P A C T I M A N U A L

PROJECT FOR THE ADVANCEMENT

OF CODING TECHNIQUES

AUGUST 1955

PACT MANUAL

Date ?

ERRATA SHEET 2

Page	Line	Correction
04·31·70	23 through 26	Delete last 2 sentences in paragraph 2.
04·31·74	7 etc.	Delete entire paragraph 7.

FORM 16-G-1 REV. 2-47

TABLE OF CONTENTS

INTRODUCTION

GENERAL REMARKS

1. The first meeting of the Policy Committee of PACT, Project for the Advancement of Coding Techniques, was held on November 15, 1954. At this time it was decided to proceed with plans for a compiling routine. The initial effort is a modest one with the idea of putting the system to practical use without much delay. Experience gained in this initial phase may be used in more elaborate extensions at a later date.

2. A working committee composed of at least one representative from each cooperating organization has held meetings regularly to discuss the details of such a routine, break it into logical units, each to be written by one of the representatives, and integrate these parts into a routine, called Pact I. Appendix 1 contains the list of the members of this working committee.

PHILOSOPHY

1. Pact I is a compiling routine which acts as an executive routine before the desired computation is started. It translates a program written in a pseudocode, incorporates the required library programs and assembles the complete program. The pseudocode is written in the single-operation, single-operand, single-result form; the operations are either symbols or mnemonic letters and the operands are expressed by alphabetic or numeric symbols, not the addresses of their storage locations. All arithmetic is done in fixed point.

2. It is hoped that Pact I will make coding easier and more accurate without much loss in efficiency of the final machine code. The operations and rules of Pact I are simpler and more nearly algebraic than most machine language codes. There are also some pseudo-operations which accomplish standardized manipulations, such as automatic coding of loops and handling arrays of variables.

3. The programming concept underlying the Pact I routine is the master routine and subroutine. Problems written in this system are translated by Pact I, a compiling routine, into machine language instructions. These instructions are later loaded into the machine and they are executed at high speed. This is in contrast to the interpretive type of operation in which each instruction is translated each time it is to be executed.

4. Although Pact I has been written for use on the IBM 701, the basic principles of the system apply to any machine of a similar type.

PROCEDURE FOR USING PACT I

1. The first stage in using Pact I is writing the steps of the pseudocode. These steps indicate the mathematical operations by means of the Pact I operations and the characters (letters and/or numbers) representing the operand, which may be a variable quantity, a constant quantity, or a reference to another region or step.

2. The Pact I routine converts these steps into machine language instructions. As a result of the Pact I routine the following items are accomplished:

A. Listing of each step and its corresponding relative instructions.

B. List of the variables and their relative locations.

C. Deck of relative binary cards.

3. A loading routine, also a result of the Pact I routine, assigns the absolute (actual) location of the variables, temporary storage locations, numbers, library programs and instructions. The program is then ready to execute in absolute machine language. These library programs include the subroutines in Pact I and also any subroutines that the programmer himself may write.

4. An explanation of programming, including variables, relative instructions relative binary cards, library programs and associated information is given in the succeeding sections.

PROGRAMMING

INTRODUCTION

1. After a problem has been carefully thought out with reference to the physical phenomena and numerical methods, programming can commence. The problem should be broken into logical blocks which make up the regions. Each region is composed of sequences of steps which are similar to the mathematical operations used to evaluate each expression.

2. A few simple examples illustrate some of the basic ideas needed to write the Pact I steps of a program. If it is desired to add two quantities, C and D, and to store the result as K for future reference, the Pact I steps would be:

OPERATION	OPERAND
	C
+	D
EQ	K

This can be read: "Take C, add D and set the sum equal to K." Each step in a Pact I program is written on one line of a code sheet and contains an operation defined by the Pact I routine and an operand. The operand may take on any one of several forms. In the above example, the quantities were indicated only by letters; such quantities are called variables. A variable may be denoted by any alphanumeric character, but each variable must have at least one alphabetic character in its so-called "name".

3. It is also possible to use a number as the operand. If the programmer wishes to calculate $K = \frac{x - 2y}{5}$, the steps can be written in the following way:

	Y
X	− 2
+	X
/	5
EQ	K

This can be read: "Take y, multiply by -2 add x, divide by 5 and set the result equal to K. It is possible to use numbers that are not integers and that have as many as ten digits. Such numbers are written in a special field on the code sheet called Numbers.

4. By means of the Pact I routine, it is easy to refer to the results of previous steps. Suppose the programmer wishes to find the value of N in the relationship:

$$N = \frac{5x - y}{z} + \frac{3x}{y}$$

The Pact I steps indicated below show how this can be coded.

Step	Operation	Operand	Remarks
1.0		X	Take X
2.0	x	5	Multiply by 5
3.0	-	Y	Subtract Y
4.0	/	Z	First term
5.0		X	Take X
6.0	x	3	Multiply by 3
7.0	/	Y	Second term
8.0	+	R 4	Plus first term
9.0	EQ	N	N

Notice that it is sufficient to add the first term to the second term by specifying in step 8: " + R 4.0". This is read: add the result of step 4. This is one of the conveniences of Pact I.

5. In addition, the operand may be a symbol representing an array of numbers; this can be either a one dimensional or two dimensional array, sometimes called a vector or a matrix, respectively. The programmer need only write the variables or numbers themselves; the Pact I routine assigns locations for them. Whenever it is necessary to store intermediate results and final results, this is done by the Pact I routine without any effort on the part of the programmer. All these locations assigned by the Pact I routine are full-words, thirty-five bits and sign.

6. All the arithmetic is done in fixed point arithmetic using full-word operands and giving full-word results. No rounding is accomplished by the arithmetic operations of the Pact I routine. Although the programmer must be aware of the scaling, or magnitude, of the operand, the Pact I routine has definite ways of handling the scaling of the operand that aid the programmer. However, he must make provision for the possibility of overflow; the Pact I routine does not handle overflow automatically. The section on scaling following later discusses this concept and any necessary restrictions.

7. Programming is greatly simplified by use of the Pact I routine. Loops can be generated by means of a pair of steps; the required set up of initial addresses and the test to repeat or leave the loop are automatically included. Regions, or whole blocks of steps, can be duplicated and subroutines can be incorporated in the program by merely specifying this in a Pact I step. These Pact I steps written by the programmer constitute the psuedocode. These steps are translated by the Pact I routine into machine language instructions; there may be many machine language instructions corresponding to each step of the pseudo-code.

8. In the succeeding sections information about the Pact I routine and details for using it are given. A novice in the computing field could write a straight forward program with a minimum of information. However, an experienced programmer with a good knowledge of the machine and the details of the Pact I routine itself will be able to write a more complex program and he will, no doubt, take much better advantage of the possibilities of Pact I.

FORM OF STEPS

1. The pseudocode for the problem is composed of a series of regions which contain sequences of steps or instructions. These individual steps are translated by the Pact I routine into relative machine language instructions. Each step is written on one line of the code sheet and contains the three following pieces of information:

A. Region and step to specify the order of execution of the step

B. Operation to be performed

C. Quantity to be operated upon as indicated by the factor, subscript, scaling, and/or number.

The layout of this format is given below. On the code sheet there is also space for notes; this space is omitted here.

Region	Step	Operation	Factor	S_1	S_2	Q	Number
9 - 11	12 - 15	16 - 19	20 - 24	25 - 27	28 - 30	31 - 32	33 - 44

Each of these fields is explained in the following paragraphs. There is a very short problem coded in the Pact I form at the end of this section.

2. REGION AND STEP (Card columns 9 through 15). The region and step provide the major and minor ordering of the steps of a program. A region may be identified by not more than three characters which can be handled by the available equipment. The step is denoted by four numeric characters; the least significant of these four digits is normally reserved for an insertion number. It is easy to make insertions in the pseudocode by means of the fourth digit in the step field. This insertion number is generally zero when the psuedocode is first written; it is possible to make as many as nine insertions between two steps having a zero insertion number. It is also possible to make deletions by simply omitting a step. The step numbers must be in increasing order as they are to be read by the Pact I routine.

3. OPERATION (Card columns 16 through 19). Available operations, their functions, and their symbols are listed below. Each operation is represented by not more than four characters. It produces one single precision result which is available for use in the next step.

Operation	Symbol	Description
Take	(No symbol)	Take the factor as the first operand of a sequence of operations.
Add	+	Add the factor to the result of the previous step.
Subtract	-	Subtract the factor from the result of the previous step.
Multiply	x	Multiply the result of previous step by the factor.
Divide	/	Divide the result of the previous step by the factor and get the quotient.
Residue	RES	Find the residue from the previous step. For division the residue is the remainder. For multiplication the residue is the remaining portion of the product after the most significant digits have been removed.
Equals	EQ	Give the name in the factor field to the result of the previous step and save this result for future use.
Absolute	ABS	Take the absolute value of the factor as the first operand of a sequence of operations.
Add Absolute	+ABS	Add the absolute value of the factor to the result of the previous step.
Subtract Absolute	-ABS	Subtract the absolute value of the factor from the result of the previous step.
Transfer (1)	T	Transfer to the step noted as the factor.
Transfer on Zero (1)	TZ	Transfer to the step noted as the factor if the result of the previous step is zero. If the result is not zero, proceed with the program.

(1) The step noted in the factor field of these instructions must be within the same region and it must be the first step in a sequence of operations which is normally a Take ( ) operation (See Table 2B). The result of the previous step is retained except for executed transfer.

Operation	Symbol	Description
Transfer on Positive (1)	TP	Transfer to the step noted as the factor if the result of the previous step is greater than zero, a positive number, not positive zero. If the result is not greater then zero, proceed with program.
Transfer on Negative (1)	TN	Transfer to the step noted as the factor if the result of the previous step is less than zero, a negative number, not negative zero. Proceed with the program if the result is zero or greater than zero.
Transfer on Overflow (1)	TF	Transfer to the step noted as the factor if the overflow indicator is on.
Halt(1)	HALT	Halt and transfer to the step noted as the factor if the start signal is given by the operator.
Sense	SE	Sense controls are defined by the S_1 field according to the machine for which the compiler is coded. This is translated into one machine language instruction; if the programmer wishes to transfer control, he must include the transfer operation in the Pact I program in the next step.
Clear	CL	Set the factor to zero. (Only one element of an array is set to zero unless the program provides for this).
Set (2)	SET	Set the subscript noted in the S_1 field to the value noted or represented in the S_2 field.
Test (2)	TEST	Increase the subscript noted in S_1 by one. If this results in a value which is greater than the value noted or represented in S_2, control goes to the next step. Otherwise, it is transferred to the step noted as the factor; if the factor is blank, then control is transferred to the step following the corresponding Set.

