HOCHSCHULE ESSLINGEN FAKULTÄT INFORMATIONSTECHNIK

MUSTERPRÜFUNG B		Blatt Nr.:	**1 von 13**
Studiengang:	**Kommunikationstechnik Softwaretechnik Technische Informatik**	Semester:	SWB4, TIB4, KTB4
Prüfungsfach:	**Computerarchitektur 3**	Fachnummer:	**4021**
Hilfsmittel:	**Vorlesungs- und Labormanuskript, Fachliteratur, Taschenrechner**	Dauer:	**90 min**

Tragen Sie hier bitte Ihren Namen ein:

Vorname: Nachname:

Lösungsvorschlag (ohne Gewähr)

Aufgabe 1: Verständnisfragen (25 Punkte)

1.1 Erklären Sie in Stichworten die Begriffe Linken (linking) und Lokieren (locating).

Lösung zu Aufgabe 1.1:

Linken:
Verbinden von Objektdateien und Bibliotheken, Auflösen referenzierter Symbole wie Konstanten, Variablen und Funktionen.

Lokieren:
Jedem Maschinesprachebefehl wird eine absolute Adresse im Speicher zugewiesen.

1.2 Daten können im Speicher in „Little Endian“ oder „Big Endian“-Form abgelegt sein. Tragen Sie unten den hexadezimalen Wert `$12345678` im „Little Endian“-Format für einen 32-Bit-Rechner ab der Adresse `$1000` ein.

Lösung zu Aufgabe 1.2:

Adresse	Wert
0x1000	$78 (LSB)
0x1001	$56
0x1002	$34
0x1003	$12 (MSB)

Bitte geben Sie alle Aufgabenblätter wieder ab!

MUSTERPRÜFUNG B		Blatt Nr.:	**2 von 13**
Studiengang:	**Kommunikationstechnik Softwaretechnik Technische Informatik**	Semester:	SWB4, TIB4, KTB4
Prüfungsfach:	**Computerarchitektur 3**	Fachnummer:	**4021**
Hilfsmittel:	**Vorlesungs- und Labormanuskript, Fachliteratur, Taschenrechner**	Dauer:	**90 min**

1.3 Der Mikrocontroller vom Typ HCS12 unterstützt verschiedene Adressierungsarten. Im Folgenden sind Beispiele gegeben. Schreiben Sie die genaue Bezeichnung der Adressierungsart jeweils hinter das Beispiel. Einen der Befehle gibt es so nicht. Markieren Sie diesen mit einem Kreuz.

Lösung zu Aufgabe 1.3:

```
          alle Angaben beziehen sich auf den expliziten Operanden
STAA 9,Y       Indexed IDX: Register-indirekt mit Index (Offset)

EXG D,X        Explizite Registeradressierung

BRA loop       Relative Adressierung

ADDD 9,X+   X  Autoinc.IDX: Register-indirekt mit Post-Inkrement
            gibt es so nicht, nur Inkrement 1 … 8 möglich
ROL [D,Y]      Indexed-Indirect[IDX2]:Speicher-indirekt mit Index
```

1.4 Markieren Sie im folgenden Assemblerlisting (einkreisen und benennen) jeweils ein Beispiel für **Maschinencode**, **Assemblerbefehl, Operand, Current Location Counter, Sprungmarke (Label)** und **Pseudoassemblerbefehl (Assembler Direktive)**.

Listing und Lösung zu Aufgabe 1.4:

```
28   000000              platzhalter DS.B 1       Direktive
29   Cur.loc.counter
30
31                       STACK_RAM: SECTION
32   000000              stack       DS.B 100
33
34
35                       MyCode:     SECTION
36                       main:
38          Maschinencode
39   000000 10EF                 CLI ← Assemblerbefehl
40   000002 CFxx xx              LDS  #stack+$100
41   000005 86FF                 LDAA #$ff ← Operand (nur #$ff)
42   000007 5A03                 STAA DDRB
43   000009 180B FF02            MOVB #255, DDRP
             00000D 5A
44   00000E 7A02 58              STAA PTP
45
46                       loop:  ← Label
47   000011 180B 01xx            MOVB #1,platzhalter
```

Bitte geben Sie alle Aufgabenblätter wieder ab!

