A Comparative Study of Flowcharts and Program Design Languages for the Detailed Procedural Specification of Computer Programs

by

H. Rudy Ramsey and Michael E. Atwood
SCIENCE APPLICATIONS, INC.

and

James R. Van Doren
OKLAHOMA STATE UNIVERSITY

SCIENCE APPLICATIONS, INC.
7935 East Prentice Avenue
Englewood, Colorado 80110

SEPTEMBER 1978

Contract DAHC 19-76-C-0040

Monitored technically by Edgar M. Johnson and Jean Nichols Hooper
Battlefield Information Systems Technical Area, ARI

Prepared for

U.S. ARMY RESEARCH INSTITUTE
for the BEHAVIORAL and SOCIAL SCIENCES
5001 Eisenhower Avenue
Alexandria, Virginia 22333

Approved for public release; distribution unlimited
U. S. ARMY RESEARCH INSTITUTE
FOR THE BEHAVIORAL AND SOCIAL SCIENCES

A Field Operating Agency under the Jurisdiction of the
Deputy Chief of Staff for Personnel

JOSEPH ZEIDNER
Technical Director

WILLIAM L. HAUSER
Colonel, US Army
Commander

Research accomplished for the
Department of the Army

Science Applications, Inc.

NOTICES

DISTRIBUTION: Primary distribution of this report has been made by ARL. Please address correspondence concerning distribution of reports to: U. S. Army Research Institute for the Behavioral and Social Sciences, ATTN: PERI-P, 5001 Eisenhower Avenue, Alexandria, Virginia 22333.

FINAL DISPOSITION: This report may be destroyed when it is no longer needed. Please do not return it to the U. S. Army Research Institute for the Behavioral and Social Sciences.

NOTE: The findings in this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.
DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DDC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.
An experiment was performed to assess the relative merits of Program Design Languages (PDLs) and flowcharts as techniques for the development and documentation of detailed designs for computer programs.

Twenty students in a computer science graduate course participated in this experiment. Working individually, the students designed a two-pass assembler for a simple minicomputer. Half the students expressed their design...
for the first pass of the assembler in the form of a flowchart, and expressed their design for the second pass in a Program Design Language. The other half of the students used a PDL for pass one, and a flowchart for pass two. Flowcharts and PDLs were compared on the basis of various measures of overall design quality, design errors, level of detail of designs, time expended in developing designs, and subjective preferences. 

Having completed this design task, the subjects then performed an implementation task. They were given fairly detailed procedural designs for a program which simulates the function of a fairly sophisticated minicomputer. They were then required to develop a working version of the program in PL/1. Although the designs were logically equivalent, half the students received their simulator design in flowchart form, and half in PDL form. Flowcharts and PDLs were compared on the basis of design comprehension test performance, various measures of overall implementation quality, implementation errors, and subjective preferences.

In the context in which this study was performed, the use of a Program Design Language (PDL) by a software designer, for the development and description of a detailed program design, produced better results than did the use of flowcharts. Specifically, the designs appeared to be of significantly better quality, involving more algorithmic or procedural detail, than those produced using flowcharts. In addition, flowchart designs exhibited considerably more abbreviation and other space-saving practices than did PDL designs, with a possible adverse effect on their readability.

When equivalent, highly readable designs were presented to subjects in both PDL and flowchart form, no pattern of short-term or long-term differences in comprehension of the design was observed. No significant differences were detected in the quality or other properties of programs written as implementations of the designs. Subjective ratings indicated a mild preference for PDLs.

Overall, the results suggest that software design performance and designer-programmer communication might be significantly improved by the adoption of informal Program Design Languages, rather than flowcharts, as a standard documentation method for detailed computer program designs.
FOREWORD

The Battlefield Information Systems Technical Area is concerned with the human resource demands of increasingly complex battlefield systems used to acquire, transmit, process, disseminate, and utilize information. This increased complexity places greater demands upon the operator interacting with the machine system. Research in this area is focused on human performance problems related to interactions within command and control centers as well as issues of system development.

It is concerned with such areas as software development, topographic products and procedures, tactical symbology, user oriented systems, information management, staff operations and procedures, and sensor systems integration and utilization.

One area of special interest involves the development of computer software to support automated battlefield systems. Software development is a costly, unreliable, not well understood process. The present research assessed the relative merits of Program Design Languages (PDLs) and flowcharts as techniques for the development and documentation of detailed designs for computer programs. The research is part of a larger effort to develop a conceptualization of the programming process and identify behavioral bottlenecks in software development. Efforts in this area are directed at improving accuracy and productivity in programming through the design of procedures, languages, and methods to enhance programmer performance.

Research in the area of human factors in software development is conducted as an in-house effort augmented contractually by organizations selected as having unique capabilities and facilities. The effort is responsive to requirements of Army Project 2Q762725A778, and to general requirements expressed by members of the Integrated Software Research and Development Working Group (ISRAD).
BRIEF

Requirement:

This experimental study was performed in an attempt to assess the relative merits of Program Design Languages (PDLs) and flowcharts as techniques for the development and documentation of detailed designs for computer programs.

Procedure:

Twenty students in a computer science graduate course participated in an experiment. Working individually, the students designed a two-pass assembler for a simple minicomputer. Half the students expressed their design for the first pass of the assembler in the form of a flowchart, and expressed their design for the second pass in a Program Design Language. The other half of the students used a PDL for pass one, and a flowchart for pass two. Flowcharts and PDLs were compared on the basis of various measures of overall design quality, design errors, level of detail of designs, time expended in developing designs, and subjective preferences.

Having completed this design task, the subjects then performed an implementation task. They were given fairly detailed procedural designs for a program which simulates the function of a fairly sophisticated minicomputer. They were then required to develop a working version of the program in PL/1. Although the designs were logically equivalent, half the students received their simulator design in flowchart form, and half in PDL form. Flowcharts and PDLs were compared on the basis of design comprehension test performance, various measures of overall implementation quality, implementation errors, and subjective preferences.
Findings:

In the context in which this study was performed, the use of a Program Design Language (PDL) by a software designer, for the development and description of a detailed program design, produced better results than did the use of flowcharts. Specifically, the designs appeared to be of significantly better quality, involving more algorithmic or procedural detail, than those produced using flowcharts. In addition, flowchart designs exhibited considerably more abbreviation and other space-saving practices than did PDL designs, with a possible adverse effect on their readability.

When equivalent, highly readable designs were presented to subjects in both PDL and flowchart form, no pattern of short-term or long-term differences in comprehension of the design was observed. No significant differences were detected in the quality or other properties of programs written as implementations of the designs. Subjective ratings indicated a mild preference for PDLs.

Utilization of Findings:

Overall, the results suggest that software design performance and designer-programmer communication might be significantly improved by the adoption of informal Program Design Languages, rather than flowcharts, as a standard documentation method for detailed computer program designs.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INTRODUCTION</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>METHOD</strong></td>
<td></td>
</tr>
<tr>
<td>Overview</td>
<td>10</td>
</tr>
<tr>
<td>Subjects</td>
<td>10</td>
</tr>
<tr>
<td>Preliminary Phase</td>
<td>11</td>
</tr>
<tr>
<td>Design Phase</td>
<td>12</td>
</tr>
<tr>
<td>Implementation Phase</td>
<td>14</td>
</tr>
<tr>
<td><strong>RESULTS</strong></td>
<td>16</td>
</tr>
<tr>
<td>Design Phase</td>
<td>21</td>
</tr>
<tr>
<td>Time to Produce Designs</td>
<td>22</td>
</tr>
<tr>
<td>Quality and Properties of Designs</td>
<td>24</td>
</tr>
<tr>
<td>Subjective Ratings of Documentation Methods</td>
<td>35</td>
</tr>
<tr>
<td>Implementation Phase</td>
<td>39</td>
</tr>
<tr>
<td>Comprehension of Simulator Design</td>
<td>39</td>
</tr>
<tr>
<td>Quality of Implementation</td>
<td>42</td>
</tr>
<tr>
<td>Subjective Ratings of Documentation Methods</td>
<td>45</td>
</tr>
<tr>
<td><strong>DISCUSSION</strong></td>
<td>49</td>
</tr>
<tr>
<td><strong>CONCLUSIONS</strong></td>
<td>54</td>
</tr>
<tr>
<td><strong>REFERENCES</strong></td>
<td>55</td>
</tr>
<tr>
<td><strong>APPENDIX - A</strong></td>
<td></td>
</tr>
<tr>
<td>Schedule of Events</td>
<td>A-1</td>
</tr>
<tr>
<td><strong>APPENDIX - B</strong></td>
<td></td>
</tr>
<tr>
<td>Consent Forms</td>
<td>B-1</td>
</tr>
<tr>
<td><strong>APPENDIX - C</strong></td>
<td></td>
</tr>
<tr>
<td>Background Questionnaire</td>
<td>C-1</td>
</tr>
<tr>
<td><strong>APPENDIX - D</strong></td>
<td></td>
</tr>
<tr>
<td>Software Design Methods Handout</td>
<td>D-1</td>
</tr>
<tr>
<td><strong>APPENDIX - E</strong></td>
<td></td>
</tr>
<tr>
<td>PDL and Flowchart Examples</td>
<td>E-1</td>
</tr>
<tr>
<td><strong>APPENDIX - F</strong></td>
<td></td>
</tr>
<tr>
<td>Basic Description of Design Problem</td>
<td>F-1</td>
</tr>
<tr>
<td><strong>APPENDIX - G</strong></td>
<td></td>
</tr>
<tr>
<td>Description of Pass 1 Design Problem</td>
<td>G-1</td>
</tr>
<tr>
<td><strong>APPENDIX - H</strong></td>
<td></td>
</tr>
<tr>
<td>Description of Pass 2 Design Problem</td>
<td>H-1</td>
</tr>
<tr>
<td><strong>APPENDIX - I</strong></td>
<td></td>
</tr>
<tr>
<td>Time Expenditure Questionnaire, Design Phase</td>
<td>I-1</td>
</tr>
<tr>
<td><strong>APPENDIX - J</strong></td>
<td></td>
</tr>
<tr>
<td>PDL-Flowchart Questionnaire</td>
<td>J-1</td>
</tr>
<tr>
<td><strong>APPENDIX - K</strong></td>
<td></td>
</tr>
<tr>
<td>Simulator Design, Flowchart Form</td>
<td>K-1</td>
</tr>
<tr>
<td><strong>APPENDIX - L</strong></td>
<td></td>
</tr>
<tr>
<td>Simulator Design, PDL Form</td>
<td>L-1</td>
</tr>
<tr>
<td><strong>APPENDIX - M</strong></td>
<td></td>
</tr>
<tr>
<td>Simulator Design Revision Memo</td>
<td>M-1</td>
</tr>
<tr>
<td><strong>APPENDIX - N</strong></td>
<td></td>
</tr>
<tr>
<td>Design Comprehension Test</td>
<td>N-1</td>
</tr>
<tr>
<td><strong>APPENDIX - O</strong></td>
<td></td>
</tr>
<tr>
<td>Time Expenditure Questionnaire, Implementation Phase</td>
<td>O-1</td>
</tr>
<tr>
<td><strong>APPENDIX - P</strong></td>
<td></td>
</tr>
<tr>
<td>Examples of Subject-Generated Designs</td>
<td>P-1</td>
</tr>
<tr>
<td><strong>APPENDIX - Q</strong></td>
<td></td>
</tr>
<tr>
<td>Example of Job-Generated Simulator Program</td>
<td>Q-1</td>
</tr>
</tbody>
</table>
TABLES

Table 1. Summary of Experimental Conditions ........................................ 18
Table 2. Means and (in Parentheses) Standard Deviations of Reported Design Times (hours) .... 23
Table 3. Means and (Standard Deviations) of Design Quality Ratings .................. 26
Table 4. Analysis of Variance of Total Design Scores .................................. 29
Table 5. Means and (in Parentheses) Standard Deviations of "Design Style" and Level-of-Detail Measures 32
Table 6. Summary of Abbreviations Used in Pass 1 Designs .......................... 34
Table 7. Means and (in Parentheses) Standard Deviations of Questionnaire Ratings after Design Phase 38
Table 8. Results of Multiple-Choice Portion of Design Comprehension Test .......... 40
Table 9. Means and (Standard Deviations) of Implementation Quality Ratings 44
Table 10. Means and (Standard Deviations) of Questionnaire Ratings after Implementation Phase 47
## FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.</td>
<td>An Example of a PDL Specification</td>
<td>6</td>
</tr>
<tr>
<td>Figure 2.</td>
<td>Grading Criteria for Assembler Designs</td>
<td>25</td>
</tr>
<tr>
<td>Figure 3.</td>
<td>Mean Total Design Score, by Group</td>
<td>27</td>
</tr>
<tr>
<td>Figure 4.</td>
<td>Grading Criteria for Implemented Simulators</td>
<td>42</td>
</tr>
</tbody>
</table>
INTRODUCTION

In recent years, several trends have emerged which have caused the Department of Defense to direct increasing attention toward the investigation of the behavioral factors which affect the development of computer software. DoD software expenditures, which probably now exceed $5 billion annually, have been increasing both in absolute terms and as a percentage of total system development costs (Boehm, 1973). At the same time, the increasing size and complexity of software systems has caused software reliability to become a dominating problem in many modern weapon and support systems (Boehm & Haile, 1972). During the 1970s, there has been an emerging awareness, both in DoD and in the computer science community at large, that the control of software costs and quality will depend ultimately on an understanding of the complex behavioral factors affecting software design, programming, documentation, etc.

The quality of software documentation can significantly influence both the quality and cost of the software (to the extent that the documentation is used during the development effort) and its maintainability and modifiability. Software documentation is an area which has long been subject to DoD standards. A number of new software documentation methods have recently emerged (e.g., HIPO charts, structure charts, program design languages). DoD is now considering recommendations that some of these techniques be incorporated more widely in its standards. One of these techniques (Program Design Languages) is the subject of this study.

The importance of documentation seems to be widely recognized in the software development community. For example, when programming managers were asked to identify, via a two-stage Delphi technique, those variables which most affect programmer productivity, documentation quality was rated the most important factor (Scott & Simmons, 1974). Yet, in spite of this
consensus about the importance of good documentation, there is considerable disagreement concerning the form it should take.

One of the reasons for the wide diversity of documentation types is the diversity of purposes for which software documentation can be used. Judd (1972) advocates the development of program design documentation, error diagnosis documentation, operational documentation, maintenance documentation, development documentation, and marketing documentation, all as separate products of a software development effort. It is undoubtedly true that documentation methods appropriate for some of these purposes may not be appropriate for others. This report focuses upon "detailed program design" documentation, typically developed by a software system designer and given to a programmer as the specification of his programming task. It is the formal medium of communication between the designer and the programmer.

Flowcharts have long been accepted in the computer science community as the standard medium for detailed program design documentation, as well as for archival documentation for later use by maintenance programmers. Recently, however, the use of flowcharts for archival documentation has been questioned by a number of critics, who suggest that flowcharts may not aid program comprehension (Weinberg, 1971), or error diagnosis (Aron, 1974), and that they are an unnecessary drain on project resources (Brooks, 1975). White (1975) has suggested that modern programming languages provide a more powerful and concise documentation medium, in their own right, than do flowcharts which attempt to convey the procedural details of a program.

Shneiderman et al. (1977) performed a series of experiments to determine whether detailed flowcharts might improve the programmer's performance in any of several programming tasks. Flowcharting before
coding did not improve the quality of the resulting program, as graded by a graduate assistant. When programs were presented with or without flowcharts, flowcharts did not result in significantly improved performance on either comprehension tests or modification tasks. In one study, comprehension even appeared to be degraded by the presence of a flowchart.

In a study in which some subjects had prior experience with flowcharts while others did not, the experienced subjects performed somewhat better with flowcharts, while the inexperienced subjects did not.

Flowcharts and similar documentation methods have also been studied outside the software context. Kammann (1975) reported an experiment involving a telephone dialing task in which subjects (housewives and technical personnel) performed better when it was presented in the form of a flowchart than when presented in the form of a text description. An experiment by Wright and Reid (1973) has identified what may be a key task variable in these studies. In their experiment, subjects were required to memorize a procedure or to execute the procedure by hand without necessarily memorizing it. When the subjects hand-executed the procedure while its description was still available to them, performance was better for procedures described by flowchart or decision table than for those described by prose or a series of "short sentences." But when the subject's task was execution of the procedure from memory, "short sentences" were best, and performance with flowcharts was rather poor. The authors suggest that rehearsal effects may account for this difference, since material presented verbally may be more readily rehearsed.

Wright and Reid suggest that information presented in flowchart form is, at least in part, encoded visually, with the result that successful transfer to long-term memory is hindered. On the other hand, information presented in short-sentence form is encoded verbally and is more likely to be successfully integrated into long-term memory. The mechanism suggested by Wright and Reid (dual encoding with differential rehearsal effects)
seems plausible (cf. Craik, 1973; Craik and Lockhart, 1972; Paivio, 1969). In any event, the study certainly suggests that these two information presentation forms have different retention patterns. It should be noted, however, that exposure time was limited in this study, and that the two presentation forms, though equal in logical information content, differed somewhat in information structure.

In general these results suggest that flowcharts may aid hand-execution of a procedure, but may not aid comprehension of the procedure. Shneiderman et al. (1977) present some data which support this hypothesis. In one study, they report separately "comprehension" questions which involve hand execution and questions which involve interpretation of the programs. Subjects who were given both flowcharts and programs did about as well on hand-execution items as those who were given only the programs. On interpretation items, though, the program-only group performed somewhat better.

Unfortunately, it may not be legitimate to generalize these findings to large software projects or to summary design documentation. Those studies which have addressed programming tasks have not only been restricted to relatively inexperienced student programmers, but have also investigated only relatively small, simple programs. Even if the program itself constitutes a better form of archival documentation than does a flowchart, the size of a program may be a significant factor. Beyond a certain size, a program(s) cannot be read directly by one person, and summary documentation of some sort is, therefore, required in large software projects. Whether flowcharts or some other form are best for this purpose is not known. Even for programs of intermediate size, the reader might be aided by such summary documentation.

Although the referenced studies have been concerned primarily with archival documentation, rather than with designer-to-programmer
design communication, they may have implications for the latter. The data suggest that a computer program, expressed in a high-level programming language, is more comprehensible than the corresponding flowchart. If this is true, then an artificial design language, with a programming-language-like syntax, might also be preferable to flowcharts for the expression of software design information. Such languages, commonly called Program Design Languages (PDLs) have been discussed in the literature for several years. PDLs have gradually evolved from the rather informal use of "pseudo-code" to express designs or algorithms. Gradually, this form of documentation became more formal, evolving into a PDL form. Although a few presumably successful case studies have been reported without substantiating data (e.g., Van Leer, 1976), the authors are unaware of any experimental studies bearing directly on the issue of PDL efficacy.

Figure 1 is an example of PDL specification for a program which computes social security withholding (FICA) amounts from a payroll data base, and prints a report of those values. This example is taken from a report (Kraly et al., 1975) concerning program design methods and documentation techniques recommended for adoption by DoD.

PDLs can vary greatly in the degree of formality. At one extreme, is a "freeform" PDL, such as that in Figure 1. In a freeform PDL, there is a minimum of syntactic and semantic constraints; the terms and expressions that are used are determined by the user. In a "formal" PDL, the syntax and semantics are highly constrained; the user is restricted to using keywords (e.g. READ, WRITE, etc.) and elements of an expression may be syntactically identified by underlines, parentheses, etc. In general, a formal PDL is more precise than an informal PDL, but it is also more difficult to learn and use. Formal PDLs tend to be preferred especially when the design specification is subject to direct computer processing, as for consistency checking.
PRINT FICA REPORT HEADER

OBTAIN FICA PERCENT AND FICA LIMIT FROM CONTRAINTS FILE

SET FICA TOTAL TO ZERO

DO FOR EACH RECORD IN SALARY FILE

   OBTAIN EMPLOYEE NUMBER AND TOTAL SALARY TO DATE

   IF TOTAL SALARY IS LESS THAN FICA LIMIT THEN

      SET FICA VALUE TO TOTAL SALARY TIMES FICA PERCENT

   ELSE

      SET FICA VALUE TO FICA LIMIT TIMES FICA PERCENT

   ENDIF

   PRINT EMPLOYEE NUMBER AND FICA VALUE

   ADD FICA VALUE TO FICA TOTAL

ENDDO

PRINT FICA TOTAL


Figure 1. An Example of a PDL Specification
There are several reasons for believing that PDLs and flowcharts may differentially affect performance in design, design documentation, and design comprehension tasks. First, information presented in these two media may be encoded in memory in different ways, at least with limited exposure time. Notice that PDL design specifications are very similar to the "short sentences" used by Wright and Reid (1973), while flowcharts are similar to their "algorithms."

Secondly, PDL and flowchart forms may differ in the processing effort required to encode them in memory even if they are encoded similarly. PDL forms are not only already verbal (and thus, perhaps, more easily integrable into long-term memory), but also appear to be more compatible with the hierarchic propositional memory representation hypothesized by Atwood and Ramsey (in press) for program comprehension. Program documentation which has a similar structure to the representation of the program in memory may require less processing in order to be encoded in memory. Given sufficient processing time, however, flowchart information might be encoded in the same way as PDL information, even if they are encoded differently with brief exposure times, as suggested by Wright and Reid in (1973) results. In most actual software development situations, exposure time is not highly constrained; thus, if this is the only difference between PDLs and flowcharts from the point of view of comprehension, it may result in only a mild preference for one technique over the other.

Third, PDLs and flowcharts may emphasize different properties of the underlying software design. At an obvious level, flowcharts appear to emphasize flow of control, while PDLs may have a greater emphasis on program structure. This may affect the performance of the designer, as well as the programmer. From the designer's viewpoint, it may be that different classes of design problems are more easily detectable with different design documentation formats. More basically, it
may simply be easier to express the properties which the designer considers important in one way than in the other. It is even conceivable that some types of design are more amenable to PDLs, while others are more compatible with flowcharts.

From the viewpoint of the programmer, the form in which the programming problem is expressed may very well affect his basic understanding of the problem. In various problem-solving tasks, it has been shown that the perceptual process is important to the formation of initial conceptual problem structures (Simon & Barenfield, 1969), and that the form in which a problem is expressed can thereby affect problem-solving strategy (Simon & Hayes, 1976) and performance (Jeffries et al, 1977; Simon & Hayes, 1976).

Fourth, and finally, it seems fairly clear that PDLs have some practical advantages, such as decreased production costs and easier maintainability, relative to flowcharts and other graphical techniques (Ortega, 1974).

Thus, an analytical comparison of PDLs and flowcharts would appear, overall, to favor PDLs for detailed design documentation. Only empirical evaluation, however, can provide really convincing evidence in favor of one or another technique. The following sections will present an experimental study performed to obtain such evidence.

The experiment reported here was intended to provide empirical evidence which might assist in the selection of flowcharts or PDLs as a medium for detailed program design documentation. The experiment was intended to investigate the effects of documentation format both on performance of the designer in a software design task and on performance of the programmer in a program implementation task. Although the study was confined to one-person design and implementation efforts, the tasks
(especially the implementation task) were much larger than those which have typically been employed in behavioral studies of software development, and were fairly "realistic" in terms of both program size and programmer experience. The study attempted to observe the effects of documentation format on design quality, detailed design properties, designer effort, design comprehension by the programmer, resulting program quality and error properties, programmer effort, and subjective preferences of the participants.
METHOD

OVERVIEW

Twenty students in a computer science graduate course participated in the experiment. Working individually, the students designed a two-pass assembler for a simple minicomputer. Half the students expressed their design for the first pass of the assembler in the form of a flowchart, and expressed their design for the second pass in a PDL. The other half of the students used a PDL for pass one, and a flowchart for pass two. This 2 X 2 repeated-measures Latin square design was employed in an effort to correct for individual differences and to increase the statistical power which could be achieved with the limited number of subjects available. Although this design has the disadvantage that any chance group difference is confounded with interactions, it was felt that the advantages outweighed this disadvantage. Flowcharts and PDLs were compared on the basis of various measures of overall design quality, design errors, level of detail of designs, time expended in developing designs, and subjective preferences.

Having completed this design task, the subjects then performed an implementation task. They were given fairly detailed procedural designs for a program which simulates the function of a fairly sophisticated minicomputer. They were then required to develop a working version of the program in PL/1. Although the designs were logically equivalent, half the students received their simulator design in flowchart form, and half in PDL form. Subjects from each design-phase group were divided equally into these two implementation-phase groups. Flowcharts and PDLs were compared on the basis of design comprehension test performance, various measures of overall implementation quality, implementation errors, time expended in developing implementation, and subjective preferences.
SUBJECTS

Subjects were 1 female and 19 male graduate students at Oklahoma State University (OSU). With one exception, subjects were Master of Science candidates majoring in Computer Science and enrolled in the course, "Computer Structure and Programming," described below. One student, who later withdrew from the course, was majoring in engineering. Because of the location of OSU away from large urban centers, and possibly other factors, students in this curriculum tend to be full-time students with no outside (non-university) employment. Subjects ranged in age from 21 to 40, with a mean age of 26.1 years. They had completed, on the average, 8.9 computer science courses (undergraduate and/or graduate) in addition to current enrollment (range 4-16).

Based on previous knowledge of the faculty, two subject factors were identified as sufficiently relevant to the subjects' performance in the study to warrant a special effort to balance the effects of these factors across experimental groups. These factors were native language (a significant number of foreign students attend OSU) and participation in a particular graduate student curriculum, discussed below. Five of the subjects were not native speakers of English. Two of these five were considered completely fluent in English by their instructor. The remaining three were considered competent in English; all three had taken prior degrees at English-language universities. As a precautionary measure, however, these three subjects were assigned (at random) to different experimental groups.

Three of the subjects were pursuing a specialized curriculum (the "Programming Language Option"), which emphasizes compilers, assemblers, and machine simulators. Since the design and programming problems used in the experiment involved an assembler and a machine simulator, these three subjects were assigned (at random) to different
experimental groups. With the two exceptions just described, subject assignment to experimental groups was entirely random.

The particular course in which the experiment was performed was entitled, "Computer Structure and Programming." As described in the department's brochure, the course covers the areas of:

- Computer structure, flow of control, arithmetic and logical operations.
- Machine instructions, indexing, addressing, linkages and input-output.
- Assembly, macro and emulation-simulation systems.

This particular required course has a reputation, among the students, of being the most difficult course in the Master's curriculum.

PRELIMINARY PHASE

The first phase of the experiment was introductory in nature and involved the same procedures for all students. The experiment began at a class session approximately 3 weeks after the start of the course (a detailed schedule of events can be found in Appendix A). In this class session, the students were given an introduction to the experiment, as well as an introduction to the concept of Program Design Languages, and were asked to provide background information describing their computer science experience and some personal characteristics.

The lecture began with a brief discussion of the nature of the experiment. A concerted effort was made to introduce the experiment in an unbiased way. Statements of specific hypotheses were avoided, and the experiment was described as an attempt to compare two techniques, each of which probably has both advantages and disadvantages. It was pointed out that participation was entirely voluntary, that the experiment was entirely consistent with course goals and would involve little additional effort by the subjects, and that subjects completing all
phases of the experiment would be paid $25. A more detailed impression of the way in which the experiment was described to the students can be gained by reading the consent form which they were given (see Appendix B).

Because flowcharts were the standard mechanism for the procedural description of programs in OSU courses prior to this experiment, most of the subjects had had little prior exposure to PDLs. Thus, it was necessary to provide them with sufficient training in the use of PDLs that they would be able to generate and comprehend specifications expressed in this form. At the same time, it was obviously desirable to minimize the biasing effect that might result from a training session devoted purely to a discussion of PDLs. We attempted to satisfy both of these objectives by presenting parallel discussions of PDLs and flowcharts. After the basic concept of a Program Design Language had been introduced, further discussion was directed primarily toward illustrating how a particular construct might be expressed, both in PDL and in flowchart form. This presentation took the form of a discussion of two handouts (Appendices D and E) which were given to the students at the start of the class session. The discussion was recorded on audio tape for later reference.

