Exercício 5 - AC 2 - João Comini César de Andrade

Atividade 1

```
# Programa que calcula z = ((12*x) + (66*y)) * 4
.data
prompt_x:   .asciz "Digite x: "
prompt_y:   .asciz "Digite y: "
result_msg: .asciz "Resultado z: "

.text
.globl main
main:
    # Lê o valor de x
    li a7, 4
    la a1, prompt_x
    ecall
    li a7, 5
    ecall
    mv s0, a0 # Salva x em s0

    # Lê o valor de y
    li a7, 4
    la a1, prompt_y
    ecall
    li a7, 5
    ecall
    mv s1, a0 # Salva y em s1

    # Calcula 12*x
    # 12*x = (8*x) + (4*x)
    slli t0, s0, 3  # t0 = x * 8
    slli t1, s0, 2  # t1 = x * 4
    add t0, t0, t1  # t0 = 12*x

    # Calcula 66*y
    # 66*y = (64*y) + (2*y)
    slli t1, s1, 6  # t1 = y * 64
    slli t2, s1, 1  # t2 = y * 2
    add t1, t1, t2  # t1 = 66*y

    # Soma os termos e multiplica por 4
    add t0, t0, t1  # t0 = (12*x) + (66*y)
    slli s2, t0, 2  # s2 = resultado * 4
```

```
    # Mostra o resultado final
    li a7, 4
    la a1, result_msg
    ecall
    li a7, 1
    mv a1, s2
    ecall

    # Fim do programa
    li a7, 10
    ecall
```

Respostas

1. Qual instrução é mais simples de executar no nível de hardware?

A instrução de deslocamento de bits (slli, srli) é muito mais simples e rápida, pois o circuito do processador para isso é menos complexo que o de multiplicação (mul).

2. O que acontece se o número for negativo e você usar srl (deslocamento lógico à direita)?

O número perde o sinal negativo e se torna positivo, fazendo com que o resultado da "divisão" fique errado. Isso ocorre porque o srl preenche os novos bits da esquerda com zeros.

3. Qual instrução deve ser usada para preservar o sinal?

Deve-se usar a instrução sra (deslocamento aritmético à direita). Ela mantém o sinal original do número (seja positivo ou negativo), garantindo que a divisão por potências de 2 funcione corretamente.

Atividade 2

```
.data
prompt:     .asciz "Digite um numero: "
newline:    .asciz "\n"

.text
.globl main
main:
    # 1. Pede e lê um número do teclado
    li a7, 4            # Código para imprimir string
    la a1, prompt       # Carrega o endereço do prompt
    ecall

    li a7, 5            # Código para ler um inteiro
    ecall
    mv s0, a0           # Salva o número lido em s0
```

```
    # 2. Testa o último bit do número com a operação AND
    #    Se (numero AND 1) == 0, o número é par.
    #    Se (numero AND 1) == 1, o número é ímpar.
    andi t0, s0, 1      # t0 = s0 & 1. Isola o bit menos significativo.

    # 3. Decide qual letra imprimir
    #    Se t0 for zero (par), pula para a seção "eh_par".
    beqz t0, eh_par

# eh_impar:
    # Se o programa chegou aqui, o número é ímpar.
    li a7, 11           # Código para imprimir um caractere
    li a1, 73           # Carrega o código ASCII de 'I'
    ecall
    j fim               # Pula para o final do programa

eh_par:
    # Se o programa pulou para cá, o número é par.
    li a7, 11           # Código para imprimir um caractere
    li a1, 80           # Carrega o código ASCII de 'P'
    ecall

fim:
    # Imprime uma nova linha para formatação
    li a7, 4
    la a1, newline
    ecall

    # Termina a execução
    li a7, 10
    ecall
```

