Programmentwurf

Systemnahe Programmierung

Task-Verwaltung

7455429

1220227

TINF14B

4. Semester

2016

Inhaltsverzeichnis

Einleitung

In diesem Projekt wird ein Task-Verwaltungs-Programm (Scheduler) für einen 8051-Mikrocontroller realisiert. Verschiedene Prozesse sollen unabhängig voneinander "parallel" ablaufen können, wobei jeder Prozess nach einem Zeitscheibenverfahren anteilig Prozessorzeit erhält. Der Scheduler soll nun die Verteilung der Prozessorzeit verwalten. Prozesse können unterschiedliche Prioritäten haben. Dies wird über die Dauer der jeweiligen Zeitscheibe realisiert.

Die einzelnen Prozesse sollen folgendes Verhalten aufzeigen:

- Es gibt einen Konsolenprozess, der auf der seriellen Schnittstelle 0 Zeichen einliest und dementsprechend Prozesse startet und stoppt. Bei dem Zeichen 'a' soll Prozess A gestartet werden. Die Eingabe 'b' startet Prozess B wobei die Eingabe 'c' den Prozess B wieder stoppen soll. Die Eingabe 'z' startet den Prozess Z, welcher danach endlos läuft.
- Der Prozess A gibt auf der seriellen Schnittstelle die Zeichenfolge 'abcde' aus und beendet sich danach.
- Der Prozess B soll in einem einsekündigen Zyklus das Zeichen '+' auf der seriellen Schnittstelle 0 ausgeben. Die Ausgabe soll nur so lange erfolgen, wie der Prozess aktiv ist.
- Der Prozess Z ist ein gegebener Prozess, der eingebunden wird, und auf der seriellen Schnittstelle 1 Ausgaben konfiguriert und erzeugt.

Beschreibung der Programmteile

"main.a51"

In der Hauptdatei ("main.a51") der Projektaufgabe wird das Programm initialisiert und die nötigen Einstellungen der Timer und der seriellen Schnittstellen vorgenommen. Außerdem ist hier festgelegt, welche Routinen bei einem Timer Interrupt aufgerufen werden. Nachdem die Konfiguration der Timer und der seriellen Schnittstelle abgeschlossen ist, wird der Stackpointer in einen reservierten Bereich geschrieben, sodass dieser nicht überschrieben werden kann. Das Problem des ersten Startens wird durch die manuelle Aktivierung des Konsolen-Prozesses gelöst.

"scheduler.a51"

Die Datei 'Scheduler' hat drei öffentliche Funktionen (scheduler, process_start, process_stop) und regelt die grundlegenden Funktionen der Task-Verwaltung. Die Funktionen "process_start" und "process_stop" werden vom Konsolenprozess benötigt um die Prozess A, B und Z zu starten und zu stoppen. Die Funktion "scheduler" enthält die eigentliche Logik der Task-Verwaltung und wird bei einem Interrupt durch den Timer 0 gestartet.

process_start

Hier wird hier die Zeitscheibendauer, die Stack-Adresse und die Start-Adresse des Programm-Codes des neuen Prozesses ermittelt. Anschließend wird der Prozess in der Prozess-Tabelle als 'aktiv' markiert und somit vom Scheduler berücksichtigt.

process_stop

Der zu stoppende Prozess wird aus der Prozesstabelle gelöscht. Durch setzen des Timer0-Interrupt Flags wird die Prozesswechselroutine aufgerufen und der Scheduling-Vorgang angestoßen.

