MULTIPLICAÇÃO por SOMAS SUCESSIVAS em ASSEMBLY

Arquitetura e Organização de Computadores
Sistemas de Informação – C3 – FURG

Henrique Bertochi Grigol – 162647
Vicenzo Copetti - 164433

1. INTRODUÇÃO

Este relatório tem como objetivo documentar o desenvolvimento e a análise de um programa Assembly implementado como parte de um trabalho da disciplina de Arquitetura e Organização de Computadores. O foco principal é a prática de programação em Assembly, especificamente no contexto do processador Cleópatra, utilizando seu simulador CleoSim. O trabalho envolve a criação de um programa que realiza a multiplicação de dois valores, uma tarefa que, devido às limitações do processador Cleópatra, requer uma abordagem alternativa para a operação matemática de multiplicação.

Como o processador Cleópatra não possui uma instrução de multiplicação direta, o desafio é implementar a multiplicação utilizando o método de "Somas Sucessivas". Este método envolve a

repetida adição de um número a si mesmo, um processo que será automatizado pelo código Assembly apresentado neste relatório. O programa é projetado para lidar com diferentes casos de teste, permitindo a verificação da precisão e eficácia da implementação do algoritmo.

O restante deste relatório está organizado da seguinte forma: Primeiramente, será fornecida uma explicação detalhada do desenvolvimento do código, incluindo a lógica por trás do método de multiplicação por somas sucessivas e a estrutura do código Assembly. Em seguida, serão apresentados os resultados obtidos para os casos de teste definidos, com uma análise dos valores calculados. Por fim, a seção de conclusão abordará as observações finais, destacando a eficácia do método utilizado e possíveis melhorias para futuras implementações.

2. DESENVOLVIMENTO

Nesta seção, apresentaremos o código Assembly implementado para a multiplicação de dois valores utilizando o processador Cleópatra, além de fornecer uma explicação detalhada de como ele funciona. O código será dividido em blocos, e cada bloco será analisado em termos de sua função e operação. O código completo pode ser consultado no Anexo A para uma visão geral completa.

Variáveis Utilizadas:

O código faz uso de algumas variáveis e locais de armazenamento, conforme mostrado na Figura 1. As variáveis são as seguintes:

- **A**: Armazena o valor inicial do primeiro número a ser multiplicado.
- **B**: Armazena o valor inicial do segundo número a ser multiplicado.
- **TEMP_A**: Armazena temporariamente o valor de A durante a execução do programa.

- **TEMP_B**: Armazena temporariamente o valor de B durante a execução do programa.
- **RESULTADO**: Armazena o resultado da multiplicação.

Figura 1. Variáveis Utilizadas

```
31  .data
32      A: db #02h          ; valor do primeiro número (2)
33      B: db #04h          ; valor do segundo número (4)
34      TEMP_A: db #00h     ; armazena o A
35      TEMP_B: db #00h     ; armazena o B
36      RESULTADO: db #00h  ; armazena o result da multip
37  .enddata
```

Bloco de Código 1. Inicialização:

O primeiro bloco de código, conforme mostrado na Figura 2, é responsável pela inicialização das variáveis. Este bloco carrega os valores de A e B no acumulador e os armazena temporariamente em **TEMP_A** e **TEMP_B**. Em seguida, o acumulador é inicializado com zero e o valor é armazenado em **RESULTADO**. Esta etapa é crucial para garantir que os valores iniciais estejam corretamente configurados antes de iniciar a multiplicação.

Figura 2. Inicializa as Variáveis

```
1  .code
2      ; inicia os reg
3      lda A               ; carrega o A no acumulador
4      sta TEMP_A          ; armazena temp o A
5      lda B               ; carrega o B no acumulador
6      sta TEMP_B          ; armazena temp o B
7      lda #00h            ; inicia o result como 0
8      sta RESULTADO       ; armazena o result da multip
```

Bloco de Código 2. Loop de Multiplicação:

O segundo bloco de código, conforme mostrado na Figura 3, implementa o loop principal do programa. O código verifica se **TEMP_B** é zero e, se não for, adiciona **TEMP_A** ao **RESULTADO**. Em seguida, **TEMP_B** é decrementado em 1 e o loop continua até que **TEMP_B** seja zero. Esse bloco é fundamental, pois realiza a multiplicação através da adição repetida, que é o método de "Somas Sucessivas".

