

东南大学

《数字逻辑与计算机体系结构（含实验）》

实验报告

实验六 汇编语言程序设计

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 12 月 3 日

报告时间：2021 年 12 月 13 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 汇编语言调试	3
2 MIPS 迭代函数	3
1 阶乘	3
2 斐波那契数列	3
3 MIPS 递归函数	4
4 MIPS 多模块	7
5 MIPS 异常处理	8
6 X86 编译	11
3 创新与提高	11
4 实验总结	12
参考文献	12
附录 A：实验报告 L^AT_EX 模板	12
附录 B：程序真伪判别	13

一. 实验目的

- 掌握汇编语言源程序设计基本概念，掌握指令和伪指令的基本使用方法，掌握程序开发的各个环节—编辑、汇编、链接、调试、运行；
- 学习掌握计算机控制台输入输出的基本概念，了解系统调用的概念；
- 学习掌握利用 QtSpim 工具软件实现 MipsR2000 汇编与运行调试功能，加深指令理解认识，加深对函数调用、多汇编语言模块组织、多模块间调用/引用方法的认识理解，实现 Console 输入输出的基本多模块框架；
- 学习掌握利用 MASM/TASM—Link/Tlink 的汇编环节，学会 Debug/TD 调试 X86 汇编语言程序方法，加深指令宏观理解认识；（课外为主，配合实验 7、8 接口技术）^①

二. 实验内容

(一) 汇编语言调试

这在上周的实验报告中已经做了详细的介绍，此处将几个重要的点再次说一下。

MIPS 使用的是 Spim (QtSpim)，我更多使用命令行的控制台版本，因为使用方便，创建使用 `touch file.asm`，编辑 `nano file.asm` 或 `vim file.asm`，`spim` 进入 Spim 编译器。

- `reinitialize` 重初始化；
- `load "file.asm"` 导入文件；
- `run <ADDR>` 运行，可选开始时的地址；
- `step <N>` 单步运行，可选运行步数；
- `print` 输出，可以是一个寄存器，如 `print $s0`，也可以是地址 `print 0x00400060`；
- `exit` 退出。

X86 测试使用 DOSBox，主要内容见第 6 节。

(二) MIPS 迭代函数

1. 阶乘

迭代法求阶乘的代码如图 1a 所示，代码自己编写，在 `loop` 至 `end` 标记之间迭代循环，运行结果如图 1b 所示，结果存储在寄存器 `$s1` 中，使用命令 `print $s1` 得到 $7! = 5040$ 的计算结果。

2. 斐波那契数列

计算 100 以内最大的斐波那契数。

修改提供的代码，如图 2a 所示，运行结果如图 2b 所示，结果存储在寄存器 `$8` 中。

```

# factorial_direct.asm
# ALL RIGHTS RESERVED (C) 2021 Wuqiong Zhao

.text
main:
    addi $s0, $0, 7      # $s0 = 7
    addi $t0, $0, 2      # iter cnt
    addi $s1, $0, 1      # result
loop: mul $s1, $s1, $t0 # $s1 = $s1 * $t0
    beq $t0, $s0, end   # if ($t0 == $s0) goto end
    addi $t0, $t0, 1     # $t0++
    j loop              # iterate
end: jr $ra               # end of program

# To see the calculation result, look into register $s1
# In Spim, just call "print $s1"

```

[Read 16 lines]

GNU nano 2.0.6 File: factorial_direct.asm Modified

Get Help WriteOut Read File Prev Page Cut Text Cur Pos Exit Justify Where Is Next Page Uncut Text To Spell

```

# ALL RIGHTS RESERVED (C) 2021 Wuqiong Zhao
nano factorial_direct.asm
Spim
Loaded: /opt/local/share/spim/exceptions.s
(spim) load "factorial_direct.asm"
(spim) run
(spim) print $s1
Reg 17 = 0x000013b0 (5040)
(spim) exit
> # The MIPS program above shows that 7! = 5040
> # The answer is stored in register $s1
~Doc/Cod/L/SEU_Digital_Experiment_spim | master | ? |

```

(a) 代码 (Nano 界面)

