TUFTS COLLEGE

SYMBOLIC CODING FOR THE SIMULATION OF SYSTEMS ON DIGITAL COMPUTERS

By
Earl J. Isaac

REPORT NUMBER 1953-494-03-09

NAVY CONTRACT NUMBER N-onr-49403

TUFTS COLLEGE
MEDFORD 55, MASSACHUSETTS

CONTENTS

Incl Part

Incl Part

Omit examples

SUMMARY

The high cost of building and operating physical models, such as mock-ups and game simulations, leads to the use of symbolic models for evaluating complex man-machine systems. This engenders extensive computational problems particularly suited to solution on digital computers because of their inherent flexibility. The complexity of the system is reflected in the complexity of the set of instructions for the computer. A large number of errors are almost inevitable when the code is written directly in the machine language, and the resulting code is extremely difficult to check. The difficulty is overcome by defining an easily understood artificial language that the coder can use for writing the set of instructions. This language is then translated by automatic machinery into explicit instructions in the machine language. The language is described for the use of the Naval Research Laboratory computer and two methods for automatic translation are explained. One uses elementary IBM machinery in a self-checking process, the other uses the computer itself for performing the translation. The IBM procedure is described in detail in the first appendix and a fully annotated code for the computer translation is included in the second appendix.

A REPORT OF RESEARCH SUPPORTED
BY OFFICE OF NAVAL RESEARCH
CONTRACT NUMBER N-onr 494(03)
PROJECT NUMBER NR145-088

DECEMBER 1953

INTRODUCTION

Much of engineering prediction is done by the use of models, and in general two types can be distinguished -- physical and symbolic. Such a distinction, of course, is not precise and mixed types are sometimes used. A physical model "looks" like the real object. A scale model of an airplane for use in a wind tunnel is an example of a physical model. Certain characteristics of the real object and its environment are abstracted and rules of correspondence between the real and simulated systems clearly explicated. Inferences are drawn about the behavior of the real object in a real environment from the behavior of the model in the model environment. A parallel situation exists for the symbolic model. Here the basic correspondence lies between characteristics of the real object in the context of its real environment and a set of symbols.

In dealing with systems, some form of model must be used since the physical reality is generally too large to be examined under controlled conditions. The choice lies between a physical model and a symbolic model. The physical model would be a mock-up of the system. Mock-ups of any reasonable size system are very costly and quite special to the particular system. Symbolic models are inexpensive and the same computing machinery can be used to evaluate the model of any system. Although inexpensive and easy to control, the symbolic model suffers from its degree of abstraction from the physical reality. Attempts to improve the correspondence lead to cumbersome and intricate descriptions which of course turn into cumbersome and intricate series of instructions to a computer.

Generally, the correspondence between model and reality is in terms of measures. A symbol is used to represent a numerical measure of some property of the real situation. The rules describing interactions between the numbers associated with the symbols are presumed to hold for interactions between the measurements of properties of the real system. These rules are generally expressed in terms of some mathematical discipline or disciplines and carry the implication that he measures and their interactions satisfy the definitions and axioms of the formal discipline. Thus there are two sets of rules which the measures must satisfy: the special, corresponding to the particular model, and the general, corresponding to the mathematical discipline. A large advantage is, of course, that many broad relationships concerning the way in which the general rules are inter-related have been previously derived within the framework of deductive logic. If, for example, the motion of a physical system can be shown to be suitably described by the differential equation

$$\ddot{X} + F(X)\dot{X} + G(X) = 0,$$

then certain conditions on F and G can be stated for which a periodic free motion of the system is possible. This is inferred without explicitly calculating the motion of the system.

Computers have now made possible the manipulation of symbolic models of a size hitherto unfeasible. Before their advent, analytic investigation was the only means of dealing with models of any degree of complexity because of the expense and difficulty of numerical calculation. Indeed analysis has paid tremendous dividends in the cost and time involved in making engineering predictions. As a result, however, the entire process of model construction has been so colored by the requirements of analysis that often needless effort is expended in putting the description for a computer into proper form for analytic inference, even when there is no intention of attempting to make such inferences.

However, if a relatively direct description of the physical system in computer terms is employed several advantages accrue. First, there is a clear correspondence between the computer variables and their interactions and the physical variables and their interactions, which leads to a computer code that is easy to check for its consistency and its closeness to reality. Second, direct description tends to remove the strictures imposed by forcing the model to conform to special mathematical properties. For example, if the problem is to compute the behavior of a system best described by a non-linear differential equation, a strong tendency exists to linearize the equation and then solve it. Actually for the computer there is only slight difference in difficulty between a linear and non-linear equation. In fact, as far as describing the system is concerned, there is little point in even writing a differential equation. The physical laws relating the measures can be written directly in computer terms. This does not imply that analytic investigation is not fruitful, but that the computer description and the analytic description are in general separate problems and should be dealt with separately. The computer description becomes then a more or less direct analogue of the real system and can be examined in much the same manner as a physical analogue. In addition to specific numerical results concerning the measures of properties associated with the system, valuable qualitative examinations are possible, given adequate presentation of the results of the computer model.

CHAPTER I

PROBLEMS IN THE CONSTRUCTION OF DIGITAL COMPUTER ANALOGUES

The description of a computer analogue, that is, the simulation of complex systems by means of a set of instructions on a large scale digital computer, leads to problems that are not ordinarily encountered in routine calculations. Most of these are engendered by the comparatively large number of instructions and logical alternatives required by the complexity of the system.

The calculation process can be broken down into five steps. These are programming, coding, code checking, check run and calculation run. The steps are not discreet but are overlapping and interacting. Programming is the planning of the general character of the model to be used. This involves the choice of character-istics to be represented, the choice of coordinate systems, the responses to be recorded, the environments to be examined, and the computational sequences to be employed. There are often several equivalent mathematical statements of a given relationship. This is particularly true of iterative processes where several sequences converge to the same result. This selection of method falls in the task of programming. Coding is the writing of the explicit instructions for the computer, whereas code checking is the review of the code writing, preferably done by someone other than the original coder. The check run is done on the computer, and the computer results are compared with a precalculated problem. When this procedure is completed, the calculation sequence is run.

For large scale computers, he cost of computer time compares with the cost of twenty-to-forty men, depending on the particular machine. Therefore, on a cost basis between twenty and forty man-hours are well spent if they save an hour of computer run, but a higher ratio than this is manifestly inefficient. In addition, there is often a large time premium when a computer model is used to aid an engineering decision. The balance of these factors varies from problem to problem. To a large extent time spent on programming cannot be exchanged for computer time. This is not true, however, for coding and code checking. A typical example is the use of floating decimal sub-routines. Scaling a particular code to retain significance under computation might take one-hundred man-hours for a computing run of five minutes, not counting time spent in reading instructions in and results out.

Although floating decimal operations might take ten times as long in computer time, their use here results in a large net reduction in total cost of computation.

Simulation problems are characterized by high ratios of instructions to variables and constants, a high percentage of comparisons among the instructions, and a requirement for extreme flexibility in description. Let us assume that a coder writes directly in the machine language. That is, he uses the numerical form of the machine addresses and the numerical form of the machine operations. The description of a small system may easily involve, say, 2,000 single address instructions and perhaps 150 variables and constants. By analogy, his problem is similar to writing a 4,000 word essay in a completely new language with the penalty that a single mistake in spelling or grammar earns him a mark of zero. The language, moreover, changes not only from problem to problem but, in general, whenever the coder makes any sort of alteration in parts of the description already written. The code checker in turn must learn this language in order to detect any errors in the "spelling", i.e., the addresses used in the instructions, or in the "grammar", i.e., the logical structure of the code. The net result is that at least for systems problems, coding in this form is either impossible or at best a long and costly process.

To obviate this difficulty, the coder should be permitted to write his description in a more understandable language. If this language is chosen so that an explicit set of rules can be provided for translating the code from this form into machine form, the translation can be performed by machinery, eliminating a large source of clerical error. Thus, a symbolic coding method saves money in time spent on coding, time spent in code-checking, and, because of the reduction in probability of error, computer time in the check run. This last advantage might be reduced, if the computer itself is used to perform the translation.

If it is postulated that a general purpose digital computer will be used for performing the translation, almost complete freedom exists in the choice of language. The only major difficulty here is the specification of the rules of translation. For example, standard algebraic notation might be used, treating numbers as members of the field of real numbers. A direct translation from this formulation would fail in many instances due to the computer limitation to a finite set of allowed numbers. At this time insufficient knowledge exists concerning this limitation, and thus, rules of direct translation cannot be formulated.

As a compromise two simplifications in the computer language can be permitted. First, instead of using explicit machine addresses the coder can specify locations by means of five arbitrary alpha-numerical symbols. This redundancy permits cueing the given names to descriptions of the variables, so that the symbol groupings have mnemonic content. Second, the coder can refer to standard patterns of instructions by a single symbol, followed by the addresses of appropriate variables for entry into the pattern.

TABLE I

INSTRUCTION CODE FOR NAREC

Operation Code	Coding Symbol	Operational Instructions
10	TCL	Transfer the control to the left-hand order of the word at storage location s.
11	TCR	Transfer the control to the right-hand order of the word at storage location s.
12	CTL	If the number in the A register is greater than or equal to zero, transfer the control to the left-hand order of the word at storage location s; otherwise, proceed to the next order of the routine.
13	CTR	If the number in the A register is greater than or equal to zero, transfer the control to the right-hand order of the word at storage location s; otherwise, proceed to the next order of the routine.
20	REL	Replace digits 0 through 12 of the word at storage location s by digits 0 through 12 of the word in the A register.
21	RER	Replace digits 24 through 36 of the word at storage location s by digits 0 through 12 of the word in

21 RER the A register.

22 INL Increase the storage-location designation of the left-hand order of the word at storage location s by one.

23 INR Increase the storage-location designation of the right-hand order of the word at storage location s by one.

30 AL Shift the contents of the A and U registers (excepting sign digits) n places to the left. The overflows on the left of the A register are successively placed in the vacated digital positions of the U register, while the vacated positions of the A register are made zero.

31 Shift the contents of the A register (excepting the sign digit) n places to the right. The vacated digital positions on the left take the same condition as the sign digit, and the overflow at the right is dropped. The contents of the U register remain unchanged during this operation.

32 RD Read the words between the "start" and "stop" indications from the magnetic tape of the magnetic-tape

32	RD	reader to consecutive storage locations beginning with storage location s.
33	RC	Record the word at storage location s on the magnetic tape of the magnetic-tape recorder.
40	UA	Transfer the number in the U register to the A register.
41	AU	Transfer the number in the A register to the U register.
42	FA	Transfer the word in the A register to storage location s.
43	FU	Transfer the number in the U register to storage location s.
50	TA	Transfer the word at storage location s to the A register.
51	MA	Transfer the negative of the number at storage location s to the A register.
52	TAB	Transfer the absolute value of the number at storage location s to the A register.

53 MAB Transfer the negative of the absolute value of the number at storage location s to the A register.

54 AD Add the number at storage location s to the word in the A register and place the result in the A register.

55 SU Add negatively the number at storage location s to the word in the A register and place the result in the A register.

56 ADB Add the absolute value of the number at storage location s to the number in the A register and place the result in the A register.

57 SUB Add negatively the absolute value of the number at storage location s to the number in the A register and place the result in the A register.

60 MU Multiply the number at storage location s by the number in the A register and form the high-order product in the A register and the low-order product in the U register.

61 MR Multiply the number at storage location s by the number in the A register and form the rounded-off high-order product in the A register.

70 DA Divide the number in the A register by the number at storage location *s* and form the rounded-off quotient in the A register.

71 X U Reset the U register to zero.

80 X A Reset the A register to zero.

81 BT Transfer the n words at storage locations s_1 to (s_1 plus (n-1)) to storage location s_2 to (s_2 plus (n-1)) respectively.

82 ST Stop machine operation.

CHAPTER II

CODE LANGUAGE AND TRANSLATION PROCESSES

In the appendices two different translation methods are described; one uses elementary IBM machinery, and the other uses a large general purpose digital computer (the Naval Research Laboratory Computer - NAREC). Both translation processes employ approximately the same basic coding language. This language has two principal features, symbolic representation of addresses, and symbolic reference to pattern subroutines. Pattern subroutines differ from transfer-of-control subroutines in that they are incorporated in the main sequence of instructions rather than occupy a set of standard locations.

If, for example, there is a subroutine for finding the dot product of three dimensional vectors, the pattern form would appear as follows (Refer to TABLE I for the meaning of the operational symbols):

```
TA    X1
MR    Y1
FA    T1
TA    X2
MR    Y2
```

AD T1

FA T1

TA X3

MR Y3

AD T1

This pattern would be included in the main sequence of instructions whenever the subroutine is used, differing only in the addresses of X1, Y1, X2, Y2, X3, Y3. As a transfer-of-control subroutine the instruction set would appear:

TA X1

FA S1

TA X2

FA S2

TA X3

FA S3

TA Y1

FA S4

TA Y2

FA S5

```
        TA      YS
        FA      S6
        TA      DPX1L   (Note:  DPX1L denotes the address where the
                         numerical value of DPX1 (an address) is stored.
                         DPX1 next instruction in main sequence.
        RE      DPX
        TC      DPENT
DPENT   TA      S1
        MR      S4
        FA      T1
        TA      S2
        MR      S5
        AD      T1
        FA      T1
        TA      S3
        MR      S6
        AD      T1
DPX     TC
```

Clearly in this case the pattern subroutine is more economical, since the set-up instructions are longer than the routine itself. It is to be noted that when instructions are acted on by other instructions, they must be tagged with the reference name.

For a further example suppose the action of a ship heading for a mooring buoy is being simulated and the information desired is the sine and cosine of the course angle to the buoy. Also assume that routine, called *SICO*, has been established that gives these for any two points. This could have two different translations, depending on whether a pattern subroutine or a transfer-of-control subroutine is used. The subroutine would appear generally as follows:

TA	X1	(1)
SU	X2	(2)
FA	X	
MR	X	
FA	T1	
TA	Y1	(3)
SU	Y2	(4)
FA	Y	

```
        MR      Y
        AD      T1
        FA      N
        TA      ONE
        FA      T1
RPT     TA      N
        DI      T1
        AD      TOL
        SU      T1
        CT      EX
        TA      R
        FA      T1
        TC      RPT
EX      TA      X
        DI      R
        FA      SIN         (5)
        TA      Y
        DI      R
        FA      COS         (6)
        AD      T1
```

AR 1

FA R

For the pattern subroutine the coder would assume the following dictionary:

(1) SHIP X - X position of ship

(2) SHIP Y - Y position of ship

(3) BUOY X - X position of buoy

(4) BUOY Y - Y position

(5) SHIP S - Sine of ship's course angle

(6) SHIP C - Cosine of ship's course angle

TOL - admissible error in calculation of square root

Thus the coder would write SICO BUOY X, BUOY Y, SHIP X, SHIP Y, SHIP S, SHIP C.

