SYSTEMS
ENGINEERING
SERVICES

PRELIMINARY MANUAL

1401

TABULAR PROGRAMMING SYSTEM

October 15, 1961
Technical Report No. 1 J 2

Thomas B. Glans
Burton Grad
William M. Selden

INTERNATIONAL BUSINESS MACHINES CORPORATION
White Plains, New York

1401 TABULAR PROGRAMMING SYSTEM

Preface

There is increasing evidence to indicate that decision tables are a useful means of analyzing and portraying program logic. With this in mind the 1401 RAMAC® Tabular Programming System was developed to encourage experiments using decision tables for a variety of business problems. The feedback from these experiments will provide a reliable evaluation of the usefulness of decision tables in problem analysis, program documentation, coding, and debugging.

Programming systems such as FARGO, Reports Generator and FORTRAN emphasize some particular aspect of data processing - inputs, outputs, mathematical equations, etc. But expressing complex decision logic has been largely ignored. While logical operations can of course be described, it is generally not in a convenient and efficient manner. Decision tables may fill this void.

A word about this teaching text. It is directed primarily at people who will be experimenting with the language. Since, many such people may be stronger in systems work than in programming, the presentation discusses programming matters more fully than an experienced programmer requires. However, this redundancy should not detract from the technical content of the manual.

Chapter 1 presents a quick sketch of the basic ideas of data processing and of decision tables without attempting to be complete. Chapter 2 covers all the essentials of tables and of the 1401 RAMAC Tabular Programming System, using a set of seven examples to introduce a few of the ideas at a time. There are also review questions and answers after each example. This chapter ends with three problems using tables. Chapter 3 contains a full-scale problem worked through in 1401 Tabular Programming language; it shows how the ideas can be applied to a specific situation. Chapter 4 is a reference section, describing the various language elements and their relationships.

We suggest that the person without extensive programming background would profit by reading the entire manual carefully, with special attention to Chapter 2. The experienced programmer will probably want to just skim Chapters 1 and 2, and turn rather quickly to a study of Chapters 4 and 3.

We wish to thank Messrs. E. L. Blossom, D. W. Hradecky, D. D. McCracken, L. R. Smith, and R. R. Smith, for their contributions to the concept and preparation of this text.

Two final comments:

(1) Because of time limitations and in light of the experimental purpose of the language many techniques that could have been included were left out. Furthermore, at this stage of language development many improvements--both minor and major--may need to be made. We anticipate that this initial version of the language will nevertheless be useful in evaluating the concept of tabular programming, and that along with the evaluation of the overall concept will come suggestions for improvement in the details. We welcome all such comments and suggestions.

(2) It should be understood that IBM makes no committment to maintain or improve this language. Also, though every reasonable attempt has been made to pretest the processor and to examine various implications of the language, we do not guarantee either the accuracy or efficiency of the processor, or the correctness of any results obtained with its use.

TABLE OF CONTENTS

Chapter 1 - A Tabular Form Language

Chapter 2 - Using the 1401 Tabular Programming System

Chapter 3 - Expense Report Program

Page

Chapter 4 - 1401 Tabular Programming System Specifications

Page

CHAPTER I

A Tabular Form Language

Suppose you have been assigned to a team that is to set up a data processing system for some application in payroll, or inventory control, or utility billing, or insurance, or even in an area of science or engineering. What do you need to know about data processing in order to use the Tabular Programming language on such a job?

To begin with, we can mention a few areas of knowledge that are not needed. You need no knowledge of electronics. You need no knowledge of mathematics beyond high school algebra (unless, of course, the problem itself is mathematical). With the Tabular Programming language you do not even need a detailed knowledge of how your particular computer system works. However, you do need to know certain facts about data processing, and eventually, if you work with the subject for a while, you will pick up certain detailed facts about your particular computer - but you do not need these now. For now, the general ideas which you should have are discussed below.

A data processing system is composed functionally of five parts, as shown in Figure 1. The input section accepts information "from the outside," and converts it into the electronic form in which it is manipulated and stored internally. Externally, information is typically recorded on punched cards, punched paper tape, or magnetic tape. In some applications, printed characters can be read directly. Presently-used business machines cannot recognize handwriting or speech. The output section of a computer has the obvious function of converting from the internal representation to some convenient external form, such as printing, punched cards, magnetic tape, punched paper tape, or a variety of

This introductory information is reprinted from the IBM Commercial Translator Manual; it also seems appropriate for this teaching text. "Tabular Programming " has been substituted for "Commercial Translator" throughout.

specialized media. Though the speeds of all of these devices are much greater than those of manual devices, they are still generally quite slow compared to speeds of internal electronic manipulation. The kind and number of input and output devices naturally depends on the particular machine and its application.

The storage section of a computer serves two important purposes. The obvious function is to hold the data on which we wish to operate. A function less obvious to the newcomer is to hold coded instructions which we place there to specify the procedure we wish to follow. A collection of such instructions is called a program. There are usually two types of storage. One type, though very fast, is of limited capacity and quite expensive; it is called main storage. What is frequently termed auxiliary storage can hold much more information, but is substantially slower.

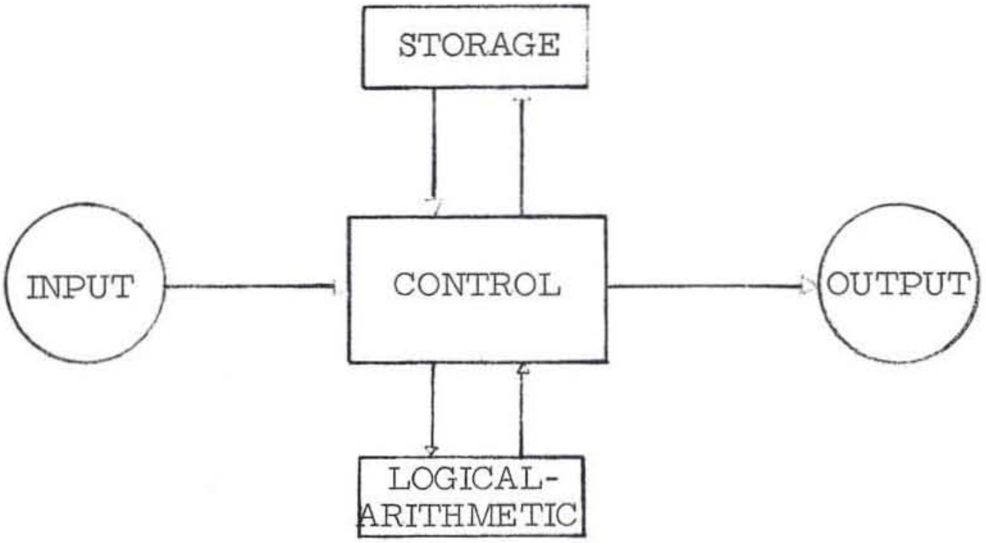


Figure 1. Schematic representation of the functional parts of a computer

The last two sections of a computer are called the logical-arithmetic section and the control section. The actual data processing is done in the logical-arithmetic section, and the control section is needed to decode and interpret the instructions in storage.

A most important feature of modern data processing machines is the way instructions are held in main storage right along with the data. For this reason we speak of a stored-program machine.

The instructions which a data processing machine can execute naturally vary from one machine to another, but they can still be grouped into general categories. One group is used for arithmetic operations, another for making the elementary "decisions" of which a data processing machine is capable. Still another group covers input/ output operations and a fourth group carries out miscellaneous control functions which are required because of the way the machine operates. Most individual operations are quite elementary, requiring a large munber of them be combined properly in order to carry out a meaningful data processing task. This work, which follows the complete definition of the processing task, is called programming.

Data processing requires an extremely precise statement of the problem. We must not say "less than 30" if we mean "less than or equal to 30." There is no way we can say, "make sure the data looks reasonable"; if we want to check the validity of data, we must specify exactly what tests are to be made on it.

With data processing, we are required to detail our procedures in advance to a degree not found in other methods. If we were asking a clerk to do a job, we might end by saying, "and if you run into anything you don't know how to handle, call me and we'll figure out what to do." In order to do a similar thing with a data processing machine, it is necessary first to define precisely what constitutes an exception, and then to write a procedure to handle it.

Decision Tables

Of the various activities that go into setting up a data processing procedure for a computer, the hardest is obtaining this precise definition of exactly what is to be done under all combinations of circumstances. Every step must be specified in detail. The conditions under which each action is to be performed must be enumerated. The exceptions to normal processing must be identified. The sequence in which operations are to be executed must be precisely stated.

Determining what is required of the computer system is called analysis; deciding just how to go about meeting these requirements is the general area of programming; communicating the chosen procedures to the computer is called coding. In each of these areas a language is needed for defining the data processing procedures. Ideally, a language should be suitable both for man-to-man communication and documentation, and for man-to-machine communication.

Many languages are used for these purposes. Procedures are often communicated to the machine in a form closely resembling the language of its own instructions, but this imposes a heavy and unnecessary burden on the person writing the procedures, since human language and the machine's language are quite different. Flow charts are widely used for man-to-man communication and documentation, but they can become confusing themselves in complex situations, and the flow chart language is not suitable for communication with the machine. Logical equations are sometimes used, but they do not display relationships in as graphical a form as we might wish, and furthermore most system designers do not find them to be a comfortable form of expression.

We propose decision tables as holding promise of meeting all of these needs of a computer language. Decision tables provide a graphical representation of complex decision procedures in a way that is easy to visualize. They show alternatives and exceptions in a much more explicit way than other languages. They present relationships among variables clearly. They show the sequences of conditions and actions in an unambiguous manner. The language of decision tables can be used equally effectively for system analysis, procedure design, and computer coding. Thus, a computer procedure written as a set of decision tables is, to a large extent, its own documentation.

There is a growing body of experience to indicate that these claims are justified. Those who have used decision tables say that programming is much faster and that program checkout time is significantly reduced. They say that the use of tables leads to greater accuracy and completeness in problem formulation. They say that program maintenance is simpler, and that a program written in tabular form is indeed a powerful communication and documentation device.

For all of these reasons, therefore, we suggest that you give decision tables serious consideration. Your experience will help in the evaluation of the concept, and will assist in improving its implementation in terms of processors to translate from tables to computer instructions.

The Structure of a Table

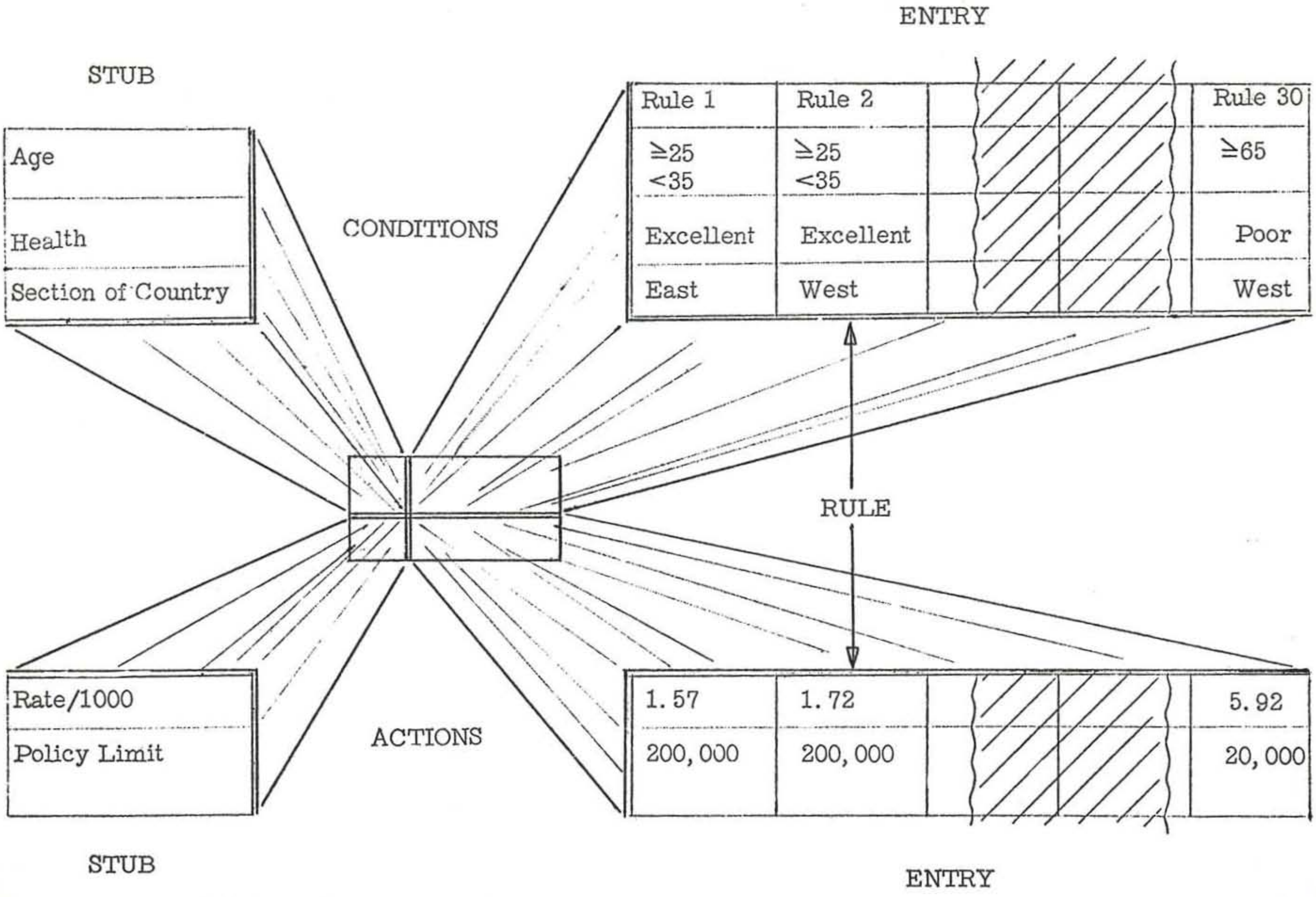
To begin to see what decision tables are all about, let's look at the simple table of Figure 2.

	Rule 1	Rule 2		Rule 30
Age	≥ 25 < 35	≥ 25 < 35		≥ 65
Health	Excellent	Excellent		Poor
Section of Country	East	West		West
Rate/1000	1.57	1.72		5.92
Policy Limit	200,000	200,000		20,000

Figure 2. An example of a decision table

The first decision rule (columns 1 and 2) can be paraphrased: If age is greater than or equal to 25 and less than 35, and health is excellent, and section of country is East, then rate per thousand is 1.57 and policy limit is 200,000. The underlined words are implied by the table layout. The other rules are alternatives to this one, so that logically, it doesn't matter which rule is examined first; only one rule can be satisfied in a single pass through this decision table.

The information in Figure 2 is shown in an exploded view in Figure 3, to show more clearly the parts of a table and the terms that are used to describe them.



Age
Health
Section of Country

Rule 1	Rule 2				Rule 30
≧25 <35	≧25 <35				≧65
Excellent	Excellent				Poor
East	West				West

Rate/1000
Policy Limit

1.57	1.72				5.92
200,000	200,000				20,000

Figure 3. Exploded view of the table of Figure 2.

The double lines serve as demarcation: CONDITIONS are shown above the horizontal double line, ACTIONS below; the STUB is to the left of the vertical double line, ENTRIES are to the right. Each vertical combination of conditions and actions is called a RULE. The essential nomenclature is completed by adding at the top of the table a title section, called a TABLE HEADER, and by adding a RULE HEADER over the entries.

Tables may be used in a slightly different way to state decision logic, as shown in Figure 4.

Rule No.	1	2	3	4
Credit Limit is OK	Y	N	N	N
Pay Experience is Favorable		Y	N	N
Special Clearance is Obtained			Y	N
Approve Order	X	X	X	
Return Order to Sales				X

Figure 4. A limited entry table

The first rule (columns 1 and 2) is read: If credit limit is OK then approve order. Again, the underlined words are implied by the form. You may note here than the form of the individual condition or action is somewhat different. In a limited entry condition or action, the entire condition or action must be written in the stub; the entry is limited to asserting, reversing, or ignoring a condition or executing an action. In contrast, an extended entry form (as in Figure 2) has part of the condition or action extended directly into the entry. Both forms may be used within one table, but any one condition or action row must be of just one form.

You may note from this example that the basic concept of a single rule in a table is quite straightforward, being based on the "if ... then" relationship. If A = B and C is greater than 5 and ... then assign the value 7 to X and GO TO Table 10. The alternative rules are implied; if the conditions in rule 1 are not met, then try rule 2. If none of the rules succeeds, then the implication is that something is wrong. Now if we have considered all significant possibilities, and we want to indicate, "Go ahead anyhow with a special routine", then we indicate "ELSE" in the last rule; otherwise, we provide for an automatic error routine. The flow chart of Figure 5 shows schematically the way in which a table is "executed."

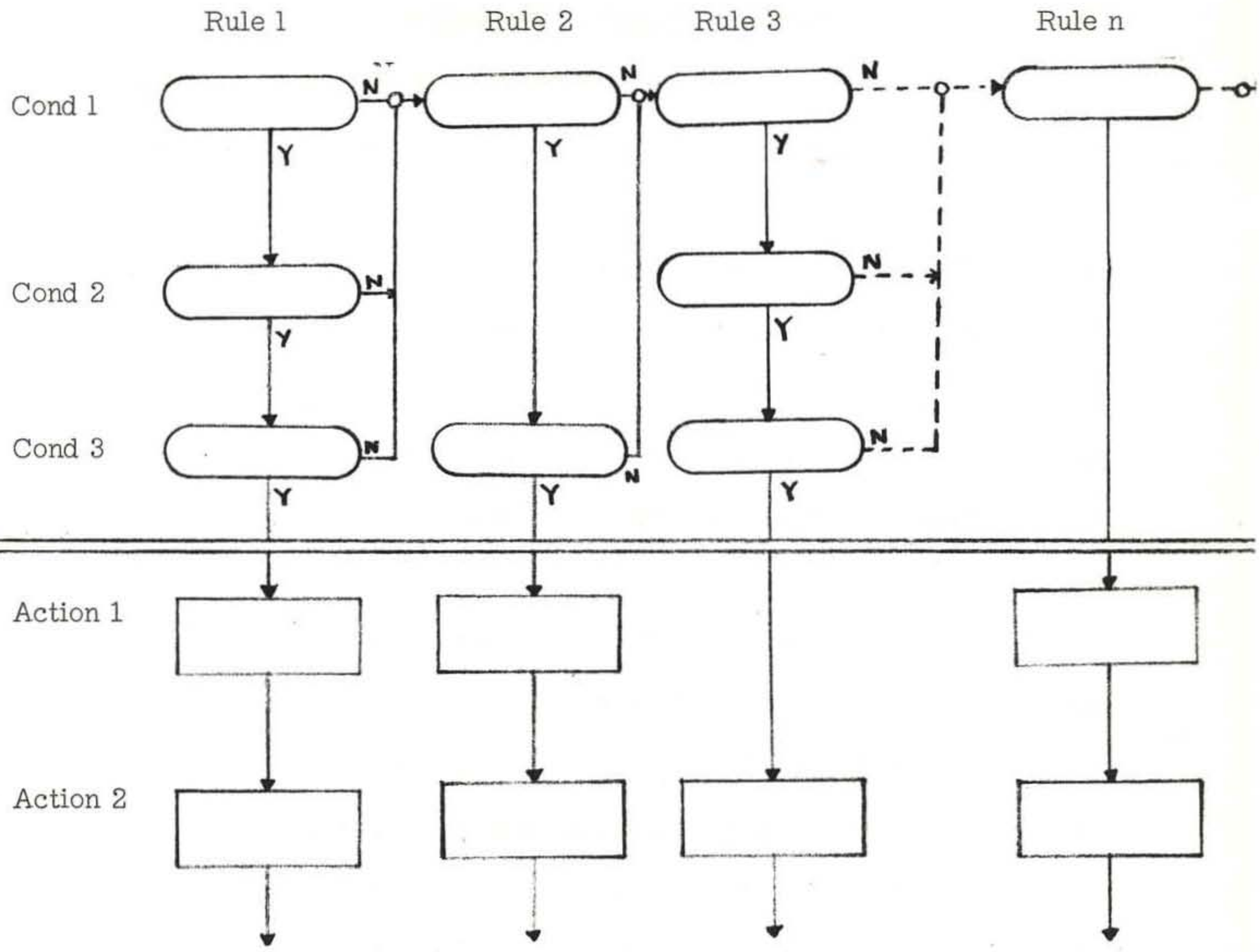


Figure 5. Schematic representation of the sequence of tests and actions in executing a table.

We shall see in the next chapter how it is possible to set up many tables so they will be executed in the proper sequence to carry out the necessary processing.

The 1401 Tabular Programming System Processor

We spoke of the use of decision tables as a language for expressing procedures to a computer. This is not literally true, of course: a table cannot be directly executed by a computer. It is necessary first to translate the decision tables into basic machine instructions. The set of tables that define the processing to be done by the computer make up the source program; this is translated into an object program -- consisting of actual machine instructions -- by a processor. It is the object program that finally directs the computer to carry out the desired processing of data.

The procedure for going from tables written on paper to final problem results is shown in Figure 6.

1. The first step is the one that is the primary subject of this text: planning the procedure and writing it as a set of tables.

2. Cards are punched and verified, producing the source program deck. The processor program, which is itself a large set of computer instructions, is also a deck of cards.

3. The processor program goes into the computer and directs the translation of the source program into an object program consisting of computer instructions. The object program is punched out as a deck of cards. There are also two listings produced by the processor, giving information about the object program.

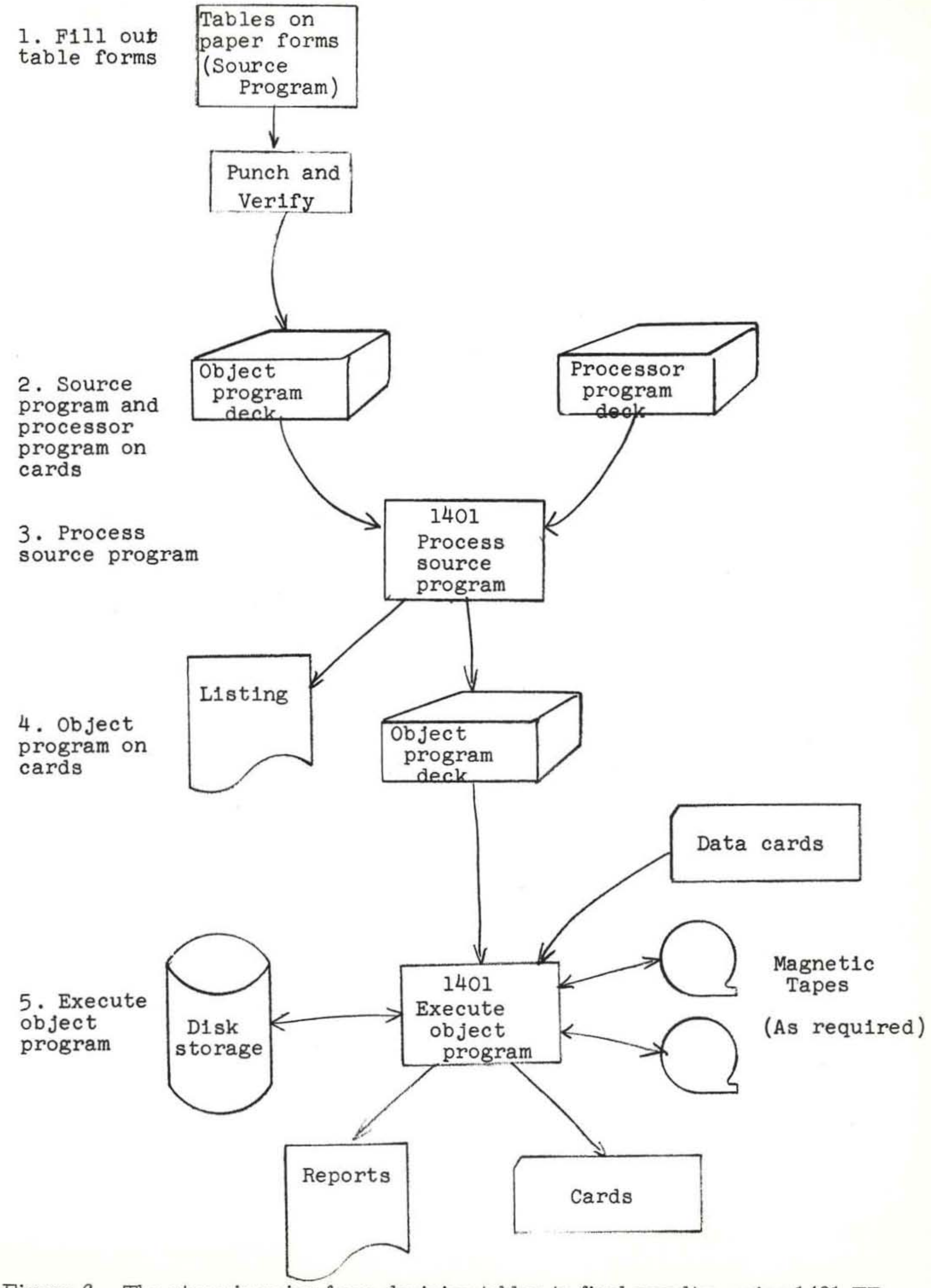

1. Fill out table forms
Tables on paper forms (Source Program)
Punch and Verify
2. Source program and processor program on cards
Object program deck
Processor program deck
3. Process source program
1401 Process source program
4. Object program on cards
Listing
Object program deck
Data cards
5. Execute object program
Disk storage
1401 Execute object program
Magnetic Tapes
(As required)
Reports
Cards

Figure 6. The steps in going from decision tables to final results, using 1401 TP.

The processing is in fact done in two steps: one to go from source program to an Autocoder program, (Autocoder is another computer language, a level above basic machine language.) and a second to go from Autocoder to machine instructions. Conceptually, however, the two parts are not essentially distinct. As a matter of practical operation, the programmer is ordinarily not even aware that an Autocoder stage is involved, if the computer details are done by an operator. Card handling procedures are described elsewhere.

4. The object program is now available as a deck of cards; it has not yet been executed.

5. The object program may now be loaded into the computer to do the data processing specified by the original tables. It is only at this stage that data cards are processed. Other computer components are used as required by the program.

The machine that executes the object program need not be the same one which processed the source program. Within certain limits, the two machines need not even have the same features. The machine configurations for each of these jobs are shown in Chapter 4.

SUMMARY

This chapter has presented the basic ideas of decision tables in rather broad outline. The next step is to learn how these ideas are implemented in the 1401 Tabular Programming language, which is done in the next chapter, using a series of examples. The first ones are quite simple; the later examples tie together the concepts introduced earlier. To help you check your understanding of the concepts in these examples, there are review questions following each example, with answers given for all questions.

CHAPTER 2

Using the 1401 Tabular Programming System

The basic ideas of the IBM 1401 Tabular Programming System are relatively easy to learn. This chapter presents these ideas in the framework of a series of problem situations, each chosen to illustrate a few new concepts. At the end of the discussion of each situation there are some review questions, with answers on the back of the same page, to permit the reader to check his understanding of the material before proceeding.

This chapter does not discuss all the features of the language. In particular it omits some of the details of alternative ways of doing things, and it omits details such as the maximum permissible sizes of fields. The complete information appears in Chapter 4. The reader may choose to refer to the appropriate parts of Chapter 4 as he studies the examples, or to give the current chapter a reading for the broad outline before becoming concerned with the details in Chapter 4. The latter course is recommended.

For a first example, consider a situation in which it is necessary to convert an automobile body classification from a code to a readable abbreviation.

OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
	TYPCØD	EQ			1		2		3
SET	MØDEL	EQ			'CNVTBL'		'HRDTØP'		'STAWGN'

Situation 1--Procedure Description Table

The table on the opposite page can be interpreted fairly readily. In English:

- If the type code of the car body is 1, then the model is a convertible.
- If the type code of the car body is 2, then the model is a hardtop.
- If the type code of the car body is 3, then the model is a station wagon.