(2) SET and TEST operations have the following restrictions. Every SET must be followed by a TEST within the same region. The subscript noted in S_1 refers only to this subscript as used after the SET and before the corresponding TEST within this region. The location of an element within an array by means of subscripts is done by the compiler by computing a function of those subscripts. The subscripts themselves are not available to the compiled program. Therefore, no operations other than SET, TEST, or USE are permitted on subscripts. Further explanation is given below in the section on Loop Writing.

FORM 16-G-1 REV. 2-47

Operation	Symbol	Description
Use (3)	USE	Set the subscript noted in the S_1 field to the value represented in the S_2 field.
Sine	SIN	Compute the sine of the factor. The angle must be in radians.
Cosine	COS	Compute the cosine of the factor. The angle must be in radians.
Arctangent	ARCT	Compute the arctangent of the factor. The result is in radians.
Square root	SQRT	Compute the square root of the factor.
Logarithm	LOG	Compute the natural logarithm of the factor.
Exponential	EXP	Compute e^x, where x is the number represented by the factor.
Duplicate	DUP	Insert at this point the region noted in the factor field. The step containing the DUP operation will not appear in the listing of the program produced by the Pact I routine. The entire region being duplicated takes the place of the one step.
Exit	EXIT	Exit from a region. The factor field has no function. If a region is executed because of a "Do Region and Return" (DO) operation, then the EXIT instruction in this region gives the return signal. An EXIT instruction is automatically placed at the end of every region except a duplicated (DUP) region.

(3) Both operations, SET and USE, set the subscript noted in the S_1 field to the value given in the S_2 field. The distinction between the two is the fact that SET is always used in conjunction with the TEST operation; USE never has a corresponding TEST operation.

Operation	Symbol	Description
Library Program	LIB (4)	Do the library program noted as the factor and return to the first step that does not contain information used by the library program.
Do Region and Return	DO (4)	Do the steps in the region noted as the factor and return to a step following the DO operation. If information is to be transmitted to the region noted, this information is written in the steps immediately following the DO operation. At the conclusion of the region noted control is returned to the first step that does not contain information associated with the DO operation.
Identification	ID (4)	The factor, S_1, S_2 and/or Q fields contain information needed by a library program or subroutine. The operation, ID, initiates the development of instructions which set the exit and the proper addresses throughout the library program or subroutine.
For	FOR (4)	The factor, S_1 and S_2 fields contain information needed by a library program or subprogram.
Call	CALL (4)	As a result of this operation, CALL, the relative machine language instructions are written in the subroutine so that information given in the psuedocode is available for use in the subroutine.
Read	READ	Read the decimal data cards until a 12 punch in card column 80 is sensed.
List	LIST	Print the quantities specified in the immediately following steps which have the ID operation.

(4) These operations are discussed in the section, Subroutines and Library Programs.

FORM 18-G-1 REV. 2-47

SAMPLE CODE SHEET

PACT I CODE

JOB NO. 1639-03 DECK NO. 610-604 PROGRAMMER JAY MALLET DATE: 7-11-55 PAGE 1 OF 1

REGION	STEP	OP.	FACTOR	S_1	S_2	± BIN. MAGN. Q	±	NUMBER	NOTES
9 11	12 15	16 19	20 24	25 27	28 30	31 32	33	34 44	45 79
12	10	DO	13						DO REGION 13 & RETURN TO NEXT STEP
12	20	HALT	10						
13	10	READ							READ DATA
	20		X			8			EVALUATE POLYNOMIAL TAKE X
	30	+	5			8			ADD 5
	40	X	X			8			MULTIPLY BY X
	50	-	3			8			SUBTRACT 3
	60	EQ	Y			8			EQUALS Y
	70	+	A			8			Y PLUS A
	80	-					+	67.56	MINUS 67.56
	90	EQ	Z			8			EQUALS Z
	100	LIST							PRINT Y AND Z
	110	ID	Y						
13	120	ID	Z						
									$X^2+5X-3=Y$
									$Y+A-67.56=Z$

4. OPERAND (Card Columns 20 through 24) The operand can represent any one of the following items: (1) Numbers, (2) names of variables (hereafter called variables), (3) results of a step, (4) another step in the same region, or (5) region. The operation and the way in which the operand is written on the code sheet determine each of these items. The fields pertaining to the operand are factor field, S_1 and S_2 fields, scaling or Q field, and Numbers field. Each of these is explained directly below.

A. Factor (Card Columns 20 through 24). The factor field is composed of 5 columns; the first column called the clue column contains blank, R, N or -; the next three card columns contain either numeric or alphabetic characters; and the last column is reserved for an insertion digit. Consequently, it is used when the factor field refers to a step.

(1) A Number is written as a three digit integer, the units position in card column 23. If a negative integer is desired, place a "-" in the clue column. The insertion column cannot be used for a Number, and the number zero cannot be used, as a zero is treated as a blank.

(2) The name of a variable is written in card columns 21, 22 and 23, having at least one alphabetic character. A "-" in the clue column calls for the negative of the variable. The insertion column is not used in connection with variables.

(3) Result of a previous step in the same region is indicated by an "R" in the clue column and the step number in the remaining columns of the factor field. If the negative of the result is desired, an "N" is written in the clue column. If card columns 21, 22, and 23 are blank, the result of the immediately preceding step is used.

(4) When transfer of control to another step within the same region is indicated, the step is written in columns 21 through 24. The clue column is blank for this situation. The step to which reference is made must be the beginning of a sequence of steps. (Refer to Table 2B)

(5) The region, designated by three characters in columns 21, 22, or 23 of the factor field, is used with certain operations such as Do Region and Return (DO) or Library Program (LIB).

If a quantity is written in both the factor field and the number field, only the quantity referred to in the number field is used. If both the factor field and number field are blank, the result of the previous step is used. When the factor field is blank but the number field is filled, then the number field is used.

B. Subscripts (Card Columns 25 through 27 for S_1 and Card Columns 28 through 30 for S_2). Subscripts are used with the factor to identify variables. When subscript fields are used with certain operations, SET, TEST, USE, FOR, and ID, they have the special meanings explained

in the description of these operations. Further explanation is given in the sections on Subroutines and Library Programs and Loop writing.

C. Scaling (Card Columns 31 through 32). The scaling desired on the result of each step is indicated in the Q field. The Q value is the number of binary places before the binary point in the result of the step. The sign of Q is placed above the digit in card column 31. If no Q is given, the binary point will be determined in the conventional manner described in the next section on Operations.

D. Number (Card Columns 33 through 44). The number field provides a means of entering numbers which will not fit in the factor field: ten decimal digits, a decimal point which occupies one card column and a sign which also occupies one card column, namely 33. If the programmer writes an eleven digit number without a decimal point, it is converted as an integer. When the factor field is blank, the quantity in the number field is used as the factor.

5. GENERAL REMARKS ABOUT THE CODE SHEET. The key-punch operator will probably find this code sheet more difficult to key-punch because any alphabetic and numeric characters or other available symbols can be used in the various fields. Hence, when the programmer competes the code sheet (and also the variable definition sheet) he should exercise great care in writing the characters in the appropriate card columns. Within some fields the programmer may select the particular card columns he wishes to use; however, once a quantity is written in a particular way, it must be written in exactly the same way throughout the program. A blank in a card column is just as significant as a character and can be considered a character. The symbol for the operation must start at the left of the operation field, card column 16. The programmer should attempt to write each symbol so clearly that the key-punch operator cannot possibly be confused; for example, try to distinguish between a capital I and the number 1. The notes can be key-punched so that they can be listed on a tabulator for checking purposes before the program is compiled by the Pact I routine. It is generally wise procedure to check this so called pre-listing of the program. Also, any numbers placed in any field except the number field must be placed to the extreme right of the field as a blank in this case is treated as a zero. For example: "one blank blank" is interpreted as 100.

Write right
Fill each
Operation

OPERATIONS

1. In order to produce the machine language program the Pact I steps are expanded into instructions that call for appropriate arithmetic and logical operations and proper addresses to be used during the execution of the programs. The Pact I compiling routine performs this expansion for the programmer. The following discussion will indicate what the programmer can expect Pact I to do with regard to operation expansion.

2. The Pact I operations which are available to the programmer are listed above in the section, Form of Instructions. With these tools the programmer will be able to code the evaluation of an equation in much the same manner that he would write the equation in mathematical symbols. The Pact I routine translates these operation symbols and produces the machine language program so that the machine can actually add, multiply or effect whatever operation is required. Usually several machine language instructions are produced for each Pact step. The machine language program operates in fixed point arithmetic, using full-word operands and giving full-word results. The operand, as indicated above in the preceding section, can be a variable, a Number, a result computed earlier in the program, or the result of the previous step.

3. The scaling, or magnitude, Q, of a result can be supplied by the programmer or will be computed by Pact I. In the first case, this means that the compiling routine incorporates the correct shift instructions to obtain the result at the desired Q. In the second instance the compiling routine calculates the Q of the result by the rules given below in paragraph 5.

4. As an illustration of how the Pact I operations are handled, assume that an Add (+) operation is written in a step. First, the Pact I routine sets up a shift operation, if necessary, and the Add (+) in machine language is produced. Secondly, if the programmer has specified a Q for the sum, the required shift operation is also produced. If no Q was specified for the sum, a Q is computed by the Pact I routine which involves no shifting operations in producing the indicated result. This is discussed more explicitly in the following paragraph. Then, the result is set up in such a manner that it will be in the proper position, that is, put in the proper register, for a similar procedure to take place in the next Pact I step. Some cases are slightly more complicated.

5. When the binary magnitude, Q, is expressed in a Pact I step on the code sheet, the result of this step will be expressed at this specified Q; the Pact I routine supplies the proper shift instructions to produce this Q. If no Q is designated, Pact I computes the Q by the following rules:

Operation	Rule
Add (A+ B)	Q_R is the larger of Q_A and Q_B.
Subtract (A - B)	Q_R is the larger of Q_A and Q_B.

Operation	Rule
Multiply (AB)	$Q_R = Q_A + Q_B$
Divide (A/B)	$Q_R = Q_A - Q_B$
Residue after multiplication	$Q_R = Q_A + Q_B - 35$
Residue after division	$Q_R = Q_A - 35$

This means, for example that the addition of a quantity to a computed result can be indicated without the coder knowing the Q of the computed result. Pact I produces the proper coding so that the addition takes place at the larger Q. However, the programmer should be constantly aware of this scaling; it is discussed in the section entitled, Scaling. The following explanation of some of the operations indicates how scaling influences the expansion of the Pact I operation into the machine language instructions.