The addresses following SICO would replace the addresses at the corresponding numbered positions in the pattern. To use a transfer-of-control, the correct locations in the set-up instructions, considering the set-up instructions as a pattern, are replaced. This would appear SICO BUOY X, BUOY Y, SHIP X, SHIP Y, SHIP S, SHIP C, NINTL.

NINT is the next instruction in the main routine.

NINTL is the location of the number NINT

When using a transfer of control subroutine, the exit to the main routine must be specified. The set-up pattern would appear.

```
     FA          (1)
     FA    X1
     TA          (2)
     FA    Y1
     TA          (3)
     FA    X2
     TA          (4)
     FA    Y2
     TA    SICIL
     RE    SICOX
     TC    SICON
SICI TA    SIN
     FA          (5)
     TA    COS
     FA          (6)
     TC          (7)
```

Additional dictionary for transfer-of-control subroutine is:

SICOX - last instructionin subroutine

SICON - address of first instruction of subroutine; (7) is the address of the address of the next instruction; in this problem NINTL

This pattern places the arguments in the correct locations for use of the standard set of instructions that comprise the subroutine. In addition, the pattern provides for proper return of control to the main sequence of instructions. In other words the set-up instructions for a transfer-of-control subroutine are treated as a pattern and translated as such.

Translation by IBM. The translation by means of IBM machinery can be divided into two main steps: first, translation from multiple address symbolic form into single address symbolic form, and second, translation from single address symbolic form into specific machine instructions.

In the first step a deck of cards containing the instructions in multiple-address form is sorted by subroutine designation and the number of occurrences of each subroutine is counted. Standard sets of cards are prepared for each subroutine. These contain the complete coded pattern of the subroutine with addresses that are the same for each use of the subroutine in symbolic form and blanks left for the positions occupied by the arguments. In addition, the subroutine decks contain punches that control the read feed of the reproducer and the timing of selectors that punch the arguments into the appropriate positions in the subroutine deck. The instruction deck is placed in the read feed; the subroutine decks are placed in the punch feed. As each instruction card passes under the reading brushes, all of the information is punched into the first card of the subroutine deck. This information is gang-punched back into successive cards of the subroutine deck while the read feed is held. Selectors then transfer the symbols from the gang-punched columns into appropriate positions on the subroutine cards. When a subroutine is completed, the next instruction card is read. This process results in a deck containing a set of instructions in terms of the elementary operations of the computer, but with ~~symbols~~ ADDRESSES still in symbolic form.

The next step is the translation of single-address symbolic form into explicit machine language. First, the deck of single-address instructions are numbered sequentially, beginning with the first address where instructions are stored. Next, those instructions that are named are extracted from the deck. These are used to form an auxiliary set of dictionary cards, containing the symbolic form of the name and the machine address assignment. These are lumped with the dictionary cards, bearing the symbolic form of variables, parameters, and pseudo-variables to form the complete dictionary deck. The dictionary deck is placed in a sorter followed by the instruction deck and sorted on the symbolic form of the name. The machine address is then gang-punched into the instruction deck. The symbolic form of the operation is similarly translated to form a complete set of instructions in machine terminology. This is now printed for transfer to tape or other means of machine input. The intermediate checking processes have not been described. Suffice to say that all operations are machine checked to arrive at a very low probability of error.

Translation by NAREC. In the translation process for the NAREC, similar procedures are followed with allowance for the capabilities and limitations of the computer. The process is predicated on the use of five-bit alpha-numerical representation of input symbols. The dictionary is read into the machine first. This consists of five five-bit symbols for each symbolic form, a total of twenty-five bits occupying the last portion of a machine word. As the dictionary is read in, an address is assigned to each symbolic form. In addition, the number of forms beginning with each of the thirty-two possible initial symbols is counted. The dictionary is now sorted into blocks corresponding to the initial symbol. This is done so that, when translating, the machine need only search on the average over 1/32 of the dictionary. Now the instructions are read in as operation reference blocks. In other words, the size of the read-in will depend on the number of addresses corresponding to the particular operation. If the instruction read-in has a name associated with it, the name and the assigned address are placed in the appropriate location in the dictionary. Since a named instruction may be referenced in some other instruction preceding it, a special section of the memory is set aside for storing such references. When the named instruction is read in, this section is checked to see if reference has been made to it and the appropriate address is ~~submitted~~ SUBSTITUTED. If the operation refers to a pattern subroutine, the arguments are inserted into the correct locations in the pattern, which is stored in the memory in symbolic form. This pattern is now moved as a block to the translation position and translated as a set of single-address instructions. Various special provisions are made for block transfer instructions and those operations for which the numerical code varies for right or left reference. In addition, various subsidiary checks are made to test memory reliability, completeness of dictionary, and completeness in pattern subroutine reference.

Detailed formulations of these processes are included in the appendices. The code for the NAREC translation is written in proper form for single address translation by IBM machinery and may be examined as an example of symbolic coding. The NAREC code has not as yet been machine checked and therefore is subject to minor revision and correction.

APPENDIX I

IBM TRANSLATION PROCESS

K. J. Isaac, H. Wenzel, S. Anderson

The IBM process translates the multiple-address symbolic code into single-address symbolic code which is then translated into explicit machine code. The code may contain individual single-address instructions, but these are separated, so that they do not run through the multiple-address process. The multiple-address process is based on the previous construction of standard decks for each subroutine used. These consist of a deck of cards where each card stands for a single-address instruction in the pattern. The cards are punched, using the operations in symbolic form (columns 10-12). If the instruction refers to a symbolic address that is the same for all repetitions of the subroutine, the card is punched with this address (columns 13-17). If the address is that of one of the arguments in the multiple-address form, the columns are left blank. Each card of the pattern deck contains a set of X punches, determing which of the nine addresses to substitute in the symbolic address position (columns 24, 30, 36, 42). See figure I. In addition, the card carries a local serial number indicating its position within the subroutine (columns 48, 54). The first card of the subroutine pattern has an X in column 18. Column assignments are shown in figure II.

The steps in the translation process now follow:

1. The code is punched into a deck of cards numbered serially in decimal form. If an instruction is named, the name is punched in columns 1-5. Either a single-address instruction or the name of a multiple-address instruction is punched in the operation columns.

2. The instruction deck is sorted on the operation columns. The single-address instructions are set aside and the number of occurrences of each different multiple-address instruction is noted.

3. The multiple-address instructions are placed in the reproduce feed of the 519. A number of standard subroutine decks equal to the number of occurrences of the multiple-address instructions are placed in the punch feed in the sorted order. The cards are then punched using Board No. 1. The cards in the punch feed are now checked in the reproduce feed with Board No. 2. This deck is then lumped with the single-address instructions to form a complete single-address instruction deck.

FIGURE I

Column Numbers	24	30	36	42
1st				
2nd			X	
3rd	X			
4th	X		X	
5th	X	X	X	
6th	X	X		
7th		X		
8th		X		X
9th		X	X	
Blank or filled		X	X	X

FIGURE II

Column Assignments

Subroutine card and/or single address card

1-5	Name of instruction
6-9	Main code serial no. (decimal)
10-12	Symbolic operation
13-17	Symbolic single address
18	First card X, subroutine deck
19-23	First multiple address
24	Substitution control X
25-29	Second multiple address
30	Substitution control X
31-35	Third multiple address
36	Substitution control X.
37-41	Fourth multiple address
42	Substitution control X
43-47	Fifth multiple address
48	Subroutine serial
49-53	Sixth multiple address
54	Subroutine serial
55-59	Seventh multiple address
60-64	Eighth multiple address
65-68	Machine location of instruction (three hexi-decimal digits and X for left-right indication)
69-70	Operation in machine code
71-73	Address in machine code
74	Left-right X for named instructions
75	Dictionary x
76-80	Ninth multiple address

4. The single address instruction deck is sorted on the decimal serial number to return it to proper sequence.

5. The instruction deck is sequence punched from a double hexidecimal numbered deck with alternate X's to indicate left and right, i. e., the deck is numbered as follows:

000

000X

001

001X

...

...

...

009

009X

00A

00AX

etc

This is the assignment of machine locations to instructions.

6. The instruction deck is sorted on column 1 and all named instructions are pulled out.

7. The named instruction cards are reproduce punched into a dictionary deck, the name going into columns 13-17, the hexi-address going into columns 71-73, and the left-right X indicator into column 74. An X is punched in column 75 of the cards in the dictionary deck.

8. A number dictionary deck is prepared containing the symbolic name of each address, which is included in the code, and yet is not the address of an instruction. This deck also contains pseudo-addresses, e.g., the number of right and left shifts. These names are punched in columns 13-17. The machine location corresponding to each name is punched in columns 71-73. In the case of a pseudo-address the hexi-equivalent of the desired number is punched in these columns. The assignment of machine locations may be done individually or sequence punched with the hexi-deck. Each card contains an X punch in column 75. These cards are lumped with the deck from step 7 to form the complete dictionary deck.

9. The dictionary deck is put in the sorter followed by the instruction deck, and the assemblage sorted on columns 13-17. Each instruction card will now follow the dictionary card containing the symbolic name referred to by the instruction.

10. This total deck is now master-card gang punched and checked, inserting the hexi-decimal address from the dictionary card into the instruction cards.

11. The dictionary cards are now removed on a sort and the deck is sorted in with an operations dictionary deck. In this process, left-right variable operations (such as IN, TC, etc) have two cards, the right reference instruction carrying an X in column 74. The operations deck is put in the sorter, followed by the instruction deck, and sorted first on column 74, then on columns 10-12. The assemblage is now master card gang-punched on columns 69-70, substituting the machine form of the operation.

12. The code is now complete and may be printed for transfer to magnetic or punch tape input.

Wiring diagrams for Boards Nos. 1 and 2 are given in figures III and IV. However, wiring diagrams for the reproducer for steps 4 and on are not included, since the processes are simpler and the requisite wiring for punching and checking is straightforward. Concerning Board No. 1, the 519 reads one card in the read feed and transfers all of its information to the first card of the subroutine deck. This is controlled by an X punch in column 18 of the first card of the subroutine deck actuating the 80-column selector. The input to the selector system which governs the substitution of multiple addresses into single address positions is from the punch direct hubs, so that substitution can be made in the first card of the subroutine. Although the read feed is held by the second card in the subroutine deck, the rollers after the first reading brushes still operate. This permits comparing the reproduce punching into the first card of the subroutine deck. Although comparing is suppressed after the first card is through the gang punch and interpreting brushes, a back circuit exists through the flux linkage of the comparing magnets via the comparing and transcribing brushes, which are now connected in parallel by the copper roller. This results in erratic gang-punching. To prevent this, the diodes are inserted in the circuit as shown, saving one pass through the reproducer with a special board for comparing the reproduce punching.

