



SEL0337 – Aplicação de Microprocessadores II

Revisão da Linguagem C

Prof. Evandro L. L. Rodrigues
Colaboração: Pedro Ivo de Castro Oyama

Estrutura de um programa C

- Diretivas de pré-processamento
- Declaração de variáveis globais
- Declaração de protótipos de funções
- Definições das funções
- Programa principal

Exemplo de um programa C

```
#include <at89x52.h> //diretiva de pré-processamento

int x,y; //declaração de variáveis globais

int soma(int, int); //declaração de protótipo de função

void main() //Programa principal
{
    x = 10;
    y = 20;
    x = soma(x,y);
}

int soma(int a, int b) // definição de função
{
    return a+b;
}
```

Diretivas de pré-processamento

- Instruções a serem executadas antes da compilação, modificando o código a ser compilado.

- Principais diretivas:

#define

Sintaxe:

#define **TEXT01** **texto2**

Função:

Qualquer ocorrência de **TEXT01** no código será substituída por **texto2**.

Ex:

#define **PI** **3.1415**

Diretivas de pré-processamento

- **#include**

Sintaxe:

`#include <arquivo>` OU `#include "arquivo"`

Função:

Adiciona ao código o conteúdo de **arquivo**.

Quando entre <> o **arquivo** é procurado no diretório de bibliotecas do compilador. Quando entre " " o **arquivo** é procurado no diretório do arquivo a ser compilado.

Ex:

`#include <math.h>`

Declaração de variáveis

- Sintaxe:
`tipo nome;` OU `tipo nome = valor_inicial;`
Ex: `int ano = 2012;`

Tipo	Bytes	Valor mínimo	Valor máximo
char	1	-168	167
short	2	-32,768	+32,767
int	2	-32,768	+32,767
long	4	-2,147,483,648	+2,147,483,647
float	4	-	-

- O modificador `unsigned` pode ser adicionado ao tipo para utilizar somente valores positivos, dobrando seu o máximo valor.
Ex: `unsigned char` (permite valores entre 0 e 255)

Representação de Valores

- Valores Numéricos

Decimal: 10

Binário: **0b**1010

Octal: **0**12

Hexadecimal: **0xA**

- Caracteres são representados entre aspas simples ('').

Ex: `char letra_a = 'a';`

Definição de True e False

- C não conta com um tipo booleano, assim os conceitos de True e False da lógica booleana são definidos como segue:
- False: 0
- True : Qualquer valor diferente de 0
- Assim, um loop infinito pode ser feito, por exemplo, com a instrução “while(1){ ... }”

Arranjos

- Arranjos são um conjunto de variáveis do mesmo tipo
- Um arranjo unidimensional é declarado da seguinte forma:
`tipo nome[número_de_elementos];`
- Exemplo:
`int conjunto_1[10]; //Declara um arranjo de 10 inteiros`

Arranjos

- Elementos de um arranjo são acessados pelo operador índice [], como a seguir:
b[1] = 5;
a = b[0];
- Os índices de um arranjo de N elementos variam de 0 a N-1;

Arranjos

- Arranjos podem ser multidimensionais:

```
float arranjo_1[5][10];
```

```
int arranjo_2[2][5][9];
```

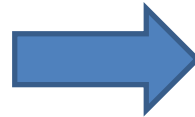
Operadores Aritméticos

Nome	Sintaxe	Exemplo	Valor de c
Atribuição	$a = b$	<code>c = 10;</code>	10
Soma	$a + b$	<code>c = 1+2;</code>	3
Subtração	$a - b$	<code>c = 3 - 4;</code>	-1
Multiplicação	$a * b$	<code>c = 3 * 2;</code>	6
Divisão	a / b	<code>c = 8 / 2;</code>	4
Módulo (Resto da divisão)	$a \% b$	<code>c = 5 \% 2</code>	1
Incremento	<code>a++</code>	<code>c = 1;</code> <code>c++;</code>	2
Decremento	<code>b--</code>	<code>c = 1;</code> <code>c--;</code>	0

Divisão de inteiro

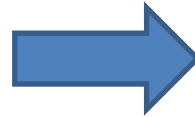
- **ATENÇÃO:** A divisão de um número inteiro retorna um inteiro.

```
float a = 7 / 2;
```



```
a = 3
```

```
int b = 5;  
float a = b / 2;
```



```
a = 2
```

