第 3 章　RISC-V 虚拟机——Ripes

Ripes 是一个教学性的 RISC-V 虚拟机[①]。它专为 RISC-V 指令集架构提供了可视化计算机架构视图，还提供了 RISC-V 汇编代码编辑、汇编和运行的功能。本章将介绍如何使用 Ripes 虚拟机，编写汇编并进行模拟运行，逐渐加深对 RISC-V 指令的掌握，并通过数据通路的可视化视图对处理器结构建立直观的认识。

3.1　Ripes 简介

3.1.1　Ripes 的下载

1. 网页版

提供了一个 Web 版本的 Ripes 程序，无须安装直接进行使用——使用浏览器访问即可使用其仿真功能。

2. 可执行程序

Ripes 是一个开源的开发项目，其项目代码位于 GitHub，可以在 Github 上的 release page 页面下载发行的软件包，针对 Linux 提供了 AppImage 运行镜像，同时也提供了 macOS 和 Windows 系统的下载文件，其中 Windows 版本可能更稳定一点。读者可根据实际的系统情况进行下载（无须安装）。

3.1.2　功能简介

Ripes 主要的功能区展示在左侧的导航栏中，分别是代码编辑器界面、数据通路界面、内存界面、缓存界面、IO 设备界面。

1. 代码编辑界面

在代码编辑器界面中，会显示左右两块代码段区域。在左侧，用户可以编写使用 RISC-V RV32(I/M/C)指令集的汇编程序。每当在此汇编程序中执行任何编辑时，假设 Ripes 并未发现语法错误，则这些汇编代码将自动被汇编并插入模拟器中，如图 3-1 所示。

2. 数据通路界面

数据通路界面是 Ripes 显示当前所选处理器以及与执行相关状态信息的窗口，如图 3-2 所示。

---

① Spike 是另一个常用的 RISC-V 模拟器。

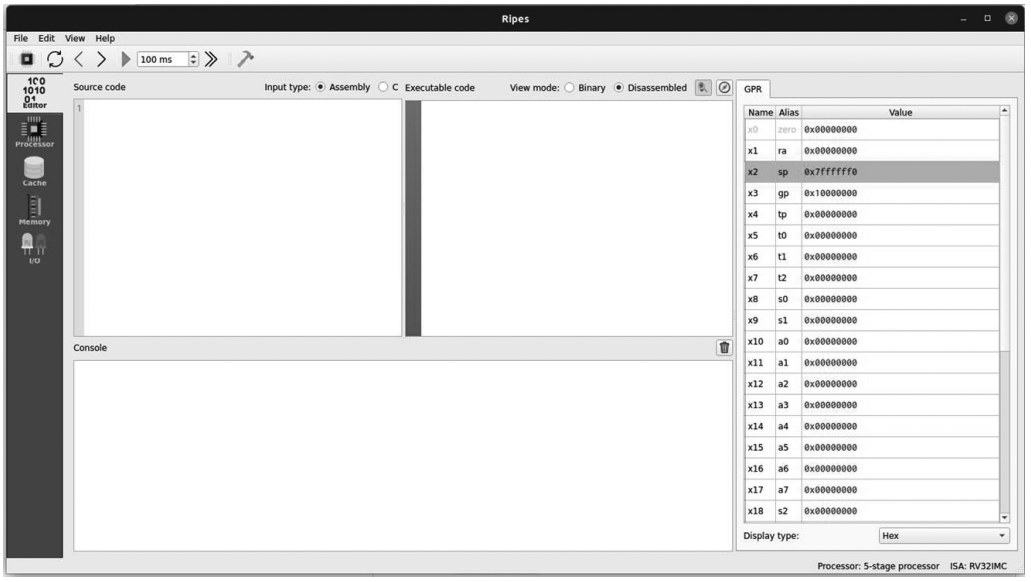


图 3-1　Ripes 的代码查看窗口

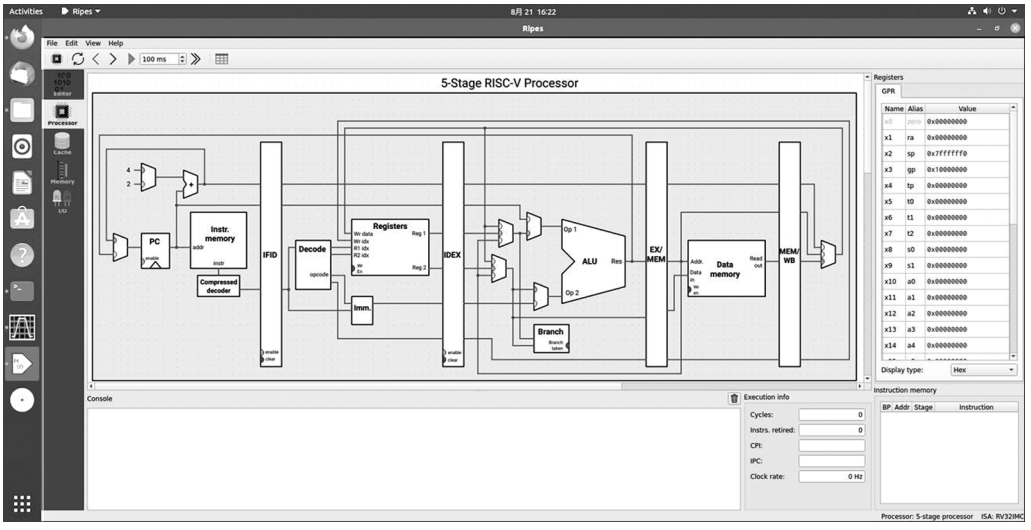


图 3-2　Ripes 处理器数据通路窗口

3. 内存界面

内存界面提供了处理器整个可寻址地址空间的视图，可以查看不同地址内的数据，如图 3-3 所示。

4. 缓存界面

缓存(Cache)界面模拟了 L1D(数据)和 L1I(指令)缓存，其中可以配置每种缓存类型的布局和行为。使用者能够分析程序的缓存性能，以了解不同的缓存设计如何与程序表现出的内存访问模式交互，如图 3-4 所示。

