Computer Systems Design

Lesson 5

Software toolchain structure for vN processor (RISC-V architecture)

Alexander Antonov, Assoc. Prof., ITMO University

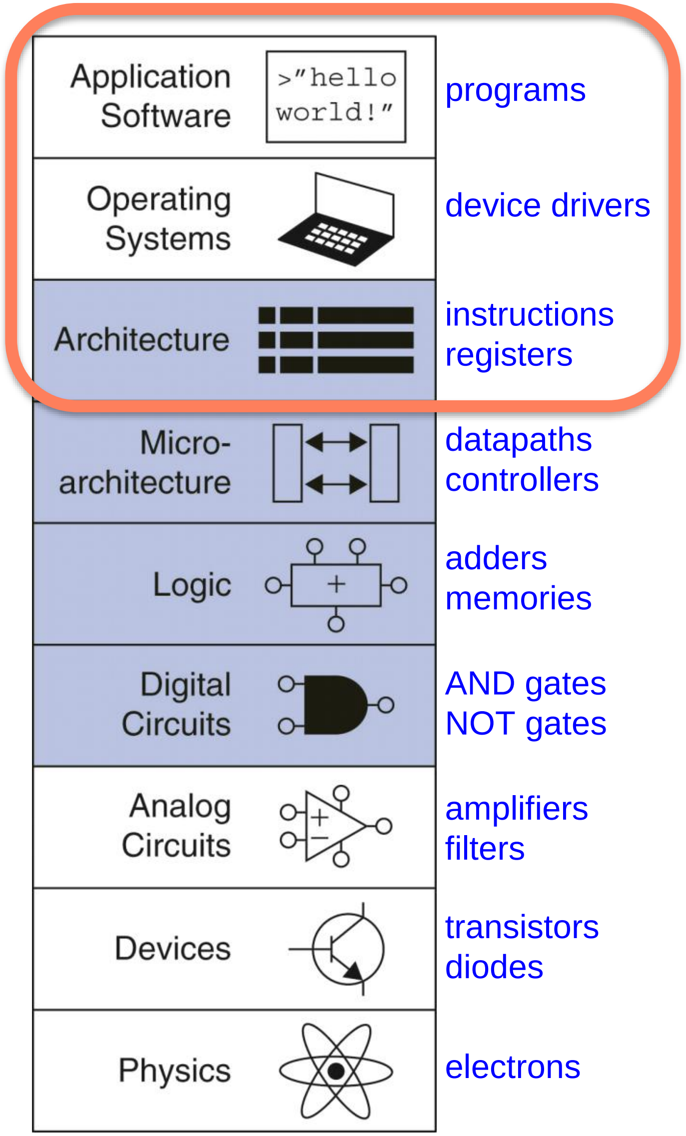
Hangzhou, 2025

Outline the lesson

- Types of software
- Types of languages
- Basic programming infrastructure
- Memory layouts
- Application Binary Interface
- CPU simulators

Types of software

- ***Bare-metal software*** – software that runs directly on hardware
- ***Operating system (OS)*** – bare-metal software that manages HW resources and provides environment for application software
- ***Hypervisor*** – bare-metal software that manages multiple operating systems and/or other SW components
- ***Application software*** – user-space programs that run within OS environment



High-level and low-level languages

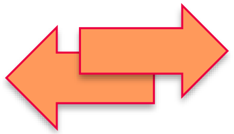
- ***High-level languages (HLL)*** – programming languages that abstract details of hardware organization

    instruction encoding, placement of data in memory, dynamic memory management, interaction with OS and peripherals, etc.

- ***Low-level languages*** – programming languages that expose details of hardware organization

High-level languages

- Expression of workload in terms of application domain
- Less qualification required
- Large space for optimizations
- Portability



Low-level languages

- Expression of workload in terms of system functions
- High engineering qualification
- Precise control over hardware resources
- Direct exposure of custom platform capabilities

ITMO

Examples of languages

- ***High-level languages***: Java, C#, Python, Scala, …

  many of them run programs in software virtual machines

- ***Intermediate category***: C, C++

  *abstract CPU instructions, but **not** placement of data in memory and dynamic memory management*

- ***Lowest-level language***: Assembly (ASM), machine code

  programs composed of CPU instructions defined in ISA

We will refer to C/C++ as high-level languages

Types of languages: translation and running

- ***Compiled*** - program is translated entirely to machine code and run afterwards

