ATmega128
Programação C

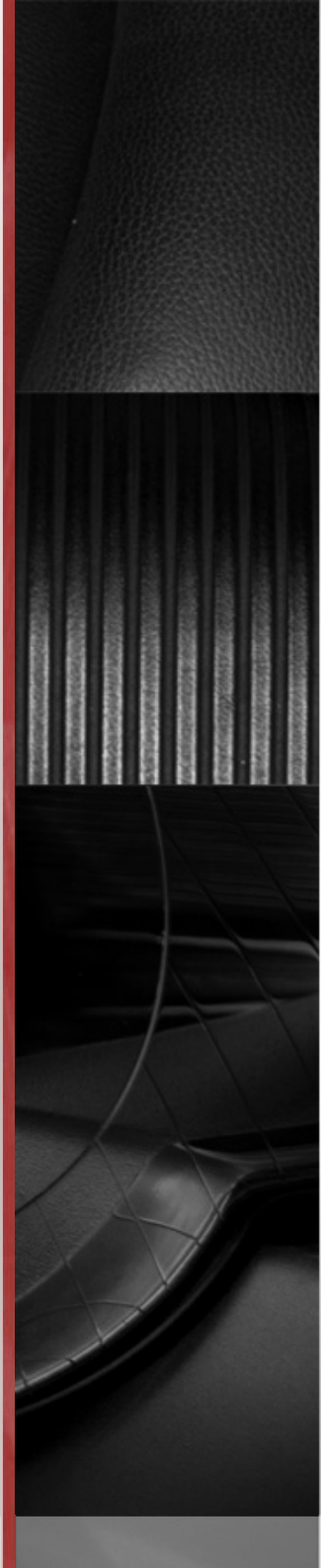
Lino Figueiredo– ISEP

Microprocessadores e Microcontroladores

Licenciatura Engenharia Electrotécnica e de Computadores

Ano Lectivo 2013-2014

Revisão 1.3 - 2022



Programação em Assembly *versus* C

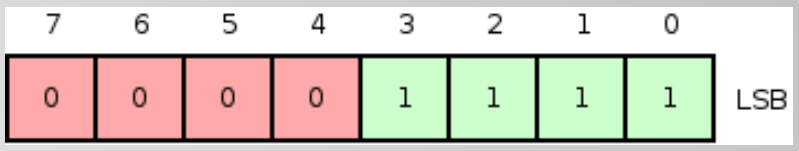
- Programação em Assembly
  - permite compreender melhor a ’máquina’ (Programação baixo nível)
  - bastante útil quando se tem requisitos temporais e de memória de programa muito apertados
- Programação em C
  - portabilidade do código mesmo entre diferentes arquitecturas
  - tempo de desenvolvimento muito inferior
  - permite o desenvolvimento distribuído mais facilmente
  - optimização do código bastante elevada

- Em muitas situações o programador só consegue escrever bom código em C se conhecer bem as instruções em *assembly* disponíveis e a forma como o código é gerado.

Programação C – Tipos de variáveis I

- O microcontrolador Atmega agrupa **8 bits** para formar um **byte** com o bit menos significativo à direita (LSB – Least Significant Bit)
- Cada bit é numerado de 0 a 7 sendo o LSB o bit 0

Exemplo: valor decimal 15 representado num byte



- 3 formas de codificar em C uma variável inicializada com o valor decimal 15

```
unsigned char a,b,c;
a = 15;          /* decimal */
b = 0x0F;        /* hexidecimal */
c = 0b00001111; /* binary */
```