5. IO 设备界面

IO 设备界面提供了三种 IO 设备，可以在虚拟机中快速实现一个小型的嵌入式系统，如

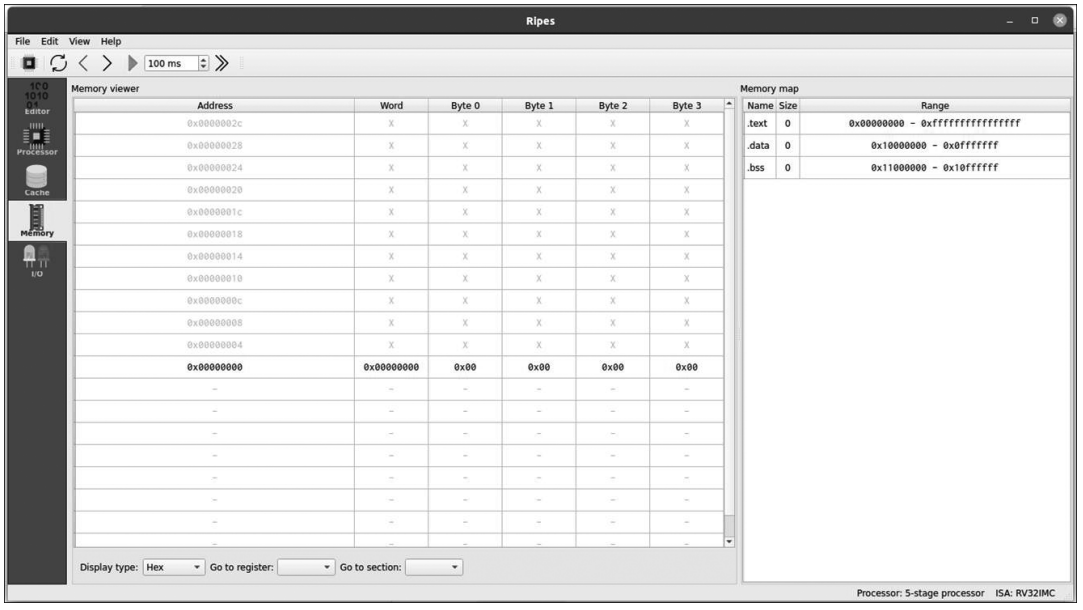


图 3-3　Ripes 内存查看窗口

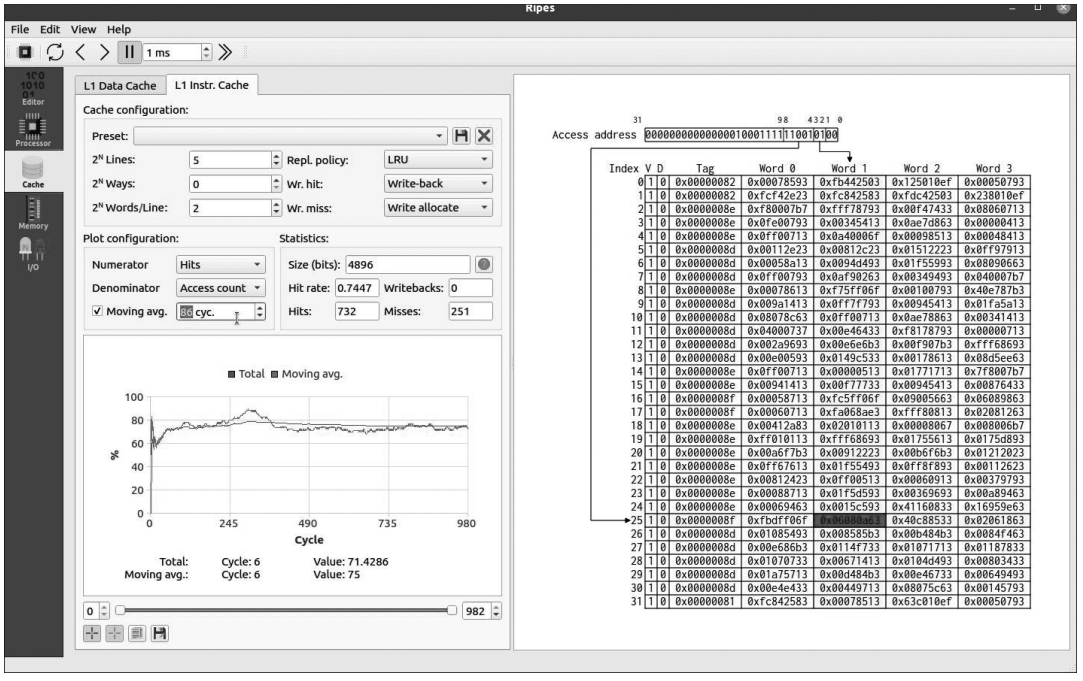


图 3-4　Ripes 的 Cache 查看窗口

图 3-5 所示。

关于 Ripes 图形化界面的详细说明信息，建议参考原始仓库中的官方文档。本书在此就不展开讲解了。

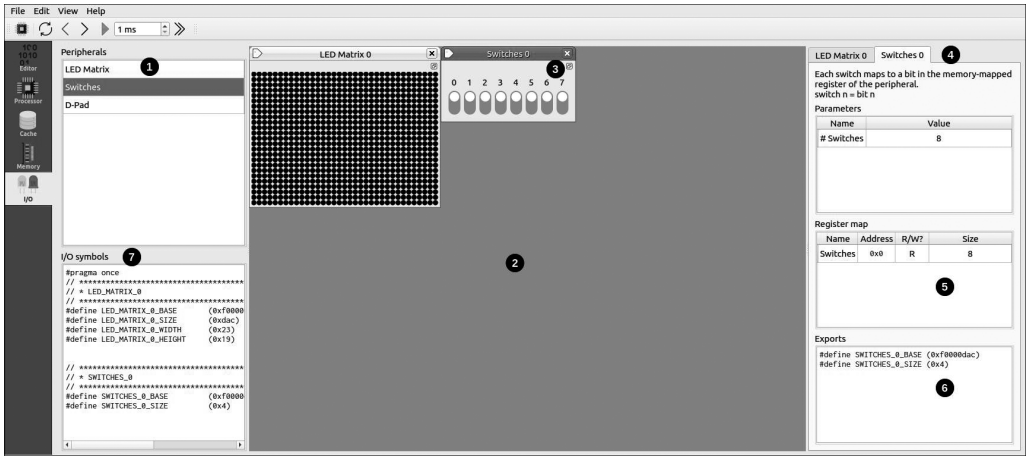


图 3-5　Ripes 的 IO 设备查看窗口

3.2　Ripes 的使用

运行代码前，下面给出两个 RISC-V 汇编程序，以供读者在 Ripes 上加载运行，也作为 RISC-V 汇编程序的学习样例代码。

3.2.1　示例代码

注意下面的代码中的.data 和.text 等，分别是数据节和代码节的开始；.global main 是 main 函数起始，global 表示 main 是一个全局符号；main:、out_loop:、Inner_loop: 则是行标号；.word 则用来为一个数据分配一个字(4 字节)，后面有多个数据则用逗号隔开；# 后面是注释；更多其他汇编格式方面的知识在遇到时再加以解释。

(1) 矩阵加法。

下面是对一段 2×2 矩阵 A、B 进行相加并将结果保存到 C 矩阵中的代码。

```
1.   .data
2.   A:   .word 1, 2, 3, 4        #矩阵 A
3.   B:   .word 5, 6, 7, 8        #矩阵 B
4.   C:   .word 0, 0, 0, 0        #矩阵 C 用于存储结果,初始化为 0
5.
6.   .text
7.   .globl main
8.   main:
9.   la t0, A                     #将矩阵 A 当前元素的地址加载到 t0
10.  la t1, B                     #将矩阵 B 当前元素的地址加载到 t1
11.  la t2, C                     #将矩阵 C 当前元素的地址加载到 t2
12.
13.  li t3, 4                     #待计算的矩阵元素数量,4->3->2->1->0
14.
15.  loop:
16.  beqz t3, end                 #如果 t3 为 0,跳转到 end
17.  lw t4, 0(t0)                 #从矩阵 A 加载当前元素到 t4
```

```
lw t5, 0(t1)                    #从矩阵 B 加载当前元素到 t5
add t6, t4, t5                  #计算 t4 + t5 并存储在 t6
sw t6, 0(t2)                    #将结果存储到矩阵 C

addi t0, t0, 4                  #将 t0 增加 4,指向下一个 A 元素
addi t1, t1, 4                  #将 t1 增加 4,指向下一个 B 元素
addi t2, t2, 4                  #将 t2 增加 4,指向下一个 C 元素
