r$	Z F	As soon as the subroutine is called this is replaced by B C(B). In this example after the subroutine is done this location will contain B 200
$150 + r + 1$	FS 2	Jump to C(B) + 2. in this example location 202

Closed A subroutines were exclusively used before the B register was added. After the addition of B register this way of implementing subroutines became unfavoured, but a lot of existing subroutines still used it as they were written before the addition. Instead of passing the location in the B register, Ap is put in the accumulator, with p being the address before the jump instruction. The subroutine header then adds $U2$, which results in $Ep + 2$. Which is the jump instruction needed to transfer control to the right location. First the setup code (the accumulator is assumed to be clear):

200	A 200	Put this instruction in C(Acc).
201	G 150	Jump to the subroutine
202	...	Control is transferred here after the subroutine has finished

Instruction $U2$ was available in storage location 3 after the start of the machine by convention as it was often needed. The subroutine boilerplate now becomes the following (the accumulator contains Ap):

150	A 3	
151	T 150 + r	Save $Ep + 2$ in the right location
152	...	Start of the actual subroutine code
...	...	Subroutine implementation
$150 + r - 1$	..	End of the actual subroutine code. Closed A subroutines are assumed to leave the accumulator clear after finishing
$150 + r$	Z F	As soon as the subroutine is called this is replaced by $Ep + 2$. Thus at this point the control will be transferred back

Note that in these examples the E or G function letters are used instead of F. Which means that $C(Acc) >= 0$ or $C(Acc) < 0$ respectively must be true at the time of the jump call. The reason for using the E function letter is because when Closed A subroutines were first created the F instruction wasn't implemented.

After the F function letter was added Closed A subroutine (calls) were ofcourse free to use F.

These days returning from a "Closed" subroutine is usually done by having using the return address which has been put on the stack. Most architectures have special instructions to enter and leave subroutines. Also argument passing is much more standardized these days, each architecture has its own calling convention which (should be) used by all code on that architecture. Thus it is no longer needed for each subroutine to specify where it expects its arguments, just what arguments there are and in which order. And like mentioned in the previous subsection, self-modifying code (as needed for both variants) is frowned upon or just simply impossible. Furthermore the EDSAC' way of implementing subroutines means that you can't call a subroutine while it is executing. This prevents implementing recursive function calls among other things.

But in retrospect, the important fact is that it was possible and very common to use subroutines. And without a stack infrastructure or special subroutine facilities in hardware, these techniques were probably the simplest and easiest ones possible.

5 Preparing and running programs on the EDSAC

Writing a program from the EDSAC is one thing. After that, it needs to be put in a form suitable for input to the machine. Then it needs to be run on the machine, after which it is likely that some debugging needs to be done and some fixes need to be applied to the program. This chapter discusses the steps to be taken to go from an initial program to something that runs and works correctly on the EDSAC.

Section 4 showed some examples of programming techniques on the EDSAC. It assumed that the programs were correctly loaded into the store. To do this the program needs to be put on tape in format the EDSAC recognises.

5.1 Tape format

When the EDSAC is started the initial program hardwired in the machine is loaded into the store and executed. This program can read information from a paper input tape, but the tape needs to be in a special format. Apart from instructions the tape can contains various control sequences. The most commonly used ones are listed below:

T m K	Cause the next order read from tape to be placed in storage location m
G K	Place the address in the current Transfer Order in storage location 42
T Z	The address in the Transfer Order is replaced by the address in storage location 42
E m K P F	Transfer control to the order in storage location m, leaving the accumulator clear

The instruction as shown in section 4 are the instruction as they are in the store. On the input tape each instruction needs to be followed by a terminal code to indicate their end. The terminal code don't just indicate the end of an instruction though, dependant on the terminal code the content of a certain store location is added. Table 2 contains the most important terminal codes, [6] contains a more complete table.

Code letter	Location whose content is added to the order	
F	41	Always zero
θ	42	Set by the G K control sequence
D	43	2^{-16}

Table 2: Terminal codes

The terminal characters allows the code and especially subroutines to be written independent of the actual address it will get in the store. Terminating a instruction with F means the instruction using an absolute address. The sequence πF never actually gets written as D also adds 2^{-16} and can thus be used as a shorthand. When terminating with θ the value in storage location 42 will be added. This means the Closed B subroutine shown in the previous section could have been written on the tape as follows:

	T 150 Z	Instruction after this will be placed at location 150
	G K	This places the current transfer order
		in location 42, in this case 150
$\theta + 0$	K r θ	place the B C(B) instruction in location
...	...	Subroutine implementation
$\theta + r$	Z F	
$\theta + r + 1$	FS 2 F	

What's especially nice about this, is that apart from the initial "T 150 Z" the code can be reused in any program without requiring to be placed at a certain address in the store.

