Pact IA*

T. B. Steel, Jr.

The RAND Corporation, Santa Monica, Calif.

The second effort of the Project for the Advancement of Coding Technique has been the construction of a compiler for the IBM 704. This effort, entitled PACT IA, grew directly out of the experience engendered by the development of the PACT I compiler for the IBM 701.

The most compelling reason behind the commencement of this new programming task was clearly the obsolescence of the 701 or, contrariwise, the advent of the 704. Additionally, however, several inadequacies in PACT I were evident. Among the principal concerns were the complexity of the rule book, insufficiency of error detection devices, difficulty of partial recompliation, rigidity of subscript manipulation, and total lack of automatic secondary storage assignment. The decision was made to code a compiler for the 704, using essentially the same language as PACT I, the identical approach to compilation from a machine viewpoint and to concentrate on simplification of the rules and error detection.

Simplification of the rules carries with it, necessarily, a relaxation of subscript constraints and permits flexibility in the matter of recompilation. Indeed, the less complicated the rules, the easier it is to locate violations of the rules. Thus, improvement of the PACT I manual became of prime importance in the new effort. A significant guide to the limits of reasonable rule changes was present in the desire to maintain compatibility with PACT I wherever possible.

After a review of the relevant discussion preliminary to PACT I, the problem of automatic secondary storage assignment was relegated to limbo. The immense difficulty of this problem is evidenced by the fact that logically it has a good deal of similarity to the celebrated "four color" problem. There is some hope that the availability of multiply buffered input-output systems will bring this problem in the range of the soluble, but for the moment it was disregarded.

The availability of built in floating point instructions on the 704 required a new decision on the question of what type of arithmetic to use. The fixed point arithmetic of PACT I has a variety of desirable features but leads to complexities in description and restrictions in practice. It was concluded that all arithmetic would be floating point but care was taken to permit later addition of fixed point operations if the need became pressing. Within this framework, then, PACT IA was coded.

Viewed as a mathematical language, the PACT IA language is akin to the descriptive language used for preparation of sheets for hand computing, having numbered steps, single operations and single factors, subscripts for notational compactness and direct naming of factors. From the machine code point of view,

* Received June, 1956. This paper is a sequel to the series of papers on PACT I appearing in the preceding issue of this Journal. Cf. Editors' Note, p. 265.

the PACT IA language speaks as a single operation, single address, single arithmetic register sequential machine, having two tags or subscripts for address modification and conventional conditional transfers of control.

In addition to the fundamental algebraic operations, the PACT IA language recognizes certain elementary functions, and provision is made for the introduction of other functions by use of library programs. The language can also call for reading of cards and printing lists.

The basic data element in the language is the variable. A variable may refer to a scalar, a vector component, or a matrix element. Particular elements are referred to in the conventional manner through the use of subscripts. The language has provision for the initial establishment of subscript values and the facility for changing and testing these values.

From a general view, a PACT IA code will appear divided into regions, each region being virtually autonomous. This was done both for compiling convenience and from a deep conviction that sound coding practice calls for a master routine-subroutine approach. Experience with PACT I has confirmed the validity of this position, at least this type of language.

The orthography of the PACT IA language is quite straightforward. One step, corresponding to one operation, appears on a line, resulting in what may be termed a vertical format. On this line, in addition to the region name which may be three or four characters, is the step number, the name of the operation to be performed and the factor, together with its associated subscripts, if any. The operations, of which there are forty altogether, have simple mnemonic abbreviations of four or fewer characters. Factors may be the names of variables, numbers or, in the case of control steps, step numbers.

Variables may be named in one of two ways. Input variables are named on a separate form known as the variable definition sheet where the names of the variables are listed, together with the maximum extent of any subscripts. Computed variables are named by their appearance in the code as factors of an "equals" operation where the result of the previous calculation is given a name. Intermediate results that are useful only in a given region need not be named directly but can be referred to as the result of a given step.

Provision is made, by the inclusion of an "identification" operation, for the carryover of variable input into closed subroutines. An interesting point here is noted by realizing that a basic linkage to a closed subroutine must perform certain generative functions. Thus, in a limited sense, PACT IA is a compiler that compiles compilers.

In sum, the PACT IA language is a machine-like language where the essential arithmetic phase stands out and the logical manipulation and cross-referencing has been disguised and submerged for easier writing and reading.

