Linux Assembly Language - Appunti

Edizioni ByteMan

Presentazione

Tutto il materiale qui presentato è frutto di mie personali ricerche. E' possibile, pertanto, che ci siano incompletezze ed errori. In tal caso, contattatemi. All information provided here derived from my own research. So, mistakes and some sort of incompleteness could exist. If you find any, please, contact me.

Non potevo, passando a Linux, abbandonare il mio linguaggio preferito. Pertanto i primi lavori di prova in questo S.O. me li scrivo in ASM. Si tratta di una raccolta di appunti, numerati progressivamente, per il momento senza alcun ordine logico nella sequenza, che è determinata solo da contingenti necessità di studio e lavoro. E' probabile che in un prossimo futuro sarà possibile una migliore consultazione, al momento ci si deve accontentare di un semplice indice progressivo. Gli esempi presentati hanno prevalentemente funzione didattica e non vengono mai resi complicati per non venir meno alla loro finalità. Mi auguro, comunque, che la fatica per la raccolta del materiale e per la riorganizzazione dello stesso possa tornare utile oltre che ai miei studenti a quanti altri volessero consultare questi appunti.
Per il momento mi atterrò alle seguenti convenzioni, fissate anche in funzione del software che uso:

- Assemblatore: nasm della **NetWide**
- Linker 1: ld
- Linker 2: quello incluso in gcc
- Nomi dei file sorgente: con suffisso .asm
- Nomi dei file oggetto: con suffisso .o
- Nomi dei file eseguibili: senza suffisso

Ecco alcune delle motivazioni:

- Come assemblatore ho scelto nasm per continuare ad usare la sintassi Intel, già utilizzata anche con altri sistemi operativi.
  Non ho intenzione, per il momento, di usare la sintassi AT&T, e quindi non considererò esempi da compilare con gas.
- Utilizzerò il linker gcc solo quando sarà necessario usare le funzioni della libreria libc. Il risultato finale sarà un po' più lungo in termini di byte, ma la semplicità con cui si importano le funzioni è, per me, didatticamente più importante. Da notare, ancora, che gcc utilizza, a sua volta, il linker ld.
- In tutti gli altri casi utilizzerò, invece, il linker ld per ottenere un eseguibile più compatto.

Introduzione ASM Linux

A partire da un file sorgente è necessario eseguire due operazioni per ottenere un file eseguibile: la compilazione e il collegamento (linking). Durante la fase di compilazione viene effettuata l'analisi necessaria per verificare la correttezza del codice e, in caso di esito positivo, si passa alla traduzione del sorgente in linguaggio macchina. Dopo la fase di compilazione è necessaria una fase di collegamento (linking) in cui si vanno a collegare più file compilati ed eventuali librerie statiche e/o dinamiche per ottenere un file eseguibile correttamente caricabile dal sistema operativo. L'intero procedimento si riduce, quindi, ai seguenti 2 passaggi:

```
nasm -f elf -o prova.o prova.asm
gcc -s -o prova prova.o
```

oppure ai seguenti 2:

```
nasm -f elf -o prova.o prova.asm
ld -s -o prova prova.o
```

In tutti e due i casi il parametro -o è seguito dal nome del file di output, il parametro -f elf specifica il formato del file oggetto, il parametro -s ordina di effettuare lo strip delle funzioni inutili dal file binario di uscita.

Per facilitare la compilazione si possono, in alternativa, usare i seguenti script, migliorabili sicuramente, ma è una cosa che farò in un secondo tempo:

```
#!/bin/sh                              #!/bin/sh
# ag.sh ::::::::::::::::::::::         # ag.sh ::::::::::::::::::::::::::::::::::::::
nasm -f elf -o $1.o $1.asm             nasm -f elf -o $1.o $1.asm && gcc -s -o $1 $1.o
gcc -s -o $1 $1.o             oppure   rm $1.o -f
rm $1.o -f                             #EOF ::::::::::::::::::::::::::::::::::::::::
#EOF ::::::::::::::::::::::::


#!/bin/sh                              #!/bin/sh
# al.sh ::::::::::::::::::::::         # al.sh ::::::::::::::::::::::::::::::::::::::
nasm -f elf -o $1.o $1.asm             nasm -f elf -o $1.o $1.asm && ld -s -o $1 $1.o
ld -s -o $1 $1.o              oppure   rm $1.o -f
rm $1.o -f                             #EOF ::::::::::::::::::::::::::::::::::::::::
#EOF ::::::::::::::::::::::::
```

La compilazione con l'uso di uno degli script si eseguirà quindi con:

```
./al.sh prova
```

Oltre alla compilazione, lo script provvederà a rimuovere il file oggetto, generalmente non utilizzato, con il comando rm.

L'esecuzione del programma, compilato nella propria cartella di lavoro, si eseguirà digitando:

```
./prova
```

Quasi il classico "Hello world"

Il primo esempio leggerà il marchio di fabbrica della CPU e lo visualizzerà sul terminale. Il codice è scritto per macchine di classe Pentium e non esegue alcun controllo per i 486 o precedenti.

```
;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
; hello_1.asm
;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
; compilare con nasm+gcc

global  main
extern  printf

section .data
cls     db 1Bh,"[2J",0
msg     db 0Dh,0Ah,"Marchio del produttore della CPU: "
idx     dd 0,0,0
        db 0Dh,0Ah,0Ah,0

section .text
main:   push    dword cls       ;Clear del terminale
        call    printf
        pop     eax
        mov     eax,0           ;Identificazione CPU
        cpuid
        mov     [idx],ebx       ;Lettura prime 4 lettere
        mov     [idx+4],edx     ;Lettura 4 lettere centrali
        mov     [idx+8],ecx     ;Lettura ultime 4 lettere
        push    dword msg       ;Output Messaggio
        call    printf
        pop     eax
        ret                     ;Uscita programma
; EOF :::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

L'esempio appena illustrato esegue quattro operazioni: la pulizia del terminale, l'identificazione della CPU, l'emissione di un messaggio, l'uscita dal programma. L'uso della printf, appartenente alla libreria libc, semplifica le operazioni di output su terminale (clear e messaggio), si noti il passaggio del parametro (puntatore alla stringa) tramite una push con conseguente pop per ripristinare lo stato dello stack. Al posto della pop eax, che coinvolge l'uso di un registro, si potrebbe usare una add esp,4 che risolve il problema del riallineamento dello stack senza coinvolgere registri. L'uscita del programma realizzata con una semplicissima ret.

Ci proponiamo di risolvere lo stesso problema con l'uso di chiamate al kernel, ovvero con le cosiddette syscall. La soluzione è leggermente più complessa, ma è di particolare interesse il passaggio dei parametri tramite registri; si noti, in particolare, che il registro eax viene utilizzato per individuare il numero della syscall da usare, mentre i parametri (par1, par2, par3, ...) vengono passati ordinatamente tramite i registri ebx, ecx, edx, esi, edi, ebp, rispettivamente.
Il confronto con la soluzione precedente evidenzia alcune cose: l'assenza della dichiarazione extern, la mancanza del terminatore null alla fine delle stringhe, la presenza di una costante per l'individuazione della lunghezza delle stringhe.