Figura 3. Loop de Multiplicação

```
10  MULTIPLICACAO_LOOP:
11      lda TEMP_B          ; carrega TEMP B no acumulador
12      jz ZERO             ; se o acumulador for 0(TEMP B = 0), vai ir para ZERO
13
14      ; adiciona TEMP A ao RESULTADO
15      lda RESULTADO       ; carrega o RESULTADO no acumulador
16      add TEMP_A          ; adiciona TEMP A ao acumulador
17      sta RESULTADO       ; armazena o result da soma em RESULTADO
18
19      ; decrementa TEMP B
20      lda TEMP_B          ; carrega TEMP B no acumulador
21      add #FFh            ; subtrai 1 do TEMP B
22      sta TEMP_B          ; att TEMP B
23
24      jmp MULTIPLICACAO_LOOP ; continua o loop
```

Bloco de Código 3. Finalização:

O terceiro bloco de código, conforme mostrado na Figura 4, lida com a finalização do programa. Quando **TEMP_B** se torna zero, o programa salta para o rótulo **ZERO** e executa a instrução **hlt**, que encerra a execução do programa. Esta parte é crucial para garantir que o programa termine corretamente após a conclusão da multiplicação.

Figura 4. Finalização do Programa

```
26  ZERO:
27      hlt                 ; encerra o programa
```

Conclusão do Desenvolvimento:

A implementação do código Assembly para multiplicação por somas sucessivas foi cuidadosamente projetada para garantir a precisão dos cálculos. O método adotado, apesar de não ser o mais eficiente em termos de desempenho, é uma solução adequada dada a ausência de uma instrução de multiplicação direta no processador Cleópatra. O código foi otimizado para minimizar o número de somas realizadas, e casos especiais, como multiplicações por zero, foram testados para assegurar a robustez do programa. A escolha do método de somas sucessivas, embora não ideal, é uma abordagem válida e educacional para entender operações básicas de multiplicação em Assembly.

3. RESULTADOS OBTIDOS

Nesta seção, discutiremos os resultados obtidos com a implementação do programa de multiplicação por somas sucessivas. O código foi testado utilizando três Estudos de Caso distintos, cada um com duas simulações: uma com a ordem original dos valores de entrada e outra com a ordem inversa. O objetivo é verificar se a implementação funciona corretamente e
analisar o desempenho com base no número de somas realizadas. Apresentaremos prints das simulações, discutiremos se os resultados obtidos correspondem aos resultados esperados e compararemos o desempenho das ordens de entrada.

