# City and Guilds Mnemonic Code

*By Andrew Herbert*

Last Revised 9th May 2015

## Introduction

City and Guilds Mnemonic Code is a machine independent assembly code created by the City and Guilds Institute to enable practical programming questions to be included in the examination for their basic and advanced Certificates for Computer Personnel (courses 319 and 320 respectively). The courses were introduced in 1964 and revised in 1968 when course 319 was renamed "Certificate in Computer Programming and Information Processing".

Very little trace of Mnemonic Code remains: searching the World Wide Web throws up a few reminiscences of people who used it as students, but no detail of the language itself. There is an ICL 1900 implementation (#XMS3) being resurrected by Brian Spoor, who has obtained a printed course specification, copyright date 1967, which contains a language definition.


I have been fortunate to obtain a copy of the Elliott 903 version via Mr Chris Pugh-Jones who contacted me after a previous article I wrote about Elliott 903 software for *Resurrection*. Mr Pugh-Jones has a collection of paper tapes for both Elliott 803 and 903 computers dating from his student days and one of these had a written label "MNEMONIC CODE CITY & GUILDS 319 903 VERSION" and a punched legible heading giving the same information plus "ISS 1" – i.e., Issue 1. The tape was accompanied by a typescript sheet listing the program entry points.

Sadly the tape did not read under initial instructions on a real 903. Investigations by myself and Terry Froggatt using our 903 simulators and associated tools found the tape to be a loader followed by a store image, but with a corrupted initial section (5 missing rows). Terry corrected this and we then had a tape that would read in and execute programs successfully. The loader was different to the other Elliott loaders we have encountered. To document the program we both set about analysing it. We now have a source that can be assembled using Elliott 903 Symbolic Input Routine to produce an identical store image to the original tape.

Subsequently, Terry obtained two tapes labelled "C & G Compiler", master and copy respectively, from the Museum of Scotland. Apart from the omission of a legible header these are identical to the corrected Pugh-Jones tape and so it can be reasonably assumed we have recovered the definitive Elliott 903 system for running City and Guilds Mnemonic Code.

The missing rows on the original tape are a mystery – the rows are physically missing, rather than damaged or unreadable suggesting some error when the tape was originally punched.

Mr Pugh-Jones also found in his archives a printed specification of the "Revised Mnemonic Code (1968)" which defines an extended order code compared to that in the document circulated by Brian Spoor so it is assumed this earlier document relates to the original 1964 specification.

**Mnemonic Code**

Mnemonic Code comprises assembler directives, machine orders (instructions) and numbers. Data are held internally in floating point form.

The code assumes a computer with 1,000 words of store. Each word can contain an order, a (floating point) number or a character for input/output. The architecture requires integers to be held with at least 7 digits precision.

The order code is of the single address form. The first 10 locations of store are reserved for registers. Location 0 always contains the value 0; it cannot be updated. Location 1 is the accumulator (A). Location 4 is used to hold the return link when entering a subroutine. All the registers can be referenced as index registers.

In addition to the index registers there is an addition control register (C) that contains the address of the next instruction to be executed. Unless updated by a jump instruction C is automatically incremented after each instruction is executed.

An order comprises four fields expressed as a seven digit number:

| | | | |
|---|---|---|---|
| F F | Q | n n n | m |

1. Order number F
2. Address n
3. Modifier m
4. Trace Q

The order number specifies the function to be performed as listed below. The machine is unusual in have orders for mathematical functions, but these could be thought of as "extracodes" as found in machines such Atlas.

The address is a natural number in the range 0-999.

The modifier is a natural number in the range 0-9 and, for orders that support modification, the effective address is computed by adding the content of the address field to the content of the nominated index register (i.e., store location).

Thus, using 0 as the modifier yields the address field unmodified. Note that, since the contents of store comprise floating-point numbers, the value of the index register has to be rounded before it is added to the address field.

The interpretation of the trace field is implementation dependent, but indicates that some sort of diagnostic output should be produced whenever the instruction is executed.

