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 suas soluções e um relatório em formato .pdf, com no máximo duas páginas, nomeado com seu nome completo e número de matrícula. O relatório deve descrever brevemente o que foi feito no trabalho, por exemplo, as decisões de implementação tomadas, dificuldades enfrentadas, testes realizados e os principais aprendizados adquiridos**. Os arquivos de código devem ser nomeados de acordo com a numeração do problema correspondente (por exemplo, prob1.s para o problema 1).
- 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 e de gerenciamento de pilha do RISC-V**. 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.
- Dica final: O Venus não possui o apelido r0 para registrador de retorno. Usem o a0 como registrador de retorno, porque no RISC-V o a0 e o r0 mapeiam para o mesmo registrador físico.

Problema 1: Cálculo de one-time pad (prob1.s) (5 pontos)

Na área de **Criptografia e Segurança da Informação**, um dos métodos de criptografia mais comum é o one-time pad (OTP). Nesse método, é utilizada uma chave secreta de tamanho igual (ou maior) que o texto que se deseja encriptar. Para cada byte da chave, é aplicada a operação de ou exclusivo (XOR) com o byte correspondente da mensagem. O interessante desse método de criptografia é que as operações de encriptar e decriptar são iguais, ou seja, ao aplicar XOR novamente entre a chave e a mensagem encriptada, é possível recuperar a mensagem original e ler o texto.

Escreva um procedimento chamado `onetime_pad` que realizar a operação de encriptação / decriptação utilizando o método de one-time pad. O primeiro argumento é o vetor contendo a mensagem, o segundo argumento é o vetor com a chave secreta e o terceiro argumento é o tamanho dos vetores. O procedimento deve **modificar** o vetor com a **mensagem**, aplicando o one-time pad.

Assuma que o tamanho da chave tem tamanho maior ou igual ao da mensagem e a memória onde os valores dos vetores serão escritos possui espaço suficiente.

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.
#

mensagem1: .word 2, 3, 4
mensagem2: .word 1, 5, 4, 7
chave: .word 3, 5, 7, 9

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

.text
  add s0, zero, zero       # s0 armazenará o número de testes que passaram
teste_1:
  # Chama procedimento
  la a0, mensagem1
  la a1, chave
  addi a2, zero, 3
  jal onetime_pad

  # Compara saída com a saída esperada
  # 2 ^ 3 = 1
  # 3 ^ 5 = 6
  # 4 ^ 7 = 3
  lw t1, 0(a0)
  li t2, 1
  bne t1, t2, teste_2

  lw t1, 4(a0)
  li t2, 6
  bne t1, t2, teste_2

  lw t1, 8(a0)
  li t2, 3
  bne t1, t2, teste_2
  addi s0,s0,1
```

```
teste_2:
  # Chama procedimento
  la a0, mensagem2
  la a1, chave
  addi a2, zero, 4
  jal onetime_pad

  # Compara saída com a saída esperada
  # 1 ^ 3 = 2
  # 5 ^ 5 = 0
  # 4 ^ 7 = 3
  # 7 ^ 9 = 14

  lw t1, 0(a0)
  li t2, 2
  bne t1, t2, FIM

  lw t1, 4(a0)
  li t2, 0
  bne t1, t2, FIM

  lw t1, 8(a0)
  li t2, 3
  bne t1, t2, FIM

  lw t1, 12(a0)
  li t2, 14
  bne t1, t2, FIM
  addi s0,s0,1
  j FIM

##### R2 START MODIFIQUE AQUI START #####
onetime_pad:
    #   a0 = endereço da mensagem
    #   a1 = endereço da chave
    #   a2 = tamanho da mensagem
    # Realiza mensagem[i] = mensagem[i] XOR chave[i]
jalr zero, 0(ra)

