Atividades - Assembly RISC-V

- Autor: Rafael Grossi

Atividade 1

Faça um programa que leia um número do teclado, guarde num registrador, e imprima esse número na tela. Você já fez algo muito similar anteriormente, pode reutilizar seu código sem problemas. Teste seu programa com números positivos e negativos, o que aconteceu?

```
main:
    addi t0, zero, 4
    ecall

    addi t0, zero, 1
    ecall

    ret
```

O numero é imprimido corretamete, porém para utilizar operações sobre os numeros necessita-se fazer alterações para tratar o codigo corretamente...

```
.globl main
main:
    li t0, 4
    ecall
    add s0, zero, a0

    bgez s0, handle_positive

handle_negative:
    addi s1, s0, -7
    j print_result

handle_positive:
    addi s1, s0, 7

print_result:
    bltz s1, print_is_negative

print_is_positive:
```

```
    add a0, s1, zero
    li t0, 1
    ecall
    j end

print_is_negative:
    li a0, 45
    li t0, 2
    ecall

    neg a0, s1

    li t0, 1
    ecall

end:
    ret
```

Atividade 2

Observe seu código e a presença dele na memória (na região de código/instruções e dados). Responda:

O que é a seção `.data` e a seção `.text`? Quais tipos de informações cada uma armazena?

- **`.text`:** É a seção de "texto", ou código, armazena instruções executáveis do programa; read-only.
- **`.data`:** É a seção de "dados". Ela armazena dados estáticos (variaveis, constantes) inicializados do programa

Qual o endereço da instrução que lê o número do teclado do código acima, e qual é a sua codificação em hexadecimal?

- `addi t0, zero, 4` => `00 40 02 93`
- `ecall` => `00 00 00 73`

Qual é o papel da diretiva `.globl main`?

A diretiva `.globl main` faz a label `main` visível para o linker, que caracteriza onde está o ponto de entrada principal do programa para o sistema operacional.

O que é um rótulo (label)? Qual a diferença entre rótulos simbólicos e numéricos?

É um nome simbólico que o programador associa a um endereço de memória específico, serve como um marcador, para a referência de endereço em outras partes do código.

- **Rótulos Simbólicos:** São nomes descritivos, como `main`, `loop`, `minha_variavel`, `print_h`.
- **Rótulos Numéricos:** São números usados como rótulos (ex: `1:`, `2:`).

O que é o contador de localização e o que ele faz durante a montagem?

É uma variável interna do *assembler* (montador), para rastrear o endereço de memória da instrução ou diretiva de dados que está sendo processada no momento.

- Quando o montador encontra uma instrução (ex: `addi`), ele aloca o tamanho dessa instrução (4 bytes) e avança o LC em 4.
- Quando encontra uma diretiva de dados (ex: `.word`), ele aloca 4 bytes e avança o LC em 4. Se for `.byte`, avança 1.
- Quando o montador vê um rótulo (ex: `loop:`), ele armazena o valor *atual* do LC na tabela de símbolos, associando o nome `loop` a esse endereço.

Adicione no código acima duas variáveis na seção `.data` ... Qual o valor do contador de localização antes e depois de cada variável?

não consegui adicionar a parte .data ao simulador do risc-v

```
********   Parser Output    ***********
Error at line 36 in instruction "   x: .word 10"
Error at line 37 in instruction "   y: .byte 20"
Parsing Failed. 2 error(s)
```

Por que o endereço de y pode não ser múltiplo de 4?

Porque a diretiva `.byte` não possui requisitos de alinhamento; ela pode ser alocada em *qualquer* endereço de byte disponível. O endereço de `y` depende inteiramente do que veio antes dele.

- No exemplo dado (`x: .word 10`), `x` ocupa 4 bytes, então `y` *acaba* ficando em um endereço múltiplo de 4 (`...04`).
- Porém, se tivéssemos `z: .byte 5` antes de `y`, `z` estaria em `...00` e `y` estaria em `...01`, que não é múltiplo de 4.

O que a diretiva `.align 2` faria nesse caso?

A diretiva `.align 2` instrui o montador a avançar o Contador de Localização (LC) até o próximo endereço que seja múltiplo de , ou seja, **múltiplo de 4**, e garante que o dado *seguinte* comece em um endereço alinhado por palavra (4 bytes).

Atividade 3

Faça um programa que leia um caractere em letras maiúsculas do teclado, guarde num registrador e imprima a versão minúscula dele na tela.

```
main:
    addi t0, zero, 5
    ecall

    ori a0, a0, 32

    addi t0, zero, 2
    ecall

    ret
```

Atividade 4

Faça um programa que leia um caractere em letras minúsculas do teclado, guarde num registrador e imprima a versão maiúscula dele na tela.

```
main:
    addi t0, zero, 5
    ecall

    # 0b11011111
    andi a0, a0, 223

    addi t0, zero, 2
    ecall

    ret
```

Atividade 5

Faça um programa que leia um caractere do teclado, guarde num registrador e imprima uma versão dele na tela. Se o caractere for uma letra minúscula, imprima

a versão maiúscula. Se for uma letra maiúscula, imprima a versão minúscula. Caso contrário, imprima o caractere sem alterações.

```
.globl main
main:
    addi t0, zero, 5
    ecall
    add t1, zero, a0

    # 65 .. 90
    li t2, 65
    blt t1, t2, check_lower
    li t2, 90
    bgt t1, t2, check_lower

    ori a0, t1, 32
    j print

check_lower:
    # 97 .. 122
    li t2, 97
    blt t1, t2, print
    li t2, 122
    bgt t1, t2, print

    andi a0, t1, 223

print:
    addi t0, zero, 2
    ecall

    ret
```

Atividade 6

Faça um programa que leia um número do teclado entre 0 e 15, guarde num registrador, e imprima esse número na tela em hexadecimal utilizando apenas um dígito. Ao final, não deixe de imprimir a letra h.

```
.globl main
main:
    addi t0, zero, 4
    ecall
    add t1, zero, a0
```

```
    # 0..9 or digit A..F
    li t2, 10
    blt t1, t2, is_digit

is_letter:
    addi a0, t1, 55
    j print_hex

is_digit:
    addi a0, t1, 48

print_hex:
    addi t0, zero, 2
    ecall

    # h
    li a0, 104
    addi t0, zero, 2
    ecall

    ret
```

Atividade 7

Faça um programa que leia vários números do teclado, guarde num registrador, e imprima cada número na tela em hexadecimal utilizando 8 dígitos hexadecimal. Ao final de cada número, não deixe de imprimir a letra h. Seu programa deve parar quando o usuário digitar o número 0.

```
.globl main
main:
    addi t0, zero, 4
    ecall

    beq a0, zero, end

    add s0, zero, a0

    # counters
    li s1, 8
    li s2, 28

hex_loop:
    # end condition
```

```
    beq s1, zero, print_h

    # 4 bits
    srl t1, s0, s2
    # s0 shift s2 and F
    andi t1, t1, 15

    li t2, 10
    blt t1, t2, is_digit_7

is_letter_7:
    addi a0, t1, 55
    j print_digit_7

is_digit_7:
    addi a0, t1, 48

print_digit_7:
    addi t0, zero, 2
    ecall

    # counters
    addi s2, s2, -4
    addi s1, s1, -1
    j hex_loop

print_h:
    # h
    li a0, 104
    addi t0, zero, 2
    ecall

    j main

end:
    ret
```