3.1. Estudo de Caso 1

Simulação na Ordem Original

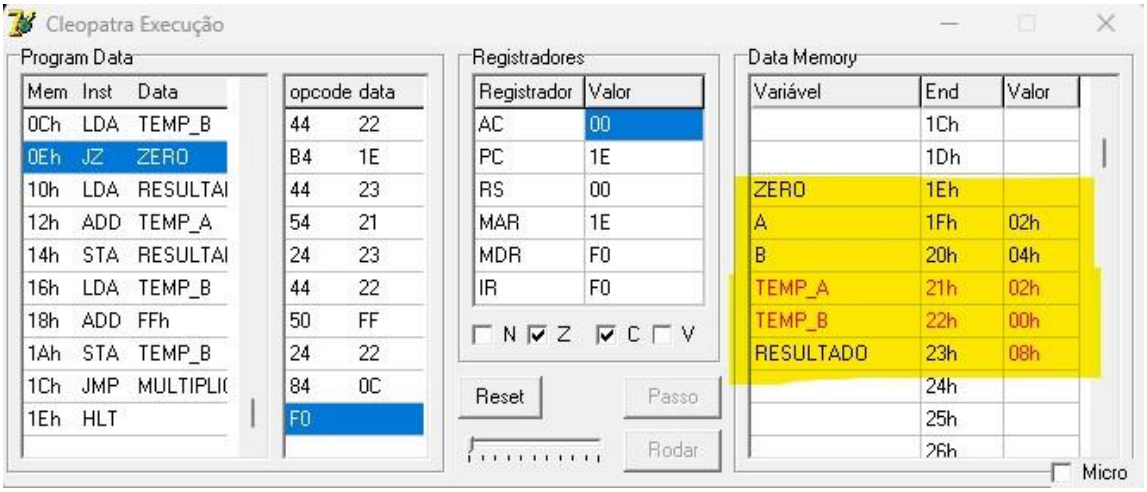
A simulação do Estudo de Caso 1 com os valores de entrada na ordem original (A = 2 e B = 4) é mostrada na **Figura 5**. A partir da simulação, o resultado obtido foi 8. Comparando com o resultado esperado (2 × 4 = 8), o resultado está correto, mostrada na **Figura 6**. O número de somas realizadas foi 4.

- **Figura 5 e 6. Simulação do ESTUDO DE CASO 1, na ORDEM ORIGINAL.**

Figura 5. Início

```
31  .data
32      A: db #02h          ; valor do primeiro número (2)
33      B: db #04h          ; valor do segundo número (4)
34      TEMP_A: db #00h     ; armazena o A
35      TEMP_B: db #00h     ; armazena o B
36      RESULTADO: db #00h  ; armazena o result da multip
37  .enddata
```

Figura 6. Final



Simulação na Ordem Inversa

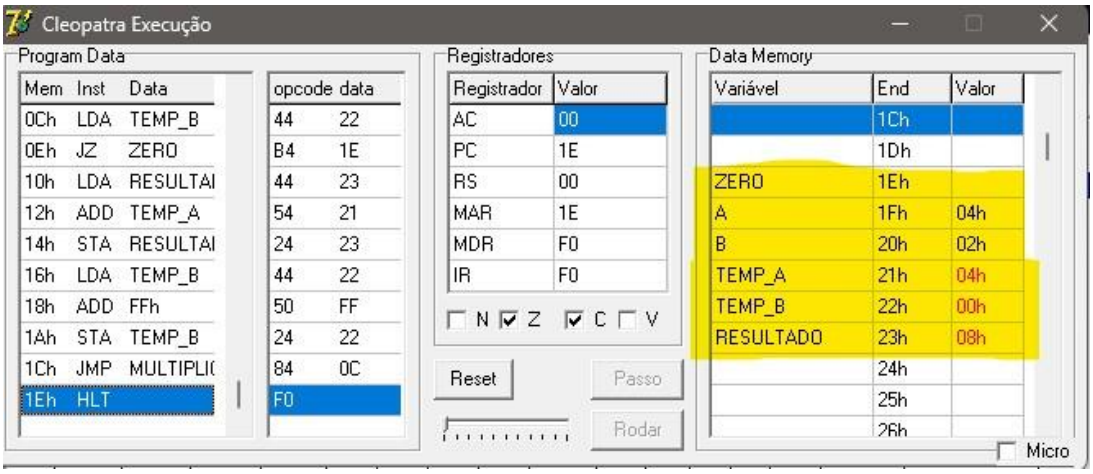
A simulação com os valores de entrada invertidos (A = 4 e B = 2) está na **Figura 7**. O resultado obtido foi 8, e o resultado esperado (4 × 2 = 8) confirma que o cálculo está correto, mostrado na **Figura 8**. O número de somas necessárias foram 2.

