A Reproduced Copy

OF

N71-25869

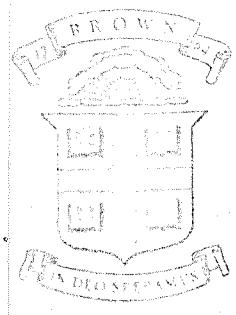
NASA CR-118636

Reproduced for NASA

by the

NASA Scientific and Technical Information Facility

FFNo 672 Aug 65

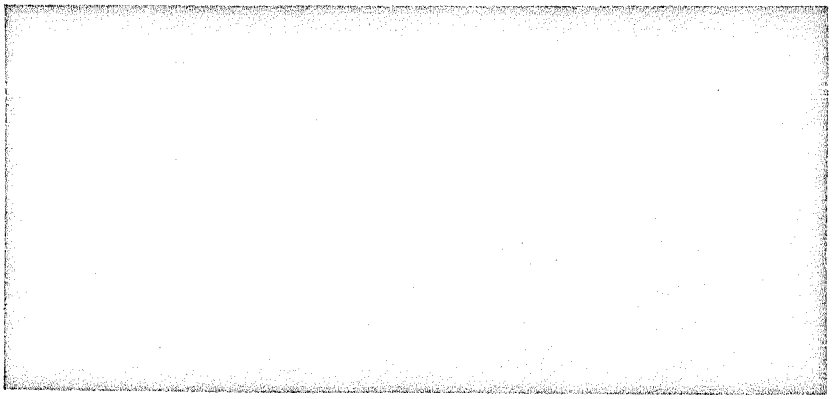


CENTER FOR

COMPUTER AND INFORMATION SCIENCES

AND

DIVISION OF ENGINEERING



BROWN UNIVERSITY

Providence

Rhode Island

COMPUTATION ON FINITE MACHINES*

by

J. E. Savage[1]

* This paper presents results of one phase of research carried out at the Jet Propulsion Laboratory, California Institute of Technology, Pasadena, Calif., under Contract NAS 7-100, sponsored by the National Aeronautics and Space Administration. Portions of the work reported were completed at Brown University with the support of the National Aeronautics and Space Administration under grant NGR 40-002-082 and the National Science Foundation under grant GK-13162.

[1] Consultant to the Communications Systems Research Section, Jet Propulsion Laboratory.

Computation on Finite Machines

by

J. E. Savage
Division of Engineering
and
Center for Computer and Information Science
Brown University
Providence, Rhode Island 02912

ABSTRACT

A measure of the "computational work" required to compute functions is introduced where functions are viewed as the objects defined by programmed general purpose machines or by special purpose machines. Computational work is an equivalent number of logical operations and it is shown that it cannot be less than the combinational complexity of the function computed. A definition of the "computing power" of storage units is given and an exchange relation is established between the number of logic elements, cycles and bits of storage required to compute functions. Four principal results are that 1) sequential storage can be exponentially less efficient than random access storage, 2) functions of more than modest relative complexity should be computed using random access storage, 3) tasks should be assigned to submachines in a computing system on the basis of the complexity of the associated functions and the "work potential" of the submachines, and 4) checking for the validity of programs requires an amount of computational work which is exponential in the maximal length of programs for most finite languages. General remarks are also presented concerning the organization of large computing systems.

I. Introduction

In his 1970 ACM Turing lecture, Minsky [1] voices a complaint which is heard frequently of late, namely, "The trouble with computer science today is an obsessive concern with form instead of content." He then suggests by analogy with physics that "the recognition of exchanges (such as the conservation laws) is often the conception of a science, if quantifying them is its birth." One purpose of this paper is to develop several relations which are almost exchange relations between the inherent complexity of a function and the time, storage capacity and number of logic elements used to compute it. Another purpose is to apply these relations to the study of the relative computing powers of storage units, the effort that must be expended to translate languages and to develop general principles for the design and use of general purpose and special purpose computers.

The starting point for analysis is to observe that computing machines are programmed to compute finite functions, that is, functions whose domain and range are finite. By focusing on the function, the program loses some of its importance since many programs exist for the same function. Next we observe that standard models for computers are required if the relative difficulty of computing functions is to be measured. Therefore, we assume that computers are reduced to sequential machines or to a collection of (not necessarily synchronous) sequential machines and that each machine be constructed from logic elements from some fixed set Ω and from memory cells which are accessible from input and output. As we shall see, this last assumption assures that computation in the models is done by logic elements

and not in part by storage units. It is then possible to define "computational work" as the number of logic uses by a model and to relate this to the combinational complexity of the function computed. This is defined as the minimum number of logic elements required to compute the function with a combinational machine using elements from Ω. These topics are discussed in Section 2.

As mentioned above, the computational work done by a computer is measured by the number of logic uses in a model for that computer which is created using logic elements from Ω and individually accessed memory cells. When models are created for either tape or random access (r.a.) storage units, it is found that the models require a number of logic elements proportional to the number of bits of storage. From this fact, it is deduced in Section 5 that to compute complex functions on general purpose computers with tape or r.a. storage requires that the product of storage space and time be large. Similar but weaker results apply to disk storage. It can also be deduced that the time required to compute a function with tape storage must grow at least as fast as the square root of the combinational complexity of the function whereas a function can always be computed in a time independent of complexity with random access storage if enough storage is available. In fact it can be shown that disk and tape storage are exponentially inferior in computation time to random access storage for most functions of n variables over a fixed alphabet. This information can be used to determine whether to allocate sequential or random access storage to a given problem on the basis of its complexity. It also suggests that Turing-like machines, that is, tape machines with a small finite control, are poor models for the computation

of functions which are more than moderately complex.

Work potential is defined in Section 6 as the maximum amount of work which a machine is capable of doing in a given time. The work potential of random access storage, for example, is proportional to storage capacity divided by cycle time. Work potential can be used to estimate the total storage required in a computing facility and to prevent mismatch between problem and machine.

Another major topic considered here is that of program verification. In Section 7 we consider the computational work required to determine whether or not a string of symbols is a member of given finite language L. We show that this work grows exponentially with the maximum length of strings for most languages over a fixed alphabet. We also bound the work required to determine whether the number of left and right parantheses is equal and we show that the work grows no faster than as the cube of the length of the longest allowable sequence.

An interesting bisociation of ideas is exhibited in Section 8. We combine the relation between work and combinational complexity with the Heisenberg uncertainty relation to show that most Boolean functions of 160 variables cannot be computed in one hour with one kilowatt of power.

In Sections 3 and 4 we examine the combinational complexity of functions and efficiency of computation, respectively. The reader interested in functional complexity will want to read these sections before going on. Others, however, may wish to jump to Section 5 and later sections after reading Section 2.

The "computational work" measure was first introduced in an earlier paper on the complexity of decoders for error correcting codes [2].

2. Machine Models and Computational Work

We shall adopt the view that computing machines are collections of sub-machines which may or may not be driven by a common clock. This is an idealization which allows us to derive results and yet gives a reasonably accurate description of existing computing systems. Each sub-machine is modeled by a sequential machine S with an appropriate input alphabet I, output alphabet O and state set S. Given that $S = \langle S, I, \delta, \lambda, O \rangle$ is in state σ and input i is applied, it moves to a next state $\delta(\sigma,i)$ and produces output $\lambda(\sigma,i)$ where δ and λ are next-state and output functions, respectively. The functions δ and λ are realized by logic and the machine state is stored in memory as indicated in Figure 1. (L designates logic and M designates memory.)

We shall recognize two types of machines, the autonomous sequential machine (ASM) and the driven sequential machine (DSM). ASM's model the conventional mode of operation of general-purpose computers where the initial state carries all the information about the function to be computed. In the ASM, we allow the state set S to have a distinguished state, H, which is called the halting state; we assume that the final output is produced as the machine halts, and that this output is some projection from the final state of the machine. An autonomous machine may produce outputs on every cycle of computation and the outputs will in general be a projection from the current state of the machine. However, since we wish to associate a function with computation by autonomous machines, we shall assume that the results of a computation are contained in the final state σ_F of the machine and designate the machine output by $\lambda(\sigma_F)$. Then, if the initial state of an ASM is σ_O, we say that it computes the function f_s defined by

$$f_s(\sigma_O) = \lambda(\sigma_F). \tag{1}$$

We also denote by $T_s(\sigma_o)$, the number of cycles executed by S with initial state σ_o.

A DSM will be given T inputs $x_1, x_2, \ldots, x_T$, some of which may be fixed, and have an initial state σ_o, which may or may not be fixed. We assume that the machine produces T outputs $y_1, y_2, \ldots, y_T$ and that it halts at the end of the T^{th} cycle. Thus, it computes a function f_s defined by

$$f_s(\sigma_o, x_1, x_2, \ldots, x_T) = (y_1, y_2, \ldots, y_T) \tag{2}$$
$$= (\lambda(\sigma_o, x_1), \lambda(\sigma_1, x_2), \ldots, \lambda(\sigma_{T-1}, x_T))$$

where

$$\sigma_i = \delta(\sigma_{i-1}, x_i) \tag{3}$$

The driven machine may be used to model computation by an autonomous machine since the output $\lambda(\sigma_F)$ can be transferred to auxiliary storage at the end of computation and the machine allowed to run until it completes T_{max} cycles where

$$T_{max} = \max_{\sigma_o} T_s(\sigma_o) \tag{4}$$

To complete the DSM model for the ASM, the appropriate amount of logic must be added to transfer the value $\lambda(\sigma_F)$ to auxiliary storage at the end of $T_s(\sigma_o)$ cycles.