6. It is desirable that the programmer know the sequence of events and also requirements or restrictions for the Pact I operations. A short discussion of these operations follows immediately.

A. Add (+) or Subtract (-). Addition type operations (addition or subtraction) always occur at the larger Q of the two terms. This means that either one of the factors may have to be shifted; this coding is automatically produced by the Pact routine. If the first term has the smaller Q, it will be shifted to the Q of the second term. If the second term has the smaller Q, the first term will be temporarily stored while the second is appropriately shifted; then, the first is recalled and added. A further shift may follow if the Q of the result has been specified.

B. Multiply (x). Multiplication takes place at the Q of each factor. If the programmer has not specified a Q for the product, this product will be left as a full-word without shifting. However, if a Q is specified for the product, the machine language program will contain the proper shift operation before the next.

C. Divide (/). Prior to division, the dividend is shifted so that the quotient is produced at the Q specified for the quotient by the programmer. If no Q is specified for the quotient, the dividend is not shifted before the division takes place. In either situation the contents of the MQ will be set to zero before shifting or division. Quotient overflow can occur and will stop the machine. It is the responsibility of the programmer to assign Q in such a way that it does not occur.

D. Take ( ). The operation, Take, will place the operand at the Q indicated in the correct register to be used in the following Pact I step.

E. Residue (RES). The term, residue, has two interpretations depending upon the operation of the previous step. For multiplication the residue is the remaining portion after the most significant digits have been removed. For division the residue is the remainder. The operation, Residue, causes the residue to be stored in temporary storage. Then, this operation behaves exactly like the operation Take, ( ). The operation, Residue, must follow immediately after a Multiply (x) or Divide (/) operation. No Pact I step can be written between a Multiply or Divide operation and a Residue operation. However, if the results of the Multiply or Divide step are referred to elsewhere in the program, the Residue operation is not invalidated. It should be noted when a division by the negative of a variable is accomplished, the residue is the negative of the remainder.

F. Equals (EQ). If Q, is specified in a step containing the Equals operation, and this Q is consistent throughout the program for this variable, the shifting is done to store the result at this Q. An error stop occurs if the Q of the same variable given in different steps is inconsistent. An Equals operation should not occur in the same step as a number, either in the factor field or the number field.

G. Halt (HALT). When a Q is specified in a step having Halt as the operation part, the shift instructions are produced prior to the halt (H) instruction. If Q is not specified, no shifting is produced unless the result of the previous step is the result of a Divide operation. In this case, the result is placed in the accumulator so that the machine logic can have access to the result. A new sequence of steps must begin after a Halt operation; also the step to which control is transferred must be the first step of a new sequence. A list of permissible operations for the first step in a sequence is given in Reference Table 2B. The factor field must never be left blank in a step whose operation is HALT.

H. Transfer (T). A new sequence of steps must begin after the T Transfer operation. In addition, this operation must transfer to the beginning of a new sequence of steps. Also, the Q has no significance on this operation.

I. Conditional Transfer Operations (TZ, TP, TN, TF). If a Q is specified in a step having any one of the conditional transfers as the operation, the shifting occurs first, then the test for the condition is made. If the condition is not satisfied, the result of the previous step after the shifting is retained. However, if the condition is satisfied, the transfer is executed. Control must be transferred to a new sequence; see Table 2B.

J. Duplicate (DUP). The Duplicate operation causes the entire region referred to in the factor field of the step to replace this step in the final program. The original step containing the Duplicate operation will not appear in the final program. Also, the region being duplicated will only appear in the final program where the original Duplicate step occurred; the duplicated region will not appear as a distinct region of its own. A transfer to a step having the Duplicate operation will transfer to the first step of the inserted region. The step containing the Duplicate operation can be written within a loop. The results or the negative of results of previous steps will not overlap during a Duplicate operation. Hence, they will remain intact throughout a distinct, final, compiled region.

7. It is frequently desirable to perform an operation on the result of the previous step. When both the factor field and number field are blank, the result of the previous step is indicated as the operand. However, this interpretation is true only for the following operations:

Add (+)	Sine (SIN)
Add Absolute (+ABS)	Cosine (COS)
Subtract (-)	Square root (SQRT)
Multiply (x)	Logarithm (LOG)
Divide (/)	Exponential (EXP)
Residue (RES)	Arctangent (ARCT)
Subtract Absolute (-ABS)	

8. There are restrictions on the use of some operations in some particular respects. The operations listed below do not recognize the Q. Also, the programmer may not refer to the results of a step having an operation that is listed below.

	Do Region and Return (DO)	Exit (EXIT)
	Library Program (LIB)	Duplicate (DUP)
(1)	Identification (ID)	Use (USE)
	For (FOR)	Clear (CL)
	Sense (SE)	Read
	Set (SET)	Transfer
	Test (TEST)	Call (CALL)
	List	

(1) Q becomes a parameter in a calling sequence, except after LIST.

VARIABLES

1. The variables for a program include the variable input data and variable data defined by an Equals (EQ) operation with a variable as a factor during the course of the program. Variable data are distinguished by the method of reference and by the condition that they may change in different cases of a program while the Numbers remain unchanged whenever the program is used. The Numbers are written directly on the code sheet; they are discussed below in the section entitled Numbers. Every variable must be defined either on the code sheet or the variable definition sheet (see paragraph 7 below). Variable cards are key-punched from the variable definition sheet; these cards are read into the machine at the time of compiling. Pact I provides for the scaling of variables as specified on the variable definition sheet or code sheet and also for the required number of storage locations. Pact I assigns locations to variables given on the variable definition sheet and the additional variables specified on the code sheet by a Pact I step having an Equals (EQ) operation. Finally, a list of the variables is printed showing each variable, its relative location and other pertinent information.

2. A variable can be a scalar, or a 1 or 2 dimensional array of data; a one-dimensional array being called a vector and a two dimensional array being called a matrix. Every such array is uniquely specified by the associated Factor, S_1 and S_2 fields on the code sheet or variable definition sheet. The factor field for a variable may contain as many as three symbols of which at least one must be an alphabetic character. The subscript fields, S_1 and S_2, can each contain any combination of three alphanumeric characters or be left blank. The sum of the number of variables and the number of Numbers must not exceed 512; an array is counted as one variable. Also, the maximum number of variables is 341.

3. SUBSCRIPTS. There are two types of subscripts: Inductive and Definitive. Inductive subscripts are used to specify a particular cell, or range of cells in the case of loops when the variable is a vector or a matrix. This type of subscript is discussed below with regard to vectors and matrices. A definitive subscript extends the name of the variable beyond the factor field into the S_1 field or the S_1 and S_2 fields. When definitive subscripts are used, the S_1 field is always used first. If two distinct variables have the same name in the factor field, then both variables must have the same number of definitive subscripts. The following situations are permissible when two variables have the same factor field:

A. The variable have different S_1 fields and no S_2 fields.

FACTOR	S_1	S_2
AH	1	
AH	2	

B. The variables each have two definitive subscripts and the S_1 fields are the same for both but the S_2 fields differ.

FACTOR	S_1	S_2
AK	3	B
AK	3	D

C. The variables each have two definitive subscripts and both the S_1 fields and S_2 fields differ for each variable.

CL	A	7
CL	B	8

4. SCALAR. A scalar occupies exactly one full-word storage location. A scalar can never have an inductive subscript, but can have 1 or 2 definitive subscripts.

5. VECTOR. A vector is a one-dimensional array; an element of a vector can be uniquely defined by a factor and one subscript. This subscript can be either S_1 or S_2 if there is no definitive subscript. If there is a definitive subscript, it (the definitive subscript) must be S_1. There must be at least two possible full-word storage locations which can be specified by the inductive subscript. Note that a scalar is not also a vector. An element of a vector having no definitive subscript can be written in either of the following ways:

KM	A	
KM		B

An element of a vector having a definitive subscript, 3, must be written:

KM	3	A

6. MATRIX. A matrix is a two-dimensional array; it is uniquely defined by a factor and two inductive subscripts, S_1 and S_2. There must be at least two possible full-word storage locations which can be specified by each of the inductive subscripts. A vector is not also a matrix. An element of a matrix can be written as:

ABC	I	J

7. VARIABLE DEFINITION SHEET AND VARIABLE CARD. Information is punched in one variable card just as it is written on one line of the variable definition sheet, which is illustrated on page 04·31·28. Rules and information for completing the variable definition sheet follow below.

A. The following variables must be specified by a variable card:

(1) Any variable that is a vector.

(2) Any variable that is a matrix.

(3) Any variable that is referred to in a Pact I step and which does not occur in any Pact I step containing an Equals (EQ) operation.

(4) Any variable that is constrained by any one of the following types of constraints: SYN, SUC, IMS or REL (see page 04·31·21.

B. There must be at least one variable card for each program; this can be a blank card having a 12 punch in card column 80.

C. The last variable card must have a 12 punch in card column 80.

D. Only one variable can be specified on each variable card.

E. Columns 1 through 8 and 43 through 79 of the variable card are not used by Pact I.

F. The fields on the variable definition sheet labeled Factor, S_1 and S_2 are filled out in the manner as for the Pact I steps except that Inductive subscripts may be omitted.

G. The scaling for the variable is specified by the Q field, a sign and two digits. If the sign column is left blank, Pact I interprets this as a positive sign.

H. The maximum dimensions of any array must be specified in the fields labeled D_1 and D_2 which must be filled with either a blank, zero or positive integer. A blank, zero or 1 are all interpreted as 1 by Pact I. Any positive integer greater than 1 represents the maximum number of possible positions which can be assumed by the corresponding inductive subscripts, S_1 or S_2.

(1) A scalar having no definitive subscripts can be written one of the following ways:

FACTOR	S_1	S_2	Q	D_1	D_2
X			4		
X			4	1	1
X			4	0	0

A scalar having a definitive subscript can be written as:

X	2		4	1	1

(D_1 and D_2 could be blank or zero.)

(2) A variable is a matrix if both D_1 and D_2 are equal or greater than 2. The following information on the variable definition sheet could indicate a matrix, A_{IJ} of 4 rows and 3 columns:

A	I	J	3	4	3

The correspondences

(A) D_1 corresponds to S_1 and only to S_1.
D_2 corresponds to S_2 and only to S_2.

are true for all matrices for both variable cards and Pact I steps.

(3) If either D_1 or D_2, but not both, is an integer greater than 1, the variable is a vector. A vector having no definitive subscripts can be expressed as:

C		5	6	

If a vector has an inductive subscript, it must satisfy the correspondences (A) above, on the variable card. For example:

D	I		10	4	
E		J	8		5

In the Pact I steps referring to these vectors either S_1 or S_2 can contain the inductive subscript.