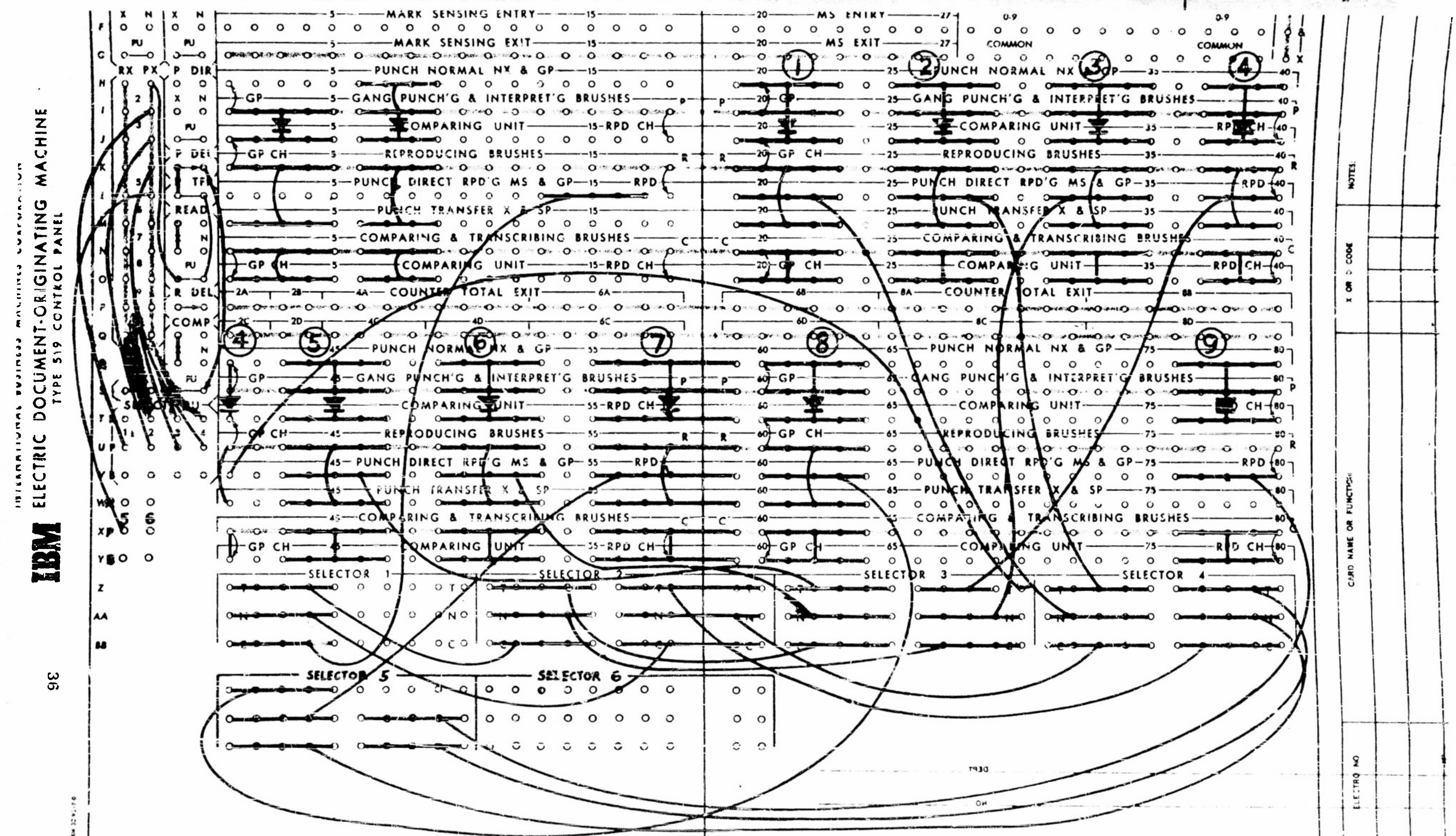


FIGURE III BOARD No. 1

INTERNATIONAL BUSINESS MACHINES CORPORATION

IBM ELECTRIC DOCUMENT-ORIGINATING MACHINE

TYPE 519 CONTROL PANEL

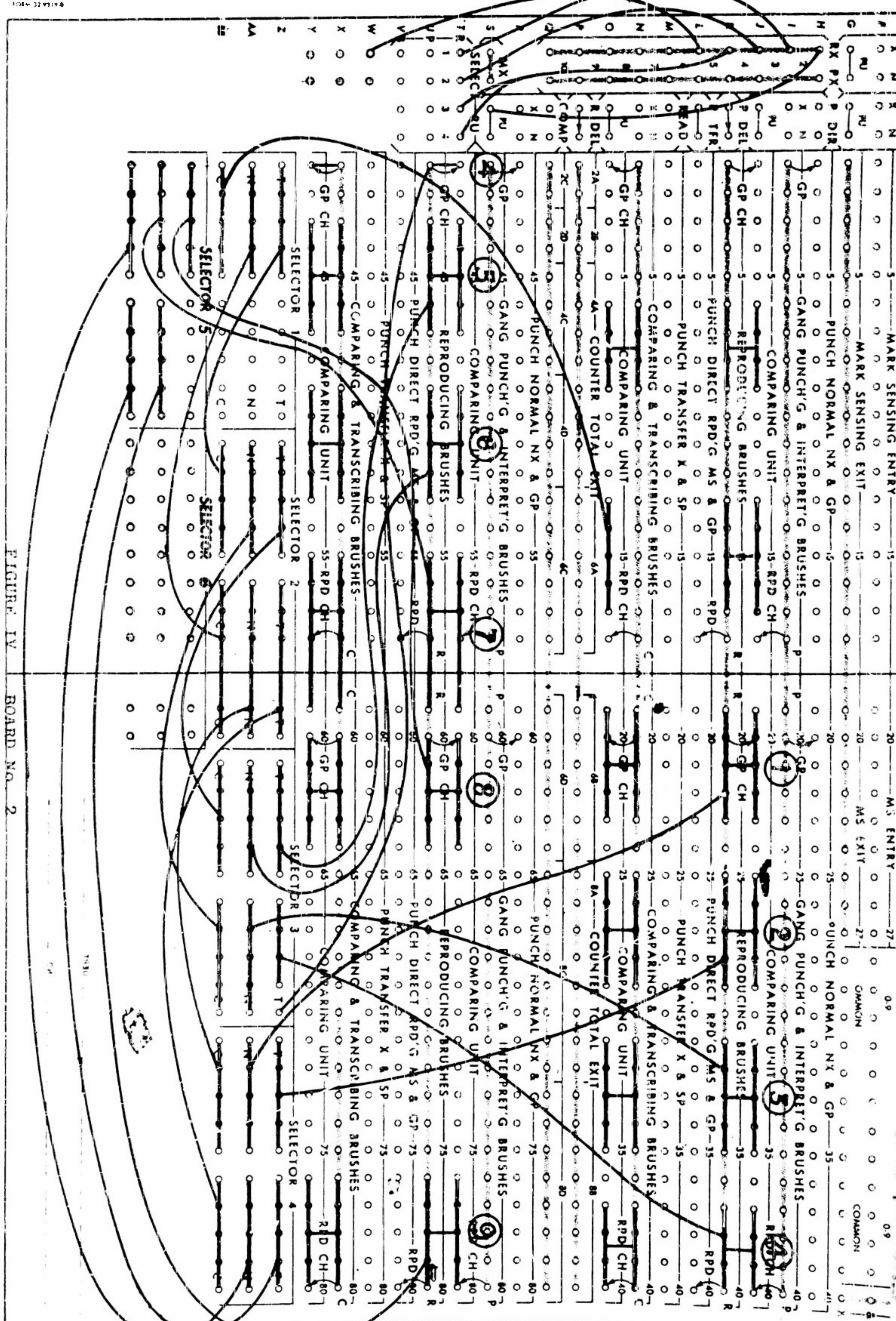


FIGURE IV BOARD No. 2

ELECTRO NO	CARD NAME OR FUNCTION	X OR D CODE	NOTES:

APPENDIX 2

NAREC TRANSLATION PROCESS

E. J. Isaac, F. Spatz

The following is a brief description of the code for the NAREC to perform the translation from symbolic to machine code. This code is itself written to be translated by the IBM method.

The first part of the code is concerned with the dictionary consisting of the variables, constants, and parameters used in the main routine. In symbolic form these are represented by five letters, numbers or a combination of both. Each letter or number is one of thirty-two possibilities and can be represented in a binary system by five bits. A complete address in symbolic form will, therefore, occupy twenty-five bits. The variables are read into the machine in symbolic form occupying the last twenty-five bits of the storage words. They are preceded by a word containing the total number of variables, constants and parameters. Addresses are assigned to the variables in order, beginning with the first. Now the dictionary words have the symbolic form in the last twenty-five bits and the machine code for address in left-hand address position. These words are then transferred to to appropriate places in the dictionary, as described below.

In addition, a blank word is stored in the last block of the dictionary. This will be useful for translating blank and no-address instructions which call for zeros in the address position.

In order to shorten the process of searching through the dictionary, the variables are grouped in the dictionary in thirty-two blocks, all the variables in the same block having the same left hand letter in symbolic form. The variables whose symbolic form starts with 00000 are stored at highest addresses of the dictionary.

Whenever a search of the dictionary is made, one needs only to deal with the appropriate block of the dictionary as determined by the left-hand letter of the symbolic form of the variable. The addresses of the first dictionary word in each block are stored in the thirty-two locations beginning with STORE.

Operations are stored in the machine in the block beginning with OPSTR. The symbolic form of the operation will appear in the last nineteen bits on the right. The machine code for the variables will appear in bits 18-25. Four extra operation words which deal with the variable operations IN, RE, TC, and CT are inserted at the beginning of the list of operations. This makes it possible for the machine itself to determine whether these operations should refer to the right or left-hand order of the storage word, when an address of the instruction refers to another instruction. The operation words following these first four will contain INR, INL, RER, REL, etc. A negative word is stored in the location immediately following these operations. This will indicate when an operation appears which does not correspond to any which are listed in the machine.

To shorten the number of words that have to be read into the machine each time, a series of instructions which appear more than once in the main routine is designated as a pattern subroutine and a name is assigned to each. The instructions comprising a pattern subroutine are stored in the machine in symbolic form. The instructions for each subroutine are preceded by a set of prewords. The first preword contains the total number of words relating to the subroutine in bits 1 through 8, total number of addresses to be inserted in subroutine in bits 9 through 13, number of prewords in bits 14 through 16, number of named instructions in subroutine in bits 17 through 19, and the symbolic name of subroutine in bits 20 through 44. The remaining prewords contain successive differences between the addresses of the instructions where the variables to be used in the subroutine have to be inserted. Each difference occupies five bits, so that each preword, excepting the first, contains eight differences. A word containing a negative number is read in after all the subroutine words. This negative word will indicate when the name of a subroutine appears in BANK for which no set of subroutine words was put into the machine in the block beginning with SUPR.

Instructions in the main routine are read in order into BANK, one at a time. Ordinary instructions (i.e., all instructions except those calling for a shift or a block transfer operation) are read in with address occupying last twenty-five bits of word on the right and operation in symbolic form on the left. An instruction which refers to no address will be read in with zeros in the last twenty-five bits. An instruction which calls for a shift is read in as a negative word. The positive form of the word will contain the operation in symbolic form in bits 5 through 19 and the number of bits to be shifted in the right address position. The first four bits are zero. An instruction which calls for a block transfer must occupy two storage words in symbolic form. The first word which must be stored negatively will contain the symbolic form of the address of first variable to be transferred in last twenty-five bits and block transfer operation on the left. The first bit will contain a 1 to distinguish it from any other negative instruction. The second word will contain the symbolic form of the address of storage location to which the first variable is to be transferred in the last twenty-five bits and the number of variables to be transferred in binary form in bits 12 through 19.

When a pattern subroutine occurs in the code a series of words are read into BANK. The first word contains the name of the subroutine on the right in the last twenty-five bits and the subroutine indicator on the left. This word is followed by a series of words containing symbolic form of variables that have to be inserted into instructions.

If the word in BANK is positive, it is tested first to see if it is the word of a named instruction. If it is a named instruction, it is tested to determine to which of the thirty-two blocks of the dictionary (originally containing variables only) it belongs. All the blocks occupying lower storage locations are moved up one to make room for the named instruction. The name indicator of the word in BANK is replaced by the address of the permanent storage location assigned to this instruction. The word in BANK is then transferred to the appropriate place in the dictionary. It now occupies the first storage location of the block of variables. The appropriate addresses contained in STORE are decreased to correspond to the new storage locations for the variables resulting from the block transfer. Instructions in HOLD are checked to see if the address in any of these instructions corresponds to the address of this named instruction. Every instruction in HOLD must be checked. If there are any whose address does correspond, these instructions are translated into machine code and put in the appropriate storage locations. Then the instruction in BANK2 following the named instruction word is translated.

If the word in BANK is not a named instruction, then it is tested to see if it is the name of a pattern subroutine. If it is the name of a pattern subroutine, then the variables to be used in the subrutine are put into the appropriate instruction words of the subroutine. Then all the instructions in the subroutine are transferred to BANK and each one is translated as an instruction.

If the instruction in BANK does not involve block transfer operation or a shift operation, it is an ordinary instruction with symbolic address in the last twenty-five bits on the right and symbolic operation on the left. A search of the dictionary is made to find the address of the variable corresponding to the symbolic address contained in the instruction. If no word can be found in the dictionary which corresponds to this address, then the address is that of a named instruction which has not yet been brought into the dictionary. The operation is translated and then stored together with the symbolic form of the address in HOLD until a new word is added to the dictionary. If the word is in the dictionary, then the address of the permanent storage location is taken out and put in a temporary storage location. A similar procedure is followed for the operation. A search is made of the list of operations. If no operation can be found which corresponds to the operation in the instruction, the machine will indicate an error. If it is found, the machine code for the operation is added to the word containing machine code for the address. We now have the instruction in the desired coded form, occupying the right-hand order of the storage word. If this is the first instruction that has been translated, or if it is an odd-numbered instruction, then the instruction is shifted to the left-hand order and stored in FORM. If it is an even-numbered instruction, it is added to the word in FORM, giving a word with an instruction in both the left-hand and the right-hand order of the word. The word is then transferred to the appropriate storage location. If the word in BANK is stored negatively, it involves either a shift or a block transfer operation. If it involves a shift operation, the number of bits to be shifted is stored in right address position of a temporary location. The operation is translated as that of an ordinary instruction and the instruction is stored in the appropriate location.

If the word in BANK is negative and involves a block transfer instruction, it must be determined whether the previous instruction that was translated occupied the right or left-hand order of storage word. If it was a left-hand instruction, zeros must be inserted in the right-hand order of word, since the block transfer instruction itself must occupy an entire word. The first word in BANK will be translated as an ordinary instruction and stored in a temporary storage location. Only the address of the second word is translated since the rest of the word is a number in binary form. Thus, the whole block transfer instruction is set up and stored in appropriate storage location.

DICTIONARY

BANK	First address of general free group of storage locations.
LVAI	Contains address of last storage location for variables plus one in the left address position.
BANK1	Address of BANK plus 1
STORE, STO31	Address of thirty-two storage locations used in routine. Ultimately these contain address in dictionary of first variable in each alphanumerical group.
STORL	Contains address STORE.
RA 32	Thirty-two in right address position
TALI	General purpose tally.
DICTL	The next-to-last address of the permanent set of storage locations to be assigned to the dictionary
DICLI	The last address of the permanent set of storage locations to be assigned to the dictionary
T1	Address of a temporary storage location
TOTAL	Contains the negative of the number of variables in dictionary
ADICT	Address of storage location used to indicate that dictionary has not been added up previously when it contains a zero, to indicate that dictionary has been added up previously when it contains a negative quantity
SUM	Contains dictionary sum
2M44	Contains the binary equivalent of 2^{-44}
ASSGN	Contains successively address to be assigned to each variable in dictionary
T3	Address of a temporary storage location
T4	Address of a temporary storage location

ILADD	Contains a one in the twelfth bit and zeros elsewhere
SUBR	Address of first storage location where pattern subroutines are stored
IINST	Contains address of storage location to be assigned to first instruction of main routine
NUMB	Contains successively address of storage location to be assigned to each instruction of main routine
STIN	Contains address of storage location where first translated instruction is to be stored
SWICH	Address of storage location used to indicate a left order instruction when it contains a negative word
BANKL	Contains the address BANK
WORK	Contains successively the address of word in BANK, etc., which is being translated
OPIND	Address of storage location used to indicate that the translation of an address should be followed by a translation of an operation when word in OPIND is negative. A non-negative word in OPIND indicates that address first translated is second one of a block transfer instruction and therefore no operation should be translated.
BTIND	Address of storage location used to indicate a block transfer instruction when it contains a negative number an ordinary instruction when it contains zero.
IRADD	Contains a one in thirty-sixth bit and zeros elsewhere
TALI3	A general-purpose tally containing negative of the number of ordinary instruction words in BANK yet to be translated

TEMP	Address of a temporary storage location
NIND	Contains the number used to indicate a named instruction in right address position
[illegible]	Contains a one in twelfth bit and in thirty-sixth bit and zeros elsewhere
LA32	Contains binary equivalent of thirty-two in left-address position
HOLDF	Contains address of first storage location where instructions are held until new words are put in dictionary
NHLD	Contains the number of instructions being held for new dictionary words
OPTEM	Contains successively symbolic form of operation of each instruction
OPSTA	Contains address of first storage location where operations are stored
HADDF	Contains address of first storage location where the instructions which ~~are being held~~ for new dictionary words are to be stored when they are translated
2M1	Contains a one in first bit and zeros elsewhere
SUBIN	Contains the number used to indicate a pattern subroutine in right-address position
SIMBL	Contains the address SUBR
WORKI	Contains successively the addresses of variables in BANK which are inserted into instructions of pattern subroutine
TALI2	This is a general purpose tally used when inserting variables in the appropriate instructions of a pattern subroutine. It contains the negative of the number of differences in preword that are yet to be considered.

