Programming the RC2014 Front Panel

Part 1: BASIC

Bruce E. Hall, W8BH

INTRODUCTION

After 3 years of fun with my RC2014 Zed computer, I finally decided to put it in a proper enclosure - complete with front panel switches, LEDs, and a 20x4 character display. You can get yours at z80kits.com. Read on if you are interested in learning more about the front panel and how to program it. I will give programming examples in BASIC and Z80 assembly.

TWO BOARDS IN ONE

The front panel has two main components. The left half of the panel consists of 9 LEDs and 9 toggle switches. The right half of the panel consists of a 20x4 character LCD module. Each half has its own circuit board, and each half interfaces with the RC2014 via its own input/output ports.

THE FRONT PANEL SWITCHES

The LEDs and switches are used by RomWBW during system boot. Their functions are covered in the RomWBW User Guide.

The front panel switches, labelled 7 through 0, allow you to control system startup. If all switches are in the down position, the system starts in the usual, default manner. Switches 7 & 6 let you choose the console used. Keep both switches in the down position to use your system's primary serial port.

Switch 5, when flipped in the up position, continues startup by loading the boot device specified by switches 4 through 0. For instance, all 5 switches down will boot to the ROM monitor. I keep switches 4 & 5 in the up position to boot to CP/M 2.2, my usual destination.

If your "hard drive" is a CF/SD card, like mine, ***and*** you installed the standard Combo Hard Disk image on it, your boot choices will look the ones shown here:

Switches #2 - #0	Switch #4 Up (DISK)	Switch #4 Down (ROM)
DDD – 0	CP/M 2.2 (slice 0)	Monitor
DDU – 1	ZSDOS 1.1 (slice 1)	Rom BASIC
DUD – 2	ZCPR 3.4 (slice 2)	ROM Forth
DUU – 3	CP/M 3.0 (slice 3)	2048 Game
UDD – 4	ZPM3 (slice 4)	ROM CP/M 2.2
UDU – 5	WordStar (slice 5)	ROM ZSDOS 1.1
UUD – 6	*Cowgol (slice 6)*	Net Boot
UUU – 7	*HiTech-C (slice 7)*	ROM user program

Notice that I have added Cowgol and HiTech-C to slices 6 & 7 of my SD card. The front panel text does a good job summarizing your options. RTFM for more details.

The panel switches are wired to the input side of I/O port 0. I describe below how these switches can be used for your own purposes. But first, let's consider the panel LEDs.

THE FRONT PANEL LEDS

There are 8 data LEDs on the front panel. These LEDs are used by RomWBW to illuminate boot progression. If startup fails at any point, the LEDs will indicate at which step the failure occurred. After boot is complete, the LEDs are used to indicate disk activity: LED #2 indicates disk #2 activity, etc. Otherwise, the LEDs are idle. Let's see how we can put them to better use.

We can directly control the LEDs by writing to I/O port #0. Boot your computer to CP/M and start Microsoft Basic (MBASIC.COM). Here is a quick MBASIC refresher:

Run a basic program called TEST.BAS	MBASIC TEST
Quit BASIC and return to CP/M	SYSTEM
List a program	LIST
Save a program to disk using ASCII format	SAVE "FNAME",A (saves FNAME.BAS)
Load NEW.BAS while in MBASIC	LOAD "NEW" (loads NEW.BAS)
Stop a running program	<ctrl>C

I recommend saving your programs using all upper-case, so that CP/M handles them correctly. For now, just start Microsoft BASIC and enter the following command:

```
OUT 0,129.
```

This will output the value 129 on port #0 and should illuminate the left and right LEDs. You may think of the LEDs as representing a single byte of data, with the left-most LED representing the most significant bit (b7) and the right-most LED representing the least significant bit (b0). The decimal value 129 = binary 10000001.

Similarly, OUT 0,&H0F should illuminate the four rightmost LEDs. In this BASIC "&H" is used to indicate a hexadecimal value. OUT 0,15 will accomplish the same thing, since hexadecimal 0F = decimal 15. Try any 8-bit value you like, from 0 through 255. Finally enter OUT 0,0 to extinguish all the LEDs.

