CONFERENCE ON AUTOMATIC PROGRAMMING OF DIGITAL COMPUTERS

MATHEMATICS DEPARTMENT

BRIGHTON TECHNICAL COLLEGE

April 1, 2, 3, 1959

AUTOMATIC PROGRAMMING ON DEUCE

by

C. ROBINSON, M.Sc.

(The English Electric Co Ltd)

Automatic Programming on DEUCE.

C. Robinson, M.Sc.

1. INTRODUCTION.

Despite the organisation of vast libraries of subroutines and programmes, and the facilities for testing new programmes, a real need has grown up for flexible and powerful schemes, capable of being used to construct new DEUCE programmes in a fraction of the normal time. During the last four years a variety of such schemes has grown up for DEUCE, and the present paper and a companion paper by Mr. S.J.M. Denison, reviews some of these, and fits them into perspective. The schemes STAC, GIP (General Interpretive Programme), TIP (Tabular Interpretive Programme), Alphacode, GEORGE, SODA, Easicode, STEVE, and the Alphacode Tabulator have been produced at intervals at various DEUCE establishments, to all of which credit should be given for the introduction of new ideas and exploitation of older ones. A study of the way in which most of these schemes develop ideas in the others, and contribute new techniques is in itself an interesting geneological exercise.

The present paper is concerned with some automatic aids to DEUCE programming, and some interpretive schemes. The companion paper discusses other interpretive schemes and translation schemes.

---

2. NORMAL DEUCE PROGRAMMING.

It would perhaps be as well to prefix any remarks on automatic programming for DEUCE with a brief reminder of the machine's order-code and storage. The high speed memory consists of 402 words of mercury delay line store, and this is backed with a magnetic drum with 8192 words capacity. The drum store consists of 256 tracks of 32 words each, and transfer to and from the high speed store is one whole track at a time, the length of most of the delay lines being such that they also store 32 words. During such transfers to and from the high speed store, the rest of the facilities of the machine are available for parallel operations. The basic instruction word specifies a source and destination and causes a number to be transferred from the Source Delay Line to the Destination delay line. Apart from Sources and Destinations which represent real machine addresses, other pseudo-sources and destinations are used for multiplication, division, the accumulators, magnetic transfers etc. The instruction word is completed by a Wait Number which specifies the minor cycle(s) in which the transfer from Source to Destination is to take place, the Characteristic which effectively specifies the duration of the transfer which may be for more than one minor cycle, the Next Instruction Source which specifies the delay line from which the next instruction is to be taken, and the Timing number which specifies the minor cycle from which the next instruction is to come. Thus whereas the planning of a programme is complete when one has written down the Sources, Destinations and durations of transfer of all the instructions which the machine is to obey, in the sequence in which it is to obey them, there remains the operation of (i) allocating storage to these orders, preferably so as to minimise time, (ii) coding, that is completing each instruction by the addition of Next Instruction Source, Wait and Timing Numbers and (iii) punching out the instructions on the input medium. Though the first two of these operations do allow scope for ingenuity and skill, all three can be regarded as fairly routine operations which need not worry the programmer. Apart from the 32 word delay lines, there are shorter lines:- 4 single word, each with associated pseudo addresses for particular arithmetic or logical operations, 3 double word, one of which is a multiplier, divider and double length accumulator, and 2 quadruple length lines. This combination of delay line lengths allows great flexibility in programming, and speed of operation, but at the same time requires care in ensuring that transfers between these delay lines and the long delay lines are in the appropriate minor cycles. Further flexibility and speed is achieved by allowing operations which do not interfere with each other to go on in parallel. Mention has already been made of magnetic transfers, and in addition the reader, punch, multiplier and magnetic tape equipment may all be working simultaneously with, say, some 'housekeeping' operation.

3. AUTOMATIC PROGRAMMING.

It is, at this stage, worth while considering what are the operations which a programmer, faced with a problem and a computer, must undertake before he arrives at a solution. Figure 1 is a pictorial representation of the steps involved between a problem and its solution. Of course the intermediate steps do not all call for the same skill or effect, or consume the same time. Most people would agree that steps 2, 3, 10 and 11 are more difficult and more time consuming than

the inverse steps. Moreover these same steps are virtually independent of the machine on which the problem is to be solved, at least if the machine's own order code is being used, and this is also true of step 9. Steps 4 and 7 have a parallel on any machine whatever its order code, and steps 5, 6 and 8 largely arise because of the facility of optimum coding on DEUCE. All programmers will, of course, be familiar with setbacks in testing and debugging their programmes, and these operations are shown on Figure 1 by the return loops at the left hand side. When working in a machine's natural order code however, it is illuminating to note how much debugging is really independent of the machine and is due to solving the wrong problem, or due to logical errors.

Having now considered figure 1, we may define as automatic programming any machine aid which allows us to short circuit any of the steps in this diagram. Figures 2, 3 and 4 show the effect of some of the DEUCE automatic programming schemes in removing the inner steps of figure 1, and, more important, removing the frustrating feedback loops at the same time. The various schemes will now be discussed in a little detail. It will be convenient to discuss first the schemes which preserve for the user the basic DEUCE order code, and then to mention those schemes in which the programmer works in an order code having little or no resemblance to DEUCE.

