Assembler für BASIC – Programmierer

Von jean-claude.feltes@education.lu

Wenn es mal schnell gehen muss: ASM und BAS mischen in BASCOM

1. Warum ?

Gibt es gute Gründe, sich mit Assembler herumzuplagen, wenn alles so schön einfach mit BASCOM geht? Ja, es gibt deren mindestens zwei:

1. Man versteht besser, was der Mikrokontroller eigentlich macht.
   (Nun ja, so eine Idee kommt bestimmt von einem Lehrer :-))

2. Wenn es zeitlich wirklich eng wird, dann geht es nicht anders.
   Dies kommt vor, wenn viele Dinge „gleichzeitig“ zu erledigen sind oder wenn z.B. bei Messungen von Impulslängen und dergleichen möglichst wenig Zeit verloren gehen darf, damit das Ergebnis genau ist.

2. Wie ?

Um ASM zu lernen gibt es mindest zwei Methoden:
Man lernt mithilfe eines ASM-Tutorials anhand von kleinen Beispielen, und / oder man sieht sich an, was BASCOM als Code generiert. Leider geht das nur durch Disassemblieren der .HEX-Datei, da BASCOM (noch) kein ASM-Listing liefert.

Einen hervorragenden Disassemler von Chris Tarnovsky (leider als DOS-Programm) gibt es hier:

http://www.ericengler.com/downloads/disavr.zip

Um diesen leichter benutzen zu können habe ich auf die Schnelle ein Frontend dafür geschrieben. Beides zusammen findet sich hier:

http://staff.ltam.lu/feljc/electronics/bascom/disasm.zip

Ansonsten kann man auch AVR Studio benutzen. Dieses zeigt die Interruptvektoren am Beginn schöner, die Spezialregister aber nicht.

Die folgenden ASM-Listings wurden mit disavr erzeugt, die Kommentare rechts wurden manuell ergänzt. Hierbei ist eine Liste der Befehle mit Erklärungen und Angabe von benötigten Taktzyklen hilfreich, entweder aus dem Datenblatt oder [1].

Alle Beispiele wurden mit einem Mega8 durchgeführt.

3. Analyse von BASIC-Programmen

3.1 Notwendiges „Gerümpel“

Schon wenn ein Programm nichts tut, ist das ASM-Listing nicht leer:

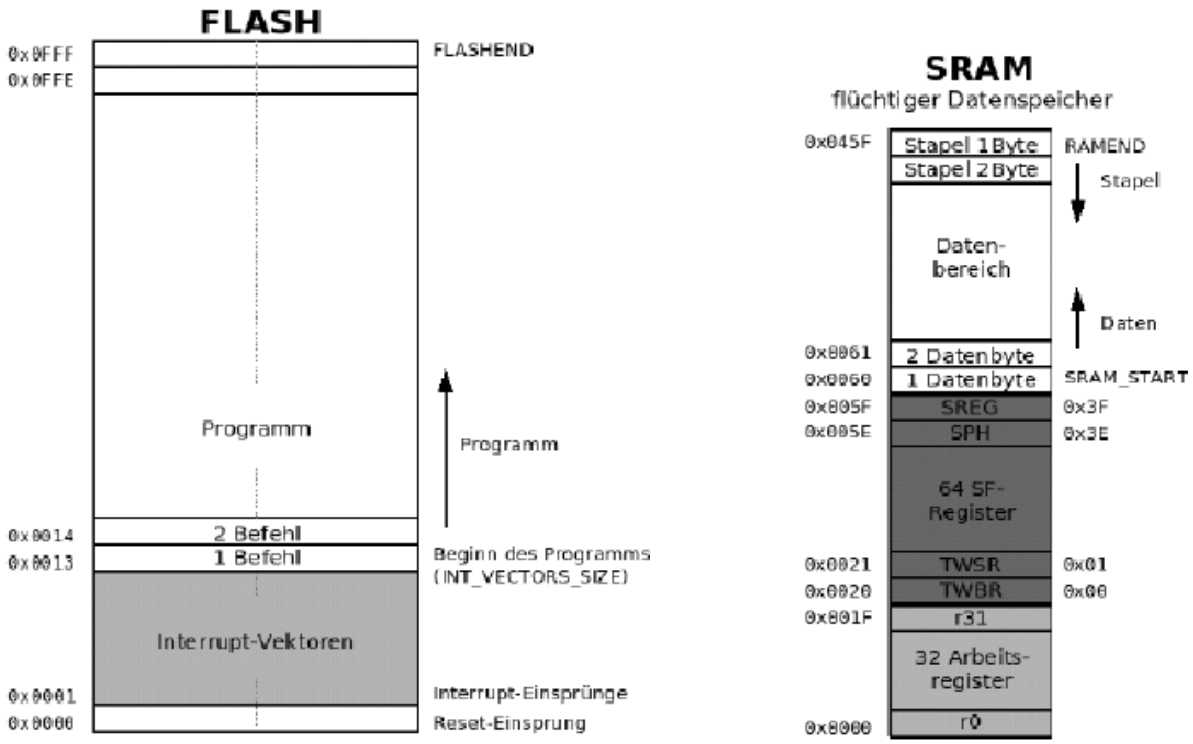
```
$regfile = "m8def.dat"
End
```

```
        ; Atmel AVR Disassembler v1.30
        ;
         .cseg
         .org     0

          rjmp    avr0013       ; 0000 C012
          reti                  ; 0001 9518       ;Interruptvektoren
          reti                  ; 0002 9518
          .........
          reti                  ; 0011 9518
          reti                  ; 0012 9518

;Beginn „Hauptprogramm"

avr0013:
      ;Stackpointer setzen u.a.
          ldi     r24, 0x5F     ; 0013 E58F
          out     SPL, r24      ; 0014 BF8D       ;SPL = 5Fh
          ldi     YL, 0x40      ; 0015 E4C0       ;YL = 40h

          ldi     ZL, 0x38      ; 0016 E3E8
          mov     r4, ZL        ; 0017 2E4E       ;R4 = 38h

          ldi     r24, 0x04     ; 0018 E084
          out     SPH, r24      ; 0019 BF8E       ;SPH = 4

          ldi     YH, 0x04      ; 001A E0D4       ;YH = 4

          ldi     ZH, 0x04      ; 001B E0F4
          mov     r5, ZH        ; 001C 2E5F       ;R5 = 4

      ;SRAM löschen von 60h aufwärts bis 60h + 3FFh
          ldi     ZL, 0xFE      ; 001D EFEE
          ldi     ZH, 0x03      ; 001E E0F3       ;Z = 3FE

          ldi     XL, 0x60      ; 001F E6A0
          ldi     XH, 0x00      ; 0020 E0B0       ;X = 60h

          clr     r24           ; 0021 2788       ;R24=0

avr0022:  st      X+, r24       ; 0022 938D       ;R24=0 an Adresse X = 060h, X inc

          sbiw    ZL, 0x01      ; 0023 9731       ;Z = Z-1

           brne   avr0022       ; 0024 F7E9       ;GOTO 0022 bis Z = 0

          clr     r6            ; 0025 2466       ;R6=0
          cli                   ; 0026 94F8       ;Clear interrupts

avr0027:  rjmp    avr0027       ; 0027 CFFF       ;Endlosschleife (Hauptprogramm)

avr0028:  sbiw    ZL, 0x01      ; 0028 9731
           brne   avr0028       ; 0029 F7F1
          ret                   ; 002A 9508
          set                   ; 002B 9468
          bld     r6, 2         ; 002C F862
          ret                   ; 002D 9508
          clt                   ; 002E 94E8
          bld     r6, 2         ; 002F F862
          ret                   ; 0030 9508
```

Der erste Befehl ist ein Sprung zum „Hauptprogramm“ avr0027, welches hier nur aus einer leeren Endlosschleife besteht.

Es folgen die Interruptvektoren an den vorgesehenen festen Speicheradressen. Da kein Interrupt benutzt wird, steht dort nur ein reti (Return from Interrupt).

Es folgt das Setzen des Stackpointers auf 45Fh.
Dazwischen und danach wird noch Y = 440h und R5:R4 = 438h gesetzt (warum?)
Anschliessend wird das SRAM von 60h aufwärts gelöscht bis 60h + 3FEh +1 = 45Eh, also bis gerade unterhalb des Stackpointers. Dies stimmt mit der Speicherorganisation überein (Datenblatt):



Ein Blick in die BASCOM-Hilfe enthüllt noch einiges:
R6 wird gelöscht da es als Speicher für einige interne Bit-Variablen benutzt wird:

> R6 is used to store some bit variables:
> R6 bit 0 = flag for integer/word conversion
> R6 bit 1 = temp bit space used for swapping bits
> R6 bit 2 = error bit (ERR variable)
> R6 bit 3 = show/noshow flag when using INPUT statement

R5:R4 = 438h ist einZeiger auf den Stack Frame, in dem temporäre Daten abgespeichert werden.
R9:R8 (hier nicht benutzt) werden vom READ-Befehl gebraucht.