```
;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
; hello_2.asm
;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
; compilare con nasm+ld

global  main

section .data
cls     db 1Bh,"[2J"
CLS     equ     $-cls
msg     db 0Dh,0Ah,"Marchio del produttore della CPU: "
idx     dd 0,0,0
        db 0Dh,0Ah,0Ah
MSG     equ     $-msg

section .text
main:                           ;Clear del terminale
        mov     edx,CLS         ;par3: lunghezza del messaggio
        mov     ecx,cls         ;par2: indirizzo del messaggio
```

```
        mov     ebx,1           ;par1: descrittore del file (stdout)
        mov     eax,4           ;numero della syscall write
        int     80h             ;chiamata kernel syscall
        mov     eax,0           ;Identificazione CPU
        cpuid
        mov     [idx],ebx       ;Lettura prime 4 lettere
        mov     [idx+4],edx     ;Lettura 4 lettere centrali
        mov     [idx+8],ecx     ;Lettura ultime 4 lettere
                                ;Output Messaggio
        mov     edx,MSG         ;par3: lunghezza del messaggio
        mov     ecx,msg         ;par2: indirizzo del messaggio
        mov     ebx,1           ;par1: descrittore del file (stdout)
        mov     eax,4           ;numero della syscall write
        int     80h             ;chiamata kernel syscall
                                ;Uscita programma
        mov     ebx,0           ;Par1: codice di ritorno
        mov     eax,1           ;numero della syscall exit
        int     80h             ;chiamata kernel syscall
; EOF ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

Se effettuiamo un confronto sulla lunghezza dei due eseguibili ottenuti, anche se i sorgenti non sono perfettamente identici, possiamo notare l'evidente maggior compattezza di hello_2 rispetto ad hello_1.

```
-rwxr-xr-x  1 byteman users 2972 nov  1 19:02 hello_1*
-rwxr-xr-x  1 byteman users  576 nov  1 19:04 hello_2*
```

Complementi

Per quanto riguarda l'identificazione della CPU, ecco quale stringa viene restituita a seconda dei principali produttori:

```
                        AMD     "AuthenticAMD"
                        Centaur "CentaurHauls"
                        Cyrix   "CyrixInstead"
                        Intel   "GenuineIntel"
                        NexGen  "NexGenDriven"
                        Rise    "RiseRiseRise"
                        UMC     "UMC UMC UMC "
```

Input/Output di testo

Nell'esempio seguente vogliamo coinvolgere alcune funzioni base necessarie in quasi tutti i programmi: l'input da tastiera, l'elaborazione, l'output su video e l'output di errori. Prenderemo in considerazione la trasformazione in maiuscolo di una stringa immessa da tastiera. Utilizzeremo solo syscall e pertanto potremo eseguire il link direttamente con ld. Creeremo le condizioni per generare un errore, nel caso di input nullo, in modo da utilizzare tre descrittori di file: STDIN per l'input, STDOUT per l'output normale, STDERR per l'output d'errore.

```
;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
; upper_1.asm
; Converte in maiuscolo l'input dell'utente.
;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
; compilare con nasm+ld

%define STDIN 0
%define STDOUT 1
%define STDERR 2
%define SYSCALL_EXIT  1
%define SYSCALL_READ  3
%define SYSCALL_WRITE 4
%define BUFLEN 256


        section .data                   ; Sezione dati inizializzati
msg1:   db "Digita una stringa: "       ; prompt
MSG1:   equ $-msg1                      ; lunghezza di msg1
msg2:   db "Originale:  "               ;
MSG2:   equ $-msg2                      ; lunghezza di msg2
msg3:   db "Convertita: "               ;
MSG3:   equ $-msg3                      ; lunghezza di msg3
msg4:   db 10, "ERRORE: input nullo.", 10    ; messaggio d'errore
MSG4:   equ $-msg4                      ;lunghezza di msg4

        section .bss                    ; Sezione dati non inizializzati
buf:    resb BUFLEN                     ; buffer di input da tastiera
newstr: resb BUFLEN                     ; buffer di output conversione
rlen:   resb 4                          ; lunghezza stringa in input (incluso l'invio)

        section .text                   ; Sezione codice
        global  _start                  ; let loader see entry point

_start:
;::::::::::::::::::::::::::::::::::::: Output Prompt
        mov     eax, SYSCALL_WRITE      ; write function
        mov     ebx, STDOUT             ; Arg1: file descriptor
        mov     ecx, msg1               ; Arg2: addr of message
        mov     edx, MSG1               ; Arg3: length of message
        int     080h                    ; syscall kernel to write

;::::::::::::::::::::::::::::::::::::: Input da tastiera
        mov     eax, SYSCALL_READ       ; read function
        mov     ebx, STDIN              ; Arg 1: file descriptor
        mov     ecx, buf                ; Arg 2: address of buffer
        mov     edx, BUFLEN             ; Arg 3: buffer length
        int     080h                    ; syscall kernel to read

;::::::::::::::::::::::::::::::::::::: Test Errore
        mov     [rlen], eax             ; salva la lunghezza della stringa
        dec     eax                     ; decrementa per escludere l'invio
        cmp     eax, 0                  ; Test su zero
        jg      read_OK                 ; Uscita read_OK se Test>0

;::::::::::::::::::::::::::::::::::::: Output messaggio d'errore
        mov     eax, SYSCALL_WRITE      ; write function
        mov     ebx, STDERR             ; Arg1: file descriptor
        mov     ecx, msg4               ; Arg2: addr of message
        mov     edx, MSG4               ; Arg3: length of message
        int     080h                    ; syscall kernel to write
        jmp     exit1                   ; salta all'uscita: errore =1
```

```
;:::::::::::::::::::::::::::::::::::::::: Conversione stringa
read_OK:
        mov     ecx, [rlen]             ; inizializza il contatore
        mov     esi, buf                ; puntatore al buffer di input
        mov     edi, newstr             ; puntatore al buffer di conversione
nextch: mov     al, [esi]               ; legge un carattere
        inc     esi                     ; aggiorna il puntatore sorgente
        cmp     al, ’a’                 ; verifica che sia minuscolo
        jb      store                   ; Se NO: Salta la conversione
        cmp     al, ’z’
        ja      store                   ; Se NO: Salta la conversione
        and     al,0DFh                 ; Se SI: Converte in maiuscolo
store:  mov     [edi], al               ; memorizza il carattere
        inc     edi                     ; aggiorna il puntatore destinazione
        dec     ecx                     ; aggiorna il contatore
        jnz     nextch                  ; ricicla

;:::::::::::::::::::::::::::::::::::::::: Output messaggio 2
        mov     eax, SYSCALL_WRITE      ; write message
        mov     ebx, STDOUT
        mov     ecx, msg2
        mov     edx, MSG2
        int     080h

;:::::::::::::::::::::::::::::::::::::::: Output stringa originale
        mov     eax, SYSCALL_WRITE      ; write user input
        mov     ebx, STDOUT
        mov     ecx, buf
        mov     edx, [rlen]
        int     080h

;:::::::::::::::::::::::::::::::::::::::: Output messaggio 3
        mov     EAX, SYSCALL_WRITE      ; write message
        mov     EBX, STDOUT
        mov     ECX, msg3
        mov     EDX, MSG3
        int     080h

;:::::::::::::::::::::::::::::::::::::::: Output stringa convertita
        mov     EAX, SYSCALL_WRITE      ; write out string
        mov     EBX, STDOUT
        mov     ECX, newstr
        mov     EDX, [rlen]
        int     080h

;:::::::::::::::::::::::::::::::::::::::: Uscita
        mov     EBX, 0                  ; exit code, normal=0
        jmp     exit
exit1:  mov     EBX, 1                  ; exit code, error=1
exit:   mov     EAX, SYSCALL_EXIT       ; exit function
        int     080h                    ; syscall kernel to take over
; EOF :::::::::::::::::::::::::::::::::::::::::::::::::::::
```

Tra le cose da notare, l’inserimento della sezione .bss per la definizione di variabili non inizializzate (i buffer: buf e newstr), in contrapposizione alla sezione .data; ed ancora le pseudoistruzioni %define per introdurre delle costanti. Infine la funzione exit riporta in ebx un codice di uscita che vale "0" se tutto si è concluso regolarmente, oppure "1" in caso di uscita su STDERR.