Appendix D served the dual functions of introducing the concept of a PDL to the student and of defining the particular PDL (and flowchart) conventions which were to be used in both the design and implementation stages of the study. The particular PDL used was a relatively unstructured or "freeform" PDL (cf. Kraly et al, 1975). That is, the user was required to use a particular basic syntax involving a particular set of basic structural constructs (PROCEDURE...END, IF...THEN...ELSE, DO...END, etc.), but was relatively unconstrained with respect to the particular statements which might appear within these basic structures. He might say INCREMENT ITERATION_COUNT, ADD ONE TO ITERATION_COUNT, UPDATE ITERATION_COUNT, ITERATION_COUNT = ITERATION_COUNT +1, etc. The
particular PDL used in the study was chosen to correspond in many respects to features of PL/1, with which students were familiar. Similarly, the basic set of flowchart symbols available to the user was specified, and consisted of the flowchart symbols in general use in OSU computer science graduate courses. The particular statements which might appear within flowchart symbols were relatively unconstrained, however.

The second handout (Appendix E) contained the specification for a very simple minicomputer simulator, expressed in both flowchart and PDL form. This handout, and its discussion, served not only to provide a concrete example of both flowchart and PDL specifications, but also to introduce the students to some of the concepts involved in machine simulators.

When the PDL-flowchart presentation had been completed, each student was given two consent forms (Appendix B) and a background questionnaire (Appendix C). The students were asked to sign the consent forms, fill out the questionnaire, and return them at the following class session, if they wished to participate in the experiment. All 20 students agreed to participate, and returned all of the requested forms.

DESIGN PHASE

In the design phase of the experiment, the subjects were given the task of designing a two-pass assembler for a minicomputer. The particular minicomputer involved was a hypothetical machine (the "small computer" which had been defined and used as an example earlier in the course. The subjects were presumably thoroughly familiar with the hardware instruction set of this machine. Earlier in the course they had been instructed in the use of the programming language APL as a notation for concise, rigorous definition of hardware instruction sets. As an exercise, they had then developed a complete specification, in APL, of the instruction set of this particular machine.
In addition to this familiarization with the machine, the students had been given instructions concerning the design of assemblers as part of the course in which the experiment was conducted. No prior discussion had occurred with respect to the possible properties of an assembler, or of an assembler language, for the "small computer," however. At the start of the design phase of the experiment, the subjects were given a handout (Appendix F) which described this assembler language, and which instructed the students that they would be required to design a two-pass assembler, documenting one pass via flowchart and the other pass via PDL.

Along with the general assembler design problem description, the subjects received a brief description of the responsibilities of the first pass of the assembler (Appendix G). These two handouts constituted the entire problem description for pass 1. All subjects designed the pass 1 program first, but half of the subjects were told to prepare their designs in flowchart form, and half in PDL form. A third handout (Appendix I) accompanied the problem description. This was a form on which the subjects were asked to record the amount of time expended on the project in each of several categories (initial design preparation, design revision, hand-checking, preparation of documentation, etc.). The subjects were told to prepare their designs and turn them in with the time questionnaire, whenever they were completed, at which time they would be given pass 2 instructions. They were told that they had at most five days to complete the first pass, and that the second pass design would be due seven days after the start of the design phase.

When a subject turned in his pass 1 design, he was given a description of the responsibilities of the pass 2 program (Appendix H), along with a time questionnaire essentially identical to that shown in Appendix I. Those subjects who had prepared their pass 1 designs in flowchart form were instructed to use the PDL form for pass 2, and vice-versa.
After completion of the pass 2 designs, subjects were given a take-home questionnaire (Appendix J) which asked them to subjectively compare flowcharts and PDLs with respect to their overall utility to the software designed and to the programmer, and a number of more detailed criteria.

In addition to the time and subjective preference data, finished designs were graded on overall quality by an expert judge, and were analyzed for the nature of design errors, level of detail of the design, and other properties. This judge was an associate professor on the computer science department faculty, very knowledgeable about software design techniques and methods, and highly experienced in developing assemblers and translators.

IMPLEMENTATION PHASE

In the implementation phase, the subjects were given a program design, expressed in either flowchart or PDL form, and were required to develop a working version of the program in PL/1. This implementation effort was the major term project for this course. The program was a simulator for the Data General MicroNova minicomputer. As minicomputers go, the instruction set and execution behavior of the MicroNova are fairly complicated. The simulator, which was to run on an IBM 360, was intended to correctly execute MicroNova machine language programs, with the exception that console operations were excluded and input/output device control was somewhat simplified. In addition, the simulator design included some specialized capabilities (Trace, Dump, etc.) which were associated with the simulator itself.

Prior to the start of the experimental portion of the implementation phase, the students studied the properties of the MicroNova, in order to gain a thorough understanding of the machine itself. This study included lectures, reading of MicroNova reference documents (e.g., Data
General Corp., 1976), and, in some cases, additional literature research initiated by the students. Thus, at the time the simulator designs were distributed, the students were presumably thoroughly familiar with the MicroNova, but had no knowledge of the detailed simulator design, which described the program they were to implement.

Half the subjects were given a simulator design expressed in flowchart form (Appendix K), and half were given a design in PDL form (Appendix L). Although several minor design errors were uncovered later and were corrected by issuing an errata sheet (Appendix M), those errors were essentially equivalent in the two documentation forms. The errata sheet was issued 12 days after distribution of the designs. A strong effort was made to ensure complete logical equivalence of the two versions of the simulator design, and only one minor logical difference between the two versions has subsequently been detected. This difference was discovered and corrected 42 days after distribution of the designs.

In order to control for any residual differential effect of the earlier ("design phase") study which might bias the subjects' performance in this second experiment, the subjects from each of the earlier groups were divided equally between the two experimental groups used in this phase of the study. Considered overall, then, there were four separate groups of 5 subjects each (ignoring attrition), which were exposed to the experimental conditions illustrated in Table 1. Although all 20 subjects successfully completed the design stage of the experiment, 4 subjects withdrew from the course, or withdrew altogether from school, and therefore failed to complete the implementation phase, as indicated in the table.

Simulator designs were handed out at the beginning of a class session. The subjects were instructed to study the designs for 45 minutes in preparation for a comprehension test. Then, in an attempt to determine
TABLE 1. Summary of Experimental Conditions

<table>
<thead>
<tr>
<th>Group</th>
<th>Number Starting</th>
<th>Number Completing</th>
<th>Order of Conditions in Design Phase</th>
<th>Condition in Implementation Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>4</td>
<td>PDL-Flowchart</td>
<td>PDL</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>5</td>
<td>PDL-Flowchart</td>
<td>Flowchart</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>4</td>
<td>Flowchart-PDL</td>
<td>PDL</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>3</td>
<td>Flowchart-PDL</td>
<td>Flowchart</td>
</tr>
</tbody>
</table>
whether PDLs and flowcharts have any detectable differential effect on early design comprehension, a comprehension test (Appendix N) was administered. The subjects were allowed 35 minutes to complete this test. The test contained a variety of items intended to provide measures of several comprehension factors:

- understanding of MicroNova function
- ability to distinguish between MicroNova and simulator functions
- comprehension of modular structure of simulator design
- comprehension of procedural operation of simulator
- ability to pinpoint probable modular location of error causing specified simulator malfunction
- ability to specify changes needed to cause a specified change in simulator function.

(A second administration of this test, which was to have occurred after one week of study of the designs by the subjects, was inadvertently delayed until after the subjects had completed their program implementation task.)

After the designs were distributed, the subjects were given 8 1/2 weeks to develop a complete, working simulator. They were allowed to establish their own schedules for completing this task. Each subject was required to maintain, and turn in at the end of the project, a project folder. The folder contained the entire printout of each computer run made in connection with the project. Whenever new source programs were compiled, listings of those programs appear in these printouts, along with any error messages issued by the compiler, link/editor, etc. Input and output of simulator test runs were also included, as was the computer's job summary for each run. These printouts thus provide data for determination of simple summary statistics, such as number of runs, CPU utilization, etc., as well as more substantive analyses of error types and locations, etc.
Each project folder also contained a "run log," in which the students recorded the purpose of each run, nature of errors encountered, etc. This is a fairly standard practice at OSU, and was included in this project as an aid to later detailed analysis of the individual runs. Finally, the project folders contained a weekly time questionnaire (Appendix O) on which subjects indicated, by category, the time they had expended on the project in the preceding week.

In addition to the project folder, subjects turned in, at the end of the project, their finished simulator source programs in computer-readable form. A very detailed test procedure was run on each of these programs in an effort to assess correctness of simulator function and to detect those kinds of conceptual or semantic errors which the experimenters were able to anticipate a priori.

After the subjects had completed the implementation project and turned in their final programs and project folders, the simulator design comprehension test (Appendix N) and the PDL-Flowchart preference questionnaire (Appendix J) was readministered.

In addition to the various qualitative and quantitative measures already mentioned, the final simulator programs were rated on overall quality by an expert judge, (previously described) who also performed fairly detailed error analyses of the final programs.
RESULTS

DESIGN PHASE

The allocation to experimental groups of non-native-English-speakers and of students in a specialized programming-language curriculum was explicitly constrained. These special inter-subject differences were dichotomous and affected small numbers of students, so that the assignment of two such subjects to a single group might significantly bias the study. Other differences were assumed to be controlled by the otherwise random subject-to-group assignment. Unfortunately, this random assignment resulted in a difference in average ability between the two groups in the design phase of the study. This difference is reflected, for example, in their Grade Point Averages (GPAs) for all computer science graduate courses.

There is at least a suggestion, in the available data, that this group difference may be related more to ability than to experience. The two groups are almost identical with respect to average number of computer science courses taken (8.7 for the group designing the first pass using a PDL versus 9.1 for the other group), average number of graduate computer science courses taken (4.2 versus 3.5), and number reporting past experience with translator implementation and related tasks (5 each).

For convenience of reference, let us refer to the group which used a PDL for pass 1 of the assembler, and flowcharts for pass 2, as the PDL-FL group. The other group will be referred to as the FL-PDL group. Then the mean GPA for the PDL-FL group was 2.986, while the FL-PDL mean was 3.587. Standard deviations were .606 and .334, respectively. Statistical tests indicate significant differences in both the means (t(15) = 2.71, p < .02) and variances (F(9,8) = 3.30, p < .10). The PDL-FL group included several students whose overall performance was relatively low, both within
this course and in graduate school in general, and who had no counterparts in the FL-PDL group, thus accounting for both differences.

This group difference does not greatly interfere with our ability to analyze the study results and draw reasonable conclusions. No such difference occurred between experimental groups in the implementation phase of the study. The use of a Latin square design in the design phase of the study sharply reduces the impact of the group difference on the analysis of that phase. In fact, the Latin-square analysis of variance employed for most of the design-phase analyses is conservative in this situation (McNemar, 1969, p. 387). The group difference should be kept in mind by the reader, however. If either flowchart or PDL is consistently preferable in the design of both pass 1 and pass 2 of the assembler, such a difference can be detected by statistical analysis. However, if for some reason a PDL is better for one pass and flowcharting for the other, perhaps due to differences in the design tasks, that interaction is confounded with the group differences in this study and cannot be detected. Based on a careful consideration of the design tasks in passes 1 and 2, no such interaction was expected.

Time to Produce Designs

Table 2 presents a summary of the students' responses to the design phase time expenditure questionnaire. On the average, the students reported spending about 7.6 hours on the pass 1 design task and about 5.4 hours on pass 2. The reported times for pass 1 include some general problem familiarization not repeated in pass 2. No statistical differences exist between PDL or flowchart conditions or between experimental groups on either the total time or any of its components. It should be noted that subjective reports of time expenditures are not generally a very reliable or sensitive measure. In any event, these data provide no basis for concluding that either documentation technique requires more time than the other.
### Table 2. Means and (in Parentheses) Standard Deviations of Reported Design Times (hours)

<table>
<thead>
<tr>
<th>Group</th>
<th>Preparing Initial Design</th>
<th>Revising Design</th>
<th>Hand Checking Design</th>
<th>Preparing Documentation</th>
<th>Just Thinking About Project</th>
<th>Other</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PDL</strong>*</td>
<td>2.18 (1.15)</td>
<td>1.96 (1.46)</td>
<td>.66 (0.57)</td>
<td>.54 (0.63)</td>
<td>1.82 (2.14)</td>
<td>.73 (0.93)</td>
<td>7.90 (3.81)</td>
</tr>
<tr>
<td><strong>FL</strong></td>
<td>2.37 (1.04)</td>
<td>1.53 (0.59)</td>
<td>.43 (0.24)</td>
<td>.50 (0.61)</td>
<td>1.99 (1.36)</td>
<td>.40 (0.70)</td>
<td>7.23 (2.79)</td>
</tr>
<tr>
<td><strong>FL</strong></td>
<td>1.42 (0.80)</td>
<td>1.83 (1.03)</td>
<td>.53 (0.50)</td>
<td>.68 (0.63)</td>
<td>.93 (0.92)</td>
<td>.19 (0.36)</td>
<td>5.57 (2.50)</td>
</tr>
<tr>
<td><strong>PDL</strong></td>
<td>1.88 (0.73)</td>
<td>1.36 (0.88)</td>
<td>.38 (0.18)</td>
<td>.26 (0.31)</td>
<td>1.24 (0.99)</td>
<td>.20 (0.35)</td>
<td>5.31 (1.73)</td>
</tr>
</tbody>
</table>

* Group 1 subjects used a PDL for pass 1, a FL for pass 2.
** Group 2 subjects used a FL for pass 1, a PDL for pass 2.
Quality and Properties of Designs

Unlike an implemented program, which can, at least conceivably, be evaluated by subjecting it to objective test, the quality of unimplemented software designs can be assessed only by subjective evaluation. Furthermore, in this particular study, the use of different documentation methods precluded "blind" ratings. Clearly, the individual evaluating the design would be aware of the experimental condition in which the design was generated. Mapping all the designs into a common notation was considered, but was not feasible without at least reducing the "naturalness" of the transformed designs.

Within these constraints, quality ratings were obtained quite carefully, with considerable attention by the rater (JRV) to the possibility of experimenter bias. A priori grading criteria were developed (Figure 2) and applied to the designs. The resulting design quality scores are summarized in Table 3. Because the pass 1 and pass 2 assembler designs were dissimilar in function, the grading categories do not correspond. The total scores, however, are intended to be comparable in significance, and are on a scale from zero to 100, where a higher score represents better performance.

Although it would have been preferable to use additional raters, both to improve rating reliability and to provide a measure of that reliability, the only other available person with the requisite qualifications was the course instructor. Since it was necessary to separate the administration of the experiment from the course being taught, the instructor could not be used as a rater. Since this task requires a large amount of time and resources, additional raters were not sought.

Figure 3 provides a graphical illustration of the effect of the experimental conditions on total design score. A corresponding analysis
Pass 1 grading categories
100 point total
1) 20 points
   Location counter logic
2) 30 points
   Error detection
      Invalid operation codes
      Labels - required
      Multiple label definitions
      Out of range location counter
      Premature end of file
3) 20 points
   Symbol table entry for defined symbols
4) 15 points
   Output for input to pass 2
5) 15 points
   General logic

Pass 2 grading categories
100 point total
1) 20 points
   Error detection
      Undefined Symbols
      Out of range absolute addresses
      and constants
      Premature end of file
2) 25 points
   Object word creation
3) 40 points
   Text buffer management
      Initialization
      Conditions for buffer output
      Logic for building buffer
      Trailer record
4) 15 points
   General logic

Note: "General logic" categories are intended for factors which do not fall under the other specified categories.

Figure 2. Grading Criteria for Assembler Designs
Table 3. Means and (Standard Deviations) of Design Quality Ratings

<table>
<thead>
<tr>
<th>Category (see Figure 2)</th>
<th>Group</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PDL</td>
<td>13.3</td>
<td>18.7</td>
<td>14.2</td>
<td>10.5</td>
<td>10.8</td>
<td>67.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(6.41)</td>
<td>(6.85)</td>
<td>(6.53)</td>
<td>(3.03)</td>
<td>(3.12)</td>
<td>(22.71)</td>
</tr>
<tr>
<td></td>
<td>FL</td>
<td>17.6</td>
<td>18.9</td>
<td>13.1</td>
<td>10.8</td>
<td>11.6</td>
<td>72.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(5.25)</td>
<td>(5.20)</td>
<td>(5.57)</td>
<td>(3.12)</td>
<td>(1.84)</td>
<td>(10.68)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Category</th>
<th>Group</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FL</td>
<td>9.2</td>
<td>12.7</td>
<td>17.6</td>
<td>7.7</td>
<td>47.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(5.07)</td>
<td>(7.38)</td>
<td>(12.76)</td>
<td>(4.16)</td>
<td>(27.76)</td>
</tr>
<tr>
<td></td>
<td>PDL</td>
<td>12.3</td>
<td>19.0</td>
<td>30.5</td>
<td>11.4</td>
<td>73.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(4.57)</td>
<td>(3.83)</td>
<td>(5.58)</td>
<td>(1.35)</td>
<td>(11.25)</td>
</tr>
</tbody>
</table>
Figure 3. Mean Total Design Score, by Group
of variance is shown in Table 4. Rated performance is significantly lower on pass 2 than on pass 1 (p < .05). These design tasks are dissimilar, although related, and we had no particular expectation with respect to their relative difficulty. The grading categories and criteria are necessarily different, and any temporal effects are confounded with the pass 1-pass 2 difference. As one might expect, given the difference in ability between the groups (more precisely, the difference in GPAs), the FL-PDL group performed somewhat better, although not significantly better, than the PDL-FL group (p < .10).

The most relevant result of this analysis, however, is the significant difference in performance as a function of documentation method (p < .025). Overall performance was better when the design was specified in PDL form than when it was specified in flowchart form.

It should be noted that for Latin square designs, such as that used here, it is assumed that all interactions will be zero. This is a necessary assumption, since main effects are confounded with interactions, and there are not enough degrees of freedom to statistically separate the interaction effects. For example, in the present design, any PDL-FL vs. FL-PDL group difference is not separable from a documentation-method-by-pass interaction. Although such an interaction may be theoretically very interesting, the observed significant difference between the GPAs for these two groups leads us to conclude that this is, in fact, a group difference and not an interaction.

In either case, however, whether this is interpreted as a group difference or an interaction, its presence reduces the probability of observing significant effects on the other variables, including documentation method. That is, this results in a conservative analysis and the actual significance levels obtained for documentation method are somewhat lower than those reported, although it cannot be determined how much lower (cf. McNemar, 1969, pp. 383-387).
Table 4. Analysis of Variance of Total Design Scores

<table>
<thead>
<tr>
<th>Factor</th>
<th>df</th>
<th>Mean Square</th>
<th>F</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Between Groups:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Experimental Group</td>
<td>1</td>
<td>2325</td>
<td>3.82</td>
<td>&lt; .10</td>
</tr>
<tr>
<td>Subjects w/i Groups</td>
<td>18</td>
<td>10952</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Within Groups:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assembler Pass</td>
<td>1</td>
<td>912</td>
<td>5.89</td>
<td>&lt; .05</td>
</tr>
<tr>
<td>Documentation Method</td>
<td>1</td>
<td>1156</td>
<td>7.47</td>
<td>&lt; .025</td>
</tr>
<tr>
<td>Residual</td>
<td>18</td>
<td>155</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Based on the total design scores, a single design was selected from each of the four groups for inclusion in this report. These designs constitute Appendix P, and represent the most nearly "average" design (in terms of design score only) from each condition.

The designs were analyzed for differences in style, level of detail, and types of constructs used which might be related to documentation method. Factors analyzed included the number of modules used in the design, the number of Boolean expressions (conditional tests), subroutine call operations, and other executable operations used, and the extent and type of abbreviation used in the design. The results, summarized in Table 5, are discussed in the following paragraphs.

PDL and flowchart conditions differed somewhat with respect to the number of modules defined in the designs. Subjects tended to use more modules in designs produced in PDL form than in designs produced in flowchart form ($F(1, 18) = 3.436, p < .10$).

It seems possible with different documentation methods that designers might vary the kinds of information contained in the design and the constructs used to express it. Such differences are very difficult to detect in an objective manner. Several manual scoring algorithms were developed and rejected for measuring usage of particular design constructs (number of conditional expressions, transfers of control, call statements, etc.) before adopting one which seemed meaningful, unbiased, and relatively unambiguous when applied to either design documentation type. By way of illustrating the problems encountered, consider transfer of control operations. In the context of either documentation type considered in isolation, it is relatively easy to count transfer of control operations. In a PDL, one might count GOTO and RETURN statements, while in flowcharts one might count RETURN operations and arcs emanating from decision diamonds. The difficulty is that these measures are not equivalent, across documentation methods, in any meaningful sense. Many
transfer-of-control operations which are represented explicitly in a
flowchart may be represented implicitly in a PDL (as in an IF...THEN
statement, DO WHILE, procedure block with an END but no RETURN, etc.).
There are other complicating considerations too detailed to warrant
discussion here.

The approach actually employed was to count the conditional
(Boolean) expressions, call operations, and other executable operations
in each design. Conditional expressions are easily recognized in dec-
sion diamonds and DO WHILE blocks in flowcharts and in IF...THEN, DO
WHILE, and similar constructs in a PDL. The number of conditional
expressions is probably related to the number of transfers of control
required to implement the design, although it may clearly be affected by
the level of detail of the design. The number of subroutine call opera-
tions might be related to modular structure, but may also be affected by
the provision of an explicit called-subroutine convention in the flow-
chart method used. The three categories, taken together, include all
the executable operations of each design, but exclude such purely
structural statements as noniterative DO, END statements, etc. The sum
of the three counts is the best metric we were able to devise for
measuring the level of detail of the designs.

Table 5 contains the results of this analysis. The principal
difference observed was a highly significant tendency for PDL designs
to contain more conditional expressions than flowchart designs
\(F(1,18) = 11.44, p < .005\). Although the PDL-FL and FL-PDL groups
differed in their frequency of use of call operations \(F(1,18) = 7.21, \ p < .05\)
and other executable operations \(F(1.18) = 4.54, p < .05\),
frequency of use of these constructs was not related to documentation
type. PDL-FL and FL-PDL groups also differed with respect to total
number of constructs used \(F(1,18) = 6.28, p < .05\), but on that measure
there was also a documentation type difference \(F(1,18) = 4.30, p < .06\).
Table 5. Means and (in Parentheses) Standard Deviations of "Design Style" and Level-of-Detail Measures

<table>
<thead>
<tr>
<th>Factor</th>
<th>Pass 1 FL</th>
<th>Pass 1 PDL</th>
<th>Pass 2 PDL</th>
<th>Pass 2 FL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of modules</td>
<td>3.50</td>
<td>3.40</td>
<td>4.80</td>
<td>2.20</td>
</tr>
<tr>
<td></td>
<td>(2.68)</td>
<td>(1.58)</td>
<td>(3.12)</td>
<td>(1.62)</td>
</tr>
<tr>
<td>Constructs used</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conditional expressions</td>
<td>15.8</td>
<td>16.3</td>
<td>19.0</td>
<td>10.9</td>
</tr>
<tr>
<td></td>
<td>(5.96)</td>
<td>(8.22)</td>
<td>(6.04)</td>
<td>(6.38)</td>
</tr>
<tr>
<td>Call operations</td>
<td>9.50</td>
<td>3.60</td>
<td>8.20</td>
<td>3.00</td>
</tr>
<tr>
<td></td>
<td>(7.62)</td>
<td>(3.41)</td>
<td>(6.30)</td>
<td>(2.67)</td>
</tr>
<tr>
<td>Other executable operations</td>
<td>37.4</td>
<td>31.7</td>
<td>40.5</td>
<td>28.2</td>
</tr>
<tr>
<td></td>
<td>(8.04)</td>
<td>(12.82)</td>
<td>(13.97)</td>
<td>(9.70)</td>
</tr>
<tr>
<td>Total</td>
<td>62.7</td>
<td>51.8</td>
<td>67.7</td>
<td>42.1</td>
</tr>
<tr>
<td></td>
<td>(17.9)</td>
<td>(18.3)</td>
<td>(20.0)</td>
<td>(16.1)</td>
</tr>
<tr>
<td>Number of variable names abbreviated</td>
<td>7.70</td>
<td>2.90</td>
<td>6.40</td>
<td>4.80</td>
</tr>
<tr>
<td></td>
<td>(6.00)</td>
<td>(3.11)</td>
<td>(4.86)</td>
<td>(3.85)</td>
</tr>
</tbody>
</table>
The latter difference, which suggests a trend toward more detail in the PDL designs than in the FL designs, is partially, but not entirely, attributable to the difference already noted in usage of conditional expressions.

Detailed study of the designs produced with the two documentation methods leads to a somewhat clearer picture, and suggests that the differences in design quality scores and level of detail are related. Designs expressed in flowchart form tended to contain greater detail in such areas as initialization than those expressed in a PDL, but much less algorithmic detail. For example, a flowchart design might specify "X=0", "Y=0", "Z=1", while a PDL design might merely say "INITIALIZE X, Y, Z". The flowchart design might only say "COMPUTE LOCATION COUNTER ADDRESS", though, while the PDL design describes the actual procedure for this computation. Somehow, these two documentation media seem to lead to a different emphasis on the part of the designer. It is important to note that this observation holds true even though the same subjects used both methods, and is not a result of the group difference.

Designs expressed in flowchart form exhibited a marked tendency toward the use of abbreviated variable names and other space-conserving practices. This tendency is probably related to the need to compress a lot of information into the rather confined space available in flowchart blocks, and was observed to a much lesser degree in PDL designs. In pass 1, flowchart subjects abbreviated an average of 7.7 unique variable names in their designs, while PDL designs contained an average of only 2.9 abbreviated symbols. This difference is significant (F(15) = 2.246, p < .05). Data from pass 2 are ambiguous, since subjects tended to continue usage of the abbreviations developed in pass 1, even though they switched documentation methods.

It is informative to consider also the way in which subjects abbreviated variable names. Table 6 lists the abbreviations used, and the number of subjects, by documentation type, whose designs contained
Table 6. Summary of Abbreviations Used in Pass 1 Designs

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Number of Designs Using Abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PDL</td>
</tr>
<tr>
<td>ERROR_MSG</td>
<td>1</td>
</tr>
<tr>
<td>HEX_INSTR</td>
<td>1</td>
</tr>
<tr>
<td>HEX_VALUE</td>
<td>1</td>
</tr>
<tr>
<td>INPUT.LOC</td>
<td>1</td>
</tr>
<tr>
<td>LOC_CTR</td>
<td>1</td>
</tr>
<tr>
<td>MULT_SYM</td>
<td>1</td>
</tr>
<tr>
<td>NUM_OPCODE</td>
<td>1</td>
</tr>
<tr>
<td>P</td>
<td>1</td>
</tr>
<tr>
<td>SC</td>
<td>1</td>
</tr>
<tr>
<td>SYM</td>
<td>1</td>
</tr>
<tr>
<td>SYM_OPCODE</td>
<td>1</td>
</tr>
<tr>
<td>VAL</td>
<td>1</td>
</tr>
<tr>
<td>EAC</td>
<td>1</td>
</tr>
<tr>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td>ADDR</td>
<td>4</td>
</tr>
<tr>
<td>ERR</td>
<td>1</td>
</tr>
<tr>
<td>HEX_IMAGE</td>
<td>1</td>
</tr>
<tr>
<td>HEX_LOC</td>
<td>1</td>
</tr>
<tr>
<td>HAC</td>
<td>1</td>
</tr>
<tr>
<td>LOC</td>
<td>3</td>
</tr>
<tr>
<td>LC</td>
<td>1</td>
</tr>
<tr>
<td>MSIZ</td>
<td>1</td>
</tr>
<tr>
<td>OP</td>
<td>1</td>
</tr>
<tr>
<td>ADDR1</td>
<td>1</td>
</tr>
<tr>
<td>ADDR2</td>
<td>1</td>
</tr>
<tr>
<td>H</td>
<td>1</td>
</tr>
<tr>
<td>L</td>
<td>1</td>
</tr>
<tr>
<td>LCTR</td>
<td>1</td>
</tr>
<tr>
<td>HEX (...)</td>
<td>3</td>
</tr>
<tr>
<td>HDP</td>
<td>1</td>
</tr>
<tr>
<td>ADR</td>
<td>1</td>
</tr>
<tr>
<td>HADD</td>
<td>1</td>
</tr>
<tr>
<td>ADD</td>
<td>1</td>
</tr>
<tr>
<td>HLOC</td>
<td>1</td>
</tr>
<tr>
<td>LCNT</td>
<td>1</td>
</tr>
<tr>
<td>INCR</td>
<td>1</td>
</tr>
<tr>
<td>ILOC</td>
<td>1</td>
</tr>
<tr>
<td>NL</td>
<td>1</td>
</tr>
<tr>
<td>IL</td>
<td>1</td>
</tr>
<tr>
<td>INSTR</td>
<td>3</td>
</tr>
<tr>
<td>J</td>
<td>1</td>
</tr>
<tr>
<td>K</td>
<td>1</td>
</tr>
<tr>
<td>I1</td>
<td>1</td>
</tr>
<tr>
<td>K1</td>
<td>1</td>
</tr>
<tr>
<td>K2</td>
<td>1</td>
</tr>
<tr>
<td>K3</td>
<td>1</td>
</tr>
<tr>
<td>K4</td>
<td>1</td>
</tr>
<tr>
<td>TABLE_VAL</td>
<td>1</td>
</tr>
<tr>
<td>OPER</td>
<td>1</td>
</tr>
<tr>
<td>I</td>
<td>3</td>
</tr>
<tr>
<td>CNTL</td>
<td>2</td>
</tr>
<tr>
<td>IMAGE_LOC</td>
<td>2</td>
</tr>
<tr>
<td>IMAGE_INSTR</td>
<td>1</td>
</tr>
<tr>
<td>HEXLOC</td>
<td>1</td>
</tr>
<tr>
<td>CC</td>
<td>1</td>
</tr>
<tr>
<td>NUM</td>
<td>1</td>
</tr>
<tr>
<td>INST</td>
<td>1</td>
</tr>
<tr>
<td>LOC_CNTR</td>
<td>1</td>
</tr>
<tr>
<td>NLOC</td>
<td>1</td>
</tr>
<tr>
<td>HLOC</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>28</td>
</tr>
</tbody>
</table>
these abbreviations. There appears to be a tendency for flowchart designs to contain a larger proportion of abbreviated variable names whose significance is not clear from the names themselves (i.e., "non-mnemonic" variable names like HL, II, etc.).