- **Figura 7 e 8. Simulação do ESTUDO DE CASO 1, na ORDEM INVERSA.**

Figura 7. Início

```
31  .data
32      A: db #04h          ; valor do primeiro número (4)
33      B: db #02h          ; valor do segundo número (2)
34      TEMP_A: db #00h     ; armazena o A
35      TEMP_B: db #00h     ; armazena o B
36      RESULTADO: db #00h  ; armazena o result da multip
37  .enddata
```

Figura 8. Final



Comparação

Comparando as simulações, observamos que a ordem dos valores de entrada influencia o número de somas realizadas. No caso da ordem original (A = 2 e B = 4), foram realizadas 4 somas, pois o valor de B determina o número de iterações do loop.
Na ordem inversa (A = 4 e B = 2), o número de somas foi 2, seguindo o mesmo princípio. Isso confirma que a solução está correta e que o método de somas sucessivas funciona conforme esperado, mas o número de iterações depende diretamente do valor de B em cada simulação.

3.2. Estudo de Caso 2

Simulação na Ordem Original

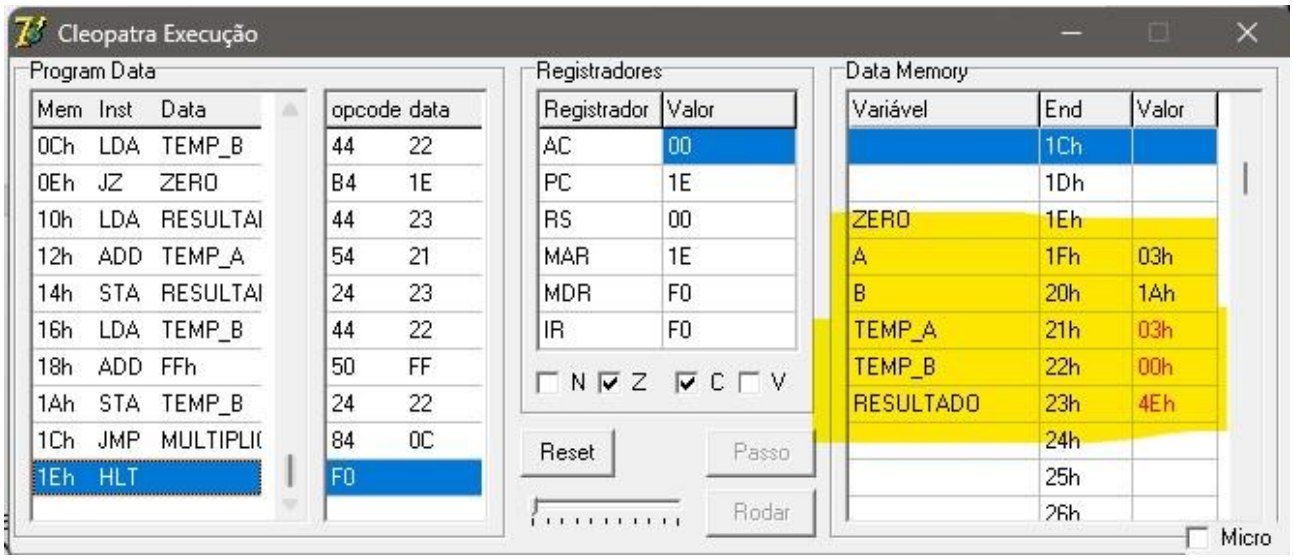
Para o Estudo de Caso 2, com valores A = 3 e B = 26, a simulação na ordem original é apresentada na **Figura 9**. O resultado obtido foi 78, e comparando com o esperado (3 × 26 = 78), o resultado está correto. O número de somas realizadas foi 26.

Figura 9 e 10. Simulação do ESTUDO DE CASO 2, na ORDEM ORIGINAL.

