BRISC-V Tool Box

Donato Kava, Sahan Bandara, Alan Ehret, and Mihailo Isakov

ASCS Lab, Boston University

briscv.dev@gmail.com

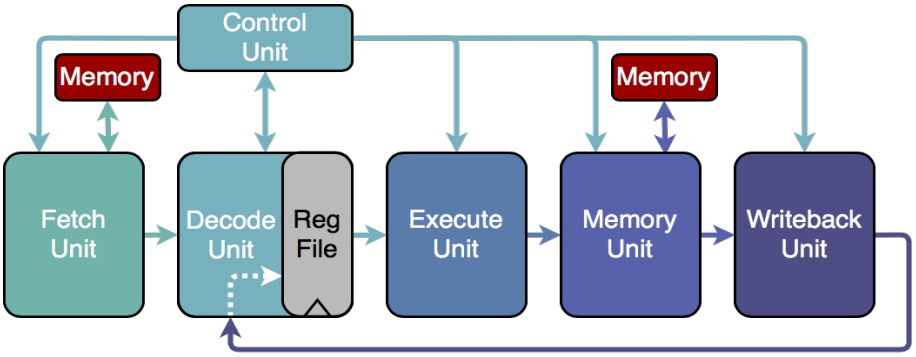
The Boston University RISC-V Processor Set (BRISC-V) is a parameterized set of modules for design space exploration for RISC-V ISA based architectures. We call the full set of processors and tools to support them the tool box. Included with the BRISC-V Tool box include (i) multiple RISC-V cores with different levels of complexity (e.g., single-cycle core, multiple-cycle, and reconfigurable pipelined), (ii) a programmable memory system with reconfigurable multi-level cache subsystems, (iii) a flexible interconnect network supporting programmable topology, router size and routing algorithm and (iv) BRISC-V explorer which is GUI software support for selecting parameterized Verilog and (v) the BRISC-V emulator for software RISC-V instruction emulation. The aim of this work is to provide an easy to use, open-source, parameterized, fully synthesizable, platform for students and researchers experimenting with the RISC-V ISA features to quickly bring up a complete and fully working architecture and start applying their own modification.

1 Getting Started

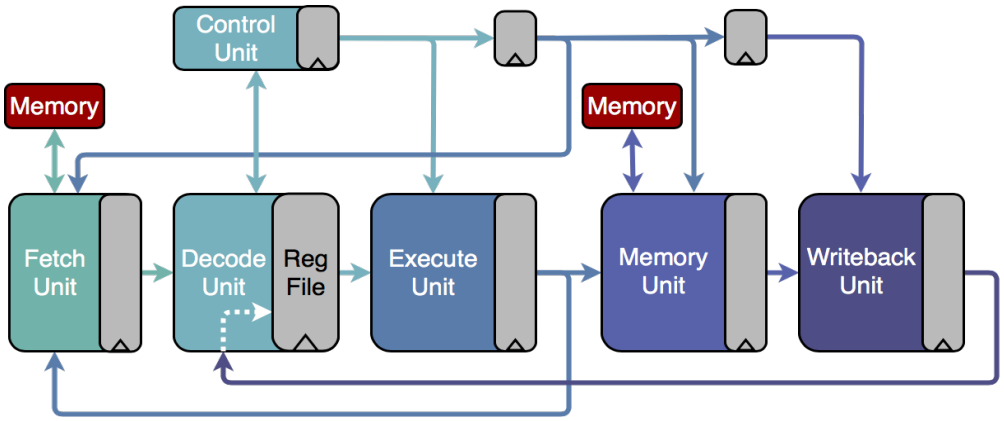
The BRISC-V tool box offers users a variety of different single-core and multi-core systems. Each hardware component of BRISC-V (the cores, cache subsystem and on-chip network) is written in parameterized Verilog HDL modules, enabling architectural changes with parameter settings. The core types currently supported by BRISC-V include RV32I implementations of a small single cycle core, five and seven stage pipeline cores and a larger RV32IF out-of-order core. The cache subsystem and on-chip network interfaces support numerous memory hierarchy configurations. An arbitrary cache size, associativity and number of levels can be selected with module parameters.

