HOCHSCHULE ESSLINGEN FAKULTÄT INFORMATIONSTECHNIK

MUSTERPRÜFUNG D		Blatt Nr.:	**1 von 14**
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 (35 Punkte)

1.1 Erklären Sie stichwortartig, in welchen Schritten aus einem C-Quellprogramm ein ausführbares Programm im Speicher des HCS12-Mikrocontrollers erstellt wird.

Lösung zu Aufgabe 1.1:

- **Compiler übersetzt C-Programm in hexadezimal codierte Maschinenbefehle (Objektcode) und reserviert Speicherplatz für Variablen und Konstanten**
- **Linker verbindet Objektmodule und Bibliotheken**
- **Locator legt absolute Speicheradressen fest (beim Codewarrior in Linker integriert), Ergebnis: Programmierdatensatz (.abs oder .s19-Datei)**
- **Debugger/Flashlader programmiert Programm in Flash-Speicher des Mikrocontrollers**

1.2 Ein HCS12 C-Programm initialisiert einen Pointer `long *p = 0x1002` und speichert dann über den Pointer mit `*p = 0x89ABCDEF` einen Datenwert im Speicher ab. Aus der zugehörigen Map-Datei sieht man außerdem, dass `&p=0x1000` ist. Geben Sie den Inhalt der folgenden Speicherzellen an. Falls Sie einen Wert nicht kennen, tragen Sie bitte „???“ ein:

Lösung zu Aufgabe 1.2:

Adresse	*Inhalt*
$1000	$10
$1001	$02
$1002	$89
$1003	$AB
$1004	$CD
$1005	$EF
$1006	???
$1007	???

Bitte geben Sie alle Aufgabenblätter wieder ab!

MUSTERPRÜFUNG D		Blatt Nr.:	**2 von 14**
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 Das unten stehende Listing zeigt einen Ausschnitt aus der Map-Datei eines Codewarrior HCS12-Projektes:

- Wieviel Speicherplatz benötigt das Programm im ROM, wieviel im RAM?
- Wie groß ist der Stackbereich und bei welcher Speicheradresse endet er?
- Bei welcher Speicheradresse beginnt die Programmausführung nach einem Reset und wie lang ist der eigentliche Programmcode?
- Stehen die „Variablen“ **`hrs`** und **`Ctext`** im RAM oder im ROM und wieviel Speicher benötigen sie?

```
********************************************************************************
SECTION-ALLOCATION SECTION
Section Name                      Size   Type     From       To       Segment
-------------------------------------------------------------------------------
.init                              718      R     0xC000     0xC2CD   ROM_C000
.data                               45    R/W     0x1000     0x102C   RAM
ticker.asm__ORG00001                 2      R     0xFFE6     0xFFE7   .absSeg0
.const                              28      R     0xC2CE     0xC2E9   ROM_C000
.stack                             256    R/W     0x102D     0x112C   RAM
.vectSeg1_vect                       2      R     0xFFFE     0xFFFF   .vectSeg1

Summary of section sizes per section type:
READ_ONLY (R):          2EE (dec:       750)
READ_WRITE (R/W):       12D (dec:       301)

********************************************************************************
VECTOR-ALLOCATION SECTION
    Address     InitValue   InitFunction
-------------------------------------------------------------------------------
     0xFFFE         0xC000   Entry

********************************************************************************
OBJECT-ALLOCATION SECTION
     Name                 Module                 Addr    hSize    dSize     Ref  Section
-------------------------------------------------------------------------------
MODULE:                   -- main.asm.o --
- PROCEDURES:
     Entry                                       C000       17       23       0   .init
     main                                        C000        0        0       0   .init
     . . .
     cont1                                       C025        2        2       0   .init
- VARIABLES:
     clockEvent                                  1000        1        1       4   .data
     hrs                                         1015        2        2       6   .data
     . . .
     Ctext                                       C2D7        9        9       1   .const
- LABELS:
     __SEG_END_SSTACK                            112D        0        0       1
     . . .
```

Lösung zu Aufgabe 1.3:

Speicherbedarf im ROM:	**750 Byte**	Speicherbedarf im RAM:	**301 Byte**
Stack-Länge:	**256 Byte**	Stack-Endadresse:	**$112C**
Programmbeginn:	**$C000**	Programmlänge:	**718 Byte**
Variable **`hrs`**:	**2 Byte im RAM**	Variable **`Ctext`**:	**9 Byte im ROM**

Bitte geben Sie alle Aufgabenblätter wieder ab!

