A PILOT LANGUAGE SYSTEM

by

LARRY TRISTAN WALKER

B. S., Georgia Institute of Technology, 1961

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas
1978

Approved by:

Myron A. Calhoun
Major Professor

TABLE OF CONTENTS

INDEX OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 PURPOSE OF THE REPORT

The purpose of this report is to describe the implementation of a microprocessor-based Computer Assisted Instruction (CAI) system for the PILOT computer language. The report provides documentation sufficient to support software maintenance and user guides for the system. PILOT stands for Programmed Inquiry, Learning or Teaching and is a relatively easy to learn computer language. This report presents an outline of the general requirements of a CAI system and rational for selecting the PILOT language for this microprocessor-based implementation.

1.2 HISTORY OF PILOT

A principal barrier to the use of CAI has been the large amount of time needed to write courses in a general purpose language such as FORTRAN. The effective teaching of a majority of subjects requires the presentation of large amounts of text and the receipt of student answers in words.

The programming languages that have significant text-processing features, like SNOBOL, appear to be well suited for CAI. These languages introduce another problem, however, as the languages are too complicated for most teachers to learn; and therefore a computer programmer is required to prepare the course software (courseware).

Languages have been developed for writing courseware. Most of them, like COURSEWRITER, PLANIT, and TUTOR are large, complex languages that require a major effort to learn. This author's experience using PLANIT at the U.S. Army Command and General Staff College, Fort Leavenworth, Kansas, was that it took twenty to thirty hours of intensive study (via CAI) for nonprogrammer graduate students to develop a rudimentary PLANIT programming skill.

In approximately 1970, John Starkweather of the University of California Medical Center developed PILOT. It was significantly different from the CAI languages seen before as it was exceptionally simple in form. It was used at Stanford Research Institute in an education research study. Findings of this study were that teachers and students learned the language very easily and were able to write programs immediately [1].

The National Library of Medicine, Bethesda, Maryland has been a strong supporter of PILOT. Through a grant from that agency, John A. Starkweather developed an assembly language version of PILOT for the Intel 8080 microprocessor [2]. This version, known as 8080 PILOT, was the starting point for this project.

1.3 SYSTEM ORGANIZATION

The system described by this report consists of an adaptation of 8080 PILOT to a specific hardware grouping,

integrated with system level software to provide for file creation, storage and retrieval. The software is divided into a PILOT author subdivision, a student subdivision and an operating system subdivision which supports both the PILOT author and student elements. Maximum practical use is made of existing software modules to perform functions of the overall system.

The hardware elements consist of a Digital Group Z-80 microcomputer system located in the Microprocessor Laboratory of the Department of Computer Science at Kansas State University. Input devices into the hardware configuration consist of keyboard, digital cassette tape and audio cassette tape. Output devices consist of CRT display, impact printer, digital and audio tape cassette.

CHAPTER 2

SYSTEM REQUIREMENTS

2.1 CLASSES OF USERS

CAI systems typically have three general classes of users. The class with the largest population is the student class. This class is characterized by having little computer experience and limited ability to follow complex instructions. Of course there are many exceptions to this profile but the developer of a CAI system should plan to support a student with these basic characteristics.

The next class of users is the CAI author class. They will be the creators of courseware (lesson material) for the system. The span of computer qualification of this class is great. The utility of the system can be greatly enhanced if it can be used by those among the lowest segment of qualification.

The last class of user is the system maintainer. Functions for this user include updates to software, backup of mass storage, purging of unwanted files and other duties as desired. He needs the widest possible scope of capabilities.

2.2 FUNCTIONAL PERFORMANCE

The student's use of the system is limited to lesson execution. In this mode he requires rapid execution to remain interactive with the system and a simple control

structure to avoid complex instructions. For example, the system should be very simple to start up, load a lesson and begin execution. It is important to note that execution should be rapid while in the lesson, i.e., no frequent long pauses.

