DTMF_DECODE_200408

```
;DTMF-Decoder ohne Goertzel-Algorithmus

;### ENDVERSION ###
;
;(c) 2008 Felix Irmscher
;-------------------------------------------------------------


.include "tn26def.inc"

;###  DEFINITION DES AUSGANGS  ###

                                ;portb   (hohe Frquenzen)
.equ  ROT_b       =  6          ;Alarm: Signal zu stark/schwach!

;###  FÜR OBERTONERKENNUNG  ###

.equ  F1209_H     =  96         ;errechnet gem. og. Formel
.equ  F1209_L     =  237

.equ  F1336_H     =  87
.equ  F1336_L     =  182

.equ  F1477_H     =  79
.equ  F1477_L     =  87

.equ  HF_NOT      =  0          ;Kodierung von High_Freq
.equ  HF_1209     =  1
.equ  HF_1336     =  2
.equ  HF_1477     =  3

.equ  Toleranz_H  =  1          ;für 16bit-Toleranzwert (m:k)
.equ  Toleranz_L  =  125        ;(Toleranz_H x 256 + Toleranz_L)

.equ  Tol_Ref     =  35         ;für Tastenvergleiche

;###  KODIERUNG FÜR TASTE  ###

.equ  EINS        =  1
.equ  ZWEI        =  2
.equ  DREI        =  3
.equ  VIER        =  4
.equ  FUENF       =  5
.equ  SECHS       =  6
.equ  SIEBEN      =  7
.equ  ACHT        =  8
.equ  NEUN        =  9
.equ  STERN       =  10
.equ  NULL        =  11
.equ  RAUTE       =  12

;###  VARIABLEN-DEFINITIONEN  ###

.def  Sicher_1    =  r0         ;für U-Programme
.def  Sicher_2    =  r1

.def  Sicher_ITR1 =  r2         ;für ITR's
.def  Sicher_ITR2 =  r3

.def  Richtung    =  r4         ;1 --> Kurve geht hoch
                                ;0 --> Kurve geht runter
.def  Max         =  r5
```

```
.def  Min         =  r6
.def  ADC_H       =  r7       ;ADC
.def  ADC_H_alt   =  r8       ;ADC von Vormessung

.def  uSec_H      =  r9       ;uSec_H:L als 16bit-Zähler
.def  uSec_L      =  r10
.def  uSec_H_ITR  =  r11      ;dauernder High-Byte-Zähler

.def  High_Freq   =  r12      ;enthält kodierten Obertonwert
.def  Faktor      =  r13      ;Zykluszahl Obertonerkennung
.def  Taste       =  r14      ;enthält kodierten Tastenwert



.def  i           =  r16      ;Hilfsregister
.def  j           =  r17
.def  k           =  r18
.def  m           =  r19
.def  n           =  r20
.def  p           =  r21
.def  q           =  r22
.def  r           =  r23

.def  BergZahl    =  r24      ;Zählt die Wellenberge



;für YH:YL  =  r29:r28  reserviert!!!
;für ZH:ZL  =  r31:r30  reserviert!!!


;############################################################


rjmp ANFANG

.org  0x0006                  ;Timer0 overflow
      rjmp  TIM0_OVF
.org  0x000B                  ;ADC Conversion complete
      rjmp  ADC_COMPLETE


ANFANG:

cli

ldi   i,LOW(RAMEND)           ;setze Stackpointer
out   SP,i

;PB4 und PB5 frei für Quarz!!

;port b
sbi   ddrb,ROT_b              ;LED für Signalstärke
sbi   ddrb,3                  ;für LED-Segmente
sbi   ddrb,4
sbi   ddrb,5

;porta
sbi   ddra,0                  ;für LED-Segmente
sbi   ddra,1
sbi   ddra,2
sbi   ddra,4

rcall ADC_START               ;ADC (free running) einschalten
```

```
rcall RESTART_uSEC           ;Zähler an!

sei


LOOP:                        ;eigentliches Hauptprogramm

   nop

rjmp  LOOP



;============================================================
ADC_START:

   in    Sicher_1,SREG

         ;76543210
   ldi   i,0b11101010
   out   ADCSRA,i            ;b7=1(ADEN) :ADC enable ein !
                             ;b6=1(ADSC) :Start first Conv.
                             ;b5=1(ADFR) :free running ein!
