First addition

SODA MANUAL OF OPERATION.

By

C.G. Bell and R.C. Brigham

Department of Electrical Engineering

N.S.W. University of Technology

Sydney, Australia

To George,

When SODA gave us trouble,
and played us pranks,
you often had the saving idea,
for which we give you thanks,
thanks also for your encouragement
all along,
you built us up when everyone else
said we were wrong.

Bob Brigham
Gordon Bell

I N D E X.

INTRODUCTION OF SODA.

The basic Deuce machine is a fast electronic digital computer capable of performing thousands of arithmetical and logical operations every second. Its efficient utilization often requires the experience of weeks. SODA is a scheme which uses the computer to perform much of the routine work associated with making Deuce function, as well as providing a system which is certainly easier to learn than is the basic one.

Before proceeding further, it seems wise to define a few terms associated with the computihg field. A computer solves a problem by executing a series of instructions associated with the problem in a predefined sequence. These instructions may cause arithmetical or logical operations to be performed, or the transfer of information within the computer or between it and the outside world, or a selection of alternative instruction sequences depending upon the results of preceding calculations. The sequence is termed a programme and the people who arrange it are programmers. It is in this field of programming that SODA attempts to relieve the human efforts at the expense of a certain percentage of efficiency in the final machine programme.

As an example of what is meant by programming, let us examine the evaluation of the following polynomial:

$$a_2 x^2 + a_1 x + a_0$$

It can be rewritten as:

$$(a_2 x + a_1) x + a_0$$

A programme to evaluate it might therefore take on the following form:

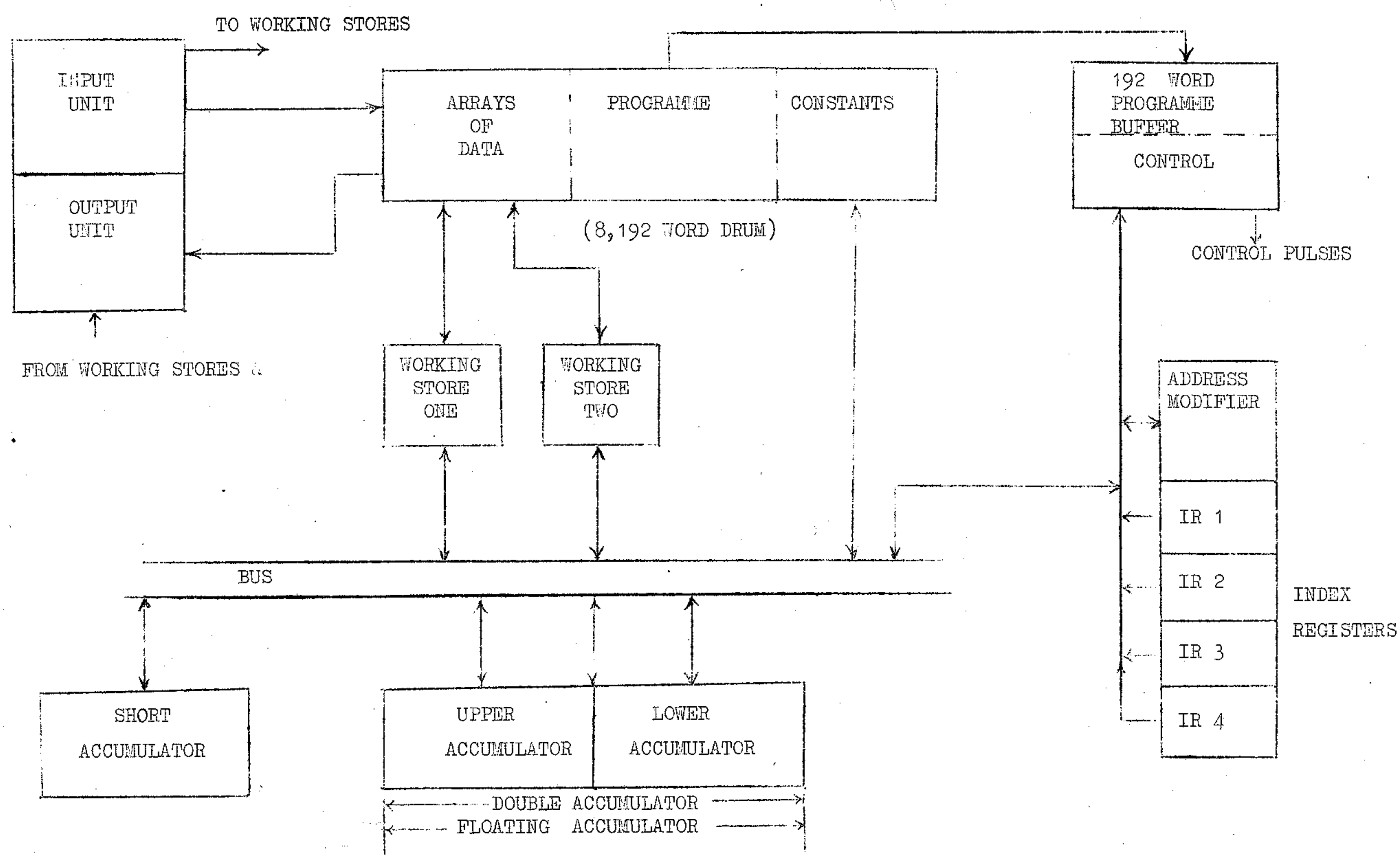
1. Multiply a_2 by x
2. Add a_1 to the result
3. Multiply the last result by x.
4. Add a_0 to this result to give the final answer.

All programming reduces to elementary steps very much like those illustrated by the above four, and it is the programmer's job to properly sequence them. It is not intended that this manual be a text in programming. However, several examples, using the SODA instructions, are given in Appendix VI, and it is hoped that some techniques will become apparent by a study of them.

THE SODA MACHINE.

SODA is a programme, called a Translation Programme, which transforms the basic Deuce machine, as far as the user is concerned, into a computer with properties entirely different from Deuce. It takes programmes written in SODA instructions and translates them to different programmes in the Deuce language, the latter being the one which finally carries out the desired computation. The machine to which Deuce is transformed by SODA is pictured in Figure 1. It is necessary that the SODA user thoroughly understands the construction of this pseudo-computer.

