TABLE OF CONTENTS

TABLE OF CONTENTS, cont...

INTRODUCTION

XBEA is a set of enhancements added to version 110 of TI EXTENDED BASIC and the TI EDITOR/ASSEMBLER through the use of a GRAM KRACKER. These enhancements have been designed to aid the Extended Basic and/or assembly language programmer. Also included are modifications to the 99/4A operating system that clear up some problems encountered with the CorComp disk controller card, and add a true lower case character set.

The enhancements to Extended Basic include added commands to aid in program editing, and modifications to some existing commands to aid in program debugging. Also added is a disk catalog routine, and commands that allow you to PEEK and POKE to all 3 kinds of memory in the computer (CPU RAM, VDP RAM, and GROM/GRAM). Another new command permits entering the Editor/Assembler without having to go back to the title screen, provided you choose the combined Extended Basic and Editor/Assembler version. Most of the new features can only be used in command mode.

Several improvements have been made to the Editor/Assembler including repeating keys during filename entry, filename recall at all prompts, and support for the MYARC ramdisk card. The Editor and Assembler programs are now stored in GRAM for quicker loading, and you may enter Extended Basic directly from the main Editor/Assembler menu. There is one drawback however. All support routines for console basic (CALL POKEV, etc.) are removed from this version of the Editor/Assembler.

System requirements are TI 99/4A console, 32K memory, disk controller and disk drive, version 110 of TI Extended Basic, TI Editor/Assembler, and an 80K GRAM KRACKER. A printer is highly recommended.

MAKING THE MODIFICATIONS

The XBEAPATCH program will make the necessary changes in your Extended Basic and/or Editor/Assembler modules to enable them with the features described in this documentation. Before running the XBEAPATCH program, you must prepare your GRAM KRACKER with the necesary modules. This is not complicated, but care must be taken to do the following steps in order. All page numbers given are references into the GRAM KRACKER manual. The patch program can produce two versions of the modifications, a combined Extended Basic and Editor/Assembler, or an Extended Basic only version. If you want the Extended Basic only version, skip any steps marked with an asterisk (*). The XBEAPATCH program also modifies your operating system in GRAM 0 to include a true lower case character set. If you do not want the operating system modifications, skip the steps marked with the pound sign (#) and leave the OpSys/GRAM 0 switch in the OpSys position while running the patch program.

SETTING UP THE GRAM KRACKER

1 - Save your Extended Basic (version 110 only) module to disk. (Page 8)

2#- Save console GROM 0 to disk. (Page 12)

3*- Save your Editor/Assembler module to disk. (Page 8)

4 - Initialize the GK module space. (Page 11)

5*- Load your saved Editor/Assembler module into the GRAM KRACKER. (Page 7)

6*- Run the E/A-MOVER program from the GRAM KRACKER utility disk (the disk that came with your GK). (Page 24) Move the Editor/Assembler to GRAM 7. DO NOT ADD THE BYTES AT g7FFC AS DIRECTED BY THE MANUAL AND THE E/A-MOVER PROGRAM!

7 - Load your saved Extended Basic module into the GRAM KRACKER. (Page 7)

8#- Load your saved console GROM 0 into GRAM 0 in the GK. (Page 12)

RUNNING THE XBEA PATCH PROGRAM

You are now ready to run the patch program. It is loaded using the 'LOAD MODULE' option of the GK menu. The Device.Filename is DSK1.XBEAPATCH, assuming that you have the patch program disk in drive #1. The patch program will run after you press the space bar when prompted. Once started, the only way to prematurely exit the patch program is with the GRAM KRACKER reset switch.

After the patch program introduction screen you are asked to select which version of the patch you want. After you make your selection you are reminded that the GRAM KRACKER must be set up as described in the preceeding section. The next screen will instruct you to configure the GRAM KRACKER switches. Be sure that all switches are set correctly before proceeding. The only switch check made by the patch program is for the BANK 2 switch. You are not warned if the other switches are not configured properly. DO NOT CHANGE ANY OF THE SWITCH SETTINGS UNTIL YOU ARE INSTRUCTED TO DO SO! If you get a 'Bank 2 not selected' error message and you do have the switch in the BANK 2 position it is an indication that either you have not correctly followed the set-up instructions or that the version of Extended Basic that you have loaded is not compatible with these modifications. In this case, start over with the set-up as described in the preceeding section. If after doing this you still get the error message you should try again using a different Extended Basic module.

