# Programmentwurf

## Systemnahe Programmierung I

Task-Verwaltung

Michael Strobel
Ernesto Elsäßer

TINF11B
4. Halbjahr
2013

# Inhalt

1 Einleitung ..... 3
2 Behandlung von doppelten Prozess-Aufrufen ..... 3
3 Kontroll-Ausgabe ..... 3
4 Beschreibung wichtiger Programmteile ..... 4
5 Kommentiertes Listing ..... 7

# 1 Einleitung

Die Projektaufgabe hat zum Ziel, ein Multiprozess-System auf einem 8051-Mikroprozessor zu realisieren. Die einzelnen Prozesse sollen unabhängig voneinander quasi-parallel ablaufen, wobei sie sich nicht kooperativ verhalten. Ein Scheduler kümmert sich um eine gerechte Verteilung der Prozessorzeit unter den aktiven Prozessen. Als Zuteilungs-Verfahren kommt beispielsweise die Zeitscheiben-Steuerung infrage, bei welcher der aktive Prozess nach einer bestimmten Zeitspanne gewechselt wird.

Folgende Prozesse sind enthalten:

Der Ausgabe-Prozess A gibt ca. einmal pro Sekunde ein 'a' auf der seriellen Schnittstelle aus.

Der Ausgabe-Prozess B gibt einmalig '54321' auf der seriellen Schnittstelle aus. Danach beendet er sich.

Der Konsolenprozess liest Zeichen auf der seriellen Schnittstelle ein. Bei 'a' und 'b' wird der Ausgabe-Prozess A gestartet bzw. beendet. Ein 'c' startet den Ausgabe-Prozess B. Werden andere Zeichen eingegeben, führt der Prozess keine Aktion aus.

# 2 Behandlung von doppelten Prozess-Aufrufen

Wird ein Prozess etwa durch Eingabe von 'aa' mehrfach aufgerufen, wird er zurückgesetzt und neu gestartet.

Das StartProcess-Unterprogramm läuft unabhängig davon ab, ob der zu startende Prozess schon gestartet wurde. Es setzt den Eintrag in der Prozesstabelle auf aktiv, schreibt die Startadresse des Prozesses in dessen Stack-Bereich und setzt dessen gesicherte Register zurück. Dies dient in erster Linie dazu, einen bereits beendeten Prozess beim erneuten Starten wieder komplett zurückzusetzen. Es hat allerdings auch denselben Effekt auf bereits laufende Programme.

# 3 Kontroll-Ausgabe

54321aaaa54321aaa54321aaaaaaa54321aa

# 4 Beschreibung wichtiger Programmteile

**main.a51**

Die Datei main.a51 ist die Hauptdatei des Projektes. Dort wird im Wesentlichen der Prozessor konfiguriert und der Scheduler gestartet.

Zuerst wird das Timer 0-Interrupt auf die Scheduler-Funktion geleitet und die Einstellungen für die verwendeten Timer sowie die serielle Schnittstelle gesetzt. Anschließend wird der Konsolenprozess gestartet, indem das Unterprogramm *startProcess* mit 0 als Parameter aufgerufen wird. Ist dies geschehen, wird der Timer für den Scheduler aktiviert und dessen Interrupt-Flag gesetzt, um ihn sofort zu starten.

Am Ende der Datei befindet sich eine Endlosschleife, die die Zeit abwarten soll, bis der Scheduler einsetzt und der Konsolenprozess zum ersten Mal aufgerufen wird.

**seriell.a51**

In dieser Datei ist das Unterprogramm *serialSend* zum Senden eines Bytes über die serielle Schnittstelle enthalten. Das zu übertragende Byte muss vor dem Aufruf in den Akkumulator geladen werden.

Das Unterprogramm deaktiviert während des Sendens alle Interrupts, um zu gewährleisten, dass die Übertragung nicht unterbrochen wird.

