RISC-V Software Ecosystem

Dr. Girish H

Professor

Department of ECE

Cambridge Institute of Technology

&

Kavinesh

Research Staff

CCCIR

Cambridge Institute of Technology

RISC-V Software Ecosystem

RISC-V Toolchain

- This is used as RISC-V C and C++ cross-compiler.
- It supports two build modes: a generic ELF/Newlib toolchain and a more sophisticated Linux-ELF/glibc toolchain.
- Github repository link

GitHub - riscv-collab/riscv-gnu-toolchain: GNU toolchain for RISC-V, including GCC

$$\begin{tikzcd}
\boxed{\text{Program}} \arrow[r, "\text{compile}"] & \boxed{\text{Executable}} \arrow[r, "\text{execute}"] & \boxed{\text{Output}}
\end{tikzcd}$$

Spike

- Spike is the golden reference functional RISC-V ISA C++ software simulator. It provides full system emulation or proxied emulation with HTIF/FESVR.
- Multiple ISAs: RV32IMAFDQCV extensions
- Multiple memory models: Weak Memory Ordering (WMO) and Total Store Ordering (TSO)
- Privileged Spec: Machine, Supervisor, User modes (v1.11)
- Single-step debugging with support for viewing memory/register contents
- Highly extensible (add and test new instructions)

Proxy Kernel

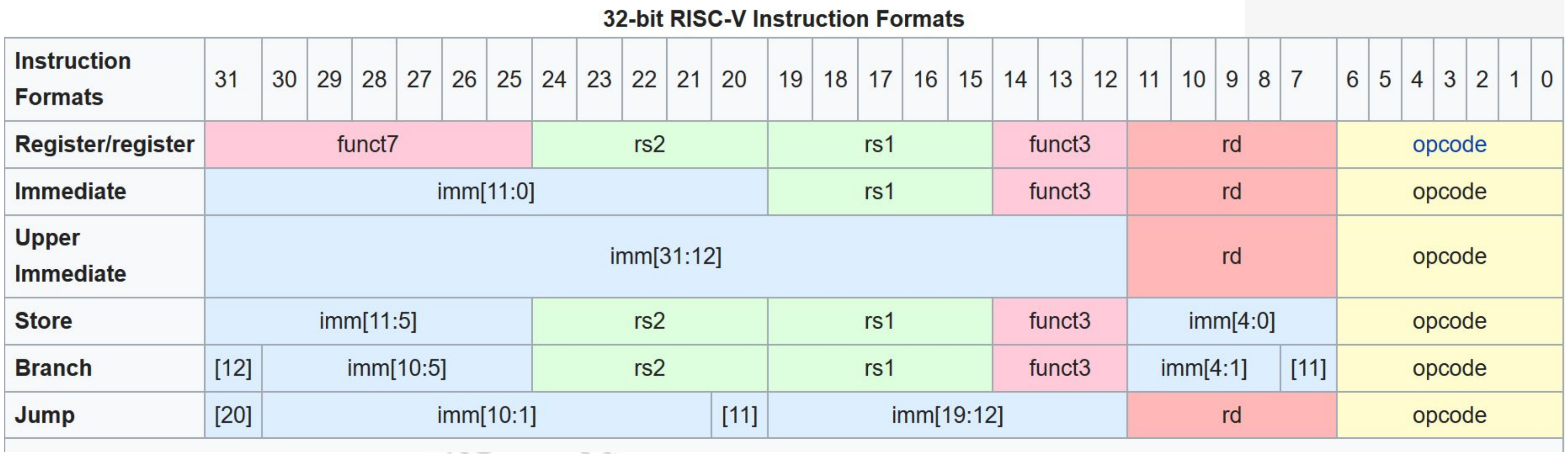
- The RISC-V Proxy Kernel, pk, is a lightweight application execution environment that can host statically-linked RISC-V ELF binaries.
- It is designed to support tethered RISC-V implementations with limited I/O capability and thus handles I/O-related system calls by proxying them to a host computer.
- This package also contains the Berkeley Boot Loader, bbl, which is a supervisor execution environment for tethered RISC-V systems.
- It is designed to host the RISC-V Linux port.

