Anwendungen von
AVR-Einchip-Prozessoren AT90S,
ATtiny, ATmega und ATxmega

DCF77 Weckuhr mit LCD
Datum und Uhrzeit

Datum und Uhrzeit mit dem AVR

Es kommt gelegentlich vor, dass man das Datum und die Uhrzeit auf einem beliebigen AVR aktuell halten und darstellen muss. Diese Seite zeigt einige Möglichkeiten auf, wie man das auf einem AVR in Assembler programmieren kann.

0 Inhalt

1 Sekunden korrekt messen

1.1 Schleifen zum Timing

Hat der AVR sonst nichts anderes zu tun (was selten vorkommt), kann man die Sekunden mit Verzögerungsschleifen abzählen. Das geht so.

Da der AVR ziemlich schnell ist, braucht man zum Abzählen einer Sekunde bei 1 MHz Takt mindestens einen Zähler der bis ca. 300.000 zählen kann. Für diese Zahl braucht man 20 Bits, ein 8- oder 16-Bit-Zähler alleine reicht daher nicht aus. Mit einer 16-Bit-Schleife und einer äußeren 8-Bit-Schleife kommt man zurecht. Die 16-Bit-Schleife sieht so aus:

```
  .equ schleifendurchlaeufe = 62499 ; Anzahl Durchlaeufe definieren
  ldi R25,High(Schleifendurchlaeufe) ; Zaehler laden, ein Takt
  ldi R24,Low(Schleifendurchlaeufe) ; noch ein Takt
Schleife16:
  sbiw R24,1 ; Abwaerts zaehlen, zwei Takte
  brne Schleife16 ; Rueckwaerts, zwei Takte bei Sprung, ein Takt
  nop
  nop
  nop
```

Die abschliessenden NOP-Instruktionen machen einen gewissen Sinn, denn die Schleife braucht

- normalerweise vier Takte pro Durchlauf, und
- drei Takte beim letzten Durchlauf.

Zusammen mit den beiden LDI-Instruktionen und den NOP ermittelt sich die Anzahl Takte zu:

$$N = 2 + 4 * (Schleifendurchlaeufe - 1) + 3 + 3$$

Das verkürzt sich zu:

$$N = 4 * (Schleifendurchlaeufe + 1)$$

Will man bei 1 MHz Takt also genau 250 ms erreichen, braucht es N = 250.000 Takte. Schleifendurchlaeufe muss dann

$$\text{Schleifendurchlaeufe} = N / 4 - 1 = 62.499$$

sein. Das schafft ein 16-Bit-Zähler gerade noch so.

Lässt man das ganze vier mal ausführen, ist man bei einer Sekunde, z.B. so:

```
  .equ z8 = 4
  .equ z16 = 62499
  ldi R16,z8 ; Anzahl Durchlaeufe aussen, ein Takt
Scheife8:
  ldi R25,High(z16) ; Zaehler laden, ein Takt
  ldi R24,Low(z16) ; noch ein Takt
Schleife16:
  sbiw R24,1 ; Abwaerts zaehlen, zwei Takte
  brne Schleife16 ; Rueckwaerts, zwei Takte bei Sprung, ein Takt
  dec R16 ; ein Takt
  brne Schleife8 ; zwei Takte bei Sprung, ein Takt beim letzten
```

Die innere Schleife braucht jetzt

$$N16 = 2 + 4 * (z16 - 1) + 3 = 4 * z16 + 1$$

und das gesamte braucht

$$N = 1 + z8*N16 + 3 * (z8 - 1) + 2 =$$
$$1 + z8 * N16 + 3 * z8 - 1 =$$
$$z8 * N16 + 3 * z8$$

Das gibt dann genau 1.000.000 Takte.

Wie gesagt: nur wenn nichts weiteres dazu kommt und wenn der Takt von einem Quarz und nicht von einem super-ungenauen internen RC-Oszillator stammt.

1.2 Timer als Taktzähler

Die elendige Taktzählerei ist man los, wenn man dem Timer das Zählen überlässt. Der kann das viel besser, genauer und zuverlässiger und lässt sich vor allem durch nichts anderes von seinem Zählen ablenken.

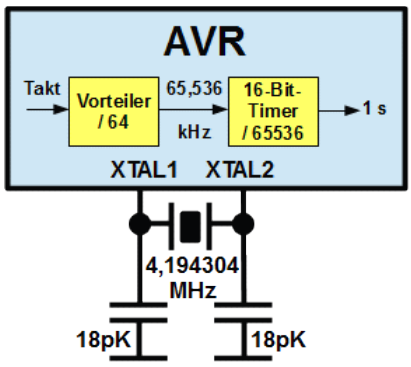
Unglücklicherweise ist bei einem MHz Takt schon bei einem Vorteiler von 64 Schluss mit lustig (ergibt 15.625 Hz), Teilerraten darüber liefern zunehmend krumme Frequenzen. Da eine Uhr aber sowieso einen Quarzoszillator braucht, damit sie einigermaßen genau geht, wählt man eben was besseres aus. Im Angebot sind z. B. 2,048 MHz, 2,097152 MHz, 2,4576 Mhz, 3,072 MHz, 3,2768 MHz oder 4,194304 Mhz.

Das hier sind die Teilerraten bei den möglichen Vorteilerwerten von 1, 8, 64, 256 und 1.024 für 8-Bit-Timer und für 16-Bit-Timer im Normalbetrieb. Man erkennt, dass der 8-Bit-Timer auch bei einem Vorteiler von 1.024 nicht so arg hoch kommt, Quarze mit diesen niedrigen Frequenzen müsste man sich handfertigen lassen und der Prozessor wäre auch lahm wie eine Ente.

Vorteiler	Timer 8-Bit	Teiler	Timer 16-Bit	Teiler
1	256	256	65.536	65.536
8		2.048		524.288
64		16.384		**4.194.304**
256		65.536		16.777.216
1024		262.144		67.108.864

Vielversprechender ist da ein 16-Bit-Timer mit einem Vorteiler von 64. Quarze mit 4,194.304 MHz gibt es an jeder Straßenecke zu kaufen und kosten gerade mal 25 Cent. Dann ist der Sekundentimer fast schon fertig, nur noch:

1. Quarz und zwei Keramikkondensatoren von 18 pF an den AVR anschließen,



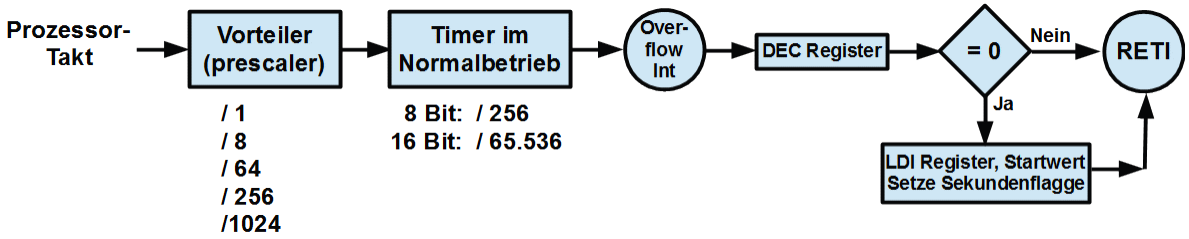
2. Fuses des Prozessors auf den externen Quarz umprogrammieren,
3. die Interrupt-Vektortabelle des Prozessors mit lauter *RETI* schreiben, außer dem Reset-Vektor (*RJMP Start* und dem Überlauf-Interrupt für TC1 (*RJMP TC1OvfIsr*),
4. die Überlauf-Interrupt-Service-Routine "TC1OvfIsr" für den Timer mit den zwei Instruktionen *SET* (Setzt die T-Flagge im Statusregister) und *RETI* (Rückkehr vom Interrupt) schreiben,
5. im Hauptprogramm "Start:" den Stapel initiieren,
6. dann den 16-Bit-Timer auf Normalbetrieb und den Vorteiler auf 64 einstellen,
7. dort auch den TC1-Überlauf-Interrupt (TOIE1) und die Interrupts generell mit *SEI* einschalten,
8. in einer Schleife immer abfragen, ob die Sekundenflagge T gesetzt ist und wenn ja, diese brav wieder löschen und das machen, was nach jeder Sekunde erledigt werden muss.

Schon ist eine sekundengenaue Quarzuhr ohne viele Umschweife fertig. Alles andere, was noch in der Schleife so gemacht wird, hat auf dieses Timing keinen Einfluss mehr: der Timer zählt stur bis er überläuft und setzt die T-Flagge.

- Wem 4,1 MHz Takt zu schnell vorkommt, weil er seine Quarzuhr mit Batterie oder Akku betreiben und wie wild Strom sparen will, oder
- wer grad gar keinen solchen Quarz an seiner Straßenecke kriegen sollte (hallo Conrad-Kunden!),
- wer den 16-Bit-Timer für andere Zwecke braucht, z. B. um mit dem Wecker schöne Melodien abzuspielen, wenn es 07:30 Uhr ist,

der kann auch andere Lösungen erfinden.

Wichtig dabei ist nur, dass die gewählte Quarzfrequenz, geteilt durch den Vorteiler und die Teilerrate durch den Timer (256 oder 65,536) keine krumme Frequenz ergibt. Bleibt beim Teilen noch eine Ganzzahl übrig, kriegt man die durch Opfern eines Registers (wenn sie kleiner oder gleich 256 ist) oder eines Registerpaars (wie z. B. R25:R24, wenn sie größer 256 ist) weg.



Mit den eingangs genannten Quarzen ergeben sich für einen 8-Bit-Timer die folgenden ganzzahligen Teilerwerte für den Registerzähler.

Quarzfrequenz	2.048.000	2.097.152	2.457.600	3.072.000	3.276.800	4.194.304
8-Bit, ./. 256	8.000	8.192	9.600	12.000	12.800	16.384
Presc, ./. 8	1.000	1.024	1.200	1.500	1.600	2.048
Presc, ./. 64	125	128	150		200	256
Presc, ./. 256		32			50	64
Presc, ./. 1024		8				16

Die grün hinterlegten Teilerwerte passen in ein 8 Bit breites Register, nur bei 3,072 MHz braucht man einen 16 Bit breiten Registerzähler.

In Assembler geht die Teilerei dann z. B. so: Das geht dann z. B. per Timer-Overflow-Interrupt so:

```
  ; Rechenkonstanten
  .equ takt = 2097152 ; Quarzfrequenz
  .equ vorteiler = 1024
  .equ timertop = 256
  .equ sekundenteiler = takt / timertop / vorteiler ; = 8
  ;
  ; Register
  .def rSreg = R15 ; Sichern des Statusports
  .def rSekundenteiler = R17 ; Abwaertszaehler
  .def rFlag = R18 ; Flaggenregister
  .equ bSek = 0 ; Sekundenflagge
;
Tc0OvflwInt:
  in rSreg,SREG ; SREG sichern
  dec rSekundenteiler ; Teilt durch 8
  brne TC0OvflwIntRet ; Noch nicht Null
  ldi rSekundenteiler,sekundenteiler ; Neustart Teiler
  sbr rFlag,1<<bSek ; Flagge setzen: Sekunde ist um
Tc0OvflwIntRet:
  out SREG,rSreg ; Wiederherstellen SREG
  reti
```

Die Methode ist nicht so arg ideal, wenn man aus anderen Gründen auf eine bestimmte Taktfrequenz angewiesen ist. Nur bei 4,0 MHz Takt und einem 8-Bit-Timer durch 256 bei kommt bei einem Vorteiler von 1 die Ganzzahl 15.625 heraus. Die kann man in das Doppelregister R25:R24 packen und bei jedem Overflow-Interrupt des Timers mit *SBIW R24,1* um Eins abwärts zählen. Erreicht der Zähler Null, ist die Sekunde um und der Zähler wird mit 15.625 wieder neu gestartet. Nicht bei vielen anderen Frequenzen funktioniert das so. Aber es gibt da eine weitere Methode.

Viele schönen Taktfrequenzen wie 1 oder 2 MHz kriegt man beim Teilen durch 256 oder 65.536 rein gar nicht zu ganzzahligen Resultaten und daher so nicht in den Griff. Da muss der Timer dann in den CTC-Modus gebracht werden, damit er auch andere Teilerraten als nur 256 kann. Beim CTC-Modus setzt der Timer sich auf Null zurück, wenn er beim vorausgehenden Takt einen Vergleichswert in einem Vergleichsregister (Compare A, bei 16-Bit-Zählern auch das Input Capture Register ICR) erreicht hatte. Da er immer nachfolgend Gleichheit prüft, muss der Vergleichswert um Eins niedriger eingestellt werden als man teilen möchte. Braucht man den gleichen Timer auch noch für andere Zwecke, z. B. als Pulsweitenmodulator, ändert sich dadurch natürlich dessen Auflösung.

Bei einem MHz Takt kann man den Vorteiler auf 64 stellen (15.625 kHz Takteingang am Timer), den Timer durch 125 teilen lassen (Compare Match auf 124) und kommt mit einem Registerteiler von 125 dann auf eine Sekunde. Viele andere Frequenzen kriegt man so auf die Sekunde heruntergeteilt.

Und schon hat man ein todsicher genaues Sekundensignal ganz ohne Zählgrab. Natürlich muss dazu

- der Zähler laufen,
- der Timer-Interrupt eingeschaltet sein (TOIE0 oder TOIE1 beim Overflow-Int bzw. OCIE0A

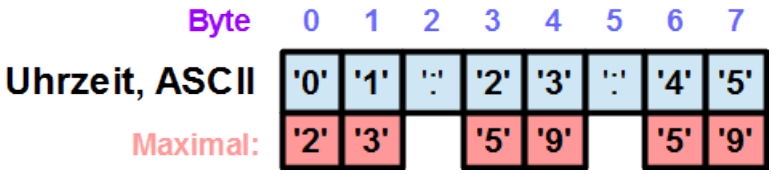
oder OCIE1A beim Compare-Match-A-Int),
- die übergeordnete Interruptflagge I im SREG gesetzt sein,
- der Interruptvektor gesetzt sein und zur Serviceroutine verzweigen, und
- der Stapel gesetzt sein und funktionieren.

