

Livro: Microcontrolador 8051 com Linguagem C

Prático e Didático

Família AT89S8252 Atmel

Denys E. C. Nicolosi Rodrigo B. Bronzeri
Editora Érica

A Linguagem C é “case sensitive”

SOMA, soma, SoMa, Soma, SomA

Palavras Reservadas

auto	break	case	char	const
continue	default	do	double	else
enum	extern	float	for	goto
if	int	long	register	return
short	signed	sizeof	static	struct
switch	typedef	union	unsigned	void
volatile	while			

compilador Cx51 (SDCC): novas palavras reservadas do C:

at	alien	bdata	bit	code
compact	data	far	idata	interrupt
large	pdata	_priority_	reentrant	sbit
sfr	sfr16	small	_task_	using
xdata				

Comentários

/*

TUDO QUE ESTIVER ENTRE OS MARCADORES DE COMENTÁRIO
É IGNORADO PELO COMPILADOR

*/

Um programa em C

```
/******  
*****  
Função: main  
Descrição: Mostrar a estrutura e o funcionamento de um programa  
em C  
Parametros: nenhum  
Retorno: nenhum  
Resultados: Coloca P1 em nível baixo (0)  
*****  
*****/
```

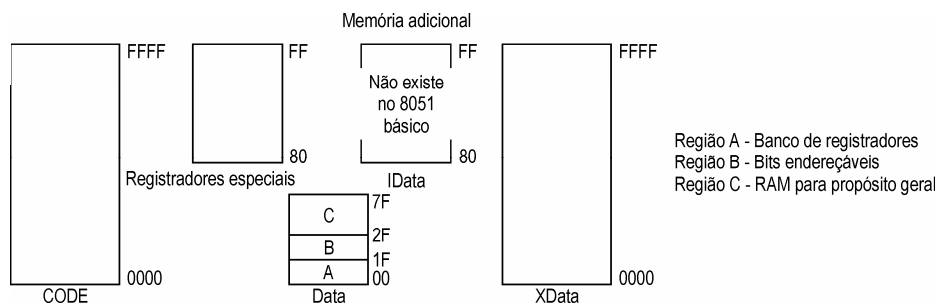
```
#include <reg51.h>
```

```
char dado;
```

```
void main(void)
```

```
{  
    dado = 0;  
    P1 = dado;  
}
```

Organização da Memória



2.2. Variáveis e Tipos de Dados Simples

Tipo de Dado	Número de Bits	Número de Bytes	Alcance (Decimal)
char	8 bits	1 bytes	-128 a +127
short e int	16 bits	2 bytes	-32768 a +32767
long	32 bits	4 bytes	-2147483648 a +2147483647
float	32 bits	4 bytes	3,4E-38 a 3,4E+38

Tipo de Dado	Número de Bits	Número de Bytes	Alcance (Decimal)
<i>unsigned char</i>	8 bits	1 bytes	0 a +255
<i>unsigned short e unsigned int</i>	16 bits	2 bytes	0 a +65535
<i>unsigned long</i>	32 bits	4 bytes	0 a +4294967294

Compiladores C

- 1- www.microcontrolador.com.br -> downloads -> software e procure: Ambiente de desenvolvimento com compilador C, Assembler, simulador e gravadores para os Kits (ambiente Windows 9x/2000/XP).
- 2- www.google.com.br -> procurar "SDCC compiler" e fazer download.
- 3- Usar o compilador do lab da FEI.
- 4- www.keilsoftware.com -> procurar compilador 8051 download (evaluation software).

2.3. Escolhendo a Região de Armazenamento dos Dados

dados na região IDATA, basta declará-la da seguinte forma:

idata unsigned int var; /* A correta leitura dessa declaração é: variável cujo nome é "var" que armazenará um dado do tipo "int" (2 bytes), que por ser antecedida pelo modificador "unsigned", possui uma alcance que vai de 0 a 65535 em decimal, e esse dado será armazenado na região IDATA. */

dados na região XDATA, basta declará-la da seguinte forma:

xdata unsigned char var; /* A correta leitura dessa declaração é: variável cujo nome é "var" que armazenará um dado do tipo "char" (1 byte), que por ser antecedida pelo modificador "unsigned", possui uma alcance que vai de 0 a 255 em decimal, e esse dado será armazenado na região xdata, que se refere à memória EXTERNA. */

dados na região DATA, basta declará-la da seguinte forma:

data unsigned char var; /* A correta leitura dessa declaração é: variável cujo nome é "var" que armazenará um dado do tipo "char" (1 byte), que por

ser antecedida pelo modificador "unsigned", possui um alcance que vai de 0 a 255 em decimal, e esse dado será armazenado na região DATA. */

Essa região DATA é considerada "default". O que significa que não havendo nenhuma escolha na declaração de uma variável, os dados são automaticamente armazenados na região DATA.

