Sommario

AN007 - Esempi applicativi del device HMI2

In questo capitolo verranno analizzati alcuni esempi di programmazione utili per poter eseguire semplici funzionalità con l'ausilio del device HMI2. Come piattaforma hardware utilizzeremo un D221, ma l'applicabilità di tali esempi, con eventuali piccole modifiche, è estesa a tutti gli hardware microQMove. È buona regola, prima di utilizzare questo device, definire una serie di valori costanti (nella sezione CONST della unit di configurazione dell'applicativo Qcl) da inserire nel modulo di configurazione per rendere migliore la leggibilità e la manutenzione dell'applicativo sviluppato.

```
;*******************************************************
;Definizione dei valori associati ai tasti
;*******************************************************
CONST

KEY_ENTER 1           ; tasto enter
KEY_CLEAR 8           ; tasto clear
KEY_PLUS  4           ; tasto +
KEY_MINUS 32          ; tasto -
KEY_F     16          ; tasto F

;*******************************************************
;Definizione dei valori associati ai led
;*******************************************************

LED_L1  2             ; led L1
LED_L2  4             ; led L2
LED_L3  8             ; led L3
LED_L4  16            ; led L4
LED_F   512           ; led tasto F
LED_AL  1             ; led ALARM

;*******************************************************
;Definizione dei valori associati ai caratteri display
;*******************************************************

CHAR      35          ; codice display per il carattere <space>
CHAR_0    0           ; codice display per il carattere 0
CHAR_1    1           ; codice display per il carattere 1
CHAR_2    2           ; codice display per il carattere 2
CHAR_3    3           ; codice display per il carattere 3
CHAR_4    4           ; codice display per il carattere 4
CHAR_5    5           ; codice display per il carattere 5
CHAR_6    6           ; codice display per il carattere 6
CHAR_7    7           ; codice display per il carattere 7
CHAR_8    8           ; codice display per il carattere 8
CHAR_9    9           ; codice display per il carattere 9
CHAR_A    10          ; codice display per il carattere a
CHAR_B    11          ; codice display per il carattere b
CHAR_C    12          ; codice display per il carattere c
CHAR_D    13          ; codice display per il carattere d
CHAR_E    14          ; codice display per il carattere e
CHAR_F    15          ; codice display per il carattere f
CHAR_G    16          ; codice display per il carattere g
CHAR_H    17          ; codice display per il carattere h
CHAR_I    18          ; codice display per il carattere i
CHAR_J    28          ; codice display per il carattere j
CHAR_K    40          ; codice display per il carattere k
CHAR_L    19          ; codice display per il carattere l
CHAR_M    43          ; codice display per il carattere m
CHAR_N    20          ; codice display per il carattere n
CHAR_O    21          ; codice display per il carattere o
CHAR_P    22          ; codice display per il carattere p
CHAR_Q    23          ; codice display per il carattere q
CHAR_R    24          ; codice display per il carattere r
CHAR_S    5           ; codice display per il carattere s
CHAR_T    25          ; codice display per il carattere t
CHAR_U    26          ; codice display per il carattere u
CHAR_V    34          ; codice display per il carattere v
CHAR_W    28          ; codice display per il carattere w
CHAR_Y    27          ; codice display per il carattere y
CHAR_UP   40          ; codice display per il carattere upper segment
CHAR_CENTER   33      ; codice display per il carattere central segment
CHAR_LOWER    36      ; codice display per il carattere lower segment
CHAR_UPCEN    39      ; codice display per il carattere upper & middle segments
CHAR_LOWCEN   37      ; codice display per il carattere lower & middle segments
CHAR_LOWUP    42      ; codice display per il carattere lower & upper segments
CHAR_LOWUPCE  38      ; codice display per il carattere lower & upper & middle  segments
CHAR_NONE     0       ; codice display per il carattere NONE (none char showed)
CHAR_POINT    &H80    ; bit che abilita il punto decimale
```

