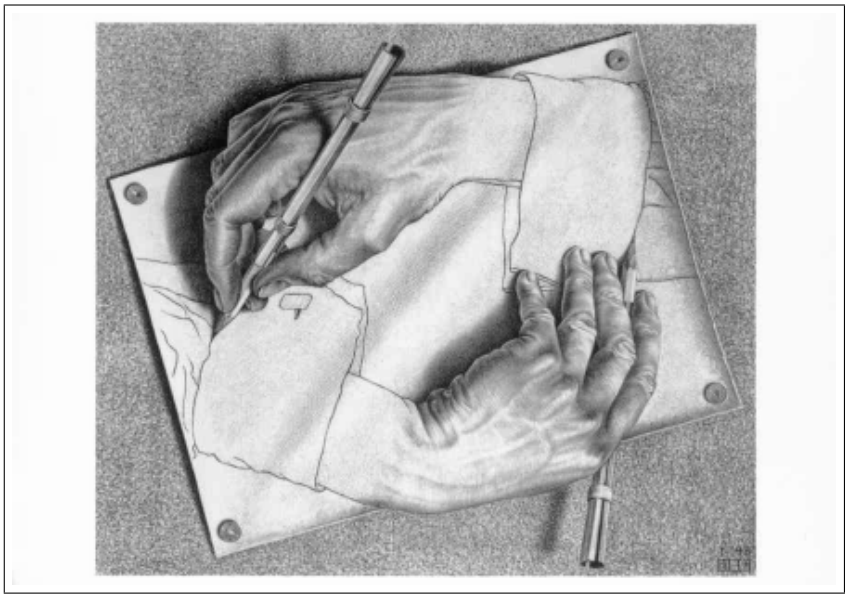
Corso di Laurea in Ingegneria Informatica

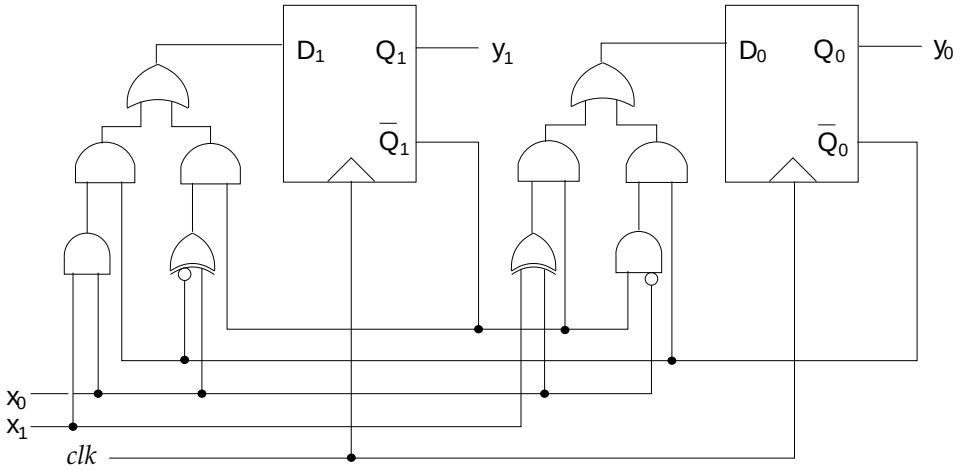
Calcolatori Elettronici I – a.a. 2002–2003

(prof. Carlo Colombo)

ESERCIZI D'ESAME



[#1/A-21/02/03] La figura mostra una macchina sequenziale sincrona di Moore con due ingressi (x_1x_0) e due uscite (y_1y_0).



Determinare:

(a_1) le equazioni di stato prossimo della macchina;

(a_2) l'evoluzione dello stato, a partire da $y_1y_0 = 00$, per ognuna delle possibili configurazioni di ingresso;

(a_3) il diagramma degli stati della macchina, segnalando le eventuali condizioni di indifferenza sugli ingressi.

