1964

Simplified language and coding for limited data processing applications

Jerry Ronald Tennant
Iowa State University

SIMPLIFIED LANGUAGE AND CODING

FOR

LIMITED DATA PROCESSING APPLICATIONS

by

Jerry Ronald Tennant

A Dissertation Submitted to the

Graduate Faculty in Partial Fulfillment of

The Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Electrical Engineering

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

Head of Major Department

Signature was redacted for privacy.

Dean of Graduate College

Iowa State University
Of Science and Technology
Ames, Iowa

1964

INTRODUCTION

This dissertation describes the characteristics of a proposed data processing system which could be operated by an inexperienced individual. The language requirements placed upon the individual are minimized by making it unnecessary for him to write programs or provide level designators on the input data. This is accomplished by having a set of programs stored in the computer memory where they are accessible to the system operator by some form of selection scheme. Furthermore, the operator is essentially forced to supply the input data in a form acceptable to the system by making him an integral part of the input loop. This has been done by implementing a new verb in the system language which will automatically provide the correct level designators to the data as it is input by the operator on an ordinary typewriter.

The constraints imposed by a system operating in the above manner are analyzed in relation to the requirements of the business environment to determine if there is a class of problems in which this system could be effectively utilized. This question is answered in the affirmative by presenting two examples of the application of a fixed-program data processor to the solution of problems in a typical small business. Analysis of these examples and the comparison of a few typical programs written in COBOL and EVE indicate that the EVE language is more suited to the type of application proposed in this system. EVE is a high-level problem oriented machine language. Programs in the EVE language are easy to write, compact in length, and require no translation by a compiler before implementation in hardware.

The two sample cases also illustrate one technique of program selection. The program names are encoded with command words, operands, and file names peculiar to the particular business problems being programmed and placed on a panel of selection buttons. Program routines are presented to illustrate the technique of data level designation and the usage of the new verb added to the EVE language.

EVOLUTION OF COMPUTER LANGUAGES

Machine Language

In general, a computer can be conceived of as a numerical-transformation machine. Numbers are the inputs to it, and the computer transforms these numbers into new numbers, which appear as the outputs (8). The automatic character of an automatic digital computer consists in its ability to perform definite sequences of arithmetic and other operations (such as looking up entries in a table). The operations in such a sequence must be specified by a corresponding sequence of computer instructions called the program (14).

In the early days of computers, all programming was done in machine language. This implies not only that the programmer was confined to only those kinds of instructions which the designers of the machine had built into its comprehension, but also that he had to work in the numeric symbology which the machine accepted (3). The ENIAC was based on the decimal system, and instructions for it were written out in a straight number language in decimal digits. For instance, 792 and 438 might represent "ADD" and "MULTIPLY". The symbol "ADD" was not an acceptable one to the machine. Computers developed after the ENIAC used the binary system, where combinations of 0's and 1's served the same purpose as the decimal digits in the ENIAC (12).

Using the machine language, the programmer was required to assign the absolute physical location and exact contents of each individual cell within the memory of the machine (1). The only way he could find information in storage was to know the address of the cell which contained it.

In principle all computer programs could be written in machine language. In practice such a situation is neither efficient nor in some cases even possible. For example, programs which are several thousand words long become impossible in machine language because an overwhelming effort is required to add or delete even a single word from the program without losing the sequence of control (15). Further difficulty arises from the fact that a programmer reading a program written in machine language obtains no sense of the operations being directed or of the way in which the computer will execute them without extensive practice and training in reading the language (1).

Symbolic Language

The first relief from these problems came with symbolic coding. In such a programming language, the machine accepted such symbols as "ADD", called mnemonic abbreviations (3). Mnemonic means "assisting the memory", and indeed programs written in a symbolic language were moderately easy to read with minimum preparation. Furthermore, the programmer was no longer bound to machine addresses. Now he could assign arbitrary mnemonic designations such as "WAGES" for the memory location containing the number representing the wages paid to someone. This procedure is known as symbolic addressing.

In preparing instructions in symbolic language, the programmer finds it much easier to make insertions and deletions of instructions than it was with a machine language. In symbolic languages, insertion can be made simply by assigning new symbolic addresses to the new instructions

and deletion by dropping instructions and correcting the symbolic addresses of the adjoining instructions (1).

The internal machine structure, however, did not change, except that with a simple table of correspondence, the machine could transform one symbol into another ("ADD" into "010"). The routine which accepts a set of symbolically coded instructions, translates them into machine language, and at the same time also assigns the symbolic addresses to absolute addresses is generally called an assembly routine (11). The particular selection of commands available in the symbolic language is slightly more extensive than that available in machine languages (1). For example, an assembler may allow for macro-instructions, which are kinds of instructions which the machine cannot execute directly but which must be broken down into combinations of basic machine instructions (3). However, the instructions with which an assembler deals are usually actual machine instructions expressed in symbolic form. That is, there is in general a one-to-one correspondence between the symbolic instructions written by the programmer and the machine instructions produced by the assembler.

Thus, the assembler helps with the symbology problem confronting the programmer, expands the number of apparent operations available from the machine, and eases the chore of assigning storage locations in the memory (3). Assemblers as defined above were in general use at least as early as late 1953. By 1955 they had reached a high degree of elegance for "machine language coding" and have not changed markedly since insofar as the language acceptable to them and the running times are concerned (11).

Automatic Coding Languages

Symbolic languages and the assembly routine concept developed into advanced programming techniques as attempts were made to lighten still further the load of the programmer or coder. Automatic coding languages use instructions that appear very much like ordinary English and thus are quite distant from machine languages -- much more so than the symbolic languages. Their use enables the programmer to make full use of the capabilities of a computer without knowing anything about the mechanical process of translation from the language to the detailed code required by the machine (6).

Because of the distance of automatic coding languages from machine languages, some means must be used to bridge the gap. A program written in symbolic language or automatic coding language is worthless until it is translated into the machine language of the computer for which it was written (1). Instead of being translated by the programmer, however, it is translated by the computer itself. The use of an assembly routine to perform the translation for a symbolic language has already been discussed. The means used to accomplish the translation in the case of an automatic coding language is to prepare or precode the computer with an intermediate, or automatic, program which is always available in the computer memory (8). This program is generally known as a "compiler" or "processor".

In converting from an automatic language such as COBOL into binary terms, the machine must play two roles. First it serves purely as a translator. It receives a "source" program on magnetic tape written in COBOL, and performs, through the use of its compiler routine, the series

of operations required to spell out the same instructions in its own binary language. The result is the production of another tape, the "object" program (12). The object program is the actual machine language program which the computer will execute in performing its computation or data processing task.

Building the translating routine, or compiler, is simply the problem of constructing a routine which makes a given physical sequential machine imitate the behavior of a nonexistant, but still sequential, machine whose language is the particular automatic coding language.

The compiling routine is an extension of the assembly routine. Generally a compiler permits more complex macro-instructions than an assembler, and it often excludes machine instructions (even in symbolic form) from the language which it can accept. While the assembler generally deals with each instruction independently of all others, the compiler attempts to capitalize on the information which is contained in the structure or logic of the problem; the context of each instruction is important. Commonly, a compiler is problem oriented in the sense that it accepts as input the language and operations of a particular class of problems (3).

There are many automatic coding languages (with their compilers) in existence. For example, the IBM FORmula TRANslating system, FORTRAN, is an automatic coding language which has a source language closely resembling the ordinary language of mathematics, and a processor which converts source programs written in the FORTRAN language into machine language object programs. The FORTRAN project was begun in the summer of 1954 and has never really ended. Its goal was to enable the programmer to specify a

numerical procedure using a concise language like that of mathematics and obtain automatically from this specification an efficient machine language program to carry out the procedure (11).

The success with algebraic language compilers, such as FORTRAN, led to an international effort to define a universal ALGOrithmic Language called ALGOL. A program written in terms of a particular language can be run on any computer for which there is an automatic program that can compile or interpret the language. This fact indicates the feasibility of the concept of establishing a universally acceptable international automatic language, analagous to the universal written language of music (8). As a result of a series of conferences toward this goal, the "Report on the Algorithmic Language ALGOL" was published by the Association for Computing Machinery in 1960 (13). The purpose of ALGOL is to assist in the programming of primarily algebraic numerical computations.

In the past few years, a number of systems especially designed for data processing applications have been developed. One of the languages most widely used by these systems is the "COmmon Business Oriented Language", or COBOL (9). It contains expressions and statements that more closely parallel the clauses, phrases, and statements of ordinary English usage. COBOL, like ALGOL, is intended to be a generally accepted language for which compilers are being written that will enable the running of a single COBOL program on many different computers. In May, 1959, representatives from the government, computer manufacturers, and industrial computer users met to form the CODASYL committee. In April, 1960, the U. S. Government Printing Office published "COBOL -- A Report to the Conference on Data Systems Languages, Including Initial Specifications for a

Common Business Oriented Language (COBOL) for Programming Electronic Digital Computers".

High-Level Machine Language

The successive development of symbolic assembly routines and compilers demonstrated that one is not limited to the basic machine language. Suitable translating routines permit a variety of input languages to be used (3). These languages, however, although problem oriented, still are somewhat determined, in format, by the machine on which they will be used. As has been indicated, these languages require a programmed translator to make themselves understood by a computer. Such a translator, in the case of FORTRAN, for example, has approximately 65,000 machine instructions (10).

Accommodating to the nature of the computer in this way requires a considerable commitment of compiling and debugging prior to the first feedback of results. Furthermore, it has been found necessary to write compilers for many problem fields, which gave this mode of solution a patchwork look (10).

For these reasons, a group of researchers at the IBM Thomas J. Watson Research Center, among others, decided to examine the problem in terms of changing the fundamental organization of the computer itself. Their goal is to return to machine language operation, but this time to develop a computer with a high-level language, at least as powerful as COBOL or ALGOL, etc., as its machine language. Such a new machine language and organization would eliminate the necessity for compilation and assembly.

In an attempt to prove that a powerful language can be implemented efficiently in hardware and that great cost or great difficulty of use are not necessary consequences of a powerful computer organization, a system called ADAM has been designed. The new language which this system uses, called EVE, is described in Appendix B (10).

