

东南大学

《数字逻辑与计算机体系结构（含实验）》

实验报告

实验五 计算机系统与指令认识

姓 名：赵舞穹

学 号：61520522

同 组：郑瑞琪

学 号：61520523

专 业：工科试验班

实 验 室：计算机硬件技术

实验时间：2021 年 11 月 26 日

报告时间：2021 年 12 月 6 日

评定成绩：

评阅教师：冯熳

目录

1	实验目的	3
2	实验内容	3
1	计算机系统环境与命令行认识	3
2	QtSpim 模拟器与 MIPS 指令验证	5
1	Spim 基本测试	5
2	QtSpim 编译 HelloWorld	6
3	选择排序的 MIPS 实现	10
3	在线编译 MIPS	15
4	通过 DOSBox 了解 X86 指令	15
3	实验创新与提高	18
4	实验总结	18
	参考文献	19
	附录 A：实验报告 L^AT_EX 模板	19
	附录 B：程序真伪判别	20

一. 实验目的

1. 掌握学习通过模拟调试工具软件认识理解计算机基本资源及动态调试程序的概念；
2. 学习掌握计算机命令行操作方法, 了解系统调用的概念；
3. 学习掌握利用 QtSpim 认识和调试 Mips 指令，加深指令理解认识；
4. 了解计算机命令行操作，认识 Debug/TD 调试器及 X86 指令调试，加深指令理解认识。^[1]

二. 实验内容

(一) 计算机系统环境与命令行认识

我对于命令行认识相对较多，平时很多工作也都是在控制台操作的。我平时使用的操作系统包括了 MacOS, Linux (Ubuntu) 和 Windows. 在 MacOS 和 Linux 下很多命令是一样的，使用起来感觉就很流畅。Windows 下的 CMD 使用不是很熟练，一般使用其 PowerShell.

新建文件使用 `touch`，修改文件名或者移动 `mv`，删除 `rm`，编辑可以用 `nano` 或者 `vim`. 控制台运行的效果如图 1 所示.

下面给出我使用 C++ 的方法：我目前直接使用 `clang++` 编译，而这样实际非常方便，正如下面的代码给出的这样¹.

compile_rp3d_clang.sh

```
1 #!/bin/sh
2
3 # compile_rp3d_clang.sh
4 #
5 # Compile ReactPhysics3d into object file using clang++.
6 # Object files are in folder 'obj'.
7 # Warnings are disabled.
8 #
9 # Teddy van Jerry
10 # 2021/09/30
11
12 # create dirs
13 mkdir obj
14 cd obj
15 mkdir body
16 mkdir collision
17 mkdir collision/broadphase
18 mkdir collision/narrowphase
```

¹完整的项目 Dice Simulation 详见我的 GitHub 项目：https://github.com/Teddy-van-Jerry/Dice_Simulation.

```
19 mkdir collision/narrowphase/GJK
20 mkdir collision/narrowphase/SAT
21 mkdir collision/shapes
22 mkdir constraint
23 mkdir engine
24 mkdir systems
25 mkdir components
26 mkdir mathematics
27 mkdir memory
28 mkdir utils
29 cd ..
30
31 # compile object files
32 clang++ -w -c "../ext/reactphysics3d/src/body/RigidBody.cpp" -std=c++11
    -I ../ext/reactphysics3d/include -o "obj/body/RigidBody.o"
33 # ...
34 # Similar commands are omitted here for the sake of space.
35 # ...
36
37 # Compile the HelloWorld example
38 cd ../src/cpp/HelloWorld
39
40 # Create object file of Main.cpp
41 clang++ -c -std=c++11 Main.cpp -I ../../../../ext/reactphysics3d/include -o
    Main.o
42
43 # Create executable
44 clang++ -std=c++11 \
45 Main.o \
46 "../../../../usr/obj/body/CollisionBody.o" \
47 "../../../../usr/obj/body/RigidBody.o" \
48 # ...
49 # Similar commands are omitted here for the sake of space.
50 # ...
51 "../../../../usr/obj/utils/DebugRenderer.o" \
52 -o HelloWorld
53
54 # Remove object file
55 rm Main.o
```

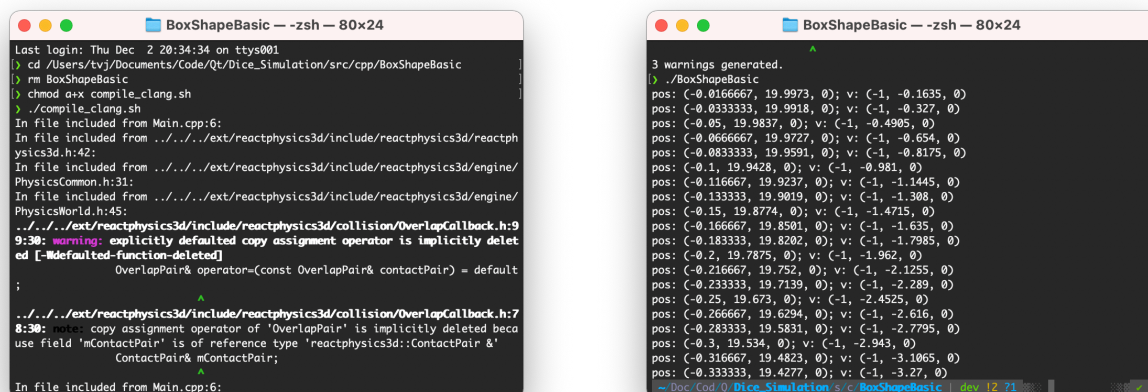


图 1: MacOS 控制台编译 C++ 输出

(二) QtSpim 模拟器与 MIPS 指令验证

1. Spim 基本测试

我在 Ubuntu 平台上，首先安装了无桌面的 Spim 进行测试，命令如下：

Install Spim on Ubuntu

```
sudo apt install spim # install the command-line version of spim
```

Terminal 如图 3 所示. 首先新建文件 `spim_test.m`，然后用 Nano 编辑内容，其内容如图 2 所示.

