**《（课程名称）》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **2021级计算机专业05、03班** | | | **姓名** | **张梓健，任俊璇** |
| **实验题目** | 浮点数的表示和计算 | | | | | |
| **实验时间** | **2023.5.28** | | **实验地点** | **竹四221** | | |
| **实验成绩** |  | | **实验性质** | **□验证性 □设计性 □综合性** | | |
| 教师评价：  □算法/实验过程正确； □源程序/实验内容提交 □程序结构/实验步骤合理；  □实验结果正确； □语法、语义正确； □报告规范；  其他：  评价教师签名： | | | | | | |
| 一、实验目的  (1) 深入掌握二进制数的表示方法以及不同进制数的转换；  (2) 掌握二进制不同编码的表示方法；  (3) 掌握IEEE 754 中单精度浮点数的表示和计算。 | | | | | | |
| 二、实验项目内容  假设没有浮点表示和计算的硬件，用软件方法采用仿真方式实现IEEE 754单精度浮点数的表示及运算功能，具体要求如下：  (1) 程序需要提供人机交互方式（GUI或者字符界面）供用户选择相应的功能；  (2) 可接受十进制实数形式的输入，在内存中以IEEE 754单精度方式表示，支持以二进制和十六进制的方式显示输出；  (3) 可实现浮点数的加减乘除运算；  (4) 可以使用80X86或MIPS或ARM汇编指令，但是不能使用浮点指令，只能利用整数运算指令来编写软件完成。 | | | | | | |
| 三、实验过程或算法（源程序）  # This is a floating point calculator（FPC） that supports addition, subtraction, multiplication and division of two input floating point numbers and returns the results in binary and hexadecimal formats.  .data  msg\_welcome: .asciiz "\nHello!I am a floating point calculator that can menu addition, subtraction, multiplication and division.\nPlease follow the instructions and input numbers to use me.\n"  msg\_menu: .asciiz "\nMenu\n1: + 2: - 3: × 4: ÷ 5: Exit\nChoose:"  msg\_exit: .asciiz "\nThanks for your using. Bye bye!\n"  msg\_first\_float: .asciiz "\nFirst floating-point value:"  msg\_sencond\_float: .asciiz "\nSecond floating-point value:"  msg\_invalid\_input: .asciiz "\nInvalid input!\n"  msg\_print\_bin: .asciiz "\nBinary result:\n"  msg\_print\_hex: .asciiz "\nHexadecimal result:\n"  error\_over\_flow: .asciiz "\nError: overflow!\n"  error\_under\_flow: .asciiz "\nError: underflow!\n"  error\_div\_zero: .asciiz "\nError: Cannot divde by zero!\n"  hex\_table: .asciiz "0123456789ABCDEF"  hex\_digits: .asciiz "XXXXXX"  string\_neg: .asciiz "-"  string\_1dot: .asciiz "1."  string\_0dot: .asciiz "0."  