Principles of Programming

Section 11: Additional Programming Methods

IBM Personal Study Program

Section 11: Additional Programming Methods

11.1 Introduction

The presentation of computer coding and programming in this book has been based primarily on the use of the Symbolic Programming System and Autocoder, in the interest of a unified presentation. The reader should realize, however, that there are many other programming techniques and systems that serve much the same purposes as SPS and Autocoder, namely, to reduce the work of initial programming and to simplify program modification. These other systems also sometimes help to take advantage of special conditions and to handle other types of problems than have been illustrated here.

In other words, SPS and Autocoder are good, and are heavily used—but they are not the whole story. In this rather brief section we shall investigate a few additional methods of which the reader should be aware. The discussion will give only a quick introduction to the various topics. Other publications are available for the reader who wishes more complete information.

The topics to be considered are: the use of decision tables in system design; the FORTRAN coding system for scientific and engineering problems; the Report Program Generator for the production of programs to write reports; and the COBOL coding system, which is a sophisticated coding language for expressing data processing procedures. No attempt will be made to evaluate these various advanced programming languages, because the whole subject is under such intensive development at the present time that any evaluation would soon be out of date.

11.2 Decision Tables

Before any coding can be done on a problem, there must be a precise definition of the procedure to be followed. As we have seen, this definition of the problem and procedure can easily require more effort than the coding and checkout which follow, and the effort is usually at a more sophisticated level. As with any other activity, this work, which is usually called systems design, requires methods of representing the actions to be carried out and the conditions under which they are to be done. In this book we have used two techniques for this purpose: narrative description and flow charts. Various other methods of describing the work are employed occasionally.

Another method that is coming into prominence is the use of *decision tables*. A decision table is a rectangular array of boxes, organized in such a way as to describe a decision system involving many variables and many results. The basic arrangement of a decision table is shown in Figure 1, where we see that a horizontal double line separates *conditions* above from *actions* below, and a vertical double line separates the *stub* on the left from the *entries* on the right. The stub contains the descriptions of the conditions and actions for each row of the table. Figure 2 is a simple example of a decision table to describe the decisions in one section of the inventory control case study of Section 10.

The basic idea of using a table is to inspect the decision parameters, one column at a time, until a column is found in which all the conditions are satisfied. When this occurs, the actions contained in that column are to be carried out. If a condition entry is blank, then that condition has no bearing on the decision as to which *rule* (column) is to be followed. For instance, if the transaction is an order, we don't care whether the transaction quantity is more or less than the quantity on hand. If an action entry is blank, no action is required for the operation named in the action stub for that row. For instance, there is no quantity shipped in this example except on an issue.

At the end of the table is ordinarily a line stating where to go if the table cannot be solved—that is, if no column's conditions are satisfied by the input. Following this is the name of the table which normally should be solved next. If the choice of the following table depends on decisions in the body of the table, as in this example, a separate row can be set up to determine this branching.

Even in a simple example like this it may be seen that a decision table provides a clear picture of the relationships between conditions and actions, and of the interrelationships of combinations of conditions. The technique has the added advantage that omissions are explicitly indicated, leading to a complete statement of the procedure early in the planning.

The decision table concept is much broader than this example might indicate, extending to such diverse areas as tape processing logic, accounting procedures, manufacturing operations, routine engineering decisions, and the writing of utility routines. Furthermore, the decision table concept is not limited to providing insights into the logic of a system. A suitably standardized form of decision table is also feasible as a source language for describing data processing procedures to a computer.

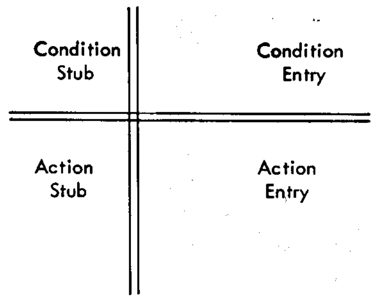


Figure 1. Schematic diagram of the parts of a decision table

Transaction Type	Issue	Issue	Receipt	Order	Adjustment
Transaction Quantity	≤ QOH	> QOH			
QOH =	QOH - TQ	0	QOH + TQ	QOH	TQ
QOO =	QOO	QOO	QOO - TQ	QOO + TQ	QOO
Quantity Shipped =	TQ	QOH			
GO TO	Extend	Shortage	Reorder	Reorder	Reorder

If unsolvable, go to CODE-ERROR

Figure 2. Decision table to describe an inventory control procedure

11.3 The FORTRAN Coding System

FORTRAN is a source program language for expressing problems in science and engineering, together with a processor that translates the source program statements into an object program that can be run on a computer. In contrast with SPS and Autocoder, the source program language has virtually no relation to the object program language. In fact, the FORTRAN user need not even know how the computer operates in order to write a program.

An example will show the broad outline of the language. Suppose that an engineer wishes to find the current flowing in an AC circuit, for frequencies between 1,000 cycles per second and 2,000 cycles per second, in steps of 50. Assume that the circuit and the operating conditions are such that the current is given by the formula:

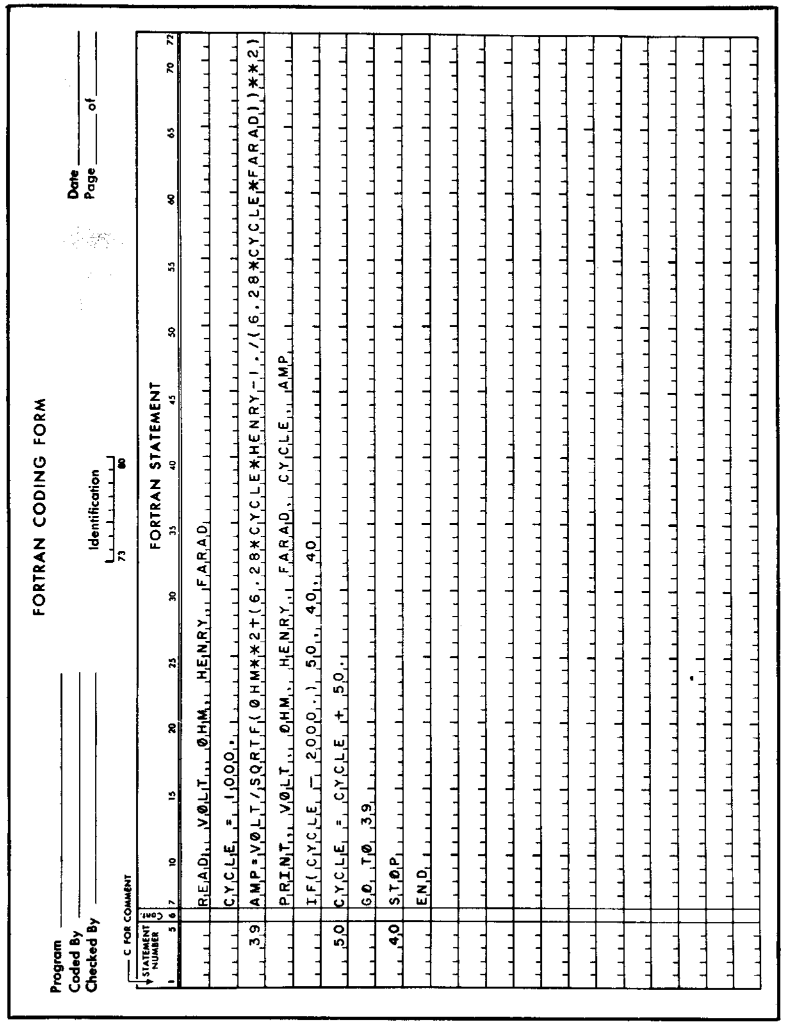
$$I = \frac{E}{\sqrt{R^2 + \left(2\pi FL - \frac{1}{2\pi FC}\right)^2}}$$

where	I	=	current, amperes
	E	=	voltage, volts
	R	=	resistance, ohms
	L	=	inductance, henries
	C	=	capacitance, farads
	F	=	frequency, cycles per second

It is of course possible to program this calculation for a computer using machine-language instructions, and a great deal of work of this type has been done in recent years. However, doing so requires the engineer to know how to program, and involves many computer considerations that really have nothing to do with the problem being solved.

FORTRAN makes it possible to state the procedure to the computer in a style which closely resembles ordinary mathematical notation, and which requires virtually no computer knowledge. The program for this computation is shown in Figure 3.

It may be seen that variables are given names in the same general fashion as with SPS. In this program we have used the units in which the quantities are expressed as their names.



FORTRAN CODING FORM

Program ______ Coded By ______ Checked By ______ Identification 73 80 Date ______ Page ______ of ______

```
STATEMENT
 NUMBER   C FOR COMMENT   FORTRAN STATEMENT
1    5  6 7                                                        72
          READ, VØLT, ØHM, HENRY, FARAD
          CYCLE = 1000.
   39     AMP=VØLT/SQRTF(ØHM**2+(6.28*CYCLE*HENRY-1./(6.28*CYCLE*FARAD))**2)
          PRINT, VØLT, ØHM, HENRY, FARAD, CYCLE, AMP
          IF(CYCLE - 2000.) 50, 40, 40
   50     CYCLE = CYCLE + 50.
          GØ TØ 39
   40     STØP
          END
```

Figure 3. A FORTRAN program

The READ statement calls for the four data values to be read from a card. The next statement establishes 1,000 as the initial value of the frequency. Statement 39, which is given a statement number because it will be necessary to return to it, calls for the actual computation. We see here the use of symbols to specify arithmetic operations, according to the convention:

Addition	+
Subtraction	−
Multiplication	*
Division	/
Exponentiation	**

We see also that the square root is called for simply by writing the name SQRTF. When the FORTRAN processor encounters this *function* name, it will incorporate into the object program a routine to take a square root. The FORTRAN programmer never has to know how to write the 15 machine-language instructions by which square roots are usually computed, or even know what the method is.

The PRINT statement leads to object program instructions to print the input data and the results. The next statement is the FORTRAN equivalent of a conditional branch. The effect, in this case, is this: If the frequency is less than 2,000, go to statement 50 where we set up the next value of the frequency, but if the frequency is equal to or greater than 2,000, go to the STOP statement. Statement 50 is another example of an arithmetic formula statement. This is not an equation, but rather a command to FORTRAN: Replace the value of the variable named on the left with the value of the expression on the right. The GO TO 39 creates in the object program a simple unconditional branch.

FORTRAN is an example of a *procedure-oriented* language; that is, the language is used to write a problem-solving procedure in terms of the method to be followed. As we shall see, COBOL is also a procedure-oriented language. This is in contrast to programming systems like SPS, where the procedure must be described more in terms of the machine operations to be executed, and which are therefore called *machine-oriented* languages. Procedure-oriented languages have a number of advantages:

1. They are generally somewhat easier to learn and use than machine-language coding systems. The beginner does not have to know anything about how the machine operates in order, for instance, to take a square root: he simply writes SQRTF. This advantage should be kept in perspective, however. It must be remembered that getting a problem solved with a computer involves many activities besides coding. The use of a language like FORTRAN or COBOL in no way reduces the careful planning that must go into getting a correct problem statement, or determining the best approach to the solution, or planning a thorough set of test cases, or documenting the program. What FORTRAN does, actually, is simplify the job of coding so the programmer can concentrate on these other things.

2. Program modifications are easier to make, because of features designed into each of the languages. In the case of COBOL, it is because the description of the procedure is kept rigidly separate from the description of the data; this means that one can be changed without having to rewrite the other.

3. The procedure statements are to a large extent independent of the machine on which the object program will be run. The FORTRAN program in Figure 3, with a few modifications, can be compiled and run on any one of a dozen or more different computers. The object programs for the various machines would be different—but the source program is largely independent of this fact.

FORTRAN has the desirable characteristic that it is attractive both to the beginner and to the expert. To the beginner, FORTRAN offers the advantage of ease of learning and the quick solution of simple problems. To the expert, it offers faster coding, ease of modification, and machine independence.

