



Computer Systems Design

Lesson 5

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

Alexander Antonov, Assoc. Prof., ITMO University

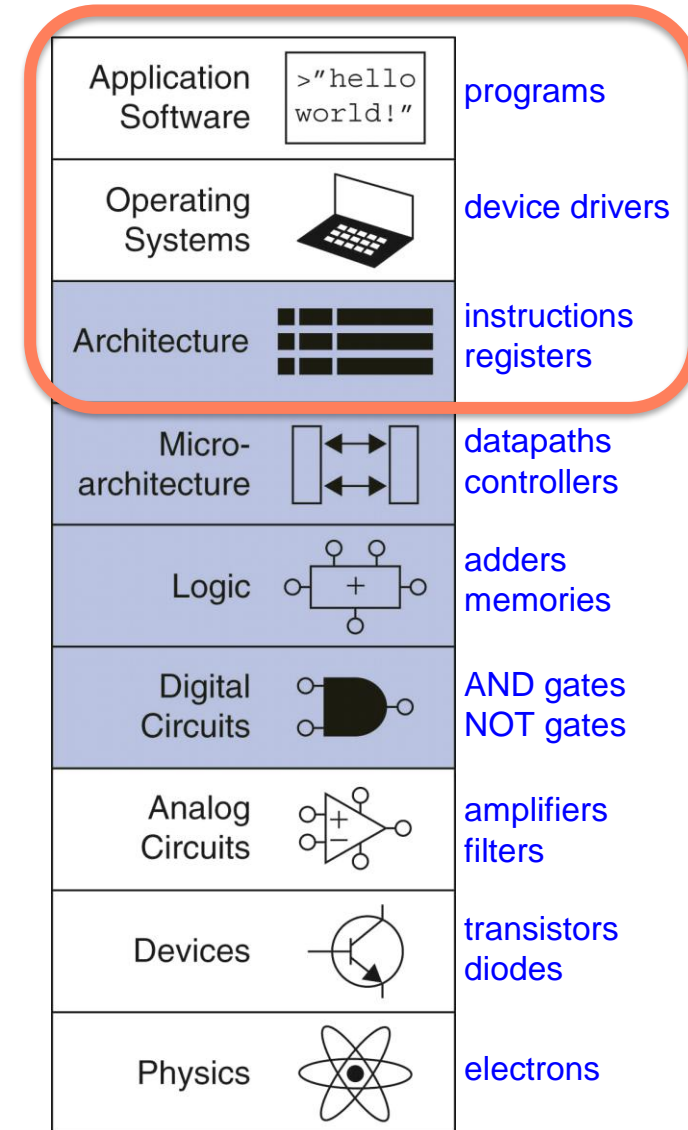
Hangzhou, 2025

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

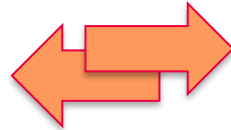


High-level and low-level languages

- **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