Seitenanfang | Sekunden | Formate | Datum

2 Uhrzeit-Formate

Uhrzeiten lassen sich in mindestens vier gebräuchlichen Formaten handhaben. Glücklicherweise lassen sich alle Formate ineinander umwandeln.

2.1 Zeit im ASCII-Format



Das auf den ersten Blick einfachste Format ist das ASCII-Format. Jede Dezimalstelle des Datums wird als ASCII-Zeichen in einem Byte gespeichert. Das ASCII-Format stammt von Militärfernschreibern der 1950er ab und ist so hartnäckig wie das Inch oder der Kubikfuß, aber älter als das Fass ("barrel"). Die Null ist danach die Dezimalzahl 48 oder hexadezimal 0x30. Das Einschließen der Zahlen und Zeichen in 'x' sagt: es handelt sich um die ASCII-Repräsentation des Zeichens (bei der '1' um dezimal 49). In AVR-Assembler geht das dann so:

```
ldi R16,'0' ; ASCII-Zeichen fuer Null, Dezimal 48
```

Das Format hat den Vorteil, dass man die so gespeicherte Uhrzeit unmittelbar auf einer LCD ausgeben kann. Daher kann man den Zahlen auch gleich die Trennzeichen : hinzufügen, die dann in einem Rutsch ebenfalls auf die LCD ausgegeben werden können.

Es werden acht Bytes gebraucht, vorzugsweise kann daher dieses Uhrzeitformat im SRAM gespeichert und gehandhabt werden. Den Platz dafür kriegt man so definiert:

```
.dseg ; Datensegment: nur Label und .byte-Direktiven
Zeit: ; Label mit Adresse des Zeit-Bereichs im SRAM
.byte 8 ; Acht Bytes Platz im SRAM reservieren
```

Um die Uhr auf Null zu stellen und die beiden Doppelpunkte schon mal in das SRAM zu schreiben kann man so vorgehen:

```
ldi XH,High(Zeit) ; MSB der SRAM-Adresse in XH
ldi XL,Low(Zeit) ; LSB in XL
ldi R16,'0' ; ASCII-Null in R16
st X+,R16 ; Schreibe Stunden-Zehner und erhoehe X
st X+,R16 ; Und in die Stunden-Einer
ldi R16,':' ; Trennzeichen
st X+,R16 ; in die erste Trennzeichenstelle
ldi R16,'0' ; ASCII-Null in das Register
st X+,R16 ; und in den Minuten-Zehner
st X+,R16 ; und in den Minuten-Einer
ldi R16,':' ; Trennzeichen
st X+,R16 ; in die zweite Trennzeichenstelle
ldi R16,'0' ; ASCII-Null in das Register
st X+,R16 ; in die Sekunden-Zehner
st X,R16 ; und die Sekunden-Einer
```

Damit ist die Uhrzeit mit den beiden Trennzeichen gespeichert.

Es geht aber auch anders, nämlich ohne das andauernde Umladen des Registers:

```
  ldi YH,High(Zeit) ; MSB der SRAM-Adresse der Zeit in YH
  ldi YL,Low(Zeit) ; LSB in YL
  ldi R16,'0' ; ASCII-Null in R16
  st Y,R16 ; in die Stunden-Zehner
  std Y+1,R16 ; in die Stunden-Einer
  std Y+3,R16 ; in die Minuten-Zehner
  std Y+4,R16 ; in die Minuten-Einer
  std Y+6,R16 ; in die Sekunden-Zehner
  std Y+7,R16 ; und in die Sekunden-Einer
  ldi R16,':' ; Trennzeichen in R16
  std Y+2,R16 ; in die erste Trennzeichenstelle
  std Y+5,R16 ; in die zweite Trennzeichenstelle
```

STD (und beim Lesen: LDD) ändert die Adresse in Y nicht sondern addiert die Zahl dahinter nur temporär vor dem Schreibvorgang. Nach dem Schreiben bleibt Y gleich wie vorher. Mit dem Doppelregister X geht das nicht, nur mit Y und Z.

Um die Uhrzeit um eine Sekunde zu erhöhen, wird mit dem hintersten Byte begonnen. Überschreitet dieses Byte nach dem Erhöhen die "9", z. B. so:

```
  ldi YH,High(Zeit) ; MSB der SRAM-Adresse der Zeit in YH
  ldi YL,Low(Zeit) ; LSB in YL
  ldd R16,Y+7 ; Lese Sekunden-Einer in R16
  inc R16 ; Erhoehe Sekunden-Einer um Eins
  std Y+7,R16 ; Schreibe erhoehte Sekunden-Einer
  cpi R16,'9'+1 ; Vergleiche mit dem naechsten Zeichen hinter '9'
  brcs Fertig ; Wenn carry gesetzt kein Ueberlauf
  ldi rmp,'0' ; Starte Sekunden-Einer neu
  std Y+7,R16 ; Schreibe Null in Sekunden-Einer
  ldd R16,Y+6 ; Lese Sekunden-Zehner
  inc R16 ; Erhoehe Sekunden-Zehner
  std Y+6,R16 ; und schreibe zurueck in Sekunden-Zehner
  cpi R16,'6' ; Sekunden-Zehner auf Sechs?
  brcs Fertig ; Nein, schon fertig
  ; ... Minuten und Stunden in gleicher Weise
Fertig:
  ; ... Hier ist die Sekundenerhoehung fertig
```

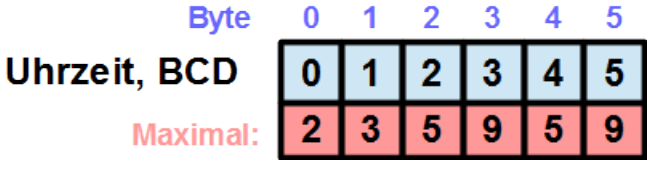
Bei der Erhöhung der Stunde ist zweierlei zu überprüfen:

- ob der Stunden-Einer größer als '9' wird, und
- ob der Stunden-Einer vier wird UND gleichzeitig der Stunden-Zehner '2' ist.

Die Handhabung der Sekundenerhöhung beim ASCII-Format ist dank der relativen Adressierung mit STD und LDD recht einfach.

2.2 Zeit im BCD-Format

Beim zweiten Format werden die Zehner und Einer von Sekunden, Minuten und Stunden nicht als ASCII-Zeichen, sondern als binär kodierte Dezimalziffern gespeichert (BCD, binary coded digit). Die Bytes haben binäre Werte



zwischen 0 und 9 (Einer) bzw. 0 bis 5 (Sekunden- und Minutenzehner) bzw. 0 bis 2 (Stunden-Zehner).

Der Vergleich, ob bei den Einern die 9 überschritten wird, erfolgt jetzt mit *CPI R16,10* statt mit *CPI R16,'9'+1*. Der Neustart der Einer erfolgt statt mit *LDI R16,'0'* mit *CLR R16* oder mit *LDI R16,0*. Alles andere im oberen Erhöhungscode bleibt gleich, bis auf die beiden fehlenden Doppelpunkte.

