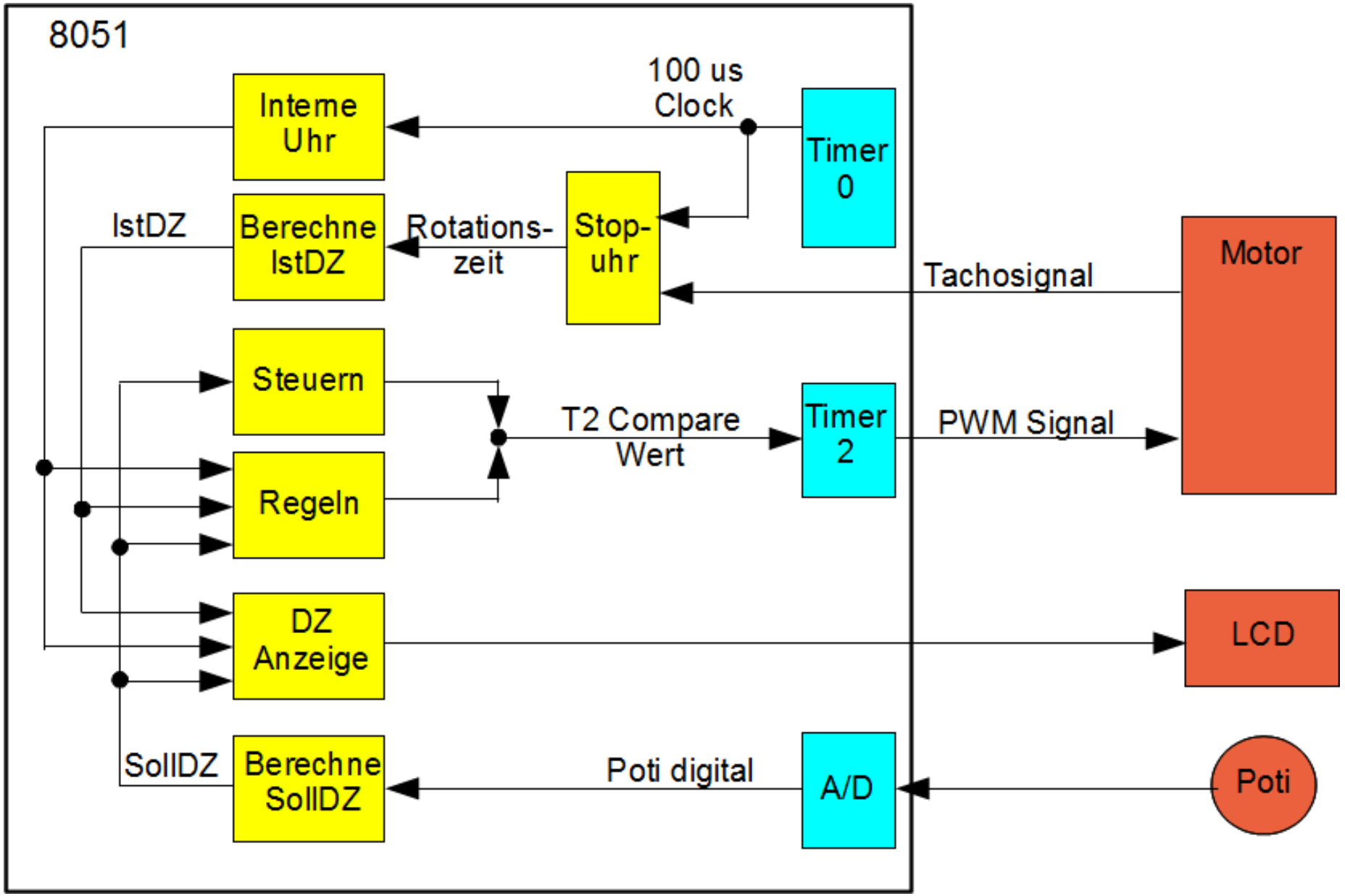
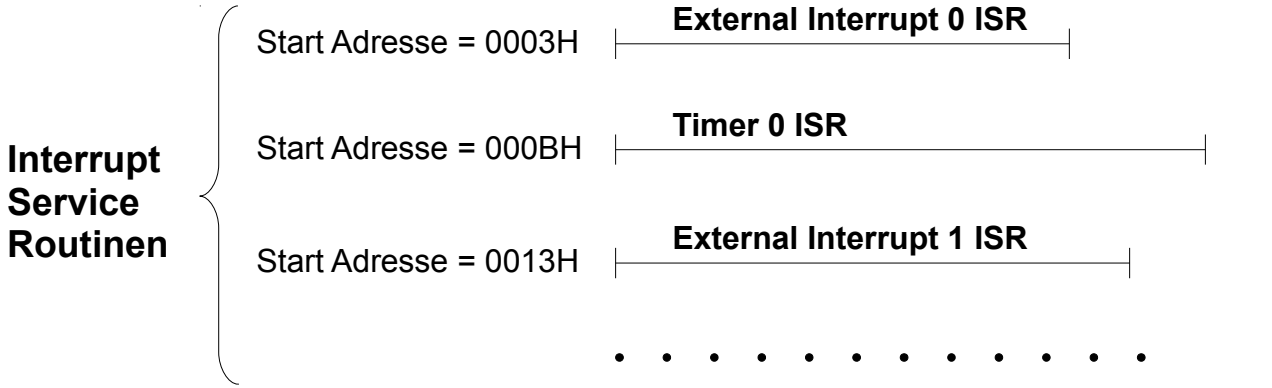
Projektstruktur

