X4. Software and sample programs. Give an indication of the programming language facilities, assemblers/compilers, program libraries, applications packages, etc., for the machine (or family of machines). Give annotated sample fragments of programs to illustrate various software features. Give a reference to any machines still in working order, and/or to existing simulators that can run (for example) on a modern PC. Give accurate references to the sources of all information – see X5 below. (Contributes to database blocks D5, D6, D7).

E5X4. Elliott 900 Series: Software and Sample Programs.

18-bit machines.

Symbolic Input Routine (SIR).

Before SIR, there was a "Non-symbolic Assembler" called "T2", which was still in use in late 1966. But even T2 accepted decimal notation for function codes, addresses, and constants; in contrast to the early software for microprocessors a decade later, whose monitor programs could only be programmed in hexadecimal.

SIR is what nowadays would be called a "Symbolic Assembler".
To quote a brochure of the era, "Elliott MCS 920B Computer Programming" [4]:

> "SIR enables machine code programs to be written with addresses referred to by invented names. It simplifies editing, as the insertion or removal of instructions does not require the wholesale modification of addresses in the rest of the program. Also it is easier to spot mistakes in names than in numbers. Store locations must be reserved by the programmer by labelling them with invented names. The addresses are then allocated to the locations at assembly time, thus removing from the programmer the burden of organising store space."

The Elliott 900-series 18-bit machines do not have what nowadays would be called an "immediate address mode". The "N" part of an instruction usually holds the *address* of the operand, not the *value* of the operand (the exception being shifts, where the shift distance is given).

As explained in the "Elliott 900 Technical Manual, Volume 2A, Programming Information" [14] part 1 section 1, SIR implements this facility in software, sharing several uses of the same constant:

> [One of the] "Advantages of Programming in SIR" [is that] "The programmer can write an instruction using a constant in the address part without having to specify where that constant is stored", [using a] "LITERAL – a constant appearing as the address part of an instruction". "The constant is placed into the address part of the

instruction. During assembly, the assembler on reading the end of program symbol % allocates a store location to the constant, places the constant therein and finally inserts the address of this location in all of the instructions using this constant".

The standard version of SIR can operate in load-and-go mode. SIR occupies from around location 5500 upwards of the 8192-word store; and it loads the user's program upwards from low memory whilst building the symbol table downwards from itself. Forward references to a constant or unresolved symbol are chained together through their "N" fields. The standard version of SIR can also produce relocatable binary tapes.

For embedded applications, "2-Pass SIR" is available. On the first pass of the source tape, just a symbol table is built in store. On the second pass, the source tape is converted to an absolute binary tape line-by-line, suitable for reading under initial instructions. This enables a program to be written occupying every store location, apart from those used for the registers, initial instructions, and 2-Pass SIR's own sum-checking loader (a mere 13 words, and even these can be used for variables, once the program is loaded).

MASIR was a later development of the standard version of SIR, providing Macro expansion facilities. It also provided some extra facilities to facilitate programming machines that had more than 8192 words of store.

SIR Subroutines.

The standard SIR library includes routines for:

- Single-length input & output,
- Single-length fixed-point mathematical functions:
  - Sine & Cosine, Square Root, ArcTangent,
  - Logarithm and Exponent,
- Interpreted double-length fixed-point arithmetic, `QDLA`,
  - including double-length input & output,
- Double-length fixed-point mathematical functions,
- Interpreted floating-point arithmetic, `QF`,
- Floating-point input & output, `QFIN/QFOUT`,
- Floating-point mathematical functions, `QFMATH`,
- Magnetic tape handling.

Two sample SIR subroutines are given below. *I've chosen them because I happen to have them available on modern media.* They are similar to those in the 903 library generally issued, but have been optimised for embedded system use. One difference is that the standard library assumes that the link location and entry point are in consecutive store locations, whereas the examples below give them separate labels. This is to permit easy separation of code and data in applications requiring a runtime store sumcheck. The SIR notation "*>N*" used below means "reserve *N* locations here as uninitialised workspace".

Sample Sine & Cosine Subroutine.

The first example is a subroutine for finding the Sine or Cosine of an angle, (or both). The maximum number of obeyed instructions is 21 for Sine or 22 for Cosine. The maximum error is believed to be $\pm 2^{-17}$.