Recuperare la command line, e non solo

Quando un programma elf viene lanciato eredita una serie di informazioni depositate come dword sullo stack. Queste informazioni riguardano sia la command line, completa di tutti gli argomenti, sia le variabili d’ambiente associate. La situazione dello stack, all’inizio dell’esecuzione, è schematizzata nella figura seguente.

argC	integer: Numero di argomenti (C=N+1)
arg0	pointer: Nome del programma
arg1	pointers: Argomenti command line
arg2	
....	
argN	
null	integer: Terminatore argomenti
env0	pointers: Variabili d’ambiente
env1	
....	
envM	
null	integer: Terminatore variabili

L’esempio seguente estrae dallo stack tutto il contenuto schematizzato nella tabella precedente e lo visualizza sul terminale. La conversione del contatore è effettuata per semplicità, nell’ipotesi che il suo valore sia minore di 10, aggiungendo semplicemente una costante opportuna.

```
;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
; commline_1.asm
; Visualizza command line e variabili d’ambiente.
;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
; compilare con nasm+ld

%define STDOUT 1
%define SYSCALL_EXIT  1
%define SYSCALL_WRITE 4

global  _start                          ;necessario per il linker ld

        section .data                   ; Sezione dati inizializzati
msg1    db      0Dh,0Ah,"Numero di argomenti: "
argc    dd      0
        db      " incluso il nome del programma.",0Dh,0Ah
MSG1    equ     $-msg1
msg2    db      0Dh,0Ah,"Elenco argomenti: ",0Dh,0Ah
MSG2    equ     $-msg2
msg3    db      0Dh,0Ah,"Elenco variabili d’ambiente: ",0Dh,0Ah
MSG3    equ     $-msg3
crlf    db      0Dh,0Ah
xflag   dd      0

        section .text
_start:
;:::::::::::::::::::::::::::::::::::::::: Visualizza Numero degli Argomenti
        pop     ecx                     ; Estrae il contatore degli argomenti
        add     ecx,20202030h           ; Conversione in ASCII (Ipotesi N<10)
        mov     [argc],ecx              ; Inserimento nel corpo di msg1
        mov     eax, SYSCALL_WRITE      ; write function
        mov     ebx, STDOUT             ; Arg1: file descriptor
        mov     ecx, msg1
        mov     edx, MSG1
        int     80h                     ; syscall kernel to write
;:::::::::::::::::::::::::::::::::::::::; Visualizza Messaggio Primo Elenco
        mov     eax, SYSCALL_WRITE      ; write function
        mov     ebx, STDOUT             ; Arg1: file descriptor
        mov     ecx, msg2
        mov     edx, MSG2
```

```
        int     80h                     ; syscall kernel to write
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; Ciclo principale estrazione/visualizzazione
                                        ;    utilizzato sia per gli argomenti sia per
                                        ;    le variabili d'ambiente
xloop:  pop     ecx                     ; Estrae un puntatore
        or      ecx,ecx                 ; Verifica se null
        jz      exit1                   ; Se SI: va a exit1
                                        ; Se NO: prosegue
        mov     esi,ecx                 ; Utilizza Puntatore esi per usare la lodsb
        xor     edx,edx                 ; Azzera contatore lunghezza (edx)
        dec     edx
strlen: inc     edx                     ; calcola la lunghezza di msg
        lodsb
        or      al,al
        jnz     strlen
                                        ; Visualizza msg
        mov     eax, SYSCALL_WRITE      ; write function
        mov     ebx, STDOUT             ; Arg1: file descriptor
        int     80h                     ; syscall kernel to write
        call    newln                   ; Aggiunge un newline
        jmp     short xloop             ; Ricicla
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; Controllo per il secondo giro
exit1:  mov     eax,[xflag]
        inc     eax
        mov     [xflag],eax
        cmp     eax,2
        je      exit
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; Visualizza Messaggio Secondo Elenco
        mov     eax, SYSCALL_WRITE      ; write function
        mov     ebx, STDOUT             ; Arg1: file descriptor
        mov     ecx, msg3
        mov     edx, MSG3
        int     80h                     ; syscall kernel to write
        jmp     short xloop             ; Rientra Secondo giro
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; Uscita
exit:   mov     EBX, 0                  ; exit code, normal=0
        mov     EAX, SYSCALL_EXIT       ; exit function
        int     80h                     ; syscall kernel to take over

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; Routine NewLine
newln:  mov     eax, SYSCALL_WRITE      ; write function
        mov     ebx, STDOUT             ; Arg1: file descriptor
        mov     ecx, crlf
        mov     edx, 2
        int     80h                     ; syscall kernel to write
        ret
; EOF ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

Poichè le variabili d'ambiente sono molte è opportuno avviare il programma con:

```
./commline_1 ... ... ... |more
```

in modo da potere scorrere agevolmente tutto l'output. Si noterà, tra l'altro, che l'ultima voce delle variabili d'ambiente contiene ancora il nome del programma, che era già presente come arg0 nella lista degli argomenti.

Talvolta è necessario, invece, avere dei puntatori diretti ad alcuni punti speciali dello stack, senza dovere essere costretti a scaricarlo tutto, ecco alcuni suggerimenti:

```
        pop     eax                     ; Lettura contatore degli argomenti
        pop     esi                     ; Il reg. esi punta il primo degli argomenti
oppure
        pop     eax                     ; Lettura contatore degli argomenti
        mov     esi,[esp+eax*4]         ; Il reg. esi punta l'ultimo degli argomenti
oppure
        pop     eax                     ; Lettura contatore degli argomenti
        mov     esi,[esp+(eax+1)*4]     ; Il reg. esi punta la prima var. d'ambiente
```

E' evidente che queste istruzioni devono essere usate proprio all'inizio del programma, prima di avere alterato lo stato dello stack.

L’esempio precedente ha, secondo me, l’inconveniente di alterare la situazione dello stack in quanto c’è una palese quantità di istruzioni di pop non equilibrate o da altrettante istruzioni di push o da una opportuna modifica del valore del registro esp. A tal proposito, c’è una scuola di pensiero che vuole che il programmatore, all’uscita del programma, lasci il puntatore di stack con lo stesso valore trovato all’inizio.
Quella che segue è una variante breve dell’esercizio precedente che illustra questo secondo modo di agire, per semplicità tratta soltanto gli argomenti della command line.

```
;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
; commline_2.asm
; Visualizza la command line
;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
; compilare con nasm+gcc

%define STDOUT 1
%define SYSCALL_WRITE 4

        global  main
        extern  printf

        section .data
        msg1    db      0Dh,0Ah,"Numero di argomenti: "
argc    dd      0
        db      " incluso il nome del programma.",0Dh,0Ah
MSG1    equ     $-msg1
format: db      ’%s’, 10, 0             ; Stringa di formato

        section .text
main:
;:::::::::::::::::::::::::::::::::::::
        mov     ecx, [esp+4]            ; Lettura di argC
        push    ecx                     ; Salvataggio ecx
        add     ecx,20202030h           ; Conversione in ASCII (Ipotesi N<10)
        mov     [argc],ecx              ; Inserimento nel corpo di msg1
        mov     eax, SYSCALL_WRITE      ; write function
        mov     ebx, STDOUT             ; Arg1: file descriptor
        mov     ecx, msg1
        mov     edx, MSG1
        int     80h                     ; syscall kernel to write
        pop     ecx                     ; Ripristino ecx
        mov     edx, [esp+8]            ; Lettura puntatore di arg0
;:::::::::::::::::::::::::::::::::::::: Ciclo di lavoro
xloop:  push    ecx                     ; Salvataggio registri usati da printf
        push    edx                     ;
;.....................................
        push    dword [edx]             ;    Passa puntatore stringa
        push    dword format            ;    Passa stringa formato
        call    printf                  ;    Visualizza argomento
        add     esp, 8                  ;    Rimozione parametri dallo stack
;.....................................
        pop     edx                     ; Ripristino registri
        pop     ecx                     ;
        add     edx, 4                  ; Modifica puntatore ad arg successivo
        dec     ecx                     ; Decrementa contatore
        jnz     xloop                   ; Ricicla
        ret
;EOF ::::::::::::::::::::::::::::::::::::::::::::::::
```

Abbiamo usato il linker gcc in quanto si è preferito, per semplicità, effettuare l’output tramite la printf; si noti, a questo proposito, l’uso della stringa di formato dentro il segmento .data per concorrere al formato della printf. Il puntatore di stack esp, alla fine del programma, rimane inalterato.

I colori sul terminale e le macro

Gli esercizi di questa sezione hanno un duplice scopo:

- realizzare l'output su terminale di un testo a colori
- utilizzare le macro in un programma nasm.

Il primo problema lo risolviamo con le sequenze di escape. In sostanza inviando una stringa opportuna al terminale è possibile reimpostare gli attributi colore. Questa stringa, in particolare, che chiameremo foba (foreground background) è composta da 9 byte ed ha la seguente struttura:

1Bh	5Bh	33h	----	3Bh	34h	----	6Dh	00h
esc	[	3		;	4		m	null
0	1	2	3	4	5	6	7	8

I byte nelle posizioni 3 e 6 rappresentano i colori di foreground e di background rispettivamente, e vanno impostati secondo la seguente tabella:

30h	0	nero
31h	1	rosso
32h	2	verde
33h	3	ocra
34h	4	blu
35h	5	fucsia
36h	6	celeste
37h	7	bianco

Ad esempio per impostare i colori rosso/blu (rosso su fondo blu) occorrerà definire una stringa che contenga in posizione 3 il byte 31h ed in posizione 6 il byte 34h, come illustrato di seguito:

```
        foba    db 1Bh,5Bh,33h,31h,3Bh,34h,34h,6Dh,00h
oppure
        foba    db 1Bh,"[31;44m",0
```

Tornando al problema del testo colorato sul terminale, una prima soluzione viene proposta nel listato seguente nel quale sono presenti delle macro sviluppate nell'ultimo listato. Per mia convenzione, i nomi delle macro asm iniziano con il carattere @ in modo che sia facile individuarle nel listato del programma. Si noti la presenza della direttiva %include che consente l'aggancio del file con le macro.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; txtcol_1.asm
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;compilare con nasm+gcc

%include "macrobase.mac"

global  main
extern  printf

section .data
foba    db 1Bh,"[37;40m",0
cls     db 1Bh,"[2J",1Bh,"[1;1f",0
crlf    db 0Dh,0Ah,0
msg20   db "Scritta in verde su fondo nero",0
msg02   db "Scritta in nero su fondo verde",0
msg71   db "Scritta in bianco su fondo rosso",0

section .text
main:   @outterm   cls                  ; Clear del terminale

        @fixcol    2,0                  ; Imposta Verde/Nero
        @outterm   msg20                ; Emette Messaggio
```

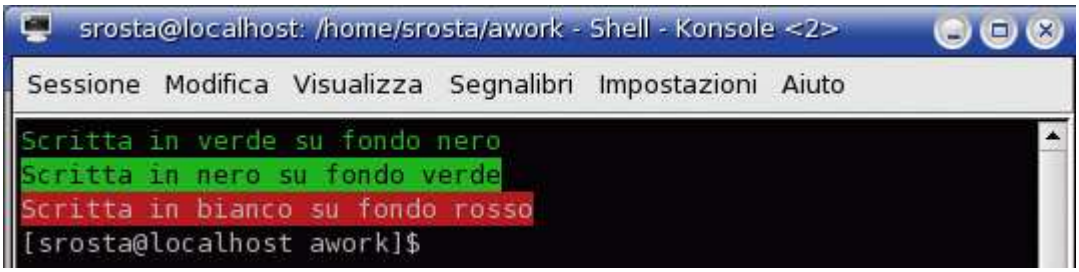
```
        @outterm   crlf                 ; Ritorno a capo

        @fixcol    0,2                  ; Imposta Nero/Verde
        @outterm   msg02
        @outterm   crlf

        @fixcol    7,1                  ; Imposta Bianco/Rosso
        @outterm   msg71
        @outterm   crlf

        @fixcol    7,0                  ; Ripristina Bianco/Nero
        ret
; EOF ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

La prima delle macro utilizzate, @fixcol, imposta i colori attivi e necessita di 2 parametri che rappresentano ordinatamente il codice colore di foreground e quello di background. La seconda macro, @outterm, serve ad emettere un messaggio di tipo AsciiZ presente come unico parametro.
Si noti nell'area dati la presenza di un'altra stringa di sequenze di escape, cls la quale serve ad effettuare il clear dello schermo ("[2J") ed il riposizionamento del cursore in alto a sinistra ("[1;1f").
Una normalissima stringa AsciiZ, crlf, serve infine per comandare il ritorno a capo tramite una @outterm.

La seconda soluzione del problema viene proposta con l'uso delle syscall e quindi le due macro precedenti, che facevano uso della printf, sono state modificate in @fixcol2 e @outterm2, in modo da essere indipendenti dalla libc. La compilazione può pertanto essere effettuata con l'uso del linker ld. Il programa segue, per il resto, l'impostazione di quello precedente.

```
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
; txtcol_2.asm
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;compilare con nasm+ld

%define STDOUT 1
%define SYSCALL_EXIT  1
%define SYSCALL_WRITE 4

%include "macrobase.mac"

global  _start                          ;necessario per il linker ld

        section .data
foba    db 1Bh,"[37;40m",0
cls     db 1Bh,"[2J",1Bh,"[1;1f",0
crlf    db 0Dh,0Ah,0
msg20   db "Scritta in verde su fondo nero",0
msg02   db "Scritta in nero su fondo verde",0
msg71   db "Scritta in bianco su fondo rosso",0

        section .text
_start:
        @outterm2 cls                   ; Clear del terminale

        @fixcol2  2,0                   ; Imposta Verde/Nero
        @outterm2 msg20                 ; Emette Messaggio
        @outterm2 crlf                  ; Ritorno a capo

        @fixcol2  0,2                   ; Imposta Nero/Verde
        @outterm2 msg02
        @outterm2 crlf

        @fixcol2  7,1                   ; Imposta Bianco/Rosso
        @outterm2 msg71
        @outterm2 crlf

        @fixcol2  7,0                   ; Ripristina Bianco/Nero
        @exit:    0                     ; uscita normale, code=0
; EOF ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

Ecco una immagine della schermata che si ottiene con i programmi presentati in questa sezione.



Ed, in ultimo, il listato delle macro utilizzate dai programmi precedenti e che viene collegato tramite la direttiva %include "macrobase.mac"

.

```
;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
; macrobase.mac
;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

%macro  @outterm  1
        push      dword %1              ;Passa il Ptr Stringa
        call      printf
        add       esp,4                 ;Riallinea lo Stack
%endmacro

%macro  @fixcol   2
        mov       byte [foba+3],%1+30h
        mov       byte [foba+6],%2+30h
        @outterm  foba
%endmacro

%macro  @fixcol2  2
        mov       byte [foba+3],%1+30h
        mov       byte [foba+6],%2+30h
        @outterm2 foba
%endmacro

%macro  @outterm2 1
        mov     ecx, %1                 ; Puntatore stringa AsciiZ
        mov     esi,ecx                 ; Utilizza Puntatore esi per usare la lodsb
        xor     edx,edx                 ; Azzera contatore lunghezza (edx)
        dec     edx                     ;
%%lp:   inc     edx
        lodsb
        or      al,al
        jnz     %%lp
        mov     eax, SYSCALL_WRITE      ; write function
        mov     ebx, STDOUT             ; Arg1: file descriptor
        int     80h                     ; syscall kernel to write
%endmacro

%macro  @exit     1
        mov       EBX, %1               ; exit code=%1
        mov       EAX, SYSCALL_EXIT     ; exit function
        int       80h                   ; syscall kernel to take over
%endmacro
;EOF ::::::::::::::::::::::::::::::::::::::::::::::::::
```

Si noti nella @outterm2 la presenza di una etichetta (label) locale %%lp contrassegnata dal prefisso %%. Una tale scelta evita errori, in compilazione, quando la macro viene usata più di una volta.
Nella @outterm2 viene effettuato il calcolo della lunghezza della stringa, passata come parametro in edx, dal momento che la SYSCALL_WRITE vuole in edx proprio questo valore.

Complementi

Con le sequenze di escape è, però, possibile aggiungere una serie di attributi per migliorare ancora di più l'output a colori. Per raggiungere lo scopo guardiamo il parametro colore di foreground come fosse un byte strutturato nella maniera seguente:

B	H	R	U	I	cB	cG	cR
7	6	5	4	3	2	1	0

I bit di posto 0,1,2 (cR,cG,cB) determinano il colore secondo la classica tripletta RGB. Gli altri bit costituiscono attributi aggiuntivi che vanno opportunamente interpretati.

