Computersysteme

QtRVSim – RISC-V Simulator

Tobias Schwarzinger

18.03.2025

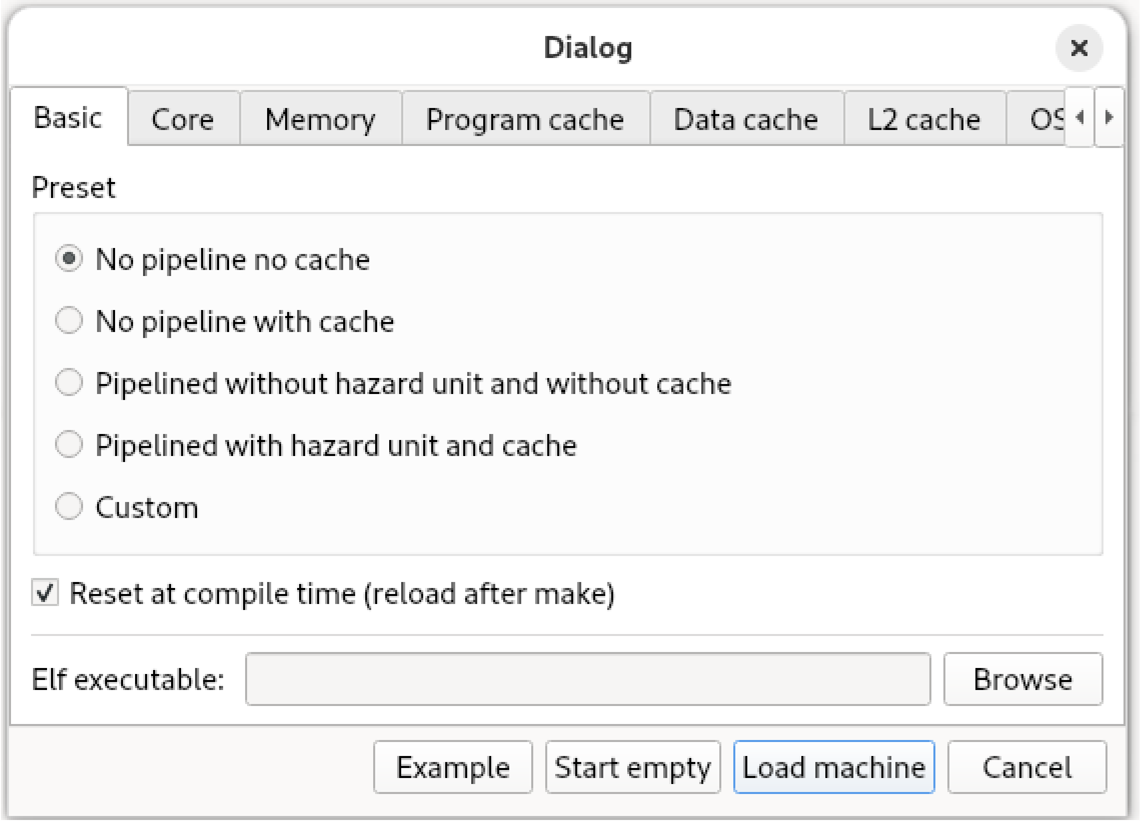
QtRVSim

QtRVSim

- Developed at the Czech Technical University
- RISC-V simulator
  - Instruction Set (Use No Pipeline, No Cache)
  - Microarchitecture (Pipeline, Caches)
- Assembly editor

- GitHub: https://github.com/cvut/qtrvsim
- Online Version: https://comparch.edu.cvut.cz/qtrvsim/app/