In this table, there is one condition row (above the double horizontal line): The action to be carried out depends on whether TYPCOD = 1, 2, or 3. There is also one action row (below the double horizontal line): set MODEL equal to CNVTBL, HRDTOP, or STAWGN, depending on whether TYPCOD = 1, 2, or 3 respectively. Thus we see the basic idea of a table, that conditions and actions are always related: if a specified condition (or, more generally, a set of conditions) is satisfied, then the corresponding action (or actions) is executed. A vertical column containing one or more conditions above the double horizontal line and one or more actions below, is called a rule. Thus, Rule 1 reads:

If TYPCOD = 1, then SET MODEL EQ 'CNVTBL'

TYPCOD is an abbreviated name for the data field containing the type code; similarly, MODEL is the name of the field that will contain the English abbreviation for the body type. TYPCOD and MODEL are variable names; they represent variables consisting in this case of a one-digit code and a six-letter abbreviation respectively. Furthermore, TYPCOD and

OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
	TYPCØD	EQ			1		2		3
SET	MØDEL	EQ			'CNVTBL'		'HRDTØP'		'STAWGN'

Situation 1--Procedure Description Table

MODEL are names of operands--that is, information fields on which operations of one sort or another are performed.

In the action part of the table, SET...EQ is an action operator which describes operations to be performed on operands when the appropriate conditions are satisfied. In this case, the action is to transfer one of the abbreviations CNVTBL, HRDTOP, or STAWGN to the field named MODEL, depending on which condition is satisfied.

The words 'CNVTBL', 'HRDTOP', and 'STAWGN' are not variable names, but rather are literals, as indicated by the quotation marks. The distinction between a variable name and a literal is most important. Without the quotation marks, Rule 1 would read:

> If TYPCOD = 1, transfer the current value of the variable (data field) named CNVTBL to the data field named MODEL.

This clearly is not the same thing at all as moving the letters CNVTBL themselves to the data field named MODEL.

In fact, everything in this table to the right of the vertical double line (the entry part) is a literal: the 2, for instance, refers to the literal value 2, not to a variable named 2. Short numeric literals are not written with quotation marks since they could not be confused with variable names--which are not allowed to be pure numeric or even to begin with a digit. Alphameric literals, on the other hand, must be enclosed in quotes to distinguish them from variable names.

OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
	TYPCØD	EQ			1		2		3
SET	MØDEL	EQ			'CNVTBL'		'HRDTØP'		'STAWGN'

Situation 1--Procedure Description Table

OP	NAME	OP	NAME	OP	NAME	OP	NAME
	PAID	EQ			'N'		'G'
SET	PCHNAM	EQ			RDNAME		RDNAME
SET	PCHAMT	EQ			RDNET		RDGRS
WRITE					CDPCH		CDPCH
READ					CDREAD		CDREAD
GØ TØ					TAB006		TAB006

Situation 2--Procedure Description Table

This table describes a data processing procedure to be carried out, and is therefore called a Procedure Description Table.

If you can correctly answer the following questions, you are ready to proceed; otherwise, some review will help you to understand the later material more quickly. Answers are given on the back of this page.

QUESTIONS

1. List the variable names in situation #1.
2. List the literals.
3. What is the difference between a variable and a literal?
4. State Rule 3 in words.
5. What is an action operator?

Situation #2 involves a utility billing procedure. The bill stub that a customer returns with his payment has punched in it a net amount and a gross amount. Before the bill stub goes into the computer system, the letter N or G is punched into the card to indicate which amount was paid; it is then unnecessary to punch the amount of the payment.

In English, this table reads:

Rule 1: If the card field named PAID contains 'N' (for net), transfer the customer's name from the card input field named RDNAME to the card punching field named PCHNAM, transfer the field named RDNET to PCHAMT, write (punch) a card, read a card, and finally GO TO table 006.

Answers - Situation 1

1. TYPCOD, MODEL.

2. 1, 2, 3, 'CNVTBL', 'HRDTOP', 'STAWGN'.

3. A variable is referred to by name, whereas a literal has only its actual value, and does not refer to anything else.

4. If TYPCOD = 3, set the variable named MODEL equal to the letters STAWGN. Stated otherwise: If TYPCOD = 3, transfer the literal 'STAWGN' to the field named MODEL.

5. A command to perform some action on one or more operands.

OP	NAME	OP	NAME	OP	NAME	OP	NAME
	PAID	EQ			'N'		'G'
SET	PCHNAM	EQ			RDNAME		RDNAME
SET	PCHAMT	EQ			RDNET		RDGRS
WRITE					CDPCH		CDPCH
READ					CDREAD		CDREAD
GØ TØ					TAB006		TAB006

Situation 2--Procedure Description Table

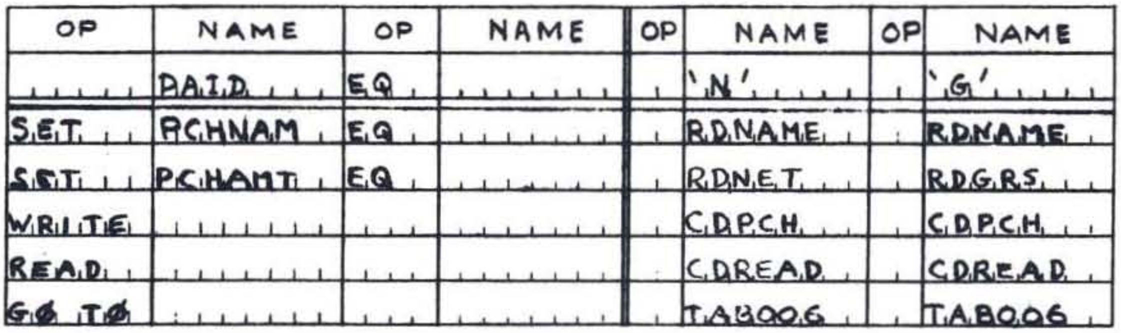
Rule 2: If the PAID field contains 'G' (for gross), carry out the same actions as for Rule 1, except transfer the gross amount from RDGRS to PCHAMT.

The condition row in this example is quite similar to that in the previous example, with just one difference: since the literals are alphabetic they must be enclosed in quotes.

The SET...EQ action operators in the first two action rows are the same as before. The WRITE operator, however, is new, requiring the introduction of some new terminology.

We have seen that a variable data field contains a value. A record is a collection of variables, and a file is a collection of one or more records associated with an input or output device. In this situation, WRITE CDPCH specifies writing a record from the file named CDPCH on the card punch, or, more specifically, punching a card containing the customer's name and the amount paid. To summarize: the WRITE operator requires a file name for its operand; the operator calls for the output of a record from this file.

A data description table is required in every program, to give the processor a variety of information about the variables, records, and files in a program. The data description table for this situation, for instance, would specify that CDPCH is the name of a file associated with the card punch. We shall investigate the contents of a data description table in a later situation, and see how to fill it out.



OP	NAME	OP	NAME	OP	NAME	OP	NAME
	PAID	EQ			'N'		'G'
SET	PCHNAM	EQ			RDNAME		RDNAME
SET	PCHAMT	EQ			RDNET		RDGRS
WRITE					CDPCH		CDPCH
READ					CDREAD		CDREAD
GØ TØ					TAB006		TAB006

Situation 2--Procedure Description Table

The READ CDREAD action, in combination with information supplied by the data description, means to read a record (card) in a file named CDREAD. This action reads in new values of the variables named PAID, RDNAME, RDNET, and RDGRS (as would be indicated in the data description).

The last action is another new one, GO TO. This causes a transfer of control to the table identified by the operand. In this case, we assume that the current table is table 006, so this same table is simply repeated with new values of the variables. A GO TO always transfers to the beginning of a table; there is no provision for executing just one rule of a table, for instance. The table to be executed at the beginning of a program must be given the number 001, since table 001 is automatically carried out first.

We are ignoring here the question of how to read the first card. This would be handled easily enough by an unconditional table, of which we shall see examples in later situations.

QUESTIONS

1. List the variable names in this table.
2. List the file names.
3. Can we tell, simply by inspecting this table alone, that RDNAME is a variable and CDREAD is a file?
4. What would happen if the READ and WRITE actions were interchanged?
5. What would happen if the last action were GO TO TAB941?

Answers - Situation 2

1. PAID, PCHNAM, RDNAME, PCHAMT, RDNET, RDGRS.

2. CDPCH, CDREAD.

3. Yes, but only by implication: the operand of a SET action cannot be a file name, and the operand in the READ and WRITE action must be a file name. The Data Description, if it had been shown, would of course have left no doubt.

4. The order of card reading and punching would be interchanged. Actions are always carried out in the sequence in which they appear in a table, from top to bottom.

5. Table 941, whatever it might be, would be executed next.

OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
	AGE			LE	25	LE	25	GR	25	GR	25
	SEX	EQ			'M'		'F'				
	ACDNTS							EQ	0	GR	0
SET	RATE			+	RSKFAC			-	SPRFAC		
SET	PURATE	EQ			RATE		RATE		RATE		RATE
WRITE	RATECD			X		X		X		X	

Situation 3--Procedure Description Table

Situation #3 involves an automobile insurance liability rate computation, and introduces several new ideas in table structure.

We have here our first example of a table in which more than one condition must be satisfied for the actions in a rule to be executed. The condition part of the first rule, for instance, asks, "Is the applicant male and less than or equal to 25 years of age?" The blank entry in Rule 1 for the accidents condition row means that the condition is not relevant for this rule: a man of 25 or under gets the risk factor RSKFAC added to his rate regardless of his driving record.

Notice that there is no combination of age, sex, and accident record that satisfied the conditions of more than one rule. This is a requirement of the system: the conditions must be set up so that at most one rule is carried out each time a table is executed.

This table introduces three of the relational operators: EQ, LE, GR. The six available relational operators and their abbreviations are:

Equal	EQ	Less than (lesser)	LR
Unequal	UN	Greater than or equal to	GE
Greater than	GR	Less than or equal to	LE

These operators are to be understood in the sense of a question to which the answer can be only yes or no. The first condition in Rule 1 thus reads: "Is the age of the applicant less than or equal to 25 years?" If the answer is "yes," the condition is satisfied. This way of viewing relational operators allows a simplification of the writing of conditions, as we shall see in a later situation.

OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
	AGE			LE	25	LE	25	GR	25	GR	25
	SEX	EQ			`M'		`F'				
	ACDNTS							EQ	0	GR	0
SET	RATE			+	RSKFAC			-	SPRFAC		
SET	PURATE	EQ			RATE		RATE		RATE		RATE
WRITE	RATECD			X		X		X		X	

Situation 3--Procedure Description Table

The SET operator has a new twist here: it is used as a two-address arithmetic operator. It is assumed that the RATE has already been computed in a previous table; the table here merely applies special factors if necessary. Rule 1 then says (if the conditions are satisfied):

Set RATE = RATE + RSKFAC

In other words, the risk factor is added to the rate already in core storage. In Rule 2 there is no second operand for this action, so nothing is done to the rate; likewise in Rule 4. In Rule 3, the action specified is to subtract a special rate factor from the previously computed rate. Note that the EQ is omitted when SET is used to perform arithmetic.

The second action row contains nothing new. The rate, as modified by now if Rule 1 or 3 is being executed, is moved to PURATE, a field in the punch area of storage.

The third action is new, but only in format. Since the same action is to be carried out for each rule, we write the complete action specification in the stub part of the table (to the left of the vertical double line) and place X's in the entry part to indicate in which rules the action is to be executed. This is called limited entry; the format of the previous tables is called extended entry, since the operand is "extended" into the entry portion of the table. The second action could also have been written in limited entry fashion, with SET PURATE EQ RATE entirely in the stub, and X's in the "OP" fields of the entries. If an action is not to be executed for a particular rule then the X should be omitted.

OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
	AGE			LE	25	LE	25	GR	25	GR	25
	SEX	EQ			'M'		'F'				
	ACDNTS							EQ	0	GR	0
SET	RATE			+	RSKFAC			-	SPRFAC		
SET	PURATE	EQ			RATE		RATE		RATE		RATE
WRITE	RATECD			X		X		X		X	

Situation 3--Procedure Description Table

OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
	STKTYP	EQ			01		02		03		ELSE
	SHARES	GR	1000000	Y				N			
SET	TYPE	EQ			'INDIV'		'BANK'		'BRØKER'		
MØVE	STKVAL	TØ			VALUE1		VALUE2		VALUE2		
SPACE	2LINES			X		X		X			
WRITE	REPØRT			X		X		X			
GØ TØ											TAB123

Situation 4--Procedure Description Table

The SET operator has a new twist here: it is used as a two-address arithmetic operator. It is assumed that the RATE has already been computed in a previous table; the table here merely applies special factors if necessary. Rule 1 then says (if the conditions are satisfied):

Set RATE = RATE + RSKFAC

In other words, the risk factor is added to the rate already in core storage. In Rule 2 there is no second operand for this action, so nothing is done to the rate; likewise in Rule 4. In Rule 3, the action specified is to subtract a special rate factor from the previously computed rate. Note that the EQ is omitted when SET is used to perform arithmetic.

The second action row contains nothing new. The rate, as modified by now if Rule 1 or 3 is being executed, is moved to PURATE, a field in the punch area of storage.

The third action is new, but only in format. Since the same action is to be carried out for each rule, we write the complete action specification in the stub part of the table (to the left of the vertical double line) and place X's in the entry part to indicate in which rules the action is to be executed. This is called limited entry; the format of the previous tables is called extended entry, since the operand is "extended" into the entry portion of the table. The second action could also have been written in limited entry fashion, with SET PURATE EQ RATE entirely in the stub, and X's in the "OP" fields of the entries. If an action is not to be executed for a particular rule then the X should be omitted.

OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
	AGE			LE	25	LE	25	GR	25	GR	25
	SEX	EQ			'M'		'F'				
	ACDNTS							EQ	0	GR	0
SET	RATE			+	RSKFAC			-	SPRFAC		
SET	PURATE	EQ			RATE		RATE		RATE		RATE
WRITE	RATECD			X		X		X		X	

Situation 3--Procedure Description Table

OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
	STKTYP	EQ			01		02		03		ELSE
	SHARES	GR	100000	Y				N			
SET	TYPE	EQ			'INDIV'		'BANK'		'BRØKER'		
MØVE	STKVAL	TØ			VALUE1		VALUE2		VALUE2		
SPACE	2LINES			X		X		X			
WRITE	REPØRT			X		X		X			
GØ TØ											TAB123

Situation 4--Procedure Description Table

The limited entry technique provides no capabilities not available with extended entry, but the simplication of the table is often a convenience. The next illustration shows how limited entry can also be used in writing conditions.

QUESTIONS

1. State Rules 3 and 4 in English.
2. List the literals.
3. List three variable names in the condition area, and two in the action area of the stub.
4. Does the blank in action 1, Rule 2, mean that RATE is set equal to zero?
5. Could action 2 have been written in limited entry form? What would this have gained?

Situation #4 example uses limited entries in another way, and introduces two new action operators. The application is the preparation of a stock classification listing.

This table again exhibits multiple conditions, but this time one of them is is in limited entry form. The second condition stub asks: "Is the number of shares greater than 100,000?" To satisfy Rule 1, the answer must be yes. In Rule 2, with its blank for this condition, it doesn't matter. In Rule 3, the answer must be no in order to satisfy the condition--that is, the number of shares must be equal to or less than 100,000. (Note that numeric literals in a table must not contain commas.)

Answers - Situation 3

1. Rule 3: If the applicant is over 25 and has had no accidents, regardless of sex, subtract the special rate factor from the previously computed rate, move the new rate to the punch area, and punch a card.

Rule 4: If the applicant is over 25 and has had any accidents, regardless of sex, the previously computed rate is used unchanged. The rate is moved to the punch area and a card is punched.

2. 'M', 'F', O, 25.

3. ACDNTS, SEX, AGE: RATE, PURATE. (RATECD is a file name, by implication: the operand of a WRITE operator must be a file name.)

4. No. A blank in an action operand field means that no action is taken.

5. Yes, saving a little writing and making it apparent at a glance that the same action is done in each rule.

OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
	STKTYP	EQ			01		02		03		ELSE
	SHARES	GR	1,00000	Y				N			
SET	TYPE	EQ			'INDIV'		'BANK'		'BRØKER'		
MØVE	STKVAL	TØ			VALUE1		VALUE2		VALUE2		
SPACE	2LINES			X		X		X			
WRITE	REPØRT			X		X		X			
GØ TØ											TAB123

Situation 4--Procedure Description Table

The three rules here do not exhaust all the combinations of the variables in the conditions. For instance, the stock type might not be 1, 2, or 3, or even if the stock type is 1 the number of shares might not be greater than 100,000. In this table we have provided for such possibilities with a special "condition" called ELSE. This rule says what to do if none of the other rules can be executed; here we simply say to go to another table to handle this possibility. In this example, it is reasonable to expect that it will frequently happen that none of the other rules will be satisfied, so that the ELSE is a more or less normal occurrence. In other tables, failure to satisfy any of the rules might represent an error in the logic of the table or in the data. In such a case we might not provide an ELSE; now, if no rule is satisfied, the program proceeds to the error table named in the table header, which we consider in situation #6.

The first action operator in the procedure table is a familiar one, in this case setting a print field equal to an alphameric literal.

The second action operator is a new one, MOVE... TO. This is the opposite of SET... EQ: It moves the quantity specified by the first operand to the field named in the second operand. The difference in usefulness of the two operators may be stated as follows: SET... EQ is valuable when one of several quantities in the second operand field is to be moved to the first operand field; MOVE... TO is valuable when a quantity in the first operand field is to be transferred to one of several fields named by the second operand.

OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
	STKTYP	EQ			01		02		03		ELSE
	SHARES	GR	100000	Y				N			
SET	TYPE	EQ			‘INDIV’		‘BANK’		‘BRØKER’		
MØVE	STKVAL	TØ			VALUE1		VALUE2		VALUE2		
SPACE	2LINES			X		X		X			
WRITE	REPØRT			X		X		X			
GØ TØ											TAB123

Situation 4--Procedure Description Table

9	10 15	16 18	19 20	21	22	23 32	33 42	43
CLASS	NAME	INTEGER	DEC	CODE	CONT			DESCRIPTION
V	STKTYP	002	00	N				
V	SHARES	008	00	N				
V	STKVAL	008	02	N				
V	TYPE	006	00	A				
V	VALUE1	014	00	M	‘$, ,	0 . ’		
V	VALUE2	014	00	M	‘$, ,	0 . ’		

Situation 4--Data Description Table (incomplete)

The third operator is new, but fairly obvious. SPACE 2LINES causes an immediate space of two lines. One, two, or three lines may be spaced with the space operator. Spacing is independent of the WRITE operator--that is, in 1401 terms, an immediate space is performed, rather than an after-print space. This action, and the writing that follows it, are in limited entry form.

The data description for the variables in this table introduces some additional processing that is not evident from a study of the procedure table alone. Consider the variable descriptions shown in the data description for Situation #4. The first three describe the variables STKTYP, SHARES, and STKVAL. For each, the V establishes the description as that of a variable. The N means that the variable is numeric. For STKTYP, the integer length is 2 and the decimal length is zero, meaning that the quantity has two places to the left of the decimal point and zero places to the right. In other words, STKTYP is defined as a two-digit integer. In a similar fashion, SHARES is defined as an eight-digit integer. STKVAL is defined to have eight places to the left of the decimal point and two to the right.

The other three variable descriptions are for the variables in the printed report. TYPE is established by the code of A as an alphameric variable. For VALUE 1 and VALUE 2 there is a code of M, which stands for masked, and the masks themselves are written in the description field with quotes. Masks are used to insert punctuation and to delete

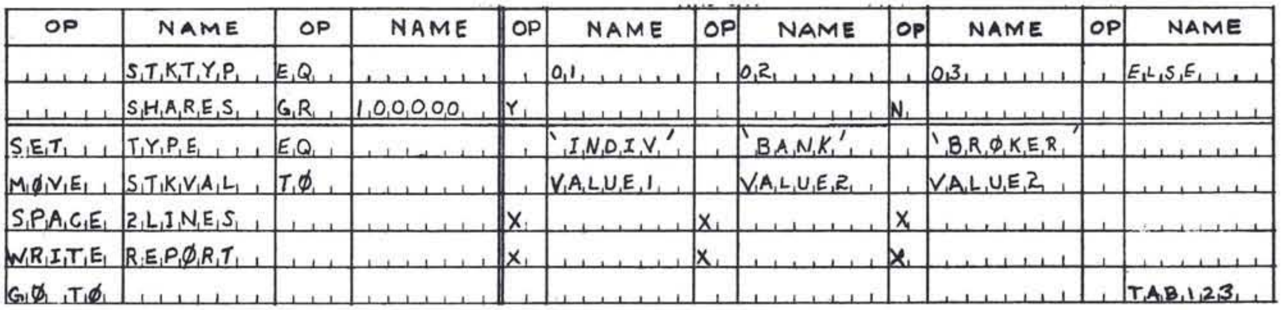
9	10 15	16 18	19 20	21	22	23 32 33 42 43
CLASS	NAME	INTEGER	DEC	CODE	CONT	DESCRIPTION
V	STKTYP	002	00	N		
V	SHARES	008	00	N		
V	STKVAL	008	02	N		
V	TYPE	006	00	A		
V	VALUE1	014	00	M		`$, , 0 . '
V	VALUE2	014	00	M		`$, , 0 . '

Situation 4--Data Description Table (incomplete)

leading zeros before printing. Masked variables are handled in a special way by the object program: When a quantity is transferred to a field that has been designated in the data description as masked, the mask is first placed in the field and then the quantity is moved. This double action gives an edited field as an end result. Observe that the use of a mask is specified in the <u>data</u> description; there is nothing in the procedure description to indicate that masking is to be done.

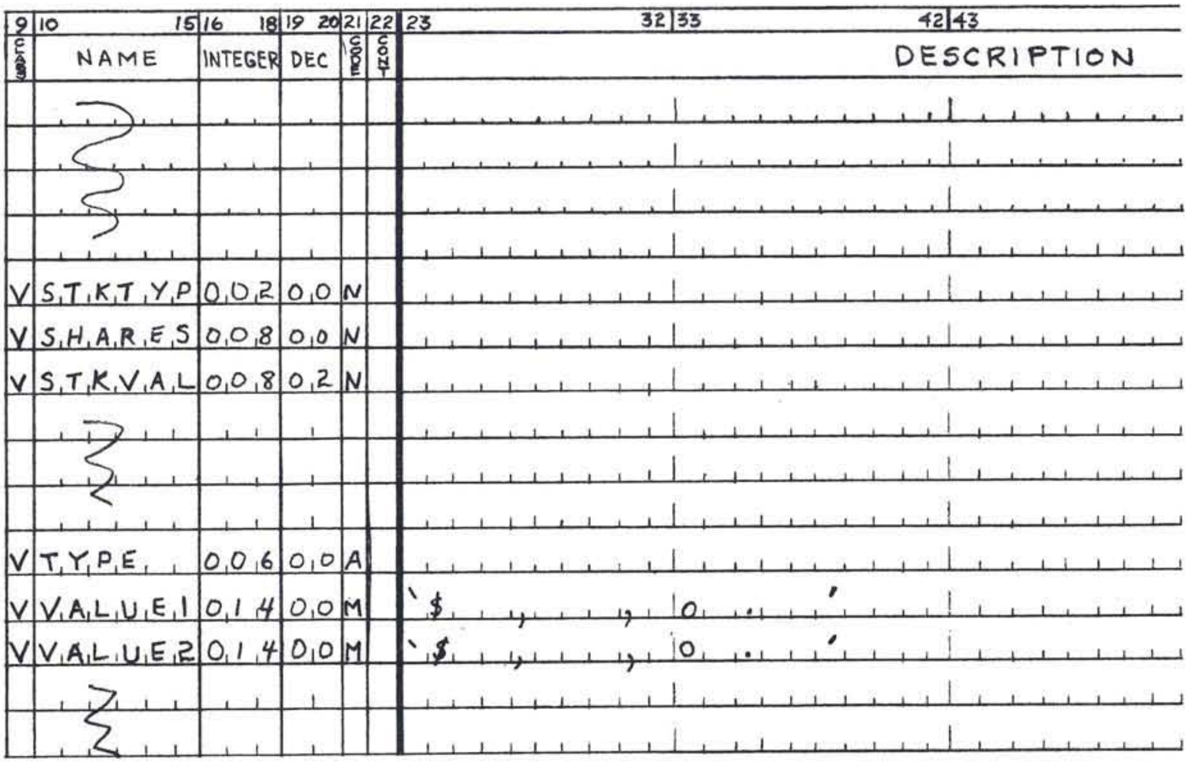
A more complete explanation of how masking operates may be found in Chapter 4 of this manual. For a quick indication of what masking does, we may note how a few numbers would be printed under control of the mask in this situation.

Number	Printed as
1234567899	$12, 345, 678. 99
0012345678	$ 123, 456. 78
0000001234	$ 12. 34
0000000000	$ 0. 00



OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
	STKTYP	EQ			01		02		03		ELSE
	SHARES	GR	100000	Y				N			
SET	TYPE	EQ			'INDIV'		'BANK'		'BRØKER'		
MØVE	STKVAL	TØ			VALUE1		VALUE2		VALUE2		
SPACE	2LINES			X		X		X			
WRITE	REPØRT			X		X		X			
GØ TØ											TAB123

Situation 4--Procedure Description Table



9	10 15	16 18	19 20	21	22	23 32 | 33 42 | 43
CLASS	NAME	INTEGER	DEC	CODE	CONT	DESCRIPTION
V	STKTYP	002	00	N		
V	SHARES	008	00	N		
V	STKVAL	008	02	N		
V	TYPE	006	00	A		
V	VALUE1	014	00	M		'$, , 0 .'
V	VALUE2	014	00	M		'$, , 0 .'

Situation 4--Data Description Table (incomplete)

QUESTIONS

1. State Rules 2 and 3 in words.
2. What is the difference between the SET and MOVE actions?
3. What is a mask, and how is a mask specified?
4. State the distinction between a procedure description table and a data description table.
5. What is meant by an ELSE condition?
6. What three letters can be used in the "code" column of a variable description, and what do they mean?
7. Must there be a Y or N in each condition entry space of a limited entry condition?

Situation # 5 requires the preparation of a listing and summary from information on a magnetic tape. Each tape record contains the following fields.

Field	Length	Type
Customer number	6	Numeric
Part number	6	Alpha
Description	17	Alpha
Quantity Sold	5	Numeric
Total Price	6	Numeric

The tape records are in ascending sequence on customer number.

Answers - Situation 4

1. Rule 2: If the stock type is 02, set TYPE equal to 'BANK', move stock value to VALUE2, space two lines on the printer, and print the record named REPORT.

Rule 3: If the stock type is 03 and the number of shares is not greater than 100,000 set TYPE equal to 'BROKER', move the stock value to VALUE2, space two lines, and print.

2. SET transfers the value of the second operand to the first; MOVE transfers the first to the second.

3. A mask specifies the insertion of punctuation and the deletion of leading zeros (plus certain other operations not discussed here). That a variable is masked is denoted in the variable description by a code of M; the mask itself is also entered in the data description.

4. The procedure description table specifies actions to be carried out, as selected by stated conditions; the data description table specifies the characteristics of the data, intermediate values, and results.