  ASM, C, C++ (in common cases), Rust, Haskell, Erlang, Go

- ***Interpreted*** - program is translated to machine code and executed on-the-go

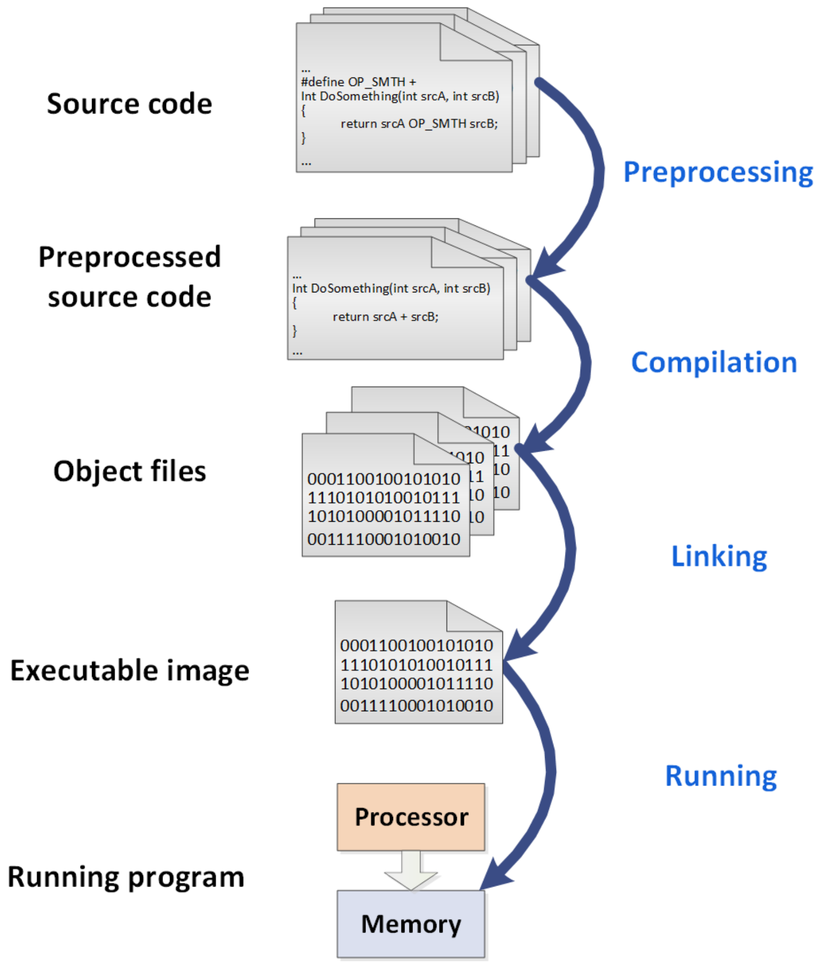
  Bash scripts, JavaScript, Python, Ruby, PHP

- ***Intermediate variants***: e.g. compilation to intermediate bytecode that is then interpreted and/or compiled to machine code in runtime (JIT) or before runtime (AOT)

  Java, Kotlin, Scala (JVM-based), C#

Low-level languages are compiled (interpreted by HW)

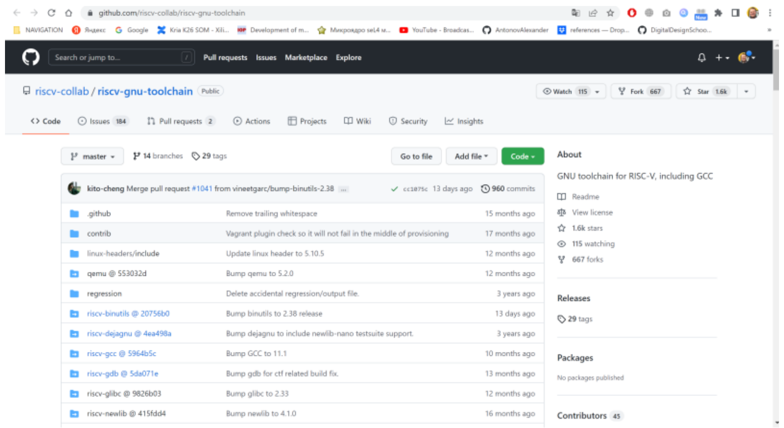
Typical SW compilation flow



ITMO

Basic programming infrastructure

- ***Compiler/cross-compiler*** – program compiling source code into object file (basic typical options: for bare-metal and Linux)
  `riscv64-unknown-elf-gcc`
- ***Linker*** – program composing object files in binary executable image
  `riscv64-unknown-elf-ld`
- ***Debugger*** – program that allows to debug the program running on actual system
  `riscv64-unknown-elf-gdb`

