Computing – 11th May 1978

CIS Cobol

'Software developers stand more chance of longterm success if they make an initial investment in portability and can then implement their software on whatever machines are successful. Portability brings advantages to the user. He can get his compilers moved to a new machine, at a price. More importantly he can, if his compiler offers a standard language, move his program from one compiler to another'

Peter Brown, Professor of Computing at Kent University

The pace of change in microprocessor hardware is great; and this presents a challenge to those who write software for micros, since the software continually needs changing to fit new environments. One way of meeting this challenge is to produce portable software. Portable software involves making an initial investment at the design stage to ensure that the software will be easy to convert to new environments. This investment is not trivial; typically portable software costs twice as much as machine-dependent software, and takes twice as long produce. However each time the portable software is implemented on a new machine there is a healthy return on the original investment. If the number of implementations reaches a dozen or so, the implementors can retire to the Caribbean.

One small British company that has embarked on this trail is Micro Focus. It has produced a portable compiler for Cobol on micros. It is called CIS Cobol, and, unlike cross-compilers, actually runs on the micro that it is compiling for. Clearly the demand for good Cobol will be almost boundless, so there is ample scope for success. Not surprisingly, however, there is plenty of competition in such an attractive market. In particular CAP in the UK has produced MicroCobol and there are numerous microprocessor Cobols being produced in the United States.

There are many interesting technical problems facing a small portable Cobol compiler. How is it made portable, and how is the portablity process started? What level of language can be offered? How can the compiler be made fit in a small space?

The purpose of this article is to explain how CIS Cobol has tackled these problems.

Compared with other software, compilers are hard to make portable. This is because not only does the compiler need to be moved to a new computer, but the action of the compiler itself needs to be changed in order that it produce code for the new computer. This problem has been successfully attacked by research workers, and several techniques suitable for use in production compilers have been developed. The most popular is the use of a machine-independent intermediate language. The first part of the compiler, often called the translator, maps the source language into the intermediate language. The second part is either a generator which translates the language down to machine code, or an interpreter, which interprets the intermediate language on the object machine. The attraction of these techniques is that the translator is machine-independent and can be coded in any suitable portable high-level language. The second part of the compiler is usually re-coded for each machine. (If it is an interpreter the second part could be coded in a machine independent high-level language like the translator, but it is not usually done this way, for reasons of efficiency.) Because of this recoding, the work needed to implement a portable compiler on a new machine is not trivial. However the work is, on average, only about one-third that of developing a new compiler from

scratch. Moreover there is proportionately less danger of the host of bugs that permeate completely new compilers when they are written in a shoddy manner.

CIS Cobol uses the intermediate language technique. Since the CIS Cobol intermediate language is designed for micros, where space is tight, it is normally interpreted. Moreover, again because space is tight, the interpreter is hand-coded in assembly language for each machine. There is, however, no reason why the intermediate language should not be compiled if it was to run on a larger machine. The CIS Cobol translator is itself written in CIS Cobol. 'Writing a compiler in itself' is a classic technique of compiler writers. It makes porting much easier. The encoding of the translator can be converted to the intermediate language by using any existing version of the compiler. Given this, once an interpreter (or code-generator) for the intermediate language has been written for a new machine, then the translator is available immediately without further ado and the compiler is complete. There is only one problem: if you need an existing implementation to help build a new one, how do you build the first implementation? In particular how is the CIS Cobol translator converted for the first time from CIS Cobol to the intermediate language?

Micro Focus solved this problem by using a macro processor. General-purpose macro processors can be used, within certain strict limits, to translate one language to another, and hence provide a natural tool. Micro Focus chose to use the Stage 2 macro processor. **One man** was given the task of writing the macros. He worked in a world of his own. He started with no knowledge of Stage 2, and moreover never met anyone who knew Stage 2 until the project was complete. I finally broke the spell by acknowledging some passing acquaintance with Stage 2. Yet he got his macros to work, and achieved this in ten weeks. Granted the macros do not run very fast. Because of space limitations with the micro on which Stage 2 runs, the macros require thirteen separate passes. The end result is that it takes thirty hours of computer time to process the translator - just right for a week-end. When the compiler can compile itself the macros will not be needed any more. Their author will heave a sigh of relief, with a mite of regret that his creation has died. The macros will have provided a quick (in people's time, not computer time) and dirty solution to a pressing problem.

It is not only the compiler writers who want portability. The end user wants it too. The best aids to portability for the user are the international standards. These standards are far from perfect, but life is much more difficult without them. Thus the Cobol user, unless he is happy to be locked in to a single hardware or software supplier, would do well to get an ANSI standard Cobol compiler.

CIS Cobol is compatible with the ANSI standard, but, being a small compiler, the language offered is not up to the minimal Level I of the standard. (Although the language does not fully cover Level I, it actually includes a few more advanced features from Level 2. The extras are good to have but do not fully compensate the omissions from Level 1.) The implication of this to the user is that if he gets Cobol programs from elsewhere he will probably have to reprogram parts of them to run on CIS Cobol. In the opposite direction, running CIS Cobol programs on larger Cobol compilers, there should be no special problems.

CIS Cobol is specially designed for interactive work on a display terminal. The Accept and Display verbs can be used to transfer blocks of data and from the screen. The format of the data is described as ordinary Cobol records.

The first implementation of CIS Cobol was done in conjunction with Dataskil. It runs on the ICL 1500 Transaction System, and needs only 8K bytes with overlaying from backing storage. All sorts of devices were needed to shoe-horn the compiler into such a tight space. One was to make the intermediate language especially concise, so that every bit earned its keep. Since both the compiler and the code it produces are represented in this intermediate language, this conciseness is one of the keys to success.

The ICL 1500 implementation was completed in six months, with a peak of ten people working on the project. The completion of the project was achieved by 'a nice piece of human engineering; a director of Micro Focus was chained to his desk until the work was finished'.

The second implementation has been done on the Intel 8080, and follows similar lines. The minimum storage requirement has been increased to 16K, and this has given scope for streamlining the compiler, improving its user interface and augmenting the language facilities. Compilation speed on the Intel 8080 is about thirty CIS Cobol statements per minute.

An advantage of the portability of the compiler is that if it is moved to an Intel 8086, it can easily be made to exploit the extra features of the newer machine.

CIS Cobol is not currently sold direct to the end user - Micro Focus is not set up to deal in this way - but is sold to vendors of computer systems. The Intel 8080 implementation has already been sold to one such vendor, and, since more and more companies are selling systems based on micros, there is plenty of promise for further sales.

Portability is going to help many software implementers along the road to a life to luxury.