QtRVSim - Startup



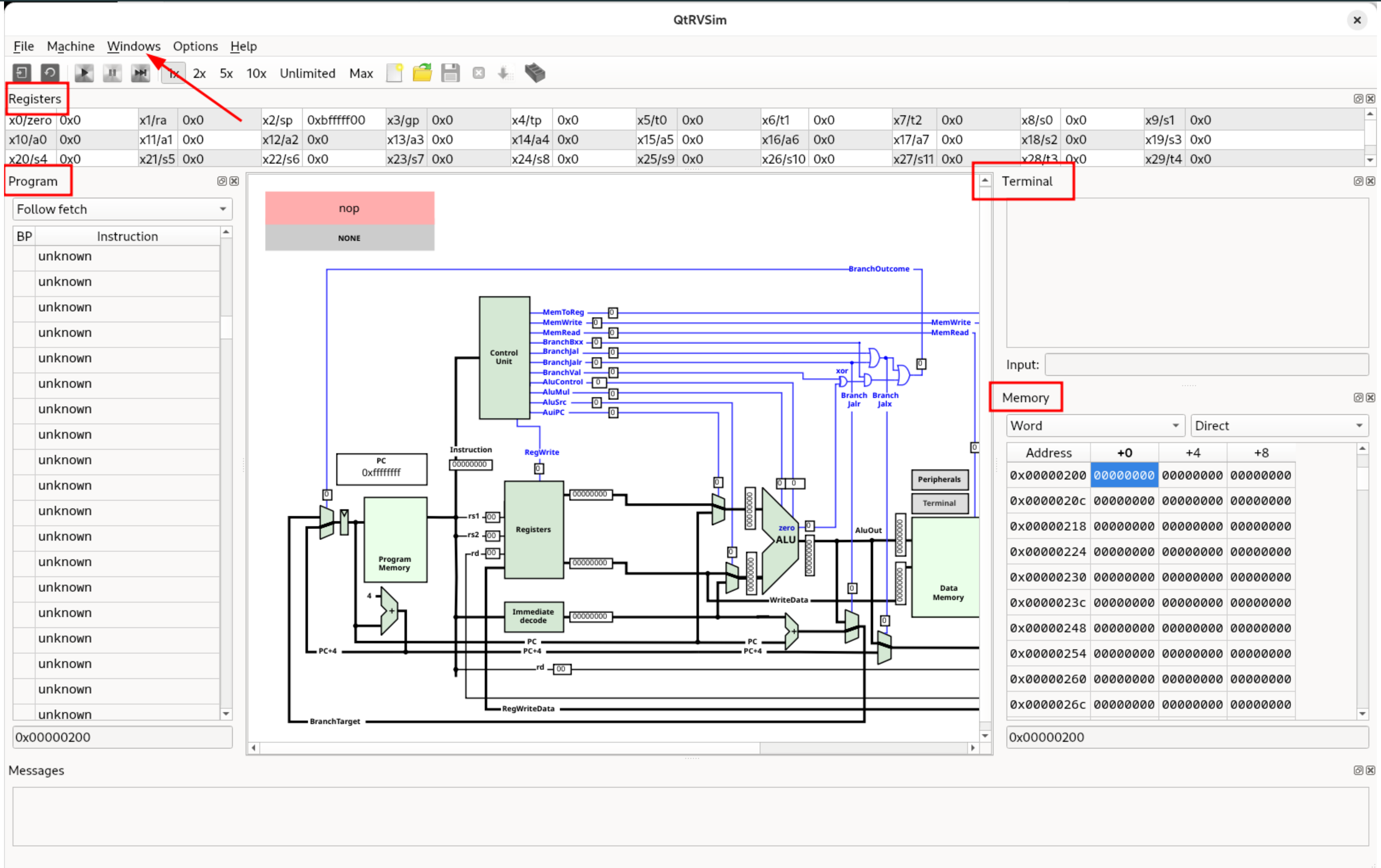
Preset

- ● No pipeline no cache
- ○ No pipeline with cache
- ○ Pipelined without hazard unit and without cache
- ○ Pipelined with hazard unit and cache
- ○ Custom

☑ Reset at compile time (reload after make)

Elf executable:

QtRVSim - Configuration



Registers

x0/zero	0x0	x1/ra	0x0	x2/sp	0xbfffff00	x3/gp	0x0	x4/tp	0x0	x5/t0	0x0	x6/t1	0x0	x7/t2	0x0	x8/s0	0x0	x9/s1	0x0
x10/a0	0x0	x11/a1	0x0	x12/a2	0x0	x13/a3	0x0	x14/a4	0x0	x15/a5	0x0	x16/a6	0x0	x17/a7	0x0	x18/s2	0x0	x19/s3	0x0
x20/s4	0x0	x21/s5	0x0	x22/s6	0x0	x23/s7	0x0	x24/s8	0x0	x25/s9	0x0	x26/s10	0x0	x27/s11	0x0	x28/t3	0x0	x29/t4	0x0

Program

Follow fetch

BP	Instruction
	unknown
	unknown
	unknown
	unknown
	unknown
	unknown
	unknown
	unknown
	unknown
	unknown
	unknown
	unknown
	unknown
	unknown
	unknown
	unknown
	unknown
	unknown
	unknown

0x00000200

Terminal

Input:

Memory

Word

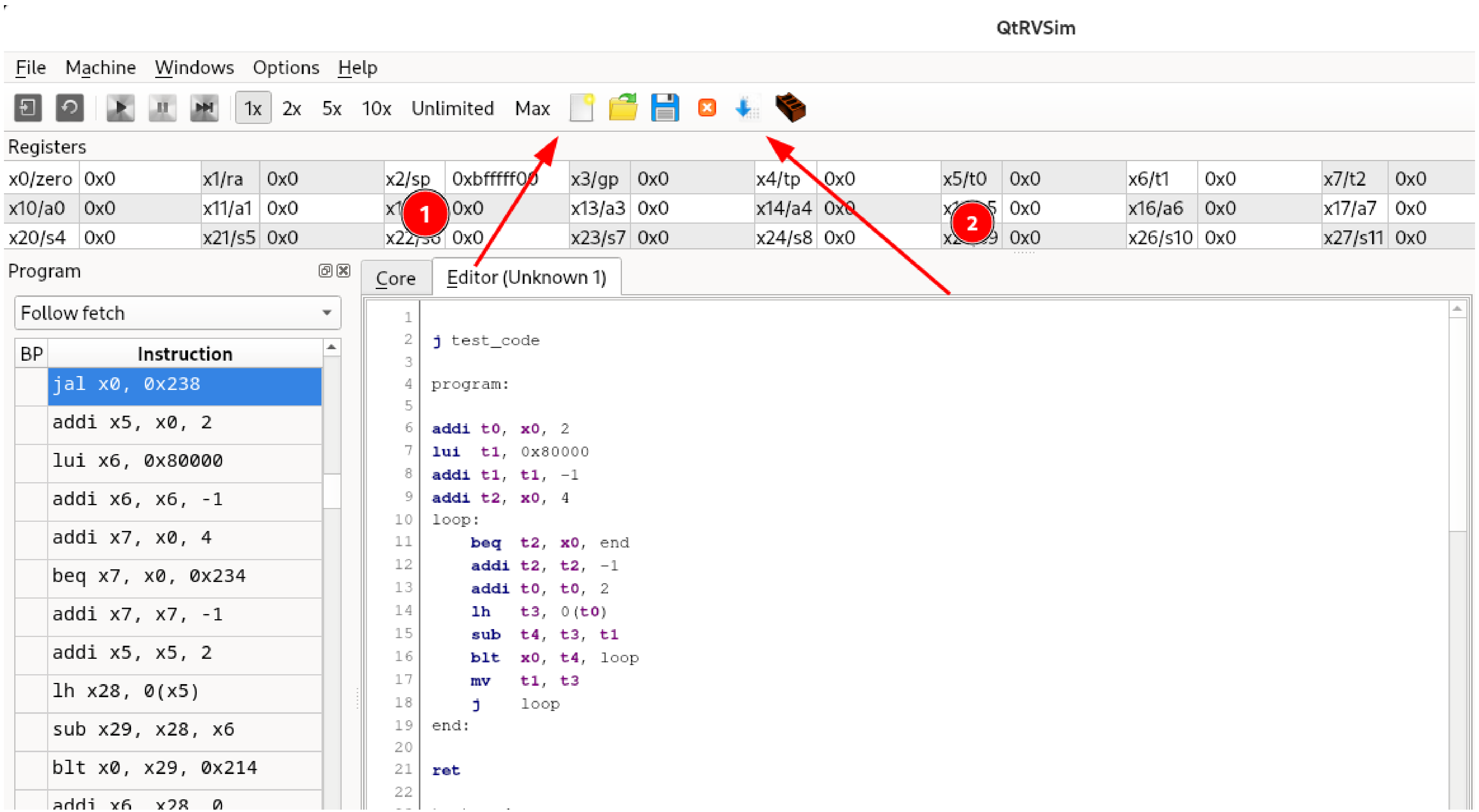
Direct

Address	+0	+4	+8
0x00000200	00000000	00000000	00000000
0x0000020c	00000000	00000000	00000000
0x00000218	00000000	00000000	00000000
0x00000224	00000000	00000000	00000000
0x00000230	00000000	00000000	00000000
0x0000023c	00000000	00000000	00000000
0x00000248	00000000	00000000	00000000
0x00000254	00000000	00000000	00000000
0x00000260	00000000	00000000	00000000
0x0000026c	00000000	00000000	00000000