Another observation by the rater was that the selection of appropriate data structures by the designers was very strongly related to their overall design scores. The rater divided the subjects into two groups (best half, worst half) on the basis of data structure usage. The average overall design scores (both passes) for these groups, 155.6 and 104.0, respectively, differ significantly ($t(18) = 4.24, p < .001$). Clearly, this is not an a priori hypothesis, nor are these two subjective assessments (ratings of data structure usage, overall design quality) in any sense independent. Even if they were, it would not be clear to what degree data structure usage is a causative factor in design performance and to what degree both are just similarly affected by subject differences in ability, background, etc. The observation is interesting, though, and suggests an area for further investigation.

**SUBJECTIVE RATINGS OF DOCUMENTATION METHODS**

A statistical analysis of the Flowchart-PDL comparative questionnaire (Appendix J) revealed several significant preferences or opinions, by the subjects, with respect to the two documentation methods. Each response to items 1-27 and 29 was encoded on a scale of 1-8, where a score of 1 represents an extreme preference for flowcharts and 8 represents an extreme preference for PDLs. Since the ratings were obtained after all subjects had been exposed to the use of both documentation methods, the PDL-FL group and the FL-PDL group were not expected to differ significantly in their responses to the questionnaire. This was verified by a series of t-tests, comparing the responses of the two groups on each item. No overall pattern of differences emerged, and the number of
significant group differences was at the chance level. This allowed us to combine the groups for the purpose of analyzing overall preferences. Means and standard deviations of the ratings by all 20 subjects are shown in Table 7.

The questionnaire items were included because they might generate some insight into designer and programmer perceptions of the documentation methods. In most cases, no clear basis existed for an a priori hypothesis concerning the direction, or even the occurrence, of a preference. In eight cases, though, an a priori hypothesis was made, and the direction of expected preference is indicated in parentheses in the corresponding row of Table 7.

Statistical analyses indicated that four of these items showed at least a moderate (p < .10) deviation from 4.50, the value expected if subjects have no preference or opinion. These four items are among the eight on which a priori hypotheses were stated, and are in the expected direction. Subjects felt that PDLs were preferable to flowcharts with respect to ease of implementation in a programming language (t(19) = 2.76, p < .02), ease of documentation (t(19) = 2.68, p < .02), encouragement of "good" design practices (t(19) = 2.45, p < .05), and ease of modification of documentation (t(19) = 1.77, p < .10). On overall preference measures, the subjects exhibited mild (non-significant) preferences for PDLs for use by the designer, but indicated an expectation that flowcharts are preferable from the viewpoint of the programmer. It should be noted that this questionnaire administration preceded the programming task. This is the only item (of eight) on which the direction of preference was opposite that predicted a priori.

Questionnaire items 16-27 asked how PDLs and flowcharts compare with respect to helping the designer notice problems with the design. Question 28 was intended to determine whether the students understood
the design properties referred to in those questions. Seven subjects reported that questions 16-27 were easy to answer, eight students indicated that they were difficult to answer because the subjects saw no difference between the documentation methods, and five indicated difficulty understanding the design errors and problems referred to in the questions.
Table 7. Means and (in Parentheses) Standard Deviations of Questionnaire Ratings after Design Phase

<table>
<thead>
<tr>
<th>Item</th>
<th>Mean (Standard Deviation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Ease of use by designer (PDL)**</td>
<td>5.00 (1.95)</td>
</tr>
<tr>
<td>2. Ease of use by programmer (PDL)</td>
<td>4.15 (1.79)</td>
</tr>
<tr>
<td>3. Helping designer understand problem</td>
<td>4.35 (1.79)</td>
</tr>
<tr>
<td>4. Encouragement of &quot;good&quot; design practices (PDL)</td>
<td>5.40* (1.64)</td>
</tr>
<tr>
<td>5. Ease of documentation (PDL)</td>
<td>5.50* (1.67)</td>
</tr>
<tr>
<td>6. Ease of modification of documentation (PDL)</td>
<td>5.05* (1.39)</td>
</tr>
<tr>
<td>7. Ease of modification of design</td>
<td>4.00 (1.70)</td>
</tr>
<tr>
<td>8. Expression of control flow (FL)</td>
<td>3.80 (2.24)</td>
</tr>
<tr>
<td>9. Expression of data flow</td>
<td>4.15 (1.87)</td>
</tr>
<tr>
<td>10. Comprehensibility of design</td>
<td>4.95 (1.93)</td>
</tr>
<tr>
<td>11. Ease of reading</td>
<td>4.65 (1.95)</td>
</tr>
<tr>
<td>12. Understanding of control flow</td>
<td>4.35 (1.98)</td>
</tr>
<tr>
<td>13. Understanding of data flow</td>
<td>4.40 (1.79)</td>
</tr>
<tr>
<td>14. Ease of implementation in programming language (PDL)</td>
<td>6.00* (1.62)</td>
</tr>
<tr>
<td>15. Detection of design problems</td>
<td>4.30 (1.84)</td>
</tr>
<tr>
<td>16. Detection of redundant operations</td>
<td>4.25 (1.86)</td>
</tr>
<tr>
<td>17. Noticing that module is too big</td>
<td>4.95 (1.50)</td>
</tr>
<tr>
<td>18. Noticing that module is too complex</td>
<td>4.45 (1.67)</td>
</tr>
<tr>
<td>19. Module has too many or too few submodules</td>
<td>4.55 (1.54)</td>
</tr>
<tr>
<td>20. Module is not cohesive</td>
<td>5.03 (1.44)</td>
</tr>
<tr>
<td>21. Definition is not detailed enough</td>
<td>4.35 (1.57)</td>
</tr>
<tr>
<td>22. Definition has too much detail</td>
<td>4.35 (1.53)</td>
</tr>
<tr>
<td>23. Detection of incomplete module</td>
<td>4.30 (1.75)</td>
</tr>
<tr>
<td>24. Modules tightly coupled</td>
<td>4.55 (1.70)</td>
</tr>
<tr>
<td>25. Design too complex</td>
<td>4.05 (1.67)</td>
</tr>
<tr>
<td>26. Data Fragmentation</td>
<td>4.30 (1.22)</td>
</tr>
<tr>
<td>27. Module is trivial</td>
<td>4.80 (1.70)</td>
</tr>
<tr>
<td>28. Overall impression (PDL)</td>
<td>4.55 (1.64)</td>
</tr>
</tbody>
</table>

* Significantly different from chance (4.5). See text.

** Indicates an a priori hypothesis that PDL would be preferred.
IMPLEMENTATION PHASE

In the implementation phase, each group was exposed to only one documentation method. If the two groups of subjects had been found to differ with respect to relevant ability or background variables, as occurred in the design phase, that difference would have been confounded with the effect of documentation type. However, the two implementation-phase experimental groups (FL, PDL) did not differ significantly on any known background or ability factors. Any performance differences in the implementation phase may therefore be attributed to the effects of documentation type.

The subjects from each design phase group were distributed equally into FL and PDL implementation phase groups. Experience during the design phase is therefore counterbalanced in the implementation phase. Factorial analyses of several key implementation-phase variables (e.g., scores on design comprehension test, grades on implementation project, properties of implemented program) reveals no pattern of interaction between design and implementation conditions. Therefore, implementation-phase results will be reported only by implementation condition (FL or PDL). Because of missing data and other indications by subjects that they were unable to produce reliable estimates of their time expenditure during the implementation phase, these data will not be reported.

Comprehension of Simulator Design

Observed effects of documentation type on design comprehension were small and not statistically significant. Table 8 shows the results of the multiple-choice portion of the design comprehension test administered after 45 minutes' study of the design (first test) and again at the end of the entire implementation phase (second test). It is important to note that results reported for the first test are based on all 20 subjects, while those for the second test exclude subjects who did not complete the study.
Table 8. Results of Multiple-Choice Portion of Design Comprehension Test

<table>
<thead>
<tr>
<th>Category</th>
<th>Items</th>
<th>First Test</th>
<th>Second Test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>FL</td>
<td>PDL</td>
</tr>
<tr>
<td>Total Score</td>
<td>All</td>
<td>17.0</td>
<td>17.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(3.86)</td>
<td>(5.72)</td>
</tr>
<tr>
<td>Comprehension of Micro Nova</td>
<td>5, 14, 26</td>
<td>1.80</td>
<td>2.50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(.789)</td>
<td>(.707)</td>
</tr>
<tr>
<td>Distinction between Micro-Nova and Simulator</td>
<td>11, 12, 24</td>
<td>1.90</td>
<td>1.80</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(.738)</td>
<td>(.919)</td>
</tr>
<tr>
<td>Debugging of Simulator</td>
<td>2, 10, 20, 29</td>
<td>1.80</td>
<td>2.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(1.03)</td>
<td>(1.25)</td>
</tr>
<tr>
<td>Comprehension of Impact of Changes</td>
<td>6, 9, 19, 27</td>
<td>2.20</td>
<td>1.50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(.919)</td>
<td>(1.18)</td>
</tr>
<tr>
<td>Comprehension of General Structure</td>
<td>4, 8, 15, 16, 21, 23, 25, 28</td>
<td>4.10</td>
<td>4.60</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(.876)</td>
<td>(1.90)</td>
</tr>
<tr>
<td>Comprehension of Procedural Detail</td>
<td>1, 3, 7, 13, 17, 18, 22, 30</td>
<td>5.20</td>
<td>4.70</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(2.26)</td>
<td>(2.26)</td>
</tr>
</tbody>
</table>
On the total score, and most of the subscores, statistically significant improvement was observed from the first test to the second, as might be expected.

On the first test, students using a PDL scored somewhat better than FL subjects on comprehension of the MicroNova ($t(18) = 2.090, p < .06$); this difference disappeared entirely on the second test. It should be noted that this test category (as well as the next category, Distinction between MicroNova and Simulator) was intended as a measure of background knowledge, rather than of information contained in the design itself. It is possible, however, that the PDL design somehow aided rehearsal of this information, or otherwise reinforced it more than did the flowchart design. The MicroNova information is derivable from the design; it was considered background information primarily because the student was expected to know it already.

Repeated-measures analyses of variance on the total score and each subscore (using data for the 16 subjects who completed the study) found no flowchart-PDL differences which were consistent across both test administrations.

As a priori hypothesis relating to subscores was that flowcharts might aid comprehension of procedural detail, while PDLs might improve comprehension of the general structure and function of the design. The data indicate mild, but statistically insignificant, tendencies in the expected directions.

No significant differences were observed between PDL and FL subjects with respect to the number of modules they were able to correctly identify from the design, or with respect to the number of calling-module/called-module relationships correctly identified. These are probably not sensitive measures of design comprehension.
Quality of Implementations

We originally intended to evaluate the simulator programs by the method of Youngs (1974), in which the final and all earlier runs are analyzed for errors in reverse chronological order. This method is very effective in detecting a variety of error types, by the programmer, including clerical, syntactic, and various types of conceptual error. However, the number of runs involved was much larger than anticipated (1192), and such an approach would have been an overwhelming task, particularly given the size of the programs (average length, 1030 source statements). As a result, only the final, completed programs were evaluated.

The completed simulator programs were evaluated by an expert rater (JRV) on the basis of the a priori grading criteria shown in Figure 4. These ratings were done on the basis of: (1) the finished source programs (as compiled, with symbol table, cross-reference table, etc.), (2) results of a fairly extensive test program, and (3) reference to earlier simulator development runs as needed. The results of this evaluation are shown in Table 9. Statistical analysis indicates that there are no significant differences between flowchart and PDL groups on total score or on any of the subscores. No particular classes of implementation error were noted which seemed to be attributable to design documentation type.

Appendix Q contains the program which was most nearly average in quality. This particular implementation was developed from a flowchart design.

The final programs were also evaluated with respect to several simple measures of programming style or programming practice. No differences were observed between FL and PDL subjects with respect to number of unique variable names, total number of variable name references,
Implementation Evaluation Categories

Category 1 (50 points)
Location counter modification and referencing logic
Incrementation associated with fetching
Conditional modification based on skip indicator
Relative addressing
Jumps and subroutine calls
Traps and interrupts
Subroutine returns
Two word block I/O instructions
Setting and testing active trace location

Category 2 (50 points)
Effective address computation and usage
Indexing and displacement addressing
Indirect addressing including chaining and auto indexing
Usage of effective addresses
Interrupts and traps
I/O
Memory reference instructions

Category 3 (15 points)
Dump and trace control logic

Category 4 (25 points)
Interrupt detection and generation including real time clock management and interrupt enable/disable logic.

Category 5 (15 points)
Instruction class resolution and invocation.

Category 6 (25 points)
Instruction simulation excluding logic covered by previous categories.

Total - 180 points

Figure 4. Grading Criteria for Implemented Simulators
Table 9. Means and (Standard Deviations) of Implementation Quality Ratings

<table>
<thead>
<tr>
<th>Category (see Figure 4)</th>
<th>Group</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FL</td>
<td>34.38</td>
<td>43.25</td>
<td>12.63</td>
<td>21.13</td>
<td>15.00</td>
<td>20.50</td>
<td>146.88</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(14.97)</td>
<td>(7.83)</td>
<td>(3.74)</td>
<td>(4.79)</td>
<td>(0)</td>
<td>(4.50)</td>
<td>(28.42)</td>
</tr>
<tr>
<td></td>
<td>PDL</td>
<td>33.75</td>
<td>43.25</td>
<td>12.75</td>
<td>19.88</td>
<td>13.75</td>
<td>20.63</td>
<td>144.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(14.40)</td>
<td>(6.45)</td>
<td>(4.17)</td>
<td>(5.19)</td>
<td>(1.75)</td>
<td>(4.27)</td>
<td>(28.66)</td>
</tr>
</tbody>
</table>
or number of usages of each variable. The flowchart group used 26% more "GO TO" statements than did the PDL group (means were 36.75 and 29.13, respectively), and this difference remained when adjustments were made from program length. However, the variances were very large, and this mean difference does not approach statistical significance.

The FL and PDL groups also exhibited differences in average program length and in the number of runs made during program development. As in the case of "GO TO" statements, the mean differences are fairly large, but are not statistically significant. The FL and PDL subjects averaged 63 and 86 runs, respectively, but this difference is not significant (F (1, 14) = 2.272, p > .10). The average program length (number of source statements) for FL and PDL subjects, respectively, was 1136 and 924, a non-significant difference (F (1, 14) = 1.893, p > .10).

Subjective Ratings of Documentation Methods

Results of the post-implementation comparative questionnaire are shown in Table 10. Overall, subjects indicated significant preference for PDL over flowchart documentation (t (15) = 2.15, p < .05). Specifically, significant PDL preferences were noted with respect to ease of documentation (t (15) = 4.04, p < .01), ease of modification of documentation (t (14) = 2.40, p < .05), ease of implementation in a programming language (t (15) = 3.95, p < .01), and ability to detect an insufficiently detailed module in the design (t (15) = 2.18, p < .05). Students tended also to prefer PDLs with respect to encouragement of "good" design practices (t (15) = 2.06, p < .10). On questionnaire items involving a priori hypotheses, eight out of eight preferences were in the hypothesized direction, with four significant at the P < .05 level (and one trend at P < .10).

The subjective ratings may have been biased by a lecture delivered at OSU by well-known computer scientist Daniel McCracken shortly before
The questionnaire was administered. In the lecture, McCracken extolled the virtues of PDLs at some length. With respect to overall preference (item 29), subjects attending the lecture produced a mean rating of 6.25, as compared with 4.75 for those not attending the lecture. Variances were large, however (standard deviations were 1.28 and 2.12, respectively), and this mean difference does not achieve statistical significance \( t(13) = 1.71, p > .10 \). The cause and effect are not entirely clear, as the more highly motivated students were probably more prone to attend the lecture. Nonetheless, this biasing factor must be considered in interpreting the results.

The results shown in Table 10 actually represent two groups (PDL, FL) which were not entirely in agreement. As one might expect, subjects' preferences tended to be influenced by the documentation method they actually used during the implementation phase. On most questionnaire items, PDL subjects indicated stronger preference for PDLs than did FL subjects. These group differences were only significant on a few items, however, on item 12, understanding of control flow, mean ratings were 5.75 and 3.63, respectively \( t(14) = 2.26, p < .05 \). On item 15, detection of design problems, means were 5.88 and 4.13 \( t(14) = 3.30, p < .01 \).

Immediately after the initial study of the designs and administration of the first comprehension test, several students using flowcharts commented spontaneously that their impression of the relative readability of the two documentation methods had changed considerably as a result of their attempt to understand the flowchart design. This was reflected later in their responses on item 11 of the questionnaire. Between administrations of the questionnaire, the ratings on this item, by students using flowcharts, changed by an average of 1.38 points in favor of PDLs (by a difference test, \( t(7) = 4.88, p < .01 \). The responses of both PDL and FL subjects also changed an average of .94 points in favor of PDLs on item 21, detection of insufficiently detailed module \( t(15) = 4.04, \)
Table 10. Means and (Standard Deviations) of Questionnaire Ratings after Implementation Phase

<table>
<thead>
<tr>
<th>Item</th>
<th>Mean (Standard Deviation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Ease of use by designer (PDL)**</td>
<td>5.25 (1.77)</td>
</tr>
<tr>
<td>2. Ease of use by programmer (PDL)</td>
<td>5.13 (2.06)</td>
</tr>
<tr>
<td>3. Helping designer understand problem</td>
<td>5.0 (1.79)</td>
</tr>
<tr>
<td>4. Encouragement of &quot;good&quot; design practices (PDL)</td>
<td>5.56* (2.06)</td>
</tr>
<tr>
<td>5. Ease of documentation (PDL)</td>
<td>5.88* (1.36)</td>
</tr>
<tr>
<td>6. Ease of modification of documentation (PDL)</td>
<td>5.4* (1.45)</td>
</tr>
<tr>
<td>7. Ease of modification of design</td>
<td>4.73 (1.87)</td>
</tr>
<tr>
<td>8. Expression of control flow (FL)</td>
<td>3.80 (1.86)</td>
</tr>
<tr>
<td>9. Expression of data flow</td>
<td>4.0 (1.52)</td>
</tr>
<tr>
<td>10. Comprehensibility of design</td>
<td>5.0 (1.96)</td>
</tr>
<tr>
<td>11. Ease of reading</td>
<td>5.0 (1.78)</td>
</tr>
<tr>
<td>12. Understanding of control flow</td>
<td>4.69 (2.12)</td>
</tr>
<tr>
<td>13. Understanding of data flow</td>
<td>4.47 (1.46)</td>
</tr>
<tr>
<td>14. Ease of implementation in programming language (PDL)</td>
<td>6.25* (1.77)</td>
</tr>
<tr>
<td>15. Detection of design problems</td>
<td>5.0 (1.37)</td>
</tr>
<tr>
<td>16. Detection of redundant operations</td>
<td>4.38 (1.45)</td>
</tr>
<tr>
<td>17. Noticing that module is too big</td>
<td>4.75 (1.44)</td>
</tr>
<tr>
<td>18. Noticing that module is too complex</td>
<td>4.38 (1.50)</td>
</tr>
<tr>
<td>19. Module has too many or too few submodules</td>
<td>4.81 (1.47)</td>
</tr>
<tr>
<td>20. Module is not cohesive</td>
<td>4.63 (1.26)</td>
</tr>
<tr>
<td>21. Definition is not detailed enough</td>
<td>5.31* (1.49)</td>
</tr>
<tr>
<td>22. Definition has too much detail</td>
<td>4.81 (1.42)</td>
</tr>
<tr>
<td>23. Detection of incomplete module</td>
<td>4.63 (1.41)</td>
</tr>
<tr>
<td>24. Modules tightly coupled</td>
<td>4.33 (1.40)</td>
</tr>
<tr>
<td>25. Design too complex</td>
<td>4.25 (1.53)</td>
</tr>
<tr>
<td>26. Data Fragmentation</td>
<td>4.20 (1.21)</td>
</tr>
<tr>
<td>27. Module is trivial</td>
<td>4.81 (1.38)</td>
</tr>
<tr>
<td>28. Overall impression (PDL)</td>
<td>5.50* (1.86)</td>
</tr>
</tbody>
</table>

* Significantly different from chance (4.5). See text.

** Indicates an *a priori* hypothesis that PDL would be preferred
p < .01). Student comments seem to indicate that these rating changes reflect problems encountered by the students during the implementation task.
DISCUSSION

The results indicate that designs produced in PDL form had higher overall quality than those produced in flowchart form. PDL designs were more detailed, particularly with respect to algorithmic detail, and involved much less abbreviation of variable names than flowchart designs. When designs were presented to subjects in PDL or flowchart form, however, no pattern of short or long-term differences was detected on measures of design comprehension. No significant differences were detected in the quality or other properties of the programs written as implementations of the designs. Subjective ratings indicated a mild overall preference for PDLs, as well as several specific criteria on which students found PDLs preferable to flowcharts.

The measures of design quality which were used to assess subject-generated designs can be criticized as too subjective. However, it is not clear that any more objective approach is available, given our current level of understanding of the software design process. In any event, the differences in logical quality of the designs were fairly large, favored PDLs fairly consistently, and were generally consistent with other, more objective observations concerning the level of detail and readability of the produced designs.

The observed difference in level of detail between flowchart and PDL designs is particularly interesting. The fact that PDL designs contained 32% more conditional expressions than did flowchart designs suggests that they contain a greater degree of algorithmic or procedural detail. This may, in fact, be a significant factor in their higher assessed quality. The finding that PDL designs contained 44% more modules than did flowchart designs is also suggestive of greater detail, although it is possible that PDLs simply encourage modularization. Notice, however, that a higher level of procedural detail implies the
existence of a larger number of non-conditional expressions, as well as a larger number of conditional expressions. Yet, a 32% difference in conditional expressions was accompanied by only a 10% difference in "other executable operations". Examination of the designs suggests that the flowchart designs contain more detailed information concerning some aspect of the design which is not associated with condition-
testing. A particularly prevalent difference concerns initialization of variables. While PDL designs tend to specify initialization with a single ambiguous statement ("INITIALIZE VARIABLES", or something similar), flowchart designs tend to contain explicit, unambiguous specifications of each initialization operation (e.g., "SET X=0", "SET Y=0", SET Z=1").

Informal program design languages are similar to natural language to a much greater degree than are flowcharts. Studies of query formulation by non-programmers (Gould, et al 1976; Miller and Becker, 1974) have observed the frequent use of procedurally ambiguous statements which require interpretation by the reader. The use of such statements as "INITIALIZE VARIABLES" in PDL designs may be related to those observations. In using natural language, people apparently tend to assume that the recipient of the communication will possess sufficient semantic information to correctly interpret such imprecise statements. The flow-
chart "language", on the other hand, is much more formal and may encourage the use of detailed, unambiguous individual statements.

PDL and flowchart usage differed not only in overall level of detail, but also in the aspects of the designs selected for detailed specification. Many would argue that it is the algorithmic detail that is most important in the specification of these designs, and that details in initialization and similar information can be provided by the program-
er. If true, the data suggests that the designer's use of PDLs encourages more detailed specification of relevant aspects of the design than does his use of flowcharts.
When subjects were presented with logically equivalent designs expressed either in PDL or flowchart form, no differences were observed in either "short-term" (45 minutes) or long-term comprehension of the designs. The lack of a short-term comprehension difference seems inconsistent with the findings of Wright and Reid (1973), whose subjects remembered a procedure better when expressed in a form somewhat similar to a PDL than when expressed in flowchart-like form. It should be noted, however, that comprehension of this simulator design is quite a formidable task when compared with their simple algorithms, which typically involved only three binary decisions. It is questionable whether 45 minutes' exposure to the simulator design represent a "short-term" presentation in the same sense as the much shorter exposure to simple algorithms in the Wright and Reid study. Furthermore, their recall measure (execution of algorithm from memory) is probably a more sensitive performance measure than our design comprehension test. Finally, the specialized background of our programmer-subjects may have led them to develop more efficient methods for processing flowchart information and encoding it in memory.

This experiment is a conservative test of the relative comprehensibility of the two documentation methods. In order to investigate differences in documentation method in isolation from other factors, it was necessary to carefully control the way in which design statements were worded. Wherever possible, complete statements were expressed identically in PDL and flowchart designs. This resulted in flowcharts which were more "wordy" than usual, but which were locally (i.e., at the level of individual statements) quite readable. The primary issue addressed in the experiment was the effect of documentation type on global comprehensibility.

The results suggest that flowcharts and PDLs are equally comprehensible if they contain similar, readable, unabbreviated statements.
If used well, either method may allow effective communication with the programmer. In actual practice, however, the use of flowcharts appears to encourage the use of highly abbreviated symbols and other space-saving techniques. Although the present study has not attempted to determine whether or not such abbreviations adversely affect the comprehensibility of designs, it would be surprising if they did not do so, at least in the short term. Thus, in actual practice flowchart designs may be less comprehensible because of the medium's effect on the behavior of the designer, rather than its effect on the programmer.

Design documentation method was not found to affect the quality or style of implementation by the programmer. It might be suggested that a good programmer should be able to understand a good design expressed in any reasonable form. In fact, such a suggestion was made by several of the students who participated in the study. The issue may not be that simple, however. The fact that the flowchart and PDL designs were logically equivalent does not imply that they were psychologically equivalent. If the programmer's perception of the design is altered by the form in which the design is presented -- for example, by causing him to attend more to control structure, or more to modularization -- he may, in effect, solve a different problem. There is evidence of such phenomena in the literature of other task domains (e.g., Simon & Hayes, 1976). There was a no a priori justification for assuming that the peculiar properties of PDLs and flowcharts would not exert such differential effects. Based on the data obtained in this experiment, however, such differential effects appear to be either insignificant in magnitude or of a type to which the analyses employed are insensitive. The latter possibility is certainly not to be ignored.

Subjective preferences, as expressed in responses to both questionnaires, and especially the post-implementation questionnaires, indicate a mild but somewhat consistent preference for PDLs over flowcharts. Subjects also indicated several criteria on which PDLs are particularly to be
preferred, including encouragement of "good" design practices, ease of documentation and of modification of documentation, and ease of implementation in a programming language.

