DOCUMENT RESUME

ED 082 490 EM 011 459

AUTHOR Brownlee, Edward H., Jr.
TITLE PAMELA: An Interactive Assembler System for the IBM System/360 Computer.
INSTITUTION North Carolina Univ., Chapel Hill. Dept. of Computer Science.
PUB DATE 70
NOTE 82p.; Thesis submitted to the Department of Computer and Information Science of the University of North Carolina

EDRS PRICE MF-$0.65 HC-$3.29
DESCRIPTORS Computer Based Laboratories; *Computer Programs; Computers; *Computer Science Education; Digital Computers; Interaction; *Man Machine Systems; Masters Theses; Program Descriptions; *Programing; Technical Reports
IDENTIFIERS CC 30 Communications; CC 30 Terminal; CC 300 Screen; CC 303 Keyboard; IBM Model 40; IBM System 360; OS 360 Assembler Language; PAMELA; *Program Assembly Monitor Execution Learn Applicat; Station

ABSTRACT

A description of the Program Assembly and Monitored Execution for Learning Applications (PAMELA) is presented. It is intended for instructors who propose to use the system and for programers who wish to modify it. PAMELA is an interactive system designed to teach the operating principles of the IBM System/360 digital computer at the machine language level. It runs on an IBM 360 Model 40, communicating with the user via a remote CC-30 Communications Station. The system can assemble and execute programs written in a subset of OS/360 Assembler Language. The source code can be entered from the CC-30 terminal, or from punched cards previously input. It can be edited by insertion, deletion or alteration of statements and instructions can be executed continuously or singly. Contents can be displayed on the CC-300 screen or modified from the CC-303 keyboard. Used in the classroom, PAMELA can provide illustrations of the primary functions of instructions, as well as secondary effects such as setting of the condition code or raising of interrupts. Outside the classroom, the system can serve as a laboratory facility, with which students test and debug their own programs or carry out exercises planned by the instructor. (Author)

FILMED FROM BEST AVAILABLE COPY

ED 06[illegible]9[illegible]

University of North Carolina at Chapel Hill

Department of Computer and Information Science

EM 011 459

ERIC

ED 082490

PAMELA: AN INTERACTIVE ASSEMBLER SYSTEM FOR THE IBM SYSTEM/360 COMPUTER

by

Edward H. Brownlee, Jr.

A thesis submitted to the faculty of
the University of North Carolina at
Chapel Hill in partial fulfillment of
the requirements for the degree of
Master of Science in the Department
of Computer and Information Science

Chapel Hill

1970

Approved by:

Peter Calingaert

Adviser

ERIC

ACKNOWLEDGEMENTS

The author is indebted to Dr. Peter Calingaert for his invaluable assistance and suggestions in the preparation of this thesis; to Dr. Gerald A. Fisher, Jr., for providing a classroom test environment for the system; and to William H. Blair and Gary D. Schultz for their wise counsel on many matters of technology

CONTENTS

I. INTRODUCTION 1

II. USER'S MANUAL 3

III. SUGGESTIONS FOR USE 30

IV PROGRAM LOGIC 34

V. TESTING PERFORMED 45

VI. MODIFYING THE SYSTEM 52

APPENDIX A -- OS Interfaces and Feature Instructions . . . 63

APPENDIX B -- Differences Between PAMELA and OSA 65

APPENDIX C -- I/O Caveats 69

APPENDIX D -- System Messages 71

APPENDIX E -- Summary of System Commands 75

APPENDIX F -- Abbreviations Used 77

APPENDIX G -- Program Listings (Separate Cover)

LIST OF REFERENCES 79

I. INTRODUCTION

PAMELA (Program Assembly and Monitored Execution for Learning Applications) is an interactive system designed for use as a tool in teaching the principles of operation of the IBM System/360 digital computer at the machine-language level. It runs, itself, on an IBM 360 Model 40, communicating with the user via a remote CC-30 Communications Station, manufactured by Computer Communications, Inc.

The system is capable of assembling and executing programs written in a subset of OS/360 Assembler Language. The source code can be entered directly from the CC-30 terminal, or by means of punched cards previously submitted to the host computer and called up from the terminal. It can later be edited by insertion, deletion, or alteration of statements. Instructions can be executed continuously, halting only at preset locations or in case of error; or they can be executed one by one, halting after each instruction until a signal from the terminal is received. The contents of core storage, registers, and program status word can be displayed on the CC-300 TV screen or modified from the CC-303 keyboard; displayed information will be automatically updated whenever instruction execution is halted.

Used in the classroom, the system can provide dynamic illustrations of the primary functions of instructions, as well as secondary effects such as setting of the condition code or raising of interrupts. Outside the classroom, the system can be used as a laboratory facility, with which students test and debug their own programs, or carry out exercises planned by the instructor

The description of PAMELA presented in this paper is intended for the instructor who proposes to use the system or the programmer who wishes to expand or otherwise modify its capabilities. Some prior knowledge of OS/360 is assumed; Operating System terminology and abbreviations have been used freely. Appendix F contains an exhaustive list of these abbreviations.

II. USER'S MANUAL

In this chapter I have explained how to invoke PAMELA, establish communication with the system, and use all of the available facilities.

1 THE CC-30 TERMINAL

The instrument through which the user communicates with the PAMELA system is the CC-30 Communications Station The basic station configuration consists of a CC-301 TV Display Controller, CC-300 TV Display, and CC-303 Alphanumeric Keyboard. These are the only components necessary for using PAMELA; additional TV monitors and the CC-304 Light Pen are helpful for classroom use (see Chapter III), but are not required

The CC-301 Controller consists of a buffer memory, a character generator, and an input/output section. Data is read from or written to the memory via a telephone line, coupled to the devices by an IBM 2701 Data Adapter at the computer site and an acoustic coupler at the remote terminal. The data transfer rate achieved with this configuration is approximately 10 characters per second

The data in the buffer is scanned 60 times per second, converted to dot patterns by the character generator, and written on the TV display screen as humanly readable characters. Up to 800 characters (20 lines of 40 characters each) can be displayed in this way.

By means of the alphanumeric keyboard, which is similar to an ordinary typewriter keyboard but contains several additional control keys, data can be keyed directly into the buffer memory from the terminal, and subsequently transmitted to the computer. Two way communication between system and user is thus established

For a more detailed description of the CC-30, consult Ref. 1.

2 INVOKING THE SYSTEM

I will now describe the procedures to be followed in activating the PAMELA system and establishing communication between the system and the CC-30 terminal. These procedures are necessarily dependent upon the computing facility and operating system being used, and the particulars are applicable only to the configuration on which the system was written (see Chapter VI).

2.1 JOB CONTROL

PAMELA is invoked by means of job control statements in the OS job stream Refer to the sample JCL of Fig. 2.1.

The JOB statement initiates the job and specifies accounting information, and must be present for every job. The TYPRUN=HOLD parameter in this statement specifies that the job is not to be run until released by the computer operator. It is coded in order that the job may be read into the system well in advance of the proposed time of use, thus avoiding delay when the system is needed

The JOBLIB DD statement names the partitioned data set on which PAMELA's load module resides. It must be present for every job unless PAMELA has been placed in the OS link library.

```
//JOBNAME  JOB  (UNC.CS.S7123,2),'BROWNLEE E H',CLASS=G,TYPRUN=HOLD
//JOBLIB    DD  DSNAME=UNC.CS.S7123.BROWNLEE.PAMELA,DISP=SHR
//        EXEC  PGM=PAMELA10
//CC30      DD  UNIT=TWX
//SOURCE    DD  UNIT=2314,SPACE=(TRK,10),DCB=(DSORG=DA,RECFM=F)
//SYSUDUMP  DD  SYSOUT=A
//PROG1     DD  *
          . . . . . . . . . .
              . . data . .
                . . . . . . . .
/*
//PROG2     DD  *
          . . . . . . . . . .
              . . data . .  .
          . . .  .  . . . .
/*
```

Fig. 2.1 -- JCL for Invocation of PAMELA

The EXEC statement specifies the member name of the particular load module which is the PAMELA system, and causes the system to be loaded into core and to begin executing. It must be present for every job.

The CC30 DD statement defines the TWX line data set for the CC-30 terminal, and must be present for every job.

The SOURCE DD statement defines the disk data set on which the source file is to reside. It must be present for every job. In this example, ten tracks of space have been requested, sufficient for somewhat over 500 records.

The SYSUDUMP DD statement defines the data set on which a dump is to be written in the event of abnormal termination of the system. This statement is never mandatory, but is always advisable, since without it an abnormal termination may be very difficult to diagnose.

The remaining DD cards define card data sets which may be referenced from the CC-30 terminal at execution time. The cards may contain assembler language source statements or data to be read by the user's GET statements. Any number of card data sets may be included with a job; none is required. Any valid DD names (see Ref. 3) may be used; the ones shown in the example have no special significance.

When it is desired to initiate the system (assuming TYPRUN=HOLD was specified), have the operator release the job invoking the system. If system initialization is successful, the message "PAMELA WAITING FOR CALL" will be written at the console. The Data Adapter is then ready to receive a call.

2.2 ESTABLISHING CONNECTION WITH THE TERMINAL

For a complete discussion of the details of the wiring and operation of the CC-30 terminal, consult Ref. 1 The instructions given here are those which apply to the PAMELA system in particular.

Insure that the alphanumeric keyboard, acoustic coupler, and (optionally) light pen are plugged into the appropriate receptacles on the CC-301 controller. Both video and sync cables of the CC-300 TV Display should be connected to the corresponding "TV" receptacles on the controller. Video and sync cables for other monitors to be used (if any) should be connected to the corresponding "MON" receptacles. The sync selector switches on the back panel of all monitors to be used should be in the down (external sync) position. No CC-30 components other than those mentioned above need be connected.

Insure that the mode switches on the front of the controller are in the BLOCK, NORMAL, ALPHA, and READ positions (these are the down positions of all four switches).

Apply power to the controller, coupler, and TV monitors. Press the CLEAR and MASTER CLEAR keys on the alphanumeric keyboard.

Press the INT key to lock the keyboard; this conditions the controller to receive a transmission from the computer. Dial the number of the computer's data adapter, and when the carrier tone is heard, press the receiver firmly into the acoustic coupler.

The salutation "PAMELA" is written in the upper left corner of the screen The cursor is placed in the command area, the keyboard unlocks, and the system enters the command mode (see Sec. 4). Sign-on is now complete

3. CC-300 TV SCREEN LAYOUT

The user communicates with the system by means of commands entered through the CC-303 keyboard. The system replies to the user with messages displayed on the CC-300 display screen. It is important to distinguish between these system commands and messages and the assembler instructions and error messages, which appear in the source file (see Sec 7).