Questa metodologia è importante applicarla a tutti i parametri formati da campi di bit come ad esempio *scflags* o *deflags*; in tal caso definiremo ad esempio:

```
SCRA_ENABLE   1       ; Bit abilitazione visualizzazione screenA
SCRB_ENABLE   2       ; Bit abilitazione visualizzazione screenB
SCRC_ENABLE   4       ; Bit abilitazione visualizzazione screenC
SCRA_DISSIGN  8       ; Bit disabilitazione segno screenA
SCRB_DISSIGN  16      ; Bit disabilitazione segno screenB
SCRC_DISSIGN  32      ; Bit disabilitazione segno screenC
SCRA_DISLZB   64      ; Bit disabilitazione Leading zero blank (LZB) screenA
SCRB_DISLZB   128     ; Bit disabilitazione Leading zero blank (LZB) screenB
SCRC_DISLZB   256     ; Bit disabilitazione Leading zero blank (LZB) screenC

DE_ENABLE     1       ; Bit abilitazione dataentry
DE_DISSIGN    4       ; Bit disabilitazione segno nel data entry
DE_ENALIM     16      ; Bit abilitazione controllo limiti
```

Inseriamo successivamente il device HMI2 con tempo di campionamento di 5ms nella apposita sezione:

```
INTDEVICE

dvHMI       HMI2     5
```

Negli esempi seguenti il nome del device sarà sempre dvHMI.

Attendere la pressione di un tasto o più tasti per un certo tempo

Si voglia scrivere un programma Qcl che attenda la pressione del tasto F per eseguire una subroutine. È sufficiente verificare che il parametro *key* abbia il bit relativo al tasto F attivo:

```
MAIN:
  IF ( dvHMI:key ANDB KEY_F )
    CALL MyFUNC
  ENDIF
  WAIT 1
  JUMP MAIN

SUB MyFUNC
  ;-------------------------
  ;codice subroutine
  ;-------------------------
ENDSUB
```

questo codice non assicura che sia premuto solamente il tasto F: la funzione MyFUNC potrebbe essere chiamata anche se fossero premuti assieme al tasto F anche altri tasti. Per assicurare l'esclusività della pressione di F il codice diventa:

```
IF ( dvHMI:key EQ KEY_F )
    CALL MyFUNC
ENDIF
```

Si voglia ora scrivere un codice che attenda la pressione contemporarea dei tasti CLEAR ed ENTER per almeno 2 secondi:

```
IF ( dvHMI:key EQ (KEY_ENTER+KEY_CLEAR) )
    IF tm01:remain EQ 0    ;verifica timer scaduto
        CALL MyFUNC
    ENDIF
ELSE
    tm01=2000           ;timer viene ricaricato
ENDIF
```

Creare una visualizzazione ricorsiva

Si voglia scrivere un programma Qcl che abiliti una visualizzazione ricorsiva sui 4 display più a sinistra con segno e 2 cifre decimali. Decidiamo per comodità di utilizzare la screenA. Dobbiamo innanzitutto impostare il numero di caratteri che vogliamo visualizzare tenendo presente che il segno occupa un carattere; possiamo quindi dire che il numero di caratteri rappresenta il numero di digit del display che vengono occupati e manipolati dalla visualizzazione stessa. I valori massimo e minimo che potremo visualizzare saranno rispettivamente 9999 e -999. Se il dato da visualizzare è inferiore a tale valore minimo oppure superiore a tale valore massimo, il display visualizzerà i caratteri di out of range $$$$.
Imposteremo quindi:

```
dvHMI:ncharA = 4
```

Metteremo la nostra visualizzazione nei display più a sinistra impostando il valore di offset pari a:

```
dvHMI:offsA = dvHMI:numdis - 4
```

Settiamo la posizione del punto decimale a 2:

```
dvHMI:decptA = 2
```

Abilitiamo la visualizzazione ricorsiva screenA settando il corrispondente bit di abilitazione della variabile scflags:

```
dvHMI:scflags = SCRA_ENABLE
```