Kompilationsinformationen von BASCOM:

```
Stack start  : 45F hex
Stack size   : 20 hex
S-Stacksize  : 8 hex
S-Stackstart : 440 hex
Framesize    : 18 hex
Framestart   : 427 hex
Space left   :  961  dec
------------------------------------------------------------------------------
HWSTACK                         Word            005D         93
SWSTACK                         Word            001C         28
FRAME                           Word            0004         4
ERR                             Bit             0006         6

------------------------------------------------------------------------------
```

```
Constant                              Value
-------------------------------------------------------------------------------
SREG                                  &H3F
SPH                                   &H3E
SPL                                   &H3D
```

3.2 Einfaches Beispiel: Blink-Programm

```
$regfile = "m8def.dat"
$crystal = 4000000
Config Pind.5 = Output


Do
    Portd.5 = 1
    Waitms 200
    Portd.5 = 0
    Waitms 100
Loop
```

```
........    (Prolog wie oben)

            sbi     DDRD, 5       ; 0026 9A8D           ; PortD.5 = Ausgang

;Hauptschleife
avr0027:  sbi     PORTD, 5      ; 0027 9A95           ;D.5 = 1

     ;Waitms 200
          ldi     r24, 0xC8     ; 0028 EC88
          ldi     r25, 0x00     ; 0029 E090           ;r25:r24 = 200dez
          rcall   avr0039       ; 002A D00E

     ;Waitms 100
          cbi     PORTD, 5      ; 002B 9895
          ldi     r24, 0x64     ; 002C E684           ;r25:r24 = 100dez
          ldi     r25, 0x00     ; 002D E090
          rcall   avr0039       ; 002E D00A

          rjmp    avr0027       ; 002F CFF7
;Ende Hauptschleife

avr0030:  sbiw    ZL, 0x01      ; 0030 9731
           brne   avr0030       ; 0031 F7F1
          ret                   ; 0032 9508
          set                   ; 0033 9468
          bld     r6, 2         ; 0034 F862
          ret                   ; 0035 9508
          clt                   ; 0036 94E8
          bld     r6, 2         ; 0037 F862
          ret                   ; 0038 9508
;Prozedur Waitms
avr0039:  push    ZL            ; 0039 93EF       ;Z retten
          push    ZH            ; 003A 93FF

    ;sofort raus wenn Wert 0
          clr     ZL            ; 003B 27EE       ;Prüfen ob R24 oder R25 = 0
          or      ZL, r24       ; 003C 2BE8
          or      ZL, r25       ; 003D 2BE9
           breq   avr0045       ; 003E F031       ;wenn ja, tschüs

    ;Beginn Zeitschleife
avr003F:  ldi     ZL, 0xE8      ; 003F EEE8       ;Z = 1000
          ldi     ZH, 0x03      ; 0040 E0F3
avr0041:  sbiw    ZL, 0x01      ; 0041 9731       ;Z = Z-1
```

```
           brne   avr0041       ; 0042 F7F1       ;solange bis Z = 0
           sbiw   r24, 0x01     ; 0043 9701       ;R25:R24 = R25:R24 - 1
           brne   avr003F       ; 0044 F7D1

      ;fertig
avr0045:   pop    ZH            ; 0045 91FF       ;Z wiederherstellen
           pop    ZL            ; 0046 91EF
           ret                  ; 0047 9508
          .exit
```

Nach dem üblichen Prolog mit Stackpointer in Ordnung bringen und RAM löschen ist die Hauptschleife leicht zu identifizieren. In dieser werden die Portpins geschaltet mit sbi und cbi.

Dazwischen wird die Waitms-Prozedur aufgerufen, welcher in R25:R24 die Wartezeit in ms übergeben wird.

Erkennbar ist auch, dass es beim Setzen einzelner Portpins egal ist, ob man ASM oder BAS beutzt, da die Setz- und Rücksetzbefehle mit sbi bzw. Cbi optimal übersetzt werden.

Kern der Waitms-Prozedur:

```
    ;Beginn Zeitschleife
avr003F:   ldi    ZL, 0xE8      ; 003F EEE8       ;Z = 1000
           ldi    ZH, 0x03      ; 0040 E0F3

      ;1000 x schleifen
avr0041:   sbiw   ZL, 0x01      ; 0041 9731       ;Z = Z-1
           brne   avr0041       ; 0042 F7F1       ;solange bis Z = 0

           sbiw   r24, 0x01     ; 0043 9701       ;R25:R24 = R25:R24 - 1
           brne avr003F         ; 0044 F7D1
```

Das Z-Register wird mit dem Wert Z=1000 geladen, dann wird die innere Schleife avr0041 R25:R24 mal aufgerufen.Die innere Schleife müsste also eine Dauer von 1 Mikrosekunde haben.
Sie enthält die Befehle sbiw und brne, welche beide je 2 Taktzyklen dauern (bei brne nur für den Zustand „non equal = non zero, sonst 1 Takt).
Insgesamt sind das 4 Takte, was bei einer Taktfrequenz von 4MHz 1 µs bedeutet.

