

# Erros básicos

- Falta de conhecimento sobre o microcontrolador e assembly
- Falta de planejamento na hora de desenvolver o programa
- Estudar os passos que devem ser feitos (ponto de partida até o objetivo)
- Escrever algum trecho de programa sem pensar se o que está sendo escrito vai ser aplicável ou não
- Copiar um trexo de código de outra pessoa esperando que funcione no seu programa
- Uma coisa é não conhecer um método de solução, outra é não saber aplicar este método

Exemplo: Separar os nibbles de 1 byte em 2 registradores

# Serial sem interrupção

LOOP:

```
MOV    SBUF, #'A'  
JNB    TI, $  
CLR    TI  
SJMP   LOOP
```

# Frase sem interrupção

INICIO:

```
JNB     RI, $           ;Espera receber algo
CLR     RI
MOV     DPTR, #FRASE   ;Início da frase
```

VERIFICA:

```
CLR     A
MOVC    A, @A+DPTR
CJNE   A, #'$', ENVIA ;Compara com o fim da frase
SJMP   INICIO
```

ENVIA:

```
MOV     SBUF, A
INC     DPTR           ;Vai para o próximo caractere
JNB     TI, $         ;Espera terminar de enviar
CLR     TI
SJMP   VERIFICA
```

# Interrupção

- Interrupção não deve segurar a execução do programa (não ficar esperando por algo)
- O programa fora da interrupção deve o mais independente possível desta (melhor ficar esperando por nada do que pela interrupção)
- Interrupção indica a ocorrência de certo evento, com isso você jamais deve esperar por aquele evento fora dela
- Se seu programa utiliza somente 1 interrupção (serial ou timer por exemplo), **não há motivo pra apagar EA**

# Serial com Interrupção

INT\_SERIAL:

```
    CLR    TI
    MOV    SBUF, #'A'
    RETI
```

-----

PROG:

```
    MOV    SBUF, #'A'    ;Inicia o envio
    SJMP   $
```

Mas por quê se preocupar tanto assim?

A responsabilidade de controlar a serial é da interrupção da serial!

**Exemplo:** Receber e Transmitir qualquer caractere ASCII pelo canal serial a um taxa de 1200 BPS, utilizando a Interrupção.

Considerar o cristal da CPU de 11,0592 MHz.

```
FLAG    ORG 0
        EQU 30H                ; Definição de Flag de Atendimento de Interrupção
        SJMP PROG

        ORG 0023H              ; Sub-rotina de atendimento da Interrupção Serial
        CLR EA                  ; Desabilita as interrupções
        JNB TI,FIM              ; Verifica se foi Recepção
        CLR TI                  ; Sendo Transmissão, limpa o Flag TI de Transmissão
        SJMP FIM1              ; Retorna
FIM:    MOV A,SBUF              ; Lê um caractere Serial
        CLR RI                  ; Sendo Recepção, limpa o Flag RI de Recepção
FIM1:   SETB EA                 ; Re-abilita as interrupções e
        SETB FLAG               ; Ativa Flag de Atendimento de Interrupção
        RETI                    ; retorna

PROG:   SETB EA                 ; Habilitação de Interrupções
        SETB IE.4               ; Habilitação da Interrupção Serial
        MOV TMOD,#20H           ; TMOD = 00100000 - Timer 1 no Modo 2, controle por software
        MOV TH1,#232            ; valor 232 em TH1 e TL1 para gerar a Taxa de Comunicação de
        MOV TL1,#232            ; 1200 BPS com fc=11,0592 MHz e K=1(default)
        SETB TR1                ; Dispara Temporizador
        MOV SCON,#40h           ; SCON = 01000000 ? modo 1 do Canal Serial
        SETB REN                ; Habilita a Recepção
LOOP:   CLR FLAG                ; Zera o Flag de Atendimento de Interrupção
        JNB FLAG,$              ; Verifica se já foi atendida a Interrupção Serial
        MOV SBUF,A              ; Transmite o caractere recebido
        CLR FLAG                ; Zera o Flag de Atendimento de Interrupção
        JNB FLAG,$              ; Verifica se já foi atendida a Interrupção
        SJMP LOOP               ; Continua a comunicação Serial (Recebe/Transmite)
```

