Teaching for transfer of assembly language concepts to C programming using computer-based instruction and traditional instruction

Hsiu-Mei Lin

Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd

Part of the Communication Technology and New Media Commons, Engineering Education Commons, and the Instructional Media Design Commons

Recommended Citation

Lin, Hsiu-Mei, "Teaching for transfer of assembly language concepts to C programming using computer-based instruction and traditional instruction" (1993). Retrospective Theses and Dissertations. 10839.

https://lib.dr.iastate.edu/rtd/10839

This Dissertation is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
Teaching for transfer of assembly language concepts to C programming using computer-based instruction and traditional instruction

Lin, Hsiu-Mei, Ph.D.
Iowa State University, 1993
Teaching for transfer of assembly language concepts to C programming using computer-based instruction and traditional instruction

by

Hsiu-Mei Lin

A Dissertation Submitted to the Graduate Faculty in Partial Fulfillment of the Requirements for the Degree of DOCTOR OF PHILOSOPHY

Department: Industrial Education and Technology Major: Industrial Education and Technology

Approved: Members of the Committee:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Ames, Iowa

1993
TABLE OF CONTENTS

CHAPTER I. INTRODUCTION 1
   Statement of the Problem 4
   Purpose of the Study 4
   Questions of the Study 5
   Hypotheses of the Study 5
   Assumptions of the Study 7
   Limitations of the Study 8
   Definition of Terms 9

CHAPTER II. REVIEW OF RELATED LITERATURE 11
   Language Transfer 11
      Natural language transfer 11
      Programming language transfer 13
   Research on Learning to Program 15
      Methodology of programming instruction 15
      Assembly language in transition 18
   Computer-Based Instruction Development 20
      Unique features in computer-based instruction 20
      Learning theory 22
      Courseware design 26
   Summary of Literature Review 30

CHAPTER III. METHODOLOGY 33
   Subjects 33
   Description of the Computer-Presented Program 34
      Arithmetic Operations 40
      Branch Operations 41
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIBLIOGRAPHY</td>
<td>91</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>100</td>
</tr>
<tr>
<td>APPENDIX A. PRETEST</td>
<td>101</td>
</tr>
<tr>
<td>APPENDIX B. POSTTEST 1</td>
<td>111</td>
</tr>
<tr>
<td>APPENDIX C. POSTTEST 2 FOR IEDT 216</td>
<td>121</td>
</tr>
<tr>
<td>APPENDIX D. POSTTEST 2 FOR IEDT 316</td>
<td>127</td>
</tr>
<tr>
<td>APPENDIX E. MEANS AND STANDARD DEVIATIONS ON PRETEST</td>
<td>134</td>
</tr>
<tr>
<td>APPENDIX F. MEANS AND STANDARD DEVIATIONS ON POSTTEST 1</td>
<td>136</td>
</tr>
<tr>
<td>APPENDIX G. MEANS AND STANDARD DEVIATIONS ON POSTTEST 2 FOR IEDT 216</td>
<td>138</td>
</tr>
<tr>
<td>APPENDIX H. MEANS AND STANDARD DEVIATIONS ON POSTTEST 2 FOR IEDT 316</td>
<td>140</td>
</tr>
</tbody>
</table>
V

LIST OF TABLES

Table 1. An ANOVA Table on Pretest by Group 57
Table 2. An ANOVA Table on Pretest by Class 58
Table 3. Means and Standard Deviations for the Assembly Language Items on Posttest 1 60
Table 4. Means and Standard Deviations for the Total Items on Posttest 1 61
Table 5. One-way ANCOVA Table for the Assembly Language Items on Posttest 1 62
Table 6. Two-way ANCOVA Table for the Assembly Language Items on Posttest 2 62
Table 7. One-way ANCOVA Table for the Total Items on Posttest 1 63
Table 8. Two-way ANCOVA Table for the Total Items on Posttest 1 63
Table 9. Means and Standard Deviations on Posttest 1 (N = 49) 65
Table 10. One-way ANCOVA Table for IEDT 216 on Posttest 2 by the Covariates of Pretest and Posttest 1 65
Table 11. Two-way ANCOVA Table for IEDT 216 on Posttest 2 by the Covariates of Pretest and Posttest 1 66
Table 12. One-way ANCOVA Table for IEDT 316 on Posttest 2 by the Covariates of Pretest and Posttest 1 66
Table 13. Regression Analysis for IEDT 216 on Posttest 2 69
Table 14. Regression Analysis for IEDT 316 on Posttest 2 69
Table 15. The Means and Standard Deviations of Items 5, 7, and 10 on Posttest 1 (N = 49) 77
Table 16. The Means and Standard Deviation of Item 1 on Posttest 1 (N = 49) 78
Table 17. The Means and Standard Deviations of Items 14 and 35 on Posttest 1 (N= 49) 79

Table 18. The Means and Standard Deviations of Items 16, 19, 21, 28, 29, and 32 on Posttest 1 (N= 49) 82

Table 19. The Means and Standard Deviation of Item 46 on Posttest 1 (N = 49) 83
LIST OF FIGURES

Figure 1. The Main Topics Page 36
Figure 2. A Sample of CPU Architecture Description 36
Figure 3. The Page for Choosing the Fetch Button 38
Figure 4. The Screen before Fetch "MOV AL,86H" 38
Figure 5. The Screen after Fetch "MOV AL,86H" 39
Figure 6. After Clicking on the Check Button, the Answer is Displayed 39
Figure 7. The Screen for the Results of Program Execution includes a "Flag" Button 42
Figure 8. The First Page Regarding Flag Information 42
Figure 9. The Second Page Regarding Flag Information 43
Figure 10. An Example of Exercises 43
Figure 11. Layout of Memory to Meet JA Condition 45
Figure 12. Layout of Memory not to Meet JA Condition 45
Figure 13. Loop Operation Program 46
Figure 14. A Window to Give the Options for Looping 47
Figure 15. Subroutine Program 48
Figure 16. Layout of Program Interrupts 49
Figure 17. The Assignment of Two Groups 52
Figure 18. The Assignment of Subjects for Two-way ANCOVA in Posttest 1 53
Figure 19. The Assignment of Subjects for Two-way ANCOVA in Posttest 2 53
CHAPTER I. INTRODUCTION

With the growing popularity of computers, computer programming has become a highly marketable skill. However, the acquisition of that skill is very challenging for the learner. Consequently, an enormous volume of articles and studies have been developed to discuss how to facilitate student's abilities to learn programming beyond the traditional method. For example, Mayer (1981) argued that knowledge of what computers do and how they work may make learning to program more meaningful. Some studies have also been conducted to examine the learning of a programming language with computer-based models (e.g., Hooper & Thomas, 1990; Lanza & Roselli, 1989; Quong & Feghali, 1991). However, very little research has been constructed to examine computer-based models of a pure assembly language, such as Intel 8086 or Motorola 68000 families, and their impact on student learning and transferability to high-level language.

Assembly language is a low-level computer language. Even with today's powerful high-level languages, assembly language is still needed in the field of computer science. One of the reasons assembly language remains a necessity is that the knowledge of assembly language eases the transition into learning other languages. For example, assembly language has several programming constructs which are also found in high-level language. The "LOOP" construct performs a recurring
operation. The "CALL" construct provides the ability to create subroutines, and the counter, registers, and memory can be manipulated by programs. With these structures the learner can solve complex problems which demonstrate essential techniques of algorithm design. Another reason assembly language remains a necessity is that the learner can understand how the computer executes a program. The programmer, with assembly language background, can clearly understand how the computer is functioning (Abel, 1989) and transfer that language to high-level programming.

The purpose of teaching assembly language in a computer application course is to introduce students to abstract concepts before delving into programming details and to combine classroom lectures with laboratory experimentation. For a novice, it is difficult to understand and learn assembly language in a short amount of time. Hence, the use of a series of computer-based programs may be helpful. Based on this perspective, a computer-based assembly language program was designed to establish a meaningful introduction for students to programming.

The design of this program is rooted in the theories of learning and instruction. Based on an analysis of learning needs, this program is a tutorial rather than a drill and practice or a simulation. It combines learning with practice and provides the structure into which a basic simulation would be designed. This program also focuses on conceptual
representations as suggested by Mayer (1989). Mayer concluded that conceptual models play a crucial role in facilitating transfer.

Research on learning environments indicates that the learning environment needs to be designed similar to the environment in which what is learned must be performed. The environment must also be adjusted to learner knowledge and skill levels (Kulik, 1983; Montague, 1988). Computer-based instruction plays a significant role in the learning environment. It enhances the acquisition of knowledge and in some instances is superior to traditional instructional environments (Kulik, 1983). In addition, researchers have been exploring the role animation can play in the learning environment. Some researchers have contended that an animated display, such as a picture, a graphic, proper use of timing, and motion, may not only grasp the learner's attention, but also express many things compactly, vividly, and more rapidly understandable (Nievergelt, 1980; Rieber, 1990; Shu, 1988).

Based on the research stated above, a computer-based assembly language program is designed by using animation to display the virtual view of how a program is executed. The way in which this view is presented enables the learner to see concrete examples of dynamic programming concepts in action to provide a basis for a mental model. It may provide an easier and more interesting way for novices to learn assembly language, and may facilitate transfer to the C language.
Statement of the Problem

In recent years, computers have made their way into two major educational activities: programming as learned by students and computer-based instruction as employed by teachers. Some studies have been conducted in an effort to improve the environment in which the student learns to program with a high-level language by applying the cognitive approaches to programming. However, studies concerning the question of whether learning assembly language is helpful for learning a high-level language are still relatively rare. In addition, research about the usage of computer-based instruction to support student understanding of assembly language is still lacking. Thus, the problem to be addressed in this study is to explore the effects of learning assembly language, either by computer-based instruction or traditional learning, on the subsequent learning of the C language.

Purpose of the Study

The purpose of this study was twofold: (1) to evaluate the effectiveness of using computer-based instruction in learning assembly language at the undergraduate level, and (2) to investigate whether learning assembly language by two instructional methods (traditional lecture/demonstration/practice vs. computer-based instruction) was transferred to writing programs in the C language to the same degree.
Questions of the Study

The questions to be addressed by the study were as follows:

1. Can a subset of concepts related to programming in assembly language be learned through computer-based instruction as effectively, as measured by traditional test scores, as the learning gained through traditional classroom instruction?

2. Are the concepts learned in assembly language transferred to the C language to the same degree for students having learned assembly language using the computer-based program as those having learned through traditional lecture/practice methods?

Hypotheses of the Study

For the purposes of this study, and to facilitate analysis, the following hypotheses are proposed:

1. There is no significant difference between the pretest means of the experimental and control groups at the 95% confidence level. That is,

\[ H_0: \mu_{E,\text{pre}} = \mu_{C,\text{pre}} \]

\[ H_a: \mu_{E,\text{pre}} \neq \mu_{C,\text{pre}} \]

This hypothesis is included to assess the degree to which random placement in the experimental (learning assembly language with the computer-based program) and control
(learning assembly language with traditional instruction) groups created samples of subjects relatively equal in prior knowledge of computers, assembly language, and the C language.

2. There is no significant difference between the experimental and control group posttest 1 means, adjusted for effects of the pretest at the 95% confidence level. That is,

\[ H_0: \mu_{E,post1} = \mu_{C,post1} \]
\[ H_a: \mu_{E,post1} \neq \mu_{C,post1} \]

This hypothesis examines the differences among the treatment groups regarding their knowledge of assembly language concepts. The pretest scores are used to reduce the error variance term of the analysis by the degree to which the posttest 1 scores are correlated with the pretest scores.

3. There is no significant difference between the experimental and control group posttest 2 means, adjusted for effects of the pretest and posttest 1 scores at the 95% confidence level. That is,

\[ H_0: \mu_{E,post2} = \mu_{C,post2} \]
\[ H_a: \mu_{E,post2} \neq \mu_{C,post2} \]

This hypothesis examines the achievement scores on C language concepts. The scores are adjusted by the covariates of pretest and posttest 1 scores to reduce the error variance.

4. The product-moment correlation between pretest and posttest 1 does not differ from 0 at the 95% confidence level.
That is,
\[ H_0: \rho_{\text{pre,post1}} = 0 \]
\[ H_a: \rho_{\text{pre,post1}} \neq 0 \]

This hypothesis examines the degree to which the pretest is related to the achievement of assembly language concepts and contributed to the reduction of the error term in hypothesis 2 above.

5. The multiple correlations of posttest 2 with the variables pretest and posttest 1 do not differ from 0 at the 95% confidence level. That is,
\[ H_0: \rho_{\text{post2;pre,post1}} = 0 \]
\[ H_a: \rho_{\text{post2;pre,post1}} \neq 0 \]

The squared multiple correlations of the achievement test scores on C language concepts regressed on both the pretest and assembly language posttest 1 reflects the proportion of posttest 2 variance in common with pretest and posttest 1 variance. This would indicate the degree to which the error term in hypothesis 3 above is reduced by the use of the covariates.

Assumptions of the Study

This study was based upon the following assumptions:

1. The errors and the test scores were normally distributed, random, and independent.

2. The assigned sample size was sufficient for
estimation of population parameters.

3. The instruments (pretest, posttest 1, and posttest 2) have adequate reliability and validity.

4. The instruction time is sufficient to measure experimental effects on student performance.

5. The differential effects of the two instructors used in the control and experimental groups was minimal.

Limitations of the Study

This study was subject to the following limitations:

1. The participants of this study were limited to those students who enrolled in IEDT 216, Computer Application and IEDT 316, Advanced Computer Application, classes during the Fall semester of the 1993 school year in the Department of Industrial Education and Technology at Iowa State University.

2. The computer-based instruction program utilized in the experimental treatment group was limited in scope due to the time available for instruction to the following topics:
   a. arithmetic operations.
   b. branch operations.
   c. loop operations.
   d. subroutines.
3. The unit of observation in this study was the student. Because students received the treatments concurrently, there was a possibility of violation of the assumption of independence by students sharing information with each other both within groups and across treatment groups.

4. The lack of a compiler for assembly language within the computer-based program limited its application.

5. The pretest, posttest 1, and posttest 2 measurement instruments were samples of the knowledge domain and have errors of measurement which might reduce their sensitivity to treatment effects.

Definition of Terms

Assembly language - a low-level language in computer programming. In this language, both machine language operation codes and operands, instead of being written in binary code, are replaced by symbolic names.

Authoring system - a computer program which is designed to enable a person to create computer-based instruction lessons with a minimum of programming ability.
Courseware - a computer software which is designed to create some sort of instructional environment for the purpose of facilitating learning (Jonassen, 1988).

Debug - a file in DOS (Disk Operating System) which can be used to write assembly language programs.

High-order thinking skill - the ability to infer, integrate, evaluate, and solve problems which require critical thinking and monitoring.

Interaction - refers to two-way communication between a learner and a computer (Steinberg, 1991b).

Natural language - refers to human speaking language.

Programming - the process of expressing the steps required to perform a task in a language which the computer can execute.

Transfer - a procedure which activates knowledge in one cognitive area for utilization in another area.
CHAPTER II. REVIEW OF RELATED LITERATURE

The purpose of this review is to explore the usefulness of providing a computer-based assembly language program as an aid to students' understanding of high-level programming language. Initially, this review presents the transfer of natural language and programming languages. This is followed by the review of research on learning to program and computer-based instruction development. Finally, 20 principles for designing courseware are summarized.

Language Transfer

Transfer, according to the psychology of learning, is considered as the imposition of previously learned patterns onto a new learning situation (Gass, 1983). Transfer of knowledge and skills may occur when information that has been acquired in a prior context is used in a new situation where the information may be related (King-Johnson, 1992; Pudlowski, 1990). Dechert & Raupach (1989) also stated that transfer between known and unknown, old and new, verbal and nonverbal is the fundamental principle of human cognition.

Natural language transfer

The initial language transfer results from natural language transfer. In natural language, transfer refers to a psycholinguistic procedure which activates knowledge in one language for utilization in another language. Faerch & Kasper
(1989) indicated that the transfer procedures can be used in production, reception, and learning. Further, they claimed,

As a production procedure, transfer refers to the activation of the first language knowledge in the establishment of an interchange plan by means of which the learner seeks to realize a communicative intention. As a reception procedure, transfer implies that the learner attempts to interpret incoming second language utterances on the basis of his or her first language. As a learning procedure, transfer is used in the learner's attempt to establish hypotheses about the second language rules and items. (p. 174)

Research on language transfer appears to be based on a widespread assumption that transfer from the first language or native language is an important characteristic of second-language acquisition (Odlin, 1989; Coder, 1983). As Odlin stated, there was some evidence to support that cross-linguistic influences can occur in all linguistic subsystems, such as grammar, vocabulary, pronunciation, morphology and syntax, etc. Koda's (1988) study found that the native language orthographic characteristics influence cognitive processing in second-language learning. He also suggested that cognitive process transfer does occur in the second language.

In contrast, several studies argued that the influence of
the first language on second-language acquisition and performance does not always reveal itself in obvious ways (e.g., Kellerman, 1983; Tzeng & Wang, 1983). However, despite the counter arguments, an abundance of research exists that indicates transfer is an important factor in second language acquisition.

**Programming language transfer**

With respect to programming language, researchers have delved into the transfer of programming to problem-solving skill. The researchers hypothesized that learning a programming language would enhance self-consciousness about the process of problem-solving (Feurzeig, et al., 1981; Linn 1985; Nickerson, 1988; Palumbo, 1990; Palumbo & Reed, 1991). Furthermore, Pea and Kurland (1984) proposed that (1) different levels of programming proficiency may enable the transfer of different concepts and skills; (2) transfer to problem-solving and planning may require considerable metacognition; (3) transfer does not occur spontaneously but requires guidance and modeling.

