Conclusions After Using the PACT I Advanced Coding Technique*

I. D. GREENWALD

The RAND Corporation, Santa Monica, Calif.

and

H. G. MARTIN

Douglas Aircraft Comany, El Segundo, Calif.

Early Experience with PACT I

Since PACT is a quite new system our experiences in using it are rather limited (in particular, at the time of the writing of this paper—early August 1955—the final system had been in operation for less than two months). Of the ten to twenty problems coded in the system we can, however, make some general comments. Going in natural order from flow diagram to final running we have observed the following:

1. Coding is easier and faster. (For most problems this time has been reduced from days to hours).
2. The time required to compile a PACT program is less than the time required to assemble a program of comparable length using our old assembly methods.
3. The number of programming errors is small (on some problems zero).
4. Programming errors are easy to find and correct.
5. The number of machine language instructions generated is comparable with that for hand written codes (sometimes even less). This feature was somewhat of a pleasant surprise to us.
6. The codes generated compare favorably from an efficiency point of view. (It should be noted that at times these generated codes will be more efficient than hand written ones due to the failings of humans in writing the latter).

From the point of view of many of us, (in particular, those of us in the Aircraft industry) the greatest contribution of PACT is a direct corollary of the preceding remarks: elapsed time from problem formulation to problem production is substantially reduced (on some problems, we have estimated as much as a ten-fold reduction in elapsed time).

There is a saying at one of the member installations that old codes never die, they just develop new parameters. Another contribution that PACT makes is in the realm of changes. New parameters quite often mean new scaling. PACT coded problems can be rescaled in a matter of minutes by merely redefining the variables involved and recompiling. Changes in equations are usually simple to make due to the master routine sub-routine approach inherent in the system. An additional feature which had not been anticipated eases the debugging problem—we discovered that break-point print out of any quantity is easily accom-

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

plished. We might add that we have had no real use for this feature to date since we have had no serious debugging problems.

One of the most important aspects of PACT is related to the experience required of those who use it. We won't go so far as to say that experienced programmers are no longer needed. However, we do know that inexperienced people can now be used efficiently to program production jobs. We will cite one example of this point. The person involved had fourteen days prior experience. He was given a job involving matrix addition and multiplication and the evaluation of a determinant and was asked to log his time carefully. Following are his figures:

1. Time spent reading the manual	= 4 hours
2. Time spent in setting up the job with the aid of the manual	= 3 hours
3. Time spent in actual coding	= 1 hour
4. Compilation time	= 3 minutes

A second example of the inexperienced programmer's usefulness is quite interesting. A new man with no previous experience in high speed computing was given a PACT manual and a problem to do on his first day at work. Three weeks later he had coded, checked out and run a job which had 1000 machine language instructions on the final compilation. It might also be noted that at this time the system was incomplete and that a good portion of the troubles this man had were due to errors in the compiler.

One of our biggest worries in the course of writing the compiler was the question of whether we were really eliminating most coding errors or just creating new types of mistakes. To a certain extent the latter is true—but most of the errors in using the system that have occurred to date result from three failings:

1. The first people to try PACT were working with an incomplete system and practically no manual. We like to call these errors *misinformation*.
2. The coding sheets first used were poorly designed and gave rise to clerical type errors (such as writing things in the wrong colunns).
3. Some of the experienced programmers read into the system things that are not there (such as misuse of the loop writing ability).

In general, errors take on the order of minutes to find and correct. An interesting aspect of PACT, that had not been completely foreseen, is that during compiling a good many coding errors are detected and flagged. For example, inconsistent scaling will cause an error stop in the compiler. Some other errors that are caught are: Misnaming variables, referring to non-existent steps, wrong operations, etc.

We shall cite one more experience: One of the members of the working committee programmed a routine to extract the square root of a matrix using an iterative technique. Coding time was one hour. He compiled the code in three minutes (some 50 PACT instructions generated 500 machine language instructions). On trying the program he discovered one error he had made and an error in one of the sections of the compiler. He phoned the person concerned with that section and a few hours later received a return call with the correction needed. He recompiled and ran the job which, by the way, had never been successfully programmed at that installation. Total elapsed time: less than one day.

Suggested Improvements

It is extremely difficult to find fault with a system that has grown before your eyes from a mere idea into a working model, a system you helped author from its infancy and guide through its adolescence. Yet, there are many shortcomings. In its present state the vast amounts of secondary storage (tapes, drums and cards) are not readily accessible, problems with large ranges of magnitudes are cumbersome (no floating point), problems with small ranges of magnitudes are wasteful of storage (no half-word arithmetic), reading and writing large arrays is laborious, operations on subscripts are limited, and new bookkeeping problems have arisen.