For purposes of the PAMELA system, the display screen should be imagined as partitioned into three areas (see Fig. 2.2). The top two lines on the screen are the message area. Here, and only here, the system displays error and other messages in reply to commands entered by the user. The bottom two lines on the screen are the input area. Here, and only here, the system reads commands, source statements, and updating information. The sixteen lines between the message area and

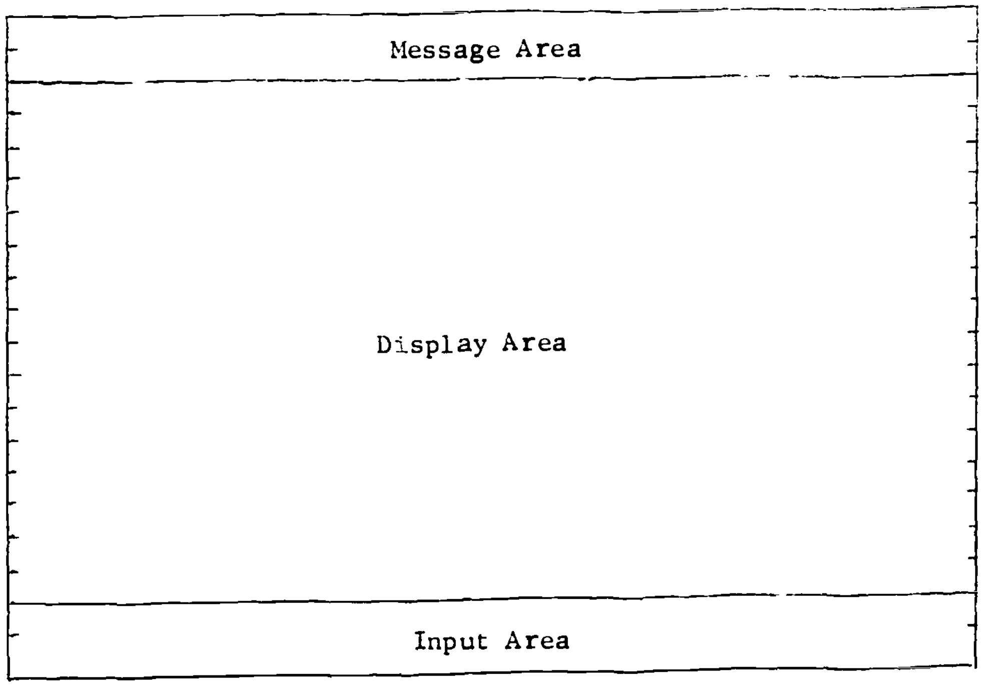

Fig. 2.2 -- Screen Layout

input area constitute the display area. Here, and only here, the system displays the contents of core storage, general purpose registers, floating point registers, and the program status word, as requested by the user. Much freedom is allowed the user in planning the format of the display area, in that each new display is placed on the line which he requests.

It should be emphasized that the input area is the only portion of the screen from which the system can read information entered from the CC-303 keyboard. It is possible, and at times may be convenient, to key information into other portions of the screen for notational purposes. Such information will, however, be completely ignored by the system.

4. SYSTEM MODES

Depending upon the type of operation being performed, the PAMELA system may be in either of two modes: command mode or repeat mode. Commands from the terminal will be recognized only when the system is in the command mode. In repeat mode, the command last recognized is executed repeatedly, once each time the INT key is pressed. New commands cannot be issued with the system in repeat mode. The repeat mode is entered only when one of the commands XEQ1, NPUT, or DERR is issued. To return from repeat mode to command mode, press the following keys in order: RESET, ←, TRANS.

5 COMMAND FORMAT

All commands in the PAMELA system must be entered in the lower right quarter of the input area, i.e., the last twenty spaces on the bottom line of the CC-300 screen. The cursor is automatically placed in the correct position for entering commands whenever the system is in command mode and is ready to accept a command. Should you disturb the position of the cursor, it can be replaced at the beginning of the command field by pressing RESET, ←—, TAB.

All commands have the following general format:

Operation	One or more blanks	Zero or more operands separated by commas	X-OFF

Fig. 2.3 -- Command Format

Note that no blanks are permitted between the beginning of the first and the end of the last operand, nor before the operation field. The 'X-OFF' character is formed by pressing the S key while holding down the SP-CODE key. It is not mandatory, but if it is omitted the entire command field following the end of the last operand must be blank. Use of the X-OFF character also shortens transmission time.

After entering a command in the correct field, press INT. The command will be read and, if no errors are detected, will be executed.

6. OPERAND TYPES

The operands required for a command depend upon the operation being performed, but will always be selected from four basic formats:

1) a decimal integer

2) a hexadecimal integer

3) a decimal number

4) a symbol

Each format is described in detail below

A decimal integer is an ordinary one- or two-digit whole number in the range 0-15 No sign, decimal point, nor leading zero is permitted. The decimal integer operand form is used only to refer to registers or to a quantity of lines in the display area.

Examples of valid decimal integers:

```
0

12
```

Examples of invalid decimal integers:

```
16      (not in range 0-15)

+4      (sign not permitted)
```

Hexadecimal integers consist of from one to six hexadecimal digits in the range 0-9 or A-F, enclosed in parentheses, and optionally followed immediately by the letter R. Hexadecimal integer operands are used only to refer to core storage locations. If the optional 'R' is coded, it specifies an address relative to location zero on the assembler listing most recently produced. Otherwise, an absolute machine address is specified If no assembly has been performed, an R is ignored

Examples of valid hexadecimal integers:

```
(02A800)

(11F1)

(30A)R
```

Examples of invalid hexadecimal integers:

```
(23H8)     (contains invalid hexadecimal digit)

(104) R    (space appears between parenthesis and R)
```

Decimal numbers consist of from one to three decimal digits in the range 0-9, optionally followed by a decimal point and from zero to three additional digits A decimal number operand may refer to an assembler language statement number, to a statement number increment, or to the quantity of statements referenced, depending upon the command in which it is used and its position therein.

Examples of valid decimal numbers:

24

019.8

169 444

Examples of invalid decimal numbers:

+16.1	(sign not permitted)
1.4800	(only 3 digits after decimal permitted)

Symbols consist of from one to eight alphabetic (A-Z,$,#,@) and numeric (0-9) characters, the first of which must be alphabetic. A symbol operand may refer to a label in an assembler language statement or to a card file DD name, depending upon the command in which it is used and its position therein.

Examples of valid symbols:

E2484

#7

PAMELA

Examples of invalid symbols:

8E5F	(does not begin with alphabetic)
C23456789	(more than 8 characters)
YO-YO	(contains character other than alphabetic or numeric)

7 SOURCE STATEMENT FORMAT

The PAMELA system includes the facilities for storing and updating a large number of assembler language ("source") statements in random access disk storage The source statements used in the system have basically the same format as those for the OS/360 assembler. Each is normally contained in the first 71 character positions of an 80-character source record (a record or card image is the disk storage equivalent of a single punched card). Column 72 is the continuation column, and should contain a blank unless a statement is to be continued on the next record. Columns 73-80 (used for a statement number under OS/360) must always be blank. If they are not blank when read in, the system input reader will make them blank.

In addition to the regular 80-character card image, the system input reader affixes a 40-character prefix to each record at the time it is placed on the disk file Before the program is assembled this prefix contains only the statement number written in its last seven positions. After assembly the prefix may contain, in addition, an error message or the location counter and object code for the statement. Whereas the 80-character card image portion of statements can be altered at will from the terminal, the 40-character prefix cannot be altered except by the system assembler.

8 INDIVIDUAL COMMAND DESCRIPTIONS

Armed with the foregoing general information, we are now prepared to discuss the individual commands available. There are 22 commands in all, and these are divided into six classes:

1) source handling commands
2) display storage commands
3) alter storage commands
4) assembler commands
5) execution commands
6) special purpose commands.

Refer to Appendix E for a tabular supplement to the discussion which follows

8 1 SOURCE HANDLING COMMANDS

The source handling commands are NPUT, DEL, DSRS, ASRS, DSRL, and ASRL They are used to read in, display, alter, or delete the assembler language statements making up a program.

The Input (NPUT) Command

This is the command used to read new source statements into the system Statements can be read either from the CC-30 terminal or from a card file previously submitted to the host computer.

SYNTAX The input command is written as follows:

```
        NPUT  decimal-number-or-symbol,decimal-number,symbol
```

The first operand is required; the remaining two are optional. If the second operand is omitted but the first and third are coded, two commas must separate the operands coded

SEMANTICS. The system immediately enters the repeat mode. If the third operand is present, the card file having the specified symbol as its DD name is opened and the entire file is read into the system memory. The system then returns to the command mode. If the third operand is omitted, the cursor will be placed at the beginning of the input area. Statements may then be keyed in from the CC-303 keyboard. Key in a statement and press INT. The statement will be read and the cursor positioned once again to enter the next statement. When all the desired statements have been entered, press RESET, ←— ,TRANS to return to command mode

Regardless of whether card or keyboard input is used, the first operand becomes the statement number of the first statement read, and the second operand is the increment added to this to obtain succeeding statement numbers. If the second operand is omitted, an increment of 1 is assumed. For numbering purposes, a statement continued on several records counts as a single statement.

Note that new statements are always collated into their correct sequential positions in the system memory. If the number of a new statement matches that of an old statement exactly, the new statement replaces the old. If the number falls between two old statements, the new statement is inserted. Thus, judiciously chosen operands can make the NPUT statement a powerful editing facility.

Examples:

```
NPUT 1
    (reads from the CC-30.  Statement numbers will be
     1,2,3, ...)
NPUT 3,0.1
    (reads from the CC-30.  Statement numbers will be
     3.0,3.1,3.2, ...)
```

```
NPUT 5,,PROG1
   (reads cards whose DD name is PROG1. Statement
    numbers will be 5,6,7,. . .)
```

The Delete (DEL) Command

This command is used to delete source statements from the system memory

SYNTAX The delete command is written as follows:

```
DEL  decimal-number-or-symbol,decimal-number
```

The second operand is optional; the first is required if the second is coded, but is optional otherwise.

SEMANTICS The system will delete from the memory the number of statements specified by the second operand. Any fractional part of this operand will be ignored. The first statement deleted will be the one whose number or label is the first operand. If the second operand is omitted but the first is coded, only the single statement named will be deleted. If both operands are omitted, the entire file will be deleted.

If a statement which is continued over several records is deleted, all records included in the statement are deleted. If an assembler LTORG or END statement is deleted, any literals inserted by the assembler following the statement will also be deleted.

Examples:

```
DEL  3,7
   (deletes seven statements beginning with the one
    numbered 3)
```

```
DEL 5 5
    (deletes statement 5.5)
DEL
    (deletes entire source file)
```

<u>The Display Source Commands</u> (DSRS and DSRL)

These commands are used to display source statements.

SYNTAX. The display source commands are written as follows:

```
     DSRS  decimal-number-or-symbol,decimal-number
  or DSRL  decimal-number-or-symbol,decimal-number
```

The first operand is required; the second is optional.

SEMANTICS. The number of statements specified by the second operand is displayed on the CC-300 TV screen, beginning with the statement whose number or label is the first operand. If the second operand is omitted, only one statement will be displayed. Immediately after the command is issued, the keyboard will unlock. Use the cursor control keys to position the cursor to the line in the display area where you wish the display to begin. Then press INT. If INT is pressed without placing the cursor in the display area, the system will select the lowermost blank line which is directly under a non-blank line (note that only main storage displays written <u>by the system</u> are considered to constitute non-blank lines). If no such line exists, the message "PLACE CURSOR" will be written in the message area, at which time the user <u>must</u> place the cursor in the display area and press INT.