# Eco e Interrupção

## Sem Interrupção

PROG:

```
JNB RI, $  
CLR RI  
MOV SBUF, SBUF
```

```
JNB TI, $  
CLR TI  
  
SJMP PROG
```

## Com Interrupção

INT\_SERIAL:

```
JB RI, RECEBEU  
CLR TI  
SJMP FIM
```

RECEBEU:

```
CLR RI  
SETB FLAG1
```

FIM:

```
RETI
```

-----  
PROG:

```
JNB FLAG1, $  
CLR FLAG1  
MOV SBUF, SBUF  
  
SJMP PROG
```

# Eco com Interrupção (de verdade)

INT\_SERIAL:

```
    JB      RI, RECEBEU    ;Verifica se recebeu
    CLR     TI             ;Se transmitiu, apaga TI
    SJMP    FIM
```

RECEBEU:

```
    CLR     RI             ;Se recebeu apaga RI
    MOV     SBUF, SBUF     ;Ecoa o caractere
```

FIM:

```
    RETI                  ;Fim da interrupção
```



# Frases com interrupção

INT\_SERIAL:

JB TI, ENVIUO ;Verifica se enviou ou recebeu

RECEBEU:

CLR RI ;Apaga a flag de recepção  
MOV DPTR, #FRASE ;Início da frase  
MOV A, @A+DPTR ;Pega o primeiro caractere  
SJMP ENVIA

ENVIUO:

CLR TI ;Apaga a flag de transmissão  
CLR A  
MOV A, @A+DPTR ;Pega mais um caractere  
CJNE A, #'\$', ENVIA ;Verifica se chegou ao fim da frase  
RETI

ENVIA:

MOV SBUF, A ;Envia o caractere  
INC DPTR ;Vai para o próximo caractere  
RETI

# Mais sobre RET e RETI

Uma função/interrupção pode ter mais de 1 RET/RETI

Exemplo:

INT\_SERIAL:

```
    JB     RI, RECEBEU
    CLR    TI
    RETI
```

RECEBEU:

```
    CLR    RI
    MOV    SBUF, SBUF
    RETI
```

# Display de 7 segmentos

INICIO:

```
MOV    A, #EFH           ;1110 1111
MOV    R0, #30H          ;End. do primeiro dígito
```

LOOP:

```
MOV    P2, A             ;Seleciona o display
MOV    P0, @R0           ;Seleciona os segmentos
ACALL  ATRASO            ;Espera
RL     A                 ;Muda de display
INC    R0                ;Muda o valor
CJNE  R0, #34H, LOOP    ;Verifica se ja fez todos
SJMP  INICIO            ;Reinicia
```

Valores que aparecerão no display estão nos endereços 30h~33h  
Isso deixa facil a alteração destes valores (inclusive via serial)

# Rotinas

- Não tente usar uma função que você não sabe o que faz, provavelmente não irá funcionar no seu programa (programas prontos ou de colegas)
- É mais eficiente perguntar como uma função é feita e reproduzi-la do que procurar algo pronto
- Se você já tem uma determinada rotina para executar uma certa função, não esqueça dela
- Saiba como utilizar as funções que você criou
- Evite *stack overflow*, não chame uma função X dentro de uma função Y que também utiliza X

# Rotinas

Utilizadas quando é necessário realizar um mesmo procedimento várias vezes e em partes diferentes do programa.

Exemplo:

- Rotinas de inicialização

- Rotinas de atraso

- Rotinas de conversão

- Dividir um byte em 2 nibbles

Assim como em C, uma rotina em assembly pode ter argumentos

# Rotinas com argumentos

PROG:

```
MOV      A, #20H
ACALL    ROTINA    ;A=21H
MOV      A, #80H
ACALL    ROTINA    ;A=81H
```

ROTINA:

```
INC      A
RET
```

# Separar em 2 nibbles para o LCD

SEPARA:

```
MOV      R1, A           ;Salva em um registrador
ANL      A, #0F0H       ;Separa o nibble MS
MOV      R2, A           ;Salva em R2

MOV      A, R1           ;Pega o valor salvo
ANL      A, #0FH        ;Separa o nibble LS
MOV      R3, A           ;Guarda em R3

RET
```