MUSTERPRÜFUNG B		Blatt Nr.:	**3 von 13**
Studiengang:	**Kommunikationstechnik Softwaretechnik Technische Informatik**	Semester:	SWB4, TIB4, KTB4
Prüfungsfach:	**Computerarchitektur 3**	Fachnummer:	**4021**
Hilfsmittel:	**Vorlesungs- und Labormanuskript, Fachliteratur, Taschenrechner**	Dauer:	**90 min**

1.5 Was bedeuten die „**xx**“ im Assemblerlisting aus Aufgabe 1.4, z.B. **CFxx xx**?

Lösung zu Aufgabe 1.5:

XXX sind relokierbare Adressen, diese werden erst beim Linken bzw. Lokieren eingesetzt.

1.6 Nehmen Sie an, der HCS12-Rechner im Labor ist gerade in eine Unterbrechungsroutine (ISR1) gesprungen und steht direkt vor dem Laden des ersten Befehls dieser ISR1 aus dem Programmspeicher. Genau in diesem Moment tritt eine zweite, höher priorisierte Unterbrechung mit zugehöriger Unterbrechungsroutine ISR2 auf. Wird der erste Befehl von ISR1 oder von ISR2 zuerst ausgeführt? Bitte genau begründen.

Lösung zu Aufgabe 1.6:

Befehl von ISR1 wird ausgeführt, da weitere Interrupts direkt nach Eintritt in eine ISR automatisch durch Setzen des I-Bits gesperrt werden.

MUSTERPRÜFUNG B		Blatt Nr.:	**4 von 13**
Studiengang:	**Kommunikationstechnik Softwaretechnik Technische Informatik**	Semester:	SWB4, TIB4, KTB4
Prüfungsfach:	**Computerarchitektur 3**	Fachnummer:	**4021**
Hilfsmittel:	**Vorlesungs- und Labormanuskript, Fachliteratur, Taschenrechner**	Dauer:	**90 min**

Aufgabe 2: Programmanalyse (25 Punkte)

Das folgende Programm stellt eine Funktion in HCS12-Assembler dar, die von einem C-Programm aufgerufen werden kann. Die C-Prototyp-Definition sieht so aus:

```
int func(int arg1, int arg2);
```

Listing zu Aufgabe 2:

```
                                ergebnis = abs(arg1*arg2)

 1  func: LEAS  -2,SP    ; int temp
 2        STD   2,-SP    ; int var2 = arg2
 3        LDD   6,SP     ; Check sign of arg1 (at 6,SP)
 4        CPD   #0
 5        BGE   B1       ; … if arg1 >= 0 goto B1
 6        LDD   0,SP     ; Check sign of arg2 (copy at 0,SP)
 7        CPD   #0
 8        BGT   B2       ; … if arg2 > 0 goto B2
 9  B1:   LDD   6,SP     ; Check sign of arg1 (at 6,SP)
10        CPD   #0
11        BLE   B3       ; … if arg1 <= 0 goto B3
12        LDD   0,SP     ; Check sign of arg2 (copy at 0,SP)
13        CPD   #0
14        BGE   B3       ; … if arg2 >= 0 goto B3
15  B2:   LDD   6,SP     ;← arg1 and arg2 have opposite signs
16        NEGA           ; calculate -arg1
17        NEGB           ; note: NEG… sets carry flag
18        SBCA  #0
19        LDY   0,SP     ; temp = -arg1 * arg2
20        EMUL
21        STD   2,SP
22        BRA   B4
23  B3:   LDD   6,SP     ;← arg1 and arg2 have the same sign
24        LDY   0,SP     ; temp = arg1 * arg2
25        EMUL
26        STD   2,SP
27  B4:   LDD   2,SP     ; return temp = abs(arg1 * arg2)
28        LEAS  4,SP     ; remove local variables
29        RTS
```

2.1 Die Funktion wird folgendermaßen aufgerufen: `e = func(-4,2)`. Welcher Wert steht nach Ausführung von Zeile 3 im D-Register?

Lösung zu Aufgabe 2.1:

`D = -4`

Bitte geben Sie alle Aufgabenblätter wieder ab!