It is this author's observation that the student is very tolerant to a minutes pause after five or ten minutes of intensive interaction. This tolerance is very high if the lesson material conducts a dialog to explain the delay, i.e., "catch your breath while I load a new lesson for you." This concept for user tolerance is a critical factor in the system configuration.

The CAI author use of the system includes lesson preparation, trial lesson execution, and lesson file storage and retrieval. The author peformance speed requirements are similar to that of the student. He expects rapid response while in the lesson creation mode (which is, in reality, an editor environment). For text changes, he expects rapid response. His tolerance for delay at the end of a lengthy editing session is not dissimilar to the student tolerance for pauses between lessons.

The maintainer can be considered as a user that could employ the most rapid response that can be achieved within the hardware limits of the system.

The very nature of CAI requires that lessons be transported easily from one system to another via a compact, inexpensive and reliable media. The above general

enumeration of user requirements dictates the system performance.

2.3 HARDWARE CONSIDERATIONS

User requirements for system performance determine the extent of hardware in general. The CAI environment typically requires rapid execution response with reasonable toleration for infrequent pauses as outlined in preceeding paragraphs.

Cost can become a principal consideration. In a Z-80 system built of The Digital Group Inc. [3] components, the following price structure is applicable: Floppy Disk Drive (Single) System, 240,000 bytes of storage, $1,195.00; Digital Cassette (Dual) Storage System, 1,300,000 bytes of storage, $675.00; Disk Storage Media $7.00; Digital Cassette Storage Media $1.80. It is significant to note that the lower cost cassette-based mass storage systems may satisfy the user's requirements as well as the higher cost disk mass storage system, since the user has a tolerance level for much longer file loading time of the digital cassette system. In a CAI environment this file loading occurs at lesson change time for the student and edit begin/end time for the CAI author. If courses are long, the cassette advantage increases as it provides for greater storage at a much lower media cost and requires less frequent changes of media to get additional lessons.

CHAPTER 3

HARDWARE IMPLEMENTATION ALTERNATIVES

3.1 HARDWARE CONFIGURATIONS AVAILABLE

The Kansas State Department of Computer Science has two microprocessor systems that are candidates for the 8080 PILOT implementation. These are the ALTAIR 8800 (Intel 8080 microprocessor) and the Digital Group (Z-80 microprocessor) systems.

The ALTAIR class of system is exactly the system upon which the 8080 PILOT was designed to be implemented. The 8080 PILOT is based on lesson loading from asynchronous paper tape via program command and character by character output of lesson material via teletype. The Digital Group Systems, on the other hand, do not have any asynchronous loading capability and the prinicipal output media is a video display system that requires far more complex logic than the simple teletype protocol the 8080 PILOT is designed to support.

3.2 SELECTION CRITERIA

The major strength of the Digital Group hardware is that it has computer-controlled mass storage media and this single feature prompted its selection as this author desired to create a practical CAI system. Mass storage under computer control, in the opinion of this author, is required

for a practical CAI system.

Further considerations were the factors of ease of start up of the two systems. The ALTAIR requires the bit by bit switch entry of a binary bootstrap program while the other system requires only pushing the "on" button as The Digital Group hardware has a Read Only Memory (ROM) start up facility. A final consideration was the fact that this author is thoroughly familiar with the Digital Group System as he owns one himself!

CHAPTER 4

SOFTWARE IMPLEMENTATION ALTERNATIVES

4.1 TOTAL SYSTEM SOFTWARE REQUIREMENTS

The student software facility requires a method of loading the PILOT interpreter initially, loading lessons initially and executing the lesson material. While executing, the student facility may require the ability to branch to a new lesson. This feature requires automatic loading of the desired lesson and execution of that lesson. Note, this automatic loading and subsequent execution is not possible under the original 8080 PILOT as it has no facility for control of processor-managed mass storage. Therefore, the automatic branching to new lessons represents a major enhancement to 8080 PILOT.

