MIPS模拟仿真实验

学号:2113637 姓名:宁可心

概要

- 在这个实验中，需要编写一个功能模拟器，模拟 MIPS 架构的一个子集。

  文件目录结构如下：

  1. “src/” 子目录：
     - “shell.h” 和 “shell.c” 文件为 shell 实现，不需要修改。
     - “sim.c” 文件为模拟器的骨架，需要你进行修改。
  2. “inputs/” 子目录：
     - “.s” 文件为模拟器的测试输入，包含 MIPS 汇编代码。
     - “asm2hex” 是一个汇编器/十六进制转储工具，用于将汇编代码转换为机器码并生成十六进制转储文件。

  需要进行的步骤如下：

  1. 修改 “sim.c” 文件，实现实验手册中指定的 MIPS 指令集。使用 “make” 命令编译模拟器。

     ```
     $ cd src/
     $ make
     ```

  2. 使用 “asm2hex” 工具将测试输入（“.s” 文件）转换为汇编后的机器码的十六进制转储文件（“.x” 文件）。

     ```
     $ cd inputs/
     $ ./asm2hex addiu.s
     ```

     通过这一步，会得到一个名为 “addiu.x” 的十六进制转储文件。对于其他的测试输入也需要重复这个步骤。

  3. 在模拟器中运行十六进制转储文件。

     ```
     $ src/sim inputs/addiu.x
     ```

     运行后会显示 “MIPS Simulator” 的信息，并将程序中的指令加载到内存中。然后输入 “go” 命令来启动模拟器的执行。由于模拟器还没有被完全实现，所以它会变得无响应。可以按下 Ctrl+C 来退出模拟器。

核心部分—Sim.c

编写思路

1. 先对指令进行解析，得到`opcode`
2. 根据`opcode`将指令分为三类，并在每一类的分支进行switch操作，模拟对应指令
3. 完善switch语句，对其进行细化

4. 完善更新PC值操作

代码简介

首先读取32位指令，并将其转换为二进制形式。然后，它判断指令是否为BREAK，如果是的话，停止运行程序并设定$v0的值为10。最后，将运行标记位RUN_BIT设为false。

```
    // 读32位指令，并把这个32位指令转换成二进制（因为mem_read_32函数返回的是32位指令的十进制形式）
    // printf("%u\n",mem_read_32(CURRENT_STATE.PC));
    uintToStr(mem_read_32(CURRENT_STATE.PC));
    strncpy(instructionCode, conversion, 33);
    // 如果指令为BREAK，那么就停止运行，设定$v0的值为10,RUN_BIT运行标记位false并返回
    // 对应每个.s文件中syscall的命令
    if (str2Uint(0, 31) == 12)
    {
        if (NEXT_STATE.REGS[2] == 10)
        {
            RUN_BIT = 0;
            return;
        }
    }
```

接下来，根据 func 的值进行 switch 语句的判断,func表示实现的 MIPS 指令集中的一些功能，包括 ADD、OR、AND、XOR、SUB 等运算操作，以及 SLL、SRL、SRA 等位移操作，还有 JUMP 和 BRANCH 指令等。

