Epson PX-8 Tips and Tricks

Martin Hepperle, November 2019

Contents

1. General

The PX-8 was small laptop-like computer produced by Epson around 1983.

While the HX-20 ran a proprietary operating system, the PX-8 was based on the CP/M system. This allowed executing a wide range of application programs.

The dot matrix LCD screen shows a window of 8 lines by 80 characters into a virtual screen which can be configured in several ways. In addition to text it can also display graphics with its resolution of 480 × 64 pixels.

The built-in cassette drive can be used to store programs and data and additional devices like external RAM disks, flexible disk drive units, modem and a barcode reader were available.

The core of the operating system is stored in a ROM and two additional “ROM capsules” carry the adapted Microsoft BASIC and additional CP/M utilities. These “replaceable ROM capsules” are a feature which was not uncommon in those years. Several software programs like “Wordstar” were available. The user can also create his own (EP)ROMs with the desired programs and data. The same concept is also used in the PX-4. Other manufacturers of early laptops used a similar concept with replaceable EPROMs, e.g. Hewlett-Packard with the MS-DOS system “Portable Plus”.

The serial RS-232C interface can be used to communicate with other computers or printers and modems.

A second “high speed serial interface” was intended to connect external disk drives. Other applications are not directly supported by BASIC but the interface can also be used at a reduced speed of 4800 baud to connect a printer. Finally a connector for a bar code reader and an analog-to-digital converter port are built-into the box.

2. Using the MH-20 Disk Drive Simulator

The PX-8 can use internal or external RAM disks, but in both cases storage capacity is severely limited and volatile if the batteries fail. Thus Epson offered external disk drive units like PF-10, TF-15 or TF-20. These are mechanical delicate and difficult to find.

The MH-20 peripheral simulator, originally developed for the HX-20, has been adapted to support the PX-8 with four external disk drives. Each diskette has an unformatted capacity of 320 KBytes. Deducting the directory and system tracks about 278 KBytes of space are available to the user. The system tracks are not used and wasted.

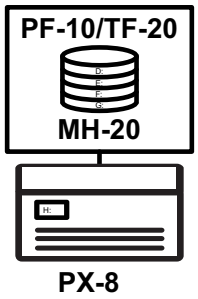

Figure 1: Connecting the MH-20 disk simulator to the PX-8.

The simulator runs on a PC and a simple connection cable between the serial port of the PC (or USB-to-RS-232C converter cable) and the PX-8 must be created. Figure 2 shows the wiring of this cable.

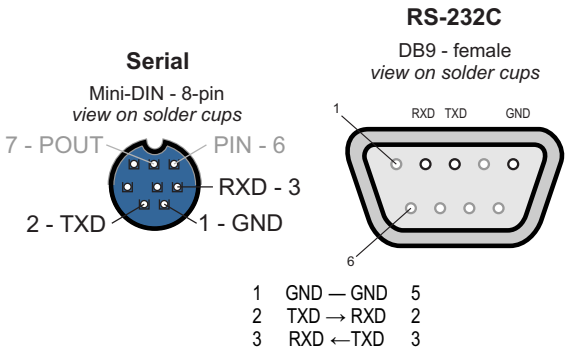

Figure 2: Left: Cable to connect the serial interface to a PC.
Top: The Mini-DIN connector must have a small outer diameter to fit into the case.

In contrast to the HX-20 the disk drive arrangement of the PX-8 can be configured and the drive letters seen in CP/M will depend on the configuration.

Therefore the simulator uses four numbered disk images "`PX_1.img`" to "`PX_4.img`". In a common standard configuration CP/M drive `D:` corresponds to "`PX_1.img`" and `G:` is mapped to "`PX_4.img`".

The simulator option `diskconfig` can be used to define the drive configuration. More information on command line parameters for the MH-20 software can be found in the "HX-20 Tips and Tricks" document.

The diskette images are simple raw images containing a sequence of logical CP/M records without any additional information or interleave. They can be easily read and written by other external tools.

Each disk drive symbol has a context menu which allows importing or exporting diskette images in D88 format. This format is used by the PX-8 simulator written by Toshiya Takeda.

Directory and file handling as well as the translation into logical records of 128 bytes are implemented inside CP/M. When the simulator is started, it looks for the four image files and reads them into memory – if no or not all four image files are found, it creates and formats new image files.

Note that these images are handled completely independent from the file system used for the HX-20, whose four drives are mapped to individual files residing in four subdirectories ("`DISK_A`" to "`DISK_D`"). There is no interference between the two, so that the PX-8 and the HX-20 can be used with the same installation of the MH-20 simulator.

If you want to use command line programs like the "CP/M-Tools" you can use the following disk definition for the image files:

```
# Epson PX-8 CP/M raw disk image
diskdef PX-8
  seclen 128
  tracks 40
  sectrk 64
  blocksize 2048
  maxdir 64
  skew 0
  boottrk 4
  os 2.2
end
```

Figure 3: Disk image parameters for cpmtools.

Some examples using `cpmtools`:

List files on disk image:

```
cpmls -f PX-8 PX_1.img
```

Copy a local file from the PC to the disk image (for user 0):

```
cpmcp -f PX-8 PX_1.img MANDEL_1.BAS 0:MANDEL_1.BAS
```

Copy a file (from user 0) from the disk image to a local file on the PC:

```
cpmcp -f PX-8 PX_1.img 0:MANDEL_1.BAS MANDEL_1.BAS
```

If you copy a file to the disk image file while the simulator is running, it will not notice this change. In this case you can use the context menu to "Reload" the modified image file into the simulator).

3. Using the MH-20 Display Simulator

Because I had already written my display controller simulator MH-20 for the HX-20 I wanted to use some of its functions also from the PX-8 and its Microsoft BASIC.

The HX-20 supports external display controllers through the high-speed serial interface directly from its Epson-modified Microsoft BASIC. However, the PX-8 only supports disk drives. This is a pity, as the operating system has all the core routines to send and receive the required EPSP packets – they are used for the external mass storage devices. I guess that the Epson engineers arrived at a point where the words of HP engineer Bill Wickes “life is short and ROM is full” became true again. So they omitted the graphics and text output functions for external screens from BASIC and also from CP/M

In general, the CP/M system is designed to make redirection of textual input and output easy. One could create a BIOS addition which would allow listing to an external screen and reading from an external keyboard. However, CP/M has no idea about graphics output so that I chose a different solution.

This is based on the assumption that most of the small programs will be written in BASIC. Thus BASIC callable functions would be convenient.

Microsoft BASIC offers two mechanisms for interfacing to machine language routines: the simple `USR` functions (which support one parameter only) and the `CALL` statements (which support multiple parameters). I decided to go with the `CALL` statement.

The `CALL` statement requires the address of the routine to call. It prepares the parameters for the subroutine and then calls the routine. Of course, the routine must be resident in memory.

On the PX-8 machine language routines can be loaded in two places:

- One can tell BASIC to reduce its size and leave some space above its allocated memory. The command line options or the `MEMSET` command are used for this purpose. Then one can load the machine language programs into this location, e.g. by using a sequence of `POKE` statements. Each time BASIC is terminated, the memory area is freed and is overwritten by other programs. Therefore the machine language program must be loaded every time when BASIC is started.