Assembly Instructions

- ❖In assembly language, each statement (called an Instruction), executes exactly one of a short list of simple commands
- ❖Unlike in C (and most other High Level Languages), each line of assembly code contains at most 1 instruction.
- ❖Instructions are related to operations (=, +, -, *, /) in C

RISC-V ISA Instruction format

There are 6 different encoding format for instructions.



32-bit RISC-V Instruction Formats

Instruction Formats	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Register/register	funct7							rs2					rs1					funct3			rd					opcode						
Immediate	imm[11:0]												rs1					funct3			rd					opcode						
Upper Immediate	imm[31:12]																				rd					opcode						
Store	imm[11:5]							rs2					rs1					funct3			imm[4:0]					opcode						
Branch	[12]	imm[10:5]						rs2					rs1					funct3			imm[4:1]				[11]	opcode						
Jump	[20]	imm[10:1]										[11]	imm[19:12]								rd					opcode						

RISC-V Instructions

- One operation per instruction, at most one instruction per line
- Assembly instructions are related to C operations (=, +, -, *, /, &, |, etc.)
  - Must be, since C code decomposes into assembly!
  - A single line of C may break up into several lines of RISC-V
- In Assembly Language, registers have no type, simply stores 0s and 1s; operation determines how register contents are treated

RISC-V Instructions

- Instruction Syntax is rigid:

  op dst, src1, src2

  - 1 operator, 3 operands
    - op = operation name (“operator”)
    - dst = register getting result (“destination”)
    - src1 = first register for operation (“source 1”)
    - src2 = second register for operation (“source 2”)
- Keep hardware simple via regularity

RISCV Instructions Example

- Suppose a $\rightarrow$ s0,b $\rightarrow$ s1,c $\rightarrow$ s2,d $\rightarrow$ s3 and e $\rightarrow$ s4.
- Convert the following C statement to RISCV:

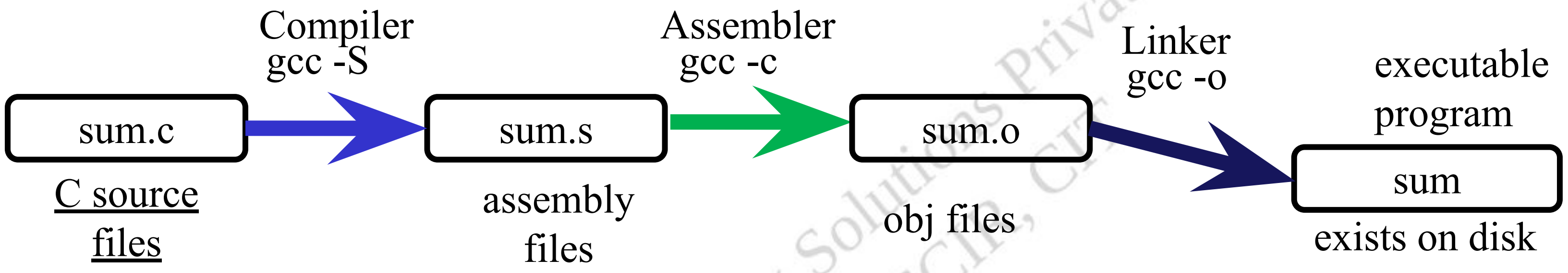
```
a = (b + c) - (d + e);
```

```
add t1, s3, s4
add t2, s1, s2
sub s0, t2, t1
```

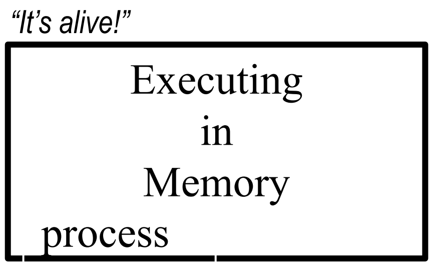
Ordering of instructions matters (must follow order of operations)

