

Relatório de Microprocessadores  
2007/2008

Engenharia Física Tecnológica

**PROGRAMAÇÃO DE UM MICROPROCESSADOR  
EM C PARA CONTROLO DE LEDs ATRAVÉS DE  
UMA CONSOLA E COMUNICAÇÃO EM SÉRIE**

---

*Laboratório II*

**Trabalho realizado por:**

André Cunha, nº53757

João Pereira, nº 55315

**Grupo 3; 5ªfeira 13:00-16:00h**

Lisboa, 10 de Outubro de 2007

## Introdução e Objectivos

---

O objectivo último deste trabalho laboratorial consiste em controlar os *leds* da placa incluída no SDK através de uma consola no PC utilizando a porta série para efectuar as comunicações.

Deverão ainda, ser feitas algumas considerações acerca das vantagens e desvantagens das comunicações assíncronas tendo em conta o caso em estudo, a porta série.

## Implementação e Procedimento

---

A implementação deverá ser atingida naturalmente através da introdução de interrupções. Tendo como base um pequeno programa facultado no enunciado que envia mensagens periódicas para a consola de acordo com um *timer* acendendo um *led* aquando do envio e gera uma interrupção sempre que um *carriage return* (CR) é detectado na consola acendendo um *led* aquando do mesmo.

### 1ª sessão de laboratório

Material utilizado:

- MPLAB IDE
- SDK PIC184550

Começou-se por tentar perceber o funcionamento do programa de exemplo fornecido pelo docente. Os comentários adicionais estão colocados abaixo:

```
//=====
// Filename: MAIN.C
//=====
// Author:   Mike Garbutt
// Company:  Microchip Technology Inc.
// Revision: 1.00
// Date:    04/26/2001
//=====
// Compiled using MPLAB-C18 V1.00.31
// Include Files: P18C452.H V1.14.2.2
//=====
//
// Example code to generate a TMR0 interrupt and toggle LEDs on pins RB0 and
// RB7. Toggles RB0 in the interrupt routine and sets RB7 to match RB0 in the
// main routine. This demonstrates that code is executing in both routines.
//
//=====

//-----
//inclusão das bibliotecas
#include <p18f4550.h>
#include <stdio.h>
#include <usart.h>
#include <string.h>

//-----
```

```

void main (void);
void InterruptHandlerHigh (void);

union
{
    struct
    {
        unsigned Timeout:1; //construção de uma estrutura que
        unsigned Rx:1; //é uma palavra de 8 bits dos quais só
        unsigned None:6; //vamos usar dois
    } Bit;
    unsigned char Byte;
} Flags;

//-----
// Main routine
char inputstr[80];
char str_pos = 0;

void main ()
{
    int i = 0;

    Flags.Byte = 0;
    INTCON = 0x20; //disable global and enable TMR0 interrupt
    INTCON2 = 0x84; //TMR0 high priority
    TMR0H = 0; //clear timer
    TMR0L = 0; //clear timer
    TOCON = 0x82; //set up timer0 - prescaler 1:8
    INTCONbits.GIEH = 1; //enable interrupts
    TRISB = 0xF0; //permite a utilização das portas B como output

//inicia USART que permite a comunicação através da porta série com as seguintes
//opções

    OpenUSART( USART_TX_INT_OFF & //envio de interrupções off
               USART_RX_INT_ON & //recepção de interrupções on
               USART_ASYNC_MODE & //modo assíncrono
               USART_EIGHT_BIT & //utilização de palavras de 8 bits
               USART_CONT_RX & //recepção continuada
               USART_BRGH_HIGH, //modo de funcionamento para baud rates
               //altas em palavras de 8 bits
               25 ); //factor que regula a baud rate usada

    while (1)
    {
        if (Flags.Bit.Timeout == 1) //timeout?
        {
            Flags.Bit.Timeout = 0; //clear timeout indicator

            printf("MicroProc Grupo %d\r", i++) //escreve a string no buffer
            // de destino.

        }

        if (Flags.Bit.Rx == 1) //foi inserido um caracter?
        {
            if(inputstr[str_pos - 1] == 0x0D) //o último caracter foi o ENTER?
            {
                printf("CR detected! %s\r", inputstr); //print "commando
                //detectado"

                str_pos = 0; //reset da posição na string de buffer
                LATBbits.LATB2 = !LATBbits.LATB2; //toggle LED on RB2

            }
        } //neste caso envia para a porta serie
    } //fim do ciclo while infinito
}

//o seguinte bloco de código representa a gestão de interrupções.

