UNIVALI – Universidade do Vale do Itajaí Disciplina: Arquitetura e Organização de Computadores
Avaliação 01 – Programação em linguagem de montagem

Lei atentamente as instruções abaixo:

1. Esta atividade pode ser realizada individualmente ou em grupo de **até DOIS** alunos (no máximo).

2. A atividade consiste na implementação de **dois programas** utilizando a linguagem de montagem do MIPS (conforme as instruções a seguir).

3. Deve ser postado um relatório, com uma capa identificando a Instituição, o curso, a disciplina, o professor, o nome da atividade, os autores do trabalho e a data em que o mesmo for entregue. O corpo do relatório deverá conter a resolução dos exercícios, incluindo: **código-fonte em linguagem de alto nível** (preferencialmente C ou C++), **código-fonte em linguagem de montagem do MIPS**, **quadro de análise das instruções** e **capturas de tela** que ***demonstrem claramente a execução correta das entradas e saídas realizadas via console do simulador e os resultados da execução dos programas***.

4. Cada código-fonte deve ser escrito em um arquivo com extensão .asm e, **obrigatoriamente**, com um nome que identifique os membros do grupo e o nome do programa (ex. ***RingoStarr_JohnLennon_Programa_01.asm***). Obs: Todos os arquivos de todos os grupos serão reunidos em uma mesma pasta e, por isso, não use nomes como Programa_01.asm.

5. Cada código fonte deve conter um cabeçalho comentado que identifique a disciplina, a atividade, o programa e os nomes dos membros do grupo. Ex:

```
# Disciplina: Arquitetura e Organização de Computadores
# Atividade: Avaliação 01 – Programação em Linguagem de Montagem
# Programa 01
# Grupo: - Ringo Starr
#        - John Lennon
```

6. Os arquivos ASM e o relatório em formato PDF deverão ser postados no ambiente Material Didático compactados em um único arquivo em formato ZIP, conforme instruções fornecidas em aula.

7. O prazo para entrega do relatório e postagem dos códigos fonte é 08h do dia **18/04/17**.

8. Não serão aceitos trabalhos entregues em atraso.

9. O professor poderá solicitar a qualquer momento que **qualquer aluno** do grupo faça uma **demonstração explicativa da execução dos códigos no MARS**.

10. **A implementação deverá apresentar resultados corretos para qualquer conjunto de dados.** Uma solução que **não execute corretamente** terá, automaticamente, um **desconto de 50% na nota**, sendo que o professor também avaliará a correção de segmentos específicos do código (controle de execução, acesso a memória,...).

11. Se forem identificados **trabalhos** com grau de **similaridade** que caracterize cópia (autorizada ou não) ou adaptação, a nota dos grupos será a **nota de um trabalho dividida** pelo número de grupos que entregou esses trabalhos similares.

EXERCÍCIOS DE AQUECIMENTO

Os exercícios propostos abaixo são sugeridos para fins de aquecimento (não valem nota). Tente(m) fazê-los antes de iniciar a implementação dos programas desta avaliação (Programa 01 e Programa 02). Isso os ajudará a consolidar os fundamentos da programação na linguagem de montagem do MIPS antes da implementação de algoritmos mais completos.

Exercício 01
Usando a instrução *syscall*, implemente um programa que: (1) solicite ao usuário que forneça dois números inteiros (X e Y); (2) realize a soma desses dois valores; e (3) apresente o resultado da soma. O programa deve apresentar no console mensagens do tipo:

```
Entre com o valor de X:
Entre com o valor de Y:
A soma de X e Y é igual a:
```

OBS: Este exercício aborda o uso da instrução `syscall`, da instrução `add` e dos registradores.

Exercício 02
Implemente um laço de repetição do tipo `for` que conte de 0 a 9 e imprima o valor de contagem no console, conforme o exemplo abaixo:

```
for (i=0; i<10; i++)
     cout << i;
```

OBS: Este exercício aborda o uso de instruções de desvio (podem ser usadas pseudo-instruções), aritmética e da instrução `syscall`.

Exercício 03
Implemente um programa que declare um vetor de inteiros com 8 elementos, solicite ao usuário a entrada dos elementos do vetor e, após a leitura dos 8 elementos, apresente o valor de cada elemento, em mensagens como as exemplificadas abaixo:

```
LEITURA DOS ELEMENTOS DO VETOR:
Entre com A[0]:
...
Entre com A[7]:

APRESENTAÇÃO DO VETOR LIDO:
A[0] = 4
...
A[7] = 7
```

OBS: Este exercício aborda o uso de instruções de desvio (podem ser usadas pseudo-instruções), da instrução `syscall` e de instruções de acesso à memória (`la`, `lw` e `sw`).

NOTAS:

- No ambiente Material Didático está disponibilizado um programa exemplo que explica como realizar a interface de entrada e de saída com o usuário via chamadas de sistema (*syscall*).

PROGRAMA 01

Enunciado:
Implemente um programa que leia um vetor via console, carregue todos os elementos do vetor na memória e, após, percorra todo o vetor realizando o somatório de todos os elementos.