A vector having a definitive subscript can be written:

F	A	I	8		5

A is a definitive subscript and I is inductive. Further examples are displayed on page 04·31·27.

I. The Constraint division of the variable definition sheet and the constraints themselves are discussed starting below in paragraph 9.

8. PRINTING VARIABLES. A list of the variables and their assigned relative locations are printed as one of the results of the Pact I compiling routine. The information is similar to that on the variable definition card with some exceptions. An illustration is displayed on page 04·31·95 and is described directly below.

A. The first column is the Tag, a letter representing the kind of variable:

S : scalar

V : vector

M : matrix

B. The second column gives the first full-word relative location assigned to the variable.

C. The third column gives the last full-word relative location assigned to the variable only if it is a vector or matrix. This column is blank for a scalar or number.

D. The Factor field is printed in the fourth column just as it is written on the variable definition sheet.

E. The fifth and sixth columns contain definitive subscripts, S_1 and S_2 if they exist, or Δ_1 and Δ_2 for a vector or matrix. The quantities Δ_1, and Δ_2 are called increments; for a one-dimensional array, Δ_2 is 2 and Δ_1 is 0; for a two-dimensional array Δ_1 equals $2D_2$, and Δ_2 always equals 2. Inductive subscripts are not printed.

F. In the seventh column the scaling, Q, is printed just as it is written for the variable on the variable definition sheet or code sheet in a step containing the Equals (EQ) operation. However, if Q is blank on the card, Q is assumed to be zero and printed as zero.

G. The remaining columns give information just as it is written in the Constraint section of the variable definition card, namely, the Factor, S_1, S_2 Type and Location. A discussion of constraints follows in the next paragraph.

9. CONSTRAINTS. In most programs it is expected that the variables can be assigned without regard to inter-relationships of the arrangement of the variables in storage. In the event that it is necessary to require certain

restrictions on the assignment of the storage locations for some variables, this can be done by the use of constraints. If one desires to specify a particular relative location for a variable, have one variable follow another or have the location of one variable overlap the storage location of another variable, one can accomplish this by the use of one of the constraints. On the variable definition sheet, there are two main sections, Variable (card columns 9 through 26) and Constraint (card columns 27 through 42).

A. A variable which has no constraint is an independent variable; also, a variable defined only on the code sheet by a step containing an Equals (EQ) operation is an independent variable. The ultimate relative locations of variables defined on the variable definition sheet depend only upon the order in which the cards containing the independent variables are read. The order of the relative locations of variables defined on the code sheet is not necessarily the same order in which they are written in the program. However, as a group they follow the independent variables defined on the variable definition sheet.

B. A variable can appear only once in the variable section of the variable definition sheet of a program.

C. When two or more variables are permitted to occupy the same storage location, an overlap of storage occurs. Then the program can be thought of as being broken into successive phases. Each variable which occupies the same storage location as others must be in a separate phase. The Pact I system provides for ten phases.

D. A relative variable is one whose relative location is specified on the variable definition sheet by writing REL in the Type field and the desired relative location, a positive even integer or zero, in the Location field. The other fields in the Constraint section are left blank.

E. A chain is the set of all those variables beginning with either an independent variable or relative variable and all the variables whose assignment depends upon the assignment of the generating independent variable or relative variable.

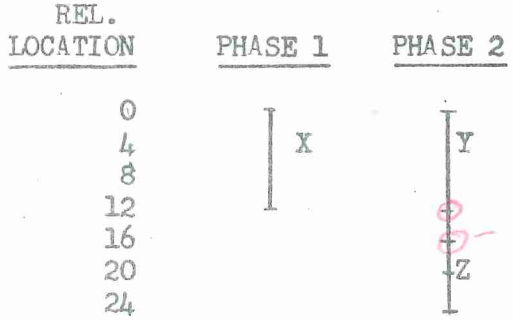
F. A subchain of a variable is defined exactly the same as a chain, except that any variable can generate a subchain of that same variable.

G. When it is desired to permit the overlapping of storage locations, the SYN (synonym) constraint is used. If the programmer wishes to place a given variable in the same storage location as another variable, hereafter called the constraining variable, the variable with any subscripts is written in the variable section of the variable definition sheet, and the constraining variable is written in the Constraint section. The Factor, S_1, S_2 (if used) and type, SYN, should be filled in for the constraining variable. This constraining variable must have been specified elsewhere in the variable section.

The variable will be in a phase position one greater than that of the constraining variable. The programmer should realize that this constraint provides the possibility of overlap rather than actual overlap of storage in every instance. If, for example, the following information is written on the variable definition sheet,

~~SUBJECT~~ VARIABLE							CONSTRAINT			
FACTOR	S_1	S_2	Q	D_1	D_2	FAC	S_1	S_2	T	LOC
X				6						
Y				8		X			SYN	
Z					4	X			SYN	

the variable, X, will be assigned storage locations. In the next phase, the variable, Y, will be assigned storage locations starting at the same relative location as the variable X. Since the variable Y will occupy more storage locations than the variable X, Z then will not overlap the storage locations of X but follow the variable Y in storage. Using lines to represent consecutive relative storage locations, we have:



Z follows Y only because the Y variable card was read by the compiler before the Z variable card.

H. The SUC (succeeds) constraint can be used for two purposes:

(1) The programmer can specify the order in which any two variables in the same phase shall be assigned, independent of the order of the original deck of variable cards.

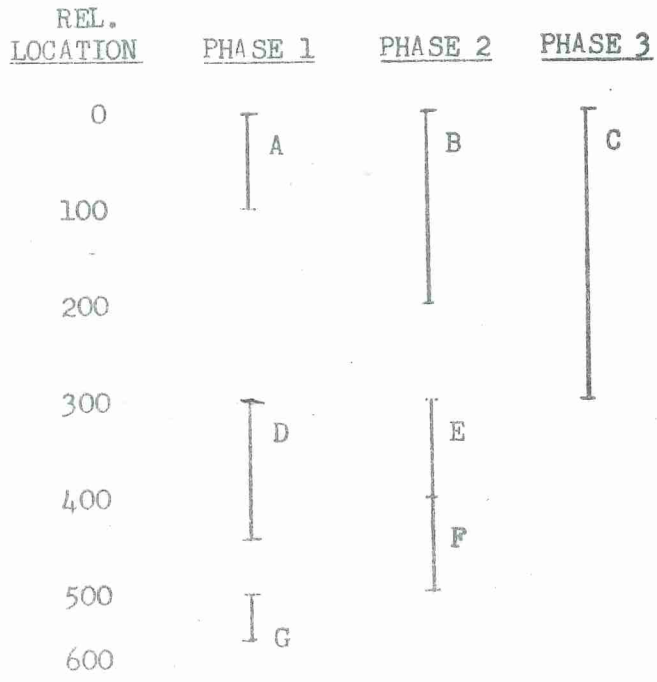
(2) The programmer can specify that two variables shall be in the same phase, assuming the order is unimportant. Normally, if a variable card contains a variable to be assigned, a constraining variable and the constraint type, SUC, the variable will be the next one assigned in the same phase as the constraining variable and no variable contained in the subchain of the constraining variable excluding the subchain of the variable will be allowed to overlap the variable to be assigned. This implies that the variable to be assigned might not follow immediately after the constraining variable.

For example,

~~SUBJECT~~ VARIABLE						CONSTRAINT				
FACTOR	S_1	S_2	Q	D_1	D_2	FAC	S_1	S_2	T	LOC
A				50					REL	0
B				100		A			SYN	
C				150		B			SYN	
D				75		A			SUC	
E				50		D			SYN	
F				25	2	D			SYN	
G				5	5					

A, B, C, D, E, and F form a chain of the constraining variable A (A is the constraining variable with the SUC constraint). Furthermore, D, E, and F form a subchain of the variable D.

Using a line diagram we have,

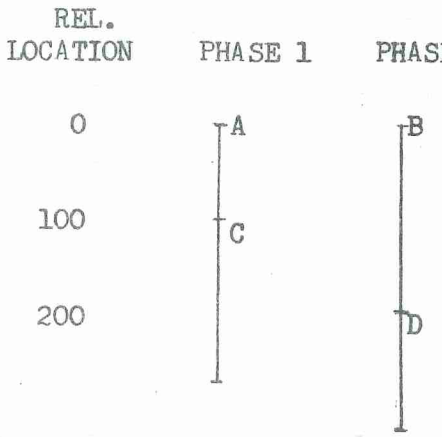


In phase 1, D is assigned next after A rather than G. A, B, and C which form a chain of the constraining variable A excluding the subchain of the variable D, are not allowed to overlap D.

I. The IMS (immediately succeeds) constraint performs the same function as the SUC constraint except that the variable is assigned immediately after the constraining variable in storage. It is possible that the chain of the constraining variable may overlap the variable assigned.

For example,

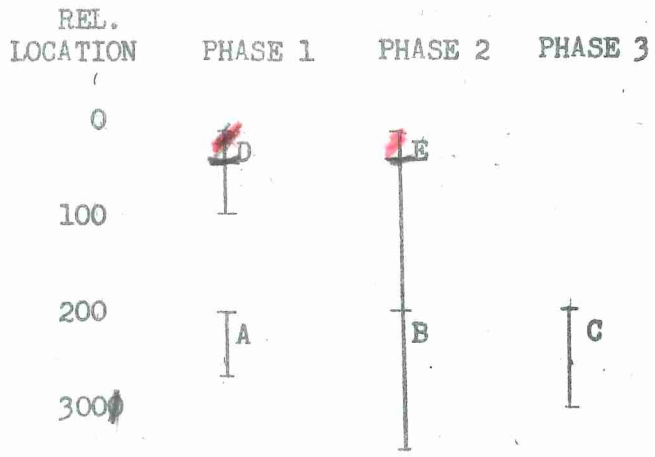
~~SUBJECT~~ VARIABLE						CONSTRAINT				
FACTOR	S_1	S_2	Q	D_1	D_2	FAC	S_1	S_2	T	LOC
B				10	10	A			SYN	
A				50						
C				75		A			IMS	
D				50		A			SYN	



The chain of A, excluding the chain of C, is allowed to overlap the variable C. This is the main distinction between IMS and SUC.