Der Inhalt des Akkumulators wird in das Register der seriellen Schnittstelle *S0BUF* geschrieben. Danach wartet das Unterprogramm so lange, bis die Übertragungs-Flag *TI0* gesetzt wurde.

**console.a51**

Der Konsolen-Prozess überprüft per Polling, ob an der seriellen Schnittstelle eine Eingabe vorliegt. Ist das der Fall, wird das eingelesene Zeichen analysiert. Handelt es sich um ein a (ASCII-Code 97), wird der Ausgabe-Prozess A über das *startProcess*-Unterprogramm mit dem Parameter 1 gestartet; bei einem b wird er über das *stopProcess*-Unterprogramm wieder beendet. Ein c startet den Ausgabe-Prozess B mit dem Parameter 2. Bei einem anderen Zeichen erfolgt keine Aktion.

Ist dieser Vorgang abgeschlossen, wiederholt sich der Prozess als Endlosschleife.

**ausgabea.a51**

Beim Ausgabe-Prozess A wird der Timer 1 per Polling überprüft, welcher als 16 Bit-Timer konfiguriert ist. Um auf das Intervall von etwa einer Sekunde zu kommen, muss der Timer eine bestimmte Anzahl oft auslösen, bevor das a über das *serialSend*-Unterprogramm ausgegeben wird. Dazu wird das Register R0 mit 0x1E geladen und bis auf 0 heruntergezählt.

Nach dem Senden wird das Register wieder zurückgesetzt, der Prozess ist ebenfalls als Endlosschleife implementiert.

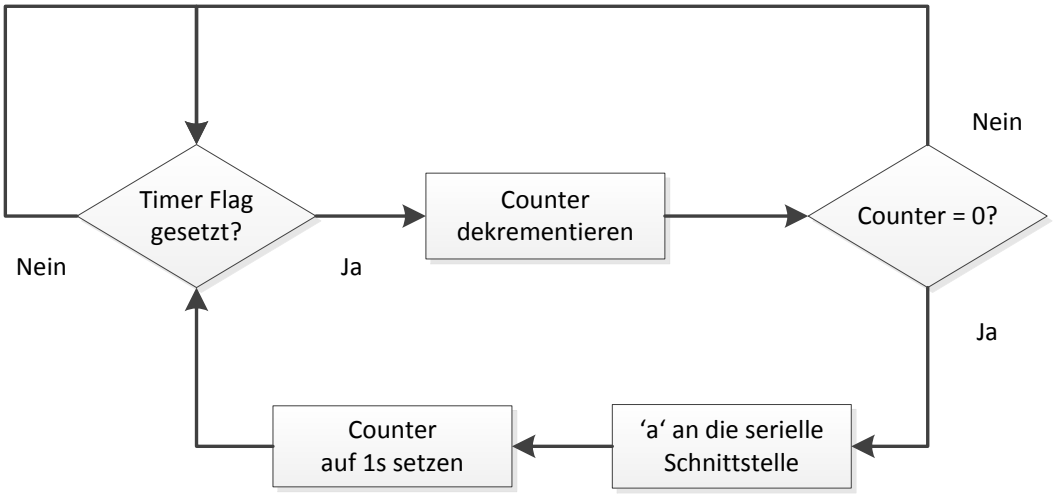


**Abbildung 1: Prozess A**

### ausgabeb.a51

Der Ausgabe-Prozess B zählt im Register R0 von 5 bis 1 herunter. Bei jedem Durchlauf wird der aktuelle Wert des Registers über das *serialSend*-Unterprogramm auf der seriellen Schnittstelle ausgegeben.

Ist die Schleife durchgelaufen, beendet sich der Prozess durch Aufruf von *stopProcess* mit 2 als Parameter selbst.

### scheduler.a51

Die Datei scheduler.a51 enthält neben dem Prozess-Scheduler die Unterprogramme *startProcess* und *stopProcess* zum Starten bzw. Beenden eines Prozesses.

Am Anfang der Datei werden Konstanten für die Scheduler-Konfiguration festgelegt, Speicherplatz für Status- und Sicherungs-Variablen reserviert sowie die Startadressen der Prozesse im Code-Speicher hinterlegt.

