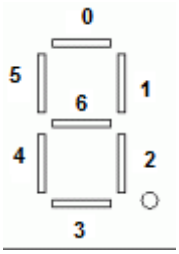
Utilizzare il display a 7 segmenti: il display è solo una rappresentazione dei primi quattro byte di memoria

I 7 bit meno significativi (l'ottavo cioè il più significativo, viene ignorato) del byte più basso sono per la cifra a 7 segmenti più a destra (gli altri 3 byte nella dword sono per gli altri tre display a 7 segmenti).

Memorizzando nella cella [0] cioè con indirizzo 0x0 (o qualunque sia il più basso indirizzo della memoria) il valore 127 (1 + 2 + 4 + 8 + 16 + 32 + 64), si accenderanno tutti i segmenti del display più a destra.

I segmenti sono numerati da 0 a 5 a partire dal segmento superiore e andando in senso orario.

1101111b ; 9 ..... anche dd 01101111b
1111111b ; 8
0000111b ; 7
1111101b ; 6
1101101b ; 5
1100110b ; 4
1001111b ; 3
1011011b ; 2
0000110b ; 1 (2+4=6)
0111111b ; 0



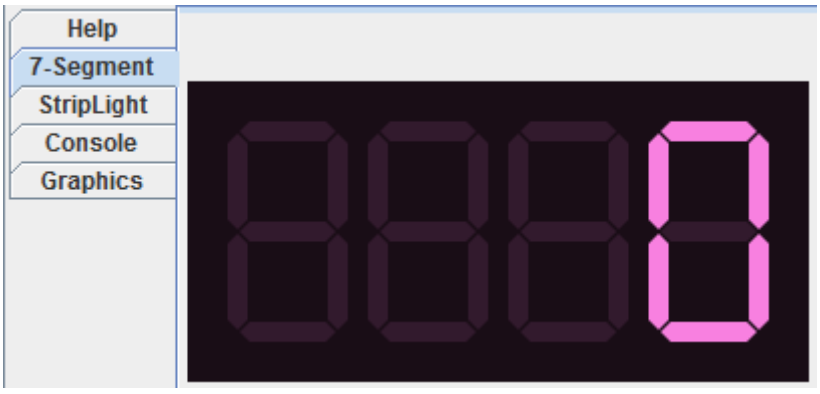
Il segmento in mezzo è numerato con 6. L'impostazione metterà in evidenza il segmento con il numero corrispondente.

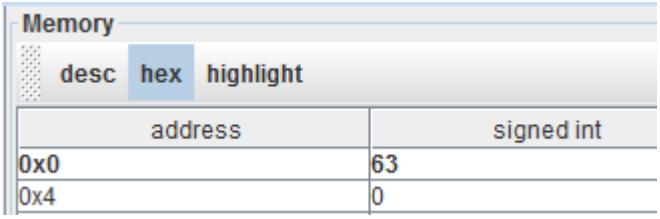
Se volessimo visualizzare un 7 avremmo bisogno di impostare i bit 0-2 (valore 7). **0000111b ; 7**

Per un 9 impostiamo tutti i bits a 1 meno il bit 4 (valore 127-8 = 119). **1101111b ; 9**

Un buon modo per codificare i numeri nella loro rappresentazione a 7 segmenti è quello di utilizzare una funzione. La tabella può essere creata con il comando "dd".

dd 0111111b





Di conseguenza, sarà possibile utilizzare questa funzione per visualizzare da A a D per codificare da 10 a 15 in esadecimale o anche per tutte le lettere dell'alfabeto (anche se un display a 7 segmenti rende la lettura del risultato piuttosto difficile).

Nb: le cifre risultano spente in corrispondenza del valore 00000000b (di default)

Per la lettera **I** come 1 → 0000110b (2+4=6)

Per la lettera **S** accesi 6+5+3+2+0 01101101b cioè 1+4+8+32+64 = 109

Per visualizzare **SI**

dd 27910 oppure in codice esadecimale **dd 0x00006D06**

analogo a

```
3 mov [0x00],6
4 mov [0x01],109
```

Memory

desc | hex | highlight | 8 Bit

address	signed int	unsigned int	hex
0x0	6	6	0x06
0x1	109	109	0x6D

Si veda [tutorial]

Per inizializzare tutte le 4 cifre a 8 (tutti i segmenti accesi):