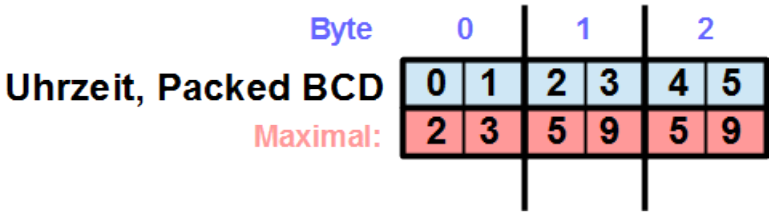
Gibt man die BCD-Zahlen auf die LCD aus, muss man einfach die ASCII-0 addieren und das Ergebnis auf die LCD schreiben. Da der AVR keine Instruktion hat, mit der sich 48 zur BCD-Zahl addieren lässt, gibt es drei Lösungen für diesen Schritt:

1. Man schreibt die 48 in ein anderes Register und addiert dieses andere Register mit *ADD R16,R17* zu demjenigen mit der BCD-Zahl.
2. Man setzt die Bits 4 und 5 in der BCD-Zahl mit *ORI R16,0x30* oder mit *SBR R16,0x30* oder auch mit *SBR R16,(1<<4)|(1<<5)* auf Eins. Auf diese Weise wird ebenfalls aus BCD-0 die ASCII-'0' und aus 9 die '9'.
3. Man subtrahiert -48 von dem Register mit *SUBI R16,-'0'*. Da aus zwei Minusvorzeichen Plus wird, kommt dasselbe wie beim Addieren von '0' heraus.

Alle drei Methoden haben den gleichen Effekt. Nur bei der ersten Methode ist vorübergehend ein weiteres Register nötig.

Beim Ausgeben auf die LCD die beiden Trennzeichen an den richtigen Stellen nicht vergessen, sonst sieht die Uhrzeit ziemlich doof aus.

2.3 Zeit im gepackten BCD-Format



Da so eine BCD-Zahl mit vier Bits auskäme, kann man zwei von denen in ein einziges Byte quetschen. In die unteren vier Bit (0..3) kommen die Einer, in die oberen vier Bit (4..7) die Zehner. Die Uhrzeit braucht dann nur drei Bytes. Das heißt dann gepacktes BCD oder "Packed BCD".

Wenn man eine gepackte BCD-Zahl um Eins erhöhen will, kann man natürlich ebenfalls *INC R16* verwenden. Es ist dann aber etwa aufwändiger festzustellen, ob das untere Nibble (halbes Byte, vier Bits) die 9 überschritten hat. Dazu wäre zuerst das obere Nibble zu löschen und danach erst zu vergleichen. Da das obere Nibble aber später wieder gebraucht wird, ist das ungünstig.

Es gibt aber eine viel einfachere Möglichkeit: das Halbübertragsbit. Es stellt beim Addieren mit *ADD* oder auch beim SUBI (nicht beim Inkrementieren mit *INC*!) fest, ob ein Übertrag vom unteren in das obere Byte erfolgt. Es ist das H-Bit im Statusregister SREG. Abhängig von der letzten Operation kann mit *BRHC* oder mit *BRHS* verzweigt werden, wenn das H-Bit gelöscht (BRHC, clear) oder gesetzt ist (BRHS, set). Addiert man statt eine 1 sieben, würde man am H-Bit erkennen, ob dabei ein Übertrag in das obere Nibble erfolgt. Ist das der Fall, dann ist alles schon ok, denn

- das untere Nibble ist durch Addieren von 7 zu 0b1001 (9) ein 0b0000 geworden,
- das obere Nibble ist durch das Addieren von 7 zu der Neun im unteren Nibble um Eins erhöht (0b0111 + 0b1001 = 0b1.0000). Das Zehner-Nibble ist daher um Eins erhöht.

Falls das H-Flag nicht gesetzt ist, dann war vor dem Addieren keine Neun im unteren Nibble. Dann müssen von den addierten 7 wieder sechs abgezogen werden. Das klingt total einfach und ist es auch.

Eine Besonderheit tritt auf, wenn zum Addieren der Sieben nicht *LDI R17,7* und *ADD R16,R17* benutzt sondern die Instruktion *SUBI R16,-7* verwendet wird. In diesem Fall kehrt sich das H-Flag um: H clear bedeutet dann einen Übertrag und H set keinen Übertrag.

Um festzustellen, ob die 59 Sekunden überschritten sind, genügt der Vergleich mit 0x60: tritt dabei ein Carry auf, dann sind die 59 Sekunden noch nicht überschritten. Wenn nicht wird das gepackte BCD der Sekunden einfach auf Null gesetzt.

Beim gepackten BCD-Format der Stunden ist das die Erkennung des Tageswechsels noch viel einfacher: mit *CPI R16,0x24* und danach gelöschtem Carry sind die 24 Stunden voll. Nichts mit zwei Bytes vergleichen.

Der Einfachheit halber die ganze Zeiterhöherei um eine Sekunde in gepacktem BCD:

```
  ldi ZH,High(Zeit) ; Z zeigt auf die Stunden in gepacktem BCD
  ldi ZL,Low(Zeit)
  ldd R16,Z+2 ; Lese die Sekunden
  subi R16,-7 ; Addiere Sieben
  brhc ChkSek ; H geloescht, Zehner erhöht, Sekunden auf 60 pruefen
  subi R16,6 ; H gesetzt, Sechs wieder abziehen
ChkSek:
  std Z+2,R16 ; Sekunden schreiben
  cpi R16,0x60 ; 60 Sekunden?
  brcs Fertig ; Nein, schon fertig
  clr R16 ; Sekunden mit 0 beginnen
  std Z+2,R16 ; Und in das SRAM schreiben
  ldd R16,Z+1 ; Lese Minuten
  subi R16,-7 ; Addiere Sieben
  brhc ChkMin ; H geloescht, Zehner erhoeht, Minten auf 60 pruefen
  subi R16,6 ; H gesetzt, Sechs wieder abziehen
ChkMin:
  std Z+1,R16 ; Und in das SRAM schreiben
  cpi R16,0x60 ; 60 Minuten erreicht?
  brcs Fertig ; Nein, schon fertig
  clr R16 ; Minuten mit Null beginnen
  std Z+1,R16 ; und ins SRAM schreiben
  ld R16,Z ; Stunden lesen
  subi R16,-7 ; Addiere sieben
  st Z,R16 ; und in das SRAM schreiben
  brhc ChkStd ; H geloescht, Stunden stimmen schon
  subi R16,6 ; H gesetzt, Sechs wieder abziehen
  st Z,R16 ; und in das SRAM schreiben
ChkStd:
  cpi R16,0x24 ; 24 Stunden voll?
  brcs Fertig ; Gesetztes Carry wenn kleiner 24
  clr R16 ; Stunden mit Null beginnen
  st Z,R16 ; ins SRAM schreiben
Fertig: ; Erhoehung fertig erfolgt
```

Das war schon alles. Wer es nicht glaubt kann den Code mit einem Simulator überprüfen. Natürlich müssen Stunden, Minuten und Sekunden im SRAM auf einen korrekten Uhrzeitwert, z. B. auf "23:59:59", gesetzt werden.