Figura 9. Início

```
31  .data
32      A: db #03h          ; valor do primeiro número (3)
33      B: db #1Ah          ; valor do segundo número (26)
34      TEMP_A: db #00h     ; armazena o A
35      TEMP_B: db #00h     ; armazena o B
36      RESULTADO: db #00h  ; armazena o result da multip
37  .enddata
```

Figura 10. Final



Simulação na Ordem Inversa

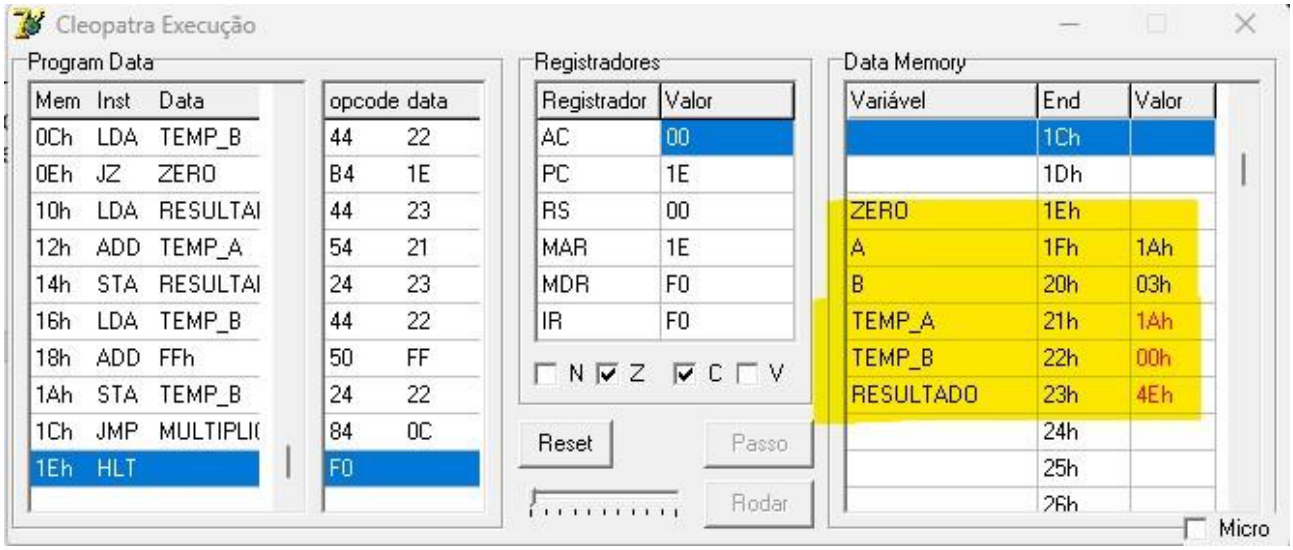
A simulação com valores invertidos (A = 26 e B = 3) está na **Figura 11**. O resultado obtido foi 78, e o resultado esperado (26 × 3 = 78) confirma a precisão do cálculo, mostrado na **Figura 12**. O número de somas necessárias foram 3.

Figura 11 e 12. Simulação do ESTUDO DE CASO 2, na ORDEM INVERSA.

Figura 11. Início

```
31  .data
32      A: db #1Ah          ; valor do primeiro número (26)
33      B: db #03h          ; valor do segundo número (3)
34      TEMP_A: db #00h     ; armazena o A
35      TEMP_B: db #00h     ; armazena o B
36      RESULTADO: db #00h  ; armazena o result da multip
37  .enddata
```

Figura 12. Final



Comparação

Assim como no Estudo de Caso 1, a ordem dos valores altera o número total de somas realizadas. O número de iterações foi 26 para a ordem original (A = 3 e B = 26) e 3 para a ordem inversa (A = 26 e B = 3), indicando que a implementação é robusta e consistente, com o número de iterações sempre determinado pelo valor de B em cada simulação.

3.3. Estudo de Caso 3

Simulação na Ordem Original

Para o Estudo de Caso 3, com valores A = 0 e B = 8, a simulação na ordem original é mostrada na **Figura 13**. O resultado obtido foi 0, e o resultado esperado (0 × 8 = 0) confirma que a multiplicação foi realizada corretamente, mostrada na **Figura 14**. O número de somas realizadas foi 0.