A <u>standard form</u> for a sequential machines is needed if the relative difficulty of computing functions is to be measured. To achieve this end, we assume that a machine is to be constructed of logic elementes from some fixed universal set Ω and from compatible and individually accessible memory cells. The set Ω might contain the set of binary gates with two inputs ("fan-in" of 2) or it might be the single element universal set consisting of the two-input NAND. The memory cells can be viewed as delay lines which store one letter

from the input-output alphabet of the logic elements in Ω. Each cell is assumed to have accessible input and output leads.

If a machine is not of standard type, we create a standard model for it to assess the amount of "computational work" it performs to compute a function. When this is done in Section 5 for several types of storage units, such as sequential and random access units, we acquire important new information about storage-time tradeoffs.

Assume now that the function f is computed in T cycles by a DSM which is in standard form and which has X logic elements. Then, such a DSM does a computational work

$$W = XT$$

to compute f. The combinational complexity of f, $C_\Omega(f)$, is defined as the minimum number of logic elements from Ω required to compute f with a combinational machine (a DSM with T = 1). We then have

THEOREM 1 Let $S_1, S_2, \ldots, S_k$ be k interconnected DSM's with X_ℓ logic elements, $1 \le \ell \le k$, and which execute T_ℓ cycles, $1 \le \ell \le k$, respectively, to compute the function f defined on their initial states and the inputs. Then, the computational work W which they perform to compute f must satisfy

$$W = \sum_{\ell=1}^{k} X_\ell T_\ell \ge C_\Omega(f) \tag{5}$$

PROOF Since storage in machine S_ℓ consists of an array of accessible cells, the state output of its logic unit can be fed directly to the state input of a copy of itself. Thus, the combinational machine of Figure 2 can be formed which realizes the function computed by S_ℓ. This new machine has $X_\ell T_\ell$ logic elements and is supplied exactly those inputs which would

be supplied to S_ℓ in time. Since each machine can be stretched this way, a combinational machine with W logic elements can be formed which computes f. Since $C_\Omega(f)$ is the minimal combinational complexity of f, (5) follows. Q.E.D.

There are several reasons for calling W computational work. First, one use of a logic element from Ω can be said to constitute one unit of computational work and the use of X logic elements T times can be said to constitute XT units of work. Second, the work performed by each machine in a collection of machines can be added and the relation given by (5) holds. This says that a minimum amount of work must be done to compute a function f. The dependence on the set Ω is important and two measurements of work must refer to the same set Ω. Usually, one would choose Ω to be a small set of basic elements since if Ω were unlimited it might contain f in which case $C_\Omega(f) = 1$.

It should be noted that (5) applies whether or not the machines $S_1, S_2, \ldots, S_k$ have a common clock, common cycle lengths or run for the same number of cycles. All that is required is that outputs are produced by a machine when needed by dependent machines. We also note that (5) implies that parallel computation on large machines cannot improve on computation time by more than the number of degrees of parallelism if computation on a single machine is done with a nearly minimum amount of work.

We turn now to the ASM and generate an inequality like (5) for it.

THEOREM 1' Let S be an ASM which in standard form has X logic elements and which executes at most T_{max} cycles on any initial state. Let S compute f and assume that the value of f, $\lambda(\sigma_T)$, is a v-tuple. Then,

$$(X + av)T_{max} \geq C_\Omega(f) \tag{6}$$

for a small constant a, if the halting condition is indicated by a binary variable.

PROOF Since the set Ω is universal, some fixed number a of elements from Ω can be combined to form a gate to transfer a digit from the v-tuple $\lambda(\sigma_o)$ to auxiliary storage under control of the halting signal. A total of av elements can be then be added to S to effect the transfer of $\lambda(\sigma_F)$ and to await the completion of T_{max} cycles. The new machine is a DSM with X + av logic elements to which (5) applies. Q.E.D.

Because of this result, we now adopt the convention that <u>every ASM runs a fixed number of cycles T_{max} to compute its associated function</u>. The computational work done by an ASM, then, is defined exactly as it is for the DSM, namely, as the product of the number of logic elements it contains and T_{max}.

3. Combinational Complexity

The combinational complexity of a function f, $C_\Omega(f)$, relative to the logic set Ω is defined above as the minimal number of logic elements with which f can be computed by a combinational machine. While our objective should be to develop simple methods of estimating $C_\Omega(f)$, this is found to be very difficult. Our results in this connection are limited to a test which guarantees a linear lower bound to $C_\Omega(f)$. The bulk of this section is a summary of the limited knowledge now available on combinational complexity.

Let $\sum_a = \{0,1, \ldots, a-1\}$ and assume without loss of generality that this is the input and output alphabet of elements in Ω. If Ω is a universal set of logic elements then any function $f: \sum_a^p \rightarrow \sum_a^q$ can be realized by a combinational machine with components from Ω and $C_\Omega(f)$ is well defined. The class of Boolean functions, namely functions with domain $\sum_2^p$ and range $\sum_2$, is well known and every Boolean function has a unique disjunctive and conjunctive

normal form expansion. In the disjunctive normal form of a function $f(x_1, \ldots, x_p)$, the logical OR is formed of minterms $x_1^{c_1} x_2^{c_2} \ldots x_p^{c_p}$ ($c_i = 0$, 1 and $x_i^0 = x_i$, $x_i^1 = \bar{x}_i$) and a minterm is included if f has value 1 when $x_i = \bar{c}_i$, $1 \le i \le p$. A minterm is the logical AND of variables x_i or their logical INVERSE (also known as negation or NOT). We conclude that the set of two-input AND, OR and NOT gates form a universal set. There are many universal sets Ω with gates which have a fan-in of 2 or more.

A Boolean function can also be given by a truth table as shown in Figure 3. (The disjunctive normal form is also given.) There are 2^p rows in a truth table for a function $f: \sum_2^p \to \sum_2$ so there are 2^{2^p} distinct Boolean functions of p variables. Similarly, there are a^{qa^p} functions $f: \sum_a^p \to \sum_a^q$. Lupanov [3]has shown that most Boolean functions of p variables can be realized with combinational complexity which does not exceed $\frac{2^p}{p}(1+\epsilon)$ where $\epsilon \to 0$ as $p \to \infty$. Here Ω is chosen as the set {AND, OR, NOT} with fan-in of at most 2. This implies that $C_\Omega(f) \le q \frac{2^p}{p}(1+\epsilon)$ for $f: \sum_2^p \to \sum_2^q$ and it can be shown that

$$C_\Omega(f) \le 4 \frac{q\, a^p (1+\epsilon)}{p} \tag{7}$$

for $f: \sum_a^p \to \sum_a^q$, $\epsilon(p) \to 0$ as $p \to \infty$, by creating a binary function for each of these functions and using Lupanov's result.

There are several special classes of functions which are of interest. One is the class of symmetric functions which are functions $f: \sum_a^p \to \sum_a^q$ which satisfy

$$f(x_1, x_2, \ldots, x_p) = f(x_{i_1}, x_{i_2}, \ldots, x_{i_p}) \tag{8}$$

where $(i_1, i_2, \ldots, i_p)$ form a permutation of $(1, 2, \ldots, p)$. There are N_s such functions where

$$N_s = a^q \binom{p+a-1}{a-1} \tag{9}$$

This is equal to 2^{p+1} for Boolean symmetric functions of p variables (a=2, q=1). It can be shown that the Boolean symmetric functions can be realized with

$$C_\Omega(f) \leq \alpha\ p^2 \log_2 p \tag{10}$$

for α a constant and Ω the set of binary connectives with fan-in of 2. The demonstration of this fact uses (5) and follows from the observation that two counters counting to t and t+1 (and containing $\leq \alpha \log_2(t+1)$ logic elements) can be used for p cycles to determine if $x_1, x_2, \ldots, x_p$ contain exactly t 1's. This can be done for each t for which f has value 1 and f formed by taking the OR of the counter outputs.

Another interesting class, the set of linear functions $f: \sum_a^p \rightarrow \sum_a^q$, can be realized by matrix-vector products. A matrix will have pq entries and for large p and q, it is clear that

$$C_\Omega(f) \leq \beta pq \tag{11}$$

for β a constant. Note that there are exactly a^{pq} linear functions.

We now apply some simple counting arguments to obtain lower bounds on the combinational complexity of most functions in a class. These arguments were introduced by Shannon [4] in the study of relay networks. We begin by overbounding the number of combinational circuits which contain C or fewer logic elements from Ω. We then choose a value of C such that the number of circuits that can be created is an asymptotically small fraction of the number of functions to be realized. This value of C is then a lower bound to the combinational complexity of most functions in the class.

Let Ω contain $|\Omega|$ elements and assume that each element of Ω has a fan-in of at most r and $\sum_a$ is the input-output alphabet of Ω. Then, a combinational circuit with p inputs, q outputs and c elements has at most q + rc leads to which signals can be applied. These are the q outputs and the inputs to the c logic elements which do not exceed rc in number. There are at most c + p + a signals corresponding to the p external inputs, the outputs of the c elements and the a constant signals 0,1,...,a-1. Assume that some fixed choice has been made for the c elements. Then, since only one signal can be attached to each lead, there are no more than $(c + p + a)^{(q+rc)}$ different ways to connect signals to leads. Additionally there are at most $|\Omega|^c$ choices for the c logic elements.

LEMMA 1 The number $N_\Omega(C,p,q)$ of distinct combinational circuits with C or fewer logic elements from Ω and having p inputs and q outputs is bounded by

$$N_\Omega(C,p,q) \le \left[|\Omega|^{1/r}(C + p + a)\right]^{(rC+q+1)} \tag{12}$$

where r is the maximum fan-in of elements in Ω, $|\Omega|$ is its size and a is the size of its input-output alphabet.

