Instruction-Level MIPS Simulator 实验报告

姜永韩　2113955

目录

1 实验目标

本次实验实现一个 C 程序，该程序能够模拟执行一组 MIPS 指令，通过读取以十六进制表示的 MIPS 指令机器码输入文件，并且能够执行每条指令后更新 MIPS 体系结构的状态，包括寄存器和内存。程序分为两部分，包括 shell 部分（处理输入文件和用户交互）和模拟部分（在 sim.c 文件中实现，负责指令的模拟执行）。通过此实验，建立一个可用于验证后续实验中代码正确性的 MIPS 指令模拟器。

本次实验相关代码及内容已上传 github:https://github.com/jiang-y-h/MIPS_Simulator

2 设计思路

为使 sim.c 能够执行 MIPS 指令机器码，本次实验采用自底向上的设计思路。

1. 为每条指令的执行单独设计函数，编写 process.c，其含有本次实验要实现的所有指令的执行函数。
2. 在 MIPS 指令中存在 I 型指令，以及 lb，lw，lh，sb，sw 等与立即数相关的指令，这样的指令读取的立即数大都不是 32 位的，因此，需要设计立即数符号扩展，无符号扩展指令。同时，对于这两种扩展还需要设计针对 16 位立即数扩展到 32 位立即数，和 8 位立即数扩展到 32 位立即数的函数。
3. 针对 32 位的 MIPS 指令结构，将输入的机器码分为 Opcode、rs、rt、rd、function、shamt、immediate、target 等部分，首先根据 Opcode 识别指令，当 Opcode 为 0 时，根据 function 识别指令，当 Opcode 为 1 时，根据 rt 识别指令，通过识别到的指令，根据第 1 步实现的对应的指令函数，完成指令功能。

3 具体函数设计

3.1 机器码解析

对于 32 位的 MIPS 指令解析其机器码作了如下的函数设计：

```
uint8_t get_op(uint32_t inst){ return inst >> 26;}

uint8_t get_rs(uint32_t inst){ return (inst >> 21) & 0x1f;}

uint8_t get_rt(uint32_t inst){ return (inst >> 16) & 0x1f; }

uint8_t get_rd(uint32_t inst) { return (inst >> 11) & 0x1f; }

uint32_t get_target(uint32_t inst) { return inst & 0x3ffffff;}

uint16_t get_imm(uint32_t inst) { return inst & 0xffff; }

uint8_t get_shamt(uint32_t inst) { return (inst >> 6) & 0x1f;}
```

```
uint8_t get_funct(uint32_t inst) { return inst & 0x3f; }
```

3.2 立即数扩展

对于立即数扩展的函数设计如下:

```
//16位符号扩展到32位
int32_t imm_sign_ext(uint16_t imm){
    int32_t ext_imm =(int16_t)imm;
    return ext_imm;
};

//16位无符号扩展到32位
uint32_t imm_usign_ext(uint16_t imm){
    uint32_t ext_imm=(uint32_t)imm;
    return ext_imm;
};

//8位符号扩展到32位
int32_t byte_sign_ext(uint8_t byte){
    int32_t signed_byte = (int8_t)byte;
    uint32_t ext_byte =(uint32_t)signed_byte;
    return ext_byte;
}

//8位无符号扩展到32位
uint32_t byte_usign_ext(uint8_t byte){
    uint32_t ext_byte= (uint32_t)byte;
    return ext_byte;
}
```

3.3 指令执行

以下以 J 型指令 j、I 型指令 beq、R 型指令 sll 为例展示指令执行过程的设计:

```
//j
void j(uint32_t imm){
    NEXT_STATE.PC = (CURRENT_STATE.PC & 0xf0000000) | (imm << 2);
};
```

```
//beq
void beq(uint8_t rs,uint8_t rt, uint16_t imm){
    int32_t offset=imm_sign_ext(imm)<<2;
    if(CURRENT_STATE.REGS[rs]==CURRENT_STATE.REGS[rt]){
        NEXT_STATE.PC = CURRENT_STATE.PC + offset + 4;
    }
    else{
        NEXT_STATE.PC = CURRENT_STATE.PC+ 4;
    }
};

//sll
void sll(uint8_t shamt,uint8_t rt, uint8_t rd){
    NEXT_STATE.REGS[rd] = CURRENT_STATE.REGS[rt] << shamt;
    NEXT_STATE.PC = CURRENT_STATE.PC + 4;
};
```