J. A relative chain is generated by a relative variable and an independent chain is generated by an independent variable. A relative chain takes precedence over any independent chain, i.e. no independent chain is allowed to intersect a relative chain. The term intersect in connection with chains is similar to overlap with variables. For example,

~~SUBJECT~~ VARIABLE						CONSTRAINT				
FACTOR	S_1	S_2	Q	D_1	D_2	FAC	S_1	S_2	T	LOC
A				5	5					
B				3	25	A			SYN	
C				5	10	B			SYN	
D				50					REL	50
E				75		D			SYN	



If A were assigned a relative location of 0 by the compiler, the independent chain A, B, C would intersect the relative chain D, E. That is, part of B and C would overlap D and E.

Some possible applications of the relative constraint are:

(1) To integrate two Pact I programs. This means that the same variable occupies the same storage locations in two different programs.

(2) To enable the programmer to make his own storage allocation. However, there can be a maximum of 120 relative variables in a program.

(3) To effect storage allocation involving complicated overlapping and ordering which cannot be effected by the other constraints.

(4) To modify a Pact I program which has been previously compiled.

(5) To preserve certain regions of storage.

SAMPLE VARIABLES

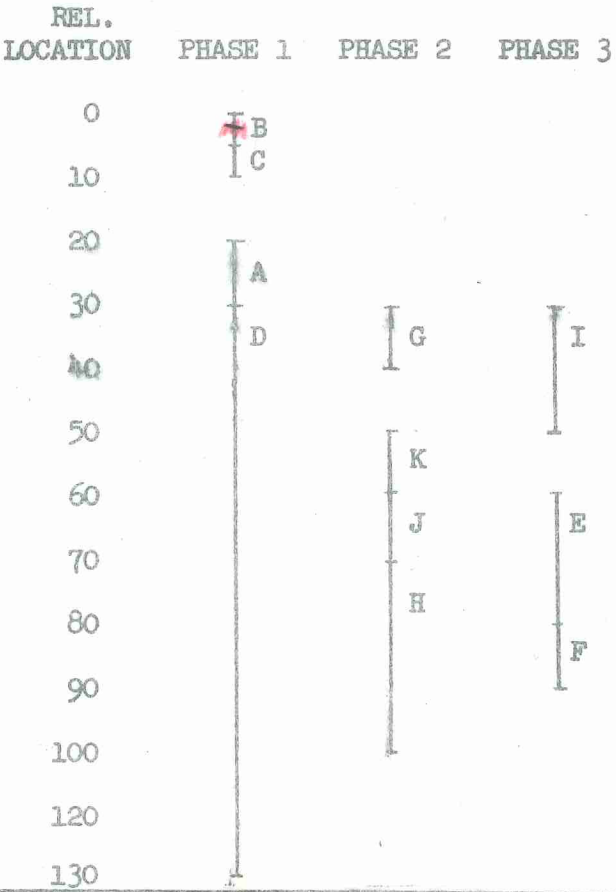
INFORMATION ON VARIABLE DEFINITION SHEET							EXPLANATION	
FAC	S_1	S_2	±Q	D_1	D_2	KIND OF VARIABLE	KIND OF SUBSCRIPT	INCREMENTS
A	I	J		5	10	5 x 10 array	I Inductive, J Inductive	Δ_1 is 20, Δ_2 is 2
B	K			10	10	10 x 10 array	K Inductive	" "
C		L		10	10	10 x 10 array	L Inductive	" "
D				10	10	10 x 10 array		" "
E	I			10		1 x 10 vector	I Inductive	Δ_1 is 0, Δ_2 is 2
F		J			10	1 x 10 vector	J Inductive	" "
G				10		1 x 10 vector		" "
H					10	1 x 10 vector		" "
I	0				10	1 x 10 vector	0 Definitive	" "
J	0	K			10	1 x 10 vector	0 Definitive, K Inductive	" "
AO					10	1 x 10 vector		" "
K						1 x 1 scalar		Δ_1 is Δ_2 is 0
L	P			0	0	1 x 1 scalar	P Definitive	" "
M	0	X		1	1	1 x 1 scalar	0 Definitive, X Definitive	" "
DPQ						1 x 1 scalar		" "
M	0	Y				1 x 1 scalar	0 Definitive, Y Definitive	" "
N	I	Y		10		Not allowed	Definitive subscript should be in S_1	
O		Y		10		Not allowed	" " " "	
P		Z				Not allowed	" " " "	

The following variables would not be allowed in the same compilation with the above variables:

A	0				10	Not allowed	An array above has the factor A
D	P	Q				Not allowed	An array above has the factor D
J	0	1				Not allowed	A vector above has the factor J and definitive subscript 0

EXAMPLE USING ALL TYPES OF CONSTRAINTS ON VARIABLES

VARIABLE						CONSTRAINT					12 Last
9-11	12-14	15-17	18-20	21-23	24-26	27-29	30-32	33-35	36-38	39-42	80
FAC	S1	S2	±Q	D1	D2	FAC	S1	S2	TYPE	LOC	
A					5				REL	20	
B											
C					5						
D				5	10						
E				10		J			SYN		
F					5	J			SYN		
G				5		D			SYN		
H				3	5	J			IMS		
I					10	G			SYN		
J				5		K			SUC		
K					5	D			SYN		Y



FORM 16-G-1 REV. 2-47

The order in which the variables in the example on the opposite page are assigned is as follows:

1. B is assigned first, because it is the first independent variable, and the independent chain generated by B does not intersect any relative chain.

2. Similarly, C is the second independent variable and independent chain.

3. D is the next independent variable, and would be assigned beginning at relative location 12, except that the chain of D (i.e., the variables D, G, I, K, J, E, H, and F) intersects the chain of the relative variable A. Therefore, A is assigned according to the relative constraint.

4. Now D is assigned beginning immediately after A.

5. G is assigned next, since it is the first variable "X" such that "X" SYN D.

6. I is assigned next, since it is the only variable "X" such that "X" SYN G.

7. K is the next variable "X" such that "X" SYN D. Note that K is assigned immediately after the subchain of G (i.e., the variables G, I). Observe that K SUC G would accomplish the same result.

8. J is the only variable "X" such that "X" SUC K.

9. E is the first variable "X" such that "X" SYN J.

10. F is the next such variable.

11. H is the only variable "X" such that "X" IMS J. Note (in contrast to 7) that H follows immediately behind J, allowing E and F to overlap H.

NUMBERS

1. The Numbers are numeric constants which are required whenever the program is run. They are written on the code sheet either in the factor field or the Number field. If a number is written in the factor field, it must be an integer having not more than three digits. When a larger number or a number having a fractional part is desired, it is written in the Number field. The Number field is described in connection with the code sheet in the section, Form of Instructions.

2. Unless the programmer specifies otherwise, the relative locations of the Numbers are assigned sequentially starting at 2. Each number appears on the listing of the program in the step in which it was written by the programmer and its relative location is given in a corresponding machine language instruction.

3. If the programmer so desires, he can specify the relative location for the origin of the Numbers; this relative location can be any even, positive integer, not zero. It can be specified only once in a program, and is indicated in the constraint section of the variable definition sheet by writing NUM in the type field and the desired relative location in the location field. No variable can be written on this same variable card. If the variable card having a NUM constraint is the last variable card, it must contain a 12 punch in card column 80. Although the Numbers are not printed, this specified relative location with the constraint type, NUM, is printed on the variable listing following all the variables.

TEMPORARY STORAGE

1. During the course of the execution of a program it is frequently necessary to store intermediate results, or, in other words, to have some working storage. Also, it is convenient to be able to refer to the result of a step without naming the result explicitly; Pact I relieves the programmer of the tedious accounting of these temporary storage locations. Within each region temporary storage locations are automatically assigned by Pact I as needed.

2. In Pact I it is possible to refer to the results of a step within a region by means of a clue, R or N, and the corresponding step number in the factor field. If a Pact I step has the operation and factor, + R 010.0, the result of step 10 is added to the result of the previous step. Or, if the operation and factor are X N 012.0, the result of the previous step is multiplied by the negative of the result of step 12. The Pact I routine assigns a temporary location to each desired intermediate result indicated by the clue, R or N. The Q must be specified for any step whose results are called for by another step having an R or N in the clue column of the factor. Also, the programmer may refer only to steps within the same region. The clue, R or N, may be used with any arithmetic operation, including the Equals (EQ) operation. The contents of the temporary storage locations utilized by the R and N factors are destroyed when another region is entered by a Do Region and Return (DO) or when an Exit (EXIT) operation is executed.

3. If the result of a step is used as a factor, a step is inserted in the Pact program having the Equals (EQ) operation and the result of the step desired as the factor. This step does not have a step number but the step is printed. For example, the second instruction in the example immediately below is printed as indicated although it does not appear in the program as originally written:

REG	STEP	OP	FACTOR	S_1	S_2	Q
DM	4.0	X	K			12
DM	.	EQ	R 4.0			12
. . .						
DM	10.0	/	R 4.0			6

4. The temporary location 0 is used for several different quantities during the execution of one region. The use of this location is automatic and beyond the control of the programmer.

LOOP WRITING

1. INTRODUCTION. An internally stored program machine has the great advantage of being able to modify its own instructions and thereby execute the instructions in the same locations many times, but with the addresses modified each time. The Pact I routine can generate loops in the compiled program. To accomplish this, the initial addresses are set up, the addresses are modified and provision is made to escape the loop after the desired number of repetitions.

2. It is relatively easy for the programmer to write Pact I steps to generate loops. The operations used in setting addresses and generating loops are SET and TEST. Each loop must be entirely within one region. The SET's and TEST's must occur in ordered pairs, each SET requiring a corresponding TEST, and no TEST appearing without a corresponding SET. If the setting of addresses without testing is needed, the operation USE is available for this purpose. A USE referring to a subscript which has not been SET, means to set up all necessary instructions, but not be paired with a TEST. It is possible, however, to have a USE between a SET and a TEST pair. Subscripts which are the objects of a SET, USE, or TEST can be equated to a number or the value of a variable (scalar) as well as compared to a number or the value of a variable (scalar). When a variable is indicated, it must be present at a Q of 17. It is particularly important to remember that such a number or variable cannot be zero when used in connection with subscripts. In conventional coding, escape from a loop can be effected either by a test on an address or a counter type test. The test on an address is shorter both with respect to space and execution time. ~~However, there are several occasions when a counter type test is required for a properly compiled code. The methods of generating both types of tests, and restrictions on their use will be described in paragraphs 6 and 7.~~

3. INITIAL ADDRESSES FOR VARIABLES THAT WILL NOT BE MODIFIED. When the steps for a program are written, a particular member of an array may be desired. There are several such situations discussed below.