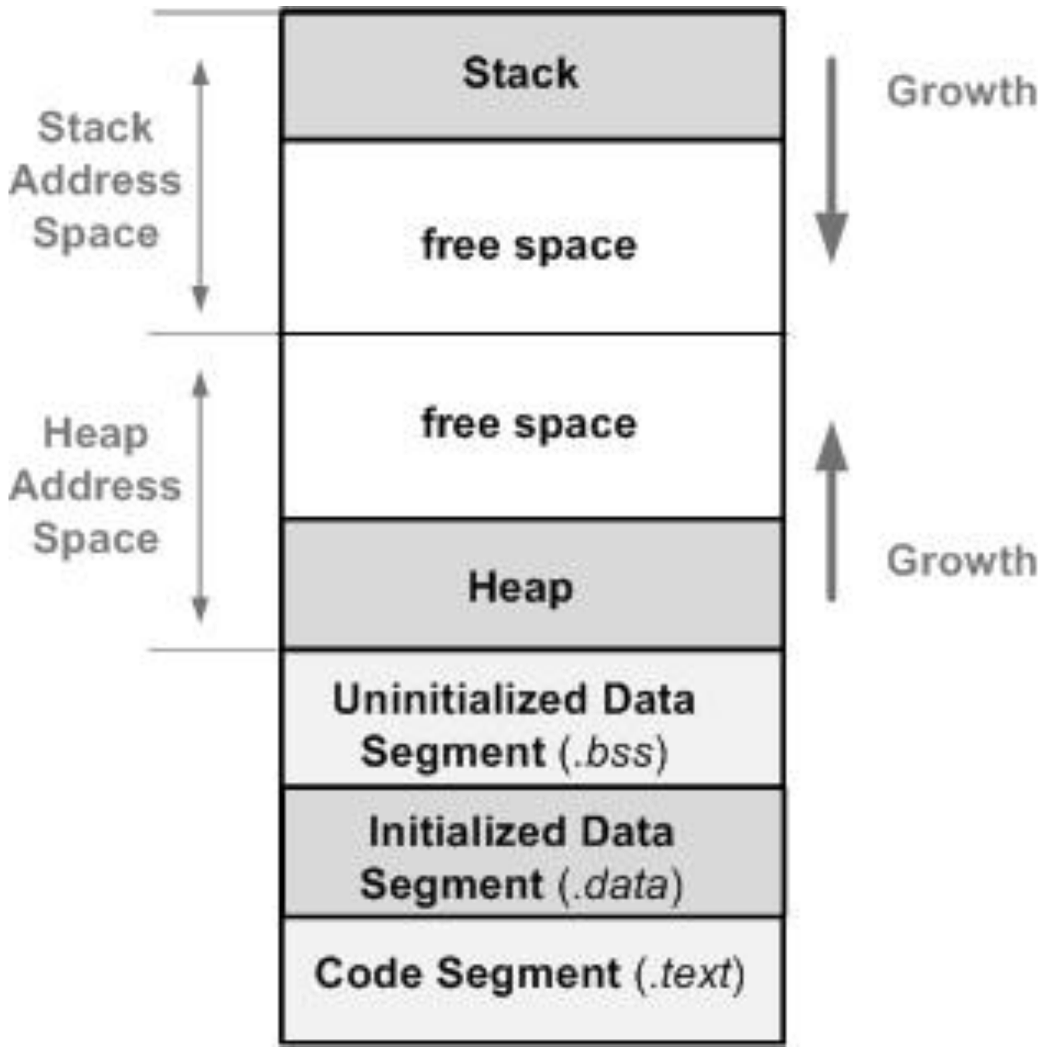


Visit for RISC-V tools exploration:
https://github.com/riscv-collab/riscv-gnu-toolchain/

C programs memory layout

Typical segments:

- Code segment (`.text`)
  CPU instructions
- Initialized data segments (`.data`)
  global and static variables
- Initialized read-only data (`.rodata`)
  global and static constants
- Uninitialized data segments (`.bss`)
  variable without initialization in source code
- Heap
  storage for dynamically created objects
- Stack
  temporary storage of data, new ranges allocated when procedure called



Application Binary Interface (ABI)

Convention of register usage by software

Defines ***recommended*** role of general-purpose registers for procedure calls, stacks, etc.