4. Optimierung von zeitkritischen Teilen in Assembler

Beispiel: Rechteckgenerator mit Timer
Mit Timer0 soll ein Rechtecksignal mit T = 200µs erzeugt werden.
In BASIC:

```
'8-bit-Timer als Rechteckgenerator T = 200us
'Timer-Intervall: 100us
'Prescaler 8 -> TCLK = 2us -> 50 Zyklen
'Preload: 256-50=206


$crystal = 4000000
$regfile = "m8def.dat"

Config Portd.5 = Output

On Timer0 Timer0serv

Config Timer0 = Timer , Prescale = 8    'Timer-CLK = 500kHz -> 2us pro Zyklus
Enable Timer0
```

```
Enable Interrupts

Const Preload = 206                     '256-254 = 2 Zyklen: 2*2=4us
Timer0 = Preload

'------------------
Do
  'tu nix
  nop
Loop
'------------------
Timer0serv:
    Timer0 = Preload
    Portd.5 = Not Portd.5
Return
```

Der Code funktioniert, aber die Periodendauer ist mit 228µs zu lang.
Dies liegt daran, dass beim Aufruf der Interruptroutine standardmässig alle Register gerettet werden, wie in der BASCOM-Hilfe beschrieben, und wie der Disassembler zeigt.

Ein Blick auf den Beginn des Listings zeigt, dass nun der Interruptvektor des Timer0 aktiviert worden ist:

```
        ; Atmel AVR Disassembler v1.30
        ;
         .cseg
         .org     0

          rjmp   avr0013      ; 0000 C012
          reti                ; 0001 9518
          reti                ; 0002 9518
          reti                ; 0003 9518
          reti                ; 0004 9518
          reti                ; 0005 9518
          reti                ; 0006 9518
          reti                ; 0007 9518
          reti                ; 0008 9518
          rjmp   avr0031      ; 0009 C027
          reti                ; 000A 9518
          reti                ; 000B 9518
...........
```

Die Adresse des Interrupt-Handlers ist also avr0031, dort finden wir

```
avr0031:  push   r0           ; 0031 920F      ;R0 bis R31 pushen
          push   r1           ; 0032 921F
          ......
          push   r25          ; 0043 939F
          push   XL           ; 0044 93AF
          push   XH           ; 0045 93BF
          push   YL           ; 0046 93CF
          push   YH           ; 0047 93DF
          push   ZL           ; 0048 93EF
          push   ZH           ; 0049 93FF

          in     r24, SREG    ; 004A B78F      ;SEG über R24 pushen
          push   r24          ; 004B 938F

      ;eigentliche Interrupt-Routine

      ;Preload-Wert setzen
          ldi    r24, 0xCE    ; 004C EC8E
          out    TCNT0, r24   ; 004D BF82      ;TCNT0 = Ceh = 206 (Preload)
```

```
    ;Portpin invertieren
        clr     XH          ; 004E 27BB
        ldi     XL, 0x32    ; 004F E3A2     ;X = 0032h
        ld      r24, X      ; 0050 918C     ;Word an Adresse X=32h nach R24
        bst     r24, 5      ; 0051 FB85     ;Bit 5 von R24 ins T-Flag im SREG
        clr     r24         ; 0052 2788     ;R24 = 0
        bld     r24, 0      ; 0053 F980     ;T-Flag in R24, bit0 kopieren
        com     r24         ; 0054 9580     ;R24 = NOT R24
        bst     r24, 0      ; 0055 FB80     ;R24, bit 0 nach T-Flag kopieren
        clr     XH          ; 0056 27BB
        ldi     XL, 0x32    ; 0057 E3A2     ;X = 0032h
        ld      r24, X      ; 0058 918C     ;Word an Adresse X=32h nach R24
        bld     r24, 5      ; 0059 F985     ;T-Flag nach R24 kopieren
        st      X, r24      ; 005A 938C     ;R24 nach Adresse X = 32h kopieren

        pop     r24         ; 005B 918F     ;SREG über R24 wiederherstellen
        out     SREG, r24   ; 005C BF8F

        pop     ZH          ; 005D 91FF     ;R31...R0 poppen
        pop     ZL          ; 005E 91EF
        pop     YH          ; 005F 91DF
        pop     YL          ; 0060 91CF
        pop     XH          ; 0061 91BF
        pop     XL          ; 0062 91AF
        pop     r25         ; 0063 919F
        .........
        pop     r0          ; 0075 900F
        reti                ; 0076 9518
```

BASCOM geht auf Nummer Sicher, indem es alle Register rettet.
Die 32 push-Befehle kosten 32 mal 2 Taktzyklen, dazu noch 3 (in + push) für das Retten von SREG, das sind 67 Takte, also 16.75µs, bevor überhaupt etwas anderes geschehen kann.
Das Gleiche gilt natürlich für die pop-Befehle zum Schluss.
Wir haben also ca. 33µs Zeit verplempert, im der nichts Vernünftiges geschehen kann.

Deswegen funktioniert das obige Beispiel auch nicht mehr, wenn wir den Preload-Wert auf 251 setzen, womit sich eine Periodendauer von 20µs ergeben müsste. Die gemessene Periodendauer liegt mit 102µs total daneben.