- Il bit di posto 3 (I) attiva l'intensità del colore, influendo quindi sulla sua tonalità. Ad esempio il colore ocra 0 011 diviene giallo se il bit di intensità è attivo 1 011.
- Il bit di posto 4 (U) abilita la sottolineatura.
- Il bit di posto 5 (R) attiva l'inversione dei colori foreground/background.
- Il bit di posto 6 (H) rende invisibile l'output (tipico il caso di input di password).
- Il bit di posto 7 (B) attiva il lampeggio o blinking (detestato e sconsigliato dallo scrivente).

Ad esempio, per ottenere un colore ocra intenso (giallo) e sottolineato il byte dovrà essere così impostato:

0	0	0	1	1	0	1	1
7	6	5	4	3	2	1	0

valore che sarà passato come parametro colore di foreground in una qualunque base numerica: 00011011b, 1Bh, 27 comprensibile dall'assemblatore.

La nuova macro @fixcol3 che interpreterà questo byte avrà il compito di comporre una precisa sequenza di escape di lunghezza variabile in funzione degli attributi aggiunti alla tripletta RGB. La macro utilizzerà una nuova stringa dati chiamata fobax, che conterrà al suo interno:

- una intestazione necessaria a resettare gli attributi impostati in precedenza;
- la foba definita nei primi esempi;
- una coda di n byte aggiuntivi indispensabili a contenere l'estensione della sequenza nel caso estremo dell'attivazione di tutti gli attributi.

L'attivazione di ciascun attributo comporta l'aggiunta delle seguenti coppie di byte/carattere alla sequenza di escape:

```
I:      db  3Bh,31h     o       db  ";1"       o       dw  313Bh
U:      db  3Bh,34h     o       db  ";4"       o       dw  343Bh
R:      db  3Bh,37h     o       db  ";7"       o       dw  373Bh
H:      db  3Bh,38h     o       db  ";8"       o       dw  383Bh
B:      db  3Bh,35h     o       db  ";5"       o       dw  353Bh
```

L'intera sequenza dovrà essere chiusa da:

```
        db  6Dh,0       o       db  "m",0      o       dw  006Dh
```

Ecco la macro @fixcol3 che realizza l'interprete del parametro colore foreground:

```
%macro  @fixcol3  2
        mov       al,%1                 ;;Lettura parametro foreground
        mov       ah,al                 ;;Impostazione colore foreground
        and       ah,07h
        add       ah,30h
        mov       [foba+3],ah
        mov       byte [foba+6],%2+30h  ;;Impostazione colore background
        mov       esi,7                 ;;Inizio Interprete attributi
        test      al,08h                ;;Test su "I"
        mov       bx,313Bh
        call      %%work
        test      al,10h                ;;Test su "U"
        mov       bx,343Bh
        call      %%work
        test      al,20h                ;;Test su "R"
        mov       bx,373Bh
        call      %%work
        test      al,40h                ;;Test su "H"
        mov       bx,383Bh
```

```
        call      %%work
        test      al,80h               ;;Test su "B"
        mov       bx,353Bh
        call      %%work
        jmp       %%exit
;.......................................;;Routine accodamento
%%work: jz        %%wxit               ;;Esce se Test=0
        mov       [foba+esi],bx        ;;Accoda bx, se Test=1
        add       esi,2
%%wxit: ret
;.......................................
%%exit: mov       bx,006Dh             ;;Chiusura Sequenza Escape
        mov       [foba+esi],bx
        @outterm2 fobax                ;;Emissione Sequenza
%endmacro
```

Inserendo la macro @fixcol3 nel file macrobase.mac è possibile compilare il programmino di prova txtcol_3.asm riportato di seguito.

```
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
; txtcol_3.asm
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;compilare con nasm+ld

%define STDOUT 1
%define SYSCALL_EXIT  1
%define SYSCALL_WRITE 4

%include "macrobase.mac"

global  _start                         ;necessario per il linker ld

        section .data
fobax   db 1Bh,"[0m"
foba    db 1Bh,"[37;40m",0,0,0,0,0,0,0,0,0,0,0
cls     db 1Bh,"[2J",1Bh,"[1;1f",0
crlf    db 0Dh,0Ah,0
msg20   db "Scritta in giallo sottolineato su fondo nero",0

        section .text
_start:
        @outterm2 cls                  ; Clear del terminale

        @fixcol3  00011011b,0          ; Imposta colori e attributi
        @outterm2 msg20                ; Emette Messaggio
        @outterm2 crlf                 ; Ritorno a capo

        @fixcol3  7,0                  ; Ripristina Bianco/Nero
exit:   mov       EBX, 0               ; exit code, normal=0
        mov       EAX, SYSCALL_EXIT    ; exit function
        int       80h                  ; syscall kernel to take over

;:::::::::::::::::::::::::::::::::::::::
```

L’accesso all’I/O, con la scusa del beep

Se occorre inviare un beep all’operatore, tramite la console, e’ sufficiente inviare in output il carattere BEL, codice Ascii 07h, come nel semplicissimo esempio sottostante.

```
;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::
; beep_1.asm
;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;compilare con nasm+gcc

global  main                    ;necessario per il linker gcc
extern  printf

        section .data
beep    db      07h,0

        section .text
main:
        push      dword beep            ;Passa il Ptr Stringa
        call      printf
        add       esp,4                 ;Riallinea lo Stack
        ret
;EOF ::::::::::::::::::::::::::::::::::
```

Ma se non ci accontentiamo del segnale acustico predefinito e ne vogliamo uno di frequenza e durata variabili ? A questo punto risulta evidente che il beep e’ la scusa per parlare di I/O. Per raggiungere l’obiettivo faremo uso del terzo timer programmabile presente nell’8253 che fara’ giungere un segnale alla frequenza desiderata all’altoparlante, che verra’ abilitato tramite l’8255. Tutto fattibile, se il controllo dell’hardware fosse sotto il nostro totale controllo. Eppure la possibilita’ esiste, basta chiedere il permesso al S.O. per lavorare a livello di root. Pertanto il nostro esempio, su macchine Intel compatibili, realizzera’ i seguenti punti:

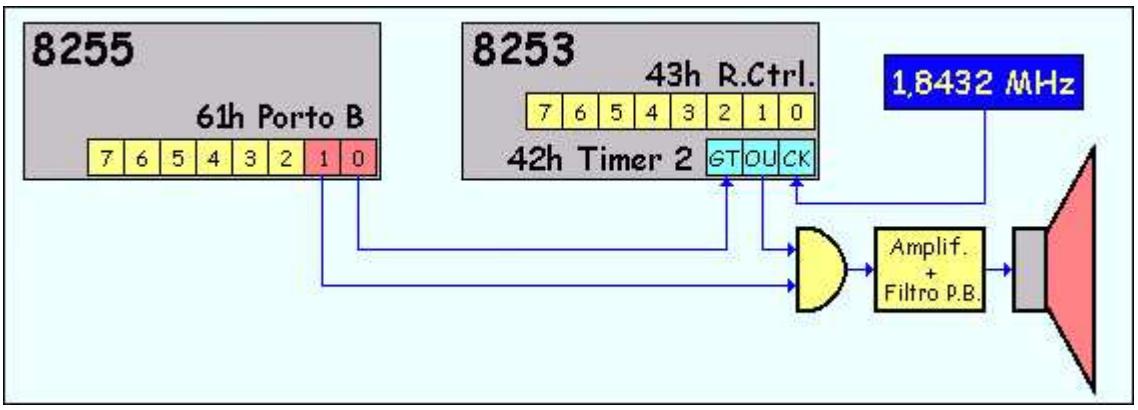
1. Richiesta autorizzazione all’uso dei dispositivi di I/O 8253 ed 8255
2. Attivazione linea altoparlante
3. Programmazione dei dispositivi di I/O per realizzare il segnale
4. Disattivazione linea altoparlante
5. Revoca autorizzazione all’uso dei dispositivi di I/O 8253 ed 8255

