# Lista de Exercícios RISC-V

Gabriel G. de Brito  Daniel Oliveira

Segundo semestre de 2024

A melhor maneira de resolver esses exercícios é com o debugger do emulador EGG. Lembre sempre da instrução *ebreak* no final!

## Sumário



## 1 Instruções Aritméticas

Para testar suas resoluções aos exercícios dessa seção, carregue seu código no debugger do emulador e use os comandos *set* e *print* para modificar manualmente o valor dos registradores e verificá-los depois.

Por exemplo, para testar a resolução a um exercício hipotético que deve somar 1 ao valor em *a0*:

```
$ egg -d teste.asm
EGG - Emulador Genérico do Gabriel - versão 3.0.0
Entre 'help' para ver uma lista de comandos
Debuggando RISC-V IM (32 bits)
egg> set a0 41
egg> continue
Parado na chamada BREAK no endereço 0x8
nenhuma instrução no endereço 0x8
Registrador a0: mudou de 0x00000000 para 0x0000002a
egg> print a0
0x0000002a
egg> q
até mais!
```

1. Escreva um programa que calcule a média entre dois números armazenados nos registradores *a0* e *a1*. Guarde o resultado em *a0*.

2. Resolva o exercício anterior sem utilizar instruções de multiplicação.

3. Escreva um programa que calcule a média entre três números armazenados nos registradores *a0*, *a1* e *a2*. Guarde o resultado em *a0*.

4. Escreva um programa que calcule a expressão $b^2 - 4 \cdot a \cdot c$ (o $\Delta$ de uma equação de segundo grau), onde $a$, $b$ e $c$ estão armazenados nos registradores *a0*, *a1* e *a2*, respectivamente. Guarde o valor em *a0*.

5. Resolva o exercício anterior utilizando somente duas instruções de multiplicação.

6. Escreva um programa que, dados números $a$ e $b$ armazenados nos registradores *a0* e *a1*, respectivamente, guarde o valor 1 em *a0* caso $a < b$, e o valor 0 caso contrário. Não use instruções de salto, *slt* ou *sltu*.

7. Escreva um programa que, dado um número armazenado em *a0*, guarde em *a0* o valor do segundo byte desse número e em *a1* o valor do quarto byte. Exemplo de execução:

```
$ egg -d solucoes/bytes.asm
EGG - Emulador Genérico do Gabriel - versão 3.0.0
Entre 'help' para ver uma lista de comandos
Debuggando RISC-V IM (32 bits)
egg> set a0 0xcafebabe
egg> c
Parado na chamada BREAK no endereço 0x18
nenhuma instrução no endereço 0x18
Registrador t0: mudou de 0x00000000 para 0x000000ba
Registrador a0: mudou de 0x00000000 para 0x000000ba
Registrador a1: mudou de 0x00000000 para 0x000000ca
egg> p a0
0x000000ba
egg> p a1
0x000000ca
egg> q
até mais!
```

8. Resolva o exercício anterior sem utilizar as instruções *and* e *andi*. Dica: use *xor* e *slli*.

9. Escreva um programa que calcule o valor em representação sinal-magnitude do número em complemento de dois armazenado no registrador *a0*. Guarde

o resultado em *a0*. Não utilize instruções de salto. Nota: números negativos em sinal magnitude vão aparecer com os últimos dígitos hexadecimais 8, 9, A, B, C, D, E ou F. Exemplo: -1 é 0x80000001.

10. (Desafio) Computadores modernos costumam suportar operações aritméticas nativas de números de 32 ou 64 bits. Caso seja necessário fazer aritmética com números maiores, é necessário implementar funções em software que utilizam as instruções da arquitetura para somar números maiores - de 128 bits por exemplo. Essas funções costumam ser chamadas de operações em "BigInt" e demoram mais, pois, ao contrário de operações em números suportados pela arquitetura, precisam de muito mais que apenas uma instrução para serem executadas.

    A arquitetura que estudamos - RISC-V 32 IM - suporta apenas números de até 32 bits. Escreva um programa que some dois números de 64 bits A e B, onde a parte baixa de A está armazenada em *a0*, a parte alta em *a1* e, semelhantemente, a parte baixa de B está armazenada em *a2* e a parte alta em *a3*. Não utilize instruções de salto. Guarde a parte baixa do resultado em *a0* e a parte alta em *a1*.

