武汉大学计算机学院

本科生实验报告

计算机组成与课程设计实验

专 业 名 称 ：计算机科学与技术

课 程 名 称 ：计算机组成与课程设计

指 导 教 师 ：蔡朝晖

学 生 学 号 ：2018302080181

学 生 姓 名 ：沈之豪

二〇二〇年三月

一.结果展示：

1. Extendedtest.asm 文件：

```
# R[00-07]=0x00000000, 0x00000000, 0x00001234, 0x98760000, 0x98761234, 0x00000000, 0x00000000, 0x00000000
# R[08-15]=0x98761234, 0x00009876, 0x00000012, 0x00003410, 0x6e600000, 0x0000c3af, 0xffffc3af, 0x00000000
# R[16-23]=0x00000000, 0x00000000, 0x00000000, 0x00000001, 0x00000000, 0x00000000, 0x00000000, 0x00000000
# R[24-31]=0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000
# m[0] = 0x98761234
# m[4] = 0xfe766e60
# m[8] = 0xc3af60af
# m[0xc] = 0x98761234
# m[0x10] = 0xffff9876
# m[0x14] = 0x00009876
# m[0x18] = 0xffffff98
# m[0x1c] = 0x00000098
# m[0x20] = 0x00000012
```

2. Mipstest_extloop.asm 文件：

```
# R[00-07]=0x00000000, 0x00000000, 0x00000007, 0x0000000c, 0x00000001, 0x0000000b, 0x00000000, 0x00000007
# R[08-15]=0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000
# R[16-23]=0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000
# R[24-31]=0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000014
# m[0x50] =          7
# m[0x54] =          7
```

3. Mipstestloop_sim.asm 文件：

```
#
# R[00-07]=0x00000000, 0x00000000, 0x00000007, 0x0000000c, 0x00000001, 0x0000000b, 0x00000000, 0x00000007
# R[08-15]=0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000
# R[16-23]=0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000
# R[24-31]=0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000
# m[0x50] =          7
# m[0x54] =          7
```

4. Mipstestloopjal_sim.asm 文件：

```

# R[00-07]=0x00000000, 0x00000000, 0x00000007, 0x0000000c, 0x00000001, 0x0000000b, 0x00000000, 0x00000007
# R[08-15]=0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000
# R[16-23]=0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000
# R[24-31]=0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000040
# m[0x50] =          7
# m[0x54] =          7
```

二.结果分析：

1. 我们最后预期的结果：

(1). Extendedtest.asm 文件：

```
# $0=0              $1=0              $2=0x00001234     $3=0x98760000
# $4=0x98761234     $5=0              $6=0              $7=0
# $8=0x98761234     $9=0x00009876     $10=0x00000012    $11=0x00003410
# $12=0x6e600000    $13=0x0000c3af    $14=0xffffc3af    $19=0x0000001
# $29=0
# [0]=0x98761234    [4]=0xfe766e60    [8]=0xc3af60af    [0xc]=0x98761234
# [0x10]=0xffff9876 [0x14]=0x00009876 [0x18]=0xffffff98 [0x1c]=0x00000098
# [0x20]=0x00000012
```

(2). Mipstest_extloop.asm 文件：

```
# $0  = 0  # $1  = 0  # $2  = 7  # $3  = c
# $4  = 1  # $5  = b  # $7  = 7  # $31  = 14h
# [0x50] = 7  [0x54] = 7
```

(3). Mipstestloop_sim.asm 文件：

```
# $0  = 0   # $1  = 0   # $2  = 7   # $3  = c
# $4  = 1   # $5  = b   # $7  = 7
# [0x50] = 7   [0x54] = 7
```

(4). Mipstestloopjal_sim.asm 文件：

```
# $0  = 0   # $1  = 0   # $2  = 7   # $3  = c
# $4  = 1   # $5  = b   # $7  = 7   # $31 = 40
# [0x50] = 7   [0x54] = 7
```