addi t3, t3, -1                 #将 t3 减 1
j loop                          #跳转回 loop 继续循环

end:
li a7,10                        #设置系统调用号为退出
ecall                           #ripes 系统调用,见 Help -> System Calls
```

(2) 数组求和。

下面给出另一段示例代码。它是遍历一个具有 5 个元素的数组,逐个元素进行求和的汇编代码。

```
.data
#定义一个数组和一个用于存储结果的变量
array: .word 1, 2, 3, 4, 5      #数组元素
size:  .word 5                  #数组大小
sum:   .word 0                  #存储结果的变量

.text
.globl main
main:
    #初始化
    la t0, array                #加载数组的基地址到 t0
    lw t1, size                 #加载数组大小到 t1
    li t2, 0                    #初始化求和结果 t2 = 0
    li t3, 0                    #初始化索引 i = 0

sum_loop:
    #检查是否达到数组大小
    bge t3, t1, end_sum         #如果 i >= size, 跳到结束

    #加载当前元素并累加
    lw t4, 0(t0)                #加载 array[i]到 t4
    add t2, t2, t4              #sum += array[i]

    #更新索引
    addi t3, t3, 1              #i++
    addi t0, t0, 4              #更新数组地址(每个元素为 4 字节)
    j sum_loop                  #返回循环开始

end_sum:
    #将结果存储到 sum 变量中
    la t0, sum                  #加载 sum 变量的地址
    sw t2, 0(t0)                #存储结果
```

3.2.2 模拟运行

下面以汇编程序片段的执行来介绍 Ripes 的使用步骤。读者阅读完后，可以运行 3.2.1 节的实例代码，并跟踪查看指令在数据通路上中流动、内存和寄存器数值的变化等过程。

(1) 汇编程序片段的编辑或加载。

请将下面这段 demo 程序键入(或复制→粘贴)到编辑器界面的 Source code 区域，并在界面上选择输入类型(Input type)为汇编(Assembly)：

```
1.          add     t0,zero,zero
2.   loop:  addi    t0,t0,0x05
3.          sw      t0, 0x20(zero)
4.          jal     zero, skip
5.          sub     t0,t0,t0
6.   skip:  jal     zero, loop
```

