EEL7030 - Microprocessadores

Prof. Raimes Moraes

EEL - UFSC

Linguagem C – 8051
Operadores
(http://www.keil.com/support/man/docs/c51/)

- Aritméticos: + − * / % (var = 9 % 2 => var = 1)
- Relacionais: == != < <= > >= && || !
- Lógicos (bit a bit): & | ^ ~ << >>
  (ANL ORL XRL CPL)
- Atribuição: = += -=
- Incremento/decremento: ++ --

Linguagem C – 8051
Estruturas para Seleção

if - else

```
if (condição_1)  declaração_1;

        else if (condição_2) declaração_2;
        else if (condição_3) {declaração_3; declaração_4}
                        .
                        .
        else if (condição_n) declaração_M;
        else faça_default;
```

switch - case

```
switch (variável)
    {
    case constante_1:    declaração_1;    break;
    case constante_2:    declaração_2;  declaração_3;  break;
                        .
                        .
    case constante_n:    declaração_L;    break;
    default:             declaração_default;
    }
```

Linguagem C – 8051
Estruturas para Repetição

for

```
for (inicialização;condição de parada;incremento/decremento) {
                                        declaração_1;
                                              ...
                                        declaração_n;
                                                                }
```

while

```
while (condição)        {       declaração_1;
                                ...
                                declaração_n;
                        }
```

do – while

(Executa ao menos, 1 vez o laço *do*)

```
do      {       declaração_1;
                ...
                declaração_n;
        }
while (condição);
```

break Interrompe a execução de um comando (switch) ou de loop (for, while, do-while). O ***break*** força a saída do laço mais interno.

continue força a execução da próxima iteração do loop

Exemplo de tipo de dados enum (ver prox. Slide)

```
enum mes {jan,fev,mar,abr,mai,jun,jul,ago,set,out,nov,dez};
mes valor;

valor = ago; /* equivalente a valor = 7 */
```

Tipos de Dados

Table 2.1 CSI data types		
Data type	**Bits**	**Range**
bit	1	0 or 1
signed char	8	−128 to +127
unsigned char	8	0 to +255
enum	16	−32768 to +32767
signed short	16	−32786 to +32767
unsigned short	16	0 to 65535
signed int	16	−32768 to +32767
unsigned int	16	0 to 65535
signed long	32	−2147483648 to 2147483647
unsigned long	32	0 to 4294967295
float	32	±1.175494E-38 to ±3.402823E+38
sbit	1	0 or 1
sfr	8	0 to 255
sfr16	16	0 to 65535

Tipos de Dados do C51

- **bit:** (0 a 1) define variáveis do tipo bit na memória ram interna (0x20 a 0x2F - *bdata*). Não pode ser utilizado em ponteiro ou *array*.

- **sbit:** (0 a 1) define variáveis do tipo bit na região de memória dos registradores de funções especiais (*special function registers – SFR)* ou em *bdata* (0x20 a 0x2F).

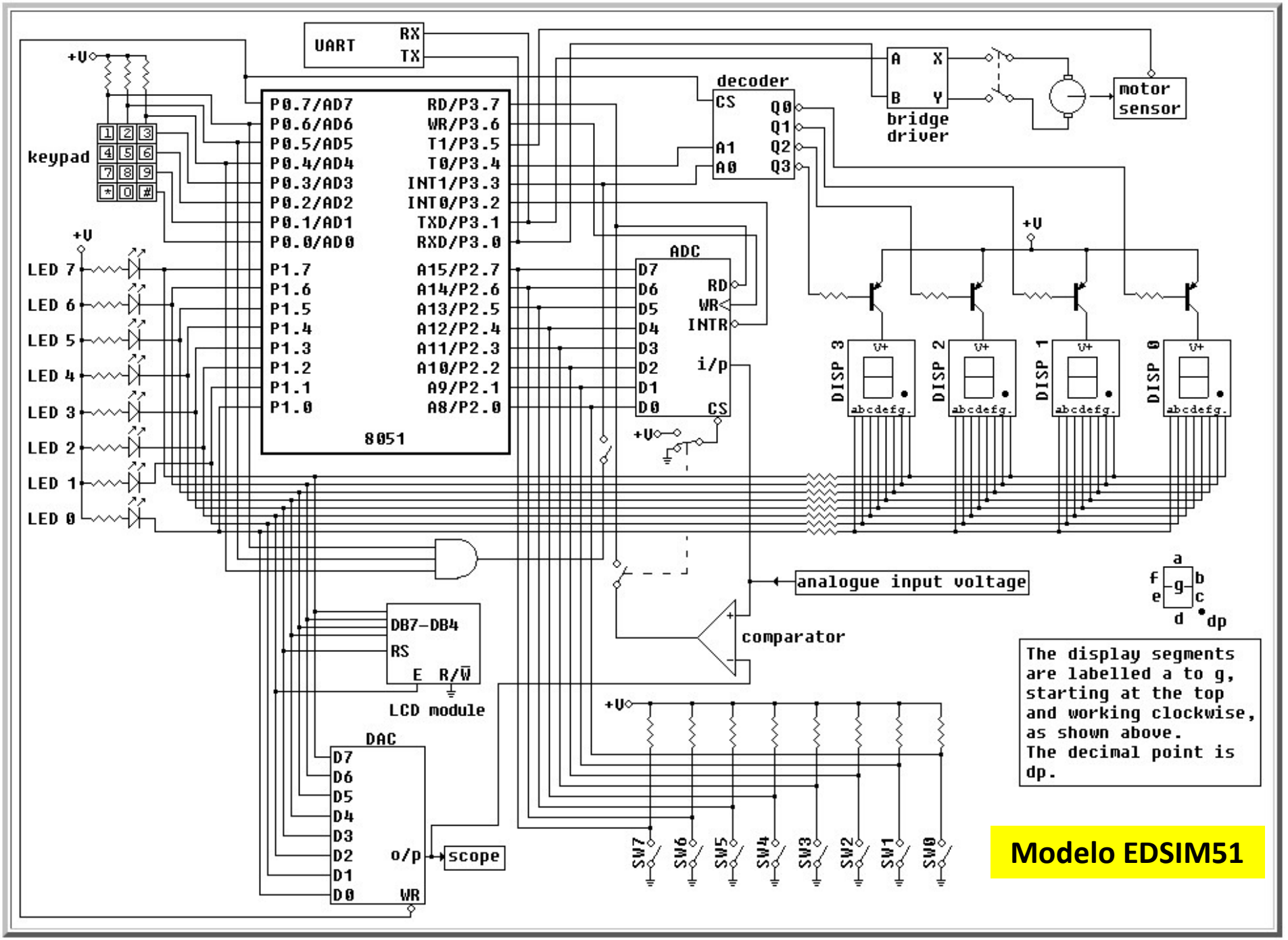
```
sbit CY = PSW^7;     sbit CY = 0xD0^7;     sbit CY = 0xD7;  //  sinônimos
char bdata ibase;                          /* Bit-addressable char*/
sbit mybit0 = ibase ^ 0;                   /* bit 0 de ibase */
```

