Cross-programming in C on the Olivetti M20

by Davide Bucci

1 - Introduction

Last month, we saw an introduction to the Olivetti M20, a rather peculiar machine from 1982 [1]. We spent some time discussing the characteristics and the quirks of the Professional Computer Operating System or PCOS. This article describes how to use a comparatively modern C compiler (specifically, a version of GCC) to develop software for it.

One of the advantages of exploiting modern computers and tools to program vintage computers is that we now have beautiful text editors, excellent compilers and efficient languages. A purist may argue that the true "1980's experience" may be lost, but this is compensated by countless advantages, especially for relatively large projects. In the last few months, I have used this approach for developing text-adventures. I have been able to exploit the portability of the C language, targeting systems as different as the Sinclair ZX Spectrum and the Olivetti M20 with almost the same source code.

I will describe in this article how to cross-compile for the M20 in C on a Unix-like operating system such as Linux or macOS. I do not have any Windows machine around, but I think that for that operating system, programs such MinGW or Cygwin may be useful. A convenient strategy is to have on the developing machine both a compiler and an emulator running for each platform. The required tools (z8k-pcos GCC, m20disk, MAME) must be downloaded and in some cases built from sources, so I hope you will not be put off by things such as GNU make.

This article is organized as follows. I will start by briefly describing the compiler and the MAME emulator for the M20. Then, I will show how to use them to compile and run some introductory examples. I will finally discuss how to transfer the executables on the real hardware and run them there. I will finally discuss a nontrivial example (a small graphic demo) before drawing some conclusions.

2 - The C cross-compiler and the emulator

Many personal computers of the 1980's could be programmed in one of the many BASIC dialects available. The Olivetti M20 was no exception and came with a reasonably complete Microsoft interpreter, called BASIC-8000. Even if BASIC was a simple language and was easy to learn, it was painfully slow in some situations. Moreover, it was not very convenient for low-level operations, not efficient for large projects and severely limited in many areas. I started programming with BASIC on my VIC-20 when I was a child and I used it for many years on the PC too, but I am not very fond of it. An assembler suite for the Z8001 was available for the M20, but handling large projects in assembly is often tedious, cumbersome and the code is not portable, even if one can possibly write extremely compact and efficient programs.

From the modern perspective, the C programming language offers a good trade-off between execution speed, ease of coding and overall efficiency on limited machines, being a remarkably efficient compiled language. I will not describe here the strengths and pitfalls of the C language (many resources and tutorials are available on the Internet for that), but modern compilers targeting 8- and 16-bit processors exist. These are for example the cc65 for the 6502, the z88dk for the Z80, etc... For the Olivetti M20, much work has been done in this direction by Christian Groessler over several years. He created a version of GCC 2.9 dedicated to the Zilog Z8001 processor and PCOS, from a compiler originally put together in 1998 by the eCos group (then part of RedHat). His work included GNU binutils as well as newlib.

GCC 2.9 does not support all the bells and whistles of recent standards for C and C++, but it is still a very decent compiler, much more powerful than the original Microsoft BASIC available on the machine. Christian distributes the compiler along with its sources for many Unix systems on his FTP site [2], and wrote an introductory article, which is available at [3].

One possible strategy to install the compiler is to use one of the available binary distributions (Chris kindly prepared packages for many Un*x flavours), or directly compile it from sources. Once everything is done, you should install the executables in /usr/local/bin or make sure that they can be reached via current shell path. If the install has succeeded, typing the following command should yield the compiler version, as follows:

```
$ z8k-pcos-gcc --version
2.9-ecosSWtools-990319-m20z8k-3
```

The compiler suite is composed of a collection of tools that appear familiar if you are used to GCC. There are versions that are dedicated to COFF executables, but we will not use them on the M20. The tools dedicated to PCOS start with the z8k-pcos prefix.

Probably, the most convenient way to cross-develop for

a vintage computer is to have an efficient compiler paired with an emulator, both available on the modern machine. The second tool we are going to use is therefore MAME, as from version v0.212, it started to partially support the M20. The implementation is still slightly buggy, but remains quite useful to rapidly test simple programs.
Benjamin Eberhardt has written a very interesting article about how to use MAME to emulate an M20 [4].

MAME can be downloaded at [5] and among other persons, many of the efforts done to emulate the M20 have been done (once again) by Christian Groessler. After the download, you will need a copy of the boot ROM code that can be found at [6], as well as an image of a boot disk containing PCOS, such as the one that is present in the archives associated to this article [7]. Once MAME is installed on your system and you have put the M20 ROM in the current directory, it can be launched with a command that has the following structure:

```
$ mame m20 -bios 0 -rompath . -flop1 <image1>
-flop2 <image2> -window
```

Now that the main tools we need are ready, in the next paragraph we are going to discuss, compile and run some simple C programs on the emulator.

