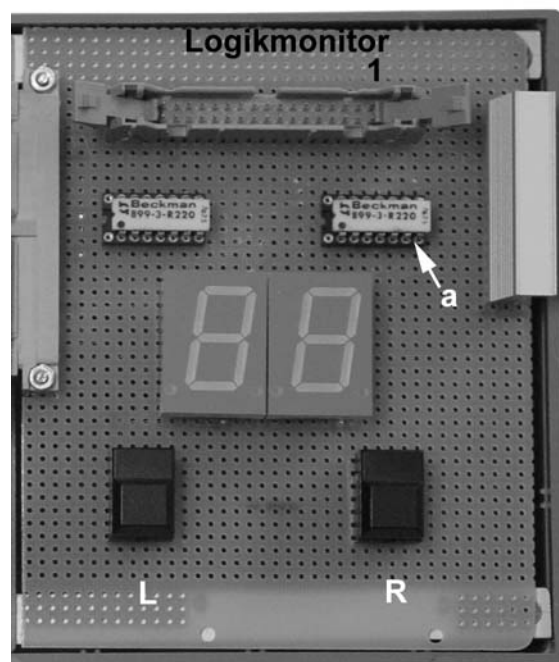


Übungsaufgaben

Stand 1.1

5. 5. 2008

1. Lottozahlen. Schreiben Sie ein Programm zum Ziehen von Lottozahlen. Darstellung auf zwei Siebensegmentanzeigen. Die Anzeigen sind segmentweise anzusteuern (aktiv Low). Prinzip: Solange eine Taste betätigt wird, zählt ein Zähler modulo 49. Bei losgelassener Taste wird der aktuelle Zählerstand als Dezimalzahl zwischen 1 und 49 angezeigt. Die Zufallswirkung ergibt sich infolge des im Vergleich zum Tastendruck sehr schnellen Zählumlaufs. Als Taste ist die rechte Taste der Anzeigetafel zu verwenden (beide Tasten der Anzeigetafel wirken aktiv Low).



Port D: linke Siebensegmentanzeige und linke Taste (L).

Port A: rechte LED-Stelle (LCD 1) + rechte Taste

7	6	5	4	3	2	1	0
Taste L	G	F	E	D	C	B	A
IN	OUT						

Port D: linke LED-Stelle (LCD 1) + linke Taste

7	6	5	4	3	2	1	0
Taste L	G	F	E	D	C	B	A
IN	OUT						

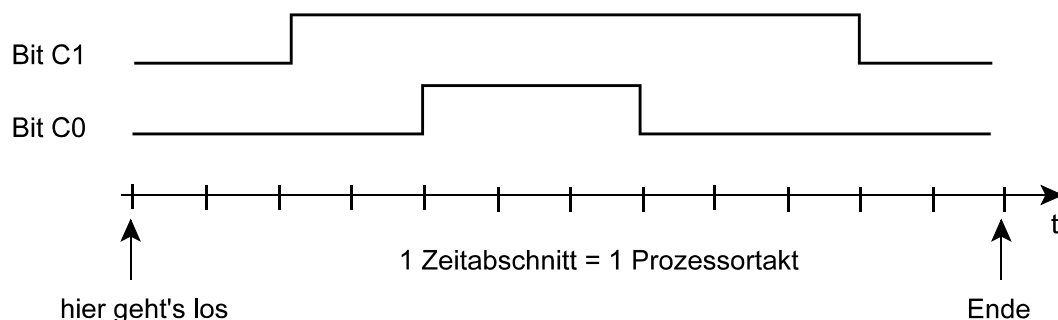
- Die Aufgabe ist in C zu lösen.
- Die Aufgabe ist mittels Assemblerprogrammierung zu lösen.

Vergleichen Sie:

- die Entwicklungszeit,
 - die Laufzeit,
 - die Größe des Maschinenprogramms.
- Stoppuhr. Verwenden Sie die Hardware von Aufgabe 1 und schreiben Sie ein Programm, das eine Stoppuhr nachbildet. Bereich: bis 9,9 s in Schritten von 0,1 s. Hierzu sind beide Tasten der Anzeigetafel auszunutzen. Linke Taste: Löschen. Rechte Taste: Wechsel zwischen Laufen (Zeitzählung) und Stop. In Stellung 9,9 s soll die Uhr grundsätzlich stehenbleiben.

Die Teilaufgaben a), b) entsprechen Aufgabe 1.

