A Method of Multiplexing Computers February 13, 1959

Part II - I/O Operations John Griffith

In this part of the memorandum we will, as promised in Part I, discuss some techniques that are applicable to multiplexing Input/Output. It is assumed that the reader has read Part I of this memorandum.

The standard example of I/O is the use of tape, and the example discussed first is that of automatizing the usual tape operation one level beyond that usually provided by the use of control words.

Example 7, shown in Figure 7, is supposed to represent the case of a computer with one tape unit for input, and we will show the application of the method discussed to completely automate the operation of the tape. It is assumed that the computer used is equipped with the control word method of controlling input/output. An example of what is meant here is the STRETCH machine's control word.

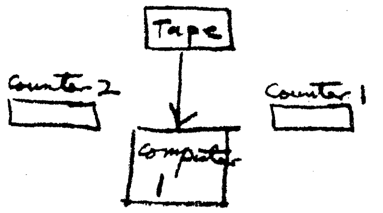

Fig. 7

Notice that another register has been added to the system usually used in Part I. This register is controlled by the computer <u>and</u> the tape unit. The operation of this register is independent of the operation of Counter 1, except for thosesituations where the program in the computer makes the operation of the two dependent. Counter 2, in this example, is controlled by a bit in the instruction much as was the case for Counters in Part I of this memorandum. However, the bit which controls this counter is not the same bit as the bit discussed in Part I. In this case, the bit is another bit reserved for this purpose; in addition, this bit causes Counter 2 to count up by one whenever the memory is "read" by the instruction in which it is situated. As soon as Counter 2 has been counted up by one, the tape unit which it controls immediately starts up and as each word is written into the memory, Counter 2 is counted down by one. The Control Word associated with that tape specifies the addresses and word counts concerned as is the case in machines that use control words to control I/O.

It will immediately be seen that this operation assumes a very simplified form of operation on the part of the program, for, if the program should not Read Out the data as fast as the tape writes it into memory, there is danger that the tape will over-run the program. Actually, of course, this cannot happen. As soon as the count in Counter 2 is reduced to zero, the tape unit disconnects until Counter 2 is counted up again. But the disconnected tape will continue to move until the end of the record being read in is reached, and at this point, it is impossible to return to the place where reading was suspended without a rather complicated procedure.

Therefore, a modification is needed. Actually, the count in Counter 2 is preferably that of a single record, or at least the amount information signified by the next control word. In this case, only when the program had consumed one batch of data would Counter 2 be counted, and this would then start up the tape and it would write into memory the number of words specified by the next control word. If the contents of more than one control word were needed, then Counter 2 should be counted up by that amount, that is, the total number of control words to be exercised.

If the situation is one where there is not enough information to set up control words in advance, then the method first discussed must be resorted to, or a dynamic program must be used which will simulate the system described in some equivalent manner. "Some equivalent manner" means that the simulated system is nothing more or less than what is done at present in situations like that described. The point here is that a system of markers must be set up so that the program is able to keep track of its place as it unravels the problem of what to read off tape and when to stop the tape drive. The counter system will accomplish this when it is known in advance where to place the marks on the instructions. If enough is not known to make the marks on the instructions, then the dynamic method (the present day method) must be used, and the routine must decide when to make "marks" which will guide it as it proceeds to the solution of the tape problem. In any case, and this is Turing's great contribution to our understanding, of this problem, it is necessary to make extraneous marks to guide the solution of the problem, and if a method of making marks and erasing them cannot be worked out, the problem may not be solvable under any circumstances.

The obvious variation on the method presented here is that of allowing the programmer to specify exactly what the count in Counter 2 signifies. For instance, suppose that the count in Counter 2 were the count of record blocks; this would imply that the count would be dependent on something other than the "Read" reference to memory. If the programmer is to specify what the meaning of the contents of Counter 2 are, then all that is necessary is that the count be made on the basis of a bit in an instruction, whether or not it has anything to do with the operation of the memory. The first case was presented for the sake of generality, but the variation being discussed here is, in this person's opinion, the most practical version.

