DCC006: Organização de Computadores I **Entrega: 21/10/2022**

Trabalho Prático #1

Professor: Omar Paranaiba Vilela Neto
Monitor: Gabriel Novy

Antes de começar seu trabalho, leia todas as instruções abaixo.

- O trabalho deve ser feito individualmente. **Cópias acarretarão em devida penalização** às partes envolvidas.
- Entregas após o prazo serão aceitas, porém haverá uma penalização. Quanto maior o atraso maior a penalização.
- **Submeta apenas um arquivo .zip** contendo as suas soluções e um arquivo .txt com seu nome e matrícula. Nomeie os arquivos de acordo com a numeração do problema a que se refere. Por exemplo, o arquivo contendo a solução para o problema 1 deve ser nomeado 1.s. Se for solicitado mais de uma implementação para o mesmo problema nomeie 1a.s, 1b.s e assim por diante.
- O objetivo do trabalho é praticar as suas habilidades na linguagem assembly. Para isso, você utilizará o ***Venus Simulator*** **(https://www.kvakil.me/venus/).** O Venus é um simulador de ciclo único que permite enxergar o valor armazenado em cada registrador e seguir a execução do seu código linha a linha. O simulador foi desenvolvido por Morten Petersen e possui a ISA do RISC-V, embora apresente algumas alterações. Você pode utilizar o seguinte link: https://github.com/mortbopet/Ripes/blob/master/docs/introduction.md para verificar as modificações da sintaxe ISA utilizada pelo simulador. Note que no livro e material da disciplina os registradores são de 64 bits, mas o simulador utiliza registradores de apenas 32 bits. Para utilizar o simulador basta você digitar seu código aba *Editor* e para executá-lo basta utilizar a aba *Simulator.*
- A correção do trabalho prático usará o Venus, e será feita de forma automatizada. Portanto, é crucial que vocês **empreguem as convenções de chamada de procedimento definidas no simulador**. Isso irá permitir que o trabalho desenvolvido seja avaliado corretamente. Para cada uma das questões, iremos definir um arquivo de base, onde está marcado a partir de qual ponto vocês deverão fazer o seu código. A avaliação irá considerar somente o que estiver escrito dentro daqueles limites (pois no momento da correção iremos alterar o início e fim do código fonte para fazer a correção).
- Eventuais testes apresentados nesta documentação são apenas para indicar a funcionalidade a ser desenvolvida. O código dos alunos deve funcionar corretamente para todo e qualquer caso, inclusive aqueles que não estão previstos nos testes, desde que sigam as especificações do trabalho. Façam testes além dos que estão descritos nesta documentação.

Problema 1: Inverter vetor recursivamente (prob1.s) (5 pontos)

Escreva um procedimento que inverta um vetor empregando um algoritmo recursivo. O seu procedimento irá receber duas posições de memória: a posição de início do vetor e a posição de fim do vetor.

Existem duas formas para implementar a recursão, a saber: inverter de dentro para fora, onde a chamada recursiva mais interior irá inverter os elementos do meio do vetor; ou inverter de fora para dentro, onde a chamada recursiva mais interior irá inverter os elementos da extremidade do vetor. Qualquer uma das possibilidades será admitida.

Seu programa deve ter um término correto, ou seja, não pode ficar em loop no Venus. **Importante: códigos que não realizarem a inversão recursivamente terão a nota zerada.**

Utilize o esqueleto a seguir para o seu arquivo **prob1.s:**

```
.data
vetor: .word 1 2 3 4 5 6 7
.text
main:
la x12, vetor
addi x13, x0, 7
addi x13, x13, -1
slli x13, x13, 2
add x13, x13, x12
jal x1, inverte
beq x0, x0, FIM
##### START MODIFIQUE AQUI START #####
inverte: jalr x0, 0(x1)

##### END MODIFIQUE AQUI END #####
FIM: add x1, x0, x10
```

Problema 2: Verificador de CPF/CNPJ (prob2.s) **(5 pontos)**

Este programa irá exercitar um conceito muito importante no assembly: a gestão da pilha, que é necessária para desenvolvermos programas em que um procedimento chama outro procedimento. Este conhecimento será exercido implementando três procedimentos:

1. verificacpf: Procedimento que recebe uma string de inteiros representando os dígitos do CPF. Ele retorna 1 se o CPF é válido e 0 se o CPF é inválido. Os argumentos do procedimento são o ponteiro de início do vetor, bem como o seu tamanho.
2. verificacnpj: Procedimento que recebe uma string de inteiros representando os dígitos do CNPJ. Ele retorna 1 se o CPF é válido e 0 se o CNPJ é inválido. Os argumentos do procedimento são o ponteiro de início do vetor, bem como o seu tamanho.
3. verificacadastro: Procedimento que verifica tanto CPF quanto CNPJ. Ele recebe três argumentos, sendo eles o o ponteiro de início do vetor, o seu tamanho e se o vetor é um CPF ou CNPJ (**zero** para CPF, **um** para CNPJ).

Os algoritmos de verificação de CPF e CNPJ são conhecidos, e estão disponíveis no link a seguir. **O procedimento verificadastro deverá chamar os procedimentos verificacpf e verificacnpj internamente. Essa verificação será feita visualmente a cada submissão, e trabalhos sem essa característica terão a nota zerada nesse exercício.**

```
.data
vetor: .word 1 2 3 4 5 6 7 8 9 10 11
.text
main:
la x12, vetor
addi x13, x0, 11
jal x1, verificacpf
beq x0,x0,FIM
##### START MODIFIQUE AQUI START #####
verificacpf:    jalr x0, 0(x1)
verificacnpj:   jalr x0, 0(x1)
verificadastro: jalr x0, 0(x1)
##### END MODIFIQUE AQUI END #####
FIM: add x1, x0, x10
```

Dicas e sugestões

- Não deixe o trabalho para o último dia. Não viva perigosamente!
- Comente seu código sempre que possível. Isso será visto com bons olhos.