mer. No restrictions exist as to number or size of these parameters. If a macro-instruction is to be generative, it contains one other part aside from the main body of stored coding. This part decides, from an interrogation of call-line parameters, which particular set of macro-instruction coding is to be included in the main routine.

Subroutines

The assembly system provides for an expandable library of subroutines to be available to the programmer. These subroutines generate assembly language pseudocode and as such may use all the assembly features such as macro-instructions, descriptor verbs, and so forth. Subroutines may be open or closed and generative or fixed.

Parameters for subroutines are specified at the pseudocode calling line. For open subroutines, parameters are incorporated during the operation of the assembler. These parameters may merely be substituted in the subroutine as in the case of a fixed routine or may be subject to considerable testing and manipulation as occurs with a generative subroutine.

Closed subroutines may either incorporate parameters during assembly or use parameters generated by the running machine program. In this case the parameters are located relative to the subroutine call-line.

The design of the system is open to the extent that any useful number of macro-instructions and subroutines may be added.

Provisions for Program Modification

The Assembly System offers two main listings for program up-dating. First, listings are given of the object machine code and the Assembly language pseudocode. Second, is a list of all symbolic addresses and those instructions referring to them. In addition, the Assembly System generates an information block preceding each object program. This block, which contains all program stops, breakpoint switches, and tape addresses is available for input to a service routine which will modify any corresponding entries within the object program.

There are two types of error indicators used by the Assembler. One causes the Assembly System to print the source of trouble and stop immediately. The other and major class consists of on-line printed statements indicating the type and location of errors. In this case the Assembly System continues its functions ignoring the "guilty" statements until all such indicators have been found. This permits the user to specify corrective measures for all errors at one time.

In summary then, the 501 Assembly System lies in an intermediate category. On the one hand, it is definitely machine oriented, amplifying the 501 instruction complement and requiring a knowledge of the 501. However, it also provides for a flexibility of order statements, not confined to the 2 address machine order code. A variable number of machine instructions are generated dependent upon the number and types of entries made on each pseudocode line. Both macro-instructions and subroutines may be of the generative type and since the library is open-ended, may be augmented whenever necessary.

In short, the RCA 501 Assembly System is a programmer's aide, enabling him to make maximum use of machine capabilities with a minimum of clerical effort.

Acknowledgment

The authors acknowledge the extensive contributions of M. J. Sendrow, who participated in the planning and creation of the RCA 501 Assembly System.

A Program to Draw Multilevel Flow Charts

LOIS M. HAIBT†

Introduction

THE preparation of a program for a digital computer is not complete when a list of instructions has been written. It still must be determined that the instructions do the required job, and if necessary the instructions must be changed until they do. Also a description of the program should be written for others who may want to understand the program. A useful tool for the last purpose is a graphical outline of the program—a flow chart.

Flow charts serve two important purposes: making a program clear to someone who wishes to know about it, and aiding the programmer himself to check that the program as written does the required job. A flow chart drawn by the programmer would serve for the first purpose, but drawing one is often a tedious job which may or may not be done well. For the second purpose, it is important to have the flow charts show accurately what the program does rather than what the programmer might expect it to do. Consequently, it was decided to write a program, the Flowcharter, for the IBM 704 to produce flow charts automatically from a list of in-

† IBM Res. Center, Yorktown Heights, N. Y.

structions. Another reason for the project was to get further insight into the characteristics of computer programs.

Since programs in many programming languages and even for different machines differ mainly in superficial aspects such as names and numbers for various operations, names and types of registers, it was decided to have the Flowcharter do the main part of its work on a common, machine-independent language and to have a set of preprocessors, each of which would translate one programming language into this common internal language.

We also felt it was desirable not to attempt to show the whole program as one chart, which for a moderate size program would either present a confusion of detail or be too general to serve the purpose. In order to provide both a good general picture of the program or of any part of it, and a more detailed description of a smaller piece of program, the Flowcharter produces a series of flow charts on a number of levels of detail; each part of a chart is shown in more detail on a succeeding chart. How to determine the makeup of the charts was one of the most difficult problems encountered in planning the Flowcharter.

