Universidade Fernando Pessoa

Arquitectura de Computadores
Ficha de Exercícios nº5

Objectivos:

- Representação em vírgula flutuante. Utilização das instruções do co-processador matemático da arquitectura do MIPS para implementar programas simples.

1. Dado o padrão de bits: 1010 1101 0001 0000 0000 0000 0000 0010, qual o seu significado, assumindo que se trata de:
   a. Um inteiro em complemento para 2 (calcule o seu valor em decimal)
   b. Um inteiro sem sinal (calcule o seu valor em decimal)
   c. Um número em virgula flutuante (precisão simples)
   d. Uma instrução do MIPS

2. Determine a representação na norma IEEE754 (precisão simples) dos seguintes valores decimais:
   a. 20.5_{10}
   b. $(-5/6)_{10}$

3. Qual a representação decimal dos seguintes valores no formato IEEE754:
   a. 0100 0110 1101 1000 0000 0000 0000 0000
   b. 1011 1110 1110 0000 0000 0000 0000 0000

4. Considere o seguinte código MIPS:

```
        ############################################################
          .data
MSG1:     .asciiz "Quantos valores pretende introduzir? => "
MSG2:     .asciiz "Indique um valor => "
MSG3:     .asciiz "A soma dos valores positivos => "
ZERO:     .float 0.0

          .text
          .globl  main

main:     li $v0,4              # Imprime MSG1
          la $a0,MSG1
          syscall
          li $v0,5              # lê o número de valores
          syscall
          move $s0,$v0          # guarda-o em $s0
          mtc1 $s0,$f1          # copia $S0 para $f1 no coproce. 1
          cvt.s.w $f1, $f1      # converte o INT em $f1 para VF em $f1
          lwc1 $f4,ZERO         # $f4=M[ZERO] em VF
          lwc1 $f2,ZERO         # $f2=M[ZERO] em VF
LOOP:     beq $s0,$zero,DONE    # Ciclo para ler os valores

          li $v0,4              # Imprime MSG2
          la $a0,MSG2
          syscall
          li $v0,6              # Chamada para ler um "float"
          syscall
          c.lt.s $f0, $f4       # Se $f0<($f4=0)a flag0 fica a 1
          bc1t NEXT             # Se a flag0 está a 1 salta para NEXT
```

```
        add.s $f2,$f2,$f0     # Se $f0 é positivo, soma-se a $f2
NEXT:   addi $s0,$s0,-1
        j LOOP                # ler o próximo...
DONE:   li $v0,4              # Imprime MSG3
        la $a0,MSG3
        syscall
        li $v0,2              # imprimir o "float" com o resultado
        mov.s $f12, $f2       # copia o valor para o $f12
        syscall
        li $v0,10             # Chamada 10; exit
        syscall
```

a. Execute-o no SPIM, analise as instruções que desconhece.
b. Escreva um programa que leia ***n*** números em vírgula flutuante do terminal e determine o maior, o menor e a média dos valores introduzidos.

5. Considere o seguinte código em linguagem C. Escreva um programa equivalente em assembly do MIPS. Tem que usar aritmética de vírgula flutuante e o coprocessador matemático.

```
#include <stdio.h>
/* print Fahrenheit-Celsius table for fahr = 0, 20, ..., 300; floating-
point version */

void main() {
      float fahr, celsius;
      float lower, upper, step;
      lower = 0.0;      /* lower limit of temperature scale */
      upper = 300.0;    /* upper limit */
      step = 20.0;      /* step size */
      fahr = lower;
      while (fahr <= upper) {
            celsius = (5.0/9.0) * (fahr-32.0);
            printf("%3.0f %6.1f\n", fahr, celsius);
            fahr = fahr + step;
      }
}
```

6. O valor da função exponencial (e^x), na vizinhança do ponto x=0 pode ser calculado pela fórmula de Taylor:

$$e^x = \sum_{n=0}^{+\infty} \frac{x^n}{n!}$$

Crie um programa em assembly do MIPS para calcular o valor da função exponencial e compare o valor calculado com o que obtém na sua calculadora. O seu programa tem que implementar duas funções:

a. power(x,n) – calcula x elevado a n
b. factorial(n) – calcula o factorial de n

O programa recebe como entrada o valor de x (o argumento da exponencial) e o valor de n (ordem até à qual devemos somar a série de Taylor) e imprime para o terminal o valor calculado. Use sempre aritmética de vírgula flutuante no programa.