5.2 Creating the program

After the job of the program to be was formulated. It was broken up in various parts. Some implemented by existing subroutines from a the subroutine library, other by newly implemented subroutines. After the program and its new subroutines were written, usually on printed program parts, it need to be put on tape for input into the EDSAC. Usually the individual new subroutines were punched on a program tape first. This was done using a keyboard perforator, usually twice. After which both tapes were feed to a comparator, which would stop as soon as a discrepancy was found. The the full test tape could be made using a tape reproducer, which could combine the new subroutine with existing subroutines from the library. Also twice and compared afterwards. Obviously this was very tedious work, so the incentive to get the code right the first time was great. After all new subroutines were tested everything could be combined onto the final program tape using the same procedure.

Another way to prepare tapes was to make the final program tape right away and use jiffy tapes. A jiffy tape was a short custom tape which could be loaded and could overwrite certain parts of the program in the store in such a way that individual parts could be tested.

5.3 Diagnosing and correcting errors

It was realized quite early on that it was quite rare that a program worked the first time:

"As soon as we started programming, we found to our surprise that it wasn't as easy to get programs right as we had thought. Debugging had to be discovered. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs" – Maurice Wilkes, 1949[4].

Because of this various techniques were created to diagnose errors. It was common practise to include a control sequence at the end of a program tape in development that would stop the EDSAC until the reset button was pressed. This allowed for the operator to instead of continuing the program, press the start button and restart the initial orders to load a jiffy tape with correction or a diagnostic routine. After the program development was done, usually a new version was made containing the correction and without the control sequence to halt the machine.

Diagnosing errors could be done in several ways. There was a overflow alarm which would go of when an overflow in the accumulator had occurred if the ø instruction was used instead of T. Several CRT monitor tubes could be used to monitor the state of selected registers or delay lines. Also just listening to the sound the machine made by having a loudspeaker connected to the bitstream of the accumulator proved to be a good way to monitor the machine. The human mind is very good in recognising and memorising the specific tones and rhythms of loops and jumps.

Various routines were in the library which could follow the steps a program took and output a trace, either of the complete program or only from a certain point or just the jumps or ... When the machine had stopped due to an error condition various "post-mortum" routines were available to inspect and print out the state of the store (or parts of it) in various formats.

As the tape preparation room was two rooms away, there was a small punching machine close to the EDSAC for quick and small corrections. Either by creating small jiffy tapes or by punching in some free space that was usually left on the program tape for this reason.

Not all faults were program errors though. Hardware faults occurred often, with an average interval of half an hour. There were good and bad days though, which meant the actual time between hardware errors was hard to predict.

5.4 Running programs and tests

During the daytime the EDSAC was run on a schedule, which included short periods of a few minutes for program testing purposes and longer production runs. If the programmer/writer wasn't available the run would be done by an operator, who had a queue of tapes ready run available. During the night there was no engineering staff available, but the EDSAC was free to use for certain groups until hardware faults made it impossible. Minor hardware faults could sometimes be fixed by changing one of the many potentiometers which changed

the gain of an aging electronic tube. While some people became quite good at this, adjusting the wrong one would actually make the problems worse.

6 Using the EDSAC

Unlike some other machine of the time the EDSAC was meant to be put to practical use and not just be demonstration that a stored program computer could be build. This chapter will focus on some applications for which the EDSAC was used, ranging from very simple demonstration applications to quite complex applications with a big scientific value.

The first useful program that ran on the EDSAC was one to find prime numbers via a simple subtraction process, which became quite slow as it progressed. In 1951 another prime number program found a 79 decimal digits prime, which was the largest prime number discovered at that time.

In 1950 R.A Fisher used a program written by David Wheeler to *determine gene frequencies* that have a defined selective advantage, not obscured by dominance. The paper in which the table was published was still referred to by geneticists until at least 1992, according to Joyce Wheeler[3].

Sir John Kendrew studied *whale myoglobin* and used the EDSAC to calculate its structure by using observed X-ray diffraction patterns. This revealed the structure of the molecular structure to a resolution of 6 angstroms. Later the EDSAC 2 was used to calculate it in a resolution of 2 angstroms. In 1962 Kendrew received a Nobel price (shared with Max Perutz) for their work in the field of molecular biology.

