PRELIMINARY DISCUSSION OF THE LOGICAL DESIGN OF AN ELECTRONIC COMPUTING INSTRUMENT

BY

Arthur W. Burks

Herman H. Goldstine

John von Neumann

TABLE OF CONTENTS

TABLE OF CONTENTS (cont'd)

PREFACE

This report has been prepared in accordance with the terms of Contract W-36-034-ORD-7481 between the Research and Development Service, Ordnance Department, U. S. Army and the Institute for Advanced Study. It is intended as the first of two papers dealing with some aspects of the overall logical considerations arising in connection with electronic computing machines. An attempt is made to give in this, the first half of the report, a general picture of the type of instrument now under consideration and in the second half a study of how actual mathematical problems can be coded, i.e., prepared in the language the machine can understand.

It is the present intention to issue from time to time reports covering the various phases of the project. These papers will appear whenever it is felt sufficient work has been done on a given aspect, either logical or experimental to justify its being reported.

The authors also wish to express their thanks to Dr. John Tukey of Princeton University for many valuable discussions and suggestions.

The Institute for Advanced Study
28 June 1946

Arthur W. Burks
Herman H. Goldstine
John von Neumann

PRELIMINARY DISCUSSION OF THE LOGICAL DESIGN OF AN ELECTRONIC COMPUTING INSTRUMENT

1.0 Principal Components of the Machine

1.1. Inasmuch as the completed device will be a general-purpose computing machine it should contain certain main organs relating to arithmetic, memory-storage, control and connection with the human operator. It is intended that the machine be fully automatic in character, i.e., independent of the human operator after the computation starts. A fuller discussion of the implications of this remark will be given in 3 below.

1.2. It is evident that the machine must be capable of storing in some manner not only the digital information needed in a given computation such as boundary values, tables of functions (such as the equation of state of a fluid) and also the intermediate results of the computation (which may be wanted for varying lengths of time), but also the instructions which govern the actual routine to be performed on the numerical data. In a special-purpose machine these instructions are an integral part of the device and constitute a part of its design structure. For an all-purpose machine it must be possible to instruct the device to carry out any whatsoever computation that can be formulated in numerical terms. Hence there must be some organ capable of storing these program orders. There must, moreover, be a unit which can understand these instructions and order their execution.

1.3. Conceptually we have discussed above two different forms of memory: storage of numbers and storage of orders. If, however, the orders to the machine are reduced to a numerical code and if the machine

can in some fashion distinguish a number from an order, the memory organ can be used to store both numbers and orders. The coding of orders into numeric form is discussed in 6.3 below.

1.4. If the memory for orders is merely a storage organ there must exist an organ which can automatically execute the orders stored in the memory. We shall call this organ the Control.

1.5. Inasmuch as the device is to be a computing machine there must be an arithmetic organ in it which can perform certain of the elementary arithmetic operations. There will be, therefore, a unit capable of adding, subtracting, multiplying and dividing. It will be seen in 6.6 below that it can also perform additional operations that occur quite frequently.

The operations that the machine will view as elementary are clearly those which are wired into the machine. To illustrate, the operation of multiplication could be eliminated from the device as an elementary process if one were willing to view it as a properly ordered series of additions. Similar remarks apply to division. In general, the inner economy of the arithmetic unit is determined by a compromise between the desire for speed of operation - a non-elementary operation will generally take a long time to perform since it is constituted of a series of orders given by the control - and the desire for simplicity, or cheapness, of the machine.

1.6. Lastly there must exist devices, the input and output organs, whereby the human operator and the machine can communicate with each other. This organ will be seen in 4.5 below, where it is discussed, to constitute a secondary form of automatic memory.

2.0 First Remarks on the Memory

2.1. It is clear that the size of the memory is a critical consideration in the design of a satisfactory general-purpose computing machine. We proceed to discuss what quantities the memory should store for various types of computations.

2.2. In the solution of partial differential equations the storage requirements are likely to be quite extensive. In general, one must remember not only the initial and boundary conditions and any arbitrary functions that enter the problem but also an extensive number of intermediate results.

a) For equations of parabolic or hyperbolic type in two independent variables the integration process is essentially a double induction: To find the values of the dependent variables at time $t+\Delta t$ one integrates with respect to x from one boundary to the other by utilizing the data at time t as if they were coefficients which contribute to defining the problem of this integration.

Not only must the memory have sufficient room to store these intermediate data but there must be provisions whereby these data can later be removed, i.e., at the end of the $(t+\Delta t)$ cycle, and replaced by the corresponding data for the $(t+2\Delta t)$ cycle. This process of removing data from the memory and of replacing them with new information must, of course, be done quite automatically under the direction of the control.

b) For total differential equations the memory requirements are clearly similar to, but smaller than, those discussed in a) above.

c) Problems that are solved by iterative procedures such as systems of linear equations or elliptic partial differential equations, treated by relaxation techniques, may be expected to require quite extensive

memory capacity. The memory requirement for such problems is apparently much greater than for those problems in a) above in which one needs only to store information corresponding to the instantaneous value of one variable (t in a) above, while now entire solutions (covering all values of all variables) must be stored. This apparent discrepancy in magnitudes can, however, be somewhat overcome by the use of techniques which permit the use of much coarser integration meshes in this case, than in the cases under a).

2.3. It is reasonable at this time to build a machine that can conveniently handle problems several orders of magnitude more complex than are now handled by existing machines, electronic or electro-mechanical. We consequently plan on a fully automatic electronic storage facility of about 4,000 numbers of 40 binary digits each. We believe that this exceeds the capacities required for problems that one deals with at present by a factor of about 10. In addition, we propose we have a subsidiary memory, which is also fully automatic, of much larger capacity on some medium such as magnetic tape.

3.0. First Remarks on the Control and Code

3.1. It is easy to see by formal-logical methods, that there exist codes that are in abstracto adequate to control and cause the execution of any sequence of operations which are individually available in the machine and which are, in their entirety, conceivable by the problem planner. The really decisive considerations from the present point of view, in selecting a code, are more of a practical nature: Simplicity of the equipment demanded by the code, and the clarity of its application to the actually important problems together with the speed of its handling of those problems. It would take us much too far to discuss these questions

at all generally or from first principles. We will therefore restrict ourselves to analyzing only the type of code which we now envisage for our machine.

3.2. There must certainly be instructions for performing the fundamental arithmetic operations. The specifications for these orders will not be completely given until the arithmetic unit is described in a little more detail.

3.3. It must be possible to transfer data from the memory to the arithmetic organ and back again. In transferring information from the arithmetic organ back into the memory there are two types we must distinguish: Transfers of numbers as such and transfers of numbers which are parts of orders. The first case is quite obvious and needs no further explication. The second case is more subtle and serves to illustrate the generality and simplicity of the system. Consider the problem of interpolation in the system. Let us suppose that we have formulated the necessary instructions for performing an interpolation of order n in a sequence of data. The exact location in the memory of the (n+1) quantities that bracket the desired functional value is, of course, a function of the argument. This argument probably is found as the result of a computation in the machine. We thus need an order which can substitute a number into a given order - in the case of interpolation the location of the nearest argument in our table to the desired value. By means of such an order the results of a computation can be introduced into the instructions governing that or a different computation. This makes it possible for a sequence of instructions to be used with different sets of numbers located in different parts of the memory.

To summarize, transfers into the memory will be of two sorts: Total substitutions, whereby the quantity previously stored is cleared out

and replaced by a new number. Partial substitutions in which that part of an order containing a memory location-number is replaced by a new memory location number.

3.4. It is clear that one must be able to get numbers from any part of the memory at any time. The treatment in the case of orders can, however, be more methodical since one can at least partially arrange the control instructions in a linear sequence. Consequently the control organ will be so constructed that it will normally proceed from place n in the memory to place (n+1) for its next instruction.

3.5. The utility of an automatic computer lies in the possibility of using a given sequence of instructions repeatedly, the number of times it is iterated being either preassigned or dependent upon the results of the computation. When the iteration is completed a different sequence of orders is to be followed, so we must, in most cases, give two parallel trains of orders preceded by an instruction as to which routine is to be followed. This choice can be made to depend upon the sign of a number (zero being reckoned as plus for machine purposes). Consequently we introduce an order (the conditional transfer order) which will, depending on the sign of a given number, cause the proper one of two routines to be executed.

Frequently two parallel trains of orders terminate in a common routine. It is desirable, therefore, to order the control in either case to proceed to the beginning point of the common routine. This unconditional transfer can be achieved either by the artificial use of a conditional transfer or by the introduction of an explicit order for such a transfer.

3.6. Finally we need orders which will integrate the input-output devices with the machine. These are discussed breifly in 6.8.

3.7. We proceed now to a more detailed discussion of the machine. Inasmuch as our experience has shown that the moment one chooses a given component as the elementary memory unit one has also more or less determined upon much of the balance of the machine, we start by a consideration of the memory organ. In attempting an exposition of a highly integrated device like a computing machine we do not find it possible, however, to give an exhaustive discussion of each organ before completing its description. It is only in the final block diagrams that anything approaching a complete unity is achieved.

4.0. The Memory Organ

4.1. Ideally one would desire an indefinitely large memory capacity such that any particular 40 binary digit number or word would be immediately - i.e., in the order of 1 to 100 μs - available and that words could be replaced with new words at about the same rate. It does not seem possible physically to achieve such a capacity. We are therefore forced to recognize the possibility of constructing a hierarchy of memories, each of which has greater capacity than the preceding but which is less quickly accessible.

The most common forms of storage in electrical circuits are the flip-flop, the gas tube, and the electro-mechanical relay. To achieve a memory of n words would, of course, require about 40 n such elements, exclusive of the switching elements. We saw earlier (cf. 2.2) that a fast memory of several thousand words is not at all unreasonable for an all-purpose instrument. Hence, about 10^5 flip-flops or analogous elements would be required! This would, of course, be entirely impractical.

We must therefore seek out some more fundamental method of storing electrical information than has been suggested above. One criterion for

such a storage medium is that the indivicual storage organs, which accomodate only one binary digit each, should not be macroscopic components, but rather microscopic elements of some suitable organ. They would then, of course, not be identified and switched to by the usual macroscopic wire connections, but by some functional procedure in manipulating that organ.

