ATLAS AUTOCODE COMPILERS FOR LARGE 1900 COMPUTERS

by

J. A. LINN

A Thesis

submitted to the University of Manchester

for the degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science January 1972.

ABSTRACT

This thesis describes the implementation of Atlas Autocode on large I.C.L. 1900 computers. The basic design and compiling algorithms are described demonstrating the use of the 1900 order set in producing efficient object code. The store layout, I/O and operation of the compiler are discussed with reference to the I.C.L. normal executive and George 3 on a 1905F and MX2 on a paged 1905E. The contrast between the different systems and computers make significant differences to the compiler where it interfaces with the system but basically the same compiler is used for all systems. The in-core compiler was implemented on the 1905F to run in a batch mode resulting in a system similar to Watfor for Fortran. The compiler has two modes of operation and is run on varying systems and therefore an attempt to assess the efficiency or otherwise of the compiler is considered. The complexity of the problem of the measuring software efficiency due to for example multi-programming systems, is discussed and initial results are given for the Atlas Autocode system.

One of the current drifts in computing is towards on-line programming and therefore the extension of the Atlas Autocode system was intrinsic to the basic design. The languages already used on-line give an introduction to the facilities required by any on-line system and methods of achieving them are discussed for Atlas Autocode. The basis of the on-line facilities is the AA statement

but it is necessary to add some extra statements to provide the user with more apposite ways of communicating or to aid compilation of particular statements. Incremental compilation has already been used to provide on-line compilers but usually the object program efficiency is low as compared to off-line compiling. The Atlas Autocode compiler is based on the incremental technique but uses a form of line table in an attempt to gain object code efficiency. The compilation difficulties encountered in Atlas Autocode are briefly outlined to indicate the use of the major compiler lists in on-line programming.

The on-line system is to some extent an initial proposal for on-line conversational compiling for Atlas Autocode and indicates the direction of future research.

PREFACE

The author graduated in 1967 with a B.Sc. degree (Mathematical Science), from the University of Edinburgh. Since then he has been at the University of Manchester where, in 1968, the degree of M.Sc. was conferred.

ACKNOWLEDGEMENTS

The author wishes to acknowledge the supervision of Dr. J. S. Rohl throughout the duration of this project. Also R. P. Yeardye who wrote the perm routines for the compiler, and the staff of the University of Manchester Regional Computer Centre for help with George 3.

The author wishes to thank the Science Research Council for financial support, and Professor T. Kilburn for providing the research and computational facilities.

INTRODUCTION

This thesis describes the design of Atlas Autocode compilers for large I.C.L. 1900 computers. By a large computer it is meant one with at least 64K of store available for use. The work was carried out on the I.C.L. 1905E (Department of Computer Science) with 32K store which is paged to give a 64x64K segmented virtual store, and the I.C.L. 1905F (University of Manchester Regional Computer Centre, UMRCC) with 128K of store. The basis of the design of the compiler was such as to make extensions possible and in particular to provide the basis for an on-line conversational compiler for Atlas Autocode. Nevertheless an efficient off-line system was required and this was done by the optimisation of the object code and by the design of the system to make best use of the operating systems.

The reasons why Atlas Autocode was used as the language for the research compilers were as follows:-

(i) An Atlas Autocode compiler was required for the 1905F (UMRCC) with the possible conversion for use on a 1906A.

(ii) The research executive MX2 and the supervisors on the 1905E required a compiler in order to make system tests etc..

(iii) It was considered that Atlas Autocode was a suitable language to investigate on-line conversational compiling techniques due to its statement by statement definition.

The background of this project is briefly outlined in the first chapter where the computers used, implementation language, operating systems and editing facilities are described in brief. The second chapter describes the basic design of the compiler considering amongst other topics the optimisation of the object code for expressions, assignments, routine over-heads, and cycle loops and the forms of syntax structures used in the compiler. Although the compiler was designed with on-line compilation in mind, the compilers implemented were primarily for off-line use. In Chapter three the compilers are described with reference to the operating systems I.C.L. normal executive, George 3 and MX2. The varying input/output systems, store layout and operating instructions are discussed. The fourth chapter considers the measurement of the efficiency of software and compares the two modes of compilation on the 1905F, batch or single, and makes a comparison with the MX2 system.

Turning from off-line to on-line chapter five gives a survey of the current conversational languages with the objective of picking out those facilities which are required in a good on-line system. Atlas Autocode is considered in the light of these other languages and a proposal is described for on-line conversational Atlas Autocode. This includes program layout, editing facilities, debugging aids and "adding-machine" facilities. The implementation of on-line compilers is discussed in Chapter six with reference to other work in the field. Incremental compilation is considered and a form of this technique is

described for Atlas Autocode. The main aim is to improve the execution efficiency and the proposal is described with reference to editing and the "adding-machine" facilities.