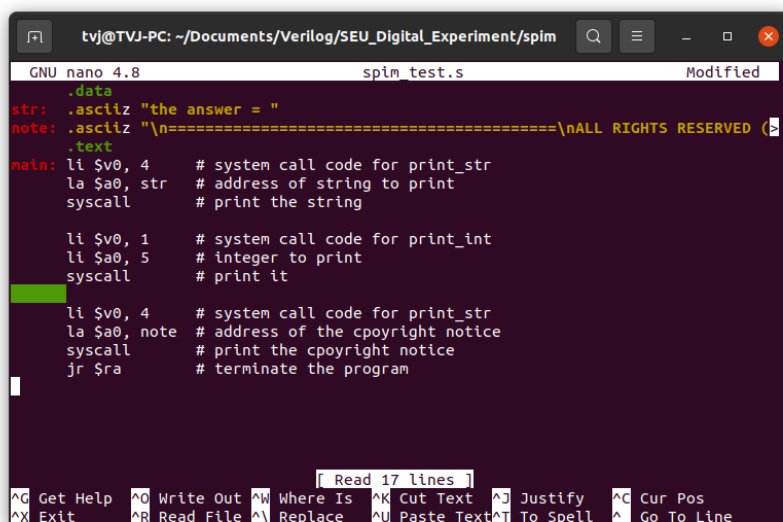


图 2: 测试代码 (Nano 窗口)

然后用 `spim` 命令进入编译器，首先导入 MIPS 文件 (`read` 或者 `load`)，`step` 命令单步运行 (其后加数字是选择步数)，期间可以用 `print` 命令查看寄存器内变量的值，可以看出 `$v0` 对值发生了变化. 直接继续使用 `continue` 运行可以看到后续的输出. 结束之后再使用 `run` 命令从指定位置开始运行，得到完整的结果. 再开始其他工作之前需要 `reinitialize`，否则即使是 `read` 了同一个文件也会出现重

```

tvj@TVJ-PC: ~/Documents/Verilog/SEU_Digital_Experiment/spim
tvj@TVJ-PC:~/Documents/Verilog/SEU_Digital_Experiment/spim$ # 61520522 Wuqiong Zhao
tvj@TVJ-PC:~/Documents/Verilog/SEU_Digital_Experiment/spim$ touch spin_test.s
tvj@TVJ-PC:~/Documents/Verilog/SEU_Digital_Experiment/spim$ nano spin_test.s
tvj@TVJ-PC:~/Documents/Verilog/SEU_Digital_Experiment/spim$ spim
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
(spim) read "spin_test.s"
(spim) step 5
[0x00400000] 0x8fa40000 lw $4, 0($29) ; 183: lw $a0 0($sp) # argc
[0x00400004] 0x27a50004 addiu $5, $29, 4 ; 184: addiu $a1 $sp 4 # argv
[0x00400008] 0x24a60004 addiu $6, $5, 4 ; 185: addiu $a2 $a1 4 # envp
[0x0040000c] 0x00041080 sll $2, $4, 2 ; 186: sll $v0 $a0 2
[0x00400010] 0x00c23021 addu $6, $6, $2 ; 187: addu $a2 $a2 $v0
(spim) step 3
[0x00400014] 0x0c100009 jal 0x00400024 [main] ; 188: jal main
[0x00400024] 0x34020004 ori $2, $0, 4 ; 5: li $v0, 4 # system call code for print_str
[0x00400028] 0x3c041001 lui $4, 4097 [str] ; 6: la $a0, str # address of string to print
(spim) print $a0
Reg 4 = 0x10010000 (268500992)
(spim) step 2
[0x0040002c] 0x0000000c syscall ; 7: syscall # print the string
the answer = [0x00400030] 0x34020001 ori $2, $0, 1 ; 9: li $v0, 1 # system call code for print_int
(spim) step
[0x00400034] 0x34040005 ori $4, $0, 5 ; 10: li $a0, 5 # integer to print
(spim) print $a0
Reg 4 = 0x00000005 (5)
(spim) continue
5
=====
ALL RIGHTS RESERVED (C) 2021 Wuqiong Zhao
(spim) run 0x00400000
the answer = 5
=====
ALL RIGHTS RESERVED (C) 2021 Wuqiong Zhao
(spim) reinitialize

```

图 3: 使用控制台 Spim 直接编译

复定义现象.

注意 1: 报错: attempt to execute non-instruction at 0x0040004c

此处最后需要加上 `jr $ra`, 否则 MIPS 程序无法终止.^a

^a参考: <https://stackoverflow.com/questions/20172655/attempt-to-execute-non-instruction-in-mips-assembler>.

2. QtSpim 编译 HelloWorld

QtSpim 的界面如图 4 所示, Console 输出如图 5 所示, 其内容与命令行版的 Spim 没有太大区别, 只是在显示结果时更加方便. Mac 版本的 QtSpim 存在 bug, 使用起来还不如直接命令行.

HelloWorld.s

```

1      .data
2 msg:  .ascii "Hello World!" # string for display
3      .extern foobar 4      # external parameter (exit)
4
5      .text
6      .globl main
7 main:  li $v0, 4           # syscall 4 (print_str)
8      la $a0, msg          # argument: string