3 - Three 'Hello World' programs

Of course, the first program that one may use to test the compiler toolchain is the very well-known Hello World program:

```
#include<stdio.h>

int main(int argc, char **argv)
{
    printf("Hello World!\n");
    return 0;
}
```

Figure 1: Hello world output in MAME

If we call this file hello.c, the command to compile it is:

```
$ z8k-pcos-gcc hello.c -o hello.cmd
```

A rather unpleasant surprise is that the executable is 16211 byte long. If it is tiny for today's standards, it is relatively large for a 1982 computer and this size is not acceptable for such a simple program. We must mitigate this problem.

The culprit is the standard library and in particular the implementation of the printf function. This function offers very flexible formatting capabilities, at the price of substantial code to be included in the executable. It is worth noting that, even if the C compiler supports floating point types such as double and float, the present implementation of scanf and printf does not handle it. For many practical purposes however, if one does not need the formatting capabilities, printf can be skipped completely. A more manageable 9963 byte long executable can be obtained with the following code:

```
#include<stdio.h>

int main(int argc, char **argv)
{
    fputs("Hello World!\n", stdout);
    return 0;
}
```

To further shrink the size of the result, an interesting technique (that makes the code non portable) is to exploit a direct PCOS system call:

```
#include<sys/pcos.h>

int main(int argc, char **argv)
{
    _pcos_dstring("Hello World!\r");
    return 0;
}
```

Once compiled, this code yields a much more manageable 2919 byte executable. This size is still much greater than the one that can be obtained with a pure assembly program, but can be acceptable. A list of the PCOS functions callable from C can be found in the pcos.h header, which closely follows the description done by Olivetti in the manual dedicated the assembly language suite [8]. The -Os and -O2 options of gcc can be used and tell the

compiler to optimize the code respectively for code or for speed. In both cases, the simple "Hello World!" program yields a 2897-byte executable. Note in the last example the use of the \r code, the newline used by PCOS in place of \n.

In my experience, it is a good practice in C to adopt a modular strategy and keep separated from the program core the routines related to input and output. When porting a relatively large program to a new computer, the latter often require an adaptation. Non portable code (such as PCOS system call) shall be confined in this part of the code.

If you would like to mix Z8001 and C code, or if you want to use the z8001-pcos-as assembler alone, this is perfectly possible. The compiler manual [9] includes some detailed instructions about how to do that and contains many example programs. If you are used to the Z80 assembly, you may find it interesting to learn the Z8001, as it was meant to be the 16-bit successor to the Z80, exploiting a segmented memory paradigm and preserving a certain degree of compatibility.

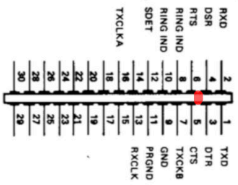
4 - Executing programs in MAME

In order to execute the Hello World program described above, we need to transfer it first into a usable disk image. MAME can read different types of disk images, the most useful file format to be used with the M20 has the extension IMG (in some older versions of MAME, only those in the MFI format could be written). There is a certain number of details to be considered when creating usable disk images, due to the head 0/track 0 format that is different from the rest of the disk in the PCOS disk organization. As said previously, a good bootable image that can be used with the emulator is the pcos20.img file, contained in the archive available from [7].

We are going to need the m20floppy utility described in [10]. Download and compile it with make, in order to obtain an executable called 'm20'. Once created and installed in your computer, to obtain a disk image called hello.img, type:

```
$ m20 hello.img new
```

By the way, m20floppy supports several commands: launch it with no arguments to obtain a brief description of each of them. At this point, the disk image is not yet usable, as the utility does not create the contents of head 0/track 0. They must be transferred manually from a disk image that contains them. The disk image example.img present in the same archive as the PCOS disk can be used for that:



Female DB9	M20 connector
2 (RXD)	1 (TXD)
3 (TXD)	2 (RXD)
6 (DSR)	3 (DTR)
4 (DTR)	4 (DSR)
7 (RTS)	5 (CTS)
8 (CTS)	6 (RTS)
5 (GND)	9 (GND)

Figure 2: RS232 cable pinout

```
$ dd conv=notrunc if=example.img of=hello.img
bs=4096 count=1
```