The basic points of research interest are drawn together to form the conclusions which end the thesis.

History of Project

Before considering the main subject matter of the thesis it is interesting to study the progress of this project since it is possible to draw on the experience gained. The project covered a period of two years and involved three people, R. P. Yeardye who wrote the perm routines and was involved for one year, Dr. J. S. Rohl who supervised the project and the author who worked for the whole two years on it. The two compilers took about three and a half man-years to write and commission. It will be observed from the diagram (Figure 1) that the final time schedule differed greatly from the initial one both in that times were increased and in the order in which the compilers were developed.

In the first year much of the time was initially spent in understanding the 1900 computer, S.P.G. and designing the compiling algorithms for Atlas Autocode. Once the 1905E computer was available with SPG a small compiler was written in order that some progress might be

made on the perm routines. Due to the size of the basic 1905E (32K) this compiler provided very limited facilities. This forced the perm routines to be written in simple AA and since the design of the perm followed the Atlas perm, the routines were written in AA 1900 machine orders.

It was also possible to develop several other parts of the compiler but the size of the total compiler was restricted. Attempts were made to improve the size by adding special dumping facilities to SPG and compiling sections of the compiler to disc but although this increased the size of program SPG compiled, the 24K available was no substitute for the 40K for which the compiler was designed.

From this it is concluded that compilers can only really be developed on machines giving at least the facilities of compiling the complete compiler and running it or else much time is wasted fitting the compiler into the available space and developing systems to do this.

As already mentioned, the perm routines were written in machine orders and this proved a major design mistake since when the author of the perm left, the perm was not fully debugged. This was due to the fact that it had never been compiled fully with the full compiler. A better approach would have been to design the complete perm on paper, i.e. using flow-diagrams and written documentation, and writing it in full Atlas Autocode. This could then have been debugged on the Atlas computer, except where the perm routines interface with the particular machine. Although the object code may have been slightly slower the commissioning time would have been reduced and

where efficiency had really suffered it would have been possible at a later date to machine code the relevant sections.

Once the 1905E became completely unusable there was a period when no computer was available and during this time the compiler was organised into its final form. When the 1905F became available, SPG was altered to run in extended store mode. The computer could only be used after the commissioning engineers had completed their work for the day and then only if the computer was still in working condition. Therefore over the next period some progress was made with the compiler but once the computer was commissioned it again became unusable. Delays in the building held back the installation at UMRCC and difficulties with air-conditioning reduced the time available to almost zero. This continued until the computer was recommissioned, but again the work could only be done at night. This proved difficult during term due to programming laboratory commitments. Also during this period there was insufficient editing facilities, only card punches being available. It should be noted in passing that the associative store for the 1905E was not commissioned till about Christmas, several months late. Even then several faults occurred and it was not completely reliable.

Night work was generally done alone and this proved, on reflection, not to be the best way of working. It is thought that more than one person could have completed the work more efficiently both in man time and computer time.

Once commissioning was completed on the 1905F, George 3 was set up and the system changed over to run under this operating system. Several difficulties were experienced both with George 3 and SPG during the change-over. At that time compiling under normal executive took about 30 minutes and this was increased to 40 minutes under George 3. This was due to the files requiring retrieval from magnetic tape. Once the computer was offering a service the time in the computer averaged about 150 minutes for 7½ minutes mill time. The maximum time noted was 5½ hours in the machine before re-emerging when the operators were closing down to go home. Development of the compiler under George therefore slowed down the progress due to the slow turn-round.

The compiler did not prove to be difficult to commission but several faults in the perm took a great deal of time to find and correct. The compiler has now been in service for several months and a few compiler and perm faults have been found by users. The batch system has not been used very much but during the next year it is going to be used for student work.

The change-over to the 1905E under MX2 was more easily done. This is probably due to the fact that others had defined compilers under this system and the system difficulties had been solved. The AA system did uncover some extracode faults which were soon corrected. The compiler on the 1905E is at the same standard as the one on the 1905F and is being used to derive statistics about the MX2 system.

To sum up, it may be concluded:-

1. Initial time schedules are very approximate, especially when they depend on the availability of computers which are in the process of commissioning or development.

2. To change from one system to another involves a distinct time overhead.

3. When the correct computer is not available the compiler system is better designed completely on paper both in general and in detail, since when it is coded the commissioning process has a natural progression which is not interrupted by system changes.

