数字逻辑与处理器基础
MIPS 汇编编程实验

无 81 马啸阳 2018011054

2020 年 4 月 30 日

1 实验一

1.1 系统调用

实验结果如图 1所示，其中 a.in 中所存整数为 42，输入 50，输出 92 并写入 a.out。

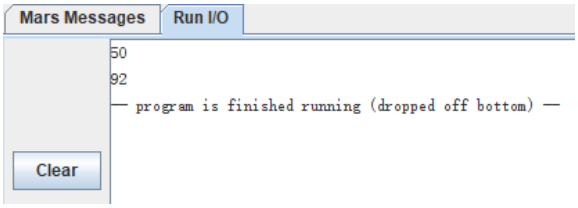

图 1: 1-1 系统调用实验结果

1.2 循环分支

对于 if-then-else 的一般性结构，应首先计算条件表达式的值，可利用 slt、seq 等比较指令，并结合分支跳转指令（beq 等）实现跳转，当条件表达式不成立时，跳转到 else，而条件表达式成立时，执行完 then 部分后跳转出条件分支。具体代码如下所示，其中条件表达式 s0==0 为例。

```
# some other instructions to compute condition
bne $s0, $0, else    # jump to else if $s0 != 0

# ... then statements
j exit               # jump through else statements
```

```
6
7  else:
8  # ... else statements
9
10 exit:
11 # ... statements after if-then-else
```

对于 while 循环的一般性结构，需要标记循环起点以跳转，然后先计算条件表达式，若不满足则跳出，然后执行循环体，循环体末尾跳回第一句指令。具体代码如下所示，其中条件表达式 s0==0 为例。

```
1  loop:
2  # some other instructions to compute condition
3  bne $s0, $0, exit    # exit if $s0 != 0
4  # ... statements inside while
5  j loop
6
7  exit:
8  # ... statements after while
```

对于 while 而言，其 continue 和 break 利用循环和退出两个标签即可实现。以前述代码中标签为例，continue 和 else 分别如下所示。

```
1  j loop   # continue statement
2  j exit   # break statement
```

do-while 循环与 while 循环类似，但在循环体最后才计算条件表达式，若成立则跳回第一句指令，结构更简单。

```
1  loop:
2  # ... statements inside do-while
3  # some other instructions to compute condition
4  beq $s0, $0, loop    # loop again if $s0 == 0
5
6  # ... statements after do-while
```

实验结果如图 2所示，这是使用辗转相减法计算输入两个整数的最大公约数，测试输入 24、30，输出 6（其它诸如第一个或第二个整数输入 0 的特殊情况也进行了测试，此处不列）。

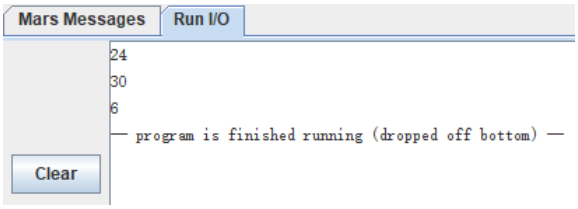

图 2: 1-2 循环分支实验结果

1.3 数组、指针

```
int A [2];       // &A=0x10010000
A[0] = 0x00000012;
A[1] = 0x00000021;
int* p_A = &A;
```

上述代码作为条件时，以下表达式值列出如下。

```
A[0]=0x00000012
A[1]=0x00000021
p_A[0]=0x00000012
p_A[1]=0x00000021
(int)p_A=0x10010000
(int)(p_A+1)=0x10010004
*(int*)((int)p_A+4)=0x00000021
```

实验结果如图 3所示，输入 n，建立一个长为 n 的数组和一个长为 n 的链表，均存储 0 至 n-1，随后分别输出，故输出两次 0 至 n-1。

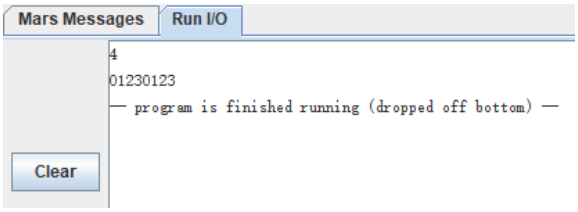

图 3: 1-3 数组、指针实验结果

1.4 函数调用

补全 sum 代码如下，$a0 传入 n 可在 $v0 返回 $\sum_{i=1}^{n} i$。

```
sum:
# save registers
addi $sp $sp -8
sw $ra 4($sp)
sw $s0 0($sp)

addi $s0 $a0 0      # $s0 = a, stores b
addi $t0 $0  0      # $t0 = 0, stores temp0
beq $s0 $zero skip  # skip if b == 0
addi $t1 $s0 -1     # $t1 = b-1, stores temp1

# call sum(temp1)
addi $a0 $t1 0      # $a0 = temp1
jal sum
addi $t0 $v0 0      # temp0 = sum(temp1)

skip:
add $s0 $s0 $t0     # b = b + temp0
addi $v0 $s0 0      # move b to $v0, return b

# restore registers
lw $ra 4($sp)
lw $s0 0($sp)
addi $sp $sp 8

jr $ra
```

2 实验二

在实验一的基础下，实验二中代码只需根据先前所述的 if-then-else、while、函数调用一般性结构对应编译即可，具体详见代码注释。三个排序代码主程序部分都是如 1.1 的文件读写，将数组读至事先开辟的 data 区的 buffer，然后调用排序函数。排序部分中快速排序和归并排序要使用递归调用，与 1.4 类似。归并排序中链表与指针操作与 1.3 类似。

调试中未出现明显问题，将代码按照逻辑拆分成主程序、排序以及一些子过程分别编写调试，以减轻调试的工作量。由于本次 c++ 代码中，比较接近汇编的执行方式，因而编译较为容易，出现的错误大多在于寄存器标号写错或者条件分支判断反了，手动记录下寄存器存储变

量的表格会方便编写。

其它一些注意到的问题如下。首先是大小端问题，注意到读写文件时，使用系统调用，因而都是小端法读写，而 MIPS 程序执行时寄存器以及算术运算都是大端法，但字符串的存储（a.in 与 a.out）似乎也是小端法？然而具体大小端法的使用只需由模拟器掌管，编写汇编代码时实际不必关心。

另一方面，对于内存空间开辟，实际操作系统分配内存空间机制更加复杂（例如现代操作系统中的页式分配等），即便是在堆栈上分配，都需要专门的数据结构来管理内存分配。本次代码中，并未进行存储管理，而只是在 data 区或在栈上分配 new 的空间，这导致最后无法 delete。一方面是因为未记录分配长度，另一方面，如果释放的是处在栈中间（而非栈顶）的空间，则栈指针仍然无法移动，从而无法释放空间。

最后，汇编代码的优化是以可读性和开发难度为代价的，例如尾递归和函数调用时需要保存的寄存器数量，都可以仔细计算考量，但考虑到调试的方便，本次代码中我只进行了部分最简单的优化。

本实验中，对于 N=1000，冒泡排序远慢于归并排序和快速排序。将排序结果与 c++ 代码运行结果比对，结果如图 4。

```
→  experiment_2 git:(master) ✗ fc.exe /B a.out bsort_a.out
正在比较文件 a.out 和 BSORT_A.OUT
FC: 找不到差异

→  experiment_2 git:(master) ✗ fc.exe /B a.out qsort_a.out
正在比较文件 a.out 和 QSORT_A.OUT
FC: 找不到差异

→  experiment_2 git:(master) ✗ fc.exe /B a.out msort_a.out
正在比较文件 a.out 和 MSORT_A.OUT
FC: 找不到差异
```

图 4: 排序实验结果