One device which displays this property to a marked degree is the iconoscope tube, which has a resolution of about 500 by 500. One is accordingly led to consider the possibility of storing electrical charges on a dielectric plate inside a cathode-ray tube. Effectively such a tube is nothing more than a myriad of electrical capacitors which can be connected into the circuit by means of an electron beam.

At the present time the Princeton Laboratories of the Radio Corporation of America are engaged in the development of a storage tube, the Selectron, of the type we have mentioned above. This tube is also planned to have a non-amplitude-sensitive switching system whereby the electron beam can be directed to a given spot on the plate within a quite small fraction of a millisecond. Inasmuch as the storage tube is the key component of the machine envisaged in this report we are extremely fortunate in having secured the cooperation of the RCA group in this as well as in various other developments.

An alternate form of rapid memory organ is the feed-back delay line described in various reports on the EDVAC. Inasmuch as that device has been so clearly reported in those papers we give no further discussion There are still other physical and chemical properties of matter in the presence of electrons or photons that might be considered but since none is yet beyond the early discussion stage we shall not make further mention of them.

4.2. We shall accordingly assume throughout the balance of this report that the Selectron is the modus for storage of words at electronic speeds. As now planned, this tube will have a capacity of $2^{12} = 4,096 \approx 4,000$ binary digits. To achieve a total electronic storage of about 4,000 words we propose to use 40 selectrons, thereby achieving a memory of 2^{12} words of 40 binary digits each.

4.3. There are two possible means for storing a particular word in the selectron memory - or in fact in either a delay line memory or in a storage tube with amplitude-sensitive deflection. One method is to store the entire word in a given tube and then to get the word out by picking out its respective digits in a serial fashion. The other method is to store in corresponding places in each of the 40 tubes one digit of the word. To get a word from the memory in this scheme requires, then, one switching mechanism to which all 40 tubes are connected in parallel. Such a switching scheme seems to us to be simpler than the technique needed in the serial system and is, of course, 40 times faster. We accordingly adopt the parallel procedure and thus are led to consider a so-called parallel machine, as contrasted with the serial principles being considered for the EDVAC. The essential difference between these two systems lies in the method of performing an addition: In a parallel machine all corresponding pairs of digits are added simultaneously, whereas in a serial one these pairs are added serially in time.

4.4. To summarize, we assume that the fast electronic memory consists of 40 selectrons which are switched in parallel by a common switching arrangement. The inputs of the switch are controlled by the Control Organ.

4.5. Inasmuch as a great many highly important classes of problems require a far greater total memory than 2^{12} words, we now consider the next stage in our storage hierarchy. Although the solution of partial differential equations frequently involves the manipulation of many thousands of words, these data are generally required only in blocks which are well within the 2^{12} capacity of the electronic memory. Our second form of storage then can be a medium which is controlled by the main Control Organ of the computer and is thus an integral part of the system, not requiring human intervention.

There are evidently two distinct problems raised above. One can choose a given medium for storage such as teletype tapes, magnetic wire or tapes, movie film or similar media. There still remains the problem of automatic integration of this storage medium with the machine. This integration is achieved logically by introducing appropriate orders into the code which can instruct the machine to read or write on the medium, or to move it by a given amount or to a place with given characteristics. We discuss this question a little more fully in 6.8.

Let us return now to the question of what properties the secondary storage medium should have. It clearly should be able to store information for periods of time long enough so that only a few percent of the total computing time is spent in re-registering information that is "fading" off. It is certainly desirable, although not imperative, that information can be erased and replaced by new data. The medium should be such that it can be controlled, i.e., moved forward and backward, automatically. This consideration makes certain media, such as punched cards, undesirable. While cards can, of course, be printed or read by appropriate orders from some machine, they are not well adapted to problems in which the output data are fed

directly back into the machine, and are required in a sequence which is non-monotone with respect to the order of the cards. The medium should be capable of remembering very large numbers of data at a much smaller price than electronic devices. It must be fast enough so that, even when it has to be used frequently in a problem, a large percentage of the total solution time is not spent in getting data into and out of this medium and achieving the desired positioning on it. If this condition is not reasonably well met, the advantages of the high electronic speeds of the machine will be largely lost.

Both light- or electron-sensitive film and magnetic tapes, whose motions are controlled by servo-mechanisms integrated with the Control Organ, seem to fulfill our needs reasonably well. In a subsequent section we discuss a few problems to show the overall efficiency of this system using magnetic tapes.

4.6. Lastly our memory hierarchy requires a vast quantity of dead storage, i.e., storage not integrated with the machine. This storage requirement may be satisfied by a library of tapes that can be introduced into the machine when desired and at that time become automatically controlled. Thus our dead storage really is nothing but an extension of our secondary storage medium. It differs from the latter only in its availability to the machine.

4.7. We impose one additional requirement on our secondary memory. It must be possible for a human to put words onto the tape or other substance used and to read the words put on by the machine. In this manner the human can control the machine's functions. It is now clear that the secondary storage medium is really nothing other than a part of our input-output system.

4.8. There is another highly important part of the input-output which we merely mention at this time, namely, some mechanism for viewing graphically the results of a given computation. This can, of course, be achieved by a selectron-like tube which causes its screen to fluoresce when data are put on it by an electron beam.

4.9. For definiteness in the subsequent discussions we assume that associated with the output of each selectron is a flip-flop. This assemblage of 40 flip-flops, we term the Selectron Register.

5.0. The Arithmetic Organ

5.1. In this chapter we discuss the features we now consider desirable for the arithmetic part of our machine. We give our tentative conclusions as to which of the arithmetic operations should be built into the machine and which should be programmed. Finally, a schematic of the arithmetic unit is described.

5.2. In a discussion of the arithmetical organs of a computing machine one is naturally led to a consideration of the number system to be adopted. In spite of the long-standing tradition of building digital machines in the decimal system, we feel strongly in favor of the binary system for our device. Our fundamental unit of memory is naturally adapted to the binary system since we do not attempt to measure gradations of charge at a particular point in the selectron but are content to distinguish two states. In delay line memories one is also content merely to recognize the presence or absence of a pulse, the flip-flop again is truly a binary device. Hence if one contemplates using a decimal system with either the iconoscope or delay-line memory one is forced into a binary coding of the decimal system - each decimal digit being represented by at least a tetrad of binary digits. Thus an accuracy of ten decimal digits requires 40

binary digits. In a true binary representation of numbers, however, only 34 digits suffice to achieve a precision of 10^{10}. The use of the binary system is therefore somewhat more economical of equipment than is the decimal.

An even more significant virtue of the binary system as against the decimal is the greater simplicity and speed with which the elementary operations can be performed. To illustrate, consider multiplication by repeated addition. In binary multiplication the product of a particular digit of the multiplier by the multiplicand is either the multiplicand or null according as the multiplier digit is 1 or 0. In the decimal system, however, this product has ten possible values between null and 9 times the multiplicand, inclusive. Of course, a decimal number has only $\log_{10}2 \sim .3$ times as many digits as a binary number of the same accuracy, but even so multiplication in the decimal system is considerably longer than in the binary system. One can accelerate decimal multiplication by complicating the circuits, but this fact is irrelevant to the point just made since binary multiplication can likewise be accelerated by adding to the equipment. Similar remarks may be made about the other operations.

The one disadvantage of the binary system from the human point of view is the conversion problem. Since, however, it is completely known how to convert numbers from one base to another and since this conversion can be effected solely by the use of the usual arithmetic processes there is no reason why the computer itself cannot carry out this conversion. It might be erroneously argued that this is a time consuming operation. But a general-purpose computer, used as a scientific research tool, is called upon to do a very great number of multiplications upon a relatively small amount of input data, and hence the time consumed in the decimal to binary conversion is only a trivial percent of the total computing time. A similar remark is applicable to the output data.

In the preceeding discussion we have tacitly assumed the desirability of introducing and withdrawing data in the decimal system. We feel, however, no great reverence for the base 10 in a scientific instrument and consequently will probably attempt to train ourselves to use numbers base 2 or 8. (The reason for the base 8 is this: Since 8 is a power of 2 the conversion to binary is trivial; since 8 is about of the size of 10 it violates many of our habits less badly than base 2.)

5.3. Several of the digital computers being built or planned in this country and England are to contain a so-called "floating decimal point". This is a mechanism for expressing each word as a characteristic and a mantissa -- e.g., 123.45 would be carried in the machine as (0.12345, 03), where the three is the exponent of 10 associated with the number. There appear to be two major purposes in a "floating" decimal point system both of which arise from the fact that the number of digits in a word is a constant, fixed by design considerations for each particular machine. The first of these purposes is to retain in a sum or product as many significant digits as possible and the second of this is to free the human operator from the burden of estimating and inserting into a problem "scale factors" -- multiplicative constants which serve to keep numbers within the limits of the machine.

There is , of course, no denying the fact that human time is consumed in arranging for the introduction of suitable scale factors. We only argue that the time so consumed is a very small percentage of the total time we will spend in preparing an interesting problem for our machine. The first advantage of the floating point is, we feel, somewhat illusory. In order to have such a floating point one must waste memory capacity

which could otherwise be used for carrying more digits per word. It would therefore seem to us not at all clear whether the modest advantages of a floating binary point offset the loss of memory capacity and the increased complexity of the arithmetic and control circuits.

There are certainly some problems within the scope of our device which really require more than 2^{-40} precision. To handle such problems we wish to plan in terms of words, whose lengths are some fixed integral multiple of 40, and program the machine in such a manner as to give the corresponding aggregates of 40 digit words the proper treatment. We must then consider an addition or multiplication as a complex operation programmed from a number of primitive additions or multiplications. There would seem to be considerable extra difficulties in the way of such a procedure in an instrument with a floating binary point.

The reader may remark upon our alternate spells of radicalism and conservatism in deciding upon various possible features for our mechanism. We feel, however, that we have a sound rationale whereby we judge the merits of an idea. We wish to incorporate into the machine -- in the form of circuits -- only such logical concepts as are either necessary to have a complete system or highly convenient because of the frequency with which they occur.