4. The perm routines should not be written in machine orders except where necessary, e.g. interfacing with the computer or where efficiency would suffer greatly.

5. Development of the compiler requiring night work should be done in groups or together with other research groups or persons.

CHAPTER 1

SURVEY OF SYSTEMS

During the period of this project several distinct computers and systems were used. This chapter gives a brief description of the computers, languages and operating systems to which reference is made throughout the rest of the thesis. The descriptions are not meant to be complete, but are concerned to give details of the systems etc. which have a bearing on the design, development or efficiency of the compiler. Thus it will not be possible to understand for example SPG, from the description given, and further information will be found in the references. Since each system was in the process of change throughout this project, some discrepancies may appear, but whenever possible the position in development is given.

Atlas Autocode

Atlas Autocode, often referred to as AA, was implemented at Manchester University on the Atlas computer in 1963. To some extent it is a variant of Algol 60, removing from it some of the complexity which leads to compiling problems. The language is designed for statement by statement compilation and therefore a one-pass compiler can be used, giving efficient compiling speed. Execution is more efficient than Algol since the language does not include such general features. On Atlas two compilers were implemented, firstly the AA compiler written

using the Compiler Compiler (Ref. 2) and secondly the AB compiler which used ad-hoc methods of compilation. The language was designed so that extensions to the facilities could be added. The AB compiler was extended to include multi-length arithmetic under the compiler name ABC (Ref. 3) and the AA compiler includes some compiler-compiler facilities (Ref. 4) and list processing (Ref. 5).

In the light of extensive use of these systems on the Atlas computer, the language was re-defined more formally in "A Definition of Atlas Autocode" J.S. Rohl (Ref. 1) and this was used to define the language implemented in the AA compiler now being described in this thesis. During the later stages of commissioning of the compiler, work was started to extend the facilities of the compiler to include simulation (Ref. 6). In addition to the language itself there was found a need to define a program description to give details of, for example, input and output. Some statements previously in the program could be more readily put in the program description, e.g. upper case delimiters.

Another new concept to AA was that the program was typed in a particular convention. These conventions were such that more media could be used and included.

(i) normal delimiters

(ii) upper case delimiters

(iii) single case

The first two have been used with the Atlas system (Ref. 7) but may now be specified in the program description thus avoiding in the second case use of normal delimiters

initially. The program description was defined in single case. The single case mode uses primes to denote delimiters, e.g.

'BEGIN' or 'begin'

since all letters were taken as lower case. In order to avoid confusion with symbols, single letter delimiters, e.g. e were typed double e.g. 'EE'. Also included were a set of delimiters for the comparators in conditions,

'EQ', 'NE', 'LT', 'GT', 'LE', 'GE'

This convention proved useful for cards, some tape punches and the consol typewriter.

A program text was therefore considered as a document written according to some convention which was specified in the program description. The convention could be changed in the program as before.

The statements of the language were essentially the same as those used in the Atlas compilers and programs usually required little change to run on the 1900. Excluded from the language were those facilities which were not well used or were expensive in computing time, e.g. test and case and arrays called by value. Included in the language for the first time were:-

(i) blisters

This is an extension to query printing and now a whole line may be compiled or not according to whether it is required. For example, when debugging,

```
a(i) = f(x,y,z)
 < caption a(;print(i,2,0); caption)
 <print fl(a(i),5)     ; comment this is a blister
```

(ii) Else after conditions

A common sequence of instructions in an AA program was

```
        if a = b then →4
        d = e
        →5
4:      d = f
5:
```

causing more labels than necessary to be used. The else statement was therefore introduced and the above example is now written,

```
if a = b then d = f else d = e
```

This is defined syntactically as (using SPG)

```
<CONDITION>   = <IF.OR.UNLESS> <COND> then
                    <UI> [else <ELSE> | <NIL>]

<ELSE>  =  <UI> | <CONDITION>
```

Thus the else may be followed by another condition but there is no problem in matching what follows the else with the condition.

