Programmentwurf

Systemnahe Programmierung 1

Task-Verwaltung

Patrick Teutsch

Johannes Merz

Yannick Chairi

TINF11B

4. Halbjahr

2013

1 INHALT

2 EINLEITUNG

Ein Prozessor dient zur Ausführung von Programmcode. Befindet sich dieser Programmcode in der Ausführung spricht man von einem Prozess. Ein Prozessor kann, bedingt durch seinen Aufbau, nur einen Prozess gleichzeitig ausführen. Dies stellt kein Problem dar, solange mehr als ein Prozess gleichzeitig ausgeführt wird.

Nun gilt es sich zu überlegen wie die Ressource Prozessor sinnvoll auf die Prozesse aufgeteilt wird. Die Zuteilung des Prozessors an die Prozesse verwaltet ein "Scheduler"

Hier gibt es verschieden Zuteilungsmethoden, zwei davon werden kurz erläutert:

Kooperatives Scheduling (non-preemptive):

Der Prozessor führt einen Prozess bis zu dessen Ende aus. Danach wird dem nächsten Prozess der Prozessor zugeteilt. Der Scheduler verwaltet hier ausschließlich die Reihenfolge der nachfolgenden Prozesse. Ein großer Nachteil an dieser Methode ist, dass nicht darauf reagiert werden kann, falls der Prozess zum Beispiel in eine Endlosschleife läuft und somit den Prozessor für alle anderen Prozesse blockiert.

nicht-Kooperatives Scheduling (preemptive):

Hier werden dem Scheduler mehr Aufgaben zugeteilt als beim kooperativen Scheduling. Jeder Prozess bekommt vom Scheduler eine bestimmte Zeit den Prozessor zugeteilt bevor der Prozess unterbrochen wird und ein anderer Prozess die Ressourcen bekommt, man spricht von einer sogenannten Zeitscheibe. Wird ein Prozess mitten in seiner Ausführung unterbrochen liegt es am Scheduler dessen gerade genutzten "Daten" persistent zu sichern. Beim Prozesswechsel muss er dessen gesicherten "Daten" wiederherstellen damit dieser dort weitermachen kann wo er aufgehört hat.

Da es häufig Prozesse gibt die wichtiger sind, muss der Scheduler sicherstellen, dass den wichtigeren Prozessen mehr Prozessorzeit zugeteilt wird. Dies lässt sich durch eine Veränderung der Zeitscheibe realisieren. Wichtige Prozesse bekommen eine größere Zeitscheibe zugeteilt.

Im Folgenden werden drei Prozesse beschrieben die vom Scheduler verwaltet werden müssen. Danach werden der Aufbau und die Funktion des Schedulers selbst beschrieben. Im Anhang finden sich dann letztlich der Quellcode und der Ablauf der Prozesse und des Schedulers in Form von Flussdiagrammen.

3 PFLICHTANGABEN

Dieses Kapitel beantwortet die, in der Aufgabenstellung gestellten, Problemstellungen und Fragen.

3.1 MEHRFACHES STARTEN EINES PROZESSES

Wenn ein Prozess bereits gestartet wurde, jedoch der Konsolenprozess einen erneuten Start dieses Prozesses initiiert, wird der Prozess innerhalb der Prozesstabelle zurückgesetzt, da er überschrieben wird.

3.2 KONTROLLAUSGABE

Beim Einbinden der „OS.inc“ Datei und Starten des Programms entsteht folgende Ausgabe:

„54321aaa54321aa54321aaaaa54a321a“

4 BESCHREIBUNG DES PROGRAMMS

4.1 PROZESS A

Funktion von Prozess A ist es im 1-Sekunden Takt den Buchstaben „a“ auf PORT1 zu schreiben. Prozess A beendet sich nicht selbstständig sondern wird durch Benutzereingaben beendet.

Funktion „**processA**“:
Zunächst wird der Stackpointer von „processA“ auf den, für Prozess A definierten, Stackbereich gesetzt. Anschließend wird das Register 5 (R5) auf 246 gesetzt. Dieser Wert ergab sich durch experimentieren, läuft der Timer 246-mal nacheinander vergehen exakt 1,007625 Sekunden. Es erfolgt der Aufruf der „mainLoop“ diese ruft abwechselnd die Funktion „printAToUART“ und „waitRoutine“ auf.

Funktion „**printAToUART**“:
„a“ wird auf PORT1 geschrieben. Danach wird durch zyklisches Abfragen von S0CON auf Beendigung des Sendevorgangs geprüft. Ist dieser erfolgt wird das TI1-Empfangsbit zurückgesetzt und der Funktionsabschnitt endet.

Funktion „**waitRoutine**“:
Der Timer 0 wird gestartet. Nun wird solange gewartet bis der Timer 0 Overflowed. Nach zurücksetzen des Watchdogs und des Timer 0 Overflow Bits wird der Wert von R5 dekrementiert. Erreicht dieser 0 ist ca. eine Sekunde vergangen und es endet die „waitRoutine“.

4.2 PROZESS B

Aufgabe von Prozess B ist die Zeichenfolge „54321“ auf PORT1 zu schreiben und sich anschließend selbstständig zu beenden.

Funktion „**processB**“:
Zunächst wird der Stackpointer von „processB“ auf den, für Prozess B definierten, Stackbereich gesetzt. Es wird der Dezimal 53 (ASCII-Wert für 5) in Register 1 (R1) kopiert. In der folgenden Schleife wird nun der Wert von Register 1 auf PORT1 geschrieben und auf Beendigung des Sendevorgangs gewartet. Der Wert von Register 1 wird dekrementiert, das „Senden Bit“ wird zurückgesetzt und die Schleife wiederholt solange bis Register 1 den Wert 48 erreicht, was dem ASCII Wert von 0 entspricht. Dadurch lässt sich die Ausgabe „54321“ generieren (ASCII-Werte : 53 = „5“, 52 = „4“, 51 = „3“, 50 = „2“, 49 = „1“, 48 = „0“). Anschließend beendet sich der Prozess selbst, indem er dem Scheduler seine Startadresse übergibt und das „isNew“ Bit auf „isDel“ setzt.

4.3 PROZESS C (KONSOLEN PROZESS)

Funktion „***processC“:***
Zunächst wird der Stackpointer von „processC“ auf den, für Prozess C definierten, Stackbereich gesetzt. Anschließend wird eine Endlosschleife gestartet. In dieser wartet eine weitere Schleife auf Eingaben auf PORT1. Ist eine Eingabe erfolgt setzt sich das RI0-Empfangsflag und die Schleife endet. Die Eingabe wird anschließend in Register 7 (R7) kopiert. Danach wird die Funktion zur weiteren Eingabebehandlung („handleSerial0Input“) aufgerufen. Das RI0-Empfangsbit wird zurückgesetzt und die Endlosschleife von „processC“ beginnt von vorne.

Zusätzlich wird in der Endlosschleife zyklisch der Watchdogtimer zurückgesetzt. Andernfalls würde der Watchdog nach einer bestimmten Anzahl von Durchläufen annehmen, das Programm befände sich in einer Verklemmung und alle Register zurücksetzen.

