A Software Shell for Visually Impaired Applications

by

Krishnaswami Srinivasan

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Electrical Engineering

APPROVED:

Dr. Charles E. Nunnally,
Chairman.

Dr. James R. Armstrong

Dr. Joseph G. Tront

June, 1986

Blacksburg, Virginia

A Software Shell for Visually Impaired Applications

by

Krishnaswami Srinivasan

Dr. Charles E. Nunnally, Chairman.

Electrical Engineering

(ABSTRACT)

An approach to introduce the visually impaired to personal computers is presented in this thesis. The PC used for this work was an IBM PC Portable. Use of the resident software developed in conjunction with a Votrax Voice Unit can greatly simplify PC applications for the visually impaired. Further, a method to communicate with a mainframe is also presented. Almost all of the commonly used DOS application software are supported by the software presented in this thesis.

Two modes of operation are possible. The advantages and differences between these two modes are considered. A detailed discussion on the software implementation is also presented. A method to develop resident programs that need to trap PC BIOS vectors is presented.

It should be noted that the shell concept presents a shell of user invoked resident applications and not a group of subprograms which can be used by other applications.

ACKNOWLEDGEMENTS

This work is dedicated to my parents, Mrs. & Mr. Srinivasan Krishnaswami.

I would like to thank Dr. C.E. Nunnally, my committee chairman and advisor for his invaluable guidance and encouragement throughout this project. I would like to take this opportunity to say that I have enjoyed every moment of working with you, sir.

I would also like to thank Susanne Boyne, Dr. Virgil Cook and Dr. Charles Holt for the help rendered by them in various ways.

Finally, I would like to thank Dr. Armstrong and Dr. Tront for kindly consenting to be on my committee.

TABLE OF CONTENTS

1.0 Introduction 1

1.1.1 General Problem 1

1.1.2 Current Support. 4

1.1.2.1 PC - Talker 4

1.1.2.2 Freedoml 5

1.1.3 An Alternate Solution 6

1.1.3.1 Reconfigurable Keyboards 6

1.1.3.2 The Shell concept. 7

2.0 System Basics 10

2.1.1 Introduction 10

2.1.2 The Votrax Features 10

2.1.2.1 Interface Details 11

2.1.2.2 Data Protocol 15

2.1.2.3 Data Types 15

2.1.2.4 Initialization 16

2.1.3 Why Assembly Language? 20

2.1.3.1 Program Concepts 20

2.1.4 Functions Provided 21

2.1.4.1 The Static Mode 21

2.1.4.2 The Dynamic Mode 23

2.1.4.3 Advantages 23

3.0 Software Algorithms 25

3.1.1 Introduction 25

3.1.2 The Main Program 26

3.1.2.1 Protection 26

3.1.2.2 Initialization 30

3.1.2.3 Vector Mapping 30

3.1.2.4 PSS Setup 31

3.1.2.5 Resident Program Concepts 34

3.1.3 Keyboard Interrupt 35

3.1.3.1 Command Structure 37

3.1.3.2 Static Operations 38

3.1.4 Video Interrupt 41

3.1.4.1 Control Criteria 41

3.1.4.2 Operation 43

3.1.5 The COM file Setup 45

3.1.5.1 Required Conditions 45

4.0 Screen Reader Functions 47

4.1.1 Introduction 47

4.1.2 The Main Transmit Routine. 47

4.1.3 STATIC Commands 53

4.1.3.1 Introduction 53

4.1.3.2 Invoking Static Functions 53

4.1.3.3 The Change Function Group 54

4.1.3.4 Help Group 59

4.1.3.5 Range Reader Group 62

4.1.4 Mode Interaction 63

5.0 Conclusion 68

5.1.1 Problems Encountered 68

5.1.2 Observations 69

5.1.3 System Cost 72

5.1.4 Future Work 72

Bibliography 74

Appendix A. PC - Votrax Termination 76

Appendix B. Users's Manual 77

Appendix C. PSS ASCII Chart. 93

Appendix D. PSS Command Initiators 94

Appendix E. Interrupt Allocation 95

Appendix F. Program Source Code 98

Vita 151

LIST OF ILLUSTRATIONS

Figure 1. AUDIODATA Keyboard 8

Figure 2. Program Shell 9

Figure 3. Votrax front panel view. 12

Figure 4. Votrax rear panel view. 13

Figure 5. System Configuration 14

Figure 6. DEFAULT.VAL File Structure. 18

Figure 7. Program Structure. 27

Figure 8. The Main Program. 28

Figure 9. BIOS Interrupts. 32

Figure 10. Modified Interrupt Structure. 33

Figure 11. Keyboard Interrupt flow chart. 36

Figure 12. DOS & Static Cursors. 39

Figure 13. Video interrupt. 42

Figure 14. Transmit Routine. 49

Figure 15. Message Identification. 50

Figure 16. Static Command Function Groups. 55

Figure 17. Change Function Description. 56

Figure 18. Mode Overlap. 64

Figure 19. Common Command Chart. 65

Figure 20. Activity Execution Chart. 67

1.0 INTRODUCTION

1.1.1 GENERAL PROBLEM

With the ever increasing presence of personal computers, it becomes more and more evident that the computer has to be used to assist the physically disabled. It is the intent of this work to present and discuss an application of personal computers (PC's) which was developed to assist individuals with visual handicaps.

With the growth of personal computers, such as the IBM PC and the development of enormous number of application programs, the sighted world is beginning to be able to do many ordinary mainframe and individual computing needs at his or her desk. With today's growth in technology, it seems appropriate that technology could tackle the problem of giving this same capability to those with visual handicaps. The thrust of this presentation is to introduce a system, which through combination of software and hardware, permits the visually handicapped to use any common software application, which can be executed on the IBM PC. The term "Visually Impaired" in this thesis, refers to users with zero vision. The IBM PC was selected as the target computer only because of availability.

The concepts and approaches presented in this thesis could be applied to any personal computer.

Further, the increasing number of visually impaired users in engineering and other programs demands that something be done. Consider, as an example the use of a FORTRAN Compiler by a visually impaired user. The time spent by the user to edit, compile and debug the program using conventional reading aids (non-PC based) would be enormous and would put the user at such a disadvantage and would tend to tax the users perseverance towards education.

Now, to elaborate on the problem of using computers by the visually impaired, consider a student in EE2570. The format for a typical assignment in EE2570 would be,

- EDIT the Program.
- COMPILE the created file.
- DEBUG the Program.
- EXECUTE the Program.
- ANALYSE the Result.

Two distinct phases can be identified from these steps. First, a method to review the screen should be provided. A review mode would enable the user to refer back to the information on the screen. The review mode would also be ideal to scroll through a listing file.

The second phase would be an interative mode. The interactive mode would provide real time responses to DOS prompts and compiler generated messages. The interactive mode may also be used to edit the program.

Further, the modes should be readily available. That is, mode invocation should not involve complicated key sequences. Now, the modes should also provide other support features, like an ability to toggle between speech formats of words and letters. If such a system were to be used for the mentioned problem, the impaired user would have almost the same capabilities as a normal user in using PC's.

The problem mentioned here (EE2570) is an ideal example for the need of a powerful utility. This particular problem was brought out by a visually impaired student, then using an APPLE Computer with a Votrax voice unit. The software used by the student did not offer many of the proposed features. Advances in technology now make it possible for the user to control what is being spoken and how.

Use of the software presented does not guarantee a 100 % improvement, but a very significant difference will be made.

The improvement results from having the PC and the attached voice unit do most of the work.

1.1.2 CURRENT SUPPORT.

Numerous utilities exist for "Talking PC's", but most of them are oriented towards particular applications. Word processing is an area that receives maximum attention. Some of the common "Talking" programs are presented in the next section of this chapter.

The software developed in this thesis serves as a general purpose interface. Any reference to the software developed in this thesis will refer to the particular software as Dynstat. Support to other programs is not restricted. The structure of the shell provides this. Programs that would be used by an engineering student are supported. Graphic capability is not provided, but then, with the standard voice output device there exists no method to achieve graphic outputs.

1.1.2.1 PC - Talker

The PC Talker software has a lot of drawbacks. Operation invoking procedures are too complex, producing a tremendous overhead on the user. In particular, problems arise when an application program and the PC Talker use the same keys for function execution. This software is a product of Talking Computers Incorporated.

1.1.2.2 Freedoml

Freedoml [12] is a product of Interface Systems International. The Freedoml approach is an elegant way to solve the problem. In this method, the user "freezes" the screen and examines the page at leisure. This is the "OFF LINE" mode. The off line mode does not provide interactive communication to the user. Freedoml would be ideal for word processing. There are 44 commands supported by Freedoml. The resident software uses data on the screen to produce audio outputs. This prevents the possibility of a real time environment. Hence, the only mode available is the screen reading mode. Freedoml can be invoked only after the "A>" prompt. That is, DOS has a higher priority.

With the software presented in this thesis, the system boots with COMMAND.COM, having control environment, therefore the " A> " is spoken.

1.1.3 AN ALTERNATE SOLUTION

The use of external hardware is a solution to interfacing PC's with voice units.

1.1.3.1 Reconfigurable Keyboards

Reconfigurable Keyboards require a different structure of keyboards. One such device exists on campus in the special services department of the Neuman Library. This device is the AUDIODATA keyboard manufactured by Maryland Computer Services. The layout of the keyboard follows the QWERTY pattern. But, the significant difference is that these keyboards are for dedicated applications. The keyboard mentioned for example, has two wiper switches and the voice unit built in. Figure 1 on page 8 shows the wiper positions. These wiper switches correspond to the cursor motion in the vertical and horizontal directions. No external connection is required by this keyboard. The AUDIODATA requires a special program to be resident on the PC. The wipers move the cursor to the line/word of interest and a key initiates speech. DOS responses are also trapped by the resident software. The major problem is with YTERM, a terminal emulation program [3]. Loss of data is very common when communicating with a mainframe, particularly when using the localnet. When two processes such as YTERM and a Speech program are being executed, a multi-tasking approach should be used. Lack of proper synchronization between the two processes is a major cause for for the loss in transmitted data. Further, anomalies in localnet behavior augment to data loss.

1.1.3.2 The Shell concept.

In the software developed, both "OFF LINE" and "ON LINE" modes are supported. These are referred to as the "STATIC" and "DYNAMIC" modes in this document. The Static mode is similar to the off line approach of Freedoml. The Dynamic mode however, provides interactive communication for all DOS responses. The Dynamic mode provides real time processing of data. The Static and Dynamic modes will be presented in the subsequent chapters.

Dynstat creates a shell between DOS and other application programs. This provides the user with the interactive feature. Figure 2 on page 9 describes the shell concept. The purpose of the shell is to serve as a buffer between DOS and other application programs. All data transfer between DOS and these programs now pass through Dynstat, which selectively passes data to the Votrax.

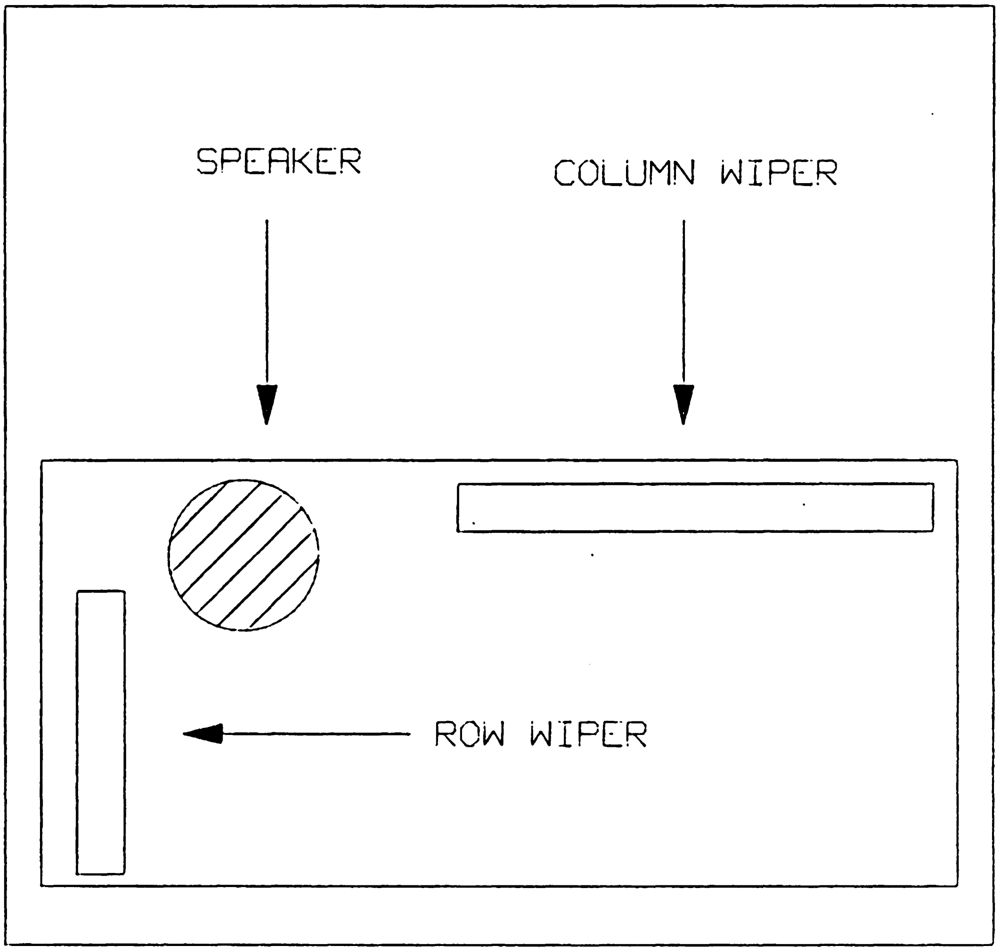


Figure 1. AUDIODATA Keyboard

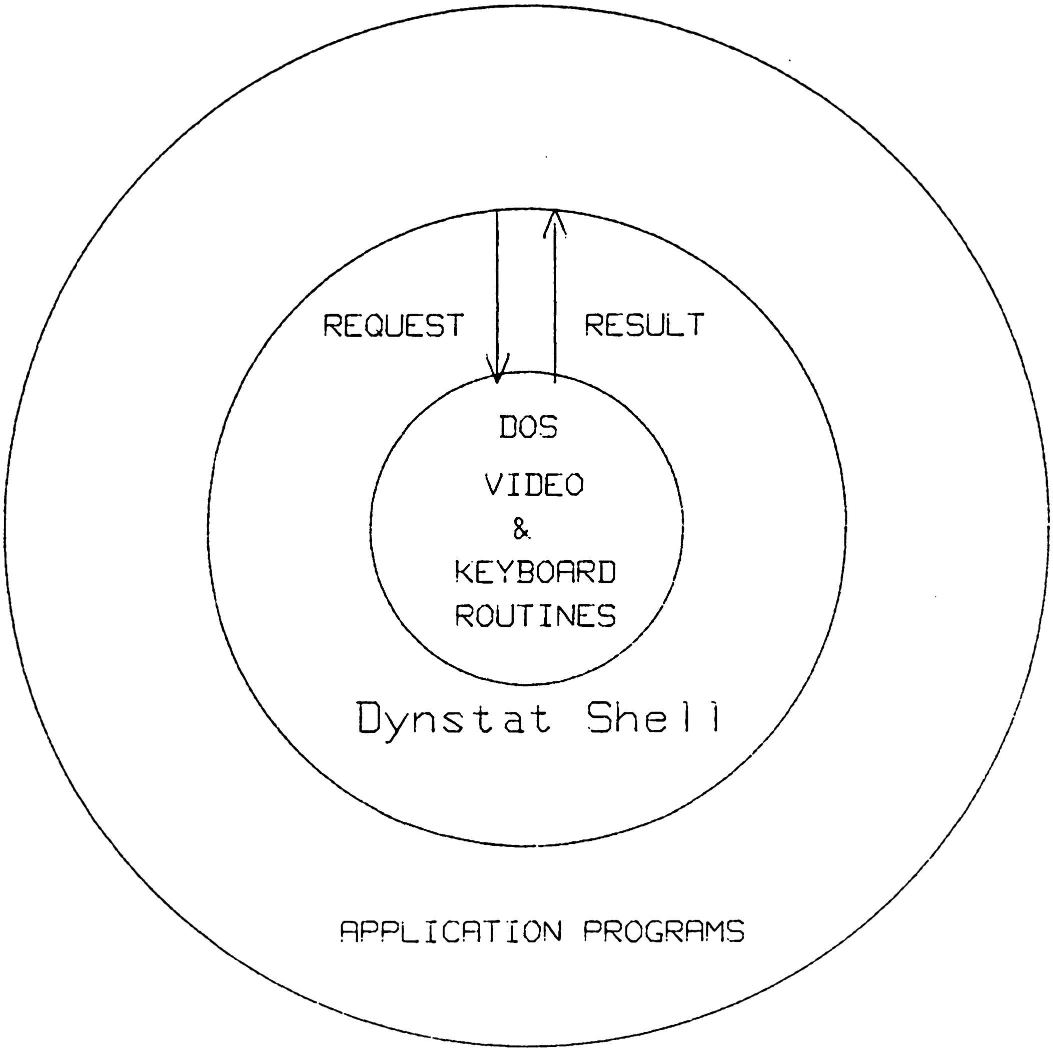


Figure 2. Program Shell

2.0 SYSTEM BASICS

2.1.1 INTRODUCTION

The first step in developing a system such as DYNSTAT should be an identification of the environment. This refers to typical applications for such a design. Further, the needs of the user should be of paramount importance. In the process of defining a specification, similar utilities were compared and special features were defined to the advantage of the user. In this chapter, an overview of some of the criteria used will be presented.

The main sections covered are:

- The Votrax System
- Why Assembly language?
- Functions Provided

2.1.2 THE VOTRAX FEATURES

The information provided in this section will cover only those aspects of the Votrax Speech System that are relevant to the software developed. Additional information can be found in the literature [1].

The Votrax voice unit or the Personnel Speech System (PSS) is the unit that was used to develop, test and run the developed software. Henceforth, the Votrax voice unit will be referred to as the PSS. This unit is very versatile, providing both the programmer and the user with a great degree of ease and flexibility. Figure 3 on page 12 and Figure 4 on page 13 show the front and rear panels of the PSS.

2.1.2.1 Interface Details

As can be seen from the rear panel, the PSS provides both a parallel and a serial interface. The program uses the serial interface. Figure 5 on page 14 Shows the system description. This choice was made due to the fact that the IBM PC printer connects to the parallel port. Further, two RS-232 serial ports were added on to the PC, giving a minimum configuration of one parallel printer, the Votrax connected to a serial port and a modem connection on the other serial port. The transmission over the PSS cable is 9600 baud with a XON/XOFF protocol in software to provide the necessary handshaking with the PSS. The baud rate, number of bits used and the transmission port can be set either through the switches on the rear panel of the PSS or through software. The switches were used, thereby ensuring a fixed setup. A special cable need to be used to connect the PSS to the PC. The details of

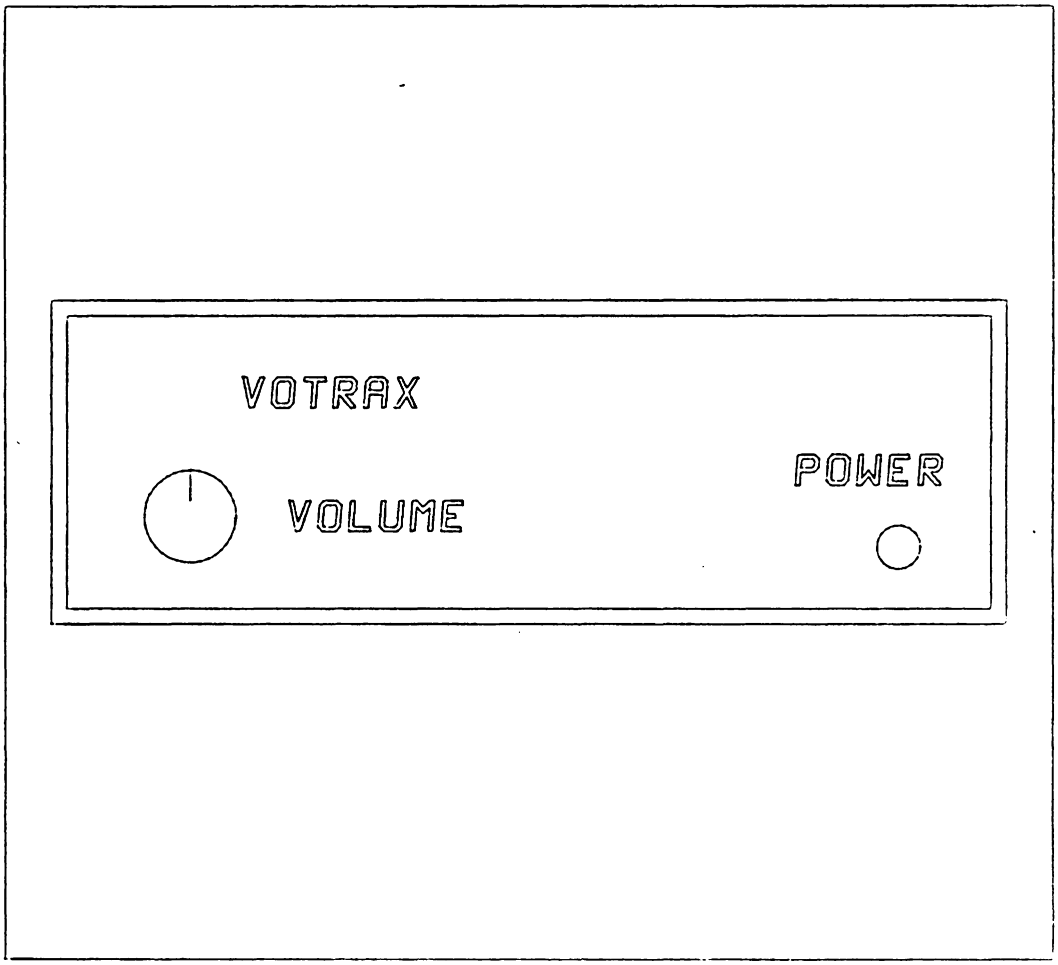


Figure 3. Votrax front panel view.

