To the Graduate Council:

I am submitting herewith a thesis written by Shu-Wen Chiu entitled "String Software Package and Editor." I recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Michael Moshell, Major Professor

We have read this thesis
and recommend its acceptance:

David L. Matuszek

Accepted for the Council:

Vice Chancellor
Graduate Studies and Research

STRING SOFTWARE PACKAGE

AND EDITOR

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Shu-Wen Chiu

August 1980

3044388

DEDICATION

This thesis is dedicated

to my parents.

ACKNOWLEDGMENTS

The author wishes to express her deep appreciation to Dr. J. Michael Moshell for his suggestion of the topic, and for his time, concern, and guidance in all phrases of this thesis. Appreciation is also expressed to Dr. David W. Straight and Dr. David L. Matuszek for their willingness to serve on this committee.

Special gratitude is expressed to Mr. Sau Fai wong for his comments, support, and encouragement. Finally, the author would like to thank Miss Huei-Huan Huang, Mr. Jiashann Chang, and her sister for their encouragement.

ABSTRACT

PL/M is a high-level, portable language which is widely used for microcomputers. This language is designed to encourage well-structured programs and to detect a group of programming errors at compile-time. It is also easy to learn, understand, and use; however, it has no string handling facilities. Therefore, to provide the character data type, some string handling operators, and built-in functions without having to change the compiler is the aim of this thesis. In addition, an editor, a version of the TECO editor, is developed for the INTEL 8080 microcomputer with CP/M. The main purpose of this editor is to demonstrate the above string software package.

The string software package and editor are written in the PL/M language, and an interface program for the string software package is written in 8080 assembly language.

It is assumed that the reader is somewhat familiar with PL/M (McCracken, 1978).

TABLE OF CONTENTS

CHAPTER		PAGE
1.	INTRODUCTION.	1
2.	STRING PACKAGE.	4
	2.1 Introduction	4
	2.2 Clement Wong's I/O Package	5
	2.3 The Character Data Type.	7
	2.4 String Operators	9
	2.5 String Built-In Functions.	13
	2.6 Error Messages	15
	2.7 An Example	16
	2.8 String Software Package v.s. PL/I String Facilities	20
3.	EDITOR.	23
	3.1 Introduction	23
	3.2 How It Works	24
	3.3 Commands	25
	3.4 Storage Organization	28
	3.5 Error Messages	30
	3.6 An Example	31
4.	IMPLEMENTATION.	34
	4.1 The Procedure Linkage Problem and its Solution .	34
	4.2 Compiler Problems and Solutions.	36
	4.3 Down-Loading Problems and Solutions.	37
	4.4 How to Recompile the Editor?	39
	4.5 Memory Organization for the String Package and Editor	41

5. CONCLUSION AND EVALUATION 45

5.1 Conclusion and Evaluation 45

5.2 Future Projects 46

LIST OF REFERENCES. 48

VITA. 51

CHAPTER 1

INTRODUCTION

PL/M is a high-level programming language used in a wide variety of systems and applications work with microcomputers. It is a highly portable language, and easy to learn, understand , and use. However, PL/M cannot handle character strings which are longer than two characters. PL/M also has no string operators or built-in string functions. The purpose of this thesis is to design a complete string software package in order to make PL/M more powerful.

In general, there are two reasons for designing this string system in a way which is in accordance with PL/I string-handling facilities.

(1) The structure of PL/M is similar to that of PL/I , and both languages are well-structured languages.

(2) PL/I has many varieties of string-handling facilities.

Some PL/I facilities which are not included in this software package are as follows:

(1) Two forms of declaration, the fixed-length string and the varying-length string, are permitted in PL/I. While using this software package, only varying-length strings are allowed.

(2) The comparison operation, one of the important string facilities , is not included in this package.

This thesis has two parts: string software package and editor. Both string software package and editor are written in the PL/M language for the INTEL 8080 microprocessor with the CP/M operating system. An interface program, written in assembly language, is used to pass parameters to subprograms. This is necessary because there is no linking loader for PL/M under CP/M.

This string package can be partitioned into three categories. In the first part, the character data type is built. In the second part, there are operations such as moving of strings, decomposition of strings into substrings, selective placement of strings, construction of new strings by concatenation , which are discussed separately. The third part consists of built-in string functions INDEX, VERIFY, and CHAR$LENGTH.

The editor described here is a version of the TECO editor, and is used to demonstrate the above string software package. Basically, it can handle the operations - insert, delete, search a specified string, etc. The linked list approach is used to manipulate storage space.

The next chapter provides detailed instructions for using this string package. This package is compared with PL/I string handling facilities, and some disadvantages are shown there. Chapter 3 introduces a text editor called PTECO. Its commands and storage organization are fully explained. Chapter 4 describes implementation problems, namely the procedure linkage problem, the down-loading

problem, and the compiler problem, along with their solutions. In the last chapter, an evaluation of this thesis and some future projects are presented.

CHAPTER 2

STRING PACKAGE

2.1 INTRODUCTION

The PL/M language has only two kinds of data types, BYTE and ADDRESS. Owing to the restriction of the data types, PL/M cannot handle strings of length longer than two characters. Moreover, some string-handling facilities like the string operators and built-in functions have not been developed in the PL/M language. Therefore, in order to provide extensive string-handling facilities for the PL/M language , this string package was designed.

Basically, the data structure of this string system is to use an array and sequential mapping. Why do we choose sequential storage instead of a character linked list? Several reasons are listed below:

(1) The sequential mapping is the simplest data structure, and can easily be applied in the various problems.

(2) It is more convenient for us in implementing the string operators and string built-in functions.

(3) The input data have already been stored in contiguous memory by using the PL/M I/O package (Wong, 1980).

(4) In order to use the PL/M I/O package, it is necessary to store the output character string in sequential memory locations.

(5) Certainly each character has the same length, and that is consistent with the fundamental nature of the sequential data structure.

In this string system, the subroutine 'CHAR' is used to declare the character data type, and further string operators and built-in functions are provided by ASSIGN, SUBSTR, REPEAT, CHAR$LAST, CONCATENATE, TRANSLATE, INDEX, VERIFY, and CHAR$LENGTH. In the next section, the string part of Clement Wong's I/O package is briefly described.

In the description below, assume the following special indicators:

'!' - indicates a special character '80H' which is used when declaring a variable of the character data type. It marks the end of the memory area reserved for the string.

'_' - indicates an unknown character.

'$' - indicates the current end of a variable length string of characters.

2.2 CLEMENT WONG'S I/O PACKAGE

This package provides the ability to read both numerical and string data and print out the results. Here the input and output format of this package is briefly explained. The format of this IO function is:

```
CODE = IO ( IOFLAG, .'AO', .X ) ;
```

The specifications of the above format consist of the following:

(1) CODE, a flag which is either 255 or 0, indicates whether an error is detected. CODE = 0 indicates that an operation is successful while CODE = 255 indicates a failure.

(2) IOFLAG is either 0 or 1 to represent input or output.

(3) 'AO' is the free format for input and output of strings.