Utilize temporary registers

From Writing to Running



When most people say "compile" they mean the entire process:
compile + *assemble* + *link*

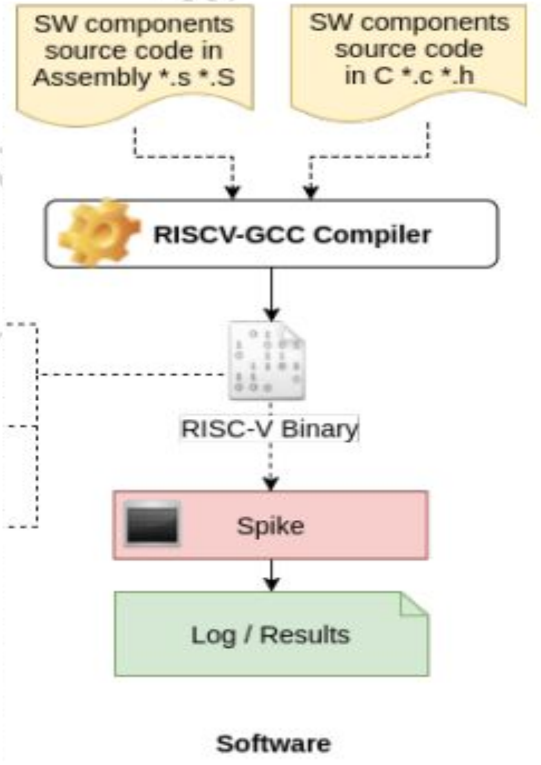


C variables vs. registers

- ❖In C (and most High Level Languages) variables declared first and given a type.
  E.g., int fahr, celsius;
  char a, b, c, d, e;
- ❖Each variable can ONLY represent a value of the type it was declared as (cannot mix and match int and char variables).
- ❖In Assembly Language, the registers have no data type
  - ▪Operation determines how register contents are treated

Software Flow

- A compilation toolchain includes The COMPILER translates high-level source code to a lower-level representation
- The ASSEMBLER is program that translates assembly language to machine language
- The LINKER combines multiple files into a single executable


SW components source code in Assembly *.s *.S
SW components source code in C *.c *.h
RISCV-GCC Compiler
RISC-V Binary
Spike
Log / Results
Software

C:

```
car *c = malloc(sizeof(car));
c->miles = 100;
float mpg = get_mpg(c);
free(c);
```

Assembly language:

```
get_mpg(car*):
        lw      a5,0(a0)
        lw      a4,4(a0)
        divw    a5,a5,a4
        fcvt.s.w        fa0,a5
        ret
```

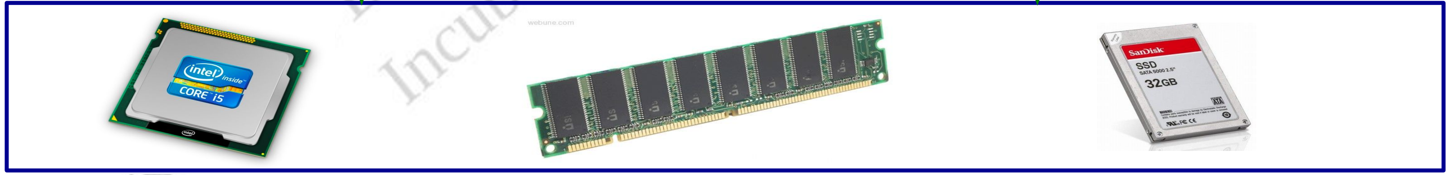
Machine code:

```
0111010000011000
100011010000010000000010
1000100111000010
110000011111101000011111
```