```

```

//há um salto para uma zona da memória endereçada por 0x08 sempre que ocorre uma
//interrupção de alta prioridade

//em geral, o pragma vai permitir a activação de opções específicas de compilação
//neste caso vai permitir que tenhamos interrupções disponíveis e subsequente gestão
//das mesmas, algo que é de outra forma impossível usando apenas o C

//-----
// High priority interrupt vector

#pragma code InterruptVectorHigh = 0x08
void
InterruptVectorHigh (void)
{
    _asm
        goto InterruptHandlerHigh //jump to interrupt routine
    _endasm
}

//rotina que gere uma interrupção de alta prioridade
//-----
// High priority interrupt routine

#pragma code
#pragma interrupt InterruptHandlerHigh

void InterruptHandlerHigh ()
{
    if (INTCONbits.TMR0IF)                //check for TMR0 overflow
    {
        INTCONbits.TMR0IF = 0;           //clear interrupt flag
        Flags.Bit.Timeout = 1;           //indicate timeout
        LATBbits.LATB0 = !LATBbits.LATB0; //toggle LED on RB0
    }

    if (PIR1bits.RCIF)                    //check RX interrupt
    {
        PIR1bits.RCIF = 0;                //então mete a variável de interrupção a 0
        Flags.Bit.Rx = 1;                 //a flag Rx vai a 1
        inputstr[str_pos] = getcUSART(); //vai buscar a inputstr à porta série
                                         //caracter a caracter (em cada iterada)

        str_pos++;                         //incrementa a posição de escrita na
                                         //string de destino

        if(str_pos == 80){str_pos = 0;} //verifica se a string de 80 caracteres
                                         //está cheia e caso o esteja faz reset da
                                         //posição de escrita na string a 0

        INTCONbits.PEIE=1;                //permite interrupções feitas por
                                         //periféricos
    }
}

//-----

```

Antes de introduzir os fluxogramas que descrevem a rotina do programa propriamente dito bem como a rotina de gestão de interrupções é conveniente explicitar alguns aspectos mais técnicos do código.

É utilizada uma estrutura que prevê a existência de interrupções de alta prioridade, no início do programa, o protocolo de comunicações através da porta série é configurado para modo continuado de recepção de dados (interrupções via porta série), assincronamente utilizando palavras de 8 bits e *baud rates* elevadas. O envio de interrupções está desactivado.