4. STAC.

One of the earlier programming aids was a scheme (from R.A.E. Farnborough) in which, the flow diagram having been prepared, and the storage allocated, DEUCE itself was made to allocate Wait and Timing numbers and so allow a by-pass from steps 5 to step 9 on figure 1. More recently this has been superseded by STAC - (Storage Allocation and Coding) which effectively mechanises that steps between 4 and 9. The preparation of a programme here ends with a flow diagram specifying merely the source, destination and length of transfer. The STAC programme will then take these instructions, and given the storagg place available will make a satisfactory job of allocating the storage and optimum coding the programme. In doing so it will insert any subroutines called for, and plant and obey their links. If the inner loops of programmes are so marked, it will give them priority in the coding, and as an additional aid to the programmer it will accept symbolic addresses for any stores and allocate suitable minor cycles in delay lines for them. Not all the instructions need be specified in the flow diagram in DEUCE order code for STAC can accept a few so called 'super instructions' which it then breaks down into the correct sequence of machine instructions. The output of STAC is a fully coded programme pack, a full decimal copy of the flow diagram, a list of the real addresses allocated to the symbolic addresses, and a statement of which storage space remains unused in case the programme should require further modification. The scheme is of greatest use to the non-professional programmer who is not as a rule impressed by efforts to cut off milliseconds from a programme, if they result in the completed programme being delayed!

5. INTERPRETIVE SCHEMES.

The General Interpretive Programme, Tabular Interpretive Programme, and Alphacode are three widely used systems for DEUCE, none of which involve the programmer in learning orthodox DEUCE programming. The programming for those schemes is so different (and so easy to learn) that it is constructive to look upon them as three alternative machines, and look at the 'machine' specifications to see whether they are suitable for the job in mind. Appendix 1 lists the qualities of these three 'machines'. A few points call for some comment.

The General Interpretive Programme, is so general that it can be adapted to any kind of work, but it is particularly well suited to performing parallel calculations on data in bulk. For the interpretation of one G.I.P. instruction one may then get thousands of arithmetic operations performed, and the interpretive time ceases to be of importance. It is thus particularly well suited to matrix operations, and the bulk of the functions available in the library are for linear algebra. A feature of this scheme has been the number of problems which have been presented to DEUCE though G.I.P. as linear algebra problems, though not appearing so at first sight. This is perhaps not so surprising when one considers how the orderly methods of matrix operations are so similar to the orderly methods one must use to programme successfully for a computor. The Tabular Interpretive Programme, as will be described later, is particularly well suited to any operation which could be performed with a desk machine and a sheet of paper ruled into rows and columns, and where one normally performs like operations on all the numbers in a column and writes the answers in a new column. It is therefore rather like G.I.P. restricted to vector operations. Alphacode, caters for the extreme case where the bulk of the operations in a programme are on one

variable at a time, that is problems in which parallel calculations are not required on a number of variables. In this way the three schemes are mutually complementary in that they each show up to best advantage on different types of problem.

Another point of interest are the unique feature of G.I.P. in that it has a variable order code in the form of a library of functions which can be used with it. In all three schemes the instruction store is quite separate from the data store, which does not however prohibit instruction modification. The number of instructions available with each of these schemes appears small at first sight, but they represent sizable programmes bearing in mind that the individual orders are so comprehensive.

6. TABULAR INTERPRETIVE PROGRAMME (T.I.P.)

This programme devised at Bristol Aero Engines is the most remarkable example of simplicity in programming. The authors, faced with the problem of arousing computer consciousness in a number of engineers solved it in a novel way. DEUCE was presented to the potential users as something with which they were already quite familiar; namely a desk machine operator with a fast machine and a huge sheet of paper ruled into rows and columns. Instructions to DEUCE are then in a form exactly analogous to those to the human operator. We may for instance be accustomed to telling an operator to write successive integers in the first column, to square these numbers and write them in the second column and then subtract the numbers in the first column from those in the second. The tabular interpretive programme will accept these instructions in an even more concise form than they have been written above. Each instruction to T.I.P. comprises four numbers a, b, c and r. The number r tells the machine which of 31 operations it has to obey, and the numbers a, b and c detail the columns concerned. Thus

a	b	c	r
1	0	0	4
1	1	2	0
2	1	3	3
3	0	0	5

is the complete programme for performing the above operation. The first instruction has r = 4 which means "read data into the column designated by a, in this case column 1". The second instruction has r = 0 which is interpreted as "multiply the numbers in column a (1 in this case) by the corresponding numbers in column b (also 1 in this case) and write the results in column c. (i.e. column 2)" The reader will deduce from the next instruction that r = 3 means "subtract the numbers in column b from the corresponding numbers in column a and writo the results in column c". Similarly r = 5 is interpreted as "print out results from column a."