Portanto, a declaração a seguir indica também os dados na região DATA:

```
unsigned char var; /* A correta leitura dessa declaração é: variável cujo nome é "var" que armazenará um dado do tipo "char" (1 byte), que por ser antecedida pelo modificador "unsigned", possui um alcance que vai de 0 a 255 em decimal, e esse dado será armazenado na região DATA. */
```

Existe também a possibilidade de criarmos tabelas de dados apenas para a leitura. Neste caso, podemos deixá-los na memória ROM (CODE).

A declaração para um dado que ficará armazenado na região de memória CODE, servindo apenas para a leitura, é:

```
code unsigned char var; /* A correta leitura dessa declaração é: variável cujo nome é "var" que armazenará um dado do tipo "char" (1 byte), que por ser antecedida pelo modificador "unsigned", possui um alcance que vai de 0 a 255 em decimal, e esse dado será armazenado na região CODE. */
```

Por fim, o compilador possibilita que criemos variáveis que comportem apenas 1 bit.

Essa variável é armazenada na parte B da região DATA da memória RAM interna e é de extrema importância no que se refere à economia de memória, sendo sua declaração mostrada em seguida:

```
bit var; /* A correta leitura dessa declaração é: variável cujo nome é "var" que armazenará um 1bit na parte B da região de memória RAM INTERNA. */
```

2.5. Iniciando Variáveis

```
#include<at89s8252.h>
idata unsigned int a = 10;
void main (void)
{
    unsigned int b = 100;
}
```

Outro exemplo:

```
#include<at89s8252.h>
idata unsigned int a = 10;
void main (void)
{
    unsigned int b = 100;
    a++; /* acréscimo em uma unidade
a=11, pois o comando a++ quer
dizer a=a+1 */
}
```

2.6. A Relação entre Tipos de Dado Simples

```
#include<at89s8252.h>
idata unsigned int a = 10;
void main (void)
{
    unsigned int b = 100;
    b = a;    /* b=10 */
}
```

	Variável "a"	Variável "b"
Valor inicial	10	100
Valor após atribuição	10	10

2.7. Tipos de Dados Compostos - Matrizes

char nome_da_variavel [2]; /* Declaração de uma matriz para duas variáveis do tipo char (1 byte). */

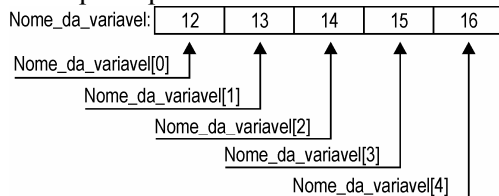
Para armazenarmos dados com essas duas variáveis, temos:


```
#include<at89s8252.h>
void main (void)
{
    char nome_da_variavel[2];
    nome_da_variavel[0] = 12;
    nome_da_variavel[1] = 13;
}
```

Agora temos uma matriz composta por cinco variáveis do tipo "char".

```
#include<at89s8252.h>
void main (void)
{
    char nome_da_variavel[5] = {12,13,14,15,16};
}
```

Para que fique mais claro como são armazenados os dados, observe o esquema da figura



 O quadrado nesse exemplo representa um espaço de memória de 8 Bits (1 byte)

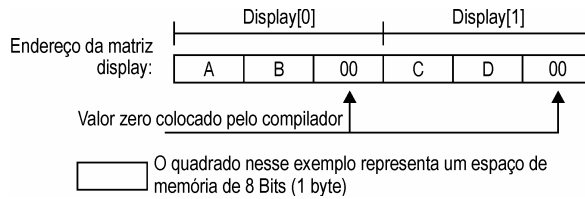
escrever em um display as letras "A" e "B"

```
#include<at89s8252.h>
void main (void)
{
    char display[2] = {"A","B"};
}
```

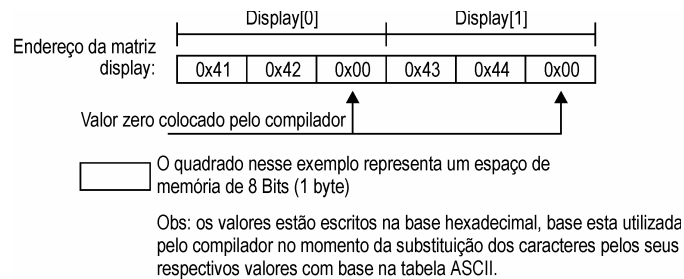
Caso quiséssemos escrever "AB" e "CD"

```
#include<at89s8252.h>
void main (void)
{
    char display[2][3] = {"AB","CD"};
}
```

O mapa da área de memória em que haveria o armazenamento dos dados é representado pela figura.



Na realidade, o verdadeiro mapa da memória é mostrado na figura.