5.4. On the basis of this criterion we definitely wish to build into the machine circuits which will enable it to form the binary sum of two 40 digit numbers. We make this decision not because addition is a logically basic notion but rather because it would slow the mechanism as well as the operator down encurmously if each addition were programmed out of the more simple operations of "and", "or" and "not". Similarly we reject the desire to form products by programming them out of additions.

The cases for division and square-root are much less clear.

It is well known that the reciprocal of a number a can be formed to any desired accuracy by iterative schemes. One such scheme consists of improving an estimate X by forming $X' = 2X-aX^2$. Thus the new error $1-aX'$ is $(1-aX)^2$, which is the square of the error in the preceding estimate. We notice that in the formation of X', there are two bonafide multiplications -- we do not consider multiplication by 2 as a true product since we will have a facility for shifting right or left in one pulse time. If then we somehow could guess 1/a to a precision of 2^{-5}, 6 multiplications -- 3 iterations -- would suffice to give a final result good to 2^{-40}. Accordingly a small table of 2^4 entries could be used to get the initial estimate of 1/a. Accordingly we see that the question of building a divider is really a function of how fast it can be made compared to the iterative method sketched above. We have however conceived a divider which is much faster than the 6 multiplications of the iterative procedure and therefore feel justified in building it, especially since the amount of equipment needed above the requirements of the multiplier is not important.

It is, of course, also possible to handle square-roots by iterative techniques. In fact if X is our estimate of $a^{1/2}$, then $X'=X/2+a/2X$ is a better estimate. We see that this scheme involves one division per iteration. As will be seen below in our more detailed examination of the arithmetic organ we do not include a square-rooter in our plans because such a device would involve more equipment than we feel is desirable in a first model.

5.5. The first part of our arithmetic organ requires little discussion at this point. It should be a parallel storage organ which can receive a number and add it to the one already in it, and which is also able to clear its contents. We will call such an organ an accumulator. It is quite conventional in principle in past and present computing machines of the most varied types. (E.g.: Desk multipliers, standard IBM counters, more modern relay machines, the ENIAC.) There are, of course, numerous ways to build such a binary accumulator. We distinguish two broad types of such devices: Static and dynamic or pulse-type accumulators. These will be discussed in 5.10, but it is first necessary to make a few remarks concerning the arithmetic of binary addition. In a parallel accumulator, the first step in an addition is to add each digit of the addend to the corresponding digit of the augend. The second step is to perform the carries, and this must be done in sequence since a carry may produce a carry. In the worst case, 39 carries will occur. Clearly it is inefficient to allow 39 times as much time for the second step (performing the carries) as for the first step (adding the digits). Hence either the carries must be accelerated, or use must be made of the average number of carries, or both.

5.6. We go to show that for a sum of binary words, each of length n, the length of the largest carry sequence is on the average not in excess of $\log_2 n$. Let $p_n(v)$ designate the probability that a carry sequence is of length v or greater in the sum of two binary words of length n. Then clearly $p_n(v)-p_n(v+1)$ is the probability that the largest carry sequence is of length exactly v and the weighted average $a_n = \sum_{v=0} (p_n(v) - p_n(v+1))\, v$ is the average length of such a carry since $\sum_{v=0} (p_n(v) - p_n(v+1) = 1$ and since $p_n(v) = 0$ if $v > n$. We notice

moreover that $a_n = \sum_{v \geq 1} p_n(v)$ and proceed to show that $p_n(v) \leq$ smaller $(1, \frac{(n-v+1)}{2^{v+1}})$.

Observe first that $p_n(v) = p_{n-1}(v) + \frac{1}{2^{v+1}} (1-p_{n-v}(v))$, if $v \leq n$. Indeed: $p_n(v)$ is the probability that the sum of two n-digit numbers contains a carry sequence of length $\geq v$. This probability obtains by adding the probabilities of two mutually exculsive alternatives: First: Even the n-1 first digits of the two numbers by themselves contain a carry sequence of length $\geq v$. This has the probability $p_{n-1}(v)$. Second: The n-1 first digits of the two numbers by themselves do not contain a carry sequence of length $\geq v$. In this case any carry sequence of length $\geq v$ in the total numbers (of length n) must end with the last digits of the total sequence. Hence these must form the combination 1, 1. The next v-1 digits must propagate the carry, hence each of these must form the combination 1, 0 or 0, 1. (The combinations 1, 1 and 0, 0 do not propagate a carry.) The probability of the combination 1, 1 is $\frac{1}{4}$, that one of the alternative combinations 1, 0 or 0, 1 is $\frac{1}{2}$. The total probability of this sequence is therefore $\frac{1}{4}(\frac{1}{2})^{v-1} = \frac{1}{2^{v+1}}$. The remaining n-v digits must not contain a carry sequence of length $\geq v$. This has the probability $1-p_{n-1}(v)$. Thus the probability of the second case is $\frac{1}{2^{v+1}} (1-p_{n-1}(v))$. Combining these two cases, the desired relation $p_n(v) = p_{n-1}(v) + \frac{1}{2^{v+1}} (1-p_{n-v}(v))$ obtains. The observation that $p_n(v) = 0$ if $v > n$ is trivial.

We see with the help of the formulas proved above that $p_n(v) - p_{n-1}(v)$ is always $\leq \frac{1}{2^{v+1}}$, and hence that the sum $\sum_{i=1}^{n} (p_i(v) - p_{i-1}(v)) = p_n(v)$ is not in excess of $\frac{n-v+1}{2^{v+1}}$ since there are $n-v+1$ terms in the sum; since, moreover, each $p_n(v)$ is a probability, it

is not greater than 1. Hence we have $p_n(v) \leq$ smaller $(1, \frac{n-v+1}{2^{v-1}})$.

Finally we turn to the question of getting an upper bound on $a_n = \sum_{v=1} p_n(v)$. Choose K so that $2^K \leq n \leq 2^{K+1}$. Then

$$a_n = \sum_{v=1}^{K-1} p_n(v) + \sum_{v=K} p_n(v) \leq \sum_{v=1}^{K-1} 1 + \sum_{v=K} \frac{n}{2^{v+1}} = K-1 + \frac{n}{2^K}.$$

This last expression is clearly linear in n in the interval $2^K \leq n \leq 2^{K+1}$, and it is $=K$ for $n = 2^K$ and $=K+1$ for $n = 2^{K+1}$, i.e., it is $= \log_2 n$ at both ends of this interval. Since the function $\log_2 n$ is everywhere concave from below, it follows that our expression is $\geq \log_2 n$ throughout this interval. Thus $a_n \leq \log_2 n$. This holds for all K, i.e., for all n, and it is the inequality which we wanted to prove.

For our case $n = 40$ we have $a_n \geq \log_2 40 \doteq 5.3$, i.e., an average length of about 5 for the longest carry sequence. (The actual value of a_{40} is 4.62.)

5.7. We propose to handle negative numbers by means of a system of complements with respect to 2 to some power. To fix on this power we further propose to place all numbers ξ in the machine so that $|\xi| < |$. Thus we express ξ temporarily as $x_1, x_2, \ldots, x_{39}$. (These digits will be prefaced by a 0-th digit expressing the sign, cf. below.) We then take all complements with respect to $2^0 = 1$ and observe that $-(\frac{1}{2} + \frac{1}{2^2} + \ldots + \frac{1}{2^{39}}) + \frac{1}{2^{39}}$. If now $\xi \geq 0$, we express it as $0.x_1, x_2, \ldots, x_{39}$; if $\xi < 0$, we express it for machine purposes as the complement of ξ, i.e., as $1.y_1, y_2, \ldots, y_{39}$ where $(0.x_1, x_2, \ldots, x_{39}) + (0.y_1, y_2, \ldots, y_{39}) = (1.0\ 0\ \ldots 0)$. We have thus a 1-1 correspondende between the interval $-1 \leq \xi < +1$ and the set of all 40- tuples of binary digits.

5.8. To effect a multiplication we first send the multiplier into a register AR, the Arithmetic Register, which is essentially a set of

40 flip-flops and whose characteristics will be discussed below. We place the multiplicand in the Selectron Register, SR, and use the accumulator, A, to form and store the partial products. We propose to multiply the entire multiplicand by the successive digits of the multiplier in a serial fashion. There are, of course, two possible ways this can be done: We either can start with the digit in the lowest position -- position 2^{-40} -- or in the highest position -- position 2^{-1} -- and proceed successively to the left or right, respectively. There are a few disadvantages from our point of view to starting with the right-most digit of the multiplier. We therefore describe that scheme. Assume we have already multiplied the multiplicand a by the n last (n = 0, 1, ..., 38) digits of the multiplier and desire to multiply by the next digit β_{39-n}. We assume further that A holds a quantity $p_n/2$. We now form $p_{n+1} = \frac{p_n}{2} + \beta_{39-n}$ xa in A and then $\frac{p_{n+1}}{2}$ by shifting the contents of A one stage to the right by means of an electronic shifter which is part of A. Clearly at the end of the operation we have - apart from possible corrections due to the use of complements and certain carry problems - in A the 40 significant figures of our product.

To complete the general picture of our multiplication technique we must consider how we sense the respective digits of our multiplier. There are two schemes which come to one's mind in this connection. One is to have a gate tube associated with each flip-flop of AR in such a fashion that this gate is open if a digit is 1 and closed if it is null. We would then need a 39 stage counter to act as a switch which would successively stimulate these gate tubes to react. A more efficient scheme is built into AR a shifter circuit which enables AR to be shifted one stage to the right each time A is shifted and to sense the value of the digit in the right-most flip-flop of AR . The shifter itself requires one gate tube per stage

We need in addition a counter to count out the 39 steps of the multiplication, but this can be achieved by a six stage binary counter. Thus the latter is more economical of tubes and has one additional virtue from our point of view which we discuss in the next paragraph.

The choice of 40 digits to a word (including the sign) is probably adequate for most computational problems but situations certainly might arise when we desire higher precision, i.e. words of greater length. A trivial illustration of this would be the computation of π to hundreds of figures. More important instances are the solutions of N linear equations in N variables for large values of N. (Probably when $N > 10$.) It is therefore desirable to be able to handle numbers of 39K digits and sign by means of program instructions -- one way to achieve this end is to regard a word of length 39K as a sequence of K words each of length 39. In order to be able to treat numbers in this manner, it is necessary to keep not 39 digits in a product, but 78; this is discussed in more detail in 6.6.3 below. To accomplish this end (conserving 78 product digits) we connect, via our shifter circuit, the right-most digit of A with the left digit of AR. Thus, when in the process of multiplication a shift is ordered, the last digit of A is transferred into the place in AR made vacant when the multiplier was shifted.

