

Specificare la Scatola con il ? e scrivere un programma che

- 1) prelevi il byte contenuto nella prima locazione e della EPROM e lo mandi nel LATCH
- 2) aspetti un tempo pari a circa 1000 istruzioni
- ...
- 7) prelevi il byte contenuto nella quarta locazione e della EPROM e lo mandi nel LATCH
- 8) aspetti un tempo pari a circa 1000 istruzioni e torni al punto 1) e così via all'infinito

ATTENZIONE: La EPROM è nello Spazio di I/O

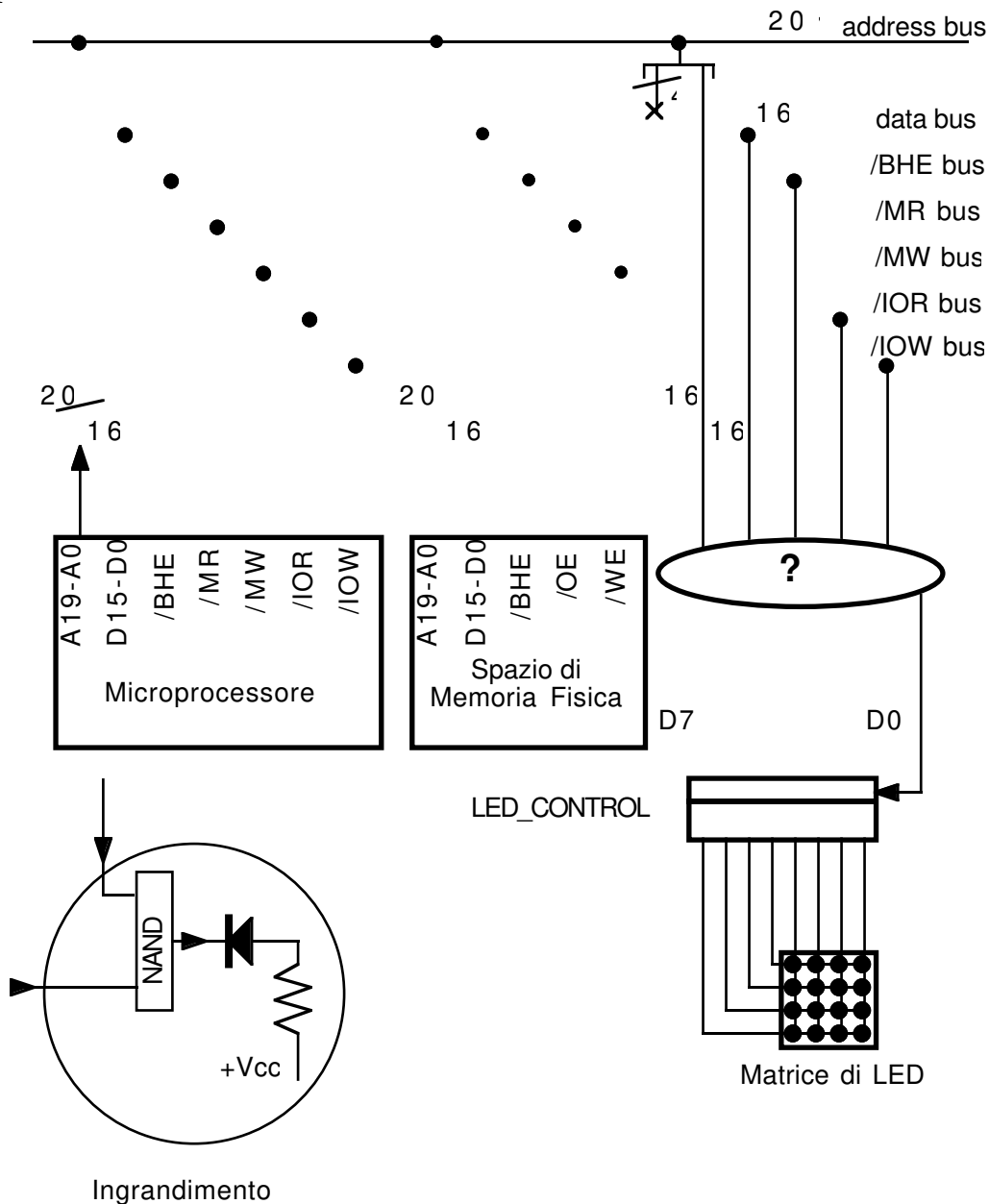
Con riferimento al sistema in figura:

1- specificare la logica contrassegnata dal ? in modo che l'istruzione:

```
MOV    DX, 7FFFH
OUT IO:[DX], AL
```

immetta nel registro LED_CONTROL il contenuto di AL.

2- scrivere un programma che prima spenga tutti il LED della matrice, poi accenda i 4 LED costituenti il quadrato interno ed infine accenda tutti e 16 i LED e così via all'infinito; ognuna delle tre configurazioni previste per i LED della matrice deve durare un tempo pari a circa 1000 istruzioni.



Scrivere la fase di chiamata di un processore che:

- ha uno spazio di memoria da 64 K byte non segmentato e non ha spazio di I/O
- non ha registri generali del tipo AX, BX,SP ma tutti gli operandi stanno in memoria
- non ha lo stack
- lavora solo su operandi a 16 bit ed ha il seguente formato istruzioni

codice operativo	!
specifiche operando destinatario	
specifiche operando sorgente	

- 00 un operando (l'istruzione ha solo due parola);
destinatario: indirizzamento diretto
- 01 due operandi;
destinatario: indirizzamento diretto, sorgente: indirizzamento immediato.
- 11 due operandi
destinatario: indirizzamento diretto, sorgente: indirizzamento diretto
- 10 due operandi
destinatario: indirizzamento diretto, sorgente: indirizzamento indiretto
(la terza parola dell'istruzione contiene l'indirizzo dell'indirizzo dell'operando sorgente)

Scrivere un sottoprogramma che trova:

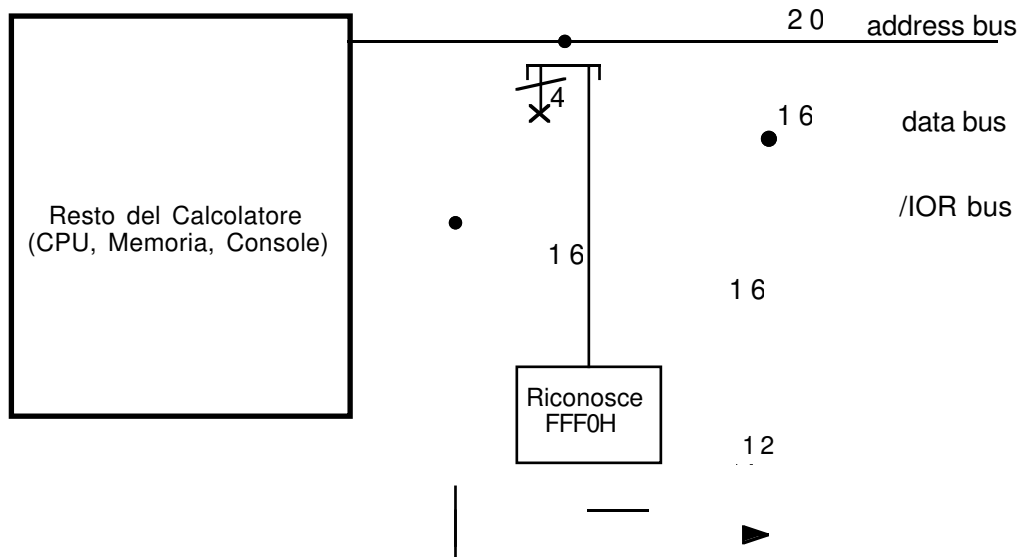
- in CL un integer ad 8 bit, chiamiamolo $e1$;
- in CH un integer ad 8 bit, chiamiamolo $e2$;
- in AX un integer a 16 bit, chiamiamolo $m1$;
- in BX un integer a 16 bit, chiamiamolo $m2$;

e lascia:

- in CL un integer ad 8 bit, chiamiamolo e ;
- in DX_AX un integer a 32 bit, chiamiamolo m

in modo che sia $m \cdot 2^e = (m1 \cdot 2^{e1}) \cdot (m2 \cdot 2^{e2})$

Opzionalmente prendere in considerazione il problema dell'eventuale overflow nel fare $e1 + e2$ ed i possibili artifici per sistemare, ove possibile, le cose.



Scrivere un sottoprogramma che aspetta che un utente prema un tasto sul dispositivo illustrato e invia al video della console la codifica ASCII di:

- 0 se l'utente ha premuto il tasto 0
- 1 se l'utente ha premuto il tasto 1
- 2 se l'utente ha premuto il tasto 2
- 3 se l'utente ha premuto il tasto 3

+Vcc

Sintetizzare, in accordo al modello basato sulle microistruzioni, l'unità con Parte Operativa e parte Controllo descritta di seguito:

DESCRIPTION

INPUTS

x : 1 bit variable.
bus : 16 bit variable.

REGISTERS

A, E, F : 1 bit registers.
B, G, D : 16 bit registers.

ATOMS

alpha(16 bits, 8 bits), beta(16 bits): 16 bits produced.
gamma(16 bits), delta(16 bits), theta(16 bits): 8 bits produced.

CONSTANTS

N is_for 0FFFEH: 16 bit constant.

BODY

```
L0:  F := 0 ; G := bus ; goto{if x eq 0 then L0 else L1}.
L1:  A := if beta(G) eq N then 1 else 0.
L2:  .
L3:  B := beta(G) ; A := 0 ; goto{if B eq N then L6 else L4}.
L4:  D := alpha(B, 2) ; goto{if gamma(B) ne delta(G) then L1 else L5}.
L5:  D := alpha(B, 2) ; F := 1.
L6:  E := 1 ; MJR := if x eq 1 then L5 else {if A eq F then L1 else L0}.
L7:  E := 0 ; F := 1 ; goto MJR.
```

END_BODY

END_DESCRIPTION

Scrivere un sottoprogramma che elabora il byte contenuto in AL e restituisce in BH e BL due byte: quello in BH è la codifica ASCII della cifra esadecimale costituita dai quattro bit in posizione pari in AL; quello in BL è la codifica ASCII della cifra esadecimale costituita dai quattro bit in posizione dispari in AL. Il bit più significativo di BH e BL è posto a 0.

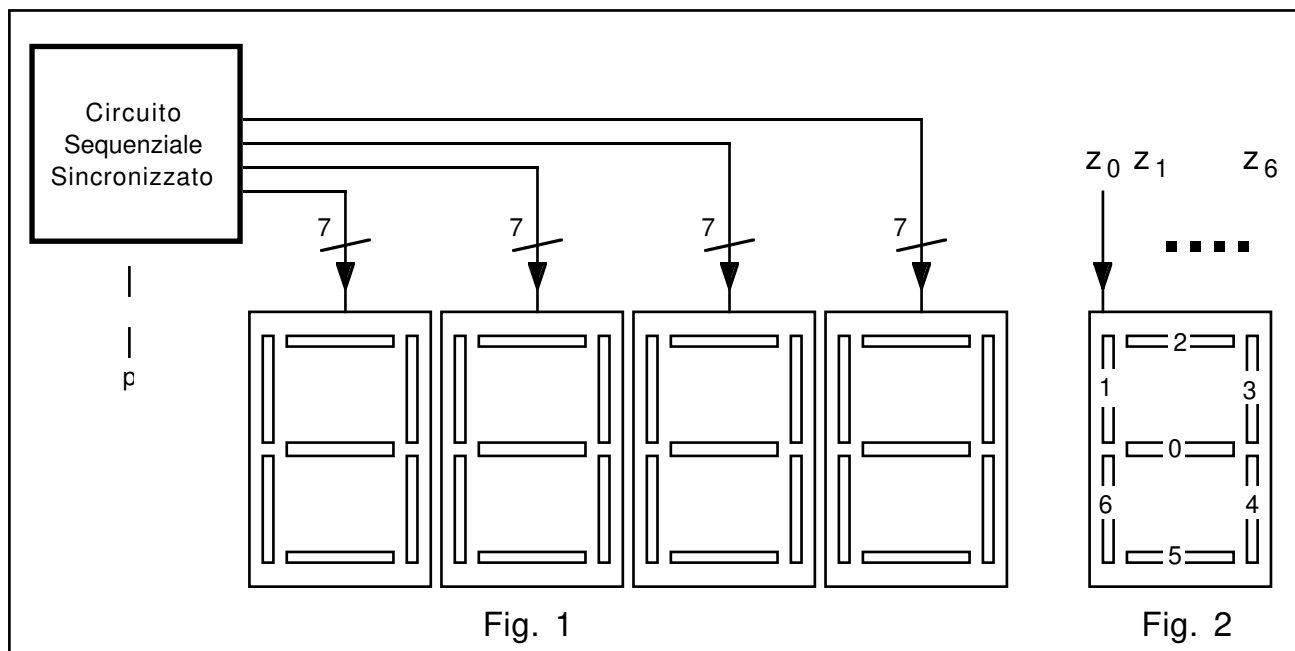
Scrivere il seguente programma:

```

          AL <- 0
CICLO:    Chiama il sottoprogramma di cui sopra
          Manda alla console BL, BH, CR, LF
          AL <- AL + 1
          Se AL diverso da 0 torna a CICLO
          EXIT
```

Il sottoprogramma elabora il byte contenuto di AL e restituisce in BH e BL due byte: quello in BH è la codifica ASCII della cifra esadecimale costituita dai quattro bit in posizione pari in AL; quello in BL è la codifica ASCII della cifra esadecimale costituita dai quattro bit in posizione dispari in AL. Il bit più significativo di BH e BL è posto a 0.

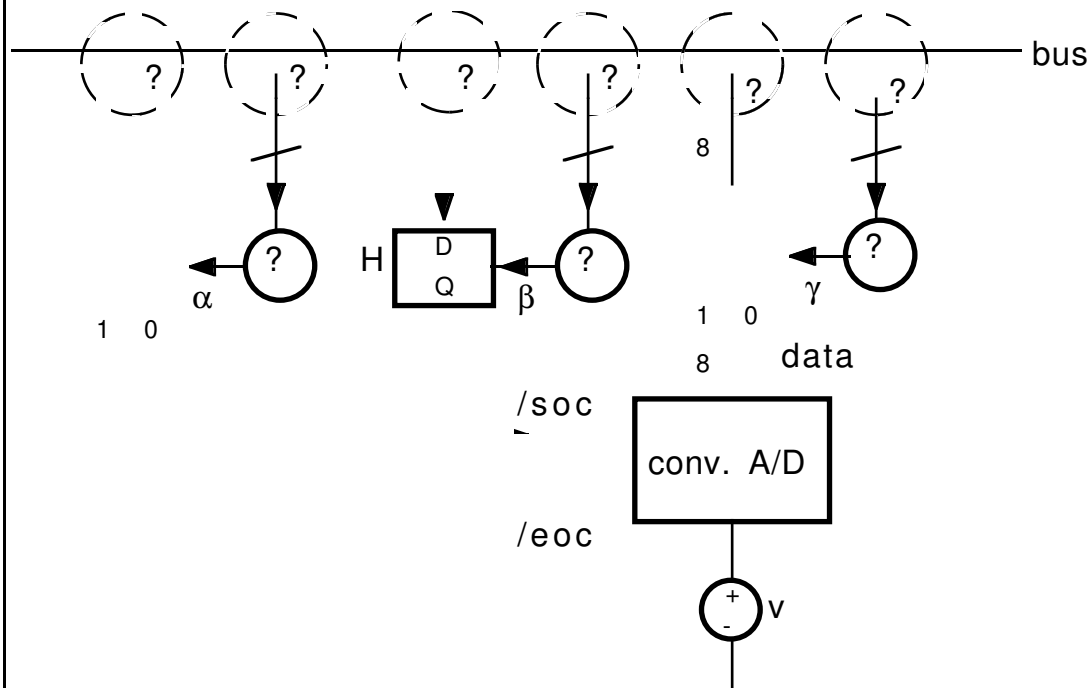
Con riferimento alla Fig. 1, progettare il Circuito Sequenziale Sincronizzato in modo che esso tenga spenti i 4 Display per 10 periodi di clock, poi li accenda uno dopo l'altro ad intervalli di 10 periodi di clock, poi li tenga accesi per 10 periodi di clock, poi li tenga spenti per 10 periodi di clock e così via all'infinito. Quando tutti e 4 i Display sono accesi si deve leggere la parola **FILA**



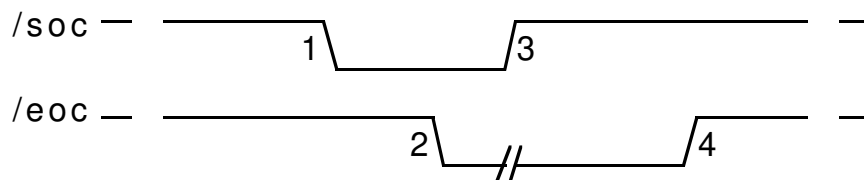
In un Display (vedi la Fig. 2) è acceso il tratto i-mo, ($i=0, 1, 2, 3, 4, 5, 6$), se la variabile z_i vale 1.

Precisare sia i collegamenti con il bus e che le reti contrassegnate con ? in modo che l'istruzione:

-) IN AL, IO:0200H porti il valore di /eoc nel bit meno significativo di AL;
-) IN AL, IO:0202H porti gli 8 bit *data* in AL;
-) OUT IO:0204H, AL porti il bit meno significativo di AL nel registro H.



Partendo da una situazione iniziale in cui /soc =1 e /eoc =1, il circuito *conv. A/D* opera come segue:



- 1) Il circuito *conv. A/D* riceve l'ordine di lavorare
- 2) Il circuito *conv. A/D* risponde che sta lavorando
- 3) Il circuito *conv. A/D* è ringraziato perché sta lavorando
- 4) Il circuito *conv. A/D* comunica di aver finito il lavoro

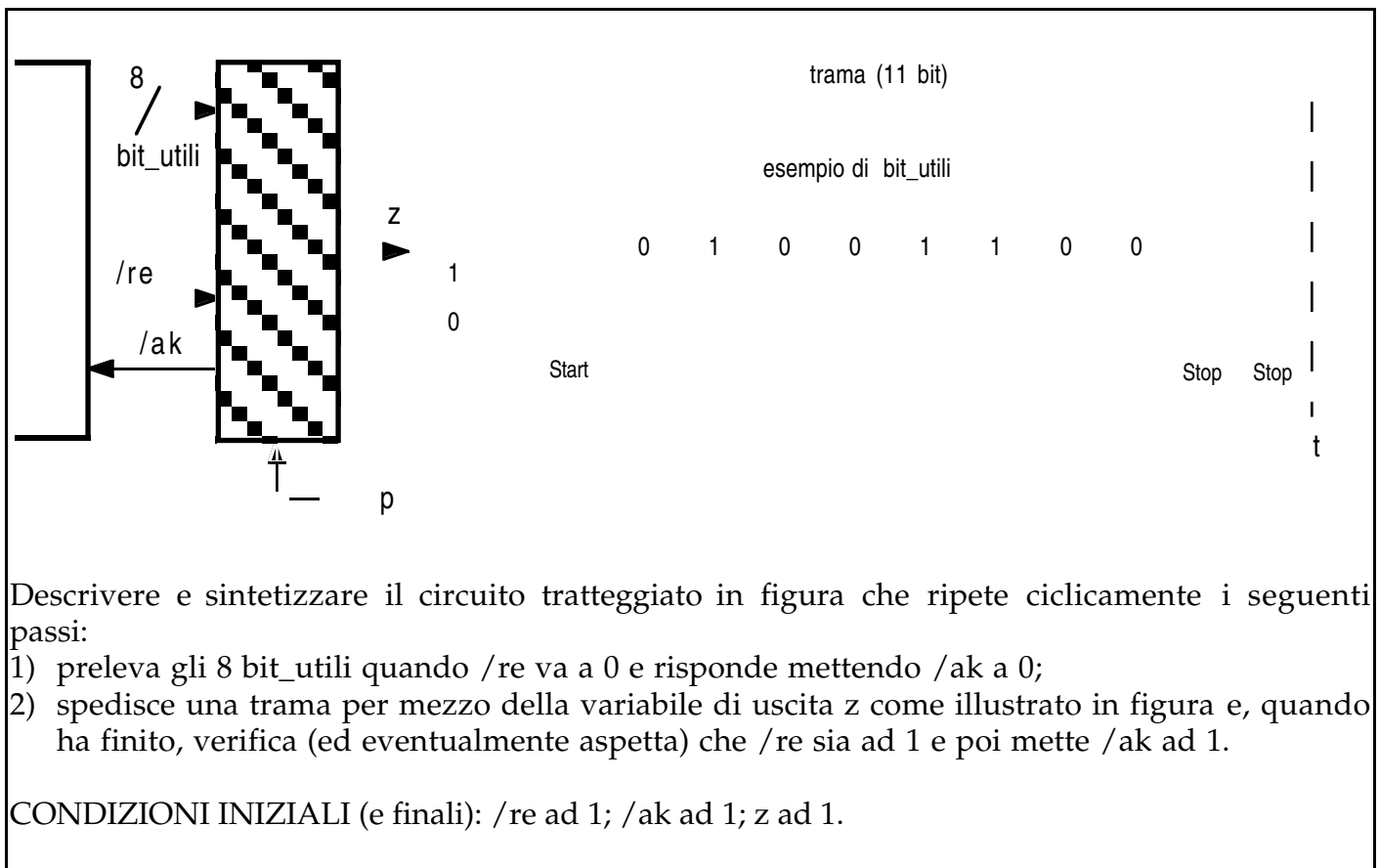
Il lavoro consiste nel preparare e presentare sugli 8 bit *data* il valore binario (senza segno) della tensione v in accordo alla relazione $data = |v / 10|$ con v da 0 mvolt a 2559 mvolt. Quando il *conv. A/D* non lavora conserva in uscita il risultato del lavoro precedente.