- Memory & data
- Arrays & structs
- Integers & floats
- **RISC V assembly**
- Procedures & stacks
- Executables
- Memory & caches
- Processor Pipeline
- Performance
- Parallelism

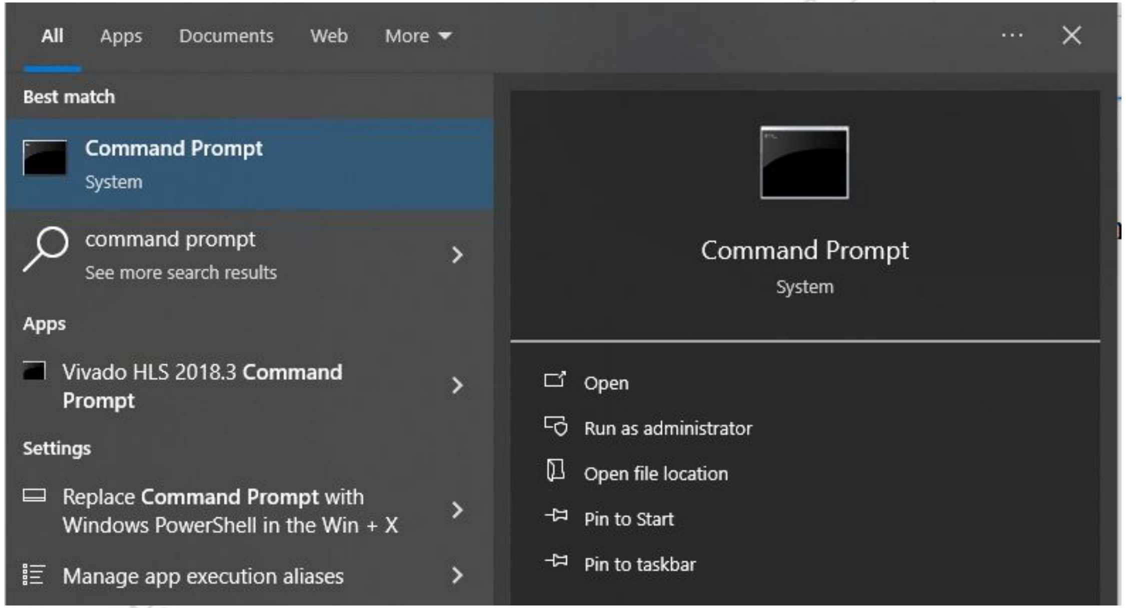
OS:

Computer system:



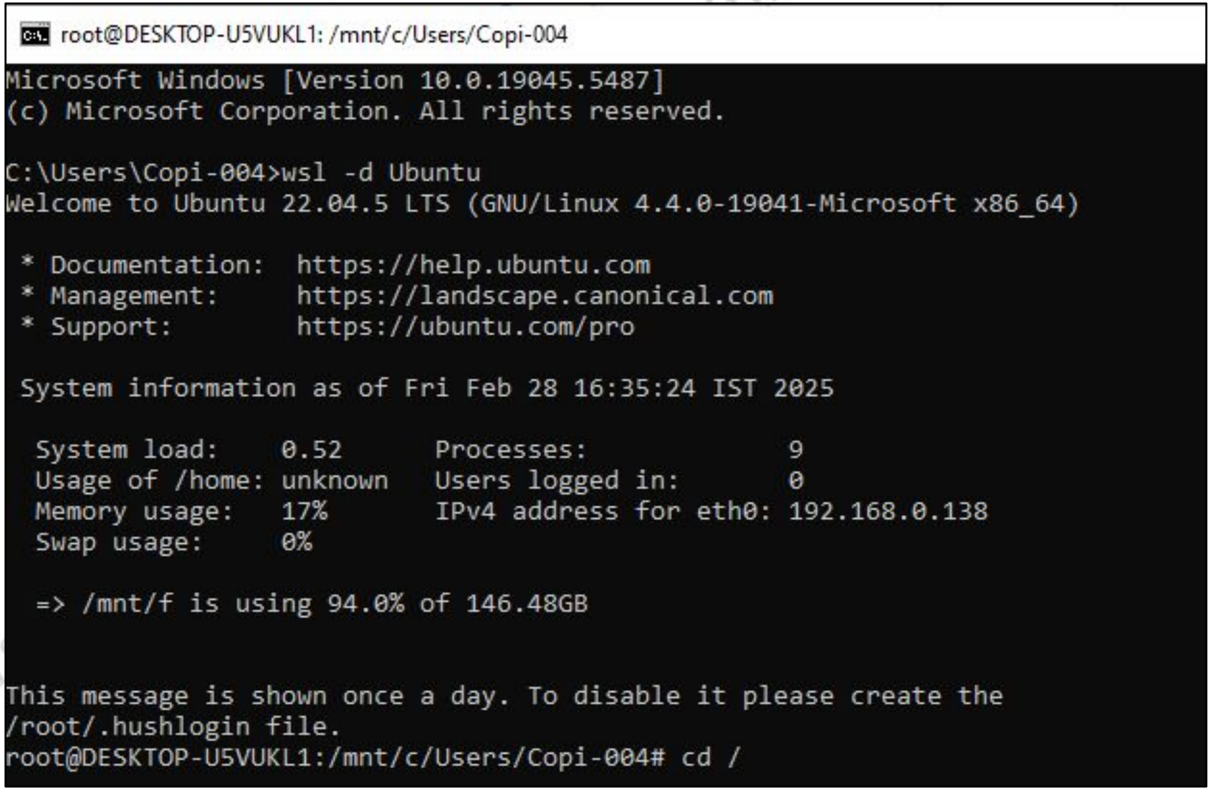
Launch the WSL software

- Search for command prompt in windows and type the following command



Spike Configuration

- Launch command prompt using wsl -d Ubuntu
- To change from existing windows directory to the main working directory in Ubuntu

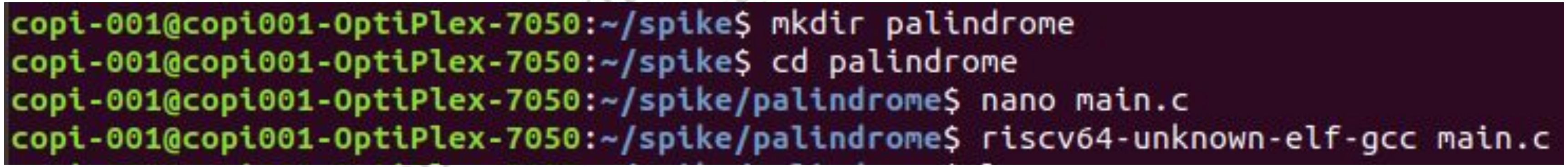


root@DESKTOP-U5VUKL1: /mnt/c/Users/Copi-004

```
Microsoft Windows [Version 10.0.19045.5487]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Copi-004>wsl -d Ubuntu
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 4.4.0-19041-Microsoft x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Fri Feb 28 16:35:24 IST 2025

  System load:    0.52      Processes:             9
  Usage of /home: unknown   Users logged in:       0
  Memory usage:   17%       IPv4 address for eth0: 192.168.0.138
  Swap usage:     0%

  => /mnt/f is using 94.0% of 146.48GB


This message is shown once a day. To disable it please create the
/root/.hushlogin file.
root@DESKTOP-U5VUKL1:/mnt/c/Users/Copi-004# cd /
```

Code Compilation using RISC-V GNU Toolchain

- Creating a Work space :
- To create a new folder in the directory type the command mkdir followed by the folder name, in which the saved program will be stored.
- Creating source codes:
- To create a new file to write the c code using editor type command nano followed by .c extension