(b) Spim 运行结果

图 1: 迭代求阶乘

```

# Find the largest Fibonacci number smaller than 100
.data
x: .space 4          # int x, y;
y: .space 4

.text
main:
    sw $0,x        # x = 0;
    addi $9,$0,1     # y = 1;
    sw $9,y
    lw $8,x
while:                  # while (y < 100) {
    slti $10,$9,100
    beq $10,$0,endw
    add $10,$9,$8    #     int t = x;
    add $8,$9,$9    #     x = y;
    add $9,$10,$9   #     y = t + y;
    j while         # }
endw:
    sw $8,x        # answer is in x
    sw $9,y
    ori $v0, $0, 10  # system call 10 for exit
    syscall          # we are out of here.

```

GNU nano 2.0.6 File: fibonacci.asm Modified

Get Help WriteOut Read File Prev Page Cut Text Cur Pos Exit Justify Where Is Next Page Uncut Text To Spell

```

nano fibonacci.asm
Spim
Loaded: /opt/local/share/spim/exceptions.s
(spim) load "fibonacci.asm"
(spim) step 5
[0x00040000] 0x8fa40000 lw $4, 0($29) ; 183: lw $a0 0($sp#arg0)
[0x00040004] 0x27e50004 addiu $5, $29, 4 ; 184: addiu $a1 $sp
[0x00040008] 0x24d60004 addiu $6, $5, 4 ; 185: addiu $a2 $a1
[0x0004000c] 0x00041000 sll $2, $4, 2 ; 186: sll $v0 $a0 2
[0x00040010] 0x00c23021 addu $6, $6, $2 ; 187: addu $a2 $a2
$v0
(spim) continue
(spim) (spim)
(spim) print $8
Reg 8 = 0x00000059 (89)
(spim) exit
~Doc/Cod/L/SEU_Digital_Experiment_spim | master | ? |

```

(a) 代码 (Nano 界面)

(b) Spim 运行结果

图 2: 迭代求阶乘

(三) MIPS 递归函数

使用递归的方法计算阶乘.

编译的 text 结果如图 3a 所示, 寄存器 $\$t1$ 中显示结果 $13b0$ 即 $7! = 5040$.

结果分析 1: 内存的变化

内存变化如图 3b 所示, 其中红框区域内显示每一次的乘数都被保存.

```

QtSpim
File Simulator Registers Text Segment Data Segment Window Help
Int Regs [16] FP Regs nt Regs [16]
Text Data
[00400010] 00c23021 addu $6, $6, $2 ; 187: addu $a2 $a2 $v0
[00400014] 0c100017 jal 0x040005c [main] ; 188: jal main
[00400018] 00000000 nop ; 189: nop
[0040001c] 3402000a ori $2, $0, 10 ; 191: li $v0 10
[00400020] 0000000c syscall ; 192: syscall # syscall 10 (exit)
[00400024] 1c800001 bgtz $4 12 [doit=0x0400024]; 12: bgtz $a0, doit
[00400028] 34020001 ori $2, $0, 1 ; 13: li $v0 1 # base case, 0! = 1
[0040002c] 03e00008 jr $31 ; 14: jr $ra
[00400030] 23bdfff8 addi $29, $29, -8 ; 16: sub $sp, 8 # stack frame
[00400034] afb00000 sw $16, 0($29) ; 17: sw $s0, 0($sp) # will use for argument n
[00400038] afbf0004 sw $31, 4($29) ; 18: sw $ra, 4($sp) # return address
[0040003c] 00048021 addu $16, $0, $4 ; 20: move $s0, $a0 # save argument
[00400040] 2084ffff addi $4, $4, -1 ; 21: sub $a0, 1 # n-1
[00400044] 0c100009 jal 0x0400024 [factorial]; 22: jal factorial # v0 = (n-1) !
[00400048] 72021002 mul $2, $16, $2 ; 23: mul $v0, $s0, $v0 # n*(n-1) !
[0040004c] 8fb00000 lw $16, 0($29) ; 25: lw $s0, 0($sp) # restore registers from stack
[00400050] 8fbf0004 lw $31, 4($29) ; 26: lw $ra, 4($sp)
[00400054] 23bd0008 addi $29, $29, 8 ; 27: add $sp, 8
[00400058] 03e00008 jr $31 ; 28: jr $ra
[0040005c] 34040007 ori $4, $0, 7 ; 30: li $s0, 7 # set the argument for the factorial
function to 7
[00400060] 23bdfffc addi $29, $29, -4 ; 31: sub $sp, 4 # create the stack frame
[00400064] afbf0000 sw $31, 0($29) ; 32: sw $ra, 0($sp) # save the return address
[00400068] 0c100009 jal 0x0400024 [factorial]; 33: jal factorial # call factorial
[0040006c] 00024821 addu $9, $0, $2 ; 34: move $t1, $v0 # save the return value
[00400070] 8fbf0000 lw $31, 0($29) ; 35: lw $ra, 0($sp) # restore the original return
address
[00400074] 23bd0004 addi $29, $29, 4 ; 36: add $sp, 4
[00400078] 03e00008 jr $31 ; 37: jr $ra
Kernel Text Segment [80000000]..[80010000]
[80000180] 0001d821 addu $27, $0, $1 ; 90: move $k1 $at # Save Sat
[80000184] 3c019000 lui $1, -28672 ; 92: sw $v0 $1 # Not re-entrant and we can't trust
.smc
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
QtSPIM is linked to the Qt library, which is distributed under the GNU Lesser General Public License version 3 and version 2.1.