The user has therefore at his dispoeal a "human operator equipped with a sheet of paper ruled into 128 columns and 30 rows and with the facility of rubbing out any columns no longer required and writing new results there." In the above programme for instance the last two instructions could have been

a	b	c	r
2	1	2	3
2	0	0	5

The order code is particularly well fitted to scientific and engineering calculations. Apart from the elementary arithmetic operations and "read" and "print", Values of r have been allocated to logarithms and exponentials, trigonometrical and inverse trigonometrical functions, square roots, modulus, and some series. The instruction r = 15 causes column specified in c to be filled with the progressive sum of the elements in column a. Other values of r are used for shifting a column of figures up or down by one row, this facility being of use in finite difference operations. Functions for interpolation among the elements of a column and for three dimensional linear interpolation in data read to the machine by a special 'read' codeword are also available.

Of course, in practise, it is frequently required to operate on a column not with the contents of another column but with a constant. For this purpose, 128 constant stores are available, 32 with built-in constants (which may however be over-ridden) and the remainder at the programmer's discretion. These stores are referred to as N 0, N 1 etc. to N 127. Thus, the codeword

a	b	c	r
1	N4	2	3

causes the constant N4 which happens to be to be subtracted from each of the numbers in column and the corresponding results written in column 2. Similarly

a	b	c	r
N4	1	2	3

causes each of the numbers in column 1 to be subtracted from
The Codeword

a	b	c	r
Na	b	c	13

causes constant number a to be transferred to row b of column c and

a	b	c	r
a	b	Nc	13

causes the number in row b column a to be transferred to the cth constant store.

Two concepts which are strange to the newcomer to programming are those of counting the number of times round a loop and of modifying instructions. In these respects, simplicity is the essence of the T.I.P. facilities. Any value a, b or c which has an asterisk against it is automatically increased by one every time it is obeyed. The instruction

a	b	c	r	
0	b	c	16	or
0	Nb	c	16	

is interpreted as "jump to instruction number c until the instructions from c onwards have been obeyed b (or Nb) times. In these instructions b and c do not refer to column numbers. If b is specified the assumption is that the programmer knows precisely how many times he wishes to go round the loop, (which probably contains some asterisked instructions); if Nb is used this constant store may well contain a number (the number of times round the loop) which has itself been computed by the programme. For example, the three instructions:-

Instruction number	a	b	c	r
c	1	100	100	2
c + 1	0	99	c	16
c + 2	0	0	0	18

cause the first 100 columns to be added together. The instruction r = 18

which may only follow an r = 16 instruction causes all the asterisked instructions in the loop ending with previous instruction to be reset to their initial value. Up to second order loops are permissible. There is one further instruction of this type for use with iterative loops:

a	b	c	r
a	Nb	c	17

means jump to instruction c repeatedly until such time as the corresponding numbers in columns a and a + 1 are all within the percentage tolerance specified in the bth constant store, and then proceed normally.

A persistent annoyance to a programmer is to be held up by a failure instruction on a machine due to trying to divide by zero, or trying to find the square root of a negative number or other error due to the data or the logic. Such errors can be time-consuming on the machine, particularly when, as is the case with T.I.P. the programmer does not do his own testing. (Indeed the illusion of the machine as a piece of paper is maintained to the point that the programmer need never have heard of a punched card - the usual DEUCE input/output medium). The present version of T.I.P. deals with such contingencies in two ways. If a tolerance only just fails - for instance trying to find the inverse cosine of a number greater than unity but less than 1 + 2 - 25, T.I.P. gives the programmer the benefit of the doubt, and assumes that the data was badly rounded off. If, on the other hand the data is quite impossible, the computer does not stop and involve a post-mortem and reference back to the programmer: it merely inserts a 'dash' in the corresponding result row. Any further operation on a 'dash' results in a 'dash' and so the final results may well be quite satisfactory apart from one dash in a particular row which the programmer may easily be able to explain.

The 'dash' technique has also been exploited to deal with discriminations. The operation r = 25 discriminates on the column of numbers specified by b , and where they are positive the corresponding numbers of column a are transferred to column c, and a 'dash' is placed in column c + 1; otherwise a 'dash' is placed in column c and the number is transferred to column c + 1. Columns c and c + 1 can then be operated upon independently by the remaining programme. Similarly columns can be merged by an r = 26 instruction.

To avoid any scaling worries, and yet to preserve the speed of fixed point working, all operations are carried out in block floating arithmetic, that is all the numbers in one column are stored to the same exponent, with the largest element filling the word and the exponent stored separately. Constants are stored as floating numbers. The number of elements in a column (usually a constant throughout a programme, but varying if finite difference operations or interpolation 'graphical' data are performed) is also stored with each column, as is the sum of all elements of the column. The latter is tested every time reference is made to the column. Up to 511 instructions can be stored in the machine at one time; in general this is more than enough, but more can be read in if necessary to overwrite part of the programme.

Generous facilities for programme testing are available, and the setting of specified keys on the machine will cause the machine to stop on a particular instruction, change it for another, or punch some (usually only the first) elements from all or specified columns. The latter limited punch out facility is of great use for the more involved errors.