Scrivere un programma che

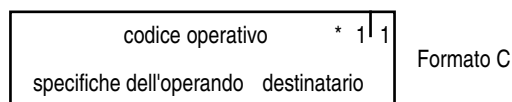
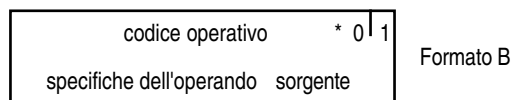
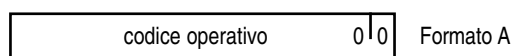
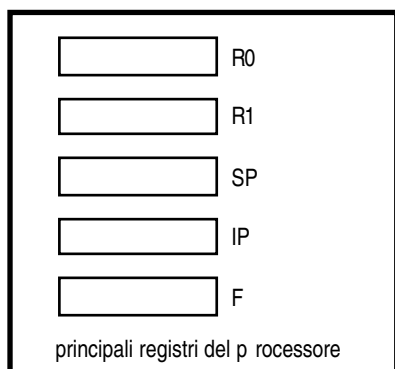
- a) preleva 100 campioni di v e ne calcola la media
- b) emette sulla console del calcolatore il messaggio

vm=

dove i puntini sono il valore medio in base dieci ed in millivolt.



Un processore, capace di lavorare solo su parole a 16 bit e di indirizzare una memoria lineare non segmentata da 64 Kbyte, ha i seguenti formati di istruzione:



* = 1 indirizzamento diretto

* = 0 indirizzamento indiretto (indirizzo dell'operando)

Durante la fase di reset, il processore inizializza SP (Stack Pointer) ed IP (Instruction Pointer) con il contenuto delle locazioni di indirizzo 0000H e 0002H ed azzerava il registro F (Flags).

Il formato A prevede istruzioni che lavorano sui registri R0 ed R1 secondo lo schema $R0 \leftarrow R0 \text{ operatore } R1$.

I formati B e C hanno istruzioni in cui l'operando può essere indirizzato in modo diretto o indiretto (non sono previste istruzioni di IO).

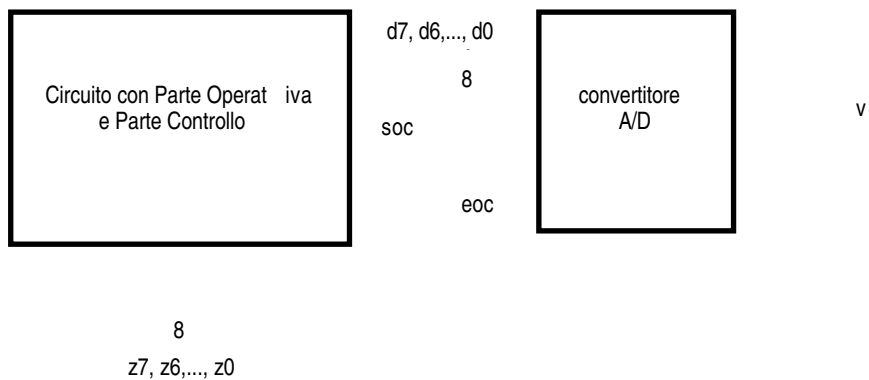
Scrivere la fase di reset, la fase di chiamata completa e la fase di esecuzione delle seguenti istruzioni (che non sono ovviamente le uniche possibili):

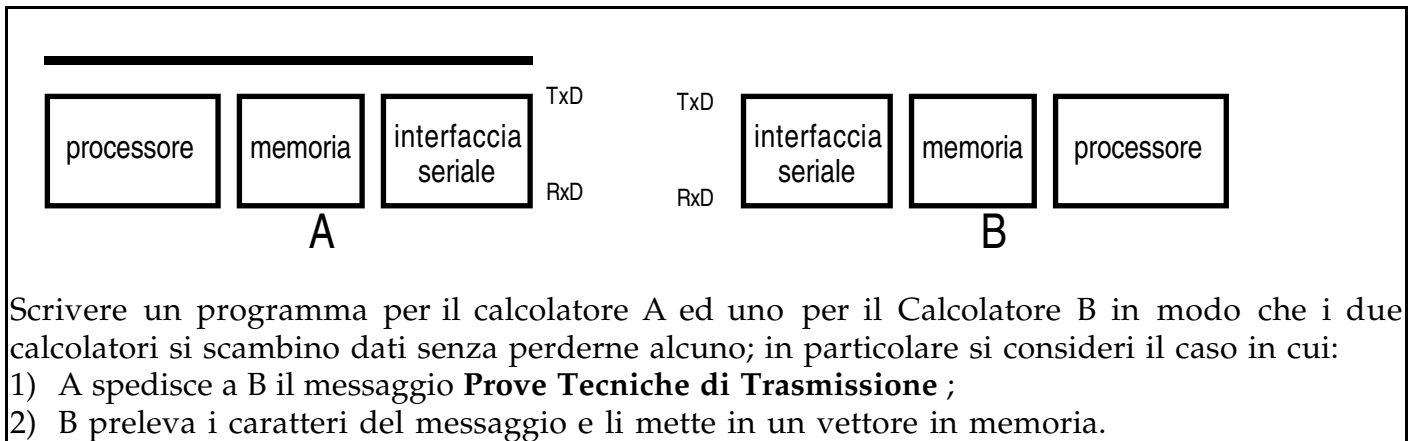
MOV R0, sorgente (formato B; $R0 \leftarrow$ operando sorgente)
 MOV destinatario, R0 (formato C; destinatario \leftarrow R0)
 POP (formato A; pop R0)
 ADD (formato A; $R0 \leftarrow R0 + R1$)

Scrivere un programma che simuli il processore di cui sopra: il segmento dati con selettore in ES sarà la memoria ed il segmento dati con selettore in DS conterrà le variabili di tipo word che rappresentano i vari registri del processore; si supponga che i codici operativi delle istruzioni prese come esempio siano:

MOV R0, sorgente ; codice operativo 0000000000000001
 MOV destinatario, R0 ; codice operativo 0000000000000011
 POP ; codice operativo 0000000000000000
 ADD ; codice operativo 0000000000000100

Descrivere e sintetizzare il Circuito con Parte Operativa e Parte Controllo in modo che prelevi 128 campioni della tensione v e ne presenti il valor medio sui pin $z7, z6, \dots, z0$. Si supponga che il convertitore A/D fornisca i campioni di v rappresentati in complemento a due



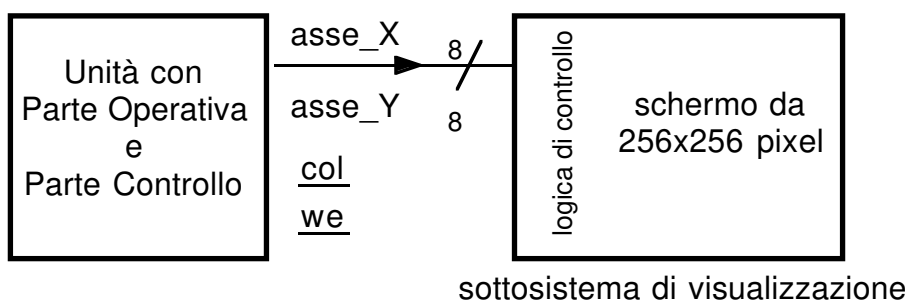


Il sottosistema di presentazione opera sul pixel di coordinate *asse_X*, *asse_Y* quando riceve un impulso tramite l'ingresso *we*; l'operazione consiste nel visualizzare il pixel se *col* vale 1, nello spengerlo se *col* vale 0.

Descrivere una unità con parte operativa e parte controllo che ripeta ciclicamente le seguenti operazioni

-) pulisce lo schermo; <-----
-) aspetta circa 1000 cicli di clock;
-) disegna un quadrato pieno di 10x10 pixel;
-) aspetta circa 1000 cicli di clock;
-) pulisce lo schermo;
-) aspetta circa 1000 cicli di clock;
-) disegna un quadrato pieno di 40x40 pixel;
-) aspetta circa 1000 cicli di clock; -----

p

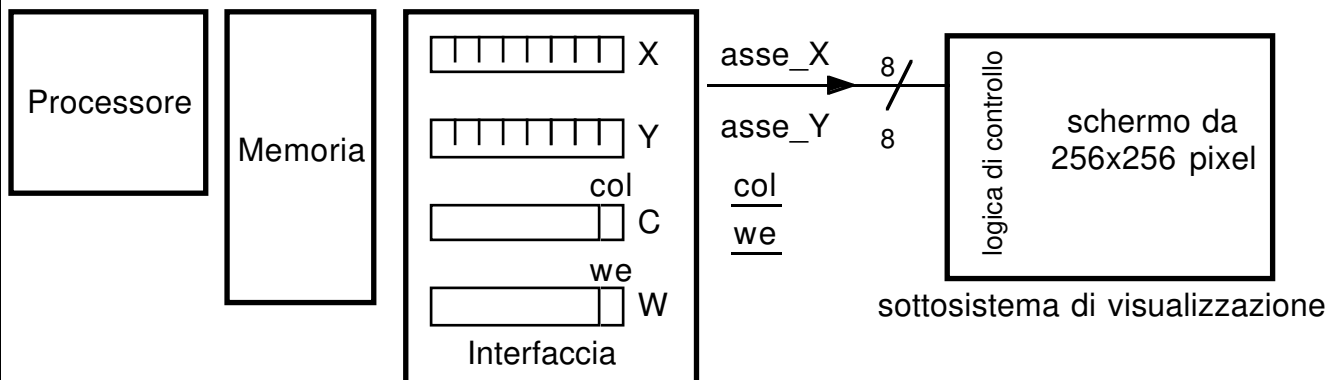


Scrivere un programma che, utilizzando l'interfaccia, compia ciclicamente le seguenti operazioni

-) pulisce lo schermo; <-----
-) aspetta un tempo pari a circa 1000 istruzioni;
-) disegna un quadrato pieno di 10x10 pixel;
-) aspetta un tempo pari a circa 1000 istruzioni;
-) pulisce lo schermo;
-) aspetta un tempo pari a circa 1000 istruzioni;
-) disegna un quadrato pieno di 40x40 pixel;
-) aspetta un tempo pari a circa 1000 istruzioni; -----

Si supponga che i quattro registri dell'interfaccia siano montati nello spazio di I/O agli indirizzi 0008H, 000AH, 000CH e 000EH.

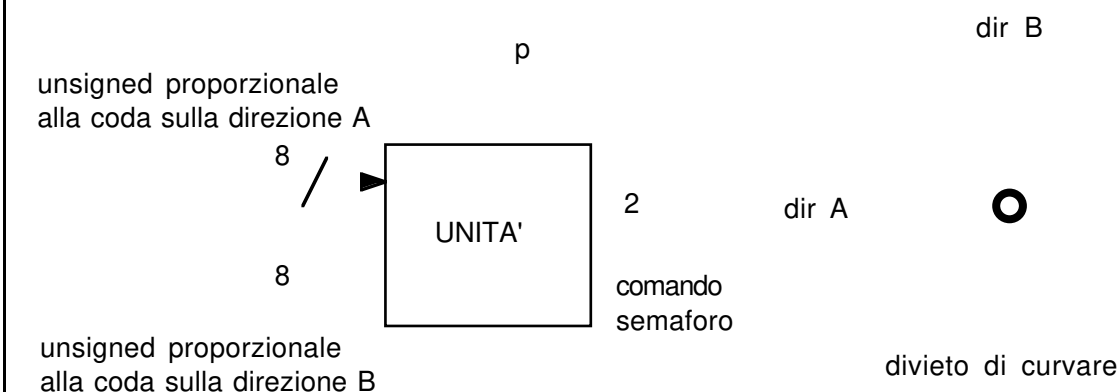
BUS



Disegnare l'interfaccia precedente e montarla nello spazio di I/O

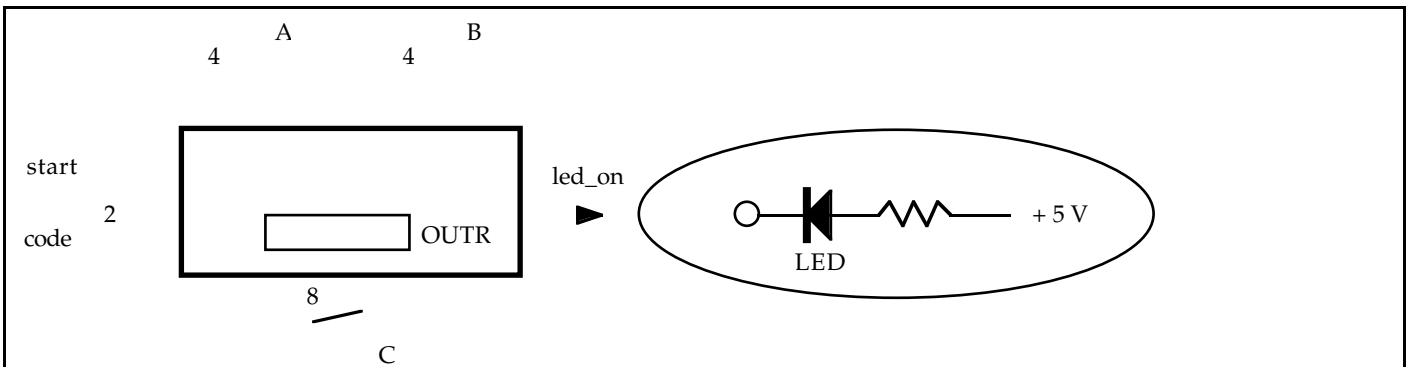
Descrivere e realizzare l'unità con parte operativa e parte controllo che partendo da una condizione iniziale in cui il semaforo è nello stato S0:

- 1) Tiene il semaforo nello stato S0 per 10.000 cicli di clock e poi lo mette nello stato S1;
- 2) Tiene il semaforo nello stato S1 per un "certo tempo" e poi lo mette nello stato S2. Il "certo tempo" è: almeno 20.000 cicli di clock, al più 60.000 cicli di clock e comunque (soddisfatti questi vincoli) il tempo in cui la coda sulla direzione A rimane maggiore di quella sulla direzione B;
- 3) Tiene il semaforo nello stato S2 per 10.000 cicli di clock e poi lo mette nello stato S3;
- 4) Tiene il semaforo nello stato S3 per un "certo tempo" e poi lo mette nello stato S0. Il "certo tempo" è: almeno 20.000 cicli di clock, al più 60.000 cicli di clock e comunque (soddisfatti questi vincoli) il tempo in cui la coda sulla direzione B rimane maggiore di quella sulla direzione A.



comando semaforo	00	implica	semaforo in stato S0 (dir A: rosso, dir B: giallo)
comando semaforo	01	implica	semaforo in stato S1 (dir A: verde, dir B: rosso)
comando semaforo	10	implica	semaforo in stato S2 (dir A: giallo, dir B: rosso)
comando semaforo	11	implica	semaforo in stato S3 (dir A: rosso, dir B: verde)

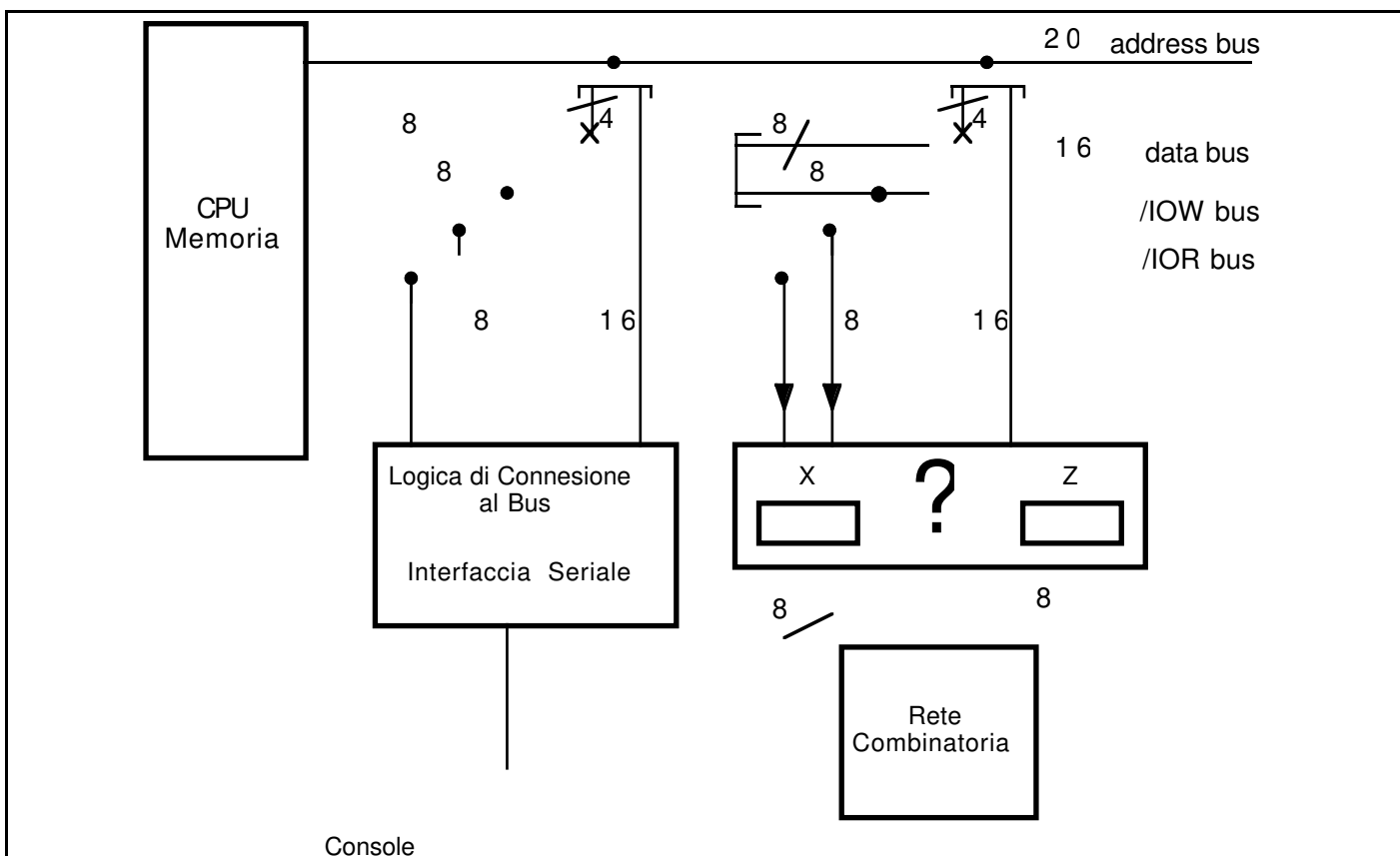
Realizzare una interfaccia per un microprocessore e montarla nello spazio di IO (ad indirizzi a vostra scelta), in modo da poter gestire il semaforo tramite un programma; scrivere un tale programma, prendendo come unità di tempo per individuare i ritardi, il tempo di esecuzione di una istruzione.



Descrivere il circuito (con parte operativa e parte controllo) in modo che, partendo da una condizione iniziale con $start=0$ e $led_on=1$ (LED acceso), si evolva all'infinito come segue. Quando esso, avendo il LED acceso, riceve un impulso sul pin $start$, spegne il LED e compie la seguente operazione (A e B sono unsigned a 4 bit):

se $code = 00$ mette in OUTR l'unsigned $A + B$ e poi accende il LED
 se $code = 01$ mette in OUTR l'unsigned $A - B$ e poi accende il LED
 se $code = 10$ mette in OUTR l'unsigned $A \times B$ e poi accende il LED
 se $code = 11$ mette in OUTR "l'unsigned" $A \text{ and } B$ e poi accende il LED

Per calcolare $A \times B$ non si faccia uso di un moltiplicatore combinatorio, ma si microprogrammi l'algoritmo di somma e shift.