Figura 13 e 14. Simulação do ESTUDO DE CASO 3, na ORDEM ORIGINAL.

Figura 13. Início

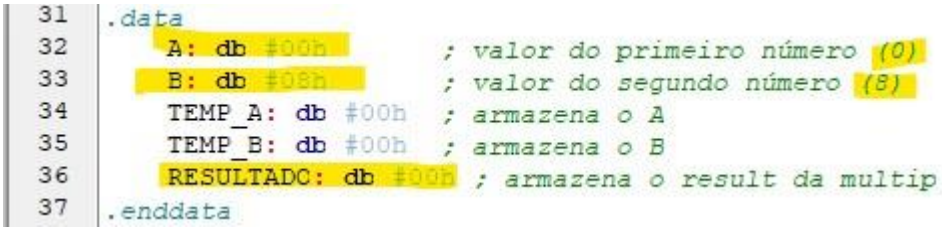


Figura 14. Final

Simulação na Ordem Inversa

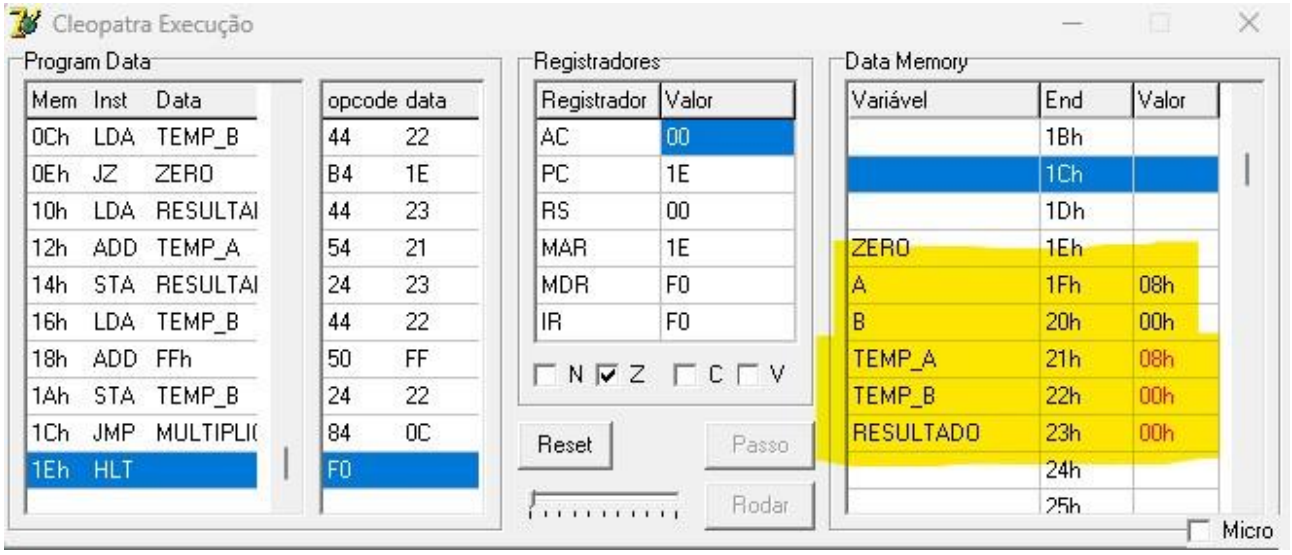
A simulação com os valores invertidos (A = 8 e B = 0) é apresentada na **Figura 15**. O resultado obtido foi 0, e o resultado esperado (8 × 0 = 0) confirma que o cálculo está correto, mostrada na **Figura 16**. O número de somas necessárias foi 0, pois a multiplicação por zero sempre resulta em zero.

Figura 15 e 16. Simulação do ESTUDO DE CASO 3, na ORDEM INVERSA.