Um gepackte BCD-Zahlen auf die LCD auszugeben, muss man die Ziffern nacheinander (das obere Nibble zuerst) in die entsprechende ASCII-Ziffer umwandeln. Um die beiden Ziffern auszugeben, muss man so vorgehen:

```
  ld R16,Z ; Z zeigt auf die Stunde, lese Stunden
  swap R16 ; Vertausche oberes und unteres Nibble
  and R16,0x0F ; Isoliere das untere Nibble
  subi R16,-'0' ; Addiere ASCII-Null
  ; R16 an LCD ausgeben
  ld R16,Z ; Stunden noch mal lesen
  andi R16,0x0F ; Isoliere unteres Nibble
  subi R16,-'0' ; Addiere ASCII-Null
  ; R16 an LCD ausgeben
```

Nach den Stunden kommt dann das Trennzeichen dran, dann die Minuten wie oben, wieder ein Trennzeichen und schließlich die Sekunden wie oben. Da die obige Routine drei Mal identisch ausgeführt wird, kann man sie auch als Unterprogramm formulieren und drei Mal aufrufen. Natürlich nachdem man den Stapel angelegt hat. Vor dem Aufruf zur Ausgabe der Minuten und Sekunden erhöht man einfach den Zeiger Z, z. B. mit *ADIW ZL,1*. Geht alles mit ganz wenigen einfachen Instruktionen und schon ist die Uhr fertig.

2.4 Zeit im Binärformat

Byte	0	1	2
Uhrzeit, Binär	1	23	45
Maximal:	23	59	59

Am Schluss die allereinfachste aller Formatierungen: Sekunden, Minuten und Stunden im binären Format. Die Einer und Zehner passen binär in ein Byte, können mit einem einfachen *INC* um eins erhöht werden und zum Feststellen, ob das Maximum erreicht ist, ist auch nur ein einfacher Vergleich nötig. Da nur drei Bytes nötig sind, kann man das zur Abwechslung mal in drei Registern erledigen. Das Erhöhen der Uhrzeit um eine Sekunde in Assembler geht dann so:

```
  .def rStd = R4 ; Stundenregister
  .def rMin = R5 ; Minutenregister
  .def rSek = R6 ; Sekundenregister
IncSek:
  ldi R16,60 ; Ende feststellen
  inc rSek
  cp rSek,R16 ; Sekunde kleiner Minutenende?
  brcs Fertig ; Nein
  clr rSek ; Sekunden neu beginnen
  inc rMin ; Minuten erhoehen
  cp rMin,R16 ; Minuten kleiner Stundenende?
  brcs Fertig ; Nein
  clr rMin ; Minuten neu beginnen
  inc rStd ; Stunden erhoehen
  ldi R16,24 ; Stundenende
  cp rStd,R16 ; Tag zu Ende?
  brcs Fertig ; Nein
  clr rStd ; Stunden auf Null
Fertig:
  ; Sekundenerhoehung beendet
```

Mit 14 Einfachst-Instruktionen für eine fertige Uhrzeit nicht gerade intellektuell anspruchsvoll. Jedenfalls kein Grund, irgendeine mächtige C-Datumsbibliothek zu laden.

Dafür muss jetzt erst jede Binärzahl vor der Ausgabe auf die LCD in zweistelliges ASCII verwandelt werden. Da alle drei Bytes in gleicher Weise auszugeben sind, macht das eine Unterroutine, der wir in R16 die auszugebende Binärzahl übergeben.

```
Bin2Dez2:
  clr R0 ; Zehner zaehlen in R0
Bin2Dez2a:
  inc R0 ; Zaehler erhoehen
  subi R16,10 ; Zehn abziehen
  brcc Bin2Dez2a ; Kein Carry, weiter abziehen
  subi R16,-10-48 ; Letztes Abziehen rueckgaengig machen (10 addieren)
                  ; und in ASCII verwandeln (48 addieren)
  push R16 ; Wird noch gebraucht, auf den Stapel
  ldi R16,'0'-1 ; Zaehler = 1 zu ASCII-Null
  add R16,R0 ; Zaehler dazu addieren
  rcall LcdChar ; R16 als Zeichen an die LCD ausgeben
  pop R16 ; Zweite Dezimalstelle vom Stapel holen
```

```
  rjmp LcdChar ; und auf die LCD ausgeben
```

Die Routine LcdChar gibt das Zeichen in R16 auf die LCD aus. Wegen des *RCALL* sowie wegen *PUSH* und *POP* muss natürlich die Stapelverwaltung funktionieren. Zum Testen und Simulieren setzt man z. B. Z auf den Beginn des SRAM (SRAM_START) und gibt mit der Routine LcdChar den Inhalt mit *ST Z+,R16* und *RET* die Zeichen statt in die LCD ins SRAM aus.

Die Uhrzeitanzeige ist dann denkbar einfach:

```
Anzeige:
  mov R16,rStd ; Stunden in R16
  rcall Bin2Dez2 ; Ausgaberoutine aufrufen
  ldi R16,':' ; Trennzeichen
  rcall LcdChar ; ausgeben
  mov R16,rMin ; Minuten in R16
  rcall Bin2Dec2 ; Ausgaberoutine aufrufen
  ldi R16,':' ; Noch ein Trennzeichen
  rcall LcdChar ; ausgeben
  mov R16,rSek ; Sekunden in R16
  rjmp Bin2Dez2 ; und ausgeben
```

Mit gerade mal 20 Instruktionen, davon 10 für die BCD-Anzeige, auch nicht gerade riesen-anspruchsvoll.

2.5 Das beste Format

... ist natürlich das Binärformat. Aber die anderen drei Formate haben auch so das Eine oder Andere auf ihrer Seite. Mach also was Du willst, es geht einfach alles.