(4) .X indicates the starting address of an array X which is used to store the input (or output) string.

(5) Input strings come from an external peripheral like a teletype , and a character '$' follows the input string to indicate the termination of the input string.

This IO function is at absolute address '2500H'. In order to communicate with the user's PL/M program, an interface routine containing a 'GO TO' statement is provided. That takes the form

```
DECLARE IO$ENTRY LITERALLY '2500H';
IO: PROCEDURE ( A, B, C ) BYTE ;
        DECLARE A BYTE, (B,C) ADDRESS ;
        GO TO IO$ENTRY ;
END IO ;
```

A simple example is the following, which reads in a strisg and echos it.

```
DECLARE  IN LITERALLY '0' ,
         OUT LITERALLY '1' ,
         RET BYTE, X(6) BYTE ,
```

```
        IO$ENTRY LITERALLY '2500H' ;
/*  THIS IS AN INTERFACE ROUTINE FOR WONG'S I/O */
/*  ROUTINE, AND IT'S NEEDED TO COMMUNICATE WITH*/
/*  ANY EXTERNAL ASSEMBLY LANGUAGE SUBROUTINE   */
/*  OR PL/M PROGRAMS WHICH ARE SEPARATELY       */
/*  COMPILED                                    */
IO : PROCEDURE ( A, B, C ) BYTE ;
        DECLARE A BYTE ,
        ( B, C ) ADDRESS ;
        GO TO IO$ENTRY ;
END IO ;
DO ;
   /*   READ IN A STRING      */
   RET = IO ( IN, .'AO', .X ) ;
   /*   ECHO IT               */
   RET = IO ( OUT, .'AO', .X ) ;
END ;
EOF ;
THE INPUT STRING IS 'TODAY$'
THE OUTPUT STRING IS 'TODAY'
```

2.3 THE CHARACTER DATA TYPE

Subroutine 'CHAR' is used to declare a variable of the character data type. By using this routine and the above IO function, a variable of varying length, of character data type is reserved. The format of this routine is:

```
CALL CHAR ( .STRING, STRING$LENGTH ) ;
```

The description of this routine is the following:

(1) The first argument is an ADDRESS type and second one is BYTE type.

(2) STRING$LENGTH is the number of characters (including the special character (80H) at the end) in the string.

(3) STRING is an array to be declared as a character type variable , and the actual length of this variable is STRING$LENGTH-1 bytes.

One thing to be noticed is that the array STRING declared in a PL/M program cannot be smaller than STRING$LENGTH; otherwise, an unexpected error will appear during the execution.

In the following segment, a variable X is declared as a character data type.

```
DECLARE IN LITERALLY '0' ;
DECLARE X(10) BYTE ;
RET = IO ( IN, .'A0', .X ) ;
CALL CHAR ( .X, 10 ) ;
```

In the above segment, if a string 'TODAY' is read in by an IO function, then the result in the variable X is

'TODAY$----'

Where '-' indicates an unknown character.

After routine 'CHAR' executed, the contents of X are

'TODAY$---!'

Where '!' gives the maximum length of the character variable.

The absolute entry point of 'CHAR' routine is 590FH in the current version.

2.4 STRING OPERATORS

There are six operators which are ASSIGN, CONCATENATE, SUBSTR, CHAR$LAST, REPEAT, and TRANSLATE. As for the usage of each operator, that is indicated by the given subroutine name. For more details, the description of each operator is shown in the following.

(1) ASSIGN Operator

This operator is used to move the string from a character variable to another. Subroutine 'ASSIGN' is specially designed for this task, and is also called by the 'CONCATENATE', and 'REPEAT' subroutines. The format is

```
CALL ASSIGN ( .STRING1, .STRING2 ) ;
```

The specifications of this operator are shown below :

(1) Both arguments are of ADDRESS types.

(2) STRING1 and STRING2 are character strings.

(3) The content of STRING1 is assigned to STRING2.

Assuming the contents of STRING1 are 'TODAY$--!', and the length of STRING2 is 4 -- that is '----!'. The result in STRING2 is

```
'TODA!'
```

(2) CONCATENATE Operator

This operator is used to build a new string by combining two small strings. Subroutine 'CONCATENATE' is implemented, and also called by subroutine 'REPEAT'. The

format is

CALL CONCATENATE (.STRING1, .STRING2, .STRING3) ;

The specifications of this operator are as follows :

(1) Three arguments are ADDRESS types.

(2) STRING1, STRING2, and STRING3 are character strings.

(3) The contents of STRING1 and STRING2 are combined into STRING3.

Assuming that

STRING1 'GOOD $!'

STRING2 'LUCK!'

and the length of STRING3 is 10, what results after executing the above statement? By dumping the memory, we find that the contents of STRING3 are

'GOOD LUCK$!'

(3) SUBSTR Operator

This operator is used to refer to a portion of the string. Subroutine 'SUBSTR' performs this operation, and is also called by 'CHARSLAST'. The format is

CALL SUBSTR (.STRING1, I, J, .STRING2) ;

The specifications of this operator are shown below:

(1) Argument 1 and 4 are ADDRESS types, and argument 2 and 3 are BYTE types.

(2) STRING1 and STRING2 are character strings, but both I and J are fixed integers.

(3) STRING2 contains the substring of STRING1 beginning at character 'I' and ending with character 'I+J-1'.

For example, assume that

STRING1 'GOOD LUCK!'

I 2

J 4

and the length of STRING2 is 2. So, the result in STRING2 is

'OO!'

(4) CHAR$LAST Operator

This operator refers to the substring which contains the last few characters of the string. The format is

CALL CHAR$LAST (.STRING1, I, .STRING2) ;

The specifications of this operator are the following:

(1) Argument 1 and 3 are ADDRESS types, but argument 2 is BYTE type.

(2) STRING1 and STRING2 are character strings, but I is a fixed integer.

(3) STRING2 has the last 'I' characters of STRING1.

(4) This operation is the same as the subroutine SUBSTR(.STRING1, CHAR$LENGTH(.STRING1)-I+1, I, .STRING2) ; Where CHAR$LENGTH is a string built-in function, described later.

Assuming that

STRING1 'GOOD LUCK!'

I 4

and the length of STRING2 is 10. The result is

STRING2 'LUCK$-----!'

(5) REPEAT Operator

A new string is produced by the original string which is repeated a few times. Subroutine 'REPEAT' is designed for this operator. The format is

```
CALL REPEAT ( .STRING1, I, .STRING2 ) ;
```

The specifications of this operator are the following:

(1) Argument 1 and 3 are ADDRESS types, but argument 2 is BYTE type.

(2) STRING1 and STRING2 are character strings, and I is a fixed integer.

(3) STRING2 is STRING1 concatenated with itself 'I' times.

Assuming that

```
STRING1          'ABC$!'
   I                2
```

and the length of STRING2 is 10. The result is

```
STRING2          'ABCABCABC$!'
```

(6) TRANSLATE Operator

This operator is used to replace the part of the specified string. Subroutine 'TRANSLATE' is designed for this operation. The format is

```
CALL TRANSLATE ( .STRING1, .STRING2, .STRING3,
                 .STRING4 ) ;
```

The specifications of this operator are the following:

(1) All the arguments are ADDRESS types.