If DSRL (Display SouRce Long) is coded, each statement displayed requires three 40-character lines of the display area, plus two additional lines for each continuation record; the prefix and all the text of each statement are displayed. All literals inserted by the assembler after a LTORG or END statement are displayed along with the statement they follow, and do not count towards the number of statements requested. If DSRS (Display SouRce Short) is coded, only the first 40 characters of each statement are displayed, except for the first record, whose prefix is displayed, also. Continuations and literals are not displayed. Thus each statement, except the first, requires only one line on the CC-300 screen.

Examples:

```
          DSRS  18.5,6
             (displays six statements beginning with statement
              number 18.5, in short form)
          DSRL  NAME1
             (displays the statement labeled NAME1 in long form)
```

<u>The Alter Source Commands</u> (ASRS and ASRL)

These commands are used to modify source statements.

SYNTAX. The alter source commands are written as follows:

```
      ASRS  decimal-number-or-symbol,decimal-number
   or ASRL  decimal-number-or-symbol,decimal-number
```

The first operand is required; the second is optional

SEMANTICS. The number of statements specified by the second operand will be displayed, one by one, in the input area, beginning with the statement whose number or label is the first operand. If the second operand is omitted, only one statement will be displayed. Immediately after each statement is displayed, transmission will halt and the keyboard will unlock. At this point use the cursor control and other keys to alter the displayed statement in any desired way. Then press INT. The revised statement will replace the old one in memory.

If ASRL (Alter SouRce Long) is coded, all the text of each statement is presented for alteration, with both lines of the input area being used. If a statement is continued on more than one record, the continuations are presented after the first record and do not count towards the number of statements requested. Literals generated by the assembler are not presented for alteration. If ASRS (Alter SouRce Short) is coded, only the first 40 characters of each statement are presented, with only the bottom line of the input area being used. Continuations are not displayed.

Note that it is not possible to alter the prefix of any record, since the prefixes are not presented for alteration by ASRS nor ASRL. Note also that unlike alteration of main storage (see Sec. 8.3), alteration of source statements does not result in an updating of information currently appearing in the display area.

Examples:

ASRS 10.67

(statement 10.67 is presented for alteration in the short form)

ASRL X,2

(two statements, beginning with the one labeled X, are presented for alteration in the long form)

8.2 DISPLAY STORAGE COMMANDS

The display storage commands are DCOR, DGPR, DFPR, DPSW, and CLR. They are used to examine the contents of core storage, registers, and the program status word in the user's program. All displays created using display storage commands are updated by the system whenever the corresponding storage contents change. After a display storage command is issued, the keyboard will unlock. Select a line for the display as described in Sec. 8.1

The Display Core (DCOR) Command

SYNTAX. The display core command is written as follows:

```
DCOR  hexadecimal-integer-or-symbol
```

The operand is mandatory.

SEMANTICS. The hexadecimal contents of 8 bytes of core storage, beginning at the location specified, are displayed on the CC-300 screen.

Examples:

```
DCOR  SAVEAREA
   (displays 8 bytes beginning with the storage
    location labeled SAVEAREA)
DCOR  (3E4)R
   (displays 8 bytes beginning with the byte whose
    assembler listing location is 3E4)
DCOR  (2A800)
   (displays 8 bytes beginning with machine location
    2A800)
```

The Display General Purpose Register (DGPR) Command

SYNTAX. The display general purpose register command is written as follows:

```
        DGPR  decimal-integer-or-symbol

     or DGPR  decimal-integer-or-symbol,decimal-integer-or-symbol
```

At least one operand is mandatory.

SEMANTICS. The 4-byte hexadecimal contents of the register or registers specified are displayed on the CC-300 screen.

Examples:

```
DGPR  14

   (displays register 14)

DGPR  4,ANYREG

   (displays register 4 and the one which has been
    equated to the symbol ANYREG)
```

The Display Floating Point Register (DFPR) Command

SYNTAX. The display floating point register command is written as follows:

```
        DFPR  decimal-integer-or-symbol
```

The operand is mandatory. Note that the only valid FPR specifications are 0, 2, 4, and 6.

SEMANTICS. The 8-byte hexadecimal contents of the floating point register specified are displayed on the CC-300 screen; the exponent and fraction parts are shown separated by a period (.).

Example:

```
DFPR 6
```

The Display Program Status Word (DPSW) Command

SYNTAX. The display program status word command is written as follows:

```
DPSW
```

SEMANTICS. The 8-byte program status word is displayed on the CC-300 screen. This display provides the following information: the current interruption code, instruction-length code, condition code, program mask, and instruction address.

The Clear (CLR) Command

SYNTAX. The clear command is written as follows:

```
CLR decimal-integer
```

The operand is optional

SEMANTICS. As pointed out earlier, any information displayed using display storage commands will be updated by the system whenever the corresponding storage contents change. Should you no longer require a particular display, the only way to remove it permanently from the screen (without the CLR command) would be to overwrite it with a different display

The clear command allows you to erase a display from the screen permanently When this command is issued with an operand, the keyboard unlocks just as for other display commands. Position the cursor on the line you wish to erase and press INT. The line is overwritten with blanks, and the system is notified not to update the line any longer. Additional lines are selected in the same way, one by one, until the number of displays specified by the operand have been erased.

If CLR is issued without operand, the entire screen (all three screen areas) will be cleared, and the system will immediately return to command mode

Note that since source statement displays are not updated by the system (they are "written and forgotten") you can erase such displays directly, simply by keying blanks over the displays from the CC-303 keyboard You can, of course, use the CLR command if you prefer.

8 3 THE ALTER STORAGE COMMANDS

The alter storage commands are ACOR, AGPR, AFPR, and APSW. They are used to alter the contents of core storage, registers, and the program status word.

SYNTAX The operand requirements of these instructions (i.e., the number, type, and interpretation thereof) are exactly the same as those of the corresponding display storage commands, DCOR, DGPR, DFPR, and DPSW

SEMANTICS. The current contents of the specified storage are written in the upper left quarter of the input area. They keyboard is then unlocked Use the cursor control and other keys to alter the hexadecimal display in any desired way. However, take care to:

1) enter only valid hexadecimal digits in the range 0-9 or A-F Otherwise the input will be rejected and the alteration will not take place.
2) avoid disturbing the format of the displayed storage contents; i.e , do not replace non-blanks by blanks nor blanks by non-blanks. Failure to observe this rule will result in unpredictable alterations to the storage.

3) avoid erasing the X-OFF character which is always written at the end of a storage display presented for alteration. Otherwise transmission time may be greatly increased.

After making the desired changes, press INT. The revised hexadecimal number will replace the old contents of the specified storage.

The system will honor all requests to display or alter storage <u>except</u>:

1) requests referencing core locations outside the area of the most recently assembled program; or
2) requests to alter any of the high-order 34 bits of the PSW. Thus only the condition code, program mask, and instruction address are subject to direct alteration.

8.4 ASSEMBLY COMMANDS

Once a program has been entered into the system memory, using the NPUT and other source handling commands, it must be assembled before it can be executed The PAMELA system includes a simple assembler for this purpose, capable of assembling a fairly generous subset of OS/360 assembler language (a summary of the restrictions defining this subset is given in Appendix B). The commands used to control assembly are ASM and DERR.

<u>The Assemble (ASM) Command</u>

SYNTAX. The assemble command is written as follows:

```
ASM
```

SEMANTICS. The assembler is invoked. A table of symbols is built which contains the label of each valid source statement assembled.

These symbols can subsequently be used as operands of system commands, when applicable. Core for the assembled program is allocated and the program is loaded into core. If any errors are detected, a message is placed in the prefix of each offending statement, and the message "ERRORS IN ASSEMBLY" is written in the message area of the CC-300 screen. If no error is detected in a statement, the location of the generated code and the hexadecimal code produced are placed in the prefix instead of an error message.

The Display Errors (DERR) Command

This command may be used to examine the error messages placed in the prefixes of source statements by the assembler.

SYNTAX. The DERR command is written as follows:

```
DERR
```

SEMANTICS The system immediately enters the repeat mode. The disk file is then scanned for a source statement which contains an error message in its prefix. When one is found, the prefix and the first 40 characters of its text are displayed in the message area of the CC-300 screen. The system then waits for the signal to scan for the next error (INT), or to return to command mode (RESET, ⟵ ,TRANS). When there are no more erroneous statements the message "END OF ASSEMBLY ERRORS" will be written in the message area.

You may return to the command mode after any error display (to alter the erroneous source statement, for example) and then re-issue DERR at a later time; the scan for errors will continue where it left off. However, once the entire source file has been scanned (i.e., the

END OF ASSEMBLY ERRORS message has appeared), DERR cannot be used to scan the file again until another assembly has been performed.

Note that assembler error messages can also be examined using the DSRL command. The DERR command merely offers the convenience of sorting out the statements which contain error messages.

8 5 EXECUTION COMMANDS

Once a program has been entered and assembled without error, it can be executed from the CC-30 terminal using the three execution commands, XEQ, XEQ1, and STOP.

The Execute (XEQ) Command

SYNTAX. The execute command is written as follows:

```
XEQ  hexadecimal-integer-or-symbol-or-decimal-number
```

The operand is optional.

SEMANTICS. Control is passed to the point in the program whose core location, label, or statement number is the operand of the command. If the symbol or decimal number form is used, the statement referenced must be a valid executable assembler language instruction. If the operand is omitted, control is passed to the instruction address specified by the current PSW.

Execution of instructions takes place continuously and without any visible indication at the terminal. Execution will halt only when one of the following conditions is satisfied:

1) a program interruption occurs;
2) a "STOP" is encountered (see STOP command, this section)
3) 1000 machine instructions have been executed;

4) an end-of-data condition is recognized on a user input file

When execution is halted for any of these reasons a message explaining the situation is written in the message area of the TV screen (the third condition above results in the message "PROGRAM APPARENTLY IN LOOP"). All storage displays on the screen are then updated.

Examples:

```
XEQ  (E6)R
```

(execution begins at the statement whose location on the assembler listing is 0000E6)

```
XEQ
```

(execution is begun at the instruction address specified by the current PSW)

<u>The Execute Single Instruction (XEQ1) Command</u>

SYNTAX. The execute single instruction command is written as follows:

```
XEQ1  hexadecimal-integer-or-symbol-or-decimal-number
```

The operand is optional.

SEMANTICS If an operand is present, the address specified immediately replaces the instruction address in the PSW; if not, the PSW remains unchanged The system then enters the repeat mode and waits for further signals from the terminal. Each time INT is pressed, the single instruction then indicated by the instruction address in the PSW is executed and all storage displays on the screen are updated.

Execution can continue in this manner indefinitely. If a "STOP" is encountered, a message to that effect is displayed but otherwise it is ignored. If a program interruption occurs a message is displayed and the operation is suppressed in the same manner as if continuous execution had been taking place (see Ref. 4). The system then stands ready to attempt to execute the next instruction. To cease executing, return to command mode in the usual way.