当然，读者也可以在本地文件中编写这段程序，然后通过 File→Open 菜单将该文件加载到 Ripes 中。

当源代码进入汇编代码编辑区域之后，右侧的展示区域会显示相应的机器码及其反汇编代码，如图 3-6 所示。

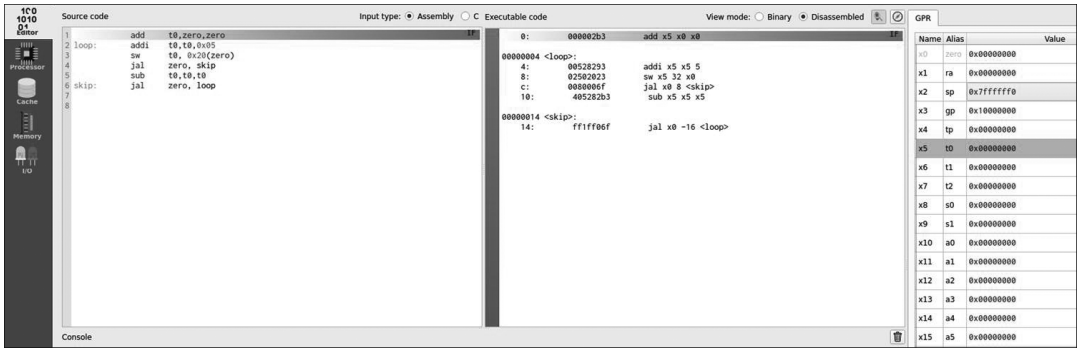


图 3-6 Ripes 代码窗口查看汇编代码的机器码及其反汇编代码

也可以选择 Binary 来查看汇编代码的机器码，上面示例代码的机器码及其二进制形式如下：

```
    0:        000002b3        00000000000000000000001010110011

00000004 <loop>:
    4:        00528293        00000000010100101000001010010011
    8:        02502023        00000010010100000010000000100011
    c:        0080006f        00000000100000000000000001101111
    10:       405282b3        01000000010100101000001010110011

00000014 <skip>:
    14:        ff1ff06f        11111111000111111111000001101111
```

读者请回顾 2.2.1 节给出的指令格式，对照此处的各条指令的相应字段，尝试从机器码进行反汇编获得其汇编指令。

（2）运行 C 语言程序。

除了直接运行 RISC-V 汇编程序，Ripes 还支持使用 C 语言代码编译运行，但需要手动指定 RISC-V 的 GCC 编译器，因此需要下载 RISC-V 编译工具链。和 Ripes 版本匹配的编译工具链可以在 SiFive 的 Github 页面下载。

完成下载之后，单击菜单栏的 Edit 菜单，将下载的 GCC 编译工具输入 Compiler 的路径中，出现绿色即说明正确识别，如图 3-7 所示。

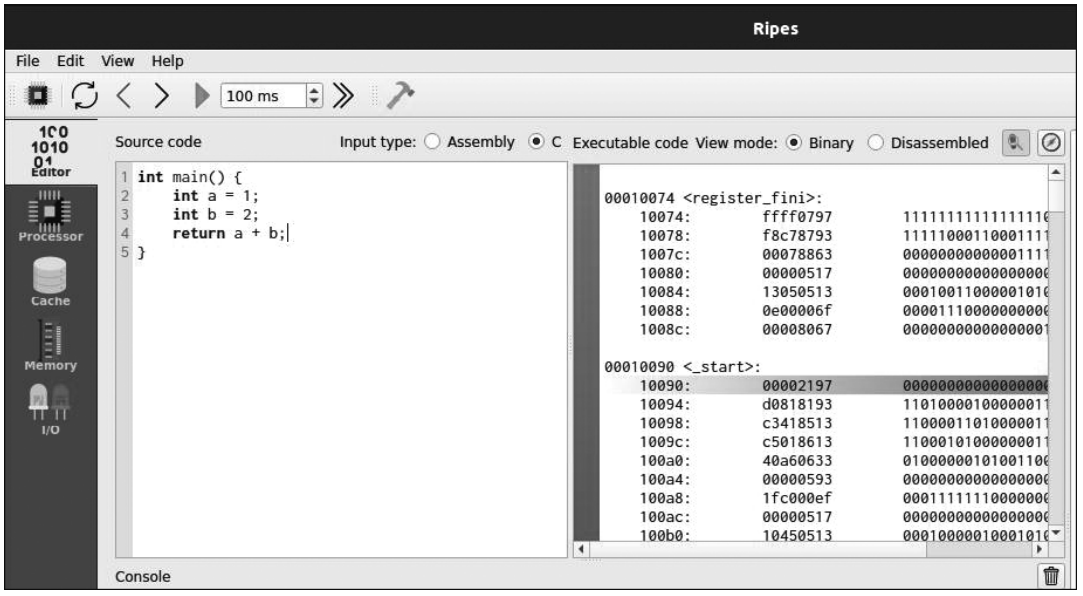


图 3-7　Ripes 代码窗口 C 代码编译成汇编

（3）转到内存界面观察内存变化情况。

需要注意的是，内存界面中具有可执行文件的概念，而不仅是代码片段，因此需要了解一定的程序链接相关的段、节的概念。如果读者还没有掌握或了解这些概念，可以参阅清华大学出版社出版的《Linux GNU C 程序观察》中链接相关的章节。