5. It designates the rule to execute if no other rule is satisfied. ELSE must therefore be the last-tested condition.

6. N for numeric, A for alphameric, and M for masked.

7. No; it may be blank.

9	10 15	16 18	19 20	21	22	23 32	33 42	43
CLASS	NAME	INTEGER	DEC	CODE	CONT			DESCRIPTION
F	TINPUT	040	01	M		TAPE1 IN, EØF TAB123		
R		040	05					
V	TCUST	006	00	N				
V	TPART	006	00	A				
V	TDESC	017	00	A				
V	TQTY	005	00	N				
V	TPRICE	004	02	N				

Situation 5--Data Description Table

A report is to be printed, with the information in each tape record becoming one line on the report. The items are to be spread out for readability, and the price edited. After printing all the records for one customer, one line will be spaced, the total price of goods sold to that customer printed, and the paper advanced to the top of the next page. The process is then repeated for the tape records of the next customer.

Two external files are involved in this application, one associated with a tape unit and one with the printer. Two other "files" are set up in core storage, to meet a requirement that every variable be a member of a file.

The first line of the data description table is the file description for the input data, which is assumed to be on tape unit 1. Reading across the line, the meanings of the various entries are:

F Identifies this description as that of a file.

INPUT The name given to the file.

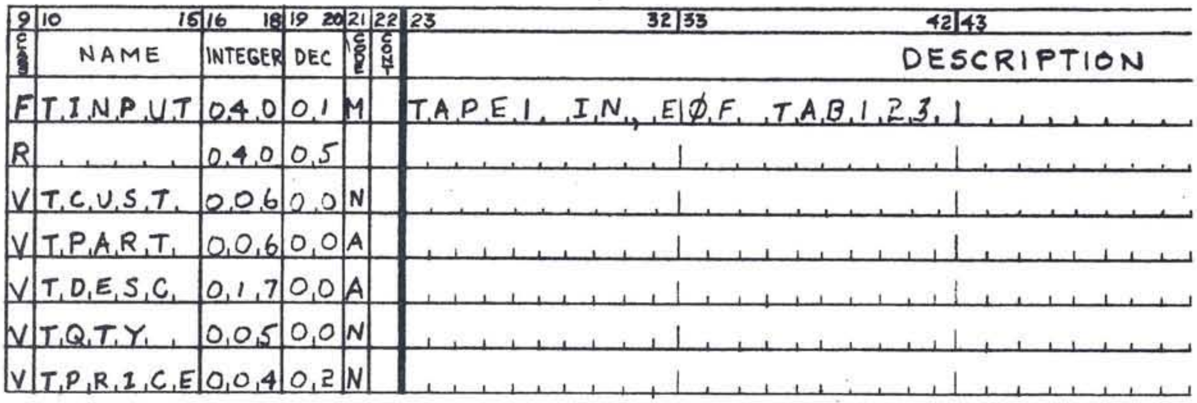
040 The number of characters in the longest record (the <u>only</u> record, in this example).

01 The number of types of records in the file.

M Reading is in the Move Mode - a 1401 detail.

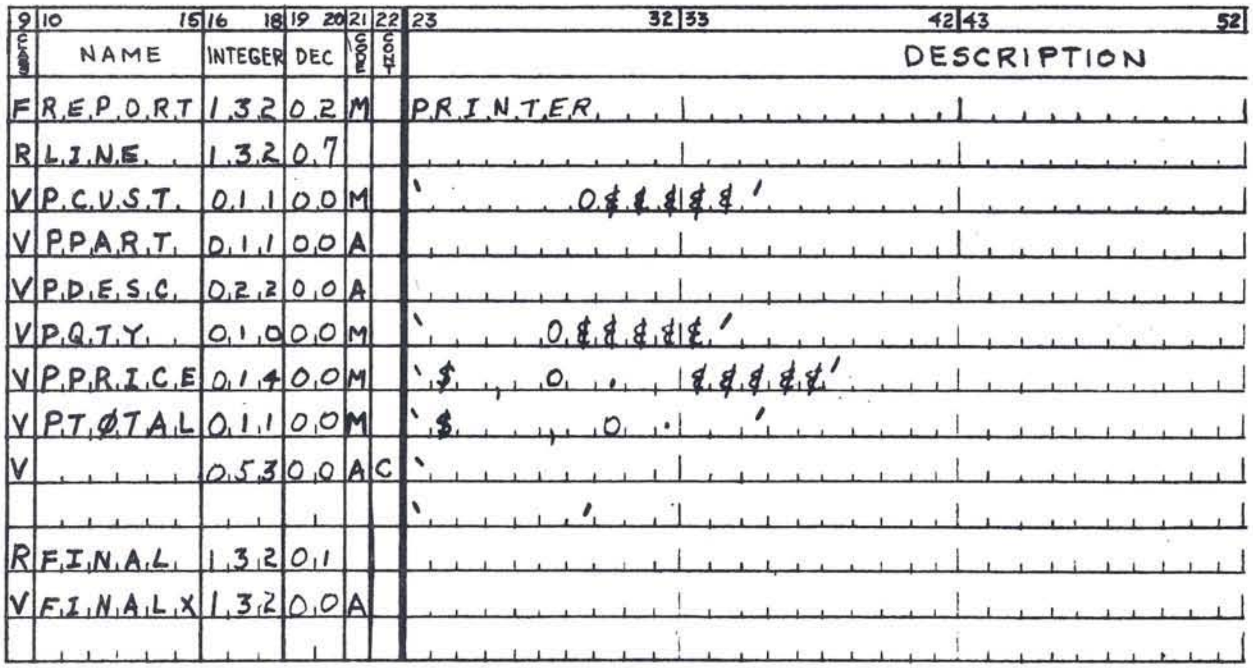
TAPE 1 IN The tape is mounted on tape unit #1, and this is an input file.

EOF TAB123 When the end-of-file indication (tape mark) is reached, go to table 123.



9	10 15	16 18	19 20	21	22	23 32 33 42 43
CLASS	NAME	INTEGER	DEC	CODE	CONT	DESCRIPTION
F	TINPUT	040	01	M		TAPE1 IN, EØF TAB123
R		040	05			
V	TCUST	006	00	N		
V	TPART	006	00	A		
V	TDESC	017	00	A		
V	TQTY	005	00	N		
V	TPRICE	004	02	N		

Situation 5--Data Description Table



9	10 15	16 18	19 20	21	22	23 32 33 42 43 52
CLASS	NAME	INTEGER	DEC	CODE	CONT	DESCRIPTION
F	REPORT	132	02	M		PRINTER
R	LINE	132	07			
V	PCUST	011	00	M		`'     0¢¢¢¢¢'`
V	PPART	011	00	A		
V	PDESC	022	00	A		
V	PQTY	010	00	M		`'    0¢¢¢¢¢'`
V	PPRICE	014	00	M		`'$ , 0 .  ¢¢¢¢¢'`
V	PTØTAL	011	00	M		`'$   , 0 .  '`
V		053	00	A	C	`'`
						`'     '`
R	FINAL	132	01			
V	FINALX	132	00	A		

Situation 5--Data Description Table, continued

The second line is a record description for the one type of record in the file. The name of the record is left blank here - this record is never referred to by name. The record contains 040 characters and 05 variables.

The variable descriptions are not greatly different from those we have seen before. All the names have been chosen to start with a T, to remind us that they are part of the tape record. This will make clearer the distinction between the variables in the tape record and those in the printer record when studying the procedure description.

The printer file description and the associated record and variable descriptions are similar to those for the tape. The main difference is that masking is used extensively, partly to insert punctuation and suppress leading zeros, and partly to obtain extra space at the right of the printing fields; this latter is the function of the ampersands (&) in the masks. To obtain extra space for the alphameric variables, which cannot be masked, the field sizes are made five characters longer than the variables to be printed.

Notice that after the variable named PTOTAL there is a variable with no name. This is called a preassigned variable since there is a "value" given for it, although in this case the value consists only of the 53 blanks required to fill out the printing line to 132 characters. Observe that the blanks are enclosed in quotes, and that a "C" appears in the continuation column of the first line to indicate that the value is continued to a second line.

9	10 15	16 18	19 20	21	22	23 32 33 42 43 52 53 62 63 72
CODE	NAME	INTEGER	DEC	MODE	CONT	DESCRIPTION
F		052	01	M		CØRE
R		052	03			
V	CUST	006	00	N		
V	TØTAL	004	02	N		
V	MESSAG	040	00	A		‘JOB FINISHED.. SAVE TAPE ØN TAPE UNIT 1.’
F		132	01	M		CØRE
R	BLANKR	132	01			
V	BLANKV	132	00	A	C	‘ ’
					C	‘ ’
						‘ ’

Situation 5--Data Description Table, continued

OP	NAME	OP	NAME	OP	NAME	OP	NAME
	TCUST	EQ	CUST	Y		N	
MØVE	BLANKR	TØ	LINE			X	
SET	PTØTAL	EQ	TØTAL			X	
SPACE	1 LINE					X	
WRITE	REPØRT					X	
MOVE	BLANKV	TØ	PTØTAL			X	
SPACE	01CHAN					X	
SET	PCUST	EQ	TCUST	X		X	
SET	PPART	EQ	TPART	X		X	
SET	PDESC	EQ	TDESC	X		X	
SET	PQTY	EQ	TQTY	X		X	
SET	PPRICE	EQ	TPRICE	X		X	
WRITE	REPØRT			X		X	
SET	TØTAL			+	TPRICE	EQ	TPRICE
SET	CUST	EQ	TCUST			X	
READ	TINPUT			X		X	
GØ TØ	TAB050			X		X	

Situation 5--Procedure Description Table

This file description contains a second type of record, consisting also of 132 characters. This record is used to contain a line of instructions to the computer operator, that is printed at the completion of the job.

The third and fourth file descriptions are for core storage "files." These are not files in the same sense as the others, since they are not associated with any input or output device; they are set up this way for consistency, and to satisfy the requirement that every variable be a member of a file. Notice that the two files are not named, and that the record in the first file is not named. The second file and record have been set up to handle a situation that arises in the procedure description, to which we now turn.

The procedure description should not be difficult to follow, with the file and variable arrangments now clearly in mind. Recall that we must list each tape record and print a summary line whenever we reach the end of the group of records for one customer. The only condition, therefore, is whether or not the tape record most recently read is for the same customer as the previous one. If it is, we move the information from the tape record to the print record, print it, add the price to the total, read another tape record, and repeat (go to) this table.

If the customer number in the tape record is not the same as the previous customer number, we prepare to print the total for the previous

OP	NAME	OP	NAME	OP	NAME	OP	NAME
	TCUST	EQ	CUST	Y		N	
MØVE	BLANKR	TØ	LINE			X	
SET	PTØTAL	EQ	TØTAL			X	
SPACE	1LINE					X	
WRITE	REPØRT					X	
MOVE	BLANKV	TØ	PTØTAL			X	
SPACE	01CHAN					X	
SET	PCUST	EQ	TCUST	X		X	
SET	PPART	EQ	TPART	X		X	
SET	PDESC	EQ	TDESC	X		X	
SET	PQTY	EQ	TQTY	X		X	
SET	PPRICE	EQ	TPRICE	X		X	
WRITE	REPØRT			X		X	
SET	TØTAL			+	TPRICE	EQ	TPRICE
SET	CUST	EQ	TCUST	X		X	
READ	TINPUT			X		X	
GØ TØ	TAB050			X		X	

Situation 5--Procedure Description Table

customer. In order to do this, it is necessary first to clear out the previous detail line contents. This is done here by moving a record consisting of all blanks to the record in the printer file. We see that the MOVE...TO operator applies to records as well as variables; this operator, SET...EQ..., and LAYOUT are the only three that do apply to records. We next put the total for the previous customer group in the print record, space a line, print, blank out this total so it won't print in the following detail lines, and skip to the top of a new page. This last is done with the SPACE operator, using an operand that refers to the carriage control tape on the printer. When all this has been done, the first record in the next customer group is still in the tape record area, and has not yet been processed. We therefore go ahead and process it just as before, with one difference: instead of adding the price to the total, we set the total equal to the price. This has the effect of discarding the previous total, which has by now been printed and is no longer of any use.

One table remains: the one that is executed when the end-of-file condition is detected on the tape, indicating that no more records follow. In the file description for the tape we specified that the end-of-file table (EOF) was 123, which will automatically be executed on a GO TO basis. When this happens, where are we? Looking back at the main table we can see that the last detail line will have been

OP	NAME	OP	NAME	OP	NAME
MØVE	BLANKR	TØ	LINE	X	
SET	PTØTAL	EQ	TØTAL	X	
SPACE	1LINE			X	
WRITE	REPØRT			X	
CLØSE	TINPUT			X	
SPACE	01CHAN			X	
LAYØUT	FINAL			X	
SET	FINALX	EQ	MESSAG	X	
WRITE	REPØRT			X	
SPACE	01CHAN			X	
GØ TØ	TABSTØP			X	

Situation 5--End-of-file Procedure Description Table printed, but not the total for the last group. In the EOF table, therefore, we begin by setting up and printing the total. Next we CLOSE the tape file, which rewinds the tape. After this we skip to the top of a new page, and prepare to write a line of instructions to the computer operator. Now we face a problem, however: we would like to move a whole line of comments, with no regard for the previous arrangement of fields but the previous word marks are in the way. To reset word marks for the new record, we use the LAYOUT operator, naming the new record. This clears previous word marks and sets the new ones. Now we can write the message, feed the paper, and STOP.

This is an example of an unconditional table, since it has no conditions. Such a table can obviously have only one rule, with the actions simply being carried out in sequence.

QUESTIONS

1. Must every file, record, and variable be given a name?
2. Are TPRICE and PPRICE the same variable?
3. What kind of operand is required with the LAYOUT operator? What does the operator do?
4. How do we indicate the procedure to be followed on an end-of-file condition?
5. What is an unconditional table?

Answers - Situation 5

1. No. Input and output files ordinarily have names, since the READ and WRITE operators require file names, but names may not be required for records (frequently) and variables (occasionally).

2. No. From the standpoint of the application they are the same quantity, but in terms of the Tabular Programming System they are entirely different: different names, different storage locations, different records, and different formats (since PPRICE is masked).

3. A record name. The core area for the file of which the record is a member is cleared of word marks and the proper word marks for the new record are set.

4. By writing the end-of-file table number in the description area of the file description.

5. A table with no conditions and only one rule.

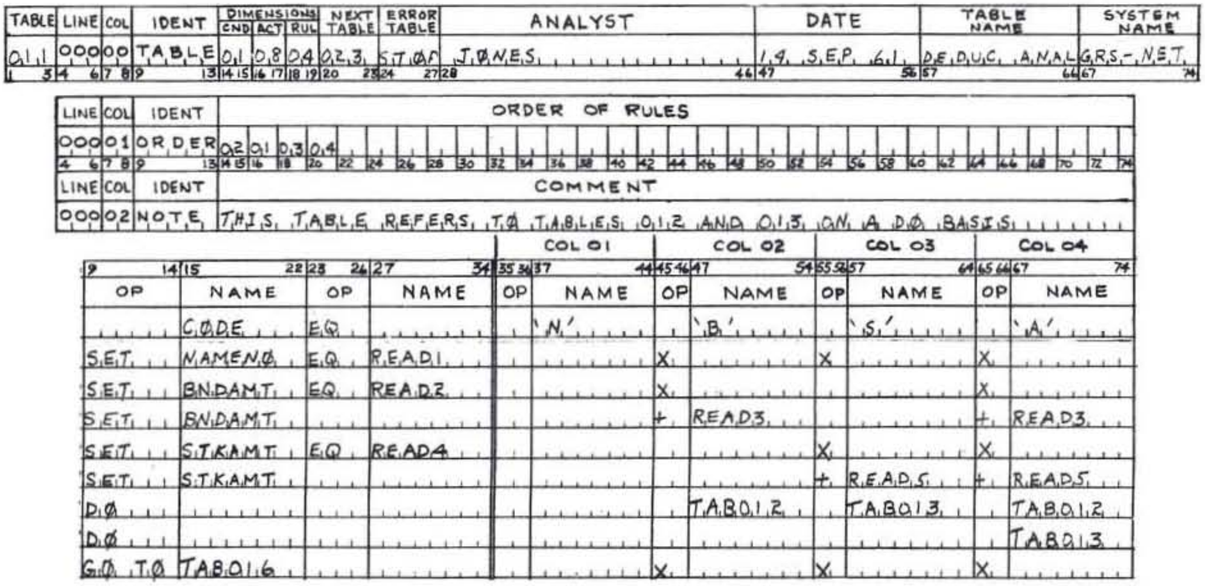


TABLE	LINE	COL	IDENT	DIMENSIONS END	ACT	RUL	NEXT TABLE	ERROR TABLE	ANALYST	DATE	TABLE NAME	SYSTEM NAME
011	000	00	TABLE	01	08	04	023	STØP	JØNES	14 SEP 61	DEDUC ANAL	GRS-NET
1 3	4 6	7 8	9 13	14 15	16 17	18 19	20 23	24 27	28 46	47 56	57 66	67 74

LINE	COL	IDENT	ORDER OF RULES
000	01	ORDER	02 01 03 04
4 6	7 8	9 13	14 15 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74

LINE	COL	IDENT	COMMENT
000	02	NOTE	THIS TABLE REFERS TØ TABLES 012 AND 013 ØN A DØ BASIS

				COL 01		COL 02		COL 03		COL 04	
9 14	15 22	23 26	27 34	35 36	37 44	45 46	47 54	55 56	57 64	65 66	67 74
OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
	CØDE	EQ			'N'		'B'		'S'		'A'
SET	NAMENØ	EQ	READ1			X		X		X	
SET	BNDAMT	EQ	READ2			X				X	
SET	BNDAMT					+	READ3			+	READ3
SET	STKAMT	EQ	READ4					X		X	
SET	STKAMT							+	READ5	+	READ5
DØ							TAB012		TAB013		TAB012
DØ											TAB013
GØ TØ	TAB016					X		X		X	

Situation 6--Procedure Description, Table 011

Situation #6 outlines a stock and bond purchase procedure, the method of which will be clear from the tables themselves as we proceed.

The Procedure Description Tables are shown with table headers; every table, whether procedure or data description, must have a header. The table header provides to the Tabular Programming System processor certain identification and control information, as may be seen by inspecting the example.

While most of the information in the procedure description header has an obvious purpose certain items require further explanation.

"Next Table" is used to designate the table to be executed next in the absence of a GO TO in the table actions for a rule. Even if every rule contains a GO TO, however, something must be written in the Next Table position. If there is a choice as to which table number to enter, the table most likely to be executed next should be specified, this may speed the execution of the program. "Error Table" is used to specify the number of the table to be executed if no set of conditions is satisfied. Since such a situation indicates an error in the data or in the logic of the table itself, the word STOP can be written.

"Order of Rules" can be used to indicate the order in which the conditions should be tested, if they should be tested in some order other than the normal left-to-right manner. The idea is that to conserve time in the running program, the first set of conditions tested ought to be the one that is most likely to be satisfied. The programmer will often write the rul.es in this order anyway, but if after filling out

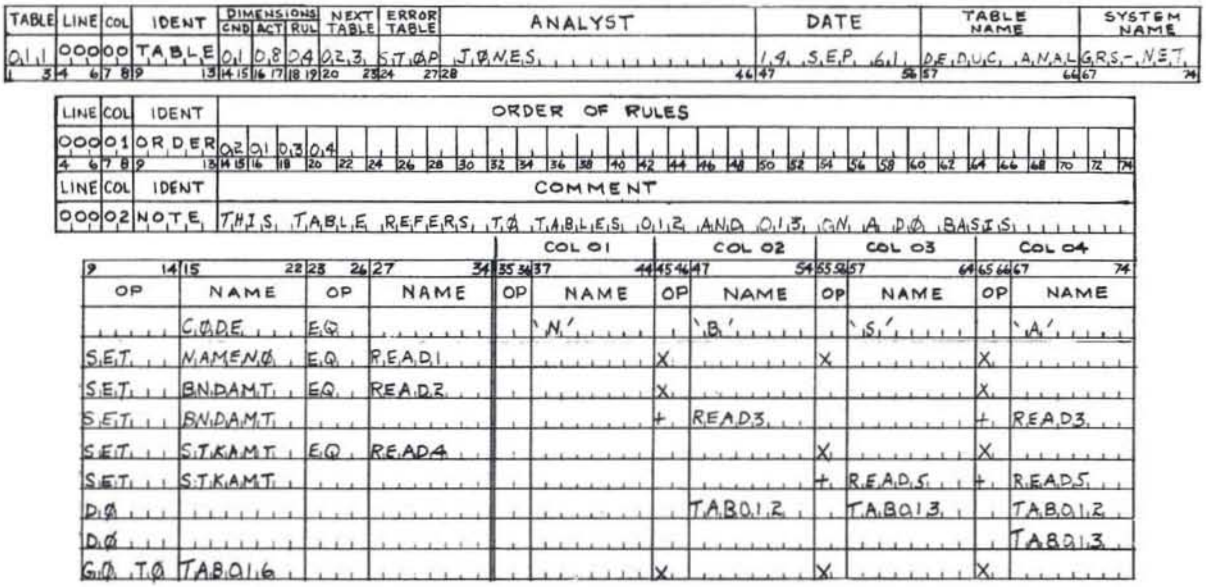


TABLE	LINE	COL	IDENT	DIMENSIONS CND	ACT	RUL	NEXT TABLE	ERROR TABLE	ANALYST	DATE	TABLE NAME	SYSTEM NAME
011	000	00	TABLE	01	08	04	023	STØP	JØNES	14 SEP 61	DEDUC ANAL	GRS-NET
1 3	4 6	7 8	9 13	14 15	16 17	18 19	20 23	24 27	28 46	47 56	57 66	67 74

LINE	COL	IDENT	ORDER OF RULES
000	01	ORDER	02 01 03 04
4 6	7 8	9 13	14 74

LINE	COL	IDENT	COMMENT
000	02	NOTE	THIS TABLE REFERS TØ TABLES 012 AND 013 ØN A DØ BASIS

				COL 01		COL 02		COL 03		COL 04	
9 14	15 22	23 26	27 34	35 36	37 44	45 46	47 54	55 56	57 64	65 66	67 74
OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
	CØDE	EQ			'N'		'B'		'S'		'A'
SET	NAMENØ	EQ	READ1			X		X		X	
SET	BNDAMT	EQ	READ2			X				X	
SET	BNDAMT					+	READ3			+	READ3
SET	STKAMT	EQ	READ4					X		X	
SET	STKAMT							+	READ5	+	READ5
DØ							TAB012		TAB013		TAB012
DØ											TAB013
GØ TØ	TAB016					X		X		X	

Situation 6--Procedure Description, Table 011

the table he decides that the order as written is not the best, the Order of Rules line can be used to specify a different testing sequence. The line is not used , if it is not needed.

The situation described by tables 011, 012, and 013 is a stock and bond deduction computation. It is assume d that prior to the execution of these tables, a card has been read that contains a code to indicate participation, as follows:

N Neither bond nor stock

B Bond

S Stock

A Bond and Stock

The card also contains:

In field named	the quantity
Read1	Name and employee number
Read2	Previous balance for bond purchase
Read3	Amount of this deduction for bond purchase
Read4	Previous balance for stock purchase
Read5	Amount of this deduction for stock purchase

Table 011 tests for participation in either or both plans. Rule 2 says that if the man is buying bonds, his name and address are placed in a punch field named NAMENO, his previous bond balance is placed in BNDAMT, the amount of this deduction is added to the balance, and we then "DO" Table 012.

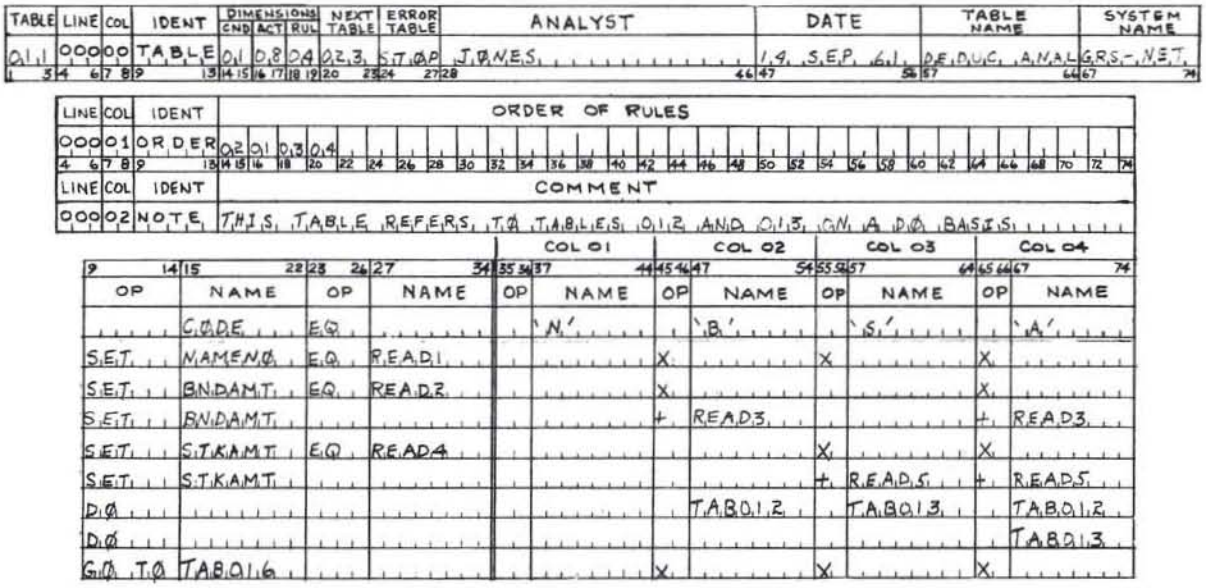


TABLE	LINE	COL	IDENT	DIMENSIONS			NEXT TABLE	ERROR TABLE	ANALYST	DATE	TABLE NAME	SYSTEM NAME
				CND	ACT	RUL						
011	000	00	TABLE	01	08	04	023	STØP	JØNES	14 SEP 61	DEDUC ANAL	GRS-NET
1 3	4 6	7 8	9 13	14 15	16 17	18 19	20 23	24 27	28 46	47 56	57 64	67 74

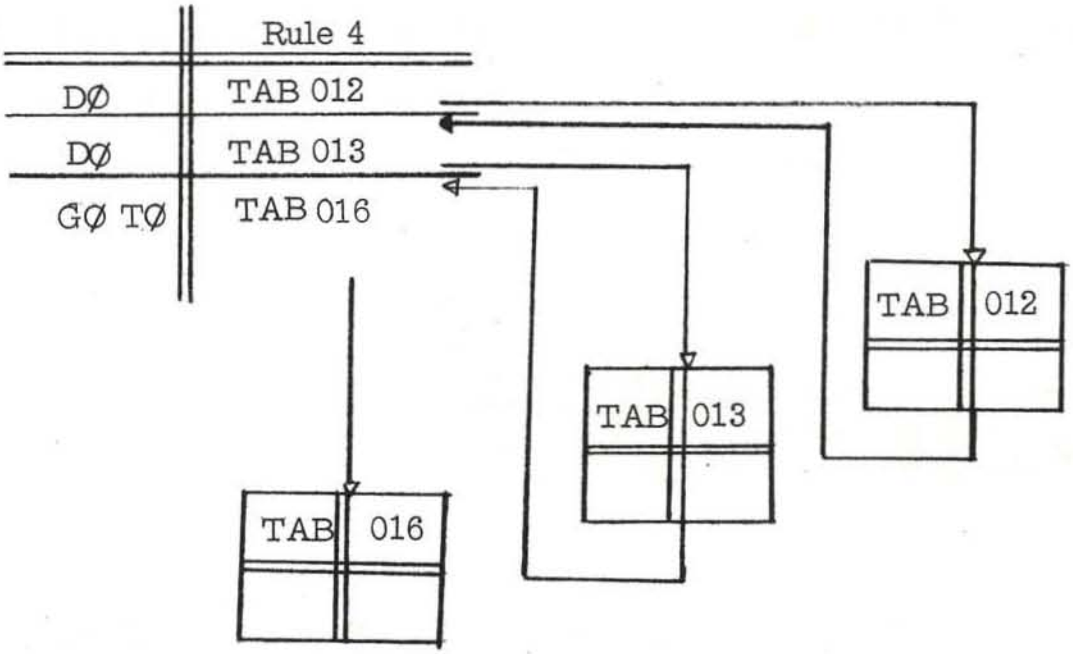
LINE	COL	IDENT	ORDER OF RULES
000	01	ORDER	02 01 03 04
4 6	7 8	9 13	14 15 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74

LINE	COL	IDENT	COMMENT
000	02	NOTE	THIS TABLE REFERS TØ TABLES 012 AND 013 ØN A DØ BASIS

				COL 01		COL 02		COL 03		COL 04	
9 14	15 22	23 26	27 34	35 36	37 44	45 46	47 54	55 56	57 64	65 66	67 74
OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
	CØDE	EQ			'N'		'B'		'S'		'A'
SET	NAMENØ	EQ	READ1			X		X		X	
SET	BNDAMT	EQ	READ2			X				X	
SET	BNDAMT					+	READ3			+	READ3
SET	STKAMT	EQ	READ4					X		X	
SET	STKAMT							+	READ5	+	READ5
DØ							TAB012		TAB013		TAB012
DØ											TAB013
GØ TØ	TAB016					X		X		X	

Situation 6--Procedure Description, Table 011

DO is a new action, which means to carry out the table specified, then return to the action following the DO. The action corresponds closely to the linkage to a closed subroutine: it sets up a transfer to the named table, and the return when that table has been executed. Looking ahead, we notice that Rule 3 calls for a DO of table 013, and Rule 4 calls for a DO of both 012 and 013. The sequence of execution of these related tables is shown more clearly in a schematic diagram:



The final action in Rule 2 is to GO TO table 016; if Rule 2 is satisfied and executed, this command overrides the "next table" specification in the header, which designates the table to execute next in the absence of a GO TO.