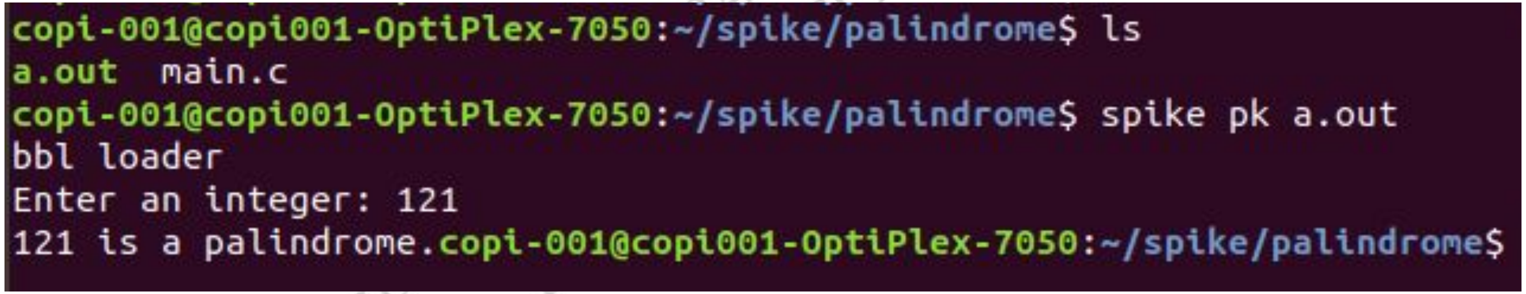


```
copi-001@copi001-OptiPlex-7050:~/spike$ mkdir palindrome
copi-001@copi001-OptiPlex-7050:~/spike$ cd palindrome
copi-001@copi001-OptiPlex-7050:~/spike/palindrome$ nano main.c
copi-001@copi001-OptiPlex-7050:~/spike/palindrome$ riscv64-unknown-elf-gcc main.c
```

Simulation on Spike RISC-V Simulator

Functional Simulation:

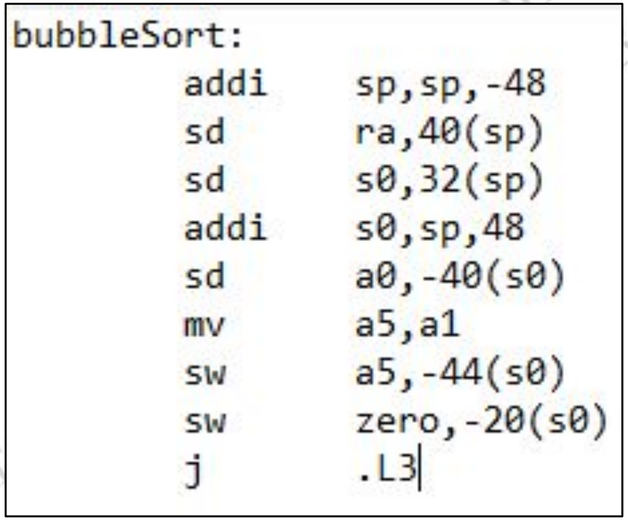
- To invoke the RISC-V Toolchain to compile the C program and generate Executable Linked Format (ELF) file type the following command followed the saved c program file



```
copi-001@copi001-OptiPlex-7050:~/spike/palindrome$ ls
a.out  main.c
copi-001@copi001-OptiPlex-7050:~/spike/palindrome$ spike pk a.out
bbl loader
Enter an integer: 121
121 is a palindrome.copi-001@copi001-OptiPlex-7050:~/spike/palindrome$
```

Assembly Instruction generation from source code

The RISC-V Toolchain enables code conversion from C source code to low level assembly version using the following command to generate assembly file with **.s extension**



```
bubbleSort:
        addi    sp,sp,-48
        sd      ra,40(sp)
        sd      s0,32(sp)
        addi    s0,sp,48
        sd      a0,-40(s0)
        mv      a5,a1
        sw      a5,-44(s0)
        sw      zero,-20(s0)
        j       .L3
```