计算机系统实验二

王需2011013248

2013 年 12 月 30 日

1 综述

在本次实验中，我根据作业说明提供的思路完成了ping/pong通信的逻辑部分，并设计了测试程序，测试该实现是有效的。实验的细节包括：添加一条指令，IEXCHANGE；修改了Y86模拟器的内存模型，设计一个测试程序。

2 IExchange实现

IExchange 指令的功能是在读取内存的同时也要修改内存的值。指令的执行过程如下图所示：

Stage	IEXCHANGE
Fetch	icode: ifunc ← $M_1[PC]$ rA: rB ← $M_1[PC]$ valC ← $M_1[PC+1]$ valP = PC + 6
Decode	valB ← R[rB]
Execute	valE ← valB
Memory	valM = $M_4[valE]$ $M_{4[valE]}$ = valE
Write Back	R[rB] ← valM
PC update	PC ← valP

图 1: iexchange

$$exchange\ (\%addr)\ \%reg \tag{1}$$

具体实现参加源码中的pipe-full.hcl文件。为了使用该条执行，我们还要将这个指令添加到yas-grammar.lex文件中以及修改isa.h和isa.c文件。

3 Y86模拟器机理及修改方法

程序模拟器首先将.ys文件编译成.yo文件，在进行模拟的时候将.yo文件拷贝到一块内存mem_t中，在执行中每次从内存中取出一条指令，进行执行。最后程序比较执行得到的内存和原来的内存块的差异，来发现程序做了什么样的改变。然后程序通过比较通过pipe方式和ISA方式的执行结果比较来判断指令执行的结果是否一致。根据程序的原理，我们做如下的操作来修改它的模拟器的逻辑。

如何使用同一块共享内存？ 在程序中定义一个共享内存的key, 每次运行的时候，使用同一块共享内存。为了能让模拟器使用这块内存，将该块共享内存映射到原来分配的mem_t中，在本次程序中，我们的共享内存映射到内存块的$1024 - 2^{15}$位置。当然把地址映射到更高的位置安全性更好，但是没有特别的必要，因为我们汇编程序最后被编译成.yo, .yo文件类似于机器代码，每一条语句只占四个字节。所以一个200行的汇编程序，只要占用800byte的地址空间，这对这个程序已经足够了。

如何访问这块共享内存？ 为了能够访问这块内存，我们主要修改了如下几个函数: 下面仅以get_word_byte函数为例，显示如何修改相应的函

```
/* Get byte from memory */
bool_t get_byte_val(mem_t m, word_t pos, byte_t *dest);

/* Get 4 bytes from memory */
bool_t get_word_val(mem_t m, word_t pos, word_t *dest);
bool_t ignore_share_get_word_val(mem_t m, word_t pos, word_t *dest);

/* Set byte in memory */
bool_t set_byte_val(mem_t m, word_t pos, byte_t val);

/* Set 4 bytes in memory */
bool_t set_word_val(mem_t m, word_t pos, word_t val);
```

图 2: functions

数。 具体修改细节，可以参看isa.h 和isa.c文件。

```
bool_t get_byte_val(mem_t m, word_t pos, byte_t *dest)
{
    if (pos < 0 || pos >= m->len)
        return FALSE;
        if (pos >= ADDR_START && pos < ADDR_END)
                *dest = sharedMem[pos - ADDR_START];
        else
            *dest = m->contents[pos];
    return TRUE;
}
```

图 3: get_word_val

4 测试

测试的第一步是检查我们对程序的修改是否对上兼容，我们运行ptest中的make命令。运行得到的结果如图：

```
xu@xu-desktop:~/Documents/sim/ptest$ make
./optest.pl -s ../pipe/psim
Simulating with ../pipe/psim
  All 49 ISA Checks Succeed
./jtest.pl -s ../pipe/psim
Simulating with ../pipe/psim
  All 64 ISA Checks Succeed
./ctest.pl -s ../pipe/psim
Simulating with ../pipe/psim
  All 22 ISA Checks Succeed
./htest.pl -s ../pipe/psim
Simulating with ../pipe/psim
  All 600 ISA Checks Succeed
```

图 4: 检查程序是否向上兼容

测试程序应该是本次作业工作量最大的一部分了，因为要用单纯的Y86汇编写一个测试程序，但是这个过程收获还是很多的，其中一点就是终于体会到从C语言到汇编这层抽象牺牲了一定程度的执行效率，C语言中一条语句是一个执行单元，那么对寄存器的利用就显得有些不是很到位了，这也是为什么我们会在一些核心的程序中使用汇编进行混编，但是汇编的特点也让他的跨平台性显得有些不足。下面介绍一下我的测试程序的思路。 程序把共享内存共分为三段，分别是ID,Ptr, Data。其中ID段用来表示程序的身份，是A还是B来进行ping/pong通信，Ptr指向数据段尚未使用的内存的首地址。Data来存储共享数据。程序启动后，首先抢占ID处的值，如果得

