Experience with Some Early Computers

R.F. Churchhouse

University of Wales College of Cardiff

1. Background

From 1946 to 1949 I was a student of Mathematics at Manchester University. The Head of the Mathematics Department was Professor M.H.A. Newman and in 1948 the logician, Alan M. Turing, joined the department. Professor Newman taught us Complex Variable Theory and Topology and Alan Turing gave us a course in our final year; not on Mathematical Logic, which was not in the syllabus at that time, but Analytic Number Theory. He was a popular lecturer, guiding us gently through a difficult subject. He had a noticeable stammer but it was not a serious impediment and, in some way, it added to our enjoyment of the lectures. It was said (truthfully) that he was a very good runner, close to international standard, who might have been selected for the Olympic Games of 1936. In 1948 I found this surprising; he looked a bit on the heavy side and slightly dishevelled, as if he was about to fall apart. I had no idea of how distinguished he was but I did notice that Professor Newman treated him with particular respect.

During 1948 I became aware that the Professor and Alan Turing were spending a great deal of time building some kind of a machine in the basement of the Physics Department (the Head of which Department, Professor P.M.S. Blackett, had just won the Nobel Prize). They were collaborating with Professor F.C. Williams, Professor of Electrical Engineering, who played a major role in the building of the machine, which became known as the Manchester Mark I computer. He was, of course, the inventor of the Williams Tube method of storage, which was the technology used on the machine.

I went to see this machine shortly after it ran its first programs in June 1948. It was a somewhat hazardous visit since, as the well-known picture shows, there were racks of valves and cathode ray tubes around the room with cables carrying heavy currents passing above and on every side so we had to tread a delicate path. Although I never used this particular machine some of my friends did and I remember them showing me it attempting to calculate a table of values of $\log \cos(x)$. However, owing to an error in the program which caused overflow, it failed to do so and broke down after a few minutes of running. Programming in those days was such an esoteric art that a successful program for computing a function such as $\log \cos(x)$ was considered to be worth an M.Sc.!

In October 1949 I went to Cambridge University to do a Ph.D. in the Theory of Numbers and was surprised to learn that Dr. M.V. Wilkes had also been building a computer, which was called EDSAC. I had assumed that building computers was an activity exclusive to Manchester (as a loyal Mancunian I subscribed to the old saying that “What Manchester does today others do tomorrow”). When a friend (C.B. Haselgrove) told me that he was using EDSAC to compute the zeros of the Riemann Zeta Function and the development of star clusters, I was very much impressed. Up to that point I’d regarded the machines as playthings but it now occurred to me that they might actually be useful.

In 1952 I had to do my National Service and was put to work in a government scientific laboratory where I came into formal contact with computers for the first time and so began my programming career.
2. The Manchester Mark I (1951)

The machine on which I began programming was one of those produced by Ferranti Ltd. (a Manchester firm) and was based upon the 1949 Manchester University machine though it was slightly fancier and larger. It was fancier only in the sense that since it had been built by Ferranti it was enclosed in cabinets so that we were not visibly surrounded by cables carrying large currents. According to Lavington [1, p.440] nine such machines were delivered. Whether any two were absolutely identical I don’t know, some may have had more (Williams tube) store than others, for example.

The basic machine had 256 words of 40 bits which were organised as eight 32-word “pages” thus introducing an important method for handling “virtual storage” (as it was later known), an idea which was fully exploited in the Manchester University Ferranti Atlas some years later and which has been widely used ever since.

Instructions occupied 20 bits which were divided into three fields
5-bit function code
5-bit designating a “B-line”
10-bit address

(this description is specific to the machine that I used).

There were eight “B-lines”. Their major role was to modify the address; thus Manchester also invented “index registers”, as they were called on later machines such as the IBM 704.

With a 10-bit address field we could only refer to up to 1k of store. We didn’t have that much so that was no problem, indeed this had minor advantages: we used to clear the accumulator by loading it from address A, which didn’t exist! We also had a magnetic drum with 16k words. Transfers between the store and drum were done in ‘pages’ of 64 20-bit words.

Not every track on the drum was considered “good” and one of my earliest programs produced a list of “doubtful” tracks. The simple method for doing this was to fill a page in store with 64 values of a (number-theoretic) function, write it to the drum, read it back into a different page and compare with the original. If they didn’t agree a bell was rung on the typewriter and the writing/reading was performed again. If the bell rang three times the location of the offending track was typed out. A list of these “doubtful” tracks was kept in the machine room and we avoided using them. In all cases where there was a transfer between store and drum we carried out the transfer two or more times and checked for agreement.