Rule 3 of table 011 specifies corresponding actions for the man who participates in the stock purchase plan only, and Rule 4 covers the employee who is in both plans.

TABLE	LINE	COL	IDENT	DIMENSIONS			NEXT TABLE	ERROR TABLE
				CND	ACT	RUL		
012	000	00	TABLE	01	05	02	DØ	
1 3	4 6	7 8	9 13	14 15	16 17	18 19	20 23	24 27

OP	NAME	OP	NAME	OP	NAME	OP	NAME
	BNDAMT			GE	18.75		ELSE
SET	PNCH1	EQ			18.75		
SET	PNCHCD	EQ			'B'		
SET	BNDAMT			-	1875		
WRITE	PUNCH			X			
DØ	TAB012			X			

Situation 6--Procedure Description Table 012

TABLE	LINE	COL	IDENT	DIMENSIONS			NEXT TABLE	ERROR TABLE
				CND	ACT	RUL		
013	000	00	TABLE	01	05	02	DØ	
1 3	4 6	7 8	9 13	14 15	16 17	18 19	20 23	24 27

OP	NAME	OP	NAME	OP	NAME	OP	NAME
	STKAMT			GE	MRKVAL		ELSE
SET	PNCH1	EQ			MRKVAL		
SET	PNCHCD	EQ			'S'		
SET	STKAMT			-	MRKVAL		
WRITE	PUNCH			X			
DØ	TAB013			X			

Situation 6--Procedure Description Table 013

Table 012 determines whether the employee's balance for bond purchase, including this pay period's deduction, is great enough to buy one bond at $18.75. If it is, the $18.75 is placed in the punch area, a code of "B" is set up in the punch area, $18.75 is subtracted from the balance for buying bonds, a card is punched and the same table is repeated. If the bond balance is not as much as $18.75, which is covered by the ELSE condition entry, we merely return to the original table. In either case, control is returned to table 011. Table 013 performs an analagous computation for stock purchase; the only difference is that the current market value of the stock is used in place of the fixed $18.75.

We may notice in the headers for tables 012 and 013 that the next table entry is "DO", which indicates to the processor that this is a closed table-- ie., that it is entered only by a DO action in some other table, and that control is always returned from this table to the "calling" table. An open table is one which does specify a next table by number. A closed (DO) table may not contain a GO TO.

QUESTIONS

1. What action is specified by a DO?
2. State rule 3 of table 011 in words.
3. When a rule does not end in a GO TO action, what table is executed next?
4. Can a closed table be referred to by a DO in more than one table?
5. Can one DO table DO another table?

Answers - Situation 6

1. Execute the table named in the operand and return to the next action in the same rule.

2. If the code is S (for stock), put the employee's name into the print area, put the previous balance for stock purchase into STKAMT, add the amount of this deduction, DO table.013, and GO TO table 016.

3. The "next table" specified in the table header.

4. Yes.

5. Yes.

Situation #7 involves an inventory file in disk storage. There are 4000 items in the inventory; each item is assigned to one sector in the range of sector addresses from 15000 to 19999, which is 5000 sectors. The part numbers are eight digits in length. The extra length over what would be required to identify each item uniquely, is used to code certain information about the parts. We are given a part number, and asked to read the corresponding record from disk storage. Other tables, not shown here, would process the record.

The fundamental problem is how to locate the proper record, or, stating it another way, how to transform the part number into a sector address. The method to be used here is fairly standard (although it should be understood that no one method is satisfactory for all applications). We divide the part number by 5000 and add the remainder to 15000 to get a sector address. The remainder on division by 5000 will of course be a number between zero and 4999; adding this to 15000 gives a number between 15000 and 19999, which is the assigned range of sector numbers. Thus, for example, the part number 39582067 leads to the sector address 17067.

The problem is, many other eight-digit part numbers also lead to the same sector address; for instance 57202067 and 12347067 also give sector address 17067. Since there are only 4000 items in the inventory, we don't expect a great many sector address duplications, but there certainly could be some. This leads to the idea of a chained file, in which we have a chain of records corresponding to the part

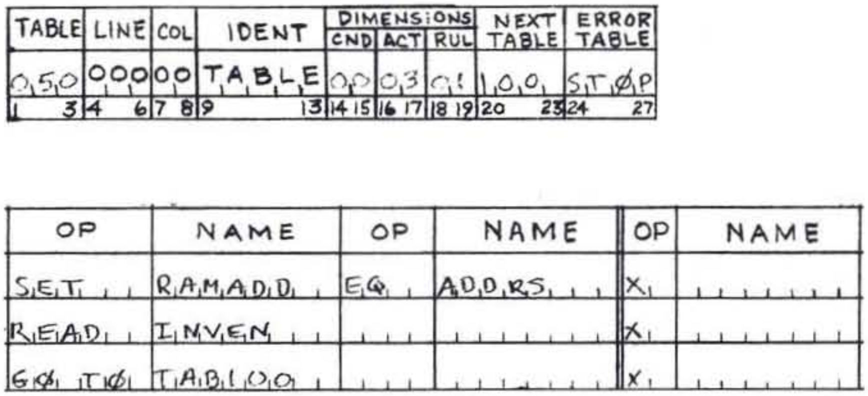
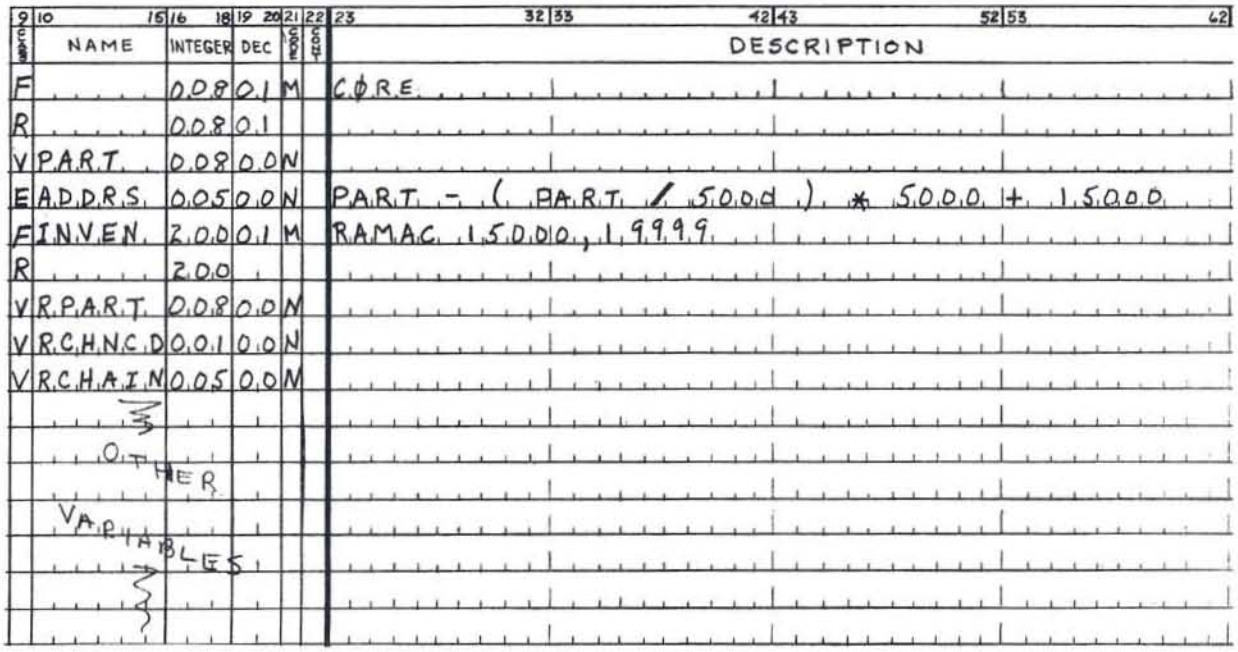


TABLE	LINE	COL	IDENT	DIMENSIONS			NEXT TABLE	ERROR TABLE
				CND	ACT	RUL		
050	000	00	TABLE	00	03	01	100	STØP
1 3	4 6	7 8	9 13	14 15	16 17	18 19	20 23	24 27

OP	NAME	OP	NAME	OP	NAME
SET	RAMADD	EQ	ADDRS	X	
READ	INVEN			X	
GØ TØ	TAB100			X	

Situation 7--Procedure Description Table 050



9	10 15	16 18	19 20	21	22	23 32 | 33 42 | 43 52 | 53 62
CODE	NAME	INTEGER	DEC	MODE	CONT	DESCRIPTION
F		008	01	M		CØRE
R		008	01			
V	PART	008	00	N		
E	ADDRS	005	00	N		PART - (PART / 5000) * 5000 + 15000
F	INVEN	200	01	M		RAMAC 15000, 19999
R		200				
V	RPART	008	00	N		
V	RCHNCD	001	00	N		
V	RCHAIN	005	00	M		
	OTHER					
	VARIABLES					

Situation 7--Data Description Table

numbers which give duplicate sector addresses. After applying the address computation formula, we read the record at that address, which is called the home record of the chain. Then we compare the part number in that record with the original part number. If they are the same, the home record is the correct one and we proceed with the computation. If they are not the same, we look in the home record for the sector address of the first overflow record of the chain, read it, and again compare part numbers. If this still isn't the proper record, we go down the chain, etc., etc.

Before proceeding to the tables for this situation, we should investigate the question of how to obtain a remainder when the programming system does not provide it directly. All that is necessary is to obtain the quotient as an unrounded integer (whole number) and then do a little additional arithmetic. For an example of the procedure, suppose we wanted the remainder from dividing 23 by 4. The exact quotient is 5.75; the unrounded integer quotient is 5 (not 6). Now multiply this quotient by the divisor, giving 20, and subtract from the dividend: $23 - (5 \cdot 4) = 3$, which is indeed the remainder.

Look now at table 050, which introduces a new feature; the first operand calls for the evaluation of an expression, named ADDRS. To see what computation is specified, we look to the data description table. There we find that ADDRS is of class E, for expression and in the description area we see the expression

```
PART - (PART / 5000) * 5000 + 15000
```

TABLE	LINE	COL	IDENT	DIMENSIONS			NEXT TABLE	ERROR TABLE
				CND	ACT	RUL		
050	000	00	TABLE	00	03	01	100	STØP
1 3	4 6	7 8	9 13	14 15	16 17	18 19	20 23	24 27

OP	NAME	OP	NAME	OP	NAME
SET	RAMADD	EQ	ADDRS	X	
READ	INVEN			X	
GØ TØ	TAB100			X	

Situation 7--Procedure Description Table 050

9	10 15	16 18	19 20	21	22	23 62
CODE	NAME	INTEGER	DEC	CODE	CONT	DESCRIPTION
F		008	01	M		CØRE
R		008	01			
V	PART	008	00	N		
E	ADDRS	005	00	N		PART - (PART / 5000) * 5000 + 15000
F	INVEN	200	01	M		RAMAC 15000, 19999
R		200				
V	RPART	008	00	N		
V	RCHNCD	001	00	N		
V	RCHAIN	005	00	N		
	OTHER					
	VARIABLES					

Situation 7--Data Description Table

TABLE	LINE	COL	IDENT	DIMENSIONS			NEXT TABLE	ERROR TABLE
				CND	ACT	RUL		
100	000	00	TABLE	02	03	03	200	
1 3	4 6	7 8	9 13	14 15	16 17	18 19	20 23	24 27

OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
	RPART	EQ	PART	Y		N		N	
	RCHNCD					EQ	1	EQ	0
SET	RAMADD	EQ	RCHAIN			X			
READ	INVEN					X			
GØ TØ					TAB200		TAB100		TAB300

Situation 7--Procedure Description Table 100

The symbols here call for arithmetic operations in a fairly obvious manner:

Addition	+
Subtraction	-
Multiplication	*
Division	/

The ordinary rules of precedence apply in an expression: parentheses indicate operations to be performed first, then all multiplications and divisions are done, then all additions and subtractions. Thus the first computation will be the division of the part number by 5000. With all variables and the final result being integers, this division will give an integer quotient, as required, and the arithmetic system supplied by the processor happens not to round--so we have exactly what we want. This quotient is next multiplied by 5000, the product subtracted from the part number giving the remainder--and the remainder is finally added to 15000, giving the sector number.

This result is the value of the expression ADDRS, which is transferred by the SET. . . EQ operator to a special variable named RAMADD. This sets up the complete seven-digit disk file address. The READ then brings the record from disk storage into core storage, and the GO TO takes us to table 100.

Table 100 determines whether this home record is the right one, and proceeds down the chain if not. Here we meet a slight problem:

TABLE	LINE	COL	IDENT	DIMENSIONS			NEXT TABLE	ERROR TABLE
				CND	ACT	RUL		
100	000	00	TABLE	02	03	03	200	
1 3	4 6	7 8	9 13	14 15	16 17	18 19	20 23	24 27

OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
	RPART	EQ	PART	Y		N		N	
	RCHNCD					EQ	1	EQ	0
SET	RAMADD	EQ	RCHAIN			X			
READ	INVEN					X			
GØ TØ					TAB200		TAB100		TAB300

Situation 7--Procedure Description Table 100

it could happen that the part number was incorrect, and that no record in this chain is the correct one. For this reason the chained records have been set up with a chain continuation code, which is one if more records in the chain remain, and zero if not.

The first rule in this table simply asks whether the part number in the disk record is the same as the part number with which we started. If it is, then we GO TO table 200 to carry on with the processing, whatever it may be. If the part numbers are not the same, rule 2 asks whether there is a record in the chain remaining. If there is, RAMADD is set equal to the address of the next record in the chain, which address is contained in this record, reads it, and returns to repeat this table. If the part numbers do not match and no records remain, then the input part number must have been wrong and we GO TO table 300, which would handle the error condition.

QUESTIONS

1. After having read the home record, would it have been possible to proceed with the part number comparison in the same table?
2. Why was the second condition in Rule 1 of table 100 left bank?
3. Why was it acceptable to omit file and record names for the core storage file? Why was it essential that a name be assigned to the disk file?
4. How is a sector address specified before reading or writing in a disk file?
5. What must be done to use an arithmetic expression?

Answers - Situation 7

1. Absolutely not! Doing so would require conditions in the action area, which is not permitted.

2. If the part number in the record is the same as the original part number, it doesn't matter whether there are more records in the chain.

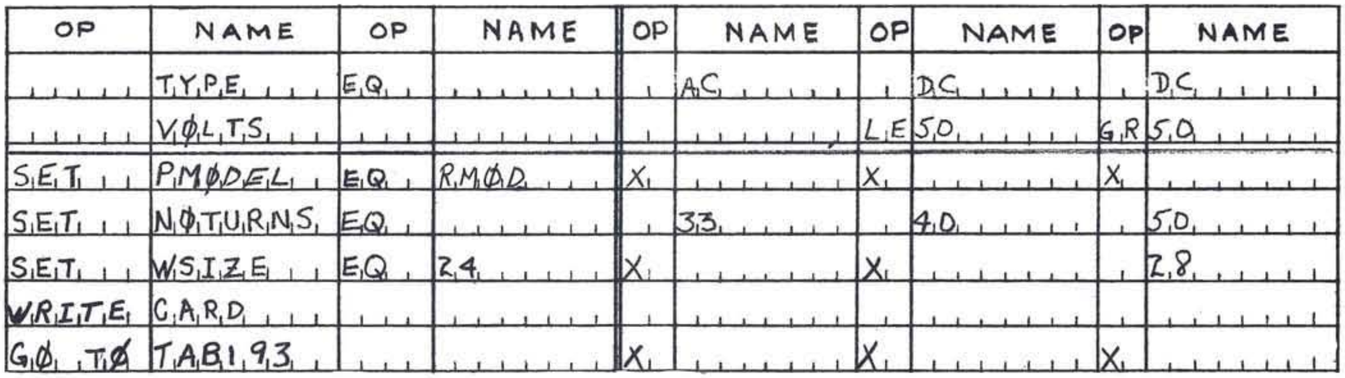
3. The core file and record are never referred to by name; the disk file is.

4. By placing the five-digit sector address in the special variable named RAMADD.

5. Write E for the class in the data description, and write the expression itself in the data description. These refer to the name of the expression in the procedure description.

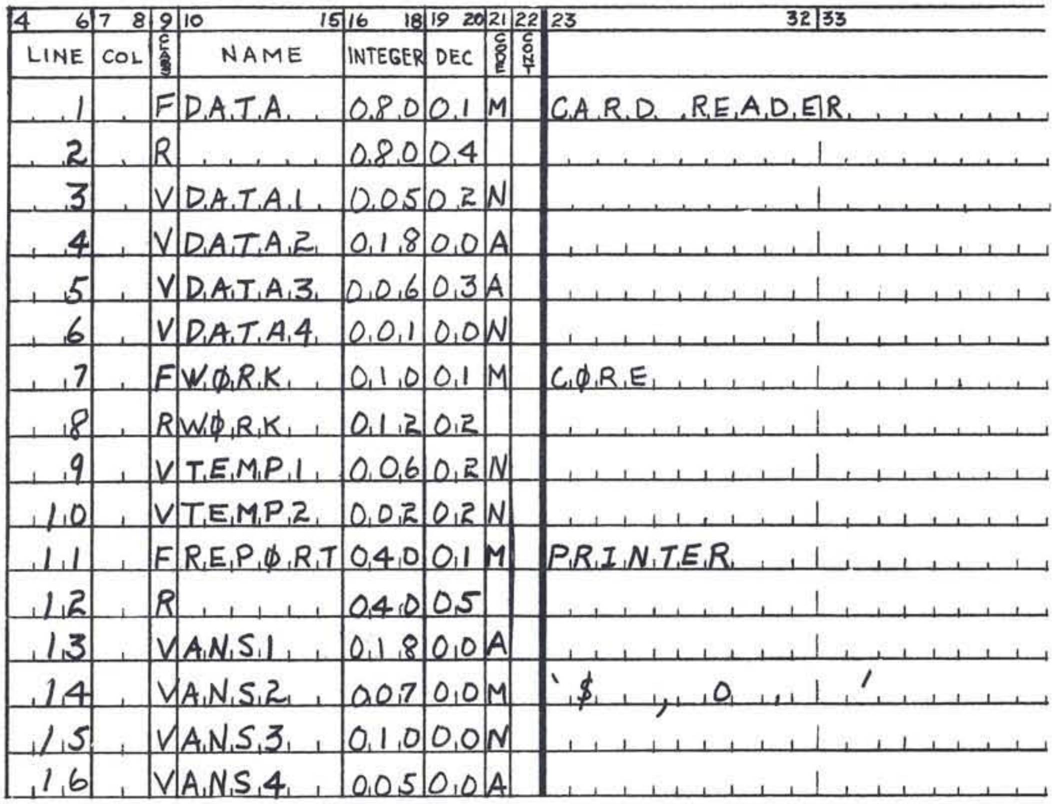
EXERCISES

The following three problem situations are accompanied by complete tables, but each table contains a few errors. Identify Them.



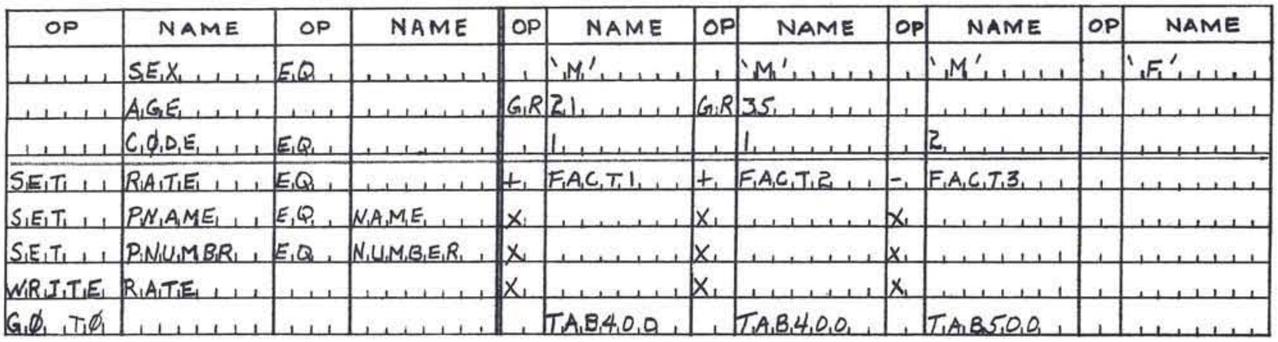
OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
	TYPE	EQ			AC		DC		DC
	VØLTS					LE	50	GR	50
SET	PMØDEL	EQ	RMØD	X		X		X	
SET	NØTURNS	EQ			33		40		50
SET	WSIZE	EQ	24	X		X			28
WRITE	CARD								
GØ TØ	TAB193			X		X		X	

Problem 1



4 6	7 8	9	10 15	16 18	19 20	21	22	23 32	33
LINE	COL	CLASS	NAME	INTEGER	DEC	CODE	CONT		
1		F	DATA	080	01	M		CARD READER	
2		R		080	04				
3		V	DATA1	005	02	N			
4		V	DATA2	018	00	A			
5		V	DATA3	006	03	A			
6		V	DATA4	001	00	N			
7		F	WØRK	010	01	M		CØRE	
8		R	WØRK	012	02				
9		V	TEMP1	006	02	N			
10		V	TEMP2	002	02	N			
11		F	REPØRT	040	01	M		PRINTER	
12		R		040	05				
13		V	ANS1	018	00	A			
14		V	ANS2	007	00	M		`$, 0	'
15		V	ANS3	010	00	N			
16		V	ANS4	005	00	A			

Problem 2



OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
	SEX	EQ			`M'		`M'		`M'		`F'
	AGE			GR	21	GR	35				
	CØDE	EQ			1		1		2		
SET	RATE	EQ		+	FACT1	+	FACT2	-	FACT3		
SET	PNAME	EQ	NAME	X		X		X			
SET	PNUMBR	EQ	NUMBER	X		X		X			
WRITE	RATE			X		X		X			
GØ TØ					TAB400		TAB400		TAB500		

Problem 3

1. A card has been read, containing certain information about a voltmeter: model number, the letters AC or DC to indicate alternating or direct current, and the voltage rating. You are to punch a card containing model number, number of turns, and wire size, with the latter two being determined by the following rule: an AC voltmeter always has 33 turns of size 24 wire; a DC voltmeter has 40 turns of size 24 wire if the voltage is less than 50, and 50 turns of size 28 wire otherwise.

2. The data description shown here contains many errors, which you should be able to identify without knowing anything about the problem.

3. The procedure description table shown here contains several errors, one of them fundamental.

						COL 01		COL 02		COL 03		COL 04		COL 05	
4 6	7 8	9 14	15 22	23 26	27 34	35 36	37 44	45 46	47 54	55 56	57 64	65 66	67 74	9 10	11 18
LINE	COL	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
01	00		SEX	EQ			'MALE'		'MALE'		'FEMALE'		'FEMALE'		'FEMALE'
02	00		DEPT	EQ			47				48				49
03	00		HØURS	LE	40	Y						Y		Y	
04	00		DEDCØD				'B'				'C'				'C'
05	00		RATE									GR	250		
06	00	GØ TØ					TAB010				TAB020		TAB020		
07	00														
08	00														
09	00														
10	00														
11	00														
12	00														
13	00														
14	00														
15	00														
16	00														
17	00														
18	00														
19	00														
20	00														

	COL 06		COL 07		COL 08		COL 09	
4 6	19 20	21 28	29 30	31 38	39 40	41 48	49 50	51 58
LINE	OP	NAME	OP	NAME	OP	NAME	OP	NAME
01		'FEMALE'		'FEMALE'		'FEMALE'		ELSE
02				50				
03	Y		Y		Y			
04				'C'				
05	GR	250			GR	250		
06				TAB020				
07								
08								
09								
10								
11								
12								
13								
14								
15								
16								
17								
18								
19								
20								

Problem 4

4. Previous tables have established values for the fields sex , department, hours worked, deduction code, and rate. A table is required to do the following:

- If the employee is a male in department 47 who worked less than 40 hours and whose deduction code is B or D, GO TO table 010.
- If the employee is a female, works in department 48, 49, or 50, worked less than or equal to 40 hours, and either has a deduction code of C, or has an hourly rate of more than $2.50 then GO TO table 020.
- If the employee satisfies neither of these conditions, GO TO table 030.

The table shown for this task is incomplete; nothing written in it is incorrect, but a number of boxes are not filled in.

Complete the table.

5. Given a deck of cards as input to a customer billing procedure. For each customer, there is a heading card giving certain information about the customer, followed by one or more cards representing purchases. You are required to print an invoice for each customer, with a heading at the top of the page, followed by a line for each purchase, followed by the total of the purchases.

Heading card

Cols 1 - 6	Account number
7 - 29	Name
30 - 65	Address
80	Zero (to identify heading card)

Purchase cards

Cols 7 - 12	Catalog number
20 - 36	Description
37 - 42	Price
80	One (to identify purchase card)

Heading line

Positions 1 - 6	Account number (suppress leading zeros)
12 - 34	Name
40 - 75	Address

Purchase line

Positions 3 - 8	Catalog number
12 - 28	Description
34 - 39	Price (suppress leading zeros and insert punctuation.)

Total line

Positions 32 - 39	Total price (suppress leading zeros and insert punctuation.)

A blank line should be left between the heading and the first purchase line, and between the last purchase line and the total line.