The PILOT author software facility requires all of the student capability for the trial execution of lessons and, in addition, requires the lesson creation ability via a lesson editor program. Again the 8080 PILOT does not have an editor and thus this lesson editor capability represents another major enhancement to 8080 PILOT. The PILOT author also requires the ability to move lesson files to and from the editor to mass storage at the start and end of lesson edit sessions.

The maintainer requires all of the PILOT author facility. In addition, the maintainer requires the ability to change the code of the PILOT interpreter, to debug

programs, to manipulate files on a group basis and to prepare backups of the mass storage media.

4.2 SOFTWARE SOURCES

The 8080 PILOT interpreter was available only as an Intel 8080 assembly listing and could not be obtained in any machine readable media [4]. This listing did include a hex listing of the machine language.

An acceptable editor was available as a standard product of The Digital Group, Inc. This editor is an adaptation of an editor previously published in Dr. Dobb's Journal [5]. No listing was available of the Digital Group Editor, and it has been extensively modified from the original version. A listing was available of the original published version.

The standard product Digital Group Operation System, PHIMON (Phideck monitor) was available to provide the required system level support and maintenance capabilities required for a complete system. A BASIC interpreter, MAXI BASIC 1.1, was available. This is a standard Digital Group Software product and has extensive display interface routines needed to adapt 8080 PILOT to the Digital Group display. No documentation was available of the internal operations of MAXI BASIC.

4.3 AVAILABLE SOFTWARE DEVELOPMENT TOOLS

Since the 8080 PILOT was available in Intel 8080 assembly language, the principal software tool needed was an Intel 8080 assembler. Unfortunately, this critical item was missing. Only a Zilog Z-80 assembler was available. While the two instruction sets are compatible, the mnemonics for the operation codes and assembler formats are drastically different. It has been this author's experience that translation from one set of instructions to the other is a very error prone operation.

In addition, a Zilog Z-80 symbolic disassembler was available along with excellent facilities for direct entry of code into memory and dumping of memory contents to display and printer.

4.4 SELECTED SOFTWARE TECHNIQUES

A decision was made to enter the 8080 PILOT code directly into memory in machine language, dump this code in hex format, manually compare the dump with the original listing for errors and modify the code at the machine language level. The Zilog Z-80 disassembler was used to document the resultant end product. This process was facilitated by having a skilled typist enter the original listing. Very few errors were discovered in the entire process. Those that were discovered have been, in the majority, uncovered by the disassembler producing nonsense

code because the errors were not logical instruction seqúences.

In cases where major changes were made to the 8080 PILOT interpreter, the new code was assembled with the Zilog Z-80 assembler and patched into the main body of machine language code. All changes made to the editor were made at the machine language level.

The display routines were developed by disassembling the MAXI BASIC routines in their original location, moving the code in memory to the PILOT machine language segment, and then correcting all nonrelative addresses and external calls.

4.5 CONCLUSION

The use of machine language level programming is a difficult process and in this project has increased the time required to complete the PILOT segment of the software. With the clarity of hindsight, the author observes that time necessary to implement an Intel 8080 assembler would probably have been recouped in saving of time spent on machine language programming. The greatest advantage to be achieved, however, would lie in increases in the ease of maintenance of the PILOT interpreter. If the 8080 PILOT source code had been available in machine readable media, the machine language approach would never have been taken.

CHAPTER 5

PILOT LANGUAGE PROCESSOR DESCRIPTION

5.1 SYSTEM LEVEL ORGANIZATION

Comments in this chapter pertain to the original 8080 PILOT interpreter by Starkweather. The reader is encouraged to refer to Appendix A, PILOT Language Definition, and reference 4, "Source Code for 8080 PILOT," as required for comprehension. The original PILOT interpreter was organized into 72 assembly language routines and storage areas. The interpreter provided for the entry of a PILOT lesson but with no provision for editing that lesson or even adding new material to the end of a lesson.