11.4 The Report Program Generator

The desirability of simpler ways of programming is of course not restricted to scientific computations. The Report Program Generator is one of several systems that provide much the same advantages for commercial data processing that FORTRAN does for scientific work. Using this system, the source-language user once again does not have to know much about machine-language coding. The procedure is stated on four types of forms which provide answers to the following questions:

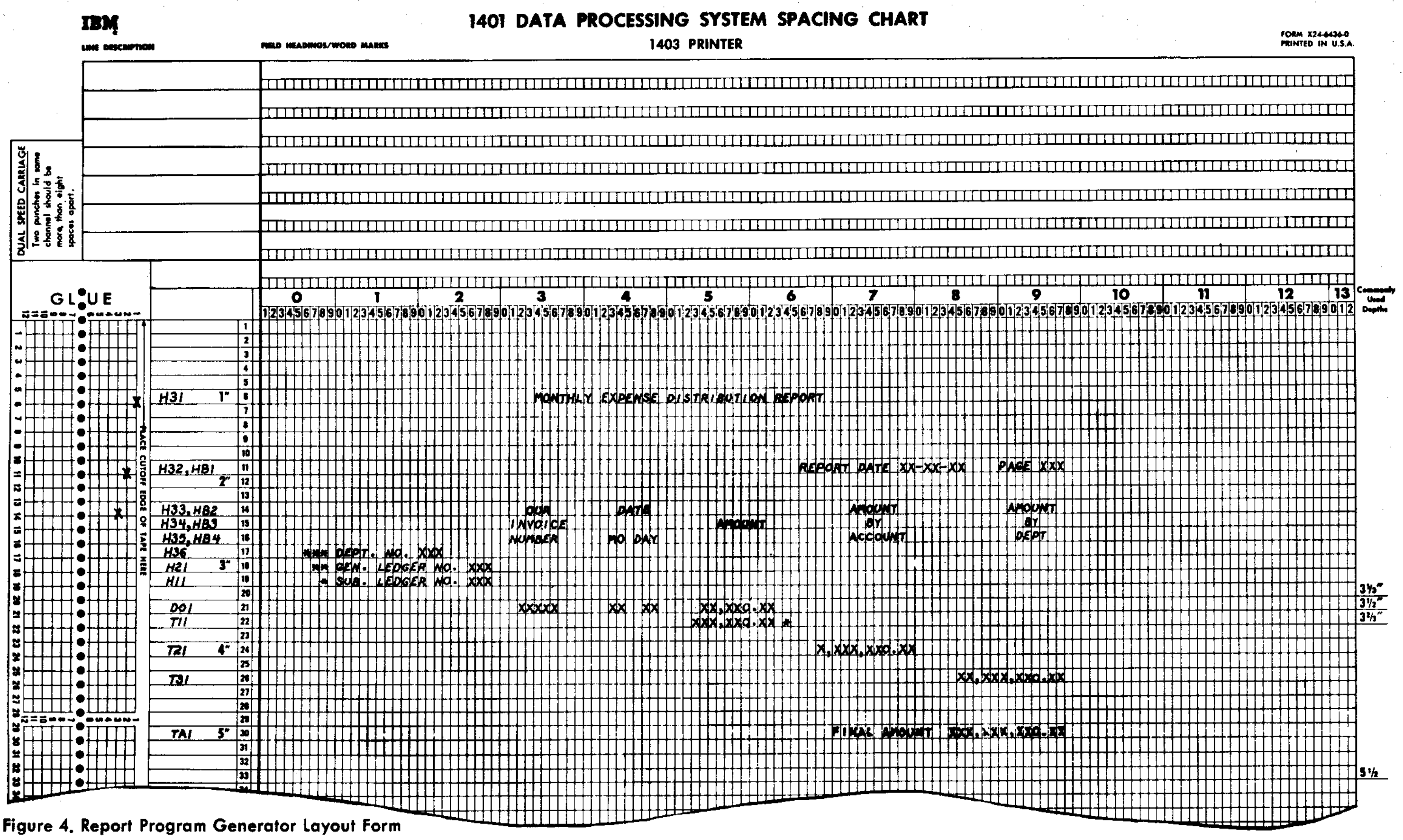
1. What are the characteristics of the file from which the data to appear in the report is obtained?

2. What type of information is to be extracted from the file and from what records may these source files be obtained?

3. What type of calculations are to be performed during the execution of the object program and how are the results of these calculations to be manipulated?

4. What will be the format of the report? What headings and constants must it contain? How should the data composing the report be edited?

These questions naturally must be answered in setting up any program. Using the Report Generator, however, the answers are presented to the computer on four types of forms that require no computer knowledge to fill out; the generator then takes over and produces an object program. The four forms are shown in Figures 4 through 7.


IBM — LINE DESCRIPTION — FIELD HEADINGS/WORD MARKS

1401 DATA PROCESSING SYSTEM SPACING CHART

1403 PRINTER

FORM X24-6436-0 PRINTED IN U.S.A.

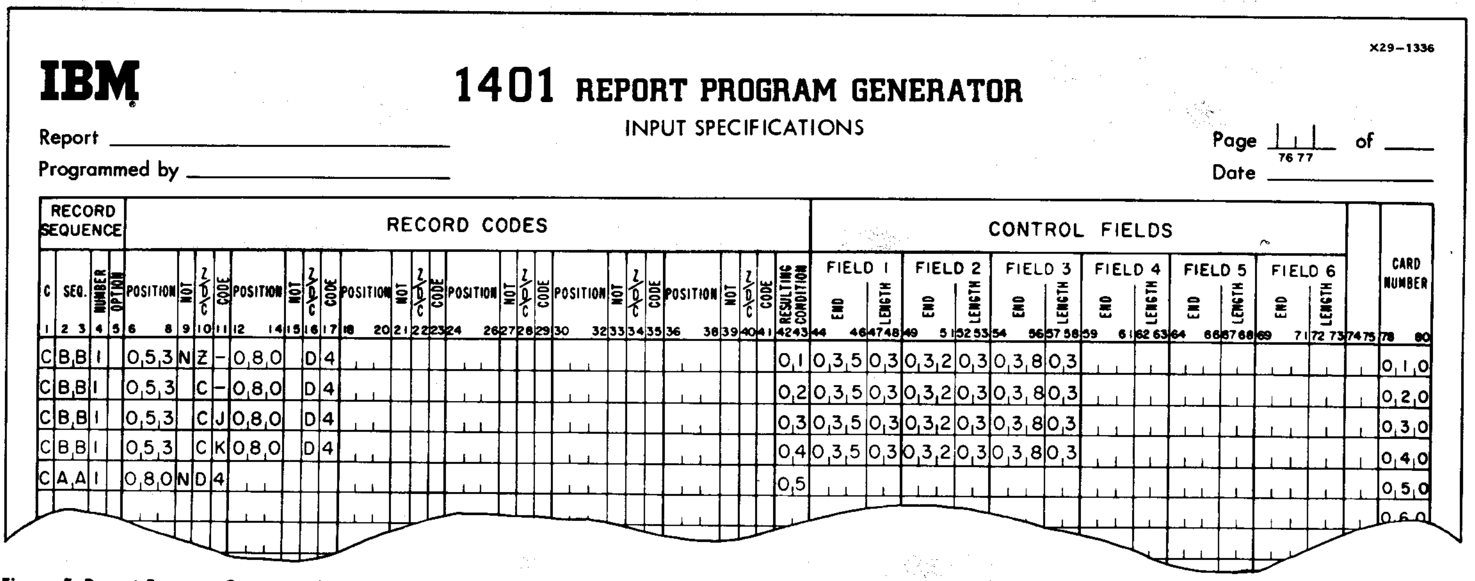
DUAL SPEED CARRIAGE — Two punches in same channel should be more than eight spaces apart.