Now, if the contents of Counter 2 specify the number of blocks, for instance, that the tape lags behind the operating program, ~~there~~ it is possible [illegible] to program the job without worrying about sampling the status of the tape unit at certain times in order to keep it going. In addition, the multiplexing of several tape units can be accomplished in an easier fashion, for if each is equipped with a counter such as is described here, it is no problem to equate the mechanical priority with the priority established by the program. Since no tape unit may get ahead of the requirements of the program, it is only necessary for the program to decide what comes next, which is equivalent to assigning a priority to the various units that may be connected at any given time.

At this point, we may as well discuss the problem of two computers which are processing information from the same tape unit. This is practical in those cases where the tape may contain records in both low activity and high activity ratios. While the low activity part of the file is being passed, Computer 1 can easily handle the work. When a high activity part of the file presents itself, Computer 1 cannot handle the load, and there will be danger of tape disconnects. In this situation, we need a modification of Counter 2 such that if it is counted down to zero, the tape connection is made to Computer 2, causing that system to start up, via a Trap Mode, and process the records that are coming off tape at the time. This is illustrated in Figure 8:

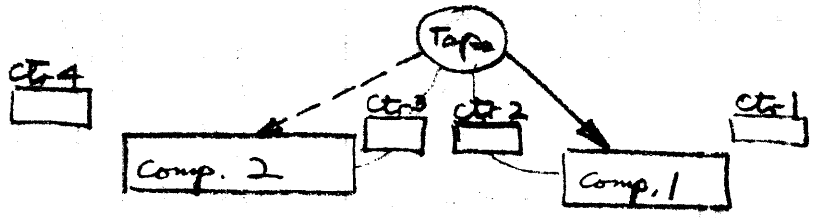

Fig 8

Notice that Computer 1, with Counters 1 and 2, is the same as in Example 7. In this Example, Example 8, the addition of Computer 2 and Counter 3, corresponding to Counter 2, now allows a stand-by machine to be inserted into the system; Counter 4 is shown in case it is desired to allow Computer 2 to work on another problem, say a large production problem, and be interrupted for a short time to help Computer 1. Counter 4 will contain the marks made by any other computer that may have been preceding Computer 2. It is thus seen that what is happening here is that Computer 2 is actually a part of another computer complex, or system, and if it is ~~the last machine~~ in a chain, it is possible to borrow it for short periods of time to help out in another computer system. For simplicity of drawing, the I/O and any other counters that might be associated with Computer 2 are not shown, but it is assumed that they are in existence, in any case.

This example has introduced another side discussion, but this is not the place to continue it, and like many other points brought up in this memorandum, it will have to be deferred until later. The point introduced in this example is that of Interrupt conditions brought on by reading zero counts in the Counters. This feature is more necessary when considering more complicated examples of computer nets, and it will be deferred until Part III for that reason, nevertheless, the reader can easily see for himself that it is a desirable feature in this Example, for it allows Computer 2 to be interrupted at any marked point in its program, and it also ~~eliminates~~ ameliorates the problems of memory assignment that plague the people now attempting to write supervisory programs to do just this type of operation. In this example, when Computer 2 sees that Counter 3 is counted down to zero, it will then know that Counter 2 has been counted up at least by one, and the Trap Mode will be suspended so that Computer 2 may return to the problem that it was previously working on. The contents of Counter 4 will indicate where Computer 2 is with respect to any precedent machine, if such exists. If Computer 2 was working alone, it will merely return to the previous status that it had, and if any I/O equipment was disconnected as a result of the interruption, restart procedures will have to be initiated before it returns to the problem.

To summarize briefly, we have introduced the following new features:

a) counters reserved for I/O operations

b) extra bit in instructions for I/O counter use

c) Trap and Interrupt procedures based on zero count

The reader will notice that many other operations may also be made dependent on the contents of the counters, and that many variations of these features may also be proposed. Leaving some of these as an exercise for the reader, we will pass on to the next example.

