The Manchester University Small-Scale Experimental Machine

Programmer's Reference Manual

Introduction

The Small-Scale Experimental Machine (SSEM) is the "official" name for the prototype Manchester Mark 1 as it existed when it successfully executed the world's first stored computer program on 21st June 1948. After that date, the machine underwent significant expansion in both capacity and functionality, including the invention of innovations such as the "B"-tube (what we today call instruction-modifier, or index register). The designers of the machine had a policy of making it work stage by stage, rather than building the whole thing before attempting to make it work. This policy is what enabled them to get the machine working at such an early date. The Programmer's Reference given here is for that original minimal machine. There was no formal document at the time - the whole description was, and only needed to be, in the designers' notebooks.

I am grateful to Keith Wood and Andy Molyneux for providing some of the information and wording used in this Programmer's Reference document.

Chris P Burton - The Computer Conservation Society Issue 2 20th August 1997

c.p.b followed by the att sign followed by envex dot demon.co.uk

2 Functional description

2.1 The Store

The Store holds 32 lines, each of 32 binary digits. The lines are numbered 0-31, (from top to bottom when viewed on the monitor screen), and the digits are numbered 0-31 from left to right, with digit 0 being the least significant. Note this arrangement of binary significance, which was commonplace in the early development of serial computers as a consequence of the least significant digit appearing first (to make arithmetic carry circuits simpler) and the usage in engineering that time is often mapped on to a conventional X-axis.

The Store holds both instructions and data - what is often called "Von Neumann" architecture.

2.2 The Accumulator

The Accumulator has a single line of 32 digits, and holds the result of arithmetic operations.

2.3 The Control

The Control normally holds a single line of 32 digits, known as the "Control Instruction", or "CI". The CI is similar to what we would today call the PC or Program Counter, and it contains the address (i.e. the line number) in the Store of an instruction. The CI is always incremented just prior to fetching an instruction for execution, so it typically holds the address of the instruction executed last, rather than to be executed next. Consequently, the instruction in line 1 will be the first to be executed when a program starts.

During the execution of an instruction, the Control transiently holds a second line known as the "Present Instruction" or "PI", containing the actual instruction fetched and being executed. While the machine is not executing instructions, the PI line does not exist.

2.4 Number format

Numbers are 32 bits long. The convention is that the most significant bit is a sign bit; negative numbers are held in complement form. There is no hardware to detect overflow or provide sign extension. Carries out of the top of a word are lost.

2.5 Instruction format

An instruction has two components: a "line number" and a "function number". The line number indicates the line of the store on which the instruction should operate, and the function number indicates the operation to be performed on that line. The "CMP" and "STOP" instructions do not use the line number, and ignore anything in that line of the store..

One line in the store holds one instruction with the following layout:

Line No.	Not used	Func. No.	Not used
0 1 2 3 4	5 6 7 8 9 10 11 12	13 14 15	16 31

Only the eight defined bits are used in an instruction - the unused bits may be used for any purpose by the programmer and will be ignored by the instruction decoder.

2.6 Instruction set

The function codes are tabulated below:

Function number	Binary code	IEE paper mnemonic	"Modern" mnemonic	Description
0	000	s, C	JMP	Copy content of Store line to CI
1	100	c + s, C	JRP	Add content of Store line to CI
2	010	-s, A	LDN	Copy content of Store line, negated, to Accumulator
3	110	a, S	STO	Copy content of Accumulator to Store line
4	001	a-s, A	SUB	Subtract content of Store line from Accumulator
5	101	-	-	Same as function number 4
6	011	Test	CMP	Skip next instruction if content of Accumulator is negative
7	111	Stop	STOP	Light "Stop" neon and halt the machine.

Some further details about instruction operation are given in Appendix 1, and some programming hints in Appendix 2.

2.7 Instruction timings

The SSEM is a serial computer, and the clock speed may be assumed to be 100kc/s, i.e. 10 microseconds per digit. The digit period at 21st June 1948 was probably between 8.5 and 9.5 microseconds, but it was fixed at 10 microseconds later.

An instruction always takes four word times to execute, and each word is 32 bits plus 4 bits in the gap between words. The time for an instruction therefore is 1.44mS, or a computer speed of about 700 instructions per second.