5.9. In the previous discussion we tacitly assumed that the multiplier and multiplicand were non-negative. If either or both are negative we must perform certain correction operations. If the multiplier is negative, we add the complement of the multiplicand into the partial product $p_{39}/2$; if the multiplicand is negative, we connect, via our shifter circuit, the left-most digit of A with the right digit of AR and pass 1 minus this digit into A. Thus, when a multiplication shift is ordered, the

complement of the multiplier is fed into the partial products -- clearly an additional complement correction pulse is also needed. Lastly, if both the multiplier and multiplicand are negative, we must not only carry out both the processes described above, but we must also introduce an additional pulse to correct the digit in the 2^0 place. We make the above discussion somewhat more precise in the next paragraphs.

5.10. To complete our discussion of the multiplicative organs of our machine we must return to a consideration of the two types of Accumulators mentioned earlier. The static adder operates by simultaneously applying static voltages to its two inputs -- one for each of the two numbers being added. When steady-state operation is reached the total sum is formed complete with all carries. For such an accumultor the above discussion is substantially complete, except that it should be remarked that such a circuit requires at most 39 rise times to complete a carry.

Each stage of a dynamic accumulator consists of a binary counter for registering the digit and a flip-flop for temporary storage of the carry. The counter receives a pulse if a 1 is to be added in at that place; if this causes the counter to go from1 to 0 a carry has occurred and hence the carry flip-flop will be set. It then remains to perform the carries. Each flip-flop has associated with it a gate, the output of which is connected to the next binary counter to the left. The carry is begun by pulsing all carry gates. Now a carry may produce a carry, so that the process needs to be repeated until all carry flip-flops register 0. This can be detected by means of a circuit involving a sensing tube connected to each carry flip-flop. It was shown in 5.6 that,on the average, 5 pulse times (flip-flop reaction times) are required for the complete carry. An alternative scheme is to connect a gate tube to each binary counter which will detect whether an incoming carry pulse would produce a carry and will, under this circumstance, pass the incoming carry pulse directly to the next stage.

This circuit would require at most 39 rise times for the completion of the carry.

We return now to the multiplication operation. In a static accumulator we order simultaneously an addition of the multiplicand or null, a shift, and a complete carry, for each of the 39 steps. In a dynamic accumulator of the second kind just described we order in succession an addition of the multiplicand or null, a complete carry, and a shift, for each of the 39 steps. In a dynamic accumulator of the first kind we can avoid losing the time required for completing the carry at each of the 39 steps. We order an addition by the multiplicand or by null, a shift, and then order one pulsing of the carry gates. This process is repeated 39 times. A simple arithmetical analysis which will be carried out in a later report, shows that at each one of these intermediate stages a single carry is adequate, and that a complete set of carries is needed at the end only. We then carry out the complement corrections, still without ever ordering a complete set of carry operations. When all these corrections are completed and after round-off, described below, we then order the complete carry mentioned above.

5.11. It is desirable at this point in the discussion to consider rules for rounding-off to n - digits. In order to assess the characteristics of alternative possibilities for such properly, and in particular the role of the concept of "unbiasedness", it is necessary to visualize the conditions under which rounding-off is needed.

Every number x that appears in the computing machine is an approximation to another number x', which would have appeared if the calculation had been performed absolutely rigorously. The approximations to which we refer here are not those that are caused by the explicitely

introduced approximations of the numerical-mathematical set up, e.g. the replacement of a (continuous) differential equation by a (discrete) difference equation. The effect of such approximations should be evaluated mathematically by the person who plans the problem for the machine, and should not be a direct concern of the machine. Indeed, it has to be handled by a mathematician and cannot be handled by the machine, since its nature, complexity, and difficulty may be of any kind, depending upon the problem under consideration. The approximations which concern us here, are these: Even the elementary operations of arithmetic, to which the mathematical approximation-formulation for the machine has to reduce the true (possibly transcendental) problem, are not rigorously executed by the machine. The machine deals with numbers of n digits, where n, no matter how large, has to be a fixed quantity. (We assumed for our machine 40 digits, including the sign, i.e., n = 39.) Now the sum and difference of two n - digit numbers are again n - digit numbers, but their product and quotient (in general) are not. (They have, in general, 2n or n digits, respectively.) Consequently, multiplication and division must unavoidably be replaced by the machine by two different operations which must produce n - digits under all conditions, and which, subject to this limitation, should lie as close as possible to the true multiplication and division. One might call them psuedo-multiplication and division; however, the accepted nomenclature terms them as multiplication and division with round-off. (We are now creating the impression as if addition and subtraction were entirely free of such shortcomings. This is only true inasmuch as they do not create new digits to the right, as multiplication and division do. However, they can create new digits to the left, i.e., cause the numbers to "grow out of range". This complication, which is, of course, well known,

is normally met by the planner, by mathematical arrangements and estimates to keep the numbers " within range".)

Thus the round-off is intended to produce satisfactory n-digit approximations for the product x y and the quotient $\frac{x}{y}$ of two n-digit numbers. Two things are wanted of the round-off: (1) The approximation should be good, i.e., its variance from the "true " xy or $\frac{x}{y}$ should be as small as practical; (2) The approximation should be unbiased, i.e., its mean should be equal to the "true" xy or $\frac{x}{y}$.

These desiderata must, however, be considered on the basis of some further comments. Specifically: (a) x and y themselves are likely to be the results of similar round-offs, directly or indirectly inherent, i.e., x and y themselves should be viewed as unbiased n-digit approximations of "true" x' and y' values; (b) By talking of "variances" and "means" we are introducing statistical concepts. Now the approximations which we are here considering are not really of a statistical nature: They are due to the peculiarities (from our point of view: inadequacies) of arithmetic and of digital representation, and are therefore actually rigorously and uniquely determined. It seems, however, in the present state of mathematical science, rather hopeless to try to deal with these matters rigorously. Furthermore, a certain statistical approach, while not truly justified, has always given adequate practical results. This consists of treating those digits which one does not wish to use individually in subsequent calculations, as random variables, with equiprobable digital values, and of treating any two such digits as statistically independent (unless this is patently false).

These things being understood, we can now undertake to discuss round-off procedures, realizing that we will have to apply them to the multiplication and to the division.

Let $x = (.\xi_1 \cdots \xi_n)$ and $y = (.\eta_1 \cdots \eta_n)$ be unbiased approximations of x' and y'. Then the "true" $xy = (.\xi_1 \cdots \xi_n \; \xi_{n+1} \cdots \xi_{2n})$ and the "true" $\frac{x}{y} = (.w_1 \cdots w_n \; w_{n+1} w_{n+2} \cdots)$ (this goes on in infinitum!) are approximations of $x'y'$ and $\frac{x'}{y'}$. Before we discuss how to round them off, we must know whether the "true" xy and $\frac{x}{y}$ are themselves unbiased approximations of $x'y'$ and $\frac{x'}{y'}$. xy is indeed an unbiased approximation of $x'y'$, i.e., the mean of xy is the mean of x $(= x')$ times the mean of y $(= y')$, owing to the independence assumption which we made above. However, if x and y are closely correlated, e.g. for $x = y$, i.e., for squaring, there is a bias. It is of the order of the mean square of $x-x'$, i.e., of the variance of x. Since x has n digits, this variance is about $\frac{1}{2^{2n}}$. (If the digits of x' beyond n are entirely unknown, then our original assumptions give the variance $\frac{1}{3 \cdot 2^{2n}}$.) Next, x/y can be written as $x \cdot y^{-1}$, and since we have already discussed the bias of the product, it suffices now to consider the reciprocal y^{-1}. Now if y is an unbiased estimate of y', then y^{-1} is not an unbiased estimate of y'^{-1}, i.e., the mean of y's reciprocal is not the reciprocal of y's mean. The difference is $\sim y^{-3}$ times the variance of y, i.e., it is of essentially the same order as the bias found above in the case of squaring.

It follows from all this that it is futile to attempt to avoid biases of the order $\frac{1}{2^{2n}}$ or less. Since we propose to use $n=39$, therefore $\frac{1}{2^{78}}$ $(\sim 3 \cdot 10^{-24})$ is the critical case. Note, that this possible bias level is $\frac{1}{2^{39}}$ $(\sim 2 \cdot 10^{-12})$ times our last significant digit. Hence we will look for round-off rules to n digits for the "true" $xy = (.\xi_1 \cdots \xi_n \; \xi_{n+1} \cdots \xi_{2n})$ and $x/y = (.w_1 \cdots w_n \; w_{n+1} \; w_{n+2} \cdots)$. The desideratum (1) which we formulated previously, that the variance should be small, is still valid. The desideratum (2), however, that the bias should be zero, need, according to

the above, only be enforced up to terms of the order $\frac{1}{2^{2n}}$.

The round-off procedures, which we can use in this connection, fall into two broad classes. The first class is characterized by its ignoring all digits beyond the n-th, and even the n-th digit itself, which it replaces by a 1. The second class is characterized by the procedure of adding one unit in the n+1-st digit, performing the carries which this may induce, and then keeping only the n first digits.