On the other hand, Linn (1985) indicated that no assessment of problem-solving skills is independent from programming transfer. Transfer could happen from one programming language to another. Linn selected twenty-four students from a sample of 2400 high school students studying BASIC. These 24 students were able to program in BASIC. They
received a brief introduction to a new elementary language called "Spider World" and were asked to write three programs. The results revealed that many of the students could apply the skills required while studying BASIC to the new programming language "Spider World." Further, Dalbey & Linn (1986) investigated the cognitive consequences of learning the Spider World programming in two junior high schools. Likewise, the results indicated that students were able to make a smoother transition to the BASIC programming language. In these two cases, the students who had experience with the initial programming environment were able to transfer the concepts to solving problems in another programming language. As these two researchers concluded, the initial programming language environment and the environment to be transferred to have many similar features, such as the function of commands and the rationale of the concepts. Thus, transfer is easier.

Learning assembly language programming is also a critical path to understanding how high-level language constructs can be implemented and about the underlying machine's supporting operating systems, languages, data bases, applications, etc. (Little & Smotherman, 1988). Donahue (1988) described the use of a simplified assembly language to teach concepts in an introductory programming course. He postulated that if structured programming is taught at the assembly level the concepts could transfer to many languages. For example, to students beginning with a high-level language, a variable name
can be a mysterious concept. Students have no idea what a variable actually is doing; whereas if students have learned the concepts of assembly language they are familiar with memory locations and their contents. Thus, the transfer is made that a variable is just a name for a memory location.

From the standpoint of cognitive processing, all knowledge is acquired in a context. For most learners, specific knowledge acquired from reading, listening, or observing permits them to apply the knowledge effectively in a variety of settings (Glover, et al., 1990). Assembly language, thus, seems to serve as a vehicle for learning high-level programming.

Research on Learning to Program

The teaching of computer programming has become a popular activity. Programming is a skill that can promote high-order thinking skills, such as problem-solving (Palumbo, 1990; Palumbo & Reed, 1991; Soloway, 1988). However, programming is a complex cognitive skill, and learning to program might result in a wide range of cognitive outcomes (Delbey & Linn, 1986). Hence, many studies have explored how to develop programming skills.

Methodology of programming instruction

In the view of cognitive psychology, learning is focused on two distinct types of knowledge: declarative knowledge and
procedural knowledge (Gagne, 1985). In essence, learning to program requires the development of both the declarative and procedural knowledge of the learner (Mandinach & Linn, 1986). That is to say, the goal of programming instruction is to teach learners the command structure of a particular language (declarative knowledge) and then provide them with situations in which they must use this information in different ways (procedural knowledge) (Palumbo, 1990). Dijkstra (1989) viewed computer programs as symbolic formulas which allow programmers to perform convenient manipulations. Cohen (as reported in Dijkstra, 1989) stated that computer science courses should provide approaches to teach students how to reason through programs. In accordance with Dalbey and Linn (1985), the most important concept to be taught when learning to program is "algorithmic thinking." That is, writing a program should be thought of as a three-phase process: (1) defining the problem, (2) creating the step-by-step solution, and (3) coding the steps as instructions for the equipment to be used. Thus, learners acquire the skill and knowledge for all three phases.

Habermann (1991) argued that improvement of writing programs can be achieved by three approaches:

(1) By replacing programming from scratch by a "systems view." A systems view of programming means that students can see a program right from the beginning as an integral part of a collection of program modules that allow them to get an idea
of what a system can do.

(2) By trading program time and space wisely for quality and reuse. It is not easy to talk about the quality of a program. Thus, the best that the instructor can do is to show the students examples of quality programs, by comparing them with similar programs and abstract models, and discuss the quality of the programs. This gives students a rich software base that offers ample opportunity for them to study and make the best choices.

(3) By building a programming theory. The purpose of a programming theory is to classify programming problems and find criterion solution in problems. The emphasis is on program schemata for standard problems, not on particular techniques.

From a cognitive psychology standpoint, the learning process is facilitated when learners have the opportunity to observe and interact with visual models of abstract concepts (Greeno, 1978). Mayer (1981) suggested that teachers must foster meaningful learning of computer concepts to novices. He argued that using concrete models to represent the computer system might enhance the learners' understanding. Hooper and Thomas (1990) investigated the effects of a manipulative model of computer memory operations on learning to program. The results showed that students who used the model prior to learning the Pascal programming language more easily acquired complex programming techniques. They also suggested that
expanded use of manipulative models could significantly improve the learning of programming.

Schweppe (1973) recognized that the basic concepts of computer organization and programming are quite simple to learn if the dynamics of change in a computer are visualized. Actually, it is often argued that visual and dynamic instructional models are motivational in learning about computer organization and programming language (e.g., Rajan, 1991; Roman & Cox, 1989; Schweppe, 1973; Shapiro & Witmer, 1974; Tangorra, 1990).

Assembly language in transition

Numerous high-level languages have been developed that permit the programmer to express instructions in a symbology that is easier to use than machine-code. However, the assembly language is often used in an introductory computer course to aid in the comprehension of the fundamentals of computer structure and organization, and to reinforce structured programming concepts as well (Cook, 1982; Donahue, 1988; Leeper & Rehmer, 1986; Mackenzie, 1988; Nickerson, 1988; Tangorra, 1990; Tran & Robillard, 1985). Tangorra indicated that learning assembly language served to reinforce many concepts, such as: the hexadecimal representation for both memory address and memory contents, the increment of the program counter based on the instruction length, the two's complement representation of negative numbers, and the ASCII
character set. Evidence exists that an understanding of assembly language can help a programmer in several ways:

1. A knowledge of assembly language can facilitate learning of any other language, including a high-level language (Abel, 1989).

2. A knowledge of assembly language and its resulting machine code provides an understanding of machine architecture that no high-level language can possibly provide (Abel, 1984). In particular, the user can understand more clearly what is really happening when the computer executes a program (Donahue, 1988; du Boulay, et al., 1981).

3. A knowledge of assembly language can help a programmer become more efficient. That is, a programmer who is familiar with assembly language can code high-level languages with an understanding of the machine code generated and efficient techniques (Abel, 1989).

4. A knowledge of assembly language can help when debugging and analyzing malfunctioning programs written in a high-level language (Yarmish & Yarmish, 1979).

5. The high-level language may produce unexpected results such as arithmetic overflow, and simple changes in a high-level language program may result in large changes in resulting time or memory usage. A knowledge of assembly language is useful for understanding and predicting these results (Kapps & Stafford, 1981).
Computer-Based Instruction Development

Computer-based instruction (CBI) can be defined as "computer-presented instruction that is individualized, interactive, and guided" (Steinberg, 1991b, p. 2). For over a quarter of a century, there has been considerable research on CBI environments. The last decade is characterized by rapid development in this field. The reason may result from current educational reform that causes development of new ways to assist learning. The other factor that contributed to the developments in computer-based learning environments may be found in learning theories and instructional science (Dijkstra, et al., 1992).

Two different theories of learning, behavioral theory and cognitive theory, dominated psychology during this century. At the beginning of CBI development, CBI was rooted in behavioral psychology. Eventually, theories of instruction and learning were developed in terms of cognitive processes of learning (Dijkstra, et al., 1992).

In addition, courseware design is a process and a discipline that basically derives from learning and instructional theories (Jonassen, 1988). The designer must approach the courseware with a critical mind.

Unique features in computer-based instruction

Evidence has shown that CBI can be as effective as traditional classroom instruction for conveying knowledge and
skill in areas of writing, mathematics, and science (e.g., Krendl & Lieberman, 1988; Niemiec & Walberg, 1987). In particular, CBI programs could be designed to take into account individual differences of each student so that each student is provided a unique path for his or her appropriate learning. In other words, it permits the learner to control the pace of the presentation of information (Gisolfi, et al., 1993; Yano, et al., 1992; Vockell & Schwartz, 1992).

In the view of Kozma (1991), the computer can influence the mental representations and cognitive processes of the learner. Computers also have the capability of creating dynamic, symbolic representations of nonconcrete, formal constructs that are frequently missed in the mental methods of the learner. More subtly, they are able to proceduralize the relationships between these objects. The learner can manipulate these representations within computer microworlds to work out differences between the incomplete, inaccurate mental models and the formal principles represented in the system. In addition, the processing capabilities of the computer can help the learner build and refine mental models to solve problems.

Furthermore, advances in technology are making the computer easier to use to represent knowledge in a variety of modes, such as visual, animation, sound, text, graphics, interaction, and freedom to make mistakes without fear of censure (Hunter, 1989; Keller & Suzuki, 1988). The related
research, for example, comparing information retrieval tasks using dynamic text and paper text, revealed that dynamic text gave better feedback in answering difficult questions and was more effective in practice (Gray, 1991). Although Rieber, et al. (1990) indicated that animation did not affect learning, their study also argued that the animated visual presentations significantly aided the learner by decreasing the time necessary to retrieve information from long-term memory and then subsequently reconstruct it in short-term memory.

Graphics, in addition to enhancing the visual appeal of screen presentations (Kulik, 1983), help to make the information concrete (Shu, 1988). Improved user interfaces help the learner communicate with the CBI program, which could lead to an increase in clarity and a decrease in the amount of information that the learner needs to learn (Shu, 1988). Further, Jacobson (1992) indicated that computers could extend the mind by providing a strong component of learning as a continuous part of an ongoing multisensory interaction.

Learning theory

For most of the first half of this century, research on learning has been conducted within a behavioral framework. Learning, according to behaviorism, consists of a stimulus hitting the eyes, the ears, or other senses that is followed by a response. The stimulus is then linked to the response by reinforcement which emphasizes changes in behavior as the
outcome of learning (McKeachie, 1991). Behaviorists such as Skinner and Watson held the view that, since thinking, intentions, emotions, and other mental events are not observable, these mental operations should not be included in the explanation of learning (Jonassen, 1991; Woolfolk, 1984).

Unlike the behaviorists, who are only concerned with what learners do, cognitive psychologists are interested in what learners know and how they acquire knowledge (Jonassen, 1991). Learning, as cognitivists view it, can be defined as a change or reorganization of insight and understanding (Morris, 1978). According to Bigge (1971), learning is

a change in knowledge, skills, attitudes, values or beliefs, and it may or may not be closely related to some change in overt behavior. One does not learn by doing except insofar as one's doing contributes to one's change in cognitive structure. (p. 271)

Namely, cognitive psychology is concerned with various mental activities, such as perception, thinking, knowledge representation, and memory, related to human information processing, and it presently represents the mainstream of thinking in education (Shuell, 1986). With regard to information processing, learning focuses on the nature of the memory system. Human brains can be compared to computers in their gathering, storing, and sorting of information.

Therefore, in the educational setting, both teachers and
students have considerable influence over how information is interpreted, how it is placed in memory, and how it is retrieved (Kiewra, 1987). In fact, learning processes are the way in which learners encode to-be-learned information. Accordingly, the mode of instruction should be intended to affect the way that learners select, organize, and integrate information (Mayer, 1984).

Furthermore, Bagley and Hunter (1992) indicated that current theory views learning as an active engagement of learners in the construction of their own knowledge and understanding of facts, processes, and concepts. This learning occurs through interaction with and support from the world of people and technologies. Learning becomes a dynamic process; teaching should involve less "telling" and more supporting, facilitating, and coaching of learners. According to these authors, students should have freedom to choose what and how they learn, with each student realizing the maximum learning possible.

In essence, learning is an active, constructive, cumulative, self-regulated, and goal-oriented process, whereby the learner manages the available cognitive resources to create new knowledge by extracting information from the environment and integrating it with information already stored in memory (Jonassen, 1988; Kozma, 1991; Shuell, 1992). However, not everyone learns in the same way, and there is not a single best way in which all learners should learn. The
learning process depends on the learner's prior knowledge and the type of outcome that he or she is trying to achieve. In addition, learning from instruction involves affective, motivational, and metacognitive processes as well as the cognitive processes that must commonly come to mind (McCombs & Whisler, 1989; Shuell, 1992; Tobias, 1989). For meaningful learning to occur, the learners have to attend to relevant information, build internal connections among the pieces of information, and build external connections between the information and relevant existing knowledge (Mayer, 1989).

Additionally, advanced knowledge acquisition becomes more complex and the relationships across the cases to which that knowledge has to be applied becomes more irregular. Thereby, it is crucial to have multiple representations, such as multiple explanations and multiple dimensions of analysis, for instruction (Spiro, et al., 1987). Further, Spiro & Jehng (1990) described a claim that "mental representations need to be open rather than rigid and closed; nonlinear instructional sequences need to be followed to avoid missing key points" (p. 169). That is to say, learning has to refer to cognitive flexibility, the ability to reassemble diverse elements of knowledge adaptively to fit the particular needs of a given understanding or problem-solving situation.

To summarize, the concepts of traditional programmed instruction were rooted in behaviorism. Although behaviorist theory is not adaptive in as many aspects of instruction as
CBI, the computer is still a good tool for implementing behavioral principles (Steinberg, 1991b). In contrast, Webb (1988) rejected the behavioristic precepts of programmed learning and argued in favor of cognitive implementations which emphasized the learner's contributions to the learning activity. Higgins and Johns (as reported in Kashihara, et al., 1992) argued that the cognitive implementation might be realized in part through the computer's adopting of such teaching "roles" as trainer, simulator, and game player at various stages through a tutorial session. However, it is apparent that CBI can benefit by incorporating these two theories to improve quality and productivity in the learning environment (Steinberg, 1991b).

Courseware design

From the cognitive view, CBI could provide learners an active, constructive, and playful environment (Saettler, 1990). Thus, courseware design is a professional activity. As Steinberg (1991b) stated, it includes "not only the text, but visuals, question-response-feedback sequences and human factors information" (p. 77). Reigeluth (1983) held the view that courseware design is "the process of deciding which methods of instruction are best for bringing about desired changes in student knowledge and skills for a specific student population" (p. 7). Therefore, courseware design must take into account cognitive psychology and interactive technology
to structure a computer-based environment to foster learning (Tennyson, 1990).

In other words, a plan for designing CBI courseware must take into account the connections among the applicable aspects of known instructional design procedures, the unique features of CBI, as well as the principles of software engineering user interface design (Hazen, 1985; Montague, 1988; Muraida & Spector, 1993; Shuell, 1992; Steinberg, 1991a; Steinberg, 1991b). As a result, the primary issue is specifying the learning environment, contents, and learner interaction. Likewise, Steinberg indicated that the courseware design should attend to all components of the CBI framework (e.g., target, population, goals, tasks, instruction, computer application, and environmental implementation) and their interactions.

In addition, another critical component of CBI is the interaction of a learner with a computer. Interaction is not original to computer courseware; it is rooted in Socratic tutorials (Jonassen, 1988; Merrill, 1987). However, CBI provides greater potential for truly interactive instruction than any mediated learning devices to date. Interactions serve two main functions in CBI: (1) the mechanics of interacting with the computer system, and (2) the acquisition of knowledge and skills (Steinberg, 1991b). Furthermore, Hazen (1985) suggested that the feature of interaction is at all times the most important feature of computing courseware.
Interactive graphing, windowing, and animation differentiate the computer as a medium from most other media.

The designer must be careful to make use of screen formatting and graphics for readability and instructional purposes, and not just for decorative or cute effects. Display design, the link between the learner and the computer, is particularly crucial in CBI (Lurie, 1992). Well-designed screens can be a motivating factor. Using different colors or highlighting with either single lines, boxes, bars, or windows for emphasis and to communicate organization can be attractive to the eye and lead to rapid recognition and identification to screen objects. Yet color combinations need to be easy on the eye. It is more effective to avoid crowding as much text as possible on a screen, since crowded text hinders comprehension. Also, creating too complex a screen (e.g., too many colors) and the use of "hot" colors (Christ, 1975; Durrett, et al., 1982; Shneiderman, 1987; Sweeters, 1985) should be avoided.

In attempting to simplify learning to improve instructional efficiency and effectiveness, educational computing technology may be short-circuiting relevant mental processing. Educators have delved into how to ensure that learners will be able to regulate their own learning effectively as they exercise the control inherent in a computer-based system. How can motivation be stimulated and maintained, so that individuals will go beyond superficial
browsing to really explore and learn? Significantly, courseware design is critical. However, courseware design cannot be reduced to a single formula. This fact evidenced in the following principles which are advocated in the literature (Cates, 1992; Cook & Kazlauskas, 1993; du Boulay, et al., 1981; Hunnum, 1988; Reid, et al., 1993; Kinzie & Berdel, 1990; Mackey, 1987; Steinberg, 1991b; Taber-Brown, 1992; Vockell & Schwartz, 1992):

1. Identify goals and objectives.
2. Identify the domain of the instructional outcome.
3. Use knowledge structure.
4. Analyze the task to be accomplished.
5. Match current instructional time restraints.
6. Focus on one concept at a time by using screen space effectively.
7. Give sufficient overview at the beginning of the program.
8. Use a branching program.
9. Provide summaries and reviews, restate important concepts.
10. Permit the learner to back up to re-examine previous screens.
11. Assist in developing learners' inquiry skills.
12. Incorporate learner control.
13. Encourage learners to think about what they know, what they are learning, and how to learn.
15. Design an interesting and motivating context.
16. Provide effective feedback.
17. Use graphics, color, etc. appropriately.
18. Design a "user-friendly" learning environment.
19. Make the product interactive in meaningful ways.