Whichever function is required, the Accumulator (viewed as holding a signed pure fraction) should contain the angle, θ, scaled by 180˚ or π radians, on entering. Thus:

the sign bit represents *minus* 180˚ or π radians,
the next bit represents *plus* 90˚ or π/2 radians,
the third bit represents *plus* 45˚ or π/4 radians, and so on.

This fixed-point representation of angles exploits the natural wrap-round properties of the underlying hardware. This representation was subsequently named "BAMS", the Binary Angle Measurement System", in some quarters, but not in 900-series software.

The instruction pair:

```
            11      SINL
            8       SINE
```

places ½ Sin θ in the Accumulator (viewed as holding a signed pure fraction);

```
            11      COSL
            8       COSE
```

places ½ Cos θ in the Accumulator (viewed as holding a signed pure fraction);

```
            11      SICOL
            8       SICOE
```

places ½ Sin θ in store location `SICOS`, and places ½ Cos θ in the Accumulator.

```
((SINE & COSINE SUBROUTINE, 1/12/71)

[SINL SINE COSL COSE SICOL SICOE SICOS]

SICOL       >1
SICOS       >1
ANGLE       >1
SINL COSL
LINK        >1
X           >1
XSQ         >1

SICOE       5       ANGLE
            11      SINL
            8       SINE
            5       SICOS
            4       SICOL
            5       COSL
            4       ANGLE

COSE        1       +.5

SINE        1       +.25
            0       +0
            9       ;+2
            0       +1

            14      1
            9       COS

SIN         1       -.5
            5       X
            12      X
            5       XSQ
            12      -307
            1       +5223
```

```
              12      XSQ
              1       -42334
              12      XSQ
              1       +102944
              12      X
              /8      END

COS           1       +.5
              5       X
              12      X
              5       XSQ
              12      -1367
              1       +16634
              12      XSQ
              1       -80852
              12      XSQ
              1       +.5
              /8      END

END           2       +0
              0       LINK
              /8      1
```

Sample Output Subroutine.

The second example is an output routine, SLOP (Single-Length OutPut), which converts the 18-bit number in the accumulator into a sequence of 7-bit characters, and outputs these by calling CHOP (CHaracter OutPut). CHOP is not given here, but in its simplest form, it just adds even parity to the 8th track and sends the character to the TeleType.

SLOP is called by the instruction pair:

```
              11      SLOPL
              8       SLOPE+N
```

were *N* determines which entry point is used and hence which format is produced:

N=0	unsigned integer with leading-zero suppression,
N=1	signed integer with leading-zero suppression,
N=2	signed pure fraction with six digits, unrounded,
N=3	unsigned octal number with six digits, prefixed by "&",
N=4	alphanumeric group, a three-character string, prefixed "£" or "#".

```
((SINGLE-LENGTH OUTPUT SUBROUTINE, 9/8/71))

(Outputs the contents of the accumulator as an
unsigned integer, signed integer, signed fraction,
octal number, or alphanumeric group)

[SLOPL SLOPE CHOPL CHOPE]

SLOPL         >1      (Link)
INTGR         >1      (Remainder of number being printed)
FLAG          >1      (Indicates format)
DIGCT         >1      (Count of digits printed)
SUBCT         >1      (Count of subtractions)
ZERMK         >1      (+0 until nonzero digit found)

SLOPT         -100000 -10000 -1000 -100 -10 -1

SLOPE         8       UINT
(+1)          8       SINT
(+2)          8       FRAC1
(+3)          8       OCTA1

(+4)          7       SINT
              5       INTGR
              4       -35        (-[#])
```

```
             8       ALPH1

UINT         5       INTGR
             4       +0
             5       FLAG
             8       MEET1+2

SINT         5       INTGR
             4       +0
             8       ;+3

FRAC1        5       INTGR
             4       +46         ([.])
             5       FLAG

             4       INTGR
             9       ;+3
             4       +43         ([+])
             8       ;+5
             2       +0
             9       OCTA1       (Jump iff &400000)
             5       INTGR
             4       +45         ([-])
             11      CHOPL
             8       CHOPE       (Print sign)

             4       FLAG
             7       MEET1+2
             8       MEET1

OCTA1        5       INTGR
             4       -38         (-[&])
ALPH1        5       FLAG
             2       +0

MEET1        11      CHOPL
             8       CHOPE       (Print [.], [&], or [#])

(+2)         5       ZERMK
             4       -5
             5       DIGCT

NEXT         4       FLAG
             9       OCALP
             7       INT2

(FRAC2)      4       INTGR
             12      +10
             3       INTGR
             8       MEET2

OCALP        2       -35
             7       ALPH2

(OCTA2)      2       INTGR
             4       INTGR
             14      3
             5       INTGR
             6       +7
             8       MEET2

ALPH2        4       INTGR
             7       EXIT
             5       ZERMK
             0       +0
             14      6
             5       INTGR

             4       ZERMK
             14      8180
```

```
              6       &77
              1       -1
              7       ;+2
              1       -61
              1       +94
              8       MEET2+1

INT2          5       SUBCT
              0       DIGCT
              4       INTGR
              9       ;+4         (Jump iff > 131071)
              /1      SLOPT+5
              9       ;+7
              8       ;+2
              /1      SLOPT+5
              5       INTGR
              10      SUBCT
              10      ZERMK
              8       ;-8

              4       ZERMK
              7       ;+5
              4       SUBCT

MEET2         1       +48         (Number+48 = [Number])
(+1)          11      CHOPL
              8       CHOPE       (Print digit, <S>, or Alphanumeric)

              10      DIGCT
              4       DIGCT
              9       NEXT
              10      ZERMK
              7       NEXT

EXIT          0       SLOPL
              /8      1
```

