Fundação CECIERJ-Vice Presidência de Educação Superior a Distância

Curso de Tecnologia em Sistemas de Computação
Disciplina: Organização de Computadores
AD2 2°semestre de 2024

Nome – Marcos Paulo Silva Antunes

OBS: As questões marcadas como Bônus não valem nota nem ponto extra e não são corrigidas. Entretanto, são importantes preparatórios para a AP e para aprimorar o conhecimento no curso.

1) (2,0 pontos) Nesta questão vamos avaliar o simulador de arquitetura de computadores apresentado em:

http://www.assis.pro.br/public_html/davereed/KandS/

Para você fazer a questão, recomendamos fortemente que experimente testar os vários parâmetros da arquitetura e veja o que ocorre ao manipular os valores. Em particular, assista esse tutorial: https://www.youtube.com/watch?v=5mkuatnA7js. **Além disso, você pode e deve contar com os tutores para tirar dúvidas!**

Nosso primeiro objetivo é ilustrar o fluxo de dados dentro de uma arquitetura típica de computadores. O programa simula como os dados se movem entre os registradores, barramentos e a ULA (Unidade Lógica e Aritmética) durante a execução de instruções, como soma e subtração. Nesta questão,você irá manipular os valores de registradores e barramentos para entender melhor o funcionamento desse fluxo de dados.

a) Vamos começar considerando o caso mais simples:

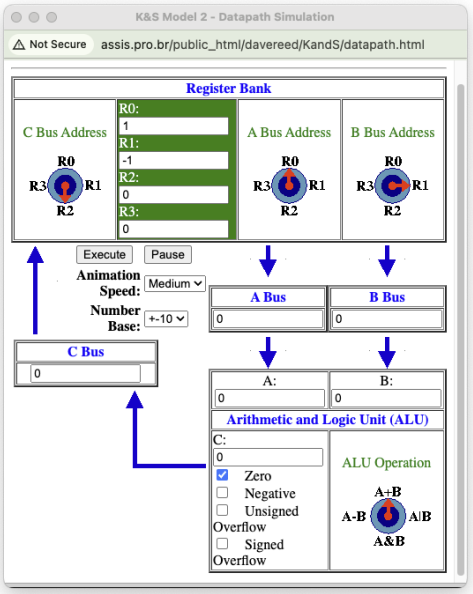
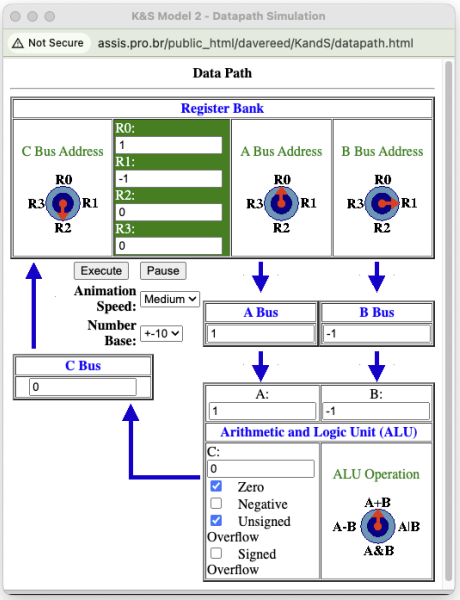
http://www.assis.pro.br/public_html/davereed/KandS/datapath.html

No simulador, considere os valores inicialmente configurados conforme a seguir, para o Banco de Registradores:

- R0:1
- R1:-1
- R2:0
- R3:0

Os outros valores são mantidos naqueles determinados por padrão pelo simulador, ou conforme figura abaixo.

Após executar o simulador, qual operação da ULA foi executada para produzir o valor 0 presente no Barramento C (C Bus)? Por que o bit “unsigned overflow” não estava inicialmente setado mas, depois da execução da instrução acima, o bit ficou setado?

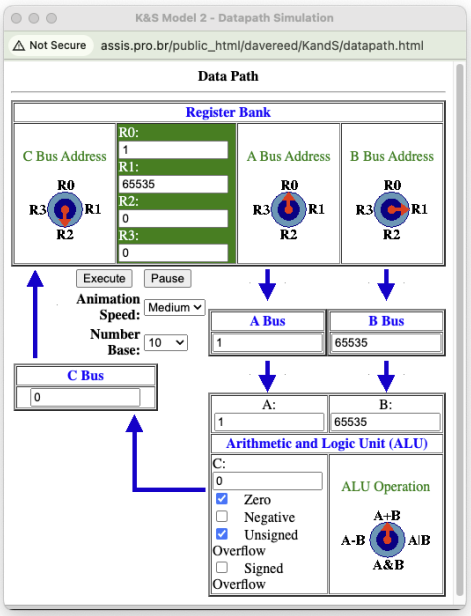
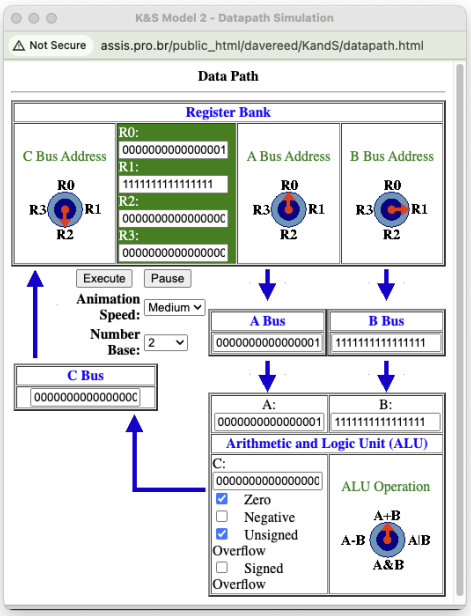
Antes da execução	Depois da execução
	

R.: Foi realizado uma operação de Soma para resultar no valor zero.

