A Method of Multiplexing Computers — 1 Dec 1958
Part I — John Griffith

Several years ago, Dr. Gene Amdahl proposed a method of interlacing two instruction sets in the same computer. The scheme was originally proposed for the Stretch machine, but was never used. This scheme, plus a large measure of understanding supplied by Turing's work, has been transformed into a method of controlling two or more computers which are working on the same problem.

This method of controlling multiplexed computers will not work for any problem, but it will work in many cases that are of present day interest. Basically, it provides a means for allowing two computers to cooperate on the solution of a given problem. The technique itself is one that Turing used in his famous "Universal" machine; namely, the technique of using a "scratch pad," or eraseable memory, to make marks on, which serve to keep track of one's place during the solution of the problem.

The effect of this method is to give two computers a little more feeling of togetherness while they compute the answer to the given problem, and this feeling is necessary to cooperation between two brains, as we all know. As the Ladies Home Journal is the magazine of togetherness, so is this writing the memo of togetherness.

Passing now to the particular idea of accomplishing cooperation between two computers, we will first outline an idea and some of the theory behind its operation, then proceed to its execution and ramifications.

The usual method of causing two computers to talk to each other so that one computer may find out what the other is doing is to use binary triggers which may be set or interrogated by either machine. These triggers, or selectors, as they are usually called, are so arranged that some particular piece of information is assigned to each one at some particular time, and the process of interrogating and setting by either machine may be taken to be completely general. The only trouble is that this method does not lend itself easily to the real time aspect of cooperation between machines, and thus, it is usually necessary for the computers involved to measure time in some manner such that the information conveyed by the condition of the selectors may be superimposed upon the demands of the programs being executed.

The idea herein presented varies from the selector approach by replacing selectors, which can take on either of two values, by counters, which can take on any of many values. An important distinction between the two is that the counter method will allow the measure of relative time, as defined by two

machine programs. It is difficult for selectors to measure time, or anything else that cannot be easily identified with two values. Let us now proceed to a simple example of this idea.

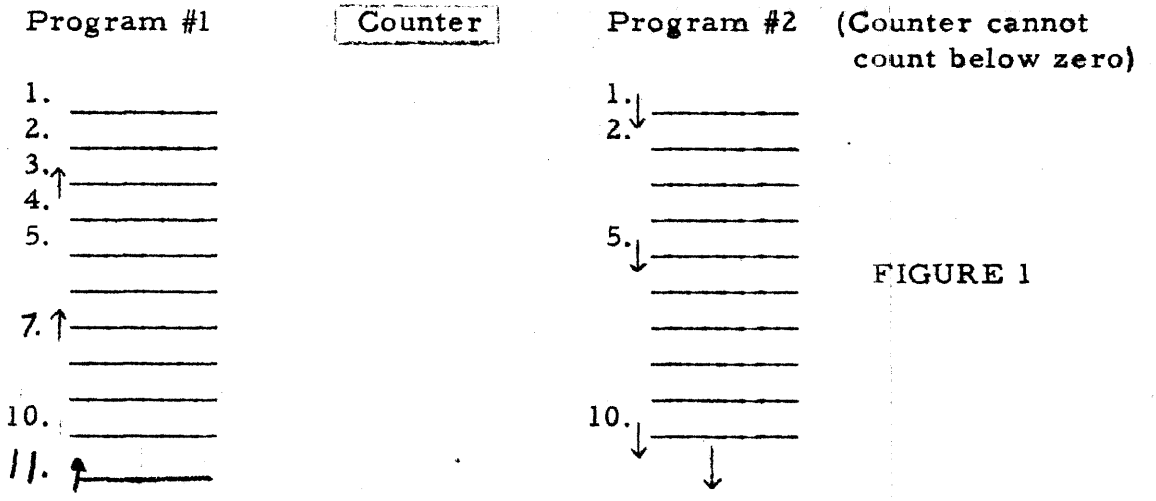

FIGURE 1

Figure 1 shows the idea in its simplest form. We may assume that we have two computers operating on the same data in the same memory box. Furthermore, the program in machine #1 must operate on the data before the program in machine #2 gets to it. The two machines will be distinguished by the Program 1 & 2 notation in the Figure. Suppose that, in addition, a counter is connected between the two machines in such a manner that it is counted up by one whenever a certain bit occurs in an instruction in Program #1. Suppose also that the same counter is counted down by one whenever a certain bit (reserved for this purpose, as in machine #1) occurs in an instruction in machine #2. Suppose, also, that this counter is constructed so that it cannot count below zero; in other words, it can contain only positive numbers. We now wish to make certain that machine #2 can never get ahead of machine #1 in the execution of the program in machine #2.