Atividade 3

```
.text
.globl main
main:
    # 1. Pede e lê um número do teclado
    li a7, 4            # Código para imprimir string
    la a1, prompt       # Carrega o endereço do prompt
    ecall

    li a7, 5            # Código para ler um inteiro
    ecall
```

```
    mv s0, a0           # Salva o número lido em s0

    # 2. Testa os dois últimos bits com a operação AND
    #    Um número é múltiplo de 4 se (numero AND 3) == 0.
    andi t0, s0, 3      # t0 = s0 & 3. Isola os dois últimos bits.

    # 3. Decide qual letra imprimir
    #    Se t0 for zero, pula para a seção "eh_multiplo".
    beqz t0, eh_multiplo

# nao_eh_multiplo:
    # Se o programa chegou aqui, o número NÃO é múltiplo de 4.
    li a7, 11           # Código para imprimir um caractere
    li a1, 78           # Carrega o código ASCII de 'N'
    ecall
    j fim               # Pula para o final do programa

eh_multiplo:
    # Se o programa pulou para cá, o número é múltiplo de 4.
    li a7, 11           # Código para imprimir um caractere
    li a1, 83           # Carrega o código ASCII de 'S'
    ecall

fim:
    # Imprime uma nova linha para formatação
    li a7, 4
    la a1, newline
    ecall

    # Termina a execução
    li a7, 10
    ecall
```

Atividade 4

```
.data
prompt:      .asciz "Digite um numero (0 para parar): "
result_msg:  .asciz "Resultado (Impares - Pares): "
newline:     .asciz "\n"

.text
.globl main
main:
    # Inicializa as somas com zero
```

```
    li s0, 0          # s0 (soma_impares) = 0
    li s1, 0          # s1 (soma_pares) = 0

loop_leitura:
    # 1. Pede um número ao usuário
    li a7, 4
    la a1, prompt
    ecall

    # 2. Lê o número
    li a7, 5
    ecall
    mv s2, a0         # Salva o número lido em s2

    # 3. Verifica a condição de parada
    #    Se o número for 0, pula para o fim do programa.
    beqz s2, fim_loop

    # 4. Testa se o número é ímpar
    #    Se (numero & 1) == 0, o número é par.
    andi t0, s2, 1
    beqz t0, eh_par     # Se o resultado for zero, pula para eh_par

# eh_impar:
    # Se chegou aqui, o número é ímpar. Adiciona a s0.
    add s0, s0, s2      # soma_impares = soma_impares + numero
    j loop_leitura      # Volta para o início do laço para ler o próximo número

eh_par:
    # Se pulou para cá, o número é par. Adiciona a s1.
    add s1, s1, s2      # soma_pares = soma_pares + numero
    j loop_leitura      # Volta para o início do laço

fim_loop:
    # O laço terminou. Agora, calcula o resultado final.
    # Resultado = Soma(ímpares) - Soma(pares)
    sub s0, s0, s1      # s0 = s0 - s1

    # Imprime a mensagem de resultado
    li a7, 4
    la a1, result_msg
    ecall

    # Imprime o valor final calculado
```

```
    li a7, 1
    mv a1, s0           # Move o resultado para o argumento de impressão
    ecall

    # Imprime uma nova linha
    li a7, 4
    la a1, newline
    ecall

    # Termina a execução
    li a7, 10
    ecall
```

Atividade 5

```
.data
prompt_segredo: .asciz "Digite o numero segredo (chave): "
prompt_numero:  .asciz "Digite o numero para codificar/decodificar: "
result_msg:     .asciz "Numero resultante: "
newline:        .asciz "\n"

.text
.globl main
main:
    # 1. Pede e lê o número segredo (chave)
    li a7, 4
    la a1, prompt_segredo
    ecall
    li a7, 5
    ecall
    mv s0, a0           # Salva a chave em s0

    # 2. Pede e lê o número a ser processado
    li a7, 4
    la a1, prompt_numero
    ecall
    li a7, 5
    ecall
    mv s1, a0           # Salva o número em s1

    # 3. Aplica a operação XOR
    #    s2 = segredo XOR numero
    xor s2, s0, s1
```

```
    # 4. Mostra o resultado final
    li a7, 4
    la a1, result_msg
    ecall
    li a7, 1
    mv a1, s2           # Move o resultado para o argumento de impressão
    ecall

    # Imprime uma nova linha para formatação
    li a7, 4
    la a1, newline
    ecall

    # 5. Termina a execução
    li a7, 10
    ecall
```

