Quickpad PRO Technical and programming information
Revision 1.01 05/01/2002

Changes from previous version:
Chapter 6, spellchecker specifications.
Chapter 8, Serial I/O

1.0 General Information

QuickpadPRO is a DOS based compatible (with a few limitations) machine with the following structure:

640K main memory (SRAM) Actually 1MB, but 640K is avail for use.

There are two flash memory chips used for storage. One is read-only and contains various system and application programs, including the BIOS, and ROM DOS. It may be 1 or 2 Mbyte depending on the model you have. The other is configured as read-write memory and may be either 1 or 2 mbyte in size. 64K is reserved, the remainder is usable as drive B: In some models part of drive B will hold some applications, as well as being used for general storage.

The system is configured with a RAM DISK or drive C: The size is 256K. QuickpadPRO uses drive C: to hold temporary files, files being edited and so on, to minimize erase/write cycles to flash memory.

The system also utilizes a compact flash card slot, which appears to the system as a removable storage device. QuickpadPRO responds with drive not ready, when the card is not installed.

Physically, QuickpadPRO consists of a motherboard, which contains the CPU, memory, power supplies, CF card logic, and keyboard interface. There is an adapter board which currently includes the following interfaces: RS232, USB Keyboard, USB Serial, Modem (2400 baud), Quickpad 1 compatible InfraRed output, and IRDA transceiver. Only 1 function can be used at a time. There are currently 3 connectors, 8 pin mini-din, which has the RS232 I/O and connections for an external power supply and nicad battery charger. There is a USB connector, and a modem connector. The modem circuitry is installed as a build option. The adapter board is currently designed to allow serial I/O communication with all of these peripherals. The serial I/O is set to COM4. (The COM4 assignment cannot be changed)

The LCD screen is 480 X 128, which is a subset of CGA. The controller in the cpu chip (Vadem VG330) has a built in controller capable of CGA or VGA modes, but it is not 100% compatible with the "PC". All screen output should go through the system BIOS, which is detailed further on. The screen is always in mode 6 (CGA).

In subsequent chapters, many interrupt and special function interrupts are documented. This document will be updated and available on Quickpad Technology's website as new functions are developed, tested and added to the firmware.

At this point, programs have been successfully written in Borland turbo C, assembly, and Microsoft Quickbasic. Quickbasic is a very quick way to build user interfaces, and demo programs, some of which have been used in QuickpadPRO. Drawbacks to Quickbasic however are speed, and directory/file management abilities.

2.0 Software Interrupts

Not all interrupts will be described here, just those that are important to be aware of when programming screen I/O, and those that are non-standard from typical PC software interrupts.

The QuickpadPRO is always in CGA mode, attempts to change it will have no effect, other than to clear the screen. A blinking cursor has been implemented through the use of a TSR that is installed at boot which intercepts some of the INT 10 functions, and *emulates* the hardware cursor. Correct programming technique where the cursor is involved requires the application program to disable the cursor prior to any character writes, and to re-enable it when done. If an application program is about to output a string of characters, or several lines of characters, then the cursor would be turned off at the beginning of the string, and re-enabled (if desired) at the end of the string, rather that before and after each individual character. Also, normally, the blinking cursor is used to indicate to the user that keyboard input is required, so would be active only when user text input was required. Another case would be a "pop-menu" by a supervisory program that might put a screen on top of one that contained a blinking cursor. The cursor should be turned off in this case as well, then restored when the screen is removed. More detail will become available later, with programming examples.

All characters at the moment consist of an 8 X 8 matrix, where the actual character pattern is 5 X 7 within the 8 X 8 field, which is standard CGA format. Remember, the screen is 60 characters X 16 lines.

In most programming environments, character display functions include "attributes", such as color, or blinking functions. In QuickpadPRO the character attributes are always zero, or none because the LCD screen is black and white, and the display controller is always in CGA mode.