- Initialisieren Sie Port A so, daß die Bitpositionen 0 und 1 als Ausgänge wirken und anfänglich mit Null belegt sind.
- Schreiben Sie einen Programmablauf, der über Port A, Bitpositionen 1 und 0 das nachfolgend gezeigte Bitmuster ausgibt.



- Schreiben Sie eine einfache Zeitschleife in Form eines Unterprogramms `TIME_COUNT`. Dessen Funktionen:
 - Laden des Registers `r16` mit dem Festwert `0x38`,
 - Vermindern des Registerinhaltes um 1,
 - Rückkehr, wenn der Registerinhalt den Wert Null erreicht hat.

Sorgen Sie dabei dafür, daß der ursprüngliche Inhalt des Registers r16 erhalten bleibt (mit anderen Worten, das rufende Programm darf gar nicht merken, daß r16 im Unterprogramm benutzt wurde).

6. Die Register r2 bis r5 enthalten jeweils 16 Bits lange Binärzahlen X und Y. Schreiben Sie einen Programmablauf für die Subtraktion X-Y. Hierbei soll das Ergebnis in den Registern r2 und r3 zu stehen kommen.

r2	LO_X
r3	HI_X
r4	LO_Y
r5	HI_Y

7. Wir beziehen uns auf Aufgabe 6. Jetzt ist aber die Subtraktion X-Y so auszuführen, daß das Ergebnis in den Registern r18 und r19 zu stehen kommt. Die Inhalte der Register r2 bis r5 sollen unverändert erhalten bleiben.
8. In den Registern r3 und r4 steht eine 16-Bit-Binärzahl INT_1. Schreiben Sie einen Programmablauf für folgende Berechnung:

$$INT_2 = INT_1 - 25BFH$$

INT_2 belegt dabei die Register r20 und r21.

r3	INT_1_LO
r4	INT_1_HI
	...
r20	INT_2_LO
r21	INT_2_HI

9. Ein Tastenfeld ist an Port B angeschlossen. Es soll abgefragt werden, ob die Taste an Bitposition 3 betätigt ist oder nicht (Tastenwirkung: aktiv low). Das Programmstück:

```

WAIT_3:
        SBIC      PORTB,3
        RJMP     WAIT_3
    
```

Wird das so funktionieren? Geben Sie ggf. eine korrigierte Befehlsfolge an.

10. Der A/D-Wandler eines Mikrocontrollers liefert einen Wert von 10 Bits Länge. Formen Sie diesen Wert in einen 8-Bit-Wert um.

ADC Data Register ADCL und ADCH:

Register	7	6	5	4	3	2	1	0	
ADCH	-	-	-	-	-	.	ADC9	ADC8	
ADCL	ADC7							ADC0	

Das gewünschte Ergebnis:

7	6	5	4	3	2	1	0
ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2

11. Schreiben Sie ein Assemblerprogrammstück, das folgenden Ablauf implementiert.

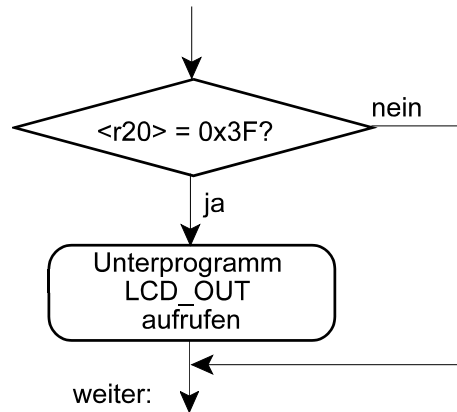


Abb. 4

12. Beim Aufruf eines Unterprogramms werden Register in den Stack gerettet. Das Unterprogramm beginnt mit den folgenden Befehlen. Geben Sie eine dazu passende Befehlsfolge an (im Anschluß an die Marke LEAVE), mit der das Unterprogramm verlassen werden kann.

```

PUSH    TEMP
IN      TEMP, SREG
PUSH    TEMP
PUSH    r5
PUSH    r6
PUSH    r24
PUSH    r25
  
```

...

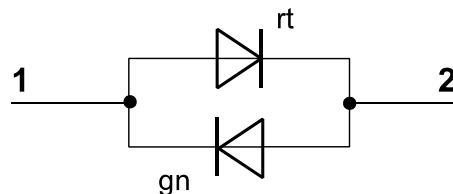
LEAVE:

13. Eine 16-Bit-Binärzahl ist zu runden. Hierzu sind die niedrigstwertigen Bits 1 und 0 auszuwerten:

- Bitbelegung 0,0: nichts tun.
- Bitbelegung 1,0: durch 0, 0 ersetzen (abrunden).
- Bitbelegungen 1,0 und 1,1: durch 0,0 ersetzen und den Rest der Zahl (von Bit 2 an) um Eins erhöhen (aufrunden). Die Zahl darf aber nicht größer werden als der Endwert FFFCH (Sättigungsarithmetik).