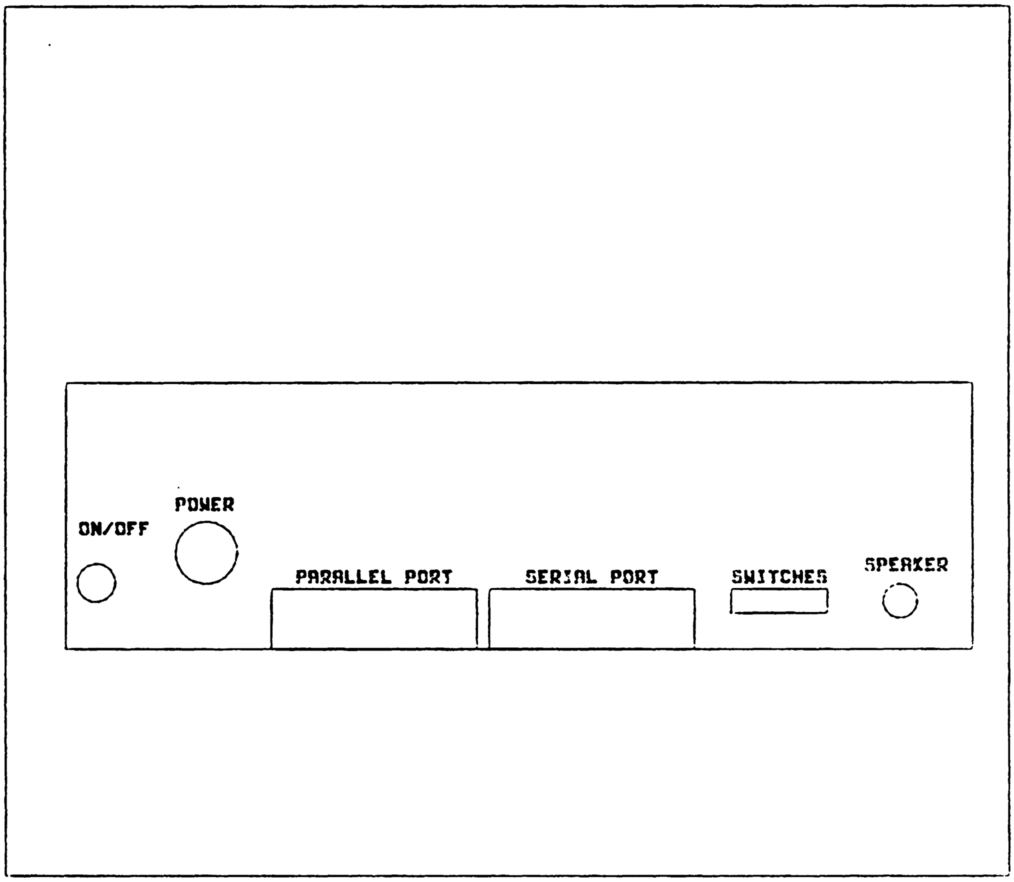


Figure 4. Votrax rear panel view.

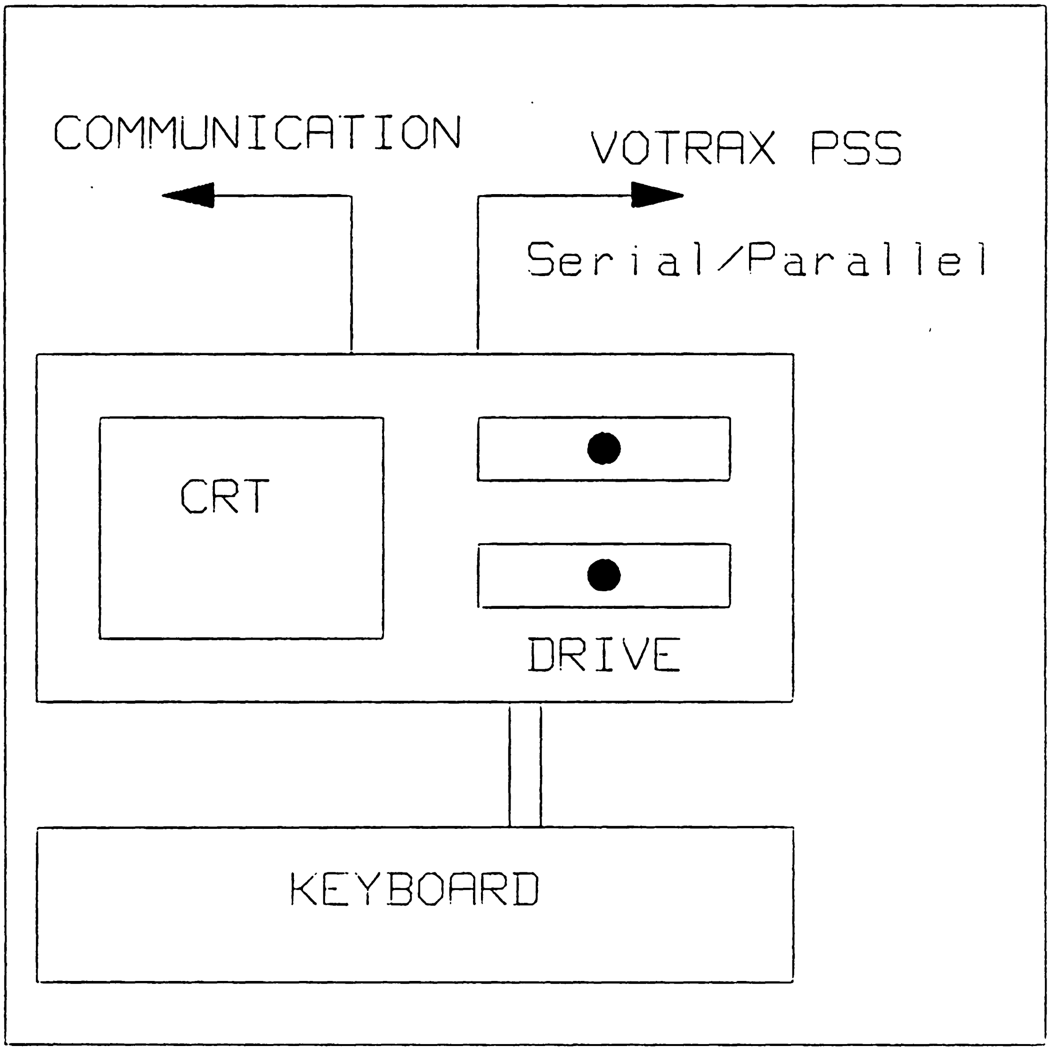


Figure 5. System Configuration

the PC termination and PSS termination of the cable are given in Appendix [A]

2.1.2.2 Data Protocol

The PSS receives data directly from Dynstat. Dynstat need not perform any translation or phonetic conversion. Standard ASCII input to the PSS produces the articulate voice output. The PSS contains a Text-To-Speech processor that translates standard English text to the phonetic code required by the synthesizer in a manner appropriate for the articulate pronunciation of the text. This frees the host computer for other operations. The translated phonetic components are used by the speech chip to generate the corresponding sounds. The speech processor uses pronunciation rules to facilitate text translation. This approach provides an unlimited vocabulary as opposed to a standard dictionary look up approach.

2.1.2.3 Data Types

The types of data that are provided to the PSS are:

- Speech Data (Text Only)
- Inflection Control
- Rate Control

To produce an audio output, Dynstat should provide both the output data (ASCII) and a terminator. The terminator is a carriage return. A terminator code is required by the PSS to signal an end of message condition. On receipt of the terminator, the PSS processes data in the buffer. The corresponding text to phonetic conversion is then created and an audio output is produced. This feature is used to control the speech of words or letters by the program. That is, characters preceding the return code are pronounced as such. Hence, Dynstat may selectively transmit return codes to provide either a word or a letter environment. For example,

- To speak 'A' ----> 41,0D. (All characters in Hex)
- For 'AND'   ----> 41,4E,44,0D.

2.1.2.4 Initialization

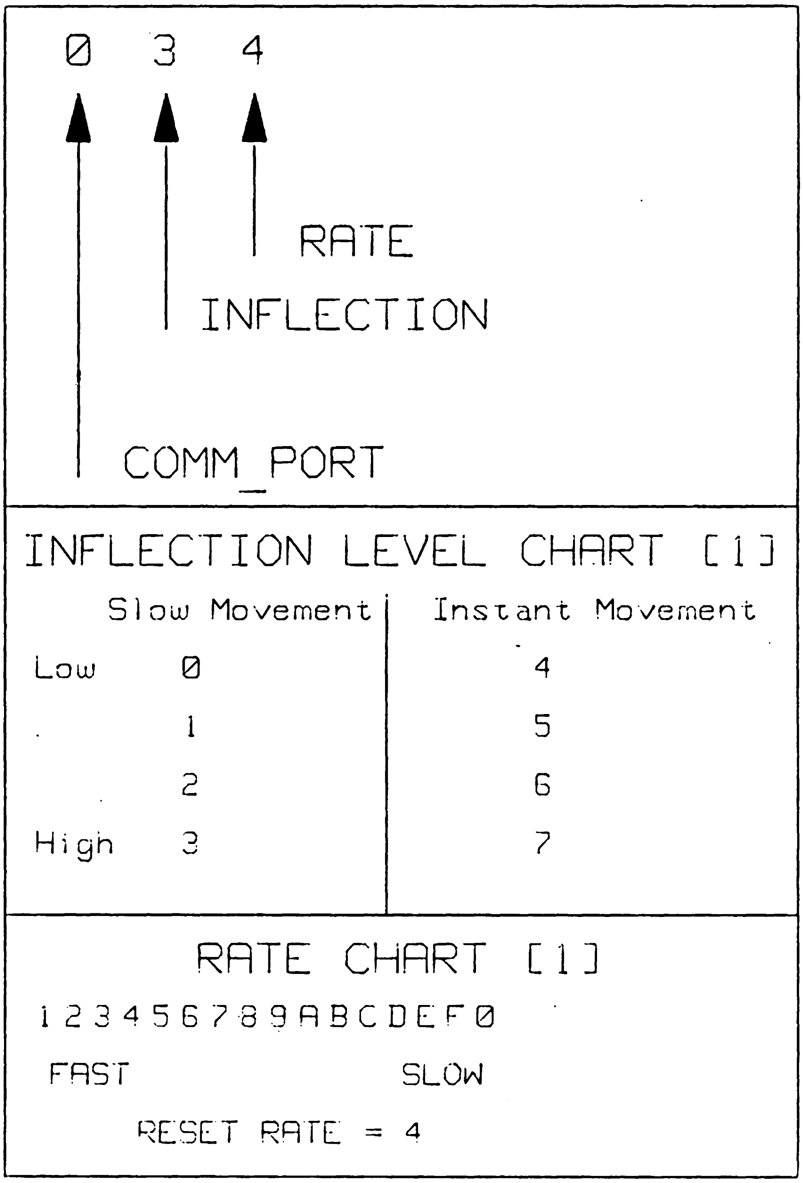
During initialization, the speech characteristic of the PSS is set to a default value. The default parameters are:

- Speech Rate
- Inflection Level

These control features continue till changed or reset. These features can be changed in the Static mode, using the Voice change command. The Voice change command supports two modifiers, namely the voice set modes and the quit speech mode. In the voice set mode, the rate and inflection of speech are changed in an interactive manner by the user. The keyboard is the input medium and the voice output serves as the final output. Default values are provided in a special file called "DEFAULT.VAL", and the system boots with these values. Figure 6 on page 18 shows the default table. Three numerical entries are found in this file. The first byte corresponds to the Comm Port being used. This is set to Comm 1. To use a Comm 2 port, this byte should be changed to a 1 etc.. Changes should be made using an editor prior to boot up with the program disk. The remaining bytes are for rate and inflection respectively. The limits for rate and inflection are shown in the default table figure. These values are updated to the new settings if changes are made and a request to save is made during initialization.

The interactive voice set initialization feature allows changes to recorded back to the disk. Changes may also be made by invoking the static mode, but note that these changes are not stored in the disk. This allows the user to set the voice to an acceptable articulate output. The quit speech feature is used extensively in the program to purge the current buffer of the PSS.

The control parameters for the PSS are :



INFLECTION LEVEL CHART [1]

	Slow Movement	Instant Movement
Low	0	4
	1	5
	2	6
High	3	7

RATE CHART [1]

1 2 3 4 5 6 7 8 9 A B C D E F 0

FAST SLOW

RESET RATE = 4

Figure 6. DEFAULT.VAL File Structure.

- INFLECTION CONTROL. @i, where i=0-7.
- RATE CONTROL. @Rr, where r=0-F.
- QUIT CONTROL. [Esc]Q.

A short discussion of the XON/XOFF protocol follows. Whenever the input buffer of the PSS comes within 30 bytes of being full, a control character (XOFF) is sent by the PSS system to the host computer. Upon receipt of this character, the host computer is to cease transmission until a XON character is sent by the PSS. A XON is sent by the PSS when the buffer returns to only 50 bytes being used. During normal operation, receipt of the XOFF character is signalled by a beep to the user. The XON/XOFF sequence can be simulated through Control S and Control Q codes.

2.1.3 WHY ASSEMBLY LANGUAGE?

Initially, it was decided to implement the user program in the 'C' programming language using the MICROSOFT C Compiler. Dynstat serves as a software shell between DOS and other application programs. Figure 2 on page 9 depicts this configuration. Access to the DOS keyboard or Video routines is through Dynstat. This approach is justified as follows:

2.1.3.1 Program Concepts

Upon closer analysis of the needed functions, it was recognized that the keyboard interrupt on the system BIOS had to be trapped. That is, the requirement of having the keyboard initiate software functions dictates this action. Characters from the keyboard are tested for Dynstat functions. Upon receipt of a key closure from the PC keyboard, the program would have to vector to the Dynstat service routine which would then pass the code to DOS, if necessary. Also, certain key combinations used to invoke the 'STATIC MODE' would require the routine to wait for further keyboard commands and not exit the keyboard service routine. To implement these functions in 'C', it was found that 'C' used the DOS function call INT 21H for keyboard requests. Since the program is logically already in the service routine, this created the need for the interrupt being recursive. Interrupt 21H on the PC DOS is not recursive.

To overcome this problem, all the modules had to be written in Assembly. The MICROSOFT ASSEMBLER for the 8086/8088, Version 3.0 [2] was used. Further, use of assembly also provided a great degree of flexibility in overcoming other difficulties encountered in the development process.

2.1.4 FUNCTIONS PROVIDED

The particular functions will be considered in this section. Detailed description regarding syntax can be found in the User's Manual, Appendix [B]. The program supports two modes:

- The DYNAMIC MODE.
- The STATIC MODE.

These can be considered to be "ON LINE" and "OFF LINE" approaches.

2.1.4.1 The Static Mode

The review or static mode provides an off line review feature. A provision exists to freeze the screen and examine every character. With this combination, the user may now exploit almost all of the available DOS programs. Use of the keyboard to control the program simplifies matters. Also, commands are invoked on single entries, which reduces the overhead on the user.

The Static mode serves more as as screen reader. Here, the entire screen of 25 lines serves as the current page. No interaction with DOS is possible or necessary in this mode. Now, to easily move through a page of information, some versatile functions have been defined. Lines can be read either as a range of lines or as an individual line, moving either in the upward or downward direction. Word/Letter forward or backward reading is also possible. String search operations are supported. Multiple occurrences of that string can be selectively scanned. Identification of the current row and column is provided. This enables the user to fix the point and make additions/corrections at that point at a latter stage. This would be ideal for users of word processing software. The user can also define 10 markers which would always locate to a predefined point. In fact, one such marker is defined by default to prefix to the point on the screen where the 'MORE..' would appear in the use of YTERM [3].

2.1.4.2 The Dynamic Mode

The Dynamic Mode is the interactive mode for the user. Here, all keyboard entries, DOS prompts and DOS responses are spoken back to the user depending on the format he has set in the Static mode. This would be ideally suited for a user of BASIC or FORTRAN or any such use requiring a keyboard input or a prompt from the user. Speech can be disabled for quiet operation. Further, provision also exists for the user to disable the console echo, that is, inhibit DOS responses from being spoken and enable the keyboard echo only to provide him with a feedback of key entry operations. Alternately, both the keyboard and console echoes can be disabled. Since carriage returns and DOS prompts are always spoken, the console echo may be disabled.

2.1.4.3 Advantages

The implications of such an approach are numerous. For example, there is no longer an imposition on the memory of the user. Access to each character on the screen is now provided. Consider the case of debugging a listing file. Any standard editor may be used to scroll through the file. For each page, the user may turn the echo off and set a marker to locate to the error count statement. There is no longer a necessity to mentally form an image of the entire screen. This should drastically reduce the time overhead on a visually impaired user. Features also exist to set certain basic modes of operation like :

- Voice Mode
- Letter/Word Mode
- All/Select Punctuations
- All/No Space Recognition
- Enable/Disable Echo Features (Dynamic Mode Only)

For a detailed summary of the program commands, refer to Appendix [B].

As part of the tests performed, many frequently used programs were run to assure compatibility. The resident software developed is totally user transparent. The subsequent chapters will provide more information on the total implementation and design features of the software system.

3.0 SOFTWARE ALGORITHMS

3.1.1 INTRODUCTION

The design specifications required an algorithm that interacts with basic DOS functions. Primary interest centers around the Keyboard and Video interrupts. These form part of the Input-Output processing used by DOS. The approach of trapping video and keyboard interrupts provides data used by DOS. Data obtained thus is used by Dynstat to activate suitable procedures.

In this chapter, interfacing with DOS is considered. The chapter is organized in the following sections:

- The Main Program.
- Keyboard Interrupt.
- Video Interrupt.
- The COM file Setup.

Figure 7 on page 27 describes the interaction between the three main modules. The function of each module is outlined in this chart. Each module has a specific function as shown and other modules use these results to perform their functions. Passing of parameters is done in the common data area.

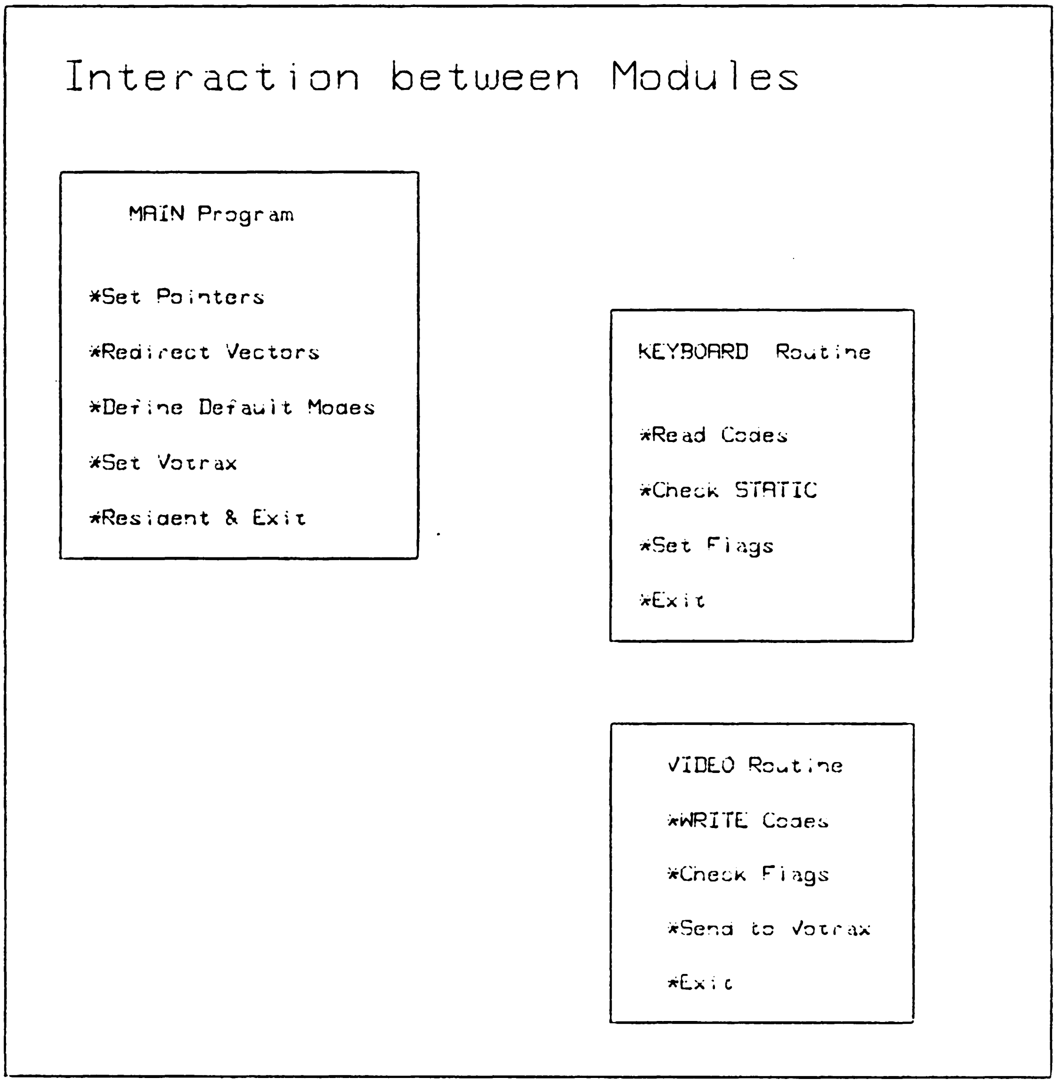


Figure 7. Program Structure.

3.1.2 THE MAIN PROGRAM

The Main program is the area where all pointers are initialized to their default values and interrupt reallocation is done. The program is so structured that the main program upon termination, returns control to DOS with Dynstat interrupt handlers resident and transparent to DOS. The entire program resides in one segment, using 10K of the system memory. Figure 8 on page 28 describes the program flow. The flow may be summarized as :

- Check Int 10H for valid addresses.
- Change BIOS Keyboard Vector.
- Initialize PSS.
- Change BIOS Video Vector.
- Terminate and Stay Resident.

The following sections discuss these operations in detail.

3.1.2.1 Protection

The program disk is configured such that the PC boots with Dynstat. As with any resident program, precautions must be taken to ensure that the program may not be executed again without re-booting. When a portion of code is to be made resident, it is necessary to pass the next free address where

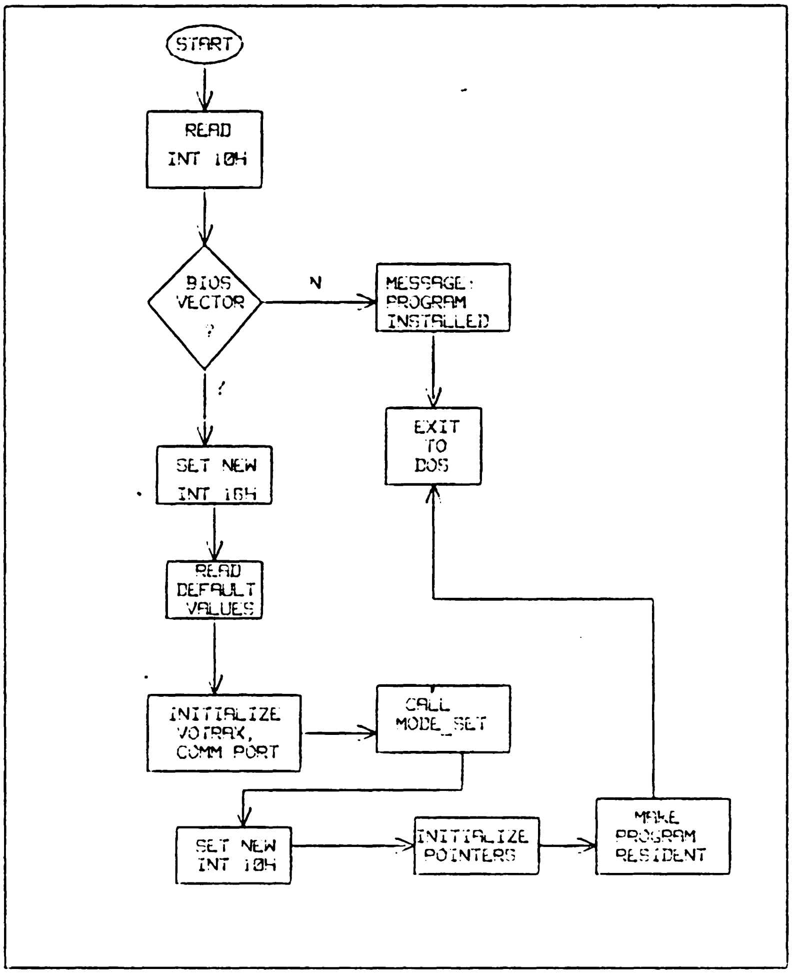


Figure 8. The Main Program.

other programs can be loaded to DOS. This is usually done as the last program statement. Once this statement has been executed, DOS "reserves" this code area and treats it as an extension of other DOS programs that are resident.