2.8. Controladores de Fluxo

"controladores de loop" e "controladores de decisão"

2.9. Operadores

2.9.1. Precedência dos Operadores

calcular a soma de 3 com 7, depois dividirmos o resultado por 2 e em seguida multiplicarmos por 10

```
#include<at89s8252.h>
void main( )
{
    int a=3+7/2*10;
}
/* a = 33, porque?*/
```

Operador	Descrição	Grupo	Avaliação
{ } :	Delimitador de bloco Quantidade de bits	Especial Especial	
() [] . ->	Indicador de função Indicador de elemento de uma matriz Membro da união/estr Apontador para membro de união/estr	Especial Especial Especial Especial	E → D
-- ++	Decremento Incremento	Aritmético Aritmético	D → E
:	Operador de Base	Ponteiro	E → D
! ~ - + & * sizeof() (tipo)	Lógico NOT Complemento bit a bit Negativo Positivo O endereço de No endereço apontado por Quantidade de bytes Cast do tipo dado	Lógico Bit a bit Aritmético Aritmético Ponteiro Ponteiro Especial Especial	D → E
* / %	Multiplicação Divisão Módulo de dois números (resto)	Aritmético Aritmético Aritmético	E → D
- +	Subtração Adição	Aritmético Aritmético	E → D
<< >>	Desloca bits para a esquerda Desloca bits para a direita	Bit a bit Bit a bit	E → D
< <= > =>	Menor que Menor ou igual Maior que Maior ou igual	Relacional Relaciona Relaciona Relaciona	E → D
== !=	Igualdade Diferente	Relaciona Relaciona	E → D
&	AND bit a bit	Bit a bit	E → D
^	OR exclusivo bit a bit	Bit a bit	E → D
	OR bit a bit	Bit a bit	E → D
&&	Lógico AND	Lógico	E → D
	Lógico OR	Lógico	E → D
? :	Condiciona - Similar à IF-ELSE que será visto futuramente	Condiciona	D → E
= *= /=	Atribuição Multiplica atribuindo Divide atribuindo	Aritmético Aritmético Aritmético	D → E
%= += -=	Resto atribuindo Soma atribuindo Subtrai atribuindo	Aritmético Aritmético Aritmético	
<<= >>=	Bits para esquerda atribuindo Bits para direita atribuindo	Bit a bit Bit a bit	
&= ^= =	AND bit a bit, atribuindo OR exclusivo bit a bit, atribuindo OR bit a bit, atribuindo	Bit a bit Bit a bit Bit a bit	
,	Separador	Especial	E → D

Regras de Precedência

- Quando dois operadores de precedência diferentes forem utilizados conjuntamente, o operador que tiver maior precedência será avaliado primeiro.
- Operadores de mesma precedência serão avaliados da esquerda para direita.
- Pode-se forçar a precedência de uma operação, colocando-a entre parênteses.

2.10. Controladores de Decisão

if *que indica que caso satisfeita a condição imposta por ele, um determinado trecho do programa será executado*

```
#include<at89s8252.h>
void main( )
{
    int a=10;
    if(a==10){a=1;}
}
```

pode vir acompanhado do comando else

```
#include<at89s8252.h>
void main( )
{
    int a=3;
    if(a==10){a=1;}
    else
    {a=2;}
}
```

controlador switch.

```
#include<at89s8252.h>
void main( )
{
    int a,b;
    a=3;
    switch(a)
    {
        case 0: b=10; break;
        case 1: b=20; break;
        case 2: b=30; break;
        case 3: b=40; break;
        default: b=0; break;
    }
}
```


2.11. Controladores de Loop

controlador while

```
#include<at89s8252.h>
void main( )
{
    int a=3;
    while(a<10)
    {
        a++;
    }
}
```

resposta sempre "verdadeiro"

```
#include<at89s8252.h>
void main( )
{
    int a=3;
    while(1)
    {
        a++;
    }
}
```

controlador do/while

```
#include<at89s8252.h>
void main( )
{
    int a=3;
    do
    {
        a++;
    }while(a<10);
}
```

controlador for

O controlador for é dividido em quatro parcelas;

```
for( parcela 1; parcela 2; parcela 4 )
{
    parcela 3
}
```

POR EX:

```
#include<at89s8252.h>
void main( )
{
    int b;           /* declaração da variável*/
    int a=3;         /* declaração e atribuição de valor à
variável*/
    while(a<4)       /* parcela condicional*/
    {
        b=a;         /* atribuição do valor de a a b*/
        a++;
    }
}
```

È IGUAL A:

```
#include<at89s8252.h>
void main( )
{
    int b,a;
    for(a=3;a<4;a++)
    {
        b=a;
    }
}
```

2.12. Comandos Relacionados ao Controle de Fluxo

Break

```
#include<at89s8252.h>
void main( )
{
    int a,b,c;
    for(a=0,b=5;a<3;a++)
    {
        if(b==5) break;
    }
}
```

Continue

```
#include<at89s8252.h>
void main( )
{
    int a,b,c;
    for(a=0,b=1;a<3;a++,b++)
    {
        if(b==2) continue;
        c=a+b;
    }
}
```

2.13. Goto e Labels

```
#include<at89s8252.h>
void main()
{
    int n=0;
    P1=0;
inicio:    P1++;
           IE=0x82;
in:        if(var==1000)
           {
               var=0;
               goto inicio;
           }
           else
           {
               while(TR0);
               goto in;
           }
}
```

2.14. Manipulando Alguns Operadores

Negação (!)

