ABSTRACT

This project develops a flexible and responsive data filing system for maintaining records and printing reports at a child day care center. Attributes of the program include the capability of the user to tailor the number of fields and the length of the fields in a file record to individual needs. Facilities included in the program provide the user with the capability to define up to eight individual lists for a file, to move a record from one of the file lists to another, to order records within a list on any field, and to specify line printer outputs. A user friendly interface for inexperienced operators is also emphasized.

A secondary objective for this project was to investigate and implement means of improving response times of interactive programs written in BASIC. Methods of achieving this goal, for the particular microcomputer being used, include the design of efficient BASIC algorithms, "moving" string data by modifying system variable addresses and using machine language subroutines. One main area of system development was writing and linking assembled subroutines, which provide more responsive input of data than could be obtained using BASIC programming, to provide editing capabilities for a forms-type screen data input.

Another area of major effort was developing procedures and algorithms for controlling a line-printer output using format information entered by the user. This includes the capability to specify entire records or record fields and provides functions for headers, footers, centering and justifying outputs. A format interpreter routine provides print-time control for the number of records and number of copies desired.

In addition to providing a needed service, this project entailed research into memory mapping, operating system functions and machine programming which provide routines and procedures directly applicable to future user-oriented interactive programming.
CONTENTS

ABSTRACT 1

CONTENTS 2

FIGURES 4

I Project Overview
1. Objectives I-1
2. Program Implementation I-1
3. Report Organization I-1
4. Program Constructs I-2
5. Conclusions I-3

II Background
1. General II-1
2. Program Structure II-3
3. User Interactive Controlling II-6
4. Programming Considerations II-8
   References II-13

III Main BASIC Program III-1

III-1 Program Initialization
1. Function III-1-1
2. Interactive Control III-1-1
3. Program Description III-1-4
   Program Identifiers III-1-7
   Initialization Listing III-1-8

III-2 Main Menu
1. Functions III-2-1
2. Interactive Control III-2-1
3. Program Description III-2-2
   Main Menu Listing III-2-3

III-3 Constants
1. Function III-3-1
2. Interactive Control III-3-1
3. Program Description III-3-2
   Constants Listing III-3-3

III-4 Records
1. Functions III-4-1
2. Interactive Control III-4-1
3. Program Description III-4-2
   Pseudocode for Records III-4-5
   Records Listing III-4-6
III-5 Formats
1. Functions III-5-1
2. Interactive Control III-5-1
3. Format Description III-5-2
4. Program Description III-5-2
  Formats Listing III-5-4

III-6 Printer
1. Functions III-6-1
2. Interactive Control III-6-1
3. Program Description III-6-2
  Pseudocode for Printer Interpreter III-6-4
  Printer Interpreter Listing III-6-5

IV Machine Language Subroutines IV-1

IV-1 Initialization and Linking
1. Functions IV-1-1
2. Main Routine Linking IV-1-1
3. Procedure Description
   Initial Linking Listing IV-1-2
   USR0: Entry for Calls IV-1-3

IV-2 USR1: String Data Moves
1. Function IV-2-1
2. Procedure Description
   USR1 Listing IV-2-1

IV-3 USR2: Screen Data Input
1. Functions IV-3-1
2. Procedure Description
   USR2 Listing IV-3-2

IV-4 USR3: String Length
  Function IV-4-1
  USR3 Listing IV-4-1

IV-5 USR4/USR5: String Address Moves
  Function IV-5-1
  USR4 and USR5 Listings IV-5-1

