FINAL

Memories of Atlas Fortran

Ian Pyle

(formerly of A.E.R.E. Harwell)

York, 27 November 2012

Abstract

The Atlas Fortran compiler and its associated programming system were intended to ease the transition in 1964 of Harwell's computing work from the then current computers to the new Atlas. To do this, we had to have the compiler available when the Atlas arrived. So we made a cross-compiler on the IBM 7090, which produced cards that could be loaded onto the Atlas, needing only the loader on the Atlas itself. By writing the compiler in Fortran, it could translate itself. I think this was the first Fortran compiler written for an non-IBM computer, and the first use of this bootstrapping technique for a compiler.

Context

Jack Howlett was the head of the computing group at Harwell when I started there in 1958. We had a Ferranti Mercury computer, but sent a lot of work to be done on the IBM 704 computer at Aldermaston. Almost all the programs were written in machine code, although an increasing number were being written in Mercury Autocode.

Howlett [ref. 1] wrote for the Silver Anniversary of Atlas about the work that had been done to get a British computer for Harwell. The Manchester design for MUSE was chosen, to be built by Ferranti, and called Atlas [see ref. 2. Also, if your read Welsh, ref. 3!]. We at Harwell prepared for the transition from about 1960. Clearly programs in machine code would have to be re-programmed – but what to?

We knew about Fortran from IBM, and about Algol, as well as machine-specific autocodes. We decided that our strategy would be to start using Fortran on the IBM computers [see ref. 4], and to make a compiler for it on Atlas. We knew that would be possible, but at that time no-one had made an Algol compiler.

In the early 1960s, it was not well-known that a program in Fortran was written as a set of sub-programs, each compiled separately, then all combined by the loader at run-time. This “divide and conquer” strategy brought two important benefits. One was that the size of a program was not limited by constraints in the compiler – you could always split a larger sub-program into smaller pieces. The second was that, when a program was changed (is it was during debugging), only the changed sub-program(s) had to be recompiled – thus avoiding the wasted unnecessary re-compilation of the unchanged parts. Both of these were important for Harwell.

Writing for the Silver Anniversary of the (Chilton) Atlas, I reported [ref.5] that the four Atlas/Titan installation representatives had met in 1962 to discuss their plans for Automatic Programming Languages. There was Ferranti providing Mercury Autocode and Nebula, Cambridge preparing CPL1 (subsequently planned to be CPL2 but actually becoming BCPL, from which C developed), London doing the CHLF3 version of Mercury Autocode and Lunacode, Manchester doing Atlas

Autocode - and NIRNS/Harwell doing FORTRAN. The present paper is about the Fortran compiler for the NIRNS/Harwell (i.e. Chilton) Atlas. Subsequently, Fortran compilers were written independently by Morris (Manchester) and Hendry (London), when the language became more popular.

We also knew of the work being done at Manchester by Brooker on the Compiler Compiler (CC), but considered it inappropriate for Fortran, for two fundamental reasons. The first reason was that Fortran (FORTRAN II) was at that time not defined formally, but by the manual and the compiler. Thus we did not have the primary definition of the language to put into the CC. The second reason was that the CC required the program text to be presented completely (including all the subroutines), with no provision for sub-programs to be compiled separately, as required for Fortran. This was why we developed BAS.

Fortran

We knew that Fortran was slightly machine-specific (it has particular instructions for dealing with the switches and lights on the console of the IBM704), but decided that a version for Atlas would be easy for Harwell programmers to adopt, after they had gained experience on the IBM computers at Aldermaston. The several versions of Fortran had slight variations, which I called "Dialects of Fortran" [ref. 6]. Atlas Fortran was in this spirit. The extensions are outlined in the implementation paper [ref.7].

We had no previous experience of writing compilers, but did have access to IBM's documentation about the Fortran II compiler for the 704, including the assembly language listing (about 7 or 8 inches high of line-printer listing paper), and the description by Sheridan [ref. 8] of the expression analyser. But for writing in Fortran, we were on our own.

Harwell's legal team wrote to IBM to ask whether IBM considered the name "Fortran" to be proprietary, in which case we would have had to invent a different name for the language. Fortunately for us, IBM raised no objection to our using the same name.