MUSTERPRÜFUNG B		Blatt Nr.:	**5 von 13**
Studiengang:	**Kommunikationstechnik Softwaretechnik Technische Informatik**	Semester:	SWB4, TIB4, KTB4
Prüfungsfach:	**Computerarchitektur 3**	Fachnummer:	**4021**
Hilfsmittel:	**Vorlesungs- und Labormanuskript, Fachliteratur, Taschenrechner**	Dauer:	**90 min**

2.2 Die Funktion wird folgendermaßen aufgerufen: `e = func(-4,2).` Welcher Wert steht nach Ausführung von Zeile 7 im D-Register?

Lösung zu Aufgabe 2.2:

`D = 2`

2.3 Die Funktion wird folgendermaßen aufgerufen: `e = func(-4,2).` Welcher Wert steht nach Ausführung von Zeile 19 im D-Register?

Lösung zu Aufgabe 2.3:

`D = 4`

2.4 Wie wird das Ergebnis an die aufrufende Funktion zurückgegeben und welchen Wert hat das Ergebnis, wenn die Funktion folgendermaßen aufgerufen wird: `e = func(-4,2).`

Lösung zu Aufgabe 2.4:

`Ergebnis im D-Register (Zeile 27), Wert ist 8.`

2.5 Erstellen Sie ein zum Assemblerlisting äquivalentes C-Programm (Hinweis: Das gibt die meisten Punkte!) und geben Sie das Ergebnis an, wenn die Funktion so aufgerufen wird:

`e = func(2,-4);`

Lösung zu Aufgabe 2.5:

```
int func(int arg1, int arg2)
{
  int erg = arg2;

  if (arg1 < 0 && arg2 > 0 || arg1 > 0 && arg2 < 0)
  {
    erg *= -arg1;
  }
  else
  {
    erg *= arg1;
  }
  return erg;
}
```

`Ergebnis: 8`

MUSTERPRÜFUNG B		Blatt Nr.:	**6 von 13**
Studiengang:	**Kommunikationstechnik Softwaretechnik Technische Informatik**	Semester:	SWB4, TIB4, KTB4
Prüfungsfach:	**Computerarchitektur 3**	Fachnummer:	**4021**
Hilfsmittel:	**Vorlesungs- und Labormanuskript, Fachliteratur, Taschenrechner**	Dauer:	**90 min**

Aufgabe 3: Adressierungsarten und Stack (25 Punkte):

3.1
In einem HCS12-Assemblerprogramm sind folgende globalen Variablen definiert:

```
.const:    SECTION
           ORG   $D000
tabelle1:  DC.B  $19, $28, $37, $46, $55, $64, $73, $82
tabelle2:  DC.W  $D002, $D004
```

Geben Sie den Inhalt der CPU-Register D, X und Y nach jedem Assemblerbefehl an, wenn das folgende Programm ausgeführt wird. Es reicht aus, wenn Sie bei jedem Befehl diejenigen Registerwerte eintragen, die sich jeweils ändern.

Assemblerbefehle	**D**	**X**	**Y**
	$0000	$0000	$0000
LDX tabelle1		$1928	
LDY #tabelle1			$D000
LDX 3, Y		$4655	
LDD 2, Y+	$1928		$D002
LDX -1, Y		$2837	
LEAY 2, +Y			$D004
LDX #4		$0004	
LDD tabelle1, X	$5564		
LDX #tabelle2		$D008	
LDD [0, X]	$3746		

Bitte geben Sie alle Aufgabenblätter wieder ab!

MUSTERPRÜFUNG B		Blatt Nr.:	**7 von 13**
Studiengang:	**Kommunikationstechnik Softwaretechnik Technische Informatik**	Semester:	SWB4, TIB4, KTB4
Prüfungsfach:	**Computerarchitektur 3**	Fachnummer:	**4021**
Hilfsmittel:	**Vorlesungs- und Labormanuskript, Fachliteratur, Taschenrechner**	Dauer:	**90 min**

3.2

In einem C-Programm seien die folgenden globalen Variablen definiert:

```
char valA, valB;
char m;
```