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

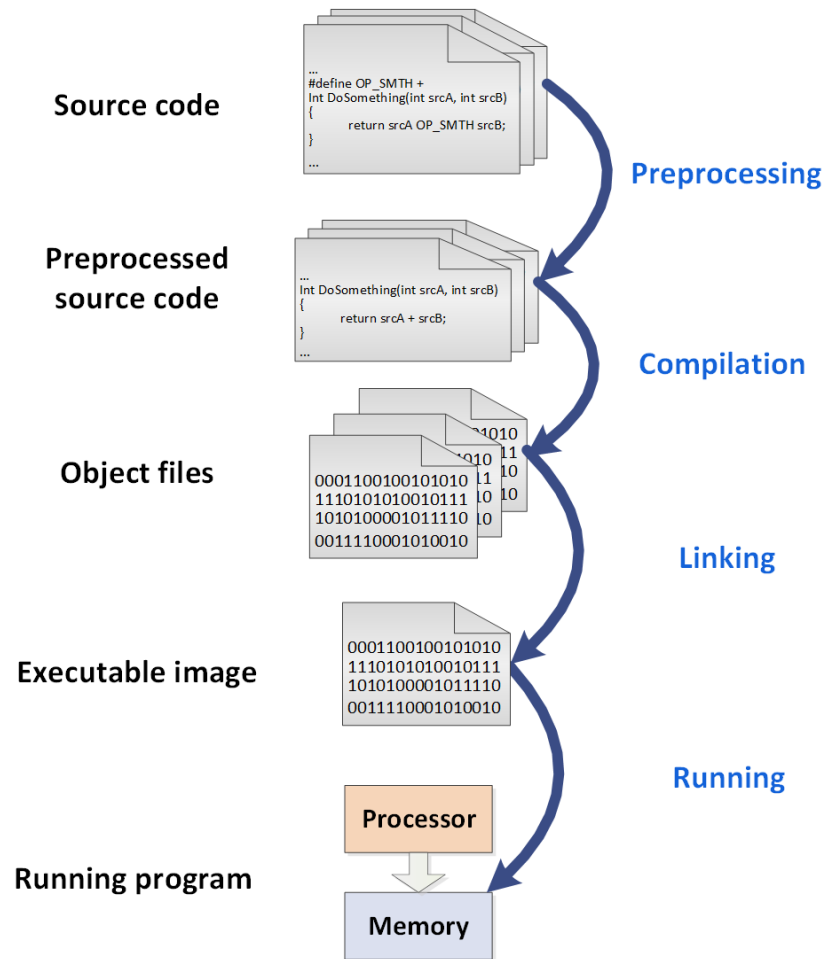
We will refer to C/C++ as high-level languages

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

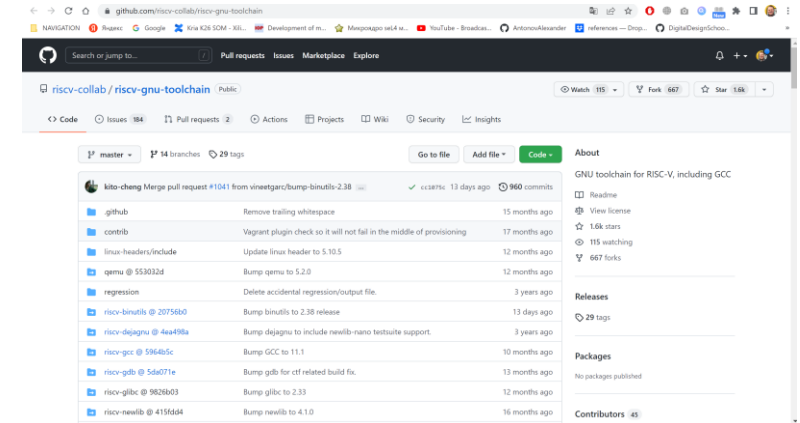


Basic programming infrastructure

- **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:

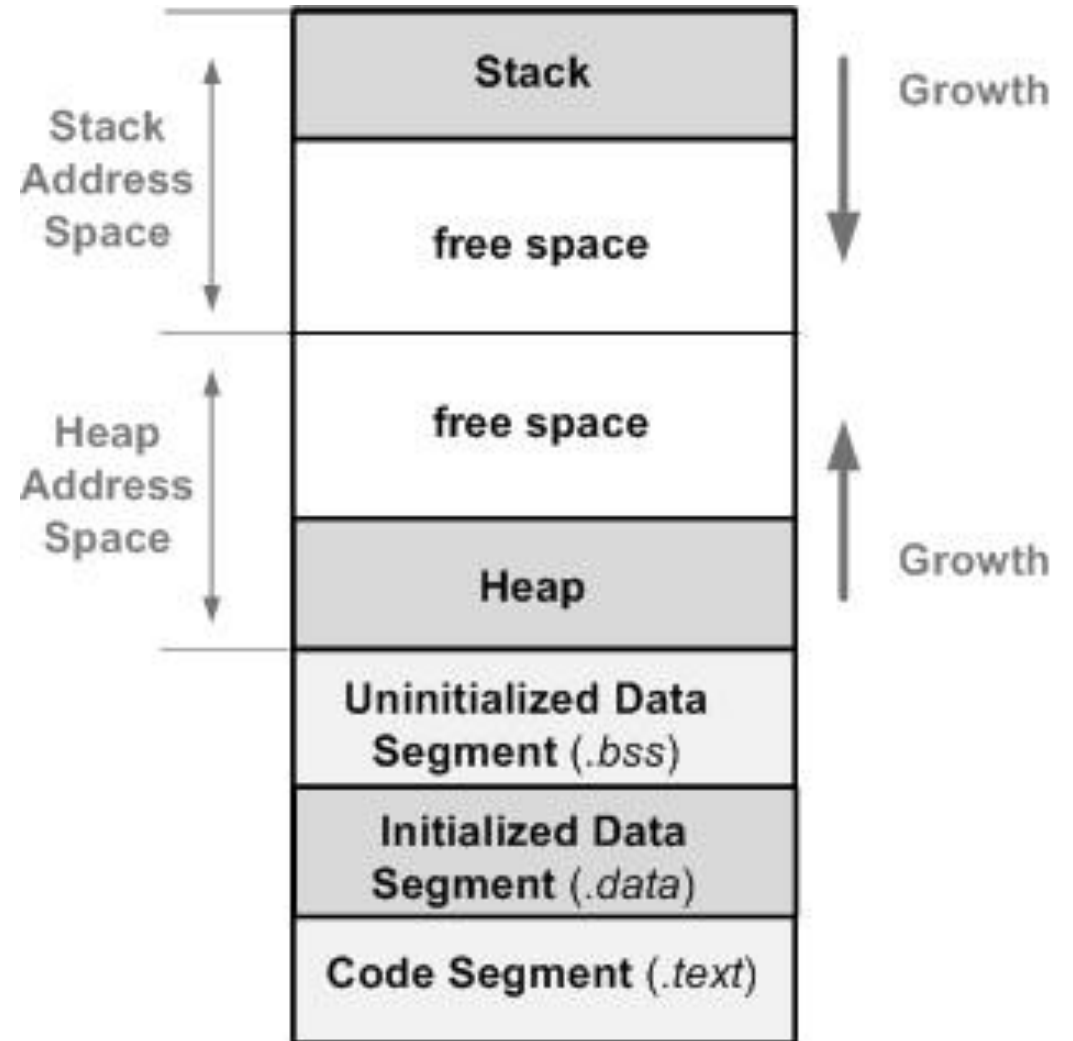
<https://github.com/riscv-collab/riscv-gnu-toolchain/>



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

```

14 |
15 | .section ".text.init"
16 | .globl _start
17 | _start:
18 | li x1, 0
19 | li x2, 0
20 | li x3, 0
21 | li x4, 0
22 | li x5, 0
23 | li x6, 0
24 | li x7, 0
25 | li x8, 0
26 | li x9, 0
27 | li x10, 0
28 | li x11, 0
29 | li x12, 0
30 | li x13, 0
31 | li x14, 0
32 | li x15, 0
33 | li x16, 0
34 | li x17, 0
35 | li x18, 0
36 | li x19, 0
37 | li x20, 0
38 | li x21, 0
39 | li x22, 0
40 | li x23, 0
41 | li x24, 0
42 | li x25, 0
43 | li x26, 0
44 | li x27, 0
45 | li x28, 0
46 | li x29, 0
47 | li x30, 0
48 | li x31, 0
49 |
50 | # initialize global pointer
51 | .option push
52 | .option norelax
53 | la gp, __global_pointer$
54 | .option pop
55 |
56 | la tp, _end + 63
57 | and tp, tp, -64
58 |
59 | # stack offset: 16KB
60 | #define STKSHIFT 14
61 | sll a2, a0, STKSHIFT
62 | add tp, tp, a2
63 | add sp, a0, 1
64 | sll sp, sp, STKSHIFT
65 | add sp, sp, tp
66 |
67 | j main
68 |

```

Example: simple C baremetal program (findmaxval.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;