Il listato e’ abbastanza lineare, ma alcuni passaggi richiedono la conoscenza dei byte di programmazione per l’8253 e le connessioni hardware tra 8255, 8253 ed altoparlante.

```
;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::
; beep_2.asm
;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;compilare con nasm+gcc

%macro  @ioperm  3
        mov     ebx,%1          ;;Numero porto iniziale;
        mov     ecx,%2          ;;Quantita’ di porti in successione
        mov     edx,%3          ;;Richiesta: 1=Abilitazione, 0=Revoca
        mov     eax,101         ;;Funzione sys_call=ioperm
        int     80h
        or      eax,eax         ;;Test esito: "jns okay"
%endmacro

global  main                    ;necessario per il linker gcc
extern  usleep, printf

        section .data
msgerr  db      "NON abilitato all’uso dei porti di I/O! "
        db      "Occorrono i privilegi di ROOT.",0Ah
        db      "Oppure si e’ verificato un ERRORE nel caso "
        db      "si sia in ROOT.",7,0Ah,0
durata  dd      200000

        section .text
```

```
main:   @ioperm 61h,1,1         ;Richiesta Permesso 8255 I/O 61h
        jns     okay1
        jmp     err
okay1:  @ioperm 42h,2,1         ;Richiesta Permesso 8253 I/O 42h e 43h
        jns     okay2
        jmp     err

okay2:  in      al,61h          ;Abilita altoparlante
        or      al,3
        out     61h,al
        mov     al,0B6h         ;Programmazione timer 8253
        out     43h,al
        mov     ax,0800h        ;Impostazione costante di frequenza
        out     42h,al
        mov     al,ah
        out     42h,al

        push    dword [durata]  ;Imposta durata in microsecondi
        call    usleep
        add     esp,4           ;Riallinea lo Stack

        in      al,61h          ;Disabilita altoparlante
        and     al,0FCh
        out     61h,al
        jmp     exit

err:    push    dword msgerr    ;Passa il Ptr Stringa
        call    printf
        add     esp,4           ;Riallinea lo Stack

exit:   @ioperm 42h,2,0         ;Revoca Permessi I/O
        @ioperm 61h,1,0
        ret
;EOF :::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

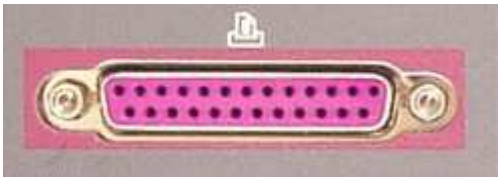
Lo schema seguente aiuta a capire alcuni passaggi relativi alle istruzioni di in e di out presenti nel programma. Come si vede, l'uscita dell'altoparlante e' condizionata sia dal segnale di out dell'8253 sia dal bit 1 del porto B 8255. A sua volta l'8253 viene abilitato al suo gate dal bit 0 del porto B 8255. Pertanto e' necessario che i 2 bit 0,1 del porto B 8255 siano alti per abilitare l'uscita del segnale sull'altoparlante (vedi or al,3), e siano invece bassi per bloccare il segnale (vedi and al,0FCh) alla fine.



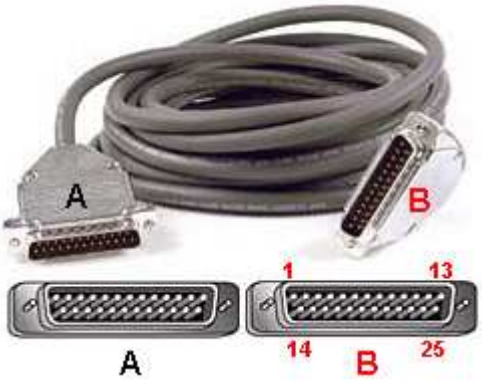
Mentre la programmazione del timer avviene in due fasi. Prima viene fissato il modo operativo dell'8253 tramite il Registro di controllo che ha sede nel Porto 43h; per funzionare come generatore di frequenza occorre inviare il byte B6h (vedi documentazione Intel). Quindi, per fissare la frequenza di funzionamento, viene inviata una word tramite il Porto 42h, e cio' si realizza con due operazioni di out, un byte alla volta.

Proviamo con la porta parallela

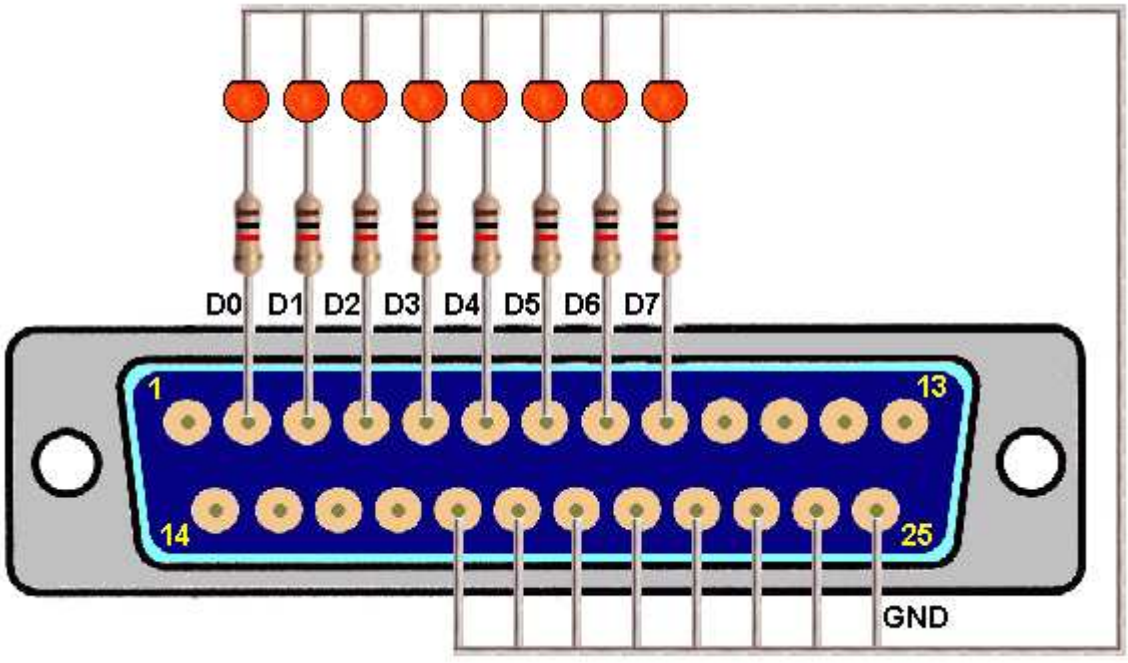
Generalmente il computer è dotato di una sola porta stampante a cui possono essere connesse la stampante ed altri dispositivi ad interfaccia parallela (ZIP, scanner, etc.). Essa è conosciuta anche come LPT, porta stampante, printer port, oppure ancora interfaccia Centronics. E' facilmente riconoscibile guardando il computer dall'esterno: si presenta come un connettore a 25 poli DB25 femmina.



Oggi spesso rimane inutilizzata in quanto molti dispositivi, tra cui le stesse stampanti, utilizzano la porta USB. La presenza di un bus I/O ad 8 bit la rende utilizzabile per interfacciare semplici circuiti elettronici. Per utilizzarla con maggiore comodità per la nostra prova ci serviremo di un cavo di prolunga parallelo maschio/maschio, come quello mostrato in fotografia, in modo da avere sul nostro tavolo i segnali provenienti dal retro del computer.



Collegheremo quindi il connettore A al computer ed il connettore B alla semplice interfaccia descritta di seguito e costituita essenzialmente da un connettore femmina, 8 led ed 8 resistenze da 1000 Ohm. Quest'ultimo connettore è mostrato dal lato delle saldature, infatti c'è una perfetta corrispondenza con la piedinatura del connettore B dove sarà inserito.



Il risultato che vogliamo ottenere è semplicemente l'accensione programmata dei vari LED. Pertanto il software, su macchine Intel compatibili, dovrà realizzare i seguenti punti:

1. Richiesta autorizzazione all'uso del dispositivo di I/O Porta Parallela
2. Programmazione delle sequenze di accensione e spegnimento dei LED
3. Revoca autorizzazione all'uso dei dispositivi di I/O Porta Parallela

L'esempio fa riferimento al dispositivo di indirizzo 378h, ma funziona ugualmente bene con il dispositivo di indirizzo 278, basta modificare nel listato il valore della costante LPTPORT.

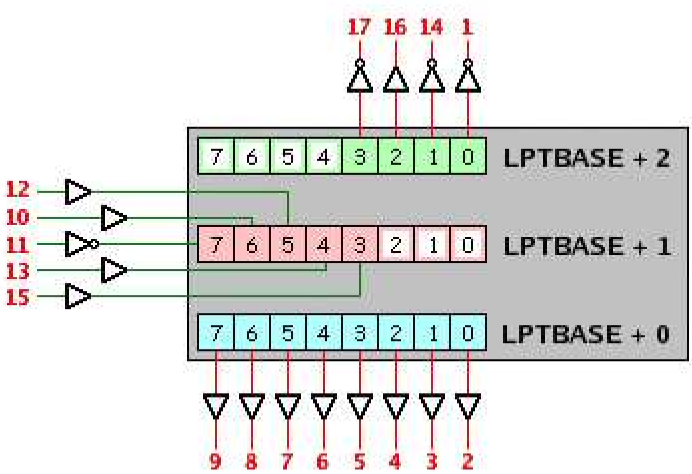
```
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
; ioled_1.asm
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;compilare con nasm+gcc