O bit “unsigned overflow” foi setado devido ao número (-1) ser convertido para $2^8 – 1$, o que resultaria em uma soma de 1 + 255.

Para ficar mais claro, podemos cnsiderar um tipo unsigned de 8 bits, -1 seria interpretado como 255 (maior número representado em 8 bits) porque -1 é equivalente a $2^8 – 1$ devido a como os computadores representam os números na memória usando o sistema binário. Logo, ao realizar a operação, o resultado seria 256 que excede o limite de 0 a 255, resultando em um overflow e o resultado será imediatamente igual a 0.

b) Troque “Number base” de+-10 para 10 e depois para 2, e explique, em cada caso, os valores que aparecem nas entradas da ULA. As imagens abaixo refletem o que ocorre com o simulador ao fazer as trocas.

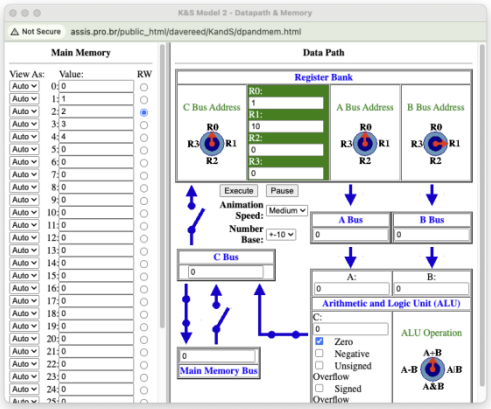
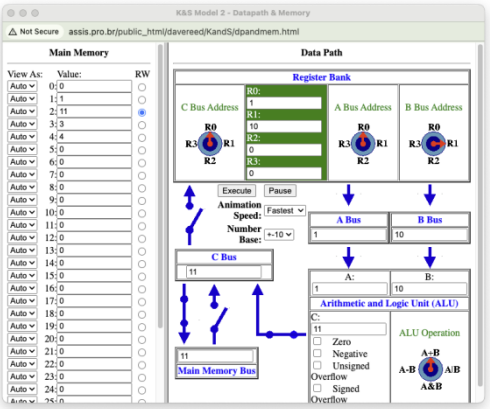
Number base = 10	Number base = 2
	

R.: Essa mudança ocorre, pois estamos mudando a base que representa este mesmo número. Na base 10, (-1) é representado por 65535, já na base 2, é representado por 1111111111111111, o mesmo irá ocorrer com o número 1 que na base 10 é igual a 1 e na base 2 é igual a 0000000000000001.
Note que, estes números estão representados em 16 bits, caso fossem representados em 8 bits, a representação númerica nas bases não seria o mesmo.

c) Agora considere a máquina com memória:

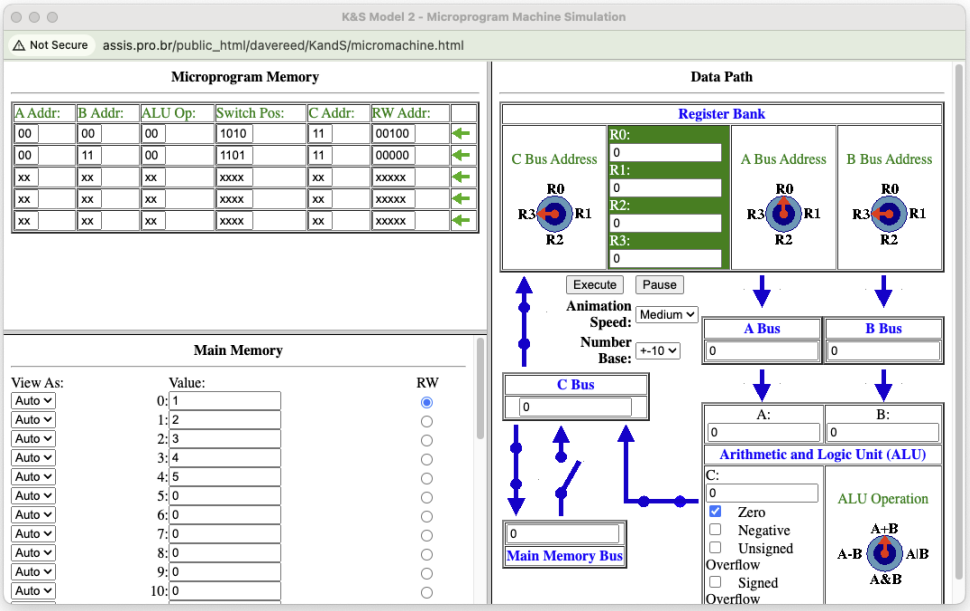
http://www.assis.pro.br/public_html/davereed/KandS/dpandmem.html

Inicialize a máquina conforme ilustrado abaixo na imagem à esquerda. Por que o programa gera o resultado mostrado na imagem à direita? Explique passo a passo.