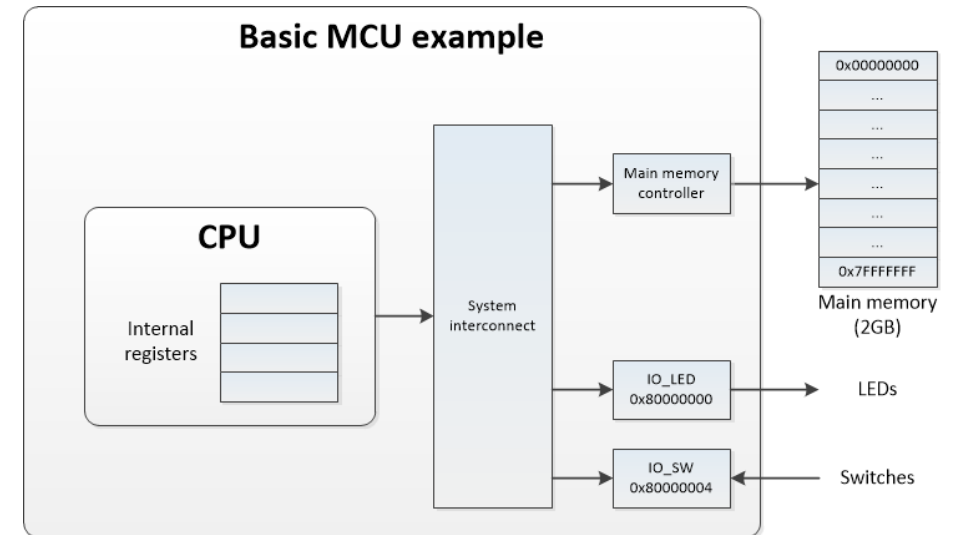
    for (int i=0; i<16; i++) {
        if (datain[i] > max_val) {
            max_val = datain[i];
            *max_index = i;
        }
    }

    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

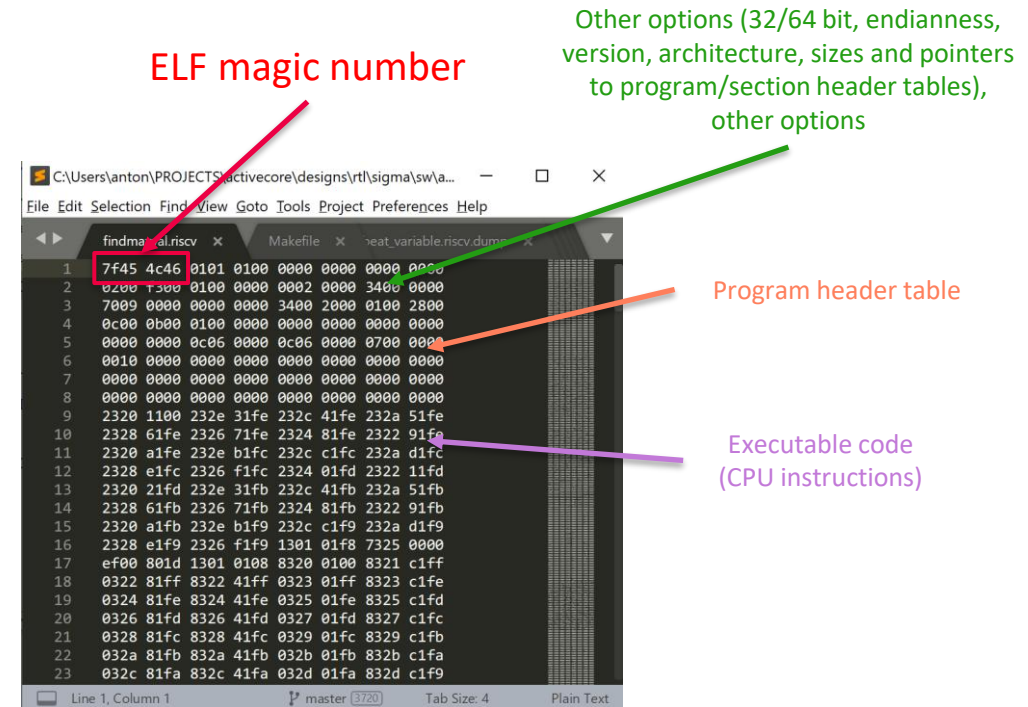
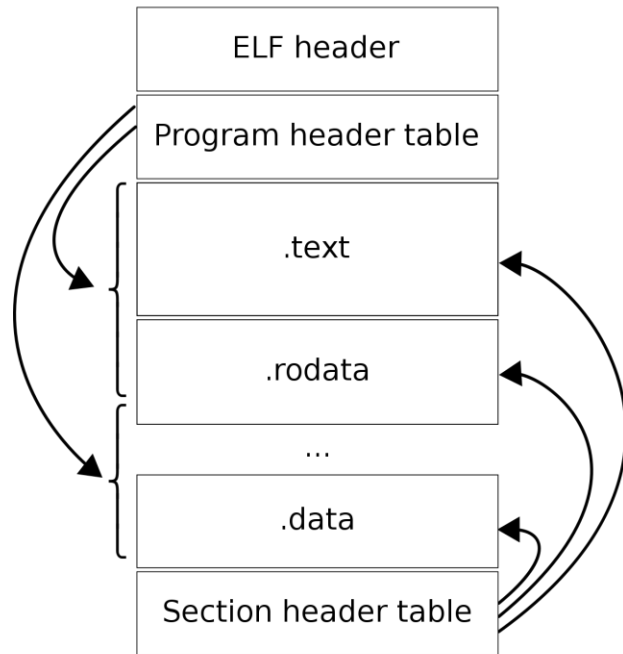
```
riscv64-unknown-elf-gcc -I../env -I../common -I./findmaxval -mcmmodel=  
medany -static -std=gnu99 -O2 -march=rv32im -mabi=ilp32 -lgcc -T <reference  
to *.ld linker script> -o findmaxval.riscv ./findmaxval/findmaxval.c  
./../common/isr.c ./../common/syscalls.c ./../common/crt.S ./../common/isr.S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

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`

Sections summary

First section

Entry point
(startup sequence)

Jump to "C-level"
entry point
("main" function)

"C-level" entry point
("main" function)