In order to work at electronic speeds, it is necessary that all information, including data and programme, be stored within the computer itself. They are, in this case, placed on a magnetic drum which is capable of retaining information over an indefinite period of time. The drum is capable of holding 8,192 words, (where a "word" is either a number or a Deuce instruction,) arranged into 256 tracks, numbered 0 - 255, of 32 words each. Space is divided between the three types of quantities stated in the diagram, but the dividing lines are not rigid. First on the drum are the constants (including any temporary storage required) and there must be less than eight tracks of these. Next comes the programme. It can extend as far as necessary, providing it does not attempt to overlap any of the data area which has been previously reserved. An error signal is given if the translation results in this. Data differs from constants only in name, but it can usually be taken to refer to an ordered array of which a repetitive calculation pertains to different elements each cycle. One would expect the programme to work with varyihg data, while the list of constants remainsthe same.



LAYOUT OF THE SODA MACHINE

FIGURE 1

It is not possible to work with data directly on the drum. It must first be placed in one of the two working storages, and if other data is to be stored on the drum, it must be written there from a working storage. For most purposes these storages can be considered as independent memory units of indefinite size, always capable of holding at one time all of the elements in a given array. Other usages require that the programmer realize that each working storage actually contains only thirty-two positions, and that the correct elements are automatically constrained to be in this group of thirty-two at the proper time.

All of the arithmetical and logical operations of this machine take place in one of the three Accumulators, the Short, the Lower, and the Upper. One of these normally contains one of the operands required by the instruction, and then holds the result of the operation. In addition, the Upper and the Lower are combined by some instructions to form an extended length accumulator known as the Double, which permits a limited amount of double length arithmetic operations.

The bus permits the flow of information between various units of the computer.

The function of the control unit is to properly interpret the SODA instructions in the correct sequence. Associated with it are four units known as index registers, and their contents can be used to modify an instruction so that different elements in a working storage can be selected using the same instruction. This is quite desirable in cyclic operations. Indeed, it is only by using these index registers that the programmer can reach the contents of the working storage. This will be explained in detail at a later point. The input unit consists of a Hollerith card reader and the output is a Hollerith card punch unit. These provide means for placing information into the computer and for punching out results.

INFORMATION REPRESENTATION IN SODA AND DEUCE.

Much of this section is taken almost verbatim from notes by Mr. R.G. Smart, Senior Lecturer in charge of the Computing Laboratory at the N.S.W. University of Technology, Sydney, Australia.

Numbers are dealt with in the machine in standard binary form. Binary numbers are similar to decimal numbers, except that the base is two instead of ten. In decimal, each digit position can take on ten different values 0, 1, 2, ...9, and the digit position relative to the decimal point indicates the power of ten by which the digit in that position must be multiplied to give the number being represented, i.e. 365.25 is

$$3 \times 10^2 \quad + \quad 6 \times 10^1 \quad + \quad 5 \times 10^0 \quad + 2 \times 10^{-1} \quad + \quad 5 \times 10^{-2}$$

In an exactly similar manner, each "binary" position can have two values, 0 and 1, and the digit position relative to the "binary" point, indicates the power of two by which the digit in that position must be multiplied to give the number being represented, i.e.

$$101.01 = 1 \times 2^2 + 0 + 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} \quad = \quad 5\tfrac{1}{4}$$

To convert a binary number to a decimal number, it is necessary to add up the powers of two corresponding with the places occupied by a one in the number, e.g.

$$10110.0 \;=\; 2^4 + 2^2 \times \; 2^1 \;=\; 22$$

To convert a decimal number to binary, find the largest power of two which does not exceed the decimal number, subtract this power of two, record a one in the corresponding digit position, and find the largest power of two which does not exceed the remainder, etc., until the remainder is zero, or sufficient binary places have been found, e.g. to find the binary of 26.5.

Now $(2^5 = 32) > 26.5 \geqslant (2^4 = 16)$ $\therefore$ record $2^4 = 10000.00$

$(26.5 - 16) = 10.5$

and $(2^4 = 16) > 10.5 \geqslant (2^3 = 8)$ $\therefore$ record $2^3 = 1000.00$

$(10.5 - 8) = 2.5$

and $(2^2 = 4) > 2.5 \geqslant (2^1 = 2)$ $\therefore$ record $2^1 = 10.00$

$(2.5 - 2) = 0.5$

and $(2^0 = 1) > 0.5 \geqslant (2^{-1} = 0.5)$ $\therefore$ record $2^{-1} = 0.10$

and the remainder is zero.

$\therefore$ 26.5 = 11010.10 in binary

Other conversion processes are available for use with desk machines. It should be noted that the "binary point" in a binary number has exactly the same meaning as the decimal point in a decimal number, that is, it divides the integral from the fractional part of the number. A binary number with say five "binary places", has digits in five binary positions to the fractional side of the binary point.

It is of considerable advantage to the programmer if provision is made within the computer to indicate automatically the sign of every number as well as its magnitude. Furthermore, if negative numbers are represented as complements, the design of the arithmetic units is simplified. Deuce, in common with many other computers, thus uses the most significant digit to indicate the sign of the number, and manipulates negative numbers as their complements with respect to two. This is explained below.

For convenience of reference the thirty-two digit positions of the computer word, in order of decreasing significance are referred to as

P_{32}, P_{31}, P_{30}.........P_3, P_2, P_1.

(These actually appear on the console of Deuce in the order P_1, P_2, P_3, ..., P_{30}, P_{31}, P_{32}.) We can also speak about P_{33}, which is the next position more significant than P_{32}. In single length arithmetic, the P_{33} is outside the word length and is lost. With double length arithmetic however, positions up to P_{64} are available.

In Deuce, P_{32} is the sign digit, and only 31 digits are available to indicate magnitude. This is sufficient for the equivalent of nine decimal digit accuracy, and integers up to $2^{31} - 1$, i.e. over 2,000,000,000 can be represented by the thirty-one digits. If a number is positive, it is represented within the machine in its normal binary form. The number of binary places is not recorded by the computer directly, but is controlled by the programmer, and is chosen so that the numbers at any stage of the calculation do not exceed the thirty-one digit capacity, and at the same time, as many binary places as necessary have been provided to give adequate accuracy. If the binary point is placed between P_{32} and P_{31}, we have thirty-one binary places and the largest number we can represent is $(1 - 2^{-31})$, so that with thirty-one binary places the positive numbers n cover the range $0 \leqslant +n \quad +1$.

