www.mikrocontroller.net

AVR Assembler Makros

Hier entsteht eine Sammlung von verschiedenen nützlichen Makros für den AVR Assembler.

Inhaltsverzeichnis

16 Bit Konstante in Z-Pointer laden

```
.MACRO SetZPtr ;(Adresse)
       ldi     ZL, LOW(@0)
       ldi     ZH, HIGH(@0)
.ENDMACRO
```

Natürlich auch möglich mit X- und Y-Pointer.

Speicher

2 Register ohne Zwischenspeicher vertauschen

```
.MACRO SWAP ;(a, b)
        eor     @0, @1
        eor     @1, @0
        eor     @0, @1
.ENDMACRO
```

Arithmetik

Konstante addieren

```
.MACRO ADDI ;(a, k)
        subi    @0, -(@1)
.ENDMACRO
```

Konstante addieren (16 Bit)

```
.MACRO ADDIW ;(RdL:RdH, k)
        subi    @0L, LOW(-@1)
        sbci    @0H, HIGH(-@1)
.ENDMACRO
```

oder (sinnlos)

```
.MACRO ADDIW ;(Rd, k)
        sbiw    @0, (-@1)
.ENDMACRO
```

DAS geht auch ohne Makro

```
        adiw a, b
```

SBIW und ADIW sind aber beide auf die Register(paare) R24, R26, R28, R30 beschränkt UND nehmen nur Zahlen <64 an.

I/O

Bei grösseren und neueren AVRs sind etliche I/O-Register nicht mit IN/OUT-Befehlen ansprechbar. LDS/STS erreicht zwar alle, ist aber bei kleineren oder älteren ineffizient.

Port lesen

```
.macro input
  .if @1 < 0x40
        in      @0, @1
  .else
        lds     @0, @1
  .endif
.endm
```

Port schreiben

```
.macro output
  .if @0 < 0x40
        out     @0, @1
  .else
        sts     @0, @1
  .endif
.endm
```

Portbit abfragen

Abfrage eines Bits eines I/O-Ports und Sprung wenn 1/0. Überschreibt u.U. ZL.

Branch if Bit in I/O-Register is Set

```
.macro bbis ;port,bit,target
  .if @0 < 0x20
        sbic    @0, @1
        rjmp    @2
  .elif @0 < 0x40
        in      zl, @0
        sbrc    zl, @1
        rjmp    @2
  .else
        lds     zl, @0
        sbrc    zl, @1
        rjmp    @2
  .endif
.endm
```

Branch if Bit in I/O-Register is Cleared

```
.macro bbic ;port,bit,target
  .if @0 < 0x20
        sbis    @0, @1
        rjmp    @2
  .elif @0 < 0x40
        in      zl, @0
        sbrs    zl, @1
        rjmp    @2
  .else
        lds     zl, @0
        sbrs    zl, @1
        rjmp    @2
  .endif
.endm
```

Delay

Verzögerung um X Nanosekunden

von Klaus2m5

Taktgenaue Verzögerung der Instruktionsausführung durch Angabe der Verzögerungszeit in Nanosekunden. Dabei werden maximal 4 Instruktionen erzeugt. Taktgenau bedeutet, dass auf die nächste volle Anzahl von Takten aufgerundet wird. Beispiel: 75ns bei 20MHZ (50ns Taktzeit) bedeutet eine tatsächliche Verzögerung von 2 Zyklen und entspricht 100ns.

Die Variable Osc_Hz muss der verwendeten Taktquelle angepasst werden und definiert die CPU-Taktfrequenz in Hertz.

wait_ns wird mit folgenden Parametern aufgerufen:

```
1. Verzögerungszeit in Nanosekunden
2. bereits verbrauchte Takte
3. ein Immediate-Register (R16-R31) als Zähler
```

Bereits verbrauchte Takte werden aus den Instruktionen errechnet, die zwischen den zu verzögernden Ereignissen liegen. Beispiel:

```
          sbi     porta,0
          wait_ns 1000,2,R16
          cbi     porta,0
```

In diesem Fall besteht die Anzahl der verbrauchten Takte lediglich aus den Instruktionen, die zum Ereignis führen. Am Ende von SBI wird die steigende Flanke,

am Ende von CBI die fallende Flanke des Signals erzeugt. Wenn wir also möglichst exakt eine Pulsbreite von einer Mikrosekunde erzeugen wollen, müssen wir die Ausführungszeit von CBI von unserer Wartezeit abziehen. Die Ausführung von CBI liegt vor dem Ereignis!