Overall, the results appear to provide a fairly strong case for the use of program design languages, in preference to flowcharts, for the expression of detailed software designs by the designer. As with all studies, it must be kept in mind that this study was done using a particular design and programming task, a particular class of designers and programmers, and a particular experimental setting. The degree to which similar results might be obtained in other software design and programming situations depends in part upon their similarity to the conditions under which this experiment was conducted. Generalization to meaningful software development settings was, of course, the primary consideration in our decision to study the behavior of experienced programmers involved in software development efforts of significant size.
CONCLUSIONS

In the context in which this study was performed, the use of a Program Design Language (PDL) by a software designer, for the development and description of a detailed program design, produced better quality designs than did the use of flowcharts. In particular, the PDL designs involved more algorithmic or procedural detail than those produced using flowcharts. In addition, flowchart designs exhibited considerably more abbreviation and other space-saving practices than did PDL designs, with a possible adverse effect on their readability.

When equivalent, highly readable designs were presented to subjects in both PDL and flowchart form, no pattern of short-term or long-term differences in comprehension of the design was observed. No significant differences were detected in the quality or other properties of programs written as implementations of the designs. Subjective ratings indicated a mild preference for PDLs.

Overall, the results suggest that software design performance and designer-programmer communication might be significantly improved by the adoption of informal Program Design Languages, rather than flowcharts, as a standard documentation method for detailed computer program designs.
REFERENCES


<table>
<thead>
<tr>
<th>Event</th>
<th>Date</th>
<th>Appendices</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Start of course</td>
<td>18 Jan</td>
<td>B, C</td>
</tr>
<tr>
<td>2. Explanation of experiment</td>
<td>18 Jan</td>
<td></td>
</tr>
<tr>
<td>3. Lecture on PDLs, flowcharts</td>
<td>8 Feb.</td>
<td>D, E</td>
</tr>
<tr>
<td>4. Pass 1 design started</td>
<td>8 Feb.</td>
<td>F, G, I</td>
</tr>
<tr>
<td>5. Pass 1 collected, pass 2 started</td>
<td>within 5 days of 4</td>
<td>H</td>
</tr>
<tr>
<td>7. PDL vs. FL questionnaire</td>
<td>17 Feb.</td>
<td>J</td>
</tr>
<tr>
<td></td>
<td>due 22 Feb.</td>
<td></td>
</tr>
<tr>
<td>8. Simulator design distributed</td>
<td>22 Feb.</td>
<td>K, L, O</td>
</tr>
<tr>
<td>9. Design comprehension test</td>
<td>1 hr.</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>after 8</td>
<td></td>
</tr>
<tr>
<td>10. Design revision distributed</td>
<td>24 Feb.</td>
<td>M</td>
</tr>
<tr>
<td>11. Simulator programs collected</td>
<td>21 Apr.</td>
<td></td>
</tr>
<tr>
<td>12. Second comprehension test</td>
<td>28 Apr.</td>
<td>N</td>
</tr>
<tr>
<td>13. PDL vs. FL questionnaire</td>
<td>28 Apr.</td>
<td>J</td>
</tr>
</tbody>
</table>
APPENDIX B - CONSENT FORMS

INFORMED CONSENT FORM

The purpose of this study is to examine the relative merits of two forms of design specification, flowcharts and program design languages, on the design and implementation of software systems. This study will in no way interfere with the stated objectives of this course. We will observe and record your performance on required course projects; you will not be required to perform significant activities in addition to normal course assignments.

We will observe your performance on two major assignments -- a design task and an implementation task. In addition, you will be given course exams and asked to complete a questionnaire describing your background in computer science. Participants in this study will remain anonymous. The sole purpose of this research is to examine alternative forms of design specification; it is not a test of your intelligence or personality. You are not required to participate in this study, and you may change your mind about participating at any time during the experiment. However, payment or other rewards can only be given to those who complete all experimental requirements. At the conclusion of this study, a complete description of this research will be given to you.

Although course grades will be determined, in part, on the basis of information collected during this experiment, experimental manipulations will not influence grades. Your performance will be compared, for grading purposes, only with that of students under identical experimental conditions. The experimental manipulations are not intended to make your task more difficult or to degrade your performance in any way. There is no reason to believe that the experimental materials that are presented to you will either help or hinder your performance relative to students given other materials. In order to insure an accurate assessment of the experimental manipulations to be performed, we ask that you do not compare materials, problems or solutions with fellow students.

Any questions concerning the procedures of this study can be discussed with Dr. Van Doren. These procedures were designed to be consistent with guidelines established by the Department of Health, Education and Welfare concerning the use of human subjects and with Oklahoma State University policies.

The results of this study are expected to be of considerable importance to the software development community. Previous studies have generally involved undergraduates with little, if any, computer science background and very small projects. The results obtained from such studies are frequently equivocal. The study in which you will participate offers the significant opportunity to observe the performance of well-qualified computer scientists on non-trivial projects. This increases the probability of finding meaningful results that can be used to improve computer science training and practices.

Your cooperation is invaluable and greatly appreciated. Because the study is expected to be interesting and beneficial to you, as well as to the software development community at large, we hope you will want to participate. However, as an additional reward for your participation, you will be paid $25.00 upon successful completion of all experimental requirements.
I have read and understand the above statement and agree to participate in this study.

Name (please print) __________________________

Signature __________________________
In analyzing the results of this study, it would be useful to have a record of your previous and current courses and the grades received. All information about grades will be treated as confidential and identified only by a code number. A master list assigning code numbers to names will be maintained only by Dr. Van Doren and will be destroyed as soon as possible after completion of this study.

Subject to the conditions stated above, I agree to the release of the information contained in my academic record.

Name (please print)

Signature
APPENDIX C - BACKGROUND QUESTIONNAIRE

Name: ______________________________

INSTRUCTIONS

This questionnaire is designed solely for experimental purposes; your responses will in no way affect your course grade. Since we will be observing your performance on course projects, it would be very useful to us to have a brief description of your background in computer science. We ask that you complete these questions accurately and honestly.
1. List previous computer science courses.

   Undergraduate

   

   

   

   

   

   

   

   Graduate

2. Are you enrolled in the computer science program? Yes ___ No ___
   If no, which program are you enrolled in? __________________________
   If yes, which graduate option? ________________________________
   How many hours have you completed in this option? ________________
   How many graduate hours have you completed in computer science?

3. List previous degrees received

   ________________________________

4. When do you expect to graduate from this program and what degree will you receive?

   ________________________________

5. Are you a native speaker of English? Yes ___ No ___
   If no, how long have you lived in English speaking countries?
   Years _______ Months
   Were your previous degrees obtained at universities where English was used? Yes ___ No ___ Some ___
   If no or some, briefly describe your educational background in terms of universities, language, years enrolled and degrees received.

   ________________________________
6. Briefly describe your experiences, outside of normal course work with the
following (list types of experience, length, where received, etc.):

- PL/1
- Flowcharts
- Program Design Languages
- Other Design Specification Methods (specify)
- Assemblers, Translators, and Compilers
- Machine Simulators
- Mini/Micro Computers
- Date General NOVA Computers

Other Related Experiences

7. Age_______
8. Sex_______
APPENDIX D - SOFTWARE DESIGN METHODS HANDOUT

Notes on Software Design Methods (Flowcharts and PDL's)

The material presented is similar but not identical to material contained in:

Structured Programming Series, Vol. 8:
Program Design Study
IBM Federal Systems Center
Gathersburg, Maryland

Produced under U.S. Air Force contract
and available from
National Technical Information Service
Ref. # RADC-TR-74-300

It is assumed that you are familiar with flowcharting and the conventions in use at Oklahoma State University. It is not assumed that you are familiar with Program Definition Languages (PDL's). PDL's typically have control structures similar to those available in block structured languages such as PL/1 and ALGOL. However, a PDL is used for software design, not as a programming language. Of course, there may be direct correspondence with some features in your programming language just as there may be with flowcharts. PDL's may be informal or formal. The latter have a very formal and strict definition. The former have certain well defined control structures but much of the processing can be specified in an informal way just as flowchart processing blocks may have varying degrees of precision and formality depending on the level of logic or design.

For purposes of the software design study being conducted in this course an informal PDL will be used. As part of your computer science knowledge you should become aware of PDL's as alternative methods for software design (and documentation as well).

The remainder of the material comprises a parallel presentation of the control structures for the informal PDL and corresponding flowcharting treatments.

D-1
Modules or procedures:

**PDL**

- Module name: **PROCEDURE** optional parameter list;

  - Sequence of PDL and/or English language statements
  - RETURN TO CALLER;
  - END module name;

**Module invocation:**

- CALL module name optional parameter list;

**Elementary decision logic:**

- IF condition THEN
  - English language or PDL statement;
  - ELSE
  - English language or PDL statement;

- IF condition Then
  - English language or PDL statement;
Looping:

**PDL**

**DO WHILE** condition;
Sequence of English language
and/or PDL statements;
**END**;

**DO UNTIL** condition;
Sequence of English language
and/or PDL statements;
**END**;

**DO** index = initial value **TO** final value **BY** incr.
Sequence of English language
and/or PDL statements;
**END**;

Multiway decision:

**PDL**

**DO** CASE arithmetic value;
**CASE0**:
English language statement
or PDL control statement;
**CASE1**:
English language statement
or PDL control statement;

**CASEn**:
English language statement
or PDL control statement;
**END CASE**;

**Note1:** If the arithmetic value is "out of range" nothing is specified.
Some conventions establish a default control path for this. Please
observe that a zero arithmetic value is considered in range for this PDL.

**Note2:** Multiway decisions such as illustrated may easily be simulated
in PL/1 with the use of a LABEL vector, the PL/1 equivalent of a "computed
GO TO" and a GO TO after each CASE block.
Sequence of design specifications:

PDL

DO;
  Sequence of English language
  and/or PDL statements;
END;

Flowchart

Additional comments:
Note that semicolons ";" are used to terminate all statements in a PDL design. This may be particularly important in making clear a sequence of processing given in the form of English language statements. Further note that indentation schemes for nested PDL specifications are considered part of the PDL method.
The illustrations are given in terms of a simulator design for the "small computer" discussed in class via APL functional descriptions. There are several items about this design to be stressed:

1) For design illustration reasons it is assumed that an op code of 0000 (binary bits) represents a "HALT" instruction and that op codes of 0011, 0100 and 0111 are invalid. The APL functional description does not include these items. It is further assumed that op codes of 1100 and 1111 represent "TRACE ON" and "TRACE OFF" simulator instructions. These two instructions are not part of the "small computer" instruction set but rather are simulator design provisions for tracing the flow of simulated execution of "small computer" programs.

2) The simulator design is a software design which contains features that are not part of the functional characteristics of the simulated machine. Obviously there are some similarities between the simulator design and the APL functional characteristics.

3) Neither of the software designs includes the mathematical precision of the the APL functional specifications. Precise details for simulating certain functional characteristics should be deduced from the APL specifications. Programming details do not require direct correspondence to APL features. You need only assure logical equivalence. The software design shows where the detailed programming for these items belongs in the general scheme of things. The notion of logical equivalence of the implemented program applies as well to software design features as it does to APL description features.

4) Memory access (MAC) is not shown explicitly as a module of the design. Some form of simulated memory access is certainly implied by provisions for instruction fetching:, retrieving an operand for adding or subtracting and for loading or storing the accumulator. Also instruction tracing is not shown as an explicit module. It may be highly desirable to implement these features, and others, as explicit subprograms. They are not shown explicitly in the sample design given because it was not deemed necessary to show any logical design for them. (Perhaps such designs would be included in the next step for "top down" design advocates.)
SMALL_COMPUTER_SIMULATOR: PROCEDURE;
    INITIALIZE_SIMULATOR;
    DO WHILE RUN_CODE ^= CONTINUE_EXECUTION_CODE;
        FETCH_INSTRUCTION;
        INCREMENTLOCATION COUNTER PART OF PROGRAM STATUS WORD;
        IF INSTRUCTION IS "HALT" INSTRUCTION THEN RUN_CODE ^= HALT TERMINATION_CODE;
        ELSE
            DO;
                DETERMINE INSTRUCTION CLASS, INSTRUCTION SUBCLASS AND
                OPERAND ADDRESS FROM INSTRUCTION;
                DO CASE(INSTRUCTION CLASS);
                    CASE0:
                        CALL PROCESS_LOAD_STORE;
                    CASE1:
                        CALL PROCESS_ADD_SUBTRACT;
                    CASE2:
                        CALL PROCESS_BRANCH;
                    CASE3:
                        CALL PROCESS_READ_WRITE_TRACE;
                END CASE;
            END;
        END;
        IF TRACE IS REQUESTED THEN PERFORM TRACE;
        INCREMENT INSTRUCTION EXECUTION COUNT;
        IF INSTRUCTION EXECUTION COUNT EXCEEDS EXECUTION LIMIT THEN
            RUN_CODE ^= EXECUTION LIMIT TERMINATION_CODE;
    END;
    PRINT SIMULATOR STATUS INFORMATION UPON TERMINATION;
    END SMALL_COMPUTER_SIMULATOR;

PROCESS_LOAD_STORE: PROCEDURE;
    DO CASE(INSTRUCTION SUBCLASS);
    CASE0:
        RUN_CODE ^= SIMULATOR FAILURE_CODE;
    CASE1:
        STORE ACCUMULATOR IN MAIN MEMORY(OPERAND ADDRESS);
    CASE2:
        LOAD ACCUMULATOR FROM MAIN MEMORY(OPERAND ADDRESS);
    CASE3:
        RUN_CODE ^= INVALID INSTRUCTION_CODE;
    END CASE;
    RETURN TO CALLER;
END PROCESS_LOAD_STORE;
PROCESS_ADD_SUBTRACT: PROCEDURE;
    RETRIEVE OPERAND FROM MAIN MEMORY(OPERAND ADDRESS);
    DO CASE(INSTRUCTION SUBCLASS);
        CASE0:
            RUN CODE ← INVALID INSTRUCTION CODE;
        CASE1:
            SUBTRACT OPERAND FROM ACCUMULATOR WITH RESULT LEFT IN ACCUMULATOR AND CARRY BIT ASSIGNED TO PROGRAM STATUS WORD;
        CASE2:
            ADD OPERAND TO ACCUMULATOR WITH RESULT LEFT IN ACCUMULATOR AND CARRY BIT ASSIGNED TO PROGRAM STATUS WORD;
        CASE3:
            RUN CODE ← INVALID INSTRUCTION CODE;
    END CASE;
    RETURN TO CALLER;
END PROCESS_ADD_SUBTRACT;

PROCESS_BRANCH: PROCEDURE;
    INTERPRET OPERAND ADDRESS AS BRANCH TARGET;
    INTERPRET INSTRUCTION SUBCLASS AS INSTRUCTION CONDITION CODE;
    IF INSTRUCTION CONDITION CODE MATCHES PROGRAM STATUS WORD CONDITION CODE THEN ASSIGN BRANCH TARGET TO LOCATION COUNTER PART OF PROGRAM STATUS WORD;
    RETURN TO CALLER;
END PROCESS_BRANCH;

PROCESS_READ_WRITE_TRACE;
    DETERMINE MAIN MEMORY BLOCK ADDRESS AND EXTERNAL MEMORY BLOCK ADDRESS FROM INSTRUCTION;
    DO CASE(INSTRUCTION SUBCLASS);
        CASE0:
            TURN TRACE REQUEST ON;
        CASE1:
            TRANSMIT 64 CONTIGUOUS WORDS FROM ADDRESSED MAIN MEMORY BLOCK TO ADDRESSED EXTERNAL MEMORY BLOCK;
        CASE2:
            TRANSMIT ADDRESSED EXTERNAL MEMORY BLOCK TO 64 CONTIGUOUS WORDS OF ADDRESSED MAIN MEMORY BLOCK;
        CASE3:
            TURN OFF TRACE REQUEST;
    END CASE;
    RETURN TO CALLER;
END PROCESS_READ_WRITE_TRACE;
PROCESS_BRANCH:

1. ENTER
2. INTERPRET O/F-RAND ADDRESS AS BRANCH TARGET
3. INTERPRET INSTRUCTION SUBCLASS AS INSTRUCTION CONDITION CODE
4. ASSIGN BRANCH TARGET TO LOCATION COUNTER PART OF PROGRAM STATUS WORD
5. INSTRUCTION CONDITION CODE MATCHES PROGRAM STATUS WORD CONDITION CODE?
   - Yes
   - No

   RETURN

PROCESS_READ_WRITE_TRACE:

1. ENTER
2. DETERMINE MAIN MEMORY BLOCK ADDRESS AND EXTERNAL MEMORY BLOCK ADDRESS FROM INSTRUCTION
3. INSTRUCTION SUBCLASS?
   - 0
   - 1
   - 2
   - 3

   TURN TRACE REQUEST ON
   - TRANSMIT 64 CONTIGUOUS WORDS FROM ADDRESSED MAIN MEMORY TO ADDRESSED EXTERNAL MEMORY BLOCK
   - TRANSMIT ADDRESSED EXTERNAL MEMORY BLOCK TO 64 CONTIGUOUS WORDS OF ADDRESSED MAIN MEMORY BLOCK
   - TURN OFF TRACE REQUEST

RETURN
APPENDIX F - BASIC DESCRIPTION OF DESIGN PROBLEM

Assembler Language Translator Design Problem

An assembler language for the hypothetical "small computer" (Small Computer Assembly Language - SCAL) is outlined below. You will be given a software design assignment requirement for a two pass translator for SCAL. One pass of the translator is to be designed with one of the two software design methods presented in class (flowcharts and PDL's). The other pass is to be designed using the other method.

Half the class will be required to use flowcharts for the first pass and half the class will be required to use PDL. The designs from those students consenting to participate in the software design study being conducted this semester will be used (anonymously) for part of this study.

Written requirements for each pass are given on a separate handout. The pass one assignment will be distributed in class Feb. 10 and will be due Feb. 15. (Turn it in to Dr. Fisher's secretary.) The pass two assignment is to be picked up in Dr. Fisher's office when you turn in the pass one part. The pass two design will be due Feb. 17.

SCAL features:

1) Symbolic machine instructions with instruction formats

Optional label | HALT | Optional symbolic or absolute address
---------------|------|-------------------------------
" " | LOAD | Symbolic or absolute address
" " | STORE | " " " "
" " | ADD | " " " "
" " | SUB | " " " "
" " | B | Numeric condition code(0-3), symbolic or absolute address
" " | READ | Symbolic or absolute address
" " | WRITE | " 

2) Constant definitions with formats

Optional label | DC | Integer constant
---------------|----|-------------------
" " | DIO | Symbolic or absolute memory block address / external store address

Note: DIO is intended for defining the addresses for READ or WRITE operations. Ordinarily a DIO would be referenced only by a READ or WRITE. Ordinarily a symbolic reference in a READ or WRITE would be a reference to a DIO. Neither of these are required, however.
3) Assembler instructions with formats

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROG</td>
<td>Absolute address (starting location of program)</td>
</tr>
<tr>
<td>Required label</td>
<td>EQUATE</td>
</tr>
<tr>
<td>LOC</td>
<td>Absolute address (reassigns assembler location counter for location of following instructions)</td>
</tr>
<tr>
<td>Optional label</td>
<td>DS</td>
</tr>
<tr>
<td>END</td>
<td>Symbolic or absolute address (address of first instruction to be executed)</td>
</tr>
</tbody>
</table>

The symbolic addresses are very simple in form and do not allow address expressions such as *+3 or SUM+2. Neither are literal operands allowed. All absolute addresses or numeric constants are assumed to be in decimal form.

Comments and observations about the assignment:

You are encouraged to design your logic at a level similar to that displayed in the software design illustrations. For example, don't be concerned with the detailed logic for "recognizing" a valid symbol or constant (lexical analysis or scanning). You should be able to complete your design in \( \sum 8\frac{1}{2} \times 11 \) pages or less.

Assume that symbol table design and processing algorithms already exist; that is to say, do not attempt to design them. Further assume that the symbol table structure is very simple in that no "type" codes are required - just symbols and absolute addresses.

Neither the PDL nor the flowchart design method provides for data structure definition. For the most part data structure requirements for the assembler are elementary and may be implied by appropriate wordage and symbolism. If certain tables or data structures used in your design are sufficiently complex that separate descriptions of them are needed, then provide them.
<table>
<thead>
<tr>
<th>HEX</th>
<th>IMAGE</th>
<th>ERR</th>
<th>INPUT IMAGE</th>
<th>HEX</th>
<th>IMAGE</th>
<th>ERR</th>
<th>INPUT IMAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCI</td>
<td>INSTR</td>
<td>CTNL</td>
<td>LOC</td>
<td>OP</td>
<td>ADDR</td>
<td>LOC</td>
<td>DRES</td>
</tr>
<tr>
<td>64</td>
<td>E 000</td>
<td>LB37</td>
<td>READ</td>
<td>IAD1</td>
<td>T0</td>
<td>6 000</td>
<td>ADD</td>
</tr>
<tr>
<td>65</td>
<td>E 000</td>
<td>READ</td>
<td>IAD2</td>
<td>T0</td>
<td>8 000</td>
<td>BRO</td>
<td>LB2</td>
</tr>
<tr>
<td>66</td>
<td>2 000</td>
<td>LOAD</td>
<td>TSTI</td>
<td>80</td>
<td>0 000</td>
<td>LB3</td>
<td>WRITE</td>
</tr>
<tr>
<td>67</td>
<td>1 000</td>
<td>STORE</td>
<td>TEST</td>
<td></td>
<td></td>
<td>HALT</td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>2 000</td>
<td>LB1</td>
<td>LOAD</td>
<td>LB10</td>
<td>C9</td>
<td>0 000</td>
<td>LOC</td>
</tr>
<tr>
<td>69</td>
<td>1 000</td>
<td>STORE</td>
<td>LB2</td>
<td>C9</td>
<td>0001</td>
<td>ONE</td>
<td>DC</td>
</tr>
<tr>
<td>6A</td>
<td>2 000</td>
<td>LOAD</td>
<td>LB11</td>
<td>CA</td>
<td>0 000</td>
<td>IAD1</td>
<td>DIF</td>
</tr>
<tr>
<td>6B</td>
<td>1 000</td>
<td>STORE</td>
<td>LB4</td>
<td>CB</td>
<td>0 025</td>
<td>IAD2</td>
<td>DIF</td>
</tr>
<tr>
<td>6C</td>
<td>2 000</td>
<td>LOAD</td>
<td>ZERO</td>
<td>CD</td>
<td>2 000</td>
<td>IAD1</td>
<td>DIF</td>
</tr>
<tr>
<td>6D</td>
<td>1 000</td>
<td>STORE</td>
<td>C</td>
<td>CE</td>
<td>5 000</td>
<td>LB10</td>
<td>LAD</td>
</tr>
<tr>
<td>6E</td>
<td>2 000</td>
<td>LB2</td>
<td>LOAD</td>
<td>A</td>
<td>CF</td>
<td>0FCC</td>
<td>TSTI</td>
</tr>
<tr>
<td>6F</td>
<td>5 000</td>
<td>LB4</td>
<td>SUB</td>
<td>B</td>
<td></td>
<td></td>
<td>MAX</td>
</tr>
<tr>
<td>70</td>
<td>6 000</td>
<td>ADD</td>
<td>C</td>
<td>DO</td>
<td>OFF</td>
<td>MAX</td>
<td>DC</td>
</tr>
<tr>
<td>71</td>
<td>1 000</td>
<td>STORE</td>
<td>C</td>
<td>A</td>
<td>DS</td>
<td>5 64</td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>2 000</td>
<td>LOAD</td>
<td>LB2</td>
<td>280</td>
<td>0 000</td>
<td>A</td>
<td>.EQUATE</td>
</tr>
<tr>
<td>73</td>
<td>6 000</td>
<td>ADD</td>
<td>ONE</td>
<td>2CO</td>
<td>0000</td>
<td>TEST</td>
<td>DC</td>
</tr>
<tr>
<td>74</td>
<td>1 000</td>
<td>STORE</td>
<td>LB2</td>
<td>E1</td>
<td>.EQUATE</td>
<td>89</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>2 000</td>
<td>LOAD</td>
<td>LB4</td>
<td>M2</td>
<td>.EQUATE</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>6 000</td>
<td>ADD</td>
<td>ONE</td>
<td>LOC</td>
<td>576</td>
<td></td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>1 000</td>
<td>STORE</td>
<td>LB4</td>
<td>B</td>
<td>DS</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>2 000</td>
<td>LOAD</td>
<td>TEST</td>
<td>CI</td>
<td>.EQUATE</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>6 000</td>
<td>ADD</td>
<td>ONE</td>
<td>C</td>
<td>DC</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>7A</td>
<td>1 000</td>
<td>STORE</td>
<td>TEST</td>
<td>END</td>
<td>LB37</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7B</td>
<td>8 000</td>
<td>BRO</td>
<td>LB3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7C</td>
<td>2 000</td>
<td>LOAD</td>
<td>MAX</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

F-3
<table>
<thead>
<tr>
<th>HEX IMAGE</th>
<th>ERR LOC INSTR &amp; CNTL</th>
<th>INPUT IMAGE LOC OP ADDR</th>
<th>HEX IMAGE</th>
<th>ERR LOC INSTR &amp; CNTL</th>
<th>INPUT IMAGE LOC OP ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 E 0CA</td>
<td>PROG 100</td>
<td>TB 6 004</td>
<td>ADD 100</td>
<td>ONE</td>
<td>TB 6 004</td>
</tr>
<tr>
<td>65 E 0CB</td>
<td>READ IAD1</td>
<td>TF 8 068</td>
<td>BRO</td>
<td>LB2</td>
<td>TF 8 068</td>
</tr>
<tr>
<td>66 2 0CF</td>
<td>READ IAD2</td>
<td>TF 1 002</td>
<td>WRITE</td>
<td>OR1</td>
<td>TF 1 002</td>
</tr>
<tr>
<td>67 1 2CO</td>
<td>LOAD TST2</td>
<td>80 0 000</td>
<td>HALT</td>
<td>LOC</td>
<td>80 0 000</td>
</tr>
<tr>
<td>68 2 0CD</td>
<td>STORE TST</td>
<td></td>
<td>LOC</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>69 1 06E</td>
<td>LOAD LB10</td>
<td>CB 0 000</td>
<td>ZER0</td>
<td>DC</td>
<td>EB 0 01</td>
</tr>
<tr>
<td>6A 2 0CE</td>
<td>STORE LB2</td>
<td>CB 0 001</td>
<td>ONE</td>
<td>DC</td>
<td>EB 0 01</td>
</tr>
<tr>
<td>6B 1 06F</td>
<td>LOAD LB11</td>
<td>CA 2 859</td>
<td>IAD1</td>
<td>DIO</td>
<td>EB 0 01</td>
</tr>
<tr>
<td>6C 2 06B</td>
<td>STORE LB4</td>
<td>CB 7 463</td>
<td>IAD2</td>
<td>DIO</td>
<td>EB 0 01</td>
</tr>
<tr>
<td>6D 1 100</td>
<td>LOAD ZERO</td>
<td>CC 1 00A</td>
<td>OAD1</td>
<td>DIO</td>
<td>EB 0 01</td>
</tr>
<tr>
<td>6E 2 280</td>
<td>STORE C</td>
<td>CD 2 220</td>
<td>LB10</td>
<td>LOAD</td>
<td>EB 0 01</td>
</tr>
<tr>
<td>6F 5 240</td>
<td>LOAD A</td>
<td>CE 5 240</td>
<td>CB</td>
<td>LOAD</td>
<td>EB 0 01</td>
</tr>
<tr>
<td>70 6 100</td>
<td>SUB B</td>
<td>CF 0 ECO</td>
<td>TST2</td>
<td>DC</td>
<td>EB 0 01</td>
</tr>
<tr>
<td>71 1 100</td>
<td>ADD C</td>
<td>DO 0 OFF</td>
<td>MAX</td>
<td>DC</td>
<td>EB 0 01</td>
</tr>
<tr>
<td>72 2 06F</td>
<td>STORE C</td>
<td></td>
<td>A</td>
<td>EQUATE</td>
<td>EB 0 01</td>
</tr>
<tr>
<td>73 6 0C9</td>
<td>LOAD LB2</td>
<td>280 0 000</td>
<td>A</td>
<td>D5</td>
<td>EB 0 01</td>
</tr>
<tr>
<td>74 1 06E</td>
<td>ADD ONE</td>
<td>280 0 000</td>
<td>TEST</td>
<td>DC</td>
<td>EB 0 01</td>
</tr>
<tr>
<td>75 2 0CF</td>
<td>STORE LB2</td>
<td></td>
<td>E1</td>
<td>EQUATE</td>
<td>EB 0 01</td>
</tr>
<tr>
<td>76 6 069</td>
<td>LOAD LB4</td>
<td></td>
<td>M2</td>
<td>EQUATE</td>
<td>EB 0 01</td>
</tr>
<tr>
<td>77 1 06F</td>
<td>ADD ONE</td>
<td></td>
<td>LOC</td>
<td>576</td>
<td>EB 0 01</td>
</tr>
<tr>
<td>78 2 2CO</td>
<td>STORE LB4</td>
<td>240 0 000</td>
<td>B</td>
<td>D5</td>
<td>EB 0 01</td>
</tr>
<tr>
<td>79 6 001</td>
<td>LOAD TEST</td>
<td></td>
<td>C</td>
<td>D5</td>
<td>EB 0 01</td>
</tr>
<tr>
<td>7A 1 2CO</td>
<td>ADD ONE</td>
<td></td>
<td>D5</td>
<td>END</td>
<td>EB 0 01</td>
</tr>
<tr>
<td>7B 8 0FF</td>
<td>STORE TEST</td>
<td></td>
<td>END</td>
<td>LB37</td>
<td>EB 0 01</td>
</tr>
<tr>
<td>7C 2 06C</td>
<td>LOAD MAX</td>
<td>F-4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNSORTED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>064</td>
<td>LB57</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>068</td>
<td>LB1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>06E</td>
<td>LB2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>06F</td>
<td>LB4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07F</td>
<td>LB3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0C9</td>
<td>ZERO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0C9</td>
<td>ONE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0CA</td>
<td>IAD1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0CB</td>
<td>IAD2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0CC</td>
<td>IAD1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0CD</td>
<td>LB10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0CE</td>
<td>LB11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0CF</td>
<td>TSTI</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0DO</td>
<td>MAX</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>280</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2CD</td>
<td>TEST</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>059</td>
<td>E1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01D</td>
<td>M2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>240</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>004</td>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>230</td>
</tr>
<tr>
<td>240</td>
</tr>
<tr>
<td>100</td>
</tr>
<tr>
<td>004</td>
</tr>
</tbody>
</table>

F-5
APPENDIX G - DESCRIPTION OF PASS 1 DESIGN PROBLEM

Assembler language Translator Design Problem, Pass 1

Name: __________________________ Design Method: Flowchart / PDL

Pass one has the following responsibilities:

To read, process and copy the SCAL program to a secondary storage
file in the form suggested in the example handout.