Bibliografia:

[1] Patterson & Hennessy – *Computer Organization and Design: The hardware/software interface 4rd Ed* – MKP 2009.
[2] SGI, "MIPSproTM Assembly Language Programmer's Guide", doc: 007-2418-006, published in: 2003-08-15

Nota: deverá explorar a utilização de ferramentas como, por exemplo, as da *GNU Binary Utilities* ou *binutils* que incluem os seguintes comandos:[1]

Principais

`as`	assembler
`ld`	linker

Adicionais

`gprof`	profiler
`addr2line`	convert address to file and line
`ar`	create, modify, and extract from archives
`c++filt`	demangling filter for C++ symbols
`dlltool`	creation of Windows dynamic-link libraries
`nlmconv`	object file conversion to a NetWare Loadable Module
`nm`	list symbols in object files
`objcopy`	copy object files, possibly making changes
`objdump`	dump information about object files (it can be used as a disassembler to view executable in assembly form)
`ranlib`	generate indexes for archives
`readelf`	display content of ELF files
`size`	list total and section sizes
`strings`	list printable strings
`strip`	remove symbols from an object file
`windmc`	generates Windows message resources
`windres`	compiler for Windows resource files

Register Name	Software Name (from `fgregdef.h`)	Use and Linkage
$f0..$f2	fv0-fv1	Hold results of floating-point type function ($f0) and complex type function ($f0 has the real part, $f2 has the imaginary part.
$f4..$f10	ft0-ft3	Temporary registers, used for expression evaluation whose values are not preserved across procedure calls.
$f12..$f14	fa0-fa1	Pass the first two single- or double-precision actual arguments; their values are not preserved across procedure calls.
$f16..$f18	ft4-ft5	Temporary registers, used for expression evaluation, whose values are not preserved across procedure calls.
$f20..$f30	fs0-fs5	Saved registers, whose values must be preserved across procedure calls.

Ilustração 1 – Registos floating point e suas convenções de utilização (ver [2])

[1] http://www.gnu.org/software/binutils/

University Fernando Pessoa

Computer Architecture

Exercise sheet nº5

Goals:

- Floating point representation (IEEE-754) and MIPS FPU programming

1. Giving the bit pattern: 1010 1101 0001 0000 0000 0000 0000 0010, what does it means, assuming that is:
    a. One integer in two's complement (calculate its value in decimal)
    b. One unsigned number (calculate its value in decimal)
    c. One floating point number in IEEE-754 (single precision)
    d. One MIPS instruction

2. Determine the IEEE754 representation (single precision) of the following decimal values:
    a. 20.5_{10}
    b. $(-5/6)_{10}$

3. What is the decimal representation of the following IEEE754 values:
    a. 0100 0110 1101 1000 0000 0000 0000 0000
    b. 1011 1110 1110 0000 0000 0000 0000 0000

4. Consider the following MIPS code:

```
       ############################################################
         .data
MSG1:    .asciiz "How many values to introduce => "
MSG2:    .asciiz "Insert one value => "
MSG3:    .asciiz "Sum of positive numbers is => "
ZERO:    .float 0.0

         .text
         .globl  main

main:    li $v0,4              # print MSG1
         la $a0,MSG1
         syscall
         li $v0,5              # read number of values
         syscall
         move $s0,$v0          # store in $s0
         mtc1 $s0,$f1          # copy $S0 to $f1 in coproc 1
         cvt.s.w $f1, $f1      # converts INT in $f1 to VF in $f1
         lwc1 $f4,ZERO         # $f4=M[ZERO] in VF
         lwc1 $f2,ZERO         # $f2=M[ZERO] in VF
LOOP:    beq $s0,$zero,DONE
         li $v0,4              # print MSG2
         la $a0,MSG2
         syscall
         li $v0,6              # read"float"
         syscall
         c.lt.s $f0, $f4       # if $f0<($f4=0) set flag0=1
         bc1t NEXT             # if flag0 == 1 jump to NEXT
         add.s $f2,$f2,$f0     # if $f0 is positive, sum to $f2
NEXT:    addi $s0,$s0,-1
         j LOOP                # read next value...
DONE:    li $v0,4              # print MSG3
         la $a0,MSG3
```

```
        syscall
        li $v0,2              # print float as result
        mov.s $f12, $f2       # copy value to $f12
        syscall
        li $v0,10             # exit
        syscall
```