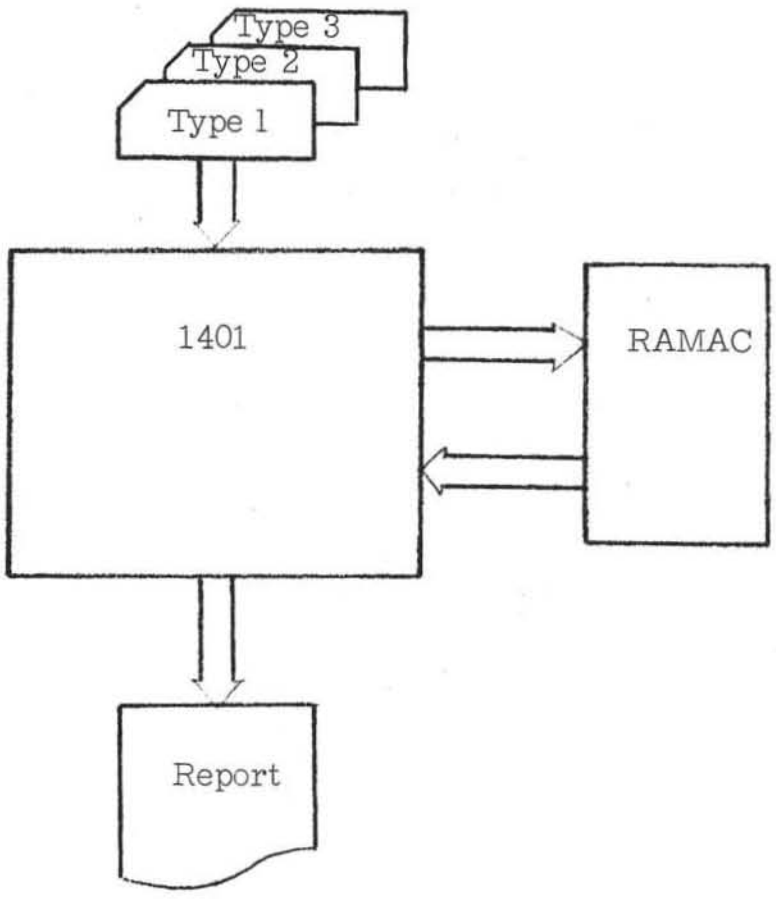
Write tables to handle this processing, including all table headers and the data description table. Include any tables necessary to read the first card and to handle the last card.

CHAPTER 3

Expense Report Program

This program prepares a weekly and monthly expense report for department managers. The report shows the expenses incurred on a month-to-date and year-to-date basis. For the managers' convenience the report shows the budget allowance for the particular time period and the actual expense variance from budget.

The overall processing diagram is shown below:



The inputs to the program are received through the card reader. There are three types of input cards.

1. Budgets - The amount of expense allowed by the budget for each type of expense.

   . code (numeric one)
   . account number
   1) Div. number
   2)Dept. number
   3) G/L
   4) S/L
   . monthly budget amount in dollars
   or
   . monthly budget allowance

2. Year to Date Correction - a correction to the year-to-date actual.

   . code (numeric two)
   . account number (same as budget
   . amount of correction

3. Actual Expenses - the expenses incurred during current week.

   . code (numeric three)
   . account number (same as budget)
   . amount ot expense

The RAMAC contains a master for each existing account number. The master fields are

. account number
- Division
- Dept.
- G/L
- S/L
. month-to-date actual
. year-to-date actual
. budget allowance per month
. budget allowance year-to-date

The RAMAC also contains a table of Sub Ledger definitions. This enables the program to interpret the S/L number and print an alphabetic description of the expense type.

The account number is eleven digits and has four significant parts.

1. Division Number
2. Department Number (within Division)
3. General Ledger (major type of expense)
4. Sub Ledger (minor expense type)

DIV.	DEPT.	G/L	S/L
2	2	4	3

Account numbers are used in much the same way that charge account numbers are used in retail sales. However, account numbers also include information on the type of expense. For example, the functional account number

4221-5100-301

is read as Division 42, Dept. 21, major expense type 5100, minor expense type 301.

For this problem the following tables define the expense types:

G/L (major expense types)

5100	Manufacturing Expense
5200	Sales Expense
5300	Advertising Expense
5400	Administrative Expense

S/L (minor expense types)

100	Salaries
200	Indirect Labor
301	Shop Supplies
305	Stationary Supplies
307	Maintenance Supplies
400	Tools
402	Tool Development
500	Overtime Payment
501	Supper Money Payments
507	Maintenance Labor
520	Second Shift Premium
521	Third Shift Premium
601	Freight In
602	Freight Out
605	Dunnage
700	Product Test
802	Medical Expense
890	Facilities Rental
900	Travel Expense
905	Sundry Expense
910	Automobile Expense
925	Legal Fees
990	Scrapped Product
991	Inventory Adjustment
999	Redistributed Expense

Using these definitions of major and minor expenses the previous example

4221-5100-301

reads Division 42, Dept. 21, Manufacturing Expense (5100), Shop Supplies (301).

Department number ranges also indicate further the type of junction involved, as only certain types of expenses can occur in certain departments.

Below 50 are manufacturing units which may have 5100 expense only. 50-69 are administrative units which have only 5400 expense. 70-79 are advertising units which have only 5300 expense. 80-99 are sales units which have only 5200 expense.

IBM®

1401 TP DATA DESCRIPTION

T P 1401

TABLE	LINE	COL	IDENT						ANALYST	DATE	TABLE NAME	SYSTEM NAME	IDENTIFICATION NAME
000	000	00	TABLE	00	00	00	0000	0000	L. R. SMITH	OCT. 17, 61	DATA	BUDGET	REPORT
1 3	4 6	7 8	9 13	14 15	16 17	18 19	20 23	24 27	28 46	47 56	57 66	67 74	75 80

4 6	7 8	9	10 15	16 18	19 20	21	22	23 32 / 33 42 / 43 52 / 53 62 / 63 72
LINE	COL	CLASS	NAME	INTEGER	DEC	CODE	CONT	DESCRIPTION
002	00	F	MASTER	200	01	M		RAMAC 10000, 59999
004	00	R	RM	200	09			
006	00	V	MDIV	002	00	N		
008	00	V	MDEPT	002	00	N		
010	00	V	MSL	003	00	N		
012	00	V	MEXP	008	02	N		
014	00	V	MYEXP	008	02	N		
016	00	V	MBUDG	008	00	N		
018	00	V	MYBUDG	008	00	M		
020	00	V	CHAIN	005	00	N		
022	00	V		152	00	A		
030	00	F	REFER	200	01	M		RAMAC 60000, 69999
032	00	R	REF	200	03			
034	00	V	SUB	003	00	N		100
036	00	V	TYPE	020	00	A		`SALARIES '
038	00	V		177	00	A		
040	00	F		200	01			CORE
042	00	R	MM	200	02			
044	00	V		048	00	A		`000000000000000000000000000000000000000000 '
046	00	V		152	00	A	C	` '
046	01	V					C	` '
046	02	V						` '

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50

IBM®

1401 TP DATA DESCRIPTION

TP 1401

TABLE	LINE	COL	IDENT						ANALYST	DATE	TABLE NAME	SYSTEM NAME	IDENTIFICATION NAME
000	000	00	TABLE	00	00	00	0000	0000					
1 3	4 6	7 8	9 13	14 15	16 17	18 19	20 23	24 27	28 46	47 56	57 66	67 74	75 80

4 6	7 8	9	10 15	16 18	19 20	21	22	23 32 33 42 43 52 53 62 63 72
LINE	COL	CLASS	NAME	INTEGER	DEC	CODE	CONT	DESCRIPTION
060		F	CARD	080	01	M		READER , EOF TAB028
062		R	EXPENS	080	13			
064		V	CODE	001	00	N		
066		V	DIV	002	00	N		
068		V	DEPT	002	00	N		
070		V	GL	004	00	N		
072		V	SL	003	00	N		
074		V	AMOUNT	008	02	N		
076		V		020	00	A		
078		V	DAYWO	002	00	M		
080		V	DAYS	002	00	N		
082		V	DAY	002	00	N		
084		V	MONTH	002	00	N		
086		V	YEAR	002	00	N		
088		V		030	00	A		
090	00	F	PUNCH	080	01	M		PUNCH
092	00	R	ERRORC	080	01			
094	00	V		080	00	A		
096	00	F	PRINT	132	01	M		PRINTER
097	00	R	PR	132	01			
098	00	V		132	00			

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50

IBM ® 1401 TP DATA DESCRIPTION TP 1401

TABLE	LINE	COL	IDENT						ANALYST	DATE	TABLE NAME	SYSTEM NAME	IDENTIFICATION NAME
000	000	00	TABLE	00	00	00	0000	0000					
1 3	4 6	7 8	9 13	14 15	16 17	18 19	20 23	24 27	28 46	47 56	57 66	67 74	75 80

LINE 4 6	COL 7 8	ENTRY 9	NAME 10 15	INTEGER 16 18	DEC 19 20	CODE 21	CONT 22	DESCRIPTION 23 72
100	00	C	HEADT	132	00	A	C	\ TYPE OF EXPENSE NO. EXPENSE BUDGE'
100	01	C					C	\T VARIANCE EXPENSES BUDGET VARIAN'
100	02	C						\CE /
102	00	C	BLANKS	007	00	A		\ '
104	00	C	ZEIGHT	008	00	N		00000000
106	00	C	ZTEN	010	00	N		0000000000
108	00	C	ZEROT	008	02	N		0000000000
110	00	E	FRACAL	001	03	N		DAYS / DAYMO
112	00	E	RAMCAL	005	00	N		MSL + 60000
114	00	E	MCAL	005	00	N		ACCTNO - (ACCTNO / 40000 * 40000) + 10000
120	00	F		132	01	M		CORE
122	00	R	HEAD	132	11			
124	00	V		006	00	A		\ DEPT '
126	00	V	RDIV	002	00	N		
128	00	V	RDEPT	002	00	N		
130	00	V		006	00	A		\ DATE '
132	00	V	RDAY	002	00	N		
134	00	V		001	00	A		\ '
136	00	V	RMONTH	002	00	N		
138	00	V		001	00	A		\ '
140	00	V	RYEAR	002	00	N		
142	00	V		047	00	A		\ MONTH TO DATE '
144	00	V		061	00	A	C	\YEAR TO DATE '
144	01	V						\ '

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50

IBM®

1401 TP DATA DESCRIPTION

TP 1401

TABLE	LINE	COL	IDENT						ANALYST	DATE	TABLE NAME	SYSTEM NAME	IDENTIFICATION NAME
000	000	00	TABLE	00	00	00	0000	0000					
1 3	4 6	7 8	9 13	14 15	16 17	18 19	20 23	24 27	28 46	47 56	57 66	67 74	75 80

LINE	COL	CLASS	NAME	INTEGER	DEC	CODE	CONT	DESCRIPTION
4 6	7 8	9	10 15	16 18	19 20	21	22	23 32 33 42 43 52 53 62 63 72
150	00	F	WORK	063	01	M		CORE
152	00	R		063	09			
154	00	V	FRACMO	001	03	N		
156	00	V	MTD	008	02	N		
158	00	V	MTDB	008	00	N		
160	00	V	YTD	008	02	M		
162	00	V	YTDB	008	00	N		
164	00	V	W	008	00	N		
166	00	V	WW	008	00	N		
168	00	N	EOFI	001	00	N		0
170	00	V	MADDR	005	00	N		
180	00	F		080	01	M		CORE
182	00	R	ADCALC	080	03			
184	00	V		001	00	N		
186	00	V	RNDMZE	011	00	N		
188	00	V		068	00	A		
190	00	F		007	02	M		CORE
192	00	R	CDCAL	007	03			
194	00	V	CDIV	002	00	N		
196	00	V	CDEPT	092	00	N		
198	00	V	CDSL	003	00	N		
200	00	R	CDCALA	007	01			
202	00	V	ACCTNO	007	00	N		

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50

IBM® 1401 TP DATA DESCRIPTION

T P 1401

TABLE	LINE	COL	IDENT						ANALYST	DATE	TABLE NAME	SYSTEM NAME	IDENTIFICATION NAME
000	000	00	TABLE	00	00	00	0000	0000					
1 3	4 6	7 8	9 13	14 15	16 17	18 19	20 23	24 27	28 46	47 56	57 66	67 74	75 80

4 6	7 8	9	10 15	16 18	19 20	21	22	23 32	33 42	43 52	53 62	63 72
LINE	COL	CLASS	NAME	INTEGER	DEC	CODE	CONT	DESCRIPTION				
210	00	F		132	01	M		CORE				
212	00	R	DETAIL	132	17							
214	00	V		001	00	A		' '				
216	00	V	DTYPE	020	00	A						
218	00	V		002	00	A		' '				
220	00	V	DNO	003	00	A						
222	00	V		001	00	A		' '				
224	00	V	DMEXPN	011	00	M		' , , -'				
226	00	V		001	00	A		' '				
228	00	V	DMBUDG	011	00	M		' , , -'				
230	00	V		001	00	A		' '				
232	00	V	DMVARN	011	00	M		' , , -'				
234	00	V		002	00	A		' '				
236	00	V	DYEXPN	011	00	M		' , , -'				
238	00	V		001	00	A		' '				
290	00	V	DYBUDG	011	00	M		' , , -'				
242	00	V		001	00	A		' '				
244	00	V	DYVARN	011	00	M		' , , -'				
246	00	V		001	00	A		' '				

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50

IBM®

1401 TP PROCEDURE DESCRIPTION

TP 1401

TABLE	LINE	COL	IDENT	DIMENSIONS CND	DIMENSIONS ACT	DIMENSIONS RUL	NEXT TABLE	ERROR TABLE	ANALYST	DATE	TABLE NAME	SYSTEM NAME	IDENTIFICATION NAME
001	000	00	TABLE	00	01	01	002		ROGER SMITH	OCT. 2, 1961	INITIALIZE	BUDGET	REPORT
1 3	4 6	7 8	9 13	14 15	16 17	18 19	20 23	24 27	28 46	47 56	57 66	67 74	75 80

LINE	COL	IDENT	ORDER OF RULES
000	01	ORDER	
4 6	7 8	9 13	14 15 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74

LINE	COL	IDENT	COMMENT
000	02	NOTE	

START NEXT CARD

TABLE	LINE	COL
		05
1 3	4 6	7 8

						COL 01		COL 02		COL 03		COL 04		COL 05	
4 6	7 8	9 14	15 22	23 26	27 34	35 36	37 44	45 46	47 54	55 56	57 64	65 66	67 74	9 10	11 18
LINE	COL	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
001	00	READ	CARD			X									
002	00														
003	00														
004	00														
005	00														
006	00														
007	00														
008	00														
009	00														
010	00														
011	00														
012	00														
013	00														
014	00														
015	00														
016	00														
017	00														
018	00														
019	00														
020	00														

IBM® 1401 TP PROCEDURE DESCRIPTION

TABLE	LINE	COL	IDENT	DIMENSIONS CND	DIMENSIONS ACT	DIMENSIONS RUL	NEXT TABLE	ERROR TABLE	ANALYST	DATE	TABLE NAME	SYSTEM NAME	IDENTIFICATION NAME
002	000	00	TABLE	01	07	02	010	STOP					
1 3	4 6	7 8	9 13	14 15	16 17	18 19	20 23	24 27	28 46	47 56	57 66	67 74	75 80

LINE	COL	IDENT	ORDER OF RULES
000	01	ORDER	
4 6	7 8	9 13	14 15 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74

LINE	COL	IDENT	COMMENT
000	02	NOTE	

START NEXT CARD

TABLE	LINE	COL
		05
1 3	4 6	7 8

						COL 01		COL 02		COL 03		COL 04		COL 05	
4 6	7 8	9 14	15 22	23 26	27 34	35 36	37 44	45 46	47 54	55 56	57 64	65 66	67 74	9 10	11 18
LINE	COL	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
001	00		CODE	EQ	4	Y		N							
02	00														
003	00	SET	FRACMO	EQ	FRACAL	X									
004	00	MOVE	DAY	TO	RDAY	X									
005	00	MOVE	MONTH	TO	RMONTH	X									
006	00	MOVE	YEAR	TO	RYEAR	X									
007	00	READ	CARD			X									
008	00	DO	TAB510			X									
009	00	GO TO							TABSTOP						
10	00														
11	00														
12	00														
13	00														
14	00														
15	00														
16	00														
17	00														
18	00														
19	00														
20	00														

IBM®

1401 TP PROCEDURE DESCRIPTION

T~~P~~ 1401

TABLE	LINE	COL	IDENT	DIMENSIONS CND	DIMENSIONS ACT	DIMENSIONS RUL	NEXT TABLE	ERROR TABLE	ANALYST	DATE	TABLE NAME	SYSTEM NAME	IDENTIFICATION NAME
010	000	00	TABLE	03	04	05	011				ACCT-CHECK		
1 3	4 6	7 8	9 13	14 15	16 17	18 19	20 23	24 27	28 46	47 56	57 66	67 74	75 80

LINE	COL	IDENT	ORDER OF RULES
000	01	ORDER	03 01 04 02 05
4 6	7 8	9 13	14 15 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74

LINE	COL	IDENT	COMMENT
000	02	NOTE	ACCOUNT NUMBER CHECK, PUNCH ERROR CARD

START NEXT CARD

TABLE	LINE	COL
010		05
1 3	4 6	7 8

						COL 01	COL 01	COL 02	COL 02	COL 03	COL 03	COL 04	COL 04	COL 05	COL 05
4 6	7 8	9 14	15 22	23 26	27 34	35 36	37 44	45 46	47 54	55 56	57 64	65 66	67 74	9 10	11 18
LINE	COL	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
001	00		DEPT	GR					49		69		79		ELSE
002	00		DEPT	LR			50		70		80				
003	00		GL	EQ			5100		5200		5300		5400		
04	00														
005	00	MOVE	EXPENS	TO	ERRORC									X	
006	00	WRITE	PUNCH											X	
007	00	READ	CARD											X	
008	00	GO TO													TAB010
09	00														
10	00														
11	00														
12	00														
13	00														
14	00														
15	00														
16	00														
17	00														
18	00														
19	00														
20	00														

IBM® 1401 TP PROCEDURE DESCRIPTION TXP 1401

TABLE	LINE	COL	IDENT	DIMENSIONS CND	DIMENSIONS ACT	DIMENSIONS RUL	NEXT TABLE	ERROR TABLE	ANALYST	DATE	TABLE NAME	SYSTEM NAME	IDENTIFICATION NAME
011	000	00	TABLE	02	01	03	013	STOP			CTL-BREAK		
1 3	4 6	7 8	9 13	14 15	16 17	18 19	20 23	24 27	28 46	47 56	57 66	67 74	75 80

LINE	COL	IDENT	ORDER OF RULES
000	01	ORDER	
4 6	7 8	9 13	14 15 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74

LINE	COL	IDENT	COMMENT
000	02	NOTE	ON CONTROL BREAK, WRITE TOTAL LINE

START NEXT CARD

TABLE	LINE	COL
		05
1 3	4 6	7 8

						COL 01	COL 01	COL 02	COL 02	COL 03	COL 03	COL 04	COL 04	COL 05	COL 05
4 6	7 8	9 14	15 22	23 26	27 34	35 36	37 44	45 46	47 54	55 56	57 64	65 66	67 74	9 10	11 18
LINE	COL	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
001	00		DIV	EQ	MDIV	Y		N							
002	00		DEPT	EQ	MDEPT	Y				N					
03	00														
004	00	DO	TAB050					X		X					
05	00														
06	00														
07	00														
08	00														
09	00														
10	00														
11	00														
12	00														
13	00														
14	00														
15	00														
16	00														
17	00														
18	00														
19	00														
20	00														

IBM®

1401 TP PROCEDURE DESCRIPTION

TABLE	LINE	COL	IDENT	DIMENSIONS CND	DIMENSIONS ACT	DIMENSIONS RUL	NEXT TABLE	ERROR TABLE	ANALYST	DATE	TABLE NAME	SYSTEM NAME	IDENTIFICATION NAME
013	000	00	TABLE	01	07	04	010						
1 3	4 6	7 8	9 13	14 15	16 17	18 19	20 23	24 27	28 46	47 56	57 66	67 74	75 80

LINE	COL	IDENT	ORDER OF RULES
000	01	ORDER	
4 6	7 8	9 13	14 15 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74

LINE	COL	IDENT	COMMENT
000	02	NOTE	

START NEXT CARD

TABLE	LINE	COL
		05
1 3	4 6	7 8

						COL 01	COL 01	COL 02	COL 02	COL 03	COL 03	COL 04	COL 04	COL 05	COL 05
4 6	7 8	9 14	15 22	23 26	27 34	35 36	37 44	45 46	47 54	55 56	57 64	65 66	67 74	9 10	11 18
LINE	COL	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
001	00		CODE	EQ			1		2		3		ELSE		
02	00														
003	00	DO	TABGOO			X		X		X					
004	00	MOVE	AMOUMT	TO	MBUDG	X									
005	00	SET	MYEXP	+	AMOUNT			X		X					
006	00	SET	MEXP	+	AMOUNT					X					
07	00														
008	00	MOVE	EXPENS	TO	ERRORC							X			
009	00	WRITE	PUNCH									X			
10	00														
011	00	READ	CARD			X		X		X		X			
12	00														
13	00														
14	00														
15	00														
16	00														
17	00														
18	00														
19	00														
20	00														

IBM® 1401 TP PROCEDURE DESCRIPTION TP 1401

TABLE	LINE	COL	IDENT	DIMENSIONS CND	DIMENSIONS ACT	DIMENSIONS RUL	NEXT TABLE	ERROR TABLE	ANALYST	DATE	TABLE NAME	SYSTEM NAME	IDENTIFICATION NAME
028	000	00	TABLE	00	05	01	010				EOF CARDS		
1 3	4 6	7 8	9 13	14 15	16 17	18 19	20 23	24 27	28 46	47 56	57 66	67 74	75 80

LINE	COL	IDENT	ORDER OF RULES
000	01	ORDER	
4 6	7 8	9 13	14 15 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74

LINE	COL	IDENT	COMMENT
000	02	NOTE	END ON READER

START NEXT CARD

TABLE	LINE	COL
		05
1 3	4 6	7 8

						COL 01		COL 02		COL 03		COL 04		COL 05	
4 6	7 8	9 14	15 22	23 26	27 34	35 36	37 44	45 46	47 54	55 56	57 64	65 66	67 74	9 10	11 18
LINE	COL	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
001	00	SET	EOFI	EQ	1	X									
002	00	DO	TAB750			X									
003	00	DO	TAB050			X									
004	00	SET	RAMADD	EQ	MADDR	X									
005	00	WRITE	MASTER			X									
06	00														
07	00														
08	00														
09	00														
10	00														
11	00														
12	00														
13	00														
14	00														
15	00														
16	00														
17	00														
18	00														
19	00														
20	00														

IBM® 1401 TP PROCEDURE DESCRIPTION T/P 1401

TABLE	LINE	COL	IDENT	DIMENSIONS			NEXT TABLE	ERROR TABLE	ANALYST	DATE	TABLE NAME	SYSTEM NAME	IDENTIFICATION NAME
				CND	ACT	RUL							
150	000	00	TABLE	01	18	02	DO				DEPT TOTAL		
1 3	4 6	7 8	9 13	14 15	16 17	18 19	20 23	24 27	28 46	47 56	57 66	67 74	75 80

LINE	COL	IDENT	ORDER OF RULES
000	01	ORDER	
4 6	7 8	9 13	14 15 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74

LINE	COL	IDENT	COMMENT
000	02	NOTE	

START NEXT CARD

TABLE	LINE	COL
		05
1 3	4 6	7 8

						COL 01		COL 02		COL 03		COL 04		COL 05	
4 6	7 8	9 14	15 22	23 26	27 34	35 36	37 44	45 46	47 54	55 56	57 64	65 66	67 74	9 10	11 18
LINE	COL	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
001	00		EOFI	EQ	1	Y		N							
02	00														
003	00	MOVE	MTD	TO	W	X		X							
004	00	MOVE	W	TO	DMEXPN	X		X							
005	00	MOVE	MTDB	TO	DMBUDG	X		X							
006	00	MOVE	YTD	TO	W	X		X							
007	00	MOVE	W	TO	DYEXPN	X		X							
008	00	MOVE	YTDB	TO	DYBUDG	X		X							
009	00	SET	MTDB	+	MTD	X		X							
010	00	SET	YTDB	+	YTD	X		X							
011	00	MOVE	MTDB	TO	DMVARN	X		X							
012	00	MOVE	YTDB	TO	DYVARN	X		X							
013	00	MOVE	'TOTAL'	TO	DTYPE	X		X							
014	00	MOVE	DETAIL	TO	PR	X		X							
015	00	MOVE	ZTEN	TO	MTD	X		X							
016	00	MOVE	ZTEN	TO	YTD	X		X							
017	00	MOVE	ZEIGHT	TO	MTDB	X		X							
018	00	MOVE	ZEIGHT	TO	YTDB	X		X							
019	00	WRITE	PRINT			X		X							
020	00	DO	TAB510					X							

IBM® 1401 TP PROCEDURE DESCRIPTION T P 1401

TABLE	LINE	COL	IDENT	DIMENSIONS			NEXT TABLE	ERROR TABLE	ANALYST	DATE	TABLE NAME	SYSTEM NAME	IDENTIFICATION NAME
				CND	ACT	RUL							
510	000	00	TABLE	00	07	01	DO				WRITE HDRS		
1 3	4 6	7 8	9 13	14 15	16 17	18 19	20 23	24 27	28 46	47 56	57 66	67 74	75 80

LINE	COL	IDENT	ORDER OF RULES
000	01	ORDER	
4 6	7 8	9 13	14 15 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74

LINE	COL	IDENT	COMMENT
000	02	NOTE	PRINT HEADING LINES

START NEXT CARD

TABLE	LINE	COL
		05
1 3	4 6	7 8

						COL 01		COL 02		COL 03		COL 04		COL 05	
4 6	7 8	9 14	15 22	23 26	27 34	35 36	37 44	45 46	47 54	55 56	57 64	65 66	67 74	9 10	11 18
LINE	COL	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
01	00	MOVE	DIV	TO	RDIV	X									
02	00	MOVE	DEPT	TO	RDEPT	X									
03	00	MOVE	HEAD	TO	PR	X									
04	00	SPACE	01CHAN			X									
05	00	WRITE	PRINT			X									
06	00	MOVE	HEADT	TO	PR	X									
07	00	WRITE	PRINT			X									
08	00														
09	00														
10	00														
11	00														
12	00														
13	00														
14	00														
15	00														
16	00														
17	00														
18	00														
19	00														
20	00														

IBM® 1401 TP PROCEDURE DESCRIPTION TP/1401

TABLE	LINE	COL	IDENT	DIMENSIONS CND	ACT	RUL	NEXT TABLE	ERROR TABLE	ANALYST	DATE	TABLE NAME	SYSTEM NAME	IDENTIFICATION NAME
600	000	00	TABLE	05	13	08	DO						
1 3	4	6 7	8 9 13	14 15	16 17	18 19	20 23	24 27	28 46	47 56	57 66	67 74	75 80

LINE	COL	IDENT	ORDER OF RULES
000	01	ORDER	
4	6 7	8 9 13	14 15 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74

LINE	COL	IDENT	COMMENT
000	02	NOTE	

START NEXT CARD

TABLE	LINE	COL
		05
1 3	4	6 7 8

						COL 01		COL 02		COL 03		COL 04		COL 05		COL 06		COL 07		COL 08	
4	6 7	8 9 14	15 22	23 26	27 34	35 36	37 44	45 46	47 54	55 56	57 64	65 66	67 74	9 10	11 18	19 20	21 28	29 30	31 38	39 40	41 48
LINE	COL	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
001	00		MSL	EQ	SL	Y		N								N					
002	00		MDEPT	EQ	DEPT	Y				N								N			
003	00		MDIV	EQ	DIV	Y						N								N	
004	00		MDIV	EQ	' '			N		N		N		Y		N		N		N	
005	00		FRACMO	EQ	1											Y		Y		Y	
06	00																				
007	00	DO	TAB750					X		X		X				X		X		X	
008	00	SET	MEXP	EQ	ZEROT			X		X		X				X		X		X	
009	00	SET	RAMADD	EQ	MADDR			X		X		X				X		X		X	
010	00	LAYOUT	CDCAL					X		X		X		X		X		X		X	
011	00	MOVE	DIV	TO	CDIV			X		X		X		X		X		X		X	
012	00	MOVE	DEPT	TO	CDEPT			X		X		X		X		X		X		X	
013	00	MOVE	SL	TO	CDSL			X		X		X		X		X		X		X	
014	00	LAYOUT	CDCALA					X		X		X		X		X		X		X	
015	00	SET	MADDR	EQ	MCAL			X		X		X		X		X		X		X	
016	00	WRITE	MASTER					X		X		X				X		X		X	
017	00	SET	RAMADD	EQ	MADDR			X		X		X		X		X		X		X	
018	00	READ	MASTER					X		X		X		X		X		X		X	
019	00	DO	TAB610					X		X		X		X		X		X		X	
120	00																				