This will be done by observing those places in the two programs where it is absolutely necessary to prevent machine #2 running ahead of machine #1. (We will assume in this example that there is no limit to the extent which machine #1 can precede machine #2). Wherever such places occur, we mark the corresponding instructions with the special bit reserved for this purpose. In the example shown, Program #1 has instructions 3, 7, and 11 marked. Program #2 has instructions 2, 5, and 10 marked. An arrow is used here for the mark to indicate the direction of counting. We may now say that the requirements for this example are that instruction 3 in Program #1 must be executed before instruction 1 in Program #2 is executed. The same is true for instruction 7 in Program #1 and instruction 5 in Program #2; likewise for instruction 11 in Program #1 and instruction 10 in Program #2.

It will be seen that in the execution of Program #1 the counter will be counted up by one whenever a marked instruction is encountered and the counter will be counted down by one whenever a marked instruction is encountered in Program #2. Suppose that we were to start up the two machines simultaneously with the two given programs in their respective memories. The value in the counter is zero. At this point, machine #2 would do nothing because the counter cannot be counted below zero, but there is nothing that would prevent machine #1 from proceeding normally. As soon as machine #1 has executed instruction 3, the counter would be counted up by one, thereby allowing machine #2 to count it down to zero. This act will allow machine #2 to proceed until it encounters instruction 5. If, at this time, machine has executed instruction 7, the counter will register a count greater than zero and machine #2 will count it down as it proceeds. If, when machine #2 arrives at instruction 5, the counter is still at zero, machine #2 will stop and wait for the counter to be counted up by one.

If machine #1 proceeds much faster than machine #2, the counter will contain a value higher than one. This value is somewhat of a measure of how far machine #2 lags behind machine #1. This is what was meant in the previous reference to the ability of a counter measuring the relative time between the execution of two machine programs. Notice that in the example given, machine #1 is a free running machine and machine #2 is free running until it threatens to pass up machine #1. The coupling between the two machines is completely specified by the marks on the appropriate instructions, and these marks allow any degree of coupling desired. Notice also that if this idea were to be carried out with a selector instead of a counter, it would be impossible for machine #1 to precede machine #2 by any great amount. Thus, the value of the selector is limited because it cannot contain a variable measure of the lag between the two machines.

At this point, it may be well to indicate the obvious analogy with conversation between machines. It can be seen that the use of selectors in this example are nothing more than the narrow bandwidth communication channel between machines. The counter is nothing more than a wider bandpass channel, and thus, it is not surprising that it is able to provide better communication between the machines.

We will now proceed to another example, this one a variation of the previous example. This example will be concerned with three machines working on the same problem, but otherwise like the first example. Figure 2 illustrates this variation.