经过仔细地比对，我们最后的结果和预期的完全一致，说明我们编写的指令集基本上没什么问题。

2. 为什么会产生这种结果（每条指令分析）

(1). Extendedtest.asm 文件：

```
lui   $3, 0x9876        # $3=0x98760000                                  # 3c039876
ori   $2, $0, 0x1234    # $2=0x1234                                      # 34021234
subu  $8, $3, $2        # $8=0x98760000-0x1234=0x9875edcc                # 00624023
xor   $9, $8, $3        # $9=0x9875edcc^0x98760000=0x0003edcc            # 01034826
addu  $10,$9, $8        # $10=0x0003edcc+0x9875edcc=0x9879db98           # 01285021
add   $10, $10, $2      # $10=0x9879db98+0x1234=0x9879edcc               # 01425020
sub   $11, $10, $3      # $11=0x9879edcc-0x98760000=0x0003edcc           # 01435822
nor   $12, $11, $10     # $12=~(0x0003edcc|0x9879edcc)=0x67841233        # 016a6027
or    $13, $11, $10     # $13=0x0003edcc|0x9879edcc=0x987bedcc           # 016a6825
and   $14, $11, $10     # $14=0x0003edcc&0x9879edcc=0x0001edcc           # 016a7024
slt   $19, $13, $12     # $19=(0x987bedcc<0x67841233)=1                  # 01ac982a
sltu  $20, $13, $12     # $20=(0x987bedcc<0x67841233)=0                  # 01aca02b
sll   $8, $8, 3         # $8=0x9875edcc<<3=0xc3af6e60                    # 000840c0
srl   $9, $8, 0x10      # $9=0xc3af6e60>>16=0xc3af                       # 00084c02
sra   $10, $8, 0x1d     # $10=0xc3af6e60>>>29=0xfffffffe                 # 00085743
ori   $11, $0, 0x3410   # $11=0x3410                                     # 340b3410
sllv  $12, $8, $11      # $12=0xc3af6e60<<16=0x6e600000                  # 01686004
srlv  $13, $8, $11      # $13=0xc3af6e60>>16=0xc3af                      # 01686806
srav  $14, $8, $11      # $14=0xc3af6e60>>>16=0xffffc3af                 # 01687007
addu  $4, $2, $3        # $4=0x1234+0x98760000=0x98761234                # 00432021
addi  $29, $0, 0x0      # $29=0                                          # 201d0000
sw    $4, 0x00($29)     # [0]=0x98761234                                 # afa40000
sw    $4, 0x04($29)     # [4]=0x98761234                                 # afa40004
sw    $4, 0x08($29)     # [8]=0x98761234                                 # afa40008
sh    $8, 0x04($29)     # [4]=0x98766e60 $8=0xc3af6e60                   # a7a80004  little endian
sh    $9, 0x0a($29)     # [8]=0xc3af1234 $9=0xc3af                       # a7a9000a  little endian
sb    $10, 0x07($29)    # [4]=0xfe766e60 $10=0xfffffffe                  # a3aa0007  little endian
sb    $8, 0x09($29)     # [8]=0xc3af6034 $8=0xc3af6e60                   # a3a80009  little endian
sb    $9, 0x08($29)     # [8]=0xc3af60af $9=0xc3af                       # a3a90008  little endian
lw    $8, 0x00($29)     # $8=0x98761234                                  # 8fa80000
sw    $8, 0x0c($29)     # [0xc]=0x98761234                               # afa8000c
lh    $9, 0x02($29)     # $9=0xffff9876 [0]=0x98761234                   # 87a90002  little endian, sign extension
sw    $9, 0x10($29)     # [0x10]=0xffff9876                              # afa90010
lhu   $9, 0x02($29)     # $9=0x00009876 [0]=0x98761234                   # 97a90002  little endian, zero extension
sw    $9, 0x14($29)     # [0x14]=0x00009876                              # afa90014
lb    $10, 0x03($29)    # $10=0xffffff98  [0]=0x98761234                 # 83aa0003  little endian, sign extension
sw    $10, 0x18($29)    # [0x18]=0xffffff98                              # afaa0018
lbu   $10, 0x03($29)    # $10=0x00000098 [0]=0x98761234                  # 93aa0003  little endian, zero extension
sw    $10, 0x1c($29)    # [0x1c]=0x00000098                              # afaa001c
lbu   $10, 0x01($29)    # $10=0x00000012 [0]=0x98761234                  # 93aa0001  little endian, zero extension
sw    $10, 0x20($29)    # [0x20]=0x00000012                              # afaa0020
```