- The PX-8 offers another option which is called User-BIOS. The `CONFIG.COM` program allows reserving a memory block between the RAM-disk and the BDOS and BIOS. Like the RAM-disk this area is protected from being overwritten by a soft reset. This makes it possible to load a “permanent” system extension.

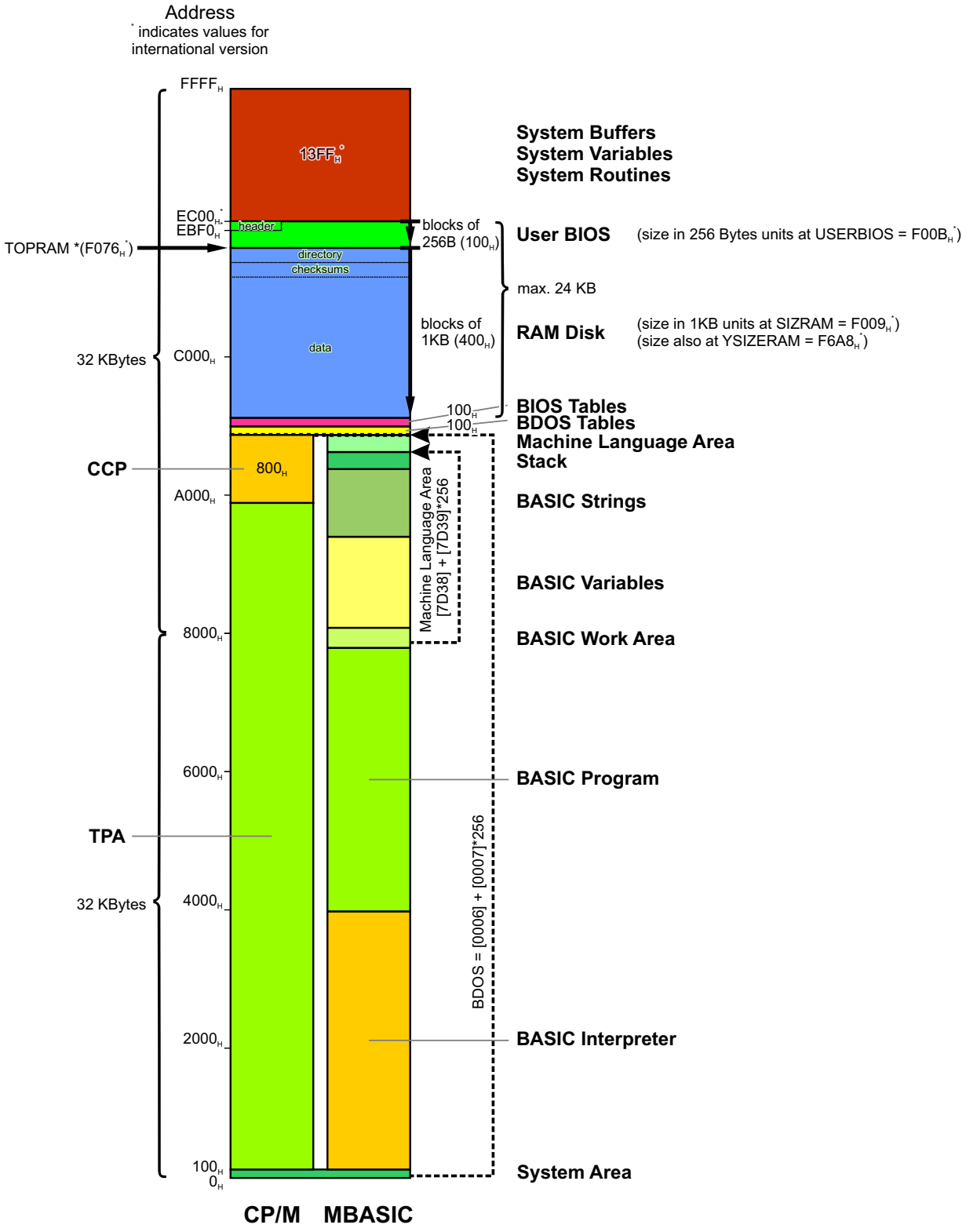

Figure 4: **Memory map of the PX-8 under CP/M and BASIC. The User-BIOS is located above the RAM disk and protected during normal CP/M system operation.**

I decided to use the User-BIOS area and to write a small extension which loads itself into this area. It contains routines which can easily be called from BASIC or other languages.

The usage of this extension requires the following steps:

- Use `CONFIG.COM` to reserve a User-BIOS area of 2 blocks (2 x 256 = 512 bytes).
- Load the extension into this area by simply executing `MHEXT.COM`. This will install the User-BIOS and replace the `PUNCH` routine.
  Executing `MHEXT.COM` again, will reload the User-BIOS and restore the original `PUNCH` routine.
- Use the correct addresses and parameter types for the individual routines.

Notes:

- It is absolutely necessary to use the given routine addresses (e.g. `&HEA35` for the `GMODE` routine). With an incorrect address you will crash your system. The names of the routines in the BASIC program are normal variables in BASIC and can be named as you like. They simply hold the addresses of the corresponding routines.
- It is equally important to use the correct parameter types – usually these are 2-byte Integer variables, which can be declared with a trailing percent "`%`" character.
- The graphics coordinates are given in pixels. The origin of the graphics screen is the upper left corner, `x` increases towards the right and `y` downwards.
- If you want to test whether the extension is properly installed, you can peek at the memory range from `&HEBF0` to `&HEBF9`. It should contain the string "`UBMH-AERO-`". "`UB`" for "User-BIOS" and "`MH-AERO-`" is my signature.

The extensions include the following functions for writing text to the external screen in Text mode and to draw lines in Graphics mode.

3.1.1. Text Mode Functions

```
TMODE = &HEA2C
CALL TMODE (BCOLOR% )
```

Mode	Can be used in Text or Graphics mode	
Description	Switch the display mode of the external display to Text mode.	
Parameter	`BCOLOR%`	[0,1,2,3]: the background color

```
TCURPOS = &HEA53
CALL TCURPOS ( C%, R% )
```

Mode	Text	
Description	Locate the cursor at a given column and row.	
Parameters	`C%`	the cursor column position
	`R%`	the cursor row position

```
TCHAR = &EA6D
CALL TCHAR ( C% )
```

Mode	Text
Description	Output a character given by its ASCII code at the current cursor position. You can use control characters like `12` to clear the test screen or `9` to tab forward.