Negative numbers are represented by their complement with respect to two. Thus, with thirty-one binary places, the negative number $-n$, is represented within the machine as $N = (2-n)$. This convention leads to a negative number range of $0 > -n \geqslant -1$ from which it follows that all negative numbers have a one in the P_{32} position, while all positives have a zero there.

For example: (remember we are considering the binary point to lie between P_{32} and P_{31}, i.e. thirty-one binary places)

$+\frac{1}{4}$ is represented as 0.0100..........0

$+\frac{3}{4}$ is represented as 0.1100..............0

$(+1-2^{-31})$ is represented as 0.1111..............1

which is the largest positive number which can be represented to thirty-one binary places. It should be quite clear that all positive numbers have a zero in P_{32} position, while all negative numbers have a one in P_{32} position resulting from the decision to represent negative numbers by their complements with respect to two and the restriction of the number ranges (for thirty-one binary places) to $0 \leqslant +n < +1$, $0 \geqslant -n \geqslant -1$.

Had we chosen to have no binary places, i.e. place the binary point below P_1, all the above binary numbers would represent numbers 2^{31} times as large. The sign convention would then become: represent the negative numbers $-n$ as $N = (2^{32} - n)$, and restrict positive numbers to the range: $0 \leqslant n \leqslant (2^{31} - 1)$ and negative numbers to the range: $0 \geqslant -n \geqslant -2^{31}$.

Thus:

0000 ....01 represents $+1$

0111 ....11 represents $+(2^{31} - 1)$

while:

100 ....00 = $(2^{32} - 2^{31})$ represents -2^{31}

111 ....11 = $(2^{32} - 1)$ represents -1

To summarise then, positive numbers are represented by the digits P_1 to P_{31} inclusive. Negative numbers are represented as a one at P_{33} minus the magnitude of the negative number. The largest positive number is represented by P_1 to P_{31} inclusive being ones, and P_{32} zero, and the smallest by P_1 to P_{32} inclusive being zeros, which always represents zero. The largest (algebraically) negative number is represented by P_1 to P_{32} inclusive being ones and the smallest by P_1 to P_{31} inclusive being zeros, and P_{32} being one.

It will be appreciated that if a number becomes too large in the course of a calculation and exceeds the thirty-one digit capacity, the result will generally be incorrect. For example, working to thirty-one binary places

$+\frac{1}{2}$ = 0.100 0

$+\frac{3}{4}$ = 0.110 0

therefore $(\frac{1}{2} + \frac{3}{4})$ = 1.010..........0 according to the computer, which in the signed convention is $(1\frac{1}{4} - 2)$ = "$-\frac{3}{4}$", instead of the required answer of $1\frac{1}{4}$. This kind of error must be guarded against.

It is useful to note that the sign of a binary number may be reversed, within the sign convention adopted for Deuce, regardless of the position of the binary point, by writing down the "not" of the number and adding a P_1. By the "not" of a binary number N, written as $\sim N$, is meant the replacing of each digit by a nought or a one, whichever was not present in the original, i.e.

if N = 1101011001

$\sim$ N = 0010100110

Thus, given N = 21 to zero binary places,

N = 000 010101

$\sim$ N = 111........101010

and $-N = (\sim N + 1)$ = 111........101011 = $(2^{32} - 21)$ which represents "-21" in the sign convention.

Or given $-n = -\frac{3}{4}$ to thirty-one binary places, this is stored as,

$$(2 - n) = (N) = 1.0100......00$$

$$\sim (N) = 0.1011......11.$$

$$+ N = \sim(N) + 1 = 0.1100.......00 = + \tfrac{3}{4}.$$

The process works either way and is independent of the position of the binary point. A proof is given for interest.

$N + (\sim N) =$ 111111........11, i.e. 32 ones

whether N is positive or negative, and

$N + (\sim N) + P_1 =$ 100000.......00, i.e. P_{33} and 32 zeros.

If we are working with integers, $P_{33} = 2^{32}$ and $P_1 = 1$

Therefore, $(\sim N) + 1 = 2^{32} - N$ which represents $-N$ if N is positive.

If N represents a negative number $-n$, then:- $N = 2^{32} - n$

Therefore, $(\sim N) + 1 = 2^{32} - (2^{32} - n) = +n$, the positive number of which N is the negative representation within the machine. A similar proof can be applied to the case where $P_{33} = 2$ and $P_1 = 2^{-31}$.

Appendix III includes lists of binary numbers and their decimal equivalents.

The contents of the accumulators, index registers, and various stores may be viewed on the two scopes on the console. The representation is in the backwards binary notation mentioned above where the digits of a word are viewed, left to right, in the order P_1, P_2, P_3, ...P_{30}, P_{31}, P_{32}.

It has been mentioned that double length manipulation of numbers is possible and that a double length number occupies two successive words of storage. These numbers are made up of the sixty-four positions P_1 to P_{64}. Of these, P_{64} is the single sign indication and P_1 to P_{63} contain magnitude of the number. When double length constants or data are placed in the machine in binary, positions P_1 to P_{32} must be placed on an even row of the input card i.e. rows Y, 0, 2, 4, 6, 8 and the corresponding P_{33} to P_{64} must be placed on the succeeding odd row, i.e. rows X, 1, 3, 5, 7, 9. When the Double Accumulator is used, the thirty-two least significant binary digits appear in the positions usually occupied by the Lower Accumulator, while the thirty-two most significant (including the sign) appear in what is normally the Upper Accumulator.

Floating point arithmetic is also possible with SODA. These numbers are of the form $a \times 2^b$ where a and b are two normal single length numbers of the type described above. 'a' is to thirty binary places and lies in the region $\frac{1}{2} \leq a < 1$ or $-\frac{1}{2} > a \geq -1$. 'b' is an integer to zero binary places and should be kept to less than 2^{15}, although no error signal is given if it should exceed this limit. This is approximately equal to 32,000 which, in practice, represents no restriction. Calculation might result in error if 'b' passes the above limit. A floating point number of this form is said to be in standard floating binary (s.f.b.) form. A floating point number is zero only if 'a' and 'b' are both zero. Use of floating point arithmetic eliminates the worrying about the binary point which is inherent in fixed point arithmetic. When reading a floating point number into the machine in binary the exponent 'b' is placed on an even row of a card and the associated mantissa 'a' is placed on the succeeding odd row.