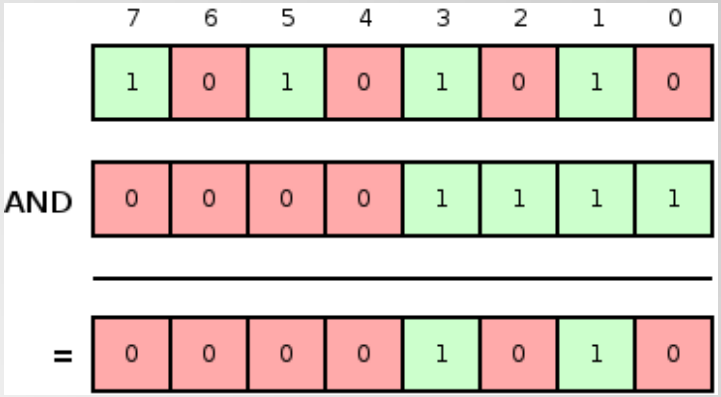
Programação C – Tipos de variáveis II

Tipo	Numero de bytes
char	1
int	2
long	4
long long	8
float, double	4

- Sempre que se define uma variável, o compilador irá reservar um espaço em memória correspondente ao tamanho que ela ocupa.
- Deve-se usar sempre o tipo de variável mais pequeno necessário para a aplicação.

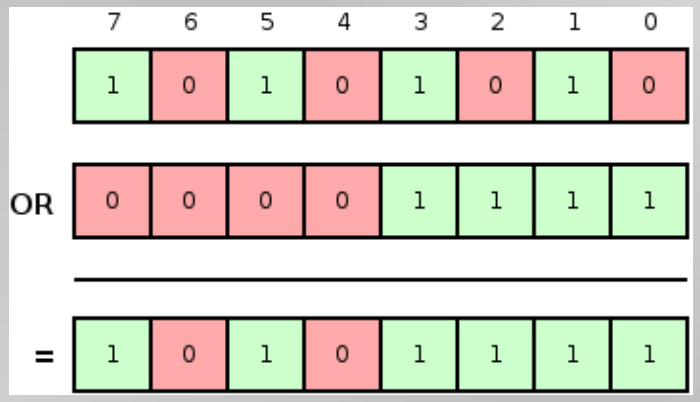
Programação C – Operações lógicas I

- AND
  - O símbolo **&** representa a operação lógica **AND** em C




```
unsigned char a,b,c;
a = 0xAA;    /* 10101010 */
b = 0x0F;    /* 00001111 */
c = a & b;   /* 00001010 */
```

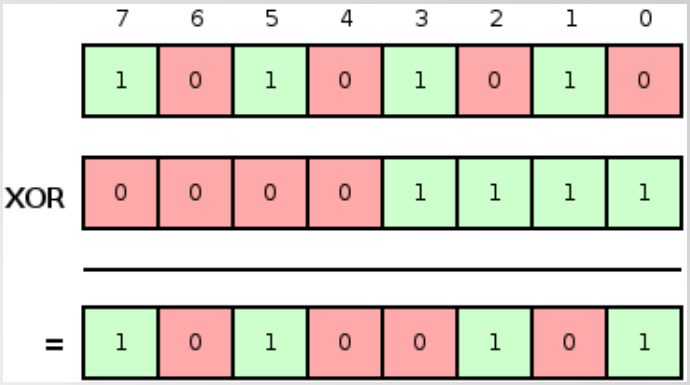
- OR
  - O símbolo **|** representa a operação lógica **OR** em C




```
unsigned char a,b,c;
a = 0xAA;    /* 10101010 */
b = 0x0F;    /* 00001111 */
c = a | b;   /* 10101111 */
```

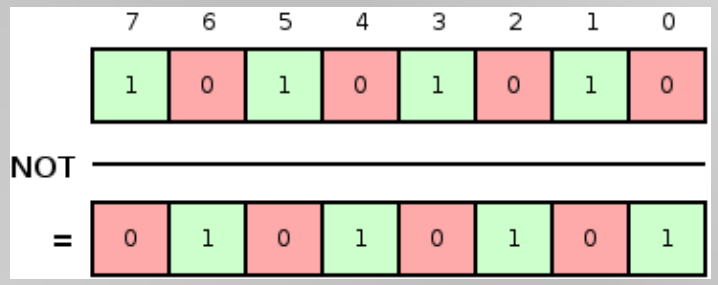
Programação C – Operações lógicas II

- XOR
  - O símbolo ^ representa a operação lógica **XOR** em C




```
unsigned char a,b,c;
a = 0xAA;   /* 10101010 */
b = 0x0F;   /* 00001111 */
c = a ^ b;  /* 10100101 */
```