Another feature of a flow chart is a description of the procedure represented by each box. The Flowcharter provides a summary of the machine input-output done in the box and a summary of the computation done in the box, listing the quantities computed and those used in the computation of each of them.

Description of the Flowcharter

The Flowcharter is composed of four main parts: the preprocessors, the flow analysis, the computation summary, and the output program.

The preprocessors each do a simple translation from the external instructions to the internal language. For most machines, an instruction may represent several different processes done in the machine, such as fetching from memory, storing the memory, and instruction sequencing. These operations are each described separately in the internal language. One external instruction is translated by the preprocessor into a suitable list of these operations.

Many of the problems which arose in designing a Flowcharter were in the section which determines what is to be shown on each chart. It is very easy for the programmer to mark off his program into logical parts, but to determine these from the program itself is quite difficult in most programming languages. We have worked out a set of techniques which we feel will do quite well for most programs and will be acceptable in other cases. We have also provided facilities for the programmer to specify how he would like the breakdown done on various levels if he does not like the choices made by the Flowcharter. The techniques used depend mainly on analysis of the flow properties of the program but provision is also made in the Flowcharter for using the data to help in the analysis. The Flowcharter is written in such a way that various techniques and combinations of techniques can be tested to see what results they give.

This flow analysis is done by iteratively forming regions from groups of subregions. The smallest subregions are individual instructions. In general, each region will be represented by one flow chart and each box drawn on the chart will represent a subregion of that region. However, when it is reasonable, two or more regions, each consisting of only two or three subregions, will be shown on one flow chart. This is done to keep the output moderately compact. Also, those regions which are formed directly from instructions are not shown as flow charts but are given as a list of the instructions in the region, with a reference to the page on which this region is shown in context. (The Appendix shows an example of this.)

The techniques used for region formation are of two kinds, combination and division. A combination technique is one which starts with individual instructions and, by repeated applications, combines them into larger and larger regions. A division technique is one which starts with the whole program and divides it into smaller parts. Each of these parts is in turn divided until each part consists of not more than six or seven of the regions formed by the techniques of the first type. Each technique is represented by a subroutine.

Each combination subroutine searches for a particular configuration of flow in the program. Three such subroutines are: STRING, DIAMND, and TEST, which look for "strings," "diamonds," and "test sets."

A "string" (see Fig. 1) is an ordered set of regions satisfying the condition that every region, except the first, has an entry only from the preceding region and each, except the last, has an exit only to the next one.

A "diamond" (see Fig. 2) is a set of regions containing a first region *F*, a last region *L*, and some intermediate blocks. Each intermediate block must not have any predecessor other than *F* nor any successor other than *L*. All successors of *F* and predecessors of *L* must be in the "diamond."

A "test set" is a set of regions which together make up a compound test. A set of regions forms a "test set" if each region ends with a test of the same special register. Also, every region except the first may have only one predecessor which must also be in the set. Finally, only the special register tested may be changed by the instructions in any of the regions except, possibly, the first one. For example, consider the 704 SAP instructions:

```
CLA    ALPHA
TZE    ISZERO
SUB    ONE
TZE    ISONE
SUB    ONE
TZE    ISTWO
SUB    ONE
TZE    ISTHRE
```

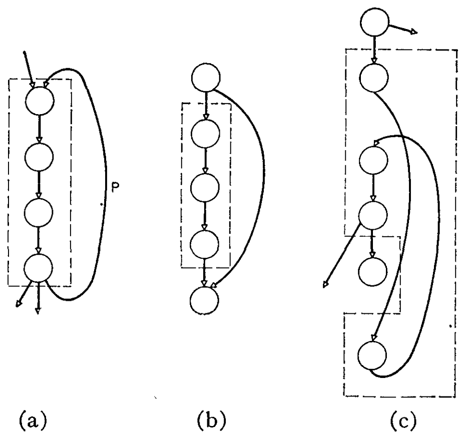

Fig. 1—In each case the dotted lines enclose a "string." (Circles represent regions formed earlier and solid lines represent paths of flow in the direction of the arrow.)