Configuração inicial	Configuração final
	

R.: O programa irá somar 1 com 10 e resultará 11, o resultado será armazenado em C bus e, então, irá armezanar o resultado na Main Memory Bus, substuindo o valor 2 que foi retirado do endereço de memória “2” e, então, este valor será guardado no endereço de memória “2”.

d) Agora considere a máquina com microcontrolador: http://www.assis.pro.br/public_html/davereed/KandS/micromachine.html O que faz o seguinte programa?



R.: Abaixo esta passo a passo do que o programa irá fazer.

1. **Primeira execução**
   a) O programa irá ler a primeira linha do Microprogram Memory e irá realizar as alterações no fluxo conforme a programação
   b) Irá ler o valor armezanado no R0
   c) Armazena os valores em A bus e B bus, respectivamente
   d) Realiza a operação de soma
   e) Armazena o valor que esta na posição de memória “4” da Main Memory na Main Memory Bus
   f) Transmite o valor da Main Memory Bus para C Bus
   g) Transmite o valor da C Bus para R3

2. **Segunda execução**
   a. O programa irá ler a segunda linha do Microprogram Memory e irá realizar as alterações no fluxo conforme a programação
   b. Irá realizar a leitura dos valores armazenados em R0 e R3
   c. A bus e B bus recebe os valores de R0 e R3, respectivamente
   d. O programa realiza a soma entre A e B
   e. Busca o valor armezanado na posição “0” da Main Memory e então armazena na Main Memory Bus
   f. Transmite o resultado da soma para C Bus
   g. Armazena o valor de C Bus na Main Memory e em R3
   h. Armazena o valor da Main Memory Bus na posição “0” da Main Memory

2) (2,0 pontos) Considere dois números inteiros X e Y representados em complemento a 2, ambos com 8 bits de tamanho cada um. Seja X = 10111011 em binário (complemento a 2). Responda os itens a seguir.

a) Assuma que Y é positivo. É possível gerar um overflow com a operação X + Y? Caso sim, qual o **menor valor de Y** para o qual X + Y gera um **overflow**? Qual o **maior valor de Y** para o qual X + Y gera um overflow?

R.: Primeiro,vamos descobrir qual é o valor de X em decimal.

X = 10111011 => 01000100 + 1 = 01000101

Logo, X = -69

Como Y é um positivo de 8 bits, ele irá variar de 0 a 127, com isso, precisamos encontrar o menor valor para que Y cause um overflow na soma.

X + Y > 127 Y > 127 + 69

-69 + Y > 127 Y > 196

Logo, para causar um overflow Y deve ser maior que 196, o que não é possível pois Y é um número complemento a 2 positivo, ou seja, ele varia entre 0 a 127.

b) Assuma que Y é negativo. É possível X - Y gerar um overflow? Caso sim, qual o **menor e o maior valor de Y** para o qual X - Y gera **overflow**? Qual o **menor valor de Y** para o qual X - Y gera um resultado válido, sem overflow?

R.: Com base na questão anterior, sabemos que X equivale a -69 em decimal, com isso teremos:

X – Y < -128 –Y < -128 + 69 Y > 59

–69 – Y < -128 –Y < -59

R.: Para que Y gere um overflow com seu **menor valor**, ele deverá assumir um valor de **-128**. agora o **maior valor** de Y que causa um overflow é **-60**. O **menor valor** para Y gerar um **resultado válido** sem overflow é de -**59**

c) O que ocorre se multiplicarmos X por 2, fazendo **shift para a esquerda**? Qual o resultado em decimal?

R.: Multiplicar um binário por dois, é a mesma coisa que deslocar seus bits para a esquerda. Observe abaixo a demonstração:

X = 10111011

2 * X = 1 01110110

Note que o número marcado é o bit carrey, que será descartado, com isso teremos 01110110, que em decimal é igual a 118. Este resultado ocorre devido ao overflow na representação binária, resultando em um valor diferente da multiplicação em decimal (-69 * 2 = -138 ≠ 118)

d) O que ocorre se dividirmos X por 2, fazendo **shift para a direita**? Qual o resultado em decimal?

R.: Dividir um binário por dois, é a mesma coisa que deslocar seus bits para a direita. Observe abaixo a demonstração:

X = 10111011 = -69 -69 / 2 = -34,5

X / 2 = 11011101 = -35 -35 ≠ -34,5 (overflow)

Como visto no cálculo, ao realizar a divisão, foi gerado um overflow devido ao fato de um número complemento a 2 de 8 bits não poder representar números fracionários.

3) (1,0 ponto) Considere um sensor que lê valores em formato IEEE-754. O sensor armazena o seguinte valor:

- **Sinal**: 0 (indicando um número positivo)
- **Mantissa**:1100100 (embinário)
- **Expoente**:-3 (emdecimal)

Qual é o valor desse número em decimal, de acordo com a notação IEEE-754? Se o número original fosse representado como mesmo expoente, mas amantissa tivesse sido truncada para os 3 primeiros bits, qual seria o novo valor do número em decimal?

$$(-1)^{Sinal}\ x\ 1.Mantissa\ x\ 2^{expoente}$$
$$(-1)^{0}\ x\ 1.1100100\ x\ 2^{-3}$$
$$1\ x\ 1.1100100\ x\ 2^{-3}$$

Converter a mantissa:

$$1.1100100 = 1 + \frac{1}{2} + \frac{1}{4} + \frac{0}{8} + \frac{0}{16} + \frac{1}{32} + \frac{0}{64} + \frac{0}{128}$$
$$= 1 + 0.5 + +0.25 + 0 + 0 + 0.03125 + 0 + 0$$
$$= 1.78125$$

Aplicar o expoente:

$$1.78125x\ 2^{-3} = 1.78125\ \mathrm{x}\frac{1}{8} = 0.22265625$$

Logo, o número em decima é 0.22265625.