Parameters	C%	[0…255]: the character code to print

3.1.2. Graphics Mode Functions

```
GMODE = &HEA35
CALL GMODE (BCOLOR% )
```

Mode	Can be used in Text or Graphics mode	
Description	Switch the display mode of the external display to GRAPHICS mode.	
Parameter	BCOLOR%	[0,1,2,3]: the background color

```
GLINE = &HEA00
CALL GLINE ( X0%, Y0%, X1%, Y1%, COLOR% )
```

Mode	Graphics mode	
Description	Draw a line with a given color.	
Parameters	X0%, Y0%	the start point in pixels
	X1%, Y1%	the end point in pixels
	COLOR%	[0…3]: the color index

```
GPEN = &EABD
CALL GPEN ( COLOR% )
```

Mode	Graphics mode	
Description	Select a pen. This pen is used for all following GDRAW commands.	
Parameter	COLOR%	[0…3]: the pens color index

```
GMOVE = &EA82
CALL GMOVE ( X%, Y% )
```

Mode	Graphics mode	
Description	Move the current pen position to the given point. Nothing is drawn on the screen. This command is useful in conjunction with the GDRAW command.	
Parameters	X%, Y%	the point in pixels

```
GDRAW = &EA8F
CALL GDRAW ( X%, Y% )
```

Mode	Graphics mode	
Description	Draw a line from the current pen position to the given point and update the current point. This function is useful when a larger polygon shall be drawn. The GLINE command requires the transfer of three additional and partially redundant parameters in this case.	
Parameters	X%, Y%	the point in pixels

3.1.3. Example

The following BASIC program contains a simple demonstration of these functions.

```
10 REM --------------------------------
20 REM DEMO for MHEXT (Epson PX-8)
30 REM MHEXT must be in USER BIOS area
40 REM Martin Hepperle, 2019
50 REM --------------------------------
60 REM The adresses of the functions:
70 TMODE=&HEA2C
80 GMODE=&HEA35
90 GLINE=&HEA00
100 CURPOS=&HEA53
110 TCHAR=&HEA6D
120 GPEN=&HEABD
130 GMOVE=&HEA82
140 GDRAW=&HEA8F
150 REM --------------------------------
160 REM Demo starts here
170 C%=0
180 CALL GMODE(C%)
190 REM GOSUB 670
200 X0%=240 : Y0%=400 : CALL GMOVE(X0%,Y0%)
210 C%=1 : R=200
220 FOR S=0 TO 32 STEP .15708
230 IF C%=3 THEN C%=1 ELSE C%=C%+1
240 CALL GPEN(C%)
250 X0%=240+R*SIN(S)
260 Y0%=200+R*COS(S)
270 R=R-1
280 CALL GDRAW(X0%,Y0%)
290 NEXT S
300 Y0%=2 : Y1%=400
310 C%=0
320 FOR X0%=1 TO 480 STEP 4
330 X1%=480-X0%
340 C%=C%+1
350 CALL GLINE(X0%,Y0%,X1%,Y1%,C%)
360 NEXT X0%
370 X0%=2 : X1%=480
380 FOR Y0%=1 TO 400 STEP 4
390 Y1%=400-Y0%
400 C%=C%+1
410 CALL GLINE(X0%,Y0%,X1%,Y1%,C%)
420 NEXT Y0%
430 FOR X%=0 TO 480 STEP 2
440 Y0%=150*(1+COS(X%*3.14/120)^5)
450 Y1%=150*(1+SIN(X%*3.14/120)^3)
460 IF Y0%>Y1% THEN C%=1 ELSE C%=0
470 CALL GLINE (X%,Y0%,X%,Y1%,C%)
480 NEXT X%
490 REM --------------------------------
500 C%=0
510 CALL TMODE(C%)
520 C%=12
530 CALL TCHAR(C%)
540 C%=12
550 CALL TCHAR(C%)
560 FOR C%=32 TO 127
570 X%=RND*80 : Y%=RND*24
580 CALL CURPOS(X%,Y%)
590 CALL TCHAR(C%)
600 IF (C% MOD 8) <>0 THEN GOTO 680
610 X%=1 : Y%=1
620 CALL CURPOS(X%,Y%)
630 T$=TIME$
640 FOR I=1 TO LEN(T$)
650 K%=ASC(MID$(T$,I,1))
```

```
660 CALL TCHAR(K%)
670 NEXT I
680 NEXT C%
690 X%=1 : Y%=1
700 CALL CURPOS(X%,Y%)
710 END
720 REM --- DEBUG print packet ---
730 R=&HEB0B
740 PRINT "EPSPKT: ";
750 FOR K=1 TO 16
760 PRINT HEX$(PEEK(R))+" ";
770 R=R+1
780 NEXT K
790 PRINT
795 REM --- DEBUG print receive buffer ---
800 R=&HEB1B
810 PRINT "RCVBUF: ";
820 FOR K=1 TO 16
830 PRINT HEX$(PEEK(R))+" ";
840 R=R+1
850 NEXT K
860 PRINT
870 R=&HEB33
880 PRINT "ERR: ";HEX$(PEEK(R))
890 RETURN
```

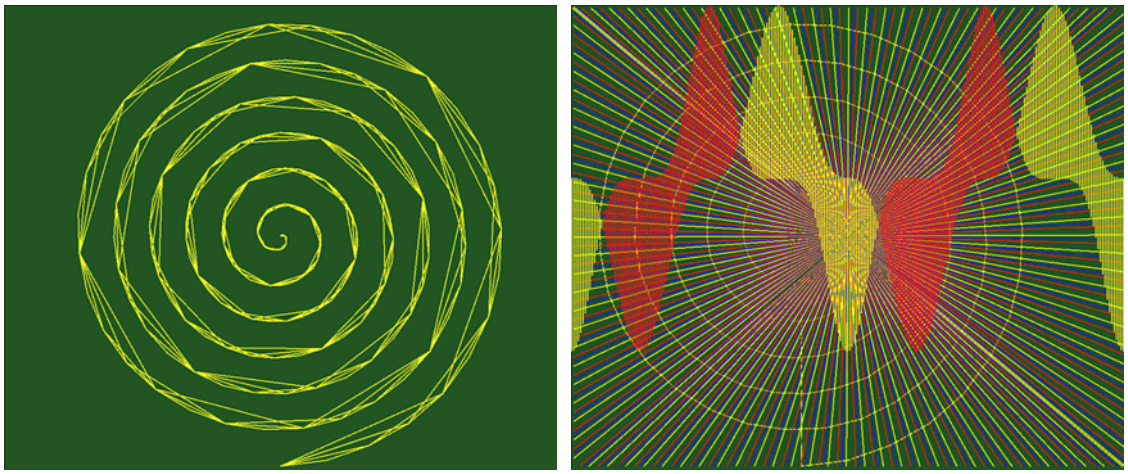

Figure 5: Some results produced by the PX-8 MHEXT-Demo program.