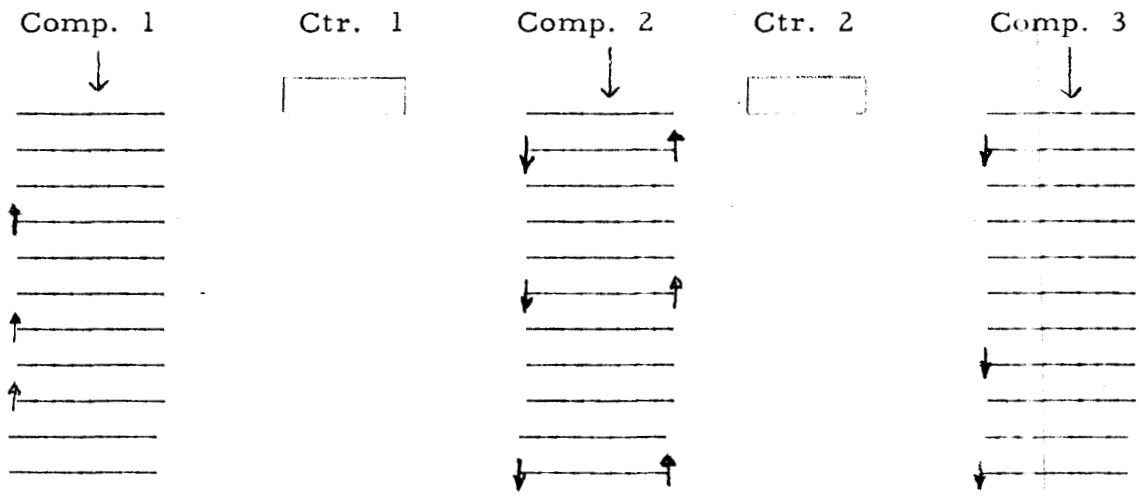

FIGURE 2

This time we have three computers working in sequence on the same data. Computers #1 and #2 operate as before, and the addition of Counter 2 allows Computer #3 to work on the data after Computer #2 is finished with it. Counter #2 is counted up by marked instructions in Computer #2 and is counted down by marked instructions in Computer #3. In addition, Counter #2 can only be counted up as in the case of Counter #1. Thus, Counter #2 prevents Computer #3 from preceding Computer #2 in exactly the same manner as Counter #1 prevents Computer #2 from preceding Computer #1 in previous example. It may be seen that a marked instruction in Computer #2 affects two counters; such an instruction causes Counter #1 to be counted down at the same time it causes Counter #2 to be counted up. By extension, it may be seen that this idea may be extended to any number of computers working on the same problem if the conditions of the problem are as described in Example #1. Next, we will take up a variation introduced by a more complicated and more realistic example.

This example concerns the recursive nature of programs. Figure 3 illustrates the situation.

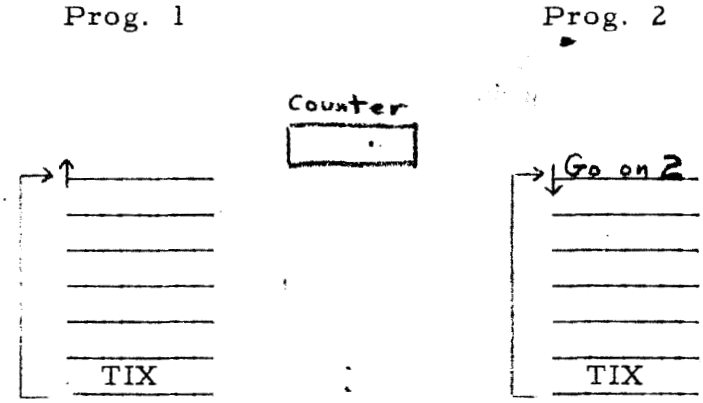

FIGURE 3

The complication included here is that the two programs are formed as loops and it is desired to control the two machines so that Computer #2 follows Computer #1 by one execution of the loop in Computer #1. In other words, we desire that Computer #2 work on the data used by Computer #1 only after Computer #1 is through with it. There is nothing new about this requirement, but the fact that the programs are in the form of loops and not open-ended sets of instructions puts an additional requirement on the counter system. This new requirement comes about in the starting procedure.

If, in this example, the two computers were already running, the situation would not be greatly different from that of Example #1. However, in order to get the situation displayed in Figure 3 going from scratch, it is necessary to introduce a special command. This command is indicated at the beginning of Program #2. The command is "Go on 2." This command is obviously a variation of the indicator bit which occurs on the instructions themselves. Except that this command affects the time at which the succeeding program steps will be allowed to proceed. In this case, the "Go on 2" command will not allow Computer #2 to proceed until the count of 2 (or other specified value) appears in the Counter. When the specified count is reached, the Counter is counted down by the amount of the count, and by this means the lag between Computers 1 and 2 is maintained at one loop execution. If the value specified by the instruction in Computer 2 were 6, the relative lag between the two machines would be five loop executions.