Instructions were written in a four character format; each character representing 5 bits, using the international teleprinter code. Instructions were either read in via a 5-hole punched paper tape or via hand switches on the console.

There was an 80-bit accumulator (“A” register, hence the use of “B” for the modifier registers) and the function codes included a multiply instruction but not division. Naturally there were no floating point instructions.

Programs were written in absolute code using the teleprinter alphabet which, in our version began

/T 3 0 9 H N M 4 L R G I P ...

so a typical instruction might be

S T J N

(Store the top 20 bits of the accumulator in the address JN modified by the contents of B-line number one).

It was essential to know this teleprinter alphabet and to be able to do arithmetic using it, eg
H+H=R (ie 5+5 = 10). Turing baffled the London Mathematical Society when he gave a lecture
to them at this period until they realised that not only was he using (mod 32) arithmetic, which was easy enough, but that the least significant digit was at the left! In the machine which I used this particular eccentricity, which was a hardware feature of the machine [Lavington, p 436] was, mercifully, hidden from us by the (otherwise very primitive) software.

There was no real programming manual as such. At Manchester University Turing had produced one for local use but I never saw a copy. I was simply given a description of what each instruction did and a program written by someone else - it was the Newton-Raphson method for finding a square root - and told to study it and then write a statistical program. Since the machine had no division order the Newton-Raphson iteration

\[ x_{n+1} = \frac{1}{2}(x_n + \frac{a}{x_n}) \]

required a more complicated program than might be expected.

When I began looking at the program I didn't know what the B-lines did but eventually came to the conclusion that if they modified the address by subtracting their contents then the program made sense. Thus enlightened I wrote the statistics program which, to my joy and amazement, "ran the first time" and the direction of my future career was settled. Had it failed I might have returned to Pure Mathematics exclusively.

Most instructions took about a millisecond but multiplication took twice as long. If we had to carry out a division we had to use an iterative method preceded by scaling e.g.

**Example** To find the reciprocal of a:

(i) scale "a" by 'shifting' so that it lies between 1/2 and 1;

(ii) put \( x_0 = 1 \);

(iii) iterate using the formula

\[ x_{n+1} = x_n(2 - ax_n) \]

until two consecutive iterates agree to within the limits of achievable accuracy:

(iv) re-scale the result.

Convergence is quite fast (quadratic).

If we had been doing some counting in a B-line (as in a statistics program) we just used repeated shifts and subtraction to get the digits in base 10 and printed them as they were found.

Almost every program was a challenge. I recall having to invert a 32x32 matrix. In a store of less than 1k the matrix couldn’t be held in one piece, so chunks were continually being transferred to and from the drum, and with no floating point or division instructions the arithmetic and storage management was tortuous.

The machine took half-an-hour to warm up every morning after being switched on and the screen on the console, which showed the contents of a page of store as rows of bright (=1) or dim (=0) dots, would be periodically swept by a kind of snowstorm as bits changed their parity.

We didn’t attempt to use the machine until the “snowstorms” had stopped, for otherwise the bits in store would be changing and the instruction codes would be altered so that an instruction to load the accumulator might be changed to one to punch cards. This made life quite exciting. Breakdowns were very frequent and it was obligatory to build “restart points” into every program that was due to run for more than a few minutes, so we would print out parameters on
the typewriter at regular intervals. These parameters would then be set on the hand-switches when it was necessary to restart.

Life was immensely enjoyable. We felt that we were a breed apart. There weren’t many of us - it seemed very unlikely that there ever would be. Every program called for ingenuity and we took great pride in efficiency, reducing execution time to a minimum. Since I/O did not proceed in parallel with computation we planted I/O instructions at very carefully timed points in the program so as to achieve optimal operation. I remember a particularly skilful programmer telling me that he felt that one of his best programs was like “a concerto for matrix and integral”, and I understood what he meant.

If programmers were a rare breed in those days operators were even rarer. The simple rule was: if your program was due to run, you ran it!

3. Interlude (1951-1955)

In the second half of the 1950’s many firms announced that they were going to produce computers. Many of these either never appeared at all or, if they did appear, were not a commercial success. Some very strange machines were proposed and may even have been produced in small numbers. I can recall three that were built (but never produced commercially):

(i) a machine that was based on decimal arithmetic;
(ii) a machine that didn’t even have an addition (or subtraction) instruction, addition was done by table look-up;
(iii) a machine that used four-address instructions (the purists would say “three and a half address”), one instruction in this machine being

\[
\begin{array}{cccc}
S & A & B & C & D \\
\end{array}
\]

which meant:

“Sort all the words between store locations A and B and put the sorted list into the addresses beginning at C then, go to location D for the next instruction”

The memory of this machine was based on delay-lines, hence the significance of the fourth address (D).