- **Solução:**

```
float a = 7.0 / 2;
```



```
a = 3.5
```

```
int b = 5;  
float a = (float) (b) / 2;
```



```
a = 2.5
```

Operadores Aritméticos

Nome	Sintaxe	Equivalência	Exemplo	Valor de c
Atribuição por soma	<code>a += b</code>	<code>a = a + b</code>	<code>c = 1;</code> <code>c += 2;</code>	3
Atribuição por subtração	<code>a -= b</code>	<code>a = a - b</code>	<code>c = 3;</code> <code>c -= 4;</code>	-1
Atribuição por multiplicação	<code>a *= b</code>	<code>a = a * b</code>	<code>c = 3;</code> <code>c *= 2;</code>	6
Atribuição por divisão	<code>a /= b</code>	<code>a = a / b</code>	<code>c = 8;</code> <code>c /= 2;</code>	4
Atribuição por módulo	<code>a % b</code>	<code>a = a % b</code>	<code>c = 5;</code> <code>c %= 2</code>	1

Operadores de Comparação

Nome	Sintaxe
Igual a	<code>a == b</code>
Diferente de	<code>a != b</code>
Maior que	<code>a > b</code>
Maior ou igual a	<code>a >= b</code>
Menor que	<code>a < b</code>
Menor ou igual a	<code>a <= b</code>

ATENÇÃO!

Não confundir o operador de igualdade (`==`) com o de atribuição (`=`)

A instrução `“if (x = 10)”` não é um erro de sintaxe, e portanto será compilada sem erro.

Operadores Lógicos

Nome	Sintaxe
Operador NOT	!a
Operador OR	a b
Operador AND	a && b

Permitem a concatenação de expressões comparativas

EX:

```
if ( x == 5 || y > 34 )  
{  
    ....  
}
```

Se x for igual a 5 ou
se y for maior que 34

```
if ( !(x > 10 && x < 20) )  
{  
    ....  
}
```

Se x não estiver entre 10
e 20 (não inclusivo)

Operadores de Bit

Nome	Sintaxe	Exemplo	Valor de c
Operador NOT bit-a-bit	$\sim a$	a = 0b1100 c = $\sim a$;	0b0011
Operador AND bit-a-bit	a & b	a = 0b1100; b = 0b1010; c = a&b;	0b1000
Operador OR bit-a-bit	a b	a = 0b1100; b = 0b1010; c = a b;	0b1110
Operador XOR bit-a-bit	a ^ b	a = 0b1100; b = 0b1010; c = a^b;	0b0110
Deslocamento à esquerda (desloca b bits de a)	a << b	a = 0b010011; c = a << 2	0b001100
Deslocamento à direita (desloca b bits de a)	a >> b	a = 0b010011; c = a >> 2	0b000100

Instruções condicionais - if

Sintaxe

```
if ( condição )
{
    //conjunto de instruções 1
}
else
{
    //conjunto de instruções 2
}
```

- O conjunto de instruções 1 só é executado se a **condição** for satisfeita
- Se a **condição** não for satisfeita o conjunto de instruções 2 é executado
- O bloco **else{ ... }** pode ser omitido

Exemplo

```
if ( x % 2 == 0 )
{
    x_eh_par = 1;
}
else
{
    x_eh_par = 0;
}
```

Instruções condicionais - switch

- Instrução que avalia uma variável e executa diferentes conjuntos de instruções de acordo com seu valor.

Sintaxe

```
switch ( variável )
{
  case valor_1 :
      //conjunto de instruções 1
      break;

  case valor_2 :
      //conjunto de instruções 2
      break;

  ...

  default:
      //conjunto de instruções default
}
```

- Se a **variável** apresentar o valor **valor_1** o conjunto de instruções 1 é executado
- Se a **variável** apresentar o valor **valor_2** o conjunto de instruções 2 é executado
- Podem ser definidos quantos casos forem necessários
- Se **variável** não apresentar nenhum dos valores definidos anteriormente, o conjunto de instruções **default** é executado (esse bloco é opcional).

Instruções condicionais - switch

Exemplo

```
switch ( x )
{
case 1 :
    y = 10;
    z = 5;
    break;

case 2 :
    y = 50;
    z = 30;
    break;

default:
    y = z = 0;
}
```

Instruções de repetição- while

Sintaxe

```
while (condição de execução)  
{  
    bloco de repetição  
}
```

- A instrução consiste dos seguintes passos:
 - 1) É verificada a **condição de execução**. Se esta for atendida o passo 2 é executado. Senão, o loop é terminado.
 - 2) As instruções contidas no **bloco de repetição** são executadas.
 - 3) Volta-se ao passo 1.