Wenn wir uns den Code ansehen, können wir uns mit der Nosave-Option selbst um die Rettung der Register kümmern, wir brauchen nur die Register zu retten, die in der Interruptroutine verwendet werden.
Dies wären hier SREG, R24 und X (=R27:R26).

Geänderter BAS-Code:

```
.......
On Timer0 Timer0serv Nosave

.......

Timer0serv:
    push R24
    push XL
    push XH

    in R24,SREG
    push R24

    Timer0 = Preload
    Portd.5 = Not Portd.5
```

```
    pop R24
    !Out Sreg , R24

    pop XH
    pop XL
    pop R24
Return
```

BASCOM versteht Assembler-Befehle, nur bei Verwechslungsgefahr mit BAS-Befehlen muss ihm durch Vorsetzen eines „!“ der Unterschied klargemacht werden, wie hier beim Out-Befehl.

Längere ASM-Passagen kann man mit „$ASM“.....“$END ASM“ einfügen ohne „!“ zu benutzen.

Eine Messung zeigt, dass wir mit 208µs statt vorher 228µs schon viel besser abschneiden.
Wir haben gegenüber den geforderten 200µs aber immer noch 8µs zuviel.

Also nochmal einen Blick auf das Listing werfen!
Beim Laden des Preload-Wertes können wir nichts verbessern.
Allerdings erscheint die Invertierung des Portpins einigermassen umständlich realisiert zu sein.

Durch EXOR mit 1 kann Bit 5 von PortD schneller invertiert werden. Ausserdem müssen dann nur 2 Register zusätzlich zu SREG gerettet werden.:

```
Timer0serv:
    push R24                            'Register retten
    push R25

    in R24,SREG
    push R24

    Timer0 = Preload                    'Preload geht in ASM nicht schneller

    in R24,portd                        'PortD invertieren durch
    ldi R25,32                          'EXOR mit 32 = 0010 0000 (Bit 5 EXOR)
    eor R24,R25
    !Out Portd , R24


    pop R24                             'Register wiederherstellen
    !Out Sreg , R24

    pop R25
    pop R24
Return
```

Eine Messung zeigt 206µs, hiermit ein leicht besseres Ergebnis.

Der Hauptvorteil der letzten Lösung ist aber, dass die Interruptroutine insgesamt weniger Zeit braucht, dem Hauptprogramm also mehr Zeit für andere Aufgaben zur Verfügung steht.

Im Simulator lassen sich die 3 Varianten schön vergleichen.

	Zyklen bis hier	t/µs	Zyklen bis Port gesetzt	Gesamtdauer in Zyklen

Timer0serv:
  Timer0 = Preload
  Portd.5 = Not Portd.5
Return
	0
	53
	55
	71
		0
		13.25
		13.75
		17.75
			
			71
			(17.75µs)
				
				71+67+1
				= 139
				(34.75µs)

Timer0serv:
  push R24
  push XL
  push XH
 	
  in R24,SREG
  push R24
 	
  Timer0 = Preload
  Portd.5 = Not Portd.5
 	
  pop R24
  !Out Sreg , R24
 	
  pop XH
  pop XL
  pop R24
Return
	0
	
	6
	
	9
	11
	
	27
	
	30
	
	36
		
		1.5
		
		2.25
		2.75
		
		6.75
		
		7.5
		
		9
			
			27
			(6.75µs)
				
				36 + 1
				= 37
				(9.25µs)

Timer0serv:
  push R24
  push R25
 	
  in R24,SREG
  push R24
 	
  Timer0 = Preload
 	
  in R24,portd
  ldi R25,32
  eor R24,R25
  !Out Portd , R24
 	
  pop R24
  !Out Sreg , R24
 	
  pop R25
  pop R24
 	
Return
	0
	
	4
	
	7
	
	9
	
	12
	
	13
	
	16
	
	20
		
		1
		
		1.75
		
		2.25
		
		3
		
		3.25
		
		4
		
		5
			
			16
			(4µs)
				
				20 + 1
				= 21
				(5.25µs)

Sieht man sich die Ergebnisse an, so sieht man, dass die Varianten 2 und 3 sowohl in der Schnelligkeit der Interruptauslösung als auch in der Zeit, die dem Programm sonst noch zur Verfügung steht, wesentlich besser abschneiden als die reine BASIC-Variante.

Was man jetzt noch tun kann ist, die Preload-Konstante anpassen, um die verlorene Zeit zu kompensieren. Das wären in der dritten Variante 16 Takte, also beim 4MHz-Quarz 16*0.25µs = 4µs. Bei einem Prescaler von 8 ist die Timerperiode 0.25µs·8 = 2µs, wir müssen also Zyklen weniger zählen.
Also wird Preload = 206 + 2 gewählt.

Damit ergibt sich dann eine Periodendauer von 198µs statt 200µs, wir haben also pro Interrupt 1µs zu wenig.