COUNTING

Here is a small BASIC program that displays an incrementing count on the front panel LEDs. Copy and paste it into your BASIC session or download it from GitHub.

```
100 FOR I=0 TO 255
110 OUT 0,I
120 FOR J=1 TO 200:NEXT J
130 NEXT I
140 GOTO 100
```

It contains a single loop which counts from 0 to 255, outputting the value to port 0 each time. Line 120 is a delay loop which slows down the counting frequency.

MAKE IT LOOK LIKE A COMPUTER

Real computers (you know, the scientific ones on TV) have lots of blinking lights, right? Check out this small but satisfying BASIC program, RANDOM.BAS:

```
100 I=INT(RND*256)
110 OUT 0,I
120 FOR J=1 TO 200:NEXT J
140 GOTO 100
```

Notice how, instead of counting 0 to 255, it uses a random value in that same range.

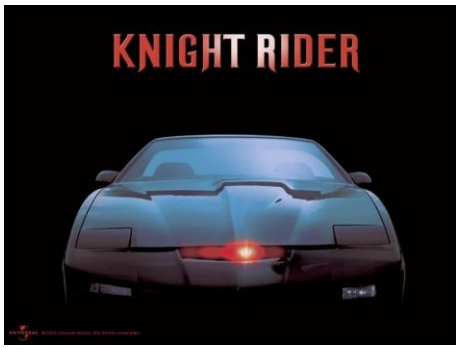
ANIMATE IT

It's time to animate our panel LEDs. Remember the Cylon from the TV series Battlestar Galactica? It had a red eye that shifted back and forth across its face. Some of you may also remember KITT in Knight Rider that used the same effect. Let's do that with our LEDs.

To start, set the right-most bit to 1 by sending the value 1 to port 0 (OUT 0,1). The next LED to turn on will be illuminated by issuing OUT 0,2. Every time we double the

value sent to Port 0, the displayed LEDs shift to the left. Conversely, the LEDs shift to the right when we halve the value. Here is a short BASIC program to demonstrate the phenomenon. You can copy this text and paste it directly into MBASIC or download the file CYLON.BAS from my GitHub pages.

```
ty100 BIT=1
110 FOR I=1 TO 8
120 OUT 0,BIT
130 BIT=BIT*2
140 FOR J=1 TO 100: NEXT J
160 NEXT I
170 BIT=BIT\2
210 FOR I=1 TO 6
220 BIT=BIT\2
230 OUT 0,BIT
240 FOR J=1 TO 100: NEXT J
260 NEXT I
300 GOTO 100
```



This simple program consists of two loops, each highlighted in yellow. The first loop outputs the bit on port 0 (OUT 0,BIT), then multiplies the bit value by 2. It does this 8 times, moving the illuminated LED from right to left. The second loop divides the bit value by 2 (BIT=BIT\2) instead, moving the illuminated LED to the right. Lines 140 and 240 are delay loops, used to oscillation frequency. What happens if you change Line 130 to BIT=BIT*2+1? Try it!

READ THE SWITCHES

The MBASIC keyword for reading an I/O port is INP(x), where X is the port number. Therefore, A=INP(0) will read the value represented by the 8 switches and put that value into variable A. An obvious choice is to read the switches and copy their value to the LEDs. It is a 3-line program in BASIC:

```
100 A=INP(0)
110 OUT 0,A
120 GOTO 100
```

We can use the switches to determine a course of action, such as choosing a program to run. Consider the following code:

```
100 B=INP(0) AND 7
110 ON B GOSUB 200,300,400,500
150 GOTO 100
```

It uses the switch input to determine which subroutine to run: a value of 1 branches to line 200, a value of 2 branches to line 300, etc. By logically ANDing the input result with 7 (binary 00000111), we limit input to the three right-most switches, ignoring the rest.

A full demo of the LEDs and switches is available on GitHub as FPDEMO.BAS.