Appendix 1 - Programming details

A1.1 The "JMP" instruction

The line number addresses a line in the store containing a line number to be transferred to CI, i.e. it is an indirect jump in modern terminology. Remember that the whole line is transferred to CI, and that CI is incremented just prior to starting a new instruction. For example:

Line number	Instruction or data	Binary	Comment
...			
6	SUB 19	11001....001..	Accumulator op., subtract contents of line 19
7	CMP	00000....011...	Test sign of Accumulator
8	JMP 10	01010....000...	Jump back to instruction in line 6 if positive
9	STOP	00000....111...	Stop if negative
10	5	10100.....000..	CI for the JMP. One less than the destination line
...			

A1.2 The "JRP" instruction

The line number addresses a line in the Store containing a line number to be added to CI, i.e. an indirect relative jump. The whole contents of the addressed Store line is added to the CI and the result kept in CI. When the next instruction is started the CI is incremented, and the least significant 5 digits used as the address for fetching the instruction. [Note. This seems to be a "luxury" instruction to have in a minimal machine. The reason is that it is trivially easy to implement, and Tom Kilburn was looking towards the imminent increased storage capacity and use of relocatable code for use in subroutines.]

A1.3 Manipulation of CI

As mentioned earlier, the CI is incremented before an instruction is fetched and executed. During program execution, if a CMP instruction finds that the Accumulator is negative, then a Test Flip Flop is set. If the Test Flip Flop has been set, then CI will next be incremented by two instead of by one, and the Test Flip Flop will be reset. Note that only the five least significant bits of CI are used to address a line in the store, so it is possible for a CMP instruction in line 30 to be followed by an instruction either in line 31 (Accumulator positive) or in line 0 (Accumulator negative).

CI is incremented whether instructions are fetched automatically or manually (see A3.3.2). This means that it is not possible to stop a program, perform a manual instruction, then resume the program where it was left. On the other hand, this does provide a mechanism for setting the CI before starting a program.

A1.4 Function bits 13-15 in CI

The SSEM was built with minimal control hardware. The fetch phase of instructions copies a word from the Store tube to the Control tube, putting it on the PI line (see section 2.3 above). This is almost the same operation as the execution phase of a JMP instruction, function 0, where a word is copied from the Store tube to the Control tube, putting it on the CI line. The same hardware is used for both sorts of operation, consequently, bits 13 to 15 of the CI must always be zero so that the decoding hardware performs the correct routing during fetch phases.

Bits 13 to 15 in the CI could become non-zero in the following ways:

a. Spurious bits in a JMP instruction operand line

b. Spurious bits in a JRP instruction operand line, or so created by the addition of the line contents to CI,

c. Allowing the CI to increment such that bits carry into bit 13. This could happen, for example, if a valid instruction in line 31 was followed by a valid instruction in line 0. In that case there will be a carry into bit 5, and if the program continues to cycle from line 31 to line 0 then eventually there will be a carry into bit 13.

The effect of non-zero bits in the function part of CI will normally be to cause the computer to "hang", because the decoding of the bits would not set up a route for the instruction from the Store into the PI. The exception is if the bits are 100 (function number 1), in which case the instruction from the Store is added to the PI left over from the previous instruction. No use is envisaged for this.

Appendix 2 Some programming hints

It is quite an art to create interesting programs with such a small storage capacity and tiny instruction set. However, the example programs HCF1, LONGDIV1 and PROG1 written by the pioneers show that much could be done with so little. Those programs suggest some useful techniques, and further hints are in the following sections. In 1948 nobody had explained that it was sinful to use instructions as data and vice versa.

A2.1 Addition in the Accumulator using the SUB instruction

The "ADD" instruction was not provided in the original machine until August 1948. The following fragment would be needed to add two numbers and place the result back in the Store.

Line number	Instruction or data	Binary	Comment
...			
3	LDN 28	00111....010..	-x to A
4	SUB 29	10111....001...	-x - y to A {= -(x + y) }
5	STO 30	01111....110...	-(x + y) to Result
6	LDN 30	01111....010...	-(-(x + y)) to A {= (x + y) }
7	STO 30	01111....110..	x + y to Result
...			
28	x		First number, x
29	y		Second number, y
30	x+y		Result, x + y

A2.2 Small constants

The indirect jump location used in JMP and JRP instructions can sometimes be used as a small constant. For example:

Line number	Instruction or data	Binary	Comment
1	LDN 30	01111....010..	First instruction in program
2	STO 30	01111....110..	Top of loop
....			
9	SUB 20	00101....001..	Decrement Accumulator
10	CMP	00000....011...	Test Accumulator
11	JMP 20	00101....000...	Loop back to line 2 if positive
...			else continue
20	1	10000....000	Constant with different uses
...			

