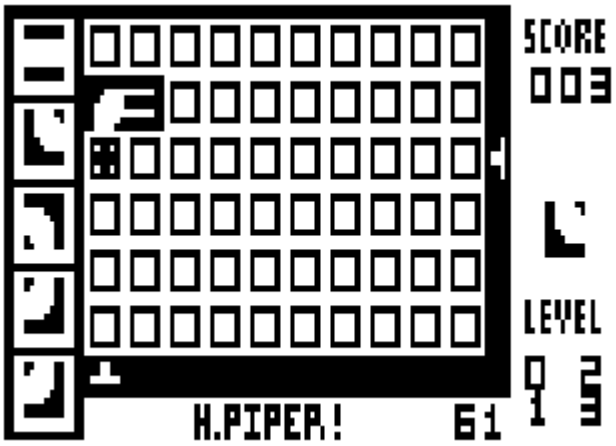
Emulators on the HP Prime Part 1 : Chip-8

Mark Power, mark.power@btinternet.com

In my article on HP Prime programming in V34N1P4 I presented a HP-PPL game that offered the sort of performance that needed machine code on the HP48. This had me thinking about some of the really good HP48 games and I realised that quite a few of the good ones were written for the Chip-8 emulator. So I wondered if the HP Prime is fast enough to implement a Chip-8 emulator written in HP-PPL.

If you are not familiar with Chip-8, it is an interpreted programming language that was designed to simplify games programming on early 8-bit RCA 1802 microprocessor systems in the mid-1970s. Its instruction set is very small, yet remarkably powerful which led to a large variety of games being implemented which mirrored the popular arcade games of the time. These included games like Pong, Space Invaders, Breakout and Lunar Lander.

When the HP48 came along there was a revival of these games, along with extensions to the Chip-8 implementation to provide additional capabilities and utilise more modern hardware with higher resolution displays. The enhanced HP48 emulator called SuperChip (SCHIP) is still available for download from hpcalc.org and still resides on my HP48GX. My favourite game of the era was H.Piper! that was a port of the popular Pipe Mania or Pipe Dream. This is what it looks like running in my emulator on the HP Prime:



Chip-8 Resources

There are great resources on the Internet describing the Chip-8. To start with Wikipedia provides a good background at https://en.wikipedia.org/wiki/CHIP-8

Then there is the archive of all of the games written for Chip-8 and its variations at http://www.chip8.com

Prime Implementation

I'm not going to reproduce the Chip-8 or SuperChip instruction set here as you can easily find them online. A great description of how to implement an emulator is given at:

http://www.multigesture.net/articles/how-to-write-an-emulator-chip-8-interpreter/

I'm also not going to reproduce the whole of the code for the implementation as it will fill an entire Datafile and take you ages to type in. Instead I want to focus on some capabilities of the Prime that I utilised to build the emulator and which might be useful in other projects.

Function Table

When researching implementing the Chip-8 emulator I came across a complete implementation in C, which being a language I'm happy to use, I thought would help explain some of the vagueness of the implementation of the instruction set. If you are into C have a look at https://github.com/Vik2015/chip8/blob/dev/chip8.c

One of the key things about this implementation is that it uses massive switch/case blocks for each instruction that is emulated. They look like this:

```
switch (inst) {
        …
        case 0x01:
                /* JP addr
                * Jump to location nnn.
                */
                PC = addr;
                break;
```

This is great for simplicity, readability and portability in C implementations. What concerned me was whether the Prime Programming Language would handle the equivalent statements efficiently. So I thought I would try and use a function pointer table instead. These should be familiar to C/C++ programmers. After a bit of experimentation it transpires that the Prime can indeed handle such constructs as shown in the code fragments below:

```
CpuOp1()
BEGIN
        // JP addr
        PC:=addr;
END;
…
```

```
// The function table from the Initialise() function
funcs:={'CpuOp0()','CpuOp1()','CpuOp2()','CpuOp3()',
        'CpuOp4()','CpuOp5()','CpuOp6()','CpuOp7()',
        'CpuOp8()','CpuOp9()','CpuOpA()','CpuOpB()',
        'CpuOpC()','CpuOpD()','CpuOpE()','CpuOpF()'};
…
// The main loop
EXPORT OpenChip8(file)
BEGIN
   Initialise();
   keepRunning:=Load(file);
   WHILE keepRunning DO
         // Fetch
         opcode:=BITOR(BITSL(SETBITS(GetMem(PC),16),8),GetMem(PC+1));
         PC:=PC+2;

         // Decode
         inst:=BITSR(BITAND(opcode,#F000:16h),12);
         x   :=BITSR(BITAND(opcode,#0F00:16h),8);
         y   :=BITSR(BITAND(opcode,#00F0:16h),4);
         addr:=BITAND(opcode,#0FFF:16h);
         kk  :=BITAND(opcode,#00FF:16h);
         n   :=BITAND(opcode,#000F:16h);

         // Execute – this is the statement that uses the function table above
         EVAL(funcs[inst+1]);

         HandleEvents();
         UpdateTimers();
         // Delay();
   END;
END;
```

Hopefully this code fragment makes sense. I think it makes the implementation a bit more modular and although I've not proved it to be faster than case statements, it is fast enough.