V Program Listings

V-1 FLX/BAS Main Program
  Identifiers V-1-1
  FLX/BAS Listing V-1-3

V-2 FLXCMD/CMD Assembled Listing V-2-1
<table>
<thead>
<tr>
<th>FIGURES</th>
<th>II-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>II-1</td>
<td>File Definitions</td>
</tr>
<tr>
<td>II-2</td>
<td>Program Structure</td>
</tr>
<tr>
<td>II-3</td>
<td>Menu Selections</td>
</tr>
<tr>
<td>II-4</td>
<td>System Design Principles for Inexperienced Users</td>
</tr>
<tr>
<td>III-1-1</td>
<td>Screen Display During Initialization</td>
</tr>
<tr>
<td>III-1-2</td>
<td>New File Screen Displays</td>
</tr>
<tr>
<td>III-2-1</td>
<td>Main Menu Screen Display</td>
</tr>
<tr>
<td>III-3-1</td>
<td>Changing Record Field Names</td>
</tr>
<tr>
<td>III-4-1</td>
<td>Screen Display for Records</td>
</tr>
<tr>
<td>III-4-2</td>
<td>Screen Display while Ordering</td>
</tr>
<tr>
<td>III-5-1</td>
<td>Formats Editing Screen Display</td>
</tr>
<tr>
<td>III-6-1</td>
<td>Screen Display for Printer</td>
</tr>
<tr>
<td>III-6-2</td>
<td>Class Roster Printout</td>
</tr>
</tbody>
</table>
I. Project Overview

1. Objectives.

   a. The primary objective of this project was to develop a general-purpose data file program for use in maintaining class rosters of children in a church-sponsored day care center. The application program was developed for use on a Radio Shack Model III microcomputer with at least one floppy disk drive and 48k bytes of user RAM. Details on the program design objectives are contained in the Background section of this report.

   b. A secondary objective of the project, from the developer’s point of view, was to create modularized routines of utilities which could be used in future program developments. This objective applies particularly to the machine-language subroutines and to a lesser extent to facilities within the main BASIC program.

2. Program Implementation.

   a. The program developed for this project, named FLX, is currently in use for its intended purpose. The primary operator, who has no background in computers, has found the program easy to use and responsive to the needs of the child care center. The user-defined report generator has been particularly useful in providing printouts in different formats used for formal class rosters, attendance records, money collection sheets and class reassignments.

   b. In addition to providing a record keeping facility for the day care center, FLX is also used for maintaining rosters for a boy scout troop, for making mailing labels for a child-care association and for maintaining household inventories.

   c. Program development has continued to evolve where an opportunity for increased efficiency has been detected. The latest program modifications included changing from the TRS-DOS to LDOS operating systems and changing the line printer used with the system. Although these changes required internal programming modifications to change the disk-directory screen display and to add new printer control commands, no apparent changes, except the disk-file directory display, were made from the user’s viewpoint.

3. Report Organization. This project report is comprised of five major sections:

   I. Project Overview. This section provides a project summary and conclusions.
II. Background. This section contains the program objectives, file definitions, interactive control design and programming considerations used for program development.

III. Main Program. The main BASIC program is divided into its principle operating modules (program initialization, main menu, change program constants, records, formats for printing and printer control) which are then further divided into functional entities for description.

IV. Assembled Subroutines. This section details the five assembled subroutines which provide supplementary functions for the main BASIC program. The most extensive of these subroutines is a video screen cursor control and data entry facility which is used for providing a variety of operator inputs to the main program. This section contains the assembly language listing for the subroutines and describes subroutine linkage and parameter passage.

V. Program Listings. This section contains the complete listing for FLX/BAS and the assembled listing for FLXCMD/CMD.

4. Program Constructs.

a. Both the main BASIC program and the assembler subroutines utilize structured programming with the use of GOTO/JP instructions reserved for constructing "while" loops or for skipping over conditional instructions in order to provide a single exit point for a called routine.

b. The case-structured ON nn GOSUB is used in conjunction with the INSTR instruction to provide an effective means of translating operator menu selections into program controls. This programming practice retains structured form and is efficient both in operation and program space utilization when compared with nested IF THEN ELSE instructions. An added benefit, from the programmer's point of view, is ease in changing menu selections by simple modifications to the INSTR hunt list and GOSUB address lists.

c. Program responsiveness has been enhanced during file processing by performing record insertions, deletions and sorting using the variable address table for a string array rather than by actual movement of string data. Although BASIC provides access to variable addresses with the VARPTR instruction, other commands in the language do not provide effective means for movement of the addresses in the tables. Two of the machine-language subroutines, therefore, are used to provide rapid block moves of address-table information for BASIC processing functions.
d. The printer format scheme used to enable operator specification of reports is the most difficult part of the program for an inexperienced operator. It is also one of the most difficult parts since the programmer has to provide a high degree of flexibility while retaining responsive operation. The requirements for responsiveness are, however, tempered to some extent by reduced user expectations towards the printer's speed to output reports. The convenience of this facility has proven far superior to the easier approach of providing hard-coded printout routines which can be altered only by modifying the program code.

