Memorial descritivo - tarefa 1

Nessa tarefa, implementaremos dois algoritmos de exponenciação distintos; um deles tem sua complexidade O(n) e o outro O(log n).

Abordaremos a forma em que os algoritmos foram implementados para Mips Assembly:

Em .data, definimos a variável result inicializada como 1.

No fragmento 1, definimos o Label **expo1** (equivalente a função do código em C) e dois label's sendo eles loop e **saiDoLoop**. Esse último, é acionado quando a verificação do loop falha, retornando assim o resultado da operação.

No fragmento 2, definimos a **expo2,** os mesmo dois label's do fragmento anterior (**loop** e **saiDoLoop**), além de uma função auxiliar, a **multpX,** que realiza a multiplicação de x por ele mesmo e guarda o resultado no próprio x.

Funcionamento:

Fragmento 1:

Esse algoritmo se comporta da seguinte forma:

1. Primeiro, é verificado se o expoente é menor ou igual a zero. Se sim, então já é retornado o valor 1, sem nem mesmo entrar no laço.
2. Caso o expoente seja maior do que zero, então multiplicamos o resultado pela base e decrementamos o expoente. Em seguida, é feita a verificação novamente.
3. Ao final, é retornado o valor resultante.

Fragmento 2:

1. Assim como no fragmento anterior, é verificado se o expoente é menor ou igual a zero; caso seja, é retornado o resultado final.
2. Caso contrário, é verificado se o expoente é ímpar. Em caso afirmativo, o resultado final é multiplicado pela base; em seguida, executa-se o passo 3. Em caso falso, é efetuado o passo 3 diretamente.
3. Após isso, o expoente é dividido por 2 e o seu quociente é o novo expoente. Ademais, a nova base é agora o quadrado da base original. Feito isso, voltamos ao primeiro passo até que o expoente seja 0.

Limitações:

Ressaltamos que as limitações de uso para os algoritmos é a própria capacidade da arquitetura referente à quantidade máxima de dígitos armazenada.

Além disso, vale salientar que para expoentes negativos, o resultado seria um número com ponto flutuante, o que foge do escopo da tarefa. Dessa forma, caso o expoente seja negativo, o

resultado será 1 por padrão – o que também acontece caso o expoente seja 0 -, para evitar cálculos de ponto flutuante.

Análise de complexidade:

Fragmento 1:

Em nosso primeiro fragmento, podemos observar que a complexidade do algoritmo é de O(n), pois a quantidade de iterações do laço de repetição é equivalente ao tamanho do n.

Suponha que queiramos executar a seguinte operação: 3^9

O fragmento realizará 3 * 3 nove vezes.

1. 3 * 1 = 3
2. 3 * 3 = 9
3. 3 * 9 = 27
4. 3 * 27 = 81
5. 3* 81 = 243
6. 2 * 243 = 729
7. 3 * 729 = 2.187
8. 3 * 2.187 = 6.561
9. 3 * 6.561 = 19.683

Fragmento 2:

Nesse algoritmo, com complexidade O(log n), essa mesma operação seria realizada em somente 4 passos.

1. 9 é ímpar, então multiplicaríamos o resultado final (1) pela base, resultando em 3. Divide-se 9 por 2, que é 4 (quociente); multiplica-se a base por ela mesma 3 * 3 = 9
2. 4 não é ímpar, dividiríamos ele por 2, resultando em 2, e multiplicaríamos a base por ela mesma (9 * 9 = 81)
3. 2 não é ímpar, divide-se por 2 (1), e multiplica-se a base (81) por ela mesma, resultando em 6.561
4. 1 é ímpar então realiza-se a multiplicação de resultado final pela base (3 * 6.561 = 19.683).

Obtemos o mesmo resultado que o fragmento 1, entretanto, com menos da metade de passos.