THE 20x4 CHARACTER LCD

Writing single characters to the LCD is easy: OUT the character to the LCD's data port, which is 219. For example, fire up MBASIC again and issue the following command:

```
OUT 219,66
```

It outputs the value 66, which is the ASCII code for the letter B, and puts a "B" on the display. Note that OUT 219,ASC("B") does the same thing. Type the OUT command again, and a second B will appear on the display.

This is an 80-character display. To get a whole screen full of "B"s, send 80 of them:

```
10 FOR I=1 to 80
20 OUT 219,66
30 NEXT I
```

Sending text is not much harder. Send the string one character at a time. The following BASIC program, LCDTEXT.BAS, accepts text input and copies it to the LCD:

```
100 INPUT "Text to display";A$
200 OUT 218,1
210 L=LEN(A$)
215 IF L=0 THEN END
220 FOR I=1 TO L
230 C$=MID$(A$,I,1)
240 OUT 219,ASC(C$)
250 NEXT I
260 GOTO 100
```

The command OUT 218,1 sends value 1 (clear screen) to the command port of the LCD controller. Lines 230 and 240 are the important lines in this program, isolating each character and sending it to the display. After you've had your fill of sophomoric humor, it's time to continue…

THE HD44780 CHARACTER SETS

Here is a table of characters that our LCD can display. Numbers, uppercase letters, and lowercase letters are in their standard ASCII positions. There is a block of unused character positions in the middle at addresses 0x80 to 0x9F. On the right is a set of Japanese katakana characters. This 'A0' character set is the set available on my display.

Some controller chips are burned with a different character set, one that contains European and Cyrillic characters instead of the Japanese ones. Interesting! Which one do you have? Let's find out.

The easiest way is to send a single character to the display and see what it looks like. For example, character 0xE0 on my display is the Greek letter alpha (α). On a chip with the A2 character set, 0xE0 is an a-grave (à).

```
OUT 219,&HE0
```

Does your controller chip have A0 or A2? Now, just for fun, let's fill up the display with non-Latin characters beginning at ASCII &HA1. Check out NONLATIN.BAS:

```
100 C=&HA0
110 FOR I=1 TO 80
120 OUT 219,C+I
130 NEXT I
```

CUSTOM CHARACTERS

The LCD controller is great for ready-made alphanumeric characters: send the ASCII code for a character in ROM, and it is displayed on the LCD. But there is also a very small amount of RAM reserved for custom characters. The controller can hold up to 8 characters of your own design. To create and display these characters, use the following procedure:

- Design your 5x8 character
- Code it into 8 bytes, one byte for each row
- Save the data in the controller's 'character generator RAM'
- Repeat above steps for up to 8 characters
- Display the character by sending the corresponding code (0x00 through 0x07)

Let's try a useful example: a battery status indicator. We will create seven symbols, going from no-charge to full-charge. Fill in the outline of the no-charge symbol, assigning a '1' to a filled square and '0' to a clear square.

0	1	1	1	0
1	1	0	1	1
1	0	0	0	1
1	0	0	0	1
1	0	0	0	1
1	0	0	0	1
1	0	0	0	1
1	1	1	1	1

Here is the battery symbol on a 5x8 grid. There are 8 rows, and each row has 5 columns. The top row is binary value '01110' or hexadecimal &H0E. The second row is '11011' or &H1B. The green box below shows how we convert the binary ones and zeros of each row to their hexadecimal equivalents.

Row 1	0b01110 = &H0E
Row 2	0b11011 = &H1B
Row 3	0b10001 = &H11
Row 4	0b10001 = &H11
Row 5	0b10001 = &H11
Row 6	0b10001 = &H11
Row 7	0b10001 = &H11
Row 8	0b11111 = &H1F

Our 8-byte representation for the battery symbol, from top to bottom, are the values &H0E, &H1B, …, &H1F.

To create a full-charge battery, just fill in the middle rows. As you fill each middle row, its value will change from 0&H11 to &H1F.