e. The overall response of the BASIC interpreter has proven adequate for most program applications involving interactive control. The requirement to develop a machine language routine for screen input data resulted from degrading operation of a BASIC routine with too many special functions such as inserting and deleting letters anywhere in an input field. In other applications, methods have been found to improve responsiveness through program structure. One of these techniques, which reduced the sorting time for file records by two-thirds, is declaring time-sensitive variables early in the program using the DIM instruction. In long programs with many variables, this practice reduces the look-up time needed to evaluate these values since they are placed near the start of the variable table.

f. The disk files for FLX have been configured using the BASIC random file facility. This provides a convenient means of updating the file parameters on disk as changes are made and guards against the loss of current data in event the program is terminated prematurely. Additionally, string data such as the printer report formats can be accessed from the disk only when needed rather than being retained in memory.

5. Conclusions.

a. Interactive program design for a microcomputer adds a new dimension to programming not required in main-frame operations involving batch processing. Superimposed upon the principal program objectives are the requirements for screen display design, operator prompts, data inputs and error trapping. These control functions can easily end up taking up the bulk of a program and are limited only by the programmer's imagination on how much and how to communicate with the operator. Depending upon the available memory which can be devoted to program as opposed to that needed for data, longer and more comprehensive programs than FLX would probably have to be broken up into overlay segments rather than loaded as a complete program.

b. The BASIC interpreter used for this project recognizes identifiers only up to two characters. Although
longer strings can be used in programming, they add nothing to the maximum number of identifiers available and introduce possible errors of containing included reserved words. As the identifier list grows to approach a hundred variables, as in FLX, it becomes difficult even for the program originator to retain them all straight in his mind. However, interpreters which allow longer, and therefore more expressive, variable definitions do so at the expense of increased memory requirements for storing the resulting longer programs.

3. The language tools and immediate interactive response of the BASIC interpreter provides a convenient and powerful tool for program development and debugging on a microcomputer. However, in a production environment, compiling a program of any length into machine code would be superior to interpreter operation. Such a compiler is soon to be released by Radio Shack for the computer used in this project and could provide an even more effective "final" FLX program.

4. Finally, there is a need for a user's manual to supplement instructions which appear on the screen in the FLX program. It was neither practical nor effective to display all aspects of the interactive control features, such as keyboard control for cursor movement while entering screen data, every time they were available. Additionally, even with an extensive screen display, all aspects of composing a printing format could not be shown in the space available on the screen. Further evaluation of the current screen messages used in FLX and developing a user's reference will be the next step of system development.
II. Background

1. General.

This project was prompted by the needs of the director of a church-sponsored child care center who maintains data files of children attending a Mother's Day Out (MDO) center. Additional benefits of a computerized data filing system include staff rosters, attendance sheets and fee collection sheets, in addition to the primary requirements for timely class rosters for each of seven rooms.

About 70 children normally attend the day care center so that the data handling requirements of the MDO Director are such that a single user with the real-time capability available with a microcomputer could adequately fulfill the MDO center's needs. The following capabilities were included as requirements for the program developed:

a. File Definition (Figure II-1).

(1). Data records are partitioned into lists or subfiles which enable the user to define up to eight individual lists for each main file. An arbitrary design limit of a maximum of eight subfiles was chosen for program development as sufficient to accommodate the maximum foreseeable number of classes in the daycare center. This feature provides the capability for either maintaining children's records in separate subfiles by room assignments or allowing single file definitions for maintaining staff rosters and inventory data.

(2). The number of fields in a record and length of the record can be specified by the user.

(3). Additions, deletions and movement of records from one file to another within a data set can be accomplished easily.

(4). Records within a file can be ordered either in ascending or descending order based upon any field.

(5). An option to use the last field of a record for numeric totaling allows the use of the program for inventory purposes.
File Definition

Lists have a common designation specified for the file and are distinguished by a sequence number in the last position of the file extension designation (such as MD0/DF0, MD0/DF1, MD0/DF2, etc). Data file #0 is used to maintain common file set parameters and is required regardless of the number of sub-files in the set. Record length for data records can vary, depending upon initialization by the user, up to the system maximum length of 255 bytes for a string variable.