**Nota:** os fluxogramas podem ser consultados em anexo no final do documento.

## 2ª sessão de laboratório

- MPLAB IDE e compilador de C para este IDE
- SDK PIC184550

Na segunda sessão procedeu-se então à elaboração do programa de acordo com os objectivos expressos tendo como base o código supracitado. Desta forma, fez-se um *highlight* apenas do corpo de código relevante para os objectivos, uma vez que existe código repetido relacionado com as interrupções, preparação do protocolo de comunicações, a própria estrutura do programa, etc.

```
//Microprocessadores, Trabalho Laboratorial n° 2
//André Cunha, n° 53757, LEFT
//João Pereira, n° 55315, LEFT
//
// Programa que permite ligar e desligar os leds através da introdução de
// comandos on, off e toggle numa consola cujas funções são explícitas
//
//=====

//-----
//inclusão das bibliotecas
#include <p18f4550.h>
#include <stdio.h>
#include <usart.h>
#include <string.h>

//-----

void main (void);
void InterruptHandlerHigh (void);

union
{
    Struct                //construção de uma estrutura que
    {                    //é uma palavra de 8 bits dos quais só
                        //vamos usar dois
        unsigned Timeout:1; //flag to indicate a TMR0 timeout
        unsigned Rx:1;      //flag to indicate a Rx interrupt
        unsigned None:6;
    } Bit;
    unsigned char Byte;
} Flags;

//-----
// Main routine
char inputstr[80];
char str_pos = 0;

void main ()
{
    int i = 0;

    Flags.Byte = 0;
    INTCON = 0x20;                //disable global and enable TMR0 interrupt
    INTCON2 = 0x84;              //TMR0 high priority
    TMR0H = 0;                   //clear timer
    TMR0L = 0;                   //clear timer
    TOCON = 0x82;                //set up timer0 - prescaler 1:8
```

```

INTCONbits.GIEH = 1;           //enable interrupts
TRISB = 0xF0;                 //permite a utilização das portas B como output

//inicia USART que permite a comunicação através da porta série com as seguintes
//opções

//inicializar os leds
LATBbits.LATB0 = 0;
LATBbits.LATB1 = 0;
LATBbits.LATB2 = 0;
LATBbits.LATB3 = 0;

OpenUSART(    USART_TX_INT_OFF &    //envio de interrupções off
              USART_RX_INT_ON &    //recepção de interrupções on
              USART_ASYNC_MODE &   //modo assíncrono
              USART_EIGHT_BIT &    //utilização de palavras de 8 bits
              USART_CONT_RX &      //recepção continuada
              USART_BRGH_HIGH,      //modo de funcionamento para baud rates
              //altas em palavras de 8 bits
              25 );                 //factor que regula a baud rate usada

//inicio do ciclo infinito
while (1)
{
    if (Flags.Bit.Timeout == 1)     //detecta se houve timeout interrupt
    {
        Flags.Bit.Timeout = 0;      //clear timeout indicator
        printf("G3 QUINTA %d\r", i++); //escreve a string no buffer de
                                        //destino.
    }
    //verifica se foi feita uma interrupção pelo teclado
    if (Flags.Bit.Rx == 1){
        //verifica se o caracter "enter" existe no ultimo input da string
        if (inputstr[str_pos - 1] == 0x0D)
        {
            //altera o caracter "enter" por um caracter "NULL" = '\0'
            //no ultimo input da string
            //desde modo a string fecha-se e fica só com o tamanho do número
            //de caracteres introduzidos na string
            inputstr[str_pos - 1]='\0';

            //inicializa o str_pos para uma nova string que é gravada no array
            // inputstr
            str_pos = 0;

            //print, dá o alarme de deteçcao e o print do
            //comando fornecido pelo utilizador
            printf("COMMAND DETECTED! %s\r", inputstr);

            //compara o inputstr com um comando valido "led t x" onde x é um
            //numero entre 0 e 3 para fazer o toggle do led correspondente x,
            //se nao introduzirmos um valor x, faz o toggle a todos leds
            if(strncmpppgm2ram(inputstr,"led t ",6)==0){

                //vai buscar o caracter que ocupa a sexta posiccao na string para
                //executar os case correspondentes
                switch(inputstr[6]){

                    case '0': printf("TOGGLE LED 0 \r");           //print
                                LATBbits.LATB0 = !LATBbits.LATB0; //toggle led 0
                                break;

                    case '1':printf("TOGGLE LED 1 \r");           //print
                                LATBbits.LATB1 = !LATBbits.LATB1; //toggle led 1
                                break;
                }
            }
        }
    }
}

```

```

        case '2':printf("TOGGLE LED 2 \r");          //print
                LATBbits.LATB2 = !LATBbits.LATB2; //toggle led 2
                break;

        case '3':printf("TOGGLE LED 3 \r");          //print
                LATBbits.LATB3 = !LATBbits.LATB3; //toggle led 3
                break;

        default : printf("TOGGLE ALL LEDS \r");      //print
                //toggle todos os leds

                LATBbits.LATB0 = !LATBbits.LATB0;
                LATBbits.LATB1 = !LATBbits.LATB1;
                LATBbits.LATB2 = !LATBbits.LATB2;
                LATBbits.LATB3 = !LATBbits.LATB3;
                break;
    }
}

//compara o inputstr com um comando valido "led on x" onde x é um numero entre 0 e 3
//para acender do led correnpondente x, se nao introduzirmos um valor x,acende todos
//leds

    if(strncmppgm2ram(inputstr,"led on ",7)==0){

//vai buscar o caracter que ocupa a setima posiccao na string para executar os case
//correspondentes

        switch(inputstr[7]){

            case '0': printf("LED 0 ON \r");          //print
                    LATBbits.LATB0 = 1;             //acende led 0
                    break;

            case '1': printf("LED 1 ON \r");          //print
                    LATBbits.LATB1 = 1;             //acende led 1
                    break;

            case '2': printf("LED 2 ON \r");          //print
                    LATBbits.LATB2 = 1;             //acende led 2
                    break;

            case '3': printf("LED 3 ON \r");          //print
                    LATBbits.LATB3 = 1;             //acende led 3
                    break;

            default : printf("ALL LEDS ON \r");      //print

                //acende todos os leds
                LATBbits.LATB0 = 1;
                LATBbits.LATB1 = 1;
                LATBbits.LATB2 = 1;
                LATBbits.LATB3 = 1;
                break;
        }
    }

//compara o inputstr com um comando valido "led off x" onde x é um numero entre 0 e 3
//para apagar do led correnpondente x, se nao introduzirmos um valor x , apaga todos
//leds

    if(strncmppgm2ram(inputstr,"led off ",8)==0){

//vai buscar o caracter que ocupa a oitava posiccao na string para executar os case
//correspondentes

        switch(inputstr[8]){

            case '0': printf("LED 0 OFF \r");          //print
                    LATBbits.LATB0 = 0;             //apaga led 0
                    break;