The proof follows from a straightforward bounding of terms in the sum involved.

To apply this result suppose that we have a class of N functions for which p, q, a are fixed and suppose that $|\Omega|$ and r are also fixed. Then, if we choose $C = C_o$ so that the bound of (12) equals $N^{1-\epsilon}$, $0 < \epsilon < 1$, then the fraction F of these functions with combinational complexity C_o or less cannot exceed F where

$$F \le N_\Omega(C_o,p,q)/N \le N^{1-\epsilon}/N = 1/N^{\epsilon} \tag{13}$$

which is small if N is large. This leads to the following theorem:

THEOREM 2 Most of the functions $f: \sum_a^p \rightarrow \sum_a^q$ in each of the following classes have a combinational complexity which is greater than the indicated C_o for a, r and $|\Omega|$ fixed:

1) Symmetric functions q fixed, $p \gg q$

$$C_o = \frac{(1-\epsilon')}{r} \frac{q(p+1)}{\log_2 p} \qquad a=2 \qquad (14a)$$

$$C_o = \frac{(1-\epsilon')}{r} q \frac{\binom{p+a-1}{a-1}}{\log_2 p} \frac{\log_2 a}{a-1} \qquad a > 2 \qquad (14b)$$

2) Linear functions p,q large

$$C_o = \frac{(1-\epsilon')}{r} \frac{(p\ q)\log_2 a}{\log_2 (p\ q)} \qquad (14c)$$

3) All functions p,q large

$$C_o = \frac{(1-\epsilon')}{r} \frac{q\ a^p \log_2 a}{p \log_2 a + \log_2 q} \qquad (14d)$$

Here ϵ' is fixed and $0 < \epsilon' < 1$.

A comparison of these bounds with the upper bounds mentioned in text shows that the bounds are sharp (In (14d) the results is sharp if $\log_2 q \ll p$). A fair criticism of the above results, however, is that they provide no information concerning the complexity of a given function. One can also criticize on the grounds that one usually does not want to compute most functions $f: \sum_a^p \rightarrow \sum_a^q$ and is interested rather in computing the less complex functions. Nevertheless, the imformation provided by the bounds has applications.

Muller [5] has observed that a change in the logic set Ω results in at most a multiplicative effect on combinational complexity. This follows because every element from the old set can be replaced by a small number of elements from the new logic set. This accounts for the insensitivity of the bounds to the size of Ω.

There is a simple test which can be applied to a function which may yield a linear lower bound to its complexity. We develop it as follows:

DEFINITION A function $f(x_1,x_2,\ldots,x_p)$ is essentially dependent on x_i if there exist values for x_j, $1 \le j \le p$, $j \ne i$, and two distinct values for x_i, namely, a_1 and a_2, such that

$$f(x_1,\ldots,x_{i-1},a_1,x_{i+1},\ldots,x_p) \neq f(x_1,\ldots,x_{i-1},a_2,x_{i+1},\ldots,x_p) \tag{15}$$

Given that $f(x_1,\ldots,x_p)$ is essentially dependent on x_i, the dependence is non-trivial if for each pair $a_1,a_2,a_1 \neq a_2$ there exist values for x_j, $1 \le j \le p$, $j \ne i$ such that (15) applies with equality.

LEMMA 2 If $f:\sum_a^p \rightarrow \sum_a^q$ is essentially and non-trivially dependent on ℓ of its variables, then

$$C_\Omega(f) \ge \ell/r \tag{16}$$

where r is the maximum fan-in of elements in Ω. When $q = 1$ and f is essentially dependent on ℓ' of its variables then

$$C_\Omega(f) \ge (\ell'-1)/(r-1) \tag{17}$$

PROOF The ℓ variables on which f is essentially and non-trivially dependent can effect the value of f but none can uniquely determine f. Therefore each of those inputs in a combinational circuit for f which correspond to these variables must pass through logic elements. Thus, at least ℓ/r elements of fan-in r are needed.

When $q = 1$, each of the ℓ' inputs must pass through logic elements so at least $(\ell'-1)/(r-1)$ are needed. Q.E.D.

It is clear from (17) that the bound of (14a) in Theorem 2 can be improved when $q = 1$. It is also clear that (16) and (17) provide linear lower bounds when $\ell = \ell' = p$. No tests are known which guarantee lower bounds

that grow faster than linearly in p.

A case that has not yet been examined is that of a function whose domain and range are not coincident with tuples over the input-output alphabet of Ω. For such functions a coding will have to be assigned to their variables and values. Such codings may have an important effect on their combinational complexity although it can be demonstrated that the combinational complexity of most functions $f: \sum_a^p \rightarrow \sum_b^q$ will not be substantially affected by a coding when p is large, $\log q/p \ll 1$ and a, b fixed.

4. Computational Efficiency

A computational procedure can be said to be efficient if the function it defines is computed with nearly minimal computational work. In this section we demonstrate that procedures exist for many functions of low as well as high complexity such that they can be computed efficiently by sequential machines. It is not clear whether the same is true of functions which have modest complexity. It may be that they are efficiently computed only by minimal combinational machines.

To substantiate the above, we consider only Boolean functions and suggest that our reasoning applies to other finite functions. Let Ω be the set of binary logic elements with r = 2 inputs and consider the function $f_A(x_1,\ldots,x_p)$ which is the AND of $x_1,\ldots,x_p$. Applying Lemma 2, equation (17) and using the fact that f_A can be realized with p-1 elements from Ω, we have $C_\Omega(f_A) = p-1$. A sequential machine which computes f_A in T cycles is shown in Figure 4. The p values for variables are grouped together ℓ at a time and stored in individually accessible memory cells. (1's are used to fill out the last row.) A group of ℓ values are AND-ed together in an ℓ-input AND and the result is then combined in a 2-input AND gate with a previous product. A total of ℓ 2-input AND gates are used in this machine. If the machine executes

T cycles, we must choose * $\ell = \lceil p/T \rceil$ and the computational work W_A done is given by

$$W_A = XT = \lceil p/T \rceil \, T \tag{18}$$
$$p \leq W_A < 2p$$

This says that f_A can be computed by sequential machines in T cycle, $1 \leq T \leq p$, with an efficiency no worse that about .5.

Another simple function is the minterm $f_B(x_1, x_2, \ldots x_p)$ given by

$$f_B(x_1, x_2, \ldots x_p) = x_1^{c_1} x_2^{c_2} \ldots x_p^{c_p} \tag{19}$$

As shown in Figure 5, f_B can be computed in T cycles by a machine with $2\lceil p/T \rceil$ gates for a computational work W_B where

$$W_B = 2 \lceil p/T \rceil \, T$$
$$2p \leq W_B < 4p \tag{20}$$

and for an efficiency no worse than about .25, since $C_\Omega(f_B) = p\text{-}1$.

Arbitrary Boolean functions are computed directly from their minterm expansions by the machine of Figure 6. This machine executes T = m cycles, where m is the number of minterms, and it has 2p logic elements for a computational work W_c given by

$$W_c = 2p \cdot m \leq 2p \; 2^p \tag{21}$$

since $m \leq 2^p$. Also, each minterm could be calculated in more than one cycle with an additional factor of no more than 2, as seen above. We demonstrated

---

* $\lceil x \rceil$ is the smallest integer greater than or equal to x.

in the last section that most Boolean functions of p variables have

$$C_{\Omega}(f) \tilde{>} 2^p/p \qquad (22)$$

which indicates most Boolean functions can be computed by sequential machines in T cycles, $1 \leq T \leq 2^p$ with an efficiency bounded below by a quantity which is inversely proportional to p^2. It is very probable that this result can be substantially improved.

5. Computation on General Purpose Computers

Storage units which can be directed to read one of many stored words possess the power to compute. In this section we measure the effective computing power of several storage types and determine relations between the time and storage required to compute functions. We also show that tape and disk storage can be exponentially inferior to random access storage in the time required to compute functions. This information can be used in multiprocessing, multiprogramming, or time-shared systems to increase system efficiency. In this section, we assume that the length of a read-write cycle is the same for all storage types and treat the other case in the next section.

Let us describe the state of a storage unit by a pair $(\vec{s}, h)$ where $\vec{s}$ is an M-tuple of words from the alphabet $\sum_w$ and h, $1 \leq h \leq M$, is an integer which is the address of the word in $\vec{s}$ being examined. If the storage unit consists of an array of individually accessible memory cells M = 1 and w is the number of states which the array can assume. In a tape unit the position of the reading head at time t, h_t, is related its position during the previous cycle by $h_t = h_{t-1} + \{0,+1,-1\}$. In a random access unit h_t and h_{t-1} are not necessarily dependent. In a disk unit $h_t = h_{t-1}+1$ if $h_{t-1} < M$ and $h_t = 1$ otherwise. Thus, a disk unit is a tape unit with tape ends joined and which moves one step to the right at the end of each cycle.

We now create models for each of the storage units using logic elements from a set Ω and individually accessible memory cells. Without any great loss of generality we let Ω be the set of binary gates of fan-in 2 and let the cells be binary. A model for a disk unit with alphabet of size w = 2 is shown in Fig. 7; it has 3 logic elements. For alphabets of size w, the model will contain P_d logic elements where

$$P_d = 3 \lceil \log_2 w \rceil \tag{23}$$

Also, P_d must satisfy

$$P_d \geq \lceil \log_2 w \rceil \tag{24}$$

since access to every bit in at least one word is required.