The PRINT command is an example of a resident program. The DOS program disk is accessed for the first PRINT command only. Subsequent PRINT commands default to the pre-loaded code. Now, if there were no software prevention, a re-run will result in the program being loaded at the next free address space and occupy system memory. More catastrophic would be the fact that interrupt reallocation would be done again, leading to unpredictable results. This is so, because the interrupt pointers now have Dynstat interrupt handler addresses in them. These should not be treated as the original BIOS vectors. If this were to happen, Dynstat addresses would be saved as DOS addresses and hence, calls to DOS vectors would result program transfer to handlers loaded earlier. That is, the keyboard interrupt would call itself again! This would lead to unpredictable results, possibly causing the system to "hang up".

The main program hence performs a software check on the vector addresses to prevent multiple executions. These addresses are compared with the original addresses for program validity. Multiple executions now result in a normal exit with no changes.

3.1.2.2 Initialization

Initialization of program pointers is the first area of interest. In this section of the program, buffers and pointers are set. These pointers and buffers are accessible to all routines and are used extensively for parameter passing. Some flags are set to "Active" values. Flags that control the word mode, punctuations and console echo are examples.

1. WORD Mode ----> Default.
2. PUNCTUATIONS ----> Select Punctuations Only.
3. CONSOLE ECHO ----> ON.

3.1.2.3 Vector Mapping

The vectors being reassigned are the Video and Keyboard interrupts. The Serial interrupt is used extensively in the program to communicate with the PSS, but the original DOS BIOS routine suits the purpose and no modification is necessary. The Video and Keyboard vectors have to be routed through Dynstat interrupt service routines since DOS uses these vectors to handle all information flow. When reallocation is done, the original addresses are stored in pointers to facilitate access to these routines. The DOS function call approach is used to change interrupt tables [4]. Figure 9 on page 32 gives the original vector map and Figure 10 on page 33 the modified map for the interrupts. Once this has been set, any reference to these interrupts will vector to Dynstat service routines [5].

3.1.2.4 PSS Setup

Information relating to the port to be used for the PSS and default voice parameters are stored in the DEFAULT.VAL file on the program disk. The next operation is to read these parameters from the disk and store them in pointers. These values are now used to establish communication with the PSS. The next step is the interactive voice set feature. Test messages are spoken and the user is prompted to change the default values for the PSS. Additional information can be found in the User's Manual, but a summary of the prompt messages are provided here.

- 'The quick brown Fox jumps over the lazy dog'
- 'Do you wish to change default values'
- 'Enter R for Rate, I for Inflection and Return to Exit'
- 'Do you wish to save these settings'

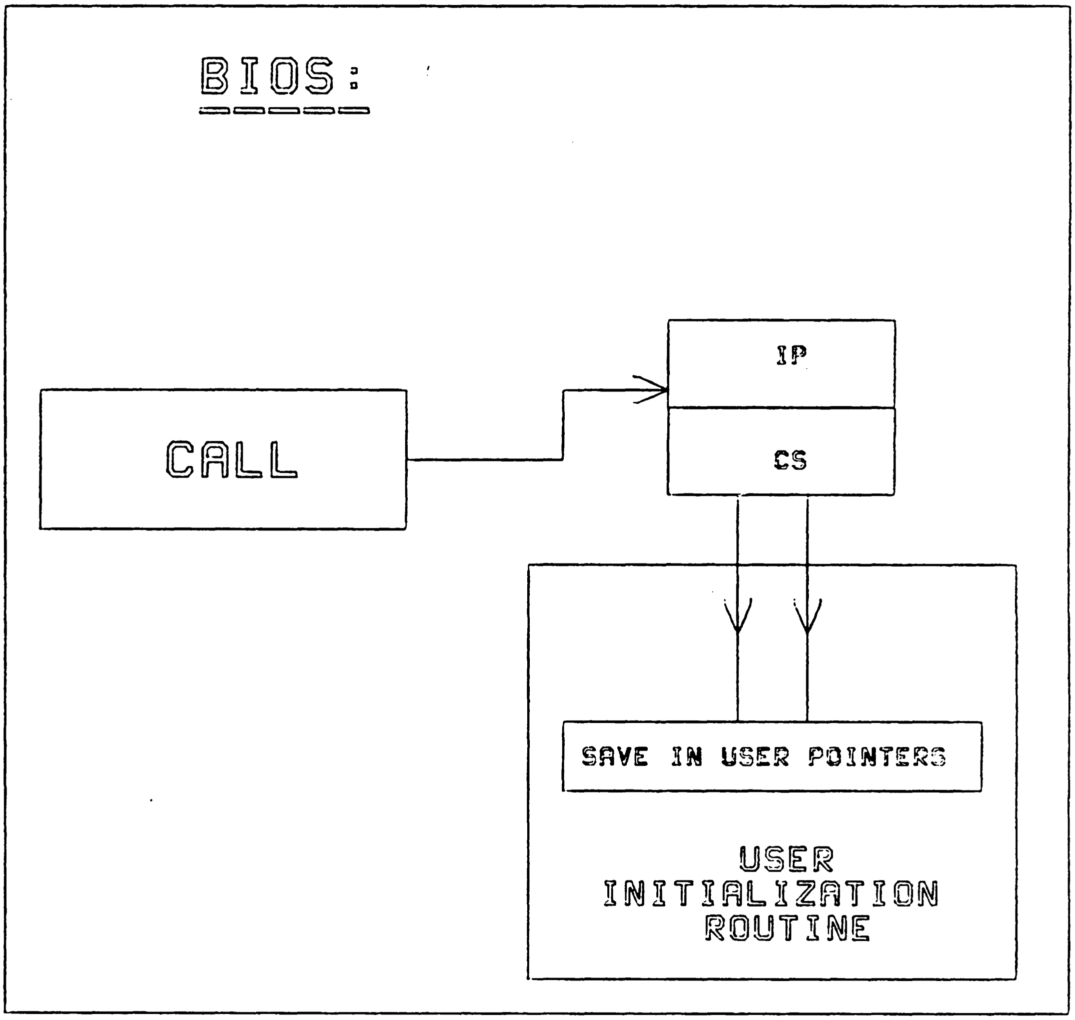


Figure 9. BIOS Interrupts.

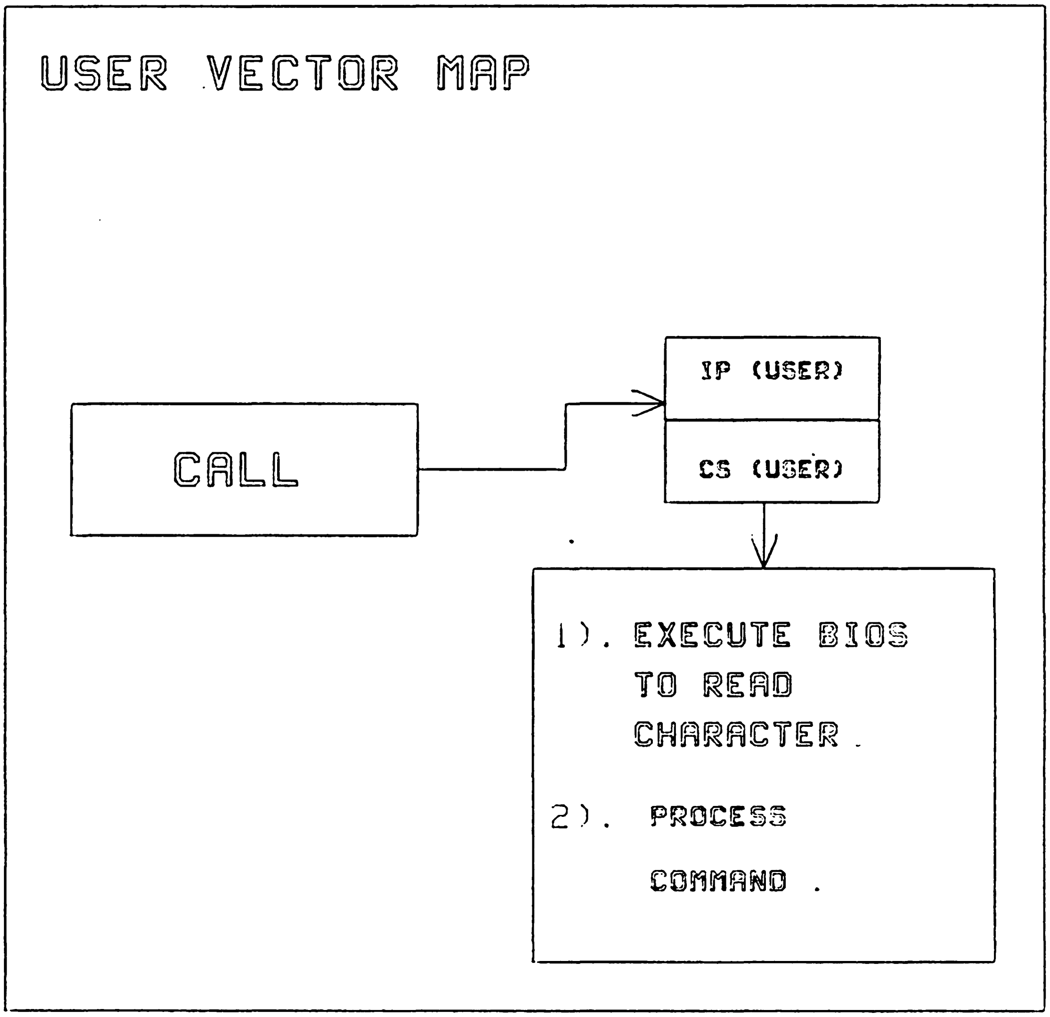


Figure 10. Modified Interrupt Structure.

The responses to these queries would be either Y for yes or N for no. Entries are not case sensitive. To change the voice level, the "Cursor Up" and "Cursor Down" keys are used as the input media and each key stroke sends a new value to the PSS. A message, "The quick brown fox jumps over the lazy dog" is spoken with the new settings. On exit, values may be stored back to the disk. An interactive menu feature prompts the user for appropriate action.

3.1.2.5 Resident Program Concepts

To make the program resident the DOS function call approach is used . Resident programs should satisfy certain rules. These rules are discussed in more detail in the last section of this chapter. Basically, the next free address where other programs can be loaded is passed to DOS. Exiting programs with the next free address in the DOS loading structure ensures that Dynstat interrupt handlers may not be overloaded by other application programs.

3.1.3 KEYBOARD INTERRUPT

Figure 11 on page 36 describes the interrupt routine.

To summarize the operations performed, consider the following list:

- Check for Keyboard Read.
- IF FALSE, perform function, return to DOS.
- IF TRUE, get code from keyboard buffer.
- Check for Static mode.
- Perform Static operations.
- ELSE, check for other keys and Exit.

DOS provides two keyboard interrupts. Namely,

- Int 09H.
- Int 16H.

Int 09H is the hardware address to which the program will vector on a key stroke. The Int 10H routine is not of much use since only scan codes are passed back. The Int 16H interrupt is the one that all application programs and DOS use to test for a key closure. Int 16H passes the ASCII codes of the keys back to Dynstat. Use of Int 16H provides a better

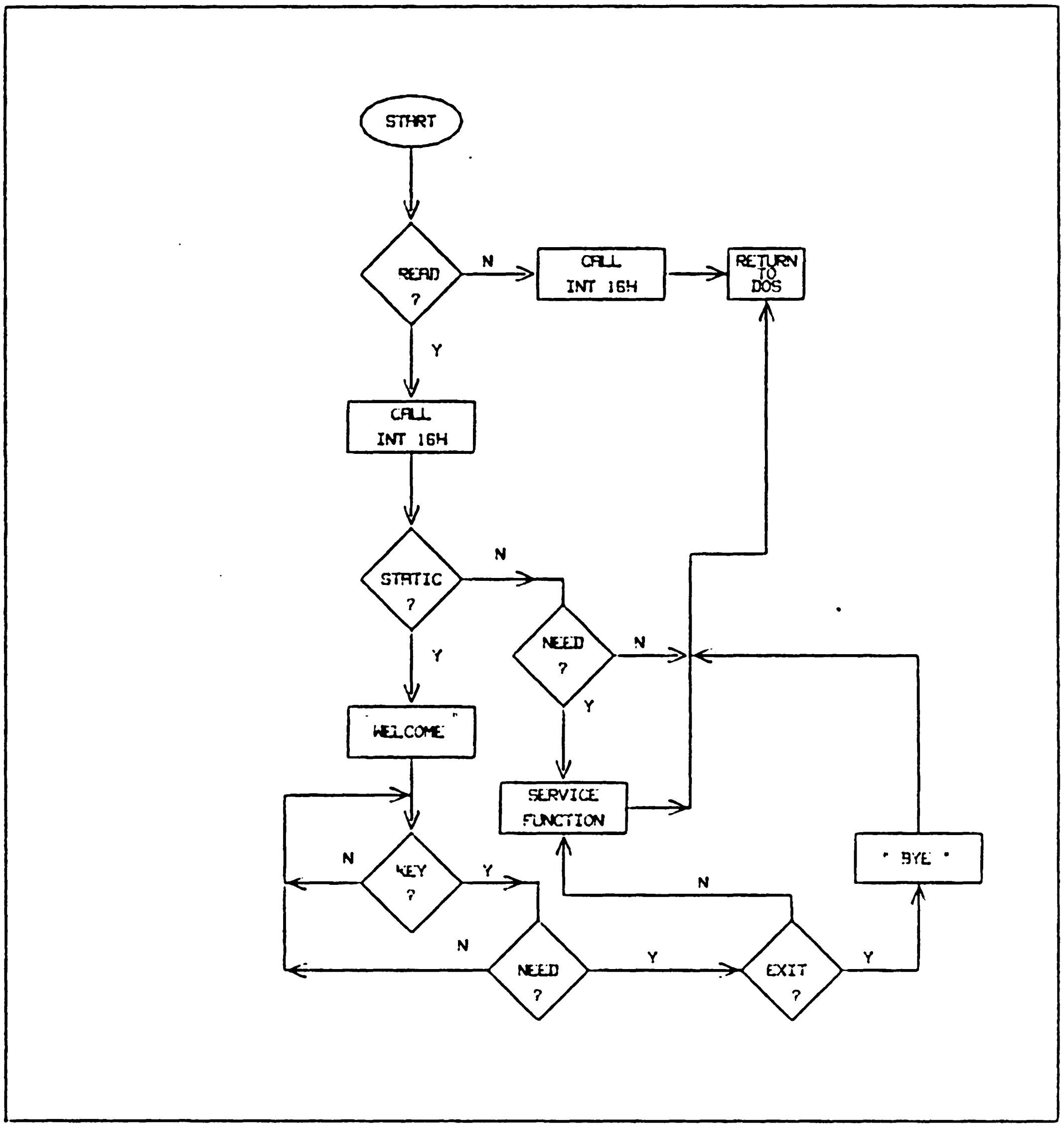


Figure 11. Keyboard Interrupt flow chart.

interface to other programs. In trapping Int 16H, characterization is to be made between read operations and all other functions. Dynstat needs codes for entries made from the keyboard. The approach here is to check codes from the keyboard and test them for Dynstat requirements. The definitions of the STATIC and DYNAMIC modes dictate that a test be performed on keyboard codes. All commands in Dynstat are invoked through keyboard operations. Hence, access to keyboard codes will help differentiate between DOS commands and Dynstat commands.

3.1.3.1 Command Structure

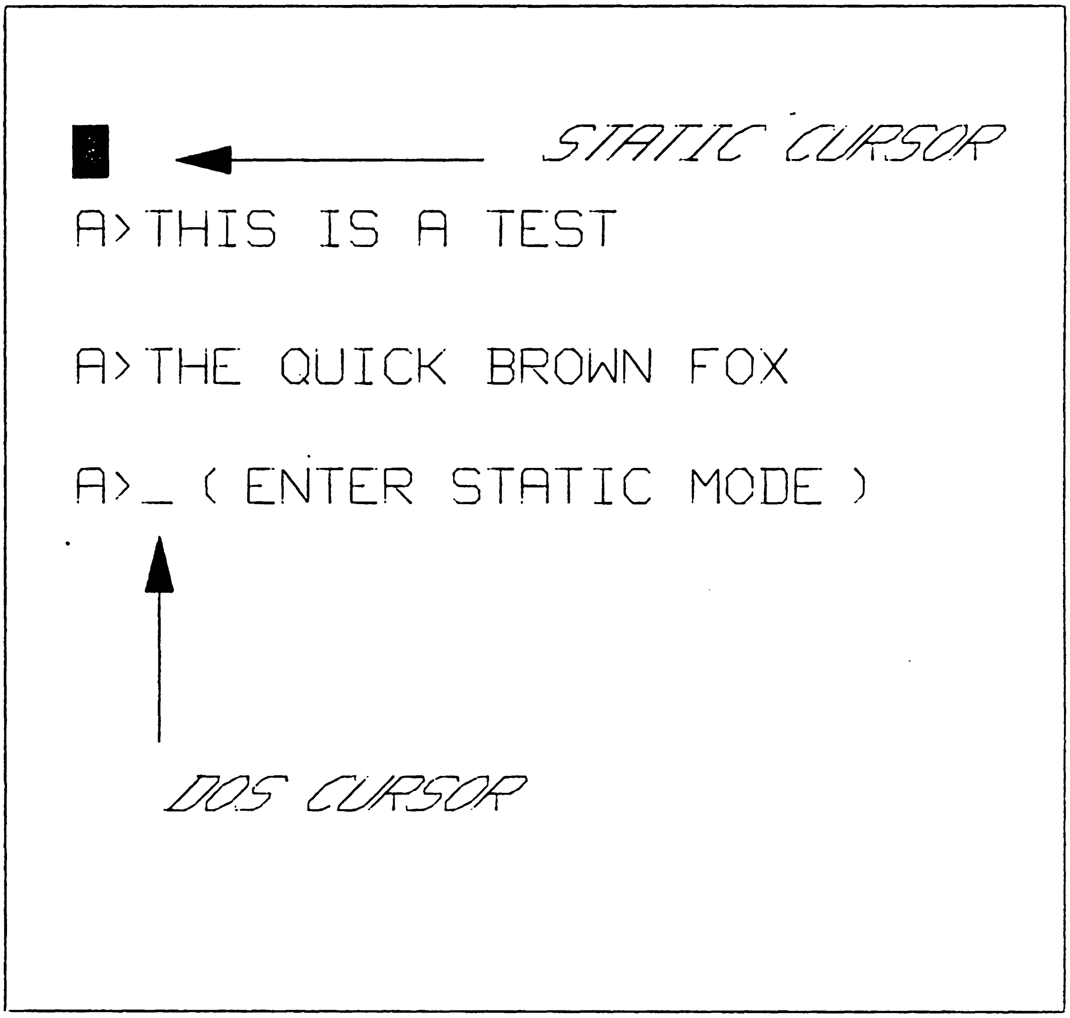
In the STATIC mode, all commands have a two level processing structure. In the first level, an identifier is set and further subcommands lead to the second level for execution. For example, "A" would identify a call to fix a marker and function keys (F1-F10) would be subcommands. Two level processing is obtained in the following manner:

If the code had been "A", a flag is set and control passes to DOS. The next key stroke would call the static procedure and is identified as a subcommand. If the key had been a function key (F1-F10), the position of the cursor is saved in the buffer address pertaining to that key. Any invalid key stroke would override the subcommand. Invalid keystrokes help in not having to execute a command if entry to it (first level) had been a typographic error.

3.1.3.2 Static Operations

When the STATIC mode code is recognized by the keyboard routine, the current position (row/column) of the cursor is saved. The cursor provided by DOS will be referred to as the DOS cursor. The movement of the cursor in the STATIC mode has no relation to the DOS cursor. The current line in the STATIC mode is the line where the cursor is. Movement of the cursor updates a software counter. The last static cursor position is saved on exit and the next entry to the STATIC mode will default to the last saved position. That is, the cursor is set to the point where the user last left off. Continuity for operations is provided automatically when the user has to toggle between modes. Figure 12 on page 39 describes the two cursors.

The Static mode may be used to check spellings, syntax,... Once an error has been found, the user may wish to return to the word processor to make the change. After this, entry to the Static mode will be at the location where an exit was requested. Elimination of the need to remember the line or column number where the next correction is to be made is a distinct advantage. On exit, the DOS cursor is restored and screen alignment is not affected.



```
A>THIS IS A TEST

A>THE QUICK BROWN FOX

A>_ ( ENTER STATIC MODE )
```

Figure 12. DOS & Static Cursors.

The user could also set a marker to default to a particular point on the screen. Commands are not case sensitive. All commands are echoed back to the user and provision exists for silent mode operation to rapidly reach the point of interest. A Help menu is provided to inform the user of current settings. Exit to DOS results in these settings being saved and used by the DYNAMIC Mode.

3.1.4 VIDEO INTERRUPT

The Video interrupt is used by DOS to control all screen operations. Any call for video functions by DOS will now pass through the Dynstat video routine. As in the case of the keyboard interrupt, certain checks are to made to ascertain if the result need be passed on to the PSS. Figure 13 on page 42 shows the program flow.

3.1.4.1 Control Criteria

Control will pass to the Dynstat program only if the following conditions are satisfied:

- Quit speech flag not active
- STATIC flag not active
- Console echo not inhibited
- The operation is a character write

The first step is to determine if any of these requisites are met. If not, console operations may be inhibited. That is, the character need not be read and sent to the PSS. The program then returns control back to DOS. One of the options in the STATIC routines is to provide a console echo. If the console echo is disabled, the original BIOS video routines are vectored to, and control is passed back to DOS. The next

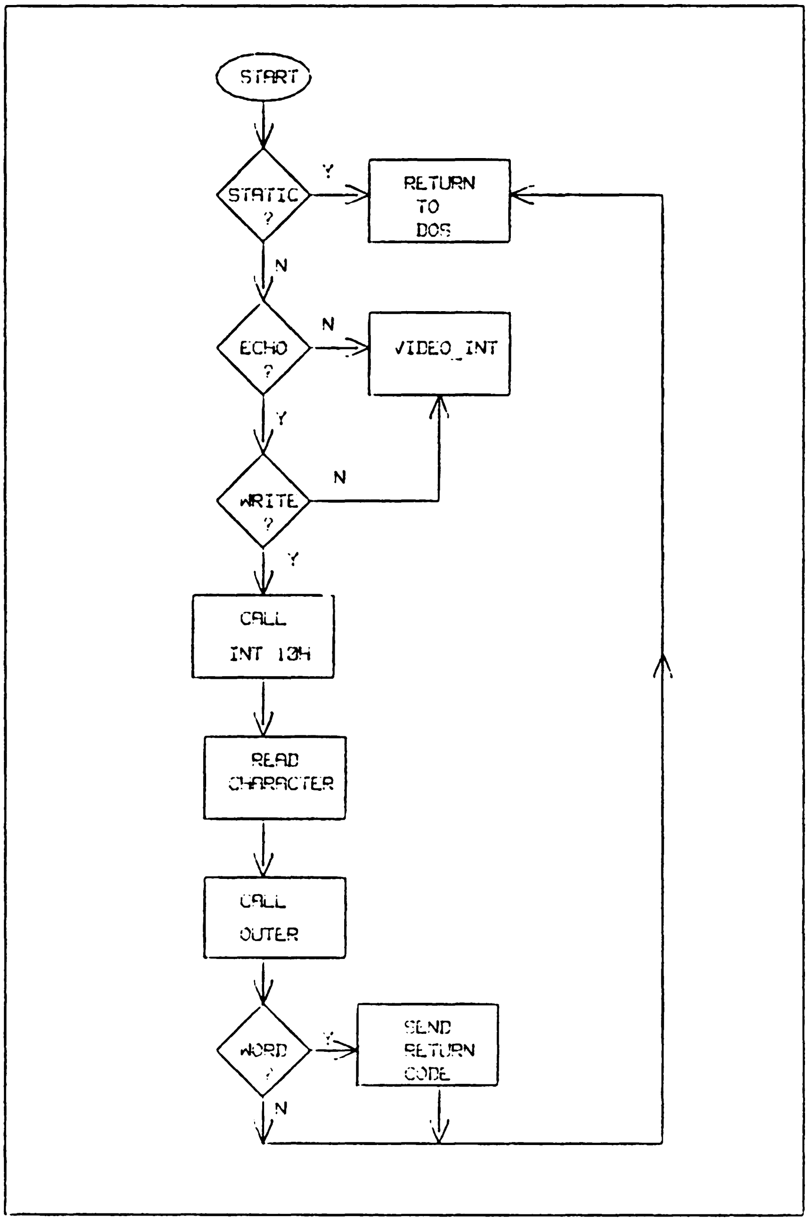


Figure 13. Video interrupt.

check is to ensure that the call is for write operations only. All other calls pass on to the original BIOS and terminate.