Considerando se a mantissa fosse truncada para os 3 primeiros bits:

$$(-1)^0 \; x \; 1.110 \; x \; 2^{-3}$$

$$(-1)^0 \; x \; \left(1 + \frac{1}{2} + \frac{1}{4} + \frac{0}{8}\right) x \; 2^{-3}$$

$$(-1)^0 \; x \; (1 + 0.5 + 0.25 + 0) x \; 2^{-3} = 1.75 \; x \frac{1}{8} = 0.21875$$

Logo, se a mantissa fosse truncada para os 3 primeiros bits, o número decimal seria 00.21875

4) (1,5 ponto) Considere os seguintes cenários envolvendo dois processadores distintos que executam a mesma tarefa:

- Processador RISC
  - Frequência de Operação: 1.2 GHz
  - Número de instruções para completar a tarefa: 3.000 instruções
  - Ciclos de clock: Em média, 2 ciclos por instrução
  - Consumo de energia: Cada instrução consome, em média, 0,8 unidades de energia.
- Processador CISC
  - Frequência de Operação: 1.8 GHz
  - Número de instruções para completar a tarefa: 1.000 instruções
  - Ciclos de clock: Em média, 4 ciclos por instrução
  - Consumo de energia: Cada instrução consome, em média, 1,2 unidades de energia.

Desconsidere o uso de pipeline e otimizações como predição de branch e execução fora de ordem.

Perguntas:

a) (0,5 ponto) Calcule o número total de ciclos de clock necessários para cada processador completar a tarefa. Considerando as frequências de operação, determine qual processador completa a tarefa mais rapidamente. Justifique sua resposta

R.: Abaixo esta o calculo para encontrar o que foi pedido.

$$.Total\ de\ Ciclos\ \boldsymbol{RISC} = 3000 \; x \; 2 = 6000 \frac{ciclos}{instrução}$$

$$Total\ de\ Ciclos\ \mathbf{CISC} = 1000 \; x \; 4 = 4000 \frac{ciclos}{instrução}$$

$$Tempo\ \boldsymbol{RISC} = \frac{6000}{1{,}2 \; x \; 10^9} = \frac{6 \; x \; 10^3}{1{,}2 \; x \; 10^9} = 5 \; x \; 10^{-6}$$

$$Tempo\ \mathbf{CISC} = \frac{4000}{1{,}8 \; x \; 10^9} = \frac{4 \; x \; 10^3}{1{,}8 \; x \; 10^9} = 2{,}22 \; x \; 10^{-6}$$

De arcodo com os calculos feitos, o processador **CISC leva menos tempo** para complementar as tarefas.

b) (0,5 ponto) Calcule o consumo total de energia para completar a tarefa em cada processador. Considerando apenas o consumo de energia, qual processador é mais eficiente?

R.: Para isso, vamos multiplicar a quantidade de instruções que cada processador executada por unidade de energia consumida por instrução.

$$Consumo\ \boldsymbol{RISC} = 3000\ x\ 0{,}8 = 2400\ unidades\ de\ energia$$
$$Consumo\ \boldsymbol{CISC} = 1000\ x\ 0{,}8 = 1250\ unidades\ de\ energia$$

Em termos de consumo de energia, o processador **CISC é mais eficiente**.

c) (0,5 ponto) Discuta os trade-offs entre as arquiteturas RISC e CISC em termos de desempenho, consumo de energia e complexidade de hardware. Em que tipos de aplicações cada arquitetura se destaca e por quê?

R.: A **arquitetura RISC** é projetada para executar um número maior de instruções simples, o que provoca a necessidade de mais instruções para completar uma tarefa, porém, isso permite a implementação eficiente de técnicas como pipeline. Além disso, o alto número de instruções implica em um consumo de energia maior,no entanto, o gasto de energia pode ser otimizado melhorando o gerenciamento de energia. Além disso, o hardware tende a ser mais simples em comparação a um dispositivo que utiliza a arquitetura CISC.

Agora, a **arquitetura CISC** é uma arquitetura focada em desempenho, logo, ela irá executar operações mais complexas em um menor número de instruções, o que faz com que o gasto de energia seja menor, mas note que em certos casos, o consumo de energia pode ser alto. Com tudo, pelo fato de executar instruções complexas, o hardware tende a ser mais complexo e caro.

A arquitetura RISC pode ser muito útil em sistemas embarcados e dispositivos móveis, onde o foco é a simplicidade e a eficiência de energia. Já a arquitetura CISC, pode ser muito bem aplicada em desktop e servidores, onde o desempenho é mais requisitado.

5) (2,0 pontos) Considere um sistema no qual o número de ciclos de relógio para uma execução de operação por programa é igual a 80 ciclos. O processador utiliza um relógio de 2.400 MHz para executar as instruções, e nenhuma transferência de dados pode ser perdida. Determine o overhead, em termos de fração de tempo de CPU consumida, que ocorre quando se utiliza a interface por programa para os seguintes dispositivos

a) (0,5 ponto) Um mouse de alta precisão deve ser interrogado 100 vezes por segundo para garantir que nenhum movimento seja perdido.