Seitenanfang	Sekunden	Uhrzeit-Formate	Datum

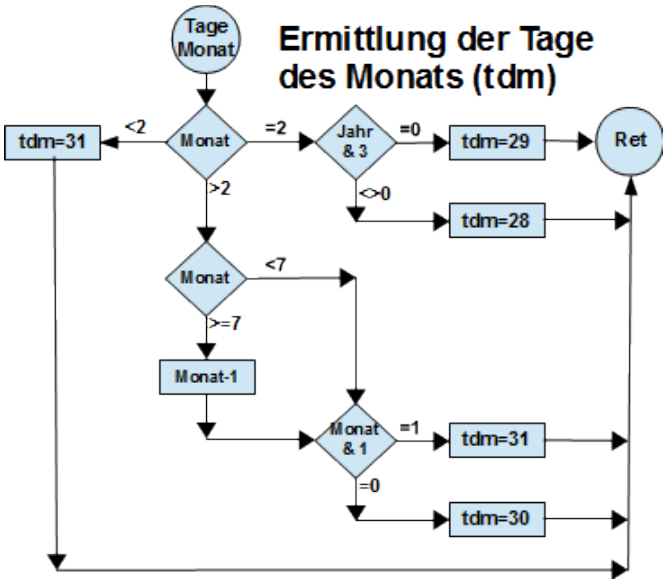
3 Uhrzeit und Datum

Nachdem wir die Uhrzeit bewältigt haben, kann uns auch das Datum nicht mehr abschrecken. Es funktioniert ganz genau so, nur

- dass Papst Gregor der XIII. in 1582 mit der päpstlichen Bulle "*Inter gravissimas*" entschieden hat, dass es im Gegensatz zu Sekunden, Minuten, Stunden und Jahren keinen Tag Null und keinen Monat Null gibt und dass diese im Gegensatz zu allem anderen beiden immer schon bei Eins anfangen,
- selbiger ferner entschieden hat, dass der Februar mal 28 und mal 29 Tage hat, je nachdem ob das Jahr durch vier teilbar ist oder nicht, aber nicht, wenn das Jahr durch 100, aber doch wenn das Jahr durch 400 teilbar ist, u.v.a.m.,
- es sieben Wochentage gibt und nicht etwa 10, wie dies in einer Dezimalgesellschaft zu erwarten wäre.

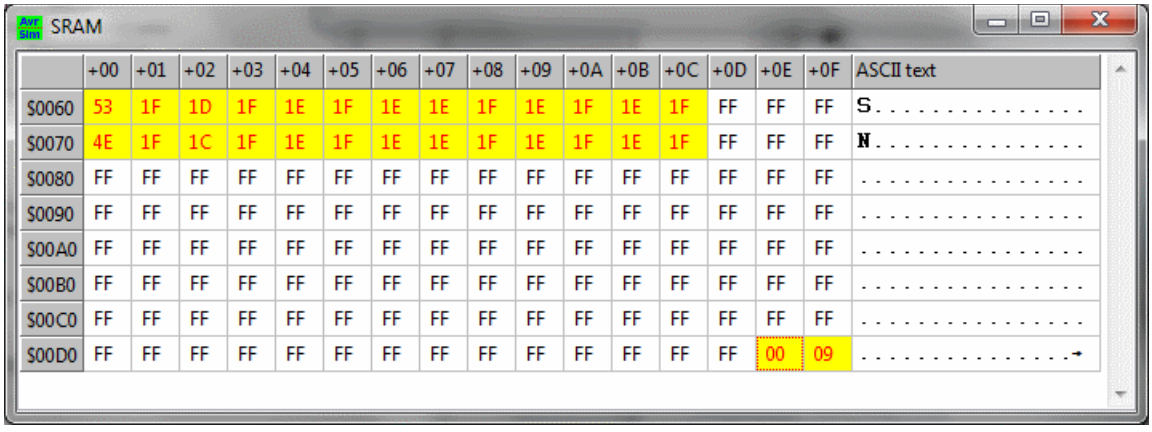
Bei der Anordnung im SRAM oder auf der LCD herrscht ebenfalls die totale Unlogik vor: Tage und Monate kommen vor die Jahre und nicht dahinter (bei den Angelsachsen ist es noch verrückter, da kommen die Monate vor den Tagen, aber die Jahre ganz hinten).

Das alles macht den Programmierer ganz wuschig. Um z.B. die Tage zu bestimmen, die ein Monat hat, ist nebenstehender Algorithmus nötig. Sieht kompliziert aus, ist aber in Assembler gar nicht so schlimm:




```
;
; Unterprogramm ermittelt die Tage des
;   Monats
;
TageMonat:
  cpi rMonat,2 ; Februar?
  brne TageMonat1 ; Nein
  ldi rTdm,28 ; Kein Schaltjahr
  mov rmp,rJahr ; Schaltjahr?
  andi rmp,0x03 ; Jahr durch vier
teilbar?
  brne TageMonatRet
  ldi rTdm,29 ; Schaltjahr
TageMonatRet:
  ret ; Fertig
TageMonat1:
  ldi rTdm,31 ; 31 Tage Januar
  brcs TageMonatRet
  mov rmp,rMonat
  cpi rmp,7
  brcs TageMonat2
  dec rmp
TageMonat2:
  ldi rTdm,31 ; Monat mit 31 Tagen
  andi rmp,0x01 ; Ungerade
  brne TageMonatRet
  ldi rTdm,30
  rjmp TageMonatRet
```

Aufgerufen mit den Monatstagen 1 bis 12 liefert die Routine für ein Schaltjahr "S" die obere Reihe, für kein Schaltjahr "N" die untere Reihe an Ergebnissen.



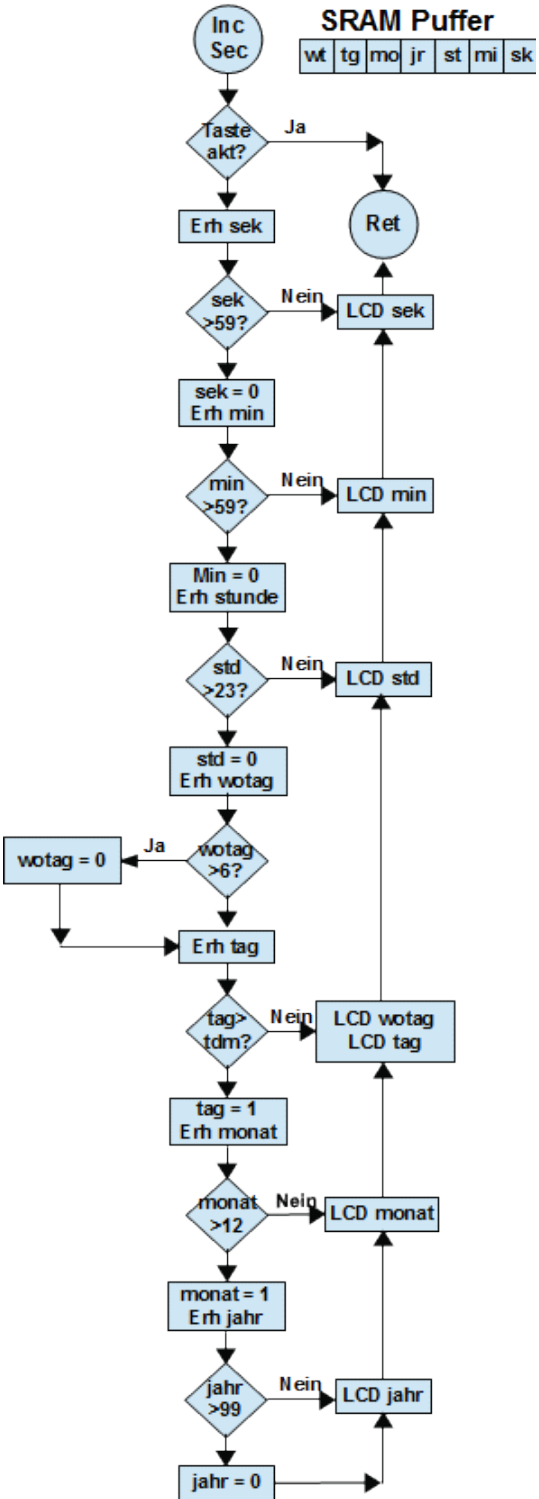
	+00	+01	+02	+03	+04	+05	+06	+07	+08	+09	+0A	+0B	+0C	+0D	+0E	+0F	ASCII text
$0060	53	1F	1D	1F	1E	1F	1E	1E	1F	1E	1F	1E	1F	FF	FF	FF	S. . . . .
$0070	4E	1F	1C	1F	1E	1F	1E	1E	1F	1E	1F	1E	1F	FF	FF	FF	N. . . . .
$0080	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	. . . . .
$0090	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	. . . . .
$00A0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	. . . . .
$00B0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	. . . . .
$00C0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	. . . . .
$00D0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	00	09	. . . . .