The organisation of the programme is fairly obvious. The 128 'columns' are in fact 128 tracks on the drum, the 30 'rows' are 30 of the 32 minor cycles per track, the remaining two being used for the no. of rows, the number of binary places and the sum check. The high speed store, is used to contain as many of the subroutines for the various r functions as can be accommodated, and enough instructions to bring down any other selected r function. It also holds, of course the columns currently being operated upon, and the routines for disentangling the a, b, c, r instructions and fetching and storing columns.

The scheme thus gives (in common with Alphacode) the opportunity of using DEUCE without having heard about punched cards, delay lines, minor cycles or magnetic drums, and one can begin to programme for T.I.P. on the day one first hears of it.

7. THE GENERAL INTERPRETIVE PROGRAMME.

In some respects, the Tabular Interpretive Programme may be regarded as a

rather special case of the General Interpretive Programme, which was developed at the National Physical Laboratory four years ago, and which has proved invaluable at most DEUCE installations. The form of the instruction word - a, b, c, r for T.I.P. was taken from G.I.P. which does not however have a limited range of functions r. Self contained programmes, (which may incidentally be used independent of G.I.P.) which satisfy a few simple rules are called bricks, and any brick may be used as a function for the purposes of this scheme. There are at present almost 200 bricks in the library and not more than 63 may be used in one programme. The scope and flexibility of the scheme is therefore enormous, and this explains why so many DEUCE installations have used G.I.P. for more than half of their work.

G.I.P. is a programme pack which performs the following operations:

(i) It reads itself into tracks 235 to 255 of the drum.

(ii) It reads in a parameter card saying how many bricks are being used in the current programme.

(iii) It reads in the appropriate number of bricks, storing them from track 234 (in decreasing track number order.) As it reads and stores these bricks it notes where it has stored each one, so that it can subsequently bring it into the high speed store and obey it. It also makes a note of the first instruction in the brick.

(iv) The G.I.P., programme is then read in. This consists of a sequence of 'a, b, c, r' codewords, where r refers to the rth brick in the sequence just read in, and a, b, and c are parameters to be provided to that brick, a, b, and c are usually drum addresses, in which case they should be small enough not to interfere with the bricks which are stored at the higher numbered tracks.

(v) G.I.P. then obeys the various codewords, starting with the first and taking them in turn, except for jumps and discriminations. 17 values of r are set aside for jump, discrimination, modification and housekeeping instructions so that the order code is quite flexible. For such values of r, the numbers a, b, and c in the corresponding codewords usually refer to codeword numbers - this also being a feature of T.I.P. Discriminations thus take the form "Jump to codeword number b if the contents of codeword a are zero, otherwise jump to codeword number c". Such a codeword implies that one codeword (in this case number a) is being used as a counter rather than as a codeword to be obeyed. There are a variety of discriminations provided, as well as an unconditional jump. Others among these special values of r cater for automatic instruction modification (modify a, b, or c automatically after the codeword has been obeyed), and provide facilities for reading in new bricks and either writing them over bricks already in store and which are no longer needed, or obeying them directly in the high-speed store. The latter facility is particularly helpful when one is embarrassed for drum storage space and there are one or more bricks which are obeyed only once in a programme. These may as well be read and obeyed directly, once and for all, and so save the storage space which they would otherwise need on the drum.

To use G.I.P., therefore, a programmer does not have to possess any detailed knowledge of DEUCE programming, providing he can achieve his object by using existing bricks. He decides what bricks are required for a particular job, and having made the programme of codewords, the operator then provides DEUCE with G.I.P., followed by the various bricks, followed by the codewords. The programmer will have had to count up how many of the 256 tracks on the drum will be needed by G.I.P. (which accounts for 21) and all the bricks he needs for his particular operation. The remaining tracks, which are always the lowest track numbers, are available for his data, and it will be necessary to know how the various bricks being used expect to find and store their data. Since virtually all bricks are built to accept and store data in a standard manner, the programmer's problem is to apportion the data tracks correctly. There are, nevertheless, features which a DEUCE programmer may use, since any codeword, as an alternative to the standard a, b, c, r form, may be replaced by a normal DEUCE instruction. The interpretive programme will detect such an instruction (by the presence of a digit normally spare), and obey it as required. For instance, when one needs to read a single number into the machine, it might be easier to use the DEUCE instruction facility rather than to call a brick to perform such a trivial operation.

The programme provides generous testing facilities in the way expected of a conventional machine order code. There is the equivalent of a "stop key" which, when on, will cause the machine to stop on every G.I.P. instruction, and display it on lights before it is obeyed. Similarly, one can force G.I.P. to stop on a specific instruction on the keys - this facility being of use when an instruction

is being examined each time it is obeyed in a loop. There are also convenient facilities for making the machine accept an instruction to replace that which it is about to obey, and to restore control or take a post-mortem in the case of a brick not behaving as expected.

The majority of the bricks available for use with G.I.P. are for linear-algebra operations, and as a result there has arisen a widespread belief that the scheme is restricted to matrix operations. This is not so, although the scheme has been used extensively for linear-algebra and, in fact, many problems which do not, at first sight, appear to be linear-algebra problems, have been solved using standard matrix bricks.