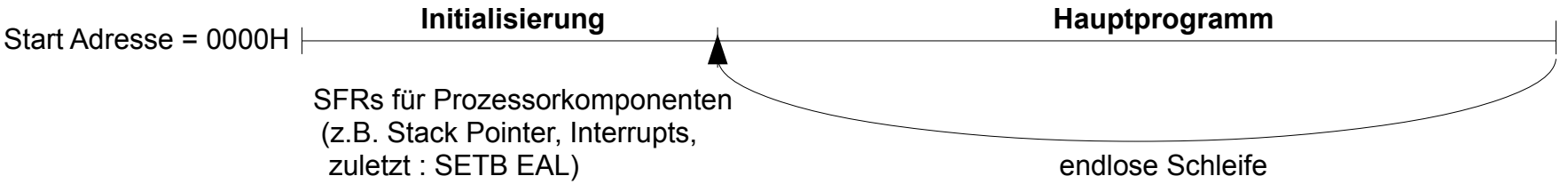


Typischer Aufbau eines 8051 Programms



(ISR Start Adressen siehe Script Tab. 6, S. 72)

Routinen auf der Basisebene
(Basic Level Routines)



SFRs für Prozessorkomponenten
(z.B. Stack Pointer, Interrupts,
zuletzt : SETB EAL)

Interne H/W-Komponenten
(z.B. Timer, A/D-Wandler)

Externe H/W-Komponenten
(z.B. LCD in der Laboraufgabe)