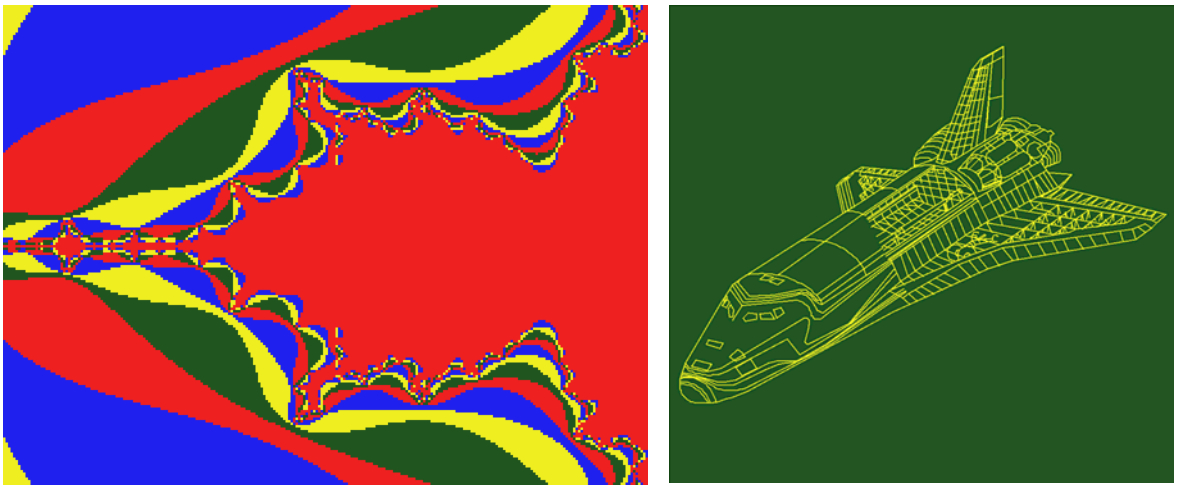

Figure 6: You can also dig deep into the red sea or fly high above it.

3.1.4. Text Output from Turbo-Pascal

The usual assignments of logical devices to physical devices are:

```
CON: is CRT:
RDR: is URl:
PUN: is UPl:
LST: is LPT:
```

You can use `WriteLn(Aux,'Hello MH-20, greetings from PX-8');` to send the string through the Serial interface and display it on the MH-20 simulator screen.

You can use `WriteLn(Lst,'Hello MH-20, greetings from PX-8');` to send the string to the RS-232C interface and display it in a terminal program. The baud rate of the RS-232C interface for printer output can be configured with the `CONFIG.COM` program.

Remember that pressing `^P` is the standard CP/M way to toggle duplicating `CON:` output to the `LST:` device, which is the RS-232C interface.