Printing, displaying, storing (on paper tape) and execution of lessons was provided. The command structure of the interpreter provided for branching to an editor but no editor was included. Provisions were also made to allow branching to a BASIC interpreter for statement evaluation but no such interpreter was provided. A facility was provided for loading a new lesson from paper tape.

5.2 PROCESSOR OPERATIONS

A typical session with PILOT starts by utilizing the "Input" routine (0CCFH of reference 4) to place a lesson in the program buffer. Once this is complete, the user can either execute this lesson or store it on paper tape. If a

choice is made to store it, control passes to the "save" routine (0C13H of reference 4).

Subsequent execution of this lesson turns control over to "Interpret Existing Program, IEP (057CH of reference 4). Execution follows a logical path of Scan Input Buffer, SCAN (0380H of reference 4); Interpret Operation Code, OPS (03CEH of reference 4) and return to SCAN for more code. Figure 5.1 illustrates this general flow. Interpret Operation Code calls other elements of the interpreter as required.

5.3 MEMORY UTILIZATION

The 8080 PILOT interpreter resides from 0080H to 0EEAH with lesson material stored from 1000H upward. String variables are stored from 1FFFH downward. The code from 02E0H to 0EEA is never modified and could be placed in Read Only Memory (ROM) if desired.

5.4 INPUT AND OUTPUT OPERATIONS

The code is remarkably well organized for input/output. A jump table (located at 0100H) defines all the locations for I/O routines. This table organization gives great flexibility to change I/O even if the interpreter is placed in ROM.

I/O capabilities include asynchronous character-by-character interface to an ASR Model 33