Inverse characters can be displayed by writing the character to the screen, then writing character 0DB hex using INT 10h function 0Ah with the BL register set to 80h. (Character 0dbh is a solid block, with all bits in the 8X8 matrix set. When BL is set to 80h, the original character at a given location is XORed with the solid block, producing an inverse character). This is a standard CGA function. In standard INT 10h functions, the user can change to alternate pages. In QuickpadPRO, however, only page 0 is currently available to the user.

The following INT 10 functions are available for application programs on QuickpadPRO.

Function 01h; set cursor type

Call with

AH = 01H
CH bits 0-4 = starting line of cursor
CL bits 0-4 = ending line of cursor

Returns:

Nothing.

Comments:

At the moment, CH and CL are "don't care", as far as cursor shape is concerned. The cursor is only one shape, and that is a large block, occupying line 0 to line 7. To **disable the cursor**, place a 20h in register CH. To **re-enable the cursor**, place a value from 0 to 7 in register CH.

Function 02h; set cursor position

Call with

AH = 02h
BH = page (leave this set to 0)
DH = row (0 to 15)
DL = col (0 to 59)

Returns;

Nothing

Comments:

Do not attempt to turn off the cursor by using "off screen" values for cursor position. Use function 01h above. This function can still be used to set cursor position (to position character writes on the screen) even when the cursor has been disabled by function 01h.

Function 03h; get cursor position

Call with:

AH = 03h
BH = page (0)

Returns:

CH = starting line of cursor (indicates cursor off if = 20h)
CL = ending line of cursor
DH = row
DL = column

Function 06h; Initialize or Scroll up window

Call with:

AH = 06h
AL = number of lines to scroll (if 0, entire window is blanked)
BH = attribute (0)
CH = start column (upper left coordinate)
CL = start row
DH = end column
DL = end row

Returns:

Nothing

Function 07h; Initialize or Scroll down window

Call with:

AH = 06h
AL = number of lines to scroll (if 0, entire window is blanked)
BH = attribute (0)
CH = start column (upper left coordinate)
CL = start row
DH = end column
DL = end row

Returns:

Nothing

Function 0Ah; Write Character at Cursor position

Call with:

AH = 0Ah
AL = character
BH= page (0)
BL= color (0), or 080h (see comments)
CX= # of char to repeat, valid for given row only.

Returns:

Nothing

Comments:

After a character is written, the cursor (visible or not) must be explicitly moved to the next desired position. If register BL is set to 80h the character specified will be XOR'd with the character currently at the cursor position. This can be used to display a character in "inverse" video; first write the desired character, the write a block character (0dbh) with BL = 80h at the same position.

Function 0Eh; Write Character in Teletype mode

Call with:

AH = 0Eh
AL = character
BH = page (0)
BL= color (0)

Returns:

Nothing

Comments:

ASCII codes for bell (07h), backspace (08h), carriage return (0Dh) and line feed (0Ah) are recognized, and the appropriate action is taken. All other characters are written to the screen and the cursor position is incremented. Line wrapping and scrolling are provided.

Function 13h; Write string in Teletype mode

Call with:

AH = 13h
AL = write mode]
    0 = cursor position is not updated at completion
    1 = cursor position is updated at completion
BH = page (0)
BL= color (0)

CX = length of character string
DH = row
DL = column
ES:BP = segment:offset of string

Returns:

Nothing

Comments:

This function can be thought of an extension to INT 10h Function 0eh. ASCII codes for bell (07h), backspace (08h), carriage return (0Dh) and line feed (0Ah) are recognized, and the appropriate action is taken. Write modes 2 and 3 are not available.

3.0 Keyboard character input

The following is a suggestion for inputting characters from the keyboard, coordinated with screen display on the screen.

