

TITLE Bin2hex.asm (binary to hexadecimal) - per compito 16/04/2007

COMMENT \*

Il buffer BINBUFF contiene N byte di dati, che vanno ricodificati in formato ASCII esadecimale, riempiendo progressivamente il buffer HEXBUFF.

Il programma deve servirsi, per la ricodifica di ciascun byte di BINBUFF, della subroutine di tipo "near" BIN2HEX. Il passaggio dei parametri tra chiamante e subroutine deve avvenire tramite stack.

Si danno tre versioni leggermente diverse del programma (il programma chiamante è unico, per semplicità). La seconda è un po' più strutturata della prima, e fa uso di una macro all'interno della subroutine. Altre varianti: nella seconda versione la subroutine è FAR anziché NEAR, e si scalano i quattro byte dei parametri direttamente come argomento dell'istruzione RET. La terza versione fa uso di una "look-up table" dove sono pre-memorizzati tutti i 16 possibili caratteri ricodificati; essa quindi differisce dalle prime due, che invece calcolano ogni volta, anziché leggerlo da memoria, il carattere ricodificato.

ultima modifica: 2 maggio 2007

\*

```
-----  
;-----  
pila SEGMENT STACK 'STACK' ; definizione del segmento di stack  
      DB      64 DUP('STACK') ; lo stack è riempito con la stringa 'stack'  
                                     ; per identificarlo meglio in fase di debug  
TOS LABEL WORD ; identifica il top of stack  
pila ENDS
```

```
-----  
;-----  
DATI SEGMENT PUBLIC 'DATA' ; definizione segmento dati
```

```
Nbin EQU 7 ; numero di byte in BINBUFF  
          ; N. B. BINBUFF corrisponde a "4C5AF23E79B81D" in HEX  
BINBUFF DB 01001100B, 01011010B, 11110010B, 00111110B, 01111001B, 10111000B, 00011101B
```

```
Correct DB 13, 10, "DESIDERATA: ",  
OKHEXBUFF db "4C5AF23E79B81D", 13, 10, '$'
```

```
Recoded_1st DB "RICODIFICATA (1st): ",  
HEXBUFF_1st db 2*Nbin dup(?), 13, 10, '$'
```

```
Recoded_2nd DB "RICODIFICATA (2nd): ",
```

```
HEXBUFF_2nd db 2*Nbi n dup(?), 13, 10, '$'
```

```
Recorded_3rd DB "RI CODI FI CATA (3rd): ",  
HEXBUFF_3rd db 2*Nbi n dup(?), 13, 10, '$'
```

```
HexValues db "0123456789ABCDEF" ; N.B. usata solo nella terza variante
```

```
offset_1st dw ? ; queste sono word perche' devono ospitare effective addresses  
offset_2nd dw ?  
offset_3rd dw ?
```

```
DATI ENDS
```

```
-----  
; macro di stampa a video  
-----  
di spl ay macro xxxx ; N.B. ogni stringa deve terminare con '$'  
    push dx  
    push ax  
    mov dx, offset xxxx  
    mov ah, 9  
    int 21h  
    pop ax  
    pop dx  
endm  
-----  
;
```

```
-----  
Codice SEGMENT PUBLIC 'CODE'
```

```
-----  
; MAIN (programma chiamante - serve per tutte e tre le versioni)  
-----
```

```
MAIN PROC FAR
```

```
    ASSUME cs: codice, DS: dati, ss: pila;
```

```
    MOV AX, seg DATI ; necessari 2 trasferimenti per assegnare a DS  
    MOV DS, AX ; l'indirizzo del segmento dati in modo esplicito
```

```
    di spl ay correct ; DISPLAY DELLA STRINGA desiderata
```

```
    mov bx, 0 ; indice della stringa BINBUFF da ricodificare
```

```
    mov offset_1st, offset hexbuff_1st  
    mov offset_2nd, offset hexbuff_2nd  
    mov offset_3rd, offset hexbuff_3rd
```

ci cl o\_pri nci pal e:

```
MOV AL, BINBUFF[BX]
```

```
push ax ; passaggio alla subroutine 1st del dato da ricodificare  
mov dx, offset_1st ;  
push dx ; passaggio alla subroutine 1st dell'indirizzo del risultato
```

```
call near ptr bin2hex_1st
```

```
add sp, 4 ; ripristino di sp al valore che aveva all'inizio del ciclo,  
; prima del salvataggio dei parametri (dx, ax)
```

```
push ax ; passaggio alla subroutine 2nd del dato da ricodificare  
mov dx, offset_2nd ;  
push dx ; passaggio alla subroutine 2nd dell'indirizzo del risultato
```

```
call far ptr bin2hex_2nd ; N.B. dopo questa non serve add sp, 4 (cfr. l'argomento della RET)
```

```
push ax ; passaggio alla subroutine 3rd del dato da ricodificare  
mov dx, offset_3rd ;  
push dx ; passaggio alla subroutine 3rd dell'indirizzo del risultato
```

```
call far ptr bin2hex_3rd ;
```

```
INC BX  
CMP BX, Nbin ; condizione di terminazione ciclo principale  
JE finish
```

```
add offset_1st, 2 ; si incrementa il puntatore a HEXBUFF_1st dei due caratteri scritti  
add offset_2nd, 2 ; si incrementa il puntatore a HEXBUFF_2nd dei due caratteri scritti  
add offset_3rd, 2 ; si incrementa il puntatore a HEXBUFF_3rd dei due caratteri scritti
```

```
JMP ci cl o_pri nci pal e
```