```

```

        case '1': printf("LED 1 OFF \r"); //print
                 LATBbits.LATB1 = 0; //apaga led 1
                 break;

        case '2': printf("LED 2 OFF \r"); //print
                 LATBbits.LATB2 = 0; //apaga led 2
                 break;

        case '3': printf("LED 3 OFF \r"); //print
                 LATBbits.LATB3 = 0; //apaga led 3
                 break;

        default : printf("ALL LEDS OFF \r"); //print

        //apaga todos os leds
        LATBbits.LATB0 = 0;
        LATBbits.LATB1 = 0;
        LATBbits.LATB2 = 0;
        LATBbits.LATB3 = 0;
        break;
    }
}
} //fim do ciclo while infinito)

```

//o seguinte bloco de código representa a gestão de interrupções.

//há um salto para uma zona da memória endereçada por 0x08 sempre que ocorre uma //interrupção de alta prioridade

//em geral, o pragma vai permitir a activação de opções específicas de compilação //neste caso vai permitir que tenhamos interrupções disponíveis e subsequente gestão //das mesmas, algo que é de outra forma impossível usando apenas o C

```

//-----
// High priority interrupt vector

```

```

#pragma code InterruptVectorHigh = 0x08
void
InterruptVectorHigh (void)
{
    _asm
        goto InterruptHandlerHigh //jump to interrupt routine
    _endasm
}

```

//rotina que gere uma interrupção de alta prioridade

```

//-----
// High priority interrupt routine

```

```

#pragma code
#pragma interrupt InterruptHandlerHigh

void InterruptHandlerHigh ()
{
    if (INTCONbits.TMR0IF) //check for TMR0 overflow
    {
        INTCONbits.TMR0IF = 0; //clear interrupt flag
        Flags.Bit.Timeout = 1; //indicate timeout
        LATBbits.LATB0 = !LATBbits.LATB0; //toggle LED on RB0
    }

    if (PIR1bits.RCIF) //check RX interrupt
    {
        PIR1bits.RCIF = 0; //então mete a variável de interrupção a 0
        Flags.Bit.Rx = 1; //a flag Rx vai a 1
        inputstr[str_pos] = getcUSART(); //vai buscar a inputstr à porta série
        //caracter a caracter (em cada iterada)
    }
}

```

```
str_pos++; //incrementa a posição de escrita na
//string de destino
if(str_pos == 80){str_pos = 0;} //verifica se a string de 80 caracteres
//está cheia e caso o esteja faz reset da
//posição de escrita na string a 0
INTCONbits.PEIE=1; //permite interrupções feitas por
//periféricos
}
}
//-----
```

**Nota:** os fluxogramas podem ser consultados em anexo no final do documento.

Tendo em conta o estudo do método de comunicações usado, a porta série, resolveu-se então de acordo com as exigências do enunciado fazer uma pequena análise do mesmo.

A porta série do PC dispõe de um conjunto de *baud rates* (bits por segundo) *standard*. Desse conjunto escolhe-se um valor de acordo com as necessidades para uma dada aplicação.

