The checking of computer logic by simulation on a computer

By M. Lehman, Rayna Eshed, and Z. Netter

This paper, which was originally presented at the Cardiff conference of the B.C.S. in September 1962, describes in detail the techniques used to check the logical design of the Sabrac computer

Introduction

The proposed specification of the Sabrac digital computer was first discussed in papers presented to the International Conference on Information Processing (Lehman, 1959) and to the First Annual Conference of the British Computer Society. Since that time the specification of the machine has changed considerably.

The revised design has produced a machine, now in operation, which despite its low cost and small size has many "Second Generation" features (Lehman, 1961a).

The incorporation of a number of improvements became possible since external circumstances considerably extended the time available for designing the machine. This extension also resulted in a considerable gap between finalization of the logical design and availability of the main rack for commissioning purposes. Thus during the winter of 1960 the question arose whether it would not be possible to make use of this waiting-time to check out the logic of the machine, exposing the inevitable design-errors, by means of a simulation of the logic on another computer. In effect, this implies the continuous solution of a set of some six hundred time-dependent, recursive, symbolic (Boolean) equations subject to some appropriate set of initial conditions. The latter arise from the initial behaviour (when applying voltages) of the machine circuit elements, from the state of any console switches, from the "initial contents" of the simulated store and, when relevant, from the "contents" of the input media.

A first study of the problem suggested that such a complete simulation would not be feasible with either of the machines at that time available in Israel. A less ambitious program, using the Weizac computer of the Weizmann Institute of Science and simulating only a part of Sabrac, did, however, appear feasible.

After outlining the relevant Sabrac characteristics the present paper describes briefly the structure and successful application of the Weizac Simulation. A description is then given of a program incorporating a general-purpose generator which will enable the complete simulation of Sabrac on the Philco 2000 (Transac). The same program, with minor modifications will also permit the simulation of other synchronous computers or computer-like devices or of any system which may be described by a set of time-dependent, recursive, symbolic equations.*

*Since this paper was prepared and presented the authors have become aware of a paper by Stockwell (Stockwell, 1962) describing similar work.

Sabrac

The machine is a 100 Kc/s serial, thirty-six bit device using dynamic (SEAC) type logic. The semiconductor circuits are assembled on a total of 214 printed-circuit, plug-in boards of 23 basic types. The relevant characteristics of these units are listed in Tables 1 and 2.

Those units whose properties are relevant to an understanding of the simulation program have a logic input in disjunctive normal form typified by:

$$G \equiv AB \ldots \vee CD \ldots \vee \ldots$$

All outputs G are available both directly (assertion) and inverted (negation).

Circuits occupying the various positions on the various cards differ in the number of their AND and OR elements (diodes). The largest AND gate has five inputs, the largest OR gate has eight inputs. Units such as registers and counters have a more complex structure, separate equations defining various control and input functions.

Thus any machine element may be described by a lower-case identification letter defining its time-behaviour and by an equation or ordered set of equations. The actual gate structure need not be specified, since the redundancies which occur are of no interest and the number of inputs used is always implicit in the defining equation(s).

The main memory of Sabrac is a 5000-word magnetic drum. In addition, a ferrite-core serial memory (Lehman, 1961b) provides immediate-access storage. The order code includes 32 basic orders which may be modified to yield a total of 197 functions. Input and output are by paper tape.

The Weizac simulation

The Weizac computer is a Princeton type, parallel, one-address machine, now some seven years old. It has a 4000-location, 16 μsec core-store, no index registers, a 50 μsec addition time, and only single-channel paper-tape terminal facilities. At the time when the first simulation was written magnetic tape was not available.

The above characteristics all clearly indicate why a full simulation was not considered feasible on this machine. Early programming studies, however, did suggest that a system involving not more than some fifty equations could be programmed. Hence the decision was taken to simulate the Sabrac multiplier with the threefold purpose of checking the feasibility of writing a simulation, of debugging this relatively complex circuit, and of estimating the value of such a project.

Table 1.—Sabrac units

CARD TYPE	FUNCTION OR NAME	CIRCUIT TYPE	NO. IN MACHINE	NO. OF CIRCUIT EQUATIONS PER CARD	TOTAL NO. OF EQUATIONS
A	Core store driver	—	15	6	90
*B,B**	Logical units with and without delay	1×b,3×1	28	4	112
C	Counter (Modulo 2^7)	—	3	4	12
D	Delay	d	8	1	8
E	Clock pulse repeaters and logical units	3×e,3×1	20	3	60
F	Flip-flop	2×f	29	2	58
*H,H**	Half-delays	3×h	5+2	3	21
*I,I′,I**	Indicator drivers	—	3+2+1	—	—
L	Main logical unit with delay	4×1	46	4	184
M	Core memory	—	3	—	—
P	Pulser (clock strobe generator)	—	5	—	—
X,X, X***	Decoders	8×b	2+1+1	8	32
*Y,Y**	Registers	—	9+9	4+0	36
ZC	Drum clock amplifier	—	1	—	—
ZD	Drum track selector	—	9	4	36
ZH	Hoot control and amplifier	1×f	1	1	1
ZI	Core-store inhibit driver	—	1	3	3
ZK	Core-store constant current driver	—	1	—	—
ZO	Master oscillator and mixer	—	1	—	—
ZQ	Paper-tape synch. control	—	1	3	3
ZR	Core-store read amplifier	—	2	1	2
ZT	Paper-tape punch driver	—	4	4	16
ZV	Voltage control	—	1	—	—
	Total		214		674

Table 2.—Circuit functions

CIRCUIT TYPE	CIRCUIT STRUCTURE
b	Disjunctive Normal Logic (d.n.l.) input stage followed by a nominally delayless output stage
d	d.n.l. followed by three output stages with delays selected to be from one to six bit-times. (b.t., one b.t. is 9 μsecs).
e	Emitter followers. Clock and Strobe pulse repeaters.
f	d.n.l. followed by flip-flop.
h	d.n.l. followed by half-delay ($\frac{1}{2}$b.t.) and output stage.
l	d.n.l. followed by delay (b.t.) and output stage.