A2.3 Indexing, or Instruction Modification

A common operation is to add a small integer to a word. An example would be where the line number part of an instruction is to be indexed. The small integer should be stored as its complement as in the following fragment.

Line number	Instruction or data	Binary	Comment
...			
6	SUB 19	11001....001..	Subtraction operation to be indexed through lines 19, 20, 21 etc.
....			
12	LDN 6	01100....010...	Complement of instruction to A
13	STO 0	00000....110...	Store in temporary space
14	LDN 0	00000....010...	Instruction to A
15	SUB 18	01001....001...	Modify
16	STO 6	01100....110...	Instruction indexed
17	JMP ..	.............000...	Next ...
18	-1	111111111....	-1
...			

A2.4 Counting

Similarly, counting is a frequent program requirement. Space can be conserved by keeping the counter in the upper part of an instruction, as in the following code.

Line number	Instruction or data	Binary	Comment
...			
6	LDN 21	10101....010....000	Instruction: counter in top 3 bits
....			
12	LDN 6	01100....010...	Complement of instruction and counter to A
13	STO 0	00000....110...	Store in temporary space
14	LDN 0	00000....010...	Instruction and counter to A
15	SUB 20	00101....001...	Increment counter
16	STO 6	01100....110...	Counter incremented
17	CMP	00000....011..	End of count?
18	JMP ..	000	No
19	JMP ..	000	Yes
20	-1 in top 3 bits	00000....0000111	-1 at top of word
...			

Appendix 3 - Physical Description

A complete contemporary description of the machine is given in a paper published in the Proceedings of the Institution of Electrical Engineers. The paper was submitted in March 1950, and published as:

Williams F.C., Kilburn T. and Tootill G.C., "Universal High-Speed Digital Computers: A Small-Scale Experimental Machine", *Proc IEE*, Vol. **98**, Part II, No 61, February 1951, pp. 13-28.

The paper will be available for downloading from the Computer Conservation Society archive site at <ftp://ftp.cs.man.ac.uk/CCS-Archive>.

A thorough, detailed understanding of that paper should be enough to enable anyone to write a program for the SSEM. However, there are subtleties of behaviour which are not made explicit. Furthermore, the exact way that the "input and output" controls appear to the user is not obvious. This appendix addresses those details where they have not already been explained earlier in this Programmer's Reference. There should be no contradictions between the IEE paper and this document, except where specifically mentioned herein.

This document mostly uses the terminology used in 1948, though some modern terminology is used where essential. For example it refers to information being held in a "line" where today it would probably be more usual to say "location" or "word". The original word "store" is used rather than the modern anthropomorphic word "memory".

A3.1 The Monitor

There is a fourth cathode ray tube not used for storage but as a Monitor tube. Three interlocking push buttons allow the Monitor to be connected to any one of the three storage tubes. The Monitor always displays 32 lines of 32 digits. In the case of the Store, this is a natural display of the whole contents of the Store. In the case of the Accumulator, which only contains one line of data, that line is repeated on all 32 lines of the monitor. In the case of the Control: while the machine is running the CI and PI appear on alternate lines of the Monitor, but if the machine is not running, then only the CI appears on all 32 lines.

The digits are represented on the monitor as a small dot for a zero, and a horizontal dash for a one. Any line of the Store is addressable for some operation and is known as the Action Line. On the monitor, the Action Line appears brightened. Manual control of the Action Line is described later in paragraph A.3.3.2.1.

A3.2 The "Stop" indicator.

A neon indicator lamp is situated near to the Monitor. It is illuminated when a Stop instruction has been decoded (see later).

A3.3 Control Switches.

There are four panels containing switches. Their correct operation by the user is essential if the computer is to operate according to this Programmer's Reference. The effect of some incorrect operations, but not all, will be explained.

A3.3.1 The Typewriter.