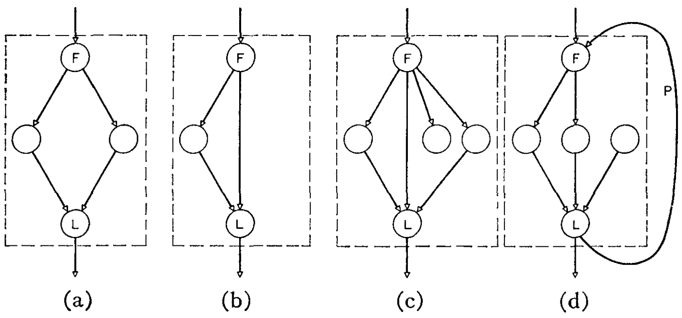

Fig. 2—In each case the dotted lines enclose a "diamond." (Circles represent regions formed earlier and solid lines represent paths of flow in the direction of the arrow.)

The pair CLA, TZE, and each pair SUB, TZE make up a region found by STRING; then these four regions will be combined by TEST.

It should be pointed out that the first two configurations, "strings" and "diamonds," are sufficient to describe most programs. Iterative loops do not have to be taken care of separately; when the program within the loop is combined into a region, the return path of the loop is also included in the region. For example, in Fig. 1(a) and Fig. 2(d), the return path, P, although not a part of the string or diamond, is a link between the subregions forming the region and is therefore included in the flow chart of that region. The example used to show the output of the Flowcharter is a program for which only STRING and DIAMND are needed.

The division subroutines attempt to discover particular configurations "in the large." Two such subroutines are UNWRAP and SPLIT which look for loops and easily separable parts of the program. The division subroutines are not allowed to separate the regions already built up by the combination routines.

UNWRAP determines if the program is essentially one large loop; that is, it has an entry block E_1, which has only one successor S, an exit block E_2, which has only one predecessor P, and there is a path from P to S. In this case, the region representing the program is made up of three subregions: E_1, E_2, and the subregion including everything else. The last now becomes the "program" to be divided further.

SPLIT looks for the situation where the program is composed of several essentially distinct parts, each of which has only one entry point and one exit point for paths to or from other parts. Each such part is one subregion and is divided further if necessary.

At present, STRING, DIAMND, and TEST are used repeatedly until none of them can do any further combining. If there are no more than six regions left, these are combined to make the region representing the entire program. If there are more than six left, UNWRAP and SPLIT are used repeatedly until they have either divided the entire program into the regions left by the combination routines or can not divide it any further. In the latter case, at present, arbitrary divisions are made until the program is so divided.

This method should be adequate for most programs; however, the Flowcharter is written in such a way that routines can be added and other methods tried easily.

In planning the computation analysis, the major problem encountered was that of determining when cells or registers were used only as temporary or erasable storage. In order to keep the amount of information down to a readable size, we wanted to list only the cells actively used in the region. We started with the idea of labeling a quantity computed but not used as "output," and those computed and then used, "tentative outputs" to indicate that they might be erasable cells. A "tentative output" was carried forward until an exit from the program or a use of the same quantity was encountered. If there was such a use, the "tentative output" became a real output—if not, it was considered erasable and would not appear further on the flow charts for that part of the program. Since a "tentative output" had to be carried forward on all possible paths but changed to a real output only on those paths on which a use was encountered, the bookkeeping necessary became unmanageable when the flow of the program was complicated.

If the computation is traced backward rather than forward, the procedure becomes much simpler. If a quantity is needed at one point of a program, it must be available along every possible path backwards from that point until some point is encountered where the quantity is computed or until an entrance to the program is encountered. In the latter case, this quantity must be available at that entrance to the program. With each region shown on a flow chart, all the quantities computed in that region are listed except those erasable cells which are used only within the region. For each quantity computed, there is given a list of quantities which are required at the entrances to the region and which enter into the computation of this item whether directly or indirectly.