The data for all 6 battery symbols, from empty to full, is here:

```
100 DATA &H0E,&H1B,&H11,&H11,&H11,&H11,&H11,&H1F: ‘empty battery
110 DATA &H0E,&H1B,&H11,&H11,&H11,&H11,&H1F,&H1F: ‘20% full
120 DATA &H0E,&H1B,&H11,&H11,&H11,&H1F,&H1F,&H1F: ‘40% full
130 DATA &H0E,&H1B,&H11,&H11,&H1F,&H1F,&H1F,&H1F: ‘60% full
140 DATA &H0E,&H1B,&H11,&H1F,&H1F,&H1F,&H1F,&H1F: ‘80% full
150 DATA &H0E,&H1F,&H1F,&H1F,&H1F,&H1F,&H1F,&H1F: ‘100% full
```

Each row of data corresponds to a battery symbol, from the no-charge symbol on top, to the full-charge symbol on the bottom.

Now that we have our artwork encoded, it’s time to load it into the character-generator RAM. Remember that we have 8 slots (of 8 bytes each), so our list of 6 symbols will fit. First, we issue a command to store data to the character-generator RAM of the HD44780 controller. Then, read all 48 data bytes and send them, one at a time:

```
190 OUT 218,&H40: ‘issue CGRAM command
200 FOR I=1 TO 48
210 READ BYTE
212 OUT 219,BYTE: ‘send data one byte at a time.
220 NEXT I
```

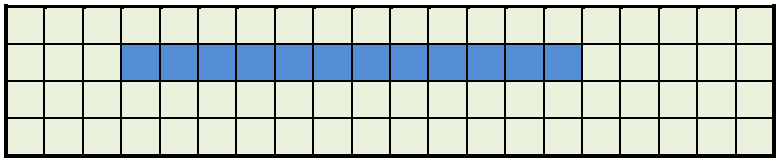
Run the statements above to load all 6 battery icons into the display controller. Then, while still in BASIC, enter OUT 291,0 to display the empty battery icon or OUT 291,5 to display the full battery icon. See BATTERY.BAS for a full program that animates the display.

SIMPLE HORIZONTAL BAR GRAPHS

An important requirement for creating detailed graphics is that the display device is dot-addressable. In other words, each display pixel can be individually addressed and programmed, separate from its neighbors. Sadly, our HD44780 controller does not give us a dot-addressable LCD display. We get only pre-determined characters, plus 8 characters of our own design. How can we possibly do artwork on that?

Well, we can’t. Go ahead, prove me wrong. Detailed graphics with this LCD module are devilishly hard to do. But that doesn’t mean we can’t create useful, simpler graphics. Bar graphs, for example.

First, consider a single, horizontal bar graph. Here is our 20x4 display, with a horizontal bar that is 12 characters wide. If we need to display data that over a small integer range, like 0-15, we can do it by repeating the solid block (0xFF) character for the desired length. You might code it like this:



```
100 ROW=1:COL=4:GOSUB 1000: ‘position cursor
110 FOR I=1 to 12
120 OUT 129,&HFF:  ‘display a solid block
130 NEXT I
```

This simple code works and is surprisingly useful. You aren’t limited to small ranges: just scale the desired range to 0-15 by the appropriate conversion factor. But your graph will always look a little coarse and chunky, since there are a limited number of possible data values/lengths.

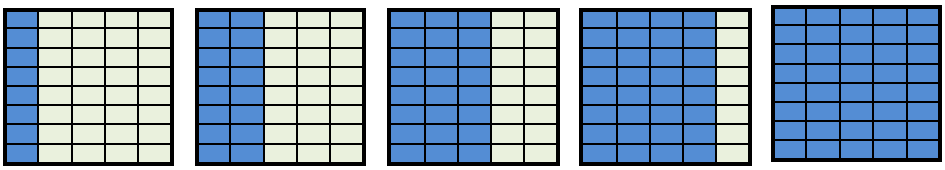
BETTER HORIZONTAL BAR GRAPHS

The graph will look better if we improve the horizontal resolution. But how? We can get a five-fold improvement in resolution if we use custom characters.

1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0

Consider the individual character. It contains 40 individual pixel “dots”, arranged in an 8 row, 5 column grid. We can’t access each individual pixel, but we can create custom symbols like this vertical bar. Display this one to the right of the 12-character bar above, and you’ve just made a bar of length 12.2!