To fill in the symbol table for defined symbols.

To detect invalid operation codes. (Substitute a "HALT 0' image
for these.)

To detect multiple definitions of a symbolic address.

To detect "out of range address" situations.

To detect other errors and take error action consistent with your
overall software design.

Note 1: The copy of the SCAL program will serve as input to the second
pass.

Note 2: Assume that all necessary information is passed in a global
sense to pass two; that is, it isn’t necessary to identify
explicit parameters.

Note 3: Make sure you fill out and turn in the time questionnaire if
you are participating in the software design experiment. This
information is needed for the study.

G-1
APPENDIX H - DESCRIPTION OF PASS 2 DESIGN PROBLEM

Assembler Language Translator Design Problem, Pass 2

Name: __________________________ Design Method: Flowchart / PDL

Pass two has the following responsibilities:

To generate absolute object code in the format indicated below on a secondary file.

To create a printed form of the original program with machine code information and error codes as suggested by the example handout.

To detect undefined symbolic addresses. (Substitute a zero for such an address in the generated code.)

To detect "out of range" absolute addresses and constants.

To detect errors that shouldn't occur if the first pass is working correctly.

To detect other errors and take error action consistent with your overall software design.

Format of generated code - 40 word records

1st word - absolute address where machine language text is to be loaded
2nd word - number of words of machine language text following Words 3 to 40 - machine language text

"trailer" record
1st word - "0"
2nd word - "0"
3rd word - Absolute address of starting instruction

Note: Make sure you fill out and turn in the time questionnaire. This information is needed for the software design study.
APPENDIX I - TIME EXPENDITURE QUESTIONNAIRE, DESIGN PHASE

TIME QUESTIONNAIRE
ASSEMBLER DESIGN PROBLEM, PASS 1

Name: __________________________

Date: ________________ Code: _______________
(Leave blank)

Please record the time you have spent on your class project in the last week, excluding time spent in class. Record time in hours and/or minutes by category.

Time spent in:

Preparing initial design .............................................. : 
Revising design ........................................................ : 
Hand-checking design ................................................ : 
Preparing documentation .............................................. : 
Just thinking about project ......................................... : 
Other (indicate nature of activity) ____________________________ : 

This information is solely for experimental purposes; your responses will in no way affect your course grade. Please be as accurate as possible.
APPENDIX J - PDL-FLOWCHART QUESTIONNAIRE

Name: ____________________________

INSTRUCTIONS

This questionnaire is designed solely for experimental purposes; your responses will in no way affect your course grade. The goal of this questionnaire is to provide us with an objective, unbiased comparison of flowcharts and program design languages (PDLs) as methods for the development and documentation of software designs. Please complete these questions honestly and thoughtfully, so that your answers reflect your opinions about these methods based on the experience you have had with them.

Most of the questions which follow ask you to compare the two methods by checking a box on a scale. In every case, check one and only one box, and place your checks within the box, rather than on boundaries between boxes.
1. From the viewpoint of the software designer who must use flowcharts or a program design language to document his design, how do the two compare?

<table>
<thead>
<tr>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>PDL</th>
<th>PDL</th>
<th>PDL</th>
<th>PDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolutely better</td>
<td>moderately better</td>
<td>slightly better</td>
<td>better</td>
<td>better</td>
<td>better</td>
<td>better</td>
<td>absolutely better</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. From the viewpoint of the programmer who must understand and implement a design expressed in either a program design language or flowcharts, how do the two compare?

<table>
<thead>
<tr>
<th>PDL</th>
<th>PDL</th>
<th>PDL</th>
<th>PDL</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolutely better</td>
<td>much better</td>
<td>moderately better</td>
<td>slightly better</td>
<td>better</td>
<td>better</td>
<td>better</td>
<td>absolutely better</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3. How do the methods compare with respect to helping the software designer understand his problem?

<table>
<thead>
<tr>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>PDL</th>
<th>PDL</th>
<th>PDL</th>
<th>PDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolutely better</td>
<td>moderately better</td>
<td>slightly better</td>
<td>better</td>
<td>better</td>
<td>better</td>
<td>better</td>
<td>absolutely better</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4. With respect to encouraging the use of "good" design practices?

<table>
<thead>
<tr>
<th>PDL</th>
<th>PDL</th>
<th>PDL</th>
<th>PDL</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolutely better</td>
<td>much better</td>
<td>moderately better</td>
<td>slightly better</td>
<td>better</td>
<td>better</td>
<td>better</td>
<td>absolutely better</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5. Ease of preparation of design documentation?

<table>
<thead>
<tr>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>PDL</th>
<th>PDL</th>
<th>PDL</th>
<th>PDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolutely better</td>
<td>moderately better</td>
<td>slightly better</td>
<td>better</td>
<td>better</td>
<td>better</td>
<td>better</td>
<td>absolutely better</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6. Ease of modification of design documentation?

<table>
<thead>
<tr>
<th>POL</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolutely better</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>much better</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>moderately better</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slightly better</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slightly better</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>moderately better</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>much better</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>absolutely better</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7. Ease of modification of design?

<table>
<thead>
<tr>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolutely better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>much better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>moderately better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slightly better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POL</td>
<td>Flowchart</td>
<td>Flowchart</td>
<td>Flowchart</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>absolutely better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>much better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>moderately better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slightly better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POL</td>
<td>Flowchart</td>
<td>Flowchart</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>absolutely better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>much better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>moderately better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slightly better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POL</td>
<td>Flowchart</td>
<td>Flowchart</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>absolutely better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>much better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>moderately better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slightly better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POL</td>
<td>Flowchart</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>absolutely better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>much better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>moderately better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slightly better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

8. Ease of expression of control flow by designer?

<table>
<thead>
<tr>
<th>POL</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolutely better</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>much better</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>moderately better</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slightly better</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slightly better</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>moderately better</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>much better</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>absolutely better</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

9. Ease of expression of data flow?

<table>
<thead>
<tr>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolutely better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>much better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>moderately better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slightly better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slightly better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>moderately better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>much better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>absolutely better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

10. From the viewpoint of the programmer who must understand and implement the design, how do the methods compare on comprehensibility?

<table>
<thead>
<tr>
<th>POL</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolutely better</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>much better</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>moderately better</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slightly better</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slightly better</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>moderately better</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>much better</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>absolutely better</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

11. How do they compare on ease of reading (i.e., effort involved, speed, etc.)?

<table>
<thead>
<tr>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolutely better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>much better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>moderately better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slightly better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slightly better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>moderately better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>much better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>absolutely better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
12. Ease of understanding control flow?

<table>
<thead>
<tr>
<th>PDL</th>
<th>PL</th>
<th>PDL</th>
<th>PL</th>
<th>PDL</th>
<th>PL</th>
<th>PDL</th>
<th>PL</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolutely better</td>
<td>much better</td>
<td>moderately better</td>
<td>slightly better</td>
<td>absolutely better</td>
<td>much better</td>
<td>moderately better</td>
<td>slightly better</td>
</tr>
</tbody>
</table>

13. Ease of understanding data flow?

<table>
<thead>
<tr>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>PDL</th>
<th>PDL</th>
<th>PDL</th>
<th>PDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolutely better</td>
<td>much better</td>
<td>moderately better</td>
<td>slightly better</td>
<td>absolutely better</td>
<td>much better</td>
<td>moderately better</td>
<td>slightly better</td>
</tr>
</tbody>
</table>

14. Ease of implementing in a programming language?

<table>
<thead>
<tr>
<th>PDL</th>
<th>PL</th>
<th>PDL</th>
<th>PL</th>
<th>PDL</th>
<th>PL</th>
<th>PDL</th>
<th>PL</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolutely better</td>
<td>much better</td>
<td>moderately better</td>
<td>slightly better</td>
<td>absolutely better</td>
<td>much better</td>
<td>moderately better</td>
<td>slightly better</td>
</tr>
</tbody>
</table>

15. As the designer develops his design, he may notice that it contains errors or undesirable properties. How do the documentation methods compare with respect to helping the designer detect such problems in his design?

<table>
<thead>
<tr>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>PDL</th>
<th>PDL</th>
<th>PDL</th>
<th>PDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolutely better</td>
<td>much better</td>
<td>moderately better</td>
<td>slightly better</td>
<td>absolutely better</td>
<td>much better</td>
<td>moderately better</td>
<td>slightly better</td>
</tr>
</tbody>
</table>

16. How do the methods compare with respect to helping the designer notice that the same operation is being performed redundantly at several points in the system he is designing?

<table>
<thead>
<tr>
<th>PDL</th>
<th>PL</th>
<th>PDL</th>
<th>PL</th>
<th>PDL</th>
<th>PL</th>
<th>PDL</th>
<th>PL</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolutely better</td>
<td>much better</td>
<td>moderately better</td>
<td>slightly better</td>
<td>absolutely better</td>
<td>much better</td>
<td>moderately better</td>
<td>slightly better</td>
</tr>
</tbody>
</table>
17. With respect to helping him/her notice that a module is too big, and should be broken into several modules?

<table>
<thead>
<tr>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>PDL</th>
<th>PDL</th>
<th>PDL</th>
<th>PDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolutely better</td>
<td>much better</td>
<td>moderately better</td>
<td>slightly better</td>
<td>absolutely much</td>
<td>much</td>
<td>moderately</td>
<td>slightly</td>
</tr>
</tbody>
</table>

18. Helping him/her notice that a module is logically too complex?

<table>
<thead>
<tr>
<th>PDL</th>
<th>PDL</th>
<th>PDL</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolutely better</td>
<td>much better</td>
<td>moderately better</td>
<td>slightly better</td>
<td>absolutely better</td>
<td>much better</td>
<td>absolutely better</td>
</tr>
</tbody>
</table>

19. That a module has too many or too few submodules?

<table>
<thead>
<tr>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolutely better</td>
<td>much better</td>
<td>moderately better</td>
<td>slightly better</td>
<td>absolutely much</td>
<td>much</td>
<td>moderately</td>
<td>slightly</td>
</tr>
</tbody>
</table>

20. That a module contains several logically unrelated functions (i.e., that the module is not cohesive)?

<table>
<thead>
<tr>
<th>PDL</th>
<th>PDL</th>
<th>PDL</th>
<th>PDL</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolutely better</td>
<td>much better</td>
<td>moderately better</td>
<td>slightly better</td>
<td>absolutely much</td>
<td>much</td>
<td>moderately</td>
<td>slightly</td>
</tr>
</tbody>
</table>

21. That a module has not been defined in enough detail?

<table>
<thead>
<tr>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>PDL</th>
<th>PDL</th>
<th>PDL</th>
<th>PDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolutely better</td>
<td>much better</td>
<td>moderately better</td>
<td>slightly better</td>
<td>absolutely much</td>
<td>much</td>
<td>moderately</td>
<td>slightly</td>
</tr>
</tbody>
</table>

22. That a module has been defined in too much detail?

<table>
<thead>
<tr>
<th>PDL</th>
<th>PDL</th>
<th>PDL</th>
<th>PDL</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolutely better</td>
<td>much better</td>
<td>moderately better</td>
<td>slightly better</td>
<td>absolutely much</td>
<td>much</td>
<td>absolutely better</td>
<td>absolutely better</td>
</tr>
</tbody>
</table>
23. That a module is not functionally complete (i.e., that necessary operations are missing)?

<table>
<thead>
<tr>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>PDL</th>
<th>PDL</th>
<th>PDL</th>
<th>PDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolutely better</td>
<td>much better</td>
<td>moderately better</td>
<td>slightly better</td>
<td>better</td>
<td>better</td>
<td>better</td>
<td>better</td>
</tr>
</tbody>
</table>

24. That modules are not sufficiently independent, and are communicating with each other more than they should (i.e., that modules are tightly coupled)?

<table>
<thead>
<tr>
<th>PDL</th>
<th>PDL</th>
<th>PDL</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolutely better</td>
<td>much better</td>
<td>moderately better</td>
<td>slightly better</td>
<td>better</td>
<td>better</td>
<td>better</td>
</tr>
</tbody>
</table>

25. That the overall design is too complex?

<table>
<thead>
<tr>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>PDL</th>
<th>PDL</th>
<th>PDL</th>
<th>PDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolutely better</td>
<td>much better</td>
<td>moderately better</td>
<td>slightly better</td>
<td>better</td>
<td>better</td>
<td>better</td>
<td>better</td>
</tr>
</tbody>
</table>

26. That modules which access the same data are widely separated in the design (data fragmentation)?

<table>
<thead>
<tr>
<th>PDL</th>
<th>PDL</th>
<th>PDL</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolutely better</td>
<td>much better</td>
<td>moderately better</td>
<td>slightly better</td>
<td>better</td>
<td>better</td>
<td>better</td>
</tr>
</tbody>
</table>

27. That a module is trivial, and should be compressed into another module?

<table>
<thead>
<tr>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>PDL</th>
<th>PDL</th>
<th>PDL</th>
<th>PDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolutely better</td>
<td>much better</td>
<td>moderately better</td>
<td>slightly better</td>
<td>better</td>
<td>better</td>
<td>better</td>
<td>better</td>
</tr>
</tbody>
</table>
28. Questions 16 through 27 have asked you how flowcharts and program design languages compare with respect to helping the designer notice a number of specific problems in his/her design. In general, did you find these questions (pick one):

( ) easy to answer.

( ) difficult to answer because you don't see any difference between the documentation methods with respect to these criteria.

( ) Difficult to answer because you couldn't visualize the kinds of design errors and problems referred to in the questions.

29. Overall, how do you think the two documentation methods compare?

<table>
<thead>
<tr>
<th>POL</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
<th>Flowchart</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolutely better</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>much better</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>moderately better</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slightly better</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart absolutely better</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

30. Are there any other documentation methods you think are better than either of these?

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

J-7
31. Please write any other comments you have on these two methods (or others you think are superior). In using both methods for a design problem, you may very well have acquired some insights we don't have. Therefore, please comment on even obvious things if they have not been addressed in this questionnaire.
Explanatory notes:

1) The location counter in this design generally points to the next instruction in contrast to the current instruction as stated in the microNova manual. This tends to simplify the design but has implications concerning tracing and dumping.

2) There are no deliberate errors in the design. If errors are detected or clarification of microNova functional characteristics occurs then revision information will be distributed in the form of memoranda.

3) Input/output devices are not fully simulated. A simulator block input and block output scheme is used instead.

4) Console operations are not simulated in the design.

5) Half of the class is receiving a PDL design and half of the class is receiving a flowchart design. These two designs contain equivalent information. If you are participating in the software design study you are requested to use only the design given you for your simulator implementation.

6) Questions concerning the design should be directed to Dr. J.R. Van Doren. Dr. D.D. Fisher should be consulted concerning questions about microNova functional characteristics.
APPENDIX L - SIMULATOR DESIGN, PDL FORM

Name___________________

COMSC 5113 SIMULATOR DESIGN Spring 1977

Explanatory notes:

1) The location counter in this design generally points to the next instruction in contrast to the current instruction as stated in the microNova manual. This tends to simplify the design but has implications concerning tracing and dumping.

2) There are no deliberate errors in the design. If errors are detected or clarification of microNova functional characteristics occurs then revision information will be distributed in the form of memoranda.

3) Input/output devices are not fully simulated. A simulator block input and block output scheme is used instead.

4) Console operations are not simulated in the design.

5) Half of the class is receiving a PDL design and half of the class is receiving a flowchart design. These two designs contain equivalent information. If you are participating in the software design study you are requested to use only the design given you for your simulator implementation.

6) Questions concerning the design should be directed to Dr. J.R. Van Doren. Dr. D.D. Fisher should be consulted concerning questions about microNova functional characteristics.
MICRONOVA_SIMULATOR:
PROCEDURE;
CALL INITIALIZE_SIMULATOR;
DO WHILE TERMINATION CODE = 0;
FETCH INSTRUCTION;
INCREMENT LOCATION COUNTER;
IF PRIOR INSTRUCTION WAS "MASK OUT" THEN
  TURN ON PRIORITY MASK UPDATE SWITCH;
CALL EXECUTE_INSTRUCTION;
IF PRIORITY MASK UPDATE SWITCH IS ON THEN
  DO;
    CURRENT PRIORITY MASK = NEW PRIORITY MASK;
    TURN OFF PRIORITY MASK UPDATE SWITCH;
  END;
INCREMENT CURRENT CYCLE COUNT BY INSTRUCTION CYCLE COUNT;
IF CURRENT CYCLE COUNT > CYCLE LIMIT THEN
  TERMINATION CODE = EXCESSIVE CYCLE CODE;
IF TRACE IS REQUESTED THEN
  DO;
    PERFORM INSTRUCTION TRACE;
    DECREMENT TRACE COUNTER BY 1;
    IF TRACE COUNTER = 0 THEN TURN OFF TRACE REQUEST;
  END;
IF SKIP INDICATOR IS ON THEN
  DO;
    TURN OFF SKIP INDICATOR;
    INCREMENT LOCATION COUNTER;
  END;
IF REAL TIME CLOCK IS ENABLED THEN
  DO;
    DECREMENT CLOCK CYCLE COUNT BY INSTRUCTION CYCLE COUNT;
    IF CLOCK CYCLE COUNT > 0 THEN
      DO;
        SETUP INTERRUPT CONDITION;
        RESET REAL TIME CLOCK;
      END;
    END;
END;
INSTRUCTION CYCLE COUNT = 0;
DO WHILE INTERRUPT CYCLE COUNT < CURRENT CYCLE COUNT;
  IF DEVICE CODE IS ON AND CURRENT PRIORITY MASK ALLOWS
     DEVICE INTERRUPT THEN SETUP INTERRUPT FROM INTERRUPT LIST;
  UPDATE INTERRUPT CYCLE COUNT TO NEXT INTERRUPT;
END;
IF AN INTERRUPT IS REQUESTED AND INTERRUPTS ARE ENABLED
AND INSTRUCTION EXECUTED WAS NOT "INTERRUPT ENABLE"
THEN CALL PROCESS_INTERRUPT;
END;
PRINT SIMULATOR STATUS INFORMATION UPON TERMINATION;
END MICRONOVA_SIMULATOR;

L-2
INITIALIZE_SIMULATOR: PROCEDURE;
  LOAD MICRONOVA PROGRAM FROM CARD DECK;
  INITIALIZE LOCATION COUNTER AND MAIN MEMORY SIZE;
  INITIALIZE ALL INTERRUPT FLAGS, INPUT/OUTPUT FLAGS, PRIORITY MASKS,
  SIMULATOR CONTROL FLAGS, ETC.;
  INITIALIZE TERMINATION CODE, CURRENT CYCLE COUNT, INSTRUCTION CYCLE
  COUNT TO ZERO;
  INITIALIZE CYCLE COUNT TABLE AND CYCLE LIMIT;
  INITIALIZE INTERRUPT LIST FOR INTERRUPT SIMULATION;
  INITIALIZE REAL TIME CLOCK RESET VALUE AND RANDOM RESET VALUE;
  PERFORM MISCELLANEOUS INITIALIZATION;
  RETURN TO CALLER;
END INITIALIZE_SIMULATOR;

EXECUTE_INSTRUCTION: PROCEDURE;
  CYCLE_TABLE_INDEX = 1;
  IF LEFTMOST BIT OF INSTRUCTION IS '1' THEN CALL TWO_ACCUMULATOR_INST;
  ELSE IF NEXT TWO BITS OF INSTRUCTION ARE NOT '11' THEN
    CALL MEMORY_REFERENCE_INST;
    ELSE
      DO;
        ASSIGN RIGHTMOST SIX BITS OF INSTRUCTION TO DEVICE CODE;
        IF DEVICE CODE = 1 THEN CALL STACK_OR_MUL_DIV;
        ELSE IF DEVICE CODE = 62 THEN CALL TRACE_DUMP;
        ELSE IF DEVICE CODE = 63 THEN CALL CPU_FUNCTION;
        ELSE CALL IO_INST;
      END;
      INCREMENT INSTRUCTION CYCLE COUNT BY CYCLE_TABLE (CYCLE_TABLE_INDEX);
  RETURN TO CALLER;
END EXECUTE_INSTRUCTION;

PROCESS_INTERRUPT: PROCEDURE;
  DISABLE INTERRUPTS;
  INCREMENT INSTRUCTION CYCLE COUNT BY INTERRUPT CYCLE COUNT;
  STORE LOCATION COUNTER IN ZERO POSITION OF MAIN MEMORY VECTOR;
  DO REQUEST LINE = 3 TO 1 BY -1;
    IF INTERRUPT REQUEST (REQUEST LINE) IS ON THEN
      DO;
        EFFECTIVE_ADDRESS = REQUEST_LINE;
        CALL FIND_INDIRECT_ADDRESS;
        ASSIGN EFFECTIVE ADDRESS TO LOCATION COUNTER;
        RETURN TO CALLER;
      END;
    END;
  TERMINATION_CODE = INTERRUPT SIMULATION FAILURE CODE;
  RETURN TO CALLER;
END PROCESS_INTERRUPT;
MEMORY_REFERENCE_INST: PROCEDURE;
CALL COMPUTE_EFFECTIVE_ADDRESS;
IF BITS 1 AND 2 OF INSTRUCTION ARE '00' THEN CALL NO_ACCUMLATOR_INST;
ELSE CALL CN3_ACCUMLATOR_INST;
RETURN TO CALLER;
END MEMORY_REFERENCE_INST;

ONE_ACCUMLATOR_INST: PROCEDURE;
EXTRACT CP CODE AND ACC#;
DO CASE (CP CODE);
CASE0:
TERMINATION CODE = SIMULATOR: FAILURE CODE;
CASE1:
CYCLE TABLE INDEX = 2;
LOAD ACCUMULATOR(ACC#) FROM MEMORY(EFFECTIVE ADDRESS);
END;
CASE2:
CYCLE TABLE INDEX = 3;
STORE ACCUMULATOR(ACC#) IN MEMORY(EFFECTIVE ADDRESS);
END;
CASE3:
TERMINATION CODE = SIMULATOR FAILURE CODE;
END CASE;
RETURN TO CALLER;
END ONE_ACCUMLATOR_INST;

NO_ACCUMLATOR_INST: PROCEDURE;
EXTRACT CP CODE;
DO CASE (CP CODE);
CASE0:
CYCLE TABLE INDEX = 2;
ASSIGN EFFECTIVE ADDRESS TO LOCATION COUNTER;
END;
CASE1:
CYCLE TABLE INDEX = 3;
ASSIGN LOCATION COUNTER TO ACCUMULATOR(3);
ASSIGN EFFECTIVE ADDRESS TO LOCATION COUNTER;
END;
CASE2:
CYCLE TABLE INDEX = 4;
INCREMENT MAIN MEMORY(EFFECTIVE ADDRESS) BY 1;
IF RESULT IS ZERO THEN TURN ON SKIP INDICATOR;
END;
CASE3:
CYCLE TABLE INDEX = 4;
DECREMENT MAIN MEMORY(EFFECTIVE ADDRESS) BY 1;
IF RESULT IS ZERO THEN TURN ON SKIP INDICATOR;
END;
END CASE;
RETURN TO CALLER;
END NO_ACCUMLATOR_INST;
COMPUTE_EFFECTIVE_ADDRESS: PROCEDURE;
DECOMPOSE 11 ADDRESS BITS INTO INDIRECT BIT, INDEXING CODE,
SIGNED DISPLACEMENT, UNSIGNED DISPLACEMENT;
DO CASE (INDEXING CODE);
CASE0;
  EFFECTIVE ADDRESS = UNSIGNED DISPLACEMENT;
CASE1;
  EFFECTIVE ADDRESS = LOCATION COUNTER - 1 + SIGNED DISPLACEMENT;
CASE2;
  EFFECTIVE ADDRESS = ACCUMULATOR(2) + SIGNED DISPLACEMENT;
CASE3;
  EFFECTIVE ADDRESS = ACCUMULATOR(3) + SIGNED DISPLACEMENT;
END CASE;
RETAIN RIGHTMOST 15 BITS OF EFFECTIVE ADDRESS;
IF INDIRECT BIT IS ON THEN CALL FIND_INDIRECT_ADDRESS;
RETURN TO CALLER;
END COMPUTE_EFFECTIVE_ADDRESS;

FIND_INDIRECT_ADDRESS: PROCEDURE;
  INITIALIZE REFERENCE COUNT TO 2;
  DO WHILE REFERENCE COUNT < 15;
    INCREMENT INSTRUCTION CYCLE COUNT BY INDIRECT ADDRESS CYCLE COUNT;
    RETRIEVE INDIRECT WORD FROM MAIN MEMORY(EFFECTIVE ADDRESS);
    IF 16 ≤ EFFECTIVE ADDRESS ≤ 31 THEN
      DO;
        IF EFFECTIVE ADDRESS ≤ 23 THEN INCREMENT INDIRECT WORD BY 1;
        ELSE DECREMENT INDIRECT WORD BY 1;
        STORE INDIRECT WORD IN MAIN MEMORY(EFFECTIVE ADDRESS);
        INCREMENT INSTRUCTION CYCLE COUNT BY AUTO INDEX CYCLE COUNT;
        INCREMENT REFERENCE COUNT BY 1;
        IF REFERENCE COUNT > 15 THEN
          DO;
            TERMINATION CODE = INDIRECT OVERFLOW CODE;
            RETURN TO CALLER;
          END;
        END;
      END;
    END;
    ASSIGN RIGHTMOST 15 BITS OF INDIRECT WORD TO EFFECTIVE ADDRESS;
    IF LEFTMOST BIT OF INDIRECT WORD = 0 THEN RETURN TO CALLER;
    INCREMENT REFERENCE COUNT BY 2;
  END;
  TERMINATION CODE = INDIRECT OVERFLOW CODE;
  RETURN TO CALLER;
END FIND_INDIRECT_ADDRESS;
TWO_ACCUMULATOR_INSTR: PROCEDURE;
    DECODE INSTRUCTION INTO SOURCE ACC#, DESTINATION ACC#, OP Code, 
    SHIFT ACTION, CARRY BIT ACTION, RESULT ACTION, SKIP ACTION; 
    IF RESULT ACTION = 0 AND SKIP ACTION = 0 THEN 
        DO;
        CALL TRAP_INST;
        CYCLE TABLE: INDEX = 4;
        RETURN TO CALLER;
    END;