Por sua vez, o CPU usado possui também um conjunto restrito de valores de *baud rate* possíveis que depende intrinsecamente do CPU usado bem como do próprio cristal oscilador utilizado. Após a consulta da documentação do CPU, chegou-se a uma fórmula que relaciona um parâmetro X de configuração do CPU (que é um inteiro) com a *baud rate* desejada. Naturalmente, se impusermos uma *baud rate* e obtermos o parâmetro X a partir desta, arredondamentos serão impostos neste factor (de forma a ser um inteiro) o que se traduz numa diferença entre a *baud rate standard* e a *baud rate real* do nosso CPU.

$$\left[ \frac{F_{osc}}{B} \times \frac{1}{16} \right] - 1 = X$$

Sabendo que o CPU utilizado avalia o valor lógico de um dado *bit* tendo em conta uma média ponderada em três momentos equidistantes dentro de um dado período, se a diferença entre a *baud rate* no PC e a aquela que temos aproximada no CPU for demasiada, o assincronismo pode gerar corrupção de dados no processo de transmissão de dados o que obviamente se pretende evitar a todo o custo.

Desta forma, utilizando a fórmula para calcular as *baud rates* do CPU a usar em cada um dos casos do conjunto *standard* de *baud rates*, assumiu-se um erro máximo de 33% (por causa do método de avaliação do valor lógico de cada *bit* acima explicado).

Desta forma construiu-se uma expressão que traduz o tempo máximo e mínimo para o envio de cada palavra de 8 bits. Seja T o período do sinal tem-se que:

$$P = 8T \pm \frac{T}{3}$$

A motivação para esta expressão é simples, ao fim de uma palavra de 8 *bits*, impomos um desvio máximo de T/3 no oitavo *bit* para garantirmos a viabilidade e a certeza de que não temos alterações no valor lógico lido na sequência. Com algumas operações simples é possível calcular as *baud rates* correspondentes.

Como a palavra tem 8 *bits*, divide-se a expressão anterior por 8 e fazendo o inverso da expressão obtida, pode-se então calcular o *baud rate* correspondente.

E construiu-se uma tabela com o conjunto *standard* das *baud rates* mais o desvio máximo (para cima e para baixo) permitido bem como o *baud rate* permitido pelo CPU e ver quando é que prevemos ter problemas.

Experimentalmente, verificou-se que existia corrupção na transmissão de dados para os valores previstos (*baud rates* superiores a 19200 *bits* por segundo a vermelho na tabela 1)

mas inesperadamente também para valores inferiores a 1200 *bits* por segundo (a laranja na tabela 1).

Após a consulta da documentação, verificou-se que ao impormos na configuração do protocolo, a utilização de *baud rates* elevadas através do parâmetro USART\_BRGH\_HIGH, que impossibilitamos a utilização de *baud rates* iguais ou inferiores a 300 *bits* por segundo tal como se verificou nos resultados experimentais.

Para acedermos a esses *baud rates*, seria necessário configurar o CPU através da *flag* indicada acima, para *baud rates* baixos.

BR std (b/s)	X arred	BR min (b/s)	BR max (b/s)	BR real (b/s)
110	2272	105.6	115	110
300	832	288	313	300
1200	207	1152	1252	1202
2400	103	2304	2504	2404
9600	25	9216	10017	9615
19200	12	18432	20035	19231
38400	6	36864	40070	35714
57600	3	55296	60104	62500
115200	1	110592	120209	125000
230400	0	221184	240417	250000
460800	0	442368	480835	250000
921600	0	884736	961670	250000

Tabela 1 – Valores teóricos e reais mais o desvio máximo e mínimo dos *baud rates* testados

Ao consultar os valores tabelados nas páginas 243 e 244 do manual, verificou-se também que os resultados obtidos fazem sentido dentro daquilo que é assegurado pelo fabricante.

## Comentários e Conclusões

A grande vantagem das comunicações assíncronas está explícita na sua natureza assíncrona. O facto do protocolo assíncrono nos permitir estabelecer comunicação entre dispositivos sem a necessidade de sincronizá-los aquando do envio de um pacote de dados, permite-nos poupar uma linha de dados que de outra forma teria obrigatoriamente que ser usada para sincronizar o *clock* dos dispositivos envolvidos no protocolo.

