大作业——ERISC 程序可视化

〇、声明

本文档旨在说明大作业基本要求。请勿外传。

设置大作业的目的是希望同学们活用课程学到的知识，锻炼解决实际问题的能力，体验团队协作过程，并从过程中学到更多课堂很难涉及的知识。

“程序设计基础”课程教学团队保留对本文档内容的最终解释权。

目录

一、背景介绍

现代的计算机处理器按其指令集（即，基本指令的集合）的复杂程度，可以分为复杂指令集（Complex Instruction Set Computer, CISC）和精简指令集计算机（Reduced Instruction Set Computer, RISC)。二者的重要区别之一，是访问内存的指令。RISC 通常仅允许少量指令访问内存（例如，load 指令将内存中的值载入寄存器，而 store 指令将寄存器的值存入内存），绝大部分指令则在寄存器中完成（关于寄存器、内存和指令，详见第四节）；反之，通常 CISC 的绝大部分指令都将访问内存作为指令的一部分，使得每种运算都因访问内存的模式的不同而产生多种“变种”。

RISC-V 是遵循 RISC 原则的公开指令集标准之一。相比其他常见的 RISC 架构（例如，ARM、MIPS 等），由于 RISC-V“开源”的特点，不需要许可费即可使用（~~吐槽：鬼知道将来收不收费~~），近年来得到了广泛的关注和长足的发展。这也是贵系著名的“造机”大作业转向基于 RISC-V 实现的重要原因之一。

此次大作业参考 RISC-V 指令集，对其进行了进一步精简，形成自定义的 ERISC（Extremely Reduced Instruction Set）指令集（本文档将介绍 ERISC 指令的功能），要求同学们在理解 ERISC 指令功能的基础上，编写程序对其执行过程进行可视化。希望能够帮助同学们深入理解程序在计算机中是如何运行的，同时能够树立信心，尽快摆脱对未知的“造机”大作业的恐惧。

二、ERISC 指令说明

2.0、内存、栈和寄存器

内存（Memory）是计算机的核心部件之一。计算机程序的二进制代码（即，指令）和数据（即，变量）都存储在内存中，供处理器（即，CPU）使用。为简化起见，在 ERISC 架构中，**内存共 4M 字节**（地址从 0x000000-0x3fffff）；指令不存放在内存中；对于指令而言，内存只有地址编号，并没有“变量名”，指令可以访问任意的内存（因此需要特别小心）。

栈（Stack）是一种数据结构，特点是“先进后出”，即先入栈的数据将会后出栈。由于其在函数调用过程中起到的特殊作用，以及函数的广泛应用，指令集中通常会要求内存中的一部分作为栈来使用，并对其进行特殊处理。为简化起见，在 ERISC 架构中，栈并不是内存的一部分，是独立存在的存储空间，**共 4M 字节**（地址从 0x000000-0x3fffff）。

寄存器（Register）是一种访问速度非常快的存储器，但由于其造价相比内存十分昂贵，通常在计算机中仅有少量配备，并作为处理器的一部分。在 RISC-V 架构中，一共有 32 个寄存器，分别取名为 x0 到 x31，每个寄存器都有别名，有些寄存器还有特殊的功能。这些寄存器及其别名的对应关系如表 2-1 所示。为简化起见，在 ERISC 架构中，**共有 32 个寄存器**，其**别名与表 2-1 相同**，但**不具有特殊用途**。

另外，为简化起见，所有寄存器的大小均为 32 位，所有内存、

栈和寄存器在程序开始时的初始值均为 0。内存地址由 0 开始，栈顶在程序开始时指向 4MB 处（即，栈空间的地址最大处）。

表 2-1、RISC-V 架构中的寄存器

名称	别名	特殊功能	名称	别名	特殊功能
x0	zero	always zero	x16	a6	
x1	ra	return address	x17	a7	
x2	sp	stack pointer	x18	s2	saved register
x3	gp	global pointer	x19	s3	
x4	tp	thread pointer	x20	s4	
x5	t0	temporary	x21	s5	
x6	t1		x22	s6	
x7	t2		x23	s7	
x8	fp	frame pointer	x24	s8	
x9	s1		x25	s9	
x10	a0	return value	x26	s10	
x11	a1	return value	x27	s11	
x12	a2		x28	t3	
x13	a3		x29	t4	
x14	a4		x30	t5	
x15	a5		x31	t6	

