#Runs through many pairs of numbers, and does each operation on them, checking if the answer matches with the FPU

.data

op1Arr: .float 2, 2, 0, 0, -1, 4, 9, 0.2354, 4.35, 10000, 99999999

op2Arr: .float 2, 0, 2, 0, -1, 2, 2, -4.234, 14.3, 0.1 , 2

opArrSize: .word 11

new\_line\_string:.asciiz "\n"

tab\_string: .asciiz "\t"

.macro

(SAME AS IN FloatingPointSimulator.asm)

.end\_macro

Testing.asm

.text

lw $s0, opArrSize

li $s1, 0 #loop counter

la $s2, op1Arr #op1 mem location

la $s3, op2Arr #op2 mem location

#store these values on the stack

addi $sp, $sp, -16

sw $s0, 12($sp)

sw $s1, 8($sp)

sw $s2, 4($sp)

sw $s3, 0($sp)

loop:

#refresh variables from last loop

lw $s0, 12($sp)

lw $s1, 8($sp)

lw $s2, 4($sp)

lw $s3, 0($sp)

beq $s0, $s1, loop\_end

addi $s1, $s1, 1 #incriment counter before stack is changed

sw $s1, 8($sp) #save new value on stack

lw $t0, 0($s2) #load op1

addi $s2, $s2, 4 #incriment to next mem location

sw $s2, 4($sp) #store mem location of op1Arr

lw $t1, 0($s3) #load op2

addi $s3, $s3, 4 #incriment to next mem location

sw $s3, 0($sp) #store mem location of op2Arr

addi $sp, $sp, -48 #push stack down

sw $t0, 44($sp) #store op1 on stack

sw $t1, 40($sp) #store op2 on stack

#get parts

lw $a0, 44($sp) #going to call get\_sign

jal get\_sign #get sign of op1

sw $v0, 36($sp) #store sign of op1

lw $a0, 44($sp) #going to call get\_exponent

jal get\_exponent #get exponent of op1

sw $v0, 32($sp) #store exp of op1

lw $a0, 44($sp) #setting up to call get\_fraction

jal get\_fraction #get fraction of op1

sw $v0, 28($sp) #store fraction of op1

lw $a0, 40($sp) #going to call get\_sign

jal get\_sign #get sign of op2

sw $v0, 24($sp) #store sign of op2

lw $a0, 40($sp) #going to call get\_exponent

jal get\_exponent #get exponent of op2

sw $v0, 20($sp) #store exp of op2

lw $a0, 40($sp) #setting up to call get\_fraction

jal get\_fraction #get fraction of op2

sw $v0, 16($sp) #store fraction of op2

#call add

lw $v0, 36($sp) #op1 sign

lw $v1, 32($sp) #op1 exp

lw $a0, 28($sp) #op1 fraction

lw $a1, 24($sp) #op2 sign

lw $a2, 20($sp) #op2 exp

lw $a3, 16($sp) #op2 fraction

li $t8, 1

jal add\_fp

sw $v0, 12($sp) #store calc add

#call subtract

lw $v0, 36($sp) #op1 sign

lw $v1, 32($sp) #op1 exp

lw $a0, 28($sp) #op1 fraction

lw $a1, 24($sp) #op2 sign

lw $a2, 20($sp) #op2 exp

lw $a3, 16($sp) #op2 fraction

li $t8, 2

jal sub\_fp

sw $v0, 8($sp) #store calc sub

#call multiply

lw $v0, 36($sp) #op1 sign

lw $v1, 32($sp) #op1 exp

lw $a0, 28($sp) #op1 fraction

lw $a1, 24($sp) #op2 sign

lw $a2, 20($sp) #op2 exp

lw $a3, 16($sp) #op2 fraction

li $t8, 3

jal mult\_fp

sw $v0, 4($sp) #store calc mult

#call divide

lw $v0, 36($sp) #op1 sign