Der Scheduler, der durch das Timer 0-Interrupt aufgerufen wird, setzt zu Beginn den Watchdog zurück. Anschließend werden die Register A, B und R0 in zuvor reservierten Variablen zwischengespeichert, da der Scheduler sie zur Adressberechnung und zum Sichern der anderen Register benötigt. Die zwischengespeicherten Register werden nun zusammen mit allen anderen (insgesamt SP, A, B, PSW, DPH, DPL, R0-R7) in den Sicherungsbereich des Prozesses gespeichert. Beim ersten Ausführen des Schedulers wird dieser Schritt übersprungen.

Danach wird der Folgeprozess ausgewählt. Dies ist immer der nächste aktive Prozess in der Prozesstabelle. Die Prioritäten der einzelnen Prozesse werden über die Dauer ihrer Zeitscheibe realisiert, indem die Konfiguration des Timers bei jedem Prozesswechsel geändert wird. Nun werden die zuvor gespeicherten Register des nachfolgenden Prozesses wiederhergestellt. A, B und R0 werden ebenfalls zuerst in die reservierten Variablen zwischengespeichert und diese zum Schluss wiederhergestellt.

Der Scheduler arbeitet nach folgendem Schema:

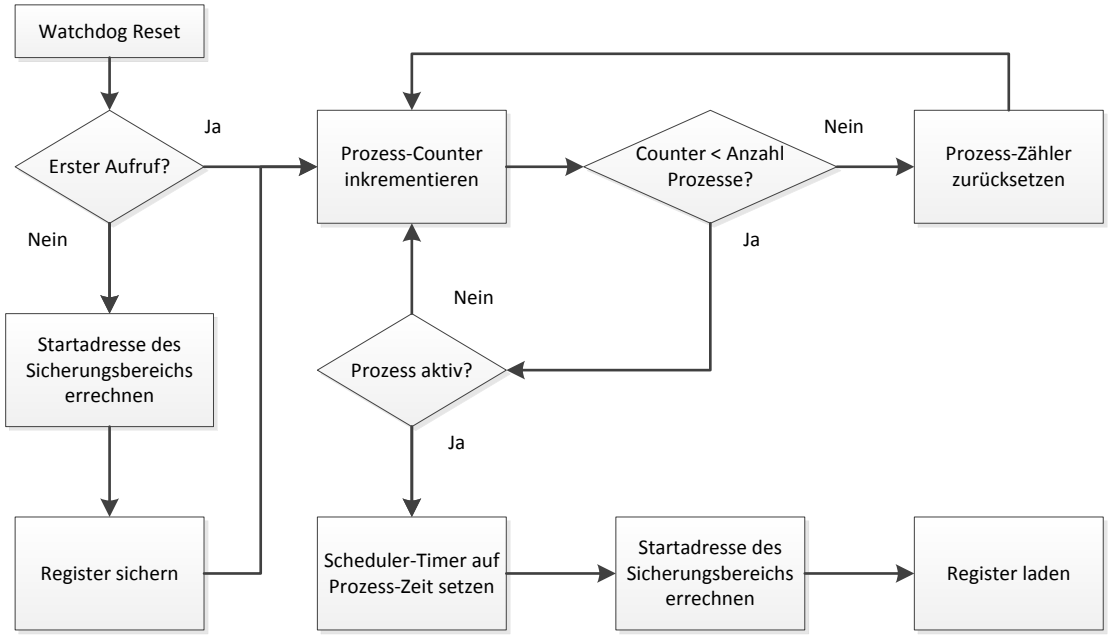


**Abbildung 2: Scheduler**

Das Unterprogramm *startProcess* startet einen Prozess. Der Index des Prozesses wird als Parameter über den Akkumulator übergeben. Der Prozess-Index dient zur Berechnung der Offsets innerhalb der Variablen für die Prozesstabelle, den Stack-Bereich und den Sicherungsbereich.