Although these principles afford some design flexibility, strict adherence to them would produce a highly structured program of a tutorial or drill and practice type. Microworlds and simulations which encourage student exploration and control are not produced through task analysis and do not contain branching and student guidance.

Summary of Literature Review

The purpose of educational systems tends to focus on developing two types of knowledge: declarative knowledge, that is, specific facts, concepts, and principles; and procedural knowledge, that is, active, strategic use of declarative knowledge. Programming languages attempt to develop both the declarative and procedural knowledge of the learners involved in the educational system. More importantly, effective instruction needs to focus on the procedural knowledge features of programming, which are more cognitively demanding. The better that a learner's knowledge about the programming domain is organized, the more likely it
is that heuristic problem-solving processes will lead to a correct solution. Accordingly, offering concrete examples that characterize programming plans or models is crucial. The information should be available to present the conceptual information that seems helpful to learn to program before units of practice, but also to keep this information available for the learners during the whole learning process.

Learning assembly language in a computer programming class should not aim exclusively at the acquisition of assembly language per se, but address learners' schematic knowledge of computer organization that is used to help learning to program in high-level languages. Therefore, the computer-presented program for assembly language should promote the elaboration of schemata of computer organization that are needed to reason through the precise conditions that enable programming.

With regard to cognitive learning, developments in technology are causing many educators to be involved in the design, development, and evaluation of a variety of courseware. Good design is conscious and systematic, part art and part science. The ultimate goal is the understanding of the intended learner; therefore, courseware design should fit the people and purpose for whom it is intended. In other words, courseware design has to take a more learner-oriented view about what has to be learned and how to support the learning of course contents. It should be designed to permit
the learner to interact with the computer, to exert some control over the technology when acquiring knowledge from the computer-based program. The more control that is exerted by the user, the more dynamic the interaction can be.

The computer gives CBI the power of flexibility and serviceability. More importantly, it gives education the opportunity whereby precise individualization could be achieved. It also permits the learner to learn domain knowledge in a variety of ways. Accordingly, there is little doubt that computers are an appropriate instruction medium if the lesson presented is effective, efficient, and applicable to the intended learners. CBI designers should continue to design creative and innovative interactive dynamics, as the literature indicated that the time necessary to design such animated visuals is worthwhile.
CHAPTER III. METHODS AND PROCEDURES

This chapter summarizes the research methodology of this study, and is organized into six sections: (1) subjects, (2) description of the computer-based program, (3) instruments, (4) research design, (5) research procedures, and (6) data analysis.

Subjects

The subjects participating in this study were students enrolled in the Computer Application class (IEDT 216) and the Advanced Computer Application class (IEDT 316) during the fall semester of 1993, in the Department of Industrial Education and Technology at Iowa State University. There were two sections of IEDT 216. Most students enrolled in the course had little programming background. There was one section of IEDT 316. The majority of students enrolled in IEDT 316 had already taken the IEDT 216 Computer Application class where they learned the basic concepts and skills of C programming.

Initially, 38 students enrolled in IEDT 216 and 16 students enrolled in IEDT 316. Because a student in IEDT 216 did not attend class in the pretest hour, a total of 53 students took the pretest at the beginning of the classes. After the first two weeks of the courses, two students dropped the IEDT 216 course, and two dropped the IEDT 316 course. Therefore, data were collected for a total of 49 subjects on
posttest 1. Also, 39 subjects were randomly selected for two-way analysis of covariance in order to obtain proportional cell sample sizes.

During the fifth through ninth weeks, five students dropped IEDT 216 and two dropped IEDT 316. Thus, data were collected for 30 subjects in IEDT 216, and 12 subjects in IEDT 316 on posttest 2. In IEDT 216, 24 subjects were also randomly selected for two-way analysis of covariance. The use of these subjects was approved by the Iowa State University Human Subjects Review Committee.

Description of the Computer-Based Program

LEARNIT, an animated computer-based program, was constructed for 8086 assembly language and run on an IBM-compatible 80486 PC. The program attempted to enhance learning by providing the learner an optimum level of interactivity and learner control. Students could proceed at their own pace, review screens, look at a procedure until it became automatic, and monitor and control their progress.

To present students with a clear, intuitive, and consistent view of program execution, LEARNIT used an "arrow" icon to trace the internal information or control flow of instructions as the fetch and execute cycles were performed. In addition, the color of the contents in the register or memory field changed, so that student's attention would be drawn to the current values of variables. It was expected
that students could visualize and build an accurate mental model of what actually was happening inside of the computer when a program was running.

Eight lessons were developed using IBM authoring software called "LinkWay Live," as shown in Figure 1: (1) Getting Started, (2) Fetch and Execute, (3) Basic Concepts (4) Arithmetic Operations, (5) Branch Operations, (6) Loop Operations, (7) Subroutines, and (8) Program Interrupts. This screen provided two buttons: the "Retrace" button, which allowed the student to return to the screen of the previous lesson; and the "Exit" button, which allowed the student to return to DOS (Disk Operation System). The choice of the five lessons (lessons 4-8) was based on the desire to give students a small but representative sample of basic programming concepts.

Lesson 1, "Getting Started," in addition to a description of the purpose of LEARNIT, described the functions of the buttons (e.g., "Main," "Bye," "Next," "Back," etc.). For example, the "Bye" icon was provided in each page for students to return to DOS at any time. A "Main" button was shown on some pages in order to return to the main topics page so that the student might choose another lesson (as shown in Figure 2). Also, for each lesson the page number was displayed in a manner that told students the amount of time a lesson should take.
Figure 1. The Main Topics Page

Figure 2. A Sample of CPU Architecture Description
Lesson 2, "Basic Concepts," presented the basic CPU (Central Process Unit) architecture, such as ALU (Arithmetic Logic Unit), general purpose registers, instruction register, instruction pointer, segments, stack pointer, flag register, etc. Figure 2 is a screen in this lesson.

The concepts of "fetch" and "execute" were introduced as the third lesson because the remaining five lessons were built upon a series of instructions to be fetched and executed. In lesson 3, a fetch and execute example was shown to provide students an example of program operation before they began the remaining lessons. As shown in Figure 3, the "Fetch" and "Execute" buttons were in the lower right on the page. When students clicked on the "Fetch" button, the fetch action was initiated. After the fetch action was completed the student could click on the "Execute" button. If the student clicked on "Execute" before clicking on "Fetch," a beep was sounded to gain the learner's attention for the error in procedure.

Figure 4 shows the screen before fetching the instruction "MOV AL, 86H." Upon fetch of the instruction, the IR (Instruction Register) displayed "XXXX." That denoted an undefined code; and the IP (Instruction Pointer) was set at "0000." Figure 5 shows the effect on the IR and IP values after the instruction was fetched by an 8086 processor. The result was that "MOV AL, 86H" was moved to the IR and the IP was updated to "0002." After the fetch action, a question was displayed and the student could type the answer in the block.
Figure 3. The Page for Choosing the Fetch Button

Figure 4. The Screen before Fetch "MOV AL, 86H"
Figure 5. The Screen after Fetch "MOV AL, 86H"

Figure 6. After Clicking on the Check Button, the Answer is Displayed
The student could then click on the "Check" button where a scrolled window with the stored answer was displayed. This is shown at the lower right in Figure 6. Likewise, the execute action was presented in a similar manner in this lesson.

Lessons 4-8 showed the execution sequence of a program. The main task of the device was to simulate the processor's work. The memory locations for a program, registers for data and status, and the ALU were used. The trace-time code was blinking three times by video highlighting with a color, so that the student would see the inner working of a microprocessor as it executed an instruction or a program which had been written in 8086 assembly language. Lessons 4-8 are described next.

**Arithmetic Operations**

The Arithmetic Operations lesson was designed to give the students an elementary example of what a processor would do when an arithmetic function was executed. Using three instructions, MOV, ADD, and MUL, the student worked through a sequential fetch and execute exercise to be familiar with the following concepts: (1) storing program code in memory locations, (2) processor action during fetch and execute cycles, (3) the functions of the IR and IP, (4) arithmetic operations computed in the ALU, and (5) the result of a computation on the status register (flag register).
The four instructions presented moved the values (86 and 6) to the registers (AL and BL), added the contents of the register AL to the value 5, multiplied the contents of the AL by the BL, and stored the result to the register AX. The results of program execution is shown in Figure 7. Students performed these instructions by using fetch and execute buttons to trace the program. Following the completed execution of "ADD" and "MUL" instructions, the student could click on the "Flag" button (also in Figure 7) to get more information about the status of the flag register (Figure 8 and Figure 9).

This lesson presented some questions to the learner during pauses in the execution of the program. Additionally, three exercises were enclosed at the end of the lesson to examine the student's understanding of the lesson. Because the questions were of the open-ended type, judging a student's response was too difficult. To circumvent this problem, clicking on the "Check" button caused the current answers to be displayed. Figure 10 is an example.

**Branch Operations**

The Branch Operations lesson was presented to facilitate student learning of the concepts of program control. The program performed a comparison of two unsigned numbers which were stored at two memory locations. After executing the program, the largest value was moved to one location and the
Figure 7. The Screen for the Results of Program Execution includes a "Flag" Button

Figure 8. The First Page Regarding Flag Information
Figure 9. The Second Page Regarding Flag Information

Zero Flag (ZF): If the result of the operation is zero and a ZF is set if the result is non-zero.
Sign Flag (SF): Is set to 1 if the result is negative, otherwise, it is cleared if the MSB is 0.
Overflow Flag (OF): Is set to 1 if the sign of the result is out of the range, otherwise it is zero. More specifically, for addition, the flag is set if 1 is when there is a carry into the MSB and none coming out of the MSB or vice versa. For subtraction, it is set if 1 when the MSB needs a borrow and there is no borrow from the MSB.

Here, the "MUL" instruction performed the multiplication 1000 x 1011.

\[
\begin{array}{c|c|c|c|c|c}
    & 1000 & 1011 & 0010 & 0010 & 0010 \\
\hline
    & 1000 & 1011 & 0000 & 0110 & 0110 \\
\end{array}
\]

Here, the flags are:
OF 1 (the sign of the result is out of the range)
ZF 1 (there is a carry)
Notice that the four flags (SF, ZF, OF, and PF) are not affected, that is, they are not changed.

Figure 10. An Example of Exercises

Exercise 3:

Write a program that can execute the arithmetic operation "EX = 5 + 2"

Click on Check to check your answer.

Back Exit
smallest value was moved to the location which followed. Actually, this program was analogous to a bubble sort program.

The demonstration was divided into two parts. First, the two values were stored in memory locations F000 and F001 respectively (Figure 11) in order to meet the conditional instruction "JA" and shift the control to label L2. Then, the contents of the two memory locations were swapped, that is, the value 66 was in F000 and 97 was in F001 (Figure 12) in order to demonstrate that when the jump condition is not met the processor continues to execute the instruction which follows the comparison test.

The purpose of this program was to help students become familiar with the following concepts: (1) program code must be stored in a code segment and data code may be stored in a data segment of memory, (2) the jump instruction must always be preceded by a comparison, (3) a comparison instruction sets the status flags to specify the jump, (4) program flow, and (5) the use of labels to represent memory locations.

Loop Operations

The Loop Operations lesson was designed to give students a concrete concept of recurrence by using the register CX as a counter to set up the recurrent cycles and a segment of code which was executed as a loop (Figure 13). In this lesson, the loop code contained three instructions: ADD AL,[SI], INC SI, and LOOP CYCLE. The purpose of the lesson was to facilitate
Figure 11. Layout of Memory to Meet JA Condition

Figure 12. Layout of Memory not to Meet JA Condition
student's understanding of the following concepts: (1) a repetitive action must use a segment of code to perform the recurrence, (2) a counter must be used to set up a repetitive operation, (3) a single loop instruction could be directly used for recurrence, and (4) after the loop cycle is completed the processor executes the instruction following the loop code in the program sequence.

In order to explain looping, the looping action was presented one loop at a time in detail. Therefore, after executing one looping code, students could choose to continue the next looping code execution or go back to review the previous looping action, as shown in Figure 14.

Figure 13. Loop Operation Program
Subroutines

This lesson was designed to explain the task of calling a procedure in a program. In this program (Figure 15), the procedure was to calculate the factorial of a number. By tracing the instructions "CALL" and "RET," students should understand how a processor transfers the flow of control. That is, the "CALL" calls a subroutine and the flow of control was transferred to the subroutine; the "RET" sends the processor back to the next instruction after the instruction "CALL" in the original sequence.

In addition to presenting the concept of calling a procedure, more importantly, this lesson provided a concrete
A Subroutine for Calculating the Factorial of a Number

The program at the right computes the factorial of a number and increases it from the result. The subroutine for factorial calculation program is put in a subroutine. This program will perform the calculation of F! as defined as

\[ F! = n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1 \]

Figure 15. Subroutine Program

illustration of using a stack to preserve the information before executing the subroutine. By looking at the screen, students could see how the processor got the instruction address required for the subroutine and to return to the main program.

Program Interrupts

The Program Interrupts lesson (Figure 16) was designed to provide students an example of the overall concepts of a computer system as well as the vital purpose and role of interrupts. This lesson emphasized the concept of microprocessor action while the computer accepted signals from the outside world or dispatched messages to output.
Therefore, two functions as interrupt examples were presented. The first function printed a string on the screen. The second function terminated the program.

From the two interrupt examples students would see the services in interrupts as well as the concept of control flow in a processor. That is, when the interrupt activity happened, the flow of control was diverted from the normal instruction sequence and, the processor was said to be performing an interrupt routine function. Hence, the memory locations for interrupt subroutine code was shown in the bottom (the second CS) in Figure 16.

![Figure 16. Layout of Program Interrupts](image)
Instruments

Three types of instruments were designed to measure a student's knowledge regarding the computer organization, assembly language, and the ability to program in C language. Likewise, these instruments were used to collect the data pertinent to this study. They were the following:

Pretest

The pretest consisted of 40 multiple-choice items. Ten items were developed to measure knowledge of basic computer organization; twenty items were developed to measure knowledge of 8086 assembly language used in arithmetic, branch, loop, subroutine, and interrupt fields. This test also had ten items to measure knowledge of C programming. Refer to Appendix A for a copy of this test. The alpha reliability statistic for this test was .49.

Posttest 1

Posttest 1 was intended to measure whether students had acquired knowledge of computer organization and assembly language during the four weeks of instruction. The test consisted of 51 items and included three parts. One part had 10 matching items regarding computer concepts such as memory and registers. The alpha reliability statistic for this part was .47. A second part had 26 multiple-choice items regarding assembly language to measure fundamental programming concepts
such as branching, loops, subroutine calls, and interrupts. The alpha reliability statistic for this part was .61. And, a third part had 15 items to measure student's comprehension of DOS commands. The alpha reliability statistic for this part was .58. The alpha reliability statistic for the 51 items was .69. Refer to Appendix B for a copy of this test.

Posttest 2

Posttest 2 for the IEDT 216 sections was constructed to measure a student's knowledge of C language. It consisted of 14 multiple-choice questions of 14 points and a 10-point programming activity. The programming activity was a short assignment to construct a C language function and could be performed on an IBM PC "clone" using the Borland C compiler. Both parts were used to measure student's knowledge regarding the concepts of variables and their types (e.g., int, real, float, char, etc.), program function, C language syntax, and program control (for, while, exit, do while, break, continue). The alpha reliability statistic for this test was .63. Refer to Appendix C for a copy of this test.

Posttest 2 for the IEDT 316 section was also constructed to measure a student's knowledge of C language. It consisted of 25 multiple-choice questions of 25 points and a 10-point programming activity. The programming activity was a short assignment to construct a C language function; students could perform this question on an IBM PC "clone"
using the Borland C compiler. Both parts were used to measure student's knowledge regarding the concepts of functions (types, arguments, prototypes, etc.), structures, classes, objects, inheritance, overloading, and pointers. The alpha reliability statistic for this test was .36. Refer to Appendix D for a copy of this test.

Research Design

The design of the experiment consisted of two groups: one experimental group and one control group. The experimental group consisted of 25 subjects who received instruction on assembly language using the computer-based program LEARNIT. The control group consisted of 24 subjects who received instruction on assembly language using live instruction (traditional lecture/demonstration/practice). The assignment of the two groups is shown in Figure 17.

<table>
<thead>
<tr>
<th></th>
<th>Control</th>
<th>Experimental</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEDT 216A</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>IEDT 216B</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>IEDT 216A+216B</td>
<td>18</td>
<td>17</td>
</tr>
<tr>
<td>IEDT 316</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>TOTAL</td>
<td>24</td>
<td>25</td>
</tr>
</tbody>
</table>

Figure 17. The Assignment of Two Groups
Furthermore, because a two-way analysis of covariance procedure was used, the assigned subjects in each group were randomly selected again in order to make each class have the same sample size in the two groups. Hence, in posttest 1, 8 and 7 subjects were selected for each class in the control group and experimental group respectively (Figure 18). In posttest 2, 7 and 5 subjects were selected for IEDT A and B sections in the control group and experimental group respectively (Figure 19).