2.1、表达方式

在本节中，使用 rd 代表目标寄存器，rs1 代表源寄存器 1，rs2 代表源寄存器 2，imm 代表立即数。

在实际指令中，rd、rs1、rs2 将被替换成 32 个寄存器之一的名称或别名。

在实际指令中，imm 将被替换成实际的数值，可以为十进制或十六进制。没有前缀的数为十进制数，如“3”、“256”；以“0x”前缀的数为十六进制数，如“0x3”、“0x100”。

2.2、内存访问指令

1）内存数据加载：load [rd],[rs]

表示将内存中地址为寄存器[rs]中值的 32 位数据加载到寄存器[rd]中。

例如：

“mov x6,0x18

load x5,x6”

表示将内存地址为 0x18 的 32 位数据加载到寄存器[x5]中，相当于 C 语言中的“x5 = *x6;”。

注意：ERISC 指令集约定为小端（Little-Endian）内存访问。在上面的例子中，如果地址位于 0x18-0x1b 的内存中存放的内容分别为 0x00、0x10、0x20、0x30，则指令执行后，x5 寄存器的内容为 0x30201000。

2）内存数据存入：store [rs],[rd]

表示将寄存器[rs]的值存入内存地址为寄存器[rd]中值的存储空间中。

例如：

“mov x6,0x18

store x5,x6”

表示将寄存器[x5]的值存入内存地址为 0x18-0x1b 的内存中，相当于 C 语言中的“*x6 = x5;”。

2.3、栈操作指令

1）入栈：push [rs]

表示将寄存器[rs]的值压入栈。

例如：

“push x5”

表示寄存器 x5 值压入栈。具体来说：如果指令执行前，栈顶指向 0x400000（即，栈空间的 4MB 处），则指令执行结束后，栈顶指向 0x3ffffc，且栈空间内 0x3ffffc-0x3fffff 的内容修改为 x5 寄存器的值（注意小端访问）。

2）出栈：pop [rd]

表示将栈顶处数据出栈，并赋值给寄存器[rd]。

例如：

“pop x5”

表示将栈顶处数据出栈，并赋值给寄存器 x5。具体来说：如果指令执行前，栈顶指向 0x3ffffc，则指令执行后，栈顶指向 0x400000，且 x5 寄存器的内容修改为栈空间内 0x3ffffc-0x3fffff 的内容。

2.4、寄存器计算指令

1）寄存器赋值：mov [rd],[rs/imm]

表示将寄存器[rs]或立即数[imm]的值赋值给寄存器[rd]。

例如：

“mov x5,x6”

表示将寄存器[x6]的值复制给寄存器[x5]，相当于 C 语言中的“x5 = x6;”。

2）寄存器加：add [rd],[rs1],[rs2/imm]

表示将寄存器[rs1]的值和寄存器[rs2]或立即数[imm]的值相加，结果赋值给寄存器[rd]。

例如：

“add x5,x6,x7”

表示寄存器 x6 值加寄存器 x7 值赋给 x5，相当于 C 语言中的“x5 = x6 + x7;”。

3）寄存器减：sub [rd],[rs1],[rs2/imm]

表示将寄存器[rs1]的值和寄存器[rs2]或立即数[imm]的值相减，结果赋值给寄存器[rd]。

例如：

“sub x5,x6,x7”

表示寄存器 x6 值减寄存器 x7 值赋给 x5，相当于 C 语言中的“x5 = x6 - x7;”。

4）寄存器乘：mul [rd],[rs1],[rs2/imm]