If you have selected the combined Extended Basic and Edtior/Assembler version you will be instructed to place a disk containing the EDIT1, ASSM1, and ASSM2 files into drive one. Be sure that these are unmodified files. To be sure, use your original E/A disk.

After the modifications are complete, restore the write protect and exit the program. You should now save your modified program(s) to disk. Be sure to save the modified GRAM 0 also. If you have selected the combined Extended Basic and Editor/Assembler version you also have to save GRAMs 1 and 2. See pages 8 and 12 in the GK manual.

If you find that the new features of Extended Basic and/or the Editor/Assembler do not work as described in the following sections, redo the patch, starting with the GK setup. If this does not solve the problems it is an indication that your module(s) are not compatible with these modifications. Try again using a different module for the patch.

OPERATING SYSTEM MODIFICATIONS

The 99/4A operating system in GRAM 0 has been modified to include a true lower case character set that replaces the standard small capitals character set. The zero character has been slashed, and the upper case O character has been rounded. Any program or module that uses the standard character loader in the operating system will now have the new character set as long as the OpSys switch on the GRAM KRACKER is in the GRAM 0 position. Returning the switch to the OpSys position will allow a program or module to use the standard character set.

Some modules contain their own character sets, and therefore will not use the new set contained in the modified operating system. There are several programs, most notable the CorComp Disk Manager and the Millers Graphics MG EXPLORER, that do not use the standard character loader and as such will always load the old character set from GRAM 0. The new character set may also cause problems with some other programs as well. In order to provide for true decenders on the new lower case characters, the upper case character definitions have all been raised by one dot. If a program re-defines only a few characters, then those characters will appear lower on the line than the new characters. This can be corrected by either removing the character definition lines from the program, or changing the character definitions to raise the re-defined characters by one dot. This will not always be possible due to protected programs, etc., but by placing the OpSys switch in the OpSys position the old character set will be used.

One other modification has been made to the operating system that corrects a problem associated with the power-up routine of the CorComp Disk Controller. This problem affected correct handling of file errors in the Editor/Assembler module. Although this problem has also been corrected in the modified Editor/Assembler, it has been included in the operating system for the times that it may be necessary to use the Editor/Assembler module in the cartridge slot on the GRAM KRACKER.

MODIFIED COMMANDS

The following commands, keywords, and subprograms have been modified:

CALL INIT

In the old Extended Basic, CALL INIT loaded more data into expansion RAM than necessary. This has been corrected.

LIST

The Extended Basic LIST routine has been modified to allow you to specify a line length if you are listing to a device such as a printer. The following syntax is used:

LIST "device name":line length:start line number-end line number

You can specify a line length ONLY if you also specify an output device. A colon must follow the line length. If not included in the LIST command, the line length is set to the default of the specified output device. The line length can range from 1 to 255. If the length specified is outside this range, a Bad Line Number Error is generated. This is due to the routine used to detect the number.

CALL LOAD

This subprogram, when used to 'poke' data into RAM (ie CALL LOAD(8192,15)), no longer checks to see if you've done a CALL INIT. If used to load an assembly object code file, the CALL INIT check is still made.

PERMANENT

This keyword, which could be used in an OPEN statement has been deleted. This is really no loss as its' use had no effect on a file anyway.

RESEQUENCE

This command has been deleted. You must use the abbreviation RES to renumber a program.