We knew that it would need a symbolic loader, to link at run-time the several sub-programs that were compiled separately, analogous to the BSS (Binary with Symbolic Subroutines) loader for the 704. Alan Curtis and I designed the BAS (Binary and Arbitrarily Symbolic) loader for the Atlas, making a slight extension to BSS, particularly to deal with Block Common in Fortran IV.

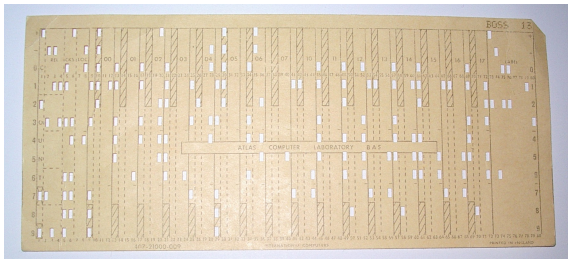
We also recognised that Fortran was intended for working with numbers, not characters. The basic technique for dealing with characters was explained in reference 9, using input/output in an unconventional way. As a demonstration of feasibility, I wrote an assembly program for Atlas, called ASP (inspired by the SAP assembler for the 704). This produced a listing in octal, and cards in BAS, from a text in a mnemonic order code for Atlas. There was a manual for it, but I can find no record of it.

BAS

Although our motivation was based on using Fortran, we recognised that other programming languages were in the air, and compilers being written for the Atlas. There was little Fortran-specific in the design of BAS (only the allocation for COMMON and Block Common variables in distinct regions of the main store), and we hoped that other compiler-writers would recognise the value of separate compilation of sub-programs, and use BAS as the target language. The paper was called "A proposed target language for compilers on Atlas" [ref.10], but I do not think anyone

else used it – this may have been because of the storage allocation (see below). Neither was much use made of the "parameter" facility which it introduced, intended to give more generality to storage allocation for large arrays.

BAS was intended to be used with punched cards. We designed a binary card format specifically for BAS. A typical card is illustrated below.



(From http://www.chilton-computing.org.uk/acl/technology/atlas/p013.htm)

BAS did not make the distinction between static and dynamic areas of storage: there was no way to designate a particular area as "read-only." Only much later did we realise the importance of separating fixed and variable areas of memory for implementing recursion and re-entrance. (The Atlas Supervisor had this distinction enforced by the hardware design: read-only fixed store, read-write working store.)

Hartran

The Atlas Supervisor was the central innovation of the Atlas concept (see Appendix). It managed all the time and space in the computer, and provided for the execution of a series of "Jobs", under the control of what the supervisor recognised as a compiler. This was not appropriate for Fortran, as there could be a number of sub-programs to be linked together. On the IBM computers this was done by the Fortran Monitor System (FMS). For Atlas, we introduced the concept of a "Local Operating System", or Programming System, to control the processing of programs for individual jobs. We called it Hartran, so a job with sub-programs in Fortran was introduced by the directive "COMPILER HARTRAN" - which led to considerable confusion. See Fossey & Stokoe [ref. 11].

In the HARTRAN system, a program can consist of a number of routines, each of which may be written in any source language for which a compiler exists, producing its output in BAS.

Compiler issues

An overview of the Atlas Fortran compiler was given in my implementation paper [ref. 7]. It was written in Fortran, as a cross-compiler producing code for the Atlas, but running (and being debugged) on the IBM 7090. When the cross-compiler was working, it was fed into itself, to produce the Fortran compiler to run on the Atlas. We described the process as "bootstrapping", analogous to the technique used for loading binary programs include IBM computers. The diagram below (copied from ref. 7) shows the bootstrap process, with the boxes showing the executing machine or language, and the translation carried out. Just follow the numbers!

Following the diagram are a few further points, which I remember rather hazily! There is a little bit of repetition from my earlier papers.