File #0

<table>
<thead>
<tr>
<th>Field</th>
<th>Length (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>18</td>
</tr>
<tr>
<td># Files in set</td>
<td>2</td>
</tr>
<tr>
<td># Fields in record</td>
<td>2</td>
</tr>
<tr>
<td>Length of record</td>
<td>2</td>
</tr>
<tr>
<td>Number of fields in record</td>
<td>2</td>
</tr>
<tr>
<td>Length of fields (7 * 2)</td>
<td>14</td>
</tr>
<tr>
<td>Code if numeric field</td>
<td>2</td>
</tr>
<tr>
<td>Code for capital key lock</td>
<td>2</td>
</tr>
<tr>
<td>Number of printer formats</td>
<td>2</td>
</tr>
<tr>
<td>Numeric value sums (8 * 4)</td>
<td>32</td>
</tr>
<tr>
<td>Number or records (8 * 2)</td>
<td>16</td>
</tr>
<tr>
<td>Field names (7 @ 14)</td>
<td>98</td>
</tr>
<tr>
<td>TOTAL Rec #1</td>
<td>192</td>
</tr>
</tbody>
</table>

Record 2

| Record 2                      |                 |
| Internal file names (8 @ 24)  | 192             |

Record 3 to n

| Record n                      |                 |
| Printer format codes (ea)     | 192             |

Maximum number of files in set 9 (incl. file #0)
Maximum records in a file Program dependant

Figure II-1
b. User Interface.

(1). The program requires little computer experience or practice on part of user to set up and run.

(2). Data can be easily entered, reviewed and edited on the screen without sluggish response to user instructions.

c. Interactive Report Formatting.

(1). A user-defined format for a line printer has been included which will accommodate header, file output and footer capabilities.

(2). Either complete records or selected fields of the records can be printed in a report.

(3). One or more files can be printed on an output page to allow composite report formats.

(4). Designated fields from one or more records can be printed on a line to allow for multiple column mailing label generation.

2. Program Structure (Figures II-2 and II-3).

```
FILE
INITIALIZATION

NEW FILE --> MASTER MENU

CONSTANTS ! RECORDS ! FORMATS ! PRINTER
```

Figure II-2. Program Structure.
Menu Selections

Initialization
- Change disk drive
- Designate old or new file
- End Program

Main Menu
- Change program constants
- Process records in selected file
- Change printer formats
- Print report
- Return to Initialization
- Change selected file (function key)

Constants
- Change internal file names
- Change field names
- Change date
- Return to master menu

Formats
- Add new format
- Edit existing format
- Return to main menu
- Change to next format (function key)

Print Report
- Select printer format (function key)
- Designate number of files to print
- Designate number of copies to print
- Return to Master menu

Records
- Select record (function key)
- Edit record
- Add Record
- Delete record
- Find record
- Move record to another file in set
- Order records
  - Ascending/Descending
  - Designate key field
- Return to Master menu

On scope record editing (all function key)
- Enter/overstrike field contents
- Insert letter
- Delete letter
- Clear to end of field
- Move forward
- Move backwards
- Tab to next multi. of 8
- Move to next field
- Move to prior field
- Enter scope data

Figure II-3
a. Initialization. This procedure meets housekeeping requirements of clearing memory, loading and linking machine language subroutines, defining constants and functions, displaying the directory of existing files and providing a choice to create a new file or process an existing file. Depending upon the choice made, the routine will either create a new file set or initialize an existing file set prior to entering the master control portion of the program. Final program termination will also be from this block so that additional file sets can be processed without the requirement to reinitialize from DOS level.

b. New File. This procedure provides for inputting information necessary to set up a new file set. Inputs include definition of field and record lengths, number of files desired in the set, and field and file internal names.

c. Master Menu. Once a file set is initialized, this procedure provides overall operating control for processing the records within the files. User choices provide for entry into submenus for altering program variables, processing records, line printer control or return to file set initialization. The capability to change which file in a set is to be processed is included in the interactive controls of this procedure.

d. Constants. This procedure provides menu selection for changing internal file titles, field names and the date. Changes to these items may be entered within their normal screen presentation locations after the item to be changed has been designated by the operator. Control is returned to master menu upon exit from the procedure.