Eseguendo l'istruzione di cui sopra abbiamo automaticamente disabilitato le altre due visualizzazioni ricorsive ed abbiamo abilitato la visualizzazione del segno sulla screenA. Nel caso avessimo voluto preservare gli stati delle altre visualizzazioni screenB e screenC avremmo dovuto scrivere:

```
dvHMI:scflags = dvHMI:scflags ORB SCRA_ENABLE    ;abilitazione screenA
dvHMI:scflags = dvHMI:scflags ANDB SCRA_DISSIGN  ;abilitazione segno screenA
```

Infine è sufficiente aggiornare la variabile screenA con il valore che vogliamo visualizzare e normalmente contenuto in un'altra variabile del nostro programma (nell'esempio supponiamo di utilizzare una variabile con il nome *count*):

```
dvHMI:screenA = count
```

L'operazione di aggiornamento di screenA dovrà essere continuamente eseguita dal nostro programma con il refresh rate più opportuno in ragione della funzionalità che il programmatore ha previsto per tale variabile.

Creare una visualizzazione di testo

Si voglia scrivere un programma Qcl che scriva sul display “HELLO” allineato a destra. Per fare ciò è sufficiente impostare nelle variabili associate ai digit del display il codice del carattere che si vuole visualizzare. Avremo quindi:

```
;Stampa "HELLO"
dvHMI:dis6 = CHAR_
dvHMI:dis5 = CHAR_
dvHMI:dis4 = CHAR_H
```

```
dvHMI:dis3 = CHAR_E
dvHMI:dis2 = CHAR_L
dvHMI:dis1 = CHAR_L
dvHMI:dis0 = CHAR_O
```

Nota:
Per poter funzionare correttamente, non devono essere attive visualizzazioni ricorsive che sovrascriverebbero tutti o parte dei digit interessati dal nostro “HELLO”. Controllare quindi che nel parametro *scflags* i bits 0,1 e 2 siano a 0 oppure forzarli a tale valore.

Creare più visualizzazioni ricorsive miste a visualizzazioni di testo

Si voglia creare una visualizzazione composta da due testi fissi e due valori ricorsivi. A titolo di esempio si pensi di visualizzare un tempo espresso in secondi ed un numero di programma. La visualizzazione voluta potrebbe essere: “t51 Pr2” dove “t” indica il tempo, “51” è il valore del tempo, “Pr” è un testo che indica il programma, “2” indica il numero del programma. Innanzitutto stampiamo i testi:

```
dvHMI:dis6 = CHAR_T
dvHMI:dis3 = CHAR_
dvHMI:dis2 = CHAR_P
dvHMI:dis1 = CHAR_R
```

Impostiamo poi i dati per la visualizzazione numerica del tempo tramite la screenA.

```
dvHMI:ncharA = 2
dvHMI:offsA = 4
dvHMI:decptA = 1
dvHMI:scflags = dvHMI:scflags ORB SCRA_DISSIGN
```

Impostiamo poi i dati per la visualizzazione numerica del programma tramite la screenB.

```
dvHMI:ncharB = 1
dvHMI:offsB = 0
dvHMI:decptB = 0
dvHMI:scflags = dvHMI:scflags ORB SCRB_DISSIGN
```

Abilitiamo le due visualizzazioni:

```
dvHMI:scflags = dvHMI:scflags ORB SCRA_ENABLE ORB SCRB_ENABLE
```

Poi ricorsivamente aggiorneremo i dati della visualizzazione:

```
dvHMI:screenA = glTime
dvHMI:screenB = glProgram
```