                             ;b4=0(ADIF) :ADC-ITR-Flag(read)
                             ;b3=1(ADIE) :ADC-ITR-Enable ein!

                             ;b2..0=010  :1MHz/4=250kHz


         ;76543210
   ldi   i,0b00110111
   out   ADMUX,i             ;b7=0 & b6=0: AVCC 5,0V stab.
                             ;b5(ADLAR)=1: left adjust AN!
                             ;b4..0=10111:
                             ;  ADC6(PA7)=pos. diff. input x20
                             ;  ADC5(PA6)=neg. diff. input x20

   out   SREG,Sicher_1
ret
;------------------------------------------------------------
ADC_COMPLETE:
                             ;Übergabevariable: ADCH
   in    Sicher_ITR1,SREG

   in    uSec_L,TCNT0        ;aktuelle Werte sichern
   mov   uSec_H,uSec_H_ITR   ;(uSec_H dient als Puffer)

   in    ADC_H,ADCH          ;HighByte = ADCH
                             ;U[V] = (Vin x 1024) / Vref
   sei                       ;wieder an wegen Timer0!


SIGNAL_TEST:                 ;zur Arbeitspunkteinstellung ..

   ldi   i,255               ;ADC_H = 255 ?
   cp    ADC_H,i
   brne  NICHT_UEBER
   sbi   portb,ROT_b         ;Rote LED: Signal zu hoch (=255)!
NICHT_UEBER:
   ldi   i,2
   cp    ADC_H,i
   brsh  NICHT_UNTER
   sbi   portb,ROT_b         ;Rote LED: Signal zu tief (<2)!
NICHT_UNTER:

;RICHTUNGS-TEST   (HOCH/RUNTER)
```

```
   tst   Richtung
   brne  HOCH

RUNTER:
   cp    ADC_H,Min
   brsh  TAL_TEST
                              ;Min-Wert nach unten hin
   mov   Min,ADC_H            ;aktualisieren

TAL_TEST:
   cp    ADC_H_alt,ADC_H
   brlo  TAL                  ;dh. Tal erkannt
   mov   ADC_H_alt,ADC_H
   rjmp  ENDE_ADC_RELAIS

TAL:
   clr   Richtung             ;Richtung <- 1 (=hoch)
   inc   Richtung
                              ;früher: PIEZO an!
   mov   ADC_H_alt,ADC_H


;SPEICHERUNG TAL-WERT im SRAM:

   st    Y+,ADC_H             ;(Y) <- ADC_H, Y <- Y + 1


   rjmp  ENDE_ADC_RELAIS

;-----

HOCH:
   cp    Max,ADC_H
   brsh  BERG_TEST
                              ;Max-Wert nach oben hin
   mov   Max,ADC_H            ;aktualisieren

BERG_TEST:
   cp    ADC_H,ADC_H_alt
   brlo  BERG                 ;dh. Berg erkannt
   mov   ADC_H_alt,ADC_H
   rjmp  ENDE_ADC_RELAIS

BERG:
   clr   Richtung             ;Richtung <- 0 (=runter)
                              ;früher:PIEZO aus!
   mov   ADC_H_alt,ADC_H

;BERG-ZAHL-TEST:

   cbi   portb,ROT_b          ;Rote LED aus (Signalstärke)!
   inc   BergZahl             ;BergZahl <- BergZahl+1


;SPEICHERUNG BERG-WERT im SRAM:

   st    Y+,ADC_H             ;(Y) <- ADC_H, Y <- Y + 1


;SIND 30 BERGE GEZÄHLT?
   cpi   BergZahl,30          ;dh. 30 Berge gezählt
   brlo  ENDE_ADC_RELAIS      ;weiter, wenn zu wenig Berge


;30 BERGE SIND GEZÄHLT!
   cli
```

```
;TIMER0 AUSSCHALTEN!
            ;76543210
   ldi   i,0b00000000         ;Timer0 stop!
   out   TCCR0,i

            ;76543210
   ldi   i,0b00000000         ;Timer0 OVF ITR disable!
   out   TIMSK,i

;-------------------------
rjmp  RELAIS_ENDE             ;Verkürzung für relative branch
   ENDE_ADC_RELAIS:
   rjmp  ENDE_ADC_COMPLETE
RELAIS_ENDE:
;-------------------------

;LEDS FÜR DISPLAY WIEDER AUS!