If all these conditions had been satisfied, the flow is through the Dynstat video interrupt handler. A call is made to the original BIOS routine to process the function (the BIOS addresses are stored in pointers). Then, the character written on the screen is read and passed to the transmit routine. Now, the PSS needs a return code (0DH) to be sent after each character to be spoken. This serves to distinguish between words and letters. In the letter mode, a return is sent immediately. Returns for word mode settings are deferred until a space or a manual end of line condition (use of the Return key) is encountered. Control of the environment is now passed back to DOS, with all registers being restored back to their entry values.

3.1.4.2 Operation

The video interrupt is used extensively by other routines that read the current line, write/read characters on the screen and set the cursor position.

The transmit routine tests flags to control speech. The flags tested are word mode, letter mode, punctuations etc... . The transmit routine also checks the buffer status on the PSS to avoid overrun errors on the PSS. The software handshaking protocol is implemented in this routine. The passing of parameters between all these routines is through flags. Each routine then, sets or resets these flags which are sampled by other routines for synchronized operation. The other factor that needs mention at this time is the flag set by a manual carriage return. This flag helps distinguish between end of line codes sent by DOS and the user.

3.1.5 THE COM FILE SETUP

Machine language programs may exist in two forms, namely EXE and COM. These extensions are created by the assembler in use. EXE files are the normal outputs of assemblers. COM files are more useful than EXE files. COM files are easier to create, debug and require less storage space in memory. COM files also execute faster than EXE files [6].

3.1.5.1 Required Conditions

To create COM files, certain rules are to be followed:

- The source code must be created without a Stack Segment.
- The Code Segment must begin at 100H.
- All Code, Procedures and Initialized Data should reside in one segment.

When the source program is created with these rules and assembled, the assembler reports one serious error,

" Warning : No Stack Segment ".

This error may be ignored as it was our intention to create a file with no Stack Segment anyway. To create the COM file, the EXE2BIN command will have to be used. This is a utility program that resides on the DOS diskette. EXE2BIN will create a file with a BIN extension. The BIN file should be renamed to a COM file. The EXE and OBJ files may be erased now.

But note that if the above rules cannot be met, an EXE file must be used.

4.0 SCREEN READER FUNCTIONS

4.1.1 INTRODUCTION

This chapter presents the main transmit routine, functions available in the Static Mode and the interaction of some of the functions between static and Dynamic Modes. Since the program serves as a shell between DOS and other application programs, the routines should provide results both to the application program and the user who needs an audio output.

The following sections are covered :

- The Main Transmit Routine.
- Static Commands.
- Mode Interaction.

4.1.2 THE MAIN TRANSMIT ROUTINE.

All communication with the PSS pass through this routine. Figure 14 on page 49 describes the logic. This routine uses the buffer on the PSS, thereby reducing system memory requirements. Use of the PSS buffer increases system throughput and results in a time efficient algorithm. The transmit feature classifies data to be sent to the PSS as fixed messages or as individual bytes of information. Figure 15 on page 50 illustrates this. Appendix [C] shows the ASCII character set supported by the PSS. Now, if any member of this set is sent to the PSS, followed by a carriage return code, the corresponding phonetic output is produced. Comparison of this set with the standard ASCII chart yields the unsupported codes which are to be handled by the routine. Most of the punctuations fall into this unsupported category. Also, codes such 'Escape', '!' and '@' cannot be sent to the PSS since they represent PSS command initiators Appendix [D]. Hence, an identification of such cases is done in the routine and messages defined during initialization are spoken. For example,

- ! would be spoken as 'Exclamation Mark'
- @ would be spoken as 'At'.

These messages are issued in a fixed format.

The use of fixed messages which are spoken independent of the mode currently active calls for the use of an exclusive routine. To illustrate, any fixed message is spoken in the word format, irrespective of the letter mode being active. The system returns to mode settings for other operations like console echo. Some keys are treated as special cases and are always spoken. Keys like 'Tab', 'Back Space' and 'Return' are

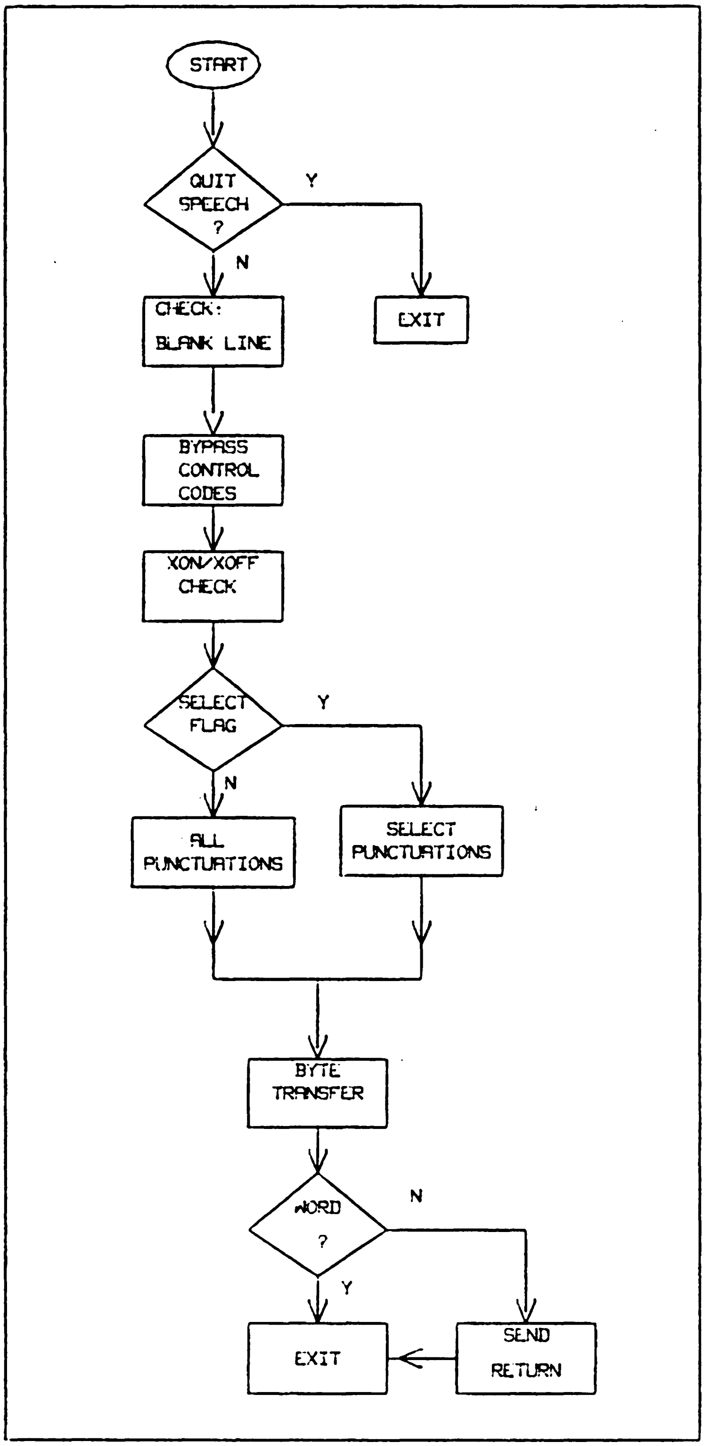


Figure 14. Transmit Routine.

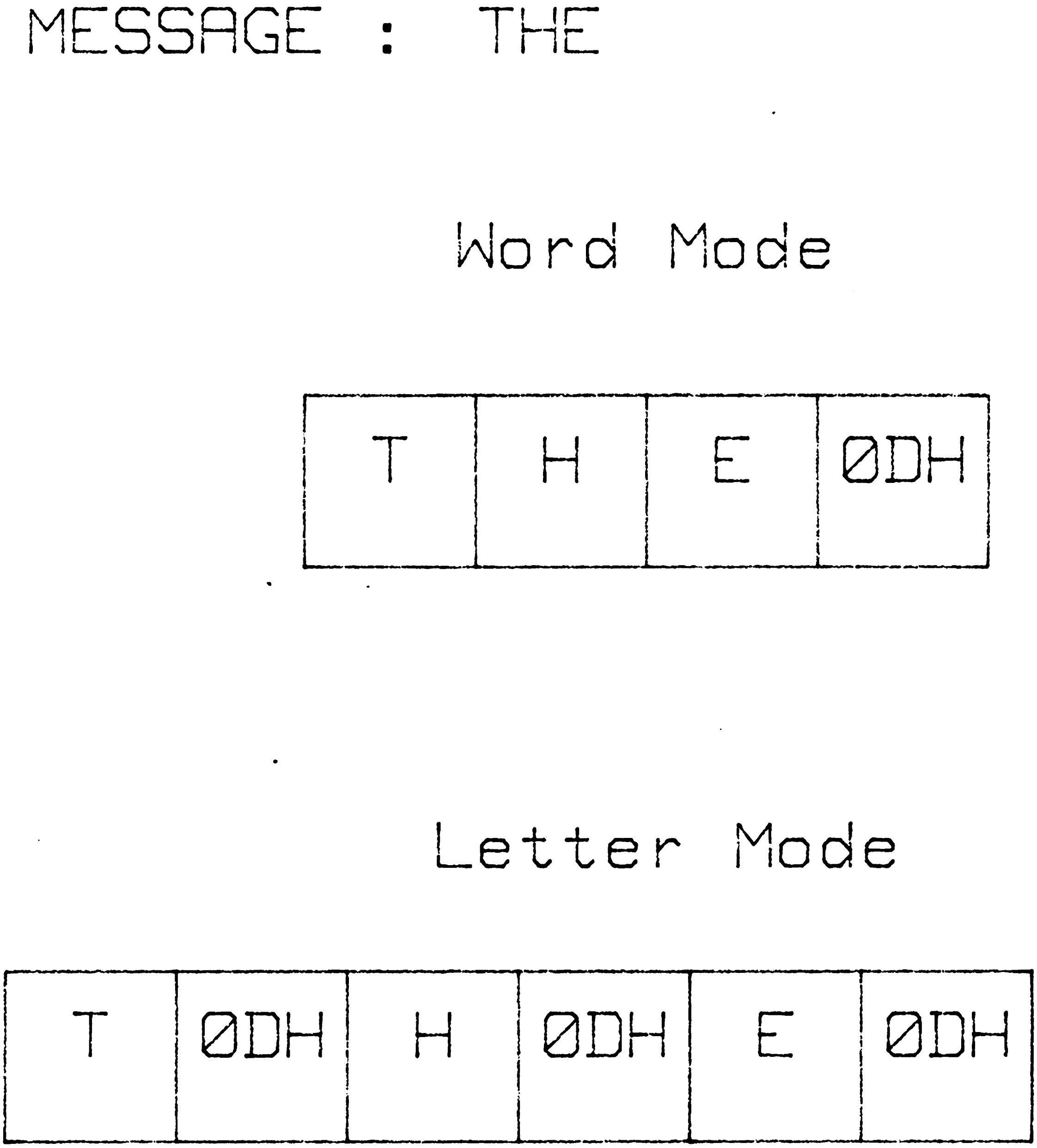


Figure 15. Message Identification.

examples. These exceptions are identified in the keyboard routine.

The routines discussed just process data and do not transmit code to the PSS. The message routines do not directly invoke the serial transmit DOS vector. Serial communication is done by a separate procedure. The PC supports 2 Serial Ports. The default port number (the port connected to the PSS) is read from the DEFAULT.VAL file and is used for transmission. The BIOS serial interrupt (Int 14H) is used to access the PSS.

Now, to route parameters to the appropriate procedure, a series of checks are carried out. The first check is on the quit speech flag. If an active condition is found, the transmit area is bypassed and no byte transfer to the PSS is done. Transmission of space codes to the PSS just produces a pause. To eliminate this delay, a check is made on the number of spaces present. If 80 spaces are found in a line, the message 'BLANK LINE' is spoken. To synchronize data transfer from the PC to the PSS, a XON/XOFF protocol is used. The PSS supports this protocol, which is implemented in software by Dynstat. The RXRDY status on the COM port will indicates presence of PSS data. If the code received is Control S, transmission freezes. The program polls the COM port for a Control Q character to resume transmission. This handshaking will produce a beep on the terminal for every Control S code received from the PSS.

4.1.3 STATIC COMMANDS

4.1.3.1 Introduction

An introduction to the static commands has been made in the earlier chapters. Now, this section presents a detailed description on the implementation of these commands. Though most of the information regarding syntax can be found in the User's Manual, a concise description will be given here to maintain continuity. Comparison between the modes will also be dealt with.

4.1.3.2 Invoking Static Functions

The Static mode may be invoked at any time. If invoked during a screen scroll, the screen freezes and passes control to the Static server routines. On exit from the Static mode, the DOS operations continue. This gives the user the advantage of working with the Dynamic mode and immediately shifting into the Static mode. As discussed earlier, the Static mode forms a separate shell by itself. That is, all input in the Static mode is checked for an exit code or other Static mode functions. Erroneous entries are ignored and the program waits for the next correct keystroke. Now, to invoke the Static mode,

- Depress the SHIFT and BACK SLASH (\) Keys simultaneously.
- The Prompt, " Enter Static Mode " will be spoken.
- Any keystroke will now be treated as a Static Command.
- " X " or " x ", will exit to DOS.

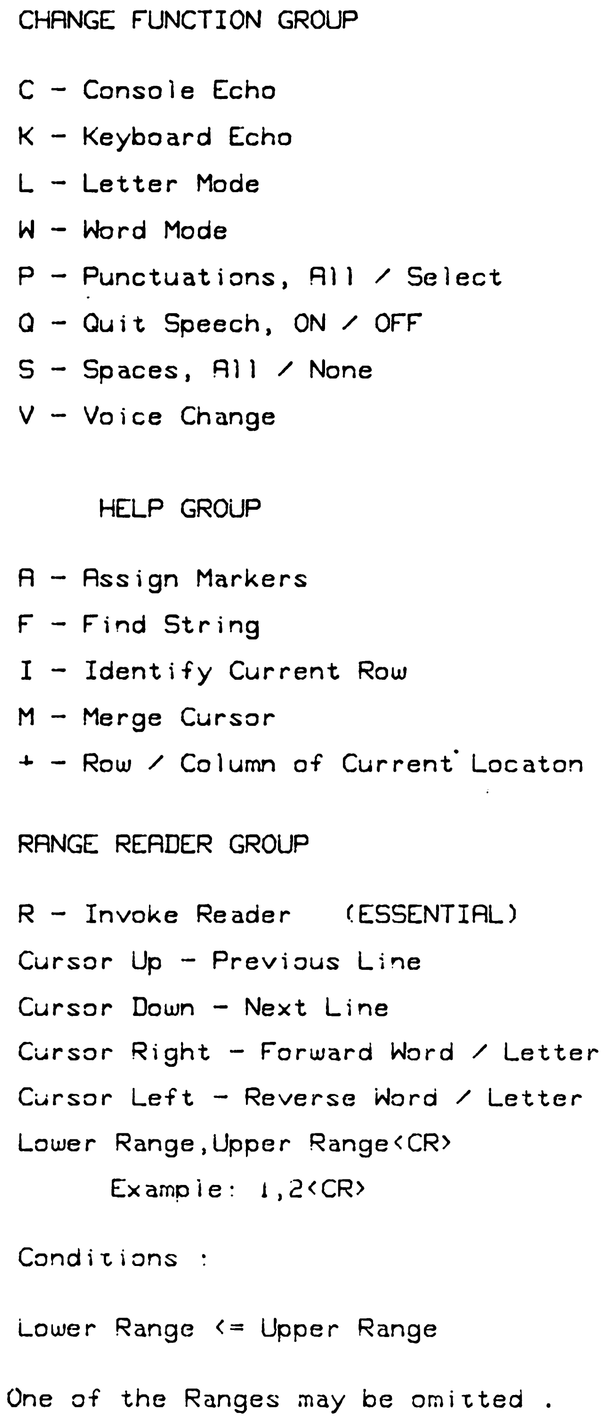
Figure 16 on page 55 pictures the command function chart. Commands may be considered to comprise of three logical function groups, namely:

- The Change Function Group,
- The Help Group,
- The Range Reader Group.

4.1.3.3 The Change Function Group

In this set, all commands affect the mode settings of the program. Hence the name, "Change Function". These commands control the entire modes of operation. Once set, their effects are immediate. These are the only commands that have any effect on the Dynamic mode of operation. A common functional diagram may be used to explain their implementation. Figure 17 on page 56 illustrates this.

In this chart, the first check is to identify the functional grouping of the command. If the test fails for the Change Function Group, the program branches to test for other valid commands. Else, the appropriate key is spoken. Next, the



CHANGE FUNCTION GROUP

C - Console Echo
K - Keyboard Echo
L - Letter Mode
W - Word Mode
P - Punctuations, All / Select
Q - Quit Speech, ON / OFF
S - Spaces, All / None
V - Voice Change

HELP GROUP

A - Assign Markers
F - Find String
I - Identify Current Row
M - Merge Cursor
+ - Row / Column of Current Locaton

RANGE READER GROUP

R - Invoke Reader (ESSENTIAL)
Cursor Up - Previous Line
Cursor Down - Next Line
Cursor Right - Forward Word / Letter
Cursor Left - Reverse Word / Letter
Lower Range,Upper Range<CR>

Example: 1,2<CR>

Conditions :

Lower Range <= Upper Range

One of the Ranges may be omitted .

Figure 16. Static Command Function Groups.

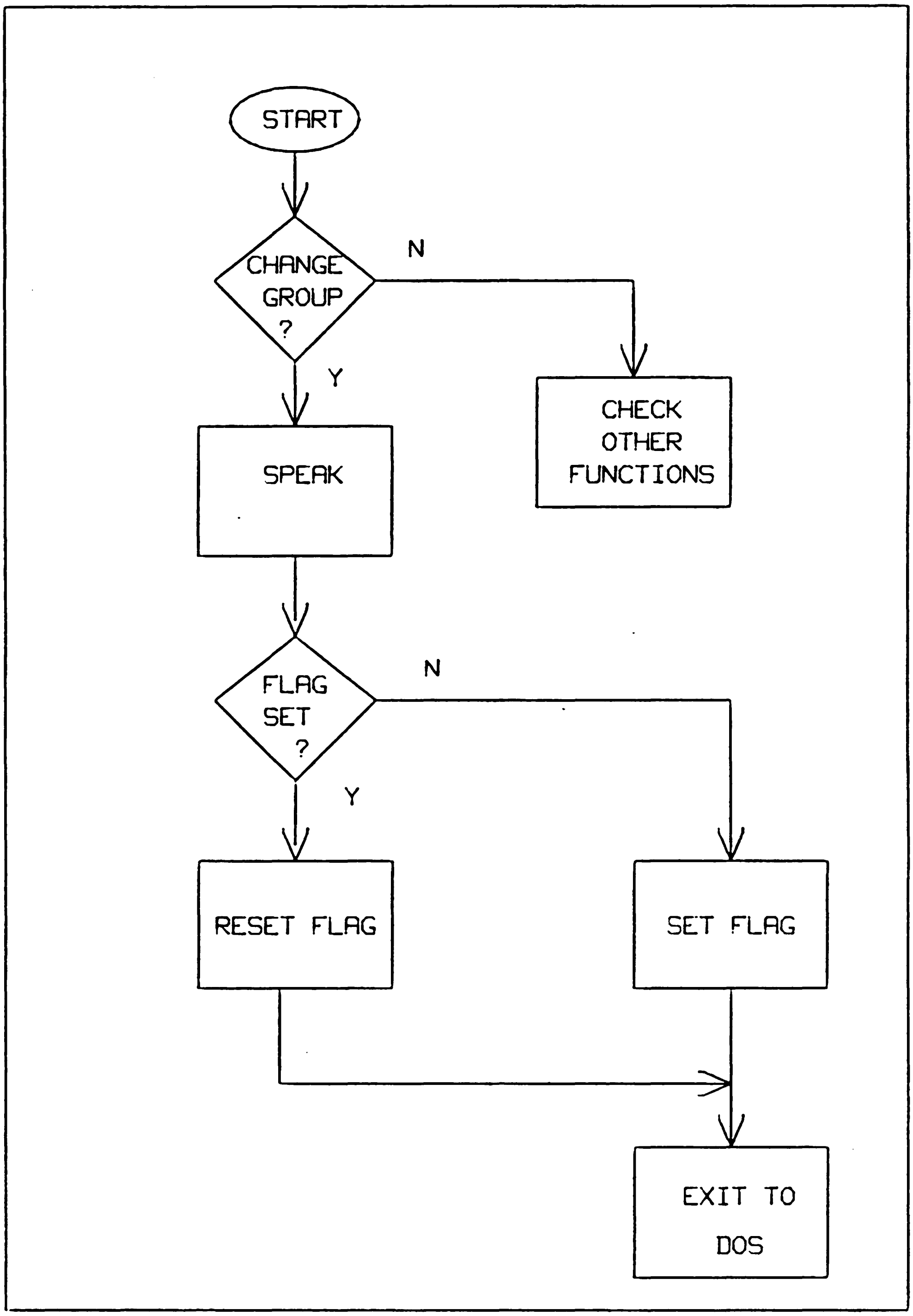


Figure 17. Change Function Description.

corresponding flag is tested for an ON (set to binary 1) condition. If the flag were found to be OFF, then it is set. Alternately, if the flag were set, it is reset. The program then exits to DOS. The next key stroke would encounter a set condition on the Static Mode and the program would vector to the Static Mode. Any of the flags set/reset in this group would be checked by all other routines to control their modes of operation.

All commands in this group operate on this principle of setting or resetting flags.

- C - Console Echo This feature controls the console echo in the Dynamic mode. If this flag were OFF, the video processing routines are bypassed in the Dynamic Mode. Note that function and other special keys may not be inhibited through this feature. By default, this flag is active, resulting in an interactive mode on boot up.

- K - Keyboard Echo This controls the audio output of keystrokes. The fixed format keys do not form a part of the set controlled by this option. Enabling this switch would cause all key entries to be treated in the letter mode. For example, the command ERASE, with the keyboard echo on, would be spoken as E R A S E. That is, each keystroke would be individually pronounced. Activating this feature automatically disables the console echo.