Clearly, this command is nothing more than a bias mechanism which introduces a measure of control on the contents of the counter. In the last case, where the "Go" instruction specified a value of 6, Computer 1 would be working on the sixth execution of its loop while Computer 2 would be working on the first execution of its loop; thus we say the lag is five, the difference between the cycles of execution of the two computers. It is also clear that the lag between the two machines can never be less than five, although it may be as much more as necessary, and would be limited, in this example, only by the size of the counter.

Another way of accomplishing the same thing would be to have the value specified by the "Go" command retained by the Counter in such a manner that the lowest possible value which would be allowed would be the value specified, in the last case, 6, until another instruction were encountered which changed the value up or down. This method of introducing bias level on the counter may be better, for it allows one to manipulate the counter contents in a more general fashion by allowing the programmer to set the virtual zero of the system at any real value of the counter he pleases.

When we mention the possibility of manipulating the contents of the Counter, we may also guess that we will wish to add to our system commands which increase and decrease the contents of the Counter independently of any other action taken by the Computers. We will likewise want the ability to compare, test, and read out the contents of the Counter as if it were any other register of the system. At this point, the reader may wonder what is different about the Counter if all of these things may be done to its contents. The answer is nothing, of course. The Counter merely serves as a scratch pad so that one computer may know where it is relative to its precedent computer.

We will now pass on to Example 4, which is the extension of Example 3 to the case of three or more machines. The conditions of the problem are the same, and we wish to give nothing more than the illustration of the starting procedure for this case.

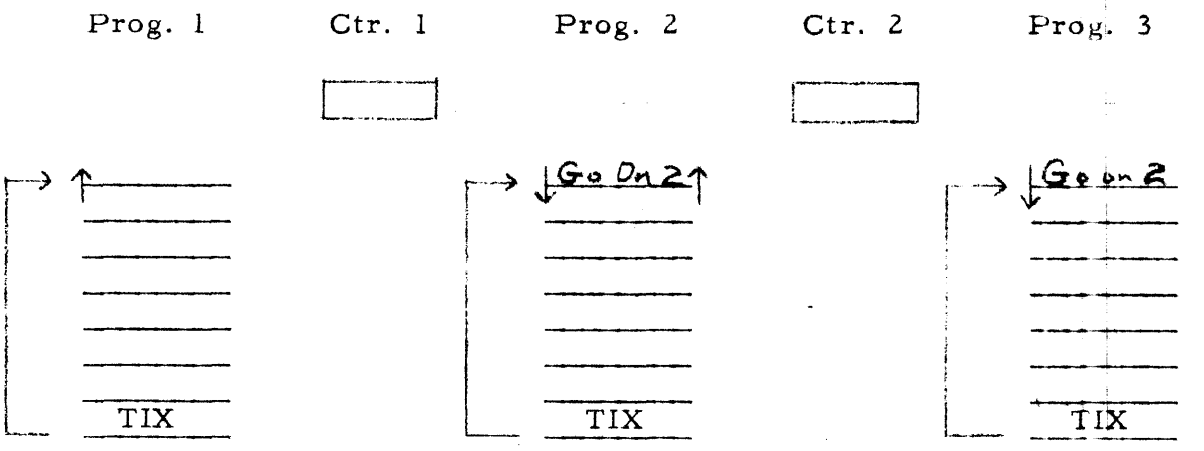

FIGURE 4

When Computer 1 starts, it will cause Counter 1 to be counted up by one. The "Go" instruction at the start of the routine in Computer 2 will prevent Computer 2 from starting until the count in Counter 1 is two. When Computer 1 begins the second pass of its loop, the contents of Counter 1 will be counted up by one, making the total two. The "Go on 2" command in Computer 2 will therefore subtract two from the contents of Counter 1 and allow the execution of the program in Computer 2 to proceed. At the same time this action occurs, Computer 2 will cause Counter 2 to be counted up by one. When Computer 2 starts its second pass, the contents of Counter 2 will be counted up by one, making the total two. Computer 3 will therefore start up, and at the same time, two will be subtracted from the contents of Counter 2. At this time also, some indication must be made, either in the instruction itself, or in hardware that the "Go" instructions are to be executed as "No Op" until the indication has been reset. The counting function of the "Go" instructions will, however, continue (as usual for any

other instruction). The net result, will be that the three computers will be executing their respective programs (loops) with a lag of at least one between each machine's execution cycle. Note also, that it is quite impossible for the machines to overrun each other, for the zero count in either counter will hold up the rest of the chain until it is safe to proceed.