(iii) Groups

This is essentially the Algol 60 compound statement but to avoid confusion with blocks a group of statements is surrounded by the delimiters start and finish. These may be used with conditions and there a group is effectively considered as a single entity, hence the else may follow the finish. Examples of this are

```
if a = b then  start        if a ≠ b then →4
               c = d        c = d
               e = f        e = f
               finish  4:
```

and

```
if a = b then start          if a ≠ b then ->4
      c = d                        c = d
      e = f                        e = f
finish else start                  ->5
      c = f                  4:    c = f
      e = d                        e = d
finish                       5:
```

Syntactically start is include in the set of UI statements, and finish is defined as

finish [else <ELSE> ! <NIL>]

SPG

The System Program Generator (Ref. 11) was designed to provide a system software language. It was developed on Atlas (Ref. 12), and was transferred to the 1900 retaining much of the terminology drawn from Atlas, e.g. B-lines. The SPG compiler is written in SPG and is split into two parts, the basic compiler and the bootstrap. The former provides the essential back-bone of the language and the latter creates some useful facilities by the use of macros. These include input and output statements and routines. The user can define his own macros and these are used in the compiler to define the interface with the machine orders whenever possible, e.g.

```
MACRO NEXT PARAMETER CODE NO TO B6  < NL>
      :: This picks up the next character from B3 and
      :: advances B3 by one character position.
*     024  6  0(3)
*     064  3  *+1
      -> EXIT
      END
```

The two machine orders are planted whenever the macro is used. The freedom to use macros was initially curtailed in the AA compiler due to the lack of store on the unpaged 1905E. The basic SPG compiler used for AA on the 1905F was converted from the original 1900 SPG compiler to run in extended store mode and had the * statement and routines added as bootstrap. The size of the constant list was extended and since the AA compiler used S.VARS frequently B2 was used as SB instead of B13.

On the 1905E the compiler ran in extended store and extended branch mode and compiled the AA compiler similarly.

The definition of the syntax dictionaries for the AA compiler was done in a similar manner to those of SPG itself using S.SYNES and S.COSYNES. For example,

```
S.SYNE <UI>  =  <VARIABLE>  =[ <EXPR> <VAL(ASSIGN)>  | £
                               <NIL> <VAL(RTN.CALL)> ]
```

and

```
S.COSYNE <N>
    IF  [SS]  -  (0)  < 0,->FALSE
    IF (9)  -   [SS]  < 0,->FALSE
    B4 =  [SS]  -  (0)
```

```
LOOP:  [SS] + 1
       IF  [SS]  -  (0)  <  0,-> END
       IF   (9)  -  [SS] <  0,-> END
       B4 + B4
       B6 = B4
       B4 + B4 + B4 + B6 + [SS]  -  (0)
          -> LOOP
END:   [SI]  =  B4
       SI + 1
       -> TRUE
       END
```

Thus with the aid of the standard SPG cosynes defined as S.COSYNES and synes defined as S.SYNES, the syntax of AA was described.

The I.C.L. 1900 Series Computer

The 1905E and the 1905F computers are part of a range of computers of the same basic design but differing in size and facilities. The basic 1900 computer has a 24-bit word and the instruction set provides all the operations for fixed and floating point arithmetic as well as character manipulation, logical operations and shifts with extracode interfacing the program with the computer.

There are eight central index registers, termed X-registers, of which three may be used for modification. With regard to Atlas Autocode compilation this was, with one exception, a sufficient number of modifiers with the help of the supplementary modifying instruction (117). The unmodified address gives access to the first 4K of store, lower data,

and normal modification increases this to 32K. Extended store mode increases the modifier to 22bits from 15bits but thereby restricts the use of some of the group 06 orders. Normal branches only access up to 32K and therefore extended branch mode is used which increases the range by use of replaced and relative branches.

The fixed point integers are held in one word and floating point numbers in two consecutive words. When overflow occurs in an arithmetic calculation the V register is set but the program is not normally interrupted. This register may be interrogated by the appropriate 074 order. There is a hardware facility for monitoring on overflow and this may be used by setting the corresponding monitor mode.

1905F (UMRCC)

This is a standard ICL 1900 computer with the F specification. This includes a .75 microsecond store (128K), high speed X-registers and hardware floating-point unit which provides for all the floating point orders (group 13) and the 076 orders. The computer has both basic and magnetic peripherals:-

1. 2 tape readers, 1000 chs/sec.
2. 2 card readers, 1600 cards/min.
3. 2 line printers, 1100 lines/min.
4. 2 tape punches
5. 1 card punch
6. 1 graph plotter
7. 4 9-track magnetic tape decks

8. 4 7-track magnetic tape decks
9. 2 magnetic drums
10. 1 fixed disc

Full details of the peripherals is given in "The Users' Handbook" UMRCC (Ref. 14). Also connected to the computer via a multiplexor is a communications link and two on-line typewriters.

It is used under normal executive but primarily under George 3 when the magnetic peripherals hold the file store. Incremental dumps are done on to the magnetic tapes for security.