- L - Letter Mode Use of this switch would return control back the letter mode environment. All key entries and DOS responses default to this format. Note that this action is complementary with the W command. Only one flag may be active at any time. Even though the same flag is used by both commands, two commands were provided for ease in use. Use of individual commands provides a means to directly switch into the desired mode.

- P,S - Punctuations and Spaces These are similar to the word or letter mode switches and no further elaboration is done.

- Q - Quit Speech This is a special command in a sense that it immediately purges the PSS buffer and terminates the current audio output. Also, a flag is set to inhibit further speech operations. This is similar to the ALT O command in the Dynamic Mode.

- V - Voice Change This command enables the user to change the rate and inflection parameters for the PSS. Changes made here may not be saved back to the disk. Operationally the procedure is similar to the boot up sequence.

4.1.3.4 Help Group

This group consists of executable commands. Executable in a sense that they do more than just set or clear flags. "Help" is a general classification, but seems justifiable since they aid the user with advanced features. These commands are effective in the Static Mode only and have do not correlate to the Dynamic Mode in any way. Hence, these may be considered to be " completely static ". Some of these functions, though invoked by single keystrokes, need further entries to complete the command.

- A - Assign Markers This is a feature that allows the user to define certain locations of interest in the screen and rapidly prefix to that point on a single keystroke. Ten such keys or " Markers " are supported by the program. For example, consider how a marker may be set:

  1. Invoke the Static Mode.
  2. Move the cursor to the point of interest.
  3. Enter "A". Wait for an audio response.
  4. Now, enter F1-F10. (the PC function keys)
  5. Exit the Static Mode.

Any future entries of the defined function key in the Static Mode will result in the cursor locating to the defined location. Note that these keys may be reassigned at any time in the Static Mode. The key F10 has been set to the location where a MORE would appear on the screen when using YTERM to communicate with IBM's VM systems. All other keys are set to location 00 (row 0, column 0) by default. These pointers, with the exception of F10 are volatile and will have to be redefined each time the system boots up.

- F - Find String This is a string search command. String of up to 80 characters are supported. No restriction is made on the nature of the string, but strings starting with a blank character will tend to slow down system performance. This is so since the string search algorithm scans the screen for a match on the first character of the string. If a tally is made, then the checksum is computed for the target string. This value is then compared with the checksum computed for the source string. If the values match, the string is spoken and the cursor defaults to the first character of the string. If no matches are made, the message " Not Found " is spoken and the cursor defaults to the top of the screen.

For repeated searches, the DEL key on the PC is used. This will locate to the next occurrence of the string. If subsequent matches are not found, the cursor defaults to the top of the screen. Further use of the DEL key will cause a wrap phenomena. That is, the first occurrence will be found.

To use this feature, enter "F" followed by the string. Terminate the string with a <CR>. Note again that commands are not case sensitive, but the characters comprising the string are!

- I - Identify Current Row This command enables the user to have the current row spoken. The format of speech depends on the change function group settings.

- M - Merge Cursor As discussed earlier, the cursor in the Static Mode in different from the DOS cursor. To locate to the current line of DOS, this command may be used.

- + - Row / Column of current cursor This will speak the row and column of the current cursor. A typical application would be use with the "A" command to set markers.

4.1.3.5 Range Reader Group

The Range Reader function presents to the user the entire screen as a function of line numbers. The alternate approach supported is to view the screen relative to the current position of the cursor. The later is considered first.

The keys used here are the four cursor control keys. The cursor up and cursor down keys read the lines above and below the current line. The cursor right and cursor left keys read words/letters to the right or left of the cursor. That is, they are word/letter advance and reverse functions. The word/letter option is specified since the audio format is determined by the setting of the change function group.

To view the screen as a function of line numbers, consider the screen to consist of 25 lines, the first line being line 1 and the last line, line 25. Lines may now be read individually or as a range. Commands should adhere to the specified syntax which can be found in the User's manual. An example in shown below:

- X,Y<CR>
- Subject to X <= Y. Where,
- X --> Lower Range .
- Y --> Upper Range.

- The comma is the terminator between X and Y. Also, either X or Y may be omitted. For example,
- X<CR>
- Y<CR>

The commands not discussed are the help command, "H" and the exit command, "X". H provides the user with a menu of the current settings and X exits to the Dynamic Mode.

4.1.4 MODE INTERACTION

Even though both modes are independent of each other, some interaction does exist between them. The Static mode for example, has to be invoked from the Dynamic mode. Further, some commands in the Static Mode have a direct effect on the Dynamic Mode of operation. It is the purpose of this section to outline these overlaps and compare these modes.Figure 18 on page 64 shows the overlap area between these modes.Figure 19 on page 65 shows commands that affect the Dynamic Mode. This chart pictures their relationship.
Mode Interaction may further illustrated by considering an example. Assume that the user had set the keyboard echo ON in the Static Mode and has exit the Static Mode. Now, since the keyboard and video interrupts have been reassigned, any key stroke will pass the ASCII code to the Dynstat interrupt handler. One of the checks in this handler would be the sta-

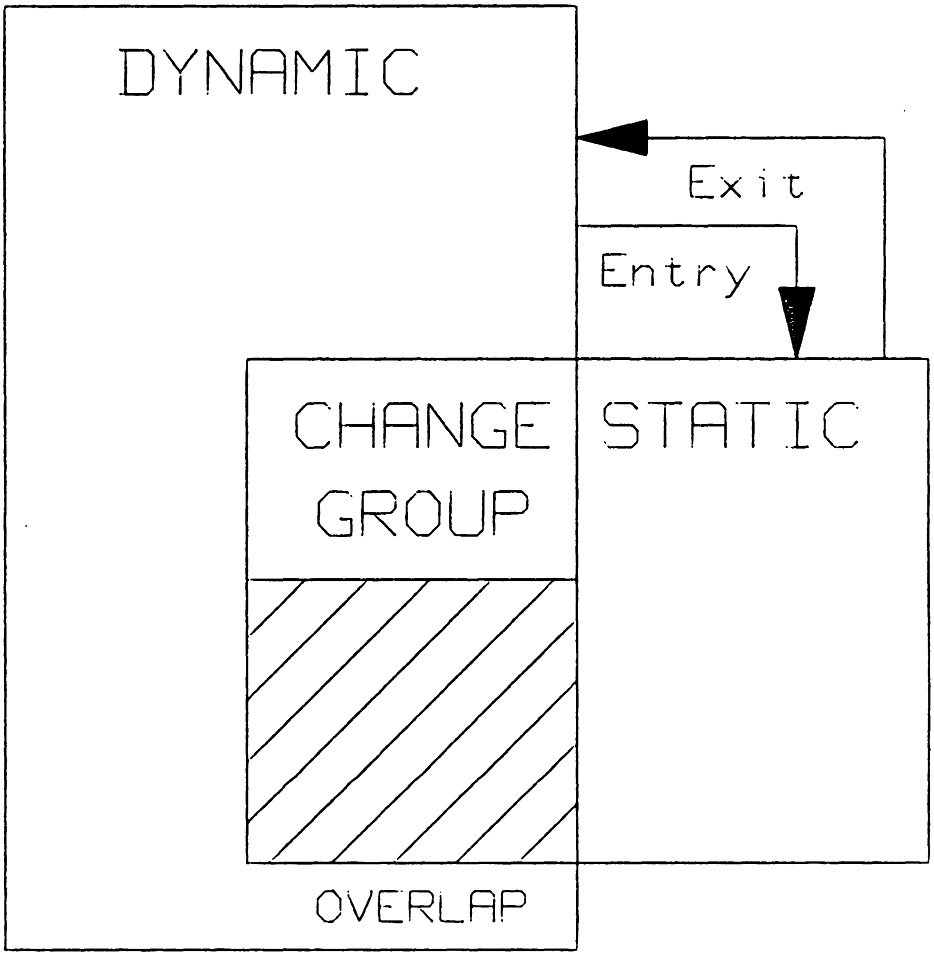


Figure 18. Mode Overlap.

STATIC	DYNAMIC	COMMON
C	NONE	
K	NONE	
L	NONE	
W	NONE	
P	NONE	
S	NONE	
V	NONE	
Q		ALT 0

Figure 19. Common Command Chart.

tus of the flag for keyboard echo. If this is active, as is in this case, the program would echo the key stroke back to the user and exit. It should be mentioned that enabling the key echo flag in the static routine automatically clears the console echo flag. When DOS calls the video interrupt to display the key struck, the Dynstat interrupt handler for video interrupts tests the console flag and disables audio output. This is best illustrated in the activity chart shown in Figure 20 on page 67. Though this just depicts one particular function, this chart may be generalized for all commands that interact with the Dynamic Mode.

The only command supported by the Dynamic Mode is the quit speech command. This is executed by the ALT 0 combination from the keyboard. This command is equivalent to the "Q" command in the Static Mode. However, note that this will not disable the the fixed message keys.

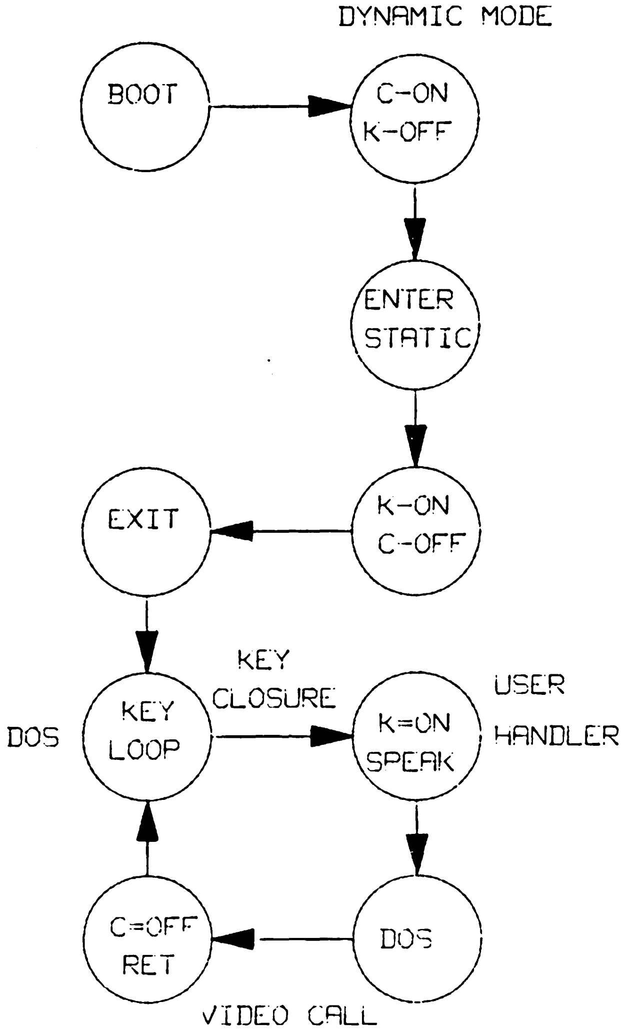


Figure 20. Activity Execution Chart.

5.0 CONCLUSION

In developing this shell, the only assumption made was that the user be familiar with the standard keyboard. However, for a new visually impaired user, this problem may be overcome by attaching braille symbols over the key tops. Hence, by the sense of touch, a person can sense the key being struck. Most of the design specifications were obtained from visually impaired users, currently using similar utilities. Further, it is planned to issue DYNSTAT to the visually impaired users on campus and update the program, based on the feedback obtained.

5.1.1 PROBLEMS ENCOUNTERED

In the process of developing this software, numerous problems on the PC DOS surfaced. The major problem that compelled the software to be written completely in assembly was the INT 21 structure. This interrupt is used by all DOS programs and the function call approach to programming specified in the manual does not mention latent faults. For example, the lack on reentrancy is a major drawback.

In the approach discussed earlier, the keyboard interrupt was trapped. In conjunction with this if an INT 21 were to be used, the program would cause the machine to hang up. The reason for this is that INT 21 checks the keyboard first. INT 21H was the main reason why the C programming language had to be given up for this application. The poor structuring of INT 21H poses a lack of flexibility to the programmer. Future developers of similar applications, beware !

Utilities like YTERM and Volkswriter write display information directly to the screen buffer.

5.1.2 OBSERVATIONS

The drawbacks imposed by INT 21H were overcome in this thesis by adopting an approach that required the entire code to be written in assembly, thereby providing easy access to BIOS routines. One of the major outcomes of this work is the knowledge of reconfiguring the IBM PC keyboard and video interrupts to almost any application. Any future work in this light may use the constructs provided here as a guideline to interface to basic system functions on the IBM PC. Examples of mapping the DOS vectors to user program interrupt handlers are given in Appendix [E]. It is suggested that the user read Appendix [E] prior to program development. Use of the approach presented will most certainly result in a saving of time and also provide the programmer with a lot of flexibility that may not be available through the use of recommended procedures in the DOS manual.

Also worth mention is the idea presented to prevent multiple execution of resident programs. Resident program requisites and problems are discussed in Chapter 3. To reiterate,

- Perform a Vector check on an interrupt being reassigned.
- If the Data matches with the unchanged values, proceed.
- Else, exit the program with NO changes on the vector.

Notable also is the provision to save a fixed setting for the PSS on the disk. The sense of hearing is a very individualistic characteristic. By providing every user with a personalized format, this thesis caters to all levels of hearing needs.

Further, this thesis also introduces a visually impaired user to mainframe usage. The use of the Static mode to review a screen presents new possibilities to unlimited usage of the developed software.

Dynstat supports a variety of application software. Some of the utilities tested are:

- Volkswriter
- YTERM
- DOS BASIC
- Microsoft FORTRAN, Macro Assemblers and Compilers
- PCX
- DVED
- EDLIN
- DEBUG

These programs are the ones that have been tested. However, other programs will also be supported. The structuring of the interrupts and the use of the Static and Dynamic modes make this possible.

The Static mode provides the user with almost all of the features available on standard word processors. Definition of marker keys is a new concept and it is bound to to serve as a powerful feature in the use of Dynstat. It should also be mentioned that normal DOS execution (from a user's point of view) is in no way affected by the use of Dynstat.

Also, real time processing has been incorporated into the software by way of the Dynamic mode. This is an entirely new idea not supported by similar utilities.

5.1.3 SYSTEM COST

Now, to consider the cost aspect. As mentioned in the second chapter, the minimum configuration would be:

- A PC with two serial ports,
- A Votrax voice unit,
- Substantial on board memory to compensate a 10k load.

The last requirement is rather critical since the shell developed is resident in nature and other application programs being run at a later stage should have enough system memory for proper operation. An optimal memory size would be around 256k.

The cost then, would be the price of the mentioned minimum configuration. The need for two serial ports in to enable the user to access the main frame, with the shell resident.

5.1.4 FUTURE WORK

To produce a powerful tool to cater to such applications, a multi-tasking approach seems to offer a lot of advantages. This is true in applications such as YTERM.

Communication with the mainframe is a problem prone area. With application software such as Dynstat performing real time communication with the PSS, mainframe communication must also be maintained to avoid loss of data either to the PSS or the mainframe. This was one of the main considerations in defining the Static and Dynamic modes. When the Static mode is active (the only mode available with YTERM), no communication is possible or necessary with the mainframe.

To provide real time processing by both YTERM and Dynstat, a time sharing approach for the CPU must be chosen. The concept of multi-tasking lends itself ideally to such a purpose. Hence, multi-taksing may be considered to be an area where further work may be done.

Further, the program may be changed to execute on a PC-AT. Currently, the machines supported are the IBM-PC Portable, IBM-PC and the PC-XT.

BIBLIOGRAPHY

1. Votrax Personal Speech System Operator's Manual,Votrax, Copyright 1982.

2. Microsoft Macro Assembler,Microsoft Corporation, Copyright 1984.

3. YTERM 1.2 Primer, Document Number PC03, Copyright 1982, 1983, 1984 Yale University. Published by User Services Department, Virginia Tech Computing Center.

4. YTERM 1.2 User's Guide, Document Number PC09, Copyright 1982, 1983, 1984 Yale University. Published by User Services Department, Virginia Tech Computing Center.

5. YTERM 1.2 Technical Reference, Document Number PC10, Copyright 1982, 1983, 1984 Yale University. Published by User Services Department, Virginia Tech Computing Center.

6. IBM DISK OPERATING SYSTEM Technical Reference, #6024125, Copyright IBM Corporation, 1983.

7. IBM Personal Computer Hardware Reference Library, Technical Reference, #6025005, Copyright IBM Corporation, 1983.

8. Holt, Charles Ashbury, Microcomputer Organization, Macmillan Publishing Company, 1985.

9. Volkswriter deluxe, Life Tree Software Company, Copyright 1983, 1984.

10. IBM BASIC Manual, #6025010, Copyright IBM Corporation, 1983.

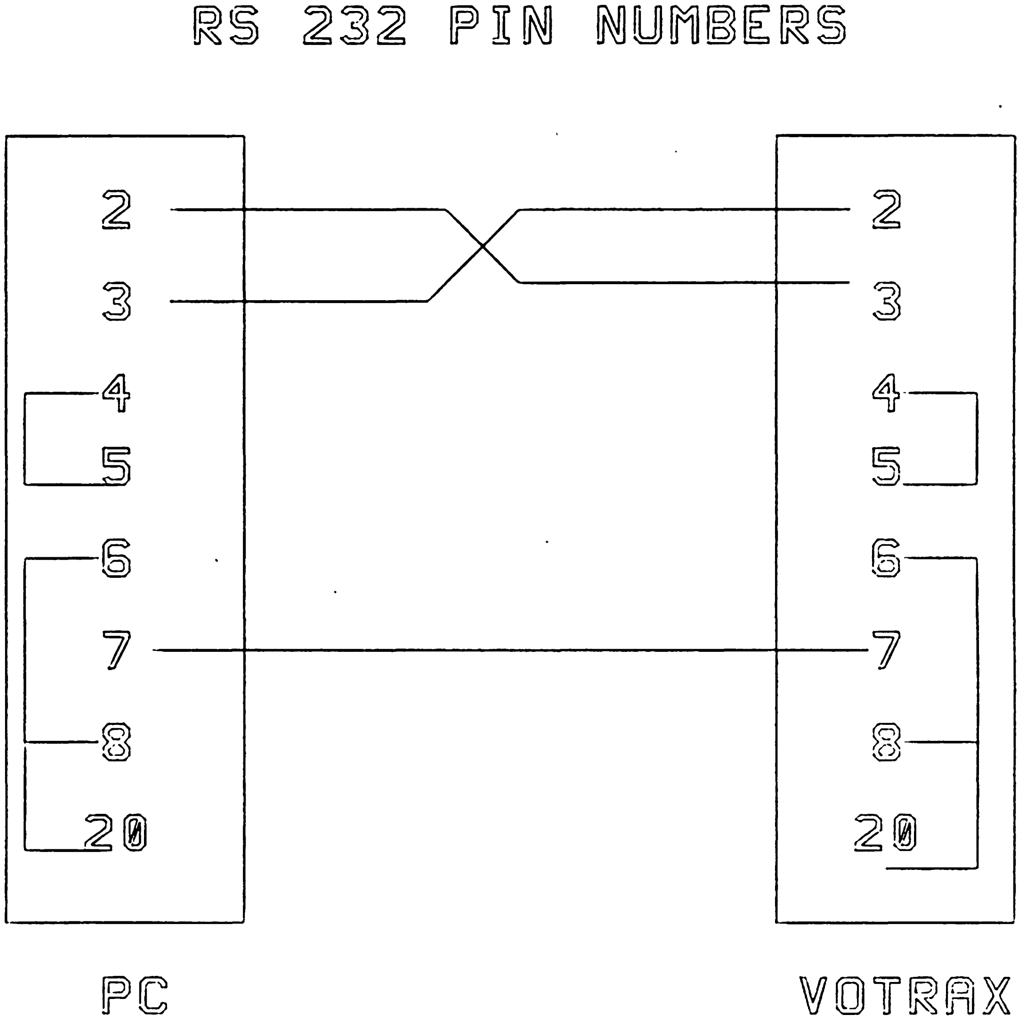
11. iAPX 8086,88 User's Manual, Copyright intel Corporation, August 1981.

12. Freedom1 User's Manual, Version 1.21, Copyright Interface Systems International, 1984.

13. King, Richard Allen, The MS-DOS Handbook, Copyright Sybex Inc., 1985.

14. Socha, John, Assembly Language Safari on the IBM PC: First Explanations, Copyright Brady Communications Company, Inc., 1984.

APPENDIX A. PC - VOTRAX TERMINATION



APPENDIX B. USERS'S MANUAL

The purpose of this manual is to provide the user with explanations regarding program syntax. Dynstat supports two modes:

1). The Static Mode

2). The Dynamic Mode

The words static and dynamic refer to the use of the functions that are available to the user.

The Static Mode corresponds to an off line approach, where the screen is held static. That is, no interaction with DOS is possible.

In contrast, the Dynamic Mode refers to the interactive mode that works with system DOS.

Installation

To execute Dynstat, the minimum requirements of the system are:

1). Two serial ports (if a modem is used),

2). The Votrax PSS voice unit connected to the Com1 port on the PC.

Now, to start, boot the speaking system with the program disk in drive A:.

You should now hear the following messages:

" The Quick Brown fox Jumps Over The Lazy Dog "

" Do you wish to change default values "

The default values on the second message refer to the voice output of the Votrax voice unit. If you need to change this format, enter "Y". Note that user responses are not case sensitve. The quick brown fox message will be echoed again and you will hear:

"Enter R for Rate, I for Inflection and Return to Exit"

R corresponds to the speaking rate of the device and I to the inflection level of the voice. A return will exit from this mode. To change the settings, the CURSOR UP and CURSOR DOWN keys are used. For each change, the test message will be spoken with the new settings. When the upper or lower limits are reached, you will hear a beep. This signifies that values above or below this may not be set.

To exit from this, just enter a return. You will hear the message,

"Do you wish to save these settings "

If you need the system to always boot with these values, type "Y". These values will then be saved on the disk. Note that the default drive is A:.

These values are saved in the file DEFAULT.VAL. If you had entered a "N", then the changes will not be saved, but the system will remain in the current format till the next boot up.

You may also wish to change the default port, Com1. To do this, edit the file DEFAULT.VAL on the program disk. The first entry in this file will be a zero (0). Change this to a one (1). This will change the default port to Com2. Of course, you will have to boot up again, with the Votrax connected to the new port.

Also note that the DIP switches on the rear panel of the Votrax should be set to the following:

Reading from left to right: 00010100. Here '0' corresponds to the switch down and '1' to the switch up.

000 corresponds to 9600 baud.
1 corresponds to the XON/XOFF protocol for the serial port.
0 corresponds to a 7 bit word plus parity.
1 the power on message of the Votrax will be spoken.
0 this is for the serial port.
0 normal operation.

Default Settings

On entry, certain values are set to their default values. The parameters set are:

1). A word format.

2). Only select punctuations such as ,.;: and > are spoken.

3). No spaces are spoken.

4). The console echo is on.

You can turn off the console echo by holding the Alt key down and pressing 0. That is, Alt 0. Note that this will not disable keys such as <CR>, space, tab etc....

Static and Dynamic Modes

You are now in the normal DOS mode with these settings. This is called the ' Dynamic Mode'. In contrast, the mode you enter to change any/all of these settings is the 'Static Mode '. To enter the static mode, depress the shift key and the back slash key simultaneously. These are the keys on the left hand side of your alphanumeric keyboard. You will hear the message,

"Enter command mode "

You are now in the static mode, where the entire screen is at your disposal. To exit the static mode and to return to DOS, just type 'X'. You will hear the message,

"Exit command mode "

You will now return to the DOS cursor location.