```

(a) Text

```

QtSpim
File Simulator Registers Text Segment Data Segment Window Help
Int Regs [16] FP Regs nt Regs [16]
Data Text
PC = 400044 EPC = 0 Cause = 0 BadAddr = 0 Status = 3000ff10
HI = 0 LO = 0
User data segment [10000000]..[10040000]
[10000000]..[1003ffff] 00000000
User Stack [7fffff020]..[80000000]
[7fffff020] 00000002 00400048 00000003 00400048 . . . H . @ . . . . H . @ .
[7fffff030] 00000004 00400048 00000005 00400048 . . . H . @ . . . . H . @ .
[7fffff040] 00000006 00400048 00000007 00400048 . . . H . @ . . . . H . @ .
[7fffff050] 00000008 00400046 00400018 00000001 . . . 1 . @ . . . @ . . .
[7fffff060] 7fffff17c 00000000 7ffffff0 7fffff9e | . . . . . . . . . . . . .
[7fffff070] 7fffff8b 7fffff77 7fffff4a 7fffff33 . . . . . . . . . . . . .
[7fffff080] 7fffff10 7fffffe4 7fffffec 7ffffhead . . . . . . . . . . . . .
[7fffff090] 7fffffe9 7fffffe70 7fffff5c 7fffff45 . . . . . . . . . . . . .
[7fffff0a0] 7fffff2d 7fffffela 7fffffdde 7fffffde0 - . . . . . . . . . . . .
[7fffff0b0] 7fffffd4 7fffffdb9 7fffffd7d 7fffffde8 . . . . . . . . . . . .
[7fffff0c0] 7fffff34 7fffffd0b 7fffffd08 7fffffd08 4 . . . . . . . . . . . .
[7fffff0d0] 7fffffc09 7fffffbcb 7fffffcaa 7fffff95 . . . . . . . . . . . .
[7fffff0e0] 7fffffc84 7fffff6a2 7fffff681 7fffff670 . . . . . . . . . . . .
[7fffff0f0] 7fffff61a 7fffff5f4 7fffff5c5 7fffff5b5 . . . . . . . . . . . .
[7fffff100] 7fffff593 7fffff57c 7fffff568 7fffff54a . . . . | . . . h . . . j . .
[7fffff110] 7fffff52a 7fffff521 7fffff4e0 7fffff4e0 * . . . ! . . . . . . .
[7fffff120] 7fffff4d4 7fffff4cd 7fffff4b4 7fffff48e . . . . . . . . . . . .
[7fffff130] 7fffff47c 7fffff461 7fffff409 7fffff3f8 | . . . a . . . . . . .
[7fffff140] 7fffff3d4 7fffff3b4 7fffff394 7fffff380 . . . . . . . . . . . .
[7fffff150] 7fffff369 7fffff314 7fffff2f4 7fffff259 i . . . . . . . . . . . .
[7fffff160] 7fffff247 7fffff1eb 7fffff1d4 7fffff1c2 G . . . . . . . . . . . .
[7fffff170] 00000000 00000000 00000000 6d6f682f . . . . . . . . . . . / hom
[7fffff180] 67642f65 6f442f6a 656d7563 273746e e / t v j / D o c u m e n t s /
[7fffff190] 69726556 2f676f6c 5f554553 69676944 V e r i l o g / S E U _ D i g i
[7fffff1a0] 5f6cc6174 65707845 656d6972 732f746e t a l _ E x p e r i m e n t / s
[7fffff1b0] 2f6d6970 74636166 61697266 73612e6c p i m / f a c t o r i a l . a s
[7fffff1c0] 3d5f006d 7273752f 6669622f 7374712f m . _ = / u s r / b i n / q t s
[7fffff1d0] 006d6970 4e5f443c 52454d55 653d4349 p i m . L C _ N U M E R I C = e
[7fffff1e0] 53555f6 4654552e 44003820 5f535542 n _ U S . U T F - 8 . D B U S -
[7fffff1f0] 53534553 5f4e4f49 5f535542 5244441 S E S S I O N _ B U S _ A D D R
[7fffff200] 3d535345 78696e75 7461703a 722f3d68 E S S = u n i x : p a t h = / r
[7fffff210] 7fffff210 7fffff210 7fffff210 7fffff210 . . . . . . . . . . . .
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
QtSPIM is linked to the Qt library, which is distributed under the GNU Lesser General Public License version 3 and version 2.1.