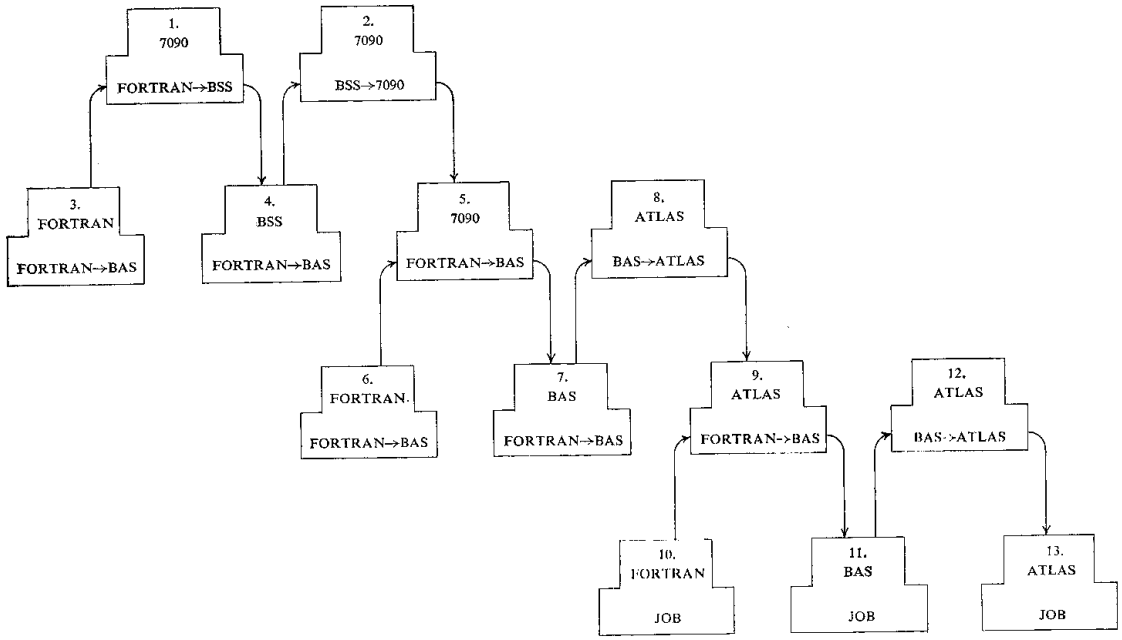


Fig. 1

Compiler subroutines

Working on the Fortran compiler, we created hundreds of subroutines. To avoid conflict about names we had a simple convention: whoever invented a subroutine gave the first letter of their surname as the first letter of the subroutine name. Thus all my subroutines began with “P”, all Barbara Stokoe's began with “S”, Bart Fossey's with “F”, etc. If the text of the compiler survives anywhere, this will allow the authors to be identified!

Character handling

The format processing facilities of Fortran are very powerful, but not normally available for internal use, as they are tied to input/output operations. The technique [ref.9] of using the input/output buffer without an actual input/output operation allowed us to read a card, than process the characters on it as though they were integers. We used this on the IBM 7090, and (presumably) made similar subroutines on the Atlas.

Statement recognition

Fortran statements are distinguished by rather complicated rules: there are no reserved words, and variable names can be the same as those that distinguish statements. The trick case is the statement

```
DO 5 I = 1.3
```

which is an assignment of the value 1.3 to the variable called DO5I! However, if the full stop is changed to a comma, it becomes a statement introducing a DO loop.

So we determine the type of each statement by a series of scans; first to look for parentheses and the equals sign, excluding comments cards and text strings (Hollerith). After excluding arithmetic statements, we distinguished further by testing the first letters of the statement. (See [ref. 7, section 12)

B-lines

The Atlas was very generous in the provision of index registers, called B-lines. However, they had to be protected from interference between one subroutine and another. It was not feasible to allocate them *relatively* (like addresses for main memory: relocatable binary), so each subprogram used as many B-lines as it needed, but had to protect them carefully. A basic design decision was whether to save and restore index registers *within* a subroutine that called another, or to rely on the called routine to protect them. To resolve this issue, I collected about 30 subroutines from typical uses, and analysed them to decide where the balance lay. It turned out that most used only a few index registers, and the best solution was to save as many as would be needed on entry to the subroutine, and restore their original values just before exit. I do not know whether anyone carried out measurements to see whether this decision was justified in practice.

I still do not know how extensively the B-lines from (say) 16 upwards (below 90) were used – I suspect very little if at all.