Command Syntax

Now, consider the functions availble in the static mode. When the cursor is mentioned in the static mode, it should be understood that this cursor is a pointer that is under your control and is in no way related to the DOS cursor. THE CURSOR IN THE STATIC MODE JUST KEEPS TRACK OF WHERE YOU ARE ON THE SCREEN AT THAT POINT IN TIME. You can always merge the static cursor to the point where the DOS cursor is, but the cursor in the static mode is an entirely different entity.

Commands in the static mode are invoked on single key strokes. The letter that you have to remember for execution of functions is generally the first letter of the associated function.

For example, a string search is invoked through the letter 'F', which is the first letter of the word 'FIND'.

A ready look up of the commands available is now provided :

A- Assign values to the 10 markers (F1 - F10) (page 86).

C- Console echo (page 87).

F- Find, a string search feature (page 85).

H- Help menu (page 90).

I- Identify current row (page 89).

K- Keyboard echo (page 87).

L- Letter mode (page 88).

M- Merge Static cursor to DOS cursor (page 89).

P- Punctuations, select/all (page 88).

Q- Quit speech (page 87).

R- Range reader (page 85).

S- Spaces spoken, all/none (page 88).

V- Voice changes (page 89).

W- Word mode (page 88).

X- Exit static mode (page 90).

+- Speak row/column of cursor (page 89).

A detailed description of these functions will now follow:

Command : 'R'

Function : Will enable the Range Reader mode. Movement through the entire screen is allowed.

Keys needed : Cursor up, Cursor down, Cursor left, Cursor right, Home and End. A beep is used to indicate that the cursor is at the top (upper left) or at the bottom (bottom right) of the screen. The up and down keys are used to move up/down by one line. The right and left keys are used to move right/left by a word/letter.

Syntax : lower range, upper range <CR>.

Example : 1,2 <CR>

Errors : Ranges in wrong format. With reference to the above example, 2,1 <CR> will be treated as an error. The first line on the screen is number 1 and the last line, number 25.

Command : 'F'

Function : String search feature. Initial searches start from the top of the screen. Subsequent searches locate the next occurence of the string. A 'Found' message will be given when the string is found.

Keys needed : The DEL key will enable multiple searches of the string.

Syntax : Fstring. Strings of upto 80 characters are allowed.

Message : A 'Not found' message is spoken and the cursor defaults to the top of the screen.

Command : 'A'

Function : Set markers. Keys F1-F10 are the marker locate keys. When any marker key is used, the cursor will locate that point and the word at that location will be spoken.

Keys needed : F1-F10.

Syntax : Position cursor at desired location. Hit 'A', then F1-F10.

Further use of set keys, F1-F10 will default to the set point.

Command : 'Q'

Function : Speech/Silent key. This works as a toggle switch. Note that this has the same function as Alt 0 in the dynamic mode.

Command : 'C'

Function : Turns on/off console echo in the dynamic mode.

Command : 'K'

Function : Enables keyboard echoes in the dynamic mode. Automatically kills console echo.

Command : 'W'

Function : Set speech to word mode. All screen read operations from this point are in word mode. Note that with console echo on, spaces are needed after each word to be spoken.

Command : 'L'

Function : Set speech to letter mode.

Command : 'S'

Function : Toggle switch to control speech of spaces.

Command : 'P'

Function : Toggle switch to speak all/select punctuations.

Command : '+'

Function : Speaks the current row and column of the cursor.

Command : 'I'

Function : Will identify the current row by speaking the entire row.

Command : 'M'

Function : Will merge the static cursor to the DOS cursor.

Command : 'V'

Function : Changes on the voice of the votrax can be made through this. The operation is simillar to the boot up sequence, but note that changes made here will not be stored back to the disk.

Command : 'H'

Function : The help menu will speak the settings currently active. The location of the markers will also be spoken.

Command : 'X'

Function : Exit to DOS. The Dynamic mode will be active now, with the modes set in the Static mode.

Conclusion

Note that marker F10 has already been set to the position where the 'MORE' command appears in the use of YTERM.

In the Dynamic mode, beeps may be heard. These can be ignored as they are part of the protocol used to communicate with the votrax.

Also note that upper case letters encountered during letter reads will be spoken with the 'CAP' prefix. The same message will occur in the key board echo mode also.

Note that the following keys will not be spoken : Cntrl, Shift, Caps Lock, Num Lock and the Scroll Lock. All other keys will be echoed back to the user, depending on the mode set.

You are encouraged to make a copy of the master diskette. The program is not copy protected and either the copy command or the diskcopy command will work. Use of the diskcopy command is preffered since the file DEFAULT.VAL should exist on the same disk as the DYNSTAT.COM file.

Program Disk Files

Your program disk should contain the following files:

* COMMAND.COM
* DYNSTAT.COM
* AUTOEXEC.BAT
* USER.DOC
* DEFAULT.VAL

COMMAND.COM is standard DOS file needed to boot up.

DYNSTAT.COM is the speech program.

The AUTOEXEC.BAT ensures booting with the speech program.

USER.DOC is the User's Manual, in a Volkswriter format.

DEFAULT.VAL is the file to store default values for the PSS.

APPENDIX C. PSS ASCII CHART.

PHONEME CONVERSION CHART

Hex Code	ASCII Char.	Phoneme Symbol	Duration (ms)	Example Word
40	@	EH3	59	jackEt
41	A	EH2	71	Enlist
42	B	EH1	121	hEAvy
43	C	PAO	47	-PAUSE-
44	D	DT	47	buTTer
45	E	A2	71	enAble
46	F	A1	103	mAde
47	G	ZH	90	meaSure
48	H	AH2	71	hOnest
49	I	I3	55	inhibIt
4A	J	I2	80	Inhibit
4B	K	I1	121	inhIbit
4C	L	M	103	Mat
4D	M	N	80	suN
4E	N	B	71	Bag
4F	O	V	71	Van
50	P	CH*	71	CHip
51	Q	SH*	121	SHop
52	R	Z	71	Zoo
53	S	AW1	146	AWful
54	T	NG	121	thiNG
55	U	AH1	146	fAther
56	V	OO1	103	lOOking
57	W	OO	185	bOOk
58	X	L	103	Land
59	Y	K	80	Kitten
5A	Z	J	47	JuDGe
5B	[	H	71	Hello
5C	V	G	71	Get
5D	]	F	103	Fast
5E	^	D	55	paiD
5F	_	S	90	paSS

APPENDIX D. PSS COMMAND INITIATORS

SUMMARY OF CONTROLS

SPEECH FEATURE
CONTROL CHARACTER — @ (Ampersand)

CONTROLLED FEATURES —	@R	rate
	@(0-7)	inflection
	@A	amplitude
	@C	conversion mode
	@V	voice mode

NON-SPEECH FEATURE
CONTROL CHARACTER — ! (Exclamation Point)

CONTROLLED FEATURES —	!(1,2,3,)	musical tone channel
	!A	alarm set
	!B	baud set
	!E	envelope set
	!F	filter set
	!L	load
	!N	noise set
	!P	prompt
	!T	tempo set
	!U	user program
	!W	wait

ATTENTION FEATURE
CONTROL CHARACTER — [ESC] (ESCAPE CODE)

CONTROLLED FEATURES —	[ESC]C	connect I/O
	[ESC]M	mode set
	[ESC]P	powerup
	[ESC]Q	quit
	[ESC]R	reserve memory
	[ESC]S	special characters
	[ESC]T	time set
	[ESC]V	speak version

APPENDIX E. INTERRUPT ALLOCATION

```
        ; Programming Examples
        ; for Interrupt Reallocation.
        ; Int 10H is used as an example here.
        ; Int 10H is the DOS ROM BIOS Video Vector
        .

        my_code segment para public 'CODE'
        assume cs:my_code,ds:my_code,ss:my_code
        ; CS and DS MUST exist in the same segment.

        ; define procedures here
        public    main
        ; Set origin for COM files format

        org  100H

        main proc far  ; This is essential !!!

start:  jmp     begin
        ; Data initializations may be done here
begin:  mov     ax,3510h     ; 35 = DOS function call,
        int     21h          ; 10 = Vector
        push    ds           ; save segment address
```

```
mov     ax,es         ; segment address of Int 10h
mov     ds,ax
mov     dx,bx         ; offset of Int 10h
mov     ax,2545h      ; 25 = DOS function call,
                      ; 45 = ramdom
int     21h
pop     ds            ; restore working area
;
; now to set up the user interrupt handler
;
mov     dx, offset video_int
mov     ax,2510h      ; 25 = change vector
int     21h           ; over
;
; now to make code resident
;
mov     ah,49h      ; free memory
int     21h
mov     dx,1024     ; 1k resident code.
mov     ax,3101h    ; function calls again !
int     21h         ; back to DOS
main        endp
video_int   proc   far
; User video routines here
video_int   endp
my_code     ends
```

```
     end          main   ; close procedure

;

; this rule applies to all BIOS vectors.

; actual addresses used here in place of

; of EQU directives.

; DOS Technical Information -

; Technical Reference, Software for function calls.

; Technical Reference, Hardware for BIOS vectors.

; Note absence of stack initialization.

;
```

