**Universidade Federal de Mato Grosso do Sul**
**Faculdade de Computação**
**Disciplina: Arquitetura de Computadores I**
**Professor: Renan Albuquerque Marks**

# Descrição do Trabalho

## 1 Introdução

Neste documento estão detalhados os procedimentos que devem ser seguidos para o desenvolvimento do Trabalho da disciplina de Arquitetura de Computadores I. Este trabalho será desenvolvido ao longo deste semestre letivo e constituirá como parte da nota final da disciplina de Arquitetura de Computadores I.

É fortemente recomendado que os estudantes acessem com freqüência este documento para esclarecer possíveis dúvidas, estar ciente do cronograma e estar a par de possíveis atualizações/alterações no trabalho.

## 2 Objetivos

Implementar algoritmos de ordenação utilizando linguagem de montagem MIPS. Os algoritmos deverão ser escritos usando o simulador MARS disponível em http://courses.missouristate.edu/kenvollmar/mars/.

## 3 O que deve ser feito?

O trabalho deve ser de projeto e desenvolvimento em linguagem de montagem MIPS de um algoritmo de ordenação de vetores. Os vetores podem ser de vários tamanhos. Cada elemento do vetor ocupará 1 palavra (4 bytes) na memória RAM. O algoritmo deve receber como entrada o endereço base do vetor e a quantidade de elementos do mesmo.

A sessão de dados do arquivo de montagem (que informa o conteúdo do vetor e seu tamanho) sempre começa no endereço 0x10010000 no simulador MARS. Dessa forma, se houverem duas variáveis do tipo word (palavra de 32 bits) no segmento de dados, a primeira variável estará no endereço 0x10010000 e a segunda no endereço 0x10010004.

O código deverá ser escrito utilizando **somente instruções reais** da ISA MIPS. As unicas exceções são as pseudo-instruções MIPS presentes na Tabela 1. Um exemplo de código para execução no MARS, que pode ser usado como *template* para seu trabalho, pode ser visto na Figura 1.

## 4 Listagem de Algoritmos

Cada grupo deve escolher um algoritmo para realizar a implementação. Cada algoritmo pode ser escolhido, no **máximo**, por **dois grupos**. Caso um algoritmo já tenha sido escolhido por **dois grupos**, ele não poderá mais ser escolhido. Os algoritmos disponíveis estão listados na Tabela 2.

```
# Faculdade de Computação
# Universidade Federal de Mato Grosso do Sul
# Arquitetura de Computadores I
# Prof. Renan Albuquerque Marks
#
# Exemplo de código de montagem de um algoritmo
#

# Seção de dados do programa.
# Usada para "nomear" com labels endereços de memória que são alocados e
# inicializados no inicício do programa.
.data
tamanho: .word 3
vetor:   .word 20, 30, 0

# Seção de texto do programa.
# Usada para informar as instruções MIPS do programa.
.text

# start()
#------------
_start:
  addi $sp, $sp, -4     # Aloca Stack-Frame 1 palavra
  sw   $ra, 0($sp)      # Salva Callee-save
  jal  main             # main()
  lw   $ra, 0($sp)      # Restaura Callee-save
  addi $sp, $sp, 4      # Desaloca Stack-Frame 1 palavra
  addi $v0, $zero, 10
  syscall               # Chamada de sistema para finalização (exit system call)

# main()
#------------
main:
  addi  $sp, $sp, -4    # Aloca Stack-Frame 1 palavra
  sw    $ra, 0($sp)     # Salva Callee-save

  la    $t0, vetor      # Carrega endereço base (inicial) do vetor para $t5
  lw    $t1, tamanho    # Carrega palavra de dados apontada pelo label "tamanho"
  addi  $t1, $t1, -1    # Decrementa tamanho em 1, para indexar o vetor de 0 a 9
  lw    $t2, 0($t0)     # Carrega primeiro elemento do vetor para $t2
  lw    $t3, 4($t0)     # Carrega segundo elemento do vetor para $t3
  add   $t2, $t2, $t3   # Soma o primeiro com o segundo elemento
  sw    $t2, 8($t0)     # Armazena a soma na terceira posicao do vetor

  lw    $ra, 0($sp)     # Restaura Callee-save
  addi  $sp, $sp, 4     # Desaloca Stack-Frame 1 palavra
  jr    $ra             # Return
```