R1	R2	R3
38H	30H	08H

# Separar em 2 nibbles para o LCD

SEPARA:

```
MOV      R1, A           ;Salva em um registrador
ANL      A, #0F0H       ;Separa o nibble MS
MOV      R2, A           ;Salva em R2

MOV      A, R1           ;Pega o valor salvo
ANL      A, #0FH        ;Separa o nibble LS
SWAP     A               ;Movimenta para o nibble MS
MOV      R3, A           ;Guarda em R3

RET
```

R1	R2	R3
38H	30H	80H



# Sobre as máscaras

Para zerar o nibble mais significativo para o LCD

```
CLR    P0.4
```

```
CLR    P0.5
```

```
CLR    P0.6
```

```
CLR    P0.7
```

Antes foi utilizada a operação AND para separar os nibbles de um byte, por quê não usar novamente pra zerar o nibble mais significativo de P0?

```
ANL    P0, #0FH
```

# Linguagem C

- Utilizada pra abstrair a linguagem de máquina
- Utilizando C, o microcontrolador não vai funcionar diferente de quando utilizando Assembly
- Mais fácil de estruturar e modular o código
- Facilita a utilização de funções
- Mais fácil de adaptar um código de um programa pra outro (ao utilizar variáveis, você se desprende dos registradores)
- Mais opções de controle de execução (loops, comparações)

# Linguagem C

Responsabilidades do programador:

- Comentar bem seu programa
- **Identar** o programa de forma correta
- Utilizar variáveis com nomes intuitivos
- Nome maior de variável não ocupa mais memória
- Gerenciar o uso de variáveis

# Comentários

Não é necessário comentar cada linha de código, uma breve descrição dos blocos chaves do programa é mais do que suficiente

Exemplo:

```
//Interrupção da serial
void intserial() __interrupt(4){
    //Recebeu
    if(RI==1){
        SBUF=SBUF;
        RI=0;
        return;
    }
    TI=0;
}
```

# Estrutura de código

```
if(SBUF<9 && SBUF>0 && var1==0){  
    var1=1;  
    if(SBUF%2==1)  
        a=1;  
    else  
        a=0;  
    var2=a;  
}
```

Nomes de variáveis não ajudam na identificação da sua função

**Identação** incorreta

# Estrutura de código

```
//É comando de movimento
if(SBUF<9 && SBUF>0 && controle==novo){
    controle=ligado;
    //Se o comando é ímpar (horário)
    if(SBUF%2==1)
        inicial=horario;
    //Se o comando é par (anti-horário)
    else
        inicial=anti;
    atual=inicial;
}
```

# Nomes para variáveis

Ponteiro para string: \*str

Ponteiro para vetor: \*ptr

Vetor de dados: vet[] dados[]

Um char genérico: ch op

Loop: i j k

Temporárias: temp

# Memória

O SDCC que faz o gerenciamento de memória do programa

Cuidados:

- Chamar função utiliza pilha
- Variáveis globais utilizam posições de memória permanentemente
- Um vetor é sempre contínuo na memória



# Variáveis e valores

O que é um char?

Uma variável de 8 bits (1 registrador)

O que é um int?

Uma variável de 16 bits (2 registradores)

Quais as diferenças entre um char e um int?

O tamanho: [-128;127] e [-32768; 32767]

Qual a diferença entre um short int e um long int?

O tamanho

O que é a tabela ASCII?

Uma relação entre uma letra ou símbolo com um número

# Variáveis e valores

O que é 'A' em C?

'A', independente do contexto terá o valor 65d de acordo com a tabela ASCII

Qual o valor de var em: int var='A';?

Tem o valor 65

Posso fazer contas com um char?

É uma variável, portanto sim

O que acontece se atingir um limite de uma variável?

A contagem passa para o lado oposto (ex: timer)

O que é o nome do vetor?

É equivalente ao endereço do primeiro elemento

# Variáveis e valores

//Supondo um vetor de 3 posições que começa em 20h  
char vetor[3]={1, 2, 3};

vetor[3]=0xFF; //Gera um warning

i=3;

vetor[i]=0xFF; //Não gera um warning

O que acontece no segundo caso?

20H	21H	22H	23H
01	02	03	255

# Variável global faz mal?

Não dentro de certos limites.