0x00000200

Messages

QtRVSim – Loading a Program



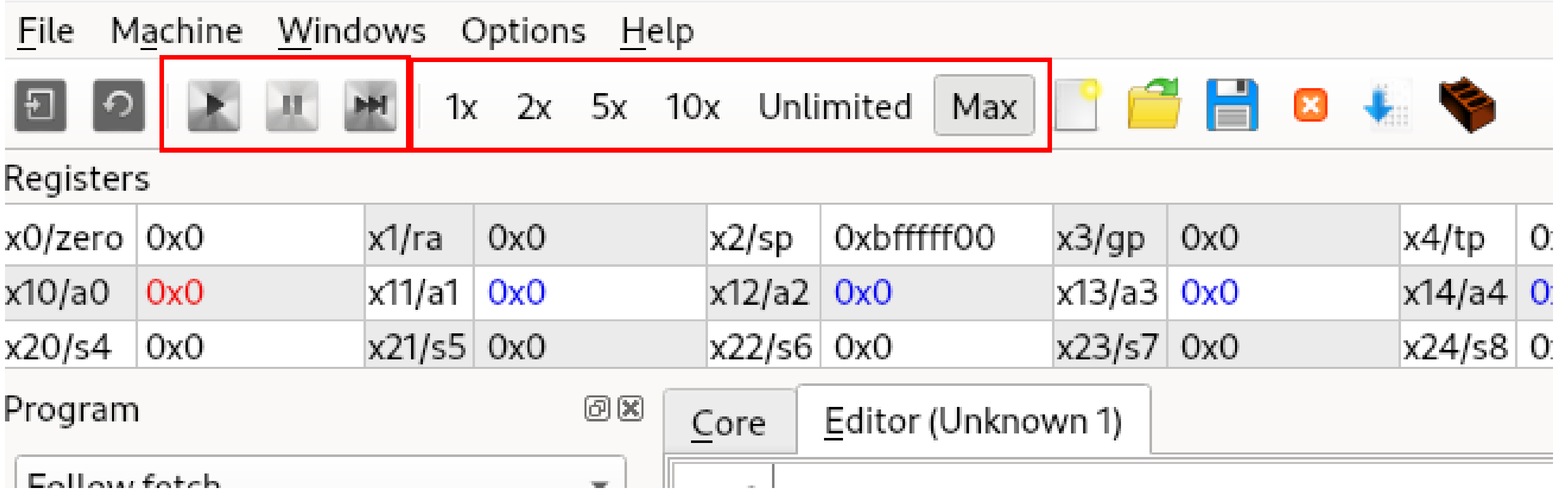
x0/zero	0x0	x1/ra	0x0	x2/sp	0xbfffff00	x3/gp	0x0	x4/tp	0x0	x5/t0	0x0	x6/t1	0x0	x7/t2	0x0
x10/a0	0x0	x11/a1	0x0	x12/a2	0x0	x13/a3	0x0	x14/a4	0x0	x15/a5	0x0	x16/a6	0x0	x17/a7	0x0
x20/s4	0x0	x21/s5	0x0	x22/s6	0x0	x23/s7	0x0	x24/s8	0x0	x25/s9	0x0	x26/s10	0x0	x27/s11	0x0

BP	Instruction
	jal x0, 0x238
	addi x5, x0, 2
	lui x6, 0x80000
	addi x6, x6, -1
	addi x7, x0, 4
	beq x7, x0, 0x234
	addi x7, x7, -1
	addi x5, x5, 2
	lh x28, 0(x5)
	sub x29, x28, x6
	blt x0, x29, 0x214
	addi x6, x28, 0

```

j test_code

program:

addi t0, x0, 2
lui  t1, 0x80000
addi t1, t1, -1
addi t2, x0, 4
loop:
    beq  t2, x0, end
    addi t2, t2, -1
    addi t0, t0, 2
    lh   t3, 0(t0)
    sub  t4, t3, t1
    blt  x0, t4, loop
    mv   t1, t3
    j    loop
end:

ret

```

QtRVSim – Working with the Simulator

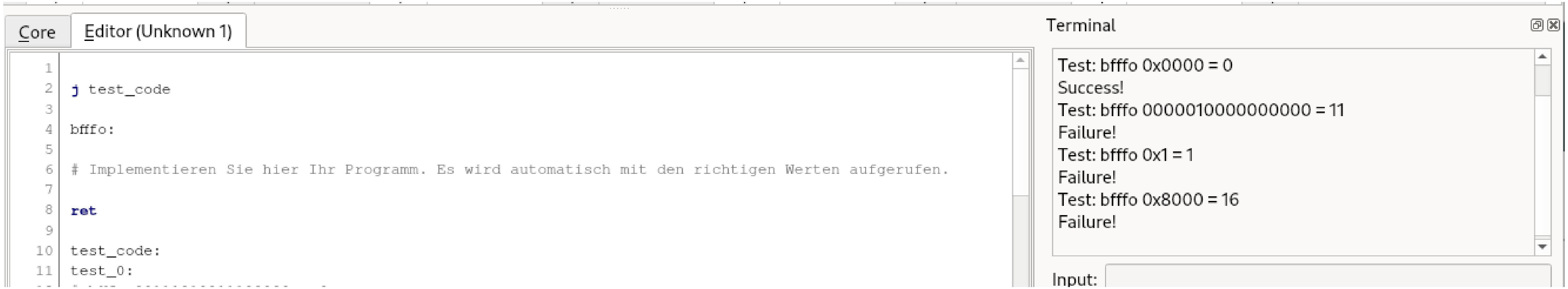
- Simulation speed can be selected
- Step-by-step execution supported
- Use the `ebreak` instruction to set breakpoints