Fundamental to an understanding of the method of compilation is the scheme of storage allocation. Five categories of storage are recognized. First, of course, is the storage area containing the machine instructions that comprise the target code. Second, there is a set of words containing numbers, those constants required in the execution of the target code that are independent of the input data

and may, hence, be generated at the time of compilation. Third, a sufficiently large block of storage must be available to contain the various variables, both input and computed. To allow the possibility of overlapping storage in this area, a set of variable constraints may be provided by the coder, thus allowing a more efficient storage utilization than might be possible if the computer were to attempt the necessary analysis. The other two categories of storage are both erasable and hence are overlapped. Temporary storage for the use of the generated code is provided as well as perishable storage for subroutines. The distinction is primarily to simplify the overlap question.

The general procedure for compilation is a series of stages, each consisting of one or more sequential tape passes. At each stage, all the processing that can be performed in parallel within the limits of the high speed storage is done. Let us consider these stages in turn.

The initial process is an input and conversion stage during which the PACT IA language code is brought into the machine, either from a card source or a tape independently prepared by peripheral equipment from a card source. The information read in is converted to a conventional character code for ease in handling and written on a tape conveniently blocked with the basic unit being a PACT IA step. This basic unit, hereafter referred to as a step-record, is added to and modified by the succeeding stages until it contains sufficient information to permit assembly of binary instruction words in the usual manner.

The second stage is an abstract assembly. This operation is just what its name suggests, a standard assembly operation of associating locations and addresses but done in the PACT language. At this stage also, certain logical features of the PACT code are checked such as the legitimacy of transfers of control and the occurrence of cyclic sequences of transfers that would produce unending loops.

The third stage of compilation is the variable assignment stage. Here the variable definition cards, naming the input variables, and the variable constraints are read into the machine. Subject to the constraints, storage for these variables and those defined by the code is allocated. Then another tape pass is required to correlate the storage assignments with variables appearing as factors. At the conclusion of this stage, the code has really been completely assembled in the abstract PACT IA language and the remaining three stages are the translation phase.

The first translation stage is an expansion into 704 machine language of the arithmetic and control operations. Here two and sometimes three consecutive step-records are examined and the orders necessary to perform the first and position the result for the next in the 704 arithmetic registers are generated. Certain mild inefficiencies in the calculation sequence are permitted here to allow flexibility in transfers of control.

The next stage is the subscript expansion. Here the necessary instructions to control the establishment and testing of subscript values are generated. A rule has been established that subscripts are defined only for a single region and, if used in another region, must be redefined. Thus restricted, it is possible to make a rather thorough examination of the calculation sequence in a region in

order to generate loop instructions efficiently. Both for simplicity and reasons of efficiency, all address modification is done by means of index-registers or B-boxes. The problem of efficient assignment of index-registers has been only partially solved but the scheme used is such that conventional iterations used in most computation will be efficient.

At this same stage, the instructions essential to subroutine basic linkages are also generated and at the conclusion of the subscript expansion stage, all the instructions necessary have been generated. Up to this point, however, all addressing has been done by means of a special relative address scheme with the regions serving as fundamental blocks. It remains only to reduce these addresses to actual machine addresses.

The final stage, again an assembly process, is identical, logically, to conventional assembly programs, reducing the internal relative referencing scheme to a machine language relative binary code. This final assembly also provides an origin table for the machine relative locations and a loading program. All this is punched into binary cards. Thus, thirty-one tape passes and five to forty minutes after the start, the PACT IA code has been translated into a 704 binary code ready for debugging.

Three aids are provided for this debugging operation in addition to the language itself which is designed to help the coder eliminate errors during the coding operation. First is a printed list of the PACT IA code together with some correlative information on the generated code. Second, the compiler, in the process of generation, checks for certain possible errors and prints a list of these. Finally, a postmortem print designed to give information in the original coding language, the PACT IA language, is provided. Considerable experience is needed before adequate evaluation of these debugging aids can be made, but preliminary study suggests that they are at least a step in the right direction.

In view of the overriding importance given to the rule book simplification, a comment on the organization of the PACT IA manual is in order. The first part of the manual is designed to provide only that information essential to the coder for construction of elementary codes. The rules are stripped to the bone and avoid exceptional cases. Then an appendix is provided, giving a complete description of the rules for the more sophisticated coder. Finally, a document containing the background information on how the compiler works and why various decisions were made is to be available. This will be designed for the expert. This approach to describing the system is, hopefully, a satisfactory way of covering the wide range of experience and programming maturity that has a potential use for such a powerful tool as the PACT IA compiling system.

In summary, the committee feels that PACT IA is more than an experiment to learn how to compile. We are looking forward to a wide variety of uses with a concurrent feedback of experience that will be of aid in the future advancement of coding techniques.