

SANDIA CORPORATION
SANDIA BASE, ALBUQUERQUE, N. M.

Via Registered Air Mail

November 6, 1959

Mr. Harlan E. Anderson
EJCC Publication Committee
Digital Equipment Corporation
Mayard, Massachusetts

Dear Mr. Anderson:

Enclosed are the required four copies of "Pattern Recognition and Reading by Machine" by W. W. Bledsoe and I. Browning of Sandia Corporation. This paper, as you know, will be presented at the 1959 Eastern Joint Computer Conference by Mr. Browning. I am also enclosing a separate set of illustrations for reproduction and a short biography of the speaker.

Mr. Browning will have with him two complete sets of $3\frac{1}{4}$ x 4 colored slides. He will not use any other visual aids.

Sincerely yours,



D. F. Rauber
Section 3423-1
Sandia Corporation

DFR:3423-1:hg

Enc.

PATTERN RECOGNITION AND READING BY MACHINE

W. W. Bledsoe and I. Browning
Sandia Corporation, Albuquerque, New Mexico

ABSTRACT

An essential part of data processing by machine is pattern discrimination, characterization, and mensuration. A system to accomplish these ends has been devised and is being used on an IBM 704. To evaluate the discriminating capability of the system, typewritten numerals, hand-block print, and handwritten script characters have been used as patterns of respectively increasing complexity and individual variability.

All of these patterns were processed by using a family of general-purpose programs based upon a central principle which embodies Gestalt recognition. Original patterns are imaged on a 10 x 15 photocell mosaic, reduced as a 150-bit binary coded image (or number), and "learned" in a memory matrix. Successive experiences are added to this memory matrix. New patterns are recognized by measuring their similarities to the learned experiences. Recognition is accomplished on a character by character basis and also by contextual relationships.

Typical results are: 1) For typewritten numerals, 100 percent recognition under all conditions tried; 2) for hand-block print, 92 percent under certain conditions; for handwritten script characters, 54 percent under certain conditions. Elaboration of the logic to include contextual reading increased the recognition of hand-block print to 100 percent and of handwritten script characters to 98 percent.

Figure Captions

- Fig. 1a -- The photomosaic and two of the randomly chosen photocell pairs. The four digital groups to the right are the four possible states of each photocell pair.
- Fig. 1b -- The system learning the letter I in a central position. Only two of the 75 pairs are shown.
- Fig. 1c -- The system learning the letter I in another position. Note that the memory experience shown in the previous figure remains.
- Fig. 1d -- The system learning the letter I in a third position. The check marks to the right show all possible combinations of these two photocell pairs for the letter I.
- Fig. 2a -- Hand-block print as it appears on IBM cards. (Top -- A, C, E; Bottom -- N, M, H.)
- Fig. 2b -- Handwritten script characters as they appear on IBM cards. (Top -- w, l, o; Bottom -- s, r, e.)
- Fig. 3 -- The memory matrix with the characters B, G, and 5 learned. Note that two G's have been learned.
- Fig. 4 -- Comparative scores of hand-block letters.
- Fig. 5 -- Comparative scores of handwritten letters.
- Fig. 6 -- Comparisons of the percent recognized for hand-block print read with different n-tuplings: $n = 1$ (hatched bars) and $n = 2$ (solid bars). Note that when all five alphabets are learned together, the percent for $n = 2$ improves. In other words, for $n = 2$, the ability to read improves with additional learning in the memory matrix.
- Fig. 7 -- Comparison of percentage recognition of hand-block print with different n-tuples. Five alphabets (labelled A, B, C, D, and E) are considered singly, and then together.

Fig. 8a -- Scores made on handwritten script letters, showing that for larger values of n , larger amounts of learning are useful.

Fig. 8b -- Material of Fig. 8a presented in different form.

Fig. 9 -- Arbitrary shapes which were taught to the system as a basic distribution pattern for the subsequent reading of alphanumeric handwritten characters. Each shape was learned in the position shown and also in several positions resulting from lateral displacement.

Fig. 10 -- Percentages of recognition for five different choices of random n -tupling.

Fig. 11 -- Handwritten letters read by context. Letters and words incorrectly identified are underscored.

Table Captions

Table I -- Typical experiments indicating experimental parameters and percentage of read-out for hand-block print.

Table II -- Typical experiments indicating experimental parameters and percentage of read-out for handwritten script characters.

Table III -- Scoring by context.

PATTERN RECOGNITION AND READING BY MACHINE

W. W. Bledsoe and I. Browning
Sandia Corporation, Albuquerque, New Mexico

INTRODUCTION

Many efforts have been made to discriminate, categorize, and quantitate patterns, and to reduce them into a usable machine language. The results have ordinarily been methods or devices with a high degree of specificity. For example, some devices require a special type font; others can read only one type font; still others require magnetic ink.

We have an interest in decision-making circuits with the following qualities: (1) measurable high reliability in decision making, (2) either a high or a low reliability input, and (3) possibly low reliability components. The high specificity of the devices and methods mentioned above was felt to be a drawback for our purposes. All of these approaches prove upon inspection to center upon analysis of the specific characteristics of patterns into parts, followed by a synthesis of the whole from the parts. In these studies, pattern recognition of the whole, that is, Gestalt recognition, was chosen as a more

fruitful avenue of approach and as a satisfactory problem for the initial phases of the over-all study.

In addition, we chose to concentrate upon the recognition of alphanumeric patterns, rather than upon other pattern types, for the following reasons:

- (1) Convenience. Results can be handled easily since it is possible to use conventional print-out equipment. Furthermore, we could exploit our own familiarity with letters and words.
- (2) Background. Research on alphanumeric pattern recognition has been vigorously pursued, and we were therefore able to make use of the relatively large literature on the subject.
- (3) Usefulness. Success in our efforts would make available a technique which society needs and can use immediately, even though such a result would be only a by-product of our over-all study.

Because typewritten numbers were recognized without error in the cases considered, the investigation quickly shifted to hand-blocked print and finally handwritten script characters as

displaying greater complexity and increasing individual variability. In this way the decision making powers of the system were more fully challenged.

Since a numerical output is the inherent mode of expression of a digital computer, our work was aimed at developing a numerical score for each pattern examined. The basic method employed to obtain these scores and to use them to identify each pattern uniquely will be described in the following section. Then various expansions and variations of the method will be covered. Finally, a method of extending identification by contextual relationships will be described briefly.

It may be mentioned at this point that this system is highly general -- that is:

- (1) It handles all kinds of patterns with equal facility.
- (2) Because it does not depend upon absolute pattern-matching, it can identify a pattern which is not exactly like, but only similar to, a pattern it has previously learned.
- (3) It does not depend significantly upon the location of a pattern on the photomosaic for identification.

- (4) It is only partially dependent upon the orientation and magnitude of a pattern for identification.

It would also be well to mention the two major disadvantages of the system:

- (1) When the learned patterns are quite variable, the memory can be saturated, especially in certain cases.
- (2) A very coarse mosaic, especially if it has inconstant photocell performance, produces images of small letters which do not contain enough information for recognition. See, for example, the sixth character, an e, in Fig. 2b. The large letters, however, do not present this problem.

However, both of these disadvantages can be at least partially overcome; the first, by various techniques to be described later; the second, by using a mosaic with more photocells.

BASIC METHOD

Of prime importance in this method is the way in which pattern discrimination is provided. The best way to describe the process is by example.

We start with a 10 x 15 photocell mosaic (this size being chosen because of immediate availability), the elements of which are related to one another as 75 randomly chosen, exclusive pairs. Fig. 1a shows the mosaic and two such randomly chosen pairs ($1_1 1_2$ and $2_1 2_2$). Images, letters for example, projected on the mosaic will produce characteristic patterns, examples of which are shown in Figs. 2a and 2b as they appear on IBM cards. For computer convenience, the light values of an image on the mosaic are rendered in a binary system which treats dark as 1 and light as 0. When an image is on the mosaic, each pair of photocells (the members of which are ordered for this purpose) will represent the light values of the image as a two-bit number. Each pair of photocells has therefore four possible states -- 00, 01, 10, and 11.

In the memory matrix of the computer, a 36-bit computer word is assigned to each state of each pair, giving four words for each photocell pair or 300 computer words for the 75 pairs.

Furthermore, each bit position in the 36-bit computer words is assigned a pattern nomenclature. The sequence used in our experiment was:

<u>Position:</u>	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...	35	36
<u>Nomenclature:</u>	.	1	2	3	4	5	6	7	8	9	a	b	c	d	e	...	y	z

This nomenclature sequence will hereafter be referred to as an "alphabet."

In order to demonstrate how patterns are "learned," we will use as an example the letter I. First, a letter I is projected on the photocell mosaic (Fig. 1b). Its image on the mosaic produces one of the pair states (00, 10, 01, 11) for each of the pairs, depending upon the amount of light falling on the pair. Since all 75 pairs are involved, the resulting 75 states address 75 words in the memory matrix. For each word addressed, a binary 1 is entered in the nineteenth position, the position corresponding to the letter being learned, I. Obviously, if the letter A were being learned, a binary 1 would be entered in the eleventh or A position, and so forth. The process described constitutes the learning of a single letter I, but whole series of letter I's, differing in shape or position or both, can be learned. For example, Figs. 1b, 1c, and 1d show the same I being learned in

different positions, while Fig. 3 shows a case in which two G's have been learned.

Since not all the letter I's will be in the same position as the first, some different computer words will be addressed. That is to say, there is a degree of individual character variability. However, no letter I or combination of I's will normally address the same 75 computer words as, say, a letter A would. This is a key point: the very shape of a character, such as the letter I, forbids certain states for certain pairs. The existence of these forbidden states lies at the heart of our method, for without them the logic would saturate. In sum, different patterns have different forbidden states and consequently score differently.

Now, suppose that we have taught the logic several alphabets, proceeding for each character as for the letter I above. We can then identify a specific unlearned character, an A for example. A letter A is "read" by imaging it on the photomosaic. Its image will address the 75 computer words in the memory matrix that correspond to the active states of the 75 pairs. Identification of the specific pattern in question is made by comparing the unknown image with the previously learned characters. In practice this is done in the following way:

- (1) The binary 1's in position one (the position corresponding to .) are added up for all of the 75 computer words addressed by the unknown pattern. The score obtained shows the similarity of the unknown pattern to the . pattern.
- (2) The same process is repeated for the other 35 positions, with the result that 36 numerical scores are obtained.
- (3) These scores are compared by the computer, and the highest score wins. That is, the unknown pattern is identified with the character occupying the position scoring highest. If there is a tie for highest score, the computer arbitrarily selects one of the highest scores as the winner. Note that the highest score possible is 75.

Fig. 4 shows an example of scoring for hand-block A and T.

Fig. 5 shows scoring for much more highly variable patterns, namely, handwritten a and t.

It will be noted that if an image corresponding exactly to the unknown image had been learned before by the matrix, a score of 75 would be made at that position. Again, if by learning several similar patterns (A's, for example), all of the pair

states now being addressed had been learned, a second 75 would be made. However, in most cases, an unlearned character will not make a perfect score. The degree of similarity is measured by comparing the magnitude of the various scores with a perfect score of 75. Discrimination is defined as the difference between the score of the correct character and the next highest score. It can be seen that what actually happens in this process is that the images, both those learned and those being read, are transformed into a new space (the memory matrix) and are there compared for identification.

LOGIC EXPANSION AND MANIPULATION

Our studies and experiments moved outwards from the basic method to include a variety of modifications and variations. An attempt is made below to evaluate each variation in terms of its final effect. It should be noted that the combination of two or more of the methods to be described results in substantial increases in correct readings.

1. Different Photocell Groupings

In the examples cited, the photocells were grouped as exclusive pairs. However, it is obviously possible to use

n-tuples in which n has any value from 1 to 150. Let us begin by comparing the system employing photocell pairing ($n = 2$) with a system in which $n = 1$. In the latter case, each individual photocell addresses only two computer words, since its possible states are 0 and 1. The difference in the behavior of the systems is striking. If we re-examine Figs. 1b, 1c, and 1d, we note that in learning several images of the letter I, with $n = 1$, every single photocell would exhibit the values 1 and 0: this is so because the position of the letter I changes. In other words, unless the image on the mosaic is held within narrow limits, the memory loses most of its discrimination value with $n = 1$.

We can say then, that position is very critical in the case of $n = 1$, and that it has less importance for $n = 2$. A direct consequence of this difference is found when the matrix is taught more than one position or more than one example of a pattern. The scores will improve if $n = 2$; for $n = 1$, they will not improve and will probably deteriorate. Figs. 6 and 7 illustrate this characteristic with respect to five alphabets learned separately and then in combination. Marked improvement in the reading of this message, which was written in hand-block print, was achieved when $n = 2$ rather than $n = 1$. For the five alphabets learned separately, the average percent of recognition

with $n = 1$ was 56.12 percent; for $n = 2$, 54.01 percent. But for the same five alphabets learned together, the percentages are 46.42 for $n = 1$, and 67.63 for $n = 2$. (See also Figs. 8a and 8b.)

Remembering that n can equal any number from 1 to 150, we can ask what effect is produced when higher n -tupling is used. The problem of pattern recognition with a multichannelled system, such as the one simulated for discussion here, has traditionally been approached from one of the two extremes, that is, $n = 1$ or $n = 150$. Consider the formula

$$S^n \times \frac{N}{n} \times C = L,$$

where

S = the number of operational states of the photocell. In the case being considered $S = 2$, for the possible photocell states are 0 and 1.

n = the parameter for n -tupling.

N = the number of photocells.

C = the number of categories of patterns learned and read (36 in the previous examples).

L = the number of storage sites in the memory matrix.

The factors held arbitrarily constant in our experiment were $n = 2$, $N = 150$, and $C = 36$. The traditional cases, as mentioned before, have involved $n = 1$ and $n = N = 150$. But the former has been shown to deteriorate or at least not to improve appreciably with learning. The latter, on the other hand, requires a prohibitively large memory matrix (36×2^{150} , using the same values as above), although its reading ability would be perfect if enough learning experience could be provided.

Let us summarize concerning these two extreme conditions. If $n = 1$, there are no forbidden combinations and therefore the memory will saturate with the learning of successive characters which vary in size, area, shape, or position. Such a logic has, consequently, an extremely limited use. If $n = 150$, saturation is impossible. But, even apart from the ^{IM}possibility of having 2^{150} computer addresses available, images being read successfully would be restricted to exactly those that had been learned before. This logic, then, has even more severe limitations.

Our method avoids these several disadvantages by concerning itself with intermediate values of n , values which provide the learning advantages of a large exponential matrix but which retain a memory matrix more comparable in size to the

photomosaic matrix. For example, with $n = 2$, the formula for the logic used gives:

$$2^2 \times \frac{150}{2} \times 36 = 10,800$$

The number of bits in the memory matrix for the simplest case of a system not position sensitive, under these conditions, is therefore 10,800.

Let us introduce another quantity, M , which will be the number of photocell n -tuples utilized in a given experiment. While M will normally be given by N/n , larger M values can be obtained by non-exclusive n -tupling of the photocells. We will have more to say about the non-exclusive cases later.

In any event, it is obvious from the formula that a larger memory matrix can be utilized if any of the variables are increased. During the course of our experiments, we used the following values:

$$n = 1, 2, 3, 5, 8$$

$$M = 30, 50, 75, 150, 128, 256, 512, 1024$$

$$C = 10 \text{ and } 36$$

The experimental data suggest that a greater amount of logic produces better discrimination. The primary effect of varying n is that as n increases, the percent of recognition increases with increased learning (Figs. 7, 8a, and 8b). However, a balance must be preserved among the various parameters in order to utilize to best advantage a given amount of logic and to minimize computing time.

2. Non-exclusive n-tupling

Some experiments were made in which non-exclusive n -tupling was used for the photocells. The number of n -tuples (M) used could in these cases have any value. Tables I and II shows that non-exclusive pairing resulted in some improvement in the percent of characters recognized. But this improvement was at the expense of more storage space and longer computing time. We feel that a larger gain in percent recognized can be realized, for the same amount of storage and same length of computing time, by increasing the number of photocells (N) and continuing to use exclusive n -tuples. In other words, we see no real advantages in non-exclusive grouping.

3. Positioning

A procedure for pre-positioning characters for learning and reading by rotating an origin was attempted and found to be profitable in special cases. This rotating-origin technique is useful for digits and for print, but will not work with handwritten script. That is to say, if a character or pattern is separate and distinct, it can have an origin rotated with respect to some reference. Handwriting (as contrasted with the separate handwritten characters which we used) has continuity, and there is no obvious origin from which to start. Some method for separating handwriting into its components would be required before the origin of such components could be rotated profitably.

For each character an origin is arbitrarily defined. The character is then successively repositioned about this origin in the following sequence of x, y values: 0,0; 1,0; 1,1; 0,1; -1,0; -1,-1; 0,-1; 1,-1; 2,0; etc. Scores are obtained for each value, and the maximum score made by a character in any of the positions is chosen as the identifying score. This program involved a considerable amount of computer time, and is of interest mainly in connection with the possibility of simulating conditions for "servoing" the "eyeball." Such a feedback system appears feasible, since effective score criteria were found.

In a variation of the positioning program, the characters were all relocated by the computer to the upper left hand corner of the rectangle. This positioning, combined with the rotating-origin program just described, gives the maximum probability of reclaiming position-dependent data. This combination provides the largest increases in effectiveness for the $n = 1$ cases, those cases which we have seen are most sensitive to position. Typical increases in percent recognized for hand-block print with these techniques are:

<u>Original</u>	<u>Positioning</u>	<u>Rotating origin</u>
80	84	89
72	88	90

4. Distribution Processing

A method of processing the data obtained from the pattern scores was tried which was based on the entire scoring pattern rather than upon the maximum score only. The principle involved becomes clear at once if Fig. 5 is re-examined. Note that the sets of scores with respect to the previously learned letters are quite different for a and t. These different values are apparently consistent in their differences. For example, t scores high for b, while a scores low for b, and so forth.

The procedure is first to teach the memory matrix several alphabets as a primary experience. Scores made by one or more additional alphabets, constituting a secondary experience, are then averaged to give a score distribution typical of each character. An unknown pattern is compared with the memory matrix in the usual way to obtain its distribution of scores. This distribution is then compared with the typical distributions and the one most similar to it is chosen. For convenience, all of the scores were normalized, so that the sum of the scores in each distribution was one. Comparisons between two distributions were made in these experiments by summing the absolute values of the differences of the corresponding scores. It might well prove useful to employ a correlation technique in which a sum is taken of the products of corresponding scores, but this has not yet been tried.

As an example of results, in one case in which handwritten script characters were being read ($n = 5$, 3 alphabets learned), we found:

Undistributed	32.3% recognized
Distributed	45 % recognized

A final approach in this effort was to introduce ten arbitrary shapes for the primary experience (Fig. 9). After these

were taught to the memory matrix, three alphabets were compared with the matrix to obtain a ten-component distribution analogous to the 36-component bar graph of Figs. 4 and 5. The three ten-component distributions were averaged. New alphabets could then be read by the distribution-comparison program. For handwritten script the results of this program were:

Undistributed	32.3% recognized
Ten-Component Distribution	51 % recognized

This program was novel in that it involved two steps of disorder; that is, two arbitrary operations -- random pairing and comparison with arbitrary configurations -- were performed on patterns before attempting to read order out of them. It is also important to note that by using only 10 shapes instead of 36, a considerable saving in computer time is realized.

5. Probability

The method of reading characters described previously utilizes a memory matrix which is taught by a given set of experience patterns. Another method was tried in which the contents of several such memory matrices were averaged to obtain a "probability" matrix which was then used as the memory matrix in the reading phase. The memory matrices used in the averaging can be taught by different sets of experience patterns. An

interesting (but not very successful) special case is one in which each of the matrices being averaged is taught only one alphabet of experience patterns.

In the few cases tried with this method, the percent of handwritten characters recognized was increased as follows:

Original	28% recognized
Probability Matrix Used	52% recognized

Certain variations of this "probability" method will undoubtedly yield some increase in percent recognition.

5. Discrimination Criteria

The scores obtained for each pattern read by any of the described methods lend themselves readily to the establishment of discrimination criteria. That is, if the standard of minimum margin is not met for a given image, a secondary program can be evoked which utilizes one of the higher (and probably slower) logic treatments for higher resolution and/or discrimination. Such a program would give the computer a second, and "more careful look" at a pattern which was not clearly recognized on the first trial.

6. Randomness

Since the elements of the photomosaic are related to each other by randomly chosen n-tuples, it was decided to test the

sensitivity of reading ability to changes in the particular organization used. The random (actually pseudo-random) n -tuples were generated by the following program. First a random permutation, $k(1), k(2), \dots, k(150)$ of the numbers $1, 2, 3, \dots, 150$ was generated. Then the elements of the mosaic E_1, E_2, \dots, E_{150} , were related in this manner:

$$(E_{k(1)}, E_{k(2)}, \dots, E_{k(n)}), (E_{k(n+1)}, \dots, E_{k(2n)}), \dots, \\ (E_{k(150-n)}, \dots, E_{k(150)}).$$

The test was made by using five different randomly chosen permutations to read the same set of patterns. The results are shown in Fig. 10. Although admittedly the sample was rather limited, indications are that the percent recognized is fairly insensitive to the variation, especially when the percent recognized is high.

7. Context

Another method to extend the basic technique deserves special attention, for it produced the highest percentage of correct readings. It is identification of letters by word context, and it operates as follows:

1. Establish the length of an unknown word by counting the number of characters between spaces.

2. Establish, by the techniques previously described, all the scores for all of the letters constituting the unknown word.
3. Using a vocabulary of words of the length in question, add in their proper order the letter scores of each word in the vocabulary to obtain a total score for each word.
4. The highest score wins.

Table III illustrates the whole process. Words can be read by context (see Fig. 11), or, since each word has a score, it would be possible to establish a similar program to deal with phrase context.

The word the in the message in Table III won against 100 other three-letter words even though it was badly misread letter by letter. The results shown in the table were obtained using a vocabulary of 677 most commonly used short words. Obviously a larger vocabulary would result in decreased recognition.

Tables I and II summarize the results obtained for reading hand-block print and handwritten script by the basic method and by the various modifications described.

DISCUSSION

The problem posed by this investigation was: Can a general program be utilized to attenuate the information contained in a higher-order matrix pattern, while at the same time retaining enough of the essence of the information to categorize the pattern. Our results clearly indicate that this is possible. And although such a program will be useful at once for purposes of character recognition as is required in general reading machines, it has a much broader import.

A general program of this sort -- as opposed to such specific logical programs as pattern matching or analytical character differentiation -- will be useful as a basic tool in our investigation of decision-making circuits. This method could be expanded into such areas as phrase context, the automatic reading of books as a service to language translation programs, etc. It should perhaps be re-emphasized that the program identified typewritten numbers without error in the cases considered. Handwritten script was purposely introduced to challenge the program by offering it patterns of high variability.

ACKNOWLEDGEMENTS

The authors wish to express their sincere gratitude to Miss Tomasa Santos for her extensive work in programming these concepts and running them on the computer, to Dr. L. S. Lockingen for his large part in interpreting the data, and to Mr. S. D. Stearns for his generous assistance and the use of his equipment.