Example:

```
              XEQ1  COMPUTE
                 (the system makes ready to execute the instruction
                  whose label is COMPUTE)
```

The Stop (STOP) Command

SYNTAX. The stop command is written as follows:

```
         STOP  hexadecimal-integer-or-symbol-or-decimal-number
```

The operand is optional.

SEMANTICS. The location specified by the operand is flagged as a stop location. If the instruction address becomes equal to a stop location following an XEQ command, execution is immediately halted (i.e., before executing the instruction at which the stop was set. Exception: if a stop is set at the first instruction specified by an XEQ command, it is ignored). Up to four stops may be set by issuing several STOP commands with different operands, and these remain in effect until a STOP command with no operands is issued, at which time all are eliminated (it is not possible to eliminate stops selectively). If more stops are entered after the fourth, the excess stops are dropped, beginning with the oldest; the most recent ones remain in effect.

Example:

```
STOP 14
    (A STOP is set at statement number 14)
```

8 6 SPECIAL PURPOSE COMMANDS

The End (END) Command

SYNTAX. The End command is written as follows:

```
END
```

SEMANTICS. The END command should be the last one issued in a PAMELA session The message "PAMELA TERMINATING" is written in the message field of the CC-30C screen. The telephone line is then disconnected and the system terminates.

The Translate (XLAT) Command

SYNTAX. The translate command is written as follows:

```
XLAT  hexadecimal-integer-or-symbol
```

The operand is mandatory.

SEMANTICS. The EBCDIC character representations of 80 bytes of core, beginning at the location specified by the operand, are written into the message area of the CC-300 screen. Characters which have no EBCDIC encoding in the CC-30 character set appear on the screen as the "CAN" character (█). Displays created using the translate command are _not_ updated if the contents of core change.

Example:

```
XLAT  BUFFER
    (displays 80 bytes beginning with the storage
     location labeled BUFFER)
```

III. SUGGESTIONS FOR USE

In Chapter II I described in detail the facilities of the PAMELA system and how these may be invoked. In this section I will make suggestions for achieving optimal efficiency in the use of the system for several applications.

The system is intended for use in either of two modes: as an ersatz lab for testing, debugging, and experimenting with assembler-language programs (student-controlled mode), or as a classroom tool for illustrating the execution of System/360 machine instructions (teacher-controlled mode).

1. TEACHER-CONTROLLED MODE

As a classroom tool, PAMELA can be used to illustrate a variety of System/360 functions, such as

1) the alteration of the registers and main core by machine instructions;
2) setting of the PSW instruction address by branch instructions;
3) setting of the condition code bits in the PSW by arithmetic and logical instructions and by the SPM instruction;
4) setting of the instruction length code bits in the PSW by all instructions;
5) halting of execution and setting of the PSW interrupt code by program interruptions;

6) suppression of program interruptions by appropriate program mask settings;

7) action of elementary I/O macros.

When you plan to use PAMELA in the classroom, prepare the illustrations beforehand and run them on the system once before class, whenever possible Try to anticipate questions which are likely to be asked, and devise and test illustrations answering them. Following this procedure will insure that the examples are correctly constructed and that any system bugs uncovered by the examples will be revealed prior to the class.

Entering source statements from the CC-30 keyboard is an extremely slow process and should be avoided except when very brief alterations or insertions must be made. Instead, punch the source programs on cards and submit them to the computer center as explained in Chapter II. Note that any number of programs can be submitted in this way and accessed in a single session. By appropriate use of the NPUT command, code sequences can even be concatenated or merged. For example, if you submit two decks with DD names PROG1 and INSERT, each consisting of 20 statements, the command sequence

```
NPUT  1,,PROG1
NPUT  4.01,0.01,INSERT
```

will result in a source file containing 40 statements. Those of INSERT will be placed between the fourth and fifth statements of PROG1, and will be numbered 4.01, 4.02,..., 4.20. Any valid PAMELA commands could have been executed between the two NPUT commands, operating only upon the statements in PROG1.

On a low-speed line, such as the one on which the system was developed, considerable time is required for transmissions from computer to terminal: about four seconds per line, or more than a minute to fill the entire display area. Therefore, if lengthy displays (e.g., an assembler listing) are to be produced in class, the lecture should be planned so as to fill the periods of transmission. If very long source code listings are desired, or if many other displays are desired in conjunction with a source code listing, it is probably wise to present the source code on paper by some conventional means.

If the CC-30 terminal used includes a CC-304 Light Pen, this device can be used as a pointer to call attention to a particular area of the CC-300 screen. At any time while the keyboard is unlocked (the blue light is on), a character detected by the light pen will appear marked by a flickering background on all connected monitors. To replace the marker elsewhere, detect another character; to remove the marker entirely, press MASTER RESET on the CC-303 keyboard. (Note: do _not_ press MASTER RESET while the terminal is sending or receiving a transmission from the computer. To do so may result in lost data, a system "hang-up", or other catastrophe.)

Some students have complained that the marker renders the marked character difficult to read. It is advisable, therefore, to erase the marker promptly after it has been placed, or else to mark a field by an unimportant character (such as a leading zero, in the case of a numeric field)

2. STUDENT-CONTROLLED MODE

I envision two possible uses of the system in the student-controlled mode. One is to have each student or team of students carry out a planned exercise prepared by the instructor. The other possibility is to make the system available to the student as a test and debug facility for his own assembler language programs.

A planned exercise has the advantage that the student need not learn the details of system operation; he can be instructed which system commands to use at each step. This method also allows the instructor more control over what material the students learn during a session.

If, on the other hand, the student is given complete control of his session (unplanned exercise), he can tailor the study to his individual ability and interest. Of course, this alternative also makes less demand on the instructor's time.

If unplanned exercises are to be used, caution the student against the inefficiency of entering source statements from the terminal, mentioned earlier in this chapter. Discourage particularly his attempting to <u>compose</u> a program at the terminal. I have found that this procedure, while tempting in an interactive environment, is difficult and frustrating when coding any but the most trivial programs in assembler language.

IV. PROGRAM LOGIC

The PAMELA system consists of five basic components:

1) the Executive

2) the I/O Interface

3) the Assembler

4) the Display and Alteration Facility (DAF)

5) the Execution Monitor.

Here follows a general description of each of these components. Refer to the overlay diagram (Fig. 4.1) to see the relationship between the various modules.

1 THE EXECUTIVE

The Executive is a single, permanently resident module (EXEC) which initiates and oversees all system activities, and is the entry point of the PAMELA system. It is responsible for system initialization, interpretation of the "operation" portion of commands, and the display of most error messages. Also in this module are data fields which must remain resident in core, notably the symbol table and the System Control Block (SYSCB). Some part of the SYSCB is used by almost every component of the system, so this data field merits some special discussion

First come the user's sixteen pseudo-registers. Then his instruction address, absolute assembler listing origin, and his condition code and other pseudo-PSW information. The next two words, LBOUND and

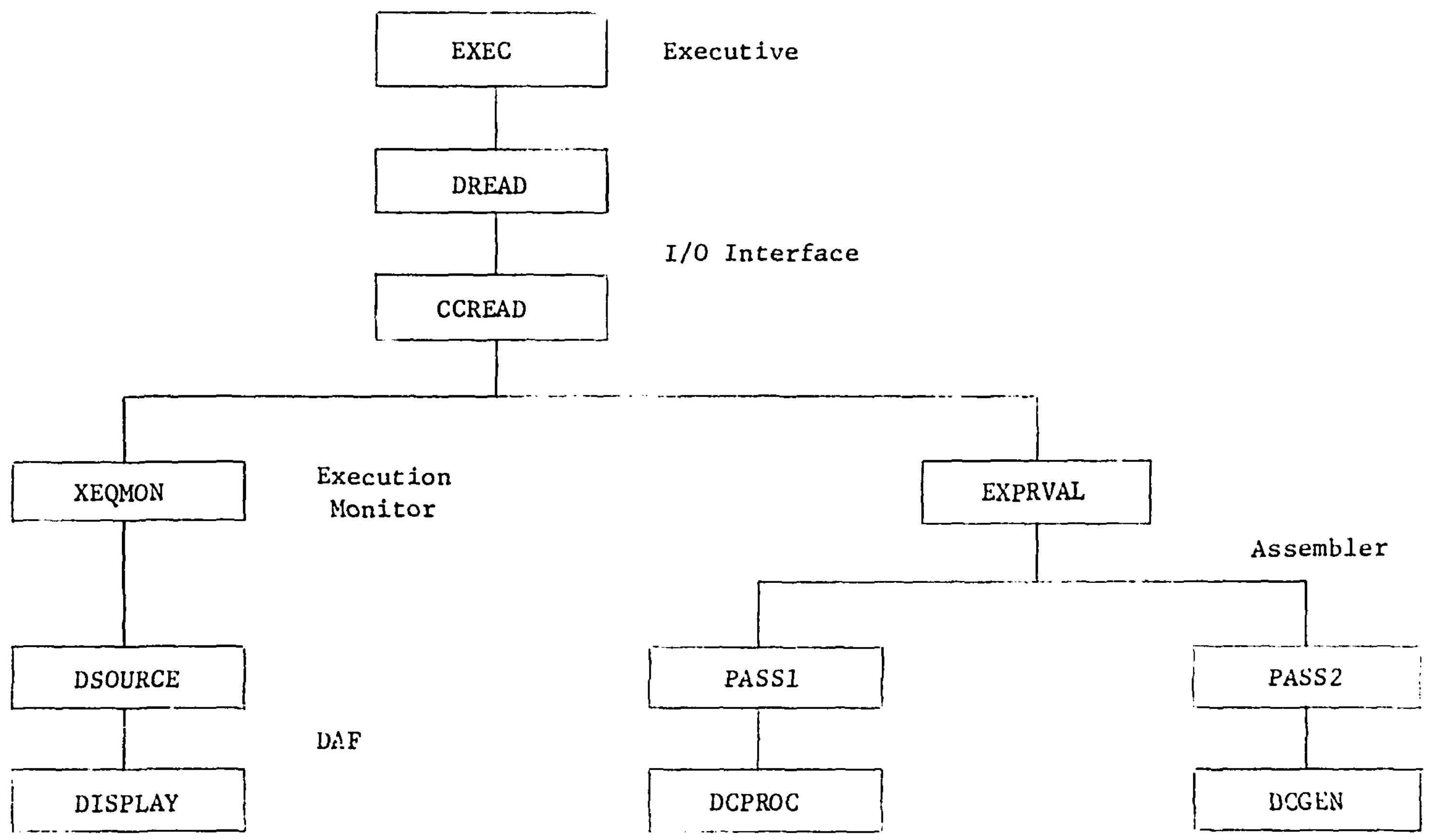

Fig. 4.1 -- Overlay Structure

UBOUND, define the limits of the user's core storage. He may not reference any location outside these limits. After these come the 88 bytes used for the display of error messages; almost all messages from peripheral routines are returned in this field. Then we have the three addresses used in symbol table maintenance. The first points to the beginning of the table, the third points to the end of the available space, and the second points to the last entry placed in the table by the most recent assembly, i.e., the last active entry. Next come the four "stop" fields, which are set by the STOP command Finally we have the screen table, where a three-word record is kept of each main storage display currently in the display area.