MUSTERPRÜFUNG D		Blatt Nr.:	**3 von 14**
Studiengang:	**Kommunikationstechnik Softwaretechnik Technische Informatik**	Semester:	SWB4, TIB4, KTB4
Prüfungsfach:	**Computerarchitektur 3**	Fachnummer:	**4021**
Hilfsmittel:	**Vorlesungs- und Labormanuskript, Fachliteratur, Taschenrechner**	Dauer:	**90 min**

1.4 a) Was geschieht, wenn in einem HCS12-Mikrocontroller gleichzeitig der RTI- und der ADC-Interrupt ausgelöst werden?

b) Was geschieht, wenn der Microcontroller bereits die ADC-Interrupt-Service-Routine ausführt und während dieser Interrupt-Service-Routine ein RTI-Interrupt auftritt?

Listing und Lösung zu Aufgabe 1.4:

a) Wenn das Interrupt-Masken-Bit in CCR gesetzt ist, passiert gar nichts.

Wenn das Interrupt-Masken-Bit in CCR gelöscht ist, wird zuerst die RTI-Interrupt-Service-Routine ausgeführt, da sie die höhere Priorität hat. Nach deren Rückkehr wird die ADC-ISR ausgeführt.

b) Innerhalb einer ISR sind weitere Interrupts normalerweise nicht möglich, daher wird die ADC-ISR vollständig ausgeführt. Anschließend wird dann die RTI-ISR ausgeführt.

Falls der Programmierer innerhalb der ADC-ISR mit CLI bewusst weitere Interrupts frei geschaltet hat, wird die ADC-ISR durch die RTI-ISR unterbrochen und erst anschließend zu Ende ausgeführt.

1.5 In Vorbereitungsaufgabe 2.6 des Laborversuchs 1 haben Sie ein Unterprogramm **`decToASCII`** entwickelt, mit dem eine Dualzahl in einen ASCII-String umgewandelt wurde. Beschreiben Sie den Algorithmus, den Sie dort realisiert haben mit Hilfe eines Programmablaufplans. Die Beschreibung soll unabhängig von der Assemblersprache des HCS12 erfolgen, so dass der Algorithmus leicht auch auf andere Mikrocontroller portiert werden kann. Soweit notwendig, dürfen Sie in der Beschreibung C-artige Datentypen und -Befehle verwenden. Zur Vereinfachung sollen Sie hier davon ausgehen, dass es sich bei der Zahl um eine 8 bit Betragszahl handelt. Das Vorzeichen muss nicht ausgegeben werden.

Lösung zu Aufgabe 1.5:

Siehe Laborversuch 1

MUSTERPRÜFUNG D		Blatt Nr.:	**4 von 14**
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 Aufgabe 1.5:

Bitte geben Sie alle Aufgabenblätter wieder ab!

MUSTERPRÜFUNG D		Blatt Nr.:	**5 von 14**
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 (35 Punkte)

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

```
unsigned int func(unsigned int *u, unsigned int *v, unsigned char n)
```

Listing zu Aufgabe 2:

```
 1  func: STAB  4,-SP    ; unsigned char vN = n (at 0,SP)
 2        CLRB
 3        CLRA
 4        STD   2, SP    ; unsigned int temp = 0 (at 2,SP)
 5        STAB  1, SP    ; unsigned int i = 0    (at 1,SP)
 6        BRA   L2       ; goto loop condition

 7   L1:  CLRA           ; loop body → scalar product of u[]*v[]
 8        ASLD           ; use i as index for vector elements
 9        PSHD           ; (i*2 because or we have int vectors!)
10        ADDD  10, SP   ; Pointer to vector element &u[i]→ X
11        TFR   D, X
12        PULD
13        ADDD  6, SP    ; Pointer to vector element &v[i]→ Y
14        TFR   D, Y
15        LDD   0, X     ; temp = u[i] * v[i] + temp
16        LDY   0, Y
17        EMUL
18        ADDD  2, SP
19        STD   2, SP
20        INC   1, SP    ; i++ (vector index)

21   L2:  LDAB  1, SP    ; loop condition: for (…; i < vN; …)
22        CMPB  0, SP
23        BLO   L1

24        LDD   2, SP    ; prepare to return temp
25        LEAS  4, SP    ; remove local variables
26        RTS
```