RES

Modifications have been made to the RES command which allow you to resequence a portion of the program. Also, please note that the new RES routine DOES NOT replace undefined line numbers with 32767. Any undefined line numbers in the program are left as is. The new command format is:

RES initial line , increment , starting line - ending line

If you want to use the default values for initial line (100) and increment (10) then you must include the two commas before specifying the starting line. The following commands are all legal:

RES 20,1,10-50	Lines 10 thru 50 are renumbered. Line 10 becomes 20, and the increment is 1.
RES ,5,700-800	Lines 700 thru 800 are renumbered. Line 700 becomes 100, and the increment is 5.
RES ,,50-80	Lines 50 thru 80 are renumbered. Line 50 becomes 100 and the increment is 10.
RES 1000,,750-	Lines 750 thru the last line in the program are renumbered. Line 750 becomes 1000 and the increment is 10.
RES ,20,-400	All lines thru 400 are renumbered. The first program line becomes 100 and the increment is 20.
RES 20,,40	Line 40 is renumbered to 20.

RES cannot be used to move lines from one location to another inside the program. If the new line numbers generated by the RES would result in a line being moved, a Bad Line Number Error is generated. A Bad Line Number Error is also given if there are no valid program lines between starting line and ending line.

TRACE

This command has been modified to allow the trace output to be sent to any output device, such as a printer. In order to send the TRACE to a device, you must OPEN file number 123 with the desired output device. This MUST be done in a program line. You must also turn the TRACE mode on, either in command mode or within your program. For example, if the following line is in your program,

```
100 OPEN #123:"PIO"
```

and the TRACE is turned on, then the trace output for all lines executed after line 100 will be sent to the PIO port. All lines executed up to and including the line containing the OPEN #123 statement will have their trace numbers printed on the video monitor as in the usual TRACE. It is recommended that you put the OPEN #123 statement in the first line of your program and then use the TRACE and UNTRACE commands to turn the trace on and off. In a nutshell, the TRACE command has been modified to look for an open file #123 before printing the trace number on the screen. If that file IS open, then the trace number is sent to the device specified in the OPEN statement. This may seem to be a backwards way of implementing this feature (as opposed to TRACE("PIO")), but it does offer a lot of flexibility in the trace printout. Consider the following example:

```
1 OPEN #123:"PIO" :: PRINT #123:"Trace of main program"
100 ! Main
110 ! program
130 ! goes
140 ! here.
150 CALL EXAMPLE
160 ! More
170 ! program
180 ! here
190 END
200 SUB EXAMPLE :: PRINT #123: :"Trace from subprogram EXAMPLE"
210 ! subprogram
220 ! goes
230 ! here
240 PRINT #123: :"Returning to main program" :: SUBEND
```

As you can see, having the trace file available to your program can be an advantage.

One disadvantage to sending the trace to a printer is the fact that very few printers will print a line a few characters at a time. For this reason, the trace routine will store the output until a full print line is accumulated. Due to this, several lines of your program will execute before their numbers are printed. You can speed this up a little by specifying a short record length in the OPEN #123 statement.

```
1 OPEN #123:"PIO",VARIABLE 40
```

will cause the trace output to print when 40 characters have been accumulated. (It will also only use half the width of your paper.) NEVER SPECIFY A RECORD LENGTH LESS THAN 9!! The results are unpredictable if you do! While on the subject, the file should always be opened in DISPLAY format, VARIABLE record lengths, and in an output mode (OUTPUT, UPDATE, or APPEND). Since UPDATE, DISPLAY, VARIABLE are the default settings, the only time you should specify anything is if you want a record length other than the default of the device specified in the OPEN statement.

The trace buffer is printed whenever any of the following occurs:

The next traced line number text would exceed the record length.

The program ends.

The UNTRACE command is executed. UNTRACE does not close file #123.

A program error is encountered.

A breakpoint is reached.

You stop the program with the BREAK (FCTN 4) key.

Since the break key will create an error if outputting to the RS232 card, a routine has been added that waits for you to release the break key before printing the trace buffer. This wait routine functions if file #123 is open, whether or not the TRACE is turned on.

If your program closes file #123 any lines executed after the CLOSE statement will be traced to the screen. It is recommended that you first print to file #123 before closing it to ensure that the buffer contents are printed. Use the following line to close the file:

```
PRINT #123: ::CLOSE #123
```

One last feature of the new TRACE routine. If the current line being traced is not the next physical program line after the last traced line, an asterisk is printed prior to the current line number. This only applies if file #123 is open.