For the example shown, it is also clear that as many computers may work on the data as is possible, if a counter is inserted between each pair in the chain. It should also be noted that the lag between any two machines may be set at any value, or number of loop executions, that is desired. The bias, mentioned above, will remain at the lowest value set, and this will effectively prevent the various loops being executed in the various machines from getting too close to each other. A practical example of the case chosen here is the usual I/O operation where one wishes to read from a tape, operate on the data, and write the updated data back out on a new tape without worrying about synchronizing the program loops. There is more to be said about such an example, but we will defer it for a later memo.

Before we leave this example, we will discuss a variation on the hardware logic used. The reader may have noticed that the values given in the "Go" instructions were relative to the Computer cycle preceding the given machine. One may ask, is this the best way? Why not use an absolute value system starting with the first machine in such a chain? There is no way to determine the best way, for it depends on the exact nature of the problem being solved. In any case, the absolute system may also be implemented, and we will refer to it in Example 5.

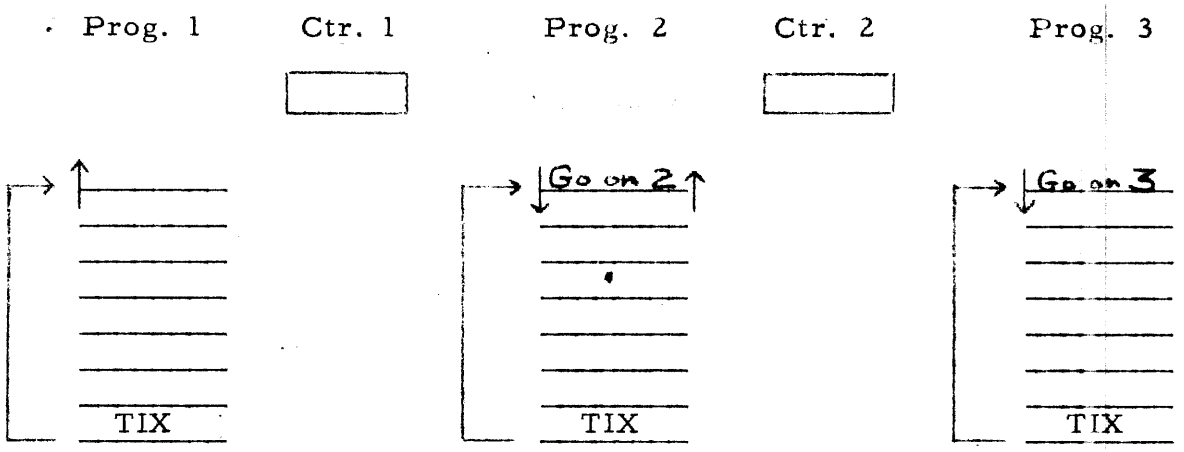

FIGURE 5

It will be noticed that the only change has been to change "Go on 2" in Computer 3 to "Go on 3." The hardware works somewhat different from that of Example 4. At the beginning, when Computer 1 starts, Counter 1

is counted up by one. Computer 2, which is waiting, immediately counts Counter 1 down by one and Counter 2 up by one. When Computer 1 begins its second pass, Counter 1 is counted up by one. Computer 2, which is still waiting, immediately counts Counter 1 down by one and Counter 2 up by one. At this time, since Counter 1 has been counted up and down a total of two, Computer 2 proceeds. When Computer 1 has begun its third pass, it will count Counter 1 up by one, and when Computer 2 begins its second pass, it will count Counter 1 down by one and Counter 2 up by one. At this time, a total of three will have been counted up in Counter 2, and Computer 3 will proceed. If the computers are to maintain the given relationship, Counter 1 must not be counted below one, and Counter 2 must not be counted below two. This implies that some additional hardware would be needed to establish and hold bias levels in the counters. However, this alternative may be justifiable for some problems.