1. Position cursor at desired row, column.
   INT 10h, function 2; AH=2, BH=0, DH= row, DL= col.
2. Turn on cursor
   INT 10H, function 1, AH=1, CH=0, CL=0
3. Wait for keyboard input
   INT 16H, function 10h, AH=10h (This routine causes Quickpad to enter power down state while waiting for a key press) INT 21h function 7 or 8 may also be used for input. When the key is pressed, the Quickpad CPU powers back up automatically. Int 16h function 10h also returns the character and scancode. (See Special Scancodes)
4. Process the key, display it if appropriate, or exit if desired key(s) have been pressed
   INT 10h, function 0Ah to display character
   Increment column
5. Turn off cursor
   INT 10h, function 1, AH=1, CH=20h, CL=0
6. Loop to item 1

We recommend interrupt 16h for input over the standard INT 21h functions because of the clarity of the information. Special function keys in INT21h require two input sequences rather than one. Special function keys are defined as non printing characters, for example the arrow keys, or the F0 through F9 keys.

4.0 Special Keyboard Scancodes

There are several special scancodes that may be returned by the INT 16h function, some of which are not the result of an actual key press. Since these are special functions, the result returned by interrupt 16h function 10h have the ASCII value set to zero, and the scancode to that shown in the table.

Key Function	Scancode	Description
Power ON/OFF	68h	Used by system BIOS to turn system on and off. Applications may be required to shut themselves down. See note 1.

Calculator	69h	Used by “shell” program to start calculator
Print File	6Ah	Used by “shell” program to start print function
Send File	6Bh	Shell program (and others) will attempt to send selected text file by starting program F2IO.EXE. Note 2.
Delete File	6Ch	Shell program will delete a selected file.
Spell Check	6Dh	Currently used by edit.exe; a built in spellchecker will verify the correctness of a word pointer to by various registers. See SPELLCHECKER.
Windows Key	6Eh	When QPPRO is used as either an IR or USB keyboard, this standard windows code is sent.
Windows Menu	6Fh	When QPPRO is used as either an IR or USB keyboard, this standard windows code is sent.
5 Minutes expired	70h	A system timer will cause this code to be placed in the keyboard buffer. Note 1
Low Battery	71h	This code will be placed in the keyboard buffer when a low battery condition has persisted for 1 minute. Note 1

Notes:
1. When an application program detects code 68h, 70h or 71h it shall close all open files and exit to DOS with that scan code returned to the parent program via DOS INT21h function 4Ch. The parent program (typically the program manager) will receive the return code via INT21h function 4Dh and save certain files in RAMDISK, such as clipbrd.txt on the flash drive B. The manager or shell program will flush system buffers to ensure disk writing is complete, then after a short wait, will shut down the system.

2. F2IO.exe is a utility for use by application programs. It will attempt to send a specified text file to a PC either via USB keyboard (if connect is successful) or Quickpad 1 compatible IR keyboard otherwise. It is intended to send a text file directly to an active editor program on a PC. The parent program should use it’s EXEC function. Command line parameters are: F2IO [path](filename) /m /#. /m and /# apply only to QP1 IR mode only and may be omitted in any case. /m caused Mac Mode to be active (PC mode is default), and /# (where # is a single digit 1 to 9, controls IR transmission speed, where 9 (default) is fastest). If a file contains non printing ASCII characters, they will be skipped. Transmission can be stopped by pressing ESC. Transmission will stop normally when an ASCII Z (zero byte) is detected, or the end of file is reached.

Low battery functions: If the system detects a low battery condition at BOOT, it will attempt to display a low battery message and will not start command.com. If during operation, a low battery condition has been persistent for 1 minute a temporary message will be displayed. If the low battery condition persists for 5 minutes the BIOS will generate the “low battery scancode” and system shutdown will proceed. Each application is responsible for detecting this scancode and exiting to the shell program. OEMs that wish to take advantage of this feature should save data, close files, then call INT 10H, function 50H to power down the QuickpadPRO.

5.0 Special function interrupts

Interrupt 60h: Spellchecker interrupt. As of this writing, the spell checker function, which will be available to all users has not been completed, but should be available soon. User Interrupt 60h is therefore reserved for the spell checker function.