The floating point operations place a floating number in the Lower and Upper Accumulators, 'b' being in the former and 'a' in the latter.

The SODA language employs alphabetic symbols which must be given a binary representation during the translation process. Appendix III contains the binary coding and punched card representation of the various characters available to the programmer.

CARD PUNCHING FOR SODA.

Following is a description of the punching required for SODA instructions. Each instruction is made up of four basic parts:- the location of the instruction itself, the operation that the instruction is to carry out, the location of the operand or number which is being operated upon, and the location of the next instruction to be obeyed in the programme. One SODA instruction is punched per card.

A. Column 1 - Control.

Column 1 is the control column. The control numbers are punched in decimal (1-5) and the use of these control instructions will be discussed in the section "Control Cards".

B. Columns 2-6 - Location of This Instruction.

These five columns contain the location or address of this instruction. This address is usually left blank. If it is blank, it is assumed that this instruction follows the preceding one, and is never jumped to from somewhere else. When a group of operations follow each other, the programme will proceed along the sequence, translating the instructions, in order, as they are read in.

If not blank, this address must be symbolic in nature. A symbolic address in SODA is composed of up to give alphabetic and numeric symbols. At least one of the symbols must be alphabetic. If a numeric address is given in the location of this instruction address, the machine assumes an error has occurred, and the translation will stop until an alphabetic address is given.

A symbolic address must be assigned to the first instruction fed into the machine. Otherwise the translator will not know of this first instruction, or, if it is the first instruction to be executed in the translated programme, there will be no means of identification that it is the first instruction. A symbolic address must also be given whenever control jumps (*), in any manner, to the associated instruction.

C. Columns 7-9 - Operation.

These three columns contain the name of the operation. It is three alphabetic characters in length, and contains strong mnemonic references to aid the programmer in both learning and using the operation code. The orders permit various manipulations in single length, double length, and floating point arithmetic. A list of the instructions is given in Appendix I, and a detailed account of their function is given in the section entitled "The SODA Instructions".

These columns must not be left blank, and must be punched correctly to select the proper operation, either in the standard order code, or from the group of orders placed in the machine for use as standard Deuce subroutines.

D. Columns 10-14 - Location of the Operand.

Columns 10-14 specify the address of the operand. These columns may be symbolic, numeric or blank. The operand is the actual number specified by the operand address.

The symbolic address either refers to a constant, which has been previously defined, or to an array of data, initially defined, which is placed in one of the two working stores. In the latter case, the element of the array is selected by the contents of one of the index registers. In addition to these conditions, a symbolic address, when used with one of the conditional jump instructions, may specify the location of the point to which the jump is addressed.

---

* a conditional jump instruction is one which enables the programme to branch to another point if certain conditions hold.

If the operand location is numeric, it usually refers to either the spaces in working stores, or various arithmetic units of SODA. The numeric values are punched in standard decimal and must be 0-95 or 100-107. 0-31 refers to the physical thirty-two positions of working store one, 32-63 to those of working store two, and 64-95 to spaces for temporary or constants store. Address 100 is the Short Accumulator, 101 the Lower Accumulator, 102 the Upper Accumulator, 103 = $P_1 = 2^0$, 104 = $P_{17} = 2^{16}$, 105 = P_{32}, 106 = 0, and 107 = $-P_1 = -2^0$ = all ones.

A number in these card columns may give the number of positions for shifts in the shifting instructions. Too large a number in this address for each shift instruction creates an error. The shifts must be less than 33 in length for single length numbers, or less than 17 for double length numbers.

If a numeric address outside the above limits is given, an error has occurred. This may be fixed externally on the console.

When a blank occurs in the operand location, it is either unused in the operation or this instruction is a jump instruction. A blank in these columns, when jump instructions are used, means the jump is to the next operation in the sequence following the actual jump instruction.

In addition to the above modes, the operand location may be used in a special manner when a large group of elements are needed which do not conform to a pattern suitable for step by step indexing. A programme of this type is one which uses a large number of elements with each being operated on only infrequently in a non-cyclic or non-repetitive nature.

In this mode, an array is identified by giving it a name in the form of a single alphabetic character, e.g. Y, in the leftmost column of the five available. When an instruction is written down to operate on an element, the operand location is written as Y $|\leftarrow j \rightarrow|$. When instructions operate in this mode, an index register (5-8) must be specified. Index registers 5, 6, 7 or 8 overwrites respectively the contents of registers 1, 2, 3, or 4 to select this jth element. If the instruction is a floating point operation, the calculation 2j + 1 is automatically made, and the corresponding position in the array selected. That is, the element selected is that one found by counting back j (or 2j + 1 in floating point operations) words from the last word in the array. A description of the functioning of the index registers is given below.

E. Columns 15-19 - Location of the Next Instruction.

These columns specify the location or address of the next instruction. This address is usually left blank when a sequence of operations is being listed. A blank assumes that the next instruction to be obeyed will follow on the succeeding card, in which case columns 2-6 of the next card must be blank.

If a symbolic address is given here, the next instruction of the sequence must also have a symbolic address in the "Location of This Instruction" address, unless the present instruction is a jump instruction. For jump instructions, either the Location of Operand, or Location of the Next Instruction or both, must contain symbolic addresses. A blank in either operand or next instruction address, means the programme will take one of the jump paths to be the next instruction on the following card of the programme being translated.

If a symbolic address is given as a location of next instruction, this same address must appear as the location of this instruction (columns 2-6) of some other SODA instruction (usually the next one).

F. Column 20 - Number of the Index Register.

This column punched 1 to 4 or 5 to 8 specifies which of the four index registers is to be used in an operation. The number is punched in decimal and, if the use of index registers is not desired for the instruction, this column remains blank. Index registers may be used only when the operand has a symbolic address referring to a data array defined by an arrays control card, when the operand is numeric, when the instruction calls for a shift, or under the special condition described in the operand address section above.

