Alternative assembler for *PicoBlaze* (KCPSM3)

Jan Viktorin

July 29, 2014

Abstract

KCPSM3 is a processor core for use in FPGA from *Xilinx*. *Xilinx* offers it to download and synthetize for free from their website. They ship it with an assembler (KCPSM3.EXE). This program can be run on *MS Windows* only, because no source code is provided. Those days there are available (as I know) two other implementations – in C++ and in Java. But nothing in pure C (that's a pity, isn't it?).

I decided that it could be nice to have an open source assembler for KCPSM3 written in C. This document describes my implementation of this idea.

Project goals I was trying to make a small program available under a GPL license, that does translation of KCPSM3 source code to a HEX[1] file. It can also produce some debugging info in a listing file.

The program is written in C99, that means that it is possible to compile it for nearly any platform with only a few changes. I'm going to use the program on PC and on an embedded device – here it can pass the assembled instructions directly to the KCPSM3 processor in a connected FPGA. The program aims to use as few memory as possible (depends on the source program). It assembles the whole program in one pass. If you define all constants at the beginning it doesn't need to store much temporary information during the runtime (only future jumps needs that).

Implementation for PC, porting issues The source code I provide is intended for use on PC. Feel free to modify it for your current needs (according to GNU GPL license). There are some issues that must be considered then.

1. The program uses system call mmap for reading the source code. On an embedded device this is useless, so there must be written a buffering strategy for that. If you need this change, see module buffer.c.

2. If you want to assemble only eg. a line of code (inline assembling?), the current implementation can not do that (directly), it always outputs all 1024 instructions (because of calling output_flush). It is not difficult to change this behaviour.

3. If you have a device with the FPGA directly connected, it is possible to send the assembled program directly there. To do this, change the module output.

[1] Careful, it is **NOT** the Intel HEX file, the output is same as the original KCPSM3.EXE does.

4. It is also possible to run more instances of the assembler in different threads without collisions (each threat uses its own data storage, there are no global variables), each assembler run is thread safe.

5. I recommend to create a special header file for every different platform, that needs some wild changes (as it is done for PC – `pc.h`).

Differences between the original and my program My implementation is **NOT** fully compatible with original program shipped with the processor core. The most important differences are mentioned here:

1. Registers in the basic form `"sX"` are case sensitive (thus `"sA"` is recognized, but `"sa"` or `"Sa"` or `"SA"` is not recognized). This is realy important difference, because the original program treats everything case insensitive but labels case sensitive.

2. The original program writes many files, eg. `VHDL` and `Verilog` source codes with the assembled program. My program writes only the file with the result and a listing file (if requested). I follow the Linux way of programming, that one program should be small and do only the basic job. It is not difficult to write a script that converts output of my program to a `VHDL` code that looks the same as the *Xilinx*'s `KCPSM3.EXE` does (eg. `https://github.com/fcelda/hex2vhd`).

3. My program supports some shotcuts[2] similar (and compatible) to those from `pBlazIDE`. Namely: `ADDC`, `SUBC`, `COMP`, `IN`, `OUT`, `EINT`, `DINT`, `RET`, `RETI`. For instructions with indirect access (`IN`, `OUT`, `FETCH`, `STORE`) it is not necessary to write paranthesis around the second register operand.

Compilation To compile the assembler, simply use *GNU make* tool:

```
$ make
```

If you want to activate the `pBlazIDE`-like extension, type this instead:

```
$ CFLAGS=-DSHORTCUTS_EXTENSION make
```

Usage of the assembler After you successfully compile the staff, you can start to use the program like this:

```
$ ./pico -i program.psm -o program.hex -l program.l
```

You can skip any of the options. When you skip `-i` parameter (the input file), program tries to read from the `stdin` (but this can fail because of using `mmap`). When you skip the `-o` option (output file), the assembled result is written to `stdout`. Skipping of `-l` option will disallow creation of the listing file.

It is possible to use another two options: `-q` (quite mode – no error messages) and `-h` (prints help).

[2]Must be compiled with preprocessor flag `-DSHORTCUTS_EXTENSION`.