计算机组成原理·实验1报告

计01 容逸朗 2020010869

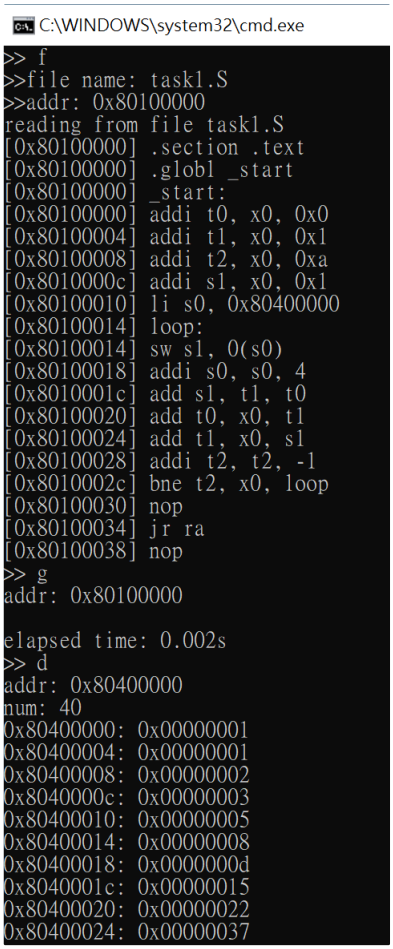
实验结果

> 1. 编写汇编程序，求前 10 个 Fibonacci 数，将结果保存到起始地址为 0x80400000 的 10 个字中，并用 D 命令检查结果正确性。

代码：

```
.section .text
.globl _start
_start:
    addi t0, x0, 0x0
    addi t1, x0, 0x1
    addi t2, x0, 0xa
    addi s1, x0, 0x1
    li s0, 0x80400000
loop:
    sw s1, 0(s0)
    addi s0, s0, 4
    add s1, t1, t0
    add t0, x0, t1
    add t1, x0, s1
    addi t2, t2, -1
    bne t2, x0, loop
    nop
    jr ra
    nop
```

实验结果：



> 2. 编写汇编程序，将 ASCII 可见字符（0x21~0x7E）从终端输出。

代码：

```
.section .text
.globl _start
_start:
    addi a0, x0, 0x20
    addi a1, x0, 0x7e
    li t0, 0x10000000
loop:
    addi a0, a0, 1

.TESTW:
    lb t1, 5(t0)
    andi t1, t1, 0x20
    beq t1, zero, .TESTW

.WSERIAL:
    sb a0, 0(t0)
    bne a0, a1, loop
    nop
    jr ra
    nop
```

实验结果：

選取 C:\WINDOWS\system32\cmd.exe

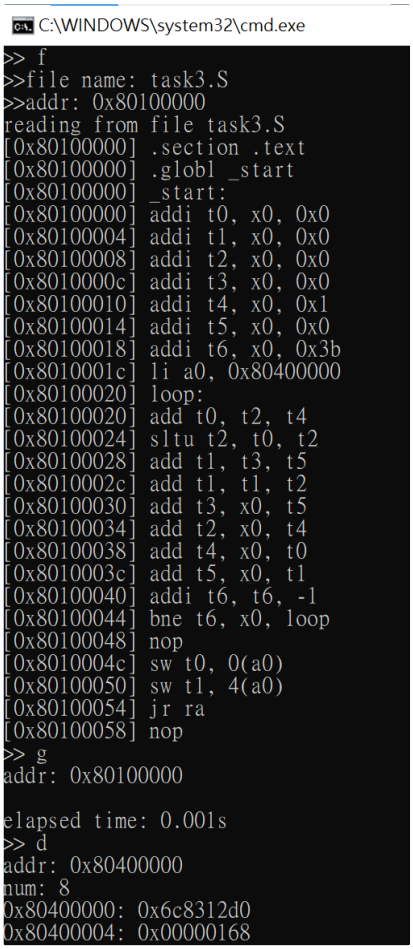
```
>> f
>>file name: task2.S
>>addr: 0x80100000
reading from file task2.S
[0x80100000] .section .text
[0x80100000] .globl _start
[0x80100000] _start:
[0x80100000] addi a0, x0, 0x20
[0x80100004] addi a1, x0, 0x7e
[0x80100008] li t0, 0x10000000
[0x8010000c] loop:
[0x8010000c] addi a0, a0, 1
[0x80100010]
[0x80100010] .TESTW:
[0x80100010] lb t1, 5(t0)
[0x80100014] andi t1, t1, 0x20
[0x80100018] beq t1, zero, .TESTW
[0x8010001c]
[0x8010001c] .WSERIAL:
[0x8010001c] sb a0, 0(t0)
[0x80100020] bne a0, a1, loop
[0x80100024] nop
[0x80100028] jr ra
[0x8010002c] nop
>> g
addr: 0x80100000
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
elapsed time: 0.002s
```