lw $v1, 32($sp) #op1 exp

lw $a0, 28($sp) #op1 fraction

lw $a1, 24($sp) #op2 sign

lw $a2, 20($sp) #op2 exp

lw $a3, 16($sp) #op2 fraction

li $t8, 4

li $t0, 0

li $t1, 0

jal div\_fp

sw $v0, 0($sp) #store calc div

#calculate actual answers using FP Unit

lw $t1, 44($sp) #load op1

mtc1 $t1, $f1 #store op1 in FPU

lw $t2, 40($sp) #load op2

mtc1 $t2, $f2 #store op2 in FPU

add.s $f4, $f1, $f2

sub.s $f5, $f1, $f2

mul.s $f6, $f1, $f2

div.s $f7, $f1, $f2

mfc1 $s4, $f4 #move answers to regular registers

mfc1 $s5, $f5 #move answers to regular registers

mfc1 $s6, $f6 #move answers to regular registers

mfc1 $s7, $f7 #move answers to regular registers

#restore calculated answers from stack

lw $s0, 12($sp)

lw $s1, 8($sp)

lw $s2, 4($sp)

lw $s3, 0($sp)

#restore operands

lw $t8, 44($sp)

lw $t9, 40($sp)

# $s0 = calc add

# $s1 = calc sub

# $s2 = calc mult

# $s3 = calc div

# $s4 = actual add answer

# $s5 = actual sub answer

# $s6 = actual mult answer

# $s7 = actual div answer

PRINT\_FLOAT($t8)

PRINT\_STR(" + ")

PRINT\_FLOAT($t9)

PRINT\_STR(" = \t")

PRINT\_FLOAT($s0)

beq $s0, $s4, goodAdd #check for equal answers

PRINT\_STR("\tActual answer: ")

PRINT\_FLOAT($s4)

PRINT\_STR("\tINCORRECT")

goodAdd:

PRINT\_NEWLINE

PRINT\_FLOAT($t8)

PRINT\_STR(" - ")

PRINT\_FLOAT($t9)

PRINT\_STR(" = \t")

PRINT\_FLOAT($s1)

beq $s1, $s5, goodSub #check for equal answers

PRINT\_STR("\tActual answer: ")

PRINT\_FLOAT($s5)

PRINT\_STR("\tINCORRECT")

goodSub:

PRINT\_NEWLINE

PRINT\_FLOAT($t8)

PRINT\_STR(" \* ")

PRINT\_FLOAT($t9)

PRINT\_STR(" = \t")

PRINT\_FLOAT($s2)

beq $s2, $s6, goodMult #check for equal answers

PRINT\_STR("\tActual answer: ")

PRINT\_FLOAT($s6)

PRINT\_STR("\tINCORRECT")

goodMult:

PRINT\_NEWLINE

PRINT\_FLOAT($t8)

PRINT\_STR(" / ")

PRINT\_FLOAT($t9)

PRINT\_STR(" = \t")

PRINT\_FLOAT($s3)

beq $s3, $s7, goodDiv #check for equal answers

PRINT\_STR("\tActual answer: ")

PRINT\_FLOAT($s7)

PRINT\_STR("\tINCORRECT")

goodDiv:

PRINT\_NEWLINE

PRINT\_NEWLINE

addi $sp, $sp, 48 #restore stack

j loop

loop\_end:

li $v0, 10 #end program

syscall

#functions to use below

(ALL FUNCTIONS OMITTED BECAUSE THEY ARE THE SAME AS IN FloatingPointSimulator)

add\_fp:

j sub\_fp

#end add

sub\_fp:

(same as in FloatingPointSimulator)

#end sub

mult\_fp:

(same as in FloatingPointSimulator)

#end mult

div\_fp:

(same as in FloatingPointSimulator)

#end div

get\_fraction:

(same as in FloatingPointSimulator)

#end get\_fraction

get\_exponent:

(same as in FloatingPointSimulator)

#end get\_exponent

get\_sign:

(same as in FloatingPointSimulator)

#end get\_sign

recombine\_fp:

(same as in FloatingPointSimulator)

#end recombine\_fp