If(!var) */* interpretação: caso a variável var não seja verdadeira, tenha valor diferente de zero, o bloco controlado pelo controlador if, por exemplo, não será executado. */*

Complemento Bit a Bit

```
void main()
{   int a,b;
    a=0;
    b=~a;
}
```

Ilustrando:

```
          a → 00000000 (0)      // 1º byte da variável
          int.
b = ~a    ∴ b → 11111111 (-1)
```

Endereço de (&)

```
a=&b;      // "a" receberá o valor do endereço em que a
           //variável "b" armazenou sua variável.
```

No Endereço (*)

```
b=1;      // é armazenado o valor "1"
a=&b;      // a recebe o endereço da variável "b"
*a++;     // o valor armazenado torna-se "2"
```

Cast do Tipo Dado

```
int a;
char b;
b=(char)a;
```

Módulo (%)

```
int a=(6%3); // resulta em zero, pois não tem resto.
```

Deslocamento à Esquerda (< <)

```
a=x<<5 ;      //desloca 5 bits para esquerda
```

Deslocamento à Direita (> >)

```
a=x>>5 ; //desloca 5 bits para direita
```

AND bit a bit (&)

2.15. Manipulando Dados Compostos

```
#include<at89s8252.h>
void main( )
{
    char a[5]={0x01,0x02,0x03,0x04,0x05};    /* valores em
    hexadecimal */
    int b,c;
    for(b=0;b<5;b++)
    {
        c+=a[b];          /* c=c+a[b] */
    }
}
```

Endereço da matriz

a:	0x01	0x02	0x03	0x04	0x05
	a[b] b=0	a[b] b=1	a[b] b=2	a[b] b=3	a[b] b=4

Espaço de memória equivalentes e 8 Bits (1 byte)

Por exemplo, se "b=2", obteremos o terceiro valor da referida matriz, pois "b" se inicializa com "b=0"

2.17. Funções

As funções em um programa em C existem para que possamos declarar blocos de códigos que serão executados várias vezes, e para que não sejamos forçados a reescrevê-los todas as vezes necessárias, fazemos com que o programa acesse esses blocos quando necessitamos. Na utilização dessa linguagem com o objetivo de fazermos programas para microcontroladores, as funções não servem apenas para executar blocos de códigos, mas também para **fazermos o atendimento às interrupções disponíveis em um microcontrolador**, como os timers e o canal serial.

```
#include<at89s8252.h>

int quadrado(int numero)      /*quadrado é o nome da função, e
                               numero é o seu PARÂMETRO*/
{
    int quad=(numero*numero); /* variável local vista apenas
                               Pela função */
    return(quad);             /* valor de retorno para uma
                               variável local do bloco principal */
}

void main( )
{
    int var[2]={10,30};       /* declaração de duas
                               variáveis locais */
    int resp[2];              /* declaração de duas
                               variáveis locais */
    resp[0]=quadrado(var[0]); /* chamada da função e local de
                               retorno */
                               /* var[0] é um ARGUMENTO */
    resp[1]=quadrado(var[1]); /* chamada da função e local de
                               retorno */
                               /* var[1] é um ARGUMENTO */
}

```

Para que possamos indicar que o que queremos é disponibilizar à função o endereço em que o dado está armazenado, e não o dado em si, devemos utilizar o operador "&", que significa "o endereço de".

Do ponto de vista da função, ela deve poder acessar o dado contido no endereço que fora indicado como argumento. Para isso, devemos utilizar o operador "*" (no endereço de) junto com o parâmetro da função. Para que fique evidente o que tentamos explicar até agora, vejamos as seguintes declarações:

quadrado (variavel);	/* forma correta de leitura dessa chamada: chamada da função quadrado cujo argumento é o valor armazenado no endereço "variável" */
quadrado (&variavel);	/* forma correta de leitura dessa chamada: chamada da função quadrado cujo argumento é o endereço "variável" */
void quadrado (int var);	/* forma correta de leitura dessa declaração: declaração de uma função do tipo "void", cujo parâmetro é a variável "var" */
void quadrado (int *var);	/* forma correta de leitura dessa declaração: declaração de uma função do tipo void, cujo parâmetro é o valor armazenado no endereço declarado como argumento, e com o nome "*var" o dado pode ser manipulado dentro da função */

A forma de utilização para uma passagem por referência é exemplificada em seguida:

```
#include<at89s8252.h>
void quadrado(int *var)
{
    *var=((*var)*(*var));
}

void main( )
{
    int variavel[2]={10,30};
    int resp[2];
    quadrado(&variavel[0]);
    quadrado(&variavel[1]);
    resp[0]=variavel[0];
    resp[1]=variavel[1];
}

```

Observação: Ao término do programa resp[0]=100 e resp[1]=900.

Desta forma conseguimos mudar o valor das variáveis "variavel[0]" e "variavel[1]", portanto o valor da resposta é dado diretamente pelo valor armazenado pela variável, enquanto anteriormente a resposta era decorrente do valor que o comando **return** repassava ao bloco principal.