Exemplo

```
int x = 1;  
while( x < 10)  
{  
    x *= 2;  
}
```

Instruções de repetição- for

Sintaxe

```
for ( inicialização; condição de execução ; fim de iteração )  
{  
    bloco de repetição  
}
```

- A instrução consiste dos seguintes passos:
 - 1) São executadas as instruções contidas na **inicialização**.
 - 2) É verificada a **condição de execução**. Se esta for atendida o passo 3 é executado. Senão, o loop é terminado.
 - 3) As instruções contidas no **bloco de repetição** são executadas.
 - 4) As instruções de **fim de iteração** são executadas.
 - 5) Volta-se ao passo 2.

Instruções de repetição- for

Exemplo – Repetição de 10 iterações

```
int x;  
for ( x = 0; x < 10; x++)  
{  
    k++;  
}
```

- Os blocos de **inicialização** e **fim de iteração** podem conter mais de uma instrução. Elas devem ser separadas por vírgula.

Exemplo

```
int x, y;  
for ( x = 0, y = 10 ; x < 10 ; x++, y+= 10)  
{  
    k += x * y;  
}
```

Instruções para controle de loop

- `break`

Termina o loop.

- `continue`

Termina a iteração e executa a próxima (se a condição de execução for satisfeita).

Funções

- Bloco de código que executa uma determinada tarefa, e pode ser chamado em qualquer parte do programa.
- Podem receber parâmetros necessários para seu processamento.
- Podem retornar um valor.

Definindo uma Função

Sintaxe

```
tipo_de_retorno nome_da_função ( lista de parâmetros )  
{  
    bloco de instruções  
}
```

- Cada parâmetro é definido pela seguinte sintaxe:
`tipo nome_do_parâmetro`
- Parâmetros são separados por vírgula.
- Se nenhum valor for retornado o `tipo_de_retorno` é `void`
- Para retornar um valor a instrução `return` seguida do valor a ser retornado deve ser utilizada.

Funções - Exemplos

Exemplo

```
int soma ( int a, int b )  
{  
    return (a+b);  
}
```

Exemplo

```
void delay ( )  
{  
    int t;  
    for( t = 0; t < 1000 ; t++ );  
}
```

Chamando funções

Exemplo

```
void main()
{
    a = soma(1, 2);
}
```

Exemplo

```
void main()
{
    a = 5;
    b = 3;
    c = soma(a, b);
}
```

Exemplo

```
void main()
{
    delay();
}
```

Passagem de parâmetros por valor

- Na definição convencional de funções a passagem de parâmetros é por valor.
- Os parâmetros passados a uma função não são modificados por ela.

Exemplo

```
int soma ( int a, int b )
{
    a += b;
    return a;
}

void main()
{
    int t = 10;
    int s;
    s = soma(t, 2);
}
```

Após a chamada da função soma, t tem o valor de 10.

Passagem de parâmetros por referência

Para as modificações feitas em parâmetros passados para uma função continuarem após o término da sua execução, deve-se utilizar a passagem de valor por referência, o que exige o uso de ponteiros.

Ponteiros

- Variáveis que guardam endereços de memória de outras variáveis.
- Sintaxe para a declaração:
`tipo *nome_do_ponteiro;`

Declara um ponteiro que aponta para uma variável do `tipo` especificado.

Ex: `int *ptr;`

Ponteiros

- Para atribuir o endereço de uma variável a um ponteiro utiliza-se o operador **&**, o qual retorna o endereço de uma variável.

Exemplo

```
int a;  
int *int_ptr;  
int_ptr = &a;
```

O ponteiro **int_ptr** passa a guardar o endereço de **a**, ou seja, **int_ptr** passa a apontar para **a**;

Ponteiros

- Para acessar o valor de variáveis apontadas por um ponteiro utiliza-se o operador `*`.

Exemplo

```
int a, b;  
int *int_ptr;  
int_ptr = &a;  
*int_ptr = 10;  
b = *int_ptr + 10;
```

A variável **a** passa a ter o valor 10 e **b** o valor de 20.

Passagem de parâmetros por referência

- São utilizados ponteiros como parâmetros.

