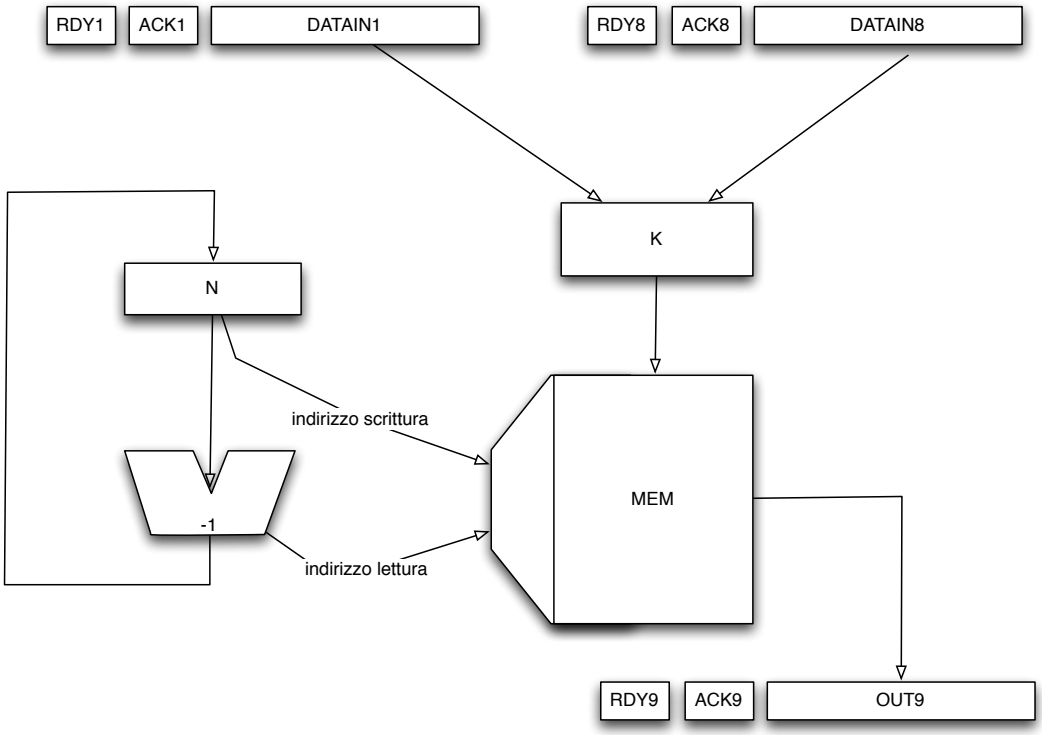
Bozza di soluzione

Domanda 1

Senza controllo residuo, ho un'unica microistruzione:

0. (zero(N),N0,RDY9,RDY8,...,RDY1=0-1 --------)
   MEM[N-1]->OUT9,N-1->N,set ACK9, reset RDY9, 0
   (=010--------) nop,0 // coda piena aspetto letture
   (=1-100000000) nop,0 // coda vuota aspetto scritture
   (=00010000000) reset RDY8, set ACK8, DATAIN8 -> MEM[N], N+1 -> N, 0
   (=00001000000) reset RDY7, set ACK7, DATAIN7 -> MEM[N], N+1 -> N, 0
   ....
   (=00010000000) reset RDY1, set ACK1, DATAIN1 -> MEM[N], N+1 -> N, 0