Creare una introduzione dati semplice

Si voglia scrivere un programma Qcl che permetta all'utente di introdurre un valore su una variabile, ad esempio una utilizzata per memorizzare un conta pezzi. Innanzitutto dovremo dichiarare tale variabile, ad esempio *cntPieces* nell'apposita sezione della unit di configurazione. Supponiamo che si voglia visualizzare il messaggio “CP” sulla parte sinistra del display ad indicare l'introduzione del conta pezzi, e che il valore da introdurre sia di 4 caratteri e posizionato sulla parte più a destra del display. Il data entry occuperà i display dis0, dis1, dis2, dis3 mentre il messaggio verrà scritto in dis5 e dis6.

```
dvHMI:dis6 = CHAR_C
dvHMI:dis5 = CHAR_P
dvHMI:deoffs = 0
dvHMI:denchar = 4
```

La posizione del punto decimale sarà ovviamente posta a 0 ed eseguiremo la copia del valore del conta pezzi attuale nel parametro *devalue* per far sì che all'ingresso dell'introduzione dati compaia tale valore sul display.

```
dvHMI:dedecpt = 0
dvHMI:devalue = cntPieces
```

Infine abiliteremo l'introduzione dati con l'apposito flag, disabiliteremo il segno (un conta pezzi non potrà essere negativo) e attiveremo la procedura d'introduzione con il comando *DATAENTRY*:

```
dvHMI:deflags = DE_ENABLE ORB DE_DISSIGN
DATAENTRY dvHMI
```

A questo punto comincerà a lampeggiare sul display il digit più significativo del valore di *cntPieces* e sarà necessario attendere che l'utente introduca il dato e lo confermi con il tasto ENTER. Successivamente si dovrà leggere il dato introdotto (nel parametro *devalue*) e copiarlo nella nostra variabile del conta pezzi *cntPieces*. Lo stato *st_dentry* ci permette di sapere se il data entry è attivo quindi attendiamo che questo vada a 0 per poi fare le operazioni di copia:

```
WHILE (dvHMI:st_dentry)
    WAIT 1
ENDWHILE
cntPieces = dvHMI:devalue
```

A questo punto la variabile *cntPieces* è aggiornata con il valore introdotto dall'utente.

Creare una introduzione dati complessa

Si voglia scrivere un programma Qcl che permetta all'utente di introdurre un valore su una variabile, come già fatto nell'esempio precedente, ma con le seguenti caratteristiche aggiuntive:

- controllare che il dato sia compreso tra 1 e 1000 ed in caso contrario visualizzare “Error” per 1 secondo e ripetere l'introduzione dati
- se viene premuto il tasto F si esca dall'introduzione dati senza memorizzare il dato introdotto e venga stampato per un secondo il messaggio “Exit F”
- se viene premuto il tasto CLEAR si esca dall'introduzione dati senza memorizzare il dato introdotto e venga stampato per un secondo il messaggio “Exit C”
- stampare per un secondo il messaggio “MOdiFY” se il dato in introduzione è stato modificato

Controllo limiti dato

Per abilitare il controllo dei limiti del dato introdotto è necessario abilitare tale funzionalità ponendo a 1 l'apposito bit del parametro *deflags* ed impostare nei parametri *deuplim* e *delowlim* i valori dei limiti superiore ed inferiore. Rispetto al codice del precedente esempio dovremo aggiungere, prima del comando *DATAENTRY*, le seguenti istruzioni Qcl:

```
dvHMI:deuplim = 1000
dvHMI:delowlim = 1
```

e sostituire l'istruzione di impostazione del parametro deflags con:

```
dvHMI:deflags = DE_ENABLE ORB DE_DISSIGN ORB DE_ENALIM
```

Configurare uno o più tasti di uscita dal data entry

Per abilitare l'uscita dal data entry con un tasto è necessario impostare il parametro *deExKeymask* cioè la maschera per i tasti di uscita. Per abilitare un tasto a funzionare come tasto di uscita dal data entry, è sufficiente attivare il bit corrispondente del parametro sopracitato. Quindi se vogliamo far si che si esca dal data entry con i tasti F e CLEAR serve inserire la seguente istruzione Qcl prima del comando *DATAENTRY*:

```
dvHMI:deExKeymask = KEY_CLEAR  ORB KEY_F
```