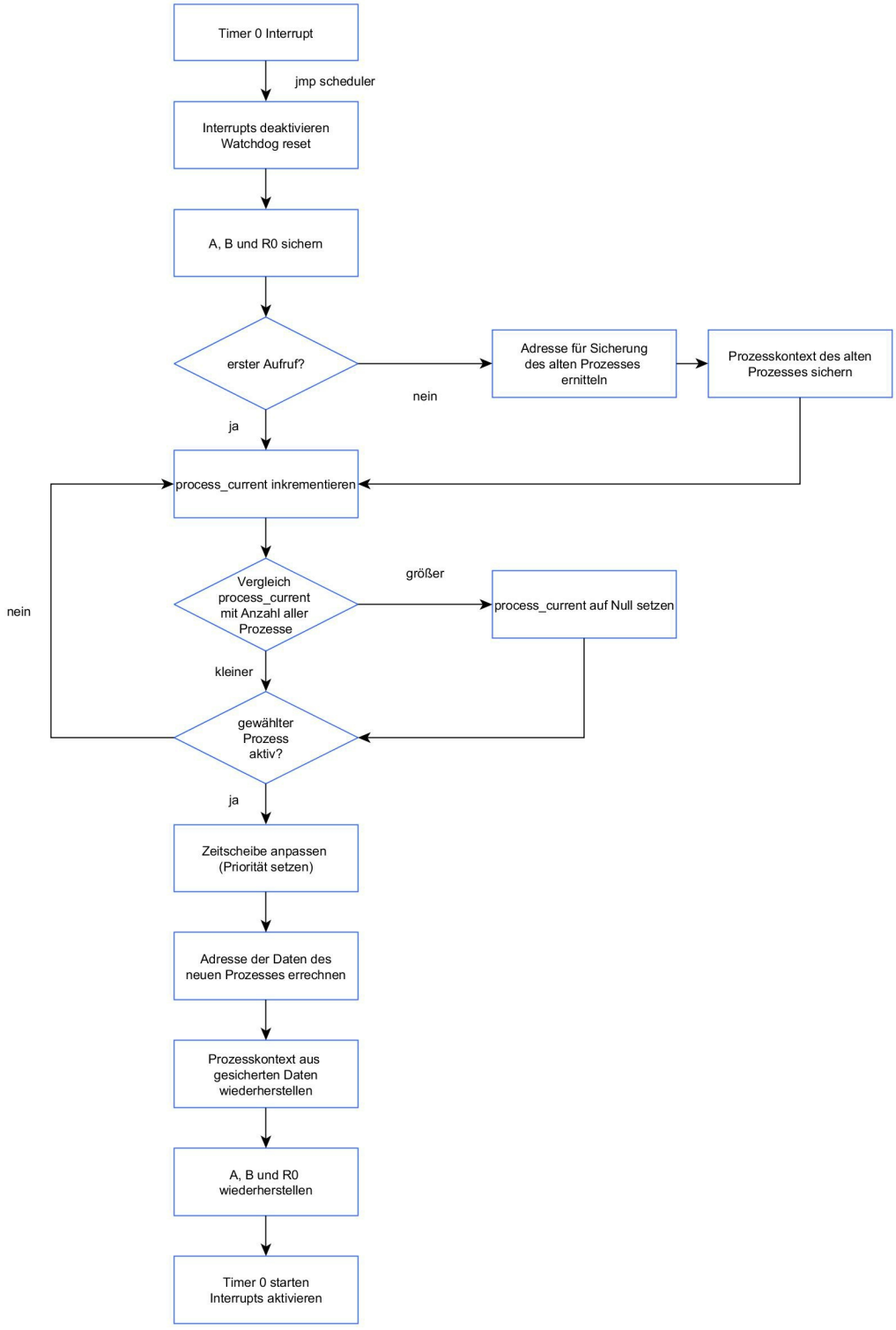
scheduler

Diese Routine wird durch einen Überlauf des Timers 0 aufgerufen und enthält die Logik zum wechseln zwischen zwei aktiven Prozessen.
Zuerst werden alle Interrupts deaktiviert, der Watchdog zurückgesetzt und die Register A, B und R0 gesichert. Dann wird überprüft, ob die Interuptroutine zum ersten Mal aufgerufen wird oder nicht. Falls nicht, muss die Adresse des vorherigen Prozesses, sowie sein kompletter Kontext, gesichert werden. Sobald der Prozesskontext komplett gesichert ist, sucht eine Routine nach dem nächsten aktiven Prozess in der Prozesstabelle. Dafür wird ausgehend vom aktuellen Prozess über die Tabelle iteriert. Der nächste aktive Prozess wird anschließend ausgewählt. Danach wird dem neuen Prozess die richtige Prozessorzeit zugewiesen und die Adresse seiner Daten berechnet. Sobald alle Adressen berechnet wurden, wird sein Prozesskontext geladen und die Register A, B und R0 wieder hergestellt. Zum Schluss werden noch die Interrupts aktiviert und der Timer 0 gestartet.

Bei der Sicherung des Stacks ist folgendes zu beachten: Abweichend von der Aufgabenstellung hat in diesem Projekt jeder Prozess einen Verfügbaren Stack-Speicher von 4 statt 10 Byte. Dies hat zum einen den Grund, dass keiner der Prozesse mehr Speicher benötigt, und zum Anderen, dass ansonsten eine Auslagerung in XDATA benötigt würde. Auf diese wurde aufgrund der Komplexität verzichtet. Angemerkt sei zudem, dass der Standard-Speicher noch eine geringfügige Erhöhung des Speicherbereichs zulassen würde.