<table>
<thead>
<tr>
<th>GROUP</th>
<th>Control</th>
<th>Experimental</th>
</tr>
</thead>
<tbody>
<tr>
<td>C IEDT 216A</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>L IEDT 216B</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>A IEDT 316</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>S TOTAL</td>
<td>18</td>
<td>21</td>
</tr>
</tbody>
</table>

**Figure 18. The Assignment of Subjects for Two-Way ANCOVA in Posttest 1**

<table>
<thead>
<tr>
<th>GROUP</th>
<th>Control</th>
<th>Experimental</th>
</tr>
</thead>
<tbody>
<tr>
<td>C IEDT 216A</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>L IEDT 216B</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>A TOTAL</td>
<td>14</td>
<td>10</td>
</tr>
</tbody>
</table>

**Figure 19. The Assignment of Subjects for Two-Way ANCOVA in Posttest 2**
Research Procedures

The study was implemented during the first eight weeks of the semester. During the first hour of the class (two-hour class lecture and two-hour lab for each week), the course syllabus was distributed and discussed. Then, during the second hour the pretest was administered. During the next class period, the students were randomly assigned to two groups: experimental group and control group. A comprehensive introduction of computer organization was conducted over 4 hours using lecture for both groups.

In the remaining sessions, the experiment was conducted in two parts: learning assembly language and learning C programming. The first part took place during the third hour of the second week through the fourth week of the semester. During this time period, the experimental group used the computer-based program LEARNIT. In addition, they practiced simple programs using the DOS "debug" program on IBM-compatible machines. The control group learned assembly language using traditional lecture and demonstration and also practiced programming in assembly language using the "debug" program. Then, during the first hour of the fifth week, the first experimental measurement, posttest 1, was given to measure the knowledge of computer concepts and assembly language.

In the second part of the experiment, the two groups
learned C programming in the same environment using lecture and programming activities on IBM PC-compatible machines. At the third hour of the ninth week, posttest 2 was given to measure the students' ability in C programming.

**Data Analysis**

Measurement data were collected from three examinations: pretest, posttest 1, and posttest 2. These examinations consisted of multiple-choice, matching, or programming items. The examinations were written by the instructor. After the students completed each exam, the items were scored by the researcher. All items were scored dichotomously (1 for correct, 0 for incorrect).

Means and standard deviations were used to describe the distribution of the data. A one-way analysis of covariance procedure was conducted to examine whether treatment effects were significant. A two-way analysis of covariance procedure was completed to determine whether there were any differences between classes (IEDT 216A, IEDT 216B, and IEDT 316) due to the types of instruction (traditional lecture and the computer-based program) or interaction of class and treatment group. The level of rejection for the null hypotheses was the probability of a larger sample statistic being less than 0.05 (p < .05).
CHAPTER IV. RESULTS

The purpose of this study was to determine if the computer-based program LEARNIT was an effective way of learning assembly language and if assembly language instruction facilitated the subsequent learning of the C language using the computer-based program LEARNIT. The criterion measurement consisted of three tests to examine the five hypotheses presented in Chapter I. The pretest was to test the basic concept of computers, assembly language, and C language. Posttest 1 was to test the computer organization, assembly language, and DOS commands. Posttest 2 was to test the knowledge of C language. The primary data analyses conducted in this study were a series of one-way ANCOVA (Analysis of Covariance) and two-way ANCOVA procedures, Pearson product-moment correlations, and squared multiple correlations. The results for the hypotheses examined are reported below.

Hypothesis 1

The first hypothesis, as stated in Chapter I, was: there is no significant difference between the pretest means of the experimental and control groups at the 95% confidence level. That is,

\[ H_0: \mu_{E,pre} = \mu_{C,pre} \]

\[ H_a: \mu_{E,pre} \neq \mu_{C,pre} \]
The purpose of this hypothesis is to confirm that the groups were not initially different in prior knowledge of computers, assembly language, and the C language. The overall mean of the pretest scores for the students in the experimental group was 17.80, and the mean for the control group was 15.79. An analysis of variance revealed that the difference between the two means was not significant, $F(1,47) = 3.22$, $p = .079$ (Table 1). Hypothesis 1 was not rejected. In other words, the random placement of subjects in the experimental and control groups created samples with relatively equal prior knowledge of computers, assembly language, and the C language.

<table>
<thead>
<tr>
<th>Source</th>
<th>SS</th>
<th>DF</th>
<th>MS</th>
<th>F</th>
<th>Prob. &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main Effects</td>
<td>49.39</td>
<td>1</td>
<td>49.39</td>
<td>3.22</td>
<td>.079</td>
</tr>
<tr>
<td>Residual</td>
<td>719.96</td>
<td>47</td>
<td>15.32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>769.35</td>
<td>48</td>
<td>16.03</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

An analysis of variance was used to examine whether there was a significant difference between the three classes on the pretest. The means for students in IEDT 216A, IEDT 216B, and IEDT 316 were 16.15, 15.13, and 19.57, respectively. These means were significantly different, $F(2,46) = 5.93$, $p = .005$ (Table 2), indicating that students who enrolled in the successive class levels (IEDT 216 and IEDT 316) had
significant different levels of prior knowledge of computers, assembly language, and the C language. In other words, students of IEDT 316 had more computer knowledge than students of IEDT 216.

Hypothesis 2

The second hypothesis, as stated in Chapter I, was: there is no significant difference between the experimental and control group posttest 1 means, adjusted for effects of the pretest at the 95% confidence level. That is,

\[ H_0: \mu_{E,\text{post}1} = \mu_{C,\text{post}1} \]

\[ H_a: \mu_{E,\text{post}1} \neq \mu_{C,\text{post}1} \]

This hypothesis examines the differences between the treatment groups regarding their knowledge of concepts emphasized in assembly language after four weeks of instruction. The pretest scores were used as a covariate to reduce the error variance term of the analysis by the degree to which the posttest 1 scores are correlated with the pretest scores.

In posttest 1, the items used to examine the acquired
knowledge of assembly language were items 11 - 36 (see APPENDIX B). These items were treated as a measure of knowledge acquisition from assembly language instruction. Means and standard deviations for the assembly language on posttest 1 are summarized in four categories in Table 3. For combined within class the statistics were 14.50 (3.20), 16.73 (4.04), and 16.36 (2.98) for IEDT 216A, IEDT 216B, and IEDT 316, respectively. When combined by group the statistics were 16.06 (3.99), 14.82 (3.38), 15.67 (2.88), and 16.87 (3.14) for the control and experimental groups of IEDT 216 and IEDT 316 respectively. Means and standard deviations were 15.9 (3.69) and 15.48 (3.38) for overall control and experimental groups respectively. The total was 15.71 (3.51). The proportion of correct responses on the selected 26 items was approximately .60 (15.71 correct on average, out of 26).

Table 4 then shows the means and standard deviations for all 51 items on posttest 1. The class statistics were 33.55 (4.10), 34.87 (5.45), and 35.79 (5.18) for IEDT 216A, IEDT 216B, and IEDT 316, respectively. The group statistics were 34.28 (5.07), 33.94 (4.41), 33.50 (6.02), and 37.50 (4.00) for the control and experimental groups of IEDT 216 and IEDT 316 respectively. Overall control and experimental groups statistics were 34.08 (5.20) and 35.08 (4.53). The total was 34.59 (4.84). The proportion of correct responses was approximately .68 (34.59 correct on average, out of 51).
Table 3. Means and Standard Deviations for the Assembly Language Items on Posttest 1

<table>
<thead>
<tr>
<th>Variable</th>
<th>N</th>
<th>Mean</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Combined within Class</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IEDT 216A</td>
<td>20</td>
<td>14.50</td>
<td>3.20</td>
</tr>
<tr>
<td>IEDT 216B</td>
<td>15</td>
<td>16.73</td>
<td>4.04</td>
</tr>
<tr>
<td>Total IEDT 216</td>
<td>35</td>
<td>15.46</td>
<td>3.70</td>
</tr>
<tr>
<td>Total IEDT 316</td>
<td>14</td>
<td>16.36</td>
<td>2.98</td>
</tr>
<tr>
<td>Combined by Group</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IEDT 216</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>18</td>
<td>16.06</td>
<td>3.99</td>
</tr>
<tr>
<td>Experiment</td>
<td>17</td>
<td>14.82</td>
<td>3.38</td>
</tr>
<tr>
<td>IEDT 316</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>6</td>
<td>15.67</td>
<td>2.88</td>
</tr>
<tr>
<td>Experiment</td>
<td>8</td>
<td>16.87</td>
<td>3.14</td>
</tr>
<tr>
<td>By Group</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>24</td>
<td>15.96</td>
<td>3.69</td>
</tr>
<tr>
<td>Experiment</td>
<td>25</td>
<td>15.48</td>
<td>3.38</td>
</tr>
<tr>
<td>Total</td>
<td>49</td>
<td>15.71</td>
<td>3.51</td>
</tr>
</tbody>
</table>

Table 5 shows a one-way analysis of covariance on the subscore for assembly language items. There was not a significant group main effect, $F(1,46) = .91, p = .346$. This indicates subjects' performance was not affected by the instructional method (traditional lecture vs. computer-based instruction) in which they participated. Therefore, Hypothesis 2 was not rejected. It can be concluded that assembly language taught through computer-based instruction is as effective as traditional classroom instruction. This answers the first question of the study raised in Chapter I.
Table 4. Means and Standard Deviations for the Total Items on Posttest 1

<table>
<thead>
<tr>
<th>Variable</th>
<th>N</th>
<th>Mean</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Combined</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>within Class</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IEDT 216 A</td>
<td>20</td>
<td>33.55</td>
<td>4.10</td>
</tr>
<tr>
<td>IEDT 216 B</td>
<td>15</td>
<td>34.87</td>
<td>5.45</td>
</tr>
<tr>
<td>Total IEDT 216</td>
<td>35</td>
<td>34.11</td>
<td>4.70</td>
</tr>
<tr>
<td>Total IEDT 316</td>
<td>14</td>
<td>35.79</td>
<td>5.18</td>
</tr>
<tr>
<td><strong>Combined</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>by Group</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IEDT 216</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>18</td>
<td>34.28</td>
<td>5.07</td>
</tr>
<tr>
<td>Experiment</td>
<td>17</td>
<td>33.94</td>
<td>4.41</td>
</tr>
<tr>
<td>IEDT 316</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>6</td>
<td>33.50</td>
<td>6.02</td>
</tr>
<tr>
<td>Experiment</td>
<td>8</td>
<td>37.50</td>
<td>4.00</td>
</tr>
<tr>
<td><strong>By Group</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>24</td>
<td>34.08</td>
<td>5.20</td>
</tr>
<tr>
<td>Experiment</td>
<td>25</td>
<td>35.08</td>
<td>4.53</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>49</td>
<td>34.59</td>
<td>4.84</td>
</tr>
</tbody>
</table>

Furthermore, to better interpret the source of interaction between group and class, a second analysis was performed, using a two-way analysis of covariance. No major group x class interaction was observed, $F(2,32) = .79, p = .463$ (Table 6). This indicates that there was not a significantly different level of assembly language concepts between groups and classes.

Finally, the sources of variation for all 51 items were analyzed using ANCOVA. Table 7 shows a one-way analysis of covariance on correct answers for all 51 items. There was not a significant group main effect observed, $F(1,46) = .02, p = \ldots$
Table 5. One-way ANCOVA Table for the Assembly Language Items on Posttest 1

<table>
<thead>
<tr>
<th>Source</th>
<th>SS</th>
<th>DF</th>
<th>MS</th>
<th>F</th>
<th>Prob. &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Covariates</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRETEST</td>
<td>34.72</td>
<td>1</td>
<td>34.72</td>
<td>2.92</td>
<td>.094</td>
</tr>
<tr>
<td>Main Effects</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP</td>
<td>10.72</td>
<td>1</td>
<td>10.72</td>
<td>.91</td>
<td>.346</td>
</tr>
<tr>
<td>Explained</td>
<td>45.43</td>
<td>2</td>
<td>22.72</td>
<td>1.92</td>
<td>.158</td>
</tr>
<tr>
<td>Residual</td>
<td>544.57</td>
<td>46</td>
<td>11.83</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>590.00</td>
<td>48</td>
<td>12.29</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6. Two-way ANCOVA Table for the Assembly Language Items on Posttest 1

<table>
<thead>
<tr>
<th>Source</th>
<th>SS</th>
<th>DF</th>
<th>MS</th>
<th>F</th>
<th>Prob. &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Covariates</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRETEST</td>
<td>20.02</td>
<td>1</td>
<td>20.02</td>
<td>1.61</td>
<td>.213</td>
</tr>
<tr>
<td>Main Effects</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP</td>
<td>4.62</td>
<td>1</td>
<td>4.62</td>
<td>.37</td>
<td>.546</td>
</tr>
<tr>
<td>CLASS</td>
<td>21.13</td>
<td>2</td>
<td>10.57</td>
<td>.85</td>
<td>.436</td>
</tr>
<tr>
<td>Interactions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP x CLASS</td>
<td>19.58</td>
<td>2</td>
<td>9.79</td>
<td>.79</td>
<td>.463</td>
</tr>
<tr>
<td>Explained</td>
<td>65.35</td>
<td>6</td>
<td>10.89</td>
<td>.88</td>
<td>.522</td>
</tr>
<tr>
<td>Residual</td>
<td>397.01</td>
<td>32</td>
<td>12.41</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>462.36</td>
<td>38</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

.900. This also indicates that students' performance on overall computer concepts was not affected by the instructional method. Table 8 shows the results of a two-way analysis of covariance. Similarly, there was no major group x
class interaction for the total items, \( F(2,32) = 1.26, p = .298 \). This indicates that there was not a significantly different level of computer concepts between groups and classes.

Table 7. One-way ANCOVA Table for the Total Items on Posttest 1

<table>
<thead>
<tr>
<th>Source</th>
<th>SS</th>
<th>DF</th>
<th>MS</th>
<th>F</th>
<th>Prob. &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Covariates</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRETEST</td>
<td>135.04</td>
<td>1</td>
<td>135.04</td>
<td>6.27</td>
<td>.016</td>
</tr>
<tr>
<td>Main Effects</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP</td>
<td>.32</td>
<td>1</td>
<td>.32</td>
<td>.02</td>
<td>.900</td>
</tr>
<tr>
<td>Explained</td>
<td>135.36</td>
<td>2</td>
<td>67.68</td>
<td>3.14</td>
<td>.053</td>
</tr>
<tr>
<td>Residual</td>
<td>990.48</td>
<td>46</td>
<td>21.53</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>1125.84</td>
<td>48</td>
<td>23.46</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 8. Two-way ANCOVA Table for the Total Items on Posttest 1

<table>
<thead>
<tr>
<th>Source</th>
<th>SS</th>
<th>DF</th>
<th>MS</th>
<th>F</th>
<th>Prob. &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Covariates</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRETEST</td>
<td>44.62</td>
<td>1</td>
<td>44.62</td>
<td>1.91</td>
<td>.177</td>
</tr>
<tr>
<td>Main Effects</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP</td>
<td>4.73</td>
<td>1</td>
<td>4.73</td>
<td>.20</td>
<td>.656</td>
</tr>
<tr>
<td>CLASS</td>
<td>10.05</td>
<td>2</td>
<td>5.03</td>
<td>.22</td>
<td>.808</td>
</tr>
<tr>
<td>Interactions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP x CLASS</td>
<td>58.73</td>
<td>2</td>
<td>29.37</td>
<td>1.26</td>
<td>.298</td>
</tr>
<tr>
<td>Explained</td>
<td>118.13</td>
<td>6</td>
<td>19.69</td>
<td>.84</td>
<td>.547</td>
</tr>
<tr>
<td>Residual</td>
<td>747.61</td>
<td>32</td>
<td>23.36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>865.74</td>
<td>38</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Hypothesis 3

The third hypothesis, as stated in Chapter I, was: there is no significant difference between the experimental and control group posttest 2 means, adjusted for effects of the pretest and posttest 1 scores at the 95% confidence level. That is,

\[ H_0: \mu_{E,post2} = \mu_{C,post2} \]

\[ H_a: \mu_{E,post2} \neq \mu_{C,post2} \]

This hypothesis examines the achievement scores on C language concepts. The scores are adjusted by the covariates of pretest and posttest 1 scores to reduce the error variance.

For IEDT 216, the total mean score was 14.73 out of 24 points and the standard deviation was 4.67. The means (and standard deviations) of the treatment groups that received the different teaching modes in assembly language were 14.07 (5.13) and 15.40 (4.22) for the control group and experimental group, respectively (Table 9). These means were adjusted by the covariates of pretest and posttest 1. The adjustment showed that there was not a significant group main effect, \( F(1,26) = .79, p = .384 \) (Table 10). This indicates that the assembly language instructional method students received had no significant effect on the posttest 2 (C language) score.

A two-way ANCOVA was used to examine whether there was an interaction between class and group. Table 11 shows that there was also no major group x class interaction, \( F(1,18) = \)
Table 9. Means and Standard Deviations for IEDT 216 on Posttest 2 (N = 30)

<table>
<thead>
<tr>
<th>Group</th>
<th>N</th>
<th>Mean</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>15</td>
<td>14.07</td>
<td>5.13</td>
</tr>
<tr>
<td>Experiment</td>
<td>15</td>
<td>15.40</td>
<td>4.22</td>
</tr>
<tr>
<td>Total</td>
<td>30</td>
<td>14.73</td>
<td>4.67</td>
</tr>
</tbody>
</table>

Table 10. One-way ANCOVA Table for IEDT 216 on Posttest 2 by the Covariates of Pretest and Posttest 1

<table>
<thead>
<tr>
<th>Source</th>
<th>SS</th>
<th>DF</th>
<th>MS</th>
<th>F</th>
<th>Prob. &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Covariates</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRETEST</td>
<td>29.49</td>
<td>1</td>
<td>29.49</td>
<td>1.74</td>
<td>.199</td>
</tr>
<tr>
<td>POSTTEST 1</td>
<td>147.53</td>
<td>1</td>
<td>147.53</td>
<td>8.69</td>
<td>.007**</td>
</tr>
<tr>
<td>Main Effects</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP</td>
<td>13.33</td>
<td>1</td>
<td>13.33</td>
<td>.79</td>
<td>.384</td>
</tr>
<tr>
<td>Explained</td>
<td>190.35</td>
<td>3</td>
<td>63.45</td>
<td>3.74</td>
<td>.023*</td>
</tr>
<tr>
<td>Residual</td>
<td>441.52</td>
<td>26</td>
<td>16.98</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>631.87</td>
<td>29</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* p < .05
** p < .01

1.46, p = .243. This indicates that there was no significant difference of C language knowledge between groups and classes.