In the first use, the operand can refer only to an array which has been placed into one of the two working storages by means of a RWO or RWT order (see below). The element of the array which is involved in the operation is selected by counting back from the last element in the array a number of elements equal to the number in the specified index register. Thus, the last element is chosen if the index register contains a zero, the next to last if it contains a one, and so forth.

Where index registers 5, 6, 7 and 8 are referred to (actually corresponding to registers 1, 2, 3 and 4, respectively), the index register is set to select the jth element of the array (or the 2j + 1 in the case of floating numbers) which has been given a single alphabetic address, followed by the 4 digit numeric address of the element within its array. This mode was previously discussed under the location of the operand.

Index registers can also be employed in conjunction with numeric addresses. In this event, the element selected with the index register set at zero is the one in working storage one, working storage two, or constant storage indicated by the numeric address. Other elements are reached by counting back from this element a number equal to that in the specified index register. Care must be taken that the number in the index register never becomes so large that the programme attempts to select an element before the first position of the physical thirty-two word section in which the given numeric address is contained, as this will result in improper operation.

Thus, if specifying an element in working storage two, the programmer must not attempt to reach an element before location 32, which is in working store one, or after location 63, which would be in constant storage. Thus, if the numeric address 93 is given along with a specification of index register 3, indexing would occur in the constant storage, and the element in location 93 is selected if index register 3 contains a 0, 92 if it contains a 1, etc. The maximum permissible value for the index register would be 29, which would cause selection of the element in location 64. A larger number would result in erroneous operation.

Index registers may also be employed with shift instructions with the exception of the two pertaining to shifting the Double Accumulator. In this event, the operand of the instruction is ignored, and the length of the shift is indicated by the number in the specified index register. Incorrect operation will result if the number in the index register is greater than thirty-two. Zero in the index register is used to specify a shift of 32 places.

When using an index register for indexing purposes (it can also be used for straight counting or temporary storage of any results), the binary numbers in them must be times 2^{16}, that is integers with 16 binary places.

G. <u>Columns 21-25 - Decrement.</u>

The decrement is contained in these five columns. A decrement may be symbolic, in which case its numeric value is found in the corresponding constant address, or it may be numeric. If it is a symbolic address, it must have at least one alphabetic character in columns 22-25. Otherwise, it is exactly like all symbolic SODA addresses. If the decrement is numeric, it may be either positive or negative. A non-zero term is punched in column 21 to indicate a negative decrement. The four decimal digits of the number are punched in columns 22-25. The decrement is employed in only 3 instructions (JIX, JXE and JXH) and its use is explained under their descriptions.

H. <u>Column 26 - Special Forward Jump Indication.</u>

Column 26 must be given a non-zero indication if columns 15-19 specify a previously unused symbolic address for the next instruction, and if columns 2-6 of the next card do not specify the same symbol as the location of that instruction. This rule does not apply to conditional jump instructions.

THE SODA INSTRUCTIONS.

This section describes in detail the functions executed by all of the SODA instructions. A thorough understanding of the preceding material is assumed. Remember, the "Operand" is the contents of the storage location referred to by the operand address (card cols. 10-14), and the number in the index register, if specified. All instructions are indexable except where otherwise stated. They are grouped so as to provide easy reference, and a complete list is given in Appendix I. The number in parenthesis is the identification for the trace. See below.

A. Instructions Pertaining to the Short Accumulator.

C A S - Clear and Add to the Short (1). The operand is placed into the Short Accumulator. (When a number is transferred from a store or an accumulator, a copy of that number remains intact in its original store or accumulator.)

C S S - Clear and Subtract from the Short (2). The negative of the operand is placed in the Short Accumulator.

A D S - ADd to Short (3). The operand is added to the word in the Short Accumulator, the result remaining in the Short Accumulator. Incorrect sums result if the capacity of the accumulator is exceeded. This is known as overflow.

S B S - SuBtract from the Short (4). The operand is subtracted from the word in the Short Accumulator, the result remaining in the Short Accumulator. Overflow may result.

S T S - STore the Short (5). The contents of the Short Accumulator replace the word indicated by the operand address.

S S L - Shift the Short to the Left (6). The contents of the Short Accumulator are shifted to the left the number of positions indicated by the operand address. If this is zero, a shift of 32 position results. This operand address is always numeric, with the least significant digit being in the rightmost column allotted to this address. The largest permissible shift is 32 positions, and demands for longer ones may result in an error in the translation process. Zeros are inserted at the least significant end of the accumulator and digits shifted off the high order end are lost. This corresponds to a multiplication by a power of 2 if no information is lost off the most significant end. All shift instructions may be indexed. In this event the operand address is ignored and the number of places shifted is equal to the number in the specified index register, which must not be greater than 32. It is cautioned that if the specified index register contains zero, a shift of 32 results.

S S R - Shift the Short to the Right (7). The contents of the Short Accumulator are shifted to the right the number of positions indicated by the numeric operand address. If the operand is zero, a shift of 32 positions results. This address must not be greater than 32. Digits shifted off the least significant end are lost. The sign digit is unaffected and the results remain correctly signed. This corresponds to a division by a power of 2. See SSL for the meaning of an indexed shift instruction.

L A S - Logical And with the Short (8). The operand is compared digit by digit with the word in the Short Accumulator. If two corresponding digits are a one, a one is placed in the same position of the Short Accumulator. If either or both are zeros, a zero is inserted in that bit position of the Short Accumulator.

L O S - Logical Or with the Short (9). The operand is compared digit by digit with the word in the Short Accumulator. If two corresponding digits are a zero, a zero is placed in the same position of the Short Accumulator. If either or both are ones, a one is inserted in that bit position of the Short Accumulator.

L N S - Logical Non-equivalence with the Short (10). The operand is compared digit by digit with the word in the Short Accumulator. If two corresponding digits are of different values (one a 1 and the other an 0), a one is placed in the same position of the Short Accumulator. If both have the same value (both an 0 or both a 1), a zero is inserted in that position of the Short Accumulator.