2. THE I/O INTERFACE

The I/O Interface consists of two permanently resident modules (DREAD and CCREAD) responsible for the details of all input and output for the system, with the exception of reading card files, which is done by the standard OS QSAM routines called in DSOURCE.

DREAD is a streamlined, specialized adaptation of IBM's Indexed Sequential Access Method (ISAM) used in maintaining the disk source file. It performs all buffer maintenance, key search, and collating of records, using a core-resident key table much like the "master index" of OS ISAM. The actual transfer of data to and from the disk is done by calling the OS BDAM and (for initial dummy records) BSAM routines.

The advantages of using this specialized routine instead of ISAM are:

1) It is not necessary to issue SETL and ESETL macros to reposition the file; repositioning is performed automatically when a specific record (as opposed to the next sequential record) is referenced.

2) OS ISAM requires the use of two different access methods (QISAM or BISAM) depending upon whether a record is to be inserted within the file or added to the end. DREAD allows records to be written anywhere in the file using the same DCB and read/write macros.

3) My initial experiments with ISAM had to be cut short due to excessive computer time consumption; DREAD performs all the services which I require at about ten times the speed.

These advantages are apparently realized because of the sacrifice of generality in DREAD.

CCREAD is the CC-30 terminal I/O interface. It is capable of reading from or writing to any specified locations on the CC-300 screen, and also of writing to the line where the user places the cursor and reporting where that line was. It also contains routines for enabling and disabling the TWX line, error recovery, and translation between EBCDIC and the backwards-ASCII-with-parity code of the CC-30.

With the exception of the initial connection, which is done via a BTAM "READ", all CC-30 I/O is done using the EXCP I/O technique; the requisite channel programs are coded explicitly in CCREAD. This affords two important advantages:

1) The Program-Controlled Interruption (PCI) facility provides elegant means for issuing the HALT-I/O instruction

required after a "Read Cursor Address Register" command to the CC-30. This is a very awkward task if BTAM is used exclusively.

2) The BTAM disconnect function aborts consistently when used with the CC-30. The simple "Disable" channel command which I issue has always completed successfully.

3. THE ASSEMBLER

The Assembler is a collection of five modules (PASS1, DCPROC, PASS2, DCGEN, and EXPRVAL) which perform the translation of a user's program from assembler language to machine code. It consists of two passes; the first overlays all the other nonresident modules, and the second in turn overlays the first.

PASS1 is the main routine for the first pass. It builds the symbol table, interprets all opcodes, allocates space for all constants, and constructs the literal table statements. Any core storage reserved for a previously assembled program is freed. Intermediate text, including some error messages, is generated and placed in the source file prefixes to be used by PASS2.

DCPROC is a subroutine used only by PASS1. It determines the length attribute, alignment, and total storage requirements for each constant encountered, whether as a literal or as the operand of a DC or DS statement. To a limited extent, it checks for formatting errors. Finally, in the case of address constants, it determines certain characteristics which must be known to allow assembly of the constant as a literal

PASS2, the main routine for the second pass, interprets the operands of all machine statements; performs base register functions, including the execution of USING and DROP statements; and outputs object code directly into core obtained by a GETMAIN macro. It completes the object code listing appearing in the prefix of each statement, or places error messages in the prefixes of erroneous statements not flagged in the first pass All records which were flagged by PASS1 are completely ignored by PASS2

The actual generation of constants is performed by DCGEN whenever PASS2 encounters a DC statement or a literal statement generated by PASS1 In generating S-constants, DCGEN utilizes the base-displacement facilities of the main routine, via the entry point BASEDISP Otherwise the module is self-contained.

EXPRVAL evaluates expressions of the type used as operands in assembler language statements. Because it is used in both passes, it must appear above PASS1 and PASS2 in the overlay structure, even though logically it is a subroutine of these modules.

3.1 INSTRUCTION SET

The PAMELA Assembler recognizes the mnemonic operation codes of all OS/360 machine instructions, including privileged operations, floating point instructions, and decimal instructions. One additional special-purpose instruction has been included in the set recognized by PAMELA: the BLUB (Branch and Link UnBridled) instruction.

The machine language format of the BLUB instruction is shown in Fig. 4.2. It performs exactly the same function as BALR, except that

OB	R_1	R_2

Fig. 4 2 -- BLUB Instruction

the Execution Monitor relinquishes control of processing, and the return address placed in R_1 will be a location in PAMELA rather than the address of the instruction following the BLUB. Most safeguards against program interruptions, infinite loops, destruction of PAMELA's resident modules, etc , are lifted. BLUB is intended for branching to system subroutines (as in GET and PUT macros) or for performing diagnostic checks of the system from the terminal Naturally, it should be used with discretion

3.2 MACRO-INSTRUCTIONS

Only five system macro instructions are recognized by the Assembler: OPEN, CLOSE, GET, PUT, and DCB. User macro definitions are not supported

All macros are expanded as if "PRINT NOGEN" were in effect, i.e., only the object code is generated and not the corresponding source statements Format requirements for the macros are quite rigid (see Appendix B).

In the case of the DCB macro, minimal error checking is performed, with all operands ignored completely except MACRF and DDNAME. Default values are assigned to all other operands, and no checks for agreement with the user's operands are performed

OPEN and CLOSE macros are expanded exactly as under OS, subject to the extra restrictions mentioned in Appendix B.

GET and PUT macros produce, under OS, a BALR to the system Get/Put routines. In order that PAMELA may relinquish control to these system subroutines, the PAMELA Assembler generates, instead, a BLUB instruction, described above. The return address passed to the subroutines is the location in the Execution Monitor from which PAMELA resumes control. Otherwise the expansions are identical to those of OS.

4 THE DISPLAY AND ALTERATION FACILITY

Two routines (DSOURCE and DISPLAY) are responsible for producing the displays and performing the alterations requested by the user. They (along with XEQMON) overlay the Assembler when called into core.

DISPLAY is the display and alteration manager for main storage: core, registers, the PSW, and stops. It is called when either 1) the user issues a command whose operand explicitly or implicitly refers to main storage; or 2) the Execution Monitor returns control to the Executive. In the latter case, DISPLAY serves to update any displays of storage changed by the user's code.

The module has three entry points. Entry point DISPLAY is used after a DCOR, DGPR, DFPR, CLR, or XLAT command, to indicate that no alteration to the user's main storage is to be made. Entry point ALTER is used following an ACOR, AGPR, AFPR, APSW, or STOP command, or after an XEQ or XEQ1 command with operand. For these entry points, an option code is passed in register 0 to indicate which command is in effect. Finally, CHECK is the entry point used upon return from the Execution Monitor; each main storage display is checked against the corresponding screen entry in the SYSCB to insure it is current. Those which are not are updated.

Most of DISPLAY consists of routines to interpret the four types of operands for the display and alteration commands, build the three-word screen entries showing what data is displayed on each CC-300 line, and generate the actual displays to be transmitted to the terminal. There are also brief sequences which make alterations to core, registers, the PSW, and the stops.

DSOURCE performs roughly the same functions for secondary storage on the disk file as does DISPLAY for main storage. It is called following a DSRS, DSRL, ASRS, ASRL, NPUT, or DEL command, and is responsible for displaying, altering, adding to, or deleting from the disk file (as in OS ISAM, deletion is accomplished by simply flagging the record with X'FF' in the first byte).

The bulk of the code in DSOURCE is concerned with interpreting the symbol or statement number operands which may appear with the source handling commands, constructing displays and transmitting them to the terminal, and building the prefixes which are affixed to input records. The DCB used for reading from a card file is in the DREAD module, so that it may be permanently resident and hence available for use by the Assembler as a skeleton in expanding the DCB macro.

5 THE EXECUTION MONITOR

The _raison d'être_ of the system, for which the remaining modules are but servants, is the Execution Monitor, which executes the user's code while protecting him against abnormal terminations due to program exceptions, infinite loops, or destruction of PAMELA's resident modules. The single control section, XEQMON (along with the DAF), overlays the Assembler when called into core.

XEQMON is entered following an XEQ or XEQ1 command, though the DISPLAY module may be called first to interpret the operand, if there is one The cycle of execution is then as follows:

1) Move a copy of the user's current instruction, as specified by the pseudo-PSW, into a special field of the Monitor. Update the pseudo-PSW. (Note: All the user's pseudo-storage is in the SYSCB.)
2) Examine the opcode and core addresses referenced, if any, and insure that the instruction does not reference any location outside the user's area of core; if it does, raise a pseudo-protection interrupt.
3) Insure that the instruction is not a BC, BCR, BAL, BALR, BCT, BCTR, BXLE, BXH, or EX, as these instructions could cause PAMELA to lose control. If it is one of these instructions, perform an interpretive pseudo-operation and go to step 7.
4) Insure that the instruction will not befoul PAMELA's "recovery" base register (see step 6). Either register 4 or 8 can be used; if the user instruction alters both register 4 and register 8 (only the LM instruction can do this) perform an interpretive pseudo-operation and go to step 7.
5) If the instruction is innocuous by all the above criteria, load all registers and the condition code from the user's pseudo-storage and allow execution of the copy of the user's instruction.
6) Recover from the demolition of PAMELA's registers wrought in step 5: replace the user's new registers and condition code in

his pseudo-storage, and re-establish the Execution Monitor's base registers.

7) Count one user instruction as having been executed. If the count passed to XEQMON in register 0 has been reached (this count will be 1 if XEQ1 was specified), or if a stop has been set at the current instruction address, return to the Executive. Otherwise, go to step 1.

Except for those errors checked for in steps 2, 3, and 4, any kind of program interrupt may occur when the user's instruction is executed in step 5. If an interrupt does occur, OS passes control to PAMELA's interruption handler, EXERR (an entry point of XEQMON), as specified by the SPIE macro issued by the Executive during system initialization. EXERR determines the type of interrupt which occurred, sets the interrupt code in the pseudo-PSW, and returns control to the Executive with an error message indicating the interrupt type.

The only known way that user code can terminate the system abnormally is for an error to occur after the Execution Monitor has relinquished control, i.e., after an SVC or BLUB instruction.

V. TESTING PERFORMED

All system testing discussed in this chapter has been performed on line, in the hardware environment described in Chapter VI. Though the tests were not all performed during the same terminal session, PAMELA was in each session run in the 44K "Graphics" partition, other tasks being concurrently active in the multiprogramming environment.

I have listed in separate paragraphs the tests aimed at the various components of the system. In fact, of course, all tests were performed on the system as a whole, and a particular test in general required action by several components.

1. EXECUTIVE

Each of the 23 commands in PAMELA's repertoire has been issued and correctly interpreted. Invalid commands, including some containing too many characters and some not properly positioned in the field, have resulted in the appropriate error message. The error display facilities have correctly displayed 20, 40, and 60 character error messages returned from peripheral routines. The assembly error scanning routine has performed correctly with both erroneous and error-free assemblies; the scan has been successfully halted and re-started.

2. DISPLAY AND ALTERATION FACILITY

Each of the 6 source handling commands has been successfully executed, with all valid combinations of omitted and specified operands tested for each command. Operands tested have included examples of both the statement number and symbol formats. Extensive source input has been accomplished both from card decks and from the terminal

Omission of operands has produced the appropriate error message for those commands requiring operands (extra operands are ignored by DSOURCE) Nonexistent labels, statement numbers, and DD names have been entered as operands and have produced the appropriate messages. Several arbitrary invalid operands were tried in both operand formats and produced the appropriate message.