soluzione

(a_1). Poiché la macchina è costruita con flip-flop di tipo D, il suo stato prossimo è dato dagli ingressi ai due flip-flop: $y_1' = D_1$ e $y_0' = D_0$. Inoltre, lo stato presente è dato dalle uscite dei flip-flop: $y_1 = Q_1$, $y_0 = Q_0$. Le due reti combinatorie a monte dei flip-flop forniscono le uscite:

$$\begin{aligned} y_1'(x_1, x_0; y_1, y_0) &= (x_1x_0) \cdot \overline{y}_0 + (x_0 \oplus y_0) \cdot \overline{y}_1 = x_1x_0\overline{y}_0 + x_0\overline{y}_1\overline{y}_0 + \overline{x}_0\overline{y}_1y_0 \ ; \\ y_0'(x_1, x_0; y_1, y_0) &= (x_1 \oplus x_0) \cdot \overline{y}_1 + (\overline{x}_0\overline{y}_1) \cdot \overline{y}_0 = x_1\overline{x}_0\overline{y}_1 + \overline{x}_1x_0\overline{y}_1 + \overline{x}_0\overline{y}_1\overline{y}_0 \ . \end{aligned}$$

(a_2). Fissata una delle quattro possibili configurazioni di ingresso $x_1x_0 = \{00, 01, 10, 11\}$, la macchina arriverà (dopo un'eventuale sequenza non periodica iniziale lunga tre stati al massimo) a ripercorrere ciclicamente una sequenza di stati di periodo non superiore a quattro. Infatti, con due variabili di stato, la macchina possiede *al più* quattro stati distinti.

Nel caso $x_1x_0 = 00$, le equazioni di stato prossimo si riducono a $y_1'(0, 0; y_1, y_0) = \overline{y}_1y_0$; $y_0'(0, 0; y_1, y_0) = \overline{y}_1\overline{y}_0$. Dunque, l'evoluzione dello stato a partire dalla configurazione $y_1y_0 = 00$ segue la tabella qui a fianco, ripetendo ciclicamente la sequenza (di periodo tre) $y_1y_0 = 00 \to 01 \to 10 \to 00 \ldots$

$x_1x_0 = 00$				
t	$y_1(t)$	$y_0(t)$	$y_1(t+1)$	$y_0(t+1)$
0	0	0	0	1
1	0	1	1	0
2	1	0	0	0

Nel caso $x_1x_0 = 01$, abbiamo $y_1'(0,1;y_1,y_0) = \overline{y}_1\overline{y}_0$; $y_0'(0,1;y_1,y_0) = \overline{y}_1$, da cui $y_1y_0 = 00 \to 11 \to 00\ldots$ (sequenza di periodo due). Nel caso $x_1x_0 = 10$, abbiamo $y_1'(1,0;y_1,y_0) = \overline{y}_1y_0$; $y_0'(1,0;y_1,y_0) = \overline{y}_1 + \overline{y}_1\overline{y}_0 = \overline{y}_1$, da cui $y_1y_0 = 00 \to 01 \to 11 \to 00\ldots$ (periodo tre). Infine, nel caso $x_1x_0 = 11$, abbiamo $y_1'(1,1;y_1,y_0) = \overline{y}_0 + \overline{y}_1\overline{y}_0 = \overline{y}_0$; $y_0'(1,1;y_1,y_0) = 0$, da cui $y_1y_0 = 00 \to 10 \to 10\ldots$ (periodo uno). Le tabelle relative agli ultimi tre casi sono

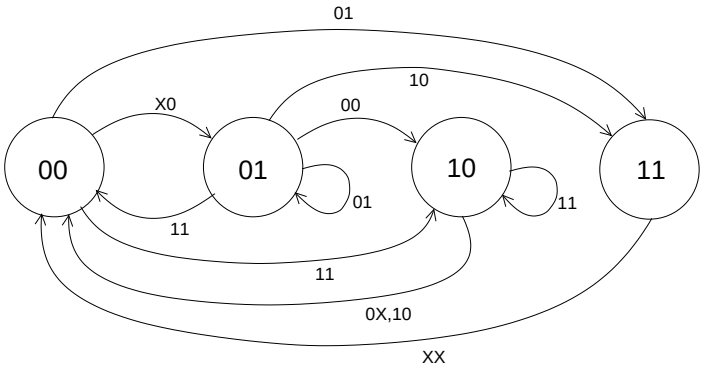
$x_1x_0 = 01$		
t	$y_1(t)\,y_0(t)$	$y_1(t+1)\,y_0(t+1)$
0	0 0	1 1
1	1 1	0 0

$x_1x_0 = 10$		
t	$y_1(t)\,y_0(t)$	$y_1(t+1)\,y_0(t+1)$
0	0 0	0 1
1	0 1	1 1
2	1 1	0 0