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

FIM:
```

Problema 2: Multiplicação de matrizes (prob2.s) (5 pontos)

Na área de Processamento de Imagens Digitais, operações matriciais são fundamentais. Uma imagem em escala de cinza pode ser representada como uma matriz de pixels, onde cada elemento armazena a intensidade luminosa de um ponto da cena. Muitas transformações sobre imagens — como rotações, filtros de nitidez, borramento, detecção de bordas e transformações geométricas — envolvem multiplicação de matrizes.

Um exemplo clássico é a aplicação de filtros (convoluções), que podem ser descritos como a multiplicação entre uma matriz que representa a imagem e uma matriz que representa a máscara (filtro). Além disso, transformações lineares em coordenadas (como rotação e escala) também são expressas por multiplicações de matrizes.

Implemente dois procedimentos em assembly, responsáveis por calcular o resultado da **multiplicação de duas matrizes quadradas (n×n)**. São eles:

- **mat_mult** :
  - **Descrição**: responsável por iterar sobre as linhas da matriz A. Para cada linha dessa matriz, esse procedimento deve iterar pelas colunas da matriz B, chamar o procedimento `dot_product` e armazenar o resultado na matriz C.
  - **Requisitos**:
    - Deve chamar internamente o procedimento `dot_product`.
    - Recebe os seguintes argumentos:
      - `a0`: endereço base da matriz A
      - `a1`: endereço base da matriz B
      - `a2`: endereço base da matriz C
      - `a3`: tamanho das matrizes (**n**)
    - O resultado (multiplicação das matrizes) deve ser armazenado na matriz C em `a2`.
- **dot_product**:
  - **Descrição:** Calcula o produto interno entre a coluna e a linha recebidas como parâmetros. Retorna o resultado da operação em `a0`.
  - **Requisitos:**
    - Recebe os seguintes argumentos:
      - `a0`: endereço base da linha da matriz A
      - `a1`: endereço base da coluna da matriz B
      - `a2`: tamanho das matrizes (**n**)
    - O resultado do produto interno deve ser retornado em `a0`.

É **obrigatório** que o procedimento **`mat_mult`** chame internamente o procedimento **`dot_product`**. Trabalhos que não seguirem essa lógica terão nota zero, sendo feita uma conferência visual do código.

Cada uma das matrizes A, B e C serão armazenadas em um vetor, com suas linhas em sequência. Por exemplo, a matriz[1 2 3 4 5 6 7 8 9] será armazenada como 1, 2, 3, 4, 5, 6, 7, 8, 9 .

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.

Utilize o esqueleto a seguir para o seu arquivo prob2.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.
#

matriz_a: .word 1, 2, 3, 4
matriz_b: .word 5, 6, 7, 8
matriz_c: .word 0, 0, 0, 0
```

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

.text
  add s0, zero, zero       # s0 armazenará o número de testes que passaram
teste_1:
  # Chama procedimento
  la a0, matriz_a
  la a1, matriz_b
  la a2, matriz_c
  addi a3, zero, 2
  jal mat_mult

  # Compara saída com a saída esperada
  # | 1  2 | X | 5  6 |  = | 19  22 |
  # | 3  4 |   | 7  8 |    | 43  50 |
  la a0, matriz_c
  lw t1, 0(a0)
  li t2, 19
  bne t1, t2, teste_2

  lw t1, 4(a0)
  li t2, 22
  bne t1, t2, teste_2

  lw t1, 8(a0)
  li t2, 43
  bne t1, t2, teste_2

  lw t1, 12(a0)
  li t2, 50
  bne t1, t2, teste_2
  addi s0,s0,1

teste_2:
  # Chama procedimento
  la a0, matriz_a
  la a1, matriz_b
  addi a2, zero, 2
  jal dot_product

  # Compara saída com a saída esperada
  # (1, 2) * (5, 7) = 19
  li t1, 19
  bne a0, t1, FIM
  addi s0,s0,1
  j FIM

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

mat_mult:
    #   a0 = endereço da matriz A
    #   a1 = endereço da matriz B
    #   a2 = endereço da matriz C
    #   a3 = tamanho das matrizes (n)
    # Realiza C = A * B

    jalr zero, 0(ra)

dot_product:
```

```
    #   a0 = endereço base da linha da matriz A
    #   a1 = endereço base da coluna da matriz B
    #   a2 = tamanho das matrizes (n)
    # Retorna a0 = produto interno entre a linha de A e a coluna de B

    jalr zero, 0(ra)

##### R2 END MODIFIQUE AQUI END #####
FIM:
```

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.