- **sfr:** define variáveis nos endereços de memória dos SFR.

```
sfr P1 = 0x90;  // P1 passa a corresponder ao endereço 90H
```

- **sfr16:** define endereços sequenciais nos SFR.

```
sfr16  DPTR = 0x82; //  DPL  em 0x82; DPH  em 0x83;
DPTR = 0xFFA0;
```


UART
RX
TX
+U
keypad
1 2 3
4 5 6
7 8 9
* 0 #
P0.7/AD7
P0.6/AD6
P0.5/AD5
P0.4/AD4
P0.3/AD3
P0.2/AD2
P0.1/AD1
P0.0/AD0
P1.7
P1.6
P1.5
P1.4
P1.3
P1.2
P1.1
P1.0
RD/P3.7
WR/P3.6
T1/P3.5
T0/P3.4
INT1/P3.3
INT0/P3.2
TXD/P3.1
RXD/P3.0
A15/P2.7
A14/P2.6
A13/P2.5
A12/P2.4
A11/P2.3
A10/P2.2
A9/P2.1
A8/P2.0
8051
decoder
CS
A1
A0
Q0
Q1
Q2
Q3
A
B
X
Y
bridge driver
motor
sensor
ADC
D7
D6
D5
D4
D3
D2
D1
D0
RD
WR
INTR
i/p
CS
LED 7
LED 6
LED 5
LED 4
LED 3
LED 2
LED 1
LED 0
DISP 3
DISP 2
DISP 1
DISP 0
abcdefg.
analogue input voltage
comparator
DB7-DB4
RS
E
R/W
LCD module
DAC
o/p
WR
scope
SW7
SW6
SW5
SW4
SW3
SW2
SW1
SW0
a
b
c
d
e
f
g
dp
The display segments are labelled a to g, starting at the top and working clockwise, as shown above.
The decimal point is dp.
Modelo EDSIM51

```
// EXEMPLO: Programa que mostra números hexadecimais (0 a F) no display 3 do EDSIM51

#include <reg51.h> // declaração de special function registers para 8051

unsigned char converte_7seg (unsigned char);

void main (void)
{
  short i;
  unsigned char j=0;

  P0=0x80;
  P3=0xFF;


  while (1)          {

                     if ( j == 16) j=0;

                     for (i = 0; i < 15000; i++); // atraso para Edsim 2.1.13 – Update freq=50000

                     P1=converte_7seg(j);
                     j++;
                     }
  }
```

```
unsigned char converte_7seg (unsigned char dado) //   função retorna valor a ser escrito…
{                                                    //… nos displays para formar o caractere
 unsigned char leds;

switch (dado)                                        // GFEDCBA
            {
                        case 0:    leds  = 0x40; break;   // "1000000";
                        case 1:    leds  = 0x79; break;   // "1111001";
                        case 2:    leds  = 0x24; break;   // "0100100";
                        case 3:    leds  = 0x30; break;   // "0110000";
                        case 4:    leds  = 0x19; break;   // "0011001";
                        case 5:    leds  = 0x12; break;   // "0010010";
                        case 6:    leds  = 0x02; break;   // "0000010";
                        case 7:    leds  = 0x78; break;   // "1111000";
                        case 8:    leds  = 0x00; break;   // "0000000";
                        case 9:    leds  = 0x10; break;   // "0010000";
                        case 10:  leds  = 0x08; break;   // "0001000";
                        case 11:  leds  = 0x03; break;   // "0000011";
                        case 12:  leds  = 0x46; break;   // "1000110";
                        case 13:  leds  = 0x21; break;   // "0100001";
                        case 14:  leds =  0x06; break;   // "0000110";
                        case 15:  leds =  0x0E; break;   // "0001110";
                        default:   leds =  0x80;         // "0000000";
      }

  return leds;

    } // end converte
```

```
/*---------------------------------------------------------------------------
```
REG51.H // Conteúdo parcial do registrador reg51.h


```
#ifndef __REG51_H__
#define __REG51_H__

/*  BYTE Register  */

sfr         P0     =     0x80;
sfr         P1     =     0x90;
sfr         P2     =     0xA0;
sfr         P3     =     0xB0;
sfr         PSW    =     0xD0;
sfr         ACC    =     0xE0;
sfr         B      = 0xF0;
sfr         SP     = 0x81;
sfr         DPL    = 0x82;
sfr         DPH    = 0x83;
...


/*  BIT Register  */
/*  PSW   */

sbit        CY   = 0xD7;
sbit        AC   = 0xD6;
.....
```

