HARVEST REPORT #9

Subject: Sequential Table Lookup (Part I)

By: W. J. Lawless, Jr.

Date: August 7, 1956

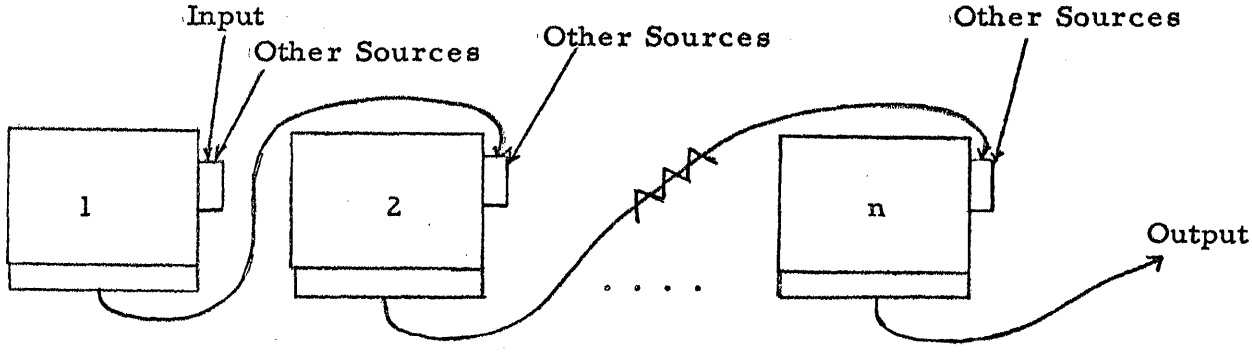
August 7, 1956

HARVEST REPORT #9

Sequential Table Lookup (Part I)

The problem of sequential table lookup in several tables wherein part of the address for the present table lookup is obtained from the previous table lookup arises in a number of applications. The basic difficulty is that the necessity to wait for the result of one table lookup before preceding to the next lookup prevents taking advantage of multiple memories to reduce the time of the several lookups.

One approach to this problem is to provide means for combining the output of a table lookup with a memory address constant (and possibly additional data from another source) to create a new lookup address for several memories simultaneously.



In this way a chaining process can be set up whereby memory 1 is making the 1st lookup on the p^{th} sequence of lookups, memory 2 is making the 2nd lookup on the $(p - 1)^{th}$ sequence, and memory n is making the last (n^{th}) lookup on the $(p - n)^{th}$ sequence. The result of this approach is that if there are n memories and n lookups to a sequence the total time for the sequence of lookups averages approximately 1 memory reference time.

The difficulty with this approach appears to be the requirement for additional and special equipment. Were there no other approach, this special equipment would be justified. However it appears that there may be ways of rearranging most problems which will eliminate the necessity for the special equipment.

The way in which problems with sequential table lookup must be reorganized in order to make maximum useage of multiple memories

will vary from problem to problem. Two specific problems have been examined and both of these appear to be subject to rearrangement in such a way that the multiple memories are fully utilized. The first of these is the problem of table search; i. e., given a table consisting of a number of groups, let us say 256, and given a specific group, search the table to see if the specific group is contained in the table.

The requirement is to utilize fully the multiple memories in this searching process. Let us assume that we have seven different memory boxes. The table of 256 groups would be sequenced and then distributed in these seven memories as follows: the middle group from the table (whenever there is an even number, take the higher of the two middle groups) will be stored identically in all seven memories. Similarly the two groups that divide the first half of the table and the last half of the table into quarters will be stored repeatedly in all seven memories. Similarly the groups that divide the four quarters of the table into eighths will also be stored in all seven memories. Thus we have a set of one group, two groups, and four groups, each of which is repeated in all seven memories. The file will continue to be broken down in the same fashion as described above such that we obtain the eight groups that divide the file into 16, the 16 groups that divide the file into 32, the 32 groups that divide it into 64, the 64 groups that divide it into 128, and the last set of 128 groups. However, the sets of 8, 16, 32, 64, and 128 groups will not be repeated in all seven memories, but rather, will be distributed through all seven memories. That is, the eight groups that divide the table into 32 will be distributed so that one of the eight groups is in each of the seven memories and the eighth group will double up in one of the memories. Similarly each of the larger sets of groups will be distributed through all seven memories.

