Observations and Conclusions

The following observations and inferences are based on the initial experiences with a multiprogramming system:

1. It has been possible to make a significant improvement in the utilization of the main-frame time of the computer with large classes of real problems running at Lewis Research Center.
2. The interrupt feature is the basic tool that permits the automatic time sharing of independently coded and unrelated problems.
3. A clock that interrupts periodically would be a highly desirable tool to be used in the management of more general multiprogramming systems. Such a clock would be useful in preventing a "looping" problem from "hogging" the computer, as well as for running-time accounting.
4. Multiprogramming provides the incentive for more efficient problem and system codes by removing the output barrier. Previously, there was little motivation to prepare efficient codes in output-limited problems, since there was little advantage to be gained. The multiprogramming system has provided a method of exploiting such gains and has produced pressure for more efficient codes.
5. Operator attention to the input-output devices, such as a tape change, no longer need delay the computer. The computer can push ahead any problem that is not using the device undergoing the change.
6. Multiprogramming seems to develop a trend towards splintering of problems and their input-output tasks. This appears to have two advantages for multiprogramming in computers of limited store and input devices. First, it provides a much better opportunity to obtain a feasible mix of problems in the high-speed memory. Splintering of problems or data-reduction tasks will increase the possibility of getting compatible problems for a parallel-operation schedule.
7. Scheduling of parallel problems could be facilitated by an inexpensive procedure for code relocation. It would be desirable to relocate the codes of a current problem in the high-speed memory without more than a nominal delay in computation.
8. On-line debugging of problems is costly of computer time and an inefficient use of computer capabilities. However, from an individual problem standpoint, it is often an effective method of debugging. Parallel operation during on-line debugging, if it could be made safe, may reduce the cost of on-line debugging to a point where it is feasible.

REFERENCE

1. Turner, L. R., and Rawlings, J. H. Realization of randomly timed computer input and output by means of an interrupt feature. *IRE Trans. EC-7*, no. 2 (June 1958), 141–149.

ALGORITHM 92
SIMULTANEOUS SYSTEM OF EQUATIONS AND MATRIX INVERSION ROUTINE

Derek Johann Roek
Applied Physics Laboratory of Johns Hopkins University, Silver Spring, Maryland

```
procedure SIMULTANEOUS (U, W, C, X, B, n, kount, eps,
  absf) ;
array U, W, C, X, B ; integer n, kount ;
  real eps; real procedure absf;
comment This procedure solves the problem Ux := b for the
  vector x. It assumes the problem written in the form x'U' := b',
  where ' denotes transpose. The procedure is completed in n
  cycles and may be iterated kount times (kount ≦ 6). The trans-
  pose of U is in U[,] and the row vector b' is in B. The integer n
  is the dimension of U, and the solution row vector x' is in X.
  The matrix C is a check of accuracy. It should have b' in its
  first row, the first element b1 of b' along its main diagonal,
  and zeros elsewhere. The real number eps checks to see how close
  the actual result is to this theoretical one. Also if we let b' :=
  (1, 0, ··· , 0), then this procedure finds the inverse W[,] of U.
  The function absf finds the absolute value of its argument. The
  procedure chooses the column vectors of U as the row vectors of
  W in the 0th cycle of the first iteration. For all subsequent itera-
  tions, the row vectors of W, computed at the nth cycle of the
  last iteration, are the row vectors of W in the 0th cycle ;
begin integer i, j, k, p ; real bh, b1, Z ;
  for j := 1 step 1 until n do
      for i := 1 step 1 until n do W[j, i] := U[i, j];
  S1: for j := 1 step 1 until n do
        for i := 1 step 1 until n do C[i, j] := 0 ;
  for j := 1 step 1 until n do
      begin for k := 1 step 1 until n do
        begin C[j, j] := C[j, j] + W[j, k] × U[k, j] end;
        if j = 1 then Z := B[j]/C[j, j] else Z := 1/C[j, j];
        for k := 1 step 1 until n do
            begin X[k] := Z × W[j, k];
              W[j, k] := X[k]
            end k;
        for k := 1 step 1 until n do
            begin if k = j then go to S2 else
              for p := 1 step 1 until n do
                  C[k, j] := C[k, j] + U[p, j] × W[k, p];
            if j = 1 then bh := B[j] else bh := 1;
            if k = 1 then b1 := B[j] else b1 := 0;
            for p := 1 step 1 until n do
            begin X[p] := bh × W[k, p] + (b1 − C[k, j]) ×
            W[j, p];
              W[k, p] := X[p]
            end p;
  S2:        if k = j ∧ j = n then go to S3
            end k;
      end j;
  S3: for j := step 1 until n do
          if absf(absf(C[j, j]) − absf(B[1])) > eps then go to S4;
      go to S6;
  S4: if kount > 0 then go to S5 else go to S6;
  S5: kount := kount − 1;
      go to S1;
  S6: for j := step 1 until n do
          X[j] := W[1, j];
  S7: end SIMULTANEOUS
```