3.1.5. Source Code of Extensions

For completeness and archiving purposes a listing of the assembler code follows.

```
;
;  MH-EXTensions  MHEXT
;
;  Martin Hepperle, 2019
;
;  Some routines for the Epson PX-8 CP/M computer
```

```
;  to send EPSP commands to an external
;  screen simulator.
```

```
;
;  - Requires two pages (512 Bytes) User-BIOS RAM)
;    This space must be reserved with CONFIG.COM.
;  - Must be loaded once into User-BIOS area by
;    executing MHEXT.COM
;  - Note that all parameters MUST be given as
;    Integers (append % to the variable names).
;
; How to call from BASIC:
; ======================
; TMODE=&HEA2C
; CALL TMODE ( BGCOLOR% )
; Any mode: Switch the display mode of the external display
;           to TEXT mode.
; BGCOLOR% = [0,1,2,3] the background color
;
; CURPOS=&HEA53
; CALL CURPOS ( C%, R% )
; TEXT mode: Locate the cursor at a given column and row.
; C%, R%   the cursor position in columns, rows
;
; CHROUT=&EA6D
; CALL CHROUT ( C% )
; TEXT mode: Output a character given by its ASCII code
;            at the current cursor position.
; C%   the character code to print
;
; GMODE=&HEA35
; CALL GMODE ( BGCOLOR% )
; Any mode: Switch the display mode of the external display
;           to GRAPHICS mode.
; BGCOLOR% = [0,1,2,3] the background color
;
; GLINE = &HEA00
; GRAPHICS mode: Draw a line with a given color.
; CALL GLINE ( X0%, Y0%, X1%, Y1%, COLOR% )
; X0%, Y0%   the start point in pixels
; X1%, Y1%   the end point in pixels
;
; GPEN = &EABD
; GRAPHICS mode: Select a pen. This pen is used for
;                all following GDRAW commands.
; CALL GPEN ( C% )
; C%   the pen color index [0...3]
;
; GMOVE = &EA82
; GRAPHICS mode: Move the pen position to the given point.
; CALL GMOVE ( X%, Y% )
; X%, Y%   the point in pixels
;
; GDRAW = &EA8F
; GRAPHICS mode: Draw a line to the given point
;                and update current pen location.
; CALL GDRAW ( X%, Y% )
; X%, Y%   the point in pixels
;
; How to build and load:
; ======================
;
;  M80 MHEXT,MHEXT=MHEXT/Z
;  L80 MHEXT/N,MHEXT/X/E
;  MHEXT
;
BDOS    EQU   00005H  ; BDOS call
WBOOT   EQU   00001H  ;
SLVFLG  EQU   0F358H  ;
UBSIZE  EQU   0F00BH  ; size of User-BIOS (paras)
```

```
TEST    EQU   0       ; simulate BASIC call

; enable Z80 Zilog Mnemonics
.Z80
;--------------------------------
   ; check User-BIOS size
   LD   A,(UBSIZE)
   CP   2
   JP   Z,CKINSTALL   ; o.k.
   LD   DE,ERRMSG
   JP DONE
;--------------------------------
CKINSTALL:
   ; check whether new PUNCH routine is installed
   LD   IX,(WBOOT)    ; address of WBOOT routine
   LD   BC,0010H
   ADD  IX,BC         ; IX -> PUNCH
   LD   B,(IX+1)      ; get current PUNCH
   LD   C,(IX+0)      ; BC: current routine
   LD   DE,NEWPUN     ; DE: new routine

   LD   A,D           ; BC == DE ?
   CP   B             ; compare only high byte
   JP   NZ,NOTINSTALLED

; installed: restore from SAVPUN
   LD   BC,(SAVPUN)   ; old PUNCH routine
   LD   (IX+1),B      ; restore PUNCH
   LD   (IX+0),C      ; into BIOS
   CALL INSTALL
   LD   DE,UNSTMSG
   JP   DONE

NOTINSTALLED:
   CALL INSTALL
   LD   (SAVPUN),BC   ; save current PUNCH
   LD   (IX+1),D      ; install NEWPUN
   LD   (IX+0),E      ; into BIOS
   LD   DE,INSTMSG

DONE:
   LD   C,09H
   CALL BDOS

   JP   0000H         ; done
;--------------------------------
INSTALL:
   ; load code into USER -BIOS memory
   EXX                ; save BC, DE, HL

   LD  HL,SOURCE
   LD  DE,DEST
   LD  BC,LEN
   LDIR               ; copy LEN bytes to DEST

   ; fix target address call
   LD HL,(WBOOT)      ; address of WBOOT routine
   LD BC,0072H
   ADD  HL,BC         ; +72H
   LD   (ADDR+1),HL

   CALL CHKSUM

   EXX                ; restore BC, DE, HL
   RET
;--------------------------------
; calculate the checksum of the User-BIOS header
CHKSUM:
```

```
   LD  IY,HEADER
   LD  B,0FH          ; 15 bytes
   LD  A,00H          ; initial value
CRCLOOP:
   SUB (IY)           ; subtract byte from A
   INC IY
   DEC B
   JP  NZ,CRCLOOP
   LD  (IY),A         ; store checksum
   RET
;-------------------------------
ERRMSG: 'Error: User-BIOS of 2 paragraphs needed',13,10,'$'
INSTMSG: 'User-BIOS installed, PUNCH replaced.',13,10,'$'
UNSTMSG: 'User-BIOS installed, PUNCH restored.',13,10,'$'
;-------------------------------

; --------------------------
   ;ASEG
SOURCE:

.PHASE  0EA00H     ; to USER BIOS area

DEST:

;============================
GLINE:
   ; BASIC CALL GLINE ( X0%, Y0%, X1%, Y1%, COLOR% )
   ; HL -> X0
   ; DE -> Y0
   ; BC -> &X1, &Y1, &COLOR

IF TEST
   ; test setup
   LD HL,X0_          ; HL -> X0
   LD DE,Y0_          ; DE -> Y0
   LD BC,PKT_         ; BC -> &X1, &Y1, &COLOR
ENDIF

   ; --- get 2 bytes from  *HL to *IX
   LD   IX,DATA       ; destination address
   CALL GETWRD

   ; get Y0 from *DE
   PUSH DE
   POP  HL            ; move DE to HL
   CALL GETWRD

   ; get X1 from *HL <- BC
   CALL GETADR        ; HL now has address of X1
   CALL GETWRD

   ; get Y1 from *HL
   CALL GETADR         ; HL has address of Y1
   CALL GETWRD

   ; get COLOR from *HL
   CALL GETADR        ; HL has address of C
   CALL GETBYT        ; get C% from *HL

GLINEX:               ; entry from GDRAW
   ; prepare EPSP packet header
   LD A,0C8H          ; FNC=0xC8: draw line
   LD (FNC),A
   LD A,08H           ; send 9 data bytes - 1 = 8
   LD B,00H           ; want no reply
   LD C,01H           ; reply: 1 retcode = 1
   JP SENDPKT
;============================
TMODE:
```

```
   ; BASIC CALL TMODE ( COLOR% )
   ; HL -> COLOR%

   LD  A,01H           ; TEXT=1
   LD  (DATA),A
   DEC A               ; GRAPHICS=0

   JP SETMODE

;============================
GMODE:
   ; BASIC CALL GSMODE ( COLOR% )
   ; HL -> COLOR%

   LD  A,00H           ; TEXT=0
   LD  (DATA),A
   INC A               ; GRAPHICS=1

SETMODE:
   LD  (DATA+1),A      ; set GRAPHICS
   LD  IX,DATA+2
   ; get COLOR from *HL
   CALL GETBYT         ; get COLOR from *HL

   ; prepare EPSP packet header
   LD A,093H           ; FNC=0x93: set screen mode
   LD (FNC),A
   LD A,02H            ; send 3 data bytes - 1 = 2
   LD B,01H            ; want reply
   LD C,08H            ; reply: 1 ret + 1 header? +
                       ;        5 header + 1 data = 8
   JP SENDPKT

;============================
CURPOS:
   ; BASIC CALL CURPOS ( X%, Y% )
   ; HL -> X% (1 byte
   ; DE -> Y% (1 byte)

   ; get X% from *HL
   LD IX,DATA
   CALL GETBYT         ; get X% from *HL
  ; get Y% from *DE
   PUSH DE
   POP  HL             ; move DE to HL
   CALL GETBYT         ; get Y% from *HL

   ; prepare EPSP packet header
   LD A,0C0H           ; FNC=0xC0: set cursor position
   LD (FNC),A
   LD A,01H            ; send 2 bytes - 1 = 1
   LD B,00H            ; want no reply
   LD C,01H            ; reply: 1 retcode
   JP SENDPKT
;============================
CHROUT:
   ; BASIC CALL CHROUT ( C% )
   ; HL -> C%

   ; get C% from *HL
   LD IX,DATA
   CALL GETBYT         ; get X% from *HL
CHREXO:
   ; prepare EPSP packet header
   LD A,92H            ; FNC=0x98: write character
   LD (FNC),A
   LD A,00H            ; send 1 byte - 1 = 0
   LD B,01H            ; want reply
   LD C,09H            ; reply: 1 ret + 1 header? +
```

```
                        ;        5 header + 2 data = 9
   JP SENDPKT
;============================
GMOVE:
   ; BASIC CALL GMOVE ( X0%, Y0% )
   ; HL -> X0
   ; DE -> Y0
   ; --- get 2 bytes from  *HL CURX
   LD   IX,CURX       ; destination address
   CALL GETWRD

   ; --- get 2 bytes from  *DE CURY
   PUSH DE
   POP  HL            ; move DE to HL
   CALL GETWRD

   RET
;============================
GDRAW:
   ; BASIC CALL GDRAW ( X1%, Y1% )
   ; HL -> X1
   ; DE -> Y1

   PUSH HL            ; save

   ; --- get 2 bytes from  CURX to X0
   LD   HL,(CURX)     ;
   LD   (X0),HL       ;
   ; --- get 2 bytes from  CURY to Y0
   LD   HL,(CURY)     ;
   LD   (Y0),HL       ;

   POP HL             ; restore

   ; --- get 2 bytes from  *HL to X1
   LD   IX,X1         ; destination address
   CALL GETWRD

   ; --- get 2 bytes from  *DE to Y1
   PUSH DE
   POP  HL            ; move DE to HL
   CALL GETWRD

   LD  A,(CURC)       ; get color
   LD  (IX),A

   ; save position for next GDRAW
   LD  HL,X1
   LD  DE,CURX
   LD  BC,04H         ; 2 x 16 bit words
   LDIR               ; save X1 and Y1 in CURX/CURY

   JP  GLINEX

;============================
GPEN:
   ; BASIC CALL GPEN ( C% )
   ; HL -> COLOR
   ; --- get 2 bytes from  *HL CURX
   LD   IX,CURC       ; destination address
   CALL GETBYT

   RET

;-------------------------
SENDPKT:
   ; data packet is ready to go, load remaining data
   ; A=SIZ, B=RCVFLG, C=RCVLEN
   LD (SIZ),A         ; datalen - 1
```

```
   ADD A,08H          ; send: 1 6301 CODE +
                      ;       6 header + (SIZ + 1) data
   LD (SNDLEN),A
   LD  A,B
   LD  (RCVFLG),A     ;
   LD  A,C
   LD  (RCVLEN),A     ;

   LD  A,080H
   LD  (SLVFLG),A     ; activate slave

   LD  DE,SLVPKT
ADDR:
   CALL 0000          ; to be patched
   LD  (ERR),A
   LD  A,00H
   LD  (SLVFLG),A     ; deactivate slave
IF TEST
   JP  0000H
ENDIF
   RET
   ;-------------------------
GETWRD:
   LD  A,(HL)         ; get byte from *HL to *(IX+1)
   LD  (IX+1),A
   INC HL
   LD  A,(HL)         ; get byte from *HL to *IX
   LD (IX),A
   INC IX             ; prepare...
   INC IX             ; ...for next item
   INC HL
   RET
   ;-------------------------
GETBYT:
   LD  A,(HL)         ; get byte from *HL to *IX
   LD  (IX),A
   INC IX             ; prepare for next item
   RET
   ;-------------------------
GETADR:               ; read address *(BC) to HL
   LD  A,(BC)         ; and increment BC to next address
   LD  L,A            ; lo to L
   INC BC             ; next byte
   LD  A,(BC)
   LD  H,A            ; hi to H
   INC BC             ; ready for next address
   RET                ; HL has address
;-------------------------------
CURX:    DW 0000H     ; previuous X
CURY:    DW 0000H     ; previuous Y
CURC:    DB 01H       ; color
;-------------------------------
EPSPKT:
CODE:    DB 64H       ; 6301 CODE: send with header
RCVFLG:  DB 00H       ; 0: no reply expected, 1: receive reply
FMT:     DB 00H       ; 0: send from PX-8, 1: received by PX-8
DID:     DB 30H       ; destination: MH-20
SID:     DB 22H       ; sender: PX-8
FNC:     DB 00H       ; function code
SIZ:     DB 00H       ; len(DATA-1)
DATA:                 ; data[0-SIZ]
X0:      DB 00H,00H   ; 16-bit integer
Y0:      DB 00H,00H   ; 16-bit integer
X1:      DB 00H,00H   ; 16-bit integer
Y1:      DB 00H,00H   ; 16-bit integer
COLOR:   DB 00H       ; 8-bit integer
EPSEND:
;-------------------------------
RCVBUF:  DB  0,0,0,0,0,0,0,0   ; here: max 16 bytes
```

```
            0,0,0,0,0,0,0,0
RCVEND:
;-------------------------------
SLVPKT:  DW EPSPKT     ; address
SNDLEN:  DW 0          ; # of bytes to send
         DW RCVBUF     ; address
RCVLEN:  DW 10H        ; must be exact # of bytes to receive
SLVEND:
;-------------------------------
ERR:     DB   0
;-------------------------------
IF TEST
; BASIC SETUP with Integer values
XO_:     DB   00H,01H ; via HL
YO_:     DB   00H,02H ; via DE
X1_:     DB   00H,40H ; block via pointer in BC
Y1_:     DB   00H,80H ; block
C_:      DB   00H,01H ; block
PKT_:    DW   X1_,Y1_,C_ ; block with parameters 3...
ENDIF
;-------------------------------
SAVPUN:  DW   0000H    ; old PUNCH routine
;-------------------------------

NEWPUN:
   ; copy CHR% to DATA
   LD A,C
   LD (DATA),A
   JP CHREXO           ; return from there

RELEASE:               ; does nothing
   RET

   FILL  EQU TOPUSER-16-$    ; length of gap to fill
         DS 00B2H      ; to move SAVPUN and HEAADER to end
                       ; must equal FILL !

.DEPHASE
.PHASE 0EBF0H
; ORG      0EBF0H      ; to top of USER BIOS area

HEADER:
         DB "UB"       ; ID "User Bios"
         DB "MH-AERO-"; name
         DB 02H        ; size
         DB 00H        ; overwrite flag
         DW RELEASE    ; release address
         DB 00H        ; ZERO
         DB 00H        ; checksum
TOPUSER:
   LEN  EQU $-DEST     ; length of code
   LEN2 EQU $-SAVPUN   ; length of HEADER

   END
```

