EXPERIENCE WITH TEACHING ASSEMBLY LANGUAGE

D. Crookes

Department of Computer Science
The Queen's University of Belfast
Belfast BT7 1NN
N. Ireland

1. INTRODUCTION

Even in a world in which the use of high level programming languages has become almost universal, assembly language programming is far from obsolete. There are times when a programmer is perfectly justified in writing in low level assembly language (AL). For this reason, the Computer Science department at Queen's University, Belfast (QUB) aims to ensure that students are taught the basic skills of AL programming. This part of their training comes after a course on high level (HL) programming, using Pascal[1].

An examination of how these two aspects of programming (HL and AL) are often taught in textbooks and elsewhere reveals an alarming dichotomy in programming methodology. For HL programming, techniques such as structured programming and stepwise refinement are advocated[2, 3]. Flowcharts, if even mentioned, are certainly frowned upon. Yet those who aim to teach AL programming very frequently recommend (or at least use) low level design techniques, often based on flowcharting[4, 5].

This dichotomy in programming methodology has led us to develop a diffenent technique for teaching AL programming at QUB. The method is rather simple and obvious - which may perhaps explain why it is rarely taught as an explicit methodology. The key to the approach is to develop a program in two phases:-

(i) *design* the program, using a HL language-like notation;

(ii) *implement* the design, by following a set of standard translation rules (i.e., doing a sort of manual compilation).

The method has been described in detail in a separate paper[6], in which programs are designed in a Pascal-like notation, and translated into AL for a system based on the Intel 8080[7]. The purpose of the present paper is to concentrate on our experience of teaching AL programming using this method. The principles of the method can be covered in about two lectures, and are reinforced by assessed practical exercises, in which the students are asked to apply the technique to a variety of problems. Practicals have been organised to cater for up to 200 students per week.

The benefits of the approach turned out to be surprisingly numerous, both for those involved in teaching it, and for the students. Of course, the approach was not without its problems and teething troubles. In presenting our experiences we first consider some of the main advantages (for the student, and then for the teacher), and then some disadvantages.

2. ADVANTAGES FOR THE STUDENT

Before introducing the present approach to AL programming, students used flowcharts to design and document their programs. It might not be too far from the truth to say that, in some cases, the program was written first, and then the flowchart drawn! It was not uncommon for a student during a practical session to appeal for help with an incorrect AL program, which had been modified several times in an attempt to get it working, and which had neither clear design nor meaningful documentation. Anyone who has demonstrated on such a course will be all too well aware of the ability of students to produce extremely complex and baffling solutions to a relatively simple problem! Now that the present method has been introduced, the situation has improved. The following are some of the more significant benefits experienced by students:

(i) By splitting the development process into two phases, the potential for programmer confusion is reduced considerably. The student can first concentrate on getting a correct and sound design, before becoming involved in detailed efficiency and optimisation considerations.

(ii) It is much easier to spot and correct logical mistakes, since they show up in the design. In fact, we are finding that students make much fewer logical mistakes using this approach. This benefit was particularly advantageous in examination conditions, where students were under pressure and prone to introduce needless complexity. The discipline of having first to produce a design was a distinct help to students under these circumstances.

(iii) The student is able to use a single (HL) programming methodology for all programming tasks. This also helps to avoid the situation occurring where the use of low level design techniques begins to rub off on a student's approach to high level programming!

(iv) Documentation is more straightforward. An AL program can be documented by giving its design, plus the implementation steps (storage allocation details, and the translation rules which have been applied).

(v) A student's approach to writing AL programs is portable. A student is better fitted to program a range of processors, rather than just the one on which he/she has been trained.

3. ADVANTAGES FOR TEACHING

On the teaching side, the main problems with the previous low level method arose from the resulting lack of clarity, structure and simplicity of students' programs. This increased the time required for practical assistance and marking. The new method has brought benefit in this area, as well as in others. The following are the more major benefits experienced by those on the teaching side:

(i) Because we have an explicit and systematic methodology, AL programming becomes less of an art which is 'caught', and more of a technique which can be clearly presented and taught.

(ii) The amount of time which demonstrators spend during practical classes unravelling and debugging students' programs has been considerably diminished. The design can be checked first, and then the translation steps.

(iii) Assessment is easier, since the problem is broken down into a logical sequence of identifiable steps. Marking of practical exercises also benefits from the fact that programs generally have a logical structure, and can be easily followed. With student numbers of the order of 200, this is significant.