Benjamin Eberhardt suggests in [4] a simple way to check if a disk image contains the data corresponding to head 0/track 0 or not. You have to inspect the first bytes of the file to see if they are different from zero. If the dd command was successful, here is what you should obtain from an image that can be successfully used in the emulator:

```
$ hexdump hello.img |head -n 1
0000000 01 04 00 23 02 10 01 00 00 0a 00 c4
00 86 1e 00
```

and here is the result with an image that can not be used, as track 0 data is missing:

```
$ hexdump bad.img |head -n 1
0000000 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00
```

When you have a complete image of an empty disk, you may want to copy the result to another file, to avoid having to repeat the process each time. We can then add the executable program to the disk image:

```
$ m20 hello.img put hello.cmd
```

You may check the contents of a disk image using the ls command of m20floppy:

```
$ m20 hello.img ls
hello.cmd
```

Once the disk image contains the executable, we are going to launch the emulator in a window, with a system disk pcos20.img in the disk drive 0: and the image hello.img in drive 1:. If both files are available in the current directory that also contains the ROM file m20.zip, the command to launch MAME is:

```
$ mame m20 -bios 0 -rompath . -flop1 pcos20.img
-flop2 hello.img -window
```

If you have put the files elsewhere, change their paths accordingly. In the emulator, after the machine has finished booting, we can type 'hello' (or simply 'he') and the Hello World program should be executed, as shown in figure 1. One may notice that we did not have to select the drive, as one of PCOS's quirks is that if a file is not found in the current drive, the other one is scanned, too. The last accessed drive becomes the current one. If you have problems with the keyboard layout, you can mitigate them by running 'sl' that allows you to change the current language. The command 'ps' saves the current PCOS configuration and the save will be permanent, as recent versions of MAME can write to file images in the img format. If you want to have descriptions of the error messages more explicit than a numerical code, you can use the 'ep' command, at the expense of 1240 bytes of free RAM. If you are getting mad at the backspace key apparently misbehaving like a Carriage Return, in [1] I suggested a simple fix for that.

The current state of the MAME emulation of the M20 is that many things can be done, but the emulation may be unstable (a warning message is in fact issued by MAME). The emulator is invaluable nonetheless for preliminary testing, as transferring files to a real machine is not entirely trivial and requires some time and effort, as we are going to see in the next paragraph.

5 - Transferring files to a real M20

There are different strategies available to transfer files towards a real M20. If you have an MS-DOS computer with a 360 KB floppy disk drive, you can use Dwight Elvey's wrm20 and rdm20 routines, as described in [11]. There are limitations, mainly because of the peculiar formatting of the track 0/head 0, that is not handled by many disk controllers in the PC world. Usually, a way to circumvent them is to format a disk on the M20 and write it on the PC using Dwight's tools, which simply skip the tracks that can not be written.

Figure 3: File transfer in action

In my case, I do not own a suitable PC and I preferred to make an RS232 null-modem cable to attempt data transfer with protocols such as XMODEM. Figure 2 shows the connections of the cable. I represented the numbering of pins in a male DB9 connector as they appear this way on the solder side of the female connector to be used for the cable. On the "modern" side, I used an USB-RS232 interface that I bought many years ago, working reliably with macOS. I wrote a small collection of utilities in BASIC described in [12] that can be used for this task. Starting from scratch may involve copying a XMODEM receive program on the M20 and then use it to transfer the more involved tools. Instead of directly typing the program, once the M20 is connected, one may redirect the input and output of the PCOS towards RS232 with following commands:

```
pl ci
rs
sc com:,9600,none,0,8,half,off,256
ci 0,o,0
+Scom:, +Dcom:
```

The first commands load the 'ci' utility as well as the RS232 driver into memory and configure the M20 for a 9600 baud 8N1 connection, with no echo nor XON/XOFF control. Then, a serial connection is open. Finally, the last command redirects input and output towards RS232. On your modern computer, if you configured the terminal program correctly (I use Minicom), you should see the PCOS prompt appearing in your terminal, replicating what the M20 writes on the screen. This is a quite convenient way to use the M20, as you can control the computer remotely. You can for example launch basic by typing 'ba' and copy/paste the whole xreceive.bas program. To do this, you should first configure your terminal program to apply a delay for each key. BASIC is not fast enough to process data continuously fed by a modern computer and the result would become mangled after a few lines. On Minicom for example, type CTRL+A, then T and set the 'TX delay' to 10 ms. You may save the transferred file (as 'xmodem.bas'), then restart the machine and reissue the first four commands seen above (as the I/O redirect must not be active to transfer files with XMODEM) and finally load and run 'xmodem.bas' within BASIC to transfer files.

