版本

更新记录	文档名	课程设计
	版本号	0.1
	创建人	计算机组成原理教学组
	创建日期	2019/12/02

更新历史				
序号	**更新日期**	**更新人**	**版本号**	**更新内容**
1	2019/12/02	杨羽频	0.1	初版

文档错误反馈:yyp@cqu.edu.cn

1 设计目的

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

2 设计条件

PC 机一台，采用 Mars 仿真软件，利用 MIPS 汇编语言中的整数运算指令实现

3 课程设计内容

3.1 课程设计题目

假设没有浮点表示和计算的硬件，使用整数运算的方法实现 IEEE 754 中单精度浮点数的加减乘除运算。

3.2 课程设计要求

1. 要求使用 MIPS 汇编指令，但是不能直接使用浮点指令，只能利用整数运算指令来编写软件完成。
2. 将实现的代码放入测试框架指定位置（见后文说明），能够通过自动化测试。
3. 可实现浮点数的加减 (或者乘除) 运算。
4. 提交一份课程设计报告，至少应包含以下部分：
    1. 功能实现方案
    2. 详细设计，包括：指令格式，编码，程序流程图等
    3. 测试
    4. 结论
    5. 参考文献

3.3 课程设计步骤

1. 将源代码第 24 行的 jal getmul 中的 getmul 改为 getadd getsub getmul getdiv 这几个中的任意一个，它们分别代表为加减乘除
2. 在 31 行开始的 yourfunc: 到 35 行的 jr $ra 之间，填入自己所需要实现功能的相关汇编代码，完成功能的实现。其中，相关汇编器的要求为：两个浮点数已经提供并且放在 $a0,$a1 中，将计算结果放入 $a2 中，临时寄存器可以随意修改，其他寄存器改了请恢复。

3.4 测试结果

经过测试框架测试之后得到的测试结果显示应该如下:

```
Hello
Good:100
Total:100
Thank you Bye
```

其中 Good 代表通过测试的记录数,Total 代表测试的总次数。

4 设计教程

4.1 数据类型

1. MIPS 使用定长指令,所有指令都是 32 位长的
2. 1 字节 =8 位,半字长 =2 个字节(32 位),1 字长 =4 个字节
3. 一个字符空间 =1 个字节
4. 一个整形 = 一个字长 =4 个字节
5. 单个字符用单引号
6. 字符串用双引号

4.2 寄存器

MIPS 下一共有 32 个通用寄存器。在汇编中,寄存器标志以 $ 开头,寄存器表示可以有两种方式:

1. 直接使用该寄存器对应的编号,例如:从 031
2. 使用对应的寄存器名称,例如 t1,t1,sp,详见下文

对于乘法和除法分别有对应的两个寄存器 lo,hi。对于以上两者,不存在直接寻址,必须通过 mfhi 和 mflo 分别来进行访问对应的内容。栈的走向是从高地址向低地址

4.2.1 MIPS 下各个寄存器编号及描述

REGISTER	NAME	USAGE
$0	$zero	常量 0(constant value 0)
$1	$at	保留给汇编器 (Reserved for assembler)
2−3	*v*0−v1	函数调用返回值 (values for results and expression evaluation)
4−7	*a*0−a3	函数调用参数 (arguments)
8−15	*t*0−t7	暂时的 (或随便用的)
16−23	*s*0−s7	保存的 (或如果用,需要 SAVE/RESTORE 的)(saved)
24−25	*t*8−t9	暂时的 (或随便用的)
$28	$gp	全局指针 (Global Pointer)
$29	$sp	堆栈指针 (Stack Pointer)
$30	$fp	帧指针 (Frame Pointer)
$31	$ra	返回地址 (return address)

4.3 程序结构

本质就是数据声明 + 普通文本 + 程序编码（文件后缀为.s 或.asm），数据声明在代码段之后（在之前也没什么问题）。

4.3.1 数据申明

数据段以.data 为开始标志。声明变量后，即在主存中分配空间。

4.3.2 代码

代码段以.text 为开始标志，程序入门为 main: 标志。

```
# Comment giving name of program and description of function
# 说明下程序的目的和作用（其实和高级语言都差不多了）
# Template.s
#Bare-bones outline of MIPS assembly language program
.data                    # variable declarations follow this line
                         # 数据变量声明
                         # ...

.text                    # instructions follow this line
                         # 代码段部分
main:                   # indicates start of code (first instruction to execute)
                        # 主程序
                        # ...
# End of program, leave a blank line afterwards to make SPIM happy
```

4.4 数据的装载和保存（Load/Store 指令）

主存（RAM）的存取 access 只能用 load / store 指令来完成。常见指令如下：

```
lw
lw register_destination, RAM_source
#从内存中复制RAM_source的内容到对应的寄存器中（w意味word，即该数据大小为4个字
    节）

lb
lb register_destination, RAM_source
#同上，lb为load byte

sw
```

```
sw register_source, RAM_destination
#指将指定寄存器中的数据写入到特定的内存中

sb
sb register_source, RAM_destination
#同lb
```