$x_1x_0 = 11$		
t	$y_1(t)\,y_0(t)$	$y_1(t+1)\,y_0(t+1)$
0	0 0	1 0
1	1 0	1 0

(a_3). Per poter disegnare il diagramma degli stati della macchina, bisogna conoscere il valore dello stato prossimo per *tutte* le possibili combinazioni di ingresso e di stato presente. Nel nostro caso, le combinazioni sono $2^2 \times 2^2 = 16$. Dalle tabelle ricavate al punto (a_2) conosciamo il valore dello stato prossimo per $3 + 2 + 3 + 2 = 10$ combinazioni ingresso/stato. I rimanenti $16 - 10 = 6$ valori, corrispondenti agli stati non raggiunti nelle sequenze che partono dallo stato $y_1y_0 = 00$, vanno desunti dalle equazioni generali ricavate al punto (a_1). La tabella della verità riportata a lato (ottenuta integrando i dieci valori già calcolati precedentemente coi sei valori mancanti, indicati da un asterisco) caratterizza completamente il comportamento della macchina, fornendone una rappresentazione equivalente alle equazioni di stato prossimo.

	x_1	x_0	y_1	y_0	y_1'	y_0'
	0	0	0	0	0	1
	0	0	0	1	1	0
$*$	0	0	1	0	0	0
	0	0	1	1	0	0
	0	1	0	0	1	1
$*$	0	1	0	1	0	1
$*$	0	1	1	0	0	0
	0	1	1	1	0	0
	1	0	0	0	0	1
	1	0	0	1	1	1
$*$	1	0	1	0	0	0
	1	0	1	1	0	0
	1	1	0	0	1	0
$*$	1	1	0	1	0	0
	1	1	1	0	1	0
$*$	1	1	1	1	0	0



Il diagramma degli stati fornisce un'ulteriore rappresentazione della macchina, equivalente alle due precedenti (equazioni di stato prossimo, tabella della verità). Il diagramma qui a lato è stato ottenuto dalla tabella della verità. In alternativa, esso avrebbe potuto essere ricavato dalle equazioni generali di stato prossimo al punto (a_1), ed utilizzato per risolvere il punto (a_2).

[#1/B-21/02/03] Una CPU a singolo bus interno colloquia con la memoria attraverso un bus indirizzi a 16 bit e un bus dati a 8 bit. Il set di istruzioni di macchina include l'istruzione `PUSH VAR`, dove `VAR` è una variabile di tipo word (2 byte). Considerando l'intero ciclo fetch-esecuzione dell'istruzione, riportare:

(b_1) l'hardware di CPU coinvolto;

(b_2) la sequenza dei passi di controllo necessari;

(b_3) il numero di CPI (clock cycles per instruction) e di accessi al bus richiesti.

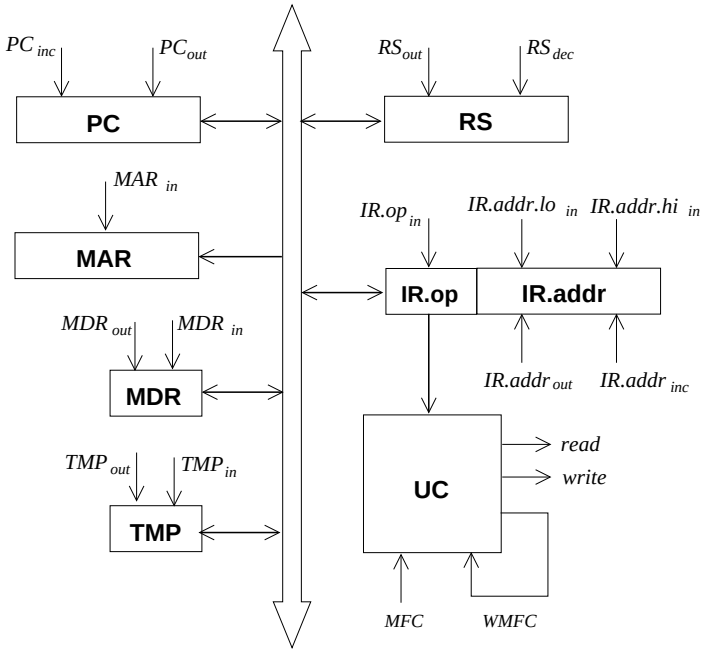
Ipotizzare che il codice operativo occupi il primo byte dell'istruzione, e che lo stack sia puntato dal registro `RS`.