Die Prozesse laufen nicht-kooperativ. Ist beispielsweise Prozess A fertig, so bleibt er während seiner Laufzeit in einer 'do nothing' Schleife.

Programmablaufplan (PAP) der "scheduler"-Funktion



"process_a.a51"

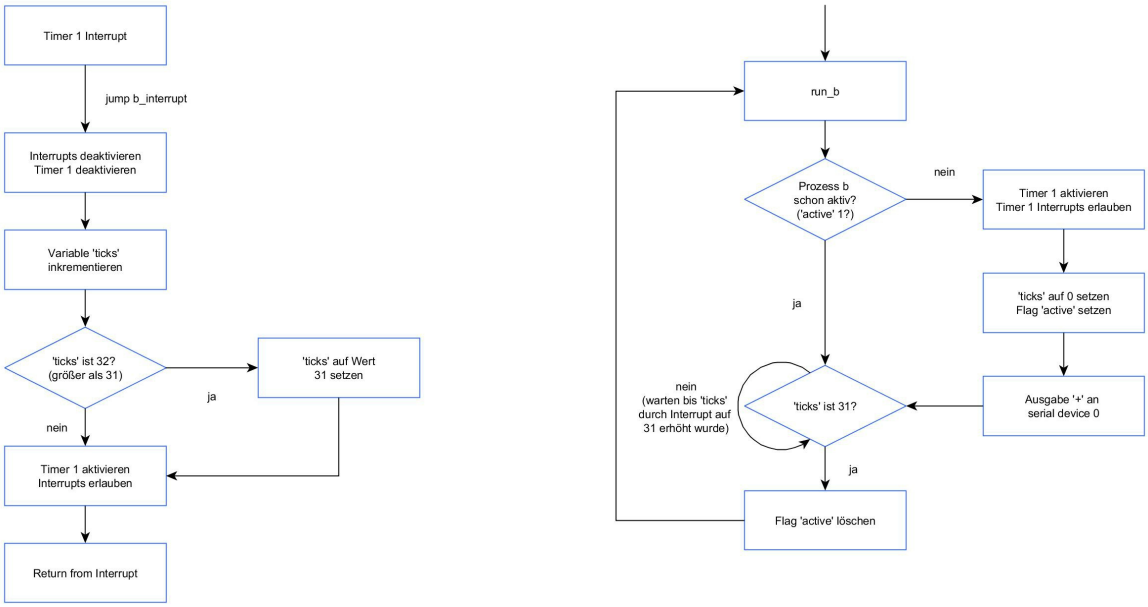
Prozess A soll die Zeichenfolge "abcde" auf der serielle Schnittstelle 0 ausgeben.
Dafür werden nacheinander die einzelnen Zeichen an die Schnittstelle gesendet. Ist das senden eines Zeichens abgeschlossen, so wird mit dem nächsten weitergemacht. Während des Sende-Vorgangs sind globale Interrupts deaktiviert, sodass keine Interferenzen zwischen den Prozessen auftreten können.

"process_b.a51"

Prozess B soll im Sekundentakt ein '+' Zeichen auf die serielle Schnittstelle 0 schreiben.
Es wird zuerst geprüft, ob der Prozess bereits aktiv ist. Falls nicht, so wird der Timer 1 gestartet und dessen Interrupt erlaubt. Anschließend wird die Zählvariable 'ticks' auf den Wert 0 gesetzt und das Flag, dass Prozess B aktiv ist, gesetzt. Die Zählvariable hat die Aufgabe, die Überläufe des Timer 1 zu zählen, denn der Timer läuft ca. 31-mal in einer Sekunde über. Anschließend wird auf die serielle Schnittstelle, das Zeichen '+' gesendet. Die Routine des Prozesses befindet sich dann in einer Endlosschleife, in der überprüft wird, ob bereits eine Sekunde vorbei ist, 'ticks' also wieder den Wert 31 erreicht hat. Falls ja, wird das Flag für den aktiven Prozess gelöscht und die Routine von vorne aufgerufen.
Im Interrupt des Timer 1, der für das Zählen der Sekunde zuständig ist, werden erst alle Interrupts und der Timer 1 deaktiviert. Anschließend wird die Zählvariable um 1 erhöht. Sollte der Wert größer als 31 sein, so wird der Wert mit 31 überschrieben, sodass er nie größer werden kann. Dies kann bei Überläufen geschehen, wenn der Prozess B aktuell nicht im Prozessor läuft. Am Ende der Routine werden dann alle Interrupts wieder erlaubt und der Timer1 wieder gestartet. Dieses Konzept verursacht, dass Prozess B bei niedriger Priorität eventuell nicht ganz im Sekundentakt arbeitet, sondern manchmal minimal länger brauchen kann, dafür ist es sehr zuverlässig.