a. Execute it in the simulator (MARS or SPIM) and analyze the unknown instructions.
b. Write an assembly program to read ***n*** floating point numbers and determines the greatest, the smallest, and the average of the introduced values.

5. Consider the following C code. Write an equivalent program in MIPS assembly. Use floating point arithmetic and the math coprocessor in your resolution.

```
#include <stdio.h>
/* print Fahrenheit-Celsius table for fahr = 0, 20, ..., 300; floating-
point version */

void main() {
      float fahr, celsius;
      float lower, upper, step;
      lower = 0.0;      /* lower limit of temperature scale */
      upper = 300.0;    /* upper limit */
      step = 20.0;      /* step size */
      fahr = lower;
      while (fahr <= upper) {
            celsius = (5.0/9.0) * (fahr-32.0);
            printf("%3.0f %6.1f\n", fahr, celsius);
            fahr = fahr + step;
      }
}
```

6. The value of the exponential function (e^x), in the neighborhood of x=0 can be calculated using the Taylor formula:

$$e^x = \sum_{n=0}^{+\infty} \frac{x^n}{n!}$$

Create a MIPS assembly program to calculate the value of the exponential function and compare the calculated value with the one given by your scientific calculator. Your program should implement the two following functions:

a. power(x,n) – calculates x raisen to n
b. factorial(n) – calculates factorial of n

The program receives, as input, value x (the argument of the exponential) and value n (the order to be used in the Taylor series sum) and should print the calculated value. Use always floating point arithmetic.

Bibliography:

[1] Patterson & Hennessy – *Computer Organization and Design: The hardware/software interface 4rd Ed* – MKP 2009.
[2] SGI, "MIPSproTM Assembly Language Programmer's Guide", doc: 007-2418-006, published in: 2003-08-15

Note: you should explore the *GNU Binary Utilities* or *binutils* including the command line tools:[2]

Main tools

`as`	assembler
`ld`	linker

Additional tools

`gprof`	profiler
`addr2line`	convert address to file and line
`ar`	create, modify, and extract from archives
`c++filt`	demangling filter for C++ symbols
`dlltool`	creation of Windows dynamic-link libraries
`nlmconv`	object file conversion to a NetWare Loadable Module
`nm`	list symbols in object files
`objcopy`	copy object files, possibly making changes
`objdump`	dump information about object files (it can be used as a disassembler to view executable in assembly form)
`ranlib`	generate indexes for archives
`readelf`	display content of ELF files
`size`	list total and section sizes
`strings`	list printable strings
`strip`	remove symbols from an object file
`windmc`	generates Windows message resources
`windres`	compiler for Windows resource files

Register Name	Software Name (from `fgregdef.h`)	Use and Linkage
$f0..$f2	fv0-fv1	Hold results of floating-point type function ($f0) and complex type function ($f0 has the real part, $f2 has the imaginary part.
$f4..$f10	ft0-ft3	Temporary registers, used for expression evaluation whose values are not preserved across procedure calls.
$f12..$f14	fa0-fa1	Pass the first two single- or double-precision actual arguments; their values are not preserved across procedure calls.
$f16..$f18	ft4-ft5	Temporary registers, used for expression evaluation, whose values are not preserved across procedure calls.
$f20..$f30	fs0-fs5	Saved registers, whose values must be preserved across procedure calls.

Ilustração 2 – Floating point registers and calling conventions (see [2])

[2] http://www.gnu.org/software/binutils/