Funktion „***handleSerial0Input***“:
Gemäß der Aufgabenstellung ist der weitere Programmverlauf abhängig von der Tastatureingabe

Taste	Aktion
A	Prozess A starten
B	Prozess A beenden
C	Prozess B starten
Sonst	Keine Aktion

Hierzu wird der bedingte Aufruf CJNE verwendet. CJNE springt an das angegebene Label wenn die Bedingung nicht erfüllt wird. War die Eingabe weder a, b oder c, so wird nur der Inhalt von Register 7 (R7) gelöscht und zurück in die Endlosschleife von „***processC***“ gesprungen. Erfolgt eine gültige Eingabe wird der jeweilige Datenzeiger kopiert. Durch das Bit „isNew“ und „isDel“ wird angegeben ob der Prozess gestartet oder beendet werden soll.

4.4 BOOTSTRAP PROZESS

Dieser Prozess wird als erstes nach Starten des Programms ausgeführt. Seine Aufgabe ist es alle nötigen Register (Timer, UART, Interrupts) sowie die Prozesstabelle zu initialisieren. Danach teilt er dem Scheduler mit, dass Prozess C gestartet werden soll und startet die Hauptschleife des gesamten Programms. In dieser Schleife befindet sich das Programm solange kein Prozess als gestartet ist.

4.4.1 Der Scheduler

Der Scheduler ist der Kern des Projekts und besteht aus verschiedenen Teilen, auf welche im Verlauf der Dokumentation weiter eingegangen wird. Die Aufgabe des Schedulers besteht darin, Prozesse zu verwalten und sie entsprechend ihrer Priorität auszuführen.

4.4.2 Datensegmente

Der Scheduler verfügt über mehrere Datensegmente, die intern verwendet werden, jedoch größtenteils von externen Prozessen gesetzt werden.

Die genutzten Datensegmente setzen sich wie folgt zusammen:

- **processTable**
  processTable stellt die vom Scheduler genutzte Prozesstabelle dar. Der genaue Aufbau kann Kapitel „Aufbau der Prozesstabelle“ entnommen werden.
- **processStartAdress**
  processStartAdress wird von externen Prozessen beim Anlegen bzw. Löschen eines Prozess dazu verwendet, dem Scheduler die Startadresse des zu startenden bzw. zu löschenden Prozesses mitzuteilen. Dieses Datensegment wird nur in Kombination mit dem newBit verwendet.
- **newBit**
  Das newBit-Datensegment wird von externen Prozessen gesetzt. Es zeigt dem Scheduler an, ob es einen zu löschenden bzw. anzulegenden Prozess gibt.
- **index**
  Das index- Datensegment zeigt auf die aktuelle Zeile innerhalb der Prozesstabelle und wird ausschließlich intern verwendet.

4.4.3 Konstanten

Die konstanten Datensegmente werden dazu verwendet, um auszuführende Aktionen zu kodieren und die Lesbarkeit zu erhöhen. Benutzt werden die folgenden Konstanten:

- **isNew**
  isNew wird in Kombination mit dem Datensegment newBit verwendet und zeigt an, ob der Scheduler einen neuen Prozess in der Prozesstabelle anlegen soll.
- **isDel**
  isDel wird in Kombination mit dem Datensegment newBit verwendet und zeigt an, ob der Scheduler einen neuen Prozess aus der Prozesstabelle entfernen soll.
- **isNon**
  isNon wird in Kombination mit dem Datensegment newBit verwendet und zeigt dem Scheduler an, dass er weder anlegen noch löschen soll.
- **isProcessA**
  isProcessA wird ausschließlich intern verwendet und entspricht der Adresse des Prozess innerhalb der Prozesstabelle.
- **isProcessB**
  isProcessB wird ausschließlich intern verwendet und entspricht der Adresse des Prozess innerhalb der Prozesstabelle.
- **isProcessC**
  isProcessC wird ausschließlich intern verwendet und entspricht der Adresse des Prozess innerhalb der Prozesstabelle.

4.4.4 Aufbau der Prozesstabelle

Im Projekt wird eine statische, 78 Byte große, Prozesstabelle verwendet, in der die Positionen der potentiellen Prozesse fest vorgegeben sind (vgl. Tabelle 1).

Offset	ist Aktiv	akt. SP	Prozessstartadresse	Prozessdaten
0	0 / 1	…	Prozess A	…
26	0 / 1	…	Prozess B	…
52	0 / 1	…	Prozess C	…
Größe:	1 Byte	1 Byte	2 Byte	22 Byte

Tabelle 1: Aufbau der Prozesstabelle

Das „ist Aktiv“-Flag wird dafür genutzt um dem Scheduler anzuzeigen, ob der jeweilige Prozess läuft oder nicht. Entsprechend wird es beim Anlegen eines Prozesses auf den Wert 1 gesetzt bzw. beim Stoppen auf den Wert 0.
Bei der Initialisierung der Tabelle wird die gesamte Spalte auf den Wert 0 gesetzt, da anfangs noch kein Prozess gestartet wurde.

Das Feld „aktueller Stackpointer“ wird verwendet um den Stackpointer eines Prozesses nach Ablauf von dessen Zeitscheibe zu sichern. Wenn der Prozess erneut an der Reihe ist, wird der Stackpointer wiederhergestellt, um sicherzustellen, dass der Prozess an der Stelle weiterläuft, an der er unterbrochen wurde.

Die Prozessstartadresse wird bereits bei der Initialisierung der Tabelle auf die entsprechende Adresse gesetzt und bleibt konstant. Die Adresse wird zur Identifikation eines Prozesses verwendet, wenn dieser gestartet oder gestoppt werden soll.

Die Prozessdaten stellen den Stack des jeweiligen Prozesses dar. Um zu gewährleisten, dass Prozesse ihre Daten in der Tabelle speichern, werden deren Stackpointer anfangs auf eine Adresse relativ zur Startadresse der Prozesstabelle gesetzt.

4.4.5 Die Interrupt-Routine

Die Interrupt-Routine, die nach Ablauf einer Zeitscheibe ausgeführt wird, enthält die Hauptfunktionalität des Schedulers. In ihr wird zunächst der Stack des unterbrochenen Prozesses in der Tabelle gespeichert. Anschließend wird solange die Prozesstabelle durchlaufen, bis ein Prozess gefunden wird, dessen „ist Aktiv“-Feld gesetzt ist. Ebenfalls werden Prozesse erstellt bzw. gelöscht, falls das newBit entsprechend gesetzt wurde. Nach erfolgreicher Identifikation eines aktiven Prozesses, wird zunächst seine Zeitscheibendauer (Priorität) festgelegt (vgl. Kapitel „Zeitscheiben (Prioritäten)“). Anschließend werden Stackpointer und Stack des Prozess wiederhergestellt und es wird zurück in den Prozess gesprungen, damit dieser seine Arbeit fortsetzen kann.