Registers

x0/zero	0x0	x1/ra	0x0	x2/sp	0xbfffff00	x3/gp	0x0	x4/tp	0:
x10/a0	0x0	x11/a1	0x0	x12/a2	0x0	x13/a3	0x0	x14/a4	0:
x20/s4	0x0	x21/s5	0x0	x22/s6	0x0	x23/s7	0x0	x24/s8	0:

Exercise 2 – Assembly Code Tests



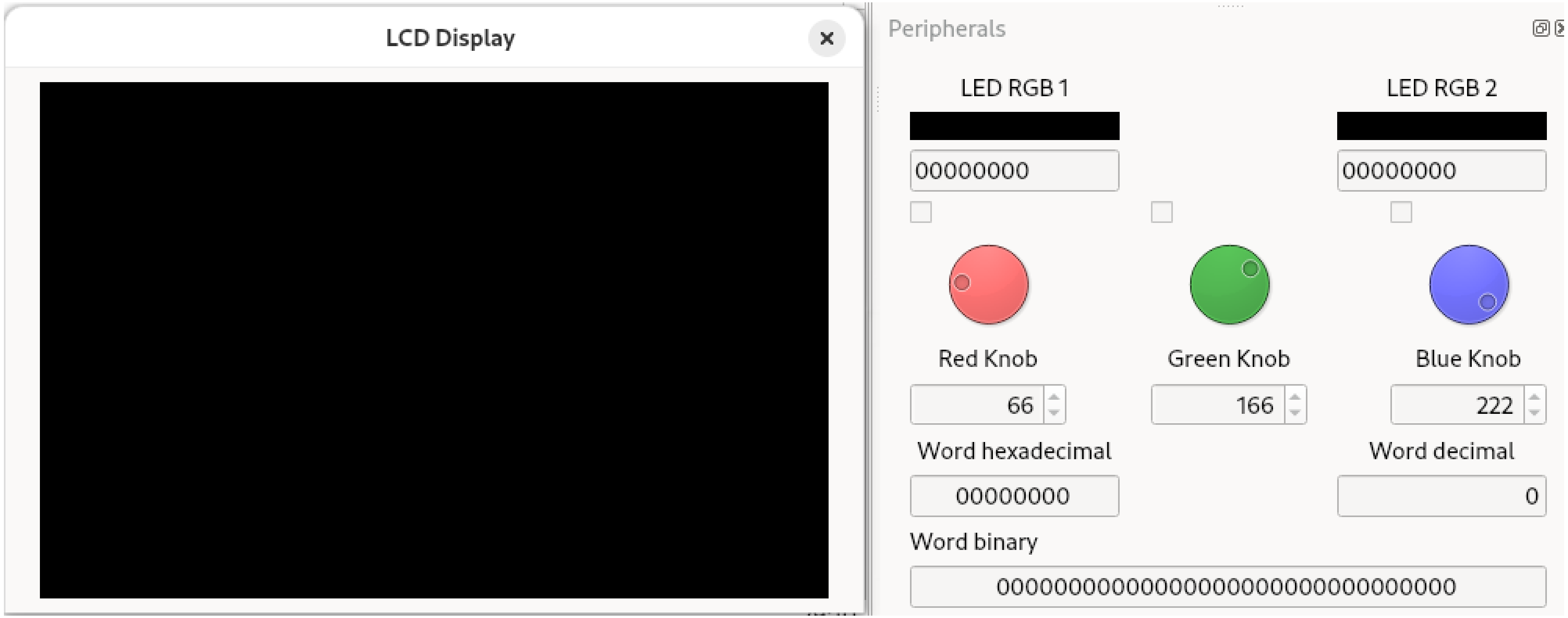
```

j test_code

bfffo:

# Implementieren Sie hier Ihr Programm. Es wird automatisch mit den richtigen Werten aufgerufen.

ret

test_code:
test_0:
```

```
Test: bfffo 0x0000 = 0
Success!
Test: bfffo 0000010000000000 = 11
Failure!
Test: bfffo 0x1 = 1
Failure!
Test: bfffo 0x8000 = 16
Failure!
```

QtRVSim – Advanced Usage

- System calls
  - https://github.com/cvut/qtrvsim?tab=readme-ov-file#system-calls-support
- Peripherals and LCD Screen
  - Simulated LEDs and Knobs
  - https://github.com/cvut/qtrvsim?tab=readme-ov-file#peripherals
- Microarchitecture visualization
- Pipeline support
- Cache support
- Branch predictor support

