体系结构第一次仿真实验说明文档

2114052 郭裕彬

实验流程

1. 使用从网上查找到的spim程序，对应修改 `asm2hex` 部分代码，使得生成的 `.x`文件是可以运行的十六进制代码；
2. 阅读lab s1和MIPSISA文档，根据给出的需要完成的指令，查询指导书中这些指令的具体实现逻辑，匹配到 `shell.c` 设置的寄存器和机制中；
3. 使用 `src/sim inputs/[testfilename].x` 命令，运行测试文件，完成后使用 `rdump` 查看所有寄存器的值，与使用MIPS在线模拟器运行后的结果相对照，判断实验结果。

代码实现

完整代码文件附在压缩包中，本文档只对各种类型的指令的实现思路作简单说明

符号扩展

编写一个函数用于符号扩展

```
uint32_t sign_extended(uint32_t imm,uint32_t bits)
{
    uint32_t temp;
    temp = imm & ((1 << bits) - 1);
    if (temp & (1 << (bits - 1))) {
        temp |= 0xFFFFFFFF << bits;
    }
    return temp;
}
```

指令分解

将指令的各个组成部分分开取得其中的值

```
uint32_t instruction=mem_read_32(CURRENT_STATE.PC);
uint32_t op=instruction>>26;
uint32_t rs=instruction>>21&0b11111;
uint32_t rt=instruction>>16&0b11111;
uint32_t rd=instruction>>11&0b11111;
uint32_t imm=instruction&0b1111111111111111;
uint32_t targ=instruction&0b11111111111111111111111111;
uint32_t shamt=instruction>>6&0b111111;
uint32_t funct=instruction&0b111111;
```

功能实现

对 `op` 字段进行判断明确指令内容，对于部分指令再进一步判断 `funct` 或 `rt` 字段，进而实现功能。每一次完成后还需要对0号寄存器赋零以保持其永零性。

I型指令

- 算术操作指令：ADDI/ADDIU/SLTI/SLTIU
  本次设计不考虑溢出，故基本操作为 `imm` 立即数符号扩展后与 `rs` 寄存器中的值进行运算或比较，结果存放在 `rt` 寄存器中。
- 逻辑操作指令：ANDI/XORI/LUI/ORI
  基本操作是 `imm` 立即数与 `rs` 寄存器中的值进行某种逻辑运算(LUI左移操作操作数只有 `imm` 本身)，结果存放在 `rt` 寄存器中。
- 加载存储指令：LB/LH/LW/LBU/LHU/SB/SH/SW
  基本操作是将符号扩展的 `imm` 立即数与 `rs` 寄存器中的值进行相加得到虚拟地址，对该地址通过 `rt` 寄存器进行不同位宽的数据的存或取。
- 转移指令：BEQ/BGTZ/BEGZ/BGEZAL/BLTZ/BLTZAL
  基本操作是满足某个条件时或无条件地将下一个状态的 `PC` 设置为 `imm` 字地址扩展后与当前 `PC` 值相加得到的值。含 `AL` 的指令还会将当前 `PC` 值+4存入31号寄存器。对于后四条指令，需要通过 `rt` 字段的值进一步判断是哪一条指令。

R型指令

- 算术操作指令：
    - ADD/ADDU/SUB/SUBU/SLT/SLTU
      基本操作是将 `rs` 和 `rt` 寄存器中的值取出进行某种算术运算，结果存放在 `rd` 寄存器中。
    - MULT/MULTU/DIV/DIVU
      基本操作是将 `rs` 与 `rt` 寄存器中的值取出进行乘除运算，对于乘法，`LO` 保存结果的低32位，`HI` 保存结果的高32位；对于除法，`LO` 保存商，`HI` 保存余数。
- 逻辑操作指令：AND/OR/XOR/NOR
  基本操作是将 `rs` 和 `rt` 寄存器中的值取出进行某种逻辑运算，结果存放在 `rd` 寄存器中。
- 移位操作指令：SLL/SLLV/SRL/SRLV/SRA/SRAV
  基本操作是对 `rt` 寄存器中的值进行逻辑或算术移动，结果存放在 `rd` 寄存器中。对于不含 `V` 的三条指令，移动位数就是 `shamt` 字段的值；对于其余指令，移动位数是通过 `rs` 寄存器前五位的值确定的。
- 移动操作指令：MFHI/MFLO/MTHI/MTLO
  基本操作是将 `LO` 或 `HI` 中的值存到或取自 `rd` 寄存器中。
- 转移指令：JR/JALR
  基本操作是将 `rs` 寄存器中的值设置为下一个状态的 `PC`，JALR指令还会将当前 `PC` 值+4存入31号寄存器。