B. Instructions Pertaining to the Lower Accumulator.

The following instructions are completely analogous to those already described for the Short Accumulator. The letters in parenthesis refer to the associated Short Accumulator instruction, and the descriptions given under them are applicable if they are taken to pertain to the Lower Accumulator.

C A L - Clear and Add to the Lower (CAS) (11).

C S L - Clear and Subtract from the Lower (CSS) (12).

A D L - ADd to the Lower (ADS) (13).

S B L - SuBtract from the Lower (SBS) (14).

S T L - STore the Lower (STS) (15).

S L L - Shift the Lower to the Left (SSL) (16).

S L R - Shift the Lower to the Right (SSR) (17).

L A L - Logical And with the Lower (LAS) (18).

L O L - Logical Or with the Lower (LOS) (19).

L N L - Logical Non-equivalence with the Lower (LNS) (20).

C. Instructions Pertaining to the Upper Accumulator.

The first ten instructions below are completely analogous to those already described for the Short Accumulator. The letters in parenthesis refer to the associated Short Accumulator instruction, and the descriptions given under them are applicable if they are taken to pertain to the Upper Accumulator.

C A U - Clear and Add to the Upper (CAS) (21).

C S U - Clear and Subtract from the Upper (CSS) (22).

A D U - ADd to the Upper (ADS) (23).

S B U - SuBtract from the Upper (SBS) (24).

S T U - STore the Upper (STS) (25).

S U L - Shift the Upper to the Left (SSL) (26).

S U R - Shift the Upper to the Right (SSR) (27

L A U - Logical And with Upper (LAS) (28)

L O U - Logical Or with the Upper (LOS) (29)

L N U - Logical Non-equivalence with the Upper (LNS) (30)

M P Y - MultiPlY (31). The word in the Upper Accumulator is multiplied by the operand. The product is correctly signed and may be 63 digits in length. The 32 least significant digits of the product appear in the Lower Accumulator, which has no sign connected with it. The 31 most significant digits of the product are placed in the 31 least significant bit positions of the Upper Accumulator. The sign of the entire number appears in the normal sign position of the Upper Accumulator. The product can be considered as a double length number residing in the Double Accumulator. The binary point of the product is located so that the resulting number of binary places is equal to the sum of those in the original two factors.

D I V - D I Vide.(32). The word in the Upper Accumulator is divided by the operand. The quotient is correctly signed, being 31 digits in length, and is placed in the Lower Accumulator. Certain relationships must be thoroughly understood before using this instruction. Let A represent the dividend (the word in the Upper Accumulator), B the divisor (the word indicated by the operand address), Q the quotient, A, B, Q, R must be within the normal sign convention for single length numbers. For division to proceed correctly $|A| < |B| \leq 2^{31}$. If A is negative and B positive, this expression becomes $|A| \leq |B| < 2^{31}$. The alarm will sound if this condition is not met. The number of places to the right of the binary point in the quotient is equal to the number in the dividend plus 31 minus the number in the divisor. A modified form of the remainder, r, resides in the Upper Accumulator. The true remainder, R, can be obtained by the following calculation: $R=\frac{1}{2}(\frac{1}{2}r + B)$.

D. Instructions Pertaining to the Double Accumulator.

When an operand address is given specifying a double length it refers to the least significant half of the number.

C A D - Clear and Add to the Double (33). The double length operand is placed into the Double Accumulator.

C S D - Clear and Subtract from the Double (34). The negative of the double length word indicated by the operand is placed into the Double Accumulator.

A D D - ADd to the Double (35). The double length operand is added to the double length word in the Double Accumulator, the result remaining in the Double Accumulator. Overflow may result.

S B D - SuBtract from the Double (36). The double length operand is subtracted from the double length word in the Double Accumulator, the result remaining in the Double Accumulator. Overflow may result.

A S D - Add a Single length word to the Double (69). Occasionally it is advantageous to add or subtract a single length word to the least significant half of the double length word in the Double Accumulator. The single length word indicated by the operand is added to the double length word in the Double Accumulator, the result remaining in the Double Accumulator. Overflow may result.

S S D - Subtract a Single length word from the Double (70). The single length word indicated by the operand is added to the double length word in the Double Accumulator. Overflow may result.

S T D - STore the Double (37). The double length word in the Double Accumulator is placed in the operand.

S D L - Shift the Double to the Left (38). This left shift is completely analogous to that described in the SSL instruction, if taken to pertain to the Double Accumulator, with the following exception:- the amount of shift, taken modulo 100, must not exceed 16. Indexed shifts are not permitted.

S D R - Shift the Double to the Right (39). This right shift is completely analogous to that described in the SSR instruction if taken to pertain to the Double Accumulator, with the following exceptions- the amount of shift, taken module 100, must not exceed 16. Indexed shifts are not permitted.

E. Instructions Pertaining to Floating Point Arithmetic.

The following instructions permit operations to be transacted with the two word standard form of floating point representation described previously. The Floating Accumulator is made up of the Lower and Upper Accumulators, the exponent being in the former and the mantissa in the latter. All floating point operations, with the exception of CAF, CSF and STF, destroy the contents of the Short Accumulator. When an address is given specifying a floating point operand, the address is that of the exponent portion of this operand.

P R F - PRepare a Floating point number (76). A number in non-standard form is placed into the Double Accumulator. The exponent of the non-standard form is in the Short Accumulator. The standard floating number appears in the Floating Accumulator.

C A F - Clear and Add a Floating point number (40). The floating point operand is placed in the Floating Accumulator.

C S F - Clear and Subtract a Floating point number (41). The negative of the floating point operand is placed into the Floating Accumulator.

S T F - STore a Floating point number (42). The floating point word in the Floating Accumulator is placed in the operand.

F A D - Floating point ADd (43). The floating point operand is added to the floating point word in the Floating Accumulator, the result remaining in the Floating Accumulator.

F S B - Floating point SuBtract (44). The floating point operand is subtracted from the floating point word in the Floating Accumulator.

F M P - Floating point MultiPly (45). The word in the Floating Accumulator is multiplied by the operand, the result remaining in the Floating Accumulator.

F D V - Floating point DiVide (46). The word in the Floating Accumulator is divided by the operand, the result remaining in the Floating Accumulator. Division by 0 is detected, and the programme halts. See Appendix IIb.