Exemplo

```
void troca ( int *a, int *b )
{
    int aux = *a;
    *a = *b;
    *b = aux;
}

void main()
{
    int x = 10;
    int y = 20;
    troca( &x, &y);
}
```

Após a chamada da função troca, **x** tem o valor de 20 e **y** tem o valor de 10.

Strings - Arranjos de char

- Uma cadeia de caracteres, ou String, pode ser declarada e inicializada utilizando-se arranjos de char:

```
char String1[] = "exemplo";
```

- O último caracter de uma String declarada desta forma é sempre `'\0'`.
- Portanto, **String1** é um arranjo de 8 elementos.

Strings - Ponteiros

- Também podem ser utilizados ponteiros para declarar e inicializar uma String:

```
char *String2 = "exemplo";
```

- O último caracter de **String2** também é '\0'.
- Os caracteres podem ser acessados com o operador **[]** ou *****.

String - Ponteiros

- O 2º character, por exemplo, pode ser acessado das seguintes formas:

```
aux = String2[1];
```

```
aux = *(String2+1);
```

- Utilizando-se ponteiros os valores da String não podem ser alterados.

Aspectos específicos de C no SDCC

Biblioteca at89x52

- É necessário incluir a biblioteca at89x52:
`#include <at89x52.h>`
- Responsável por permitir o acesso a recursos do microcontrolador.

Acesso aos SFRs

- Os SFRs podem ser acessados diretamente pelos seus nomes (em letras maiúsculas).
- Seus bits são acessados acrescentando `_n` ao nome do SFR, onde `n` é a posição do bit

Ex:

```
P1 = 0xA1;
```

```
P1_0 = 1;
```

```
P2_5 = 0;
```

```
TR1 = 1;
```

```
incoming_char = SBUF;
```


Definindo SFRs

- `__sfr nome;`

Exemplo

```
__sfr PORTA2;  
PORTA2 = P2;  
PORTA2 = 0xA0;
```

- `__sfr __at endereço nome;`

Exemplo

```
__sfr __at 0x90 PORTA1;  
PORTA1 = 0xFF;
```

Definindo bits de SFRs

- `__bit nome;`

Exemplo

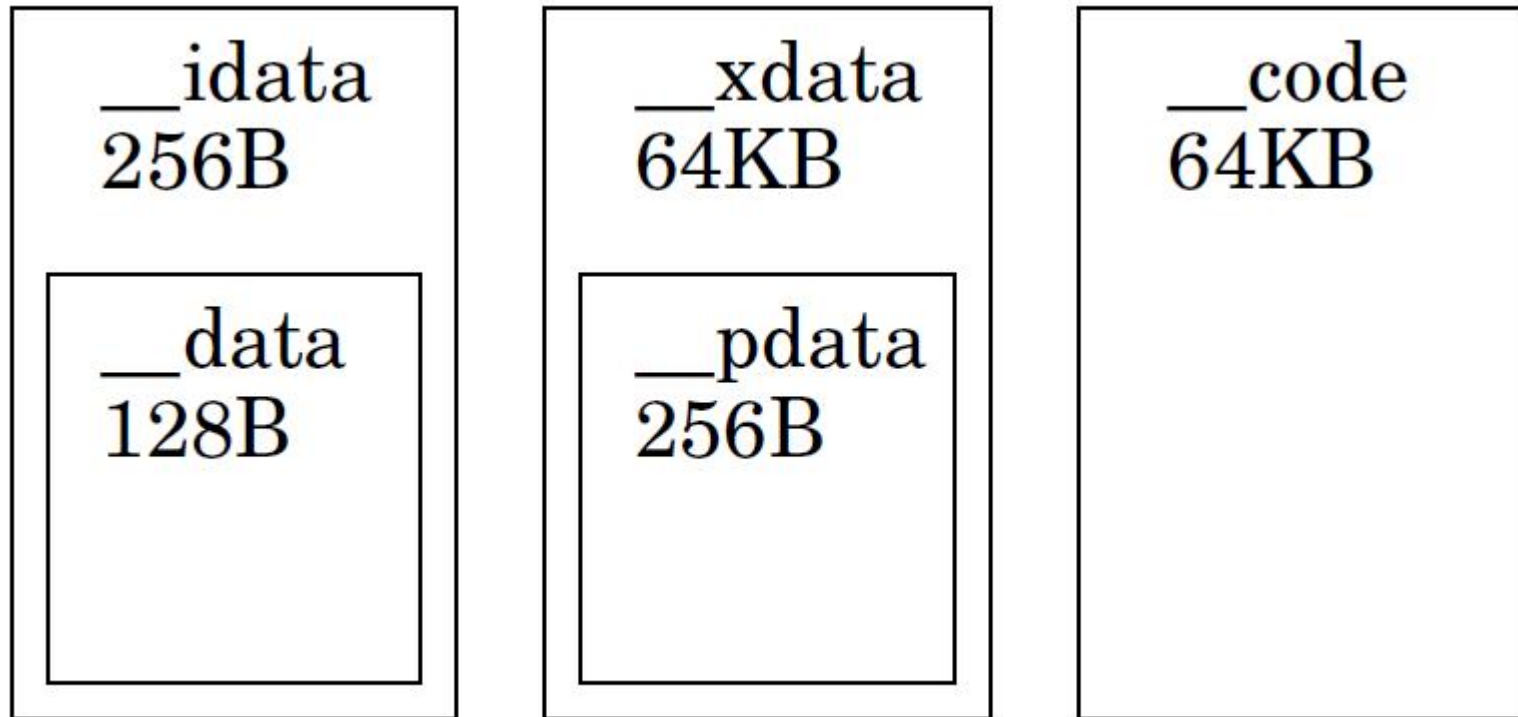
```
__bit BIT2_1;  
BIT2_1 = P2_1;  
BIT2_1 = 0;
```