QtRVSim - Peripherals



LED RGB 1: 00000000

LED RGB 2: 00000000

□ □ □

Red Knob: 66

Green Knob: 166

Blue Knob: 222

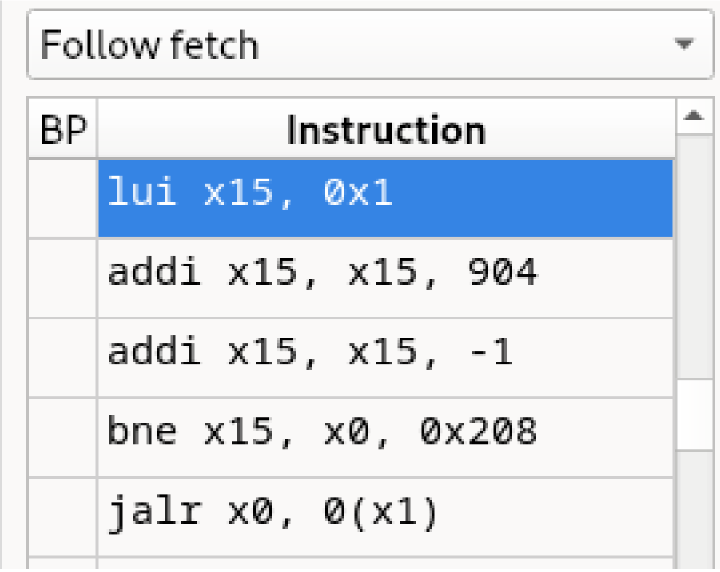
Word hexadecimal: 00000000

Word decimal: 0

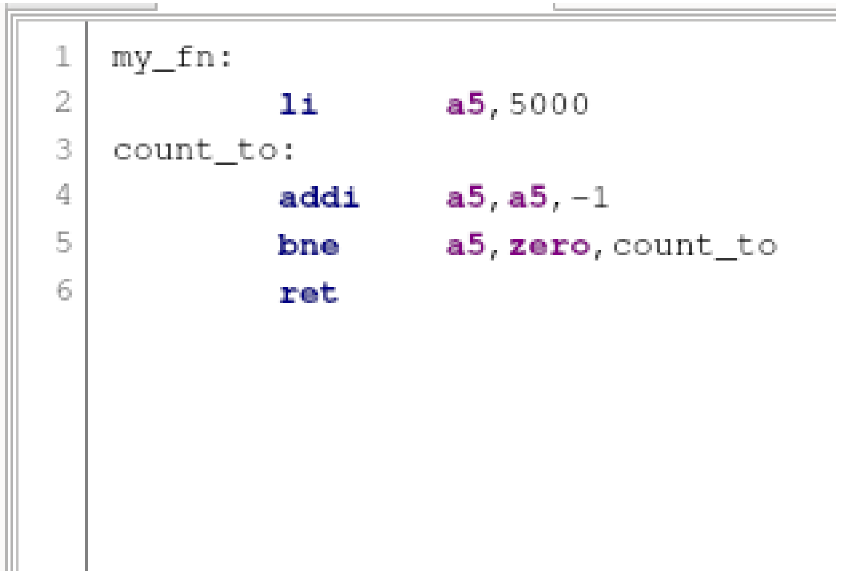
Word binary: 00000000000000000000000000000000

Assembler Support

- Generate machine code from assembly
- Pseudo instructions
- Use Labels for Branch and Jump instructions
- Provide information to the linker
- ...



BP	Instruction
	lui x15, 0x1
	addi x15, x15, 904
	addi x15, x15, -1
	bne x15, x0, 0x208
	jalr x0, 0(x1)



```
my_fn:
        li      a5,5000
count_to:
        addi    a5,a5,-1
        bne     a5,zero,count_to
        ret
```

Compiler Explorer

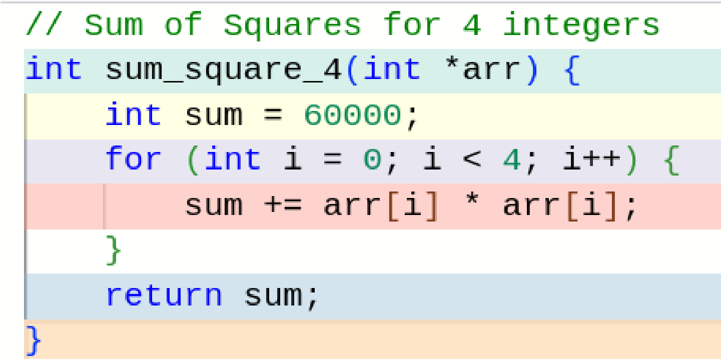
Compilers

- Crucial for most developer workflows
- Compilers can translate code from a source language to a target language
  - For example, C -> RISC-V assembly code
- Assemblers and linkers then generate an executable
- Many compilers out there
  - Popular for the C-family: gcc, clang (LLVM)
- Plethora of optimizations and options to control them
- Compiler Explorer allows for an easy comparison between compilers
  - https://godbolt.org/

A Simple C Program

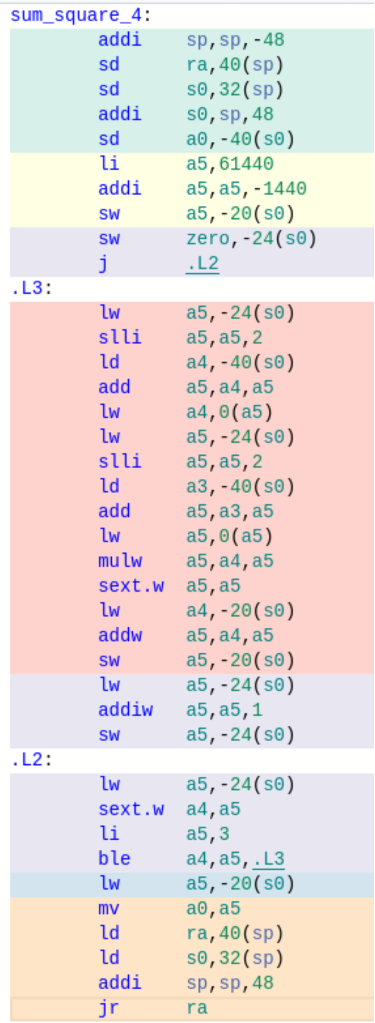
```c
// Sum of Squares for 4 integers
int sum_square_4(int *arr) {
    int sum = 0;
    for (int i = 0; i < 4; i++) {
        sum += arr[i] * arr[i];
    }
    return sum;
}
```

Translation with Different Optimization Levels



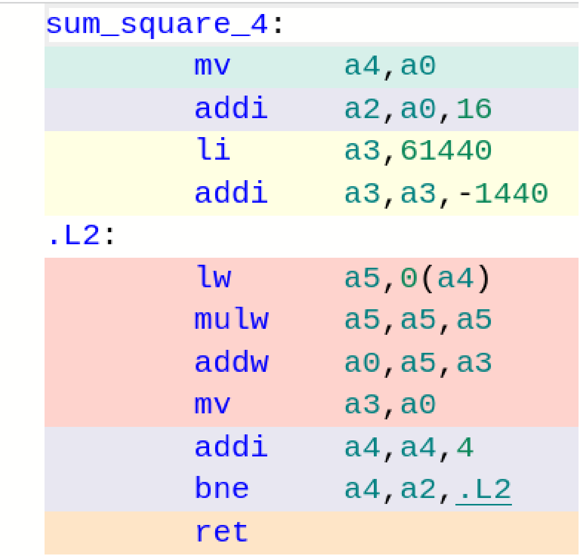
```
// Sum of Squares for 4 integers
int sum_square_4(int *arr) {
    int sum = 60000;
    for (int i = 0; i < 4; i++) {
        sum += arr[i] * arr[i];
    }
    return sum;
}
```

