|  |  |  |
| --- | --- | --- |
| 《计算机组成与体系结构》实验报告 |  | 天津大学本科生实验报告 |
| 学院 智算学部 年级 2022 级 班级 5 班 组号 20  课程名称 计算机组成与体系结构 成绩  组长 李国鸿 同组实验者 | **三．实验原理与步骤（不需要贴代码）**  请参考TEMU所需支持的10项功能和golden trace，简单描述各项功能是如何实现的？注意：如果需要贴代码，只需要贴关键代码块，不要整段粘贴。   1. help, c, q   都已在框架中实现。   1. si [N]   直接调用 cpu\_exec，如果 N 未给出默认为 1。   1. info SUBCMD   分成 r 和 w 两种情况。  r 情况，直接调用 display\_reg()。  w 情况，递归输出每个监视点的信息：  void print\_wp\_rcs(WP \*wp){      if(wp == NULL){          return;      }      print\_wp\_rcs(wp->next);      printf("%d\t%s\t0x%08x\n", wp->NO, wp->expr, wp->value);  }   1. x N EXPR   调用表达式求值，得到地址值。  然后用 mem\_read()，顺次访存 N 个字节，每四个进行一次换行。   1. w EXPR   从 \_free 中取出一个新的监视点，插入到 head 前面，并调用表达式求值。  在每条指令执行后，都计算监视点中所有表达式的新值，比较和旧值是否相同，如果不同，则输出信息。  /\* TODO: check watchpoints here. \*/          WP \*wp = get\_head();          while(wp != NULL)          {              bool success = true;              uint32\_t value = expr(wp->expr, &success);              if(value != wp->value)              {                  // 输出信息              }              wp = wp->next;          }  **四．实验结果（请给出仿真器TEMU的截图）**  仿真程序1：  logic（所给示例程序）  指令过程：  用 lui 和 ori 给 $at 和 $v0 赋值，  然后用 and 令 $t0 = $at & $v0。  结果截图：    Golden Trace：    仿真程序3：  addi  指令过程：  用 lui 和 ori 给 $at 赋值，  然后用 addi 令 $v0 = $at - 2。  结果截图：    Golden Trace：    3. nop 指令的识别  如果一个指令的 32 位全 0， 就应当被识别为 nop，即什么也不做。  但是不特殊处理这种情况的话，全 0 指令可能会被识别为：  sll $zero, $zero, 0  虽然这样识别出来也没什么关系（因为也是什么都没有做），但输出的汇编信息可能会造成一些困扰。于是在 sll 的 make\_helper 特殊处理如下：  make\_helper(sll) {      if(instr == 0){          sprintf(assembly, "nop");      }  else{  ...  }  }    **教师签字：**  **年 月 日** |
| 实验项目名称 指令集仿真器 -- TEMU   1. **实验目的**   采用高级语言（C/C++、Java或Python等）设计面向32位MIPS或LoongArch指令集子集的指令集仿真器 -- TEMU，用于实现对程序执行过程的模拟和调试。该指令集仿真器可模拟常见指令，支持单步执行、断点、显示寄存器信息、显示特定存储单元信息等功能，为后续的处理器设计实验提供仿真测试工具。  模拟指令的执行，产生 golden trace，方便后续与板上 SoC 进行交叉验证。   1. **实验内容** 2. 实现调试功能   完善仿真器的调试功能，提供如下交互命令：  单步执行、  打印寄存器值 / 监视点值、  表达式求值、  扫描内存、  设置监视点、  删除监视点。   1. 模拟指令集   根据已有的框架代码，模拟每条指令的取值、译码、执行、访存、写回。对于所给的汇编代码，可以在仿真器上进行执行，并实时监测寄存器和内存中的值。   1. 记录 Golden Trace   对于每条指令，如果涉及到对寄存器的写入，要在 Golden Trace 文件中记录：  指令的 pc 值、  寄存器的编号、  写入的值。   1. 汇编测试   自行学习汇编代码的编写，测试指令集的模拟是否符合预期。  6. d N  从 head 开始，循环找到 N 号监视点，删除。  WP \*p = get\_head();      while(p != NULL){          if(p->NO == n){              free\_wp(p);              printf("Delete watchpoint #%d.\n", n);              return 0;          }          p = p->next;      }  7. Golden Trace  仿照 log.txt 的记录，进行 golden.txt 的记录。  增加文件 golden.h 和 golden.c 实现对 Golden Trace 的初始化和记录功能。  FILE \*golden\_fp = NULL;  void init\_golden() {      golden\_fp = fopen("golden.txt", "w");      Assert(golden\_fp, "Can not open 'golden.txt'");  }  void golden\_write(uint32\_t pc, uint32\_t reg, uint32\_t value) {      fprintf(golden\_fp, "0x%08x\t%u(%s)\t0x%08x\n", pc, reg, regfile[reg], value);  }  然后在指令的 make\_helper 中，如果该指令涉及寄存器写入，就记录 Golden Trace。  以 lui 为例，在最后记录 pc，写寄存器，和寄存器新值：  make\_helper(lui) {      decode\_imm\_type(instr);      reg\_w(op\_dest->reg) = (op\_src2->val << 16);      sprintf(assembly, "lui   %s,   0x%04x", REG\_NAME(op\_dest->reg), op\_src2->imm);      golden\_write(cpu.pc, op\_dest->reg, reg\_w(op\_dest->reg));  }  仿真程序2：  mem （所给示例程序）  指令过程：  用 lui 和 ori 给 $at 赋值，  用 sw 和 sb 将 $at 存到内存，  再用 lw 和 lb 从内存读取到 $v0。  结果截图：    Golden Trace：    **五．实验中遇到的问题和解决办法，并谈一下通过本次实验所获得的收获。**  1. 大端序和小端序的判断  其实课上已经讲过了，但实验时候还是希望能确定一下，这里通过指令输出形式判断。  观察输出二进制指令处，根据循环看出是从高地址向低地址输出的。  void print\_bin\_instr(uint32\_t pc) {      int i;      int l = sprintf(asm\_buf, "%8x:   ", pc);      for(i = 3; i >= 0; i --) {          l += sprintf(asm\_buf + l, "%02x ", instr\_fetch(pc + i, 1)); // little endian      }      sprintf(asm\_buf + l, "%\*.s", 8, "");  }  再结合输出的二进制指令，和参考手册中的编码方式作对比。  1fc00014: 4a 00 00 00 temu\_trap  可以发现高地址对应指令的大端，**低地址对应指令的小端**。  由此确定该 MIPS 处理器的存储方式为小端序，这样方便对示例程序 mem 的预期结果作判断。   1. 符号扩展的实现   一个简单的思路是判断最高位是 0 还是 1，然后扩展补相应的值。  不过经思考后，得到一种写法相对简单的做法：  // sign extent 16 bits to 32 bits  int32\_t sext(int32\_t x) {      return (int32\_t)((int16\_t)x);  }  即利用 c 语言类型转换的特性，先截断成 16 位（因为只有低 16 位是有效的），再符号扩展到 32 位。 |
|  |  |  |