For IEDT 316, the total mean score was 21.50 out of 35 points and the standard deviation was 3.58. The means of the treatment groups that received the different teaching modes in assembly language were 20.80 (2.17) and 22.00 (4.43) for the control group and experimental group, respectively. These means were adjusted by the covariates of pretest and posttest 1.
Table 11. Two-way ANCOVA Table for IEDT 216 on Posttest 2 by the Covariates of Pretest and Posttest 1

<table>
<thead>
<tr>
<th>Source</th>
<th>SS</th>
<th>DF</th>
<th>MS</th>
<th>F</th>
<th>Prob. &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Covariates</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRETEST</td>
<td>49.50</td>
<td>1</td>
<td>49.50</td>
<td>3.52</td>
<td>.077</td>
</tr>
<tr>
<td>POSTTEST 1</td>
<td>81.07</td>
<td>1</td>
<td>81.07</td>
<td>5.76</td>
<td>.028*</td>
</tr>
<tr>
<td>Main Effects</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP</td>
<td>28.90</td>
<td>1</td>
<td>28.90</td>
<td>2.05</td>
<td>.169</td>
</tr>
<tr>
<td>CLASS</td>
<td>16.35</td>
<td>1</td>
<td>16.35</td>
<td>1.16</td>
<td>.296</td>
</tr>
<tr>
<td>Interactions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP x CLASS</td>
<td>20.53</td>
<td>1</td>
<td>20.53</td>
<td>1.46</td>
<td>.243</td>
</tr>
<tr>
<td>Explained</td>
<td>196.35</td>
<td>5</td>
<td>39.27</td>
<td>2.79</td>
<td>.049*</td>
</tr>
<tr>
<td>Residual</td>
<td>253.48</td>
<td>18</td>
<td>14.08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>449.83</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* p < .05

Table 12. One-way ANCOVA Table for IEDT 316 on Posttest 2 by the Covariates of Pretest and Posttest 1

<table>
<thead>
<tr>
<th>Source</th>
<th>SS</th>
<th>DF</th>
<th>MS</th>
<th>F</th>
<th>Prob. &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Covariates</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRETEST</td>
<td>13.11</td>
<td>1</td>
<td>13.11</td>
<td>1.17</td>
<td>.310</td>
</tr>
<tr>
<td>POSTTEST 1</td>
<td>34.29</td>
<td>1</td>
<td>34.29</td>
<td>3.07</td>
<td>.118</td>
</tr>
<tr>
<td>Main Effects</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP</td>
<td>4.20</td>
<td>1</td>
<td>4.20</td>
<td>.38</td>
<td>.557</td>
</tr>
<tr>
<td>Explained</td>
<td>51.60</td>
<td>3</td>
<td>17.20</td>
<td>1.54</td>
<td>.278</td>
</tr>
<tr>
<td>Residual</td>
<td>89.40</td>
<td>8</td>
<td>11.18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>141.00</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The results showed that there was not a significant main group effect $F(1,8) = .38$, $p = .557$ (Table 12), indicating that the instructional method students received had no significant effect on the posttest 2 (C language) score.

These results also answer the second question of the study raised in Chapter I. That is, the concepts that were learned in assembly language transferred to C language to the same degree for students who learned assembly language through the computer-based instruction as for those who learned assembly language through traditional lecture/practice method.

Hypothesis 4

The fourth hypothesis, as stated in Chapter I, was: the product-moment correlation between pretest and posttest 1 does not differ from 0 at the 95% confidence level. That is,

\[
H_0: \rho_{\text{pre,post}1} = 0
\]

\[
H_a: \rho_{\text{pre,post}1} \neq 0
\]

This hypothesis examines the degree to which the pretest is related to the achievement of assembly language concepts and thereby contributes to the reduction of the error term in hypothesis 2 above.

The product-moment correlation coefficient was computed for all subjects. The value of the correlation was $r = .346$, $p < .05$. The pretest measures correlated positively with posttest 1. Thus, Hypothesis 4 was rejected. This result for Hypothesis 4 corresponds to the result for Hypothesis 2, as
shown in Table 7. The significance level for the pretest covariate was \( p = .016 \).

**Hypothesis 5**

The fifth hypothesis, as stated in Chapter I, was: the multiple correlation of posttest 2 with the variables pretest and posttest 1 does not differ from 0 at the 95% confidence level. That is,

\[
H_0: \; \rho_{\text{post2};\text{pre},\text{post1}} = 0 \\
H_a: \; \rho_{\text{post2};\text{pre},\text{post1}} \neq 0
\]

This hypothesis examines the squared multiple correlation of the achievement test scores on C language concepts regressed on both the pretest and assembly language posttest 1. The statistical analysis for evaluating this hypotheses assesses the proportion of posttest 2 variance in common with pretest and posttest 1 variance. The analysis also indicates the degree to which the error term in hypothesis 3 was reduced by the use of the covariates.

For IEDT 216, the squared multiple correlation was .288, \( F = 5.47, \; p = .010 \) (Table 13), indicating that student performance on posttest 2 was correlated well with the pretest and with posttest 1. For IEDT 316, the squared multiple correlation was .365, \( F = 2.59, \; p = .129 \) (Table 14). This result indicates that student performance on posttest 2 was
Table 13. Regression Analysis for IEDT 216 on Posttest 2

<table>
<thead>
<tr>
<th>Source</th>
<th>SS</th>
<th>DF</th>
<th>MS</th>
<th>F</th>
<th>Prob. &gt; F</th>
<th>R²</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression</td>
<td>182.16</td>
<td>2</td>
<td>91.08</td>
<td>5.47</td>
<td>.010**</td>
<td>.288</td>
</tr>
<tr>
<td>Residual</td>
<td>449.71</td>
<td>27</td>
<td>16.66</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>631.87</td>
<td>29</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

** p < .01

Table 14. Regression Analysis for IEDT 316 on Posttest 2

<table>
<thead>
<tr>
<th>Source</th>
<th>SS</th>
<th>DF</th>
<th>MS</th>
<th>F</th>
<th>Prob. &gt; F</th>
<th>R²</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression</td>
<td>51.52</td>
<td>2</td>
<td>25.76</td>
<td>2.59</td>
<td>.129</td>
<td>.365</td>
</tr>
<tr>
<td>Residual</td>
<td>89.48</td>
<td>9</td>
<td>9.94</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>141.00</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

correlated poorly with the pretest and with posttest 1. In other words, for IEDT 216, Hypothesis 5 was rejected. Yet, for IEDT 316, Hypothesis 5 was not rejected. These results also correspond to the results for Hypothesis 3, as shown in Table 10 (p = .007 for posttest 1 covariate) and Table 12.
CHAPTER V. SUMMARY, DISCUSSION, RECOMMENDATIONS, AND CONCLUSIONS

Summary

The purpose of this study was twofold. The study was conducted to evaluate the effectiveness of using the computer-based program LEARNIT in teaching assembly language. It was also conducted to investigate whether learning assembly language was helpful in writing programs in the C language.

The population for this study consisted of 49 students: 35 students were in the Computer Application class (IEDT 216A and 216B); 14 students were in the Advanced Computer Application class (IEDT 316). These students were enrolled during the fall semester of 1993, in the Department of Industrial Education and Technology at Iowa State University.

Students in the three classes were randomly assigned to one of the two groups; 25 subjects and 24 subjects were in the experimental group and control group respectively. The research design contained 2 (group) x 3 (class) and 2 (group) x 2 (class) factorial design in posttest 1 and posttest 2 respectively.

Assembly language was taught during the first four weeks of the semester. The traditional lecture was presented to both groups during the first four hours. Then, the experimental group received instruction on assembly language using the computer-based program LEARNIT. The control group
received instruction on assembly language using the traditional instruction method. The assembly language topics taught to the two groups consisted of basic computer organization, arithmetic operations, branch operations, loop operations, subroutines, and program interrupts. The posttest 1 was administered to measure students' knowledge of computer organization and assembly language.

During the fifth through ninth weeks, the two groups learned C programming in the same environment using lecture and programming activities on computers. In the ninth week, posttest 2 was administered to measure the students' ability in C programming.

Based on the data collected and the analyses performed, the results of this study were:

1. Random placement of students in the experimental and control groups created samples of subjects relatively equal in prior knowledge of computers, assembly language, and the C language.

2. Students who enrolled in IEDT 316 had more computer knowledge than students enrolled in IEDT 216.

3. The subjects' performance on the assembly language subscore was not affected by the instructional method (traditional lecture vs. computer-based instruction) in which they participated.
4. No major group x class interaction was observed on posttest 1.

5. The subjects' performance on posttest 1 was related to their performance on the pretest.

6. The subjects' performance on C language was not related to a difference in mean levels of performance of assembly language between the experimental and control groups.

7. For IEDT 216, the subject's performance on C language was correlated strongly with the performance of assembly language. Yet, for IEDT 316, the relationship between assembly language and C language was not evident.

Discussion

This discussion is organized into three subsections: prior knowledge of students about computers, effects of LEARNIT on the performance of assembly language, and the relationship of C programming knowledge to assembly language. In the first subsection, the pretest that illustrates the prior knowledge of computers that students had is discussed. In the second subsection, the effects of LEARNIT are discussed related to the results of posttest 1. In addition, the items for which mean scores are quite high or quite low are
discussed. Lastly, the relationship of performance on assembly language and C language tests (posttest 2) is discussed.

Prior knowledge of students about assembly language

In the pretest, the mean of correct responses on all items was approximately 40 percent. The scores were much higher than the author expected. This result may be due to the use of simple items, and the fact that students had some previous knowledge about computers. The mean scores among the three classes were significantly different. IEDT 316 students got higher scores than IEDT 216 students. Most students in IEDT 316 have already taken IEDT 216 where they learned C language, so that they had more knowledge about computers than IEDT 216 students who rarely had programming experience. As Linn (1985) indicated, transfer could happen from one programming language to another, it may be possible that the IEDT 316 students had applied the knowledge of C language to understand the basic concepts of computers. However, the comparing of scores between the experimental and control groups revealed no significant difference.

Effects of LEARNIT on the performance of assembly language

Posttest 1 was used to test hypotheses 2 and 4. The results revealed that there were no reliable overall differences between control and experimental groups among the three classes. On the assembly language items the percentage
of correct responses was approximately 60 percent. This showed that students had some understanding of the assembly language, but did not understand many of the concepts inherent to the execution of programs using assembly language. The concepts covered were branch, loop, subroutine, and interrupt. The lack of understanding may be due to the inherent complexity of assembly language. Taking more time to cover assembly language may also improve students' learning.

In general, a few students in both groups showed some improvement when comparing their pretest scores with their posttest 1 scores. Others showed only a minor improvement in assembly language knowledge. It is interesting to note that five students in the experimental group had very high posttest scores on computer concepts and assembly language parts, whereas their pretest scores on these two parts were very low. In contrast, no students in the control group improved like those five students. The author speculates that although these five students had little prior knowledge about computers, the computer-based program LEARNIT was beneficial in helping them make a large advancement in comprehension of assembly language.

Some students really enjoyed working with the LEARNIT program. They came to the classroom to practice with the program much earlier than the regular class time. The interactive nature of LEARNIT seemed very appealing to these students. They were eager to learn assembly language and ask
questions.

However, some students showed only a small improvement in their scores. One reason may be due to the fact that their pretest scores were quite high compared with the other students. These students were distributed over both groups. Overall, the small improvement by these students may be due to the limitations of the measurement of the instruments (sometimes called "ceiling effect").

In addition, although the number of instructional topics were chosen as narrowly as possible, the amount of time to focus on each topic was limited. The fast pace made understanding a difficult task for students who had only a sketchy conceptual framework, or little concept of assembly language. One would expect a significant improvement in students' scores if they were allowed to have more time to learn and practice these topics. Acquisition of the basic concepts of program execution using assembly language may not be feasible in such a short period of time.

Another factor which may cause the students difficulty in understanding can be explained in two parts. For students in the control group, they saw each program demonstrated one after the other during the short class hours. Apart from the effect of fatigue, the students might have confused the runtime actions of different programs. In the experimental group, although students could see the program execution actions simulated on the screen, the effects were limited.
Some students looked at each lesson once only, and did not think in more detail how the actions happened in the computer. It is likely that they did not link the computer organization concepts with assembly language. One feature of CBI is that it takes into account self-regulated learning (Gisolfi, et al., 1993; Yano, et al. 1992; Vockell & Schwartz, 1992); however, it does not occur for every student. Some students benefitted from CBI, but others did not. Also, the lack of a compiler in LEARNIT limited the students writing their own programs to explore what would happen when they input the programs. Probably, only reading programs may not obtain the threshold of knowledge required for transfer.

Overall, the results of the statistical analyses show that LEARNIT may not help all students to clarify all the concepts inherent in program execution. Also, the results do not show what effect LEARNIT has on the student's conceptual model of program execution. Therefore, more detailed experiments are necessary in order to determine how the conceptual model changed after the demonstrations had been seen.

The item means (see APPENDIX F) corresponding to the proportion of correct responses are now discussed in more detail. Among 10 matching items, the proportion of correct responses for items 5, 7, 10 are very low. These three items and their means and standard deviations are presented in Table 15.
Table 15. The Means and Standard Deviations of Items 5, 7, and 10 on Posttest 1 (N = 49)

<table>
<thead>
<tr>
<th>Item</th>
<th>Mean</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>5. The register which often holds the base address offset of an array of data in memory.</td>
<td>.143</td>
<td>.354</td>
</tr>
<tr>
<td>7. The register which holds the offset of the data address when calling the DOS service to print a string.</td>
<td>.163</td>
<td>.373</td>
</tr>
<tr>
<td>10. The type of memory which must be frequently &quot;refreshed&quot; in order to retain its binary data.</td>
<td>.082</td>
<td>.277</td>
</tr>
</tbody>
</table>

For items 5 and 7, the correct answers are BP and DX respectively. In many cases, BP (base pointer) and DX (data register) have specified functions in the CPU. Students should probably have actual programming experiences using these registers in order to better understand their use. It is not surprising that the correct answer proportion was low, since the students did not have much time to practice their understanding.

For item 10, the correct answer is DYNAMIC. This item refers to the computer hardware design. Although the instructor had explained the characteristics of dynamic and static memories, obviously, students failed to grasp this concept. Perhaps, if students do not have basic concepts of circuit theory and computer hardware, it is not surprising that they are not familiar with characteristics of memory systems. They may have difficulty distinguishing between
Table 16. The Mean and Standard Deviation of Item 1 on Posttest 1 (N = 49)

<table>
<thead>
<tr>
<th>Item</th>
<th>Mean</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. memory which contains a &quot;boot&quot; program.</td>
<td>.878</td>
<td>.331</td>
</tr>
</tbody>
</table>

dynamic and static memories since the terms do not readily convey the differences in operation. Thus, it is suggested that students having knowledge of hardware circuits could better understand computer operations.

In contrast, item 1 seems to be an easy question. The mean score on this item was .878 (Table 16). It revealed that students have the basic concept of ROM (read only memory). Most students could distinguish the different characteristics between ROM and RAM (random access memory). Moreover, they understand that a ROM contains a "boot" program.

Of the 26 multiple choice items, the mean scores of items 14 and 35 were much lower than others. These two items are shown in Table 17. The correct answer to item 14 is the choice of "A." Twelve out of 49 students got the correct answer. It seems that students do not understand the calculation of an actual address which consists of a segment register and an offset address. In fact, the concept of an absolute address in Intel 8086/88 processors is not easy to understand, especially since students did not have actual experience in determining the absolute addresses in programs. It is suggested that this item should be abandoned when there
Table 17. The Means and Standard Deviations of Items 14 and 35 on Posttest 1 (N = 49)

<table>
<thead>
<tr>
<th>Item</th>
<th>Mean</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.</td>
<td>.225</td>
<td>.422</td>
</tr>
<tr>
<td>35.</td>
<td>.204</td>
<td>.407</td>
</tr>
</tbody>
</table>

14. To obtain the actual (absolute) address of a location in memory using the Intel segmented address mode, you must
A. shift the segment register four bits to the left and add the offset address.
B. shift the offset address four bits to the left and add the segment address.
C. shift the segment register address four bits to the right and add the offset.
D. shift the offset address four bits to the right and add the segment address.
E. add the segment address and the offset address and shift the answer left four bits.

35. The word "Message" in "Message DB 'Hello,world!', 13,10, '$'" is used as:
A. an address.
B. a variable (or variables).
C. a subroutine name.
D. a comment.

is only a short duration of assembly language instruction time.

The correct answer to item 35 of posttest 1 is also the choice "A." The mean score was .204 which was the lowest score among the assembly language items. If we examine pretest item 28 (mean = .122) (see APPENDIX E), the same item as item 35, we find that students had little knowledge gain in the use of a variable to substitute for an address. In this case, the variable name "Message" is used instead of the offset address stored in the register DX. This was not an
easy concept to understand even after students had practiced related programs with "debug."

