DCC006: Organização de computadores I

Trabalho Prático #1

Professor: Daniel Fernandes Macedo

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

- O trabalho deve ser feito individualmente. Cópias de trabalho 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 te 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 simulador, 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 (por exemplo, irei usar registradores "sx", que são salvos pelo chamador, para realizar a contabilização automática de testes que foram executados 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 somente 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 códigos, desde que sigam as especificações do trabalho. Façam testes além dos que estão descritos nesta documentação.

Problema 1: Números primos (prob1.s) (5 pontos)

Escreva um procedimento que calcule os números primos na faixa a ser indicada, salvando o dado em uma posição de memória especificada. O seu procedimento deve ter o nome "primos", e irá receber os seguintes parâmetros:

- a0: valor de início para a pesquisa (inclusive)
- a1: valor de fim para a pesquisa (inclusive)
- a2: ponteiro para posição de memória aonde os números primos devem estar

O seu programa deve retornar quantos números primos foram encontrados na sequência. Assuma que a memória onde os números primos serão escritos possui espaço suficiente para que todos eles sejam escritos.

Utilize o esqueleto a seguir para o seu arquivo **prob1.s (repare que a parte acima e abaixo do MODIFIQUE AQUI poderá ser alterada pelo professor/monitor no momento da correção:**

```
.data
vetor: .word 0 0 0 0
##### START MODIFIQUE AQUI START #####
#
# Este espaço eh para você definir as suas constantes e vetores auxiliares.


##### END MODIFIQUE AQUI END #####

.text
main:
add s0, zero, zero
#Teste 1
addi a0, zero, 5
addi a1, zero, 7
la a2, vetor
jal ra, primos
addi t0, zero, 2
beq t0, a0, OK1
beq zero, zero, T2
OK1: addi s0, s0, 1 #Teste ok, passou

#Teste 2
T2: addi a0, zero, 1
addi a1, zero, 6
la a2, vetor
jal ra, primos
addi t0, zero, 3
beq t0, a0, OK2
beq zero,bzero, FIM
OK2: addi s0, s0, 1 #Teste ok, passou
beq zero,bzero, FIM

##### START MODIFIQUE AQUI START #####
primos: jalr zero, 0(ra)

##### END MODIFIQUE AQUI END #####
FIM: add zero, zero, zero
#Final da execucao, s0 deve ter o valor igual a 2.
```

Problema 2: Validador de cartão de crédito (prob2.s) (5 pontos)

Este programa irá exercitar um conceito muito importante no assembly: a gestão da pilha de chamadas de forma correta, que é necessária para desenvolvermos programas em que um procedimento chama outro procedimento ou para que seja possível usar bibliotecas de terceiros. Este conhecimento será exercido implementando dois procedimentos. Os procedimentos vão seguir o algoritmo de Luhn, que valida números de cartão de crédito. **Observação: o código de correção automática pode fazer a chamada dos procedimentos separadamente para fins de verificação de que a gestão da pilha está correta.** Uma descrição do algoritmo pode ser encontrada na página a seguir: https://suporte.braspag.com.br/hc/pt-br/articles/360050638051-Como-validar-um-cart%C3%A3o-Mod10

1. verifica: Procedimento que recebe uma *string* de inteiros representando os dígitos do cartão de crédito. Ele retorna 1 se o cartão é válido e 0 se o cartão é inválido. O único argumento é o ponteiro para os dígitos do cartão de crédito.
2. multvetores: Procedimento que executa o passo 2 do algoritmo apresentado na página Web. Em outras palavras, ele calcula a multiplicação do vetor que contém os dígitos do cartão de crédito pelo vetor de base (um vetor com o valor 1 para os índices ímpares e 2 para os índices pares, começando o vetor com o índice zero). O retorno deste procedimento deve ser o escalar que resultante da multiplicação dos dois vetores. O único argumento deste procedimento é o vetor com os dígitos do cartão de crédito.

```
.data
cartao: .word 4 9 1 6 6 4 1 8 5 9 3 6 9 0 8 0
##### START MODIFIQUE AQUI START #####
#
# Este espaço eh para você definir as suas constantes e vetores auxiliares.


##### END MODIFIQUE AQUI END #####

.text
main:
la a0, cartao
jal ra, verifica
beq zero,zero,FIM
##### START MODIFIQUE AQUI START #####
verifica:    jalr zero, 0(ra)
multvetores:    jalr zero, 0(ra)
##### END MODIFIQUE AQUI END #####
FIM: add zero, zero, zero
```

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.