Anbei ein Programmablaufplan der Interrupt-Routine und der Prozess-Routine:

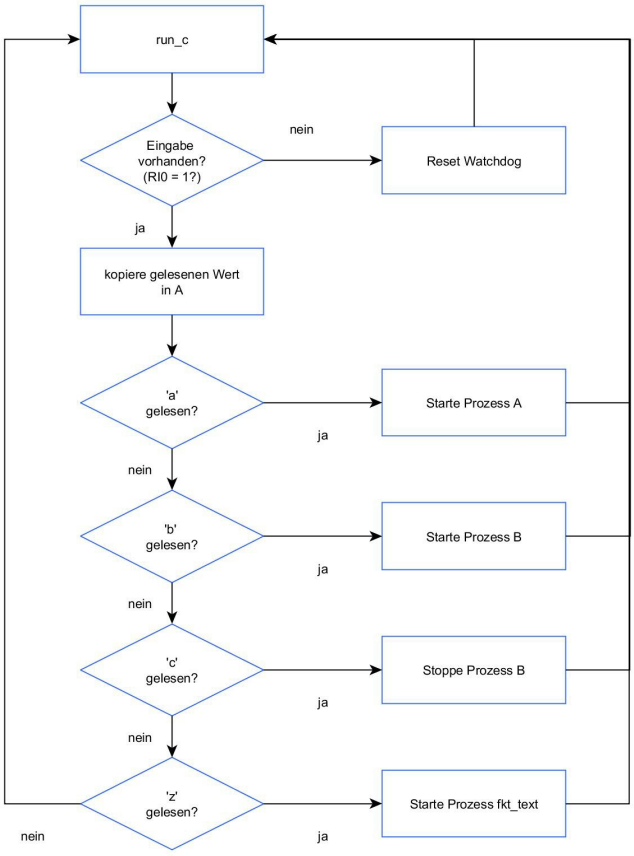
Programmablaufplan (PAP) Prozess B und Interrupt-Routine für Timer 1



"process_c.a51"

Der Prozess C (Konsolenprozess) läuft in einer Endlossschleife, in welcher permanent abgeprüft wird, ob ein Zeichen an die seriellen Schnittstelle gesendet wurde. Falls ja, so wird der geschriebene Wert in das Register A geladen und nacheinander auf die Werte 'a' (Prozess A starten), 'b' (Prozess B starten), 'c' (Prozess B stoppen) und 'z' (Prozess Z starten/stoppen) überprüft. Falls eins der Zeichen eingelesen wurde, so wird die entsprechende Routine zum Starten bzw. Stoppen des jeweiligen Prozesses ausgeführt. Sollte keines der Zeichen geschrieben worden sein, so beginnt der Prozess wieder am Anfang, mit dem Warten auf eine Eingabe.

Programmablaufplan (PAP) des Konsolen Prozesses



Fragen:

1. Wie behandeln sie den doppelten Aufruf eines Prozesses solange dieser aktiv ist ? (z. B. bei der Befehlsfolge: ’b’, ’b’)

- Prozess A kann theoretisch von einer erneuten Eingabe von 'a' unterbrochen werden, er wird dann einfach neu gestartet. Dafür müsste jedoch auf der Konsole mit einer unnatürlichen Geschwindigkeit zweimal hintereinander ein 'a' eingegeben werden.
- Wenn Prozess B mit der Benutzereingabe 'b' auf der seriellen Schnittstelle 0 gestartet wird, so wird ein Flag gesetzt. Wird B erneut aufgerufen, so wird zuerst überprüft, ob das Flag gesetzt ist. Prozess B wird nicht erneut ausgeführt, sofern das Flag gesetzt ist.

2. Wie reagiert ihr Programm auf die Benutzung der Seriellen Schnittstelle 0 durch mehrere Prozesse ? (z.B.: Können sich die Prozesse blockieren ? Können ”gesendete“ Zeichen ”verschwinden“ ? ...)