However, it should be noted that 5 of 14 students in IEDT 316 had the correct answer; and all of them were in the experimental group. As a speculation, it may be that these students acquired this knowledge from the LEARNIT program. In the Program Interrupts lesson, in addition to presenting the concept of an interrupt, the screen also presents the contents of the memory addresses with ASCII code that displays a string. By stepping through the instructions, students could observe that the string codes are sequentially stored from the offset address stored in the DX register. They may have observed that the word "Message" in item 35 represented an address ("an address" is the correct answer of this item). The students in the control group did not appear to have gained this understanding. We may speculate why a difference was not observed when comparing experimental and control groups in IEDT 216. One possible explanation is that the students in IEDT 216 have had little experience with the concept of a label to represent a variable. In contrast, IEDT 316 students have had more programming experience using variables. When IEDT 316 students look at the Program Interrupts lesson in LEARNIT, they are more likely to understand the use of the label "Message" and its function. If so it may be concluded that transfer occurred between two programming languages with respect to the use of labels to
represent a value. In other words, because the students acquired the concept of variable labels in C programming first, when they look at how an assembly language program displays a string, they better understand that the string is to be stored in the memory locations referenced by a label. Items 16, 19, 21, 28, 29, and 32 over assembly language (Table 18) seem to be easy questions for the students. The high mean scores of items 16 and 19 shows that most students understood how to convert numbers from one base to another. The remaining four items (items 21, 28, 29, and 32) were used to measure the knowledge of assembly language instructions. Apparently, these instruction mnemonics facilitated students' understanding the purpose of the instructions.

Overall, students understood the DOS commands very well. It may be inferred that because computers are popular nowadays most students have had experience using computer programs (e.g., games). They appear to understand the functions of most DOS commands. However, the mean of item 46 was only .694 (Table 19). It appears that most students understood the use of <CTRL><ALT><DEL> to reboot the system, while 31% did not. Since the system can be turned off and then back on to reboot, the use of key combinations to reboot may not have been "important" enough to remember.
Table 18. The Means and Standard Deviations of Items 16, 19, 21, 28, 29, and 32 on Posttest 1 (N = 49)

<table>
<thead>
<tr>
<th>Item</th>
<th>Mean</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>16. The binary equivalent of the hexadecimal value AE is</td>
<td>.980</td>
<td>.143</td>
</tr>
<tr>
<td>A. 11001110</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B. 10011101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C. 10101011</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D. 10101110</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E. 11101010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19. The decimal equivalent of the hexadecimal value 1B is</td>
<td>.878</td>
<td>.331</td>
</tr>
<tr>
<td>A. 27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B. 13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C. 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D. 21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E. 1,611</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21. &quot;JA&quot; is</td>
<td>.816</td>
<td>.391</td>
</tr>
<tr>
<td>A. an unconditional jump instruction.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B. a conditional jump instruction.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C. a branch instruction.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D. an instruction to call a subroutine.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E. a comparison instruction.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28. The function of the instruction &quot;CMP&quot; is to:</td>
<td>.918</td>
<td>.277</td>
</tr>
<tr>
<td>A. increase the contents of CX.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B. decrease the contents of CX.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C. test whether CX = 0.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D. jump to a label.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E. store a new value in CX.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29. Which instruction is used to decrease a counter by 1?</td>
<td>.980</td>
<td>.143</td>
</tr>
<tr>
<td>A. CMP CX,0.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B. DEC CX.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C. JE OVER.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D. JUMP COUNT.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E. ADD AL,2.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32. The instruction &quot;CALL&quot; means to:</td>
<td>.898</td>
<td>.306</td>
</tr>
<tr>
<td>A. do a loop manipulation.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B. do an arithmetic operation.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C. interrupt to the DOS system.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D. call a subroutine.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E. none of the above.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 19. The Mean and Standard Deviation of Item 46 on Posttest 1 (N = 49)

<table>
<thead>
<tr>
<th>Item</th>
<th>Mean</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>46. Reboots the system.</td>
<td>.694</td>
<td>.466</td>
</tr>
</tbody>
</table>

The relationship of C programming knowledge to assembly language

There were no differences observed between experimental and control groups on posttest 2 over C language concepts. The different methods of instruction in assembly language had no measurable effect on C language achievement. Some past researchers have documented the transfer of a programming language to either problem solving or another language (Feurzeig, et al., 1981; Linn 1985; Little & Smotherman, 1988; Nickerson, 1988; Palumbo, 1990; Palumbo & Reed, 1991). In this study, only the method of learning the first language was tested. This study could be expanded to include groups which have and do not have assembly language prior to C language instruction. Unfortunately, there were not enough subjects available to examine the transfer question in this study.

There are some factors to consider in teaching these two programming languages. The first factor is that in order to measure the effectiveness of transfer on learning, both languages need more instructional time than was available in this study. Our experience in teaching assembly and C languages is that both include complex concepts that require
extensive practice, reading, experimentation, and thinking time.

The second factor that must be considered is the selected population. In this study, the population from which the sample was drawn was all of the students who enrolled in these three classes. The attitudes of students toward learning programming probably varied. Some students might not be willing to do their best in learning programming. This motivation factor may play a crucial role in understanding assembly language and the transfer to C language learning. A measure of interest/motivation would make a reasonable additional covariate, along with aptitude for a replicate study.

The third factor to be considered is the domain of measurement. Because of many concepts to be tested, only a few items were used to judge the knowledge in each area. Therefore, either the number of topics covered must be narrower or the number of questions on the tests must be greater to determine students' achievement more accurately. Based on the three factors stated above, the sensitivity to differential effects of instruction method for assembly language knowledge on C language achievement would be increased. Replication over more subjects would also increase sensitivity to possible effects.

However, it must be noted that, for IEDT 216, the performance on C language was positively correlated with the
effects of learning assembly language, whereas the effects were not evident for the IEDT 316 students. These results seem to provide important information that the effectiveness of assembly language learning may depend on when it is administered within an instructional sequence of C language (IEDT 216 and IEDT 316). As a speculation, learning assembly language may have provided the IEDT 216 students with meaningful knowledge of computer program execution. This schema might have served to organized C language concepts which were learned later. That is to say, students' performance on C language was enhanced by previous experience with the complex, abstract concepts of computer organization which related to C programming. This influence is greatest for meaningful learning where existing knowledge often serves as an anchoring device or schema for learning new knowledge (Brant, et al., 1991).

In contrast, the IEDT 316 student already had experiences of C programming. Undoubtedly, they had a little knowledge of computer organization and program execution. The assembly language instruction did not provide much motivation for these students. In other words, the assembly language materials did not provide new concepts but may have activated relevant knowledge which already existed within the students' cognitive structure. Therefore, it may be suggested that if the students already had their own meaningful "models" for C language, the effectiveness of assembly language instruction
was not sufficient to produce measurable transfer.

Recommendations

Based on the results of this study, the following recommendations are made for future research:

1. If this study is replicated, the experimental time needs to be extended. The learners should have sufficient time not only to practice the LEARNIT lessons, but also to learn the C language.

2. Future research in the transfer of assembly language should include the evaluation of student protocols of writing programs in the C language. The process students use in program development may yield more precise information about the effectiveness of LEARNIT.

3. More exercises should be involved in the computer-based program. Thus, the learner could choose the exercises which are appropriate to his/her ability.

4. The improved future generation of LEARNIT should be able to go to DOS in order to execute any assembly language programs using "debug" as well as "assembler" files. Thus, the program would provide a basis for student experimentation to enhance learning.
5. An alternative of the second recommendation stated above is to use another authoring system which supports use of available compilers. It should permit the learners to input any programming instructions and data to develop programming skills. The result would be a simulation rather than simply a tutorial program and would provide activities closer to true programming activity.

6. The computer-based program should be designed to record student responses, choices, errors, time, and achievement to provide a basis for further modification and improvement.

7. Future research will be necessary to explore which lessons of LEARNIT are most helpful in subsequent achievement in the C language.

8. Future research could compare a program using static screens, and with the animated features of LEARNIT.

9. The computer-based program could be designed to teach C statements first and from these to generate assembly code to teach assembly concepts concurrently with associated hardware concepts such as registers, memory, interrupts, etc.
10. Future research should expand the sample size used in the investigation; also, the use of assembly language instruction before the C language instruction should be compared with the use of assembly language following the C language.

Conclusions

The purpose of teaching assembly language in IEDT 216 and IEDT 316 was to introduce students to the underlying mechanization of programs, to facilitate learning the C language. Indeed, an understanding of computers and how they relate to the abstractions of programs is a vital and challenging part of both classes. The assembly language course should focus on teaching the concepts pertinent to the fetching and executing of instructions in both the abstract and the real sense, the transition of the instructions from human readable language to the machine language, and how the computers operate in an effective manner. Accordingly, the concepts of assembly language must be presented in a straightforward way to foster learning (Little & Smotherman, 1988).

Based on the viewpoint stated above, a computer-based program LEARNIT was designed to examine its effectiveness for learning assembly language. Furthermore, this study compared student achievement in learning C language following assembly language instruction by two methods.
Initially, this study was expected to strengthen the concepts of C programming. It was hoped that LEARNIT would provide a more efficient method to acquire concepts subsequently useful in C programming. Several approaches for investigating student behavior in writing programs, such as protocol analysis and problem solving (e.g., Hooper, 1986) have been used to measure the relationship between two programming languages. This study measured achievement in general concepts of C programming in a manner representative for accessing course progress in IEDT 216 and IEDT 316.

Many researchers have examined the potential that computer-based instruction offers as a vehicle for acquiring higher cognitive skills. The results of this experiment provided evidence that learning assembly language through the computer-based program LEARNIT was as effective as using traditional instruction. This finding suggests that computer-based instruction of this type provides an additional option for learning programming.

Research has indicated that learning assembly language could help programming language learning (e.g., Donahue, 1988). This study suggests that assembly language taught before the C language learning is helpful. However, this potential is not realized after the learner has some experience of C programming.

It may be concluded that the sample size was too small (30 subjects were in IEDT 216 and 12 subjects were in IEDT
316) to measure effectively. In addition, the short duration for this experiment was another factor.

It may also be concluded that because of the number of concepts to be covered, there is a lack of consensus of transfer between the cognitive outcomes on learning assembly language and C language at this time. However, it is hoped that the outcomes of learning assembly language have positively influenced (albeit unobtrusively and imperceptibly) C language learning. After all, education is not necessarily a linear phenomenon. The outcomes may be seen later.
BIBLIOGRAPHY


Hooper, E. J. (1986). Using programming protocols to investigate the effects of manipulative computer models on student learning. Doctoral dissertation, Iowa State University, Ames, IA.


Tzeng, O. J. L., & Wang, W. S. Y. (1983). The first two R's: The way different languages reduce speech to script affects how visual information is processed in the brain, American Scientist, 71, 238-143.


ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to Dr. William Miller, my major professor, who assumed that I would indeed develop the computer-based assembly language program. His pressure pushed me to complete the program LEARNIT. Also, his guidance and endless support through the study provided me with encouragement and strongly influenced my thinking during my graduate studies.

I am especially grateful to Dr. Rex Thomas. The reading of his computer-based program COMPOPS gave me the idea from which the computer-based program grew. In addition, his unique insights into the program design and research were invaluable for this study.

Appreciation is also expressed to the other members of my committee, Dr. Mack Shelley, Dr. John Beno, and Dr. Larry Bradshaw, who have provided me with a great deal of wise words during this study.

Gratitude is also extended to the students who participated in the experiment. Without their participation, this study could not have been completed.

Finally, I would like to express the deepest appreciation to my family, who have been a constant source of love and support in my life.
APPENDIX A. PRETEST
PRETEST OVER COMPUTER CONCEPTS

DIRECTIONS: Select the letter of the choice for each statement below which best completes the statement or answers the question. Record the letter of your choice in the blank in front of the item number.

Computer Organization

1. A typical computer should include four primary parts:
   (A) CPU, registers, memory, printer.
   (B) CPU, memory, input, output.
   (C) CPU, ROM, RAM, input.
   (D) ROM, RAM, PROM, EPROM.

2. Which of the following statements about a CPU is false?
   (A) The CPU can decode the instructions from a program stored in the memory.
   (B) The CPU can perform arithmetic computations, but it can not perform logical computations.
   (C) The CPU can perform logical computations.
   (D) The CPU controls the operations of the whole systems.

3. Which of the following statements is true about the general purpose registers?
   (A) They can store any temporary data.
   (B) They can not hold memory addresses.
   (C) They can not hold instructions.
   (D) They can handle control buses.

4. Which register's function is to indicate the address of the next instruction to be executed?
   (A) stack pointer.
   (B) instruction register.
   (C) instruction pointer (program counter).
   (D) index register.

5. Flag values are found in the status register. Which of the following statements about flags is false?
   (A) Flags are divided into conditional flags and control flags.
   (B) If the result of an addition manipulation is zero, the zero flag (ZF) would be set to 1.
   (C) In general, a conditional jump in a program is recognized by some flags' status.
   (D) A programmer can decide the conditional flags' status.
6. If a computer's address bus has 20 bits, both of the EA (effective address) and the segment address are 16 bits. Given the EA of a data is "1000", and the data segment (DS) = 490B, what is the actual address of the data?
(A) 590B0.
(B) 500B0.
(C) 490C0.
(D) 4A0B0.

7. Which of the following statements about the "stack" is false?
(A) The stack occupies memory.
(B) The instructions "PUSH" and "POP" are used to operate the stack.
(C) We can access any part of the stack.
(D) We cannot access any part of the stack except the top.

8. If a microprocessor has a 20-bit address, then the memory size is:
(A) 20 bytes.
(B) $2^{20}$ bytes.
(C) $2^2$ bytes.
(D) $20^2$ bytes.

9. Which language has a one-to-one correspondence with machine language?
(A) assembly language.
(B) BASIC.
(C) Pascal.
(D) Turbo C++

10. Which of the following statements is false?
(A) An assembly language instruction always consists of a mnemonic (also called op code) and operand (or operands).
(B) When you write a program, the program can be stored in either random-access memory (RAM) or read-only memory (ROM).
(C) After a program is stored in the memory, the individual instructions are fetched from the memory and executed.
(D) In general, interrupt is a way to transfer data between the CPU and peripheral devices.
Assembly Language

For questions 11 - 14, use the program listed below:

```
MOV AL, 3
MOV BL, 2
ADD AL, BL
ADD AL, 4
SUB AL, BL
```

11. After executing the instruction "MOV AL, 3", the contents of the AL register is:
   (A) 3.
   (B) 2.
   (C) 4.
   (D) it depends on the original contents of AL.

12. After executing the instruction "MOV BL, 2", the contents of the BL register is:
   (A) 5.
   (B) 2.
   (C) 3.
   (D) it depends on the original contents of BL.

13. After executing the instruction "ADD AL, BL", the contents of the AL register is:
   (A) 5.
   (B) 2.
   (C) 3.
   (D) it depends on the original contents of AL.

14. After executing the instruction "SUB AL, BL", the contents of the AL register is:
   (A) 4 - 2.
   (B) 3 + 2 - 4.
   (C) 3 + 2 + 4 - 2.
   (D) 3 + 2 - 2.
Questions 15 - 19 below use the following program used to compare two unsigned numbers:

```
MOV AL, [F000]
MOV CL, [F100]
CMP AL, CL
JA L2
L1: MOV [F200], CL
    JMP DONE
L2: MOV [F200], AL
DONE: HLT
```

15. "JA" is
   (A) an unconditional jump instruction.
   (B) a conditional jump instruction.
   (C) a branch instruction.
   (D) a call subroutine instruction.

16. The instruction "CMP AL, CL" is used to decide:
   (A) the contents of the AL register.
   (B) the contents of the BL register.
   (C) the contents of AL-BL.
   (D) the flags' status in order to execute next instruction "JA L2".

17. The instruction "JMP DONE" means:
   (A) an unconditional jump to the label name "DONE".
   (B) if AL > BL jump to the label name "DONE".
   (C) if BL > AL jump to the label name "DONE".
   (D) if AL = BL jump to the label name "DONE".

18. The purpose of this program is to:
   (A) transfer the contents in the memory to the registers.
   (B) transfer the contents in the register to the memory.
   (C) compare two numbers, then place the larger one in a memory location.

19. There are three labels: L1, L2, and DONE; they represent:
   (A) buses.
   (B) registers.
   (C) constants.
   (D) memory locations.
Use the following portion of a program to answer questions 20 - 24.

```
MOV AL, 5
MOV CX, 8
COUNT: ADD AL, 2
       CMP CX, 0
       JE OVER
       DEC CX
       JUMP COUNT
OVER: MOV BL, AL
```

20. The register CX is used as:
   (A) a counter.
   (B) an array.
   (C) an adder.
   (D) stack pointer.

21. The function of the instruction "CMP" is to:
   (A) increase the contents of CX.
   (B) decrease the contents of CX.
   (C) test whether CX = 0.
   (D) jump to a label.

22. Which instruction is used to decrease the counter by 1?
   (A) CMP CX, 0.
   (B) DEC CX.
   (C) JE OVER.
   (D) JUMP COUNT.

23. The purpose of these instructions is to do:
   (A) arithmetic manipulation.
   (B) branch manipulation.
   (C) loop manipulation.
   (D) subroutine manipulation.