A. Perhaps the simplest way to specify a particular member of an array is to use single or double numeric subscripts. For example, if it is desired to use one specific member, A_{23}, of a 2-dimensional array, the designation A in the factor field, 2 in the S_1 field and 3 in the S_2 field, causes A_{23} to be selected. The address of a quantity so specified is not modified during the program execution.

B. When the subscripts of an arrayed variable (1 or 2 dimensional array) are fixed by the operation Use (USE), no modification of address takes place during the execution of the program. Two typical situations are given as examples; the steps might be included in a sequence of steps that form a loop.

(1) Numerical subscripts

REG	STEP	OP	FAC	S_1	S_2	Q
K	8.0	USE		I	2	
K	9.0	USE		J	3	
K	10.0		A	I	J	

not necessary

(This can be read as: Use I equals 2, Use J equals 3, Take A_{IJ}.) This combination of steps could be used within a loop in which I and J take on various values, but in step 10.0, A_{23} is always taken.

better to write $A_{2,3}$

(2) Alphabetic subscripts

REG	STEP	OP	FAC	S_1	S_2	Q
L	13.0	USE		I	M	
L	14.0	USE		J	N	
L	15.0		A	I	J	
L	16.0	X	G			

The variables M and N are scalars and must be at a binary magnitude of 17. The numerical values of M and N may be calculated by the program or loaded as input data prior to the execution of this part of the program.

4. ADDRESSES MODIFIED ON ONE SUBSCRIPT. If a value, number or variable, is assigned to a subscript by means of the Set (SET) operation, the addresses of instructions using this subscript will be set up at program execution time and will remain the same until a Test (TEST) is encountered. Then the subscript is increased by one, and control is transferred according to the definition of the Test operation. An example of such a loop follows:

AK	1.0	SET		I	1	
AK	2.0	CL	A	I		
AK	3.0	TEST		I	10	

These steps clear A_i, i = 1 to 10; the ten consecutive full-word locations beginning at the location of A_1 are cleared to zero. The programmer may wish to use a particular member of the array in a loop each time the loop is repeated. The operation Use (USE) will accomplish this. An example of this situation is:

RAE	12.0	SET		I	1	
RAE	13.0	USE		J	N	
RAE	14.0		A	J		
RAE	15.0	EQ	A	I		
RAE	16.0	TEST	14.0	I	6	

Once a subscript is the object of a USE, it remains at that status for the entire region unless another USE or a SET on this subscript follows before the end of the region. Thus step 15 is modified at step 16, while step 14 remains the same. Another situation may occur when a variable having double subscripts is used, but it is desired to generate a loop using only one of the subscripts. Then this subscript is set by the Set operation but the other subscript remains the same throughout the loop. For example:

M	1.0	SET		I	1	
M	2.0	CL	A	I	3	
M	3.0	TEST		I	6	

These steps will clear the locations of A_{13}, A_{23}, ..., A_{63} to zero. Only one Set (SET) step and one Test (TEST) step are required in this situation.

5. ADDRESSES MODIFIED ON TWO SUBSCRIPTS. The address of a variable having double subscripts can be modified on both of the subscripts; this requires two Set operations and two Test operations. The Test operations are written in the inverse order of the Set operations with reference to the subscripts. The following example illustrates this type of address modification:

REG	STEP	OP	FAC	S_1	S_2	Q
F	1.0	SET		I	1	
F	2.0	SET		J	1	
F	3.0	CL	A	I	J	
F	4.0	TEST		J	7	
F	5.0	TEST		I	5	

The locations containing A_{11}, A_{12},..., A_{17}, A_{21}, A_{22},...A_{27},..., A_{57} will be cleared to zero.

For double subscripted variables, the initial address is set up in the compiled program at the step in which the first, or outside, subscript was set. Thus, in the above example, no set up orders would appear at step 2 in the compiled program.

6. TEST FOR ESCAPE FROM LOOP. The number of times a sequence of orders is repeated depends on the amount against which a subscript is tested and the value to which it was SET. As noted in paragraph 2 above, two types of tests can be written. The type depends upon the kind of subscript, numeric or alphabetic, in the S_2 field of the Test step.

A. The test will be on addresses when the quantity in the S_2 field is numeric. An example follows using a variable having double subscripts:

KL	1.0	SET		I	1	
KL	2.0	SET		J	1	
KL	3.0		A	I	J	
KL	4.0	X	R			
KL	5.0	EQ	A	I	J	
KL	6.0	TEST		J	4	
KL	7.0	TEST		I	3	

Both tests will be address type tests.

B. The next example shows that interlocking subscripts are permissible.

REG	STEP	OP	FAC	S_1	S_2	Q
FGK	1.0	SET		I	W	
"	2.0	"		J	2	
"	3.0	"		K	1	
"	4.0		A	I	J	
"	5.0	X	B	J	K	
"	6.0	EQ	C	I	K	
"	7.0	TEST		K	7	
"	8.0	"		J	5	
"	9.0	"		I	3	

All tests will be the address type.

C. A counter type test is supplied by Pact I when an alphabetic character is placed in the S_2 field of the test step. In the following example both tests are the counter type.

G	1.0	SET		I	1	
"	2.0	"		J	1	
"	3.0	CL	A	I	J	
"	4.0	TEST		J	V	
"	5.0	TEST		I	W	

D. The programmer may wish to give one numeric character and one alphabetic character in the S_2 field of the test steps. The type of tests supplied by Pact I depends upon the order in which these test steps are written. If the first test step contains a number in the S_2 field and the second test step contains an alphabetic character, the first test is made on addresses and the second on a counter. For example:

BD	1.0	SET		I	1	
"	2.0	SET		J	1	
"	3.0	CL	A	I	J	
"	4.0	TEST		J	5	
"	5.0	TEST		I	W	

The test on J is an address and the test on I is a counter.

E. However, if the information in the S_2 field is given in the reverse order, both tests will be the counter type. In the following example counter tests are used for both I and J.

REG	STEP	OP	FAC	S_1	S_2	Q
BG	1.0	SET		I	1	
"	2.0	"		J	1	
"	3.0	CL	A	I	J	
"	4.0	TEST		J	W	
"	5.0	TEST		I	5	

7. REMARKS ON LOOP GENERATION

A. There are certain conditions that require counter tests rather than address tests. One condition is encountered when a variable with a double subscript occurs in the loop, one subscript is designated by a Use (USE) step, the other is indicated by a Set (SET) step, and there is no single subscripted variable with the subscript that was set. This is illustrated by the example below.

E	1.0	USE		I	W	
"	2.0	SET		J	1	
"	3.0	CL	A	I	J	
"	4.0	TEST		J	P	

The test on J is a counter (a test against a symbol); it is improper to place a number in the S_2 field of step 4.0. The locations containing A_{WJ}, where J = 1, 2...P are cleared to zero.

B. Another condition which requires a counter (or symbol) test is the case where no subscripted variable occurs between the SET and TEST operations.

B	1.0	SET		I	1	
"	2.0		M			
"	3.0	X	-1			
"	4.0	EQ	M			12
"	5.0	TEST		I	N	

SUBROUTINES AND LIBRARY PROGRAMS

1. INTRODUCTION. When programming for a high speed digital computer, it is frequently to one's advantage to make use of available routines written independently and in a generalized sense, or even to write subroutines applicable to certain types of problems. Examples of subroutines include a general type input or output routine, elementary functions, or some polynomial approximation. It is necessary to have some means of transmitting required information between the various blocks or routines of a complete program. This information includes initial locations for arguments, magnitudes of numbers, etc. A common solution to this problem is to use a standardized calling sequence to relay messages between routines or program blocks.

2. CALLING SEQUENCE. This calling sequence, or linkage, tells the generalized routines where to find the required information and where to return after completing its function. Suppose that the generalized routine is an input subroutine which needs only the starting location for storing data and the return point to perform its function. Then, the calling sequence may be:

LOCATION	INSTRUCTION		REMARKS
a	+Reset and add	a	Return address anchor
a + 1	+Transfer	k	Entry point of subroutine
a + 2	-Halt	L(b)	Location to store first piece of data
a + 3			Return point

This calling sequence makes the necessary information available to the input subroutine. Thus, this one input subroutine can be used with any program or from different points within the same program. There are operations, (DO, LIB, CALL, ID, and FOR), in Pact I which produce calling sequences similar to the one in this paragraph.

3. SUBROUTINE - TYPE OPERATIONS. There are six Pact I operations which may be coded in the same manner as arithmetic operations, but which actually produce the proper calling sequence for the desired subroutine. These operations are:

Sine	SIN
Cosine	COS
Square root	SQRT
Exponential	EXP
Natural logarithm	LOG
Arctangent	ARCT

For example, if the programmer wishes to obtain sin X, it is only necessary to write the step

REG	STEP	OP	FAC	S_1	S_2
AL	2.0	SIN	X		

The relative instructions corresponding to this step comprise the calling sequence for sine subroutine. Information concerning scaling restrictions and the calling sequences of each of these subroutine type operations are given in Table 3 of the Reference Tables.

4. EXISTING SUBROUTINES IN MACHINE LANGUAGE. Any installation that has been operating for even a short time undoubtedly has a library of subroutines already in use. These existing subroutines can be utilized with ease. The Pact I operation, Library Program (LIB), in conjunction with the Identification (ID) and For (FOR) operations can be used to develop a set of machine instructions which is identical to the required calling sequence. It is necessary for the programmer to know the required calling sequence for each subroutine so that he can write Pact I operations that will be converted into this sequence. Details concerning the Pact I operations involved in such a sequence are given below in paragraph 6. The subroutines may be available to the Pact I program either on a so-called library tape or on binary cards; these library subroutines may be incorporated in the program during compiling or loading.

5. PACT I SUBROUTINES. Since Pact I is built around the master routine - subroutine concept, there must be some facile method of going from one region to another region of the program. A region is considered to be a self-contained logical group of steps. In fact, a Pact I program can be made up of a series of such regions. The operation, Do Region and Return (DO), offers the means of a transfer to another region and return. If the DO operation is followed by Identification (ID) and For (FOR) operations, information can be transmitted at the same time from one region to another region. However, the region which is indicated in the factor field of a step having the DO operation must begin with a Call (CALL) operation followed by the same number of ID and FOR operations in the same sequence and of the same type. This is also true for any library programs that are written in the Pact I program.

6. DETAILS OF OPERATIONS ASSOCIATED WITH A CALLING SEQUENCE. As the Pact I routine converts the Pact I steps written by the programmer to the relative instructions in machine language, the machine language instructions are influenced by the information placed in the factor field and subscript fields and also the order in which the Pact I operations are written. The functions and uses of operations associated with a calling sequence are explained in considerable detail directly below.