Interrupt 61h: User interrupt 61h is reserved for use by the cursor emulation TSR (terminate and stay resident) program; CSR.EXE. This program is loaded at boot time. In addition to cursor emulation, CSR.EXE also implements many of the special function interrupts and should not be deleted by any OEMs writing their own programs.

INT10H Function 50h: System Shutdown
Enter with AH = 50h. Nothing is returned. A shutdown message is displayed, a disk reset is issued (buffers are flushed), system power is shutdown, and the CPU enters standby. Power consumption is reduced to 500 microamps.

INT 10H Function 51h: Set Idle Timer. Determines when system will attempt to turn off.
Enter with AH =51h, BX = 1 to 75 (decimal minutes), or the value 100d (64h) to disable the auto-shutdown feature. Example: set AH = 51h, set BX = 10d. After 10 minutes of no keyboard, or disk activity, the system will begin generating a key scancode of 70h 18 times per second. Any program that is running (such as an editor) will see this as a key press. If programmed to recognize scancode 70h, the editor can save any open files and exit. In the case of QuickpadPRO, the editor implements this function and exits. The PRO "shell" also recognizes this character and saves any open files it may have in RAM DISK, then calls the system shutdown interrupt INT 10H Function 50h.

INT 10H Function 52h is reserved, planned for special character functions.

INT 10H Function 53h LCD Display auto-off.
Enter with AH = 53h, AL = 0 or 1. (1 = auto off is enabled)
Controls whether the display shuts off after approximately 2 minutes of system inactivity. At boot, QuickpadPRO sets this function on, meaning that after 2 minutes of inactivity the LCD display will shutoff. Any keystroke will wake it up again. It has been implemented to save power.

INT 1AH Function 50h Reserved power management function

INT 1AH Function 51h: Shutdown.
Enter with AH = 51H
Puts CPU into standby and removes system power. Unlike interrupt call 10h function 50h, this does not display any message and it does not do a disk reset to ensure that disk buffers are empty..

INT 1AH Function 52h Reserved Video function.

INT 1AH Function 53h Screen Magnify On
Enter with AH = 53H.
Causes characters to be displayed double size. The screen effectively becomes 8 lines X 60 char (vs 16 X 60). This is only a screen function, DOS and BIOS are unaware. It is provided as a quick way to increase screen readability.

INT 1AH Function 54h: Screen Magnify Off
Enter with AH = 54H
Characters are displayed in normal size, 16 rows X 60 columns.
Note: this function is also available to the user by pressing FN-Menu keys simultaneously. (The Menu key is the windows menu key, to the left of the left arrow key on the Quickpad Pro.

INT 1AH Function 55h: Reserved BIOS information call. Provides BIOS and DOS revisions.

Important Note; User Timer Interrupt 1Ch

The user timer interrupt is utilized by the cursor emulation TSR, which is loaded at boot time. If you need to use the timer interrupt, you may do so by setting up your interrupt vector, then when you are finished with your timer function, you must execute a far jump to the interrupt vector you replaced, with all registers and flags in place. Your timer function code should be concise, and quick! The QuickpadPRO cpu runs at just 24mhz to minimize power consumption.

6.0 Spellchecker specifications: 5/1/2002

Any program may use Quickpad Pro's built in spell checker. It may be accessed by using DOS interrupt 60h, one of the "user" interrupts. When Quickpad Pro boots, a small TSR called SPCK.EXE will be installed that installs vectors at INT 60H. The TSR will know how to access the Spell Checker dictionary stored in ROM.

To use the INT 60h call, first call INT 21h function 35h to verify that INT 60 has been installed:
Enter with:

```
AH=35h
AL=60h
```

Make call:

```
INT   21h
```

Returns:

ES:BX = Segment:offset of interrupt handler.