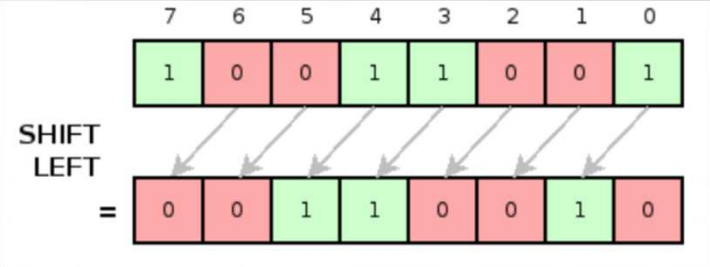
- NOT
  - O símbolo ~ representa a operação lógica **NOT** em C

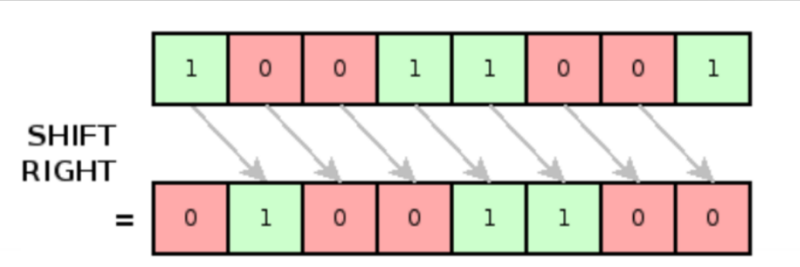



```
unsigned char a,b,c;
a = 0xAA; /* 10101010 */
b = ~a;   /* 01010101 */
```

Programação C – Deslocar bits (*shift*)

- A operação *shift* desloca todos os bits de um byte para a esquerda ou para a direita
  - O símbolo **<<** desloca todos os bits para a esquerda
  - O símbolo **>>** desloca todos os bits para a direita






```
unsigned char a,b,c;
a = 0x99; /* 10011001 */
b = a<<1; /* 00110010 */
c = a>>3; /* 00010011 */
```

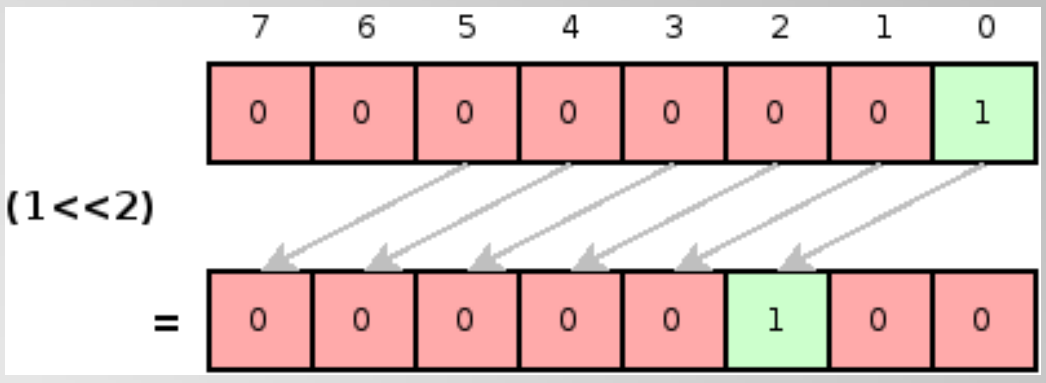
Programação C – Manipular Bits I

- Para manipular um bit de um byte usa-se a mascara do bit

  Exemplo: 00000100 é a mascara do bit 2

  01000000 é a mascara do bit 6

- Para criar a mascara de um bit usa-se a operação *shift* à esquerda



- Código C em que a variável **a** representa a mascara do bit 2

```
unsigned char a,b,c;

a = (1<<2);        /* 00000100 */
```