In the early days, the majority of problems arising came from the aircraft industry and were presented as linear-algebra operations. The standard operation to calculate $A_{11} - A_{12} A_{22}^{-1} A_{12}'$ given A_{11}, A_{22} and A_{12} uses bricks to read, invert, multiply, transpose, subtract, and punch out matrices, and the complete G.I.P. instructions for performing this operation are:

Codeword	a	b	c	r		
0	-	-	32	7	Read A_{22} into track 32 onwards.	
1	32	0	-	8	Prepare for inversion.	These
2	-	-	-	9	Reduce.	perform
3	-	-	106	11	Back substitute and store (A_{22}^{-1})	the
			At track 106 onwards			inversion.
4	-	-	-	12	Check store used for reduction.	
5	-	-	146	7	Read A_{12} into track 146 onwards.	
6	146	106	106	1	Form $A_{12} A_{22}^{-1}$ in track 106 onwards.	
7	106	146	0	1	Form $A_{12} A_{22}^{-1} A_{12}'$ in track 0 onwards	
8	-	-	63	7	Read A_{11} into track 63 onwards.	
9	63	0	126	4	Form $A_{11} - A_{12} A_{22}^{-1} A_{12}$ in track 126 onwards.	
10	126	-	-	6	Punch out.	
11	-	-	0	33	Return to codeword 0 to repeat with other matrices.	

This programme will solve the problem for A_{22} of order less than or equal to 31 by 31 and for A_{11} less than or equal to 41 by 31. It uses 12 bricks, for the multiplication brick has three sections and counts three fold, and the inversion brick counts five fold. The bricks used are

		No. of Tracks.
1,2,3	Matrix Mult.	16
4,5	Subtract	9
6	Punch Matrix	3
7	Read Matrix.	3
8,9,10,11,12	5 sections to the invert matrix.	18
	Total:	48 tracks.

The 48 tracks of bricks (and 21 tracks for G.I.P. itself) leaves 187 tracks on the drum for the data, and the selection of the a, b, and c values in the above programme, have been chosen so as to make the best use of this available space. It will be noted that no reference is made at all to the size of the matrices. The scheme is so arranged that the dimensions of matrices are stored with the matrices themselves, and all bricks check for compatibility. In this way the programmer merely has to specify the first of the (consecutive)

drum tracks on which the matrix is stored. A consequence of this is that if a G.I.P. programme is made to cope with a given maximum size of matrices (the restriction being storage space on the drum) then the same programme, without any alteration, may be used for any matrices whose size does not exceed the given maximum. In nearly all bricks the numbers are stored in block-floating form, that is with all elements to the same number of binary places, and with the largest element shifted up to fill the storage register. This method successfully combines the speed and storage economy of fixed-point working with the flexibility and accuracy of floating-point operation.

More recently G.I.P. has been used for a wide range of other problems, not so obviously presented as linear-algebra. This has been facilitated by the case with which a G.I.P. programme can be constructed, and by the fact that even if all the required bricks do not exist, most will be available , and one or two special purpose bricks can be used. For instance much statistical work is done on DEUCE by using G.I.P. - particularly in conjunction with a few special purpose statistical bricks. The Simplex and Transportation programmes also use G.I.P. - again with a few special bricks written for these purposes.

Since each function being carried out under the control of G.I.P., may be a complete DEUCE programme in itself, which will be obeyed at optimum speed and which may be operating on a large amount of data, the time spent in interpreting the codeword and fetching the brick may not be a significant amount. In fact it is about half a second. G.I.P. therefore shows up to its best advantage when it is required to operate upon large quantities of data in parallel, as typified by sizeable matrix operations. It is least efficient whenever the operations are trivial or only affect a small amount of data. It is to cope adequately with calculations on small amounts of data that T.I.P. and Alphacode were developed.

Particular features of standard bricks are the elaborate checks on arithmetic accuracy and the reliability of the operator or programmer. This standard of checking was set with the first batch of bricks, and has been followed by later programmes. All matrices stored on the magnetic drum have the grand sum of all the elements stored with them and, whenever any bricks makes reference to the matrix, the grand sum is checked. This is important, not so much as an arithmetical check, but as a check that the programmer has not unwittingly overwritten the "tail end" of a matrix (which will in general be stored on a series of consecutive tracks). Similarly checks on the compatibility of matrices in, for instance, addition or multiplication are incorporated, and such checks have proved themselves very worth while - particularly in complicated operations where a programmer may mistake the dimensions of his matrices.

G.I.P. is therefore an ideal scheme for all calculations needing bulk calculations in parallel on relatively large amounts of data.

