# Producing Computer Instructions for the PACT I Compiler*

ROBERT C. MILLER, JR. AND BRUCE G. OLDFIELD

*Naval Ordnance Test Station, China Lake, Calif.*

*Introduction*

The Project for the Advancement of Coding Techniques (PACT) is an experiment in automatic programming for a high-speed digital computer, the IBM Type 701. The PACT program is designed to produce a machine language program from a symbolic program, which is more nearly algebraic than the machine language. The production of the proper machine language is a basic problem in any compiler.

This paper describes how this problem was solved for the part of PACT which is primarily concerned with the arithmetic operations and scaling. Typical operations are + (add), X (multiply), and some of the logical operations such as TZ (transfer on zero), TP (transfer on positive), etc. The paper is primarily concerned with the underlying philosophy and techniques that were used in designing the program, rather than the actual details of the program.

*Background of the Problem*

Why did the PACT Working Committee want arithmetic operations that would require expansion into machine language? The foremost reason is that we wanted to eliminate programming errors as we knew them at the time PACT was started. While the errors in programming arithmetic and scaling for a machine were not the greatest source of errors, they did account for about 15 % of all the errors that occurred. There were two other reasons for deciding to use a language in PACT which is not in a one-to-one correspondence to the machine's language. First, it was felt that coding in a language nearer to the symbolism of mathematics would be easier and faster. Second, it would not be necessary for new coders to spend several weeks learning the details of the machine.

The program that PACT produces for the coder is the machine language required for fixed point arithmetic and scaling. This program is not interpreted by some other program as in the case of some floating point abstractions which are popular. Two things seemed desirable for the PACT-produced machine language: we wanted an efficient machine language code and we wanted PACT to produce it rapidly.

*The Use of Tables*

Our first approach was to consider the large complex flow diagram which would be required to describe the entire program for expanding the PACT operations to machine language. Several of its characteristics were immediately obvious. First,

* Presented at the meeting of the Association, Sept. 14–16, 1955.

TABLE I

*Machine Operations Required for PACT Operation + (ADD)*

| RS | RA | AR | A (T) | A | S | CLUE | NOTES | | SHIFT |
|---|---|---|---|---|---|---|---|---|---|
| | x | x | x | | | R | Previous Result in T-1 | $Q_f < Q_{r-1}$ | $Q_f - Q_{r-1}$ |
| | | | | x | | R | Previous Result in Acc | $Q_f \geq Q_{r-1}$ | NONE |
| x | | x | x | | | N | Previous Result in T-1 | $Q_f < Q_{r-1}$ | $Q_f - Q_{r-1}$ |
| | | | | | x | N | Previous Result in Acc | $Q_f \geq Q_{r-1}$ | NONE |

Note: (a) − with R clue is the same as + with N clue. (b) − with N clue is the same as + with R clue.

this flow diagram would be so large and complex that it would be quite difficult to make and also difficult to understand when completed. The corresponding code that would then be required would be hard to check out. Finally, it would be extremely difficult to revise such a code.

The idea which led us to our final solution is that it is usually easier to take something apart than to put it together in the first place. This implies putting the intelligence into the program in a fixed form rather than in a form such that the program must always compute the results. An example of this is the use of sine tables—a fixed form—as against the use of a subroutine to compute the value of the sine.

This idea appeared to have some merit: it seemed that the needed information could be supplied by using tables. Let us consider Table 1. It supplies us with the information as to which machine operations to use for several conditions when the PACT operation is + (add). This PACT operation means to "add the operand to the previous result". The letters on the top left are symbols for the machine operations. RA means "reset and add"; A means "add"; AR means "shift the contents of the accumulator right". The A(T) indicates that one of the operands to be added has been stored in a known temporary location. The x's below these symbols indicate which of the machine operations to take for the several conditions on the right. It should be noted that the symbols for the machine instructions are ordered and that if a particular set indicated by the x's in a row is taken then the machine operations are in the order in which the machine should perform them. The column marked CLUE corresponds to the CLUE of the PACT language. The notes indicate whether the previous result is in a known temporary location or in the machine accumulator. The shift column indicates how much the operand is to be shifted. The quantity $Q_f$ is the magnitude of the present operand and $Q_{r-1}$ is the magnitude of the previous result. This table supplies most of the information needed for the PACT operation + (add). The notes below the table indicate that the information needed for the PACT operation − (subtract) is also available in this table.

*The Evolution*

We began by making similar tables for the various PACT operations. These tables were more extensive than the one in Table 1. They include the shifting of

the result of the PACT operation to the magnitude, Q, specified by the coder and also the setup for the next PACT operation. An example of the latter is the placing of the result in the multiplier register if the next PACT operation is X (multiply). This set of tables was then combined into a large table which included all the PACT operations.

The approach to the producing of the machine instructions for a PACT operation was to do it in three phases. The first produces the machine instructions corresponding to the meaning of the PACT operation. The second produces the shift for the result of phase one that is requested by the coder. The third positions the result of phase two as required by phase one of the next PACT operation. This means that we have three tables, one for each phase, each containing the information needed for all the PACT operations to be considered by this program.

The closer we approached the final coding of this part of PACT, the more obvious it became that it would be more efficient not to extract the information from the tables themselves but to use the tables of the three phases to make flow diagrams for the coding. The resulting flow diagrams were extremely simple and easy to code. It is doubtful that we could have arrived at this simplicity if we had worked only with the original complex flow diagram.

The final program for this part of PACT contains four phases and in most cases only requires information from the operation part of two adjacent PACT steps. The first phase always looks ahead to see if a Q has been specified by the programmer on the next step. If Q has not been specified, then a Q is computed by a set of rules, but if Q has been specified, then phase one does nothing and phase two is entered. In phase two the machine language corresponding to the PACT operation is produced. In phase three the machine language is produced to effect the shifting of the result of phase two according to the Q specified by the coder. Phase three does not always produce machine language. Phase four produces the machine language necessary to position the results of the present PACT step in the most efficient position for the next PACT step. Then phase one is entered again and the next PACT step is considered.

*Conclusions*

Does PACT produce an efficient machine language code? We feel that in general the answer is yes. There are cases where it will produce an inefficient machine code, but this is caused by the flexibility which PACT allows the coder; if the coder uses it inefficiently, PACT has no choice. In some cases an experienced coder can outcode PACT but in general PACT codes efficiently and consistently without error. The fact that the code is error-free is a very important feature.

We feel that our approach to coding this section of PACT can be justified. The flow diagrams finally used were very simple and consequently the final coding and check-out of the program were easy. The revision of this section of PACT has been extremely straightforward. It is no problem to incorporate a new PACT operation and this has been done several times. The rapid production of machine

language code is also an indication that the approach is justified. An example of this is that PACT produced approximately 1000 machine language steps for about 300 PACT steps in 25 seconds. This includes not only the time spent in generating the machine code but also the time required to read the data from the magnetic tape and then write the results on another tape.

We are convinced that the approach which we have discussed can be used to advantage when producing machine language code for other large scale digital computers. It should be fairly easy to use this approach on the IBM Type 704 and even on computers that are not single address type computers.