Figura 1: Exemplo de código de montagem MIPS para execução no MARS.

| Pseudoinstrução | Sintaxe | Significado |
|---|---|---|
| `la` | `la dst, label` | Load Label Address into Register |
| `lw` | `lw dst, label` | Load Word From Label into Register |
| `bgt` | `bgt src1, src2, label` | Branch if greater than |
| `blt` | `blt src1, src2, label` | Branch if less than |
| `ble` | `ble src1, src2, label` | Branch if less or equal than |
| `bge` | `bge src1, src2, label` | Branch if greater or equal than |

Tabela 1: Pseudoinstruções aceitas no trabalho

| Algoritmo | Referência |
|---|---|
| Quick Sort | https://en.wikipedia.org/wiki/Quicksort |
| Merge Sort | https://en.wikipedia.org/wiki/Merge_sort |
| Heap Sort | https://en.wikipedia.org/wiki/Heapsort |
| Insertion Sort | https://en.wikipedia.org/wiki/Insertion_sort |
| Comb Sort | https://en.wikipedia.org/wiki/Comb_sort |
| Cocktail Sort | https://en.wikipedia.org/wiki/Cocktail_sort |
| Selection Sort | https://en.wikipedia.org/wiki/Selection_sort |

Tabela 2: Listagem de algoritmos de ordenação.

# 5 Atribuição de grupos

Os trabalhos deverão ser feitos por grupos contendo até 2 (dois) integrantes.

# 6 Cronograma

- **Início dos trabalhos:** 13/05/2024;
- **Entrega:** 17/06/2024 — Cada grupo deve fazer uma **breve apresentação** (5 min) em sala durante a aula, utilizando o MARS, da implentação ordendando um vetor de inteiros aleatório contendo 50 posições. Além disso, deve submeter via AVA o código junto de um relatório que contenha:
  - Integrantes do grupo;
  - Explicação detalhada do funcionamento do algoritmo;
  - Explicação e organização do código:
    - Clareza de comentários;
    - Algoritmo (laços, etc);
    - Funções, parâmetros, variáveis;
  - Dificuldades encontradas;
  - Soluções de implementação;

# 7 Avaliação do trabalho

- **Nota da Versão Final:** valor no intervalo [0,10] que será atribuído de acordo com com o cumprimento das atividades solicitadas para a versão da entrega final. Também serão considerados: organização do código, respeito à ABI do MIPS e comentários no código.

  **Atenção:** casos de plágio serão tratados com extremo rigor.

# 8 Dicas e Sugestões

- Inicie o trabalho o **quanto antes**. O tempo **voa**!
- Baixe o simulador MARS e explore-o durante um período, para se sentir confortável com a ferramenta;
- O simulador MARS possui diversos recursos avançados de programação e depuração de programas, como break-points;
- Retire as dúvidas quanto ao entendimento dos elementos que compõem o algoritmo e o simulador MARS. Isso possibilitará detectar possíveis falhas a tempo de corrigí-las.
- Cheque com frequência a documentação de ajuda (*Help*) do MARS; Muitas informações úteis podem ser encontradas lá;
- Siga **à risca** a ABI MIPS para chamada de funções, pois isso pode poupá-lo de horas de depuração devido a um simples engano na hora de manipular *stack-frames* e/ou registradores *caller-save*/*callee-save*.
- É permitido o uso de compiladores para geração de código MIPS. Entretanto, o código gerado não é compatível com a sintaxe do MARS. Por isso, é recomendado implementar o código ASM à mão.