Games That Teach the Fundamentals of Computer Operation*

DOUGLAS C. ENGELBART†, MEMBER, IRE

To the Reader:
This paper is different. Its engineering content is neither novel nor difficult. It is, however, addressed to you, the computer engineering audience, to inform you of and instruct you in a novel method of teaching a group of laymen about computers. Very often schools call upon scientists and engineers in their vicinity to give presentations to their science or mathematics clubs. This is a valuable service we can provide. The techniques described in this paper are most appropriate to that end, and have been used by your editor and others, as well as by the author, who first suggested them. Try them if called upon, enjoy them, and if you find no other group to use them on, try them at your next party—it should be a howling success!
Reprints of this article may be obtained from WGBE, the Working Group for Better Education, 815 Washington St., Newtonville 60, Mass., for $0.75.—*The Editor*

Summary—**One who wishes to give a group of laymen a feeling for the way we computer engineers can coax sophisticated information-handling behavior from an organization of simple physical elements can provide a striking on-the-spot example by training his laymen to simulate various kinds of simple elements and by organizing them into a network whose behavior is obviously more sophisticated than that of any element. Each individual watches the up-down hand position of one or two others, and adjusts his own hand position according to a response task which is equivalent to that of an AND, OR, NOT, or flip-flop element—although task assignments are made in such a way that the participants don't hear a single esoteric word, nor realize that they might be doing "logic." Counters, shift registers, and adders may be organized and operated in a way which proves very entertaining to participants and on-lookers, and yet which provides them with very realistic basic concepts about how a computer might work.**

INTRODUCTION

THIS PAPER presents some examples of a form of teaching method that I have found to be very effective in giving people of all levels of sophistication a useful insight into the mysteries of digital-computer techniques. The novel feature of the teaching method is that it makes use of human participants to simulate the function of logical elements that are typical of those used in digital computers. A group of such participants can be "wired" into a network that will function in a manner very similar to that of an actual digital network. Simulation procedure is given in detail for two such examples which have proven to be particularly suitable for this purpose. Other exercises are included in complete but brief form.

Aside from the general parlor-game atmosphere of fun induced by these exercises, there are educational results of basic worth to be gained by the students. The mystery associated with computers tends to be dissipated when a person is assigned a very low-order task in a system of like elements, where no single element comprehends the over-all significance of its role, and yet the behavior of the combination of elements is obviously fairly sophisticated. In addition, it takes no great stretch of intuitive comprehension for the average layman to feel that the construction of simple circuits that can perform tasks analogous to those assigned to him and his fellow participants would put no great strain on an electronics engineer. After participating in or witnessing the exercises described below, a student can be expected to extend this feeling of acceptance to functional computer structures such as counters, registers, and adders, and it asks little more of him to accept the claim that other functions, such as those required of the different blocks in a block-diagram representation of a digital computer, can be realized by similar networks of similar elements. Other concepts that may be demonstrated by these exercises are pointed out later.

The descriptions of the two exercises that are given below will be presented as if the reader were being given explicit instructions for running a group of students through these exercises. Since I have found it to be a more effective presentation, the analysis of purpose will often be left until proper function has been achieved and recognized. This is done partly in order that the student may learn to appreciate the "moronic" and uncomprehending role of the building-block elements, and partly so that there is a chance for a few of the basic concepts to dawn spontaneously upon the student before they are discussed.

FIRST SIMULATION EXERCISE

Obtain four participants and arrange them side by side, facing the class, so that the class can easily watch them. Fig. 1 shows a possible arrangement. The "signal source" is you, the instructor. The instructions you give to the participants should be heard by everyone. You need not identify individual participants differently,

* Received by the PGEC, July 28, 1960.
† Stanford Res. Inst., Menlo Park, Calif.

other than will be done during the instructing period—the identifying letters in Fig. 1 are useful mainly for me to tell you what to do.

Give the following instructions: 1) Each participant is to be recognized as being only in one of two conditions, or states. Either his right hand is up (UP state), or his right hand is down (DOWN state). No other characteristic or attitude is to be recognized, and it is well to request that participants be very definite about the positions of their hands so that no ambiguity is presented to an observer. 2) Each participant or element will stay in whatever state he happens to be until he gets a particular signal (each will await a different signal, to be specified later). When he gets the signal he has been told to wait for, he responds by *changing* the position of his hand. If he is UP when he gets his signal, he changes to DOWN, and vice versa.

This completely defines the operating characteristics of the elements, and it only remains to identify for them their respective signal sources to obtain a working network of "electronic elements." 3) Tell element *D* that he gets his signal from *C*; that every time *C* drops his hand (goes from UP to DOWN), *D* is to recognize this as his signal and is to respond by changing the position of his hand. He is not to pay any attention to *C*'s changing from DOWN to UP, nor is he to be concerned with anything at all being done by you or by elements *A* or *B*. 4) Element *C* gets his signal from element *B*, watching for and responding to the change from UP to DOWN in the same narrow-minded fashion that was described above. Similarly, *B* gets his signal from element *A*, and changes the position of his (*B*'s) hand whenever *A* goes from UP to DOWN. This leaves only the establishment of the signal to which *A* is to respond, to complete the design of this piece of "electronic equipment." You tell *A* that he is to receive his signal through his ears instead of through his eyes, as did the other elements. His attention is to be given only to you, the primary "signal source," and whenever he hears you emit an audible signal of specified nature, he is to change the position of his hand. I usually use a hand clap for this signal, but any unique sound will do.

After giving the above instructions, and when it seems reasonably sure that they have been understood, give the four elements a test run to check their performance. Have them all go to their DOWN states, and then in your role as signal source begin emitting some signals. Pause between successive signals until everyone seems sure that he has done the right thing as a result of your last signal and the changes it has induced. You and the class watch for and point out errors in performance. Fig. 2 shows the sequence of states which should occur. Notice that after sixteen signal "pulses" the elements are all in the DOWN state again, and the same sequence of states will begin to repeat itself. It generally takes but one or two cycles (sixteen to thirty-two signals) to obtain fairly reliable performance from the elements.