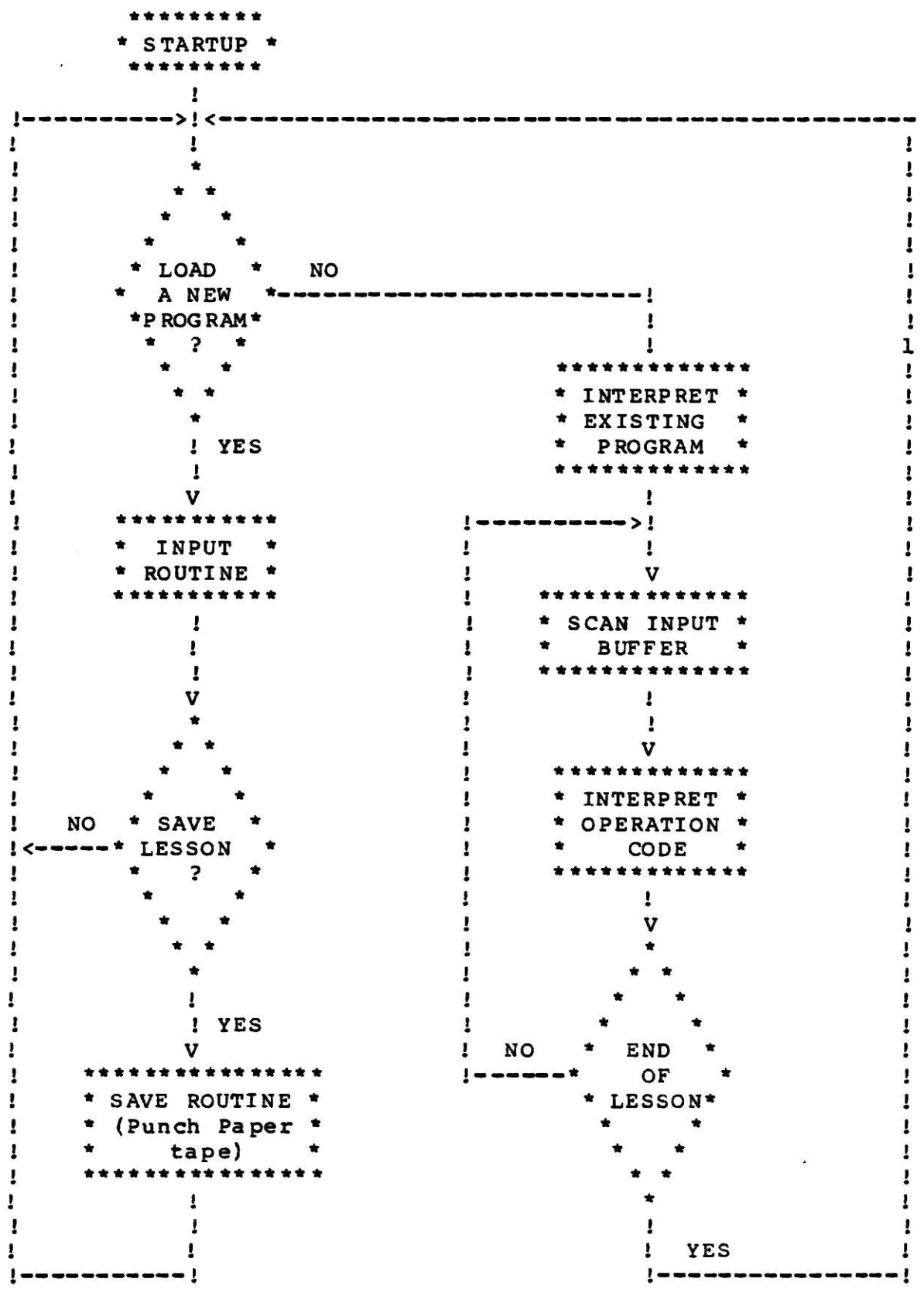


FIGURE 5.1 PILOT GENERAL FLOW

Teletype, jump to Editor (null condition, returned to monitor), jump to BASIC (null condition, returned with no action) and jump to a monitor. This monitor was not part of the 8080 PILOT implementation but was assumed to reside in the system in high memory.

CHAPTER 6

PILOT LANGUAGE PROCESSOR MODIFICATIONS

6.1 SYSTEM LEVEL REQUIREMENTS

The major changes required to the PILOT processor involved integrating it into an operating system structure to provide for lesson files loading, interface to the keyboard and display system of the Digital Group Z-80 microprocessor system, and changing the internal operation of the processor to provide for automatic lesson execution on initial start up. This last feature was deemed necessary for creation of a very simple student environment.

6.2 DISPLAY DRIVER

The source of the display driver code was the Digital Group MAXI BASIC interpreter. It appeared ideally suited to interfacing a character-by-character output into the Digital Group Display. The MAXI BASIC routine was first disassembled to determine external routines that it addressed. It was found that it addressed only the "Character Out" and "Space Out" routines of the Digital Group standard ROM. Since PILOT uses the standard ROM area for stack space, it was not possible to use these "Character Out" and "Space Out" routines. Similar routines exist in PHIMON and references to these routines were substituted for the ROM routines. This created a small constraint. PHIMON

must be resident in the system for the Display Driver to work. Calls to "Character Out" (E6B2H) were inserted at 0F79H, 0F84H, and 0FHFH (see Appendix C for symbolic code listing).

The display buffer for the screen requires 1024 decimal bytes. Memory locations 1000H through 13FFH are used for this purpose. Nonrelative addresses of all the elements of the code were identified, the code was moved to 0F00H with a small memory move program, and all nonrelative addresses were adjusted for the new location.

6.3 KEYBOARD DRIVER

The source of the keyboard driver was the MAXI BASIC interpreter. The MAXI BASIC keyboard routine is contiguous with the MAXI BASIC display routine. This keyboard routine was moved and had nonrelative addresses adjusted concurrently with the display driver routine as described in Section 6.2 In addition, a subroutine call was changed as described below. The keyboard driver called a "Keystroke In" routine located in low memory of the MAXI BASIC interpreter. This call (located in the keyboard driver at OFB2H) was changed to call a "Keystroke In" routine (located at E36AH) of PHIMON. The use or the PHIMON "Keystroke In" routine eliminated the requirement to include a "Keystroke In" routine in PILOT.