4. Opening the Case

Refer to the service manual to open the device. A few additional words may be helpful.

First unplug and remove the main battery to avoid short circuiting it. You should also switch off the backup battery using the slide switch in the battery compartment.

Before removing the printed circuit board you must remove the small cover to the right of the LCD display. It is held by a single screw from the back. Careful: the small spring pushing the display unlock slide lever may be easily lost. When this cover is removed you can easily unplug the display flexprint

cable. This is explained in the manual, but easily overlooked. Another spring loaded component to be lost is the rod which pushes the display lid open. It is held in place by the large hollow screw on the metalized insulation sheet.

Two additional flexprint connectors for the cassette drive and for the keyboard can be found under the bottom shell, but these can be unplugged from the bottom without the risk of damage.

5. Battery Replacement

The PX-8 has two batteries: a larger 4.8 V main battery which is identical to the pack used in the HX-20 except for the plug, and a second smaller 4.8 V pack which is soldered to the main board. The latter battery powers the RAM disk and maintains system settings. The charging circuits are designed for Ni-Cd cells so that Ni-MH cells can be used, but should not be left too long in the charging state.

You can compose a new main battery from four Sub-C Ni-Cd cells. These should be as short as possible as the compartment is rather tight. You could also use four slightly smaller cells – todays cells have higher capacity than the old Ni-Cd cells.

The internal buffer battery can be rebuilt from four cells of 1/3 AA Mignon size, which must be arranged in line. By soldering two leads cut e.g. from a resistor or capacitor you can solder it to the PCB like the original battery. Otherwise a strip of self adhesive foam tape may be your friend.

6. Capacitor Replacement

Like on the HX-20 of the same era all electrolytic capacitors in the PX-8 tend to leak. Especially on the larger capacitors this is visible in the form of white crystals on the leads and dull solder points. The smaller ones are also leaking, but due to the small amount of electrolyte this is less visible. Like in the HX-20 some capacitors are tricky to remove due to the small hole diameters in the PCB. Often, adding some fresh solder helps to remove the old solder. Replace all 21 electrolytic capacitors, ideally with miniature types. If you use larger ones you may have to mount them flat on the PCB to avoid interference with the case.

Designation	Capacity	Voltage	Dimensions
C1, C6, C7	47 µF	10V	∅5.5 × 9.5 mm
C2, C3, C4, C5	33 µF	10V	∅5.5 × 9.5 mm
C8, C9	220 µF	10V	∅8.5 × 9.5 mm
C10, C11, C74	10 µF	16V	∅5.5 × 9.5 mm
C12, C13, C14, C15	100 µF	16V	∅8.5 × 9.5 mm
C16	330 µF	16V	∅10.5 × 13 mm
C17	33 µF	25V	∅6.5 × 9.5 mm
C18, C19, C73	1 µF	50V	∅5.5 × 9.5 mm

Table 1: These electrolytic capacitors have to be replaced on the main PCB.

7. ROM Failure!

While I was working on my BIOS extension, suddenly after a few hours of work, the PX-8 would not access the `B:` and `C:` drives (ROM capsules) anymore. I always received a BDOS error message. It was still possible to access the RAM disk as well as external disks simulated with my MH-20 peripheral simulator. Argh!

After the first frustration I started to look into the Technical Manual. I learned that there is an extra power regulator which supplies 5 V to the ROM capsules. To conserve energy, this voltage is only provided when the ROMs are used.

Closer examination showed that a voltage was indeed applied each time together with a low voltage on the Chip Enable signal CE/ (Output Enable OE/ was always low).

However, on my PX-8 all I could measure was only about 2.5 V at the ROM sockets. No change was observed when the ROMs were removed. This low voltage was not enough to drive the ROMs.

The voltage must be applied rapidly when the ROM is activated and the high power demands of the ROM must be supported immediately. For this purpose the relatively complex power booster circuit driven by a clock of 35 kHz is built into the system. This circuit avoids that the power peak of switching the ROMs on leads to a voltage drop in the main system.

At the end of this circuit we find a Zener diode which stops the clock signal when the nominal voltage of 5 V is exceeded.