Anfangswerte von Variablen

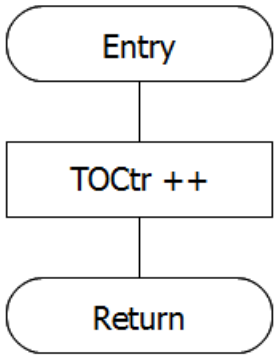
Zeitgerechte Befehlsausführung

Aufgabenstellung : Alle 16 ms soll eine bestimmte Routine ausgeführt werden

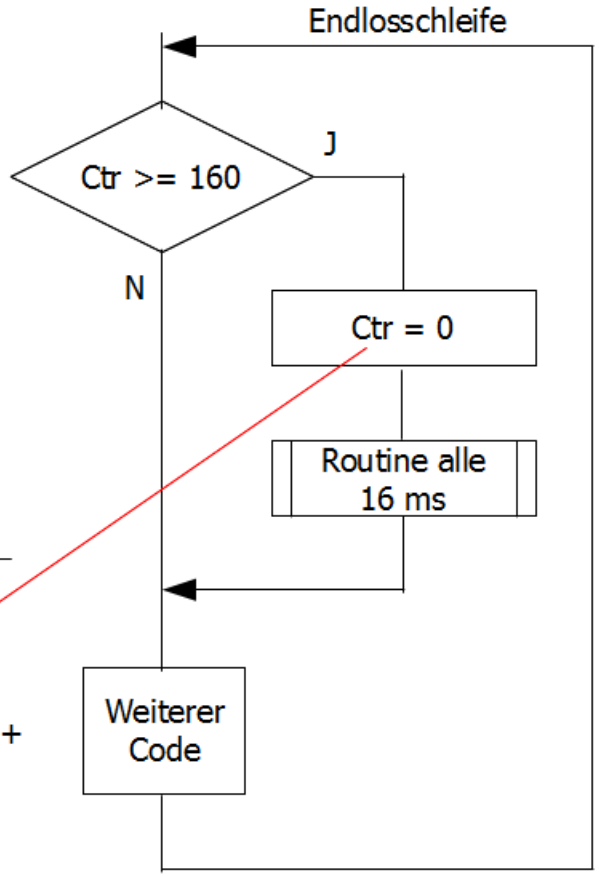
Timer 0 (oder 1)

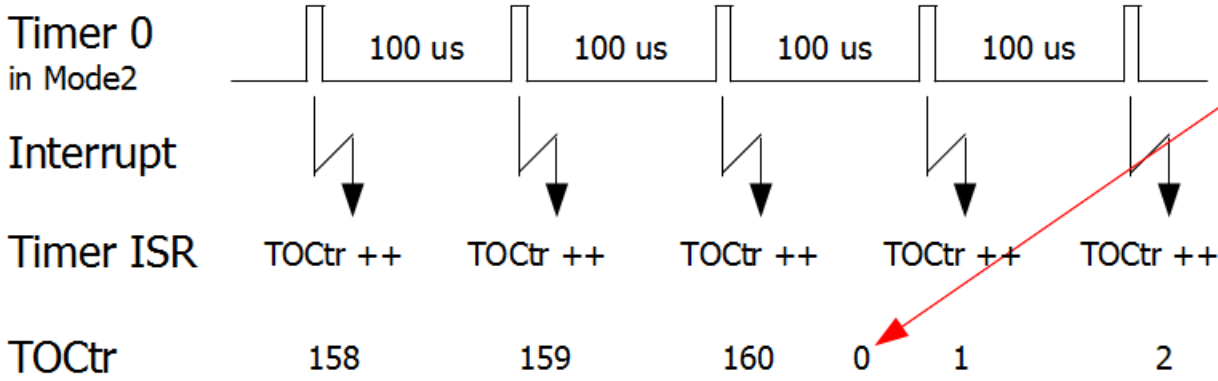
- Steuerung durch Oszillatortakt
- Oszillatorfrequenz 10 MHz
- Periode : 0,6 us
- Mode 2 : 8 Bit Timer --> 0 ... 150 us
  Reload automatisch