Es können keine gesendeten Zeichen verschwinden, da während der Ausgabe auf der seriellen Schnittstelle 0 alle Interrupts deaktiviert sind und somit auch kein Prozesswechsel stattfinden kann. Diese Implementierung wurde so gewählt, um Interferenzen durch Prozesswechsel während eines Schreibvorgangs zu vermeiden.

3. Wie haben sie die Anforderung ”Prioritäten“ gelöst?

Eine hohe Priorität bedeutet bei dieser Umsetzung eine hohe Zeitscheibendauer für einen Prozess. Ein Prozess mit niedriger Priorität dagegen erhält nur für kurze Zeit die CPU. Die Prioritäten werden beim Starten eines Prozesses vergeben. In unserer Implementierung erhält der Konsolen-Prozess die höchstmögliche Priorität, Prozess Z eine hohe und Prozess A und B eine niedrige Priorität.

Kontrollausgaben

Das Programm liefert folgende Ausgaben bei den einzelnen Dateien auf den seriellen Schnittstellen:

OS1.inc

Serielle Schnittstelle 0: "abcde+++abcde++abcde+++++abcde++"
Serielle Schnittstelle 1: ""
Aus Datei: "abcde+++abcde++abcde+++++abcde++"

OS2.inc

Serielle Schnittstelle 0: "abcdeabcdeabcde"
Serielle Schnittstelle 1: "DHBW@STUTTGART__D"
Aus Datei: "abcdeDHBW@STUTTGART__abcdeDabcde"

OS3.inc"

Serielle Schnittstelle 0: "++++++++++++++++++++abcde++++++abcde++++abcde"
Serielle Schnittstelle 1: "DHBW@STUDHBD"
Aus Datei: "DH+B++++W++++@++++S+++T++++U+abcde++++++abcdeD++++HBabcdeD"

Listing

main.a51

```
$NOMOD51
#include <REG517A.h>

NAME main

    EXTRN CODE(serial_init)                                ; from init.a51
    EXTRN CODE(timer_init)                                 ; from init.a51
    EXTRN CODE (scheduler, process_start, process_stop) ; from scheduler.a51
    EXTRN CODE (b_interrupt)                               ; from process_b.a51

    main_data SEGMENT DATA

        PUBLIC next_process
        PUBLIC next_process_priority
        RSEG main_data

        ; current process: 0:console, 1:prozessA, 2:processB, 3:processZ
        next_process: DS 1

        ; priority of next process: high value = low priority (low runtime)
        next_process_priority: DS 1

        ; default space for stack
        stack: DS 4

    main_data_bit SEGMENT BIT
        PUBLIC first_run
        RSEG main_data_bit
        ; flag, used in scheduler to check first run
        first_run: DBIT 1

    _code SEGMENT CODE
        RSEG _code


    CSEG                ; start
    ORG 0               ; at adress 0
    JMP start           ; jump to program start

    ORG 000Bh           ; timer 0 interrupt (timer 0 = scheduler)
    JMP scheduler

    ORG 001Bh           ; timer 1 interrupt (timer 1 = counter)
    JMP b_interrupt
```

```
    ; start routine
    start:

        SETB EAL        ; allow global interrupts
        SETB ET0        ; allow timer 0 interrupt

        CALL serial_init    ; initialize serial interface 0
        CALL timer_init     ; initialize timers

        MOV SP, #stack      ; set stack pointer to reserved space

        SETB first_run      ; set bit "first_run" active (1)

        ; set next_process to 0 (console)
        MOV next_process, #0

        ; set next_process priority to 0 (highest)
        MOV next_process_priority, #0x00

        ; call process_start (to start process from next_process)
        CALL process_start

        SETB TR0            ; start timer 0
        SETB TF0            ; jump to timer 0 interrupt routine

    ; nop loop as default behavior if some program parts crash
    loop:
        NOP
        JMP loop
END
```

init.a51

```
;disable predefined 8051 registers and include CPU definition file (for
example, 8052)
$NOMOD51
#include <REG517A.h>

NAME init

    PUBLIC timer_init
    PUBLIC serial_init

    init_code SEGMENT CODE
        RSEG init_code


    timer_init:
        MOV TMOD, #0x11         ;set Timer 0/1 to 16-Bit
        SETB ET0                ;set Timer 0 interrupt
        RET

    serial_init:
    ;initialize serial interface

    CLR     SM0         ;Mode 1
    SETB    SM1         ;

    SETB BD             ; intern baud generator enabled
    MOV PCON, #0x00     ; SMOD = 0
    MOV S0RELH, #0x03   ; baudrate = 14423
    MOV S0RELL, #0xE6   ;

    SETB REN0           ; activate recieve
    MOV S1CON, 0x20
    RET
END
```