```

Sections:
Idx Name          Size      VMA           LMA           File off  Algn
0 .text.isr       00000104 00000000 00000000 00000000 2**0
1 .text.init      00000044 00000200 00000200 00000200 2**0
2 .text           00000274 000002a4 000002a4 000002a4 2**2
3 .text.startup   0000008c 00000518 00000518 00000518 2**2
4 .rodata         00000040 000005a4 000005a4 000005a4 2**2
5 .data           00000028 000005e4 000005e4 000005e4 2**2
6 .comment        00000028 00000600 00000600 0000060c 2**0
7 .riscv.attributes 00000021 00000600 00000600 00000634 2**0

Disassembly of section .text.isr:
00000000 <isr_entry>:
00: 00112023      sw ra,0(sp)
04: fe312023      sw tp,-4(sp)
08: fe412023      sw tp,-8(sp)
0c: fe512023      sw tp,-12(sp)
10: fe612023      sw tp,-16(sp)
14: fe712023      sw tp,-20(sp)
18: fe812023      sw tp,-24(sp)
1c: fe912023      sw tp,-28(sp)
20: fea12023      sw tp,-32(sp)
24: feb12023      sw tp,-36(sp)
28: fec12023      sw tp,-40(sp)
2c: fed12023      sw tp,-44(sp)
30: fee12023      sw tp,-48(sp)
34: fef12023      sw tp,-52(sp)
38: fd012023      sw tp,-56(sp)
3c: fd112023      sw tp,-60(sp)
40: fd212023      sw tp,-64(sp)
44: fd312023      sw tp,-68(sp)
48: fd412023      sw tp,-72(sp)
4c: fd512023      sw tp,-76(sp)
50: fd612023      sw tp,-80(sp)
54: fd712023      sw tp,-84(sp)

```