Variáveis globais ocuparão constantemente posições na memória, enquanto que variáveis locais ocupam registradores somente enquanto as funções estão em execução.

Variável local da main sempre estará alocada pois a main nunca termina!

# Variáveis globais

Quando é interessante utilizar variável global?

Enviar variáveis por parâmetro ou endereço pode ser útil em casos, mas atrapalhar em outros. Ao utilizar funções com argumentos, o **dobro de memória é ocupada** (salvo exceções). Uma variável global pode solucionar este problema.

O dobro de memória é ocupada pois há a variável local da função. O mesmo ocorre com passagem por parâmetro pois um ponteiro será alocado.

Vetores são exceção pois é necessário somente um ponteiro na função.

# Memória

Internal RAM layout:

```
   0 1 2 3 4 5 6 7 8 9 A B C D E F
0x00: |0|0|0|0|0|0|0|0|a|a|a|a|a|a|a|a|
0x10: |a|a|Q|Q| | | | | | | | | | | |
0x20: |B|T|I|I|I|I|I|I|I|I|I|I|I|I|I|I|
0x30: |I|I|I|I|I|I|I|I|I|I|I|I|I|I|I|I|
0x40: |I|I|I|I|I|I|I|I|I|I|I|I|I|I|I|I|
0x50: |I|I|I|I|I|I|I|I|I|I|I|I|I|I|I|I|
0x60: |I|I|I|I|I|I|I|I|I|I|I|I|I|I|I|I|
0x70: |I|I|I|I|I|I|I|I|I|I|I|I|I|I|I|I|
0x80: |I|I|I|I|I|I|I|I|I|I|I|I|I|I|I|I|
0x90: |I|I|I|I|I|I|I|I|I|I|I|I|I|I|I|I|
0xa0: |I|I|I|I|I|I|I|I|I|I|I|I|I|I|I|I|
0xb0: |I|I|I|I|I|I|I|I|I|I|I|I|I|I|I|I|
0xc0: |I|I|I|I|I|I|I|I|I|I|I|I|I|I|I|I|
0xd0: |I|I|I|I|I|I|I|I|I|I|I|I|I|I|I|I|
0xe0: |I|I|I|I|I|I|I|I|I|I|S|S|S|S|S|S|
0xf0: |S|S|S|S|S|S|S|S|S|S|S|S|S|S|S|S|
```

0-3:Reg Banks, T:Bit regs, a-z:Data, B:Bits, Q:Overlay, I:iData, S:Stack, A:Absolute

Stack starts at: 0xea (sp set to 0xe9) with 22 bytes available.

Other memory:

Name	Start	End	Size	Max
PAGED EXT. RAM			0	256
EXTERNAL RAM			0	65536
ROM/EPROM/FLASH	0x0000	0x07ab	1964	65536

# E GOTO, faz mal?

GOTO em C é o equivalente a SJMP em assembly. Em assembly, era um comando muito útil e inevitável, mas e em C?

GOTO pode ser usado se bem cuidadoso. É difícil tanto pro programador quanto para outra pessoa seguir a execução de um programa com GOTO. Além disso, a linguagem C permite que você faça isso de outras maneiras

# “Pulos” em C

Em C, pulos são feitos automaticamente com as funções tradicionais como if-else, for, while, break, continue, etc...

Além disso, as funções RET e RETI de assembly correspondem a return em C (com ou sem um valor)

Assim como em assembly, nunca pule pra fora de uma função em C



# Return em C

```
void int_serial(void) __interrupt(4){
    //Enviou
    if(TI){
        TI=0;
        return;
    }
    //Recebeu
    else{
        RI=0;
        SBUF=SBUF; //Ecoa
        return;
    }
}
```

# Registradores

Evitar a utilização de registradores do microcontrolador

```
if(RI){ //Recebeu  
    A=SBUF;  
    if(A=='0')  
    ...
```

Pode ser substituído por:

```
if(RI){ //Recebeu  
    if(SBUF=='0')  
    ...
```

# Registradores

Outra opção é criar uma variável local ou global

```
char ch;
```

```
void serial(void) __interrupt(4){  
    if(RI==1){ //Recebeu  
        ch=SBUF;  
        ...  
    }  
}
```