Mit diesem Handwerkszeug kann man sich an die Programmierung von Datum und Uhrzeit machen. Als Format ist binär gewählt, die Lokalisierung im SRAM-Puffer zeigt die Anordnung oben. Nur die jeweils geänderten Werte werden auf der LCD aktualisiert. Zu Beginn ist noch ein Schalter eingebaut, der die Uhrzeit- und Datumserhöhung unterbindet, wenn daran manuell gearbeitet wird (mit Eingabetasten).

Das sieht alles sehr kompliziert aus, ist aber immer dasselbe nur mit kleinen Varianten. Hier nur die

Erhöhungs-Routine ohne die LCD-Ausgaben.

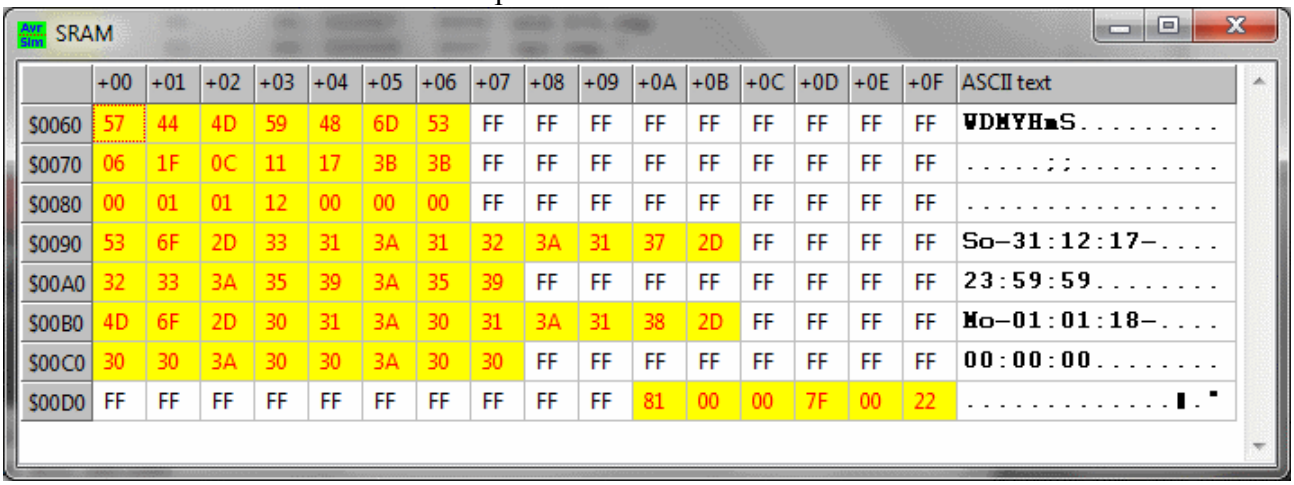
```
;
; Sekunde erhoehen
;
IncSec:
  ldi ZH,High(DatumZeit)
  ldi ZL,Low(DatumZeit)
  ldd rmp,Z+6 ; Sekunden
  inc rmp
  std Z+6,rmp
  cpi rmp,7
  brcs IncSecRet
  clr rmp
  std Z+6,rmp
  ldd rmp,Z+5 ; Minuten
  inc rmp
  std Z+5,rmp
  cpi rmp,60
  brcs IncSecRet
  clr rmp
  std Z+5,rmp
  ldd rmp,Z+4 ; Stunden
  inc rmp
  std Z+4,rmp
  cpi rmp,24
  brcs IncSecRet
  clr rmp
  std Z+4,rmp
  ld rmp,Z ; Wochentage
  inc rmp
  st Z,rmp
  cpi rmp,7
  brcs IncDay
  clr rmp
  st Z,rmp
IncDay:
  rcall DaysOfMonth ; Tage
  inc rmp
  mov rData,rmp
  ldd rmp,Z+1
  inc rmp
  std Z+1,rmp
  cp rmp,R0
  brcs IncSecRet
  ldi rmp,1
  std Z+1,rmp
  ldd rmp,Z+2 ; Monate
  inc rmp
  std Z+2,rmp
  cpi rmp,13
  brcs IncSecRet
  ldi rmp,1
  std Z+2,rmp
  ldd rmp,Z+3 ; Jahre
  inc rmp
  cpi rmp,100
  std Z+3,rmp
  brcs IncSecRet
  clr rmp
  std Z+3,rmp
IncSecRet:
  ret
```



Auch das sollte bewältigbarer Code sein, wenn man die wenigen vorkommenden Instruktionen mal verstanden hat.

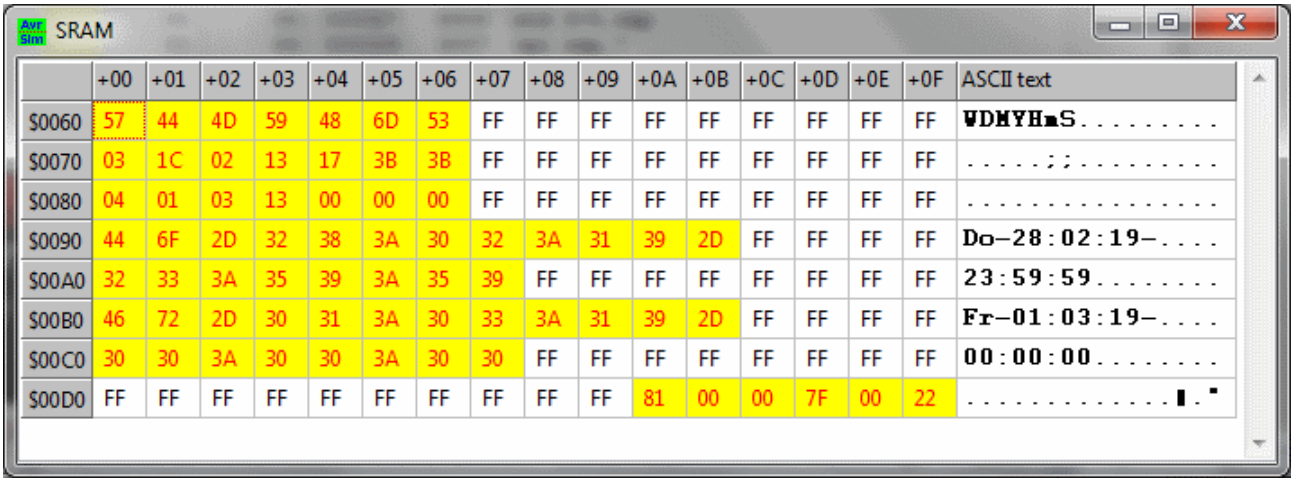
Auch hier wieder ein paar Tests. Zuerst der Jahreswechsel am 31.12.2017. Die oberste Reihe an $0060 zeigt die Anordnung der Binärbytes an. In der Zeile darunter steht das Ausgangsdatum binär und darunter das Enddatum binär nach der Erhöhung. Die Ausgaben auf der LCD stehen darunter und sind im Textbereich der Anzeige im Klartext zu sehen. Der Jahreswechsel funktioniert einwandfrei.