PROGRAM-SELECTION DATA PROCESSING SYSTEM

The previous section presented a brief outline of some of the methods which men have utilized to lighten the load of programming drudgery and minimize the probability of human error in providing instructions to machines. It was shown that digital computers evolved in their own technical environment, and to a degree independently of the problem environment. Also, it has been necessary to have computing centers with staffs of programmers as intermediaries between machines and users. As the inadequacy of the arrangement became apparent, problem oriented languages were written, with compiler programs to allow the machines themselves to do the conversion to their own machine language. Recently, a system has been designed to implement a problem oriented language directly (10). These measures have all been undertaken at one time or another to make computers easier and more efficient to use, and thereby to make them available to more users with more problems.

The remaining sections of this dissertation will attempt to illustrate a data processing system which further reduces the user's communication problems and again broadens the scope of computer applicability for a limited class of problems.

Programming With Programs

With the advent of automatic coding languages, the computer operator needed only to write a program, in a language closely resembling his own, and to supply it to the computer for translation. The system to be proposed here will make it unnecessary for the user himself to write programs,

thereby relieving him of the task of learning a programming language or of constructing workable programs. The manner in which this is accomplished may best be explained by referring to Figure 1. At the outset, before a user purchases the computer system, his needs and proposed areas of computer application are analyzed by a computer agency. A list of all the standard programs which he will require over a period of time is drawn up and the actual programs are written by trained programmers. These programs are then stored in the computer memory where they may be called on whenever they are needed. The names of these programs are composed of simple command words, operands, and file names which are listed on rows of selection buttons on the control console. Then, whenever the operator wants to perform a standard routine, such as "COMPUTE WEEKLY PAYROLL", he has only to press the appropriate combination of selection buttons, supply the input data in a predetermined manner, depending on the selected routine, and the computer handles the rest of the details and carries out the actual information processing.

At first glance, it may seem that this type of system places rather severe restrictions upon the operator by limiting him to only those programs which have been stored in the machine in advance. Whether this is a serious limitation, however, depends upon the type of service which is expected from the computer. Obviously, such a system cannot be used in a situation where new programs are required every day, such as in a scientific laboratory or computation center where the same problem may never be solved twice. So unless every possible program could be thought of in advance and stored in the machine, this kind of problem will have to be left to other types of systems.

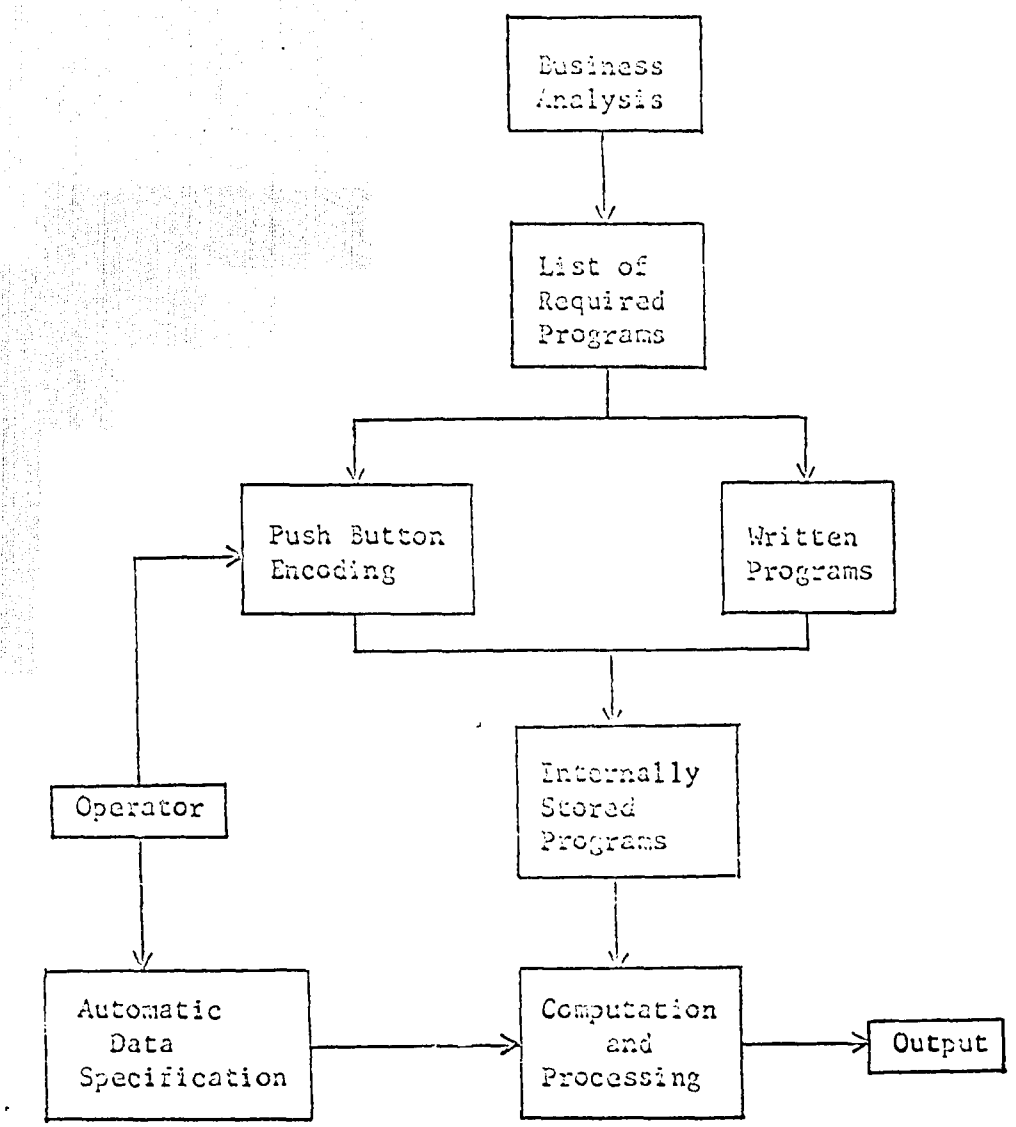

Figure 1. Program-selection data processing system

The number of programs which will be required, therefore, is another factor to consider in examining possible areas of application of the program-selection system. The number of programs must be of small enough number such that the memory capacity of the computer is not exceeded. This places a limitation on the scope of the problems which the system will be able to handle in any given situation.

With these considerations in mind, it appears that the proposed data processing system would be most applicable in a small business environment. In such a situation, programming needs would be relatively constant over a period of time, and therefore could be anticipated in advance. Moreover, it would seem that a fairly small number of programs would handle most of the data processing tasks which any given business might carry out on a routine basis, and that these programs would not be of excessive length.

In order to demonstrate that these optimistic speculations have some basis in fact, and to illustrate in a more concrete manner how the proposed system might be utilized, two typical data processing tasks will be analyzed in detail. They are "typical" of the small business environment previously suggested, in that these tasks would ordinarily require the attention of one to five persons.

Graduate Student Records

One application in which a small computer could be used effectively is in processing records of graduate students in a particular department of a university. Information regarding each student in the department might be contained in a series of files which contain records such as those in Figures 2, 3, and 4. One report which is required on a regular basis is

the Quality-Point Report shown in Figure 5, which indicates the number of credits which a student has accumulated and his grade average to date. A list of some of the other processing operations which could be carried out on the basis of the information contained in these files is presented in Figure 6.

It can be seen that the number of operations is quite small. A program could be written for each of the sixteen operations and easily stored in the memory of a computer. The structure of the program names suggests that these names can be conveniently encoded on groups of selection buttons for automatic reference by the computer operator. This has been done in Figure 7. The left column contains standard command words, the center column two operand names, and the right column the file names. The two prepositions serve to indicate whether information is being put into or taken from a file. With this fairly small number of selection buttons, the operator has the capability of selecting any one of the sixteen internally stored routines to perform his particular data processing problems.

In a situation such as that described in this example, it is likely that the file structure and type of processing conducted upon these files will not change much from one quarter to the next. Hence, after the original analysis, program writing, and modifications have been made, it seems reasonable to assume that a program selection system would adequately meet the needs of student record processing as described above.

Name _____ Date of Birth _____ Date Entered _____

Highest Degree Held _____ Institution _____

Degree Working On _____ Research Area _____

Completion Dates:

English _____ Master's Final Exam _____

French _____ Preliminary Exam _____

German _____ Doctoral Final Exam _____

Russian _____ Degree Granted _____

Other _____

Figure 2. Record from graduate student file

Name	Quarter	Year
Course	Credit	Grade

Figure 3. Record from transcript file

NAME _____

M. S. Committee:

MAJOR _____ Professor _____

MINOR _____ Professor _____

Professor _____

Professor _____

Professor _____

Ph. D. Committee

MAJOR _____ Professor _____

FIRST MINOR _____ Professor _____

SECOND MINOR _____ Professor _____

Professor _____

Professor _____

Figure 4. Record from committee file

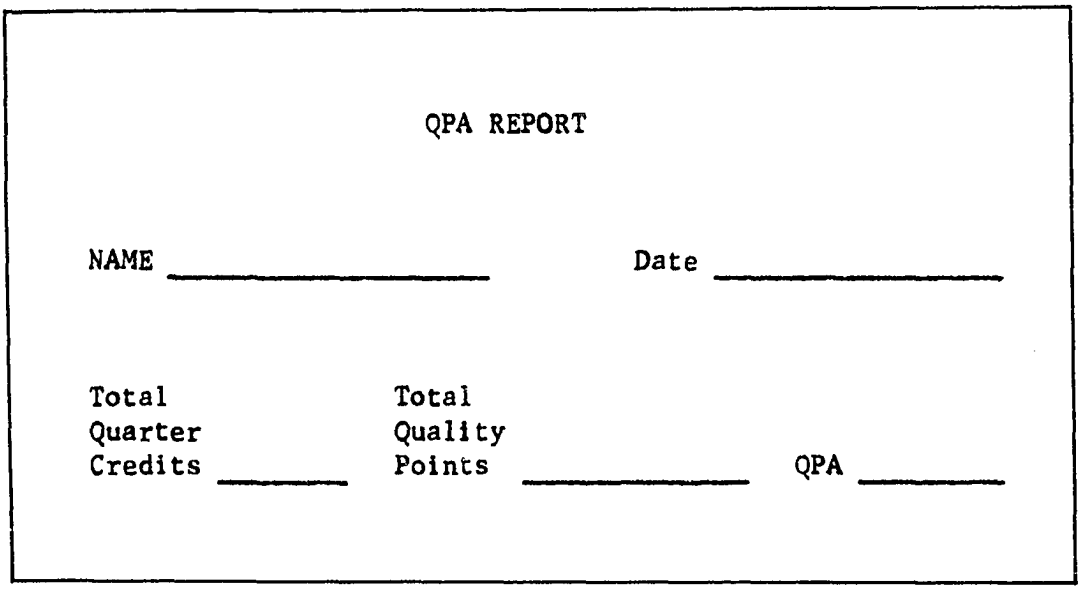

Figure 5. Record format of quality point average report

1. OUTPUT GRADUATE STUDENT FILE
2. OUTPUT COMMITTEE FILE
3. OUTPUT TRANSCRIPT FILE
4. OUTPUT RECORD FROM GRADUATE STUDENT FILE
5. OUTPUT RECORD FROM COMMITTEE FILE
6. OUTPUT RECORD FROM TRANSCRIPT FILE
7. OUTPUT ITEM FROM GRADUATE STUDENT FILE
8. OUTPUT ITEM FROM COMMITTEE FILE
9. OUTPUT ITEM FROM TRANSCRIPT FILE
10. ADD RECORD TO GRADUATE STUDENT FILE
11. ADD RECORD TO COMMITTEE FILE
12. ADD RECORD TO TRANSCRIPT FILE
13. DELETE RECORD FROM GRADUATE STUDENT FILE
14. DELETE RECORD FROM COMMITTEE FILE
15. DELETE RECORD FROM TRANSCRIPT FILE
16. COMPUTE QUALITY POINT AVERAGE REPORT