   cbi   portb,3
   cbi   portb,4
   cbi   portb,5

   cbi   porta,0
   cbi   porta,1
   cbi   porta,2
   cbi   porta,4


;##### VERGLEICH MIT OBERTON #####

   ldi   k,Toleranz_L         ;Toleranzwert einlesen
   ldi   m,Toleranz_H

   ldi   ZL,LOW(HIGH_TONE*2)  ;Adresse von Label HIGH_TONE
   ldi   ZH,HIGH(HIGH_TONE*2)

LOOP_LED:
   lpm   High_Freq,Z          ;Kodierter Oberton
   tst   High_Freq            ;bei 0 => keine Freq. gefunden!
   breq  VAR_RESET
   adiw  ZL,1
   lpm   i,Z                  ;Low-Byte Referenzfrequenz
   adiw  ZL,1
   lpm   j,Z                  ;High-Byte Referenzfrequenz
   adiw  ZL,1
   lpm   n,Z                  ;LED-Farbe (von Vorversion..)
   adiw  ZL,1

   rcall FREQ_CHECK

   tst   High_Freq            ;High_Freq von U-Prg. ggf. geändert!
   breq  LOOP_LED             ;bei 0 => nächster Freq.-Test

;OBERTON GEFUNDEN:

   rcall SIGNAL_CHECK         ;Berechnung "Interferenz-Muster"

   rcall CHECK_TASTE          ;Kontrolle auf 2. Frequenz

VAR_RESET:

   rcall RESTART_uSEC         ;Timer0 wieder hochzählen


ENDE_ADC_COMPLETE:
```

```
   out   SREG,Sicher_ITR1

reti

;-----------------------------------------------------------
TIM0_OVF:                     ;Zählt High-Byte von uSec_H:L hoch

   in    Sicher_ITR2,SREG

   inc   uSec_H_ITR

   out   SREG,Sicher_ITR2
reti
;-----------------------------------------------------------
RESTART_uSEC:
                              ;bereitet für neue Zählung von
                              ;X Frequenz-Bergen vor
   in    Sicher_1,SREG

   clr   ADC_H_alt            ;Reset ADC
   clr   Richtung
   clr   BergZahl

   ldi   YL,LOW(0x0061)       ;Reset Berg-Wert-Speicher im SRAM
   ldi   YH,HIGH(0x0061)

   ldi   i,1                  ;Reset Timer-Werte
   out   TCNT0,i
   clr   uSec_L
   clr   uSec_H
   clr   uSec_H_ITR

   clr   Max                  ;Max <- 0
   ldi   i,255                ;Min <- 255
   mov   Min,i

                              ;TIMER0 WIEDER STARTEN!
            ;76543210
   ldi   i,0b00000001         ;bit2..0=001 0=> Takt=CK(1MHz)
   out   TCCR0,i

            ;76543210
   ldi   i,0b00000010         ;bit1 =1 => Timer0 OVF ITR enable
   out   TIMSK,i

   out   SREG,Sicher_1
ret
;-----------------------------------------------------------
```

```
FREQ_CHECK:
                              ;Übergabevariablen:
                              ;rein:   FREQUENZVORGABE:
                              ;        j(High-Byte):i(Low-Byte)
                              ;        TOLERANZWERT:
                              ;        m(High-Byte):k(Low-Byte)
                              ;raus:   High_Freq
   in    Sicher_1,SREG


   add   i,k                  ;[j:i] <- [j:i] + [m:k]
   adc   j,m

   cp    uSec_L,i
   cpc   uSec_H,j
   brsh  NO_FREQ

   sub   i,k                  ;[j:i] <- [j:i] - [m:k]
   sbc   j,m

   sub   i,k                  ;[j:i] <- [j:i] - [m:k]
   sbc   j,m

   cp    uSec_L,i
   cpc   uSec_H,j
   brlo  NO_FREQ

;FREQUENZ GEFUNDEN!