Man sieht, dass sich durch die Verwendung von Assembler einiges verbessern lässt.

Bei der Verwendung von Interrupten muss man sich aber bewusst sein, dass es kein messerscharfes Timing geben kann. Laut [1] dauert es mindestens 4 Systemtakte (hier 1µs) bis wir in der Interruptroutine landen. Der Controller muss ja die Rücksprungadresse (2 Bytes) auf dem Steck speichern, den Stackpointer um 2 dekrementieren und einen relativen Sprung in die Interruptroutine tun. Wenn gerade ein Befehl ausgeführt wurde, der mehrere Takte dauert, wird dieser noch zuerst ganz ausgeführt. Dies erklärt, dass die im Simulator gemessenen Zeiten manchmal um Bruchteile von µs vaiieren können, je nachdem womit das Hauptprogramm gerade beschäftigt ist.

Wenn wir gerade vom Hauptprogramm reden: man sollte die Interruptroutine (sobald sie funktioniert) einmal mit ein paar einfachen Befehlen wie X=X+1: PRINT X in der Hauptschleife testen, um sicher zu sein, dass dafür genügend Zeit bleibt, und dass nirgends ein push oder pop vergessen wurde oder eines zuviel ist.

Literatur:

[1] W. Trampert
AVR-RISC Mikrokontroller
Franzis

[2] R. Walter
AVR Mikrocontroller Lehrbuch
PS Verlag

[3] C. Kühnel
Programmieren der AVR RISC Mikrocontroller
Libri Books on demand

[4] G.Weiler
Einführung MICEL
http://www.weigu.lu/a/avr.html

ANHANG

Wichtige ASM-Befehle (Beispiele):

Befehl	*Bedeutung*	*Wirkung*	*Takt-zyklen*
nop	No operation	nix	1
Portbefehle			
sbi portd, 5	Set Bit immediate	D.5 = 1	2
cbi portd.5	Clear bit immediate	D.5 = 0	2
out PortB, R24	out register to port	PortB = R24	1
in R24, PortB	in port to register	R24 = PinB	1
Registerbefehle			
clr R24	Clear register	R24 = 0	1
ldi R24, 56	Load immediate	R24 = 56	1
mov R24, R25	Move (copy register)	R24 = r25	1
push R24	R24 retten	R24 → Stack	2
pop R24	R24 wiederherstellen	Stack → R24	2
Sprungbefehle			
rjmp loop	Relative jump	Springe nach „loop“	2
bre loop	Branch if equal	Springe nach „loop“ wenn gleich	1,2 *)
brne loop	Branch if not equal	Springe nach „loop“ wenn ungleich	1,2 *)
sbic PinD.4	Skip if IO bit cleared	Überspringe nächste Zeile wenn D.4=0	1,2,3 *)
sbis PinD.4	Skip if IO bit set	Überspringe nächste Zeile wenn D.4=1	1,2,3 *)
rcall sub1	Call subprogram	Call subprogram	3
ret	Return	Return	4
reti	Return from Interrupt	Return from Interrupt	4
Interrupts			
cli	Clear Global Interrupt Flag	Interrupts aus	1
sei	Set global Interrupt Flag	Interrupts ein	1
Arithmetik			
dec R24	Decrement	R24 = R24 – 1	1
inc R24	Increment	R24 = R24 + 1	1
Logik			1
and R24, R25	Logical AND	R24 = R24 AND R25	1
andi R24, 8	Logical AND immediate	R24 = R24 AND 8	1
or R24, R25	Logical OR	R24 = R24 OR R25	1
ori R24, 8	Logical OR immediate	R24 = R24 OR 8	1
eor R24, R25	Logical EXOR	R24 = R24 XOR R25	1

*) je nach Fall

Wissenswertes (ASM)

- Quelltext als ASM markieren (nicht immer nötig):
  $asm
  .....ASM - Code
  $end asm

  Normalerweise erkennt der Compiler ASM-Befehle.

- Befehle die verwechselt werden können wie OUT müssen ein vorgestelltes „!“ erhalten
- Achtung: ASM-Labels müssen ganz links und allein in einer Zeile stehen.
- Arbeitsregister R0...R31 werden mit MOV geladen, für die Special Function Register werden OUT und IN verwendet
- Das Statusregister SREG enthält die Flags. Es muss beim Aufruf einer Interruptroutine immer gerettet werden, ausser wenn die Flags sicher nicht beeinflusst werden.
  Dies geht z.B. mit:

```
in R24,SREG
push R24
```

- SREG kann nicht direkt gepusht werden
- OUT muss über ein Zwischenregister geschehen, direkte Ausgabe geht nicht.
  Zum Beispiel:

```
ldi R24, 6
!Out Sreg , R24
```

