RISC-V Spike Simulator Usage Guide

- Homework 3: Codegen
- NTHU Compiler Design (CS340400)

Outline

1. Generate Executable
    i. codegen.S
    ii. Compile Executable
2. RISC-V Spike Simulator
    i. Spike Introduction
    ii. Spike Usage

1. Generate Executable

1-1. codegen.S

- The rules are totally the same as those codegening for Andes Corvette-F1-N25, including but not limited to the following items:
  - Same set of Testcases
  - Implement `delay`, `digitalWrite`
  - `.global codegen`
  - ...

1-2. Compile Executable

- TAs provide a tweaked version of the `assembly` sample project, which includes:
  - `main.c` : The main program
  - `codegen.S` : The same one as in the `assembly` project

1-2. Compile Executable (cont.)

- To compile your `codegen.S` into an executable, use `riscv64-unknown-elf-gcc`
  - E.g. `riscv64-unknown-elf-gcc -o sample_prog main.c codegen.S` in the `assembly` folder
    - The above command does the following:
      a. Compile `main.c`
      b. Assemble `codegen.S`
      c. Link them together to produce `sample_prog`
    - `sample_prog` is the executable we want

2. RISC-V Spike Simulator

https://github.com/riscv/riscv-isa-sim

2-1. Spike Introduction

- Spike is a function-level simulator for the RISC-V ISA
- It operates in a bare-metal manner, i.e. it behaves like a hardware without OS

> In HW 3 Spike, we need OS support for `printf` in `assembly/main.c`, so we make use of the `pk` utility provided by the RISC-V community (`pk` stands for "proxy kernel").

2-2. Spike Usage

- Suppose we have our compiled `sample_prog`, to execute it, run:
  - `spike pk sample_prog` in the `assembly` folder
  - You should have a correct invocation log of `delay` and `digitalWrite` as output
    - This is the correct output for the `assembly` sample project

```
bbl loader
Arduino digitalWrite(27, 1);
Arduino delay(1000);
Arduino digitalWrite(27, 0);
Arduino delay(1000);
```

2-2. Spike Usage (cont.)

- The Spike flag `-d` puts it in interactive debug mode:
  - E.g. `spike -d pk sample_prog` in the `assembly` folder
  - `: help` shows the usage manual

  > We only enable single core, so for `<core>` mentioned in the manual, it's usually `0`

> `: until pc 0 0x0000000000010210` would stop the debugger at the entry of `codegen.S`, where `0x0000000000010210` is the location of `codegen:` in your `codegen.S`

Thanks