Diese Variablen werden im folgenden C-Programm verwendet, das Sie „von Hand“ in die entsprechenden HCS12-Assemblerbefehle übersetzen sollen. Die Definition der globalen Variablen muss nicht übersetzt werden. Assemblerdirektiven wie XDEF, XREF, INCLUDE, SECTION usw. dürfen weggelassen werden.

a) Geben Sie den Assembler-Programmcode an:

Lösung zu Frage 3.2a:

C-Programm

```
//***** Hauptprogramm *****
void main(void)
{    . . .
     m = max(valA, valB);
     . . .
}



//***** Unterprogramm *****
char max(char a, char b)
{    char tempMax = b;


     if (tempMax < a)  //Zeile (*)
          tempMax = a;



     return tempMax;
}
```

HCS12-Assembler-Programm

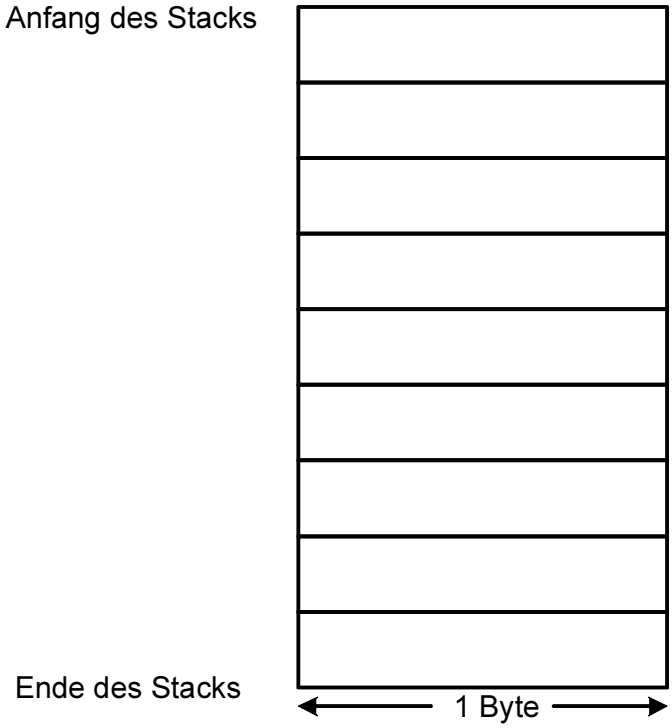
```
      LDAB  valA  ; Parameter-
      PSHB        ; übergabe
      LDAB  valB
      JSR   max
      STAB  m     ; Rückgabewert
      LEAS  1,+SP ;Stack abräumen



Max:  LEAS  1,-SP  } STAB 1,-SP
      STAB  0, SP  }


    ( LDAB  0, SP  ) überflüssig
      CMP   3, SP
      BGE   w1
      MOVB  3, SP, 0, SP


w1:   LDAB  0, SP  } LDAB 1,SP+
      LEAS  1,+SP  }
      RTS
```

Bitte geben Sie alle Aufgabenblätter wieder ab!

MUSTERPRÜFUNG B		Blatt Nr.:	**8 von 13**
Studiengang:	**Kommunikationstechnik Softwaretechnik Technische Informatik**	Semester:	SWB4, TIB4, KTB4
Prüfungsfach:	**Computerarchitektur 3**	Fachnummer:	**4021**
Hilfsmittel:	**Vorlesungs- und Labormanuskript, Fachliteratur, Taschenrechner**	Dauer:	**90 min**

b) Tragen Sie in die folgende Tabelle den Zustand des Stacks zu dem Zeitpunkt ein, zu dem die als „Zeile (*)“ markierte C-Anweisung ausgeführt wird und geben Sie an, auf welche Speicherzelle der Stack Pointer zu diesem Zeitpunkt zeigt.