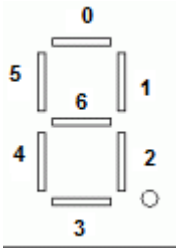
; inizializzo display tutte le cifre a 8 (127 in ognuno dei primi 4 byte)
; mov [0x0], 1111111b
; mov [0x1], 1111111b
; mov [0x2], 1111111b
; mov [0x3], 1111111b
;dd 2139062143 ; analogo in decimale (cella a 32 bit all'indirizzo 0x0)
dd **0x7F7F7F7F** ; analogo per inizializzare tutte le cifre a 8 in esadecimale

Per inizializzare le 2 cifre meno significative a 8 (le due cifre più significative spente):

; inizializzo display (127 in ognuno dei 2 byte meno significativi)
;mov [0x0], 1111111b
;mov [0x1], 1111111b
;mov [0x2], 0000000b ; *di default*
;mov [0x3], 0000000b ; *di default*
; dd 32639 ; analogo in decimale
; (cella a 32 bit all'indirizzo 0x0)
dd 0x00007F7F ; analogo in **esadecimale**

Per inizializzare le 2 cifre meno significative a 0 (le due cifre più significative spente):

; inizializzo display con le due cifre meno significative a 0 e le altre spente
; mov [0x0], 0111111b
; mov [0x1], 0111111b
 ; dd 16191 ; analogo per inizializzare in decimale
 dd 0x00003F3F ; analogo per inizializzare in esadecimale



Per stampare **ERR** se dispari:

per **E 0+5+6+4+3 111 1001b 1+8+16+32+64 121**

per **R 0+1+2+4+5+6 111 0111b 1+2+4+16+32+64 119**

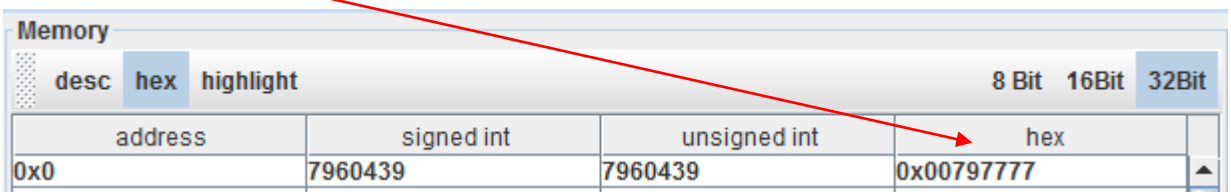
; mov [0x0], **1110111b**
; mov [0x1], **1110111b**
; mov [0x2], **1111001b ; lettera E**

Oppure

mov [0x0], 119
mov [0x1], 119
mov [0x2], 121

Analogo a
dd 7960439 ; non primo : ERR su display 7 seg

oppure **dd 0x00797777** ;analogo - codice esadecimale - in cella 0x0 (32b bit)



Nb: attenzione ad usare registri a 32 bit

; Scrivere un programma assembly che verifica se un numero n >1 è **primo**
; **SI** significa numero primo
; **ERR** non primo

```
num:
dd 2            ; primo
nonprimo:
dd 0x00797777 ;analogo - codice esadecimale -  in  cella 0x0 (32b bit)
;dd 7960439   ;   non  primo : ERR  su display 7 seg
primo:
dd 0x00006D06 ; analogo - codice esadecimale -  in  cella 0x0 (32b bit)
;dd 27910 ;  primo : SI  su dispaly 7 seg
MOV AX, [num] ;   Copia del numero in AX
CMP AX  ,2 ;   soluzione bug (*)
JE pari
MOV BX, AX          ; Inizializza BX ....
SUB BX, 1           ; ... ad AX-1
uno:
DIV BX  ; Divide AX per BX
CMP DX,0 ; Controlla il resto della divisione
JE due      ; Se il resto è zero, salta a due
MOV DX,0
MOV AX,[num] ; Ripristina AX
DEC BX ; Decremento BX
CMP BX,1 ; Confronto BX con 1
JG uno
pari:
mov edx, primo                                  ;  Il numero è primo
mov ECX, [primo]
mov [0x00],ECX ; solo primi 4 byte di memoria visualizzati
JMP tre;  Salta alla fine
due:                                            ; Il numero NON è primo
mov edx, nonprimo
mov ECX, [nonprimo]
mov [0x00],ECX ; solo primi 4 byte di memoria visualizzati
JMP tre ;  Salta alla fine
tre:
mov bx, [edx]
mov [0xC], bx
```