24. The three instructions "CMP CX, 0", "DEC CX", and "JMP COUNT" could be replaced by which of the following instruction:
   (A) LOOP
   (B) LOOPE
   (C) LOOPZ
   (D) LOOPNZ.
Questions 25 - 27 are answered using the following program segment which contains a subroutine:

```
MOV CL,3
CALL SUM
INC AX
INT 20H
SUM:  MOV AL,CL
  LI:  CMP CL,1
       JZ L2
       DEC CL
       MUL CL
       JMP LI
L2:  RET
```

25. The instruction "CALL" means to:
   (A) do a loop manipulation.
   (B) do an arithmetic operation.
   (C) call a register.
   (D) call a subroutine.

26. The word "SUM" represents:
   (A) a calculation.
   (B) a subroutine name.
   (C) a label name.
   (D) a jump name.

27. Which of the following statements is false?
   (A) In this program, "SUM" represents a subroutine name, "LI" and "L2" represent two labels.
   (B) At the end of the program, the instruction "RET" is needed in order to return to the main program.
   (C) The purpose of the register CL in this program is to be an adder.
   (D) Before executing the subroutine, the beginning address of "INC AX" would be automatically pushed onto the stack.
Questions 28 - 30 are answered using the following program.

```
.MODEL small
.STACK 100H
.DATA
Message DB 'Hello,world!',13,10,'$'
.CODE
MOV AX,@data
MOV DS, AX
MOV AH, 9
MOV DX,OFFSET Message
INT 21H
MOV AH, 4CH
INT 21H
END
```

28. The word "Message" in "Message DB 'Hello,world!'", 13,10,'$' is used as:
(A) an address.
(B) a variable (or variables).
(C) a subroutine name.
(D) a comment.

29. The word "INT" is:
(A) an interrupt instruction.
(B) a instant data.
(C) a branch instruction.
(D) a call subroutine instruction.

30. Which of the following statements is true?
(A) "INT 21H" is an BIOS interrupt instruction.
(B) After executing this program, the screen would show the string "Hello, world!".
(C) The words "STACK", "DATA" and "CODE" are registers in CPU.
(D) Using the word "DB" defines a data field by one word in length.
C programming

---31. Which of the following statement is false?
(A) A program which is executable must be in the form of machine language.
(B) The text editor produces C or C++ source files.
(C) The linker produces .OBJ object files.
(D) The compiler is a part of the IDE.

---32. Which of the following statement is false?
(A) C programs consist of functions.
(B) The white space characters, except in strings are ignored by C.
(C) main() is the first function executed in a C program.
(D) Uppercase and lower case letter are the same in the C program.

---33. Which of the following expression is correct?
(A) sum = count = 0;
(B) int sum; count;
(C) int sum = count = 0;
(D) none of the above

---34. Which of the following is the address operator?
(A) %
(B) &
(C) *
(D) #

---35. Which of the following will be executed at least once?
(A) The body of an if statement.
(B) The body of a for loop.
(C) The body of a while loop.
(D) The body of a do while loop.

---36. What is the resulting value of S, in the following code segment?

```c
i=0; s=0;
while (i < 5)
{
    i = i+2;
    s += i;
}
```

(A) 2
(B) 4
(C) 6
(D) 12
---37. In the following code segment, the third statement will be executed how many times?

```c
for(i=0; i<3;i++)
    for(j=0;j<2;j++)
        printf("\n%d",i+2*j);
```

(A) 6
(B) 5
(C) 3
(D) 2

---38. When you pass an array as an argument to a function, what is actually passed?

(A) The value of all elements in the array
(B) The value of the first element in the array
(C) The size of the array
(D) The address of the first element in the array

---39. Which is more appropriate for printing out a string?

(A) gets()
(B) puts()
(C) printf()
(D) scanf()

---40. How many bytes is required for every character stored in the VGA VIDEO memory?

(A) 1
(B) 2
(C) 3
(D) 4
APPENDIX B. POSTTEST 1
DIRECTIONS: Read the directions for each section below and record your answer or choice in the area provided. Record both your name and social security number before turning in your test.

COMPUTER CONCEPTS:

MATCHING: Select the letter of the term or phrase on the right which most closely matches the term or phrase of the item on the left. Record the letter in the blank in front of the item.

--- 1. Memory which contains a "boot" program. A. AX  
     B. BX  
--- 2. The register which is automatically decremented by a loop instruction. C. CX  
     D. DX  
     E. CS  
--- 3. Memory that retains its information when the power is off but which can be reprogrammed with new information. F. DS  
     G. IP  
     H. SP  
     I. BP  
--- 4. The register used to indicate the interrupt service number of the DOS 21 interrupt. J. RAM  
     K. ROM  
     L. EPROM  
     M. CACHE  
--- 5. The register which often holds the base address offset of an array of data in memory. N. DYNAMIC  
     O. STATIC  
     P. STATUS  
     Q. BUS  
--- 6. Fast memory used to hold a block of code for rapid access by the CPU. R. AL  
     S. AH  
     T. BL  
--- 7. The register which holds the offset of the data address when calling the DOS service to print a string. U. BH  
     V. CL  
     W. CH  
     X. DL  
--- 8. The register whose offset points to the memory location of the next instruction to be executed. Y. DH  
--- 9. The register which holds the segment part of the address of an instruction to be executed.  
--- 10. The type of memory which must be frequently "refreshed" in order to retain its binary data.
ASSEMBLY LANGUAGE

MULTIPLE CHOICE: Select the letter of the ONE choice which best completes the statement or answers the question. Place the letter of your choice in the blank preceding the item number.

--- 11. A program contains the instruction CMP BX,CX. If the values in the registers are equal, which of the following instructions would cause a branch to a new instruction address?
   A. JNE
   B. JA
   C. JZ
   D. LOOP
   E. JNA

--- 12. Which of the following instruction is illegal, that is, would cause the DEBUG program to show "error message?"
   A. MOV [BX],AX
   B. MOV DS,120
   C. MOV AX,[BX]
   D. MOV AX,CS
   E. MOV AX,08

--- 13. Which status flag indicates whether the count of bits in a register of the AX register pair is even or odd?
   A. Zero
   B. Sign
   C. Overflow
   D. Auxiliary
   E. Parity

--- 14. To obtain the actual (absolute) address of a location in memory using the Intel segmented address mode, you must
   A. shift the segment register four bits to the left and add the offset address.
   B. shift the offset address four bits to the left and add the segment address.
   C. shift the segment register address four bits to the right and add the offset.
   D. shift the offset address four bits to the right and add the segment address.
   E. add the segment address and the offset address and shift the answer left four bits.
15. The value of "k" in an expression like "640k of memory" is
   A. 1000
   B. 1024
   C. 2048
   D. 4096
   E. 8192

16. The binary equivalent of the hexadecimal value AE is
   A. 11001110
   B. 10011101
   C. 10101011
   D. 10101110
   E. 11101010

17. The instruction "MOV AX,FF00" is assembled at location 100h in the computer. The hexadecimal equivalent of the binary values in memory locations 101 and 102 would be
   A. FOFO
   B. FFOO
   C. OOFF
   D. OFOF
   E. None of the above.

18. Execution of the instruction INT 21 by the computer results in
   A. the value of the instruction pointer and code segment changing to address a different place in memory.
   B. the execution of code which is part of the Disk Operating System.
   C. placing the current instruction pointer and segment register values on the stack.
   D. all of the above.
   E. none of the above.

19. The decimal equivalent of the hexadecimal value 1B is
   A. 27
   B. 13
   C. 10
   D. 21
   E. 1,611
--- 20. Pressing the <ESC> key (escape key) on the keyboard results in the sending of which of the following pattern of bits to the computer memory?
A. 00011011
B. 00001101
C. 00001010
D. 00010101
E. None of the above.

Questions 21-26 below use the following program segment:

```assembly
MOV AL, [BX]
MOV CL, [BX+1]
CMP AL, CL
JA L2
L1: MOV [BX+2], CL
    JMP DONE
L2: MOV [BX+2], AL
DONE: MOV AH, 4C
    INT 21
```

--- 21. "JA" is
A. an unconditional jump instruction.
B. a conditional jump instruction.
C. a branch instruction.
D. an instruction to call a subroutine.
E. a comparison instruction.

--- 22. The instruction "CMP AL, CL" will:
A. change the contents of the AL register.
B. change the contents of the BL register.
C. save the result of AL-BL in the DX register.
D. affect the zero status flag.
E. change the data segment register.

--- 23. The instruction "JMP DONE" means:
A. an unconditional jump to the address corresponding to the label name "DONE".
B. if AL > BL jump to the address corresponding to the label name "DONE".
C. if BL > AL jump to the address corresponding to the label name "DONE".
D. if AL = BL jump to the address corresponding to the label name "DONE".
E. to immediately stop the execution of the program.
The purpose of this program is to:

A. place the smallest of numbers in a memory location.
B. place the largest of two numbers in a memory location.
C. add two numbers and place the result in a memory location.
D. test which of two numbers is a negative value.
E. change the zero flag of the status register.

There are three labels: L1, L2, and DONE, they represent:

A. compiler directives.
B. CPU registers.
C. constants.
D. memory locations.
E. macros.

The symbol [BX+1] in the instruction MOV CX, [BX+1] means:

A. move the sum of the hexadecimal values BX and 1 to the CX register.
B. move the contents of the register one higher than the BX register to the CX register.
C. add the contents of the memory at DS:BX to one and store the answer in the CX register.
D. move the contents of the memory at DS:BX to the CX register and add 1.
E. add 1 to the value in the BX register and move the contents of memory at that offset from the DS register to the CX register.

Use following portion of a program to answer questions 27-31.

```
MOV AL, 5
MOV CX, 8
COUNT:  ADD AL, 2
        CMP CX, 0
        JE OVER
        DEC CX
        JMP COUNT
OVER:   MOV BL, AL
```

The register CX is used as:

A. a counter.
B. the base address of an array.
C. a place to accumulate values.
D. a stack pointer.
E. a pointer to a memory location.
--- 28. The function of the instruction "CMP" is to:
   A. increase the contents of CX.
   B. decrease the contents of CX.
   C. test whether CX = 0.
   D. jump to a label.
   E. store a new value in CX.

--- 29. Which instruction is used to decrease a counter by 1?
   A. CMP CX,0.
   B. DEC CX.
   C. JE OVER.
   D. JUMP COUNT.
   E. ADD AL,2.

--- 30. The purpose of these instructions is to:
   A. repeatedly add a value for a specified number of times.
   B. check the values at sequential memory locations for a specified value.
   C. multiply one value by another value.
   D. make the computer "hang up" for a specified time delay.
   E. none of the above.

--- 31. The three instructions "CMP CX,0", "DEC CX", and "JMP COUNT" could be replaced by which of the following instruction:
   A. LOOP
   B. LOOPE
   C. LOOPZ
   D. LOOPNZ.

Questions 32-34 are answered using the following program segment which contains a subroutine:

    MOV CL,3
    CALL SUM
    INC AX
    INT 20H

SUM:

    MOV AL,CL
    L1: CMP CL,1
        JZ L2
        DEC CL
        MUL CL
        JMP L1
    L2: RET
32. The instruction "CALL" means to:
A. do a loop manipulation.
B. do an arithmetic operation.
C. interrupt to the DOS system.
D. call a subroutine.
E. none of the above.

33. The word "SUM" represents:
A. a calculation.
B. a subroutine name.
C. a label name.
D. a jump name.
E. a compiler directive.

34. Which of the following statements is true?
A. In this program, "SUM", "L1", and "L2" represent address labels.
B. At the end of the program, the instruction "RET" is needed to end the program.
C. The purpose of the register CL in this program is to accumulate a sum.
D. Before executing the subroutine, the beginning address of "INC AX" would be automatically pushed onto the stack.

Questions 35-36 are answered using the following program:

```
.MODEL small
.STACK 100H
.DATA
Message DB 'Hello,world!',13,10,':'
.CODE
MOV AX, @data
MOV DS, AX
MOV AH, 9
MOV DX, OFFSET Message
INT 21H
MOV AH, 4CH
INT 21H
END
```

35. The word "Message" in "Message DB 'Hello,world!',
13,10,'$'" is used as:
A. an address.
B. a variable (or variables).
C. a subroutine name.
D. a comment.
36. Which of the following statements is true?
A. "INT 21H" is an DOS interrupt instruction.
B. After executing this program, the screen would show the string "Hello, world!".
C. The words "STACK", "DATA" and "CODE" are segments in CPU.
D. Using the word "DB" defines a data field by one word in length.
E. All of the above are true.
DOS COMMANDS

MATCHING: Select the letter of the choice in the right column that best matches the item in the left column. Record the letter in the blank preceding the item number.

--- 37. Sends the contents of a text file to the printer.  
A. CLS
B. MD
C. REN
D. TREE
E. MORE
F. FORMAT
G. BACKUP
H. DISKCOPY
I. XCOPY
--- 38. Sends the contents of a text file to the screen.
--- 39. Prepare a new disk to receive output from the computer.
--- 40. Copies a group of files from one disk drive to another.
--- 41. Identifies the version of DOS currently loaded in the computer.
--- 42. Copies all files on one disk to another disk.
--- 43. renames a file.
--- 44. Displays a list of files and directories on a disk.
--- 45. Clears the screen.
--- 46. Reboots the system.
--- 47. Outputs its input one screen full at a time.
--- 48. deletes a file.
--- 49. deletes a directory.
--- 50. creates a directory.
--- 51. changes to another directory.
APPENDIX C. POSTTEST 2 FOR IEDT 216
Part I. Multiple Choice: Record the letter of the best choice which completes or answers the question in the blank provided next to the item.

1. Which is the following is NOT a correct way to increment an integer variable i?
   A. i = i + 1;
   B. i++;
   C. i += 1;
   D. i >> 1;

2. Which of the following statement is FALSE?
   A. C programs consist of functions.
   B. Each function should return a value.
   C. Uppercase and lowercase letter are different in the C program.
   D. main() is the first function executed in a C program.
   E. None of the above.

3. The break; statement when used in a switch loop produces
   A. a transfer of control out of the loop.
   B. an exit from a function in the loop.
   C. an exit from the program.
   D. a transfer to the next case statement in the loop.
   E. None of the above.

4. The break; statement when used in a for loop produces
   A. a transfer of control out of the loop.
   B. an exit from a function in the loop.
   C. an exit from the program.
   D. a transfer to the next statement in the loop.
   E. None of the above.
5. The `continue;` statement, when found in a `for` loop, produces
A. a transfer of control out of the loop.
B. an exit from a function in the loop.
C. an exit from the program.
D. a transfer to the next statement in the loop.
E. None of the above.

6. Which of the following is a correct `for` loop?
A. `for (i=1, i < value; i++) printf("%d",i);`
B. `for (i=j=1; (i*j) < 100; i++,j++) printf("%d",i*j);`
C. `for (i=1; i < value; i++) printf("%d",i);`
D. `for i=1; i < value; i++; printf("%d",i);`

7. What is the resulting value of Y in the following program segment?

```c
int x = 2;
int y;
switch (x) {
    case 1: Y = 10;
    case 2: Y = 20;
    case 3: Y = 30;
    default: Y = 100;
}
```

A. 10  B. 20  C. 30  D. 100

8. Which of the following is a macro?
A. `#define MaxValues 100`
B. `#include <stdio.h>`
C. `#define triple(X) ((X) * (X) * (X))`
D. `#include "myfile.h"`

9. The following function is observed in a program:

```c
plusone(int x) { return (++x); }
```

The function `plusone`
A. is of type `int`.
B. returns a value with a fractional part.
C. is a void function.
D. contains two arguments.
10. A float function named DoIt is called by the statement
   \( X = \text{DoIt}(\&Y, \&Z); \)
   We may correctly conclude that
   A. the values of the variables \( Y \) and \( Z \) are passed to the function.
   B. the memory addresses of \( Y \) and \( Z \) are passed to the function.
   C. the values of \( Y \) and \( Z \) cannot be changed by the function.
   D. the function does not contain a return statement.

11. Which of the following statements is false?
   A. An executable file has the extension .exe or .com.
   B. The text editor produces C source files.
   C. The linker produces .exe object files.
   D. The IDE is required to execute a program written in C.

12. Which of the following will be executed at least once?
   A. The body of an if statement.
   B. The body of a for loop.
   C. The body of a while loop.
   D. The body of a do while loop.
   E. None of the above.

13. What is the resulting value of \( s \) in the following code segment?
   \[
   \begin{align*}
   i &= 0; s = 0; \\
   \text{while } (i < 5) & \{ \\
   & \quad i = i + 2; \\
   & \quad s += i; \\
   & \} \\
   \end{align*}
   \]
14. In the following code segment, assume the content of ch is 'x' and the values of A and B are 2 and 3. What is the result after executing the following code segment?

```c
switch (ch)
{
    case '+': printf("%d", A+B); break;
    case 'x':
    case '*': printf("%d", A*B); break;
    default : printf("ERROR");
}
```

A. 5  
B. 6  
C. ERROR  
D. none of the above
Part II. Programming (10 points): Construct a computer program to calculate the monthly payment on a loan. The payment on a loan is calculated as follows:

\[ \frac{(I \times A)}{N} \]

where \( P \) is the Payment,
\( I \) is the interest rate, e.g. 0.075,
\( A \) is the amount borrowed,
\( N \) is the number of payments per year, e.g. 12
\( Y \) is the number of years over which the money was borrowed.