R.: Com o mouse sendo consultado 100 vezes por segundo, cada operaão de consulta consome 80 ciclos. Logo, serão necessários 8000 (100 x 80) ciclos / segundo para as operações de consulta ao mouse.

$$Overhead = \frac{8000}{2{,}4\ x\ 10^8} = \frac{8\ x\ 10^3}{2{,}4\ x\ 10^8} = 3{,}33\ x\ 10^{-5} = 0{,}0000333\ ou\ 0{,}00333\%$$

b) (0,5 ponto) Uma webcam de alta resolução transfere dados para o processador em unidades de 256 bits e possui uma taxa de transferência de dados de 1.200 MB/segundo.

$Taxa\ de\ transferência = 1200 \frac{MB}{s} = 1258291200 \frac{\text{B}}{\text{s}}$

$Em\ cada\ operação\ é\ transferiada\ uma\ unidade\ de\ 256\ bits\ ou\ 32\ bytes.$

$Operações\ por\ segundo = \frac{1258291200}{32} = 393321600$

$Total\ de\ ciclos = 393321600\ x\ 80 = 3145728000 = 3145728\ x\ 10^3 \frac{ciclos}{s}$

$Overhead = \frac{3145728\ x\ 10^3}{2{,}4\ x\ 10^8} = 1310720\ x\ 10^{-5} = 13{,}11 \text{ ou } 1311\%$

c) (0,5 ponto) Uma impressora 3D recebe comandos do processador em unidades de 512 bits e possui uma taxa de transferência de dados de 300 MB/segundo.

$Taxa\ de\ transferência = 300 \frac{MB}{s} = 314572800 \frac{\text{B}}{\text{s}}$

$Em\ cada\ operação\ é\ transferiada\ uma\ unidade\ de\ 512\ bits\ ou\ 64\ bytes.$

$Operações\ por\ segundo = \frac{314572800}{64} = 4914575$

$Total\ de\ ciclos = 4914575\ x\ 80 = 393166000 = 393166\ x\ 10^3 \frac{ciclos}{s}$

$Overhead = \frac{393166\ x\ 10^3}{2{,}4\ x\ 10^8} = 163819{,}2\ x\ 10^{-5} = 1{,}6382 \text{ ou } 163{,}82\%$

d) (0,5 ponto) Para os valores obtidos nos itens anteriores, discuta brevemente as implicações do overhead obtido para cada dispositivo e como a operação de E/S por interrupção poderia reduzir essas implicações.

Overhead do Mouse: o overhead é extramente baixo, indicando que a CPU dedica uma fração mínima de seu tempo para interrogar o mouse, sugerindo que a interrogação do mouse não impacta de forma significativa o desempenho geral do sistema. A operação de E/S por interrupção não é estritamente necessária neste caso, porém poderia garantir uma eficiência maior.

Overhead da Webcam: o overhead elevado indica que a CPU gasta muito tempo processando as transferências de dados deste dispositivo, resultando em baixo desempenho. Com isso, a utilização de E/S por interrupção permitiria que a CPU procesasse outras tarefas enquanto aguardava a transferência de dados, reduzindo o overhead.

Overhead da Impressora 3D: também apresenta um overhead alto, fazendo com que boa parte do tempo da CPU seja consumida por transferência de dados, causando possíveis travamentos. Para este caso, a implementação de E/S por interrupção seria bastante útil para diminuir a demanda da CPU.

6) (1,5 pontos) Crie um conjunto de instruções de um operando definidas em Linguagem Assembly, utilizando endereçamento direto e imediato, necessárias para a realização de operações aritméticas. Elabore um programa para o cálculo da equação abaixo, utilizando endereçamento direto para acessar os valores de X, Y e W armazenados em posições de memória, e endereçamento imediato para os valores constantes, como 4, 2 e 3. O resultado deve ser armazenamento na posição de memória Z. X, Y, W e Z são endereços de memória. O seu conjunto de instruções deve explicitamente mencionar quando o endereçamento é direto ou imediato:

Z = X * (Y + 4) – 2 * (W – 3)

Orientações:

- Para endereçamento imediato, utilize o símbolo # antes do valor. Exemplo: ADD #5 (soma o valor imediato 5 ao acumulador).
- Para endereçamento direto, utilize o nome da variável (ou seu endereço de memória). Exemplo 1: LOAD [A] (carrega o valor armazenado em A no acumulador) Exemplo 2: Se B representar o endereço de memória 5 que contém ovalor 3, as seguintes instruções são equivalentes
    - LOAD[B] (carrega o valor armazenado em B = 5 no acumulador)
    - LOAD[5] (carrega o valor armazenado na posição de mem.5 no acumulador)
    - LOAD #3(carrega o valor 3 no acumulador)
- Você pode utilizar até 5 registradores (R1 a R5) para armazenar resultados intermediários

Explique brevemente porque o endereçamento imediato é útil ou necessário no contexto desta operação.

Observação:A questão bônus a seguir pode ser útil para ajudar a resolver este problema.