    SET CARRY BIT ACCORDING TO CARRY BIT ACTION;
    OPERAND = ACCUMULATOR(SOURCE ACC#);
    DO CASE (OP CODE);
    CASE 0:
        PERFORM COMPLEMENT OF OPERAND WITH RESULT ASSIGNED TO SHIFTER;
    CASE 1:
        PERFORM TWO'S COMPLEMENT NEGATION OF OPERAND WITH RESULT 
        ASSIGNED TO SHIFTER;
    CASE 2:
        ASSIGN OPERAND TO SHIFTER;
    CASE 3:
        ADD 1 TO OPERAND WITH RESULT ASSIGNED TO SHIFTER;
    CASE 4:
        ADD COMPLEMENT OF OPERAND TO ACCUMULATOR(DESTINATION ACC#) WITH 
        RESULT ASSIGNED TO SHIFTER;
    CASE 5:
        SUBTRACT OPERAND FROM ACCUMULATOR(DESTINATION ACC#) WITH 
        RESULT ASSIGNED TO SHIFTER;
    CASE 6:
        ADD OPERAND TO ACCUMULATOR(DESTINATION ACC#) WITH RESULT 
        ASSIGNED TO SHIFTER;
    CASE 7:
        LOGICALLY "AND" OPERAND BITS WITH ACCUMULATOR(DESTINATION ACC#) 
        BITS WITH RESULT ASSIGNED TO SHIFTER;
    END CASE;
    SET LEFTMOST BIT OF SHIFTER ACCORDING TO FUNCTIONAL SPECIFICATIONS;
    PERFORM SPECIFIED SHIFT ACTION ON SHIFTER;
    SET SKIP INDICATOR ACCORDING TO SKIP ACTION SPECIFICATIONS;
    IF RESULT ACTION = 1 THEN ACCUMULATOR(DESTINATION ACC#) = 
        RIGHTMOST 16 BITS OF SHIFTER;
    ELSE IF SKIP ACTION = 0 THEN TERMINATION CODE = SIMULATOR FAILURE CODE;
    CYCLE TABLE INDEX = 1;
    RETURN TO CALLER;
    END_TWO_ACCUMULATOR_INSTR;

TRAP_INST: PROCEDURE;
    ASSIGN LOCATION COUNTER TO MAIN MEMORY(38);
    EFFECTIVE ADDRESS = 39;
    CALL FIND INDIRECT_ADDRESS;
    ASSIGN EFFECTIVE ADDRESS TO LOCATION COUNTER;
    RETURN TO CALLER;
    END TRAP_INST;
STACK CR_NULL DIV: PROCEDURE;
  IF MULTIPLY OR DIVIDE INSTRUCTION THEN
    DO;
      IF DIVIDE INSTRUCTION THEN
        DO;
          IF ACCUMULATOR(0) ≥ ACCUMULATOR(2) THEN
            DO;
              CARRY BIT = 1;
              OVERFLOW CODE = 1;
              RETURN TO CALLER;
            END;
            PERFORM DIVISION OF CATENATED ACCUMULATORS 0 AND 1
            BY ACCUMULATOR(2);
            ASSIGN QUOTIENT AND REMAINDER TO ACCUMULATORS 0 AND 1,
            RESPECTIVELY;
            RETURN TO CALLER;
          END;
          PERFORM MULTIPLICATION OF ACCUMULATOR(1) BY ACCUMULATOR(2) WITH
          ACCUMULATOR(0) ADDED TO RESULT;
          ASSIGN RESULT TO ACCUMULATORS 0 AND 1;
          RETURN TO CALLER;
        END;
        DECOMPOSE INSTRUCTION INTO STACK OP CODE AND ACC#;
        DO CASE (STACK OP CODE);
        CASED:
          DO;
            CYCLE TABLE INDEX = 1;
            ASSIGN ACCUMULATOR(ACC#) TO FRAME POINTER;
          END;
        CASE1:
          DO;
            CYCLE TABLE INDEX = 1;
            ASSIGN FRAME POINTER TO ACCUMULATOR(ACC#);
          END;
        CASE2:
          DO;
            DO LOOP = 0 TO 2 BY 1;
            INCREMENT STACK POINTER BY 1;
            IF STACK OVERFLOW THEN
              DO;
                SETUP STACK INTERRUPT REQUEST;
                RETURN TO CALLER;
              END;
              ASSIGN ACCUMULATOR(Loop) TO MAIN MEMORY(STACK POINTER);
            END;
            INCREMENT STACK POINTER BY 1;
            CYCLE TABLE INDEX = 7;
            ASSIGN FRAME POINTER TO MAIN MEMORY(STACK POINTER);
            INCREMENT STACK POINTER BY 1;
            ASSIGN CARRY BIT AND ACCUMULATOR(3) TO MAIN MEMORY(STACK POINTER);
            ASSIGN STACK POINTER TO FRAME POINTER AND TO ACCUMULATOR(3);
          END;
  END;
CASE3:
DO;
    CYCLE TABLE INDEX = 6;
    ASSIGN FRAME POINTER TO STACK POINTER;
    ASSIGN CARRY BIT AND LOCATION COUNTER FROM MAIN MEMORY(STACK POINTER);
    DECREMENT STACK POINTER BY 1;
    DC LOOP = 3 TO 0 BY -1;
    ASSIGN MAIN MEMORY(STACK POINTER) TO ACCUMULATOR(LOOP);
    DECREMENT STACK POINTER BY 1;
END;
END CASE;
CASE4:
DO;
    CYCLE TABLE INDEX = 1;
    ASSIGN ACCUMULATOR(ACC#) TO STACK POINTER;
END;
CASE5:
DO;
    CYCLE TABLE INDEX = 1;
    ASSIGN STACK POINTER TO ACCUMULATOR(ACC#);
END;
CASE6:
DO;
    CYCLE TABLE INDEX = 3;
    INCREMENT STACK POINTER BY 1;
    IF STACK OVERFLOW THEN SETUP STACK INTERRUPT;
    ELSE ASSIGN ACCUMULATOR(ACC#) TO MAIN MEMORY(STACK POINTER);
END;
CASE7:
DO;
    CYCLE TABLE INDEX = 3;
    ASSIGN MAIN MEMORY(STACK POINTER) TO ACCUMULATOR(ACC#);
    DECREMENT STACK POINTER BY 1;
END;
END CASE;
RETURN TO CALLER;
END STACK_OR_MUL_DIV;
CPU_FUNCTIONS: PROCEDURE;
  DECOMPRESS INSTRUCTION INTO OP CODE, ACC#, INTERRUPT SPECIFICATION;
  DO CASE (OP CODE);
    CASE0:
      IF INSTRUCTION IS "INTERRUPT ENABLE" AND INTERRUPTS ARE NOT
      ENABLED THEN IDENTIFY INSTRUCTION FOR MAIN SIMULATOR LOOP;
    CASE1:
      EXECUTE "CASE2" CODE;
    CASE2:
      DO CASE (ACC#);
        INNER CASE0:
          RESET BUSY FLAGS, DONE FLAGS, DEVICE ON FLAGS,
          CURRENT PRIORITY MASK;
        INNER CASE1:
          DISABLE REAL TIME CLOCK;
        INNER_CASE2:
          DO;
            ENABLE REAL TIME CLOCK;
            INITIALIZE REAL TIME CLOCK TO RANDOM RESET VALUE;
          END;
        INNER_CASE3:
          TERMINATION CODE = INVALID INSTRUCTION CODE;
      END CASE;
    CASE3:
      SET ACCUMULATOR(ACC#) BITS ACCORDING TO "INTERRUPT ACKNOWLEDGE"
      SPECIFICATIONS;
    CASE4:
      DO;
        ASSIGN ACCUMULATOR(ACC#) TO NEW PRIORITY MASK;
        IDENTIFY "MASK OUT" INSTRUCTION FOR MAIN SIMULATOR LOOP;
      END;
    CASE5:
      TERMINATION CODE = INVALID INSTRUCTION CODE;
    CASE6:
      DO;
        TERMINATION CODE = "HALT" INSTRUCTION CODE;
        ENABLE INTERRUPTS;
        RETURN TO CALLER;
      END;
  END CASE;
  ENABLE OR DISABLE INTERRUPTS ACCORDING TO INTERRUPT SPECIFICATION;
  RETURN TO CALLER;
END CPU_FUNCTIONS;
IO_INST: PROCEDURE;
  DECOMPOSE INSTRUCTION INTO IO OP CODE AND DEVICE CODE;
  IF IO OP CODE = 7 THEN
    DO;
      SET SKIP INDICATOR ACCORDING TO "ICSKIP" INSTRUCTION SPECIFICATIONS;
      RETURN TO CALLER;
    END;
  IF IO OP CODE IS A "DATA IN" OPERATION THEN
    DO;
      IF DEVICE CODE = 40 THEN
        DO;
          PERFORM SIMULATOR BLOCK INPUT ACCORDING TO THE
          FOLLOWING INSTRUCTIONS;
          EFFECTIVE ADDRESS = LOCATION COUNTER;
          INCREMENT LOCATION COUNTER BY 1;
          CALL FIND INDIRECT ADDRESS;
          INPUT 16 WORDS FROM SYSIN ASSIGNING THEM TO MAIN MEMORY
          STARTING AT EFFECTIVE ADDRESS;
        END;
        ELSE IGNORE "DATA IN OPERATION FOR SIMULATION PURPOSES;
      END;
    ELSE IF IO OP CODE IS A "DATA OUT" OPERATION THEN
      DO;
        IF DEVICE CODE = 41 THEN
          DO;
            PERFORM SIMULATOR BLOCK OUTPUT ACCORDING TO THE
            FOLLOWING INSTRUCTIONS;
            EFFECTIVE ADDRESS = LOCATION COUNTER;
            INCREMENT LOCATION COUNTER BY 1;
            CALL FIND INDIRECT ADDRESS;
            STARTING AT EFFECTIVE ADDRESS IN MAIN MEMORY
            OUTPUT 16 WORDS TO SYSPRINT;
          END;
          ELSE IGNORE "DATA OUT" OPERATION FOR SIMULATION PURPOSES;
        END;
    END;
  SET BUSY AND DONE FLAGS ACCORDING TO SPECIFICATIONS;
  RETURN TO CALLER;
END IO_INST;
TRACE_DUMP: PROCEDURE;

EXTRACT IO FORMAT ACC# FROM INSTRUCTION;
EXTRACT COUNT FROM BITS 5 TO 9 OF INSTRUCTION;
DO CASE (ACC#);
CASE0:
  DO;
    IF TRACE REQUEST IS NOT ON THEN
      DO;
        IF COUNT = 0 THEN RETURN TO CALLER;
        ASSIGN LOCATION COUNTER - 1 TO ACTIVE TRACE LOCATION;
        TURN ON TRACE REQUEST;
        TRACE COUNT = COUNT;
        RETURN TO CALLER;
      END;
    IF ACTIVE TRACE LOCATION \ne LOCATION COUNTER - 1 THEN
      RETURN TO CALLER;
    DECREMENT COUNT BY 1;
    STORE COUNT BITS BACK INTO CORRESPONDING INSTRUCTION BITS
    AND RESTORE MODIFIED INSTRUCTION TO MAIN MEMORY;
    IF COUNT = 0 THEN
      DO; ACTIVE TRACE LOCATION = 0; TURN OFF TRACE REQUEST; END;
    END;
  CASE1:
    DO; ACTIVE TRACE LOCATION = 0; TURN OFF TRACE REQUEST; END;
  CASE2:
    PERFORM DUMP;
  CASE3:
    DO; PERFORM DUMP; TERMINATION CODE = DUMP EXIT CODE; END;
END CASE;
RETURN TO CALLER;
END TRACE_DUMP;
APPENDIX M - SIMULATOR DESIGN REVISION MEMO

Errors and corrections in the simulator design (both versions)

1) STACK.OR.MUL.DIV module (page 6)
The relational operator of the division overflow test is missing or incomplete. It should be

\[ \text{accumulator(0)} \leq \text{accumulator(2)}. \]

2) STACK.OR.MUL.DIV module (page 6)
The accumulator numbers for the placement of quotient and remainder results for division are reversed.

3) STACK.OR.MUL.DIV module (page 6)
The stack overflow test should be performed after stacking is completed.

4) TWO-ACUMULATOR-INST (pages)
The result action bit test for detecting a TRAP instruction is inverted.

5) INITIALIZE_SIMULATOR (page 2)
Initialization of main memory size should be performed before "loading" the micro Nova program.
APPENDIX N - DESIGN COMPREHENSION TEST

Name: ____________________________

QUIZ

This quiz consists of thirty multiple-choice questions and two fill-in questions. Please pace yourself so that you can complete all questions in the time allowed. Answer all questions. If you don't know the answer to a multiple-choice question, eliminate any responses which are clearly wrong, and guess among the remaining responses.
DO NOT TURN THE PAGE until you have completed the work on this page.  
Then do not return to this page.

List below the names of all the modules in the simulator. List them 
exactly if you can; if you are unable to reproduce a module name 
exactly, then come as close as you can.
For each of the multiple-choice questions which follow, select the one best alternative by circling the letter which precedes it.

Example: Most automobiles are powered by
(a) steam engines.
(b) turbine engines.
(c) internal-combustion engines.
(d) horses.

1. According to the design, a simulator entity called DEVICE CODE
   (a) is used to identify a peripheral device for I/O instructions.
   (b) is used to prevent some instruction classes from being interpreted as I/O instructions.
   (c) both (a) and (b).
   (d) none of these.

2. Assume that the microNova simulator has been implemented in accordance with the design you have been given, and that you are now debugging the simulator. You observe that the "Load Accumulator" instruction works correctly, but the "Store Accumulator" instruction does not. Which module probably contains the error?
   (a) EXECUTE_INSTRUCTION
   (b) CPU_FUNCTIONS
   (c) COMPUTE_EFFECTIVE_ADDRESS
   (d) MEMORY_REFERENCE_INST

3. According to the design, the location counter is incremented
   (a) immediately before instruction fetching.
   (b) immediately after instruction fetching.
   (c) immediately after instruction execution.
   (d) if and only if the skip indicator is on.

4. The multiway branch specifications used in the design specify for an "out of range" condition that
   (a) a precise, but variable, action is always to be taken.
   (b) nothing is to be done as a consequence.
   (c) an error message is to be printed and execution continued.
   (d) the termination code is to be set to an "abort" code.

5. How many input/output device codes does the microNova allow?
   (a) 16
   (b) 32
   (c) 64
   (d) 128
6. Suppose the MEMORY REFERENCE INST module is invoked immediately upon entry to the EXECUTE_INSTRUCTION module. (Assume the invocation in the current design is removed.) The implications are that

(a) redundant processing will occur but logical equivalence with the current design results.
(b) logical equivalence will result with no redundant processing.
(c) the CYCLE TABLE INDEX value may be incorrectly set.
(d) incorrect simulation may result.

7. The manner in which the current cycle count is used in the simulator will

(a) prevent infinite loops in the simulator program.
(b) prevent infinite loops in the microNova program.
(c) both (a) and (b).
(d) none of these.

8. Suppose the simulator will eventually be used to simulate microNova programs in object deck form whose format is not yet known. In order to simulate these programs

(a) the instruction execution module will have to be redesigned.
(b) simulation of interrupts and I/O will need revision.
(c) simulator initialization will require revision.
(d) the simulator main procedure will need revision.

9. If the specifications for FETCH INSTRUCTION and invocation of EXECUTE_INSTRUCTION are interchanged, the implications are that

(a) logical equivalence with the current design will result.
(b) logical equivalence with the current design will result if simulator initialization is modified to fetch the first instruction.
(c) logical equivalence with the current design will result if all location counter value references are changed and simulator initialization is modified to fetch the first instruction.
(d) logical equivalence with the current design will require substantial redesign of several modules.

10. Suppose you are debugging the simulator, and you observe that you can successfully turn the trace on, that trace information is correctly printed, but that you cannot turn the trace off. Which module probably contains the error?

(a) MICRONOVA SIMULATOR (main procedure)
(b) EXECUTE_INSTRUCTION
(c) TRACE_DUMP
(d) IO_INST
11. In the interrupt processing procedure there is a section of logic for assigning a failure code to the termination code. If this logic is executed the meaning is that

(a) there is an error in the microNova program.
(b) there is an error in the indirect address computation.
(c) there is an error in the simulator program.
(d) none of these.

12. The failure code reference in question 11 represents

(a) a simulation of a microNova hardware feature.
(b) a simulation of a feature in the microNova interrupt service program.
(c) a feature of the simulation design.
(d) all of the above.

13. The current cycle count represents

(a) the number of microNova instructions simulated.
(b) the index of the main simulator loop.
(c) an approximation to the time of execution of the simulator program.
(d) an approximation to the time of execution of the microNova program.

14. In microNova instructions which involve relative addressing, a part of the instruction word contains displacement information. How is that information represented?

(a) one's complement
(b) two's complement
(c) excess-64
(d) excess 128

15. If an effective address is computed which is outside the range of addresses of the machine being simulated, the design

(a) clearly specifies that "wraparound" addressing results.
(b) specifies that an "abort" termination code is to be set.
(c) is not very precise about the action to be taken.
(d) none of these.
16. According to the design, the procedure labelled FIND INDIRECT ADDRESS

(a) may be used to follow a chain of indirect addresses.
(b) is always used to retrieve the contents of exactly one word; thus a chain of indirect addresses is processed by repeated invocation of the named procedure.
(c) may not be invoked until address modification due to indexing (if any) has been performed.
(d) could easily be absorbed into the design of the module for memory reference instructions.

17. The cycle table index is used for

(a) indexing a table used for recording statistics about the simulation process.
(b) retrieving approximate instruction execution times.
(c) a loop index during cycle table processing.
(d) none of these.

18. The index bits for memory reference instructions, interpreted as a nonnegative integer value are used in the simulator design

(a) as an index to a vector of size four used for simulating the accumulators used as index registers.
(b) to modify the signed displacement.
(c) as a decrement value for auto index word references.
(d) to determine an "intermediate" effective address just before possible modification due to indirect addressing.

19. If the skip indicator is removed from the design, then it will be necessary to make which one of the following additional changes in the design?

(a) immediately after returning from the instruction execution routine, the instruction just executed should be tested to determine whether the location counter is to be incremented.
(b) modify the design for simulation of the JUMP instruction.
(c) whenever an instruction is fetched, it should be tested immediately for a skip specification; if one is present, the location counter should be incremented.
(d) wherever the skip indicator is turned on in the current design, replace that with a location counter increment.

20. Suppose you are debugging the simulator, and you observe that "Trap" instructions work correctly, and that "Load Accumulator" and "Jump" instructions do not work when they use indirect addressing, but succeed otherwise. Which module probably contains the error?

(a) FIND INDIRECT ADDRESS
(b) COMPUTE EFFECTIVE ADDRESS
(c) MEMORY REFERENCE INST
(d) EXECUTE INSTRUCTION
21. For purposes of illustration, suppose there are 25 instructions in the microNova instruction set. An alternative way of designing the simulator might be to provide a procedure to determine which one of the 25 instructions is to be simulated and then have a 25-way branch to 25 different sets of code which perform the actual simulation. Compared to the design given to you, the implications of such an approach are that

(a) substantial duplication of code might result unless such duplication is removed to subprograms.
(b) modification of the resulting simulator would be simpler, although initial implementation would take longer.
(c) execution time for the simulator would be significantly shorter.
(d) none of these.

22. The purpose of the CPU_FUNCTIONS module is to

(a) assign values to CPU registers from a special input device.
(b) simulate instructions associated with interrupts.
(c) simulate the microNova CPU.
(d) none of these.

23. The outermost loop of the simulator is controlled by

(a) the instruction cycle count.
(b) pending interrupts.
(c) a loop index.
(d) a termination code.

24. The tracing and dumping features may be used for

(a) debugging microNova programs.
(b) debugging the simulator.
(c) both (a) and (b).
(d) none of these.

25. If an interrupt is to be simulated, the interrupt processing procedure

(a) fetches the first instruction of the interrupt service routine.
(b) decrements a value in simulated memory by one.
(c) invokes the procedure for computation of effective addresses for proper determination of the location counter value.
(d) none of the above.
26. The function of the microNova shifter, when executing a left shift operation, can best be described as a

(a) 16-bit end-around shift.
(b) 16-bit no-end-around shift.
(c) 17-bit end-around shift.
(d) 17-bit no-end-around shift.

27. Suppose the design is to be revised so the body of the outer loop of the simulator is shorter but the entire design is logically equivalent to the original design. Without changing any other procedures, it would be possible to put some of the processing in

(a) the simulator initialization module.
(b) the instruction execution module.
(c) the module which computes effective addresses.
(d) any of the above.

28. The purpose of the simulator initialization procedure is to

(a) load a microNova machine language program into simulated memory.
(b) initialize simulated machine flags and registers.
(c) initialize simulator program control variables.
(d) all of the above.

29. Suppose you are debugging the simulator, and you observe that "Interrupt Enable" instructions do not work correctly. Which module probably contains the error?

(a) CPU FUNCTIONS
(b) PROCESS_INTERRUPT
(c) IO_INST
(d) EXECUTE_INSTRUCTION

30. The maximum number of microNova instructions simulated in one pass through the body of the outermost simulator loop is

(a) 1.
(b) 2.
(c) 3.
(d) 4 or more.
Shown on the next page is an adjacency matrix. The matrix is to be used to indicate which modules invoke which other modules directly. Rows represent calling modules, while columns represent called modules. You are to enter an "X" in each cell which represents a calling-module/called-module relation in the simulator design. Be sure you understand the example below before you fill in the matrix on the next page.

Example: Suppose that a design contains modules A, B, C, and D. Suppose that A calls B and C, and that B calls C and D. Then the matrix for this design has the form:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Called</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>module</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Calling module</td>
<td>TRACE_DUMP</td>
<td>INITIALIZE_SIMULATOR</td>
<td>COMPUTE_EFFECTIVE_ADDRESS</td>
<td>IO_INST</td>
</tr>
<tr>
<td>----------------</td>
<td>------------</td>
<td>----------------------</td>
<td>--------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>TRACE_DUMP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INITIALIZE_SIMULATOR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMPUTE_EFFECTIVE_ADDRESS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO_INST</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MICRONOVA_SIMULATOR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEMORY_REFERENCE_INST</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIND_INDIRECT_ADDRESS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRAP_INST</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU_FUNCTIONS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXECUTE_INSTRUCTION</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STACK_OR_MUL_DIV</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TWO_ACCUMULATOR_INSTR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROCESS_INTERRUPT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX O - TIME EXPENDITURE QUESTIONNAIRE, IMPLEMENTATION PHASE

WEEKLY TIME QUESTIONNAIRE (Implementation Phase)

Name: ____________________________

Date: _______________ Code: ______________________

Please record the time you have spent on your class project in the last week, excluding time spent in class. Record time in hours and/or minutes by category.

Time spent in: hrs : min

Initial code preparation ......................................................... : 
Modification of existing code ................................................. : 
Keypunching ................................................................. : 
Preparation of test cases ...................................................... : 
Execution and interpretation of test runs ................................. : 
Preparation of documentation ............................................... : 
Just thinking about project ................................................. : 
Other (indicate nature of activity) ...................................... : 

This information is solely for experimental purposes; your responses will in no way affect your course grade. Please be as accurate as possible.
APPENDIX P - EXAMPLES OF SUBJECT-GENERATED DESIGNS

Pass 1

Description of Procedure: Pass 1

Pass One: Use one of the assemblers described for Scan

Decode: Used to decode instructions and aid Pass One

Assemble: Assemble the instructions

Copy: A procedure to copy the assembled and

Input image to a file for future use

in Pass Two

Pass Two: Pass two of the assemblers described for Scan

Op Code Table: This procedure does a table look up

and returns the value specified below for each op code

Store 1 Write 13 DC 20

Load 2 Read 14

Sub 5 Div 0

Add 6 Halt 0

Bro 8 Prog 16

Br1 9 Equate 17

Br2 10 Loc 18

Br3 11 DS 19

Table: A procedure to build the symbol (table)

Check: A procedure to check for multiple symbol table entries

"Average quality" design from Group A/ Pages 1, A/4, Condition
**Procedure variables:**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COUNTER</td>
<td>The counter value</td>
</tr>
<tr>
<td>CODE</td>
<td>The operation code of the input instruction</td>
</tr>
<tr>
<td>TEMP</td>
<td>Value returned by OP CODE table</td>
</tr>
<tr>
<td>IMAGE</td>
<td>The image of the assembled instruction</td>
</tr>
<tr>
<td>ERR</td>
<td>The error field of the instruction</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ERR values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error</td>
</tr>
<tr>
<td>1</td>
<td>Invalid OP code</td>
</tr>
<tr>
<td>2</td>
<td>Missing END statement</td>
</tr>
<tr>
<td>3</td>
<td>Misplaced PROG statement</td>
</tr>
<tr>
<td>4</td>
<td>Missing label</td>
</tr>
</tbody>
</table>

| Loc | The label in the input image |
| Addr | The address field of the input image |

<table>
<thead>
<tr>
<th>Comments</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. If a PROG statement does not appear, just the location counter is set to 0. A PROG appearing anywhere else in the program is considered to be an error.</td>
<td></td>
</tr>
</tbody>
</table>
PASS ONE: Procedure;

    COUNTER = 0;

    Do until all instructions are processed;

    Get next instruction;

    If CODE = END then do;

        Call Check;

        Call PASS TWO;

        END;

    Else do;

        Call DECODE (CODE);

        Call Copy;

        END;

    END;

    END PASS ONE;

Procedure Check

    Sort the symbol table into alphabetic order;

    Check to see if any two consecutive entries are the same;

    END CHECK:
DECODER: PRESERVE (CODE);

CALL OP-CODE-TABLE (CODE, TEMP);

IMAGE:= IMAGE + CODE;

IF CODE<0 THEN DO:

IMAGE := 0

ERR := 1

END;

DO CASE TEMP:

CASE 1:

IF COUNTER = 0 THEN DO:

IMAGE := 0

ERR := 3

END;

ELSE IF LOC > MEMORY_SIZE THEN DO:

ERR := 5;

IMAGE := 0

END;

ELSE COUNTER := LOC;

RETURN

END;
CASE 17: DO;
    
    IF LOC = BLANK THEN DO;
        IMAGE = 0;
        EER = Q;
    END;
    
    ELSE CALL TABLE (LOC, COUNTER);
    RETURN;
    END;

CASE 18: DO;
    
    IF LOC > MEMORY SIZE THEN EER = 5
    ELSE COUNTER = LOC ;
    RETURN;
    END;

CASE 19: DO;
    
    IF LOC # BLANK THEN CALL TABLE (LOC, COUNTER);
    COUNTER = COUNTER + ADDR;
    IF COUNTER > MEMORY SIZE THEN EER = 5;
    IMAGE = 0
    RETURN
    END;
CASE AE: DO:

IF ADOR > MEMORY - SIZE THEN EIR = 5;

IMAG = ADOR;

END;

END;

IF LOC ≠ BLANK THEN CALL TABLE (LOC, COUNTER);

COUNTER = COUNTER + 1

IF COUNTER > MEMORY - SIZE THEN EIR = 5;

RETURN;

END DECIDE;
"Average quality" design from Group B, Pass 1 Flowchart Condition

[Flowchart diagram with steps and conditions labeled]
Decode_operation:

Isolate:
Card_image, label, operation, errors?

Did missing operation or string error occur?

Identify:
Operation, class, value, errors.

Return:

Operation string classifications:
Class Op. Mem. (Value)
1. PROG
2. DC, D10
3. DS
4. LIST(1), TITLE(2), PAGE(3), LINE(4)
5. EQUATE(1), LOC(2), END(3)
6. HALT(230), STORE(1560), LOAD(2303), SUB(2303), ADD(2303), BR( ), WRITE(1203), READ(1203)
7. ECVL(1), BRWSY(2), BW3SY(3)
8. UNDEFINED

Base 16 values

p-9
FLTFCH_CARD:

READ A CARD INTO CARD_IMAGE

IS THE INPUT A COMMENT?