string\_totwo: .asciiz "\*2^"  string\_to16: .asciiz "\*16^"  string\_0: .asciiz "0"  string\_1: .asciiz "1"  string\_hex0: .asciiz "000000\*16^0"  .text  # -- 欢迎 --  welcome:  la $a0, msg\_welcome # 输出欢迎语  li $v0, 4 # 打印字符串  syscall  j menu # 跳转到menu  # -- 功能菜单 --  menu:  la $a0, msg\_menu # 输出菜单提示消息  li $v0, 4 # 打印字符串  syscall  # 读取用户的输入  li $v0, 5 # 读取整数  syscall  move $v1, $v0 # 将输入值存入$v1  j branch # 跳转到branch  branch:  beq $v1, 1, read\_float # 如果输入值为1，跳转到read\_float  beq $v1, 2, read\_float # 如果输入值为2，跳转到read\_float  beq $v1, 3, read\_float # 如果输入值为3，跳转到read\_float  beq $v1, 4, read\_float # 如果输入值为4，跳转到read\_float  beq $v1, 5, exit # 如果输入值为5，跳转到exit  # 其他的输入则提示输入错误，重新输入  la $a0, msg\_invalid\_input # 输出输入错误提示消息  li $v0, 4 # 打印字符串  syscall  j menu # 跳转到menu  read\_float:  # 读取第一个浮点数  la $a0, msg\_first\_float # 输出第一个浮点数提示消息  li $v0, 4 # 打印字符串  syscall  li $v0, 6 # 读取浮点数  syscall  mfc1 $t0, $f0 # 将浮点数存入$t0寄存器  srl $s0, $t0, 31 # 保存第一个浮点数的符号至$s0    sll $s1, $t0, 1 # 保存第一个浮点数的指数至$s1  srl $s1, $s1, 24  sll $s2, $t0, 9 # 保存第一个浮点数的尾数至$s2  srl $s2, $s2, 9  addi $s2, $s2, 0x00800000 # 尾数补前导位1 16进制数  # 读取第二个浮点数  la $a0, msg\_sencond\_float # 输出第二个浮点数提示消息  li $v0, 4 # 打印字符串  syscall  li $v0, 6 # 读取浮点数  syscall  mfc1 $t0, $f0 # 将浮点数存入$t0寄存器  srl $s3, $t0, 31 # 保存第二个浮点数的符号至$s3  sll $s4, $t0, 1 # 保存第二个浮点数的指数至$s4  srl $s4, $s4, 24  sll $s5, $t0, 9 # 保存第二个浮点数的尾数至$s5  srl $s5, $s5, 9  addi $s5, $s5, 0x00800000 # 尾数补前导位1 16进制数  beq $v1, 1, add # 如果输入值为1，跳转到add  beq $v1, 2, sub # 如果输入值为2，跳转到sub  beq $v1, 3, multiply # 如果输入值为3，跳转到multiply  beq $v1, 4, divide # 如果输入值为4，跳转到divide  la $a0, msg\_invalid\_input # 加载输入错误提示消息  li $v0, 4 # 打印字符串  syscall  j menu # 跳转到menu    # -- 加法 --  add:  sub $t0, $s1, $s4 # 计算两个指数的差  bltz $t0, adjust\_first\_operand # 如果差值为负，跳转到 adjust\_first\_operand  bgtz $t0, adjust\_second\_operand # 如果差值为正，跳转到 adjust\_second\_operand  beq $t0, $0, judge\_sign # 如果差值为零，跳转到 judge\_sign  adjust\_first\_operand: # 对第一个数进行调整  addi $s1, $s1, 1 # 指数加1  srl $s2, $s2, 1 # 尾数右移1位  j add # 跳回 add 重新进行计算    adjust\_second\_operand: # 对第二个数进行调整  addi $s4, $s4, 1 # 指数加1  srl $s5, $s5, 1 # 尾数右移1位  j add # 跳回 add 重新进行计算  judge\_sign: # 符号判断  xor $t3, $s0, $s3 # 对两个符号进行异或操作，结果存放在$t3中  beq $t3, 0, same\_sign # 如果两符号相同，跳转到 same\_sign  beq $t3, 1, diff\_sign # 如果两符号不同，跳转到 diff\_sign  same\_sign: # 符号相同的情况  add $t3, $s2, $s5 # 尾数相加  move $t2, $s1 # 将指数移动到$t2  move $t1, $s0 # 将符号移动到$t1  bge $t3, 0x01000000, carry # 如果尾数加法后进位，跳转到 carry  j