# INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Engenharia de Eletrónica e Telecomunicações e de Computadores

Engenharia de Informática, Redes e Telecomunicações

Engenharia Informática e de Computadores


2.º Trabalho Prático

de

Arquitetura de Computadores

## Introdução à programação em *assembly*

7 de abril de 2017

## 1. Objetivos

Este trabalho tem como principais objetivos o exercício da programação em linguagem *assembly*, incluindo a organização dos programas em funções[1] e a introdução a um ambiente de programação em *assembly*.

## 2. Descrição do Trabalho

O trabalho consiste na implementação e teste de programas em *assembly* envolvendo *i*) operação de números inteiros, *ii*) manipulação de *arrays* em memória e *iii*) invocação de funções.

Os programas devem ser escritos em linguagem *assembly* do PDS16, podendo o seu teste ser realizado recorrendo ao simulador do PDS16 ou sobre o sistema SDP16.

Para cada um dos exercícios propostos, deve ser escrito um programa de teste que permita verificar e demonstrar o comportamento da função realizada, em diversos cenários de utilização.

## 3. Convenções

Na escrita dos programas, deve ter em conta as convenções seguintes, relativas a tipos, parâmetros, retorno de valores, preservação de registos e alojamento de memória.

### 3.1. Tipos

Na especificação dos exercícios, os tipos numéricos definidos têm os significados seguintes:

| | | | |
|---|---|---|---|
| **`int8`** | - inteiro com sinal a 8 *bits* | **`uint8`** | - inteiro sem sinal a 8 *bits* |
| **`int16`** | - inteiro com sinal a 16 *bits* | **`uint16`** | - inteiro sem sinal a 16 *bits* |

### 3.2. Parâmetros

Os parâmetros das funções são passados nos registos do processador, ocupando a quantidade necessária, por ordem: 1.º, r0; 2.º, r1; 3.º, r2; etc..

### 3.3. Valor de retorno

O valor de retorno de uma função, caso exista, é devolvido no registo r0.

### 3.4. Preservação de registos

Como consequência da chamada a uma função, podem ser modificados os registos seguintes:

- r5 (*link*) e r6 (*PSW*);
- Os registos usados para passar os parâmetros à função chamada.

Os registos restantes devem ser preservados. Se a função os utilizar, deve armazená-los em memória e, no final, repor os respetivos conteúdos anteriores.

### 3.5. Alojamento de memória para variáveis das funções ou preservação de registos

O espaço de memória para variáveis das funções ou preservação de registos deve:

- Ser reservado na área acessível por *load* e *store* com endereçamento direto;
- Ser identificado por um símbolo (*label*), com um prefixo que indique a respetiva função utilizadora.

1 - Por consistência com o código apresentado, a palavra "função" é utilizada com o mesmo significado que na linguagem C – como uma rotina que pode ou não ter valor associado.

## 4. Especificação dos exercícios

4.1. Implemente, em *assembly* do PDS16, a função `is_leap` que verifica se o ano indicado no parâmetro `year` é bissexto[2].

```
int8 is_leap(uint16 year) {
      return year % 4 == 0;
}
```

4.2. Implemente, em *assembly* do PDS16, a função `year_days` que determina o número de dias que decorreram desde o início do ano indicado no parâmetro `year` até ao mês e dia indicados, respetivamente, nos parâmetros `month` e `day`.

```
int16 month_days[] =
    {0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365};

int16 year_days(uint16 year, uint8 month, uint8 day) {
      return month_days[month - 1] +
            (month > 2 && is_leap(year) ? 1 : 0) + day - 1;
}
```

4.3. Considere a seguinte definição da função `days_since`, que determina o número de dias que decorreram desde o início do ano indicado no parâmetro `year_ref` até à data definida pelos parâmetros `year`, `month` e `day`.

```
uint16 days_since(uint16 year_ref,uint16 year,uint8 month,uint8 day){
      uint16 days = 0;
      for (uint16 y = year_ref; y < year; ++y)
            days += 365 + is_leap(y);
      return days + year_days(year, month, day);
}
```

a) Implemente, em *assembly* do PDS16, a função `days_since`.

b) Determine, em número de *bytes*, a quantidade de memória de código ocupada pela função `days_since`.

c) Supondo que a função `days_since` tem como argumentos `year_ref` = 2000, `year` = 2017, `month` = 4 e `day` = 20, indique o número de ciclos de relógio gastos na execução da função desenvolvida, excluindo os ciclos gastos na execução das funções chamadas.