1/05

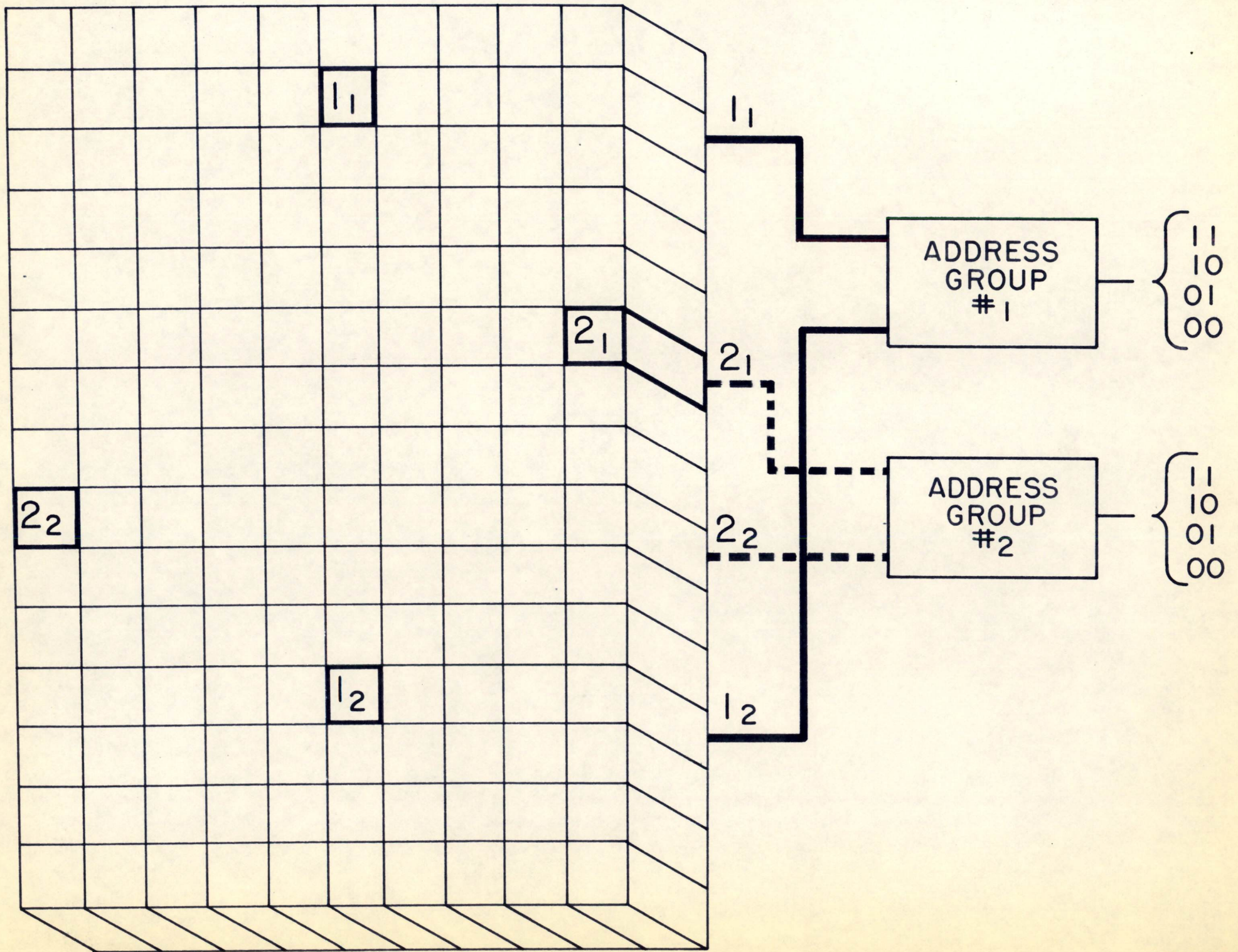


Fig. 1a

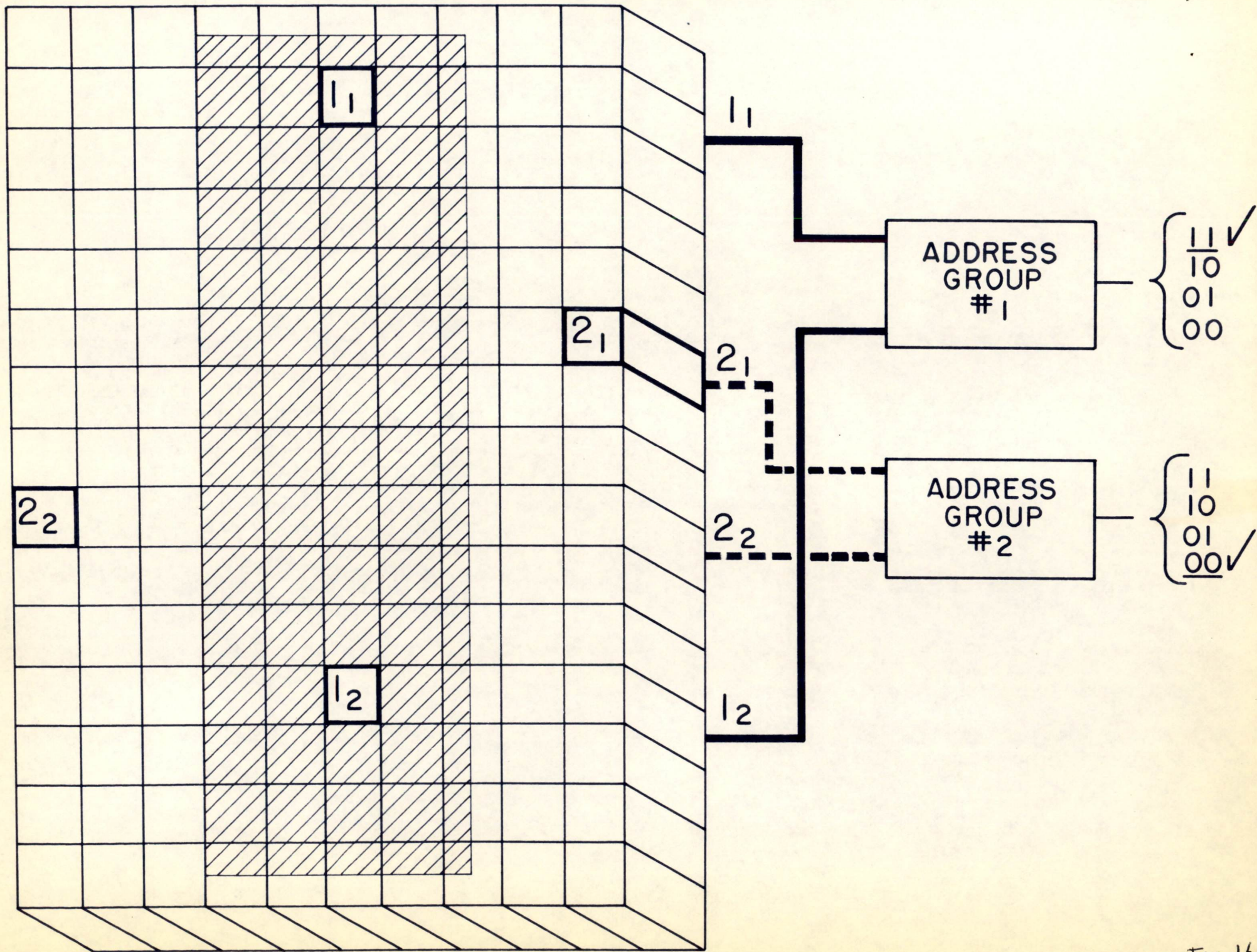


Fig. 16

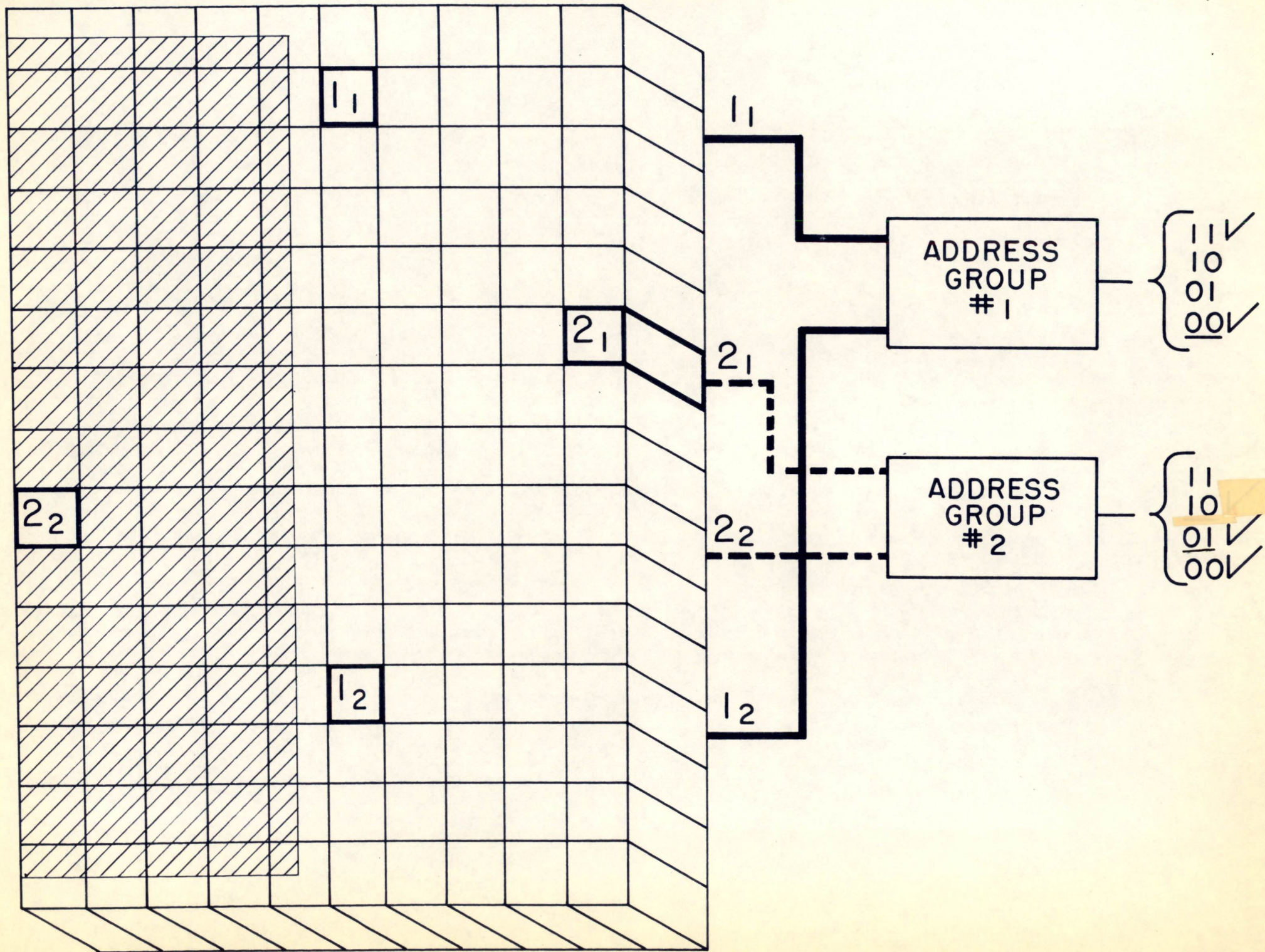


Fig. 1c →

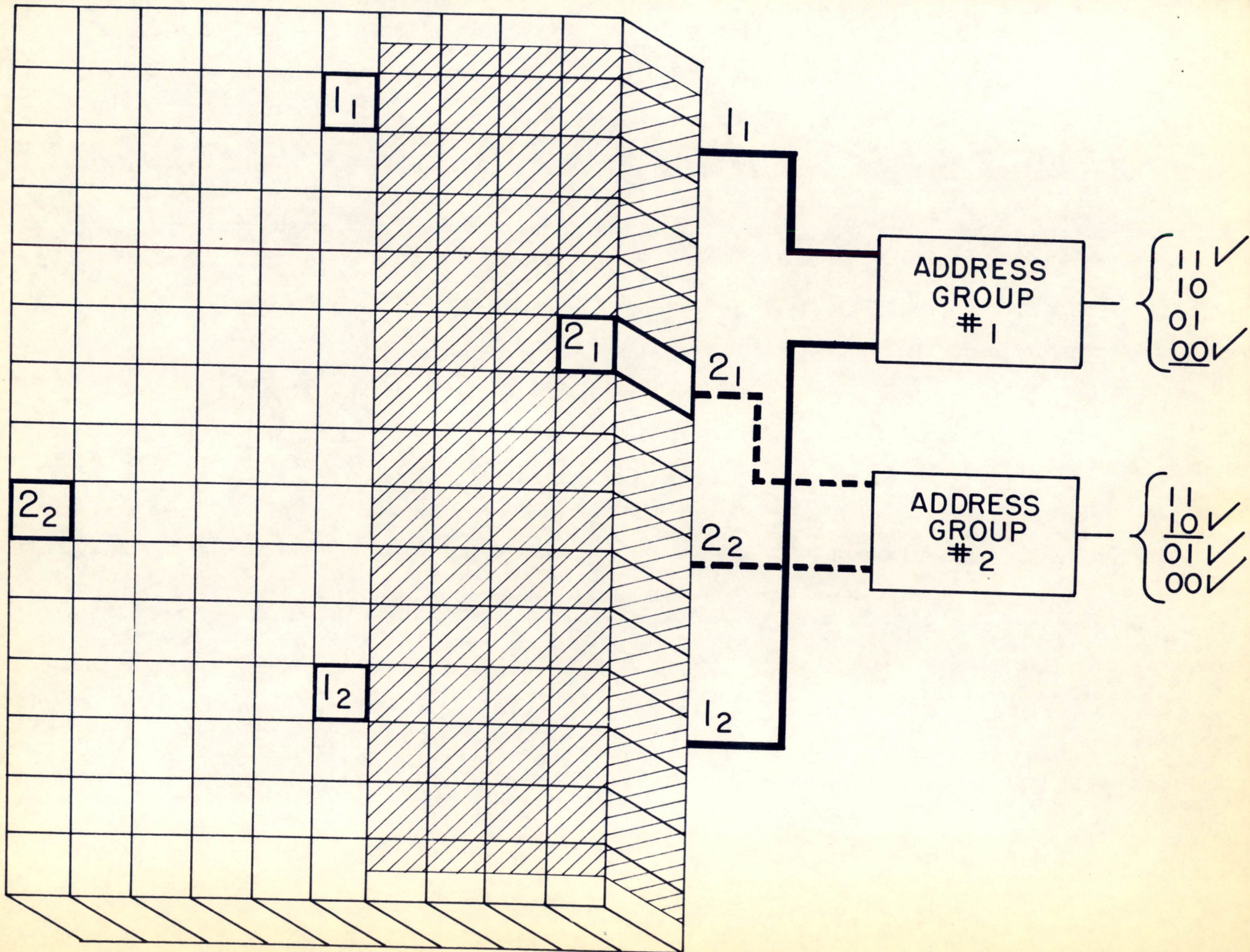
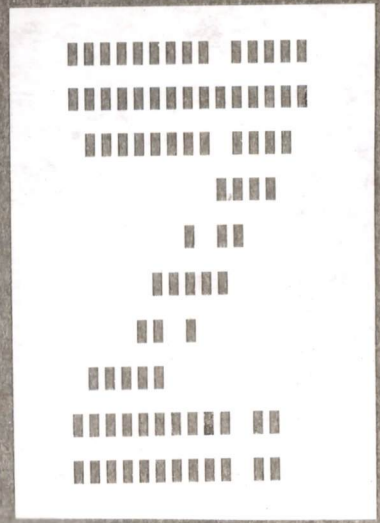
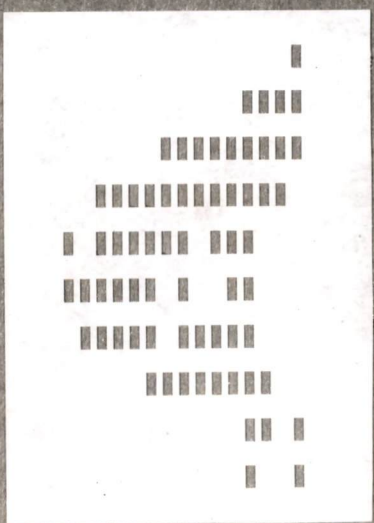
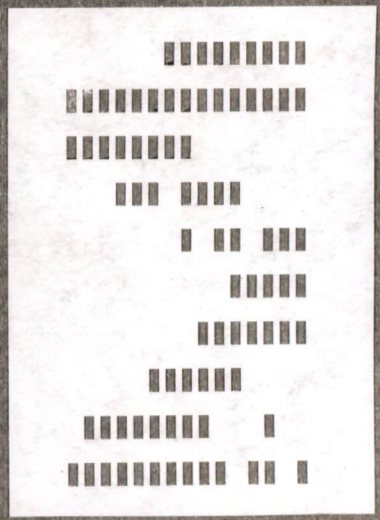
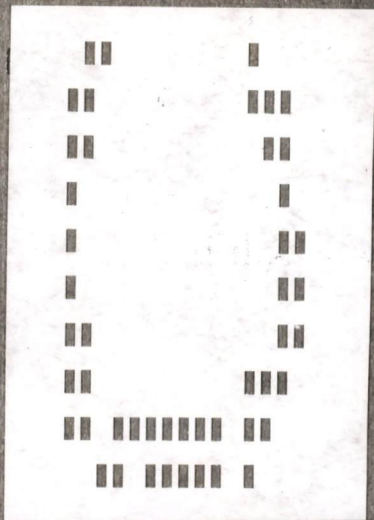
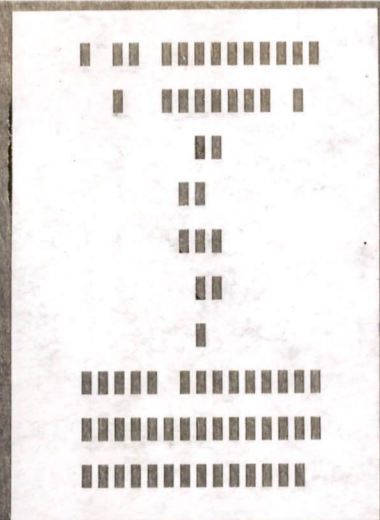
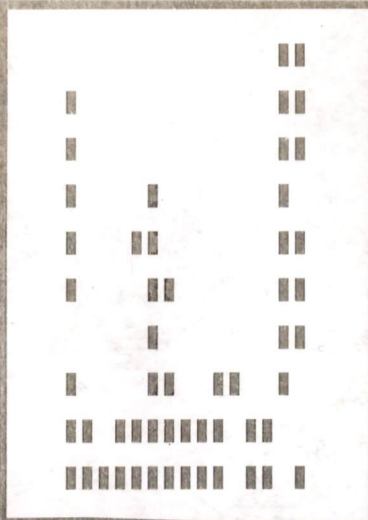


Fig 1d

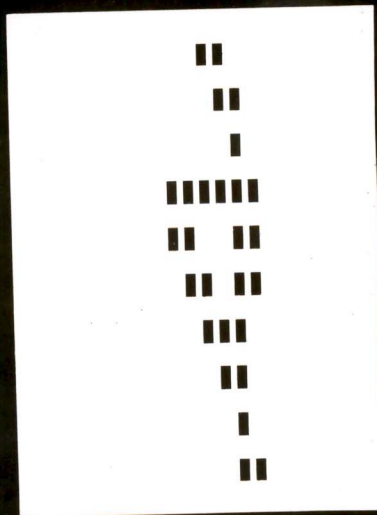
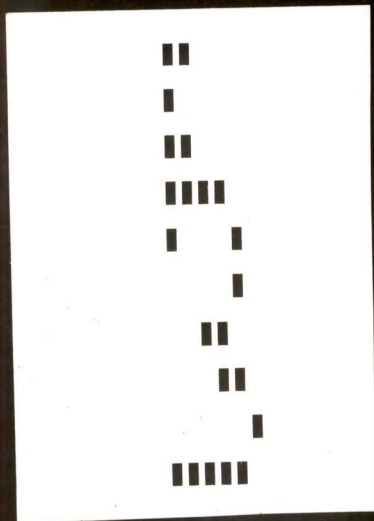
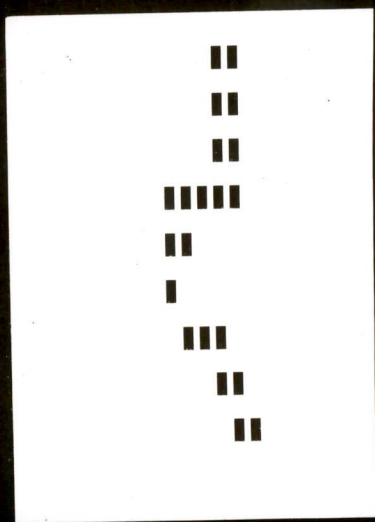
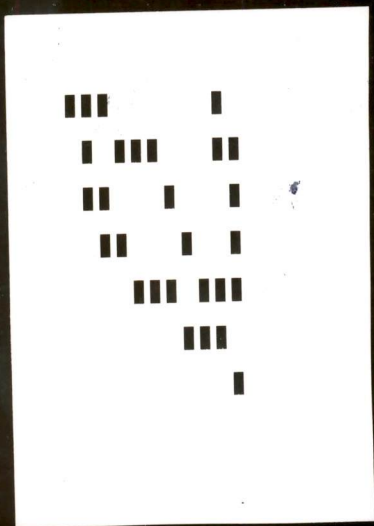
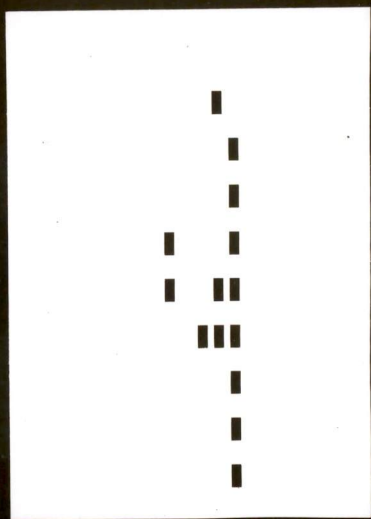
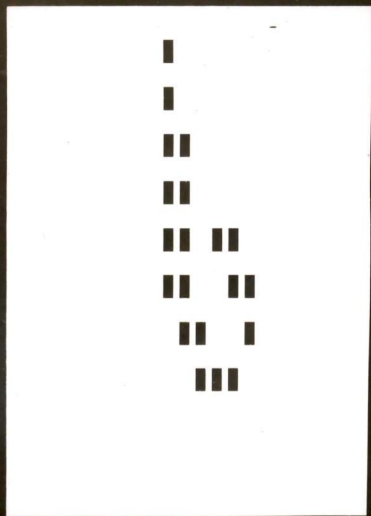


SANDIA CORPORATION
PHOTOGRAPHIC LABS
Negative No. D9-12712

Copy No. _____

UNCLASSIFIED

Fig. 2a



SANDIA CORPORATION
PHOTOGRAPHIC LABS
Negative No. D9-12711

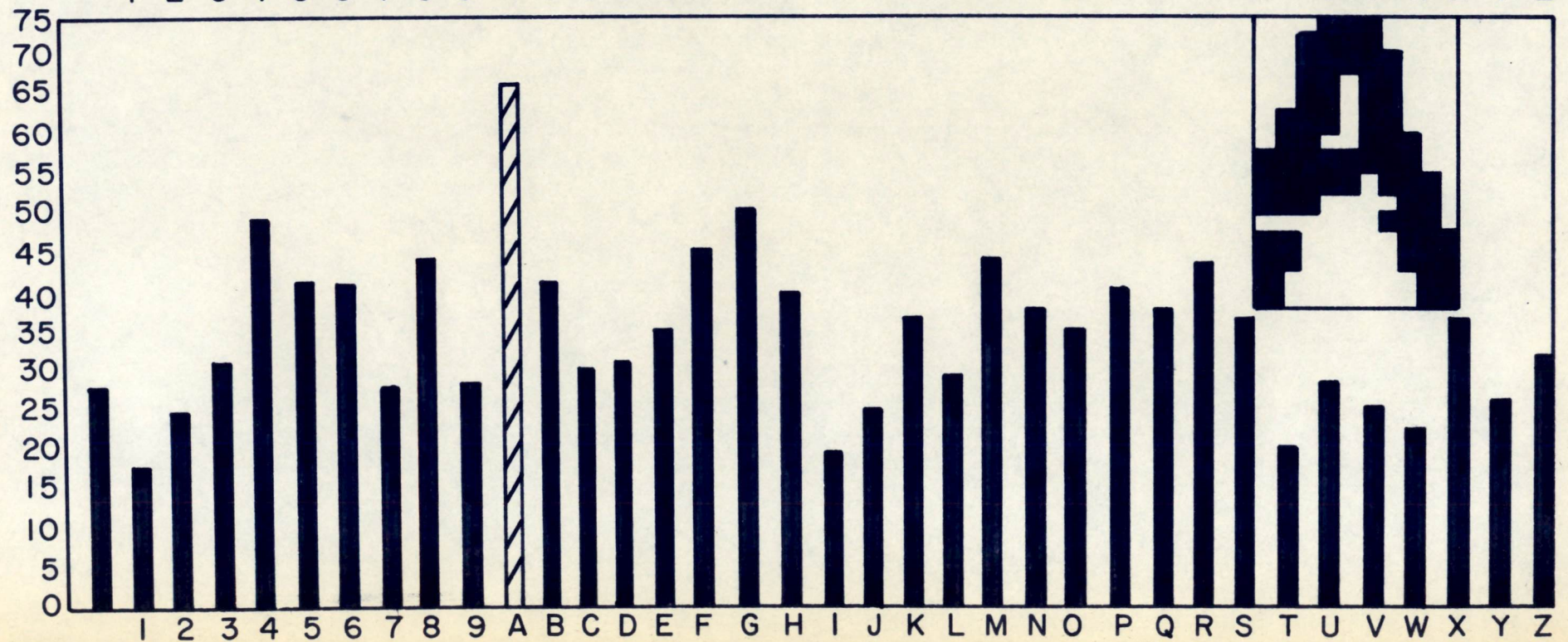
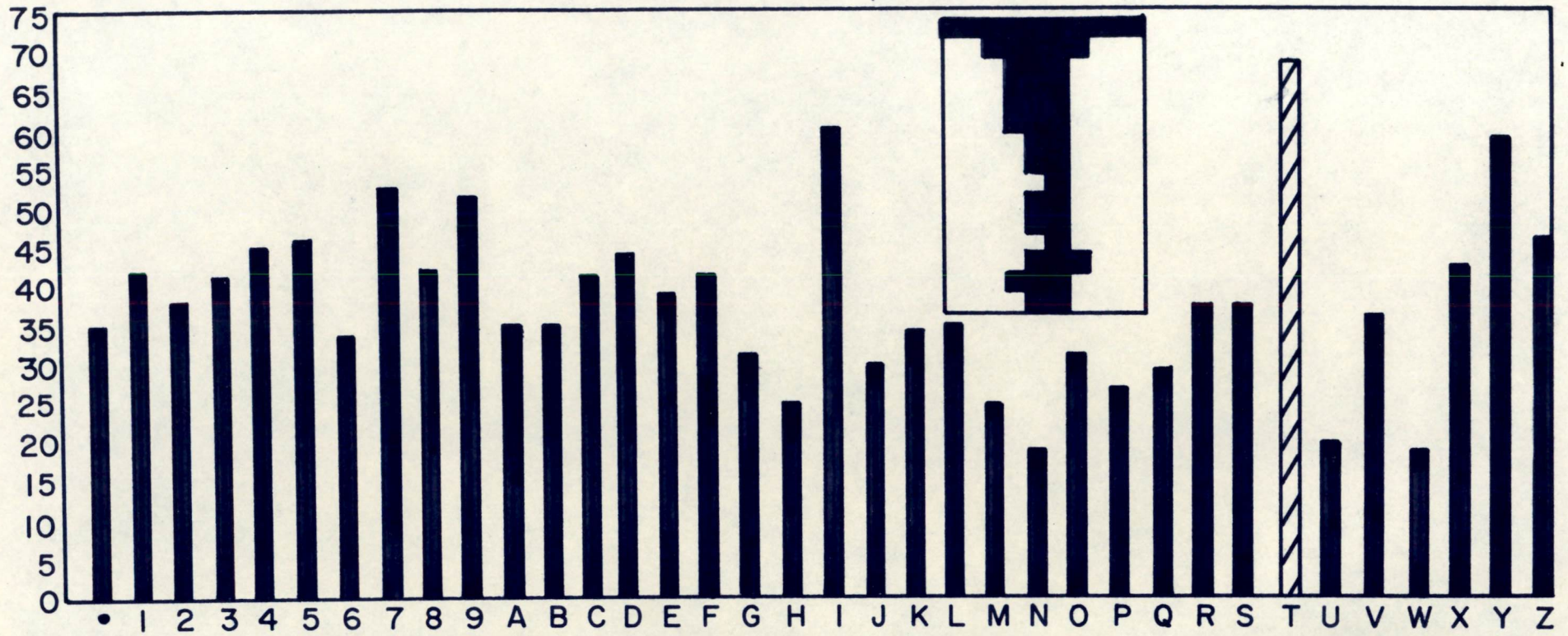
Copy No. _____
UNCLASSIFIED

Fig 26

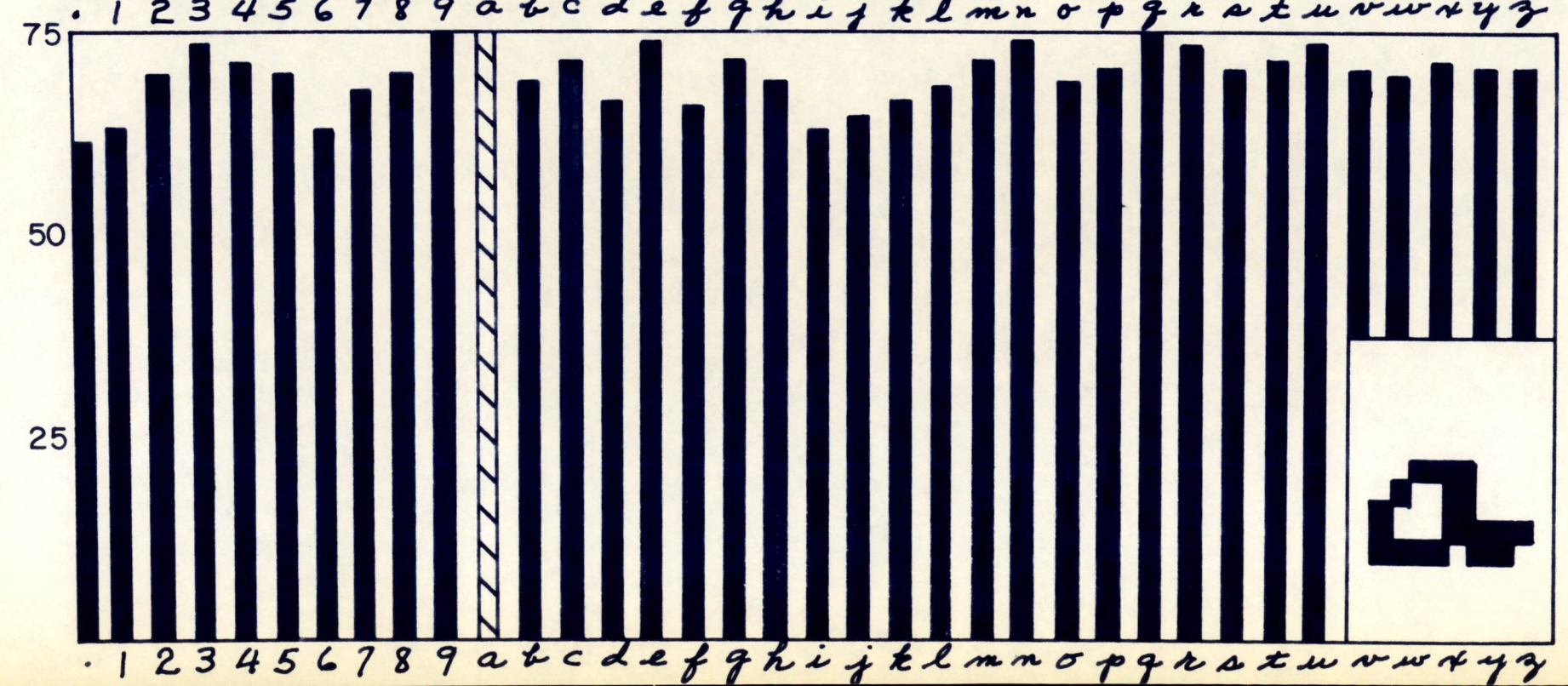
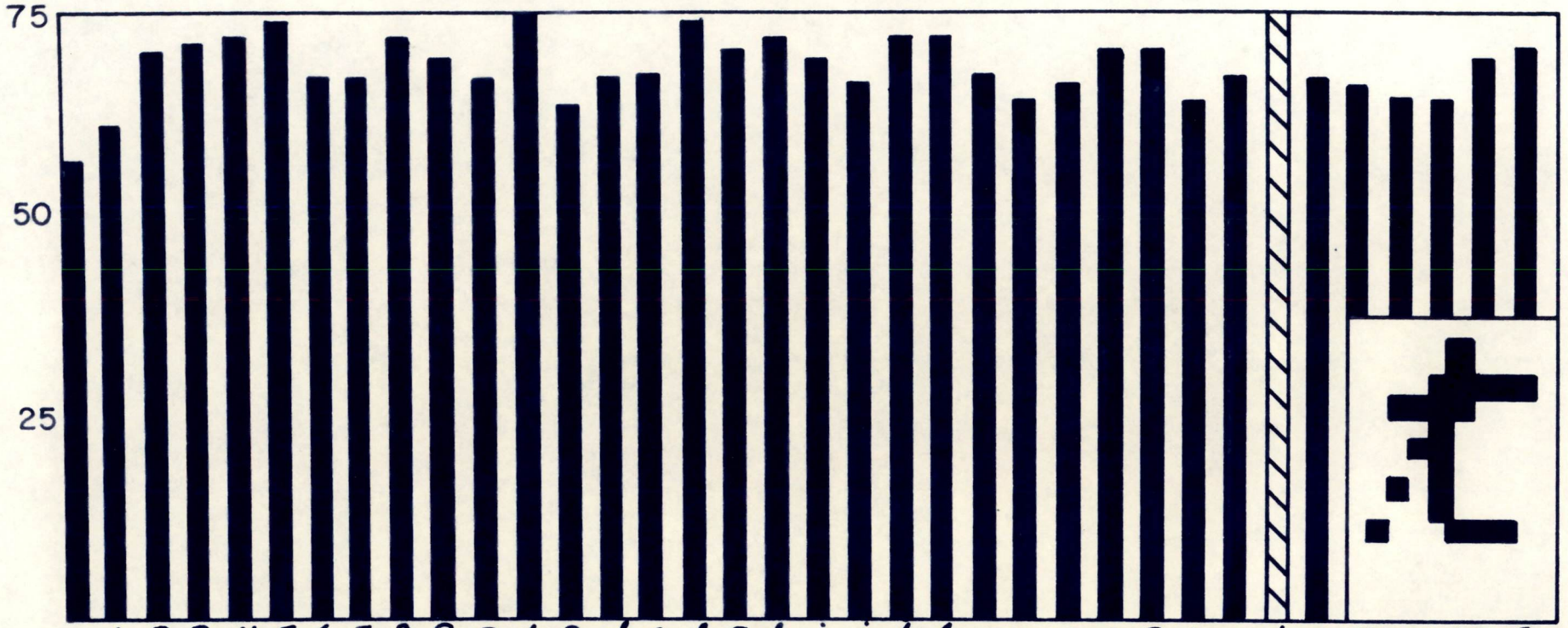
STATE		1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z				
1ST PAIR	00																																							
	01																																							
	10																																							
	11																																							
2ND PAIR	00																																							
	01																																							
	10																																							
	11																																							
3RD PAIR	00																																							
	01																																							
75TH PAIR	00																																							
	01																																							
	10																																							
	11																																							