GLUE — PLACE CUTOFF EDGE OF TAPE HERE

Commonly Used Depths: 3⅓″ 3½″ 3⅔″ 5½

Line	Line description	Print line
6	H31 1″	MONTHLY EXPENSE DISTRIBUTION REPORT
11	H32, HB1	REPORT DATE XX-XX-XX PAGE XXX
12	2″	
14	H33, HB2	OUR DATE AMOUNT AMOUNT
15	H34, HB3	INVOICE AMOUNT BY BY
16	H35, HB4	NUMBER MO DAY ACCOUNT DEPT
17	H36	*** DEPT. NO. XXX
18	H21 3″	** GEN. LEDGER NO. XXX
19	H11	* SUB. LEDGER NO. XXX
21	D01	XXXXX XX XX XX,XX0.XX
22	T11	XXX,XX0.XX *
24	T21 4″	X,XXX,XX0.XX
26	T31	XX,XXX,XX0.XX
30	TA1 5″	FINAL AMOUNT XXX,XXX,XX0.XX

Figure 4. Report Program Generator Layout Form


X29-1336

IBM **1401 REPORT PROGRAM GENERATOR**

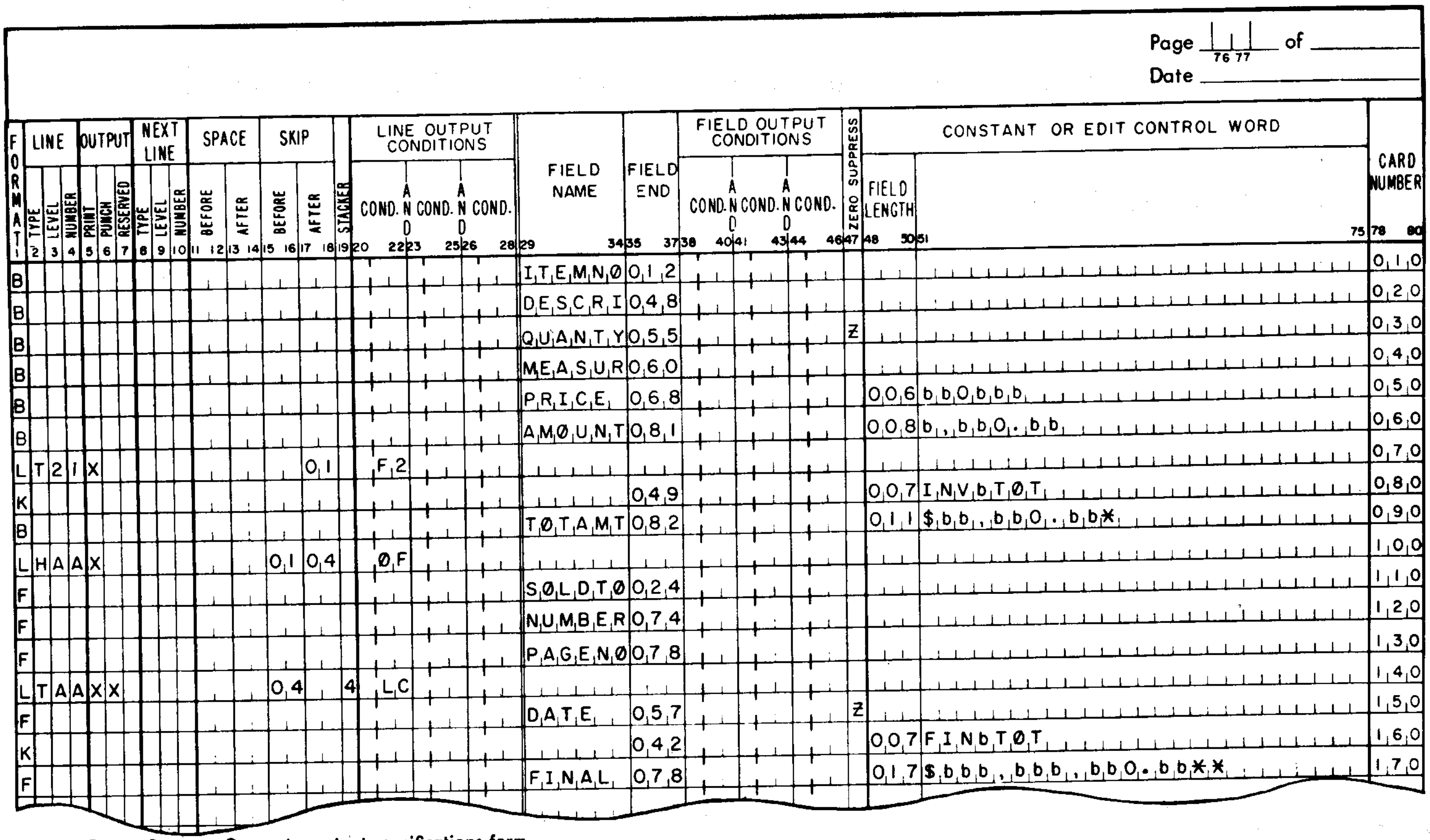
INPUT SPECIFICATIONS

Report ____________ Page ⌊⌊ of ____ (76 77)