LDA Y	=> (ACC) ← (Y)
ADD #4	=> (ACC) ← (ACC) + 4
MUL X	=> (ACC) ← (ACC) * (X)
SUB #2	=> (ACC) ← (ACC) – 2
STR R1	=> (R1) ← (ACC)
LDA W	=> (ACC) ← (W)
SUB #3	=> (ACC) ← (ACC) – 3
MUL #2	=> (ACC) ← (ACC) * 2
STR R2	=> (R2) ←(ACC)
LDA R1	=> (ACC) ← (R1)
SUB R2	=> (ACC) ← (ACC) – (R2)
STR Z	=> (Z) ← (ACC)

Questão bônus: esta questão é para os alunos familiarizados com Python. Considere o seguinte programa em Python. Neste programa, a ULA dentro da CPU recebe como argumento dois operandos e um operador, e gera um resultado. Ajuste o programa para considerar uma versão da ULA mais simples na qual as instruções envolvem apenas um operando explícito, sendo que o segundo operando é sempre o registrador acumulador.

```
ClassCPU:
    def __init__(self):
        self.memory = [0] * 256 # Simula a memória com 256 posições.
    def load_to_memory(self, value, address):
        self.memory[address] = value
    def fetch(self, address):
        return self.memory[address]
    def execute (self, op, val1, val2):
        if op == 'add':
            return val1 + val2
        elif op == 'sub':
            return val1 - val2
        elif op == 'mul':
            return val1 * val2
        elif op == 'div':
            return val1 // val2 # Divisão inteira para simplificar.
    def run(self):
        val1 = int(input("Digite o primeiro valor: ")) val2 = int(input("Digite o
        segundo valor: "))
        op = input("Escolha a operação (add, sub, mul, div): ")
        self.load_to_memory(val1, 0)
        self.load_to_memory(val2,1) fetched_val1 = self.fetch(0) fetched_val2 =
        self.fetch(1)
        result = self.execute(op, fetched_val1, fetched_val2) print(f'Resultado
        de{fetched_val1}{op}{fetched_val2}={result}')

if __name__=="__main__":
    cpu = CPU()
    cpu.run()
```

Em particular, preencha as lacunas no código abaixo para implementar o simulador básico de CPU proposto. O simulador deve ser capaz de carregar valores na memória e no acumulador, buscar dados da memória, e executar operações aritméticas simples. Cada operação deve modificar o estadodoacumulador.Aplique os conhecimentos de arquitetura de computadores que você aprendeu para realizar as operações corretamente.***Quais as vantagens e desvantagens de se ter uma arquitetura em que todas as instruções só recebem um operando explícito?***

```
ClassCPU:
        def __init__(self):
                self.memory=[0] * 256 # Simula a memória com 256 posições.
                self.accumulator = 0 # Registrador acumulador
        defload_to_memory(self, value, address):
                # Lacuna 1: Carregar um valor na memória no endereço especificado.
                pass

        Def fetch(self, address):
                #Lacuna 2: Retornar o valor do endereço de memória especificado.
                pass

        defload_to_accumulator(self,value):
                #Lacuna 3: Carregar um valor no acumulador.
                pass

        def execute(self, op, mem_address):
                memory_value=self.fetch(mem_address)
                if op == 'add':
                        #Lacuna 4: Adicionar o valor da memória ao acumulador.
                        pass
                elifop=='sub':
                        #Lacuna: Subtrair o valor da memória do acumulador.
                        pass
                elifop=='mul':
                        #Lacuna 6:Multiplicar o acumulador pelo valor da memória.
                        pass
                elifop=='div':
                        if memory_value != 0:
                                #Lacuna 7:Dividir o acumulador pelo valor damemória.
                                pass
                        else:
                                print("Erro: Divisão por zero")
                elif op == 'neg':
                        #Lacuna 8: Inverter o sinal do acumulador.
                pass

        defrun(self):
                initial_value=int(input("Digite o valor inicial para o acumulador: "))
                self.load_to_accumulator(initial_value)
                mem_address=int(input("Digite o endereço de memória: "))
                op = input("Escolha a operação (add, sub, mul, div, neg): ")
                self.execute(op, mem_address)
                print(f'Resultado no acumulador= {self.accumulator}')

if __name__ =="__main__":
        cpu = CPU()
        cpu.run()
```

R.: Abaixo esta código com as lacunas preenchidas.

```python
class CPU:
    def __init__(self):
        self.memory = [0] * 256  # Simula a memória com 256 posições.
        self.accumulator = 0  # Registrador acumulador

    def load_to_memory(self, value, address):
        # Lacuna 1: Carregar um valor na memória no endereço especificado.
        self.memory[address] = value

    def fetch(self, address):
        # Lacuna 2: Retornar o valor do endereço de memória especificado.
        return self.memory[address]

    def load_to_accumulator(self, value):
        # Lacuna 3: Carregar um valor no acumulador.
        self.accumulator = value

    def execute(self, op, mem_address):
        memory_value = self.fetch(mem_address)
        if op == 'add':
            # Lacuna 4: Adicionar o valor da memória ao acumulador.
            self.accumulator += memory_value
        elif op == 'sub':
            # Lacuna 5: Subtrair o valor da memória do acumulador.
            self.accumulator -= memory_value
        elif op == 'mul':
            # Lacuna 6: Multiplicar o acumulador pelo valor da memória.
            self.accumulator *= memory_value
        elif op == 'div':
            if memory_value != 0:
                # Lacuna 7: Dividir o acumulador pelo valor da memória.
                self.accumulator //= memory_value
            else:
                print("Erro: Divisão por zero")
        elif op == 'neg':
            # Lacuna 8: Inverter o sinal do acumulador.
            self.accumulator = -self.accumulator

    def run(self):
        initial_value = int(input("Digite o valor inicial para o acumulador: "))
        self.load_to_accumulator(initial_value)

        mem_address = int(input("Digite o endereço de memória: "))
        value = int(input("Digite o valor para carregar na memória: "))
        self.load_to_memory(value, mem_address)

        op = input("Escolha a operação (add, sub, mul, div, neg): ")
        self.execute(op, mem_address)

        print(f'Resultado no acumulador = {self.accumulator}')


if __name__ == "__main__":
    cpu = CPU()
    cpu.run()