NEW FEATURES OF GK EXTENDED BASIC

SCREEN AND CHARACTER COLORS

The screen color has been changed to dark blue, and the character color has been changed to white. These color changes are in effect only while in command mode. When a program is RUN the screen and character colors revert to black on cyan.

CHARACTER PATTERNS AND ERROR MESSAGES

The cursor is re-defined to an underline. Error messages are changed from all upper case to upper/lower case. If the OpSys switch on the GRAM KRACKER is in the GRAM 0 position, a new character set with true lower case characters will be used. In this character set the zero character is slashed and the O character is rounded.

QUIT KEY CHANGE

Upon entry into Extended Basic, the QUIT key (FCTN =) is disabled. You may use CALL QUITON (see below) to re-enable the QUIT key.

AUTO LOAD BYPASS

Upon entering Extended Basic, the attempt to run a program named LOAD from DSK1 can now be bypassed. By holding down any alpha-numeric key on the keyboard while entering Extended Basic, the load attempt is bypassed. Any key includes the numeric key pressed to select Extended Basic from the main menu screen. If you do not wish to bypass the auto load, you must release the menu selection key quickly.

CURSOR MOVEMENT

Additional cursor control has been added when entering or editing a program line. These new movements are implemented by pressing the SHIFT key along with the FCTN key and an arrow key. Pressing FCTN-SHIFT S (left arrow) will return the cursor to the beginning of the input line. FCTN-SHIFT D (right arrow) will position the cursor after the last non space character in the line. FCTN-SHIFT E (up arrow) will move the cursor up one line, provided the cursor is not already in the first screen line of the input. Pressing FCTN-SHIFT X (down arrow) will cause the cursor to move down one line, provided the cursor is not already in the last screen line of the input. These additional cursor movement controls will also work when responding to an INPUT or ACCEPT statement in a running Extended Basic program.

NEW COMMANDS

The following commands and subprograms have been added to Extended Basic.

COPY

The COPY command is used to copy a program line or block of program lines to any other location in the program. The format is:

COPY starting line - ending line , new starting line , increment

The original line or lines remain intact. The block to be copied is defined by starting line and ending line. If either of these numbers are ommited, the defaults are the first program line and the last program line. However, at least one number and a dash must be entered (you cannot omit both), and there must be at least one valid program line between starting line and ending line. To copy one line you must enter it as both the starting and ending line number. If any of the above conditions are not met, a Bad Line Number error will result.

The new starting line number defines the new line number of the first line in the block to be copied. This number must be entered. There is no default. The increment defines the line number spacing of the copied lines and may be ommited. The default is 10.

There must be sufficient space in the program for the copied segment to fit between new starting line number and the next program line following the location where the block will be moved. If not, a Bad Line Number error message is generated. This problem can be corrected by using a smaller increment, or by using RES to open up space for the segment. A Bad Line Number error also results if the copying process would result in a line number higher than 32,767.

The COPY routine does not change any program references to the copied lines. It is an exact copy of the source lines with new line numbers. A check for sufficient memory space is made before each line is copied. If space is not available the copying process is halted and a Memory Full error results. PLEASE NOTE - the COPY command copies the lines in reverse order. If the copying process is halted due to insufficient memory space, any uncopied lines will be at the beginning of the block.

Before the first line is copied, any open files are closed and all variable values are lost.

DEL

This new command will delete a line or group of lines from your Extended Basic program. The format is:

DEL starting line - ending line

Starting line and ending line define the block of lines to be deleted. If starting line is ommited, line deletion will begin at the first line of the program. In this case, ending line must be preceeded by a dash. If ending line is ommited and starting line is followed by a dash, then program lines from starting line through the end of the program will be deleted. At least one valid program line must exist between starting line and ending line. If not, a Bad Line Number error will result. If only one number is given, without a dash, then that one line will be deleted, if it exists. If it does not exist, a Bad Line Number error is generated.

After the DEL command has executed any open files are closed and all variable values are lost.