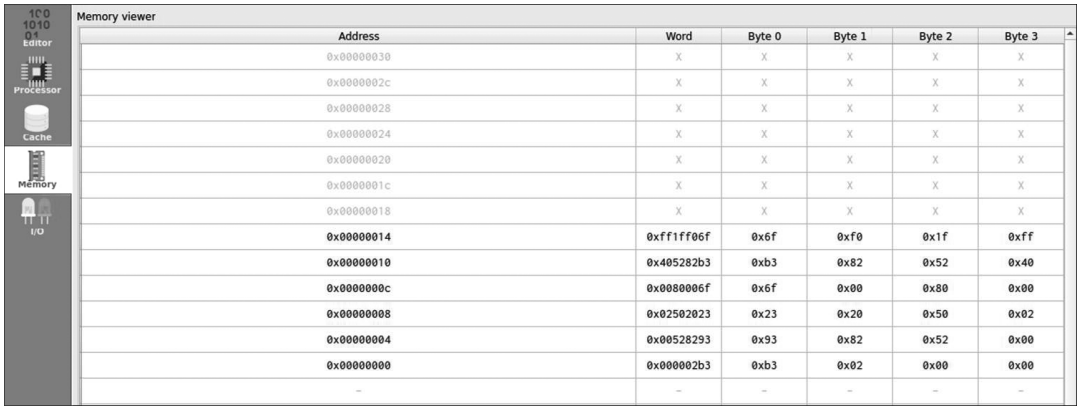
Ripes 内存界面提供了以下功能。

- 滚动显示内存区域。
- Go to register 会将内存界面滚动到该寄存器当前存储的内存地址，并在内存界面中显示该地址对应的内容。
- Go to section 会把内存界面滚动到内存中给定段值的地址（即指令内存.text 段、静态数据.data 段等）。

每当处理器复位时，程序执行期间写入的所有内存都将复位到其初始状态。

在内存界面可以看到加载的程序指令已经被装载到内存的代码段区域，当执行访存的指令对内存进行写入时，相应的内存区域数值也会发生更改，如图 3-8 所示。

在 memory map 区域内则显示程序各段/节的地址范围，如图 3-9 所示。



Address	Word	Byte 0	Byte 1	Byte 2	Byte 3
0x00000030	x	x	x	x	x
0x0000002c	x	x	x	x	x
0x00000028	x	x	x	x	x
0x00000024	x	x	x	x	x
0x00000020	x	x	x	x	x
0x0000001c	x	x	x	x	x
0x00000018	x	x	x	x	x
0x00000014	0xff1ff06f	0x6f	0xf0	0x1f	0xff
0x00000010	0x405282b3	0xb3	0x82	0x52	0x40
0x0000000c	0x0080006f	0x6f	0x00	0x80	0x00
0x00000008	0x02502023	0x23	0x20	0x50	0x02
0x00000004	0x00528293	0x93	0x82	0x52	0x00
0x00000000	0x000002b3	0xb3	0x02	0x00	0x00
-	-	-	-	-	-

图 3-8　Ripes 内存窗口查看指令段

Memory map

Name	Size	Range
.text	168	0x00000000 - 0x000000a7
.data	29	0x10000000 - 0x1000001c
.bss	0	0x11000000 - 0x10ffffff

图 3-9　Ripes 中查看程序的段/节

3.2.3　IO 外设

Ripes 提供了三种 IO 外设：LED 矩阵，switch 开关和十字方向键。有了 IO 外设的存在，就可以使用 Ripes 模拟 RISC-V 程序对外部设备的操作。

在 Ripes 中，可以通过加载文件的方式加载 Ripes 提供的示例代码 leds.s，这是一个与 LED 点阵设备交互的 RISC-V 程序，如图 3-10 所示。

在运行该程序之前，还需要单击“LED Matrix”选项来实例化 LED 外设。该程序的功能是在 LED 外设中不停使用渐变色来一行行描绘 LED 中的像素点，如图 3-11 所示。

3.2.4　数据通路观察

处理器设计中很重要的一个工作是处理器核的设计，其中数据通路的设计和处理器性能密切相关。在后面设计 RISC-V 的数据通路之前，可以借助 Ripes 建立数据通路的一个粗略认识，了解数据通路的单周期和流水线形态，以及了解冒险的现象。