表示将寄存器[rs1]的值和寄存器[rs2]或立即数[imm]的值相乘，结果赋值给寄存器[rd]。

例如：

“mul x5,x6,x7”

表示寄存器 x6 值乘寄存器 x7 值赋给 x5，相当于 C 语言中的“x5

= x6 * x7;”。

5）寄存器除：div [rd],[rs1],[rs2/imm]

表示将寄存器[rs1]的值除以寄存器[rs2]或立即数[imm]的值向零舍入，结果赋值给寄存器[rd]。

例如：

“div x5,x6,x7”

表示寄存器 x6 值除寄存器 x7 值赋给 x5，相当于 C 语言中的“x5 = x6 / x7;”。

6）寄存器取余：rem [rd],[rs1],[rs2/imm]

表示将寄存器[rs1]的值除以寄存器[rs2]或立即数[imm]的余数，结果赋值给寄存器[rd]。

例如：

“rem x5,x6,x7”

表示寄存器 x6 值模寄存器 x7 值赋给 x5，相当于 C 语言中的“x5 = x6 % x7;”。

7）寄存器位与：and [rd],[rs1],[rs2/imm]

表示将寄存器[rs1]的值和寄存器[rs2]或立即数[imm]的值按位与，结果赋值给寄存器[rd]。

例如：

“and x5,x6,x7”

表示寄存器 x6 值和寄存器 x7 值按位与并赋给 x5，相当于 C 语言中的“x5 = x6 & x7;”。

8）寄存器位或：or [rd],[rs1],[rs2/imm]

表示将寄存器[rs1]的值和寄存器[rs2]或立即数[imm]的值按位或，结果赋值给寄存器[rd]。

例如：

“or x5,x6,x7”

表示寄存器 x6 值和寄存器 x7 值按位或并赋给 x5，相当于 C 语言中的“x5 = x6 | x7;”。

2.5、控制指令

0）行标识：[行标识]:

行标识表示代码位置的名字。与其他指令相比，只有行标识所在行有冒号“:”。执行到行标识所在行时无需任何操作，直接执行下一行指令即可。行标识是下列控制指令的基础。

1）无条件跳转：jal [行标识]

[行标识]应替换为一个需要跳转到的行标识。表示程序指令接下来将从[行标识]处开始执行。

例如：

```
jal hello
...
hello:
...
```

表示执行 jal 语句后，程序将继续从“hello:”所在的下一行开始执行。

2）条件跳转：beq/bne/blt/bge [rs1],[rs2],[行标识]

表示将寄存器[rs1]和寄存器[rs2]的值进行比较，如果比较结果满足条件，则跳转至[行标识]处，否则继续执行下一条指令。

简化起见，具体分为四种：

beq，寄存器[rs1]和[rs2]的值相等时发生跳转。相当于 C 语言中的“if (rs1 == rs2)”。

bne，寄存器[rs1]和[rs2]的值不相等时发生跳转。相当于 C 语言中的“if (rs1 != rs2)”。

blt，寄存器[rs1]的值小于寄存器[rs2]的值时发生跳转。相当于 C 语言中的“if (rs1 < rs2)”。

bge，寄存器[rs1]的值大于等于寄存器[rs2]的值时发生跳转。相当于 C 语言中的“if (rs1 >= rs2)”。

例如：

```
beq x5,x6,over
...
over:
...
```

表示执行 beq 语句后，程序将在 x5 等于 x6 时，将继续从“over:”所在的下一行开始执行，否则（即，x5 不等于 x6 时）将继续从“beq”的下一行开始执行。

3）函数调用：call [行标识]

调用[行标识]处的函数。与函数返回配合使用。

4）函数返回：ret

返回函数调用处。与函数调用配合使用。

例如：

```
isPrime:
...
ret
...
call isPrime
```

从“isPrime:”到“ret”实际上成为了一个函数定义。“call isPrime”为一次函数调用。具体执行效果为：当执行到“call”指令时，将“call”指令所在行的下一行的行号入栈（参见 2.3 栈操作指令），程序将继续从行标识“isPrime:”的下一行开始执行；当执行到“ret”指令时，出栈得到一个行号（参见 2.3 栈操作指令），程序将继续从这一行号开始执行。注意：call 指令和 ret 指令会操作栈空间。