When your four-element system seems to be functioning smoothly, it is time to introduce some simple props. You will need four cards, large enough so that the numeral drawn on each can be seen clearly by everyone in the class. On one side of each card will be drawn the number 1, 2, 4, or 8. It serves a definite purpose to keep the four elements from knowing what is on the cards (in fact, if you don't even let them guess that the faces contain numbers, so much the better), and so you should explain that each is to hold his card so that the class (but not he or his neighbors) can see its face when his hand is UP, but when his hand is DOWN, nobody (himself, neighboring element, or class) can see the face of the card. You then give the leftmost element (*A*) the card with the numeral 1, give the 2 to *B*, the 4 to *C*, and the 8 to *D*. It is your responsibility to place the cards in their hands so the numerals will be right-side up when

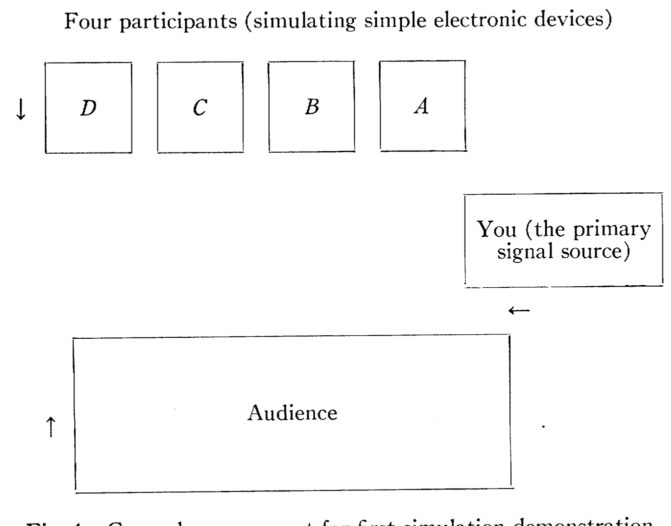
Four participants (simulating simple electronic devices)
D
C
B
A
You (the primary signal source)
Audience

Fig. 1—General arrangement for first simulation demonstration.

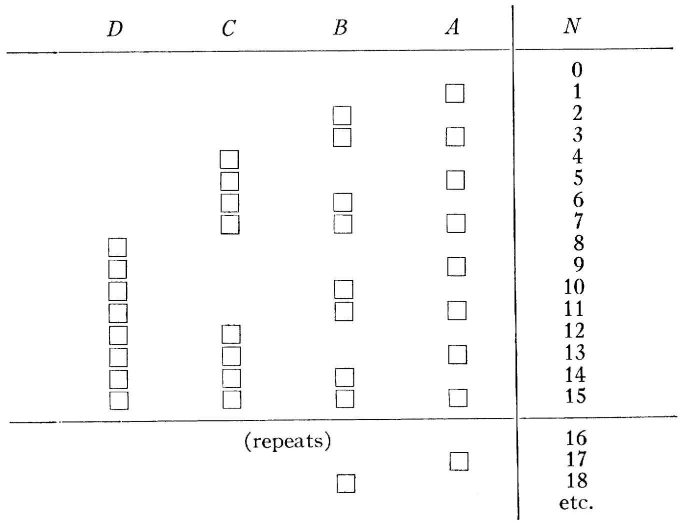

Fig. 2—Chart showing sequence of hand positions for elements of Fig. 1 after successive signals. (Flag indicates hand UP, no flag indicates hand DOWN. *N* indicates number of signals which signals source emitted before the four hands arrived at the indicated combination of positions.)

the elements are UP. It improves the general effect later if the class doesn't see what's on the cards until the four-element system begins to operate again.

With their numbered cards in their hands, and their hands DOWN, the four elements are again ready to receive signal pulses. As you emit succeeding pulses, the array of numerals appearing before the class should follow the sequence shown in Fig. 3 (which is the same as the sequence of Fig. 2, with the numerals drawn in). Without your saying anything, many of the people in the class will catch on to the way the sum of the visible digits represents the pulse count, and it is fun to watch the spontaneous comprehension grow.

I generally run the four elements quite slowly through the first sixteen-signal cycle without giving any hint to the class as to what to expect. Then I might suggest that they try "putting together" what is visible on the cards at any time, before I run through another cycle. Not until most of the class appears to see that counting is being accomplished do I let the elements see what is on the face of their cards.

What has been simulated by your human participants is, of course, a binary counter—a device which is built up of very simple elements, but which nonetheless is capable of performing a relatively sophisticated task. Pointing out that none of the individual elements knew what was on the face of this card, nor was otherwise given any idea of the significance of the very simple task which he was asked to perform, brings out the very important idea that proper organization can give a group of elements a capability which is significantly greater than any element alone can possess. Later expansion of this idea, in the next exercise, does a great deal to orient students about this basic procedure which has been followed to obtain all of our electronic digital computers. In every case, we build upon the functional capabilities of very simple basic elements, organizing multitudes of them into a system which is very sophisticated.

It should also be pointed out here that the peculiar way in which we have given a numerical weight to each of these elements has allowed us to represent numbers quite handily, even though the individual elements each had only two states of existence. You can mention that this is an example of a binary numbering system, which is used in one form or another by essentially every electronic digital computer. If plenty of time is available, one may digress at this point to explain further about binary numbers. However, the next simulation exercise requires no deeper understanding than is usually obtained directly from this first exercise.

D	*C*	*B*	*A*	*N*
				0
			$\boxed{1}$	1
		$\boxed{2}$		2
		$\boxed{2}$	$\boxed{1}$	3
	$\boxed{4}$			4
	$\boxed{4}$		$\boxed{1}$	5
	$\boxed{4}$	$\boxed{2}$		6
	$\boxed{4}$	$\boxed{2}$	$\boxed{1}$	7
$\boxed{8}$				8
$\boxed{8}$			$\boxed{1}$	9
$\boxed{8}$		$\boxed{2}$		10
$\boxed{8}$		$\boxed{2}$	$\boxed{1}$	11
$\boxed{8}$	$\boxed{4}$			12
$\boxed{8}$	$\boxed{4}$		$\boxed{1}$	13
$\boxed{8}$	$\boxed{4}$	$\boxed{2}$		14
$\boxed{8}$	$\boxed{4}$	$\boxed{2}$	$\boxed{1}$	15
				16
			$\boxed{1}$	17
				etc.