Especificação de Regiões de Memória

Alocação de dados/variáveis na memória :

- **code**: memória de programa acessada por MOVC A,@A+DPTR

  Ex: unsigned char code strcte[] =“exemplo“;

- **data**: memória de dados interna acessada por endereçamento direto (0 a 0x7F)

- **idata**: memória de dados interna acessada por endereçamento indireto (0 a 0xFF).

- **bdata**: memória de dados interna acessada por endereçamento bit a bit (0x20 a 0x2F). Ex: unsigned char bdata dado;

- **xdata**: memória de dados externa acessada por MOVX @DPTR

- **pdata**: memória de dados externa acessada por MOVX A,@Ri
  (permite acessar somente os primeiros 256 bytes de XDATA: 0x0000-0x00FF).

_at_: Indica onde variável deve ser alocada na memória

```
unsigned char xdata  RTC  _at_  0xfa00 ;


// variável RTC é armazenada no endereço 0xfa00 da memória de dados externa

char xdata text[256]  _at_ 0xE000;

// vetor text é armazenado no endereço 0xE000 da memória de dados externa
```

OBS: Como não se sabe onde o compilador aloca outra variáveis, tal tipo de alocação deve ser utilizada com cautela.

```
#include <reg51.h> //Mostra números hexadecimais nos 4 displays do EDSIM51

unsigned char converte_7seg (unsigned char);

volatile unsigned char bdata cntdsp;  // volatile não necessário – apenas para apresentar
 sbit CS=P0^7;
 sbit cntdsp_1=cntdsp^1;
 sbit cntdsp_0=cntdsp^0;
 sbit END1=P3^4;
 sbit END0=P3^3;

void main (void)   {
   short i;
   unsigned char j;

  CS=0;   cntdsp=3;   j=0;

   while (1)    {
                                          END1=cntdsp_1;    END0=cntdsp_0;    //end. dsp7
                                          P1= converte_7seg(j);          j++;
                                           CS=1;
                    for (i = 0; i < 22000; i++); // atraso Edsim 2.1.13 – Update freq=50000
                                          if (cntdsp==0)  cntdsp=4;
                                          cntdsp--;  CS=0;
                                           if ( j == 16) j=0;
               }  // end of while
                     } //end of main
```

Type Qualifiers

const

Utilizado para definir objetos cujos conteúdos não podem ser alterados.
const float xdata pi = 3.1415927;

volatile

Informa compilador para não otimizar instruções com a variável, pois a mesma pode ser alterada além do controle do compilador (não pelo software, mas pelo hardware)

```
unsigned char reg1;   // Hardware Register #1
unsigned char reg2;   // Hardware Register #2

void func (void)
{
while (reg1 & 0x01)   // Repeat while bit 0 is set
  {
  reg2 = 0x00;        // Toggle bit 7
  reg2 = 0x80;
  }
}
```

Após otimização pelo compilador:

Reg1 não é atualizado entre iterações do loop, apesar de modificado pelo hardware

```
while (1) reg2 = 0x80;
```