-O0



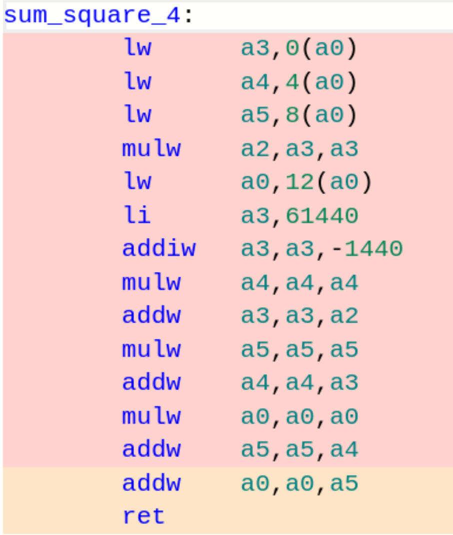
```
sum_square_4:
        addi    sp,sp,-48
        sd      ra,40(sp)
        sd      s0,32(sp)
        addi    s0,sp,48
        sd      a0,-40(s0)
        li      a5,61440
        addi    a5,a5,-1440
        sw      a5,-20(s0)
        sw      zero,-24(s0)
        j       .L2
.L3:
        lw      a5,-24(s0)
        slli    a5,a5,2
        ld      a4,-40(s0)
        add     a5,a4,a5
        lw      a4,0(a5)
        lw      a5,-24(s0)
        slli    a5,a5,2
        ld      a3,-40(s0)
        add     a5,a3,a5
        lw      a5,0(a5)
        mulw    a5,a4,a5
        sext.w  a5,a5
        lw      a4,-20(s0)
        addw    a5,a4,a5
        sw      a5,-20(s0)
        lw      a5,-24(s0)
        addiw   a5,a5,1
        sw      a5,-24(s0)
.L2:
        lw      a5,-24(s0)
        sext.w  a4,a5
        li      a5,3
        ble     a4,a5,.L3
        lw      a5,-20(s0)
        mv      a0,a5
        ld      ra,40(sp)
        ld      s0,32(sp)
        addi    sp,sp,48
        jr      ra
```

-O1



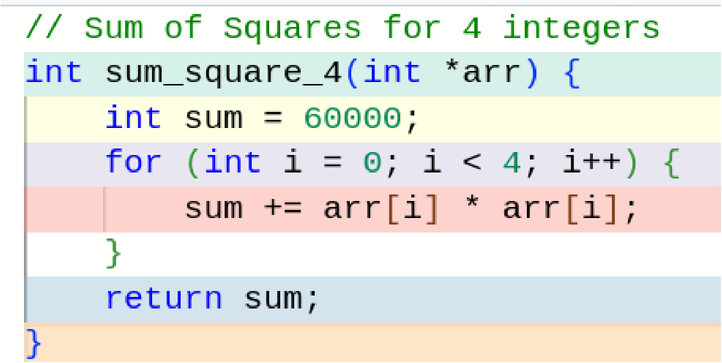
```
sum_square_4:
        mv      a4,a0
        addi    a2,a0,16
        li      a3,61440
        addi    a3,a3,-1440
.L2:
        lw      a5,0(a4)
        mulw    a5,a5,a5
        addw    a0,a5,a3
        mv      a3,a0
        addi    a4,a4,4
        bne     a4,a2,.L2
        ret
```

-O3



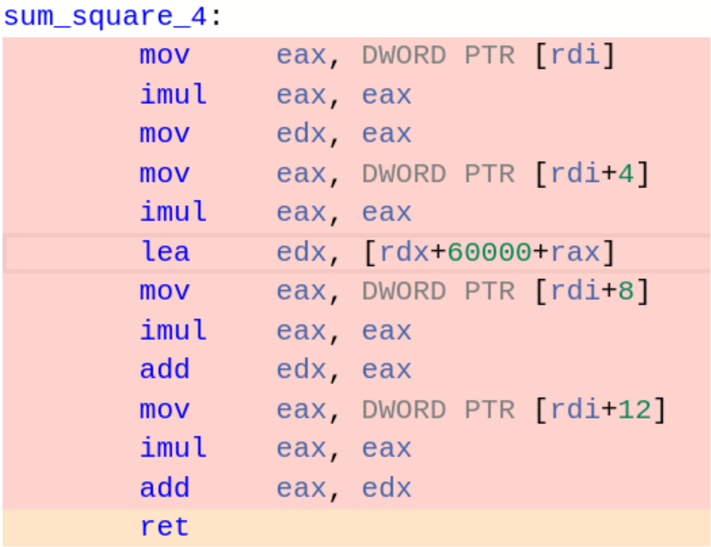
```
sum_square_4:
        lw      a3,0(a0)
        lw      a4,4(a0)
        lw      a5,8(a0)
        mulw    a2,a3,a3
        lw      a0,12(a0)
        li      a3,61440
        addiw   a3,a3,-1440
        mulw    a4,a4,a4
        addw    a3,a3,a2
        mulw    a5,a5,a5
        addw    a4,a4,a3
        mulw    a0,a0,a0
        addw    a5,a5,a4
        addw    a0,a0,a5
        ret
```

Translation to Different Architectures (x86-64)