Programmed by ____________ Date ____________

RECORD SEQUENCE				RECORD CODES																									CONTROL FIELDS													
																													FIELD 1		FIELD 2		FIELD 3		FIELD 4		FIELD 5		FIELD 6			
C	SEQ.	NUMBER	OPTION	POSITION	NOT	Z/D/C	CODE	POSITION	NOT	Z/D/C	CODE	POSITION	NOT	Z/D/C	CODE	POSITION	NOT	Z/D/C	CODE	POSITION	NOT	Z/D/C	CODE	POSITION	NOT	Z/D/C	CODE	RESULTING CONDITION	END	LENGTH	END	LENGTH	END	LENGTH	END	LENGTH	END	LENGTH	END	LENGTH		CARD NUMBER
1	2 3	4	5	6 8	9	10	11	12 14	15	16	17	18 20	21	22	23	24 26	27	28	29	30 32	33	34	35	36 38	39	40	41	42 43	44 46	47 48	49 51	52 53	54 56	57 58	59 61	62 63	64 66	67 68	69 71	72 73	74 75	78 80
C	BB	I		053	N	Z	-	080		D	4																	01	035	03	032	03	038	03								010
C	BB	I		053		C	-	080		D	4																	02	035	03	032	03	038	03								020
C	BB	I		053		C	J	080		D	4																	03	035	03	032	03	038	03								030
C	BB	I		053		C	K	080		D	4																	04	035	03	032	03	038	03								040
C	AA	I		080	N	D	4																					05														050
																																										060

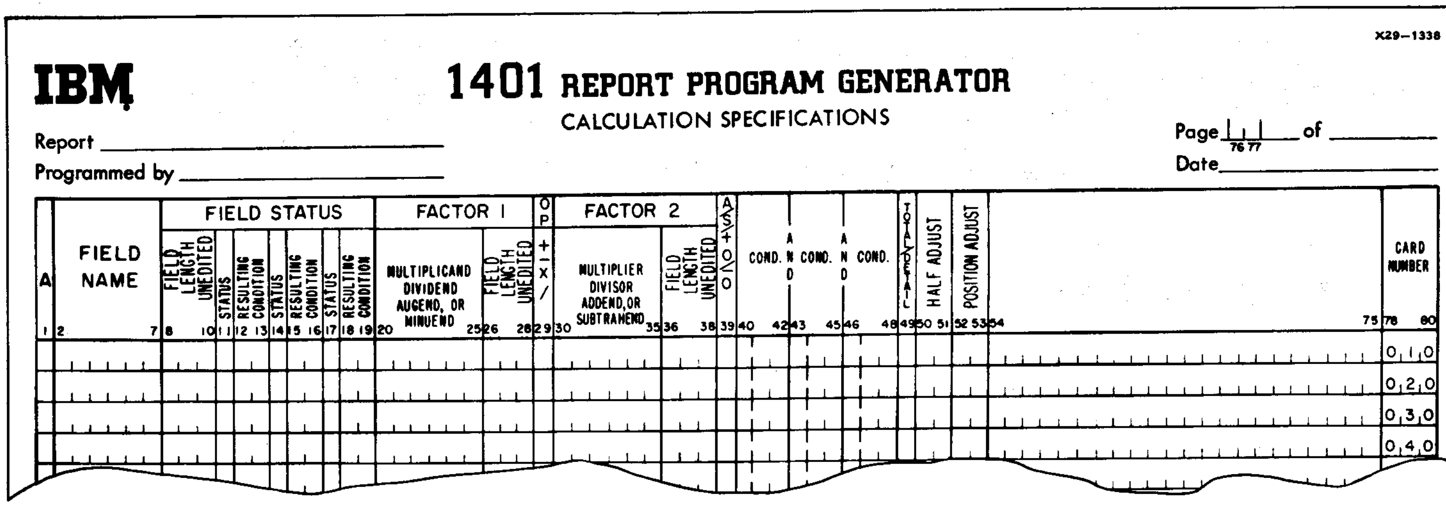
Figure 5. Report Program Generator input specifications form



Page __ of __
Date __

FORMAT	LINE TYPE	LINE LEVEL	LINE NUMBER	OUTPUT PRINT	OUTPUT PUNCH	OUTPUT RESERVED	NEXT LINE TYPE	NEXT LINE LEVEL	NEXT LINE NUMBER	SPACE BEFORE	SPACE AFTER	SKIP BEFORE	SKIP AFTER	STACKER	LINE OUTPUT CONDITIONS COND.	AND COND.	AND COND.	FIELD NAME	FIELD END	FIELD OUTPUT CONDITIONS COND.	AND COND.	AND COND.	ZERO SUPPRESS	FIELD LENGTH	CONSTANT OR EDIT CONTROL WORD	CARD NUMBER
1	2	3	4	5	6	7	8	9	10	11 12	13 14	15 16	17 18	19	20 22	23 25	26 28	29 34	35 37	38 40	41 43	44 46	47	48 50	51 75	78 80
B																		ITEMNØ	012							010
B																		DESCRI	048							020
B																		QUANTY	055				Z			030
B																		MEASUR	060							040
B																		PRICE	068					006	bb0bbb	050
B																		AMØUNT	081					008	b,bb0.bb	060
L	T	2	1	X									01		F2											070
K																			049					007	INVbTØT	080
B																		TØTAMT	082					011	$bb,bb0.bb*	090
L	H	A	A	X								01	04		ØF											100
F																		SØLDTØ	024							110
F																		NUMBER	074							120
F																		PAGENØ	078							130
L	T	A	A	X	X							04		4	LC											140
F																		DATE	057				Z			150
K																			042					007	FINbTØT	160
F																		FINAL	078					017	$bbb,bbb,bb0.bb**	170

Figure 6. Report Program Generator output specifications form



X29-1338

IBM

1401 REPORT PROGRAM GENERATOR

CALCULATION SPECIFICATIONS

Report __
Programmed by __

Page __ of __
Date __

A	FIELD NAME	FIELD STATUS FIELD LENGTH UNEDITED	STATUS	RESULTING CONDITION	STATUS	RESULTING CONDITION	STATUS	RESULTING CONDITION	FACTOR 1 MULTIPLICAND DIVIDEND AUGEND, OR MINUEND	FACTOR 1 FIELD LENGTH UNEDITED	OP + - X /	FACTOR 2 MULTIPLIER DIVISOR ADDEND, OR SUBTRAHEND	FACTOR 2 FIELD LENGTH UNEDITED	A S + 0 - 0	COND.	AND COND.	AND COND.	TOTAL/DETAIL	HALF ADJUST	POSITION ADJUST		CARD NUMBER
1	2 7	8 10	11	12 13	14	15 16	17	18 19	20 25	26 28	29 30	30 35	36 38	39	40 42	43 45	46 48	49	50 51	52 53	54 75	78 80
																						010
																						020
																						030
																						040

Figure 7. Report Program Generator calculation specifications form

This system applies to any task that involves extracting information from a file, summarizing it, performing calculations on it, editing it, and printing the results. For the purposes of this discussion, a "report" is just about anything that can be printed on a printer: a sales summary, a payroll check, an earnings statement, an invoice, a customer bill, etc.