ALGOL Compiler.

To quote the brochure “Elliott MCS 920B Computer Programming” [4] again:

> “ALGOL is the international scientific-oriented language that allows programs to be written in a combination of English words and mathematical expressions. The saving in programming and proving time through using ALGOL is very great, whereas the increase in computation time is less significant. ALGOL greatly increases the general purpose computing power of the MCS 920, and also permits the interchange of programs with different computers. With compatibility in mind, the IFIP subset of ALGOL 60 was chosen for implementation on the 920.”
>
> “Compilation is two-pass, i.e. the ALGOL program is first put through a translation stage giving an Intermediate Code on paper tape, which is re-input by a loader routine, and the program is then executed by an interpreter which interprets the Intermediate Code. There is a special facility which allows ALGOL programs to call up procedures written in SIR code. It is valuable, amongst other things, for speeding up matrix operations and making use of special peripheral equipment.”

The Intermediate Code is “reverse-Polish”. The same technique, of translating into a Virtual Machine code, which is then interpreted, is used nowadays, to run “Java” programs across the Internet.

The minimum configuration for running ALGOL 60 is a 920B or 903. *I believe that the Flexowriter-code version would also run on a 920A.* Only the standard 8192-word store is required. If a 16384-word store is available, there is a "load and go" version, where the translator and interpreter occupy an 8192-word store module each, (which avoids a considerable amount of paper-tape rewinding), and a "long program" version. See the "Elliott 900 Technical Manual, Volume 2A, Programming Information" [14] part 1 section 2.

FORTRAN Compiler.

There are two FORTRAN Compilers for the 900-series 18-bit machines.

The first is described in the "Elliott 900 Technical Manual, Volume 2A, Programming Information" [14] part 1 section 3:

> "The 900 FORTRAN system described in this document is designed for use on a 903 computer" (or 920B) "with 8192 words of store". "The language incorporates most of the features of basic ASA FORTRAN. It incorporates, in addition, certain features of full ASA FORTRAN." "The compiler translates FORTRAN programs into SIR. The resultant SIR program is subsequently translated and run in conjunction with a special FORTRAN functions package."

As with ALGOL, if a 16384-word store is available, there is a "load and go" version, where the translator and assembler occupy an 8192-word store module each, (which avoids a considerable amount of paper-tape rewinding), and a "long program" version.