Table 3.—Multiplication variants

BIT =	R_2	R_1	$R_0(R_1=0)$	$R_0(R_1=1)$
0	Integer	Single-length	Unrounded	Non-accumulative
1	Fractional	Double-length	Rounded	Accumulative

Like other orders, basic Sabrac multiplication is specified by a particular combination of the five "F-bits" (function-bits) of the 18-bit instruction word. Variations on the order are then defined by different combinations of the three "R-bits". The meaning of these for multiplication is given in Table 3.

The Weizac program required, for each run, the initial specification of three R-bits, a 36-bit multiplicand and a 36-bit fractional, or seven-bit integral, multiplier. This input sufficed to define the first operation. Subsequent multiplications in the same run were then performed on pseudo-random data generated by the program.

After punching out a record of the given or generated data, the program determined directly the true product according to the specification of the option, using the appropriate one of eight subroutines.

The simulation was then performed by successive iteration on the set of symbolic equations describing the units of the multiplier circuit. Each such iteration determines the value of each equation as a function of the values of its input-variables. For inputs originating from a source with a delay, the values are those obtained during the *previous* iteration. Where the source is a non-delay, or buffer-type unit the values are those obtained during the *present* iteration. From this it will be clear that during the "Scan" the buffers must be evaluated before delays. A Sabrac interconnection rule, that "Buffers cannot feed buffers", obviates the need to scan the buffer-type units in any specific order.

In this special-purpose simulation it was not deemed necessary to simulate those phases of the operation which

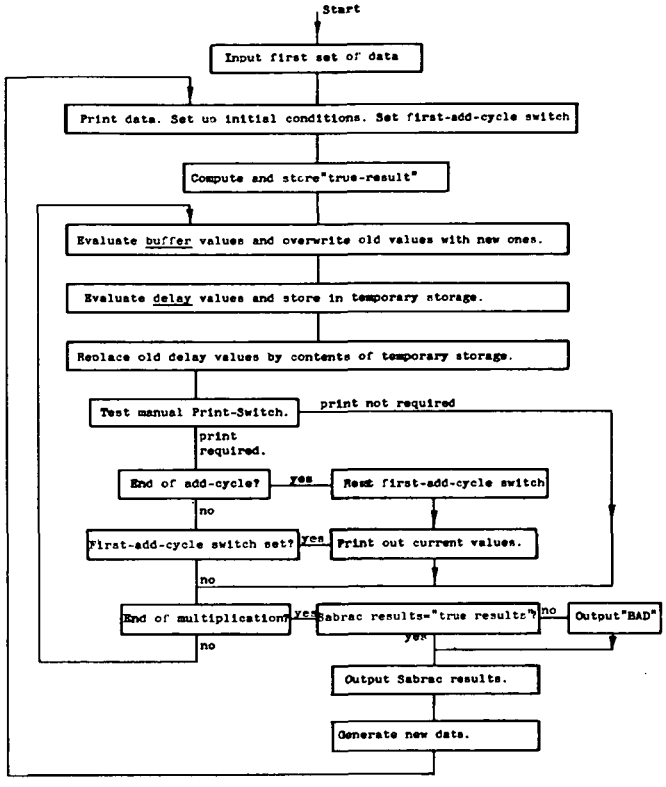

Fig. 1.—The Weizac multiplier-simulation

select, extract and interpret the function bits determining the order (in this case multiplication) to be performed. At the end of these phases the units of the multiplier, apart from those in which the data are recorded, are in a known state. Thus, at the beginning of each multiplication, after reading in or generating the new data, the Weizac locations whose sign bits represent the present values of the units could be preset to the initial state representing the beginning of the multiplication cycle. A flow diagram of the Weizac program is given in Fig. 1.

The simulation process may be divided into two phases, *Scan* and *Interscan*. At the beginning of each scan the current value (state at the output) of each of the thirty or so equations describing the multiplier is stored in the sign-position of one of a group of Weizac store locations (500 to 5FF) termed Current Value (CV_i). For convenience in scanning, the logical complements of these values are stored in a second group of locations (600 to 6FF) termed Current Negated Values (CNV_i). During Scan, the equations are evaluated from the contents of CV_i and CNV_i, for each equation, using an individual order-sequence with a general structure according to the scheme of Fig. 2. Computed values of equations representing buffers, and their negations, immediately replace the previous values in CV_i and CNV_i. The current value of delay-type units, on the other hand, may not be displaced until completion of the scan. Hence their values, as computed, are stored in a third group of locations (700 to 7FF) termed Future Values (FV_i).