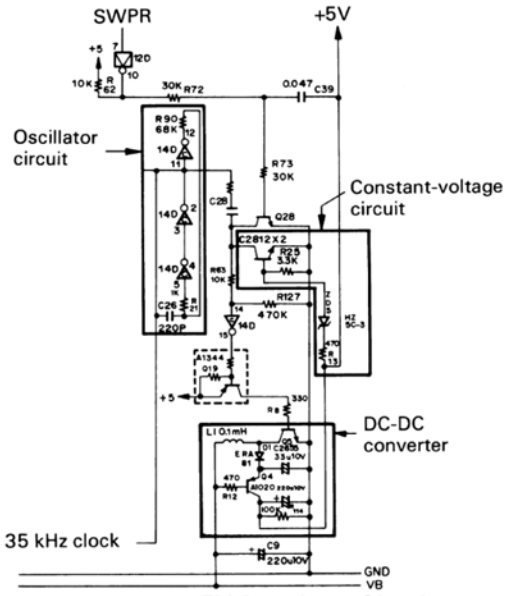

Fig. 2-14 +5V Regulator Circuit

Figure 7: The 5 V voltage regulator circuit.

Note the Zener diode inside the „Constant-voltage circuit“. The regulated output voltage is provided at the upper right. Q22 (below Q28) stops the clock leaving the „Oscillator circuit“ by connecting it to ground.

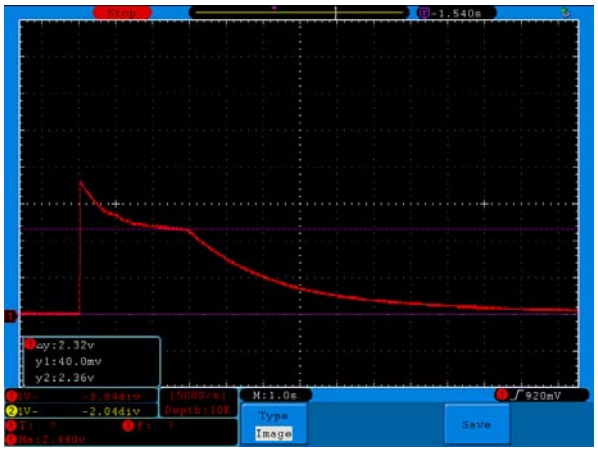

Figure 8: Initial configuration: ROMs and Zener Diode ZD5 installed. Voltage at ROM socket.

Here the ROM is activated by issuing a `DIR C:` command and after a short time (BDOS error) the voltage falls slowly back to zero, The voltage at VSS of the ROM socket is about 2.4V, with a short peak to 3.7 V. This is not enough for the ROM, it should see about 5V.

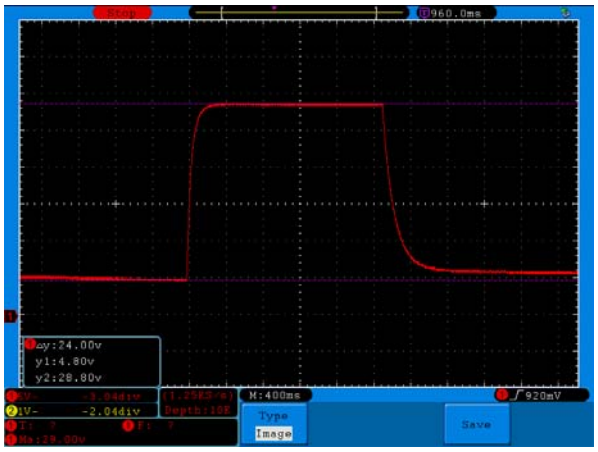

Figure 9: Configuration 1: Zener Diode ZD5 clipped, ROMs removed.

After removing the diode ZD5 (HZ 5C-3) we measure a voltage between 4.8 … 28.8V, ΔU = 24V at VSS of the ROM socket. This voltage is of course, too high, but demonstrates that the booster circuit works.

In order to test the power characteristics of the circuit, I added a 168 Ω resistor to simulate the load of one ROM capsule (about 30 mA).

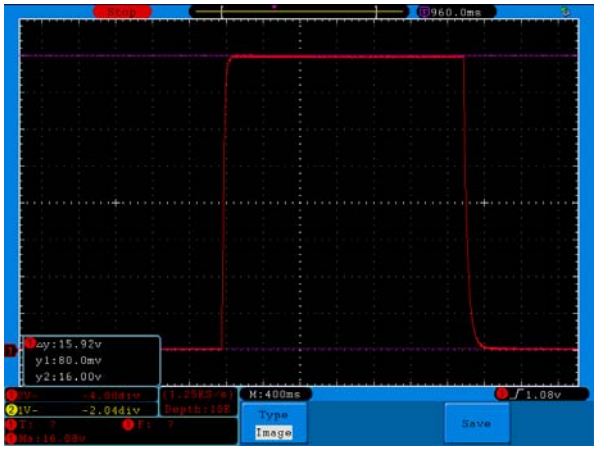

Figure 10: Configuration 2: Zener Diode ZD5 clipped, ROMs removed, load resistor added.

We now notice that the OFF voltage drops to 0V and the voltage in the ON condition reaches about 16V. This is still too high, but also shows that the free running circuit can supply the desired power.

Hoping that the troublemaker it was not Q22 but ZD5 the next logical step was to replace the Zener diode.

I only had a 5.1 V Zener at hand and replaced it. The load resistor stayed in place to be safe. Then a `DIR C:` command was issued again and the oscilloscope greeted me with a nice constant voltage under load of about 4.7 V. This is a bit on the low side (a 5.3 V or 5.6 V Zener would probably be a better replacement), but sufficient.

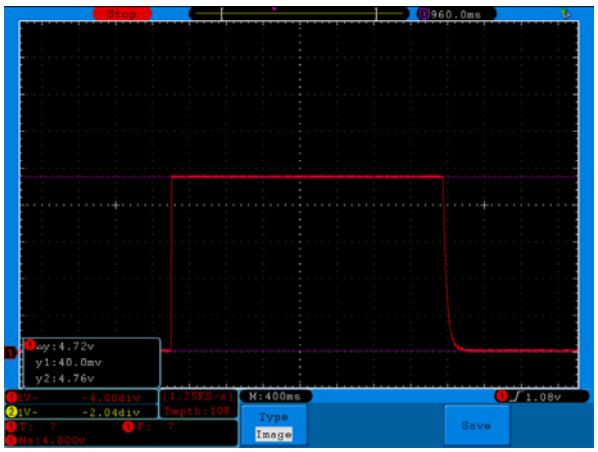

Figure 11: Configuration 3: Zener Diode ZD5 replaced by 5.1 V diode, load resistor added.

We also note the nice sharp corner when the circuit is started, delivering the required constant voltage almost immediately without exceeding the target.

Removing the test equipment and replacing the ROM capsules showed that I can now access `B:` and `C:` again. Phew!

Figure 12: The Zener Diode ZD5 in its natural habitat.

8. What about Speed?

For training I wrote an assembler version of the BYTE Eratosthenes Sieve benchmark and ran it to perform the required 10 loops. The PX-8 took about 12.5 seconds for this test. The time does not include the loading time from disk, which adds a few seconds to the total execution time.