Type=1,2,3分别表示R型，J型，I型指令

```
//R
switch (funct)
        {
        case 32: // ADD
            // strcpy(instruction, "ADD");
            sprintf(instruction, "%s %s %s %s %s", "instruction ADD", rdName,
rsName, rtName, "executed");
            NEXT_STATE.REGS[rd] = CURRENT_STATE.REGS[rs] +
CURRENT_STATE.REGS[rt];
            break;
        case 33: // ADDU
            sprintf(instruction, "%s %s %s %s %s", "instruction ADDU", rdName,
rsName, rtName, "executed");
            NEXT_STATE.REGS[rd] = CURRENT_STATE.REGS[rs] +
CURRENT_STATE.REGS[rt];
            break;
        case 37: // OR
            sprintf(instruction, "%s %s %s %s %s", "instruction OR", rdName,
rsName, rtName, "executed");
            NEXT_STATE.REGS[rd] = CURRENT_STATE.REGS[rs] |
CURRENT_STATE.REGS[rt];
            break;
        case 36: // AND
            sprintf(instruction, "%s %s %s %s %s", "instruction AND", rdName,
rsName, rtName, "executed");
```

```
            NEXT_STATE.REGS[rd] = CURRENT_STATE.REGS[rs] &
CURRENT_STATE.REGS[rt];
            break;

          …

        }
//J
switch (type)
        {
        case 2: // J
            // strcpy(instruction, "J");
            NEXT_STATE.PC = (uint32_t)address - 4;
            sprintf(instruction, "%s %d %s", "instruction J", address,
"executed");
            break;
        case 3: // JAL，跳转到指定地址，并将返回地址存储在寄存器31（%ra）中
            // Salvando RA
            // strcpy(instruction, "JAL");
            NEXT_STATE.REGS[31] = CURRENT_STATE.PC + 4;
            // ***这里需要+4，JAL放入PC的是下一条指令的地址
            // ***死循环就是因为这里没+4
            NEXT_STATE.PC = address - 4;
            // ***这里要-4，因为switch出去之后+4
            sprintf(instruction, "%s %d %s", "instruction JAL", address,
"executed");
            break;
        }
//I
 switch (type)
        {
            // ***下面使用complemento2函数为了求补码，一个数求两次补码变成本身，mips指令中
I型指令立即数为补码形式

        case 1: // BLTZ
            // strcpy(instruction, "BLTZ");
            if (rt == 0)
            { // BLTZ
                sprintf(instruction, "%s %s %d %s", "instruction BLTZ", rsName,
immediate, "executed");
                if (NEXT_STATE.REGS[rs] & 0x80000000)
                {
                    NEXT_STATE.PC += immediate * 4;
                }
            }
            else if (rt == 1)
            { // BGEZ
                sprintf(instruction, "%s %s %d %s", "instruction BGEZ", rsName,
immediate, "executed");
                if (!(NEXT_STATE.REGS[rs] & 0x80000000))
                {
                    NEXT_STATE.PC += immediate * 4;
                }
            }
          …
```

```
 }
```

str2Uint函数

作用是将一个表示二进制数的字符串（instructionCode）从索引start到end的位转换为一个32位无符号整数。它通过遍历指定范围内的每一位，并根据位的值（'0'或'1'）计算对应的数值。最后，将所有数值相加得到结果

```
int str2Uint(int start, int end)
{ // 取1-5位：索引0到4
    int temp = 0;
    int length = end - start + 1;
    for (int i = start; i <= end; i++)
    {
        if (instructionCode[i] == '1')
        {
            temp += (int)pow(2, length + start - 1 - i);
            // printf("%s %d\n","add",(int)pow(2, length - 1 - i));
        }
    }
    return temp;
}
```

uintTostr函数

作用是将一个32位无符号整数转换为一个表示其二进制字符串形式的字符串。它首先将一个32位无符号整数转换为一个8位的十六进制字符串，并在需要时在左侧补充零。然后，根据每个十六进制字符的值，将其转换为一个4位的二进制字符串。最终，将所有的二进制字符串连接起来得到结果

```
void uintToStr(uint32_t u)
{
    char c[50];
    char h[50];
    int size = 0;
    sprintf(c, "%x", mem_read_32(NEXT_STATE.PC));
    // Tamanho a partir do primeiro não nulo
    while (c[size] != '\0')
    {
        size++;
    }
    // Poe zeros à esquera
    for (int i = 0; i < 8 - size; i++)
    {
        h[i] = '0';
    }
    for (int i = 8 - size; i < 8; i++)
    {
        h[i] = c[i - 8 + size];
    }
    h[8] = '\0';
    printf("%s 0x%s\n", "Hexadecimal instruction code is", h);
    // Converte de HEX para BINARIO
    int j;
    for (int i = 0; i < 8; i++)
    {
```

```
        j = i * 4;
        switch (h[i])
        {
        case '0':
            conversion[j] = '0';
            conversion[j + 1] = '0';
            conversion[j + 2] = '0';
            conversion[j + 3] = '0';
            break;
        case '1':

        …
            break;
        }
    }
}
```

输入文件处理

运用了qtspim将MIPS汇编转换为二进制机器码

在软件中导入.x文件，随后文件会将每行命令生成为机器码指令，由此便实现了MIPS汇编的.s文件向机器码.x文件的转化

具体步骤

1. 打开qtspim并导入MIPS汇编程序：启动qtspim后，选择"File"菜单中的"Open"选项，打开它。
2. 模拟程序并生成机器码：在qtspim的界面上，你可以看到MIPS汇编程序的内容。点击"Simulator"菜单中的"Simulate"选项以模拟程序的执行。qtspim将逐行读取汇编程序并将其转换为机器码指令。程序执行过程中的寄存器和内存状态也会显示在qtspim的界面上。
3. 导出机器码：一旦程序成功模拟执行并且你确认生成的机器码是正确的，可以将机器码导出为二进制文件。选择"File"菜单中的"Dump Memory"选项，然后选择导出机器码的内存段范围和文件格式（如二进制格式）。然后点击"Save"按钮。