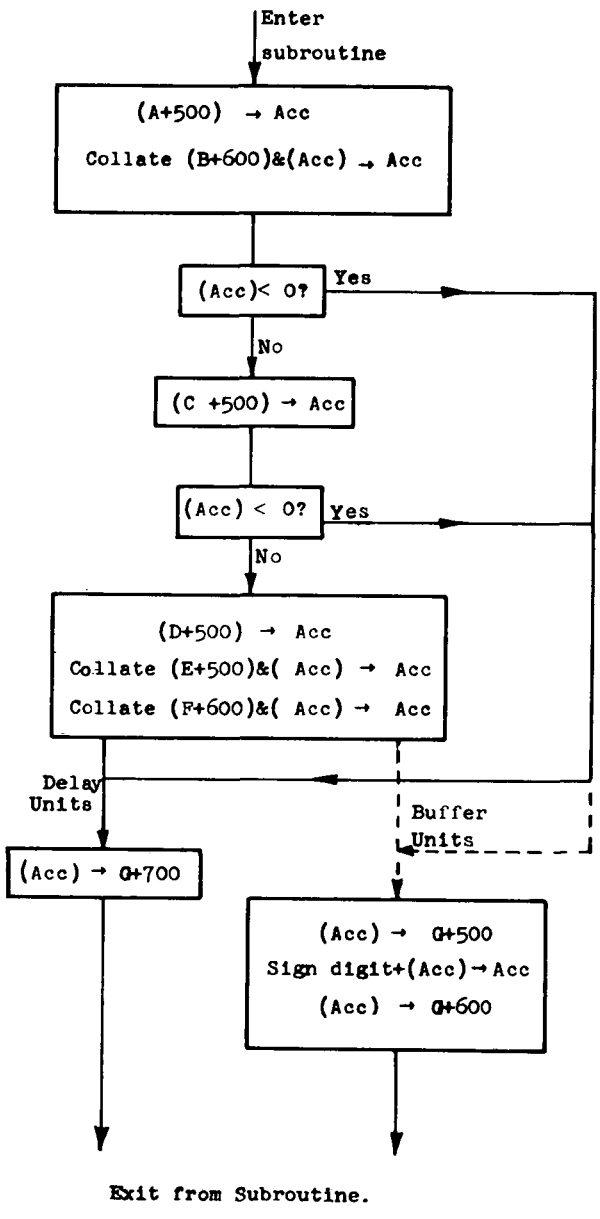

Fig. 2—Weizac evaluation of the typical equation $G = A\bar{B} \vee C \vee DE\bar{F}$

Sabrac Registers and Counters differ from other units in that they store not one bit but 36 and seven bits respectively. Furthermore, they are represented by sets of equations which in addition to the input, define functions such as shift, count and reset as applicable. Thus the Scan and Interscan routines for these units differ somewhat from the more standard structure of other units.

On completion of Scan, the following functions are performed during Interscan. (See Fig. 1.)

(*a*) Transfer all the FV_i to the corresponding CV_i and CNV_i.

(*b*) If required, print out intermediate values.

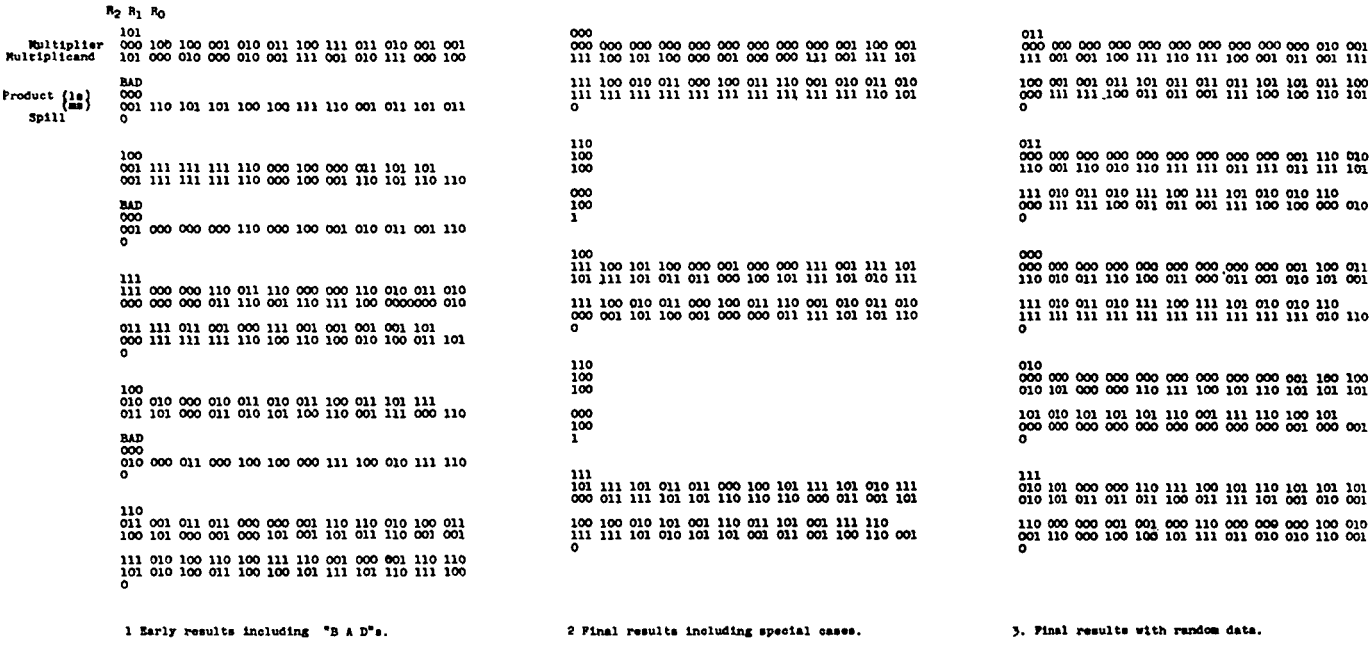

Fig. 3—Sample results from Weizac multiplication simulation

(*c*) If multiplication is not completed, re-enter Scan.