One might also notice a slight difference between the operation of the two alternatives illustrated in Examples 4 and 5. In the case of Example 4, each machine must wait until its precedent had started before it could start. In Example 5, each time a machine starts up, there is an immediate rippling of the count across all the counters in the system. This implies that any computer in the chain may start up on any cycle after the initial Computer has begun its cycling; however, it is still necessary that the machines execute their cycles after the preceding Computer, therefore, the effect is still the same as if the values used were relative to the precedent Computer instead of the starting Computer. However, there is a variation of this arrangement which is somewhat more flexible and powerful, and this variation requires the absolute, or relative to the starting Computer, values to be used in the Counters. At this point, however, the excursion would be too far from our present line of thought, and it must wait for a later discussion.

To summarize briefly, Examples 4 and 5 illustrate the logic of starting and maintaining a minimum lag between Computer execution cycles for the case of simple recursive programs in each machine and for the case of more than two computers in the chain. A variation in the method of specifying lag was covered: there are two ways of specifying the lag, either specifying it relative to the precedent computer (Example 4) or specifying it relative to the starting, or first computer in the chain (Example 5).

We will now extend the line of thought indicated by these two examples to a more complicated case. This case has as its salient feature a higher level of recursiveness than either of the two preceding examples. We may take the case shown in Example 5 with the additional requirement that the programs in each of the computers have both inner and outer loops.

In particular, the loops pass over the data used by the precedent computer. The important change in this example is that the chain of computers is a closed loop, not an open loop, as in all of the previous examples. In other words, Computer 1, in this example, will have Computer 3 as its precedent, once started, and the effect will be that of a closed loop of computers operating in order on a closed loop of data. In the previous examples, the chain of computers was an open loop of machines operating on an open loop of data. In this example, we have the additional condition that Computer 1, when it arrives at the end of the data, will then start over on the same data, but we must provide means for preventing Computer 1 from operating on the data before Computer 3 is through with it. This example is somewhat more realistic, than some of the past examples, as the reader will realize, for when Computer 1 starts the second pass on the data, the data may have been replaced, with new data, and it is still necessary to make sure that Computer 1 does not overtake Computer 3. In any case, the process described is a common one, although the exact treatment of the data between Computer 3 and Computer 1 passes may vary. We may easily modify this example to take into account the replacement of the data between passes by Computer 3 and Computer 1, by proposing a fourth Computer to be inserted between Computer 3 and Computer 1 whose function is to replace the data after Computer 3 is through with it. But input-output discussions will be held for a later memo, as promised above.

Another aside has been introduced by the previous sentence. The reader may have noticed that the theory underlying the idea presented in this memo may be illustrated by imagining that a complete computer is available to solve each of the various parts of a problem to be done. For instance, we may propose whole computers assigned to the function actually taken by a subroutine. The problem is then: how does one connect the various computers so that the problem will be solved correctly? The answer to this question cannot be given in this memo, as it is probably an unsolvable problem. Another question: Can a complex of computers be devised that will solve a given problem? The answer to this is yes, as long as the complex satisfies certain conditions. But we are drifting away from the example at hand, and must defer questions of this sort until later.

To return at last to the example promised, Example 6, we see in the diagram that an extra Counter has been added.