(2) All the variables are character strings.

(3) STRING4 is a string identical to STRING1 except that any character of STRING1 which is also in STRING3 is

replaced by the corresponding character in STRING2.

(4) If the length of STRING2 is too short, then blanks are padded at the end.

Assuming that

STRING1	'ABCDE$!'
STRING2	'12$!'
STRING3	'BDF!'

and the length of STRING4 is 5. The result is 'A1C2E!'.

In this package, the entry points for the above operators are given below:

ASSIGN	5912H
CONCATENATE	5900H
SUBSTR	5906H
CHAR$LAST	5909H
REPEAT	5903H
TRANSLATE	590CH

2.5 STRING BUILT-IN FUNCTIONS

VERIFY, INDEX, and CHAR$LENGTH described here are the last part of the string software package. The detailed operation of the above functions is explained in this section.

(1) VERIFY Function

The format of this function is

```
RESULT = VERIFY ( .STRING1, .STRING2 ) ;
```

The description of the function is the following :

(1) Both arguments are ADDRESS types, and point to

character strings.

(2) RESULT is a fixed integer indicating the position of the first character in STRING1 which is not in STRING2.

(3) If all the characters in STRING1 are also in STRING2, the result is 0.

Assuming that

STRING1 'ABCDE!'

STRING2 'BCDE!'

and RESULT is 1.

(2) INDEX Function

This function is used to search a given string. The format of this function is

RESULT = INDEX (.STRING1, .STRING2) ;

The description of the function is the following:

(1) Both arguments are ADDRESS types and point to character strings.

(2) RESULT is a fixed integer which gives the leftmost position in STRING1 where STRING2 begins.

(3) If STRING2 doesn't appear as a substring of STRING1, the RESULT is zero.

Assuming that

STRING1 'ABCD!'

STRING2 'BCD!'

and RESULT is 2.

(3) CHAR$LENGTH Function

This function is to return an actual length for a given string, and is called by all the routines and functions in

this package.

The format of this function is

RESULT = CHAR$LENGTH (.STRING) ;

The description of this function is the following:

(1) The given string is a character string.

(2) RESULT is a fixed integer indicating the current length of the specified string.

For example:

X	'ABCD$--!'
Y	'ABC!'
Z	'ABCDEF$!'

By using this function, the lengths of X, Y, and Z are 4, 3, 6 respectively.

In this system, the entry points for the above functions are given below:

VERIFY	5915H
INDEX	5918H
CHAR$LENGTH	591BH

2.6 ERROR MESSAGES

Error messages are printed via CP/M during the execution of this package. While the error is detected, the appropriate error message comes with three stars (***). This form is consistent with the error messages from the PL/M I/O package which is written by Clement Wong.

The error messages provided by this string package are as follows:

1. *** NO CHAR DATA TYPE DECLARATION ***
2. *** STRING LENGTH TRUNCATED ***
3. *** 'REPEAT' ARG1 IS TOO LARGE ***
4. *** 'SUBSTR' STRING POSITION > STRING LENGTH ***
5. *** 'SUBSTR' ILLEGAL FORMAT ***
6. *** 'CHAR$LAST' ARG2 > STRING LENGTH ***

2.7 AN EXAMPLE

This section provides a sample program which uses a selected group of string operators and built-in functions. This program is to reverse a string and print it out.

100H :

```
   /*   TO REVERSE A STRING AND PRINT IT OUT       */
   /*   FOR EXAMPLE :                              */
   /*       THE INPUT STRING  IS '123456'          */
   /*       THE OUTPUT STRING IS '654321'          */
   DECLARE IN              LITERALLY '0' ,
           OUT             LITERALLY '1' ,
           IO$ENTRY        LITERALLY '2500H' ,
           CHAR$ENTRY      LITERALLY '590FH' ,
           SUBSTR$ENTRY    LITERALLY '5906H' ,
           LENGTH$ENTRY    LITERALLY '591BH' ,
           CONCAT$ENTRY    LITERALLY '5900H' ;
   DECLARE IN$STRING(11)   BYTE ,
           OUT$STRING(11)  BYTE ,
           CHAR$STRING(2)  BYTE ,
         (RET, IN$LENGTH)  BYTE ;
```

```
        /*   INTERFACE ROUTINE FOR I/O ROUTINE  */
IO : PROCEDURE ( A, B, C ) BYTE ;
        DECLARE A BYTE, ( B, C ) ADDRESS ;
        GO TO IO$ENTRY ;
END IO ;
        /*   INTERFACE ROUTINE FOR CHAR ROUTINE */
CHAR: PROCEDURE ( A, B ) ;
        DECLARE ( A, B ) ADDRESS ;
        GO TO CHAR$ENTRY ;
END CHAR ;
        /*   INTERFACE ROUTINE FOR SUBSTR       */
        /*   ROUTINE                            */
SUBSTR : PROCEDURE ( A, I, J, B ) ;
        DECLARE ( A, B ) ADDRESS ,
                ( I, J ) BYTE ;
        GO TO SUBSTR$ENTRY ;
END SUBSTR ;
        /*  INTERFACE ROUTINE FOR CHAR$LENGTH   */
        /*  ROUTINE                             */
CHAR$LENGTH : PROCEDURE ( A ) BYTE ;
        DECLARE A ADDRESS ;
        GO TO LENGTH$ENTRY ;
END CHAR$LENGTH ;
        /*   INTERFACE ROUTINE FOR CONCATENATE  */
        /*   ROUTINE                            */
CONCATENATE : PROCEDURE ( A, B, C ) ;
        DECLARE ( A, B, C ) ADDRESS ;
```

```
        GO TO CONCAT$ENTRY ;
END CONCATENATE ;
        DO WHILE '1' ;
           /*   READ IN AN INPUT STRING          */
           RET = IO ( IN, .'AO', .IN$STRING ) ;
           /*   DECLARE IN$STRING, OUT$STRING    */
           /*   AND CHAR$STRING AS CHARACTER     */
           /*   TYPES                            */
           CALL CHAR ( .IN$STRING, 11 ) ;
           CALL CHAR ( .OUT$STRING, 11 ) ;
           CALL CHAR ( .CHAR$STRING, 2 ) ;
           /*   GET THE LENGTH OF INPUT STRING   */
           IN$LENGTH = CHAR$LENGTH ( .IN$STRING ) ;
           /*   INITIALIZE OUT$STRING            */
           OUT$STRING(0) = '$' ;
           DO WHILE ( IN$LENGTH > 0 ) ;
              /*   EXTRACT A CHARACTER FROM      */
              /*   A SPECIFIED POSITION          */
              CALL SUBSTR ( .IN$STRING, IN$LENGTH, 1,
                            .CHAR$STRING ) ;
              /*   PUT IT INTO OUT$STRING        */
              CALL CONCATENATE ( .OUT$STRING, .CHAR$STRING,
                                 .OUT$STRING ) ;
              /*   GET ANOTHER POSITION          */
              IN$LENGTH = IN$LENGTH - 1 ;
           END ;
           RET = IO ( OUT, .'AO', .OUT$STRING ) ;
```

```
        END ;
EOF
```

