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 TRÊS** alunos (no máximo).

2. A atividade consiste na implementação de **dois programas** utilizando a linguagem de montagem do RISC-V32i (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 RISC-V**, **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 **18h59** do dia **20/04/23**.

8. Não serão aceitos trabalhos entregues em atraso.

9. O professor solicitará 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 `ecall` e de instruções de acesso à memória (`la`, `ld` e `sd`).

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 (*ecall*).

PROGRAMA 01

Enunciado:
Implemente um programa que leia dois vetores via console e, após a leitura dos vetores, troque os conteúdos dos 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` e `Vetor_B`, 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 trocar os valores dos elementos dos vetores de mesmo índice.

6. Ao final, o programa deve apresentar novamente os dois vetores no console (de forma similar ao Requisito 4).

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 RARS (conecte a ferramenta ao RISC-V antes de executar a simulação.

PROGRAMA 02

Enunciado:
Considere uma disciplina com 16 dias de aulas e que é necessário registrar a presença de cada aluno em cada dia, consumindo o mínimo de variáveis em memória. Assumindo que a turma tenha no máximo 32 alunos, implemente um programa que utilize um vetor de 16 elementos inteiros (um elemento para cada dia) para registrar a frequência dos alunos de cada disciplina e que ofereça ao professor funções para registrar a presença e ausência de qualquer aluno em qualquer dia.

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`), declare um vetor com 16 elementos inicializados com todos os seus bits em 1 para assumir, por padrão, que todos os alunos estão presentes. A forma mais fácil de fazer isso é utilizar a notação hexadecimal para representar valores com todos os bits em 1 (0xFFFFFFFF).

2. O programa deve ficar em laço infinito solicitando ao usuário para indicar um dia de aula, o aluno a ser registrado e o registro a ser feito (presença ou ausência) e então realizar as operações indicadas nos itens a seguir. Na interface, utilizes mensagens como estas:

    a. Entre com o número da aula (de 0 a 15):
    b. Entre com o número do aluno (de 0 a 31):
    c. Entre com o tipo do registro (presença = 1; ausência = 0):

3. Após a leitura dos dados, dentro do laço, o programa deve construir uma máscara de bits adequada para realizar a operação solicitada (registro de presença ou de ausência) sobre o bit do aluno indicado no dia indicado. Exemplos:

    a. Para registrar a ausência do aluno de número 2, deve ser construída uma máscara de 32 bits apropriada para desligar o bit correspondente (bit 2) e manter os demais inalterados.
    b. Para registrar a presença do aluno de número 20, deve ser construída uma máscara de 32 bits apropriada para desligar o bit correspondente (bit 17) e manter os demais inalterados.

    NOTA: Para construir a máscara de bits, será necessário utilizar operações lógicas de deslocamento e de manipulação de bit.

4. Ainda dentro do laço, o programa deve realizar a operação lógica correspondente ao registro solicitado pelo usuário e usando a máscara de bits construída previamente.

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

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

7. No seu relatório, apresente uma análise indicando a contagem de instruções executadas para cada de classe utilizando a ferramenta Instruction Statistics do RARS (conecte a ferramenta ao RISC-V antes de executar a simulação.