Technology was advancing rapidly; core stores replaced electrostatic storage so memories were becoming faster and larger, magnetic tapes, initially regarded as an unreliable storage medium, were becoming accepted, logic circuits were becoming much faster and I/O devices now included line printers (150 lines a minute), card readers/punches and faster tape readers.

Some computer firms (eg Univac) were paper-tape oriented whilst others (eg IBM) based their I/O on cards. There was much debate as to the merits of cards and paper tape eg cards could get out of order but could easily be replaced, tapes could become excessively long and get into a tangle - at Manchester University they used to unroll tangled tapes from the balcony of the third floor. A fast-moving paper tape could also inflict a nasty cut, akin to a razor blade, if you happened to touch its edge whilst it was being read. On the other hand data on tape couldn’t get out of order. Some people felt very strongly on this matter: I recall one man who detested cards so strongly that he tried to avoid using the word “card” at all. On one occasion, speaking to me, he used it and immediately added: “Now I must go and wash out my mouth since it has been defiled”. He didn’t, in fact.

4. The IBM 704 (1955)

IBM had delivered its first general-purpose computer, the IBM 701, to a customer in 1953. I used this machine only indirectly, as I shall explain, so there is little that I can say about it from a personal point of view. I was aware that it used Williams tube electrostatic storage but there
was more of it and with the bigger word size (36 bits) we had many more bits of storage available than on the Manchester machine. By the time that I saw the 701 however it was due to be replaced by the IBM 704, a machine that was an enormous commercial success, with sales of many hundreds (against an original forecast of 50). When first announced (May 1954) the 704 was also to have electrostatic memory but in fact all the machines were delivered with the (then new) magnetic core memories.

In preparation for the arrival of the 704 a simulator of that machine on the 701 had been provided by IBM. This enabled us to “debug” (the word was now in use) our 704 programs on the existing machine. Programming was done in Assembly Language, which I found very easy to use compared to Absolute Code. The Assembler was quite primitive: each line of Assembly code was compiled to one line of machine code, i.e. there were no “macros”. However the use of symbolic names for variables and storage seemed a great step forward, particularly when changes had to be made.

It may seem incredible today but there were programmers who were very reluctant to use Assembly Language, claiming that it would result in inefficient programs. Such people liked to know exactly where every variable was located in the store. My argument that since there was a one-to-one mapping there couldn’t be any loss in efficiency, apart from the small overhead of the Assembler converting the program to machine code and producing binary punched cards, fell on deaf ears.

The 704 had a word size of 36 bits and came with 4K words of core storage memory. This seemed luxurious and we were incredulous when we were told by Herb Grosch of General Electric that his 704 had 8K. The instruction field occupied 10 of the 36 bits but of the 1024 theoretically possible instructions only about 70 existed. One of my colleagues wrote a program to generate all possible 1024 instructions to see what would happen, printing out the contents of the A (accumulator) and Q (quotient) registers, the index registers (of which there were 3), and the storage location addressed before and after each instruction. If I recall correctly only the “all zero” code and the “all one” code caused a fault. All the rest did something or other but the only useful discovery was that on our machine, which had no floating point (which was an optional extra), the instruction “un-normalised floating add” cleared the accumulator - the equivalent of loading from A/ on the Manchester machine!

The reliability of the 704 was a revelation, no doubt due in large part, to the core store. It was also much faster than the Manchester machine; most instructions took 24 microseconds though multiplication and division took longer and one interesting order, CAS, (Compare Accumulator and Storage) took 36. The operation was:

“If the contents of the accumulator are (less than, equal to, greater than) the contents of the specified storage location go to (the next, the next but one, the next but two) locations for the next instruction”

- When Fortran appeared some time later the “IF” statement seemed to have been designed with CAS in mind.

Virtuosi programmers tried to out-do each other with clever, or compact, programs. I recall one who squeezed a program onto a single (24 instruction) binary card, punching all the holes on a hand-punch. To achieve this he had to suppress the check sum; 23 instructions were not quite enough!

The 704 had one feature which I found very annoying; transfers to and from magnetic tape used the Q register as a buffer; an extraordinary economy. Consequently we couldn’t use Q for 500 microsecs after a tape write or read. More than once I forgot this and discovered that Q was full of garbled data and I/O. This was my only complaint though, I was a great admirer of the 704 for which I wrote something like 100 programs.