Lösung zu Frage 3.2b:



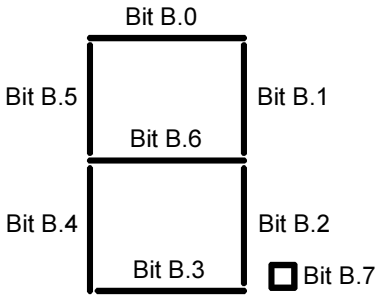
Stackbelegung	**Anfang**	**. . .**
		. . .
	SP →	**tempMax**
		Rücksprung-
		Adresse (16bit)
		Parameter a
		Belegter Bereich
	Ende	**. . .**

MUSTERPRÜFUNG B		Blatt Nr.:	**9 von 13**
Studiengang:	**Kommunikationstechnik Softwaretechnik Technische Informatik**	Semester:	SWB4, TIB4, KTB4
Prüfungsfach:	**Computerarchitektur 3**	Fachnummer:	**4021**
Hilfsmittel:	**Vorlesungs- und Labormanuskript, Fachliteratur, Taschenrechner**	Dauer:	**90 min**

Aufgabe 4: HCS12-Peripheriebausteine (35 Punkte):

Im Programmfragment auf der folgenden Seite werden der A/D-Umsetzer, die PWM- und RTI-Einheit des HCS12 sowie eine Stelle der Sieben-Segment-Anzeige des Dragon12-Entwicklungsboards verwendet. Dabei soll die Helligkeit der Sieben-Segment-Anzeige vom Benutzer durch das Potentiometer am A/D-Umsetzer Kanal 7 eingestellt und über das PWM-Signal P.0 verändert werden können.

Die Kathoden der LEDs der Sieben-Segment-Anzeige sind miteinander verbunden und an Port P.0 angeschlossen. Die Anoden sind über Vorwiderstände an den Port B.7…0 angeschlossen. Die Zuordnung der Segmente ist wie folgt:



4.1
Die Initialisierung der verschiedenen Peripheriekomponenten erfolgt in den Unterprogrammen **`initPorts, initADC, initPWM, initRTI`**. Geben Sie den Programmcode für das Unterprogramm **`initPorts`** an, das alle notwendigen Portanschlüsse für den Betrieb der Sieben-Segment-Anzeige geeignet initialisiert. Die Sieben-Segment-Anzeige soll dabei eingeschaltet werden und den Wert ‚8' anzeigen. Die ebenfalls an Port B angeschlossenen 8 einzelnen LEDs auf dem Dragon12-Board sowie die drei anderen an P.1 bis P.3 angeschlossenen Sieben-Segment-Anzeigen sollen dunkel bleiben, wenn das Programm läuft. Vergessen Sie nicht, den Programmcode so zu kommentieren, dass der Sinn der verschiedenen Befehle klar wird.

Lösung zu Frage 4.1:

```
initPorts:                  ; Initialize ports B, J, P
     BSET DDRJ, #2          ; Port J.1 as output
     BSET PTJ,  #2          ; Port J.1 = 1 --> Deactivate LEDs

     MOVB #$FF, DDRB        ; Port B as output
     MOVB #$7F, PORTB       ; Port B = Seven Segment Display '8'

     BSET DDRP, #$0F        ; Port P.3…0 as outputs (only P.0 required)
     MOVB #$0E, PTP         ; P.3..1 inactive high, P.0 active low
     RTS
```

Bitte geben Sie alle Aufgabenblätter wieder ab!

MUSTERPRÜFUNG B		Blatt Nr.:	**10 von 13**
Studiengang:	**Kommunikationstechnik Softwaretechnik Technische Informatik**	Semester:	SWB4, TIB4, KTB4
Prüfungsfach:	**Computerarchitektur 3**	Fachnummer:	**4021**
Hilfsmittel:	**Vorlesungs- und Labormanuskript, Fachliteratur, Taschenrechner**	Dauer:	**90 min**