- Konditionelle Bitabfrage (Beispiel):

```
$crystal = 4000000
$regfile = "m8def.dat"

Config Portc = Input
Config Portd = Output
Portc = &HFF                              'interne Pullup

$asm
Loop:
      sbic Pinc,5     ;Überspringen, wenn C.5=0
      sbi portd,5      ;D.5=1   wenn C.5=1
      sbis Pinc,5     ;Überspringen wenn C.5=1
      cbi Portd,5      ;D.5=0   wenn C.5=0
      rjmp loop
$end Asm
```

Wichtige Register

Statusregister SREG

Bits	7	6	5	4	3	2	1	0
	I	T	H	S	V	N	Z	C

- I = Global Interrupt Enable
  (1 = Interrupts aktiviert, müssen aber noch einzeln freigeschaltet werden.)
  Kann mit SEI gesetzt und mit CLI rückgesetzt werden.

- T = Bit copy storage
  Temporäres Register für Bits
  Einzelne Bits eines Registers können mit BST nach T kopiert und mit BLD in ein Register geschrieben werden.

- H = Half Carry Flag
  nützlich bei BCD-Operationen.

- S = Sign Bit
  S = N ⊕ V

- V = Zweierkomplement Overflow Flag

- N = Negative Flag

- Z = Zero Flag

- C = Carry Flag

Timer-Register (Mega8)

Gemeinsame Register für alle Timer:

TIMSK Timer/Counter Interrupt Mask Register
schaltet für die 3 Timer gemeinsam die Timer-Interrupts ein und aus (1 = Interrupt eingeschaltet)

Bits	7	6	5	4	3	2	1	0
	OCIE2	TOIE2	TCIE1	OCIE1A	OCIE1B	TOIE1	-	TOIE0

Timer0:
TOIE = Timer0 Overflow Output Enable

Timer1:
TICIE1 = Timer/Counter1 Input Capture Interrupt Enable
OCIE1A = Timer/Counter1 Output Compare A Match Interrupt Enable
OCIE1B = Timer/Counter1 Output Compare B Match Interrupt Enable
TOIE1 = Timer/Counter1 Overflow Interrupt Enable

Timer2:
OCIE2 = Timer/Counter2 Output Compare Match Interrupt Enable
TOIE2 = Timer/Counter2 Overflow Interrupt Enable

TIFR Timer/Counter Interrupt Flag Register

Bits	7	6	5	4	3	2	1	0
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	-	TOV0

Timer0:
TOV0 = Timer0 Overflow Flag

Timer1:
TOV1 = Timer/Counter1 Overflow Flag
OCF1A = Timer/Counter1 Output Compare A Match Flag
OCF1B = Timer/Counter1 Output Compare B Match Flag
ICF1 = Timer/Counter1 Input Capture Flag

Timer2:
TOV2 = Timer/Counter2 Overflow Flag
OCF2 = Timer/Counter2 Output Compare Flag

Timer0

TCCR0 Timer/Counter 0 Control Register

legt den Prescaler-Faktor fest, und schaltet den Timer ein und aus

Bits	7	6	5	4	3	2	1	0
						CS02	CS01	CS00

CS02	CS01	CS00	Prescaler
0	0	0	Timer gestoppt
0	0	1	1
0	1	0	8
0	1	1	64
1	0	0	256
1	0	1	1024
1	1	0	Ext. Takt an T0, fallende Flanke
1	1	1	Ext. Takt an T0, ansteigende Flanke

TCNT0 Timer/Counter Register
Zählerregister 8 bit

Bits	7	6	5	4	3	2	1	0

Timer1

TCCR1 Timer/Counter 1 Control Register

TCCR1A Timer/Counter 1 Control Register A:

Bits	7	6	5	4	3	2	1	0
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10

TCCR1B Timer/Counter 1 Control Register B:

Bits	7	6	5	4	3	2	1	0
	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10

COM1: Compare Output Mode for channel A/B
stellen das Verhalten der Pins OC1A, OC1B ein.
Eine 1 bewirkt, dass das normale Verhalten des Pins aufgehoben wird und ein Ausgangssignal erzeugt wird, wenn der Port als Ausgang geschaltet ist (Bit im DDR gesetzt).

Verhalten der OC-Pins (ausser PWM-Modus):

COM1x**1**	COM1x**0**	
0	0	OC1-Pins funktionieren normal (disconnected)
0	1	Toggle on compare match
1	0	Clear on compare match
1	1	Set on compare match

mit x = A/B

Verhalten der OC-Pins im Fast PWM-Modus:

COM1x**1**	COM1x**0**	
0	0	OC1-Pin funktioniert normal (disconnected)
0	1	Wenn WGM13:0 = 15: Toggle OC1A bei Compare Match, OC1B disconnected ansonsten: OC1-Pins funktionieren normal (disconnected)
1	0	Clear on compare match Set at TOP
1	1	Set on compare match Clear at TOP

mit x = A/B

Verhalten der OC-Pins im normalen PWM-Modus:

COM1x**1**	COM1x**0**	
0	0	OC1-Pin funktioniert normal (disconnected)
0	1	Wenn WGM13:0 = 9 oder 14: Toggle OC1A bei Compare Match, OC1B disconnected ansonsten: OC1-Pins funktionieren normal (disconnected)
1	0	Clear on compare match beim Aufwärtszählen Set on compare match beim Abwärtszählen
1	1	Set on compare match beim Aufwärtszählen Clear on compare match beim Abwärtszählen

mit x = A/B

FOC1: Force Output Compare for channel A/B
(nur schreiben, gelesen wird immer 0)

WGM13....WGM10 Waveform Generation Mode
stellen PWM-Modus ein

0000 = PWM aus
0001 = PWM 8 bit
0010 = PWM 9 bit
0011 = PWM 10 bit
weitere siehe Datenblatt

ICNC1: Input Capture Noise Canceller
Filterung des ICP1-Pins

ICES1: Input Capture Edge Select
Triggermodus für Input Capture
0 = fallende Flanke
1 = ansteigende Flanke

CS12, CS11, CS10: Clock Select
wie bei Timer0

TCNT1 Timer/Counter Register
Zählerregister 16 bit

Bits	7	6	5	4	3	2	1	0	
									TCNT1H
									TCNT1L

OCR1A, OCR1B Output Compare Register A und B
Vergleichsregister 16 bit,wird mit TCNT1 verglichen.
Bei Übereinstimmung kann ein Interrupt ausgelöst, oder der Zustand der OC1-Pins verändert werden.

Bits	7	6	5	4	3	2	1	0	
									OCR1H
									OCR1L

ICR1 Input Capture Register
Wenn ein Capture-Ereignis am ICP1-Pin auftritt (oder optional am Komparator-Ausgang), wird der Inhalt des Timer-Registers TCNT1 ins Captureregister ICR1 geschrieben (16 bit)

Bits	7	6	5	4	3	2	1	0	
									OCR1H
									OCR1L

Timer2

TCCR2 Timer/Counter 2 Control Register

Bits	7	6	5	4	3	2	1	0
	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20

FOC2: Force Output Compare
(nur schreiben, gelesen wird immer 0)

WGM21, WGM20 Waveform Generation Mode
stellen PWM-Modus ein

WGM21	WGM20	
0	0	Normal
0	1	PWM phasenkorrekt
1	0	Clear Timer on Compare Match
1	1	Fast PWM

COM21, COM20: Compare Output Mode for channel A/B
stellen das Verhalten des Pins OC2 ein.
Eine 1 bewirkt, dass das normale Verhalten des Pins aufgehoben wird und ein Ausgangssignal erzeugt wird, wenn der Port als Ausgang geschaltet ist (Bit im DDR gesetzt).

Verhalten der OC-Pins (ausser PWM-Modus):

COM21	COM20	
0	0	OC2-Pin funktioniert normal (disconnected)
0	1	Toggle OC2 on compare match
1	0	Clear OC2 on compare match
1	1	Set OC2 on compare match

Verhalten der OC-Pins im normalen (phasenkorrekten) PWM-Modus:

COM21	COM20	
0	0	OC2-Pin funktioniert normal (disconnected)
0	1	-
1	0	Clear on compare match Set at TOP
1	1	Set on compare match Clear at TOP

Verhalten der OC-Pins im normalen PWM-Modus:

COM21	COM20	
0	0	OC2-Pin funktioniert normal (disconnected)
0	1	-
1	0	Clear on compare match when upcounting Set on compare match when downcounting
1	1	Set on compare match when upcounting Clear on compare match when downcounting

CS22, CS21, CS20: Clock Select

Achtung: ähnlich wie bei Timer0, aber z.T. andere Teilverhältnisse!

CS22	CS21	CS20	Prescaler
0	0	0	Timer gestoppt
0	0	1	1
0	1	0	8
0	1	1	**32**
1	0	0	**64**
1	0	1	**128**
1	1	0	**256**
1	1	1	**1024**

TCNT2 Timer/Counter Register
Zählerregister 8 bit

Bits	7	6	5	4	3	2	1	0

OCR2 Output Compare Register
Vergleichsregister 8 bit,wird mit TCNT2 verglichen.
Bei Übereinstimmung kann ein Interrupt ausgelöst, oder der Zustand der OC2-Pins verändert werden.

Bits	7	6	5	4	3	2	1	0

ASSR Asynchronous Status Register
erlaubt den Betrieb mit internem Takt 1MHz für den Prozessor und Uhrenquarz 32768Hz für Timer2.

Bits	7	6	5	4	3	2	1	0
	-	-	-	-	AS2	TCN2UB	OCR2UB	TCR2UB

AS2 = 1: asynchrone Operation mitb Uhrenquarz
weitere Détails siehe Datenblatt.