三、指令操作可视化

1）程序结束符：end

指令 end 表示程序结束。此时，应输出程序的结束状态（具体要求见 4.2.2 小节）。

2）绘图指令：draw

执行 draw 指令时，应输出可视化的程序历史状态（具体格式要求见 4.2.1 小节），并重置历史状态记录。

历史状态记录，指程序从程序开始或上一次 draw 指令执行后到此次 draw 指令之间，程序对寄存器、内存、栈空间的访问情况。

可视化作图按如下结构绘制：

<table>
<tr><td>x0</td><td>x1</td><td>x2</td><td>x3</td><td>x4</td><td>x5</td><td>x6</td><td>x7</td><td>1</td><td>2</td><td>3</td><td>4</td><td rowspan="4"></td></tr>
<tr><td>x8</td><td>x9</td><td>x10</td><td>x11</td><td>x12</td><td>x13</td><td>x14</td><td>x15</td><td>5</td><td>6</td><td>7</td><td>8</td></tr>
<tr><td>x16</td><td>x17</td><td>x18</td><td>x19</td><td>x20</td><td>x21</td><td>x22</td><td>x23</td><td>9</td><td>10</td><td>11</td><td>12</td></tr>
<tr><td>x24</td><td>x25</td><td>x26</td><td>x27</td><td>x28</td><td>x29</td><td>x30</td><td>x31</td><td>13</td><td>14</td><td>15</td><td>16</td></tr>
<tr><td colspan="8">寄存器</td><td colspan="4">内存空间</td><td>栈空间</td></tr>
</table>

其中：左边表示 32 个寄存器；中间表示 4MB 内存空间，均分成 16 等份，1 号区域对应地址最低的 256KB 空间（即，地址 0x000000-0x03ffff），16 号区对应地址最高的 256KB 空间（即，地址 0x3c0000-0x3fffff）；右边表示栈空间。

在记录历史状态的期间：若对某寄存器进行过写入操作，则在可视化图中该寄存器显示红色（RGB：255,0,0）；若对某寄存器进行过读取操作，则在可视化图中该寄存器显示蓝色（RGB：0,0,255）；若

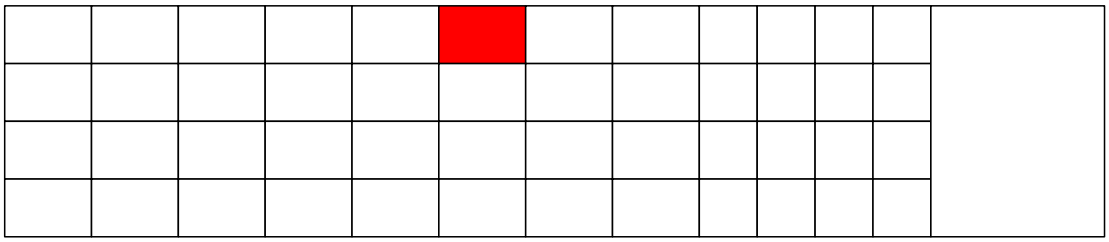
对某寄存器又读又写，则在可视化图中该寄存器显示紫色（RGB：255,0,255）；若对数据空间进行过读写操作，读写过的地址对应的块显示绿色（RGB：0,255,0）；若涉及入栈出栈操作，栈空间显示橙色（RGB：255,192,0）。

例如：

a)

“mov x5, 1

draw”



