Laboratório 2

A intenção desse laboratório é que o aluno pratique o uso de mais instruções do Assembly do RISC-V. Para isso, as tarefas começarão a ganhar um pouco de complexidade mas não muito por enquanto! Pense em soluções simples!

Dicas

- Você não precisa entregar nenhum código como resposta. Procure entender os conceitos e explorar as variações.
- Você pode utilizar o simulador RISC-V para testar os códigos que você desenvolver. Para isso, veja as dicas fornecidas logo abaixo.

Gerenciando um pouco mais de abstração no seu código

Os exemplos do laboratório anterior colocavam comentários ao final da linha, indicando a funcionalidade de cada uma delas. Os comentários começam sempre pelo caracter **#**. O simulador que você está utilizando não permite colocar acentos nem mesmo dentro dos comentários.

A partir de agora, a sugestão é que você comente trechos de código e procure defini-los como partes de um programa que realizam uma atividade em conjunto ao invés de comentar linha a linha com a funcionalidade básica dela.

Por exemplo, no laboratório anterior tínhamos esse trecho de código:

```
addi t0, zero, 4   # escolhe a operacao de leitura de inteiro (4)
ecall              # efetua a operacao de leitura de inteiro
```

que fica melhor comentado como:

```
# Executa a operacao de sistema de leitura de um numero inteiro (4)
# e armazena o resultado em a0
addi t0, zero, 4
ecall
```

Note como o novo comentário é mais intuitivo. Vamos fazer assim daqui para frente?

Instruções de salto

As instruções de salto servem para alterar o fluxo (caminho) da execução do seu programa. Nossos exemplos anteriores foram extremamente simples e não tinham que tomar nenhuma decisão nem mesmo executar por muito tempo. Agora é hora de fazer isso.

Você se lembra da atividade que lia um número do teclado e somava 2 a ele? Que tal ler múltiplos números do teclado e somar 2 a cada um deles, imprimindo o resultado de cada soma na tela? Para isso, vamos

precisar de uma estrutura de repetição.

A primeira versão do seu código vai repetir infinitamente as leituras e impressões, gerando um loop infinito. Então, os comentários (note a ausência de acentos) do código ficam assim:

```
main:
  # Le um numero do teclado e armazena em a0


  # Soma 2 ao valor de a0


  # Imprima o valor de a0 na tela


  # Va novamente para o main e recomece a execucao
  j main
  ret
```

A instrução **j** serve para fazer um salto para uma determinada posição de memória. Nesse caso, a posição está especificada através do *label* **main**. Um *label* é um nome que você dá a uma posição de memória. Ele é utilizado para facilitar a leitura do código e para facilitar a manutenção do código.

Toda vez que seu programa chega na instrução **j main** ele volta para main e recomeça a execução (loop infinito).

Atividade 1

Complete as instruções no código acima para que seu programa cumpra exatamente o que está dito no comentário. Você deve usar a instrução **j** para fazer o loop infinito.

Mas eu quero que meu programa pare!

Faz todo sentido querer executar uma tarefa apenas um número finito de vezes. Para isso, você tanto pode definir um número fixo fazendo um laço ao estilo **for** ou pode definir uma condição fazendo um laço ao estilo **while**.

Vamos começar com o **while**. Vamos alterar o código para que ele continue executando enquanto o número digitado for diferente de 0 (zero). Para isso, é importante que você também salve o número digitado em outro registrador ao invés de mante-lo apenas no **a0** que será atualizado logo em seguida.

O RISC-V tem instruções de salto condicional, dentre elas, o **beq** que significa *branch if equal* (salta se dois valores forem iguais) e o **bne** que significa *branch if not equal* (salta se dois valores forem diferentes). Supondo que seu código guardou o valor lido no registrador **s0** também, o final do código modificado fica:

```
  # Se o valor lido for diferente de zero, va para o inicio
  bne s0, zero, main
  ret
```

Dica

A instrução **bne** tem 3 operandos, o primeiro é o registrador que contém o valor a ser comparado, o segundo é o registrador que contém o valor a ser comparado e o terceiro é o *label* para onde deve ser feito o salto. Nenhum deles pode ser um imediato (número) diretamente. Se você precisar de um número, basta coloca-lo em algum registrador anteriormente e utilizar esse registrador na comparação.