Als erstes wird die Start-Adresse des Prozesses in dessen Stackbereich geschrieben. Darauf folgt das Zurücksetzen des Sicherungs-Bereichs: Der Stack Pointer wird auf den Stack-Bereich des Prozesses gesetzt, alle anderen Register werden auf 0 zurückgesetzt.

Zum Schluss wird der Prozess in der Prozesstabelle als aktiv markiert und kann beim nächsten Durchlauf des Schedulers ausgewählt werden.

Das *stopProcess*-Unterprogramm erhält den Prozess-Index ebenfalls als Parameter über den Akkumulator. Es setzt den Eintrag in der Prozesstabelle wieder auf 0 zurück. Anschließend wird die Timer 0-Flag wieder gesetzt, um den Scheduler sofort zu starten. Dies dient dazu, sofort einen anderen Prozess auszuwählen, wenn der gerade aktive Prozess beendet wird (z.B. Ausgabe-Prozess B).

# 5 Kommentiertes Listing

### Listing 1: main.a51

```
; C517A-Symbole verfügbar machen
$NOMOD51
#include <Reg517a.inc>

NAME main

; Symbole aus den Modulen importieren
EXTRN CODE (scheduler, startProcess)

; Variablen anlegen
dataSegment    SEGMENT DATA
RSEG dataSegment

STACK:    DS    4


; Interrupt-Routinen definieren
CSEG

ORG       0x0B  ; Timer 0 Interrupt
JMP       scheduler

; Systemstart-Anweisungen
ORG 0
JMP       start


codeSegment SEGMENT CODE
RSEG codeSegment

start:

;
; Prozessor-Konfiguration
;

; Interrupt-Flags
SETB EAL        ; Interrupts global aktivieren
SETB ET0        ; Timer 0-Interrupt für den Scheduler

; Timer-Konfiguration
MOV       TMOD,#00010000b ; Timer 1: 16 Bit Timer, Timer 2: 8 Bit Timer
SETB TR1        ; Timer 1 für Prozess A aktivieren

; Serial Mode 1: 8bit-UART bei Baudrate 9600
CLR       SM0
SETB SM1

SETB REN0             ; Empfang ermöglichen
SETB BD               ; Baudraten-Generator aktivieren
MOV       S0RELL,#0xD9      ; Baudrate einstellen
MOV       S0RELH,#0x03      ; 9600 = 03D9H

; Stack Pointer auf reservierten Bereich setzen
```

```
MOV       SP,#STACK


;
; Initialisierungs-Programm
;

; Konsolenprozess starten
MOV       A,#0 ; Prozess 0
MOV       B,#0 ; höchste Priorität
CALL startProcess

; Timer 0 für Scheduler-Interrupt
SETB TR0

; Scheduler-Interrupt starten
SETB TF0

infiniteLoop:

     NOP
JMP       infiniteLoop

END
```

**Listing 2: seriell.a51**

```
; C517A-Symbole verfügbar machen
$NOMOD51
#include <Reg517a.inc>

; Symbol-Exporte
NAME seriell
PUBLIC    serialSend


codeSegment SEGMENT CODE
RSEG codeSegment

;
; Sendet ein Byte vom Akkumulator auf den seriellen Port
;
serialSend:

     ; Interrupts für die Dauer der seriellen Übertragung deaktivieren
     CLR        EAL

     ; Daten schreiben
     MOV        S0BUF,A

     ; warten, bis Senden abgeschlossen (TI0 gesetzt wurde)
     sendWait:
          NOP
          JNB         TI0,sendWait

     CLR        TI0         ; nach Senden TI0 zurücksetzen

     ; Interrupts wieder aktivieren
     SETB EAL
```

```
RET

END
```