F S R - Floating point Square Root (47). The square root of the floating point number in the Floating Accumulators is placed in the Floating Accumulator. Attempts to calculate the square root of a negative number are detected, and the programme halts. See Appendix IIb.

F L G - Floating point LoGarithm (48). The logarithm to the base e of the floating point number in the Floating Accumulator is placed in the Floating Accumulator. Indexing has no significance. Range, $-30 < b(\text{exponent}) < 32$, $a > 0$ - may fail without warning outside the range, see Appendix IIb.

F E X - Floating point E X ponential (49). e (2.718281828), raised to the power indicated by the Floating point word in the Floating Accumulator, is placed in the Floating Accumulator. Indexing has no significance. Range, $2^{-512} < |x| < 2^{12}$ - will operate incorrectly outside range, see Appendix IIb.

F S N - Floating point S i N e (50). The sine of 11/2 times the floating point word in the Floating Accumulator is placed in the Floating Accumulator. Indexing has no significance. Range, x 2^{32}. See Appendix IIb.

F C S - Floating point C o S ine (51). The cosine of 11/2 times the floating point word in the Floating Accumulator is placed in the Floating Accumulator. Indexing has no significance. Range, x 2^{32}. See Appendix IIb.

F A T - Floating point Arc-Tangent (52). The arc-tangent, times 2/11, of the floating point word in the Floating Accumulator is placed in the Floating Accumulator. Indexing has no significance.

F. Instructions Pertaining to the Index Registers.

All of the following operations are non-indexable:-

L X P - Load an indeX register Positive (53). The operand is placed in the specified index register.

L X L - Load an indeX register positive Loss one (71). The operand, decreased by a P_{17}, is placed in the specified index register.

L X N - Load an indeX register Negative (54). The negative of the operand is placed in the specified index register.

A D X - ADd to an indeX register (55). The operand is added to the word in the specified index register, the result remaining in the index register.

S B X - SuBtract from an indeX register (56). The operand is subtracted from the word in the specified index register, the result remaining in the index register.

S T X - STore an indeX register (57). The contents of the specified index register are placed in the operand.

G. Decision or Jump Instructions.

All of the following instructions permit the selection of one of two paths in a programme, depending upon the quantities in various units of the computer. Both the operand address and the next instruction address contain symbolic instruction locations to which a jump can be made. The former is chosen if the condition specified by the instruction is realised, the latter if the conditions is not met.

As is normally the case with next instruction addresses, either the operand or next instruction positions may be left blank, and the corresponding jump condition then automatically leads to the next instruction in the SODA programme pack. Of course, both addresses may contain symbolic addresses. Both may not be blank. This error is detected, and the translation halts. For most of the instructions, indexing has no significance. It is not permitted in those operations employing a decrement. No trace identification number is associated with these instructions as the trace does not detect them.

J S Z - Jump if the Short is Zero. A jump to the operand address is made if the Short Accumulator contains a zero.

J S P - Jump if the Short is Positive. A jump to the operand address is made if the sign digit of the Short Accumulator is a zero.

J L Z - Jump if the Lower is Zero. A jump to the operand address is made if the contents of the Lower Accumulator are zero.

J L P - Jump if the Lower is Positive. A jump to the operand address is made if the sign digit of the Lower Accumulator is a zero.

J U Z - Jump if the Upper is Zero. A jump to the operand address is made if the Upper Accumulator contains a zero.

J U P - Jump if the Upper is Positive. A jump to the operand address is made if the sign digit of the Upper Accumulator is zero.

J D Z - Jump if the Double is Zero. A jump to the operand address is made if the Double Accumulator contains a zero.

J D P - Jump if the Double is Positive. A jump to the operand address is made if the sign digit of the Double Accumulator is a zero.

J I X - Jump on IndeX. The value of the decrement is subtracted from the specified index register, the result remaining in the index register. A jump to the operand address is made if this result is positive. The capacity of the index register must not be exceeded.

J X H - Jump on indeX High or equal. The value of the decrement is compared with that in the specified index register. A jump to the operand address is made if the latter value is algebraically greater than or equal to the value of the decrement.

J X E - Jump on indeX Equal. The value of the decrement is compared with that in the specified index register. A jump to the operand address is made if the two are equal.

H. Instructions Permitting the Block Transfer of Data.

R W O - Read into Working storage One (65). The array specified by the operand is transferred into working storage one. The original version on the drum remains unchanged. For some purposes it might be necessary to realise that it is actually the last track of the array which is initially placed in the thirty-two word storage representing working storage one. Other tracks are brought in at the correct time and normally need not be the concern of the programmer. Indexing has no significance.

R W T - Read into Working storage Two (66). The description given under RWO is applicable here if it is taken to pertain to working storage two.

W W O - Write from Working storage One (67). The array in working storage one is stored on the drum, in the array from which it originally came. In actuality, the thirty-two words representing working storage one are stored in the proper position of the array on the drum. The operand address of this instruction is ignored and is therefore of no significance. The name of the array involved can be included in the operand for clarification purposes when listing the programme. Indexing has no significance. It should be noted that the only time this instruction is given is when a series of calculation on an array has terminated, or is interrupted by placing a different array in the same working storage.

W W T - Write from Working storage Two (68). The description given under WWO is applicable here if it is taken to pertain to working storage two.

I. Instructions Pertaining to Input and Output.

All read and punch orders are not traceable.

R D O - ReaD into One from the reader. The binary information on the next three cards in the reader is placed into the thirty-two positions of working storage one. The first word read is in line Y of the first of the three cards, and the last is in line 5 of the third card. They go respectively to the first and thirty-second positions of working storage one. Line 6 of the third card contains a check sum representing the sum of the preceding elements with all overflows ignored. This number is compared with a sum calculated by the machine during the reading in process. If the two are the same computation continues. If they are different, the programme halts with 2, 16, and 16 on the N, S and D lights respectively. Depression of the single shot key causes the programme to continue. This check ensures that the information is read in correctly. The operand address is ignored and, therefore, has no significance. Indexing has no significance.

R D T - ReaD into Two from the reader. The description given under RDO is applicable here if it is taken to pertain to working storage two.