When applied to a number of the form $(.v_1 \cdots v_n\ v_{n+1}\ v_{n+2} \cdots)$ (in infinitum), the effects of either procedure are easily estimated. In the first case we may say that we are dealing with $(.v_1, \cdots, v_{n-1})$ plus a random number of the form $(.0 \cdots 0 v_n\ v_{n+1}\ v_{n+2} \cdots)$, i.e., random in the interval $0, \frac{1}{2^{n-1}}$. Comparing with the rounded off $(.v_1 \cdots v_{n-1} 1)$, we therefore have a difference random in the interval $-1/2^n, 1/2^n$. Hence its mean is 0 and its variance $\frac{1}{3 \cdot 2^{2n}}$. In the second case we are dealing with $(.v_1 \cdots v_n)$ plus a random number of the form $(.0 \cdots 00\ v_{n+1}\ v_{n+2} \cdots)$, i.e. random in the interval $0, \frac{1}{2^{n-1}}$. The "rounded-off" value will be $(.v_1 \cdots v_n)$ increased by 0 or by $1/2^n$, according to whether the random number in question lies in the interval $0, \frac{1}{2^{n+1}}$ or in the interval $\frac{1}{2^{n+1}}, 1/2^n$. Hence comparing with the "rounded-off" value, we have a difference random in the intervals $0, \frac{1}{2^{n+1}}$ and $0, -\frac{1}{2^{n+1}}$, i.e. in the interval $-\frac{1}{2^{n+1}}, \frac{1}{2^{n+1}}$. Hence, its mean is 0 and its variance $\frac{1}{12 \cdot 2^{2n}}$.

If the number to be rounded off has the form $(.v_1 \cdots v_n v_{n+1} v_{n+2} \cdots v_{n+p})$ (p finite), then these results are somewhat affected. The order of magnitude of the variance remains the same, indeed for large p even its relative change is negligible. The mean difference may deviate from 0 by amounts which are easily estimated to be of the order $\frac{1}{2^n} \cdot \frac{1}{2^p} = \frac{1}{2^{n+p}}$.

In division we have the first situation, $x/y = (.w_1 \cdots w_n w_{n+1} w_{n+2} \cdots)$, i.e. p is infinite. In multiplication we have the second one, $xy = (.\xi_1 \cdots \xi_n \xi_{n+1} \cdots \xi_{2n})$, i.e. $p = n$. Hence for the division both methods are applicable without modification. In multiplication a bias of the order of $1/2^{2n}$ may be introduced. We have seen that it is pointless to insist on removing biases of this size. We will therefore use the unmodified methods in this case, too.

It should be noted that the bias in the case of multiplication can be removed in various simple ways. Gauss' famous method of "rounding the ambiguous case to the nearest even number" is one instance of this. However, for the reasons set forth above, we shall not complicate the machine by introducing such corrections.

Thus we have two standard "round-off" methods, both unbiased to the extent to which we need this, and with the variances $1/3\ 2^{2n}$ and $1/12 \cdot 2^{2n}$, that is, with the dispersions $\frac{1}{\sqrt{3}} \frac{1}{2^n} = .58$ times the last digit and $\frac{1}{2\sqrt{3}} \frac{1}{2^n} = .29$ times the last digit. The first one requires no carry facilities, the second one requires them.

Inasmuch as we propose to form the product $x'y'$ in the accumulator, which has carry facilities, there is no reason why we should not adopt the rounding scheme described above which has the smaller dispersion, i.e., the one which may induce carries. In the case, however, of division we wish to avoid schemes leading to carries since we expect to form the quotient in the arithmetic register, which does not permit of carry operations. The scheme which we accordingly adopt is the one in which w_n is replaced by 1. This method has the decided advantage that it enables us to write down the approximate quotient as soon as we know its first (n-1) digits. It will be seen in 6.6.4 below that our procedure for forming the quotient of two numbers will

always lead to a result that is correctly rounded in accordance with the decision just made. We do not consider as serious the fact that our rounding scheme in the case of division has a dispersion twice as large as that in multiplication since division is a far less frequent operation.

A final remark should be made in connection with the possible, occasional need of carrying more than n = 39 digits. Our logical control is sufficiently flexible to permit treating k (= 2, 3, ...) words as one number, and thus effecting n = 39k. In this case the round off has to be programmed for this n. The multiplier produces all 78 digits of the basic 39 by 39 digit multiplication: The first 39 in the accumulator, the last 39 in the register. These must then be manipulated in an appropriate manner. (For details cf. 6.6.3.) The divider works for 39 digits only: In forming x/y, it is necessary, even if x and y are available to 39k digits, to use only 39 digits of each, and a 39 digit result will appear. It seems most convenient to use this result as the first step of a series of successive approximations. The successive improvements can then be obtained by means of the well known iteration formula (cf. 5.4). For k = 2 one such step will be needed, for k = 3, 4 two steps, for k = 5, 6, 7, 8 three steps, etc.

5.12. We might mention at this time a complication which arises when a floating binary point is introduced into the machine. The operation of addition which usually takes about 1/10 of a multiplication time becomes much longer in a machine with floating binary point since one must perform shifts and round-offs as well as additions. It would seem reasonable in this case to place the time of an addition as about $\frac{1}{2}$ of a multiplication. At this rate it is clear that the number of additions in a problem is as important a factor in the total solution time as are the number of multiplications.

5.13. We conclude our discussion of the arithmetic unit with a description of our method for handling of the division operation. To perform a division we wish to store the denominator in the selectron register, the partial remainder in the accumulator and the partial quotient in the arithmetic register. Before proceeding further let us consider the so-called restoring and non-restoring methods of division. Normally when one divides -- for the moment we assume the numerator and denominator are both positive -- one subtracts the denominator from the numerator, or partial remainder, if and only if the former does not exceed the latter. If the denominator is in excess of the numerator a null is put in the quotient, the remainder is shifted one place to the left and the computation proceeds. This so-called restoring scheme utilizes, in a number system to the base m, the digits 0, 1, ..., m-1 in each place in the quotient. The difficulty of this scheme for machine purposes is that the only economical method for comparing two numbers as to size is to subtract one from the other. If the partial remainder r_n were less than the denominator d, one would then have to add d back into r_n-d in order to restore the remainder. Thus at every stage an unnecessary operation would be performed. A more symmetrical scheme is obtained by not restoring. In this method one compares the signs of r_n and d; if they are of the same sign, the denominator is repeatedly subtracted from the remainder until the signs become opposite; if they are opposite, the denominator is repeatedly added to the remainder until the signs again become like. In this scheme the digits that may occur in a given place in the quotient are evidently ± 1, ± 2, ..., $\pm(m-1)$, the positive digits corresponding to subtractions and the negative ones to additions of the denominator to the remainder.

Thus we have two (m-1) digits instead of the usual m digits. In

the decimal system this would mean 18 digits instead of 10. This is a redundant notation. The standard form of the quotient must therefore be restored by subtracting from the aggregate of its positive digits the aggregate of its negative digits. This requires carry facilities in the place where the quotient is stored.

We propose to store the quotient in AR, which has no carry facilities. Hence we could not use this scheme if we were to operate in the decimal system.

The same objection applies to any base m for which the digital representation in question is redundant -- i.e. when $2(m-1) > m$. Now $2(m-1) > m$ whenever $m > 2$, but $2(m-1) = m$ for $m = 2$. Hence, with the use of a register which we have so far contemplated, this division scheme is excluded from the start unless the binary system is used.

Let us now investigate the situation in the binary system. We inquire if it is possible to obtain a pseudo-quotient by using the non-restoring scheme and by using the digits 1,0 instead of 1, -1. Or rather we have to ask this question: Does this bear a simple relationship to the true quotient?

Let us momentarily assume this question can be answered affirmatively and describe the division procedure. We store the numerator initially in A, the denominator in SR and wish to form the quotient in AR. We now either add or subtract the contents of SR into A, according whether the signs in A and SR are opposite or the same, and insert correspondingly a 0 or 1 in the right-hand place of AR. We then shift both A and AR one place left -- note therefore that A as well as AR must now be capable of shifting in either direction -- and proceed, carrying out this process 39 times. Denote this digit, which is 0 or 1, by c. The true digit, which is -1 or 1,

is then 2c-1. However, we record, as indicated, c and not 2c-1.

We answer now the question raised above, namely, how can we correct the pseudo-quotient C now in AR so that it is the correct quotient Q. First we recall that all numbers in the machine -- apart from the sign digit -- are less than 1. Hence in forming N/D, we first must adjust N and D so that the result will be in the machine range. We have at the n-th stage of the division process sketched above, $2^n r_n = 2^n N - 2^{n-1}(2c_0-1)D - 2^{n-2}(2c_1-1)D - \ldots = 2^n N - 2^{n+1}(C_0/2 + C_1/2^2 + \cdots + C_{n+1}/2^n)D + \sum_1^{n-1} 2^i$. With the help of this relation we see that $N = (2C-1+1/2^{39})D + r_{39}$ and hence that $Q = 2C - 1 + 1/2^{39}$. Thus to form Q from C we shift C one place left -- this makes the 2^{-39} position null -- insert 1 in the 2^{-39} position -- note that this is our last mentioned rounding-off rule -- and add 1 to the 2^0 position, which is the sign flip-flop. Thus the 2^0 flip-flop must be changed to a binary counter, i.e., a flip-flop with its two input leads tied together. Note that neither 1 which we added in produces a carry: The 1 in the 2^0 position produces none, since it is in the highest position already. The 1 in the 2^{-39} position produces none since it is added to a 0 there.

6.0. The Control

6.1. It has already been stated that the computer will contain an organ, called the Control, which can automatically execute the orders stored in the Selectrons. Actually, for a reason stated in 6.3, the orders for this computer are less than half as long as a forty binary digit number, and hence the orders are stored in the Selectron memory in pairs.

Let us consider the routine that the control performs in directing a computation. The control must know the location in the Selectron memory of the pair of orders to be executed. It must direct the Selectrons to transmit this pair of orders to the Selectron Register and then to itself. It must then direct the execution of the operation specified in the first of the two orders. This usually involves causing the Selectrons to transmit a specified number to the Selectron Register and causing the arithmetic unit to perform some operation on this number. The process must then be repeated with the second order of the order pair. This entire routine is repeated until the end of the problem.

6.2. It is clear from what has just been stated that the control must have a means of switching to a specified location in the Selectron memory, for withdrawing both numbers for the computation and pairs of orders. Since the Selectron memory (as tentatively planned) will hold $2^{12} = 4096$ forty-digit words (a word is either a number or a pair of orders), a twelve-digit binary number suffices to identify a memory location. Hence a switching mechanism is required which will, on receiving a twelve-digit binary number, select the corresponding memory location.