1.1 The Processors

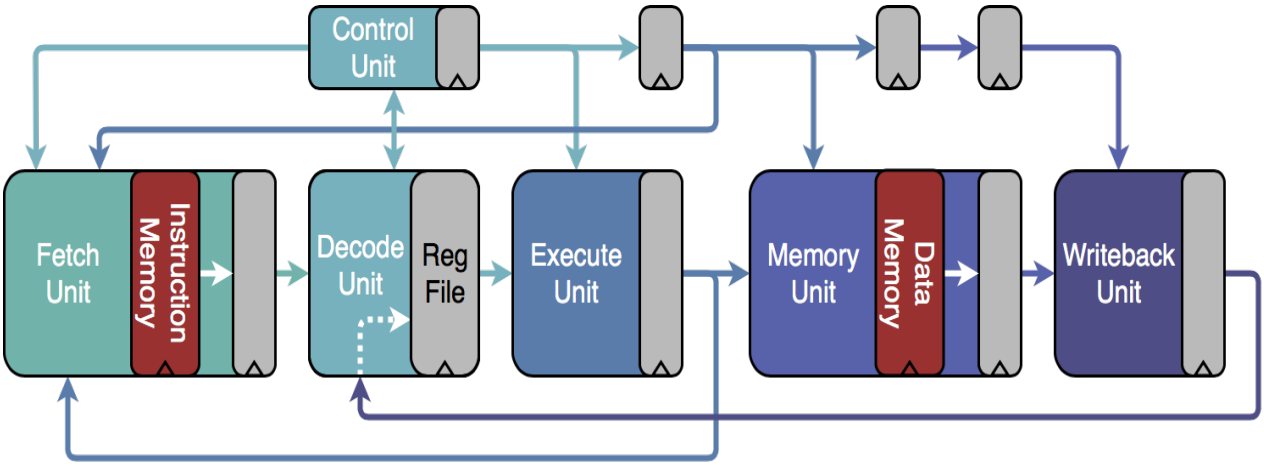
1.1.1 The Single Cycle RV32i Processor



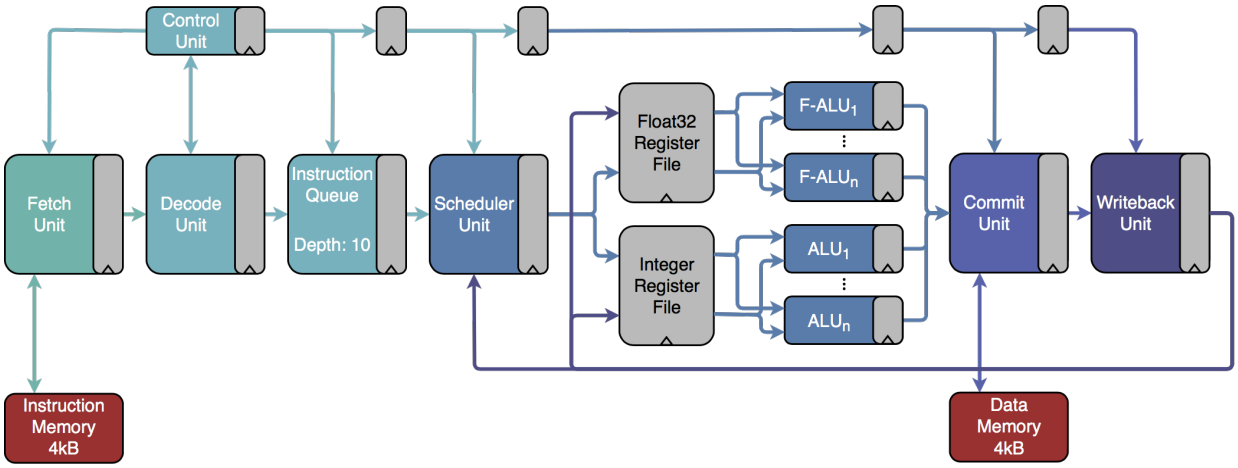
1.1.2 The Five Cycle RV32i Processor



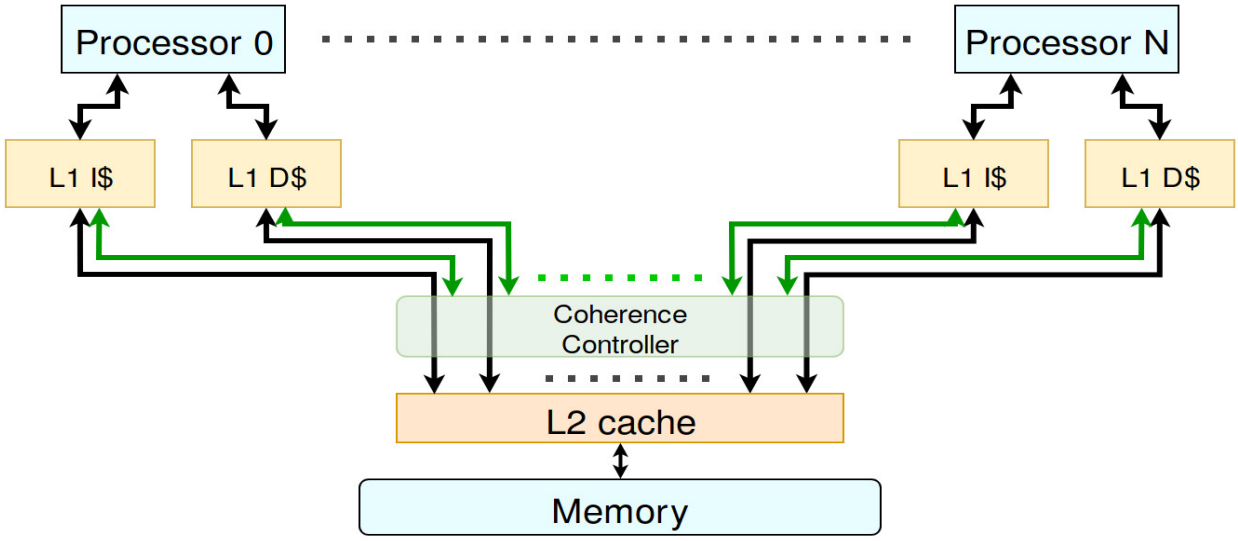
1.1.3 The Seven Cycle RV32i Processor



1.1.4 The Out of Order RV32i Processor



1.2 Cache System



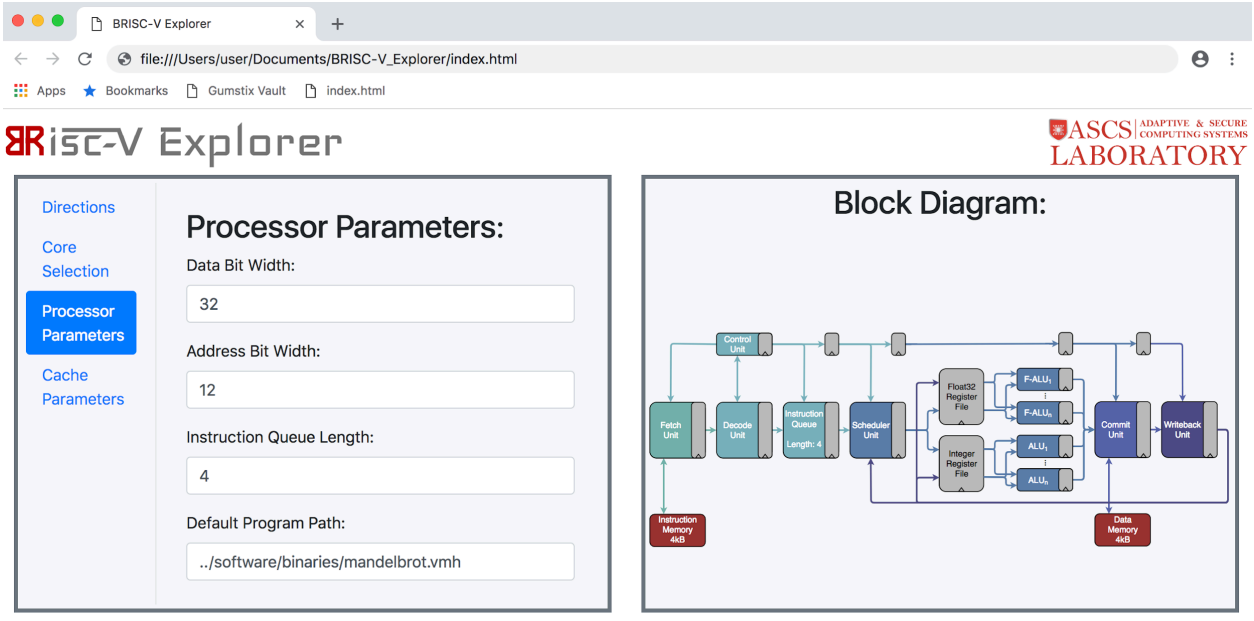
1.3 Software Tools

The BRISC-V toolbox is not limited to only Verilog designs, it offers two interactive tools for working with RISC-V. The BRISC-V explorer adds a visual aid for visualizing hardware parameterization. The BRISC-V emulator is