A. The Library Program (LIB) operation is normally converted into two machine instructions, a return address anchor and an unconditional transfer to the location specified. In addition, the result of the previous step is placed in the machine MQ register so that the information can be used by the subroutine.

B. The operation, Do Region and Return (DO), is converted into two machine instructions, a return address anchor and an unconditional transfer to the first instruction of the region specified. This operation is used to refer to other regions within the program itself rather than subroutines on tape or cards.

C. In the case of Identification (ID), the forms depend upon their position in the program as well as other fields used, such as factor S_1, S_2, or Q.

(1) A simple use of this operation is illustrated by this example:

PACT STEP							MACHINE LANGUAGE		
REG	STEP	OPER	FACTOR	S_1	S_2	Q	LOC	OP	ADDRESS
AB	12.0	LIB	DIA				a	+RA	a
							a + 1	+T	L (Region DIA)
AB	13.0	ID	K				a + 2	-H	L(K)
AB	14.0	ID	4				a + 3	+H	4

Note than an alphabetic character in the factor field of the ID step yields the machine instruction -H L(K) while a number in the factor field yields +H 4.

(2) If the ID operation with a subscripted variable follows either a LIB or DO operation and is not in a loop, these Pact I steps and their expanded instructions are:

B	5.0	LIB	KY				a	+RA	a
							a + 1	+T	L (Region KY)
B	6.0	ID	A	I	J		a + 2	-H	L (A_{11})
							a + 3	+H	Δ_1
							a + 4	+H	Δ_2

where Δ_1 and Δ_2 are the storage increments of the left and right subscripts, respectively.

(3) If the ID operation with a subscripted variable follows either a LIB or DO operation and is inside a loop on both subscripts, the Pact I steps and relative instructions (for LIB and ID only) follow:

PACT STEP							MACHINE LANGUAGE		
REG	STEP	OPER	FACTOR	S_1	S_2	Q	LOC	OP	ADDRESS
C	4.0	SET		I	1				
C	5.0	SET		J	1				
. .	. . .	. . .							
C	8.0	LIB	LM				a	+RA	a
							a + 1	+T	L(LM)
C	9.0	ID	A	I	J		a + 2	-H	$L(A_{11})_{IJ}$
. .	. . .	. . .							
C	12.0	TEST		J	6				
C	13.0	TEST		I	6				

Notice that in this situation there is only one relative instruction corresponding to the ID step. Also, the address of this relative instruction is modified each time this loop is repeated.

(4) When the ID operation with a subscripted variable follows either a LIB or DO operation and is inside a loop on only one subscript, the Pact I steps and relative instructions (for LIB and ID only) follow:

D	3.0	SET		I	1				
. .	. . .	. . .							
D	7.0	USE		J	3				
D	8.0	LIB	DK				a	+RA	a
							a + 1	+T	L (DK)
D	9.0	ID	A	I	J		a + 2	-H	$L(A_{13})$
							a + 3	+H	Δ_2
. .	. . .	. .							
D	15.0	TEST		I	4				

FORM 18-6-1 REV. 2-47

where Δ_2 is the storage increment of the subscript that is not in the loop. A Use (USE) step is required to define the subscript that is not inside a loop; otherwise, the loop writing routine would look in vain for a SET and TEST on this subscript.

D. The operation FOR will be expanded by the Pact I routine so that either two or three machine language instructions are produced. A blank factor is the same as a zero, but S_1 and S_2 must not be blank.

(1) If the factor and both subscript fields contain information, there are three machine instructions corresponding to this Pact I step. If there are numbers in the three fields, these numbers become the addresses in the machine language instructions.

PACT STEP							MACHINE LANGUAGE		
REG	STEP	OPER	FACTOR	S_1	S_2	Q	LOC	OP	ADDRESS
BK	9.0	LIB	ABC				a	+RA	a
							a+1	+T	L (Region ABC)
BK	10.0	FOR	2	5	6		a+2	+H	2
							a+3	+H	5
							a+4	+H	6

(2) If there are variables in the factor and subscript fields, the locations of these variables become the addresses in three machine language instructions:

BM	7.0	LIB	BDE				a	RA	a
							a + 1	T	L (Region BDE)
BM	8.0	FOR	G	H	T		a + 2	-H	L(G)
							a + 3	-H	L(H)
							a + 4	-H	L(T)

(3) A combination of numbers and variables may be used in the factor and subscript fields. The use of a variable results in an address which is the location of the variable. For example:

PACT STEP							MACHINE LANGUAGE		
REG	STEP	OPER	FACTOR	S_1	S_2	Q	LOC	OP	ADDRESS
MN	12.0	LIB	BM				a	+RA	a
							a + 1	+T	L(Region BM)
MN	13.0	FOR	A	3	D		a + 2	-H	L(A)
							a + 3	+H	3
							a + 4	-H	L(D)

E. An important use of the FOR operation occurs with variables having subscripts. The variable can be specified in a Pact I step having the Identification (ID) operation, and the subscript* associated with the variable is indicated in the factor field of a Pact I step having the For (FOR) operation. For example:

MP	9.0	LIB	MMP				a	+RA	a
							a + 1	+T	L (Region MMP)
MP	10.0	ID	A	I	J		a + 2	-H	L (A_{11})
							a + 3	+H	Δ_1
							a + 4	+H	Δ_2
MP	11.0	ID	B	J	K		a + 5	-H	L(B_{11})
							a + 6	+H	Δ_1
							a + 7	+H	Δ_2

* The symbols used as subscripts here must not be used as variables in any other portion of the problem; for if the symbol(s) in the factor represent a variable then the FOR operation is assumed to be transmitting 3 independent pieces of information.

PACT STEP							MACHINE LANGUAGE		
REG	STEP	OPER	FACTOR	S_1	S_2	Q	LOC	OP	ADDRESS
MP	12.0	ID	C	I	K		a + 8	-H	$L(C_{11})$
							a + 9	+H	Δ_1
							a + 10	+H	Δ_2
MP	13.0	FOR	I	1	10		a + 11	+H	1
							a + 12	+H	10
MP	14.0	FOR	J	1	5		a + 13	+H	1
							a + 14	+H	5
MP	15.0	FOR	K	1	6		a + 15	+H	1
							a + 16	+H	6

F. The operation Call (CALL) is similar in effect to Do Region and Return (DO). It acts as a language translator between the main routine and a subroutine.

7. REMARKS. The examples above indicate that it is possible to generate the required calling sequence by means of the Identification (ID) and For (FOR) operations in conjunction with either Library Program (LIB) or Do Region and Return (DO) operations. These operations afford a flexible method of transmitting information between regions of the program and between the main program and subroutines. It is essential that proper sequence of steps be provided by the programmer so that the subroutine or other region can be used correctly.

8. THE CALL OPERATION. The Call (CALL) operation may be used to generate instructions which will interpret the Pact I steps having Identification (ID) or For (FOR) operation in any region which is not the object of a Duplicate (DUP). The compiler will place a CALL in front of such a region if there is none there; however, if the DO corresponding to the CALL is followed by ID's and/or FOR's, the CALL must be written by the coder. Otherwise, only the exit from the region containing the CALL will be set, and no means of examining the ID's and FOR's will be provided. The CALL is actually a signal to the compiler that this region is to be called by a DO operation, possibly from many different points within the program. Therefore, some means must be provided to interpret varying information in ID's and FOR's following a DO. The function of the CALL can best be explained with examples.

A. Suppose we have a problem containing a series of matrix multiplications. By using the CALL operation we can write one region which will do all of these multiplications for us. First, let us write the multiplication $A_{IJ} B_{JK} = C_{IK}$ where each matrix is a 3 x 3 matrix with interer elements.

REG	STEP	OPER	FACTOR	S_1	S_2	Q
MP	1	SET		I	1	
MP	2	SET		K	1	
MP	3	CL	C	I	K	
MP	4	SET		J	1	
MP	5		A	I	J	
MP	6	X	B	J	K	35
MP	7	+	C	I	K	
MP	8	EQ	C	I	K	35
MP	9	TEST		J	3	
MP	10	TEST		K	3	
MP	11	TEST		I	3	

Now we must some how generalize this region so that we can call the factors anything we wish and use any size matrices we wish. Of course, we will assume that the coder will only multiply matrices which are conformable. To do this, we simply write a CALL operation as the first operation of the region and follow the CALL with ID's and FOR's naming the items to be changed. In our example these steps would be:

MP	0	CALL	MP			
MP	0.1	ID	A	I	J	
MP	0.2	ID	B	J	K	
MP	0.3	ID	C	I	K	
MP	0.4	FOR	I	1	3	
MP	0.5	FOR	J	1	3	
MP	0.6	FOR	K	1	3	
MP	1	SET		I	1	

These steps tell the compiler that immediately following all DO's referring to this region, MP, there will be three steps with ID operations and three steps with FOR operations. The factors of the ID's will give the names of the matrices to be used in the multiplication in the corresponding order - (left matrix first, right matrix second, and answer matrix last). The FOR's will define the subscript limits in the same order - left subscript of the first matrix, right subscript of the first matrix (left subscript of second matrix), and the right subscript of the second and last matrices. The compiler will then write the interpreting instructions which will cause the subprogram, region MP, to use the proper factors.

To compute $X_{AB} \; Y_{BC} = Z_{AC}$ in region CON, all we have to do now is to give the following steps.

REG	STEP	OPER	FACTOR	S_1	S_2	Q
CON	10	DO	MP			
CON	11	ID	X	a	b	
CON	12	ID	Y	b	c	
CON	13	ID	Z	a	c	
CON	14	FOR	a	1	5	
CON	15	FOR	b	1	10	
CON	16	FOR	c	1	6	

Therefore, to perform the two matrix products

$A_{IJ}\ B_{JK} = C_{IK}$ and $X_{AB}\ Y_{BC} = Z_{AC}$ we would write the following steps, assuming, of course, that the magnitudes of the corresponding matrices are the same.