# 2 Instruções de salto

Semelhantemente à seção anterior, utilize os comandos *set* e *print* para testar as soluções.

1. Escreva um programa que calcule o fatorial de um número armazenado no registrador *a0*, guardando o resultado em *a0*.

2. Escreva um programa que calcule o n-ésimo número da sequência de Fibonacci, onde n está no registrador *a0*, armazenando-o no registrador *a0*. Note que tanto o primeiro quanto o segundo número de Fibonacci são 1.

3. Escreva um programa que calcule a quantidade de bits 1 em um número binário armazenado no registrador *a0*. Guarde o resultado em *a0*.

4. Escreva um programa que conte a quantidade de "trailing zeros" de um número binário armazenado no registrador *a0*, guardando o resultado em *a0*. "Trailing zeros" são os zeros à direita do número, ou seja, os bits zero antes do primeiro bit 1. Por exemplo, o número de "trailing zeros" do número binário 1011000 é 3.

5. Escreva um programa que verifique se um número armazenado em *a0* é primo. Guarde 1 em *a0* caso seja, e 0 caso contrário.

6. Escreva algoritmos (em prosa ou pseudocódigo) para transformar os 3 tipos de loop da linguagem C (while, for e do-while) em Assembly.

7. (Desafio) Escreva um programa que faça um loop 5 vezes, porém a única instrução de salto permitida é *jalr*, e não é permitido o uso do seu imediato (ou seja, o imediato de todas as ocorrências de *jalr* no código deve ser 0). Chamadas de ambiente (*ecall* e *ebreak*) também não são permitidas. Dica: utilize as instruções *slti* e *andi*.

# 3 Instruções de acesso à memória

Semelhantemente as seções anteriores, utilize os comandos *set* e *print* para testar as resoluções, dessa vez também na memória (com a sintaxe @). Crie também vetores no próprio código.

Nessa seção, é de extrema importância lembrar que inteiros são armazenados em 4 bytes na memória de processadores RISC-V!

Alguns exercícios dessa seção exigem conhecimento sobre a tabela ASCII.

1. Escreva um programa que armazene os n primeiros números naturais em um vetor, onde n está armazenado no registrador *a0*.

2. Escreva um programa que some os elementos de dois vetores A e B na memória, guardando em um terceiro vetor C, tal que $C_i = A_i + B_i$.

3. Escreva um programa que armazene a sequência de Fibonacci até o n-ésimo número na memória, onde n está armazenado no registrador *a0*. Para simplificar, considere apenas $n \geq 2$.

4. Escreva um programa que calcule a média aritmética dos valores em um vetor e guarde o resultado no registrador *a0*.

5. Dados dois vetores de mesmo tamanho: um vetor de valores e outro de pesos, escreva um programa que calcule a média ponderada dos valores do primeiro vetor com seus respectivos pesos no segundo vetor. Guarde o resultado no registrador *a0*.

6. (ASCII) Escreva um programa que transforme um texto de letras minúsculas para maiúsculas, utilizando um loop com instruções de salto. Use *ecall* para testar. Exemplo:

```
addi a7, zero, 3
addi a0, zero, texto
addi a1, zero, 13 ; tamanho do texto
ecall

; aqui vai o código para transformar o texto.

addi a7, zero, 3
addi a0, zero, texto
addi a1, zero, 13 ; tamanho do texto
```

```
  ecall

  ebreak

texto:
#isso eh texto
```

# 4 Chamada de funções

Siga a convenção de registradores e chamada de função SEMPRE!

1. Traduza o seguinte programa de C para Assembly. Use *ecall* para traduzir as chamadas *printf*. Considere que o argumento "a0" da função main é o registrador *a0*, use o comando *set* do emulador para testar o programa com diferentes números.

```
int fatorial(int n) {
  int f = 1;
  for (int i = 2; i <= n; i++) {
    f = f * i;
  }
  return f;
}

int main(int a0) {
  f = fatorial(a0);
  if (f > a0 * 10) {
    printf("O fatorial eh mais que 10x maior.\n");
  } else {
    printf("O fatorial eh menos que 10x maior.\n");
  }
}
```

2. Resolva o exercício anterior, porém troque a implementação da função de fatorial por uma recursão.