T3	Address of a temporary storage location
T6	Address of a temporary storage location
T4	Address of a temporary storage location
TO	Address of a temporary storage location
T7	Address of a temporary storage location
29M41	Contains binary equivalent of $2^{-9} - 2^{-41}$
5RADD	Contains binary equivalent of five in right-address position
LORR	Used to indicate that address of instruction refers to a left order instruction when it contains a negative word, refers to a right order instruction when it contains a zero
FOUR	Contains binary equivalent of four in left-address position
FORM	Address of storage location where left order instructions which have been translated are stored temporarily
STBT	Address of storage location where left order of block transfer instruction is stored temporarily
MEMER	Contains a number indicating error in dictionary memory
SRER	Contains a number indicating an error in pattern sub-routines
OPSER	Contains a number indicating an error in storage of operations
DICL1	Address of storage location succeeding DICTL
LRDEP	Used to indicate, when an instruction is taken out of HOLD, whether or not it deals with one of the variable operations
HDEP	Used to indicate when the instruction is to be stored in HOLD after operation has been translated

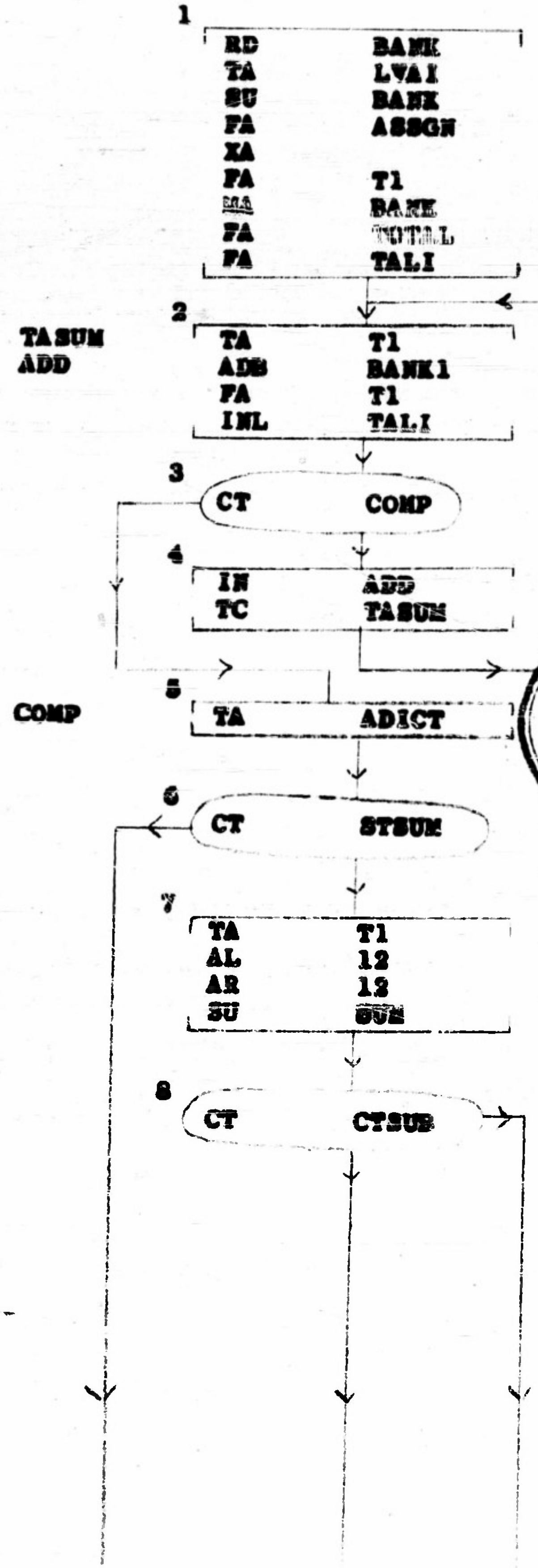
INSTRUCTION CODE FOR THE NAREC

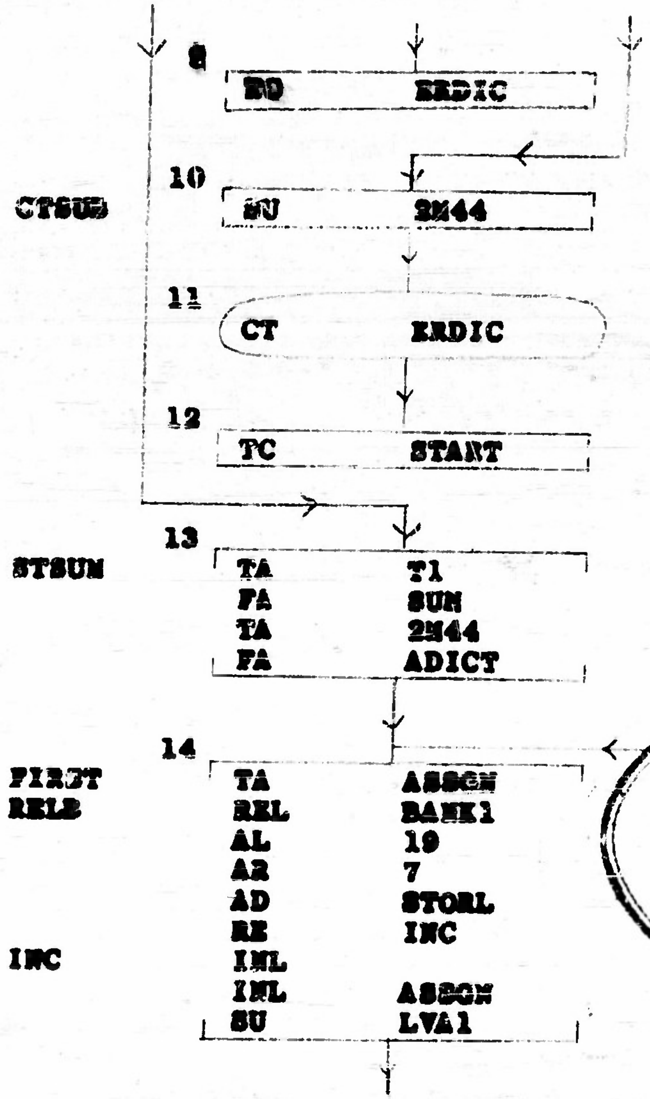
Operation Code	Coding Symbol	OPERATIONAL INSTRUCTIONS
10	TCL	Transfer the control to the left-hand order of the word at storage location m.
11	TCR	Transfer the control to the right-hand order of the word at storage location m.
12	CTL	If the number in the A register is greater than or equal to zero, transfer the control to the left-hand order of the word at storage location m; otherwise, proceed to the next order of the routine.
13	CTR	If the number in the A register is greater than or equal to zero, transfer the control to the right-hand order of the word at storage location m; otherwise proceed to the next order of the routine.
20	REL	Replace digits 0 through 12 of the word at storage location m by digits 0 through 12 of the word in the A register.
21	RER	Replace digits 24 through 36 of the word at storage location m by digits 0 through 12 of the word in the A register.
22	INL	Increase the storage-location designation of the left-hand order of the word at storage location m by one.
23	INR	Increase the storage-location designation of the right-hand order of the word at storage location m by one.
30	AL	Shift the contents of the A and U registers (excepting sign digits) n places to the left.

		The overflows on the left of the A register are successively placed in the vacated digital positions of the U register, while the vacated positions of the A register are made zero.
31	AR	Shift the contents of the A register (excluding the sign digit) n places to the right. The vacated n digital positions on the left take the same condition as the sign digit, and the overflow at the right is dropped. The contents of the U register remain unchanged during this operation.
32	RD	Read the words between the "start" and "stop" indications from the magnetic tape of the magnetic-tape reader to consecutive storage locations beginning with storage location s.
33	RC	Record the word at storage location s on the magnetic tape of the magnetic-tape recorder.
40	U A	Transfer the number in the U register to the A register.
41	A U	Transfer the number in the A register to the U register.
42	F A	Transfer the word in the A register to storage location s.
43	F U	Transfer the number in the U register to storage location s.
60	T A	Transfer the word at storage location s to the A register.

51	M A	Transfer the negative of the number at storage location s to the A register.
52	TAB	Transfer the absolute value of the number at storage location s to the A register
53	MAB	Transfer the negative of the absolute value of the number at storage location s to the A register
54	A D	Add the number at storage location s to the word in the A register and place the result in the A register.
55	S U	Add negatively the number at storage location s to the word in the A register and place the result in the A register
56	ADB	Add the absolute value of the number at storage location s to the number in the A register and place the result in the A register
57	SUB	Add negatively the absolute value of the number at storage location s to the number in the A register and place the result in the A register
60	M U	Multiply the number at storage location s by the number in the A register and form the high-order product in the A register and the low-order product in the U register
61	M R	Multiply the number at storage location s by the number in the A register and form the rounded-off high-order product in the A register
70	D A	Divide the number in the A register by the number at storage location s and form the rounded-off quotient in the A register
71	X U	Reset the U register to zero

80	X A	Reset the A register to zero
81	B T	Transfer the n words at storage locations s_1 to (s_1 plus (n-1)) to storage location s_2 to (s_2 plus (n-1)) respectively
82	STOP	Stop machine operation


1
RD BANK
TA LVAI
SU BANK
FA ASSGN
XA
FA T1
MA BANK
FA TOTAL
FA TALI
2
TASUM
ADD
TA T1
ADB BANK1
FA T1
INL TALI
3
CT COMP
4
IN ADD
TC TASUM
5
COMP
TA ADICT
6
CT STSUM
7
TA T1
AL 12
AR 12
SU SUM
8
CT CTSUB


9
RD ERDIC
10
CTSUB
SU 2M44
11
CT ERDIC
12
TC START
13
STSUM
TA T1
FA SUM
TA 2M44
FA ADICT
14
FIRST
RELB
TA ASSGN
REL BANK1
AL 19
AR 7
AD STORL
RE INC
INC
INL
INL ASSGN
SU LVA1

```
15
          CT        SORT

16
          IN        FIRST
          IN        FIELD
          TC        FIRST

17
SORT      MA        RA32
          FA        TALI
          TA        DICTL
          FA        T4
          XA
          FA        T2

18
CUM       TA        T2
          FA        T1
ADD 1     TA        STORE
          FA        T2
          TA        T4
ADD 2     SU        T1
          FA        STORE
          FA        T4
          IN        TALI

19
          CT        TRAN

20
          IN        ADD1
          IN        ADD2
          TC        CUM

21
TRAN      MA        BANK
          FA        TALI
```

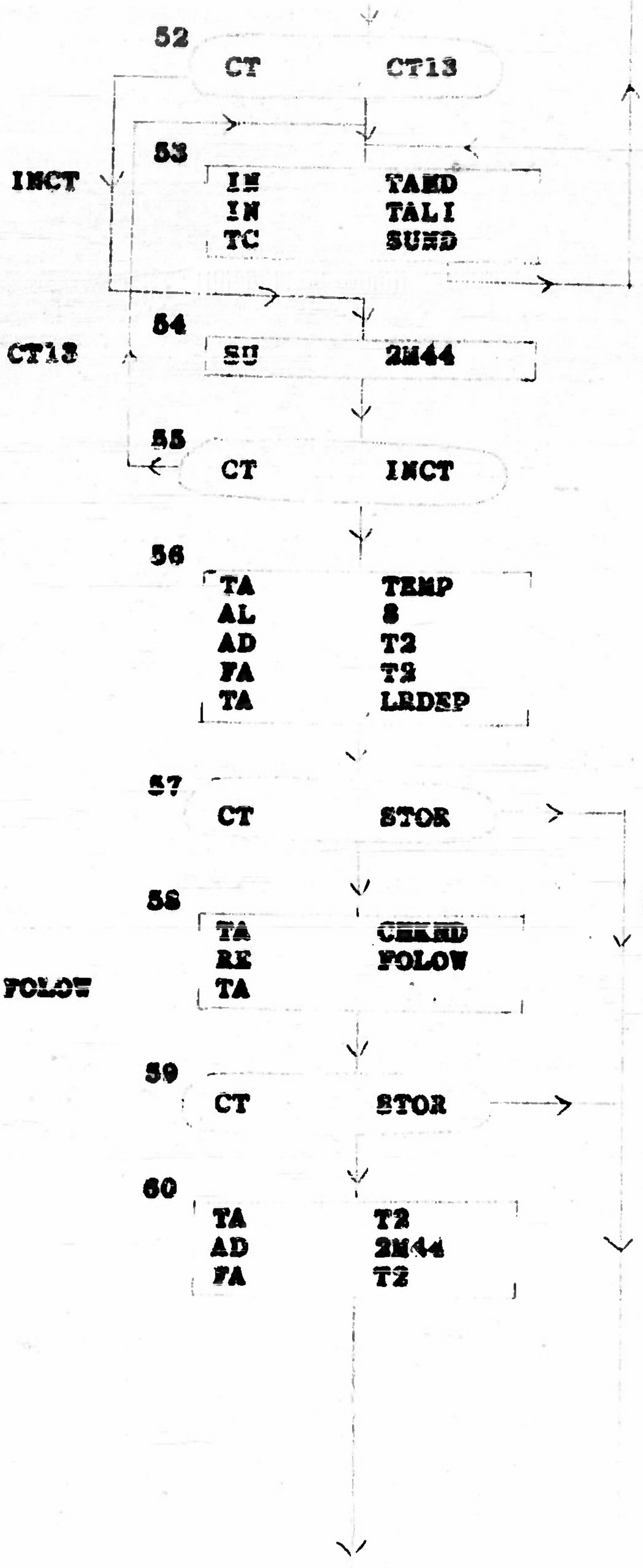
```
        22
TRANS        TA      BANK1
             FA      T1
             AL      12
             AD      STORL
             RE      GET
             RE      [illegible]
GET          TA
             REL     LOC
             TA      T1
LOC          FA
             TA      LOC
             SU      ILADD
SEND         REL
             INL     TAL1

        23
             CT      AA

        24
             IN      TRANS
             TC      TRANS

        25
AA           XA
             FA      DICL1

             TA      LOC

             REL     DICTR
             SU      ILADD
             RER     DICTR
             RD      SUBR
             TA      IINST
             FA      NUMB
             TA      STIN
             RE      PLACE
             MA      2M44
             FA      SWICH
```

```
        26
RDST         RD     BANK
             TA     BANKL
             FA     WORK
             MA     2E44
             FA     OPIND
             XA
             FA     BTIND
             MA     IRADD
             FA     TAL13
             TA     STO31
             RE     ADD
             TC     TAAD

        27
START        TA     WORK
             RE     TABK
TABK         TA

        28
             CT     CLEAR

        29
             TC     BEGIN

        30
CLEAR        XU
             AL     19
             FA     TEMP
             UA
             SU     NIND

        31
             CT     CT2

        32
             TC     OPER
```

```
              33
CT2                SU         2M<.4

              34
                   CT         OPER

              35
                   TA         TEMP
                   AR         39
                   AL         32
                   FA         T2
                   AD         STCRL
                   RE         CHNG1
                   RE         CHNG2
                   RE         LAST
                   RE         HIGH
HIGH               TA
                   SU         STOS1
                   AR         32
                   AD         DICTR
                   SU         IRLAD
                   FA         DICTR
DICTR              BT

                   TA         T2
                   SU         LA32
                   FA         TALI

              36
CHNG1              TA
                   SU         ILADD
CHNG2              FA
                   INL        TALI

              37
                   CT         LAST
```