Die Binärzahl steht in den Registern ZL (Bits 7...0) und ZH (Bits 15...8). Sie dürfen beliebig viele Arbeitsregister verwenden.

14. Ein Mikrocontroller soll eine Zweifarben-LED ansteuern. Anschluß: Pin 1 an Port B, Bit 0, Pin 2 an Port B, Bit 1 (natürlich über Serienwiderstand...). Die weiteren Bits 7...2 sollen zu Ausgabezwecken verwendet werden.
- wie ist die LED anzusteuern, damit sie (1) grün, (2) rot und (3) gar nicht leuchtet? Geben Sie die Belegungen mit Einsen und Nullen an, z. B. Pin 1 = 1, Pin 2 = 0.
 - Initialisieren Sie die Port B so, daß die LED anfänglich nicht leuchtet.
 - Schreiben Sie ein Programm, das die LED zyklisch (ewig) rot – grün – rot ... blinken läßt. Die Blinkzeit erzeugen Sie mit einem (fertigen) Unterprogramm `MILLISEC`, dem die Zeit in Millisekunden in den Registern `r24` und `r25` übergeben wird (haben wir im Praktikum ausprobiert). Stellen Sie das Blinkintervall auf 500 ms ein.



15. Schreiben Sie eine Unterbrechungsbehandlungsroutine, die Zeichen auf eine LCD-Anzeige ausgibt. Die Zeichen kommen von der seriellen Schnittstelle. Jedes ankommende Zeichen löst die zu schreibende Behandlungsroutine aus. Es befindet sich im Register `UDR` der seriellen Schnittstelle. Zur Ausgabe auf die Anzeige ist ein fertiges Unterprogramm `DISPLAY` zu verwenden. Es erhält das darzustellende Zeichen im Register `r16`. Das Unterprogramm benötigt zudem die Register `r18`, `r19`, `r24` und `r25`.
16. Modifizieren Sie die Unterbrechungsbehandlungsroutine von Aufgabe 15 so, daß jedes ankommende Zeichen als zweistellige Hexadezimalzahl dargestellt wird (zuerst die höheren, dann die niederen vier Bits)¹. Geben Sie nach diesen beiden Zeichen ein Leerzeichen aus (ASCII 20H).

Hinweis: Die ASCII-Codes der Ziffern 0...9: 30H...39H, der Zeichen A...F: 41H...46H.

17. Eine 16-Bit-Zahl in den Registern `r1` und `r2` ist um drei Bits nach links zu verschieben. Die frei werdenden Bitpositionen sind mit Nullen aufzufüllen.

r1	Bits 7...0
r2	Bits 15...8

¹ Geheimtip: Achten Sie darauf, welche Register Ihre HEX-Wandlung braucht ...

18. Eine 16-Bit-Zahl in den Registern r20 und r21 ist um zwei Bits arithmetisch nach rechts zu verschieben.

r20	Bits 7...0
r21	Bits 15...8

19. Ein Tastenfeld ist an Port B angeschlossen. Es soll abgefragt werden, ob die Taste an Bitposition 3 betätigt ist oder nicht (Tastenwirkung: aktiv low). Das Programmstück:

```
WAIT_3:
        SBIC     PORTB,3
        RJMP    WAIT_3
```

Wird das so funktionieren? Geben Sie ggf. eine korrigierte Befehlsfolge an.

20. Ihr Programm enthält diese Befehlsfolge:

```
        CP      TEMP, LIMIT
        BRNE    BEGIN
        IN      TEMP, PIND
```

Bisher lief alles. Nachdem jedoch an verschiedenen (anderen) Stellen im Programm geändert wurde, wird beim Assemblieren der BRNE-Befehl als fehlerhaft gekennzeichnet:

```
xyz.asm(80): error: Relative branch out of reach
```

- Weshalb?
- Wie helfen Sie sich?

A7: Eingabe (Taste),
 A6...0: Ausgabe (Einer)
 D7: Eingabe
 D6...0: Ausgabe (Zehner)
 Ports B und C: Eingabe;
 Pull-up's ein