2.17.1. Atendimento às Interrupções

Número para interpretação do compilador	Interrupção	Endereço
0	INT0	0x0003
1	Timer 0	0x000B
2	INT1	0x0013
3	Timer 1	0x001B
4	Serial	0x0023
5	Timer 2	0x002B

A declaração de uma função que atenda, por exemplo, a interrupção proveniente do timer_1 fica sendo:

```
#include<at89s8252.h>
    void timer_1() interrupt 3      /* função de atendimento à
                                    interrupção do timer 1 */
{}                                  /* bloco de código que
                                    será escrito e será
                                    executado caso haja a
                                    interrupção */

void main()
{   while(1){};                  /* loop infinito */
}
```

Podemos, por exemplo, declarar uma função como "crítica". Com essa declaração, caso seja chamada a função, todas as interrupções são desabilitadas (EA=0), e após finalizada (ou em outras palavras, após haver o retorno da função), as interrupções são novamente habilitadas (EA=1). Sua declaração deve ser feita da seguinte maneira:

```
int funcao ( ) critical
{   ...
...
}
```

Quando vamos atender a uma interrupção, às vezes o código nela existente é muito pequeno. Quando uma função de atendimento apresentar pouco código, é aconselhável que, em vez de usarmos a linguagem C, usemos diretamente o assembly. Para que possamos escrever uma função em assembly, devemos associar à sua declaração a palavra "**_naked**".

Esta é uma informação dada pelo próprio compilador, pois o código por ele gerado após a compilação, para uma função, em geral é muito extenso. Veja o exemplo:

Programa em C	Código em assembly após compilação pelo compilador SDCC
<pre>#include <at89s8252.h> char variavel; void funcao() interrupt 1 { variavel++; } void main() { funcao(); }</pre>	<pre>_funcao: ar2=0x02 ar3=0x03 ar4=0x04 ar5=0x05 ar6=0x06 ar7=0x07 ar0=0x00 ar1=0x01 push acc push b push dpl push dph push psw mov psw,#0x00 ;variavel.c:4: { variavel++; ; genPlus ; genPlusIncr inc _variavel 00101\$: pop psw pop dph pop dpl pop b pop acc reti</pre>

Em situações como esta, a fim de evitarmos esse enorme código, podemos escrever o programa da seguinte maneira:

Em C	Em Assembly
<pre>#include <at89s8252.h> char variavel; void funcao() interrupt 1 _naked { +asm inc variavel reti; /* A instrução é necessária (reti) em funções _naked, uma vez que o compilador não fará a compilação desse trecho */ _endasm; } void main() { funcao(); }</pre>	<pre>_funcao: ; naked function: no prologue. ;variavel.c:7: _endasm; ; genInLine ; inc variavel reti A diferença é enorme. Todo o código antes desnecessário, nesta ocasião não é inserido, e assim diminuimos muito o programa.</pre>

2.18. Static

Ao declaramos uma variável "static", os dados não são perdidos, pois mesmo que o bloco seja encerrado, o endereço da variável já foi previamente estabelecido. Para que fique mais clara essa diferença, observe:

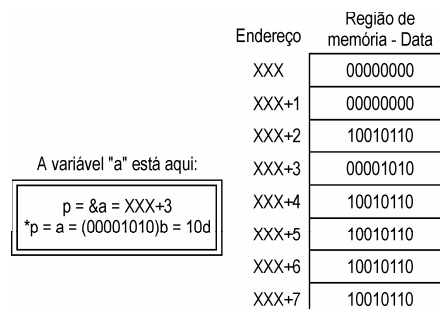
```
#include<at89s8252.h>
void somar(int a,int b)
{
    int c+=(a+b);
}
void main( )
{
    int var[10]={0,1,2,3,4,5,6,7,8,9};
    somar(var[0],var[1]);
    somar(var[2],var[3]);
    somar(var[4],var[5]);
    somar(var[6],var[7]);
    somar(var[8],var[9]);
}
```

Neste exemplo a variável "c" perderá seu valor toda vez que for retornado o fluxo de processamento ao bloco principal, porém, caso declarada como **"static int c"**, no final do programa terá seu valor igual à soma total dos valores das variáveis pertencentes à matriz "var".

2.19. Ponteiros

Ponteiro nada mais é do que uma variável que, em vez de receber um valor de uma outra variável, recebe o seu endereço, pois o ponteiro indicará à variável a região de memória em que os dados foram armazenados. Observe o programa a seguir e a figura 2.7:

```
#include<at89s8252.h>
void main()
{
    char a=10;          /* dado do tipo int cujo endereço é a */
    char *p;            /* variavel do tipo ponteiro */
    p=&a;               /* variavel ponteiro recebendo o endereço
                        da variável a, isto é "p" aponta para o
                        endereço de "a" */
    *p++;               /* incremento do valor 10 em uma unidade */
    /* fim do programa e a=11 */
}
```