soluzione

Siccome il bus dati è a 8 bit, per completare la fase di fetch dell'istruzione servono tre successive letture dalla memoria. Infatti, nell'ipotesi che il codice operativo occupi (insieme ai bit necessari a codificare gli altri eventuali modi di indirizzamento dell'operando, ad es. il modo registro in `PUSH <registro>`) il primo byte, servono altri due byte per codificare l'indirizzo a 16 bit di `VAR`, e quindi l'istruzione è lunga complessivamente 3 byte. Il significato dell'istruzione `PUSH VAR` è "preleva il contenuto della locazione di memoria `VAR` e mettilo in cima allo stack". Dato che l'operando è di 16 bit, servono (fase di esecuzione) due cicli di lettura per prelevarlo dalla memoria, e due di scrittura per copiarlo nello stack. Il numero totale di accessi alla memoria è dunque di 3 (instruction fetch) + 2 (operand load) + 2 (operand store) = 7. Questo risolve la seconda parte del quesito (b_3). Riguardo alla prima parte di (b_3), è sufficiente (nell'ipotesi che la memoria sia abbastanza veloce da non introdurre cicli di attesa in CPU) contare i passi della sequenza di controllo ottenuta come soluzione del quesito (b_2).

(b_1, b_2). Si danno due diverse soluzioni: la seconda richiede meno CPI e un hardware semplificato rispetto alla prima. In entrambe le soluzioni, si fa uso di registri contatori per incrementare (`PC`) o decrementare (`RS`) il valore del registro senza passare dalla `ALU`. Poiché il nostro stack cresce verso indirizzi bassi, il relativo registro contatore viene decrementato in un "push", e incrementato in un "pop". Notare che i byte dell'operando vanno salvati nello stack nell'ordine inverso a quello usato per leggerli, ossia `VAR[0]` $\longrightarrow$ `[RS-2]`, `VAR[1]` $\longrightarrow$ `[RS-1]`.

Nella prima soluzione si fa uso di un registro dati temporaneo `TMP` da 8 bit. Si adopera inoltre un registro indirizzi temporaneo di 16 bit `IR.addr`, suddiviso in due campi da 8 bit e considerato come campo indirizzo del registro di istruzione `IR`.

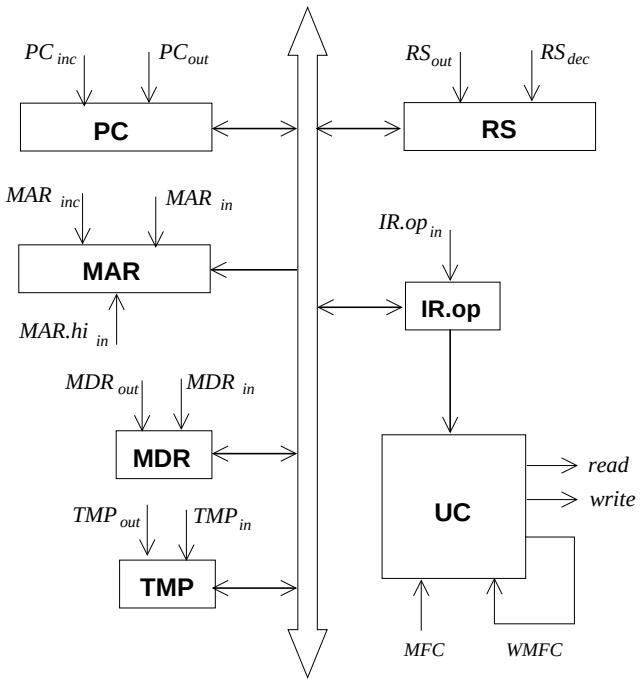



```
//fetch dell'istruzione
```
1. $\texttt{PC}_{out}$, $\texttt{MAR}_{in}$, $read$, $WMFC$;
2. $\texttt{MDR}_{out}$, $\texttt{IR.op}_{in}$, $\texttt{PC}_{inc}$;
3. $\texttt{PC}_{out}$, $\texttt{MAR}_{in}$, $read$, $WMFC$;
4. $\texttt{MDR}_{out}$, $\texttt{IR.addr.hi}_{in}$, $\texttt{PC}_{inc}$;
5. $\texttt{PC}_{out}$, $\texttt{MAR}_{in}$, $read$, $WMFC$;
6. $\texttt{MDR}_{out}$, $\texttt{IR.addr.lo}_{in}$, $\texttt{PC}_{inc}$;

```
//caricamento dell'operando
```
7. $\texttt{IR.addr}_{out}$, $\texttt{MAR}_{in}$, $read$, $WMFC$;
8. $\texttt{MDR}_{out}$, $\texttt{TMP}_{in}$, $\texttt{IR.addr}_{inc}$, $\texttt{RS}_{dec}$;
9. $\texttt{IR.addr}_{out}$, $\texttt{MAR}_{in}$, $read$, $WMFC$;

```
//salvataggio nello stack
```
10. $\texttt{RS}_{out}$, $\texttt{MAR}_{in}$, $write$, $WMFC$;
11. $\texttt{TMP}_{out}$, $\texttt{MDR}_{in}$, $\texttt{RS}_{dec}$;
12. $\texttt{RS}_{out}$, $\texttt{MAR}_{in}$, $write$, $WMFC$;