Fig. 3—Representation of what audience sees, after successive signal pulses, when "Elements" hold number-weight cards in their hands.

Second Simulation Exercise

Here you will need nineteen participants, besides yourself. Nine of these will be of one general type, designated by numbers, and ten will be of another basic type, designated by letters. There are many possible ways in which the elements may be arranged for this exercise, but the arrangement shown in Fig. 4 is what I usually use. The arrows indicate the preferred direction in which the respective participants should face. Deploy your participants and give each a card, similar to those used in the first exercise, that will carry his identifying letter or number. Most of the lettered cards have numbers on

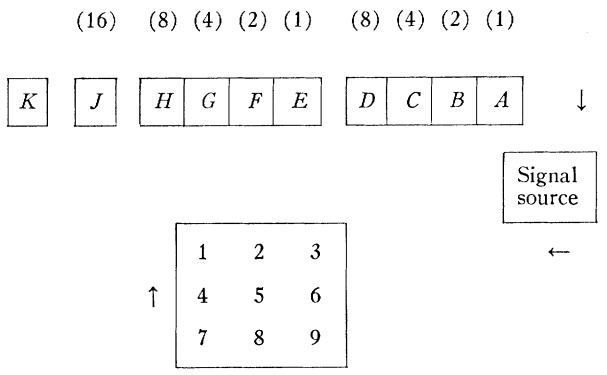

Fig. 4—Recommended arrangement of human elements for second simulation exercise. (Symbols in parentheses refer to the number weight to be given to the corresponding element. Groupings as shown are helpful for functional identification later. I usually seat numbered elements, with remainder of audience moved away a bit to set participants apart, and have lettered elements stand at front.)

the backs (see numbers in parentheses in Fig. 4), but these are to be ignored for the time being.

Next, each individual element must be instructed as to his task. Consider the numbered elements first. Each has a job of watching one or two other elements, and *continually* adjusting his state to respond to the elements he watches according to some assigned simple rule. The assignment of elements for each to watch, and the nature of his response to the states of these elements, is listed in Table I. The interpretations of these task assignments are as follows:

1) An UP-task element wants to keep his hand UP, but finds it possible only when both of the elements he watches are UP (otherwise, then, his hand is DOWN).

2) A DOWN-task element wants to keep his hand DOWN, but finds it possible only when both of the elements he watches are DOWN (otherwise, then, his hand is UP).

3) An OPPOSITE-task element holds his hand in the state opposite from that of the element he watches.

TABLE I
ASSIGNMENTS FOR THE NUMBERED ELEMENTS IN THE SECOND EXERCISE

Element	Watches	Task
1	*A*, *E*	UP
2	1, *K*	UP
3	*A*, *E*	DOWN
4	1, *K*	DOWN
5	3, 4	UP
6	3, *K*	DOWN
7	5	OPPOSITE
8	6, 7	UP
9	2, 8	DOWN

Participants with OPPOSITE tasks usually need no training or practice, but I generally give the other types of participants some practice. For instance, after deploying the nine numbered elements and giving them their identification cards, I ask Participants 1, 2, 5, and 8 to stand. I explain that later each will be assigned to watch some two particular elements, and tell them how an UP element is supposed to respond. For practice, I ask them to pretend that they have been assigned to watch both of my hands, and then I run through different up and down combinations of my two arms and make sure that they all respond correctly. When they seem to have learned, I ask them to be seated and then ask Participants 3, 4, 6, and 9 to stand. Their DOWN task is described, and I give them practice arm signals in a similar fashion. Despite the inverse similarity between UP and DOWN tasks, it always seems to take longer to train the DOWN elements.

It may help to have the individual's task designation written on his identification card, and then to write these task interpretations on the blackboard for all to see. Ask all participants to hold their identification cards in their UP-DOWN hands so that those assigned to watch them will have easier tasks. Also, remind the participants that UP and DOWN indications should be definite, *i.e.*, their hands should be all the way up or all the way down, never at ambiguous in-between positions.

Now instruct the lettered elements. They do not change their states in the same way as do the numbered elements, which change continually to adjust to the states of their watched elements. The lettered elements hold a given state until they get a signal from the signal source. This signal is composed of two parts. First comes a "bonk" (or other suitable signal, preferably a meaningless noise), whereupon each lettered element is to note the state of the particular element which he is assigned to watch, and then comes a "bleep," whereupon each lettered element assumes the particular state which he (just previously) observed his watched element to have when the "bonk" signal occurred. The lettered elements are not to change state until the "bleep" signal. It is roughly fifty per cent probable that at the time of the "bonk" signal, a lettered element will observe his watched element to be in the same state that he is, in which case he will not need to change his state when the "bleep" signal occurs.

I usually give these lettered elements a training session in a manner similar to those given the numbered elements. In this case, I ask them all to pretend they are assigned to watch my right arm, and I put it in different positions and emit the "bonk-bleep" signals. I usually try to trick them by changing my hand immediatately after the "bleep" (which may happen in actual operation, and to which they aren't supposed to respond). For some reason, this temporary-storage task is the *most* likely of any to give you trouble later on during operation, and it is worthwhile to spend a little extra time in training.

The element which each particular lettered element is assigned to watch is shown in Table II. Again it would be wise to ask everyone to hold his identification card in his UP-DOWN hand, and to assume very definite hand positions. It is also worth stressing that, while the numbered elements respond instantly to any change in their watched elements, no matter when they

TABLE II
ASSIGNMENTS FOR THE LETTERED ELEMENTS IN THE SECOND EXERCISE

Element	Watches
A	*B*
B	*C*
C	*D*
D	Dummy Down*
E	*F*
F	*G*
G	*H*
H	*J*
J	9
K	5

* *D* pretends to be watching a "dummy" element whose hand is always down.

occur, the lettered elements only change when they are told to do so by the signal source, no matter what their watched elements do.