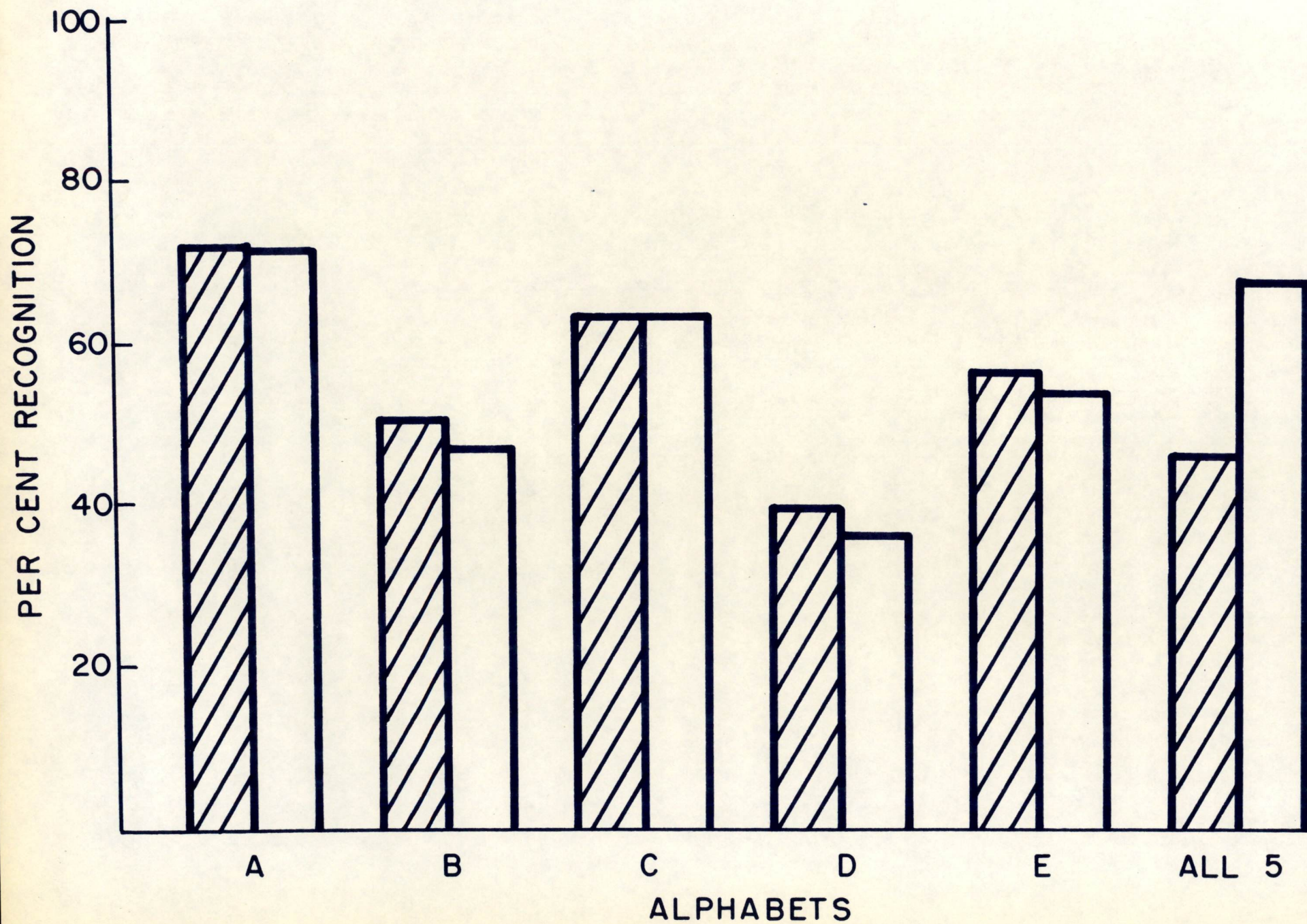
Fig 3

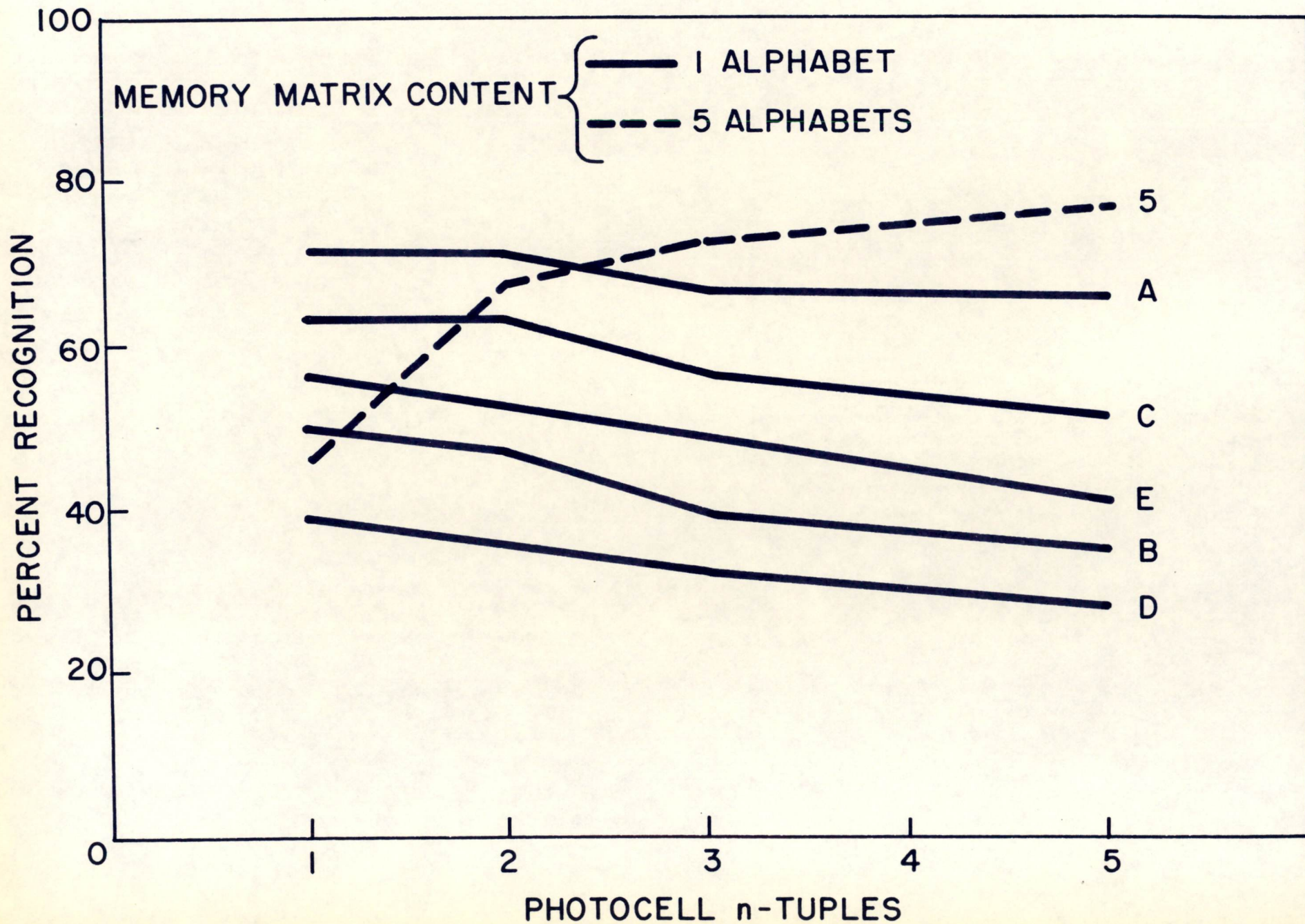
COMPARATIVE SCORES OF HAND BLOCK PRINT



COMPARATIVE SCORES OF HANDWRITTEN LETTERS







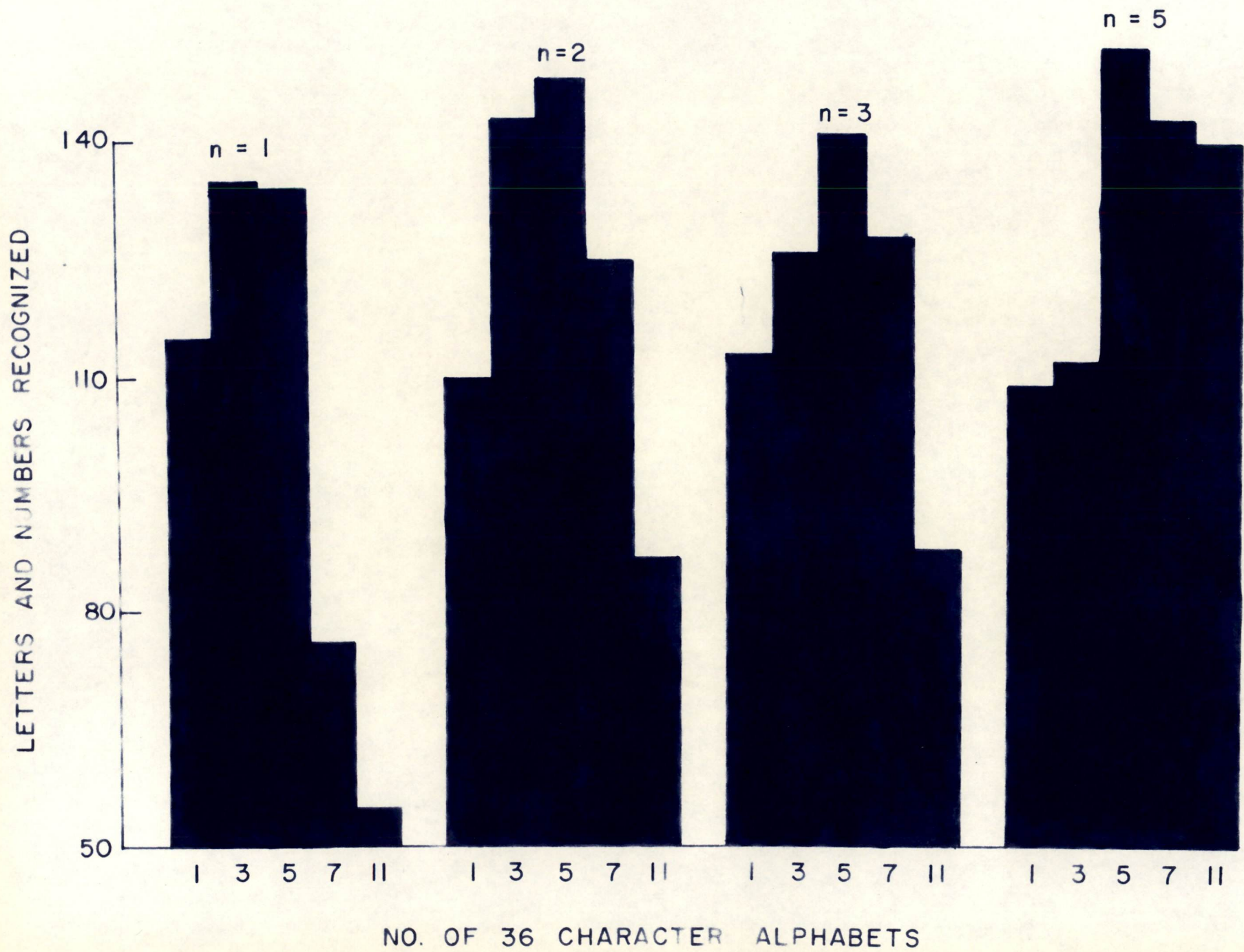
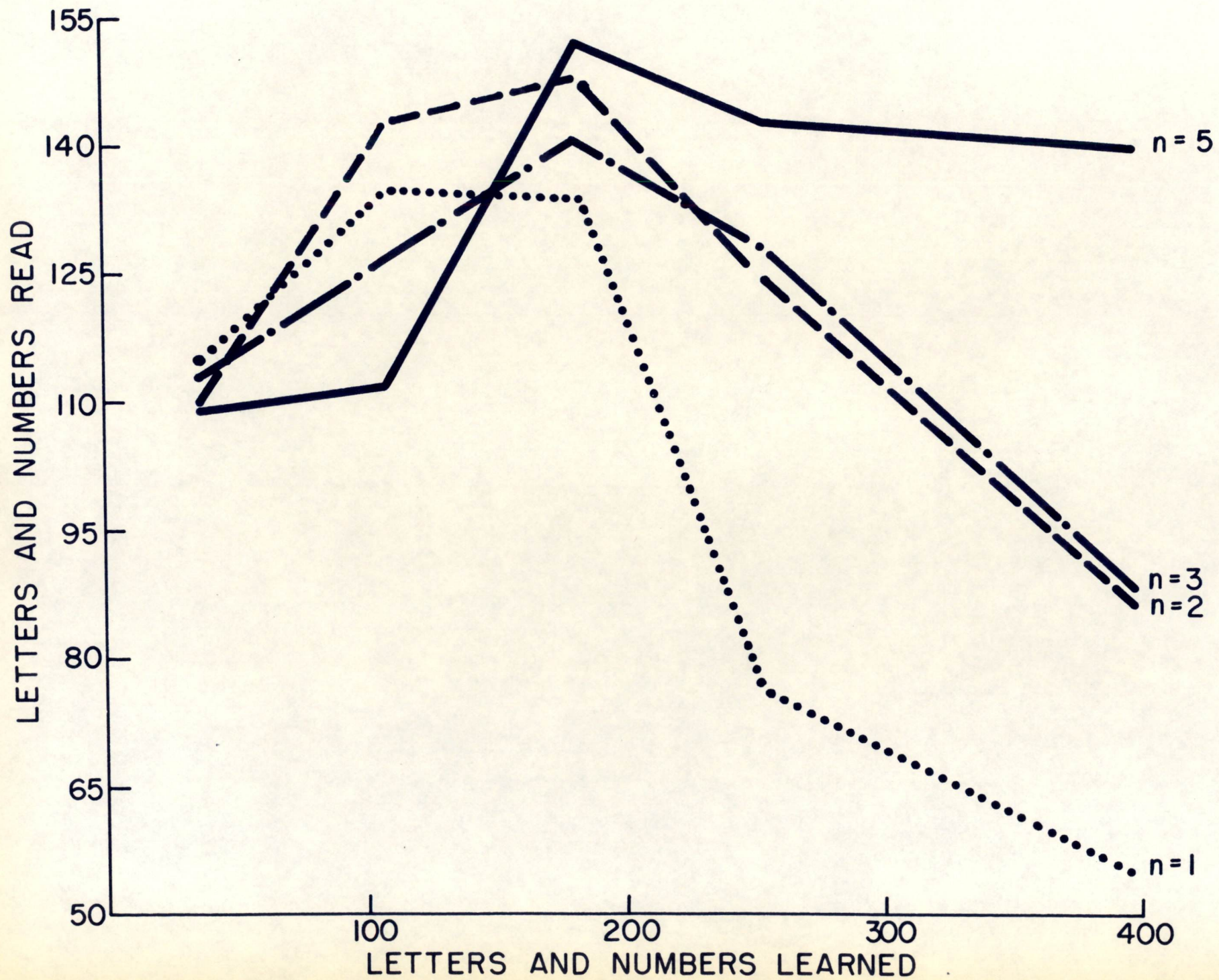
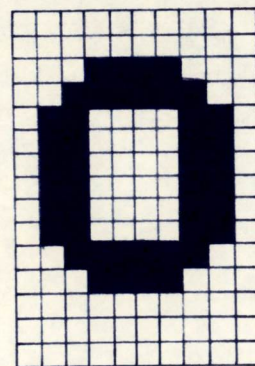
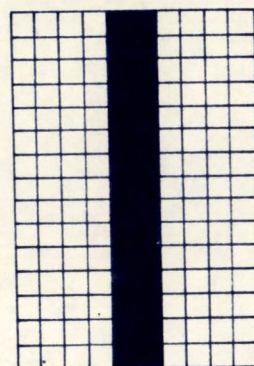
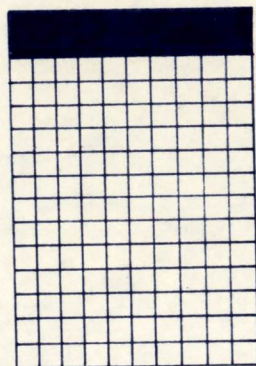
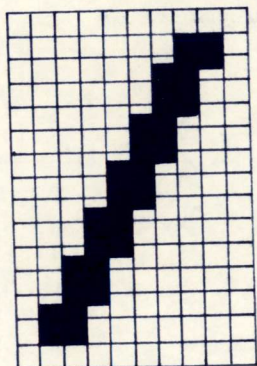
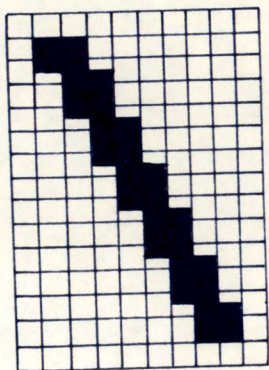
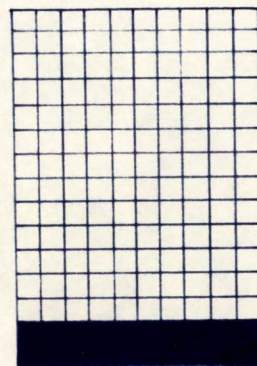
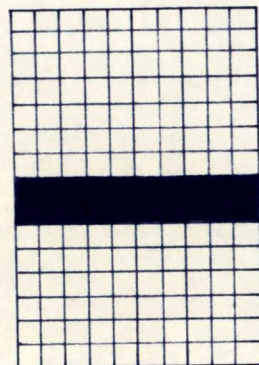
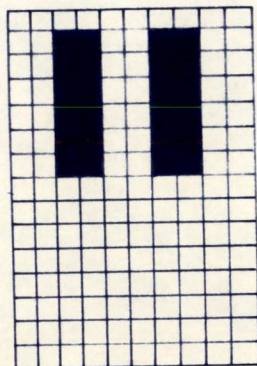
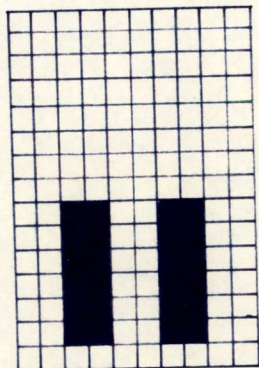
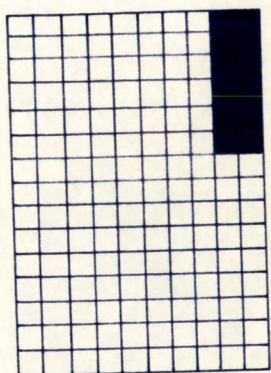


Fig 8a





n	CHOICE NUMBER					MEAN	σ
	I	II	III	IV	V		
HANDWRITING:							
2	31%	27%	32%	35%	35%	32%	3.0
5	36%	33%	33%	37%	36%	35%	1.7
HAND BLOCK PRINT:							
2	78.5%	78.2%	77.2%	77.8%	80.1%	78.4%	1.0

MESSAGE

THE COMPUTATION IS DONE BY THE USUAL MACHINE

FOR $n = 2$ (11 ALPHABETS)

LETTERS

TKU GXMPYTTTEN LU DEYT FY TTU UUEET MNQHTUU

CONTEXT

THE COMPUTATION IT DONE BY THE GREAT MACHINE

FOR $n = 5$ (11 ALPHABETS)

LETTERS

TKE GVMSUTUTIVN 2U DVUM BY TKU USMAD MNCHTUE

CONTEXT

THE COMPUTATION IS DONE BY THE USUAL MACHINE

PROGRESS IN READING HAND BLOCK PRINT

n- TUPLING		NO. ALPHABETS LEARNED	MANIPULATION	PERCENT READ
EXCLUSIVE	NON-EXCLUSIVE			
1		1	NONE	39- 72
5		1	NONE	28- 66
1		5	NONE	46
3		3	NONE	78
2,3, 5		2	PROBABILITY	77- 84
2,3, 5		3	DISTRIBUTION	80-84
	2, 3, 5	1	NONE	80-85
2, 3, 5		4	ROTATING ORIGIN	88-92
	3	1	ROTATING ORIGIN	96
2, 3, 5		1	CONTEXT	94-100
2,3, 5		1	CONTEXT- POSITIONING	98-100

Table I

PROGRESS IN READING HANDWRITING

n - TUPLING		NO. ALPHABETS LEARNED	MANIPULATIONS	PERCENT READ
EXCLUSIVE	NON-EXCLUSIVE			
1		1	NONE	26.14
1		3	NONE	30.68
2		1	NONE	25.00
2		5	NONE	33.64
5		5	NONE	34.55
5		3	DISTRIBUTION	43.84
	5	5	NONE	24.55
	5	5	POSITIONING	53.15
	5	11	NONE	50.00
	5	11	POSITIONING	58.56
	5	11	ROTATING ORIGIN	60.00
5		11	CONTEXT - POSITIONING	94.32

SCORING BY CONTEXT

	.	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
T	30	35	42	46	44	45	39	43	46	45	39	48	37	40	40	43	42	46	42	43	43	48	38	40	35	41	48	39	39	49	39	36	36	39	43	41
H	25	37	40	40	40	45	47	37	45	38	35	48	32	37	33	46	43	47	45	41	48	47	31	31	34	37	44	31	31	47	33	33	30	33	37	41
E	38	43	43	47	49	46	42	42	46	46	48	46	49	39	50	42	46	42	42	41	41	44	42	50	44	48	48	49	47	47	50	46	44	48	46	43

NOTE: WINNERS WERE T; K; U AND E TIED

n = 3 M = 50 CI = 396 Mu = 0

CORRECT WORD:

t = 49; *h* = 47; *e* = 50;

the = 49 + 47 + 50 = 146

OTHER WORDS:

a = 39; *r* = 31; *e* = 50;

are = 39 + 31 + 50 = 120

l = 48; *i* = 45; *e* = 50;

lie = 48 + 45 + 50 = 143

t = 49; *i* = 45; *e* = 50;

tie = 49 + 45 + 50 = 144



SPACE TECHNOLOGY LABORATORIES, INC.

4150 DOWNEY AVENUE, SUITE 100, DOWNEY, CALIFORNIA 90241
ARCHIVE REFERENCE: ORGARD 01515-241 CENTER, WASHINGTON 20547

BIOGRAPHY

Thomas G. Sanborn received a bachelors degree in Civil Engineering from Montana State College in 1951, and a masters degree in Mathematics from Ohio State University in 1954. From 1951 to 1953 he worked in both military and civilian capacities in the Aeronautical Research Laboratory, Wright Air Development Center, where he was engaged in the analysis of dynamic systems such as aircraft landing gear, flexible gun mounts, etc. From 1954 to 1958 he was employed by the El Segundo Division of Douglas Aircraft Company as a computer programmer in the Engineering Computing Group.

Mr. Sanborn joined Space Technology Laboratories in 1958 where, since November of that year, he has been working on SIMCOM, a compiler for generating computer simulation programs.

SIMCOM - The Simulator Compiler

(Abstract)

SIMCOM is a specialized compiler used in preparing computer-simulation programs. The input language consists of stylized "sentences" which specify the configuration of the computer to be simulated, and describe the bit-wise effect of each of the simulated computer's operations. Related sentences are grouped into "paragraphs" to minimize coded cross references in the simulation program. Because the compiler output is a symbolic code it can be generated during a single pass through the source program. This symbolic code is then processed by a conventional two-pass assembly program. Automatic storage allocation, and a novel form of subroutine add to the power of the system.

Thomas G. Sanborn
Member of the Technical Staff
Data Processing and Operations Dept.
Computation and Data Reduction Center
Space Technology Laboratories, Inc.

SIMCOM - The Simulator Compiler*

Thomas G. Sanborn
Space Technology Laboratories, Inc.

November 4, 1959

In many present-day activities involving the use of digital computers, the need often arises to run programs on a computer other than the one for which they are written. For example, the computer on which a program is intended to be run may exist only as a proposed design, or it may be in some stage of construction, or it may simply be at a remote location. One solution to the problem posed by such a situation is to prepare a program for an available computer which, in effect, transforms the available computer into the unavailable computer. Such a transformation program is called a computer simulation program, since it gives one computer the ability to simulate another.

Because of the intricate logical relationships which prevail in computers, the preparation of a simulation program is time consuming and fraught with opportunity for error. Furthermore, changes in the specifications of the computer being simulated may necessitate a major overhaul of the simulation program. For these reasons a new programming language and its associated compiler, SIMCOM (standing for Simulator Compiler), are being developed to assist in the preparation and modification of simulation programs which are to be run on the IBM 709.

It must be clearly understood that SIMCOM is not, itself, a simulation program. It is a generating program which accepts statements written in a specialized simulation-oriented source language, and from these statements generates instructions in SCAT language similar to those a human programmer would write in preparing a simulation program.

* Presently being developed under a purchase order from Thompson Ramo Wooldridge, Inc., in support of their contract to supply technical direction to the Automatic Data Processing Facility, U. S. Army Electronic Proving Ground, Fort Huachuca, Arizona.

The fundamental unit of SIMCOM coding is the statement. Each statement is either a definition of a component of the simulated computer or a description of some data manipulation or control function which occurs during the execution of instructions within the simulated computer. These two kinds of statements are known, respectively, as definition statements, and procedural statements. Related statements are grouped into paragraphs. SCAT coding, including SCAT-type remarks, may be intermixed with the paragraphs should the SIMCOM language prove inadequate for describing some involved procedure. The characters which may be used to write statements include the upper-case Roman letters, the decimal digits, and certain special characters. Combinations of alpha-numeric characters are called symbols. The three uses of symbols are: 1) to represent components of the simulated computer; 2) to identify locations within the simulation program; and 3) to denote integers. Every symbol, unless it represents an integer, must contain at least one alphabetic character, and no symbol may be identical to a word of the basic SIMCOM vocabulary.

A simulation program written in the SIMCOM language consists of three parts: the "Machine Definition;" the "Instruction Interpretation;" and the "Panel Operation." These sections describe, respectively, the static machine, the machine in operation, and the effect of operator intervention.

The Machine Definition is given in six paragraphs labeled "REGISTERS," "MEMORY," "INPUT," "OUTPUT," "KEYS," and "INDICATORS." Each definition statement describes a machine component or cell, giving its name, bit structure, and, if appropriate, its address or range of addresses. For coding convenience, a register may be defined as being synonymous with part or all of another register. Furthermore, registers can be defined which have no counterpart in the real computer being simulated. No distinction is made by SIMCOM between so-called primary and secondary storage.

Figure 1 shows examples of some typical definitions selected from several well-known computers. This figure also illustrates the basic requirements of the form on which SIMCOM coding is to be written. Some users

may wish to use a form on which each card column is marked since blank positions are frequently essential in the language. Note that the first line of each paragraph is indented to column 16 and that subsequent lines begin in column 8. The compiler uses this convention to aid in distinguishing between SIMCOM statements and SCAT instructions which may be included in the source code.

In all of the machine component definitions, the symbols used to identify the various devices are quite arbitrary, the only limitations being that they conform to the previously stated rules pertaining to symbols, and that they do not conflict with the basic vocabulary. The most common method of selecting symbols will undoubtedly be to adopt those used by the machine manufacturer in his manual since these are usually highly mnemonic.

The Instruction Interpretation and Panel Operation sections of the simulation program are written in terms of procedural statements. A procedural statement consists of a primary operation together with one or more operands called expressions, arranged to form a stylized sentence. Each primary operation is denoted by one or more words from the basic SIMCOM vocabulary. This vocabulary includes words for transferring, clearing, complementing, testing, comparing, and shifting arrays of bits in the various components of the simulated computer, plus words which control the logical flow of the simulation program and the compilation process itself.

The expressions upon which the primary operation act consist of symbols combined by means of secondary operations. These secondary operations include + (add), - (subtract), * (multiply), \$ (indirect address), and certain words of the basic vocabulary which denote logical arithmetic and scaling. A symbol in an expression may have bit designators appended to it if only part of the component identified by the symbol is to participate in the operation.

Figure 2 shows a few paragraphs of procedural statements. In the figure the expressions are underscored for emphasis but, of course, this

would not be the normal practice on a coding form. Note that the first paragraph bears a location symbol. In many instances, however, cross references between paragraphs are implicit in their relationship to one another. Therefore, many paragraphs will need no location symbols attached to them.

The instruction interpretation section is the heart of the source program. It will normally contain statements which describe the procurement of instructions from the simulated computer's storage, followed by statements which describe the effects of each instruction. The various instruction interpretations can usually best be initiated by use of a "table look-up" statement. Depending on the complexities of the instructions and the associated timing, each instruction description may require as little as one simple statement or as many as several paragraphs including, perhaps, entries to subroutines and additional table look-ups. Figure 2 illustrates a simple example of this technique.

A "Table," as understood by SIMCOM, is an ordered set of paragraphs of procedural statements, each paragraph being identified by an integer. The table look-up operation provides a means for selecting one of these paragraphs for execution, depending on the value of the argument expression. If a paragraph in a table does not terminate with an explicit transfer of control to some other point in the simulation program, then control returns to the statement following the "LOOK UP ..." statement which invoked the paragraph. Thus each paragraph in a table is like a closed subroutine.

The panel operation section of the simulation program includes an interrogation of the status of each console key and a description, written in SIMCOM statements, of the behavior of the simulated computer if the key has been activated.

It is not uncommon for certain keys on computer consoles to be so constructed that they are turned off as soon as the function which they perform has been initiated. The programmer's statements must include this action, if appropriate. Furthermore, in some cases certain keys are inoperative unless other keys or indicators are in a particular status. The programmer must also provide this logic.

One of the most interesting features of the system is the subroutine library. Subroutines are stored in the library in the SIMCOM language, except that the symbols denoting the subroutine parameters are replaced by variable symbols of a special kind. At compilation time, as a subroutine is called from the library, its special variable symbols are replaced by the parameter symbols given in the library call statement, and its location symbols are replaced by arbitrary unique symbols. The subroutine is then inserted into the source code where the SIMCOM decoder and instruction generator processes it in the same manner as any other set of SIMCOM statements. This process is illustrated in Figure 3.

A given subroutine may be called from the library any number of times during one compilation and, depending on the parameters listed in the library call statement, each version may give rise to a different number of 709 instructions. Each version of a subroutine called from the library is a "closed" routine which can be executed from any point in the simulation program.

The output from SIMCOM is a translation into SCAT language of the source program. This includes a direct expansion of the procedural statements, plus certain pseudo operations for assigning storage and certain utility routines whose necessity is only implied by the source language. These include routines for loading the simulated computer, diagnostic output routines, and, of fundamental importance, a routine which allocates the simulated computer's storage to the various 709 storage media. This storage management routine must partition oversize words, should such have been defined, into the 36-bit words of the 709, and shuttle simulated computer storage to and from 709 tape units if it exceeds the capacity of the 709 core storage. The endowment of the compiler with the ability to generate efficient storage management routines is, without doubt, the most challenging problem facing the creators of SIMCOM.

Figure 4 is a schematic representation showing the allocation of the generated program to the various parts of the 709. The heavily outlined areas indicate the parts of the 709 used to represent the various registers and storage of the simulated computer. The remainder of the 709 contains

the generated simulation program and its associated utility routines. The arrows indicate the communication paths between the various areas of the 709.

Because the SIMCOM output is in SCAT language, the compiler need not contain within itself an assembly program, nor does it have to be able to process the SCAT instructions included in the input code other than to recognize them as SCAT instructions. Most important, however, is the fact that generation from SIMCOM statements to SCAT instructions can be done during a single pass through the source program. In addition to generating SCAT language instructions, the compiler transforms each line of SIMCOM coding into a SCAT-type remark (* in column 1) and inserts each paragraph into the generated code immediately ahead of its SCAT language expansion. Thus each paragraph serves as commentary to describe the function of the generated SCAT instructions which follow.

The SIMCOM language is such that apparently minor modifications to the input statements can completely alter the character of the generated program. For example, a change in the definition of a register of the simulated computer may cause SIMCOM to generate instructions to do multiple precision arithmetic where single precision arithmetic was formerly sufficient, or a change in word size in the simulated computer may cause SIMCOM to reorganize completely the simulated computer's storage in the 709 core. Thus, changes which could be made to a machine-like language (such as SCAT) simulation program only by completely rewriting the program, can be incorporated into a SIMCOM-written simulation program by a simple re-compilation.

The SIMCOM system will provide a means whereby users who are not necessarily professional programmers may prepare simulation programs for binary computers in a language not unlike that used by computer manufacturers in their manuals. There seems to be no escaping the fact that the user will need to be more than casually familiar with the computer to be simulated before he can write an adequate simulation program, even with SIMCOM.

LOCAT.	TEXT		
1	6	8	16
			72
			REGISTERS. MR(S,I-35). AC(S,Q,P,I-35).
		PCR(35-0).	UAK(I4-0) SYN PCR(29-15).
			MEMØRY. CØRE(S,I-35) 0-4095.
		DRUM(35-0)	16384-32767.
			INPUT. CARD(S,I-35).
			ØUTPUT. PRNTR(S,I-35).
			KEYS. RESET. LØAD. START. MJI. MJ2. MSI. MS2.
			INDICATØRS. ØVFLØW. IØCK. DVCK. EØTA. RUN.
		TRAP.	

2992

FIGURE 1. SIMCOM CODING FORM SHOWING TYPICAL DEFINITION PARAGRAPHS.

LOCAT.	TEXT		
1 6	8	16	72
LØØP		<u>IC</u> \$ TØ <u>IR</u> . <u>IC + 1</u> TØ <u>IC</u> . LØØK UP <u>IR(1-5)</u> IN	
	<u>ØPER</u> TABLE.	CLØCK <u>4</u> . EXECUTE <u>LØØP</u> .	
ØPER		TABLE.	
		0. <u>IR(6-17)</u> TØ <u>IC</u> . TURN <u>RUN</u> ØFF.	
		1. <u>IR(6-17)</u> TØ <u>IC</u> .	
		2. IF <u>ØVFLØW</u> IS <u>ØN</u> , TURN <u>ØVFLØW</u> ØFF.	
	<u>IR(6-17)</u>	TØ <u>IC</u> .	
		5. EXECUTE <u>GET</u> . EXECUTE <u>SUBTRA</u> .	
		CLØSE.	

2993

FIGURE 2. TYPICAL PROCEDURAL PARAGRAPHS SHOWING INSTRUCTION PROCUREMENT AND INTERPRETATION TECHNIQUE. "EXPRESSIONS" UNDERSCORED FOR EMPHASIS.

THE SUBROUTINE IS ORIGINALLY CODED AS

```
        SUBROUTINE ADD, A, C.  
A + B TØ C. CLØSE.
```

THE LIBRARY MAINTENANCE ROUTINE WILL PLACE THE SUBROUTINE IN
THE LIBRARY IN THE FORM

```
ADD          V1 + B TØ V2. CLØSE.
```

AT A SUBSEQUENT COMPILATION, A STATEMENT OF THE FORM
LIBRARY ADD, P, Q.

WILL CAUSE THE SUBROUTINE TO BE INCORPORATED INTO THE PROGRAM AS
ADD P + B TØ Q. CLØSE.