The reader will have noticed that the two previous examples talked about "I/O" operations, but included only input operations in the examples. The extension to include output operations is in fact very easy, and we will pause here and include it for the record. Example 9, shown in Figure 9, is the same as Example 7 except that it includes both an input and an output tape.

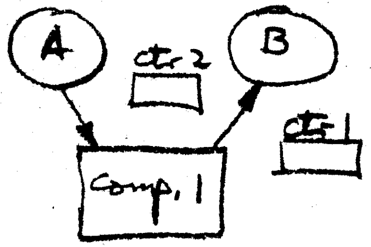

Fig 9

In this example, Counter 2 now has been modified so that it's operation is controlled by the conjoint operation of the input tape, A, and the output tape, B. We assume in this example, that the job is more or less of simple updating of records and that the input and output records are always the same length. In this case, whenever an output block is written by the program, an input block is automatically called in from the input tape under control of Counter 2. This is done by causing Counter 2 to count up by one whenever a block is written on the output tape, and the operation of the input tape will cause Counter 2 to be counted down by one. In this case, the program will have to get the situation between the two tapes started by a start procedure similar to the start procedure used in Part ~~X~~ I of this memorandum. In this case, it would likely consist of the program calling a few blocks off the tape without any counter action. This, then will get the system started and the action will be automatic until the last blocks are written on the output tape, which will also take a special procedure. This example is admittedly very simple, but the reader will see the function of the counter in this case.

A more complicated version of the same example will be now discussed as example 10, shown in Figure 10:

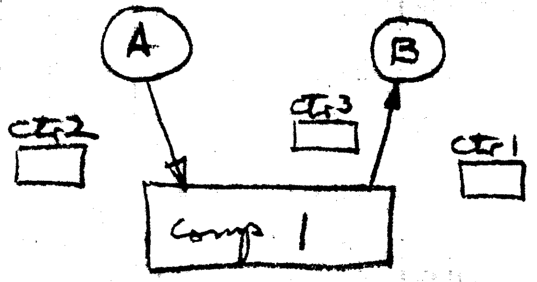

Fig. 10

This example is like the previous one, except that the input and output blocks do not have to be the same length nor do they have to be the same in number. Since each tape unit must have its own table of control words stored in memory, the use of two counters in addition to Counter 1, used as always, is necessary to provide a larger degree of decoupling between the action of the two tape units. This example provides for a greater degree of independence between input and output operations, and in fact, it provides for the greatest amount of independence that can be used. At this point, the two tape units do not have to have any specific logical functions at all and one may simply refer to them as Tape A and Tape B without specific reference to their function. For I/O purposes, however, we wish to Read Tape A and Write on Tape B. Therefore, when we select Tape A with a Read operation, Counter 2 will be counted up, and when the block is written into memory, Counter 2 will be counted down. When Tape B is selected with a Write operation, Counter 3 will be counted up, and when the block is written on Tape B, Counter 3 will be counted down. Notice that the action of the counters is the same in either operation. This is desirable, for it allows the program to specify the tape function without having to recognize the different counter function that may be implied. The use of control words here allows great flexibility in I/O operations, and the reader will have noticed that the use of these counters will be seriously hampered in any machine that does not use control word logic in I/O operations.

To make this example reflect a real situation, we will now consider a modification of it, suggested by Miss Elaine Boehm, which incorporates "stand-by" tape units for some of the input and output files. This example, shown in Figure 11 and to be known as Example 11, will illustrate another feature that is desirable to have when I/O operations are being controlled by counters.