The last part of the Flowcharter arranges and prints the results of the other sections. The appearance of the final flow charts will be one of the most important features to anyone using the Flowcharter and will be as much like hand-drawn flow charts as possible. Each page will show one region composed of as many as six or seven smaller regions. Each of the boxes will indicate the entry and exit locations of the subregion it represents, the number of the page that has a detailed flow chart of the subregion, the location of any exits and entrances in the middle of the region and the exit conditions for any exit. Each box will be outlined by asterisks and all transfer paths will be represented by lines to the appropriate points. If the lines go outside the region, the name and page number of the instruction at the other end will be given at the top or bottom of the page for entrances or exits, respectively. The boxes themselves will be arranged in two dimensions to show the flow in the region as clearly as possible, using horizontal as well as vertical displacements of the boxes. On the right-hand side of the page will be listed further information in three columns: the names and numbers of any input-output units used, and which memory cells were involved in each case; a list of those quantities for which values must be available at the entrance to the region; and the computation summaries mentioned above. Provision is also made for reproducing comments, box titles, page titles, and other comments given by the programmer. (See the Appendix for an example of the output.)

Status of the Program

On February first, the program described here was nearly complete and checked out with a preprocessor for 704 SAP language. The parts not yet finished were UNWRAP, SPLIT, the drawing of the boxes and lines in the output program, and provision for some of the specifications by the programmer. In each case, much or all of the planning has been done.

As soon as these are complete, it is planned to write 705 and FORTRAN preprocessors and at least one region forming subroutine which considers mainly the data used by the program, and has much less emphasis on the flow properties than the routines described here. Also planned is some experimentation with various methods of region formation.

Acknowledgment

The author wishes to acknowledge the contributions of Alex Bernstein, who collaborated on the initial phases of the project, and of James Lagona, who wrote certain of the subroutines. The author also would like to thank colleagues in the Programming Research Department for helpful discussions and advice.

Appendix

The source program flow charted here is:

```
C     A PROGRAM TO MULTIPLY TWO MATRICES AND SUBSTITUTE PLUS ZERO FOR
C     EACH ZERO ELEMENT, PLUS ONE FOR EACH POSITIVE ELEMENT, AND MINUS
C     ONE FOR EACH NEGATIVE ELEMENT.
        10  READ 200 ((M(I, J), I=1, 3), J=1, 4), ((N(J, K), J=1, 4), K=1, 5)
        20  DO 140 I=1, 3
        30  DO 130 K=1, 5
        40  L(I, K)=0
        50  DO 60 J=1, 4
        60  L(I, K)=L(I, K)+M(I, J) * N(J, K)
        70  IF (L(I, K) 120, 100, 80
        80  L(I, K)=+1
        90  GO TO 130
       100  L(I, K)=0
       110  GO TO 130
       120  L(I, K)=-1
       130  CONTINUE
       140  CONTINUE
       150  PRINT 200 ((L(I, K), I=1, 3), K=1, 5)
       160  STOP
       200  FORMAT (15I4)
```