During the design of the EDSAC 2, a distribution of microinstructions over a matrix of memory cores in such a way that it met various requirements was needed. The EDSAC was used to calculate such a distribution. Making it one of the earliest examples of one computer helping to automate the design of another computer in a non-trivial way. The EDSAC was also used to generate the wiring schedules of the EDSAC 2.

Apart from these uses, the EDSAC was used in various other ways in the fields of astronomy, physics, economics and other. A more complete overview of the various applications can be found in a paper by Joyce Wheeler in the IEEE Annals of the History of Computing[3].

7 Other computers in the UK

During the time-period of the EDSAC, there were various other projects to build digital computers. This chapter will have a brief look at some of the other machines in the United Kingdom.

7.1 Manchester Small-Scale Experimental Machine

The Manchester SSEM, or Manchester Baby was, as the name implies, an experiment to build a stored-program electronic computer. It was developed by Frederic C. Williams and Tom Kilburn at the University of Manchester. It was the first computer to employ the concepts of the Von Neumann architecture, which in general means it had a control unit and a shared store containing both instructions and data.

The SSEM was a fully functional machine, but it had only a limited number of seven instructions and input and output were somewhat cumbersome; input was done in binary using a simple keyboard, the output of the program was shown on a display tube.

It ran its first succesful program on June 21, 1948.

7.2 Manchester Mark I

After the succesful demonstration of the SSEM, development continued to deliver a full-scale calculator. The key changes were a magnetic drum instead of paper tape for input, and an additional register.

With these modifications, the Mark I would have been the first practical stored-program computer. However, it ran its first succesful program in the night of June 16–17, 1949, only a few weeks later than EDSAC. The final specification version ran succesfully in October 1949.

8 After the EDSAC

Like many major achievements, EDSAC had some influence on later machines. Its influence on EDSAC II being the most direct. This chapter will discuss various items introduced by EDSAC, which had a profound impact on later projects.

8.1 LEO

The Lyons Electronic Office (LEO) project was started by food and catering business J. Lyons and Company, who after a study tour to the USA in May 1947 decided that they'd want to have a machine to automate certain clerical tasks. This was foremost a decision by John R.M. Simmons and T. Raymond Thompson – both graduees from Cambridge with honours – responsible for office management at Lyons.

Simmons and Thompson visited the USA "to look at the so-called giant brains then being constructed"[2], "to find out what use might be made of such machines for routine clerical work"[2]. They had a meeting with Herman H. Goldstine at Princeton and – according to Thompson – within an hour they understood how a computer would work and how it could be applied to clerical tasks.

It is ironic that it was in the USA that Simmons and Thompson discovered that construction of such a machine had already begun in the UK, namely at Cambridge and when they met Wilkes, they "were considerably impressed by the construction of the EDSAC, then in its early stages"[2]. In October 1947 the Board of Lyons decided to give Wilkes financial help for the project and to investigate the requirements to a computer for automation of clerical work.

The collaboration became very close in 1948, when Derek Hemy, member of Lyons' staff designed and wrote a program that would do payrolls on EDSAC. It never ran, but it gave very clear and vital indications for the modifications that were needed to the design of the EDSAC if Lyons' machine was to run clerical applications efficiently.

Using the plans of EDSAC and the modifications suggested by Hemy's work, the plan for LEO 1 was written. Initially, Lyons wanted the machine to be built by a contractor, but as no company had ever done anything like this, no contractor could be found who was willing to do it. So it was decided to build the machine in-house. The decision to actually start constructing LEO 1 came immediately after EDSAC ran its first program in May 1949, which printed a table of squares.

While the logical design of EDSAC remained largely unchanged in LEO 1, the physical units were constructed in a way that allowed easier replacement in case of failure and the LEO 1 had forced draft cooling, while EDSAC didn't have any cooling fans at all.

Derek Hemy's study showed that for Lyons the memory store should be doubled to 64 mercury delay lines. In addition, the input/output would need buffers and hence even more delay lines were added. LEO 1 got five mercury memory batteries.

LEO 1 had around five thousand vacuum tubes, which would make it relatively unreliable (vacuum tubes lose performance over time). Rapid fault finding was therefore a high priority and elaborate testing schemes and maintenance procedures were conceived. It was possible to isolate certain parts of the system and test them individually. Special test programs would use specific tubes, in such a way that the output of the program would show whether a certain tube was defective.