Im Hauptprogramm, das diese Funktion aufruft, sind folgende Variable definiert:

```
unsigned int C[5] = { 1, 2, 3,  4,  5 };
unsigned int S[5] = { 2, 4, 8, 16, 32 };
unsigned int erg;
```

Aus der MAP-Datei ist bekannt, dass das Array `C` bei der Adresse `$1000` und das Array `S` bei der Adresse `$100A` im Speicher beginnt.

Der Aufruf der Funktion erfolgt im Hauptprogramm mit

```
erg = func( C, S, 5 );
```

Bitte geben Sie alle Aufgabenblätter wieder ab!

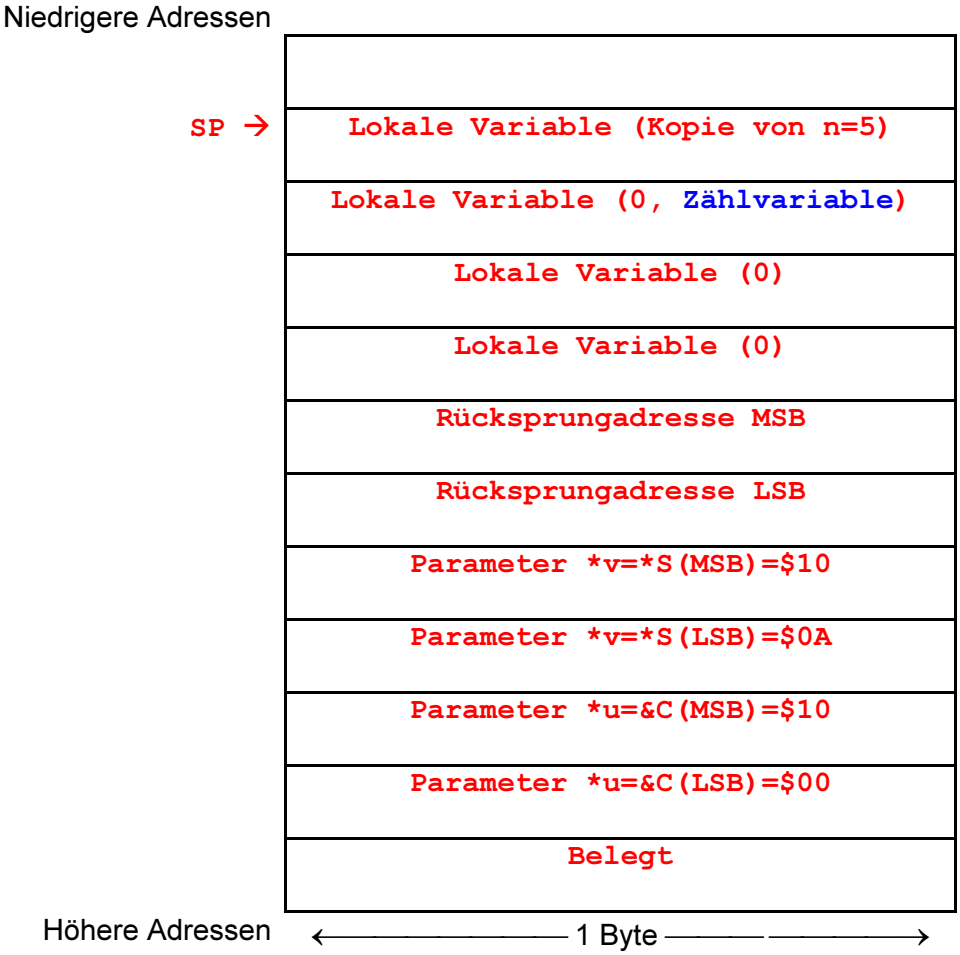
MUSTERPRÜFUNG D		Blatt Nr.:	**6 von 14**
Studiengang:	**Kommunikationstechnik Softwaretechnik Technische Informatik**	Semester:	SWB4, TIB4, KTB4
Prüfungsfach:	**Computerarchitektur 3**	Fachnummer:	**4021**
Hilfsmittel:	**Vorlesungs- und Labormanuskript, Fachliteratur, Taschenrechner**	Dauer:	**90 min**