11.5 The COBOL Programming System

COBOL, which stands for COmmon Business Oriented Language, is a procedure-oriented language for stating procedures in business data processing. As such, it shares the advantages of procedure-oriented languages stated above in connection with FORTRAN.

A COBOL source program is composed of three sections:

1. Procedure division: the procedure statements that specify how the data is to be processed.

2. Data division: description of the format and organization of the data and results.

3. Environment division: a description of the equipment to be used by the object program.

These three divisions are separated in writing the source program, which leads to one of the major advantages of COBOL and languages like it—namely, that changing the procedure does not require changing the data descriptions, and vice versa. Considering that in most programs for commercial data processing the object program and the data arrangement are strongly interrelated, this becomes a sizable advantage. In the frequent situation where the data arrangement must be changed slightly, it is necessary only to modify the data division and recompile. This is in contrast to the situation with actual machine-language coding, for instance, where in extreme cases a single added digit in the data can force reprogramming most of the problem.

The independence of the procedure and data divisions leads to another major advantage of COBOL, which is also shared with FORTRAN and a number of other systems: machine independence. With rather minor changes, a COBOL source program can be compiled for running on any computer for which a COBOL processor exists. The data division does depend somewhat on the object machine, to take account of such machine characteristics as variable vs. fixed word length, the handling of signs, and tape formats. It usually turns out, however, that changing the data division is far less work than rewriting the whole program, and the relative machine independence is in fact achieved.

Procedure statements in COBOL are written in a form closely paralleling ordinary English construction. In fact, a completed COBOL program can in some cases be read like an English description of the procedure. There is, however, very little flexibility in the way the "sentences" can be written. English language readability is an advantage of COBOL, but a secondary one.

11.6 Fundamentals of COBOL Programming

To give a little better idea of what programming is like using procedure-oriented languages, we shall consider a little more fully the characteristics of one of them—namely, COBOL. This will be done first with a series of short examples that will introduce the fundamental ideas, and then the inventory control program of Section 10 will be rewritten in COBOL.

The central feature of a billing procedure is the multiplication of unit price by the quantity sold. A sentence in a COBOL program to do this could be as shown in Figure 8. In this sentence, the word MULTIPLY is a verb. The sentence, furthermore, is an imperative sentence. When the COBOL processor translates this sentence into actual machine instructions, it will create the instructions necessary to bring about the action specified by the verb.

```
01  MULTIPLY UNIT-PRICE BY QUANTITY GIVING TOTAL-PRICE.
```

Figure 8.

As can be seen from the example, a COBOL sentence is very similar to an ordinary English statement in construction and format. Actually the parallel extends to some aspects of punctuation. A COBOL sentence must always end with a period. Data is referred to by *name,* such as "TOTAL PRICE." We notice immediately an exception, however, in that words which are to be considered as combined in one name must be hyphenated, because in COBOL a blank space always indicates the beginning of a new word. Furthermore, although it is not apparent from this one example, the structure of a COBOL sentence must follow very precisely the rules laid down in the COBOL manual. It would not be possible to rewrite this sentence as THE TOTAL PRICE IS COMPUTED BY MULTIPLYING UNIT PRICE AND QUANTITY. This would be the English language equivalent, but COBOL would not accept such a sentence. We shall not attempt to give the precise rules for forming each type of COBOL sentence; this information may readily enough be obtained from a COBOL manual.

For a second example, consider a part of an inventory calculation. One sentence of the COBOL procedure for determining whether or not to place an order might be as shown in Figure 9. The first word "REORDER-ROUTINE" is called a *procedure name.* A procedure name provides a way of referring to the sentences that follow.

```
01  REORDER-ROUTINE.  IF QUANTITY-ON-HAND IS LESS THAN
02        MINIMUM MOVE ORDER-QUANTITY TO PURCHASE-AMOUNT.
```

Figure 9.

The sentence illustrates a *conditional expression* involving a simple relation between two quantities. If the quantity on hand is less than the reorder point, the action specified following the conditional clause is carried out. If the quantity on hand is not less than the reorder point, the action is not carried out. Figure 10 shows in schematic form the structure of this sentence. A *conditional clause*, which is introduced by the word IF, in effect asks a question to which the answer must be yes or no. We shall speak of each relation involved in a conditional expression as being true or false, or satisfied or not satisfied. This example uses the IS LESS THAN relation. The allowable relations in COBOL language are shown in Figure 11.

The example in Figure 9 also shows a different command, MOVE TO. The action called for is the copying of information within storage. The information named ORDER-QUANTITY is to be copied and called PURCHASE-AMOUNT.

It is probably apparent by now that certain words have special meanings in COBOL language: In the present examples, the words IF, THEN, MOVE, TO and the phrase IS LESS THAN all have special meaning, and confusion would result if we tried to interpret these words in any other way. Such words are a fixed part of the language and are called *reserved words*. A complete list of the reserved words in COBOL language may be found in a COBOL manual. Reserved words must not be used to mean anything but what COBOL defines them to mean.

To introduce a few more features of COBOL language, we may use a common payroll example as shown in Figure 12.

The conditional clause in this example is different from what we have seen previously. The first part of the clause consists of just the word HOURLY, which is called a *condition name*. HOURLY is one of the possible values which can be taken on by the implied data name PAYROLL TYPE, the other values being EXEMPT, SALARIED, TEMPORARY. Since there are only a few of these conditions, it is convenient for the programmer to use his normal terminology. The actual machine instructions are set up to work with the coded representation of these values, for instance, the numbers 1, 2, 3 and 4. Some way must be provided to correlate the condition names with the corresponding values. Establishing this correspondence is one of the many functions of the data division. Figure 13 shows the appropriate part of the data division to establish this correspondence.

PAYROLL TYPE is defined as a level 3 entry, which indicates its relative importance with respect to other elements of data. EXEMPT, SALARIED, HOURLY and TEMPORARY are named as the four conditions, and the code number used for each is given. Thus in this example, the value of PAYROLL TYPE is 3 whenever HOURLY is meant.