4.4.6 Zeitscheiben (Prioritäten)

Im Projekt werden die Zeitscheiben über einen 16-Bit-Timer realisiert, um diesen komfortabel manipulieren zu können. Da in ProzessA ein 13-Bit-Timer (8-Bit mit prescaling) verwendet wird, werden die Zeiten relativ zu diesem verteilt und wurden folgendermaßen berechnet:

Relative Priorität = $(2^{16} - 2^{13}) + \sum_{i=0}^{P} 2^{12-i}$, mit P = absolute Priorität eines Prozesses

Die hierzu verwendeten, absoluten Prioritäten lauten wie folgt:

Prozessname	Priorität
Prozess A	0
Prozess B	1
Prozess C	2

Ein geringerer Wert bedeutet eine höhere Priorität. Die Prioritäten wurden so gewählt, dass ProzessA möglichst häufig an der Reihe ist, um die Ausgabe der „a“ möglichst nahe bei einer Sekunde zu halten. Da ProzessB ein relativ kurzer Prozess ist, da keine Endlosschleife enthalten ist, besitzt er die nächst höhere Priorität. Aufgrund der Tatsache, dass Benutzereingaben verhältnismäßig selten sind, hat somit ProzessC die niedrigste Priorität erhalten.

5 LISTING

5.1 PROZESS A

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

NAME processA
EXTRN DATA (processTable)
PUBLIC processA

; create code segment for this process
processASegment SEGMENT CODE
RSEG processASegment

processA:
        ; set stackpointer relative to the
        ; processTable
        MOV A, #processTable
        ADD A, #4D
        MOV SP, A

        ; magic loop number
        MOV R5, #0xF6

        mainLoop:
                CALL printAToUART
                CALL waitRoutine
        JMP mainLoop

; write the character 'a' to UART0
```

```
printAToUART:
        MOV S0BUF, #'a'

        waitForSendFinished:
                MOV     A, S0CON
        JNB     ACC.1, waitForSendFinished

        ; reset TI0
        ANL A, #11111101b
        MOV S0CON, A
RET

; loops for about 1 second
waitRoutine:
        ; enable timer0
        MOV A, TCON
        ORL A, #00010000b
        MOV TCON, A

        ; wait for timer0 overflow
        timerPollingLoop:
                MOV A, TCON
        JNB ACC.5, timerPollingLoop

        CALL resetWD

        ; reset TCON
        MOV A, TCON
        ANL A, #11011111b
        MOV TCON, A

        ; return to timerPollingLoop if
        ; routine did not wait 1s
        DJNZ R5, timerPollingLoop

RET

resetWD:
        ; reset watchdog timer
        SETB WDT
        SETB SWDT
RET
END
```

5.2 PROZESS B

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

NAME processB

EXTRN DATA (processTable, processStartAdress, newBit, isDel)
PUBLIC processB

; define local code segment
processBSegment SEGMENT CODE
RSEG processBSegment

processB:

        ; set stack pointer relative to processTable
        MOV A, #processTable
        ADD A, #30D
        MOV SP,A

        CALL printToUART
        CALL cleanUp

; prints the characters '54321' to UART0
printToUART:
        ; initialize counter with ascii value
        ; of the character '5'
        MOV R1, #53d

        ; loop while counter > '1'
        countDownLoop:
                MOV S0BUF, R1

                ; loop until output of single character is finished
                waitForSendFinished:
                        MOV     A, S0CON
                JNB     ACC.1, waitForSendFinished

                DEC R1

                ; reset TI0 flag for further output
                ANL A, #11111101b
                MOV S0CON, A
```

```
        CJNE R1, #48d, countDownLoop
RET

cleanUp:
        ; tell the scheduler to delete processB
        ; from the processTable
        MOV DPTR, #processB
        MOV processStartAdress + 1, DPL
        MOV processStartAdress + 0, DPH
        MOV newBit, #isDel

        ; loop until processor time of processB is over
        doNothingLoop:
                NOP
                NOP
        JMP doNothingLoop
END
```

5.3 PROZESS C

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

NAME processC

EXTRN CODE (processA, processB)
EXTRN DATA (processStartAdress, newBit, processTable)
EXTRN NUMBER (isNew, isDel, isNon)
PUBLIC processC

; create code segment for this process
processCSegment SEGMENT CODE
RSEG processCSegment

processC:
        ; set stackpointer relative to
        ; processTable
        MOV A, #processTable
        ADD A, #56D
        MOV SP, A

        endlessLoop:
                ; reset watchdog timer
                SETB WDT
                SETB SWDT

                ; wait for input on UART0
                loopRec:
                        MOV A, S0CON
                JNB RI0, loopRec

                ; save received input in R7
                ; and call the input handler
                MOV R7, S0BUF
                CALL handleSerial0Input

                CLR RI0
        JMP endlessLoop
RET

; triggers creation or deletion of a process
; according to the received input
handleSerial0Input:
```

```
        ; check input on R7 and set parameters accordingly
        CJNE R7,#'a', afterA
                ; trigger creation of processA
                inputA:
                        MOV DPTR, #processA
                        MOV processStartAdress + 1, DPL
                        MOV processStartAdress + 0, DPH
                        MOV newBit, #isNew
                        JMP afterC
        afterA:
        CJNE R7,#'b', afterB
                ; trigger deletion of processA
                inputB:
                        MOV DPTR, #processA
                        MOV processStartAdress + 1, DPL
                        MOV processStartAdress + 0, DPH
                        MOV newBit, #isDel
                        JMP afterC
        afterB:
        CJNE R7,#'c', afterC
                ; trigger creation of processB
                inputC:
                        MOV DPTR, #processB
                        MOV processStartAdress + 1, DPL
                        MOV processStartAdress + 0, DPH
                        MOV newBit, #isNew
        afterC:

        ; reset R7
        MOV R7, #0x00
RET

END
```

5.4 BOOTSTRAP / SCHEDULER

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

EXTRN CODE (processA, processB, processC)
PUBLIC Delete, New, processStartAdress, newBit, processTable
PUBLIC isNew, isDel, isNon

;------------------------------------------------------------------------------
; Put the STACK segment in the main module.
;------------------------------------------------------------------------------
?STACK SEGMENT IDATA ; ?STACK goes into IDATA RAM.
RSEG ?STACK ; switch to ?STACK segment.
DS 25 ; reserve your stack space

; reserve data segments for the scheduler
mainData SEGMENT DATA
RSEG mainData
        ; processTable of the scheduler
        processTable: DS 78

        ; public data segmet which is used to tell the
        ; scheduler which process has to be started or
        ; stopped
        processStartAdress: DS 2

        ; points to the current row of the processTable
        index: DS 1

        ; public data segment which tells the scheduler
        ; if the process stored in processStartAdress
        ; has to be started or stopped.
        ; Should be set to isNew, isDel or isNon!
        newBit: DS 1

; define constants for semantical usage
isNew EQU 1
isDel EQU 2
isNon EQU 0

; define addresses of processTable rows
isProcessA EQU processTable
isProcessB EQU processTable + 26
isProcessC EQU processTable + 52
```

```
; Timer Interrupt
CSEG AT 0x1B
JMP             timer1Interrupt

CSEG AT 0
JMP             bootStrap

; create data segment for the scheduler
mainSegment SEGMENT CODE
RSEG mainSegment

; interrupt routine that loops through the
; processTable and determines the next process.
; it also creates new processes in the table or
; delete processes
timer1Interrupt:
        CLR TF1

        ; backup registers of current process
        JMP pushRegisters
        returnPushRegisters:

        ; save StackPointer in processTable
        ; and set SP to stack of scheduler
        MOV R0, index
        INC R0
        MOV @R0, SP
        MOV SP, #?STACK

        ; iterate through table until an active process
        ; is found
        processTableLoop:
                ; reset watchdog timer
                SETB WDT
                SETB SWDT

                ; Increment Index
                MOV A, index
                CJNE A, #processtable + 52, notOffset52;
                        ; reset index if it already points
                        ; to the last row of the processTable
                        MOV A, #processTable
                        JMP writeBack
```

```
        notOffset52:
                ; set pointer to the next row
                ADD A, #26d
        writeBack:
                MOV index, A

        ; update table if newBit is set to isNew
        MOV R0, newBit
        CJNE R0, #isNew, afterNew
                JMP new
        afterNew:
                ; update Table if newBit is set to isDel
                CJNE R0, #isDel, newOrDeleteFinished
                        JMP delete
        newOrDeleteFinished:

        ; reset newBit
        MOV newBit, #isNon

        ; check active flag
        MOV R1, index
CJNE @R1,#0x01, processTableLoop

; set timer according to priority
MOV TL1, #0x00
CJNE R1, #isProcessA, notProcessA
        CLR TR1
        MOV TH1, #0xE0
        SETB TR1
notProcessA:
CJNE R1, #isProcessB, notProcessB
        CLR TR1
        MOV TH1, #0xF0
        SETB TR1
notProcessB:
CJNE R1, #isProcessC, notProcessC
        CLR TR1
        MOV TH1, #0xF8
        SETB TR1
notProcessC:

JMP loadStackPointer
returnLoadStackPointer:
```

```
        JMP popRegisters
        returnPopRegisters:
RETI

bootStrap:
        ; set SP to a new stack for the scheduler
        MOV SP,#?STACK

        CALL init
        CALL callProcessC

        ; endless loop to make sure the scheduler
        ; never ends
        endlessSchedLoop:
                NOP
                NOP
                NOP
                NOP

                ; reset watchdog timer
                SETB WDT
                SETB SWDT
        JMP endlessSchedLoop

; set the flags so that console process
; is started by the schedulers interrupt routine
callProcessC:
        MOV DPTR, #processC
        MOV processStartAdress + 1, DPL
        MOV processStartAdress + 0, DPH
        MOV newBit, #isNew
RET

; enables interrupts and UARTs, sets timermodes and
; initializes the processTable
init:
        ; enable all interrupts and the specific
        ; serial0-interrupt
        SETB EAL
        SETB IEN0.3

        ; set UART mode to 8-bit
        CLR     SM0
        SETB SM1
```

```
; enable receive bit
SETB REN0

; enable baud rate generator
SETB BD

; set baud rate to 9600
MOV     S0RELL,#0xD9
MOV     S0RELH,#0x03

; set mode of timer1 to 16-bit
MOV A, TMOD
ANL A, #11001111b
ORL A, #00010000b
MOV TMOD, A

; start timer1
SETB TR1

; initialize newBit to 0
MOV newBit, #isNon

; initialize processTable "processStartAdress" columns and
; reset "Active" columns

; Process A
MOV DPTR, #processA
MOV processTable + 3, DPL ; ff 09
MOV processTable + 2, DPH
MOV processTable, #0x00

; Process B
MOV DPTR, #processB
MOV processTable + 29, DPL
MOV processTable + 28, DPH
MOV processTable + 26, #0x00

; Process C
MOV DPTR, #processC
MOV processTable + 55, DPL
MOV processTable + 54, DPH
MOV processTable + 52, #0x00
```

```
        ; init index with correct offset of the processstable
        MOV index, #processTable
RET

; called from the interrupt routine if isDel flag was set.
; sets the according process to inactive in the processTable.
delete:
        ; set DPTR to the Lable Adress given by the console process
        MOV DPH, processStartAdress + 0
        MOV DPL, processStartAdress + 1
        MOV R1, DPL
        MOV R2, DPH

        ; determine the process to delete in the processTable
        ; and set its active flag to 0
        MOV A, processTable + 2
        checkProcessA:
                CJNE A, 2, checkProcessB
                        MOV A, processTable + 3
                        CJNE A, 1, checkProcessB
                                MOV R0, #processTable + 0
                                MOV @R0, #0x00
                                JMP endDelete

        checkProcessB:
                MOV A, processTable + 28
                CJNE A, 2, checkProcessC
                        MOV A, processTable + 29
                        CJNE A, 1, checkProcessC
                                MOV R0, #processTable + 26
                                MOV @R0, #0x00
                                JMP endDelete

        checkProcessC:
                MOV A, processTable + 54
                CJNE A, 2, endDelete
                        MOV A, processTable + 55
                        CJNE A, 1, endDelete
                                MOV R0, #processTable + 52
                                MOV @R0, #0x00
                                JMP endDelete

        endDelete:
        NOP
```

```
JMP newOrDeleteFinished

new:
        ; set DPTR to the Lable Adress given by the console process
        MOV DPH, processStartAdress + 0
        MOV DPL, processStartAdress + 1
        MOV R1, DPL
        MOV R2, DPH

        MOV R3, #0x00

        ; determine the according row for the process to create in the
        ; processTable and store its startadress and some empty registers
        ; on the stack within the processTable.
        MOV A, processTable + 2
        newCheckProcessA:
                CJNE A, 2, newCheckProcessB
                        MOV A, processTable + 3
                        CJNE A, 1, newCheckProcessB

                                ; move stack pointer to the begin of the stack within
                                ; the processTable
                                MOV SP, #processTable + 4

                                ; push startadress of the process on the stack
                                PUSH 1
                                PUSH 2

                                ; push empty registers on the stack
                                PUSH 3
                                PUSH 3
                                PUSH 3
                                PUSH 3
                                PUSH 3
                                PUSH 3
                                PUSH 3
                                PUSH 3
                                PUSH 3
                                PUSH 3
                                PUSH 3
                                PUSH 3

                                ; store the changed stackpointer in the processTable
```