The panel contains 40 push-buttons arranged as eight columns of five buttons. The buttons are numbered 0 to 31 starting at top left and running down a column and then down succeeding columns from left to right. Only the first 32 buttons are connected.

The Typewriter is normally used while the machine is at Stop, i.e. not executing instructions. If the Write/Erase Switch (see below) is at "Write", then pushing a typewriter button causes a "one" to be written into the corresponding digit position of the Action Line. If the Write/Erase Switch is at "Erase", then a zero is written into the position. If more than one button is pressed simultaneously, then the result is usually but not always ineffective.

If the machine is running, then operation of the Typewriter will affect all the Action Lines encountered while the button is pressed.

A3.3.2 The Staticisor ("Stat") Switch.

There is a toggle switch labeled "Auto/Man". The switch determines the operation of five "Line" or "L-Stat" toggle switches, and three "Function" or "F-Stat" toggle switches. The machine behaves according to the setting of these switches as follows:

A3.3.2.1 Switch at either "Man" or "Auto", computer not running.

> The "Line" switches select which line in the store is to be the Action Line. Any line-oriented operation e.g. Typewriter and KLC (see later), affects the Action Line. A "Line" switch is down for on, up for off. The "Function" switches do nothing.

A3.3.2.2 Switch at "Manual", computer running.

> The instruction represented by the setting of the "Line" and "Function" switches is executed repeatedly. "Single Step" mode (see later) is a special case of computer running where just one instruction is executed per single step. CI will be incremented.

A3.3.2.3 Switch at "Auto", computer running.

> Instructions are fetched from the Store and executed. The instructions fetched, as well as the CI itself, pass through the "Line" and "Function" switches on the way to the staticisor; accordingly they must all be set to "on". If any switch is "off", then the corresponding digit in the instruction or coming from CI is forced to zero, with the predictable effect.

If the Typewriter is operated, in conjunction with the Write/Erase Switch, then all lines being accessed by the running program will be appropriately affected. This may corrupt the program, for example if the function or line fields of instructions are changed.

A3.3.3 Write/Erase Switch.

As mentioned in the description of the Typewriter, this switch determines whether a "one" or a "zero" is "typed" into the Store in the selected line. It must be in the "Write" position while the computer is running. Should it be left in the "Erase" position, then all action lines would be corrupted.

A3.3.4 Storage clearing keys.

There is a set of key switches each of which clears the appropriate storage locations to zeroes. Their use while the computer is running is unpredictable. They are non-locking, i.e. return to the neutral position when released.

A3.3.4.1 Key Line Clear (KLC)

Clears the Action Line. If this key is operated while a program is running, then all lines being accessed by the program will be cleared, thereby corrupting the program.

A3.3.4.2 Key Store Clear (KSC)

Clears the whole Store. Obliterates any existing program, whether the computer is running or not.

A3.3.4.3 Key Control Clear (KCC)

Despite its name this key clears both the Control tube and the Accumulator tube. If the key is operated while a program is running, then the result is not predictable, and in some circumstances the program will be corrupted.

A3.3.4.4 KAC, KBC, KEC, KMC

Although physically present, these switches are not connected.

A3.3.5 Monitor selector

These interlocking push buttons enable the Monitor tube to be connected to the output of one of the three Storage tubes (see A3.1 above)

A3.3.6 CS switch.

The Completion Signal (or Prepulse) toggle switch, CS, has an up position marked "Stop" and a down position marked "Run". (The term "Completion Signal" is a hangover from the early days when the designers thought of the pulse which initiates a new instruction as actually completing the previous instruction. In June 1948 they were in transition to calling it the "Prepulse").

In the "Stop" position, prepulses are inhibited, so the machine does not run and execute instructions automatically. In the "Run" position, prepulses are generated automatically at the end of each instruction, so causing the next instruction to be fetched and executed. Prepulses are inhibited if a Stop instruction has been encountered and the "Stop" neon is illuminated.

A3.3.7 Key Completion, (KC) also known as Key Single Prepulse (KSP).

This is effectively the non-locking Single Step key. It is effective when the CS switch is at Stop, and causes one prepulse to be generated. Thus one instruction will be fetched and executed. If such instruction is not a Stop instruction, then incidentally the "Stop" neon, if illuminated, will be extinguished.