APPENDIX F. PROGRAM SOURCE CODE

```
title DYNSTAT.ASM

page ,132

;*********************************************************************
;*                                                                   *
;*          DYNSTAT : An Interactive Software for the Votrax PSS.    *
;*          Host System : IBM PC - Portable, PC and the XT.          *
;*          Related files : DEFAULT.VAL                              *
;*                                                                   *
;*********************************************************************

my_code segment para public 'CODE'

assume cs:my_code,ds:my_code,ss:my_code

;*********************************************************************
;*                                                                   *
;*      Procedure definitions are made here                          *
;*                                                                   *
;*********************************************************************

        public  main                    ; main program
        public  value                   ; decimal conversion
        public  mode_set                ; PSS voice set
        public  checkup                 ; screen upper boundry
        public  checkdn                 ; screen lower boundry
        public  rowcol                  ; row/col speech
        public  case                    ; upper case letters
        public  getter                  ; find string
        public  summer                  ; check sum
        public  kbdint                  ; keyboard interrupt
        public  outer                   ; main transmit routine
        public  stat                    ; XON/XOFF protocol
        public  nonint                  ; Static Mode
        public  adjust                  ; word right
        public  adjlt                   ; word left
        public  next                    ; fixed messages
        public  spkron                  ; speaker on
        public  video                   ; video interrupt
        public  rot1                    ; cursor manipulation
        public  rot2
        public  rot3
        public  rot4
        public  rot5
        public  again                   ; transmit fixed messages
        public  setter                  ; serial interrupt
        public  line                    ; space check
        public  mbx                     ; convert hex line numbers
        public  view                    ; screen reader
        public  quit                    ; purge PSS buffer
        public  quitl
        public  helper                  ; help menu

;*******************************************************************
;                                                                  *
;         Main Program Begins                                      *
;         Org at 100h to satisfy COM file requirements             *
;                                                                  *
;*******************************************************************

;       Set up origin for COM file format

        org     100h

main    proc    far

;*******************************************************************
;                                                                  *
;         Procedure main: Vector checks, reallocation and voice levels *
;                         are set here.                            *
;                                                                  *
;*******************************************************************

start:  mov     ax,3510h                ; to check if program
        int     dosint21                ; already installed
        cmp     bl,65h                  ; int 10h vector
        jnz     firstl                  ;
        cmp     bh,0f0h                 ; 0f065h BIOS Int 10h address
        jnz     firstl
        jmp     begin                   ; no, first time
firstl: mov     ax,cs
        push    ds
        mov     ds,ax                   ; already installed
        lea     bx,install
        call    next
        pop     ds                      ; restore segment
        mov     ah,4ch                  ; exit to DOS
        int     dosint21

;       Data Segment initializations are done here

buffer          dw  80 dup (00)         ; word buffer for reverse read

bytes_store     db  6  dup(00)          ; range buffer

string_loc      db  80 dup(00)          ; source string
```

```
sum_string      db  80 dup(00)          ; screen string
dosint21        equ 21h                 ; dos int 21h
dosint16        equ 16h                 ; dos int 16h
dosint10        equ 10h                 ; dos int 10h
int46           equ 46h                 ; user int 46h
dosint14        equ 14h
loc_screen      dw  00                  ; for multiple searches
static_exit     db  00                  ; static operations
mult_try        dw  00                  ; multiple reads
fun_flag        db  01                  ; markers set
fun_key1        dw  00                  ; F1
fun_key2        dw  00                  ; F2
fun_key3        dw  00                  ; F3
fun_key4        dw  00                  ; F4
fun_key5        dw  00                  ; F5
fun_key6        dw  00                  ; F6
fun_key7        dw  00                  ; F7
fun_key8        dw  00                  ; F8
fun_key9        dw  00                  ; F9
fun_key0        dw 00                   ; F10
entry_save      db  00                  ; static mode character save
count_save      db  00                  ; multiple searches
string_count    db  00                  ; multiple searches
comm_port       dw  00                  ; RS-232 0/1
cur_count       db  00                  ; range reader
numlck          db  00                  ; num lock key
byte_count      db  00                  ; range reader
```

```
low_range     db  00              ; first line
high_range    db  00              ; last line
termin        db  00              ; terminator for range reader
read_screen   db  00              ; enable flag
con_echo      db  00              ; console echo
key_echo      db  01              ; keyboard echo
flag_space    db  00              ; DOS screen mode
hold_flag     db  00              ; XON/XOFF protocol flag
alt_shift1    db  01              ; default, word mode
alt_shift0    db  00              ; quit speech
alt_shift4    db  01              ; default, select punctuations
alt_shift3    db  00              ; default, one space only
no_write      db  00              ; default values not saved
normal_flag   db  00              ; normal exit
new_row       db  00              ; DOS end of line
spacer        db  00              ; contol spaces
old_row       db  00              ; current row indicator
cret_flag     db  00              ; manual retn. code
loc_saver     dw  00              ; start location for search
dumb_saver    dw  00              ; DOS cursor saver
dta           db  ?               ; disk read parameters
fcb           db  0               ; disk read parameters
              db  'default val'   ; file name
              db  24 dup(0)       ; other locations
ibuf          db  00              ; inflexion
rbuf          db  00              ; rate
rate_flag     db   00             ; rate changed
infl_flag     db   00             ; inflexion changed
```

```
key_flag       db   00                  ; to indicate key code, not DOS
word_flag      db   01                  ; default word mode
sum            dw   00                  ; check sum from screen
check_sum      dw   00                  ; check sum
cksum_save     dw   00                  ; check sum save
               even
kbd_int        dd    ?                  ; user int 16 routine
               even
video_int      dd    ?                  ; user int 10 routine
;
;              User prompts/messages
;
msg            db  ' return$'
escape         db  ' escape$'
aat            db  ' at$'
exclaim        db  ' exclamation mark$'
home           db  ' home$'
string_found   db  ' found$'
not_found      db  ' not found$'
curup          db  ' up$'
curdn          db  ' down$'
currt          db  ' right$'
curlft         db  ' left$'
pgdn           db  ' page down$'
pgup           db  ' page up$'
ins            db  ' insert$'
del            db  ' delete$'
mes2           db  ' end$'
rownum         db  ' row$'
```

```
colnum        db  ' column$'
tab           db  ' tab$'
back          db  ' back space$'
caps          db  ' cap$'
shift         db  ' shift$'
ctrl          db  ' control$'
break         db  ' break$'
backslash     db  ' back slash$'
alt           db  ' alt$'
space         db  ' space$'
install       db  ' program already installed$'
errorl        db  ' bad range numbers$'
blank         db  ' blank line$'
period        db  ' period$'
comma         db  ' comma$'
openq         db  ' open quote$'
endq          db  ' end quote$'
quote         db  ' quote$'
colen         db  ' colen$'
semic         db  ' semi colen$'
openb         db  ' open brace$'
closeb        db  ' close brace$'
opens         db  ' open bracket$'
closes        db  ' close bracket$'
openbr        db  ' open parantesis$'
closebr       db  ' close parantesis$'
minusc        db  ' minus$'
uscore        db  ' under score$'
```

```
quest          db  ' question mark$'
carrat         db  ' carrat$'
tilde          db  ' tilde$'
alt1off        db  ' letter mode$'
alt1on         db  ' word mode$'
alt3off        db  ' all spaces spoken$'
alt3on         db  ' no spaces spoken$'
alt4on         db  ' select punctuations$'
alt4off        db  ' all punctuations$'
salt0          db  ' alt 0$'
fox            db   ' the quick brown fox jumps over the lazy dog$'
default        db   ' do you wish to change default values$'
mode           db   ' enter r for rate i for inflection '
               db   'and return to exit$'
save           db   ' do you wish to save these settings$'
welcome        db   ' enter command mode$'
bye            db   ' exit command mode$'

;       end Data Segment initialization

begin:  mov     ax,3516h                ; to save and set int 16h
        int     dosint21
        push    ds                      ; save & set up segments
        mov     ax,es
        mov     ds,ax
        mov     dx,bx
        mov     ax,2545h                ; Int 45h original vector
        int     dosint21
        pop     ds
        mov     di, offset kbd_int      ; save int 16h in kbd_int
        mov     0[di],bx
        mov     2[di],es
        mov     dx,offset kbdint        ; new int 16 vector
        mov     ax,2516h
        int     dosint21
;
;       set up flags for disk read/write
;
        mov     dx,offset dta           ; set dta
        mov     ah,1ah
        int     dosint21
        mov     dx,offset fcb           ; open file
        mov     ah,0fh
        int     dosint21
        mov     word ptr fcb + 0ch,0    ; set record size, rr/cr field
        mov     word ptr fcb + 0eh,1
        mov     fcb + 20h,0
        mov     ah,14h                  ; sequential read
        int     dosint21
        mov     ah,dta                  ; store in comm_port location
        push    dx
        mov     dl,ah
        sub     dl,30h                  ; convert from ASCII to Decimal
        mov     dh,00
        mov     comm_port,dx            ; set up com port address
        pop     dx
        mov     ah,14h                  ; sequential read
        int     dosint21
        mov     ah,dta                  ; store in inflexion location
        mov     ibuf,ah
        mov     ah,14h                  ; sequential read
        int     dosint21
        mov     ah,dta                  ; store in rate location
        mov     rbuf,ah
        mov     ah,14h                  ; read for level value
        int     dosint21
        mov     dx,comm_port
        mov     ax,0f3h                 ; f3 for 9600 baud
        int     dosint14                ; to serial port - votrax.
        call    mode_set
        cmp     no_write,01
        jz      vn
        cmp     normal_flag,00
        jz      write
        mov     normal_flag,00
nrmal:  mov     ah,rate_flag
        mov     al,infl_flag
        or      ah,al
vn:     jz      nowrt
sl:     lea     bx,save                 ; save default settings
        call    again
        mov     ah,07
        int     dosint21
        cmp     al,'y'
        jz      write                   ; user response
        cmp     al,'Y'                  ; save
        jz      write
        cmp     al,'n'
        jz      nowrt                   ; do not save
        cmp     al,'N'
        jz      nowrt
        jmp     sl

write:  mov     dx,offset fcb           ; set up record length
        mov     ah,16h
        int     dosint21
        mov     word ptr fcb + 0ch,0
        mov     word ptr fcb + 0eh,1
        mov     fcb + 20h,0
        mov     ah,ibuf
        mov     dta,ah                  ; write back data pointers
        mov     ah,15h
        int     dosint21
        mov     ah,rbuf
        mov     dta,ah
        mov     ah,15h
        int     dosint21
        mov     ah,10h
        int     dosint21
nowrt:  mov     ah,0fh                  ; # of rows/col on display
        int     dosint10
        mov     no_write,00
        mov     flag_space,ah
first:  mov     ax,3510h                ; read BIOS Int 10h
        int     dosint21
        mov     di,offset video_int
        mov     0[di],bx
        mov     2[di],es
        push    ds                      ; save segment
        mov     ax,es
        mov     ds,ax
        mov     dx,bx
        mov     ax,2546h                ; video interrupt
        int     dosint21
        pop     ds
        mov     dx,offset video         ; change
        mov     ax,2510h
        int     dosint21
        push    dx
        push    bx
        mov     dh,23                   ; default for 'MORE'
        mov     dl,60
        mov     bx, offset fun_key0
        mov     [bx],dl
        inc     bx
        mov     [bx],dh
        pop     bx
        pop     dx
        jmp     al                      ; end of program
```

```
main    endp


kbdint  proc    far

; User keyboard interrupt routine

        cmp     ah,00                   ; check for key board reads
        jz      iwant
        pushf                           ;
        call    cs:kbd_int              ; normal DOS operations
        ret     2                       ; clear stack


iwant:  pushf
        call    cs:kbd_int              ; key board reads to pass here
        push    ds
        push    ax                      ; save registers
        push    cs
        pop     ax                      ; change segments
        mov     ds,ax
        pop     ax
        cmp     al,'|'                  ; static mode ?
        jz      chknxt
        cmp     al,00                   ; extended key board codes ?
        jz      funct
        jmp     nofunct
funct:  cmp     ah,129                  ; alt 0, quit speech
        jnz     altnxt
        push    bx
        lea     bx,salt0                ; ALT 0 message
        call    again
        pop     bx
        cmp     alt_shift0,00
        jz      alset0
        mov     alt_shift0,00           ; reset
        jmp     leave
alset0: mov     alt_shift0,01
        call    quit1                   ; purge votrax buffer
leave1: jmp     leave
altnxt: cmp     ah,59                   ; F1
        jc      leave1
        cmp     ah,69                   ; F10 + 1
        jnc     keyjmp
        cmp     ah,68
        jz      f10
        push    ax
        mov     al,'f'
        call    setter                  ; speak 'f' key number
        mov     al,' '
        call    setter
        pop     ax
        push    ax
        sub     ah,58
        add     ah,30h                  ; adjust offsets
        mov     al,ah
        call    setter
f10x:   mov     al,0dh
        call    setter
        pop     ax
        jmp     leave                   ; exit
f10:    cmp     ah,68
        jz      f10z
keyjmp: jmp     key1
f10z:   push    ax                      ; exception for F 10
        mov     al,'f'
        call    setter
        mov     al,' '
        call    setter
        mov     al,31h
        call    setter
        mov     al,30h
        call    setter
        jmp     f10x
chknxt: push    bx
        lea     bx,welcome              ; enter static mode
        call    again
        pop     bx
        call    rot1                    ; get cursor points
        call    rot5
        mov     entry_save,al           ; save entry points
        mov     loc_saver,dx
lbck:   mov     ah,0                    ; wait for further commands
        int     45h                     ; execute orignal BIOS vector
        cmp     al,'x'                  ; exit code
        jz      rev
        cmp     al,'X'                  ; upper case
        jz      rev
        cmp     fun_flag,01             ; function keys ?
        jz      funbeg
oth:    jmp     other
rev:    jmp     revv
funbeg: cmp     ah,59                   ; function keys handled here
        jc      oth
        cmp     ah,69
        jnc     oth
        push    ax
        mov     al,'f'                  ; speak key
        call    setter
        pop     ax
        cmp     ah,68                   ; F10
        jz      pf0
        sub     ah,58
        push    ax
        add     ah,30h
        mov     al,ah
        call    setter
```

```
         mov     al,0dh
         call    setter
         pop     ax
         mov     bx, offset fun_key1      ; base
ixadd:   dec     ah
         jz      anfun
         inc     bx                       ; next key buffer
         inc     bx
         jmp     ixadd
anfun:   mov     dl,[bx]
         inc     bx
         mov     dh,[bx]
         call    rot2
         mov     dumb_saver,dx            ; save in pointer
         push    dx
         call    rot2
         call    spwdrt                   ; to speak word/letter
         pop     dx
         call    rot2                     ; restore to begining
         call    spkron
         jmp     lbck
pf0:     mov     al,'f'
         call    setter
         mov     al,20h
         call    setter
         mov     al,31h
         call    setter
         mov     al,30h                   ; ASCII equivalents
         call    setter
         mov     al,0dh
         call    setter
         mov     bx,offset fun_key0
         jmp     anfun
other:   call    nonint                   ; static server routine
         jmp     lbck
revv:    mov     al,entry_save
         mov     ah,00
         push    bx
         mov     static_exit,01           ; set flag to ignore all writes
         mov     dx,loc_saver
         call    rot2
         lea     bx,bye                   ; exit message
         call    again
         pop     bx
         jmp     leave
alset1:  mov     alt_shift1,01            ; set flag
         jmp     leave
nofunct:cmp      al,0dh                   ; return
         jnz     nexl
         push    bx
         lea     bx,msg                   ; car. retn.
         mov     cret_flag,01
         jmp     spkmsg
nexl:    cmp     al,20h                   ; space
         jnz     leel                     ; quit code
         push    bx
         lea     bx,space
         jmp     spkmsg                     ; space
lee1:    cmp     ah,15                      ; tab
         jnz     bspace
         push    bx
         lea     bx,tab
         jmp     spkmsg
bspace:  cmp     ah,14                      ; back space
         jnz     keypad
         call    quit1                      ; purge buffer
         push    ax
         push    dx
         call    rot1                       ; read last character also
         push    dx
         dec     dl
         call    rot2
         call    rot3                       ; manipulate cursor to do so
         pop     dx
         call    rot2
         pop     dx
         pop     ax
         push    bx
         lea     bx,back                    ; back space
spkmsg:  call    again
         pop     bx
         jmp     leave
keypad:  cmp     al,1bh                     ; escape
         jnz     key1
         push    bx
         lea     bx,escape
         jmp     spkmsg
key1:    cmp     ah,71                      ; key pad operations
         jnz     key2
         push    bx
         lea     bx,home                    ; home
         jmp     spkmsg
key2:    cmp     ah,72
         jnz     key3
         push    bx
         lea     bx,curup                   ; cursor up
         jmp     spkmsg
key3:    cmp     ah,73
         jnz     key4
         push    bx
         lea     bx,pgup                    ; page up
         jmp     spkmsg
key4:    cmp     ah,75
         jnz     key5
         push    bx
         lea     bx,curlft                  ; cursor left
         jmp     spkmsg
key5:    cmp     ah,77
         jnz     key6
         push    bx
```

```
        lea     bx,currt                ; cursor right
        jmp     spkmsg
key6:   cmp     ah,79
        jnz     key7
        push    bx
        lea     bx,mes2                 ; end
        jmp     spkmsg
key7:   cmp     ah,80
        jnz     key8
        push    bx
        lea     bx,curdn                ; cursor down
        jmp     spkmsg
key8:   cmp     ah,81
        jnz     key9
        push    bx
        lea     bx,pgdn                 ; page down
        jmp     spkmsg
key9:   cmp     ah,82
        jnz     key10
        push    bx
        lea     bx,ins                  ; insert
        jmp     spkmsg
key10:  cmp     ah,83
        jnz     keyecho
        push    bx
        lea     bx,del                  ; delete
        jmp     spkmsg
keyecho:cmp     key_echo,01             ; set ?
        jz      leave
        push    ax
        call    case                    ; check for upper case
        mov     al,0dh
        call    setter
        pop     ax                      ; restore registers
leave:  pop     ds
        ret     2                       ; clear stack


kbdint  endp


mode_set proc   near

; change voice level on the Votrax PSS

        mov     al,'@'                  ; initialize votrax
        call    setter
        mov     al,'R'
        call    setter
        mov     al,ibuf
        call    setter                  ; default values used
```

```
          mov     al,20h
          call    setter
          mov     al,'@'
          call    setter
          mov     al,rbuf
          call    setter                  ; inflexion
          mov     al,20h
          call    setter
          lea     bx,fox                  ; test message
          call    again
def:      lea     bx,default
          call    again
          mov     ah,00                   ; wait for y/n
          int     45h
          cmp     al,'y'                  ; change
          jz      change
          cmp     al,'Y'
          jz      change
          cmp     al,'n'                  ; no
          jnz     no
          jmp     nowrite
no:       cmp     al,'N'
          jnz     def
nowrite:mov       no_write,01
          ret
normal:   mov     normal_flag,01
          ret
change:   lea     bx,mode
          call    again
          mov     ah,00                   ; wait for response
          int     45h
          cmp     al,0dh                  ; exit code ?
          jnz     m1
          jmp     normal
m1:       cmp     al,'i'                  ; rate
          jz      rate
          cmp     al,'I'
          jz      rate
          cmp     al,'r'                  ; inflexion
          jz      inflex
          cmp     al,'R'
          jz      inflex
          jmp     change                  ; wrong key stroke
rate:     mov     rate_flag,01
          call    quit
          cmp     ah,48h                  ; increase
          jz      plus
          cmp     ah,50h
          jz      minus
          cmp     al,0dh
          jnz     m2
          jmp     normal
m2:       cmp     al,'r'
          jz      inflex
          cmp     al,'R'
          jz      inflex
          jmp     rate

plus:     cmp     rbuf,37h                  ; increase value
          jz      limit
          inc     rbuf
pmout:    mov     al,'@'                    ; speech control char.
          call    setter
          mov     al,rbuf                   ; inflexion character
          call    setter
          mov     al,20h
          call    setter
          jmp     rate

limit:    call    spkron                    ; boundry
          jmp     rate

minus:    cmp     rbuf,30h
          jz      limit                     ; decrease value
          dec     rbuf
          jmp     pmout
inflex:   mov     infl_flag,01
          call    quit                      ; purge buffer,
          cmp     ah,48h                    ; speak with new settings
          jz      iplus
          cmp     ah,50h
          jz      iminus
          cmp     al,0dh                    ; exit ?
          jnz     m3
          jmp     normal

m3:       cmp     al,'i'                    ; rate changes
          jnz     rorl
          jmp     rate
rorl:     cmp     al,'I'                    ; both cases handled
          jnz     r2
          jmp     rate
r2:       jmp     inflex

iplus:    cmp     ibuf,30h                  ; increase value
          jz      ilimit
          inc     ibuf
bbb:      mov     al,'@'                    ; new settings
          call    setter
          mov     al,'R'                    ; rate modes set here
          call    setter
          mov     al,ibuf
          call    setter
          mov     al,20                     ; first character
          call    setter
          jmp     inflex

iminus:   cmp     ibuf,31h
          jz      ilimit                    ; decrease value
          dec     ibuf
```

```
        jmp     bbb

ilimit: call    spkron
        jmp     inflex                  ; lower boundry

mode_set endp


outer   proc    near

; Main transmit routine

        push    dx
        push    ax
        push    bx
        push    cx                      ; save registers
        push    ds
        push    ax
        push    cs
        pop     ax
        mov     ds,ax                   ; change to new data segment
        pop     ax                      ;
        xor     cl,cl                   ;
        cmp     alt_shift0,01           ; ignore code ?
        jnz     bl                      ; pass, if not set
pass:   jmp     ignore
bl:     cmp     hold_flag,01            ; start transmit code
        jnz     nl
        jmp     ignore
nl:     call    line                    ; check for number of spaces
        cmp     spacer,02
        jnc     pass
        cmp     al,0dh
        jz      p4                      ; out if return
        cmp     al,'!'                  ; check PSS command initiators
        jz      passl
        cmp     al,'@'                  ; @ and !
        jz      passl
        cmp     al,2lh                  ; less than 20h
        jnc     upper                   ; no, check upper range
        jmp     ignore                  ; exit if less
passl:  cmp     al,'!'
        jnz     at
        lea     bx,exclaim              ; speak
        call    again
        jmp     pass
at:     lea     bx,aat                  ; @
        call    again
        jmp     pass                    ; exit
upper:  cmp     al,7fh                  ; greater or equal must exit
        jc      p4
        jmp     ignore                  ; bypassed
```

```
p4:     mov     dx,comm_port          ; selected RS-232 card
        call    stat                  ; check for XON/XOFF
punct:  push    bx
        lea     bx,period
        cmp     al,2eh                ; select punctuations are
        jz      outpun                ; spoken here
        lea     bx,comma
        cmp     al,2ch
        jz      outpun
        lea     bx,colen              ; , . ; are examples
        cmp     al,3ah
        jz      outpun
        lea     bx,semic
        cmp     al,3bh
        jz      outpun
        cmp     alt_shift4,00         ; all punctuations ?
        jz      proced
        pop     bx
        jmp     punct1
outpun: jmp     aoutpun
proced: lea     bx,quote              ; define other punctuations
        cmp     al,22h
        jz      outpun
        lea     bx,openq
        cmp     al,27h
        jz      outpun
        lea     bx,endq
        cmp     al,60h
        jz      outpun
        lea     bx,openb
        cmp     al,7bh
        jz      aoutpun
        lea     bx,closeb
        cmp     al,7dh
        jz      aoutpun
        lea     bx,opens
        cmp     al,5bh
        jz      aoutpun
        lea     bx,closes
        cmp     al,5dh
        jz      aoutpun
        lea     bx,openbr
        cmp     al,28h
        jz      aoutpun
        lea     bx,closebr
        cmp     al,29h
        jz      aoutpun
        lea     bx,tilde
        cmp     al,7eh
        jz      aoutpun
        lea     bx,carrat
        cmp     al,5eh
        jz      aoutpun
        lea     bx,backslash
        cmp     al,5ch
        jz      aoutpun
        lea     bx,minusc
        cmp     al,2dh
        jz      aoutpun
        lea     bx,uscore
        cmp     al,5fh
        jz      aoutpun
        lea     bx,quest
        cmp     al,3fh
        jnz     punctl1
aoutpun:call    again                   ; this will transmit
        pop     bx
        jmp     l2
punctl1:pop     bx
punctl: call    setter
        cmp     al,'>'
        jnz     l2
        mov     al,0dh
        jmp     punctl                  ; to speak DOS prompt
l2:     cmp     alt_shiftl,01           ; word/letter ?
        jz      ignore
        mov     al,0dh                  ; letter mode
        call    setter
ignore: pop     ds                      ; restore segments
        pop     cx
        pop     bx
        pop     ax
        pop     dx
        ret                             ; exit

outer   endp


stat    proc    near

; this handles the XON/XOFF protocol

        push    ax
        push    bx
        push    cx
        cli                             ; clear interrupts
        mov     hold_flag,00            ; clear flag
hold:   mov     ah,03                   ;
        int     dosint14                ; read serial port
        test    ah,01
        jz      stat1
        mov     ah,02
        int     dosint14                ; test for set/reset
        cmp     al,93h                  ; conditions
        jz      set
        cmp     al,11h                  ; Ctrl S and Ctrl Q codes
        jz      stat3
```

```
stat1:  cmp     hold_flag,01            ; hold flag set ?
        jz      hold
        sti
        pop     cx
        pop     bx                      ; exit
        pop     ax
        ret
stat2:  cmp     hold_flag,01h
        jnz     set
stat3:  mov     hold_flag,00            ; reset flag.
        call    spkron                  ;
        jmp     stat1
set:    mov     hold_flag,01h           ; set flag
        call    spkron
        jmp     stat1

stat    endp


nonint  proc    near

; Static mode

nonin:  cmp     al,'a'                  ; set markers
        jz      setfun
        cmp     al,'A'                  ; upper case
        jz      setfun
        jmp     nofun
setfun: mov     fun_flag,01             ; set flag
        call    setter
        mov     al,0dh
        call    setter
        call    spkron                  ; ready
        call    rotl                    ; get cursor position
        mov     ah,0
        int     45h                     ; wait for fun key
        cmp     ah,59                   ; F1-1
        jc      nofun
        cmp     ah,69                   ; F10+1
        jnc     nofun                   ; quit for wrong keys
        cmp     ah,68                   ; F10
        jz      pf10
        push    ax
        mov     al,'f'
        call    setter                  ; speak f
        pop     ax
        sub     ah,58
        push    ax                      ; fix offset for key
        add     ah,30h
        mov     al,ah
        call    setter
        mov     al,0dh
```

```
         call    setter
         pop     ax
         mov     bx, offset fun_key1     ; base
fixadd:  dec     ah
         jz      ranfun
         inc     bx                      ; next key buffer
         inc     bx
         jmp     fixadd
ranfun:  call    rotl                    ; save location
         mov     [bx],dl
         inc     bx
         mov     [bx],dh
         jmp     boock
pf10:    mov     al,'f'
         call    setter
         mov     al,31h
         call    setter
         mov     al,30h
         call    setter
         mov     al,0dh
         call    setter
         mov     bx,offset fun_key0
         jmp     ranfun
nofun:   cmp     al,'r'                  ; 'R'
         jz      nr                      ; next letter
         cmp     al,'R'
         jnz     nword
nr:      mov     al,'r'
         call    setter
         mov     al,0dh
         call    setter
         call    rotl
         mov     read_screen,01          ; set flag
         mov     dx,dumb_saver
         mov     high_range,dh           ; initialize ranges
         mov     low_range,dh
         call    rot2                    ; set cursor to last exit
         jmp     rl                      ; location
nword:   cmp     al,'c'                  ; 'C'
         jnz     noon                    ; console echo
         mov     al,'c'
         call    setter
         mov     al,0dh
         call    setter
         cmp     con_echo,01             ;
         jnz     cset                    ; set/reset console echo
         mov     con_echo,00
         jmp     boock
cset:    mov     con_echo,01
         jmp     boock
noon:    cmp     al,'q'                  ; 'Q'
         jnz     noom
         mov     al,'Q'                  ; quit speech
         call    setter
         mov     al,0dh
```

```
        call    setter
        cmp     alt_shift0,00           ; reset flag
        jz      qset
        mov     alt_shift0,00
        jmp     boock
qset:   mov     alt_shift0,01           ; set flag
        call    quit1
        jmp     boock
noom:   cmp     al,'w'                  ; 'W', read in words
        jnz     noop
        mov     al,'W'
        call    setter
        mov     al,0dh
        call    setter
        mov     read_screen,01
        mov     word_flag,01            ; set word flag
        mov     alt_shift1,01
        jmp     boock
noop:   cmp     al,'l'                  ; 'L', read letters
        jnz     nooq
        mov     al,'L'
        call    setter
        mov     al,0dh
        call    setter
        mov     word_flag,00            ; reset word flag
        mov     read_screen,01
        mov     alt_shift1,00           ; set letter flag
        jmp     boock
nooq:   cmp     al,'p'                  ; 'P', punctuations
        jnz     noorx
        mov     al,'P'
        call    setter
        mov     al,0dh
        call    setter
        cmp     alt_shift4,01           ; select punctuations mode
        jnz     spun
        mov     alt_shift4,00           ; set punctations
        jmp     boock
spun:   mov     alt_shift4,01
        jmp     boock
noorx:  cmp     al,'s'
        jz      space1                  ; all/no spaces
        cmp     al,'S'
        jnz     noor
space1: call    setter
        mov     al,0dh
        call    setter
        cmp     alt_shift3,00           ; space flag
        jz      spset
        mov     alt_shift3,00           ; reset
        jmp     boock
spset:  mov     alt_shift3,01
        jmp     boock
noor:   cmp     al,'h'                  ; 'H', help menu
        jnz     nooj
        mov     al,'H'
        call    setter
        mov     al,0dh
        call    setter
        call    helper
        jmp     boock
nooj:   cmp     al,'i'
        jz      ident                   ; read current line
        cmp     al,'I'
        jnz     noos                    ; no, next code
ident:  call    setter
        mov     al,0dh
        call    setter
        mov     dx,dumb_saver
        mov     high_range,dh
        mov     low_range,dh
        mov     ch,dh                   ; store in end location
        mov     spacer,00
        mov     ah,15
        int     int46                   ; get page number & columns on pc
        mov     flag_space,ah
        mov     cl,ah
        call    view                    ; call speak screen routine
        jmp     boock
noos:   cmp     al,'f'
        jz      ppl                     ; string search
        cmp     al,'F'
        jnz     oos
ppl:    call    setter
        mov     al,0dh
        call    setter
        mov     bx,offset string_loc    ; start of buffer