Followed by programmers and compiler developers to make composable software

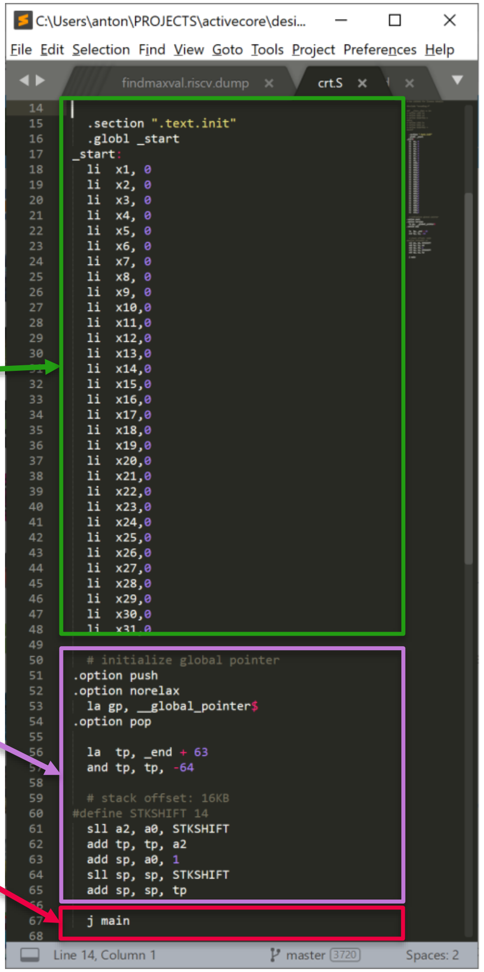
ISA spec, Vol. 1, unprivileged spec, p. 137

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5	t0	Temporary/alternate link register	Caller
x6–7	t1–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller
f0–7	ft0–7	FP temporaries	Caller
f8–9	fs0–1	FP saved registers	Callee
f10–11	fa0–1	FP arguments/return values	Caller
f12–17	fa2–7	FP arguments	Caller
f18–27	fs2–11	FP saved registers	Callee
f28–31	ft8–11	FP temporaries	Caller

Example: startup sequence (`crt.s`)

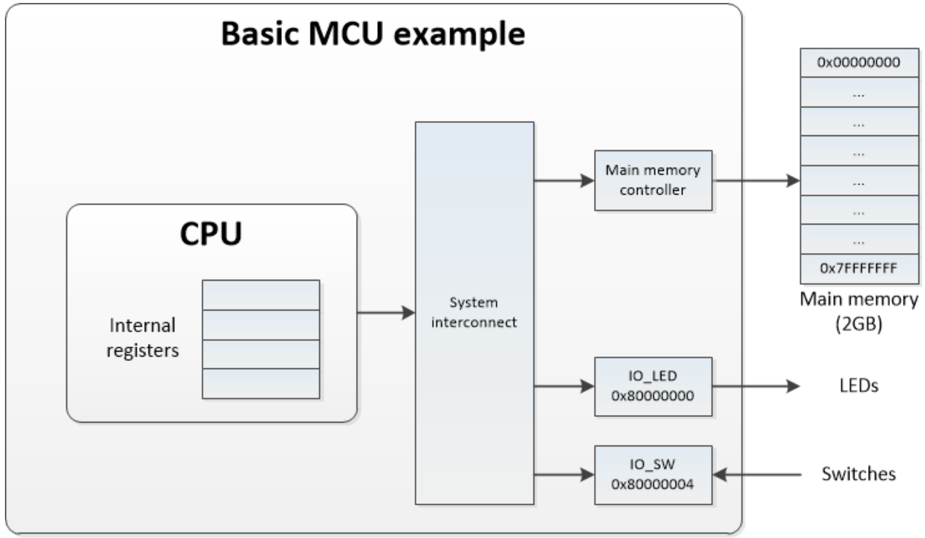
Minimum functionality:

- clearing general-purpose registers
- initialization of pointers
- branch to main program