| Numeric Function | Mnemonic | Operation | Remarks |
|---|---|---|---|
| 00 | `LDA n, m` | A:=(n+(m)) | Load operand into cleared accumulator |
| 01 | `ADD n, m` | A:=(A)+(n+(m)) | Add operand |
| 02 | `SUB n, m` | A:=(A)-(n+(m)) | Subtract operand |
| 03 | `MLT n, m` | A:=(A)*(n+(m)) | Multiply by operand |
| 04 | `DIV n, m` | A:=(A)/(n+(m)) | Divide by operand |
| 10 | `LDAN n` | A:=n | Load integer |
| 11 | `ADDN n` | A:=(A)+n | Add integer |
| 12 | `SUBN n` | A:=(A)-n | Subtract integer |
| 13 | `MLTN n` | A:=(A)*n | Multiply by integer |
| 14 | `DIVN n` | A:=(A)/n | Divide by integer |
| 20 | `STA n,m` | n+(m):=(A) | Store (A) without clearing accumulator |
| 30 | `JUN n, m` | C:=n+(m) | Jump unconditionally |
| 31 | `JGR n, m` | If (A)>0 C:=n+(m) | Jump if A>0 |
| 32 | `JEQ n,m` | If (A)=0 C:=n+(m) | Jump if A=0 |
| 33 | `JSR n, m` | 4:=Link; C:=n+(m) | Set link and jump; link is the address of the instruction following JSR |
| 34 | `JST n, m` | Wait; C:=n+(m) | Wait; jump when start button operated |
| 40 | `SQT n, m` | A:=sqrt(A) | If (A)<0, jump to n+(m) |
| 41 | `EXP n, m` | A:=exp(A) | If (A) too large jump to n+(m) |
| 42 | `LGN n, m` | A:=ln(A) | If (A)<0, jump to n+(m) |
| 43 | `SIN` | A:=sin(A) | (A) in radians |
| 44 | `COS` | A:=cos(A) | (A) in radians |
| 45 | `ARC` | A:=arctan(A) | (A) in radians |
| 46 | `ENT` | A:=entier(A) | Integral part of (A) to A |
| 50 | `RCT n, m` | character to n+(m) | Read single character from tape |
| 51 | `PCT n, m` | (n+(m)) to tape | Punch single character to tape |
| 52 | `RNT n, m` | number to A | Read number from tape; jump to n+(m) if error in the number |
| 53 | `PNT n, m` | (A) to tape | Print signed number in A to tape with n integral and m fractional digits |

| | | | |
|---|---|---|---|
| 54 | PNL | | Punch the characters for new line |
| 60 | RCC n, m | characters to n+(m) | Read characters from card |
| 61 | PCC n, m | (n+(m)) to card | Punch characters to card |
| 62 | RNC n, m | number to A | Read number from card; jump to n+(m) if error in the number |
| 63 | PNC n, m | (A) to tape | Print signed number in A on to card with n integral and m fractional digits |

If an instruction is followed by the letter Q, the trace digit will be set.

The 1968 specification changes the specification of some orders and adds additional ones, mostly concerned with a richer input/output model as tabulated below.

| Numeric Function | Mnemonic | Operation | Remarks |
|---|---|---|---|
| 10 | LDAN n | A:=n+(m) | Load integer |
| 11 | ADDN n, m | A:=(A)+n+(m) | Add integer |
| 12 | SUBN n, m | A:=(A)-n+(m)] | Subtract integer |
| 13 | MLTN n, m | A:=(A)*n+(m) | Multiply by integer |
| 14 | DIVN n, m | A:=(A)/n+(m)] | Divide by integer |
| 31 | JEQ n, m | If (A)=0, C:=n+(m) | Jump if (A)=0 |
| 32 | JNE n, m | If (A)<>0, C:=n+(m) | Jump if (A)<>0 |
| 33 | JLE n, m | If (A)<=0, C:=n+(m) | Jump if (A)<=0 |
| 34 | JGE n, m | If (A)>=0, C:=n+(m) | Jump if (A)>=0 |
| 35 | JLT n, m | If (A)<0, C:=n+(m) | Jump if (A)<0 |
| 36 | JGR n, m | If (A)>0, C:=n+(m) | Jump if (A)>0 |
| 37 | JSR n, m | As above | |
| 38 | JST n, m | As above | |
| 39 | LOP n, m | If (5)>0 after (5)-1, C:=n+(m) | Jump if after subtracting 1 from location 5, (5)>0 |
| 50 | ARD n, m | | Allocate input device n+(m) to program |
| 51 | AWD n, m | | Allocate output device n+(m) to program |
| 52 | RNA n, m | A:=number | Read number; jump to n+(m) if error in number |
| 53 | WNA n, m | Write (A) | Write (A) with n integral and m fractions digits. If n and m are zero, floating point is implied |
| 60 | RCH m, n | Read one character to n+(m) | Read character to store |
| 61 | WCH n, m | Write one character from n+(m) | Write character from store |

| | | | |
|---|---|---|---|
| 62 | RNB n, m | Read characters to n+(m) | Read block of characters into locations starting at n+(m) |
| 63 | WNB n, m | Write characters from n+(m) | Write block of characters from locations starting at n+(m) |
| 64 | WNL n, m | Write n+(m) newlines | |
| 65 | WSS n, m | Write n+(m) spaces | |
| 66 | CNN n, m | Convert character string to number | String starts at location n+(m); result put in A |
| 67 | CNC n, m | Convert number to character string | Number is in (A); string starts in location n+(m) onwards |
| 70 | ACB n, m | Access block n+(m) | For direct access devices n+(m) defines the sector; for other devices the effect is to skip n+(m) blocks |
| 71 | BSP n, m | Backspace n+(m) blocks | |
| 72 | RWD | Rewind device | |
| 99 | STOP | Stop | Return control to operating system |

The revised specification also defines a collating sequence for characters since these can be manipulated as numbers in this version of the language.