Como desvantagens, podemos notar que se pretendermos estabelecer comunicação entre vários dispositivos (não só dois como neste caso) utilizando um protocolo genericamente assíncrono, o controlo do erro pode ser mais complicado que no presente caso dependendo naturalmente da natureza das linhas e dispositivos envolvidos.

Outra desvantagem é possivelmente quando se pretende alta precisão no processo de comunicação como por exemplo num qualquer processo de aquisição em tempo real, em

que é fulcral assegurar que os dispositivos de envio e aquisição estão sincronizados de forma muito precisa.

Outra desvantagem que me parece possível é talvez ao trabalhar com palavras muito extensas porque o erro entre as *baud rates standard* e as *baud rates* reais é cumulativo e como tal cresce com o aumento do número de *bits*. Logo quanto maior for a palavra envolvida, mais cuidado teremos que ter ao efectuar a transferência de dados.

Olhando para a expressão de duração máxima e mínima de cada palavra e extrapolando-a para palavras maiores, é imediato que o factor de erro permitido se torna menor (em termos relativos) porque o factor que multiplica  $T$  aumenta.

BR std (b/s)	T std (s)	Factor X	BR real /b/s)	T real (s)	Mód. Assinc. 8º bit (s)	Mód. Assinc. Máx. (s)	BR min (b/s)	BR max (b/s)
110	9,091E-03	2272	109,99	9,09E-03	0,000008727	0,000378788	106	115
300	3,333E-03	832	300,12	3,33E-03	0,000010667	0,000138889	288	313
1200	8,333E-04	207	1201,92	8,32E-04	0,000010667	0,000034722	1152	1252
2400	4,167E-04	103	2403,85	4,16E-04	0,000005333	0,000017361	2304	2504
9600	1,042E-04	25	9615,38	1,04E-04	0,000001333	0,000004340	9216	10017
19200	5,208E-05	12	19230,77	5,20E-05	0,000000667	0,000002170	18432	20035
38400	2,604E-05	6	35714,29	2,80E-05	0,000015667	0,000001085	36864	40070
57600	1,736E-05	3	62500,00	1,60E-05	0,000010889	0,000000723	55296	60104
115200	8,681E-06	1	125000,00	8,00E-06	0,000005444	0,000000362	110592	120209
230400	4,340E-06	0	250000,00	4,00E-06	0,000002722	0,000000181	221184	240417

Tabela 2 – Valores teóricos e reais mais o desvio máximo e mínimo dos *band rates* mais valores temporais respectivos

Figura 1 – Fluxograma para a rotina principal do programa de exemplo facultado na aula

