# BLUE RAM - BASIC
# OPERATING SYSTEM (1.0)

INTRODUCTION. The BLUE RAM - BASIC OPERATING SYSTEM (1.0) is a machine-code program which runs in the Blue Ram. It is composed of a GOSUB linker and an advanced editor. The GOSUB linker provides all of the linkages necessary to access BASIC subroutines resident in the Blue Ram. In addition, the GOSUB statement has been enhanced to provide parameter passing as part of the syntax. Also, additional control structures have been implemented for abnormal subroutine exits to higher levels as well as the current level. The editor provides facilities for entering BASIC program statements into the Blue Ram, LISTing Blue Ram BASIC statements, CLEARing Blue Ram BASIC memory, and a special new command for reordering statements or changing, inserting, or deleting any character, word or phrase within any existing statement, whether it is located in normal (screen) memory or in the Blue Ram. The BLUE RAM - BASIC OPERATING SYSTEM (1.0) allows programs of up to 5600 characters (1800 normal memory + 3800 Blue Ram) to be entered, edited, and run with the same ease and simplicity as with previous BASIC programs.

OPERATION. Depress RESET and load the BLUE RAM - BASIC OPERATING SYSTEM (1.0) program tape. It is fully loaded when the screen clears except for:

```
    BLUE RAM OP SYS (1.0)        (C) PERKINS ENGINEERING
BR>
```

The BR> is displayed to indicate that you are operating under the BLUE RAM - BASIC OPERATING SYSTEM as opposed to the normal > prompt. To re-enter the BR> mode at any time in the future (after an error, or after performing a non-BR> function) use CALL24576 or GOTO 1999 (see options 1 and 2). It is important to observe the prompt (BR> or >) because certain commands have different effects when performed in the BR> mode.

INITIAL CONDITIONS. When the BLUE RAM - BASIC OPERATING SYSTEM is first loaded (after a RESET) it is initialized to a "no program" state, i.e. it assumes that there is no BASIC program currently in either normal or Blue Ram memory. To verify this fact, use: PRINT RM . Note that 3800 is printed to indicate that all 3800 characters of Blue Ram memory remain to be programmed. Note also that "PRINT" is not a BR> function so that you are returned to the normal > mode. Any command may be initiated while in the BR> mode but if the command was not processed by the BR> mode, you are returned to the > mode upon completion of the function. Remember that you can re-enter the BR> mode at any time by using CALL24576 or GOTO 1999. Now use: PRINT SZ to verify that normal memory is essentially empty also (1776). The remaining 24 characters of the 1800 total are used for the line number separator (see option 3) and linkage to Blue Ram program statements. Just consider these 24 characters reserved and don't worry about them; they are a necessary part of the program to be entered and must not be removed. Line 1999 is the only visible line (via LIST) and is used both for linkage to the BR> mode and as the highest normal memory line number (line number separator). Enter GOTO 1999.

ENTERING A PROGRAM. Enter program statements into memory in the usual way beginning with a line number. The normal > mode will only enter program lines into normal memory. However, the BR> mode will enter program lines into either memory according to the line number used. Line numbers greater than the line number separator are automatically sorted and entered into Blue Ram memory. Line numbers less than the line number separator are sorted and entered into normal memory. The standard line number separator is 1999. It is the last line number in normal memory and is entered automatically as part of the loading process. DO NOT attempt to enter line numbers higher than this one under the normal > . Doing so destroys the linkage code immediately behind the line separator and prevents the running of Blue Ram BASIC programs. To be on the safe side, use the BR> mode to enter program lines remembering which memory they will end up in. Try entering a few lines in the BR> mode on either side of the line number separator ... say, lines 10, 20, 30, 2000, 2010, 2020. Now use the LIST command to get a listing of the program so far. LIST is a function processed in the BR> mode and may be used to list the entire program. Normal memory lines are listed first followed by a CALL24576, followed by Blue Ram memory lines. The CALL24576 is the dividing line in the listing between the two memory areas. It is also used to arm the BR> mode when loading the program after it is dumped to tape using CALL24576;LIST for the BR > mode. Entering programs into both memories when in the BR> mode works exactly as in the normal > mode, as does the LIST function. CLEAR is another function processed by the BR > mode, however it functions quite differently from its counterpart in the normal > mode. As you are aware, in the normal > mode, the CLEAR command erases the screen. In the BR> mode, the CLEAR command erases Blue Ram BASIC memory, resetting it to the "no program" state. It has no effect on normal memory.