**Listing 3: console.a51**

```
; C517A-Symbole verfügbar machen
$NOMOD51
#include <Reg517a.inc>

; Symbol-Im- und -Exporte
NAME console
PUBLIC   processConsole
EXTRN CODE (startProcess, stopProcess)

codeSegment SEGMENT CODE
RSEG codeSegment

;
; Liest Zeichen von der seriellen Schnittstelle ein
; a: Startet Prozess AusgabeA
; b: Beendet Prozess AusgabeA
; c: Startet Prozess AusgabeB
; anderes Zeichen: Keine Aktion
;
processConsole:

     ; Serielle Schnittstelle durch Polling auslesen
     JNB        RI0,processConsole

     MOV        A,S0BUF          ; Daten auf Port1 lesen
     CLR        RI0              ; Empfangs-Flag wieder löschen

     ;
     ; Gelesenes Zeichen analysieren
     ;

     CJNE A,#97,consoleNotA

     ; a gelesen: Prozess AusgabeA starten

     MOV        A,#1
     MOV        B,#0xff ; niedrigste Priorität
     CALL startProcess

     JMP        processConsole

     consoleNotA:

     CJNE A,#98,consoleNotB

     ; b gelesen: Prozess AusgabeA beenden

     MOV        A,#1
     CALL stopProcess

     JMP        processConsole
```

```
     consoleNotB:

     CJNE A,#99,processConsole

     ; c gelesen: Prozess AusgabeB starten

     MOV        A,#2
     MOV        B,#0x88
     CALL startProcess

     JMP processConsole

END
```

**Listing 4: ausgabea.a51**

```
; C517A-Symbole verfügbar machen
$NOMOD51
#include <Reg517a.inc>

; Symbol-Im- und -Exporte
NAME ausgabea
EXTRN     CODE  (serialSend)
PUBLIC    processAusgabeA

codeSegmentPA SEGMENT CODE
RSEG codeSegmentPA

;
; Sendet jede Sekunde ein 'a' auf dem seriellen Port
;
processAusgabeA:

     ; sofort erstes 'a' senden
     MOV R0,#1

processAloop:

     ; Timer 1 Polling
     JNB  TF1,processAloop
     CLR        TF1

     ; Counter für 1s
     ; Rechnung mit 24MHz Takt:
     ; ((((24 * 10^6) / 12) / 2^16) / 30) = 1,01..
     DJNZ R0,processAloop
     MOV        R0,#0x1E

     MOV        A,#97 ; 'a' ASCII
     CALL serialSend

     JMP        processAloop

RET

END
```

**Listing 5: ausgabeb.a51**

```
; C517A-Symbole verfügbar machen
$NOMOD51
#include <Reg517a.inc>

; Symbol-Im- und -Exporte
NAME ausgabeb
EXTRN     CODE  (serialSend, stopProcess)
PUBLIC    processAusgabeB


codeSegment SEGMENT CODE
RSEG codeSegment

;
; Sendet einmalig 54321 auf dem seriellen Port
;
processAusgabeB:

     MOV        R0,#53      ; 53 == 5 ASCII

     processBLoop:

          MOV         A,R0
          CALL  serialSend

          DEC         R0

     CJNE R0,#48,processBLoop


     ; Prozess beenden
     MOV        A,#2
     CALL stopProcess

RET

END
```