There are two ways to determine if the spellchecker is installed. 1, If ES:BX that is returned from above call is zero, then no spellchecker is installed. 2, (Preferred) Examine the contents of word ptr [ES:BX+2]: The installed spellchecker signature is: 5432h. If no handler has been installed, the spell check should be skipped.

Note: this does not need to be called each time, only once during a programs execution to determine the presence of the spell checker function.

Spell check functions:

Enter with:

```
AH=01       ;spell check function
DX:SI       ;segment:offset of ASCII string to be checked (your text string)
```

Make call:

```
INT   60h
```

Returns: (On words that are spelled correctly)

```
CY=0        ;if the spell check found a valid word
DX:SI       ;offset of current word terminator + 1. (of your text string)
DX:DI       ;original entry start point (provided for convenience)
```

Returns: (On misspelled words)

```
CY=1        ;word misspelled (according to dictionary)
DX:SI       ;segment:offset of first suggestion.
CX          ;number of suggestions
```

The first suggestion result is located at DX:SI, and consists of an ASCII zero terminated string, with a maximum of 16 characters (including the terminator). To obtain the next suggestion simply add 16 to the original suggestion pointer (DX:SI+16).

When the editor, or other program using the spell checker is checking words, there is no need to display any activity, other than a simple message that "spell checking" is in progress. If the spell checker returns with some suggestions, then they should probably be displayed as a list, that the user can scroll through, and select. The user should be able to "ignore" a suggestion, "accept" a suggestion (in which case the editor will replace the original word with the suggestion), or to edit the word in question himself.

The spell checker utilizes a maximum word length of 15 characters. Words longer than 15 characters will return a "no suggestions" result. The spell checker checks one word at a time, and the DX:SI register which is returned upon a "good word" can simply be used to call the spell checker for the next word. This assumes that the ASCII string to be spell checked is of course contiguous in memory. The spell checker knows nothing about the structure of any program using it, so caution should be exercised not to go past the end of a file or data structure.

A word to be checked is "terminated" by any non alphabetic character. For example, the word "isn't" is actually two words to the spell checker, the word "isn" and the word "t". This was an easy way to get around all the possible abbreviations of words that exist in the English language. So, "isn" was added as a valid word, and all single letter occurrences are also considered valid.

When supplying the spellchecker with words, the user does not need to point to the first alphabetic character. Example: A word might be preceded by several spaces, tabs, or "carriage return" characters. The spell checker will scan for the first alphabetic character before actually doing a spell check. An alphabetic character is defined as upper case A through Z or lower case a through z.

Please note that the spell checker is case insensitive, and does not know any proper names. Also, the spell checker does not currently have a learn mode.

7.0 Programming Examples

The following examples are given to show the programmer how to get started writing code for the QuickpadPRO. This is not a programming tutorial!! Examples are shown for cases that either require or perform significantly better with specific code. In Basic, you may use all of the file functions as is. The areas of difference are primarily LCD screen output, and serial I/O. Keyboard input may be enhanced by using interrupt code.

7.1 Quick Basic Examples

The following examples show portions of code that were written using Microsoft QuickBasic version 4.5. With the exception of some of the special interrupt function calls (power off, etc) much of the users program can be debugged on a standard DOS PC, or on a DOS Window under Windows 95 or 98.

Example 1; "Hello World!"

```
SCREEN 2                    'SET TO CGA MODE
CLS                         'CLEAR SCREEN
LOCATE 8,25                 'APPROX IN MIDDLE OF SCREEN
PRINT "Hello World"         'SHOW GREETING
SYSTEM                      'EXIT.
```

To run this program on the quickpad, you will have to save and compile it, and generate a stand-alone EXE program. Please refer to your Quickbasic manual on how to do this. This program will run on your PC and the Quickpad! **Note:** Microsoft Quick Basic SCREEN 2 is actually CGA mode 6. Why Microsoft used a numbering system different from standard BIOS calls is anyone's guess.