Example: simple C baremetal program (`findmaxval.c`)

```c
#define IO_LED          (*(volatile unsigned int *)(0x80000000))
#define IO_SW           (*(volatile unsigned int *)(0x80000004))

unsigned int FindMaxVal(unsigned int* max_index, unsigned int datain[16])
{
  unsigned int max_val = 0;
  *max_index = 0;

  for (int i=0; i<16; i++) {
    if (datain[i] > max_val) {
      max_val = datain[i];
      *max_index = i;
    }
  }

  return max_val;
}

//---------------------------------------------------------------------------
// Main

int main( int argc, char* argv[] )
{
  unsigned int max_index;
  unsigned int max_val;
  unsigned int datain[16] = { 0x112233cc, 0x55aa55aa, 0x01010202, 0x44556677, 0x00000003, 0x00000004, 0x00000005,
0x00000006, 0x00000007, 0xdeadbeef, 0xfefe8800, 0x23344556, 0x05050505, 0x07070707, 0x99999999, 0xbadc0ffe };
  IO_LED = 0x55aa55aa;
  max_val = FindMaxVal(&max_index, datain);
  IO_LED = max_index;
  IO_LED = max_val;
  while (1) {}
}
```



Example: compilation command for simple RV32I MCU

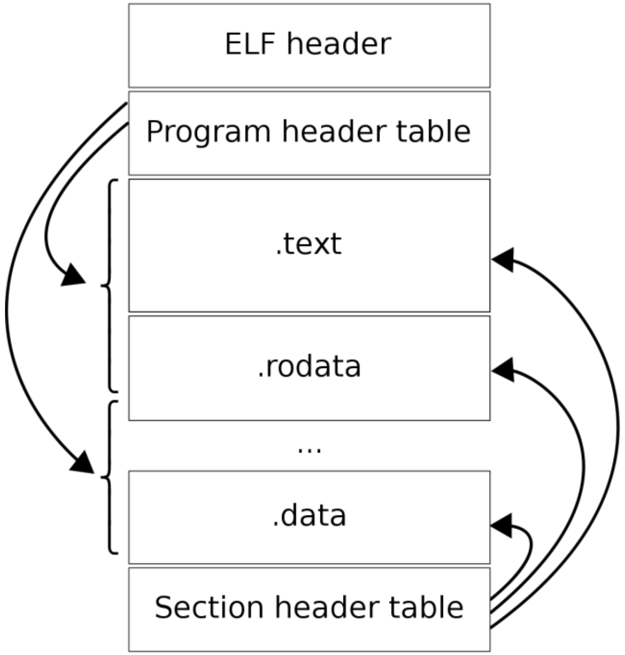
```
riscv64-unknown-elf-gcc -I./../env -I./../common -I./findmaxval -mcmodel=
medany -static -std=gnu99 -O2 -march=rv32im -mabi=ilp32 -lgcc -T <reference
to *.ld linker script> -o findmaxval.riscv ./findmaxval/findmaxval.c
./../common/isr.c ./../common/syscalls.c ./../common/crt.S ./../common/isr.S
```

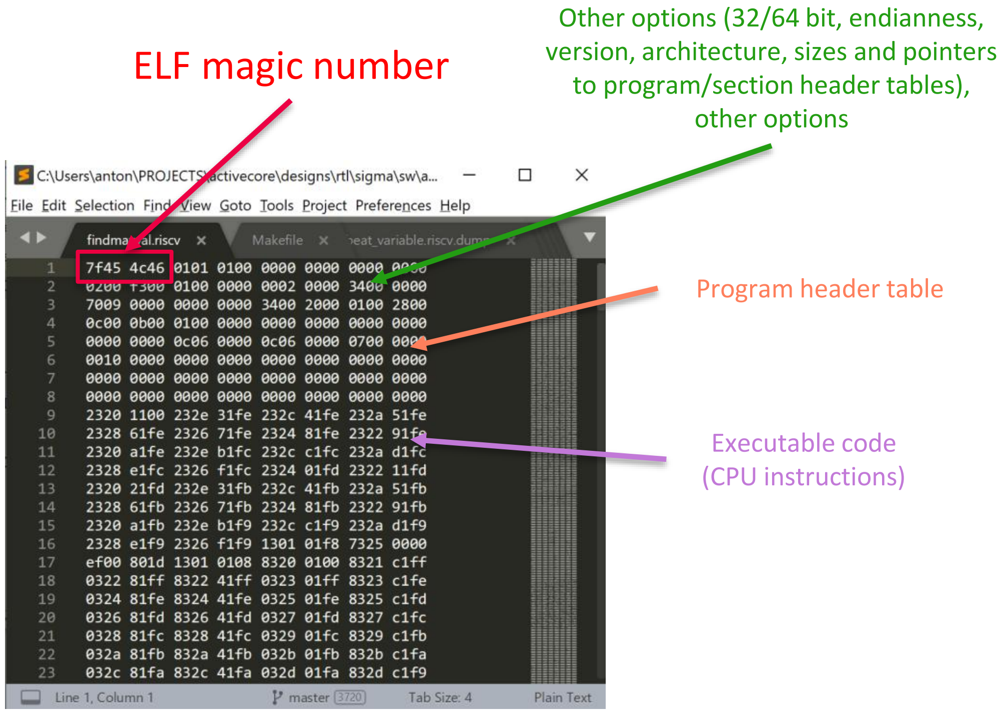
Structure of compilation command:

- Directories with header files
- Code model (medium data and code)
- Static linking (including libraries in executable)
- Standard: C99 standard
- Optimization level
- Target architecture: RV32I
- Application binary interface (long long – 2x 32-bit registers)
- Linking with libgcc.a library
- Linker script (control placement of data sections in memory)
- Name of output file
- Source files

Compiled image: ELF format

Executable and Linkable Format (ELF) – common standard file format for object code and executable files

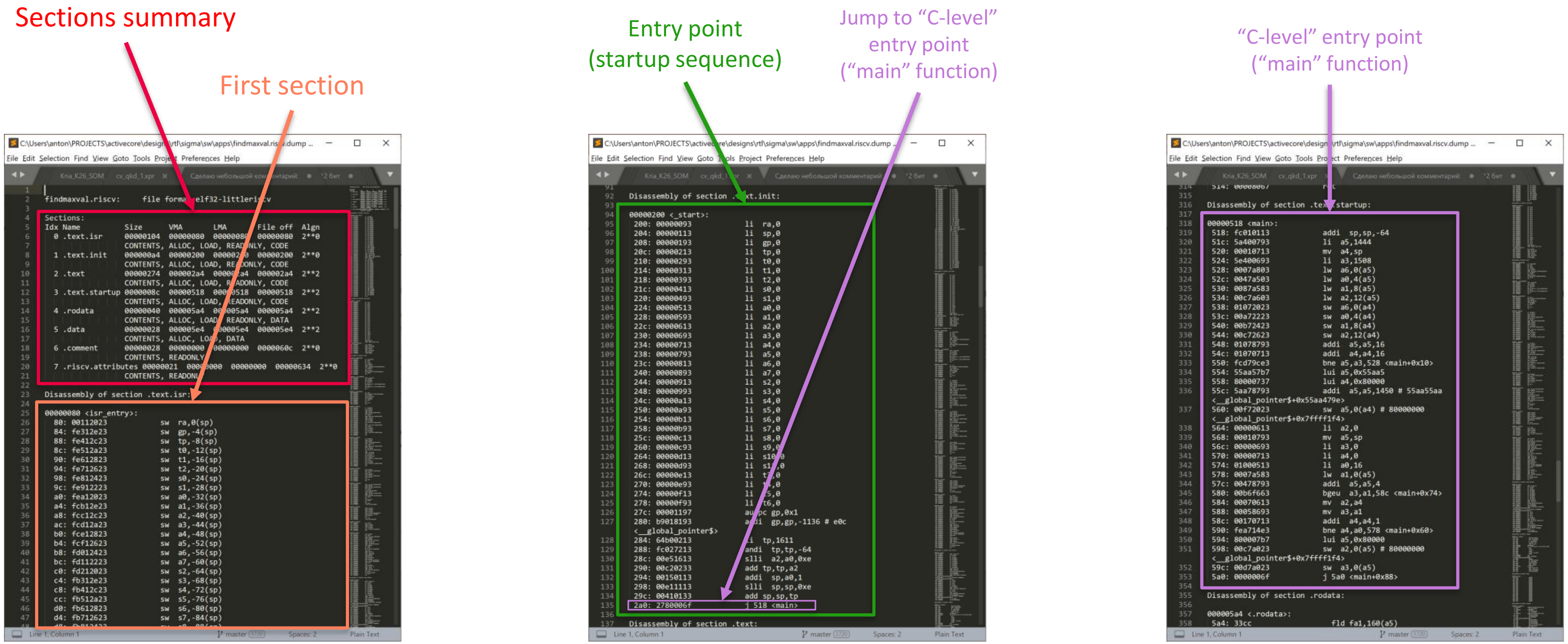




Example reference: https://en.wikipedia.org/wiki/Executable_and_Linkable_Format

Image disassembly

riscv64-unknown-elf-objdump --disassemble-all --disassemble-zeroes -h **findmaxval.riscv >**
findmaxval.riscv.dump