Assuming that the input strings are as follows:

```
GOOD LUCK$
GOOD BYE$
```

The output strings are

```
KCUL DOOG
EYB DOOG
```

If only one input string is to be processed, then the DATA statement or the INITIAL statement can be used to hold the input string. Therefore, in this example, the CALL statement of IO and CHAR along with their interface procedures can be replaced by the following DATA statements

```
DECLARE IN$STRING DATA ( 'GOOD LUCK$',80H ) ;
DECLARE OUT$STRING DATA ('$          ',80H) ;
DECLARE CHAR$STRING DATA ( ' ',80H ) ;
```

or the following INITIAL statements.

```
DECLARE IN$STRING(11) BYTE
        INITIAL ( 'GOOD LUCK$',80H ) ;
DECLARE OUT$STRING(11) BYTE
        INITIAL ( '$          ',80H ) ;
DECLARE CHAR$STRING(2) BYTE
        INITIAL ( ' ',80H ) ;
```

From the above program, three points have to be noticed.

(1) The input data must be read into the variable before we declare it as a character data variable.

(2) The special character '80H' occupies the last byte of a variable, so the maximum usable length of the character variable is the number of reserved bytes minus 1.

(3) A character '$' indicates the actual termination of the current string. Therefore, The first byte of the character variable contains a character '$', and that indicates the null string in the variable.

2.8 STRING SOFTWARE PACKAGE V.S. PL/I STRING FACILITIES

This string package is designed according to the PL/I string handling facilities. Basically, the differences between them are listed below:

(1) The character data type of PL/I exists at compile-time, but that of this string package is developed at running time.

(2) PL/I provides two types of character data: varying and fixed length types, but this string package only gives the varying length type.

(3) Data in PL/I can be got from a READ statement, an INITIAL statement, or assigned to a character variable directly, but this string package accepts strings from the keyboard, an INITIAL statement, a DATA statemant, or a dot operator like the form (.'constant-list').

(4) PL/I can recognize the signs - equal sign and two vertical bars - as the assignment operator and concatenate operator, but in this package two subroutines called ASSIGN and CONCATENATE are used instead.

(5) In PL/I, the third argument of the function 'SUBSTR' is optional. In this system, subroutine 'SUBSTR' is designed for the same operation but the third argument is required.

Furthermore, some disadvantages in using this string package are the following:

(1) It is less convenient to add an interface procedure and a statement in order to achieve a simple operation. For example, PL/I has the statement

```
A = B ;
```

but in this string system an interface routine and a statement are used to achieve the same operation. Those are:

```
DECLARE ASSIGN$ENTRY LITERALLY '5912H' ;
ASSIGN : PROCEDURE ( X, Y ) ;
        DECLARE ( X, Y ) ADDRESS ;
        GO TO ASSIGN$ENTRY ;
END ASSIGN ;
CALL ASSIGN ( .B, .A ) ;
```

(2) It is more complicated to declare the character data type. In PL/I, the character data type can be handled by a 'DECLARE' statement, but in this package an interface routine 'CHAR' and a CALL statement must be added to the program. If several variables are declared, then a lot of CALL statements appear in the program.

(3) In PL/I, a lot of errors can be detected during compiling time, but in this string package the same errors

are detected during running time.

This string package is implemented without changing any compiler code, so it is impossible to design it as well as PL/I string facilities. The only way to remove those disadvantages is to implement the character data type in the PL/M compiler.

CHAPTER 3

EDITOR

3.1 INTRODUCTION

This text editor, here called PTECO, is used to create and alter CP/M source files. The string package described in the preceding chapter is also part of the editor; the main purpose for designing this editor is to demonstrate the string package.

There are some reasons for designing PTECO to resemble the TECO editor, which are shown below:

(1) TECO is extensively used in a variety of systems.

(2) TECO is an editor which is easy to learn, understand, and use.

The basic operations of this editor are summarized in the following:

(1) At the beginning of an editing session, the editor reads in the text from the disk automatically, if the file already exists.

(2) Two imaginary pointers are the line pointer and the character pointer. The line pointer can be moved up and down a specified number of lines. The character pointer can be moved forwards and backwards a specified number of characters.

(3) The editor can delete a specified number of lines or characters by moving the line or character pointer.

(4) It can also insert a few lines or characters after

the line or character pointer.

(5) It can type a specified number of lines ahead of or behind the line pointer.

(6) At the end of an editing session, the editor restores the text from memory to disk and closes the file.

This package is written in the PL/M language.

3.2 HOW IT WORKS

PTECO is initiated by typing the following command:

A> PTECO cr

PTECO is loaded into TPA (the Transient Program Area) and executed. The following 'prompt' is printed:

FILENAME :

While seeing the above prompt, we type the filename and filetype (extension) and enter a carriage return. 'Filename' can consist of up to eight letters and numbers , and 'filetype' can consist of up to three letters or numbers. The filetype is optional. Next, one of the messages (1) OLD FILE EDITED (2) NEW FILE INPUT is printed. An asterisk indicates the command mode, and commands can be accepted and executed in this mode. Finally, the E and Q commands cause the text to be restored to disk.

PTECO will not allow you to type in more than one command at a time; if you do otherwise, the appropriate error message will show on the screen. The memory buffer is a block of 8 kilobytes memory which is used to store the source file and the revised file. The file is stored by

lines, which are separated by a carriage return and a line feed. A line linked list is used to handle the order of the lines. A character pointer points somewhere between two characters, at which point commands are executed. Usually the line pointer and character pointer belong to different lines, and the line pointer points to the line which precedes the line containing the character pointer. But, one execption is in the first line. Here, it is allowed to have the character pointer and line pointer at the same line. The user is not actually aware of the line pointer. All operations are based on the character pointer. In the memory, the carriage return and line feed are considered as two characters.

3.3 COMMANDS

According to the features of the commands, all the commands which are available in this editor can be divided into two categories. One is text transfer commands, such as E command and Q command. Another is memory operation commands such as C command, D command, I command, K command, L command, T command, and S command.

While examining the commands carefully, we found two general forms which represent the whole set of commands.

The first form is

snC cr

Where s indicates a sign, a plus or minus can be given. If the sign is not mentioned, a plus sign is assumed. The n

is an integer number, and its range is from 0 to 65535. If it is ignored, then 1 is assumed. The C is a single letter for invoking the different memory operations which are C command, D command, L command, T command, and K command. Finally, a carriage return (cr) which follows the selected command should be supplied, and the appropriate routine will be executed.

The second form is

C cr

This form has two different types. In the first type, at least one string is provided after the above form. In the second type, the designated command is executed immediately.

The S command and the I command are included in the first type, and at the end of the input string the ESC key has to be punched and a character '$' echoed on the screen. The second type contains the E and Q commands.

The operation for each available command is separately described below.

(1) C command form : snC

The C command is used to move the character pointer forwards and backwards, and also causes the line pointer to be moved up and down when the end of a line is crossed.

