Using a High Level Language as a Cross Assembler

Paul Rutter
Computer Systems Research
Philips Labs
Briarcliff Manor, NY 10510

Abstract

This paper describes how existing high level languages can be used to simplify assembling moderate size programs for new machines. A cross assembler for the Motorola 68000 has been built using these ideas.

Approach

The basic idea is to write a high level language procedure for each type of machine instruction. Operands to the instruction are written as arguments to the procedures. An assembly language program is written as a sequence of procedure calls. Some additional procedures must be written for defining labels, reserving storage, and handling symbolic references. The object program is produced by compiling the entire program and executing it. As each procedure call is executed it produces the appropriate machine instruction for the target machine. If there is sufficient space on the host machine, a simplification is possible by constructing the entire object program in a memory array rather than writing the instructions out to a file as they are generated. This is generally easier and faster than doing file I/O and also simplifies the process of patching up forward references. Symbol table managment can be done in any way that is convenient; I found that using a combined symbol and patchup table was particularly easy. Each table entry stores the symbol itself, an address, and a flag telling whether the address is the value of the symbol (the entry is a definition), or the address of a location that is to be patched with the value of the symbol when it is defined (the entry is a reference). At the end of the assembly language program, a patchup procedure is run which searches the table and makes all the necessary patches to the object program. It also checks for duplicate symbol definitions and undefined symbol references. Then the array containing the machine language program is written out in whatever format is needed.

Advantages

The principal advantage of this approach to writing a cross assembler is that it is reasonably easy to do. Although so-called universal assemblers exist which allow one to define new machine architectures relatively easily (e.g. the UCSD Adaptable Assembler) it can still be a difficult task to become sufficiently familiar with these tools to understand and debug them. Such tools also tend to be large programs and usually constrain the input and output file formats to a particular style which may be unsuitable. In constrast, the high level language approach guarantees that all the mechanics of the assembler are accessible, and assumes only the availablity of a high level language.

The high level language simplifies things tremendously by already having built-in facilities to evaluate arithmetic, conditional, and iterative expressions, manipulate strings, and handle I/O. It also gives one the ability to call arbitrarily complex procedures at assembly time, something which cannot be done even with powerful macro assemblers. Writing conditional and iterative macros, and constructing complex initialized data tables is much easier than with a normal assembler. One additional capablility is that of redefining a procedure for debugging purposes. For example, the procedure which defines labels can be redefined so that in addition to defining the label, it outputs machine code that prints the symbol on the target machine's terminal -- thus obtaining a symbolic trace of program execution with very little effort.

Disadvantages

The main disadvantage of this approach is that most high level languages do not have a syntax similar to the traditional assembler style, which is basically line oriented with tokens separated by spaces. This generally means that it takes a few additional characters to write each assembly language statement as a high level language procedure call than it would in a normal assembler.

Example:

```
   Pascal "Cross Assembler"                Typical Assembler

   l('loop');  add  (r1,r2);               loop  add  r1,r2
               cmpc (r1,1000);                   cmp  r1,#1000
               blt  ('loop');                    blt  loop
```

Doing a prepass over the text to eliminate this problem was considered, but rejected on the grounds that the additional complication was not worth it and might lead to syntactic problems of mixing "assembly" procedure calls with "normal"

procedure calls. Another potential problem with this approach is assembly speed. If the host machine is not very fast and the high level language does not allow for separate compilation of the assembler procedures, then the overhead of recompiling these procedures with each assembly may be painful. For both speed and space reasons, this type of assembler is probably not suitable for writing very large assembly language programs. Since I believe assembly language should only be used in bootstrap situations or for writing kernels, this was not a problem for me.

Experience with Pascal and the Motorola 68000

The technique described above has been used in constructing an assembler for the Motorola 68000. The assembler was written in UCSD Pascal and runs on a DEC LSI-11. The application (a small special purpose interpreter) only requires that about 90% of the instructions and 75% of the address mode possibilities of the 68000 be used, so only those cases are implemented by the assembler. Implementing the instructions, some macros, the symbol/patchup table procedures, and all the error and I/O routines takes about 700 lines of Pascal. Writing the assembler was completed before the manufacturer's cross assembler (written in Fortran and requiring a large host machine) became available.

Certain pecularities of the 68000 architecture were masked by the assembler to give the appearance of a simpler machine. For example, to set a register to a constant is just one instruction in the assembly language, but the procedure implementing it chooses from four possible machine instructions depending on the size of the constant and whether the register is an address or data register. Overall, my experience with the assembler was quite satisfactory; the inability to write procedures in Pascal which take a variable number of arguments or arguments of varying type made things somewhat tedious at times, but not impossible.

Summary

While the idea seems strange at first, using a high level language compiler as an assembler is relatively easy to do and results in an understandable and maintainable tool that is more than sufficient for assembling reasonable size programs. Once one learns to live with the syntax of the host language for assembly language programming, an assembler of this kind offers the power of a high level language system to help one write and debug assembly language programs.