Declarar as variáveis como *volatile*

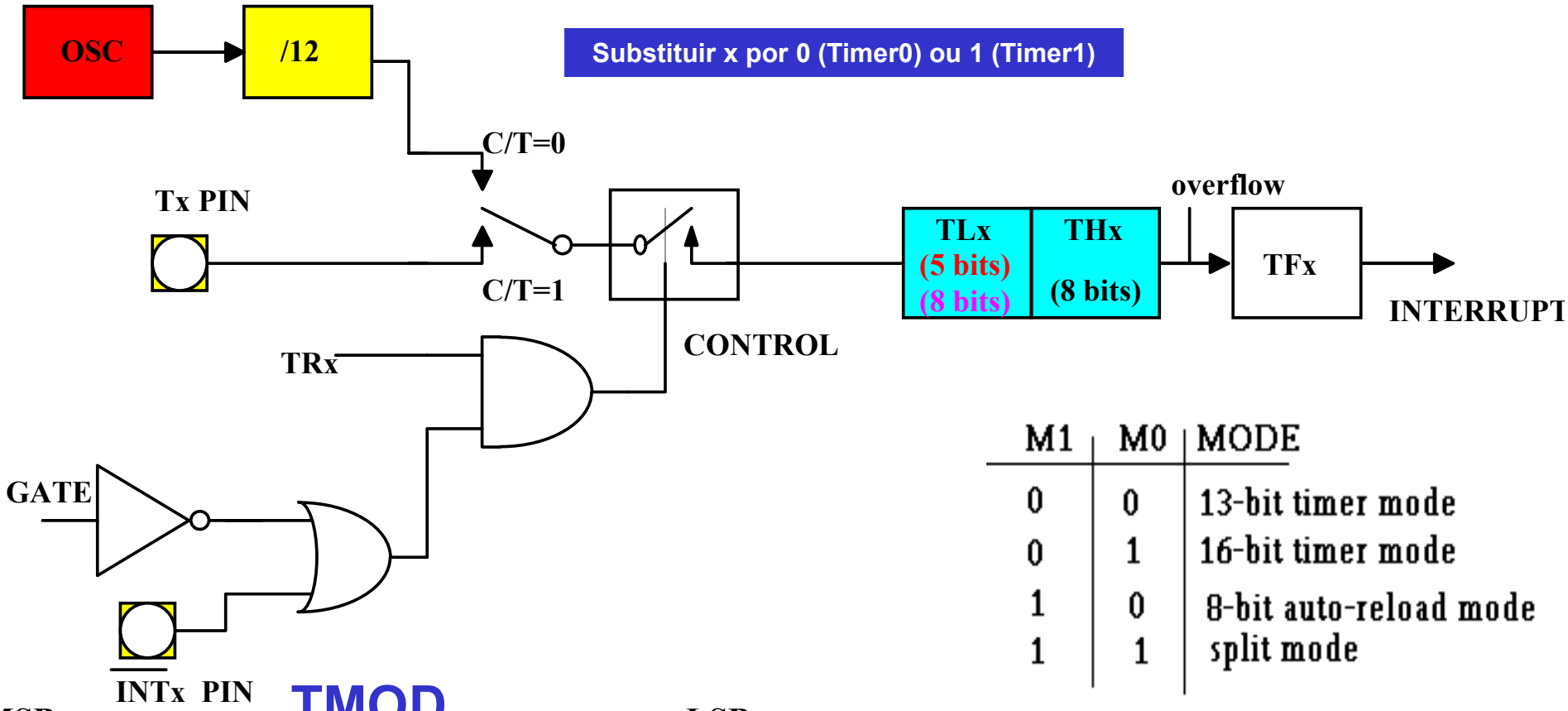
```
#include <reg51.h> //Escreve números hexadecimais  (0 a F) nos 4 displays do
//...EDSIM51.  Utilizando timer 1 no modo 1 para inserir atraso

unsigned char converte_7seg (unsigned char);
void delay (unsigned char);


volatile unsigned char bdata cntdsp;


 sbit  CS=P0^7;          sbit  cntdsp_1=cntdsp^1;   sbit cntdsp_0=cntdsp^0;
 sbit  END1=P3^4;      sbit  END0=P3^3;

void main (void)          {
unsigned char j;

   TMOD = 0x10;       TCON = 0;
    CS=0;                  cntdsp=3;              j=0;

    while (1)    {                                 END1=cntdsp_1;      END0=cntdsp_0;    //end. dsp7
                                                   P1= converte_7seg(j);              j++;
                                                   CS=1;
                                                   delay(3);
                                                   if (cntdsp==0)  cntdsp=4;
                                                   cntdsp--;  CS=0;
                                                    if ( j == 16) j=0;
                    }  // end of while
                                                   } //end of main
```

//Rotina de atraso para o exemplo anterior utilizando timer1

```
void delay (unsigned char nro_contagens) {
unsigned char contador;

    TH1 = 0x00;
    TL1 = 0x00;
    TR1 = 1;

      for (contador=0; contador < nro_contagens; contador++) {

                                                          while (!TF1);

                                                          TF1=0;

                                                                  } /* end of  for*/
     TR1 = 0;

                    } /* end of delay */
```

Temporizadores / Contadores

Modos 0 (13 bits) e 1 (16 bits)



M1	M0	MODE
0	0	13-bit timer mode
0	1	16-bit timer mode
1	0	8-bit auto-reload mode
1	1	split mode

TMOD

MSB LSB

GATE	C/T	M1	M0	GATE	C/T	M1	M0	89H

TIMER 1 TIMER 0

TCON

MSB LSB

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	88H

Estrutura de Programa em C

1. **Diretivas de Compilação e Include files**
2. **Declaração de funções (Protótipos)**
3. **Declaração de variáveis globais e constantes**
4. **Função Main**
5. **Funções**
6. **Tratadores de interrupção**

Startup.a51

I. **Apaga RAM interna e externa**

II. **Inicializa variáveis e *stack pointer***

III. **Executa JMP “main”**

Exemplo

(Simular no Sistema de Desenvolvimento)

```
#pragma LARGE                  // modelo de memoria. Large: Variáveis em xram
                        // Compact: Variáveis em páginas de 256 bytes de xram (movx @r0)
                               // Small: Variáveis em iram
#include <reg51.h>             // declaração de special function registers para 8051
#include <stdio.h>             //  inclui funções de I/O

#define  mensagem "Hello World\n"

/************************/
/* main program     */
/************************/

void main (void)  {

   SCON  =          0x40;        // SCON: modo 1, 8-bit UART, não habilita recepção
   TMOD |=          0x20;        // TMOD: timer 1, modo  2
   TH1     =        0xf4;        // TH1:  valor de recarga para 2400 baud; clk = 11,059MHz
   TR1     =        1;           // TR1:  dispara timer 1
   TI      =        1;           // Gera int. serial

  printf (mensagem) ;             //   'printf'  tem saída para serial
  while (1);
}
```

Modelos de Memória
Compilador Keil C51

- **small**: Variáveis alocadas na memória interna do 8051. Vantagem: velocidade. Limitação: tamanho da memória interna.

- **compact**: Variáveis residem em uma página (256 bytes) da memória externa de dados. Acesso por endereçamento indireto através dos registradores R0 e R1 (MOVX @R0 ou @R1). Vantagem: maior espaço para variáveis. Desvantagem: menor velocidade.

- **large**: variáveis alocadas na memória externa de dados (64kiB). Vantagem: maior espaço para variáveis. Desvantagem: menor velocidade.