(2) D command form : snD

The D command is used to delete the characters ahead of or behind the character pointer. Meanwhile, the line pointer is automatically moved. The character pointer

points right after the deleted characters.

(3) E command form : E

The E command is used to leave from the editor and save the revised text in the disk file. Moreover, a BAK file which contains the old text is provided on disk.

(4) I command form : I

The I command is used to insert a text, a few lines or characters after the character pointer. The character pointer always points right after the inserted characters. if a few lines are inserted, then the line pointer points to the last inserted line.

(5) K command form : snK

The K command is used to delete lines before or after the line pointer. if the character pointer is in the middle of the line, then the effects are (1) if the K is supplied, then the rest of this line is deleted and the beginning of this line is connected with the next line. (2) If OK is supplied, then the beginning of this line is deleted. (3) If -K is supplied, then the beginning of this line and the previous line are deleted.

(6) L command form : snL

The L command is used to move the line pointer up and down. If the character pointer is in the middle of the line, then the effects are (1) If L is supplied, then the character pointer is moved to the beginning of the next line. (2) If OL is supplied, then the character pointer is moved to the beginning of this line. (3) If -L is supplied,

then the character pointer is moved to the beginning of the previous line.

(7) Q command form : Q

The Q command is used to quit editing and make no changes to the file. If this command is provided for a new file, then the file doesn't appear in the directory.

(8) S command form : S

The S command is used to search for a given string forward from the character pointer until the string is found or the text ends. if the given string is found, then the character pointer points right after the searched string; otherwise, an error message 'SEARCH FAILS' appears.

(9) T command form : snT

The T command is used to type a specified number of lines before or after the line pointer. If the character pointer is in the middle of the line, then the effects are (1) if T is supplied, then the rest of line is typed. (2) If OT is supplied, then the beginning of the line is typed. (3) if -T is supplied, then the previous line and the beginning of this line are typed.

3.4 STORAGE ORGANIZATION

The manipulation of memory storage is an important point for writing a big program. In this package, a linked list with dynamic memory allocation is used.

For designing this editor, two linked lists are set up which are the line linked list and the block memory linked

list. The former is a doubly linked list which is used to keep track of the inserted or deleted lines, handle the line up and down, and record the starting address of each line. These records take the form:

LEFT LINK	STARTING ADDRESS FOR EACH LINE	RIGHT LINK

The block memory linked list is a single linked list which is used to keep the freed memory spaces and link them together. It takes the form:

LINK	MEMORY STARTING ADDRESS	MEMORY SIZE

Moreover, if two freed memory blocks are continuous storage areas, then they are collected to form a larger block of memory. For example, assuming that two freed memory blocks have already existed in the block memory linked list, the form looks like:

	LINK	MEMORY STARTING ADDRESS	MEMORY SIZE
(1)	2	1000	100 BYTES
(2)	0	3000	200 BYTES

While freeing 200 bytes from memory location 800, the linked list is changed into the following:

	LINK	MEMORY STARTING ADDRESS	MEMORY SIZE
(1)	2	800	300 BYTES
(2)	0	3000	200 BYTES

When a block of memory is required from this linked list, the 'First Fit Method' is used to search for a block

memory of the appropriate size. What's the first fit method? It is to return the first memory location which is available of the required memory size. If the searched memory size is larger than the required memory, then the rest of the block is left in the linked list and the starting memory address points to the beginning of the rest of available memory. In the above example, when the memory size 200 bytes is requested, the linked list is changed as follows.

	LINK	MEMORY STARTING ADDRESS	MEMORY SIZE
(1)	2	1000	100 BYTES
(2)	0	3000	200 BYTES

3.5 ERROR MESSAGES

The errors considered in the editor are as follows:

(1) ***ERROR*** FILE NOT PRESENT

(2) ***ERROR*** TRANSFER ERROR

(3) ***ERROR*** NO DIRECTORY SPACE

(4) ***ERROR*** CLOSE ERROR

(5) ***ERROR*** DISK WRITE ERROR

(6) ***ERROR*** ILLEGAL FILENAME

(7) ***ERROR*** ILLEGAL COMMAND

(8) ***ERROR*** TOO MANY LINES

(9) ***ERROR*** NO MEMORY LINK SPACE

(10)***ERROR*** NO MEMORY SPACE AVAILABLE

(11)***ERROR*** SEARCH FAILS

Error 4 and error 5 come while executing a text transfer command. If errors 8, 9 10 are encountered, then

the source document is too large. The user must reduce its size or recompile the editor. We will discuss how to recompile the editor in the next chapter.

3.6 AN EXAMPLE

This example is to show how to use the PTECO editor. All the commands are used in this example. The initial text contains two errors: 'BASICK' and 'ASED' to be corrected.

```
A>PTECO cr
FILENAME : TEST.DOC cr
NEW FILE INPUT
*I cr
PART 1 : HOW TO USE THE VIDEO-TAPE AND GUIDE
PART 2 : BASICK ORIENTATION
PART 3 : HIGHLIGHTS OF THE VIDEO-TAPE
PART 4 : SOME FREQUENTLY ASED QUESTIONS
PART 5 : A REVIEW OF GUIDELINES FOR THE TUTOR
$ ( ESC key typed, '$' echoed )
*-100L cr
*100T cr
PART 1 : HOW TO USE THE VIDEO-TAPE AND GUIDE
PART 2 : BASICK ORIENTATION
PART 3 : HIGHLIGHTS OF THE VIDEO-TAPE
PART 4 : SOME FREQUENTLY ASED QUESTIONS
PART 5 : A REVIEW OF GUIDELINES FOR THE TUTOR
*+L cr
*T cr
```

```
PART 2 : BASICK ORIENTATION
*+14C cr
*+D cr
*OL cr
*T cr
PART 2 : BASIC ORIENTATION
*S cr
ASED$
*T cr
 QUESTIONS
*OT cr
PART 4 : SOME FREQUENTLY ASED
*-4D cr
*I cr
ASKED$
*OL cr
*T cr
PART4 : SOME FREQUENTLY ASKED QUESTIONS
*+100L cr
*-K cr
*E cr
A> TYPE TEST.DOC
PART 1 : HOW TO USE THE VIDEO-TAPE AND GUIDE
PART 2 : BASIC ORIENTATION
PART 3 : HIGHLIGHTS OF THE VIDEO-TAPE
PART 4 : SOME FREQUENTLY ASKED QUESTIONS
```

From the above example, a general rule for searching a

specified string is summarized below:

(1) Move the line pointer to the line containing the specified string, and move the character pointer to point to the beginning of the string.

(2) Use the S command to search the specified string; the character pointer will point right after the searched string.

CHAPTER 4

IMPLEMENTATION

4.1 THE PROCEDURE LINKAGE PROBLEM AND ITS SOLUTION

PL/M exists in two main forms; one is as part of the ISIS operating system on the Intel INTELLEC 80 microcomputer development system, and the other is as a stand-alone cross-compiler, written in FORTRAN. It is the latter that we have. Because it is a cross-compiler, it has no linkage software with it, but just produces "naked" hex-code files. When several procedures were compiled separately, the procedure linkage is a considerable problem.