CON	1	DO	MP			
CON	2	ID	A	I	J	
CON	3	ID	B	J	K	
CON	4	ID	C	I	K	
CON	5	FOR	I	1	3	
CON	6	FOR	J	1	3	
CON	7	FOR	K	1	3	
CON	8	DO	MP			
CON	9	ID	X	A	B	
CON	10	ID	Y	B	C	
CON	11	ID	Z	A	C	
CON	12	FOR	A	1	5	
CON	13	FOR	B	1	10	
CON	14	FOR	C	1	6	
CON	15	DO	PRT			
CON	16	HALT	1			

MP	1	CALL	MP			
MP	2	ID	A	I	J	
MP	3	ID	B	J	K	
MP	4	ID	C	I	K	
MP	5	FOR	I	1	3	
MP	6	FOR	J	1	3	
MP	7	FOR	K	1	3	
MP	8	SET		I	1	
MP	9	SET		K	1	
MP	10	CL	C	I	K	
MP	11	SET		J	1	
MP	12		A	I	J	
MP	13	X	B	J	K	35
MP	14	+	C	I	K	
MP	15	EQ	C	I	K	35
MP	16	TEST		J	3	
MP	17	TEST		K	3	
MP	18	TEST		I	3	

The CALL followed by ID's and/or FOR's with corresponding DO's followed by the same number of ID's and/or FOR's of the same form will make any desired region a generalized subprogram if we simply list the quantities which will vary at the top of the region.

B. Remarks:

(1) The ID's and FOR's with the DO must be in the same positions relative to the DO as the corresponding ID's and FOR's are relative to the CALL. They must be of the same form, i.e. a vector, matrix, scalar, element of a matrix, etc.

(2) The CALL does not provide for rescaling. Therefore, all corresponding variables must have the same Q's.

(3) All subscript limits must be of the same form (integer or variable). If the limit noted after the CALL is an integer, then the corresponding limit noted after the DO must be an integer.

CONTROL REGION

1. The programmer generally has written his program in logical blocks or regions. He must make provision in his program to execute a new region after the conclusion of the one just executed. It is not allowable to transfer from one region to another, so a step having the operation, Do Region and Return (DO) and the name of the next region in the factor field is necessary to insure continuity of an entire program.

2. By means of a control region, all of the regions may be called for in the order specified, which makes an orderly way of executing the program. For example, suppose regions 26, 27, 28, and 29 make up the regions of the program. The control region, called region 1 in the example, then can be written:

REGION	STEP	OP	FACTOR
1	1.0	DO	26.
1	2.0	DO	27.
1	3.0	DO	28.
1	4.0	DO	29.
1	5.0	HALT	1.

Each region, 26, 27, 28, and 29, will be executed in turn and a stop will occur at step 5.0 of the control region. The last step could be written:

1	5.0	T	1.	;

in this case, control is transferred back to step 1.0 of the control region and the entire program is repeated.

ILLUSTRATIVE PROBLEM

1. A simple problem has been coded, the program has been compiled by the Pact I routine, and then the machine language program has been executed. There are illustrations on the following pages of all the information written by the programmer, the listings produced by the Pact I routine, and the answers produced by the machine language program. Since the problem is straightforward, no detailed explanation is given.

2. The problem solved is:

$$Y_I = X_I^2 + (\sin K)X_I - 1.06e^{-M}$$

$$Z = \sum_{I=1}^{10} Y_I$$

It is desired to print the answers, Y_I, I = 1, 2, . . . 10 and Z. The variable list, each region of the program, and the answers are printed on a separate page and appear in this fashion on the succeeding pages.

CODE SHEET

PACT I CODE

JOB NO. 0533-03 DECK NO. 610-604 PROGRAMMER JAY MALLET DATE: 7-11-55 PAGE 1 OF 1

REGION	STEP	OP		FACTOR	S_1	S_2	± SIN. MAGN. Q	±	NUMBER	NOTES
9 11	12 15	16 19	20	24	25 27	28 30	31 32	33	34 44	45 79
A	10	DO		B						CONTROL PROGRAM
A	20	DO		C						
A	30	HALT		10						
B	10	READ								READ DATA
B	20			K			03			
B	30	SIN								
B	40	EQ		L			01			L IS SIN K
C	10			M			02			
	20	EXP	N	1			01			e^{-M}
	30	X					02	+	1.06	$1.06\,e^{-M}$
	35	CL		Z						SET Z TO ZERO FOR SUMMATION
	40	SET			I	1				GENERATE LOOP
	50			X	I		02			EVALUATE POLYNOMIAL
	60	+		L			03			X+L
	70	X		X	I					(X+L)X
	80	-	R	30			05			$(X+L)X-1.06e^{-M}$
	90	EQ		Y	I		05			Y SUB I
	100	+		Z			08			FORM SUM
	110	EQ		Z			08			Z
	120	LIST								PRINT Y SUB I
	130	ID		Y	I					
	140	TEST			I	10				TEST TO ESCAPE LOOP
	150	LIST								PRINT Z
C	160	ID		Z						

FORM 18-G-1 REV. 2-47

8-1-55

VARIABLE DEFINITION SHEET

PACT I — VARIABLE DEFINITION SHEET

JOB NO. 0533-03 DECK NO. 610-604 PROGRAMMER JAY MALLET DATE: 7-11-55 PAGE 1 OF 1

81-220 02

IDENTIFICATION	VARIABLE FACTOR	S_1	S_2	±	Q	D_1	D_2	CONSTRAINT FACTOR	S_1	S_2	TYPE	LOCATION	12 LAST	NOTES
1 8	9 11	12 14	15 17	18	19 20	21 23	24 26	27 29	30 32	33 35	36 38	39 42	80	DO NOT KEY-PUNCH
05330001	K			+	03									K
2	M			+	02									M
3	X	I		+	02	10								X SUB I
4	Y	I		+	05	10								Y SUB I
05330005	Z			+	08								Y	Z

12

Page 04.31.93

FORM 16-G-1 REV. 2-47

DECIMAL DATA

PACT I DECIMAL DATA

JOB NO. 0533-03 DECK NO. 6/0-604 PAGE 1 OF 1

PROGRAMMER: JAY MALLET CHECKED BY LEN BEDFORD DATE: 7-11-55

IDENTIFICATION (1–8)	LOC. (9–27–45–63)	NUMBERS (13–31–49–67, 22–40–58–76)	P (23–41–59–77)	Q (25–43–61–79, 26–44–62–80)	12-LAST	DO NOT KEY-PUNCH / DESCRIPTION
0533031	0000	+5793	+01	+03		K
	02	+3842	+01	+02		M
	04	-11	+01	+02		X_1
	06	+242	+01	+02		X_2
0533032	08	+156	+01	+02		X_3
	10	+75	+00	+02		X_4
	12	+1007	+01	+02		X_5
	14	+198	+01	+02		X_6
0533033	16	+1472	+01	+02		X_7
	18	+2012	+01	+02		X_8
	20	+289	+01	+02		X_9
	0022	+356	+01	+02	Y	X_{10}

FORM 18-G-1 REV. 2-47

VARIABLE LIST

TAG	REL. LOC. 1ST	LAST	VARIABLE FACT.	S1	S2	Q	CONSTRAINT FACT.	S1	S2	TYPE	LOC.
S	0		K			3					
S	2		M			2					
V	4	22	X		2	2					
V	24	42	Y		2	5					
S	44		Z			8					
S	46		L			1					

RUN

8-1-55

Page 04.31.95

PROGRAM LISTING

REGION A

REG	STE	P	OP	C	FAC	T	S1	S2	Q	N
A		.	CALL		A					
A	1.0		DO		B					
A	2.0		DO		C					
A	3.0		HALT		1.0					
A		.	EXIT							

LOC	OP	T	REL	OP	T	REL	OP	T	REL	OP	T	REL
0	+T	I	2	+T	A	0	+A	I	4	+SA	I	1
	+N	A	2									
5	+RA	I	5	+T	S	2						
7	+RA	I	7	+T	S	4						
11	+H	I	5									
12	+T	I	1									

PROGRAM LISTING

REGION B

REG	STE	P	OP	C	FAC	T	S1	S2	Q	N	LOC	OP	T	REL	OP	T	REL	OP	T	REL	OP	T	REL
B		.	CALL		B						0	+T	I	2	+T	A	0	+A	I	4	+SA	I	1
												+N	A	2									
B	1.0		READ								5	+RA	I	5	+T	S	6						
B	2.0				K				3		7	-RA	V	0	+LR	A	35						
B	3.0		SIN						1		11	+RA	I	11	+T	S	10	+H	A	3			
B	4.0		EQ		L				1		14	-ST	V	46									
B		.	EXIT								15	+T	I	1									

PROGRAM LISTING

REGION C

REG	STE P	OP	C	FAC	T	S1	S2	Q	N
C	.	CALL		C					
C	1.0			M				2	
C	.	EQ	R	1.0				2	
C	2.0	EXP	N	1.0				1	
C	3.0	X						2	1.060000000
C	.	EQ	R	3.0				2	
C	3.5	CL		Z					
C	4.0	SET				I	1		
C	5.0			X		I		2	
C	6.0	÷		L				3	
C	7.0	X		X		I		5	
C	8.0	-	R	3.0				5	
C	9.0	EQ		Y		I		5	
C	10.0	+		Z				8	
C	11.0	EQ		Z				8	
C	12.0	LIST							
C	13.0	ID		Y		I		5	
C	14.0	TEST				I	10		
C	15.0	LIST							
C	16.0	ID		Z				8	
C	.	EXIT							
	.0	HALT		.0			0		
C	.0	HALT		.0			0		

LOC	OP	T	REL	OP	T	REL	OP	T	REL	OP	T	REL
0	+T	I	2	+T	A	0	+A	I	4	+SA	I	1
	+N	A	2									
5	-RA	V	2									
6	-ST	T	2									
7	-RS	T	2	+LR	A	35	+RA	I	11	+T	S	12
	+H	A	32	+H	A	1	+LR	A	35			
16	-M	N	2									
17	-ST	T	4									
20	-RA	N	0	-ST	V	44						
22	+RA	I	77	+SA	I	52	+SA	I	44	+RA	I	100
	+SA	I	30	+SA	I	37						
30	-RA	V	4	-ST	T	0						
32	-RA	V	46	+LR	A	1	-A	T	0	+LR	A	1
	+LR	A	35									
37	-M	V	4	-ST	T	0						
41	-RS	T	4	+LR	A	3	-A	T	0			
44	-ST	V	24	+LR	A	3						
46	-A	V	44									
47	-ST	V	44									
50	+RA	I	50	+T	S	14						
52	-H	V	24	+H	A	5						
54	+RA	I	44	+S	I	73	+SA	I	44	+SA	I	52
	+RA	I	30	+S	I	73	+SA	I	30	+SA	I	37
	+S	I	76	+TP	I	30						
66	+RA	I	66	+T	S	14						
70	-H	V	44	+H	A	8						
72	+T	I	1									
73	+H	A	2	+H	A	1	+H	A	0	-RA	V	24
77	+H	V	24	+H	V	4						

ANSWERS

01.705130622
04.694352003
01.676430871
00.186670269
00.517226398
02.965499330
01.451044336
03.078178070
06.968780994
10.974852110
034.21816499