Ponteiros - Compilador Keil C51

Genéricos/*Untyped:* Apontam para qq área de memória do 8051(possuem 3 bytes: região de memória (1º byte) e endereço (MSB-LSB) – mais lentos)

1. Ponteiro armazenado na IRAM (*default*)

```
char    *s;        /* ponteiro para char */
int     *numptr;   /*  ponteiro para int    */
```

2. Ponteiro armazenado em outras áreas de memória

```
char    * xdata strptr;     /* ponteiro alocado em xdata para dado char */
int     * pdata numptr;     /* ponteiro alocado em pdata para dado int */
```

Memory-Specific/Typed (especifica a região de memória a ser acessada)
Gera menor código e é mais rápido: ocupa 1 byte (ptrs para idata, data, bdata, pdata) ou 2 bytes (ptrs para code, xdata)).

1. Ponteiro armazenado na IRAM (*default*)

```
char data *str;             /* ponteiro para dado char alocado em data */
int xdata *numtab;          /* ponteiro para dado int alocado em xdata */
```

2. Ponteiro armazenado em outras áreas de memória

```
int xdata *pdata numtab;    /* ponteiro alocado em pdata para  dado int em xdata*/
long code *xdata powtab; // ponteiro alocado em xdata para dado long alocado em code
```

```
#include <reg51.h>            //Programa que ilustra a utilização de ponteiros
#include <stdio.h>

unsigned char code  tabela[]="Hello";
//unsigned char code tabela[]={'a','b','c','d','e'}; /* outra possibilidade de
inicializacao da tabela */

void main (void)  {
short m;
unsigned char code *y;

   SCON  =        0x40;     // SCON: modo 1, 8-bit UART, não habilita recepção
   TMOD |=        0x20;     // TMOD: timer 1, modo  2
   TH1    =       0xf4;     // TH1:  valor de recarga para 2400 baud; clk = 11,059MHz
    TR1   =       1;        // TR1:  dispara timer 1
    TI    =       1;        // Gera int. serial

  y = tabela;                          // inicializacao do ponteiro
  printf ("Hello World\n");            // transmite toda a mensagem

  for (m=0;m< 5;m++)  printf("%c\n",*y++); // transmite caractere por caractere

while (1);
                      }
```

Declaração de Função

Compilador Keil C51

[tipo_de_dado_a_retornar]

nome_da_função ([parâmetros])

[modelo_de_memória]

[reentrant]

[interrupt *n*]

[using *n*]

Exemplo de protótipo de função: float teste(short);

Exemplo de declaração de função

```
float teste (short r) {
float area;

   area = pi*r*r;
   return area;
                        }
```

Passagem de Parâmetros para a Função

Compilador Keil C51

[parâmetros] => Para maior velocidade de execução, até 3 parâmetros são passados para função através de registradores (diretiva de compilação **REGPARMS)**. A diretiva **NOREGPARMS** força a passagem de todos os parâmetros através da memória.

Parâmetro	char, ptr de 1-byte	Int, 2-byte ptr (msb – lsb)	long float (leee)	ptr genérico (tipo, msb, lsb)
1	R7	R6 & R7	R4-R7	R1-R3
2	R5	R4 & R5	R4-R7	R1-R3
3	R3	R2 & R3		R1-R3

```
#pragma NOREGPARMS
extern int new_func (int, char)
```

Se houver mais que 3 parâmetros, estes são passados através de endereços da memória. Se tiver parâmetro tipo bit, declará-lo por último.

Retorno de Valores para a Função

Compilador Keil C51

[tipo_de_dado_a_retornar]

Tipo	Registradores de retorno	Retorna
bit	Carry Flag	Único bit em flag de carry
char, unsigned char, ponteiro (ptr) de 1-byte	R7	Único byte em R7
int, unsigned int, ptr de 2-bytes	R6 & R7	MSB em R6 - LSB em R7.
long, unsigned long	R4-R7	MSB em R4 - LSB em R7.
float	R4-R7	Formato 32-Bit IEEE
ptr genérico	R1-R3	Tipo de memória em R3, MSB em R2 - LSB em R1.

[modelo_de_memória] => o modelo default para a função é o determinado pela opção de compilação. Contudo, é possível especificar o modelo para uma função em particular (*small*, *compact*, *large*)

[reentrant] => função que pode ser chamada por diferentes processos (programa principal e interrupção). Quando a mesma está sendo executada, outro processo pode interrompê-la e reiniciar sua execução ou a mesma pode ser paralelamente executada por mais de um processo (situações de tempo real). Para cada chamada da função, área é criada na memória para armazenar parâmetros, variáveis e simular pilha. Não suporta variáveis do tipo bit.

[using *n*] => *n* assume valores de 0 a 3.

- **Especifica banco de registradores a ser utilizado pela função ou tratador de interrupção. Permite que a função utilize um banco diferente do programa principal, evitando que o conteúdo dos registradores tenha de ser salvo em pilha.**

- **Armazena o número do banco em uso na pilha e o recupera ao final da execução da função (PUSH/POP PSW). Não deve ser utilizado em funções que retornam valor em seus registradores (≠void).**

- **Não inclua tal declaração no protótipo da função.**

Registerbank

- Ao ser resetado, o 8051 utiliza banco 0. Para modificar o banco para funções subsequentes:

#pragma REGISTERBANK(1)

[interrupt *n*] => *n* assume valores de 0 a 4.

A diretiva interrupt *n* especifica que a função na qual se encontra declarada deve ser executada em resposta à solicitação da fonte de interrupção **n**, onde **n** assume valores de 0 a 4, com a seguinte correspondência => 0: EXT0; 1: TIMER0; 2: EXT1; 3: TIMER1; 4: SERIAL.