7. OTHER SCHEMES.

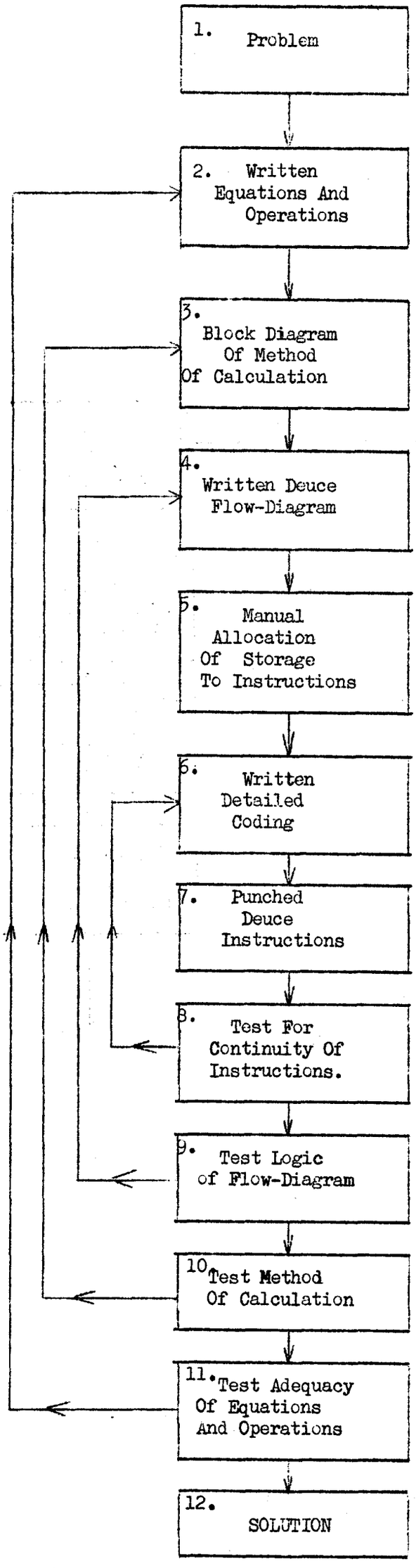
Other schemes, including Alphacode which is complementary to G.I.P., and T.I.P., in its facilities, are discussed in a comparison paper.

APPENDIX.

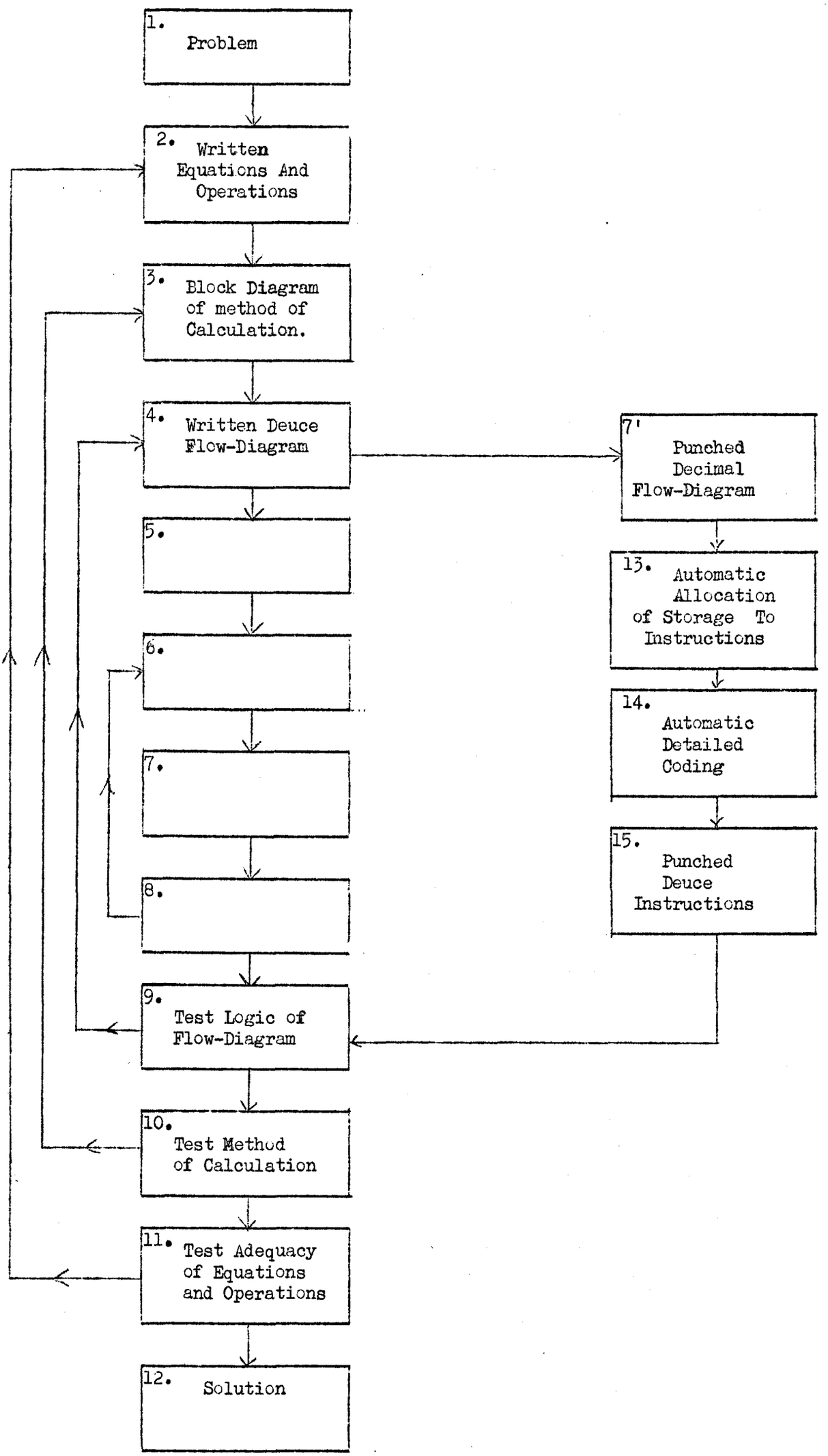
Specification of three Hypothetical Machines as seen to the Programmer of one of three Interpretive Schemes.