An NPUT command causing statement number overflow has been tried and correctly flagged. An input source deck was read which contained more than 16 consecutive continuation cards; input was halted and the appropriate message generated

Each of the 11 main storage display and alteration commands has been successfully executed with all valid combinations of specified and omitted operands tested for each command. All four operand formats have been correctly interpreted.

All four types of main storage display have been correctly created, and have been updated following both alter-storage and execution commands The XLAT command has correctly translated codes belonging to the CC-30 chracter set, and has produced the "CAN" character in place of other codes, as intended. More than four STOP commands were issued without an intervening null STOP command, and the excess stops were dropped beginning with the oldest, as intended.

Omission of operands has produced the appropriate error message for those commands requiring operands (extra operands are ignored by

DISPLAY). Nonexistent labels and statement numbers have been entered as operands and have produced the appropriate error messages. Several invalid operands were tried in all four operand formats and as input to main storage alteration commands, and produced the appropriate message. Attempts were made to reference core locations outside the program area and to modify the 34 high-order bits of the PSW; in each case the target was properly protected.

3. ASSEMBLER

Each of the 160 System/360 instructions bearing mnemonic operation codes (the Diagnose instruction does not) has been submitted with appropriate operands to PAMELA's assembler, and has been assembled correctly, in the sense that the object code was the same as that produced by the OS assembler. This includes the extended mnemonics for the BC and BCR instructions. Operands were specified with both explicit and implied lengths, explicit and implied base registers, and expressions of maximum allowable complexity, as set forth in Appendix B

Each of the assembler instructions honored by PAMELA has been issued and correctly executed. In the case of CNOP, operands have been included requiring the generation of one, two, and three NOPR instructions. In the case of the DC and DS instructions, each of the 12 constant types recognized by PAMELA has been correctly generated, with both explicit and implied lengths, and with multiple suboperands where applicable. Several different replication factors, including zero, were correctly applied.

Each of the 5 I/O macros recognized by PAMELA has been issued and correctly expanded, in the sense that the generated code has successfully performed input and output functions, including recognition of and end-of-data condition.

The following assembler language errors were submitted to the assembler and correctly identified:

- Symbols beginning with a numeric character, or consisting of more than eight characters, or containing an illegal character;
- An unlabeled EQU instruction;
- Labeled CNOP, END, and ORG instructions;
- Literals longer than 61 characters;
- More than one consecutive continuation card, or a continuation card with non-blanks in columns 1-15;
- Undefined or missing opcodes;
- Statements following the END card;
- Statements which reserve core storage preceding a START statement, or more than one START statement in a single program;
- Missing and extra operands with both machine instructions and assembler instructions;
- Duplicate or undefined labels;
- Constant definitions (both in DC statements and as literals) specifying invalid types, invalid lengths, excessive magnitudes, invalid data, and extra or omitted suboperands;
- Constant definitions with missing parentheses or quote marks;
- DCB instructions with invalid or missing MACRF or DDNAME specifications;
- An absolute expression as the operand of an ORG instruction;
- Relocatable expressions as the operands of START and DROP;

- Expressions containing consecutive operators, invalid self-defining terms, undefined operators, and operators as the final characters;
- Expressions containing two levels of parentheses;
- Expressions containing a literal in conjunction with other operands;
- Expressions consisting of the sum of two relocatable terms, or containing a * or / operation applied to a relocatable term;
- Expressions having a value greater than $2^{24} - 1$;
- Relocatable expressions used as register, length, displacement, or immediate data specifications;
- Register, length, displacement, or immediate data specifications outside permissible range;
- A DROP instruction referencing a register not in use;
- An unaddressable expression used as the second operand of an RX instruction;
- Literals appearing twice in a single instruction, or as the destination.field of an instruction which alters core storage;
- Alignment errors on double-word, full-word, and half-word instructions;
- OPEN and CLOSE instructions with invalid option specifications;
- Programs causing overflow of the symbol table, literal table, or core storage pool.

4 I/O INTERFACE

Each of the functions of CCREAD has been performed successfully: connecting and disconnecting of the TWX line, reading from a specified area of the CC-30 memory, waiting for an interrupt from the terminal, writing to a program-specified screen location, and writing to a location designated from the terminal. In the last case, both manual and default placement of the cursor have been accomplished, with the appropriate message being generated when default placement is requested but is not applicable.

CC-30 I/O errors, presumably due to normal line noise, have occurred in the course of testing, and in every case recovery has been successfully accomplished.

Each of the functions of DREAD has been performed successfully: creation and expansion of the source file; reading of records, both sequentially and randomly; and writing of records at the end of the file (concatenating), between two existing records (inserting), and in place of existing records (updating). Deleted records (bearing the X'FF' flag in the first byte) have been ignored, as intended.

The end-of-data condition has always been correctly raised by returning a code of 4 in register 15.

By varying the SPACE parameter of the SOURCE DD card, I have on different occasions caused both the physical disk space and the index space allocation to be exceeded. In each case the system terminated abnormally with a completion code of 12, as intended.

5. EXECUTION MONITOR

Each of the 11 machine instructions which are performed interpretively by the execution monitor, and at least one instruction having each of the five basic instruction formats (RR, RX, RS, SI, and SS), has been successfully executed. Successful execution means that all main storage (including the fields of the PSW) was correctly set following execution of the instructions.

Execution has been halted and the appropriate message displayed by: each of the 15 types of program interruptions, encountering a "step" set 'y the user, exhaustion of the instruction count passed in register 0, and an end-of-data condition on a user input file

VI. MODIFYING THE SYSTEM

The comments of this chapter are intended to serve as guidelines in the implementation of future modifications or extensions to the system. Known quantitative limitations on various system features are also described.

1 DIAGNOSTIC FACILITIES

To aid in the testing of modifications made to the system (and in the elimination of bugs which may turn up in the present system), two diagnostic facilities have been incorporated into PAMELA. They are the BLUB instruction and the BOMB command.

The BLUB instruction, described in Chapter IV, Sec. 3.1, was originally intended as a device to allow the execution monitor to relinquish control to Operating System subroutines, such as GET and PUT. It was later discovered that through judicious use of the instruction, certain protective restrictions imposed by PAMELA could be permanently removed, thus allowing the powerful storage display and alteration facilities to be used on the system itself. For example, one could compose a program from the terminal which would reset the upper and lower program area boundaries in the SYSCB to X'FFFFFF' and X'000000', respectively, and branch to this routine via a BLUB. After the routine executed and returned control to PAMELA, the DCOR and ACOR commands could be used to examine core contents anywhere in the partition: the save area for PAMELA's registers, for instance. This technique was in fact used on a number of occasions in debugging the system.

The BOMB command is a special system command which is issued just as any other command, but was not mentioned in Chapter II because it is not intended for general use. It takes no operand. When issued, it causes the message "PAMELA TERMINATING" to be written in the message area, after which the system terminates abnormally with a completion code of 16. If a SYSUDUMP DD card was included with the job, a dump will be produced, which can then be used for debugging.

2. THE SOURCE FILE

The number of records which can be stored on the disk source file is limited by two factors: the volume of disk storage available, and the amount of core allocated for the index.

The 120-byte source records are blocked into units of 14 records each. Each track allocated to the data set, assuming an IBM 2314 disk drive, will hold four such blocks; thus one cylinder is sufficient for 1120 source records. The SPACE parameter of the //SOURCE DD card for a given run should request at least N/56 tracks, where N is the largest number of records which will occupy the main source file at any one time during the run. Considerable margin (perhaps 20%) above this minimal figure should be allowed if the file may be modified from the terminal, as the space occupied by records deleted during use may under certain conditions remain temporarily unavailable.

Each block of records requires an 8-byte entry in the INDEX, an area of main core located in DREAD. In the present system, space for

50 index entries has been provided; this is sufficient to maintain 700 source records. To change this capacity, simply make the replication factor of the INDEX field equal to the number of blocks desired.

If either the disk space or index space requirements exceed that which is available during a run, PAMELA terminates abnormally with a completion code of 12.

3. THE SYMBOL TABLE

The symbol table, created by the Assembler and occupying an area of core in EXEC, contains a 16-byte entry for each symbol or literal appearing in an assembly. In the present system, space for 50 such entries is provided in the field labeled SYMTBL. Overflow of this table results in immediate termination of the current assembly: the system does not ABEND. To change the capacity of the symbol table, increase or decrease the size of the SYMTBL field by four full words for each entry space to be added or deleted.

A desirable sophistication would be to have core for the symbol table allocated dynamically at assembly time. However, I can think of no way to accomplish this which does not involve cumbersome (i.e., chained) table search techniques, additional auxiliary storage, or a third assembler pass. A much simpler alternative would be to add a special system command which would allow the user to specify the size of the symbol table from the terminal if the default size were unsuitable. A GETMAIN would be issued and the appropriate addresses stored in the SYSCB.

4. LITERALS

Another assembler limitation is upon the total length, in characters, of literal constant definitions occurring between successive LTORG statements, or between the last LTORG and the END statement. Literal definitions are stored in the LITTBL field in PASS1 until a LTORG or END statement causes them to be dumped into the source file, at which time LITTBL is re-initialized. Literals stored in the literal table require one byte for each character in the definition, excluding the initial '='. Thus, for example, the literals

```
=E'10 5'
=5PL4'6'
=C'&&ABCDEF'
```

would require a total of 7+7+11=25 bytes in the literal table. In the present system, 256 bytes are provided. If the table overflows, the current assembly is terminated immediately, but the system does not ABEND. To change the capacity of the literal table, alter the length modifier of the LITTBL DS statement.

In addition to the above limitation on the combined length of literals, the length of a single literal definition, excluding the initial '=', may not exceed 61 characters. If this restriction is violated, the statement containing the literal is flagged, but assembly continues. The purpose of the restriction is to insure that the assembler-generated literal statements following a LTORG or END instruction will each fit on only one record, in columns 10-71. To remove the restriction, add a sequence to the LTORG statement handler in PASS1 allowing it to generate continuation cards, and remove the test for literal length performed in the literal processing (LTPROC) section.

5. HARDWARE AND SUPERVISOR SERVICES

PAMELA was designed to run on an IBM 360/40 computer, under the control of OS/360 with MFT-II, Release 16. The system instruction set should include both the decimal and floating point feature instructions. The protection feature is not necessary, as it is partially simulated by PAMELA. Approximately 23K of core is required for the system, including all overhead but excluding space for the user's object program and buffers.

The I/O facilities used are:

1) an IBM 2314 Direct Access Storage Facility;
2) an IBM 2540 Card Reader/Punch;
3) an IBM 2701 Telegraph Adapter, Type II (TWX);
4) a Computer Communications CC-30 Communications Station, basic configuration, specially modified for TWX compatibility.

I have not attempted to run PAMELA with any other computer, operating system, or I/O configuration, so I cannot state with authority exactly what changes would have to be made in order to do so. However, I will now suggest trouble spots which are likely to arise when attempting to implement PAMELA in a different environment, based upon my knowledge of the environment-sensitive aspects of the system.