- 1) Disegnare la scatola ? (cioè l'interfaccia e la logica di connessione al bus) in modo che il registro X sia accessibile con l'istruzione:

```
OUT IO:0020H, AL
```

ed il registro Z sia accessibile con l'istruzione:

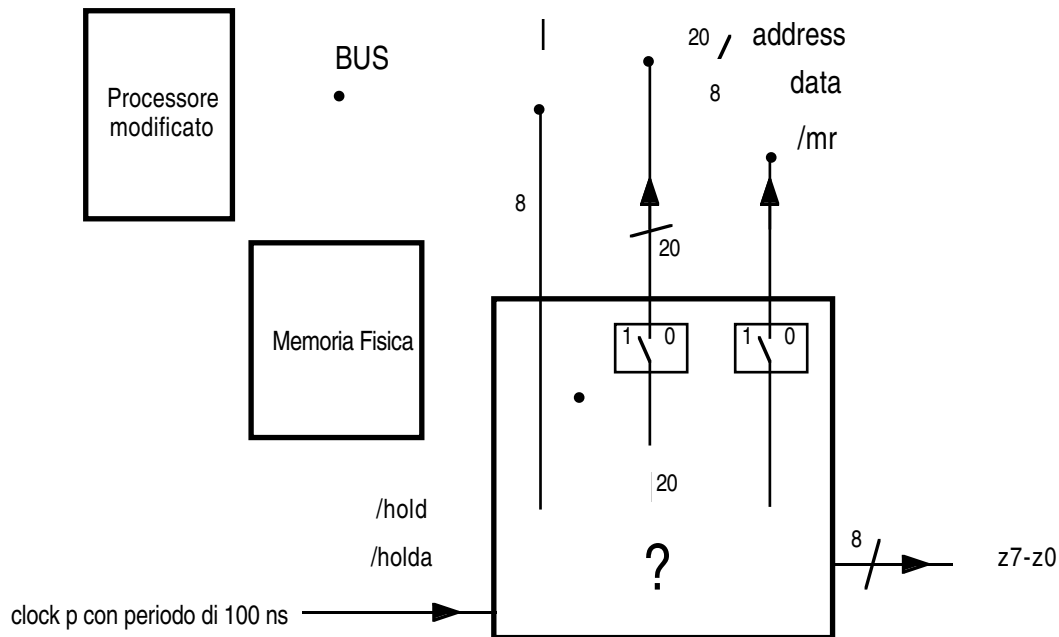
```
IN AL, IO:0022H
```

- 2) Scrivere un programma che faccia comparire sul video della console la tabella di verità della rete combinatoria nella seguente forma:

```
00:stato di uscita della rete su due cifre esadecimali
01:stato di uscita della rete su due cifre esadecimali
....
....
FF:stato di uscita della rete su due cifre esadecimali
```

L'interfaccia seriale è quella fatta a lezione ed illustrata nel testo: per gli indirizzi dei registri, usare quelli riportati nel testo.

- 3) Sostituire la CPU, la Memoria ed il Programma con un circuito sequenziale che faccia esattamente le stesse cose.

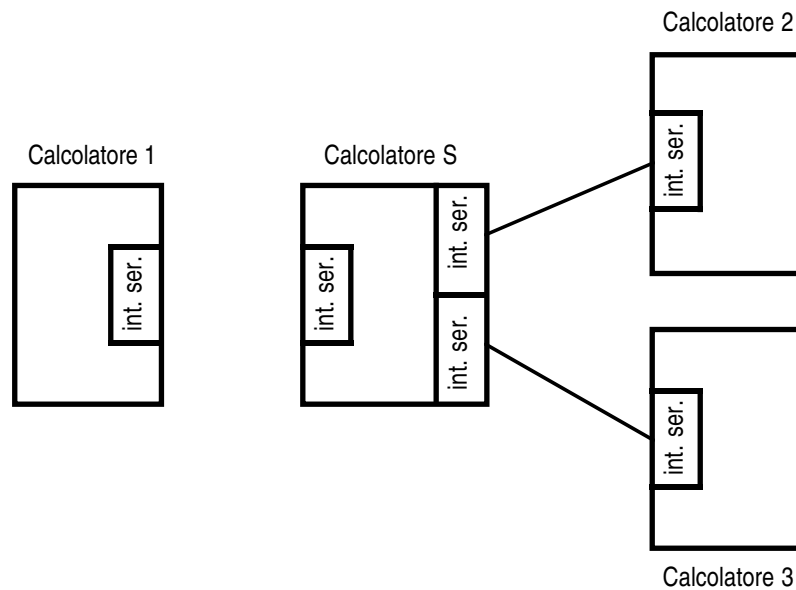


Il processore visto a lezione è stato modificato nel seguente modo:

- 1) tutti i suoi piedini (esclusi i piedini per il clock, il reset e la massa) sono supportati da porte a tre-stati, cosicché il processore può isolarsi completamente dalla memoria e dallo spazio di I/O ponendo tali porte in alta impedenza;
- 2) è stata aggiunta la variabile di ingresso */hold* e la variabile di uscita */holda* (non supportata da una porta a tre stati);
- 3) quando */hold* viene messa a 0 dalle circuiterie esterne, il processore, entro un tempo massimo di 1 μ s, si *blocca* cioè si isola dalla memoria e dallo spazio di I/O, non compie alcuna evoluzione e pone */holda* a 0; quando */hold* viene riportato ad 1 il processore, entro un tempo massimo di 0.2 μ s, si *sblocca* cioè rimuove il suo isolamento, pone */holda* ad 1 e riprende la sua normale evoluzione.

Descrivere il circuito sincronizzato di figura in modo che esso emetta sui pin *z7-z0* un nuovo byte ogni 3 μ s. I byte emessi nel tempo sono 64K e sono prelevati, uno dopo l'altro dal segmento di memoria di selettore A000H. Quando il circuito intende accedere alla memoria si preoccupa di bloccare il processore; quando invece non sta accedendo alla memoria si isola dalla memoria stessa mettendo le sue porte a tre stati in alta impedenza e sbloccando il processore. Si descriva il circuito in modo da dare il minor disturbo possibile al processore. Si supponga la memoria così veloce, da non dover inserire stati di attesa nei cicli di lettura.

Es. 1



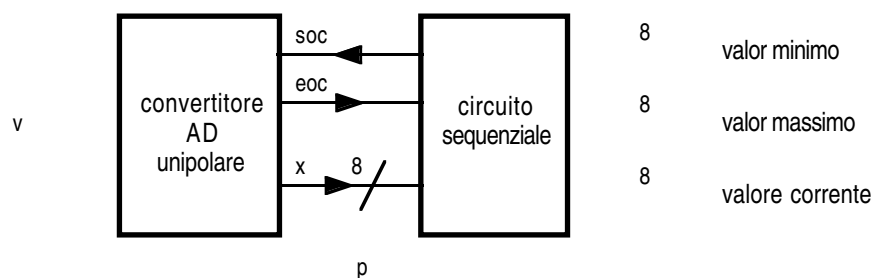
Scrivere un **programma** per il calcolatore S, un **sottoprogramma** per il calcolatore 1, un **sottoprogramma** per il calcolatore 2 ed un **sottoprogramma** per il calcolatore 3 in modo che sia espletabile il seguente servizio (una volta che sia stato preventivamente lanciato il programma sul calcolatore S):

- 1) Un programma principale sul calcolatore 1 può utilizzare il sottoprogramma per inviare il corpo di un segmento o al calcolatore 2 o al calcolatore 3;
- 2) Un programma principale sul calcolatore 2 può utilizzare il sottoprogramma per prelevare i byte che arrivano dall'interfaccia seriale e immetterli come corpo in un segmento;
- 3) Un programma principale sul calcolatore 3 può utilizzare il sottoprogramma per prelevare i byte che arrivano dall'interfaccia seriale e immetterli come corpo in un segmento.

Si precisino, nel modo che si ritiene il più funzionale possibile, le modalità di passaggio dei parametri tra programmi principali e sottoprogrammi e gli offset nello spazio di I/O dei registri delle interfacce seriali. Ci si ricordi anche dei problemi inerenti al controllo del flusso dei dati che viaggiano da un calcolatore ad un altro.

Es. 2

Descrivere e realizzare in dettaglio il circuito sequenziale di figura

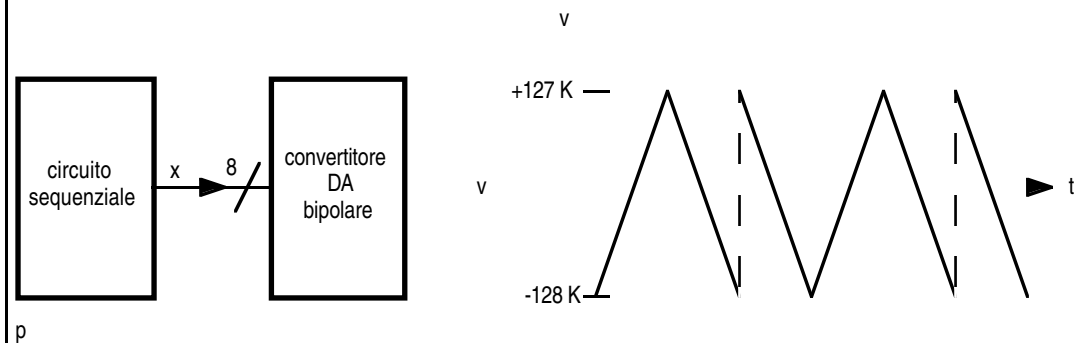


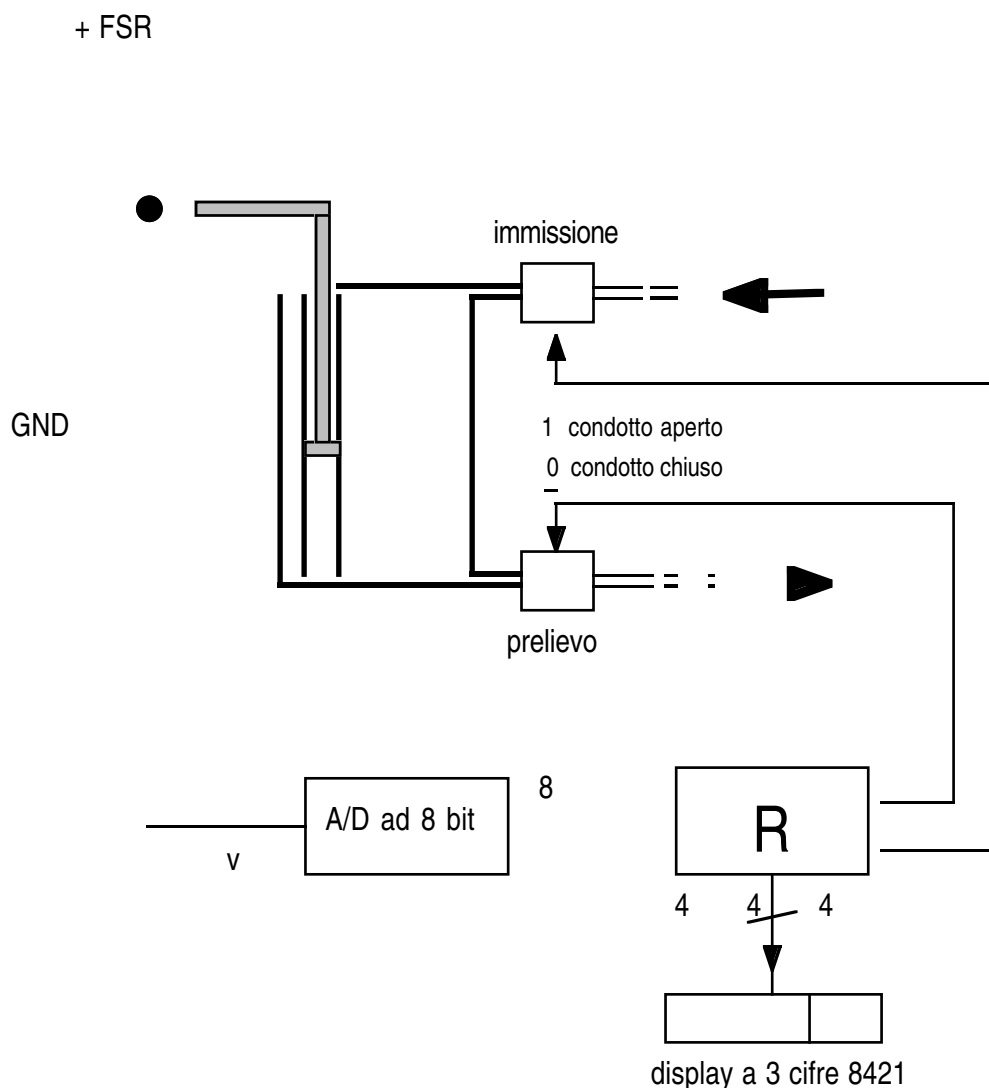
Es. 1

Scrivere un sottoprogramma che dato un numero integer rappresentato in complemento alla radice, in base 2 e memorizzato nel registro AX visualizzi sul video della console il numero in base 10.

Es. 2

Descrivere e realizzare il circuito sequenziale di figura in modo che la tensione v abbia la forma sottoriportata.





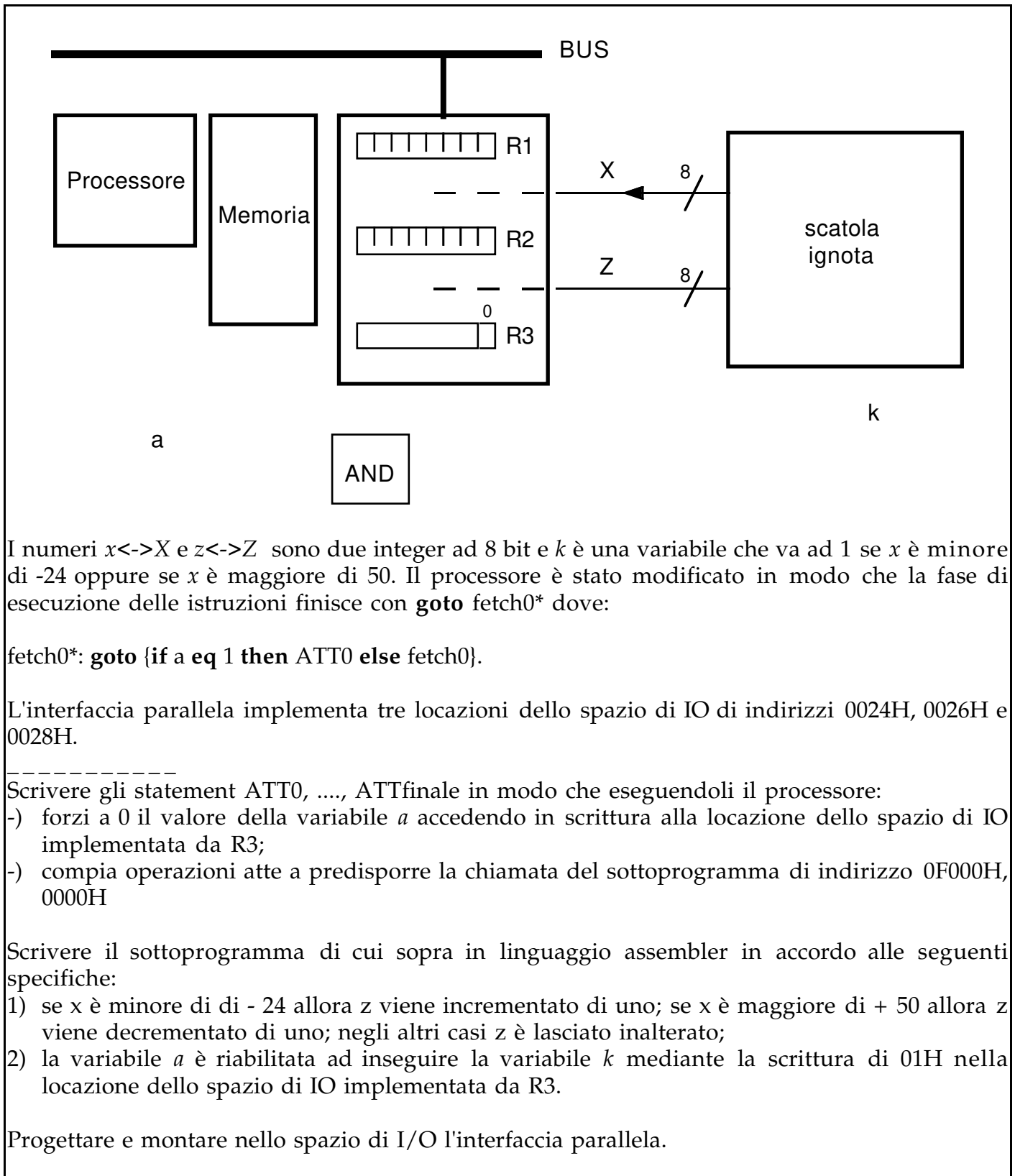
Progettare (in modo euristico) la rete R in modo che:

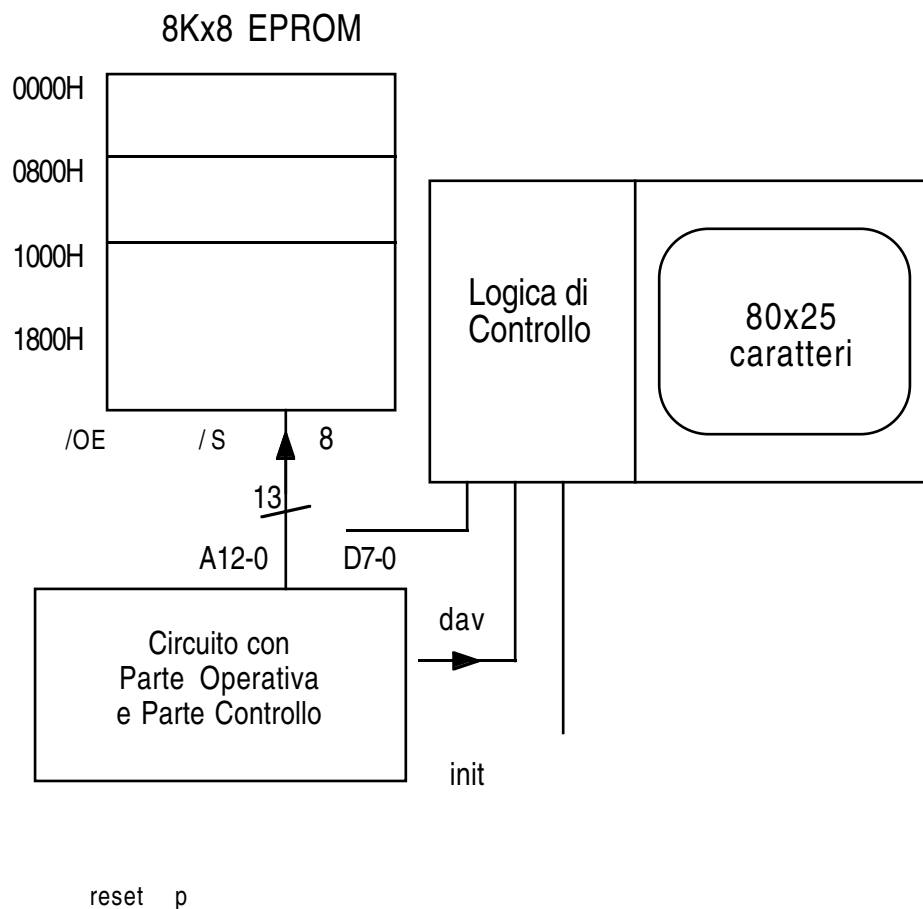
- 1) il serbatoio tenda ad essere pieno senza traboccare;
- 2) quando il livello scende sotto il dieci per cento (circa) venga comunque inibito il prelievo
- 3) sui display venga visualizzato in decimale il livello normalizzato a 100 (circa).