offered as a way to emulate RISC-V programs without the need for hardware and shows the state of the processor as each instruction is ran.

1.3.1 BRISC-V Explorer

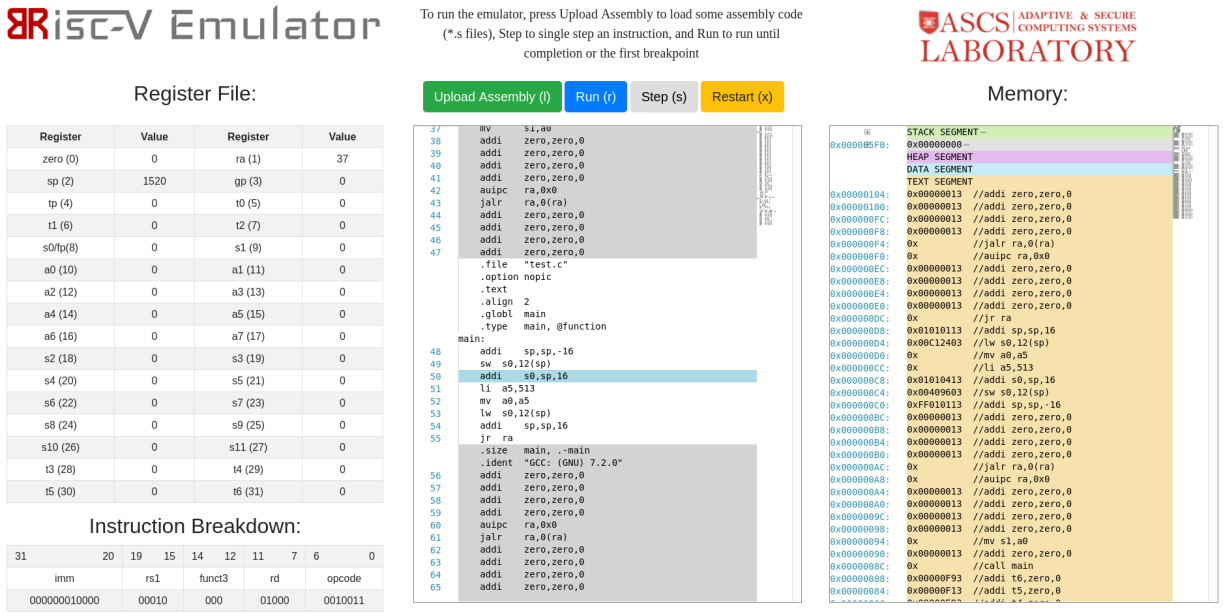
The BRISC-V Explorer provides a user friendly way to choose parameters and visualize a BRISC-V system. The application runs in a browser allowing users to easily run it on Windows, Linux or Mac. In the BRISC-V Explorer, users can 1) select their desired core type, 2) enter parameters such as memory size for that core, and 3) configure cache parameters including block size and associativity.

Once a user is sure of their processor and generates their design a verilog the explorer tool will generate a verilog design based around the users selections. From there its up to the users imagination on what to do with it.



1.3.2 BRISC-V Emulator

The BRISC-V Emulator visually shows the state of the processor at every instruction and allows for exploration of a compiled code behaving as expected. Being an in browser tool OS dependencies are avoided allowing for an easy, fast, and intuitive exploration. The register file, instruction break down, memory state and program list are all displayed as the program operates.