The type of circuit we propose to use for this purpose is known as a decoding or many-one function table. It has been developed in various forms independently by J. Rajchman and P. Crawford. It consists of n flip-flops which register an n digit binary number. It also has a maximum of 2^n output wires. The flip-flops activate a matrix in which the interconnections between input and output wires are made in such a way that one and only one of 2^n output wires is selected (i.e., has a positive voltage applied to it). These interconnections may be established by means of resistors or by means of non-linear elements (such as diodes or rectifiers); all these various methods are under investigation. The Selectron is so designed that four such function table switches are required, each with a three digit entry and eight (2^3) outputs. Four sets of eight wires each are brought out of the Selectron for switching purposes, and a particular location is selected by making one wire positive with respect to the remainder. Since all forty Selectrons are switched in parallel, these four sets of wires may be connected directly to the four function table outputs.

6.3. Since most computer operations involve at least one number located in the Selectron memory, it is reasonable to adopt a code in which twelve binary digits of every order are assigned to the specification of a Selectron location. In those orders which do not require a number to be taken out of or into the Selectrons these digit positions will not be used.

Though it has not been definitely decided how many operations will be built into the computer (i.e. how many different orders the control must be able to understand), it will be seen presently that there will probably be more than 2^5 but certainly less than 2^6. For this reason 6 binary digits are assigned for the order code. (It is worth noting

that a choice of six digits as compared to five, has the advantage that the code may be expressed outside the machine in the octal system.) It thus turns out that each order consists of eighteen binary digits, the first twelve identifying a memory location and the remaining six specifying an operation. It can now be explained why orders are stored in the memory in pairs. Since the same memory organ is to be used in this computer for both orders and numbers, it is efficient to make the length of each about equivalent. But numbers of eighteen binary digits would not be sufficiently accurate for the problems which this machine will solve. Rather, an accuracy of at least 10^{-10} or 2^{-33} is required. Hence the numbers are made long enough to accomodate two orders.

The specification of an operation in an order occurs in binary form, so that another many-one or decoding function table is required to decode the order. This function table will have six input flip-flops and 2^6 or 64 outputs, each corresponding to a possible order. Since there will not be 64 different orders, not all 64 outputs need be provided. However, it is perhaps worthwhile to connect the outputs corresponding to unused order possibilities to a checking circuit which will give an indication whenever a code word unintelligible to the control is received in the input flip-flops.

The function table just described energizes a different output wire for each different code operation. As will be shown later, many of the steps involved in executing different orders overlap. (For example, addition, multiplication, division, and going from the Selectrons to the register all include transferring a number from the Selectrons to the Selectron Register. For this reason it is desirable to have an additional set of control wires, each of which is activated by any particular combination of different code

words. These may be obtained by taking the output wires of the many-one function table and using them to operate tubes which will in turn operate a one-many (or coding) function table. Such a function table consists of a matrix, as before, but in this case only one of the input wires is activated at any one time, while various sets of one or more of the output wires are activated. This particular table may be referred to as the re-coding function table.

The twelve flip-flops operating the four function tables used in selecting a Selectron position, and the six flip-flops operating the function table used for decoding the order, are referred to as the Function Table Register, FR.

6.4. Let us consider next the process of transferring a pair of orders from the Selectrons to the control. These orders first go into SR. The order which is to be used next may be transferred directly into FR. The second order of the pair must be removed from SR (since SR may be used when the first order is executed), but can not as yet be placed in FR. Hence a temporary storage is provided for it. This storage means is called the Control Register, CR, and consists of eighteen flip-flops, capable of receiving a number from SR and transmitting a number to FR.

As already stated (6.1.), the control must know the location of the pair of orders it is to get from the Selectron memory. Normally this location will be the one following the location of the two orders just executed. That is, until it receives an order to do otherwise, the control will take its orders from the Selectrons in sequence. Hence the order location may be remembered in a twelve stage binary counter (one capable of counting to 2^{12}) to which one unit is added whenever a pair of orders is executed. This counter is called the Control Counter, CC.

The details of the process of obtaining a pair of orders from the Selectron are thus as follows. The contents of CC are copied into FR, the proper Selectron location is selected, and the contents of the Selectrons are transferred to SR. FR is then cleared, and the contents of SR are transferred to it and CR. CC is advanced by one unit so the control will be prepared to select the next pair of orders from the memory. (There is, however, an exception from this last rule for the so-called transfer orders, cf. 3.5. This may feed CC in a different manner, cf. 6.6.6.) First the order in FR is executed and then the order in CR is transferred to FR and executed. It should be noted that all these operations are directed by the control itself, not only the operations specified in the control words sent to FR but also the automatic operations required to get the correct orders there.

Since the method by means of which the control takes order pairs in sequence from the memory has been described, it only remains to consider how the control shifts itself from one sequence of control orders to another in accordance with the operations described in 3.5. The execution of these operations is relatively simple. An order calling for one of these operations contains the twelve-digit specification of the position to which the control is to be switched, and these digits will appear in the left-hand twelve flip-flops of FR. All that is required to shift the control is to transfer the contents of these flip-flops to CC. When the control goes to the Selectrons for the next pair of orders it will then go the location specified by the number so transferred. In the case of the unconditional transfer, the transfer is made automatically; in the case of the conditional transfer it is made only if the sign counter of the accumulator registers zero.

6.5. In this report we will discuss only the general method by means of which the control will execute specific orders, leaving the details until later. It has already been explained (5.5) that when a circuit is to be designed to accomplish a particular elementary operation (such as addition), a choice must be made between a static type and a dynamic type circuit. When the design of the control is considered, this same choice arises. The function of the control is to direct a sequence of operations which take place in the various circuits of the computer (including the circuits of the control itself). Consider what is involved in directing an operation. The control must signal for the operation to begin, it must supply whatever signals are required to specify that particular operation, and it must in some way know when the operation has been completed so that it may start the succeeding operation. Hence the control circuits must be capable of timing the operations. It should be noted that timing is required whether the circuit performing the operation is static or dynamic. In the case of a static type circuit the control must supply static control signals for a period of time sufficient to allow the output voltages to reach the steady-state condition. In the case of a dynamic type circuit the control must send various pulses at proper intervals to this circuit.

If all the circuits of a computer are static in character, the control timing circuits may likewise be static, and no pulses are needed in the system. However, though some of the circuits of the computer we are planning will be static, they will not all be so, and hence pulses as well as static signals must be supplied by the control to the rest of the computer. There are many advantages in deriving these pulses from a central source, called the clock. The timing may then be done either by means of counters counting clock pulses or by means of electrical delay lines (an RC circuit is

here regarded as a simple delay line). Since the timing of the entire computer is governed by a single pulse source, the computer circuits will-be said to operate as a synchronized system.

The clock plays an important role both in detecting and in localizing errors made by the computer. One method of checking which is under consideration is that of having two identical computers which operate in parallel and automatically compare each others results. Both machines would be controlled by the same clock, so they would operate in absolute synchronism. It is not necessary to compare every flip-flop of one machine with the corresponding flip-flop of the other. Since all numbers and control words pass through either the Selectron Register or the Accumulator soon before or soon after they are used, it suffices to check the flip-flops of the Selectron Register and the flip-flops of the Accumulator which hold the number registered there; in fact, it seems possible to check the accumulator only (cf. the end of 6.6.2). The checking circuit would stop the clock whenever a difference appeared. Every flip-flop of each computer will be connected to a neon indicating lamp which will be located at a convenient place. In fact, all neons will be located on one panel, the corresponding neons of the two machines being placed in parallel rows so that one can tell at a glance where the discrepancies are.

The merits of any checking system must be weighed against its cost. Building two machines may appear to be expensive, but since most of the cost of a scientific computer lies in development rather than production, this consideration is not so important as it might seem. Experience may show that for most problems the two machines need not be operated in parallel. Indeed, in most cases purely mathematical, external checks are possible: Smoothness of the results, behavior of differences of various types, validity of suitable identities, redundant calculations, etc. All of these methods are usually adequate to disclose the presence or absence of error in toto, their drawback is only that they may not allow the detailed diagnosing and locating of errors at all or with ease. When a problem is run for the first time, so that it requires special care, or when an error is known to be present, and has to be located -- only then will it be necessary as a rule, to use both machines in parallel. Thus, they can be used as separate machines most of the time. The unique feature of such a method of checking lies in the fact that it checks the computation at every point (and hence detects transient errors as well as steady-state ones) and stops the machine when the error occurs so that the process of localizing the fault is greatly simplified. These advantages are only partially gained by duplicating the arithmetic part of the computer, or by following one operation with the complement operation (multiplication by division, etc.), since this fails to check either the memory or the control (which is the most complicated, though not the largest, part of the machine).

The method of localizing errors, either with or without a duplicate machine, needs further discussion. It is planned to design all circuits (including those of the control) of the computer so that if the clock is stopped between pulses the computer will retain all its informa-

tion in flip-flops so that the computation may proceed unaltered when the clock is started again. This principle has already demonstrated its usefulness in the ENIAC. This makes it possible for the machine to compute with the clock operating at any speed below a certain maximum, as long as the clock gives out pulses of constant shape regardless of the spacing between pulses. In particular, the spacing between pulses may be made indefinitely large. The clock will be provided with a mode of operation in which it will emit a single pulse whenever instructed to do so by the operator. By means of this, the operator can cause the machine to go through an operation step by step, checking the results by means of the indicating lamps connected to the flip-flops. It will be noted that this design principle does not exclude the use of delay lines to obtain delays as long as these are only used to time the constituent operations of a single step, and have no part in determining the machine's operating repetition rate. Timing coincidences by means of delay lines is excluded since this requires a constant pulse rate.

6.6. The orders which the control understands may be divided into two groups: Those that specify operations which are performed within the computer and those that specify operations involved in getting data into and out of the computer. At the present time the internal operations are more completely planned than the input and output operations, and hence they will be discussed more in detail than the latter (which are treated briefly in 6.8). The internal operations which have been tentatively adopted are listed in Table 1. It has already been pointed out that not all of these operations are logically basic, but that many can be programmed by means of the others. In the case of some of these operations the reasons for building them into the control have already

been given. In this section we will give reasons for building the other operations into the control and will explain in the case of each operation what the control must do in order to execute it.

In order to have the precise mathematical meaning of the symbols which are introduced in what follows clearly in mind the reader should consult for each new symbol the table at the end of the report in addition to the explanations given in the text.