Example 2; Character input. Note; the following code uses standard Quickbasic language and syntax; it is shown just for reference! You may use this in your code, but it is not recommended because power saving mode (CPU in standby between keystrokes) is never entered.

```
WAITKEY:  X$ = INKEY$                    'X$ = key press value
          IF X$ = "" THEN GOTO WAITKEY   'loop until key press
```

The programmer would then build his input routine having retrieved an ASCII code in variable X$. In this example, the code is continually looping until a key is pressed. There is a much better method (albeit more complex) that allows the CPU to shutdown between keystrokes, saving power.

Example 3; Character input using BIOS interrupts.

First, when this code is compiled, you must use the /L option on the command line when you load your program into Quickbasic. If your program is called "TEST" you would begin with:

QB TEST /L

This allows Quickbasic access to the library files (part of basic) that give access to software interrupts. You must then include several other items in your code.

```
'Program to perform keyboard input using interrupts  (Works on PC and
'QuickpadPRO)


DIM SHARED IREG(10) AS INTEGER      'space for 8086 registers
DIM SHARED OREG(10) AS INTEGER
DIM SHARED CHAR AS INTEGER          'variables used in input routine
DIM SHARED CODE AS INTEGER
DIM SHARED X$
```

```
DECLARE SUB WAITKEY ()              ‘define subroutine

CONST AX = 0                        ‘define 8086 registers
CONST BX = 1
CONST CX = 2
CONST DX = 3
CONST BP = 4
CONST SI = 5
CONST DI = 6
CONST FL = 7
CONST DS = 8
CONST ES = 9


           SCREEN 2                 '(CGA MODE)
TEST0:     CLS
           PRINT "CHARACTER INPUT TEST ROUTINE"
TEST:      LOCATE 2, 1
           WAITKEY                          ‘WAIT FOR A KEY
           PRINT X$, CHAR, SCANCODE         ‘SHOW WHAT WE GOT
           IF X$ = "X" THEN SYSTEM          ‘IF “X” THEN QUIT
           IF X$ = "C" THEN GOTO TEST2      ‘IF “C” SHOW MESSAGE
           GOTO TEST                        ‘LOOP

TEST2:     LOCATE 5,1                       ‘FURTHER DOWN ON SCREEN
           PRINT “Letter C entered”         ‘ACKNOWLEDGE CHARACTER
           SLEEP 2                          ‘WAIT A BIT
           LOCATE 5,1
           PRINT “                “         ‘CLEAR THE MESSAGE
           GOTO TEST                        ‘LOOP



‘Waitkey subroutine to retrieve a keypress using interrupt 21h

SUB WAITKEY
        IREG(AX) = &H700                    ‘Keyboard input function
        CALL INT86OLD(&H21, IREG(), OREG()) ‘Call int 21h
        CHAR = (OREG(AX) AND &HFF)
        X$ = ""
        SCANCODE = 0
        IF CHAR = 0 THEN
           IREG(AX) = &H700                    ‘KEYBOARD INPUT
           CALL INT86OLD(&H21, IREG(), OREG()) ‘Call function 2nd time
           SCANCODE = (OREG(AX) AND &HFF)      ‘use lower 8 bits only
        ELSE
           X$ = CHR$(CHAR)                     ‘put ASCII code in string X$
        END IF
END SUB


‘Alternate Waitkey subroutine using interrupt 16h
‘NOTE: INT 16h returns AH=Scancode, and AL=ASCII CODE, so we must
‘extract the AH and AL register with simple math.

SUB WAITKEY
```

```
        IREG(AX) = &H1000                       'Keyboard input function
        CALL INT86OLD(&H16, IREG(), OREG())     'Call int 16h
        CHAR = (OREG(AX) AND &HFF)              'extract register AL
        CODE = (OREG(AX) AND &HFF00)/256        'extract register AH
        IF CHAR = 0 THEN
           X$ = ""
        ELSE
           X$ = CHR$(CHAR)
        END IF
END SUB
```