图 5: Figure

到的值是0,标明是A号，如果抢到的不是0，则标明该程序的身份是B.身份确定后，A，B开始通信，A向B发送i个数据为2i +1的数据，标明这是A写入的数据，B则A写入数据的期间，一直检查Ptr的值，如果Ptr的值发生改变说明A已经写完了数据，B可以写自己的数据。B向A写入i个2*i的数据。如此循环下去，当双方各发送了100次数据后,消息发送结束，程序分别将ID位置的数增加1，通知对方已经执行完成，然后两个程序分别检查这个位置是否为3，如果是3则退出，否则一直等待下去。这样的话我们的预期结果应该是3,2，55，44，777，666...
在本次作业中的程序命名为test.ys，在/sim/misc文件下，在执行时需要先将其编译为.yo文件，执行以下命令： 为了启动两个模拟器分别执行如下两条

```
./yas test.ys
```

图 6: compile

指令, 分别将执行结果保存到1.txt，和2.txt中。 最后得到的结果对比如下：

```
xu@xu-desktop:~/Documents/sim$ pipe/psim -v 1 misc/test.yo -l 1000000000 -t >1.t
xt
```

图 7: start

两个程序启动的时间不同，自此共享内存的状态已经有了一定的变化，从图中可以看书在0x400处的值已经被修改成为了1.且数据的顺序保持良好。
最后执行ISA和pipe的两套模拟器的执行结果，得到的两个文件中的结果如下： 由于程序的一直执行过程中，我们一直使用的是同一块内存，如果不对共享内存进行初始化就会发生冲突，但是我们不能指望在模拟器中初始化这段内存，因为可能另外的程序已经修改了这个值，因此模拟器每次启动时都要申请共享内存模拟成功后删除这段共享内存，linux系统每次在创建一块共享内存的时候会将这块内存初始化为0，删除这块内存时并不是立即将内存删除，而是当对这块内存的引用都退出后才删除这块内存。

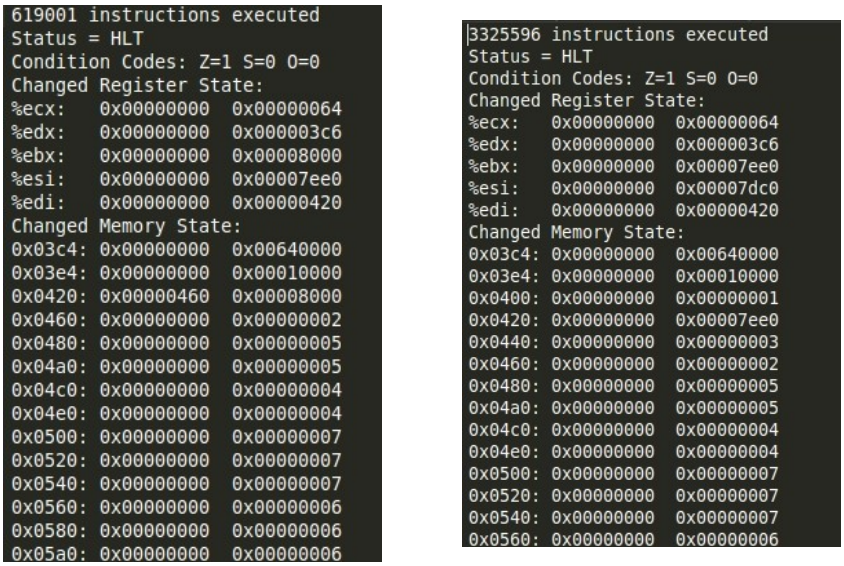

图 8: result

图 9: ISA和pipe的比较

在pipe和isa两种模拟方式中，共享内存已经被修改因此会发生冲突。我们可以使用让模拟器休眠的方法，等待程序退出后在进行第二次模拟，比如，首先，当pipe 模拟结束后，将模拟器睡眠5s，等待两个模拟程序都执行完成，然后将执行结果从共享内存中拷贝出来，在睡眠5s去清理共享内存。清理完成后等5s就可以重新执行isa 的模拟方法。这种方法看上去并不是十分合理，所以在模拟器中，我申请了两块共享内存，一块用于pipe模拟，一块用于ISA模拟。而在测试程序中，两个程序执行完成后将1024内存的值加1，通知对方消息已经发送完毕，然后不断查看1024 共享内存的状态，如果发现为3，说明两个程序都已经完成，测序退出。最后我们发现，最后得比较在大多数情况下都会显示成功，偶尔会显示错误，差别正好是两个寄存器的值相反，但这并不是我们的程序有问题，而是说两个程序分别第一次执行时身份为A，第二次执行时身份为B.具体修改见psim.cp

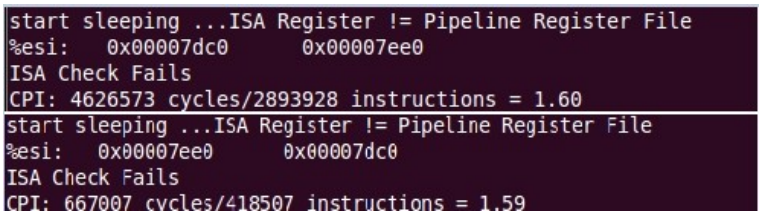

图 10: 两个程序分别执行了不同身份后的结果

5 总结

本次程序完成了作业的基本要求，并进行了严密的测试，程序结果运行正常。在大作业的完成过程中，我学习了pipe的流水线设计的思路，以及尝试对共享内存进行编程，在汇编程序的书写中吃了不少苦头，但是也学习到了很多的东西。应该说，这半年来从处理器的角度重新认识了程序的设计，感觉对理解C语言，理解程序设计都有很大的帮助。感谢老师在这半年的辛勤工作，希望您身体健康，工作顺利。