Verificare se il dato introdotto è nei limiti

All'uscita dal data entry (quindi con lo stato *st_dentry* = 0), è sufficiente controllare il valore degli stati *st_uplim* e *st_lowlim* per sapere se il dato introdotto è superiore ai limiti impostati. Se *st_uplim* vale 1 significa che il valore introdotto è maggiore del limite superiore, mentre se *st_lowlim* vale 1 significa che il valore introdotto è minore del limite inferiore. Quindi controlleremo tali stati e faremo una chiamata alla subroutine ERROR (che si occuperà di visualizzare il messaggio di errore per 1 secondo) nel caso di superamento dei limiti.

```
;Controllo limiti dati
IF ( dvHMI:st_uplim OR dvHMI:st_lowlim )
    CALL ERROR                  ;stampa messaggio d'errore
    JUMP Dentry                  ;ritorno introduzione datai
ENDIF
```

Controllare il tasto di uscita dal data entry

Controllando il parametro *deExitKey* e gli stati *st_modified* ed *st_exitcmd*, è possibile capire in quale modo si è usciti dal data entry. La seguente tabella riassume le possibili condizioni:

deExitKey	***st_exitcmd***	**Descrizione**
0	0	Uscita con conferma tramite pressione del tasto ENTER o tramite comando *EXITDEC*
0	1	Uscita senza conferma tramite domando *EXITDE*
!=0	X	Uscita senza conferma tramite pressione del tasto identificato dal valore del parametro *deExitKey*

Verificare se il dato è stato modificato

Per verificare se il dato introdotto è stato modificato basta semplicemente controllare lo stato *st_modified*. Esso assume valore 1 se il valore introdotto è diverso dal valore che aveva il parametro devalue prima del comando *DATAENTRY*.
Il programma completo sarà quindi:

```
LABEL0:

    dvHMI:dis6 = CHAR_C
    dvHMI:dis5 = CHAR_P
    dvHMI:dis4 = CHAR_
    dvHMI:deoffs = 0
    dvHMI:denchar = 4
    dvHMI:dedecpt = 0
    dvHMI:devalue = cntPieces
    dvHMI:deuplim = 1000
    dvHMI:delowlim = 1
    dvHMI:deExKeymask = KEY_CLEAR    ORB KEY_F
    dvHMI:deflags = DE_ENABLE ORB DE_DISSIGN ORB DE_ENALIM
    DATAENTRY dvHMI

    WHILE (dvHMI:st_dentry)
        WAIT 1
    ENDWHILE

    IF dvHMI:deExitKey
        ;--Uscita da data entry con tasti d'uscita
        dvHMI:dis6 = CHAR_E
        dvHMI:dis5 = CHAR_H
        dvHMI:dis4 = CHAR_I
        dvHMI:dis3 = CHAR_T
        dvHMI:dis2 = CHAR_
        dvHMI:dis1 = CHAR_
        IF dvHMI:deExitKey EQ KEY_F
            dvHMI:dis0 = CHAR_F         ;Pressione tasto F
        ELSE
            IF dvHMI:deExitKey EQ KEY_CLEAR
                dvHMI:dis0 = CHAR_C     ;Pressione tasto CLEAR
            ENDIF
        ENDIF
    ELSE
        ;--Uscita da data entry con conferma
        ;--Controllo limiti
        IF ( dvHMI:st_uplim OR dvHMI:st_lowlim )
            CALL ERROR                  ;stampa mesaggio d'errore
            JUMP LABEL0                 ;ritorna all'introduzione dati
        ENDIF
        ;--Verifica se i dati sono stati modificati
        IF dvHMI:st_modified
            dvHMI:dis6 = CHAR_     ;stampa messaggio "MODIFY"
            dvHMI:dis5 = CHAR_M
            dvHMI:dis4 = CHAR_O
            dvHMI:dis3 = CHAR_D
            dvHMI:dis2 = CHAR_I
            dvHMI:dis1 = CHAR_F
            dvHMI:dis0 = CHAR_Y
            tm01 = 1000
            WAIT tm01
        ENDIF
        cntPieces = dvHMI:devalue        ;memorizza valore introdotto
    ENDIF

MAIN:
    WAIT 1
    JUMP MAIN


SUB ERROR
    dvHMI:dis6 = CHAR_               ;stampa messaggio "ERROR"
    dvHMI:dis5 = CHAR_E
    dvHMI:dis4 = CHAR_R
    dvHMI:dis3 = CHAR_R
    dvHMI:dis2 = CHAR_O
    dvHMI:dis1 = CHAR_R
    dvHMI:dis0 = CHAR_
    tm01 = 1000
    WAIT tm01
ENDSUB

END
```