CALL EA

This call will pass computer control to the Editor/Assembler, provided you have choosen the combined Extended Basic and Editor/Assembler version of the patch program. Any Extended Basic program stored in memory will be lost and any open files are closed before the Editor/Assembler is entered.

NOTE - 32K memory expansion is required for the operation of the Editor/Assembler.

MOVE

The MOVE command is used to move a program line or block of program lines to another location in the program. The format is:

MOVE starting line - ending line , new starting line , increment

The block of lines to be moved is defined by starting line and ending line. If either of these numbers are ommited, the defaults are the first program line and the last program line. However, at least one number and a dash must be entered (you cannot omit both), and there must be at least one valid program line between starting line and ending line. To move one line you must enter it as both the starting and ending line number. If any of the above conditions are not met, a Bad Line Number error will result.

The new starting line number defines the new line number of the first line in the moved segment. When the block is moved it will be renumbered and all references to those lines in the program will be changed to reflect the new line numbers. The new starting line number MUST be entered. There is no default. The increment defines the line number spacing of the moved lines, and may be ommited. The default value is 10.

There must be sufficient space in the program for the moved segment to fit between new starting line number and the next program line following the location where the block will be moved. If not, a Bad Line Number error message is generated. This problem can be corrected by using a smaller increment, or by using RES to open up space for the segment. A Bad Line Number error also results if the renumbering process would result in a line number higher than 32,767.

Although moving lines within the program does not increase the size of the program, this new command requires 4 bytes of program space for each line to be moved. This memory use is temporary, but it must be available in order to move the block. If sufficient memory is not available a Memory Full error results and no lines are moved. This problem can usually be worked around by moving the block a few lines at a time.

Before the block is moved any open files are closed and any variable values are lost.

CALL PEEKG(grom/gram address,numeric variable list)

This subprogram reads data from GROM or GRAM into the variable(s) specified. It functions identical to the regular PEEK subprogram except that it reads from GROM or GRAM.

GROM/GRAM addresses above 32,767 must be converted to a negative number by subtracting 65,536 from the desired address.

CALL POKEG(gram address,value list)

This subprogram writes the data in value list to GRAM at the specified address. It functions identical to CALL LOAD except that it writes to GRAM. Assembly language object files cannot be loaded with CALL POKEG. Use CALL LOAD for that. GRAM addresses above 32,767 must be converted as described in CALL PEEKG.

NOTE - all GRAMs except 1 2 in the GRAM KRACKER are write protected when the BANK switch is in the W/P position. Moving this switch to either BANK 1 or BANK 2 may cause the Extended Basic interpreter to crash. Use POKEG with caution!

CALL PEEKV(vdp address,numeric variable list)

This subprogram reads data from VDP RAM into the variable(s) specified. It functions identical to the regular PEEK subprogram except that it reads from VDP.

The VDP address should not exceed 16,383.

CALL POKEV(vdp address,value list)

This subprogram writes the data defined in value list to VDP RAM at the specified address. It functions identical to CALL LOAD except that it writes to VDP. Assembly language object files cannot be loaded with CALL POKEV. Use CALL LOAD for that.

The VDP address should not exceed 16,383.

CALL QUITON

Makes the QUIT (FCTN=) key functional. There are no optional parameters. You may need to use this command before running certain programs that use the QUIT key to end.

CALL QUITOFF

Disables the QUIT (FCTN=) key. The QUIT key is automatically disabled when you enter Extended Basic.

XB CALLS

The new CALLs described on page 25 of the GRAM KRACKER manual (CALL NEW, CALL BYE, CALL CLSALL, CALL CLOCK, CALL CLKOFF, and CALL CAT) are also included in this modified version of Extended Basic. The operation of these CALLs is described in the GK manual. When executing CALL CAT, pressing the SPACE BAR will pause the listing and pressing BREAK (FCTN 4) will abort the catalog.

EDITOR/ASSEMBLER MODIFICATIONS

SYSTEM SUPPORT

The modified version of the Editor/Assembler no longer supports the 99/4 computer. A 99/4A is required. All TI BASIC support routines (CALL INIT, CALL LINK, CALL LOAD,CALL PEEK, CALL PEEKV, CALL POKEV, and CALL CHARPAT) have been removed from the Editor/Assembler. If you have a program that must be run from TI BASIC and requires these routines, you must plug an Editor/Assembler module into the cartridge connector on the GRAM KRACKER. If you do this, be sure to place the GRAM 1-2 switch in the TI BASIC position.

