FREDERICK G. KILGOUR

Symbol-Manipulative Programing for Bibliographic Data Processing on Small Computers

Drawing upon experience in the processing of bibliographic data on a small computer, the author makes suggestions for appropriate programs for production runs. Subjects covered by these suggestions include programing language, procedures of program preparation, coding and flagging, field length, and hints on effective program application.

RELATIVELY LITTLE has been written on symbol-manipulative programing, and papers that have appeared discuss "high-level" languages to be run on large machines.[1] Furthermore, publications in the general field of language-data processing are concerned, for the most part, with machine translation or information retrieval from natural language—two procedures which require large computers. However, in 1963 Don S. Culbertson proposed that COBOL should be the standard computer language for library data processing,[2] and in the same year I. H. Pizer, D. R. Franz, and Estelle Brodman raised objections to Culbertson's plea.[3] These two discussions appear to be the total literature on programing for bibliographic data processing. This paper is based on, and will report experience in programing the processing of bibliographic information on a small computer; namely, an IBM 1401 having a 4 K core, two tape drives, and advanced programing features. Most of the programs are production programs and are run on a daily production schedule, but some are run monthly. Therefore, the paper presents suggestions for programs for production runs, not for "one-shot" programs.

Perhaps the cardinal principle of a bibliographic data processing system is that the machine must not be allowed to impose its characteristics on the data or the procedure. In the case of library procedures, long experience has accrued; indeed, libraries are thousands of years old, while books have been printed for hundreds of years. Lessons learned empirically, decades and perhaps centuries ago, should not be discarded be-

[1] "ACM Conference on Symbol Manipulation, May 20-21, 1960," *Communications of the ACM*, III (April 1960), 183-234; "Design, Implementation and Application of IR-Oriented Languages," in "Papers . . . at . . . Princeton, N.J., October 20-21, 1961," *Communications of the ACM*, V (January 1962), 8-46; [Symposium on Symbolic Languages in Data Processing, *Rome, 1962.*] *Symbolic Languages in Data Processing* (New York: Gordon and Breach, 1962), p. 114-85; Philip M. Sherman, *Programming and Coding Digital Computers* (New York: John Wiley and Sons, [1963]). p. 294-326; Daniel G. Bobrow and Betram Raphael, "A Comparison of List-Processing Computer Languages," *Communications of the ACM*, VII (April 1964), 231-40.

[2] Don S. Culbertson, "Another Tower of Babel?" *Library Journal*, LXXXVIII (March 1963), 940-943.

[3] Irwin H. Pizer, Donald R. Franz, Estelle Brodman, "Mechanization of Library Procedures in the Medium-Sized Medical Library: I. The Serial Record," *Bulletin of the Medical Library Association*, LI (July 1963), 331-38.

Mr. Kilgour is Associate Librarian, Yale University.

cause of machine characteristics or because of difficulties in program planning or coding. For instance, it is not necessary to sacrifice lower-case letters in printout from high-speed printers; furthermore, final printout can be done on typewriters controlled by computer-produced punched cards or punched tapes, thereby achieving upper and lower case print.

Perhaps the most important characteristic of the computer to be used is that it should have variable word length storage in its internal memory. In short, its core should be able to store variable length words characteristic of natural language. In the years ahead, language-data processing will be being done on such machines, and a start made now should be in the right direction. The speed of the computer matters little, and indeed a slow computer may be better than a fast machine and most certainly will do better if high speed is acquired with the penalty of fixed word length.

A programing language to be used for bibliographic data processing on a small computer must be machine oriented. In other words, the programing language of choice is a symbolic-assembly language as close to machine language as available. Macros can be wasteful of space, and if used, should be only in the form of closed subroutines or routines used but once in the main program. The so-called high-level languages are certainly easier to learn but are inefficient and must be run on large machines. Moreover, these languages restrict a computer in the operations it can perform so that they lend a provincial quality to their programs, whereas symbolic-assembly languages make possible the writing of sophisticated programs taking full advantage of the computer's capabilities. Indeed, many workers employing large computers for language-data processing have increasingly used machine-oriented languages.

Culbertson, in the paper referred to above, urged that librarians adopt COBOL for library computer programs. He based his suggestion on the attractive premise that use of COBOL would yield standardized programs that could be run on a large array of computer models produced by various manufacturers. However, he recognized that to achieve such standardization only one of the COBOL dialects could be used. To compile COBOL, a slightly larger computer configuration must be available since four tape drives are required with a 4 K core. Furthermore, COBOL is a problem-oriented language and such languages are not inherently suitable for symbol-manipulative programing. Still, Culbertson was quite correct in pointing out that use of a symbolic-assembly language may mean that an extensive and expensive reprograming task could accompany a change in equipment.