YES

SAVE
"print only"
@1
@2
CARD_IMAGE
(@, @, @,...)

NO

RETURN
Primitives

1) SAVE - save code and card image for Pass 2
   Input - Pass 2 classification
   - Storage length
   - Instruction code/Pass 2 operation code
   - card image
   - Pass 1 error codes (vector)

2) INSERT - insert symbol into symbol table
   Input - Symbol
   - Value
   Output - Multiply def. error

3) RETRIEVE - retrieve symbol value from symbol table
   Input - Symbol
   Output - Value
   - Undefined error

4) ISOLATE - scan card image for label and operation string
   Input - Card image
   Output - Label string, Operation string
   - Scanner errors (invalid label, missing operation string)

5) IDENTIFY - look up operation string
   Input - Operation string
   Output - Class
   - Opcode/Assembler_operation code
   - Undecodable operation string error
     (class and opcode set to zero)

6) EVALUATE - evaluate operands
   Output - Operand value
"Average quality" design from Group A. Pass 2. Flowchart condition

START

Open the file containing Pass 2 information

Read the entire record (i.e., line of Pass 1 output)

Open a new file to generate absolute op code

Counter = 0

IS THIS STATEMENT PROGRAM NO

READ ANOTHER RECORD

SYMABLE TABLE

IS THIS A PSEUDO- INSTR.

YES

ERROR MESSAGE

IS ADDR. OUT OF RANGE?

YES

1st WORD (NEW FILE) ← ADDRESS

3rd WORD (TRAILING REG) ← ADDRESS

NO

P-12
WRITE HEX-IMAGE ON OUTPUT FILE

COUNTER = COUNTER

COUNTER ≤ 38

NO

LEADING RECORD
2ND WORD ← COUNTER
TRAIL'N 1ST WORD ← 0

END OF OUTPUT BINARY FILE

YES

RALLY 2ND ← 0
3RD ADDR: NEXT INSTR.
BEGINNING IN NEXT RECORD

STOP

P-13
PASS_TWO: procedure;

   do while run_code = continue execution code;
   read record if end of file then run_code = halt termination code;
   else do;
   
   determine instruction_class;
   do case (instruction_class);
   
   case 0:
       call process_symbolic_machine_instruction;
   
   case 1:
       call process_constant_definitions;
   
   case 2:
       call process_assembler_instructions;
   
   case 3:
       call process_invalid_instructions;
       end case;
   
   end;

   print pass_two status information;

   output the output buffer if not empty;

   1st_word < 0

   2nd_word < 0

   3rd_word < abs_addr_of_starting_instr.

   end pass_two;

"Average quality design from group by pass 2, PDL condition"
PROCESS ASSEMBLER INSTRUCTIONS: PROCEDURE;

CALL OUTPUT;

RETURN TO CALLER;

END PROCESS ASSEMBLER INSTRUCTIONS;
PROCESS_SYMBOLIC_MACHINE_INSTRUCTIONS:  Procedure:  

IF op = B THEN IF ADDR2 = SYMBOLIC THEN CALL SEARCH_SYMBOL_TABLE(HAD, ADDR
ELSE HAD <- ADDR2;  
ELSE IF ADDR1 = SYMBOLIC THEN CALL SEARCH_SYMBOL_TABLE(HAD, ADDR)
ELSE HAD <- ADDR1;

INSTR <- HOP(HAD);

CALL OUTPUT;

CALL LOAD(INSTR, HL); RETURN TO CALLER;

END PROCESS_SYMBOLIC_MACHINE_INSTRUCTION,
PROCESS_INVALID_INSTRUCTION: PROCEDURE;

CALL OUTPUT;

CALL LOAD (INSTR, HLOC);

RETURN TO CALLER;

END PROCESS INVALID INSTRUCTION;
PROCESS_CONSTANT_DEFINITION: PROCEDURE

IF OP = DC THEN DO;  
   CALL OUTPUT;
   CALL LOAD (INSTR, HLOC);
END;

ELSE DO;
   IF ADDR1 = SYMBOLIC THEN CALL SEARCH_SYMBOL_TABLE (HAD, ADDR1);
   ELSE HAD < ADDR1;
   IF ADDR2 = SYMBOLIC THEN CALL SEARCH_SYMBOL_TABLE (HAD, ADDR2);
   ELSE HAD < ADDR2;
   LAST
   INSTR < 1ST 10 BITS OF ADDR1 || LAST 6 BITS OF ADDR2;
   CALL OUTPUT;
   CALL LOAD (INSTR, HLOC);
END;

END PROCESS_CONSTANT_DEFINITION;
LOAD: PROCEDURE (INSTR, HLOC);

IF 2nd WORD OF BUFFER = 38 OR HLOC + 2nd WORD + 1st WORD OF BUFFER OR

IF LOAD IS CALLED THE 1ST TIME

THEN DO;

IF LOAD IS NOT CALLED THE 1ST TIME THEN OUTPUT BUFFER;
ELSE SAVE ABS_ADDR OF STARTING INSTRUCTION;
1ST WORD OF BUFFER ← HLOC;

2nd WORD OF BUFFER ← 1;

3rd WORD OF BUFFER ← INSTR;

END;

ELSE DO;

INCREMENT 2nd WORD OF BUFFER BY 1;

(2nd WORD + 2)th WORD OF BUFFER ← INSTR;

END

RETURN TO CALLER;

END LOAD;
SEARCH_SYMBOL_TABLE; PROCEDURE (HAD, AD)

SEARCH SYMBOL TABLE FOR AD AND RETURN HES_LOCATION (HAD)
of AD to caller; IF SYMABLE NOT FOUND HAD ← 0

END SEARCH_SYMBOL_TABLE;
OUTPUT: PROCEDURE

PRINT HEX IMAGE, ERROR NOTES, INPUT IMAGE

END OUTPUT
PROCEDURE OPTICS (MAIN);

COMSC 5113 TERM PROJECT — SIMULATION OF THE DATA GENERAL

MICRONOVA COMPUTER ON THE IBM SYSTEM 370

AUTHOR: [Redacted]

DUE DATE: 4-20-77

STATEMENT OF THE PROBLEM: TO IMPLEMENT AND TEST THE MICRONOVA

COMPUTER BY DATA GENERAL; THE INSTRUCTION SETS USED WHERE

TWO ACCUMULATOR, MEMORY REFERENCE, STACK INSTRUCTIONS, AND

INTERRUPT INSTRUCTIONS, A MORE DETAILED DESCRIPTION OF THE COMPUTER

ACCOMPANIES THIS PROJECT AND A DETAILED DESCRIPTION OF THE LANGUAGE

CAN BE FOUND IN THE DATA GENERAL MICRONOVA USER MANUAL.

INPUT REQUIREMENTS: ON EACH DATA CARD

COL 1-4 CONTAIN THE STARTING LOCATION TO LOAD CODE

COL 5-8 CONTAIN THE MAXIMUM NUMBER OF WORDS TO BE LOADED

(A MAX OF 18)

IF A RECORD HAS ZEROS IN FIELDS ONE AND TWO IT IS ASSUMED TO BE THE
END OF THE DATA WITH THE STARTING LOCATION IN THE NEXT F(4,0) FIELD

PROCEDURES USED:

CPU — A MAIN PROCEDURE THAT CONTROLS THE SIMULATOR

INSTRUCTION FETCH IS DONE HERE.

FUNCTION THAT CONVERTS CHAR(1) HEX INPUT TO

A_BIT(4)_BINARY

FUNCTION THAT CONVERTS A BIT(4) BINARY

VALUE TO A CHAR(1) HEX VALUE

LOAD — A PROCEDURE TO INITIALIZE THE SIMULATOR AND

LOAD MEMORY

TRACE_DUMP — A PROCEDURE TO EXECUTE A TRACE INSTRUCTION

DUMP — PRINTS A MEMORY DUMP

TRACE — PRINTS A TRACE

EXC — A PROCEDURE TO DECODE AND EXECUTE INSTRUCTIONS

APPENDIX Q — EXAMPLE OF JOB-GENERATED SIMULATOR PROGRAM
<table>
<thead>
<tr>
<th>CARD</th>
<th>EFFECTIVE</th>
<th>A PROCEDURE TO COMPUTE AN EFFECTIVE ADDRESS</th>
<th>RWR 520</th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>SHIFT</td>
<td>A PROCEDURE TO PERFORM THE SPECIFIED SHIFT</td>
<td>RWR 540</td>
</tr>
<tr>
<td>59</td>
<td>OPERATION</td>
<td>RWR 560</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>CARRY</td>
<td>A PROCEDURE TO INITIALIZE THE CARRY BIT</td>
<td>RWR 570</td>
</tr>
<tr>
<td>61</td>
<td></td>
<td>RWR 580</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td></td>
<td>RWR 590</td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>SKIP</td>
<td>A PROCEDURE TO PERFORM THE SPECIFIED SKIP OPERATION</td>
<td>RWR 610</td>
</tr>
<tr>
<td>64</td>
<td>IND</td>
<td>A PROCEDURE TO COMPUTE THE INDIRECT ADDRESS</td>
<td>RWR 630</td>
</tr>
<tr>
<td>65</td>
<td></td>
<td>RWR 640</td>
<td></td>
</tr>
<tr>
<td>66</td>
<td></td>
<td>RWR 650</td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>/*</td>
<td>RWR 660</td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>1/*</td>
<td>RWR 670</td>
<td></td>
</tr>
<tr>
<td>69</td>
<td></td>
<td>RWR 680</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td></td>
<td>RWR 690</td>
<td></td>
</tr>
<tr>
<td>71</td>
<td></td>
<td>RWR 700</td>
<td></td>
</tr>
<tr>
<td>72</td>
<td></td>
<td>RWR 710</td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>SYMBOL</td>
<td>DIMENSION DESCRIPTION</td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>AA</td>
<td>4,16 ACCUMULATORS</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>BB</td>
<td>3,24,16 DEVICE BUFFERS</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>MM</td>
<td>32768,16 MEMORY</td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>A</td>
<td>ACTIVE TRACE LOCATION</td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>B</td>
<td>SIMULATOR CONTROL</td>
<td></td>
</tr>
<tr>
<td>79</td>
<td></td>
<td>RWR 740</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td></td>
<td>RWR 750</td>
<td></td>
</tr>
<tr>
<td>81</td>
<td></td>
<td>RWR 760</td>
<td></td>
</tr>
<tr>
<td>82</td>
<td></td>
<td>RWR 770</td>
<td></td>
</tr>
<tr>
<td>83</td>
<td></td>
<td>RWR 780</td>
<td></td>
</tr>
<tr>
<td>84</td>
<td>C</td>
<td>CARRY BIT</td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>D</td>
<td>INTERRUPT FLAGS</td>
<td></td>
</tr>
<tr>
<td>86</td>
<td></td>
<td>RWR 800</td>
<td></td>
</tr>
<tr>
<td>87</td>
<td></td>
<td>RWR 810</td>
<td></td>
</tr>
<tr>
<td>88</td>
<td></td>
<td>RWR 820</td>
<td></td>
</tr>
<tr>
<td>89</td>
<td></td>
<td>RWR 830</td>
<td></td>
</tr>
<tr>
<td>90</td>
<td></td>
<td>RWR 840</td>
<td></td>
</tr>
<tr>
<td>91</td>
<td></td>
<td>RWR 850</td>
<td></td>
</tr>
<tr>
<td>92</td>
<td></td>
<td>RWR 860</td>
<td></td>
</tr>
<tr>
<td>93</td>
<td></td>
<td>RWR 870</td>
<td></td>
</tr>
<tr>
<td>94</td>
<td></td>
<td>RWR 880</td>
<td></td>
</tr>
<tr>
<td>95</td>
<td></td>
<td>RWR 890</td>
<td></td>
</tr>
<tr>
<td>96</td>
<td>E</td>
<td>SCALAR EFFECTIVE ADDRESS</td>
<td></td>
</tr>
<tr>
<td>97</td>
<td>F</td>
<td>FRAME POINTER</td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>G</td>
<td>STACK POINTER</td>
<td></td>
</tr>
<tr>
<td>99</td>
<td>H</td>
<td>STATLS</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td>RWR 900</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td></td>
<td>RWR 910</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
<td>RWR 920</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>I</td>
<td>16 INSTRUCTION REGISTER</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>J</td>
<td>SCALAR INDIPS FOR CYCLE TABLE TCYCLE</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>K</td>
<td>16 INTERMEDIATE REGISTER</td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>L</td>
<td>16 RIGHT EXTENSION FOR K</td>
<td></td>
</tr>
<tr>
<td>107</td>
<td>M</td>
<td>16 PRIORITY MASK</td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>N</td>
<td>16 PRIORITY MASK</td>
<td></td>
</tr>
<tr>
<td>CARD</td>
<td>P 16</td>
<td>LOCATION COUNTER</td>
<td>RWR 1060</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>------------------</td>
<td>--------</td>
</tr>
<tr>
<td>110</td>
<td>D 16</td>
<td>INTERMEDIATE ADDRESS</td>
<td>RWR 1070</td>
</tr>
<tr>
<td>111</td>
<td>R 64</td>
<td>INTERRUPT REQUEST BY DEVICE</td>
<td>RWR 1080</td>
</tr>
<tr>
<td>113</td>
<td>S 17</td>
<td>SHIFTER</td>
<td>RWR 1090</td>
</tr>
<tr>
<td>114</td>
<td>T 64</td>
<td>INTERMEDIATE-RESULTS-OR-TEMPORARY-STORAGE</td>
<td>RWR 1100</td>
</tr>
<tr>
<td>115</td>
<td>J 64</td>
<td>BUSY FLAGS</td>
<td>RWR 1110</td>
</tr>
<tr>
<td>116</td>
<td>V 64</td>
<td>CONE FLAGS</td>
<td>RWR 1120</td>
</tr>
<tr>
<td>117</td>
<td>X 64</td>
<td>DEVICE CODES ON</td>
<td>RWR 1130</td>
</tr>
<tr>
<td>118</td>
<td>Y 64</td>
<td>DEVICE PRESENT MASK</td>
<td>RWR 1140</td>
</tr>
<tr>
<td>119</td>
<td>CT 2</td>
<td>RESET VALUE, &quot;RANDOM RESET VALUE&quot;</td>
<td>RWR 1150</td>
</tr>
<tr>
<td>120</td>
<td>CYC 4</td>
<td>INDIRECT ACCESS COUNT</td>
<td>RWR 1160</td>
</tr>
<tr>
<td>121</td>
<td>DEL 2</td>
<td>PRIMARY, SECONDARY DELAYS</td>
<td>RWR 1170</td>
</tr>
<tr>
<td>122</td>
<td>KEY 5</td>
<td>CONSOLE (HAND HELD) FUNCTION II</td>
<td>RWR 1180</td>
</tr>
<tr>
<td>123</td>
<td>LOOP 5</td>
<td>COUNTER FOR SAV, RET INSTRUCTIONS</td>
<td>RWR 1190</td>
</tr>
<tr>
<td>124</td>
<td>SCALAR</td>
<td>COUNTER FOR CURRENT INSTRUCTION, REAL TIME</td>
<td>RWR 1200</td>
</tr>
<tr>
<td>125</td>
<td>SIZ 5</td>
<td>CLOCK CURRENT VALUE</td>
<td>RWR 1210</td>
</tr>
<tr>
<td>126</td>
<td>BT 2</td>
<td>PRIMARY, SECONDARY DELAYS</td>
<td>RWR 1220</td>
</tr>
<tr>
<td>127</td>
<td>SCALAR</td>
<td>CONSOLE (HAND HELD) FUNCTION II</td>
<td>RWR 1230</td>
</tr>
<tr>
<td>128</td>
<td>SCALAR</td>
<td>COUNTER FOR SAV, RET INSTRUCTIONS</td>
<td>RWR 1240</td>
</tr>
<tr>
<td>129</td>
<td>SCALAR</td>
<td>COUNTER FOR CURRENT INSTRUCTION, REAL TIME</td>
<td>RWR 1250</td>
</tr>
<tr>
<td>130</td>
<td>SCALAR</td>
<td>CLOCK CURRENT VALUE</td>
<td>RWR 1260</td>
</tr>
<tr>
<td>131</td>
<td>SCALAR</td>
<td>MACHINE RUN STATE</td>
<td>RWR 1270</td>
</tr>
<tr>
<td>132</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1280</td>
</tr>
<tr>
<td>133</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1290</td>
</tr>
<tr>
<td>134</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1300</td>
</tr>
<tr>
<td>135</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1310</td>
</tr>
<tr>
<td>136</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1320</td>
</tr>
<tr>
<td>137</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1330</td>
</tr>
<tr>
<td>138</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1340</td>
</tr>
<tr>
<td>139</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1350</td>
</tr>
<tr>
<td>140</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1360</td>
</tr>
<tr>
<td>141</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1370</td>
</tr>
<tr>
<td>142</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1380</td>
</tr>
<tr>
<td>143</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1390</td>
</tr>
<tr>
<td>144</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1400</td>
</tr>
<tr>
<td>145</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1410</td>
</tr>
<tr>
<td>146</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1420</td>
</tr>
<tr>
<td>147</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1430</td>
</tr>
<tr>
<td>148</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1440</td>
</tr>
<tr>
<td>149</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1450</td>
</tr>
<tr>
<td>150</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1460</td>
</tr>
<tr>
<td>151</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1470</td>
</tr>
<tr>
<td>152</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1480</td>
</tr>
<tr>
<td>153</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1490</td>
</tr>
<tr>
<td>154</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1500</td>
</tr>
<tr>
<td>155</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1510</td>
</tr>
<tr>
<td>156</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1520</td>
</tr>
<tr>
<td>157</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1530</td>
</tr>
<tr>
<td>158</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1540</td>
</tr>
<tr>
<td>159</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1550</td>
</tr>
<tr>
<td>160</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1560</td>
</tr>
<tr>
<td>161</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1570</td>
</tr>
<tr>
<td>162</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1580</td>
</tr>
<tr>
<td>163</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1590</td>
</tr>
<tr>
<td>164</td>
<td>SCALAR</td>
<td>INSTRUCTIONS</td>
<td>RWR 1600</td>
</tr>
<tr>
<td>CARO</td>
<td>163</td>
<td>RUN FIXED BINARY 'INITIAL' (1);</td>
<td>WR 1480</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>---------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>164</td>
<td>0</td>
<td>/* INITIALIZE THE SIMULATOR */</td>
<td>WR 1480</td>
</tr>
<tr>
<td>165</td>
<td>CALL LOAD;</td>
<td></td>
<td>WR 1620</td>
</tr>
<tr>
<td>166</td>
<td>0</td>
<td>/* RUN INTEGRATOR-LGCP */</td>
<td>WR 1620</td>
</tr>
<tr>
<td>167</td>
<td>CALL DUMP;</td>
<td></td>
<td>WR 1640</td>
</tr>
<tr>
<td>168</td>
<td>DO WHILE (RUN=0);</td>
<td></td>
<td>WR 1640</td>
</tr>
<tr>
<td>169</td>
<td>IF SUBSTR(D.0,1)=&quot;08&quot; THEN</td>
<td></td>
<td>WR 1660</td>
</tr>
<tr>
<td>170</td>
<td>0</td>
<td>/* DECODE AND UPDATE THE LOCATION COUNTER */</td>
<td>WR 1670</td>
</tr>
<tr>
<td>171</td>
<td>DO;</td>
<td></td>
<td>WR 1680</td>
</tr>
<tr>
<td>172</td>
<td>UNSPEC(T)=(&quot;08&quot;)</td>
<td></td>
<td>WR 1680</td>
</tr>
<tr>
<td>173</td>
<td>T=T+1;</td>
<td></td>
<td>WR 1700</td>
</tr>
<tr>
<td>174</td>
<td>RETURN;</td>
<td></td>
<td>WR 1710</td>
</tr>
<tr>
<td>175</td>
<td>0</td>
<td>/* CHECK FOR MEMORY OVERFLOW */</td>
<td>WR 1720</td>
</tr>
<tr>
<td>176</td>
<td>IF T&lt;MSIZ THEN P=SUBSTR(K,2,15);</td>
<td></td>
<td>WR 1730</td>
</tr>
<tr>
<td>177</td>
<td>ELSE IF MSIZ=32768 THEN</td>
<td></td>
<td>WR 1740</td>
</tr>
<tr>
<td>178</td>
<td>DO;</td>
<td></td>
<td>WR 1750</td>
</tr>
<tr>
<td>179</td>
<td>P=&quot;&quot;</td>
<td></td>
<td>WR 1760</td>
</tr>
<tr>
<td>180</td>
<td>PUT LIST ('ERROR 1');</td>
<td></td>
<td>WR 1770</td>
</tr>
<tr>
<td>181</td>
<td>CALL DUMP;</td>
<td></td>
<td>WR 1780</td>
</tr>
<tr>
<td>182</td>
<td>END;</td>
<td></td>
<td>WR 1790</td>
</tr>
<tr>
<td>183</td>
<td>ELSE DO;</td>
<td></td>
<td>WR 1800</td>
</tr>
<tr>
<td>184</td>
<td>P=&quot;1111111111111111&quot;</td>
<td></td>
<td>WR 1810</td>
</tr>
<tr>
<td>185</td>
<td>PUT LIST ('ERROR');</td>
<td></td>
<td>WR 1820</td>
</tr>
<tr>
<td>186</td>
<td>CALL DUMP;</td>
<td></td>
<td>WR 1830</td>
</tr>
<tr>
<td>187</td>
<td>END;</td>
<td></td>
<td>WR 1840</td>
</tr>
<tr>
<td>188</td>
<td></td>
<td></td>
<td>WR 1850</td>
</tr>
<tr>
<td>189</td>
<td>0</td>
<td>/* FETCH THE NEXT INSTRUCTION */</td>
<td>WR 1860</td>
</tr>
<tr>
<td>190</td>
<td>CALL MAC (&quot;08&quot;&quot;);</td>
<td></td>
<td>WR 1870</td>
</tr>
<tr>
<td>191</td>
<td>I=KI;</td>
<td></td>
<td>WR 1880</td>
</tr>
<tr>
<td>192</td>
<td>0</td>
<td>/* IF NECESSARY SET THE PRIMARY DELAY FLAGS FOR MSKO */</td>
<td>WR 1890</td>
</tr>
<tr>
<td>193</td>
<td>IF SUBSTR(D.0,1)=&quot;1&quot; THEN SUBSTR(D.10.1)=&quot;1&quot;;</td>
<td></td>
<td>WR 1900</td>
</tr>
<tr>
<td>194</td>
<td>SUBSTR(D.7,2)=&quot;0&quot;;</td>
<td></td>
<td>WR 1910</td>
</tr>
<tr>
<td>195</td>
<td>0</td>
<td>/* INCREMENT THE CYCLE COUNT */</td>
<td>WR 1920</td>
</tr>
<tr>
<td>196</td>
<td>SCYC(1)=SCYC(1)+SCYC(2);</td>
<td></td>
<td>WR 1930</td>
</tr>
<tr>
<td>197</td>
<td>0</td>
<td>/* CALL EXC TO EXECUTE THE INSTRUCTION */</td>
<td>WR 1940</td>
</tr>
<tr>
<td>198</td>
<td>CALL EXC;</td>
<td></td>
<td>WR 1950</td>
</tr>
<tr>
<td>199</td>
<td>0</td>
<td>/* CHECK FOR A TRACE REQUEST */</td>
<td>WR 1960</td>
</tr>
<tr>
<td>200</td>
<td>IF SUBSTR(D.5,1)=&quot;1&quot; THEN</td>
<td></td>
<td>WR 1970</td>
</tr>
<tr>
<td>201</td>
<td>DO;</td>
<td></td>
<td>WR 1980</td>
</tr>
<tr>
<td>202</td>
<td>CALL TRACE;</td>
<td></td>
<td>WR 1990</td>
</tr>
<tr>
<td>203</td>
<td>TCOUNT=TCOUNT-1;</td>
<td></td>
<td>WR 2000</td>
</tr>
<tr>
<td>204</td>
<td>IF TCOUNT=0 THEN SUBSTR(D.5,1)=&quot;0&quot;;</td>
<td></td>
<td>WR 2010</td>
</tr>
<tr>
<td>205</td>
<td>END;</td>
<td></td>
<td>WR 2020</td>
</tr>
<tr>
<td>206</td>
<td>0</td>
<td>/* INCREMENT THE CYCLE COUNT BY THE CYCLES NEEDED TO EXECUTE THE */</td>
<td>WR 2030</td>
</tr>
<tr>
<td>207</td>
<td>INSTRUCTION */</td>
<td></td>
<td>WR 2040</td>
</tr>
<tr>
<td>208</td>
<td>SCYC(1)=SCYC(1)+SCYC(2);</td>
<td></td>
<td>WR 2050</td>
</tr>
<tr>
<td>209</td>
<td>/* CHECK FOR CYCLE LIMIT EXCEEDED */</td>
<td></td>
<td>WR 2060</td>
</tr>
<tr>
<td>210</td>
<td>IF SCYC(1)&gt;SCYC(0) THEN DO;</td>
<td></td>
<td>WR 2070</td>
</tr>
<tr>
<td>211</td>
<td>PUT 5KIF LIST ('ERROR -- CYCLE LIMIT EXCEEDED');</td>
<td></td>
<td>WR 2080</td>
</tr>
<tr>
<td>212</td>
<td>RUN=0;</td>
<td></td>
<td>WR 2090</td>
</tr>
<tr>
<td>213</td>
<td>ENDDO;</td>
<td></td>
<td>WR 2100</td>
</tr>
<tr>
<td>214</td>
<td>0</td>
<td>/* CHECK AND RESET THE PRIMARY DELAY FLAGS */</td>
<td>WR 2110</td>
</tr>
<tr>
<td>215</td>
<td>IF SUBSTR(D.10.1)=&quot;0&quot; THEN</td>
<td></td>
<td>WR 2120</td>
</tr>
<tr>
<td>216</td>
<td>DO;</td>
<td></td>
<td>WR 2130</td>
</tr>
</tbody>
</table>
80/80 LIST

0000000011111111111122222222233333333344444445555555555555555555
1234567890123456789012345678901234567890123456789012345678901234567890

CARD

217 SUBSTR(D,9,2)='00';  RWR 2140
218 M=N;  RWR 2150
219 END;  RWR 2160
220 0 /* REinitialize the priority mask */  RWR 2170
221 IF SUBSTR(D,7,1)='1'B THEN  RWR 2180
222 DO;
223 T=MOD((1+T),32768);  RWR 2210
224 K=UNSPEC(T);  RWR 2220
225 SUBSTR(D,7,1)='0'B;  RWR 2230
226 0 /* CHECK FOR MEMORY OVERflow */  RWR 2240
227 IF MSIZ > T THEN P=SUBSTR(K,2,15);  RWR 2250
228 ELSE IF MSIZ=32768 THEN  RWR 2260
229 DO;
230 P='0';  RWR 2270
231 PUT LIST (*ERROR 3');  RWR 2280
232 CALL DUMP;  RWR 2290
233 END;
234 ELSE DO;
235 P='1';  RWR 2300
236 PUT LIST (*ERROR 4');  RWR 2310
237 CALL DUMP;  RWR 2320
238 END;
239 IF SUBSTR(D,2,1)='1'B THEN SUBSTR(D,2,1)='0'B;  RWR 2330
240 0 /* CHECK FOR A REAL TIME CLOCK INTERRUPT */  RWR 2340
241 IF SUBSTR(D,11,1) THEN DO;
242 SCYC(3)=SCYC(3)-SCYC(2);  RWR 2410
243 IF SCYC(3)<0 THEN DO;
244 SUBSTR(D,5,1)='1'B;  RWR 2420
245 SCYC(3)=1200;  RWR 2430
246 END;
247 END;
248 0 /* CHECK FOR AN INTERRUPT ENABLE */  RWR 2440
249 ELSE IF SUBSTR(C,1,1)==SUBSTR(D,4,1)||SUBSTR(D,5,1)  RWR 2450
250 THEN DO;
251 SUBSTR(C,1,1)='0'B;  RWR 2460
252 SUBSTR(D,11,1)='0'B;  RWR 2500
253 SUBSTR(D,3,1)='0'B;  RWR 2510
254 UNSPEC(J)='0'B;  RWR 2520
255 J=1;  RWR 2530
256 P=SUBSTR(UNSPEC(J),2,15);  RWR 2540
257 CALL MAC ('0'B,'5', '0'B);  RWR 2550
258 E=5-INDEX(SUBSTR(D,4,3),'1'B);  RWR 2560
259 0 CALL INC(E);  RWR 2570
260 0 /* CHECK FOR MEMORY OVERflow */  RWR 2580
261 IF MSIZ > E THEN P=SUBSTR(K,2,15);  RWR 2590
262 ELSE IF MSIZ=32768 THEN  RWR 2600
263 DO;
264 P='0';  RWR 2610
265 PUT LIST (*ERROR 5');  RWR 2620
266
80/80 LIST