In der untersten Reihe ist noch der Stapel zu sehen.



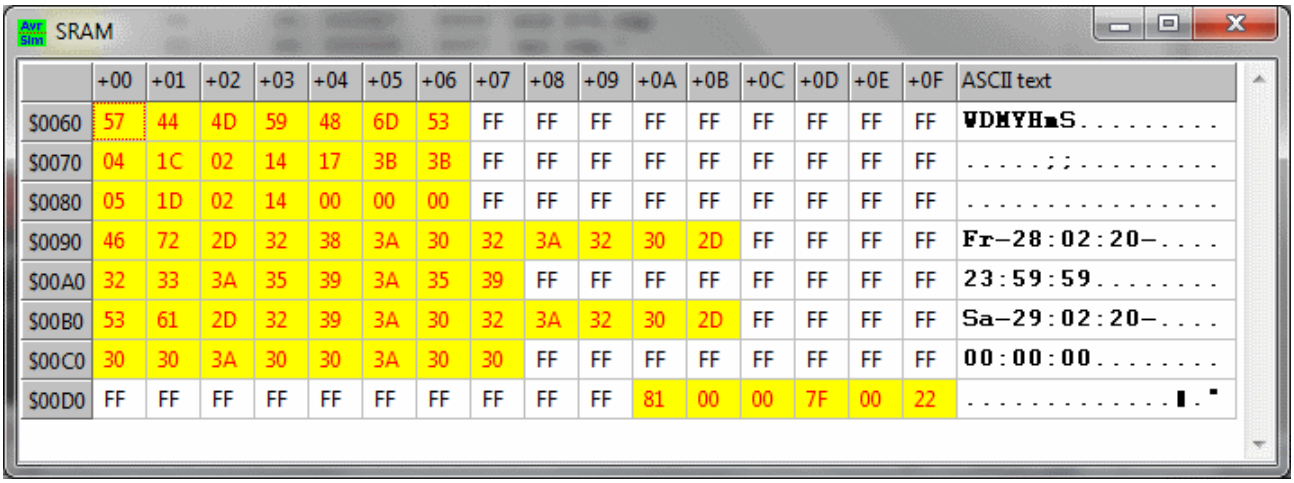
	+00	+01	+02	+03	+04	+05	+06	+07	+08	+09	+0A	+0B	+0C	+0D	+0E	+0F	ASCII text
$0060	57	44	4D	59	48	6D	53	FF	FF	FF	FF	FF	FF	FF	FF	FF	WDMYHmS. . . . .
$0070	06	1F	0C	11	17	3B	3B	FF	FF	FF	FF	FF	FF	FF	FF	FF	. . . . . ;; . . . . .
$0080	00	01	01	12	00	00	00	FF	FF	FF	FF	FF	FF	FF	FF	FF	. . . . .
$0090	53	6F	2D	33	31	3A	31	32	3A	31	37	2D	FF	FF	FF	FF	So-31:12:17-. . . .
$00A0	32	33	3A	35	39	3A	35	39	FF	FF	FF	FF	FF	FF	FF	FF	23:59:59. . . . .
$00B0	4D	6F	2D	30	31	3A	30	31	3A	31	38	2D	FF	FF	FF	FF	Mo-01:01:18-. . . .
$00C0	30	30	3A	30	30	3A	30	30	FF	FF	FF	FF	FF	FF	FF	FF	00:00:00. . . . .
$00D0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	81	00	00	7F	00	22	. . . . . ▮ . "

Das ist die Sekundenerhöhung am 28.02.2019, kein Schaltjahr.



	+00	+01	+02	+03	+04	+05	+06	+07	+08	+09	+0A	+0B	+0C	+0D	+0E	+0F	ASCII text
$0060	57	44	4D	59	48	6D	53	FF	FF	FF	FF	FF	FF	FF	FF	FF	WDMYHmS. . . . .
$0070	03	1C	02	13	17	3B	3B	FF	FF	FF	FF	FF	FF	FF	FF	FF	. . . . . ;; . . . . .
$0080	04	01	03	13	00	00	00	FF	FF	FF	FF	FF	FF	FF	FF	FF	. . . . .
$0090	44	6F	2D	32	38	3A	30	32	3A	31	39	2D	FF	FF	FF	FF	Do-28:02:19-. . . .
$00A0	32	33	3A	35	39	3A	35	39	FF	FF	FF	FF	FF	FF	FF	FF	23:59:59. . . . .
$00B0	46	72	2D	30	31	3A	30	33	3A	31	39	2D	FF	FF	FF	FF	Fr-01:03:19-. . . .
$00C0	30	30	3A	30	30	3A	30	30	FF	FF	FF	FF	FF	FF	FF	FF	00:00:00. . . . .
$00D0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	81	00	00	7F	00	22	. . . . . ▮ . "

Und das die Sekundenerhöhung am 20.02.2010, einem Schaltjahr. Alles korrekt.



	+00	+01	+02	+03	+04	+05	+06	+07	+08	+09	+0A	+0B	+0C	+0D	+0E	+0F	ASCII text
$0060	57	44	4D	59	48	6D	53	FF	FF	FF	FF	FF	FF	FF	FF	FF	WDMYHmS. . . . .
$0070	04	1C	02	14	17	3B	3B	FF	FF	FF	FF	FF	FF	FF	FF	FF	. . . . . ;; . . . . .
$0080	05	1D	02	14	00	00	00	FF	FF	FF	FF	FF	FF	FF	FF	FF	. . . . .
$0090	46	72	2D	32	38	3A	30	32	3A	32	30	2D	FF	FF	FF	FF	Fr-28:02:20-. . . .
$00A0	32	33	3A	35	39	3A	35	39	FF	FF	FF	FF	FF	FF	FF	FF	23:59:59. . . . .
$00B0	53	61	2D	32	39	3A	30	32	3A	32	30	2D	FF	FF	FF	FF	Sa-29:02:20-. . . .
$00C0	30	30	3A	30	30	3A	30	30	FF	FF	FF	FF	FF	FF	FF	FF	00:00:00. . . . .
$00D0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	81	00	00	7F	00	22	. . . . . ▮ . "

Viel Erfolg beim Selbermachen.

Seitenanfang	Sekunden	Uhrzeit-Formate	Datum