(*d*) If multiplication is completed compare the Sabrac (simulated) and Weizac (computed) products and if these differ print out "BAD".

(*e*) In either case, print out the binary, double-length product, as obtained from the simulation (with least-significant zero suppression). Also print out the contents of the unit indicating Spill. (Spill may occur either on accumulation or on multiplication of minus one by minus one).

(*f*) Generate a new, "random", multiplicand and R-bits (options).

(*g*) Enter a new multiplication using the old multiplicand as the new multiplier.

Results of the Weizac simulation

The original program was written in the space of about fifteen days, of which two days were required for familiarization with Weizac and its code, three days to develop the general flow diagram, and about ten days for coding. Coding of the Scan sections was done directly from multiplier logical-circuit diagrams and equation lists. Debugging of the resultant program required three ten-minute runs on the machine. It is of interest to note that several design errors were already detected during writing and debugging of the program.

During this phase of the work it was realized that a disproportionate amount of time was being spent on output, each eight-minute run requiring only about one minute of computation. The program was therefore modified so that output, other than the actual result, was optional. In the subsequent work it was found that even this minimum print-out generally sufficed to pin-point circuit faults, and on only two occasions was it necessary to rerun a multiplication with full print-out.

Altogether, in a total of eight hours of machine time, twelve faults were discovered and corrected. A total of some 500 multiplications, comprising both random and potentially pathological operands, were performed. A final run of some one hundred multiplications, without a single "BAD" result, suggested that the circuit was substantially error-free. Some results of the Weizac Simulation are shown in Fig. 3.

Conclusions drawn from the Weizac simulation

The relative ease with which the Sabrac multiplier had been debugged provided convincing evidence that the simulation technique was a usable and apparently worthwhile method for the commissioning of logical circuits. Our belief on conclusion of this work, substantiated by subsequent experience in the operation and commissioning of actual hardware, was that the hardware and software approaches are complementary, each capable of making a significant contribution to the commissioning of a new machine. Use of simulation techniques during design, and before a rack is wired, would obviously be of special importance, since correction of errors at this stage involves no changes in hardware.

Thus it appeared well worth-while to go ahead with the writing of a general-purpose simulator for the whole of Sabrac, to be run on the Philco 2000 then on order for the Israel Government. However, on the basis of experience with the Weizac program the following changes appeared desirable.

(1) Programming of the equations should be automated by means of a general-purpose generator.

Input to this would be the list of equations to be simulated, each punched in suitable form on one or more cards. Its output would then comprise the Scan phase of an overall simulation of the system represented by the equations. This approach would not only minimize effort in using the simulator. It would also eliminate the difficulties encountered in the Weizac program, where correction of any equation often entailed modifications to a major part of the Scan, and led to program inefficiency and errors.

In practice the ideal of a general-purpose generator was not realized. The limited high-speed storage originally available to us on Transac, combined with a desire to minimize simulation costs, suggested programming techniques which restricted the size of gates to the maximum occurring in Sabrac (five-way AND, eight-way OR). The punching conventions adopted for generator inputs were also tailored for the known Sabrac structure. In both cases, however, restrictions arising from the adoption of these conventions could easily be removed if required.

(2) In the first simulation program, scanning of individual units, as shown in Fig. 2, was based on a direct evaluation of each equation, that is, Scan was "By Inputs". This method is clearly inefficient since most signals in Sabrac are zero most of the time. It seemed desirable to make use of this known property of the inputs, and the description to follow will introduce the far more efficient method of scanning "By Outputs".

(3) In addition to the generated scan, the Transac program would include sections simulating memory, peripheral, and console equipment. It would also have to include an arrangement for controlling print-out (printed on the Transac line-printer). This should permit the free selection of equations whose values were to be printed, the unit being the behaviour of any equation during one complete word-time. Finally, the program must contain sections which, during any run, would permit checking of the simulated (Sabrac) results against a true result calculated directly from the "order specification".

Scanning by outputs

The sequence of orders in the Weizac routine, outlined in Fig. 2, scans the terms of each equation by inputs. That is, it collates together the contents of the appropriate locations CV_i or CNV_i for each AND-gate combination, until either all inputs to some one gate are found to be in the ONE state (are one) or until all inputs to the equation have been scanned. In view of the fact that during any bit-time most of the equations are zero, the method appears almost to maximise the number of inputs inspected. In a hand-written program an attempt can be made to avoid the most obvious inefficiencies, but in an automatic generator this is more complex.

The alternative, "Scan by Outputs", adopted for the present program, attempts to minimize the work-load at the expense of increasing the storage required.