The models for tape and random access (r.a.) units are more complicated. In an r.a. unit an external control produces one of M signals designating the word to be read. We represent such a signal with $n = \lceil \log_2 M \rceil$ bits and create a circuit of n inputs whose 2^n outputs are the 2^n different minterms in n variables. Thus, a given output is 1 if some one pattern n inputs occurs. The minterm outputs are then used to gate inputs and outputs from the storage unit through circuits like that shown in Fig. 8. Here i_1 is the first bit of information to be gated into a cell and it is gated into the cell shown if the minterm T_2 has value 1. If w is the size of word alphabet, T_2 and each other minterm will control $\lceil \log_2 w \rceil$ cells. Thus, the model will contain $4 \lceil \log_2 w \rceil M$ gates for the M words of storage and require $\lceil \log_2 w \rceil$ (M-1) OR gates to combine the outputs of cells. The circuit $\mathcal{M}(n)$ to create the minterms can be iteratively realized as shown in Figure 9 with no more than $2^{n+1}-4 \leq 4(M-1)$ gates. Thus, a model for random access storage can be realized with P_{ra} logic elements where

$$P_{ra} \leq M(4+5 \lceil \log_2 w \rceil) \tag{25}$$

It is also true that

$$P_{ra} \geq M\lceil \log_2 w \rceil \tag{26}$$

since every bit in each of the M words must be accessible through logic.

The difference between tape and r.a. storage is that only two bits of control information are provided to a tape head to direct it to move right, left, or not at all. The position of the head can be retained in dyadic form in an auxiliary storage unit and a change in head position effected by adding +1, -1 or 0 to the integer in storage. The adder can be realized in logic with a number of logic elements proportional to $(\log_2 M)$. The $\lceil \log_2 M \rceil$ bits representing head position are then supplied to the model for random access storage. The tape model has P_t logic elements

$$P_t \leq M(4+5\lceil \log_2 w \rceil) + \alpha(\log_2 M) \tag{27}$$

for some small constant $\alpha > 0$. For the same reasons given above for random access storage,

$$P_t \geq M\lceil \log_2 w \rceil \tag{28}$$

DEFINITION The computing power P of a storage unit is the minimum number of logic elements from Ω needed to simulate the unit with gates from Ω and individually accessible cells.

Bounds on the computing power of disk, random access and tape storage units are given by (23) to (28) when Ω is the set of binary gates with two inputs. These results apply to one track tape units and one loop disk units. Clearly if a tape or disk unit has m tracks or m loops, the bounds given are to be multiplied by m.

The significance of computing power can be demonstrated as follows: Let S be a small sequential machine which acts as a finite control for a storage unit. Let S in standard form have X elements and as stated at the end of Section 2, assume that S and the storage unit execute T cycles to compute f on every point of its domain. Then, from (5) we have

$$(X + P)T \geq C_{\Omega}(f) \tag{29}$$

because S and the storage unit in standard form have X + P logic elements.

THEOREM 3 Consider storage units which have M words of storage over $\sum_w$ and assume that w is a power of 2. Let the total storage measured in bits, $\mathscr{S} = M \log_2 w$ be much larger than the equivalent number of logic elements X in the associated control S. Then, the following relations apply to r.a. storage, m-tape and m-disk storage with equal length tapes and disks

$$\begin{aligned} &\text{random access} && \mathscr{S} T_{ra} \geq C_{\Omega}(f)/[9(1+\epsilon_1)] \\ &\text{tape} && \mathscr{S} T_{t} \geq C_{\Omega}(f)/[9(1+\epsilon_2)] \\ &\text{disk} && T_{d} \geq C_{\Omega}(f)/[3m(1+\epsilon_3)] \end{aligned} \tag{30}$$

Also, on tape machines

$$T_t \geq \sqrt{C_{\Omega}(f)/[9m(1+\epsilon_2)]} \tag{31}$$

Here ϵ_1, ϵ_2, and ϵ_3 approach zero with increasing m and/or $\mathscr{S}$.

PROOF The derivation of (30) follows from the preceeding arguments. Equation (31) follows (30) by observing that $T_t \geq \mathscr{S}/m$ since 1) all m heads are placed at one end of their tapes before the computation begins and 2) if $T_t < \mathscr{S}/m$, the m heads cannot use a total of $\mathscr{S}$ or more bits of storage. Thus, $\mathscr{S} T_t \leq m T_t^2$. Q.E.D.

The first two inequalities of (30) suggest equivalence between random access and tape machines, an equivalence which may hold for functions of

small complexity but certainly cannot hold for most functions as we show below. The last inequality of (30) and that of (31) suggest a very clear superiority of tape over disk, at least for functions for which $C_\Omega(f)$ is large. These results also suggest a hierarchy with disk at the bottom and random access at the top and a marked difference between the time in which each can compute complex functions.

THEOREM 4 For p,q large, most functions $f: \sum_a^p \to \sum_a^q$ will require $\mathscr{S}_o$ or more bits of storage for their computation on general purpose computers, where

$$\mathscr{S}_o = (1-\epsilon) q\, a^p \log_2 a \tag{32}$$

and $0 < \epsilon < 1$, ϵ fixed. Also, any of these functions can be computed (with unlimited storage) by a general purpose r.a. machine in T_{ra} cycles, where

$$T_{ra} = (p+q) \frac{\log_2 a}{\log_2 w} \tag{33}$$

but most of them will require

$$\begin{aligned} T_t &\geq \mathscr{S}_o/m \\ T_d &\geq \mathscr{S}_o/m \end{aligned} \tag{34}$$

on general purpose m-tape and m-disk machines, respectively, when p and q are large.

PROOF The storage required to compute f on a general purpose machine cannot be less than the storage required to describe f with a program. But at most $\sum_{j=1}^{M} w^j$ distinct programs (sequences) can be given of length M or less. This sum is no more than $2w^M$ for $w \geq 2$ and if we choose M to satisfy

$$2w^M = a^{qa^p(1-\epsilon)} \tag{35}$$

then the fraction of functions which can be programmed with M or fewer words approaches zero with increasing p and q. Thus, most functions will require an M larger than the solution to (35) or $\mathscr{S} = M\log_2 w \geq \mathscr{S}_o$ bits of storage.

Equation (33) follows from the fact that "table look-up" can be used with r.a. storage to compute functions and the time required by this procedure is the time to read the data point on which f is to be computed plus the time to read its value. This is given by (33). The bounds of (34) follow from the fact that $T_t \geq \mathscr{S}/m$ and $T_d \geq \mathscr{S}/m$, as discussed above. Q.E.D.

The implications of Theorem 4 are 1) that a lot of storage is required to compute most functions $f:\sum_a^p \rightarrow \sum_a^q$ (which is not unexpected) and 2) that sequential storage can be exponentially inferior to random access storage in its use of computation time.

COROLLARY For p,q large procedures for the computation of most functions $f:\sum_a^p \rightarrow \sum_a^q$ on tape or disk machines must be grossly inefficient. In fact, they will require a computational work bounded by

$$W_t > (1+\epsilon_2)\mathscr{S}_o^2/m$$
$$W_d > (1+\epsilon_3)\mathscr{S}_o^2/m \qquad (36)$$

where ϵ_2, ϵ_3 and $\mathscr{S}_o$ are quantities defined previously. On the other hand, any of these functions can be computed by an r.a. machine with a work

$$W_{ra} \leq \frac{(1+\epsilon_1)}{(1-\epsilon)} \mathscr{S}_o(p+q) \frac{\log_2 a}{\log_2 w} \qquad (37)$$

where ϵ is that chosen for (32) and ϵ_1 has been previously defined.

We conclude that for most functions random access machines are far superior as computers to machines which use sequential storage. The bound on T_t in Theorem 3, equation (31) and the result of (33) however, suggest that

tape and random access may be nearly equivalent for the computation of functions whose complexity is not too great. This in turn suggests that problems might be assigned in a large computing system to each of the storage units on the basis of their complexity. There is no doubt that this point could use further study.

The comparisons made in this section between the several models of storage units have been made under the assumption that the read-write cycle for each is the same. In the next section we consider the "work potential" of machines for which this requirement is relaxed.

6. Work Potential

The potential for computational work by a machine or work potential $W(t)$ is the maximum amount of work of which a machine is capable in t seconds. As we shall see, this measure will give some rough idea of the quantity and type of storage to purchase for a computing facility and will suggest a strategy for the assignment of storage units to problems in a large computing system.

Consider a system consisting of the machines $S_1, S_2, \ldots, S_k$ and assume that they have $X_1, X_2, \ldots, X_k$ logic elements in standard form and that the length of their cycles are $\tau_1, \tau_2, \ldots, \tau_k$. Then, the maximum amount of work of which this system is capable in t seconds is given by

$$W(t) = \sum_{\ell=1}^{k} X_\ell \left(\frac{t}{\tau_\ell}\right) \tag{38}$$

since at most t/τ_ℓ cycles can be completed by S_ℓ in t seconds. If

$$X_\ell/\tau_\ell \gg X_j/\tau_j , \qquad j \neq \ell \tag{39}$$

then S_ℓ may carry the major portion of the work required of the system.

Suppose the k machines $S, \ldots, S_k$ are each general purpose with a large amount of storage. Then,

$$X_\ell \simeq P_\ell \tag{40}$$

where P_ℓ is the computing power of the ℓ^{th} storage unit. Combining (39) and (40), we see that the size of P_ℓ/τ_ℓ, which is proportional to total storage for random access and tape is an important determinant in the amount of work which a machine can produce.