O compilador grava no endereço correspondente da memória de código desvio para apontar para o tratador.

Funções que atendem interrupções não podem receber parâmetros ou retornar valores. Saberia a razão?

Não devem ser chamadas fora de interrupções, pois o compilador inclui a instrução RETI.

Numeração dos Vetores de Interrupção

Número da Int.	Descrição	Endereço
0	Externa 0	03 h
1	Timer/Counter 0	0B h
2	Externa 1	13 h
3	Timer/Counter 1	1B h
4	Porta Serial	23 h

Interrupções

(Exemplo de declaração de tratador de interrupção)

```
unsigned char data segundo;
.....
void c_timer0 (void)  interrupt 1 using 2
{
static short contaseg;
if (++contaseg == 4000)        {
                                        segundo++;
                                        contaseg = 0;
                                }
}
```

CLASSES DE ARMAZENAMENTO

- **static: - quando declarada dentro da função, retém valores entre chamadas.**
  - quando declarada fora da função, não pode ser acessada fora do código fonte no qual foi declarada.

  static *data-type name* [=valor da mesma];

- **extern variável declarada dentro de outro arquivo fonte.**

  extern *data-type name*;

- **register variável deve ser alocada em registrador(es) e não na RAM. Utilizado dentro de funções.**

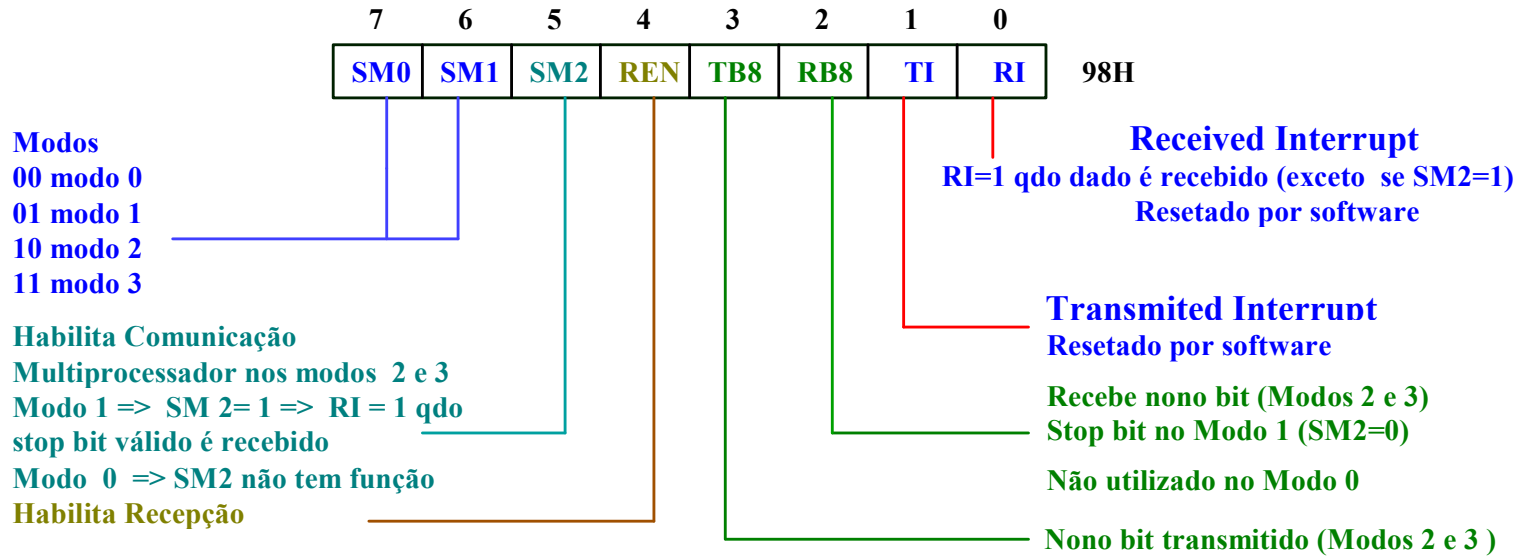
  register *data-type name* [= valor da mesma];

```
#include <reg51.h> //Programa que escreve números hexadecimais  (0 a F) nos 4 displays do
                   //...EDSIM51.  Utilizando timer para permitir visualização dos dados mostrados

void c_timer0 (void);
unsigned char converte_7seg (unsigned char);

unsigned char atraso;
volatile unsigned char bdata cntdsp;

 sbit CS=P0^7;  sbit cntdsp_1=cntdsp^1;   sbit cntdsp_0=cntdsp^0;   sbit END1=P3^4;   sbit END0=P3^3;

void main (void)        {
unsigned char j;

  TMOD = 0x01;   TCON = 0;  ET0=1;   EA=1;  // timer0 no modo 1 – interrupção habilitada
  CS=0;    cntdsp=3;   j=0;

  while (1) {                          END1=cntdsp_1;       END0=cntdsp_0;
                                       P1=converte_7seg(j);   j++;  CS=1;
                                       TH0 = 0x00;   TL0 = 0x00;
                                       TR0=1;                  atraso=0;
                                       while (atraso!=2);
                                       TR0=0;
                                       if (cntdsp ==0)  cntdsp=4; cntdsp--;
                                       CS=0;
                                       if ( j == 16) j=0;
                  }  // end of while
                                       } //end of main

void c_timer0 (void)  interrupt 1 { atraso++; }
```

```
// Programa transmite (0x41 a 0x61) e recebe dados pela serial utilizando tratador de interrupção
// Dados recebidos são armazenados em 32 posições da memória externa. Ao receber 32, volta a preencher o buffer do início.
// Qdo recebe 'x', pára comunicação --  Modelo de compilação: LARGE

#include <reg51.h>              /* endereços dos special function registers */
#include <stdlib.h>

void serial(void);

unsigned char xdata mempool [100];  // aloca 100 bytes no pool de memória
unsigned char xdata *regPtr;
unsigned char xdata *Reg1;

void main (void)  {

init_mempool (&mempool, sizeof(mempool)); //inicializa rotinas de memory management

regPtr = calloc (32, sizeof (unsigned char)); // aloca 32  dados do tipo unsigned char na memória
Reg1=regPtr;

  SCON  = 0x50;                 // SCON: modo 1, 8-bit, recepcao habilitada
  TMOD  |= 0x20;                // TMOD: timer 1, modo 2
  TH1   = 0xf4;                 // TH1:  valor de recarga para 2400 baud; clk = 11,059 MHz
  TR1   = 1;                    // TR1:  dispara timer

  ES    = 1;   EA    = 1;       // habilita interrupcao serial
  SBUF  = 0x41; ;               // envia ‘A'

  while (SBUF != 'x');          //enquanto não detecta um 'x' na porta serial,  executa...
   ES    = 0;           //  inibe interrupcao pela porta serial
   free (regPtr);
   while (1);                   //  necessário para evitar  travamento
                    }
```

```
void serial(void) interrupt 4 { // especifica interrupção serial (4)
      static unsigned char tm=0x42;       // incializa tm  com o ASCII =  'B'
      static unsigned char count = 0;

      if (RI) {          // testa se dado recebido - buffer de recepção cheio

                              RI=0;

                              *regPtr++ = SBUF;

                              count++;

                              if(count==32)    {regPtr = Reg1;  count=0;  };
            }

      if (TI) {          // testa se buffer de transmissão vazio

                              TI=0;
                              SBUF = tm;
                              tm++;
                              if  (tm==0x62)   tm=0x41;

            }
                               } // end of serial
```