There are some assembly language programs that access data internal to the Editor/Assembler cartridge. These programs will not run correctly due to the re-structuring of the data in the Editor/Assembler module. For these programs you must use your Editor/Assembler cartridge.

COSMETIC CHANGES

A large portion of the Editor/Assembler modifications are purely cosmetic and have no effect on the operation of the package. The character and screen colors have been changed to white on dark blue. All menu page headings have been centered and menu item descriptions are now in upper/lower case. If the OpSys switch on the GRAM KRACKER is in the GRAM 0 position the new character set with true lower case characters will be used.

KEYBOARD INPUT MODIFICATIONS

Repeating keys have been added to the keyboard input routine used at filename prompts. In addition, the ERASE (FCTN 3) key has been activated. Pressing FCTN 3 will erase the entire input line at these prompts. A new key and two additional cursor movement keys have been added. Pressing CLEAR (FCTN 4) will clear the input line from the position of the cursor to the end of the line. Pressing FCTN-SHIFT S (left arrow) will return the cursor to the beginning of the input line and pressing FCTN-SHIFT D (right arrow) will place the cursor after the last character on the line.

FILENAME RECALL

When you make a menu selection that requires a filename, device name, or option entry, your last input (up to 30 characters) at that prompt will be recalled. If you wish to use that entry again, just press the ENTER key. If you want to enter a different name, press ERASE and the input line will be cleared. You may also edit the recalled input by using the left and right arrow keys along with the DELETE and INSERT editing functions.

The input line recalled at all EDITOR menu page selections (Load, Save, and Print) will be the same. (If you LOAD DSK1.EXAMPLE, then select SAVE from the menu, DSK1.EXAMPLE will be recalled to the input line.)

NOTE - In order for the filename recall feature to operate correctly, the GRAM 1-2 / TI BASIC switch on the GRAM KRACKER must be in the GRAM 1-2 position. If this switch is in the TI BASIC position, meaningless characters will be recalled to the input line. The input line can be cleared by pressing ERASE. Also, since Grams 1 and 2 in the GRAM KRACKER are not write protected, it may be possible for the data stored there to be destroyed by system crashes or runaway assembly language programs. If that happens the recalled input line may be incorrect. Use ERASE to clear the line and then type the correct name.

ASSEMBLER MODIFICATIONS

The Assembler programs (ASSM1, ASSM2) now load from GRAM rather than disk. When you select Asssemble from the main Editor/ Assembler menu, the following prompt appears:

```
Loader switch OFF,
GRAM 1-2 enabled? (Y/N)
```

Since the assembler is stored in GRAMS 1 and 2, this switch must be in the GRAM 1-2 position. The GRAM KRACKER Loader also occupies space in GRAM 1, so the LOADER switch must be OFF. After you ensure that the switches are configured properly, press the Y key and the assembler will load from GRAM. If you have not configured the switches properly you will get a Checksum Error when the program attempts to load the assembler. You may also get a Checksum Error message if the data in GRAM 1 or 2 have been damaged. Since GRAMs 1 and 2 are not write protected it is possible for this data to be overwritten by system crashes or runaway assembly language programs. If you get a Checksum Error message and the GRAM KRACKER switches are properly configured you must re-load the GRAM 1 and 2 files.

After the assembler has loaded, operation continues as usual. The only other change that has been made to the assembler is to the error messages. They are now in upper/lower case.

EDITOR MODIFICATIONS

The editor program (EDIT1) is now loaded from GRAM rather than from disk. The GRAM KRACKER switches do not have to be in any special position for the editor to load. There is a checksum check made when the editor is loaded. If you recieve a Checksum Error message when the program attempts to load the editor, you must re-load the entire XBEA module.

The only other modifications to the editor are cosmetic changes to the editor prompts.