e. Records. This procedure is the primary control for record processing after the desired file is selected under control of the Master Menu. A submenu is provided to allow choice between adding, deleting, editing, locating, ordering and moving records. Interactive controls allow easy review of file records in either a forward or backward direction as well as the facility to locate a record with a given substring input. Screen input data and editing of records is provided by this menu. The file being processed is rewritten if changes have been made to the file and control returned to the Master Menu upon exit from the Records procedure.

f. Formats. This routine allows entering and editing the format records which are used by the printer routine to provide user-defined report generation. Menu selections are used to add a new format, edit the format currently displayed or return to the main menu. The down arrow is used as a function key to index to the next, or first, existing printer format.

g. Printer. The line printer output is controlled by two user inputs, one to designate the format
of the reports and the other to specify the records and number of copies to be printed. This procedure provides for choosing the format desired for printing and for controlling the line printer operation. Interactive choices provide for designating which format is desired, the number of files within a file set to be printed and the number of copies desired. Control is returned to the Master Menu upon exit from this procedure.

3. User Interactive Control.

a. Two principal aspects of interactive control in a real-time environment are the program's instructions to the user and its response time to user inputs. The format used for interactive instructions can vary greatly depending upon the experience of the operator and functional controls available on a given keyboard. Particular care is needed in the design of screen control instructions for the inexperienced user. These principles, defined by Zimmerman (1977) as contained in Bailey (ref A page 295), are detailed in Figure II-4.

b. Responsiveness, on the other hand, does not lend itself to exact qualification but rather seems to be a function of the procedure being accomplished by the computer, coupled with the operator's expectations (ref A, page 322). Continuity of data input and response of less than a second to simple requests appears to be the normal expectations of most users (ref A).

c. Although quantitative analysis of improvements gained in response times was desirable for a project such as this, the hardware is limited to one second timing accuracy. This precluded detailed analysis of all aspects of improvements in response time between various algorithms. However, in some cases such as the sort routine for records in a file, program looping provided sufficient delay times to compare the effectiveness of different BASIC algorithms. These findings are detailed in the Records section of the report.
1. Initiative should always come from the computer.

2. Each required entry should be brief.

3. Entry procedures should be constructed so that they are consistent with user expectations.

4. No special training should be required.

5. The system should act upon certain entries only after a confirmation by the user.

6. Computer messages should be clear and unequivocal.

7. Computer messages should be composed of a simple, generally comprehensible vocabulary.

8. User decision-making should be facilitated by offering a small number of simultaneous options that require only one decision per step.

9. User decision-making should be facilitated by a distinct request from the computer that a decision be made.

10. The user should be able to comfortably control the pace of the human/computer interaction.

11. The user should have the option to easily request both computer and human assistance.

Figure II-4
4. Programming Considerations.

a. Programming Language.

(1) The driver program is written in BASIC to take advantage of the housekeeping facilities available with a higher level language and the interactive control capabilities provided by the BASIC interpreter. There are, however, a variety of shortcomings in BASIC's ability to provide versatile and responsive interactive control which are recognized in microcomputer circles and provide the central theme for numerous magazine articles as well as for Rosenfelder's book, "BASIC Faster and Better" (ref I). Enhancing the responsiveness of the main BASIC program while retaining the facilities that the BASIC interpreter provides for interactive control of the microcomputer being used was a prime objective of this project.

(2) Ideas on how to implement effective user menu selections and screen displays using BASIC are included in most books on BASIC programming and are a popular topic in microcomputer magazines. Of necessity, a major consideration in program development was to aspects of interactive control such as single key entry of alphabetic letters for menu selections, error trapping and error messages, and understandable screen displays. This process of providing a friendly interface can, however, result in reduced responsiveness of the program to user inputs if excessive IF-THEN-ELSE statements or extended line searching by the BASIC controller are required to implement the controlling input. A variety of different techniques are employed by Rosenfelder to improve responsiveness of BASIC programs, many of which fall into the "tricky" category normally discouraged or prohibited by software restrictions in mainframe program structuring. Techniques such as use of Boolean operators where feasible to replace IF-THEN-ELSE statements does improve program speed and were incorporated into the final program.