SCON - Serial Port Control Register - Bit Addressable



			---------- Timer 1 ----------		
Baud Rate	**Fosc**	**SMOD**	**C/T**	**Mode**	**Reload Value**
Mode 0 Max: 1 MHz	**12 MHz**	**X**	**X**	**X**	**X**
Mode 2 Max: 375K	**12 MHz**	**1**	**X**	**X**	**X**
Modes 1, 3: 62,5K	**12 MHz**	**1**	**0**	**2**	**FFH**
19,2K	**11,059 MHz**	**1**	**0**	**2**	**FDH**
9,6K	**11,059 MHz**	**0**	**0**	**2**	**FDH**
4,8K	**11,059 MHz**	**0**	**0**	**2**	**FAH**
2,4K	**11,059 MHz**	**0**	**0**	**2**	**F4H**
1,2K	**11,059 MHz**	**0**	**0**	**2**	**E8H**
137,5K	**11,986 MHz**	**0**	**0**	**2**	**1DH**
110K	**6 MHz**	**0**	**0**	**2**	**72H**
110K	**12 MHz**	**0**	**0**	**1**	**FEEBH**

Algumas funções do C

abs	acos	asin	atan	atan2
atof	atoi	atol	calloc	ceil
cos	cosh	exp	fabs	floor
free	getchar	gets	isalnum	isalpha
iscntrl	isdigit	isgraf	islower	isprint
ispunct	isspace	isupper	isxdigit	labs
log	log10	longjmp	malloc	memchr
memcmp	memcpy	memmove	memset	modf
pow	printf	putchar	puts	rand
realloc	scanf	setjmp	sin	sinh
sprintf	sqrt	srand	sscanf	strcat
strchr	strcmp	strcpy	strcspn	strlen
strncat	strncmp	strncpy	strpbrk	strrchr
strspn	tan	tanh	tolower	toupper
va_arg	va_end	va_start		

Exemplo do uso de estrutura em C - I

```
#include <reg51.h>

#define     TC_MODO_0           0x0
#define     TC_MODO_1           0x1
#define     TC_MODO_2           0x2
#define     TC_MODO_3           0x3

#define     TC_CNT              0x4
#define     TC_TMR              0x0

#define     TC_GATE_HIGH        0x8
#define     TC_GATE_LOW         0x0

struct  TmodInitStruct  { char mode;    char cntr_tmr;   char gate; } Timer0, Timer1 ;

void  InitTimerFunction (void) {  // função para inicializar Timers 0 (modo 1)  e 1 (modo 2) como timers

char temp;

            Timer1.mode         =           TC_MODO_2;
            Timer1.cntr_tmr     =           TC_TMR;
            Timer1.gate         =           TC_GATE_LOW;

            temp        =           (Timer1.mode|Timer1.cntr_tmr|Timer1.gate) << 4;

            Timer0.mode         =           TC_MODO_1;
            Timer0.cntr_tmr     =           TC_TMR;
            Timer0.gate         =           TC_GATE_LOW;

            temp                |=          (Timer0.mode|Timer0.cntr_tmr|Timer0.gate);

            TMOD        =           temp;    }
```

Exemplo do uso de estrutura em C - II

```
void main (void) {
    char dado_P1=0x80;

// inicialização dos Timers 0 e 1 como timers nos modos 1 e 2, respectivamente

        InitTimerFunction();

// inicia contagem por parte do Timer 1

        TR1=1;

// complementa nível lógico do de P1.7 a cada estouro da contagem do timer1

        while (1) {
                while (!TF1);                    // polling
                TF1=0;
                P1 = P1^dado_P1;
            }
}
```

In-line Assembly

- Código Assembly pode ser inserido no código C para se obter maior velocidade ou aproveitar rotinas já escritas em Assembly.