In this second exercise, I usually describe the response characteristics of every element (and give him some training exercise) before I define the "interconnections," *i.e.*, before I define where each of them is to look for his particular hand-changing information. I often prepare a small card for each participant, describing his particular task and telling him whom to watch, and then as I pass these out I repeat this information aloud so that everyone can follow the process of "wiring together this network of electronic elements." If you can put Tables I and II on a blackboard, or reproduce them and hand them out, this will be particularly helpful to nonparticipating observers. (Interested and observant nonparticipants can often help you spot malfunctioning elements during operation of this "network." Encouraging this can provide you with a real help, and will give the nonparticipants some sense of participation.)

You are now ready to try out your twenty-element digital system (which includes you, the signal source) to see how it performs. You should expect some malfunctions at first. Warn your group that you are getting ready for them to operate as a system, and have everyone pay attention while you set Elements *A* through *K* to specific initial states. Ask that the lettered elements be sure to show the lettered identification sides of their cards to the main group when their hands are in the UP position. Then try the following initial setting, for example. Have *A*, *B*, *F*, and *G* take the UP state, and the rest of the lettered elements take the DOWN state. Make a note of this initial state in some conspicuous way so that you and the group can refer to it later.

Now tell all of your numbered elements to assume the states dicated by their tasks and by the states of their assigned elements, and to keep doing this through all of the successive changes of this operation. When the different hands have stopped oscillating between the UP and DOWN states, say "bonk," pause a moment for the lettered elements to decide what they are supposed to do, and then say "bleep." You will want to give four more of these "bonk-bleep" signals (five in all), pausing between each to allow the elements to perform their tasks and come to equilibrium, before the particular operation of this system is finished.

After the fifth "bonk-bleep" signal, you should find (in this example) that only *E* and *H* of the lettered elements are in the UP state (states of numbered elements are not pertinent at this time). If the system has given you this result, note this in the same manner as you noted the initial states of the lettered elements, and then have these two elements turn their cards so that the group can see their associated number weights. Point out that the associated 1 and 8 serve to indicate that the states of Elements *E* to *J* now represent the number 9. Then ask the lettered elements to assume the states they held at the start of this operation (*A*, *B*, *F*, *G* in UP state), with the numbered sides of their cards showing, and point out that the states of the two groups *A* to *E* and *F* to *J* now represent the two numbers 3 and 6.

This is the point at which people are to realize that all of the activity associated with the five "bonk-bleep" signals was the "thinking" process by which this little computing system added the numbers 3 and 6 to get 9. After these five signals, the digit held by elements *A* to *D* will have disappeared, while that held by elements *E* to *J* will have been replaced by the sum of the original two digits.

In case the five "bonk-bleep" signals did not result in having only *E* and *H* of the lettered elements in the UP state, you must do some trouble-shooting in your computer to find out which element or elements failed to function properly. Establish the initial conditions again, and begin going through the sequence of operations. Use of the sequence chart of Fig. 5 should enable you to locate the "defective" component(s), and from there it is a matter of judgment whether the individual situations call for repair or replacement. Very often a guilty element will realize its mistakes and cure itself, so you may never find what caused a given malfunction of the network.

Once your little computing system seems to be working reliably, you can try adding other pairs of digits. In each case, decide upon the two digits to be added, and enter them into the two "registers" (have the register elements *A* to *D* and *E* to *J* show the number-weight sides of their cards during the setting operation, and try to get them to decide for themselves which elements should be UP when they are told the digit which they are to represent). When the registers are set, have the lettered elements turn their cards so as to show their respective identities during the subsequent "thinking" operations. Now the numbered elements are to become

Element	Initial State	After "Bonk-Bleep" Signals				
		1	2	3	4	5
A	*X*	*X*				
B	*X*					
C						
D						
E		*X*	*X*			*X*
F	*X*	*X*			*X*	
G	*X*			*X*		
H			*X*			*X*
J		*X*			*X*	
K			*X*	*X*		
1		*X*				
2						
3	*X*	*X*	*X*			*X*
4		*X*	*X*	*X*		
5		*X*	*X*			
6	*X*	*X*	*X*	*X*		*X*
7	*X*			*X*	*X*	*X*
8	*X*			*X*		*X*
9	*X*			*X*		*X*

Fig. 5—Sequence of states, for each element of the system identified by Fig. 4 and Tables I and II, during an operation cycle which began with only *A*, *B*, *F*, *G* of lettered elements set to the UP state. (*X* indicates UP state, blank space indicates DOWN state.)

active, and assume states in accordance with their tasks and the states of their watched elements. Then you are to emit your sequence of five judiciously-spaced "bonk-bleep" signals, after which you ask the E to J elements to show their number weights so that the result of the addition operation may be seen.

It is interesting to test this little computer in several ways. For instance, try adding zero to a number, or add some number to zero. Also, add two numbers whose sum is greater than fifteen, to give element J a chance to end in the UP state. (The maximum sum that can be represented is 31, and if the actual sum, S, which should result from a given addition operation is greater than 31, then the number to expect as a result is $S-32$.)

General Discussion

It may be helpful to the class to show them a sketch such as Fig. 6, where the general function of the different groups of elements used in the second simulation exercise is portrayed. They seem to appreciate being told that this is the type of so-called block diagram which a computer engineer would use to represent this little computer. During the successive addition steps, the patterns which initially represented the addend and augend are shifted to the right, to make the successive rightmost bits of information available to the logical network of the adder. In this type of addition, there is a carry to consider between successive steps just as there is between the successive steps involved in adding multidigit decimal numbers. Between each "bonk-bleep" signal, our adder network takes into account the states of the current "least significant digits" in the registers, and the state of the carry-storage element K, to decide what the next states of the elements J and K should be. At the next "bonk-bleep" signal, J and K assume their new states as determined for them by the "logic" network in the adder during the preceding interval, while the remaining patterns in the two registers move another step to the right. In this fashion, the two original numbers shift off the right end of their registers, while the sum is being constructed bit by bit and moved into the upper register from the left.