The second is described in "905 FORTRAN" [16]:

> "905 FORTRAN can be used on any 905 Series 18-bit machine which includes teleprinter, punch and reader facilities and has a minimum store size of 16K". "905 FORTRAN is ASA standard FORTRAN, as defined in USASI document X3-9-1966 with the extensions and restrictions [given in Appendix 2]".
>
> 905 FORTRAN and MASIR use the same inter-module calling conventions. 905 FORTRAN can be run in a stand-alone paper tape environment, or within the FAS (magnetic tape) or RADOS (disk) operating systems [page 107].

The 905 version of the compiler implements, for example, Double Precision and Complex Arithmetic, which are not available in the 903 version of the compiler. *Page 82 of [16] implies that programs compiled with 905 FORTRAN on a 920C/905 may subsequently be run on a 920B/903 (provided that any assembly-code inserts avoid 920C/905-specific instructions).*

PERT Package.

Although the 900-Series computers were designed primarily for embedded and on-line applications and for scientific use, the PERT package deserves a mention, as possibly the first time (October 1967) that project planning software became available to the people doing the project. This was a precursor to some of the desktop "Office" software available nowadays, but it used considerably less computing power.

900 PERT is described in the "Elliott 900 Technical Manual, Volume 2C, Programming Information" [15] part 6 section 1:

> "This is a program for performing time analysis on a project."
> "In the planning and analysis of a project, one of the most useful approaches is to draw a network diagram. This diagram is a schematic representation of the relationship between various parts of the project in the form of a network of points in time (representing events) joined by straight lines (representing activities)."
>
> "Using the critical path analysis technique, a time analysis can be performed on the network. This will schedule each activity in the project, and will identify those activities which affect the scheduled finish date of the project, i.e. are critical. Also this technique will allow the network to be reviewed as the project proceeds."

The examples given in the manual [15] show dates in "14APR67" format. No millennium compliance statement is offered. Maybe I should try it out.

CORAL Compilers.

Two completely independent CORAL 66 compilers were produced for the 900-series 18-bit computers.

The first, 900 CORAL, was produced by what was then Marconi-Elliott Computer Systems Ltd., and it is described in [17]:

> Compilation can be performed on a 920B/903 or later machine, having at least a 16384-word store. The compiled code uses the 920B/903's relative addressing mode throughout, so programs can be run on any 920B/903 or later machine, with 8192 words of store or more, as required by the application.
>
> The compiler uses a smaller first pass to perform macro expansion and to convert the expanded source into an Intermediate Code, and a larger second pass to generate a Relocatable Binary tape from this, which is then read into store using a relocating loader.

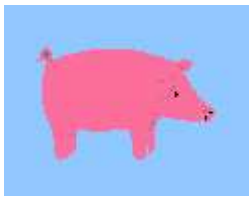
The second, 920C CORAL, was produced by CAP (Reading) Ltd., for the Royal Aircraft Establishment (Farnborough), *I believe for use in the Nimrod mark II aircraft,* and it is described in [18]:

Compilation requires at least a 920C/905 or later machine, having at least a 16384-word store. The compiled code uses the 920C/905's absolute addressing mode for programs needing more than an 8192-word store, but smaller programs can be run on a 920B/903.

The compiler uses several passes:
an optional macro expansion pass,
pass 1A which performs syntax checks,
pass 1B which performs semantic checks,
pass 2 which generates the object code,
a relocating linking loader,
which loads into store, or punches an absolute binary tape.

An equivalent cross-compiler, hosted on the GEC 4080, but still producing 900-series code, was also produced.

BASIC Interpreter.



This was produced for the Elliott Computer Users Association in 1978, by which date many research organisations were giving their 903 machines second-hand to local schools. It runs on any 900-series 18-bit machine that can read all 8 tracks of a paper tape.

It provides the usual interactive editing of the BASIC program text, simple variables `A` to `Z` and `A0` to `Z9`, arrays `A(..)` to `Z(..)` with one or two dimensions, floating-point arithmetic with slightly wider range and better accuracy than ALGOL or FORTRAN, built-in functions (`ABS`, `ATN`, `CLG`, `COS`, `EXP`, `INT`, `LOG`, `RND`, `SGN`, `SIN`, `SQR`, `TAN`), user-defined functions `FNA` to `FNZ`, `IF` statements permitting `AND` & `OR`, and `FOR` & `NEXT` statements and `GOSUB` & `RETURN` statements with no arbitrary depth limitations.