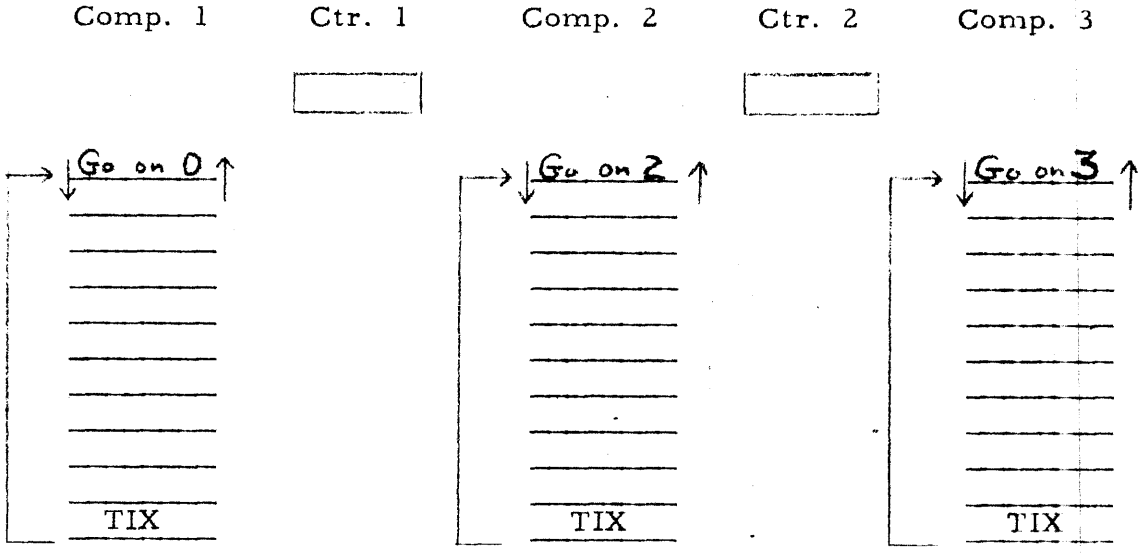

FIGURE 6

This example assumes the same conditions as before with the Counters retaining values (lags) in the "absolute" system or referred to Computer 1 as the origin. It should be noticed that Computer 1 now counts Counter 3 down as it counts Counter 1 up. This feature closes the loop of machines, and allows the whole chain to operate on data in a recursive fashion. The starting commands operate as usual, except that "Go on 0" in Computer 1 refers to the contents of Counter 3. In this Example, the contents of the Counters are held as bias levels or lags between respective machine cycles. Thus, when Computer 1 starts its second pass, Computer 2 will start its first pass. When Computer 1 starts its third pass, Computer 2 will start its second pass, and Computer 3 will start its first pass. This will get the chain started, and as Computer makes its passes over the data Counter 3 will be counted up. When Computer 1 gets ready to operate on the data finished by Computer 3, Computer 1 will count down Counter 3 as it does so. Notice that Computer 1 may not start its fourth pass until Computer 3 has finished its first pass. This is an interpretation of the present Example, for we assume, for simplicity, that there are only three batches of data to be worked on, and that the order for each machine will be batches 1, 2, 3, 1, 2, 3 . . . etc. Thus, Counter 3 interlocks the chain such that Computer 1 cannot start its second pass over data batch 1 until Computer 3 has finished its first pass over the same batch. If there are more than three batches of data, ten batches for instance, the lag between Computer 3 and Computer 1 will be large, but in any case, Computer 1 will never be able to overrun Computer 3. When all passes have been made through all the data, the exit from each of the loops must provide for one last count which will allow the following machines to pass over the same data.

At this point, we have introduced the basic method by which a complex of computers may be brought to bear on a given problem if the problem satisfies certain conditions. The principal condition that such a problem must satisfy is that it must be possible to solve the problem by dividing up the work of the solution such that the various parts follow each other in a fixed sequence. If this is possible, then a computer may be assigned to compute each part of the work, and the sequence of the computers will be the same as that of the various parts of the work. Therefore, in order to guarantee a correct solution, it is only necessary to make sure that succeeding computers do not overrun each other. It is not necessary to keep the lag between machines to a minimum, but it is desirable to do so. The method outlined in this writing will not minimize the time required for a solution, but it will guarantee the correctness of the solution.

If one has a problem whose solution cannot be broken into a sequence of parts, there is an extension of the method described here that will relax certain of the requirements imposed here, but it still will not allow the general case to be solved. This variation will be covered in Part II, along with a discussion of multiplexed Input/Output considerations.