fi ni sh:

```
DISPLAY Recoded_1st ; si stampa la stringa ricodificata con la subroutine 1st  
DISPLAY Recoded_2nd ; si stampa la stringa ricodificata con la subroutine 2nd  
DISPLAY Recoded_3rd ; si stampa la stringa ricodificata con la subroutine 3rd  
; (devono risultare identiche alla stringa "correct")
```

```
MOV AH, 4CH ; si restituisce il controllo al DOS  
INT 21h
```

```
MAIN ENDP
```

```
-----
```

```
-----  
; Subroutine B2h prima versione  
-----
```

```
bin2hex_1st proc near
```

```
    push ax  
    push cx  
    push si  
    push bp
```

```
    mov cl, 4  
    mov bp, sp  
    add bp, 2*6 ; si scende lungo lo stack di 6 parole. Infatti, il contenuto dello stack  
                ; e' qui {bp, si, cx, ax; IP_return; dx_chi amante, ax_chi amante}
```

```
    mov ax, [bp] ; in al il byte da riconvertire: AL=ALH, ALL  
    mov ah, al  
    shr ah, cl ; in ah andra' la versione ricodificata di ALH  
    cmp ah, 9  
    jg lettera_ah
```

```
ci fra_ah:
```

```
    or ah, 30h  
    jmp check_al
```

```
lettera_ah: ; N.B. se il dato binario da convertire e' in [00001010B=0Ah, 00001111B=0Fh]  
    sub ah, 9 ; allora l'ASCII esadecimale corrispondente e' in [41h='A', 46h='F']  
    or ah, 40h ; quindi bisogna sottrarre 9 e aggiungere 40h
```

```
check_al:
```

```
    and al, 0fh ; in al andra' la versione ricodificata di ALL  
    cmp al, 9  
    jg lettera_al
```

```
ci fra_al:
```

```
    or al, 30h  
    jmp write2mem
```

```
lettera_al:
```

```
    sub al, 9  
    or al, 40h
```

```
write2mem:
```

```
    mov si, [bp-2] ; indirizzo dove salvare i due byte ricodificati  
    mov [si], ah  
    mov [si+1], al
```

```
    pop bp  
    pop si  
    pop cx  
    pop ax
```

```
    ret
```

```
bin2hex_1st endp
```

```
-----
```

```

;-----
; Macro usata solo nella seconda versione di bin2hex (per la logica di funzionamento cfr. bin2hex_1st)
;-----
reg2hex macro reg          ; reg e' il parametro della macro
local  ci fra, lettera, esci ; le etichette usate all'interno di macro vanno dichiarate cosi'
    cmp reg, 9
    jg lettera
ci fra:
    or reg, 30h
    jmp esci
lettera:
    sub reg, 9
    or reg, 40h
esci :
endm
;-----

; Subroutine BIN2HEX seconda versione
;-----
bin2hex_2nd proc far

    push ax
    push cx
    push si
    push bp

    mov bp, sp
    add bp, 2*7 ; si scende lungo lo stack di 7 parole. Infatti, il contenuto dello stack
                ; e' qui {bp, si, cx, ax; IP_return; CS_return, dx_chi amante, ax_chi amante}

    mov ax, [bp] ; in al il byte da riconvertire: AL=ALH, ALL
    mov ah, al
    and ah, 0fh
    reg2hex ah ; in ah il carattere ricodificato
    mov cl, 4
    shr al, cl ; l'immediata shl al, 4 e' illecita
    reg2hex al ; in al il carattere ricodificato
    mov si, [bp-2] ; indirizzo dove salvare i due byte riconvertiti
    mov [si], ax ; N.B. per salvare una word, al e ah vanno scelti invertiti (LITTLE ENDIAN):
                ; ALH->AL, ALL->AH (l'alternativa e' salvare byte a byte, come fatto in bin2hex_1st)

    pop bp
    pop si
    pop cx
    pop ax

    ret 4 ; si riaggiusta sp, aggiungendo i 4 byte dei parametri, che non servono piu' in stack
bin2hex_2nd endp
;-----

```

-----  
; Subroutine BIN2HEX terza versione  
-----

bin2hex\_3rd proc far

push ax  
push bx  
push cx  
push si  
push bp

mov bp, sp  
add bp, 2\*8 ; si scende lungo lo stack di 8 parole. Infatti, il contenuto dello stack  
; e' qui {bp, si, cx, bx, ax; IP\_return; CS\_return, dx\_chi amante, ax\_chi amante}

mov bh, 0  
mov ax, [bp] ; in al il byte da riconvertire: AL=ALH, ALL  
mov ah, al  
and ah, 0fh

mov bl, ah ; in bx qui un numero da 0 a 15: si usa come indice per la tabella  
mov ah, HexValues[bx] ; in ah il valore ricodificato

mov cl, 4  
shr al, cl  
mov bl, al ; in bx qui un numero da 0 a 15: si usa come indice per la tabella  
mov al, HexValues[bx] ; in al il valore ricodificato

mov si, [bp-2] ; indirizzo dove salvare i due byte riconvertiti  
mov [si], ax ; N. B. ALH->AL, ALL->AH (cfr. bin2hex\_2nd)

pop bp  
pop si  
pop cx  
pop bx  
pop ax

ret 4

bin2hex\_3rd endp  
-----

Codice ENDS

END MAIN ; il programma comincia all'indirizzo di MAIN