Unvollständiger Programmcode zu Aufgabe 4:

```
        XDEF Entry, main
        XREF __SEG_END_SSTACK
        INCLUDE 'mc9s12dp256.inc'

.data: SECTION
value: DS.B 1

.const:SECTION
decoder: DC.B $3F, $06, $5B, $4F, $66, $6D, $7D, $07
         DC.B $7F, $6F, $77, $7C, $39, $5E, $79, $71

.init:  SECTION

;***** Hauptprogramm **********************************
main:
Entry:  LDS  #__SEG_END_SSTACK
        CLI

        JSR   initPorts    ; Initialisiere die Hardware-Peripherie
        JSR   initADC
        JSR   initPWM
        JSR   initRTI

loop:   LDAB  value        ; Letzter ADC Messwert als Duty Cycle
        STAB  PWMDTY0      ; für PWM Kanal 0
        LSRB               ; Verwende die unteren 4 bit des letzten
        LSRB               ; ADC Messwertes
        LSRB
        LSRB
        CLRA
        TFR   D, X
        LDAB  decoder, X   ; … und gebe den Wert codiert auf der
        STAB  PORTB        ;   Sieben-Segment-Anzeige aus

        BRA   loop

;***** Unterprogramme *********************************
initADC:
        MOVB  #%11000000, ATD0CTL2    ; ADC freigeben, Polling
        MOVB  #%00001000, ATD0CTL3    ; Einzelne Wandlung (SC=1)
        MOVB  #%00000101, ATD0CTL4    ; 10bit Auflösung, 2MHz Takt
        MOVB  #%10000111, ATD0CTL5    ; Start 1.Wandlung für Kanal 7
        RTS
```

Fortsetzung auf der nächsten Seite

MUSTERPRÜFUNG B		Blatt Nr.:	**11 von 13**
Studiengang:	**Kommunikationstechnik Softwaretechnik Technische Informatik**	Semester:	SWB4, TIB4, KTB4
Prüfungsfach:	**Computerarchitektur 3**	Fachnummer:	**4021**
Hilfsmittel:	**Vorlesungs- und Labormanuskript, Fachliteratur, Taschenrechner**	Dauer:	**90 min**

```
initPWM:                            ; PWM Einheit konfigurieren
        MOVB   #$01, PWMCLK         ; Taktsignal     P.0 arbeitet mit TSA
        MOVB   #$07, PWMPRCLK                        X = 7
        MOVB   #$05, PWMSCLA                         Y = 5
        MOVB   #$00, PWMPOL         ; Polarität      P.0 beginnt mit Low
        MOVB   #255, PWMPER0        ; Periodendauer 255 * TSA
        BSET   PWME, #$01           ; PWM Kanal 0 freigeben
        RTS
```

4.2

Betrachten Sie die Endlosschleife ab der Marke `loop` im Hauptprogramm. Nehmen Sie dabei an, dass die globale Variable den Wert `value = $80` hat.

a) Welche hexadezimalen Zahlenwerte stehen in den Registern `D` und `X`, wenn das Programm den Befehl `BRA loop` erreicht?

Lösung zu Frage 4.2a:

D = **$007F_H$** X = **0008_H**

b) Welche Anzeige sehen Sie dann auf der Sieben-Segment-Anzeige

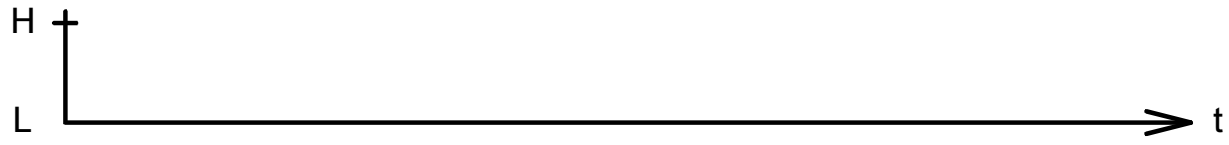
Lösung zu Frage 4.2b:

Die Zahl 8 (alle Segmente ein ausser B.7)

4.3

Im Unterprogramm `initPWM` wird die PWM-Einheit konfiguriert. Skizzieren Sie den zeitlichen Verlauf des Signals am Portanschluss P.0, nachdem `initPWM` aufgerufen und die Hauptprogrammschleife bis zum Befehl `BRA loop` durchlaufen wurde, wobei die Variable den Wert `value = $40` haben soll. Geben Sie die Impuls- und die Periodendauer des Signals in Mikrosekunden an und tragen Sie die Werte in das Zeitdiagramm ein:

Lösung zu Frage 4.3: Zeitverlauf des PWM-Signals P.0



Signalperiode beginnt mit Low-Phase Dauer 3,4ms, dann High-Phase Dauer 10,2ms

Bitte geben Sie alle Aufgabenblätter wieder ab!