Atividade 6

```
.data
prompt_segredo: .asciz "Digite o numero segredo (chave): "
prompt_numero:  .asciz "Digite um numero para codificar (0 para parar): "
result_msg:     .asciz "Codificado: "
newline:        .asciz "\n"

.text
.globl main
main:
    # 1. Pede e lê o número segredo (chave) UMA VEZ
    li a7, 4
    la a1, prompt_segredo
    ecall
    li a7, 5
    ecall
    mv s0, a0           # Salva a chave em s0 para uso futuro

loop_codifica:
    # 2. Pede um número ao usuário
    li a7, 4
    la a1, prompt_numero
    ecall

    # 3. Lê o número
    li a7, 5
```

```
    ecall
    mv s1, a0           # Salva o número lido em s1

    # 4. Verifica a condição de parada
    #    Se o número for 0, pula para o fim do programa.
    beqz s1, fim_programa

    # 5. Codifica o número com a chave usando XOR
    xor t0, s1, s0      # t0 = numero_lido XOR chave_secreta

    # 6. Imprime a mensagem "Codificado: "
    li a7, 4
    la a1, result_msg
    ecall

    # 7. Imprime o número já codificado
    li a7, 1
    mv a1, t0
    ecall

    # Imprime uma nova linha para melhor formatação
    li a7, 4
    la a1, newline
    ecall

    # 8. Volta ao início do laço para ler o próximo número
    j loop_codifica

fim_programa:
    # Termina a execução
    li a7, 10
    ecall
```

Atividade 7

```
.data
prompt:      .asciz "Digite um numero decimal: "
result_msg:  .asciz "Binario: "
newline:     .asciz "\n"

.text
.globl main
main:
    # 1. Pede e lê o número do teclado
```

```
    li a7, 4
    la a1, prompt
    ecall
    li a7, 5
    ecall
    mv s0, a0           # Salva o número a ser convertido em s0

    # Imprime a mensagem de resultado
    li a7, 4
    la a1, result_msg
    ecall

    # 2. Inicializa o contador do laço
    li s1, 32           # Vamos repetir 32 vezes, para cada bit

loop_imprime_bit:
    # 3. Verifica se o laço terminou
    #    Se o contador (s1) chegou a zero, pula para o fim.
    beqz s1, fim_loop

    # 4. Testa o bit mais significativo (MSB)
    #    A instrução 'bltz' (Branch if Less Than Zero) faz isso.
    #    Se o número em s0 for negativo, seu MSB é 1.
    bltz s0, imprime_um

# imprime_zero:
    # Se o programa chegou aqui, o MSB é 0.
    li a7, 11           # Código para imprimir um caractere
    li a1, '0'          # Carrega o caractere '0'
    ecall
    j continua_loop     # Pula a seção 'imprime_um'

imprime_um:
    # Se o programa pulou para cá, o MSB é 1.
    li a7, 11           # Código para imprimir um caractere
    li a1, '1'          # Carrega o caractere '1'
    ecall
    # Não precisa pular, a execução continua normalmente.

continua_loop:
    # 5. Desloca o número 1 bit para a esquerda
    #    Isso move o próximo bit para a posição mais significativa.
    slli s0, s0, 1
```

```
    # 6. Decrementa o contador
    addi s1, s1, -1

    # 7. Volta para o início do laço
    j loop_imprime_bit

fim_loop:
    # Imprime uma nova linha para formatação
    li a7, 4
    la a1, newline
    ecall

    # Termina a execução
    li a7, 10
    ecall
```