2994

FIGURE 3.
SAMPLE SUBROUTINE SHOWING GENERALIZED VARIABLE TECHNIQUE

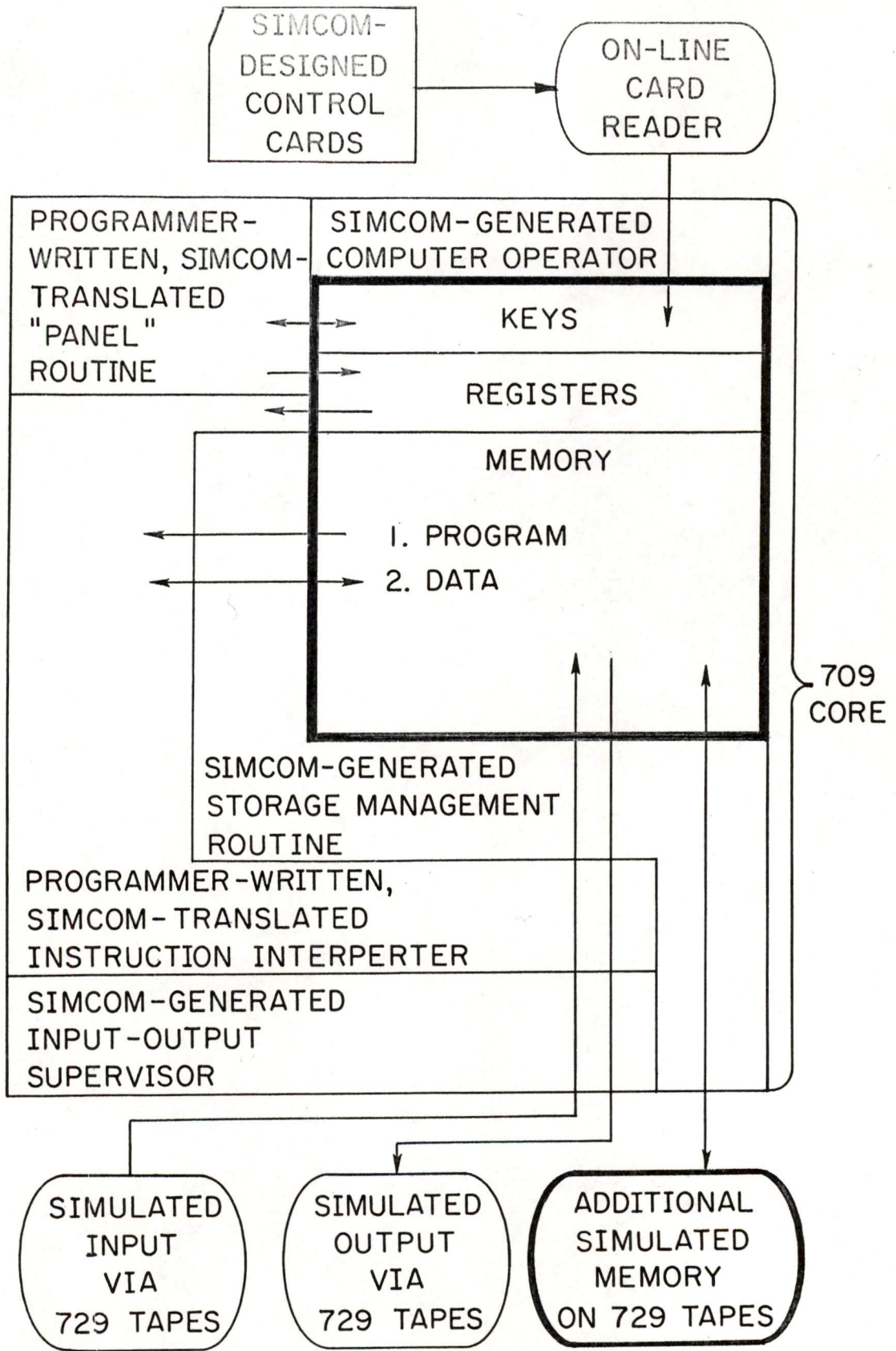


FIGURE 4. ALLOCATION OF ELEMENTS OF A SIMCOM-PREPARED SIMULATION PROGRAM TO 709 CORE AND TAPES



SPACE TECHNOLOGY LABORATORIES, INC.
POST OFFICE BOX 95001 • LOS ANGELES 45, CALIFORNIA
ARBOR VITAE FACILITY ORCHARD 0-1311 • R&D CENTER OSBORNE 5-4677

November 9, 1959

AIR MAIL

Mr. Harlan E. Anderson
EJCC Publications Committee
Digital Equipment Corporation
Maynard, Massachusetts

Dear Mr. Anderson:

Enclosed are four (4) copies of the manuscript of my paper "SIMCOM - The Simulator Compiler," together with the glossy prints for use by the publisher, and the biographical sketch requested in Mr. Felker's letter of September 24.

In the oral presentation I will be using 35 mm glass slides mounted in metal frames 2" x 2", and approximately 3/32" thick.

Yours very truly,

SPACE TECHNOLOGY LABORATORIES, INC.

Thomas G. Sanborn

Thomas G. Sanborn
Member of the Technical Staff
Computation and Data Reduction Center

TGS:el

Encl.

GILBERT
SUPERASE BOND
25% COTTON FIBRE

File 607

SYLVANIA *Electronic Systems*

A DIVISION OF SYLVANIA ELECTRIC PRODUCTS INC.



DATA SYSTEMS OPERATIONS
189 B Street, Needham 94, Mass. • Hillcrest 4-3940

November 16, 1959

Harlan E. Anderson
Digital Equipment Corp.
Maynard, Mass.

Dear Mr. Anderson:

Enclosed are the drawings from S. Chao and A. Ashley. Permission granted for one day's grace beyond deadline date of November 15th for E.J.C.C. paper via telephone conversation between H. Anderson and R. Graham, Friday morning, November 13, 1959.

Very truly yours,



Robert L. Graham
Technical Writing Department

File Copy

The System Organization of MOBIDIC B

By Stanley K. Chao

Data Processing Laboratory
Sylvania Electric Products, Inc.
Needham, 94, Massachusetts

SUMMARY

MOBIDIC B is an all transistorized, militarized computer mounted in a standard Army trailer. It is a general-purpose, parallel, binary, synchronous, fixed point, and duplexed data processing system.

It contains two basic processors, identical in characteristics and internally tied to the same system transfer bus. Both processors share a common set of input-output devices and each processor is capable of operating on an independent program without interference. They are also capable of duplex operations, allowing either processor to monitor and exert control over the other.

In addition to the 8192-word high speed Core Memory in each processor, there exists a 50 million-bit Mass Memory. This memory is treated as an input-output device, addressable by in-out instructions. A Data Retrieval Unit is incorporated to facilitate data searching from the Magnetic Tape and the Mass Memory. The control console is also duplexed, containing two independent and identical panels, one for each processor.

INTRODUCTION

There are a number of large-scale, general-purpose, mobile digital computers being developed at Sylvania¹. They are known as the MOBIDIC family of computers. MOBIDIC C and D are identical in internal design of MOBIDIC A². They differ only in peripheral equipment.

MOBIDIC B is the second member in the MOBIDIC family. It will be used primarily in the field to meet the Army's data processing requirements, and is to be installed in a standard 30 feet Army trailer.

As subsequently shown, MOBIDIC B contains all the instructions utilized in MOBIDIC A and also an additional 12 new instructions. To minimize equipment, some of the instructions have been made subroutines which are initiated automatically.

MOBIDIC B is a duplexed data processing system. It contains two identical central processors connected to a common system transfer bus. In addition, each central processor has an individual Real Time System which provides direct communication with external FIELDATA equipment. Each of the two central processors is a general-purpose computer. They can use any of the input-output devices available and either processor can run separate programs without interference from the other. In addition, the two processors can operate together and communicate with each other through an in-out device such as a magnetic tape. Either processor can use the other's core memory through an input-output device. It is also possible for either processor to monitor and exert control on the other through the direct connections between them.

SPECIAL REQUIREMENTS

There are 4 major requirements of a special nature imposed on this computer. The first is that MOBIDIC B is to be a duplexed data processing system. It contains two basic central processors, identical in characteristics, capable of either simplex or duplex operation. The second requirement is that it must be compatible with other MOBIDIC computers. Compatibility is sought not only through instruction type and word format, but also through the interchangeability of input-output devices and the physical element-cards and packages. Minimum equipment is another design criterion. This is achieved by having all full-length registers simulated in the core memory and also by mechanizing some of the infrequently used instructions through automatic subroutines. Only two

full-length working registers are used within each processor. Data retrieval capability is the fourth of the special requirements. Data stored on either the magnetic tape or the mass memory must be retrieved at full device speed. This continuous and speedy retrieval ability necessitated the incorporation of a Data Retrieval Unit together with the addition of some special instructions

INSTRUCTIONS AND WORD FORMAT

INSTRUCTIONS

There are 64 instructions in MOBIDIC B (Table 1). They are classified into 3 categories, namely, Directly-Mechanized Instructions, Subroutine Instructions, and Special Instructions. There are a total of 40 Directly Mechanized Instructions. In most cases, the execution time has a variation of 2 μ s. The instruction LGM, for example, may sometimes be executed in 36 μ s instead of the 38 μ s shown in Table 1. The variation is due to the fact that the system bus is time-shared by both processors. The system bus may or may not be available to the processor which is executing the instruction that requires access to it at that particular instant. The times given in Table 1 are maximum.

There are 15 Subroutine Instructions. These instructions are performed through subroutine programs, which must be stored in memory and executed automatically. As such, all instructions used in the program to execute the Subroutine Instruction must be of the Mechanized Type.

Nine of the MOBIDIC B OP codes are unassigned. They are available to perform any special subroutine operation that may be required in a particular application. Whenever one of these "special" instructions is decoded, operation is automatically transferred to a unique location from which the program is re-routed to the desired subroutines.

According to their logical functions, the 64 instructions can be classified as follows: 16 arithmetic operations, 17 sequencing and indexing, 10 input-

output; 12 editing and data handling; and 9 special purpose instructions.

WORD-FORMAT

The length of the MOBIDIC B word is 38 bits; the length used in other MOBIDIC computers. The word format is illustrated in Figure 1. Numerical data is represented by a fixed point, magnitude and sign conventions. Magnitude is registered in bits 1 to 36. The binary point is understood to be placed between bits 36 and 37. Bit 37 is used for sign of a number stored and bit 38 is used as a parity check bit. Alphanumeric data is represented in the same manner as numeric data, except that the sign bit is eliminated and the 36 bits are grouped into 6 alphanumeric characters. Since MOBIDIC uses a weighted code, alphanumeric data can be sorted without conversion to binary form by direct use of the logical and arithmetic operations.

A standard instruction word is divided into 6 parts:

- 1) Major address (α): Bits 1 to 12 specify a memory address while bits 13 to 15 specify which memory will be used. Since many of the internal MOBIDIC B registers are addressable, one of the eight configurations for bits 13 to 15 represents internal register addresses. The actual register addresses in these cases are specified by bits 1 to 5.
- 2) Minor Address (β): Bits 16 through 27, the β bits, have several uses, depending on the particular instruction being performed. They may be loaded into or added to the contents of an index register. The β bits, either alone or in combination with γ may also be used to specify a second address.

- 3) Index Register (γ): Bits 28 to 30, γ bits, are used primarily for indexing. They specify which, if any, of the Index Registers are to be used with the instruction. For some instructions, γ is used as part of a second address.
- 4) OP Code: Bits 31 to 36 designate the instruction to be performed.
- 5) Spare: Bit 37 is a spare.
- 6) Parity: Bit 38 is the parity bit.

The format for input-output instructions is identical to that of standard instructions except for the assignments made to bits 16 to 30, the j and k portions of an in-out instruction. The j bits, 16 to 21, are used to specify the particular input-output device addressed. Sixty-three input-output devices maybe handled in this manner. The k bits, 22 to 30, are used to specify the number of words or blocks to be processed.

OVERALL SYSTEM DESCRIPTION

The block diagram of the MOBIDIC B system is given in Figure 2. In it are shown: two identical computers (processor 1 and processor 2), two In-Out Converters, a Data Retrieval Unit, a Mass Memory Unit, two Real Time Systems, and a family of In-Out Devices. The two processors are fed by a common system clock to facilitate system synchronization. They are connected to a common system transfer bus and also share the same in-out system. The processors can operate independently as separate computers or can communicate with each other through magnetic tape. It is possible for either of the two processors to give a tape instruction which will read from or write into the memory of the other processor. It is also possible for one processor to monitor the other through the signal lines connecting them directly. Furthermore it is possible for one

processor, to control and give commands to the other processor.

The In-Out Converters serve as data-synchronizing format converters. They are capable of handling all devices including the mass memory unit. The converters are assigned to either of the two processors on a first-come-first-serve basis. Since the two processors are connected to a common system bus, it is entirely possible that both may want to get on the system bus at the same instant. To avert this uncertainty, the System Bus Control circuit is used to continuously assign the system bus to the processors alternately. Each is given a 2 μ s period to use it. Therefore, a waiting period of 2 μ s may be required at an arbitrary time. The danger of both processors trying to get to the same converter is also avoided by the same control circuit.

In addition to the standard family of MOBIDIC in-out devices, the MOBIDIC B system includes a 50-million-bit memory which is also treated as an in-out device. Data transfer to and from the mass memory will be handled by the converter through the in-out bus. The data in the mass memory is arranged in blocks, separated by block start and block end marks. The block number on each track is addressable. Both the track and block addresses must be loaded into the Mass Memory Control Unit prior to giving the write or read instruction. These addresses are sent out from the processor through the converter, similar to the manner by which the data is transferred.

The Data Retrieval Unit is designed to assist the open format search program. A retrieval program examines the specified fields to determine whether or not the search criteria are satisfied. Closed format search can be accomplished entirely by programming and does not require any auxiliary equipment.

The two Real Time Systems enable the processors to communicate with other data processing equipment external to the MOBIDIC B system. Each has a fixed assignment to serve one definite processor and is not accessible to the other processor.

On the other hand, the balance of the entire input-output system does

not have a fixed assignment. Operating on a first come first serve basis, it is entirely possible for one processor to automatically monopolize the use of both converters, the Data Retrieval Unit and a complement of devices. Without a converter the other processor cannot reach any device even if it is available. Under such a circumstance, the other processor would have no choice but to wait for a converter to become available. In some special cases, the other processor may have just received, through its own Real Time System, an urgent request which requires data processing through the service of a converter. Upon receiving such a request, it is possible for the other processor to shorten the In-Out operation of the first processor, making the converters available.

BRIEF DESCRIPTION OF PROCESSOR

Since minimum equipment is a major design requirement, most full length registers normally existing in other MOBIDIC Computers are stored as locations in the memory. These reserved memory locations are referred to as simulated registers. Simulated registers include the Accumulator, the Q-register, the B-register, the Program Counter, the Program Counter Store, and seven index registers. They can be addressed in exactly the same way as their counterparts in other MOBIDIC computers. As shown in Figure 2, only two full-length physical registers are used in each MOBIDIC B processor, the Memory Register (MR) and the Control Register (CR). These registers and other essential parts in the processor are described as follows:

Timer The timer, containing three counters, receives pulses from the system clock. The processor executes instructions by proceeding through a sequence of events. Certain events, such as memory operations, occur so frequently that a separate counter TM is used to control these operations. The execution of instructions require several memory operation. Counter TI indicates which of the several memory operations is currently in progress. Finally, if

an input-output access to the central processor is required, the instruction execution must be interrupted and a new sequence of events must start. A third counter TB controls these input-output processing operations.

Core Memory Each processor of the MOBIDIC B system is provided with two 4096-word core memories with a read-write cycle of 8 us. It can be readily expanded to 4 memories per processor when desired. It can be ultimately expanded to 7 memories, totaling 28,384 words.

Memory Register The MR is directly connected to the memory. It is a 38-bit register and is used as the memory in-out register. The MR is also used as an arithmetic register during execution of the instructions.

Control Register The CR serves primarily as a 37-bit arithmetic register corresponding to the accumulator in MOBIDIC. In addition, the first 15 bits of CR are also used as the memory address register during initial access to the high-speed memory.

Decoder Register This is a 6-bit register used to store the instruction while it is being executed. Its output interprets the instruction stored and energizes appropriate control lines to initiate execution of the instruction specified.

Control The control unit contains the logical circuits to control all of the detailed operations of the computer.

Special Address Control The special address control unit contains the decoders and control circuits to address the core memory locations which are reserved for special registers of the processor and the Data Retrieval Unit. Among the registers specified by the special address control unit are most of the simulated registers. The contents of these registers can be transferred to the MR and then to the CR whenever it is requested.

T. Counter The T Counter is a 7-stage counter used in both shifting and multiplying operations.

System Clock The system clock provides standard p and t pulses spaced one microsecond apart to the entire system. There is a separate clock for each processor. This makes the processors identical. One system clock may be used to control the entire system operation when the processors are working together depending upon which processor is in full control of the program.

System Bus Control The system bus control circuits regulate and direct the flow of traffic between the two In-Out Converters, the Data Retrieval Unit, and the two processors. These control circuits give either processor access to the system bus.

INPUT-OUTPUT SYSTEM

A more detailed description of the various components in the In-Out system will now be given. Reference should be made to Figure 2.

In-Out Converter There are two In-Out Converters in MOBIDIC B. They are used as buffers between the input-output devices and the central processors. They assemble data coming in from a device and put it into MOBIDIC word format before transferring it to the central processor. Conversely when data is to be sent out to the device from the central processor, the converters decompose the standard MOBIDIC word and reassemble it into the proper format for the particular device which is to receive it. In addition, converters also have the function of synchronizing the operation of the devices with that of the central processor. In this way, the information transfer between the converter and the device can take place quite independently from the operation of the central processor. Internal computation is only interrupted during access to the memory.

Data transfers between converters and processor memories are handled on a "busy-bit" basis over the system bus. As soon as a converter is selected

the processor will transfer the entire In-Out Instruction to the Converter. The processor subsequently goes on to execute the next instruction and exerts not further control to the converter. The converter, taking upon itself to execute the instruction it has just received, proceeds to communicate with the device addressed and sends out or receives data from the device. When the converter is ready to send in or to receive another word from the processor, it will signal the processor by raising a busy-bit. Detecting a busy bit, the processor will interrupt its operation and take care of the converter.

Real-Time System There are two identical Real Time Systems. Each system consists of an input register, an output register and an input address register. Each Real Time System is assigned to a processor which provides the communication link between the two central processors of a single MOBIDIC B system between two MOBIDIC B systems, between a MOBIDIC B and MOBIDIC A computer or with other FIELDATA computers and communication equipment. In all cases except the last, the Real Time Output Register in one Real Time System can be directly connected to the Real Time Input Register of the other Real Time System. In the FIELDATA application, a buffer unit may be required between the Real Time System and the external equipment. For example, A Kineplex is required when the teletype communication equipment is connected to MOBIDIC B.

Data Retrieval Unit The DRU is a special unit designed to assist the data retrieval program from the storage files. It is connected to the system bus as well as the In-Out bus. It will examine all the data being transferred from the Magnetic Tape (or Mass Memory) to the converter. After extracting the desired portion, the data is then sent to the core memory for further processing.

Mass Memory The Mass Memory is needed to provide an exceptionally large data storage capability. It consists of 8 magnetically coated discs, giving

a total of 16 usable disc sides. There are 4,096 tracks on which the data can be stored. Storing is done in a serial-serial manner which can be continuous from one track to the next. Track switching is done automatically. There are two magnetically engraved clock tracks, one at 150 KC bit rate and the other at 225 KC bit rate. The maximum random access time of the mass memory is less than 0.5 seconds.

Magnetic Tapes A total of 8 magnetic tapes are currently provided in MOBIDIC B. They could be either the commercial FR-300 type or the militarized type. They have 8 channels which incorporate a parity error detection channel. The nominal tape speed is 150 inches per second (reversible) with approximate start and stop time of 1.5 ms. The nominal character rate is 45 KC.

Flexowriter The Flexowriter is a special electric typewriter that operates at a speed of 10 characters per second. It can be used on- or off-line or as an output device for producing hard copy or punched paper tape.

Paper Tape Reader There are two photoelectric paper tape readers, one being a 5-hole and the other an 8-hole reader. These input devices have a nominal reading rate of 270 characters per second.

Paper Tape Punch The two Paper Tape Punches include a 5-hole punch and an 8-hole punch; both are directly adaptable to the reader. Both types prepare punched paper tape at a nominal character rate of 100 per second.

DUPLEXING CAPABILITIES

The two MOBIDIC B processors are tied to a common system bus and share a common set of in-out equipment. It is beyond the scope of this paper to give a detailed description of the entire duplexing capabilities existing in MOBIDIC B. A separate technical paper is to be published treating this subject in greater detail. Briefly, there exists a set of control lines connecting the

two processors directly. Through these lines the operating status of one processor can be monitored by the other processor. Through these control lines also, one processor can exert control and give command to the other processor in a limited manner. In particular, one processor can prevent the other from coming to a complete halt condition, thus keeping the other processor in a state of readiness to accept information which may be transferred into it from the first processor. Moreover, one processor can restart the other after that processor has completed a program. In addition, one processor can give an input-output instruction which is to be executed by the other processor. For example, one processor may give a write instruction to have information contained by the other processor written out onto a magnetic tape. That same controlling processor can subsequently give a read instruction to have the same information read from the magnetic tape back into its own memory for immediate use. Thus one processor can effectively make use of the data stored in the other processor's memory. Conversely, one processor could give a set of instructions which will result in the transformation of data from its memory into the other processor. For this type of operation the programs in the two processors must be coordinated. Some circuits are built into the MOBIDIC B system to direct such traffic between the processors and avoid any uncertainty as to the direction of information flow between them.

Full utilization of the "built-in" duplexing capabilities of MOBIDIC B should provide a challenge to the imagination and foresight of programmers. Many programs could be written to take advantages of these duplexing facilities. For example, one processor could enter into a different program as a result of the decisions made by the other processor. Also, one processor could take over the other's task if it discovered that the other processor was either overloaded or incapacitated.

MARGINAL CHECK AND CONTROL CONSOLE

An automatic marginal checking system is incorporated in MOBIDIC B. The checking circuit is so designed that the bias voltage in each row of every rack in the computer is modified by a predetermined amount. While the bias voltage of the row is maintained at this changed value, a simple check program, which is stored in the computer, will be run through once. The result of this program can be observed, an error condition will be indicated by a pilot light on the console. All rows are automatically tested in succession in such a manner. It is possible to bypass some racks in the computer so that marginal voltages are not applied and checking is not performed on them. This is necessary to enable one processor and some associated input-output equipment to be in continuous operation while the other processor is undergoing test.

The control console for MOBIDIC is also duplexed. It consists of two independent and identical consoles assembled side by side. Each console is permanently connected to one processor and thus communicates only with its assigned processor. For ease of operation, each control console is divided into 6 horizontal areas. At the very top of the console is located the marginal check voltage control. Immediately below this area is the control for the power to the computer. The master clock selector is also located in this area. The display register is situated next in line, extending completely across the console. This is a full length indicating light register which is used to display the contents of any register or memory location selected by the operator. The area below holds the controls for the flip-flop and error detection. Directly beneath are the controls for the insertion of manual instructions. Initiation control for the computer such as start, halt, and single pulse, are laid out in the bottom row on the control console.

Inasmuch as the control consoles are assigned to their respective processors, operation of one console is entirely independent from the other. However, since the two consoles are located side by side, the operator can easily observe the

status of both processors, allowing convenient control of both processors during duplex operations.

CONCLUDING REMARKS

Since there are only two physical working registers in each central processor, connections from the central processor to the in-out system are made only through the Memory Register. Traffic coming in and out of the memory as well as going to and from the in-out system frequently create logical problems. Resolution of these logical problems result in a slight reduction of computation speed.

The internal duplexing features introduced in MOBIDIC B represent a new approach in the design of a large scale, general purpose data processing system. There will undoubtedly be many areas where such an approach is highly desirable from the standpoint of reliability and economy. Full utilization of the built-in duplexing capabilities in MOBIDIC B will unquestionably challenge the imagination and foresight of the users, programmers and engineers alike.

ACKNOWLEDGMENT

Many people have contributed to this project. In particular, Messrs. W. S. Humphrey, Jr., and J. Terzian, who have been directing the system and logical design effort on this project have made numerous suggestions and contributed heavily towards the original concept for this entire system. Mr. G. Rocheleau was instrumental in the detail design of the central processor and the design of duplexing capability in MOBIDIC B.

REFERENCES

- 1 Terzian, J. "MOBIDIC Computer Series" THE SYLVANIA TECHNOLOGIST.
Vol. XII, No. 3 July, 1959, pp. 58-64.

2 Terzian, J. "System Organization of MOBIDIC A" presented at the
1957 WESCON Convention, August 20, 1957, San Fransisco,
California.

TABLE I
MOBIDIC B INSTRUCTION REPERTOIRE

<u>OP CODE</u>	<u>ABER.</u>	<u>TIME (us)</u>	<u>TYPE</u>			<u>INSTR. DESCRIPTION</u>
			<u>DIRECT</u>	<u>SUB</u>	<u>SPECIAL</u>	
00	HLT	24	D			HALT
01	RPT	-		R		REPEAT
02	LGM	38	D			LOGICAL MULTIPLY
03	LGA	38	D			LOGICAL ADD
04	LGN	36	D			LOGICAL NEGATION
05	SEN	28	D			SENSE
06	SNS	28	D			SENSE AND SET
07	SNR	28	D			SENSE AND RESET
10	CLA	36	D			CLEAR AND ADD
11	CAM	36	D			CLEAR AND ADD MAGNITUDE
12	ADD	44	D			ADD
13	ADM	44	D			ADD MAGNITUDE
14	CLS	36	D			CLEAR AND SUBTRACT
15	CSM	36	D			CLEAR AND SUBTRACT MAGNITUDE
16	SUB	44	D			SUBTRACT
17	SBM	44	D			SUBTRACT MAGNITUDE
20	MLY	88-774	D			MULTIPLY
21	MLR	88-786	D			MULTIPLY AND BOUND
22	DVD	-		R		DIVIDE
23	DVL	-		R		DIVIDE LONG
24	ADB	-		R		ADD BETA
25	SBB	-		R		SUBTRACT BETA
26	BSPL	-			S	MOB "B" SPECIAL

TABLE I

MOBIDIC B INSTRUCTION REPERTOIRE (Cont'd)

<u>OP CODE</u>	<u>ABER.</u>	<u>TIME (μs)</u>	<u>TYPE</u>			<u>INSTR. DESCRIPTION</u>
			<u>DIRECT</u>	<u>SUB</u>	<u>SPECIAL</u>	
27	BSPL	-			S	MOBIDIC B SPECIAL
30	SHL	30-66	D			SHIFT LEFT
31	SLL	46-118	D			SHIFT LEFT LONG
32	SHR	30-66	D			SHIFT RIGHT
33	SRL	-		R		SHIFT RIGHT LONG
34	CYS	30-66	D			CYCLE SHORT
35	CYL	-		R		CYCLE LONG
36	SLA	30-66	D			SHIFT LEFT A REGISTER
37	NRM	-		R		NORMALIZE
40	TRU	28	D			TRANSFER UNCONDITIONAL
41	TRL	-		R		TRANSFER & LOAD PCS.
42	TRS	-		R		TRANSFER TO PCS.
43	TRX	-		R		TRANSFER ON INDEX
44	TRP	26	D			TRANSFER ON POSITIVE
45	TRZ	26	D			TRANSFER ON ZERO
46	TRN	26	D			TRANSFER ON NEGATIVE
47	TRC	-		R		COMPARE
50	STR	34	D			STORE
51	LOD	36	D			LOAD
52	MOV	36	D			MOVE
53	LDX	-		R		LOAD INDEX
54	RPA	-		R		REPLACE ADDRESS
55	MSK	-		R		MASK

TABLE I

MOBIDIC B INSTRUCTION REPERTOIRE (Cont'd)

<u>OP CODE</u>	<u>ABBR.</u>	<u>TIME (μs)</u>	<u>DIRECT</u>	<u>TYPE</u> <u>SUB</u>	<u>SPECIAL</u>	<u>INSTR. DESCRIPTION</u>
56	BSPL	-			S	MOBIDIC B SPECIAL
57	TRY	38	D			TRANSFER ON INDEX B
60	BSPL	-			S	MOBIDIC B SPECIAL
61	BSPL	-			S	MOBIDIC B SPECIAL
62	BSPL	-			S	MOBIDIC B SPECIAL
63	BSPL	-			S	MOBIDIC B SPECIAL
64	BSPL	-			S	MOBIDIC B SPECIAL
65	BSPL	-			S	MOBIDIC B SPECIAL
66	SKP	30	D			SKIP
67	BSP	30	D			BACK SPACE
70	RAN	30	D			READ ALPHANUMERIC
71	RRV	30	D			READ REVERSE
72	ROK	30	D			READ OCTAL
73	SCH	30	D			SEARCH
74	WAN	30	D			WRITE ALPHANUMERIC
75	WWA	30	D			REWRITE ALPHANUMERIC
76	WOK	30	D			WRITE OCTAL
77	RWD	30	D			REWIND