Programação C – Manipular Bits II

- Colocar bit a **1** usando mascara do bit

```
unsigned char a,b,c;
a = 0x08;        /* 00001000 */
                 /* set bit 2 */
a = a | (1<<2); /* 00001100 */
```

Ou

```
unsigned char a,b,c;
a = 0x08;        /* 00001000 */
                 /* set bit 2 */
a |= (1<<2);     /* 00001100 */
```

- Alterar vários bits

```
unsigned char a,b,c;
a = 0x08;              /* 00001000 */
                       /* set bits 1 and 2 */
a |= (1<<2)|(1<<1);    /* 00001110 */
```

- Colocar bit a **0** usando mascara do bit

```
unsigned char a,b,c;
a = 0x0F;          /* 00001111 */
                   /* clear bit 2 */
a = a & ~(1<<2);   /* 00001011 */
```

Ou

```
unsigned char a,b,c;
a = 0x0F;          /* 00001111 */
                   /* clear bit 2 */
a &= ~(1<<2);      /* 00001011 */
```

- Alterar vários bits

```
unsigned char a,b,c;
a = 0x0F;                  /* 00001111 */
                           /* clear bit 1 and 2 */
a &= ~((1<<2)|(1<<1));     /* 00001001 */
```

Programação C – Manipular Bits III

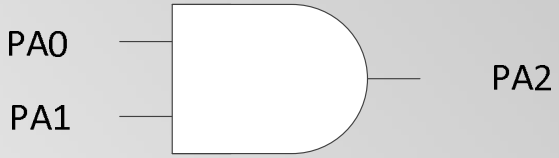
- Inverter o estado de um bit (*Toggle*)

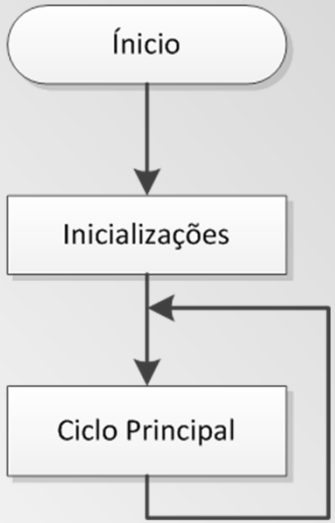
```
unsigned char a;
a = 0x0F;           /* 00001111 */
                    /* toggle bit 2 */
a = a ^ (1<<2);     /* 00001011 */
a = a ^ (1<<2);     /* 00001111 */
```

Ou

```
unsigned char a;
a = 0x0F;           /* 00001111 */
                    /* toggle bit 2 */
a ^= (1<<2);        /* 00001011 */
a ^= (1<<2);        /* 00001111 */
```

Porta Lógica AND - Programa C






```
#include <avr/io.h>                 // biblioteca microcontroladores AVR

void init()
{
    DDRA = 0b00000100;              // configura PORTA
}                                   // pino 2 saída, restantes entradas

int main(void)
{
    init();                         // chama função init

    while(1)                        // ciclo principal
    {
        if ((PINA & 0b00000001) && (PINA & 0b00000010))
            PORTA |= (1<<2);        // coloca pino 2 PORTA a 1
        else
            PORTA &= ~(1<<2);       // coloca pino 2 PORTA a 0
    }
}
```

