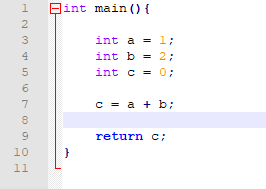
RISC-V User Guide

## Preface

This document outlines how to use the RISC-V compilers and the RISC-V processor via MSYS (compilers) and Xilinx’s Vivado tool suite (core). If the RISC-V compilers have not already been installed, please refer to the “RISC-V Compiler Installation Guide” document found within the “Documentation” folder. It is highly recommended that users install Notepad++ ( <https://notepad-plus-plus.org/download/v7.6.3.html> ) for viewing compiler outputs files. It is also recommended that users review the RISC-V Specification ( <https://riscv.org/specifications/> ) and consider associate books ( <https://riscv.org/risc-v-books/> ).

## Compilers

To utilize the RISC-V compilers please open an instance of “MSYS2 MinGW 32-bit”. This will open a bash command line interface to the “fake” linux environment. From here it is recommended to create a programs directory via the “mkdir” command. Within this directory create a folder for the program you would like to compile. In the example presented the program to be compiled is “addVar” as follows:



Open this folder in your file explorer by navigating to “C:\msys64\home\user\programs\add\_var”. Copy your C program into the corresponding folder i.e. “\programs\<user\_program>”. Then copy the scripts found in the “Scripts” folder into this directory.  
  
Return to the “MSYS2 MinGW 32-bit” terminal and type “./compile.sh <program\_name.c> <output\_name>”. This will run the GCC compiler which converts your C code into RISC-V assembly code. This script will then call the “assemble.sh” script which converts the RISC-V assembly into the outputs we require for the RISC-V core. If you have written RISC-V assembly then type “./assemble.sh <program.s> <output\_name>”. The outputs from these scripts are as follows:

* **.S** -> This is the output of the GCC compiler, when opened in a program like Notepad++ the RISC-V assembly equivalent of your C code can be seen, these are the files user’s will write RISC-V assembly with and will call the “./assembly.sh” script on.
* **.ELF** -> (Executable and Linkable Format File) is a common standard file format for executables and object code. In this context is the output of the assembler and will not be used directly by the user.
* **.OUT** -> This is the output of the linker which resolves jump and branch addresses and register assignments.
* .**ELF\_DUMP** -> This is the human readable version of the “.OUT” file, it will display the memory addresses of the instructions, the hexadecimal equivalent of the instruction and its assembly equivalent. This will be the most useful file for users.
* .**HEX\_LITTLE/BIG** -> These are the hexadecimal (little/big endian) outputs of the compilers, these files are usually only viewable on Linux, so most users need not worry about them.
* **.TB** -> This is the file users will use to directly load programs into the Vivado VHDL testbench to run on the core.
* **.MACRO** -> This is the file users will use to load the code into the FPGA from ViciLab

## RISC-V Processor

Navigating to the “RISC-V\_Top/vhdl/xilinxprj” and opening the “RISC-V” Xilinx project will yield a structure as follows:



The RISC-V processor is represented here by “RISC-V\_Top”. The processor has been wrapped in ViciLogic compatible top modules i.e. “singleCycCompAndMem” and “singleCycCompTop”. This is to allow the RISC-V processor to be implemented on ViciLogic.

Open the generated “.TB” file from the compiler tutorial and open the “singleCycCompTop\_TB” testbench VHDL file within the simulation sources tab in Vivado. Scroll down to approximately line 115 to the “instrArray”. Copy and the contents of the “.TB” file into the uppermost part of the “instrArray”.  
  
On the left click “Run Simulation”. This will open a waveform with pre-configured signals of all the important modules present in the project. From here the user can navigate the operate of the system to their hearts content.