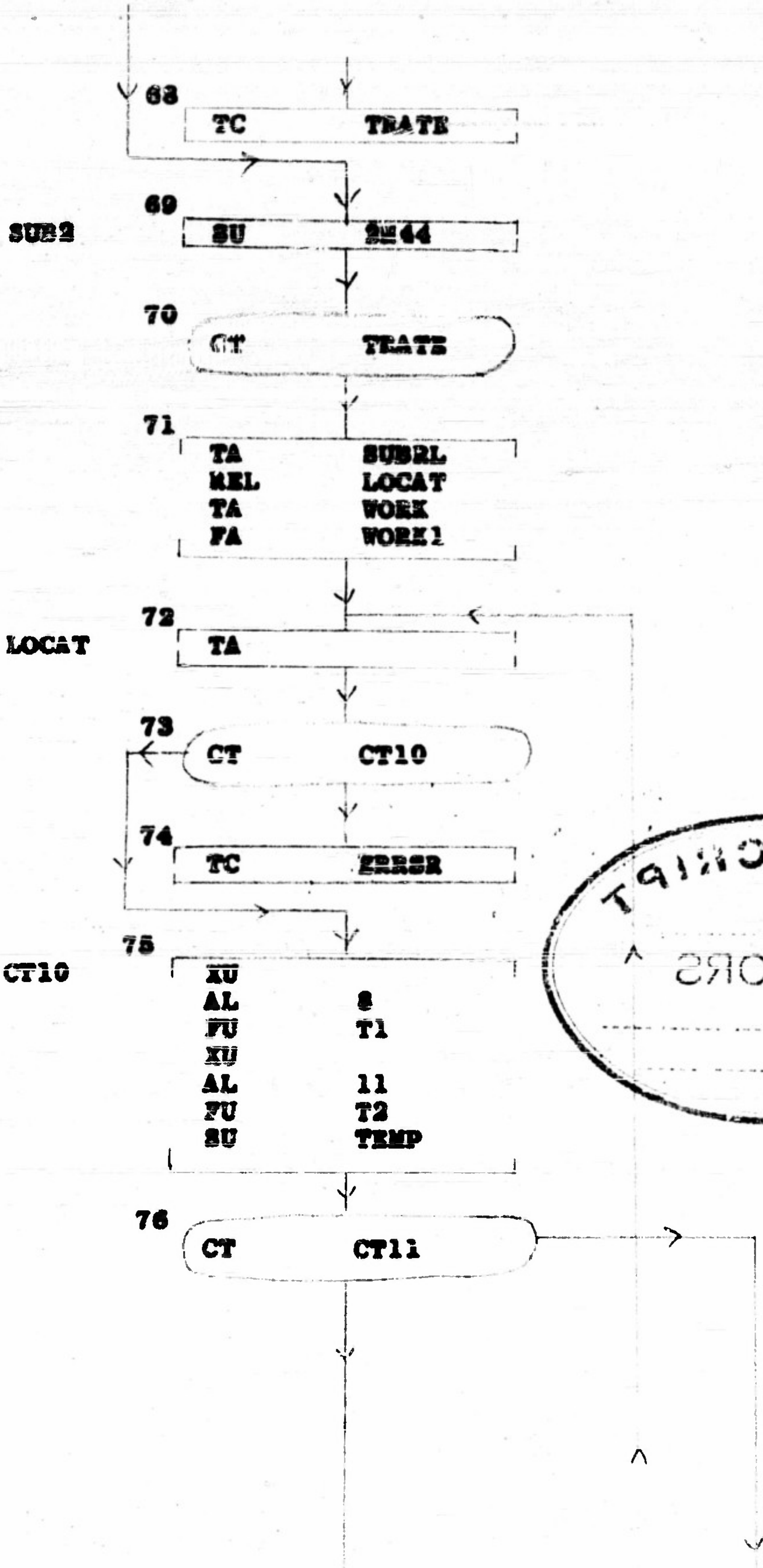
```
        33
                IN      CHNG1
                IN      CHNG2
                TC      CHNG1

        39
LAST            TA
                RE      PUTIN
                RE      CHKED
                RE      RECH
                RE      FIXT
                TA      NUMB
                REL     BANK
                TA      SWICH

        40
                CT      TAB

        41
                MA      BANK
                TC      PUTIN

        42
TAB             TA      BANK

        43
PUTIN           FA
                INL     WORK
                TA      SUM
FIXT            ADD
                FA      SUM
                TA      TOTAL
                SU      ILAD
                FA      TOTAL
                FA      TALI
```

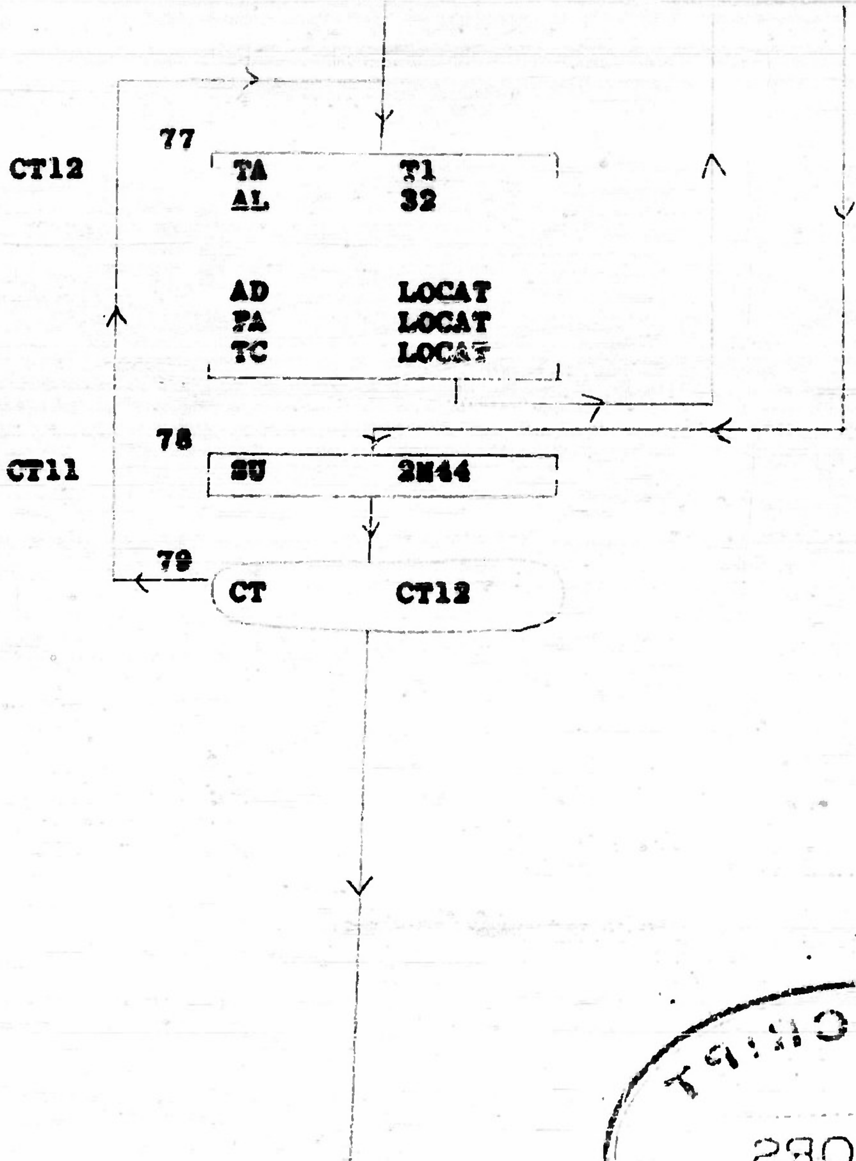
```
        44
CHKND         TAB
              XU
              AL          19
              FA          T1
              FU          TEMP
              TA          HOLDF
              RE          TAAD
              XA
              FA          TALI

        45
SUND          SU          NHLD

        46
              CT          OPER

        47
TAAD          TA

        48
              CT          TC606

        49
              FA          T2
              MA          2N44
              FA          LRDEP
              MA          T2
              TC          CLRU

        50
TC606         XA
              FA          LRDEP

        51
CLRU          XU
              AL          36
              AR          36
              FA          T2
              UA
              AL          19
              SU          T1
```


52
CT CT13
53
INCT
IN TAMD
IN TALI
TC SUMD
54
CT13
SU 2M44
55
CT INCT
56
TA TEMP
AL 8
AD T2
FA T2
TA LRDEP
57
CT STOR
58
TA CHKMD
RE FOLOW
FOLOW
TA
59
CT STOR
60
TA T2
AD 2M44
FA T2

```
        61
STOR          TA        HADDF
              AD        TAL1
              RE        REO
              RE        REO1
              REL       TRADD
              SU        ILADD
              RER       TRADD
REO           TA
              AL        43
              SU        2M1

        62
              CT        CT16

        63
REO1          TA
              RE        RE1
              RE        RE2
RE1           TA
              AD        T2
RE2           FA
              TC        TC3

        64
CT16          TA        T2
              AL        24
              FA        T2
              TC        REO1
```

```
        65
TC3         TA      HOLDF
            AD      TALI
            REL     TRHLD
            AD      ILADD
            RER     TRHLD
            TA      NHLD
            SU      ILADD
            TA      NHLD
            SU      TALI
            AR      32
            FA      T3
            AD      TRHLD
            FA      TRHLD
            TA      T3
            AD      TRADD
            FA      TRADD
TRHLD       BT
TRADD       BT
            TC      INCT

        66
OPER        TA      WORK
            RE      OP
OP          TA
            XU
            AL      19
            FA      TEMP
            UA
            SU      SUBIN

        67
            CT      SUB2
```

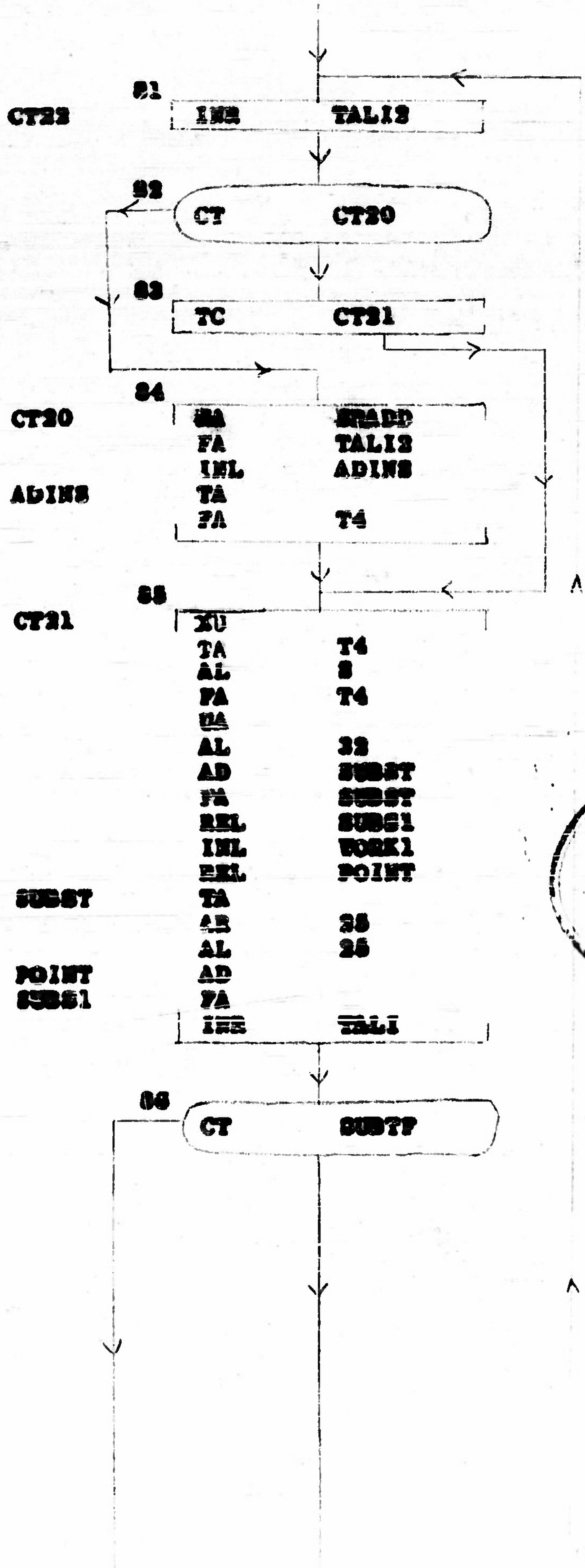

68
TC TRATE
69
SUB2
SU 2M44
70
CT TRATE
71
TA SUBRL
REL LOCAT
TA WORK
FA WORK1
72
LOCAT
TA
73
CT CT10
74
TC ERROR
75
CT10
XU
AL 8
FU T1
XU
AL 11
FU T2
SU TEMP
76
CT CT11