6.6.1. The addition operations [S(x) → Ac+, S(x) → Ac-, S(x) → Ah+, and S(x) → Ah-] involve the following possible steps: Clearing the Selectron Register; transferring to it the number at S(x); possibly clearing the Accumulator; adding the number in the Selectron Register (or its complement with respect to $1-2^{-39}$) into the Accumulator; possibly adding 2^{-39} to the Accumulator; and then performing a complete carry. (In a static adding circuit the last three steps are combined into one or two.) The operations S(x) → AcM, S(x) → Ac-M, S(x) → AhM, S(x) → Ah-M involve an extra step, namely that of determining the sign of the number in the Selectron Register and using this information to determine whether it is to be added or subtracted into the Accumulator. Building these operations into the control requires only a few tubes to perform this step. This is a small addition and hence would seem of itself to justify adding the operations of plus and minus absolute value. But it should be noted that these operations can be programmed out of the other operations of Table 1 with correspondingly few orders (three for absolute value and five for minus absolute value), so that some further justification for building them in is required. The absolute value order is used frequently in connection with the orders L and R (see 6.6.8), while the minus absolute value order makes the detection of a zero very simple by merely detecting the sign of -N (if $-|N| \geq 0$, then $N = 0$).

6.6.2. The operation of $S(x) \rightarrow R$ involves the following steps: Clearing the Selectron Register and transferring $S(x)$ to it, and clearing the Arithmetic Register and adding the number in the Selectron Register into it. The operation of $R \rightarrow A$ merits more detailed discussion, since there are alternative ways of removing numbers from AR. Such numbers could be taken directly to the Selectrons as well as into the Accumulator, and they could be transferred to the Accumulator in parallel, in sequence, or in sequence parallel. It should be recalled that while most of the numbers that go into AR have come from the Selectrons and hence need not be returned to them, the result of a division and the right-hand 39 digits of a product appear in AR. Hence while an operation for withdrawing a number from AR is required, it is relatively infrequent and therefore need not be particularly fast. It is reasonable to do the transferring at least partially in sequence and to use the shifting properties of the Accumulator and the Register for this. Transferring the number to the Selectron via the Accumulator is also desirable if the dual machine method of checking is employed, for it means that even if numbers are only checked in their transit through the Accumulator, nevertheless every number going into the Selectron is checked before being placed there.

6.6.3. Multiplication involves the following steps: Clearing the Selectron Register and transferring $S(x)$ (the multiplicand) to it; if the multiplicand is negative, connecting the 2^{-39} stage of the Arithmetic Register to the 2^{0} stage of the Accumulator in such a manner that when a right shift occurs the complement of the multiplier with respect to $1-2^{-39}$ is gradually fed into the Accumulator; clearing the Accumulator; 39 steps of (a) adding the multiplicand into the Accumulator if the multiplier digit then being used is 1, with partial or complete carry, and (b) shifting the contents of both the Accumulator and the Arithmetic Register to the right; adding 2^{-39} into the Accumulator if the multiplicand was negative; subtracting the multiplicand into the Accumulator if the multiplier was negative; adding 1 into the Accumulator if both the multiplier and multiplicand were negative;

possibly rounding off the answer held in the Accumulator; and completing the carries if a partial carry system has been employed. If only 39 digits of the product are to be used, the answer should be rounded-off, otherwise not (cf. 5.11). Hence some means of selecting whether or not the answer is to be rounded-off must be provided. This may be done either with a second multiplication order, or (since this choice can probably be made for a complete problem) by means of a manual switch.

It will be noted that since any number held in the Accumulator at the beginning of the process is gradually shifted into the Arithmetic Register, it is impossible to accumulate sums of products in the Accumulator without storing the various products temporarily in the Selectrons. While this is undoubtedly a disadvantage, it cannot be eliminated without constructing an extra register, and this does not at this moment seem worth while.

On the other hand, saving the right hand 39 digits of the answer is accomplished at very little extra equipment, since it only means connecting the 2^{-39} stage of the Accumulator to the 2^{-1} stage of the Arithmetic Register during the shift operation. The advantage of saving these digits is that it makes possible the handling of numbers of any finite size in the computer (cf. 5.11). Any number of 39k binary digits (where k is an integer) and sign can be divided into k parts, each part being placed in a separate selectron position. Addition and subtraction of such numbers may be programmed out of a series of additions or subtractions of the 39-digit parts, the carry-over being programmed by means of $C^{C} \rightarrow S(x)$ and $C^{C'} \rightarrow S(x)$ operations. (If the 2^{0} stage of the Accumulator registers negative after the addition of two 39 digit parts, a carry-over has taken place and hence 2^{-39} must be added to the sum of the next parts.) A similar procedure may be followed in multiplication if all 78 digits of the product of the two 39 digit parts are

kept, as is planned. Since it would greatly complicate the computer to make provision for holding and using a 78 digit denominator, it is planned to program this operation in the manner described in 5.11.

6.6.4. The operation of division $A \div S(x) \rightarrow R$ involves the following steps: Clearing the Selectron Register and transferring S(x) (the denominator) to it; clearing the Arithmetic Register; 39 steps of (a) adding or subtracting the denominator into the remainder (depending upon the comparative signs of the remainder and the denominator) with complete carry and placing 0 or 1 into the Arithmetic Register and (b) shifting the accumulator and the Arithmetic Register to the left; and adding 2^0 and 2^{-39} to the Arithmetic Register.

For the purpose of timing the 39 steps involved in division a six stage counter (capable of counting to $2^6 = 64$) will be built into the control. This same counter will also be used for timing the 39 steps of multiplication, and possibly for controlling the Accumulator when a number is being transferred from or to a tape or film to it (see 6.8).

6.6.5. The substitution operations [$At \rightarrow S(x)$, $Ap \rightarrow S(x)$, and $Ap' \rightarrow S(x)$] involve transferring all or part of the number held in the Accumulator into the Selectrons. This will be done by means of gate tubes connected to the registering flip-flops of the Accumulator. Forty such tubes are needed for the total substitution, $At \rightarrow S(x)$. The partial substitution $Ap \rightarrow S(x)$ and $Ap' \rightarrow S(x)$ require that the left-hand tewlve digits of the number held in the Accumulator be substituted in the proper places in the left-hand and right-hand orders respectively. This may be done by means of extra gate tubes, or by shifting the number in the Accumulator and using the gate tubes required for $At \rightarrow S(x)$.

The importance of the partial substitution operations can hardly be overestimated. It has already been pointed out (3.3) that they allow the

computer to perform operations it could not otherwise perform, such as making use of a function table stored in the Selectron memory. Furthermore, these operations remove a very sizable burden from the person coding problems, for they make possible the coding of the classes of problems, in contrast to coding each individual problem separately. Because Ap→S(x) and Ap'→S(x) are available, any program sequence may be stated in general form (that is, without Selectron location designations for the numbers being operated on), and the Selectron locations of the numbers to be operated on substituted whenever that sequence is used. As an example, consider a general code for n-th order integration of m total differential equations for p steps of independent variable t, formulated in advance. Whenever a problem requiring this rule is coded for the computer the general integration sequence can be inserted into the statement of the problem along with coded instructions for telling the sequence where it will be located in the memory (so that the proper S(x) designations will be inserted into such orders as Cu→S(x), etc.). Whenever this sequence is to be used by the computer, it will automatically substitute the correct values of m, n, p, and Δt, as well as the locations of the boundary conditions and the descriptions of the differential equations, into the general sequence. A library of such general sequences will be built up, and facilities provided on the typewriter for convenient insertion of any of these into the coded statement of a problem (cf. 6.8.4). When such a scheme is used, only the unique features of a problem need be coded.

6.6.6. The manner in which the control shift operations [Cu→ S(x), Cu' → S(x), Cc → S(x), and Cc' → S(x)] are realized has been discussed in 6.4 and needs no further comment.

6.6.7. One basic question which must be decided before a computer is built is whether the machine is to have a so-called floating binary (or

decimal) point. While a floating binary point is undoubtedly very convenient in coding problems, building it into the computer adds greatly to its complexity, and hence a choice in this matter should receive very careful attention. However, it should first be noted that the alternatives ordinarily considered (building a machine with a floating binary point v.s. doing all computation with a fixed binary point) are not exhaustive and hence that the arguments generally advanced for the floating binary point are only of limited validity. Such arguments overlook the fact that the choice with respect to any particular operation (except for certain basic ones) is not between building it into the computer and not using it at all, but rather between building it into the computer and programming it out of operations built into the computer.

Building a floating binary point into the computer will not only complicate the control but will also increase the length of a number and hence increase the size of the memory and the arithmetic unit. Every number is effectively increased in size, even though the floating binary point is not needed in many instances. Furthermore, there is considerable redundancy in a floating binary point type of notation, for each number carries with it a scale factor, while generally speaking a single scale factor will suffice for a possibly extensive set of numbers. By means of the operations already described in the report a floating binary point can be programmed. While additional memory capacity is needed for this, it is probably less than that required by a built-in floating binary point since a different scale factor does not need to be remembered for each number.

To program a floating binary point involves detecting where the first zero occurs in a number in the Accumulator. Since the Accumulator has shifting facilities this can best be done by means of them. In terms of the operations previously described this would require taking the given number

out of the Accumulator and performing a suitable arithmetical operation on it; For a (multiple) right shift a multiplication, for a (multiple) left shift either one division, or as many doublings (i.e. additions) as the shift has stages. However, these operations are inconvenient and time-consuming, so we propose to introduce two operations (L and R) in order that this (i.e. the single left and right shift) can be accomplished directly. These operations make use of facilities already present in the Accumulator and hence add very little equipment to the computer. It should be noted that in many instances a single use of L and possibly of R will suffice in programming a floating binary point. For if the two factors in a multiplication have no superfluous zeros, the product will have at most one superfluous zero (for if $\frac{1}{2} \leqq |X| < 1$ and $\frac{1}{2} \leqq |Y| < 1$, then $\frac{1}{4} \leqq |XY| < 1$). This is similarly true in division (for if $\frac{1}{4} \leqq |X| < \frac{1}{2}$ and $\frac{1}{2} \leqq |Y| < 1$, then $\frac{1}{4} < |\frac{X}{Y}| < 1$). In addition and subtraction any numbers growing out of range can be treated similarly. Numbers which decrease in these cases, i.e., develop a sequence of zeros at the beginning , are really (mathematically) losing precision. Hence it is perfectly proper to omit formal readjustments in this event. (Indeed, such a true loss of precision cannot be obviated by any formal procedure, but, if at all, only by a different mathematical formulation of the problem.)