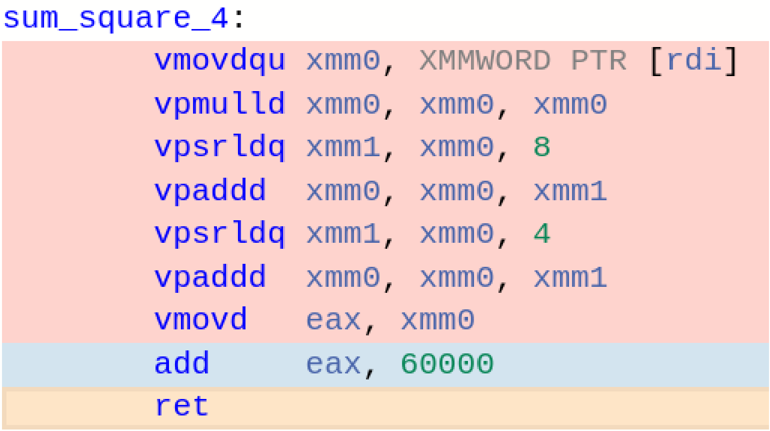
```c
// Sum of Squares for 4 integers
int sum_square_4(int *arr) {
    int sum = 60000;
    for (int i = 0; i < 4; i++) {
        sum += arr[i] * arr[i];
    }
    return sum;
}
```

-O3 -march=raptorlake -mno-sse



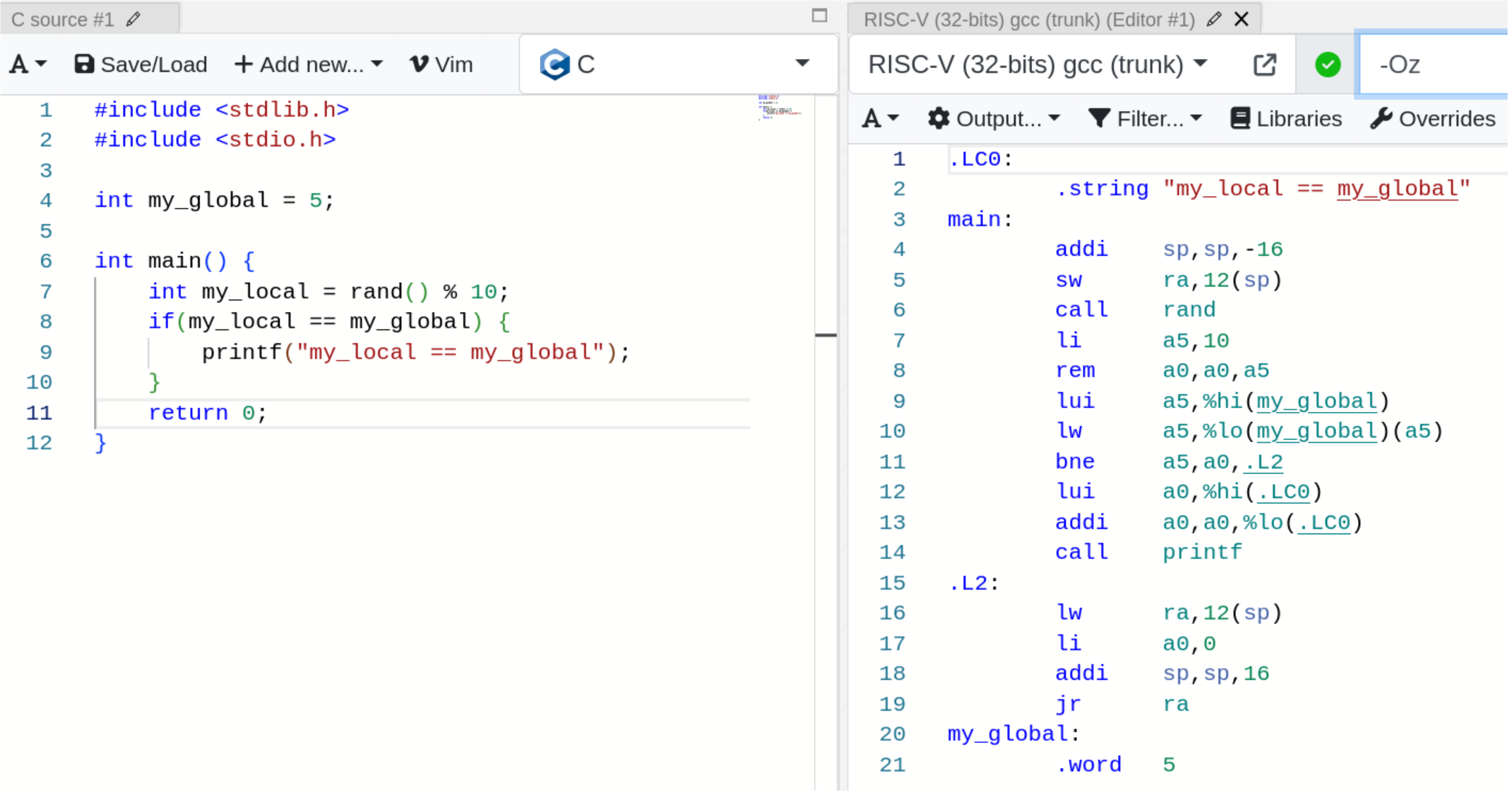
```asm
sum_square_4:
        mov     eax, DWORD PTR [rdi]
        imul    eax, eax
        mov     edx, eax
        mov     eax, DWORD PTR [rdi+4]
        imul    eax, eax
        lea     edx, [rdx+60000+rax]
        mov     eax, DWORD PTR [rdi+8]
        imul    eax, eax
        add     edx, eax
        mov     eax, DWORD PTR [rdi+12]
        imul    eax, eax
        add     eax, edx
        ret
```

-O3 -march=raptorlake



```asm
sum_square_4:
        vmovdqu xmm0, XMMWORD PTR [rdi]
        vpmulld xmm0, xmm0, xmm0
        vpsrldq xmm1, xmm0, 8
        vpaddd  xmm0, xmm0, xmm1
        vpsrldq xmm1, xmm0, 4
        vpaddd  xmm0, xmm0, xmm1
        vmovd   eax, xmm0
        add     eax, 60000
        ret
```