Example 4; Character output using BIOS interrupts

Although you can easily use Basics standard PRINT routine to output to the screen, this limits you standard character display only. BASIC was designed to work with the PC with a minimum of 16 colors. The QuickpadPRO only has 2 colors! Black and white!! In order to do anything different in CGA mode you must use BIOS interrupts to display things. The following example implements a selection list that the user controls by the up/down arrow keys (or space key). The "selected" line is shown in inverse video. It is suggested to run this program on the QuickpadPRO before examining the code. You'll have a much better understanding what is going on!! To compile, just cut and paste into the Quickbasic editor, then save. Don't forget to start the editor/compiler with the /L option on the command line.

```
DIM SHARED IREG(10) AS INTEGER
DIM SHARED OREG(10) AS INTEGER
DIM SHARED CHAR AS INTEGER
DIM SHARED CODE AS INTEGER
DIM SHARED HILITE AS INTEGER
DIM SHARED X$
DIM SHARED ROW AS INTEGER, COL AS INTEGER

DECLARE SUB WAITKEY ()
DECLARE SUB PCHAR (X$)
DECLARE SUB PSTRING (X$)
DECLARE SUB INC (Y)
DECLARE SUB DEC (Y)
DECLARE SUB DRAWBOX (A, B, X, Y)

CONST AX = 0
CONST BX = 1
CONST CX = 2
CONST DX = 3
CONST BP = 4
CONST SI = 5
CONST DI = 6
CONST FL = 7
CONST DS = 8
CONST ES = 9

CONST PGMCOUNT = 6
            SCREEN 2   '(CGA MODE)
            PGMSEL = 0
            CLS
DEMO1:      DRAWBOX 1, 15, 3, 46
```

```
            LOCATE 2, 17
            PRINT "  Welcome to QuickPAD Pro!"
            DRAWBOX 5, 15, PGMCOUNT + 6, 46
            PGM$(0) = "   Text Editor    "
            PGM$(1) = "   Calculator     "
            PGM$(2) = "Personal Organizer"
            PGM$(3) = "   File Transfer  "
            PGM$(4) = "   File Manager   "
            PGM$(5) = "   Spread Sheet   "

            RUN$(0) = "EDIT.EXE"
            RUN$(1) = "CALC.EXE"
            RUN$(2) = "PO.EXE"
            RUN$(3) = "QPADSVR.EXE"
            RUN$(4) = "QPADMGR.EXE"
            RUN$(5) = "SC.EXE"

SHOWPGMS:       FOR SEL = 0 TO PGMCOUNT - 1
               LOCATE SEL + 6, 22
               IF PGMSEL = SEL THEN
                  HILITE = 1
                  PSTRING PGM$(SEL)
               ELSE
                  PRINT PGM$(SEL);
               END IF
               HILITE = 0
            NEXT SEL
            WAITKEY
            IF CHAR = 13 THEN GOTO PGMRUN
            IF CHAR = &H20 THEN
               GOTO SELDOWN
            END IF
            IF CHAR = 0 THEN
               IF CODE = 80 THEN GOTO SELDOWN
               IF CODE = 72 THEN GOTO SELUP
            END IF
            IF CHAR = 27 THEN GOTO DEMOEXIT
            GOTO SHOWPGMS
SELDOWN:        INC PGMSEL
            IF PGMSEL = PGMCOUNT THEN PGMSEL = 0
            GOTO SHOWPGMS
SELUP:          IF PGMSEL = 0 THEN
               PGMSEL = PGMCOUNT - 1
               GOTO SHOWPGMS
            END IF
            DEC PGMSEL
            GOTO SHOWPGMS

DEMOEXIT:       CLS
            SYSTEM

PGMRUN:         DRAWBOX 10, 17, 13, 48
            LOCATE 11, 18
            PRINT "     BASIC demo program!     ";
            LOCATE 12, 18
            PRINT "      RUN:                   ";
            LOCATE 12, 30
```

```
            PRINT RUN$(PGMSEL)
            WAITKEY
            GOTO DEMO1


SUB DEC (Y)
   Y = Y - 1
END SUB

SUB DRAWBOX (A, B, X, Y)
'Draw the first horizontal line..
   LOCATE A, B
   PRINT CHR$(&HDA);
   FOR N = B + 1 TO Y - 1
      LOCATE , N
      PRINT CHR$(&HC4);
   NEXT N
   LOCATE , Y
   PRINT CHR$(&HBF);
'Now draw the middle lines..
MID:
   A = A + 1
   LOCATE A, B
   PRINT CHR$(&HB3);
'   FOR N = B + 1 TO Y – 1      ‘this code clears the middle
'      LOCATE , N               ‘of the box drawn on screen
'      PRINT " ";               ‘was taken out because it’s
'   NEXT N                      ‘faster just printing message
   LOCATE , Y
   PRINT CHR$(&HB3);
   IF (A + 1) < X THEN GOTO MID
'Now draw the final horizontal line..
   LOCATE X, B
   PRINT CHR$(&HC0);
   FOR N = B + 1 TO Y - 1
      LOCATE , N
      PRINT CHR$(&HC4);
   NEXT N
   LOCATE , Y
   PRINT CHR$(&HD9);
END SUB

SUB INC (Y)
   Y = Y + 1
END SUB

SUB PCHAR (X$)
   AL = ASC(X$)
   IREG(AX) = &H900 + AL
   IREG(BX) = 1
   IREG(CX) = 1
   CALL INT86OLD(&H10, IREG(), OREG())
   IF HILITE = 1 THEN
      IREG(AX) = &H900 + &HDB   'BLOCK CHAR'
      IREG(BX) = &H80
      IREG(CX) = 1
      CALL INT86OLD(&H10, IREG(), OREG())
```

```
   END IF
END SUB

SUB PSTRING (X$)
   L = LEN(X$)
   FOR N = 1 TO L
      XX$ = MID$(X$, N, 1)
      PCHAR XX$
      X = POS(0)
      LOCATE , X + 1
   NEXT N
END SUB

SUB WAITKEY
      IREG(AX) = &H700         'KEYBOAR INPUT
      CALL INT86OLD(&H21, IREG(), OREG())
      CHAR = (OREG(AX) AND &HFF)
      X$ = ""
      CODE = 0
      IF CHAR = 0 THEN
         IREG(AX) = &H700         'KEYBOAR INPUT
         CALL INT86OLD(&H21, IREG(), OREG())
         CODE = (OREG(AX) AND &HFF)
      ELSE
         X$ = CHR$(CHAR)
      END IF
END SUB
```