Wenn die Verzögerungszeit kleiner als die Anzahl bereits verbrauchter Taktzyklen ist, wird keine weitere Verzögerung erzeugt. Die maximale Verzögerung ist 767 Takte entsprechend 38350ns bei 20MHZ. Bei niedrigeren Frequenzen wird eine längere Verzögerung erreicht, allerdings nimmt dann auch die Genauigkeit der Verzögerung ab (exakt bis +1 Takt).

```
;
; wait_ns  waittime in ns , cyles already used , waitcount register
;
;     cycles already used will be subtracted from the delay
;     the waittime resolution is 1 cycle (delay from exact to +1 cycle)
;     the maximum delay at 20MHz (50ns/clock) is 38350ns
;     waitcount register must specify an immediate register
;
.set     Osc_Hz         = 7372800                    ;7,3728 MHz (Baudrate xtal)
.set     cycle_time_ns  = (1000000000 / Osc_Hz)      ;clock duration
.macro   wait_ns
      .set cycles = ((@0 + cycle_time_ns - 1) / cycle_time_ns - @1)
      .if (cycles > (255 * 3 + 2))
        .error "MACRO wait_ns - too many cycles to burn"
      .else
        .if (cycles > 6)
          .set  loop_cycles = (cycles / 3)
          ldi   @2,loop_cycles
          dec   @2
          brne  pc-1
          .set  cycles = (cycles - (loop_cycles * 3))
        .endif
        .if (cycles > 0)
          .if   (cycles & 4)
            rjmp  pc+1
            rjmp  pc+1
          .endif
          .if   (cycles & 2)
            rjmp  pc+1
          .endif
          .if   (cycles & 1)
            nop
          .endif
        .endif
      .endif
.endmacro
```

Strukturierte Programmierung

SAM (Structured Assembly Macros)

von Klaus2m5

SAM unterstützt strukturiertes Programmieren durch If-Then-Else und Do-Loop Makros. Beliebige Verschachtelung und Mehrfachbedingungen sind möglich. Läuft unter aktuellen Versionen von AVRASM2.

Einige Beispiele:

Verschachteltes If-Then-Else

```
          cpi   r16,'a'
          ifeq  a_chr
            cpi   r17,'b'
            ifeq   a_and_b
              ;...a&b
            else   a_and_b
              ;...a&-b
            end    a_and_b
          else  a_chr
            cpi r17,'c'
            ifeq  c_but_no_a
              ;...-a&+c
            else  c_but_no_a
              ;...-a&c
            end   c_but_no_a
          end   a_chr
```

Mehrere und/oder verknüpfte Bedingungen

```
          cpi   zh,high(end_buffer)
          ifeq_and    end_buffer_reached
          cpi   zl,low(end_buffer)
          ifeq        end_buffer_reached
            ldi   zh,high(buffer)       ;wrap buffer
            ldi   zl,low(buffer)
          end         end_buffer_reached
          ld    r0,z+                   ;read buffer
```

Das Gleiche als Do-Loop

```
; reading a buffer until end
; similar to: for z = buffer to end_buffer
          ldi   zh,high(buffer)   ;init buffer-pointer
          ldi   zl,low(buffer)
          do    read_buf
            ld    r0,z+
            cpi   zh,high(end_buffer)
          loopne read_buf
            cpi   zl,low(end_buffer)
          loopne read_buf
```

Mehr Beispiele und das SAM-include als Download.

Von „http://www.mikrocontroller.net/articles/AVR_Assembler_Makros“
Kategorie: AVR