You will need to use the pow() function included in the math.h header file. The pow() function returns the value of a number raised to some power - in this case \( N \times Y \) power. For example, the following short program would print the value of some value \( X \) raised to the \( Y \) power:

```c
#include <stdio.h>
#include <math.h>
void main(void)
{
    double x,y,z;

    printf("Enter the value of X : ");
    scanf("%lf", &x);
    printf("Enter the power to raise X : ");
    scanf("%lf", &y);
    z = pow(x,y);
    printf("The value of %lf raised to %lf is %lf\n", x,y,z);
}
```

You may write your program on the back sides of your test pages.
DIRECTIONS: Print your name and social security number above.
Circle the one best response to each item below. If you need to change a response, completely erase before marking the new answer.

1. ASCII is a representation of
   a. A Simple C Instruction Implementation.
   c. Asynchronous Input Interface.
   d. Application System Code for Interactive Interface.

2. Which of the following declarations would result in the fewest bytes of memory allocation by the compiler?
   a. char ch;
   b. char *ch;
   c. unsigned ch;
   d. int ch;

3. Which of the following is a relational operator?
   a. =
   b. |
   c. &
   d. >

4. Which of the following for loops is incorrect?
   a. for( ; i < 5; ) printf("%d",i++);
   b. for (i = 0, j = 100; i < nosteps; i++,j--) printf(%d%d",i,j);
   c. for ( i = start; i < endpt; i++)
   {printf("%lf",table[i]);}
   d. for ( i =0, i < 10, i++) printf("i = %d",i);

5. A while loop is more appropriate than a for loop when
   a. the terminating condition occurs unexpectedly.
   b. the body of the loop will be executed at least once.
   c. the program will be executed at least once.
   d. the number of times the loop will be executed is known before the loop is executed.
6. A **do while** loop is useful when
   a. the body of the loop will never be executed.
   b. the body of the loop will be executed at least once.
   c. the body of the loop may never be executed.
   d. the loop will likely be interrupted by a break.

7. If X is 25, what is the value of the following conditional expression?

   $$ (X > 20) ? 0 : X \times 4; $$

   a. 0
   b. 4
   c. 25
   d. 100

8. Which of the following cannot be passed to a function via arguments?
   a. constants
   b. variables (with values)
   c. preprocessor directives
   d. expressions (that evaluate to a value)
   e. functions (that return values)

9. What does this combination of statements do?

   ```
   #define LIM 50
   char collect[LIM];
   ```

   a. makes LIM a subscript.
   b. makes LIM a variable of type float.
   c. makes collect[] an array of type LIM.
   d. makes collect[] an array of size LIM.

10. What will happen if you put too few elements in an array when you initialize it?
    a. nothing.
    b. possible system malfunction.
    c. an error message from the compiler.
    d. unused elements will be filled with 0s or garbage.

11. Which of the following is more appropriate for reading in a string in which blanks are interspersed with characters?
    a. `gets()`
    b. `reads()`
    c. `scan()`
    d. `fscanf()`
12. A function is called as `getavg(&sum, &novals)`. What purpose do the `&sum` and `&novals` serve?
   a. They are integer values being passed to the function.
   b. They are the addresses of the function and the calling program.
   c. They are addresses of the variables where we want values returned or modified in the calling program.
   d. They are the addresses of library routines needed by the function.

13. In the expression `double *dptr;` what has type double?
   a. the variable `dptr`
   b. the address of `dptr`
   c. the variable pointed to by `dptr`
   d. None of the above

14. Given the following C statements, what would be printed?

   ```c
   int *value;
   int *newvalue;
   int oldvalue = 5;
   value = &oldvalue;
   newvalue = value;
   printf("%d",*newvalue);
   ```

   a. 5
   b. The address of oldvalue.
   c. The address of the pointer newvalue;
   d. The address of the pointer value.

15. Different C language compiler models are used
   a. depending on the kind of computer being used.
   b. depending on the type of graphic display being used.
   c. depending on the space required for data and program code.
   d. depending on the availability of a math coprocessor.

16. The hexadecimal value equivalent of the binary value `10101111` is
   a. AF
   b. 827
   c. 257
   d. BE
17. The decimal equivalent of the hexadecimal number 2F is
   a. 101111
   b. 47
   c. 57
   d. 215

18. In graphics mode, the higher the resolution mode of operation chosen
   a. the fewer the number of pixels that may be addressed.
   b. the fewer the number of colors that may be concurrently displayed.
   c. the fewer the number of graphic commands that may be used.
   d. the fewer the number of machines available to use the program.

19. An object in C++ is
   a. an instance of an object.
   b. a graphic image.
   c. an encapsulated function.
   d. a polymorphic constant.

20. Polymorphism refers to
   a. the ability to combine data and functions.
   b. a union of multiple structures.
   c. the ability to assume different attributes.
   d. different functions that perform the same task.

21. Inheritability permits
   a. using a function written for one program in another program.
   b. creating a new class using other class definitions.
   c. creating a new object from other objects.
   d. all of the above.

22. In the following C statement, `firstfile` is a variable of which type?

   ```c
   firstfile = fopen(filename,"r");
   ```

   a. int
   b. float
   c. unsigned int
   d. pointer
23. In the following statement, what would the value of \( C \) be used for?

\[ \text{fwrite}(A, B, C, D); \]

a. The size of a structure to be written.
b. The name of the file pointer variable.
c. The number of structures to be written.
d. The address of the structure variable to be written.

24. In the statement \( \text{fp} = \text{fopen(myfile,"r"}); \), we know that the file to be read

a. is a binary type file.
b. is a text file.
c. may be either a text or binary file.
d. does not contain end of line characters in it.

25. If the statement \( \text{file1} = \text{fopen("MYFILE","r"}); \) is executed but \( \text{MYFILE} \) does not exist, the variable \( \text{file1} \) will contain

a. garbage.
b. an error number.
c. an end-of-file character.
d. the null character.
Write a function which switches from graphic mode to text mode, clears the screen and opens a window 4 rows high and 40 characters long which has a blue background color and yellow text color. Have the function display

```
ENTER FILE PATH AND NAME:
```

and return a filename. The function should restore the screen to the previous graphic mode before completing its task.

Complete the function (10 points) below:

```c
void GetFile (char filename[])
{
    // Function implementation
}
```
APPENDIX E. MEANS AND STANDARD DEVIATIONS ON PRETEST
<table>
<thead>
<tr>
<th>Item #</th>
<th>Mean</th>
<th>SD</th>
<th>Item #</th>
<th>Mean</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.755</td>
<td>.435</td>
<td>21</td>
<td>.612</td>
<td>.492</td>
</tr>
<tr>
<td>2</td>
<td>.347</td>
<td>.481</td>
<td>22</td>
<td>.735</td>
<td>.446</td>
</tr>
<tr>
<td>3</td>
<td>.592</td>
<td>.497</td>
<td>23</td>
<td>.449</td>
<td>.503</td>
</tr>
<tr>
<td>4</td>
<td>.347</td>
<td>.481</td>
<td>24</td>
<td>.122</td>
<td>.331</td>
</tr>
<tr>
<td>5</td>
<td>.306</td>
<td>.466</td>
<td>25</td>
<td>.653</td>
<td>.481</td>
</tr>
<tr>
<td>6</td>
<td>.184</td>
<td>.391</td>
<td>26</td>
<td>.490</td>
<td>.505</td>
</tr>
<tr>
<td>7</td>
<td>.327</td>
<td>.474</td>
<td>27</td>
<td>.245</td>
<td>.435</td>
</tr>
<tr>
<td>8</td>
<td>.429</td>
<td>.500</td>
<td>28</td>
<td>.122</td>
<td>.331</td>
</tr>
<tr>
<td>9</td>
<td>.592</td>
<td>.497</td>
<td>29</td>
<td>.653</td>
<td>.481</td>
</tr>
<tr>
<td>10</td>
<td>.469</td>
<td>.504</td>
<td>30</td>
<td>.612</td>
<td>.492</td>
</tr>
<tr>
<td>11</td>
<td>.755</td>
<td>.435</td>
<td>31</td>
<td>.245</td>
<td>.435</td>
</tr>
<tr>
<td>12</td>
<td>.816</td>
<td>.391</td>
<td>32</td>
<td>.490</td>
<td>.505</td>
</tr>
<tr>
<td>13</td>
<td>.674</td>
<td>.474</td>
<td>33</td>
<td>.143</td>
<td>.354</td>
</tr>
<tr>
<td>14</td>
<td>.449</td>
<td>.503</td>
<td>34</td>
<td>.225</td>
<td>.422</td>
</tr>
<tr>
<td>15</td>
<td>.388</td>
<td>.492</td>
<td>35</td>
<td>.265</td>
<td>.446</td>
</tr>
<tr>
<td>16</td>
<td>.306</td>
<td>.466</td>
<td>36</td>
<td>.102</td>
<td>.306</td>
</tr>
<tr>
<td>17</td>
<td>.592</td>
<td>.497</td>
<td>37</td>
<td>.204</td>
<td>.407</td>
</tr>
<tr>
<td>18</td>
<td>.633</td>
<td>.487</td>
<td>38</td>
<td>.306</td>
<td>.466</td>
</tr>
<tr>
<td>19</td>
<td>.408</td>
<td>.497</td>
<td>39</td>
<td>.163</td>
<td>.373</td>
</tr>
<tr>
<td>20</td>
<td>.367</td>
<td>.487</td>
<td>40</td>
<td>.245</td>
<td>.435</td>
</tr>
</tbody>
</table>
APPENDIX F. MEANS AND STANDARD DEVIATIONS ON POSTTEST 1
### Means and Standard Deviations on Posttest 1

<table>
<thead>
<tr>
<th>Item #</th>
<th>Mean</th>
<th>SD</th>
<th>Item #</th>
<th>Mean</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.878</td>
<td>0.331</td>
<td>27</td>
<td>0.735</td>
<td>0.446</td>
</tr>
<tr>
<td>2</td>
<td>0.714</td>
<td>0.456</td>
<td>28</td>
<td>0.918</td>
<td>0.277</td>
</tr>
<tr>
<td>3</td>
<td>0.633</td>
<td>0.487</td>
<td>29</td>
<td>0.980</td>
<td>0.143</td>
</tr>
<tr>
<td>4</td>
<td>0.735</td>
<td>0.446</td>
<td>30</td>
<td>0.653</td>
<td>0.481</td>
</tr>
<tr>
<td>5</td>
<td>0.143</td>
<td>0.354</td>
<td>31</td>
<td>0.469</td>
<td>0.504</td>
</tr>
<tr>
<td>6</td>
<td>0.796</td>
<td>0.407</td>
<td>32</td>
<td>0.898</td>
<td>0.306</td>
</tr>
<tr>
<td>7</td>
<td>0.163</td>
<td>0.373</td>
<td>33</td>
<td>0.714</td>
<td>0.456</td>
</tr>
<tr>
<td>8</td>
<td>0.735</td>
<td>0.446</td>
<td>34</td>
<td>0.408</td>
<td>0.497</td>
</tr>
<tr>
<td>9</td>
<td>0.449</td>
<td>0.503</td>
<td>35</td>
<td>0.204</td>
<td>0.407</td>
</tr>
<tr>
<td>10</td>
<td>0.082</td>
<td>0.277</td>
<td>36</td>
<td>0.592</td>
<td>0.497</td>
</tr>
<tr>
<td>11</td>
<td>0.571</td>
<td>0.500</td>
<td>37</td>
<td>0.959</td>
<td>0.200</td>
</tr>
<tr>
<td>12</td>
<td>0.449</td>
<td>0.503</td>
<td>38</td>
<td>0.918</td>
<td>0.277</td>
</tr>
<tr>
<td>13</td>
<td>0.796</td>
<td>0.407</td>
<td>39</td>
<td>0.980</td>
<td>0.143</td>
</tr>
<tr>
<td>14</td>
<td>0.225</td>
<td>0.422</td>
<td>40</td>
<td>0.857</td>
<td>0.354</td>
</tr>
<tr>
<td>15</td>
<td>0.408</td>
<td>0.497</td>
<td>41</td>
<td>0.939</td>
<td>0.242</td>
</tr>
<tr>
<td>16</td>
<td>0.980</td>
<td>0.143</td>
<td>42</td>
<td>0.878</td>
<td>0.331</td>
</tr>
<tr>
<td>17</td>
<td>0.408</td>
<td>0.497</td>
<td>43</td>
<td>0.837</td>
<td>0.373</td>
</tr>
<tr>
<td>18</td>
<td>0.429</td>
<td>0.500</td>
<td>44</td>
<td>0.980</td>
<td>0.143</td>
</tr>
<tr>
<td>19</td>
<td>0.878</td>
<td>0.331</td>
<td>45</td>
<td>0.959</td>
<td>0.200</td>
</tr>
<tr>
<td>20</td>
<td>0.469</td>
<td>0.504</td>
<td>46</td>
<td>0.694</td>
<td>0.466</td>
</tr>
<tr>
<td>21</td>
<td>0.816</td>
<td>0.391</td>
<td>47</td>
<td>0.960</td>
<td>0.200</td>
</tr>
<tr>
<td>22</td>
<td>0.469</td>
<td>0.504</td>
<td>48</td>
<td>0.857</td>
<td>0.354</td>
</tr>
<tr>
<td>23</td>
<td>0.776</td>
<td>0.422</td>
<td>49</td>
<td>0.918</td>
<td>0.277</td>
</tr>
<tr>
<td>24</td>
<td>0.551</td>
<td>0.503</td>
<td>50</td>
<td>0.959</td>
<td>0.200</td>
</tr>
<tr>
<td>25</td>
<td>0.510</td>
<td>0.505</td>
<td>51</td>
<td>1.000</td>
<td>0.000</td>
</tr>
<tr>
<td>26</td>
<td>0.408</td>
<td>0.497</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX G. MEANS AND STANDARD DEVIATIONS ON POSTTEST 2 FOR IEDT 216
<table>
<thead>
<tr>
<th>Item #</th>
<th>Mean</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.903</td>
<td>.301</td>
</tr>
<tr>
<td>2</td>
<td>.871</td>
<td>.341</td>
</tr>
<tr>
<td>3</td>
<td>.548</td>
<td>.506</td>
</tr>
<tr>
<td>4</td>
<td>.677</td>
<td>.475</td>
</tr>
<tr>
<td>5</td>
<td>.774</td>
<td>.425</td>
</tr>
<tr>
<td>6</td>
<td>.419</td>
<td>.502</td>
</tr>
<tr>
<td>7</td>
<td>.419</td>
<td>.502</td>
</tr>
<tr>
<td>8</td>
<td>.613</td>
<td>.495</td>
</tr>
<tr>
<td>9</td>
<td>.419</td>
<td>.501</td>
</tr>
<tr>
<td>10</td>
<td>.710</td>
<td>.461</td>
</tr>
<tr>
<td>11</td>
<td>.516</td>
<td>.508</td>
</tr>
<tr>
<td>12</td>
<td>.710</td>
<td>.461</td>
</tr>
<tr>
<td>13</td>
<td>.194</td>
<td>.402</td>
</tr>
<tr>
<td>14</td>
<td>.290</td>
<td>.461</td>
</tr>
<tr>
<td>15</td>
<td>6.839</td>
<td>2.464</td>
</tr>
</tbody>
</table>
APPENDIX H. MEANS AND STANDARD DEVIATIONS ON POSTTEST 2 FOR IEDT 316
<table>
<thead>
<tr>
<th>Item #</th>
<th>Mean</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.917</td>
<td>.288</td>
</tr>
<tr>
<td>2</td>
<td>.667</td>
<td>.492</td>
</tr>
<tr>
<td>3</td>
<td>.833</td>
<td>.389</td>
</tr>
<tr>
<td>4</td>
<td>.083</td>
<td>.289</td>
</tr>
<tr>
<td>5</td>
<td>.917</td>
<td>.289</td>
</tr>
<tr>
<td>6</td>
<td>.917</td>
<td>.289</td>
</tr>
<tr>
<td>7</td>
<td>.667</td>
<td>.492</td>
</tr>
<tr>
<td>8</td>
<td>.500</td>
<td>.522</td>
</tr>
<tr>
<td>9</td>
<td>.917</td>
<td>.289</td>
</tr>
<tr>
<td>10</td>
<td>.833</td>
<td>.389</td>
</tr>
<tr>
<td>11</td>
<td>.583</td>
<td>.515</td>
</tr>
<tr>
<td>12</td>
<td>.833</td>
<td>.389</td>
</tr>
<tr>
<td>13</td>
<td>.667</td>
<td>.492</td>
</tr>
<tr>
<td>14</td>
<td>.167</td>
<td>.389</td>
</tr>
<tr>
<td>15</td>
<td>.583</td>
<td>.515</td>
</tr>
<tr>
<td>16</td>
<td>1.000</td>
<td>.000</td>
</tr>
<tr>
<td>17</td>
<td>1.000</td>
<td>.000</td>
</tr>
<tr>
<td>18</td>
<td>.083</td>
<td>.289</td>
</tr>
<tr>
<td>19</td>
<td>.750</td>
<td>.452</td>
</tr>
<tr>
<td>20</td>
<td>.667</td>
<td>.492</td>
</tr>
<tr>
<td>21</td>
<td>.250</td>
<td>.452</td>
</tr>
<tr>
<td>22</td>
<td>.833</td>
<td>.389</td>
</tr>
<tr>
<td>23</td>
<td>.750</td>
<td>.452</td>
</tr>
<tr>
<td>24</td>
<td>.500</td>
<td>.522</td>
</tr>
<tr>
<td>25</td>
<td>.750</td>
<td>.452</td>
</tr>
<tr>
<td>26</td>
<td>4.750</td>
<td>2.051</td>
</tr>
</tbody>
</table>