Figure 6. Processing operations for graduate student files

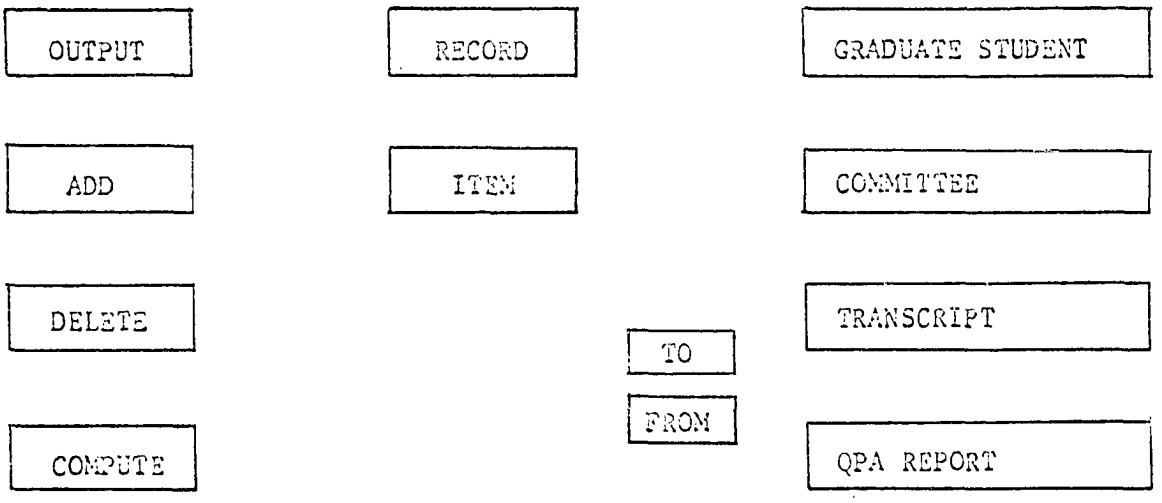

Figure 7. Program selection buttons for graduate student case

Wholesale News Agency

A somewhat more elaborate use of the program-selection data processor will now be typified by applying it to the problems encountered in the operation of a wholesale news agency. Such an agency receives shipments of magazines from various publishing companies and distributes them to retail dealers throughout a city for sale to the public. The wholesale agency under consideration services more than 50 dealers in a town with a population of 30,000 people.

Briefly, the major activities associated with the conduction of the agency's business for a one week period of time will be described as follows. Magazines are shipped by truck, and various shipments are received every day. As the magazines come in, their arrival is noted in a general receiving book. Twice each week, an "order" is tied and delivered. This means that each retail dealer is supplied with a certain number of each magazine included in the day's order which he wishes to sell, the quantity of each title being determined by his customers' interests, his volume of business, etc. Any single order usually contains about 50 different titles, most of which differ from those of the previous order, since monthly magazines can be distributed only once each month and weeklies, only once a week, etc. Over a period of time, nearly 400 different magazines will be circulated for sale.

The first step in making up an order consists of filling out a distribution sheet. On this form, the manager indicates the quantity of each title which each dealer is to receive, checking with the receiving book to make sure that every title on the order has been received. From the

information on the distribution sheet, an invoice is made out for each dealer, which lists the number of each magazine he will receive and the total charge for his order. The charge for each dealer is entered in the accounts receivable file, and the invoice serves as the dealer's bill. The magazines indicated on the invoices are then tied into bundles and delivered to the respective dealers throughout the city.

Once a week, a representative from the news agency visits each dealer to complete a check-up study. At this time, he will pick up out-of-date magazines which have not been sold and give the dealer credit for them. A publisher may have a particular interest in the sales progress of a new issue, in which case the agent will note the number of copies of the magazine still on the dealer's rack for comparison with the number of copies that dealer originally received and with the results of other dealers. If sales are better in one area, some of the issues may be taken from areas of slow sales to the newsstands having the higher demand.

Upon his return to the office, the agency man will record the credit extended to various dealers in the accounts receivable file and perhaps analyze the percentage of sales of the magazines he has checked on the newsstands.

These are a few of the routine responsibilities and duties of a wholesale news agency. The procedure changes very little throughout the year, and much of the work is mechanical and time-consuming. The agency employs three people on a full-time basis, and two part-time assistants.

Here, then, is a situation in which the application of an inexpensive, easy-to-operate data processing system would be quite useful. As in the previous example, the pertinent information would be contained in files

such as those of Figure 8. A few of the records from two of these files are presented in Figures 9 and 10. The standard operations, numbering about 80, which would be performed by a computer using the information in these files, are listed in Figure 11. The 80 routines may be selected by pressing a combination of the coded buttons in Figure 12.

System Feasibility

The two cases just discussed answer a few of the questions which were raised regarding the possible limitations of a fixed-program data processing system. The cases illustrate by example the fact that businesses or offices having unchanging modes of operation do exist, and that these operations can be carried out by a collection of programs which can conveniently be stored in a computer. It is easy to imagine many other instances with similar characteristics, such as a drug store, lumber yard, etc., where the same principles would apply.

The examples further demonstrate the manner in which one of the stored programs could be called into service by pressing a combination of a smaller number of program-selection buttons.

Thus it has been shown that a data processing system which has all its routines stored in advance, available to the user by means of encoded selection buttons, is indeed feasible from a business point of view. The next two sections will consider the organization of such a system from an engineering standpoint. The length of some of the programs mentioned in the two sample cases will be examined and an appropriate language chosen for use in writing these programs.

Master File -- Contains approximately 400 records consisting of magazine titles, the dealer price of each, the year-to-date sales of each, and the publisher of each.

Dealer File -- Lists the number, name, and address of each dealer serviced.

Distribution File -- Input file describing the quantity and title of the magazines which each dealer will receive in a given order.

Check-Up File -- Records the number of copies of each issue of any given title returned by the dealers and the resulting percentage of sales for that issue.

Dealer Invoice File -- Output file presenting, by dealer, the titles and quantities of magazines of each price group and the total price of the order.

Payroll File -- Records the salary of each employee and the number of hours each has worked during a particular pay period.

Accounts Receivable File -- Amount due from each dealer.

Accounts Payable File -- Amount owed to each publisher.

Operating Expenses File -- Records the building expenses, truck gasoline and service, and other miscellaneous expenses.

Figure 8. Description of wholesale news agency files

TITLE	PRICE	YTD-SALES	PUBLISHER
Argosy	0.50	62.85	Popular
Cosmopolitan	0.35	105.15	International
Family Circle	0.15	475.30	Independent
Field & Stream	0.35	34.05	Curtis
Fur, Fish, & Game	0.25	15.85	Capital
Good Housekeeping	0.50	135.70	International
House Beautiful	0.60	215.40	International
Ladies Home Journal	0.35	165.90	Curtis
Look	0.25	220.25	Curtis
Modern Screen	0.35	128.30	Dell
Movie Star Parade	0.25	145.70	PDC
Outdoor Life	0.35	64.35	Select
Popular Photography	0.50	27.40	Triangle
Reader's Digest	0.35	835.15	Select
Redbook	0.35	143.80	Select
Saturday Evening Post	0.20	472.00	Curtis
Seventeen	0.50	79.10	MLA
Time	0.35	35.60	Select
Woman's Day	0.15	678.20	Fawcett

Figure 9. Sample records from master file

DEALER NUMBER	DEALER NAME	ADDRESS
1	Bill's Newsstand	201 6th
2	Hogan Drug	426 Central Ave.
3	Super Value	1700 2nd Ave. N.
4	Safeway	320 2nd Ave. S.
5	Kresges	550 Central Ave.
6	Warden Cigar Shop	104 9th St.
7	Central Drug	915 Central Ave.
8	Rexall Drug	406 Elm St.
9	Donavan News	1105 1st Ave. N.
10	Jones Grocery	1826 S. 18th St.