Scrivere un sottoprogramma che:

- 1) riceve in DX, BX l'indirizzo logico della prima locazione di un buffer di memoria, nel quale sono contenute le cifre decimali (codificate ASCII) di un numero unsigned;
- 2) emette sulla console la codifica ASCII delle cifre esadecimali dello stesso numero

ATTENZIONE: si supponga che il numero abbia 6 cifre decimali



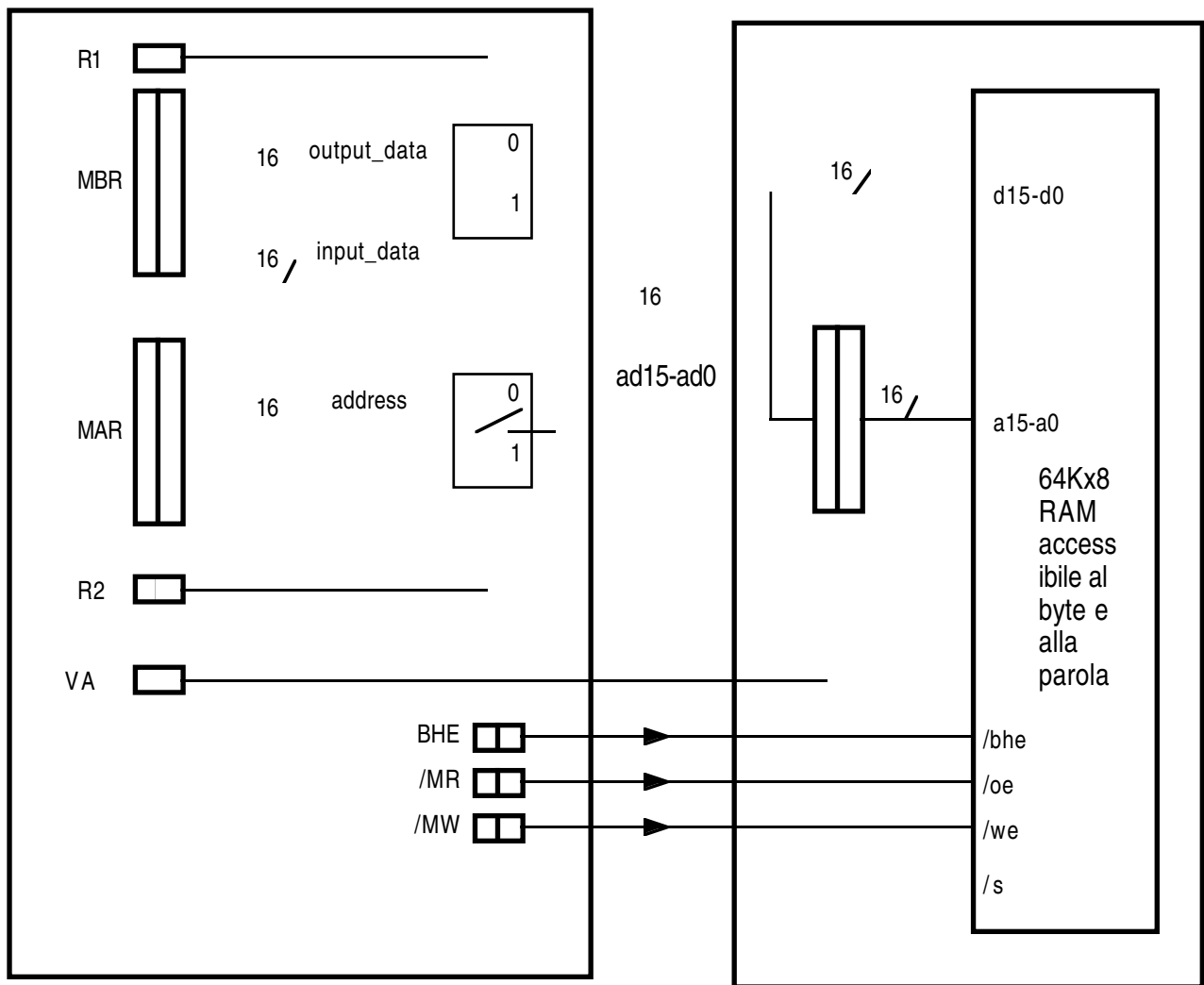


Descrivere il Circuito Sequenziale con Parte Operativa e Parte Controllo in modo che al reset iniziale emetta un impulso 0-1-0 tramite *init*. e poi esegua all'infinito i seguenti passi:

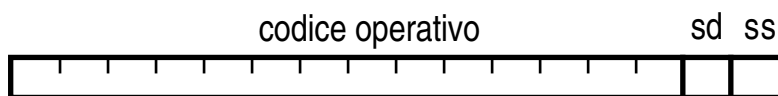
- 1) legga per 1000 volte i primi 2000 byte a partire dall'indirizzo 0000H;
- 2) legga per 1000 volte i primi 2000 byte a partire dall'indirizzo 0800H;
- 3) legga per 1000 volte i primi 2000 byte a partire dall'indirizzo 1000H;
- 4) legga per 1000 volte i primi 2000 byte a partire dall'indirizzo 1800H;

Attenzione: Ogni volta un nuovo byte è pronto per la Logica di Controllo, il circuito emette un impulso 0-1-0 di notifica tramite *dav*.

Si supponga che il tempo di accesso della EPROM stia fra 1 e 2 periodi del clock *p*..



Il processore di figura ha uno spazio di memoria lineare da 64Kbyte e non ha spazio di I/O. Per accedere alla memoria usa un unico bus per gli indirizzi e per i dati. Durante un ciclo di accesso alla memoria, il processore prima immette sul bus gli indirizzi e poi utilizza il bus per i dati: gli indirizzi rimangono per circa due periodi del clock e alla fine del primo periodo essi sono notificati dalla variabile di uscita del registro VA che viene messa ad 1 (e poi riportata a 0). Il formato delle istruzioni è il seguente:



codice operativo

- 00---- Istruzioni senza operandi
- 01---- Istruzioni con un operando in memoria
- 1----- Istruzioni con due operandi in memoria

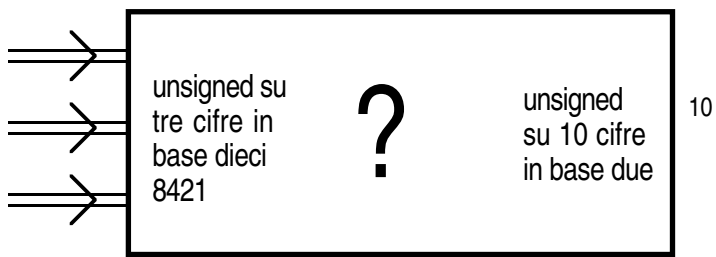
I bit *sd* ed *ss*, quando significativi, indicano:

sd

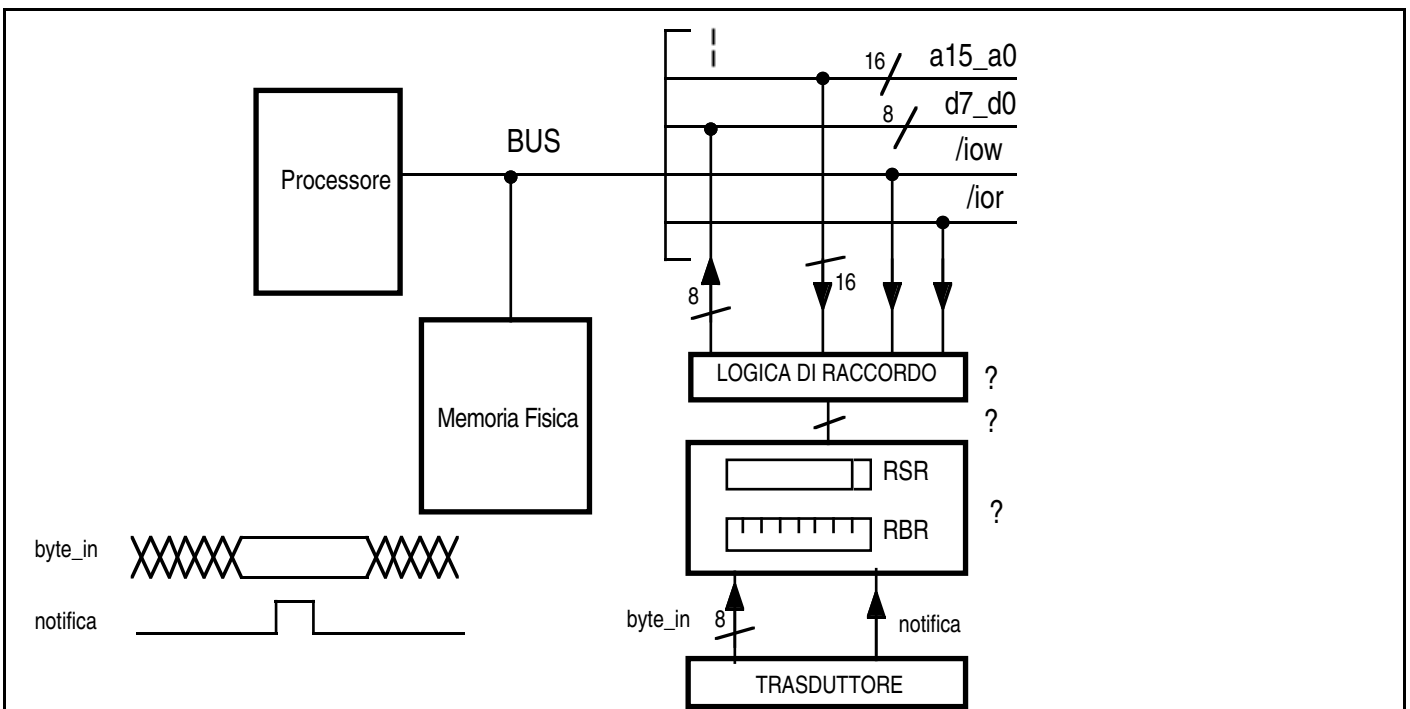
- 0 l'indirizzo dell'operando destinatario sta in una ulteriore parola dell'istruzione.
- 1 l'indirizzo dell'indirizzo dell'operando destinatario sta in una ulteriore parola dell'istruzione (indirizzamento indiretto).

ss

- 0 l'indirizzo dell'operando sorgente sta in una ulteriore parola dell'istruzione.
- 1 l'indirizzo dell'indirizzo dell'operando sorgente sta in una ulteriore parola dell'istruzione (indirizzamento indiretto).



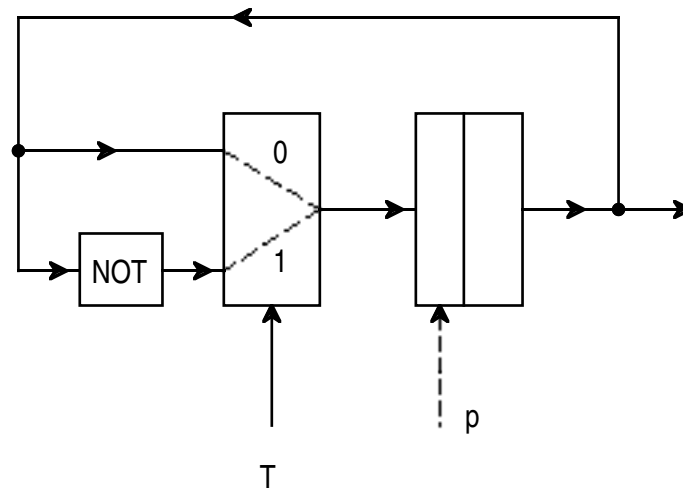
Disegnare uno schema a blocchi significativo per la rete combinatoria ?.



Definire i fili di collegamento ?, la logica di raccordo ? e l'interfaccia ? in modo che il processore possa leggere il registro di Stato RSR con l'istruzione `IN AL, 0024H` ed il registro dati RBR con l'istruzione `IN AL, 0025H`: si predispongano le cose in modo che l'essere il bit n. 0 di RSR ad 1 significhi che RBR è pieno. Il colloquio tra il trasduttore e l'interfaccia è riportato in figura: un byte utile è notificato dal trasduttore mediante un breve impulso sul filo **notifica**. Non è previsto un filo di risposta al trasduttore in quanto si suppone che esso invii byte utili molto raramente, per cui: 1) il processore può comodamente prelevarli (purché il bit n.0 del registro di stato nell'interfaccia indichi che RBR è pieno) e 2) l'interfaccia è per il trasduttore sempre pronta. "Breve Impulso" significa che esso dura un tempo minore rispetto al tempo di esecuzione di una istruzione.

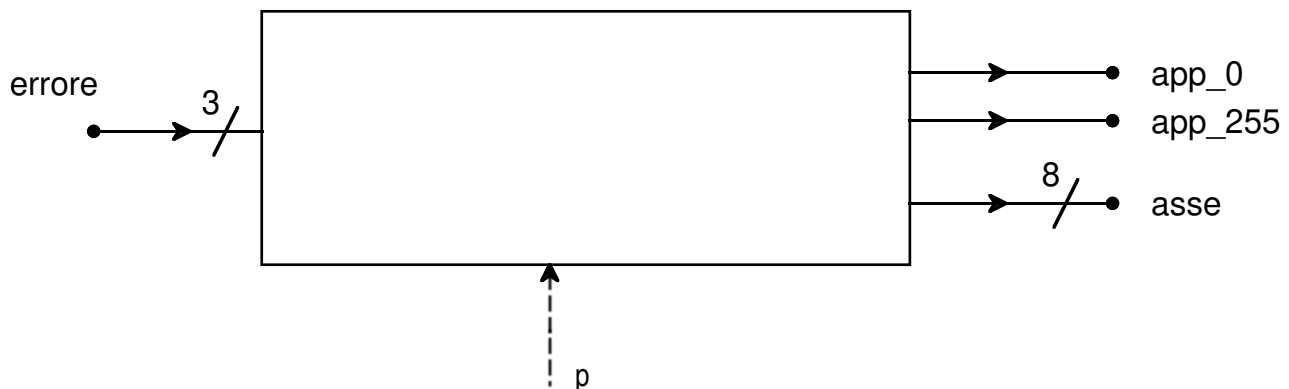
Scrivere un sottoprogramma che moltiplichi un unsigned a 32 bit (parte più significativa in CX, parte meno significativa in BX) per un integer memorizzato nei 12 bit meno significativi di AX e lasci il risultato (un integer) in memoria a partire dall'indirizzo logico (DS, DI). Si verifichi se il risultato è rappresentabile su 32 bit ed in tal caso si metta a 0 il registro AL.

Il Flip-Flop T-positive edge triggered può essere realizzato come in figura.



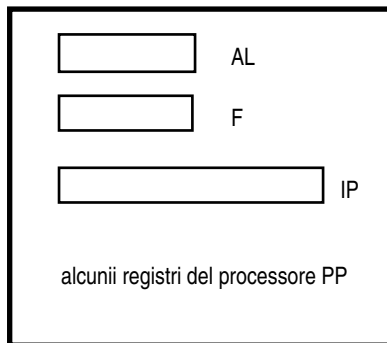
Descriverlo in accordo alla teoria dei circuiti sequenziali asincroni vedendolo come un circuito unico (vecchi programmi) ovvero come una coppia master-slave (nuovi programmi).

Realizzare nel modo che si ritiene più opportuno il circuito di figura, purché esso si comporti come segue:



All'arrivo del fronte in salita del clock il circuito aggiorna l'uscita *asse* in base alle indicazioni dell'ingresso *errore*. L'uscita *asse* va pensata come un unsigned a 8 bit e l'ingresso *errore* come un integer in modulo e segno: il segno di *errore* indica se *asse* va diminuito od aumentato; il modulo di errore dà l'entità della variazione da apportare ad *asse*. Qualora l'aggiustamento di *asse* tenda a produrre un numero negativo o superiore a 255 (ricordare che *asse* è un unsigned) *asse* viene approssimato nel modo migliore (a 0 o a 255) e quindi viene segnalato l'inconveniente portando ad 1 *app_0* o *app_255* (normalmente tali variabili sono a 0).

Un processore PP, capace di lavorare solo su byte e di indirizzare una memoria lineare non segmentata da 64 Kbyte, ha la seguente struttura:



MOV AL,operand	CMP AL,operand
MOV AL,MEM:offset	JZ MEM:offset
MOV MEM:offset,AL	JNZ MEM:offset
ADD AL,MEM:offset	JMP MEM:offset
SUB AL,MEM:offset	

Definirsi un linguaggio macchina a piacere e completare il seguente programma in modo che "simuli" un calcolatore basato sul processore di cui sopra:

```

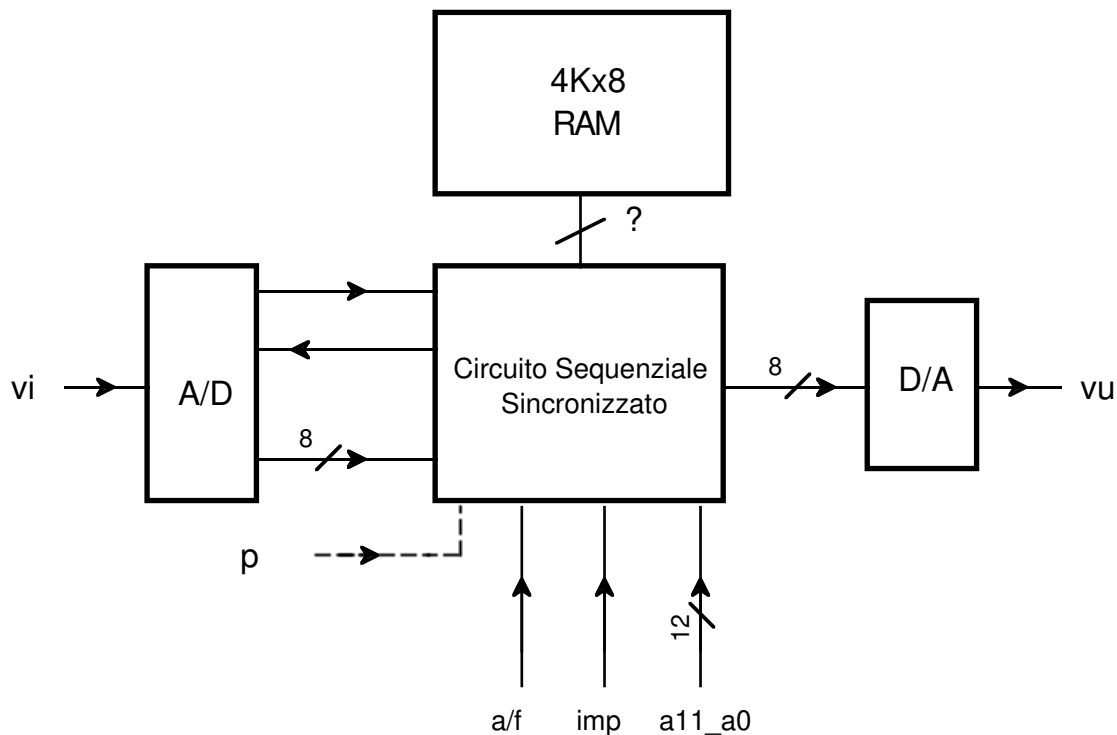
MEM_SIM SEGMENT
MEM      DB 65536 DUP (?)
;se ci mettessi le istruzioni di PP in
;linguaggio macchina, la loro esecuzione
;sarebbe simulata
MEM_SIM ENDS

CPU_SIM SEGMENT
AL_SIM   DB (?)
F_SIM    DB (?)
IP_SIM   DW (?)
OPCODE_SIM DB (?)
.....
.....
CPU_SIM ENDS

SIMULA SEGMENT
ASSUME CS:SIMULA, DS:MEM_SIM, ES:CPU_SIM
START:   MOV AX, SELECTOR MEM_SIM
         MOV DS, AX
         MOV AX, SELECTOR CPU_SIM
         MOV ES, AX

RESET:   MOV IP_SIM, 0000H

FETCH0:  MOV DI, IP_SIM
         MOV AH, MEM[DI]
         MOV OPCODE, AH
         INC DI
         .....
         .....
SIMULA ENDS
END START
    
```



Descrivere il Circuito Sequenziale che, partendo da una condizione iniziale di reset, esamina "periodicamente" la variabile a/f . Se scopre che a/f vale 1 si comporta come illustrato nel punto a1, mentre se scopre che a/f vale 0 si comporta come illustrato nel punto f1.

- a1) quando tramite la variabile imp arriva un segnale di notifica del tipo 0->1->0, il circuito preleva lo stato di $a11-a0$ e lo considera come un indirizzo per la RAM; poi il circuito preleva un campione della tensione vi colloquiando con il convertitore A/D e memorizza nella RAM questo campione (si supponga che i segnali che arrivano tramite imp siano così distanziali nel tempo da non creare alcun problema al circuito)
- f1) il circuito legge la RAM byte dopo byte partendo dall'indirizzo 000H ed invia ogni byte al convertitore D/A mantenendo un ritmo pari a 4 periodi di clock.

Scrivere un programma che 1) preleva dalla tastiera un primo numero X a 4 cifre decimali, 2) preleva un secondo numero Y a 4 cifre decimali e 3) emette un risultato secondo le specifiche che seguono (il programma fa anche i soliti controlli, scarta eventuali caratteri in ingresso che non siano cifre decimali, fa l'eco sul video di ogni cifra utile e va a capo quando necessario).

Specifiche per calcolare il risultato:

Il programma interpreta X ed Y come le rappresentazioni in base β =dieci di due integer x ed y , in complemento alla radice. Emette la rappresentazione in modulo e segno e su 5 cifre decimali di $x+y$.

