Mikrocontrollertechnik MC/ Automatisierungstechnik AU1/ Hard- und Software-Engineering HS1

Klausur vom 17. 3. 2015

Aufgaben und Musterlösungen

1. Welchen Wertebereich hat eine 12 Bits lange ganze Binärzahl in Zweierkomplementdarstellung?
(5 Punkte)

Von –2048 bis 2047.

2. Eine Unterbrechungsbehandlungsroutine arbeitet mit den Registern r0, r1, r22, r23, r24 und r25. Schreiben Sie zwei Assemblerprogrammstücke, eines zum Eintritt in die Unterbrechungsbehandlug (ENTER) und eines zum Verlassen (EXIT).
(10 Punkte)

ENTER:	EXIT:
push r0 push r1 push r22 push r23 push r24 push r25 in r25,sreg push r25	pop r25 out sreg,r25 pop r25 pop r24 pop r23 pop r22 pop r1 pop r0 reti
Alle Register auf den Stack, Reihenfolge egal. Das Statusregister (sreg) auf den Stack.	Alle Register zurück, das Statusregister (sreg) zurück. Umgekehrte Reihenfolge. Mit reti-Befehl abschließen (Rückkehr aus Unterbrechungsbehandlung).

3. Schreiben Sie ein Assemblerprogrammstück, das folgenden Ablauf implementiert (Abb. 1).
(8 Punkte)

```
sbrs r12,3          ; überspringen, wenn Bit 3 in r12 gesetzt ist
call REQUEST
```

Für umständlichere, aber funktionsfähige Lösungen gibt es ein paar Punkte weniger...

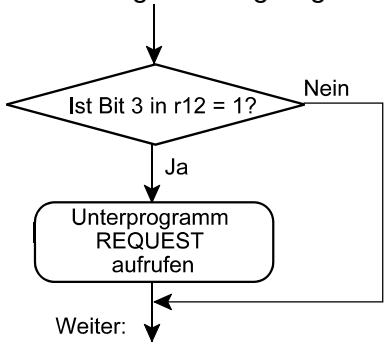

Abb. 1

4. Erläutern Sie wenigstens drei Möglichkeiten der Parameterübergabe beim Unterprogrammruf.
(6 Punkte)

- In Registern
- Im Speicher
- Im Stack
- Im Anschluß an den Aufrufbefehl

5. Erläutern Sie kurz (Aufzählung), aus welchen Teilen eine typische Anweisungszeile in einem Assemblerprogramm besteht (alle Teile aufzählen, die in einer solchen Zeile vorkommen können).
(5 Punkte)

- Marke (Label)
- Operationscode
- Operand(en) (einer oder zwei)
- Kommentar

6. Der E-A-Port C eines AVR-Mikrocontrollers wird vollständig zu Ausgabezwecken verwendet. Es soll aber nur eine Ausgabe in die die höherwertige Tetrade (Bits 7...4) erfolgen. Die Bitpositionen 3...0 sollen bleiben, wie sie sind (Abb. 2).

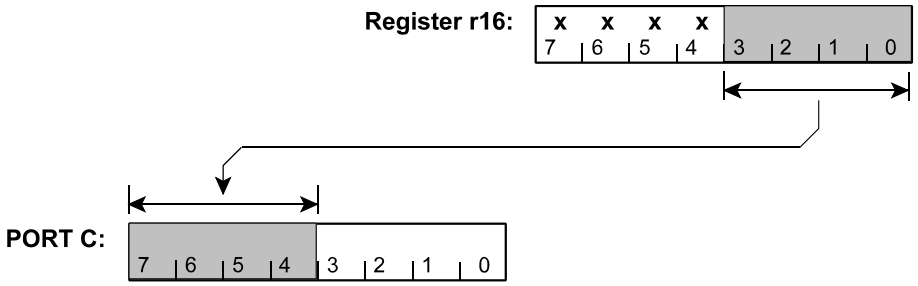

Abb. 2

Schreiben Sie ein geeignetes Assemblerprogrammstück. Die auszugebenden Bits stehen in der niederwertigen Tetrade von Register r16. Der Inhalt der höherwertigen Tetrade ist unbestimmt (beliebige Einsen und Nullen). Der Inhalt von r16 darf verändert werden. Auch dürfen Sie beliebig viele weitere Register verwenden.
(10 Punkte)

```
in r17,portc        ; Port C einlesen
andi r17,0x0f       ; die höherwertige Tetrade löschen
swap r16            ; die niederwertige Tetrade von r16 zur höherwertigen machen
andi r16,0xf0       ; die niederwertige Tetrade löschen
or r17,r16          ; die höherwertige Tetrade von r16 in r17 einfügen
out portc,r17       ; nach Port C ausgeben
```