Rules of Thumb

1. If possible, assign tasks to machines on the basis of the size of P_ℓ/τ_ℓ.

2. To prevent one storage unit from assuming most of the work load, choose units to satisfy

$$\frac{P_\ell}{\tau_\ell} \simeq \frac{P_j}{\tau_j} \tag{41}$$

In the absence of other criteria these rules of thumb may prove useful. In particular, it shows the extent to which slow but large tape storage units may be comparable to faster but smaller random access units. Since tape units can be a factor of 100 or 200 times slower than random access (core) units, at least a factor of this size in storage is necessary if tape is to play an important role in general purpose computers.

Work potential can also give some idea of the total storage required of a given type when the maximum allowable run time is fixed and the combinational complexity of the most complex problem the machine is likely to be given is known. Suppose random access storage alone is to be used of cycle length

$\tau = 10^{-6}$ sec. and total storage $\mathcal{S}$. Suppose also that the most complex problem has complexity $C_\Omega(f) = p^4$, $f: \sum_2^p \rightarrow \sum_2$, for $p = 10^4$. Then, if t = 1 hour is to be made available for this problem, we require that

$$\frac{\mathcal{S}}{\tau} t \gtrsim C_\Omega(f) = 10^{16} \tag{42}$$

This implies that

$$\mathcal{S} \gtrsim 2.8 \times 10^6 \text{ bits} \tag{43}$$

of storage will be required or if each word contains 32 bits this would correspond to about 90 thousand words of storage. This is considered a substantial amount of core storage by today's standard. It is interesting to note that at least 2.8×10^8 bits of storage would be required with tape if the tape cycle length is 100 times smaller.

7. Program Translation

We shall use the term "compiler" as the generic name for any machine or program which translates one language into another. Thus, a compiler might translate Fortran into machine language. In this section we show that one problem associated with compiling, namely, verifying that statements are valid, can be very difficult, that is, require a great deal of computational work.

A finite language L of length n over the alphabet $\sum_a$ is a subset of the set of n-tuples, $\{\sum_a\}^n$. To each language L we associate a function $f_L(\vec{x})$ defined by

$$f_L(\vec{x}) = \begin{cases} 1 & x \in L \\ 0 & x \notin L \end{cases} \tag{44}$$

These functions determine whether or not a program is valid and can be difficult to compute as the following theorem shows.

THEOREM 5 For large n and most finite languages $L \subset (\sum_a)^n$, the functions $f_L(\vec{x})$ require a computational work W which is bounded by

$$W \geq \frac{(1-\epsilon)}{r} \quad \frac{a^n}{n \log_2 a} \tag{45}$$

where $0 < \epsilon < 1$ is fixed and r is the fan-in of the logic set used to measure work.

PROOF The proof follows from Lemma 1, equation (12) and the fact that there are 2^{a^n} distinct languages $L \subset (\sum_a)^n$. Q.E.D.

Theorem 5 establishes a good case for limiting the number of programming languages which are developed. If the number becomes too large, it is certain that some will require an enormous amount of work just to verify that programs are valid.

Many higher level languages (which are not modeled as finite languages but used that way) allow the use of left and right parentheses to segment statements. We model this property of languages with the following finite language L_p:

$$L_p = \{\lambda^j \rho^j b^{n-2j} \mid 1 \leq j \leq n/2, \ n \text{ even}\} \tag{46}$$

Here λ designates a left paranthesis, ρ a right paranthesis and b designates a blank.

THEOREM 6 Let $f_{L_p}(\vec{x})$ be computed by a general purpose machine with random access or tape storage with storage alphabet and input alphabet of fixed size w. Then, the work required to compute $f_{L_p}(\vec{x})$, W_p, for large n is bounded by

$$(1-\epsilon)\ n \log_2 n / \log_2 w \leq W_p \leq \alpha\ n^3 / \log_2^2 w \tag{47}$$

for some fixed ϵ, $0 < \epsilon < 1$ and some small constant $\alpha > 0$.

PROOF The work done is $W_p = X_p T_p$, where X_p includes all logic elements in the standard form of the general purpose tape or random access machine. Then, $X_p \geq \mathcal{A}$, the total storage outside of that in the small and fixed finite control. We now show that $\mathcal{A} \geq (1-\epsilon) \log_2 n$.

Suppose that the n symbols in a string from L_p are read and that the total number of states which the machine can be in is no more than $n^{1-\epsilon'}$. The machine must execute T_p cycles, where $T_p \geq n/\log_2 w$, to read the n symbols so let $s_1, s_2, \ldots, s_{T_p}$ be the sequence of states assumed by the machine. Clearly, for n large there must be at least two states s_i, s_j, with $i < j \leq n/2$ such that $s_i = s_j$. This implies that the response of the machine to $\lambda^j \rho^j b^{2n-j}$ and $\lambda^i \rho^j b^{2n-j}$ must be the same. But this cannot be true so the machine must have more than $n^{1-\epsilon'}$ state. Accounting for the fixed number of states in the finite control, this implies that $\mathcal{A} > (1-\epsilon)\log_2 n$ for $\epsilon > \epsilon'$, n large. Therefore, $X_p T_p \geq \mathcal{A} T_p$ and the lower bound of (47) follows.

To derive the upper bound, we program a tape machine with $\mathcal{A} = n/\log_2 w$ words of storage to execute no more than a number of cycles proportional to n^2 to run up and down a string of n symbols checking off λ's and ρ's in pairs.

Q.E.D.

Undoubtedly, the bounds on W_p given above can be improved. The point of the theorem is that estimates can be made of the amount work which will be necessary to perform important compiling functions.

8. A Quantum - Mechanical Bound on Complexity

In this section we derive a bound on the maximum complexity of any function that can be computed in t seconds with E units of energy. We assume

that the speed of operation of computers is so large that the quantum-mechanical limit is approached and that the computers must be realized with logic elements and individually accessible memory cells (delay lines).

The logic elements have several inputs and we view the action of one element as that of determining the state of each of its inputs by measuring energy levels and computing and registering an output state. We assume (as is true for solids) that to discriminate between two energy levels with separation ΔE requires the expenditure of ΔE units of energy. Then, the maximum number of logic elements X which can be used if no more than E units of energy are to be expended satisfies

$$X \leq E/\Delta E \tag{48}$$

where ΔE is the minimum separation of energy levels in the computer.

Each logic element has a switching time Δt which cannot be less than the time to measure the states of its inputs. Then, the number of cycles which a machine can complete in t seconds, T satisfies

$$T \leq t/\Delta t. \tag{49}$$

Also, ΔE and Δt are related by the Heisenberg uncertainty relation as follows:

$$\Delta E \Delta t \geq h/2\pi \tag{50}$$

where h is Planck's constant. That is, a reliable measurement of an energy difference ΔE requires at least Δt seconds where Δt satisfies (50).

Then, for a function f to be computable in t seconds with E joules requires that

$$C_{\Omega}(f) \leq XT \leq (Et) \times 10^{34} \tag{51}$$

where Ω is the set of logic elements used for the realization of the "quantum-mechanical computers." We take Ω to be the set of 2-input binary logic elements. It is doubtful whether the limit of (51) will ever be approached. Nevertheless, it is instructive to observe the following:

THEOREM 7 Subject to the conditions given above, most Boolean functions $f:\sum_2^p \rightarrow \sum_2$ with $p = 160$ or more cannot be computed in one hour with one kilowatt of power (1 joule = 1 watt-second).

While it is difficult to believe that one would want to compute the most complex Boolean functions of p variables, it is interesting that with $p = 160$ they cannot be computed with a very sizable amount of power in a considerable length of time.

9. Conclusions

The thread that binds the many topics of this paper together is the measure of computational work and its relation to combinational complexity. The numerous applications of this relation have provided new information on the computation of functions on general purpose computers and have led to quantitative comparisons of disk, tape and random access storage. We have discussed the work potential of a computing system, commented on the problem of program verification and summarized the available knowledge on combinational complexity. We have also given an amusing quantum-mechanical bound on complexity.

It is hoped that more detailed models for computers than those offered here can be developed so that more precise and useful results on the computation of functions on finite machines can be derived.

ACKNOWLEDGMENT

The author acknowledges the encouragement and support provided by Drs. E. C. Posner and S. Butman as well as conversations with them and other members of the Communication Systems Research Section, JPL, and conversations with Drs. L. Kleinrock of UCLA and L. H. Harper, U.C. Riverside.

REFERENCES

1. Minsky, M., "Form and Computer Science," Journal, ACM, Vol. 17, No. 2, pp. 197-215, (April, 1970).

2. Savage, J. E., "The Complexity of Decoders - Part II: Computational Work and Decoding Time," to be published, IEEE Trans. on Information Theory, (Jan. 1971).

3. Lupanov, O. B., "A Method of Circuit Synthesis," Izv. VUZ, Radiofizika, No. 1, 120 (1958).

4. Shannon, C. E., "The Synthesis of Two-Terminal Switching Circuits," Bell System Technical J., Vol. 28, pp. 59-98, (1949).

5. Muller, D. E., "Complexity in Electronic Switching Circuits," IRE Trans. on Electronic Computers, Vol. EC-5, pp. 15-19, (March, 1956).