1° Esercizio

Con riferimento al sistema in figura:

a) specificare il fascio dei fili ? e progettare le scatole ? in modo che le istruzioni:

IN AL, IO:0051H

IN AL, IO:0053H

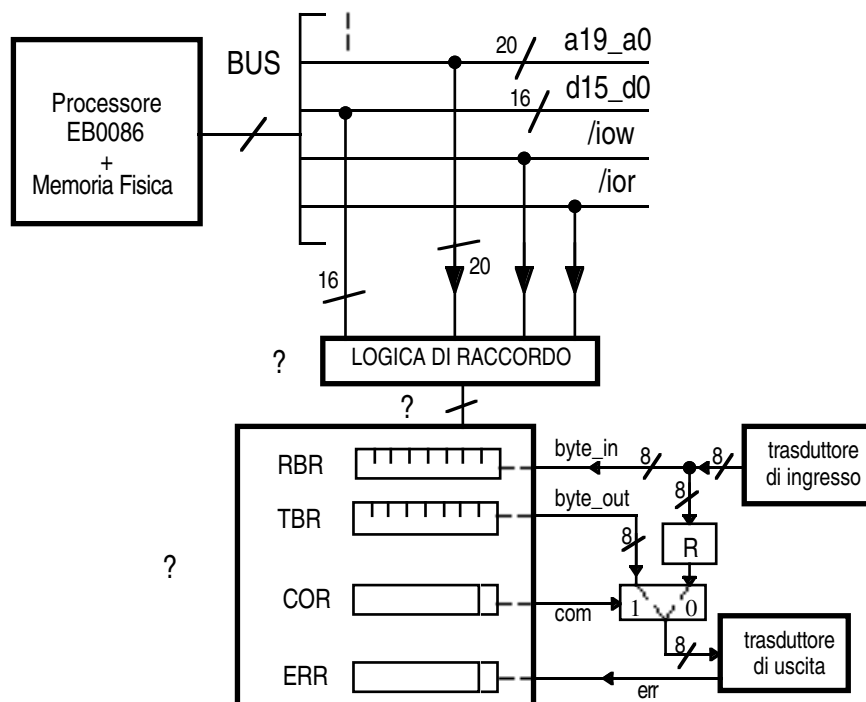
permettano di accedere in lettura ai registri RBR, ERR, mentre le istruzioni:

OUT IO:0055H, AL

OUT IO:0057H, AL

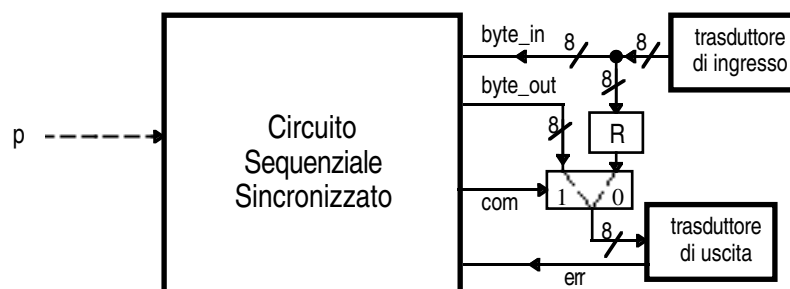
permettano di accedere in scrittura ai registri TBR, COR

b) scrivere un programma ciclico che preleva il valore della variabile *err* e se *err* = 0 tiene *com* a 0 mentre se *err* = 1 compie le seguenti elaborazione: preleva *byte_in* e lo interpreta come un integer in complemento alla radice (base dieci, due cifre codificate 8421) ; se *byte_in* è negativo invia al trasduttore di uscita un integer in complemento alla radice (base dieci, due cifre codificate 8421) con valore pari all'estremo inferiore negativo; se *byte_in* è positivo invia al trasduttore di uscita un integer in complemento alla radice (base dieci, due cifre codificate 8421) con valore pari a +01.



2° Esercizio

Descrivere il Circuito Sequenziale Sincronizzato che, ai fini dei trasduttori, svolga le stesse funzioni descritte nel precedente punto b).



3° Esercizio

Data la mappa in figura:

- indicare e classificare tutti gli implicant principali;
- trovare la lista (o le liste) di copertura per la forma minima di tipo SP specificando dettagliatamente tutte le fasi della sintesi;
- individuare e classificare le eventuali alee del primo ordine presenti in una delle forme minime trovate per il punto b), modificare la corrispondente forma in modo da eliminare le alee e disegnare la rete risultante;
- effettuare una nuova sintesi della mappa usando esclusivamente le porte NOR.

Attenzione: specificare gli implicant e le espressioni utilizzando esclusivamente le variabili e l'ordinamento usati in figura!!!!

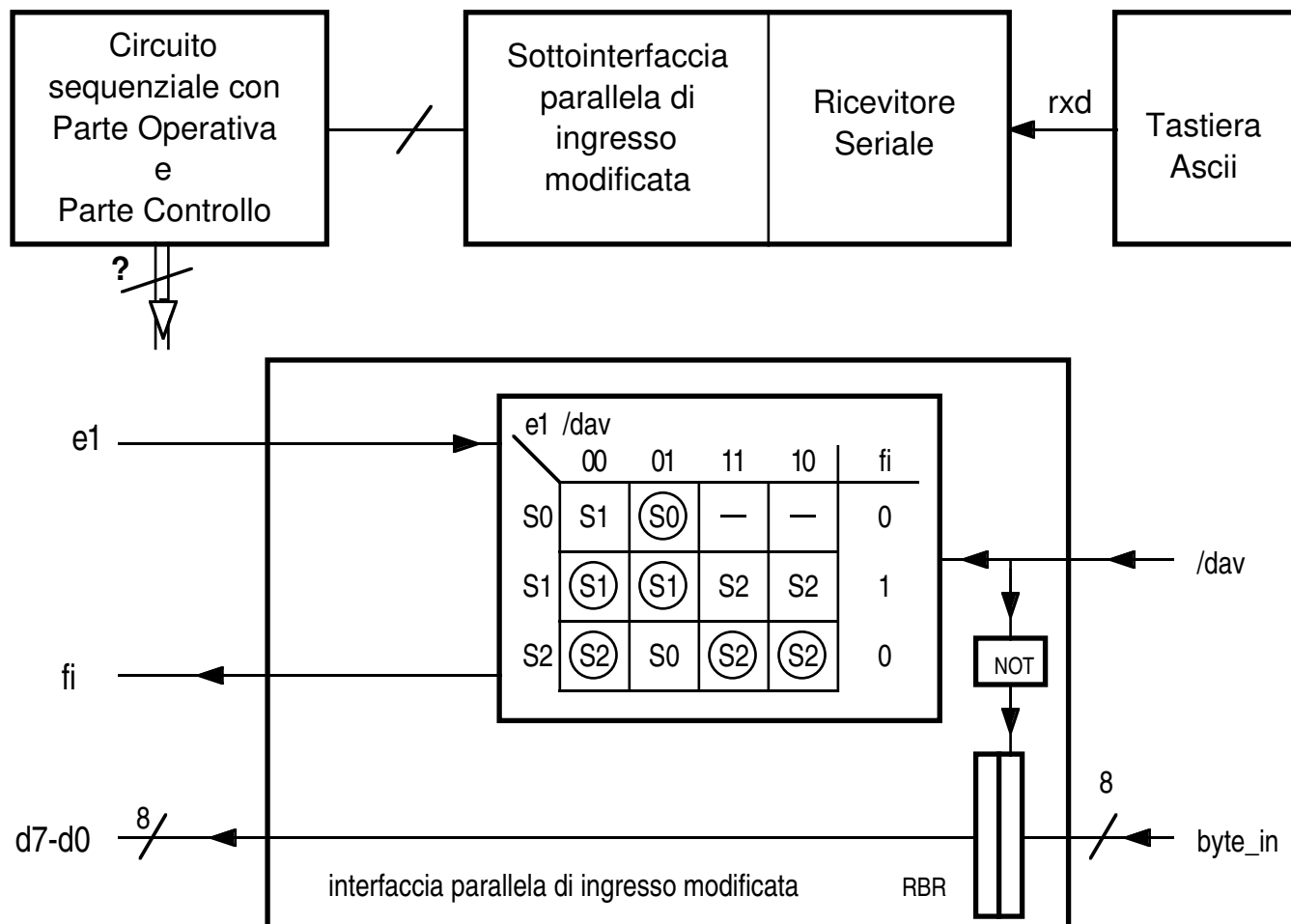
		x3 x2			
		00	01	11	10
x1x0	00	1	0	1	1
	01	1	-	1	0
	11	-	1	1	0
	10	1	0	0	-

1° Esercizio.

Descrivere il Circuito Sequenziale Sincronizzato (e darne la sintesi della parte operativa) in modo che esso ripeta all'infinito le seguenti operazioni:

- preleva (correttamente) un unsigned in base dieci a quattro cifre decimali codificate ASCII
- emette tramite le uscite ? lo stesso unsigned in base dodici.

La sottointerfaccia parallela è stata semplificata come rappresentato in figura.

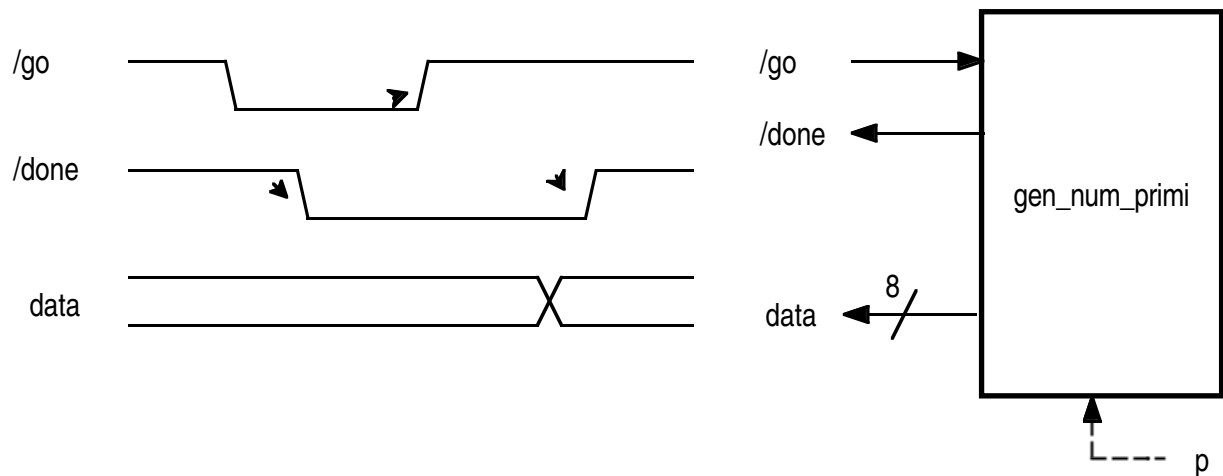


2° Esercizio.

Descrivere come rete sequenziale di **Mealy** e sintetizzare un contatore **up/down espandibile** in base 3 ad 1 cifra. Per la descrizione usare una lista di statement, una tabella di flusso o un grafo di flusso.

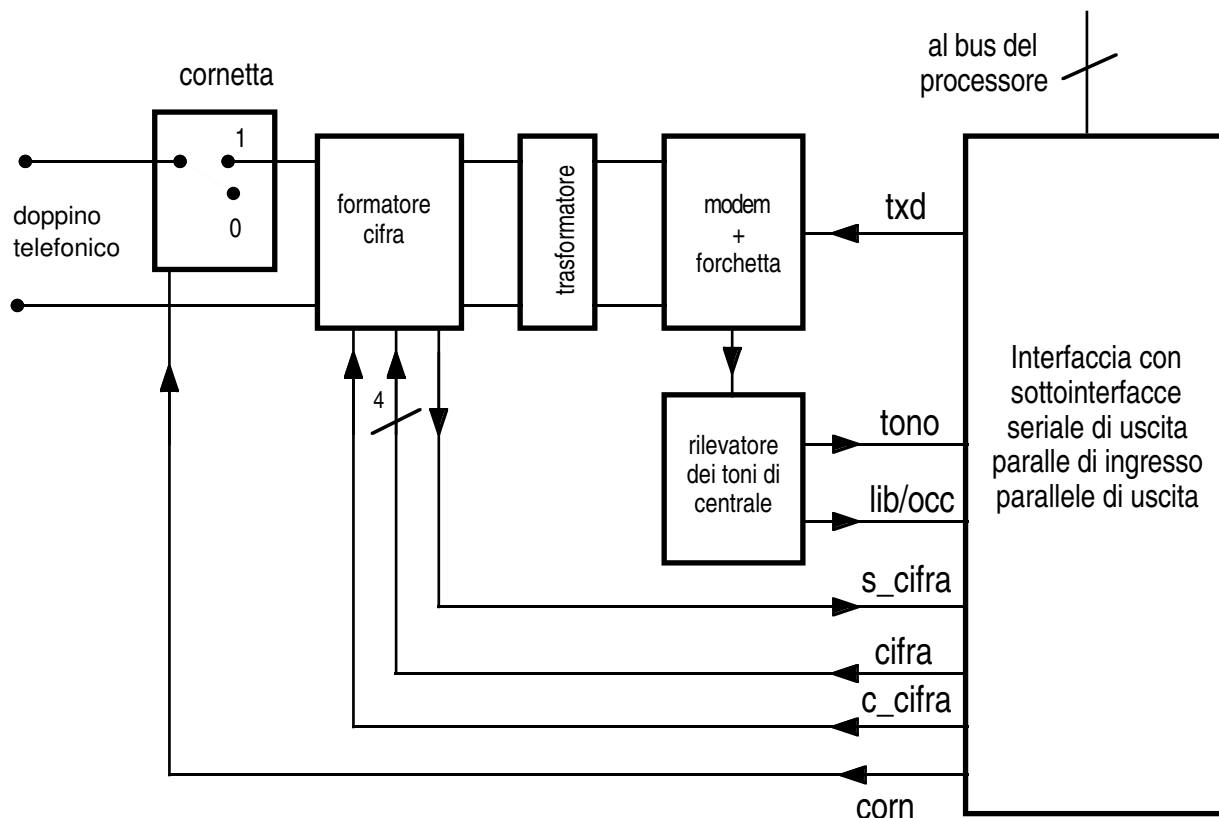
3° Esercizio (difficoltà nulla).

Scrivere un programma che ciclicamente preleva dalla tastiera due cifre binarie codificate ASCII e le interpreta come lo stato di ingresso di un flip-flop J-K. Il programma attende l'arrivo di un ritorno carrello e lo interpreta come il fronte in salita del clock del flip-flop. Il programma emette quindi il nuovo stato di uscita del flip-flop come cifra binaria codificata ASCII.



Partendo da una situazione iniziale in cui */go* = 1 e */done* = 1, il circuito *gen_num_primi* opera come segue: 1) riceve l'ordine di lavorare; 2) risponde che sta lavorando; 3) è ringraziato perché sta lavorando; 4) comunica di aver finito il lavoro. Il lavoro consiste nel calcolare un nuovo numero primo e presentarlo tramite gli 8 bit *data*., e così via all'infinito (ciclicamente).

Descrivere il circuito con un linguaggio di trasferimento tra registri e sintetizzarlo fino a definire la Parte Operativa ed il microprogramma che caratterizza la ROM della Parte Controllo. Si supponga che il generatore funzioni in modo ciclico, partendo da una condizione di reset iniziale in cui emette 1.



In figura è riportato un possibile schema a blocchi di un trasmettitore telefonico. Per stabilire una connessione con un ricevitore telefonico, occorre:

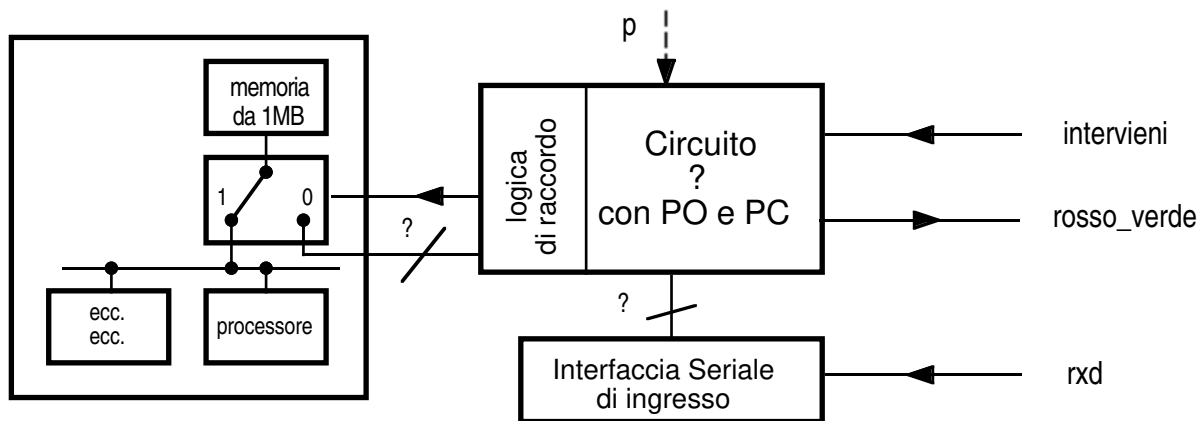
- 1) alzare la cornetta (portare *corn* ad 1)
- 2) attendere il tono di centrale (attendere che *tono* vada ad 1)
- 3) inviare, cifra dopo cifra, verso il formatore di cifre il numero telefonico del ricevitore: per ogni cifra verificare che *s_cifra* sia ad 1, presentare la cifra su 4 bit e mettere *c_cifra* prima ad 1 e poi a 0
- 4) attendere e testare il tono di libero/occupato (

<i>tono</i>	<i>lib/occ</i>	= 0	-	assenza di tono
<i>tono</i>	<i>lib/occ</i>	= 1 1		tono di libero
<i>tono</i>	<i>lib/occ</i>	= 1 0		tono di occupato
- 5) ripetere la procedura se il ricevitore chiamato è occupato (ricordarsi di riabbassare la cornetta) altrimenti attendere la scomparsa del tono di centrale e considerare la connessione stabilita.

Disegnare l'interfaccia in termini di una sottointerfaccia seriale di uscita, di una sottointerfaccia parallela di ingresso e di una o più sottointerfacce parallele di uscita. e Connettere l'interfaccia al bus del computer visto a lezione in modo che i vari registri siano accessibili tramite le istruzioni IN ed OUT ad indirizzi da definire a piacere.

Scrivere un programma che invii al ricevitore con numero telefonico 265 il messaggio PROVA DI COLLEGAMENTO e poi chiudere la connessione abbassando la cornetta.

1° Esercizio.



Descrivere la logica di raccordo ed il Circuito con Parte operativa e parte controllo in accordo alle seguenti specifiche:

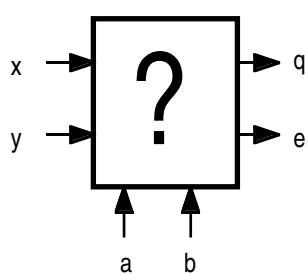
- Quando la variabile di ingresso *intervieni* vale 0 il circuito si trova in una condizione di riposo in cui non influenza la normale evoluzione del computer e tiene a 0 la variabile di uscita *rosso_verde*
- Quando la variabile di ingresso vale *intervieni* va ad 1, il circuito passa in una condizione di lavoro in cui mette ad 1 la variabile di uscita *rosso_verde*, disconnette la memoria dal processore ed entra in un ciclo in cui preleva 256 byte dall'interfaccia seriale e li scrive nella memoria a partire dall'indirizzo fisico FFF00H. Quando questa operazione è compiuta, il circuito mette a 0 la variabile di uscita *rosso_verde*, attende che la variabile di ingresso *intervieni* torni a 0, dopo di che si porta nella condizione di riposo e così via all'infinito.

2° Esercizio.

Scrivere un programma che preleva dalla tastiera un unsigned X a 4 cifre decimali (codificate ASCII) ed emette un unsigned Y a 4 cifre ottali (codificate ASCII) in accordo alle seguenti specifiche: X ed Y rappresentano lo stesso integer k in complemento alla radice (il primo in base dieci ed il secondo in base otto).

Il programma dovrà i) scartare eventuali caratteri in ingresso che non siano cifre decimali, ii) fare l'eco su video di ogni carattere corretto, iii) andare a capo quando ritenuto necessario e iv) emettere "EEEE" quando k non è rappresentabile in base otto su 4 cifre.

2° Esercizio.