All of the numbered elements belong in the functional group designated as the "adder." These are the decision-making elements, and are commonly called "logic" elements by people in the computer art. There is a way to interpret the roles of the numbered elements, different from that given by the assignments and task descriptions associated with Table I, which brings out the concept of "logic elements" quite clearly. (I purposely avoid mentioning either "logic" or this other interpretation until after the exercise has worked, so that it is easier to prove to the class that the elements themselves don't know about logic—they are designed to respond to UP's and DOWN's in certain ways, and it is *our interpretation* that invests them with the property of doing logic.) An UP-task element can be called an AND element. For instance, Element 1 will be in the UP state if Element A is UP *and* Element E is UP. A DOWN-task element can be called an OR element (*e.g.*, Element 3 will be in the UP state if Element A is UP *or* Element E is UP, or if both are UP). An OPPOSITE-task element can be called a NOT element (*e.g.*, Element 7 is UP if Element 5 is *not* UP). The function of the adder can be designated by two statements, one of which states the conditions of A, E, and K for which the state of J should go to UP at the next "bleep" signal, and the other of which does likewise for K. These statements are compounded only from (A UP), (E UP), (K UP), (AND), (OR), and (NOT) statement components, and the logic network of the adder compounds these same things in a physical manner.

For instance, J should go to the UP state next if any one, or else all three, of A, E, and K are UP. If (A) is used to designate the statement, "A is UP," and (NOT A) for "A is not UP" (and so on for the other elements), we can use our AND, OR and NOT elements to generate the condition equivalent to the foregoing statement: (J) next if [(A) AND (NOT E) AND (NOT K)]

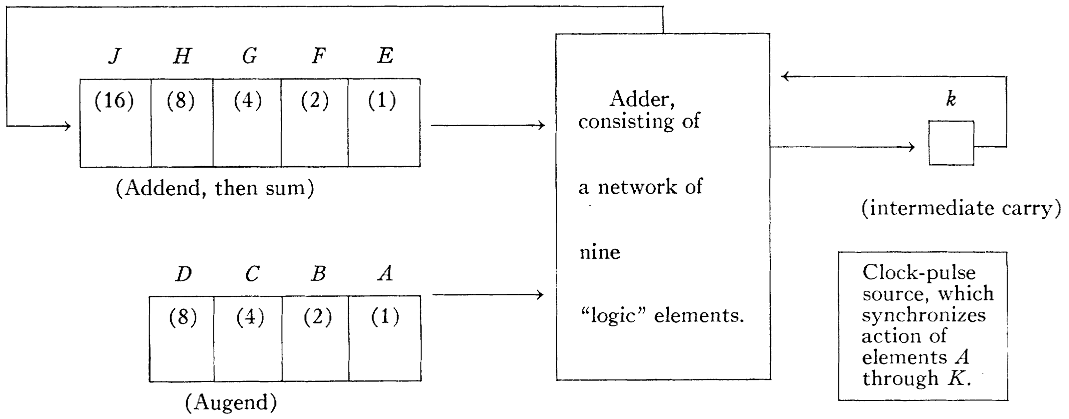
J H G F E
(16) (8) (4) (2) (1)
(Addend, then sum)
D C B A
(8) (4) (2) (1)
(Augend)
Adder, consisting of a network of nine "logic" elements.
k
(intermediate carry)
Clock-pulse source, which synchronizes action of elements A through K.

Fig. 6—Functional diagram of simple adding machine which can be simulated by the twenty human "Elements" (including clock source) of Fig. 4.

OR [(NOT A) AND (E) AND (NOT K)] OR [(NOT A) AND (NOT E) AND (K)] OR [(A) AND (E) AND (K)].

It can be noted that there are more AND-OR-NOT's used here than in our adder network. Part of the problem of a computer engineer is to devise logical networks which use the fewest elements for each job, and the network used for our adder is the result of just such a component-minimization study. It would take quite a bit of study for someone not a computer engineer or logician to analyze our adder network and see that its logical condition for the next J state is equivalent to the ones given above.

Each of the lettered elements provides the function of temporary storage for some UP-DOWN state that represents information which we want temporarily to save. At each "bonk-bleep" signal, a storage element takes new information from either another storage element, or a logic element whose state depends upon that of one or more storage elements within some AND-OR-NOT condition. This is the very essence of the workings of any digital computer.

Conclusion

Several important benefits which students can gain from these exercises are mentioned in the opening paragraphs of the paper. These lessons can be better appreciated after having read through the exercise descriptions, and will be especially appreciated after actually running through the exercises with a group. It might be worth tabulating these and other benefits so that you can better point them out to your class:

1) Organization of functional elements of a given degree of capability can yield a system which possesses a considerably higher degree of capability.
2) Any given participating element need not have the faintest comprehension as to the function of the system, or as to his role in that system.
3) From 1) and 2) it is easy to realize that no mystical powers of comprehension or intelligence have to be provided by physical phenomena. All of the capability in a computer is achieved by repeated application of principle 1). First we organize raw physical phenomena into special shapes and a special environment to get unit elements possessing capabilities such as simulated by our participants. Then we organize groups of these elements to get functional blocks, which we further organize to get a basic computer, capable of performing a limited repertoire of very specific tasks. After this we must organize the sequences of task commands given to the computer in order to obtain a "routine" which enables the computer to do a fairly complicated task. Generally, then, we must cleverly assemble groups of routines into a "program" which, finally, in conjunction with the physical machine we have constructed, gives us the impressive information-handling facility with which the "computer" awes the uninitiated.
4) Just as in real electronic computers, the proper result in our little system depended upon every element operating correctly every time. This points out the tremendous emphasis that has to be placed upon reliability in the engineering design of a computer. If we used enough people, we could simulate an entire computer with the same kinds of "elements" that we used above in the second simulation exercise; however, people are far too unreliable to allow such a computer to give dependable results.