```

As **vantagens** de se ter uma arquitetura em que todas as instruções só recebem um operando explícito são: **simplicidade de decodificação**, **menor uso de memória** e **eficiência em operações sequenciais**.
As desvantagens são: **depêndecia do acumulador**, **dificuldade em escalar aplicações mais complexas**.

Contexto adicional para questão 1 sobre simulador:

Neste apêndice, incluímos contexto adicional para a questão 1, sobre o simulador.

1. **Banco de Registradores (Register Bank)**:O banco de registradores é uma pequena memória de alta velocidade que armazena dados temporários usados durante o processamento. Esses registradores, como R0, R1, R2 e R3, podem conter valores que são lidos ou escritos durante a execução das operações.
2. **Barramentos A, B e C (A Bus, B Bus, C Bus)**: Barramentos são vias de comunicação usadas para transferir dados entre os diferentes componentes do processador. O barramento A conecta um registrador à ULA, assim como o barramento B. O barramento C recebe o resultado da operação da ULA e o envia de volta para o banco de registradores.
3. **Unidade Lógica e Aritmética (ULA)**: A ULA realiza operações como soma, subtração, AND lógico, OR lógico, entre outras. Os valores que entram na ULA vêm dos barramentos A e B, e o resultado da operação é enviado pelo barramento C.

Item a) Para responder ao item 1, pode ser interessante refletir sobre os seguintes pontos:

- A ULA está configurada para receber dois valores, um do barramento A e outro do barramento B. A operação realizada pela ULA resulta em um valor que é transmitido pelo barramento C. Sabendo que o valor do barramento C é 0, qual operação foi realizada pela ULA?

- A ULA, além de realizar operações, também gera sinais de status, como a flag de Zero (quando o resultado é 0), Negativo (quando o resultado é negativo), ou Overflow (quando há um estouro no limite dos valores). Com base no resultado da operação realizada, qual das condições de status seria ativada?

Item b) No simulador K&S, o "Number base" define a forma como os números são representados. Quando trocamos a base numérica, estamos alterando a forma como o valor é interpretado pela ULA (Unidade Lógica e Aritmética). A base "+-10" é uma representação com sinal (onde números podem ser positivos ou negativos). Já a base "10" é a representação decimal sem sinal e a base "2" é a binária, que também pode ser sem sinal.

1. **Explicação em detalhe:**
    - Quando a base é**+-10**, o simulador entende os números como inteiros com sinal, ou seja, tanto números positivos quanto negativos podem ser manipulados. Por exemplo, se R1 tem o valor "-1" e R0 tem "1", a ULA opera com essas representações com sinal.
    - Quando a base é trocada para **10**, os números passam a ser interpretados como **valores sem sinal** (unsigned integers).
    - Com a base **2** (binária), a ULA irá operar com números em binário puro. Um número como 3 será representado por "11" em binário
2. **Passo a Passo no simulador:**
    - Troque a base no simulador e observe como os valores das entradas da ULA mudam. Isso permitirá que você compreenda a influência da representação dos números no resultado das operações da ULA.

Item c) O simulador agora não se limita ao uso de registradores e barramentos. Ele introduz a memória, onde você pode armazenar e recuperar valores, expandindo a capacidade de manipulação de dados. Quando uma instrução é executada, ela pode carregar ou gravar dados em posições específicas da memória.

No exemplo em particular mostrado nesta questão, note que a seleção da posição de memória 2 faz o papel de indicar que esta será a posição de memória afetada pela instrução, o que corresponde a termos o registrador REM setado em 2.

Você deve usar o simulador que inclui memória no K&S para observar como as instruções afetam o conteúdo da memória.

1. **Explicação detalhada:** No exemplo proposto,o registrador REM (Registro de Endereço de Memória) é configurado para a posição de memória 2. Isso significa que a instrução a ser executada irá afetar essa posição específicada memória.

2. **Passo a Passo:**
    - Inicialize a máquina conforme ilustrado no simulador.
    - Execute a instrução que seleciona a posição de memória 2.
    - A memória será modificada de acordo com a operação realizada. O valor final armazenado na posição de memória dependerá da operação realizada pela ULA e dos valores presentes nos barramentos A e B.
    - Explique como o resultado gerado corresponde à modificação da posição de memória 2.

    Item d) Note que os valores dos códigos de instruções são intuitivos.