The output of language-data processing is often characterized by having one section which is fixed in its two dimensional design and other sections which may be varied two dimensionally. For instance, printed text on a page may vary in number of lines and in line length from book to book, but it is essentially a block of words. On the other hand, the running head and page number vary extensively in position. An analogous fixed and varied format is the card found in the conventional library card catalog. Here, bibliographic data describing books occur in the same relative positions on cards in most libraries. However, call numbers for books and headings for subjects, titles, editors, etc., are placed in varying positions according to individual library practices. Programs can be written for formats having varied and fixed characteristics by assigning the processing of the fixed characteristics to the main program. In addition, a generator program can be written which will operate on data from a control card whereby the generator will write brief programs to determine position of that part of the format which may vary from one product to another. Such generator programs give flexibility to bibliographic data processing. Since language-data processing requires relatively large work areas in the computer's internal memory, the generator should be written in the work area and erased after it has written its programs to be used by the main program. Program tables can also be devised to attain similar flexibility.

It is not possible to employ a monitor program in the conventional sense of that phrase in a small computer, but a somewhat similar effect can be attained by employing a systems tape, if a program is too large to fit into core. In such a circumstance, the program should be divided into three or more sections, one section being common to the others. In assembling the program, the common section can be assembled either first or last. Should it be first, the others should be written on tape with one of the tape sections in core at the start of processing. In the case of three sections, the third section may be left in core and written on tape during processing and the second section called in to replace it. This procedure may, of course, be reversed, for when the third section is needed, the common section containing calling instructions can bring it back into core. Also, the systems tape may be used to store data for recall in subsequent processing.

If a large amount of processing is necessary, several programs may be needed. In such a circumstance, the programs can be planned for a sequential system wherein they are linked together. The program system that yielded most of the experience on which this paper draws consists of four major programs, two of which have generators, and two that employ a systems tape. The four major programs have been linked together so that when the first program has completed its processing of the data, it loads the second program automatically. The third and fourth are similarly loaded. Such a program system greatly simplifies operation, for the operator need only place the data cards after the first program and depress the load button.

Symbol-manipulative programs are characterized by their high percentage of logic instructions. However, symbol-manipulative programs for language-data processing have an additional typical feature in that the core position of data is locked into the logic. It is this characteristic of logic interlocked with position that places exceptionally heavy requirements on indexing features of a computer and on the programer to contrive efficient, closed subroutines to maintain index registers at correct values.

Another characteristic of such symbol-manipulative programs is a high density of labels. If the assembly program has a limitation of the number of labels it will process in one iteration, care should be taken to keep the number of labels below that limit by use of actual address or by other means. Reiterative assembly of one large program for processing bibliographic information on a 1401 requires 25 minutes and uses a box and a third of punched cards. Since de-bugging may sometimes constitute as much as 90 per cent of the total time of development of a computer program,[4] it is important to minimize assembly time.

If bibliographic data is to be sorted and arranged alphabetically, such sorting will constitute a major programing problem. Moreover, to obtain differentiation in sorting, it will most likely be necessary to sort on more characters than originally estimated. The technique for doing such sorting is, of course, to establish a separate sort control on which the computer actually operates. Initial articles should be removed from this control, and when various languages are involved, separate article tables for each

[4] Sherman, *op. cit.*, p. 398.

language must be established in the program. Also, various spellings of "Mac" and many other letter, or letter and diacritical mark, combinations should or should not be brought together in the sort control, depending upon the filing rules adopted. Finally, there will be some elements by which filing is to take place such as numbers in a title. An example is *1066 and All That.* If this title is to be sorted as though it were *Ten Sixty-six and All That,* there is no program which can instruct the machine to do it. Here it is necessary for the person processing the data to write out a separate sort control in the form of TEN SIXTY-SIX AND ALL THAT, and the programer must provide for a signal which will enable the computer to recognize that it must use this human determined sort control.

Another characteristic of language-data processing on a small computer is that the preparation of the data involves linking it with the program to be used. In general, it is necessary to identify each internal category of data with an exclusive code, and to assign such codes so that the program can recognize when processing moves from one category to another. Also, codes must be designed to identify sections of data within each category. These three simple coding requirements yield enormous flexibility in processing.

Codes or flags should be positive and exclusive. Negative flags, such as blank spaces, should be avoided, for experience has shown that they can be troublesome. In establishing flags, one should use symbols that occupy but one column on a punched card. As a concomitant of this rule, upper-case characters should not be used as flags, for in many circumstances it will necessitate the use of another flag to indicate upper-case. Hence, the flag would actually occupy two columns. Experience has shown that such two-column flags are difficult to program. Also, symbols should not be used which are in the regular printout of the system; it is preferable to use non-printing characters despite the unquestionable fact that they complicate the de-bugging process since they do not appear in a post listing of a program.

A further caution about preparing data for processing is to avoid having fixed internal fields. In most, if not all cases, it will be necessary to have a fixed overall record length, but it is unnecessarily restrictive of the data to have fixed fields within the record length, albeit that such fixed fields facilitate programing.

Finally, if a major bibliographic data processing system is to be established, it is most desirable that there be one operational application made early in the development. Such an application will reveal various difficulties which it may be necessary to solve by altering the preparation of the data, by elaborating the program, or by using both these techniques. Should it be necessary to change the data preparation, much future grief will be avoided, and it is for this reason that an early application is advantageous. ■■