2.1 Welcher Wert steht nach Ausführung von Zeile 5 im D-Register?

Lösung zu Aufgabe 2.1:

D = 0

2.2 Tragen Sie in die folgende Tabelle den Zustand des Stacks (einschließlich Stackpointer) ein, nachdem der Befehl in Zeile 5 ausgeführt wurde. Geben Sie jeweils den Inhalt der Speicherzellen sowie ihre Funktion für das Unterprogramm **`func`** an.



2.3 Welcher Wert steht im D-Register, wenn die Zeile 19 zum ersten Mal ausgeführt wird?

Lösung zu Aufgabe 2.3:

D = 2

2.4 Wie wird das Ergebnis an die aufrufende Funktion zurückgegeben und welchen Wert hat das Ergebnis, wenn die Funktion mit den oben angegebenen Werten aufgerufen wird?

Lösung zu Aufgabe 2.4:

Ergebnis im D-Register (nach Zeile 24), Wert ist 258.

Bitte geben Sie alle Aufgabenblätter wieder ab!

MUSTERPRÜFUNG D		Blatt Nr.:	**7 von 14**
Studiengang:	**Kommunikationstechnik Softwaretechnik Technische Informatik**	Semester:	SWB4, TIB4, KTB4
Prüfungsfach:	**Computerarchitektur 3**	Fachnummer:	**4021**
Hilfsmittel:	**Vorlesungs- und Labormanuskript, Fachliteratur, Taschenrechner**	Dauer:	**90 min**

2.5 Das Unterprogramm enthält eine `for()`-Schleife:

- In welchem Register und/oder in welcher Speicherzelle wird die Zählvariable der Schleife gespeichert?
- In welcher Programmzeile wird die Zählvariable initialisiert?
- Wo wird sie beim Schleifendurchlauf verändert?
- Wo wird die Endebedingung für die Schleife abgeprüft?

Lösung zu Aufgabe 2.5:

Zählvariable: **Auf dem Stack SP+1, z.B. Zeile 5, 20, 21**

Initialisierung der Zählvariable: **Mit 0 in Zeile 5**

Ändern der Zählvariable: **Inkrementieren in Zeile 20**

Abprüfen der Endebedingung: **in Zeile 21 bis 23 (Abfrage in Zeile 22)**

2.6 Erstellen Sie ein zum Assemblerlisting äquivalentes C-Programm:

Lösung zu Aufgabe 2.6:

```
unsigned int func(unsigned int *u, unsigned int *v, unsigned char n)
{

     unsigned char i;
     unsigned int val=0;

     for (i=0; i < n; i++)
     {    val += u[i] * v[i];
     }
     return val;



}
```

Bitte geben Sie alle Aufgabenblätter wieder ab!

MUSTERPRÜFUNG D		Blatt Nr.:	**8 von 14**
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 (25 Punkte):

3.1

In einem HCS12-Assemblerprogramm sind folgende globalen Variablen definiert:

```
.const: SECTION
          ORG   $D000
tabelle1: DC.B  $11, $22, $33, $44, $55, $66, $77, $88
tabelle2: DC.W  $0123, $4567, $89AB, $CDEF
tabelle3: DC.W  tabelle1, tabelle2
```

Geben Sie den Inhalt der CPU-Register D, X und Y an, nachdem die im jeweiligen Tabellenfeld stehenden Assemblerbefehle nacheinander ausgeführt wurden. Es reicht aus, wenn Sie bei jedem Befehl diejenigen Registerwerte eintragen, die sich jeweils ändern.

Assemblerbefehle	D	X	Y
	$0000	$0000	$0000
LDD tabelle2	$0123		
LDD tabelle2+3	$6789		
LDD tabelle1	$1122		
LDD tabelle1+4	$5566		
LDX #tabelle1+4 LDAB 0, X	$5555	$D004	
LDY #tabelle2 LDD 4, Y	$89AB		$D008
LDD -4, Y	$5566		
LDD 2, -Y	$7788		$D006
LDX #$0002 LDD tabelle2, X	$4567	$0002	
LEAY tabelle3, X			$D012
LDD [tabelle3, X]	$0123		

Bitte geben Sie alle Aufgabenblätter wieder ab!