Figure 3 shows a file transfer between my MacBook pro using Minicom and the Olivetti M20, thanks to a USB to RS232 interface and the cable I built. Figure 4 shows the Hello World program running on my machine.

6 - A non-trivial example: memory access for graphics

Of course, programming in C offers a great deal of

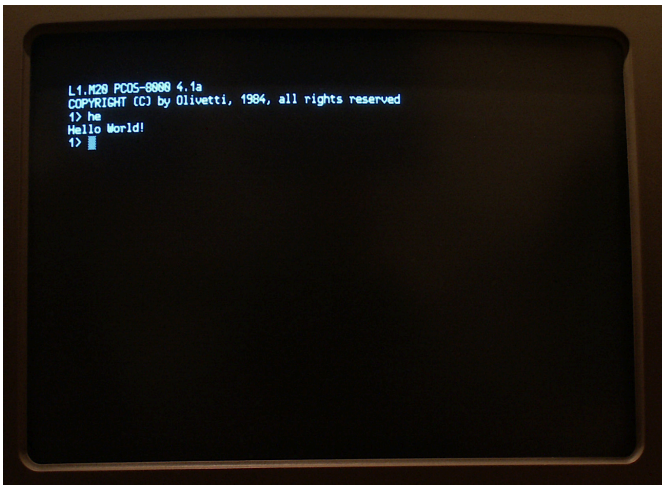


Figure 4: Hello world on a real M20

possibilities and the code in listing 1 shows two functions and a macro that can be used to draw on the screen by directly accessing the memory (on a B/W machine):

- The 'scrclear' function clears the screen (there is no difference between graphics and text modes on the M20, the screen always displays graphics).
- The 'PSET_M' macro draws a pixel on a grid of 512x256.
- The 'line' function draws a segment with the Bresenham algorithm [13].

The result can be seen in figure 5. Of course, such an implementation may be improved, but gives an idea of the expressiveness of the C language. If you really feel the need to get your hands dirty, the gcc manual [9] describes in detail the integration of C code with Z8001 assembly, taking for example different versions of the 'scrclr' function. As said earlier, if you already are familiar with the Z80 assembly language, you may find yourself at home with the Z8001, after all. Among the tools that come with the GCC compiler, the z8k-pcos-as assembler is quite powerful and convenient.

Conclusion

In this article, we briefly described how to cross-program the Olivetti M20 focusing on the C language. After a short introduction, we discussed the tools that we choose for the task, namely a special version of GCC tailored for the Z8001 processor and the PCOS operating system, as well as the MAME emulator.

We then introduced the classic Hello World program and we saw how to reduce the size of the executable produced by the compiler. We discussed how to execute it in the emulator and how to transfer files on a real machine. We finished our discussion by presenting an example of direct memory access. The compiler manual [9] written by Chris is definitely worth reading if you want to go beyond what I describe in this article.

By the way, I almost forgot! This article (as the one you read last month) has been entirely written using Oliword on my Olivetti M20. Text files have then been transferred using RS232 on a modern MacBook Pro, where the final editing has been done.

All the source code discussed in the article is contained in an archive available at [7]. It contains disk images of the discussed examples, as well as the Olivetti M20 version (they are available for many 8 and 16 bit computers) of two text adventure games I developed: The Queen's Footsteps and Two Days to the Race. Enjoy!

Acknowledgments

I would like to thank Christian Groessler for the amazing tools, the constant commitment to the M20, as well as for the countless fruitful discussions we had in the last fifteen years. Concerning the MAME emulator, I could never be able to emulate an M20 without the help of Benjamin Eberhardt, to whom I would like to express my gratitude. Benjamin also kindly prepared the PCOS disk images in the IMG file format and provided useful remarks on early versions of this article.

This paper would have been probably awkward to read without the kind and attentive proofread by Chris Carter. The remaining errors are mine.