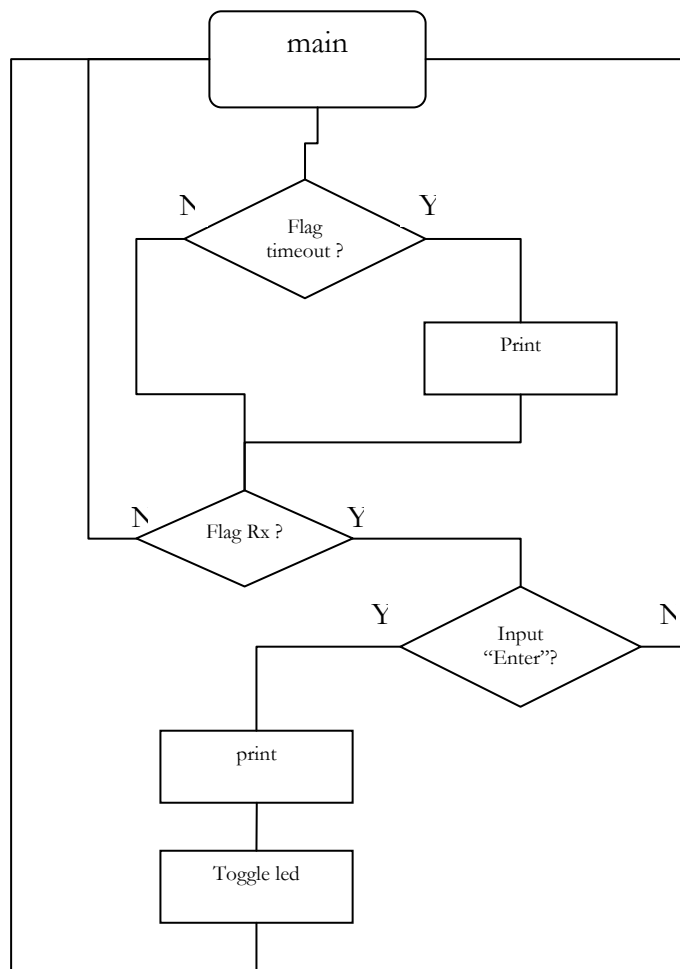


Figura 2 – Fluxograma para a rotina principal do programa realizado para cumprir os objectivos

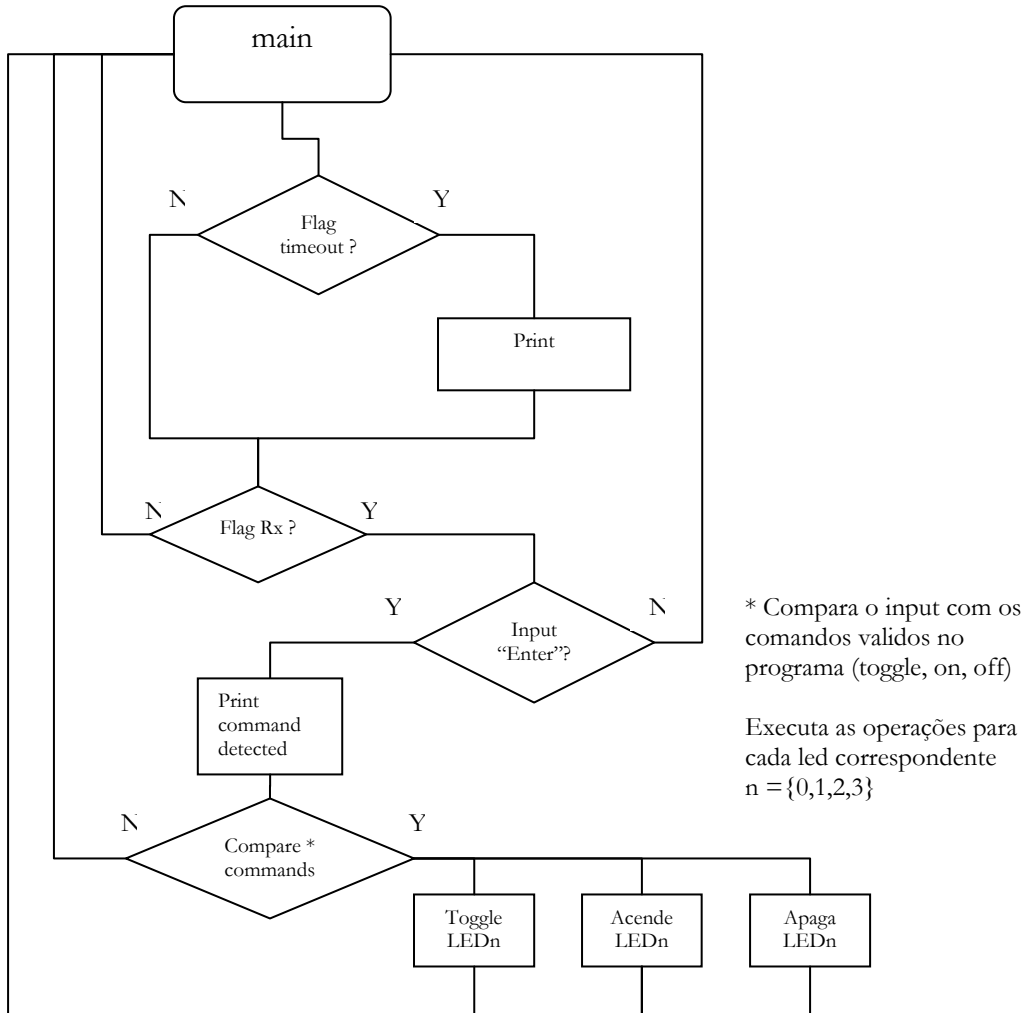


Figura 3 – Fluxograma do *interruption handler*, comum a ambos os programas