```

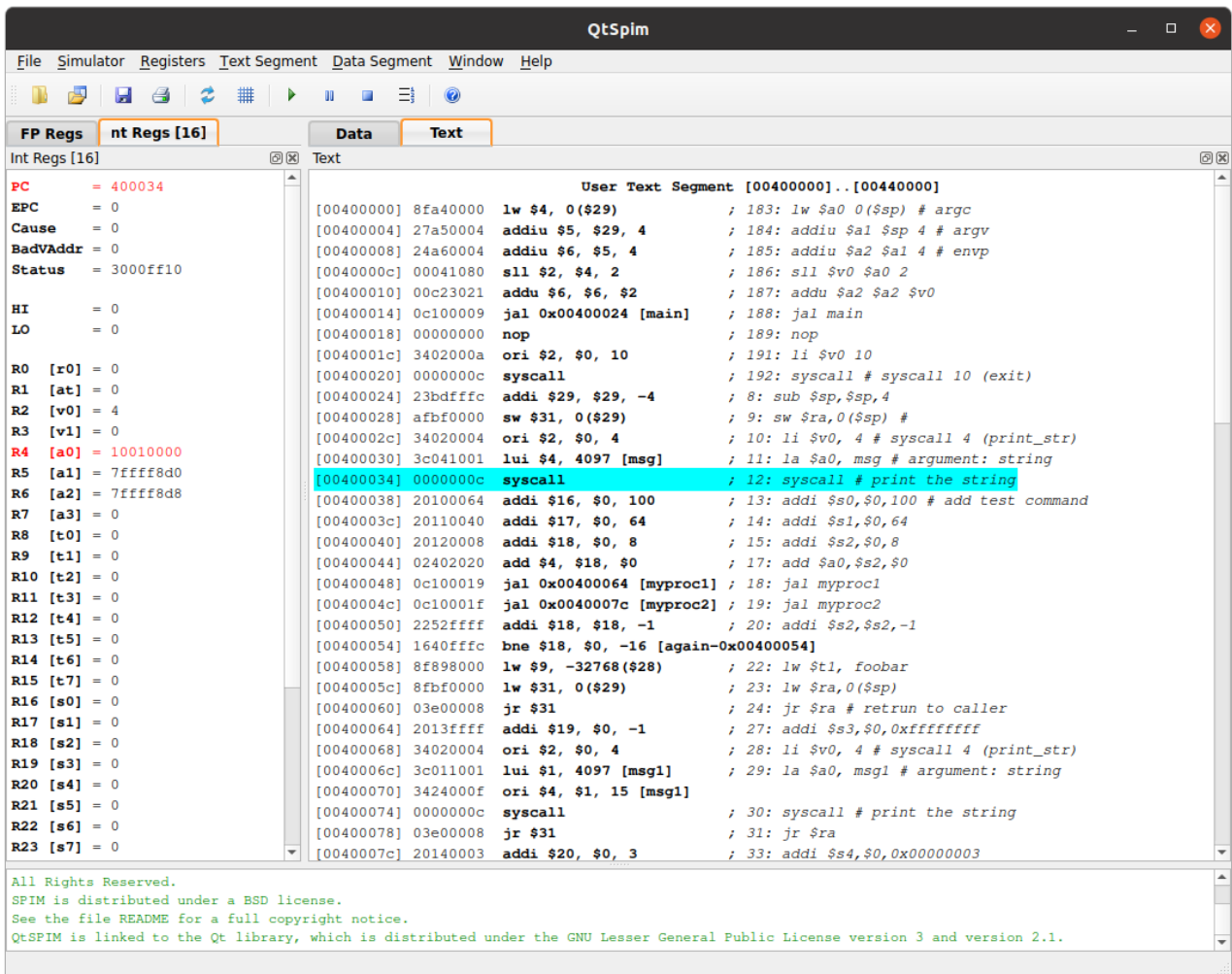


图 4: QtSpim 界面（正在单步调试）

```

9      syscall                # print the string
10     lw $t1, foobar
11
12     jr $ra                # return to caller

```

HelloWorld2.s

```

1      .data
2  msg:  .ascii "Hello World!\n\n" # string for display
3  msg1:  .ascii "Hello Friend!\n\n" # string for display
4      .extern foobar 4           # external parameter (exit)
5
6      .text
7      .globl main
8  main: sub $sp,$sp,4
9      sw $ra,0($sp) #

```

```
10      li $v0, 4          # syscall 4 (print_str)
11      la $a0, msg        # argument: string
12      syscall            # print the string
13      addi $s0,$0,100    # add test command
14      addi $s1,$0,64
15      addi $s2,$0,8
16 again:
17      add $a0,$s2,$0
18      jal myproc1
19      jal myproc2
20      addi $s2,$s2,-1
21      bne $s2,$0,again
22      lw $t1, foobar
23      lw $ra,0($sp)
24      jr $ra             # return to caller
25 myproc1:
26      addi $s3,$0,0xffffffff
27      li $v0, 4          # syscall 4 (print_str)
28      la $a0, msg1       # argument: string
29      syscall            # print the string
30      jr $ra
31 myproc2:
32      addi $s4,$0,0x00000003
33      jr $ra
```

HelloWorld3.s

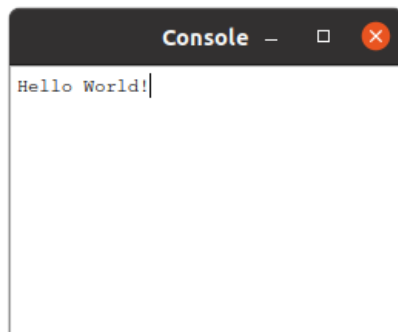
```
1      .data
2 msg:   .ascii "Hello World!\n\n" # string for display
3 msg1:  .ascii "Hello Friend!\n\n" # string for display
4      .extern foobar 4           # external parameter (exit)
5 vb1:   .word 0x12345678
6 vb2:   .word 0xabcdef12
7
8      .text
9      .globl main
10 main: sub $sp,$sp,4
11      sw $ra,0($sp) #
12      li $v0, 4      # syscall 4 (print_str)
13      la $a0, msg    # argument: string
14      syscall        # print the string
```



```

15     la $s4,vb1
16     lbu $s5,2($s4)
17     la $s6,vb2
18     lb $s7,1($s6)
19     sb $s5,1($s6)
20     sb $s7,2($s4)
21     addi $s0,$0,100 # add test command
22     addi $s1,$0,64
23     addi $s2,$0,8
24 again:
25     add $a0,$s2,$0
26     jal myproc1
27     jal myproc2
28     addi $s2,$s2,-1
29     bne $s2,$0,again
30     lw $t1, foobar
31     lw $ra,0($sp)
32     jr $ra          # return to caller
33 myproc1:
34     addi $s3,$0,0xffffffff
35     li $v0, 4        # syscall 4 (print_str)
36     la $a0, msg1     # argument: string
37     syscall          # print the string
38     jr $ra
39 myproc2:
40     addi $s4,$0,0x00000003
41     jr $ra

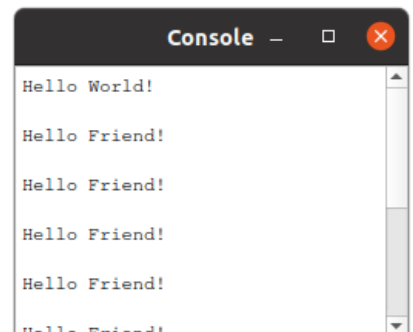
```