An interface procedure must be provided for a PL/M program to communicate with an assembly subroutine or a PL/M subroutine that was separately compiled. As a rule, the interface procedure only contains one jump statement 'GO TO'. The following segment is an interface routine for the INDEX function.

```
DECLARE INDEX$ENTRY LITERALLY '5918H' ;
INDEX :  PROCEDURE ( A, B ) BYTE ;
         DECLARE ( A, B ) ADDRESS ;
         GO TO INDEX$ENTRY ;
END INDEX ;
```

INDEX$ENTRY is the entry point of the INDEX assembly language subroutine.

PL/M uses the following conventions for passing parameters: (Intel, 1975)

(1) A single byte parameter is passed in register C. A single address parameter is passed in register B (high order byte) and C (low order byte).

(2) If there are two parameters, the first is passed as described above; the second is passed in register D (high order byte, if any) and E (low order byte).

(3) When there are more than two parameters, the last two are sent as described above, and the remainder are assigned directly, prior to the actual call.

In accordance with the above conventions, in passing two or less parameters, it is no problem to hold the value in the register pairs and enter them into the assembly language subroutine. If there are more than two parameters, their values are assigned directly to the variables of the interface procedure. Then, when the program jumps into the assembly subroutine, the assigned values are lost. How can we avoid the assigned values being lost? That problem can be solved by the following steps.

First, by tracing the program, we found that the register pair H, L holds the address of the last parameter, if the last parameter is a BYTE type. If the last parameter is an ADDRESS type, then the HL pair points to the high order byte of it. According to the parameters which are declared by BYTE or ADDRESS, we keep on decrementing the H, L pointer to get the address of the assigned value. For example:

```
DECLARE SUBSTR$ENTRY LITERALLY '5906H' ;
SUBSTR : PROCEDURE ( N, I, J, K ) ;
        DECLARE N ADDRESS ,
                ( I, J ) BYTE ,
                    K ADDRESS ;
```

```
        GO TO SUBSTR$ENTRY ;

    END SUBSTR ;
```

The following figure shows the parameters of the 'SUBSTR' routine and the location which the HL pair points to.

```
          ***********
1         *         *   N
2         *         *
          ***********
3         *         *   I
          ***********
4         *         *   J
          ***********
5         *         *   K
6         *         * -- H,L HOLDS THE ADDRESS HERE
          ***********
```

In this case, if we want to get the address of I, the statement

```
    DCX     H
```

should be executed three times. The address of N is accessed by executing the above statement four times. The H, L pair points to the high order byte of a 16-bit variable. The contents of the above addresses are stored in a temporary area by our interface program.

Second, we compiled the string package and found the actual addresses of argument 1 and argument 2, which correspond with "N" and "I" here. Then, we assigned the above stored values to these addresses respectively.

As described above, an interface program has been written to pass the parameters successfully.

4.2 COMPILER PROBLEMS AND SOLUTIONS

A problem is encountered when the PL/M program is too big to be handled by the PL/M compiler. Several error

messages are produced at compile-time. Generally speaking, the error messages can be summarized by the following:

(1) Symbol table overflow

(2) Virtual memory overflow

One way used to remove these error messages is to recompile the revised compiler which has a larger size of the symbol table and virtual memory. As for the steps which enlarge the size of the symbol table and virtual memory, those are provided with the PL/M compiler program which is available from Dr. Moshell. We will not discuss them here.

Another way to solve this problem is to separate the program into several smaller programs which can be run in the standard compiler version. In practice, that is usually more convenient for PL/M users in compiling programs. Although a lot of interface procedures will be needed in the program, this simple way is still suggested to handle this problem.

4.3 DOWN-LOADING PROBLEMS AND SOLUTIONS

The PL/M compiler is written in the FORTRAN language, and it can be run on any machine which supports this language and has enough memory. Memory requirements depend on the size of the PL/M programs being compiled. Here, DEC-10 is the only machine which has been set up to compile PL/M programs. Because of this, it is necessary to transfer

the Hex file into the 8080 microcomputer after compiling. H. R. Nayak's Communication Control Program, called TRADE, is used to doing this job. However, some errors occurred in using the 'TRADE' program. They were

(1) Explicit error

One or two characters in several lines are lost.

(2) Implicit error

Incorrect characters are passed in a few lines.

When any mistakes are in the Hex file, the file cannot successfully be entered via DDT (CP/M Dynamic Debugging Tool) which responds with the checksum error indicator '?'. The form is like:

```
A> DDT PTECO
   DDT VERS 1.4
   ?
```

How can we remove the mistakes from the Hex file? Two methods described in the following are used to solve the explicit error.

(1) In order to reduce the error possibilities in transferring the file, divide the Hex file into several smaller parts.

(2) A text editor is used to amend all incorrect parts of the Hex file.

When method(1) is chosen, PIP (CP/M Peripheral Interchange Program) is used to combine all the segments into a complete Hex file on the 8080 computer. It takes the form:

```
A> PIP
*A.HEX=X.HEX,Y.HEX,Z.HEX
```

The above form is to merge the X.HEX, Y.HEX, Z.HEX into a file called A.HEX. If we apply one of the above methods, the Hex file probably still can't be entered into DDT. We infer that the file has some implicit errors. Then, the following steps are suggested:

(1) In DDT, disassemble the Hex code into the assembly code.

(2) The binary search method is applied to find part of the disassembled codes which are different from the correct assembly file.

(3) An editor is used to correct the Hex file.

Sometimes, even if a small program is transferred, the 'TRADE' program cannot receive it successfully. Therefore, it is the safer way to change the error parts of the Hex file by using an editor, but it takes a lot of time to get a correct Hex file. It is necessary to revise or replace the 'TRADE' program.

4.4 HOW TO RECOMPILE THE EDITOR ?

There are three memory space-related conditions to cause this package to be recompiled.

(1) No line link space available

(2) No block memory link space available

(3) No memory space available

In the description below, please refer to Figure 2.

Memory Map for Editor.

In the first case, to enlarge the line linked storage space is necessary. There are two changes listed below:

(A) In the literal declarations of program 'EDIT2',

```
DCL             START  LITERALLY '6000H'
                RLINK$START    LITERALLY '6200H'
                LLINK$START    LITERALLY '6400H' ;
```

The locations '6000H', '6200H', '6400H' can be changed to any other available locations to provide more spaces.

(B) In the global declarations of program 'EDIT2',

```
DCL     LLINK BASED LLINK$PTR (250) ADDRESS,
                RLINK BASED RLINK$PTR (250) ADDRESS,
           LINE$ADDR BASED LINE$PTR (250) ADDRESS ;
DCL MAX$LINE ADDRESS INITIAL ( 250 ) ;
```

'250' is the maximum number of lines which will be consistent with the size given in the literal declaration.

(C) In the literal declarations of program 'EDIT1',

```
DCL     START LITERALLY '6000H' ;
```

'6000H' should be changed. This literal declaration has to conform to the definition of START in 'EDIT2'.