```

Disassembly of section .text.init:
00000200 <start>:
000: 00000093      li ra,0
004: 00000113      li sp,0
008: 00000103      li tp,0
00c: 00000213      li tp,0
010: 00000293      li tp,0
014: 00000313      li tp,0
018: 00000393      li tp,0
01c: 00000413      li tp,0
020: 00000493      li tp,0
024: 00000513      li tp,0
028: 00000593      li tp,0
02c: 00000613      li tp,0
030: 00000693      li tp,0
034: 00000713      li tp,0
038: 00000793      li tp,0
03c: 00000813      li tp,0
040: 00000893      li tp,0
044: 00000913      li tp,0
048: 00000993      li tp,0
04c: 00000a13      li tp,0
050: 00000a93      li tp,0
054: 00000b13      li tp,0
058: 00000b93      li tp,0
05c: 00000c13      li tp,0
060: 00000c93      li tp,0
064: 00000d13      li tp,0
068: 00000d93      li tp,0
06c: 00000e13      li tp,0
070: 00000e93      li tp,0
074: 00000f13      li tp,0
078: 00000f93      li tp,0
07c: 00001013      li tp,0
080: 00001093      li tp,0
084: 00001113      li tp,0
088: 00001193      li tp,0
08c: 00001213      li tp,0
090: 00001293      li tp,0
094: 00001313      li tp,0
098: 00001393      li tp,0
09c: 00001413      li tp,0
0a0: 00001493      li tp,0
0a4: 00001513      li tp,0
0a8: 00001593      li tp,0
0ac: 00001613      li tp,0
0b0: 00001693      li tp,0
0b4: 00001713      li tp,0
0b8: 00001793      li tp,0
0bc: 00001813      li tp,0
0c0: 00001893      li tp,0
0c4: 00001913      li tp,0
0c8: 00001993      li tp,0
0cc: 00001a13      li tp,0
0d0: 00001a93      li tp,0
0d4: 00001b13      li tp,0
0d8: 00001b93      li tp,0
0dc: 00001c13      li tp,0
0e0: 00001c93      li tp,0
0e4: 00001d13      li tp,0
0e8: 00001d93      li tp,0
0ec: 00001e13      li tp,0
0f0: 00001e93      li tp,0
0f4: 00001f13      li tp,0
0f8: 00001f93      li tp,0
0fc: 00002013      li tp,0

```