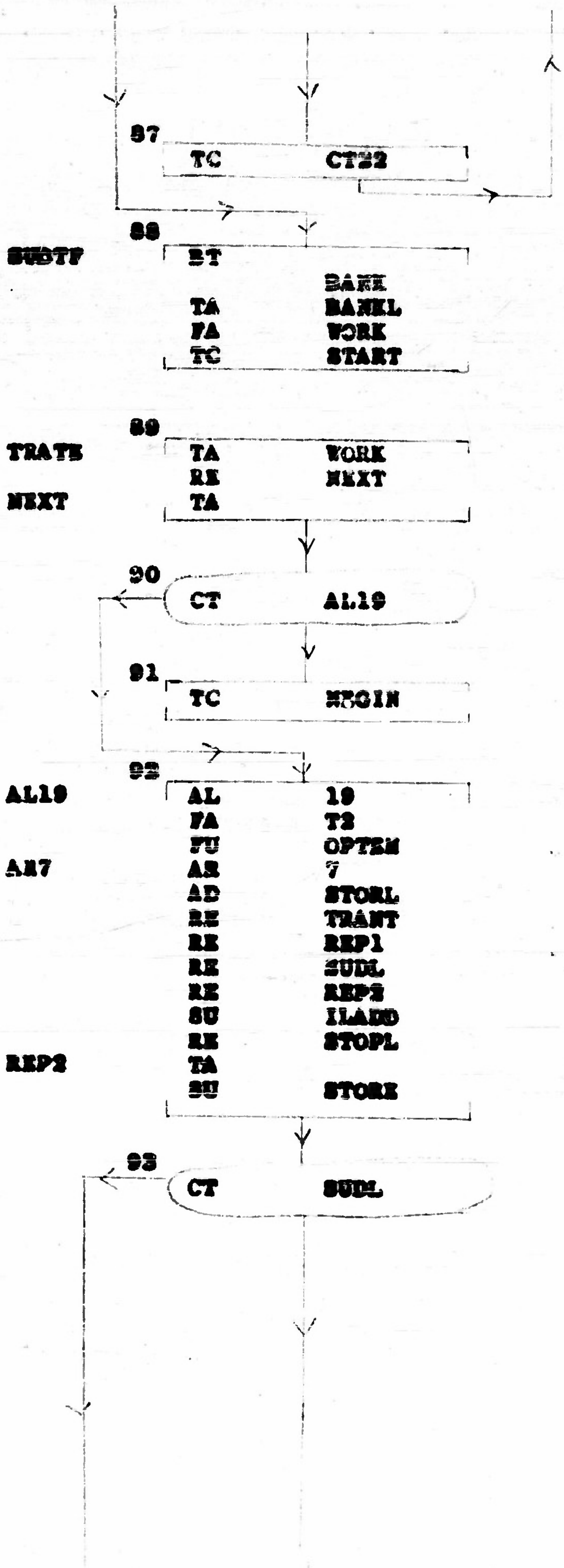


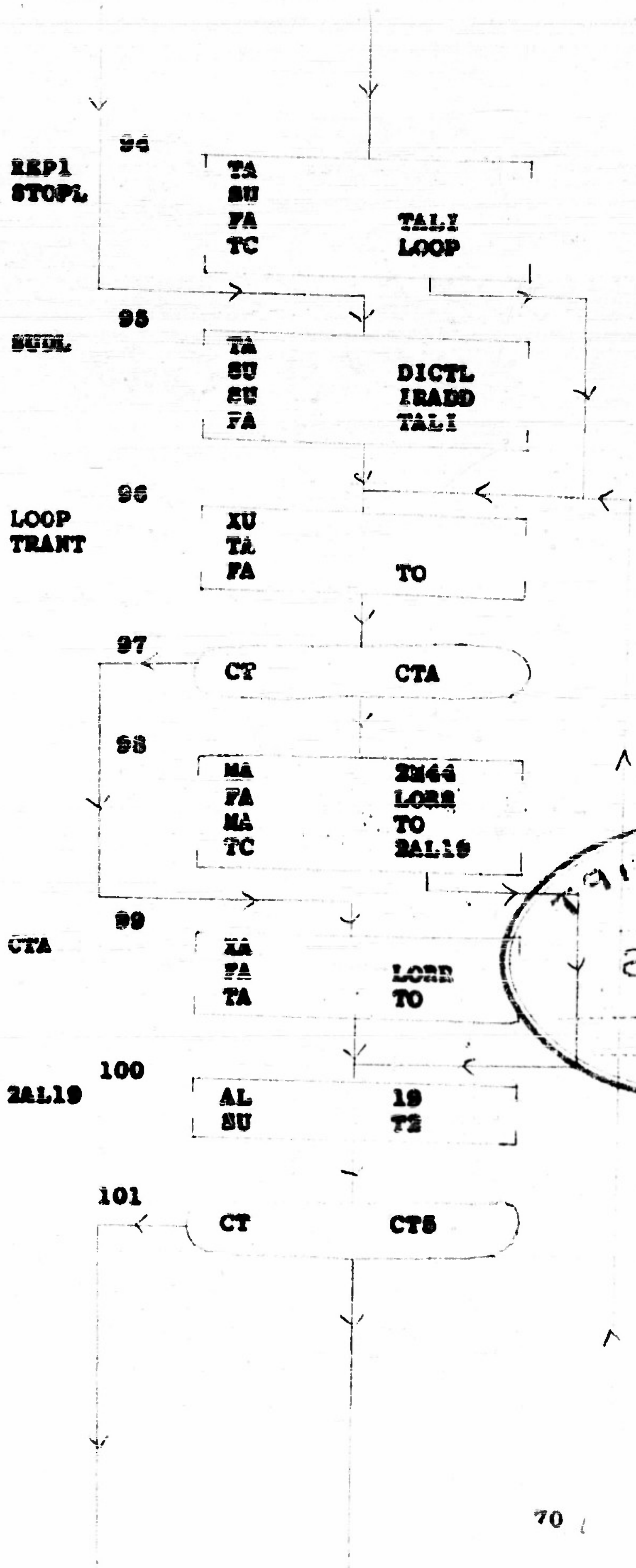
77
CT12
TA T1
AL 32
AD LOCAT
FA LOCAT
TC LOCAT
78
CT11
SU 2M44
79
CT CT12

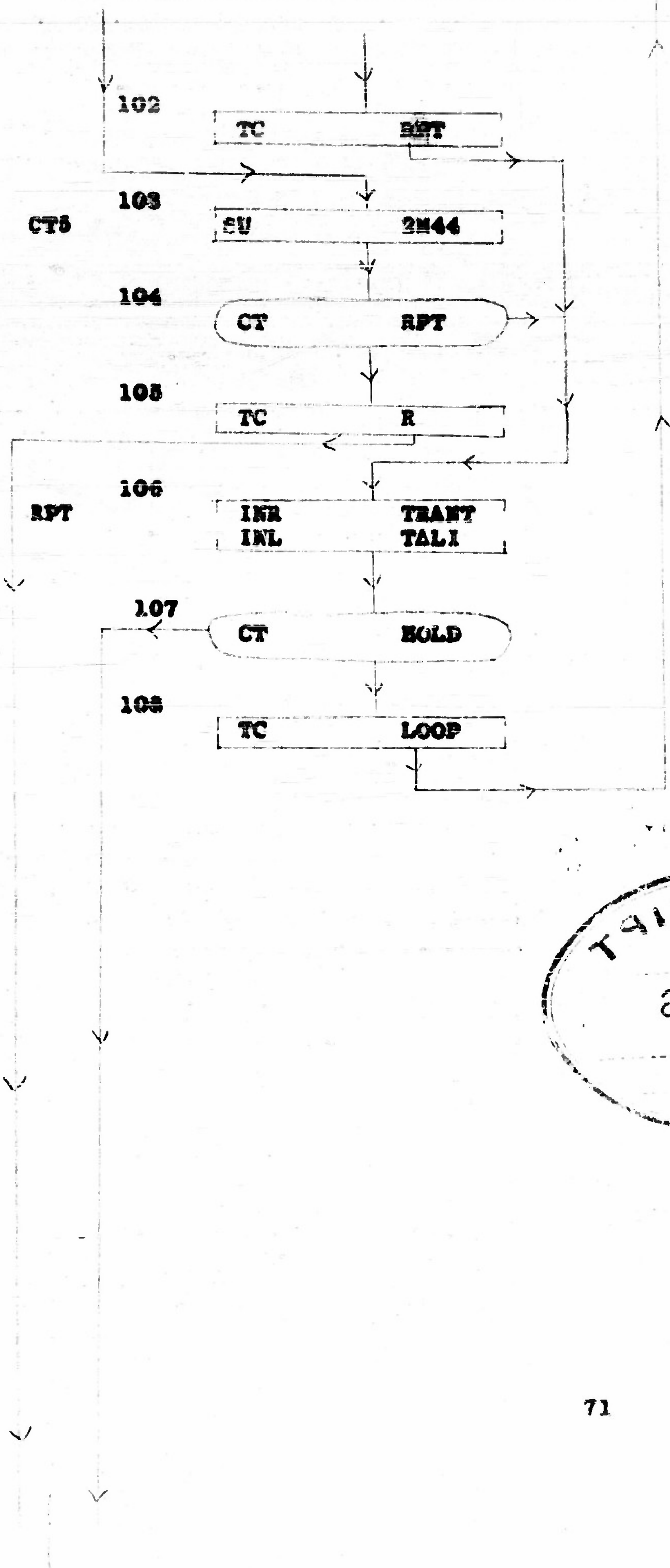
80

```
XU
FU    TAL12
TA    T2
AL    30
FU    T3
XU
AL    3
FA    T6
UA
AL    41
MU    29M41
AD    T1
FA    T1
TA    LOCAT
REL   ADINS
REL   SUBTF
AR    8
AL    8
AD    T1
FA    SUBTF
REL   SUBST
MA    T3
AL    8
FA    TALI
TA    T6
AR    41
SU    T1
AL    36
AR    4
FA    TAL13
```

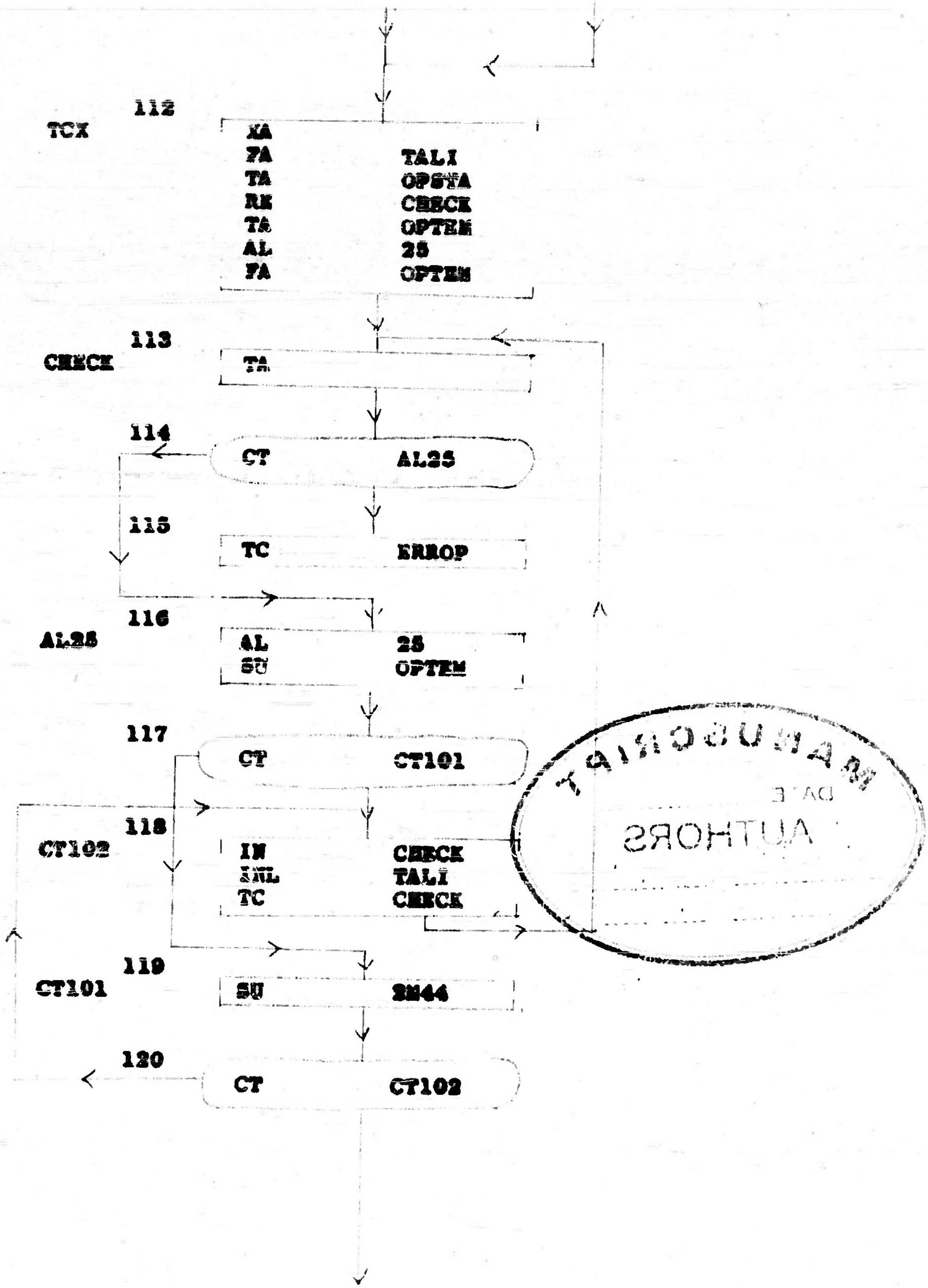

81
CT22
INR TALI2
82
CT CT20
83
TC CT21
84
CT20
UA SRADD
FA TALI2
INL ADINS
ADINS
TA
FA T4
85
CT21
XU
TA T4
AL 8
FA T4
UA
AL 32
AD SUBST
FA SUBST
RSL SUBS1
INL WORK1
RSL POINT
SUBST
TA
AR 25
AL 25
POINT
AD
SUBS1
FA
INR TALI
86
CT SUBTF