Lo schema relativo è

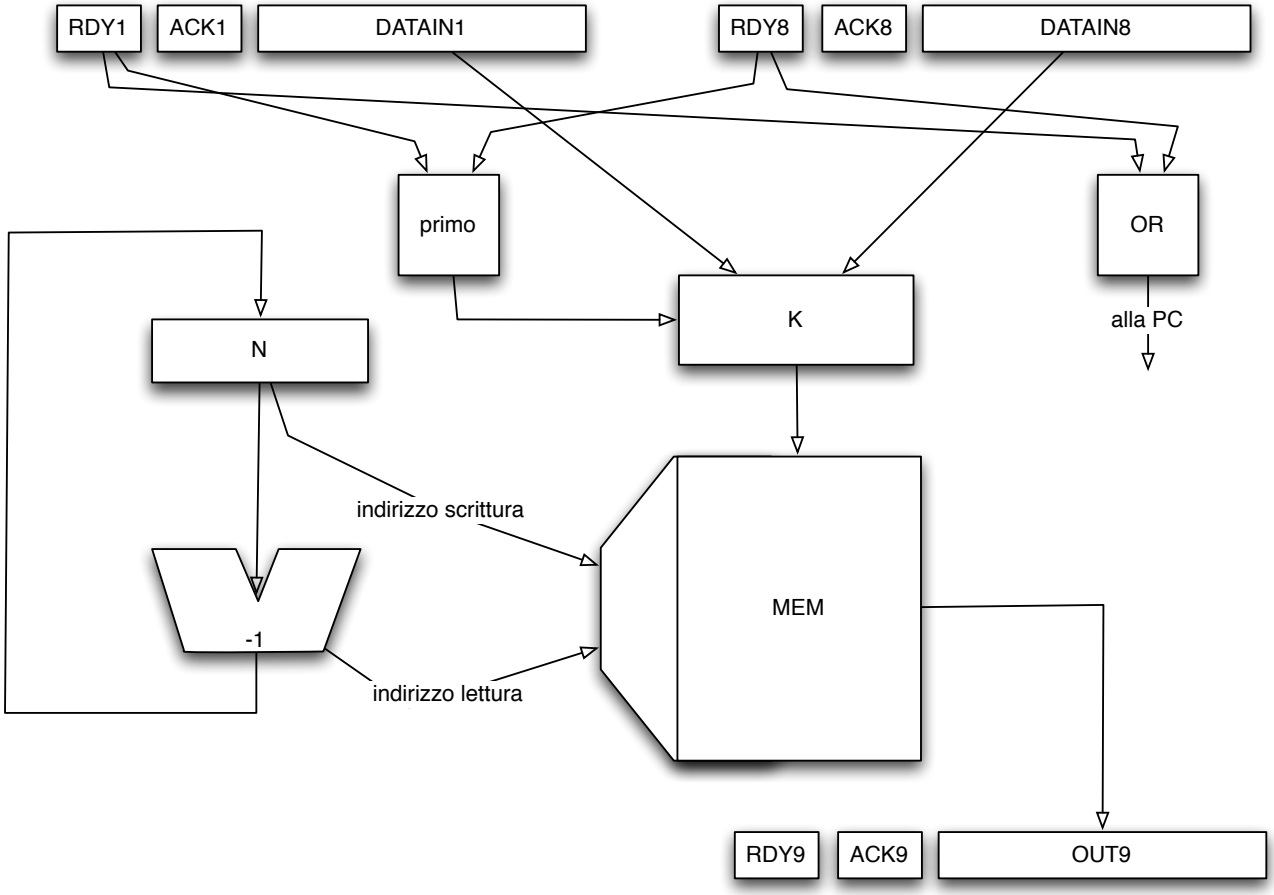


In questo caso il ciclo di clock può essere calcolato come segue:
t omega PO = 5 tp (per calcolare zero(N))
t omega PC = 3 tp (più di otto ingressi)
t sigma PC = 0 (c'e' uno stato solo ...)
t sigma PO = 15 tp (alu + mem)

per un totale di 5 tp + 3tp + 15tp + tp = 24 tp

ATTENZIONE: Riportare su tutti i fogli consegnati, nome, cognome, numero di matricola e corso.
I risultati e il calendario degli orali saranno pubblicati sulla pagina WEB del corso appena disponibili.

Utilizzando la tecnica del controllo residuo si può ottimizzare la PC, riducendo sensilmente il numero delle variabili di condizionamento. Supponendo di avere una funzione *primo* che restituisce il log in base 2 dell'indice del bit a 1 di più alto peso in una configurazione di 8 bit, e assumendo che la PO sia realizzata come:



possiamo scrivere il microprogramma come segue

0. (zero(N),N0,RDY9,OR(RDY1-8)=0-1-)
MEM[N-1] -> OUT9, reset RDY9, set ACK9, N-1 -> N,0 / / lettura
(=010-) nop,0
(=1001) nop,0
(=0001) DATAIN[primo(RDY1-RDY8)] -> MEM[N], N+1->N, set ACK(primo(RDY1-RDY8), reset RDY(primo(RDY1-RDY8))

in questo caso, la parte controllo risulta assolutamente più semplice, per via del minor numero di variabili di condizianamento. Possiamo assumere classicamente che t omega PC sia pari a 2 tp. D'altro canto l'OR sui RDYi viene calcolato in parallelo alo zero(N) e quindi non impatta sui tempi della PO. Il ciclo di clock sarà quindi di un tp minore rispetto all'altro caso. Non significativo, da un certo punto di vista, se non fosse che la PC è decisamente più semplice del caso precedente.

ATTENZIONE: Riportare su tutti i fogli consegnati, nome, cognome, numero di matricola e corso.
I risultati e il calendario degli orali saranno pubblicati sulla pagina WEB del corso appena disponibili.

In entrambi i casi, visto che la PC ha un solo stato, andava detto che l'unità può essere realizzata come una rete sequenziale (e magari darne la realizzazione di massima).

ATTENZIONE: Riportare su tutti i fogli consegnati, nome, cognome, numero di matricola e corso.
I risultati e il calendario degli orali saranno pubblicati sulla pagina WEB del corso appena disponibili.