PROGRAM EDITING. A new command has been implemented which only functions in the BR> mode. It is the RPL function and it is used to reorder program lines and/or edit individual program lines. The RPL command takes the form: RPL nnnn/xxx/yyyy where nnnn is an existing line number, xxx is existing text to be deleted, and yyyy is new text to be inserted at that same point in the line. For example, consider the line: 10 PRINT "THIS IS A TEST". The command: RPL 10/A/ANOTHER results in line 10 reading: 10 PRINT "THIS IS ANOTHER TEST". Now try: RPL 10/IS/ISN'T. As you can see, the RPL command found the first instance of "IS" in the word "THIS" so that line 10 now reads: 10 PRINT "THISN'T IS ANOTHER TEST". Care must be taken to give the RPL command enough text such that the substitution is unambiguous. We could have used: RPL 10/ IS/ ISN'T to yield: 10 PRINT "THIS ISN'T ANOTHER TEST". Any length phrase may be used as the existing text or the new text but only the first instance of the existing text in the line will be replaced by the new text. Note that RPL 10/NOTHER/ performs a simple delete since no new text is provided: 10 PRINT "THISN'T IS A TEST". The final (/) is optional in this case. The other form of the RPL command is: RPL nnnn//mmmm where nnnn is an existing line number and mmmm is a new line number. The effect of this command is to delete the existing line number and re-enter the same line elsewhere in the program at the new line number. Try: RPL 10//8000 and notice that line 10 is missing and line 8000

now reads: 8000 PRINT "THISN'T IS A TEST". A simple: RPL 8000 /ISN'T/IS restores the original **statement**: 8000 PRINT "THIS IS A TEST". As you can see, the new RPL command is very powerful and can be a real convenience. Note that the slash (/) was used in each case as a text string delimiter to separate the existing text from the new text. Actually this delimiter may be any character that will not be confused with the existing text. The RPL command uses the first character after the exixting line number as the delimiter and assumes all text until a similar delimiter is existing text.

BLUE RAM PROGRAMMING CONVENTIONS. It is important to remember whether a given program statement is in Blue Ram memory or normal memory because of the way in which they are accessed. Remembering is a simple task since line numbers below the line number separator are in normal memory and those above are in the Blue Ram. Normal memory program lines are accessed in the normal way using GOTO for branching and GOSUB for subroutine calling. For Blue Ram program lines, however, a separate, more flexible syntax is provided. Let us review Bally's subroutine structure for a moment. A subroutine is a segment of program which ends with a RETURN statement and is envoked via the GOSUB statement. When the RETURN is encountered, program control picks up at the point just beyond the GOSUB which envoked the subroutine. The record of where to return to is kept in an internal (invisible) "stack" in Bally memory. Each time one subroutine envokes another, its corresponding return pointer is saved in the stack. Since returning to the envoking statement is the only use of the return pointer, that stack "level" is released during the return process. Each time a return pointer is saved, the stack is pushed to a lower level and each time a pointer is released, the stack is "popped" to a higher level. The Bally's stack is about 25 entries worth (shared with FOR/NEXT loops), meaning that the main program can envoke a subroutine, which can envoke a subroutine, which can envoke a subroutine, etc., to 25 levels (assuming no loops). Blue Ram program segments are assumed to be subroutines unto the main program and are therefore envoked via the GOSUB statement. The format for this statement can be as simple as: GOSUB 2000 or as complex as: GOSUB 2000,7,Q+5,"XYZ"*** In the simple case the GOSUB 2000 simply envokes a subroutine beginning at line number 2000. The second example is roughly equivalent to: A=7;B=Q+5;C=text beginning pointer;D=text ending pointer;RETURN;RETURN;GOTO 2000. The syntax of the simple GOSUB statement has been expanded for accessing Blue Ram program lines, AND BLUE RAM PROGRAM LINES ONLY! Normal memory lines may not be referenced using this expanded syntax. The syntax expansion is made up of two optional parts: 1) parameter passing, and 2) control level selection. Each value following the GOSUB line number and separated by commas is considered to be a parameter to be passed to the line number being referenced. Parameters are passed via the letter variables A through Z. The first parameter is passed via variable A, the second via variable B, etc. Therefore, the first two parameters in the above example are passed through A and B just as if the statement A=7;B=Q+5 had preceded the GOSUB statement. In the case of the third parameter, it was a text string XYZ which cannot be passed in a variable. What is passed in the next two variables (C and D in this example) are pointers to the text string, the beginning of the string in the first

variable (C) and the end of the string+2 in the second variable (D). If the string is the last parameter and is terminated by a GO (end-of-line) then the second variable points to the GO (end-of line) character. If you do not wish to contend with this anomoly, terminate the last string with the closed quotes. Note: the single quote (') cannot be used to identify a string parameter. The beginning and end pointers can be used by the referenced subroutine to "peek" the string from memory using the %() form. For example:

```
 10 GOSUB 2000,"SAMPLE TEXT STRING"
  :
2000 FOR A=A TO B-2;TV=%(A);NEXT A;RETURN
```