1. 处理器架构设置

进入 select processor 界面可以选择不同的 RISC-V 处理器架构，如图 3-12 所示。

在选择 RISC-V 处理器架构的界面上，Ripes 提供了 6 种架构进行模拟。

(1) 单周期的处理器。

(2) 五级流水的处理器。

(3) 没有直接通路或冒险检测的五级流水处理器。

(4) 带有直接通路、没有冒险检测的五级流水处理器。

(5) 没有直接通路，但是有冒险检测的五级流水处理器。

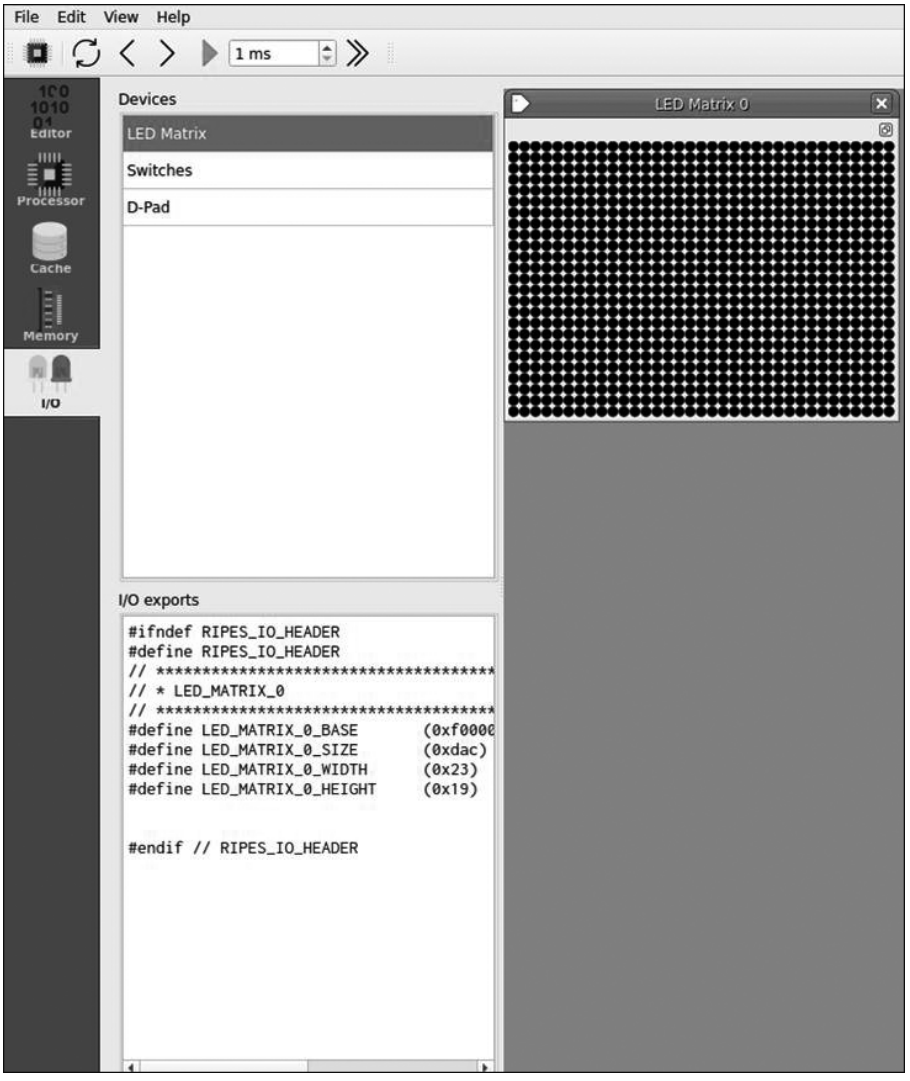


图 3-10　Ripes 中创建所需的外设

（6）六级双发射顺序处理器。

在选择处理器架构的界面还可以选择是否可以包含某些标准扩展指令（M，A），同时能够设置某些关键寄存器的初始值，比如 sp（栈指针）和 gp（全局指针）。

在数据通路界面中，除了处理器架构示意图之外，还包含以下视图。

（1）寄存器组：处理器所有寄存器的列表。可以通过单击给定寄存器的值来编辑寄存器值。

（2）指令存储器：模拟器中加载的当前程序的视图。

（3）统计：基于周期计数和当前退役指令数的各种统计。统计信息包括当前已执行的周期数，已执行完的指令数，CPI（每条指令需要的周期数），IPC（每周期执行的指令数）以及时钟频率。