Domanda 2

Supponendo che il buffer del canale di comunicazione sia di 8K (1K*64bit), il codice di cliente e servente, relativo alle comunicazioni. potrebbe essere:

```
Client i ::
   channel out toServer, channel in fromServer(1);
  ...
  for(int i=0; i<1K; i++) {
    send(toServer, matrice[i][]);
  }
  ...
  receive(fromServer,vettoreRisposa);
  ---

Server ::
  channel out toClient[nClient], channel in fromClients[nClients] (nClients);
  ...
  while(true) {
    alternative {
      priority P , receive(fromClient[1], matr[0][]) do {
         for(i=1; i<1K; i++) {
            receive(fromClient[1], matr[i][]) ;
         }
         vettoreRisultato = elaboraMatrice(matr);
         send(toClient[i], vettoreRisultato);
         }
      }
      or priority P, receive(fromClient[2], matr[0][] do {
         / / stesso codice ... per tutti i canali client
      }
    }
  }
  ...
```

In questo caso si usa un comando alternativo con n guardie, una per canale, di pari priorità.

Una versione alternativa prevede l'utilizzo di variabili di tipo canale. Il client dichiara un canale in ingresso e passa tale canale al server, spedendolo sull'unico canale in ingresso del server. Il server, alla ricezione di un messaggio di questo tipo, manda al client sul suo canale in ingresso un canale (in ingresso al server) da utilizzare per la spedizione della matrice ed entra in un ciclo nel quale si attende tutti gli invii relativi alla matrice, processa la matrice e ricava il risultato ed infine invia il risultato sul canale in ingresso al client. Solo

a questo punto si prepara a ricevere nuovamente un messaggio da un client, magari diverso.

Il codice relativo alla spedizione della matrice da parte del clinent, potrebbe essere il seguente:

```
        ADD R0,R0,Ri
        MOV RbaseM, Rbuf
        MOV #1K, RmsgLenSend            ; inizializzazione registro lungh msg
        LD RtabChan,Rchan,RchanSend     ; inizializzazione indirizzo canale
loop:   MOV Rbuf, RbufSend              ; indirizzo del messaggio
        CALL Rsend, Rret
        INC Ri                          ; incremento var iterazione
        IF= Ri, #1K, cont               ; se ho già fatto 1K inivii mi fermo
        ADD #8K, Rbuf                   ; aggiusta indirizzo  buffer
        GOTO loop                       ; e rispedisci
cont:   ...
```

Domanda 3

1. In una cache associativa su insiemi la traduzione dell'indirizzo avviene considerando i primi *k* bit dell'indirizzo (quelli più significativi) come numero dell'insieme, gli ultimi *m* bit come offset e il resto come tag. Il numero dell'insieme identifica in modo univoco un gruppo di *w* linee (o blocchi), ciascuno di 2^m parole. Quale fra le linee sia quella effettivamente indirizzata dipende dal *tag* che viene confrontato con quello ricavato dall'indirizzo. Una cache associativa su insiemi può essere implementata con una memoria di *w * #bit(tag)* bit per parola che contiene, alla posizione *i* tutti i tag dell'insieme *i*. L'accesso alla memoria abilità *k* confrontatori che danno il segnale di abilitazione per una delle *k* memorie che effettivamente contengono le parole delle *k* linee nell'insieme. Lettura e scrittura vengono comandate in queste memorie con i parametri dell'operazione richiesta dall'utente. L'effetto della politica write through si apprezza durante le scritture: ogni scrittura in una linea di un insieme della cache provoca una richiesta di scrittura verso il livello superiore della gerarchia di memoria. Le scritture nel livello successivo sono asincrone, quindi la minor banda del livello di memoria superiore non influenza il tempo richiesto per una scrittura in cache *a patto che* le sequenze di scritture avvengano con una banda minore.

2. Nelle comunicazioni fra processi interni e processi esterni le interruzioni del processo esterno vengono virtualizzate in modo tale da far sì che la corrispondente primitiva (virtuale) di comunicazione sia completata. Nel caso di un processo interno che esegua una receive verso un processo esterno, l'avvenuto completamento della copia del messaggio nel canale di comunicazione (possibilmente in DMA) verrà segnalata con un'interruzione dal dispositivo. Il trattamento di tale interruzione effettuerà tutti le operazioni di sincronizzazione necessarie a risvegliare il destinatario dal suo stato di atteso, e quindi a completare la send virtuale eseguita dal dispositivo. Analogamente, in caso di processo interno che esegua una send verso un processo esterno, l'interruzione da parte del processo esterno verrà gestita come segnalazione dell'avvenuta ricezione del messaggio e quindi potrà portare al rsiveglio del processo mittente, in caso di comunicazione sincrona.