6.4 FILES INTERFACE

The files interface required the creation of a linkage to allow file commands for loading new lessons to be passed from the PILOT lesson under execution to the PHIMON operating system and, if required, error messages to be passed back to the user. The technique used was to dynamically modify a segment of PHIMON code to change it from a general purpose overlay to a large subroutine to accomplish the specific file loading required. The modified code was altered so that it no longer appeared as a correct system overlay. This alteration prevented PHIMON attempting to use the modified code as a general purpose overlay.

In addition, formatting was performed on the PILOT command to convert it to the form needed by PHIMON. The formatted command was then moved from the PILOT input buffer to PHIMON's command buffer.

The Z-80 Assembler was used to produce this code. A message editor that resides in PHIMON was used to output error messages. A detailed, commented listing of the code is in Appendix D, Assembly Language Listing of Files Interface for Z-80 PILOT. This code is envoked by a call located at 0BE7H of the PILOT interpreter.

The PILOT command format for loading new lessons from deck 0 is:

"LOAD: FILENAME"

(where FILENAME is any legal PHIMON filename). The command for loading from other tape decks is:

"LOAD:#X FILENAME"

(where X is the drive number). If an error occurs, the logical sequence of the lesson is broken, so the routine gives the student the error message:

"PILOT LESSON DESIRED NOT ON THIS TAPE. ASK FOR HELP."

The PILOT system is then halted and control is given to PHIMON.

6.5 AUTOMATIC START UP

Originally PILOT expected the user to start the system into executing the lesson or preparing a new lesson. If the alternative of preparing a new lesson was selected, the system immediately cleared the lesson (program) buffer of all its contents and prompted the user to enter a new lesson from the keyboard. In summary, PILOT started up under control of the student, and unless the student proceeded properly, the lesson prepared by the author would never gain control of the teaching environment.

This new implementation of PILOT has changed that sequence so that the PILOT system initially starts up under control of the lesson rather than under control of the student. The student merely has to push a button to turn the system on and then enter:

"RUN PILOT" (carriage return)

The system starts up under control of the author prepared lesson.

