Relatório do primeiro trabalho

Nome: Lívia Lutz dos Santos
Matrícula: 2211055
Turma: 3WB

Para os testes feitos com os valores considerados, o programa funciona bem, sem erros aparentes.

Estratégia de testes:

Utilizei a metodologia TDD como inspiração para montar minha rotina dos testes de cada função.

Primeiro, eu escrevia as funções responsáveis por testar cada função, considerando os valores mais adequados para cada caso de teste e então escrevia o código da função a ser testada. Após isso, eu executava o programa de teste para verificar se algum teste daquela função falhava, se houvessem falhas, o programa de teste exibe onde elas aconteceram,o que facilitava o meu entendimento do possível erro na função ou na função de teste e então eu começava a refatorar o código testado (tanto a função de teste quanto a função testada) e repetia o processo até que todos os testes funcionassem da maneira esperada.

Criei também uma função de base que comparava 2 arrays BigInt para facilitar a verificação do resultado retornado pela função sendo testada com o resultado esperado dentro das funções de teste.

```c
/* Função para testar se a resposta obtida é igual a resposta esperada para cada teste */

int testa_certo(BigInt a, BigInt b, char *msg, int test) {

   /* Variável para contar erros */

   int erro = 0;

   /* Percorre os arrays BigInt comparando cada byte */

   for (int i = 0; i < (NUM_BITS/8); i++){

      /* Se os bytes foram diferentes, aponta o erro e define que erro = 1 */

      if (a[i] != b[i]){
         printf("Erro: %02x no byte %d\n",a[i],i);
         erro = 1;
      }

   }

   /* Se teve erro, exibe mensagem indicando onde está o erro */

   if (erro == 1){
      printf("Erro em %s no teste %d\n", msg, test);
   }

   /* Retorna erro */

   return erro;
}
```

Usei a função “big_val” dentro de cada função de teste para gerar o array BigInt de cada valor usado para ser testado e para ser verificado pelas funções aritméticas e de deslocamento como no exemplo da função big_comp2:

```
int teste_da_big_comp2(char*msg) {

   /* Variável para contar erros nos testes */

   int erros = 0;

   /* Variáveis para guardar os resultados gerados por cada chamada de função para número positivo */

   BigInt res,res1;

   /* Testando o complemento a 2 de um número positivo (5) */

   /* Estratégia usada: Estendi o valor que usarei para fazer o teste e o valor do resultado esperado de um long para BigInt para
     depois comparar os arrays dos 2 números estendidos, aplicando essa estratégia em todo programa de teste */

   big_val(res, 5);
   big_val(res1, -5);

   big_comp2(res, res);

   if (testa_certo(res, res1, msg, 1) == 1){
      erros += 1;
   }

   /* Testando o complemento a 2 de um número negativo (-7) */

   BigInt res2,res3;

   big_val(res2,-7);

   big_val(res3,7);

   big_comp2(res2, res2);

   if (testa_certo(res2, res3, msg, 2) == 1){
      erros += 1;
```

Casos de teste :

Para cada função, considerei usar valores extremos, os quais poderiam potencialmente gerar erros nos testes de cada uma delas.

Para a função de atribuição big_val, usei números positivos, negativos, zero e muito grandes para verificar se a função estendia corretamente considerando o sinal do número e seu tamanho.

Nas funções de atribuição, usei os mesmos casos de teste da função de atribuição, porém testei as operações entre números de sinais diferentes para testar se o sinal do resultado estava correto, com números muito grandes para considerar overflow e com 0 para testar se o número era alterado na soma e na subtração ou se o resultado seria 0 na multiplicação. Para a função big_comp2 testei números positivos e negativos para verificar se seu complemento a 2 tinha sinal oposto ao número original.

Por fim, nas funções de deslocamento, usei números positivos e negativos para verificar se os bits estavam sendo preenchidos corretamente com 0 ou com o bit mais significativo, usei também valores de shift iguais a 0, que é um caso extremo das funções de deslocamento para verificar se o número seria alterado, múltiplos de 8, para verificar se os shifts de cada byte inteiro estavam corretos ,127, já que é um caso extremo da função para verificar se apenas o

último bit do array permanecia e não múltiplos de 8, para verificar se a função deslocava corretamente a quantidade pedida no parâmetro

Arquivo de teste:

O arquivo de teste usado contém as funções descritas acima com uma função main que conta quantos erros foram obtidos à medida que cada função de teste é chamada,exibindo no final do programa quantos e quais testes obtivemos o resultado esperado ou não da quantidade de testes realizados.

```c
int main() {

    /* Variável para contar erros nos testes */
    int erros = 0;

    /* Para cada chamada de cada função de teste, somamos o retorno a erros */

    erros += teste_da_big_val("big_val");
    erros += teste_da_big_comp2("big_comp2");
    erros += teste_da_big_sum("big_sum");
    erros += teste_da_big_sub("big_sub");
    erros += teste_da_big_mul("big_mul");
    erros += teste_da_big_shl("big_shl");
    erros += teste_da_big_shr("big_shr");
    erros += teste_da_big_sar("big_sar");

    /* Exibimos quantos erros tivemos em comparacao com a quantidade de testes feitos */

    printf("Total de falhas %d de 45\n", erros);
    return 0;
}
```