5.1 SYSTEMS WITHOUT FLOATING POINT OR DECIMAL FEATURE

The floating point feature instructions are required only in the execution of system commands referencing the floating point registers, (DFPR and AFPR) and in the generation of floating point constants during assemblies. In the absence of the floating point feature these two services are meaningless anyhow, and the code performing them should simply be removed. Appendix A includes a tabulation showing the location of each occurrence of floating point instructions.

Lack of the decimal feature is a more serious problem. Decimal instructions are used in such diverse activities as numbering source statements, interpreting constant definitions in assemblies, and formatting displays (see Appendix A). Decimal arithmetic functions will have to be replaced by binary arithmetic followed by conversions. Formatting will have to be made less sophisticated than that afforded by the ED instruction (e.g., suppression of leading zeros), or else code mimicking the functions of ED will have to be written.

5.2 SYSTEMS WITH INSUFFICIENT CORE STORAGE

By current standards the core storage requirement of PAMELA is quite modest, and will not present a problem on most IBM/360 models unless very large user programs are to be run. Indeed, the system was written with compactness in mind, and if core storage should become a problem I can offer but few suggestions for reducing the requirements significantly. Such suggestions as I do make will be concerned with reducing the size of tables and buffers, rather than executed code.

The bulkiest parts of the system other than the executed code are, in order of decreasing size:

1) the two disk I/O buffers in DREAD (3360 bytes combined);

2) the error messages in PASS1 and PASS2 (1276 bytes combined);

3) the symbol table in EXEC (800 bytes).

Two buffers are required in DREAD in order to avoid an unreasonable amount of reading and writing during insert operations. Each is 1680 bytes long, accommodating one block of 14 records each 120 bytes long. The size of the buffers could be reduced by decreasing the blocking factor of the source file. This can be achieved simply by reducing the value of the BLKF symbol as defined in an EQU statement in DREAD and re-assembling the module. The buffers lengths and all other blocksize-dependent values, such as the BLKSIZE operand of the DCB's, are expressed in terms of BLKF and will be adjusted automatically. Naturally, such a reduction would result in slower I/O operation and decreased space utilization efficiency on the disk.

The size of the error message tables in the assembler suggests placing all error messages in a third assembler pass and identifying errors in the first two passes by means of a single number, say. The third pass could overlay the first two, thus requiring no additional core, and could substitute an actual message for the corresponding numbers. Note that since PASS1 and PASS2 overlay each other already, this procedure will not result in a savings of a full 1276 bytes; in fact the amount saved will be equal to the length of the larger of the tables in PASS1 and PASS2 (roughly half of 1276).

I have explained earlier how the size of the symbol table can be reduced by decreasing the possible number of entries. It would be desirable to reduce the size of each entry below the present 16 bytes; but I don't believe this is possible without eliminating some of the services (e.g., recognition of statements by label) which use the table.

5.3 OTHER OPERATING SYSTEMS

Running PAMELA under operating systems other than OS/360 Release 16 may require sweeping changes in I/O segments, but otherwise should present few problems. Moreover, almost all I/O functions have been carefully relegated to those modules whose sole responsibility is the handling of I/O details, i.e., DREAD and CCREAD. The only exceptions are the macros in DSOURCE which are used in reading source code from cards. Thus, only these three modules need be modified in order to run PAMELA under different I/O supervision. On the other hand, if the user is to be permitted to perform I/O functions in his object programs, it may be necessary to modify the macro-expanding portions of PAMELA's Assembler, or else require the user to expand his own macros as ordinary machine instructions.

There is one particular modification that may well have to be made even when running under OS Release 16, at installations other than the one on which PAMELA was developed. The convention here is to "spool" card files onto temporary disk storage at the time a job is entered into the system. Such files are blocked 5 cards (400 bytes) to a block. Since this is a purely local convention, it will quite likely be necessary to change the BLKSIZE parameter of the card DCB (CRDCB) in DREAD for operating at any other installation

Aside from I/O, the only interfaces with the Operating System are through the Overlay Supervisor, and the following macros: GETMAIN (Get Main Storage), FREEMAIN (Free Main Storage), ABEND (Abnormal End), and SPIE (Specify Program Interruption Exit).

I will make no attempt to discuss in greater detail the substitutions which will be necessary to run PAMELA under particular operating systems. To aid in such an endeavor, however, Appendix A includes an exhaustive tabulation of all PAMELA/OS interfaces (system macros) and their locations in the source listings.

5.4 OTHER I/O HARDWARE

With the exception of the channel programs for the 2701/CC-30 in CCREAD, all I/O in PAMELA is done by standard OS access methods. Thus it should be device-independent in an OS environment and I foresee no difficulties in using PAMELA with other card readers, such as the IBM 2501, or other disk drives, such as the IBM 2311. Note, however, that if a 2311 disk drive is used, optimal space utilization may require some experimenting with the source file blocking factor.

Operating with a terminal other than the CC-30 or through a data adapter other than the IBM 2701 will probably require sweeping changes in the CCREAD module, because the device-dependent EXCP access method is used. In addition, a few CC-30 control character manipulations take place in modules other than CCREAD; all such occurrences are listed in Appendix A.

Note also that a specially modified CC-30 terminal is required: the X-OFF character must be added to the controller's character set in order to make the terminal TWX-compatible This modification will be made by the manufacturer if requested.

5. MULTIPLE TERMINALS

The PAMELA system has not been written for use with more than one terminal connected simultaneously. Moreover, I have done little investigating of the possibility. I will only state that all the system modules, with the exception of the root segment (EXEC, DREAD, and CCREAD), are serially reusable but not re-entrant. EXEC is not serially reusable because it contains the SYSCB and symbol table. DREAD contains the disk buffers, index, and a few pointers which must remain unaltered from one invocation to the next, as well as several DCB's. CCREAD contains the DCB for the CC-30.

6. A SAVE FACILITY

It would be desirable to allow a user to save a disk source file for later use, once he has debugged a program or example. He can, in fact, do this now by specifying a permanent data set in the //SOURCE DD statement and signing off the system when his source file is satisfactory. However, he cannot in the same session cause a file to be saved and then continue with another problem.

One way to achieve this latter capability would be to implement a new system command which, when issued, would close the current source file and open a new one, defined by a different DD statement, whose name could be specified by the operand of the command. The new DDNAME would replace the old one in the source file DCB's, as is done for card files now. The number of different files which could be saved would then be limited only by the number of DD statements defining source data sets for the job.

A (less cumbersome) variation on this would be to provide a command which causes the current source file to be printed or punched. After the command is executed, the file could be cleared and re-used in the usual manner.

APPENDIX A

OS Interfaces and Special Feature Instructions

In this appendix I have tabulated the occurrences of PAMELA/OS interfaces and special feature instructions, as an aid to implementing PAMELA on another computer or operating system. Entries in this list have the following format: XXXX nn (YYYY), where XXXX is the module name, nn the page number in the assembler listing of the module (see Appendix G), and YYYY the particular instruction being cited, if any.

Direct Modifications of System Control Blocks:

CCREAD 7	CCREAD 9	DSOURCE 8
PASS2 16	PASS2 17	

CC-30 Control Character Manipulations (other than in CCREAD):

DISPLAY 6	DISPLAY 9

Occurrences of Decimal Feature Instructions:

DISPLAY 5 (ED)	DISPLAY 5 (ED)	DCPROC 1 (ZAP)
DCGEN 2 (ZAP)	DSOURCE 3 (ZAP)	DSOURCE 8 (AP)
DSOURCE 8 (ED)		

Occurrences of Floating Point Feature Instructions:

DISPLAY 9 (LD)	DISPLAY 12 (STD)	DCGEN 10 (LD)
DCGEN 10 (LD)	DCGEN 10 (MDR)	DCGEN 10 (MDR)

DCGEN 10 (SDR)	DCGEN 11 (LD)	DCGEN 11 (MDR)
DCGEN 11 (ADR)	DCGEN 11 (DD)	DCGEN 11 (LNDR)
DCGEN 11 (STD)		

Occurrences of System Macros:

CCREAD 6 (CLOSE)	CCREAD 7 (EXCP)	CCREAD 7 (WAITR)
CCREAD 7 (ABEND)	CCREAD 9 (OPEN)	CCREAD 9 (READ)
CCREAD 9 (ABEND)	CCREAD 9 (WTO)	CCREAD 10 (WAITR)
CCREAD 14 (DCB)	CCREAD 14 (DFTRMLST)	PASS2 2 (GETMAIN)
PASS2 2 (SPIE)	PASS2 22 (SPIE)	DREAD 4 (CLOSE)
DREAD 4 (OPEN)	DREAD 4 (WRITE)	DREAD 5 (CHECK)
DREAD 5 (CLOSE)	DREAD 5 (OPEN)	DREAD 7 (READ)
DREAD 7 (READ)	DREAD 7 (WRITE)	DREAD 7 (WRITE)
DREAD 7 (CHECK)	DREAD 8 (READ)	DREAD 8 (READ)
DREAD 9 (ABEND)	DREAD 10 (DCB)	DREAD 11 (DCB)
DREAD 12 (DCB)	EXEC 1 (SPIE)	EXEC 5 (ABEND)
DSOURCE 8 (OPEN)	DSOURCE 8 (GET)	DSOURCE 10 (CLOSE)
DSOURCE 12 (CLOSE)	PASS1 2 (FREEMAIN)	

APPENDIX B

Differences Between PAMELA and OS Assembler Languages

The PAMELA Assembler is faster and requires much less core and secondary storage than the OS Assembler for a given assembler language program. These advantages are realized at the cost of some generality. Here I have listed all the known differences between PAMELA assembler language and OS assembler language (F-level) (see Ref. 2). Although the list appears lengthy, I claim that most of the omitted features are not likely to be needed in a program written by a beginning student of OSA, nor in an illustrative program for such a student.

Only five system macro instructions are recognized by the assembler: OPEN, CLOSE, GET, PUT, and DCB. User-written macro definitions are not supported. For the five system macros, the operand formats are more restricted than under OSA, as noted below.

All macro-instructions must be written in the "normal" format (adopting the terminology of Ref. 2). The "alternate" format, wherein blanks and/or comments may follow any operand of a macro-instruction which is to be continued on a subsequent card, is not permitted.

The OPEN and CLOSE macros must be written as follows:

```
      OPEN  (expression,(INPUT))
   or OPEN  (expression,(OUTPUT))
   or OPEN  (expression)            (INPUT is assumed)
```

```
        CLOSE  (expression)
```

In particular, note that all parentheses are required, that the register form of specification (as CLOSE ((2))) is not permitted, that only the INPUT or OUTPUT option is permitted in OPEN, and that only one DCB may be opened or closed by each macro-instruction.

The GET and PUT macros must be written as follows:

```
        GET  expression,expression
        PUT  expression,expression
```

In particular, note that the register form of specification is not permitted, and that both expressions are required (i.e., only the "move" form of the macros may be used).