Porta Lógica AND – Programa Assembly

com optimização (-O1)

```
    12: {
    13:     DDRA = 0b00000100;          // configura PORTA
0000004F 84.e0                LDI R24,0x04      Load immediate
00000050 8a.bb                OUT 0x1A,R24      Out to I/O location
00000051 08.95                RET       Subroutine return
    13:     DDRA = 0b00000100;          // configura PORTA
    14: }                               // pino 2 sa◆da, restantes entradas
    15:
    16: int main(void)
    17: {
    18:     init();                     // chama fun◆◆o init
00000052 fc.df                RCALL PC-0x0003       Relative call subroutine
    22:         if ((PINA & 0b00000001) && (PINA & 0b00000010))
00000053 c8.9b                SBIS 0x19,0       Skip if bit in I/O register set
00000054 04.c0                RJMP PC+0x0005        Relative jump
--- No source file -----------------------------------------------------------
00000055 c9.9b                SBIS 0x19,1       Skip if bit in I/O register set
--- No source file -----------------------------------------------------------
00000056 02.c0                RJMP PC+0x0003        Relative jump
--- C:\Users\Lino\HOMES LINO\ISEP\SDIG2\Programacao ATmega128\programas\PortaAND_C\
    23:             PORTA |= (1<<2);    // coloca pino 2 PORTA a 1
00000057 da.9a                SBI 0x1B,2        Set bit in I/O register
00000058 fa.cf                RJMP PC-0x0005        Relative jump
    25:             PORTA &= ~(1<<2);   // coloca pino 2 PORTA a 0
00000059 da.98                CBI 0x1B,2        Clear bit in I/O register
0000005A f8.cf                RJMP PC-0x0007        Relative jump
--- No source file -----------------------------------------------------------
```

Optimizações do compilador AVR-GCC

- As optimizações não são mais do que um algoritmo que analisa o código assembly gerado a partir do C e verifica se existem partes do código que se podem reduzir ou mesmo eliminar.
  - Deve-se ter as optimizações ligadas por forma a que o código gerado seja o mais rápido possível (menos código que tem de ser executado);
  - Menos código implica também menos espaço ocupado na memória de programa;

- O compilador AVR-GCC tem quatro opções de optimização (Além das optimizações desligadas -O0):
  - -O1, -O2 e -O3 são diferentes níveis de optimizar em termos de velocidade de execução;
  - -Os é a opção para optimizar em termos de tamanho do código do programa

Porta Lógica AND – Programa Assembly

sem optimização (-O0)

Rotina *init()*

```
    12: {
0000004F cf.93                PUSH R28      Push register on stack
00000050 df.93                PUSH R29      Push register on stack
00000051 cd.b7                IN R28,0x3D       In from I/O locati
00000052 de.b7                IN R29,0x3E       In from I/O locati
    13:     DDRA = 0b00000100;          // configura PORTA
00000053 8a.e3                LDI R24,0x3A      Load immediate
00000054 90.e0                LDI R25,0x00      Load immediate
00000055 24.e0                LDI R18,0x04      Load immediate
00000056 fc.01                MOVW R30,R24      Copy register pair
00000057 20.83                STD Z+0,R18       Store indirect wit
    14: }                               // pino 2 sa◆da, restantes
00000058 df.91                POP R29       Pop register from stac
00000059 cf.91                POP R28       Pop register from stac
0000005A 08.95                RET       Subroutine return
```