Each equation E_i may be represented by a storage location EV_i in which the values of the individual bits EV_{ij} represent the state of the various inputs. A second, history, location H_i records the current value of E_i in the most significant bit position H_{i0}. For the purpose of output it also records past values of E_i at bit-time intervals. Given a list of the input-bits to which each equation feeds, the program may proceed to scan this list, filling in the EV_{ij} if and when they occur. That is, if H_{k0} is one, a "one" is introduced into those EV_{ij} which represent an input E_k. Alternatively, where H_{k0} is zero, a "one" is introduced into the EV_{ij} representing the negated feeds $\bar{E}_k$. Since the majority of feeds are assertive, and since most equations have the value zero most of the time, this arrangement is far more efficient than scanning by inputs.

During the second part of the scan the EV_i are inspected for "Five-Ones" groups. According to the existence or non-existence of such a group for each equation, the new one or zero value is inserted into the H_{i0}, the H_i having previously been shifted down one place.

The *TAC* (Translator Compiler) *DORMS* order permits the insertion of a "one" or group of ones held in the Transac *D*-register into corresponding bit positions of a given memory location *M*. Thus, if the EV_i are initially cleared, the Scan procedure outlined above is obtained from a sequence of order pairs:

Transfer $MASK_{f(j)}$ to *D*-register
D or EV_i to EV_i.

In short: $TMD\ MK_{f(j)}$; $DORMS\ EV_i$
which inserts H_{k0} or $\bar{H}_{k0}$ as required into the appropriate EV_i.

In this sequence the address $MK_{f(j)}$ refers to a list of standard bit patterns or masks. The simplest of these each consists of a single bit in one of the 40 positions which an input may occupy. Since not all gates have five inputs, a second group of 40 masks is required to enable the unconditional filling of unused inputs together with the scanning of the last input of an under-sized group. Thus in inserting the value of *A* in the equation

$$G = AB \vee BCD$$

a mask of form

10000,00000, . . . 00000

would be used, whereas the appropriate mask for *D* is

00000,00111,00000, . . . 00000.

Further time and space are saved by the building up of compound masks where a feed appears more than once in the same equation. The mask for inserting *B* in the above equation, for example, becomes

01111,10000,00000, . . .00000.

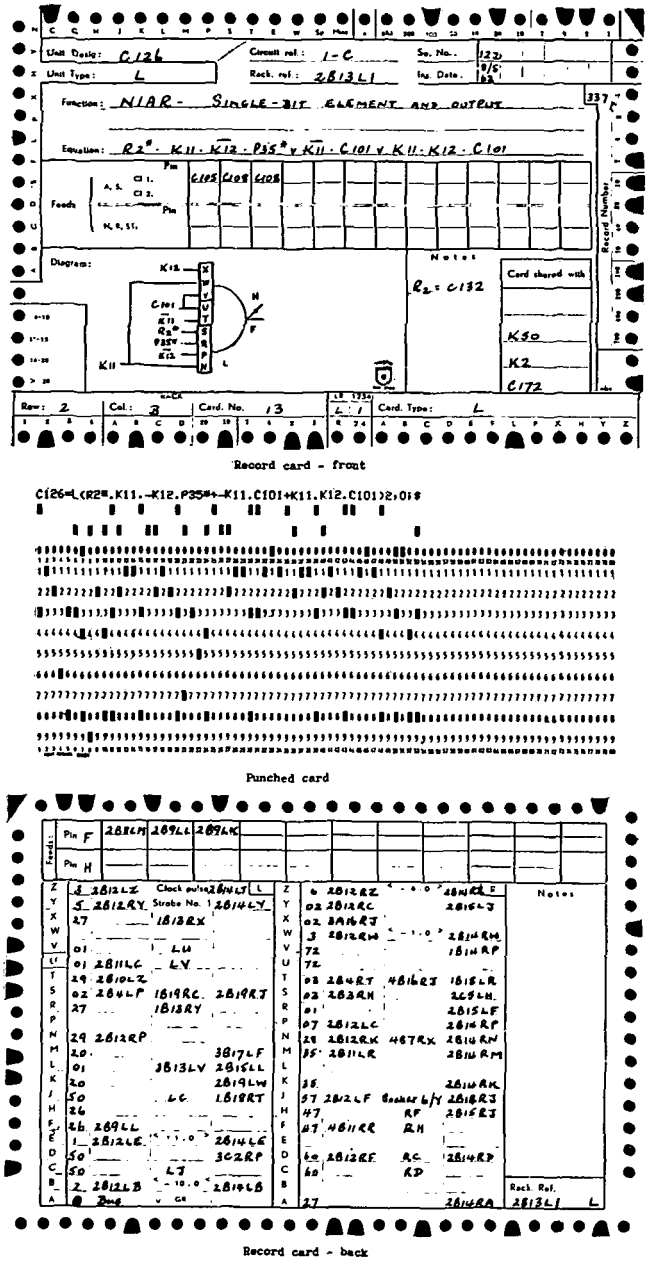

Fig. 4—A Sabrac record card and its punched-card equivalent.

The generator

Fig. 4 shows one of the Sabrac record cards on which are recorded all the relevant details appertaining to one equation and the unit which realizes it. The punched card illustrated records all the information required for a simulation, and in general includes:

(1) The name of the unit.
(2) The type—which defines its delay characteristic and any special properties.
(3) The equation.*

*At least one of the present authors (M.L.) objects strongly to the use of addition and multiplication symbols to represent the logical functions $\vee$ — OR and $\wedge$, & — AND, but from the set of Transac symbols available these were the most appropriate.