7. Ein 16-Bit-Wort ist um 2 Bits nach rechts zu rotieren.

a) Überlegen Sie,was herauskommen muß. Wir betrachten hierzu ein 16-Bit-Wort in zwei Registern:

Bitposition:	7	6	5	4	3	2	1	0
r16:	7: **1**	6: **0**	5: **0**	4: **1**	3: **1**	2: **0**	1: **1**	0: **1**
r17:	15: **1**	14: **1**	13: **1**	12: **0**	11: **0**	10: **0**	9: **0**	8: **0**

Tragen Sie hier ein, wie das Ergebnis aussieht:

Bitposition:	7	6	5	4	3	2	1	0
r16:	7: **0**	6: **0**	5: **1**	4: **0**	3: **0**	2: **1**	1: **1**	0: **0**
r17:	15: **1**	14: **1**	13: **1**	12: **1**	11: **1**	10: **0**	9: **0**	8: **0**

b) Schreiben Sie ein Assemblerprogrammstück, das einen solchen Rotationsablauf implementiert. Das Wort steht – s. oben – in den Registern r16 und r17. Sie dürfen beliebig viele weitere Register verwenden. Der ursprüngliche Inhalt dieser Register muß aber erhalten bleiben.

(15 Punkte)

Beim Rotieren müssen die links oben ausgeschobenen Bits rechts unten wieder einlaufen (in Bezug auf die obigen Abbildungen). Es gibt mehrere Möglichkeiten. Hier nehmen wir ein weiteres Register, um die ausgeschobenen Bits zeitweise zu speichern.

```
push r18          ; das zusätzliche Register retten
ldi r18,0         ; das zusätzliche Register mit Null füllen
lsr r17           ; die erste Rechtsverschiebung
ror r16
ror r18
lsr r17           ; die zweite Rechtsverschiebung
ror r16
ror r18
or r17,r18        ; die in r18 gespeicherten Bits in r17 einfügen
pop r18           ; den alten Inhalt von r18 wieder herstellen
```

8. Zu einer 16-Bit-Binärzahl in den Registern r10 und r11 (Abb. 3) ist der Festwert 4567H zu addieren. Bei Bedarf dürfen weitere Register nach Belieben genutzt werden. Deren bisheriger Inhalt darf aber nicht verlorengehen.

(10 Punkte)

r10	Bits 7...0
r11	Bits 15...8

Abb. 3

Mit den Registern r10 und r11 kann man keine Direktwertbefehle ausführen. Deshalb nehem wir r16 zu Hilfe.

```
push r16              ; das zusätzliche Register retten
ldi r16,low(0x4567)   ; das niederwertige Byte laden
add r10,r16           ; und addieren
ldi r16,high(0x4567)  ; das höherwertige Byte laden
adc r11,r16           ; und mit Ausgangsübertrag addieren
pop r16               ; den alten Inhalt von r16 wieder herstellen
```

9. Wozu dienen **org**-Anweisungen?

(5 Punkte)

Um die Startadresse zum Assemblieren einzustellen.

10. Abb. 4 zeigt unvollständige Befehlsformate am Beispiel von Additionsbefehlen (ADD). Der Operationscode ist vorhanden, alles andere fehlt. Vervollständigen Sie die Formate für a) eine Akkumulatormaschine, b) eine Stackmaschine, c) eine CISC-Maschine mit Universalregistersatz, d) eine Zweiadreß-RISC-Maschine. Erläutern Sie kurz, wie die Befehle jeweils wirken. Beispiel: "b) Addiert den Inhalt der ersten Adresse zum Inhalt der zweiten Adresse und speichert das Ergebnis im Akkumulator".