ISR

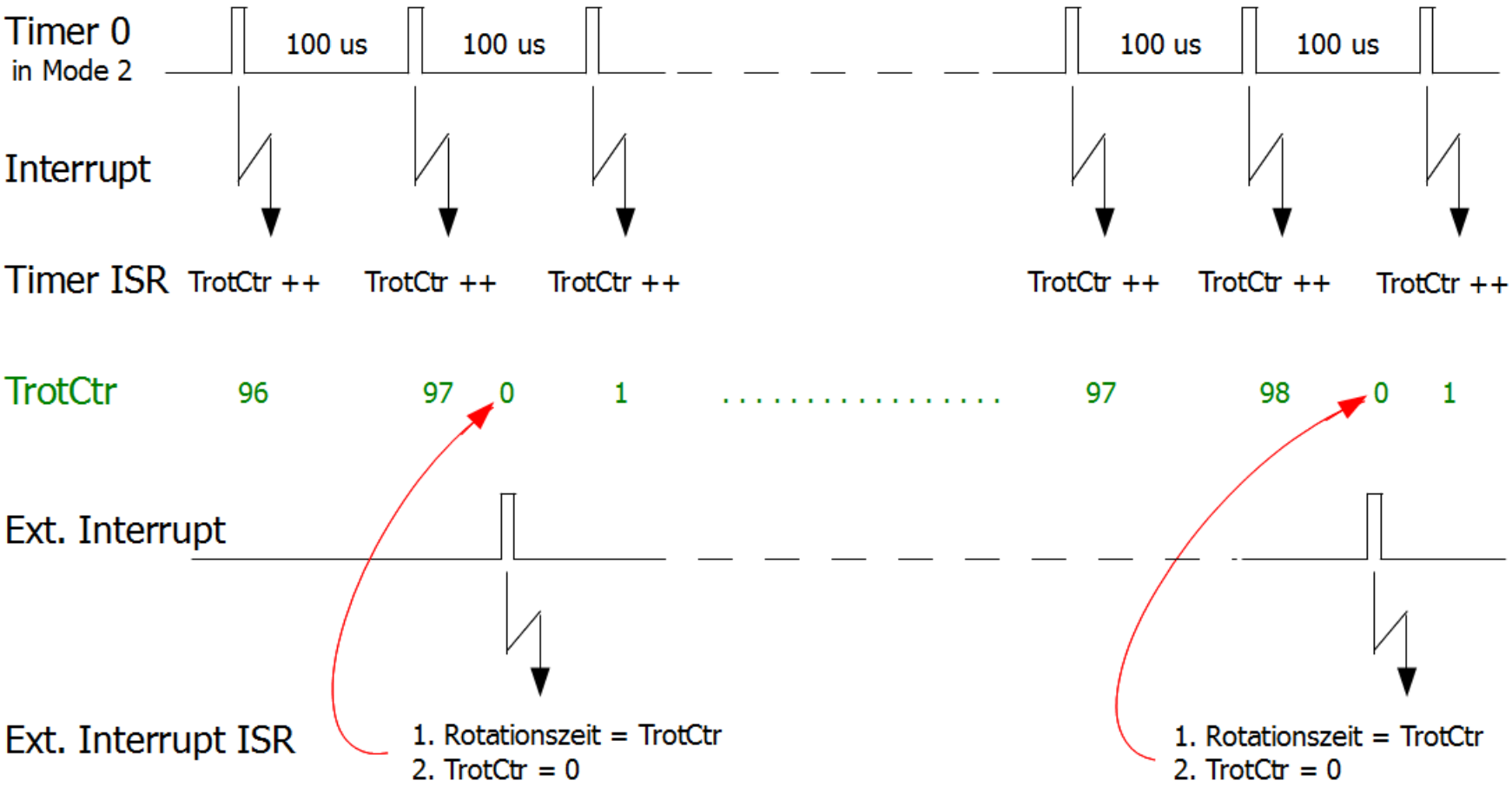


Hauptprogrammcode





Rotationszeitmessung



Gefahr für Datenintegrität

Beispiel : Rotationszeit Messen
Rotationszeit : 51,1 ms = 511 Ticks
2 Byte Zähler wird benötigt

Annahme : Der Timer Interrupt hat eine höhere Priorität als der Ext. Interrupt,
UND die Timer 1 ISR unterbricht die Ext.Int.ISR genau zwischen den beiden Befehlen, UND Ctr+1 hat den Wert 0xFF.

Timer ISR :

```
MOV   A,TrotCtr+1  ;FF
ADD   A,#1
MOV   TrotCtr+1,A  ;00
MOV   A,TrotCtr    ;01
ADDC  A,#0
MOV   TrotCtr,A    ;02
```

Ext.Interrupt ISR :

```
MOV   Trot+1,TrotCtr+1   (1)
MOV   Trot,TrotCtr       (3)
```

Wenn anfangs TrotCtr, TrotCtr+1 auf 0x01FF stehen,
liest die Ext.Interrupt ISR TrotCtr+1 = 0xFF aus, (1)
dann inkrementiert die Timer ISR den Zähler auf 0x0200, (2)
und dann liest die Ext.Interrupt ISR TrotCtr = 0x02 aus. (3)
Der korrekte Rot.zeit-Wert wäre aber 0x01FF, nicht 0x02FF.

Das Problem ist, dass die Operation mit niedriger Priorität nicht **atomisch** ist, also unterbrochen werden kann.

Abhilfe : Interrupts vor der kritischen Sequenz disablen und danach wieder enablen.