The statement at line 2000 used the pointers (A and B in this case since these were the first two parameters) to "peek" the string from memory and print it on the screen. If a string of no length is passed, i.e. "" , the subroutine can test for B>A+1. If B is greater than A+1 then at least one character is contained in the string parameter. Actually, B-A-1 is equal to the string length. The asterisks, with no comma separators, are for control level selection. Each asterisk results in the return pointer stack being "popped" up one level. Remembering that the GOSUB itself "pushed" a return pointer into the stack, the first asterisk pops that pointer out of the stack and returns the stack to the level of the statement which envoked the GOSUB. The net effect is a GOTO type of statement, since we end up at the referenced line number at the same level. This single asterisk form must be used in place of a GOTO statement when referencing Blue Ram program lines. Use of a GOTO with a Blue Ram line number has the effect of restarting the current subroutine at its original beginning line number, regardless of the line number used with the GOTO. Since an asterisk used in conjunction with a GOSUB to a Blue Ram line number has the effect of popping a return pointer out of the the stack and raising the stack one level, a return to the point just beyond the GOSUB is never taken, and anything else on that line will never be seen by the program. It is just like GOTO 10;PRINT "HELLO". The PRINT following the GOTO can never execute. Each asterisk has the effect of popping out a return pointer and raising the stack one level. Two asterisks, therefore, pop two return pointers out of the stack and three pop three, etc. This feature gives the subroutine the choice of returning to the envoking line (via the normal RETURN statement) or returning to some other line even at some higher level. For example:

```
 10 GOSUB 2000
  :
2000 K=KP;GOSUB 2020,K
2010 RETURN
2020 IF (A<48)+(A>58) GOSUB 2000**
2030 PRINT A-48," IS A NUMBER KEY"
2040 RETURN
```

This sample program (beginnigg at line 10) envokes the Blue Ram subroutine at line 2000. This subroutine accepts a single key from the keyboard and passes it as a parameter to a lower level subroutine at line 2020. The subroutine at 2020 checks the passed parameter for a number key and if it is not, exits abnormally (up one level) to line 2000 to get another key. Otherwise, it prints

the corresponding number and returns normally. This type of control structure is especially valuable for aborting a processing sequence when an error is detected several subroutine levels down.

FINAL NOTES: The recommended command for dumping a Blue Ram BASIC program onto tape is: :PRINT ;NT=1;CALL24576;LIST ;:RETURN . The CALL24576 is required immediately before the LIST to cause the LIST command to processed in the BR> mode, thereby listing the entire program, including the Blue Ram part. Remember that the BLUE RAM - BASIC OPERATING SYSTEM (1.0) must be loaded prior to loading a Blue Ram BASIC program tape. It is a good idea to put as much of your program as possible in the Blue Ram since Blue Ram memory is non-volatile and will actually run programs up to 30% faster than the same program run in normal (screen) memory. Be especially careful not to accidentally perform a CLEAR operation while in the BR> mode since this will clear Blue Ram BASIC memory, not the screen. If normal memory is accidentally cleared via the RESET button, the Blue Ram BASIC program part can be recovered by reloading the BLUE RAM OPERATING SYSTEM and then entering this command statement: &(192)=0;%(24997)=2000;%(24999)=16705;&(64)=0 , and then immediately dumping the Blue Ram BASIC program to tape using the command statement: ;PRINT ;NT=1;CALL24576;LIST ;:RETURN . This action saves the Blue Ram BASIC program part on tape where it can be loaded again in the normal way, except that the first line of the Blue Ram BASIC program is number 2000 (whether or not it was originally) and the first two characters of that line are now AA. Use RPL to repair the first line to its original form.

OPTIONS: The following options are available:

1. Line 1999 is initially set to CALL24576 to provide linkage to the BR> mode via the GOTO 1999 command. Eight characters can be saved by changing this line to: 1999. .

2. Line 1999 may be used to directly link normal memory to Blue Ram memory by changing it to: 1999 GOSUB 2000 assuming of course that the first line number in the Blue Ram is 2000.

3. If a line number separator other than 1999 is desired, enter the following command statement: %(%(20050)-28)=nnnn where nnnn is the new line number separator value. Note that this option must be taken before 1 and 2. Do not use RPL for this action.

4. While 3800 additional characters of Blue Ram memory may be programmed in BASIC, that same memory is necessarily shared with the Keyboard driver (if you have a Blue Ram Keyboard), the RPL command, and any other machine-code services you may be using. Up to 3416 characters may be programmed without distroying either the Keyboard driver or the RPL command. At this point, (RM = 400 or so) you may dump the Blue Ram BASIC program entered so far, CLEAR Blue Ram program memory, and then enter the remaining 400 characters. Then simply dump the remaining program lines on tape directly behind the first part. This method works because neither the Keyboard driver nor the RPL command need to be in memory to load or run a Blue Ram BASIC program.

Perkins Engineering
1004 Pleasant Avenue
Boyne City, Michigan
49712