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 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.

Problema 1: Mínimo múltiplo comum (prob1.s) (5 pontos)

Escreva um procedimento, chamado "mmc", que irá calcular o mínimo múltiplo comum de dois números. O seu programa deverá receber os seguintes parâmetros.

- a0: primeiro número
- a1: segundo número

O retorno deve ser o mínimo múltiplo comum. 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
        addi a0, zero, 10
        addi a1, zero, 2
        jal ra, mmc
        addi t0, zero, 10
        bne a0,t0,teste2
        addi s0,s0,1
teste2: addi a0, zero, 6
        addi a1, zero, 4
        jal ra, mmc
        addi t0, zero, 12
        bne a0,t0, FIM
        addi s0,s0,1
        beq zero,zero,FIM

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

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

FIM: addi t0, s0, 0
```

Problema 2: Primos de Mersenne (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.

Os números de mersenne são os números na forma 2^n-1, por exemplo, 1, 3, 7, 15, 31, 63…. Alguns destes números são primos, e são chamados de primos de Mersenne. Os primeiros números primos de Mersenne são $M_1 = 3$, $M_2 = 7$, $M_3 = 31$, $M_4 = 127$, $M_5 = 8.191$, $M_6 = 131.071$, $M_7 = 524.287$

Você irá fazer três procedimentos, como especificados a seguir:

- Procedimento **geramersenne**, que recebe dois argumentos (a quantidade de números de mersenne a serem gerados) e um vetor onde os números serão armazenados, nesta ordem.
- Procedimento **eprimo**, que recebe como argumento um número inteiro. Ele retorna 1 se não for primo, e 0 se for primo.
- Procedimento **primosmersenne**, que retorna o enésimo primo de mersenne. Considere que o índice começa em 1, como mostrado na lista acima de primos de mersenne.

O procedimento **primosmersenne** deve usar internamente os procedimentos **geramersenne** e **eprimo**. Trabalhos que não operarem desta forma terão a nota zerada (será feita uma conferência visual do código). Vale salientar também que todos os procedimentos serão testados também de forma individual, ou seja, deve ser possível fazer a chamada de procedimentos do programa de teste para cada um dos procedimentos individuais.

Devido ao fato do Venus usar variáveis de 32 bits, o cálculo dos números primos de mersenne será limitado até o oitavo número primo de Mersenne, que é 2^{32}-1 (veja a lista de números conhecidos em https://www.mersenne.org/primes/). Segue um exemplo de programa de teste:

```
.data

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

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

vetor1: .word 1 2 3 4 5 6 7 8 9 10 #Primeiro vetor

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

.text
        add s0, zero, zero
        la a0, 4
        jal ra, eprimo
        addi t0, zero, 0
        bne a0,t0,teste2
        addi s0,s0,1
teste2: addi a0, zero, 2
        jal ra, primosmersenne
        addi t0, zero, 7
        bne a0,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

geramersenne: jalr zero, 0(ra)
eprimo: jalr zero, 0(ra)
primosmersenne: 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.