By 1955 some systems of “autocoding” had been produced in various Universities and laboratories. My own first encounter with “higher level languages” didn’t however come until
late 1956 or early 1957. I'd been asked to write a program to do with the placement of core stores which involved the numerical evaluation of elliptic integrals. I was just about to start coding this, in Assembler, when I was given a package from IBM: cards and instructions for using a system of: "Formula Translation". I read it, thought it looked interesting but was rather sceptical. However I thought I'd try the elliptic integral program using it and then write a "proper" program later to check the results. It took me only an hour or so to write it in Fortran, compared to ten times as long in machine code. The results were identical and the Fortran program wasn't very much slower than the machine code. I told everyone around of this marvellous new system. The "old school" ("efficiency must be impaired now" - some truth this time) would have nothing do with it and persisted in this attitude for quite a long time (months). Of course there were some programs for which Fortran was unsuitable, such as those involving extensive bit manipulation, which, as one programmer succinctly put it "using Fortran for bit-manipulation is like shelling peas wearing boxing gloves"; but for scientific calculations it was a great help.

5. The Univac 1103 (1957)

The Univac 1103 was an interesting machine from a programming point of view and, although it was overshadowed by the more commercially successful IBM 704, it is remembered with affection by those of us who used it and it deserves to be accorded a prominent place in the history books.

The story of how the Univac 1103 came to be built is a fairly complex one and those interested will find a fairly detailed account by Tomash in [3, 485-496].

Like the IBM 704 the Univac 1103 was a 36-bit machine with core store of 4K words upwards and a magnetic drum as secondary storage. Although they also had magnetic tapes they were quite different from the IBM types, which became the "industry standard". The main points of difference were:

(i) it was based on two-address logic;
(ii) it had no index register ("B-lines");
(iii) it had a "repeat" instruction;
(iv) it was paper-tape oriented.

In terms of processing power it was quite comparable to the 704; instruction times, typically 36 microsecs, were longer than those of the 704 but the two-address logic meant that fewer instructions were needed to accomplish the same task and, in my experience, programming the same jobs in machine code on both machines, there was little to choose between them.

The 1103 came with only the most rudimentary software; the company did not regard the provision of software as part of its remit. In 1958 I was asked "to write a program so that people who can't program in machine code can write programs in something like English". I invented a simple language (I called it "Pipedream") based upon 17 commands (calculate, input, print, jump, test etc); it took me 3 weeks of actual coding, spread over 6 months; 2,000 machine code instructions in all. There were no books on how to do it; I just worked it out as I went along. No-one pestered me, I went at my own pace. Happy days! When it was finished I gave a half-day course on how to use it and wrote a short report; dozens of people used it over several years.

To return to the two-address logic. The instruction format was

- 6-bit instruction code
- 2-bit "repeat" parameter
- 14-bit first address (u)
- 14-bit second address (v)

There were 62 actual instruction out of the 64 possible, instructions; 00 to 77 (octal) didn't exist.
The most fundamental instruction was “Transmit positive”, 11 (in octal), TP (when we had an assembler):

\[
\text{TP } u \ v
\]

copied the contents of location \( u \) to location \( v \).

(on the 704 this took 2 instructions:

\[
\begin{align*}
\text{LDA } u & \quad \text{Load accumulator from } u \\
\text{STA } v & \quad \text{Store accumulator in } v
\end{align*}
\]

it also changed the accumulator).

The novel, and powerful, Repeat Instruction, 75 (in octal), RPT in assembler, caused the instruction which followed it to be repeated a specified number of times the \( u \), \( v \) addresses being increased by one each time, or not, according to the “repeat parameter” (which was denoted, in assembler by \( J \)).

Thus:

\[
\text{RPT 3 1000 W}
\]

\[
\text{TP } u \ v
\]

meant:

Repeat the next instruction (TP) 1000 times,

increasing both \( (J=3) \) the \( u \) and \( v \) addresses by 1 each time and then go to location \( w \) for the next instruction.

So, in this case, a block of 1000 consecutive words would be copied to new locations.

Although the Repeat Instruction did not make up for the lack of Index Registers it provided a very efficient way of moving arrays, etc. particularly since the instruction which was repeated ran at a higher speed than normal (about 25% faster).

The magnetic tapes were made of thin steel and were very heavy. When a bad section of the tape was discovered the engineers punched holes in the steel at either end of the bad patch and the tape drives ignored it. They worked quite well but were not taken up by any other manufacturers, so far as I am aware.

Finally, for those who don’t know, why was the Univac 1103 given such a curious number? Because it and the 1102 were successors to the Univac 1101, the first machine in the series, which was originally given the code name “Task 13” by its designers and 13 is 1101 in binary.

References


(3) Metropolis, N; Howlett, J; Rota, G-C (Eds): “A History of Computing in the Twentieth Century”, Academic Press (1980);
