DCC006/DCC017 Organização de computadores I/Introdução à Arquitetura de Computadores

Trabalho Prático #1

Professores: Daniel Fernandes Macedo e Omar Paranaiba Vilela Neto

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 prob1.s. Se for solicitado mais de uma implementação para o mesmo problema, nomeie prob1a.s, prob1b.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, iremos usar registradores “sx”, que são salvos pelo chamador, para realizar a contabilização automática de testes que foram executados corretamente. em outras palavras, caso seu procedimento use registradores "sx" eles deverão ser salvos na pilha). 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.
- Em ambos os problemas especificados estamos fornecendo alguns exemplos de execução, para que saibam como o código será avaliado. A sugestão é que enviem o código baseado nestes exemplos, modificando apenas a região delimitada. Com isso será possível executar scripts que substituem, automaticamente, a parte exemplo do programa com o código de testes. Caso sejam feitas alterações fora destas partes, o seu programa corre o risco de funcionar parcialmente ou mesmo não funcionar.

Problema 1: Números múltiplos de X (prob1.s) (5 pontos)

Escreva um procedimento que conte a quantidade de números que são múltiplos de X um vetor. O seu procedimento deverá ter o nome "multiplos", e irá receber os seguintes parâmetros.

- a0: endereço de início do vetor
- a1: tamanho do vetor
- a2: inteiro que deve ser verificado (quantos múltiplos do mesmo existem)

O retorno deve ser a quantidade de múltiplos no vetor. O Venus não mapeia o registrador r0 para x10, então usem x10 ou a0 como o nome do registrador de retorno em ambos os problemas.

Assuma que a memória onde os números do vetor 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

##### R1 START MODIFIQUE AQUI START #####
#
# Este espaço é para você definir as suas constantes e vetores auxiliares.
#

vetor: .word 1 2 3 4 5 6 7 8 9 10


##### R1 END MODIFIQUE AQUI END #####

.text
        add s0, zero, zero #Quantidade de testes em que seu programa passou
        la a0, vetor
        addi a1, zero, 10
        addi a2, zero, 2
        jal ra, multiplos
        addi t0, zero, 5
        bne r0,t0,teste2
        addi s0,s0,1
teste2: la a0, vetor
        addi a1, zero, 10
        addi a2, zero, 3
        jal ra, multiplos
        addi t0, zero, 3
        bne r0,t0, FIM
        addi s0,s0,1
        beq zero,zero,FIM

##### R2 START MODIFIQUE AQUI START #####
multiplos: jalr zero, 0(ra)

##### R2 END MODIFIQUE AQUI END #####

FIM: addi t0, s0, 0
```

Problema 2: Covariância (prob2.s) (5 pontos)

Este programa irá exercitar um conceito muito importante no assembly: a gestão da pilha de chamadas de subrotinas 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. Em outras palavras, você terá que manipular o espaço da memória denominado *stack* (pilha de funções) para armazenar corretamente o endereço de retorno para as próximas instruções.

O seu papel é desenvolver um programa que calcule a covariância de dois vetores. Na estatística, a covariância mede qual é o grau de independência entre a variação de duas grandezas. Ela é usada para determinar se existe uma correlação entre ambas. Por exemplo, quando alteramos a aceleração de um carro, a sua velocidade tende a variar. Da mesma forma, a venda de sorvetes pode estar fracamente correlacionada com a venda de ventiladores (provavelmente porque ambos vendem mais quando está fazendo calor). A fórmula da covariância é a seguinte:

$cov_{x,y} = \frac{\Sigma \quad (x_i - \underline{x})(y_i - \underline{y})}{N-1}$, onde xi e yi são os i-ésimos elementos da sequência de valores, N é a quantidade de valores de cada sequência (considere que ambos possuem a mesma quantidade de valores), e $\underline{x}$ é a média dos valores de x (o mesmo vale para os valores de y). Tipicamente, se a covariância se aproxima de zero, as duas sequências são completamente independentes. Se a covariância é positiva, os valores de x são diretamente proporcionais aos valores de y, e caso seja negativa há uma relação de proporcionalidade inversa. O programa em assembly a ser desenvolvido deve possuir duas funções: `media` e `covariancia`. A primeira calcula a média de um vetor, passando como argumentos o ponteiro de início do vetor e a quantidade de elementos. Já o segundo procedimento recebe como argumentos, nesta ordem, o ponteiro de início do primeiro vetor, o ponteiro de início do segundo vetor, e a quantidade de elementos em cada vetor (lembrando que ambos terão a mesma quantidade de elementos). A função média retornará o valor médio dos elementos do vetor, enquanto a covariância irá retornar a covariância dos dois vetores. Segue abaixo um trecho de código de exemplo de como o seu programa pode vir a ser testado. **Atenção: o cálculo deve ser feito com aritmética inteira.**

```
.data

##### R1 START MODIFIQUE AQUI START #####

#
# Este espaço é para você definir as suas constantes e vetores auxiliares.
#

vetor1: .word 1 2 3 4 #Primeiro vetor
vetor2: .word 1 1 1 1 #Segundo vetor

##### R1 END MODIFIQUE AQUI END #####

.text

        add s0, zero, zero
        la a0, vetor1
        addi a1, zero, 4
        jal ra, media
        addi t0, zero, 2
        bne r0,t0,teste2
        addi s0,s0,1
teste2: la a0, vetor2
        addi a1, zero, 4
        jal ra, media
        addi t0, zero, 1
        bne r0,t0, FIM
```

```
        addi s0,s0,1
        beq zero,zero, FIM

##### R2 START MODIFIQUE AQUI START #####

# Esse espaço é para você escrever o código dos procedimentos.
# Por enquanto eles estão vazios

media: jalr zero, 0(ra)
variancia: jalr zero, 0(ra)


##### R2 END MODIFIQUE AQUI END #####

FIM: add t0, zero, s0
```

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.