Figure 10. Illustrative records from the dealer file

INPUT (Any File)

OUTPUT (Any File)

OUTPUT ITEM FROM (Any File)

OUTPUT RECORD FROM (Any File)

ADD RECORD TO (Any File)

DELETE RECORD FROM (Any File)

ADD ITEM TO (Any File)

DELETE ITEM FROM (Any File)

COMPUTE (Any File)

Figure 11. Processing operations for wholesale news agency

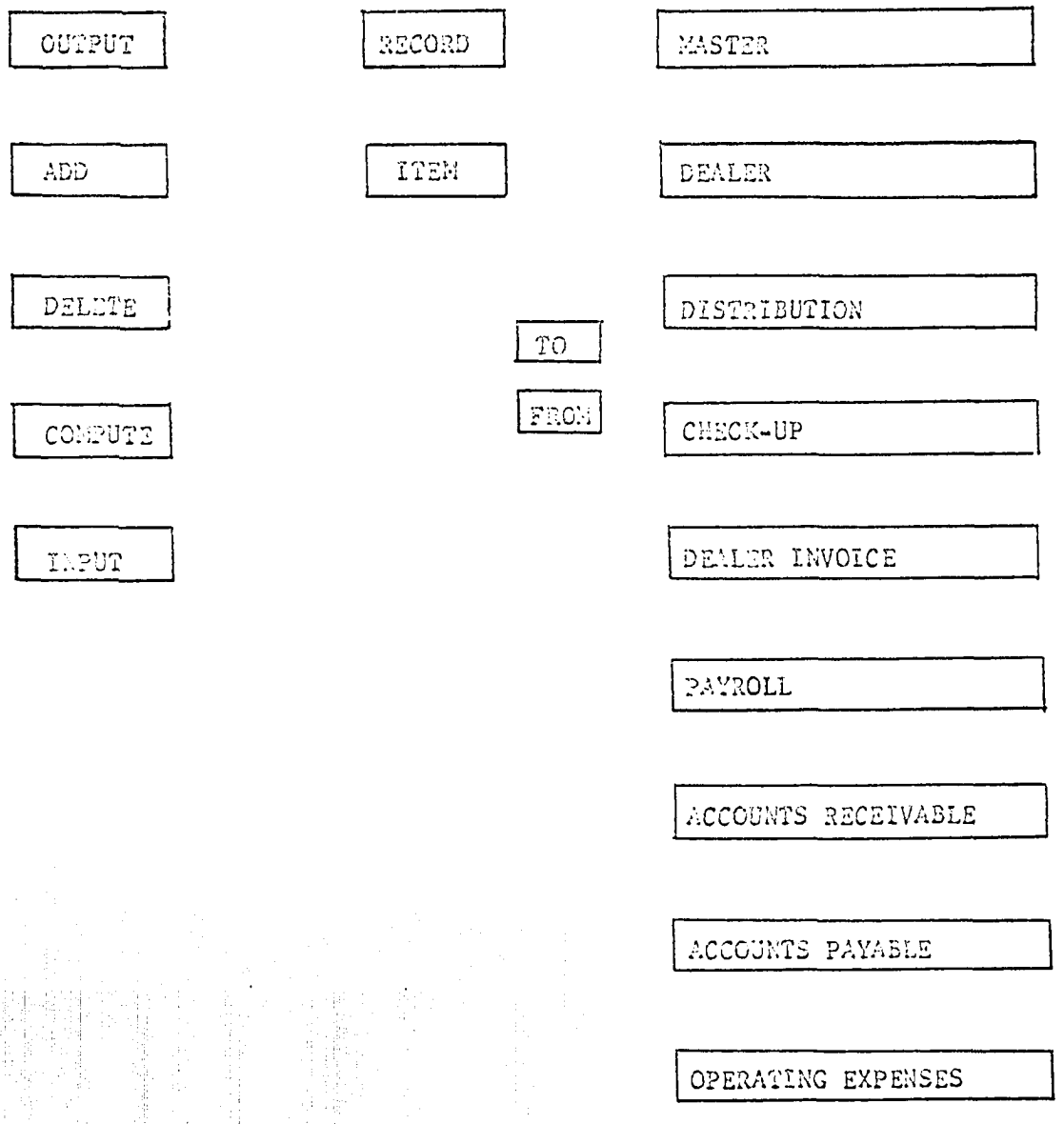

Figure 12. Program selection buttons for wholesale news agency case

SYSTEM DESIGN CONSIDERATIONS

Language Criteria

There are three factors which must be considered in choosing a programming language for the fixed-program data processor:

1. Length of programs written in the language
2. Ease of writing or changing the program
3. Ease of implementing the language

Since all the programs are to be stored in the computer memory, it is obvious that a language which describes a sequence of operations with the least number of characters is the most desirable. If the programs are as short as possible, more room is available in the memory for additional programs, allowing the system to be utilized in more complex assignments.

The ease of writing the programs originally is also important. The data processor has been proposed for usage by a small business manager who wants the most inexpensive system possible. His original investment will be less if the time required to write the necessary programs is minimized. An easy-to-use language also reduces the time which would be needed to revise and up-date the programs at a later time, again keeping the costs down.

Finally, the implementation of the language must be considered. The most simple and direct programming language might require the most complex hardware to use the resulting programs. This would greatly increase the cost of the system, and thus a compromise would have to be made between the optimum language and the most economical hardware design.

The COBOL Language

The most logical choice of a language which would satisfy the above requirements is COBOL, since it was designed specifically for use in business applications. It closely resembles English, so one would expect COBOL programs to be fairly easy to write. Therefore, two COBOL programs will be written to illustrate this and also to determine how long the COBOL routines will be. The reader is referred to Appendix A for details concerning the COBOL language.

One routine listed in both sample cases was that for adding a record to a file. As an example of this type of routine, a COBOL program will be written which will add a record to the Master File in the news agency case.

As shown in Figure 9, the Master File contains about 400 magazine records, indexed by title, consisting of information about each magazine. Adding a record to this file consists of inserting a new title at its proper alphabetical location in the file. In order to do this, the titles to be added must be supplied by an input file, compared with the titles in the old Master File, and a new Master File written which includes the new titles in their proper location.

Each title with its corresponding information in each file is called a record of that file, and is referred to in the program by name. The name of a record in the old Master File is called OM for Old Master; that of the new Master File, NM; and that of the input file, IN-REC.

A flow diagram illustrating the instructions and decisions involved in inserting new records into the old Master File is shown in Figure 13. The procedure division of the COBOL program, written on the basis of the flow diagram, is presented in Figure 14. It contains about 600 characters.

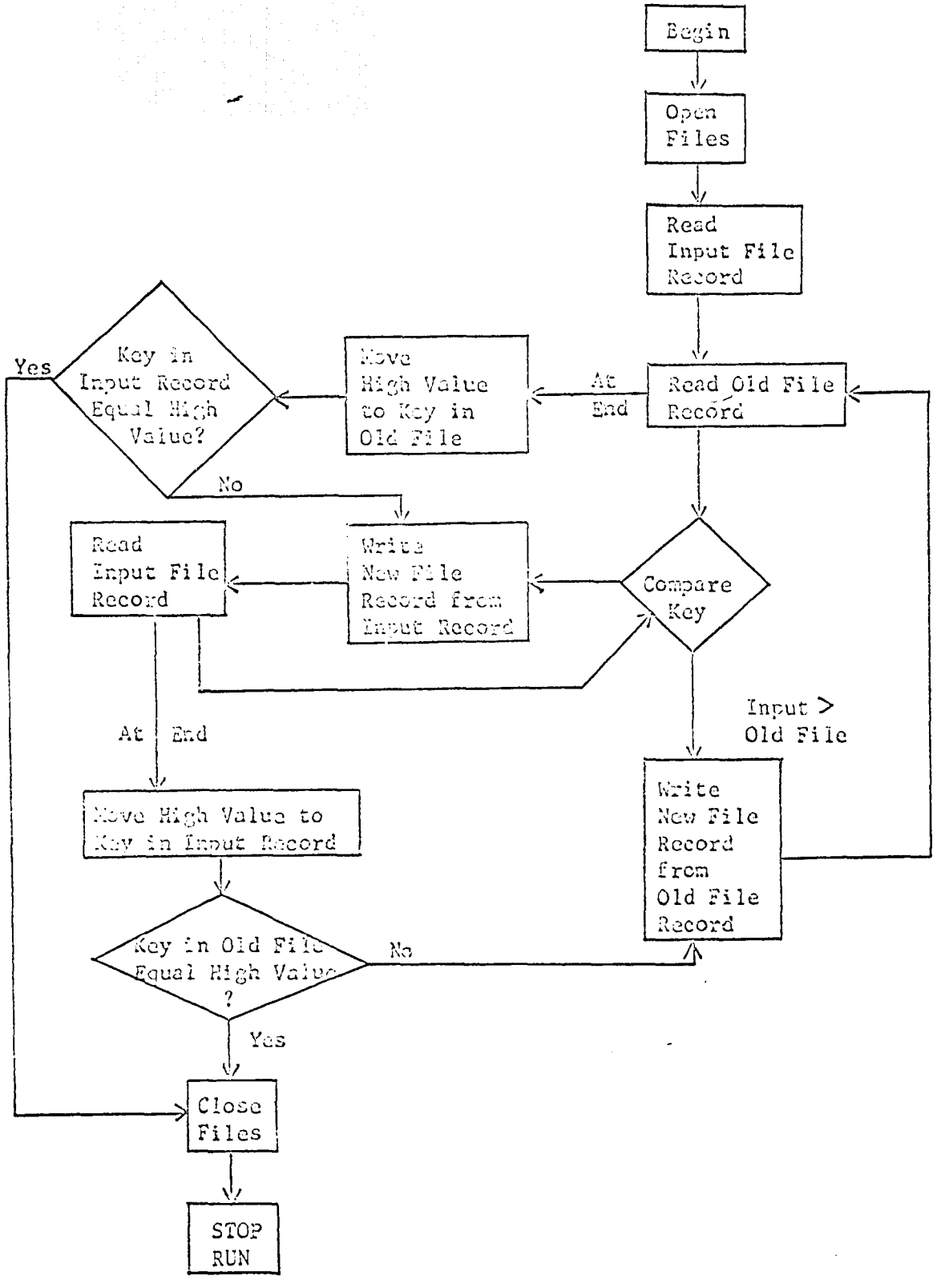

Figure 13. Flow diagram for adding a record to a file

```
PROCEDURE DIVISION.

BEGIN. OPEN INPUT INPUT-FILE, OLD-MASTER-FILE, OUTPUT NEW-
     MASTER-FILE.  READ INPUT-FILE RECORD.

OLD-FILE-READING.  READ OLD-MASTER-FILE RECORD; AT END GO
     TO END-OF-FILE.

KEY-COMPARISON. IF TITLE IN OLD-MASTER-FILE IS NOT LESS THAN
     TITLE IN IN-REC GO TO RECORD-ADDITION.

COPY-OLD-RECORD. WRITE NM FROM OM. GO TO OLD-FILE READING.

RECORD-ADDITION. WRITE NM FROM IN-REC. READ INPUT-FILE
     RECORD; AT END MOVE HIGH-VALUE TO TITLE IN IN-REC AND
     GO TO OLD-FILE-END-CHECK. GO TO KEY-COMPARISON.

END-OF-FILE. MOVE HIGH-VALUE TO TITLE IN OM. IF TITLE IN
     IN-REC IS EQUAL TO HIGH-VALUE GO TO WRAPUP. GO TO
     RECORD-ADDITION.

OLD-FILE-END-CHECK. IF TITLE IN OM IS NOT EQUAL TO HIGH-
     VALUE GO TO COPY-OLD-RECORD.

WRAPUP. CLOSE OLD-MASTER-FILE, INPUT-FILE, NEW-MASTER-FILE.
     DISPLAY 'JOB FINISHED'. STOP RUN.
```

Figure 14. COBOL procedure division for adding a record to a file

This is a general program for adding a record to a file. If the file name happened to be changed, very little program modification would be needed. Thus, this particular routine would probably be one of the standard routines used by all fixed-program data processors regardless of the particular collection of files involved, thereby reducing the completely new programming effort needed to set up an operating system for each individual business.