J型指令

- J/JAL
  转移到新的指令地址，其中低28位是 `target` 字段的字地址，高4位值当前 `PC` 值对高4位。JAL指令还会将当前 `PC` 值+4存入31号寄存器。

异常相关指令

- SYSCALL
  当2号寄存器的值为 `0x0a` 时，将 `RUN_BIT` 置位为0，终止main simulation loop。

除了有特定说明的指令，其余指令在执行完功能后都要将下一个状态的 `PC` 设置为当前 `PC`+4。

实验验证

辅助验证结果选择使用在线平台JsSpimJsSpim - Online MIPS32 Simulator Based on Spim (shawnzhong.github.io)

对比结果如下：

---

addiu.s

```
Instruction Count : 7
PC                : 0x00400018
Registers:
R0: 0x00000000
R1: 0x00000000
R2: 0x0000000a
R3: 0x00000000
R4: 0x00000000
R5: 0x00000000
R6: 0x00000000
R7: 0x00000000
R8: 0x00000005
R9: 0x00000131
R10: 0x000001f4
R11: 0x00000243
R12: 0x00000000
```

此测试程序简单与源代码对照即可判断正确实现。

---

arithtest.s

```
Instruction Count : 17
PC                : 0x00400040
Registers:
R0: 0x00000000
R1: 0x00000000
R2: 0x0000000a
R3: 0x00000800
R4: 0x00000c00
R5: 0x000004d2
R6: 0x04d20000
R7: 0x04d2270f
R8: 0x04d2230f
R9: 0x00000400
R10: 0x000004ff
R11: 0x00269000
R12: 0x004d2000
R13: 0x00000000
R14: 0x00000000
R15: 0xfffffb01
R16: 0x00000000
R17: 0x00640000
R18: 0x00000000
```

General Registers

```
R0 (r0)  = 00000000
R1 (at)  = 00000000
R2 (v0)  = 0000000a
R3 (v1)  = 00000800
R4 (a0)  = 00000c00
R5 (a1)  = 000004d2
R6 (a2)  = 04d20000
R7 (a3)  = 04d2270f
R8 (t0)  = 04d2230f
R9 (t1)  = 00000400
R10 (t2) = 000004ff
R11 (t3) = 00269000
R12 (t4) = 004d2000
R13 (t5) = 00000000
R14 (t6) = 00000000
R15 (t7) = fffffb01
R16 (s0) = 00000000
R17 (s1) = 00640000
R18 (s2) = 00000000
```

---

memtest0.s

```
Instruction Count : 32
PC                : 0x0040007c
Registers:
R0: 0x00000000
R1: 0x00000000
R2: 0x0000000a
R3: 0x10000004
R4: 0x00000000
R5: 0x000000ff
R6: 0x000001fe
R7: 0x000003fc
R8: 0x0000792c
R9: 0x000000ff
R10: 0x000001fe
R11: 0x000003fc
R12: 0x0000792c
R13: 0x000000ff
R14: 0x000000ff
R15: 0x000001fe
R16: 0x000003fc
R17: 0x0000881d
R18: 0x00000000
```

General Registers

```
R0 (r0)  = 00000000
R1 (at)  = 00000000
R2 (v0)  = 0000000a
R3 (v1)  = 10000004
R4 (a0)  = 00000000
R5 (a1)  = 000000ff
R6 (a2)  = 000001fe
R7 (a3)  = 000003fc
R8 (t0)  = 0000792c
R9 (t1)  = 000000ff
R10 (t2) = 000001fe
R11 (t3) = 000003fc
R12 (t4) = 0000792c
R13 (t5) = 000000ff
R14 (t6) = 000000ff
R15 (t7) = 000001fe
R16 (s0) = 000003fc
R17 (s1) = 0000881d
R18 (s2) = 00000000
```