   rjmp  FREQ_CHECK_ENDE

NO_FREQ:
   clr   High_Freq

FREQ_CHECK_ENDE:

   out   SREG,Sicher_1
ret
;-----------------------------------------------------------
HIGH_TONE:

.db   HF_1209,F1209_L,F1209_H,32 ;8,16,32 codieren die LEDs
.db   HF_1336,F1336_L,F1336_H,16 ;für Befehl <OR PORTB,N>
.db   HF_1477,F1477_L,F1477_H,8
.db   0,0
;-----------------------------------------------------------
SIGNAL_CHECK:
                              ;Signal-Modifikationen
   in    Sicher_1,SREG

;AC-ANTEIL SELEKTIEREN:       ;Signal wird von DC-Anteil befreit
   clr   j                    ;übrig bleibt AC-Anteil
   clr   Max                  ;Max neu definiert: Max. AC-Anteil
   ldi   YL,LOW(0x0061)
   ldi   YH,HIGH(0x0061)

LOOP_SUB_MIN:

   inc   j
   ld    i,Y                  ;i <- (Y)
   sub   i,Min                ;i <- i-Min
   cp    Max,i
   brsh  SAVE_AC
   mov   Max,i                ;Max <- i
 SAVE_AC:
   st    Y+,i                 ;(Y) <- i, Y <- Y+1
```

```
   cpi   j,59
brlo  LOOP_SUB_MIN

;### MAX MINDESTENS 100! ###

LOOP_PRE_ZOOM:
   ldi   i,100               ;um den Faktor mit 8bit zu reali-
   cp    Max,i               ;sieren, dürfen nur Werte zwischen
   brsh  FIND_FAKTOR         ;100 .. 255 verwendet werden!

   clr   j
   ldi   YL,LOW(0x0061)
   ldi   YH,HIGH(0x0061)
   ADD   Max,Max             ;Max <- 2 x Max

 PRE_ZOOM:
   inc   j
   ld    i,Y                 ;i <- (Y)
   add   i,i                 ;i <- i+i          (= 2xi   )
   st    Y+,i                ;(Y) <- (Y)+i      (= 2x (Y))
   cpi   j,59
 brlo PRE_ZOOM
rjmp  LOOP_PRE_ZOOM

;### WERTE AUF 0..255 AUFWEITEN ###

FIND_FAKTOR:                 ;mit Faktor = 25500 / Max
   ldi   j,0x63
   ldi   i,0x9C              ;[j:i] <- 25500 (255 x 100)
   clr   Faktor
   clr   m                   ;für [m:Max]

LOOP_FAKTOR:                 ;Faktor <- 0x639C / Max
   cp    i,Max
   cpc   j,m                 ;[j:i] < [m:Max] ?
   brlo  ENDE_LOOP_FAKTOR
   inc   Faktor
   sub   i,Max
   sbc   j,m                 ;[j:i] <- [j:i] - [m:Max]

rjmp LOOP_FAKTOR
ENDE_LOOP_FAKTOR:

                             ;Faktor ist gefunden
   ldi   YL,LOW(0x0061)
   ldi   YH,HIGH(0x0061)
   clr   p

FAKTOR_MULTIP:
   clr   i                   ;zuerst (Y) x Faktor rechnen
   clr   j
   clr   k
   clr   n

   inc   p
   cpi   p,60                ;nur 59 mal durchlaufen!
   brsh  ENDE_FAKTOR_MULTIP

   ld    m,Y                 ;m <- (Y)

 LOOP_MULTIP:                ;Einfaches Multiplikationsprogramm:
   cp    k,Faktor            ;[j:i] <- m x Faktor
   brsh  FAKTOR_DIVIS
   inc   k
   add   i,m                 ;[j:i] <- [j:i] + m
   adc   j,n
```

```
  rjmp   LOOP_MULTIP


 FAKTOR_DIVIS:                ;Einfaches Divisionsprogramm:
   ldi   m,100                ;[j:i] <- [j:i] / 100 ([n:m])
   clr   n
   clr   k

 LOOP_DIVIS:
   cpi   j,0
   brne  DIVIS_1
   cpi   i,100
   brsh  DIVIS_1
   rjmp  STORE_DIVIS          ;wenn: [j:i] < 100 !

 DIVIS_1:
   inc   k                    ;k = Ergebnis der Division
   sub   i,m                  ;[j:i] <- [j:i] - 100
   sbc   j,n
 rjmp LOOP_DIVIS

 STORE_DIVIS:
   st    Y+,k                 ;(Y) <- k, Y <- Y + 1

rjmp  FAKTOR_MULTIP



ENDE_FAKTOR_MULTIP:


out   SREG,Sicher_1

ret
;-------------------------------------------------------------
CHECK_TASTE:

   in Sicher_1,SREG

   clr   j                    ;gibt die dekodierte Zeile an

LOOP_ZEILE:
   inc   j
   cpi   j,5
   brlo  ENDE_LOOP_ZEILE
   rjmp  ENDE_CHECK_TASTE     ;alle Zeilen durchgearbeitet
                              ;  => keine Taste !