(a) HelloWorld



(b) HelloWorld2



(c) HelloWorld3

图 5: HelloWorld 的 Console 输出

3. 选择排序的 MIPS 实现

使用 MacPort 安装控制台版的 Spim，命令如下：

Install Spim on MacOS with MacPort

```
1 sudo port install spim # install the command-line version of spim
```

通过学习网上的代码，修改相应内容（因为原来的代码是无法运行的，例如 `subi` 等命令，与 Spim 的要求不完全匹配），整理设计，得到了如下的代码：

sort.s

```
1      .text
2      j      main          # Jump to main-routine
3
4      .data
5  str1:  .ascii "Insert the array size: "
6  str2:  .ascii "Insert the array elements, one per line: \n"
7  str3:  .ascii "The sorted array is : \n"
8  str5:  .ascii "\n"
9
10     .text
11     .globl main
12  main:
13     la     $a0, str1      # Print of str1
14     li     $v0, 4         #
15     syscall              #
16
17     li     $v0, 5         # Get the array size(n) and
18     syscall              # and put it in $v0
19     move   $s2, $v0       # $s2=n
20     sll    $s0, $v0, 2    # $s0=n*4
21     sub    $sp, $sp, $s0  # This instruction creates a stack
22                               # frame large enough to contain
23                               # the array
24     la     $a0, str2      #
25     li     $v0, 4         # Print of str2
26     syscall              #
27
28     move   $s1, $zero     # i=0
29
30  for_get:
31     bge    $s1, $s2, exit_get # if i>=n go to exit_for_get
```

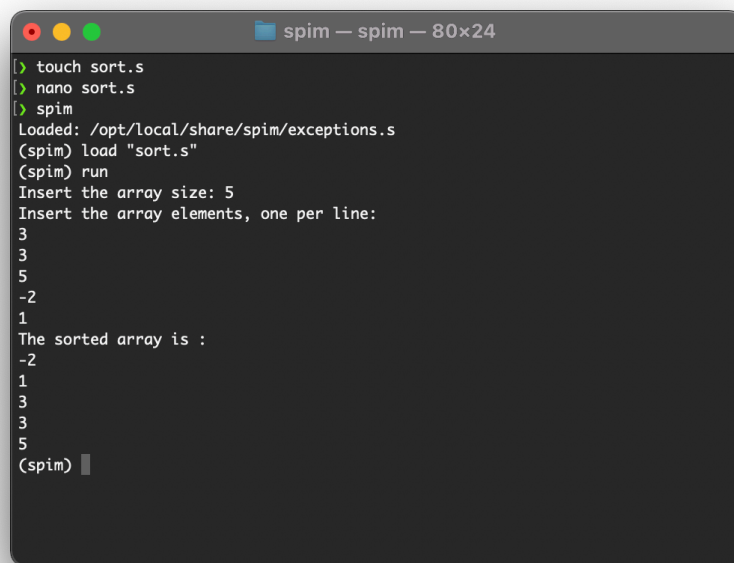
```
32      sll      $t0, $s1, 2      # $t0=i*4
33      add      $t1, $t0, $sp    # $t1=$sp+i*4
34      li       $v0, 5          # Get one element of the array
35      syscall                                #
36      sw       $v0, 0($t1)      # The element is stored
37                                # at the address $t1
38      addi     $s1, $s1, 1      # i=i+1
39      j        for_get
40
41 exit_get:
42      move     $a0, $sp         # $a0=base address af the array
43      move     $a1, $s2         # $a1=size of the array
44      jal      isort            # isort(a,n)
45                                # In this moment the array has been
46                                # sorted and is in the stack frame
47      la       $a0, str3        # Print of str3
48      li       $v0, 4           #
49      syscall
50      move     $s1, $zero       # i=0
51
52 for_print:
53      bge      $s1, $s2, exit_print # if i>=n go to exit_print
54      sll      $t0, $s1, 2      # $t0=i*4
55      add      $t1, $sp, $t0    # $t1=address of a[i]
56      lw       $a0, 0($t1)      #
57      li       $v0, 1           # print of the element a[i]
58      syscall                                #
59
60      la       $a0, str5
61      li       $v0, 4
62      syscall
63      addi     $s1, $s1, 1      # i=i+1
64      j        for_print
65
66 exit_print:
67      add      $sp, $sp, $s0     # elimination of the stack frame
68
69      li       $v0, 10          # EXIT
70      syscall
71
72 # selection_sort
```

```
73 isort:
74     addi    $sp, $sp, -20    # save values on stack
75     sw      $ra, 0($sp)
76     sw      $s0, 4($sp)
77     sw      $s1, 8($sp)
78     sw      $s2, 12($sp)
79     sw      $s3, 16($sp)
80
81     move    $s0, $a0        # base address of the array
82     move    $s1, $zero      # i=0
83
84     addi    $t0, $0, 1      # $t0 = 1
85     subu    $s2, $a1, $t0   # length -1
86
87 isort_for:
88     bge     $s1, $s2, isort_exit # if i >= length-1 -> exit loop
89
90     move    $a0, $s0        # base address
91     move    $a1, $s1        # i
92     move    $a2, $s2        # length - 1
93
94     jal     mini
95     move    $s3, $v0        # return value of mini
96
97     move    $a0, $s0        # array
98     move    $a1, $s1        # i
99     move    $a2, $s3        # mini
100
101     jal     swap
102
103     addi    $s1, $s1, 1     # i += 1
104     j       isort_for      # go back to the beginning of the loop
105
106 isort_exit:
107     lw      $ra, 0($sp)    # restore values from stack
108     lw      $s0, 4($sp)
109     lw      $s1, 8($sp)
110     lw      $s2, 12($sp)
111     lw      $s3, 16($sp)
112     addi    $sp, $sp, 20   # restore stack pointer
113     jr      $ra           # return
```

```
114
115
116 # index_minimum routine
117 mini:
118     move    $t0, $a0        # base of the array
119     move    $t1, $a1        # mini = first = i
120     move    $t2, $a2        # last
121
122     sll     $t3, $t1, 2     # first * 4
123     add     $t3, $t3, $t0   # index = base array + first * 4
124     lw      $t4, 0($t3)     # min = v[first]
125
126     addi    $t5, $t1, 1     # i = 0
127
128 mini_for:
129     bgt     $t5, $t2, mini_end # go to min_end
130
131     sll     $t6, $t5, 2     # i * 4
132     add     $t6, $t6, $t0   # index = base array + i * 4
133     lw      $t7, 0($t6)     # v[index]
134
135     bge     $t7, $t4, mini_if_exit # skip the if when v[i] >= min
136
137     move    $t1, $t5        # mini = i
138     move    $t4, $t7        # min = v[i]
139
140 mini_if_exit:
141     addi    $t5, $t5, 1     # i += 1
142     j       mini_for
143
144 mini_end:
145     move    $v0, $t1        # return mini
146     jr      $ra
147
148 # swap routine
149 swap:
150     sll     $t1, $a1, 2     # i * 4
151     add     $t1, $a0, $t1   # v + i * 4
152
153     sll     $t2, $a2, 2     # j * 4
154     add     $t2, $a0, $t2   # v + j * 4
```

```
155
156      lw      $t0, 0($t1)      # v[i]
157      lw      $t3, 0($t2)      # v[j]
158
159      sw      $t3, 0($t1)      # v[i] = v[j]
160      sw      $t0, 0($t2)      # v[j] = $t0
161
162      jr      $ra
```