---

memtest1.s

```
Instruction Count : 32
PC                : 0x0040007c
Registers:
R0: 0x00000000
R1: 0x00000000
R2: 0x0000000a
R3: 0x10000004
R4: 0x00000000
R5: 0x0000cafe
R6: 0x0000feca
R7: 0x0000beef
R8: 0x0000efbe
R9: 0x000000fe
R10: 0x000000ca
R11: 0xffffffef
R12: 0xffffffbe
R13: 0x0000cafe
R14: 0x0000feca
R15: 0xffffbeef
R16: 0xffffefbe
R17: 0x000179ea
R18: 0x00000000
```

General Registers

```
R0 (r0)  = 00000000
R1 (at)  = 00000000
R2 (v0)  = 0000000a
R3 (v1)  = 10000004
R4 (a0)  = 00000000
R5 (a1)  = 0000cafe
R6 (a2)  = 0000feca
R7 (a3)  = 0000beef
R8 (t0)  = 0000efbe
R9 (t1)  = 000000fe
R10 (t2) = 000000ca
R11 (t3) = ffffffef
R12 (t4) = ffffffbe
R13 (t5) = 0000cafe
R14 (t6) = 0000feca
R15 (t7) = ffffbeef
R16 (s0) = ffffefbe
R17 (s1) = 000179ea
R18 (s2) = 00000000
```

---

brtest0.x

```
Instruction Count : 10
PC                : 0x00400050
Registers:
R0: 0x00000000
R1: 0x00000000
R2: 0x0000000a
R3: 0x00000000
R4: 0x00000000
R5: 0x00000001
R6: 0x00001337
R7: 0x0000d00d
R8: 0x00000000
R9: 0x00000000
R10: 0x00000000
```

General Registers

```
R0 (r0)  = 00000000
R1 (at)  = 00000000
R2 (v0)  = 0000000a
R3 (v1)  = 00000000
R4 (a0)  = 00000000
R5 (a1)  = 00000001
R6 (a2)  = 00001337
R7 (a3)  = 0000d00d
R8 (t0)  = 00000000
R9 (t1)  = 00000000
R10 (t2) = 00000000
```

---

brtest1.x

观察到，在线平台在编译.s文件时会在程序开头添加初始化语句，这会影响到十六进制代码的地址，进而影响r31和其关联的寄存器的值，故通过步进验证run 1和rdump来监测寄存器的变化和程序的跳转，同时对照同学的实验结果，验证成功。

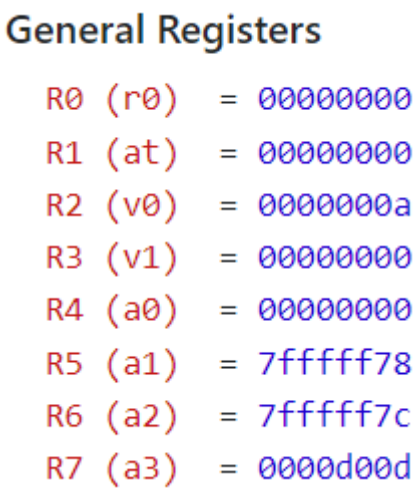
```
Instruction Count : 30
PC                : 0x0040008c
Registers:
R0: 0x00000000
R1: 0xbeb0063d
R2: 0x0000000a
R3: 0x00000001
R4: 0xffffffff
R5: 0xbef01a5e
R6: 0x00000000
```

```
R30: 0x00000000
R31: 0x00400080
HI: 0x00000000
LO: 0x00000000
```

---

brtest2.x

```
Instruction Count : 6
PC                : 0x00400020
Registers:
R0: 0x00000000
R1: 0x00000000
R2: 0x0000000a
R3: 0x00000000
R4: 0x00000000
R5: 0x00000000
R6: 0x00000000
R7: 0x0000d00d
R8: 0x00000000
```



其中，r5和r6是在线平台在程序开头添加的初始化代码影响的，与测试文件原本的代码没有关联。