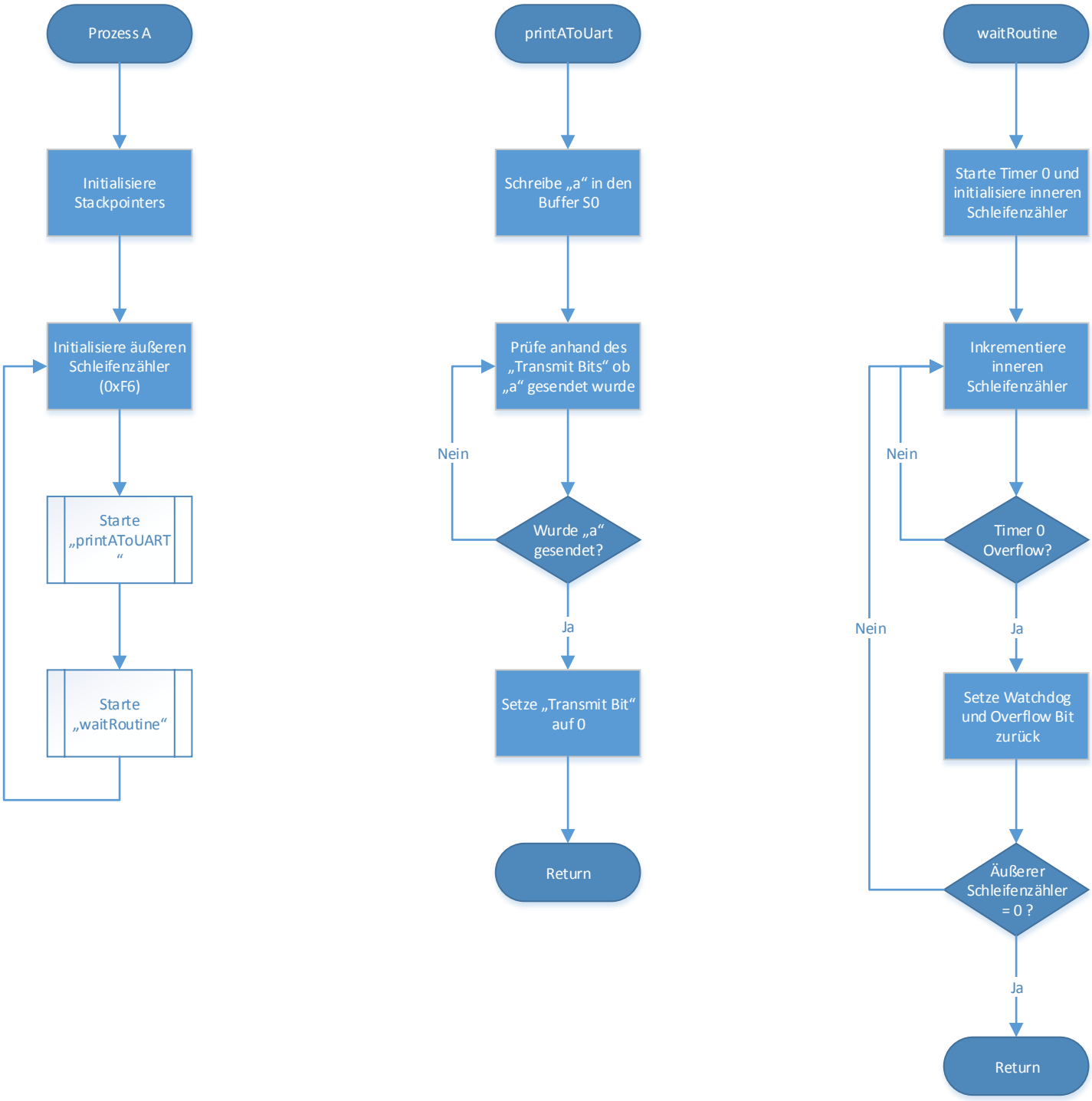
```
                    ; and set the active flag of the process to 1
                    MOV processTable + 1, SP
                    MOV processTable + 0, #0x01

                    JMP endNew

newCheckProcessB:
        MOV A, processTable + 28
        CJNE A, 2, newCheckProcessC
                MOV A, processTable + 29
                CJNE A, 1, newCheckProcessC

                    ; move stack pointer to the begin of the stack within
                    ; the processTable
                    MOV SP, #processTable + 30

                    ; push startadress of the process on the stack
                    PUSH 1
                    PUSH 2

                    ; push empty registers on the stack
                    PUSH 3
                    PUSH 3
                    PUSH 3
                    PUSH 3
                    PUSH 3
                    PUSH 3
                    PUSH 3
                    PUSH 3
                    PUSH 3
                    PUSH 3
                    PUSH 3
                    PUSH 3
                    PUSH 3
                    PUSH 3

                    ; store the changed stackpointer in the processTable
                    ; and set the active flag of the process to 1
                    MOV processTable + 27, SP
                    MOV processTable + 26, #0x01

                    JMP endNew
newCheckProcessC:
        MOV A, processTable + 54
        CJNE A, 2, endNew
```

```
                MOV A, processTable + 55
                CJNE A, 1, endNew

                        ; move stack pointer to the begin of the stack within
                        ; the processTable
                        MOV SP, #processTable + 56

                        ; push startadress of the process on the stack
                        PUSH 1
                        PUSH 2

                        ; push empty registers on the stack
                        PUSH 3
                        PUSH 3
                        PUSH 3
                        PUSH 3
                        PUSH 3
                        PUSH 3
                        PUSH 3
                        PUSH 3
                        PUSH 3
                        PUSH 3
                        PUSH 3
                        PUSH 3
                        PUSH 3

                        ; store the changed stackpointer in the processTable
                        ; and set the active flag of the process to 1
                        MOV processTable + 53, SP
                        MOV processTable + 52, #0x01

                        JMP endNew
        endNew:

JMP newOrDeleteFinished

; pushes all needed registers on the stack
pushRegisters:
        PUSH PSW
        PUSH 0
        PUSH 1
        PUSH 2
        PUSH 3
        PUSH 4
```

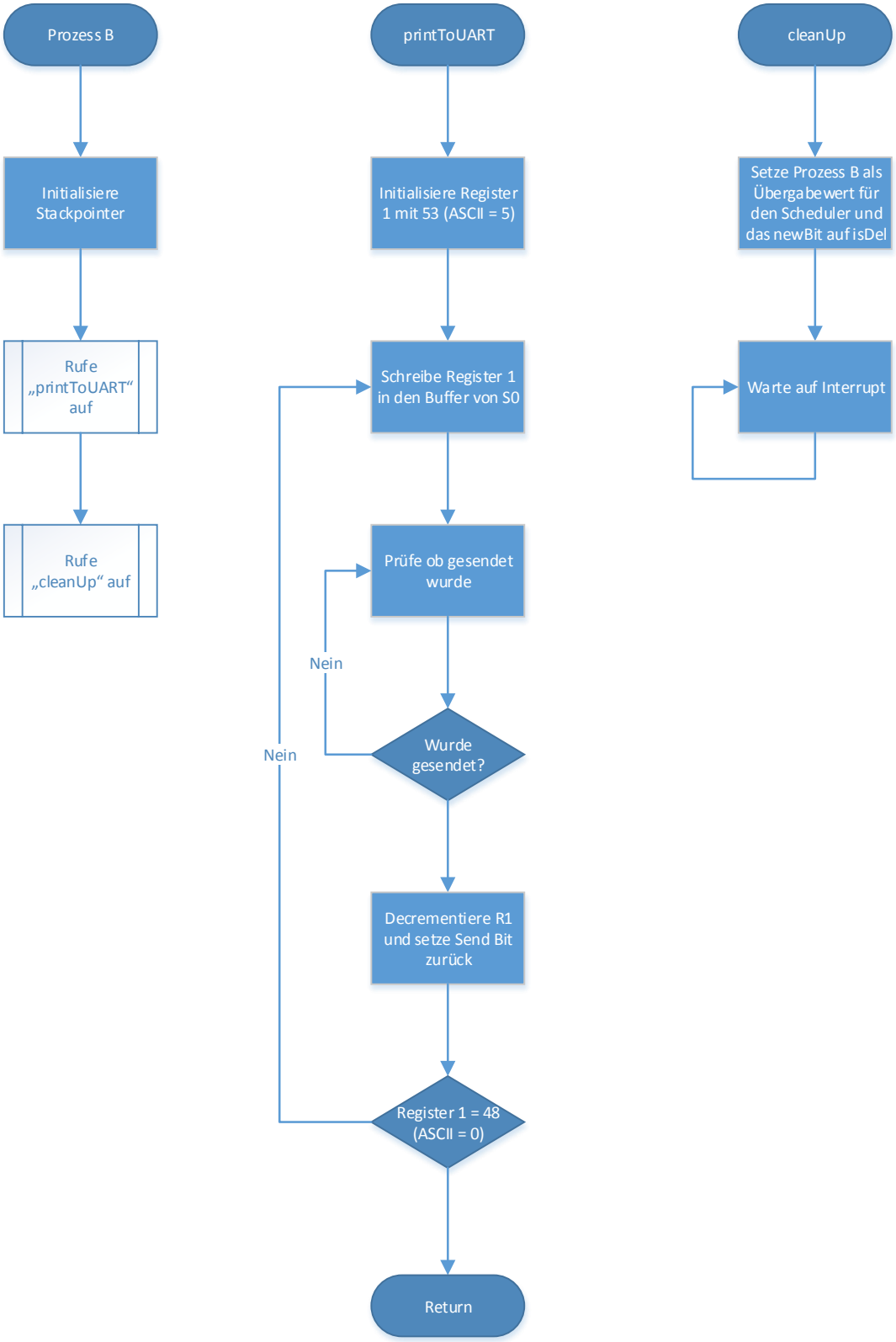
```
        PUSH 5
        PUSH 6
        PUSH 7
        PUSH ACC
        PUSH B
        PUSH DPH
        PUSH DPL
JMP returnPushRegisters

; pops all needed registers from the stack
popRegisters:
        POP DPL
        POP DPH
        POP B
        POP ACC
        POP 7
        POP 6
        POP 5
        POP 4
        POP 3
        POP 2
        POP 1
        POP 0
        POP PSW
JMP returnPopRegisters

; restores the SP of the next process to run
; from the processTable
loadStackPointer:
        MOV R0, index
        INC R0
        MOV SP, @R0
JMP returnLoadStackPointer

END
```