（4）输出：通过 ecall 打印函数的任何输出都将显示在此处。

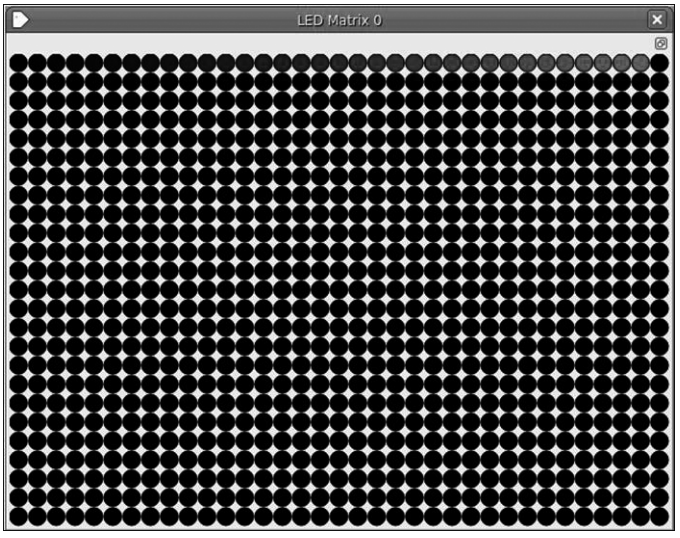


图 3-11　Ripes 的 LED 外设点亮示意图

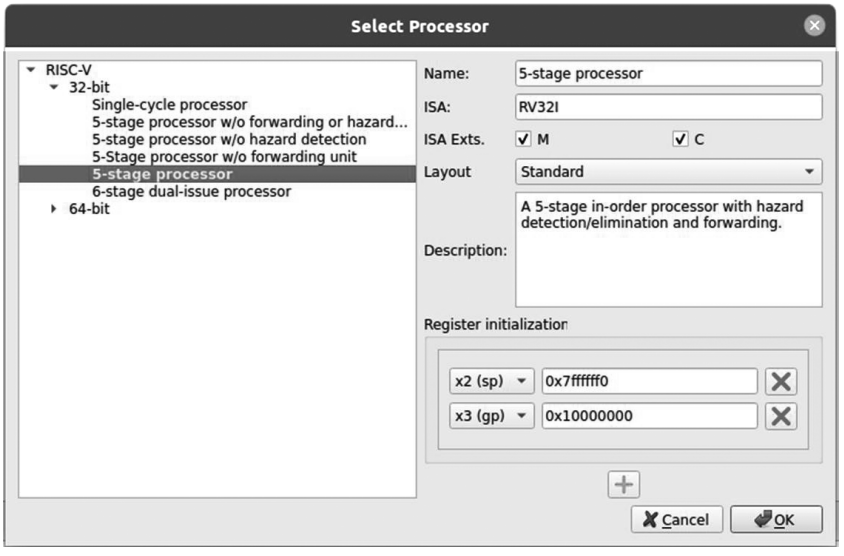


图 3-12　Ripes 中的 RISC-V 处理器架构选择

2. 流水运行

以 3.2.2 节载入的 demo 汇编程序为例，演示一下 Ripes 处理器运行程序的过程，理解 RISC-V 处理器的运行。

需要说明的时，在数据通路界面中，单击菜单栏中的大于号进行单步执行，单击菜单栏的绿色右箭头可以让处理器持续运行。

以默认的五级流水的处理器为例，当按下右箭头的单步执行之后，第一条指令 add 被加载到处理器的取指阶段，如图 3-13 所示。

继续使用单步执行按钮，可以看到当一条指令从一个阶段移动到下一个阶段，后面的指令同时会加载到处理器中，如图 3-14 所示。

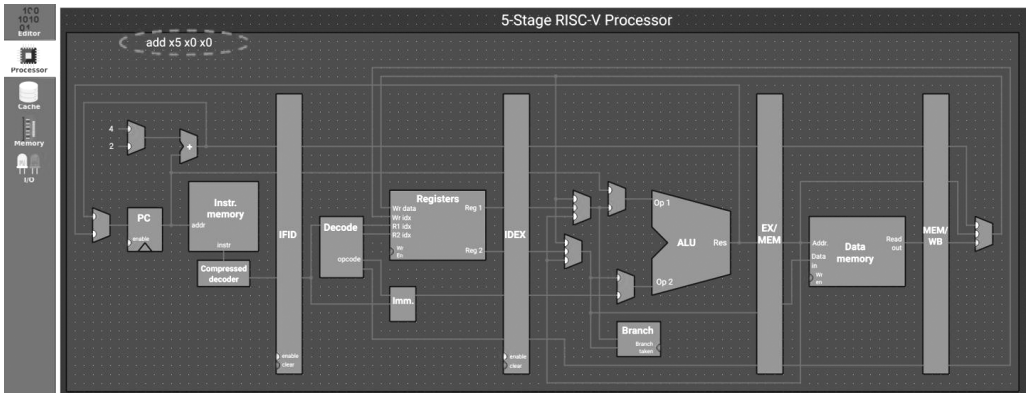


图 3-13　Ripes 流水数据通路窗口（add 指令进入）

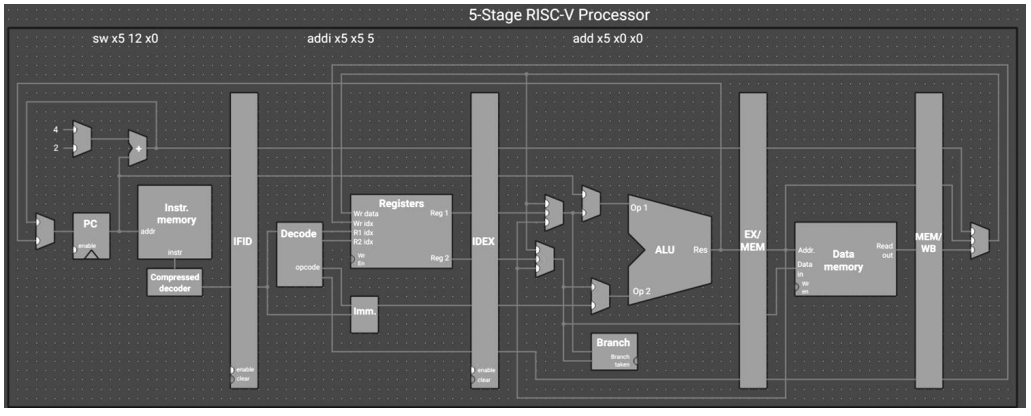


图 3-14　Ripes 流水数据通路窗口（add 指令进入 Exe 阶段）

在指令运行过程中，代码编辑区域也会同步显示当前指令的执行进度，Ripes 会使用红色高亮标注正在执行的指令，右侧还会显示该指令所处的运行阶段，如图 3-15 所示。