ENDE_LOOP_ZEILE:
   clr   n                    ;n bezieht sich auf die Y. Stelle

LOOP_Y:
   rcall SET_Z                ;Z-Label auf Anfangswert setzen !

   inc   n
   cpi   n,50                 ;anstatt 60
   brsh  LOOP_ZEILE           ;diese Taste nicht gefunden
                              ; => nächste Zeile abfragen
```

```
;zunächst passende SRAM-Start-Adresse (=n) suchen

   clr   m

   ldi   YL,LOW(0x0060)
   ldi   YH,HIGH(0x0060)

SEARCH_SRAM:
   inc   m                    ;m <- m+1
   adiw  YL,1                 ;Y <- Y+1
   cp    m,n
   breq  LOOP_VGL
rjmp  SEARCH_SRAM

LOOP_VGL:
   lpm   p,Z                  ;p <- (Z)
   adiw  ZL,1                 ;Z <- Z+1
   cpi   p,0
   breq  FOUND                ;Übereinstimmung bis zum letzten
                              ;Wert => Taste erkannt !

   ld    k,Y+                 ;k <- (Y), Y <- Y+1

ZU_TIEF:
   ldi   m,Tol_Ref
   cp    m,p                  ;untere Grenze ausgeschaltet
   brsh  ZU_HOCH
   sub   p,m                  ;p <- p - Tol_Ref
   cp    k,p
   brlo  LOOP_Y               ;Y-Stelle um 1 weiterrücken

   add   p,m                  ;p auf Anfangswert zurückstellen

ZU_HOCH:
   add   p,m                  ;p <- p + Tol_Ref
   brcs  LOOP_VGL             ;Addition löst Carry aus
                              ;=> obere Grenze ausgeschaltet
   cp    p,k
   brlo  LOOP_Y               ;Y-Stelle um 1 weiterrücken

;(Z) = (Y) +/- TOL_Ref:
   rjmp  LOOP_VGL

;------------------

FOUND:
   in    i,porta              ;LED-Segmente an!
   or    i,q
   out   porta,i

   in    i,portb
   or    i,r
   out   portb,i

ENDE_CHECK_TASTE:

   out SREG,Sicher_1

ret

;-------------------------------------------------------------
```

```
SET_Z:

   in    Sicher_2,SREG        ;High_Freq -> Spalte 1..3 (3 Stk.)
                              ;        j -> Zeile  1..4 (4 Stk.)
;Spalte prüfen                ;Hilfsvariable:   q
   mov   q,High_Freq          ;Ausgabe:         ZH:ZL
                              ;                 q
SPALTE_1209:                  ;                 Taste: EINS, ..
   cpi   q,HF_1209
   brne  SPALTE_1336
   rjmp  F_1209_1

SPALTE_1336:
   cpi   q,HF_1336
   brne  SPALTE_1477
   rjmp  F_1336_1

SPALTE_1477:
   rjmp  F_1477_1


F_1209_1:
   cpi   j,1
   brne  F_1209_2
   ldi   ZL,LOW(Label_1*2)
   ldi   ZH,HIGH(Label_1*2)
   ldi   q,EINS
   mov   Taste,q
   ldi   q,0b00010100
   ldi   r,0b00000000
   rjmp  ENDE_SET_Z

F_1209_2:
   cpi   j,2
   brne  F_1209_3
   ldi   ZL,LOW(Label_4*2)
   ldi   ZH,HIGH(Label_4*2)
   ldi   q,VIER
   mov   Taste,q
   ldi   q,0b00010101
   ldi   r,0b00010000
   rjmp  ENDE_SET_Z

F_1209_3:
   cpi   j,3
   brne  F_1209_4
   ldi   ZL,LOW(Label_7*2)
   ldi   ZH,HIGH(Label_7*2)
   ldi   q,SIEBEN
   mov   Taste,q
   ldi   q,0b00010100
   ldi   r,0b00001000
   rjmp  ENDE_SET_Z


F_1209_4:
   ldi   ZL,LOW(Label_STERN*2)
   ldi   ZH,HIGH(Label_STERN*2)
   ldi   q,STERN
   mov   Taste,q
   ldi   q,0b00010111
   ldi   r,0b00010000
   rjmp  ENDE_SET_Z
```

```
F_1336_1:
   cpi   j,1
   brne  F_1336_2
   ldi   ZL,LOW(Label_2*2)
   ldi   ZH,HIGH(Label_2*2)
   ldi   q,ZWEI
   mov   Taste,q
   ldi   q,0b00000110
