

# 项目申请书

- 项目名称: SpiderMonkey 的 64 位 RISC-V 后端修缮工作
- 项目编号: 25f3e0428
- 项目主导师: 陈萱
- 申请人: 鲍溶
- 申请人邮箱: [webmaster@csmantle.top](mailto:webmaster@csmantle.top)

## Contents

1. 项目背景 .....	1
2. 实现方案 .....	2
2.1. 失败原因分类 .....	2
2.2. 指令实现 .....	2
2.3. 回退与性能测试 .....	3
2.4. Patchset 提交 .....	3
3. 时间计划 .....	3
3.1. 2025/07/01 – 2025/08/15: 开发第一阶段 .....	3
3.2. 2025/08/16 – 2025/09/30: 开发第二阶段 .....	3
4. 个人期望 .....	4

## 1. 项目背景

SpiderMonkey 是 Mozilla 多个项目中使用的 JS 和 WASM 引擎。直接解析、运行解释型语言开销较大，因此 SpiderMonkey 采用了层次化的优化引擎，发现所执行 JS 程序或 WASM 字节码中的热点部分，将其即时编译为宿主机原生指令，实现运行加速。

由于 SpiderMonkey 需要产生原生指令，因此需要为不同的 ISA 分别编写特定的指令发射后端。遗憾的是，针对 RISC-V 架构编写的发射后端在单元测试通过率上表现不佳。以 [commit eff23b2](#) 为例，目前有超过四分之一的 JIT 测试在自带的 RISC-V 模拟器上失败。



其中，大多数错误为由开发者标记为“尚未实现”的功能，如下图所示。

```
[15248] Hit MOZ_CRASH(IMPLEMENT ME) at D:/Workspace/gecko-dev/js/src/jit/MacroAssembler.cpp:7985
#01: js::jit::MacroAssembler::convertWasmAnyRefToValue (D:/Workspace/gecko-dev/js/src/jit/MacroAssembler.cpp:7679)
#02: GenerateJitEntry (D:/Workspace/gecko-dev/js/src/wasm/WasmStubs.cpp:1283)
#03: js::wasm::GenerateEntryStubs (D:/Workspace/gecko-dev/js/src/wasm/WasmStubs.cpp:3244)
#04: js::wasm::GenerateEntryStubs (D:/Workspace/gecko-dev/js/src/wasm/WasmStubs.cpp:3209)
#05: js::wasm::ModuleGenerator::finishTier (D:/Workspace/gecko-dev/js/src/wasm/WasmGenerator.cpp:1219)
#06: js::wasm::ModuleGenerator::finishModule (D:/Workspace/gecko-dev/js/src/wasm/WasmGenerator.cpp:1243)
#07: js::wasm::CompileBuffer (D:/Workspace/gecko-dev/js/src/wasm/WasmCompile.cpp:1044)
```

```
[27804] Hit MOZ_CRASH(needs to be implemented on this platform) at D:\Workspace\gecko-dev\js\src\jit\riscv64\MacroAssembler-riscv64.cpp:3138
#01: js::jit::MacroAssembler::sub32FromMemAndBranchIfNegativeWithPatch (D:\Workspace\gecko-dev\js\src\jit\riscv64\MacroAssembler-riscv64.cpp:3138)
#02: js::wasm::BaseCompiler::addHotnessCheck (D:\Workspace\gecko-dev\js\src\wasm\WasmBaselineCompile.cpp:1178)
#03: js::wasm::BaseCompiler::beginFunction (D:\Workspace\gecko-dev\js\src\wasm\WasmBaselineCompile.cpp:656)
#04: js::wasm::BaselineCompileFunctions (D:\Workspace\gecko-dev\js\src\wasm\WasmBaselineCompile.cpp:12495)
```

因此，本项目致力于提升 SpiderMonkey JIT 在 RISC-V 平台上的通过率，使得失败测试数量减少至 50 个以下。本项目中还可以同时学习 Mozilla 开源项目协作的相关知识。

## 2. 实现方案

### 2.1. 失败原因分类

需要根据单元测试失败的原因，将所有失败测试点分成几类：

- **未实现。** 类似上图所示，该功能被开发者明确标记为尚未实现。对于此种问题，在获悉正确行为后实现即可。
- **行为错误。** 虽然该测试点已实现，但行为不符合原 JS 或 WASM 代码语义。
- **其他。** 剩下无法归类的问题，可能是由宿主模拟器环境与 RISC-V 实机行为不一致，或是测试点为架构特定测试、不适用于 RISC-V 架构，等等。需要与导师与社区开发者沟通处理。