Symbol table

Names in Fortran could be up to six characters long, starting with a letter. (This corresponded to six 6-bit characters in a 36-bit word on the 704/7090). They did not have to be declared before use. So the compiler has to build a symbol table to contain the details of each name in the subprogram. In order to be able to access the table quickly, we arranged for it to be sorted to spread out likely names, accessed by an encoding of the name itself. This technique was subsequently called “hashing”. I do not remember it being written up anywhere.

Bob Hopgood has pointed out that the technique had been used earlier (in the IBM701 assembler); we had not known about it at the time.

Main store allocation

The BAS loader made full use of the main (one-level) store of Atlas, treating major regions of address space for different purposes, according to the most significant octal digit of the address. Addresses starting with 0 were the “global region” (for ordinary sub-programs with their local variables), those starting 1 were the “chapter region” (for sub-programs in separate chapters, corresponding to the use of CHAIN in Fortran); those starting 2 were the “block region” (for Block common variables) and those starting 3 were the “common region” (for COMMON variables). Of course, addresses with most significant octal digit 4 and above were reserved for special uses by the supervisor.

The strategy of Fortran was to associate local variables with their particular sub-program, without distinguishing between static and dynamic areas of memory. It did not recognise “block” structure as in Algol60 and later languages.

Measurements (absence of)

In retrospect, I regret not having arranged to measure anything about the compiler, so I have no information about the number of subroutines, frequency (or paucity) of errors found during debugging, throughput or execution time. Perhaps someone else did that?

Consequences

In retrospect, I think there are two important consequences from our work on Atlas Fortran. The success of separate compilation of sub-programs was emphasised in the Flowers report [ref.12] which called for the following features of Compilers:

> 1. Fast Fortran and Algol compilers capable of producing good object code in relocatable binary.
> 2. An assembly language with mnemonic and macro instructions.
> 3. Facilities by which a program having any combination of subroutines written in Fortran, Algol, Assembly language or relocatable binary can be executed.
> 4.If other compilers are available they also should produce the same relocatable binary.
> 5. The system should be card oriented because for most purposes cards are more convenient, but paper tape facilities should be included since experimental data is frequently produced in this form."

In other words, the Fortran model of a program was implied, disregarding the fact that Algol60 did not have the concept of separate compilation of sub-programs. This was not specific to Atlas Fortran, but was made very evident because of it.

The other important outcome, not directly from Atlas, but from Harwell's decision to program in Fortran, was the Harwell Subroutine Library. This was started by Mike Powell and Mike Hopper (and others at Harwell in 1961 and 1962 - thanks to Bob Hopgood for this!), and made public in 1963, now called HSL and developed at the Rutherford Appleton Laboratory.

Remembering Alan Curtis

I think it fitting to conclude with a tribute to Alan Curtis, who died in 2008. Alan was a wise and inspirational group leader, whose insights and guidance were influential on all who worked with him. Primarily a mathematician and numerical analyst, he significantly influenced our thinking on system software, computer languages and computer-based systems. Alan had been involved with Atlas from the design of the order codes, and was the strategic thinker behind the work reported here. Under his guidance, I had the most productive years of my life.

Appendix: About the Atlas Supervisor

Before the Atlas became available, a computer user had access to an effectively empty machine. There was only a very basic loader, which put in one's program into the store, to do whatever one wanted – or more often not, because of mistakes in the program. The program was strictly sequential, so that any input-output operation took as long as the hardware needed (e.g. to get the next row from paper tape or punched card, or up to several seconds for a line printer's page throw). When a program came to its end (or was stopped), the human operator had to reset the machine, and start loading the next job's program.

Various improvements came with experience: a timer could stop a program if it exceeded its allotted time (often because it got into an infinite loop); hardware devices were given more autonomy (but then had to report completion of an operation to the program); faster input-output devices such as magnetic tape were introduced as intermediaries between tape/cards and printer; and a standard program was arranged to execute between jobs for administration (identifying each user's results on the output printer, and recording timing, use of resources etc., for charging).

The Atlas supervisor [ref. 13] was a radical solution to this situation, integrating the treatment of peripheral equipment, input/output handling, the one-level main store, and the backing store on magnetic tape. It also controlled the sequence of users' jobs for execution, connecting each to its own input and output channels.

The Atlas supervisor had broken important new ground, and should be recognised as the first “operating system” as the phrase is now understood, covering the management of time, space, and peripherals of a computer in an organised way. It had to provide multi-programming facilities for handling the peripherals (using interrupts), which later operating systems also applied to user programs, giving multi-access systems.