		G.I.P.	T.I.P.	Alphacode.
1.	Form of instruction word	a, b, c, r	a, b, c, r	$x_1 = x_2$ function x_3
		(all three address plus function, r being function in G.I.P. and T.I.P.)		
2.	No. of different kinds of machine order.	A library of about 200 functions (r) is available from which the programmer selects those which he wants.	31	64
3.	Kinds of order.	All types - but usually for dealing with bulk data as typified by matrix operations.	A set of orders suitable for scientific /engineering work.	A large set of orders suitable for scientific /engineering work.
		(In all schemes new functions can be inserted in place of existing ones not required).		
4.	Housekeeping Instruction	16 special values of r set aside for discrim. Inst. Mod. etc.	Automatic Instruction Modification and Simple Counting.	
5.	Storage Capacity.	Up to 7000	4196	2500
6.	Knowledge of DEUCE.	Some, particularly of drum.	None.	None.
7.	Arithmetic.	Usually Block Floating.	Block floating.	Fully floating.
8.	No. of orders held in machine at a time.	Usually 96 (more if necessary).	512	Up to 512 (More if necessary).

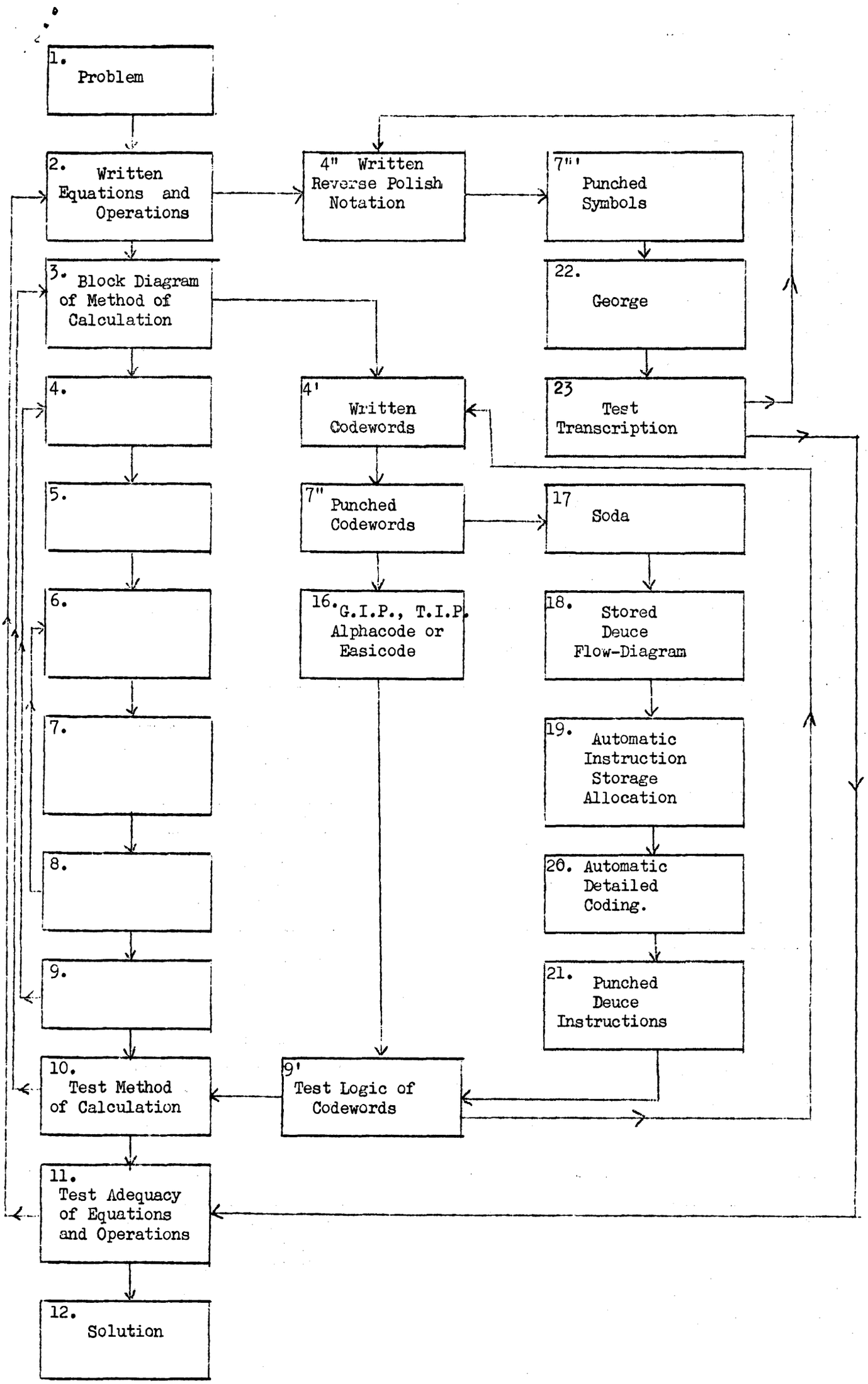
	G.I.P.	T.I.P.	Alphacode.
9. Interpretation time.	$\frac{1}{2}$ second per order.	.7 second is average interpretation and execution time for a full column.	.03 to .5 sec. depends on the function.
10. Type of calculation suitable for.	Bulk calculations in Parallel.	Tabular Work.	Long sequence of calculation on a single variable.
11. Special features.	Flexibility - DEUCE Instruction can be inserted.		Pulling File of instruction words.