scheduler.a51

```
$NOMOD51
#include <REG517A.h>

NAME scheduler

    PUBLIC scheduler
    PUBLIC process_start
    PUBLIC process_stop

    EXTRN CODE(run_a)                   ; from process_a
    EXTRN CODE(run_b)                   ; from process_b
    EXTRN CODE(run_c)                   ; from process_c
    EXTRN CODE(fkt_text)                ; from fkt_text

    EXTRN DATA (next_process)           ; from main
    EXTRN DATA (next_process_priority)  ; from main
    EXTRN BIT (first_run)               ; from main

    schedule_data SEGMENT DATA          ; scheduler data space

        RSEG schedule_data

        puffer_a: DS 1                  ; backup space for A
        puffer_b: DS 1                  ; backup space for B
        puffer_r0: DS 1                 ; backup space for register R0

        ; active processes are stored here
        process_table: DS 4

        ; process times to affect priorities
        process_time: DS 4

        ; current process
        process_current: DS 1

        ; 4 byte as stack for 4 processes
        process_stack: DS 16

        ; 4 * 14 byte to save register etc. for the processes
        process_state: DS 56

    _code SEGMENT CODE
        RSEG _code

    process_locations: DW run_c, run_a, run_b, fkt_text


    process_start:                      ; "starts" a process. scheduler will
handle this process now
```

```
    MOV A, next_process

    ADD A, #process_time            ; set process running time (priority)
    MOV R0, A
    MOV @R0, next_process_priority

    MOV A, next_process             ; get process stack adress
    MOV B, #4
    MUL AB
    ADD A, #process_stack
    MOV R1, A                       ; stack start adress of process in r1

    MOV A, next_process             ; calculate process start adress
    MOV B, #2
    MUL AB
    MOV R6, A
    MOV DPTR, #process_locations
    MOVC A, @A+DPTR
    MOV R5, A
    MOV A, R6
    INC A
    MOVC A, @A+DPTR

    MOV DPL, A
    MOV DPH, R5

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

    MOV A, next_process             ; load A with next_process

    MOV B, #14
    MUL AB
    ADD A, #process_state
    MOV R0, A                       ; R0 start adress for saving state

    MOV A, R1
    MOV @R0, A                      ; set stack to beginning


    MOV A, next_process
    ADD A, #process_table
    MOV R0, A
    MOV @R0, #0xFF
    RET

process_stop:                       ; stop a process from running

    MOV A, next_process
    ADD A, #process_table
```

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

    ; return to scheduler routine with timer 0 interrupt
    SETB TF0
    RET

scheduler:

    CLR EAL
    CLR TR0

    SETB WDT                        ; Watchdog reset
    SETB SWDT

    MOV puffer_a, A                 ; temp save A
    MOV puffer_b, B                 ; temp save B
    MOV puffer_r0, R0               ; temp save R0

    ; saving everything is not necessary on first run
    JBC first_run, find_process

    MOV A, process_current

    ; save state of current running process
    MOV B, #14
    MUL AB
    ADD A, #process_state
    MOV R0, A

    ; save state data of current process
    MOV @R0, SP
    INC R0
    MOV @R0, puffer_a
    INC R0
    MOV @R0, puffer_b
    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, puffer_r0
    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

; find next process (next active process of process table)
find_process:

    INC process_current
    MOV A, process_current
    CJNE A, #4, check_process
    MOV process_current, #0

check_process:

    MOV A, #process_table
    ADD A, process_current
    MOV R0, A
    CJNE @R0, #0xff, find_process

; calculate priority (process running time)
MOV A, #process_time
ADD A, process_current
MOV R0, A
MOV A, @R0
MOV TH0, A

; calculate adress of upcoming process
MOV A, process_current
MOV B, #14
MUL AB
ADD A, #process_state
MOV R0, A

; restore state of upcoming process
MOV A, @R0
MOV SP, A
INC R0
MOV puffer_a, @R0
INC R0
```

```
        MOV puffer_b, @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 puffer_r0, @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

        MOV A, puffer_a                         ; restore A
        MOV B, puffer_b                         ; restore B
        MOV R0, puffer_r0                       ; restore R0

        SETB TR0                                ; re-activate timer 0
        SETB EAL                                ; allow global interrupts
        RETI                                    ; return from interrupt