File Copy

FIG. 1. MOBIDIC B WORD FORMAT

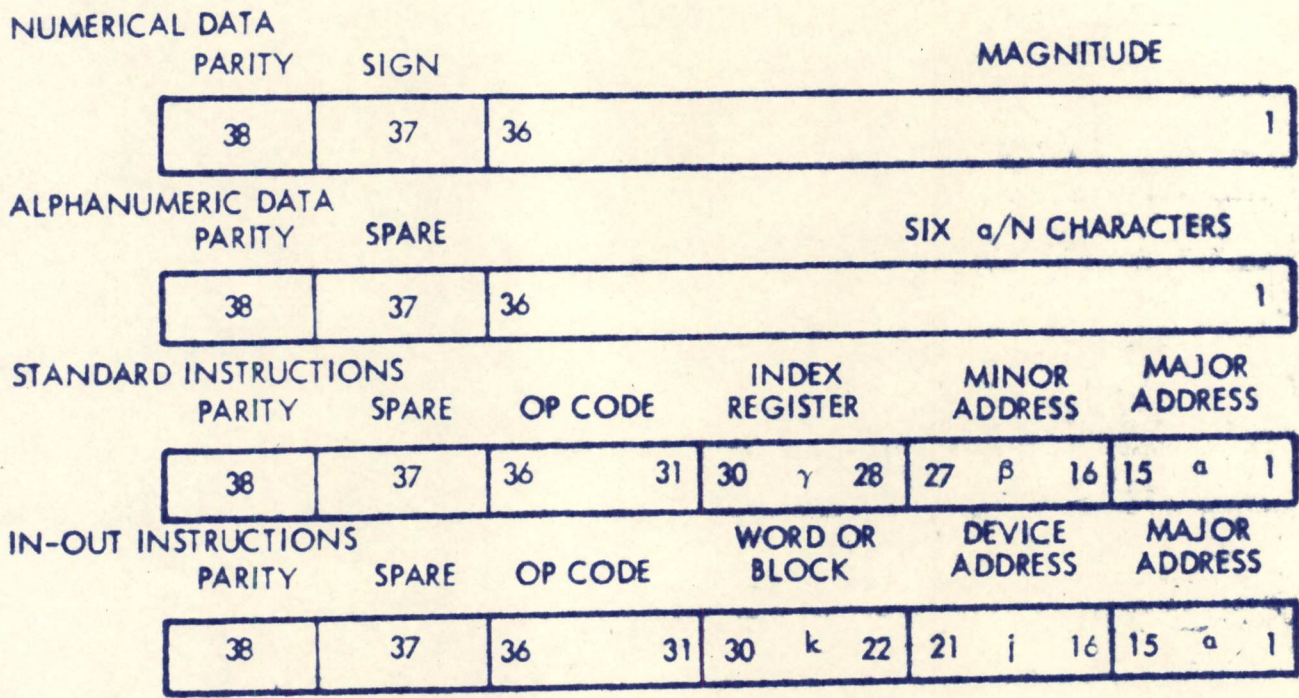


FIGURE 2. MOBIDIC B SYSTEM BLOCK DIAGRAM

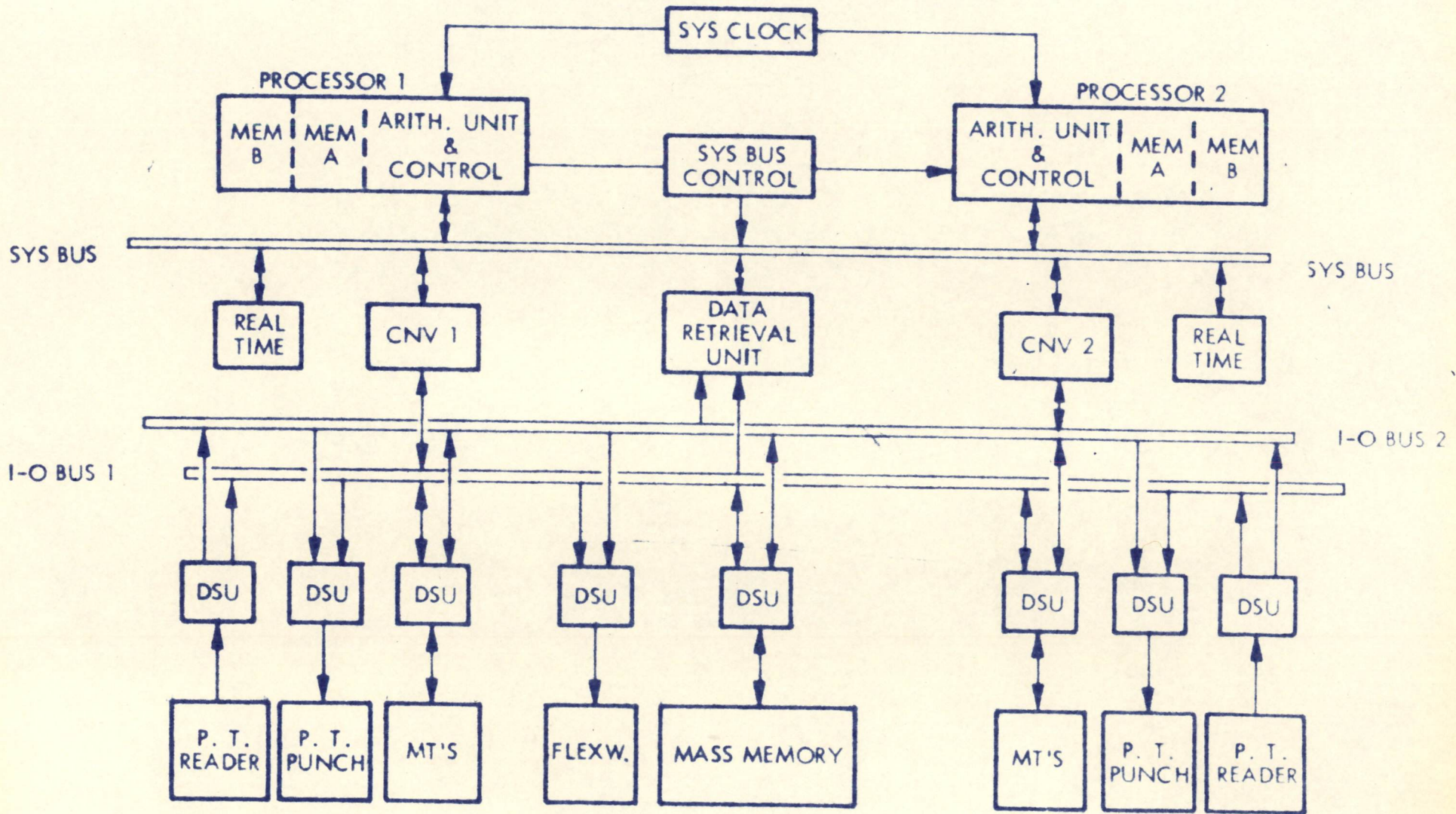
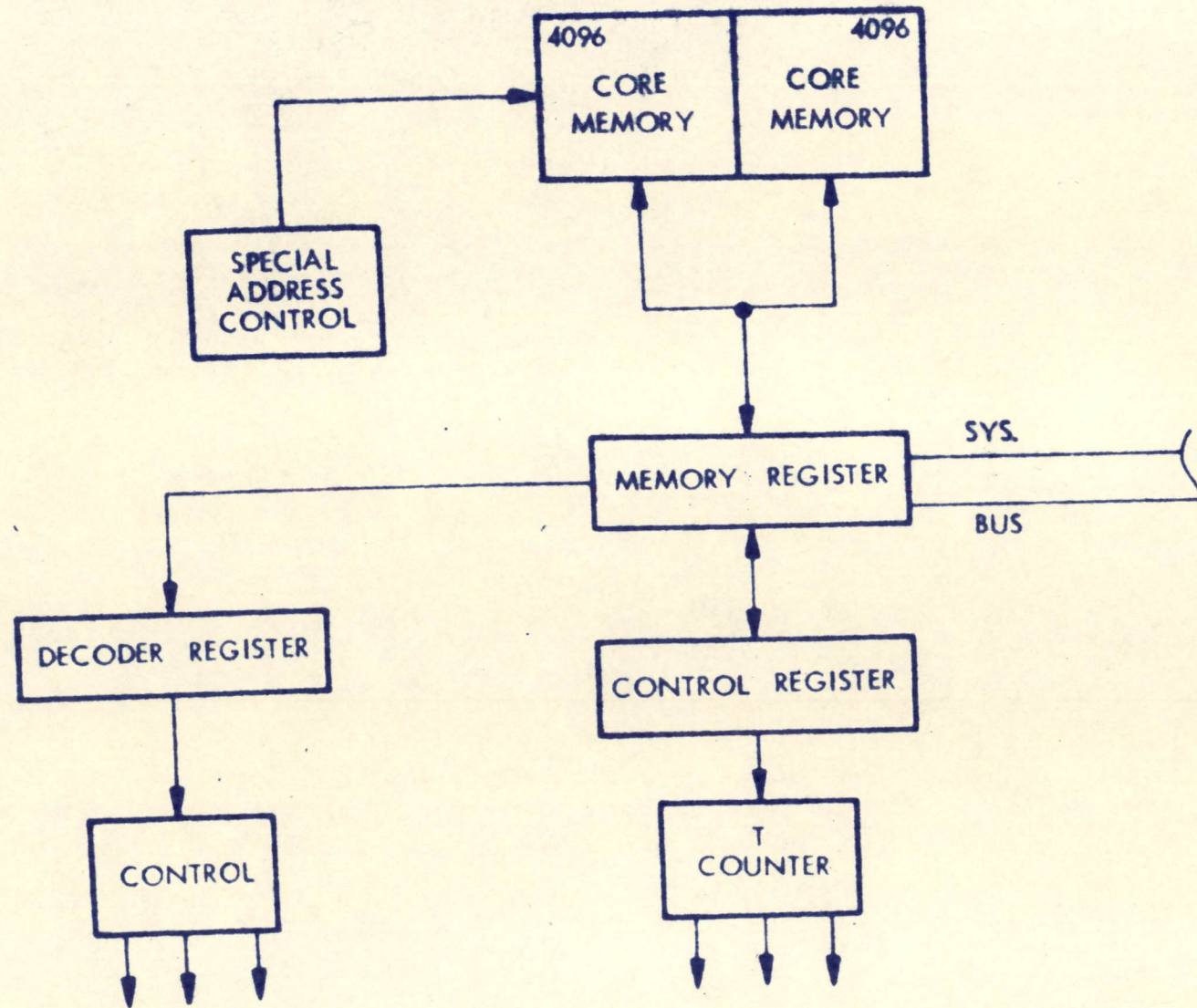


FIGURE 3. MOBIDIC B PROCESSOR



SYLVANIA *Electronic Systems*

A DIVISION OF SYLVANIA ELECTRIC PRODUCTS INC.



DATA SYSTEMS OPERATIONS

189 B Street, Needham 94, Mass.

Hillcrest 4-3940

November 13, 1959

Mr. Harlan E. Anderson
EJCC Publications Committee
Digital Equipment Corporation
Maynard, Massachusetts

Dear Mr. Anderson:

Enclosed please find four copies of the manuscript for the EJCC paper and the biography of the author. The drawings will be hand-carried to your office on Monday, November 16, as per your telephone conversation with Bob Graham of our Technical Services group.

The slides to be used during the talk will be 35 mm.

Yours very truly,

SYLVANIA ELECTRIC PRODUCTS INC.
COMPUTER ADVANCED DEVELOPMENT DEPT.

A. H. Ashley
A. H. Ashley

dlf

Enclosures

cc: E. Cohler
L. Rooney

TEMPERATURE COMPENSATION FOR A CORE MEMORY

Albert H. Ashley
Edmund U. Cohler
Watts S. Humphrey, Jr.

Data Processing Laboratory
Sylvania Electric Products Inc.
Needham 94, Massachusetts

ABSTRACT

For fixed installation, it is often possible to control the temperature of ferrite core memories within narrow limits. However, in a small mobile computer designed to operate over world-wide conditions, this control is not feasible because of the added weight, volume, and cost encountered. A memory designed for such application has been temperature compensated by the use of temperature sensitive components in the current sources to the x-y drivers and in the power supplies for the Z drivers. In addition, core derived strobing has provided peaking time compensation for the sense amplifiers as changes in transistor characteristics delay or advance drive current. This compensation permits operation of an 8192 word 38 bit transistorized memory running at an 8 microsecond cycle time in an ambient environment which may vary between -30°C and $+55^{\circ}\text{C}$.

INTRODUCTION

Most computers use some form of temperature control to maintain the operating temperature of the ferrite cores within very close limits. This precaution is required because of the sensitivity of the ferrite material to ambient temperature variations. When the environmental temperature goes up, the coercive force will go down and the material then loses some of its squareness, consequently becoming more disturb-sensitive as shown in Figure 1. Therefore, if the drive current is

held constant while the temperature rises, the cores switch faster, giving greater amplitude to the output, and the cores are more sensitive to disturbance by the half-selecting drive pulses. The reverse effect is observed as environmental temperature is lowered. Below 0°C, the drive which has proven satisfactory at 25°C will produce a ONE about half as great as previously observed at normal room temperature. Under this condition, there would not be an output from a conventional sense amplifier. Moreover, a fixed strobe would miss the peak signal-to-noise time if the switching characteristics were changed by such an amount.

TEMPERATURE CONTROL

The general solution to the temperature problem has been to control the temperature within the memory enclosure within a few degrees Centigrade. While at first this appears to be a simple solution to the problem, it has proven unsatisfactory over a large temperature range. To maintain the temperature at 95°C above the ambient (say at 65°C in a -30°C ambient) it will be required to install a rather large insulating oven complete with blowers and high wattage heaters and provisions for creating turbulence for proper mixing. When operating in conjunction with accurate thermostatic equipment, it will suffer from the inherent disadvantages of all mechanical components. The reliability from such components will result in degrees of magnitude lower than that of the memory or the accompanying solid-state circuitry. Moreover, the cost of a good air thermostat is considerably greater than that of the few electrical components required to do the job.

Temperature Compensating the Drive-Currents

Another alternative to control of the temperature of the memory cores is control of the drive-current amplitude. If the drive current is varied

with temperature so that half selected cores are not disturbed but the fully selected cores are properly driven for full switching output, satisfactory operation is obtained. From the memory cores of the type used in Sylvania's MOBIDIC it was determined that drive-current compensation aimed solely at maintaining constant switching time resulted in a considerably lower output signal amplitude at the low end of the temperature range. Since the cores are less disturb-sensitive at lower temperatures, it is feasible to compensate for constant output-voltage-amplitude. The constant amplitude compensation below 20°C minimizes sense amplifier problems since no variation in strobe level is required. The overall compensation curve, shown in Fig. 2, results in a constant core output below 20°C and constant switching time above that temperature.

X-Y Drive-Current Sources

The drive currents for the X-Y coordinates originate from high impedance current sources, each source consisting of a power transistor connected in the common base configuration. The high impedance is required to maintain good current regulation under varying load conditions. The circuit for the current source is shown in Figure 3. Current is supplied to the emitter of the current-source transistor by a source consisting of a reference voltage V_{REF} applied across a variable resistance network. The resistance is partially variable to compensate for initial differences in transistor parameters. The reference voltage V_{REF} is common to all current sources in the X-Y circuitry.

Choice of Compensation Technique

The output current may be varied with temperature by one of two methods. Either the external emitter resistors may have a positive temperature coefficient, or the Voltage reference may have a negative temperature coefficient. There are no positive temperature coefficient resistors available with sufficient power

capability for the first method. Even if they were available they would not be very practical to use because of drive current tolerances. The latter method is considerably better since it employs only one temperature-sensitive network per memory and uses readily available negative temperature coefficient elements. Moreover, the common compensation assures that all drivers vary equally, thus minimizing drive current tolerance problems.

Voltage Reference Design

Thermistors (negative temperature coefficient resistors) have a relatively low dissipation coefficient (watts/°C rise). It is, therefore, advisable to maintain a negligible dissipation within them in order to have their resistance remain a function of true ambient temperature without side effects from internal heating. Consequently, the thermistor network is buffered by a power amplifier with a unity voltage gain and a high input impedance, allowing the use of current as low as 5 ma in the network.

The thermistor network and buffer amplifier are shown in Figure 4. Notice that the V_{REF} is derived from the -20 volt supply as indicated in Figure 3. This means that variation in the -20 volt supply will not affect the current source accuracy. Two thermistors are necessary to provide the proper compensation characteristics over the entire temperature range. A constant current of 5 ma through the 1k precision resistor provides a constant drop of five volts. The thermistor network with 5 ma through it will add a voltage drop of 1.0 volt at +25°C, 2.2 volts at -30°C, and 0.52 volts at +55°C. The overall curve between temperature end points is nearly linear, (note that Figure 2 is on an expanded scale) in great part due to the constant five volts superimposed on the temperature-sensitive voltage.

Results Follow Theoretical Curve

The oscillographs in Figure 5 were taken from an experimental system

consisting of transistor core drivers and a memory core. The drive current varies through the desired pattern, although the compensation at this time was slightly less than that shown in Fig. 2. Even so, the ONE at -30°C is within 10% of the ONE's at the other temperatures. A subsequent slight revision of the thermistor network was made to increase drive at the lower temperatures, resulting in a higher output at the low end without affecting the drive at other temperatures. The final compensation characteristic is as shown in Fig. 2.

Compensating the Z-Drive Current

The Z-drivers do not employ high impedance transistorized current sources such as those used for the X-Y drivers, because of less stringent current tolerances. The current for each Z winding is determined by the power supply voltage across a fixed resistance in series with the winding, as shown in Fig. 6. In order to vary the current with temperature, either the resistance or the total voltage across the resistance must be varied. The first method was impractical, because resistors with large positive temperature coefficients are not available. To vary the whole supply-voltage with temperature is not practical due to complications in the power supply design. To overcome these problems, one end of the current determining resistance R_1 was connected to a fixed close-tolerance power supply (used elsewhere in the memory); and the emitter of the output transistor, (Q3, Fig. 6), was returned to a temperature sensitive supply V_{TEMP} . This supply was designed to vary from ± 0.5 volts at $\pm 55^{\circ}\text{C}$ to ± 5.0 volts at -30°C . Because the maximum voltage swings up to 5 volts, considerably less power is involved in the temperature sensitive control than if the entire 20 volt supply were to vary from -20 volts to -25 volts, and the percentage variation is less critical.

Temperature Sensitive Emitter Supply

Because the thermistor network used in the X-Y coordinate has a quasi-linear-resistance-temperature-characteristic, an identical network was used

to derive the emitter supply for the Z-drivers as shown in the circuit of Fig. 7. A stage of inversion with a voltage gain of 2 is interposed between the thermistor network and the power amplifier in order to provide the proper phase and amplitude to the variation. The output is clamped to ground on the low end by the transistor and to 45 volts (44.4 plus diode drop) on the high end by a diode. This clamping insures against overvoltages on the Z-driver transistors. When none of the Z-drivers are in operation, 7.5 amps. are conducted to ground by the output transistor (Q5 of Fig. 7) of the V_{TEMP} circuit. When Z-drivers are being pulsed, the V_{TEMP} output conducts the difference between 7.5 amps and the average Z-driver drain.

Compensation with Core-Derived Strobe

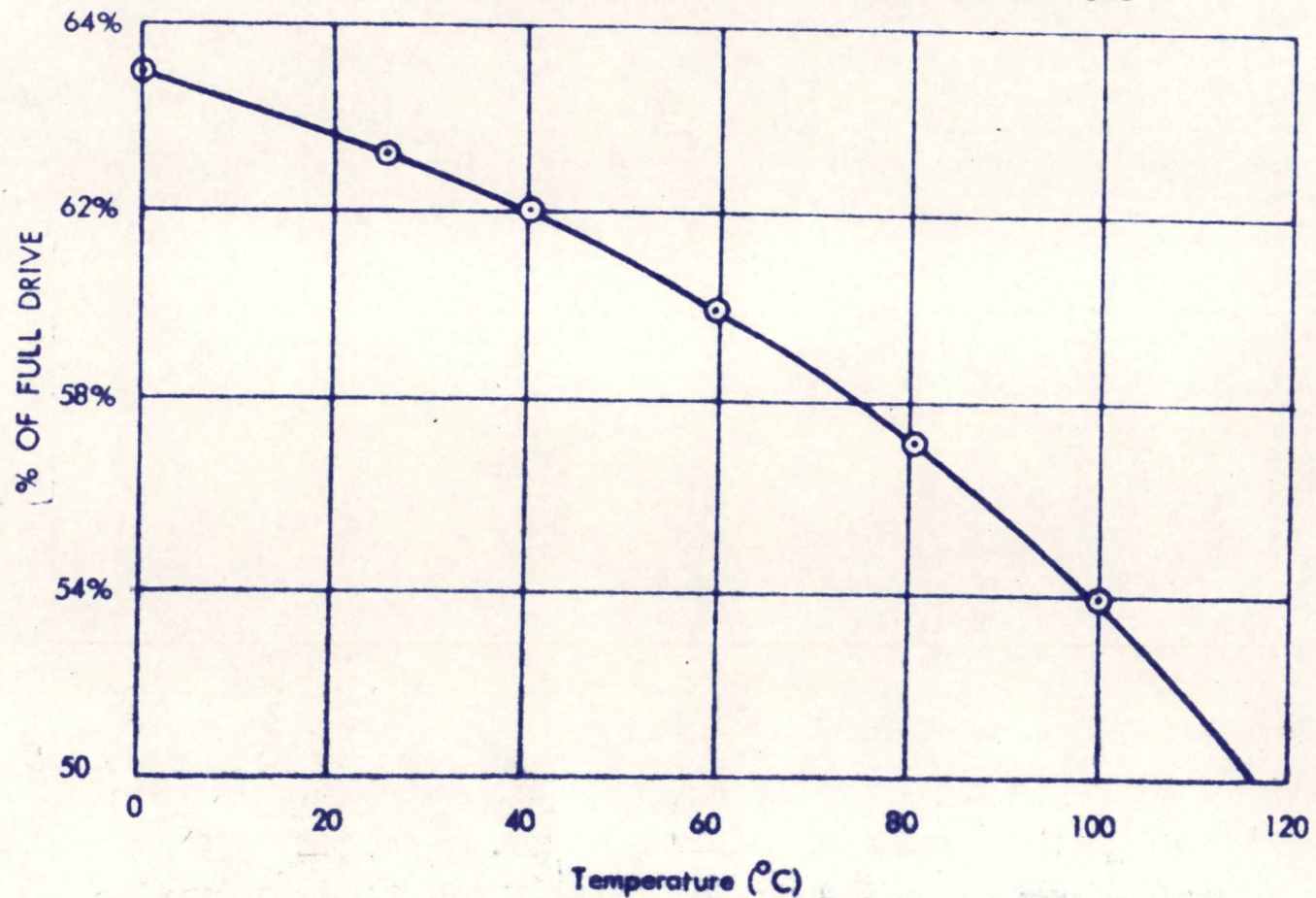
The compensation of the drive currents still allows some variation in the peaking time of the core output, even for perfect amplitude compensation. Moreover, temperature affects the drive circuit delay. These effects can be observed in Figure 5. The compensation for this variation is made completely and simply by the use of a core-derived strobe pulse. The time-discriminating-strobe is derived from a standard core receiving the same current as the selected cores in the memory. That core is essentially wired to receive a full read and full write from the x and y drivers selected to supply the rest of the memory. The output of the core is therefore a standard ONE produced at the same time as all other ONE's being read out. This output is then properly shaped and suitably delayed to supply a strobe pulse for the memory sense amplifiers. The block-schematic in Figure 8 gives the outline of the method employed. Experimental results in a full memory show that variations in the sense amplifier output of 0.8 microseconds may occur and are compensated by the core-derived strobing even when ZERO's, are larger than ONE's (under virgin-checkerboard test).

CONCLUSION

The operational limits of the memory were extended by the combination of core-derived strobing and temperature compensated drive currents, as shown in Figure 9, which is a "shmoo" plot of temperature versus discrimination level limits of the sense amplifiers. The smaller area with cross-hatching shows the limit with core derived strobe but without temperature compensation; the larger encompassing area shows extension of those limits by the temperature compensation. With neither core-derived strobing nor temperature compensation, the limits are reached at 10°C and 45°C .

Because the MOBIDIC computer in which this memory is being used is intended for battlefield operations, provision is made for retention of the information in the memory even after the computer is shut down. Conceivably the information could be read into the memory at one temperature and later read out at another. Tests performed on the cores showed no measureable difference between ONE's read out at a given temperature regardless of the core's temperature when the information was stored in it. Thus retention of memory is possible even if the machine is shut down and restarted in a new and widely different environment.

FIGURE 1. PERCENTAGE OF FULL DRIVE WHICH WILL DISTURB STORED INFORMATION VS. CORE TEMPERATURE



Percentage of Full Drive Which Will Disturb Stored Information vs. Core Temperature

FIGURE 2. CORE CURRENT REQUIRED VS. TEMPERATURE

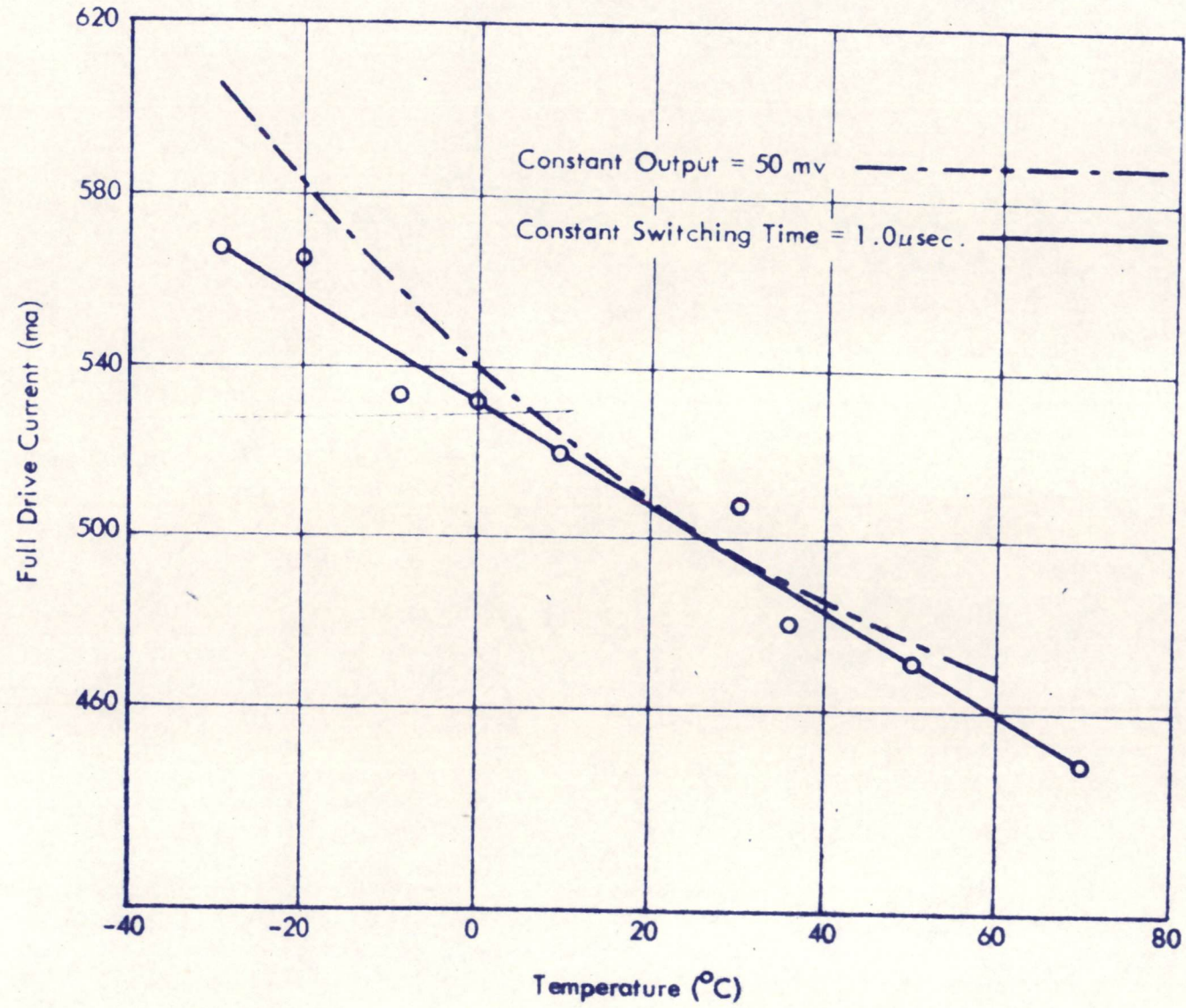


FIGURE 3. CURRENT SOURCE

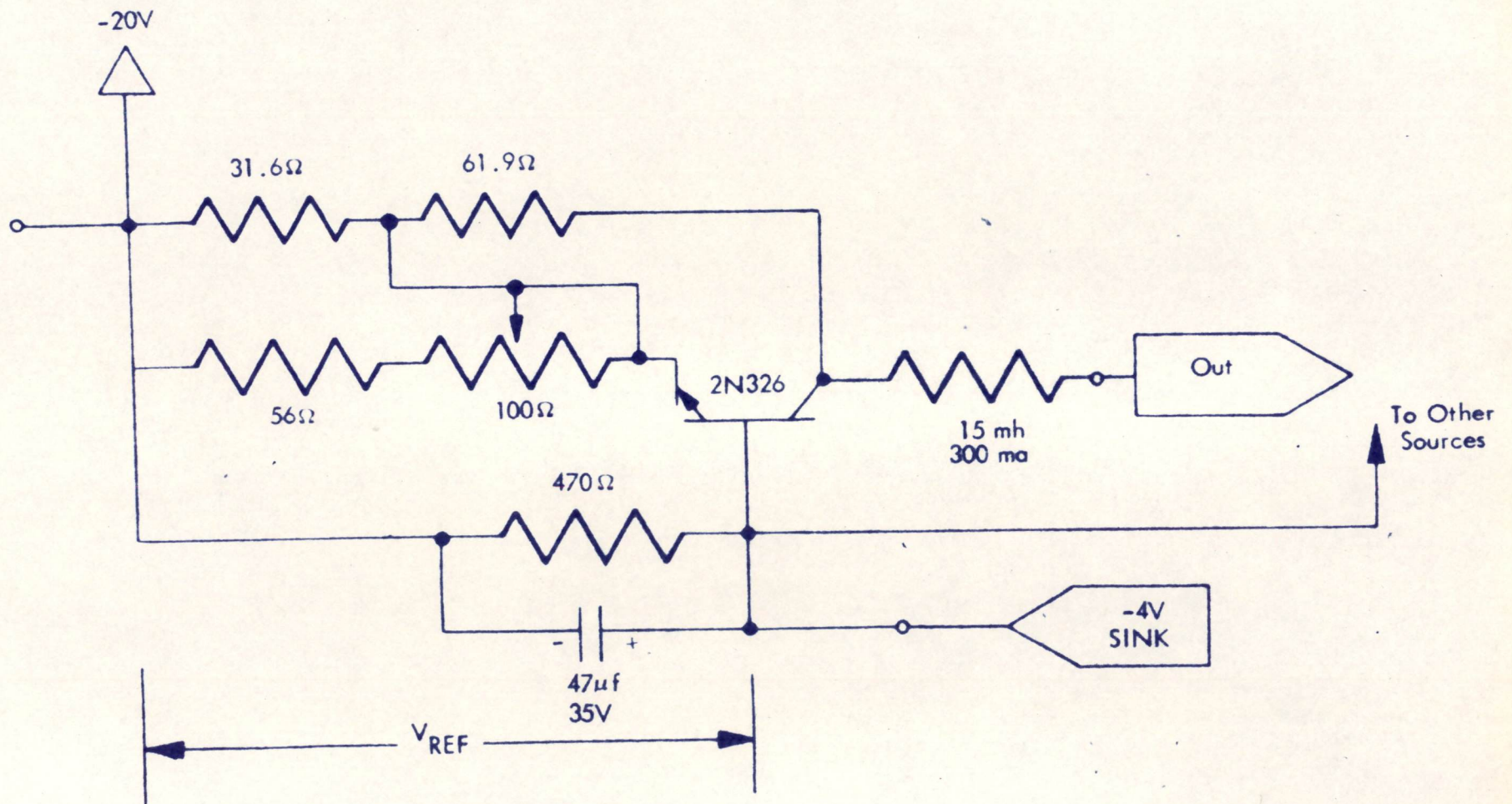


FIGURE 4. TEMPERATURE COMPENSATED VOLTAGE REFERENCE (X-Y)