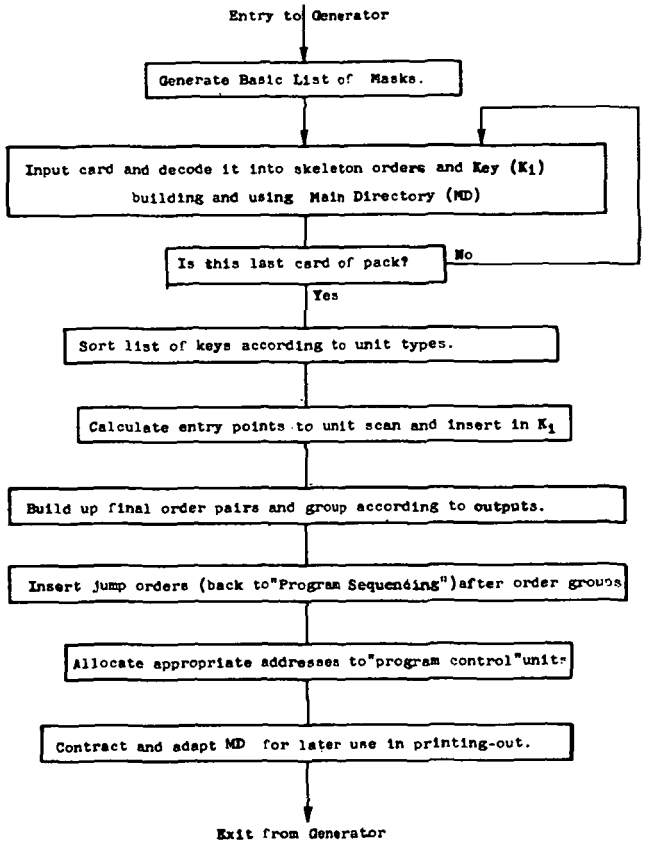

Fig. 5—Simplified block-diagram of equation-scan generator (Transac)

(4) The number of assertive and negative loads. A multiple feed to another equation is recorded as a single load since it requires only a single (compound) mask.

Equations are punched directly, without special column allocation, the variable-length fields being separated by control symbols. Thus in the example given the equation becomes:

$$C126 = L(R2^* \,.\, K11. - K12 \,.\, P35^* + - K11 \,.\, C101 + K11 \,.\, K12 \,.\, C101)2, 0; \$$$

The combination 2, 0; indicates that $C126$ feeds to two equations assertively and to no equations negatively. The $ symbol terminates the data for this unit. If necessary, the data for one unit may be spread over several cards.

The purpose of the generator is to convert the "statement" comprising the set of symbolic (Boolean), time-dependant equations, recorded as above, into a program which evaluates them at discrete bit-time intervals.

Fig. 5 shows a much simplified flow diagram of the generator. Entry is via a Mask Generator which generates and lists the eighty basic masks. The compound masks are generated as the need arises, during "Card Decode", and are stored as a continuation of the same list.

Cards are decoded as read, the equation being translated, in the first place, into a key, K_i, and a sequence of skeleton orders as described below. In addition, a "Main Directory" MD records the following details of each unit.

(1) The address of K_i.
(2) The number of assertive feeds.
(3) The number of negative feeds.

K_i includes

(1) The address of the associated entry in MD.
(2) The unit-type.
(3) An index number i determining the addresses EV_i and H_i.
(4) Assertion entry point to "Unit Scan".
(5) Negation entry point to "Unit Scan".

Fig. 5 shows that the last two items are calculated and inserted into K_i after completion of Input.

The Key-list is maintained during input in the order in which equation cards are read. The Main Directory is in modified alphanumeric order. An algebraic relation between the equation name and the MD address permits minimization of the storage space required by the list, by removing the need to reserve large blocks of locations for non-existent blocks of names.

The one-word skeleton order pair contains at this initial stage

(1) The name $\pm E_k$ of the feed unit.
(2) The appropriate Mask address $M_{f(j)}$ derived from EV_{ij}.
(3) The address EV_i.

When reading of the pack has been completed, the key-list is re-sorted by unit-types, the sequence within types remaining aribitrary. The blocks of types are ordered according to their delay characteristics, since the latter determine the order in which equations must be scanned.

Key-sort is followed by the insertion of the *TAC* function codes *TMD, DORMS*, into the skeleton order-pairs, yielding the final order pairs. After sorting into groups, according to the $\pm E_k$, these order-pairs form the heart of the scan program whose flow diagram is given in Fig. 6. Each order group, representing a set of assertive or negative feeds, is terminated by a jump order to "Program Sequencing", which controls the Scan sequence.

To some extent the program is controlled by the state of a small number of specific units, such as those indicating the end of a word-time or the beginning of a store-read. Thus the index number i of these units must be recorded by the generator for use during Interscan, since there is no recorded connection, after generation, between the name of the unit and its H_i.

The final task of the generator is to provide a list of data for use during print-out. Each item in the list comprises the alphanumeric name of the unit, its address H_i and a flag, which may be inserted or modified at the beginning of each run, defining whether the unit is to be printed or not printed.