    - A Addr: registrador de onde será feita a leitura para A bus
    - B Addr:registrador de onde será feita a leitura para B bus
    - ALU Op: 0 = soma, 1 = ou, 2 = e, 3 = subtração
    - Switch Pos: os 4bits controlam os estados dos 4 switches (chaves)
    - C Addr: registrador onde será feita a escrita do C bus
    - RW Addr: posição de memória de onde será feita leitura ou escrita.

Contexto: No simulador com microcontrolador, as instruções não são apenas operadas diretamente pelos barramentos e registradores, mas por um conjunto de comandos programados que manipulam as leituras e gravações de dados nos registradores e na memória. A execução dessas instruções segue a lógica de um microcontrolador simples.

Explicação detalhada:

- O programa simulador realiza operações específicas nos registradores e na memória. Por exemplo:
    - **A Addr** indica o registrador cujo valor será lido para o barramento A.
    - **B Addr** indica o registrador que será lido para o barramento B.
    - **ALU Op** define a operação que a ULA executará (soma, subtração, AND lógico, etc.).
    - **C Addr** é oregistrador onde o resultado será armazenado, ou seja, ele recebe o valor do barramento C.

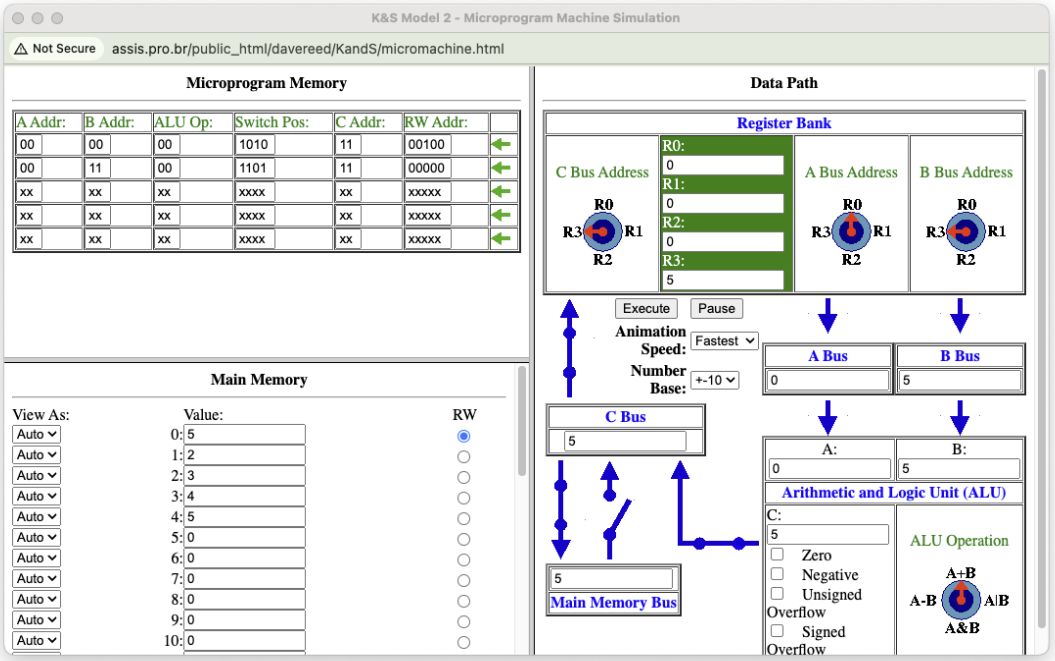
- **RWA ddr** define a posição de memória que será lida ou escrita.

Passo a Passo:

- Execute o programa no simulador. À medida que as instruções são processadas, observe como os valores são transferidos entre os registradores e a memória, de acordo com as instruções.
- Explique o comportamento do programa com base nos valores dos barramentos e nas operações da ULA, verificando o que o microcontrolador faz com os valores.

Importante! Para experimentar com oprograma,tenteclicaremcimadassetasazuis (switches) do lado direito e em seguida clique na seta verde (apontador) do lado esquerdo, no microcontrolador. Assim, você irá entender a relação entre o microprograma e a posição dos switches.

Dica: a saída do programa é esta



Contexto adicional para questão 2:

Parte a)

- **Dicas**:
    - O valor de X = 10111011 em binário é negativo em complemento a 2.
    - Overflow ocorre quando o resultado de uma soma de números com sinal excede o limite da representação de 8 bits. Para números positivos, o overflow ocorre quando o resultado ultrapassa o valor máximo representável (127 em decimal).
    - Tente diferentes valores de Y até encontrar o limite no qual o resultado de X + Y causa overflow.

Parte b)

- **Dicas**:
    - O valor de Y será negativo, então subtrair Y é equivalente a soma do valor positivo de Y em complemento a 2.
    - Explore o intervalo de Y negativo e veja quando X – Y ultrapassa o limite representável.

Parte c)

- **Dicas**:
    - Multiplicar um número por 2 equivale a deslocar seus bits uma posição à esquerda (shift left).
    - Verifique se o valor resultante ainda pode ser representado com 8bits ou se ocorre overflow durante a operação.
    - Como o número X é negativo, observe o comportamento do bit de sinal após o deslocamento.

Parte d)

- **Dicas**:
    - Dividir um número por 2 e quivale a deslocar seus bits uma posição à direita (shift right).
    - Para números negativos em complemento a 2, o deslocamento para a direita deve manter o bit de sinal (arithmetic shift right).
    - Compare o resultado com o valor decimal esperado após dividir por 2.