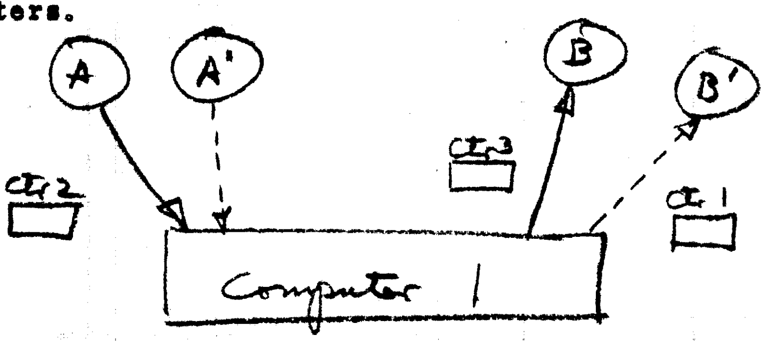

Fig. 11

This example is the same one as shown in Figure 10, except that a stand-by tape unit has been added to the input and likewise for the output. In this example, the input is taken from Tape A until the end of the reel is reached; at this time, the program will switch to Tape A' as input while Tape A is being reloaded by the operator. The same goes for output Tapes B and B'. The important point to notice is that there is only one counter for the Tapes A and A'. Thus, when the switch is made from Tape A to Tape A', the use of Counter 2 is also switched to operate between the program and A'. One could just as well have a separate counter for each Tape, but the point at which the switch is made requires that the program transfer the information from the counter associated with Tape A to the counter associated with Tape A'. This is becuase the information in the counter has been placed there by the program, and, therefore, it is information which is significant to the program's control of the tape units. Since there is no logical difference between the two tape units that is recognized by the program, it behooves the counter to be available on the basis of a channel assignment, rather than a tape unit assignment.

The same sort of remarks apply to the B Tapes, and it is clear that the counters in an example of this type will be better used if they are assignable on a logical basis, rather than an absolute basis.

Example 12, the next example, is a further modification of Example 11:

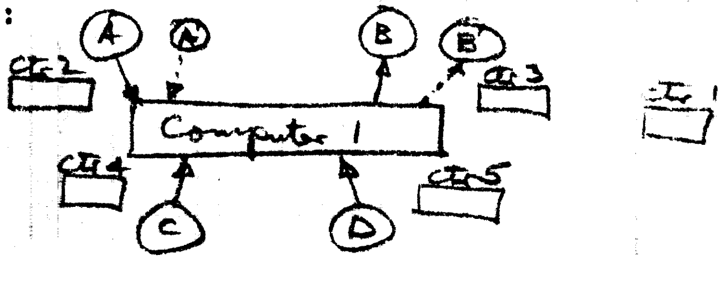

Fig 12

This example includes two more tape units with associated counters. The point of this example is to illustrate a further variation on the use of counters in I/O situations. Let us suppose that Tapes A and A' are master file tapes, Tapes B and B' are output report tapes, Tapes C and D are detail or other report tapes. The problem being considered is that of preparing a new report on the basis of matching records from tapes C and D against Master file records. It is probably more convenient to use as counts in the various counters blocks of data from the various tapes. Notice that the blocks do not have to be the same size on all tapes, for the counters merely represent, at any given instant, the number of blocks that remain to be read in or written out in order for any given tape to catch up to the processing program. If the operation proceeds fast enough, all tapes will be kept rolling continuously. If any of the counters reach zero, this will cause the computer to trap and at this time the supervisory routing can examine the status of the I/O unit indicated in order to decide what to do. It may be that the processing speed is too slow or too fast. In any case, the supervisory routine can at this time make a decision as to what to do. In case the tape has stopped because of a bad spot, necessitating a re-read and bad spot procedure, the other units might come to a halt. But this fact will be evident to the supervisory routine and the Trap can be ignored, if it pertains to the other tape units. In order to restart the whole operation, it is only necessary for the supervisory routine to restart the tape units and proceed as usual.