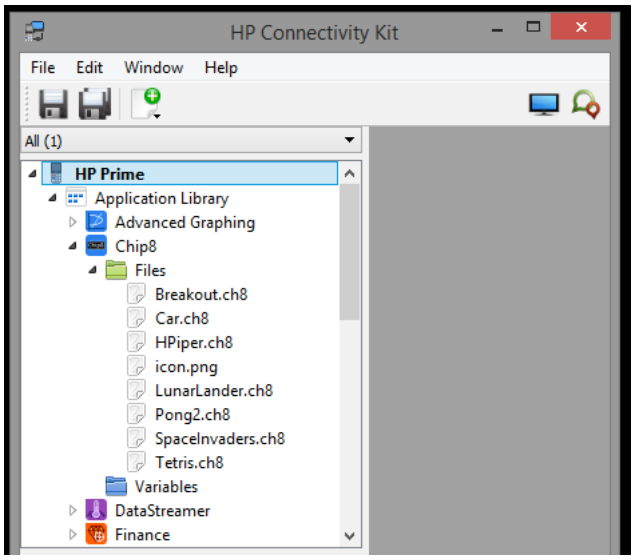
App Framework and Files

My next problem to solve during implementation was that I wanted to be able to load native Chip-8 ROM files into the Prime without having to pre-process them beforehand to turn them input a format that the Prime can handle. What I didn't want was what I subsequently observed in another HP Prime implementation of Chip-8 that contains a string containing a hex dump of the ROM as this would mean that every time you wanted to add or delete a ROM you would need to modify the Chip-8 program itself. Fortunately a recent firmware update to the Prime solved my problem with the addition of AFiles() and AFilesB().

If you have a look on the Museum of HP Calculators forum you can find a HP Prime PNG Image Viewer. This viewer uses both the App framework, which is used by the native Prime Apps, and binary files (the PNG images), which are kept separate from the program that allows you to select a file and display it.

The App framework allows you to access the Chip-8 emulator from the Apps button rather than having to select it from the Toolbox button and then the User program area, which can quite easily become a very long list that you have to scroll through. Apps can be sorted chronologically so you can have Chip-8 at the top of the screen if you wish.

As outlined in the Prime manual the App framework allows you to assign operations to some of the black keys at the top of the keyboard. For the Chip-8 emulator, and for consistency with the PNG Viewer, I have used the View button for the main functionality. From there you can look at the help, open a ROM, delete ROMs that you no longer want to play (or to free memory) and look at the About screen.



When Apps are transferred to a PC they have the advantage that Notes go with them. The App icon and any other files are kept in a file structure that allows them to be manipulated easily. So to add ROMs to your calculator you download them from http://www.chip8.com, rename them to give them shorter filenames ending in ".ch8" and drop them into the Files area of the Chip-8 App in the Connectivity Kit as shown on the left.

If you want to delete a ROM, you can either delete it from the calculator as previously described or you can delete it in the App structure in the Connectivity Kit and re-sync the App with the calculator.

Chip-8 and SuperChip

Whilst looking at other Chip-8 implementations it became apparent to me that some of the instructions are quite vaguely described and this is particularly true when looking at the SuperChip extensions. So whilst I've tested my implementation with a number of ROMs, I can't guarantee that they will all work.

If you find a ROM that behaves oddly then let me know and I'll fix the emulator. If you fancy doing so yourself you can press the D key during gameplay and enter the much-improved Prime debugger yourself.

Speed

Part of the reason for building this emulator was to see how fast the Prime is. Unfortunately as PPL seems to be interpreted, what I've ended up with is an interpreter for one architecture interpreting code for a different architecture so the speed is down on what I'd like it to be. Loading Pong and trying it out was good to prove that using ISKEYDOWN() does indeed allow two players to be pressing buttons at the same time, but the actual game play is a bit disappointing.

Strangely what appear to be more complex games like Breakout, Lunar Lander and H.Piper! seem to run much better, although responsiveness to the keyboard is not perfect.

I did a bit of analysis and where the Prime seems slow is in drawing the sprites, so suggestions on how to improve that routine would be great. Lack of different types also contributes to a somewhat less than ideal implementation. Indeed I've resorted to using lists for the emulated memory and registers. Also particularly annoying for me as a C/C++ programmer is that indexes for lists (and arrays) start at 1 rather than 0, which means my emulator code is littered with variable+1 or variable-1. This of course also impacts the speed of the emulator.

Implementations of Chip-8 for other platforms require a Delay() function to slow the emulation down. I haven't found it necessary to implement this in the Prime code just yet. However you may find that if you play games through the HP Prime Virtual Calculator on your PC the speed is incorrect because I've omitted this.

Sound

A final shortcoming of the Prime version of the emulator is that that hardware doesn't support sounds, which is a shame. Maybe this is something that we can look forward to with a Prime 2 or maybe a Tony Duell modification to the hardware.

And finally

If you want to look at the whole of the code and, better still, play some of the games you can download the entire package from the HPCC web site by clicking on the HP Prime on the front screen (or picking it from the Calculator page) and scrolling down to the programs section. Alternatively download it directly from:

http://hpcc.org/programs/hpprime/Chip8/Chip8.hpappdir.zip