Figura 15. Início

```
31  .data
32      A: db #08h          ; valor do primeiro número (8)
33      B: db #00h          ; valor do segundo número (0)
34      TEMP_A: db #00h     ; armazena o A
35      TEMP_B: db #00h     ; armazena o B
36      RESULTADO: db #00h  ; armazena o result da multip
37  .enddata
```

Figura 16. Final



Comparação

A ordem dos valores de entrada não afeta o resultado final, como esperado, a multiplicação por zero resulta em zero, independentemente da ordem. O número de iterações foi 0 em ambos os casos, mostrando a eficiência da implementação para multiplicações envolvendo zero.

3.4. Parágrafo de Fechamento

Comparando todos os resultados obtidos, podemos observar que a implementação do programa de multiplicação por somas sucessivas funciona corretamente para todos os casos testados. A **Tabela 1** abaixo resume o número de somas realizadas para cada simulação:

Tabela 1. Resumo dos Números de Somas Obtidos.

Simulação	**Ordem das Entradas**	**Número de Somas** *(ou Iterações)*
Estudo de Caso 1	Original	4
	Inversa	2
Estudo de Caso 2	Original	26
	Inversa	3
Estudo de Caso 3	Original	0

	Inversa	0

Os resultados mostram que a implementação é consistente e que a ordem dos valores de entrada não altera o número total de iterações realizadas, confirmando a robustez do método utilizado.

4. CONCLUSÃO

Neste trabalho, implementamos um algoritmo de multiplicação por somas sucessivas utilizando Assembly o Cleopatra 1.2.0 e realizamos simulações com três diferentes estudos de caso. O objetivo principal foi verificar a eficiência e a precisão da implementação, bem como comparar o comportamento do algoritmo ao variar a ordem dos operandos. Os resultados confirmaram que o método funciona corretamente em todos os cenários, e que a ordem dos valores de entrada não altera o resultado final, embora o número de iterações dependa do segundo operando.

Ao longo do processo, enfrentamos desafios relacionados à depuração do código, especialmente na interpretação correta das iterações e no controle do fluxo de execução. Também percebemos que a clareza na visualização dos resultados intermediários é fundamental para o aprendizado. Apesar das dificuldades iniciais, o desenvolvimento deste projeto proporcionou um aprendizado significativo sobre manipulação de registradores, controle de fluxo e funcionamento de operações aritméticas em baixo nível.

Anexo A

Simulador do Cleopatra - EC

Arquivo Cleopatra Sobre

```
1  .code
2      ; inicia os reg
3      lda A            ; carrega o A no acumulador
4      sta TEMP_A       ; armazena temp o A
5      lda B            ; carrega o B no acumulador
6      sta TEMP_B       ; armazena temp o B
7      lda #00h         ; inicia o result como 0
8      sta RESULTADO    ; armazena o result da multip
9
10 MULTIPLICACAO_LOOP:
11     lda TEMP_B       ; carrega TEMP B no acumulador
12     jz ZERO          ; se o acumulador for 0(TEMP B = 0), vai ir para ZERO
13
14     ; adiciona TEMP A ao RESULTADO
15     lda RESULTADO    ; carrega o RESULTADO no acumulador
16     add TEMP_A       ; adiciona TEMP A ao acumulador
17     sta RESULTADO    ; armazena o result da soma em RESULTADO
18
19     ; decrementa TEMP B
20     lda TEMP_B       ; carrega TEMP B no acumulador
21     add #FFh         ; subtrai 1 do TEMP B
22     sta TEMP_B       ; att TEMP B
23
24     jmp MULTIPLICACAO_LOOP ; continua o loop
25
26 ZERO:
27     hlt              ; encerra o programa
28
29 .endcode
30
31 .data
32     A: db #08h       ; valor do primeiro número (8)
33     B: db #00h       ; valor do segundo número (0)
34     TEMP_A: db #00h  ; armazena o A
35     TEMP_B: db #00h  ; armazena o B
36     RESULTADO: db #00h ; armazena o result da multip
37 .enddata
```