Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Programação de Sistemas Computacionais

Verão de 2020/2021

Série de Exercícios 2

Nos exercícios seguintes é proposta a escrita de funções em *assembly* para a arquitetura x86-64, usando a variante de sintaxe AT&T, e seguindo os princípios básicos de geração de código do compilador de C da GNU. A resolução de cada exercício (código *assembly*) deve ser apresentado em conjunto com o respectivo programa de teste escrito em linguagem C. Tenha em consideração que os exercícios que não forem demonstrados a funcionar serão considerados como não tendo sido realizados. Não se esqueça de testar devidamente o código desenvolvido, **invocando as funções escritas em *assembly* com parametrizações diferentes**, bem como de o apresentar de forma cuidada, apropriadamente indentado e comentado. Não é necessário relatório. Contacte o docente se tiver dúvidas. Encoraja-se também a discussão de problemas e soluções com colegas, mas salienta-se que a partilha direta de soluções leva, no mínimo, à anulação das entregas de todos os estudantes envolvidos.

1. Programe em *assembly* a função `bitmap_rotate_right` que roda 90º a figura indicada pelo parâmetro `bitmap_src` e deposita a figura rodada no local indicado por `bitmap_dst`. A figura é constituída por uma matriz quadrada de píxeis monocromáticos de dimensão 8. É representada sobre um *array* de 8 palavras de 8 *bits* em que cada palavra armazena os píxeis de uma linha da figura.

   ```
   void bitmap_rotate_right ( uint8_t *bitmap_dst, uint8_t *bitmap_src );
   ```

2. Programe em *assembly* a função `my_memset` segundo a definição da função `memset` na biblioteca normalizada da linguagem C. Esta função preenche a zona de memória definida pelo ponteiro `ptr` e dimensão `num`, com o valor passado em `value`. Apesar deste parâmetro ser do tipo `int`, a função trata-o como se fosse do tipo `unsigned char`. Procure minimizar o número de acessos à memória efetuando acessos alinhados a palavras com múltiplos *bytes*.

   ```
   void *my_memset ( void *ptr, int value, size_t num );
   ```

3. Considere a função `compare_data_value`, cuja definição em linguagem C se apresenta a seguir. Implemente esta função em *assembly* em duas versões, uma para cada definição da *struct* `Dataset`.

   a) `typedef struct { const int *id; unsigned length; Data **data; } Dataset;`

   b) `typedef struct { const int *id; unsigned length; Data *data[]; } Dataset;`

   ```
   typedef struct { char label[7]; short value; } Data;


   int compare_data_value ( Dataset *set1, Dataset *set2, unsigned index ) {
       if (index >= set1->length || index >= set2->length ||
              NULL != set1->data[index] || NULL != set2->data[index])
              return 0;
       return set1->data[index]->value == set2->data[index]->value;
   }
   ```

4. Apresenta-se abaixo uma implementação do algoritmo para pesquisa binária segundo a definição da função `bsearch` na biblioteca normalizada da linguagem C. Implemente a função `binary_search` em linguagem *assembly*.

```
void *binary_search ( const void *key, const void *base, size_t nel, size width,
                int (*compar) ( const void *, const void * ) ) {

    size_t low = 0, high = nel - 1;
    while (low <= high) {
          size_t mid = (low + high) / 2;
          char *midp = (char *)base + mid * width;
          int cond;
          if ((cond = (*compar)(midp, key)) < 0)
                low = mid + 1;
          else if (cond > 0) high = mid - 1;
                else return midp;
    }
    return NULL;
}
```

Data recomendada para conclusão: 21 de Maio de 2021

ISEL, 27 de Abril de 2021