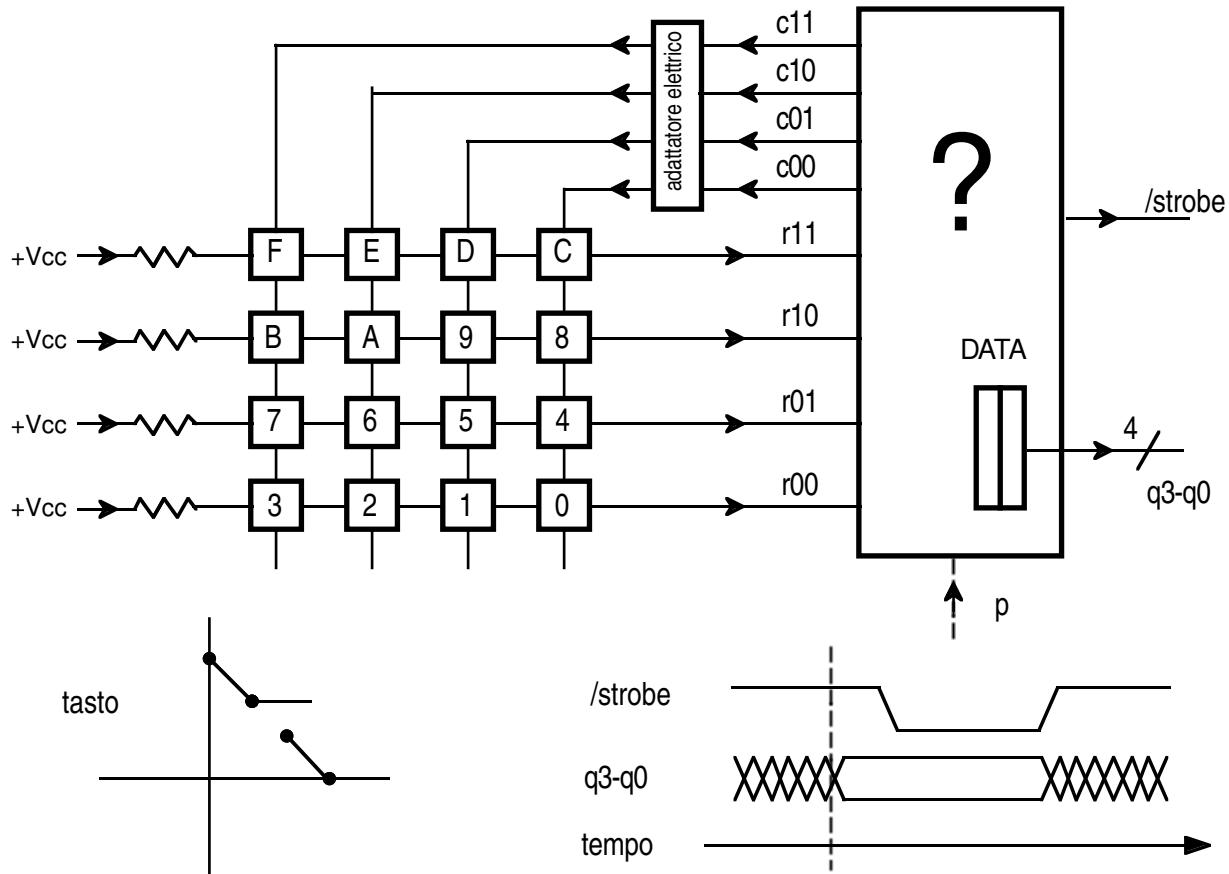


a	b	q
0	0	$ x + y _2$
0	1	$ x - y _2$
1	0	$ x \cdot y _2$
1	1	$\lfloor x/y \rfloor$

a	b	x	y	q	e
0	0	0	0		
1	1	1	1		

Disegnare la tabella di verità e quindi sintetizzare, mediante porte NOR, la rete combinatoria in figura in modo che l'uscita **q** presenti il valore indicato in tabella. L'uscita **e** vale 1 in caso di errori (overflow, operazione non eseguibile), vale 0 altrimenti.

1° Esercizio.



Descrivere il Circuito ? con il modello Parte Operativa e Parte Controllo in accordo alle seguenti specifiche:
 Il circuito, tenendo la variabile */strobe* ad 1, "scandisce" la matrice dei tasti fino a quando riscontra che un tasto è stato premuto; a questo punto il circuito emette, tramite le variabili *q3-q0*, la codifica binaria su 4 bit del carattere stampigliato sul tasto, mette a 0 la variabile */strobe* ed attende che il tasto venga rilasciato. Quando ciò accade, il circuito riporta ad 1 la variabile */strobe* e torna a scandire la matrice dei tasti e così via all'infinito.

Sintetizzare la Parte Operativa, utilizzando (fra altre) la rete combinatoria descritta di seguito per implementare la porzione che riguarda il registro DATA.

z1	z0 = case	x11	x10	x01	x00	of	{
		0	-	-	-	:	11,
		1	0	-	-	:	10,
		1	1	0	-	:	01,
		1	1	1	0	:	00,
		others				:	-- }.

NOTE:

Si supponga che i tasti vengano premuti uno alla volta e per un tempo sufficientemente lungo da non creare alcun problema di temporizzazione.

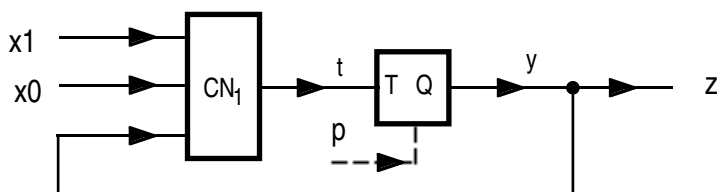
Scandire significa, fra l'altro, emettere tramite le variabili *c11*, *c10*, *c01*, *c00* sequenze di stati del tipo 0111, 1011, ecc.

2° Esercizio.

Sintetizzare il circuito sequenziale di Moore descritto sotto, utilizzando come elemento di marcatura degli stati interni il flip-flop T (all'arrivo del fronte in salita del clock, il flip-flop conserva se $T=0$, commuta se $T=1$).

x1 x0		00	01	11	10	z
S0	S0	S0	S1	S1	S0	0
	S1	S1	S0	S1	S0	1

tabella di flusso



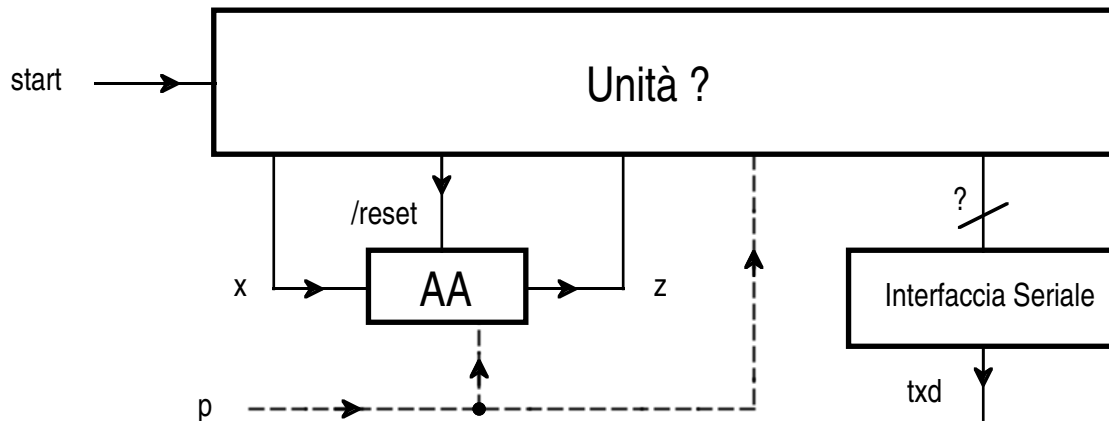
modello strutturale

15 luglio 93

1° Esercizio.

Il circuito AA è un circuito sequenziale di Moore con una variabile di ingresso ed una variabile di uscita che, partendo da uno stato interno iniziale (impostabile tramite la variabile per il reset asincrono */reset*), riconosce una sequenza costituita da $N = 8$ stati di ingresso.

Descrivere l'Unità ? che: i) inizia ad esaminare il circuito AA quando la variabile di ingresso *start* viene portata prima da 0 ad 1 e poi nuovamente a 0, ii) porta avanti l'esame con il fine di scoprire la sequenza riconosciuta da AA e iii) emette la sequenza individuata tramite l'interfaccia seriale.



NOTE

Il circuito AA si porta in uno stato interno iniziale se la variabile per il reset asincrono */reset*, normalmente ad 1, viene portata prima a 0 e poi ad 1. È importante ricordare che ogni fronte in salita del clock (dopo che */reset* è tornata ad 1) notifica un nuovo stato di ingresso per AA: in altre parole analizzare bene i problemi di temporizzazione.

Nella fase di emissione tramite l'interfaccia seriale gli 0 e gli 1 vanno codificati ASCII. L'interfaccia seriale è quella vista a lezione: non divagare ... ma limitarsi a gestirla correttamente.

SUGGERIMENTI:

Risolto l'esercizio, testarlo supponendo $N = 2$.

2° Esercizio.

Si consideri la relazione $z = /a*b*/c + a*b*c + a*/b*d + a*/c*/d$.

Una rete combinatoria sintetizzata in accordo a tale relazione è affetta da alee?

Se sì, come si possono eliminare ?

Prova Pratica

Prelevare due unsigned X ed Y e generare una coppia (segno, unsigned Z) tali che:

X ha 4 cifre in base 2 e rappresenta un integer x in complemento

Y ha 2 cifre in base 10 e rappresenta un integer y in complemento

Z ha 2 cifre in base 8 e la coppia (segno, Z) rappresenta $x+y$ in modulo e segno.

Andare a capo quando opportuno e ricordare che entrano ed escono codifiche ASCII.

1° Esercizio.

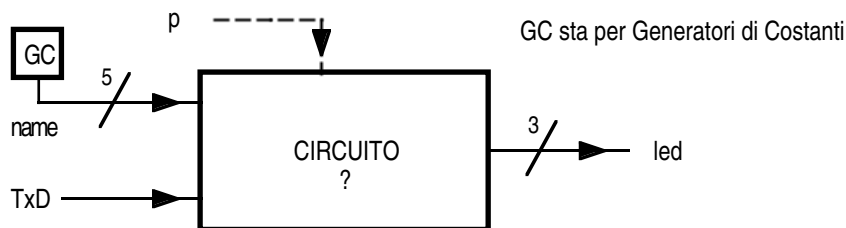
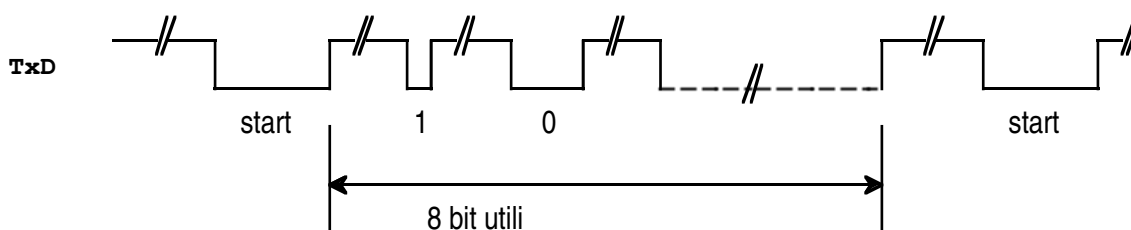


Fig. 1

Il circuito di Fig. 1 è, rispetto alla variabile di ingresso TxD, un ricevitore seriale di trame con 8 bit utili. Ogni volta che ha ricevuto una trama ed ha estratto gli 8 bit utili, il circuito confronta i 5 bit più significativi con il valore della variabile *nome*: se il confronto dà esito positivo, il circuito emette i rimanenti 3 bit tramite la variabile *led*, altrimenti torna ad aspettare una nuova trama, e così via all'infinito. Il formato di una trama è illustrato in Fig. 2 (ed è del tutto diverso da quello delle trame riportate sul testo). Il bit utile meno significativo è quello che segue il bit di start.

Descrivere il circuito, in modo che al reset esso inizi ad aspettare l'arrivo di una trama per estrarne i bit utili e procedere come illustrato sopra.



start: il valore di TxD sta a 0 per più di 20 e meno di 24 cicli del clock p

1: il valore di TxD sta a 0 per più di 4 e meno di 8 cicli del clock p

0: il valore di TxD sta a 0 per più di 12 e meno di 16 cicli del clock p

La condizione TxD ad 1 permane sempre per un tempo sufficientemente lungo da non creare alcun problema

Fig. 2

2° Esercizio.

Data la mappa in figura:

- indicare tutti gli implicant principali;
- trovare la lista (o le liste) di copertura irridondanti SP;
- individuare le eventuali alee del primo ordine presenti in una delle liste trovate nel punto b) e modificare la corrispondente lista in modo da eliminare le alee.

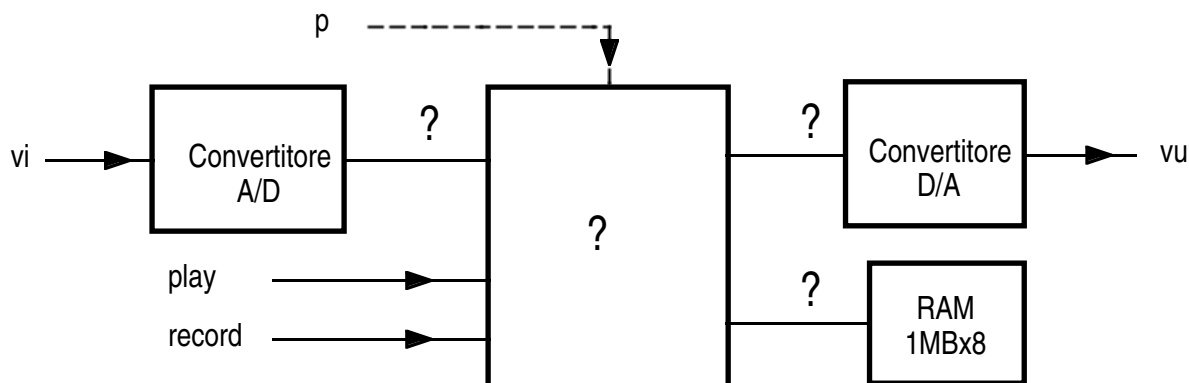
		x3 x2			
		00	01	11	10
x1 x0	00	0	1	1	1
	01	1	-	-	0
	11	1	-	-	0
	10	1	0	0	1

Prova Pratica: Scrivere un programma che

- legge da terminale un numero naturale X su 4 cifre in base 6;
- calcola $Y = |X^2 + 3|_{59 \text{ decimale}}$
- emette su video i numeri X e Y in decimale andando opportunamente a capo
- disegna su video una striscia di Y asterischi, andando a capo

5. torna al punto 1

1° ESERCIZIO



Descrivere il circuito ? che opera come segue.

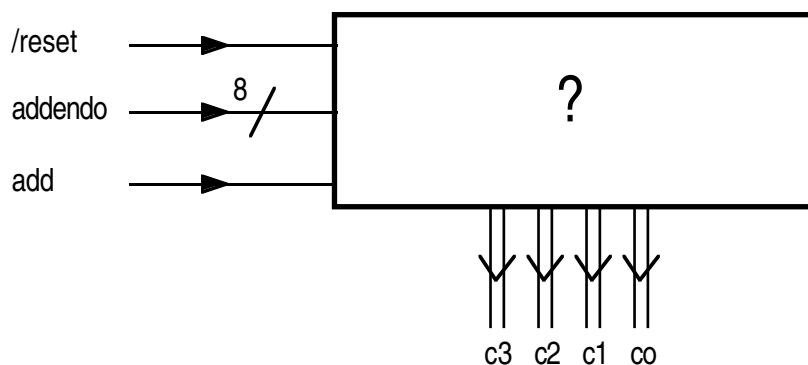
Ripete all'infinito:

Se la variabile *play* vale 0, ogni 125 impulsi del clock *p* legge dalla RAM 1 byte (a partire dall'indirizzo zero) e lo invia al convertitore D/A. Termina quando *play* vale 1 oppure quando tutta la memoria è stata letta.

Se la variabile *record* vale 0, ogni 125 impulsi di clock *p* preleva un campione dal convertitore A/D e lo memorizza in RAM (a partire dall'indirizzo zero). Termina quando *record* vale 1 oppure quando tutta la memoria è stata scritta.

Se entrambe le variabili *play* e *record* valgono 1 aspetta; è escluso il caso in cui entrambe le variabili *play* e *record* valgano 0.

2° ESERCIZIO



Progettare con il miglior dettaglio possibile il circuito ? che opera come segue.

La variabile */reset* resetta un registro accumulatore interno in cui, ogni volta che arriva un impulso tramite la variabile *add*, vengono sommati i numeri naturali binari a 8 bit supportati dalla variabile *addendo*. Il contenuto dell'accumulatore è emesso sotto forma di 4 cifre decimali *c3*, *c2*, *c1*, *c0* con codifica 8421. Si supponga che il totale nell'accumulatore non superi mai 9999.

PROVA PRATICA

Siano *a* e *b* due integer con $a \Leftrightarrow A$ e $b \Leftrightarrow B$ (rappresentati in complemento su 3 cifre in base 6). Scrivere un programma che preleva dalla tastiera i due numeri naturali *A* e *B*, ed emette $p=ab$ rappresentato in modulo e segno in base 10

Una stringa di bit ha le seguenti caratteristiche:

- lunghezza = 24K bit;
- primo bit = 0;
- numero massimo di 0 consecutivi = 255;
- numero massimo di 1 consecutivi = 255.

La stringa di bit va elaborata producendo una sequenza di numeri naturali (in base 2 su 8 bit) in accordo alle seguenti specifiche:

- Il primo numero naturale è il numero di 0 consecutivi con cui inizia la stringa;
- Il secondo numero naturale è il numero di 1 consecutivi che segue gli 0 di cui sopra;
- Il terzo numero naturale è il numero di 0 consecutivi che segue gli 1 di cui sopra;
-

Esempio di stringa:

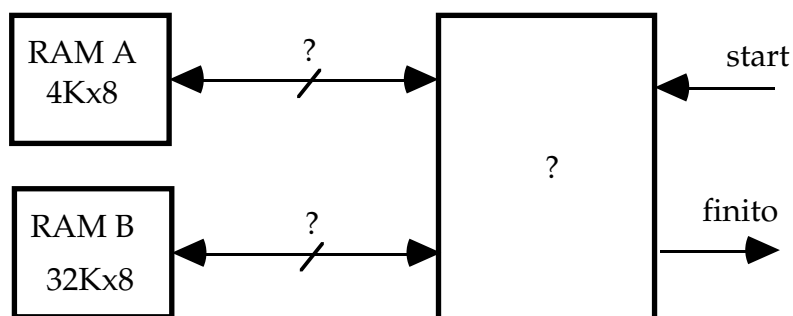
0 111 0000 111111 00 1...

Sequenza di numeri naturali da produrre:

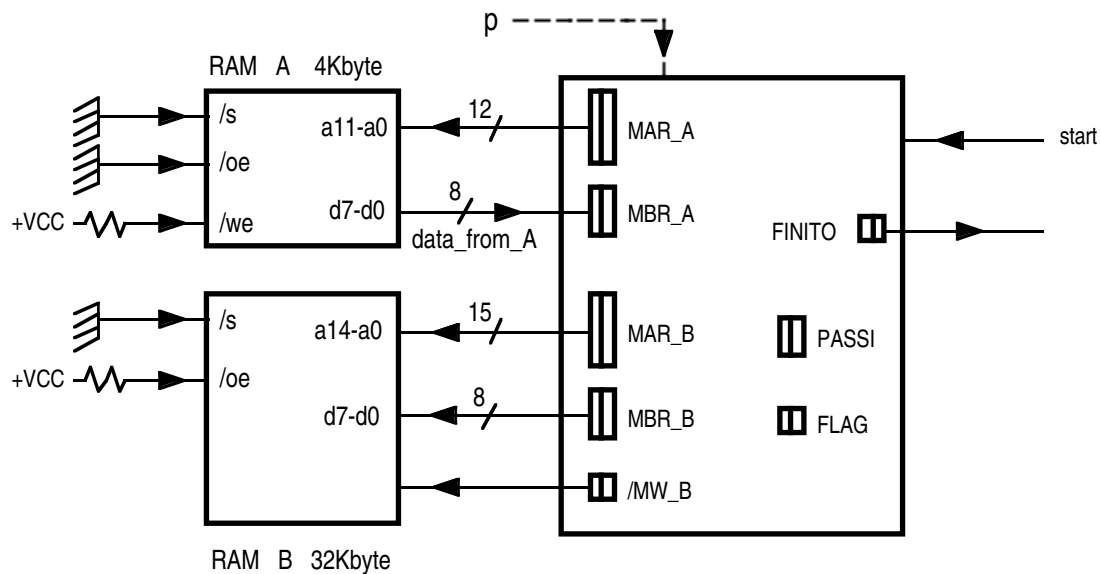
1, 3, 4, 6, 2, ...