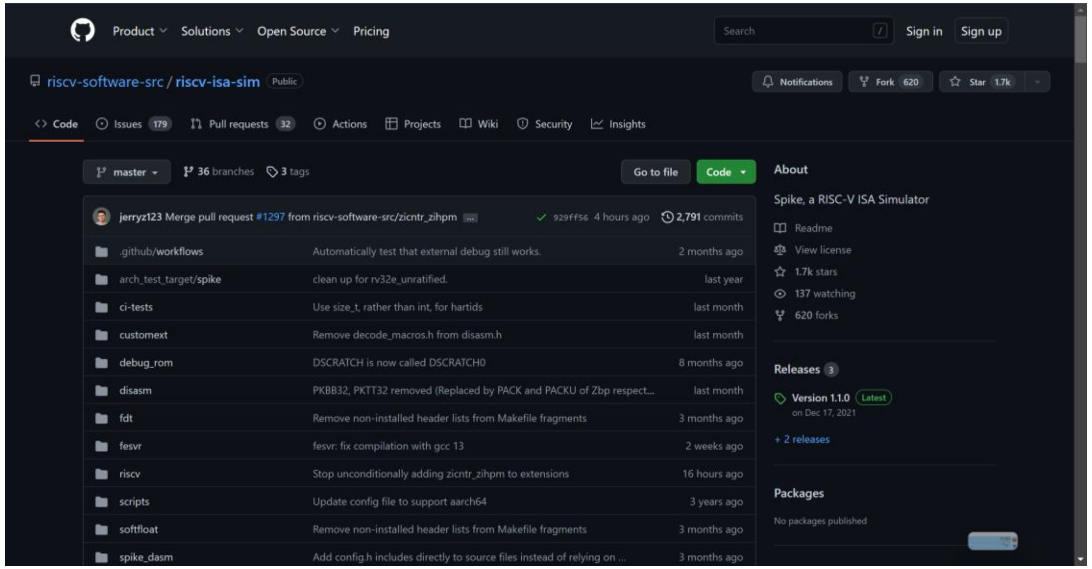
ITMO

Additional tooling: ISA simulators

ISA simulator: software able to run foreign binaries (e.g. RISC-V) on instrumental computer

Used for running and debugging software binaries

Spike – official ISA simulator for RISC-V

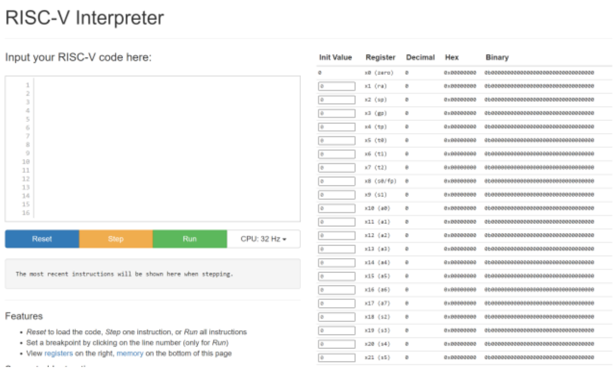


Repo: https://github.com/riscv-software-src/riscv-isa-sim

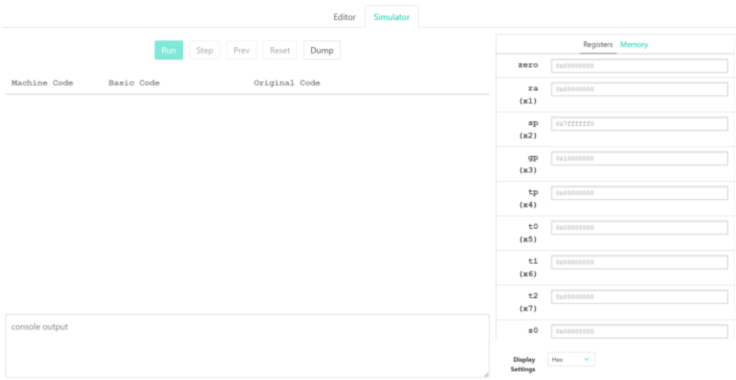
Typically used for SW debugging (by SW engineers) and as golden reference HW model (by HW engineers)