6 PROGRAMMFLUSS (DIAGRAMME)

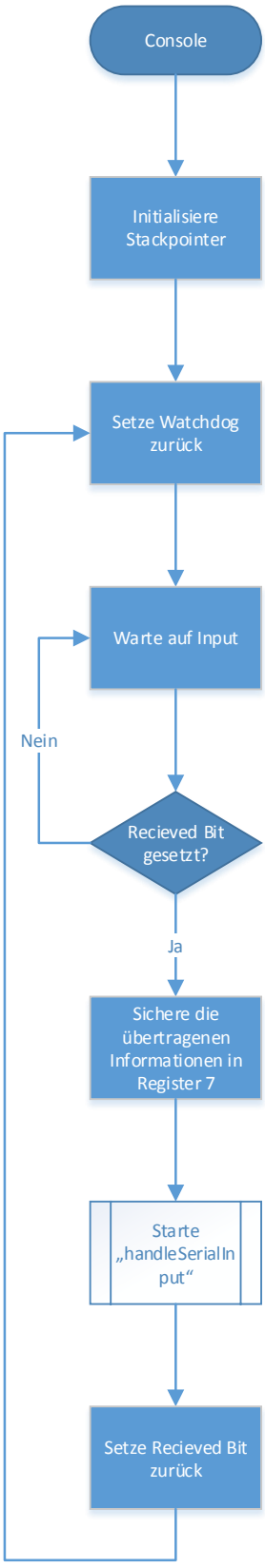
6.1 PROZESS A

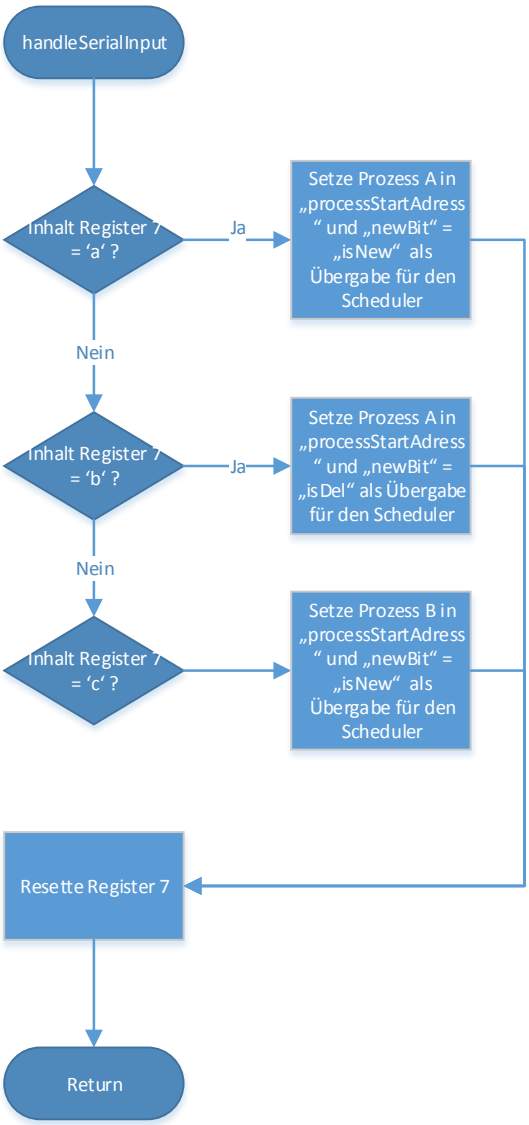


6.2 PROZESS B

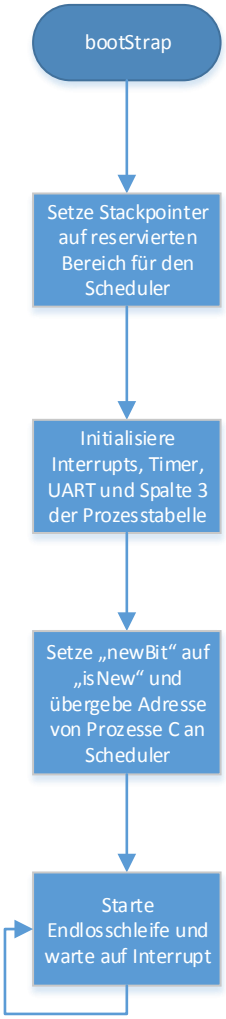

Prozess B
Initialisiere Stackpointer
Rufe „printToUART“ auf
Rufe „cleanUp“ auf
printToUART
Initialisiere Register 1 mit 53 (ASCII = 5)
Schreibe Register 1 in den Buffer von S0
Prüfe ob gesendet wurde
Nein
Wurde gesendet?
Nein
Decrementiere R1 und setze Send Bit zurück
Register 1 = 48 (ASCII = 0)
Return
cleanUp
Setze Prozess B als Übergabewert für den Scheduler und das newBit auf isDel
Warte auf Interrupt