%macro  @ioperm  3
        mov     ebx,%1          ;;Numero porto iniziale;
        mov     ecx,%2          ;;Quantita' di porti in successione
        mov     edx,%3          ;;Richiesta: 1=Abilitazione, 0=Revoca
        mov     eax,101         ;;Funzione sys_call=ioperm
        int     80h
        or      eax,eax         ;;Test esito: "jns okay"
%endmacro

%define LPTPORT 378h            ;Da modificare, se necessario

global  main                    ;necessario per il linker gcc
extern  usleep, printf

        section .data
msgerr  db      "NON abilitato all'uso dei porti di I/O! "
        db      "Occorrono i privilegi di ROOT.",0Ah
        db      "Oppure si e' verificato un ERRORE nel caso "
        db      "si sia in ROOT.",7,0Ah,0
durata  dd      500000

        section .text
main:   @ioperm LPTPORT,1,1     ;Richiesta Permesso LPTPORT
        jns     okay
        jmp     err
okay:   call    seq01           ;Chiamate output LPTPORT
        call    seq02
        jmp     exit

;:::::::::::::::::::::::::::::::;ON e OFF di ciascun LED da D0 a D7
seq01:  mov     ecx,8           ;
        mov     al,1            ;Imposta D0 acceso (ON)
        mov     dx,LPTPORT
wrk01:  out     dx,al
        call    delay
        shl     al,1            ;Passa al LED successivo
        loop    wrk01
        ret

;:::::::::::::::::::::::::::::::;8 lampeggi di tutti i LED
seq02:  mov     ecx,8           ;
        mov     dx,LPTPORT
wrk02:  mov     al,0FFh         ;Imposta tutti accesi
        out     dx,al
        call    delay
        mov     al,0            ;Imposta tutti spenti
        out     dx,al
        call    delay
        loop    wrk02
        ret

;:::::::::::::::::::::::::::::::;Routine di ritardo
delay:  push    dword [durata]  ;Imposta durata in microsecondi
        call    usleep
        add     esp,4
        ret
```

```
;::::::::::::::::::::::::::::::::;Gestione Errori e Uscita
err:    push    dword msgerr    ;Passa il Ptr Stringa
        call    printf
        add     esp,4           ;Riallinea lo Stack
exit:   @ioperm LPTPORT,1,0     ;Revoca Permessi I/O
        ret
;EOF :::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

Nella tabella seguente c'è una descrizione sintetica dei singoli pin:

	Parallela Normale		Parallela ECP		
Pin	**Nome**	**Descrizione**	**Nome**	**Descrizione**	**Annotazioni**
1	/STROBE	Strobe	nStrobe	Strobe	out - Validità dei dati disponibili da D0 a D7
2	D0	Data Bit 0	data0	Address, Data or RLE Data Bit 0	out - Byte inviato alla stampante
3	D1	Data Bit 1	data1	Address, Data or RLE Data Bit 1	out - Byte inviato alla stampante
4	D2	Data Bit 2	data2	Address, Data or RLE Data Bit 2	out - Byte inviato alla stampante
5	D3	Data Bit 3	data3	Address, Data or RLE Data Bit 3	out - Byte inviato alla stampante
6	D4	Data Bit 4	data4	Address, Data or RLE Data Bit 4	out - Byte inviato alla stampante
7	D5	Data Bit 5	data5	Address, Data or RLE Data Bit 5	out - Byte inviato alla stampante
8	D6	Data Bit 6	data6	Address, Data or RLE Data Bit 6	out - Byte inviato alla stampante
9	D7	Data Bit 7	data7	Address, Data or RLE Data Bit 7	out - Byte inviato alla stampante
10	/ACK	Acknowledge	/nAck	Acknowledge	in - Disponibilità a ricevere il prossimo dato
11	BUSY	Busy	Busy	Busy	in - Stampante/Buffer impegnata/o
12	PE	Paper End	PError	Paper End	in - Stampante senza carta
13	SEL	Select	Select	Select	in - Stampante pronta
14	/AUTOFD	Autofeed	/nAutoFd	Autofeed	out - Avanzamento riga
15	/ERROR	Error	/nFault	Error	in - Errore rilevato dalla stampante
16	/INIT	Initialize	/nInit	Initialize	out - Inizializza la stampante
17	/SELIN	Select In	/nSelectIn	Select In	out - Invia una richiesta alla stampante
18	GND	Signal Ground	GND	Signal Ground	gnd - Massa
19	GND	Signal Ground	GND	Signal Ground	gnd - Massa
20	GND	Signal Ground	GND	Signal Ground	gnd - Massa
21	GND	Signal Ground	GND	Signal Ground	gnd - Massa
22	GND	Signal Ground	GND	Signal Ground	gnd - Massa
23	GND	Signal Ground	GND	Signal Ground	gnd - Massa
24	GND	Signal Ground	GND	Signal Ground	gnd - Massa
25	GND	Signal Ground	GND	Signal Ground	gnd - Massa