IBM®

1401 TP PROCEDURE DESCRIPTION

T/P 1401

TABLE	LINE	COL	IDENT	DIMENSIONS CND	DIMENSIONS ACT	DIMENSIONS RUL	NEXT TABLE	ERROR TABLE	ANALYST	DATE	TABLE NAME	SYSTEM NAME	IDENTIFICATION NAME
610	000	00	TABLE	05	09	08	DO						
1 3	4 6	7 8	9 13	14 15	16 17	18 19	20 23	24 27	28 46	47 56	57 66	67 74	75 80

LINE	COL	IDENT	ORDER OF RULES
000	01	ORDER	
4 6	7 8	9 13	14 15 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74

LINE	COL	IDENT	COMMENT
000	02	NOTE	

START NEXT CARD

TABLE	LINE	COL
610		05
1 3	4 6	7 8

						COL 01		COL 02		COL 03		COL 04		COL 05		COL 06		COL 07		COL 08	
4 6	7 8	9 14	15 22	23 26	27 34	35 36	37 44	45 46	47 54	55 56	57 64	65 66	67 74	9 10	11 18	19 20	21 28	29 30	31 38	39 40	41 48
LINE	COL	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
001	00		MSL	EQ	SL	Y		N						N							
002	00		MDEPT	EQ	DEPT	Y				N						N					
003	00		MDIV	EQ	DIV	Y						N						N			
004	00		MDIV	EQ	' '			N		N		N		N		N		N		Y	
005	00		CHAIN	EQ	' '			N		N		N		Y		Y		Y			
06	00																				
007	00	SET	MADDR	EQ	CHAIN			X		X		X									
008	00	SET	RAMADD	EQ	MADDR			X		X		X									
009	00	READ	MASTER					X		X		X									
010	00	DO	TAB610					X		X		X									
011	00	DO	TAB620											X		X		X			
012	00	MOVE	SL	TO	MSL															X	
013	00	MOVE	DEPT	TO	MDEPT															X	
014	00	MOVE	DIV	TO	MDIV															X	
015	00	DO	TAB630											X		X		X			
16	00																				
17	00																				
18	00																				
19	00																				
20	00																				

IBM®

1401 TP PROCEDURE DESCRIPTION

T P 1401

TABLE	LINE	COL	IDENT	DIMENSIONS			NEXT TABLE	ERROR TABLE	ANALYST	DATE	TABLE NAME	SYSTEM NAME	IDENTIFICATION NAME
				CND	ACT	RUL							
620	000	00	TABLE	00	02	01	DO						
1 3	4 6	7 8	9 13	14 15	16 17	18 19	20 23	24 27	28 46	47 56	57 66	67 74	75 80

LINE	COL	IDENT	ORDER OF RULES
000	01	ORDER	
4 6	7 8	9 13	14 15 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74

LINE	COL	IDENT	COMMENT
000	02	NOTE	

START NEXT CARD

TABLE	LINE	COL
		05
1 3	4 6	7 8

4 6	7 8	9 14	15 22	23 26	27 34	35 36	37 44	45 46	47 54	55 56	57 64	65 66	67 74	9 10	11 18
						COL 01		COL 02		COL 03		COL 04		COL 05	
LINE	COL	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
01	00														
002	00	SET	RAMADD	+	1	X									
003	00	READ	MASTER			X									
04	00														
05	00														
06	00														
07	00														
08	00														
09	00														
10	00														
11	00														
12	00														
13	00														
14	00														
15	00														
16	00														
17	00														
18	00														
19	00														
20	00														

IBM®

1401 TP PROCEDURE DESCRIPTION

TABLE	LINE	COL	IDENT	DIMENSIONS CND	DIMENSIONS ACT	DIMENSIONS RUL	NEXT TABLE	ERROR TABLE	ANALYST	DATE	TABLE NAME	SYSTEM NAME	IDENTIFICATION NAME
630	000	00	TABLE	01	10	02	DO						
1 3	4 6	7 8	9 13	14 15	16 17	18 19	20 23	24 27	28 46	47 56	57 66	67 74	75 80

LINE	COL	IDENT	ORDER OF RULES
000	01	ORDER	
4 6	7 8	9 13	14 15 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74

LINE	COL	IDENT	COMMENT
000	02	NOTE	

START NEXT CARD

TABLE	LINE	COL
		05
1 3	4 6	7 8

						COL 01		COL 02		COL 03		COL 04		COL 05	
4 6	7 8	9 14	15 22	23 26	27 34	35 36	37 44	45 46	47 54	55 56	57 64	65 66	67 74	9 10	11 18
LINE	COL	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
01	00		MDIV	EQ	BLANKS	Y		N							
02	00														
03	00	MOVE	REF	TO	RM	X									
04	00	DO	TAB620					X							
05	00	SET	CHAIN	EQ	RAMADD	X									
06	00	SET	RAMADD	EQ	MADDR	X									
07	00	SET	MADDR	EQ	CHAIN	X									
08	00	WRITE	MASTER			X									
09	00	MOVE	MM	TO	RM	X									
10	00	MOVE	SL	TO	MSL	X									
11	00	MOVE	DEPT	TO	MDEPT	X									
12	00	MOVE	DIV	TO	MDIV	X		X							
13	00														
14	00														
15	00														
16	00														
17	00														
18	00														
19	00														
20	00														

IBM® 1401 TP PROCEDURE DESCRIPTION T/P 1401

TABLE	LINE	COL	IDENT	DIMENSIONS CND	DIMENSIONS ACT	DIMENSIONS RUL	NEXT TABLE	ERROR TABLE	ANALYST	DATE	TABLE NAME	SYSTEM NAME	IDENTIFICATION NAME
750	000	00	TABLE	01	25	02	DO				STORE P.R.		
1 3	4 6	7 8	9 13	14 15	16 17	18 19	20 23	24 27	28 46	47 56	57 66	67 74	75 80

LINE	COL	IDENT	ORDER OF RULES
000	01	ORDER	
4 6	7 8	9 13	14 15 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74

LINE	COL	IDENT	COMMENT
000	02	NOTE	

START NEXT CARD

TABLE	LINE	COL
		05
1 3	4 6	7 8

						COL 01		COL 02		COL 03		COL 04		COL 05	
4 6	7 8	9 14	15 22	23 26	27 34	35 36	37 44	45 46	47 54	55 56	57 64	65 66	67 74	9 10	11 18
LINE	COL	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
001	00		FRACMO	EQ	1	Y		N							
002	00														
003	00	MOVE	MSL	TO	DNO	X		X							
004	00	SET	RAMADD	EQ	RAMCAL	X		X							
005	00	MOVE	MBUDG	TO	W	X		X							
006	00	SET	W	EQ	CALBUD			X							
007	00	MOVE	W	TO	DMBUDG	X		X							
008	00	SET	MTDG	+	W	X		X							
009	00	SET	MTD	+	MEXP	X		X							
010	00	MOVE	MEXP	TO	WW	X		X							
011	00	MOVE	WW	TO	DMEXPN	X		X							
012	00	SET	WW	EQ	W	X		X							
013	00	SET	W	-	MEXP	X		X							
014	00	MOVE	W	TO	DMVARN	X		X							
015	00	SET	WW	+	MYBUDG	X		X							
016	00	SET	MYBUDG	EQ	WW	X									
017	00	SET	YTDB	+	WW	X		X							
018	00	MOVE	WW	TO	DYBUDG	X		X							
019	00	SET	YTD	+	MYEXA	X		X							
020	00	MOVE	MYEXP	TO	W	X		X							

IBM® 1401 TP PROCEDURE DESCRIPTION TP 1401

TABLE	LINE	COL	IDENT	DIMENSIONS CND	DIMENSIONS ACT	DIMENSIONS RUL	NEXT TABLE	ERROR TABLE	ANALYST	DATE	TABLE NAME	SYSTEM NAME	IDENTIFICATION NAME
750	000	00	TABLE										
1 3	4 6	7 8	9 13	14 15	16 17	18 19	20 23	24 27	28 46	47 56	57 66	67 74	75 80

LINE	COL	IDENT	ORDER OF RULES
000	01	ORDER	
4 6	7 8	9 13	14 15 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74

LINE	COL	IDENT	COMMENT
000	02	NOTE	CONTINUATION

START NEXT CARD

TABLE	LINE	COL
		05
1 3	4 6	7 8

						COL 01		COL 02		COL 03		COL 04		COL 05	
4 6	7 8	9 14	15 22	23 26	27 34	35 36	37 44	45 46	47 54	55 56	57 64	65 66	67 74	9 10	11 18
LINE	COL	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
101	00	MOVE	W	TO	DYEXPN	X		X							
102	00	SET	WW	—	MYEXPN	X		X							
103	00	MOVE	WW	TO	MYVARN	X		X							
104	00	READ	REFER			X		X							
105	00	MOVE	TYPE	TO	DTYPE	X		X							
106	00	MOVE	DETAIL	TO	PR	X		X							
107	00	WRITE	PRINT			X		X							
08	00														
09	00														
10	00														
11	00														
12	00														
13	00														
14	00														
15	00														
16	00														
17	00														
18	00														
19	00														
20	00														

CHAPTER 4

1401 Tabular Programming System Specifications

Although the basic philosophy and techniques of 1401 Tabular Programming have been explained in previous sections, detailed specifications for writing Tabular Programs are required. This section covers the basic language of 1401 Tabular Programming and the preparation of the data description table and procedure tables.

A 1401 Tabular Program is assembled in two major phases. The source program (punched cards) is first converted to Autocoder language by the TP Processor. Minimum 1401 system requirements for this procedure are:

- 1401 Processing Unit with 8K Storage, Compare Feature, and Advanced Programming Feature.
- 1402 Card Read Punch.
- 1403 Printer with 132 positions.
- 1405 Disk Storage Unit (1 access arm).

* A second 1405 access arm is used if provided.

Next the Autocoder program is converted to a machine language object program by the Autocoder processor. Minimum 1401 system requirements for this procedure are:

- 1401 Processing Unit with 4K Storage, Compare Feature, and Advanced Programming Feature.
- 1402 Card Read Punch.
- 1403 Printer with 132 print positions.
- 729 II or IV Magnetic Tape Units (4 required).

*Additional core storage is used if provided.

Minimum 1401 system requirements to run the object program are:

- 1401 Processing unit with 4K Storage, Compare Feature, and Advanced Programming Feature.
- 1402 Card Read Punch.
- 1405 Disk Storage Unit (1 access arm).

Additional features that can be used by the Object Program are:

- Additional Core Storage
- 1403 Printer with 132 print positions.
- A second access arm for the 1405.
- 729 II or IV Magnetic Tape Units.
- Multiply-Divide Feature.

Basic Language

Character set

Tabular Programming for the 1401 assumes the use of IBM Character Set H. Character Set H is shown in its collating sequence in Figure 1.

All source programs must be punched using this character set. The user can also use the other characters in the 1401 set for data, but should be careful when so doing. For example, certain characters might result in unexpected results when operated on by the MOVE record-name command since the machine move record command is employed. Also, the input/output commands will not operate according to specifications if the characters to terminate reads and writes are included within the data. Likewise, some of the special characters will not print; see the 1401 reference manual for a complete list of characters.

Written Character	Card Code	1401 BCD Code	Print as
Blank		C	
.	12-3-8	B A 8 2 1	.
)	12-4-8	C B A 8 4	⌑
+	12	C B A	&
$	11-3-8	C B 8 2 1	$
*	11-4-8	B 8 4	*
-	11	B	-
/	0-1	C A 1	/
,	0-3-8	C A 8 2 1	,
(	0-4-8	A 8 4	%
=	3-8	8 2 1	#
'	4-8	C 8 4	@
A	12-1	B A 1	A
B	12-2	B A 2	B
C	12-3	C B A 2 1	C
D	12-4	B A 4	D
E	12-5	C B A 4 1	E
F	12-6	C B A 4 2	F
G	12-7	B A 4 2 1	G
H	12-8	B A 8	H
I	12-9	C B A 8 1	I
J	11-1	C B 1	J
K	11-2	C B 2	K
L	11-3	B 2 1	L
M	11-4	C B 4	M
N	11-5	B 4 1	N
O	11-6	B 4 2	O
P	11-7	C B 4 2 1	P
Q	11-8	C B 8	Q
R	11-9	B 8 1	R
S	0-2	C A 2	S
T	0-3	A 2 1	T
U	0-4	C A 4	U
V	0-5	A 4 1	V
W	0-6	A 4 2	W
X	0-7	C A 4 2 1	X
Y	0-8	C A 8	Y
Z	0-9	A 8	Z
0	0	C 8 2	0
1	1	1	1
2	2	2	2
3	3	C 2 1	3
4	4	4	4
5	5	C 4 1	5
6	6	C 4 2	6
7	7	4 2 1	7
8	8	8	8
9	9	C 8 1	9

Figure 1 - Character Set in its collating sequence.

Names

A standard name consists of one to six alphabetic or numeric characters (not special characters) of which the first character must be alphabetic. The special characters are used to form masks for data to be edited for printing. Special characters are also used to form expressions and literals.

Tabular Programming for the 1401, similar to other types of symbolic programming, utilizes specified operands and operators to state the logic of the program in the form of one or more procedure tables. There are seven types of operands: files, records, variables, constants, expressions, literals, and special operands. Except for literals and special operands, all operands are identified by unique names, which must be defined and described in the data description table. There are five types of operators: conditional, input/output, assignment, sequence control, and special purpose. The section on Procedure Tables contains detailed specifications for the use of these operators.

Operands

The five types of operands which must be explicitly defined are shown first under the heading Data Description. Following this, other permissible operands are described.

Data Description

Tabular Programming for the 1401 utilizes one or more procedure tables to express the logic of a program. The data referred to by these decision tables is described in detail in the data description table. A blank data description form is shown on the next page (Figure 2).

IBM®

1401 TP DATA DESCRIPTION T/P/1401

TABLE	LINE	COL	IDENT						ANALYST	DATE	TABLE NAME	SYSTEM NAME	IDENTIFICATION NAME
000	000	00	TABLE	00	00	00	0000	0000					
1 3	4 6	7 8	9 13	14 15	16 17	18 19	20 23	24 27	28 46	47 56	57 66	67 74	75 80

4 6	7 8	9	10 15	16 18	19 20	21	22	23 32	33 42	43 52	53 62	63 72
LINE	COL	CLASS	NAME	INTEGER	DEC	CODE	CONT	DESCRIPTION				

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50

Figure 2

Each 1401 Tabular Program utilizes only one data description table. However, this table may require a number of pages (see sample problem, Chapter 3). The table is divided into a number of lines and columns. The first line is the table header which contains certain identifying information that is punched into a single IBM card. Subsequent lines each contain required information concerning a single operand. This information is expressed in columnar form according to established rules. The data from each of these lines is also punched into a single IBM card.

The table header contains preprinted information in columns 1 through 27. This information must always be keypunched in order to properly identify the data description table during program assembly. Columns 28 through 80 contain spaces for information that may be entered by the programmer at his discretion. Columns 75-80 contain the program identification and should be punched in every card of the program.

As stated previously, subsequent lines of the data description table are divided into columns each of which contains specified information concerning a single operand. The first column is headed LINE, and is used to number in ascending order each item of the table. Usually one or more consecutive numbers are omitted between each line number to allow for later insertion of items that may have been overlooked. The second column is headed COL for column number. The digits 00 are usually entered in this space. Exceptions to this can be constants, preassigned variables, and expressions. These can require more than one line of the table, in which case the line number remains the same and the column number is increased to 01, 02, etc., as required. The processor will check to see that the cards are in ascending order based on table number (000), line number, and column number.

The third column is headed CLASS for operand class. Of the seven operand classes utilized in 1401 Tabular Programming five may appear in the data description table. These are file, record, variable, constant, and expression. For each operand the first letter (F, R, V, C, and E) of the operand class appears in the class column.

The fourth column is headed NAME, and is used to designate the unique name for each operand. As in other types of symbolic programming, the name of an operand is usually selected for its mnemonic value. When an operand is not referenced in the program, it need not be named, and the NAME column can be left blank.

The fifth column is headed INTEGER, and is usually used to indicate the number of integer positions in the operand.

The sixth column is headed DEC for decimal, and is usually used to indicate the number of decimal positions in the operand. Exceptions are noted under FILE and RECORD headings below.

The seventh column is headed CODE, and is used to further describe the operand. In the case of FILE, the letter M for move or the letter L for load is placed in this column, depending on the mode of operation desired. For other types of operands (except records, where it is not used), the letter A for alphameric, N for numeric, or M for masked is placed in the code column.

The eighth column is headed CONT for continued, and is used in the case of operands that require more than one line on the data description table. As stated previously these types of operands are constants, preassigned variables, and expressions. The letter C is placed in this column for all lines of a continued item except the last. For example, if an expression takes two lines on the data description form, the first line would contain a C and the second line would have a blank in the CONT column.

The ninth and last column of the data description table is headed DESCRIPTION. This column is used to specify the device where a file is stored: RAMAC, reader, punch, or tape for example. It is also used to specify the exact values of constants, preassigned variables, and masks (edit words) for masked variables. Expressions are written in this portion of the form also.

Note that each named operand used in a 1401 Tabular Program must be specified in the data description table. This includes operands used to designate blank spaces on input or output media and also to specify work areas in core storage.

Each of the five types of operands that must appear in the data description table is now discussed in detail. Specifications for completing the data description for each type of operand are also given.

Files

A file is a collection of records in a single input, output or storage device. Its purpose is to associate an area in core storage with a specified input/output device, or to reserve a work area in core storage.

Each reserved area in core storage is assigned a file name. Only one record can be stored in a file area at a time. However, at different times, a maximum of 9 types of records can be stored, i.e. a single file can contain records with different formats, each of which can be described independently. Each of these records in turn can contain a maximum of 99 variables.

As an example of the organization of a file in core storage, consider a deck of 80-column cards representing inventory data. The file is named INV, and contains three different types of cards (records); ADD, DELETE and CHANGE. Each record-type contains one or more variables (fields). A description of this file might be:

File name:	INV	
Record name:	ADD	(representing item added to inventory)
Variable name:	PARTNO	
Variable name:	QOH	
Variable name:	USAGE	
Record name:	DELETE	(representing item deleted from inventory)
Variable name:	INVNO	
Variable name:	FILL	
Record name:	CHANGE	(representing change in stock status)
Variable name:	NUMBER	
Variable name:	QUSED	
Variable name:	QRCVD	

If the name INV is associated with the card reader, the instruction READ INV in a procedure table causes a card to be read. No name, other than INV, should be associated with the card reader, and INV should not refer to any other file. Furthermore, the names PARTNO, INVNO and NUMBER all refer to the part number, but different names are required because each variable must be uniquely named.

A complete description of each file, record and variable must appear in the data description. Only a single file can be associated with the card reader, punch, printer, tape units, or region of the RAMAC (more than one region can be defined, however). Work areas in core storage, and the inquiry station (see restrictions below) can have more than one file associated with them. Entries in the data description table to describe a file are:

CLASS	F
NAME	standard
INTEGER	000-999 (maximum length of any record type appearing in the file -- 000 represents 1000)
DEC	1st position V, if records are variable length (otherwise zero or blank) 2nd position 1-9, number of record types in file.
CODE	L (load mode) or M (move mode)
CONT	blank
DESCRIPTION	see Figure 3

Directly following each file description must appear descriptions of each of the one or more records that are a part of that file. Each record description entry is followed by descriptions of the one or more variable fields that are a part of that record. The formats of the different possible file entries that may appear in the data description table are illustrated in Figure 3. The exact formats shown must be used since the processor will pick up the information based on position.

IBM ® 1401 TP DATA DESCRIPTION T/P 1401

TABLE	LINE	COL	IDENT						ANALYST	DATE	TABLE NAME	SYSTEM NAME	IDENTIFICATION NAME
000	000	00	TABLE	00	00	00	0000	000					
1 3	4 6	7 8	9 13	14 15	16 17	18 19	20 23	24 27	28 46	47 56	57 66	67 74	75 80

LINE 4 6	COL 7 8	9	NAME 10 15	INTEGER 16 18	DEC 19 20	CODE 21	CONT 22	DESCRIPTION 23 52	53 72
		F		080	n	m	READER	normal pocket
		F		080	n	m	READERn	n = 1 or 2
		F		080	n	m	READERn, EOF TABnnn	
		F		080	n	m	READER , EOF TABnnn	
		F		080	n	m	PUNCH	normal pocket
		F		080	n	m	PUNCHn	n = 4 or 8
		F		132	n	m	PRINTER	
		F		nnn	vn	L/M	TAPEn IN	n = 0-5
		F		nnn	vn	L/M	TAPEn IN, EOF TABnnn	n = 0-5
		F		nnn	vn	L/M	TAPEn IN, TAPEm OUT	n = 0-5
		F		nnn	v	L/M	TAPEn IN, TAPEm OUT, EOF TABnnn	n = 0-5, m = 0-15, m≠n
		F		nnn	vn	L/M	TAPEm OUT	m = 0-5
		F		nnn	vn	L/M	TAPEm OUT, EOF TABnnn	m = 0-5
		F		nnn		L/M	RAMAC lllll, hhhhh	lllll = low Limit
								hhhhh = high Limit
		F		nnn		n	CORE	of Form FFTTS, where
								FF = Face, TT = Track, S = Sector
		F		nnn		n	INQUIRY	
		F		nnn		n	INQUIRY, OM TABnnn	

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50

FIGURE 3

The different devices associated with files are now taken in turn to point out any peculiarities or special restrictions that are imposed by the system.

READER -- Only one input file name can be associated with the READER. INTEGER length is always 080, designating the number of columns in a standard IBM card. One of the digits 1 through 9 is placed in the units position of the decimal column, indicating the number of different record types in the READER file (the format of each record must then be displayed). Code is always M for move mode. The word READER, followed by the digit 1 or 2, is written in the description column. The number indicates the pocket into which cards are to be stacked. If no number appears the "normal" pocket is used. An end-of-file (last card) condition is indicated by the letters EOF followed by a blank position, the letters TAB, and three digits. The three digits indicate the table number to which control is transferred on a GO TO basis when an end-of-file condition occurs. When the end-of-file indication is not given, and the program tries to read a card after end-of-file is reached, the program will stop.

PUNCH -- Only one output file name can be associated with the PUNCH. INTEGER length is always 080, indicating the number of columns in a standard IBM card. One of the digits 1 through 9 is written in the units position of the decimal column, indicating the number of different record types in the output file. Code is always M for move mode. The word PUNCH, followed by the number 4 or 8 is written in the description column. The number indicates the pocket into which cards are to be stacked. If no number appears, cards are stacked in the "normal" pocket. Note that pocket 8 on the punch side of the 1402 and pocket 2 on the reader side are the same pocket, hence, an inter-leaving of cards is possible. No end-of-file condition is possible with the punch.

PRINTER -- Only one output file name can be associated with the PRINTER. INTEGER length is always 132, indicating the number of print positions required for the 1403 Printer. One of the digits 1 through 9 is written in the units position of the decimal column, indicating the number of record types in the file. Code is always M for move mode. The word PRINTER is written in the description column.

TAPE -- Magnetic tape units can be used either as input or output devices. Therefore, each tape unit used in the program must be designated as either an input or an output device, and only one input or output file name can be associated with each tape unit. However, the same file name can be associated with both an input and output tape unit. Integer length is 000 through 999 (where 000 indicates 1000), indicating the maximum length of any record type in the file. The letter V is written in the tens position of the decimal column if variable length records appear in the tape file. One of the digits 1 through 9 is written in the units position of the decimal column, indicating the number of record types in the tape file. Code can be either M for move mode or L for load mode, depending on the mode of operation desired. Move mode is used when word marks are not to appear on the input or output data. Load mode is used when word marks are to read in as written out with the word marks associated with the fields.

For an input tape unit, the letters TAPE followed by one of the digits 0 through 5, a blank position, and the letters IN are written in the description column. The digit indicates the number of the tape unit to be assigned. An end-of-file condition is indicated in the same manner as for the card reader.

For an output tape unit, the word TAPE followed by one of the digits 0 through 5, a blank position, and the letters OUT are written in the data description column. For an output tape file, end-of-file is indicated in the same manner as for the reader and input tape files. The same file name may be used for both an input and an output tape unit, but different tape units must be specified.

RAMAC -- Each continuous region in RAMAC can be designated as an independent file, each with a different file name. Only one file name can be associated with each RAMAC region, but this name can be used for both input and output operations. INTEGER length, designating the length of records within a file, must be either 200 or

000 (1000) for move mode, or 176 or 880 for load mode. One of the digits 1 to 9 is written in the units position of the decimal column, indicating the number of different record types in the file. Code can be either M for move mode, or L for load mode. The word RAMAC followed by a blank position, a 5 digit number, a comma, and another 5 digit number are written in the description column. The two 5 digit numbers represent the beginning and ending addresses respectively of each RAMAC region. These numbers are of the format 'fftts', where ff = face, tt = track and s = sector. No end-of-file condition is possible for RAMAC operations.

CORE -- All intermediate values, utilizing work areas in core storage, must be part of a record, which in turn is part of a file. There may be any number of files in core storage, each file having a separate area of storage assigned to it. INTEGER length (000 through 999) specifies the maximum length of any record type in the file. One of the digits 1 to 9 is written in the units position of the decimal column, indicating the number of different record types in the file. The code column is left blank, because mode is not appropriate for core files. The word CORE is written in the description column. An end-of-file condition is not possible for intermediate core storage.

INQUIRY -- Two types of input and one type of output operation are possible with the 1407 Console Inquiry Station. Each of these types of operation may be designated by a distinct file name. INTEGER length (000 through 999) designates the maximum length of any record type in the file. One of the digits 1 through 9 is written in the units position of the decimal column to indicate the number of record types in each file. Code can be either M for move mode or L for load mode.

For the first type of input operation, the word INQUIRY is written in the description column. When the appropriate file name for this type of operation appears in the program, the INQUIRY latch is tested. If the latch is not on, the program stops and waits for the latch to be turned on, manually, by the operator. At this time, data required by the program may be typed by the operator.

For the second type of input operation, the word INQUIRY followed by a comma, the word ON, a blank position, the word TAB, and a 3 digit number is written in the description column. This type of operation is indicated when the operator may desire to make an inquiry of the program. In this case, the program is written to periodically test to see if the inquiry latch is ON. When it is not, the program continues, when it is ON the program transfers on the GO TO basis to the table number indicated by the 3 digit number following the word TAB.

For the single type of output operation allowed, the word INQUIRY is written in the description column. This allows the 1407 to be used as an output typewriter.