```

Disassembly of section .text.startup:
00000518 <main>:
518: fcd10113      addi sp,sp,-64
51c: 5e400693      li a5,1444
520: 00010713      mv a4,sp
524: 5e400693      li a3,1508
528: 0007a803      lw a6,0(a5)
52c: 0007a503      lw a0,4(a5)
530: 0007a583      lw a1,8(a5)
534: 00c7a603      lw a2,12(a5)
538: 0107a823      sw a6,0(a4)
53c: 00072223      sw a0,4(a4)
540: 00072423      sw a1,8(a4)
544: 00c72623      sw a2,12(a4)
548: 0107a823      addi a5,a5,16
54c: 0107a713      addi a4,a4,16
550: fcd79ce3      bne a5,a3,528 <main+0x10>
554: 55aa5707      lui a5,0x55aa5
558: 00000727      lui a4,0x00000
55c: 5aa78793      addi a5,a5,1450 # 55aa55aa
560: 00772023      sw a5,0(a4) # 80000000
564: 00000613      li a2,0
568: 00010793      mv a5,sp
56c: 00000693      li a3,0
570: 00000713      li a4,0
574: 01000513      li a0,16
578: 0007a583      lw a1,0(a5)
57c: 0007a793      addi a5,a5,4
580: 0006f663      bgeu a3,a1,58c <main+0x74>
584: 0007a613      mv a2,a4
588: 00058093      mv a3,a1
58c: 00170713      addi a4,a4,1
590: fea714e3      bne a4,a0,578 <main+0x60>
594: 000007b7      lui a5,0x00000
598: 0007a823      sw a2,0(a5) # 80000000
59c: 0007a023      sw a3,0(a5)
5a0: 0000006f      j 5a0 <main+0x88>

```

Additional tooling: ISA simulators

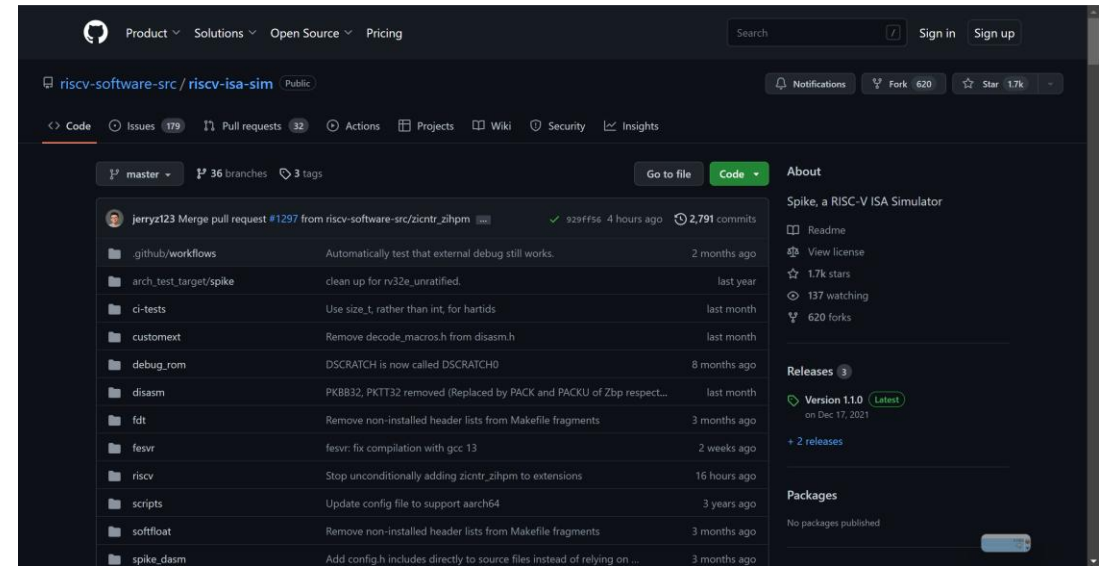
ISA simulator: software able to run foreign binaries (e.g. RISC-V) on instrumental computer

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

RISC-V Interpreter

Input your RISC-V code here:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
```

Reset Stop Run CPU: 32 Hz

The most recent instructions will be shown here when stepping.

Features

- Reset to load the code, Step one instruction, or Run all instructions
- Set a breakpoint by clicking on the line number (only for Run)
- View registers on the right, memory on the bottom of the page

Init Value	Register	Decimal	Hex	Binary
0	x0 (zero)	0	0x00000000	00000000000000000000000000000000
0	x1 (ra)	0	0x00000000	00000000000000000000000000000000
0	x2 (sp)	0	0x00000000	00000000000000000000000000000000
0	x3 (gp)	0	0x00000000	00000000000000000000000000000000
0	x4 (tp)	0	0x00000000	00000000000000000000000000000000
0	x5 (t0)	0	0x00000000	00000000000000000000000000000000
0	x6 (t1)	0	0x00000000	00000000000000000000000000000000
0	x7 (t2)	0	0x00000000	00000000000000000000000000000000
0	x8 (t3)	0	0x00000000	00000000000000000000000000000000
0	x9 (t4)	0	0x00000000	00000000000000000000000000000000
0	x10 (t5)	0	0x00000000	00000000000000000000000000000000
0	x11 (t6)	0	0x00000000	00000000000000000000000000000000
0	x12 (t7)	0	0x00000000	00000000000000000000000000000000
0	x13 (t8)	0	0x00000000	00000000000000000000000000000000
0	x14 (t9)	0	0x00000000	00000000000000000000000000000000
0	x15 (s0)	0	0x00000000	00000000000000000000000000000000
0	x16 (s1)	0	0x00000000	00000000000000000000000000000000
0	x17 (s2)	0	0x00000000	00000000000000000000000000000000
0	x18 (s3)	0	0x00000000	00000000000000000000000000000000
0	x19 (s4)	0	0x00000000	00000000000000000000000000000000
0	x20 (s5)	0	0x00000000	00000000000000000000000000000000
0	x21 (s6)	0	0x00000000	00000000000000000000000000000000
0	x22 (s7)	0	0x00000000	00000000000000000000000000000000
0	x23 (s8)	0	0x00000000	00000000000000000000000000000000
0	x24 (s9)	0	0x00000000	00000000000000000000000000000000
0	x25 (s10)	0	0x00000000	00000000000000000000000000000000
0	x26 (s11)	0	0x00000000	00000000000000000000000000000000
0	x27 (s12)	0	0x00000000	00000000000000000000000000000000
0	x28 (s13)	0	0x00000000	00000000000000000000000000000000
0	x29 (s14)	0	0x00000000	00000000000000000000000000000000
0	x30 (s15)	0	0x00000000	00000000000000000000000000000000