sca:    mov     ah,0
        int     45h                     ; wait for key closure
        call    setter                  ; speak key
        push    ax
        mov     al,0dh
        call    setter
        pop     ax
        cmp     al,0dh                  ; exit code
        jz      fetch
        mov     [bx],al                 ; move to buffer
        mov     ah,00
        add     check_sum,ax            ; create check sum
        inc     string_count
        inc     bl                      ; next count
        jmp     sca
fetch:  call    getter                  ; end of string
        mov     ax,loc_screen
        mov     dumb_saver,ax           ; location to start
        mov     ax,check_sum            ; search for
        mov     cksum_save,ax           ; stiring
        mov     al,string_count
        mov     count_save,al           ; for multiple searches
        mov     loc_screen,00
```

```
         push     ax
         mov      ax,00                    ; reset sum
         mov      sum,ax
         mov      check_sum,ax
         mov      string_count,al          ; reset count
         pop      ax
         jmp      boock
oos:     cmp      ah,83                    ; Del key ?
         jnz      loos
         lea      bx,del                   ; delete message
         call     again                    ; invoke multiple searches
         mov      al,count_save
         cmp      al,00
         jnz      doock                    ; exit on Zero entry
         jmp      boock
doock:   mov      string_count,al          ; restore operations
         mov      dx,mult_try
         mov      loc_screen,dx            ; get old pointers
         mov      ax,cksum_save
         mov      check_sum,ax             ; simulate original conditions
         jmp      fetch
loos:    cmp      ah,71                    ; home key ?
         jnz      lsoo
         push     bx
         lea      bx,home
         call     again                    ; goto top of screen
         pop      bx
         mov      dx,00
         mov      dumb_saver,dx            ; top of page
         call     rot2
         jmp      boock
lsoo:    cmp      ah,79                    ; end key ?
         jnz      soos
         push     bx
         lea      bx,mes2
         call     again
         pop      bx
         mov      dx,1800h
         mov      dumb_saver,dx            ; end of page
         call     rot2
         jmp      boock
soos:    cmp      al,'+'                   ; speak row/column of cursor
         jz       soo2
         jmp      sool
soo2:    push     ax
         mov      al,'+'
         call     setter
         mov      al,0dh
         call     setter
         pop      ax
         call     rotl                     ; get cursor location
         call     rowcol                   ; call convert
         jmp      boock                    ; to ASCII
sool:    cmp      al,'m'                   ; merge cursors
         jz       soset
```

```
        cmp     al,'M'
        jnz     soo3
soset:  call    setter
        mov     al,0dh
        call    setter
        mov     dx,loc_saver
        mov     dumb_saver,dx           ; set all cursor pointers
        mov     high_range,dh           ; to the DOS cursor
        mov     low_range,dh
        call    rot2
        jmp     boock
soo3:   cmp     al,'k'
        jz      keyset                  ; keyboard echo
        cmp     al,'K'
        jnz     soo4
keyset: push    ax                      ;
        call    setter
        mov     al,0dh
        call    setter
        pop     ax
        cmp     key_echo,00
        jz      keye
        mov     key_echo,00             ; enable key echo
        mov     con_echo,01             ; diable console echo
        jmp     boock
keye:   mov     key_echo,01             ; disable key echo
        jmp     boock
soo4:   cmp     al,'v'                  ; mode set feature
        jz      sove
        cmp     al,'V'
        jnz     soo5                    ; set speech
sove:   call    setter
        mov     al,0dh
        call    setter
        call    mode_set
        jmp     boock
soo5:   cmp     read_screen,01          ; read screen mode
        jnz     boock
        mov     dx,dumb_saver           ; essential to invoke
        call    rot2                    ; screen reader functions
        call    rscrn                   ;
boock:  ret                             ; exit

rscrn:  cmp     ah,72                   ; up
        jnz     nnnn
        jmp     upline
nnnn:   cmp     ah,80                   ; down
        jnz     mmmm
        jmp     dnline
mmmm:   cmp     ah,77                   ; right
        jnz     mmmr
        jmp     wrdrt
mmmr:   cmp     ah,75                   ; left
        jnz     lt72
```

```
         jmp     wrdlt
lt72:    cmp     al,','                  ; comma
         jz      lt73
         cmp     al,0dh                  ; return
         jz      lt73
         cmp     al,30h
         jc      exc
         cmp     al,3ah                  ; highest + 1
         jc      lt73                    ; go to the service routine
exc:     jmp     or3                     ; else, exit
lt73:    mov     bx,offset bytes_store   ; key range buffer
         add     bl,byte_count           ; set buffer
         cmp     al,','                  ; comma
         jnz     retnn
         cmp     byte_count,00
         jz      rtnl
         push    ax
         mov     al,','                  ; terminator
         jmp     rder
retnn:   cmp     al,0dh                  ; return
         jnz     ordi
         cmp     byte_count,00           ; zero count
         jnz     rtn
rtnl:    ret                             ; just exit
rtn:     push    ax
         jmp     endee
ordi:    push    ax
rder:    call    outer
endee:   mov     al,0dh
         call    outer
         pop     ax
         mov     [bx],al                 ; get code in buffer
         inc     byte_count
         cmp     al,0dh                  ; return key
         jz      rnger
         mov     al,00
or1:     mov     dh,high_range           ; set curor to end of range
         dec     dh
         mov     dl,00
or2:     mov     dumb_saver,dx
         call    rot2                    ; reset ranges
or3:     ret
rnger:   mov     bx,offset bytes_store   ; begin processing
         cmp     byte_count,03           ; both ranges present ?
         jnle    boths
         mov     al,[bx]
         cmp     al,0dh                  ; just return or comma
         jz      or3
         cmp     al,','
         jz      or3
         mov     byte_count,00
         mov     cur_count,00            ; reset counts
         mov     termin,0dh
         call    mbx                     ; this works tat way
         mov     low_range,al
         jmp     yxxxx
boths:   mov     termin,','                      ; comma
         mov     cur_count,00
         mov     byte_count,00
         call    mbx                             ; form hex equivalents
         mov     al,high_range
         mov     low_range,al                    ; store in start location
         inc     byte_count
         mov     cur_count,00
urnge:   mov     termin,0dh                      ; return
         call    mbx
yxxxx:   mov     byte_count,00
         mov     cur_count,00                    ; check for valid ranges
         cmp     high_range,00
         jnz     lornge
         jmp     eerl
lornge:  cmp     low_range,00
         jnz     look
         jmp     eerl                            ; valid lower range
look:    mov     al,high_range
         cmp     al,low_range                    ; to check ranges
         jnc     aoky
eerl:    lea     bx,errorl                       ; error message
         call    again
or2go:   jmp     or2
aoky:    mov     ch,high_range                   ; store in end location
         mov     spacer,00
         mov     ah,15
         int     int46                           ; get page number & columns on pc
         mov     flag_space,ah
         mov     cl,ah
         dec     ch
         dec     low_range
         call    view                            ; call speak screen routine
         mov     dh,high_range
         mov     dl,00
         jmp     or2

upline:  mov     dx,dumb_saver                   ; cursor up
         cmp     dh,00                           ;
         jnz     hr                              ; read the entire line
         call    spkron                          ; above the current line
         jmp     rl
hr:      mov     high_range,dh
         dec     high_range                      ; fix ranges
         mov     al,high_range
         mov     low_range,al
         mov     ch,high_range                   ; store in end location
         mov     spacer,00
         mov     ah,15
         int     int46                           ; get page number & columns on pc
         mov     flag_space,ah
         mov     cl,ah
         call    view                            ; call speak screen routine
```

```
        mov     dl,00
        dec     dh
        mov     dumb_saver,dx           ; update cursor pointer
        jmp     rl

dnline: mov     dx,dumb_saver           ; this will speak the line
        cmp     dh,24                   ; below the curent cursor
        jnz     dn
        call    spkron                  ; limits create a beep
        jmp     rl
dn:     mov     high_range,dh
        inc     high_range              ; increase count
        mov     al,high_range
        mov     low_range,al
        mov     ch,high_range           ; store in end location
        mov     spacer,00
        mov     ah,15
        int     int46                   ; get page number & columns on pc
        mov     flag_space,ah
        mov     cl,ah
        call    view                    ; call speak screen routine
        mov     dl,00
        inc     dh
        mov     dumb_saver,dx           ; update pointer
        jmp     rl


wrdrt:  cmp     word_flag,00            ; word right
        jz      letter                  ; cursor right key
        call    spwdrt
        jmp     rl
letter: call    spltrt                  ; letter right
        jmp     rl

wrdlt:  cmp     word_flag,00            ; word left
        jz      letlft
        call    spwdlt
        jmp     rl
letlft: call    spltlt                  ; letter left
        jmp     rl

rl:     mov     dx,dumb_saver           ; exit area
        call    rot2                    ; update again
        ret

spwdrt: call    rotl                    ; speak the word till
        cmp     dh,24                   ; a space is encounterd
        jnz     colcnt
        cmp     dl,79
        jnz     incur                   ; then prefix to the
```

```
spkwrl: call    spkron                  ; start of the word
        mov     dx,2479                 ; bottom of screen
        mov     dumb_saver,dx
        call    rot2_                   ; fix cursor to the bottom
        ret
colcnt: cmp     dl,79                   ; next line ?
        jz      roinc
incur:  call    rot4                    ; get char at cur. loc.
        cmp     al,20h                  ; word ?
        jz      spkwrd
        inc     dl
        cmp     dl,80
        jnz     www                     ; end of line ?
roinc:  inc     dh
        cmp     dh,25
        jz      spkwrl                  ; end of screen ?
        mov     dl,00
www:    mov     dumb_saver,dx           ; no, start at begining of
        call    rot2_                   ; next line
        jmp     incur
spkwrd: mov     al,0dh
        call    setter
        call    adjust                  ; locate to begining
        ret


spwdlt: call    rot1                    ; get cursor position
        cmp     dh,00
        jnz     top
        cmp     dl,00
        jz      top3
top:    call    adjlt                   ; word left
        mov     bx,offset buffer        ; store in buffer
        mov     cl,01
decur:  call    rot5                    ; get char at cur. loc.
        cmp     al,20h
        jz      sendl
        mov     [bx],al
        inc     bx                      ; next letter
        inc     cl
        dec     dl
        cmp     dl,00                   ; check row/col boundries
        jnz     grt
        cmp     dh,00
        jz      send
        dec     dh
        cmp     dh,00
        jz      topl
        mov     dl,79
grt:    call    rot2
        jmp     decur
sendl:  dec     bx                      ; up 1 row
        dec     cl
send:   mov     al,[bx]                 ; transmit
        call    outer
        dec     bx
        dec     cl
        cmp     cl,00
        jz      top2
        jmp     send
top3:   call    spkron                  ; top of screen, warning
        ret
top2:   mov     al,0dh
        call    setter
        call    adjlt
        mov     dumb_saver,dx           ; save location
        call    rot2
top1:   ret

spltrt: call    rot5                    ; speak letter right
        call    case                    ; to check case
        call    checkup                 ; check screen boundries
        cmp     al,20h
        jnz     splx
        push    bx
        lea     bx,space                ; speak spaces too
        call    again
        pop     bx
splx:   ret

spltlt: call    rot5                    ; get cursor position
        call    case                    ; check case
        call    checkdn                 ; check screen boundries
        cmp     al,20h
        jnz     sprx
        push    bx
        lea     bx,space                ; speak spaces
        call    again
        pop     bx
sprx:   ret

nonint  endp


case    proc    near

; check for upper case letters

        cmp     al,41h
        jc      calout                  ; call outer
        cmp     al,5bh
        jnc     calout                  ; lower case
        push    bx
        push    ax
        lea     bx,caps                 ; 'CAPS'
        call    again
        pop     ax
        pop     bx
calout: call    outer                   ; letter
        ret

case    endp


value   proc    near

; convert hex digits to ASCII outputs

        push    ax
        push    bx                      ; save registers
        push    cx
        push    dx
        mov     cx,00                   ; initialize
        mov     dl,00
        mov     dh,al
        cmp     al,0ah                  ; lowest
        jc      lo9
        cmp     al,10h
        jnc     up15
        sub     dh,0ah
        add     dh,30h                  ; ASCII
        mov     al,31h
        call    setter
        mov     al,dh
        jmp     conex
up15:   mov     cl,04                   ; greater than 0fh
        mov     dh,al                   ; save parameter passed in al
        and     al,0f0h                 ; get upper byte
        ror     al,cl
        mov     cl,al
        add     al,00
        daa                             ; decimal adjust
        cmp     cl,00
        jz      secnd
dcml:   add     al,15h                  ;
        daa
        dec     cl
        jnz     dcml
        mov     dl,al                   ; save
secnd:  and     dh,0fh
        mov     al,dh
        add     al,00                   ; decimal adjustments
        daa
        add     al,dl
        daa
        mov     ch,al
        and     al,0f0h
        mov     cl,04               ; multiple digits
        ror     al,cl
        add     al,30h              ; individually converted
        cmp     al,30h
        jz      upr
        call    setter
upr:    mov     al,ch               ; spoken here
        and     al,0fh
lo9:    add     al,30h              ; ASCII offset
conex:  call    setter
        mov     al,0dh
        call    setter
        pop     dx
        pop     cx                  ; restore registers
        pop     bx
        pop     ax
        ret

value   endp


helper  proc    near

; all current settings spoken here

        push    bx
        cmp     alt_shift1,00       ; word flag
        jz      letmode
        lea     bx,alt1on           ; word mode
nxt1:   call    again
        cmp     alt_shift4,00       ; punctuations
        jz      allpun
        lea     bx,alt4on
nxt2:   call    again
        cmp     alt_shift3,00       ; spaces
        jz      allsp
        lea     bx,alt3off
hlpx:   call    again
hlpxyz: pop     bx
        ret
letmode:lea     bx,alt1off          ; flag name
        jmp     nxt1                ; followed by flag status
allpun: lea     bx,alt4off
        jmp     nxt2
allsp:  lea     bx,alt3on
        jmp     hlpx
mark:   mov     bx,offset fun_key1  ; marker locations
        push    dx                  ; spoken here
        mov     cl,31h
ark:    mov     al,'f'
        call    setter
        mov     al,cl
        call    setter
        mov     al,0dh                          ; row/col information
        call    setter
        mov     dl,[bx]
        inc     bx
        mov     dh,[bx]
        inc     bx
        push    cx
        call    rowcol                          ; speak row/col
        pop     cx
        inc     cl
        cmp     cl,3ah
        jnz     ark
        mov     al,'f'                          ; special case f10
        call    setter
        mov     al,'1'
        call    setter
        mov     al,'0'
        call    setter
        mov     al,0dh
        call    setter                          ; send row number
        mov     bx,offset fun_key0
        mov     dl,[bx]
        inc     bx
        mov     dh,[bx]
        call    rowcol                          ; send col number
        pop     dx
        jmp     hlpxyz

helper  endp


rowcol  proc    near

; take the hex row/col number from dx and speak decimal equivalents

        push    dx
        push    bx
        lea     bx,rownum                       ; row number
        call    again
        pop     bx
        mov     al,dh                           ; send row number
        inc     al
        call    value                           ; convert to ASCII
        push    bx
        lea     bx,colnum                       ; column number
        call    again
        pop     bx
        pop     dx
        mov     al,dl                           ; send column number
        inc     al
        call    value                           ; ASCII
        ret
rowcol  endp


adjust  proc    near

; fix to begining of next word

        push    ax
        call    rot1                    ; get cursor
        cmp     dh,24
        jnz     fine                    ; check screen boundries
        cmp     dl,79
        jnz     fine                    ; okay
        call    rot2
        jmp     ound                    ; error
fine:   call    rot5                    ; retn char. in al, do not send
        cmp     al,20h                  ; space character
        jnz     found
        inc     dl                      ; next location
        cmp     dl,80                   ; end of screen
        jnc     newrow
        mov     dumb_saver,dx
        call    rot2                    ; save
        jmp     fine
newrow: inc     dh                      ; new row
        cmp     dh,25                   ; row limit
        jz      ound
        mov     dl,00                   ; start of line
        call    rot2
        mov     dumb_saver,dx
        jmp     fine
ound:   call    spkron                  ; border, warning
        mov     dx,184eh                ; fix cursor to bottom of screen
found:  mov     dumb_saver,dx
        call    rot2
        pop     ax                      ; exit
        ret

adjust  endp


adjlt   proc    near

; move from left to right, ie., locate start of word

        push    ax                      ; save registers
        push    bx
        push    cx
        call    rotl                    ; get cursor
        cmp     dl,00
        jnz     fin                     ; check boundries
        cmp     dh,00                   ; top of screen
        jnz     dcrdh
        jmp     ond                     ; error
dcrdh:  dec     dh                      ; upper line
        mov     dl,80
        call    rot2                    ; set cursor
        mov     dumb_saver,dx
fin:    call    rot5_                   ; retn char. in al, do not send
        cmp     al,20h                  ; space
        jnz     fond
        dec     dl                      ; col = col - 1
        cmp     dl,00                   ; end of screen
        jz      newrw
        call    rot2
        mov     dumb_saver,dx           ; save new location
        jmp     fin_
newrw:  cmp     dh,00                   ; top of row
        jz      ond
        dec     dh
        mov     dl,80
        call    rot2                    ; upper line
        mov     dumb_saver,dx
        jmp     fin_
ond:    call    spkron                  ; warning
fond:   mov     dumb_saver,dx
        pop     cx
        pop     bx
        pop     ax                      ; restore registers
        ret

adjlt   endp


checkup proc    near

; check current location for top of screen

        call    rotl                    ; to check border of screen.
        cmp     dl,80
        jnz     ckone                   ; not over
        cmp     dh,24
        jnz     cko
        call    spkron                  ; over, warning
outck:  mov     dumb_saver,dx
        call    rot2_                   ; save and exit
        ret
cko:    inc     dh                      ; else, next row
        mov     dl,00
        jmp     outck
ckone:  inc     dl                      ; else, next row
        jmp     outck

checkup endp


checkdn proc    near

; check for screen top

        call    rotl                    ; to check lower limit on screen.
        cmp     dl,00
        jnz     cdone
        cmp     dh,00
        jnz     cdo                     ; warning
        call    spkron
outdn:  mov     dumb_saver,dx
        call    rot2                    ; set cursor
        ret
cdo:    dec     dh
        mov     dl,80
        jmp     outdn                   ; next
cdone:  dec     dl
        jmp     outdn

checkdn endp


getter  proc    near

; get first match of string

        mov     dx,loc_screen           ; get screen location
        mov     dumb_saver,dx           ; set cursor
        call    rot2
ref:    mov     bx, offset string_loc   ; buffer head
        mov     cl,[bx]
        call    rot5                    ; read character
        cmp     al,cl
        jnz     chno                    ; match ?
        cmp     al,20h                  ; leading space ?
        jnz     nospace
        call    adjust                  ; align to next letter
        dec     dl
        call    rot2                    ; set back to start of string
nospace:call    rotl
        mov     loc_screen,dx           ; save location
        cmp     string_count,01
        jz      cdl
        call    summer                  ; compute check sum
        call    rotl
        mov     mult_try,dx             ; save end location
        sub     ax,check_sum
        jz      cdd
        mov     dx,loc_screen
        jmp     chno
cdd:    push    bx
        push    di
        push    dx
        mov     bx, offset string_loc   ; reference
        mov     di, offset sum_string   ; screen string
        mov     dh,string_count
        inc     dh                      ; to pass all
scheck: cmp     dh,00                   ; end of string ?
        jz      spass
        dec     dh                      ; keep count
        mov     al,[bx]                 ; string check
        mov     dl,[di]
        inc     bx
        inc     di                      ; set to next locations
        cmp     al,dl                   ; tally ?
        jz      scheck
        pop     dx
        pop     di                      ; wrong string
        pop     bx
        jmp     chno
spass:  pop     dx                      ; tally ho !
        pop     di
cdl:    mov     dx,loc_screen
        mov     dumb_saver,dx
        call    rot2                    ; set cursor pointers
        call    spkron                  ; Beep
        lea     bx,string_found
        jmp     eeed
chno:   cmp     dl,80
        jnz     sset
        cmp     dh,25
        jnz     ssset
        push    dx
        mov     dx,0000
        mov     mult_try,dx             ; wrap back to first occurence
        pop     dx
        push    bx
        lea     bx,not_found
        mov     dx,00
        call    rot2
eeed:   call    again
        pop     bx
        ret
ssset:  inc     dh
        mov     dl,00
        jmp     jk
sset:   inc     dl
```

```
jk:     mov     loc_screen,dx
        call    rot2                    ; set cursor to next byte
        jmp     ref

getter  endp


summer  proc    near

; create chech sum for found string

        push    cx
        push    bx
        mov     bx, offset sum_string   ; check string
        mov     cl,string_count         ; string size
sumb:   mov     ah,00
        mov     [bx],al
        inc     bx                      ; next
        add     sum,ax                  ; sum
        cmp     dl,80
        jnz     ffff                    ; ranges
        cmp     dh,25
        jz      fff0
        mov     dl,00
        inc     dh
        jmp     jja
ffff:   inc     dl
jja:    push    cx                      ; set cursor to next location
        call    rot2
        call    rot5                    ; read character
        pop     cx
        dec     cl                      ; decrement count
        jnz     sumb                    ; go back
fff0:   pop     bx
        mov     ax,sum                  ; result in ax
        mov     cx,00
        mov     sum,cx                  ; reset sum
        pop     cx
        ret

summer  endp


next    proc    near

; fixed message output

        push    ax
ext:    mov     al,[bx]                 ; buffer top
```

```
        cmp     al,'$'                  ; terminator
        jz      k2
        cmp     al,' '                  ; space ?
        jnz     cont0
        mov     al,0dh                  ; purge buffer on spaces
cont0:  call    setter
        inc     bx
        jmp     ext
k2:     mov     al,0dh                  ; speak
        call    outer
        pop     ax
        ret

next    endp


spkron  proc    near

; turn the speaker ON
        push    ax
        push    bx                      ; routine from DOS
        push    cx                      ; technical reference
        mov     bx,80h
        in      al,61h                  ; hardware reference manual
        push    ax
k65:    and     al,0fch
        out     61h,al
        mov     cx,48h
k66:    loop    k66
        or      al,02
        out     61h,al
        mov     cx,48h
k67:    loop    k67
        dec     bx
        jnz     k65
        pop     ax
        out     61h,al
        pop     cx
        pop     bx
        pop     ax
        ret

spkron  endp


video   proc    far

; video interrupt

        sti                             ; enable interrupts
```

```
          push    ds
          push    es
          push    ax
          push    cs                        ; save registers
          pop     ax
          mov     ds,ax
          pop     ax                        ; prefix to user code area
          cmp     static_exit,01            ; is flag set
          jnz     nstatic                   ; set for static mode
          mov     static_exit,00
          pop     es                        ; no console operations
          pop     ds
          iret
nstatic:cmp       con_echo,01               ; console echo ?
          jnz     prod                      ; no
          jmp     cont01
prod:     cmp     alt_shift0,01             ; return to normal DOS
          jnz     alcur                     ; silent mode
          jmp     cont01
alcur:    cmp     al,0dh                    ; return compensation
          jnz     gon
          cmp     cret_flag,01
          jz      gon
          push    ax
          call    setter
          pop     ax
gon:      cmp     ah,09h                    ; allow write codes only.
          jc      cont01
          cmp     ah,0bh
          jz      cont01                    ; graphic reads & status
          cmp     ah,0dh                    ; codes exit here
          jz      cont01
          cmp     ah,14
          jz      cont01
          cmp     ah,0fh
          jnz     cont
cont01:   pop     es                        ; restore segments
          pop     ds
          jmp     cs:video_int              ; DOS routine

cont:     pop     es                        ; codes written on the
          pop     ds                        ; screen will be spoken
          int     int46
          push    ax
          push    bx
          push    cx
          push    dx
          push    ds
          push    ax
          push    cs                        ; change segments
          pop     ax
          mov     ds,ax
          pop     ax
x10:      mov     ah,new_row
```

```
        mov     old_row,ah              ; new row ?
        mov     ah,15
        int     int46
        mov     flag_space,ah           ; get # of colums on display
        mov     ah,03
        int     int46
        mov     new_row,dh
        cmp     dh,old_row
        jz      x3
        mov     al,0dh                  ; new row, clear votrax
        call    outer
x3:     mov     cret_flag,00
        pop     ds
        pop     dx
        pop     cx                      ; restore registers
        pop     bx
        pop     ax
        push    ds
        push    ax
        push    cs
        pop     ax
        mov     ds,ax
        pop     ax
        mov     ah,08                   ; read character
        int     int46
        cmp     al,00
        jnz     nex
        mov     al,20h
nex:    call    outer                   ; speak
        cmp     alt_shiftl,01
        jz      exit                    ; word ?
        mov     al,0dh
        call    outer
exit:   pop     ds
        iret

video   endp


rotl    proc    near

        push    ax                      ; Read row/column of cursor
        push    bx
        push    cx
        push    di
        push    si
        push    ds
        mov     ax,0050h
        mov     ds,ax
        mov     ah,03                   ; DOS data segment area
        mov     bh,00
        int     int46
        pop     ds
        pop     si
        pop     di                      ; restore registers
        pop     cx
        pop     bx
        pop     ax
        ret

rot1    endp


rot2    proc    near

        push    ax                      ; Set row/col of cursor.
        push    bx
        push    ds
        mov     ax,0050h
        mov     ds,ax
        mov     ah,15
        int     int46
        mov     ah,02                   ; set function
        int     int46
        pop     ds
        pop     bx
        pop     ax
        ret

rot2    endp


rot3    proc    near

        push    ax
        call    rot5                    ; Read character, send return
        call    outer
        mov     al,0dh
        call    outer                   ; speak code
        pop     ax
        ret

rot3    endp

rot4    proc    near

        call    rot5                    ; Read char., send to votrax
        call    outer
        ret

rot4    endp


rot5    proc    near

        push    bx                      ;Read char., return in al.
        push    cx
        push    dx
        push    ds
        mov     ax,0050h
        mov     ds,ax
        mov     ah,15
        int     int46
        mov     ah,08                   ; read function
        int     int46
        pop     ds
        pop     dx
        pop     cx
        pop     bx
        ret

rot5    endp

again   proc    near

; standard routine to transmit messages with spaces

        push    ax
gain:   mov     al,[bx]                 ; buffer top
        cmp     al,'$'                  ; terminator
        jz      over
aaa:    call    setter
        inc     bx
        jmp     gain                    ;
over:   mov     al,0dh                  ; speak
        call    setter
        pop     ax
        ret

again   endp


setter  proc    near

; actual transmission to the serial port on the pc
        push    dx
        mov     dx,comm_port            ; selected RS-232 card.
        mov     ah,01
        int     dosint14                ; DOS interrupt
        pop     dx
        ret

setter  endp


line    proc    near
```

```
; check for the number of spaces

         push     ax
         cmp      al,20h
         jnz      line2
line1:   inc      spacer                   ; increment space count
         cmp      spacer,01
         jnz      nu                       ; pause for 1 space
         call     setter
         mov      al,0dh
         call     setter
exi:     pop      ax
         ret                               ; exit
nu:      mov      ah,50h
         cmp      spacer,ah
         jnz      exi1
         push     bx
         lea      bx,blank
         call     again
         pop      bx
line2:   mov      spacer,00
         jmp      exi
exi1:    cmp      alt_shift3,00            ; default, no spaces
         jz       exi
         push     bx
         lea      bx,space
         call     again
         pop      bx
         jmp      exi

line     endp


mbx      proc     near

; form hex ranges from the input and terminators

mbx1:    mov      bx,offset bytes_store
         add      bl,byte_count            ; set buffer
         mov      al,[bx]
         mov      cl,termin
         cmp      al,cl                    ; terminator
         jz       innex
         inc      byte_count               ; next code
         inc      cur_count
         jmp      mbx1
innex:   cmp      cur_count,01             ; single digit
         jz       lobyte
         sub      bl,cur_count             ; top of buffer
```

```
        mov     al,[bx]
        sub     al,30h
        cmp     al,00h                  ; to take care of leading zeros
        jz      set0
        mov     dl,10                   ; tens
        imul    dl                      ; form tens digit
        inc     bx
        mov     ah,[bx]                 ; lower byte
        sub     ah,30h
        cmp     ah,00h                  ; to take care of trailing zeros
        jz      fini
        add     al,ah                   ; form decimal range
fini:   mov     high_range,al
        cmp     high_range,26           ; to avoid lines above 25
        jnc     set0
        jmp     rrr
lobyte: dec     bl
        mov     al,[bx]
        sub     al,30h
        cmp     al,00h                  ; to take care of leading zeros
        jnz     nozro
set0:   mov     high_range,00
        jmp     rrr
nozro:  mov     high_range,al
rrr:    ret

mbx     endp


view    proc    near

; speak the ranges- lo_range to hi_range

        push    cx
        mov     ah,03
        int     int46                   ; dx - row/column of cursor
        pop     cx                      ; parameters passed through
        push    dx                      ; high_range & low_range
        mov     dl,00
        mov     dh,low_range
        push    ds
        mov     ax,0050h
        mov     ds,ax
alt53:  mov     ah,02                   ; set cursor to top of screen
        int     int46
        mov     ah,08h                  ; read character
        int     int46
        or      al,al
        jnz     alt51
        mov     al,20h
alt51:  call    outer
nin:    inc     dl
        cmp     cl,dl
        jnz     alt53
        xor     dl,dl
        inc     dh
        cmp     ch,dh
        jnc     alt53                   ; to include present line also
alt52:  pop     ds
        pop     dx
        mov     ah,02
        int     int46                   ; read character
        ret

view    endp


quit    proc    near

; speak test message, kill last speech

        lea     bx,fox                  ; test message
        call    again
        mov     ah,00
        int     16h                     ; wait for key closure
        call    quit1
        ret

quit    endp


quit1   proc    near

; purge PSS buffer

        push    ax
        mov     al,1bh                  ; escape code
        call    setter
        mov     al,'Q'                  ; kill code
        call    setter
        mov     al,'.'                  ; terminator
        call    setter
        pop     ax
        ret

quit1   endp


a1:     mov     ah,49h                  ; free allocated memory
        int     dosint21
        mov     dx,10240                ; 10k residency
        mov     ax,3101h                ; function call
        int     dosint21                ; exit

my_code ends

end main
```

The vita has been removed from the scanned document