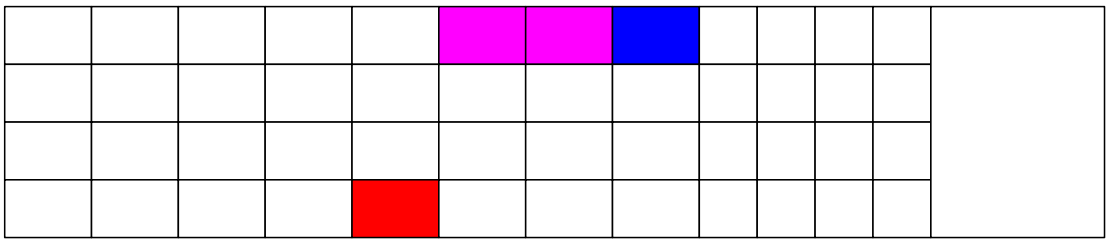
b)

“add t0,t1,t2

sub t1,t0,t2

mul t3,t0,t2

draw”



c)

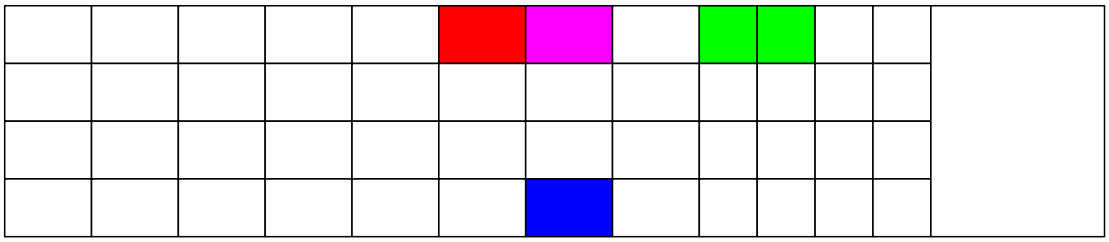
“mov x6,0x00000

load x5,x6

mov x6,0x44444

store t5,x6

draw”

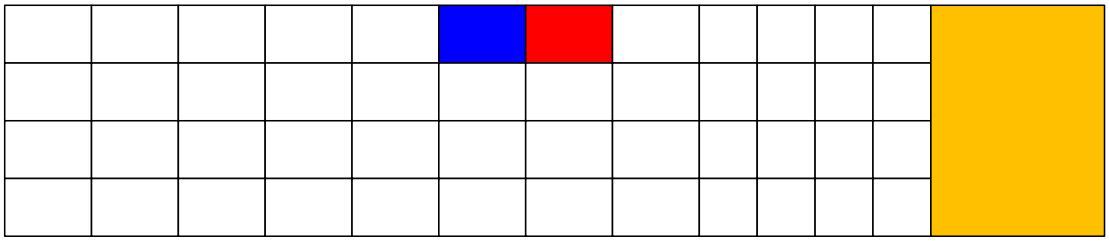


d)

“push t0

pop t1

draw”



四、输入输出要求

4.1、输入

程序输入为一个.risc 文件，其内容为 ERISC 指令的程序（具体格式见第二节）。输入方式不限，可在程序中固定输入文件路径名，亦可允许用户输入文件路径名。

4.2、输出

输出分为两部分，.bmp 文件和.txt 文件。

4.2.1、.bmp 文件输出

在每次执行 draw 指令时输出一个.bmp 文件。.bmp 文件是二进制的图像文件，输出内容为执行 draw 指令时的运行状态可视化效果图（示例见第三节）。.bmp 文件的文件名为执行 draw 指令的次数，例如，第一次执行 draw 指令产生 1.bmp，第二次执行产生 2.bmp。

