Exercício 5

- Autor: Rafael Grossi

Atividade 1

Faça um programa em assembly que que leia x e y do teclado, e realize a equação (utilizando deslocamentos):

`z=((12*x)+(66*y))*4;`

Compare o comportamento com o uso das instruções mul e div (quando disponíveis).

```
main:
  addi t0, zero, 4
  ecall
  add s0, s0, a0

  addi t0, zero, 4
  ecall
  add s1, s1, a0

  # 12x
  slli t0, s0, 3
  slli t1, s0, 2
  add s0, t0, t1

  # 66y
  slli t0, s1, 6
  slli t1, s1, 1
  add s1, t0, t1

  # z
  add a0, s0, s1
  slli a0, a0, 2

  # addi, t0, zero, 1
  # ecall

  ret
```

Responda:

a)

- Qual instrução é mais simples de executar no nível de hardware?

Instruções de deslocamento (slli, srai, srli) e soma (add) são muito mais simples de executar no nível de hardware do que uma instrução de multiplicação (mul)

b)

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

```
main:
  addi a0, a0, -2
  srli a0, a0, 2
  ret
```

Como o deslocamento imediato (srli), preenche com 0, o bit de sinal seria deslocado para direita, logo o número seria lido incorretamente, para resolver pode-se usar o aritimético (srla), que preencheria os bits com 0 ou 1 dependendo do bit de sinal original.

c)

- Qual instrução deve ser usada para preservar o sinal?

Vide letra b), a instrução ideal é o deslocamento aritimético, que mantem o bit de sinal propriamente.

Atividade 2

Faça um programa que leia um número do teclado e imprima a letra I se o número for ímpar ou a letra P se o número for par. Busque os códigos da letra I e P na tabela ASCII para poder imprimir o carecter corretamente.

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

  # check parity
  andi t0, a0, 1

  beq t0, zero, even
  j odd
```

```
odd:
  li a0, 73
  j print

even:
  li a0, 80
  j print

print:
  addi t0, zero, 2
  ecall
  ret
```

Atividade 3

- Você consegue fazer outra versão do programa que detecte se um número é múltiplo de 4? Nesse caso, imprima S para sim e N para não.
- Dica: A instrução OR faz uma operação OU lógica bit a bit, para todos os bits do número. Então, um OR entre os números 6 (0110) e 5 (0101) tem como resposta o número 7 (0111) pois todos os bits com valor 1 em, ao menos, um dos números foram mantidos como 1 no resultado final.

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

  # 10011 & 11
  # 10 & 11
  # 11 & 11
  # 100 & 11
  andi t0, a0, 3
  bne t0, zero, no

  j yes

yes:
  li a0, 83
  j print

no:
  li a0, 78
  j print

print:
```

```
  addi t0, zero, 2
  ecall
  ret
```

Atividade 4

- Faça um programa que leia múltiplos números do teclado. Seu programa deve parar quando for digitado o valor 0. Ao final do programa, ele deve imprimir o resultado da soma de todos os ímpares subtraindo da soma de todos os pares. Resultado = Soma(ímpares) - Soma(pares).

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

  # check for end loop
  beq a0, zero, print

  # check even
  andi t0, a0, 1
  bne t0, zero, odd
  j even


odd:
  add s0, s0, a0
  j main

even:
  add s1, s1, a0
  j main


print:
  sub a0, s0, s1
  addi t0, zero, 1
  ecall
  ret
```

Atividade 5

- Faça um programa que leia dois números do teclado: o segredo e o número a codificar. O programa deve imprimir o número codificado. Para isso, utilize a instrução XOR. Note que esse programa servirá tanto para codificar quanto para decodificar o número.

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

  add s0, zero, a0

  # code
  addi t0, zero, 4
  ecall

  xor a0, s0, a0

  addi t0, zero, 1
  ecall

  # undo secret
  xor a0, s0, a0

  addi t0, zero, 1
  ecall
  ret
```

Atividade 6

- Faça um programa que leia o segedo do teclado e, depois, leia vários números do teclado imprimindo cada um codificado em sequência. O programa deve parar quando for digitado o valor 0.

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

  add s0, zero, a0

input:
  # code
  addi t0, zero, 4
  ecall
  beq a0, zero, end

  xor a0, s0, a0
```

```
  addi t0, zero, 1
  ecall

  j input


end:
  ret
```

Atividade 7

- Faça um programa que leia um número do teclado e imprima o número em binário. Para isso, utilize as instruções AND e de deslocamento de bits. Como os números do seu processador são de 32 bits, você deve ter um laço for no seu código com 32 interações.

```
main:
  # input
  addi t0, zero, 4
  ecall
  add s0, zero, a0

  # counter start
  add s1, zero, zero
  # n bits
  li t1, 32

loop:
  beq s1, t1, end
  # increase counter
  addi s1, s1, 1

  # check MSB
  bltz s0, odd

even:
  addi a0, zero, 0
  j print

odd:
  addi a0, zero, 1

print:
  slli s0, s0, 1
  addi t0, zero, 1
  ecall
```

```
  j loop

end:
  ret
```