A program with some computation involved is the routine for computing the grade point average and preparing the report of Figure 5. The information needed to compile this report is obtained from the Transcript File of each student. In order to keep track of the succeeding steps in the computation, the computer must be able to distinguish between each course name and its respective credit value and grade. This is done by supplying each space with a different symbolic name, as shown in Figure 15. COURSE-3, for example, is a distinctive name which refers to the name of the course contained in the data record, TRANS, at this particular location.

Again, a flow chart, Figure 16, has been prepared to show more clearly the steps of the processing which are carried out by the COBOL procedure division of Figure 17. In the course of the computation, the computer makes use of various temporary working storage locations which are defined in the data division of the complete COBOL program. This procedure division contains about 1200 characters.

Some of the programs listed in the graduate student and news agency cases would probably be longer and more complex, while some would be shorter than the two described above, but their average length is

TRANS

NAME	QUARTER	YEAR
COURSE	CREDIT	GRADE
COURSE-1	CREDIT-1	GRADE-1
COURSE-2	CREDIT-2	GRADE-2
COURSE-3	CREDIT-3	GRADE-3
COURSE-4	CREDIT-4	GRADE-4
COURSE-5	CREDIT-5	GRADE-5

Figure 15. Transcript file record format

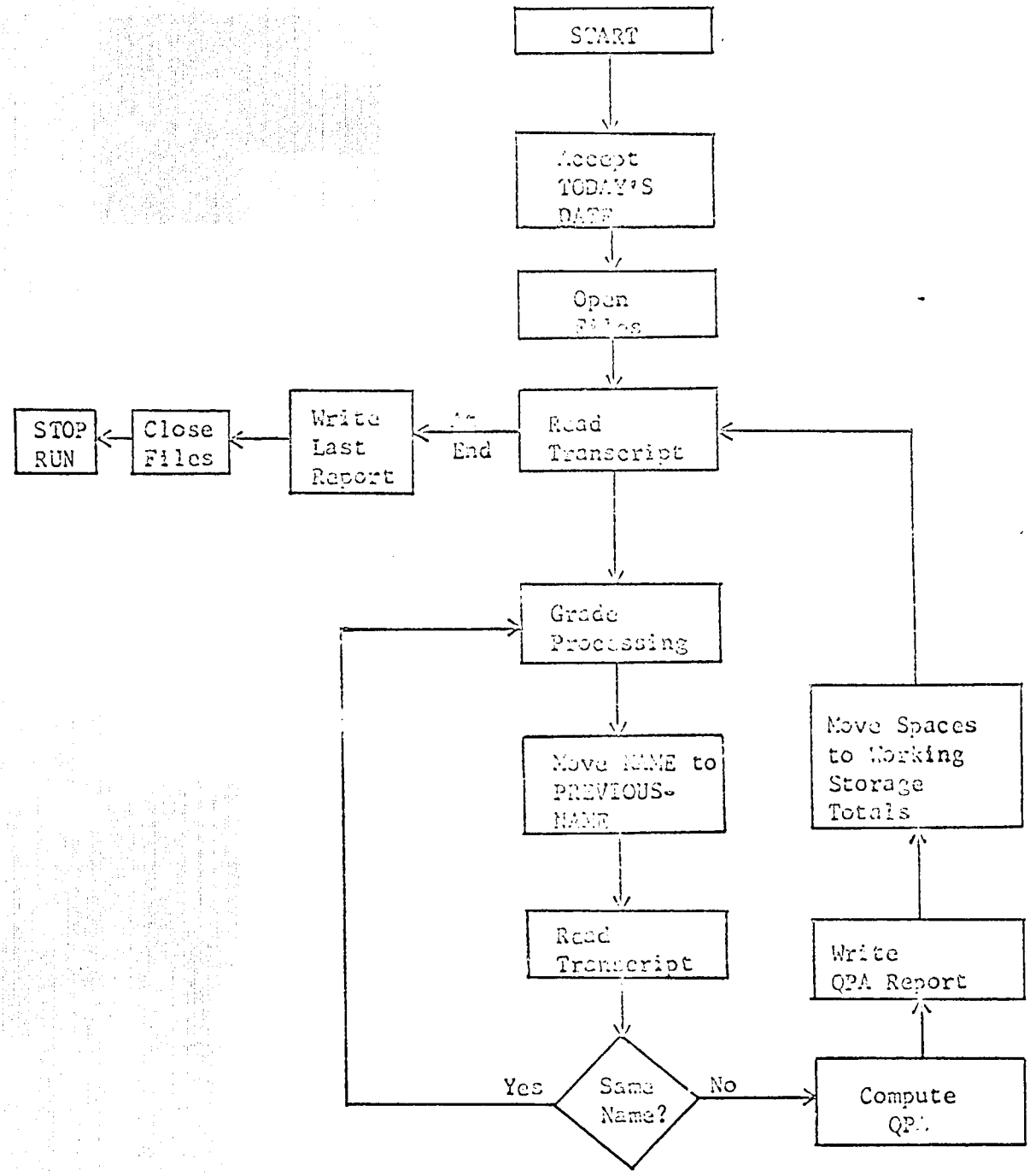

Figure 16. Flow diagram for quality point average computation

```
PROCEDURE DIVISION.

BEGIN. ACCEPT TODAYS-DATE. OPEN INPUT TRANSCRIPT, OUTPUT QPA-
     REPORT. MOVE ZEROS TO CREDIT-SUM. MOVE ZEROS TO QUALITY-
     POINT-SUM.

TRANSCRIPT-READING. READ TRANSCRIPT RECORD; AT END GO TO WRAPUP.

GRADE-PROCESSING. ADD CREDIT-1 TO CREDIT-SUM. MULTIPLY CREDIT-1
     BY GRADE-1 GIVING POINTS. ADD POINTS TO QUALITY-POINT-SUM.
     ADD CREDIT-2 TO CREDIT-SUM. MULTIPLY CREDIT-2 BY GRADE-2
     GIVING POINTS. ADD POINTS TO QUALITY-POINT-SUM.
     ADD CREDIT-3 TO CREDIT-SUM. MULTIPLY CREDIT-3 BY GRADE-3
     GIVING POINTS. ADD POINTS TO QUALITY-POINT-SUM.
     ADD CREDIT-4 TO CREDIT-SUM. MULTIPLY CREDIT-4 BY GRADE-4
     GIVING POINTS. ADD POINTS TO QUALITY-POINT-SUM.
     ADD CREDIT-5 TO CREDIT-SUM. MULTIPLY CREDIT-5 BY GRADE-5
     GIVING POINTS. ADD POINTS TO QUALITY-POINT-SUM.
     MOVE NAME TO PREVIOUS-NAME. READ TRANSCRIPT RECORD. IF NAME
     IN TRANS IS EQUAL TO PREVIOUS-NAME GO TO GRADE-PROCESSING.

QPA-COMPUTATION. DIVIDE QUALITY-POINT-SUM INTO CREDIT-SUM GIVING
     QPA. MOVE PREVIOUS-NAME TO NAME IN QPA-REPORT. MOVE TODAYS-
     DATE TO DATE IN QPA-REPORT. MOVE CREDIT-SUM TO TOTAL-QUARTER-
     CREDITS. MOVE QUALITY-POINT-SUM TO TOTAL-QUALITY-POINTS.
     WRITE QPA-REPORT. MOVE ZEROS TO CREDIT-SUM. MOVE ZEROS TO
     QUALITY-POINT-SUM. GO TO TRANSCRIPT-READING.

WRAPUP. CLOSE TRANSCRIPT, QPA-REPORT. DISPLAY 'JOB FINISHED'.
     STOP RUN.
```

Figure 17. COBOL procedure division for quality point average program

certainly not prohibitive and many of these programs could be stored in a computer memory at one time. Thus, COBOL satisfies the first two requirements of an appropriate language for use in the proposed data processing system. It has been pointed out previously, however, that COBOL requires an extensive compiler for translating the written program into machine language. This leads to a complex and expensive system in terms of the hardware needed to operate with the COBOL language. Although the use of COBOL appears appropriate on the basis of the first two specifications, it is possible to find a language which is easier to implement and which is also acceptable in the same respects as COBOL. Such a language is EVE, described in detail in Appendix B.

The EVE Language

As was indicated in a previous section of this paper, EVE is a problem oriented language which can be directly implemented without the need of an intermediate compiler. For comparison with the COBOL programs, the routine for computing a student's grade point average will be written in the EVE language.

The EVE language uses identifier numbers to indicate the structure of the data contained in a file. Using this type of notation, the Transcript File will appear as in Figure 18. The program will use the information contained in this file throughout the course of its computation. The program itself is presented in Figure 19. This program contains about 450 characters, as compared to 1200 for the equivalent COBOL program.

It should be pointed out that the EVE program contains instructions which specify the format of the output data. In the COBOL program, this

must be accomplished by the data division. Consequently, the EVE program actually performs an extra service, if a comparison is to be made of the procedure portions of the programs in each language. Thus it appears that an EVE program will be from one-half to one-fourth as long as an equivalent COBOL program.

The EVE language employs symbols more than does COBOL, and may present a few more difficulties to the new learner. However, once he becomes familiar with these techniques, he will find that EVE programs are easier to write than the COBOL programs, partly because of the more convenient manner in which data organization is specified by the EVE language.

The EVE language, then, satisfies all three criteria better than COBOL. It is easier to use; it results in shorter programs; and it can be directly implemented. For these reasons, it is recommended that EVE, or a similar language, be used as the operating language in a fixed-program data processing system.