NEW MAIN MENU SELECTIONS

6 - Extended Basic

Selecting menu item 6 will take you to Extended Basic. Any text in the editor buffer will be lost. If you want the auto load feature in XB to execute, you must release the 6 key quickly.

7 - Format RAMdisk

This feature works only if you have a Myarc 128K memory card in your system. When you make this selection from the menu you will be asked to verify that you want to format the RAMdisk. Answering N to the prompt will return to the menu without doing anything. Answering Y to the verify prompt will result in the following calls to the Myarc RAMdisk:

```
CALL PART(096,000)
CALL EMDK(5)
```

Refer to the Myarc manual for an explaination of these calls. If you would like a different default configuration for the RAMdisk and print spooler, or a different default RAMdisk drive number, see the technical section of the instructions. For an occasional configuration that differs from the Editor/Assembler defaults, return to Extended Basic and issue the CALLs from command mode.

If your system does not contain a Myarc RAMdisk card, selecting this option from the menu will result in an I/O Error.

8 - Catalog Disk

This menu selection will catalog to the screen any device connected to the system that contains a "catalog" routine. The disk controller card contains such a routine, as does the Myarc RAMdisk and hard disk controllers. For most devices, the device name must be followed with a period. For example, to catalog a disk in drive one, you would respond to the Device name prompt with:

DSK1.

The catalog information displayed on the screen includes device name, disk name, sectors available and used, and the file name, size, and type for each file in the device. The scrolling of the listing can be paused and restarted by pressing the space bar, and can be halted by pressing the break key (FCTN 4). After the catalog is completed, pressing ENTER will return you to the main Editor/Assembler menu.

The information presented in this section is intended as an aid to those persons interested in making further modifications to the XBEA system. Each GRAM will be discussed separately, with important memory locations and unused memory space noted. All memory locations are referenced in hexadecimal, prefaced by the > symbol. You may use the Memory Editor provided in the GRAM KRACKER to make changes if you wish. It is suggested that you save the modified program to disk with the SAVE MODULE or SAVE CONSOLE function of the GRAM KRACKER. See the GRAM KRACKER manual for details on these operations.

GRAM 0

>0724 thru >072A contains the pattern for the zero character (7 bytes)

>07FD thru >0803 contains the pattern for the uppercase O character (7 bytes)

The new lower case character set is stored beginning at >1800.

There are 1,792 free bytes of memory in GRAM 0, from >1900 thru >1FFF.

GRAM 1

The entire memory contents of GRAM 1 are used to store a portion of the assembler program.

GRAM 2

The remainder of the assembler program resides from >4000 thru >5207.

There are 3,276 free bytes of memory from >5208 thru >5ED3.

The section of GRAM 2 from >5ED4 thru >5FFF is used by the Editor/Assembler to store input lines for recall.

GRAM 3

The majority of GRAM 3 is used by the Extended Basic interpreter. Important memory locations are described below. Numbers in parentheses represent the value stored at the location(s).

>635C thru >6363 (>00,>00,>00,>00,>00,>7E,>7E)
This is the character pattern for the cursor used in Extended Basic.

>693A (>04)
This is the screen color for command mode in Extended Basic.

>6948 (>F0)
This is the character color for command mode in Extended Basic. The first hex digit (>F) represents the foreground color, the second hex digit (>0) represents the background color.

>7785 (>07)
This is the screen color for RUN mode in Extended Basic.

>7789 (>10)
This is the character color for RUN mode in Extended Basic. The first hex digit (>1) represents the foreground color, the second hex digit (>0) represents the background color.

There are 2 free bytes of memory in GRAM 3, >77FE and >77FF. Locations >7800 thru >7FFF are used to store a portion of the Editor program.

GRAM 4

The majority of GRAM 4 is used by the Extended Basic interpreter. There are 10 free bytes of memory in GRAM 4 from >97F6 thru >97FF. Locations >9800 thru >9FFF are used to store a portion of the Editor program.

GRAM 5

The majority of GRAM 5 is used by the Extended Basic interpreter. There are 511 free bytes of memory from >B601 thru >B7FF and 256 free bytes of memory from >BF00 thru >BFFF in GRAM 5. Locations >B800 thru >BEFF are used to store the remainder of the Editor program.

