Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Programação de Sistemas Computacionais

Inverno de 2024/2025

Trabalho prático 1

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.

Deve submeter a sua realização de cada exercício aos testes anexos a este enunciado. As respetivas instruções de utilização estão incluídas no próprio pacote de testes.

Tenha o cuidado de apresentar o código de forma cuidada, apropriadamente indentado e comentado (siga as recomendações em anexo). Não é necessário relatório.

Encoraja-se a discussão de problemas e soluções com colegas. Tenha consciência que os exercícios só são benéficos na aprendizagem se forem realizados com honestidade académica. Contacte o docente se tiver dúvidas.

1. Escreva em *assembly* a função **`rotate_right`** que desloca para a direita (no sentido de maior peso para o de menor peso) o valor a 128 *bit*, que recebe no parâmetro **`value`**, o número de posições indicadas no parâmetro **`n`**. O valor numérico de 128 *bit* é formado pela concatenação de dois valores a 64 *bit* armazenados num *array* com duas posições, segundo o formato *little-endian*. Os *bits* que saem da posição de menor peso entram, pela mesma ordem, na posição de maior peso. Utilize a instrução *assembly* **shrd**.

   ```
   void rotate_right(unsigned long value[], size_t n);
   ```

2. Escreva em *assembly* a função **`my_memcmp`** segundo a definição de **`memcmp`** tal como está definida na biblioteca *standard* da linguagem C. Esta função compara os conteúdos de memória referenciados pelos parâmetros **`ptr1`** e **`ptr2`**. Na programação, procure minimizar o número de acessos à memória realizando, sempre que possível, acessos a palavras de 64 *bits* em endereços alinhados (i.e., endereços múltiplos de 8) e minimizar o número de iterações, mesmo que para isso tenha que aumentar a memória ocupada por código.

   ```
   int my_memcmp( const void *ptr1, const void *ptr2, size_t num );
   ```

3. Considere a função **`get_val_ptr`**, cuja definição em linguagem C se apresenta abaixo.

   ```
   struct data { short flags:6; short length:10; short vals[]; };

   struct info { double ref; struct data **data; int valid; };

   short *get_val_ptr(struct info items[],
                      size_t item_idx, size_t data_idx, size_t val_idx, short mask)
   {
       return items[item_idx].valid
              && val_idx < items[item_idx].data[data_idx]->length
              && (items[item_idx].data[data_idx]->flags & mask)
                     ? &items[item_idx].data[data_idx]->vals[val_idx]
                     : NULL;
   }
   ```

   a. Implemente a função **`get_val_ptr`** em *assembly* x86-64.

b. Escreva um programa de teste em linguagem C, que defina uma estrutura de dados estática, invoque a função escrita em *assembly* e verifique se o valor retornado é correto.

4. Considere a função **array_remove_cond**, cuja definição em linguagem C se apresenta a seguir.

```
size_t array_remove_cond(void **array, size_t size,
                    int (*eval)(const void *, const void *), void *context)
{
    for (void **current = array, **last = array + size; current < last; ) {
          if (eval(*current, context)) {
                memmove(current, current + 1, (last - current - 1) * sizeof(void *));
                size -= 1;
                last -= 1;
          }
          else {
                current += 1;
          }
    }
    return size;
}
```

a. Implemente a função **array_remove_cond** em *assembly* x86-64.

b. Escreva em linguagem C, um programa de teste da função **array_remove_cond**. Este programa deve remover de um *array* de ponteiros para **struct student**, os ponteiros que correspondam a estudantes com número superior ao dado como argumento do executável. No programa, deve constar a definição estática do *array* de ponteiros e das instâncias apontadas, a definição da função de verificação e a chamada à função **array_remove_cond**.

```
struct student {
      const char *name;
      int number;
};
```

Data para conclusão: 10 de Novembro de 2024

ISEL, 7 de Outubro de 2024

Anexo

Recomendações para a escrita de programas em *Assembly*.

- O texto do programa é escrito em letra minúscula, exceto os identificadores de constantes.
- Nos identificadores formados por várias palavras usa-se como separador o carácter '_' (sublinhado).
- O programa é disposto na forma de uma tabela de quatro colunas. Na primeira coluna insere-se apenas a *label* (se existir), na segunda coluna a mnemónica da instrução ou a diretiva, na terceira coluna os parâmetros da instrução ou da diretiva e na quarta coluna os comentários até ao fim da linha (começados por '**#**' ou envolvidos por ***/* */***).
- Cada linha contém apenas uma *label*, uma instrução ou uma diretiva.
- Para definir as colunas deve usar-se o carácter TAB configurado com a largura de oito espaços.
- As linhas com *label* não devem conter nenhum outro elemento. Isso permite usar *labels* compridas sem desalinhar a tabulação e criar separações na sequência de instruções, que ajudam na interpretação do programa.