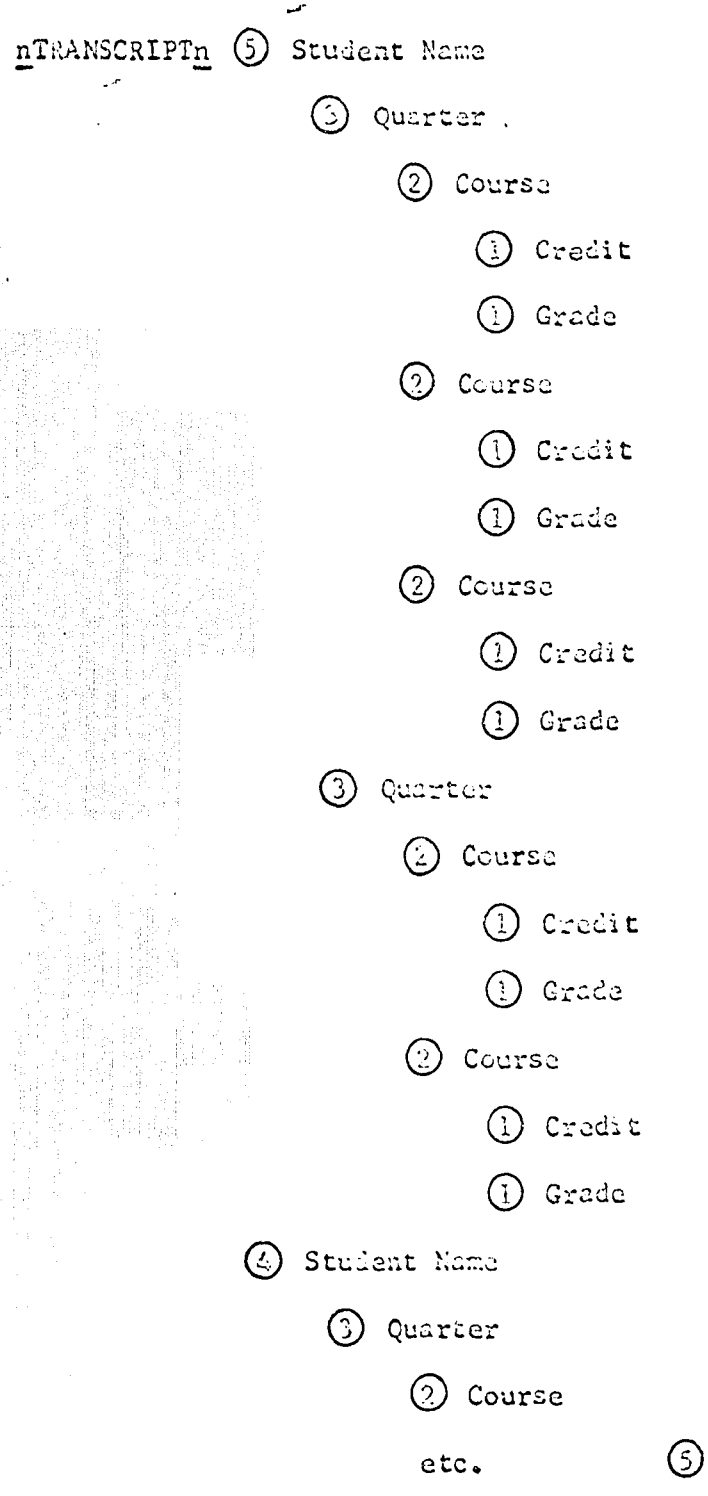

Figure 18. Transcript file format in the EVE language

```
vSTARTv        ③ unit "26" ⟶ Record
               ③ "0" ⟶ Totalhours ② "0" ⟶ Totalcredits
               ③ "2" ⟶ I ② "2" ⟶ J
               ③ "NAMEbbb...bbbTOTAL CREDIT HOURSbbb...bbbTOTAL QUALITY
                 POINTSbbb...bbbQPAbb" ⟶ unit "13" line ↑$ 1
vCOMPUTEv      ③ Record③↑ I ②↑ J ①↑ 2 ⟶ CH
               ③ Record③↑ I ②↑ J ①↑ 3 ⟶ LC
               ③ if LG = "A" ② CH* "4" ⟶ QP
               ③ if LG = "B" ② CH* "3" ⟶ QP
               ③ if LG = "C" ② CH* "2" ⟶ QP
               ③ if LG = "D" ② CH* "1" ⟶ QP
               ③ if LG = "F" ② CH* "0" ⟶ QP
               ③ CH + Totalhours ⟶ Totalhours
               ③ QP + Totalcredits ⟶ Totalcredits
               ③ if Record ③↑ I ②↑ J ①↑ 3 reach ③ ②┌ NEXTQUARTER
               ③ J + "1" ⟶ J ②┌ COMPUTE
vNEXTQUARTERv  ③ if Record ②↑ I ②↑ J ①↑ 3 reach ④ ②┌ PRINT
               ③ "2" ⟶ J ② I + "1" ⟶ I
               ③ ┌ COMPUTE
vPRINTv        ③ Record ③↑$ 0 edit "PPP...PPPbb" ⟶ LINE ②↑ 1 ①↑ 1
               ③ Totalhours edit "M03PPP." ⟶ LINE ①↑ 2
               ③ Totalcredits edit "M04PPPP." ⟶ LINE ①↑ 3
```

Figure 19. EVE program for calculating grade point average

③ Totalcredits/Totalhours edit "M01P.ZZ" → LINE ① ↑4

③ LINE → unit "13" line ⇳ 1

③ if Record reach ⑤ ② stop

③ delete LINE

③ ⌜ START ③

Figure 19 (Continued)

CONSOLE OPERATION AND DATA ENTRY

Previous sections of this dissertation have illustrated the manner in which a number of programs can be written in a problem oriented machine language, stored in a computer memory, and selected by the operator by means of program-selection buttons or something equivalent. The remaining problem to be discussed is that of supplying the required input data to the system.

Description

In the matter of data entry, as in program selection, the goal is to provide a method of operation which anyone can accomplish without special training. For this reason, a typewriter with an ordinary keyboard could be used as the input-output device. The entire console could consist of this typewriter keyboard, a few additional command switches as shown in Figure 20, and the program-selection buttons such as those of Figure 7 or Figure 12. The operation of the data processing system will be described with reference to these figures.

When the operator wants the system to perform a certain routine, he presses his selection on the program-selection panel. If he presses the wrong buttons, or decides to change his selection, he may clear the selection panel with the RESET button. When his selection has been made, the operator presses the BEGIN button to start the computer operation. The computer follows the instructions in the selected program until it comes to a point at which input data is required. The machine then instructs the typewriter to type out a message which asks the operator for the appropriate information. The operator responds by typing the information as the

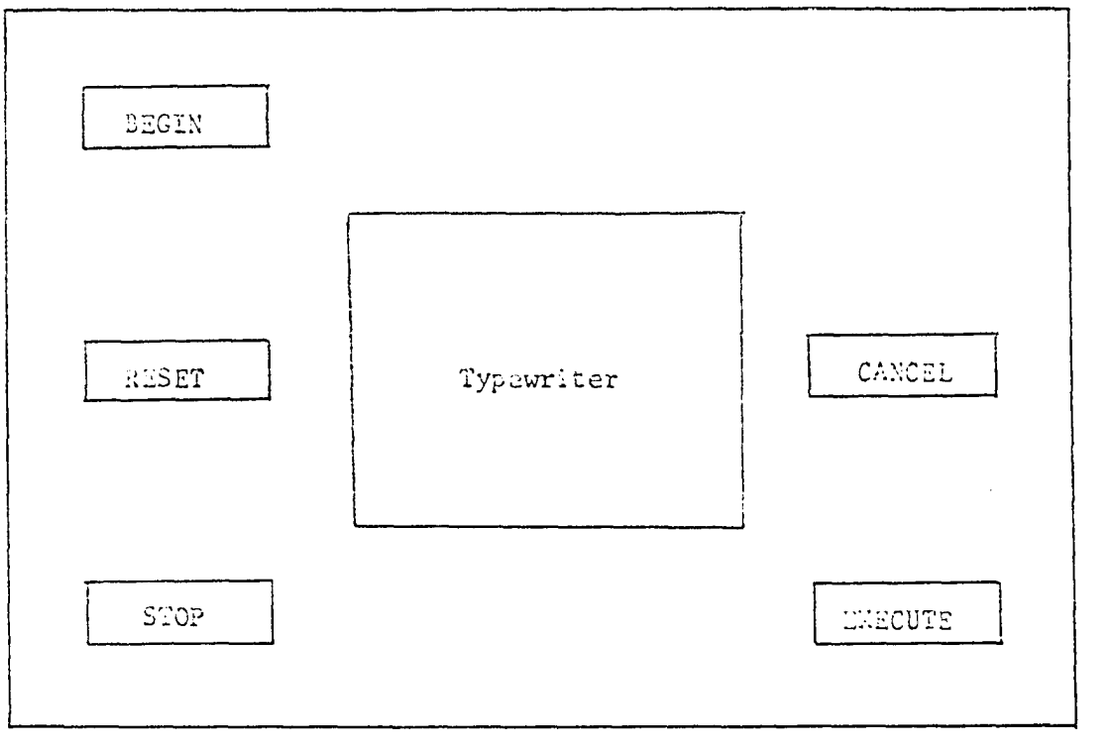

Figure 20. Console layout

answer to the machine's question. The characters he types are kept in a temporary storage memory until all the information is typed correctly. If he makes a typing error, the operator may press the CANCEL button (Figure 20) and the question will be asked again, allowing him to re-type his answer. When all the input data has been supplied correctly to temporary storage, the operator presses the EXECUTE button. This enters the data from temporary storage into the main computer memory for use in processing by further instructions in the program being run. This same procedure is followed at any time during the running of a particular program when input data is required. Each time, the operator is told what information he must furnish and just how to furnish it. Since the data is supplied to the computer in the form of a response to a statement or question of the machine, the operator does not have to worry about the data format or level indication. The description and format of each file and record is stored in advance in the computer memory in a form such as that of Figure 18. This format description, together with the questions asked by the machine at various points in a program execution, assures that the operator automatically supplies information in a form which the computer can recognize and use properly.

Example

A simple example will illustrate the method of supplying input data described above. Suppose the operator of the wholesale news agency system wants to add another magazine and associated information to the magazine master file. The description of the format of the records in the master

file has been previously stored in the computer memory in the following form:

nMASTERn ③ TITLE

① PRICE

① YTD-SALES

① PUBLISHER

② TITLE

① PRICE

etc.

The noun "RECORD" has also been named to describe one of these records and has the following structure:

nRECORDn ② TITLE

① PRICE

① YTD-SALES

① PUBLISHER ②

Now when the operator wants to add a record to the master file, he will first press selection buttons "ADD RECORD TO MASTER FILE" to select the proper internally stored routine. When the selection has been made, the operator then presses the BEGIN switch and the computer begins to follow the instructions of the selected program, a portion of which is reproduced in Figure 21.