Let’s make a set of vertical bar symbols, progressively increasing the number of columns in the symbol



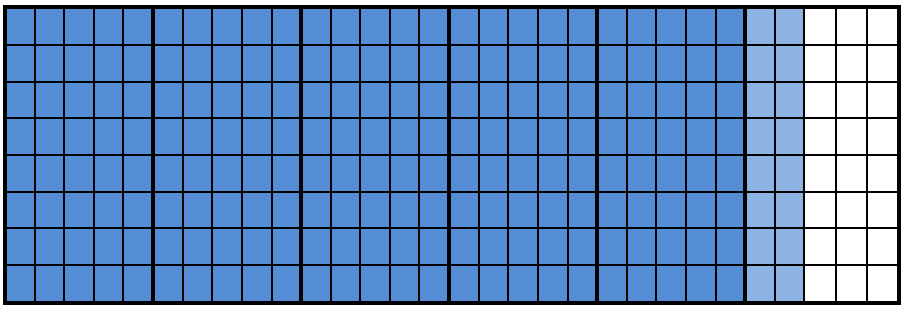
0.2 0.4 0.6 0.8 1.0

Now we can increase the length of our horizontal bar in fractions of a character, improving the horizontal resolution of our graph. It’s time to code it. First, create a set of symbols, like we did for the battery:

```
100 DATA &H10,&H10,&H10,&H10,&H10,&H10,&H10,&H10: ‘1 bars
110 DATA &H18,&H18,&H18,&H18,&H18,&H18,&H18,&H18: ‘2 bars
120 DATA &H1C,&H1C,&H1C,&H1C,&H1C,&H1C,&H1C,&H1C: ‘3 bars
```

```
130 DATA &H1E,&H1E,&H1E,&H1E,&H1E,&H1E,&H1E,&H1E: ‘4 bars
```

Now we need a routine to draw the horizontal bar for a given length. To simplify things, let’s stay with integer lengths, and give each vertical bar a length of one (instead of 0.2). In this system our original 12 character bar is 12*5 = 60 units long. Any length can be represented by a combination of ‘blocks’ which contain 5 bars each (symbol &HFF), followed by a terminal character containing less than 5 bars. For example, a bar graph of length 27 will be 5 full blocks (5*5=25), followed by a character containing the last 2 bars:



Bar of 27 units =

5 full blocks +
1 partially Filled block

We can calculate the number of full characters by integer division: length\5. And the number of bars in the final, partially filled character is just the remainder: length mod 5. A function for writing the horizontal bar sends the required number of solidly-filled characters, then a single, partially filled character. The entire listing for HBAR.BAS is below:

```
100 DATA &H10,&H10,&H10,&H10,&H10,&H10,&H10,&H10: '1 bars
110 DATA &H18,&H18,&H18,&H18,&H18,&H18,&H18,&H18: '2 bars
120 DATA &H1C,&H1C,&H1C,&H1C,&H1C,&H1C,&H1C,&H1C: '3 bars
130 DATA &H1E,&H1E,&H1E,&H1E,&H1E,&H1E,&H1E,&H1E: '4 bars
190 OUT 218,&H40: 'issue CGRAM command
200 FOR I=1 TO 32
210 READ BYTE
212 OUT 219,BYTE: 'send data one byte at a time.
220 NEXT I
300 INPUT"Bar length (0-100)";LENGTH
305 OUT 218,1
310 BLOCKS=LENGTH\5
320 BARS=LENGTH MOD 5
330 FOR I=1 TO BLOCKS
340 OUT 219,&HFF
350 NEXT I
360 IF BARS>0 THEN OUT 219,BARS-1
370 GOTO 300
```

From here you can create your own horizontal bar graphs. Or, with a different set of icons, create vertical bar graphs. Have fun. Program FPDEMO.BAS contains all the examples above. Continue to Part 2 for more examples in Z80 assembly language.

Bruce.

Last Updated: 30 Aug 2026