There are 4 directives to the assembler:

(TITLE) Reads the next line as the program title and copies to the output.
(STORE n) Stores the following program from location n onwards.
(WAIT) Pauses input, awaiting a further program tape to be input.
(EXECUTE n) Marks program complete and sets location n as the location at which execution should commence.

Note there are no facilities for giving textual names to store locations (i.e., labels) or any form of relative addresses: all addresses in a program are absolute. Nor are there facilities for literal addresses – i.e., writing a numeric value in the address field and having the assembler automatically allocate store to to hold the value. This makes program structure very fragile – adding or removing an instruction or number can require wholesale editing. This is a major omission and great inconvenience to the programmer. The lack of any form of comment facility further compounds the opacity of programs.

**The Elliott Implementation**

Terry Froggatt has a sale brochure dating from 1967 that mentions the availability of "City and Guilds 319 compiler" so it is assumed the system was distributed by Elliotts. Whether the program originated in-house or not is unknown. There is circumstantial evidence that the compiler may have originated on an Elliott 920A, a predecessor of the 903/920B.

The Elliott implementation of Mnemonic Code is very straightforward, comprising an assembler, an interpreter and an editor. There are two primary entry points: one to read in and assemble a Mnemonic Code source paper tape (entry at location 8) and the second to run a previously assembled program (entry at location 9). These are set up by setting the address on the 903 control panel keys and pushing the JUMP button. There are secondary entry points to enable a program made up of several tapes to be read in (Interrupt 1), to resume execution after a stop (Interrupt 2) and to enable execution tracing for debugging purposes (Interrupt 3). These are exercised by pressing the appropriate manual interrupt button on the control panel.

**Example program**

```
(TITLE)
SIMPLE TEST
(STORE 12)
LDAN 1
ARC 16
MLTN 4
PNT 1,6
JST
(EXECUTE 12)
```

**Output**

```
SIMPLE TEST
 3.141593
```

All input and output is in Elliott 903 telecode, an ancestor of ASCII. The language implemented is the simpler 1964 version. The card reader and punch instructions are not supported as the 903 is a paper tape based machine.

The interpreter has an internal character code which is neither 900 telecode (with or without parity) or the 6 bit code used by other Elliott software (e.g., the Algol compiler). Nor does the code comply with the collating sequence given in the 1968 specification. It does however show some similarity with the earlier flexowriter code used on the Elliott 503 and 920A.

The assembler makes comprehensive checks of the input source, but halts after each error report, which consists of an error number and the store location in which the next instruction or datum will be loaded. In principle the operator can re-enter the assembler to move past the error but this becomes a tedious procedure especially if there are a lot of errors in the input.

**Typical error report**

```
ERR  2    12
```

The interpreter also makes comprehensive checks to ensure that programs do not address outside of the available store and also that no attempt is made to treat instructions as data or vice versa. (The latter is not a requirement in any of the language specifications and rules out writing any sort of assembler or loader in Mnemonic Code, arguably a significant restriction). Execution errors are reported as a numerical error type code followed by the location of the failing instruction.

Internally the interpreter does not use the function codes given in the language specification: instructions cannot be processed as data so this is not visible externally.

With tracing is enabled, instructions with the Q field set, when executed, produce a line of output showing the current value of the Control Register (i.e., program counter) and the Accumulator.

**Typical trace output**

```
Q   16   9.000000?+00
Q   16   8.000000?+00
Q   16   7.000000?+00
Q   16   6.000000?+00
Q   16   5.000000?+00
Q   16   4.000000?+00
```

Floating point arithmetic in the interpreter uses what appears to be an early issue of the standard Elliott QF/QFMATH/QFINOUT packages.

The editor is effectively a separate program co-resident in store with the assembler/interpreter and has entry points to read in a steering tape followed reading in a source tape for editing.

The editor would appear to be based on the Elliott EDIT utility. It has simple commands to copy, insert and delete text, searching by line or character string. Unlike EDIT it lacks the facility to read back and check the output tape has been punched correctly.

**Concluding Remarks**

Following recovery of a paper tape for the Elliott 903 implementation of Mnemonic Code it has been possible to reverse engineer the system and compare to the City and Guilds Institute specification. There are some minor divergences but the system is clearly suitable for running 1964 C&G Mnemonic Code programs on a 903.

Mnemonic Code itself is an impoverished assembly language and the choice of floating point as the basic data format is unconventional. The code can at best be used for numerical calculations. Any sort of textual or logical processing is inconvenient if not nigh on impossible. It is not surprising that has almost

disappeared without trace, and indeed the author feels some guilt at resurrecting it.