In the DCB macro, all operands are completely ignored except MACRF and DDNAME. Both of these operands are required, but may appear anywhere in the operand field. DDNAME is written just as in OSA. MACRF must be coded as follows:

```
        MACRF=(GM)       (for an input data set)
     or MACRF=(PM)       (for an output data set)
```

Except for these two operands, default values are assumed for all DCB fields, and the user's specifications, if any, are not checked for agreement. The default values are as follows:

```
        On Input:    DSORG=PS,RECFM=FB,BLKSIZE=400,LRECL=80
        On Output:   DSORG=PS,RECFM=FA,BLKSIZE=133
```

An end-of-data condition on an input file always results in halted execution and a message on the CC-30 screen. The user must then direct further processing through the use of system commands.

Since all programs assembled by the PAMELA assembler are self-contained and are loaded directly into core, the following OS external reference facilities are not provided:

1) The instructions CSECT, DSECT, EXTRN, ENTRY, CXD, DXD, and COM.
2) V- and Q-constants.

The following macro language and conditional assembler facilities are not supported:

1) The instructions ACTR, AGO, AIF, ANOP, GBLA, GBLB, GBLC, LCLA, LCLB, LCLC, MACRO, MEND, MEXIT, MNOTE, SETA, SETB, and SETC.
2) Symbolic parameters and the system variables &SYSNDX, &SYSECT, and &SYSLIST.

The following assembly formatting instructions are not supported:

EJECT, ICTL, ISEQ, PRINT, PUNCH, REPRO, SPACE, TITLE.

The CCW and COPY assembler instructions are not supported.

Only the length attribute, L', may be used in expressions. The type, scaling, integer, count, and number attributes are not supported.

Only one continuation card is permitted for any statement, including macro-instructions.

At most three terms and one level of parentheses may be used in expressions. Complex relocatability is not supported, since there are no external symbols. No attributes other than the length attribute are supported.

The following facilities for the definition of constants as literals or as operands of DC and DS statements are not supported:

1) V- and Q-constants.

2) Exponent and scale modifiers. Length modifiers must be unsigned decimal integers; bit length specifications and expressions as modifiers are not supported.

3) Multiple operands. However, multiple constants in a single operand may be specified for all types except C, X, B, and S.

4) Length specifications greater than 256 bytes, even in DS statements with C or X types.

Only one register may be specified in each USING or DROP instruction.

Each operand of a CNOP instruction must be a decimal digit; expressions are not permitted

The operand of an END instruction, if coded, will be ignored. The entry point of a program is determined, instead, by the XEO or XEO1 system commands.

APPENDIX C

I/O Caveats

Because I/O functions require intervention by the Operating System, many of the safeguards against abnormal termination which PAMELA normally provides are necessarily circumvented during I/O operations. To some extent, therefore, a user must perform input and output at his own risk. Whenever practicable, I recommend that the Display and Alteration Facilities be used to simulate card reader and printer I/O, as the former provide a reasonably efficient and much less hazardous means of modifying and examining the contents of core storage. If it is necessary to perform "live" I/O, however, the following rules should be observed.

Insure that all DDNAME parameters in DCB instructions are valid DD names of card files submitted with the job, and that each DCB is open before a GET or PUT referring to it is executed. Failure to observe this rule will probably result in an abnormal termination with a system completion code of 0C1.

Never allow the object code generated by I/O macros to be modified in core. In particular, the common practice of dynamically modifying the EODAD field of a DCB is not permissible under PAMELA, and will very likely cause abnormal termination of the system if the end-of-data condition is raised.

Insure that any existing data sets are <u>closed</u> before issuing an ASM system command; otherwise, the new program may overlay the old DCB fields. General havoc will then descend when the Operating System attempts to free the devices bound to the nonexistent control blocks.

APPENDIX D

System Messages

Error and informational messages are written by the system in the message area of the CC-30 TV screen. Messages once written are not automatically erased. They may be blanked over using the space bar if desired, but otherwise they will remain displayed until overwritten by a different message. If, after a particular message is displayed, the same error is committed a second time, the cursor will trace over the message a second time; thus there is no danger of confusion as to whether or not a message is still in effect.

Below is an exhaustive, alphabetized list of all system messages and an explanation of each.

DATA INTERRUPT -- A type 7 program interruption has occurred during execution. See Ref. 4 for details.

DECIMAL DIVIDE INTERRUPT -- The quotient from a Divide Decimal (DP) instruction was too large. A type 11 program interruption occurred.

DECIMAL OVERFLOW -- A decimal arithmetic instruction caused an overflow. A type 10 program interruption occurred.

END OF ASSEMBLY ERRORS -- A DERR system command was issued, but all errors in the most recent assembly had already been displayed.

END OF FILE -- An end-of-data condition was raised. Either a card source file, a card data file, or the disk source file has been read in its entirety

ERRORS IN ASSEMBLY -- An ASM command was issued, and assembler language errors were detected in the source code

EXECUTION INTERRUPT -- An Execute (EX) machine instruction referred to another Execute instruction. A type 3 program interruption occurred

EXPONENT OVERFLOW -- A floating-point arithmetic instruction caused an overflow. A type 12 program interruption occurred.

FILE NOT FOUND -- An NPUT system command specified a card file as source input, but no file with the specified DD name was included in the job

FIXED-POINT DIVIDE INTERRUPT -- The quotient from a Divide (D or DR) instruction was too large A type 9 program interruption occurred.

FIXED-POINT OVERFLOW -- A fixed-point arithmetic instruction caused an overflow. A type 8 program interruption occurred

FLOATING-POINT DIVIDE INTERRUPT -- A Floating Point Divide (DE, DD, DER, or DDR) instruction specified a divisor of zero. A type 15 program interruption occurred

INSTRUCTION COUNTER OUT OF RANGE -- The instruction address specified by the current PSW is outside the program area

INSUFFICIENT CORE STORAGE -- The program being assembled could not be
ASSEMBLY TERMINATED

loaded into the main storage space available.

INVALID COMMAND -- A command was issued which was not a valid operation, or was not in the proper format

INVALID OPERAND -- The input to an alter storage command, or one or more system command operands, were not in an acceptable format

LABEL NOT FOUND -- The operand of a system command was taken to be a label, but no statement having that label had been correctly assembled.

LINE ERROR -- WILL RETRY -- A CC-30 I/O error occurred during transmission or reception of data The operation will be retried one time. If the keyboard does not unlock shortly after this message is written, the system has probably terminated.

LOCATION OUTSIDE PROGRAM AREA -- An ACOR or DCOR system command has requested a core location outside the program area.

MISPLACED START -- ASSEMBLY TERMINATED -- Statements causing core storage to be reserved were encountered prior to the START statement in an assembly; or, more than one START statement was encountered.

NO 'END' -- ASSEMBLY TERMINATED -- No END statement was found in an assembly.

NO SUCH STATEMENT CORRECTLY ASSEMBLED -- A label or statement number was specified as the operand of a system command, but the specified statement contained an error, or else no assembly has been performed at all upon it.

OPERATION INTERRUPT -- A type 1 program interruption has occurred during execution. See Ref. 4 for details

PAMELA TERMINATING -- An END system command has been issued. After the system writes this message, the phone connection will be broken.

PLACE CURSOR -- Default cursor placement was requested, but is not applicable under the rules of Chapter II.

PRIVILEGED OPERATION INTERRUPT -- A type 2 program interruption has occurred during execution See Ref. 4 for details.

PROGRAM APPARENTLY IN LOOP -- One thousand machine instructions have been executed without encountering a stop, error, or end-of-data condition.

RECORD NOT FOUND -- A source handling system command has specified a label which has not been assembled into the symbol table, or a statement number which does not appear in the source file

REFERENCE TO LOCATION OUT OF RANGE -- A machine instruction specified a core location outside the program area.

SEQUENCE NUMBER TOO LARGE -- An NPUT system command specified an ini-
INPUT TERMINATED

tial statement number and increment which caused an input statement to be assigned a number greater than 999.999.

SIGNIFICANCE INTERRUPT -- A type 14 program interruption has occurred during execution. See Ref 4 for details.

SPECIFICATION INTERRUPT -- A type 6 program interruption has occurred during execution. See Ref. 4 for details.

"STOP" ENCOUNTERED -- Execution was halted because a stop was set at the current instruction address.

TABLE OVERFLOW -- ASSEMBLY TERMINATED -- The number of symbols or the total character length of literals appearing in a program is too large.

YOU MAY NOT ALTER THE 34 HIGH-ORDER BITS -- An APSW system command
OF THE PSW

was issued, and an attempt was made to alter one or more of the system mask, protection key, AMWP, interruption code, or instruction length code fields.

APPENDIX E

Summary of

System Commands

Command	Function	Operand 1	Operand 2	Operand 3
ACOR	Alter Core	HL(R)		
AFPR	Alter F-P Reg	IL(R)		
AGPR	Alter Gen Reg	IL(R)	IL	
APSW	Alter PSW			
ASM	Assemble			
ASRL	Alter Source Long	NL(R)	N	
ASRS	Alter Source Short	NL(R)	N	
CLR	Clear	I		
DCOR	Disp Core	HL(R)		
DEL	Delete Source	NL	N	
DERR	Disp Errors			
DFPR	Disp F-P Reg	IL(R)		
DGPR	Disp Gen Reg	IL(R)	IL	
DPSW	Disp PSW			
DSRL	Disp Source Long	NL(R)	N	
DSRS	Disp Source Short	NL(R)	N	
END	End Session			
NPUT	Input Source	NL(R)	N	L

Command	Function	Operand 1	Operand 2	Operand 3
STOP	Set Stops	HNL		
XEQ	Execute	HNL		
XEQ1	Execute Single	HNL		
XLAT	Translate	HL(R)		

Guide to Operand Codes:

I -- Decimal Integer

H -- Hexadecimal Integer

N -- Decimal Number

L -- Symbol

(R) -- Required

APPENDIX F

Abbreviations Used

BDAM	Basic Direct Access Method
BISAM	Basic Indexed Sequential Access Method
BSAM	Basic Sequential Access Method
BTAM	Basic Telecommunications Access Method
DAF	Display and Alteration Facility
DCB	Data Control Block
DD	Data Definition
EBCDIC	Extended Binary Coded Decimal Interchange Code
EXCP	Execute Channel Program
IBM	International Business Machines
ISAM	Indexed Sequential Access Method
I/O	Input/Output
JCL	Job Control Language
MFT	Multiprocessing with Fixed number of Tasks
OS	Operating System
OSA	Operating System Assembler language
PAMELA	Program Assembly and Monitored Execution for Learning Applications
PCI	Program Controlled Interruption
PSW	Program Status Word
QISAM	Queued Indexed Sequential Access Method

QSAM --	Queued Sequential Access Method
SYSCB --	System Control Block
TV --	Television
TWX --	Teletypewriter Exchange

LIST OF REFERENCES

1. Computer Communications, Inc., CC-30 Communications Station Reference Manual, 1968.

2. International Business Machines Corp., IBM System/360 OS Assembler Language, Form C28-6514, 1964.

3 International Business Machines Corp., IBM System/360 OS Job Control Language, Form C28-6539, 1968.

4. International Business Machines Corp, IBM System/360 Principles of Operation, Form A22-6821, 1968.