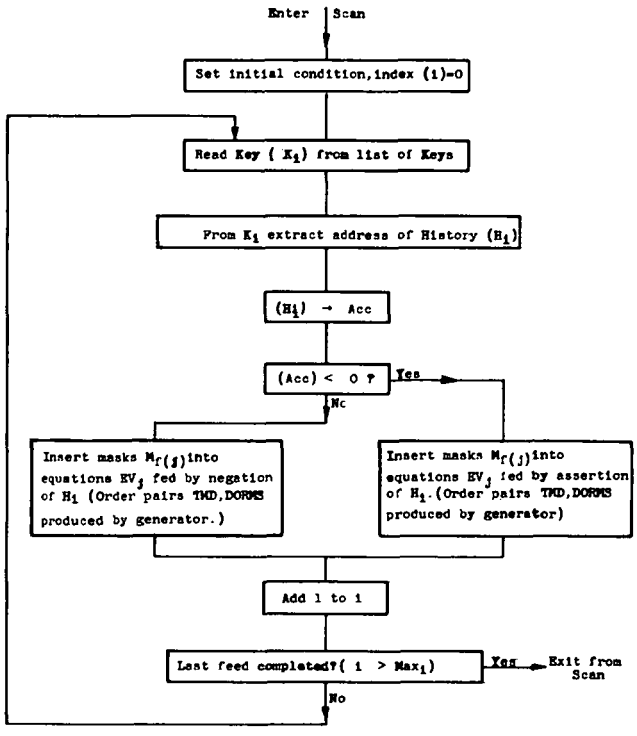

Fig. 6—Block diagram of "Scan by outputs"

The generator is largely self-checking and, in particular, any deviation from the permitted card-format is noted and printed out. Similarly the stated number of loads for any output is compared with the number actually occurring in the equations, and any deviation is once again noted. In general, an attempt has been made to foresee those errors which are likely to occur, and to detect them when they do occur.

The program (Interscan)

Apart from its general supervisory activity, which includes the sequencing of Scan, the program has the following functions:

(1) Timing.
(2) Initial Conditions.
(3) Simulation of all hardware not represented by symbolic equations.
(4) Checking of results.
(5) Print-out.

Timing

The timing of the simulation in "Sabrac time" is intrinsically determined by the Scan, each complete cycle representing one bit-time (approximately 9 μsec.) However, the existence of asynchronous devices for input and output makes it imperative to provide the program with a clock which is advanced by the 9 μsec unit during each scan. The use of this clock will become clear from the subsequent description.

Initial conditions. This part of the program makes provision for presetting the contents of the Sabrac store as simulated in Transac. It also permits the specification of any switches initially required to be in the ON condition. Finally, the contents of any "paper tape required to be read by Sabrac" are also stored in the Transac store.

Hardware simulation. The following components require special treatment:

(1) The core store.
(2) The drum.
(3) Console switches and push-buttons.
(4) The paper-tape readers.
(5) The paper-tape punches.

The core-store. The serial Sabrac store consists of 224, 36-bit locations divided into three systems: P_0 of 32 locations, and Systems I and II each of 96 locations. At any one time P_0 and one of the Systems function together as a normal program store, termed the *P-system*, while the remaining system, termed the *T-system*, is available for concurrent (autonomous) input, output or core–drum transfers. A "Change system and transfer control" order may at any time interchange the function of the two Systems. The simulation of this compound store presents no special difficulties since its whole behaviour is determined by the equations of the associated units. Sabrac core-store is simulated by a block of locations in the Transac core-store, with a simple one-to-one correspondence between the Sabrac and Transac addresses. The program has then to calculate the relevant bit-position in the Transac store whenever the simulated logic indicates the appearance of a bit on the Sabrac read or inhibit wires, and to read from or write into that position.

The drum. The drum consists of a 4096-word main backing store and a 1024-word library. It is represented in Transac by a block of core locations, with each Transac address fully specified by the Sabrac track number and track location, obtained from the logic. The track location (or drum position) which on the drum is determined by an "address track" is specified in the simulation by a counter associated with the main clock, counting once per word-time, modulo 32. The "word-rate clock track" is simulated by a decoder fed from the main clock, while the "bit-rate clock track" which in effect governs the speed of the machine, need not be simulated at all.

Switches and push-buttons. All switches and push-buttons which feed to the logic appear as normal inputs in the equations. Hence, for the purposes of the simulation they are given pseudo-equations with their state defined by their H_i. As previously mentioned, switches that are required to be "ON" at the commencement of the run are preset by "Initial Conditions". A full simulation of the machine requires the occasional operation of some switches or push-buttons at times which are random relative to the program. This effect is obtained by a facility which permits the change of state of a switch or the "pressing of a button" at "times" determined by the main clock. Such a time may be absolute or relative to some specified event.

Input and output. Sabrac is equipped with two input and two output channels. The former are fed by a pair of 300 characters-per-second paper-tape readers; the latter control two 110 characters-per-second punches. The problem in simulating this equipment lies in its asynchronous nature, and in the desirability to simplify possible changes in the specification or behaviour of the equipment. The timing of synchronizing signals is obtained from counters whose properties are specified by pseudo-equations of the same structure as the counter units of the machine. The outputs of the counters are compared, after each scan, with appropriately randomized target numbers, to determine when synchronizing signals are received. The pseudo-equations are punched and inserted with the main pack of equation cards, and properties of the peripheral and other asynchronous units may therefore be changed as required.

Checking of results. Experience has shown that simulation of any but the simplest functions is useless unless the results of each simulation are automatically checked. Thus each run requires the inclusion of appropriate subroutines which predetermine, according to the order specification, the results required from "Sabrac". Thus on conclusion of each order or group of orders as required, an automatic comparison can be made and any deviation noted.