GRAM 6

GRAM 6 contains the remainder of the Extended Basic interpreter and most of the code for additional Extended Basic features. Important memory locations are described below. Numbers in parentheses represent the value stored at the location(s).

>D123 (>B6)
Changing this location to >00 will prevent the QUIT key from being disabled upon entry into Extended Basic.

>D789 (>00,>00)
This is the end of the link table used by subprogram calls in Extended Basic.

There are 1,773 free bytes of memory in GRAM 6 from >D8FB thru >DFE7. Locations >DFE8 thru >DFFF contain a branch table for routines called from other GRAMs. Do not change any values in this area.

GRAM 7 — NOTE - XB only version - THIS GRAM IS NOT USED -

The Editor/Assembler module is stored in GRAM 7. Important memory locations are described below. Numbers in parentheses represent the value stored at the location(s).

>E52C (>F4)
This is the character color byte. The first hex digit (>F) represents the foreground color, the second hex digit (>4) represents the background color.

>E537 (>F4)
This byte determines the screen color for the Editor/Assembler module functions and the screen and text color for the Editor program. The first hex digit (>F) represents the character color in the Editor. The second hex digit (>4) represents the screen color for both the Editor and the Editor/Assembler module functions.

>EB79 thru >EB80 (>08,>0C,>FE,>FF,>FE,>0C,>08,>00)
This is the pattern for the arrow used to mark certain menu selections.

There are 2,609 free bytes of memory in GRAM 7, from >F5CE thru >FFFF.

CHANGING THE RAMDISK FORMAT

As described in the section on the Editor/Assembler, the Format RAMdisk menu selection mimics the following commands in Extended Basic.

```
CALL PART(096,000)

CALL EMDK(5)
```

If your Myarc RAMdisk card has memory greater than 128K or you prefer a different format than above, you can change the defaults.

The RAMdisk allocation number (096) is stored at >F312 thru >F314.

The print spool allocation number (000) is stored at >F318 thru >F31A.

The RAMdisk drive number (5) is stored at >F324.

Use the memory editor in the GRAM KRACKER to change these locations to the desired values. Be sure to change them in the ASCII mode. Both the RAMdisk and print spooler allocation numbers must occupy 3 bytes. NOTE - No error checking is done in the format routine with the exception of checking for the existance of the Myarc card. Before changing the numbers in the locations described above make sure those numbers will work without error in the Extended Basic CALL statements.

After making the changes, re-save the program to disk with the Save Module function of the GRAM KRACKER.

CHANGING THE CHECKSUM FOR THE EDITOR AND ASSEMBLER LOADER

If you make any changes to the Editor or Assembler programs it will be necessary to change the checksum byte for the program modified. Also, it is possible that TI produced different versions of those programs. If so, your Editor or Assembler may not load without error after installing the XBEA modifications.

To change the checksum for the editor follow these instructions to the letter!

1E - Using the GRAM KRACKER memory editor, change the byte at GRAM location >F4DB from >83 to >E3.

2E - Exit from the memory editor and enter the Editor/Assembler.

3E - Select Edit (1) from the main menu, then select Edit (2) from the Editor menu.

4E - When the Checksum Error message appears, leave the Editor/Assembler and enter the GRAM KRACKER memory editor.

5E - Replace the >E3 byte at GRAM location >F4DB with >83.

6E - Note the value (1 byte) at CPU memory location >E300.

7E - Change the byte at GRAM location >F513 to the value found at CPU location >E300.

To change the checksum for the assembler, do the following.

1A - Repeat steps 1E and 2E above, then continue with step 2A.

2A - Select Assemble (2) from the main menu. Configure the GRAM KRACKER switches correctly (LOADER OFF, GRAM 1-2 enabled) and press the Y key.

3A - Repeat steps 4E, 5E, and 6E above, then continue with step 4A.

4A - Change the byte at GRAM location >E66F to the value found at CPU location >E300.

After making the changes, re-save the program to disk with the Save Module function of the GRAM KRACKER.