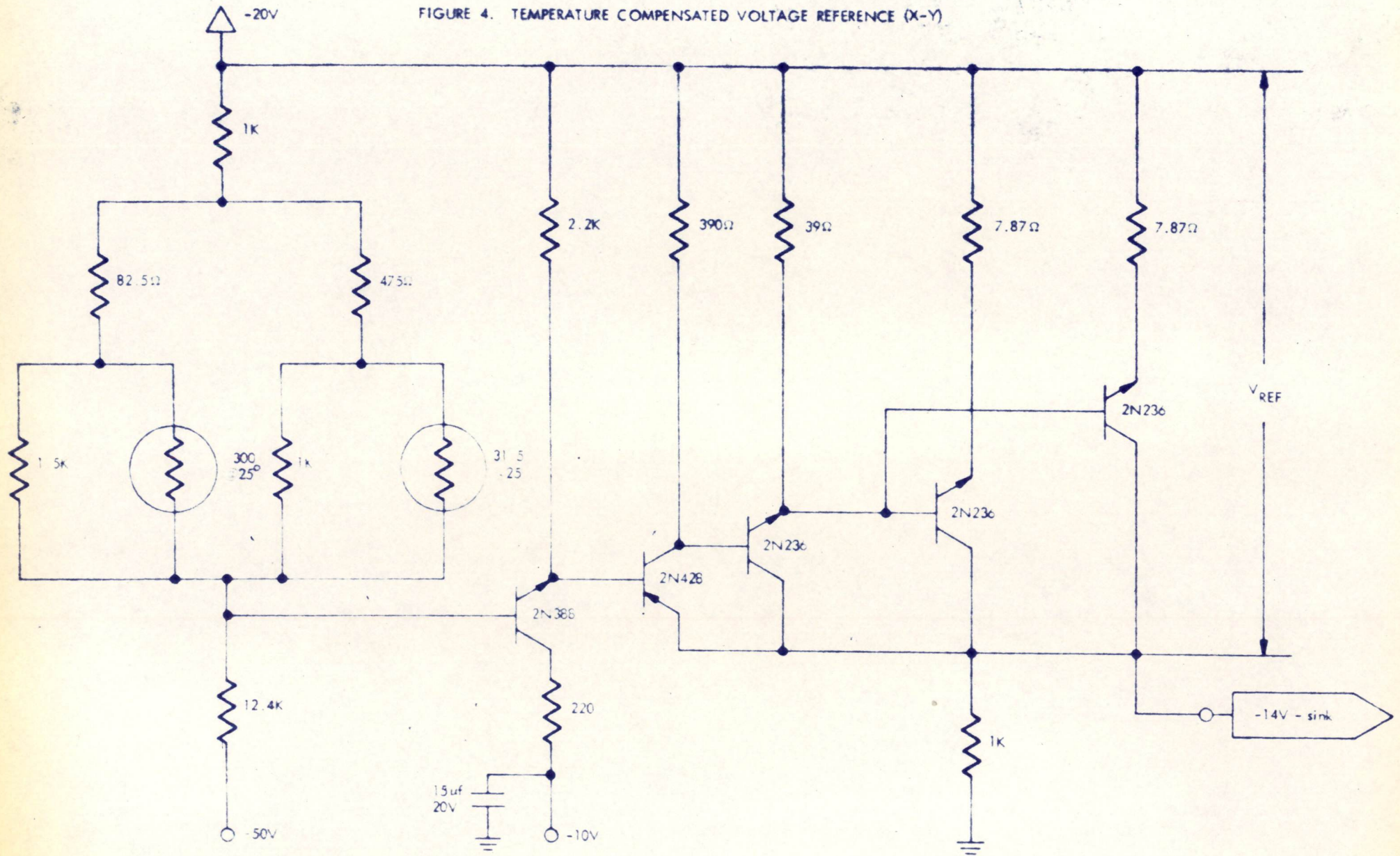


FIGURE 5. TEMPERATURE COMPENSATED OUTPUTS

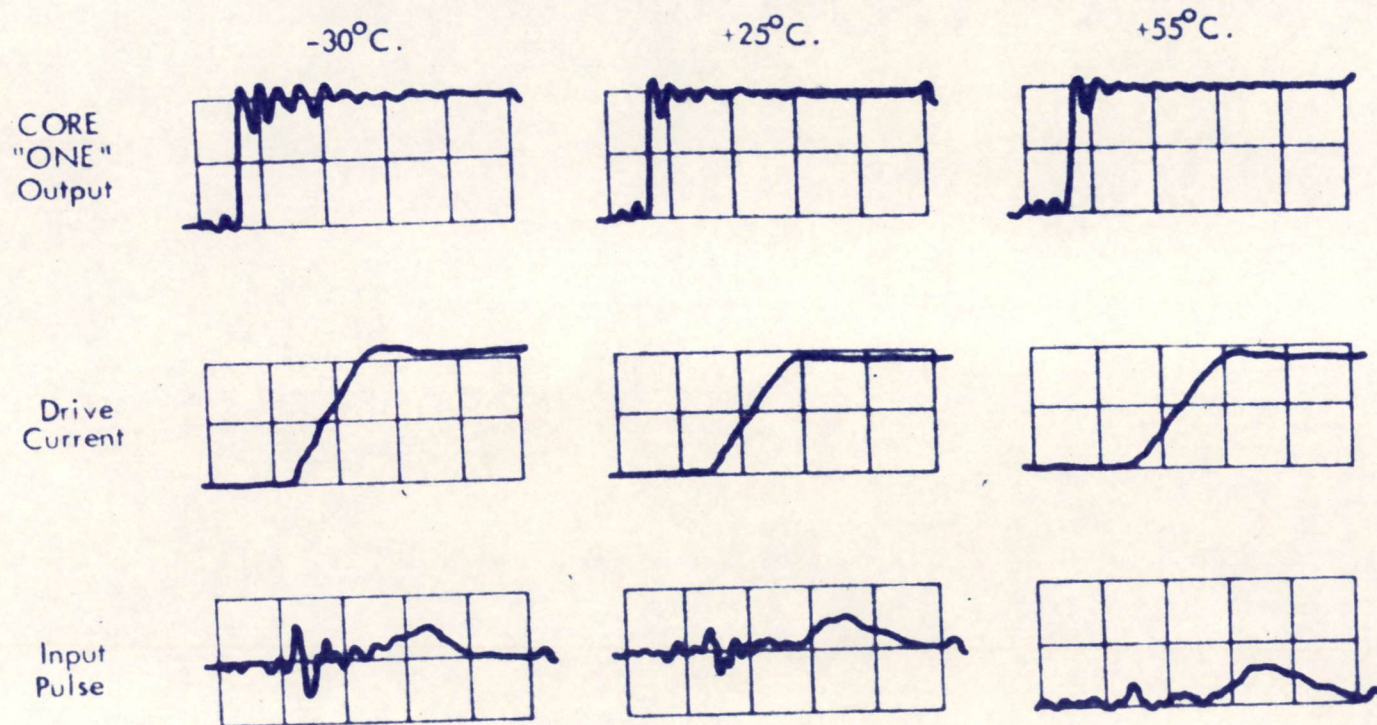


FIGURE 6. CIRCUIT FOR Z-DRIVER

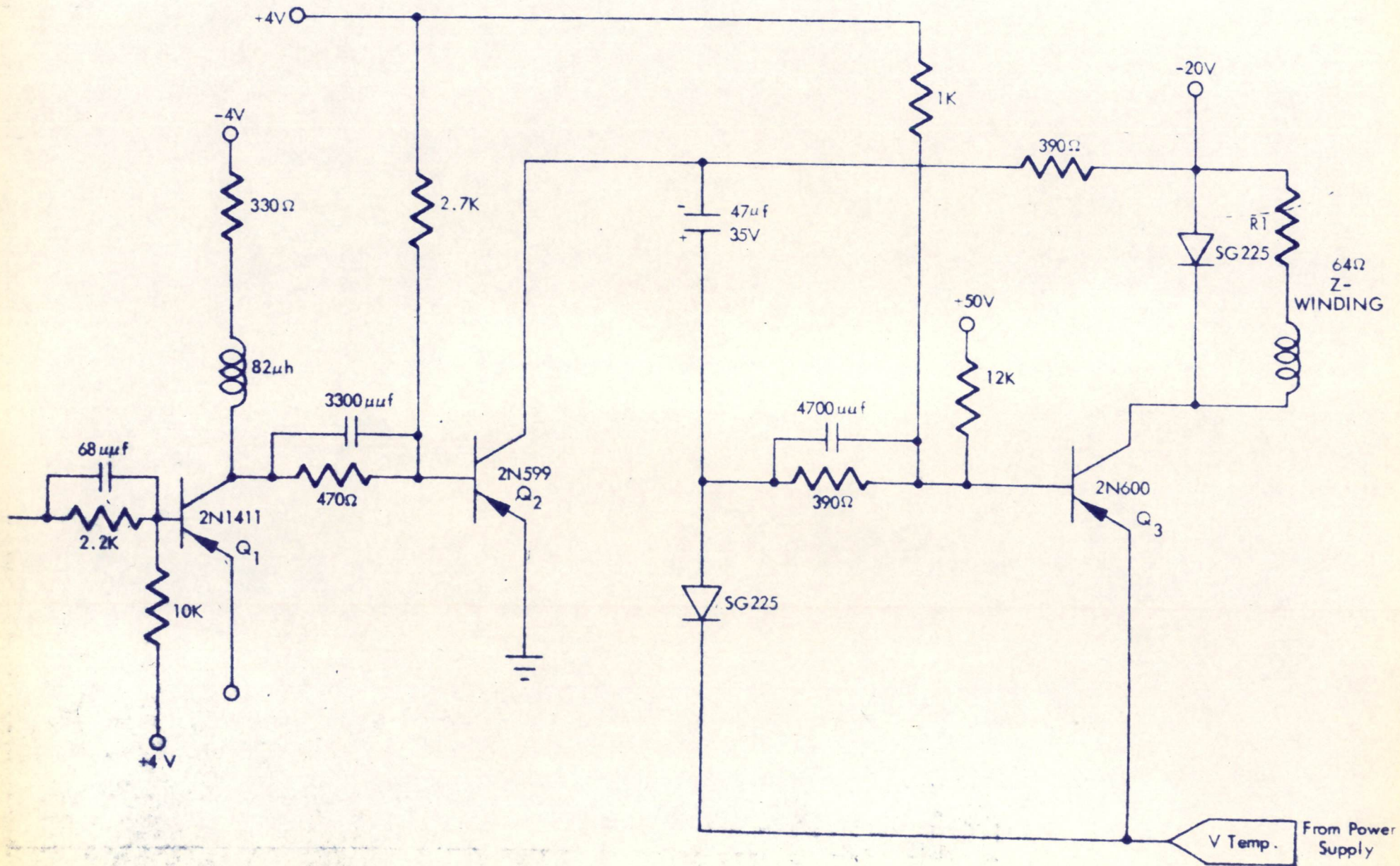


FIGURE 8. CORE-DERIVED STROBE SYSTEM

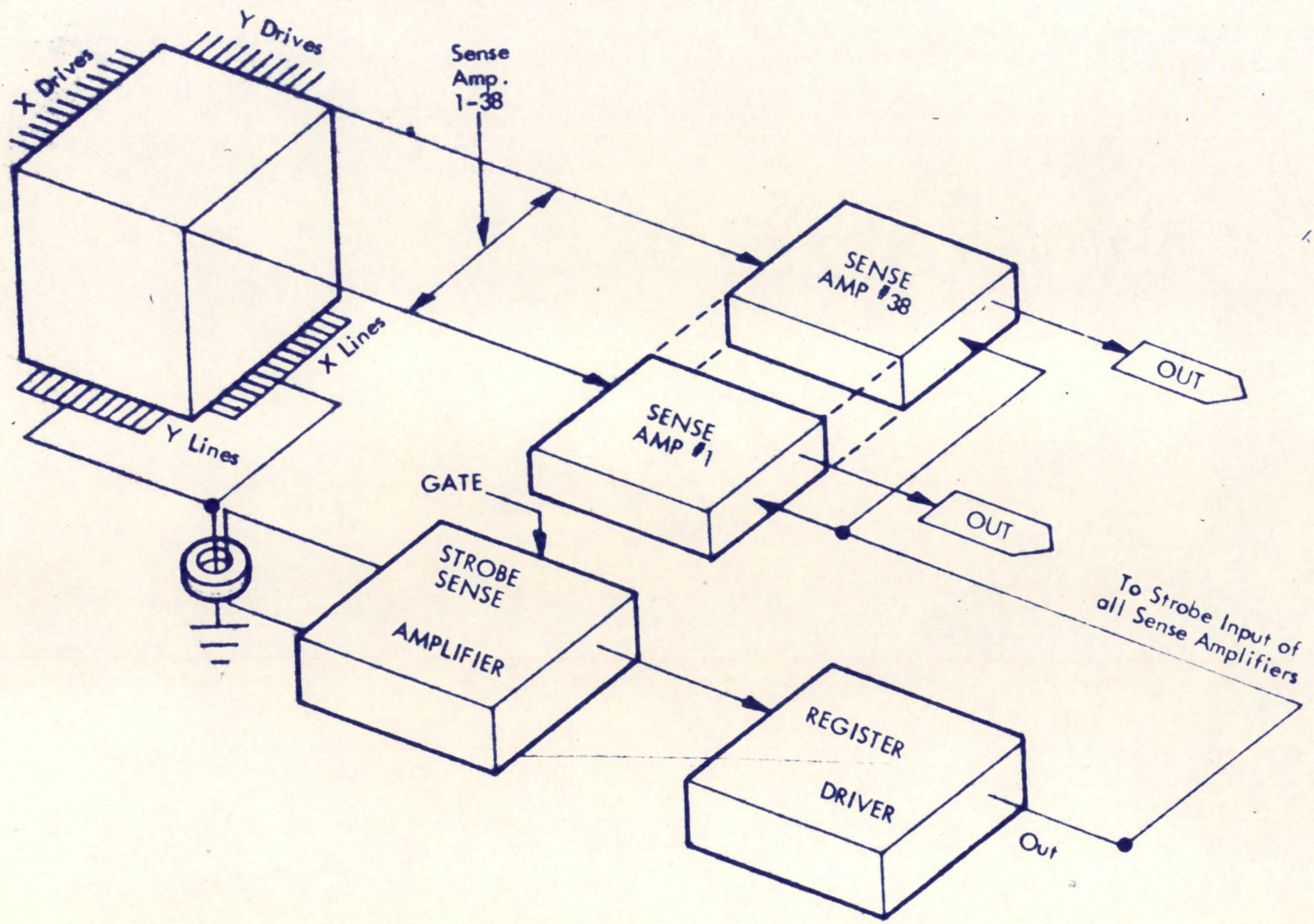


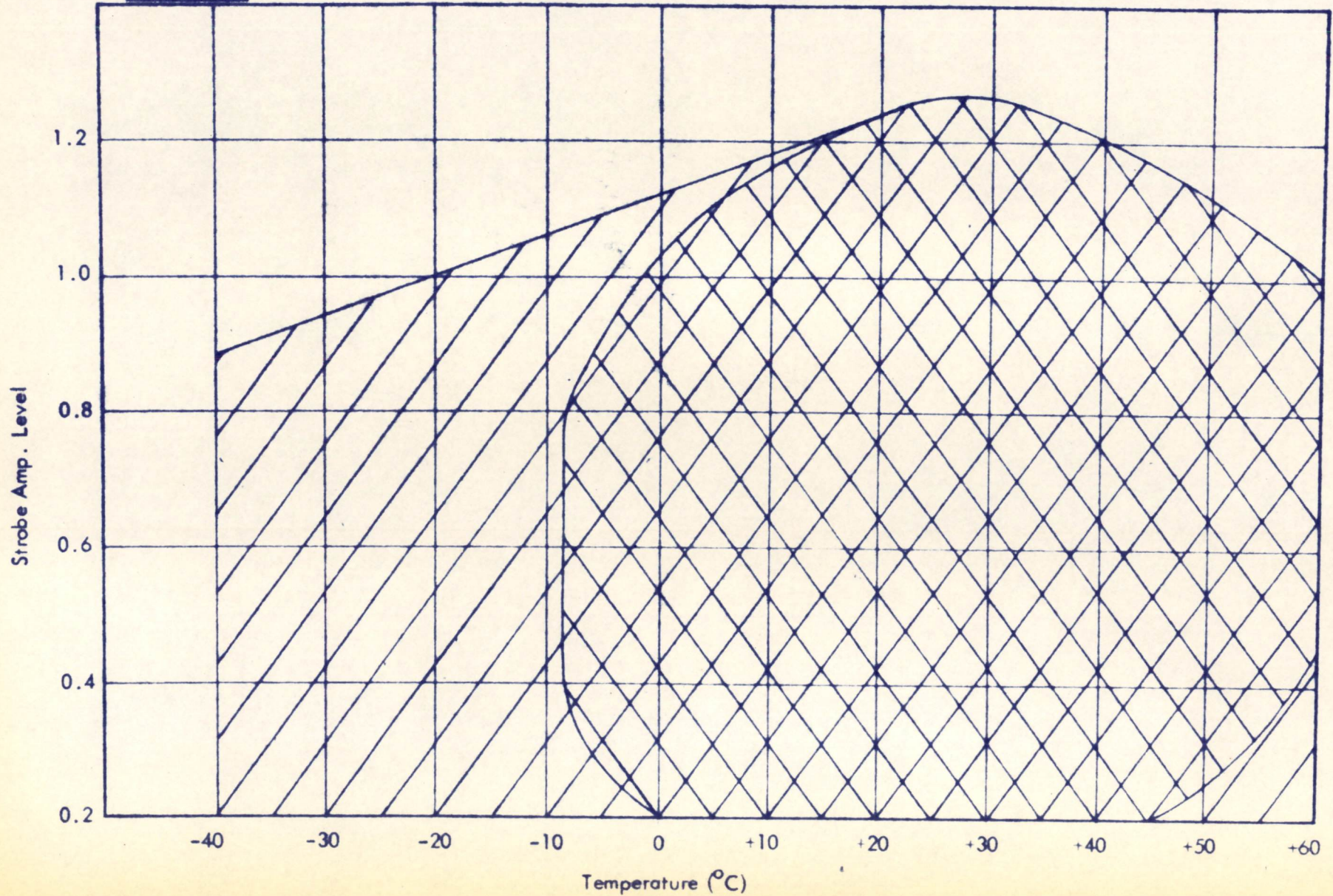
FIGURE 9. OPERATION SHMOO FOR SENSE AMPLIFIERS



WITH CORE-DERIVED STROBE & NO COMPENSATION



" " " & TEMPERATURE COMPENSATION



U. S. DEPARTMENT OF COMMERCE
NATIONAL BUREAU OF STANDARDS

ADDRESS REPLY TO
NATIONAL BUREAU OF STANDARDS
WASHINGTON 25, D. C.

IN YOUR REPLY
REFER TO FILE NO.

November 12, 1959

12.4

*

Mr. Harlan E. Anderson, Chairman
1959 EJCC Publication Committee
Digital Equipment Corporation
Maynard, Massachusetts

Dear Sir:

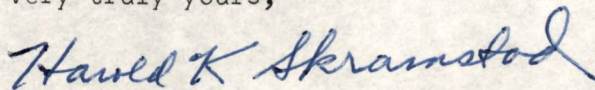
Enclosed are four copies of the manuscript of the paper
"A Combined Analog-Digital Differential Analyzer" for inclusion in
the Proceedings of the 1959 Eastern Joint Computer Conference.

Standard 3-1/4"x 4" lantern slides will be used in the
presentation.

A biographical sketch is also enclosed.

If you wish any further information, I will be pleased to
furnish it.

Very truly yours,



Harold K. Skramstad
Assistant Chief
Data Processing Systems Division

Enclosures.

File copy

A COMBINED ANALOG-DIGITAL DIFFERENTIAL ANALYZER

By
Harold K. Skramstad
National Bureau of Standards
Washington, D.C.

ABSTRACT

An analog-digital differential analyzer has been designed which combines the analog advantages of high speed and continuous representation of variables with the digital capability of high precision and dynamic range. It is based on representing dependent variables by two quantities, a digital number representing the more significant part, and an electrical voltage representing the less significant part. As in the electronic analog computer, time is the independent variable.

The design of components required to build a computer of this combined type, such as integrators and multipliers, are given, and examples of how the solution of a few elementary differential equations would be carried out are presented.

File Copy

A COMBINED ANALOG-DIGITAL DIFFERENTIAL ANALYZER

By

Harold K. Skramstad
National Bureau of Standards
Washington, D.C.

Introduction

The electronic analog computer, although very useful in solving many problems, and particularly useful in solving dynamic problems described by differential equations, suffers from limitations of accuracy and dynamic range. The digital differential analyzer, although capable of providing any required degree of accuracy or dynamic range, is slow in operation and subject to possible instability of solution due to quantization and the use of finite difference calculus in integration. By combining analog and digital techniques, it is possible to combine the analog advantages of high speed and continuous representation of variables with the digital capability of high accuracy and high dynamic range.

Dependent variables in such a combined system are represented by two quantities, a digital number, representing the more significant part, and an electrical voltage representing the less significant part. As in the electronic analog computer, the independent variable is always time. Let us consider what form some of the required computer components, such as integrators and multipliers, would take in such a combined system.

Integrator

Assume we wish to obtain the following:

$$y = y_0 + \frac{1}{T} \int_0^t x dt \quad (1)$$

where x and y are functions of the time, and T is the "time constant" of the integration. As in the digital differential analyzer, it is necessary that the problem be scaled so that the maximum value of all dependent variables will not exceed unity. Let each of the two dependent variables x and y consist of a digital part and an analog part, denoted by the subscripts D and A , respectively. Thus, we have:

$$x = x_D + x_A \quad (2)$$

$$y = y_D + y_A \quad (3)$$

$$y = y_{0D} + y_{0A} + \frac{1}{T} \int_0^t (x_D + x_A) dt \quad (4)$$

Let us assume time to be divided into discrete equal intervals of duration Δt , and that the digital parts of x and y can change only at times which are integral multiples of Δt . We may then write for the value of y at a time t somewhere in the n th interval:

$$y = y_{0D} + y_{0A} + \frac{1}{T} \left[\sum_{i=1}^{n-1} (x_D)_i \Delta t + (x_D)_n \left\{ t - (n-1)\Delta t \right\} + \int_0^t x_A dt \right] \quad (5)$$

where $(x_D)_i$ is the value of x_D during the i th interval Δt . Figure 1 shows a curve of x as an arbitrary function of t . The area under this curve from

$t = 0$ to any arbitrary t would equal y in equation (5). In this special case, the first two terms on the right of equation (5) (y_{oD} and y_{oA}) are zero. The first term in the bracketed expression, represented by area 1, is the integral of the digital part of x up to the time $(n-1)\Delta t$. The second term in the bracket expression, represented by area 2, is the integral of the digital part of x between $(n-1)\Delta t$ and t . The third term, represented by area 3, is the integral of the analog part of x from $t = 0$ to t .

Figure 2 is a block diagram of an integrator unit. It contains an input digital register x_D , a digital register R , two digital-to-analog converters, an analog integrator, a special resettable analog integrator, an analog summer, and a comparator unit. The register y_D shown on the far right of the figure is the input register of the next component to which this unit might be connected in solving a problem. E is the analog reference voltage supplied to the digital-to-analog converters, and a is the digital value of the reference voltage E , chosen for any given problem so as to provide the desired compromise between speed of solution and precision, subject to the limitation that $|dx/dt|_{\max}$ should not exceed $a/\Delta t$. The number of digits required in the x_D and R registers will depend upon the minimum value of a for which provision is to be made; the minimum value of a will be one in the least significant digit of the x_D register.

At the beginning of each Δt period, the values x_D and R are sampled and converted to analog voltages which are held constant during the period, unaffected by future changes in x_D or R which occur during the period. The value of x_D is then algebraically added to the R register. The

voltage V_1 , which represents that portion of the prior summation of $(x_D)_1 \Delta t$ which is of analog magnitude is given during the n th interval Δt by:

$$V_1 = -ER_n \quad (6)$$

The voltage V_2 , which provides integration of the current x_D value within the n th interval Δt , and which is reset to zero at the end of this interval, is given by:

$$V_2 = -\frac{E(x_D)_n \{t - (n-1)\Delta t\}}{\Delta t} \quad (7)$$

The voltage V_3 , which represents the purely analog integration of the continuously varying analog part of x , is given by:

$$V_3 = \frac{-Ey_{oA}}{\alpha} - \frac{\alpha}{\Delta t} \int_0^t \frac{Ex_A}{\alpha} dt \quad (8)$$

These three voltages are added in the analog summer to give voltage V .

The analog part of the output of the integrator is equal to:

$$\frac{Ey_A}{\alpha} = V = - (V_1 + V_2 + V_3) \quad (9)$$

If, at any time during a period Δt the voltage V at the output of the analog summer exceeds a predetermined upper threshold, this is sensed by the comparator, and during the next Δt interval, immediately following the addition of x_D to R , unity is subtracted from the R register and the number α is added to the input register of the following unit (y_D in Figure 2). Conversely, if the voltage V falls below a predetermined threshold, unity is added to the R register and the number α is subtracted from the input register of the following unit.

The time constant T of this integrator unit is equal to $\frac{\Delta t}{\alpha}$, as can be seen from the following. Assume that from time 0 up to a time t during the n th interval Δt , the comparator has initiated N subtractions of unit from the R register, and the addition of $N\alpha$ to the y_D register. The contents of the R register at this time is:

$$R = \sum_{i=1}^{n-1} (x_D)_i - N \quad (10)$$

and the value of y_D is given by

$$y_D = y_{OD} + N\alpha \quad (11)$$

Substituting equations (6), (7), (8) and (10) into (9) and solving for y_A , we obtain:

$$y_A = \alpha \sum_{i=1}^{n-1} (x_D)_i - N\alpha + \frac{\alpha}{\Delta t} (x_D)_n \{t - (n-1)\Delta t\} \quad (12)$$

$$+ y_{OA} + \frac{\alpha}{\Delta t} \int_0^t x_A dt$$

Adding equations (11) and (12), we have

$$y = y_D + y_A = y_{OD} + y_{OA} + \frac{\alpha}{\Delta t} \left[\sum_{i=1}^{n-1} (x_D)_i \Delta t + (x_D)_n \{t - (n-1)\Delta t\} + \int_0^t x_A dt \right] \quad (13)$$

Equation (13) is seen to be identical to equation (5) if $T = \frac{\Delta t}{\alpha}$.

Multiplier

Let us now investigate the form taken by a combined analog-digital multiplier. Suppose we wish to obtain the product $z = xy$. Assuming, as before, that each variable consists of a digital part and an analog part, we have:

$$z_D + z_A = x_D y_D + x_A y_D + x_D y_A + z_A y_A \quad (14)$$

where the subscripts D and A signify the digital and analog parts, respectively. Assume, as before, that time is divided into equal intervals of duration Δt , and that the digital parts x_D and y_D can change only at times which are integral multiples of Δt . Figure 3 is a block diagram of a multiplier unit. It has three digital registers for x_D , y_D , and R, three digital-to-analog converters, an analog summer, an analog multiplier, and a comparator unit. As before, E is the analog reference voltage and α is the digital value of the reference voltage E, chosen for any given problem so as to provide the desired compromise between speed and accuracy, subject to the condition that neither $|\frac{dx}{dt}|_{\max}$ nor $|\frac{dy}{dt}|_{\max}$ should exceed $\alpha/\Delta t$.

At the beginning of each period Δt , the values of x_D , y_D , and R are sampled and converted to voltages which are held constant during the period. If, during the period, x_D receives an increment (or decrement) α from another unit, y_D is added to (or subtracted from) R; and if y_D receives an increment (or decrement) α from another unit, x_D is added to (or subtracted from) R. If both x_D and y_D change during Δt , the additions to R must either be done serially, using the new x_D or y_D obtained after each addition to R, for the next addition to R, or some other system must be used to obtain a true digital

product $x_D y_D$. The quantity $x_D y_D$ will contain twice as many digits as x_D or y_D ; the more significant half will be of digital magnitude, and appear in z_D , the input register of the following unit; and the less significant half will be of analog magnitude, and remain in the R register. The reference voltage E is applied to the digital-to-analog converter connected to register R, producing an output voltage $V_1 = ER$; the input voltage $y_A E/\alpha$ is applied to the converter connected to the register x_D producing an output voltage $V_2 = E \frac{x_D y_A}{\alpha}$, and the input voltage $E x_A/\alpha$ is applied to the converter connected to register y_D producing an output voltage $V_3 = \frac{E y_D x_A}{\alpha}$. An analog multiplier is connected to the two analog inputs $E y_A/\alpha$ and $E x_A/\alpha$. Its output, attenuated by α , produces a voltage $V_4 = \frac{E x_A y_A}{\alpha}$. An analog summer sums the voltages V_1, V_2, V_3 , and V_4 to produce a voltage V equal to $-\frac{E z_A}{\alpha}$.

During the next Δt after the voltage V exceeds (or falls below) predetermined threshold voltages, unity is subtracted from (or added to) the R register and the number α is added to (or subtracted from) the input register of the following unit (z_D in Figure 4).

It should be noted that for small values of α the analog multiplier may be omitted, producing a maximum error of α^2 . For values of α less than the precision of the analog components, say .001 or less, this error is negligible.

If one of the factors to be multiplied is a constant, the equipment required is simplified, since only one digital register needs to be capable of accepting increments, and the R register receives additions from only one other register. If the factor is a purely digital number, one of the digital-to-analog converters and the analog multiplier may be omitted.

Summing

Summing may most easily be done by permitting each integrator or multiplier unit to accept digital increments and analog voltages from several units. For example, in the integrators of Figure 2, if the $\frac{1}{a}$ increments from a number of other units are connected to its x_D register, and if the sum of the increments put out by these units is Na during any period, the increment in x_D would equal Na . The analog outputs from the other units would each be connected to an input summing resistor in the analog integrator.

In the case of the multiplier unit, if the $\frac{1}{a}$ increments from a number of other units are connected to its x-register, and if the sum of the increments put out by these units is Na , y_D would be summed into the R register N times. The analog outputs from the other units would be connected to inputs of an analog summer whose output would form the analog input $\frac{x_A E}{a}$ to the multiplier.

Solution of Simple Differential Equations

Examples of the operation of this proposed combined system can be seen from following in detail how some simple differential equations would be solved.

Let us consider first the differential equation

$$\dot{x} = -x \tag{15}$$

Figure 4 shows a block diagram of how a single integrator unit with output fed back into its input would solve this equation. The voltages V_1 , V_2 , V_3 , and V are those defined in equation (6) to (9).

Differentiating equations (6) to (9), we obtain, since $\dot{V}_1 = 0$

$$\dot{V} = \frac{Ex_D}{\Delta t} - \dot{V}_3 \quad (16)$$

From the interconnections of Figure 4, the following expression must hold:

$$\dot{V}_3 = \frac{\alpha}{\Delta t} V \quad (17)$$

V will then be given by the following differential equations:

$$\dot{V} + \frac{\alpha}{\Delta t} V = \frac{Ex_D}{\Delta t} \quad (18)$$

Subject to the initial conditions that at $t = 0$, $x_D = x_{OD}$, and $-V = \frac{Ex_{OA}}{\alpha}$

the differential equation will have the following solution:

$$V = \frac{Ex_D}{\alpha} - \frac{E(x_{OD} + x_{OA})}{\alpha} e^{-\frac{\alpha}{\Delta t} t} \quad (19)$$

and x will be given by

$$x = x_D - \frac{\alpha V}{E} = (x_{OD} + x_{OA}) e^{-\frac{\alpha}{\Delta t} t} \quad (20)$$

Figure 5 shows in detail the quantities that would appear in the x_D and R registers, the voltages V_1 , V_2 , V_3 , and V as functions of the time, using the following parameters:

$$E = 100 \text{ volts} \quad \alpha = 0.1 \quad \Delta t = 0.1 \text{ sec}$$

$$x_{OD} = 0.5 \quad x_{OA} = 0$$

The threshold values on V for initiating decrements to the x_D register and subtracting one from the R register are plus and minus 50 volts. In this example, since $\alpha = 0.1$, the precision obtained in solving this equation should be 10 times that which would be obtained in solving this on a purely analog computer with analog components of equivalent precision to those of the combined system.

Another example of the operation of the proposed combined system is the solution of the following simple differential equations:

$$\dot{x} = x \quad (21)$$

$$\dot{x}' = -x' \quad (22)$$

Figure 6 shows a block diagram of how two integrator units would be interconnected to solve these equations. The voltages V_1, V_2, V_3, V are those defined by equations (6) to (9), and occur in the integrator containing x ; the primed voltages are those which occur in the integrator containing x' . Differentiating equations (6) to (9), we have, since $\dot{V}_1 = \ddot{V}_1 = 0$:

$$\dot{V} = \frac{x_D^E}{\Delta t} - \dot{V}_3 \quad (23)$$

$$\dot{V}' = \frac{x_D'^E}{\Delta t} - \dot{V}_3' \quad (24)$$

From the interconnections of Figure 6, the following expressions are seen to hold:

$$\dot{V}_3 = + \frac{\alpha}{\Delta t} V' \quad (25)$$

$$\dot{V}_3' = - \frac{\alpha}{\Delta t} V \quad (26)$$

The quantities V and V' therefore will be given by the following differential equations:

$$\dot{V} = \frac{x_D E}{\Delta t} - \frac{\alpha}{\Delta t} V' \quad (27)$$

$$\dot{V}' = \frac{x_D' E}{\Delta t} + \frac{\alpha}{\Delta t} V \quad (28)$$

Subject to the condition that at $t = 0$, $x_D = x_{OD}$, $x_D' = x_{OD}'$, $V = \frac{x_{OA}' E}{\alpha}$,

$-V' = \frac{x_{OA}' E}{\alpha}$; these differential equations will have the following solutions:

$$V = \frac{E}{\alpha} x_D + \frac{E}{\alpha} (x_{OD}' + x_{OA}') \cos \frac{\alpha t}{\Delta t} + \frac{E}{\alpha} (x_{OD} + x_{OA}) \sin \frac{\alpha t}{\Delta t} \quad (29)$$

$$V' = \frac{E}{\alpha} x_D' - \frac{E}{\alpha} (x_{OD}' + x_{OA}') \sin \frac{\alpha t}{\Delta t} + \frac{E}{\alpha} (x_{OD} + x_{OA}) \cos \frac{\alpha t}{\Delta t} \quad (30)$$

and the quantities x and x' will be given by the following expressions:

$$x = x_D - \frac{\alpha}{E} V' = (x_{OD} + x_{OA}) \cos \frac{\alpha t}{\Delta t} - (x_{OD}' + x_{OA}') \sin \frac{\alpha t}{\Delta t} \quad (31)$$

$$x' = x_D' + \frac{\alpha}{E} V = (x_{OD}' + x_{OA}') \cos \frac{\alpha t}{\Delta t} + (x_{OD} + x_{OA}) \sin \frac{\alpha t}{\Delta t} \quad (32)$$

Figure 7 shows in detail the voltages which would appear as a function of the time when solving the above equation, using the following values of the various parameters:

$$\begin{array}{llll} E = 100 \text{ volts} & \alpha = 0.1 & \Delta t = 0.1 \text{ sec} & \\ x_{OD} = 0.5 & x_{OA} = 0 & x_{OD}' = 0 & x_{OA}' = 0 \end{array}$$

As in the previous example, the threshold values on V and V' for initiating positive or negative increments to the x_D' or the x_D register, are plus and minus 50 volts, respectively. Figure 8 shows the results of combining the digital and analog portions. The broken curve is the digital part only; the smooth curve is the sum of the digital and analog parts. As in the first example, the precision attained should be 10 times that which would be obtained solving these equations on an analog computer with analog components of equivalent precision to those used in the combined system.

Component Requirements

Now a few words on the characteristics needed in the hardware to realize the above components. For a maximum speed-precision product to be obtained, the value of Δt should be as small as possible consistent with hardware limitations. The smaller Δt is made, however, the greater the bandwidth required in the operational amplifiers, since the analog voltages must be capable of a full scale voltage excursion E during the time Δt . The digital-to-analog converters should be capable of holding their output values constant during each period of Δt and equal to its value at the beginning of the period, and then rapidly changing to its new value at the beginning of the next period. In the case of the integrator, the necessary additions of x_D to R , subtractions of $+1$ from R , and incrementing the input registers of following units, and in the case of the multiplier, the additions of x_D and y_D to R , subtractions of $+1$ from R , and incrementing the input registers of following units must all be completed within the period Δt . In the integrator unit, the resettable analog integrator might well consist of two analog integrators with switching between them so that each is used to integrate during alternate Δt periods while the other is being reset.