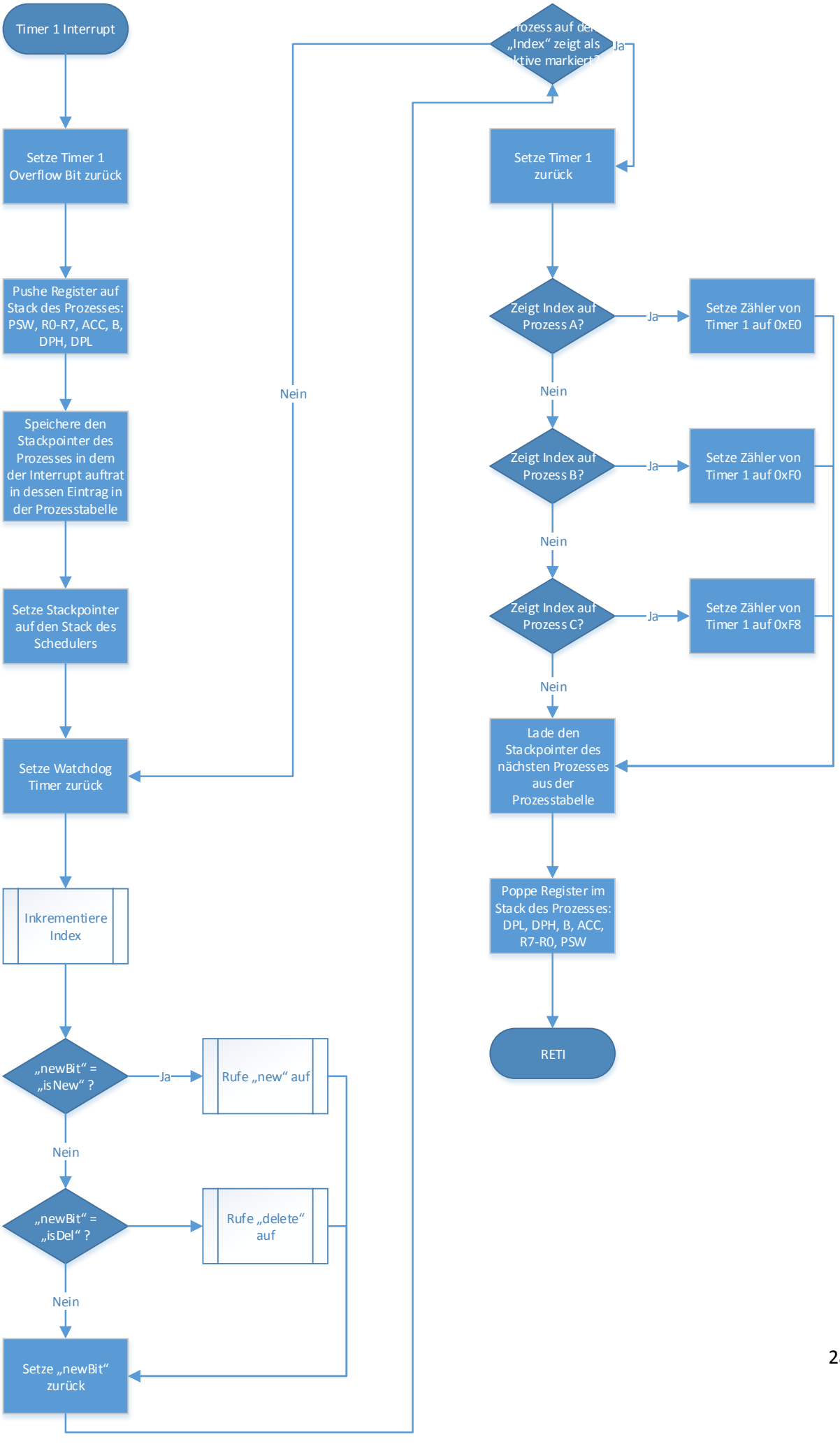
6.3 PROZESS C

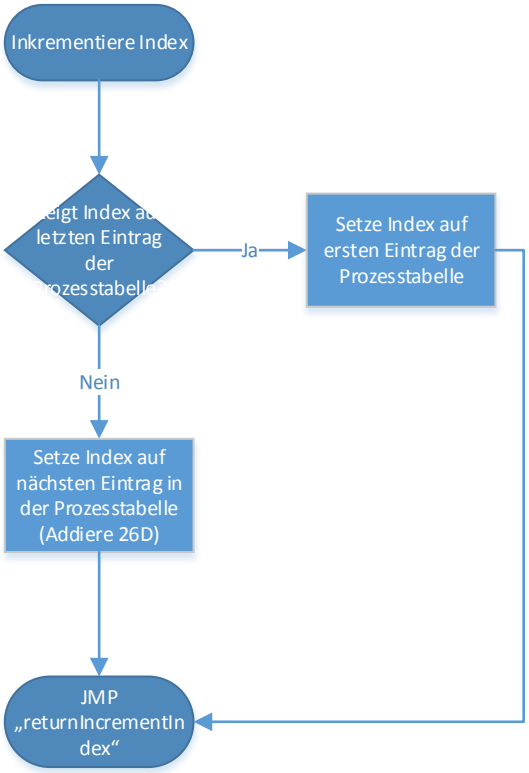

Console
Initialisiere Stackpointer
Setze Watchdog zurück
Warte auf Input
Nein
Recieved Bit gesetzt?
Ja
Sichere die übertragenen Informationen in Register 7
Starte „handleSerialInput“
Setze Recieved Bit zurück

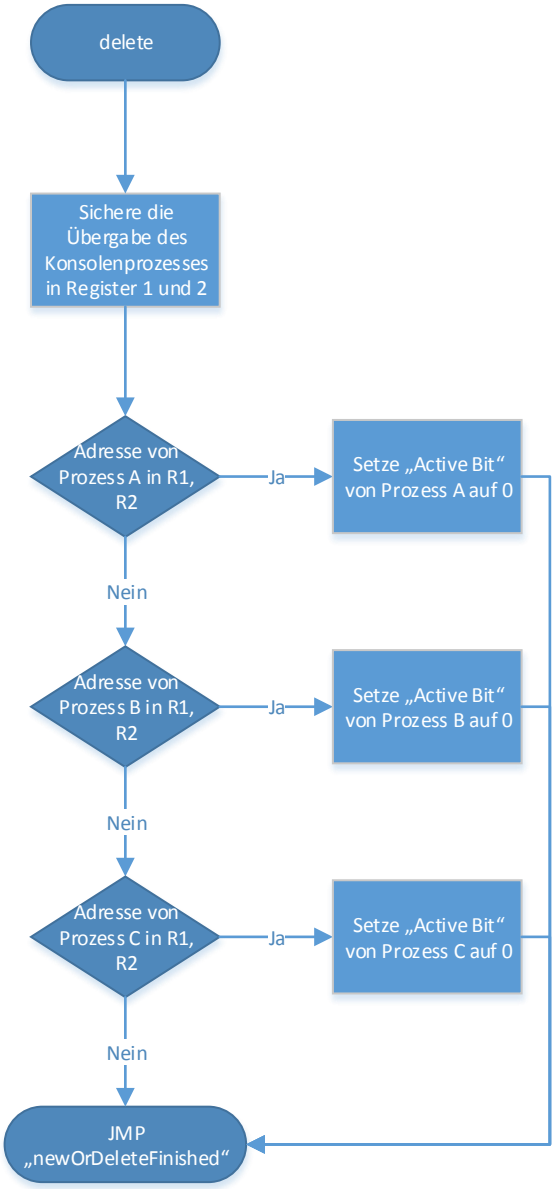

handleSerialInput
Inhalt Register 7 = ‘a‘ ?
Ja
Setze Prozess A in „processStartAdress“ und „newBit“ = „isNew“ als Übergabe für den Scheduler
Nein
Inhalt Register 7 = ‘b‘ ?
Ja
Setze Prozess A in „processStartAdress“ und „newBit“ = „isDel“ als Übergabe für den Scheduler
Nein
Inhalt Register 7 = ‘c‘ ?
Setze Prozess B in „processStartAdress“ und „newBit“ = „isNew“ als Übergabe für den Scheduler
Resette Register 7
Return