The procedure would be as follows: 7 specific groups to be looked up in the table are made available. The first of the seven groups is compared with the middle group of the table in memory 1. The second of the seven groups is compared with the middle group of the table in memory 2, etc. up to the seventh group being compared with the middle group in the seventh memory. The results of the comparison will be temporarily stored in a register or in selectors. The first group from the same seven groups is now compared with one of the two groups that divided the file into quarters in memory 1. The choice of which of the two is determined by the result of the previous comparison with the middle group of the file. Then the second of the seven groups will be compared with the appropriate one of the two groups which divide the file into quarters and which are stored in the second memory. Similarly then for all seven

groups in all seven memories. Again the results of this comparison are temporily stored. Now we go back and start over again making a comparison of the first group of the seven with the appropriate one of the four groups stored in the first memory which divided the file into eights and again the second and succeeding groups are compared with the appropriate group from the set of four in each of the other seven memories. After comparing with the appropriate one of the four groups in the seventh memory we have reached the point where we change our procedure slightly. We are now about to compare the first of the seven groups with the appropriate one of eight groups that divided our file into 16ths. Which of these groups is determined by the results of the last comparison but in addition which of the seven memories is also determined by the results of the last comparison, since these eight groups are distributed through all seven memories. When we advance to the second of the seven groups to be compared against the set of eight we again go to the appropriate memory based upon the last comparison. We are from now on in this process assuming that the random nature of the groups being looked up in the file will result in having these searches distributed over all seven memories fairly evenly. Thus we would expect a considerable amount of overlap of memory accesses in the seven memories. The process is continued through the sets of 16, 32, etc. groups until we have exhausted our search for each of the seven groups being looked up. After we have located or found no match for all seven groups we are ready to bring in a new set of seven groups to be looked up in the table.

In this procedure I have not taken into account the possibilities presented by the fact that an equal condition may arise at any time through the successive binary searches of the table. We could further increase the effectiveness of the procedure by starting a new group through the successive searches as soon as any group has been matched. This would require more complex bookkeeping but otherwise would work out satisfactorily.

The second sequential table search problem has been written up as an attachment on separate sheets.

I have tentatively drawn the conclusion from the consideration of these two problems that most successive table lookup problems will prove to be susceptible to reorganization in such a way that we will obtain fairly full multiple memory utilization if there are a number of groups to be looked up. If not, then it is not too important to maintain maximum efficiency in the process anyway.

WJL/jh

W. J. Lawless, Jr.

August 30, 1956

Addenda to the Sequential Table Lookup- Part I

In discussing the preceding part of this writeup with several members of the Plantation group, other ways of rearranging the table search problem were discussed. In addition one point with respect to the technique outlined above was considered to require some statistical analysis.

One of the other approaches to the problem, suggested by John Rixse, will be described. Let us again assume a table of 256 groups, and let us assume that seven memories are available. We would take the seven groups from the table of 256 which divide the table into eights and store one of these seven groups in each of the seven memories. Then take the seven groups that divide 1/8th of the file into 64ths and store one of these groups in each of the seven memories. Similarly take the other sets of seven groups that divide the other eights of the file into 64ths and store them one group from each seven in each of the seven memories. Continue this process until the file is all stored. This will result in having 1/7th of the groups in the file in each of the seven memories. Thus, there is a small decrease in the total amount of memory storage used as compared with the procedure described above. Now we bring in one group rather than seven groups to be looked up in the file. (In some problems the fact that one group is brought in at a time rather than seven groups would have a distinct advantage.) This group is now compared with the seven groups that divide the file of 256 into eights. That is, the group is compared with one group from each of the seven memories. At the end of these seven comparisons, which are made essentially simultaneously as far as memory references are concerned, we then know with which seven groups from the seven memories to compare our group being looked up on the second round. This procedure is continued until the group is either matched or found to be unequal with any of the groups in the table. The procedure in this particular case would require the time of three memory accesses. We assume that the comparing could be done during this time. This procedure obviously makes full use of the multiple memories but is less efficient in terms of number of comparisons. Under certain circumstances, however, this may well be the best procedure to use.

With respect to the procedure described in the original Part I of this paper, it appears that it would be quite important to investigate the statistics involved in the rand m access to the several

memories. We must determine how often we would refer to a particular memory and find that memory in use. On the assumption that we simply go right ahead with our other lookups when a given memory is in use and on the further assumption that we need not keep all of these operations in phase, that is, we could be making the fourth lookup on one group and the sixth lookup on another group, it appears that every time that we refer to a memory and find it in use, we waste the time that it took us to determine that that memory was in use. This could become a serious loss of time, and therefore we propose to investigate the statistics with respect to this operation.

WJL/jh

W. J. Lawless, Jr.