RISCV CPU Documentation Notes

# Installation Instructions

we should set environment variable $RISCV to the –prefix we want (currently /mnt/d/riscv/$tool/)

## RISC-V GNU Compiler Toolchain installation

Using Ubuntu (bash) via Windows Subsystem for Linux

# get the source files from github

git clone --recursive <https://github.com/riscv/riscv-gnu-toolchain>

# install build process dependencies

sudo apt-get install autoconf automake autotools-dev curl python3 libmpc-dev libmpfr-dev libgmp-dev gawk build-essential bison flex texinfo gperf libtool patchutils bc zlib1g-dev libexpat-dev

# configure MAKE and build Newlib cross-compiler (for embedded)

./configure --prefix=/opt/riscv/rv32i --with-arch=rv32i --with-abi=ilp32

make -j4

Note:

see ./configure –help for more build options

change -prefix to change the install directory

from the built directory (/mnt/d /riscv/rv23i/bin) you can now run ./riscv32-unknown-elf-gcc to compile stuff

# Using the Toolchain

**Compiling a source file:**

riscv32-unknown-elf-gcc hello.c -march=rv32i -mabi=ilp32 -o a.out

**Getting the assembly from the compiled products:**

riscv32-unknown-elf-objdump a.out -d

**Building raw hex** **for VHDL memory data for programs** we use elf2hex to convert the ELF file produced by riscv32-unknown-elf-gcc into a hex file <https://github.com/sifive/elf2hex>:

elf2hex --bit-width 32 --input a.out --output a.hex

# ISA Notes

We will be targeting the base integer ISA ‘RV32I’ (RISC V, 32-bit, Integer)

We will not support Variable Length instruction encoding (at this time)

Support for hardware multiply and software divide (ISA extension ‘M’) may be added later

This minimal RV32I Instruction Set has only 40 instructions, although check to see if we really need ECALL/EBREAK instructions (system calls?)

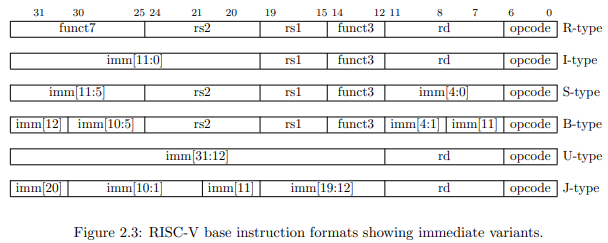
## Instruction Formats

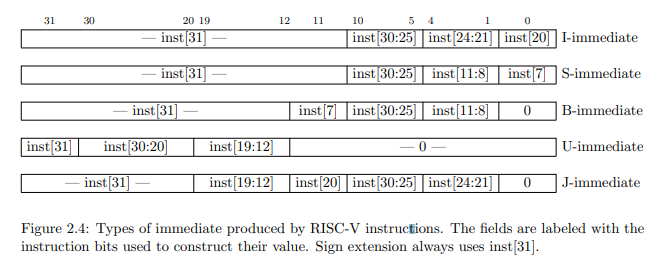
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | | | 5 | | | 5 | | | 3 | | | 5 | | | 7 | | | **Length** |
| 31 |  | 25 | 24 |  | 20 | 19 |  | 15 | 14 |  | 12 | 11 |  | 7 | 6 |  | 0 | **Position** |
| Funct7 | | | Rs2 | | | Rs1 | | | Funct3 | | | Rd | | | Opcode | | | **R Type** |
| Imm[11:0] | | | | | | Rs1 | | | Funct3 | | | Rd | | | Opcode | | | **I type** |
| Imm[11:5] | | | Rs2 | | | Rs1 | | | Funct3 | | | Imm[4:0] | | | Opcode | | | **S type** |
| Imm[31:12] (20 bits) | | | | | | | | | | | | Rd | | | opcode | | | **U type** |

There are 2 other variant formats (B and J) that provide other ways of handling Immediates. See page 16 of the ISA spec for more details.

B and S, and format instructions are very similar, with B and J instructions used for multiples-of-2 immediates (like addresses). The difference is how the immediates are decoded out of the instructions (see Fig 2.4 below)

Sign extensions are always performed using bit31 of an instruction.





## Integer Computation Instructions

Use either I-type (register-immediate) or R-type (register-register) instructions, with rd as the destination register.

### Integer Register-Immediate Instructions

## Control Transfer Instructions

## Load and Store Instructions

## Memory Ordering Instructions

## Environment Call and Breakpoints

# Implementation Notes

Memory Mapping – memory address overflow happens in a circular way

## Registers

There are 32 32-bit registers x0 – x31. x0 is hardwired to 32b0 and the other 31 registers are general purpose.

The program counter pc holds the address of the current instruction and is separate from the compute registers

Convention states that register x1 is the Return Address of a called function, with x5 available as an alternate link register. Convention also states that x2 is the Stack Pointer.

We may wish to add extra hardware to accelerate function calls using x1 and x5 (see JAL and JALR)

# 

# Full RV32I Instruction listing