### 2.2. 指令实现

根据 RISC-V ISA 的特点，实现 IR 语义到机器指令的 lowering 过程。主要在 js/src/jit/riscv64/MacroAssembler-riscv64.cpp 中完成汇编器部分的实现。

MacroAssembler 已经较好地包装了相关的 API。参考已有代码，部分较为底层的功能可以直接发射寄存器到寄存器的机器指令，如以下已有代码：

```
void MacroAssembler::convertInt64ToDouble(Register64 src, FloatRegister dest)
{
    fcvtd_l(dest, src.scratchReg());
}
```

较为复杂的逻辑可以通过构造 MIR 的方式描述，如以下已有代码：

```
void MacroAssembler::ceilFloat32ToInt32(FloatRegister src, Register dest,
                                         Label* fail) {
    UseScratchRegisterScope temps(this);
    ScratchDoubleScope fscratch(*this);
    Label performCeil, done;
    // If x < -1 or x > 0 then perform ceil.
    loadConstantFloat32(0, fscratch);
    branchFloat(Assembler::DoubleGreaterThan, src, fscratch, &performCeil);
    loadConstantFloat32(-1.0, fscratch);
    branchFloat(Assembler::DoubleLessThanOrEqual, src, fscratch, &performCeil);

    Register scratch = temps.Acquire();
    // If binary value is not zero, the input was not 0, so we bail.
    {
        fmv_x_w(scratch, src);
        branch32(Assembler::NotEqual, scratch, zero, fail);
    }
    bind(&performCeil);
}
```

```
Ceil_w_s(dest, src, scratch);
ma_b(scratch, Imm32(1), fail, NotEqual);
bind(&done);
}
```

这种描述方式与我熟悉的逆向工程中指令提升（**lifting**）过程有相通之处，SpiderMonkey 提供的接口也与大多数主流反编译框架提供的 IR/IL 类似。区别在于，**lifting** 过程将机器码转化为 IR，而本项目中需要将 IR 转化为机器码。

实现指令时需要考虑 RISC-V 的相关指令集扩展的存在，如 **Atomic**、**Vector** 和大量的 **Zxx** 扩展。使用这些指令在部分微架构上能够显著提升性能，但是否通用则需要与 Mozilla 贡献者进一步沟通。

### 2.3. 回退与性能测试

为保证新的更改不 **break** 任何已有实现，需要时刻关注是否出现新的失败单元测试。新的功能可能会需要额外的单元测试来覆盖更多的 **edge cases**，需要和贡献者详细沟通。

目前，`./mach jit-test` 运行结果中只包含了正确性结果，即是否通过测试，而没有包含测试性能效率等的指标。也需要和贡献者沟通关于 **JIT** 的实际运行效率问题。

### 2.4. Patchset 提交

代码实现和文档均完成后，需要在导师指导下向 **Mozilla** 提交补丁。该过程可以使用 mach 半自动化完成，但是具体的流程仍然需要确认，包括是否需要在 **Bugzilla** 上提交 **bug**，需要得到哪些社区贡献者的签发与 **review**，以及需要在哪些 **CI** 流程中通过，等等。我认为，我的开源项目贡献经验能够让我顺利完成该部分的流程，当然，若有导师的指导，这部分流程定能事半功倍。

## 3. 时间计划

项目期望完成时间：200 小时

### 3.1. 2025/07/01 – 2025/08/15：开发第一阶段

- 与 Mozilla 开发者和导师沟通，学习 SpiderMonkey JIT 的工作机制；
- 标记所有诸如 `MOZ_CRASH` 之类 **Todo** 部分，查阅手册并与 Mozilla 开发者沟通，了解期望行为；
- 重复迭代开发，实现 **Todo** 后运行测试，检验行为是否符合语义；
- 及时沟通新发现的 **edge cases**，并适时添加相应的新测试。

### 3.2. 2025/08/16 – 2025/09/30：开发第二阶段

- 对第一阶段完成的内容进行更详细的测试；
- 按照 Mozilla 贡献 workflow，与导师沟通，准备提交 **patch**；
- 对第一阶段的完成内容进行总结，并输出相关文档内容；
- 思考改进点与尚不完善之处。

## 4. 个人期望

我希望能够借助该项目的机会，锻炼在大型项目中与其他开发者协作的能力，进一步提升自身跨平台开发能力与对 **RISC-V ISA** 的了解，同时更深入掌握 **JIT native code generation** 和编译原理相关专业知识，为进一步学习提升打好基础。