Universidade Fernando Pessoa

Arquitectura de Computadores
Ficha de Exercícios nº2

Objectivos:

- Compreender o funcionamento do simulador do MIPS, o MARS.
- Criar pequenos programas em assembly do MIPS que façam uso das chamadas ao sistema, das operações aritméticas, da memória, que incluam ciclos e operações lógicas.

1. Considere o programa seguinte:

```
## O seu primeiro programa em assembly do MIPS
## Note o estilo: 3 colunas:
## (1) "Labels", opcional
## (2) Instruções e seus operandos
## (3) comentários, opcional

      .data                         # Zona estática de dados

str:  .asciiz "Olá Mundo!\n"        # Uma "label", str, e uma string
                                    # terminda pelo carácter '\0'
                                    # usando a directiva .asciiz

      .text                         # Zona de código
      .globl main                   # declara `main' como um símbolo
                                    # global

main:
      li $v0, 4                     # Código para a chamada ao sistema
                                    # print_str
      la $a0, str                   # Endereço da string a imprimir
      syscall                       # Executa a chamada
Exit:
      li $v0, 10                    # Código para a chamada ao sistema
                                    # "Exit"
      syscall                       # Executa a chamada
```

   a. Crie um ficheiro com o editor de texto incluído dentro do simulador MARS contendo o programa apresentado. Procure no manual do MARS o significado dos símbolos desconhecidos.
   b. Execute o programa depois de o assemblar.
   c. Leia as páginas do apêndice "Assemblers, Linkers and SPIM Simulator" do livro da cadeira, disponível em PDF na pasta prática/ferramentas no e-learning, de forma a conhecer a organização do simulador de MIPS e as chamadas ao sistema disponíveis.

2. Considere o seguinte programa.

```
        .data
MSG1:   .asciiz "Indique o primeiro número => "
MSG2:   .asciiz "Indique o segundo número => "
SUM:    .asciiz "Soma = "
SUB:    .asciiz "Diferença = "

        .align 2              #
```

```
INPUT1: .space 4                #
INPUT2: .space 4                #

        .text
        .globl  main

main:   li      $v0,4           #
        la      $a0, MSG1
        syscall
        li      $v0,5           #
        syscall
        la      $t5, INPUT1
        sw      $v0, 0($t5)     #
        li      $v0,4           #
        la      $a0, MSG2
        syscall
        li      $v0,5           #
        syscall
        la      $t6, INPUT2
        sw      $v0, 0($t6)     #
        lw      $t0, 0($t5)     #
        lw      $t1, 0($t6)
        li      $v0,4           #
        la      $a0, SUM
        syscall
        add     $t3, $t0, $t1   #
        li      $v0,1           #
        move    $a0, $t3        #
        syscall
        li      $v0,4           #
        la      $a0, SUB
        syscall
        sub     $t4, $t0, $t1   #
        li      $v0,1           #
        move    $a0, $t4        #
        syscall
        li      $v0,10          #
        syscall
```

a. Crie um ficheiro no editor de texto com este programa acrescentando em frente dos símbolos # o comentário referente à instrução correspondente.

b. O `li` e o `move` são pseudo – instruções, isto é, são instruções que não existem no conjunto de instruções do MIPS, mas que são usadas para facilitar a escrita/leitura de programas. Como podem estas duas ser implementadas à custa das instruções nativas do MIPS que conhece até ao momento?

c. Execute-o passo a passo explicitando os valores dos registos que vão sendo alterados.

3. Escreva um programa que, dado um número inteiro ***n***, calcule a soma 1+2+3+...+ ***n***, e imprima o resultado para terminal.

4. Dados os programas seguintes em linguagem C escreva o seu equivalente em assembler do MIPS.

a.

```
int A[]={1,3,5,7,9,11,13,15};

main() {
      int i=0, s=0;
      for (i=0; i<8; i++) {
            s=s+A[i];
      }
      printf("O valor final é %d\n",s);
}
```

b.

```
int A[]={1,3,5,7,9,11,13,15};

main() {
      int i=0, s=0;
      while (i<8) {
            s=s+A[i];
            i=i+1;
      }
      printf("O valor final é %d\n",s);
}
```

c.

```
int A[]={1,3,5,7,9,11,13,15};

main() {
      int i=0, s=0;
      do {
            s=s+A[i];
            i=i+1;
      } while(i<8);
      printf("O valor final é %d\n",s);
}
```

d.

```
main() {
      int i, j, sum=0;
      for (i=0;i<20;i+=2)
            for (j=0;j<5;j++)
                  sum+=i-j;
      printf("O valor final é %d\n",sum);
}
```

5. Escreva um programa que, dado um número inteiro ***n*** (menor do que 20), leia ***n*** números do terminal e os guarde em posições consecutivas de memória a partir da "label" **Vector**. No fim o programa deve imprimir no terminal a diferença entre a soma dos números de índice par e a soma dos números de índice ímpar do vector.

   Exemplo: vector (1,3,5,4,2,4,1) resultado (1+5+2+1) - (3+4+4) = 9 – 11 = -2

6. Escreva um programa que, dado um número inteiro ***n***, receba ***n*** números do terminal, calcule a soma dos números positivos e imprima o resultado.