Additional tooling: online ISA simulators

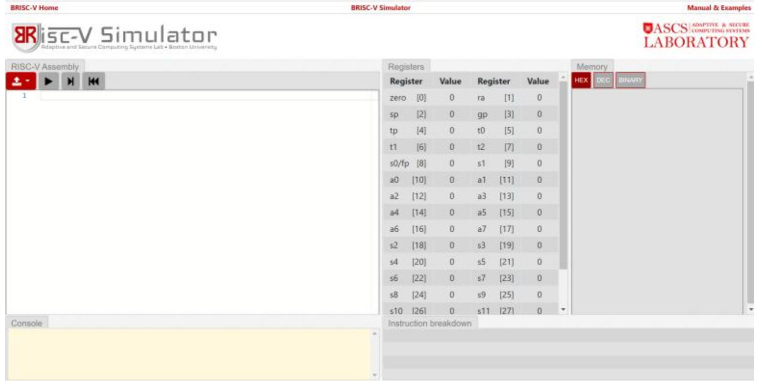
C/assembly code simulators with basic debugging capabilities



Cornell University RISC-V Interpreter
https://www.cs.cornell.edu/courses/cs3410/2019sp/riscv/interpreter/#



Venus
https://venus.kvakil.me/



ASCS Laboratory BRISC-V
https://ascslab.org/research/briscv/simulator/simulator.html

Typically used for entry-level education purposes (CPU architecture, low-level programming)

Additional tooling: full-system simulators

Additional tooling: full-system simulators

Full-system simulators – (micro)architectural simulators of CPU-enabled computer systems

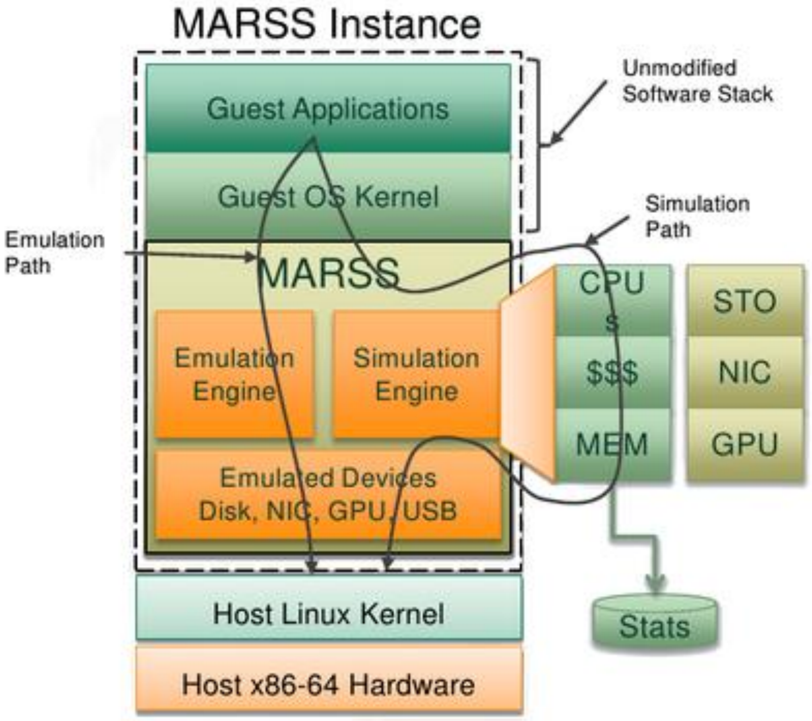
Common simulations:

- timing of instruction execution
- timing of communications and cache subsystem
- system calls to OS
- peripheral devices
- thermal/power effects

Projects:

- QEMU
- PTLsim/MARSS
- gem5
- Zsim
- Simics

and others …



Typically used by system architects

Thank you for the lesson!

Alexander Antonov, Assoc. Prof., antonov@itmo.ru

Hangzhou, 2025