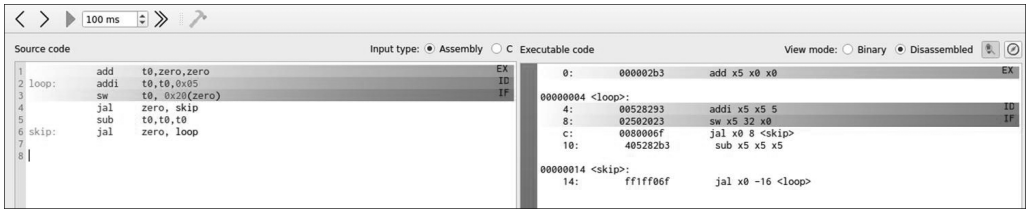


图 3-15　Ripes 运行过程中个指令所处不同流水级

当某些指令更改了目标寄存器的值时，在右侧的寄存器组的对应寄存器也会发生变化，如图 3-16 所示。

3. 冒险

冒险又称作竞争，是指在计算机 CPU 的微体系结构中，指令流水线乱序执行中的一些问题可能会导致得到不正确的计算结果。有三类典型的冒险：

- 数据冒险。
- 结构冒险。

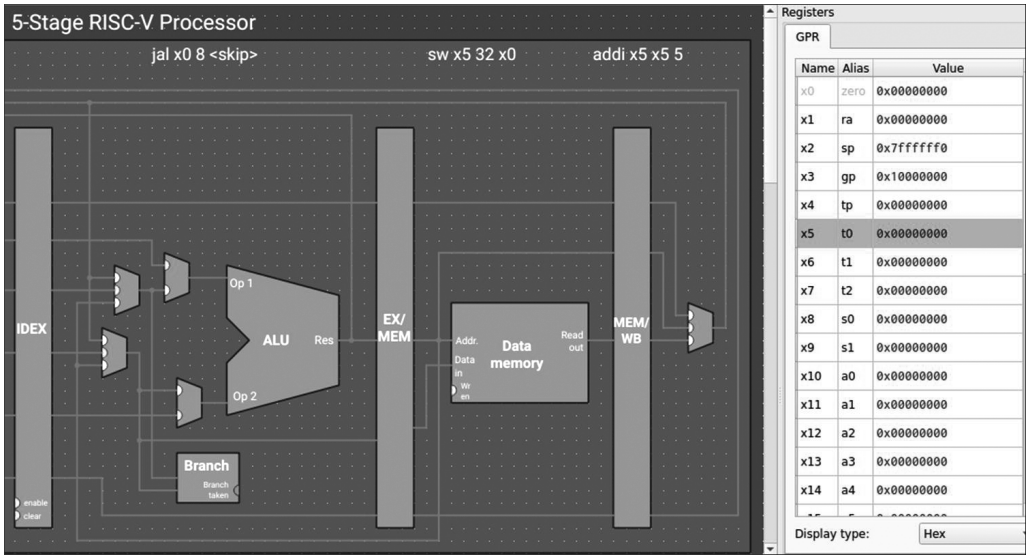


图 3-16　Ripes 运行过程中查看寄存器变化

- 控制冒险(分支冒险)。

解决冒险的方案有使用空指令使流水线停顿或者使用直接通路技术(forwarding/bypassing)。

假如不断运行 demo 程序,则当第四条指令执行到访存阶段时,Ripes 会检测到控制冒险。

冒险分析:指令 jal zero, skip 用于跳转到 skip 标签处,而由于流水线技术的存在,jal 后面两条指令在执行跳转前已经被加载到处理器中,如图 3-17 所示。

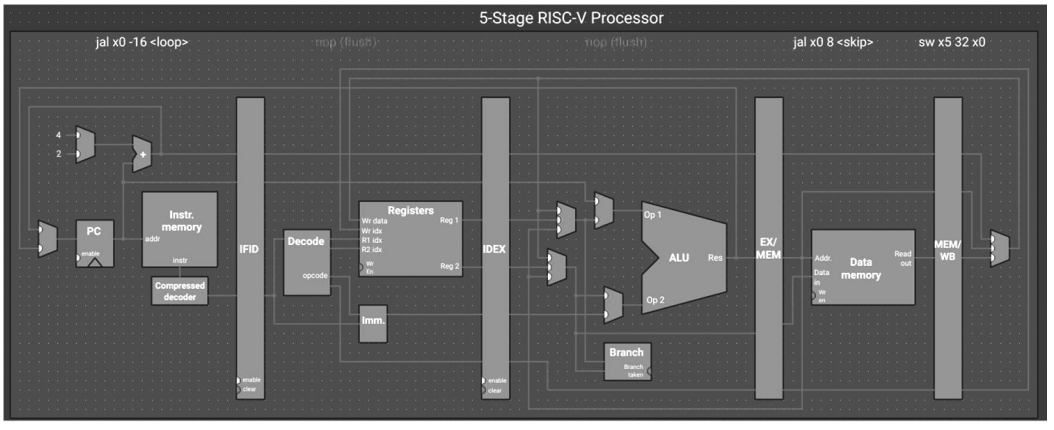


图 3-17　Ripes 流水数据通路中出现气泡

这意味着,即使程序已经跳转,控制流仍然会继续执行 jal 后面的两条指令,从而导致控制冒险的发生。

为了解决控制冒险,Ripes 采用加入空指令的方式避免后续指令被执行。其他冒险请

读者自行观察，例如编写一个指令序列形成对同一寄存器进行先写后读的两条指令，然后设置处理器架构启用或者禁用直接通路，观察指令执行的差异。后续在 riscv-mini 的冒险检测电路分析中还会仔细探讨这个问题。

3.3　小结

本章主要讲解了教学用 RISC-V 虚拟机 Ripes，内容涵盖该工具的安装和使用。本章不仅介绍了 Ripes 的各个功能区域，还详细讲解了如何使用 Ripes 观察 RISC-V 程序在处理器中的运行，帮助读者理解指令在 RISC-V 处理器中的运行方式和信号的流动方式。

通过本章的学习，读者应该能够使用 Ripes 观察自己编写的 RISC-V 汇编程序，并理解 RISC-V 指令如何在 CPU 流水数据通路上执行。此外，读者将对 RISC-V 处理器的数据通路有基本的认识，能够识别处理器内部数据通路上各个功能模块及其作用，为后续处理器设计打下基础。

练习

1. 使用 3.2.1 节的 4×4 矩阵加法汇编程序，在 Ripes 上运行和观察该程序在五级流水的 Ripes 处理器数据通路上的执行细节，特别注意跳转指令引起的流水停顿现象；打开 Ripes 的 view->show processor signal values 选项，观察并记录一条加法指令在 ID 阶段时寄存器文件输入、输出和控制端信号值；在 EXE 阶段时，ALU 输入端和输出端的信号；以及在 WB 阶段时，寄存器文件的输入、输出和控制信号的值。观察记录寄存器先写后读引起的数据冒险、条件跳转引起的气泡现象。

2. 实现矩阵乘法的分块算法，对比常规三重循环的矩阵乘法，观察各自的 Cache 命中率/缺失率并分析原因。