```

(b) data

图 3: QtSpim

我又在提供代码的基础上增加了输入输出，得到了 `factorial_pro.asm`，代码如下，运行结果如图：

```

factorial_pro.asm
1 # Find the largest Fibonacci number smaller than 100
2
3 .data
4
5 x:    .space 4          # int x, y;
6 y:    .space 4
7 msg1:   .asciiz  "Enter A: "
8 msg2:   .asciiz  "Enter B: "
9 msg3:   .asciiz  "Largest Fibonacci No. <100)= "
10 newline: .asciiz  "\n"
11
12
13 .text
14
15 main:
16
17     sw    $0 ,x          # x = 0;
18     addi $9,$0,1         # y = 1;
19     sw    $9,y
20     lw    $8,x
21
22 while:                      # while (y < 100) {
23     slti $10,$9,100
24     beq  $10,$0,endw
25     add   $10,$0,$8      #     int t = x;
26     add   $8,$0,$9      #     x = y;
27     add   $9,$10,$9     #     y = t + y;
28     j    while           # }
29
30 endw:
31     sw    $8,x          # answer is in x
32     sw    $9,y
33
34 # Print string msg3
35     li    $v0, 4
36     la    $a0, msg3
37     syscall

```

```

38
39      # Print sum
40      li    $v0,1          # print_int syscall code = 1
41      lw    $a0,x           # int to print must be loaded into $a0
42      syscall
43
44      # Print \n
45      li    $v0,4          # print_string syscall code = 4
46      la    $a0, newline
47      syscall
48
49
50      ori   $v0, $0, 10   # system call 10 for exit
51      syscall            # we are out of here.

```

```

(spim) load fibonacci.asm
(spim) run
(spim) print $8
Reg 8 = 0x00000059 (89)
(spim) reinitialize
Loaded: /opt/local/share/spim/exceptions.s
SPIM Version 9.1.22 of May 9, 2020
Copyright 1990-2017 by James Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
(spim) load fibonacci_pro.asm
(spim) run
Largest Fibonacci No. <100>= 89
(spim) exit

```

图 4: factorial_pro.asm 运行结果

(四) MIPS 多模块

把 `fact.asm` 作为子模块源程序，主程序 `fact_main.asm` 中 `.extern factorial 32` 子模块源程序。`globl fact` 说明。