In the second case, it is necessary to enlarge the link size for the free block memory. The changes in program 'EDIT1' are as follows:

```
DCL     SIZE(50) BYTE,
                START$ADDR(50) ADDRESS,
                LINK(50) BYTE ;
DCL     MAX$NO BYTE INITIAL(50) ;
```

The number '50' can be changed into any number which the PL/M compiler can accept.

In the third case, to enlarge the size of memory is necessary. The changes are as follows:

(A) In the literal declarations of program 'EDIT2',

```
DCL     M$START LITERALLY '3000H' ;
```

Where M$START is the starting location for the memory buffer.

(B) In the global variable declarations of program 'EDIT2',

```
DCL MEM$SIZE ADDRESS  INITIAL(8100) ;
DCL MEM BASED M$PTR INITIAL(8100) ;
```

'8100' is the size of the memory buffer, and it can be changed into any number which can be accepted by the 8080 microcomputer. The changed memory size should start from the starting memory location M$START.

4.5 MEMORY ORGANIZATION FOR THE STRING PACKAGE AND EDITOR

The string package consists of two programs and can be used with the PL/M I/O package. Two programs are

1. STRING --- handle character data type and further string operators and string built-in functions.

2. INTFAC --- communicate between the string routine and the user's program.

The memory map of this package is shown in Figure 1.

The editor consists of four programs, two of those programs are STRING and INTFAC. The remainder are as

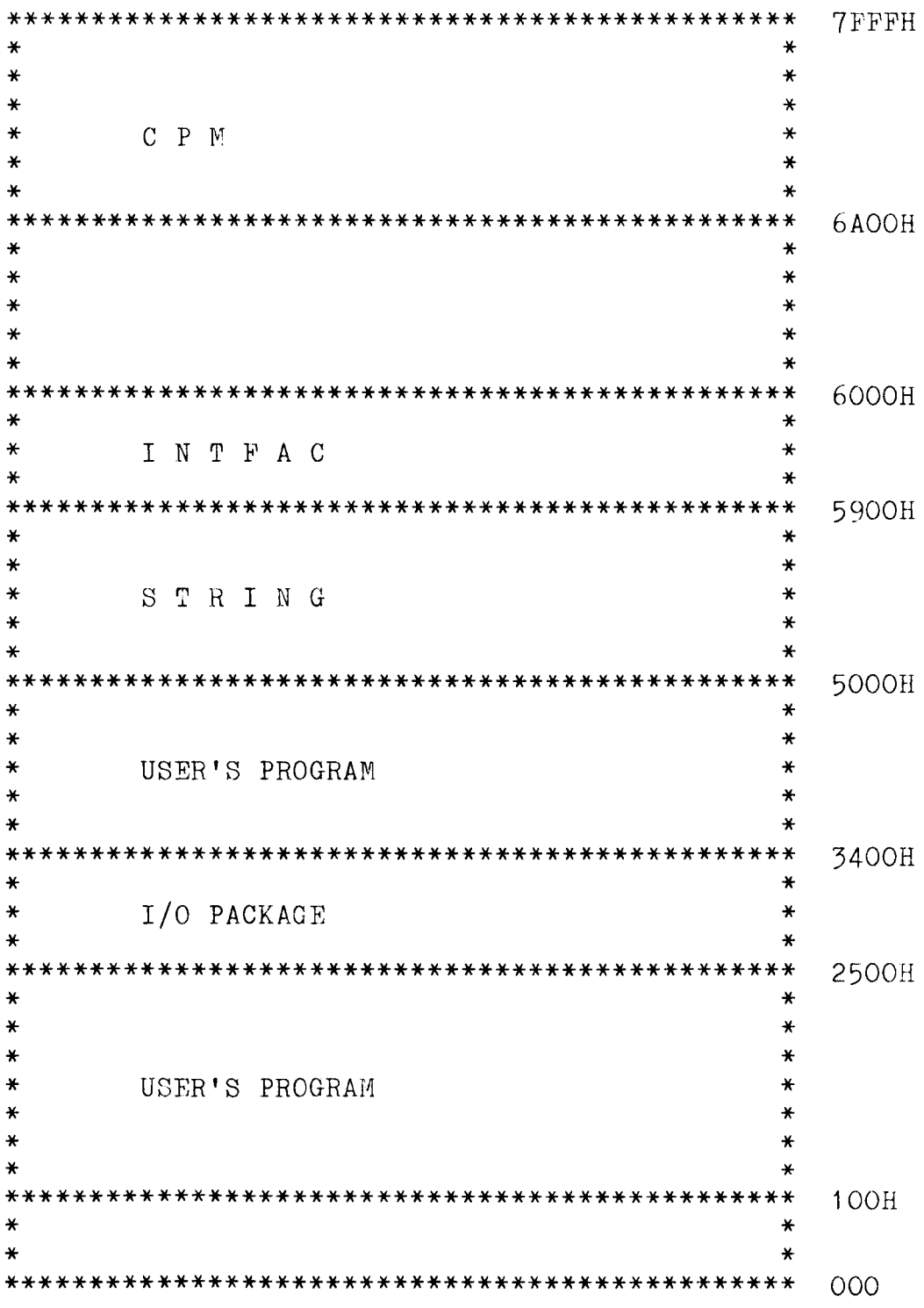

Figure 1. Memory Map for String Package

follows:

1. EDIT1 --- sets file control block, handles the freed memory block linked list , reads in a command, and handles the E command and Q commands.

2. EDIT2 --- the central part of this editor which handles all the commands except the Q and E commands , transfers the file into memory, and sets up the line linked list.

The memory map of this editor is shown in Figure 2.