Props needed for these first two simulation exercises include the following:

1) For the first exercise you need four cards (I use 4- by 6-inch white cards), each with a different one of the numerals 1, 2, 4, or 8 drawn clearly and heavily on one side, leaving the other side blank.
2) For the second exercise you need nineteen similar cards. On nine of these are printed, respectively, the digits 1 through 9, preferably on both sides of the card. On one side of each of the other ten is printed a different one of the letters A, B, C, D, E, F, G, H, J, K, and on the other sides of the first nine of these are printed the numbers associated with the given lettered elements in Fig. 4. Element K can have its letter printed on both sides.

Appendix I

Additional Exercises

A condensed description is included here for each of three other human-simulated digital networks that has proven useful to me in the past. Anyone who enjoys logical design will no doubt generate his own types and variations of such exercises if he becomes interested.

Decimal Counter—(Fig. 7 and Table III)

The first exercise (Fig. 1) demonstrated a pure binary counter, to which could be added more, similar, counting elements to provide count capacity up to ever higher numbers of significant binary "digits." It is interesting to demonstrate that a slight complication in the design can provide decimal presentation of the count. The arrangement shown in Fig. 7 accomplishes this. As can be noted in Table III, the lettered elements have tasks which are the same as those of the corresponding counting elements in the first exercise, except that here most of them must be alert to a secondary signal that may tell them to go unconditionally to the DOWN state. The numbered elements 1 and 3 have decision tasks

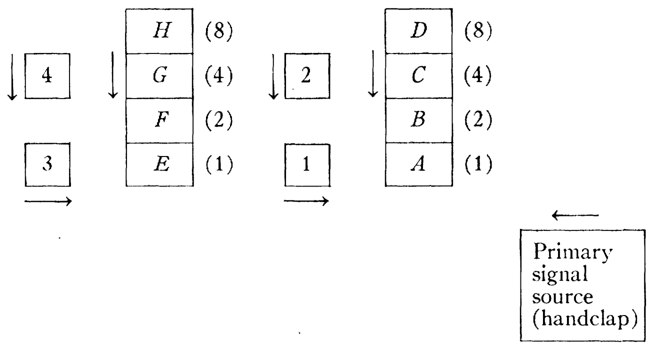

Fig. 7—Distribution of elements for a two-stage decimal counter.

TABLE III
TASK ASSIGNMENT FOR THE TWO-STAGE DECIMAL COUNTER OF FIG. 7

Element	Task
A	Changes state whenever hears clap.
B	Changes state whenever sees *A* go from UP to DOWN except goes to DOWN when hears "alpha," no matter what *A* does then.
C	Changes state whenever sees *B* go from UP to DOWN, except goes to DOWN when hears "alpha," no matter what *B* does then.
D	Changes state whenever sees *C* go from UP to DOWN, except goes to DOWN when hears "alpha," no matter what *C* does then.
E	Changes state whenever hears "alpha."
F	Changes state whenever sees *E* go from UP to DOWN, except goes to DOWN when hears "beta," no matter what *E* does then.
G	Changes state whenever sees *F* go from UP to DOWN, except goes to DOWN when hears "beta," no matter what *F* does then.
H	Changes state whenever sees *G* go from UP to DOWN, except goes to DOWN when hears "beta," no matter what *G* does then.
1	Goes UP whenever *B* and *D* are both UP, otherwise is DOWN.
2	Emits the word "alpha" whenever it sees 1 go to UP.
3	Goes UP whenever *F* and *H* are both UP, otherwise is DOWN.
4	Emits the word "beta" whenever it sees 3 go to UP.

exactly similar to those of UP elements in the adder network of the second exercise. Elements 2 and 4 are of a new type; each is a kind of a signal translator and distributer. Element 1 recognizes the binary ten-count state of the first stage, and Element 2 translates this into the verbal "alpha" signal which is recognized as a set-to-zero signal by the first stage and a count-one signal by the second stage. Similarly, Element 3 recognizes the ten-count state of the second stage (after the tenth "alpha" signal), whereupon element 4 translates this into a "beta" signal that tells the second stage to reset to zero, and which can be used as a count-one signal by a third stage if desired.

Register Transfer Control—(Fig. 8 and Table IV)

This exercise provides some conceptual insight into the manner in which information may be transferred from one place to another under automatic control. The temporary-storage Elements *A* through *H* operate just as did those in the adder network of the second exercise,

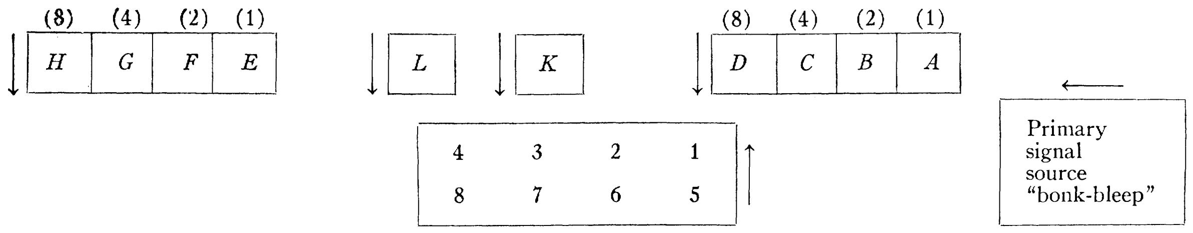

Fig. 8—Distribution of elements for register-transfer-control exercise.