print\_result # 否则直接输出结果  diff\_sign: # 符号不同的情况  move $t2, $s1 # 将指数移动到$t2  sub $t3, $s2, $s5 # 尾数相减  bgtz $t3, first\_operand\_bigger # 如果差值为正，跳转到 first\_operand\_bigger  bltz $t3, second\_operand\_bigger # 如果差值为负，跳转到 second\_operand\_bigger  beq $t3, $0, print\_zero # 如果差值为零，输出0  first\_operand\_bigger: # 第一个数大的情况  move $t1, $s0 # 将符号移动到$t1  j adjust\_sub # 跳转到 adjust\_sub进行调整  second\_operand\_bigger: # 第二个数大的情况  move $t1, $s3 # 将符号移动到$t1  sub $t3, $s5, $s2 # 将尾数相减  j adjust\_sub # 跳转到 adjust\_sub进行调整  adjust\_sub: # 进行尾数调整  blt $t3, 0x00800000, adjust\_sub1 # 如果尾数小于0x00800000，跳转到 adjust\_sub1  j print\_result # 否则直接输出结果  adjust\_sub1: # 进行尾数调整1  beq $t2, 0, error\_underflow # 如果指数为0，抛出下溢错误  addi $t2, $t2, -1 # 指数减1  sll $t3, $t3, 1 # 尾数左移1位  blt $t3, 0x00800000, adjust\_sub1 # 如果尾数小于0x00800000，继续调整  j print\_result # 否则直接输出结果  carry: # 进位处理  beq $t2, 255, error\_overflow # 如果指数为255，抛出上溢错误  srl $t3, $t3, 1 # 尾数右移1位  addi $t2, $t2, 1 # 指数加1  j print\_result # 输出结果  # -- 减法 --  sub:  xori $s3, $s3, 0x00000001 # 对第二个浮点数符号进行取反  j add # 然后执行加法  # -- 乘法 --  multiply:  beq $s1, 0, mult\_first\_exp\_zero # 如果第一个操作数的指数为0，跳转到mult\_first\_exp\_zero  beq $s4, 0, mult\_second\_exp\_zero # 如果第二个操作数的指数为0，跳转到mult\_second\_exp\_zero  j mult\_operands\_nonzero # 如果两个操作数的指数都不为0，跳转到mult\_operands\_nonzero    mult\_first\_exp\_zero: # 第一个操作数指数为0的处理过程  beq $s2, 0x800000, mult\_operand\_has\_zero # 如果第一个操作数的尾数为最大值，跳转到mult\_operand\_has\_zero  beq $s4, 0, mult\_second\_exp\_zero # 如果第二个操作数的指数为0，跳转到mult\_second\_exp\_zero  j mult\_operands\_nonzero # 如果以上都不满足，跳转到mult\_operands\_nonzero  mult\_second\_exp\_zero: # 第二个操作数指数为0的处理过程  beq $s5, 0x800000, mult\_operand\_has\_zero # 如果第二个操作数的尾数为最大值，跳转到mult\_operand\_has\_zero  j mult\_operands\_nonzero # 如果不满足，跳转到mult\_operands\_nonzero  mult\_operand\_has\_zero: # 操作数中存在0的处理过程  li $t1, 0 # 设置结果的符号为0  li $t2, 0 # 设置结果的指数为0  li $t3, 0 # 设置结果的尾数为0  j multiply\_end # 跳转到multiply\_end结束此过程  mult\_operands\_nonzero: # 两个操作数都不为0的处理过程  add $t2, $s1, $s4 # 指数部分相加  li $t4, 127 # 设置一个中间变量$t4为常数127  sub $t2, $t2, $t4 # 指数相加的结果减去127，得到新的指数  mult $s2, $s5 # 尾数部分相乘  mfhi $t3 # 取乘法结果的高位，HI: 16位0, 2位整数部分, 14位小数部分  mflo $t4 # 取乘法结果的低位，LO: 