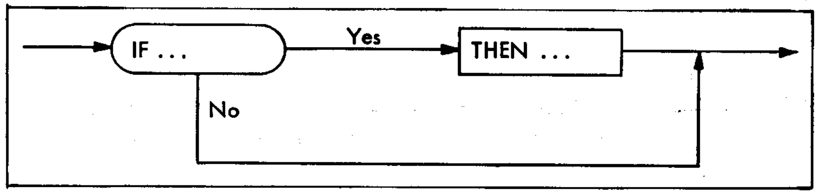


Figure 10.

Long Form	Short Form
IS EQUAL TO	EQUAL TO
IS NOT EQUAL TO	NOT EQUAL TO
IS LESS THAN	LESS
IS NOT LESS THAN	NOT LESS
IS GREATER THAN	GREATER
IS NOT GREATER THAN	NOT GREATER

Figure 11.

```
01  IF HOURLY AND HOURS-WORKED IS LESS THAN 40
02          GO TO GROSS-PAY,  OTHERWISE GO TO NET-PAY.
```

Figure 12.

```
01  3 PAYROLL-TYPE
02          88 EXEMPT VALUE IS 1
03          88 SALARIED VALUE IS 2
04          88 HOURLY VALUE IS 3
05          88 TEMPORARY VALUE IS 4
```

Figure 13.

The second part of the conditional clause is:

HOURS WORKED IS LESS THAN 40

In this case, the value of the data name HOURS WORKED is compared with the number 40; 40 is not to be interpreted as a data name but literally is the value of 40 itself. We speak of 40 as a *numerical literal.*

The second part of the conditional clause is joined to the first part by the reserved word AND, which specifies that both the first and the second part of the clause must be satisfied before carrying out the operations that follow the conditional clauses. This is shown schematically in Figure 14, which emphasizes that *both* parts of the conditional clause must be true before carrying out the action. If either or both of the parts is false, the action specified after OTHERWISE is executed. If the sentence is written without the word OTHERWISE the program continues with the following sentence.

In the COBOL sentence under consideration, the action specified is a new command, GO TO. The GO TO command makes it possible to get out of the one-after-the-other sequential execution of sentences and instead execute next the sentence named by the GO TO.

The fourth example is based on a part of an inventory control calculation. Assume as we have previously that there are just four types of transactions—recounts, receipts, orders and issues. The part of the job that we wish to consider is how to take action appropriate to the type of transaction. The program shown in Figure 15 is a little longer than the previous ones but most of the ideas are already familiar.

The first line of this program brings into play a processor command

NOTE INVENTORY RECORD MAINTENANCE

NOTE indicates that what appears in the rest of the sentence is information for the reader of the program; is not for the COBOL processor, which ignores it. The programmer is permitted and encouraged to use notes freely in order to make the program more intelligible to the reader. These play the same part as the comments in SPS and Autocoder programs.

The GO TO shown on the second line is a more powerful form of the command than we have seen before. This is called a *selective* GO TO. For any one transaction, only one of the four procedures named in parentheses will be performed. The one selected will depend on the current value of the transaction code, which can be from 1 to 4. These numbers correspond to the names within the parentheses. If the value is 1, the first name will be selected, etc. This is summarized in Figure 16.

An assigned GO TO provides a multiple branch or switching point. Lines 4 and 5 of Figure 15 illustrate another COBOL verb, namely, SUBTRACT. The SUBTRACT verb may also be used in the form:

SUBTRACT TRANSACTION-QUANTITY FROM QUANTITY-ON-HAND GIVING QUANTITY-ON-HAND

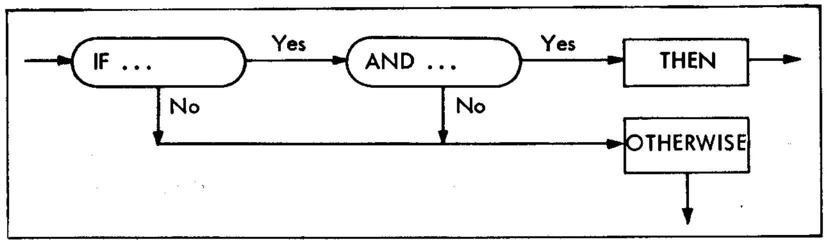


Figure 14.

```
01 NOTE INVENTORY RECORD MAINTENANCE
02      GO TO (RECOUNT-ROUTINE, ORDER-ROUTINE, RECEIPT-PROCEDURE,
03      ISSUE-PROCEDURE) DEPENDING ON TRANSACTION-CODE.
04 ISSUE-PROCEDURE.  SUBTRACT TRANSACTION-QUANTITY,
05      QUANTITY-ON-HAND.  PERFORM REORDER-CALCULATION.
06      GO TO NEXT-ITEM.
```

Figure 15.

If the current value of TRANSACTION-CODE is	Then GO TO:
1	RECOUNT-ROUTINE
2	ORDER-ROUTINE
3	RECEIPT-PROCEDURE
4	ISSUE-PROCEDURE

Figure 16.

In the condensed form used in Figure 15 the meaning is exactly the same as this—that is, the value corresponding to the first data name is subtracted from the value corresponding to the second data name. These are the only two ways the SUBTRACT verb may be used.

The last line of Figure 15 shows another type of transfer of control:

PERFORM REORDER-CALCULATION

The PERFORM verb may be thought of as meaning "go to the place named, do whatever it says to do, and come back." In our case, it is used to transfer to a procedure named REORDER-CALCULATION and to set up a return path so that after executing the procedure, control will return to the sentence immediately following PERFORM. PERFORM thus provides the facilities for a subroutine linkage. After performing the reorder calculation, control will return to and execute the GO TO NEXT ITEM sentence in line 6.

As another illustration of the use of COBOL we may use a savings bank procedure: updating an account record to indicate interest payment. The program might be as shown in Figure 17. Line 1 shows again the use of a procedure name to provide a named point to which the program can transfer. In this case it precedes a slightly different type of conditional clause. Instead of simply comparing two values as we have before, the programmer has indicated that he wishes to see whether the value of an arithmetic expression (.03*PRINCIPAL) is less than a numerical literal (1.00). The asterisk is used to indicate multiplication. It is quite valid to incorporate an arithmetic expression within a conditional expression. For clarity it may often be desirable to use parentheses to denote the beginning and end of such an arithmetic expression.

```
01  INTEREST-CALCULATION.  IF .03 * PRINCIPAL IS LESS
02        THAN 1.00 GO TO END OTHERWISE MULTIPLY
03        PRINCIPAL BY 1.03 GIVING ACCOUNT BALANCE,
04        MOVE 'INTEREST' TO ACTION
```

Figure 17.