Atividade 2

Altere seu programa para executar enquanto o número digitado for diferente de zero. Você deve usar a instrução **beq** para fazer o loop.

Atividade 3

Ficou meio estranho calcular a soma com 2 para o valor 0 que é o valor utilizado para sair do programa. Altere seu programa para não imprimir o resultado da soma com 2 se o número digitado for zero. Não se esqueça de atualizar o comentário ou colocar novos se achar conveniente.

Dica

Você pode criar um novo *label* chamado **fim** ao final do programa e saltar para ele se achar conveniente.

Variações sobre o mesmo tema

As atividades abaixo servem para você praticar um pouco mais as instruções de salto. O RISC-V possui mais instruções de salto que você poderá utilizar, elas estão mostradas na tabela abaixo:

Instrução	Formato	Uso	Descrição
Branch if ==	B	BEQ rs1, rs2, label	Salta se rs1 for igual a rs2
Branch if !=	B	BNE rs1, rs2, label	Salta se rs1 for diferente de rs2
Branch if <	B	BLT rs1, rs2, label	Salta se rs1 for menor que rs2
Branch if >=	B	BGE rs1, rs2, label	Salta se rs1 for maior ou igual a rs2
Branch if <	B	BLTU rs1, rs2, label	Salta se rs1 for menor que rs2 considerando números sem sinal

Instrução	Formato	Uso	Descrição
Branch if >=	B	BGEU rs1, rs2, label	Salta se rs1 for maior ou igual a rs2 considerando números sem sinal

Dica

Note a adição do sufixo **U** ao final das instruções de salto para considerarem números sem sinal. Isso é importante para evitar problemas com números negativos.

Atividade 4

Altere seu programa para que ele execute exatamente 10 vezes. Você deve usar a instrução **bge** para fazer o loop.

Dica

Você pode utilizar outro registrador para contar a quantidade de interações e definir o limite de execução. É muito comum, nos casos onde temos uma contagem fixa, fazer a contagem de forma regressiva (indo do maior número para zero) de forma a aproveitar o registrador **zero** na comparação final. Se você quiser, você também pode colocar o valor limite em outro registrador.

Atividade 5

Altere seu programa para que ele leia de 2 em 2 números e sempre imprima o maior deles. Você deve encerrar quando qualquer um dos dois for zero.

Desafio final

Alguns colegas de vocês tentaram imprimir um número negativo no laboratório anterior e se surpreenderam com a impressão de um número muito grande. Esse "número grande" é a representação interna de números negativos dentro do computador, chamada de **complemento de dois** (falaremos mais sobre isso mais adiante no curso). Veja o exemplo de código abaixo que lê um número do teclado e imprime o resultado da subtração de 10 dele. Experimente digitar 5 como entrada e veja o resultado.

```
main:
    # le um numero do teclado
  addi t0, zero, 4
  ecall
    # subtrai 10 do numero lido
  addi a0, a0, -10
    # imprime o resultado
  addi t0, zero, 1
  ecall
  ret
```

Se você digitou 5 como entrada, a sua saída foi o número 4294967291, que é a representação sem sinal do número -5. Certamente você não quer fazer essa conta toda vez que precisar gerar um número negativo, certo?

Atividade 6

Você deve ajustar o código acima para que ele imprima o número positivo caso seja positivo e o número negativo caso seja negativo.

Dicas

- Números negativos são menores que zero!
- Uma forma de imprimir um número negativo é imprimindo o sinal de menos (**-**) antes do número e depois imprimir a representação positiva dele.
- Você consegue imprimir um caracter utilizando a chamada de sistema número 2 (colocando 2 em t0 antes do ecall) e colocando o código do caracter em em a0. O código do sinal de menos é 45 (existe uma tabela chamada ASCII que representa todos esses códigos, você pode buscar outros símbolos lá).

Conclusões

Você desenvolveu programas com uso das instruções de salto e agora já é capaz de fazer estruturas mais complexas. Você também mudou a forma de escrever seus comentários para que eles fiquem mais claros e fáceis de entender.

Resumo

Você aprendeu a representar laços **for** e **while**, além de **if** em assembly. Você também aprendeu a escrever comentários mais claros e fáceis de entender. Além disso, ficou com uma pulga atrás da orelha ao saber que todos os caracteres que você imprime no terminal são representados por números e buscou a tabela ASCII no Google!