TABLE IV
TASK ASSIGNMENTS FOR THE ELEMENTS* OF THE REGISTER-TRANSFER-CONTROL EXERCISE SHOWN IN FIG. 8

Temporary Storage Elements		Logic Decision Elements		
Element	Watches	Element	Watches	Task
A	*B*	1	*K*	OPPOSITE
B	*C*	2	1, *A*	UP
C	*D*	3	*K*, *E*	UP
D	4	4	2, 3	DOWN
E	*F*	5	*L*	OPPOSITE
F	*G*	6	5, *E*	UP
G	*H*	7	*L*, *A*	UP
H	8	8	6, 7	DOWN
K	Stays as set			
L	Stays as set			

* Lettered elements note position of their assigned element at sound of "bonk," then assume that position at sound of "bleep," no matter what the assigned element does at "bleep." Numbered elements wait for no external signal, but respond immediately to any change of their assigned element(s).

and so, with four successive "bonk-bleep" signal pairs, any number-representing UP-DOWN pattern in either register will be shifted out the right ends, and some new pattern will be shifted in from the left. For any given replacement cycle like this, the states held by Elements *K* and *L* during the cycle determine what pattern is shifted anew into each register. If *K* is UP, Register *ABCD* ends up with the pattern originally in Register *EFGH*, and if *K* is DOWN, *ABCD* ends with the same pattern it originally held. If *L* is UP, Register *EFGH* ends up with the pattern originally held in Register *ABCD*, and if *L* is DOWN, *EFGH* ends with the same pattern it originally held. By suitably setting *L* and *K*, we can make either one of the original patterns end up in both registers, make the patterns interchange between registers, or make the patterns both return to their original locations.

Since students have already learned that the patterns represent information, they can get a feeling for the possibilities of moving information here and there in a machine. From experience with the adder network, or with further reflection upon this exercise, they can see that *L* and *K* could be elements of other registers, or otherwise be temporary-storage elements whose states are controlled by decision elements and elsewhere-contained information, so that the concept of automatically controlled information transfer gains some substance with them.

Expanded Adder Network—(Fig. 9 and Table V)

Four innovations are incorporated in the exercise described here (innovations to the exercise in Fig. 4), and any one innovation may be introduced by itself. Refer to the distribution of elements in Fig. 9, and their task assignments in Table V.

Saving the augend: The addition of one temporary-storage element to the complement of Fig. 4 allows the original contents of Register *ABCD* to be returned to it, which seems to enrich the demonstration for most laymen: Add Element *L*, assigned to watch Element *A*, and reassign Element *D* to watch Element *L*.

Optional add or subtract: The addition of one decision element in the adder allows the "operator" of this network the option of having his five "bonk-bleep" signal pairs result in either adding or subtracting the number in the *ABCD* register to (or from) the number in the *EFGHJ* register: Introduce Element 10, assigned to watch Element *A*. Its task is different depending upon whether we want to add or subtract. We also give permanent reassignments to Elements 1 and 3, which must each substitute Element 10 for Element *A* in its respective watching assignment. Element *K* is also involved in the operational changes. Here is the procedure then: We enter the desired numbers into the two registers. If we want an addition operation, Element 10 is instructed to duplicate every move of Element *A*, and everything else is done as before. If we want a subtraction opera-

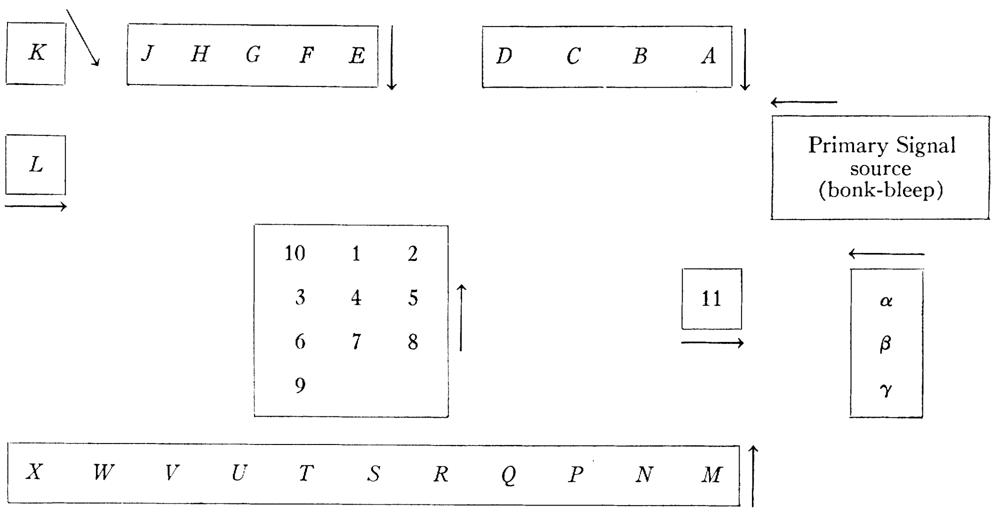

Fig. 9—Distribution of elements for expanded adder network, with simplified tasks for temporary storage (lettered) elements, with automatically metered "bonk-bleep" signals, and with subtraction option.

TABLE V

Task Assignments for the Elements in the Expanded Adder Network of Fig. 9*

Decision Elements			Temporary-Storage Elements			
			"Bonk" changers		"Bleep" changers	
Element	Watches	Task	Element	Watches	Element	Watches
10	*A*	Same, for add Opposite, for subtraction	*M*	*B*	*A*	*M*
1	10, *E*	Up	*N*	*C*	*B*	*N*
2	1, *K*	Up	*P*	*D*	*C*	*P*
3	10, *E*	Down	*Q*	*L*	*D*	*Q*
4	1, *K*	Down	*R*	*F*	*E*	*R*
5	3, 4	Up	*S*	*G*	*F*	*S*
6	3, *K*	Down	*T*	*H*	*G*	*T*
7	5	Opposite	*U*	*J*	*H*	*U*
8	6, 7	Up	*V*	9	*J*	*V*
9	2, 8	Down	*W*	5	*K*	*W*
11	α, γ	Up	*X*	*A*	*L*	*X*

Counting Elements

Element	Responds to
α	"Bleep"
β	α changing from UP to DOWN
γ	β changing from UP to DOWN

* The primary signal source emits "bonk-bleep" signals until Element 11 changes to UP. Each temporary-storage element, when he hears his part of the "bonk-bleep" signal, is to change immediately to the state held at that time by his assigned or watched element. Each counting element responds by changing state when it gets its assigned signal. A numbered element waits for no signal, but adjusts immediately when its watched element(s) change.

tion, Element 10 is instructed always to do the opposite from what Element *A* does. (Being a numbered element, Element 10 waits for no signals, but adjusts instantly to any change of Element *A*.) Furthermore, for subtraction we ask Element *K* to begin in the UP state, but otherwise we proceed with the five "bonk-bleep" signal pairs in exactly the same manner as before.

