Assembly Programming on Spike

Prerequisite

Go through this guide Installation Guide for Spike.docx to set up the environment (skip if already done).

Walkthrough

This reference manual illustrates how to run a bare-metal program on Spike.

1. Create an assembly file (**test.S**) on the following template:

```
.text

// start code here

// Insert your code here

// end code here

write_tohost:

li x1, 1

sw x1, tohost, t5

j write_tohost

.data

// start data section here

// Initialize data here (if required)

// end data section here

.align 12

.section ".tohost","aw",@progbits;

.align 4; .global tohost; tohost: .dword 0;

.align 4; .global fromhost; fromhost: .dword 0;
```

2. Create a simple linker file (**link.ld**) , providing the starting address(es) of code and data section.

3. Run the following command to compile your assembly program and get the executable (**test.elf**) file.

```
riscv64-unknown-elf-gcc -march=rv32g -mabi=ilp32 -nostdlib
-nostartfiles -T link.ld test.S -o test.elf
```

You can check the dis-assembly from the executable using:

```
riscv64-unknown-elf-objdump -D test.elf
```

4. Use the following command to run the executeable on Spike and get the trace file (**test.log**). Use the **test.log** to debug your assembly program.

```
spike -l --log-commits --isa=rv32gc test.elf 2>test.log
```

5. Or you can use the interactive debug mode and step through every instruction by using the following command:

```
spike -d --log-commits --isa=rv32gc $(which pk) test.elf
```

Interactive Debug Mode

To invoke interactive debug mode, launch spike with -d (as shown in step 5 above).

To see the contents of an integer register (0 is for core 0):

```
: reg 0 a0
```

To see the contents of a floating point register:

```
: fregs 0 ft0
```

or

```
: fregd 0 ft0
```

depending upon whether you wish to print the register as single- or double-precision.

To see the contents of a memory location (physical address in hex):

```
: mem 2020
```

To see the contents of memory with a virtual address (0 for core 0):

```
: mem 0 2020
```

You can advance by one instruction by pressing the enter key. You can also execute until a desired equality is reached:

```
: until pc 0 2020                              (stop when pc=2020)

: until reg 0 mie a                            (stop when register mie=0xa)
```

You can continue execution indefinitely by:

```
: r
```

To end the simulation from the debug prompt, press `<control>-<c>` or:

```
: q
```