- `__bit __at endereço nome;`

Exemplo

```
__bit __at 0x95 BIT1_5;  
BIT1_5 = 1;
```

Espaço de endereçamento



Declarando constantes na memória de programa

- Utiliza-se a palavra chave `__code` na declaração das constantes.

Ex:

```
__code unsigned char dado = 0x50;  
__code char frase[] = {'t', 'e', 's', 't', 'e'};  
__code char *frase2 = "exemplo";
```

Declarando constantes na memória de programa

- Strings podem ser gravadas na memória de programa utilizando-se a diretiva **#define**

Ex:

```
#define frase3 "exemplo"
```

Assim, a String **"exemplo"** pode ser acessada através do nome **frase3** como se este fosse um ponteiro e, portanto, seus caracteres podem ser acessados através dos operadores [] e *.

Ex:

```
aux = *(frase3 + 3);    //aux = 'm'
```

Utilizando código Assembly

- Um trecho de código Assembly pode ser inserido em um código C. Para isso ele deve estar entre as palavras chaves `__asm` e `__endasm`;

Exemplo

```
void delay_asm()
{
    __asm
    mov r0 #0xFF
00001$: djnz r0, 00001$
    __endasm;
}
```

Os labels devem ter o formato **nnnnn\$**, onde **nnnnn** é um número menor que 100, o que limita o número de labels por função a 100.

Interrupções

- Para se definir uma função para tratar uma interrupção utiliza-se a seguinte sintaxe:

```
void nome (void) __interrupt (prioridade)
```

Interrupção	Prioridade
External Interrupt 0	0
Timer 0 Interrupt	1
External Interrupt 1	2
Timer 1 Interrupt	3
UART Interrupt	4

Interrupções - Exemplo

Exemplo

```
void recepcao_serial() __interrupt(4)
{
    if( RI )
    {
        RI = 0;
        incoming_char = SBUF;
    }
}
```


Variáveis volatile

ATENÇÃO:

Variáveis modificadas por rotinas de interrupção que são acessadas por outras partes do programa devem ser declaradas **volatile**.

Variáveis volatile - Exemplo

Exemplo

```
volatile unsigned char incoming_char = '\0';
```

```
void main
```

```
{
```

```
    while (incoming_char != 'E')
```

```
    {
```

```
        do_something();
```

```
    }
```

```
}
```

```
void recepcao_serial() __interrupt(4)
```

```
{
```

```
    if( RI )
```

```
    {
```

```
        RI = 0;
```

```
        incoming_char = SBUF;
```

```
    }
```

```
}
```

Desabilitando Interrupções

- Utilizando a palavra-chave `__critical` as interrupções serão desativadas durante a execução de um bloco de código ou de uma função.

Exemplo

```
void funcao_critica() __critical
{
    ...
}
```

Exemplo

```
__critical
{
    ....
}
```

FIM