实现成功，效果如图 5 所示。图 5b 中显示，需要将两个文件都 `load` 才可以，否则会有符号未定义的错误。

```
GNU nano 2.0.6           File: fact.asm      Modified

factorial:
    bgtz $a0, doit
    li $v0, 1          # base case, 0! = 1
    jr $ra
doit:
    sub $sp,$8          # stack frame
    sw $s0,0($sp)       # will use for argument n
    sw $ra,4($sp)        # return address
    sub $s0,1            # n-1
    jal factorial        # v0 = (n-1)!)
    mul $v0,$s0,$v0       # n*(n-1)!

    lw $s0,($sp)
    lw $ra,4($sp)        # restore registers from stack
    add $sp,$8
    jr $ra
```

```
spim -- zsh - 80x24

) # ALL RIGHTS RESERVED (C) 2021 Wuqiong Zhao
) nano fact.asm
) nano fact_main.asm
) spim
Loaded: /opt/local/share/spim/exceptions.s
(spim) load "fact_main.asm"
(spim) run
The following symbols are undefined:
fact

Instruction references undefined symbol at 0x00400030
[0x00400030] 0x00000000 jal 0x00000000 [fact] ; 12: jal fact
    # call fact
(spim) load "fact.asm"
(spim) run
(spim) print $t1
Reg 9 = 0x000013b0 (5040)
(spim) exit
) # The file "fact.asm" where the function fact is defined also should be loaded
.
.

) The calculation result is stored in $t0.
zsh: command not found: The
~ Doc/Cod/L/SEL_Digital_Experiment/spim | master | 1 76 | 127 >
```

(a) fact.asm 代码 (Nano 界面)

(b) Spim 运行结果

图 5: 迭代求阶乘

(五) MIPS 异常处理

首先运行了代码，得到的结果为 0 并且有地址错误，进行 step 调试，关键的步骤如图 6 所示。

图 6: exception7.s 调试

结果分析 2: exception7.s 异常处理

从图 6 中可以看到几个不对的地方，第一，跳转 `loop` 仅有 8 次，加上第一次进入也只有 9 次，此外，使用命令 `print $t2` 可以得到结果 36，而这个值恰好是 $\sum_{i=0}^8 i$ ，因此我们可以认为迭代

循环次数存在问题。将其修改后问题仍然没有解决，我不使用地址指针，而是直接使用寄存器 \$t2 的值，可以得到正确的结果 45，如图 7 所示。

```

# ALL RIGHTS RESERVED (C) 2021 Wuqiong Zhao
nano exception7_solved.s
(spim)
Loaded: /opt/local/share/spim/exceptions.s
(spim) load "exception7_solved.s"
(spim) run
The value of sum is: 45
ALL RIGHTS RESERVED (C) 2021 Wuqiong Zhao
(spim) print $t2
Reg 10 = 0x0000002d (45)
(spim)

```

图 7：修改后成功运行的代码输出

此外也可以观察到寄存器 \$t2 中的值也是 45。

修改后的代码：

```

exception7_solved.asm

1 # Program for Laboratory 6: Exception Handling
2 #     This program generates exception number 7 by referring to an address
3 #     which is out of range
4 # Implements:
5 #         sum = 0;
6 #         i = 9;
7 #     do {
8 #         sum = sum + A[9-i]
9 #         B[9-i] = sum;
10 #        i--;
11 #    until (i == 0)

12
13 # Solved by 61520522 Wuqiong Zhao

14
15     .data
16 sum:    .word    0
17 A:      .word    0, 1, 2, 3, 4, 5, 6, 7, 8, 9
18 B:      .word    10, 11, 12, 13, 14, 15, 16, 17, 18, 1