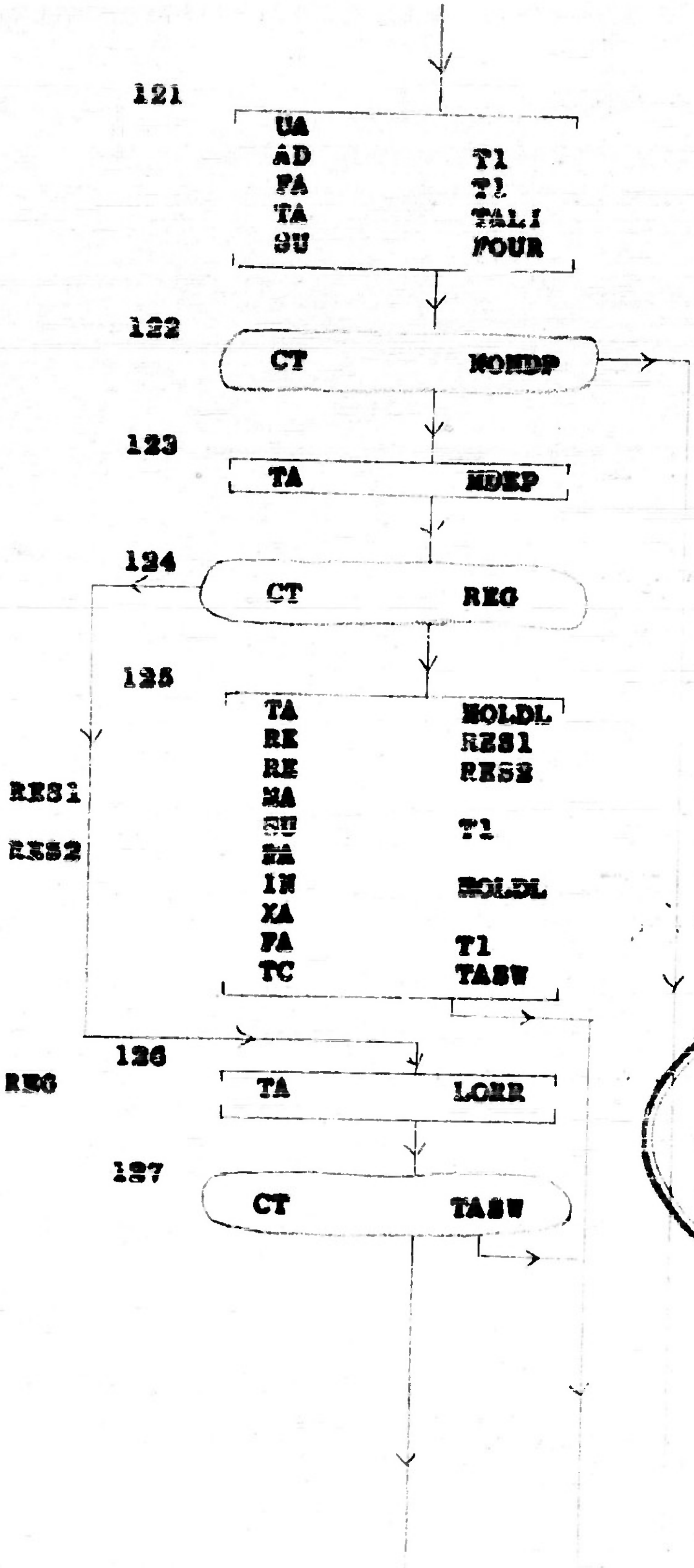

87
TC CT32
88
SUBTF BT
BANK
TA BANKL
FA WORK
TC START
89
TRATE TA WORK
RE NEXT
NEXT TA
90
CT AL19
91
TC BEGIN
92
AL19 AL 19
FA T2
FU OPTEM
AR7 AR 7
AD STORL
RE TRANT
RE REP1
RE SUDL
RE REP2
SU ILAND
RE STOPL
REP2 TA
SU STORE
93
CT SUDL


94
REP1
STOPL
TA
SU
FA TALI
TC LOOP
95
TA
SU DICTL
SU IRADD
FA TALI
96
LOOP
TRANT
XU
TA
FA TO
97
CT CTA
98
MA 2M44
FA LORR
MA TO
TC 2AL19
99
CTA
XA
FA LORR
TA TO
100
2AL19
AL 19
SU T2
101
CT CT5


102
TC RPT
103
CT5
SU 2M44
104
CT RPT
105
TC R
106
RPT
INR TRANT
INL TALI
107
CT HOLD
108
TC LOOP

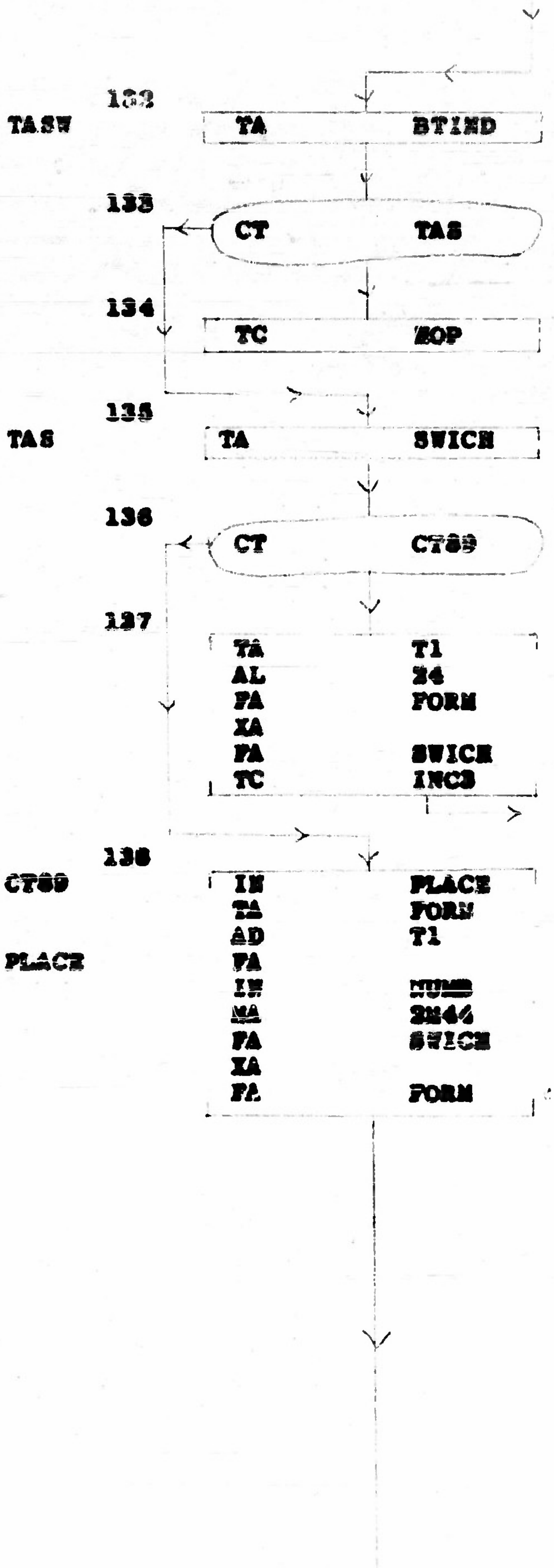
```
         109
HOLD            TA          HOLDL
                RE          HLD
                TA          WORK
                RE          TAB
TAB             TA
                AL          19
                FU          OPTEM
                AR          11
HLD             FA
                MA          2M44
                FA          HDKP
                XA
                FA          T1
                IN          HHLD
                TA          HADDL
                RE          STADD
                TA          PLACE

                RE          ADS
                MA          SWICH
ADS             AD
STADD           FA
                IN          HADDL
                TC          TCH

         110
R               U
                AL          2
                FA          T1
                TA          OPIND

         111
                CT          STOP
```


112
TCX
XA
FA TALI
TA OPSTA
RE CHECK
TA OPTEM
AL 25
FA OPTEM
113
CHECK
TA
114
CT AL25
115
TC ERROP
116
AL25
AL 25
SU OPTEM
117
CT CT101
118
CT102
IN CHECK
INL TALI
TC CHECK
119
CT101
SU 2M44
120
CT CT102



121
UA
AD T1
FA T1
TA TAL1
SU FOUR
122
CT NONDP
123
TA MDEP
124
CT REG
125
TA HOLDL
RE RES1
RE RES2
MA
SU T1
MA
IN HOLDL
XA
FA T1
TC TASW
RES1
RES2
126
REG
TA LOEE
127
CT TASW

```
        128
                TA          T1
                AD          2N44
                FA          T1
                TC          TASW

        129
HONDP           TA          HDSP

        130
                CT          TASW

        131
                TA          HOLDL
                RE          RET1
                RE          RET2
RET1            TA
                AD          T1
RET2            FA
                XA
                FA          T1
                TC          TASW
```


132
TASW
TA BTIND
133
CT TAS
134
TC ZOP
135
TAS
TA SWICH
136
CT CT89
137
TA T1
AL 24
FA FORM
XA
FA SWICH
TC INC3
138
CT89
IN PLACE
TA FORM
AD T1
PLACE
FA
IN NUMB
MA 2M44
FA SWICH
XA
FA FORM

```
        139
INCB          INR     TAL13

        140
              CT      RPST

        141
              INL     WORK
              TC      START

        142
NRGIN         FA      T0
              MA      T0
              FA      T0
              AR      43
              SU      2M44

        143
              CT      BTIN

        144
              TA      T0
              AL      19
              FU      OPTEM
              AR      19
              FA      T1
              TC      TCX

        145
BTIN          TA      SWICH

        146
              CT      CTY

        147
              TC      TCS
```

```
        148
CTY             TA          PLACE
                RE          RE12
                TA          FORM
RE12            FA
                IN          PLACE
                MA          ZM44
                FA          SWICE
                IN          NUMB

        149
TCS             MA          ZM44
                FA          BTIND
                TC          TRATE

        150
HOP             XA
                FA          OPIND
                TA          T1
                AL          24
                FA          STBT
                IN          WORK
                RE          B1
B1              TA
                AL          19
                FU          T7
                FA          T3
                TC          AR7

        151
BTOP            TA          T1
                AD          T7
                AD          STBT
                TC          PLACE
```

```
        152
ENDIC            RC          MEMER
                 ST

        153
ERRSR            RC          SRER
                 ST

        154
ERROP            RC          OPSER
                 ST
```

1. All the variables, constants, and parameters used in the main routine are read into machine in lowest address first. These variables are in relative form and occupy the last twenty-five bits of the storage words beginning with BANK1. They are preceded by a word in BANK containing the total number of variables, constants, and parameters located in the left-address position.

ASSGN is set up with the address in the machine form to be assigned to the first variable.

The negative of the total number of variables, etc., is put in TOTAL.

T1 and TALI are set up. T1 contains successively the partial sums obtained after adding each variable in the dictionary. TALI contains the negative of the number of variables yet to be added to the sum. When TALI becomes zero, all the variables have been added.

2. The next variable is added to the partial sum in T1 and the new partial sum is stored in T1.

TALI is increased and modified word in TALI appears in A.

3. If TALI is now zero all the words have been added. Control is transferred to block 5.

4. If TALI is negative, the instructions in 2 are adjusted and control is transferred to 2 to repeat the process until all the words have been added.

5. Word in ADICT is put in A.

Note: ADICT is used to indicate whether or not the dictionary was added up. A negative number in ADICT indicates that the dictionary has been summed up previously; a non-negative number indicates that it has not been summed.

6. If ADICT is non-negative, control is transferred to 13 where ADICT is changed to a negative number.

7. The sum obtained previously from adding up the dictionary is subtracted from SUM.

The first twelve bits of the result are eliminated by a left and right shift.

NOTE: The first time dictionary is added up, the words contain only the symbolic forms of variables. After this the dictionary words contain also the machine code in left-address position. We want only to compare the sums of the variables in symbolic form.

8. If the difference between the two sums is not negative, control is transferred to 10 to see if the difference is zero.

9. If the difference between the two sums is negative, then the two sums obtained by adding dictionary at different times are not equal, so control is transferred to ERDIC to indicate an error in the memory.

10. If the difference is not negative, it must be determined whether or not it is zero. The smallest possible quantity is subtracted from the difference.

11. If the result is not negative, then the difference was greater than zero and control is transferred to ERDIC to indicate an error in the memory.

12. If result is negative, the difference between the two sums is zero. Control is transferred to START.

13. After the dictionary is summed the first time a negative word is transferred to ADICT and the sum is stored in SUM.

14. Address is assigned to first variable in BANK1, and is stored in the left-address position of word in BANK1. From the left-hand letter of symbolic form of the word in BANK1, it is determined

which of the thirty-two words beginning with STORE should be increased.

The address in ASSGN is increased. The last address to be assigned to variables is subtracted from it.

15. If the difference is positive then all the variables in BANK have been taken care of and control is transferred to SORT (20).

16. If difference is negative, then more variables have to be dealt with. The instructions in 17 are adjusted to repeat process and control is transferred to 17.

NOTE: At this point all the words beginning with BANK1 have symbolic form of address in last twenty-five bits and machine code in left-address position. In addition the number of variables having each of thirty-two possible left-hand letters is stored in the left-hand address positions of the thirty-two words in block of storage locations from STORE to STO31. The number in STORE is the number of variables which have 00000 as left hand letter.

17. Instructions are set up for replacing the numbers in the set of words beginning with STORE by the address in the dictionary of the last word beginning with the corresponding left-hand letter.

The negative of 32 is put in TALI.

The address of the last storage location to be assigned to the dictionary is put in T4.

Zero is put in T2

18. T4 contains the last storage location assigned to the previous block of variables.

The number of variables in the previous block is put in T1.

T1 is subtracted from T4 giving the last storage location for the block of variables. This is stored in the appropriate location beginning with STORE. (The first time the machine goes through these instructions DICTL, the last storage location for the dictionary is put in STORE and in T4. The next time the number of variables having 00000 as left-hand letter is subtracted from DICTL to give last address of variables having 00001 as left-hand letter. This address is stored in STOR1.)

TALI is increased.

19. If TALI is 0 all the 32 words in STORE have been replaced by the appropriate addresses, so control is transferred to TRAN.

20. Instructions in 21 are adjusted to repeat the process and control is transferred to 21.

NOTE: At this point the storage locations from STORE to STO31 contain the last address where the block of variables having a corresponding left-hand letter will be stored.

21. The negative of the number of variables in the dictionary is put in TALI.

22. From left-hand letter of variable in BANK1 it is determined to which block of variables it belongs. The variable is stored in the last location allotted to that set of variables. The word in STORE is decreased by one so that next variable having same left-hand letter will be stored in the next lower storage location of that block.

TALI is increased.

23. If TALI is not negative, then all the variables have been transferred to the appropriate places in the dictionary, so control is transferred to AA(28).

24. If TALI is negative, all variables have not been transferred. Instructions in 25 are adjusted to repeat process. Control is transferred to 25.

25. Transfer instruction which occurs later on is set up. Pattern subroutines are read into the machine. Operation words are read into the machine.

The location to be assigned to the first instruction is put in NUMB.

The address of the location where first coded instruction word is to be stored in put in the address position of PLACE.

SWICH is set up with a negative number. (SWICH is used to indicate whether instruction is to be in the left or right-hand order of control word.