R D C - ReaD a Card. This instruction has an absolute address from 0 - 31, 32 - 63, and 64 - 95 and reads the 12 rows of a card within a working store or constant store beginning at the absolute address, e.g. if RDC 00034 was given, the card would be read into spaces 34 - 45 of working store two. If RDC 00062 was given, the 12 rows would be read into 62, 63, 32 - 41. Not indexable.

P H O - PuncH from working storage One. The binary information in the thirty-two positions of working storage one is punched out onto three cards in the punch unit. The first of the thirty-two words is placed in line Y of the first of the three cards and the thirty-second word is placed in line 5 of the third card. A check sum of the elements punched, with all overflows ignored, is placed in line 6. The operand address is ignored and therefore has no significance. Indexing has no significance.

P H T - PuncH from working storage Two. The description given under PHO is applicable here if it is taken to pertain to working storage two.

P H C - PuncH a Card. This instruction punches from the working stores, or constants store, and operates in the same manner as RDC. Not indexable.

R D A - ReaD Array. The binary information on the next group of cards in the reader is placed on the drum in the array position which is indicated by the operand. The number of elements read is equal to the number in index register 1, an integer to 16 binary places. They are stored on the drum in such a manner that the last element is placed in the thirty-second position of the last track associated with the array, as defined previously by an arrays control card. The numbers are read in 12 to a card beginning at line Y of the first card. After the specified number of elements have been read in, a check sum of all the preceding elements, with all overflows ignored, is read from the next card row and compared with a similar sum calculated by the machine during the reading in process. If the two are the same, computation continues. If they are different, the programme halts with 2, 13, and 13 on the N, S and D lights respectively. Depression of the single shot key causes the programme to continue. This check ensures that the information is read in correctly. Indexing has no significance.

P H A - PuncH Array. The binary information in the array position which is indicated by the operand is punched onto cards in the punch unit. The number of elements punched is equal to the number in index register 1, an integer to 16 binary places. They are taken from memory in such a manner that the last element punched is in the thirty-second position of the last track associated with the array as defined previously by an arrays control card. The numbers are punched 12 to a card beginning at line Y of the first card. After the specified number of elements have been punched, a check sum of all the preceding punched elements, with all overflows ignored, is punched onto the next card row. Indexing has no significance.

Examples of Input and Output Data.

Data read onto the drum initially as arrays is of the form given in the section "Control Cards", under the discussion of definition of arrays. Data read in or punched using the orders RDO, RDT, PHO, PHT, RDA and PHA is of the following form:-

The array is read in or punched out with the first element of the array on the Y Row of the first card followed by the succeeding elements. Thus, for an array of 65 elements the first 60 would be placed on 5 cards in order of their appearance on the drum; the remaining 5 elements would be placed in rows Y through 2, followed by a check sum of the elements on row 3 of the sixth card. More than 32 elements may be read or punched at one time only by using an RLA or PHA instruction.

In the case of RDO, RDT, PHO and PHT, the results, as they appear on cards, either for read or punch, is as follows. Using working store 2.

Absolute Address of Element	Card	Row
32	1	Y
33	1	X
34	1	0
.	.	.
.	.	.
.	.	.
62	3	4
63	3	5
(Sum of elements 32-63)	3	6

When handling floating or double length numbers, theorder in which they appear on the card follows the rules as discussed under information representation. It must be remembered that when absolute addresses are given for these double length numbers, only even numbered ones are valid in specifying the number. The single, even address together with an instruction implying two words is sufficient to specify both parts of the number.

Assume we defined an array of 36 elements with the last track number 215. When an order such as Read Array (RDA) is given, the card punching and drum locations would be as follows:-

Track	Position on Track	Card No.	Row	Element
214	29	1	Y	1
214	30	1	X	2
214	31	1	0	3
214	32	1	1	4
215	1	1	2	5
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
215	30	3	7	34
215	31	3	8	35
215	32	3	9	36
-	-	4	Y	Sum of Elements 1-36.
-	-	4	X-9	Blank

If the array was already on the drum, as the result of a series of calculations, the order Punch Array (PHA) would produce the above cards. The number 36 would have to be in index register 1 in either case (as a binary integer to 16 binary places).

J. Miscellaneous Instructions.

H P R - Halt and PRoceed This instruction causes the computation to stop. If the operand contains 0, a 1 appears on both the S and D lights. If the operand contains a decimal number from 1 to 21, the binary equivalent will appear on both the S and D lights. This permits an indication at which point in the programme the stop occurred. This instruction is useful mainly when a new programme is being tested, and it is desired to stop in order to view intermediate results or to initiate the trace facility. An address greater than 21 is an error and is detected by the translation programme. A symbolic operand address is not permitted. Indexing has no significance. Depression of the single-shot key causes the programme to continue in its normal sequence. After a programme is made to work correctly, it may be retranslated with the HPR instructions removed. This instruction is not traceable, and the instruction also turns off the read or punch - when tracing.

S T P - SToP. This halts the computation. An alarm is sounded to herald the completion of the calculation. A one appears on both the S and D lights, and depression of the single-shot key has no effect. Indexing has no significance.

S T Z - STore a Zero (60). A zero is placed in the operand.

S T O - STore a One (61). A one (P_1) is placed in the operand.

S T A - STore an Address unit (62). A P_{17} is placed in the word indicated by the operand.

S T H - STore a High position bit (63). A P_{32} (sign bit) is placed in the operand.

S T M - STore a Minus one (64). Minus one (all ones) is placed in the operand.

A C A - ACtivate Alarm (58). The alarm is turned on. Indexing has no significance.

S P A - StoP Alarm (59). The alarm is turned off. Indexing has no significance.

R I L - Read from the Input Lights (79). The contents of the input lights are placed in the word indicated by the operand.

W O L - Write into the Output Lights (77). The operand is accumulated into the output lights, the original version remaining unchanged. None of the output lights are turned off by this instruction. Only new ones may be lit.

C O L - Clear the Output Lights (78). The content of the output lights are set to zero. Indexing has no significance.

C W O - Clear Working storage One (72). The thirty-two physical positions of the Deuce machine corresponding to working storage one are set to zero. Indexing has no significance.

C W T - Clear Working storage Two (73). The thirty-two physical positions of the Deuce machine corresponding to working storage two are set to zero. Indexing has no significance.