32位小数剩余部分  sll $t3, $t3, 9 # 将高位左移9位  srl $t4, $t4, 23 # 将低位右移23位  or $t3, $t3, $t4 # 高位和低位进行逻辑或操作，得到新的尾数  # 归一化  srl $t4, $t3, 24 # 将尾数右移24位，取得第25位  beq $t4, $0, after\_norm # 如果第25位为0，跳过归一化过程  srl $t3, $t3, 1 # 将尾数右移一位  addi $t2, $t2, 1 # 指数加1，完成归一化  after\_norm:  slti $t4, $t2, 0 # 如果指数小于0，$t4为1，否则为0  beq $t4, 1, error\_underflow # 如果$t4为1，表示指数下溢，跳转到error\_underflow处理过程  li $t4, 255 # 设置$t4为常数255  slt $t4, $t4, $t2 # 如果$t4小于$t2，$t4为1，否则为0  beq $t4, 1, error\_overflow # 如果$t4为1，表示指数上溢，跳转到error\_overflow处理过程  xor $t1, $s0, $s3 # 对两个操作数的符号位进行异或操作，得到结果的符号位  j multiply\_end # 跳转到multiply\_end结束此过程  multiply\_end: # 乘法过程结束  j print\_result # 跳转到输出    # -- 除法 --  divide: # 定义除法过程  beq $s1, 0, div\_first\_exp\_zero # 如果被除数的指数为0，跳转到div\_first\_exp\_zero  j div\_operands\_nonzero # 如果被除数的指数不为0，跳转到div\_operands\_nonzero  div\_first\_exp\_zero: # 被除数指数为0的处理过程  beq $s2, 0x800000, div\_first\_operand\_zero # 如果被除数的尾数为最大值，跳转到div\_first\_operand\_zero  j div\_operands\_nonzero # 如果被除数的尾数不为最大值，跳转到div\_operands\_nonzero  div\_first\_operand\_zero: # 被除数为0的处理过程  li $t1, 0 # 设置结果的符号为0  li $t2, 0 # 设置结果的指数为0  li $t3, 0 # 设置结果的尾数为0  j div\_end # 跳转到div\_end结束此过程  div\_operands\_nonzero: # 被除数不为0的处理过程  bne $s4, 0, normal # 如果除数的指数不为0，跳转到normal  bne $s5, 0x800000, normal # 如果除数的尾数不为最大值，跳转到normal  j error\_divided\_by\_zero # 如果除数为0，跳转到error\_divided\_by\_zero处理过程  normal: # 正常的处理过程  sub $t2, $s1, $s4 # 指数部分相减  addi $t2, $t2, 127 # 结果加上常数127，得到新的指数  xor $t1, $s0, $s3 # 对两个操作数的符号位进行异或操作，得到结果的符号位  div $s2, $s5 # 尾数部分相除  mflo $t3 # 取除法结果的低位，作为结果的尾数的整数部分  mfhi $t4 # 取除法结果的高位，作为新的尾数  beq $t3, $0, div\_end # 如果结果的尾数为0，跳转到div\_end结束此过程  li $t5, 1 # 设置一个中间变量$t5为1  div\_loop1: # 第一个循环，确定整数部分的位数  srlv $t6, $t3, $t5 # 将尾数右移$t5位  bne $t6, $0, div\_loop1 # 如果右移后的尾数不为0，继续循环  li $t6, 1 # 设置一个中间变量$t6为1  sub $t5, $t5, $t6 # 将$t5减去1，得到新的$t5  add $t2, $t2, $t5 # 指数部分加上$t5，得到新的指数  slti $t4, $t2, 0 # 如果新的指数小于0，$t4为1，否则为0  beq $t4, 1, error\_underflow # 如果$t4为1，表示指数下溢，跳转到error\_underflow处理过程  li $t4, 255 # 设置$t4为常数255  slt $t4, $t4, $t2 # 如果$t4小于$t2，$t4为1，否则为0  beq $t4, 1, error\_overflow # 如果$t4为1，表示指数上溢，跳转到error\_overflow处理过程  li $t7, 23 # 设置一个中间变量$t7为23  sub $t7, $t7, $t5 # 将$t7减去$t5，得到新的$t7  li $t6, 0 # 设置一个中间变量$t6为0  div\_loop2: # 第二个循环，计算小数部分的位数  sll $t4, $t4, 1 # 将尾数左移1位  div $t4, $s5 # 将新的尾数除以除数  mflo $t8 # 取除法结果的低位，作为新的尾数  mfhi $t4 # 取除法结果的高位，作为新的尾数  sll $t3, $t3, 1 # 将结果的尾数左移1位  add $t3, $t3, $t8 # 将新的尾数加到结果的尾数上，得到新的尾数  addi $t6, $t6, 1 # 将计数器$t6加1  beq $t6, $t7, div\_end # 如果计数器等于$t7，跳转到div\_end结束此过程  beq $t4, $0, div\_comp\_dec # 如果新的尾数为0，跳转到div\_comp\_dec  j div\_loop2 # 否则，继续循环  div\_comp\_dec: # 尾数为0时的处理过程  sub $t6, $t7, $t6 # 将$t7减去$t6，得到新的$t6  sllv $t3, $t3, $t6 # 将结果的尾数左移$t6位，得到新的尾数  div\_end: # 除法结束  j print\_result # 跳转到print\_result输出结果  # -- 错误信息输出 --  error\_divided\_by\_zero:  la $a0, error\_div\_zero # 被零除错误消息  li $v0, 4 # 打印字符串  syscall  j exit # 跳转到程序结束  error\_overflow:  la $a0, error\_over\_flow # 溢出错误消息  li $v0, 4 # 打印字符串  syscall  j exit # 跳转到程序结束  error\_underflow:  la $a0, error\_under\_flow # 下溢错误消息  li $v0, 4 # 打印字符串  syscall  j exit # 跳转到程序结束  # -- 输出结果 --  print\_zero: # 输出0  move $a0, $0  li $v0, 1 # 打印整数  syscall  li $v0, 11 # 换行  li $a0, '\n'  syscall  j menu # 跳转到计算过程  print\_result: # 输出计算结果  li $v0, 4 # 打印字符串  la $a0, msg\_print\_bin # 结果二进制消息  syscall  # 判断结果是否为0  beq $t1, 0, check\_bin\_exp\_zero # 如果结果符号位为0，跳转到check\_bin\_exp\_zero  j print\_bin\_not\_zero # 否则跳转到print\_bin\_not\_zero  check\_bin\_exp\_zero:  beq $t2, 0, print\_bin\_zero # 如果结果指数为0，跳转到print\_bin\_zero  j print\_bin\_not\_zero # 否则跳转到print\_bin\_not\_zero  print\_bin\_zero:  la $a0, string\_0dot # "0."  syscall  move $a1, $t3 # 将结果尾数存入$a1  li $a2, 22 # 设置打印位数为22  jal print\_bits # 打印结果尾数  la $a0, string\_totwo # "\*2^"  syscall  move $a0, $t2 # 结果指数存入$a0  li $v0, 1 # 打印整数  syscall  jal print\_bin\_end # 跳转到二进制输出结束  j menu # 跳转到计算过程  print\_bin\_not\_zero:  beq $t1, 0, skipBinNeg # 如果结果符号位为0，跳过负号  la $a0, string\_neg # "-"  syscall  skipBinNeg:  la $a0, string\_1dot # "1."  syscall  move $a1, $t3 # 将结果尾数存入$a1  li $a2, 22 # 设置打印位数为22  jal print\_bits # 打印结果尾数  la $a0, string\_totwo # "\*2^"  syscall  addi $a0, $t2, -127 # 结果指数减去127  li $v0, 1 # 打印整数  syscall  jal print\_bin\_end # 跳转到二进制输出结束  j menu # 跳转到计算过程  print\_bin\_end:  li $v0, 4 # 打印字符串  la $a0, msg\_print\_hex # 结果十六进制消息  syscall  beq $t1, 0, check\_hex\_exp\_zero # 如果结果符号位为0，跳转到check\_hex\_exp\_zero  j print\_hex\_not\_zero # 否则跳转到print\_hex\_not\_zero  check\_hex\_exp\_zero:  beq $t2, 0, print\_hex\_zero # 如果结果指数为0，跳转到print\_hex\_zero  j print\_hex\_not\_zero # 否则跳转到print\_hex\_not\_zero  print\_hex\_zero:  la $a0, string\_0dot # "0."  