EXPLANATION OF CONTENTS OF BOXES IN THE FLOW CHARTS

```
        PATHS INTO THIS SUBREGION
             .           .           .
             V           V           V
* * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*   FIRST                               LAST  *      READING          VALUES         COMPUTATION
* LOCATION                          LOCATION  *      WRITING         REQUIRED           DONE
*                                             *
*          WHERE TO FIND MORE                 *
*          DETAILS ABOUT THIS                 *
*          SUBREGION                          *
*                                             *
*                                             *
*            EXIT CONDITIONS                  *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * *
             .           .           .
             .           .           .
             V           V           V
        PATHS OUT OF THIS SUBREGION
```

```
                                         PAGE 1
                   A PROGRAM TO MULTIPLY TWO MATRICES AND SUBSTITUTE PLUS ZERO FOR
                   EACH ZERO ELEMENT, PLUS ONE FOR EACH POSITIVE ELEMENT, AND MINUS
                   ONE FOR EACH NEGATIVE ELEMENT.
     ENTRANCE TO                     READING          VALUES          COMPUTATION
       PROGRAM                       WRITING         REQUIRED            DONE
          .
          .
          V
* * * * * * * * * * * * * *
* 10                 20 *         READ CARDS . . .                   M(I, J) . . . CARDS
*        P.3            *           M(I, J)                          N(J, K) . . . CARDS
*                       *           N(J, K)                          I . . . +1
*      UNCOND           *
* * * * * * * * * * * * * *
        .
        .     . . . . . . . . . . .
        .     .                   .
        V     V                   .
* * * * * * * * * * * * * *       .
* 30                 30 *         .                                  K . . . +1
*        P.3            *         .
*                       *         .
*      UNCOND           *         .
* * * * * * * * * * * * * *       .
        .                         .
        .     . . . . . . . .     .
        .     .             .     .
        V     V             .     .
* * * * * * * * * * * * * * .     .
* 40                130 *   .     .                 I                K . . . K
*        P.2            *   .     .                 K                     +1
*                       *   .     .                 M(I, J)          L(I, K) . . . +0
*      K IS TO 5        *   .     .                 N(J, K)                        -1
* GREATER    LESS, =    *   .     .                                                +1
* * * * * * * * * * * * * * .     .
        .     .             .     .
        .     . . . . . . . .     .
        .                         .
        V                         .
* * * * * * * * * * * * * *       .
* 140               140 *         .                 I                I . . . I
*        P.3            *         .                                         +1
*                       *         .
*      I IS TO 3        *         .
* GREATER    LESS, =    *         .
* * * * * * * * * * * * * *       .
        .     .                   .
        .     . . . . . . . . . . .
        .
        V
* * * * * * * * * * * * * *
* 150               160 *            PRINT . . .        L(I, K)
*        P.3            *              L(I, K)
*                       *
*       STOP            *
* * * * * * * * * * * * * *
          .
          .
          V
      EXIT FROM
       PROGRAM
```

```
                                                            PAGE 2

                                 30                                                   READING  VALUES    COMPUTATION
                                 P.1                                                  WRITING REQUIRED      DONE
                                  .
                                  .         . . . . . . . . . . . . . . . . . . . . .
                                  .         .                                       .
                                  V         V                                       .
                          * * * * * * * * * * * * * *                               .
                          * 40                   50 *                               .          I         L(I, K) · · · +0
                          *          P.3            *                               .          K         J · · · +1
                          *                         *                               .
                          *         UNCOND          *                               .
                          * * * * * * * * * * * * * *                               .
                                  .                                                 .
                                  .         . . . . . . . . . . .                   .
                                  .         .                   .                   .
                                  V         V                   .                   .
                          * * * * * * * * * * * * * *           .                   .
                          * 60                   60 *           .                   .          I         L(I, K) · · · L(I, K)
                          *          P.3            *           .                   .          J                       M(I, J)
                          *                         *           .                   .          K                       N(J, K)
                          *        J IS TO 4        *           .                   .          M(I, J)   J · · · J
                          * GREATER         LESS, = *           .                   .          N(J, K)           +1
                          * * * * * * * * * * * * * *           .                   .          L(I, K)
                                  .         .                   .                   .
                                  .         . . . . . . . . . . .                   .
                                  .                                                 .
                                  V                                                 .
                          * * * * * * * * * * * * * *                               .
                          * 70                   70 *                               .          I
                          *          P.3            *                               .          K
                          *                         *                               .          L(I, K)
                          *    L(I, K) IS TO 0      *                               .
                          * GREATER    =       LESS *                               .
                          * * * * * * * * * * * * * *                               .
                                  .    .    .                                       .
              . . . . . . . . . . .    .    . . . . . . . . . . .                   .
              .                        .                        .                   .
              .                        .                        .                   .
              V                        .                        .                   .
* * * * * * * * * * * * * *            .                        .                   .
* 80                   90 *            .                        .                   .          I         L(I, K) · · · +1
*          P.3            *            .                        .                   .          K
*                         *            .                        .                   .
*         UNCOND          *            .                        .                   .
* * * * * * * * * * * * * *            .                        .                   .
              .                        .                        .                   .
              .                        V                        .                   .
              .           * * * * * * * * * * * * * *           .                   .
              .           * 100                 110 *           .                   .          I         L(I, K) · · · +0
              .           *          P.3            *           .                   .          K
              .           *                         *           .                   .
              .           *         UNCOND          *           .                   .
              .           * * * * * * * * * * * * * *           .                   .
              .                        .                        .                   .
              .                        .                        V                   .
              .                        .            * * * * * * * * * * * * * *     .
              .                        .            * 120                 120 *     .          I         L(I, K) · · · −1
              .                        .            *          P.3            *     .          K
              .                        .            *                         *     .
              .                        .            *         UNCOND          *     .
              .                        .            * * * * * * * * * * * * * *     .
              .                        .                        .                   .
              .                        .                        .                   .
              .                        .                        .                   .
              . . . . . . . . . . .    .    . . . . . . . . . . .                   .
                                  .    .    .                                       .
                                  .    .    .                                       .
                                  V    V    V                                       .
                          * * * * * * * * * * * * * *                               .
                          * 130                 130 *                               .          K         K · · · K
                          *          P.3            *                               .                            +1
                          *                         *                               .
                          *        K IS TO 5        *                               .
                          * GREATER         LESS, = *                               .
                          * * * * * * * * * * * * * *                               .
                                  .         .                                       .
                                  .         .                                       .
                                  .         . . . . . . . . . . . . . . . . . . . . .
                                  .
                                  V
                                 140
                                 P.1
```

```
                                   PAGE 3
INSTRUCTIONS                                                          FOR CONTEXT
                                                                       SEE PAGE
C    A PROGRAM TO MULTIPLY TWO MATRICES AND SUBSTITUTE PLUS ZERO FOR
C    EACH ZERO ELEMENT, PLUS ONE FOR EACH POSITIVE ELEMENT, AND MINUS
C    ONE FOR EACH NEGATIVE ELEMENT.
            10  READ 200 ((M(I, J), I=1, 3), J=1, 4), ((N(J, K), J=1, 4), K=1, 5)    1
            20  DO 140 I=1, 3

            30  DO 130 K=1, 5                                                       1

            40  L(I, K)=0                                                           2
            50  DO 60 J=1, 4

            60  L(I, K)=L(I, K) M(I, J) *N(J, K)                                    2
                   (END OF DO AT 50)

            70  IF (L(I, K) 12, 10, 80                                              2

            80  L(I, K)=+1                                                          2
            90  GO TO 130

           100  L(I, K)=0                                                           2
           110  GO TO 130

           120  L(I, K)=-1                                                          2

           130  CONTINUE                                                            2
                   (END OF DO AT 30)

           140  CONTINUE                                                            1
                   (END OF DO AT 20)

           150  PRINT 200 (L(I, K), I=1, 3), K=1, 5)                                1
           160  STOP
```