(iv) The HL design approach is a very useful medium for presenting algorithms during lectures in a concise yet understandable way (e.g., for developing subroutines for programmed I/O, integer multiplication, etc.).

4. DISADVANTAGES

Certain difficulties were experienced in implementing the method. The following are some problems which were discovered, and which someone might be expected to encounter when adopting the approach:

(i) The main problem initially encountered by students was in grasping the translation process. During the first practical exercise, considerable help has been necessary to guide them through the various steps (although one exercise was generally sufficient for the process to be mastered). In many respects, this difficulty is inevitable; but it can be reduced if students are given some prior experience of AL programs (not written by themselves). This can be achieved by having students examine and analyse AL programs which have been constructed using the technique.

(ii) A successful HL design will often contain some low-level machine-orientated operations (e.g., 'shift SUM left 1 place'). Students can find it difficult to strike the right balance between high level and low level content during the design phase.

(iii) It is possible that a student could produce a design which appears correct, but which is extremely inefficient to implement. This pitfall can really only be avoided through experience, and by having some foresight of the form of the final solution. In the initial stages, students can be judiciously guided along the correct path.

(iv) The approach requires a prior knowledge of a HL language. If the course structure does not provide this, the method is difficult to adapt.

(v) The rewards of using the approach are not always apparent when the programs attempted are all very short. When this is the case, the methodology seems burdensome (though the teacher tends to be more conscious of this than the student).

(vi) Going into detail on the rules for storage allocation tends to confuse some students. We have found it best to cover just the simpler (and more common) cases explicitly. Given this, most students have had little difficulty in coping with more complex cases. In the early stages, optimisation is not actively encouraged.

5. CONCLUSIONS

A major aim of using a HL design method has been to mould the students' approach to writing AL programs so that it becomes natural to think of an AL program in high level terms. It is this 'way of thinking' which will be most beneficial to students who at some time in the future find themselves having to write AL programs. An important feature of the approach is its machine independence.

If the purpose of considering AL programming is not so much to be able to write AL programs as to be able to read them, or just to understand the concept of AL programs, then it is best to concentrate on using the design merely as a means of representing control flow and program structure.

There is no doubt that the skill of students in writing AL programs has been considerably improved by this approach (Notably, the weaker students appear to benefit significantly). From the teaching side, the advantages have also been significant, mainly because the approach enables more efficient utilisation of teaching resources.

ACKNOWLEDGEMENT

Thanks are due to Mrs. Jenny Johnston and the course demonstrators, for providing useful insight and feedback on students' reactions during practical classes.

REFERENCES

1. Jensen, k. and Wirth, N., 'Pascal user manual and report', Second Edition, Springer-Verlag (1978).

2. Dahl, O.J., Dijkstra, E.W., and Hoare, C.A.R., 'Structured programming', Academic Press, New York (1972).

3. Wirth, N., 'Program development by stepwise refinement', Comm. ACM, 14, No. 4 (1971), 221-227.

4. Wester, J.G., and Simpson, W.D., 'Software design for Microprocessors', Texas Instruments Inc. (1976).

5. Halsall, F., and Lister, P., 'Microprocessor fundamentals', Pitman (1980).

6. Crookes, D., 'Teaching assembly language progamming: a high level approach', Software and Microsystems, Vol. 2, No. 2 (1983).

7. 'MCS-80/85 family user's manual', Intel Corporation (1979).

ACM PUBLISHES "TOPICS ON COMPUTER EDUCATION FOR COLLEGES OF EDUCATION"

"Computer Education for Colleges of Education" is the latest in a series of special "TOPICS" on education published by the Association for Computing Machinery. This 115 page document contains reports from fifteen different colleges and universities about their programs for training pre-college teachers, including pre-service, in-service and advanced degree programs. Also included are suggestions for content of specific courses and a report from the ACM Elementary and Secondary School subcommittee which delineates the current situation in computer education for pre-college teachers and makes a number of specific recommendations for improvement.

This issue follows a similar "TOPICS" containing recommendations for computer use in elementary and secondary schools, also produced by the ACM Education Board. Both publications are available from:

ACM Order Department
P. O. Box 64145
Baltimore, MD 21264

Topics: Computer Education for Colleges of Education (1983)

ACM Order No. 812830
ACM members $ 9.00
non-members $12.00

Topics: Computer Education for Elementary and Secondary Schools (1981)

ACM Order No. 812810
ACM members $ 7.00
non-members $10.00