END
```

process_a.a51

```
$NOMOD51
#include <REG517A.h>

NAME process_a

    PUBLIC run_a

    _code SEGMENT CODE
        RSEG _code

    run_a:                  ; main routine of process a
        MOV B, #'a'
        CALL write
        MOV B, #'b'
        CALL write
        MOV B, #'c'
        CALL write
        MOV B, #'d'
        CALL write
        MOV B, #'e'
        CALL write
        JMP donothing

    write:
        CLR EAL             ; deactivate global interrupts
        MOV S0BUF, B        ; move B to serial device 0 output
    loop:
        SETB WDT            ; refresh watchdog
        SETB SWDT
        JNB TI0, loop
        CLR TI0
        SETB EAL            ; activate interrupts after sending to serial
        RET

    donothing:
        SETB WDT
        SETB SWDT
        JMP donothing
END
```

process_b.a51

```
$NOMOD51
#include <REG517A.h>

NAME process_b

    PUBLIC run_b
    PUBLIC b_interrupt

    _data SEGMENT DATA
        RSEG _data
        ; ticks : 31 ticks are ~ one second in 8051, 1 tick is 1 overflow of
timer 1
        ticks: DS 1

    _code SEGMENT CODE
        RSEG _code

    run_b:                          ; main routin of process b
        MOV ticks, #0
        SETB TR1
        SETB ET1

        MOV B, #'+'                 ; write 'b' to serial device 0
        CALL write

        MOV R0, #31                 ; R0  is check variable

        loopa:

            MOV A, ticks
            SUBB A, R0

            ; compare, wether ticks is 31, if no, stay in loop
            CJNE A, #0, loopa
            JMP run_b               ; jmp to run_b and start again

    write:
        CLR EAL                     ; deactivate global interrupts

        ; move content from B into S0BUF (write to serial device 0)
        MOV S0BUF, B
    loop:
        SETB WDT                    ; refresh watchdog
        SETB SWDT
        ; loop as long serial device 0 is sending
        JNB TI0, loop
        CLR TI0

        ; activate interrupts after sending to serial
```

```
        SETB EAL
        RET

    b_interrupt:                    ; interrupt routine of timer 1

        CLR EAL                     ; deactivate interrupts
        CLR TR1

        MOV A, #32
        INC ticks                   ; increment 'ticks'
        ; if 'ticks' is greater than 31, it should have 31
        CJNE A, ticks, end_interrupt
        MOV ticks, #31

        end_interrupt:              ; end interrupt

            SETB TR1
            SETB EAL
            RETI

END
```

process_c.a51

```
$NOMOD51
#include <REG517A.h>

NAME process_c

    PUBLIC run_c

    EXTRN DATA (next_process)               ; from main
    EXTRN DATA (next_process_priority)      ; from main

    EXTRN CODE (process_start)              ; from scheduler
    EXTRN CODE (process_stop)               ; from scheduler

    _code SEGMENT CODE
        RSEG _code

    run_c:
        ; Jump to input switch, if entry exists, else this is main loop of
console
        JBC RI0, input_switch

        SETB WDT                            ; Refresh Watchdog
        SETB SWDT

        JMP run_c

    input_switch:
        MOV A, S0BUF

    check_a:                                ; character a
        ; Jump to next label if input is not character a
        CJNE A, #'a', check_b
        MOV next_process, #1
        MOV next_process_priority, #0xAA
        CALL process_start
        JMP run_c

    check_b:                                ; character b
        ; jump to next label if input is not character b
        CJNE A, #'b', check_c
        MOV next_process, #2
        MOV next_process_priority, #0xCC
        CALL process_start
        JMP run_c

    check_c:                                ; character c
        ; jump no next label if input is not c
        CJNE A, #'c', check_z
        MOV next_process, #2
```

```
        CALL process_stop
        JMP run_c

    check_z:                                ; character z
        CJNE A,#'z', run_c

        ; jump back to main loop of console process if input is not z
        MOV next_process, #3
        MOV next_process_priority, #0x55
        CALL process_start
        JMP run_c