(2). Mipstest_extloop.asm 文件：

```
#       Assembly              Description           Address    Machine
main:   addi $2, $0, 5        # initialize $2 = 5     0        20020005
        addi $3, $0, 12       # initialize $3 = 12    4        2003000c
        addi $7, $3, -9       # initialize $7 = 3     8        2067fff7
        addi $1, $0, 76       # initialize $1 = 76    c        2001004c
call_a: jalr $31,$1           # jump to cal           10       0020f809
        or   $4, $7, $2       # $4 <= 3 or 5 = 7      14       00e22025
        and  $5, $3, $4       # $5 <= 12 and 7 = 4    18       00642824
        add  $5, $5, $4       # $5 = 4 + 7 = 11       1c       00a42820
        beq  $5, $7, end      # shouldn't be taken    20       10a70017
        slt  $4, $3, $4       # $4 = 12 < 7 = 0       24       0064202a
        beq  $4, $0, around   # should be taken       28       10800001
        addi $5, $0, 0        # shouldn't happen      2c       20050000
around: slt  $4, $7, $2       # $4 = 3 < 5 = 1        30       00e2202a
        add  $7, $4, $5       # $7 = 1 + 11 = 12      34       00853820
        sub  $7, $7, $2       # $7 = 12 - 5 = 7       38       00e23822
        sw   $7, 68($3)       # [80] = 7              3c       ac670044
        lw   $2, 80($0)       # $2 = [80] = 7         40       8c020050
        j    end              # should be taken       44       08000020
        addi $2, $0, 1        # shouldn't happen      48       20020001
cal:    sll  $7, $7, 1        # $7 << 1 = 6           4c       00073840
call_b: jal  cal2             # jump to cal2          50       0c000017
        addi $31,$0,20        # $31<= 20              54       201f0014
        jr   $31              # return to call_a      58       03e00008
cal2:   lui  $1, 0x55AA       # $1 <= 0x55AA0000      5c       3c0155aa
        slti $1, $1, 0x55AA   # $1 <= 0               60       282155aa
        bne  $1, $0, end      # shouldn't be taken    64       14200006
        ori  $7, $7, 5        # $7 <= 6 or 5 = 7      68       34e70005
        andi $1, $7, 5        # $1 <= 7 and 5 = 5     6c       30e10005
        addu $1, $7, $1       # $1 = 7+5 = 12         70       00e10821
        subu $1, $1, $1       # $1 =12-12 = 0         74       00210823
        srl  $7, $7, 1        # $7 >> 1 = 3           78       00073842
        jr   $31              # return to call_b      7c       03e00008
end:    sw   $2, 84($0)       # write adr 84 = 7      80       ac020054
loop:   j    loop
```

(3). Mipstestloop_sim.asm 文件:

```
#       Assembly              Description           Instr   Address Machine
main:   addi $2, $0, 5        # initialize $2 = 5     00      0       20020005
        addi $3, $0, 12       # initialize $3 = 12    01      4       2003000c
        addi $7, $3, -9       # initialize $7 = 3     02      8       2067fff7
        or   $4, $7, $2       # $4 = (3 or 5) = 7     03      c       00e22025
        and  $5, $3, $4       # $5 = (12 and 7) = 4   04      10      00642824
        add  $5, $5, $4       # $5 = 4 + 7 = 11       05      14      00a42820
        beq  $5, $7, label2   # shouldn't be taken    06      18      10a7000a
        slt  $4, $3, $4       # $4 = (12 < 7) = 0     07      1c      0064202a
        beq  $4, $0, label1   # should be taken       08      20      10800001
        addi $5, $0, 0        # shouldn't happen      09      24      20050000
label1: slt  $4, $7, $2       # $4 = (3 < 5) = 1      0A      28      00e2202a
        add  $7, $4, $5       # $7 = 1 + 11 = 12      0B      2c      00853820
        sub  $7, $7, $2       # $7 = 12 - 5 = 7       0C      30      00e23822
        sw   $7, 68($3)       # [80] = 7              0D      34      ac670044
        lw   $2, 80($0)       # $2 = [80] = 7         0E      38      8c020050
        j    label2           # should be taken       0F      3c      08000011
        addi $2, $0, 1        # shouldn't happen      10      40      20020001
label2: sw   $2, 84($0)       # write adr 84 = 7      11      44      ac020054
loop:   j    loop             # dead loop             12      48      08000012
```

(4). Mipstestloopjal_sim.asm 文件:

```
#       Assembly              Description           Instr   Address Machine
main:   addi $2, $0, 5        # initialize $2 = 5     00      0       20020005
        addi $3, $0, 12       # initialize $3 = 12    01      4       2003000c
        addi $7, $3, -9       # initialize $7 = 3     02      8       2067fff7
        or   $4, $7, $2       # $4 = (3 or 5) = 7     03      c       00e22025
        and  $5, $3, $4       # $5 = (12 and 7) = 4   04      10      00642824
        add  $5, $5, $4       # $5 = 4 + 7 = 11       05      14      00a42820
        beq  $5, $7, label2   # shouldn't be taken    06      18      10a7000a
        slt  $4, $3, $4       # $4 = (12 < 7) = 0     07      1c      0064202a
        beq  $4, $0, label1   # should be taken       08      20      10800001
        addi $5, $0, 0        # shouldn't happen      09      24      20050000
label1: slt  $4, $7, $2       # $4 = (3 < 5) = 1      0A      28      00e2202a
        add  $7, $4, $5       # $7 = 1 + 11 = 12      0B      2c      00853820
        sub  $7, $7, $2       # $7 = 12 - 5 = 7       0C      30      00e23822
        sw   $7, 68($3)       # [80] = 7              0D      34      ac670044
        lw   $2, 80($0)       # $2 = [80] = 7         0E      38      8c020050
        jal  label2           # should be taken       0F      3c      0c000011
        addi $2, $0, 1        # shouldn't happen      10      40      20020001
label2: sw   $2, 84($0)       # write adr 84 = 7      11      44      ac020054
loop:   j    loop             # dead loop             12      48      08000012
```