Nella seconda soluzione si fa uso di un registro dati/indirizzi temporaneo `TMP` da 8 bit, e si elimina il registro `IR.addr`, facendone svolgere le funzioni dalla coppia di registri `MAR`, `TMP`. Il registro `MAR` deve potere essere incrementato (con $\texttt{MAR}_{inc}$), e scritto in tutti i suoi 16 bit (con $\texttt{MAR}_{in}$) o solo nei suoi 8 bit più significativi (con $\texttt{MAR.hi}_{in}$).




```
//fetch dell'istruzione
```
1. $\texttt{PC}_{out}$, $\texttt{MAR}_{in}$, $read$, $WMFC$;
2. $\texttt{MDR}_{out}$, $\texttt{IR.op}_{in}$, $\texttt{PC}_{inc}$;
3. $\texttt{PC}_{out}$, $\texttt{MAR}_{in}$, $read$, $WMFC$;
4. $\texttt{MDR}_{out}$, $\texttt{TMP}_{in}$, $\texttt{PC}_{inc}$;
5. $\texttt{PC}_{out}$, $\texttt{MAR}_{in}$, $read$, $WMFC$;
6. $\texttt{MDR}_{out}$, $\texttt{MAR}_{in}$, $\texttt{PC}_{inc}$;

```
//caricamento dell'operando
```
7. $\texttt{TMP}_{out}$, $\texttt{MAR.hi}_{in}$, $read$, $WMFC$;
8. $\texttt{MDR}_{out}$, $\texttt{TMP}_{in}$, $\texttt{MAR}_{inc}$,
   $\texttt{RS}_{dec}$, $read$, $WMFC$;

```
//salvataggio nello stack
```
9. $\texttt{RS}_{out}$, $\texttt{MAR}_{in}$, $write$, $WMFC$;
10. $\texttt{TMP}_{out}$, $\texttt{MDR}_{in}$, $\texttt{RS}_{dec}$;
11. $\texttt{RS}_{out}$, $\texttt{MAR}_{in}$, $write$, $WMFC$;

[#1-#2/D-21/02/03] Per esigenze di visualizzazione dati, si rende necessario convertire il contenuto binario di una parola di 8 bit in formato ASCII binario [esadecimale]. Ad esempio, la conversione del byte 01101010 produce la stringa di 8 byte `'01101010'` [la stringa di 2 byte `'6A'`].

(d_1) Scrivere una procedura assembly 8086, "`Bin2ABin`" ["`Bin2AHex`"], che converta il contenuto del registro `AL` in una stringa il cui indirizzo di memoria è contenuto in `BX`;

(d_2) modificare la procedura del punto precedente, in modo che la nuova procedura "`Bin2ABinS`" ["`Bin2AHexS`"] prelevi dallo stack, alla chiamata, sia il valore del byte da convertire che l'indirizzo (offset) della stringa;

(d_3) dopo avere definito opportunamente i segmenti codice, dati e stack del programma, scrivere il codice del chiamante che, facendo uso di `Bin2ABinS` [`Bin2AHexS`], consenta di convertire il byte di memoria `BinVAR` nella stringa `BinBUF` [`HexBUF`].