4 功能测试

4.1 机器码生成

利用 MARS 对 MIPS 指令的汇编代码进行机器码转换，从而生成本次实验 MIPS 指令模拟器的输入文件。

Bkpt	Address	Code	Basic		
	0x00400000	0x2402000a	addiu $2,$0,0x0000000a	2: __start:	addiu $v0, $zero, 10
	0x00400004	0x24080005	addiu $8,$0,0x00000005	3:	addiu $t0, $zero, 5
	0x00400008	0x2509012c	addiu $9,$8,0x0000012c	4:	addiu $t1, $t0, 300
	0x0040000c	0x240a01f4	addiu $10,$0,0x000001f4	5:	addiu $t2, $zero, 500
	0x00400010	0x254b0022	addiu $11,$10,0x00000022	6:	addiu $t3, $t2, 34
	0x00400014	0x256b002d	addiu $11,$11,0x0000002d	7:	addiu $t3, $t3, 45
	0x00400018	0x0000000c	syscall	8:	syscall

Figure 1: 机器码生成

4.2 模拟功能验证

将生成的机器码文件作为模拟器的输入，利用 rdump 查看运行结果, 如 figure 2。

```
PS E:\ar\simulator\src> .\sim ../inputs/addiu.x
MIPS Simulator

Read 7 words from program into memory.

MIPS-SIM> go

Simulating...

Simulator halted

MIPS-SIM> rdump
```

Figure 2: 模拟器测试

通过对比以机器码作为输入的模拟器运行结果和以汇编代码作为输入的 MARS 运行结果，如 figure 3，可以得到两者运行结果相同，从而验证得到模拟器的功能正确。

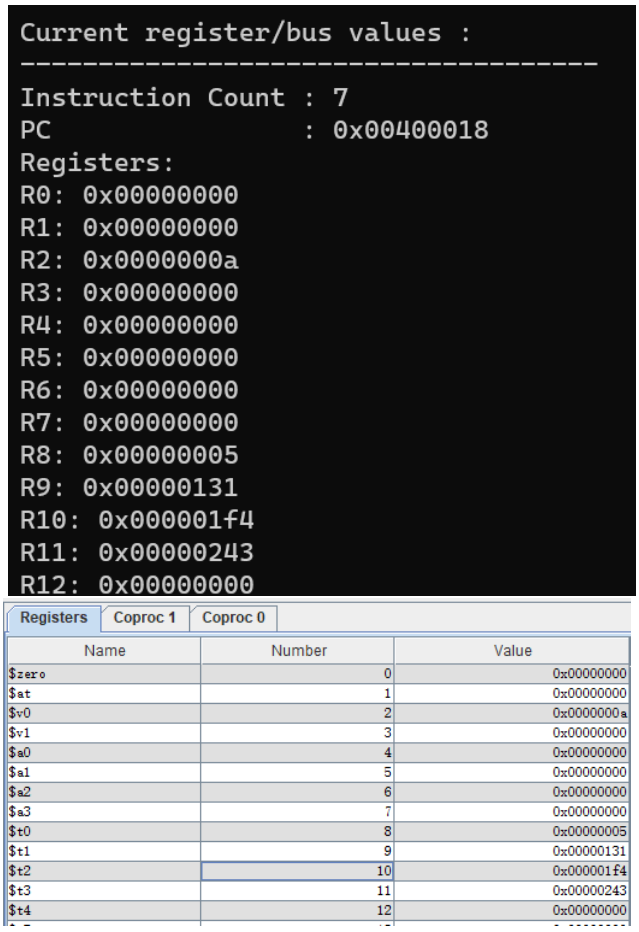

Figure 3: 运行结果对比 (上为本次实验实现的模拟器，下为 MARS)