```

```

19 Four:    .word    4
20 Nine:    .word    9
21 Minus1:  .word   -1
22 msg:     .asciiz "The value of sum is: "
23 notice:  .asciiz "\nALL RIGHTS RESERVED (C) 2021 Wuqiong Zhao\n"
24
25         .text
26         .globl main
27 main:
28         add    $t2, $0, $0      # sum = 0 (sum is in $t2)
29         add    $t1, $0, $0      # Set $t1 to point to beginning of data,
30         la     $t1,sum          # that is, to sum
31         lw     $t4, 84($t1)    # Constant 4 stored in $t4
32         lw     $t0, 88($t1)    # Constant 9 stored in $t0 (i)
33         lw     $t5, 92($t1)    # Constant -1 stored in $t5
34         add    $t1, $t1,$t4    # A starts at 4 (move $t1 to point to A)
35 loop:
36         lw     $t3, 0($t1)     # $t3 has A[i]
37         add    $t2, $t2, $t3    # sum = sum + A[i]
38         sw     $t2, 40($t1)    # B[i] = sum
39         add    $t1, $t1, $t4    # update address pointer
40         ##### EDIT HERE #####
41         beq    $t0, $0, done    # if i == 0 go to done
42         add    $t0, $t0, $t5    # i--
43         ##########
44         j      loop           # go to loop
45 done:
46         ##### DELETE TWO LINES #####
47         li     $v0, 4            # print_str
48         la     $a0, msg
49         syscall
50         li     $v0, 1            # print_int
51         ##### EDIT HERE #####
52         move   $a0, $t2
53         syscall
54         ##########
55
56         ##### EDIT HERE #####
57         li     $v0,4             # print_string
58         la     $a0, notice        # print copyright notice
59         syscall

```

```

60      ######
61      jr      $ra          # return to main

```

(六) X86 编译

我编译了一个输出日期时间的汇编程序。在正式开始之前，我先配置了 DOSBox 的环境：

```

1 mount C: ~/Programmes/DOSBox
2 mount D: ~/Documents/LaTeX/SEU_Digital_Experiment/dosbox
3 path C:;D:

```

然后使用 MASM 和 LINK 先生成 obj 文件再编译成 exe 文件，过程如图 8 所示。

```

DOSBox 0.74-3-3, Cpu speed: 3000 cycles, Frameskip 0, Program: TIME
.
<DIR> 13-12-2021 12:08
.
<DIR> 06-12-2021 8:13
DEBUG COM 22,476 08-09-2011 1:30
DS_STU~1 6,148 06-12-2021 8:56
EDIT COM 69,886 17-06-2004 20:00
FACT ASM 2,420 13-12-2021 12:06
TEST1 ASM 144 06-12-2021 8:58
TIME OBJ 293 13-12-2021 12:08
6 File(s) 101,367 Bytes.
2 Dir(s) 262,111,744 Bytes free.

D:>link
Microsoft (R) Overlay Linker Version 3.60
Copyright (C) Microsoft Corp 1983-1987. All rights reserved.

Object Modules [.OBJ]: time
Run File [TIME.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
LINK : warning L4021: no stack segment

D:>time
The Time is:12:09
The Date is:13/12/2021 14

```

图 8: X86 DOSBox 编译

三. 创新与提高

首先，我完成了第 5 节的异常解决工作，成功通过 Spim 命令行完成。

另外，我也对 MIPS 中数据类型转换做了相应的研究，有以下的代码，将 double 转为 int 型。

```

convert_type.asm

1 .data
2
3 number: .double 1.3
4
5 .text
6
7 l.s $f2, number

```

```

8
9 main:
10    l.d      $f0, fp1
11    l.d      $f1, fp2
12
13    mtc1    $a0, $f1
14    cvt.d.w $f1, $f1
15    div.d   $f3, $f1, $f2
16    cvt.w.d $f3, $f3
17    mfc1    $s2, $f3

```

四. 实验总结

这次的实验包括了 MIPS 和 X86 平台汇编代码的训练，整体来说较为简单，是基于一定的训练，有了较高的熟练度，可以像 C++ 一样结合互联网完成所要达成的代码目的。说一些感性的内容，我其实非常喜欢汇编语言的样子，非常的简洁（当然只是每一句话，整体必然会很长），配上命令行简直是绝配。我们现在写代码基本不会使用汇编，但是任何的编译性语言都基于了汇编，我在写 C++ 的时候就会思考我的代码究竟应该怎样才能优化更多。

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 %%%%%%%%%%%%%%%%
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 \extutor{冯漫} % 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 和 MacOS 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 13-12-2021 即报告撰写日期。