This computer is used to provide a computer service for Manchester University and the universities in the surrounding region.

I.C.L. 1900 Executive

The standard 1900 executive provides all the expected functions of an executive and also a means of controlling programs from the operators consol. The executive can therefore be used as an operating system under the control of the operator who either types commands or gives them via the F-buttons on the consol type-writer. Peripherals are used in an on-line manner by means of the 157 extra-code, where the program is able to control the input or output from or to the peripheral. The executive is more fully described in "The Central Processors Manual" (Ref. 15, also Ref. 16) where some details of the operations are given.

The executive provides some facilities which can be used to aid compiler implementation:-

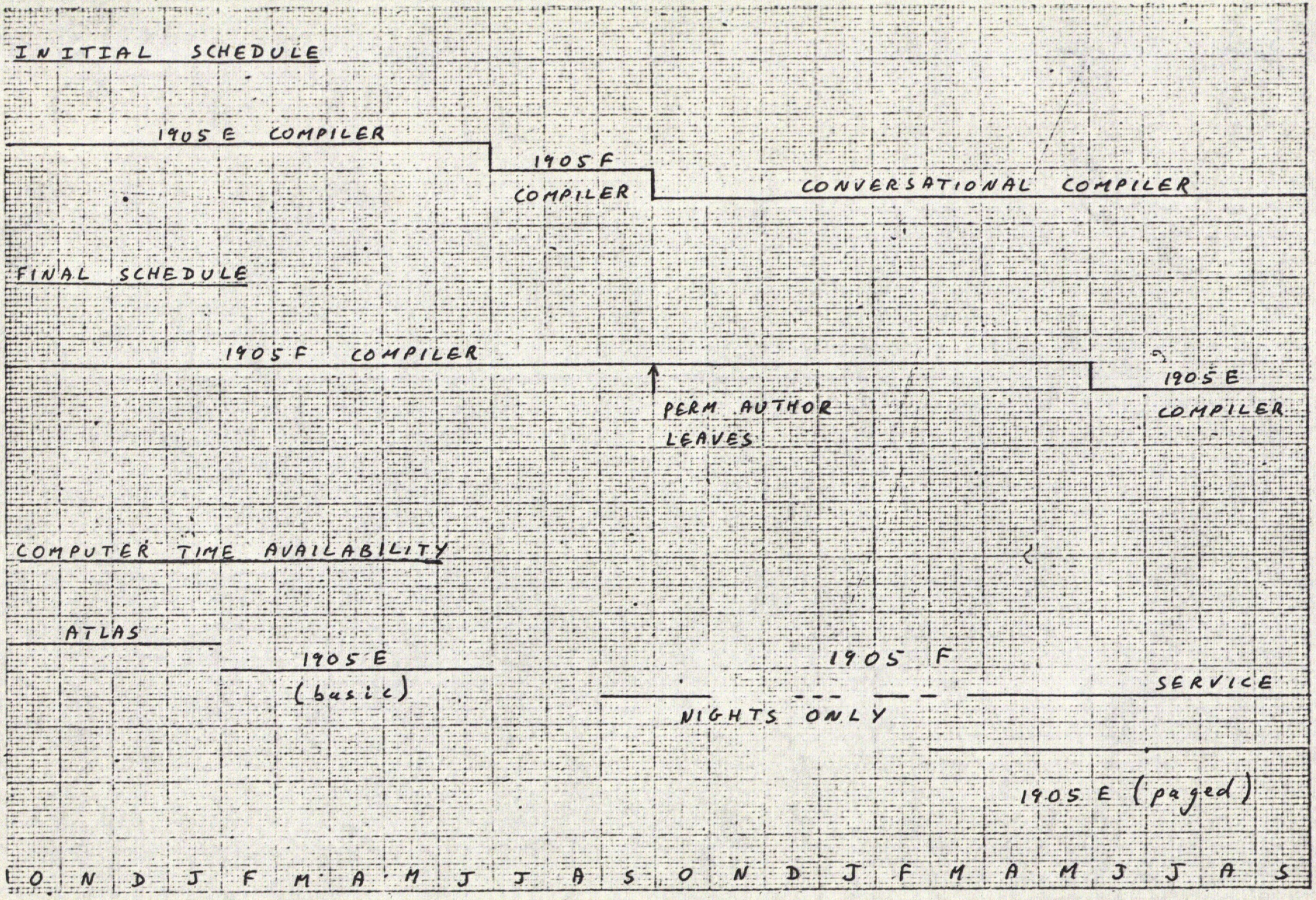

Figure 1. History of Project