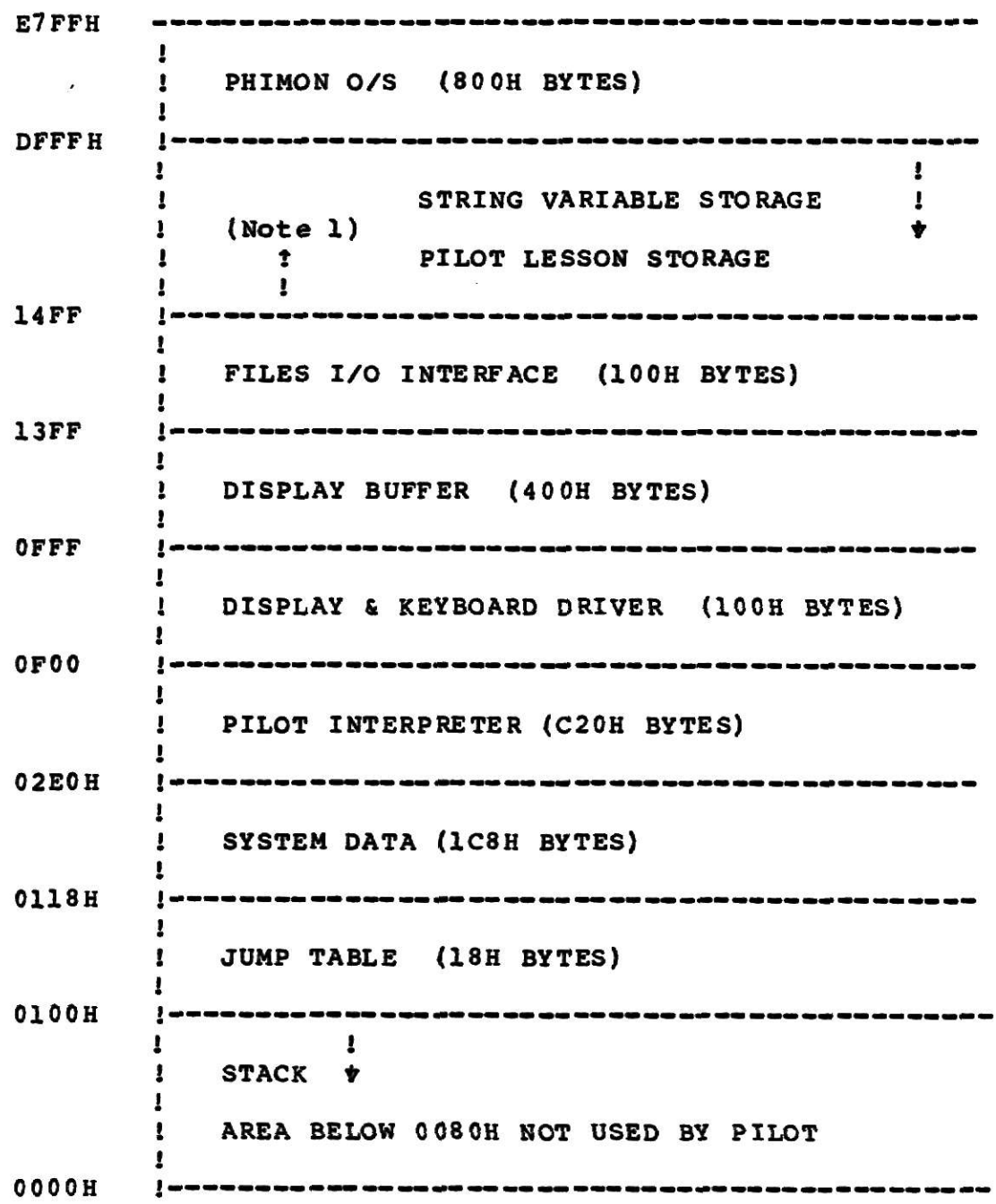
This modification was accomplished by moving the start up buffer location from the input buffer (0E72H of reference 4) to the program buffer, the location where PILOT lessons are loaded. This creates one potential problem. If PILOT is run without any lesson loaded, it will attempt to execute whatever is in memory in the lesson storage area. This is simple to overcome by always having some small lesson loaded as part of the PILOT system. Once the details of the PILOT interpreter were understood, this modified start up procedure was simple to accomplish. The first line of the "Initialize Data for New Program" routine (0356H of reference 4) was changed from:

```
INIT LXI H, IBUFF
        TO
INIT LXI H, PBUFF.
```

In addition, in this same "INIT" routine the calls to "NEWN" and "INITV" were eliminated. This retains all variables from one lesson to the next. The variables can be cleared if desired by the incoming lesson so all alternatives are possible under this arrangement.

6.6 MEMORY UTILIZATION

A map of memory for the new PILOT interpreter is contained in Figure 6.1. Digital Group Systems operating under PHIMON require memory from E000H to E7FFH for PHIMON root code and overlays. A ROM is located from 0000H to 00FFH for dead-starting the system and for limited display



Note 1: Upper limit is a function of available memory up to a maximum of DFFFH. This maximum provides 51,968 decimal bytes of lesson storage.

FIGURE 6.1 Z80 PILOT MEMORY ALLOCATION

I/O routines.

PILOT requires that the ROM be disabled prior to running because PILOT uses the top of the ROM area for stack. This configuration for PILOT will operate in a system as small as 8K bytes (counting the high memory PHIMON space). This would leave only 768 decimal bytes for lesson and string variable storage. In a 10K system, 2816 decimal bytes would be available for lesson and string variable storage. This 10K system represents the minimum level memory that the author recommends.

Adapting PILOT to various size memory configurations is very simple. Bytes 0360H and 0361H (low address, high address) should be set to the top of the PILOT lesson storage address.