三.碰到的问题

1. Modelsim 中的 Windows 路径问题

描述：Windows 下路径由反斜杠分隔，但是 Modelsim 不识别这种格式的路径。
解决：将路径的分隔符改成 UNIX-like 的正斜杠分隔的形式。

2. Modelsim 中无法读取机器指令的问题：

描述：为了让我们的 CPU 代码更像一个产品，所有文件都得由相对路径来寻找，但是 Modelsim 中无法识别相对路径的文件。
解决：改成绝对路径。

3. $display 的位置问题：

描述：在 debug 的过程中，可能是我们部件代码出了问题。为了定位到底是哪里出了问题，我们得在该代码中添加 display 进行寄存器结果的查看，但是偶尔会出现 unexpected 错误。
解决：原来 display 得在 always 代码块中。如果想要添加 display 的话，就用 always 包裹。

4. 如何退出仿真问题：

描述：由于仿真是建立在已经编译好的文件上，如果我们修改了文件，就又要重新编译。这时候我们不得不关掉软件，再次开启，特别耗时。
解决：在控制台中输入 quit -sim，可退出仿真模式，接下来可以直接进行编译。

5. verilog 不区分大小写问题：

描述：verilog 对于大小写敏感，但是在编译的时候却不会报错，比如我只声明了 imm，但是后来使用了未定义的 lmm，后果就是不会对 imm 有影响，而且 lmm 每一位都是高阻态。
解决：这个得自己改动大小写。

6. Verilog 未定义变量的问题：

描述：作为编译型语言，使用变量的时候应该提前声明，但是 verilog 不声明变量也不会报错，会出现很奇怪的错误。
解决：每次使用变量的时候，一定要去看看是否有相应的声明

7. Verilog 重名的模块问题：

描述：在写 testbench 的时候，由于大部分都是重复的代码，所以我直接复制粘贴了，忘了将模块的名称修改过来。没想到 verilog 竟然没报错。它把同名的模块给覆盖了。
解决：修改模块名称即可。

8. 信号清零问题：

描述：对于每一种指令而言，不会用到所有 control 的信号。但是那些没有用到 control 的信号，对我们的指令也有潜移默化的影响。比如，如果我上一个是 branch 指令，那么 branch 的信号就为 1。下一个指令比如是 add 指令，他就没有用到 branch 指令，如果此时的 branch 信号不清 0，那我们的 PC 就会发生未知的变动！
解决：在 control 模块中，每一条信号最后都添加一个 else，保证没有到这个信号的指令，执行到 control 的时候，都会自动将这个信号清零。

9. SLT 进行有符号检测的问题：

描述：verilog 在比较我们信号的时候，都是按照无符号数的标准来考虑的。对于有符号数而言，由于负数的最高位是 1，反而会出现负数比正数大的诡异情况。

解决：用一个小技巧来解决$[A-B]_{补}=[A]_{补}+[-B]_{补}$

我们传入的数据其实都是原数据的补码，为了得到$A-B$的符号，我们可以判断$[A-B]_{补}$的

最高位。另外，我们有$[-B]_{补}=\sim[B]_{补}+1$。综上所述，我们只要求得$A+\sim B+1$的最高位

结果就行了！

10. SRA/SRAV 算数右移问题：

描述：虽然 verilog 中已经为我们提供了算数右移的操作符，但是在仿真的时候，就是这个 SRA/SRAV 的结果总是不对，难道这个>>>有问题？
解决：实操中发现>>>有时会在右移时仍然补零，即使符号位为 1。这是因为>>>会先判断这个操作数是否有符号数。如果是无符号数，则补零，是有符号数，才会补符号位。而一般使用的 reg operand; 这种变量定义法默认所定义的变量为无符号数，因此只补零。解决办法是利用 verilog 的内置函数$signed()，将被移位的操作数转为有符号数类型。