It is possible, but not included in this example, to call in another computer to help out, as illustrated in example 8; this would be one use of the Trap Mode on reaching zero in certain of the counters. In some cases, if more processing speed is needed, the supervisory routine will be able to decide that such is the case, especially in the case of break-in by Counter 2, some of the records may be shunted to the second computer, under control of another counter, as shown in Example 8, but this complication leads to a discussion best left to the reader to figure out for himself, and thus we will happily leave this example with the reminder that we have introduced Trapping on zero counts for supervisory control purposes. Clearly, this is a slight variation on the Trap introduced in Example 8, but a very necessary variation, nevertheless.

At this point, we now remind the reader that we promised in the last paragraph of Part I to say more about problems whose solutions cannot be broken into a sequence of operations. What was meant there was problems such as multiplexed I/O operations, whose solution is not known in advance. This memorandum has taken several examples of I/O operations and shown the approach that might be used to solve the problems when counters are available. There are other problems whose solution cannot be prescribed in advance and real time problems are the most common example. Since I/O operations are nothing more than relatively simple examples of real time problems, we will say little more about them in this memorandum. Miss Elaine Boehm has kindly consented to work out a more professional example of I/O operations under full multiplexed conditions and we refer the reader to her memorandum, which should be forthcoming in due time.

When, in the last paragraph of Part I, we mentioned relaxing certain of the conditions that were taken in examples of Part I, we meant that the use of Trapping and Interrrupt on various conditions in the Counters would often allow a program to find its way out of a maze. We have in this part of this memorandum, discussed Traps and Interrupts which occur when a Counter is counted down to zero during the execution of a program. In Part I, it was assumed that stopping the machine involved was all that was required. This was true, and still is, for the cases taken. However, when one introduced real time considerations, one is then faced with the problem of keeping track of what is going on while the program is being executed. Counters that cause some sort of break-in action are, therefore, desired, and break-in actions that are caused by being counted down to zero allow programs to be interrupted at significant, non-arbitrary places where control is $\overset{more}{\wedge}$ easily exercised. Since the counters are, for the most part, controlled by the program, it is reasonable to believe that the program will have a much easier time of unraveling the problem posed by a zero count than the problem posed by, for example, controlling two computers without any counters between them.

We will pass now to a special gimmick that may be accomplished with counters. This example concerns a method of causing two tables of control words to be automatically merged as they are executed. This operation is most commonly found in sorting routines, and its operation will be most obvious to those familiar with the techniques usually found in sorting routines.

Figure 13, and we shall denote it as example 13, shows the situation assumed:

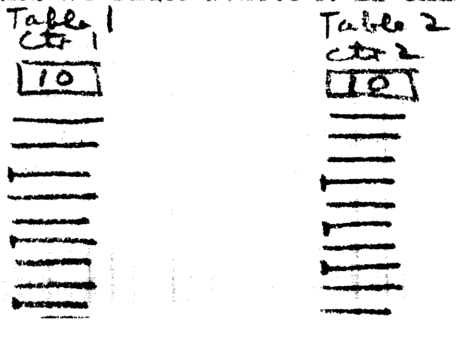

Fig 13

A Method of Multiplexing Computers

In this example, we assume that we have two tables of control words, each control word representing a block of information to be written on an output tape. In addition, we assume that the control words in each table have been sorted in order within the table. In the illustration, each table contains ten control words, and the counter associated with each table contains the number of words in the table. Now, the merge routine must pass through the two tables, and indicate the breaks in sequence that occur in each table. In the figure, we may say that the first three control words in Table 1 are high, then the first four words in Table 2 are next. The break-in sequence is denoted by a mark in the third control word in Table 1. The next break in sequence is at word #4 in Table 2 and it is also marked by a bit in the control word. The next break is shown in control word #6 in Table 1, and so on. Once the breaks have been marked, the output tape is selected and the operation begins.

Beginning in Table 1, as each control word is executed, the Counter corresponding to the table in which the control word is located is counted down by one. When a marked control word is encountered, the next control word is taken from the opposite table and from the position noted in the counter. Thus, the control words are executed one by one in a sequence controlled by the marks and the counter contents.

