计算机体系结构第一次仿真实验

实验的执行过程

对于这个任务，编写一个C程序，它是一个针对MIPS指令集的有限子集的指令级模拟器。这个指令级模拟器将对每个指令的行为进行建模，并允许用户运行MIPS程序并查看它们的输出。并使用此模拟器作为参考来验证以后的实验是否正确执行代码

实验提供了一个外壳$shell.c$，是整个$simulator$的框架，我们需要实现的是$sim.c$，是整体$simulator$的实现函数，我们需要在$process_instruction$函数中实现所有的指令以及对应的输出。

仿真实验的流程如下所示：

- 实现$process_instruction$函数
- 使用$asm2hex$工具将$/inputs$中的样例测试输入文件从MIPS汇编转换为二进制的机器码，即实现从后缀为.s到后缀为.x的过程
- 调用对应的文件，测试$simulator$的功能

在$asm2hex$工具中，需要配置$spim$的工具，$spim$是一个开源的MIPS指令模拟器，实际上我们的目的就是使用该工具实现汇编到机器码的转变。我们可以配置对应的$spim$的路径，下载对应工具。最终得到我们想要的.x后缀文件。

设计思路

上述内容是配置环境以及搞清楚整体$simulator$的执行流，我们实现$sim.c$的过程实际上是对MIPS指令集进行一系列的解析操作。我们考虑MIPS的指令：由于$simulator$实际上是接受32位的二进制的机器码，这32位机器码就决定了我们的模拟器会执行什么样的操作。所以在进行解析之前，我们首先需要熟悉MIPS32位的指令的各个字段：指令在计算机中以二进制机器码的形式存储在内存中。同样是32位的长度，根据不同的功能，我们将指令分为三种类型：

- I型指令：I型指令通常用于实现立即数运算和数据传输等操作，其格式为op $t, $s, imm，其中op表示操作码，$t和$s表示目标寄存器和源寄存器，imm表示一个16位的立即数。常见的I型指令包括addi、lw、sw、andi、ori等。例如，addi $t0, $s0, 100的机器码为0x21080064。
- R型指令：R型指令通常用于实现寄存器之间的运算和移位等操作，其格式为op $d, $s, $t，其中op表示操作码，$d、$s和$t表示目标寄存器、源寄存器和第二个源寄存器。常见的R型指令包括add、sub、and、or、sll、srl等。例如，add $t0, $s0, $s1的机器码为0x02118020。
- J型指令：J型指令通常用于实现无条件跳转操作，其格式为j $target$，其中target表示跳转目标地址。常见的J型指令包括j、jal等。例如，j 0x00400014的机器码为0x0800000d。

在计算机体系结构中，R型、I型和J型指令是不同类型的机器指令，它们的位字段（位数组）组成略有不同。这些指令类型通常用于不同的操作和操作数。

1. **R型指令（寄存器型指令）**：
    - R型指令通常用于执行寄存器之间的操作，如加法、减法等。

- R型指令的位字段通常包括：
  - 操作码（Opcode）：指示要执行的操作。
  - 目标寄存器（Destination Register）：指示结果存储的寄存器。
  - 源操作数寄存器1（Source Register 1）：第一个操作数的来源寄存器。
  - 源操作数寄存器2（Source Register 2）：第二个操作数的来源寄存器。
  - 移位量（Shift Amount）：用于一些指令的位移操作。
  - 功能码（Function Code）：指示具体的操作，通常与操作码一起使用。

2. **I型指令（立即型指令）**：
   - I型指令通常用于执行带有立即数（立即值）的操作，如加载、存储、分支等。
   - I型指令的位字段通常包括：
     - 操作码（Opcode）：指示要执行的操作。
     - 目标寄存器（Destination Register）：指示结果存储的寄存器。
     - 源操作数寄存器（Source Register）：通常表示一个寄存器操作数。
     - 立即数（Immediate Value）：包含要用于操作的常数或立即数。
3. **J型指令（跳转型指令）**：
   - J型指令通常用于实现跳转（无条件或条件）操作。
   - J型指令的位字段通常包括：
     - 操作码（Opcode）：指示要执行的操作。
     - 目标地址（Target Address）：包含跳转目标的地址或偏移量。

将指令先做解析，得到对应的opcode，由于对于所有的指令，其opcode都是6位的，所以我们可以先根据opcode，将指令划分为三种类型，然后在每条指令的内部分别用$switch--case$语句模拟对应的指令。由于我们所要模拟的是MIPS程序的执行过程，需要考虑如何读取指令以及如何更新：我们在读入操作时，将所有的指令的机器码都写入了内存的相应区域，并且使用了一个长度为32位的数组来模拟MIPS的32个通用寄存器的值，在MIPS的内部，有一个程序计数器$program$ $counter$用于指向当前执行到的指令对应的内存区域，所以我们可以通过$mem_read_32(pc)$来读取对应的指令，在每次执行完当前的指令后，可以通过更改pc的值来达到访问不同的指令区域的效果。更新pc时只需要$pc=pc+4$即可；而对于分支跳转指令，则需要将pc的值更新为跳转后的目的地址。

以add指令为例

```
uint32_t instruction = mem_read_32(CURRENT_STATE.PC);
uint32_t opcode = (instruction & 0xFC000000) >> 26;
uint32_t funct = instruction & 0x3F;
uint32_t rs = (instruction & 0x03E00000) >> 21;
uint32_t rt = (instruction & 0x001F0000) >> 16;
uint32_t rd = (instruction & 0x0000F800) >> 11;
uint32_t shamt = (instruction & 0x000007C0) >> 6;
NEXT_STATE.REGS[rd] = CURRENT_STATE.REGS[rs] + CURRENT_STATE.REGS[rt];
printf("There is ADD instructions!");
printf("The operands is:%d
%d\n",CURRENT_STATE.REGS[rs],CURRENT_STATE.REGS[rt]);
printf("The answer is :%d\n",NEXT_STATE.REGS[rd]);
```

仿真结果

```
call for go
Simulating...

Now the address is :4194304
instruction is : 604110858
opcode is : 9
the imm is : 10
now is: 0
the answer is : 10
Now the address is :4194308
instruction is : 604504069
opcode is : 9
the imm is : 5
now is: 0
the answer is : 5
Now the address is :4194312
instruction is : 621347116
opcode is : 9
the imm is : 300
now is: 5
the answer is : 305
Now the address is :4194316
instruction is : 604635636
opcode is : 9
the imm is : 500
now is: 0
the answer is : 500
Now the address is :4194320
instruction is : 625672226
opcode is : 9
the imm is : 34
now is: 500
the answer is : 534
Now the address is :4194324
instruction is : 627769389
opcode is : 9
the imm is : 45
now is: 534
the answer is : 579
Now the address is :4194328
instruction is : 12
opcode is : 0
Now the address is :4194332
instruction is : 0
opcode is : 0
Simulator halted
```

仓库链接

https://github.com/NKUshuo/Simulation