Bibliografia:

[1] Patterson & Hennessy – *Computer Organization and Design: The hardware/software interface 4rd Ed* – MKP 2009.

University Fernando Pessoa

Computer Architecture

Exercise sheet nº2

Goals:

- Understand the working of the MIPS simulator MARS
- Create small MIPS assembly programs that made use of system calls, arithmetic operations, memory, cycles and logic operations

1. Consider the following MIPS assembly program:

```
## Your first MIPS assembly program
## Observe the 3 column style:
## (1) Labels, optional
## (2) Instructions and operands
## (3) Comments, optional

      .data                         # Static memory

str:  .asciiz "Hello World!\n"      # Label, str, and one string
                                    # null terminated '\0'
                                    # using directive .asciiz

      .text                         # Text segment
      .globl main                   # declares label 'main'
                                    # as a global symbol

main:
      li $v0, 4                     # "print_str" System call code
      la $a0, str                   # Address of string to print
      syscall                       # Execute system call
Exit:
      li $v0, 10                    # "Exit" System call code
      syscall                       # Execute system call
```

   a. Assemble and execute (run) the program inside the MARS simulator. Read the MARS simulator manual to better understand the previous program.
   b. Read the Appendix "Assemblers, Linkers and SPIM Simulator" from the course text book "Computer Organization and Design"

2. Consider the following MIPS assembly program.

```
        .data
MSG1:   .asciiz "Introduce the first number => "
MSG2:   .asciiz "Introduce the second number => "
SUM:    .asciiz "Sum = "
SUB:    .asciiz "Difference = "

        .align 2              #

INPUT1: .space 4              #
INPUT2: .space 4              #

        .text
        .globl  main
```

```
main:   li      $v0,4           #
        la      $a0, MSG1
        syscall
        li      $v0,5           #
        syscall
        la      $t5, INPUT1
        sw      $v0, 0($t5)     #
        li      $v0,4           #
        la      $a0, MSG2
        syscall
        li      $v0,5           #
        syscall
        la      $t6, INPUT2
        sw      $v0, 0($t6)     #
        lw      $t0, 0($t5)     #
        lw      $t1, 0($t6)
        li      $v0,4           #
        la      $a0, SUM
        syscall
        add     $t3, $t0, $t1   #
        li      $v0,1           #
        move    $a0, $t3        #
        syscall
        li      $v0,4           #
        la      $a0, SUB
        syscall
        sub     $t4, $t0, $t1   #
        li      $v0,1           #
        move    $a0, $t4        #
        syscall
        li      $v0,10          #
        syscall
```

a. Create the previous program in the MIPS simulator. For each instruction, complete the program with the comments after # symbol in each line.
b. The `li` and `move` are pseudo instructions. They are not MIPS native instructions. They are instructions implemented using native instructions, to simplify the writing/reading of assembly programs. How can these two pseudo instructions be implemented using the native MIPS instructions that you know already?
c. Execute the program, step by step, inside the MARS simulator, and explain the changing values in the MIPS registers.

3. Write, using the MIPS simulator, a MIPS assembly program that, given an integer number ***n***, calculates the sum 1+2+3+...+ ***n***, and print the result in the terminal
4. Consider the following C programs. Convert each to an equivalent program in MIPS assembly.
    a.

a.

```
int A[]={1,3,5,7,9,11,13,15};

main() {
      int i=0, s=0;
      for (i=0; i<8; i++) {
            s=s+A[i];
      }
      printf("O valor final é %d\n",s);
```

```
}
```

b.

```
int A[]={1,3,5,7,9,11,13,15};

main() {
      int i=0, s=0;
      while (i<8) {
            s=s+A[i];
            i=i+1;
      }
      printf("O valor final é %d\n",s);
}
```

c.

```
int A[]={1,3,5,7,9,11,13,15};

main() {
      int i=0, s=0;
      do {
            s=s+A[i];
            i=i+1;
      } while(i<8);
      printf("O valor final é %d\n",s);
}
```

d.

```
main() {
      int i, j, sum=0;
      for (i=0;i<20;i+=2)
            for (j=0;j<5;j++)
                  sum+=i-j;
      printf("O valor final é %d\n",sum);
}
```

5. Write, using the MIPS simulator, a MIPS assembly program that, given an integer number ***n*** (smaller than 20), reads ***n*** numbers from the terminal and stores those numbers in consecutive positions in memory after the label **Vector**. At the end, the program should print in the terminal the difference between the sum of the vector numbers with even index and the vector numbers with odd index such as is shown in the example bellow.
   Example: vector (1,3,5,4,2,4,1) result (1+5+2+1) - (3+4+4) = 9 – 11 = -2

6. Write, using the MIPS simulator, a MIPS assembly program that, given an integer number ***n***, reads ***n*** numbers from the terminal, calculates the sum of those numbers which are positive, and prints the result in the terminal.

Bibliography:

[1] Patterson & Hennessy – *Computer Organization and Design: The hardware/software interface 4rd Ed* – MKP 2009.