Instrução *if*

```
    22:         if ((PINA & 0b00000001) && (PINA & 0b00000010))
00000060 89.e3                LDI R24,0x39      Load immediate
00000061 90.e0                LDI R25,0x00      Load immediate
00000062 fc.01                MOVW R30,R24      Copy register pair
00000063 80.81                LDD R24,Z+0       Load indirect with displacement
00000064 88.2f                MOV R24,R24       Copy register
00000065 90.e0                LDI R25,0x00      Load immediate
00000066 81.70                ANDI R24,0x01     Logical AND with immediate
00000067 99.27                CLR R25       Clear Register
00000068 00.97                SBIW R24,0x00     Subtract immediate from word
00000069 a1.f0                BREQ PC+0x15      Branch if equal
--- No source file -----------------------------------------------------------
0000006A 89.e3                LDI R24,0x39      Load immediate
0000006B 90.e0                LDI R25,0x00      Load immediate
0000006C fc.01                MOVW R30,R24      Copy register pair
0000006D 80.81                LDD R24,Z+0       Load indirect with displacement
0000006E 88.2f                MOV R24,R24       Copy register
0000006F 90.e0                LDI R25,0x00      Load immediate
00000070 82.70                ANDI R24,0x02     Logical AND with immediate
00000071 99.27                CLR R25       Clear Register
00000072 00.97                SBIW R24,0x00     Subtract immediate from word
--- No source file -----------------------------------------------------------
00000073 51.f0                BREQ PC+0x0B      Branch if equal
--- C:\Users\Lino\HOMES LINO\ISEP\SDIG2\Programacao ATmega128\programas\PortaAND
    23:             PORTA |= (1<<2);    // coloca pino 2 PORTA a 1
00000074 8b.e3                LDI R24,0x3B      Load immediate
00000075 90.e0                LDI R25,0x00      Load immediate
00000076 2b.e3                LDI R18,0x3B      Load immediate
00000077 30.e0                LDI R19,0x00      Load immediate
00000078 f9.01                MOVW R30,R18      Copy register pair
00000079 20.81                LDD R18,Z+0       Load indirect with displacement
0000007A 24.60                ORI R18,0x04      Logical OR with immediate
0000007B fc.01                MOVW R30,R24      Copy register pair
```

Programação C – Temporização I

- Rotina de *Delay*

```
#include <avr/io.h>

void init()
{
    DDRA = 0b00000001;
    PORTA = 0b00000000;
}

void delay(double i)
{
    for( i=i; i!=0;i--);
}

int main(void)
{
   init();

        while(1)
        {
            delay(500);
            PORTA = PORTA ^ 0b00000001;
        }

}
```

- Inverte o estado do pino 0 da PORTA com um intervalo de tempo definido na rotina *Delay*
- O tempo de rotina de *Delay* é calculado multiplicando o tempo que demora uma iteração do ciclo **For** pelo o valor da variável **i**
- O tempo que demora a executar uma iteração do ciclo **FOR** depende do oscilador

Programação C – Temporização II

- Biblioteca *Delay.h*

```
#include <avr/io.h>
#define F_CPU 16000000UL
#include <util/delay.h>

void init(void)
{
    DDRA = 0b00000001;
    PORTA = 0b00000000;
}

int main(void)
{

        init();
        while(1)
        {
            _delay_ms(5);
            PORTA = PORTA ^ 0b00000001;
        }
 }
```

- A rotina _delay_ms(i) temporiza i milisegundos

```
_delay_ms(5);
```

5 milisegundos

- O valor do oscilador é definido através do comando:

```
#define F_CPU 16000000UL
```

Oscilador 16MHz

Programação C – Interrupções I

- Actualizar, com intervalo de 5ms, o display de 7 segmentos com o número do *switch* que foi pressionado

```c
#include <avr/interrupt.h>

// criar tabela em memória de programa - const
const unsigned char digitos[] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82};

// Variaveies globais
volatile unsigned char dado;    // variavel usada na rotina de interrupção
unsigned char switches;

void inic(void)
{
    DDRA = 0b11000000;      // configura PORTA
    PORTA = 0b11000000;     // pinos 6,7 saídas, restantes entradas

    DDRC = 0xFF;            // configura PORTC
    PORTC = 0xFF;           // todos os pinos saidas

    OCR0 = 77;              // configura TC0 @16MHz
    TCCR0 = 0b00001111;     // 5ms, prescaler 1024, CTC
    TIMSK |= 0b00000010;    // interrupção TC0

    SREG |= 0x80;           // activa flag I do SREG
}
// rotina interrupção TC0
ISR(TIMER0_COMP_vect)
{
    if (dado != 0)
    {
        PORTC = digitos[dado];  // coloca no display o digito definida por dado
        dado = 0;
    }
}
```

```c
// rotina principal
int main(void)
{
    inic();

    dado = 1;
    PORTC = digitos[dado];  // coloca digito 1 no display

    while(1)
    {
        switches = PINA & 0b00000111;   // testa entradas

        switch(switches)
        {
            case 0b00000110:            // SW 1
            dado= 1;
            break;

            case 0b00000101:            // SW 2
            dado=2;
            break;

            case 0b00000011:            // SW 3
            dado=3;
            break;
        }
    }
}
```