If the number in the *ABCD* register is larger than the initial number in the *EFGHJ* register, the subtraction operation will result in the number $32-d$ remaining in the *EFGHJ* register, where $-d$ is the negative difference one would expect to find. I usually artfully avoid this result if I don't have time to discuss it, by making sure that the difference will be a positive number. If there is time, the students can be given a feeling for how this situation can be automatically detected (watch *K*, if it is UP at the end, the answer is right as it is, but if *K* is DOWN it is known that what we see is actually $32-d$), and automatically corrected (*e.g.*, make Elements *E*, *F*, *G*, *H*, *J*, and *K* change state, direct Element 10 to stay clamped to DOWN, and go through another series of five "bonk-bleep" signal pairs, with everyone except

Element 10 responding as usual. This will result in a number in Register *EFGHJ* that we know to be the negative difference, and in Register *ABCD* still holding its original number.)

Simpler temporary-storage elements: As noted earlier in the paper, there seems usually to be more trouble with temporary-storage elements than with any of the other kinds. If enough participants are available, it is easy to get around this by adding more elements so that all can have simpler tasks—in a manner exactly analogous to what is often done with the electronic components. Extra Elements *M* through *X* are added, as shown in Fig. 9, and the watching assignments of all lettered elements are changed, as indicated in Table V. This allows each lettered element to be able to switch to the state its watched element *is* in when the given element receives his particular single signal, rather than listening for two signals and having to remember on the second signal what he saw on the first signal. This eliminates a source of trouble, and also allows more people to participate. It also is a much easier operating system to troubleshoot, in case things do go wrong.

"Bonk-bleep" signal metering: Sometimes, under the pressure of monitoring the behavior of so many elements, it becomes hard for you, the primary signal source, to keep track of how many "bonk-bleep" signal pairs you have emitted. To help in this situation, to give the students a picture of how further automatic control is brought into the picture, and to make use of more participants, you can utilize four more participants to keep track of this for you and automatically stop you when you have emitted the right number of signal pairs.

Elements α, β, and γ are organized into a counting chain, with assignments exactly similar to those elements in the first exercise. They count how many "bleeps" have been emitted. Element 11, assigned to watch Elements α and γ with the UP task, will go to the UP state after the fifth "bonk-bleep" signal pair has been emitted. To get your automatic signal metering, then, you make a point of seeing that Elements α, β, and γ know they are to be all in the DOWN state whenever your arithmetic operation is to begin, and then you continue emitting your "bonks" and your "bleeps" and monitoring everyone's performance until you see Element 11 go to the UP state. Commenting upon the trouble of keeping count of your signals, and then installing this metering system, after the rest of the adder network has been broken in, is something which somehow delights most audiences, as they see you incorporate a digital design for which they already have a background to provide a service they can easily understand.

Discussion

The Editor can report the following experiences using Engelbart-style human computers at the high-school level.

1) In using the binary counter (Engelbart's Fig. 1) it was found that the participants "cheated," *i.e.*, they soon sensed the rhythm of the procedure and all began using the audible source signal to govern their actions. To avoid this the source signal was changed to a touch on the left arm of Element *A*, and so was not observable by the other elements. In other words we observed crosstalk and eliminated it!

2) A simple shift register, with end-around shift, proved to be a very rewarding exercise. Attempts at high speed provoked breakdown, and showed clearly several distinct types of error.

3) One group contained only 15 members. Hence we could not use the serial adder (Engelbart's Fig. 4). We could have shortened the registers, but preferred to reduce the number of logic elements by using an "inequivalence" element that responds by assuming the UP state whenever the two elements it is watching are *DIFFERENT*, and the DOWN state otherwise. This concept proved easy for the students (even though it's not so easy to fabricate as a single electronic element). As a result, four logic elements were saved, and the show went on. The revised assignments for logic elements are given in Table A-I. The assignments for register elements remain unchanged, and are given in Engelbart's Table II. See Engelbart's Second Simulation Exercise for further explanation.

TABLE A-I
ASSIGNMENTS FOR LOGIC ELEMENTS FOR A SERIAL ADDER USING INEQUIVALENCE

Element	Watches	Task
1	*A*, *E*	Inequivalence
2	1, *K*	UP
3	*A*, *E*	UP
5	2, 3	DOWN
9	1, *K*	Inequivalence

4) The serial adder has provoked a reaction among bright high-school students to the effect that we are using an awful lot of machinery and time for a simple process. We then point out, of course, that the registers can be lengthened at will to add larger numbers with little additional equipment, but admit that time certainly is spent. As a counter-example, a parallel adder may be built. The number of elements then required certainly is large, but the time required for addition is small. A design for two stages of a parallel adder using inequivalence is shown in Table A-II. The first (least-significant) and the general (*i*th) stage are tabulated. The last (most-significant) stage, *n*, is like the general (*i*th) stage, except that K_n, the carry-out, should be interpreted as the $(n+1)$th bit of the total sum. The total number of participants required is $7n-3$, *i.e.*, 4, 11, 18, $\cdots$.

TABLE A-II
ASSIGNMENTS FOR ELEMENTS IN AN ENGELBART-STYLE PARALLEL ADDER

Element	Watches	Task	Function
A_1	—	Hold assigned position	Input bit
B_1	—	Hold assigned position	Input bit
K_1	A_1, B_1	UP	Carry out
S_1	A_1, B_1	Inequivalence	Sum bit
A_i*	—	Hold assigned position	Input bit
B_i	—	Hold assigned position	Input bit
P_i	A_i, B:	UP	Logic
N_i	A_i, B_i	Inequivalence	Logic
Q_i	N_i, K_{i-1}	UP	Logic
K_i	P_i, Q_i	DOWN	Carry out
S_i	N_i, K_{i-1}	Inequivalence	Sum bit

* $i=2, 3, \cdots, n$.

In each case we have found it most desirable to have the instructions for each participant typed at the head of his 4 by 6 identification card.

H. E. TOMPKINS
Univ. of New Mexico
Albuquerque, N. M.