26. First word or set of words regarding instruction(s) are read into BANK.

Address of BANK is put in WORK.

OPIND, BTIND, TALI2, are set up.

27. Put the word in BANK in A.

28. If it is a positive word, control is transferred to 33.

29. If it is a negative word, control is transferred to NEGIN which deals with negative instruction words.

30. If it is a positive word, it is tested first to see if it is a named instruction word by subtracting the number used to indicate a named instruction word from appropriate part of the word in BANK.

31. If result is not negative, it must be determined whether it is zero or greater than zero, so control is transferred to CT2

32. If result is negative, it is not a named instruction and control is transferred to OPER.

33. Smallest possible quantity is subtracted from the difference.

34. If result is not negative, the word in BANK is not a named instruction. Control is transferred to OPER.

35. If result is negative, the word in BANK is a named instruction with symbolic form of the name of the instruction in the last twenty-five bits.

It is determined from the left-hand letter of the name of the variable into which block of the dictionary it should be inserted. This word will be inserted at beginning of the block.

All variables in lower storage locations are transferred to one lower storage location to make room for this word in the dictionary.

TALI is set up with the number of addresses in STORE which have to be changed as a result of this block transfer.

36. The first word in STORE to be changed is decreased by one. TALI is increased one.

37. If TALI is not negative, all the appropriate addresses have been changed so control is transferred to LAST.

38. Adjustments are made in the instructions in 39 to repeat the process and control is transferred to 39.

39. The address to be assigned to the instruction is put in the left-address position of the word in BANK.

SWICH is put in A.

40. If SWICH contains a nonnegative number, the instruction is a right-order one, so control is transferred to TAB (45)

41. If SWICH contains a negative number, the instruction is a left-order one. Therefore the negative of the word is stored in the dictionary. The negative of the word in BANK is put in A.

42. The word in BANK is put in the A register.

43. Word in A is transferred to appropriate place in dictionary.

WORK is increased.

The dictionary sum and the total number of words in the dictionary are changed to take account of the new word that has just been inserted.

44. Now the instructions in HOLD must be checked to see if the symbolic address just inserted in the dictionary has been used in previous instructions.

T1 is set up with the symbolic form of the address just inserted into the dictionary.

TEMP is set up with the machine form of the address.

TALI is set to zero.

Appropriate address is inserted in MAP.

45. The number of instructions in HOLD is subtracted from TALI.

46. If the result is not negative then there are no more instructions in HOLD to check, so control is transferred to OPER.

47. If the result is negative, the next word in HOLD is put in A register.

48. If the instruction word in HOLD is stored positively, control is transferred to 53.

49. If the word in HOLD is negative, the instruction involves one of the variable operations. The word is stored in T2. A negative word is put in LRDEP to indicate that operation must be adjusted if address appears in the right-hand order of the storage word.

The negative of T2 is put in A.

Control is transferred to 53.

50. If instruction word in HOLD is stored positively, then a non-negative word is put in LRDEP.

51. Operation in machine form of the address, in symbolic form of the word in HOLD, is separated. Operation is stored in T2. The symbolic form of the address of named instruction which was just inserted in

the dictionary is subtracted from symbolic form of the address of this instruction in HOLD.

52. If result is not negative, control is transferred to 56, to determine whether result is indeed zero.

53. If the difference between the two addresses is not zero, adjustments are made in appropriate instructions in, and control is transferred to, 48 to repeat the process with the next instruction in HOLD.

54. SU 2M44

55. If result is still non-negative, control is transferred to 53.

56. If the addresses do correspond, the machine code of the address which has been stored in TEMP is added to the machine form of operation which has been stored in T2. The result is stored in T2. The number in INDEP is put in A.

57. If number is not negative, then the operation is not a variable one. Control is transferred to 63.

58. If the number is negative, then the operation is a variable one and it must be determined whether the address refers to the right or left hand order of a storage word.

Therefore the word in dictionary corresponding to the address of this instruction is put in the A register.

59. If the word is stored positively then the address is of an instruction which is stored in the left-hand order of the storage word. Nothing need be done to alter operation. Control is transferred to STOR.

60. If the word is stored negatively, then the address is of an instruction which is stored in the right hand order of the storage word. The machine code of operation is increased by one and stored in T2.

61. TALI is added to address in HADDF to determine where the instruction is to be stored.

The word containing the address of storage location where the instruction is to be stored is put in the A register.

The word in A is shifted forty-three places to the left so that only the last bit appears in the A register.

2M1 is subtracted from the number appearing in A.

62. If the result is not negative, there was a one in the last bit indicating that instruction is to be stored in the left-hand order of the control word.

Control is transferred to CT18(67).

63. If result is negative there is a zero in last bit indicating that instruction goes in right hand order of the storage word.

The instruction in T2 is added to what already appears in the storage location and is stored there.

Control is transferred to TC3

64. Instruction in T2 is shifted to left hand order of word and is stored in T2.

Control is transferred to 61.

65. Block transfer instructions are set up.

NELD is decreased.

66. The appropriate word in BANK is tested to see if it is the name of a pattern subroutine by subtracting the number used to indicate a subroutine from appropriate part of the word.

67. CT SUB2

68. If result is negative, the word is not a pattern subroutine, so control is transferred to TRATE.

69. SU 2M44

70. CT TRATE

71. The appropriate instructions below are set up to see which of the pattern subroutines in SUBR corresponds to the one in BANK.

72. First word of pattern subroutine is put in A register.

73. If word is not negative, control is transferred to CT10 (78) to see whether name of subroutine corresponds to name of the one in BANK.

74. If the word is negative, there is no subroutine in SUBR which corresponds to the one in BANK. Control is transferred to ERRSR to indicate an error in a pattern subroutine.

75. The different bits of information are separated from the first preword of the pattern subroutine and stored.

TEMP, the name of the subroutine in BANK is subtracted from the name of the subroutine in SUBR.

76. CT CT11

77. LOCAT is adjusted so that process may be repeated with the next pattern subroutine in SUBR.

78. SU 2MAS

79. CT CT12

80. When appropriate subroutine in SUBR is found TALI2 is set to zero. The number of addresses to be inserted in subroutine is stored in T3. The block transfer instruction which transfers instructions in subroutine is stored in T3. The block transfer instruction which transfers instructions in subroutine to BANK is set up. TALI is set up with negative of the number of addresses to be inserted in subroutine in right-address position. TALI3 is set up with the negative of the number of words in subroutine referring to ordinary instructions.

81. TALI2 is increased.

NOTE: TALI2 contains the negative of the number of differences in preword yet to be considered.

82. If TALI2 is not negative, control is transferred to 87.

83. If TALI2 is negative, control is transferred to 88.

84. TALI2 is set up with binary equivalent of five in right-address position. The next preword of differences is put in T4.

85. First difference is taken out of T4. Address of instruction where variable is to be inserted is determined and the variable is inserted. TALI is increased.

86. If all the variables have been inserted in appropriate instruction words, control is transferred to 91.

87. If all the variables have not been inserted, control is transferred to CT22 (84).

88. When all variables have been inserted in the appropriate instruction words, the instruction words of the subroutine are transferred to BANK. Control is transferred to START, so that each instruction may be translated.

89. Word in BANK is put in the A register.

90. If word is not negative, control is transferred to AL19(93).

91. If the word is negative, control is transferred to NEGIN.

92. If the word in BANK is an ordinary instruction, the address and operation are separated. Address is stored in T2 and the operation is stored in OPTEM. From left-hand letter of the address it is determined to which block of THE DICTIONARY it belongs. Address in STORE is subtracted from the first address of this block of variables. The difference will be negative or zero.

93. If the difference is zero, control is transferred to SUBL.

94. If the difference is negative, the negative of the number of variables in block is obtained by subtracting address of the first variable in block just below from the address of first variable in

this block. This number is stored in TALI. Control is transferred to LOOP.

95. If difference is zero, the block of variables that has to be searched is the last block in the dictionary. The negative of the number of variables in this block is found by subtracting 1 plus DICTL from address of first variable in block. This is stored in TALI.

96. The dictionary word is put in the A register.

97. CT CTA

98. If the dictionary word is stored negatively, the address is of a left-order instruction so a negative word is put in LORR.

NOTE: LORR will be referred to later only if the operation is a variable one. For example, if the operation is IN, LORR will indicate whether right or left.

The negative of the dictionary word is put in the A register. Control is transferred to [illegible](103).

99. If dictionary word is positive, zero is stored in LORR. The dictionary word is put in A.

100. The symbolic address of instruction is subtracted from symbolic form in dictionary.

101. CT CT5

102. TC RPT

103. SU ZL64

104. CT RPT

105. TC E

106. Adjustments are made in instructions to repeat the process with next word in dictionary.

TALI is increased.

107. If no word in dictionary is found which corresponds to address of instruction, control is transferred to HOLD to stow instruction until new address is inserted in dictionary

108. If there are more variables to work with, control is transferred to LOOP (58) to repeat the process.

109. Operation and address are separated.

Operation stored in OPTEM

Address is stored in symbolic form after the last instruction in HOLD.

A negative word is put in HDEP, to indicate when operation is translated, that it is to be stored in HOLD.

A zero is put in T1.

NHLD is increased.

Address of storage location where instruction is to be stored is stored in HADDL with the negative of the number in SWICH in the forty-fourth bit of the storage word to indicate whether the instruction is to be stored in the left or right-hand order of the storage word.

HADDL is increased.

Control is transferred to TCE.

110. Machine form of address is stored in right-address position of T1.

Word in OPIND is put in the A register.

111. A non-negative word indicates that word is the second one of a block transfer word. There is no operation to be translated, so control is transferred to ETOP.

112. TALI is set to zero.

Adjustments are made to find the machine code for operation.

113. Operation word is put in A register.

114. CT AL28

115. If operation word is negative, no word in list of operations is found to correspond to operation of instruction. Control is transferred to ERROP to indicate error in operation.

116. Symbolic form of operation of instruction is subtracted from symbolic form of operation stored in machine.

117. CT CT101 (125)

118. If difference is not zero, instructions are modified and TALI is increased. Control is transferred to CHECK to repeat process with next operation stored in the machine.

119. SU ZM44

120. CT CT102

121. Machine code of operation is added to address stored in T1 and result is stored in T. Four is subtracted from number in TALI to see if operation is one of the first four variable instructions.

122. If operation is not variable, control is transferred to NONDP.

123. If operation is variable, HDEP is put in A register to see if instruction is one that is to put in HOLD until new addresses are added to dictionary.

124. If HDEP is not negative, instruction is not to be stored in HOLD. Control is transferred to REG.

125. If the instruction involves an address which has not yet been put in the dictionary and if the instruction involves one of the variable operations, then the machine form of operation is added to the symbolic form of the address and the result is stored negatively in HOLD. Control is transferred to TASW.

126. If the instruction involves a variable operation but is not to be stored in HOLD, then it is determined from the word in LOAR whether or not the address referred to in the instruction is the right or left-hand order of the storage word.

127. If the word in LORR is not negative, the address refers to the left-hand order of the storage word. Nothing need be done about altering the operation, so control is transferred to TASW.

128. If the instruction involves a variable operation and if LORR indicates that the address refers to the right-hand order of the storage word, the machine code for operation is increased by one and result is stored in T1. Control is transferred to TASW.

129. If instruction does not involve one of the variable operations, it is determined from the word in HDEP whether or not instruction is to be stored in HOLD.

130. If HDEP contains a non-negative word, instruction is not to be stored in HOLD and control is transferred to TASW.

131. If HDEP contains a negative word, the machine form of operation is added to symbolic form of address and result is stored positively in HOLD. Control is transferred to TASW.

132. The word in STIND is put in A.

133. A Positive number in STIND indicates an ordinary instruction and control is transferred to TAS (125).

134. A negative word in STIND indicates a block transfer instruction. Control is transferred to MOP to take care of the next word.

135. The word in SWICH is put in the A register.

136. If word is not negative, control is transferred to CT89.

137. A negative word in SWICH indicates that the instruction is to be stored in the left-hand order of the control word. The coded instruction in T1 is shifted to the left-hand order of the word and

is stored in FORM. SWICH is reset and control is transferred to INC3.

138. A positive word in SWICH indicates a right-order instruction. Instruction in T1 is added to instruction in FORM to form a word with anright and left-order instruction, SWICH is reset and FORM is set to zero.

139. TALI3 is increased.

140. If TALI3 is not negative, there are no more words in BANK to be taken care of, so control is transferred to EDST.

141. If TALI3 is negative, there are more words in BANK to translate. Work is reset and control is transferred to START.

142. If word in BANK is negative, it will be either a shift or a block transfer instruction. The positive form of the word is put in TC and in A. Word is shifted so that only the first bit is in A. SM44 is subtracted from part remaining in A.

143. If result is not negative, then instruction involves a block transfer, so control is transferred to BTS.

144. If result is negative, instruction involves a shift. The number of bits to be shifted is stored in T1. Control is transferred to TCS to translate the operation.

145. SWICH is put in the A register.

146. A positive word in SWICH indicates that the instruction that was just translated was a left-hand one and was stored in T1. Control is transferred to CYT.

147. If previous instruction was a right-order one, control is transferred to TCS (149).

148. Since a block transfer instruction takes up a whole word, the right-hand order of previous instruction is set to zero if previous instruction was a left-order one. Instruction is stored. SWICH is reset.

NUMB is increased.

549. BTIND is set and control is transferred to TRATE to translate the first part of the instruction as an ordinary instruction.

550. When first word of block transfer instruction is translated, it is stored in left-hand order of STBT. OPIND is set so that only the address of the next word will be translated. FORK is reset. The two parts of the second block transfer word are separated. The number of words to be transferred is stored in T7. Address to be translated is stored in T2. Control is transferred to AR7 to translate the address.

551. The complete block transfer instruction is set up in A register in translated form. Control is transferred to PLACE to store instruction in appropriate place.

552. MEMER is recorded on magnetic tape to indicate an error in the dictionary memory.

553. SAER is recorded on magnetic tape to indicate an error in a pattern subroutine.

554. OPSER is recorded on magnetic tape to indicate an error in the operations.

ACKNOWLEDGMENTS

The work on Appendix I was done at the Naval Research Laboratory with the assistance of E. Wassel of the Naval Research Laboratory, and S. Anderson, formerly of Tufts College and now at the Naval Research Laboratory. The work on Appendix II was done with the assistance of F. Spatz of Tufts College. The coding system is an extensive elaboration of previous work done at Cornell Aeronautical Laboratory under a contract supported by the Office of Naval Research.