syscall  la $a0, string\_hex0 # "000000\*16^0"  syscall  j print\_hex\_end # 跳转到十六进制输出结束  print\_hex\_not\_zero:  lw $t1, 28($sp) #取出结果符号位  beq $t1, 0, skipHexoutNeg # 如果结果符号位为0，跳过负号  la $a0, string\_neg # "-"  syscall  skipHexoutNeg:  addi $t7, $t2, -127 # 结果指数减去127  bltz $t7, hex\_exp\_negative # 如果结果指数小于0，跳转到hex\_exp\_negative  andi $t4, $t7, 0x3 # 取结果指数除以4的余数，存入$t4  srl $t5, $t7, 2 # 取结果指数除以4的商，存入$t5  j prepare\_hex\_output # 跳转到prepare\_hex\_output  hex\_exp\_negative:  li $t4, 0 # 初始化计数器$t4  move $t6, $7 # 将计数器$t6初始化为23  hex\_out\_loop:  andi $t7, $t6, 0x3 # 取计数器$t6除以4的余数，存入$t7  beq $t7, 0, hex\_out\_loopEnd # 如果余数为0，跳转到hex\_out\_loopEnd  addi $t6, $t6, -1 # 计数器$t6减1  addi $t4, $t4, 1 # 计数器$t4加1  j hex\_out\_loop # 继续循环  hex\_out\_loopEnd:  srl $t5, $t6, 2 # 取计数器$t6除以4的商，存入$t5  prepare\_hex\_output:  li $t7, 23 # 设置中间变量$t7为23  sub $t6, $t7, $t4 # 将$t7减去$t4，得到新的$t6  srlv $t6, $t3, $t6 # 将结果尾数右移$t6位  # 输出小数点前部分  move $a0, $t6 # 将新的尾数存入$a0  li $a1, 0 # 设置$a1为0  jal convert\_to\_hex # 跳转到convert\_to\_hex  # "."  li $v0, 11 # 打印字符  la $a0, '.' # "."  syscall  addi $t6, $t4, 9 # 计算小数点后的位数  sllv $t6, $t3, $t6 # 将结果尾数左移$t6位  # 输出尾数  move $a0, $t6 # 将新的尾数存入$a0  li $a1, 1 # 设置$a1为1  jal convert\_to\_hex # 跳转到convert\_to\_hex  # "\*16^"  li $v0, 4 # 打印字符串  la $a0, string\_to16 # "\*16^"  syscall  # 输出t5  li $v0, 1 # 打印整数  move $a0, $t5 # 将新的尾数存入$a0  syscall  print\_hex\_end:  li $v0, 11 # 换行  li $a0, '\n'  syscall  j menu # 跳转到计算过程  print\_bits: # 要显示的内容存在a1，从第a2位（0开始）开始输出  addi $sp, $sp, -32 # 分配栈空间  sw $t1, 28($sp) # 保存寄存器$t1  sw $t6, 24($sp) # 保存寄存器$t6  sw $ra, 20($sp) # 保存返回地址  sw $fp, 16($sp) # 保存帧指针  addiu $fp, $sp, 28 # 设置新的帧指针    move $t6, $a2 # 将startIndex存入$t6  li $v0, 4 # 打印字符串  bit\_shift\_loop:  srlv $t1, $a1, $t6 # 将内容右移$t6位，存入$t1  andi $t1, $t1, 0x1 # 取$t1的最低位，存入$t1  beqz $t1, print\_zero\_bit # 如果最低位为0，跳转到print\_zero  