测试结果如图 6 所示，成功完成排序。



```
[> touch sort.s
[> nano sort.s
[> spim
Loaded: /opt/local/share/spim/exceptions.s
(spim) load "sort.s"
(spim) run
Insert the array size: 5
Insert the array elements, one per line:
3
3
5
-2
1
The sorted array is :
-2
1
3
3
5
(spim)
```

图 6: sort.s 输出结果

结果分析 1: MIPS 编写心得

MIPS 的学习过程和 Unix、Linux 命令行的学习很像，就是一开始觉得十分繁琐，永远也记不下来的命令，但是在一定的锻炼之下，有了整体的脉络了解后，注重结构性和严谨性（不能像某些提供的代码一样，格式乱七八糟，缩紧不注意，这都是影响代码最后的可读性）才不容易写错。汇编写起来要求是高的，因为内存管理需要手动操作 C/C++ 用起来也相对较难是因为指针的封装也没有摆脱内存管理的工作（其实 C++ 有一定办法回避，例如引用和使用 std 标准库），对个人的提升是很高的。

(三) 在线编译 MIPS

我将 WeMips 的代码 fork 到了我自己新搭建的网站 <https://mips.teddy-van-jerry.org>，经过测试效果很好，网页如图 7 所示。

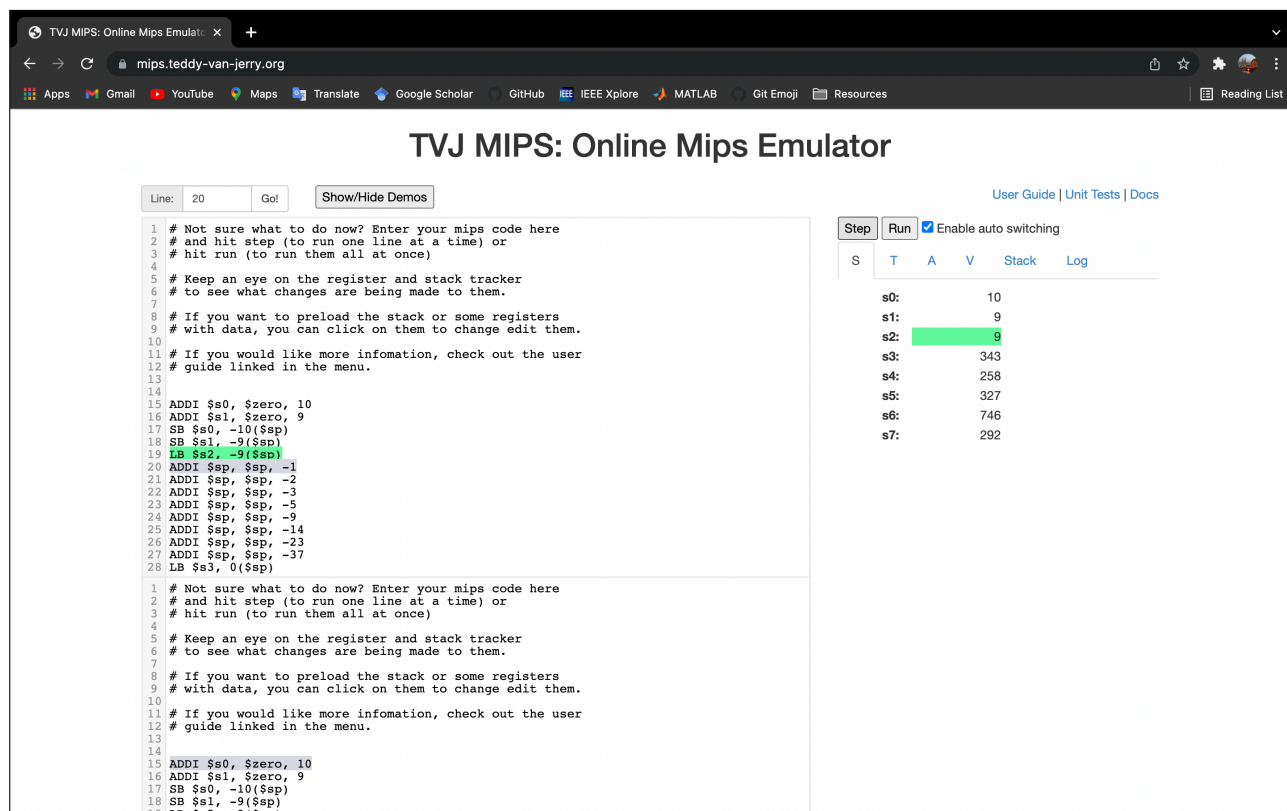


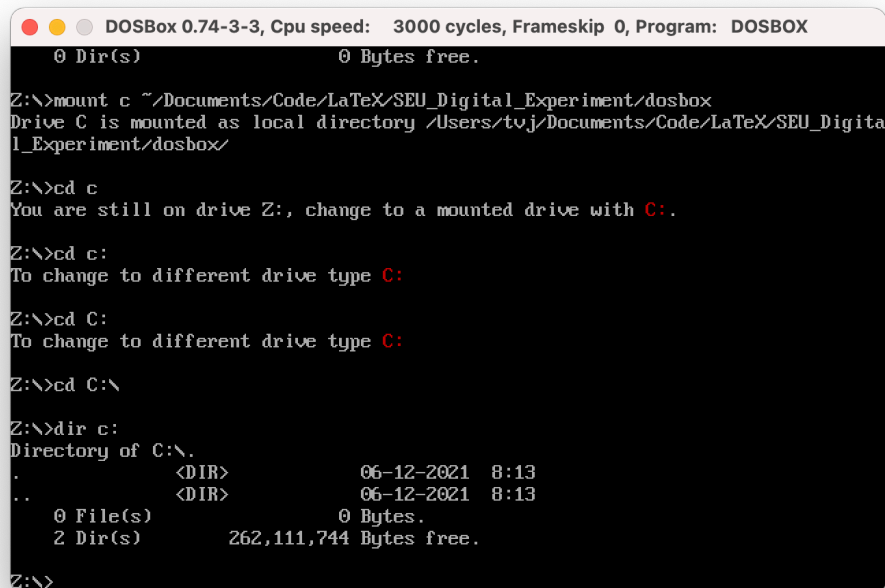
图 7: TVJ MIPS 在线 MIPS 编译

(四) 通过 DOSBox 了解 X86 指令

DOSBox 的实验我在 MacOS 上完成，指导书^[1]附带的 debug.exe 无法运行，我下载了 com 版本的文件²，在 MacOS 上使用顺利。