XXX pode ter qualquer valor dentro do limite da memória

2.20. Diretivas de Pré-processamento

2.20.1. Diretiva `#include`

```
#include "nome_do_arquivo.h"  
#include <nome_do_arquivo.h>
```

São algumas das bibliotecas disponibilizadas pelo compilador SDCC, ilustradas no quadro seguinte:

ds400rom	8051	8052	80c51xa
regc515c	ds80c390	errno	float
string	sab80515	sdcc-lib	ser
assert	time	tinibios	typeof
limits	at89c55	at89S8252	at89x051
ser_ir	malloc	math	mc68hc908qy
mcs51reg	at89x52	ctvpe	stdlib
stdarg	reg51	reg764	stdio

2.20.2. Diretiva `#define`

Com essa diretiva podemos elaborar macros com argumentos. Sua sintaxe é muito simples e apresentada a seguir:

```
#define nome argumento
```

Exemplo:

```
#define pi 3,1416;  
void main()  
{ float var;  
  var=pi;  
}
```

2.20.3. Diretivas `#if`, `#elif`, `#else` e `#endif`

As diretivas condicionais possuem a mesma funcionalidade dos controladores condicionais em C, porém são testadas na hora da compilação.

Sintaxe:

```
#if <constante - expressão>  
#else  
#endif  
  
#if <constante - expressão>  
#elif <constante - expressão >  
#endif
```

```

#if constante-expressão-1
<bloco 1>
<#elif constante-expressão-2 novalinha bloco-2>
.
.
<#elif constante-expressão-n novalinha bloco-n>
<#else <novalinha> final-bloco>
#endif

```

2.20.4. Diretivas **#ifdef** e **#ifndef**

#ifdef retorna verdadeiro se o símbolo identificado for previamente definido com uma diretiva **#define**, enquanto **#ifndef** retorna verdadeiro caso o símbolo não tenha sido definido.

Sintaxe:

```

#ifdef <identificador> e #ifndef <identificador>

```

2.20.5. Diretiva **#undef**

Essa diretiva permite que possamos eliminar uma definição previamente feita pela direta **#define**.

Sintaxe:

```

#undef <identificador>

```

2.20.6. Diretiva **#error**

Sintaxe:

```

#error <mensagem>

```

Se esta linha do código for compilada pelo compilador, uma mensagem de erro pode aparecer uma vez que tenha sido declarada.

Exemplo:

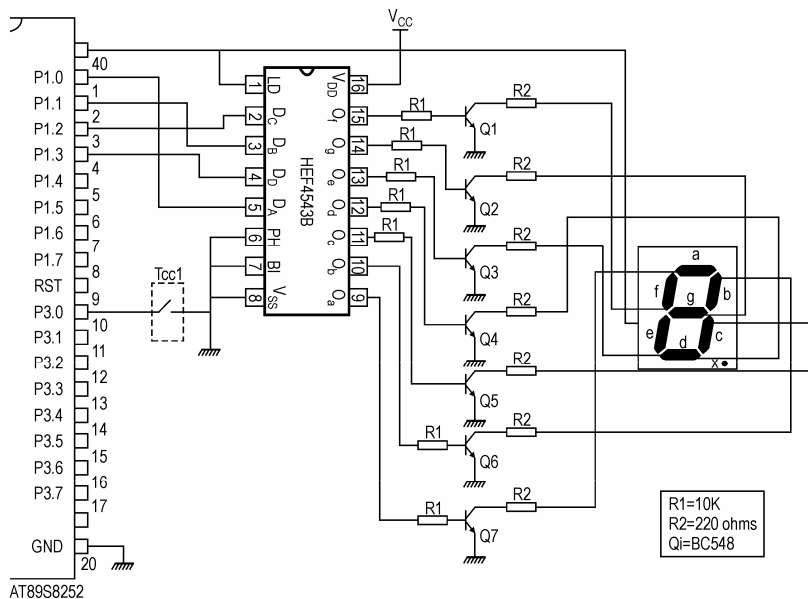
```

#if !defined(variavel)
#error Fatal erro

```

2.20.7. Diretiva **#pragma**

Pragmas são diretivas especiais, e cada compilador possui uma lista em particular. Como as pragmas relacionam-se com funções especiais de um compilador, é necessário conhecermos os recursos disponibilizados por este, para que assim possamos escolher entre usá-los ou não.



EX1-Exemplo com LEDs: apresenta os números 1, 3, 4 e 7 no display em função de se apertar tecla em P3.0:

EX1-Descrição do Programa Elaborado em "Linguagem Descritiva"

Diretivas de pré-processamento:

Bibliotecas de sintaxe: **at89s8252.h**

Diretiva define: Nomear o port **P3.0** como **tec1**

Nomear port **P1.0** como saída "A" para o conversor BCD

Nomear port **P1.1** como saída "B" para o conversor BCD

Nomear port **P1.2** como saída "C" para o conversor BCD

Nomear port **P1.3** como saída "D" para o conversor BCD

Variáveis globais: **nenhuma**

Blocos externos ou funções: **nenhum**

Bloco principal: **main**

Tipo do bloco: **void**

Retorno: **nenhum**

Variáveis locais:

1. Nome ou endereço: **valor**

Tipo do dado: **simples**

Região de armazenamento do dado: **default - data**

Quantidade de bits a reservar: **int -16bits**

Instruções:

Atribuir valor 0 à variável **valor**

Controlador **while**:

Função: loop infinito

Bloco do controlador:

Controlador **if**:

Função: esperar que a tecla seja apertada

Comando **else**: no caso de a tecla ser apertada, espera que a soltemos

Comando **switch**:

Variável: **valor**

Caso **valor = 0** apresenta valor 1 no display

Caso **valor = 1** apresenta valor 3 no display

Caso **valor = 2** apresenta valor 4 no display

Caso **valor = 3** apresenta valor 7 no display

Caso **valor diferente** dos anteriores: default apresentar valor 9

Incrementar variável **valor**

Controlador **if**:

Função: verificar valor da variável "**valor**": caso igual a 7, atribuir valor "0", possibilitando assim que o comando switch seja sempre executado

Programa

```
#include<AT89s8252.h>
#define tec1 P3_0      // nomeia port P3.0 como tecla 1 -
                        //resistor de pull-up interno.
#define DA P1_0 // nomeia port P1.0 como saída A para o
                //conversor bcd.
#define DB P1_1 // nomeia port P1.1 como saída B para o
                //conversor bcd.
#define DC P1_2 // nomeia port P1.2 como saída C para o
                //conversor bcd.
#define DD P1_3 // nomeia port P1.3 como saída D para o
                conversor bcd.

void main()
{   int valor;        // declara as variáveis locais: tempo e valor
    valor=0;
    while(1) // loop infinito
    {   if(tec1) // espera que uma tecla seja apertada
        {   continue;   }
        else
        {   while(tec1==0); // trava até soltarmos a tecla
            }
        switch(valor)
        {   case 0:      DA=1;
                                DB=0;
                                DC=0;
                                DD=0;
                                break; // escreve 1 no display
            case 1:      DA=1;
                                DB=1;
                                DC=0;
                                DD=0;
                                break; // escreve 3 no display
            case 2:      DA=0;
                                DB=0;
                                DC=1;
                                DD=0;
                                break; // escreve 4 no display
            case 3:      DA=1;
                                DB=1;
                                DC=1;
                                DD=0;
                                break; // escreve 7 no display
            default:     DA=1;
                                DB=0;
                                DC=0;
                                DD=1;
                                break; // escreve 9 no display
        }
        valor++;
        if(valor==7) valor=0;
    }
}
```

EX2-Exemplo com LEDs: apresenta os números 1, 4, 4 e 8 no display em função de se apertar tecla em P3.0. Variável como "vetor".

EX2-Descrição do Programa Elaborado em "Linguagem Descritiva"

Diretivas de pré-processamento:

Bibliotecas de sintaxe: **at89s8252.h**

Nomear o port **P3.0** como **tec1** - diretiva define

Variáveis globais: **nenhuma**

Blocos externos ou funções: **nenhum**

Bloco principal: **main**

Tipo do bloco: **void**

Retorno: **nenhum**

Variáveis locais:

1. Nome ou endereço: **vetor**

Tipo do dado: **simples**

Região de armazenamento do dado: **default - data**

Quantidade de bits a reservar: **char - 8 bits**

2. Nome ou endereço: **dados**

Tipo do dado: **composto**

Quantidade de variáveis: **4**

Região de armazenamento dos dados: **code - dados para leitura**

Quantidade de bits a reservar para cada variável: **int - 16 bits**

Instruções:

Iniciar matriz dados com os valores: 0x01, 0x04, 0x07, 0x08 - notação hexadecimal

Zerar port P1 para que não haja nenhum valor inicial no display

Controlador **while**:

Função: **loop infinito**

Bloco do controlador:

Controlador **for**:

Cláusula para **inicialização de variáveis**: **vetor=0**

Cláusula condicional: **vetor<4**

Cláusula para incremento de variáveis: **vetor+1**

Bloco do controlador:

Controlador **while**:

Função: verificar acionamento da tecla **tec1 - P3.0**

Bloco do controlador: **nenhum código**

Controlador **while**:

Função: verificar se tecla **tec1 - P3.0** foi solta

Bloco do controlador: **nenhum código**

Atribuir o valor ao **port P1** da variável (**dados+vetor**)

Programa

```
#include<at89s8252.h>
#define tec1 P3_0
void main()
{   char vetor;
    code int dados[4]={0x01,0x04,0x07,0x08};
    P1=0;
    while(1)
    {   for(vetor=0;vetor<4;vetor++)
        {   while(tec1);           // espera que uma tecla seja
apertada

                while(tec1==0);    // trava até soltarmos a tecla
                P1=dados[vetor];

        }
    }
}
```

EX3-Controla display 7 seg.e desenha numeros 1, 4,7 e 8, com tempo de 120ms gerado por um Timer de 60ms.