   ldi   r,0b00111000
   rjmp  ENDE_SET_Z

F_1336_2:
   cpi   j,2
   brne  F_1336_3
   ldi   ZL,LOW(Label_5*2)
   ldi   ZH,HIGH(Label_5*2)
   ldi   q,FUENF
   mov   Taste,q
   ldi   q,0b00010001
   ldi   r,0b00111000
   rjmp  ENDE_SET_Z

F_1336_3:
   cpi   j,3
   brne  F_1336_4
   ldi   ZL,LOW(Label_8*2)
   ldi   ZH,HIGH(Label_8*2)
   ldi   q,ACHT
   mov   Taste,q
   ldi   q,0b00010111
   ldi   r,0b00111000
   rjmp  ENDE_SET_Z

F_1336_4:
   ldi   ZL,LOW(Label_NULL*2)
   ldi   ZH,HIGH(Label_NULL*2)
   ldi   q,NULL
   mov   Taste,q
   ldi   q,0b00010111
   ldi   r,0b00101000
   rjmp  ENDE_SET_Z


F_1477_1:
   cpi   j,1
   brne  F_1477_2
   ldi   ZL,LOW(Label_3*2)
   ldi   ZH,HIGH(Label_3*2)
   ldi   q,DREI
   mov   Taste,q
   ldi   q,0b00010100
   ldi   r,0b00111000
   rjmp  ENDE_SET_Z

F_1477_2:
   cpi   j,2
   brne  F_1477_3
   ldi   ZL,LOW(Label_6*2)
   ldi   ZH,HIGH(Label_6*2)
   ldi   q,SECHS
   mov   Taste,q
   ldi   q,0b00010011
   ldi   r,0b00111000
   rjmp  ENDE_SET_Z

F_1477_3:
```

```
   cpi   j,3
   brne  F_1477_4
   ldi   ZL,LOW(Label_9*2)
   ldi   ZH,HIGH(Label_9*2)
   ldi   q,NEUN
   mov   Taste,q
   ldi   q,0b00010101
   ldi   r,0b00111000
   rjmp  ENDE_SET_Z

F_1477_4:
   ldi   ZL,LOW(Label_RAUTE*2)
   ldi   ZH,HIGH(Label_RAUTE*2)
   ldi   q,RAUTE
   mov   Taste,q
   ldi   q,0b00010010
   ldi   r,0b00110000
   rjmp  ENDE_SET_Z

ENDE_SET_Z:

   out   SREG,Sicher_2

ret


;============================================================
LABEL_1:

.db   250,60,129,89,232,19,175,125,191,18,0,0

;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
LABEL_2:

.db   10,200,140,160,20,210,130,145,30,230
.db   120,130,40,240,90,120,75,250,0,0

;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
LABEL_3:

.db   215,25,172,88,236,33,161,73,253,37,149,55,236,51
.db   150,43,246,72,157,38,0,0

;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
LABEL_4:

.db   240,69,120,106,215,8,216,106,122,71
.db   237,16,187,137,148,44,0,0

;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
LABEL_5:

.db   246,58,117,103,224,17,172,144,174,18
.db   229,109,117,50,249,0

;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
LABEL_6:

.db   237,19,162,108,229,15,175,127,218,10,194,123,191,8
.db   212,116,181,11,229,116,164,17,239,0

;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
LABEL_7:

.db   224,16,241,67,139,136,181,27,246,27,187,134,137,68,0,0
```

```
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
LABEL_8:

.db   237,28,196,143,147,60,244,48,164,149
.db   179,38,243,75,126,124,208,23,0,0

;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
LABEL_9:

.db   7,199,102,185,23,238,76,149,64,254,28,168,94,0

;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
LABEL_NULL:

.db   103,122,91,206,8,234,55,145,131,181
.db   41,253,32,192,126,139,66,0

;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
LABEL_STERN:

.db   66,146,140,156,65,228,10,250,45,190,110,120,110,200,0,0

;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
LABEL_RAUTE:

.db   254,34,181,104,193,15,241,48,163,83,224,19,0,0

;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```