The following source code can be compiled with the Microsoft M80 assembler.

```
;
;  BYTE Eratosthenes Sieve Benchmark
;
;  For Z-80 and CP/M
;
;  Martin Hepperle, 12/2019
;
   .Z80

   ASEG           ; COM file
   ORG 100H

BDOS   EQU   0005H
PRINTS EQU   0009H

START:

   LD   DE,MSG_1
   LD   C,PRINTS
   CALL BDOS


   LD    A,10      ; 10 iterations loop

NEXT:
   PUSH  AF

   LD   DE,0000    ; clear count
   LD   (COUNT),DE

   ; set all FLAGS to 1
   LD   A,1        ; fill value
   LD   DE,1       ; increment
```

```
   LD   HL,FLAGS   ; start and index
   LD   BC,FLEND   ; end
   CALL FILL

   ; loop
   LD   IY,0       ; i
   LD   HL,FLAGS
L2:

   LD   A,(HL)   ; get FLAGS[HL]
   CP   1
   JP   NZ,SKIP

   ; FLAGS[HL] == 1

   PUSH HL       ; --- save FLAGS+i

   PUSH IY       ; copy i...
   POP  IX       ; ...to IX
   ADD  IX,IX    ; i+i
   INC  IX
   INC  IX
   INC  IX       ; i+i+3
   PUSH IX
   POP  DE       ; increment = P
   LD   A,0      ; fill value

   ADD  HL,DE    ; FLAGS+i+P

   CALL FILL

   LD   DE,(COUNT)
   INC  DE
   LD   (COUNT),DE

   POP  HL       ; --- restore FLAGS+i

SKIP:
   INC  IY       ; i=i+1
   INC  HL       ; FLAGS+i
                 ; loop while HL < FLEND
   PUSH HL
   AND  A        ; clear Carry
   SBC  HL,BC    ; compare with FLEND
   POP  HL
   JP   C,L2     ; HL > BC


DONE:
   POP  AF
   DEC  A
   JP   NZ,NEXT

   LD   DE,(COUNT) ; should be 1899d = 076Bh

   LD   DE,MSG_2
   LD   C,PRINTS
   CALL BDOS

   JP 0000H

; -------------------------------
; fill memory starting at HL
; up to BC (exclusive)
; with value in A.
; increment is in DE
FILL:
   PUSH HL       ; save
   AND A         ; clear carry
```

```
   SBC HL,BC      ; compare with FLEND
   POP HL         ; restore
   JP  NC,FILEX   ; HL > BC
   LD (HL),A      ; set flag
   ADD HL,DE      ; increment HL=HL+De
   JP FILL
FILEX:
   RET

MSG_1:  'Sieve started.',7,13,10,'$'
MSG_2:  'Sieve stopped.',7,13,10,'$'

COUNT:  DB 0,0    ; will become 1899d

FLAGS:  DS 8191
FLEND:

   END
```

9. Connecting to a P-40 Printer

The Epson P-40 is a small, battery powered thermal printer. It has a very small buffer of two characters only. Therefore proper handshaking is essential for interfacing. The printer uses hardware handshaking by switching its DTR line. This line must be connected to the CTS line on the PX-8.

I prefer to use two cables: the first cable for the PX-8 provides an IBM-AT compatible 9-pin D-SUB connector and the second cable for the P-40 provides the matching counterpart. With this system I can easily connect most of my computers and devices. If you prefer a single cable you can combine both into a specific PX-8 – P-40 cable.

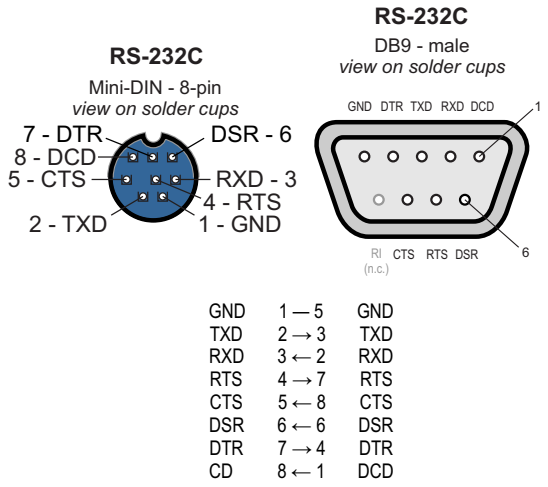

Figure 13: Cable to convert the RS232 interface to a PC-AT style layout.

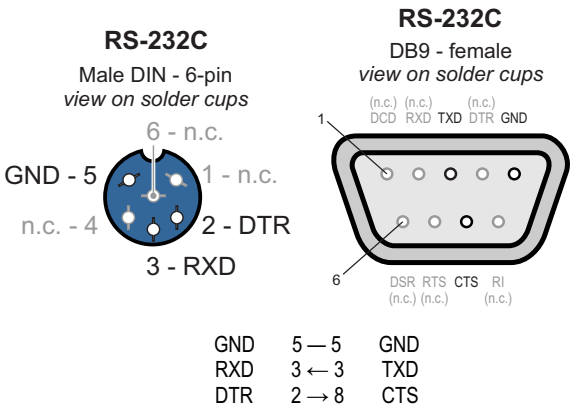

Figure 14: Cable to connect the PC-AT-Style connector to the P-40 printer.

10. Measuring Voltages

The PX-8 comes with a simple Analog-To-Digital converter built in. While its accuracy is rather limited to six bits, it is sufficient to test batteries and to perform simple tests. With a small battery it can also be used to interface to a potentiometer or a one-dimensional control device (e.g. a lever or a steering wheel).

Figure 15: Cable and plug box to connect 2 V or 5 V to the ADC input. The two resistors (e.g. 3.47 kΩ and 2.3 kΩ) must have a ratio of 1.5 to reduce 5V to 2V.

The converter reads up to 2V so that a voltage divider must be used to measure higher voltages. I designed a small box for connecting 2V and 5V input signals for quick testing of dry batteries and Lithium cells. A calibration with a digital voltmeter should be performed to define the final scaling constants.

The BIOS also offers a function to read the voltage of the PX-8 battery. Here an offset has to be taken into account, the constants in the program were derived from a graph in the Technical Manual.

The reading and display can be performed with a short Turbo-Pascal program.

```
{$C-,U-}   { for KeyPressed }
Program ADC;
Const
   ADC    = $25; { $6F / 3 }
   U_EXT  = $0000;
   U_BATT = $0003;
Var
   ADC_Value : Byte;
   GoOn      : Boolean;

Begin
   Write(Chr(27),'2');  { hide Cursor }
   Write(Chr(27),'*');  { clear screen }

   GoOn := True;

   While GoOn Do
   Begin
      If KeyPressed Then GoOn := False;

      Write(Chr(27),'=',Chr(32),Chr(32));   { home }

      { EXTERNAL Voltage }
      ADC_Value := Bios(ADC,U_EXT);
      WriteLn(ADC_Value/127.5:6:3,' V [2V external]');
      WriteLn(ADC_Value/51.7:6:3, ' V [5V external]');
```

```
      { BATTERY Voltage }
      ADC_Value := Bios(ADC,U_BATT);
      WriteLn(ADC_Value/41.84-0.479:6:3,' V [battery]');
   End;
   Write(Chr(27),'3');   { show Cursor again }
End.
```