2 Manual Parameter Selection

Outside of the explorer tool where parameters are selected when generating, parameter selection in the processor verilog files is just as easy. Parameters are at the top of the RISC_V_Core.V file. As seen below, in an example from the 7 cycle processor, the parameters are laid out and named as clearly as possible. Its is important to properly fill in the PROGRAM parameter to match the path on your computer where a generated verilog hex file is located.

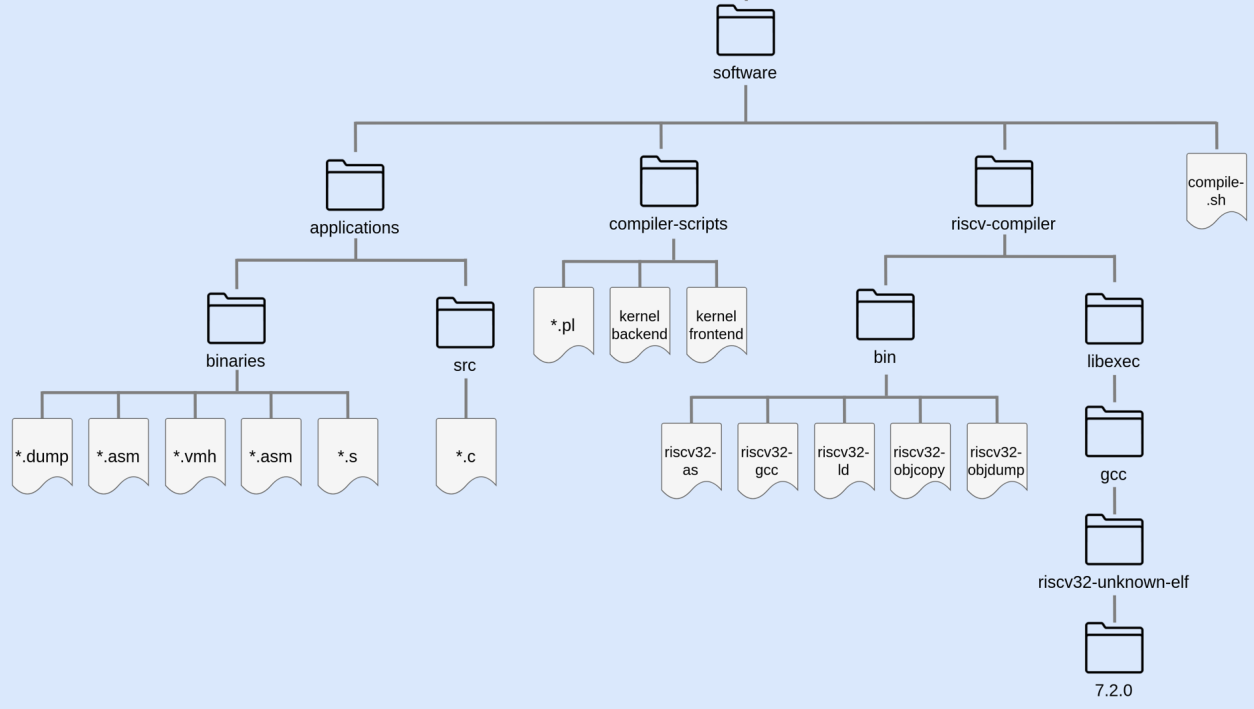
When making personal changes to the processors we recommend to use the parameters to keep the parametric property of the processor set and for ease of experiments later on.

```
parameter CORE = 0,
parameter DATA_WIDTH = 32,
parameter INDEX_BITS = 6,
parameter OFFSET_BITS = 3,
parameter ADDRESS_BITS = 12,
parameter PRINT_CYCLES_MIN = 0,
parameter PRINT_CYCLES_MAX = 15,
parameter PROGRAM =
"../software/applications/binaries/short_mandelbrot.vmh"
```