There is one other new point to notice in this example. On line 4 we have:

MOVE 'INTEREST' TO ACTION

The quotes indicate that the word INTEREST *itself* is to be moved to the area named ACTION. Thus, INTEREST is identified by the quotes as being an *alphameric literal*.

There are six verbs that handle all our input-output problems. In terms of the 1401, the verb ACCEPT calls for data to be read from cards. This would normally be written in the form ACCEPT data-name FROM CARD READER, where we would write the name of the card record for data-name. Printing and punching are handled by the DISPLAY verb. Here, we write the word DISPLAY followed by the names of the data to be printed, followed by the word UPON, followed by the word PRINTER or CARD PUNCH.

The verbs OPEN and CLOSE have the same meanings that they have in the Autocoder Input/Output Control System. That is, OPEN checks labels and positions tape to read the first tape block. CLOSE handles the trailer label and rewinds the tape.

The COBOL equivalent of the Input/Output Control System GET is called READ and performs exactly the same functions—that is, it makes available an input record either by advancing to a new record if records in the block remain to be processed or by actually reading a tape block if all the records in the input area have been processed. The routine compiled from the READ verb performs the same error checking and checks for the end of the file. We specify in the source program what should be done if the end of the file is reached, by writing the words AT END, followed by any imperative statement.

The COBOL equivalent of the Input/Output Control System PUT is called WRITE and performs the same function. We are not required in using READ and WRITE to be concerned with whether the object program will use indexing or a work area, and similar matters. All such considerations are handled by the COBOL processor.

Examples of these verbs appear in the program of the next subsection.

This has by no means been a complete exposition of the COBOL language. It is hoped, however, that this discussion, together with the extended example which follows, will provide some insight into the nature of COBOL programming and its advantages.

11.7 COBOL Program for Inventory Control Case Study

Figure 18 is a COBOL program to carry out the operation described in the block diagram of Figure 1 of Section 10. Since the program is written in English it is largely self-explanatory.

```
PROCEDURE DIVISION.
    OPEN INPUT OLD-MASTER-FILE OUTPUT NEW-MASTER-FILE.
    ACCEPT TRANSACTION-CARD FROM CARD-READER.
MASTER-READING.  READ OLD-MASTER-FILE RECORD AT END GO TO
    WRAPUP-TEST.
COMPARISON.  IF PART-NUMBER OF OLD-MASTER IS GREATER THAN
    PART-NUMBER OF TRANSACTION-CARD GO TO WRAPUP-TEST.
    IF PART-NUMBER OF OLD-MASTER IS EQUAL TO PART-NUMBER OF
    TRANSACTION-CARD GO TO CODE-TESTING-ROUTINE OTHERWISE
    GO TO MASTER-WRITING.
CODE-TESTING-ROUTINE.  GO TO RECOUNT, RECEIPT, ORDER, ISSUE
    DEPENDING ON TRANSACTION-CODE.  MOVE 'BAD CLASS
    CODE JOB HALTED' TO MESSAGE.
    DISPLAY MESSAGE ON PRINTER.  STOP 1.
RECOUNT.  MOVE TRANSACTION-QUANTITY TO QUANTITY-ON-HAND.
    GO TO REORDER-ROUTINE.
RECEIPT.  ADD TRANSACTION-QUANTITY, QUANTITY-ON-HAND.
    SUBTRACT TRANSACTION-QUANTITY, QUANTITY-ON-ORDER.
    GO TO REORDER-ROUTINE.
ORDER.  ADD TRANSACTION-QUANTITY, QUANTITY-ON-ORDER.
    GO TO REORDER-ROUTINE.
ISSUE.  IF QUANTITY-ON-HAND IS NOT LESS THAN TRANSACTION-
    QUANTITY GO TO SUBTRACTION-ROUTINE.  MOVE
    QUANTITY-ON-HAND TO MESSAGE-A.  MOVE PART-NUMBER OF
    TRANSACTION-CARD TO MESSAGE-B.  MOVE TRANSACTION-QUANTITY
    TO MESSAGE-C.  DISPLAY MESSAGE ON PRINTER.  MOVE
```

Figure 18. COBOL program for inventory control case study

```
    TRANSACTION-QUANTITY TO QUANTITY-ON-HAND.
SUBTRACTION-ROUTINE.  SUBTRACT TRANSACTION-QUANTITY,
    QUANTITY-ON-HAND.  MULTIPLY UNIT-PRICE BY TRANSACTION-
    QUANTITY GIVING TOTAL-PRICE.  ADD TOTAL-PRICE,
    YEAR-TO-DATE-SALES.
REORDER-ROUTINE.  IF QUANTITY-ON-HAND + QUANTITY-ON-ORDER
    IS GREATER THAN REORDER-POINT GO TO LAST-CARD-ROUTINE.
    MOVE PART-NUMBER OF TRANSACTION-CARD TO CARD-1.  MOVE
    MASTER-CODE TO CARD-2.  MOVE REORDER-QUANTITY TO CARD-3.
    DISPLAY CARD ON CARD-PUNCH.
LAST-CARD-ROUTINE.  IF LAST-CARD GO TO REPLACEMENT-ROUTINE.
    ACCEPT TRANSACTION-CARD FROM CARD-READER.  GO TO
    COMPARISON.
REPLACEMENT-ROUTINE.  MOVE HIGH-VALUE TO PART-NUMBER OF
    TRANSACTION-CARD.
    MASTER-WRITING.  WRITE NEW-MASTER FROM OLD-MASTER.  GO TO
    MASTER-READING.
WRAPUP-TEST.  IF LAST-CARD GO TO CLOSEOUT.  MOVE
    'FILE OR DATA ERROR JOB HALTED' TO MESSAGE.
    DISPLAY MESSAGE ON PRINTER.  STOP 2.
CLOSEOUT.  CLOSE OLD-MASTER-FILE, NEW-MASTER-FILE.  MOVE 'JOB
    FINISHED' TO MESSAGE.  DISPLAY
    MESSAGE ON PRINTER.  STOP 3.
```

Figure 18 Continued