Descrivere e sintetizzare una unità con Parte Operativa e Parte Controllo che trova la stringa di bit di ingresso memorizzata byte dopo byte in una RAM A da 4Kx8, elabora la stringa e lascia i numeri naturali risultato della elaborazione nella RAM B da 32Kx8, operando come segue:

- al reset iniziale asincrono si porta in uno stato interno iniziale L0 in cui pone **FINITO** a 0 e attende che la variabile di ingresso **start** vada ad 1 prima di iniziare l'elaborazione;
- elabora la stringa di bit di ingresso memorizzata nella RAM A, e scrive i risultati dell'elaborazione nella RAM B;
- pone ad 1 la variabile di uscita **finito**;
- attende che **start** passi a 0 e torna nello stato interno L0



Soluzione Prova Dicembre 93



DESCRIPTION

INPUTS

start: 1 bit variable.
data_from_A: 8 bit variable.

REGISTERS

/MW_B, FLAG, FINITO: 1 bit registers.
PASSI: 3 bit register.
MBR_A, MBR_B: 8 bit registers.
MAR_A: 12 bit register.
MAR_B: 15 bit register.

BODY

```

reset0:  /MW_B<-1; MAR_A<-0; MAR_B<-0; FINITO<-0 .
reset1:  MBR_B<-0; FLAG<-0; goto {if start eq 0 then reset1 else leggi}.

leggi:   MBR_A<-data_from_A; PASSI<-7; MAR_A<-increment(MAR_A);
        goto {if MAR_A eq 3072 then scrivi0 else scandisci}.

scandisci: MBR_B<-if bit7(MBR_A) eq FLAG then increment(MBR_B) else MBR_B;
        goto {if bit7(MBR_A) eq FLAG then continua else scrivi0}.

continua: PASSI<-decrement(PASSI); MBR_A<-rotate_left(MBR_A);
        goto {if PASSI eq 0 then leggi else scandisci}.

scrivi0:  /MW_B<-0.
scrivi1:  /MW_B<-1.
scrivi2:  MAR_B<-increment(MAR_B); MBR_B<-0; FLAG<-not(FLAG);
        goto {if MAR_A eq 3073 then fine else scandisci}.

fine:     FINITO<-1; goto {if start eq 1 then fine else reset0}.
    
```

END_BODY

END_DESCRIPTION

Nome, Cognome, Numero di matricola _____

X=

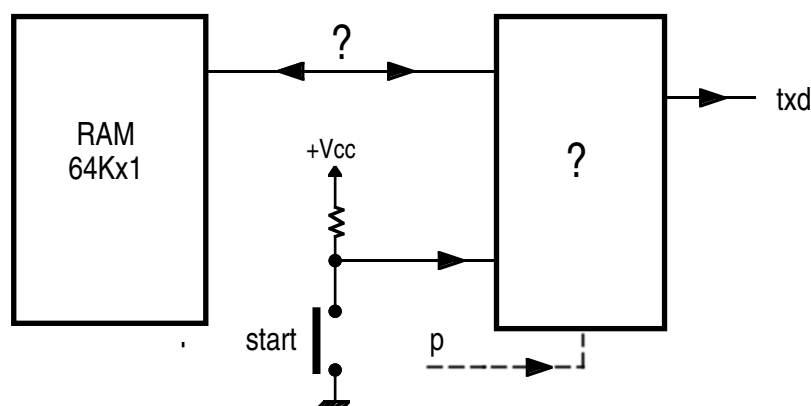
7			
---	--	--	--

ultime 3 cifre del numero di matricola

Esercizio 1

Sia a un intero rappresentato in complemento alla radice dal numero naturale X (su 4 cifre in base 12) costruito sopra. Determinare la rappresentazione Y di a su 5 cifre in base 8.

Esercizio 2.



Il circuito ? deve trasmettere il contenuto della memoria RAM da 64Kx1 bit tramite la linea seriale txd , emettendo un bit ad ogni ciclo di clock (*senza perdere mai il passo*). La trasmissione avviene come segue:

1. al reset, il circuito pone txd a 0 e attende che venga premuto e rilasciato il tasto *start*
2. il circuito trasmette la sequenza 01111110 (detta sequenza FLAG)
3. il circuito trasmette il contenuto della memoria, inserendo, dopo cinque bit 1 consecutivi, un bit 0.
4. il circuito trasmette la sequenza 01111110 (FLAG), torna al punto 1) e così via

Esempio: se il contenuto della memoria è 000111111100111110111...

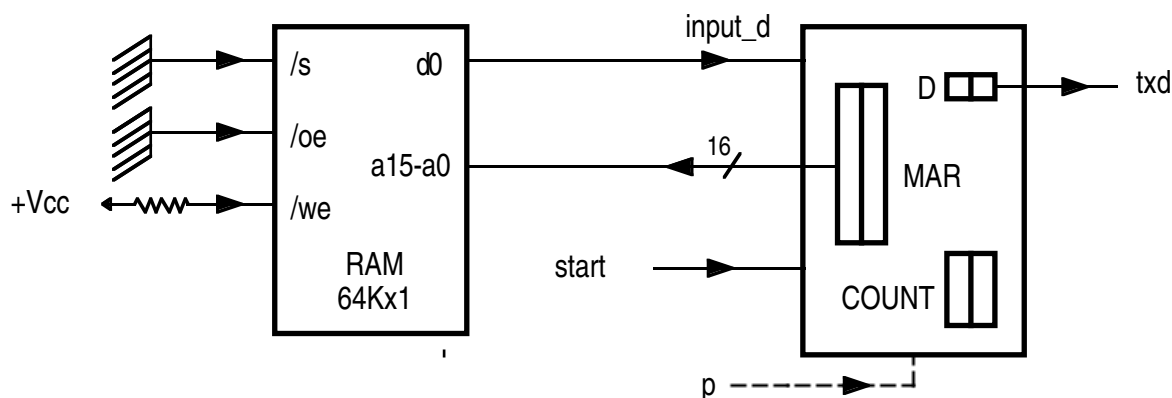
la sequenza trasmessa è (sono evidenziati la sequenza FLAG e gli 0 inseriti dal circuito dopo cinque bit 1):

0111111000011111011001111100111.....01111110

NOTA. La RAM è sufficientemente veloce da non dare alcun problema. Sono ammessi salti a tre vie del tipo **goto {if then else {if then else}}**

Esercizio 3.

Sintetizzare la **parte operativa** del circuito di cui sopra, descrivendo a livello di porte **tutti gli atomi** (anche quelli predefiniti quali increment, decrement, ecc.) eventualmente utilizzati.



DESCRIPTION

INPUTS

start, input_d: 1 bit variables.

REGISTERS

D: 1 bit registers.
COUNT: 3 bit register.
MAR: 16 bit register.

ATOMS

cinquel(1 bit, 3 bit): 1 bit produced.

BODY

```
Reset0: D<-0; MAR<-0000H;
        goto {if start eq 1 then Reset0 else Reset1}.
Reset1: goto {if start eq 0 then Reset1 else PR0}.

PR0:    D<-0; COUNT<-0.
PR1:    D<-1; COUNT<-increment(COUNT);
        goto {if COUNT eq 5 then PR2 else PR1}.
PR2:    D<-0; COUNT<-0.

RAM0:   D<- input_d; MAR<-increment(MAR);
        COUNT<-if input_d eq 0 then 0 else increment(COUNT);
        goto {if cinquel(input_d,COUNT) eq 1 then ZERO else
                if MAR eq 0FFFFH then PS0 else RAM0}.

PS0:    D<-0; COUNT<-0.
PS1:    D<-1; COUNT<-increment(COUNT);
        goto {if COUNT eq 5 then PS1 else Ps2}.
PS2:    D<-0; goto Reset0.

ZERO:   D<-0; COUNT<-0;
        goto {if MAR eq 0000H then PS0 else RAM0}.
```

END_BODY

END_DESCRIPTION

cinquel(input_d,COUNT) dà 1 se input_d è 1 e COUNT contiene 4

[illegible]

Scrivere un programma che legge da tastiera un numero N in base 10 ($1 \leq N \leq 10$) ed emette la figura sotto indicata, di larghezza $2N+1$ caratteri.

A 20x20 grid of dots with 'X' marks forming a diamond shape. The 'X' marks are located at positions (row, column) where the absolute value of the row index minus the absolute value of the column index is less than or equal to 9. The diamond is centered at (10, 10).

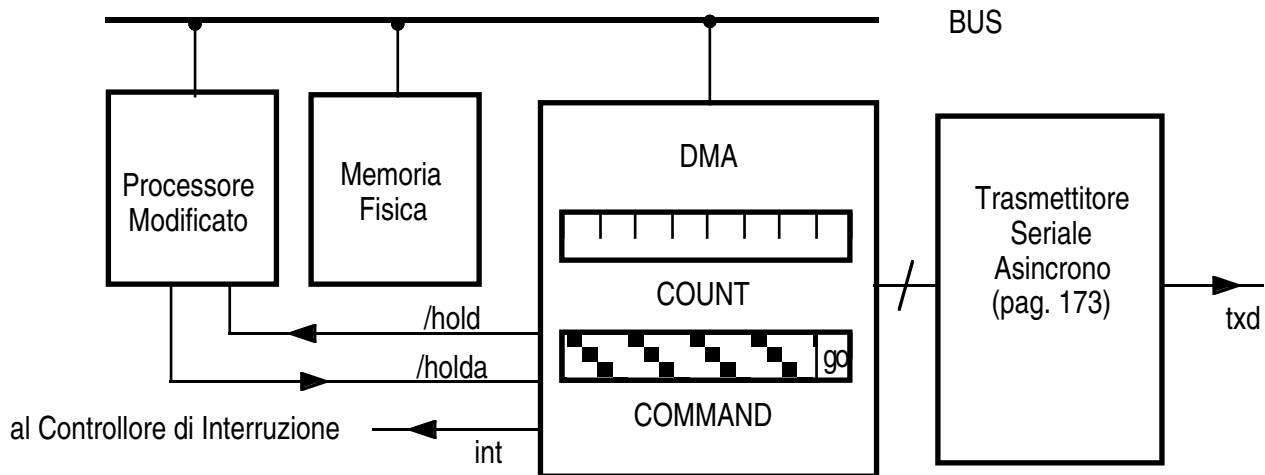
Scrivere un programma che:

- ```

1 Preleva dalla tastiera un primo numero naturale X a 2 cifre decimali, scartando eventuali
 caratteri spuri e facendo l'eco sul monitor delle cifre utili.
2 Va a capo su una nuova linea.
3 Preleva dalla tastiera un secondo numero naturale Y a 3 cifre decimali, scartando eventuali
 caratteri spuri e facendo l'eco sul monitor delle cifre utili.
2 Va a capo su una nuova linea.
4 Emette sul monitor, cifra dopo cifra, il numero naturale $P = X \cdot Y$ a 5 cifre decimali
5 Termina

```

Esercizio 1



Il processore visto a lezione è stato modificato nel seguente modo:

- 1) tutti i suoi piedini (esclusi i piedini per il clock, il reset e la massa) sono supportati da porte a tre-stati, cosicché il processore può isolarsi completamente dalla memoria e dallo spazio di I/O ponendo tali porte in alta impedenza;
- 2) è stata aggiunta la variabile di ingresso */hold* e la variabile di uscita */holda* (non supportata da una porta a tre stati);
- 3) quando */hold* viene messa a 0 dalle circuiterie esterne, il processore si isola dalla memoria e dallo spazio di I/O, non compie alcuna evoluzione e pone */holda* a 0; quando */hold* viene riportato ad 1 il processore rimuove il suo isolamento, pone */holda* ad 1 e riprende la sua normale evoluzione.

Progettare il modulo DMA in modo che esso rispetti le seguenti regole:

- a) E' gestibile software con il seguente programma:

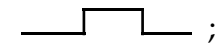
```

STI ;solo per chi conosce l'interrupt
...
preparazione nel segmento di selettore 0100H a partire dall'offset 0000H
degli N byte da emettere via DMA (e Trasmettitore Seriale)

MOV sincrout,0 ;solo per chi conosce l'interrupt
MOV AL,N
OUT IO:OFFSET COUNT, AL ; offset COUNT = 0010H
MOV AL,01H
OUT IO:OFFSET COMMAND, AL ; offset COMMAND = 0011H
MOV AL,00H
OUT IO:OFFSET COMMAND, AL
...
aspetta: CMP sincrout,0 ;solo per chi conosce l'interrupt
 JE aspetta
 ...

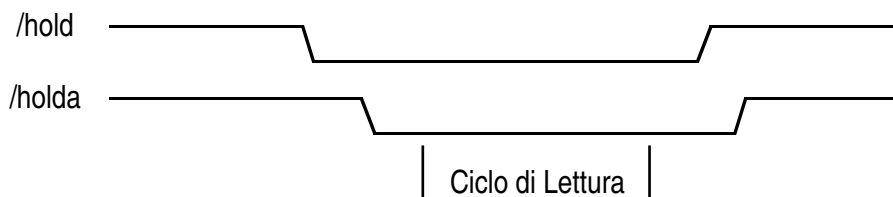
```

dove *sincrout* è una variabile definita in un segmento di selettore simbolico *PARAMETER*

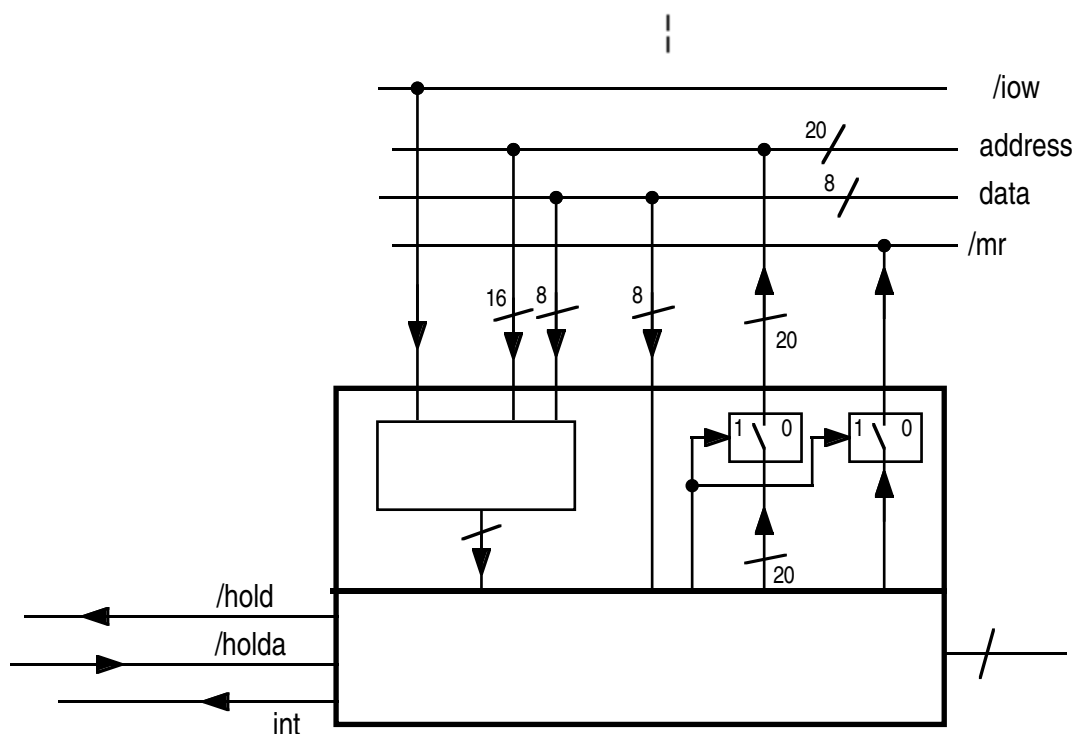
- b) Si attiva quando *go* va prima ad 1 e poi a 0 e compie tanti passi per quanti ne sono scritti in COUNT, dopo di che porta *int* prima ad 1 e poi di nuovo a 0  ;
- c) Ogni passo consiste nel prelevare un byte dalla memoria e nel passarlo al Trasmettitore Seriale in modo corretto; i prelievi vengono fatti dal segmento di selettore 0100H a partire dall'offset 0000H in poi;



- d) Rispetta il seguente handshake con il Processore, in modo che: i) quando intende leggere in memoria blocca il processore spingendolo ad isolarsi e ii) quando non sta accedendo alla memoria si isola dalla memoria stessa e sblocca il processore.



**SUGGERIMENTO:** Progettare il modulo DMA utilizzando interfacce parallele di uscita senza handshake per implementare i registri COUNT e COMMAND e una unità con parte operativa e parte controllo (di quest'ultima unità solo la descrizione) per portare avanti l'algoritmo e pilotare il Trasmettitore Seriale. Si inseriscano e si comandino opportune porte a tre-stati per assicurare, quando necessario, l'isolamento del modulo DMA dalla memoria.



## Esercizio 2

Nuovo Programma i) Vecchi Programmi ii) :

- Scrivere un *driver* per la gestione dell'unica richiesta di interruzione emessa, nell'ipotesi che il controllore di interruzione e la tabella delle interruzioni siano già sistemati.
- Inserire un registro di stato nel DMA con un flag *fi* che va ad 1 quando *int* (si supponga che l'intervallo tra due impulsi di *int* sia così grande da non dare alcun problema) e modificare opportunamente il programma di gestione del modulo DMA.

### Prova Pratica

Scrivere un programma che simula un flip-flop edge-triggered con slave SR e si comporta come segue:

**d**= attesa di 0 od 1 dalla tastiera

**Fronte su p** attesa di 0 e poi 1 dalla tastiera

**Master:**

**Stato Interno** = .....

**Stato di uscita**

**S**= .....

**R**= .....

**Slave:**

**Stato di uscita**

**Q**= .....

-----

Organizzare due sottoprogrammi che permettano di gestire comodamente il monitor in modalità alfanumerica 25 righe x 80 colonne.

- 1) nome: ini\_monitor  
funzione: inizializza il monitor pulendolo
- 2) nome: mostra  
funzione: aggiunge un carattere sul monitor  
parametri da passare: AL <= codifica ASCII del carattere  
CL <= coordinata di colonna (da 0 a 79)  
DL <= coordinata di riga (da 0 a 24)

Scrivere un programma che faccia sul monitor il seguente disegno:

```
 *
 * *
* *
```

```
*
*
*
```

## Esercizio n.1

Nome, Cognome, Numero di matricola \_\_\_\_\_

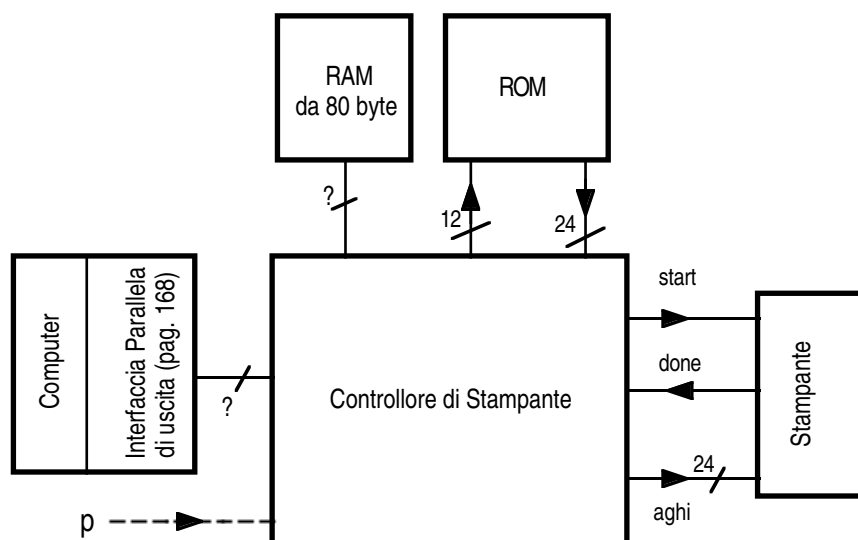
X=

|   |  |  |  |
|---|--|--|--|
| 7 |  |  |  |
|---|--|--|--|

ultime 3 cifre del numero di matricola

Sia  $a$  un intero rappresentato in complemento alla radice dal numero naturale  $X$  (su 4 cifre in base 14) costruito sopra. Determinare la rappresentazione  $Y$  di  $a$  su 5 cifre in base 12.

## Esercizio n. 2



Descrivere il controllore di stampante riportato in figura nell'ipotesi che esso operi in accordo a quanto segue.

Al reset asincrono il controllore si porta nello stato corrispondente al passo 0 ed esegue all'infinito i seguenti passi.