²资源: <https://www.japheth.de/Download/Debug/DEBUG125.zip>，包括源代码和 com 文件。

图 8 为配置 DOSBox 的虚拟挂载路径. 取消 mount 需要使用命令 `mount -u <PATH>`.



```
DOSBox 0.74-3-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
0 Dir(s)          0 Bytes free.

Z:\>mount c ~/Documents/Code/LaTeX/SEU_Digital_Experiment/dosbox
Drive C is mounted as local directory /Users/tvj/Documents/Code/LaTeX/SEU_Digital_Experiment/dosbox/

Z:\>cd c
You are still on drive Z:, change to a mounted drive with C:.

Z:\>cd c:
To change to different drive type C:

Z:\>cd C:
To change to different drive type C:

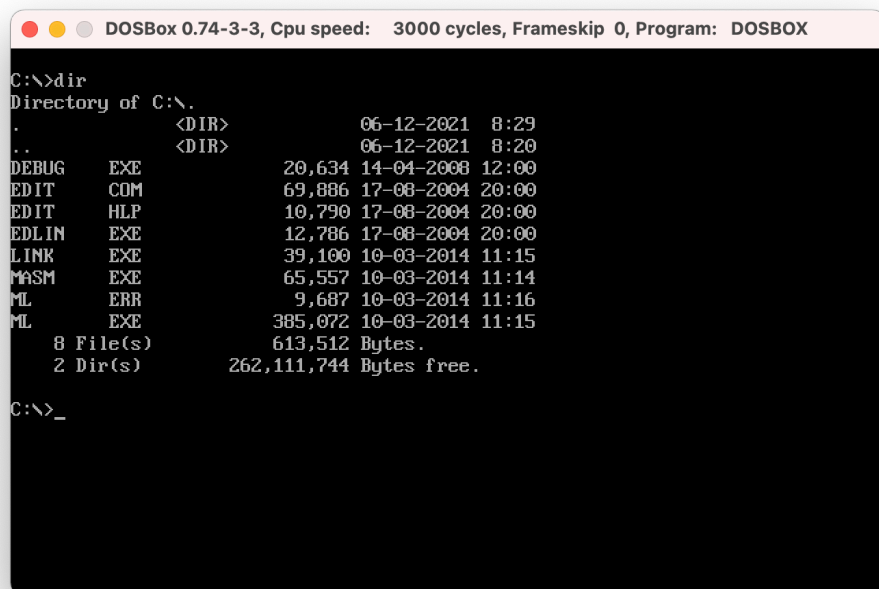
Z:\>cd C:\

Z:\>dir c:
Directory of C:\.
.                <DIR>                06-12-2021  8:13
..               <DIR>                06-12-2021  8:13
0 File(s)        0 Bytes.
2 Dir(s)         262,111,744 Bytes free.

Z:\>
```

图 8: DOSBox – mount

图 9 为通过 `dir` 命令查看文件夹下的文件和文件夹.