All of this, and meaningful error messages, occupies only about 3000 words of store, plus 120 locations of static workspace, 754 locations for simple variables and array & function descriptors, and a small amount for a general-purpose stack. Over half of an 8192-word store is available for the user's program and for array variables.

Being an interpreter, it isn't very fast. But it does avoid the endless rewinding of paper tape editors, translators, and executables, associated with a class full of school children all trying to develop even the smallest programs in SIR, ALGOL or FORTRAN. See "900 BASIC" [19].

Simulators.

A simulator was written in around 1963, to run on an Elliott 803B, to simulate a 920A with just 2k of store, so as to facilitate the early development of 920A software. See [20].

A simulator was written (experimentally) in the early 1980s, to run on machines which came later than the 900-series, in order to be able to continue running 900-series 18-bit

software with no access to a machine. Other benefits are that it can perform multi-pass compilations without user intervention, and it can access disk files in lieu of paper tape.

It is written in Ada83, and ran initially on a DEC VAX, but it was subsequently easily ported to run on any IBM PC running DOS. It is available (in executable form) on the Internet, complete with the tape image of the 900 ALGOL load-and-go system (described above). So if you want to run ALGOL 60 programs on your PC, see [21].

More recently, in 2002, a very thorough simulation of the 900-series 18-bit machines was written by Hans Pufal at ACONIT. It runs under Windows 95 or later, complete with a graphical representation of a 903 control panel. *I'm not sure if this is available in the public domain yet.* See [22].

Survivors.

An Elliott 920B was donated to the London Science Museum in 1987, by Paul Rayner of GEC, as reported in [23]. *I'm not sure what other 900-series 18-bit machines the Science Museum has, but I believe they have the 905 previously owned by Strathclyde University.*

CCS Member Don Hunter has owned a 16k Elliott 903 for many years. I'm unsure of its current working status.

CCS Member Terry Froggatt has had a 16k ARCH 9000 (equivalent to a 920B/903) since 1978, which he interfaced to a Pentium in 2001, although it is now not quite working. He also has had a 903 since 1981 for spares, which has never quite worked. Given a little spare time he's sure he could get one of his machines fully working again.

12/13-bit machines.

I know nothing about the 13-bit ARCH 102, except that it might have been coded in a notation the same as the 18-bit T2 (described above) but using 5-track tape: see [24]. The rest of this section relates to the later 12-bit machines.

Simulators.

A simulator was completed in 1968, to run on 900-series 18-bit machines, to simulate the 12-bit machines, so as to facilitate the early development of the 12-bit software. See "900 Series 18-bit Computer Aids to 12-bit Computer Users" [25].

I am not aware of any simulators that might have been written, to run on machines that came later than the 900-series, in order to be able to continue running 900-series 12-bit software with no access to a machine. (But no doubt the 12-bit software could be run in the 12-bit-in-18-bit simulator, running inside either of the 18-bit-in-PC simulators).

Symbolic Assembly Program (SAP).

SAP is the native assembler on the 12-bit machines, and is the equivalent of 2-pass SIR on the 18-bit machines, with added facilities to cope with addressing data on 128-word pages and to cope with calling subroutines on a 12-bit machine within a 15-bit address space. See “MC2/144 902 Programming Manual” [26] part 2 section 2.

A cross-assembler was also completed in 1969, called EHB, running on 900-series 18-bit machines, generating code for the 12-bit machines, so as to facilitate the early development of the 12-bit software. See “900 Series 18-bit Computer Aids to 12-bit Computer Users” [25].

FORTRAN Compiler.

“902 FORTRAN” was issued by Marconi-Elliott Computer Systems Ltd., *although I think that it was written for them by a software house.* This is a native compiler, requiring at least an 8192-word store both for compilation and execution. See “MC2/144 902 Programming Manual” [26] part 2 section 3.

CORAL Compiler.

“12/12 CORAL” was written by System Designers limited, using their Portable CORAL Compiler technology. This was a cross-compiler, hosted on a GEC 4080, with no native equivalent. See “Systems Designers Limited CORAL 66 Reference Manual” [27].

Survivors.

I am not aware of any surviving 900-series 12-bit or 13-bit machines.

Terry Froggatt, April 2004.