Programação C – Interrupções II

```
#include <avr/interrupt.h>
```

- Para usar interrupções deve incluir-se a biblioteca *interrupt.h*

```
// rotina interrupção TCO
ISR(TIMER0_COMP_vect)
{
    if (dado != 0)
    {
        PORTC = digitos[dado];  // coloca no display o digito definida por dado
        dado = 0;
    }
}
```

- O nome da rotina de interrupção é definido por:
  - ISR(NOME_INTERRUPÇÃO_vect)
  - NOME_INTERRUPÇÃO – nome atribuído ao vector de interrupção (ex:, INT0)

```
// Variaveies globais
volatile unsigned char dado;     // variavel usada na rotina de interrupção
unsigned char switches;
```

- Sempre que uma função não é chamada explicitamente (ex: **ISR(TIMER0_COMPA_vect)**), o compilador pode optimizar o acesso às variáveis (ex: **dado**) de forma a que uma alteração na rotina de interrupção seja ignorada na *main*. Assim, para evitar este problema deve-se declarar as variáveis como ***volatile***

Integração *Assembly* e C

- Faz o *toggle* de um bit no tempo definido pela rotina **DelayD()** implementada em *assembly*

```c
#include <avr/io.h>

// declara a função Delay_D como externa
extern void DelayD(void);

unsigned char val_D, n_DelayD = 0;

void inic(void)
{
    DDRA = 0b00001010;
    PORTA = 0;
}

int main(void)
{
    inic();

    while(1)
    {
        val_D=10;           // inicializa o valor do contador
                            // da rotina DelayD
        DelayD();           // chama rotina em Assembly

        PORTA ^= (1<<1);    // faz toggle do bit1
                            // sempre que chama rotina DelayS

        if (n_DelayD == 10) // verifica se rotina DelayS foi chamada 10 vezes
        {
            PORTA ^= (1<<3);    // faz toggle do bit3
            n_DelayD=0;         // inicializa n_DelayS
        }
    }
}
```

- Programa em C

```
#include <avr/io.h>

.extern val_D       ; declara variaveis como externas
.extern n_DelayD

.global DelayD      ; declara label global

;***************************************************************
;           rotina (função)  DelayD
DelayD:
        push    r20             ; guarda na stack o valor dos registos
        push    r21             ; que são usados na rotina

        lds     r21,SREG        ; guarda SREG
        push    r21

        lds     r20,val_D       ; inicializa os registos com os valores
        lds     r21,n_DelayD    ; das variaveis definidos no programa C

ciclo_d:
        dec     r20             ; decrementa registo de contagem
        brne    ciclo_d

        inc     r21             ; incrementa e guarda nº de vezes
        sts     n_DelayD,r21    ; que a rotina é chamada

        pop     r21
        sts     SREG,r21        ;Repõe SREG

        pop     r21             ; repõe os valores dos registos
        pop     r20
        ret                     ; fim da rotina
```

- Função DelayD em Assembly

Bibliografia

- Microcontroladores - João Paulo Baptista, ISEP
- Programação em C para uC – Nuno Dias , ISEP
- AVR Tutorial – Micah Carrick
- Arquitectura de Sistema Computacionais – Paulo Sampaio, UMA