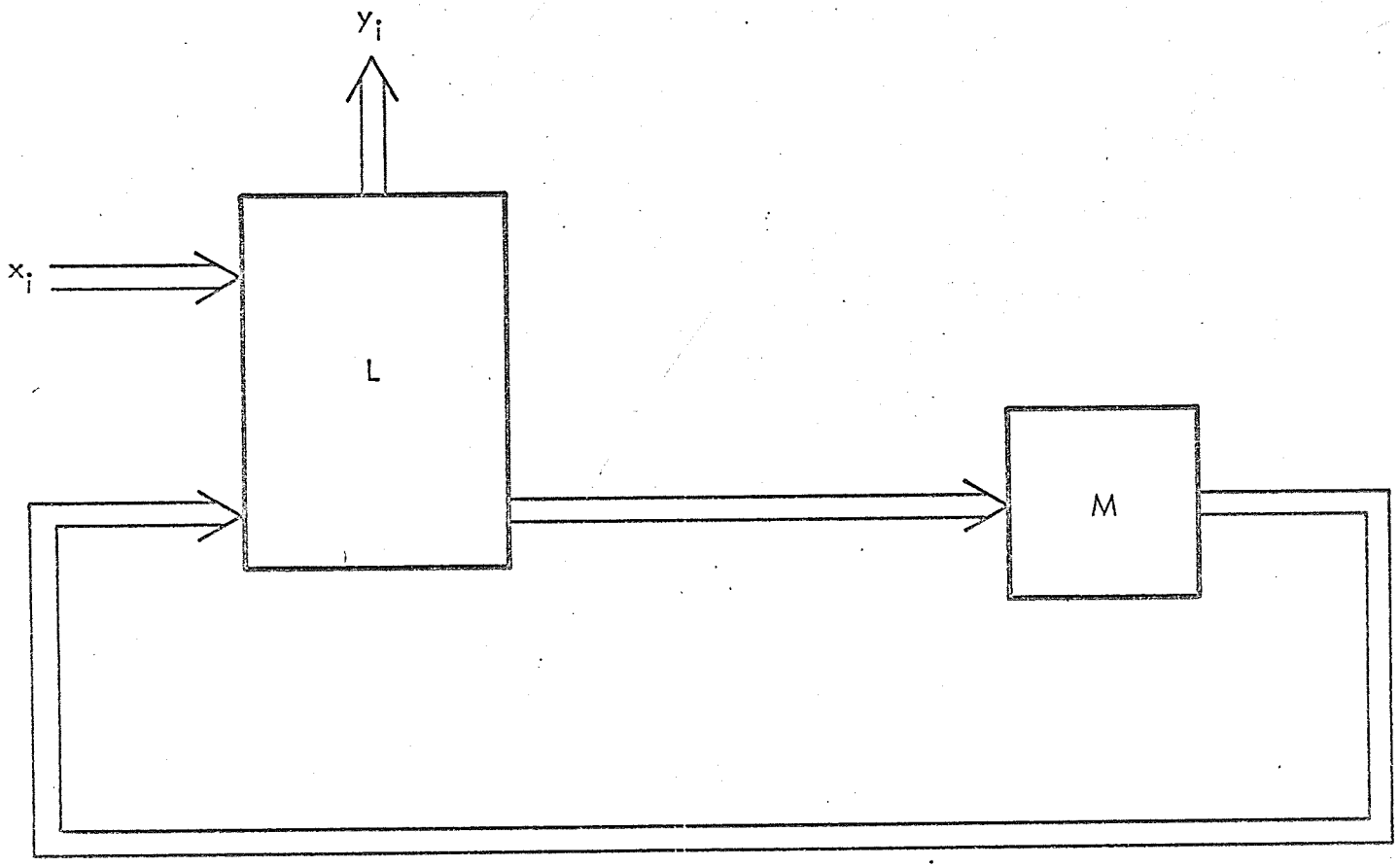


Fig. 1. Sequential Machine Model

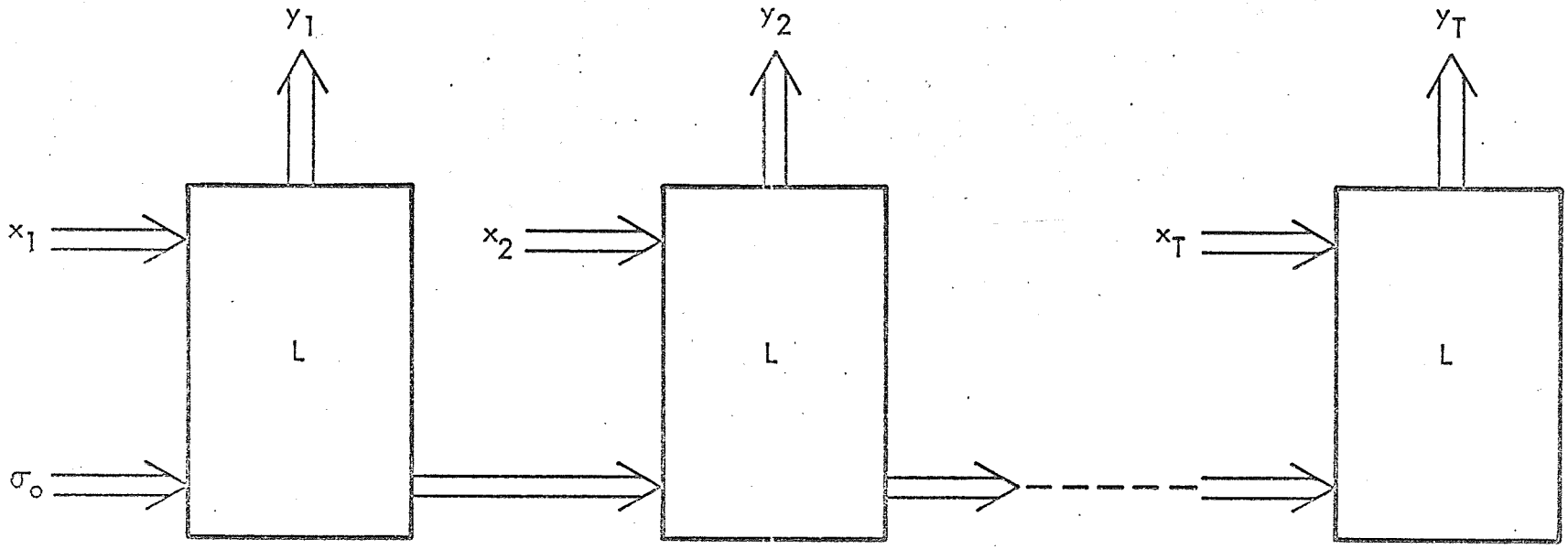


Fig. 2. Combinational Equivalent to Sequential Machine

x_1	x_2	x_3	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

$$f(x_1\, x_2\, x_3) = \overline{x_1}\,\overline{x_2}\, x_3 + \overline{x_1}\, x_2\, x_3 + x_1\,\overline{x_2}\,\overline{x_3}$$