6.7. Table 1 shows that many of the operations which the control is to execute have common elements. Thus addition, subtraction, multiplication, and division all involve transferring a number from the Selectrons to the Selectron Register. Hence the control may be simplified by breaking some of the operations up into more basic ones. A timing circuit will be provided for each basic operation, and one or more such circuits will be involved in the execution of an order. The exact choice of basic operations will depend upon how the arithmetic unit is built.

In addition to the timing circuits needed for executing the orders of Table 1, two such circuits are needed for the automatic operations of transferring orders from the Selectron Register to Control Register and Function Table Register, and for transferring an order from the Control Register to the Function Table Register. In normal computer operation these two circuits are used alternately, so a binary counter is needed to remember which is to be used next. In the operations $Cu' \rightarrow S(x)$ and $Cc' \rightarrow S(x)$ the first order of a pair is ignored, so the binary counter must be altered accordingly.

The execution of a sequence of orders involves using the various timing circuits in sequence. When a given timing circuit has completed its operation, it emits a pulse which should go to the timing circuit to be used next. Since this depends upon the particular operation being executed, these pulses are routed according to the signals received from the decoding and recoding function tables activated by the six binary digits specifying an order.

6.8. In this section we will consider what must be added to the control so that it can direct the mechanisms for getting data into and out of the computer. Three different kinds of input-output mechanisms are planned. First, a magnetic tape or film storage operated by a servomechanism controlled by the computer; second, some viewing tubes for graphical portrayal of results; and third, a typewriter for feeding data directly into the computer (not to be identified with the typewriter used for making tapes or films).

6.8.1. Though a choice between a magnetic medium (wire or tape) and movie film has not yet been made, for terminological simplicity we shall speak in terms of tape hereafter. Since there already exists a way of transferring numbers between the Selectrons and the Accumulator, the Accumulator

may be used for transferring numbers from and to a tape. Whether the latter transfer will be done serially or in parallel will depend upon the input and output speed, a question to be answered only by experimentation. If it is done serially the shifting facilities of the Accumulator will be employed. Using the Accumulator for this purpose eliminates the possibility of computing and reading or writing on the tapes simultaneously. However, simultaneous operation of the computer and the input-output organ requires additional temporary storage and introduces a synchronizing problem, and hence it is not being considered for the first model.

Since, at the beginning of the problem, the computer is empty, facilities must be built into the control for reading a set of numbers from a tape when the operator presses a manual switch. As each number is read from a tape into the Accumulator, the control must transfer it to its proper location in the Selectrons. The control counter may be used to count off these positions in sequence, since it is capable of transmitting its contents to the Function Table Register. A detection circuit on the control counter will stop the process when the specified number of numbers has been placed in the memory, and the control will then be shifted to the orders located in the first position of the Selectron memory.

It has already been stated that the entire memory facilities of the tapes should be available to the computer without human intervention. This means that the control must be able to select the proper set of numbers from those going by. Hence additional orders are required for the code. Here, as before, we are faced with two alternatives. We can make the control capable of executing an order of the form: Take numbers from positions p to p+s on tape k and place them in Selectron locations v to v+s. Or we can make the control capable of computing and reading or writing simultaneously, there is little

merit in the former scheme so we shall adopt the latter.

The computer must have some way of finding a particular number on a tape. One method of arranging for this is to have each number carry with it its own location designation. A method more economical of tape memory capacity is to use the Selectron memory facilities to remember the position of each tape. For example, the computer would hold the number t_1 specifying which number on the tape is in position to be read. If the control is instructed to read the number at position p_1 on this tape, it will compare p_1 with t_1; and if they differ cause the tape to move in the proper direction. As each number on the tape passes by one unit is added or subtracted to t_1 and the comparison repeated. When $p_1 = t_1$ numbers will be transferred from the tape to the Accumulator and then to the proper location in the memory. Then both t_1 and p_1 will be increased by 1, and the transfer from the tape to Accumulator to memory repeated. This will be iterated, until $t_1 + s$ and $p_1 + s$ are reached, at which time the control will direct the tape to stop.

Under this system the control must be able to execute the following orders with regard to each tape: Start the tape forward, start the tape in reverse, stop the tape, transfer from tape to Accumulator, and transfer from Accumulator to tape. In addition, the tape must signal the control as each digit is read and when the end of a number has been reached. Conversely, when recording is done the control must have a means of timing the signals sent from the Accumulator to the tape, and of counting off the digits if serial transmission is adopted. The 2^6 counter used for multiplication and division may be used for the latter purpose, but other timing circuits will be required for the former.

If the method of checking by means of two computers operating simultaneously is adopted, and each machine is built so that it can operate

independently of the other, then each will have a separate input-output mechanism. The process of making tapes for the computer must then be duplicated, and in this way the work of the person making a tape can be checked. Since the tape servomechanisms cannot be synchronized by the central clock, a problem of synchronizing the two computers when the tapes are being used arises. It is probably not practical to synchronize the tape feeds to within a given digit, but this is unnecessary since the numbers coming into the two Accumulators need not be checked as the individual digits arrive, but only prior to being deposited in the Selectron memory.

6.8.2. Since the computer operates on the binary system some means of decimal-binary and binary-decimal conversion need to be provided. Various alternative ways of handling the problem are under consideration. The conversion can be made in the typewriter and printer, so that the tape only receives binary numbers. Alternatively, each decimal digit could be expressed in a binary code, requiring four binary digits. Then the standard 40 binary digit word in the Selectrons would accomodate only ten decimal digits. There are similar alternatives with respect to binary-decimal conversion.

6.8.3. It is possible to use Selectrons for the viewing tubes, in which case programming the viewing operation is quite simple. The viewing Selectrons can be switched by the same function tables that switch the memory Selectrons. A single order will then serve to graph a point on the memory Selectrons. By means of the substitution operation $Ap \rightarrow S(x)$ and $Ap' \rightarrow S(x)$, six-digit numbers specifying the abscissa and ordinate of the point (six binary digits represent a precision of one part in $2^6 = 64$, i.e. of about 1.5%, which seems reasonable in such a component) can be substituted in this order, which will specify that a particular one of the viewing Selectrons is to be activated.

6.8.4. The typewriter and printer used for preparing tapes and printing from tapes will be built separately from the machine and hence will have independent program controls. We will not discuss these here except to formulate the requirements they must satisfy in order to simplify the preparation and checking of tapes to a degree consistent with the facilities of the computer itself. As was shown in 6.6.5, the statement of a new problem on a tape involves data unique to that problem (and hence put in via the typewriter) interspersed with data found on a previously prepared tape (from the library). Hence the typewriter control must have a means whereby the operator can order any particular set of numbers from the library tape to the new tape. In addition, the operator must be able to print any part of what is on a tape, to check one tape against another automatically, and to easily correct errors found on a tape. It is frequently very convenient to introduce data into a computation without producing a new tape. Hence it is planned to build one simple typewriter as an integral part of the computer. By means of this typewriter the operator can stop the computation, type in a memory location (which will go to the Function Table Register), type in a number (which will go to the Accumulator and then be placed in the first mentioned location), and start the computation again.

6.8.5. There is one further order that the control needs to execute. There should be some means by which the computer can signal to the operator when a computation has been concluded. Hence an order is needed which will tell the computer to flash a light or ring a bell.

TABLE 1

	SYMBOLIZATION		
	Complete	Abbreviated	
1.	S(x)→Ac+	x	Clear accumulator and add number located at position x in the selectrons into it.
2.	S(x)→Ac−	x −	Clear accumulator and subtract number located at position x in the selectrons into it.
3.	S(x)→AcM	x M	Clear accumulator and add absoulte value of number located at position x in the selectrons into it.
4.	S(x)→Ac−M	x −M	Clear accumulator and subtract absolute value of number located at position x in the selectrons into it.
5.	S(x)→Ah+	x h	Add number located at position x in the selectrons into the accumulator.
6.	S(x)→Ah−	x h−	Subtract number located at position x in the selectrons into the accumulator.
7.	S(x)→AhM	x hM	Add absolute value of number located at position x in the selectrons into the accumulator.
8.	S(x)→Ah−M	x h−M	Subtract absolute value of number located at position x in the selectrons into the accumulator.
9.	S(x)→R	x R	Clear register* and add number located at position x in the selectrons into it.
10.	R→A	A	Clear accumulator and shift number held in register into it.
11.	S(x) XR→A	x X	Clear accumulator and multiply the number located at position x in the selectrons by the number in the register, placing the left-hand 39 digits of the answer in the accumulator and the right-hand 39 digits of the answer in the register.
12.	A÷S(x)→R	x −	Clear register and divide the number in the accumulator by the number located in position x of the selectrons, leaving the remainder in the accumulator and placing the quotient in the register.
13.	Cu→S(x)	x C	Shift the control to the left-hand order of the order pair located at position x in the selectrons.
14.	Cu'→S(x)	x C'	Shift the control to the right-hand order of the order pair located at position x in the selectrons.
15.	Cc→S(x)	x Cc	If the number in the accumulator is ≥ 0, shift the control as in Cu→S(x).
16.	Cc'→S(x)	x Cc'	If the number in the accumulator is ≥ 0, shift the control as in Cu'→S(x).
17.	At→S(x)	x S	Transfer the number in the accumulator to position x in the selectrons.
18.	Ap→S(x)	x Sp	Replace the left-hand 12 digits of the left-hand order located at position x in the selectrons by the left-hand 12 digits in the accumulator.
19.	Ap'→S(x)	x Sp'	Replace the left-hand 12 digits of the right-hand order located at position x in the selectrons by the left-hand 12 digits in the accumulator.
20.	L	L	Multiply the number in the accumulator by 2, leaving it there.
21.	R	R	Divide the number in the accumulator by 2, leaving it there.

* Register means arithmetic register.