MUSTERPRÜFUNG B		Blatt Nr.:	**12 von 13**
Studiengang:	**Kommunikationstechnik Softwaretechnik Technische Informatik**	Semester:	SWB4, TIB4, KTB4
Prüfungsfach:	**Computerarchitektur 3**	Fachnummer:	**4021**
Hilfsmittel:	**Vorlesungs- und Labormanuskript, Fachliteratur, Taschenrechner**	Dauer:	**90 min**

Fortsetzung der Lösung zu Frage 4.3: PWM arbeitet mit f_{BUSCLK}=24MHz

T_A = 2^7 / 24MHz = 5,3µs $T_{SA} = 2 \cdot 5 \cdot T_A$ = 53µs

- Periodendauer des PWM-Signals P.0: $T_P = 255 \cdot T_{SA} = 13,6ms$
- Dauer der High-Phase des PWM-Signals P.0: $T_{High} = T_P - T_{LOW} = 10,2µs$
- Dauer der Low-Phase des PWM-Signals P.0: $T_{Low} = 64 \cdot T_{SA} = 3,4ms$

4.4

Das vorgegebene Programm soll um eine Interrupt-Service-Routine **`isrRTI`** erweitert werden, die durch den RTI-Interrupt periodisch aufgerufen wird. Die Initialisierung und der Start des RTI sollen in der Funktion **`initRTI`** erfolgen.

a) Die erforderlichen Teilerfaktoren für den RTI-Taktteiler werden mit X=7 und Y=0 gewählt. Mit welcher Frequenz wird der RTI-Interrupt aufgerufen?

Lösung zu Frage 4.4a: RTI arbeitet mit f_{OSCCLK}=4MHz

Frequenz des RTI-Interrupts: $f_{RTI} = 4\ MHz / (65536 \cdot 1) = 61Hz$

b) Innerhalb der Interrupt-Service-Routine sollen folgende Aufgaben erledigt werden:

- Warten, bis ein gültiges A/D-Wandlungsergebnis vorliegt
- Abspeichern der oberen 8 bit des 10 bit Wandlungsergebnisses in der globalen Variablen **`value`**
- Starten der nächsten A/D-Wandlung des Kanals 7

Geben Sie den vollständigen, kommentierten (!) Programmcode für die Initialisierungsfunktion **`initRTI`**, die Interrupt-Service-Routine **`isrRTI`** sowie sämtliche darüber hinaus notwendigen Änderungen oder Ergänzungen des vorgegebenen Programms an, damit die Interrupt-Service-Routine korrekt übersetzt und ausgeführt wird.

Lösung zu Frage 4.4b:

```
; --- ROM: Interrupt vector entries
.vect: SECTION
      ORG $FFF0
int7: DC.W isrRTI           ; Interruptvector for interrupt 7 (RTI)

; --- Initialize RTI interrupt
initRTI:
      MOVB #$70, RTICTL     ; RTI interrupt frequency 61Hz
      BSET CRGINT, #$80     ; Enable RTIE interrupt
      RTS
```

Bitte geben Sie alle Aufgabenblätter wieder ab!

MUSTERPRÜFUNG B		Blatt Nr.:	**13 von 13**
Studiengang:	**Kommunikationstechnik Softwaretechnik Technische Informatik**	Semester:	SWB4, TIB4, KTB4
Prüfungsfach:	**Computerarchitektur 3**	Fachnummer:	**4021**
Hilfsmittel:	**Vorlesungs- und Labormanuskript, Fachliteratur, Taschenrechner**	Dauer:	**90 min**

Fortsetzung der Lösung zu Frage 4.4b:

```
; --- Interrupt Service Routine for RTI interrupt
isrRTI:
      BRCLR ATD0STAT0, #$80, isrRTI ; Wait for end of ADC conversion
      LDD  ATD0DR0           ; Read ADC result
      LSRD                   ; … reformat 8 bit right justified
      LSRD
      STAB value             ; … and store in global variable

      MOVB #%10000111, ATD0CTL5 ; Start next measurement on ADC ch. 7

      BSET CRGFLG, #$80      ; Clear interrupt flag

      RTI
```

Bitte geben Sie alle Aufgabenblätter wieder ab!