Globals and Text



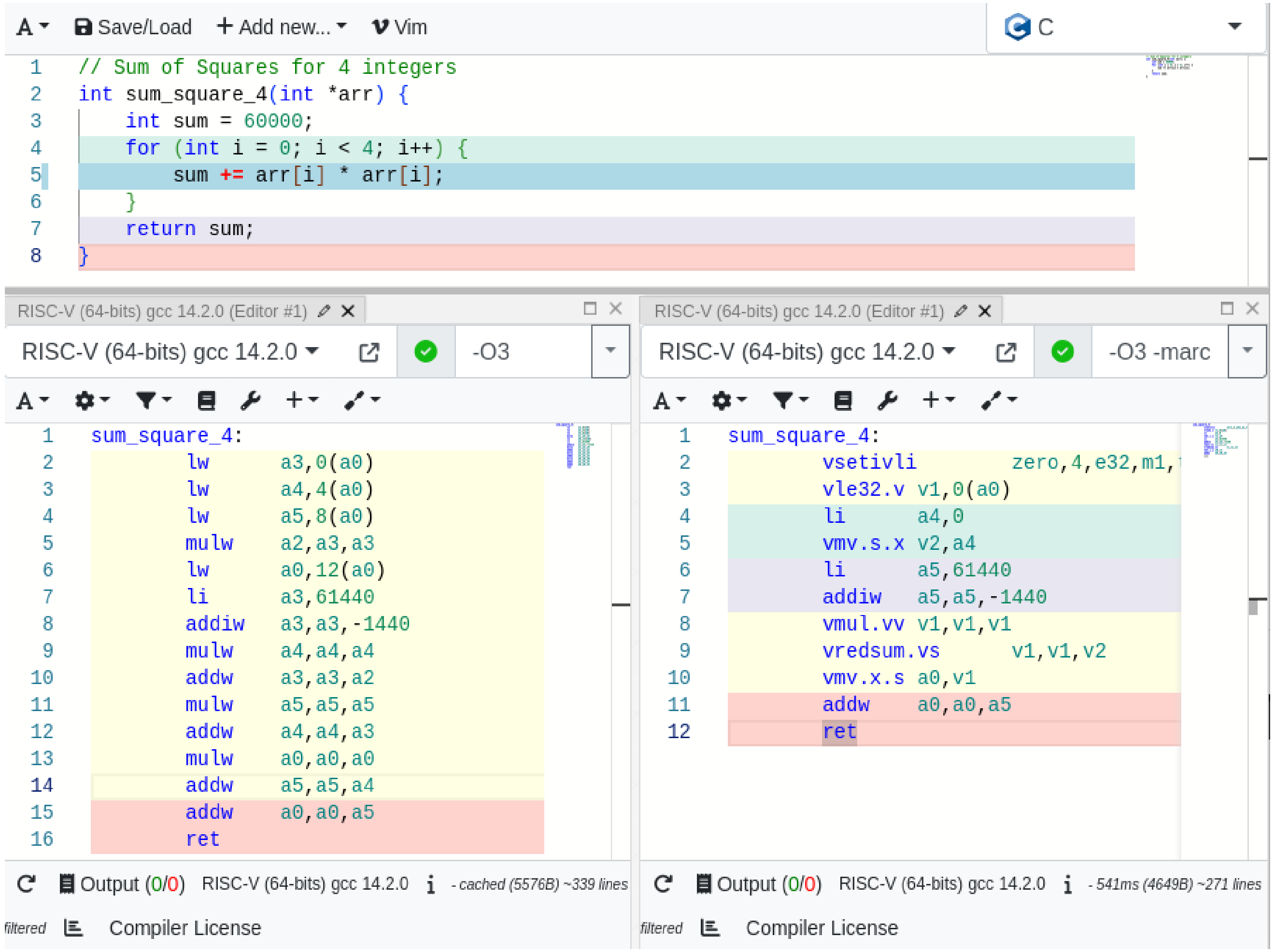
RISC-V (32-bits) gcc (trunk)

-Oz

```c
#include <stdlib.h>
#include <stdio.h>

int my_global = 5;

int main() {
    int my_local = rand() % 10;
    if(my_local == my_global) {
        printf("my_local == my_global");
    }
    return 0;
}
```

```
.LC0:
        .string "my_local == my_global"
main:
        addi    sp,sp,-16
        sw      ra,12(sp)
        call    rand
        li      a5,10
        rem     a0,a0,a5
        lui     a5,%hi(my_global)
        lw      a5,%lo(my_global)(a5)
        bne     a5,a0,.L2
        lui     a0,%hi(.LC0)
        addi    a0,a0,%lo(.LC0)
        call    printf
.L2:
        lw      ra,12(sp)
        li      a0,0
        addi    sp,sp,16
        jr      ra
my_global:
        .word   5
```

Compiler Explorer



```c
// Sum of Squares for 4 integers
int sum_square_4(int *arr) {
    int sum = 60000;
    for (int i = 0; i < 4; i++) {
        sum += arr[i] * arr[i];
    }
    return sum;
}
```

RISC-V (64-bits) gcc 14.2.0 (Editor #1)

RISC-V (64-bits) gcc 14.2.0 -O3

```
sum_square_4:
        lw      a3,0(a0)
        lw      a4,4(a0)
        lw      a5,8(a0)
        mulw    a2,a3,a3
        lw      a0,12(a0)
        li      a3,61440
        addiw   a3,a3,-1440
        mulw    a4,a4,a4
        addw    a3,a3,a2
        mulw    a5,a5,a5
        addw    a4,a4,a3
        mulw    a0,a0,a0
        addw    a5,a5,a4
        addw    a0,a0,a5
        ret
```

Output (0/0) RISC-V (64-bits) gcc 14.2.0 - cached (5576B) ~339 lines filtered

Compiler License

RISC-V (64-bits) gcc 14.2.0 (Editor #1)

RISC-V (64-bits) gcc 14.2.0 -O3 -marc

```
sum_square_4:
        vsetivli        zero,4,e32,m1,
        vle32.v v1,0(a0)
        li      a4,0
        vmv.s.x v2,a4
        li      a5,61440
        addiw   a5,a5,-1440
        vmul.vv v1,v1,v1
        vredsum.vs      v1,v1,v2
        vmv.x.s a0,v1
        addw    a0,a0,a5
        ret
```

Output (0/0) RISC-V (64-bits) gcc 14.2.0 - 541ms (4649B) ~271 lines filtered

Compiler License

Live Demo