END
```

fkt_text.a51

```
$NOMOD51
#include <Reg517a.inc>

public  fkt_text

my_code segment CODE
    rseg    my_code

fkt_text: ; Ausgabe auf Seriel 1 (S1BUF)
;------------------------------
; Serial Interface 1
; 8 Datenbits, variable Baudrate
;   -> Mode B
;   => SM = 1
;   SETB    SM; geht nicht, s.o.
; Empfang freigeben -> REN1 = 1
;   SETB    REN1; geht nicht, s.o.
; noch nichts empfangen -> TI1 =0
;   CLR TI1; geht nicht, s.o.
;   aber S1CON = SM|.|.|REN1|.|.|TI1|.
    MOV A, S1CON
    ORL A, #10010000B
    ANL A, #1111$1101B
    MOV S1CON, A
; Baudrate
;   Baudrate = XTAL /(32*(2^10-S1REL))
;   9600 1/s -> S1REL = 3B2h
;       => S1RELH = 3
;       => S1RELL = B2H
    MOV S1RELH, # 3
    MOV S1RELL, #0B2h
; senden bereit
    ORL S1CON,#0000$0010B ;Set of TI1
;---------------------------------------
txt_init:
    mov r7, #0xff;
txt_schleife:
    mov r0, #0
    inc r7
    anl 7, #0x0f    ; eigendlich ANL R7, #0x0F; zum zählen bis 15
    ;
    mov a, r7
    call    Stelle5
    mov a, r0
    rlc a
    mov r0, a
    ;
    mov a, r7
    call    Stelle4
```

```
    mov a, r0
    rlc a
    mov r0, a
    ;
    mov a, r7
    call    Stelle3
    mov a, r0
    rlc a
    mov r0, a
    ;
    mov a, r7
    call    Stelle2
    mov a, r0
    rlc a
    mov r0, a
    ;
    mov a, r7
    call    Stelle1
    mov a, r0
    rlc a
    mov r0, a
    ; nach
    mov a, r0
    add a, #64
    mov r0, a
    nop
txt_warten:
;   jbc TI0, txt_ausgabe
;   setb    wdt
;   setb    swdt
    ; Warteschleife zur Zeitverzögerung
    mov r3, #16
txt_R3:
    mov r2, #0
txt_R2:
    mov r1, #0
txt_R1:
    nop
    djnz    r1, txt_R1
    nop
    djnz    r2, txt_R2
    nop
    djnz    r3, txt_R3

;   jmp txt_warten

txt_ser:
;test ob gesendet S1
    MOV A,S1CON
    JB ACC.1, txt_ausgabe_S1 ;TI1
    jmp txt_ser
;-----------
```

```
txt_ausgabe_S1:
    ANL S1CON,#1111$1101B ;Resetting of TI1
    MOV S1BUF, r0
    jmp txt_schleife
;
    ret ; sollte nie erreicht werden
; ------------------------------------------------------
Stelle5:
    mov c, acc.0
    anl c, acc.1
    anl c, /acc.3
        mov f1, c       ; sichern
    mov c, acc.3
    anl c, /acc.1
        orl c, f1
        mov f1, c
    mov c, acc.1
    anl c, acc.2
        orl c, f1
        mov f1, c
    mov c, acc.0
    anl c, acc.2
        orl c, f1
    ret

Stelle4:
    mov c, acc.0
    anl c, /acc.1
    anl c, /acc.2
    anl c, /acc.3
        mov f1, c
    mov c, acc.1
    anl c, acc.2
    anl c, acc.3
        orl c, f1
    ret
Stelle3:
    mov c, acc.0
    anl c, acc.1
    anl c, /acc.3
        mov f1, c
    mov c, acc.0
    cpl c
    anl c, /acc.1
    anl c, /acc.2
        orl c, f1
        mov f1, c
    mov c, acc.3
    anl c, /acc.2
    anl c, /acc.0
        orl c, f1
        mov f1, c
```

```
    mov c, acc.0
    anl c, /acc.1
    anl c, acc.3
        orl c, f1
        mov f1, c
    mov c, acc.2
    anl c, acc.1
        orl c, f1
    ret
Stelle2:
    mov c, acc.1
    anl c, /acc.2
    anl c, /acc.3
        mov f1, c
    mov c, acc.0
    anl c, /acc.1
    anl c, acc.2
    anl c, /acc.3
        orl c, f1
        mov f1, c
    mov c, acc.1
    anl c, /acc.0
    anl c, /acc.2
        orl c, f1
        mov f1, c
    mov c, acc.3
    anl c, acc.2
    anl c, /acc.0
        orl c, f1
        mov f1, c
    mov c, acc.1
    anl c, acc.2
    anl c, acc.3
        orl c, f1
    ret
Stelle1:
    mov c, acc.0
    anl c, acc.2
    anl c, /acc.3
        mov f1, c
    mov c, acc.1
    anl c, acc.3
        orl c, f1
        mov f1, c
    mov c, acc.0
    anl c, acc.1
        orl c, f1
    ret

end
```