LEO 1 could be programmed using the instructions of EDSAC. The set of initial orders was also equivalent. When LEO 1 got a punch-card reader, a single initial instruction to read in the first card was added, which could contain up to 24 instructions on one card, sufficient to read in the rest of the cards. The addition of input and output buffers required some additions in the input/output instructions and the concept of a B register was also added to LEO 1.

Because of the equivalent instruction set, programs written for EDSAC could be run on LEO 1 without modification, which enabled a number of outside users, such as the Royal Ordnance Board, to use LEO 1 before it had taken up its office work.

LEO 1 took up its routine office work in January 1954.

8.2 EDSAC 2

When work on EDSAC had finished and it had been running for well over a year, Wilkes set forth some conclusions on the work, based on his experiences with the building and usage of the machine. He said that the "first consideration for a designer, at the present time, is how he is to achieve the maximum degree of reliability in his machine"[5]. He also mentioned that the reliability of a machine is defined by its components, especially "the amount of equipment it contains, the complexity, and the degree of repetition of units"[5]. This call for low complexity and repeated use of the same units, led the arithmetic design of EDSAC 2 to become more parallel than of its predecessor.

However, the speed of the computer had now increased so much that the outdated mercury memory could never keep up. Therefore a new type of memory had to be found. The solution came in 1953 when the first ferrite-core memory was installed in the Whirlwind computer at MIT. Ferrite-core memory consists of many small ferrite rings which can be set magnetically in two polarities – thus representing the binary values true and false – and then read back electrically by using its induced field effects.

EDSAC 2 received a separate read-only store for library subroutines. This contributed to the speed of the machine, because it didn't need to load them every time and were always available.

It was also to be one of the first computers with a microprogrammed control, a concept where the instruction set would not be hard-wired in the machine, but a microprogram would describe the instructions. This allows for changes in the instruction set late in the design process and for more complex instructions.

EDSAC 2 came into operation in 1958 after five years of design and construction and was closed down in November 1965.

9 Concluding remarks

Perhaps the greatest achievement of the EDSAC is that it showed that computers could be used for practical purposes and not just as a demonstration of engineering capabilities. Most of the technologies used to build and use the machine – Mercury delay lines, valves, uniselector switches, paper tape – are no longer used. But the basic principle of a stored program computer as used in the EDSAC is still in use today.

And even though compared to the design of modern CPU the EDSAC is extremely primitive, the very basic instructions: Add, Subtract, Load value into a register and the various jumps instruction are still comparable to those on a modern machine. Programs are written in basically the same way, by combining various newly written combined with an existing libraries of subroutines. Obviously the problem of combining all these in memory is no longer something the programmer needs to deal with.

The way modern programs are debugged is still comparable. Modern debuggers can be used to trace the flow of a program, crash dumps to do "postmortem" analysis etc.. Just like could be done with the various available routines for the EDSAC.

The biggest change since the time of the EDSAC is that computing is now common-place and reliable. It is no longer a laborious task to prepare a program and having to deal with hardware failures is rare. But apart from that, things are not so different then in the times of the EDSAC.

To honour his achievements and underline the importance of his developments, Maurice Wilkes was knighted in the 2000 New Year Honours List "for services to computing"[1].

References

[1] New Years Honours List – United Kingdom. *The London Gazette of Thursday 30 December 1999 Supplement No. 1*, (55710), 1999.

[2] John H.M. Pinkerton, Derek Hemy, and Ernest H. Lenaerts. The influence of the cambridge mathematical laboratory on the leo project. *IEEE Annals of the History of Computing*, 14(1):41–48, 1992.

[3] Joyce M. Wheeler. Applications of the EDSAC. *IEEE Annals of the History of Computing*, 14(1):27–33, 1992.

[4] Maurice V. Wilkes. *Memoirs of a Computer Pioneer*. The MIT Press, 1985.

[5] Maurice V. Wilkes. EDSAC 2. *IEEE Annals of the History of Computing*, 14(1):49–56, 1992.

[6] Maurice V. Wilkes, David J. Wheeler, and Stanley Gill. *The Preparation of Programs for an Electronic Digital Computer*. Addison-Wesley Publishing Company, inc., second edition, 1957.

Acronyms

EDSAC Electronic Delay Storage Automatic Calculator

SSEM Small-Scale Experimental Machine

LEO Lyons Electronic Office