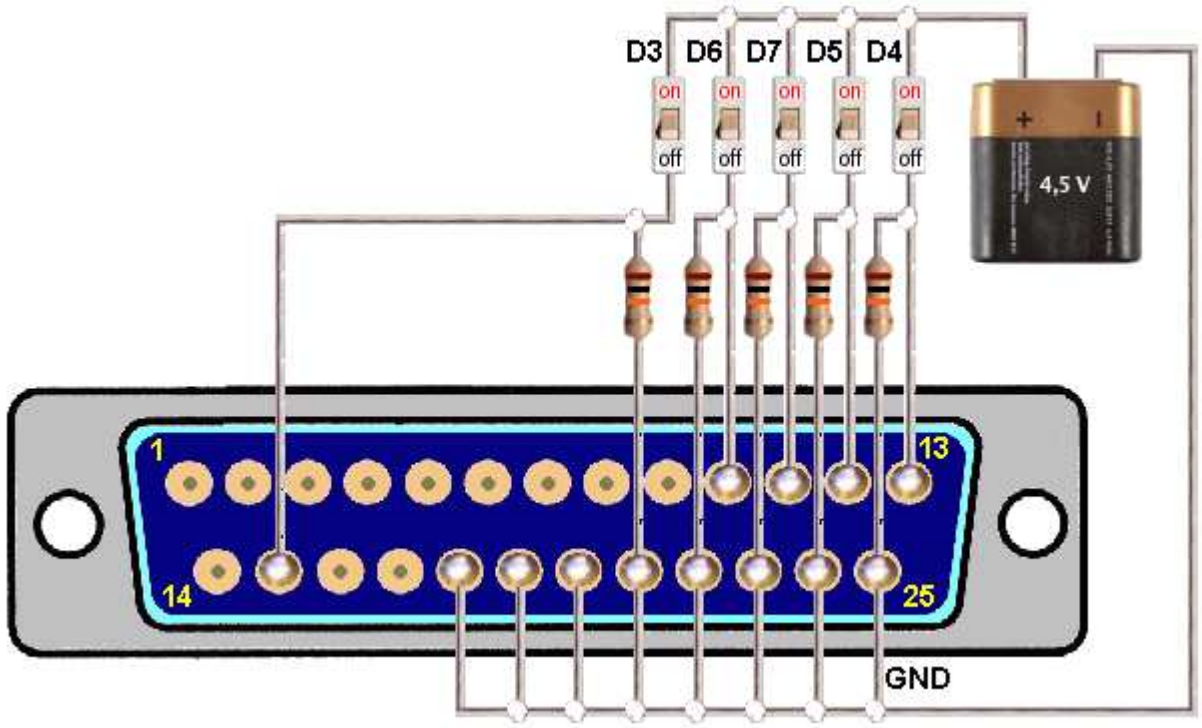
Si tenga presente che la porta parallela è stata sviluppata, all'origine, per la gestione della stampante, pertanto alcuni segnali, presenti sul connettore esterno, risultano invertiti (negati) come stato logico. In sostanza ciò che viene trattato dal programmatore a livello di registri LPTBASE+? viene ulteriormente modificato, su alcune linee, da porte logiche invertenti. La figura seguente illustra meglio la situazione.



Possiamo notare che oltre la LPTPORT+0, che abbiamo usato come output nel nostro primo esempio, ci sono anche la LPTPORT+1, che dispone di 5 ingressi, e la LPTPORT+2 che dispone di altri 4 pin di output. Il programmatore agisce sui 3 registri interni (area in grigio), mentre sui piedini del connettore (numeri in rosso) arrivano dei segnali che transitano attraverso dei buffer che, in qualche caso, sono invertenti. Attenzione: se ne tenga conto nelle applicazioni.

Complementi 1

Proponiamo, di seguito, un altro piccolo esperimento che consente di verificare l'acquisizione dei 5 bit della LPTPORT+1. La pila da 4,5 V consente di avere lo stato logico alto (valore compreso tra 1,8 e 5,1 Volt), quando l'interruttore è posizionato su on (interruttore chiuso). La resistenza da 10.000 Ohm mantiene lo stato logico basso quando l'interruttore è posizionato su off (interruttore aperto).



Il telaio del software è lo stesso di quello visto in precedenza, pertanto ci soffermiamo soltanto sulla parte relativa all'uso della LPTPORT+1. Poichè il bit 7 risulta invertito dall'hardware occorrerà un piccolo intervento, dopo l'input, per reinvertirne lo stato.

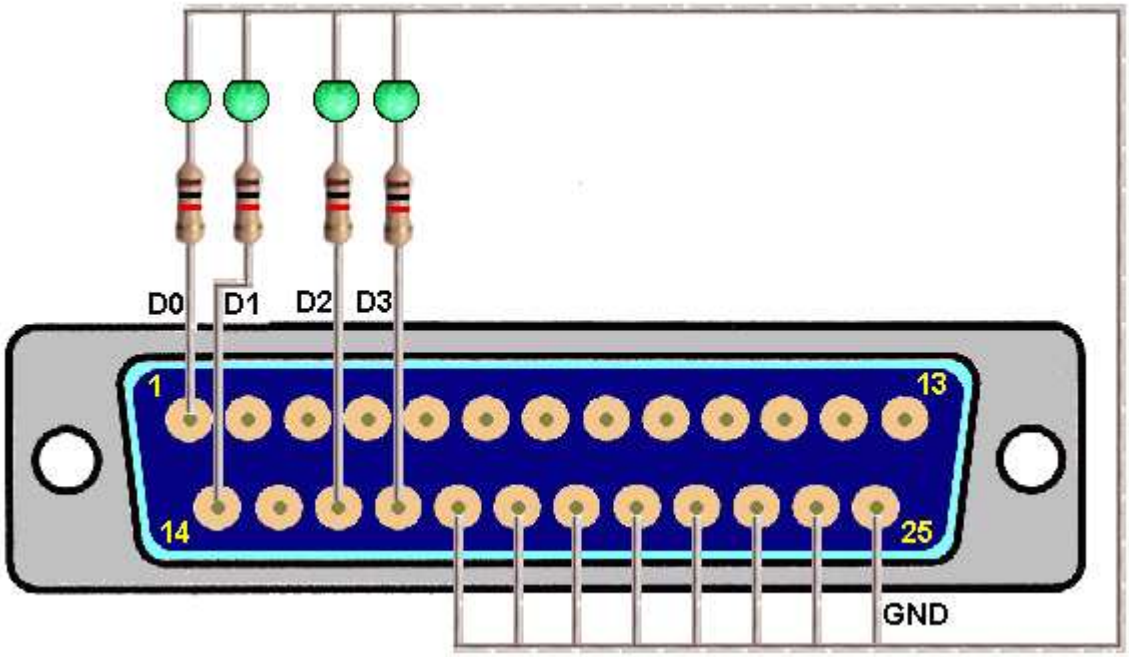
```
;:::::::::::::::::::::::::::::::::::; Acquisizione dell'input (Lettura)
        mov     dx,LPTPORT+1        ;
        in      al,dx
;:::::::::::::::::::::::::::::::::::; Inversione del bit più significativo
        mov     ah,al               ; duplica l'input nel registro ah
        not     ah                  ; inverte tutti i bit di ah
        and     ah,80h              ; maschera i bit da 6 a 0 di ah
        and     al,7Fh              ; maschera il bit 7 di al
        or      al,ah               ; ricostruisce il byte
;:::::::::::::::::::::::::::::::::::; Opzioni correttive
;       and     al,0F8h             ; opzione 1 (decommentare se necessario)
;       shr     al,3                ; opzione 2 (decommentare se necessario)
```

Attenzione: lo stato degli ultimi 3 bit non deve essere preso in considerazione in quanto non riferibile a nessun input proveniente dal connettore; pertanto la penultima riga di codice (opzione 1) può eliminare questa ambiguità azzerando sistematicamente i 3 bit. Un'altra possibilità è offerta dalla opzione 2 che, facendo scorrere verso destra il byte di 3 posti, azzera i 3 bit più significativi ed obbliga ad una rinumerazione verso il basso dei segnali ricevuti; a seconda delle situazioni e delle preferenze personali si può optare per le varie soluzioni.

Complementi 2

L'ultimo esperimento consiste nell'uso di LPTPORT+2 per l'emissione di 4 bit, tenendo conto sia della presenza di 3

inverter, sia del fatto che non dobbiamo alterare lo stato dei 4 bit più significativi.




```
;:::::::::::::::::::::::::::::::::::::; Emissione output (Scrittura)
        mov     dx,LPTPORT+2            ;
        in      al,dx                   ; Lettura dello stato preesistente
        mov     ah,pattern              ; Simulazione input del pattern
        and     ah,0Fh                  ; Maschera i 4 bit più significativi
        or      al,ah                   ; Inserzione dei 4 bit del pattern in al
;:::::::::::::::::::::::::::::::::::::; Inversione dei 3 bit negati
        mov     ah,al                   ; duplica l'output nel registro ah
        not     ah                      ; inverte tutti i bit di ah
        and     ah,0Bh                  ; maschera i bit 7,6,5,4,2 di ah
        and     al,0F4h                 ; maschera i bit 3,1,0 di al
        or      al,ah                   ; ricostruzione del byte
;:::::::::::::::::::::::::::::::::::::;
        out     dx,al                   ; E, finalmente, invio sul porto
```