The first thing this part of the program does is to have the typewriter print out the uncompleted statement "TITLE = ", since the identifying key name in a master file record is a magazine title. The operator then types the title of the magazine to be added after the equal sign:

TITLE = NEWSWEEK

```
③ "TITLE = " ⟶ unit 1
   (Typing and Carriage Return)
③ unit 1 ⟶ Temporary Title
③ surround Temporary Title with "①"
③ "PRICE = " ⟶ unit 1
   (Typing and Carriage Return)
③ unit 1 ⟶ Temporary Price
③ surround Temporary Price with "①"
③ "YTD-SALES = " ⟶ unit 1
   (Typing and Carriage Return)
③ unit 1 ⟶ Temporary Sales
③ surround Temporary Sales with "①"
③ "PUBLISHER = " ⟶ unit 1
   (Typing and Carriage Return)
③ unit 1 ⟶ Temporary Publisher
③ surround Temporary Publisher with "①"
   (EXECUTE command)
③ Temporary Title ⟶ RECORD ①↑ 1
③ Temporary Price ⟶ RECORD ①↑ 2
③ Temporary Sales ⟶ RECORD ①↑ 3
③ Temporary Publisher ⟶ RECORD ①↑ 4
```

Figure 21. Input-control portion of an EVE program

When the operator has finished typing the magazine title name, he punches the typewriter carriage return button, or some such indicating device. This acts as a signal to the computer to proceed with the next instructions, which ask about the price of the magazine:

```
TITLE = NEWSWEEK
PRICE =
```

The operator types the correct price value and returns the carriage. This process continues until all the questions relating to a master file record have been asked by the computer and answered by the operator. The typed information would then appear as follows:

```
TITLE = NEWSWEEK
PRICE = 0.30
YTD-SALES = 0.00
PUBLISHER = CURTIS
```

At any time during his typing, the operator can read what he has typed. If he has made an error, he can press the CANCEL command to remove what he has typed from the temporary storage location and have the question repeated. As an alternative, a back space and strike over might be used for corrections.

As the data which the operator has typed is entered into the appropriate temporary storage location, it is automatically supplied with the proper level indicators. This is accomplished by the surround verb defined as follows. The instruction surround TEMPORARY with "②" will surround whatever characters that happen to be in the location called TEMPORARY with the ② level indicator.

If the computer fails to ask a question when the carriage has been returned, it means that all the required questions have been asked for the moment. An alternative procedure might be to have the typewriter indicate

that no more immediate input is required. The operator will then press the EXECUTE switch to instruct the computer to continue following the instructions in the program, making use of the data which has just been entered into the computer memory. In the example just discussed, the location called RECORD now contains the new data and appears as follows:

nRECORDn ② NEWSWEEK
① 0.30
① 0.00
① CURTIS ②

Succeeding instructions in the program may direct the computer to search the master file and insert this new record in the proper alphabetical location. Or the data could be inserted in the existing string of data comprising the master file without having to change the location of any of the symbols in the string by means of some type of linking mechanism. This link would connect the insert to the desired point in the string. The linking mechanism is under control of the program through the insert instruction.

It is assumed that this insert instruction is able to make provision for handling the level indicators at the end of a file. If the end-of-file indicator is a ④ , for example, the implied indicators ①, ②, and ③, are also physically present with the ④ in the computer memory. The sequence of the ending characters in the file would thus appear as:

③ Record ① ② ③ ④

Now if a new record, of level ③, is to be inserted at the end of the file, the insert instruction would essentially "move" the ① ② ③ ④ indicator series to the right, leaving space for the record being added.

The result would be as follows:

③ Record ③ New Record ①②③④

The above technique of providing input data may be used by all of the internally stored routines. Whenever a routine needs information from the operator, it tells the operator which information is needed and how it is to be given. The operator has only to "fill in the blanks". He does not have to be concerned with the structure or level of the particular data which he is providing to the computer.

SUMMARY AND CONCLUSIONS

A data processing system having certain functional properties has been proposed in this dissertation. These properties are:

1. The system can be operated by an inexperienced person
2. The system contains a fixed set of stored programs suited to its particular application
3. Program names are encoded and a program selected by the operator by means of a push-button scheme or an equivalent technique
4. The proposed system uses a high-level machine language
5. The operator is included in the input loop and forced to automatically supply the correct level designation of the input data
6. The input-output device is an ordinary typewriter

An inexperienced person can operate the proposed system because the knowledge of computer programming as a prerequisite to system operation has been removed. This has been accomplished in two ways. In the first place, all the necessary programs for application of the system to a particular set of problems are written in advance, by someone trained in business analysis and programming, and stored in the computer memory. Here a program may be selected at random by the operator by means of a push-button scheme or some equivalent selection technique.

Secondly, the operator has been included in the input loop and is automatically forced to supply the input data in a form acceptable to the system. This technique was accomplished by adding a "surround" verb to the EVE language to make provision for automatic level designation of data in the programs themselves.

A high-level problem oriented machine language, known as EVE, was found to be better suited to this type of system than an automatic coding language such as COBOL. In comparing a few sample programs in each language, it was found that EVE and COBOL both provide easy-to-use, efficient programming vehicles for business problems. The EVE language, however, had the additional desirable property of not requiring a compiler. Programs written in the EVE language can be implemented directly in hardware. This makes it possible to change programs in the field without having to re-compile the modified programs. It was also found that EVE could easily handle the automatic level designation of input data by the addition of a new verb as mentioned above.

The development and implementation of these new system concepts are discussed in detail in various sections of this dissertation. It has also been shown, by two illustrative examples, that there are certain classes of problems which are particularly suited to the application of these data processing techniques.

This research indicates that a system operating under the constraints of the concepts and objectives described above is possible from the viewpoint of the user, and realizable in terms of existing software and hardware technology.

LITERATURE CITED

1. Chapin, N. Programming Computers for Business Applications. New York, N. Y., McGraw-Hill Book Co., Inc. 1961.

2. COBOL 1961. Revised Specifications for a Common Business Oriented Language. Washington, D. C., U. S. Government Printing Office. 1961.

3. Elbourn, R. D. and Ware, W. H. The Evolution of Concepts and Languages of Computing. Institute of Radio Engineers Proceedings. 50: 1059-1066. 1962.

4. Fleishman, A. M. Future is Bright for EDP in Smaller Companies. Automation. 8: 66-72. 1961.

5. Greenburger, M. Management and the Computer of the Future. New York, N. Y., John Wiley and Sons, Inc. 1962.

6. Jeenel, J. Programming for Digital Computers. New York, N. Y., McGraw-Hill Book Co., Inc. 1959.

7. Katz, J. H. and McGee, W. C. An Experiment in Non-Procedural Programming. American Federation of Information Processing Societies Conference Proceedings. 24: 1-13. 1963.

8. Ledley, R. S. Programming and Utilizing Digital Computers. New York, N. Y., McGraw-Hill Book Co., Inc. 1962.

9. McCracken, D. D. A Guide to COBOL Programming. New York, N. Y., John Wiley and Sons, Inc. 1963.

10. Mullery, A. P., Rice, R., and Schauer, R. F. Adam--A Problem-Oriented Symbol Processor. American Federation of Information Processing Societies Conference Proceedings. 23: 367-380. 1963.

11. Orchard-Hays, W. The Evolution of Programming Systems. Institute of Radio Engineers Proceedings. 49: 283-295. 1961.

12. Pfeiffer, J. The Thinking Machine. New York, N. Y., J. B. Lippincott Co. 1962.

13. Sammet, J. E. Basic Elements of COBOL 61. Communications of the Association for Computing Machinery. 5: 237-253. 1962.

14. Serrell, R., Astrahan, M., Patterson, G. W., and Pyne, I. B. The Evolution of Computing Machines and Systems. Institute of Radio Engineers Proceedings. 50: 1039-1058. 1962.

15. Sharpe, R. A. Computer Software. Unpublished paper presented at Electrical Engineering Seminar, Ames, Iowa, Sept. 1962. Mimeo. Ames, Iowa, Iowa State University of Science and Technology, Department of Electrical Engineering. ca. 1962.

ACKNOWLEDGEMENT

The author wishes to thank his major professor, Dr. A. V. Pohm, for suggesting the topic and for offering helpful comments throughout the development of this dissertation.

APPENDIX A

This appendix will not contain the rules needed to write correct COBOL programs, since to do this would be equivalent to reproducing the complete specifications. The treatment given below, therefore, is a much simplified version of COBOL-1961. For a more extensive description of the COBOL language, the reader is directed to references 2, 8, 9, and 13, from which the following basic features have been taken.

The COBOL Language

COBOL (Common Business Oriented Language) is a problem oriented language which is a subset of normal English, and is suitable for expressing the solutions to business data processing problems. These problems involve the systematic manipulation of large masses of data. Hence to perform a COBOL computation the computer requires two kinds of information:

1. The program, in COBOL language, that is to be executed in order to perform the processing
2. The data which is being processed

The basic principle in considering the data is grouping of related data. A file is the largest set of related information, and consists of any number of records directed toward some one purpose. These records are in turn made up of sub-records of smaller sizes called data items, which are recorded in a predetermined format. A piece of data which is never subdivided is called an elementary item. Each record in a file contains a key word by which it can be identified for processing. All

records of a similar type, that is, belonging to a given file, are processed, one at a time, in sequence.

A program written in COBOL is known as a source program. It can be translated by a compiler into a machine language form known as the object program which will run on the specified computer and produce answers to the problem. A source program for data processing will contain four elements:

1. An identification of the program, the author, and the date
2. A description of the equipment being used in the processing
3. A description of the data being processed
4. A set of procedures to specify how the data is to be processed

COBOL is divided into four divisions, one for each of the above elements. The divisions are inserted into the computer in the order listed. The first and second divisions are known as the Identification and Environment Divisions, respectively. The last two, the Data Division and the Procedure Division, will be described further in the following few paragraphs.

Data Division

In order that the computer should be able to distinguish the different data items, the information on which processing is to be carried out, the computer must be "told" the format, with the different data items given data names. The Data Division specifies the characteristics and organization of the data records and effectively specifies symbolic addresses to the data items.

The entries in a COBOL record description are built around the concept of a level of data. The record itself is the highest level and must always have the level number 1. Within a record there may be groups of elementary items which may have level numbers as high as 49. Groups and elementary items may both have data names which can be referred to in the Procedure Division. The level structure is presented in the Data Division by writing the level number of each record, group, or elementary item before the name.