8.0 Serial I/O; how to use the serial port

The serial port (**factory configured as COM2**) in the QuickpadPRO consists of a 16450 compatible UART. This device does not have a built in FIFO. Fast assembly language can easily use the part at it's fastest speed or 115200 baud. Other programming languages with more overhead may be restricted to slower speeds.

The Quickpad Pro consists of a two part assembly, a motherboard and an adapter board. The UART is contained within the CPU chip on the motherboard, but because all of our I/O (USB, IR, RS232, MODEM) goes through the adapter board, the serial I/O lines must be directed to the appropriate peripheral. This is easily done by using the exec function from either BASIC or C to call "setio.exe". From the DOS command line the syntax is as follows:

setio /m1 /b8

m1 refers to rs232 mode, and in the example above /b8 sets the baud rate to 57600. Other baud rates may be selected as follows:

0 = 300
1 = 600
2 = 1200
3 = 2400
4 = 4800
5 = 9600
6 = 19200

7 = 37600
8 = 57600
9 = 115200

Basic supports COM2 but only to baud rates of 9600. At this point we have not attempted to use Basic for any serial communication.

Borland Turbo C, which as we understand it, is the last remaining compiler to support DOS, supports all available com ports and baud rates. All of our communication programs have been written in Turbo C.

Serial port programming examples will be added at a future date.