MUSTERPRÜFUNG D		Blatt Nr.:	**9 von 14**
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 arrayA[8], valA;
int  arrayB[8], valB, *pB, i;
```

Diese Variablen werden im folgenden C-Programmausschnitt 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. Geben Sie den Assembler-Programmcode an:

Lösung zu Frage 3.2:

C-Programm	*HCS12-Assembler-Programm*
`i = 3;`	`MOVW #3, i`

`valA = arrayA[i];`	`LDX  i`
	`LDAB arrayA, X`
	`STAB valA`
	`// MOVB arrayA, X, valA → GEHT NICHT`

`valB = valA;`	`LDAB valA`
	`TFR  B, D (Erweiterung 8 → 16 bit)`
	`STD  valB`

`valA = arrayA[6];`	`MOVB arrayA+6, valA`

`valB = arrayB[i];`	`LDD  i`
	`ASLD`
	`TFR  D, X`
	`LDD  arrayB, X`
	`STD  valB`
	`// MOVW arrayB, X, valB  → GEHT NICHT`

`pB = &valB;`	`MOVW #valB, pB`
`*pB = 4;`	`LDX  pB` `MOVW #4, 0, X`

Bitte geben Sie alle Aufgabenblätter wieder ab!

MUSTERPRÜFUNG D		Blatt Nr.:	**10 von 14**
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: Temperaturregelung mit HCS12 (40 Punkte):

Mit einem Dragon12-Entwicklungsboard soll der Prototyp einer einfachen Temperaturregelung und Temperaturüberwachung aufgebaut werden.

Die Messung der Isttemperatur ϑ_{ist} erfolgt durch einen temperaturabhängigen Widerstand. Der Widerstand wird in einer Spannungsteilerschaltung betrieben, dessen Ausgangsspannung durch den Kanal AD0.6 des ADC des Mikrocontrollers gemessen wird. Das Ergebnis der AD-Umsetzung ist die Zahl N_{ist}.

Die Solltemperatur ϑ_{soll} wird über ein Potentiometer am Kanal AD0.7 des ADC vorgegeben und in die Zahl N_{soll} umgesetzt. Die übrigen ADC-Eingänge werden nicht verwendet. Der Zusammenhang zwischen Temperatur ϑ und Spannung u an beiden ADC-Eingängen ist

$$u = \frac{\vartheta + 40°C}{80°C} \cdot U_{ref}.$$

Der Spannungsteiler und der ADC arbeiten mit der Referenzspannung U_{ref} = 5V. Der ADC wird mit Hilfe des folgenden Unterprogramms initialisiert:

```
AdcInit:  MOVB #%11000000, ATD0CTL2
          MOVB #%00010000, ATD0CTL3
          MOVB #%00000101, ATD0CTL4
          MOVB #%10010110, ATD0CTL5
          RTS
```

Das Stellventil der Heizungsregelung wird über den PWM-Ausgang Port P.0 des HCS12 angesteuert. Das Tastverhältnis des PWM-Ausgangs sei $\lambda = T_E / T_P$. Dabei soll λ bei konstanter Periodendauer T_P durch den Regelalgorithmus vergrößert werden, wenn $\vartheta_{soll} > \vartheta_{ist}$ ist. Die übrigen PWM-Ausgänge werden nicht verwendet.

Der Regelalgorithmus soll periodisch ca. alle 100ms berechnet werden. Dazu wird der RTI-Interrupt des HCS12 eingesetzt. Die Quarzfrequenz auf dem Entwicklungsboard beträgt 4MHz, die Ausgangsfrequenz der PLL ist 24MHz.

Falls die Isttemperatur ϑ_{ist} < 5°C ist, soll über den Piepser am Port T.5 ein Warnsignal mit einer Frequenz von 500Hz ausgegeben werden. Die Ein- und Ausschalten des Piepsers erfolgt über ein Unterprogramm **`void beeper(unsigned int frequenz)`**, wobei zum Einschalten die Frequenz direkt in Hz übergeben werden darf. Mit **`frequenz=0`** wird der Piepser abgeschaltet. Eine vorherige Initialisierung ist nicht erforderlich.

4.1

Geben Sie den Programmcode für das Unterprogramm **`RtiInit`** in HCS12-Assembler an, mit dem der RTI-Interrupt initialisiert wird. Außerdem soll der notwendige Eintrag in der Assembler-Vektor-Tabelle für die zugehörige Interrupt-Service-Routine **`RtiIsr`** angelegt werden. Vergessen Sie nicht, diese und alle anderen Programme mit verständlichen Kommentaren zu versehen! (Hinweis: Die ISR selbst muss hier noch nicht geschrieben werden.)

Bitte geben Sie alle Aufgabenblätter wieder ab!

MUSTERPRÜFUNG D		Blatt Nr.:	**11 von 14**
Studiengang:	**Kommunikationstechnik Softwaretechnik Technische Informatik**	Semester:	SWB4, TIB4, KTB4
Prüfungsfach:	**Computerarchitektur 3**	Fachnummer:	**4021**
Hilfsmittel:	**Vorlesungs- und Labormanuskript, Fachliteratur, Taschenrechner**	Dauer:	**90 min**

Lösung zu Frage 4.1:

```
RtiInit:   MOVB  #$75, RTICTL     ; Timer-Periode ca. 98ms (X=7, Y=5)
           BSET  CRGINT, $80      ; RTI Interrupt freigeben
           RTS
```

Eintrag in die Interrupt-Vektor-Tabelle:

```
.vect:     SECTION
           ORG   $FFF0            ; Interruptvektor 7: RTI Interrupt
rtiInt:    DC.W  RtiIsr
```

4.2

Als Taktquelle für den PWM-Ausgang P.0 wird Takt T_{SA} eingesetzt. Die Teilerfaktoren sollen mit $x_A = y_{SA} = 7$ und PWMPER=255 konfiguriert werden. Welche Periodendauer T_P wird das PWM-Signal haben? Geben Sie den Programmcode für das Unterprogramm **`PwmInit`** in HCS12-Assembler an, mit dem die PWM-Einheit initialisiert werden muss.

Lösung zu Frage 4.2:

Periodendauer $T_P = PWMPER \cdot 2 \cdot y_{SA} \cdot 2^{x_A} / f_{BUSCLK} = 255 \cdot 2 \cdot 7 \cdot 2^7 / 24MHz = 19ms$

```
PwmInit:                    ; Initialisierung P.0, andere Kanäle aus
     MOVB #$01, PWMCLK      ; Kanal 0 mit TSA
     MOVB #$07, PWMPRCLK    ; xA  = 7
     MOVB #$07, PWMSCLA     ; ySA = 7
     MOVB #$01, PWMPOL      ; Periode beginnt mit H (vorgegeben ΔN↑->λ↑)
     MOVB #255, PWMPER0     ; PWMPER = 255
(    MOVB #128, PWMDTY0     ; PWMDTY = 128 nicht vorgegeben)
     BSET PWME, #$01        ; Kanal P.0 freigeben
     RTS
```

Bitte geben Sie alle Aufgabenblätter wieder ab!

MUSTERPRÜFUNG D		Blatt Nr.:	**12 von 14**
Studiengang:	**Kommunikationstechnik Softwaretechnik Technische Informatik**	Semester:	SWB4, TIB4, KTB4
Prüfungsfach:	**Computerarchitektur 3**	Fachnummer:	**4021**
Hilfsmittel:	**Vorlesungs- und Labormanuskript, Fachliteratur, Taschenrechner**	Dauer:	**90 min**

4.3

Welche Zahlenwerte N_{min} und N_{max} liefert der ADC, wenn die Temperatur die Werte $\vartheta_{min} = -40°C$ bzw. $\vartheta_{max} = +40°C$ hat?

Der Regelalgorithmus ist $\lambda = K1 * (N_{soll} - N_{ist}) + K2$. Wie groß dürfte der positive Wert `K1` höchstens gewählt werden, wenn `K2 = +128` festgelegt wird und für die Ansteuerung der PWM-Einheit im Mikrocontroller $0 \le \lambda \le 255$ gelten muss?

Lösung zu Frage 4.3:

N_{min} = 0 N_{max} = 1023

$K1_{max}$: K1 * (1023 – 0) + 128 = 255 → $K1_{max}$= 127/1023 = 0,124

4.4

Schreiben Sie die Interrupt-Service-Routine `RtiIsr`. Die ISR soll folgendes erledigen:

- Die Soll- und die Isttemperatur sollen vom AD-Umsetzer eingelesen werden. Die vorderen 8bit des 10bit Messwertes N_{ist} sollen in der globalen 8bit Variable `TempIst` abgespeichert werden.
- Die Regeldifferenz $N_{soll} - N_{ist}$ soll berechnet und im Register D an das Unterprogramm `Regler` übergeben werden. (Hinweis: Das Unterprogramm `Regler` selbst muss nicht geschrieben werden.)
- Mit der vom Unterprogramm `Regler` berechneten Stellgröße λ (Rückgabe im Register B) wird der PWM-Ausgang P.0 angesteuert.
- Am Ende der Interrupt-Service-Routine wird die nächste AD-Umsetzung gestartet.

Lösung zu Frage 4.4:

```
RtiIsr:                    ; *** RTI Interrupt-Service-Routine ***
     BRCLR ATD0STAT0, #$80, RtiIsr    ; warten auf Wandlungsende

     LDD   ATD0DR0         ; Ergebnis Kanal 6 (Istwert)
     LSRD                  ; obere 8bit abspeichern
     LSRD
     STAB  TempIst
```

Bitte geben Sie alle Aufgabenblätter wieder ab!

MUSTERPRÜFUNG D		Blatt Nr.:	**13 von 14**
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 4.4:

```
      LDD   ATD0DR1           ; Ergebnis Kanal 7 (Sollwert)
      SUBD  ATD0DR0           ; Regeldifferenz Soll - Ist in D
      JSR   Regler            ; Regler aufrufen, Rückgabewert in B

      STAB  PWMDTY0           ; Tastverhältnis für PWM P.0

      MOVB #%10010110, ATD0CTL5  ; AD-Umsetzung für Kanal 6+7 starten

      BSET  CRGFLG, $80       ; RTI Interruptflag zurücksetzen
      RTI
```

4.5

Das Hauptprogramm ruft zunächst die Initialisierungfunktionen für die ADC-, PWM- und die RTI-Einheit auf und geht dann in eine Endlosschleife. In der Endlosschleife wird die Ist-Temperatur (in der Variable **`TempIst`**, siehe Frage 4.4) ständig abgefragt und mit Hilfe eines Aufrufs der Funktion **`beeper()`** der Warnton eingeschaltet wird, falls die Temperatur ϑ_{ist} < 5°C ist. Bei größeren Temperaturen soll der Piepser stumm bleiben.

Schreiben Sie den Programmcode des Hauptprogramms. Definitionen wie **`XDEF`**, **`INCLUDE`**, **`SECTION`** usw. dürfen weggelassen werden. Die Unterprogramme **`AdcInit()`**, **`Beeper()`** usw. müssen nicht (mehr) geschrieben werden.

Lösung zu Frage 4.5:

```
WARNTEMP: EQU  ((5+40)*1024/(80*4))   ; Umrechnung der Warntemperatur
                                      ; 5°C in normierten 8bit Wert
main:
Entry:
      LDS   #__SEG_END_SSTACK
      CLI
      MOVB    #0,TempIst    ; Globale Variable initialisieren

      JSR   PwmInit         ; PWM initialisieren
      JSR   AdcInit         ; ADC initialisieren
      JSR   RtiInit         ; RTI initialisieren
```

Bitte geben Sie alle Aufgabenblätter wieder ab!

MUSTERPRÜFUNG D		Blatt Nr.:	**14 von 14**
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 4.5:

```
loop:    LDAB    TempIst          ; Temperatur unter 5°C ?
         CMPB    #WARNTEMP
         BLO     beepOff          ; --> ja: Piepser ein

beepOn:  LDD     #500             ; Piepserfrequenz 500Hz
         JSR     Beeper
         BRA     loop

beepOff:LDD      #0               ; Piepserfrequenz 0, d.h. aus
         JSR     Beeper
         BRA     loop
```