Discussion and Conclusions

There are a number of problems associated with this combined system that have not yet been investigated. One of these, which this computer has in common with both analog and digital differential analyzers is that of proper scaling so as to prevent overflowing of the digital registers or saturation of the analog integrators. Another is the choice of the threshold voltages for the comparator units. It is possible that the best value for both upper and lower threshold voltages should be zero - requiring a positive or negative digital increment to be sent to the next unit each Δt , depending upon the sign of the voltage V . The problem of scaling and choice of thresholds are interrelated, and it should be possible to exercise some control over the overloading of analog integrators by proper choice of the threshold voltages.

The number of digits to be carried in the digital registers depends, of course, on the minimum value of a for which provision is to be made. In general, the R registers should contain one more binary digit than the other registers to prevent overflow under conditions where the large x_D of the same sign as R is added to R .

For any particular problem, the value of a to be chosen depends upon the particular compromise between precision and speed of solution desired. As a simple illustration, consider integration of the function $x = A \sin \omega t$, and assume Δt equals .001 second, $a = .001$, and A is 1. Since the maximum time rate of change of this function $A\omega$ should not exceed $\frac{a}{\Delta t}$, the highest frequency representable at full-scale amplitude would be $\omega = \frac{a}{A\Delta t} = 1$ radian

per second, and the precision (assuming an analog resolution of .001) would be one part in one million. If we chose $\alpha = .1$, the highest frequency representable at full scale amplitude would be 100 radians per second, and the precision would be one part in ten thousand.

The combined analog-digital differential analyzer of the type described shows promise of overcoming some of the limitations of present designs of differential analyzers which use purely analog or purely digital techniques, and of providing a much greater precision-speed product than is possible with existing digital differential analyzers. If analog components of sufficient bandwidth are available, the precision-speed product of this combined system should be greater than that possible with a parallel digital differential analyzer of equal length digital registers and equal iteration rate, by a factor dependent upon the resolution of the analog components - perhaps a factor of one thousand.

The greatest usefulness of the proposed system is believed to be on problems where the precision required is of the order of 10 to 100 times greater than that obtainable by analog methods, yet requires the real-time speed of analog methods. In this case, integrators and multipliers of the combined system would contain short digital registers and only moderate requirements would be put on the speed of switching circuits and the bandwidth of the analog components.

Work is under way at the National Bureau of Standards to construct breadboards of two integrator and two multiplier units, each capable of receiving inputs from two other units. The digital registers and digital-to-analog converters are being constructed from transistorized digital

building block packages developed at NBS, and the analog components from commercially available operational wide band amplifiers. These units are planned to have 7 bit plus sign input registers, 8 bit plus sign "R" registers, an analog reference voltage of 10 volts, and operate with a Δt of one millisecond.

Acknowledgment

The author wishes to acknowledge the assistance of the Navy Bureau of Aeronautics in supporting the construction of the breadboard units for evaluating the proposed computer system.

Nov. 12, 1959

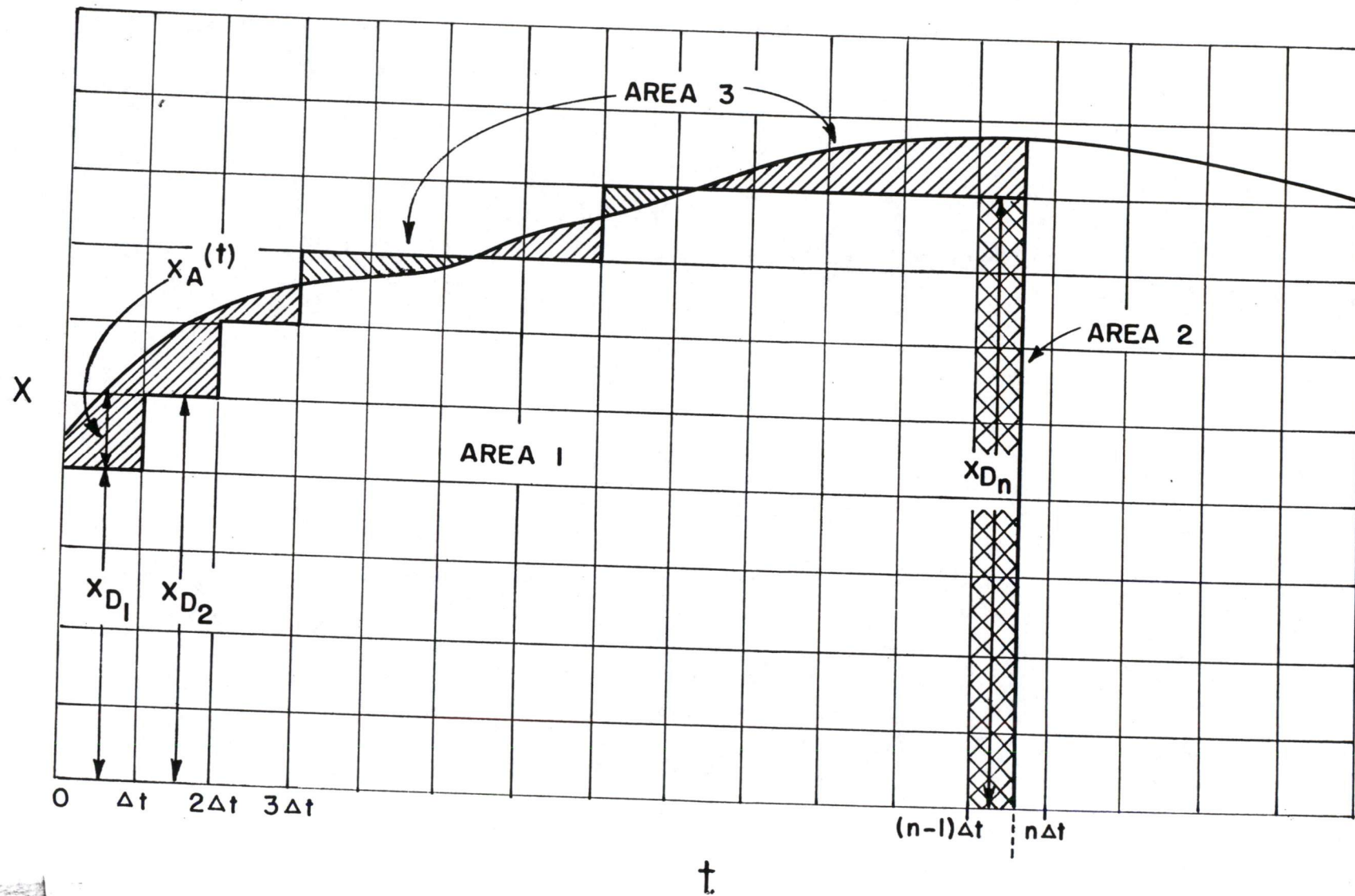


FIGURE 1 DIAGRAM OF INTEGRATION METHOD

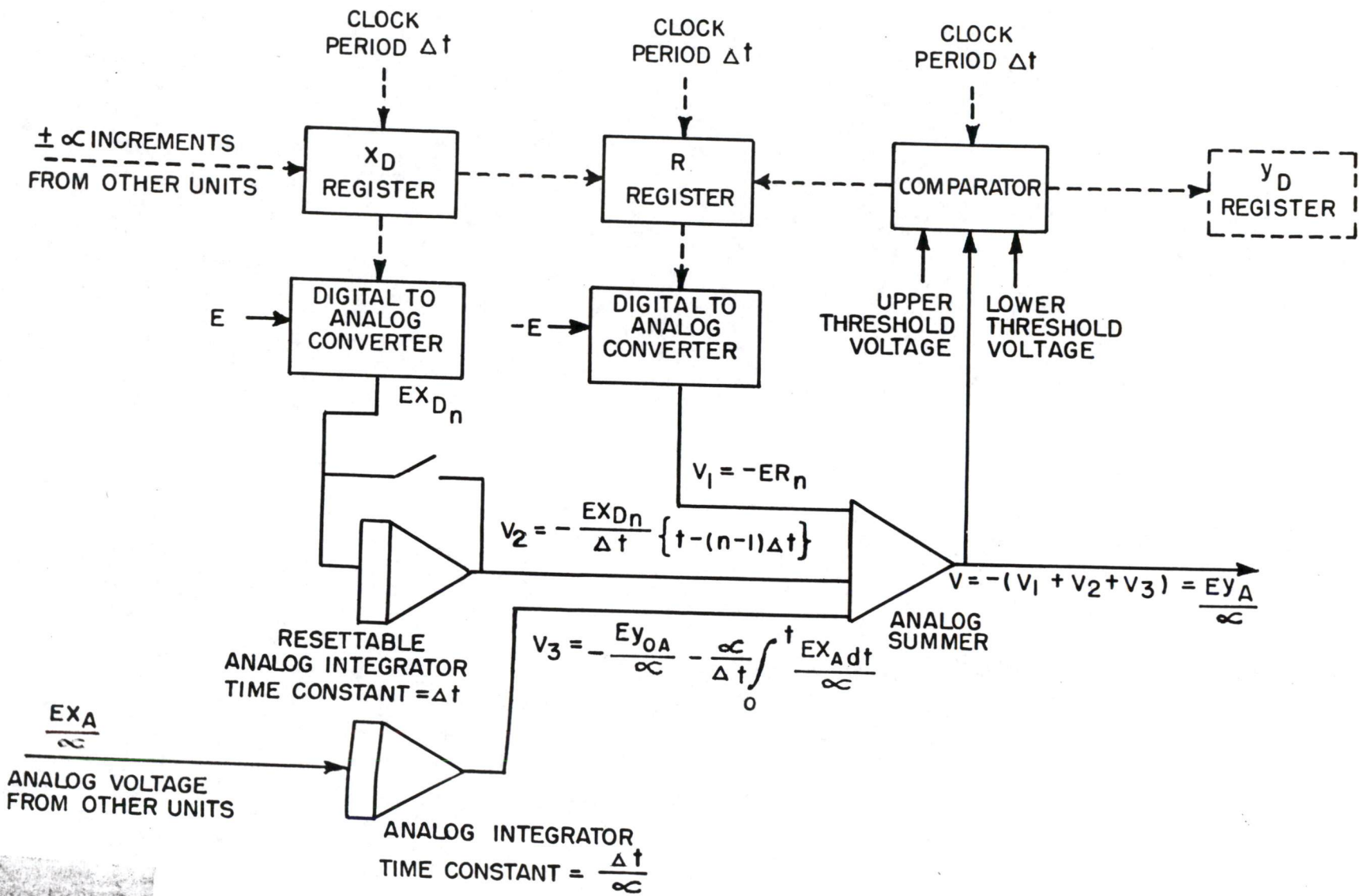


FIGURE 2 INTEGRATOR UNIT

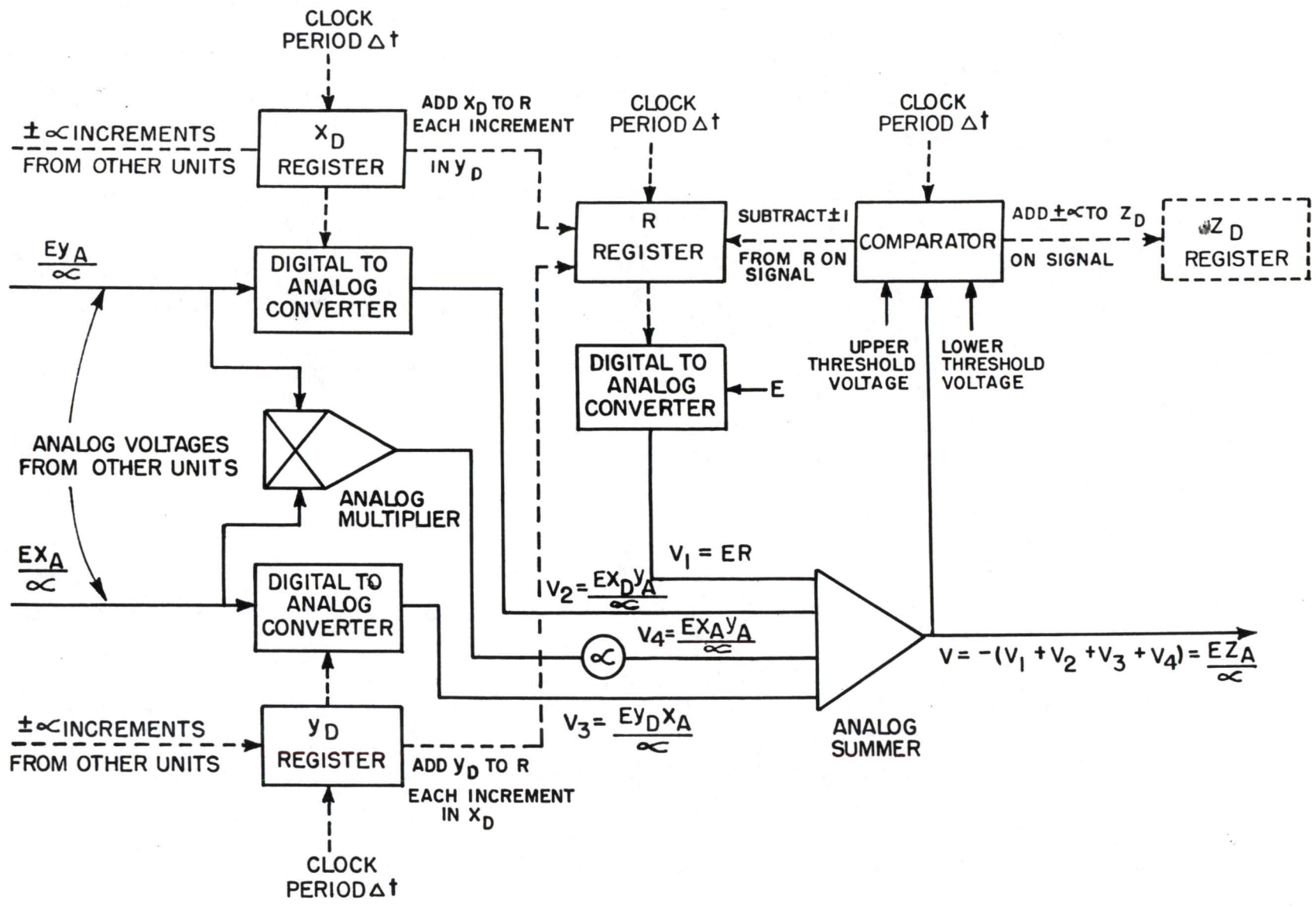


FIGURE 3

MULTIPLIER UNIT

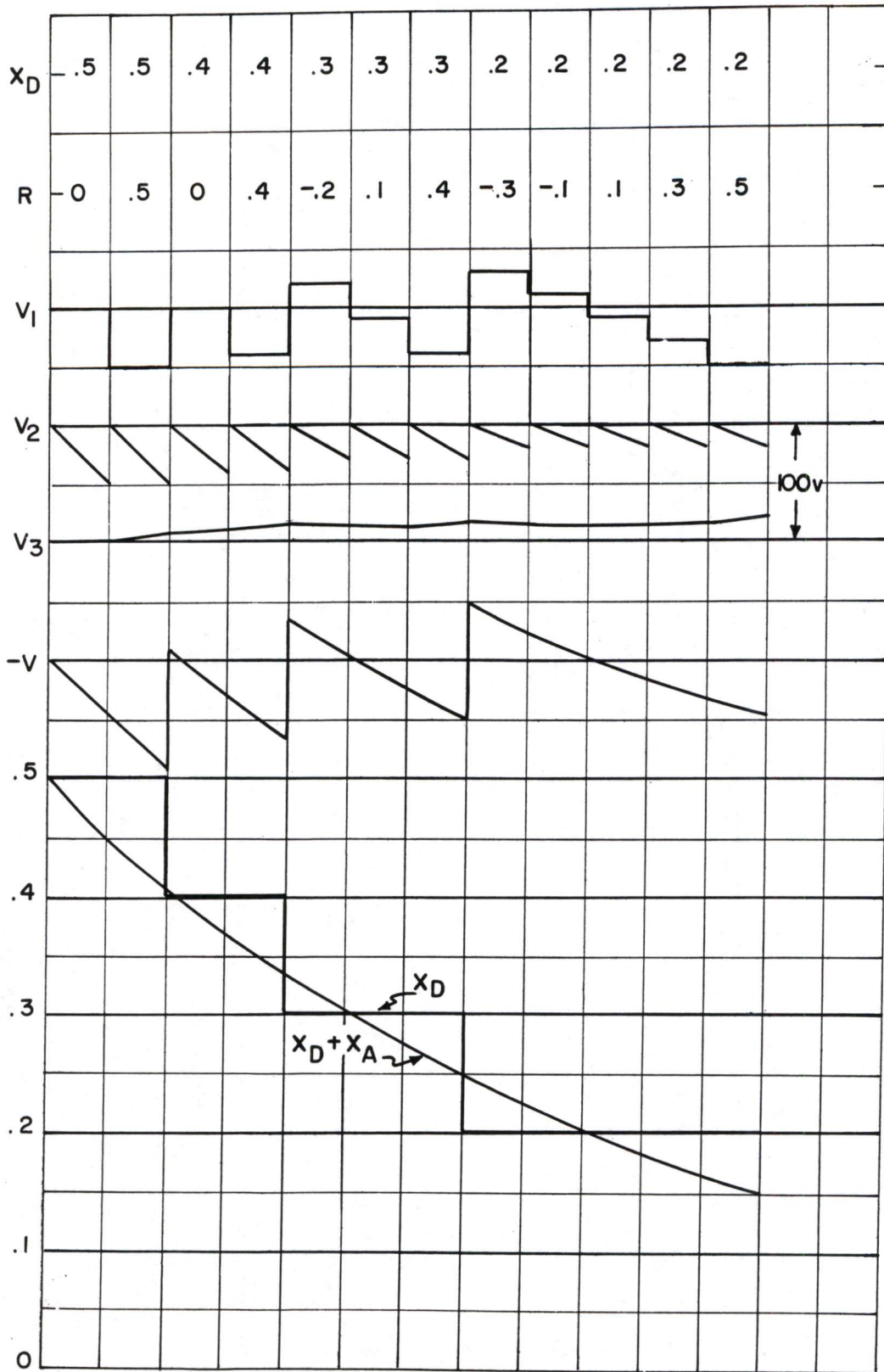


FIGURE 5 TIME HISTORY OF SOLUTION OF $\dot{x} = -x$

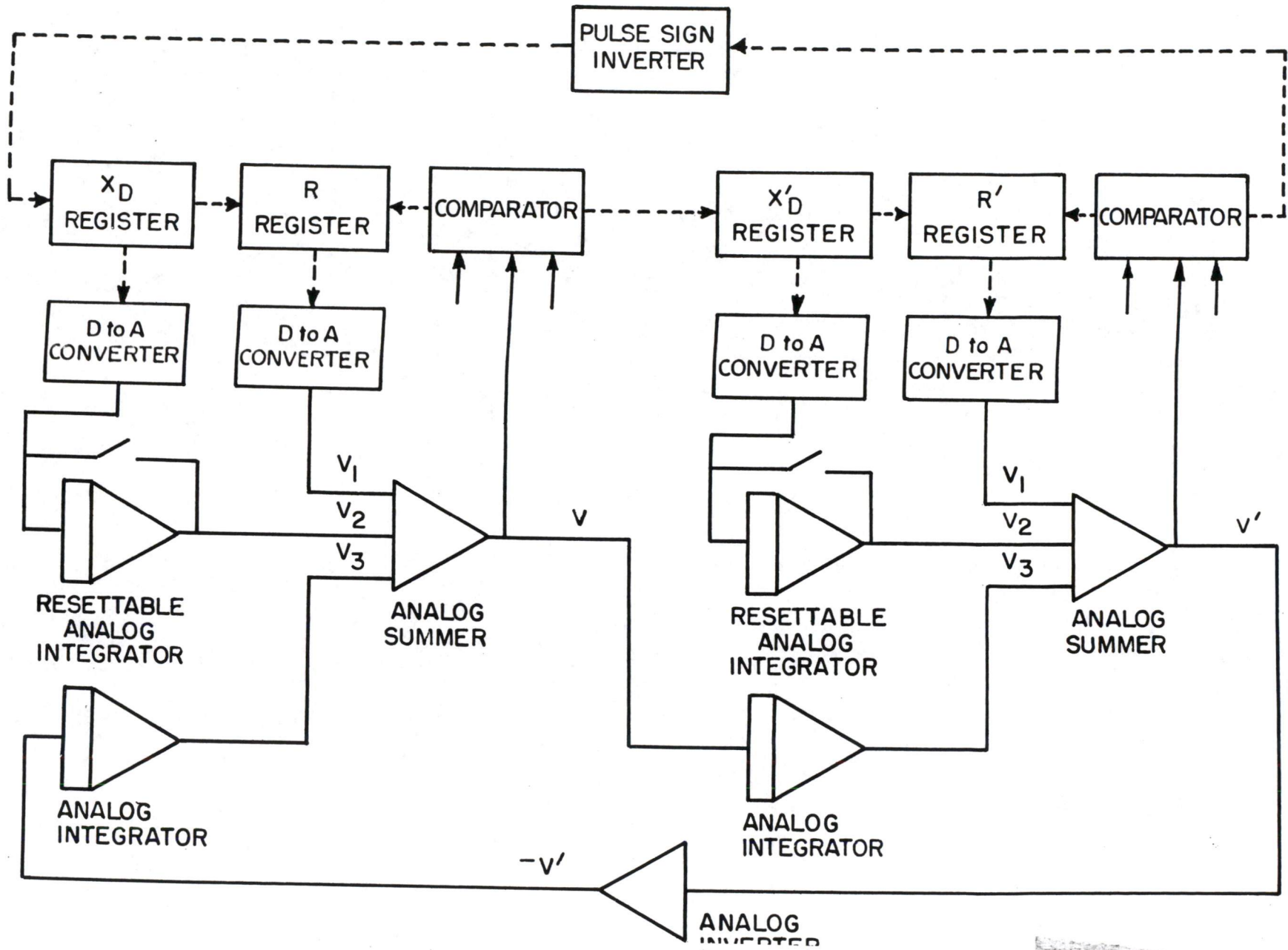


FIGURE 6 TWO INTEGRATORS INTERCONNECTED TO SOLVE: $\dot{x}' = x$, $\dot{x} = -x'$

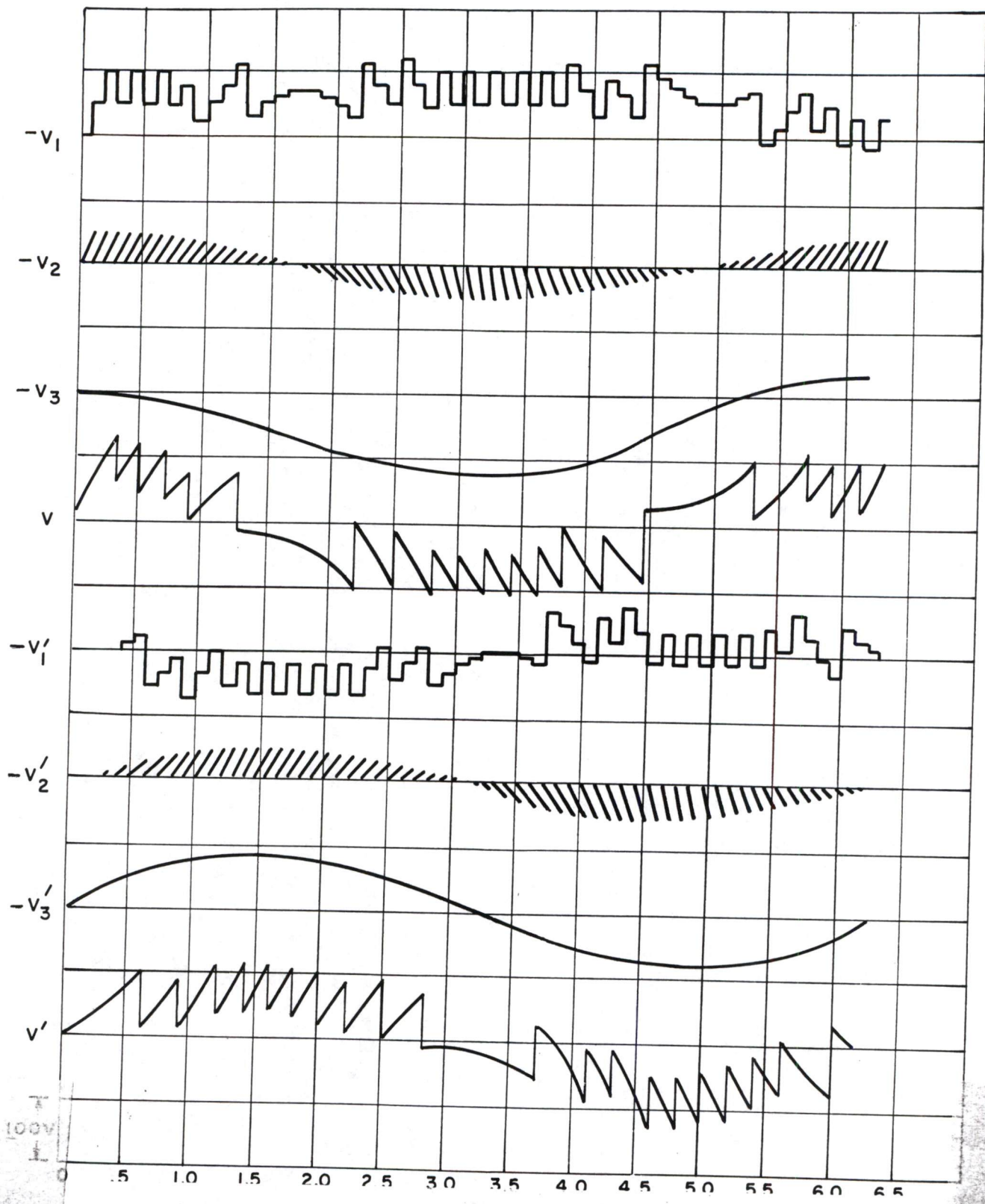


FIGURE 7 TIME HISTORY OF SOLUTION OF $\dot{x}' = x, \dot{x} = -x'$

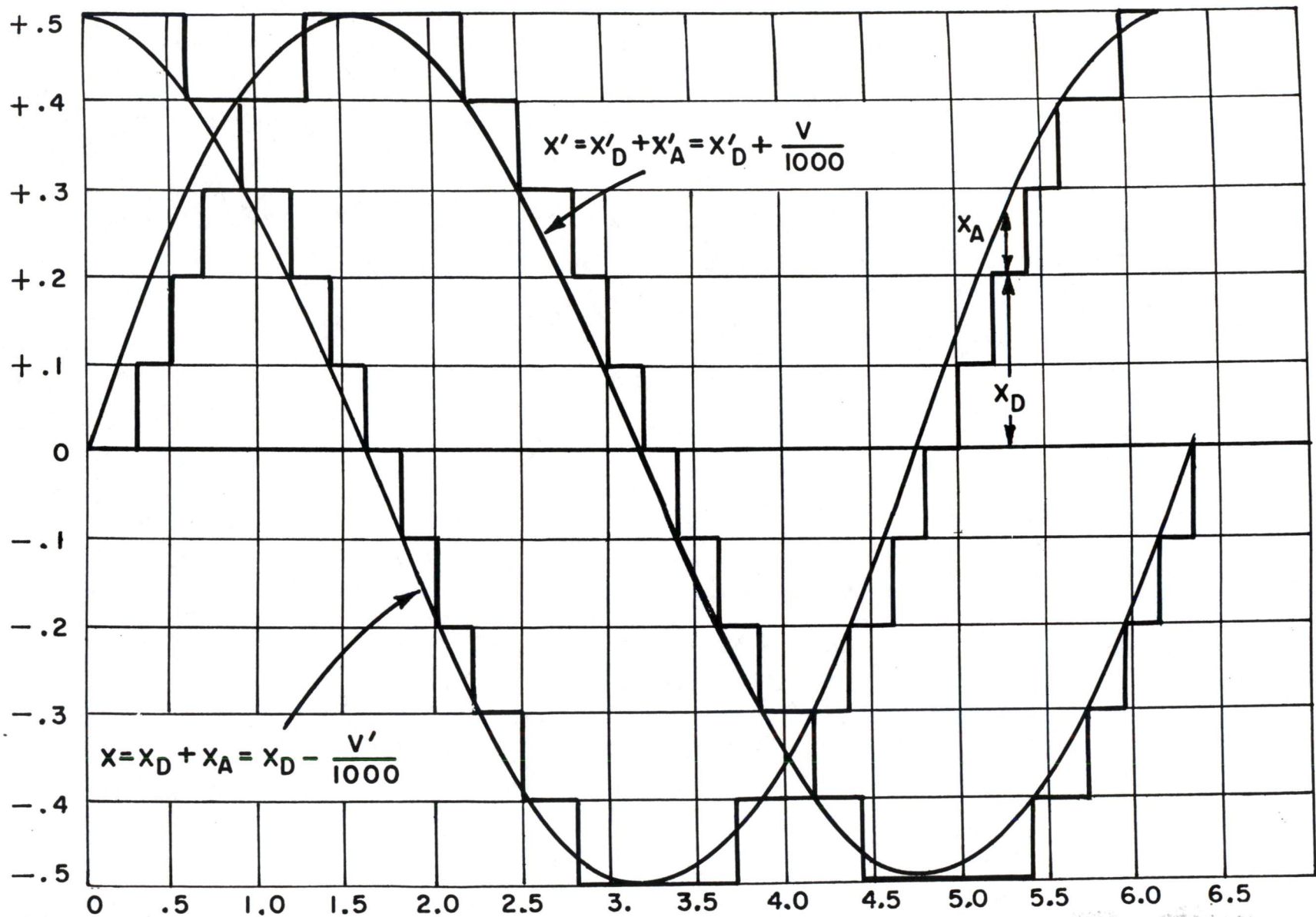


FIGURE 8 PLOT OF x and x' AS FUNCTIONS OF TIME

THE UNIVERSITY OF MICHIGAN
ANN ARBOR

File Copy

November 9, 1959

Harlan E. Anderson, Chairman
EJCC Publication Committee
Digital Equipment Corporation
Maynard, Massachusetts

Dear Mr. Anderson:

As requested, I have enclosed 4 copies of the final manuscript, 4 copies of the abstract and of a biography, and the originals plus 3 copies of the drawings. During the talk itself I will be using standard 3 x 4 slides.

Will it be possible to obtain reprints of the printed paper. The Logic of Computers Group would be willing to pay the expense of additional copies of the paper.

If there is any help I can give you with regard to the paper, please ask me.