(20 Punkte)

a) Akkumulatormaschine

ADD	

b) Stackmaschine

ADD	

c) CISC-Maschine mit Universalregistersatz

ADD	

d) Zweiadreß-RISC-Maschine

ADD	

Abb. 4

a) Akkumulatormaschine

ADD	Operandenadresse

b) Stackmaschine

ADD

c) CISC-Maschine mit Universalregistersatz

ADD	R1	R2	Displacement

d) Zweiadreß-RISC-Maschine

ADD	R1	R2

a) Addiert den Operanden gemäß Operandenadresse zum Inhalt des Akkumulators und speichert das Ergebnis im Akkumulator.

b) Entfernt beiden obersten Stackeinträge aus dem Stack, addiert sie zueinander und legt das Ergebnis auf den Stack. Operationsbefehle echter Stackmaschinen haben keinen Adreßteil...

c) Addiert den Operanden zum Inhalt des Registers R1 und speichert das Ergebnis in R1. Die Operandenadresse ergibt sich aus dem Inhalt des Registers R2 (Basisadresse) und dem Displacement.

d) Addiert beide Registerinhalte zueinander und speichert das Ergebnis in R1.

11. Ein AVR-Mikrocontroller soll als Teil eines Prüfgerätes arbeiten. Die zu prüfende Einrichtung ist an Port C angeschlossen. Der Prüfablauf besteht darin, Bitmuster über Port C einzulesen und mit Sollwerten zu vergleichen, die als Bytekette im Flash gespeichert sind. Schreiben Sie ein entsprechendes Assemblerprogrammstück. Hierzu dürfen Sie beliebige Register verwenden. Der Prüfablauf im Überblick:

    1. Die Bytekette adressieren. Sie ist im Flash über eine Marke SOLLWERTE erreichbar. Die Bytes sind Binärzahlen bzw. Bitmuster.
    2. Das erste Byte der Zeichenkette gibt deren Länge an, und zwar vom zweiten Byte bis zum Ende. Das zweite Byte ist das erste Sollwertbyte.

3. Der eigentliche Prüfablauf: von Port C einlesen und mit dem Sollwertbyte vergleichen. Bei Ungleichheit zu einer Marke ERROR verzweigen. Bei Gleichheit den Ablauf mit dem nächsten Sollwertbyte wiederholen (Prüfschleife). Wurde das letzte Sollwertbyte verglichen, ist der Prüfablauf zu Ende. Dann zu einer Marke READY verzweigen.

(15 Punkte)

```
ldi zl,low(SOLLWERTE*2)          ; die Zeichenkettenadresse ins Z-Register
ldi zh,high(SOLLWERTE*2)

lpm r16,z+                       ; das Längenbyte nach r16

testloop:                        ; die Prüfschleife
lpm r17,z+                       ; das Sollwertbyte nach r17
in r18,pinc                      ; den Prüfwert einlesen (vom PIN, nicht vom PORT)
cp r17,r18                       ; sind beide gleich?
brne ERROR                       ; NEIN. Fehler
dec r16                          ; JA. Den Längenzähler vermindern
brne testloop                    ; zurück zur Schleife, wenn nicht Null
rjmp READY                       ; Längenzähler = 0. Fertig
```

12. Es geht nochmals um eine 16-Bit-Binärzahl in den Registern r10 und r11 (vgl. Abb. 3). Geben Sie Assemblerprogrammstücke an, die folgende Aufgaben lösen (hierzu dürfen Sie beliebig viele weitere Register einsetzen):

    a) Die Zahl um drei Bits nach links verschieben. Die frei gewordenene Bitpositionen mit Einsen auffüllen.
    b) Die Zahl um zwei Bits arithmetisch nach rechts verschieben.
    c) Die Differenz 1234 – Zahl ausrechnen. Ergebnis in r10 und r11.
    Also: 1234 – <r11,r10> => <r11,r10>

(15 Punkte)

a)	**b)**	**c)**

lsl r10
rol r11
lsl r10
rol r11
lsl r10
rol r11
ldi r16,0x07
or r10,r16
	asr r11
	lsr r10
	asr r11
	lsr r10
		ldi r16,low(1234)
		ldi r17,high(1234)
		sub r16,r10
		sbc r17,r11
		mov r10,r16
		mov r11,r17