A Compiler Capable of Learning

RICHARD F. ARNOLD[†]

Introduction

WE WOULD like to consider a new approach to the general problem of programming computers. To date, the methods of handling programming problems can be roughly classified into two families, each of which have certain characteristic advantages and disadvantages which seem to complement those of the other.

The first group, developed from the subroutine philosophy, includes all interpretive schemes, as for example the "Bell Labs Interpretive System" for the IBM 650. The advantages of interpretive routines are that they are very versatile in the languages they can interpret and are comparatively easy to write. It is a fairly simple matter to write an interpretive routine to simulate another computer and thus achieve program compatibility between different machines. The crippling drawback is the excessive time needed to execute routines interpretively. Higher order interpretive schemes increase executions time exponentially.

The second group consists of compilers and assembly programs. They are characterized by the fact that, unlike interpretive routines, they produce object programs which may be executed in reasonable amounts of time. Compilers, however, are difficult to write. "Fortran," for example, took twenty-five man years to write. A second difficulty of compilers such as "Fortran," is that although they are becoming more and more versatile, they still fail to express certain types of operations, and it has become necessary to make it possible to adapt the compiler so that the "Fortran" language may be temporarily left and programming done in a language closer to the initial machine language. Of course, this is a desirable feature for a compiler to have, but it does not solve the initial problem for which it was created, namely, to avoid machine languages completely. A further disadvantage is that as a compiler system becomes adapted for use on more than one computer, many of the "coding tricks" will have to be avoided. This may be desirable from the point of view of the compiler writer, but

[†] Michigan State University, East Lansing, Mich. This research was supported in part by a grant from the Natl. Sci. Found. and taken from a thesis written under the direction of G. P. Weeg.