注意，文件名为执行 draw 的次数，不是 draw 指令出现在代码中的次数，也不是其行号。

.bmp 文件格式参见百科资料（课堂上也会简单讲解）。

4.2.2、.txt 文件输出

在执行 end 指令时，输出一个 result.txt 文件。这是一个文本文件，输出内容为整个程序运行结束后 32 个寄存器和内存里所有值

的结果。result.txt 文件格式如下：

第一行包含 32 个整数 x0、x1、…、x31，分别为 32 个寄存器的值，以 16 进制输出（不需要 0x 开头），每两个数之间用空格分隔。

接下来若干行，以字节为单位按地址从小到大的顺序输出内存中的所有值（16 进制），每输出 64 个数换行，每行内的数之间用空格分隔。

五、必做任务

5.1、程序可视化

针对下列由 ERISC 指令组成的程序，能够正确可视化其运行状态，并正确记录其运行结果。

```
mov t0,0x1234
mov t1,0xabcd
add t2,t0,t1
sub t3,t1,t0
mul t4,t1,t0
push t0
rem t5,t1,t0
div t6,t1,t0
and t0,t2,t3
or t1,t4,t5
pop t2
draw
end
```

5.2、程序可视化

针对下列由 ERISC 指令组成的程序，能够正确可视化其运行状态，并正确记录其运行结果。

```
mov t0,x0
mov t1,1
mov t2,16
jal loop
draw
loop:
store t1,t0
call calnext
bne t2,x0,loop
draw
end
calnext:
add t1,t1,1
sub t2,t2,1
add t0,t0,0x40000
draw
ret
```

5.3、任务可视化

使用 ERISC 指令，编写一个计算 1+2……+50 的程序，将结果存放至 a0 寄存器中。能够正确运行并记录其运行结果。

5.4、任务可视化

使用 ERISC 指令，编写一个筛法判断 2～1,000,000 内所有数是否为素数的程序，将结果存储在内存中。程序运行结束时，内存中地址 i 处（2 <= i <= 1,000,000）的 1 字节的内容表达 i 是否为素数（0 代表不是素数，1 代表是素数）。在整个程序结束（即 end 指令之前）执行 draw 指令。能够正确可视化其运行状态，并正确记录其运行结果。

六、组队、提交要求及计分说明

6.1、组队要求

要求 3-5 人一组组队完成，推荐 4 人一组。每组选出一名队长。

6.2、提交要求

每组由队长在网络学堂提交一个.zip 压缩包。组员可以（但不必须）在网络学堂提交作业时说明队长姓名学号，但不要提交附件。发现组员在网络学堂提交附件的，将单独扣分。

压缩包内应包括但不限于：

1）程序源文件。放置在 src 文件夹下，可以有多级目录，应包括全部工程编译所需文件。

2）输入文件。放置在 input 文件夹下，应包含必做任务（包括 5.1、5.2、5.3、5.4 四个任务的输入文件）和扩展任务的输入文件。

3）说明文档。一个 Word 或 Pdf 文档，放置在压缩包根目录下，内容应包括但不限于：小组人员（姓名、学号、班级）、基本功能完成情况、扩展功能说明（可选）、分工情况（注意：这部分将作为小组内同学评分依据）。

4）演示视频。放置在压缩包根目录下，内容应包括但不限于：小组人员展示、基本功能演示、扩展功能演示（可选）。建议长度不超过 3 分钟，大小不超过 30MB。

如果上述内容过大无法上传至网络学堂，可分别用一个内含有效

链接的.txt 文件代替。

6.3、计分说明

1、要求主体使用 C/C++实现。完整实现第二、三、四、五节规定的全部任务就可以获得 100%的大作业分值。

2、允许调用其他库，允许调用其他语言编写的函数。但如这样做，应在说明文档中给出说明，并扣除由此节省的工作量所对等的分值。

3、本文档中，扩展功能指本文档中未说明的指令（即，扩展本文档第二节内容）、更多的可视化效果（即，扩展本文档第三节内容）、更多的输入输出交互模式（即，扩展本文档第四节内容）；扩展任务指本文档中未说明的任务（即，扩展本文档第五节内容）。实现扩展功能或扩展任务有额外加分，分值视重要程度而定。

4、**“抑制内卷条款”**：实现扩展功能并不能获得超过 100%的大作业分值，但可以用来补足由于第 2 条规定导致的扣分。请各位同学在小组内充分沟通，根据小组同学的兴趣、能力和时间安排，选择是否实现扩展功能。