(3) Programming methods needed to minimize the memory requirements for an interpretive program are diametrically opposite to good documentation procedures developed for compiled programs where remarks and other unneeded space (insofar as the machine is concerned) can be left in the source code. The overhead for each BASIC line entry is five bytes, two for next line pointer, two for the line number and one for a terminator (ref B, p208), thus requiring six bytes for a one byte NEXT statement entered with a separate line number while only two bytes, including delimiter, are required in compound statement lines. In addition to taking up unnecessary memory, single statement, commented program lines run slower than compressed, multistatement code since more lines have to be examined for statements such as the GOSUB. One solution to documenting compressed programs, as taken by Rosenfelder
(ref I), is to show the code as it should be in the functioning program and use external documentation for program explanations. Other authors, such as Barden (ref B), prefer to show a 'textbook' solution of the code, along with a comment that remarks should be removed and the code should be compressed in order to run effectively. The first solution does not require redundant coding and appears to be an effective manner for writing and documentation of a lengthy BASIC program in which responsiveness is a primary consideration.

b. File Maintenance.

(1) The objective of the program is to maintain records which can be easily defined and manipulated by the user. System planning requirements indicate that the program should be capable of moving records from one file to another in an active file set of up to eight subfiles and provide the capability to add, modify, delete, and reorder records within a given file. Providing the user with total flexibility to define the number of fields in a file, length of fields and the option for a numeric summation field will add complexity to the program but are within the capabilities of BASIC programming techniques.

(2) No apparent advantage for a choice between sequential or random access file storage was initially foreseen since reordering files after records are added or deleted, which is a principal use for the program, will require that a file be stored in RAM and rewritten after being edited. A final choice for random files was made based on easier programming to guard against not having changes posted to the disk files if the program is terminated prematurely.

(3) The microcomputer used for this project has sufficient RAM to allow the total amount of data envisioned for the child care center to be stored in one disk file and loaded completely into memory as an array. Improved system response in switching from one list to another might be gained by having all records in memory at the same time. However, the disadvantages introduced in program complexity (multiple subscripting of records), unused memory allocation for arrays of the maximum possible number of total records, and possible loss of all updating in event of program interruption outweigh the disadvantages of multiple disk reads and writes required in processing single files for this application.

(4) The BASIC built-in function for sorting files allows ordering only on the first field in a record. The program must therefore provide its own sorting routine to meet stated requirements. Extensive experience with a microcomputer is not needed to recognize that sorting by
string movement is sluggish in BASIC and should be avoided when inserting, deleting or reordering records. Options for avoiding this include the use of subscripted subscript notations, ordering data addresses rather than data using the VARPTR (variable parameter address) function to reorder a string array directory (ref I) or calling machine language subroutines to reorder strings. The sort routine in FLX is an insertion sort using a BASIC driver to handle data comparisons and an assembled subroutine to move the record addresses in the string array directory.

c. Data Input.

(1) The BASIC INPUT command provides an efficient method to enter program variables from the screen but is not an effective way to input information for an interactive data file. Specifications for commercial data handling programs shows such as Radio Shack's "PFS:FILE" and "Profile III" (1984 catalog) almost universally provide an input scheme in which the user can enter and edit data easily within a record displayed on the screen. The desirability to provide such a "forms" type of input for BASIC programs is a topic included in both Barden's and Rosenfelder's books (refs B & I) on advanced programming techniques. However, neither of these references provide a completely comprehensive approach to integrating program responsiveness with effective interactive interfacing with the user.

(2) In his book, Barden includes a BASIC algorithm which provides a good starting point for a routine to provide for effective interactive input of data (ref B, p 57). Data input is allowed by reading the keyboard with INKEY$ and printing the characters one at a time to the screen with suppressed carriage return. Although this process may appear a simple matter at first glance, complications are introduced when requirements are introduced for keeping track of the cursor's location to prevent overshooting the input field and to allow tabbing and free movement from one input field to another. Additional complications are introduced if text editing to include the capability to overstrike, insert and delete characters is combined with screen data input. In addition to the requirement of screening the keyboard input for special characters which must be introduced to signal that a particular editing action is desired, the capability to shift the remainder of a field beyond an insert/delete point must be provided. Preliminary experimentation with algorithms written in BASIC to provide editing capability for screen input of data showed that BASIC is just too slow for rapid input by a good typist and that a machine language subroutine was required to to control the screen effectively.
(3) Displaying data on the screen is one matter, reading data from it is another. The PRINT command quickly moves data from variables to any location on the screen without noticeable delay. However, to implement the screen input scheme described above, direct movement of data from one memory location to another as needed to transfer data from the screen buffer to the designated record. One solution was to move data from the screen with a PEEK and POKE loop which transfers the information one letter at a time from the screen buffer to the VARPTR address of a variable. This method produced a noticeable lag when data of any length over just a few characters was moved from the screen to a memory location and is a classic application for a machine language subroutine using the Z-80’s LDIR/LDDR block move commands (refs C & M). Development of the screen data entry algorithm discussed above therefore included transfer of the edited input data to the proper variable in memory.

d. Line Printer Control.