[*Suggerimento: servirsi dell'istruzione assembly* `SHR <registro>,CL`*, che realizza lo scorrimento a destra del contenuto del registro del numero di bit specificato in* `CL`.]

soluzione

Le due procedure richieste al punto (d_1) vengono descritte dettagliatamente nella prossima pagina. A seguire è riportato, con i necessari commenti, il programma completo richiesto al punto (d_3), che include le varianti del punto (d_2). Per quanto riguarda il significato della tabella ("look-up table") di supporto `HexTab` utilizzata nella procedura di conversione in ASCII esadecimale, cfr. la sua definizione nel segmento dati del programma completo.

(d_1).

```
Bin2ABin PROC NEAR

INIZ:
    MOV AH,AL   ; copia il carattere da convertire nel registro temporaneo AH
    ADD BX,7    ; serve a scrivere la stringa a partire dal byte meno significativo
    MOV CL,1    ; serve a fare lo shift di un bit nell'istruzione SHR
    MOV DH,0    ; contatore di ciclo: non si usa CX, perch CL e' gi impegnato altrimenti
CICLO:
    AND AH,01H  ; si azzerano tutti i bit di AH tranne il meno significativo
    CMP AH,0    ; confronto tra AH e 0 (il risultato viene scritto nel flag ZF)
    JE ZERO     ; se AH=0 si salta all'etichetta ZERO
    MOV DL,'1'  ; AH=1: si pone il carattere ASCII '1' in DL
    JMP MEMO    ; salta all'etichetta MEMO
ZERO:
    MOV DL,'0'  ; AH=0: si pone il carattere ASCII '0' in DL
MEMO:           ; memorizzazione del contenuto di DL nella stringa puntata da BX
    MOV [BX],DL ; il contenuto di DL e' posto in DS:BX (con BX=7,6,...,0)
    DEC BX      ; BX viene decrementato, per puntare a un byte piu' significativo
    SHR AL,CL   ; il contenuto di AL e' diviso per due (scorrimento a destra di 1 bit)
    MOV AH,AL   ; copia del nuovo valore di AL nel registro temporaneo AH
    INC DH      ; il contatore di ciclo viene incrementato...
    CMP DH,8    ; ...e confrontato col suo valore limite
    JNE CICLO   ; si salta all'etichetta CICLO se DH e' inferiore a 8

    RET         ; ritorno dalla procedura: siccome e' NEAR, viene modificato il solo IP

Bin2ABin ENDP


Bin2AHex PROC FAR

INIT0:
    PUSH AX           ; salvataggio nello stack, per uso futuro, del byte da convertire
    MOV CL,4          ; serve a fare lo shift di quattro bit nell'istruzione SHR
    MOV DI,1          ; contatore per il ciclo esterno (due sole iterazioni: DI=1, DI=0)
LOOP0:
INIT1:
    AND AX,000FH      ; si isola il "nibble" (semiparola di 4 bit) meno significativo di AL
    MOV SI,0          ; contatore per il ciclo esterno (da una a 16 possibili iterazioni)
LOOP1:
    CMP AX,SI         ; confronto di SI col valore del "nibble" da convertire
    JE FOUND          ; se AX=SI, si salta all'etichetta FOUND
    INC SI            ; altrimenti, si incrementa SI...
    JMP LOOP1         ; ...e si esegue una nuova iterazione del ciclo interno
FOUND:                ; il carattere ASCII corrispondente al "nibble" e' prelevato...
    MOV DL,HexTab[SI] ; ...dalla posizione SI della tabella di supporto HexTab,...
    MOV [BX][DI],DL   ; ...e salvato in memoria all'indirizzo DS:(BX+DI)
    DEC DI            ; il contatore esterno viene decrementato...
    CMP DI,0          ; ...e confrontato con 0 (valore limite)
    JL EXIT           ; se DI  minore di 0, si esce dal ciclo esterno, altrimenti...
    POP AX            ; ...e' DI=0: si recupera dallo stack il byte da convertire...
    SHR AL,CL         ; ...e se ne riporta il "nibble" piu' significativo nei 4 LSB di AL
    JMP LOOP0         ; si esegue il secondo ciclo di LOOP0
EXIT:
    RET               ; ritorno dalla procedura: siccome  FAR, sono modificati sia IP che CS

Bin2AHex ENDP
```