In addition to describing data on input and output files, the Data Division also describes constants and temporary data which may be held in "working storage" locations in the computer memory for use during the processing.

Procedure Division

The Procedure Division specifies the steps that the user wishes the computer to follow in processing the data. These steps are expressed in terms of meaningful English words, statements, sentences, and paragraphs. Verbs denote actions, sentences describe procedures, and conditional statements provide alternative paths of action. This aspect of the overall system is often referred to as the "program"; in reality it is only part of the total specification of the problem solution.

The Procedure and Data Divisions are kept separate in the source program -- but everything in the object program depends on how the data is stored and organized. Keeping the divisions separate makes it easy to change the program slightly or correct errors. The program is also easier to write in the first place, because the writer does not have to be

continually conscious of the data structure while trying to concentrate on the instruction sequences.

Characters

In defining any language it is necessary to state clearly the set of characters which can be used, and the rules for grouping them. A character is the most elementary unit of data, and may be a numeric digit, a letter, or a special character. COBOL-61 contains 51 characters used for words, editing, punctuation, formulas, and relations.

The following lists show the specific COBOL character set:

Characters Used in Words

0,1,...,9

A,B,...,Z

- (hyphen)

Characters Used in Editing

$ (dollar sign)

* (check protection symbol)

, (comma)

. (actual decimal point)

Characters Used in Punctuation

" (quotation mark)

((left parenthesis)

) (right parenthesis)

(space)

; (semicolon)

Characters and English Equivalents Used in Relations and Formulas

> GREATER THAN

< LESS THAN

= EQUALS

+ PLUS

- MINUS

* MULTIPLIED BY

/ DIVIDED BY

** EXPONENTIATED BY

Words

A COBOL word is composed of not more than 30 characters chosen from the set of numerals, letters, and the hyphen. A word may not begin or end with a hyphen. A word may be terminated with a space or one of several punctuation marks. A few of the more important types of words will be discussed briefly.

A noun is a single word which is a data name, literal, condition name, or procedure name. The data name is the name (or address) of a data item. That is, each of the data items, which themselves never appear explicitly in a COBOL program but reside on the data tape, must be referred to by a data name. A data name must contain at least one letter, and it must not contain any spaces.

There are occasions when it is desirable to include a data item explicitly in the COBOL program itself; such data items are called literals. Nonnumerical literals must be placed within quotation marks so that they can be distinguished from data names.

It is often desired to give names to the literals themselves, and these are called condition names. Thus the value of a condition name is a literal, which appears explicitly in a COBOL code; the value of a data name is a data item on the data tape.

In COBOL the individual statements in the Procedure Division cannot be labeled. Instead, only a paragraph of statements can be labeled, and such labels are called procedure names.

A verb denotes an action and appears only in the Procedure Division. The verbs are divided into the following categories: arithmetic, input-

output, procedure branching, data movement, ending, and compiler directing. The complete list of verbs is given below:

Verbs (23)

Arithmetic (5)

ADD
SUBTRACT
MULTIPLY
DIVIDE
COMPUTE

Procedure Branching (3)

GO TO
ALTER
PERFORM

Data Movement (2)

MOVE
EXAMINE

Ending (1)

STOP

Input-Output (6)

READ
WRITE
OPEN
CLOSE
ACCEPT
DISPLAY

Compiler Directing Declaratives (3)

DEFINE
USE
INCLUDE

Compiler Directing Verbs (4)

ENTER
EXIT
INCLUDE
NOTE

Reserved words are required key words which have a precise meaning in a COBOL program, such as the verbs and other words associated with verb formats. These reserved words must not be used for data names or for any other purpose than that specified. Since only a few reserved words are hyphenated, invented data names may be hyphenated to ensure that a reserved word has not been used. A digit may be put into an invented name for the same purpose, since no reserved words contain digits.

Statements, Sentences, and Paragraphs

All statements begin with verbs, followed by other words of a type and sequence dictated by the allowable format of each verb. The statements

in COBOL are placed together to form sentences. A sentence is composed of one or more statements separated by a comma, a semicolon, the word AND, or the word THEN, and ends with a period and a space.

A group of successive sentences that may be related in function form a paragraph, which is written with a physical separation. Only a paragraph can have a label, called a procedure name, which is placed as a heading to the paragraph. The purpose of paragraphs is to permit the user to name groups of sentences so that control can be passed to them. When a paragraph is referenced by a GO TO verb, control is passed to the first sentence of the named unit.

APPENDIX B

The EVE Language

The following presents a brief summary of the basic features of the EVE language. For more detailed specifications, see reference 10.

Syntax

The EVE language consists of verbs, nouns, and modifiers, and rules for their use. These rules are the syntax of the language. An English-like structure is used. A noun with any number of adjectives modifying it makes up a noun phrase. All adjectives follow the modified noun. An adjective is considered to modify not just the noun, but the noun as modified by any adjective preceding the particular adjective.

A verb and any number of adverbs modifying it makes up a verb phrase. An adverb precedes the modified verb.

An operation is a combination of at least one verb phrase and one noun phrase which accomplish some process. A sentence consists of at least one operation.

Nouns

A data sequence may be given a name. These names become the nouns of the language. The name can be formed from any combination of any number of the general characters (small or capital letters, numbers, or special symbols). The first character of a name must be a letter. These nouns may be given any definition the programmer desires; however, names must be unique in that the name must refer to only one data sequence at a given

time. Names of data must be surrounded on each side by a name symbol (n). The data being defined is the data following the name.

Certain defined nouns are provided in the language. These are:

1. it -- This will indicate the preceding name in the operating sequence.
2. this -- This will indicate the sentence in the instruction which is currently being interpreted.
3. unit -- This will indicate the input-output unit whose number is indicated by the following noun.

Verbs

In the machine language, a set of verbs with defined meaning would be built into hardware. A few of these are described in the list that follows.

1. add (+) -- The numeric data indicated by the subject is added to the numeric data indicated by the object.
2. subtract (-) -- The numeric data indicated by the object is subtracted from the numeric data indicated by the subject.
3. multiply (*) -- The numeric data indicated by the subject is multiplied by the numeric data indicated by the object.
4. divide (/) -- The numeric data indicated by the subject is divided by the numeric data indicated by the object.
5. set -- The numeric data indicated by the subject is set to the number of places beyond its absolute decimal place indicated by the object.
6. truncate -- The data indicated by the subject is truncated to a number of characters indicated by the object.

7. edit -- The data indicated by the subject is modified according to the format noun indicated by the object.
8. define (→) -- The information indicated by the subject is placed at the indicated location.
9. delete -- The data indicated by the object is deleted.
10. search -- The data indicated by the object is searched for an element which meets the condition specified by the conditional statement which follows.
11. transfer (⌜) -- Program control is transferred to the instruction sentence, or phrase or symbol within the sentence, indicated by the object.
12. if -- If the conditional statement following if is satisfied, perform the remaining operations in the sentence; otherwise, transfer control to the next sentence.
13. stop -- This instruction causes the computer to stop.

New verbs may also be defined using the previously defined verbs and nouns. In addition, the noun op n is available for use in communication between the defining instruction and the calling instruction. This noun will indicate the nouns in the calling verb phrase. If any one noun in the calling verb phrase is required -- the n^{th} noun, for example -- then this is indicated by writing op n. The only requirement for a particular combination of letters to be used as a verb is that this combination must have a unique definition.

Adverbs

1. round -- Round the last digit of the result of the set or truncate verb which this adverb modifies.
2. name -- This adverb will indicate that the modified verb is to apply only to the names in the data specified.

Adjectives

1. following ((i) ↑ j) -- This will indicate, if j is positive, the j^{th} item at level i following the data previously specified.
2. set current ((i) $ j) -- This will indicate, if j is positive, the j^{th} item at level i following the data previously specified. This new item will then become the "current" item in the data string.
3. current ((i) ¢ j) -- This will indicate, if j is positive, the j^{th} item at level i following the current location in the data previously specified. This new item will then become the "current" item in the data string.
4. line -- This acts like a mention-identifier and, used with the set current and current adjectives, will indicate the line number and the line number beyond the current line, respectively.

Conditional statements

A conditional statement results in a determination of the condition of either the truth or falsity of the statement. The verbs used in a conditional statement are: equal, not equal, less than, less than or equal, greater than, greater than or equal, and reach. In using the reach verb, the identifier following the item of data specified by the subject

is compared against the identifier mentioned as the object of this verb to establish the condition of equality.

Format nouns

The format noun indicates the editing to be performed on the data symbol indicated by the subject of the edit verb of which the format noun or its name is the object.

The positioning of the decimal place or right-hand character within the resulting field is accomplished through the use of the Position Decimal Point indicator (M) and Position Right indicator (A). The floating or float-inserting of a character is accomplished through the use of the Float (K) or Float Insert (I) indicators. The insertion, deletion, and leaving unchanged of characters is accomplished through use of the Character Delete (N), Print (P), Significant Conditional Print (W), Positive Conditional Print (X), Negative Conditional Print (Y), and Zero Print (Z) indicators.

Data

Data consists of strings of symbols upon which is usually imposed some form of grouping or hierarchy. These groupings are indicated by some form of identifiers. In describing data, the EVE language makes use of the following groups and their associated identifiers:

Group	Identifier	
	USE	MENTION
Character	(Implied)	⓪
Symbol	①	①
Phrase	②	②

Sentence	③	③
Paragraph	④	④
Chapter	⑤	⑤
Book	⑥	⑥

When the structure of a string of data or instructions is to be identified, then the "USE" identifiers are used. If some group of data is being referred to in the instructions, then the "MENTION" identifiers are used. The ① identifier indicates the end of a symbol and the start of a new symbol. These identifiers form a hierarchy in that a ② also implies a ①, a ③ implies a ②, and a ①, etc.

The names of data will appear with the data itself. A name can be given to any string of data at any level from Symbol to Book. This string may contain named groups of data. Thus, named groups within named groups are allowed. The name character n will surround the name when it appears with the data. The first character following the second n will be a "MENTION" identifier which will indicate the level of the data being named.

A name of an instruction or group of instructions must be defined as a verb, and surrounded by the character v.

Input-output

Data is of variable field length, and all memory assignment of named sequences of data and instructions is accomplished by the machine. The programmer need only provide the machine with an input list indicating where named blocks are located and a priority, and an output list indicating where data should go and a priority. The input and output list

will have a structure similar to the data itself. Together, the input list and output list should form a paragraph and must be given the name "IO List".