4.5 算术运算指令

算术运算简称运算。指按照规定的法则和顺序对式题或算式进行运算，并求出结果的过程。包括：加法、减法、乘法、除法等几种运算形式。

在 MIPS 汇编指令集中，算术指令最多有三个操作数，操作数只能是寄存器，不允许出现地址，并且所有指令统一是 32 位。常见指令如下：

```
add $t0,$t1,$t2      # $t0 = $t1 + $t2; add as signed (2's complement) integers
sub $t2,$t3,$t4      # $t2 = $t3 - $t4
addi     $t2,$t3, 5  #  $t2 = $t3 + 5;   "add immediate" (no sub immediate)
addu     $t1,$t6,$t7 #  $t1 = $t6 + $t7;   add as unsigned integers
subu     $t1,$t6,$t7 #  $t1 = $t6 + $t7;   subtract as unsigned integers
mult     $t3,$t4     #  multiply 32-bit quantities in $t3 and $t4,
                     #  and store 64-bit
                     #  result in special registers Lo and Hi:  (Hi,Lo) = $t3 *
                         $t4
                     #  运算结果存储在hi,lo（hi高位数据，  lo低位数据）
div $t5,$t6          #  Lo = $t5 / $t6   (integer quotient)
                     #  Hi = $t5 mod $t6   (remainder)
                     #  商数存放在 lo，余数存放在 hi
mfhi     $t0         #  move quantity in special register Hi to $t0:   $t0 = Hi
                     #  不能直接获取 hi 或 lo中的值，  需要mfhi，
                     #  mflo指令传值给寄存器
mflo     $t1         #  move quantity in special register Lo to $t1:   $t1 = Lo
                     #  used to get at result of product or quotient
move     $t2,$t3     #  $t2 = $t3
```

4.6 控制流

4.6.1 分支(if else 系列)

分支指令会在满足一定条件的情况下，进行跳转。常见指令如下：

```
b    target          #  unconditional branch to program label target
beq $t0,$t1,target   #  branch to target if   $t0 = $t1
blt $t0,$t1,target   #  branch to target if   $t0 < $t1
```

```
ble $t0,$t1,target #  branch to target if  $t0 <= $t1
bgt $t0,$t1,target #  branch to target if  $t0 > $t1
bge $t0,$t1,target #  branch to target if  $t0 >= $t1
bne $t0,$t1,target #  branch to target if  $t0 <> $t1
```

4.6.2 跳转(while,for,goto 系列)

跳转指令会令当前运行地址无条件的跳转到指定位置。常见指令如下:

```
j   target          #  unconditional jump to program label target
                              看到就跳，  不用考虑任何条件
jr  $t3   #  jump to address contained in $t3 ("jump register")
                              类似相对寻址，跳到该寄存器给出的地址处
```

4.6.3 子程序调用

子程序调用会调用指定的程序并且在执行结束后返回到当前指令。常见指令如下:

```
jal sub_label   #  "jump and link"
```

4.7 系统调用与输入/输出

系统调用指令需要使用 syscall, 参数所使用的寄存器:$v0,$a0,$a1, 返回值使用:$v0。常见系统调用指令为:

Service	Code	Arguments	Result
print integer	1	$a0 = integer to print	
print float	2	$f12 = float to print	
print double	3	$f12 = double to print	
print string	4	$a0 = address of null-terminated string to print	
read integer	5		$v0 contains integer read
read float	6		$f0 contains float read
read double	7		$f0 contains double read
read string	8	$a0 = address of input buffer $a1 = maximum number of characters to read	See note below table
sbrk (allocate heap memory)	9	$a0 = number of bytes to allocate	$v0 contains address of allocated memory
exit (terminate execution)	10		
print character	11	$a0 = character to print	See note below table
read character	12		$v0 contains character read

5 C 语言到汇编的过程

5.1 环境安装

本实验是在 Ubuntu 16.04 x64 环境下，使用 mips 交叉编译工具进行编译。其中，mips 交叉编译工具安装的参考命令如下：

```
sudo apt-get update
sudo apt-get install emdebian-archive-keyring
sudo apt-get install linux-libc-dev-mips-cross
sudo apt-get install libc6-mips-cross libc6-dev-mips-cross
sudo apt-get install binutils-mips-linux-gnu gcc-5-mips-linux-gnu
sudo apt install gcc-mips-linux-gnu
```

5.2 编译

示例代码：

```
#include<stdio.h>
int main()
{
    int a=1,b=2,c=0 ;
```

```
    c=a;
    a=b;
    b=c;
    return 0;
}
```

编译命令:

mips-linux-gnu-gcc -S 2.c -mips32 -fno-stack-protector -static

会在目录下生成 mips 汇编文件 2.S 其核心代码为:

```
main:
        .frame  $fp,32,$31              # vars= 16, regs= 1/0, args= 0, gp= 8
        .mask   0x40000000,-4
        .fmask  0x00000000,0
        .set    noreorder
        .set    nomacro
        addiu   $sp,$sp,-32
        sw      $fp,28($sp)
        move    $fp,$sp
        li      $2,1                    # 0x1
        sw      $2,8($fp)
        li      $2,2                    # 0x2
        sw      $2,12($fp)
        sw      $0,16($fp)
        lw      $2,8($fp)
        sw      $2,16($fp)
        lw      $2,12($fp)
        sw      $2,8($fp)
        lw      $2,16($fp)
        sw      $2,12($fp)
        move    $2,$0
        move    $sp,$fp
        lw      $fp,28($sp)
        addiu   $sp,$sp,32
        j       $31
        nop
```

注意 j $31 这样的指令在仿真的时候,请写为 jr $31