(d_2, d_3).

```
;******************************** STACK ***********************************************
STACK SEGMENT STACK 'STACK'    ; definizione del segmento di stack

      DB      64 DUP('S')      ; 64 caratteri inizializzati con 'S'

STACK ENDS

;******************************** DATI ************************************************
DATA SEGMENT PUBLIC 'DATA'           ; definizione segmento dati

    BinVAR    DB 01101010B           ; byte da convertire
    BinBUF    DB 8 DUP ('?')         ; stringa ASCII binario
    HexTab    DB '0123456789ABCDEF'  ; tabella di supporto
    HexBUF    DB 2 DUP ('?')         ; stringa ASCII esadecimale

DATA ENDS

;******************************** CODICE **********************************************
CSEG SEGMENT PUBLIC 'CODE'

MAIN PROC FAR                        ; il programma principale e' una procedura FAR
      ASSUME CS:CSEG,DS:DATA,SS:STACK;   ; direttiva per l'assemblatore

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

                                     ; CONVERSIONE BINARIO - ASCII BINARIO
         MOV AL,BinVAR               ; il byte da convertire e' posto in AL...
         PUSH AX                     ; ...e poi salvato nello stack
         MOV BX,OFFSET BinBUF        ; lo spiazzamento della stringa BinBUF e' posto in BX...
         PUSH BX                     ; ...e poi salvato nello stack
         CALL NEAR PTR Bin2ABinS     ; chiamata alla procedura (NEAR) Bin2ABinS
         ADD SP,4                    ; il puntatore allo stack viene ripristinato...
                                     ; ...al valore che aveva prima dei due salvataggi...
                                     ; ...che precedono la chiamata a procedura

                                     ; CONVERSIONE BINARIO - ASCII ESADECIMALE
         MOV AL,BinVAR               ;
         PUSH AX                     ;
         MOV BX,OFFSET HexBUF        ; (come sopra. L'unica differenza e' che qui la
         PUSH BX                     ; procedura Bin2AHexS e' FAR)
         CALL FAR PTR Bin2AHexS      ;
         ADD SP,4                    ;

         mov ah,4ch                  ; uso dell'interrupt 21H (funzione 4Ch)
         int 21h                     ; per restituire il controllo al DOS

MAIN ENDP                            ; fine della procedura MAIN (direttiva per l'assemblatore)
```

```
;------------------------------ Bin2ABinS ----------------------------------------
Bin2ABinS proc near

        MOV BP,SP              ; si copia SP in BP per accedere allo stack come RAM

        MOV AL,[BP+4]          ; il byte da convertire e' "sepolto" sotto due word
        MOV AH,AL              ; da 16 bit: (1) l'offset (rispetto al segmento di codice)
        MOV BX,[BP+2]          ; di ritorno dalla procedura NEAR (che e' posto in SS:SP,
        ADD BX,7               ; e andra' in IP), e l'offset (rispetto al segmento dati)
        MOV CL,1               ; della stringa che accogliera' il risultato (e' in SS:(SP+2))
        MOV DH,0
CICLO:
        AND AH,01H
        CMP AH,0
        JE ZERO
        MOV DL,'1'
        JMP SCRIVI
ZERO:   MOV DL,'0'
SCRIVI: MOV [BX],DL
        DEC BX
        SHR AL,CL
        MOV AH,AL
        INC DH
        CMP DH,8
        JNE CICLO

        ret

Bin2ABinS endp



;------------------------------ Bin2AHexS ----------------------------------------
Bin2AHexS proc far

        MOV BP,SP              ; come sopra, solo che qui l'indirizzo di ritorno
                               ; da procedura FAR e' completo, e quindi occupa non 1,
        MOV BX,[BP+4]          ; ma due byte in stack (i byte da SS:SP a SS:(SP+3))
        MOV AL,[BP+6]
INITO:
        PUSH AX
        MOV CL,4
        MOV DI,1
LOOP0:
INIT1:
        AND AX,000FH
        MOV SI,0
LOOP1:
        CMP AX,SI
        JE FOUND
        INC SI
        JMP LOOP1
FOUND:  MOV DL,HexTab[SI]
        MOV [BX][DI],DL
        DEC DI
        CMP DI,0
        JL EXIT
        POP AX
        SHR AL,CL
        JMP LOOP0
EXIT:
        RET

Bin2AHexS ENDP



CSEG ENDS                      ; fine del segmento di codice (direttiva per l'assemblatore)

END MAIN                       ; il programma comincia all'indirizzo di MAIN
```