Fig. 3. Tabular Definition of Function and Minterm Expansion

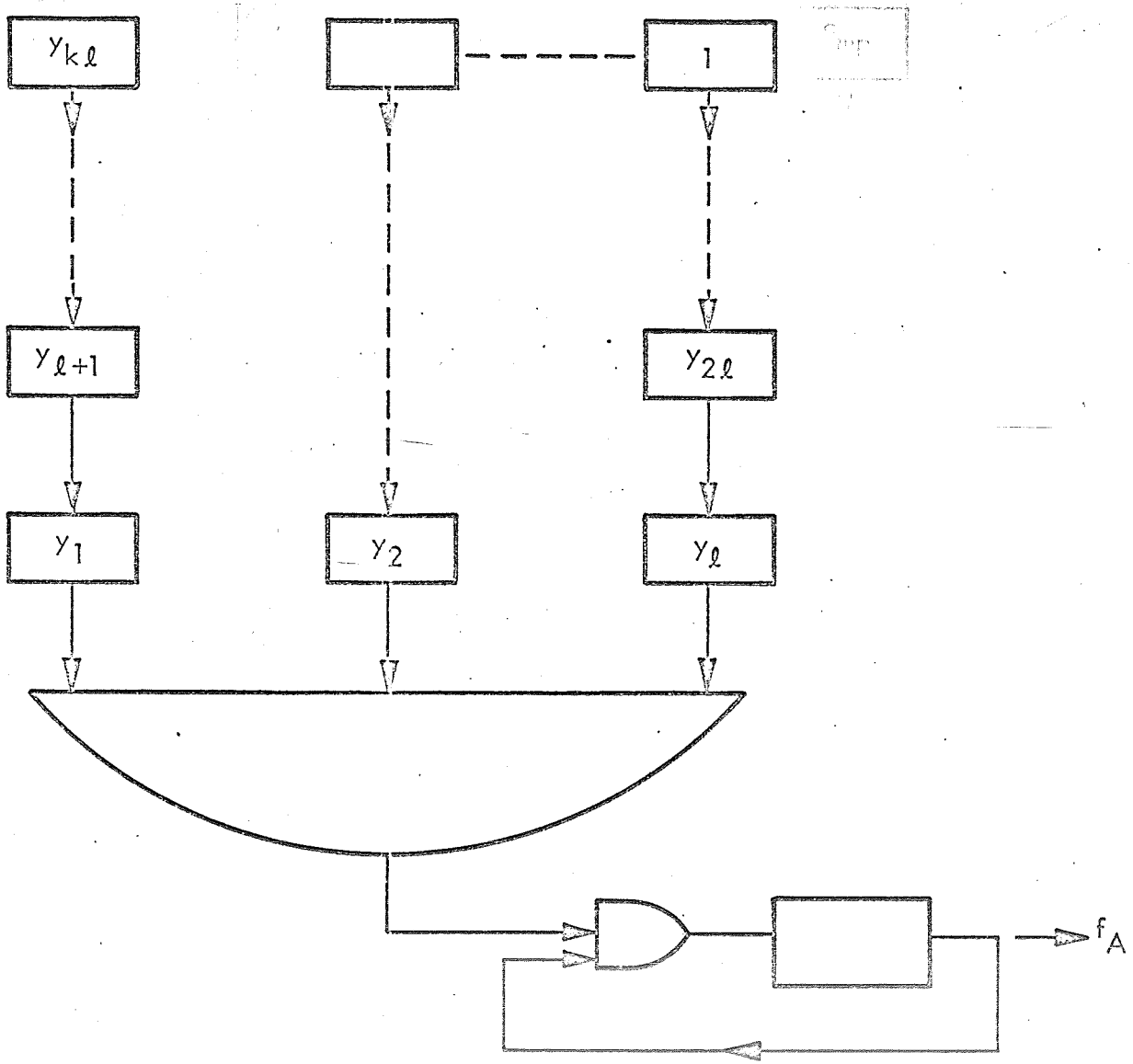


Fig. 4. Sequential Machine Which Computes f_A

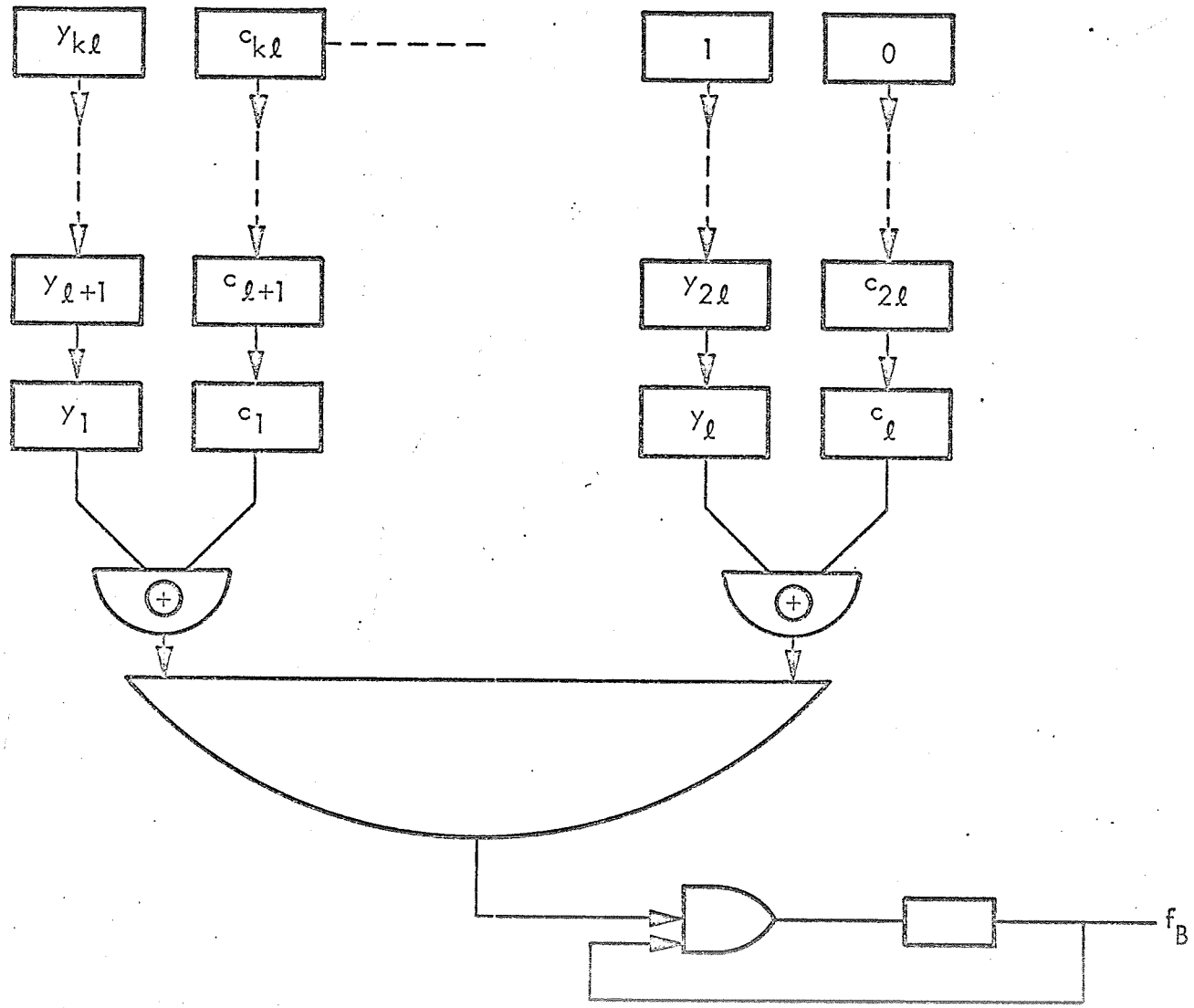


Fig. 5. Sequential Machine Which Computes f_B

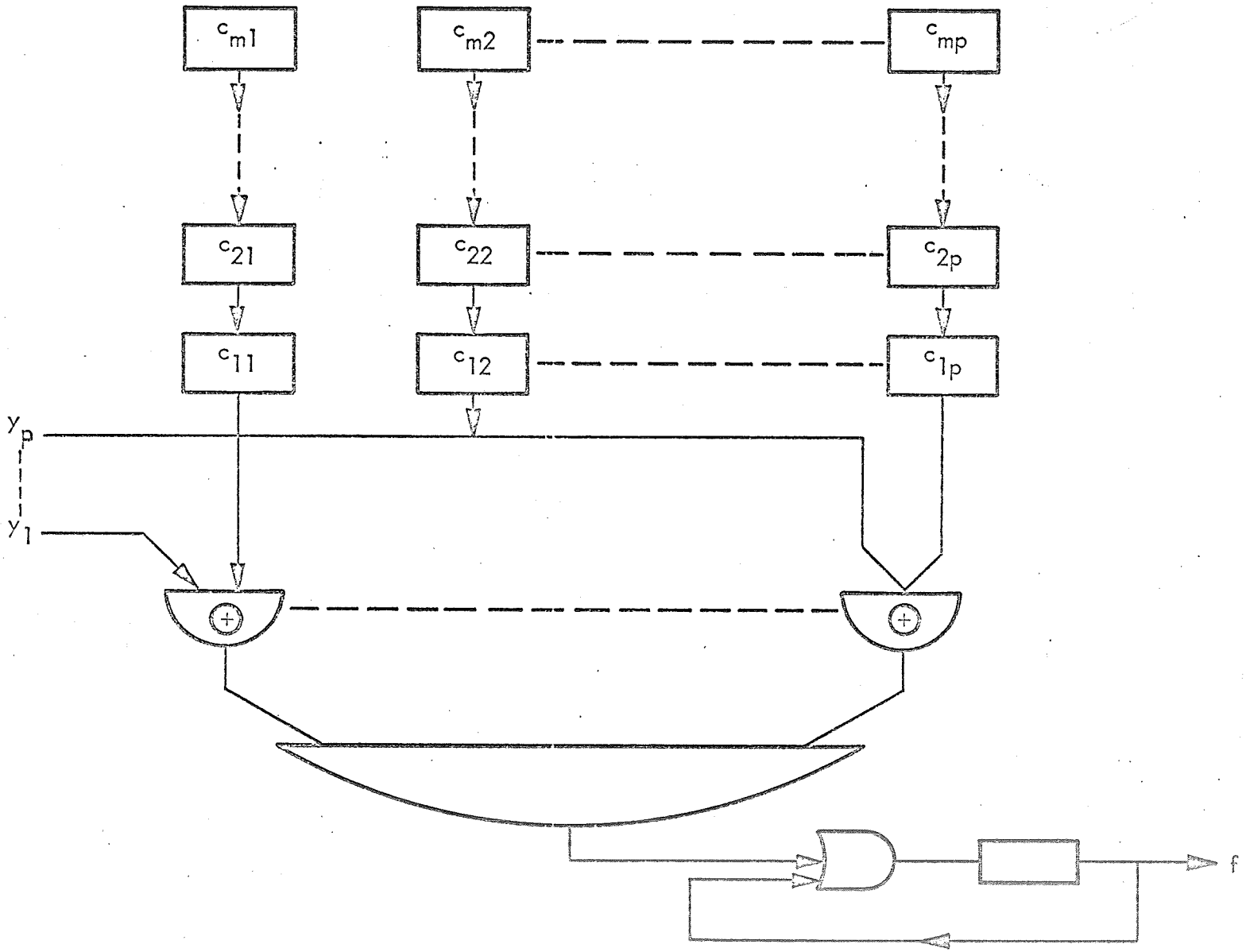