(1) User definition of printer output is a desirable feature which provides independence of data output format from the control program and does not require that someone with programming knowledge adapt the program to unforeseen outputs. Although many books on BASIC provide extensive coverage in writing printer output routines for inclusion in a program, none thus far reviewed consider algorithms for converting user inputs to printer outputs during run time.

(2) Commercial programs, such as the word processor being used for this proposal, provide an insight into desirable capabilities of user-defined printer output. Formatting controls should provide for user definition of top, bottom, right and left margins, number of lines between paragraphs or lines, centering, flush right or left, and header and footer control. Additional capabilities which are desirable for data files include individual column control for mailing labels and control of how many files are grouped on each outputted page.

(3) An extensive interpretive routine is included to provide user input definition and control flexibility for a line printer. The BASIC routine used shows some obvious delays in composing string outputs, however, the slower printer device also requires much less responsiveness than data input to the screen and a machine language subroutine was not included to provide user control of a line printer.

e. Machine Language Subroutines.

(1) Considerations in using machine language subroutines with a BASIC program as with any other language
include not only writing the subroutine itself in an assembly language but also in linking the programs together. A number of excellent references (ref C,E,K,M) are available pertaining to programming the Z-80 CPU and the vendor provides technical information on routines which can be accessed in ROM or the DOS (ref F,G,H). None of these references, however, provide a comprehensive approach to all of the requirements for data input from the screen identified above. Most of the authors agree that the effective use of machine language subroutines includes considerations such as passing multiple parameters with the USR functions, defining multiple utilities, considering relocatable vs nonrelocatable routines and calling available operating system routines when appropriate.

(2) Prior to initiating any planning for machine language subroutines, a decision between using relocatable or nonrelocatable routines must be made. Both Barden (ref B) and Rosenfelder (ref I) prefer relocatable routines that, after assembly, can be carried alone with the BASIC program as an encoded DATA input or as a data file record. Advantages of this method include transportability from a system of one memory size to another without wasted memory on the larger machine, no requirement to reserve high memory for machine language upon entering the BASIC interpreter and maintaining both BASIC and machine language routines in a single BASIC program. Disadvantages include length constraints of 255 bytes for routines if maintained as string variables, loading delay time if read in as DATA, the possibility for unintentional destruction by using a BASIC variable for machine code and restricting the use of Z-80 CALL and LD instructions which require explicit addresses.

(3) The disadvantages of providing relocatability were considered to outweigh the advantages for a machine program of any length when viewed with respect to the limitations which would be placed on use of the Z-80 instruction set and the possibility of inadvertent interference by the main routine. These restraints would inhibit structured programming by requiring that only relative jumps of less than 127 bytes rather than calls with returns be used and would limit the storage of variables used in the subroutines to registers or the CPU stack. The disadvantages of not maintaining relocatability can be overcome by reassembling the source program for different machine memories sizes, if ever necessary, and by providing automatic DOS initialization of memory for machine subroutines to relieve the inexperienced operator of this task.
References


B. Barden, William Jr 'Programming Techniques for Level II BASIC' Radio Shack, 1980

C. Barden, William Jr 'TRS-80 Assembly Language Programming' Radio Shack, 1979

D. Furrour, James Lee 'TRSDOS 2.3 Decoded and Other Mysteries' IJG Inc, 1982

E. Radio Shack Handbook 'Series I Editor Assembler' 1981


I. Rosenfelder, Lewis 'BASIC Faster and Better & Other Mysteries' IJG Inc, 1981


L. Wagner, Michael J. 'Machine Language Disk I/O & Other Mysteries' IJG Inc, 1982

M. Zaks, Rodnay 'Programming the Z80' SYBEX Inc, 1980