**Listing 6: scheduler.a51**

```
; C517A-Symbole verfügbar machen
$NOMOD51
#include <Reg517a.inc>

; Symbol-Im- und -Exporte
NAME scheduler
PUBLIC    scheduler, startProcess, stopProcess
EXTRN     CODE  (processConsole, processAusgabeA, processAusgabeB)


; Konstanten, Scheduler-Konfiguration
numberProcesses EQU   3     ; Anzahl maximal verwalteter Prozesse
stackSize       EQU   4     ; Stack-Bereich pro Prozess je 4 Bytes
statusSize            EQU   14    ; Status-Bereich pro Prozess je 14 Bytes
```

```
; Variablen
dataSegment SEGMENT DATA
RSEG dataSegment

processTable:   DS    numberProcesses   ; Welche Prozesse sind gerade
     aktiv? (je 1 Byte)
processTime:    DS    numberProcesses   ; Zeitscheiben-Länge der Prozesses
     - Priorität (je 1 Byte)
currentProcess: DS    1     ; Welcher Prozess läuft gerade?
processStack:   DS    numberProcesses*stackSize     ; Stack für alle
     Prozesse

; Gesicherte Register für alle Prozesse
; SP,A,B,PSW,DPH,DPL,R0..R7
processStatus: DS     numberProcesses*statusSize

; Zwischen-Sicherungsvariablen für A, B und R0
backupA:        DS    1
backupB:        DS    1
backupR0:       DS    1

firstRun:       DS    1     ; Flag, ob der Scheduler bereits gelaufen ist


codeSegment SEGMENT CODE
RSEG codeSegment

; Start-Adressen der Prozesse
; Anzahl muss mindestens der von numberProcesses entsprechen
processLocations: DW processConsole, processAusgabeA, processAusgabeB


;
; Prozess-Scheduler
;
scheduler:

     CLR TR0 ; Scheduler-Timer anhalten

     ; Watchdog-Reset
     ; muss periodisch ausgeführt werden, sonst setzt der Watchdog die
     CPU zurück
     SETB WDT
     SETB SWDT

     ; A, B und R0 vorsichern, da zur Offset-Berechnung benötigt
     MOV        backupA,A
     MOV        backupB,B
     MOV        backupR0,R0

     ; Sicherung überspringen, wenn der Scheduler zum ersten mal läuft
     MOV        A,firstRun
     CJNE A,#0xff,schedulerFindProcess

     MOV        A,currentProcess

     ; Status des Prozesses sichern
     MOV        B,#statusSize     ; Größe des Status-Bereichs pro Prozess
```

```
MUL         AB
ADD         A,#processStatus
MOV         R0,A

; R0: Startadresse des Statusbereichs (alter Prozess)

; Reihenfolge: SP,A,B,PSW,DPH,DPL,R0..R7

MOV         @R0,SP
INC         R0
MOV         @R0,backupA
INC         R0
MOV         @R0,backupB
INC         R0
MOV         A,PSW
MOV         @R0,A
INC         R0
MOV         A,DPH
MOV         @R0,A
INC         R0
MOV         A,DPL
MOV         @R0,A
INC         R0
MOV         @R0,backupR0
INC         R0
MOV         A,R1
MOV         @R0,A
INC         R0
MOV         A,R2
MOV         @R0,A
INC         R0
MOV         A,R3
MOV         @R0,A
INC         R0
MOV         A,R4
MOV         @R0,A
INC         R0
MOV         A,R5
MOV         @R0,A
INC         R0
MOV         A,R6
MOV         @R0,A
INC         R0
MOV         A,R7
MOV         @R0,A

; Nächsten Prozess auswählen
schedulerFindProcess:

     ; Prozesse 0,1 und 2 durchlaufen
     INC         currentProcess
     MOV         A,currentProcess

     CJNE  A,#numberProcesses,schedulerNoReset
     MOV         currentProcess,#0 ; Zähler zurücksetzen

schedulerNoReset:

     ; Überprüfen ob currentProcess aktiv ist
```

```
    MOV         A,#processTable
    ADD         A,currentProcess
    MOV         R0,A

    CJNE  @R0,#0xff,schedulerFindProcess ; wenn nicht, weitersuchen

; Zeitscheibe konfigurieren
MOV        A,#processTime
ADD        A,currentProcess
MOV        R0,A
MOV        A,@R0