Segmente

- Für die logische Strukturierung eines Projekts werden Segmente verwendet.
- Zur Erinnerung : Es gibt Codesegmente (für Programmspeicherspezifikationen) und Datensegmente (für Datenspeicherspezifikationen).
  - Segmente für Datenspeicher gibt es für den :
    - direkt adressierbaren internen Datenspeicher (DSEG, DATA),
    - indirekt adressierbaren internen Datenspeicher (ISEG, IDATA,
    - externen Datenspeicher (XSEG, XDATA),
    - Bitspeicherbereich (BSEG, BIT).
- Zur Erinnerung : Es gibt absolute und relokative Segmente
  - bei absoluten Segmenten spezifiziert der Programmierer die Startadresse des Segments (z.B. CSEG AT 0)
  - bei relokativen Segmenten legt der Linker die Startadressen fest (Relokation).
- Die folgenden Bilder illustrieren die Verwendung von Symbolen und der zugehörigen Referenzen.

Programmsymbol und Referenz

Datei1.a51

```
PSeg1  SEGMENT   CODE
       RSEG      PSeg1
          :
          :
Label1:
          :
          :
       LJMP      Label1
          :
          :
```

Datensymbol und Referenz

Datei1.a51

```
PSeg1  SEGMENT   CODE
       RSEG      PSeg1
          :
          :
Label1:
          :
          :
       MOV       A,OneBy
          :
          :
```

```
DSeg1  SEGMENT   DATA
       RSEG      DSeg1
          :
          :
          :
          :
OneBy  DS        1
TwoBy  DS        2
          :
          :
```

Referenzen in anderer Datei

Datei1.a51

```
            PUBLIC      Label1
            PUBLIC      TwoBy

PSeg1       SEGMENT     CODE
            RSEG        PSeg1
                :
            MOV         A,OneBy
                :
Label1:
                :
            LJMP        Label1
                :
```

```
DSeg1       SEGMENT     DATA
            RSEG        DSeg1
                :

                :
                :
OneBy       DS          1
TwoBy       DS          2
                :
                :
```

Datei2.a51

```
            EXTRN     CODE(Label1)
            EXTRN     DATA(TwoBy)

PSeg2       SEGMENT   CODE
            RSEG      PSeg2
                :
                :
            MOV       A,TwoBy
                :
                :
            LJMP      Label1
                :
```

Aufruf einer C-Funktion aus Assemblercode

Datei1.a51

```
        EXTRN   CODE(CFunc1)

PSeg1   SEGMENT  CODE
        RSEG     PSeg1
            :
            :
Label1:
            :
            :
        CALL     CFunc1
            :
            :
```

Datei3.c

```
void    CFunc1      void;
void    CFunc1      void
  {
            :
            :
            :
            :
            :
  }
```

Gemeinsame Variable für C-Funktion und Assemblercode

Datei1.a51

```
        EXTRN   CODE(CFunc1)
        PUBLIC  OneBy,TwoBy

PSeg1   SEGMENT  CODE
        RSEG     PSeg1
            :
            :
Label1:
            :
        MOV      OneBy,#15
        CALL     CFunc1
            :
```

```
DSeg1   SEGMENT  DATA
        RSEG     DSeg1
            :

            :
            :
OneBy   DS       1
TwoBy   DS       2
            :
            :
```

Datei3.c

```
extern char OneBy;
extern int TwoBy;

void    CFunc1      void;
void    CFunc1      void
  {
              :
   TwoBy ++;
              :
              :
              :
  }
```

Parameterübergabe über Register

Datei1.a51

```
        EXTRN   CODE(_lcd_curs)

PSeg1  SEGMENT  CODE
        RSEG    PSeg1
            :
            :
Label1:
            :
        MOV     R7,#20
        CALL    _lcd_curs
            :
            :
```

Parameter passing in Register

lcd4x20v2.c

```
void lcd_curs(char);
void lcd_curs(unsigned char pos)
  {
            :
            :
            :
            :
            :
  }
```

Details in KEIL uVision4 User's Manual :
Cx51 Compiler User's Guide → Advanced Programming → Interfacing C to Assembler → Function Parameters → Passing in Registers