Secondary Storage: The problem of utilizing secondary storage can be solved temporarily by writing routines for use with PACT. For instance, the READ and LIST operations as they now stand make no use of the FACTOR field. It should not be too difficult to recognize codes in this field to mean read or write on a tape or drum. Easily remembered codes such as LIST T1 or READ D4 could be used to cause a sub-routine to be generated which would write a record on tape unit number 1 or read from drum 4, respectively. And with the Identification (ID) operation blocks of data, regions of instructions, sub-routines, or combinations of these could be placed on a tape with comparative ease. As a point of interest, suppose that we have arrays in our machine called I, B, and N and a region of instructions called HAD. To place these on tape 2 all we have to do is write:

```
LIST    T2
ID      I
ID      B
ID      N
ID      HAD
```

This is just one way one can use the secondary units, although it is not a very efficient way.

Floating Point Arithmetic: The next suggested improvement is the incorporation of floating point arithmetic. Floating point operations are essential to some problems, and they can be a great aid in scaling. If we can devise a compiler which will take the same instructions as the present one and carry the maximum number of significant digits at the same time, we have a system for quick problem solution and also a way to analyze quickly the magnitudes of intermediate results. Thus, with one method of writing we can have both accuracy and efficient use of machine time.

Additional Input-Output Devices: The problem of large amounts of data brings to mind another suggested change—additional input and output operations. Many people object to the present READ and LIST operations on the basis that it is difficult to read or list matrices and vectors. This type of input-output was not provided in the present system because the committee felt that each organization already had library routines available to perform these functions, and experience has proven that the LIB operation makes the incorporation of these routines a simple matter. Since these existing routines are so easy

to incorporate into the system, we feel that other suggested changes, such as the automatic secondary storage assignments, are far more important.

New Bookkeeping Problems: When we were setting the specifications for this advanced coding system we all agreed that one of our major problems was to relieve the programmer of the difficult bookkeeping tasks that he had in the old-fashioned programming systems. But, while the major portion of this burden has been removed from the coder some of the problems still remain. PACT-1 allows the coder to use many different variables, but it insists that the coder be consistent. He must place the symbols for the variables in the same relative position within the fields, for, if he doesn't, the compiler thinks it is a different variable. He must write numbers to the right of each field, for, if he doesn't, the compiler thinks the number is larger than he intended it to be. And he must write symbols in a distinct manner so that the key puncher will punch the proper code. These problems are trivial compared to those of other systems where the coder has to remember at least the relative or symbolic locations of all of his data and instructions. In fact, experience and good coding forms may reduce PACT's bookkeeping problems to insignificant amounts. Still, we have the problem, and it is certainly something to consider when thinking of possible improvements.

Substantiation of Early Decisions

Now that we have examined some of the shortcomings of PACT, let us look back at some of the decisions that were made while we were setting the specifications for the system. At its third meeting (12–2–54) the PACT Working Committee had agreed that the language of the coding system should use mnemonic symbols, be concise, be flexible and be easy to learn. It had agreed that the product should be efficient, handle systematic arrays up to two dimensions, permit easy incorporation and use of both library and personal subroutines, permit direct descriptions, be capable of being used in parts, and produce routines with a high salvage value. With the limited experience that we have with PACT, we think that we can in all truthfulness say that the coding system which we have developed has all of these qualities. It is interesting to note that it was not until January of 1955 that the initial form of the present coding format was agreed upon and that this format is identical to the one we are now using with great success. There was a long debate about this format, for some of the committee members wanted horizontal writing while others wanted vertical writing. After many hours of heated discussions we finally decided that the flexibility of the vertical format was more desirable. We are happy to relate that this decision has been substantiated by usage, and we hope the other decisions will also be.

Conclusion

In closing we would like to mention again that this project was accomplished through joint effort on the part of many companies where competition is ex-

tremely high and almost all discoveries and techniques are considered proprietary information. In the midst of this highly competitive industry, a group of far-sighted men in charge of computing groups realized a need for a better coding technique and established what is now known as the Project for the Advancement of Coding Techniques. The formation of the project was unique, and, we feel, the result of the project is also unique. And it is our sincere desire that the spirit of cooperation exemplified here will continue and possibly prove contagious.