Programação em Linguagem *Assembly*

Exercícios para resolver nas aulas teórico-práticas

Arquitetura de Computadores

Departamento de Engenharia Eletrónica e de
Telecomunicações e Computadores

4 de Março de 2023

Introdução

Este documento apresenta os exercícios fundamentais a resolver nas aulas teórico-práticas de Arquitetura de Computadores (AC) nos três módulos de 1,5h destinados à apresentação dos tópicos sobre programação estruturada em linguagem *assembly*. Considera-se como caso de estudo o processador P16 e a sua linguagem *assembly*.

Objetivos

O conjunto de exercícios considerados neste documento tem como principal objetivo uniformizar a apresentação, aos alunos que frequentam as aulas das várias turmas de AC, das principais técnicas e metodologias que devem ser utilizadas no desenvolvimento de programas escritos em linguagem *assembly*, incluindo a estruturação dos programas em rotinas e a forma de organização dos programas em memória.

Assim, estes exercícios envolvem *i*) a operação de números inteiros, com e sem sinal, e de caracteres, *ii*) a manipulação de *arrays* em memória, *iii*) a invocação e retorno de rotinas, *iv*) a passagem de argumentos e a devolução de valores de rotinas, *v*) a exploração das regras da convenção P16 e *vi*) a utilização de variáveis em memória.

Os algoritmos subjacentes são propositadamente simples e, em princípio, já do conhecimento dos alunos, por forma a que o esforço de aprendizagem esteja concentrado nos objetivos enunciados.

Tipos

Na especificação dos exercícios consideram-se os tipos definidos na biblioteca C, porquanto os tipos numéricos apresentados são representados na base 2 e têm os seguintes significados:

`int8_t` – inteiro a 8 bits, com sinal	`uint8_t` – inteiro a 8 bits, sem sinal
`int16_t` – inteiro a 16 bits, com sinal	`uint16_t` – inteiro a 16 bits, sem sinal
`int32_t` – inteiro a 32 bits, com sinal	`uint32_t` – inteiro a 32 bits, sem sinal

O tipo de dados `char` é utilizado para representar caracteres segundo o padrão *American Standard Code for Information Interchange* (ASCII), ocupando cada carácter 8 bits com valores possíveis entre 0 e 127.

Considera-se ainda o tipo de dados composto *array* da linguagem C para representar coleções de um ou mais valores, todos do mesmo tipo, armazenados em posições de memória contíguas e acessíveis por um único nome.

Uma *string* é um caso particular do tipo de dados composto *array*, em que os seus elementos são do tipo `char` e o último elemento da coleção é o carácter '\0'.

Lista de Exercícios

Aula 8

1. Implementar a função `abs`, que devolve o valor absoluto do argumento do parâmetro `x`.

```
int16_t abs( int16_t x ) {

     if( x < 0 ) {
          x = -x;
     }
     return x;
}
```

2. Implementar a função `min`, que devolve o argumento do parâmetro com o menor valor.

```
uint16_t min( uint16_t a, uint16_t b ) {

     return ( ( a < b ) ? a : b );
}
```

3. Com as adaptações necessárias, repetir o exercício 2 considerando que o tipo do parâmetro é `int16_t`.

4. Implementar a função `add32`, que calcula e devolve o valor da soma dos argumentos dos parâmetros `a` e `b`.

```
uint32_t add32( uint32_t a, uint32_t b ) {

     return ( a + b );
}
```

Aula 9

5. Implementar a função `max`, que devolve o maior valor presente no *array* `a`, com `n` elementos. A constante `UINT16_MIN` corresponde ao menor valor possível de codificar numa variável com tipo `uint16_t`.

```
#define UINT16_MIN 0

uint16_t max( uint16_t a[], uint16_t n ) {

     uint16_t max = UINT16_MIN;

     for( uint16_t i = 0; i < n; i++ ) {
          if( a[i] > max ) {
               max = a[i];
          }
     }
     return max;
}
```

6. Com as adaptações necessárias, repetir o exercício 5 considerando que os elementos do *array* são do tipo `int16_t`.

7. Implementar a função `to_upper`, que devolve a maiúscula correspondente ao carácter do tipo letra passado à função no parâmetro `ch`. Caso o argumento de `ch` não corresponda a um carácter do tipo letra, a função devolve esse valor inalterado.

```
char to_upper( char c ) {

      char uc = c;

      if( c >= 'a' && c <= 'z' ) {
            uc = c + ( 'A' - 'a' );
      }
      return uc;
}
```

8. Implementar a função `str_upper`, que altera a *string* `str` de modo a que todos os seus caracteres do tipo letra sejam maiúsculas. A função devolve o número total de caracteres que compõem a *string*.

```
uint16_t str_upper( char str[] ) {

      uint16_t i;

      for( i = 0; str[i] != '\0'; i++ ) {
            str[i] = to_upper( stri[i] );
      }
      return i;
}
```

Aula 10

9. Implementar a função `str_copy`, que copia todos os caracteres que compõem a *string* `src` para a *string* `dst`.

```
void str_copy( char dst[], char src[] ) {

      uint16_t i = 0;

      while( src[i] != '\0' ) {
            dst[i] = src[i];
            i++;
      }
      dst[i] = '\0';
}
```

10. Implementar o programa de teste para a função `str_copy`.

```
#define MAX_PHRASE_SIZE 100

uint8_t phrase_orig[] = "the quick brown fox jumps over the lazy dog";

uint8_t phrase_copy[MAX_PHRASE_SIZE];

void main( void ) {

     str_copy( phrase_copy, phrase_orig );
}
```

11. Implementar a função `selection_sort`, que ordena os `n` elementos do *array* `a` por ordem crescente dos seus valores.

```
void selection_sort( int16_t a[], uint16_t n ) {

     uint16_t i, j;
     int16_t temp;

     if ( n != 0 ) {
          for( i = 0; i < n-1; i++ ) {
               j = min_idx( a, i, n );
               temp = a[i];
               a[i] = a[j];
               a[j] = temp;
          }
     }
}
```

A função `min_idx` devolve o índice do *array* `a` correspondente ao menor valor presente entre os índices `f` e `l`.

```
uint16_t min_idx( int16_t a[], uint16_t f, uint16_t l ) {

     uint16_t i = f;

     for( uint16_t j = f+1; j < l; j++ ) {
          if( a[j] < a[i] ) {
               i = j;
          }
     }
     return i;
}
```

12. Implementar o programa de teste para a função `selection_sort`.

```
int16_t array[] = {-2, 45, 0, 11, -9};

void main( void ) {

     selection_sort( array, 5 );
}
```