3 Compilation Software

3.1 Compile Tool

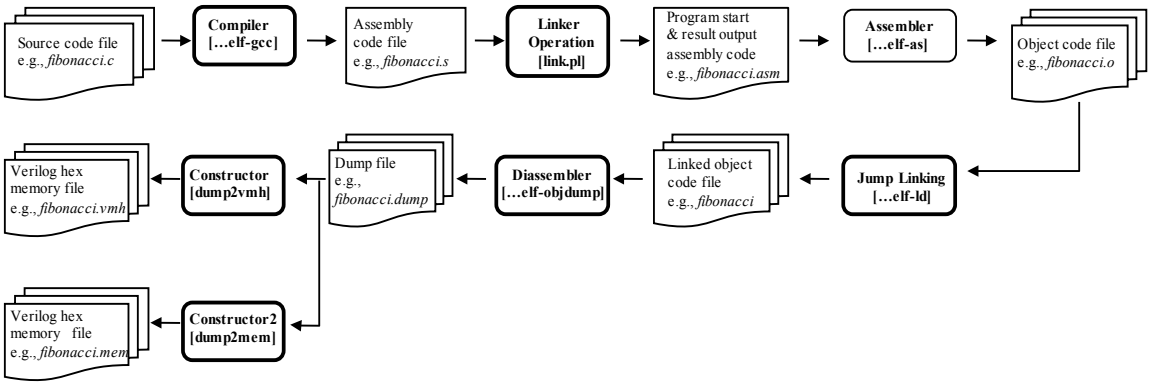
In the software directory there are three folders called ***applications***, ***compiler-scripts*** and ***riscv-compiler***. There is also a file named compile.sh which will generate all the binary files in the binaries directory, from the C programs in ***applications/src/***, using the ***riscv-compiler***. Editing ***compile.sh*** can allow for not provided program binaries to be generated. In the applications folder there are two sub folders, ***binaries*** and ***src***. In the ***src*** folder there are 12 sample C programs and in the binaries folder there are sample program's .asm, .dump, .mem, .s, and .vmh versions. In compiler-scripts there are perl scripts used to arrange the BRISC-V© program kernel. The folder ***riscv-compiler*** contains two folders named ***bin*** and ***libexec***. Libexec contains a number of sub-folders which are empty while bin contains the RISC-V gcc tools used for generating binary files to compile and generate binaries for the RISC-V rv32 ISA.



3.1 Compile Process

To assist in developing software for the different BRISC-V processor, it is accompanied with a GCC RISC-V cross-compiler. The figure above depicts the software flow for compiling a C program into the compatible BRISC-V instruction code that can be executed on the processor. The compilation process consists of a series of seven steps.

1. First, the user invokes ***riscv32-unknown-elf-gcc*** to translate the C code into assembly language (*e.g., ./riscv32-unknown-elf-gcc -S fibonacci.c*).
2. In step 2, the assembly code is then run through the linker to set up the stack pointer and return value registers (e.g., *./link.pl fibonacci.s*). Its output is a .asm file.
3. In step 3, the user compiles the assembly file into an object file using the cross-compiler. This is accomplished by executing ***riscv32-unknown-elf-as*** on the .asm file (e.g., *./riscv32-unknown-elf-as fibonacci.asm –o fibonacci.o*).
4. In this step, all the jump addresses are properly linked with *./riscv32-unknown-elf-ld -N -Ttext 0x0004 --unresolved-symbols=ignore-all fibonacci.o –ofibonacci.*
5. In step 5, the object file is disassembled using the ***riscv32-unknown-elf-objdump*** command (e.g., *./ riscv32-unknown-elf-objdump fibonacci.o*). Its output is a .dump file.
6. In step 6, the constructor script is called to transform the dump file into a Verilog memory .vmh file format (e.g., *./riscv32-unknown-elf-objcopy fibonacci.dump*).
7. Finally, a second constructor script is called to transform the dump file into another Verilog memory .mem file format (e.g., ./dump2vmh fibonacci.dump). Different Verilog simulations or FPGA synthesis tools use different formats, i.e., .vmh or .mem. They contain the same data. Programs/Applications that have some initial values/data stored in memory will also have a data file generated for them (e.g., data_fibonacci.vmh/mem).



For script-based compilation, if you run ./***compile.sh***, it will take a set of predefined C applications/programs in the ***application/src*** folder and compile all of them. If you would like to compile your own application (e.g., *albert_s_ beautiful_code.c*) with your own stack pointer size (*albert_s_ stack*, a decimal number), you can execute ***./compile.sh albert_s_ beautiful_code.c albert_s_ stack***. (e.g., *./compile.sh foo.c 128*).


BR
isc-V
Adaptive and Secure Computing
Systems Lab • Boston University