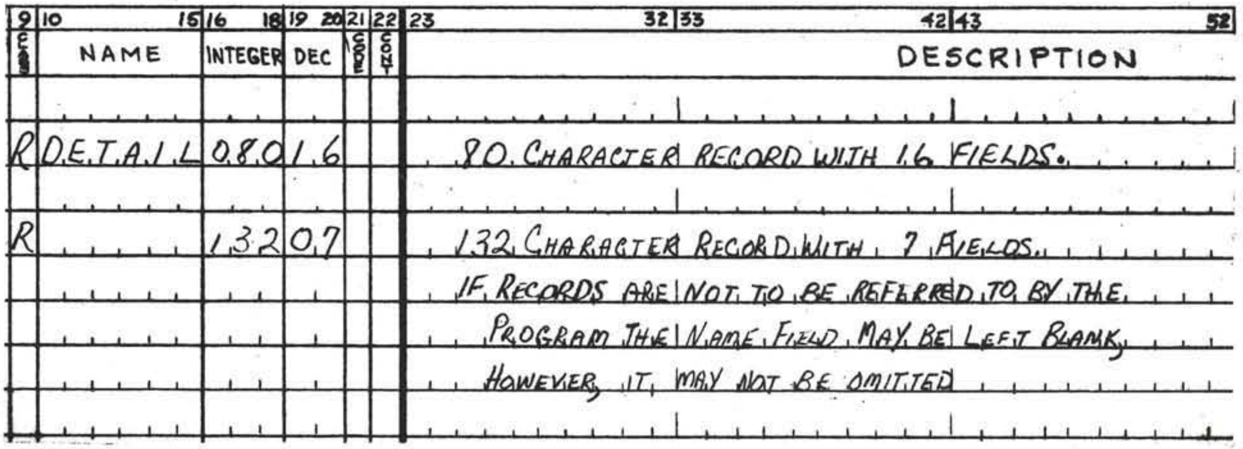
Records

A record consists of one or more related variable fields. Typical records are 80 column cards, 132 character print lines, magnetic tape records, RAMAC records, and reserved work areas in core storage. Although a file can provide only one record at a time, a number of record types can be represented by a file name, each record type identified by a unique standard name. The entries made in the data description table to describe a record are:

CLASS	R
NAME	standard name
INTEGER	000-999 (number of characters per record, including blank fields -- 000 represents 1000)
DEC	01-99 (number of variables per record, including blank fields)
CODE	blank
CONT	blank
DESCRIPTION	blank

When a file contains more than a single record type, each of the different records must be described along with the variables of which it is composed. An illustration of this was given under the FILE heading, where a card file contained three types of cards -- CHANGE, DELETE and ADD -- each with different formats. Each format is shown to enable the processor to suitably address these fields when they are used in procedures. Also, the processor uses these descriptions to correctly place the void marks in the core areas associated with the record. The processor automatically sets up word marks for the last (or only) record defined for each file. The LAYOUT record-name command, which will be described under Procedure Tables, is used to clear old word marks and to set new word marks corresponding to the desired record format, during the running of the object program.

The following illustration shows the format of the RECORD entry:



9	10 15	16 18	19 20	21	22	23 32 33 42 43 52
CLS	NAME	INTEGER	DEC	CODE	CONT	DESCRIPTION
R	DETAIL	080	16			80 CHARACTER RECORD WITH 16 FIELDS.
R		132	07			132 CHARACTER RECORD WITH 7 FIELDS.
						IF RECORDS ARE NOT TO BE REFERRED TO BY THE
						PROGRAM THE NAME FIELD MAY BE LEFT BLANK,
						HOWEVER, IT MAY NOT BE OMITTED

The description for each record type must be followed by a description for each of the variable fields within that record type. Note that every variable field must be a part of a record, including blank fields between data fields.

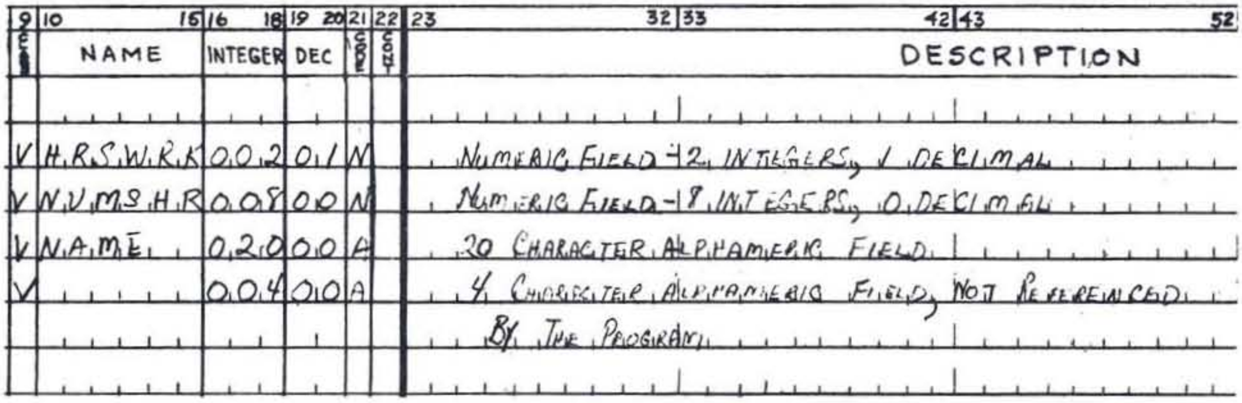
Variables

Variables are the operands most used in any data processing program. Determination of the values of these variables is usually the purpose of the program. Variable names may represent input, intermediate, or output values, depending on how these values are developed in the program. As noted previously, each variable must be uniquely named and must be a part of a record, which in turn is a part of a file.

There are three types of variables used in 1401 Tabular Programming ; unassigned, preassigned, and masked variables. Unassigned variables are used when the value is developed through an input operation or by calculation. Preassigned variables are similar to unassigned variables, except that they can be assigned an initial value in the description column of the data description table. Masked variables are used for printed output of numeric data, under the control of a mask (edit word) that is written in the description column of the data description table.

Unassigned Variables -- An unassigned variable can be either numeric with a maximum of 12 integer and 6 decimal positions, or alphameric, with a maximum of 999 positions, including blanks. Entries in the data description table to describe an unassigned variable are:

CLASS	V
NAME	standard name
INTEGER	numeric 000-012, alphameric 001-999
DEC	numeric 00-06, alphameric 00
CODE	N (numeric), A (alphameric)
CONT	blank
DESCRIPTION	blank

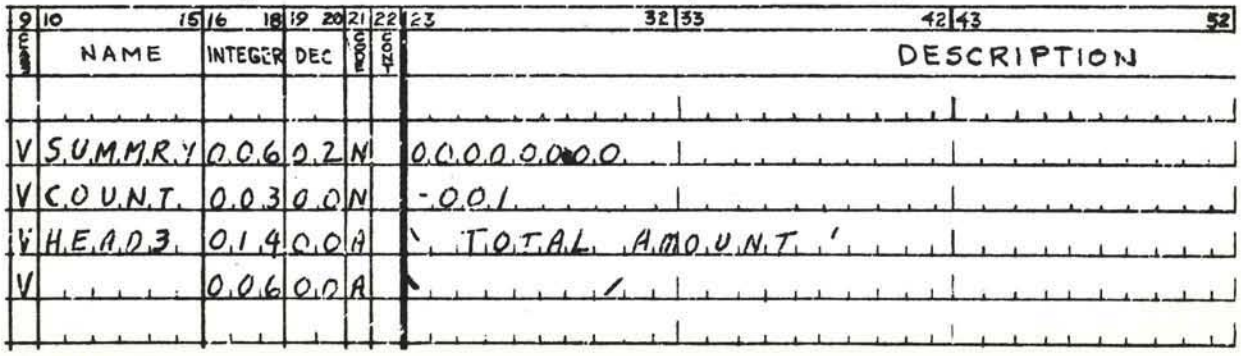


9	10 15	16 18	19 20	21	22	23 32 33 42 43 52
CLASS	NAME	INTEGER	DEC	CODE	CONT	DESCRIPTION
V	HRSWRK	002	01	N		NUMERIC FIELD - 2 INTEGERS, 1 DECIMAL
V	NUMSHR	008	00	N		NUMERIC FIELD - 8 INTEGERS, 0 DECIMAL
V	NAME	020	00	A		20 CHARACTER ALPHAMERIC FIELD
V		004	00	A		4 CHARACTER ALPHAMERIC FIELD, NOT REFERENCED
						BY THE PROGRAM

Preassigned Variables -- A preassigned variable is used in the same way as an unassigned variable, except that an initial value is assigned to it. If numeric, the initial value and values that the variable assumes during the program are limited to 12 integer and 6 decimal positions. The initial value appears in the data description with its sign (to the left of the value) and decimal point. The decimal point must be written unless the value is an integer. When unsigned, a positive value is assumed. If alphameric, both the initial and assumed values of the variable are limited to 999 positions, including blanks. The initial value of an alphameric variable appears in the data description enclosed in quotation marks. Entries in the data description table to describe a preassigned variable are:

CLASS	V
NAME	standard name
INTEGER	numeric 000-012, alphameric 001-999
DEC	numeric 00-06, alphameric 00
CODE	N (numeric), A (alphameric)
CONT	C, if preassigned value will not fit on a single line and an additional card(s) is required. Otherwise, blank.
DESCRIPTION	Actual preassigned value. If alphmeric, enclose in quotes.

If an alphameric value will not fit on a single line, each line up to but not including the last must have a C in the continuation column, all lines must have the quote mark in the first and last positions of the description field.



9	10 15	16 18	19 20	21	22	23 32	33 42	43 52
CLASS	NAME	INTEGER	DEC	CODE	CONT			DESCRIPTION
V	SUMMRY	006	02	N		000000.00		
V	COUNT	003	00	N		-001		
V	HEAD3	014	00	A		' TOTAL AMOUNT '		
V		006	00	A		' '		

Masked Variables -- A masked variable is an unassigned numeric variable with a maximum of 12 integer and 6 decimal positions. Although the variable to be masked must be numeric, the mask itself (edit word) associated with this variable is limited to a maximum of 48 alphameric characters enclosed in quotation marks, and appearing in the description column of the data description table. Whenever information is assigned to this variable it will be operated on by the mask in the description area. The process first moves in the mask and then the value to be assigned. This is used primarily in developing a print line. Entries in the data description table to describe a masked variable are:

CLASS	V
NAME	standard name
INTEGER	001-048 (length of mask)
DEC	00
CODE	M
CONT	blank
DESCRIPTION	Actual mask (edit word) enclosed in quotes

In forming masks, certain characters have special significance. These characters and the functions they perform are listed below. Further information on editing can be referenced in the 1401 Reference Manual, A 24-1403-4.

Control Character — Function

b (blank) — This is replaced with the character from the corresponding position of the sending field.

0 (zero) — This is used for zero suppression, and is replaced with a corresponding character from the sending field. Also the right-most "O" in the control word indicates the right-most limit of zero suppression.

. (period) — This is undisturbed in the punctuated data field, in the position where written. It functions as a significant character.

, (comma) — This is undisturbed in the punctuated data field, in the position where written, unless zero suppression takes place, and no significant numeric characters are found to the left of the comma.

CR (credit) — This is undisturbed if the data sign is negative. It is blanked out if the data sign is positive. Can be used in body of control word without being subject to sign control.

- (minus) — This is the same as CR.

& (ampersand) — This causes a space in the edited field. It can be used in multiples.

* (asterisk) — This can be used in singular or in multiple, usually to indicate class of total.

$ (dollar sign) — This is undisturbed in the position where it is written.

9	10 15	16 18	19 20	21	22	23 32 33 42 43 52
	NAME	INTEGER	DEC	CODE	CONT	DESCRIPTION
V	TOTAL	013	00	M		'$, , 0b.bb'
V	NET	009	00	M		' , . -'

Constants

A constant is a named operand whose value is assigned in the data description and cannot be changed. A numeric constant can have a maximum of 12 integer positions and 6 decimal positions plus sign and decimal point. If unsigned, a positive value is assumed. An alphameric constant can have a maximum of 999 positions, including blanks, and must be enclosed in quote marks in the data description. Examples of named constants might be TXRATE, SSRATE, and PI. Constants should be entered in the data description form along with the description of intermediate variables and expressions, rather than with the description of input/output files. Entries in the data description table to describe a constant are:

CLASS	C
NAME	standard name
INTEGER	numeric 000-012, alphameric 001-999
DEC	numeric 00-06, alphameric 00
CODE	N (numeric), A (alphameric)
CONT	C, if alphameric exceeding 48 positions. Otherwise, blank.
DESCRIPTION	Actual value of constant. If alphameric, enclose in quotes.

	NAME	INTEGER	DEC			DESCRIPTION
C	PI	001	04	N		3.1416
C	DISCNT	000	02	N		-.85
C	MESSAG	019	00	A		'TYPE 16 ERROR'
C	HDR1	132	00	A	C	' # ØPERAND 1 ØPERATØR OPER'
					C	'AND 2 RULE 1 RULE 2 RULE 3 RULE 4'
						' RULE 5 '

Expressions

An arithmetic expression is a combination of operands and arithmetic operators, which when executed yield a numeric value. The operands can be previously defined unsigned numeric literals, numeric constants, or numeric variables. Numeric literals may have up to 12 integer positions and 6 decimal positions. A maximum of 26 operands can appear in an expression. The operators + - * / (representing addition, subtraction, multiplication and division respectively) can be used.

Parentheses serve to delimit the effect of any operator, but where not specified by parenthesis the hierarchy of operations (from left to right through an entire expression) is first, multiplication and division, and second, addition and subtraction. An equal number of left and right parentheses must be used. All operands, operators, and parentheses should be separated from each other by one or more blanks. An example of typical expression is:

A + B * (C - D)

This would be executed -- subtract D from C, multiply by B, and add to A. FORTRAN rules of hierarchy and parenthesizing are used in determining the order of evaluating an arithmetic expression.

The name of an expression, together with the actual terms of the expression, appears in the data description. Where more than one line is required to write an expression, an operand must not be split between two lines. An expression is not evaluated unless it is called by the appearance of its name in a procedure table. Then the expression is evaluated, using up to 12 integer and six decimal positions, as required. After this, sufficient low-order decimal positions are truncated to satisfy the number of decimal positions (maximum of 6) specified for the expression name. Sufficient high-order integer positions are truncated to satisfy the number of integer positions (maximum of 12) specified for the expression name. For example, the computed value of an expression, for which eight integer and four decimal positions are specified, might be:

987654321.123456

After truncation, the value appears as:

87654321.1234

The example shows that care must be exercised to specify sufficient integer and decimal positions.

Entries in the data description table to describe an expression are:

CLASS	E
NAME	standard name
INTEGER	000-012 (integer length of computed value of expression)
DEC	00-06 (decimal length of computed value of expression)
CODE	N
CONT	C, if length of expression exceeds 50 positions. Otherwise, blank
DESCRIPTION	Actual expression.

	NAME	INTEGER	DEC			DESCRIPTION
E	ADCALC	005	00	N		ACCTNO - (ACCTNO / RANGE * RANGE) + BASEAD
E	DIST	004	00	N		RATE * TIME
E	NET	004	02	N		GROSS * .82 - DPNDTS * 13.00
E	PRICE	006	02	N		CØST * QUANTY
E	Y	008	06	N		A * (X + B * (X + C))

Other Operands

As mentioned earlier, there are two other types of operands that need not be explicitly defined in the data description table. The first are literals (actual values, rather than names of values) and the second, special operands, which are part of the system vocabulary and have special meaning.

Literals

A literal is a signed or unsigned numeric value or group of alphameric characters that are used in the program procedure table. Numeric literals can also be used in the data description table in defining an expression. A numeric literal can have a maximum of six positions plus sign and decimal point in a procedure table, and a maximum of 12 integers and 6 decimals in defining expressions. If unsigned, a positive value is assumed; the sign precedes the literal value. The decimal point is used to show the assumed decimal position. An alphameric literal can have a maximum of six positions and must be enclosed in quote marks (designated by an apostrophe in character set H). Examples of literals are:

+123.45

-987.654

456789

'STKOH'

'4THQ'

'PLANT3'

All special characters of set H can be used to form a literal except the quote mark, which is reserved for designation of alphameric literals.

Special Operands

Six special operands are reserved for specific conditions and actions in procedure tables. These operands may not be used for any other purpose, and do not appear in the data description table. The special operands and their purposes are:

LINE(S)	1403 carriage line spacing
CHAN	1403 carriage skipping
RAMADD	1405 address conversion
TAB	abbreviation of "table"
TABSTOP	stop the program
ELSE	unconditional rule

Detailed instructions for the use of the special operands appear in the section titled Procedure Tables.

Procedure Tables

In tabular programming, a procedure table is an organized means for expressing the logic associated with the solution to a problem. With other programming methods this logic is usually expressed in the form of a flow chart, after which the program is coded in some symbolic language; hence two steps are required. In tabular programming only one step is required since the procedure tables serve both as a flow chart and the resulting program.

A solution to a problem can usually be described by indicating sets of logical conditions and the actions that result. In tabular programming, these conditions, actions, together with certain other information is recorded in a procedure table. The general form of a procedure table is shown in the illustration.

Table Header	
Condition Stub	Condition Entry
Action Stub	Action Entry

The general form (Figure 4) should be compared with an actual procedure table to identify the elements of the table. The table header requires certain information, portions of which cannot be provided until the rest of the table has been completely written. The condition and action sections contain the logic of the problem to be solved. These sections are further divided into stub and entry portions. Note that the double line separating

IBM® 1401 TP PROCEDURE DESCRIPTION TP 1401

TABLE	LINE	COL	IDENT	DIMENSIONS CND	DIMENSIONS ACT	DIMENSIONS RUL	NEXT TABLE	ERROR TABLE	ANALYST	DATE	TABLE NAME	SYSTEM NAME	IDENTIFICATION NAME
	000	00	TABLE										
1 3	4 6	7 8	9 13	14 15	16 17	18 19	20 23	24 27	28 46	47 56	57 66	67 74	75 80

LINE	COL	IDENT	ORDER OF RULES
000	01	ORDER	
4 6	7 8	9 13	14 15 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74
LINE	COL	IDENT	COMMENT
000	02	NOTE	

START NEXT CARD

TABLE	LINE	COL
		05
1 3	4 6	7 8

						COL 01		COL 02		COL 03		COL 04		COL 05	
4 6	7 8	9 14	15 22	23 26	27 34	35 36	37 44	45 46	47 54	55 56	57 64	65 66	67 74	9 10	11 18
LINE	COL	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
01	00														
02	00														
03	00														
04	00														
05	00														
06	00														
07	00														
08	00														
09	00														
10	00														
11	00														
12	00														
13	00														
14	00														
15	00														
16	00														
17	00														
18	00														
19	00														
20	00														

FIGURE 4

conditions and actions is not shown on the form. This allows the number of actual condition rows to determine the location of the double line.

Limited and Extended Entry

There are two basic ways of writing condition and action rows, limited entry and extended entry. Limited entry, (when a condition or action is written entirely in the stub section of the table) is generally used when the condition or action is common to many rules. For example, if for every rule of a table we want to read a card, the action is written in the stub and marked with an X in each entry box. Likewise if a particular condition applies to a number of rules. The condition entry portion of the table is then marked with a Y (yes) or N (no) to indicate whether the condition must be true or not true to satisfy that condition. If a condition is not pertinent to a rule, the OP column is left blank. Specific actions to be executed for a rule are indicated by writing the letter X in the first position of the OP column. The limited entry technique is illustrated by this simplified version of a procedure table. This table is written entirely in limited entry form, in the 1401 Tabular Programming System, it is possible to mix limited and extended entry form.

	Rule 1	Rule 2	Rule 3	Rule 4
A EQ B	Y	Y		N
C GR D	Y		N	Y
E LE F	N	N	Y	N
MOVE X TO Y	X	X		
READ CARD	X		X	X
GO TO TAB 123		X		X

In a limited entry table, all operands and operators appear in the condition and action stub portion of the table. The only entries in the entry section of the table are the letters Y and N, indicating required conditions, and the letter X, indicating required actions.

In extended entry form, the second operand of a condition or an action can be written in the entry portion of the table. Thus, a single row can contain a number of different conditions or actions. For example, in a condition row the stub might contain an operand which is to be compared with a number of values (Type code = 1, 2, 3, 4. . .). The table below shows a combination of limited and extended entry form.

STUB				ENTRY					
				RULE 1		RULE 2		RULE 3	
OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
	METAL	EQ			'ALUM'				'COPR'
	VOLTS	GR	110			Y		N	
	A	GR			B		C		
READ	MASTER			X				X	
MOVE	BLANKS	TO			TOTAL		SUBTOT		
GO TO					TAB023		TAB264		TAB116

Table Header

The first three lines of a procedure table are reserved for table header information. With certain exceptions, columns 1-27 of the first line must be completed. Columns 28-80 can be completed at the discretion of the programmer, as can the second and third lines of the header. Each of the three header lines is key punched into an IBM card.

A three-digit table number (e.g. 001, 012, 987) is written in columns 1-3 of the first line. Because the first table executed in a program is table 001, there must always be a table 001. Any number up through 999 can be used for subsequent tables. Although not shown on the form, table number must be key punched in columns 1-3 of all cards associated with a procedure table.

Pre-printed information appears in columns 4-13. Columns 14-19 specify the dimensions (number of condition rows, actions rows, and rules) of the table. Usually, the table proper must first be completed to determine its dimensions.

With certain exceptions columns 20-23 of the first line specify the NEXT TABLE to be executed. This is indicated by a left-justified three-digit number. If a table is executed as a result of a DO command (see Action Operators) appearing in one or more other tables, the word DO is written in columns 20-21. This causes control to be returned to the originating table after execution. If a table terminates a program, the word STOP is written in columns 20-23. Except for a DO-type table, the next table to be executed may also be indicated by a GO TO command as the last action of one or more rules of a table. In this case, the next table entry is ignored.

Columns 24-27 of the first line specify an ERROR TABLE. Should none of the sets of conditions in a table be satisfied, an error in programming logic is indicated. In this case, the number of the table in error is printed and the program transfers to the error table (a left-justified three-digit number). If a program is not to continue after an error, the word STOP is written in columns 24-27. An ERROR TABLE may not be indicated in a closed table, if an error occurred, there is the error type out and the program returns to the table where the DO occurred. However, for a non DO-type table, transfer to an error table can be effected by a GO TO command as the last action under an ELSE condition entry (see Condition Operands). Although

ELSE serves other purposes, its appearance as a condition of a table precludes the need for an error table entry.

As stated previously, columns 28-80 of the first line are used at the discretion of the programmer. However, this information can be useful when a printed program listing is made. Note that columns 75-80 are used for IDENTIFICATION NAME, and that these columns are reserved for all cards of the procedure and data description tables.

The second line of the procedure table header is used to specify the order in which the rules in the table proper are to be considered. Columns 4-13 contain pre-printed information. Columns 14-73 allow a maximum of 30 two-digit (COL) numbers to be written. The rules are considered in the sequence these numbers are written. Thus, rules that are written in some logical order can be considered in a different order. This is done by putting the high frequency rules to the left to improve program efficiency. If more than 30 rules are required in a table, columns 14-73 cannot be used; the table itself must be arranged for efficiency. If no order of rules is indicated, the rules are considered in sequence from left to right, as written in the table proper. Column 74 is left blank, and columns 75-80 are reserved for IDENTIFICATION NAME.

The third header line is reserved for comments. Column 4-13 contain pre-printed information. Any comments that the programmer desires can be entered in columns 14-74. Columns 75-80 are reserved for IDENTIFICATION

NAME. If more than one comment card is desired, subsequent cards can be used, but the number in columns 7-8 must be incremented for each subsequent card.

Stub and Entry Sections

Each line of the table proper is key punched into an IBM card. The table number (1-3) is punched in every card. Line numbers (4-6) must also be punched for every card (must be in ascending sequence) as well as column numbers (7-8). Column number will be 00 for the first card of each line in the table proper. Because of the 80-column size of an IBM card, the stub section and only four rules can be punched into a single card. If more rules are required, additional cards can be punched. Each additional card has space for six rules (10 card columns per rule) plus identifying information in columns 1-8 and 75-80. The standard procedure table form provides space for 13 rules. Rules 5-10 are punched into the second card, and rules 11-13 are punched into the third card. In each of these cards, table number is punched in columns 1-3, and line number in columns 4-6. The first rule number (COL) in each additional card is punched in columns 7-8 (05 for the second card, 11 for the third card, etc.).

If more than 13 rules are required, the form can be extended by using additional forms. Rules 14-16 are continued (after 11-13) in the third card. Rules 17-22 are punched in card columns 9-68 of the fourth card, rules 23-28 in the fifth card, etc., up to a maximum of 64 rules. As noted previously, card columns 1-8 of each additional card must always be punched. Columns 7-8 are punched with the first rule number appearing in that card (17 for the fourth card, 23 for the fifth card, etc.).

The number of condition rows and action rows in the stub and entry sections of the procedure table can exceed the 20 for which space is provided. In this case, the table proper section of additional forms can be used. Note that the pre-printed line numbers (card columns 4-6) must be changed to show the numbers 021 and higher. A new header is not permitted, but all other parts of the form are used exactly as for the original form.

Because of limitations in storage and in the 1401 tabular language processor the following restrictions apply:

1. Maximum of 999 tables.
2. Maximum of 64 rules per table.
3. Maximum of 54 condition rows plus action rows (line numbers) per table.
4. The larger the number of rules per table, the smaller the number of conditions plus actions that are allowed, as shown below.

No. of Rules	No. of Conditions & Actions	Rules	No. of Conditions & Actions	Rules	No. of Conditions & Actions
1	54	10	14	23-25	6
2	42	11	13	26-30	5
3	34	12	12	31-37	4
4	29	13	11	38-47	3
3	25	14	11	48-64	2
6	22	15	10		
7	19	16-17	9		
8	17	18-19	8		
9	15	20-22	7		

Writing a Program

A tabular program is written by entering specified operands and operators in the columns of the stub and entry sections of one or more procedure tables. Whether limited entry, extended entry, or a combination of both types is used, at least one column in the entry section (to the right of the double line) must be used. Each such column, together with matching entries in the stub section, constitutes a rule. A rule is composed of a unique set of conditions and actions; the actions are to be performed if all the conditions in that rule are satisfied. In some special cases a rule can consist of only conditions or only actions.

With any specified set of input data, only one set of conditions can be satisfied; and therefore only one set of actions can be executed. Note that it is possible that no set of conditions is satisfied. This can be the result of a logical error in writing the program, or it can be the result of deliberate action on the part of the programmer. For example, the presence of invalid data can be checked. If none of the conditions are satisfied, the data is invalid, and the program transfers to the error table. If any set of conditions is satisfied, the data is valid and transfer to the next table occurs. In this case, the 'next table' contains the next logical steps of the program.

The various types of operands that can be used in 1401 tabular programming have already been discussed. Specific rules govern the use of these operands in condition and action statements of a procedure table.

The operators that can be used in 1401 tabular programming are divided into three categories. Conditional operators, which can be used only in the condition section of a procedure table, action operators, which can be used only in the action section, and special operators, which can be used in certain condition and action statements.

The conditional operators and their meanings are:

LR	Less than (lesser)
LE	Less than or equal to
GR	Greater than
GE	Greater than or equal to
EQ	Equal to
UN	Unequal

Action operators are further divided into three classes; input/output, assignment, and sequence control.

The input/output operators and their meanings are:

READ read a file from card reader, tape, RAMAC or Inquiry

WRITE write a file on printer, card punch, tape, RAMAC or Inquiry

CLOSE rewind input tape, or write tape mark and rewind output tape.

Assignment operators are used to control the handling of data and to perform certain arithmetic operations.

SET. . . EQ. . sets the first data field equal to the value of the second data field

MOVE. . . TO. . . moves the value of the first data field to the location of the second data field -- similar to SET. . . EQ. . . except sending and receiving fields are reversed

SET. . . +. . . increases the value of the first data field by the value of the second data field

SET. . . -. . . decreases the value of the first data field by the value of the second data field

MOVE. . . +. . . similar to SET. . . +. . . except fields are reversed -- hence second field value is increased by first field value

MOVE...-... Similar to SET...-... except fields are reversed

LAYOUT controls the setting of field word marks in record areas.

Sequence control operators are used to control the sequence of the program.

GO TO transfers control from one procedure table to another

DO similar to GO TO, except that at the conclusion of the second table the program returns to the first procedure table

Special operator to control the 1403 Printer Carriage.

SPACE causes line spacing, and skipping of the printer carriage under carriage tape control

Detailed specifications for writing the conditions and actions of a tabular program, using the operands and operators previously described, are given in the following sections.