// Pode ser necessário adicionar arquivo C51S.LIB junto com o programa fonte

// Em Options for file selecionar: Generate SRC, Assemble SRC

// http://www.keil.com/support/docs/2308.htm

….

#pragma asm

(código assembly aqui)

#pragma endasm

Interface de funções C-Assembly

Compartilhamento de variáveis globais se em outro código fonte

C

unsigned char eel7030;

Assembly
(deve-se conhecer o tipo da variável)

EXTERN DATA(eel7030)

...

MOV eel7030,#10

Link de C e Assembly

Compilador Keil C51

http://www.keil.com/support/docs/50.htm

1. Escrever uma função simples em C que recebe parâmetros e retorna valores compatíveis com a atividade a ser executada em Assembly pela função.

2. Utilize a diretiva #PRAGMA SRC no início do arquivo contendo a função para que o compilador C gere arquivo .SRC (em vez de arquivo .OBJ).

3. Compile. Há emissão de mensagem de erro (não encontra obj). Aquivo .SRC é gerado contendo código Assembly criado a partir da função escrita em C.

4. Edite o arquivo .SRC, inserindo o código Assembly que deseja ser executado pela função.

5. Renomeie arquivo .SRC para arquivo .asm. Incluir este último no projeto e exclua o .C original da função. Recompile o projeto.

Exemplo Link de C e Assembly

Compilador Keil C51

```
#include <reg51.h>


/* prototipo de funcao externa */
extern unsigned short   inc_arg(unsigned short);

void main(void)
{
unsigned short teste;

teste=0;

while(1) {

            teste=inc_arg(teste);  // função salva em outro arquivo...
                                   // ... lrá incrementar o parâmetro passado
          }

}
```

```
#pragma src  // função  que incrementa o argumento passado e o retorna
#pragma small                  // ao compilar, gera arquivo com extensão .SRC...
                                // ... que deve integrar o projeto com o main.

/* prototipo de funcao externa */

extern void main(void);

/* codigo da funcao */


unsigned short  inc_arg(unsigned short dado)
{
                        #pragma asm
                        MOV   A,B
                        #pragma endasm
return dado;
}
```

Listagem do arquivo .SRC gerado

Compilador Keil C51

```
; Assc_2.SRC generated from: Assc_2.c
; COMPILER INVOKED BY:
;        C:\Keil\C51\BIN\C51.EXE Assc_2.c BROWSE DEBUG OBJECTEXTEND


NAME     ASSC_2

?PR?_inc_arg?ASSC_2  SEGMENT CODE
?DT?_inc_arg?ASSC_2  SEGMENT DATA OVERLAYABLE
             PUBLIC   _inc_arg

             RSEG  ?DT?_inc_arg?ASSC_2
?_inc_arg?BYTE:
      dado?040:   DS   2
; #pragma src  // programa que incrementa o argumento passado e o retorna
; #pragma small                    // ao compilar, gera arquivo com extensão .SRC...
;                                  // ... que deve integrar o projeto com o main.
;
; /* prototipo de funcao externa */
;
; extern void main(void);
```

```
; /* codigo da funcao */
;
;
; unsigned short  inc_arg(unsigned short dado)

                RSEG  ?PR?_inc_arg?ASSC_2
_inc_arg:
                USING   0
                                        ; SOURCE LINE # 12
                MOV     dado?040,R6
                MOV     dado?040+01H,R7
; {
                                        ; SOURCE LINE # 13
;                       #pragma asm
;                       MOV   A,B
                 MOV   A,B
;                       #pragma endasm
; return dado;
                                        ; SOURCE LINE # 17
                MOV     R6,dado?040
                MOV     R7,dado?040+01H
; }
                                        ; SOURCE LINE # 18
?C0001:
                RET
; END OF _inc_arg

                END
```

```
; modificação do programa anterior para incrementar o parâmetro e retorná-lo à função principal
; /* codigo da funcao */
;
;
; unsigned short  inc_arg(unsigned short dado)

                RSEG  ?PR?_inc_arg?ASSC_2
_inc_arg:
                USING   0

;               MOV     dado?040,R6             COMENTAR
;               MOV     dado?040+01H,R7         COMENTAR

                        MOV   A,R7              ; inserir código
                        ADD   A,#01H
                        MOV   R7,A
                        CLR   A
                        ADDC  A,R6
                        MOV   R6,A
;
; return dado;

;               MOV     R6,dado?040             COMENTAR
;               MOV     R7,dado?040+01H         COMENTAR
; }

?C0001:
                RET
; END OF _inc_arg

                END
```

Segmentos do Código Macro-assembly Gerado

Compilador C51 Keil

Segment Prefix	Memory Type	Description
?PR?	**program**	Executable program code
?CO?	**code**	Constant data in program memory
?BI?	**bit**	Bit data in internal data memory
?BA?	**bdata**	Bit-addressable data in internal data memory
?DT?	**data**	Internal data memory
?FD?	**far**	Far memory (RAM space)
?FC?	**const far**	Far memory (constant ROM space)
?ID?	**idata**	Indirectly-addressable internal data memory
?PD?	**pdata**	Paged data in external data memory
?XD?	**xdata**	Xdata memory (RAM space)