Editor Simulator

Run Step Prev Reset Dump

Machine Code Basic Code Original Code

Registers Memory

zero 00000000

ra 00000000

sp 00000000

gp 00000000

tp 00000000

t0 00000000

t1 00000000

t2 00000000

t3 00000000

t4 00000000

t5 00000000

t6 00000000

t7 00000000

t8 00000000

t9 00000000

s0 00000000

s1 00000000

s2 00000000

s3 00000000

s4 00000000

s5 00000000

s6 00000000

s7 00000000

s8 00000000

s9 00000000

s10 00000000

s11 00000000

s12 00000000

s13 00000000

s14 00000000

s15 00000000

Display Settings

console output

BRISC-V Simulator

Registers Memory

zero [0] 0 ra [1] 0

sp [2] 0 gp [3] 0

tp [4] 0 t0 [5] 0

t1 [6] 0 t2 [7] 0

s0/tp [8] 0 s1 [9] 0

a0 [10] 0 a1 [11] 0

a2 [12] 0 a3 [13] 0

a4 [14] 0 a5 [15] 0

a6 [16] 0 a7 [17] 0

s2 [18] 0 s3 [19] 0

s4 [20] 0 s5 [21] 0

s6 [22] 0 s7 [23] 0

s8 [24] 0 s9 [25] 0

s10 [26] 0 s11 [27] 0

Instruction breakdown

Cornell University RISC-V Interpreter
<https://www.cs.cornell.edu/courses/cs4410/2019sp/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

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

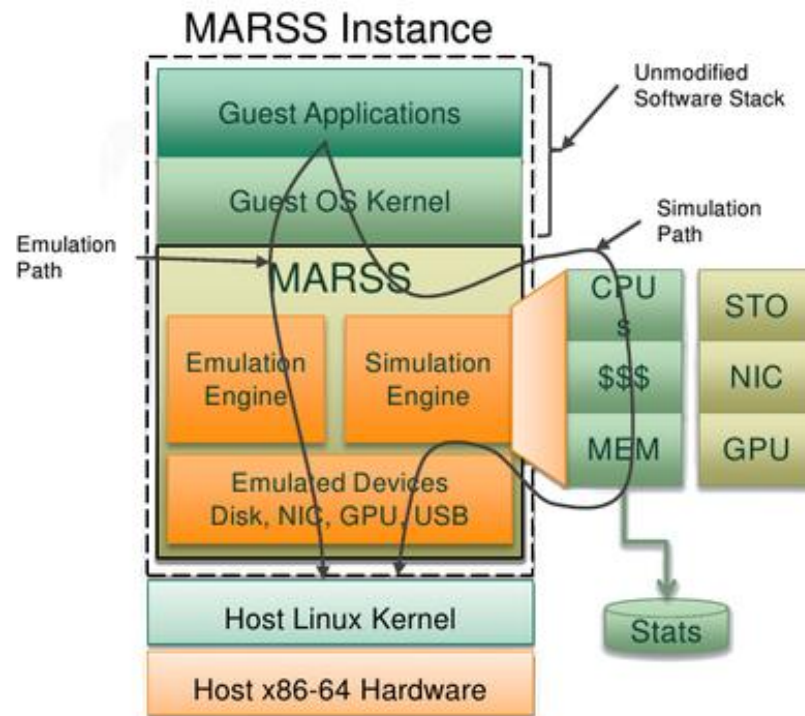
Common simulations:

- 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