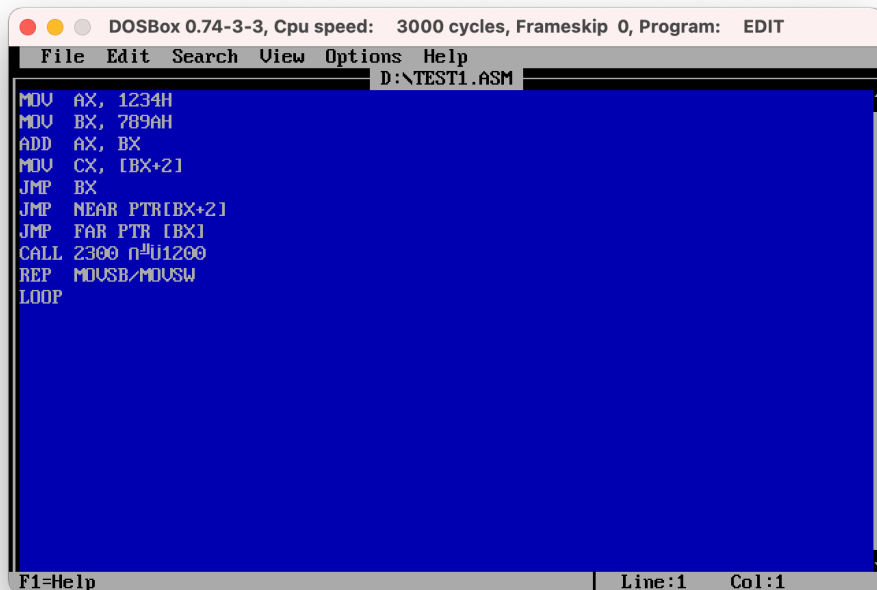
```
DOSBox 0.74-3-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX

C:\>dir
Directory of C:\.
.                <DIR>                06-12-2021  8:29
..               <DIR>                06-12-2021  8:20
DEBUG    EXE          20,634 14-04-2008 12:00
EDIT     COM          69,886 17-08-2004 20:00
EDIT     HLP          10,790 17-08-2004 20:00
EDLIN    EXE          12,786 17-08-2004 20:00
LINK     EXE          39,100 10-03-2014 11:15
MASM     EXE          65,557 10-03-2014 11:14
ML       ERR           9,687 10-03-2014 11:16
ML       EXE          385,072 10-03-2014 11:15
8 File(s)    613,512 Bytes.
2 Dir(s)     262,111,744 Bytes free.

C:\>_
```

图 9: DOSBox – dir

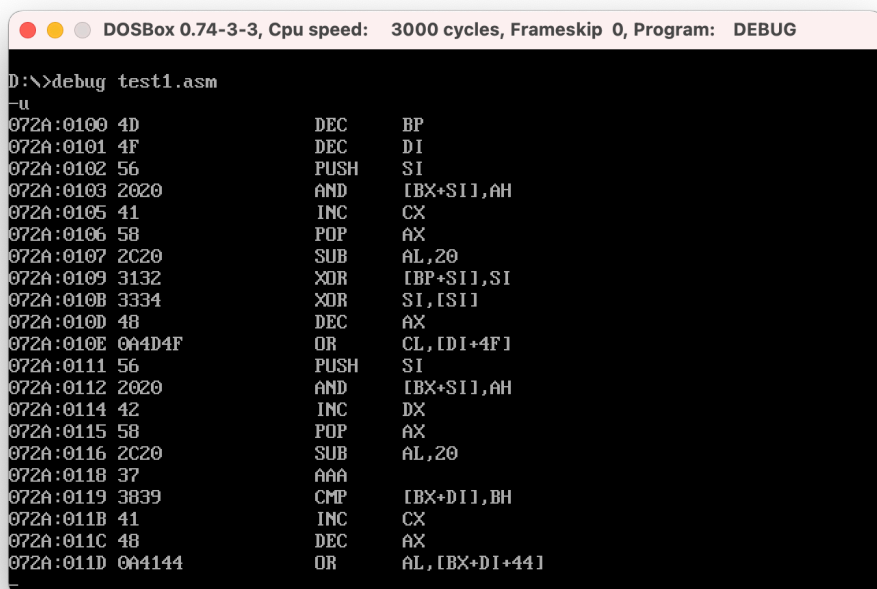
图 10 使用 edit.com 编辑 test1.asm，测试基本的 x86 命令.



```
DOSBox 0.74-3-3, Cpu speed: 3000 cycles, Frameskip 0, Program: EDIT
File Edit Search View Options Help
D:\TEST1.ASM
MOV AX, 1234H
MOV BX, 789AH
ADD AX, BX
MOV CX, [BX+2]
JMP BX
JMP NEAR PTR[BX+2]
JMP FAR PTR [BX]
CALL 2300 n^u1200
REP MOUSB/MOSW
LOOP
F1=Help Line:1 Col:1
```

图 10: DOSBox – edit

图 11 用 debug.com 运行了刚刚编辑的汇编代码，效果不错.



```
DOSBox 0.74-3-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG
D:\>debug test1.asm
-u
072A:0100 4D      DEC     BP
072A:0101 4F      DEC     DI
072A:0102 56      PUSH    SI
072A:0103 2020    AND     [BX+SI],AH
072A:0105 41      INC     CX
072A:0106 58      POP     AX
072A:0107 2C20    SUB     AL,20
072A:0109 3132    XOR     [BP+SI],SI
072A:010B 3334    XOR     SI,[SI]
072A:010D 48      DEC     AX
072A:010E 0A4D4F OR      CL,[DI+4F]
072A:0111 56      PUSH    SI
072A:0112 2020    AND     [BX+SI],AH
072A:0114 42      INC     DX
072A:0115 58      POP     AX
072A:0116 2C20    SUB     AL,20
072A:0118 37      AAA
072A:0119 3839    CMP     [BX+DI],BH
072A:011B 41      INC     CX
072A:011C 48      DEC     AX
072A:011D 0A4144 OR      AL,[BX+DI+44]
Line:1 Col:1
```

图 11: DOSBox – debug

图 12 真的成功让我的 Mac 持续发出固定频率的声音，验证了 DOSBox 在 Mac 上适配性较好. (这么古老的软件居然可以挺让我震惊的.)