Leia atentamente cada requisito listado abaixo, pois o atendimento deles será considerado para fins de avaliação (o não atendimento de algum requisito implica em desconto de nota).

Requisitos:

1. Na seção de declaração de variáveis (.data), o vetor deve ser declarado com 8 elementos inicializados em 0. Ele deve ser claramente identificado com um nome como Vetor_A, por exemplo.

2. O programa deve solicitar o número de elementos do vetor, aceitando no máximo vetores com 8 elementos. Para leitura, deve ser apresentada uma mensagem solicitando a entrada desse valor, indicando o seu limite máximo. Ex: "`Entre com o tamanho do vetor (máx. = 8)`".

3. O programa deve solicitar a entrada do número de elementos até que ele seja maior que 1 e menor ou igual a 8. Ou seja, deve implementar um mecanismo de filtragem que não aceite entrada diferente da especificada. No caso de entrada inválida, o programa deve imprimir uma mensagem de advertência antes de solicitar novamente a entrada. Ex: "`Valor inválido`".

4. Para leitura, o programa deve solicitar ao usuário a entrada de cada elemento do vetor, um a um, com mensagens do tipo:
```
Vetor_A[0] =
Vetor_A[1] =
...
```

5. Só após a entrada de todos os elementos do vetor e do armazenamento deles na memória, o programa deve percorrer o vetor e realizar a soma de todos os elementos.

6. Ao final, o programa deve informar ao usuário as informações identificadas utilizando uma mensagem como o seguinte exemplo: "`A soma de todos os elementos do vetor é 48`".

7. O programa deve ser escrito respeitando o estilo de programação ASM, usando tabulação para organizar o código em colunas (rótulos, mnemônicos, operandos e comentários).

8. Procure comentar ao máximo o seu código. Isto é um hábito da programação *assembly*.

9. No seu relatório, **apresente uma análise** indicando a contagem de instruções executadas para cada de classe utilizando a ferramenta "Instruction Statistics" do MARS (conecte a ferramenta ao MIPS antes de executar a simulação), conforme o exemplo abaixo:

Classe	No de execuções	Percentual
Aritmética e lógica (ALU)	4	67%
Desvio incondicional (Jump)	0	0%
Desvio condicional (Branch)	0	0%
Acesso à memória (Memory)	2	33%
Outras (Other)	0	0%

PROGRAMA 02

Enunciado:
Implemente um programa que leia dois vetores via console e, após a leitura dos vetores, produza um terceiro vetor em que cada elemento seja o maior elemento de mesmo índice dos outros dois vetores. Por fim, o programa deve imprimir esse novo vetor na tela.

Leia atentamente cada requisito listado abaixo, pois o atendimento deles será considerado para fins de avaliação (o não atendimento de algum requisito implica em desconto de nota).

Requisitos:

1. Na seção de declaração de variáveis (`.data`), cada vetor deve ser declarado com 8 elementos inicializados em 0. Eles devem ser claramente identificados com nomes como `Vetor_A`, `Vetor_B` e `Vetor_C`, por exemplo.

2. O programa deve solicitar o número de elementos dos vetores aceitando no máximo vetores com 8 elementos. Para leitura, deve ser apresentada uma mensagem solicitando a entrada desse valor, indicando o seu limite máximo. Ex: "`Entre com o tamanho dos vetores (máx. = 8).`".

3. O programa deve solicitar a entrada do número de elementos até que ele seja maior que 1 e menor ou igual a 8. Ou seja, deve implementar um mecanismo de filtragem que não aceite entrada diferente da especificada. No caso de entrada inválida, o programa deve imprimir uma mensagem de advertência antes de solicitar novamente a entrada. Ex: "`Valor inválido`".

4. O programa deve primeiramente ler todos os elementos do `Vetor_A` antes de iniciar a leitura dos elementos do `Vetor_B` . Para leitura, o programa deve solicitar ao usuário a entrada de cada elemento do vetor, um a um, com mensagens do tipo:

```
Vetor_A[0] =
Vetor_A[1] =
...

Vetor_B[0] =
Vetor_B[1] =
...
```

5. Após a entrada dos elementos dos dois vetores, o programa deve comparar os elementos de mesmo índice dos dois vetores (`Vetor_A[i]` e `Vetor_B[i]`) e armazenar o maior no elemento de posição correspondente do `Vetor_C`.

6. Ao final, o programa deve apresenta ao usuário os elementos do terceiro vetor com mensagens do tipo:
```
Vetor_C[0] =
Vetor_C[1] =
...
```

7. Os programas devem ser escritos respeitando o estilo de programação ASM, usando tabulação para organizar o código em colunas (rótulos, mnemônicos, operandos e comentários).

8. Procure comentar ao máximo o seu código. Isto é um hábito da programação *assembly*.

9. No seu relatório, **apresente uma análise** indicando a contagem de instruções executadas para cada de classe utilizando a ferramenta "Instruction Statistics" do MARS (conecte a ferramenta ao MIPS antes de executar a simulação), conforme o exemplo anterior.