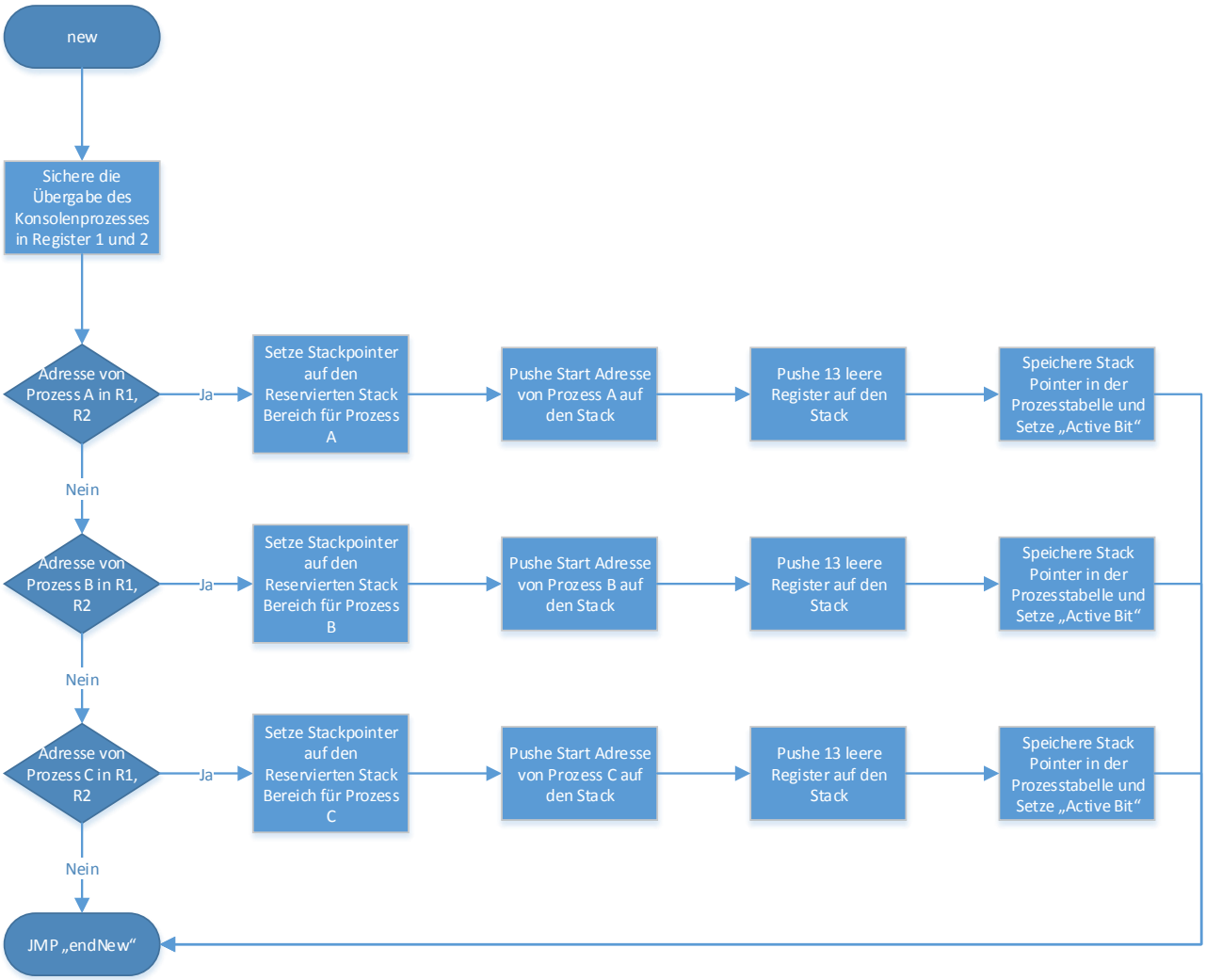
6.4 BOOTSTRAP / SCHEDULER




Timer 1 Interrupt
Setze Timer 1 Overflow Bit zurück
Pushe Register auf Stack des Prozesses: PSW, R0-R7, ACC, B, DPH, DPL
Speichere den Stackpointer des Prozesses in dem der Interrupt auftrat in dessen Eintrag in der Prozesstabelle
Setze Stackpointer auf den Stack des Schedulers
Setze Watchdog Timer zurück
Inkrementiere Index
„newBit“ = „isNew“ ?
Ja
Rufe „new“ auf
Nein
„newBit“ = „isDel“ ?
Rufe „delete“ auf
Nein
Setze „newBit“ zurück
Nein
Ja
Setze Timer 1 zurück
Zeigt Index auf Prozess A?
Ja
Setze Zähler von Timer 1 auf 0xE0
Nein
Zeigt Index auf Prozess B?
Ja
Setze Zähler von Timer 1 auf 0xF0
Nein
Zeigt Index auf Prozess C?
Ja
Setze Zähler von Timer 1 auf 0xF8
Nein
Lade den Stackpointer des nächsten Prozesses aus der Prozesstabelle
Poppe Register im Stack des Prozesses: DPL, DPH, B, ACC, R7-R0, PSW
RETI


Inkrementiere Index
igt Index a
letzten Eintrag
der
ozesstabell
Ja
Setze Index auf
ersten Eintrag der
Prozesstabelle
Nein
Setze Index auf
nächsten Eintrag in
der Prozesstabelle
(Addiere 26D)
JMP
„returnIncrementIn
dex“


delete
Sichere die
Übergabe des
Konsolenprozesses
in Register 1 und 2
Adresse von
Prozess A in R1,
R2
Ja
Setze „Active Bit“
von Prozess A auf 0
Nein
Adresse von
Prozess B in R1,
R2
Ja
Setze „Active Bit“
von Prozess B auf 0
Nein
Adresse von
Prozess C in R1,
R2
Ja
Setze „Active Bit“
von Prozess C auf 0
Nein
JMP
„newOrDeleteFinished“


new
Sichere die Übergabe des Konsolenprozesses in Register 1 und 2
Adresse von Prozess A in R1, R2
Ja
Setze Stackpointer auf den Reservierten Stack Bereich für Prozess A
Pushe Start Adresse von Prozess A auf den Stack
Pushe 13 leere Register auf den Stack
Speichere Stack Pointer in der Prozesstabelle und Setze „Active Bit“
Nein
Adresse von Prozess B in R1, R2
Ja
Setze Stackpointer auf den Reservierten Stack Bereich für Prozess B
Pushe Start Adresse von Prozess B auf den Stack
Pushe 13 leere Register auf den Stack
Speichere Stack Pointer in der Prozesstabelle und Setze „Active Bit“
Nein
Adresse von Prozess C in R1, R2
Ja
Setze Stackpointer auf den Reservierten Stack Bereich für Prozess C
Pushe Start Adresse von Prozess C auf den Stack
Pushe 13 leere Register auf den Stack
Speichere Stack Pointer in der Prozesstabelle und Setze „Active Bit“
Nein
JMP „endNew“