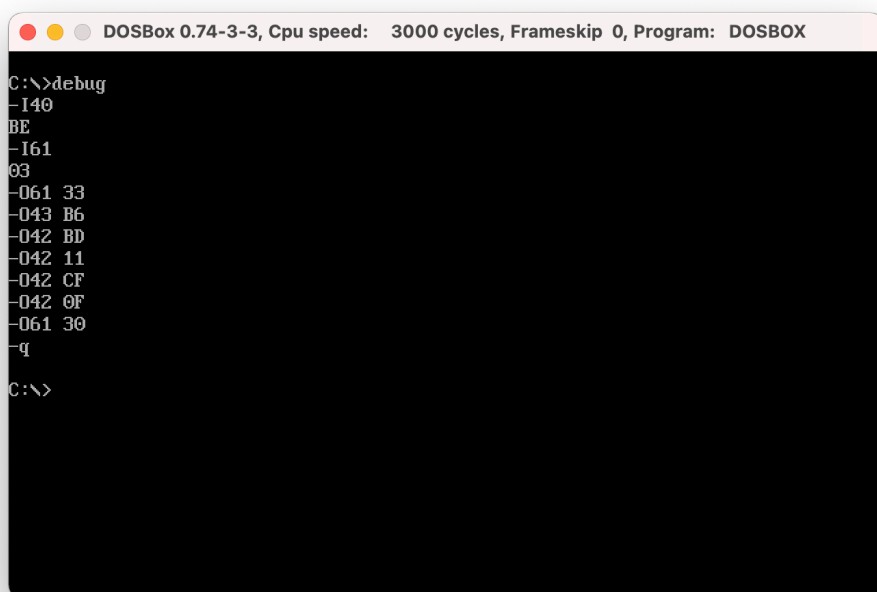


图 12: DOSBox – debug (扬声器)

结果分析 2: DOSBox 使用心得

有的时候使用这种古老的东西很有那种味道，就像我现在比较喜欢用命令行做一些事情一样. 能够看到古董软件在现代平台上正常运行真实感到不错呢.

三. 实验创新与提高

- 多平台使用 QtSpim (Windows, Linux, MacOS), 并且使用了命令行版的 Spim;
- 编写排序算法, 对于汇编的整体架构和使用有了更为深入的了解;
- 在自己的网站上搭建了 MIPS 编译器;
- DOSBox 中积极解决在 MacOS 上运行的问题, 并操作了如 `mount -u` 等命令.

四. 实验总结

关于 MIPS 的总结详见结果分析 1, 关于 DOSBox 的总结详见结果分析 2. 总体来说, 由于我平时有一些命令行的相关了解, 也在 Unix, Linux 平台上做一些工作, 整体的实验比较顺利且有趣. 这次实验也将更广阔更底层的东西展示给了我, 拓展了我的视野.

实验器材 1

- QtSpim 9.1.21/22
- Spim 8.0 (For Ubuntu), Spim 9.1.22 (For MacOS) （无桌面版本）
- DOSBox 0.74-3-3
- Ubuntu 20 (X86_64) / Windows 11 (X86_64) / MacOS Big Sur (M1 chip) (QtSpim 实验于 Ubuntu 和 MacOS 上完成, DOSBox 于 MacOS 上完成)
- Debian GNU/Linux 10 x86_64 (TVJ MIPS 的服务器)

参考文献

- [1] 计算机硬件实验室. 《数字逻辑与计算机体系结构》实验指导书[A]. 南京: 东南大学, 2021.

附录 A：实验报告 L^AT_EX 模板

实验报告使用自己编写的 L^AT_EX 模板 (SEU-Digital-Report.cls), 在基本适配 Microsoft Word 版报告的格式要求之外, 增加了更多的功能, 使得报告看起来更加优雅多彩.

后续升级后, 报告模板将于 <https://github.com/Teddy-van-Jerry/TVJ-Digital-Report> 基于 MIT License 开源共享.

编译需要使用 XeLaTeX + Biber, 封面页修改如下内容即可.

```

1  %% 使用实验报告模板类（字体大小 11pt 约为五号字）
2  \documentclass[11pt]{SEU-Digital-Report}
3
4  %%%%%%%%%%% 报告基本信息 %%%%%%%%%%%
5  \expno{五} % 实验序号
6  \expname{计算机系统与指令认识} % 实验名称
7  \expauthor{赵舞穹} % 姓名
8  \expID{61520522} % 学号
9  \expmates{郑瑞琪} % 同组
10 \expmatesID{61520523} % 学号（同组）
11 \expmajor{工科试验班} % 专业
12 \explab{计算机硬件技术} % 实验室
13 \expdate{2021年11月26日} % 实验日期
14 \expreportdate{\today} % 报告日期
15 \expgrade{} % 成绩评定
16 \exptutor{冯熳} % 评阅教师
17 %%%%%%%%%%%

```

附录 B：程序真伪判别

1. QtSpim 我在报告中使用 Ubuntu 20（GNOME 桌面）和 MacOS Big Sur，窗口非常具有标志性；
2. Ubuntu 的 Terminal 我加有注释 # 61520522 Wuqiong Zhao 和 ALL RIGHTS RESERVED (C) 2021 Wuqiong Zhao；
3. DOSBox 使用 MacOS，并且其中的文件日期 06-12-2021 即报告撰写日期.