Creare una visualizzazione mista non ricorsiva

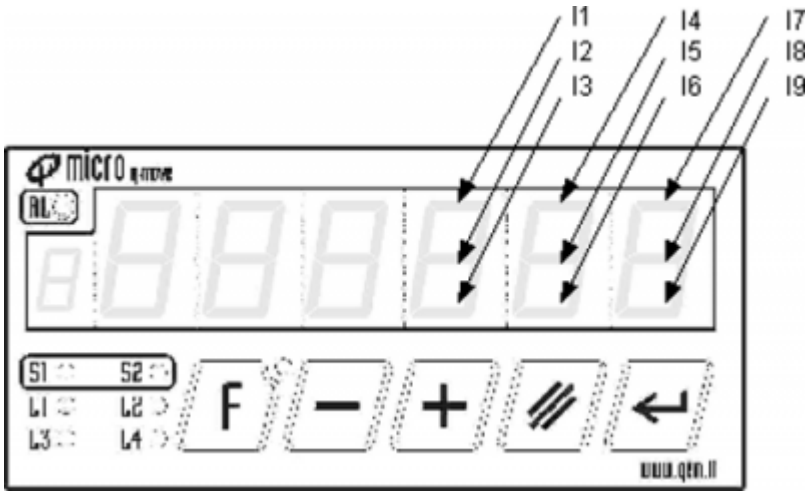
Si voglia creare una visualizzazione di un messaggio composto dalla stringa “Error” e da un numero identificativo dell'errore che appaia quando si verifica un errore, mentre normalmente venga visualizzato, in maniera ricorsiva, il valore di un conteggio. Per realizzare questo, sfruttiamo il funzionamento di sola visualizzazione di un valore numerico presente nella funzionalità del comando *DATAENTRY* ed abilitata impostando a 0 il bit *DE_ENABLE* del parametro *deflags*. Per semplicità, realizzeremo una condizione fittizia di errore tramite la fine di un timer caricato a 5 sec. Come vedremo, sarà importante ricordarsi di disabilitare la visualizzazione ricorsiva prima di visualizzare il messaggio di errore, altrimenti il risultato non sarà quello che ci si aspetta.
Il codice risulta essere:

```
MAIN:

    ;stampa "C"
    dvHMI:dis6 = CHAR_C

    ;configura e abilita screenA
    dvHMI:ncharA = 6
    dvHMI:offsA = 0
    dvHMI:decptA = 0
    dvHMI:scflags = dvHMI:scflags ORB SCRA_ENABLE

    tm01 = 5000        ;utilizzo del timer per causare un errore

LOOP:

    dvHMI:screenA = (dvHMI:screenA + 1) % 999999

    ;Errore?
    IF tm01
```

```
        ;disabilita screenA
        dvHMI:scflags = dvHMI:scflags ANDB ( NOT SCRA_ENABLE )
        CALL ERROR
        errNum = errNum + 1
        JUMP MAIN
    ENDIF

    WAIT 1
    JUMP LOOP

SUB ERROR
    ;stampa "ERROR"
    dvHMI:dis6 = CHAR_E
    dvHMI:dis5 = CHAR_R
    dvHMI:dis4 = CHAR_R
    dvHMI:dis3 = CHAR_O
    dvHMI:dis2 = CHAR_R


    ;stampa errore con identificativo
    dvHMI:deoffs = 0
    dvHMI:denchar = 2
    dvHMI:dedecpt = 0
    dvHMI:devalue = errNum
    dvHMI:deflags = DE_DISSIGN
    DATAENTRY dvHMI

    ;attesa 2 secondi
    tm01 = 2000
    WAIT tm01

ENDSUB

END
```