BIBLIOGRAPHY

[1] D. Bucci "The Olivetti M20 and the history of a website" RetroMagazine World #2, August 2020.
[2] C. Groessler, personal FTP site: ftp.groessler.org
[3] C. Groessler, D. Bucci "Cross-programming for the Olivetti M20 using GCC," available at http://www.z80ne.com/m20/index.php?argument=sections/download/z8kgcc/z8kgcc.inc
[4] B. Eberhardt, "Emulating the M20 with MAME," available at: www.z80ne.com/m20/index.php?argument=sections/tech/mame_m20.inc
[5] MAME official website: https://www.mamedev.org
[6] Olivetti M20 ROMs available at https://wowroms.com/en/roms/mame/olivetti-l1-m20/89051.html
[7] http://www.retromagazine.net/download/m20inC_sources_and_disk_images.zip
[8] Olivetti "M20 Assembler language user guide," release 2.0, March 1983
[9] C. Groessler, "GCC Z8001 user manual," 2009, available at: http://www.z80ne.com/m20/sections/download/z8kgcc/z8kgcc.pdf
[10] C. Groessler, "Manipulate disk images," available at: http://www.z80ne.com/m20/index.php?argument=sections/transfer/imagehandle/imagehandle.inc
[11] D. Elvey "How to read and write disk images for

the M20 system," available at: http://www.z80one.com/m20/index.php?argument=sections/transfer/imagereadwrite/imagereadwrite.inc
[12] D. Bucci "Transferring files using a RS232 connection" http://www.z80one.com/m20/index.php?argument=sections/transfer/serial/serial.inc
[13] N. Johnson, "Advanced Graphics in C," ed. Osborne, McGraw-Hill 1987. http://www.z80one.com/m20/index.php?argument=sections/download/z8kgcc/z8kgcc.inc

Listing 1: C code for direct access to video RAM

```
/* Segment #3: begin of video RAM for a
B/W machine*/
unsigned short *screen = (unsigned
short *)0x3000000;

#define SCREEN_WIDTH    512
#define SCREEN_HEIGHT   256
#define SCREEN_SIZE     (SCREEN_WIDTH /
16 * SCREEN_HEIGHT)   /* words */
#define ABS(a) ((a)>0 ? (a):(-a))
#define MAX(a,b) (((a)>(b))? (a):(b))
#define SIGN(a) ((a)>0 ? 1 : ((a)==0 ?
0 : (-1)))
#define TRUE -1
#define FALSE 0

/*  Fills the screen memory with a
defined word. */
void fillscr(unsigned short p)
{
    unsigned short *s;
    for (s=screen; s <
screen+SCREEN_SIZE; ++s)
        *s = p;
}

/*  Just turn on a pixel by accessing
directly to the video RAM. */
#define PSET_M(x,y) *(screen +
(((y)<<5) | ((x)>>4)))|=1<<(15-((x) &
0x000F))

/*  Plot a line using the Bresenham
algorithm.
    from Nelson Johnson, "Advanced
Graphics in C"
    ed. Osborne, McGraw-Hill 1987. */
void line(unsigned short x1, unsigned
short y1,
    unsigned short x2, unsigned short
y2)
{
    short dx=x2-x1, dy=y2-y1;
    short ix=ABS(dx), iy=ABS(dy);
    short inc=MAX(ix, iy), plotx=x1,
ploty=y1, i, plot;
    short x=0, y=0;

    PSET_M(plotx,ploty);  /* Plot the
first pixel */
    for(i=0; i<=inc; ++i) {
        x += ix;
        y += iy;
        plot=FALSE;
        if (x>inc) {
            plot=TRUE;
            x-=inc;
            plotx+=SIGN(dx);
        }
        if (y>inc) {
            plot=TRUE;
            y-=inc;
            ploty+=SIGN(dy);
        }
        if (plot)
            PSET_M(plotx,ploty);
    }
}

int main(int argc, char **argv)
{
    int i;
    fillscr(0);
    for (i=0; i<512; i+=10) {
        line(0,0,i,128);
        line(0,255,i,128);
        line(511,255,i,128);
        line(511,0,i,128);
    }
    return 0;
}
```

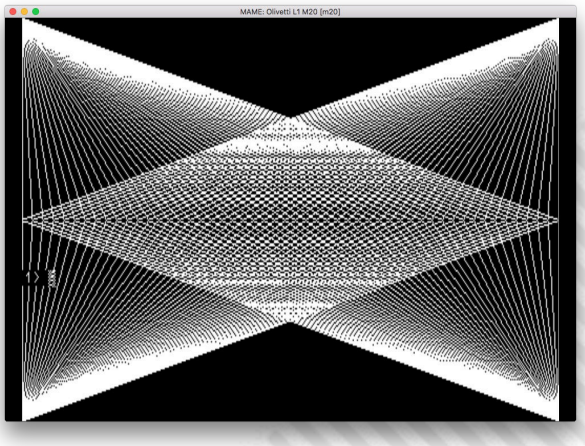


Figure 5: The result produced by listing 1