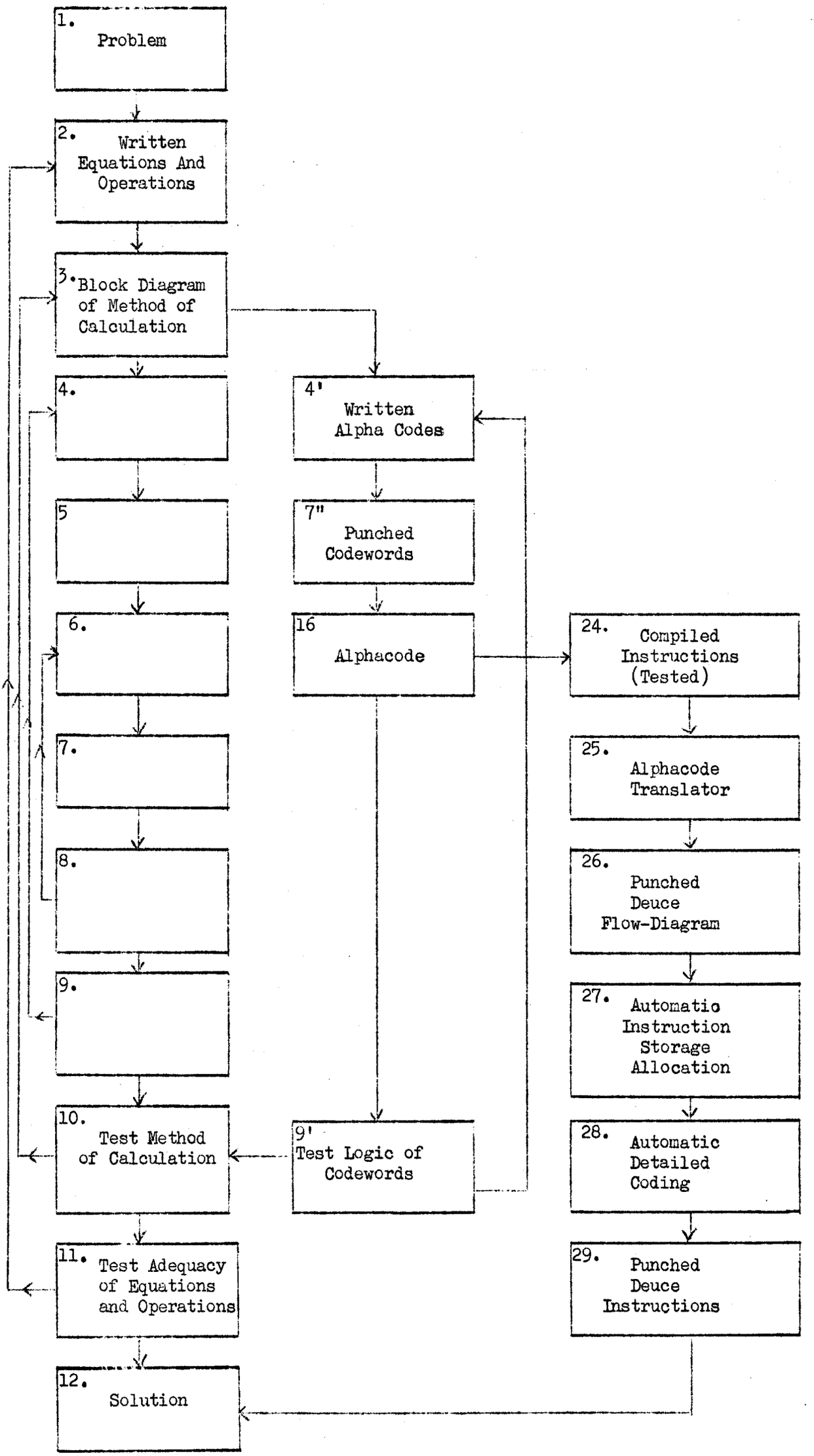
ORTHODOX DEUCE PROGRAMMING



Existing Schemes using Written DEUCE Flow-Diagram.



Existing Schemes not using Written DEUCE Flow-Diagram.



A Development in Hand.