But one of the tables will be exhausted before the other. This will be indicated by the zero count being reached in the appropriate counter. When the zero count is reached, this causes an immediate trap to a supervisory routine which then causes the next table of control words to be moved into the table area. The merge routine then merges this new table with the remaining control words in the other table and fires the tape up again. Whichever counter reaches zero first, will cause another breaking and the procedure will be repeated by the supervisory routine for a new table of control words, etc. Miss Boehm has suggested that a more realistic case would involve two areas for each table of ten control words each. This allows a third area to be used as input and one table area can be filled while another is being emptied. Figure 14, example 14, illustrates this;

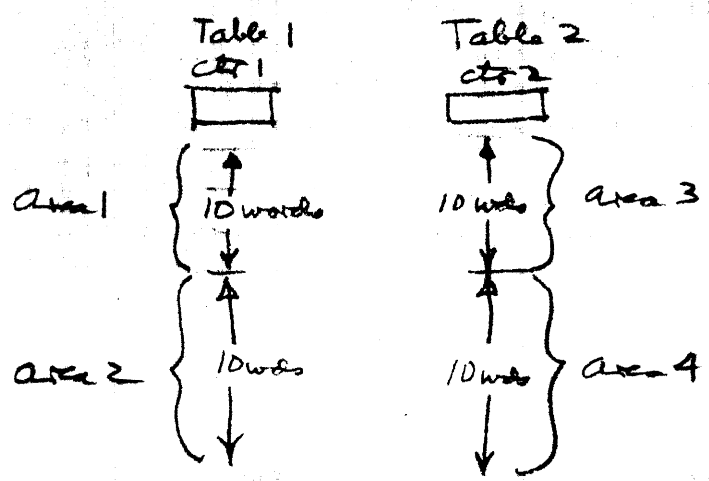

Fig. 14

[illegible] The counters have been added to make the operation self-sufficient as an I/O operation.

The operation will start by merging tables from areas 1 and 3. If area 1 is exhausted first, area 2 is then marked and merged against the remains of area 3. While area 2 is being used, area 1 is being refilled. If area 2 is exhausted before area 3, area 1 will come into play again. When area 3 is finally exhausted, area 4 will be marked against either area 1 or 2, whicever is being used at the time. In any case, one of the pair of areas 1 and 2 is always being merged with one of the pair of areas 3 and 4. The alternate area of either pair is being filled while the other area is being used. The point to notice here is that the output operation of merging is completely automatic until a Trap is executed which allows the supervisory routine to decide which areas need to be filled and marked for merging. This operation can proceed while the rest of the computer is busy on another problem, for the zero count in either of the counters will cause a break-in that will allow the supervisory routine to keep the operation going with a minimum of attention.

The starting point of this idea is Dr. Amdahl's. His method involved the marking and switching back and forth. The counters have been added to make the operation self-sufficient as an I/O operation. The reader will easily see that this scheme, without the counters, will also allow two instruction sets to be merged in one computer, and it is this idea that was referred to in the opening remarks of Part I of this memorandum.

There is another matter that can also be handled by counters in an easier ~~a relatively easy~~ manner, and that is memory assignment. It is possible, with the aid of an assignable counter or two, plus a variety of "Go" commands, to treat the problem of memory assigament merely as another case of multiple I/O operations. However, this matter is one that is usually handled in supervisory routines and, therefore, we will wait for Miss Boehm's memorandum which will cover memory assignment along with multiplexed I/O operations.

The reader will be glad to know that we have at last arrived at the end of this Part. Some of the explanations herein may not be too clear to the reader, but part of the trouble is that this memorandum is a description of an approach, not an explicit technique, and as we drift farther and farther from the rigid examples of Part I, we also drift farther from the area where rigid explanations are possible. We will now pass on to Part III, where we will take up the discussion of nets, or arrays, of computers.