j print\_one\_bit # 否则跳转到print\_one  print\_zero\_bit:  la $a0, string\_0 # "0"  j print\_bin # 打印0  print\_one\_bit:  la $a0, string\_1 # "1"  j print\_bin # 打印1  print\_bin:  syscall  addi $t6, $t6, -1 # startIndex减1  bgez $t6, bit\_shift\_loop # 如果startIndex大于等于0，跳转到bit\_shift\_loop  # -- 将计算结果转换为16进制 --  convert\_to\_hex: # a0为要输出的数，a1 = 0 时，输出3:0，否则输出31:8  bne $a1, 0, high # 如果a1不为0，跳转到high  low: # 输出3:0  andi $a0, $a0, 0xf # 取$a0的低4位  lb $a0, hex\_table($a0) # 将对应的16进制字符加载到$a0  li $v0, 11 # 打印字符  j convert\_to\_hexEnd # 跳转到convert\_to\_hexEnd  high: # 输出31:8  srl $a0, $a0, 8 # 将$a0右移8位  li $t9, 5 # 设置计数器$t9为5  convert\_to\_hexLoop:  andi $t7, $a0, 0xf # 取$a0的低4位，存入$t7  lb $t8, hex\_table($t7) # 将对应的16进制字符加载到$t8  sb $t8, hex\_digits($t9) # 存储到hex\_digits数组中  sub $t9, $t9, 1 # 计数器$t9减1  srl $a0, $a0, 4 # 将$a0右移4位  bgez $t9, convert\_to\_hexLoop # 如果计数器$t9大于等于0，继续循环  la $a0, hex\_digits # 加载hex\_digits数组地址到$a0  li $v0, 4 # 打印字符串  convert\_to\_hexEnd:  syscall  jr $ra # 返回跳转处  # -- 退出程序 --  exit:  la $a0, msg\_exit # 输出再见语  li $v0, 4  syscall  li $v0, 10 # 退出程序  syscall | | | | | | |
| 四、实验结果及分析和（或）源程序调试过程    将asm文件导入mars中即可运行    运行代码，显示欢迎语和选择界面    输入1选择加法，并输入两个加数，得到结果的二进制和十六进制    输入2选择减法，并输入被减数和减数，得到结果的二进制和十六进制    输入3选择乘法，并输入两个因数，得到结果的二进制和十六进制    输入4选择除法，并输入被除数和除数，得到结果的二进制和十六进制    当不规范输入时，输出Invalid input!    输入5退出  遇到的问题：   1. 想要程序最终实现一个效果，就是用户在menu选择calculate后不断循环calculate模块，也就是说在为用户计算一次后回到calculate页面，而不是回到menu页面，因为大部分情况下用户都需要连续计算几个值，所以这样设计能减少操作的繁琐，提升用户的使用体验。但是一开始按这样设计时，发现重复计算时如果上次计算结果是负值，下一次计算时会带上一个负号。   原因：程序结构有问题，重复计算时存储符号的寄存器值没有被更新。  解决：优化了程序结构，将原本的function模块合并进calculate模块，计算结构正确。   1. 为了解决问题1，将原本的function模块合并进calculate模块来优化整体结构，但改完代码运行后发现执行计算的结果变得很奇怪，不符合预期。   原因：原来是转移代码时将分支判断延后了，而储存用户输入的分支选择的寄存器在分支判断前又被使用赋值了，导致分支判断混乱。  解决：在获取用户输入的分支选择后将其值转移到另一个不用的寄存器中，分支判断正确。   1. 计算器计算加法和减法时，如果结果为0，会直接退出程序。   原因：计算加法和减法时如果计算得出的结果判断为0，会直接跳转到output\_zero模块，而output\_zero最后的跳转是跳转到exit，所以会直接退出程序。  解决：将output\_zero模块中的“j exit”修改为“j calculate”，使其输出0后再次跳转到计算模块。   1. 计算器输出16进制计算结果时，结果为负不会显示负号，但尾数超过7位就会显示符号。   原因：存储计算结果符号位的$t1寄存器在输出2进制结果时被使用到了，并且改变了值，导致在输出16进制结果时判断符号位异常，输出负号异常。  解决：    先将$t1里的值压入栈中，然后在输出16进制结果判断符号位时从栈中取出，保证符号位值不受到改变。 | | | | | | |