4.4. Implemente a função `main` na qual se determina o número de dias entre duas datas.

```
uint16 days;
uint16 year1 = 2017;
uint8 month1 = 3;
uint8 day1 = 30;
uint16 year2 = 1995;
uint8 month2 = 3;
uint8 day2 = 8;

void main() {
      days = days_since(1970, year1, month1, day1) -
             days_since(1970, year2, month2, day2);
}
```

2 - Definição válida para datas entre 1-1-1901 e 31-12-2099. A definição exata de ano bissexto pode ser consultada aqui: https://pt.wikipedia.org/wiki/Ano_bissexto.

## 5. Avaliação

O trabalho deve ser realizado em grupo e conta para a avaliação da disciplina, estando sujeito a discussão final. A sua apresentação decorrerá em data a combinar com o docente da turma.

Cada grupo deverá entregar o trabalho desenvolvido, na forma de listagens dos programas realizados, devidamente indentadas e comentadas.

No caso do exercício 4.3, deverá incluir as respostas na própria listagem, na forma de comentários.

## Anexo

**Recomendações para escrita de programas em linguagem a*ssembly***

- 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 caráter '_' (sublinhado).
- O programa é disposto na folha, 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 da diretiva, na terceira coluna os parâmetros da instrução ou da diretiva e na quarta coluna os comentários até ao fim da linha (começados por '**;**'). Cada linha contém apenas uma *label*, instrução ou diretiva.
- Para definir as colunas deve usar-se caracteres TAB. A largura mais conveniente é 8.
- As linhas com *label* não devem conter instrução ou diretiva. Isso permite usar *labels* compridas sem desalinhar a tabulação e cria separações na sequência de instruções.
- Exemplo:

```
        .section startup
        .org     0
        jmp      main

        .section directdata
        .org     4

/*      #define TABLE_DIM 6
        uint16 table1[TABLE_DIM] = {10, 20, 5, 6, 34, 9};
        uint16 table2[TABLE_DIM - 3] = {11, 22, 33};
        int16 p, q;

        void main() {
                p = search(table1, TABLE_DIM, 20);
                q = search(table2, TABLE_DIM - 3, 31);
        }
*/
        .data
        .equ     TABLE_DIM, 6
table1:
        .word    10, 20, 5, 6, 34, 9
table2:
        .word    11, 22, 33

        .section directdata
p:
        .word    0
```

```
q:
        .word    0

        .text
main:
        ldi      r0, #low(table1)
        ldih     r0, #high(table1)
        ldi      r1, #TABLE_DIM
        ldi      r2, #20
        jmpl     search
        st       r0, p

        ldi      r0, #low(table2)
        ldih     r0, #high(table2)
        ldi      r1, #TABLE_DIM - 3
        ldi      r2, #30
        jmpl     search
        st       r0, q

        jmp      $

/*      r0                      r0                  r1          r2
        int16 search(uint16 array[], uint8 array_size, uint16 value) {
                 for( uint8 i = 0; i < array_size && array[i] != value; ++i)
                          ;
                 if( i < array_size)
                          return i;
                 return -1;
        }
*/
        .section directdata
search_r3:
        .word    0
search_r4:
        .word    0

        .text
search:
        st       r3, search_r3    ; saving r3 and r4
        st       r4, search_r4    ; since these are not parameters
        ldi      r3, #0           ; r3 - i
for:
        sub      r6, r3, r1 ; i - array_size
        jnc      for_end
        ld       r4, [r0, r3]     ; array[i] != value
        sub      r6, r4, r2
        jz       for_end
        inc      r3               ; ++i
        jmp      for
for_end:
        sub      r6, r3, r1 ; if (i < array_size)
        jnc      if_end
        mov      r0, r3           ; return i
        jmp      search_end
if_end:
        ldi      r0, #0           ; return -1
        dec      r0
search_end:
        ld       r3, search_r3
        ld       r4, search_r4
        ret

        .end
```