Sincerely,

John Holland

John H. Holland
Logic of Computers Group
4001 Angell Hall

JHH:kt

File Copy

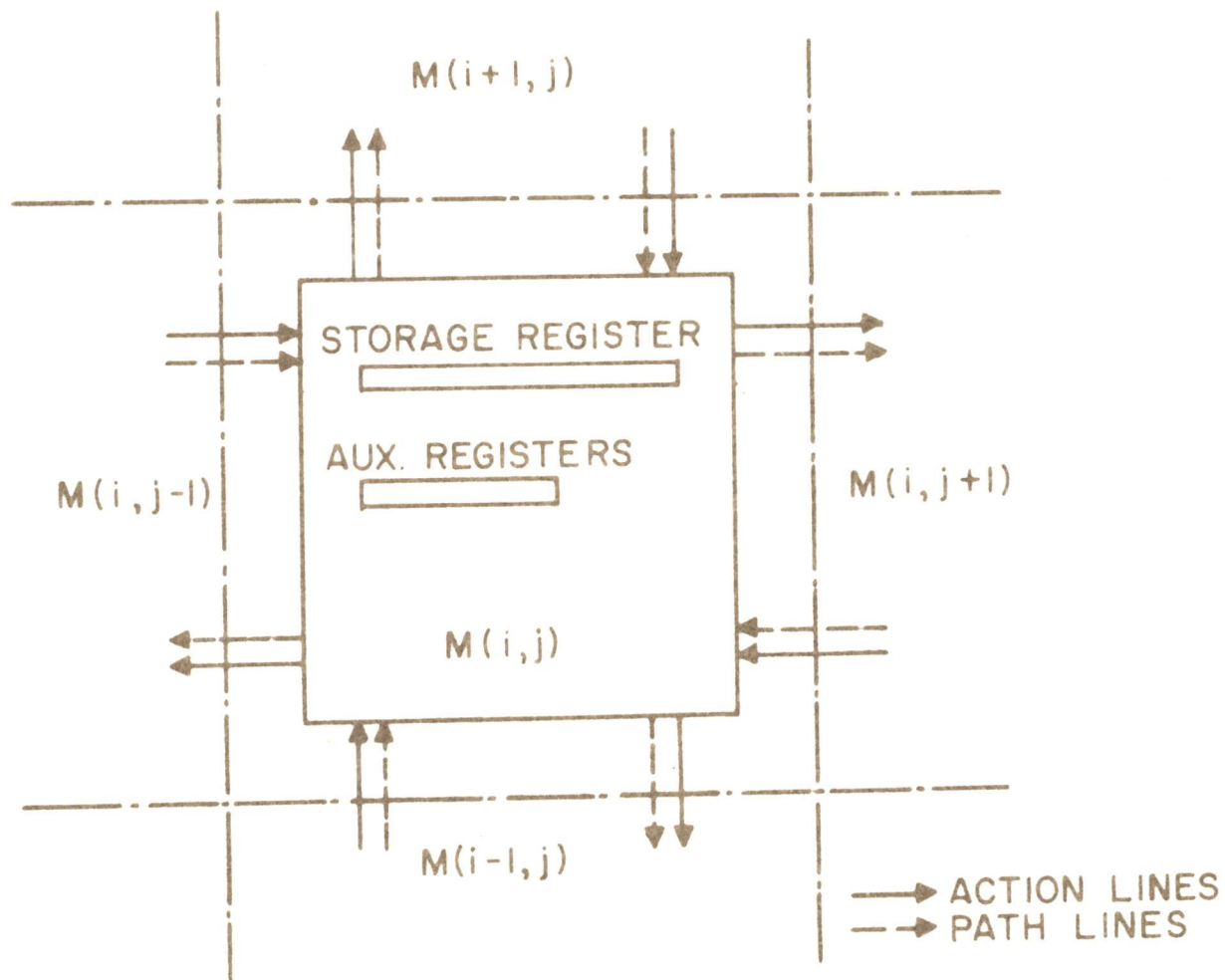


Fig. 1 - A module

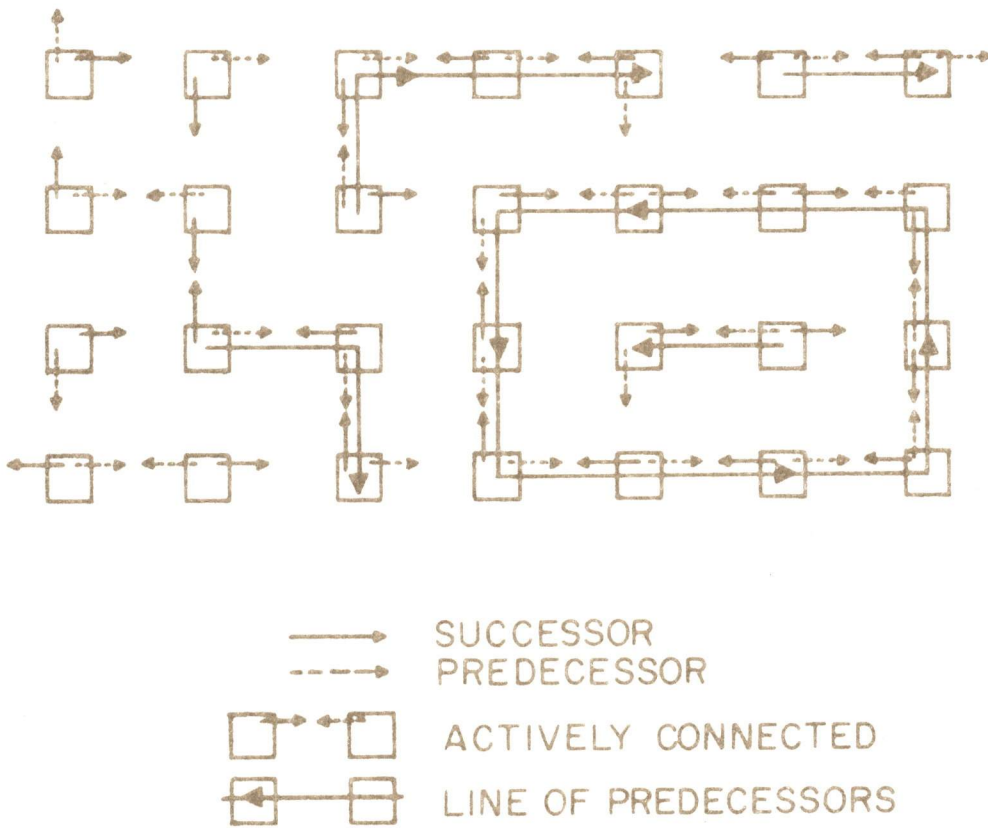
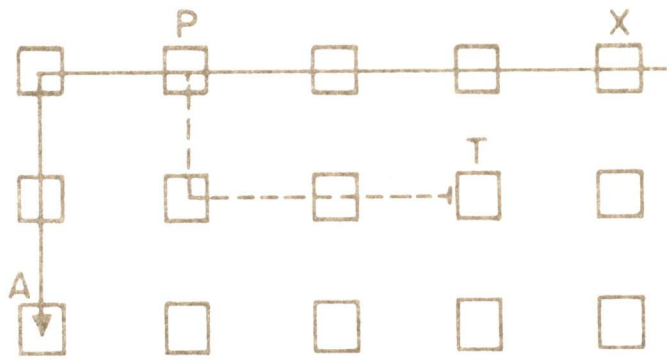


Fig. 2 - Lines of predecessors



X ACTIVE MODULE ("INSTRUCTION")
 P MODULE IN P-STATUS
 T PATH TERMINATION ("OPERAND")
 A MODULE IN A-STATUS ("ACCUMULATOR")



Fig. 3 - Modules used in the execution of an instruction

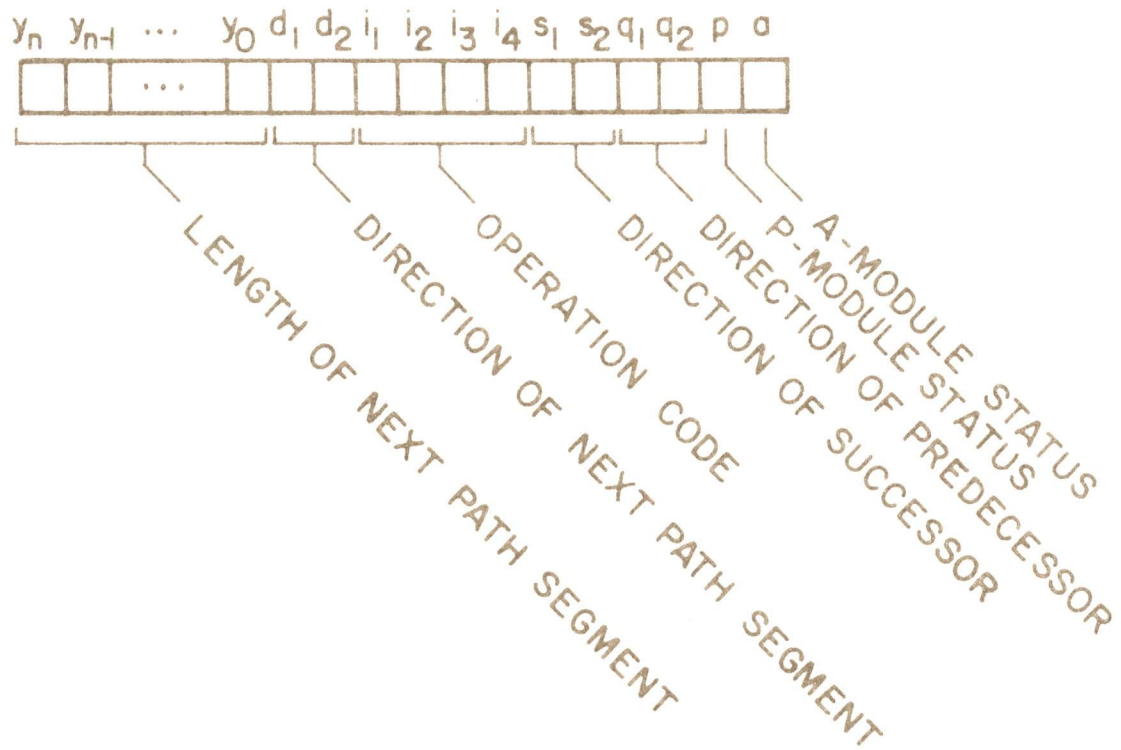


Fig. 4 - Control of module by storage register

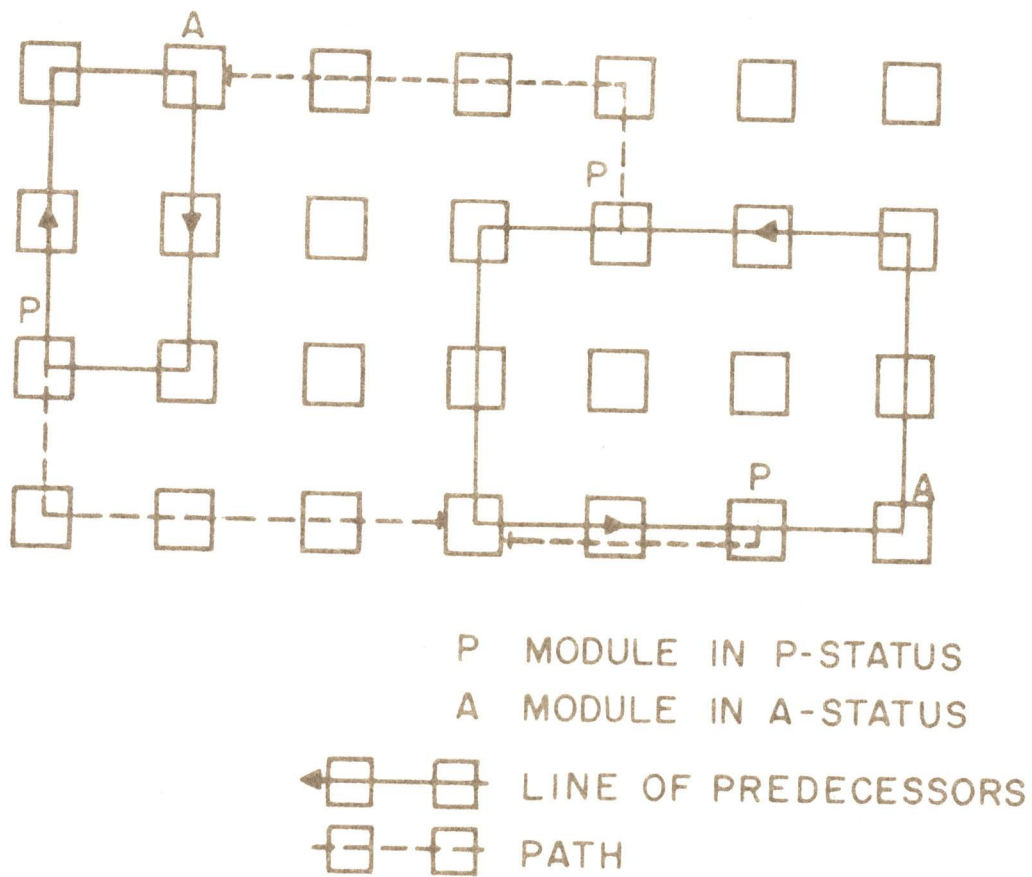


Fig. 5 - Two interacting subroutines

A Universal Computer Capable of Executing an Arbitrary Number of Sub-Programs Simultaneously.

John Holland

1. Introduction

This paper describes a universal computer capable of simultaneously executing an arbitrary number of sub-programs, the number of such sub-programs varying as a function of time under program control or as directed by input to the computer. Three features of the computer are:

- (1) The structure of the computer is a 2-dimensional modular (or iterative) network so that, if it were constructed, efficient use could be made of the high element density and "template" techniques now being considered in research on microminiature elements.
- (2) Sub-programs can be spatially organized and can act simultaneously, thus facilitating the simulation or direct control of "highly-parallel" systems with many points or parts interacting simultaneously (e.g. magneto-hydrodynamic problems or pattern recognition).
- (3) The computer's structure and behavior can, with simple generalizations, be formulated in a way that provides a formal basis for theoretical study of automata with changing structure (cf. the relation between Turing machines and computable numbers).

The computer presented here is one example of a broad class of universal

computers which might be called universal iterative circuits. This class can be rigorously characterized and formally studied (the characterization will be published in another paper). The present formulation is intended as an abstract prototype which, if current component research is successful, could lead to a practical computer.

2. General Description

The computer can be considered to be composed of modules arranged in a 2-dimensional rectangular grid; the computer is homogeneous (or iterative) in the sense that each of the modules can be represented by the same fixed logical network. The modules are synchronously timed and time for the computer can be considered as occurring in discrete steps, $t = 0, 1, 2, \dots$.

Basically each module consists of a binary storage register together with associated circuitry and some auxiliary registers (see fig.1). At each time-step a module may be either active or inactive. An active module, in effect, interprets the number in its storage register as an instruction and proceeds to execute it. There is no restriction (other than the size of the computer) on the number of active modules at any given time. Ordinarily if a module $M(i,j)$ at coordinates (i,j) is active at time-step t , then at time-step $t+1$, $M(i,j)$ returns to inactive status and its successor, one of the four neighbors $M(i+1,j)$, $M(i,j+1)$, $M(i-1,j)$, or $M(i,j-1)$, becomes active. (The exceptions to this rule occur when the instruction in the storage register of the active module specifies a different course of action as, for example, when the instruction is the equivalent of a transfer instruction).

The successor is specified by bits s_1, s_2 in $M(i,j)$'s storage register. If we define the line of successors of a given module as the module itself, its successor, the successor of the successor, etc., then a given sub-program in the computer will usually consist of the line of successors of some module. Since several modules can be active

at the same time the computer can in fact execute several sub-programs at once. We have noted parenthetically that there are orders which control the course of action - there are also orders equivalent to store orders which can alter the number (and hence the instruction) in a storage register. Therefore the number of sub-programs being executed can be varied with time, and the variation can be controlled by one or more sub-programs.

The action of a module during each time-step can be divided into three successive phases:

(1) During phase one, the input phase, a module's storage register can be set to any number supplied by a source external to the computer. The input phase will be discussed in section 5.

(2) During phase two, an active module determines the location of the operand, the storage register upon which its instruction is to operate. This the module does by, in effect, opening a path (a sequence of gates) to the operand. Phase two, called the path-building phase, will be discussed in section 3.

(3) During phase three, the execution phase, the active module interprets and executes the operation coded in its storage register.

The execution phase will be discussed in section 4.

3. Path-building

An active module determines the location of the storage register upon which its instruction is to operate by, in effect, opening a path to it. The path-building action depends upon two properties of modules:

First, by setting bit p in its storage register equal to 1, a module may be given special status which marks it as a point of origination for paths; the module is then called a P-module.

Secondly, each module has a neighbor, distinct from its successor, designated as its predecessor by bits q_1, q_2 in its storage register; the line of predecessors of a given module M_0 is then defined as the sequence of all modules $[M_0, M_1, \dots, M_k, \dots]$ such that, for each k , M_k is the predecessor of M_{k+1} and M_{k+1} is the successor of M_k (see fig.2). Note that the line of predecessors may in extreme cases be infinitely long or non-existent. The line of predecessors of an active module ordinarily serves to link it with a P-module (through a series of open gates). During the initial part of phase two the path specification bits y_0, \dots, y_n and d_1, d_2 , in the storage register of an active module M_0 , are gated down its line of predecessors to the nearest P-module (if any) along that line. The path specification bits are then used by the P-module to open a path to the operand (the storage register addressed by the active module).

Each path must originate at a P-module and only one path can originate at any given P-module. The path originating at a P-module is gated by means of a sequence of auxiliary registers called *-registers. Each module possesses 4 *-registers and if the module belongs to a path in direction

(b_1, b_2) the appropriate *-register, $(b_1, b_2)^*$, is turned on. When $(b_1, b_2)^*$ is on it gates lines (to be described) from the module $M(i, j)$ to its neighbor $M(i+b_1, j+b_2)$ permitting certain signals coming into $M(i, j)$ to be passed on to $M(i+b_1, j+b_2)$ and vice-versa. Since each *-register gates a separate set of lines, a module may (with certain exceptions) belong to as many as four paths. Once a *-register is turned on it stays on until turned off; thus a path, once marked, persists until erased.

The modules belonging to a given path can be separated into sub-sequences called segments. Each segment of the path is the result of the complete phase two action of a single active module. A segment consists of y modules extending parallel to one of the axes from some position (i, j) through positions $(i+b_1, j+b_2)$, $(i+2b_1, j+2b_2)$, ..., $(i+(y-1)b_1, j+(y-1)b_2)$, where $b_1 = \pm 1$ or 0 and $b_2 = \pm(1 - |b_1|)$; the module at $(i+yb_1, j+yb_2)$ will be called the termination of the segment (note that the termination of the segment is not a member of the segment). The segments are ordered so that the first segment constructed has as its initial module the P-module. The k^{th} segment constructed as part of the path has as its initial module the termination of the $(k-1)^{\text{th}}$ segment. If the path consists of n segments, the termination of the n^{th} segment (the last segment constructed) will be called the path termination (see fig.3).

As noted, the path specification bits y_n, \dots, y_0 and d_1, d_2 are gated down the line of predecessors from the storage register of the active module to the nearest P-module at the start of phase two. If $y_n = 0$,

bits y_{n-1}, \dots, y_0 and d_1, d_2 determine the length and direction, respectively, of the new segment. The total number of digits y_{n-1}, \dots, y_0 equal to 1 gives the length of the segment - if j of the digits are equal to 1 then the segment will be j modules long. The digits d_1, d_2 turn on and set an auxiliary register, the direction register, in the initial module of the new segment. This gives the direction b_1, b_2 of the segment. The direction registers of the other modules belonging to the segment are all off, but each of the modules belonging to the segment (including the initial one) has its $(b_1, b_2)^*$ register turned on.

When $y_n=1$, the final segment of the path originating at the P-module is erased. That is, the direction register in the initial module of the last segment is turned off and, as a consequence, all *-registers marking the last segment are turned off. If the path consists of a single segment or none at all the effect of $y_n=1$ is to turn off the direction register in the P-module thereby making the P-module the termination of the path. That is, in this latter case, the path has no segments but it does have a termination - the P module itself (note that the status of the P-module is unchanged).

The following additional rules apply to paths:

(1) When a given module is the termination of several paths and direction register on-pulses arrive over more than one path at the same time, t , the result is no action - the direction register is not turned on and none of the paths are extended.

(2) Only one path can proceed through a module in which the direction register is on. Whenever the direction register of a given module M is turned on or given a new setting, any paths already running through that module will now have it as their termination. Furthermore, for each such

path, the portion lying between M and the previous path termination is at once erased - the *-registers and direction registers marking that portion of the path are turned off.

(3) No P-module can belong to any part of a path other than its origin. If a path in the process of construction reaches a P-module then all construction ceases and the P-module becomes the termination of the path regardless of the value of digits y_n, y_{n-1}, \dots, y_0 . Further extension of the path will not be carried out unless the P-module's status is changed (its p bit set to zero).

4. Execution

Three modules play an important role during the execution phase of an active module: the active module itself holds the order code in bits i_1, i_2, i_3 of its storage register; the storage register of the nearest path termination contains the word to be operated on (the operand); finally there must be a module which serves as accumulator (see fig.3). In order to serve as an accumulator, the storage register of a module must first have bits (p,a) in it set to the value $(0,1)$, giving the module special status - A-module status. (Note that this means a module in P-module status, $p = 1$, cannot be an A-module). If $M(i,j)$ is an active module then the first A-module along its line of predecessors serves as the accumulator. An A-module serves, in effect, to terminate a line of predecessors, since it can have no designated predecessor (see the rules at the end of this section).

In the present formulation there are eight basic orders:

(1) The arithmetic operation ADD. Execution of ADD causes the number in the storage register at the nearest path termination (the operand) to be transferred to the nearest A-module and there added to whatever number is in the storage register of the A-module. (By using complements and iteration all the arithmetic operations, such as subtraction and multiplication, can be accomplished by means of this operation).

(2) The storage operation STORE. Execution of STORE causes the number in the storage register of the nearest A-module to be transferred to the storage register at the operand.

(3) The transfer operation TRANSFER ON MINUS. Execution of TRANSFER ON MINUS depends upon the number in the storage register in the

nearest A-module. If $y_n = 0$ in this number then the active module, after completing phase two, becomes inactive and its successor becomes active. If $y_n = 1$ then the module at the nearest path termination, rather than the successor, becomes active.

(4) The index operation ITERATE SEGMENT. If $y_n = 0$ in the nearest A-module, execution of ITERATE SEGMENT (upon completion of phase two) reduces the number in the A-module by 1 and the active module remains active without causing its successor to become active. If $y_n = 1$, then execution of the order simply causes the successor to become active and the active module inactive at the completion of phase one. This operation provides a convenient means of building long paths in a given direction since, if N is the number in the nearest A-module, the path-building phase of the active module is iterated N times.

(5) SET REGISTERS causes the first 9 bits of the number in the nearest A-module to be used to set all 9 auxiliary registers at the nearest path termination, the j^{th} register being set on if the j^{th} bit is a one (see section 6). It is important that the SET REGISTER order can give the operand module active status by setting the appropriate auxiliary register. In this case the active module gives rise to two active modules on the next time-step, its successor and the operand module. By this means one sub-program can initiate activity in another.

(6) RECORD REGISTERS causes the state of the 9 auxiliary registers at the nearest path termination to be recorded in the first 9 bits of the nearest A-module (in the same order as used by the SET REGISTERS instruction).

(7) NO ORDER causes the execution phase to pass without the execution of an order.

(8) STOP causes the active module to become inactive without passing on the activity to its successor at the next time-step.

With the exception of the STOP, ITERATE SEGMENT, and TRANSFER orders, the active module becomes inactive and its successor becomes active at the conclusion of the execution of an order.

It is possible for a given active module to have no nearest P-module (or A-module) for any one of three reasons: (1) the module does not have a line of predecessors, (2) none of the modules along the line of predecessors is currently designated a P-module (or A-module), (3) there is no P-module along the line of predecessors between the active module and the nearest A-module. If there is no nearest P-module then there is neither path-building nor execution of instruction with respect to the active module (regardless of the content of its storage register). If there is no nearest A-module along the line of predecessors then the instruction of the active module is not executed although the path-building phase will be carried out (assuming a nearest P-module).

The following additional rules apply to active modules and their action with respect to P-modules and A-modules:

(1) If M_0 belongs to the line of predecessors of M_1 , if the nearest P-module of M_0 is also the nearest P-module of M_1 , and if M_0 and M_1 are both active, then the action of M_0 proceeds normally but M_1 's action is as if it had no nearest P-module.

(2) If M_0 and M_1 are situated as in rule (1) except that they have the same nearest A-module, without sharing the same P-module, then the action of M_0 proceeds normally but M_1 acts as if it were executing a NO ORDER instruction.

(3) As mentioned earlier, a module can be given A-module status by setting the pair of bits (p,a) to the value $(0,1)$. This turns on an auxiliary register in the module, the A-register. At the same time the bits of another auxiliary register pair, the (D_1,D_2) -register, are set to match the bits s_1,s_2 in the module's storage register; i.e., when the A-register is on the (D_1,D_2) -register indicates the successor of the A-module.

Once a module is given A-module status it can be returned to normal status only in one of two restricted ways. The first way requires that a STORE order be executed by an active module which has the given A-module as its operand module (nearest path termination). Then, if bit a is 0 or bit p is 1 in the number being stored, the A-module reverts to normal status and the word in its storage register is that specified by the STORE instruction. Otherwise the A-module is unchanged, the STORE order not being executed. The other way of returning an A-module to normal status requires that the A-module receive external input during phase one (see section 5). The above restrictions prevent the A-module from changing status when numbers are placed in its storage register during the normal course of its operation as an accumulator. During the time a module is an A-module the bits in its storage register are not interpreted in any way except as the digits of a binary number.

(4) A module in A-module status can become part of a path (or several paths) so long as it is not to be the initial module of a path segment. In this latter case the path-building action, which would make the A-module the initial module of a segment, is not carried out - the A-module remaining the termination of the path.

(5) A given module can be acted upon simultaneously by 2, 3, or even 4 STORE instructions if it is the termination of more than one path. Some provision must then be made to resolve conflicts when the numbers being stored are not identical. In the present formulation the conflict is resolved digit by digit: a 1 is stored at bit j in the storage register if and only if at least one of the incoming numbers has a 1 at position j .

(6) When a STORE instruction changes the word in the storage register of a module it is assumed that this change does not take place until the completion of phase three. Thus, for example, there is no conflict when the STORE instruction of an active module acts upon that module's own line of predecessors or, for that matter, upon the module itself.

(7) If an active module has an A- or P-module as successor then, at the next time-step, the successor of the A- or P-module becomes active, rather than the A- or P-module itself (unless, of course, the instruction just executed specifies otherwise).

5. Input

During phase one, the initial phase of each time-step, a module's storage register can be set to any arbitrarily chosen value and its auxiliary registers to any desired condition. The numbers and conditions thus supplied are the computer's input. Although the number in the storage register can be arbitrarily changed at the beginning of each time-step, it need not be; for many purposes the majority of modules will receive input only during the first few moments of time ("storing the program") or only at selected times t_1, t_2, \dots ("data input"). Of course, some modules may have a new number for input at each time-step; in this case the modules play a role similar to the inputs to a sequential circuit.

6. Summary of Organization and Symbols

As noted in the general description of section 2, each module consists of a storage register plus some auxiliary registers. The discussions of sections 3,4 and 5 indicate that the auxiliary registers required are:

1) the E-register, a one bit register which is on if and only if the given module is active;

2) the A-register, a one bit register which is on if and only if the given module is an A-module (see rule (3) of section 4);

3) the D-register, a one bit register which is on if and only if the given module is the initial module of a path segment;

4) the (D_1, D_2) -register, a register, with two bits of storage, which indicates the direction (b_1, b_2) of a segment if and only if the D-register is on and which indicates the direction of the module's successor if and only if the A-register is on.

5) the $(b_1, b_2)^*$ -registers, each is a one bit register which is on if and only if the given module is a member of a path segment with direction (b_1, b_2) .

For formal purposes we can symbolize the state of a given register, X, at coordinates (i, j) and time t by the predicate $X(i, j, t)$ with $X(i, j, t) = 1$ if the given register is on at time t.

The storage register of each module in the present formulation consists of $n+12$ bits (see fig.4) labelled in the following order:

bit number:	$n+12$	$n+11$...	12	11	10	9	8	7	6	5	4	3	2	1
label:	y_n	y_{n-1}	...	y_0	d_1	d_2	i_1	i_2	i_3	s_1	s_2	q_1	q_2	p	a

The bits s_1, s_2 and q_1, q_2 designate the successor and predecessor, respectively,

of the module. If bit p is 1 the module has P-module status. If the pair of bits (p,a) are set to the value $(0,1)$ as the result of input or a STORE operation, the module has A-module status. During the path-building phase bits y_n, \dots, y_0 and d_1, d_2 in an active module are interpreted as segment length and direction respectively. During the execution phase bits i_1, i_2, i_3 in an active module are interpreted as the operation to be performed. The word in the storage register of an A-module is treated strictly as a binary number with y_n being the sign bit and the other $n + 11$ bits being arranged as indicated with y_{n-1} being the high order bit and a being the low order bit.

7. Comment

A universal machine in which the programs have a spatial organization has several properties over and above those usually associated with Turing machines and their concrete counterparts. For example, cycles in the program can actually be stored as cycles (of successors) in the rectangular grid (see fig.5). This, in effect, provides each cyclic sub-program with an instruction address counter which counts modulo the number of instructions in the sub-program (cf. an index register which can be set to cycle modulo any base number). Furthermore, each sub-program can be allotted a certain area in the grid and this allows the spatial arrangement of the sub-programs to match, for example, the structural organization of a process which is being simulated - each subprogram in this case directly simulating one of the components of the process.

Efficient programming of certain types of problem will require techniques similar to those required for asynchronous operation. That is, when several sub-programs are operating simultaneously, each sub-program will from time to time require results from other sub-programs, however these results will not in general be available at just the time desired. In problems like this, usually arising in the control or simulation of "highly-parallel" systems with many points or parts interacting simultaneously, the programmer will employ many of the techniques of the logical designer.

Problems such as the one just discussed emphasize the desirability of a computer formulation amenable to theoretical investigation. The present formulation is one example of a broad class of computers which can be rigorously characterized and investigated by abstract deductive techniques. Actually, the definition of this class of computers comes as part of an effort

to provide a formal basis for the study of growing automata. By considering the rectangular grid to be infinite (or potentially infinite) in each of its dimensions (in analogy to the infinite tape of a Turing machine) many problems of automata theory can be expressed in a formal framework similar to that provided by Turing machines for problems of computability. Thus, for example, models of various processes can be stated as programs, or classes of programs, for the machine and investigated both directly and theoretically.

There are several variants of the formulation given here which yield computers which are either more flexible or have simpler modules. As a single instance, the path-building procedure could be altered to make branching paths possible; in this way the same sub-program could operate on several storage registers simultaneously.

A final word about concrete realization of such a computer: a partial rendering of the logical diagrams for a module in the described computer indicates that a module with a 40 bit storage register could be constructed with approximately 1000 basic elements. If this is actually the case and if switching is accomplished with micromodular densities, say 10^8 elements per cubic foot, then the basic portion of a computer with 100,000 modules should be realizable within a volume of a few cubic feet (exclusive of input-output equipment, power supply, etc.).

Abstract: A Universal Computer Capable of Executing an Arbitrary Number of Sub-Programs Simultaneously.

John Holland

The paper describes a universal computer capable of simultaneously executing an arbitrary number of sub-programs, the number of such sub-programs varying as a function of time under program control or as directed by input to the computer. Three features of the computer are:

- (1) The structure of the computer is a 2-dimensional modular (or iterative) network; thus, in a concrete realization, high density elements and "template" techniques now being developed would be efficiently used.
- (2) Sub-programs can be spatially organized and can act simultaneously thus facilitating simulation or direct control of "highly-parallel" systems with many points or parts interacting simultaneously (for example, magneto-hydrodynamic systems or pattern recognition problems).
- (3) The structure and behavior of the computer can be given a rigorous symbolic formulation. Thus by considering the rectangular grid to be infinite in each dimension (in analogy to the infinite tape of a Turing machine), many problems of automata theory can be expressed in a formal framework similar to that provided by the Turing machine for problems of computability. In particular, problems concerning growing automata (cf. Von Neumann's scheme for self-reproducing automata and Church's potentially infinite automata) can be investigated both directly and theoretically.

Biography

Dr. John H. Holland currently holds a dual appointment at the University of Michigan as Associate Research Mathematician with the Logic of Computers Group and Lecturer in Psychology. After obtaining a BS in physics from M.I.T. in 1950 he joined IBM and participated in the planning and logical design of the IBM 701 computer. From 1952-1956 he was a consultant for IBM on the theory of "concept-forming machines" while doing graduate work at the University of Michigan. Dr. Holland received an M.A. in mathematics in 1954, joined the University of Michigan Engineering Research Institute in 1956, and received his Ph.D. in the communication sciences in 1959.