This was possible because the program for the supervisor was permanently resident in the store (i.e. the fixed store, addresses 4000,000.0 (octal) upwards), with its own reserved work-space in subsidiary core store (addresses 5000,000.0 upwards), accessing the control and status registers for all the peripherals in the V-store (addresses 6000,000.0 upwards). Not only was this ground-breaking, but the programming support was excruciating - debugging involved physically manipulating the pieces on material in the fixed store.

The concept of multi-tasking was much in the air at the time, and was (much later) described by Dijkstra in his famous paper [ref.14], in which he introduced his elegant solution to the problem of *communicating* sequential processes.

The Atlas supervisor was well ahead of IBM's Operating System OS/360, and of GE's Multics, which preceded Bell Lab's Unix. Now, even a Raspberry Pi has to have an operating system: it cannot be made to work without one, as there is no loader!

In summary, the Supervisor was the first to have the characteristics of a modern Operating System: time management for multi-tasking of sequential processes (among peripherals handlers, using interrupts), store management (giving the one-level Virtual Memory) and I/O management for all its online devices. The Atlas Supervisor (although it had only sequential job-scheduling) was a significant achievement.

References

1. Howlett, J.: "Atlas Laboratory: Origins," Flagship issue 5, Sept 1989 (Atlas Silver Anniversary Issue): *in* <http://www.chilton-computing.org.uk/ccd/literature/ccd_newsletters/flagship/p005.htm#s5 >

2. Atlas Lab 1999(?); "Atlas Computer, Overview" *at* http://www.chilton-computing.org.uk/acl/technology/atlas/overview.htm

3. Pyle, I C: "Y Cyfrifydd Atlas", Y Gwyddonydd, vol **1**(3) pp 120-125 (1963) *at* <https://journals.library.wales/view/1394134/1394449/15# >..(In Welsh.) Note that when this paper was written, the word for "computer" was "cyfrifydd" (also meaning "accountant"); it is now "cyfrifiadur".

4. Curtis, A R: "Preparation for a Possible ATLAS Computer at AERE" (1960) in <http://www.chilton-computing.org.uk/acl/literature/earlyhistory/p003.htm>

5. Pyle, I C: "Atlas 1 25 years on" Flagship issue 5, Sept 1989 (Atlas Silver Anniversary Issue): *in* <http://www.chilton-computing.org.uk/ccd/literature/ccd_newsletters/flagship/p005.htm#s5 >

6. Pyle, I C: "Dialects of Fortran", CACM vol **6**(8) pp 462-467 (1963)

7. Pyle, I C: "Implementation of Fortran on Atlas", *in* "Introduction to System Programming", *ed*. P. Wegner, Chapter 6, pp86-100, Academic Press (1964)

8. Sheridan, P: "The Arithmetic Translator-Compiler for the IBM Fortran Automatic Coding System", Comm A.C.M. vol 2(2) (1959)

9. Pyle, I C: "Character manipulation in Fortran" CACM vol **5**(8) (1962)

10. Curtis, A R & Pyle I C: "A proposed target language for compilers on Atlas", Computer Journal vol 5(2) pp 100-106 (1962).

11. Fossey E B & Stokoe B: "Fortran on Atlas: The Hartran System" Atlas Computer Laboratory (1966), *in* <http://www.chilton-computing.org.uk/acl/applications/hartran/overview.htm>

12. Flowers, B H: "A report of a Joint Working Group on Computers for Research" (The Flowers Report), Cmnd. 2883, HMSO (January 1966), *at* https://www.chilton-computing.org.uk/acl/literature/othermanuals/flowers/

13. Kilburn, T., Payne, R. B., & Howarth, D. J. "The Atlas Supervisor", Proc. E.J.C.C., December 1961; *also (1962 version) in* http://www.chilton-computing.org.uk/acl/technology/atlas/p019.htm

14. Dijkstra, E W: "The Structure of the "THE"-Multiprogramming System", Presented at an ACM Symposium on Operating System Principles, Gatlinburg, Tennessee, October 1-4, 1967, *published in* Communications of the ACM, **11**(50), pp 341-346 (May, 1968)