Fig. 6. Sequential Machine Which Computes Boolean Functions

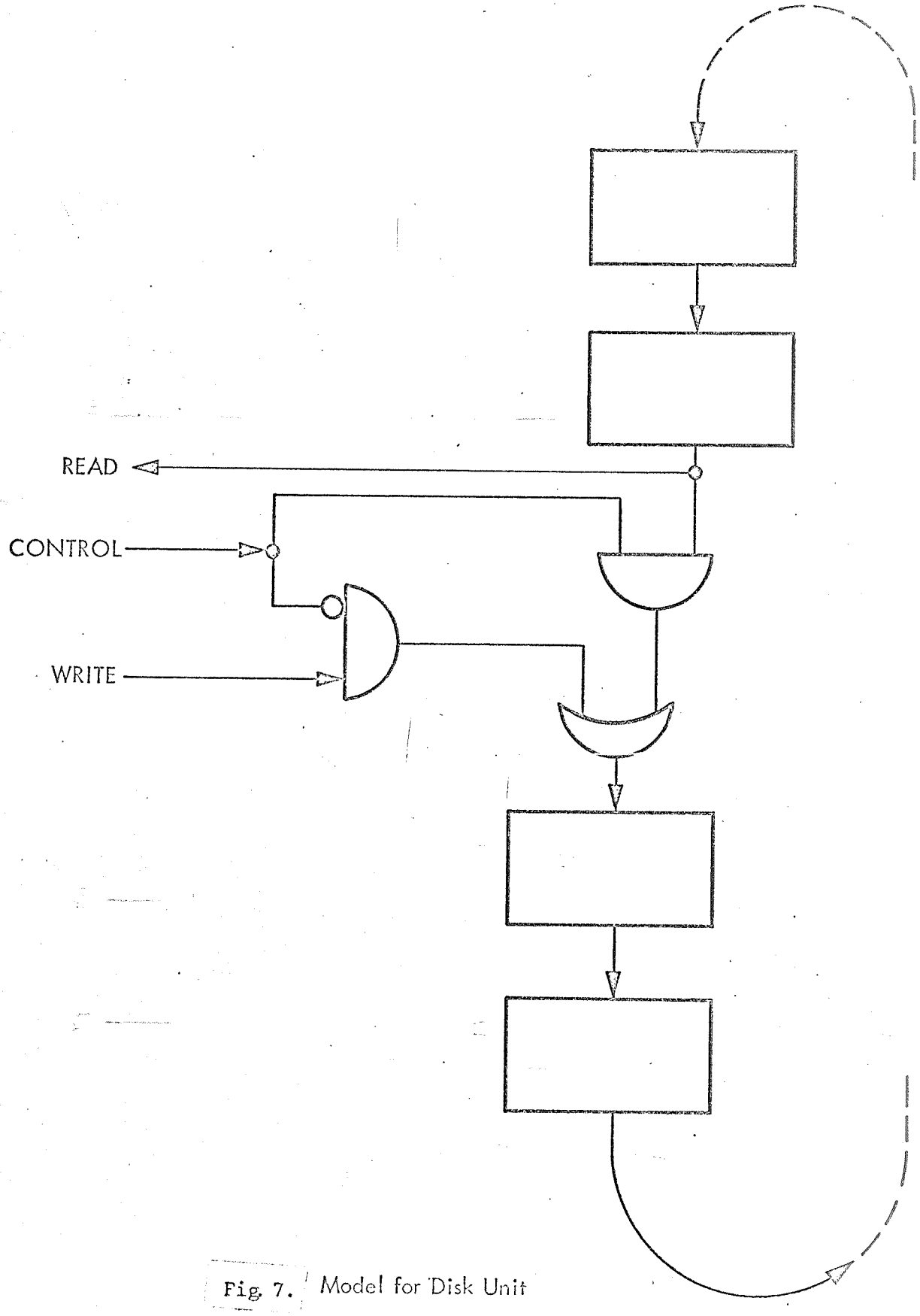


Fig. 7. Model for Disk Unit

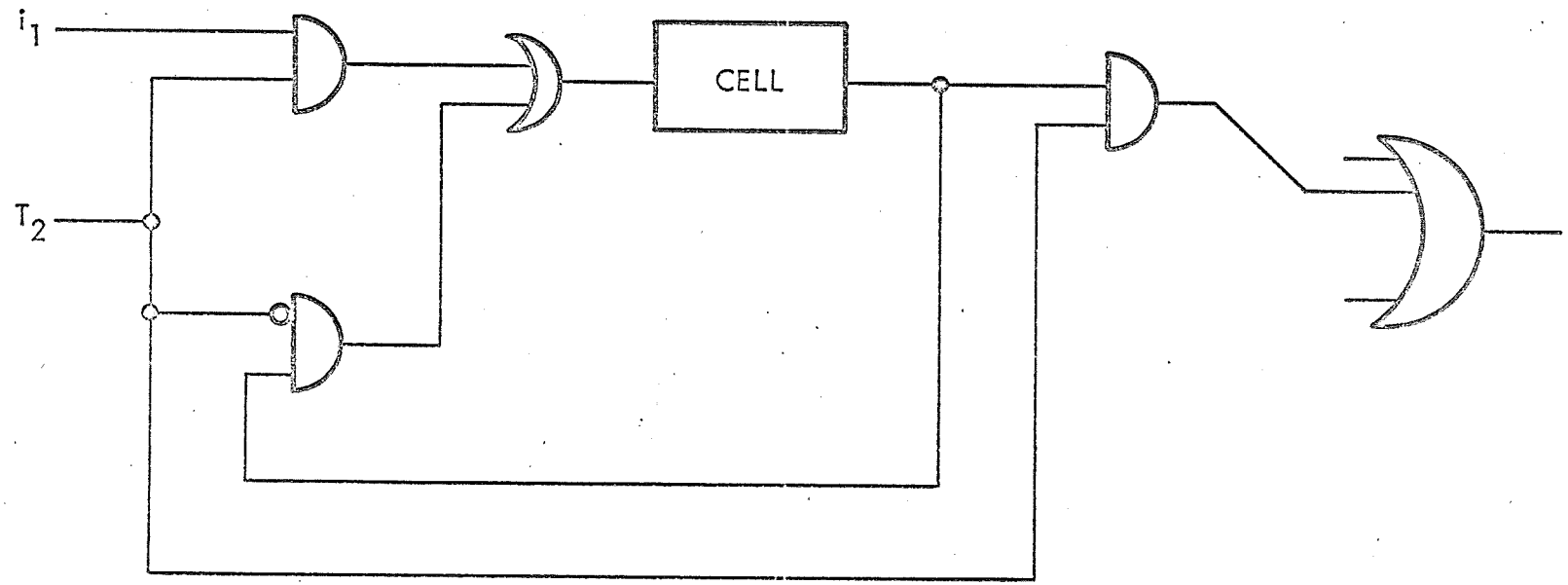


Fig. 8. Storage Cell and Control

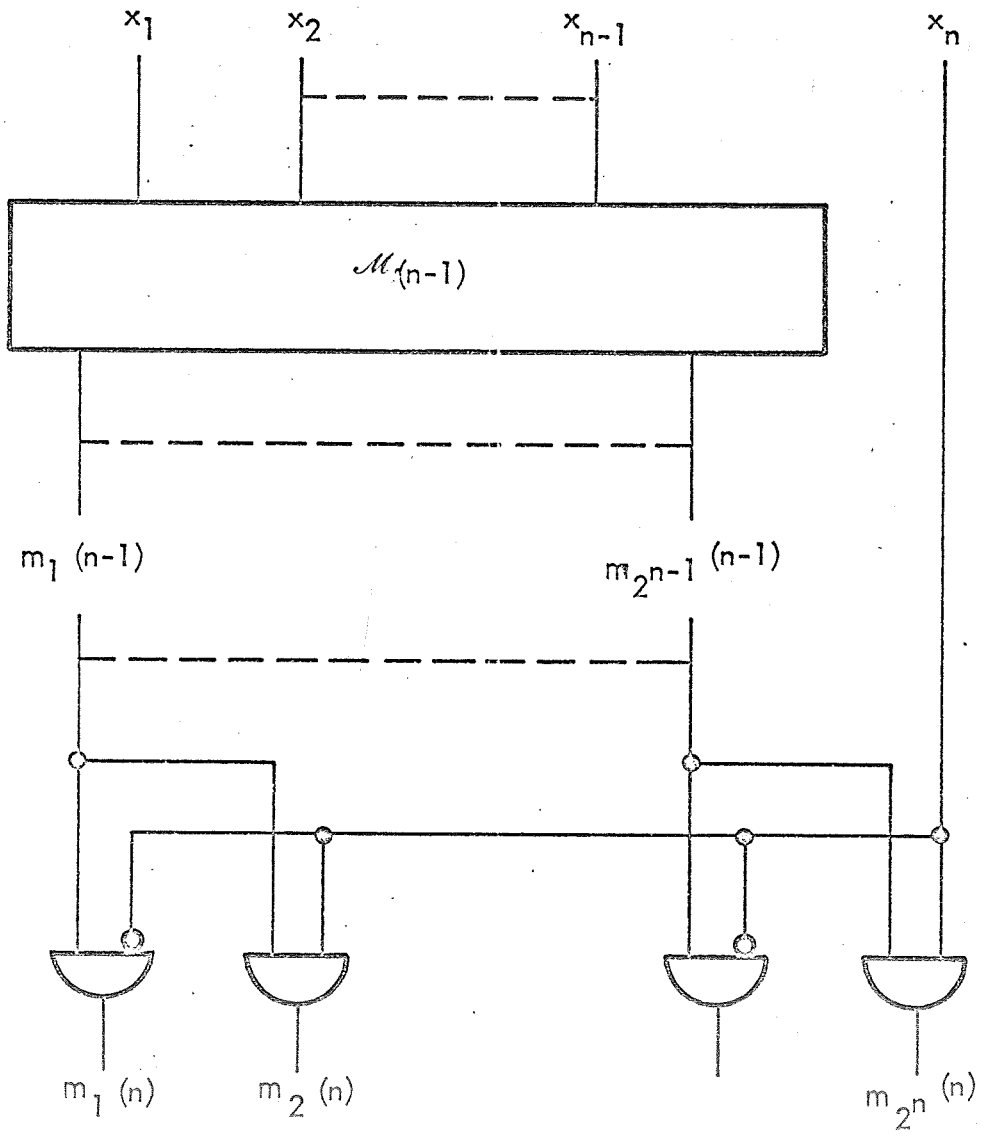


Fig. 9. Circuit for Realizing $\mathcal{M}(n)$ from $\mathcal{M}(n-1)$