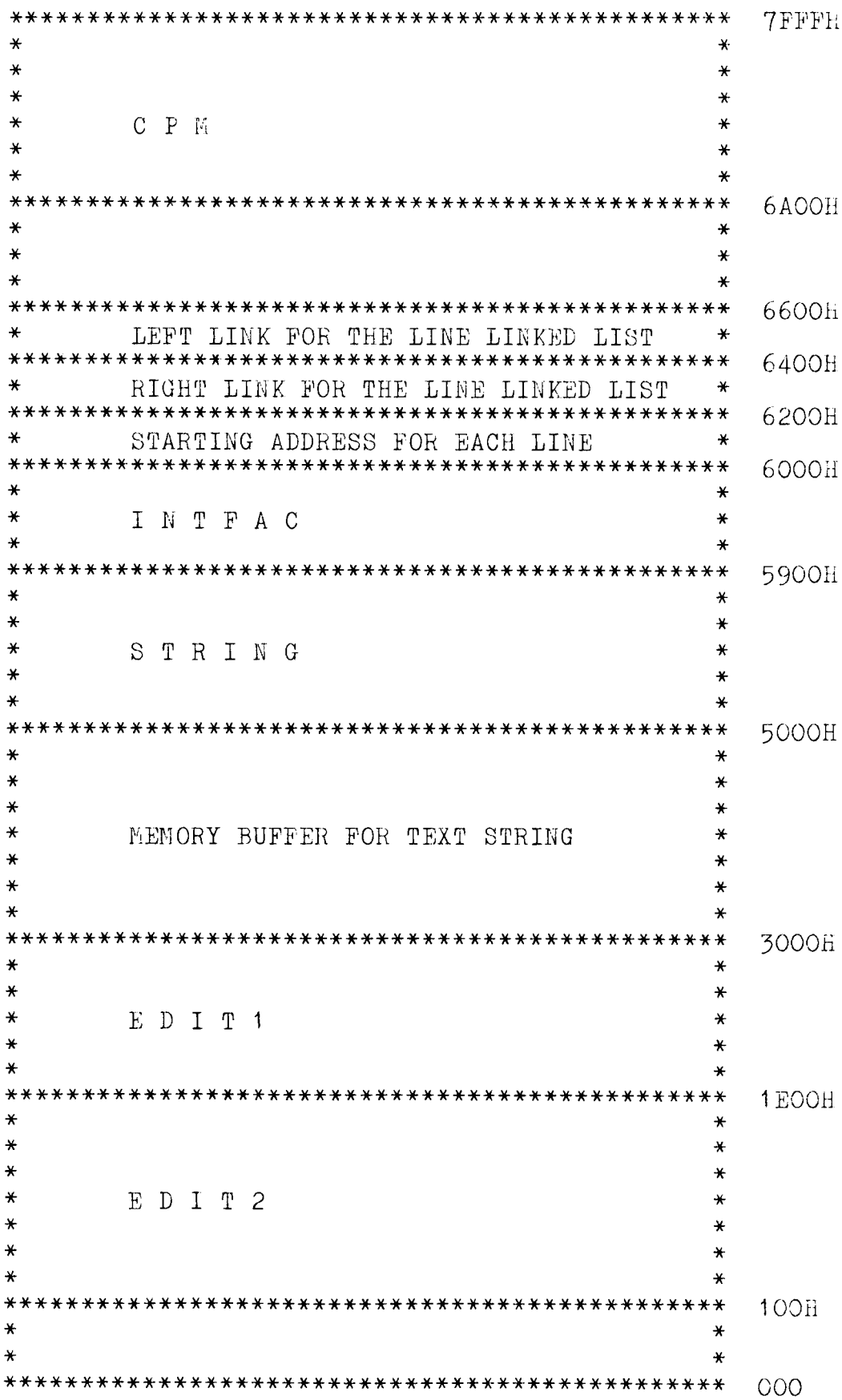

Figure 2. Memory Map for Editor

CHAPTER 5

CONCLUSION AND EVALUATION

5.1 CONCLUSION AND EVALUATION

This thesis consists of two parts. One is to design a string package for the PL/M language. Another is an editor which is used to demonstrate the PL/M string package. We tried to consider all cases in writing the PL/M string package and editor, but some problems happened while using it.

(1) As we know, PL/M can handle character strings whose length is no more than two, so a statement in PL/M language has the form

```
A = 'BC' ;
```

which is allowed. But, while using the PL/M string package, the above input string can not be recognized by the string operators and string built-in functions.

(2) Two statements usually used to read in data and declare the variable as a character data type are as follows:

```
/*  THIS STORES THE INPUT STRING IN AN  */
/*  ARRAY OF THE VARIABLE X             */
RET = IO ( IN, .'AO', .X ) ;
/*  DECLARE THE VARIABLE X AS A CHAR    */
/*  DATA TYPE AND THE MAXIMUM RESERVED  */
/*  LENGTH IS NINE                      */
CALL CHAR ( .X, 10 ) ;
```

But, if we write them in reversed order, then that takes the form:

```
CALL CHAR( .X, 10 ) ;
RET = IO ( IN, .'AO', .X ) ;
```

In this case, 'proper' input strings whose length are less than ten are handled correctly. But, if the length of the input string is larger than nine, then '8OH' which indicates the longest length of the character variable is overlaid by the input string. Of course, errors will appear in manipulating the string operators and built-in functions.

(3) In the editor, the character 'S' is not allowed to enter into the text because it is already reserved as the current end mark of strings.

All the programs for the string package and editor are available from Dr. Moshell.

5.2 FURTHER PROJECTS

This thesis opens up the following projects:

(1) In the string package, the string comparison operation should be implemented .

(2) In the editor, there are several facilities which should be implemented .

1. Several commands can be typed in at the same time and executed in order.

2. To provide the 'F' command which is used to find a given string, then delete and replace it with another specified string.

3. To provide the 'J' command which is used to move the line and character pointer to the beginning of the file.

4. To provide the 'Z' command which is used to move the line and character pointer to the end of the file.

5. Control I which represents a TAB if there's not the TAB key should be provided.

LIST OF REFERENCES

LIST OF REFERENCES

1. Barron D. W.
"An Introduction to the Study of Programming Language"
Cambridge University Press, (1977)

2. Conway Richard and Gries David
"An Introduction to Programming"
Winthrop Publishers, Inc.
Cambridge, Massachusetts (1975)

3. Digital Equipment Corporation
"1978 CP/M User's Manual"
Pacific Grove, California 93950 (January 1978)

4. Elson Mark
"Concepts of Programming Languages"
Science Ressearch Associates, Inc., (1973)

5. Horowitz Ellis and Sahni Sartaj
"Fundamentals of Data Structures"
Computer Science Press, Inc.
Woodland Hills, California 91364 (1976)

6. Intel Corporation
"8008 and 8080 PL/M Programming Manual"
Santa Clara, California 95051 (1975)

7. Intel Corporation
"8080 PL/M Compiler Operator Manual"
Santa Clara, California 95051 (1975)

8. Intel Corporation
"8080 PL/M Compiler"
Santa Clara, California 95051 (1975)

9. Intel Corporation
"1977 PL/M User"s Manual"
Santa Clara, California 95051 (1975)

10. Kernighan Brian W.
"Software Tools"
Addison-Wesley Publishing Company, (1976)

11. McCracken D. D.
"A Guide to PL/M Programming for Microcomputer Applications"
Addison-Wesley Publishing Co., (1978)

12. Moshell J. M.
"Highlevel Lenguage for Microcomputers"
Class Notes
Knoxville, Tennessee 37916 (Summer 1979)

13. Nayak H. R.
"Telecommunication Processing Link between
DECsystem 10 and CPM Based Operating System"
Knoxville, Tennessee (1979)

14. Wong Sau-Fai Clement
"Input/Output and Floating-Point Arithmetic Package"
Knoxville, Tennessee 37916 (1980)

VITA

Shu-Wen Chiu was born in Taipei, Taiwan on July 12, 1954. She attended The Provincial Taipei First Girls High School in the same city, graduating in June, 1972. The following September she entered The National Taiwan University at Taipei, and in June , 1976 she received a Bachelor of Arts degree in Economics.

After graduation, she worked as an assistant financial analyst in the Sportsman Corporation. In September, 1977, she began studying for the Master of Science degree in Computer Science at the University of Tennessee, Knoxville, and taught Chinese in Asian Study Department. In August 1980, she received the Master of Science degree with a major in Computer Science.