Print-out. Full facilities are provided for printing out the contents, over word-time intervals, of the H_i for all or any desired list of units. Thus the behaviour of specified units may be monitored to enable the analysis of misbehaviour and the detection of faults.

The state of the Transac simulation

At the time of writing (August, 1962) the generator has been completed and is operating. The program is being written and it is hoped to commence the first simulations in the autumn. Early runs will be of part of the machine only. The accommodation of a full 600-equation simulation (with an average of six inputs per equation) in the existing 8000-location Transac store, would have been possible only if the program had included the use of magnetic tapes. In view of the fact that it is hoped to increase the store size to 16,000 locations before the end of 1962, the use of tapes did not appear justified. Thus a full simulation will have to await the arrival and commissioning of the new store.

Future plans and summary

Hardware commissioning of Sabrac has gone ahead much faster and more easily than had originally been anticipated. Thus it is not clear how big a part the simulation program will play in the commissioning of this machine. It may be available in time to assist in clearing up some of the more obscure faults which *may*

appear in the future. Even if this should not be so, the work will by no means have been wasted, since a program is available to check the logic of additions to or changes in Sabrac and of any new machines or devices as these are designed. The advantages of such parallel checking are obvious and need not be detailed.

An extension of the present program would also permit its use for the simulation of asynchronous devices, but no work has yet been undertaken in this direction.

The present project has also suggested a further research program which would investigate the possibility of mechanizing all or part of the actual design process. It would appear possible to derive automatically a set of equations which mechanize the execution of any specified (computable) function. Several possible approaches to this problem exist, and it is hoped to undertake an investigation which will determine methods yielding useful results in a reasonable amount of machine time.

Acknowledgements

The work described in this paper was carried out in the Scientific Department, Israel Ministry of Defence.

References

LEHMAN, M. (1959). "Specification of a Cost-Limited Digital Computer", *Information Processing*. Proc. International Conference on Information Processing. Paris 1959. UNESCO-Paris, Oldenbourg-Munich, Butterworth-London, 1960, p. 365.

LEHMAN, M. (1961). "The SABRAC Digital Computer", Technical Report No. 134, Sci. Dept. Israel Min. of Defence. February, 1961.

LEHMAN, M. (1961). "Serial Matrix Storage Systems", *IRE Trans. Electronic Computers*, EC-10, No. 2 June 1962, p. 247.

STOCKWELL, G. (1962). "Computer Logic Testing by Simulation", *IRE Trans. Military Electronics*, July 1962, p. 275.

Book review: Management

Management and the Computer of the Future, edited by M. Greenberger, 1962; 340 pages. (Cambridge, Mass.: *The M.I.T. Press*; London: *The Book Centre, Ltd.*, 57s.)

This book consists of the transcription of a fascinating series of eight evening lectures given at the Massachusetts Institute of Technology in 1961, its centenary year. Even more interesting are the discussions, recorded here in full. Many well-known names in the American computer scene appear in them, with that of J. McCarthy well to the fore, attached to frequent penetrating remarks.

Although the focal point was management, the lectures represented a wide field, and much that was of interest to managers (and, indeed, to all of us) came out of frank speaking on a variety of technical subjects. McCarthy himself, for example, speaking on "Time-Sharing Computer Systems," gives a picture of the way he sees the computer business developing in the future. Later he gets involved in an argument with J. R. Pierce who, in a talk on "What Computers Should Be Doing," deprecates attempts to make computers do things that people can obviously do better; McCarthy feels that this is not looking far enough ahead.

J. G. Kemeny, a mathematician and editor, describes his idea of "A Library for 2000 A.D.": many millions of volumes stored on tape, accessible from anywhere in the country by closed circuit television. McCarthy feels that computer techniques at least as powerful as those projected by Kemeny will be available in 1965. On the indexing problem, however, little light is shed: Kemeny's picture is naïve, but no confident alternatives are put forward.

To those of us who face it, "The Computer in the University" as proposed by A. J. Perlis is the biggest attraction in the book. Perlis is an extreme visionary, and even if we do not share his views we can gain a lot from his stimulating account of future classroom life with the computer. He shrinks from no aspect of the subject. Some of his comments seem revolutionary, at least to us in Britain, and yet the lively discussion which follows brings out no real disagreement among those present. It is, in fact, a clear illustration of how the climate of intellectual thought in American universities has accepted the computer; a result, no doubt, of the policy of the National Science Foundation which Perlis praises for having provided "funds so that universities not only could have computers, but could have good ones."

Indeed, it is a depressing reflection that one simply cannot imagine discussions of such breadth and depth as those reported in this book, occurring here. Even those talks which present no really new material, "Managerial Decision Making," by J. W. Forrester, "Simulation of Human Thinking," by H. A. Simon and A. Newell, and "A New Concept in Programming," by G. W. Brown, represent viewpoints which, though well established in the U.S.A., would be somewhat revolutionary in this country.

There is some comfort in the fact that the introductory lecture, which would in fact have been more appropriate as an epilogue, was given by an Englishman, Sir C. P. Snow. Snow has no detailed knowledge of computers, but his perspicacity points out their social consequences more clearly than others could. His audience, led by E. E. Morison and Norbert Wiener, rises to the occasion. Why do people like Sir Charles not address British audiences on these matters? Are there no invitations? Or are we so lacking in our grasp of the new technology that their words would be wasted?

S. GILL.