MOV        TH0,A

; Status des Prozesses wiederherstellen
MOV        A,currentProcess
MOV        B,#statusSize     ; Größe des Status-Bereichs pro Prozess
MUL        AB
ADD        A,#processStatus
MOV        R0,A

; R0: Startadresse des Statusbereichs (neuer Prozess)

MOV        A,@R0
MOV        SP,A
INC        R0
MOV        backupA,@R0
INC        R0
MOV        backupB,@R0
INC        R0
MOV        A,@R0
MOV        PSW,A
INC        R0
MOV        A,@R0
MOV        DPH,A
INC        R0
MOV        A,@R0
MOV        DPL,A
INC        R0
MOV        backupR0,@R0
INC        R0
MOV        A,@R0
MOV        R1,A
INC        R0
MOV        A,@R0
MOV        R2,A
INC        R0
MOV        A,@R0
MOV        R3,A
INC        R0
MOV        A,@R0
MOV        R4,A
INC        R0
MOV        A,@R0
MOV        R5,A
INC        R0
MOV        A,@R0
MOV        R6,A
INC        R0
MOV        A,@R0
```

```
    MOV       R7,A

    ; an anderer Stelle zwischengespeicherte Werte für A, B und R0
    einsetzen
    MOV       A,backupA
    MOV       B,backupB
    MOV       R0,backupR0

    ; Flag setzen, dass der Scheduler durchgelaufen ist
    MOV       firstRun,#0xff

    SETB TR0 ; Scheduler-Timer starten

RETI


;
; Prozess starten
; A: Prozess-Index
;    0 = console
;    1 = ausgabea
;    2 = ausgabeb
; B: Zeitscheibe
;    TH0 des Timers
;    höherer Wert -> kürzere Zeitscheibe
;
startProcess:

    ; R7: Zwischenspeicher für Prozess-Index
    MOV       R7,A

    ; Zeitscheibendauer
    ADD       A,#processTime
    MOV       R0,A
    MOV       @R0,B

    ; Stack-Adresse ermitteln
    MOV       A,R7
    MOV       B,#stackSize      ; Größe des Stack-Bereichs pro Prozess
    MUL       AB
    ADD       A,#processStack
    MOV       R1,A

    ; R1: Stack-Startadresse des Prozesses

    ; Prozess-Startadresse ermitteln
    MOV       A,R7
    MOV       B,#2  ; jede Prozess-Adresse belegt 2 Byte
    MUL       AB
    MOV       R6,A  ; R6: Zwischenspeicher für den Adress-Offset des
    Prozesses
    MOV       DPTR,#processLocations

    MOVC A,@A+DPTR   ; High Byte auslesen und in R5 speichern
    MOV       R5,A

    MOV       A,R6  ; Offset für zweites Byte erhöhen
    INC  A
```

```
    MOVC A,@A+DPTR   ; Low Byte auslesen

    ; Adresse in den DPTR schreiben
    MOV        DPL,A
    MOV        DPH,R5

    ; R5: High Byte der Prozess-Adresse
    ; R6: Offset der Prozess-Adresse

    MOV        @R1,DPL
    INC        R1
    MOV        @R1,DPH

    MOV        A,R7

    ; Status des Prozesses zurücksetzen
    MOV        B,#statusSize     ; Größe des Status-Bereichs pro Prozess
    MUL        AB
    ADD        A,#processStatus
    MOV        R0,A

    ; R0: Startadresse des Statusbereichs

    MOV        A,R1
    MOV        @R0,A ; Stack auf Anfang setzen

    MOV        A,R0
    INC        R0
    MOV        R1,#1

    ; R1: Zählvariable 1-14

    startProcessStatusResetLoop:
        MOV         @R0,#0
        INC         R0
        INC         R1
    CJNE R1,#statusSize,startProcessStatusResetLoop


    ; Eintrag in Prozess-Tabelle aktivieren
    MOV        A,R7
    ADD        A,#processTable
    MOV        R0,A
    MOV        @R0,#0xff

RET

;
; Prozess beenden
; A: Prozess-Index
;    0 = console
;    1 = ausgabea
;    2 = ausgabeb
;
stopProcess:

    ; setzt den Eintrag in der Prozesstabelle zurück
    MOV        B,A
    ADD        A,#processTable
```

```
    MOV         R0,A
    MOV         @R0,#0
    MOV         A,B

    ; Scheduler-Interrupt starten
    SETB TF0

RET


END
```