000000000111111112222222333333334444444455555555566666666667777777778
1234567890123456789012345678901234567890123456789012345678901234567890

CARD
271 CALL DUMP;
272 END;
273 ELSE DO:
274 PUT LIST ("ERROR 5")
275 CALL DUMP;
276 END;
277
278 SUBSTR(D.A.+1)="1"B;
279 END;
280 1 TRACE DUMP:PROCEDURE:
281 0 /* THIS PROCEDURE EXECUTES A TRACE/DUMP INSTRUCTION */
282 DC AC FIXED BINARY:
283 DCL COUNT FIXED BINARY;
284 UNSPEC(AC)=0000000000000000||SUBSTR(I,.2);
285 UNSPEC(COUNT)=0000000000||SUBSTR(I,6,5);
286 0 /* IF TRACE REQUEST IS NOT ON THE CHECK THE COUNTER, IF IT IS */
287 / THEN RETURN, ELSE ASSIGN LOCATION COUNTER TO ACTIVE TRACE LOCATION, */
288 THEN TURN ON THE TRACE REQUEST, INITIALIZE THE TRACE COUNT, AND RETURN */
289 IF AC=0 THEN
290 IF SUBSTR(R.S,1)="0"B THEN
291 IF COUNT=0 THEN RETURN;
292 ELSE DO:
293 A="P";
294 SUBSTR(B.S,1)="1"B;
295 T=COUNT;
296 RETURN;
297 END;
298 ELSE IF A="P" THEN RETURN;
299 END:
300 ELSE DO:
301 COUNT=COUNT-1;
302 SUBSTR(I,6,5)=UNSPEC(MOD(COUNT,31));
303 CALL A C ("0"B||P, S, 1);
304 IF COUNT=0 THEN
305 A="A"B;
306 END:
307 SUBSTR(B,S,1)="0"B;
308 END;
309 RETURN;
310 END;
311 0 /* IF THE ACUMULATCR=1 THEN SET THE ACTIVE TRACE LOCATION TO 0 AND */
312 TURN OFF THE TRACE REQUEST */
313 ELSE IF AC=1 THEN
314 DO:
315 A="0"B;
316 SUBSTR(B.S,1)="0"B;
317 RETURN;
318 END;
319 0 /* IF THE ACUMULATCR=2 THEN PERFORM A DUMP */
320 ELSE IF AC=2 THEN
321 DO:
322 CALL DUMP;
323 RETURN;
324 END;
**80/80 LIST**

```
00000000111111111112222222222333333334444444444445555555555555555555666666666677777777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890
CARO
012345678901234567890123456789012345678901234567890123456789012345678901234567890
325 0 IF ACCUMULATOR=3 THEN DUMP AND SET THE RUN CODE TO 0 */
326 ELSE IF AC=3 THEN                      RWR 3220
327 DO;
328 CALL DUMP;
329 RETURN;
330 END DUMP:
331 IF AC=3 THEN                      RWR 3240
332 CALL DUMP:
333 RETURN;
334 END DUMP:
335 IF AC=3 THEN                      RWR 3260
336 CALL DUMP:
337 RETURN;
338 END DUMP:
339 IF AC=3 THEN                      RWR 3280
340 CALL DUMP:
341 RETURN;
342 END DUMP:
343 IF AC=3 THEN                      RWR 3300
344 CALL DUMP:
345 RETURN;
346 CALL DUMP:
347 RETURN;
348 CALL DUMP:
349 RETURN;
350 CALL DUMP:
351 RETURN;
352 CALL DUMP:
353 RETURN;
354 CALL DUMP:
355 RETURN;
356 CALL DUMP:
357 RETURN;
358 CALL DUMP:
359 RETURN;
360 CALL DUMP:
361 RETURN;
362 CALL DUMP:
363 RETURN;
364 CALL DUMP:
365 RETURN;
366 CALL DUMP:
367 END;
368 0 IF DUMP IS A PROCEDURE CALLED FROM TRACE_DUMP TO PRINT THE CONTENTS OF MEMORY */
369 RWR 3690
370 RWR 3690
371 RWR 3690
372 RWR 3690
373 RWR 3690
374 RWR 3690
375 RWR 3690
376 RWR 3710
377 RWR 3720
378 RWR 3730
379 RWR 3740
380 RWR 3750
```
80/80 LIST

000000001111111112222222223333334444444444455555555555555555
123456789012345678901234567890123456789012345678901234567890

CARD

333 333 333 333 333 333 333 333 333 333 333 333 333 333 333 333
444 444 444 444 444 444 444 444 444 444 444 444 444 444 444 444
555 555 555 555 555 555 555 555 555 555 555 555 555 555 555 555
666 666 666 666 666 666 666 666 666 666 666 666 666 666 666 666
777 777 777 777 777 777 777 777 777 777 777 777 777 777 777 777
888 888 888 888 888 888 888 888 888 888 888 888 888 888 888 888
999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999

433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459

MM(ADR)=CVH(SUBSTR(R,1,4))|CVH(SUBSTR(R,5,4))| CVH(SUBSTR(R,9,4))|CVH(SUBSTR(R,13,4));

K=R;

RETURN;

END MAC;

1

0

CVB:PROCEDURE (R) RETURNS (BIT(4));

/* CVB IS A PROCEDURE THAT CONVERTS A HEX NUMBER IN CHAR(1) FORMAT */

DCL R CHAR(*),B BIT(4),

HEX CHAR(16) INITIAL ('0123456789ABCDEF'),

DCL K BIT(16),J FIXED BINARY;

J=INDEX(HEX,R)-1;

K=UNSPEC(J);

B=SUBSTR(HEX,R+1,1);

RETURN (K);

END CVB;

1

DCL: PROCEDURE;

DCL (START,NUMBER,J) FIXED BINARY;

DCL (A1,A2) CHAR(4);

DCL (R) FIXED BINARY;

/* CVB CONVERTS A BIT(4) REPRESENTATION TO A CHAR(1) */

UNSPEC(R)='0000000C00000B'|B;

HEX CHAR(16) INITIAL ('0123456789ABCDEF');

RETURN (B);

END CVB;

1

LOAD: PROCEDURE;

DCL (START,NUMBER,J) FIXED BINARY;

DCL (A1,A2) CHAR(4);

DCL (R) FIXED BINARY;

/* LOAD SUBROUTINE SETS THE INITIAL CONDITIONS OF THE SIMULATOR */

SUBSTR(0,8,1)|='!';B;

U='0'1';

V='0'1';

DO J=0 TO 3;

AA(J)=='0'1';

END;

DO WHILE ('1'='9');

GET EDIT (A1,A2)(A4),A4);

UNSPEC(START)=CVB(SUBSTR(A1,1,1))|CVB(SUBSTR(A1+2,1,1));

CVB(SUBSTR(A1,1,1))|CVB(SUBSTR(A1+4,1,1));

UNSPEC(NUMBER)=CVB(SUBSTR(A2,1,1))|CVB(SUBSTR(A2+2,1,1));

CVB(SUBSTR(A2,1,1))|CVB(SUBSTR(A2+4,1,1));

0 /* CHECK FOR MEMORY "OVER LOAD" */

IF START NUMBER > #SZ THEN DO;

IF START=0 & (NUMBER=0) THEN DO;

GET EDIT (A1)(A1);

UNSPEC(START)=CVB(SUBSTR(A1,1,1))|CVB(SUBSTR(A1+2,1,1));

R=

RUN=0;

RETURN;

END;

/* CHECK FOR AN END OF PROGRAM */

IF (START=0) & (NUMBER=0) THEN DO;

GET EDIT (A1)(A1);

UNSPEC(START)=CVB(SUBSTR(A1,1,1))|CVB(SUBSTR(A1+2,1,1));

R=

RUN=0;

RETURN;

END;

/* CHECK FOR AN END OF PROGRAM */

IF (START=0) & (NUMBER=0) THEN DO;

GET EDIT (A1)(A1);

UNSPEC(START)=CVB(SUBSTR(A1,1,1))|CVB(SUBSTR(A1+2,1,1));

R=

RUN=0;

RETURN;

END;
80/80 LIST

00000000001111111111222222222222222333333134444444444445555555555555555566666666666666777777777
1234567890123456799012345678901234567990123456789012345678901234567890123456789012345678901234567890

CARD
487 CVG(SUBSTR(#1,3,1))(CVB(SUBSTR(#1,4,1))); RWR 4840
488 P=SUBSTR(UNSPEC(START)),2,15); RWR 4850
489 RETURN; RWR 4860
490 END; RWR 4870
491 DO JJ=START TO (NUMBER + START - 1); RWR 4880
492 GET EDIT((M),(J )); (A(4)); RWR 4890
493 END; RWR 4900
494 IF NUMBER >= 1E THEN GET_SKIP; RWR 4910
495 END; RWR 4920
496 BOTTOM: END LOAD;
497 PROCEDURE;
498 DCL CODE FIXED BINARY;
499 DCL AC FIXED BINARY;
500 DCL (AC, ACS) FIXED BINARY;
501 DCL (KK, KK1, KK2) FIXED BINARY (31, 0).J1 FIXED BINARY INITIAL (0).
502 K1 BIT(16);
503 /* AC IS THE DESTINATION ACCUMULATOR */
504 /* ACS IS THE ADDRES ACCUMULATOR */
505 /*
506 */ THIS PROCEDURE PERFORMS INSTRUCTION EXECUTION */
507 PUT SKIP (2);
508 PUT SKIP DATA (P);
509 PUT SKIP DATA (I);
510 J=0;
511 UNSPEC(J)="0'B"|P;
512 IF MM(J)=0000 THEN DO;
513 RUN=0;
514 RETURN;
515 END;
516 UNSPEC(J)="0000000000"|SUBSTR(I,11,6);
517 IF J=62 THEN DO;
518 CALL TRACEDUMP;
519 RETURN;
520 END;
521 UNSPEC(ACD)="0000000000000000"|SUBSTR(I,4,2);
522 UNSPEC(ACS)="0000000000000000"|SUBSTR(I,2,2);
523 /* FIRST CHECK FOR A TWO ACCUMULATOR INSTRUCTION */
524 IF SUBSTR(I,1,1) THEN DO:
525 /* TWO ACCUMULATOR INSTRUCTIONS */
526 K9000000000000000|SUBSTR(I,11,6);
527 K=9000000000000000|SUBSTR(I,11,6);
528 UNSPEC(J)=K;
529 IF J=8 THEN DCI;
530 /* TRAP */
531 CALL MAC("00000000000101010'B",P);
532 E=39;
533 CALL INDE();
534 IF RUN0 THEN RETURN;
535 P=SUBSTR(UNSPEC(E)),2,15);
536 SUBSTR(D,8,1)="1'B";
537 SCYC(2)=SCYC(1);
538 RETURN;
539 END;
540 SCYC(2)=SCYC(1);
CALL CARRY;

IF JJ = 1 THEN CC;

NEGATE * /

K = AA(ACS);

IF SUBSTR(S, 1, 1) THEN SUBSTR(S, 1, 1) = ¬C;

ELSE SUESTR(S, 1, 1) = C;

END;

ELSE IF JJ = 4 THEN DO;

ADD CUMPLEMENT * /

K = AA(ACS);

IF SUBSTR(S, 1, 1) THEN SUBSTR(S, 1, 1) = ¬C;

ELSE SUESTR(S, 1, 1) = C;

END;

ELSE IF JJ = 5 THEN DO;

SUBTRACT * /

S = SUBSTR(S, 1, 1);

IF SUBSTR(S, 1, 1) THEN SUBSTR(S, 1, 1) = ¬C;

ELSE SUESTR(S, 1, 1) = C;

END;

ELSE IF JJ = 6 THEN DO;

ADD * /

S = SUBSTR(S, 1, 1);

IF SUBSTR(S, 1, 1) THEN SUBSTR(S, 1, 1) = ¬C;

ELSE SUESTR(S, 1, 1) = C;

END;

END;

COMPLEMENT * /

S = 10B || ¬AA(ACS);

MOVE * /

S = C || AA(ACS);

INCREMENT * /

S = C || AA(ACS);

J = J + 1;

S = C || UNSPEC(J);

END;

AND * /

S = 0B || (AA(ACS) & AA(ACD));

CALL SHIFT;
CALL MAC(K)*F*,K;  RWF 6460
UNSPEC(J)=K;  RWF 6470
IF SUBSTR(1,5,1) THEN J=J-1;  RWF 6480
ELSE J=J+1;  RWF 6490
K=UNSPEC(J);  RWF 6500
CALL MAC(K)*S*,K;  RWF 6510
IF J=0 THEN DO;  RWF 6520
UNSPEC(J)="0'B"|F;  RWF 6530
J=J+1;  RWF 6540
P=SUBSTR(UNSPEC(J),2,15);  RWF 6550
SCYC(2)=SCYC(2)+STCYCL(0);  RWF 6560
END;  RWF 6570
END;  RWF 6580
RETURN;  RWF 6590
END;  RWF 6600
ELSE DO;  RWF 6610
CHECK FOR A STACK INSTRUCTION */
UNSPEC(J)="000000000'B"|SUBSTR(1,1,6);  RWF 6620
IF J=1 THEN DO;  RWF 6630
UNSPEC(J)="0000000000'B"|SUBSTR(1,6,3);  RWF 6640
UNSPEC(AC)="0000000000000'B"|SUBSTR(1,4,2);  RWF 6650
IF J=0 THEN DC;  RWF 6660
/* MOVE TO OR FROM FRAME POINTER */
SCYC(2)=STCYCL(1)+SCYC(2);  RWF 6670
IF SUBSTR(1,9,1) THEN AA(AC)="0'B"|F;  RWF 6680
ELSE F=SUBSTR(AC(AC),2,15);  RWF 6690
RETURN;  RWF 6700
END;  RWF 6710
IF J=2 THEN DC;  RWF 6720
/* MOVE TO OR FROM STACK POINTER */
SCYC(2)=SCYC(2)+STCYCL(1);  RWF 6730
IF SUBSTR(1,9,1) THEN AA(AC)="0'B"|G;  RWF 6740
ELSE G=SUBSTR(AA(AC),2,15);  RWF 6750
RETURN;  RWF 6760
END;  RWF 6770
IF J=3 THEN DC;  RWF 6780
/* PUSH OR POP THE ACCUMULATOR */
SCYC(2)=SCYC(2)+STCYCL(3);  RWF 6790
IF SUBSTR(1,9,1) THEN DO;  RWF 6800
CALL MAC("0'B"|G)*F*,AA(AC));  RWF 6810
UNSPEC(J)="0'B"|G;  RWF 6820
J=J+1;  RWF 6830
G=SUBSTR((UNSPEC(J)),2,15);  RWF 6840
RETURN;  RWF 6850
END;  RWF 6860
ELSE DO;  RWF 6870
UNSPEC(J)="0'B"|G;  RWF 6880
J=J+1;  RWF 6890
G=SUBSTR((UNSPEC(J)),2,15);  RWF 6900
CALL MAC("0'B"|G)*S*,AA(AC));  RWF 6910
IF MOD(J,256) = 0 THEN RETURN;  RWF 6920
SUBSTR(0,3,2)="11'B";  RWF 6930
ELSE DO;  RWF 6940
RETURN;  RWF 6950
END;  RWF 6960
ELSE DO;  RWF 6970
RETURN;  RWF 6980
END;  RWF 6990
RETURN;  RWF 7000
END;  RWF 7010
RETURN;  RWF 7020
80/80 LIST

CARD
757  KK2=J;
758  KK=KK1=KK2=85555;
759  AA(0)=SUBSTR((UNSPECKK1),1,16);
760  AA(1)=SUBSTR((UNSPECKK1),17,16);
761  SCYC(2)=SCYC(2)+STCYCL(8);
762  RETURN;
763  END;
764  ELSE DO;
765  UNSPEC(J)=AA(0);
766 KK1=J;
767  UNSPEC(J)=AA(2);
768  KK2=J;
769  IF KK1<KK2 THEN DO;
770  KK=KK1=KK2;
771  AA(1)=SUBSTR((UNSPECKK1),17,16);
772  KK=KK1=KK2;
773  AA(0)=SUBSTR((UNSPECKK1),17,16);
774  C=0'B;
775  SCYC(2)=SCYC(2)+STCYCL(9);
776  RETURN;
777  END;
778  ELSE DO;
779  SUBSTR(H,3,1)=1'B;
780  C=1'E;
781  RETURN;
782  END;
783  END;
784  ELSE PUT_SKIP_LIST('OP_CODE_ERROR_IN_STACK_INSTR');
785  END;
786  /* CHECK FOR AN INTERRUPT INSTRUCTION */
787  unspec(J)=0000000000'B|SUBSTR(I,1,16);
788  IF J=6 THEN DO;
789  SCYC(2)=SCYC(2)+STCYCL(1);
790  UNSPEC(J)=0000000000'B|SUBSTR(I,1,6,3);
791  IF J=0 THEN DC;
792  /* INTERRUPT ENABLE OR DISABLE */
793  IF (SUBSTR(I,1,1)=0'B)|(SUBSTR(I,10,1)=1'B) THEN
794  SUBSTR(C,2,1)=1'B;
795  END;
796  ELSE IF J=2 THEN DO;
797  /* RESET BUSY 4C CCEN FLAGS */
798  IF SUBSTR(I,4,2)=00'B THEN DO;
799  v='7B;
800  u='0';
801  n='0';
802  END;
803  SUBSTR(C,5,1)=SUBSTR(I,4,1);
804  END;
805  ELSE IF J=4 THEN DC;
806  /* REAL TIME CLOCK ENABLE OR DISABLE */
807  SUBSTR(D,1,1)=1';
808  UNSPEC(J)=0000000000'B|SUBSTR(I,4,2);
809  N=AA(J);
810  END;
<table>
<thead>
<tr>
<th>CARD</th>
<th>CALL TND(E);</th>
<th>RWR 8626</th>
</tr>
</thead>
<tbody>
<tr>
<td>866</td>
<td>SUBSTR(1,5,1)=&quot;0&quot;B;</td>
<td>RWR 8630</td>
</tr>
<tr>
<td>867</td>
<td>DO LOOP=E TO E+16;</td>
<td>RWR 8640</td>
</tr>
<tr>
<td>868</td>
<td>PUT SKIP LIST (WW(LOOP));</td>
<td>RWR 8650</td>
</tr>
<tr>
<td>869</td>
<td>END;</td>
<td>RWR 8660</td>
</tr>
<tr>
<td>870</td>
<td>ELSE PUT SKIP LIST (<em>INVALID DEVICE CODE FOR DATA OUT</em>);</td>
<td>RWR 8680</td>
</tr>
<tr>
<td>871</td>
<td>CALL BUSY_DONE;</td>
<td>RWR 8700</td>
</tr>
<tr>
<td>872</td>
<td>END;</td>
<td>RWR 8720</td>
</tr>
<tr>
<td>873</td>
<td>IOSKIP: PROCEDURE;</td>
<td>RWR 8740</td>
</tr>
<tr>
<td>874</td>
<td>IF (J=0) &amp; (SUBSTR(U,DCODE,1)=&quot;8&quot;B) THEN SUBSTR(D,8,1)=&quot;1&quot;B;</td>
<td>RWR 8750</td>
</tr>
<tr>
<td>875</td>
<td>ELSE IF (J=3) &amp; (SUBSTR(V,DCODE,1)=&quot;0&quot;B) THEN SUBSTR(D,8,1)=&quot;11&quot;B;</td>
<td>RWR 8760</td>
</tr>
<tr>
<td>876</td>
<td>ELSE IF (J=2) &amp; (SUBSTR(V,DCODE,1)=&quot;0&quot;B) THEN SUBSTR(D,8,1)=&quot;11&quot;B;</td>
<td>RWR 8770</td>
</tr>
<tr>
<td>877</td>
<td>ELSE RETURN;</td>
<td>RWR 8780</td>
</tr>
<tr>
<td>878</td>
<td>END;</td>
<td>RWR 8790</td>
</tr>
<tr>
<td>879</td>
<td>BUSY_DONE: PROCEDURE;</td>
<td>RWR 8800</td>
</tr>
<tr>
<td>880</td>
<td>UNSPEC(J)=&quot;00000000000000&quot;B</td>
<td>SUBSTR(1,9,2);</td>
</tr>
<tr>
<td>881</td>
<td>IF J=1 THEN DO:</td>
<td>RWR 8820</td>
</tr>
<tr>
<td>882</td>
<td>/* START DEVICE */</td>
<td>RWR 8830</td>
</tr>
<tr>
<td>883</td>
<td>SUBSTR(U,DCODE,1)=&quot;1&quot;B;</td>
<td>RWR 8840</td>
</tr>
<tr>
<td>884</td>
<td>SUBSTR(V,DCODE,1)=&quot;0&quot;B;</td>
<td>RWR 8850</td>
</tr>
<tr>
<td>885</td>
<td>END;</td>
<td>RWR 8860</td>
</tr>
<tr>
<td>886</td>
<td>ELSE IF (J=2 THEN DC;</td>
<td>RWR 8870</td>
</tr>
<tr>
<td>887</td>
<td>SUBSTR(U,DCODE,1)=&quot;0&quot;B;</td>
<td>RWR 8880</td>
</tr>
<tr>
<td>888</td>
<td>SUBSTR(V,DCODE,1)=&quot;1&quot;B;</td>
<td>RWR 8890</td>
</tr>
<tr>
<td>889</td>
<td>END;</td>
<td>RWR 8900</td>
</tr>
<tr>
<td>890</td>
<td>ELSE IF J=3 THEN DC;</td>
<td>RWR 8910</td>
</tr>
<tr>
<td>891</td>
<td>SUBSTR(U,DCODE,1)=&quot;1&quot;B;</td>
<td>RWR 8920</td>
</tr>
<tr>
<td>892</td>
<td>SUBSTR(V,DCODE,1)=&quot;0&quot;B;</td>
<td>RWR 8930</td>
</tr>
<tr>
<td>893</td>
<td>END;</td>
<td>RWR 8940</td>
</tr>
<tr>
<td>894</td>
<td>END_BUSY_DONE;</td>
<td>RWR 8950</td>
</tr>
<tr>
<td>895</td>
<td>END;</td>
<td>RWR 8960</td>
</tr>
<tr>
<td>896</td>
<td>1 SKIP: PROCEDURE;</td>
<td>RWR 8970</td>
</tr>
<tr>
<td>897</td>
<td>DCL (J,K,L) FIXED BINARY;</td>
<td>RWR 8980</td>
</tr>
<tr>
<td>898</td>
<td>UNSPEC(J)=&quot;00000000000000&quot;B</td>
<td>SUBSTR(1,14,3);</td>
</tr>
<tr>
<td>901</td>
<td>UNSPEC(K)=5;</td>
<td>RWR 9000</td>
</tr>
<tr>
<td>902</td>
<td>IF J=0 THEN RETURN;</td>
<td>RWR 9010</td>
</tr>
<tr>
<td>903</td>
<td>ELSE IF J=1 THEN SUBSTR(D,7,1)=&quot;1&quot;B;</td>
<td>RWR 9020</td>
</tr>
<tr>
<td>904</td>
<td>ELSE IF (J=2) &amp; (C=&quot;0&quot;B) THEN SUBSTR(D,7,1)=&quot;1&quot;B;</td>
<td>RWR 9030</td>
</tr>
<tr>
<td>905</td>
<td>ELSE IF (J=3) &amp; (C=&quot;0&quot;B) THEN SUBSTR(D,7,1)=&quot;1&quot;B;</td>
<td>RWR 9040</td>
</tr>
<tr>
<td>906</td>
<td>ELSE IF (J=4) &amp; (K=0) THEN SUBSTR(D,7,1)=&quot;1&quot;B;</td>
<td>RWR 9050</td>
</tr>
<tr>
<td>907</td>
<td>ELSE IF (J=5) &amp; (K=0) THEN SUBSTR(D,7,1)=&quot;1&quot;B;</td>
<td>RWR 9060</td>
</tr>
<tr>
<td>908</td>
<td>ELSE IF (J=6) &amp; (C=&quot;0&quot;B) THEN SUBSTR(D,7,1)=&quot;1&quot;B;</td>
<td>RWR 9070</td>
</tr>
<tr>
<td>909</td>
<td>ELSE IF (J=7) &amp; (C=&quot;0&quot;B) THEN SUBSTR(D,7,1)=&quot;1&quot;B;</td>
<td>RWR 9080</td>
</tr>
<tr>
<td>910</td>
<td>IF SUBSTR(D,7,1).THEN SCYC(2)=SCYC(2)+STCYCL(0);</td>
<td>RWR 9090</td>
</tr>
<tr>
<td>911</td>
<td>RETURN;</td>
<td>RWR 9100</td>
</tr>
<tr>
<td>912</td>
<td>END SKIP;</td>
<td>RWR 9110</td>
</tr>
<tr>
<td>913</td>
<td>END EFFECTIVE: PROCEDURE;</td>
<td>RWR 9120</td>
</tr>
<tr>
<td>914</td>
<td>DCL (J,K,L,N) FIXED BINARY;</td>
<td>RWR 9130</td>
</tr>
<tr>
<td>915</td>
<td>UNSPEC(J)=&quot;00000000000000&quot;B</td>
<td>SUBSTR(1,7,2);</td>
</tr>
<tr>
<td>916</td>
<td>UNSPEC(E)=&quot;00000000000000&quot;B</td>
<td>SUBSTR(1,9,8);</td>
</tr>
</tbody>
</table>
80/80 LIST

00000000111111112222223333334444444444555555555566666666677777777778
1234567890123456789012345678901234567890123456789012345678901234567890

CARD
973 IF CT>15 THEN CC; RWR 9700
974 RUN=0; RWR 9710
975 CALL TRACE; RWR 9720
976 PUT SKIP LIST ('CHAINING TO LONG IN IND ROUTINE'); RWR 9730
977 RETURN; RWR 9740
978 END; RWR 9750
979 UNSPEC(Q)='0'B || SUBSTR(A,2,15); RWR 9760
980 IF (16<0) & (31)=0) THEN DO; RWR 9770
981 IF Q< 23 THEN Q=Q+1; RWR 9780
982 ELSE Q=C+1; RWR 9790
983 B=UNSPEC(C); RWR 9800
984 CALL MAC((UNSPEC(E)),*S',B); RWR 9810
985 CT=CT+1; RWR 9820
986 IF CT > 15 THEN DO; RWR 9830
987 RUN=0; RWR 9840
988 CALL TRACE; RWR 9850
989 PUT SKIP LIST ('CHAINING TO LONG IN IND ROUTINE'); RWR 9860
990 RETURN; RWR 9870
991 END; RWR 9880
992 UNSPEC(E)='0'B || SUBSTR(A,2,15); RWR 9890
993 IF SUBSTR((UNSPEC(Q)),1,1)='0'B THEN RWR 9900
994 RETURN; RWR 9910
995 END; RWR 9920
996 ELSE DO; RWR 9930
997 UNSPEC(E)='0'B || SUBSTR(A,2,15); RWR 9940
998 IF SUBSTR(A,1,1)='0'B THEN RWR 9950
999 RETURN; RWR 9960
1000 END; RWR 9970
1001 END; RWR 9980
1002 END IND; RWR 9990
1003 0 END CPU; RWR 10000
1004 END CPU; RWR 10010
1005 0 END CPU; RWR 10020
1006 END CPU; RWR 10030
1007 END CPU; RWR 10040
1008 $DATA