EX3-Descrição do Programa Elaborado em "Linguagem Descritiva" - timer

Diretivas de pré-processamento:

Bibliotecas de sintaxe: **at89s8252.h**

Variáveis globais: **nenhuma**

Blocos externos ou funções:

inicia(): Função: **atribuir o valor de contagem e iniciar a operação do timer 0**

Tipo: **void**

Retorno: **nenhum**

Parâmetros: variável **vezes** - quantidade de vezes que deve ser iniciada a operação do timer 0; no caso 2 vezes - 120ms.

Bloco de código:

Selecionar o modo de trabalho para o **timer 0 - modo1 16 bits**

Atribuir valor aos registradores para termos uma **contagem de 1ms**

Iniciar contagem

timer0: Função: atendimento da **interrupção** proveniente do **timer 0**

Tipo: **void**

Retorno: **nenhum**

Parâmetros: **nenhum**

Bloco de código:

Zerar o flag TR0 a fim de parar a contagem após overflow do timer 0.

Bloco principal: **main**

Tipo do bloco: **void**

Retorno: **nenhum**

Variáveis locais:

1. Nome ou endereço: **vetor**

Tipo do dado: **simples**

Região de armazenamento do dado: **default - data**

Quantidade de bits a reservar: **char - 8 bits**

2. Nome ou endereço: **dados**

Tipo do dado: **composto**

Quantidade de variável: **4**

Região de armazenamento dos dados: **code - dados para leitura**

Quantidade de bits a reservar para cada variável: **int - 16 bits**

Instruções:

Iniciar matriz dados com os valores: **0x01, 0x04, 0x07, 0x08** - hex

Zerar port P1 para que não haja nenhum valor inicial no display

Habilitar a interrupção do timer 0

Controlador while:

Função: **loop infinito**

Bloco do controlador:

Controlador for:

Cláusula para **inicialização de variáveis**: **vetor=0**

Cláusula condicional: **vetor<4**

Cláusula para incremento de variáveis: **vetor+1**

Bloco do controlador:

Atribuir o valor ao port P1 da variável dados+vetor

Fazer a chamada da função **inicia** . O **argumento** é o **valor 2** que será recebido pelo parâmetro **vezes**.

Controlador while:

Função: Esperar que seja atendida a interrupção do timer 0. Como no atendimento será atribuído nível "0" ao flag TR0, o controle é feito com base nesse valor. Isso significa que, havendo o término da contagem, é acionada a interrupção do timer 0. A função responsável pelo atendimento tem seu bloco de código executado, em que, por fim, tem-se a atribuição do nível "0" ao "flag" TR0.

Programa

```
#include<at89s8252.h>
void inicia(char vezes)
{
    while(vezes)
    {
        TMOD=0x01;           // TIMER 0 - modo 1 ( 16 bits)
        TH0=~(60000/256);     // atribui o valor 0x15h, figura 1.12
        TL0=-(60000%256);     // atribui o valor 0x9Fh, figura 1.12
        TR0=1;                // seta o flag TR0 dando início à contagem
        while(!TF0);          // espera término da contagem
        vezes--;              // decrementa a variável vezes -
                               // determina 60ms x 2!
    }
}

void timer0( )interrupt 1 _naked // atende a interrupção após
                                // overflow - TF0=1
{
    _asm                        // indica início da declaração
                                // do código em assembly
    clr TR0                     // atribui nível lógico "0"
    ao                           // flag TR0 parando o timer 0
                                // /* instrução de retorno ao
                                // programa principal obrigatória
                                // em funções naked */
    reti;                       // indica término da
                                // do código em assembly
}

void main()
{
    int vetor;
    code int dados[4]={0x01,0x04,0x07,0x08};
    P1=0;
    IE=0x82;                    // habilita interrupção (timer0)
    while(1)
    {
        for(vetor=0;vetor<4;vetor++) // incremento dado pelo timer
        {
            P1=dados[vetor];
            inicia(2);
            /* indica na chamada da função que haverá a contagem de
            duas vezes o valor atribuído aos registradores TH0 e
            TL0, conforme demonstra o código dentro da função */
        }
    }
}
```

Observação: As operações que movimentam os valores para os registradores TH0 e TL0 funcionam da seguinte forma: o valor de registro inicial, considerando os dos registradores, é 0xFFFF. Se atribuirmos (operador "=") um determinado valor ao registrador, esse valor é alterado. Se atribuirmos e subtraímos (operador "-=") um valor desse registrador, teremos como resultado o valor já registrado 0xFF menos o valor atribuído. Na atribuição do valor para TH0, primeiramente divide-se o valor de contagem por 256 e inverte-se o resultado da divisão. A seguir, temos exposta esta operação:

$(60000d/256d) = 234d = 1101010b \therefore \text{o inverso } (\sim) = 0010101 = 15h \text{ ou } 0x15$

Já na atribuição do valor de registro para o registrador TL0, temos a utilização do operador "%", referente ao resto da divisão entre o dividendo 60000 e o divisor 256. O resto dessa divisão é igual a 96 (0x60). Portanto, subtraindo de 0xFF de 0x60 resulta 0x97.