FP8-17: Software Programmable Signal Processing Platform Analysis Exercises for Episode 3

Andrzej Wąsowski

Wednesday, 29 March 2006

Exercise 3.1 Write a short C program exhibiting a type error. Run it using your compiler and observe the error message.

Exercise 3.2 Write a C program containing an undeclared identifier. What mechanism in the compiler detects this error?

Exercise 3.3 List the entries in symbol table (mapping from names to types) at all the points of the following program:

```
int f(int x) {
    int y = x + x;
    if (y < 10) return f(y);
    else return y-1;
}
```

Exercise 3.4 Find the lecture slide showing assigning of registers to parameters. Discuss the examples with your group, comparing them to the general rules of the previous slide. Also check this on some of your actual code (perhaps as a part of the next exercise).

Exercise 3.5 (big) Compile some C code of your project, producing the assembler files (option `-k`). Indicate the various elements of C67xx calling convention in the generated assembly code. It is useful to use the `-s` (or `-ss`) option, to interline the C source with the assembly. The `-O0` may cause the generated code to be more readable (lowest optimization level).

Exercise 3.6 (a micro project) In continuation of the previous exercise consider examining a function that is frequently called. Can performance be gained by avoiding some of the call overhead? Consider inlining a function using the `inline` keyword, inline manually (perhaps using a preprocessor macro), or inline and implement in assembly. Each time examine the output assembly code and benchmark efficiency if possible. If you have any interesting observations then do not forget to document them in the project report.

Exercise 3.7 (project hint) Read SPRU 187, p. 7–11 on near and far pointers, and the role of the DP (data pointer) register in accessing global and static data. Compare the access times to global data for near pointers (a single `LDW` instruction) and the time required for far pointers (two moves and one `LDW`).

Analyze the assembly code for efficiency critical part of your project. Check which memory model is used by the compiler, and check whether your data fits in the 32K bytes requirement. If not try to move to the smaller memory model, to enforce more efficient data access. Another idea would be to give up global data in favor of local data allocated in stack frames.

Exercise 3.8 For brave: explain to Andrzej what happens on the snippet of assembly code for the *Hello world* program presented in episode 2.