Diagnostica Ingressi

Si voglia creare una visualizzazione che rappresenti lo stato di 9 ingressi digitali. Lo stesso esempio potrà essere poi utilizzato per la rappresentazione di uscite digitali. Assegneremo perciò, ad ogni ingresso, uno dei segmenti di ciascuno dei tre digit più a destra e lo attiveremo quando il corrispondente ingresso sarà attivo.
La figura mostra l'assegnamento scelto per gli ingressi ed i segmenti dei digit del display:



Innanzitutto dichiareremo, nella unit di configurazione, 9 variabili di dimensione FLAG il cui valore simulerà lo stato dei 9 ingressi digitali.

```
GLOBAL
gfInp01         F
gfInp02         F
gfInp03         F
gfInp04         F
gfInp05         F
gfInp06         F
gfInp07         F
gfInp08         F
gfInp09         F
```

Dichiareremo poi anche un array global da 8 elementi che servirà per contenere i codici dei caratteri da stampare per ogni combinazione degli ingressi.

```
ARRGBL
    diagnTab        B    8           ;tabella caratteri per diagnostica
```

Infatti per ogni gruppo di tre ingressi associati ad uno dei tre digit sul display avremo 8 possibili combinazioni. La tabella riassume, ad esempio, per la combinazione degli ingressi I7,I8 ed I9, i possibili lo stati del digit associato:

I7	I8	I9	Display
0	0	0	
0	0	1	
0	1	0	-
0	1	1	=
1	0	0	

I7	I8	I9	Display
1	0	1	~
1	1	0	*
1	1	1	$

Servirà infine anche la definizione di alcune costanti da utilizzare come maschera per bit generici di un byte:

```
CONST

;-- Generic bit field mask ---------
B_00            &H01      ; value for bit 00
B_01            &H02      ; value for bit 01
B_02            &H04      ; value for bit 02
B_03            &H08      ; value for bit 03
B_04            &H10      ; value for bit 04
B_05            &H20      ; value for bit 05
B_06            &H40      ; value for bit 06
B_07            &H80      ; value for bit 07
```

Il codice completo per ottenere la funzione di diagnostica è:

```
;inizializza tabella
    diagnTab[1] = CHAR
    diagnTab[2] = CHAR_UP
    diagnTab[3] = CHAR_CENTER
    diagnTab[4] = CHAR_UPCEN
    diagnTab[5] = CHAR_LOWER
    diagnTab[6] = CHAR_LOWUP
    diagnTab[7] = CHAR_LOWCEN
    diagnTab[8] = CHAR_LOWUPCE

    ;stampa messaggio "INP."
    hmi:dis6 = CHAR_I
    hmi:dis5 = CHAR_N
    hmi:dis4 = CHAR_P ORB CHAR_POINT

MAIN:
  hmi:dis2 = diagnTab[(gfInp01 * B_00 + gfInp02 * B_01 + gfInp03 * B_02) + 1]
  hmi:dis1 = diagnTab[(gfInp04 * B_00 + gfInp05 * B_01 + gfInp06 * B_02) + 1]
  hmi:dis0 = diagnTab[(gfInp07 * B_00 + gfInp08 * B_01 + gfInp09 * B_02) + 1]

  WAIT 1
  JUMP MAIN
END
```