3. 编写汇编程序，求第 60 个 Fibonacci 数，将结果保存到起始地址为 0x80400000 的 8 个字节中，并用 D 命令检查结果正确性。

代码：

```
.section .text
.globl _start
_start:
    addi t0, x0, 0x0
    addi t1, x0, 0x0
    addi t2, x0, 0x0
    addi t3, x0, 0x0
    addi t4, x0, 0x1
    addi t5, x0, 0x0
    addi t6, x0, 0x3b
    li a0, 0x80400000
loop:
    add t0, t2, t4
    sltu t2, t0, t2
    add t1, t3, t5
    add t1, t1, t2
    add t3, x0, t5
    add t2, x0, t4
    add t4, x0, t0
    add t5, x0, t1
    addi t6, t6, -1
    bne t6, x0, loop
    nop
    sw t0, 0(a0)
    sw t1, 4(a0)
    jr ra
    nop
```

实验结果：



代码分析

终端程序（Term）

1. term.py
    - Main/ 欢迎语句，进入循环
    - MainLoop/ 用户交互：支持八种指令，每次调用指令后把数据转为二进制传入串口
    - run_A/ 代码转为二进制写入串口
    - run_F/ 文件读入版本的 A 指令
    - run_R/ 顺序输出寄存器内容
    - run_D/ 输出指定位置及长度的内存内容
    - run_U/ 调用反汇编程序并输出结果
    - run_G/ 运行程序，接收监控程序信息，负责计时、终止程序等操作

监控程序（Kernel）

1. kernel.asm
    - 记录不同函数间的相互调用位置及方式
2. evec.S
    - 程序入口位置，跳转到 init.S/START
3. init.S

    主要负责各种信息的初始化。
    - 初始化 bss 段

- 寄存器初始化（如 mtvec, sp, fp 等）
- 配置各种串口
- 页表空间映射
- 最后输出启动成功的信息，等待用户交互（跳转至 shell.S/SHELL）

4. shell.S

   负责用户指令交互。

   - 读取用户指令，配合 util.S 的函数读写串口
   - R: 读出用户寄存器内容并写入串口
   - D: 读取特定位置和长度的内存，循环遍历输出
   - A: 往特定内存位置写入二进制代码
   - G: 执行内存某处的代码，同时向终端通信、开始计时、保存用户寄存器内容，若超时则向终端发出终止信号，当程序终止或完成后恢复指针数据。
   - T: 打印页表内容
   - 指令操作完成后跳转至 SHELL，等待用户输入新指令

5. utils.S

   负责串口的写入和输出。

   - 检查串口状态（是否可以读写）
   - 提供不同的串口写入方式及输出方式
   - 建基于 WRITE_SERIAL 和 READ_SERIAL

6. test.S

   用于性能测试。

   - 无冲突数据测试，用于性能标定
   - 冲突数据测试，用于测定效率
   - 指令冲突测试
   - 访存冲突测试

思考题目

> 1. 比较 RISC-V 指令寻址方法与 x86 指令寻址方法的异同。

同：两者在任意指令下都可以直接访问寄存器的的数据。在使用特定指令的情况下，两者都可以利用立即数寻址、寄存器寻址、间接寻址、基址(及偏移量)寻址：

异：RISC-V 不能直接操作内存数据，需要使用 load/store 系列指令把内存数据写入寄存器中再使用，而 x86 则可以在任意指令中直接访问内存数据；在此基础上，RISC-V 支持 PC 相对寻址，x86 还支持变址寻址以及比例变址寻址的方式。

> 2. 阅读监控程序，列出监控程序的 19 条指令，请根据自己的理解对用到的指令进行分类，并说明分类原因。

19 条指令如下所示：

- 算术逻辑指令: ADD, ADDI, AND, ANDI, OR, ORI, XOR, SLLI, SRLI
- 内存读写指令: LB, LW, SB, SW
- 跳转指令: BEQ, BNE, JAL, JALR
- 长型立即数指令: LUI, AUIPC

这样分类的原因在于同组指令具有相似的功能，因此采用这一分组方式是恰当的。

> 3. 结合 term 源代码和 kernel 源代码说明 term 是如何实现用户程序计时的。

在 term.py 的 run_G 函数中可见，当监控程序给出 '0x06' 表示程序开始计时，给出 '0x07' 代表程序正常中止。与此同时，我们也可以从 kernel/common.h 中清楚看到 '0x06' 和 '0x07' 分别代表启动、停止计时。根据两信号间的时间差即可计算用户程序运行时间。

> 4. 说明 kernel 是如何使用串口的。

由 kern/utils.S 的 WRITE_SERIAL 和 READ_SERIAL 函数可知，kernel 会根据状态寄存器 (0x10000005, 由 serial.h 中定义) 的指示，读取数据寄存器 (0x10000000, 同上) 一字节的数据或向数据寄存器写入一字节的数据。

> 5. 请问 term 如何检查 kernel 已经正确连入，并分别指出检查代码在 term 与 kernel 源码中的位置。

kernel 进入主线程后，首先通过 init.S 第 389 行调用 init.S/monitor_version (第 90 行) 把字符串 "MONITOR for RISC-V - initialized.V" (33 个字符) 写入串口，再由第 390 行调用 utils.S/WRITE_SERIAL_STRING 输出。

对于 term，在完成连接后跳转至 Main 函数 (第 441 行)，然后在第 444/445 行等待 kernel 传来长(至少)为 33 个字符的信息后会回传一个 'W' (第 449 行) 表示 kernel 已经正确连入。