Conditions

In a procedure table, a rule is formed from one or more conditions together with one or more resultant actions. All conditions within a rule must be satisfied for the actions of that rule to be executed. A condition tests the value of one operand against the value of another operand to determine whether a specified relationship is satisfied. The general form of a conditional statement is:

Operand 1 Conditional Operator Operand 2

This general form can be expanded to show the exact locations of the operands and the operator in the stub and entry portions of a procedure table for both limited and extended entry formats.

	Stub			Entry	
Operator	Operand	Operator	Operand	Operator	Operand
	1	C	2	Y	
	1	C	2	N	
	1	C			2
	1			C	2

The numerals 1 and 2 designate operand 1 and operand 2. The letter C designates the conditional operator. The letters Y and N (yes and no) are extensions of the conditional operator used only in the limited entry form. The first two forms shown are limited entry, and the last two extended entry.

Each operand can be:

1. Numeric or alphameric literal.
2. Numeric or alphameric named constant.
3. Named expression.
4. Numeric or alphameric named variable.

However, the following restrictions apply:

1. Both operands cannot be expression names.

2. Both operands cannot be literals. Both operands cannot be constants names. One operand cannot be a constant name if the other operand is a literal.
3. Both operands must be numeric or both must be alphameric.
4. A masked variable cannot be used as either operand.
5. All operands are written left justified in the spaces provided.

The procedure for comparison of numeric values is different from that for alphameric values.

For numeric values, if the number of decimal and integer positions for both values agree, a simple comparison of the values is made. Otherwise, the value with the lower number of decimal positions is zero filled in its low order positions until the number of decimal positions in both values agree. The value with the lower number of integer positions is zero filled in its high order positions until the number of integer positions for both values agree. For example:

Operand 1	87654.32
Operand 2	321.98765

Three low-order zeros are added to Operand 1, and two high-order zeros are added to Operand 2, giving:

Operand 1	87654.32000
Operand 2	00321.98765

The values are now compared, position by position. Note that all negative numbers are considered smaller than positive numbers, all zeroes are equivalent

$(\overset{+}{0} = \overset{-}{0} = 0)$. Furthermore, when two negative numbers are compared, the number with the smaller absolute value is considered the larger number.

For alphameric values, if the number of positions in both values agrees, a simple comparison is made. Otherwise, the value with the smaller number of positions is filled with blanks in its low order positions until the number of positions in both values agree.

For example:

Operand 1	ABCDEF
Operand 2	ABCD

Two blanks are added to the right of Operand 2, giving:

Operand 1	ABCDEF
Operand 2	ABCDbb

Comparison of the values is then made, position by position.

The special operand **ELSE** is used to designate a condition that is always satisfied when none of the other conditions in a procedure table are satisfied. **ELSE** can only appear in the last rule to be examined, and is always the only condition in that rule. It is usually followed by one or more actions, the last of which may be a GO TO command that transfers control to an error table or a regular processing table.

Actions

As noted previously, a rule usually consists of one or more conditions followed by one or more actions that are executed if all the conditions are satisfied. These actions will be executed in the sequence written. An action is used to assign values, obtain input data, provide output data, and control the sequence of the program.

An action statement can include either one or two operands. When a one-part operator (e.g., READ...) is used only one operand is required. When a two-part operator (e.g., MOVE... TO...) is used two operands are required. Furthermore, an action can be represented in either limited entry or extended entry form. The general forms in which an action can be written are:

STUB				ENTRY	
Operator	Operand	Operator	Operand	Operator	Operand
s	1			X	
s					1
d	1	i	2	X	
d	1	i			2
d	1			i	2

In the illustration, the numerals 1 and 2 represent Operand 1 and Operand 2, respectively. The letter s represents a single (one-part) operator. The letters d and i represent the designator and the indicator portions, respectively, of a two-part operator. Note that the first and third examples are limited entry form, utilizing the letter X as an extension of the action operator, while the remaining examples are in extended entry form. Detailed specifications for the use of each of the action operators are now given.

Input/Output Operators

READ -- This operator is used in conjunction with a single operand to obtain data from an input device (card reader, RAMAC, magnetic tape unit, or inquiry station) and store it in a predetermined location in core storage. The operand must always be a file name; this represents the input device and the associated area in core storage, as defined in the data description table. Complete data description is important because it controls such things as the area reserved in core storage, mode of data movement, stacker selection, and end-of-file procedure. The general forms of the READ action is shown.

READ	MASTER			X	
READ					MASTER

Note that the operand (file name) MASTER can appear in either of two locations. The first example shows limited entry form, the second, extended entry form.

When the operand in a READ action represents a file in the card reader, the next card is read. When the operand represents a file in a magnetic tape unit, the next tape record is read. When the operand represents a file in RAMAC a record is read from the disk. However, previously the proper RAMAC address must have been assigned to the special system location called RAMADD. This must be accomplished by one of the assignment operators.

SET	RAMADD	EQ	Expression name or variable name		
MOVE	expression name or variable name	TO	RAMADD		

Before any read command can reference a RAMAC file, some value (computed or determined in some fashion) must be placed in RAMADD. RAMADD is defined as a five digit integer value of the form ff tts--where ff= face, tt= track and s = sector. If the programmer wishes to reference the file by the chaining techniques he must write the procedure himself, since no standard chaining method is automatically supplied.

When there are no more cards in the reader or when a tape mark is sensed, an end-of-file condition exists and a transfer of control is effected to the table specified in the data description. If none is written, a stop will occur. WRITE -- This operator is used in conjunction with a single operand to provide data to an output device (printer, card punch, RAMAC, magnetic tape unit, or inquiry station) from a predetermined location in core storage. The operand must always be a file name that represents the output device and the associated area in core storage, as defined in the data description table. Complete data description is important because it controls such things as the area reserved in core storage, mode of data movement, stacker selection and end-of-file (for tape only). The general form of the WRITE action is shown.

WRITE	PAYREG			X	
WRITE					PAYREG

Note that the operand (file name) PAYREG (pay register) can appear in either of two locations. The first example shows limited entry form, the second, extended entry form.

When the operand in a WRITE action represents a file for the printer, a line is printed. When the operand represents a file for the card punch, a card is punched. When the operand represents a file for a magnetic tape unit, a record is written on magnetic tape. When the operand represents a file in RAMAC a record is written on disk utilizing the address found in RAMADD. As in the READ action, a value must be assigned to RAMADD before the WRITE occurs.

After a WRITE action using the printer, channel 12 of the carriage tape is checked. If the carriage tape is at channel 12, control is transferred automatically on a DO basis, to table number 999. If the program does not have a table 999, a standard routine will advance the carriage tape to channel 01, and the program continues. If the carriage tape is not at channel 12, the program continues.

A tape end-of-file condition exists when the EOF reflective spot on the tape has been sensed; if an end-of-file table has been specified control will revert to that table, otherwise a stop will be executed.

CLOSE -- This operator is used in conjunction with a single operand to control input and output magnetic tape units only. With an input tape unit, the CLOSE action causes the tape unit to rewind to the load point. With an output tape unit, the CLOSE action causes a tape mark to be written and then the tape unit to be rewound to the load point. The operand must always be a file name that represents the input or output tape unit desired, as defined in the data description table. The general form of the CLOSE action is shown.

CLOSE	INV			X	
CLOSE					INV

Note that the operand (file name) INV (inventory) can appear in either of two locations. The first example shows the limited entry form, and the second example the extended entry form.

Assignment Operators

MOVE... TO... -- This two-part operator is used in conjunction with two operands to move values from one field or record to another field or record. Several types of operands can be used, but only these combinations of Operand 1 and Operand 2 are allowed:

1. Operand 1 - record name
   Operand 2 - record name

2. Operand 1 - literal
   constant name
   expression name
   variable name
   RAMADD
   Operand 2 - variable name
   RAMADD

The general form of the MOVE... TO... action is shown.

STUB				ENTRY	
Operator	Operand	Operator	Operand	Operator	Operand
MOVE	1	TO	2	X	
MOVE	1	TO			2

Operand 1 is always the sending area, and Operand 2 the receiving area. When both operands are record names, data in Operand 1 is moved to the storage area represented by Operand 2 in the move mode. Thus, word marks in Operand 1 are not moved and any word marks in Operand 2 remain unchanged. The lengths of the records must either be the same, or the sending record shorter. When the receiving record is the longer record, remaining characters in the receiving record are unchanged, except that one extra character is moved (a characteristic of the 1401) thus destroying one position of the receiving record.

When Operand 1 and Operand 2 are other than record names, the shorter of the two operands, as specified in the data description table, delimits movement of data. When numeric data is being moved, and Operand 1 is the shorter field, low order decimal positions and high order integer positions in the receiving field are zero filled up to the number of decimal and integer positions specified for the receiving field. For example:

Before the MOVE... TO... action:

Operand 1	987.65
Operand 2	6 integer and 4 decimal positions specified

After the MOVE... TO... action:

Operand 1	987.65
Operand 2	000987.6500

When Operand 2 is the shorter field, low order decimal positions and high order integer positions of Operand 1 are truncated to satisfy the number of decimal and integer positions specified for Operand 2. For example:

Before the MOVE... TO... action:

Operand 1	98765.4321
Operand 2	4 integer and 2 decimal positions specified

After the MOVE... TO... action:

Operand 1	98765.4321
Operand 2	8765.43

When alphameric data is being moved and Operand 2 is the longer field, low order positions of Operand 2 are blank filled. For example:

Before the MOVE... TO... action:

Operand 1	ABCD
Operand 2	6 positions specified

After the MOVE... TO... action:

Operand 1	ABCD
Operand 2	ABCDbb

When Operand 2 is the shorter field, low order positions of Operand 1 are truncated. For example:

Before the MOVE... TO... action:

Operand 1	ABCDEF
Operand 2	4 positions specified

After the MOVE... TO... action:

Operand 1	ABCDEF
Operand 2	ABCD

Numeric data can be moved either to a numeric field or to an alphameric field. When moved to an alphameric field, the numeric field is treated as though it were an alphameric field. Alphameric data can be moved only to an alphameric field. When numeric data is moved, the sign of the sending field (Operand 1) is moved to the receiving field (Operand 2). When Operand 1 is an expression name, the value of the expression is computed, and then this value is moved to the receiving field.

When Operand 1 is a numeric literal, its decimal and integer length is established as written. The rules for zero filling and truncation of the receiving field are the same as for other types of operands in the sending field. Note that the total number of decimal and integer positions for a numeric literal cannot exceed 6. When Operand 1 is an alphameric literal, its length is established as written. The rules for blank filling and truncation of the receiving field are the same as for other types of operands in the sending field. Note that the number of positions for an alphameric literal cannot exceed 6.

The sending field (Operand 1) cannot be a masked variable. Furthermore, the receiving field (Operand 2) can be a masked variable only if the sending field is numeric.

The receiving field (or sending) can be the special operand RAMADD, which is required for READ and WRITE actions to determine the RAMAC address.

SET. . . EQ. . . -- This two-part operator is used in conjunction with two operands to set the value of one field or record equal to the value of another field or record. Several types of operands can be used, but only these combinations of Operand 1 and Operand 2 are allowed:

1. Operand 1 - record name
   Operand 2 - record name
2. Operand 1 - variable name
                RAMADD
   Operand 2 - literal
                constant name
                expression name
                variable name
                RAMADD

The general form of the SET. . . EQ. . . action is shown.

Operator	Operand	Operator	Operand	Operator	Operand
SET	1	EQ	2	X	
SET	1	EQ			2
SET	1			EQ	2

Operand 1 is always the receiving area, and Operand 2 the sending area. Otherwise, SET... EQ... is identical to MOVE... TO... for all types of operands specified.

MOVE... +...
MOVE... -... These two-part operators are used in conjunction with two operands to add the value of the first operand to the value of the second operand or to subtract the value of the first operand from the value of the second operand. Several types of operands can be used, but only these combinations of Operand 1 and Operand 2 are allowed:

Operand 1 -literal
constant name
expression name
variable name
RAMADD

Operand 2 - variable name
RAMADD

The general form of the MOVE ± action is shown.

MOVE	1	±	2	X	
MOVE	1	±			2
MOVE	1			±	2

The operands used with the MOVE ± action can be numeric values only. Algebraic rules are followed when considering the sign of the action operator in conjunction with the signs of the two operands.

The integer and decimal lengths of the receiving field (Operand 2) determine the number of integer and decimal positions in the total resulting from the addition of Operand 1 to Operand 2 or the subtraction of Operand 1 from Operand 2. Thus, the decimal positions in Operand 1 in excess of the number specified for Operand 2 are truncated. High order integer positions in Operand 1, beyond the number specified for Operand 2 are truncated. Any carry from the high order position of Operand 2 is also truncated. Consider these examples, in which the sign and number of integer and decimal positions shown for each operand is that specified in the data description table.

Operand	Sign of Operator	Before MOVE ± Action	After MOVE ± Action
1	+	+321.654	+321.654
2		+456.123	+777.777
1	+	+456.123	+456.123
2		-123.456	+332.667
1	-	-123.654	-123.654
2		+456.321	+579.975
1	+	+123.65	+123.65
2		+456.321	+579.971
1	+	+ 23.654	+ 23.654
2		+456.321	+479.975
1	+	+ 23.65	+ 23.65
2		+456.321	+479.971
1	+	+654.321	+654.321
2		+543.876	+198.197
1	+	+321.654	+321.654
2		+ 23.45	+ 45.10

SET...+...
SET...-...

These two-part operators are used in conjunction with two operands to add the value of the second operand to the value of the first operand or to subtract the value of the second operand from the value of the first operand. Several types of operands can be used, but only these combinations of Operand 1 and Operand 2 are allowed:

Operand 1 - variable name
RAMADD

Operand 2 - literal
constant name
expression name
variable name
RAMADD

The general form of the SET ± action is shown.

SET	1	±	2	X	
SET	1	±			2
SET	1			±	2

Except that Operand 1 is the receiving field and Operand 2 is the sending field, the SET ± action is identical to the MOVE ± action. That is, Operand 2 is added to Operand 1 or subtracted from Operand 1. Otherwise, all rules applying to the MOVE ± action apply to the SET ± action.

LAYOUT -- This operator is used in conjunction with a single operand, which must always be a record name, to provide word marks in the area reserved for the file to which this record belongs. The general form of the LAYOUT action is shown.

LAYOUT	Record name			X	
LAYOUT					Record name

The LAYOUT action is required only when there are multiple record types in a file, and when these records are read in move mode. Execution of this action causes the word marks required by a specific record type within a file, to be inserted in the core storage area reserved for that file. The word marks are inserted according to the lengths of the variable fields, as specified for the record in the data description table. Any word marks currently in the file area in core storage are removed. In a procedure table, the LAYOUT action should always precede any MOVE or SET action in which the receiving field does not contain the proper word marks. Thus, for example if YTDPAY (year to date pay) is a record type within the file named PAYREG (pay register), and GRPAY (gross pay) is to be added to YTDTOT (year to date total) to obtain the value of the new year to date total prior to printing PAYREG, the following actions should be written in the procedure table, if the file area is not already set up with the format of YTDPAY.

Operator	Operand	Operator	Operand	Operator	Operand
LAYOUT	YTDPAY			X	
MOVE	GRPAY	+	YTDTOT	X	
WRITE	PAYREG			X	

When only one type of record appears in a file, the programmer need not concern himself with word marks since they are automatically installed by the processor. It is only when the same core area must handle more than one format (different length fields, thus different word mark positioning) that the LAYOUT action must be used to insure that the proper word marks are in place. By analyzing the different formats described in the data description, the processor is able to generate the proper word marks. This action takes place when the LAYOUT record name command is executed.

Consider a detail file with two types of records (change and delete).

Change --	xx	xxxxxx	xxx	xxxxxxxx
	C1	C2	C3	C4

Delete --	xx	xxxx	xxxxxxxxxxxxx
	D1	D2	D3

C1 EQ	'CH'	'DE'
LAYOUT	CHANGE	DELETE
MOVE C2 TO PRNT 1	X	
SET C4 + C3	X	
MOVE C4 TO PRNT 2	X	
WRITE PRINT	X	
MOVE D3 TO PCH 1		X
READ CARD	X	X
GO TO	TAB022	TAB023

<u>Sequence Control Operators</u>

DO... This operator is used only in conjunction with the special operand TABnnn, where nnn is the number of a closed procedure table. The general form of the DO action is shown.

DO	TABnnn			X	
DO					TABnnn

The DO action calls for the execution of another table. For example, if in performing actions in one table it is desired to test the conditions and perform the action of another table, the DO TABnnn action is used. Control will then revert to the beginning of TABnnn, it will be executed, and control will then revert to the original table to the action following the DO.

A closed procedure table (a table executed by a DO) is one that may not permanently transfer control to another table, and therefore it cannot contain a GO TO action or a NEXT TABLE number in the table header. Instead, the NEXT TABLE entry contains the word DO, which causes control to automatically return to the calling table in the program when it has been completed. It acts like a closed subroutine in normal programming.

Any procedure table (closed or open) can contain any number of DO's. Closed tables can be nested up to a level of 10. That is, each closed procedure table can in turn DO another closed procedure table up to a maximum of 10 such referenced tables. A table can DO itself, or any table previously done in a nest of DO's, but each such DO counts as one level of the nest.

While it is possible that none of the sets of conditions of a closed procedure table will be satisfied, an ERROR TABLE number must not appear in the table header, since this would break the linkage to the calling table in the program. If no conditions are satisfied, an error message is printed and control reverts to the calling table.

GO TO... This operator is used only in conjunction with the special operands TABnnn and STOP, where nnn is the number of an open procedure table. The general form of the GO TO action is shown.

GO TO	TABnnn			X	
GO TO					TABnnn
GO TO	TABSTOP			X	
GO TO					TABSTOP

An open procedure table is one that can permanently transfer control to another table, and therefore cannot be referenced by a DO action in a previous table. Permanent transfer of control is achieved by a GO TO action in the rule or a NEXT TABLE number, or an ERROR TABLE number in the table header.

When a rule contains a GO TO action, it must be the last action in the rule. When the operand TABSTOP is used with GO TO, the program stops. STOP can be used alone in the NEXT TABLE and ERROR TABLE entries.

Any procedure table (open or closed) can contain any number of DO's and these can be nested up to a level of 10 (see DO...).

Special Operator

SPACE... This operator is used only in conjunction with the special operands nLINE, nLINES, and nnCHAN to control spacing and skipping of the 1403 Printer carriage. The general form of the SPACE action is shown.

SPACE	nLINE			X	
SPACE	nLINES			X	
SPACE					nLINE
SPACE					nLINES
SPACE	nnCHAN			X	
SPACE					nnCHAN

The SPACE actions with the operands nLINE or nLINES cause the printer to line space n lines, where n can be 1, 2, or 3. The SPACE action with the operand nnCHAN causes the printer to skip to channel nn where nn can be 01 through 12 (see WRITE... for special actions after sensing channel 12).

The SPACE action is independent of the WRITE action and can be used at any point and any number of times in a procedure table.

Program Assembly and Operation

The assembly and operation of a 1401 Tabular Programming System is described with reference to the 1401 TP system diagram.

STEP 1

Step 1 consists of writing the program, both data description table and procedure tables, keypunching the cards, listing the cards on a 407, and visual checking for accuracy.

Important items to check are correct column alignment, correct spelling, correct usages of zero (0) and the letter O (Ø), correct usage of 1 and I, and other normal keypunching errors. Special points to check for in the data description are: (1) There must be one and only one header card for Table 000, which comes from the first sheet. For the procedure tables, there must be at least one header card for each table. Also, there can be an "order or rules card" and one or more comments cards. All these cards must appear before the first line of the procedure table proper. If a procedure table must be continued on a second page, the header on the second page cannot be used. All cards, both data description and procedure, must be in ascending sequence (based on columns 1-8). Expressions must follow description of the operands it references.

STEP 2

The TP processor deck is loaded into the 1401-1405. When the card reader stops, press the START key on the card reader to read the last two cards. The processor is now loaded.

STEP 3

Place the primer program (about six or seven cards) in the card reader, followed by the source language program prepared in Step 1. Press the LOAD button on the card reader. When the card reader stops, press the START key to read the last two cards. An END card is not required at the end of the source deck. During running of the processor, an Autocoder language program is punched and an index of the object program is printed by the 1403.

The deck of cards contains a control card and a job card. Mark the top of the deck so that proper sequence is maintained.

STEP 4

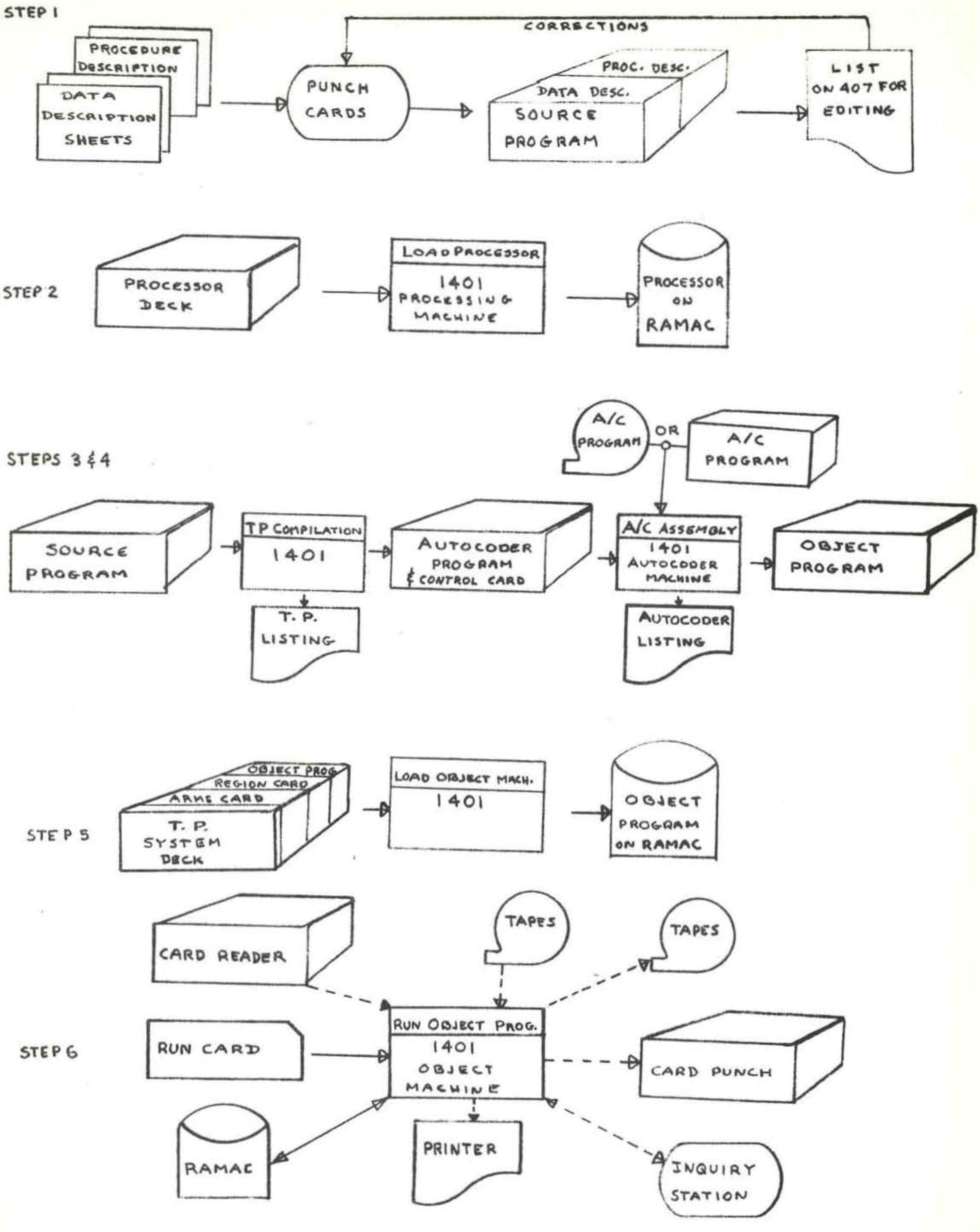
An Autocoder assembly is next performed, using the cards punched in Step 3. The control card furnished is 661 to produce a system to run on a 16000 character 1401. This machine is available at Endicott; if the machine to run the Autocoder assembly is smaller, you should make a substitution of the control card. Mount the Autocoder system tape on Tape Unit 1, and work tapes on units 4, 5, and 6. Place the card deck in the card reader and use the tape load procedure. A listing of the card deck is produced by the Autocoder processor. Note should be made of the error messages during the Autocoder assembly. In general, the program produced by the Autocoder will be in the same order as the program written in TP source language. An examination of the listing will show the source of duplicate names, undefined names, and whether the object program is too large for storage. Two exceptions to the general rule are that constants and expressions appear in each decision table where they are used rather than remaining with the data description where they were written.

STEP 5

The object program is loaded into the 1401. The procedure is similar to Step 2 except that this time the program consists of the object control system deck followed by the object program. These decks are put in the card reader, card load is performed, and again at the end of the cards the START key is pressed to feed the last two cards.

STEP 6

The object program can now be run. The object program primer deck of about 5 or 6 cards is placed in the card reader, and card load is executed. The object program begins with execution of table 001 and operates in accordance with the instructions written in the source language program.



1401 T.P. SYSTEM DIAGRAM

ANSWERS

Problem 1

1. Quotes missing around "AC" and "DC".
2. The specification on DC meters said "less than 50 volts." The relational operator is incorrectly written as LE, which means "less or equal." Change to LR. Similarly, change GR to GE.
3. The variable name NOTURNS has seven characters; only six are permitted.
4. The action line for wire size is an illegal combination of limited and extended entry. Place the 24 in Rules 1 and 2.
5. The WRITE CARD line will do nothing; place X's in the rules.

Problem 2

Line 001. Word CARD should be deleted.

Line 002. The number of characters in the four variables does not add to 80 as stated here.

Line 005. The decimal length of an alphabetic variable must be zero.

Line 007-008. The file description says the longest record has ten characters, but the record description says it has twelve. The name WORK is used twice, which is illegal.

Line 011. A printer file must be of length 100 or 132.

Line 012. This record description says there are five variables in the record, but only four variable descriptions are given.

Line 014. The integer length should be ten, the total number of characters in the masked field, rather than seven, the number of digits in the variable.

Problem 3

The main trouble is that the conditions for Rules 1 and 2 are both satisfied by a male with code 1 who is over 35; this violates a basic assumption of the system. Other errors: there should be no EQ on the set when used as an arithmetic operator; RATE cannot be used both as a variable name and a file name--or else the writer of the table forgot that the operand on a WRITE must be a file name.

Problem 4

Shown on table.

Problem 5

No answer.

						COL 01		COL 02		COL 03		COL 04		COL 05		COL 06		COL 07		COL 08		COL 09	
4 6	7 8	9 14	15 22	23 24	27 34	35 36	37 44	45 46	47 54	55 56	57 64	65 66	67 74	9 10	11 18	19 20	21 28	29 30	31 38	39 40	41 48	49 50	51 58
LINE	COL	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME	OP	NAME
01	00		SEX	EQ			'MALE'		'MALE'		'FEMALE'		'FEMALE'		'FEMALE'		'FEMALE'		'FEMALE'		'FEMALE'		ELSE
02	00		DEPT	EQ			47		47		48		48		49		49		50		50		
03	00		HØURS	LE	40	Y		Y		Y		Y		Y		Y		Y		Y			
04	00		DEDCØD				'B'		'D'		'C'				'C'				'C'				
05	00		RATE									GR	250			GR	250			GR	250		
06	00	GØ TØ					TAB010		TAB010		TAB020		TAB020		TAB020		TAB020		TAB020		TAB020		TAB030
07	00																						
08	00																						
09	00																						
10	00																						
11	00																						
12	00																						
13	00																						
14	00																						
15	00																						
16	00																						
17	00																						
18	00																						
19	00																						
20	00																						

Answer Problem 4