- P0) Pone *start* a 0 (il suo effetto è di comandare il posizionamento su una nuova linea del carrello della stampante).
- P1) Preleva dall'interfaccia parallela di uscita e memorizza nella RAM una sequenza di codifiche ASCII ad 8 bit arrestandosi quando si accorge di aver prelevato la codifica ASCII 0AH (newline) o comunque quando ha prelevato e memorizzato 80 codifiche ASCII. Si dica  $N$  il numero delle codifiche ASCII memorizzate, esclusa la eventuale codifica 0AH.
- P2) Attende che *done* passi a 1 e quando ciò accade pone *start* a 1
- P3) Attende che *done* passi a 0 e quando ciò accade comanda la stampa degli  $N$  caratteri la cui codifica ASCII è in RAM, emettendo, secondo le specifiche dettagliate di seguito e tramite le variabili *aghi*, una sequenza di  $N * 16$  stati di uscita a 24 bit, al ritmo di uno stato di uscita ogni 10 cicli di clock.
- P4) Torna al passo 0

La sequenza di stati di uscita da emettere va prelevata dalla ROM, tenendo presente che essa contiene, per ogni possibile codifica ASCII, l'immagine del carattere corrispondente e che questa immagine occupa 16 locazioni da 24 bit a partire dalla locazione di indirizzo *codifica ASCII* \* 16

### Prova Pratica

Scrivere un programma che simula il funzionamento dell'unità POPC descritta di seguito:

#### INPUTS

**d\_in:** 8 bit variable.

#### REGISTERS

**X, Y, COUNT:** 8 bit registers.

#### BODY

**L0:** X←-d\_in; Y←-X; COUNT←-3.

**L1:** X←- Y; Y←-X; COUNT←-decrement(COUNT);  
goto {if COUNT eq 0 then L0 else L1}.

#### END\_BODY

Partendo da una condizione iniziale in cui X, Y e contengono una codifica ASCII casuale, COUNT contiene una cifra decimale casuale e STAR contiene la codifica dello stato interno di etichetta L0, il simulatore si comporta come segue:

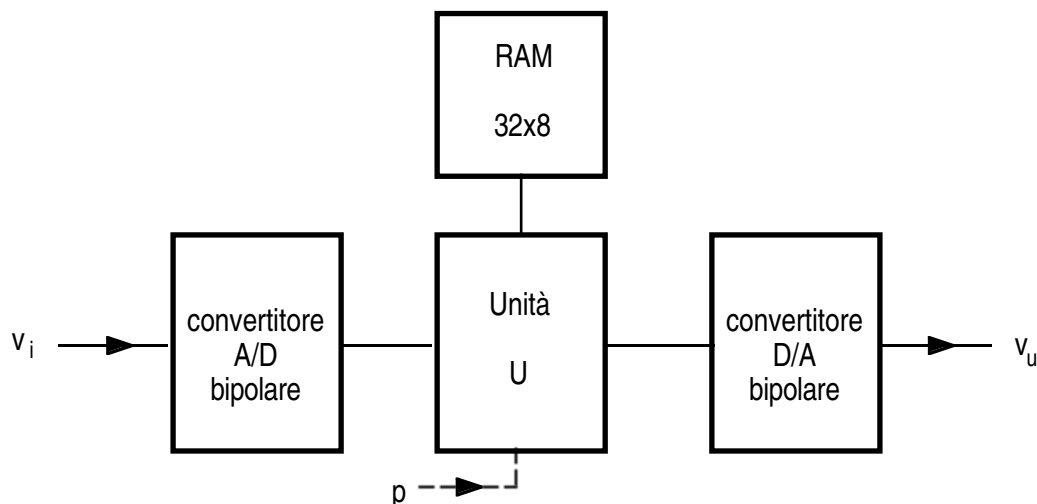
- 1) emette (su una nuova linea del video) l'etichetta dello stato interno, i caratteri contenuti in X e Y e la cifra decimale contenuta in COUNT
- 2) attende che venga premuto un tasto e quando ciò accade interpreta la codifica ASCII corrispondente come il nuovo valore per *cod\_ASCII*
- 3) attende che venga premuto ENTER e quando ciò accade interpreta questo evento come l'arrivo di un nuovo fronte di salita del clock, e così via ....

### Esercizio 1

**Descrivere** l'unità U sotto indicata. L'unità U, a regime, mantiene gli ultimi 32 campioni della tensione  $v_i$  nella RAM, ne presenta il valor medio al convertitore D/A, preleva un nuovo campione di  $v_i$  dal convertitore A/D e così via all'infinito. L'unità deve compiere un ciclo completo (presentazione di un nuovo valor medio e prelievo e memorizzazione di un nuovo campione di  $v_i$ ) **esattamente in 15 periodi di clock**. Si supponga che il tempo di conversione del convertitore A/D, anche se da considerarsi non noto, non influisca per più di 5 periodi di clock.

L'unità al reset iniziale inizializza la RAM con valori corrispondenti a  $v_i=0$ , dopo di che si considera a regime.

**ATTENZIONE:** i convertitori lavorano in **binario bipolare** su 8 bit.

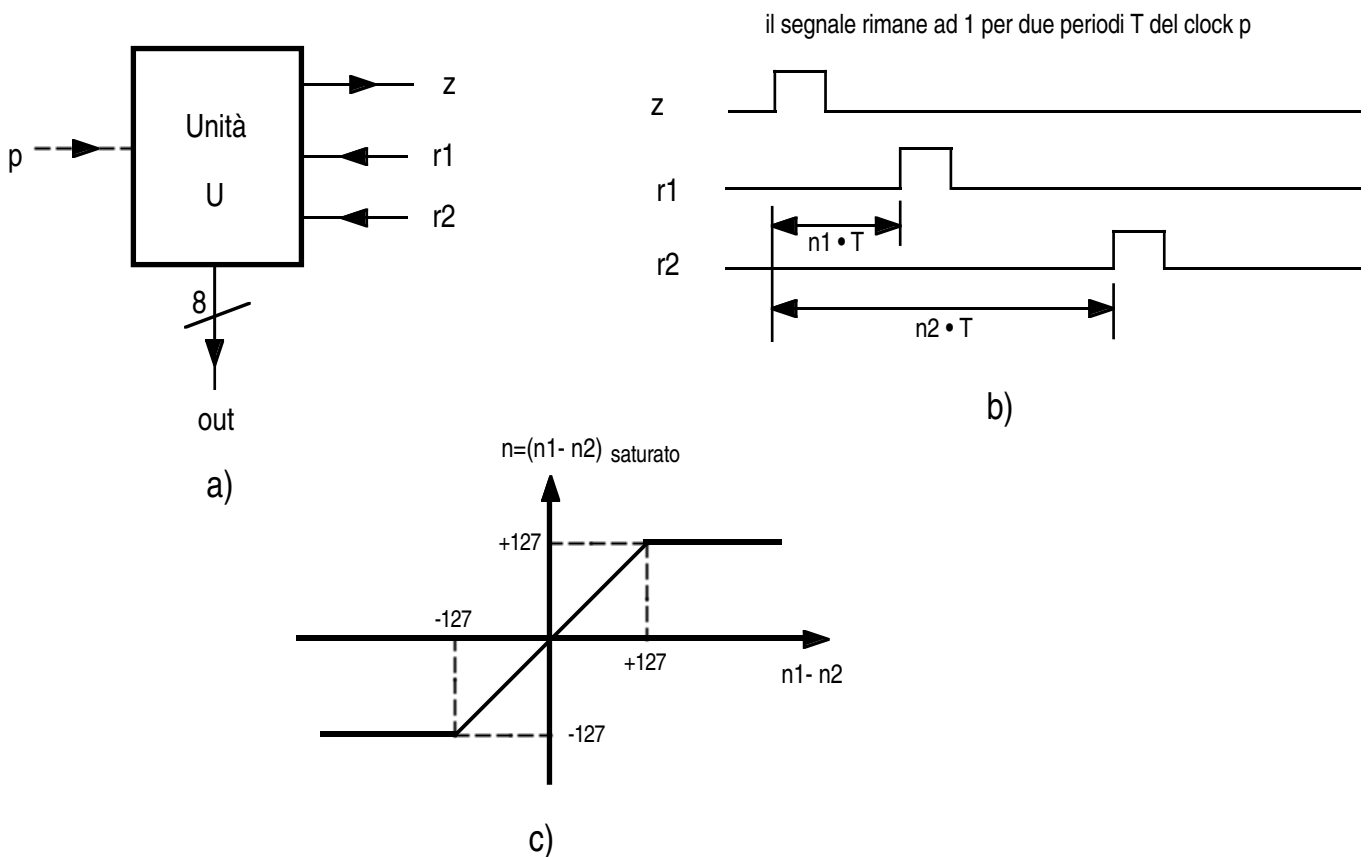


### Esercizio 2

**Descrivere e sintetizzare** un circuito sequenziale **asincrono** con due variabili di ingresso  $a$  e  $b$ , con una variabile di uscita  $z$  e che si evolve come segue: partendo da una condizione di stabilità in cui  $a = 0$ ,  $b = 0$  e  $z = 0$ , la variabile  $z$  va ad 1 solo quando entrambe le variabili di ingresso sono andate ad 1; la variabile  $z$  torna a 0 non appena una delle due variabili di ingresso torna a 0, ma può tornare ad 1 solo se entrambe le variabili di ingresso ripassano per 0.

**Esercizio 1**

Si consideri l'unità riportata in fig. (a). Essa è dotata di una variabile di uscita  $z$  ad un bit, di due variabili di ingresso  $r1$  ed  $r2$  ad un bit e di una variabile di uscita  $out$  ad 8 bit. L'unità emette tramite  $z$  un segnale (fig. (b)) e ne raccoglie tramite  $r1$  l'eco proveniente da un ostacolo e tramite  $r2$  l'eco proveniente da un diverso ostacolo (l'intervallo temporale tra l'arrivo di un'eco e l'altra è un indice della distanza dei due ostacoli). L'unità emette tramite la variabile  $out$  un integer  $n$  in complemento a due che rappresenta il numero dei periodi di clock che intercorrono tra l'arrivo di un'eco e l'altra, con una saturazione a  $+127$  se  $n$  supera  $127$  e a  $-127$  se  $n$  è inferiore a  $-127$  (fig. (c)). Ogni eco torna all'unità al più dopo 950 periodi di clock e l'unità si evolve in modo ciclico, emettendo un nuovo segnale e un nuovo numero  $n$  esattamente ogni 1000 cicli di clock.



Descrivere l'unità supposto che al reset asincrono tutti i registri della parte operativa siano a zero; si pensi inoltre di disporre di un atomo

*satura*(10 bit, 10 bit): 8 bit produced

che riceve due numeri naturali  $n1$  ed  $n2$  a 10 bit, proporzionali rispettivamente al ritardo dell'eco su  $r1$  ed al ritardo dell'eco su  $r2$ , e che genera correttamente il numero  $n$  da emettere tramite  $out$ .

**Esercizio 2**

Progettare l'atomo *satura*

## Soluzione luglio 94 (III appello)

### DESCRIPTION

#### INPUTS

r1,r2: 1 bit variables.

#### REGISTERS

Z: 1 bit register .

OUT: 8 bit register .

CT1, CT2, CT1000: 10 bit registers.

#### ATOMS

satura(10 bits,10 bits): 8 bit produced .

#### BODY

L0: OUT<-satura(CT1,CT2); Z<-1; CT1<-0; CT2<-0; CT1000<-0.

T: Z<-1; CT1000<- increment(CT1000).

WAIT: Z<-0; CT1000<- increment(CT1000);  
CT1<-if r1 eq 0 then increment(CT1) else CT1;  
CT2<-if r2 eq 0 then increment(CT2) else CT2;  
goto {if r1 eq 1 then W2 else {if r2 eq 1 then W1 else WAIT}}.

W1: CT1000<- increment(CT1000);  
CT1<-if r1 eq 0 then increment(CT1) else CT1;  
goto {if r1 eq 1 then W1000 else W1}.

W2: CT1000<- increment(CT1000);  
CT2<-if r2 eq 0 then increment(CT2) else CT2;  
goto {if r2 eq 1 then W1000 else W2}.

W1000: CT1000<- increment(CT1000);  
goto {if CT1000 eq 998 then L0 else W1000}.

END\_BODY

END\_DESCRIPTION

## Altra Soluzione luglio 94 (III appello)

### DESCRIPTION

#### INPUTS

r1,r2: 1 bit variables.

#### REGISTERS

Z, F1, F2: 1 bit register .

OUT: 8 bit register .

CT1, CT2, CT1000: 10 bit registers.

#### ATOMS

satura(10 bits,10 bits): 8 bit produced .

#### BODY

L0: OUT<-satura(CT1,CT2); Z<-1; CT1<-0; CT2<-0; CT1000<-0;  
F1<- 0; F2 <- 0.

T: Z<-1; CT1000<- increment(CT1000).

WAIT: Z<-0; CT1000<- increment(CT1000);  
CT1<-if F1 eq 0 then increment(CT1) else CT1;  
CT2<-if F2 eq 0 then increment(CT2) else CT2;  
F1<- or(r1, F1); F2<- or(r2,F2);  
goto {if CT1000 eq 998 then L0 else WAIT}.

END\_BODY

END\_DESCRIPTION

Un processore è capace di lavorare solo su byte e di indirizzare una memoria lineare non segmentata da 256 byte.

Si simuli un calcolatore basato su tale processore, nell'ipotesi che le istruzioni che esso esegue siano tre (la memoria è indirizzata con l'indirizzamento diretto):

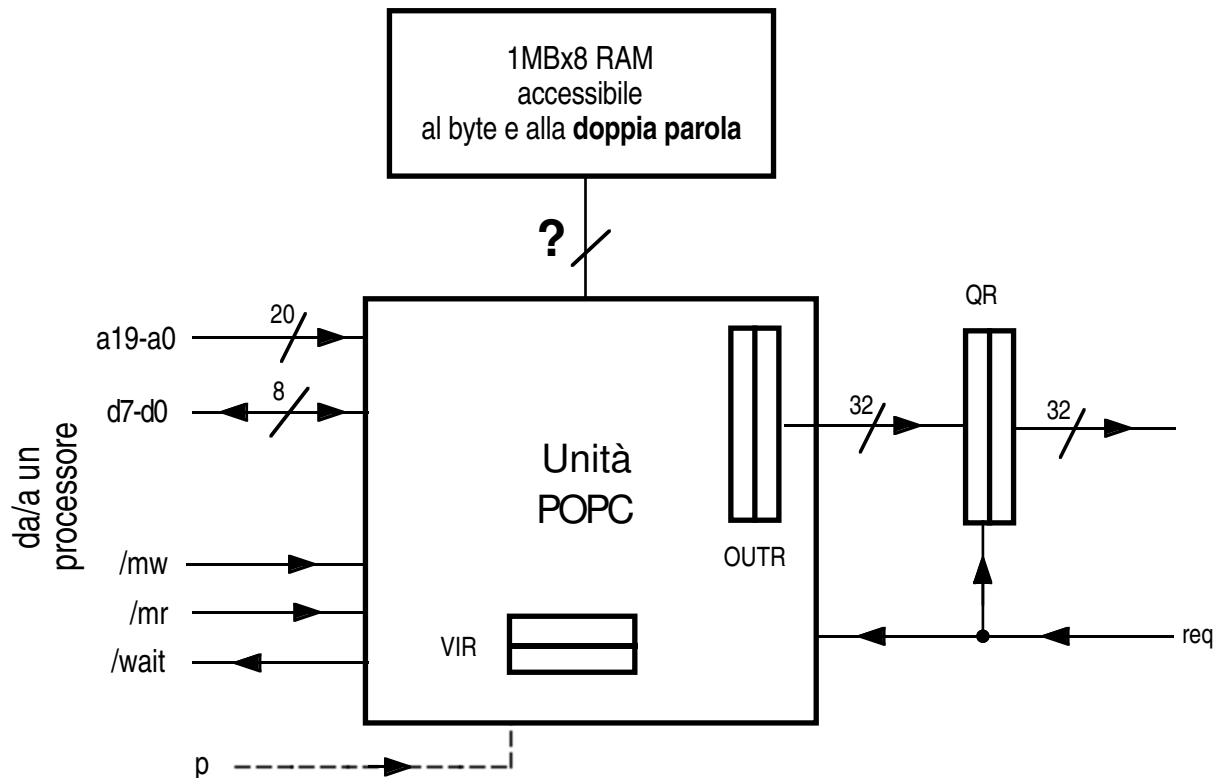
|          |  |        |  |          |  |                                      |
|----------|--|--------|--|----------|--|--------------------------------------|
| 00000000 |  | offset |  |          |  | ; OUT dalla memoria al video         |
| 00000001 |  | offset |  |          |  | ; IN dalla tastiera alla memoria     |
| 00000010 |  | offset |  | operando |  | ; ADD memoria con operando immediato |

Nel simulare le istruzioni di IN ed OUT si usino i sottoprogrammi di utilità INPUT ed OUTPUT, in modo che tali istruzioni siano utilizzabili in pratica. In particolare si adorni il sottoprogramma INPUT in modo che l'istruzione IN prelevi da tastiera solo codifiche di lettere maiuscole.

Si metta nella memoria simulata del processore e lo si faccia partire al reset (simulato), un programma (in linguaggio macchina del processore simulato) che:

- 1) preleva la codifica ASCII di una lettera maiuscola (tramite l'istruzione IN);
- 2) modifica la codifica iniziale in modo da renderla, alla fine, codifica della corrispondente lettera minuscola;
- 3) emetta la codifica finale (tramite l'istruzione OUT).



**Esercizio n. 1**

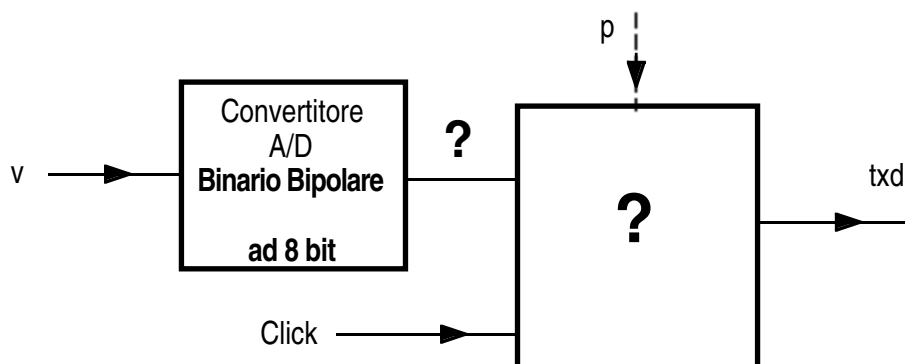
Realizzare la memoria RAM in figura partendo da 4 moduli standard da 256Kx8. Definire a piacere le modalità secondo cui indicare gli accessi ai byte e alle doppie parole

**Esercizio n. 2**

Descrivere l'unità con parte operativa e parte controllo che, partendo da una condizione iniziale in cui VIR contiene 0, svolge ciclicamente le seguenti due attività dando la precedenza, in caso di richieste contemporanee, alle attività N. 2

- 1) Quando riceve un impulso su *req* copia il contenuto di OUTR in QR (azione già supportata nello schema) e compie un ciclo di lettura in RAM della doppia parola indirizzata dal contenuto di VIR; il risultato della lettura va in OUTR ed il contenuto di VIR viene adeguatamente aggiornato.
- 2) Risponde alle richieste di lettura (/mr a 0) e di scrittura (/mw a 0) di un byte che provengono dal processore. La variabile /wait inserisce stati di wait nel processore, nel senso che quando è a 0 prolunga, nel processore, i cicli di lettura/scrittura, con il risultato che il processore non conclude e non riporta ad 1 la variabile /mr (lettura) o /mw (scrittura). L'unità tiene quindi /wait normalmente a 0, la porta ad 1 solo quando essa è in grado di completare un ciclo di lettura/scrittura comandato dal processore e la rimette a 0 quando /mr (lettura) o /mw (scrittura) torna ad 1.

## 1° Esercizio



Descrivere e **sintetizzare** il circuito ? che si evolve ciclicamente come segue:

- campiona ed emette ogni 50 cicli del clock  $p$  il valore della tensione  $v$  espresso in complemento a due, con la sostituzione del byte 10000000 (espressione dell'estremo negativo di  $v$ ) con il byte 10000001 (espressione del valore più vicino all'estremo negativo);
- quando arriva un impulso su *click*, termina la eventuale emissione in corso e poi emette il byte 10000000.

L'emissione viene effettuata in seriale start/stop utilizzando il modulo Trasmettitore riportato nel libro a pag. 173 all'interno della fig. 22b). Il convertitore A/D ed il Trasmettitore seriale NON VANNO DESCRITTI. Il convertitore è sufficientemente veloce da convertire in un tempo inferiore (quanto basta) a quello corrispondente ai 50 cicli di clock. Non confondere *click* con clock.

## 2° Esercizio

**Siano**

numero di matricola modulo 6  
 $a \leftrightarrow$ 

|   |   |
|---|---|
| 4 | 1 |
|---|---|

 su 2 cifre in base 6 in complemento

numero di matricola modulo 5  
 $b \leftrightarrow$ 

|   |   |
|---|---|
| 2 | 1 |
|---|---|

 su 2 cifre in base 6 in complemento

**Trovare**

$a \ b \leftrightarrow$ 

|  |  |  |  |
|--|--|--|--|
|  |  |  |  |
|--|--|--|--|

 su 4 cifre in base 8 in complemento