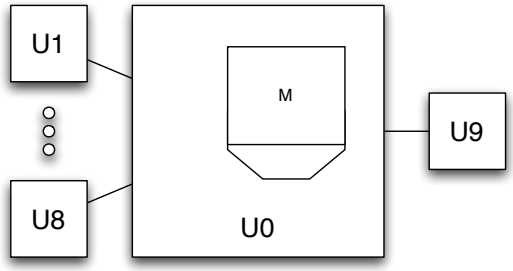
Architettura degli elaboratori A.A. 2007-2008

Terzo appello - 10 giugno 2008

Domanda 1

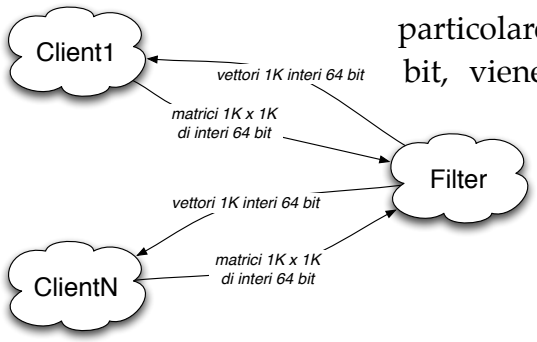
Una unità U_0 è connessa a 9 altre unità (U_1, U_2, ... U_9). Ciascuna delle unità da U_1 ad U_8 invia alla unità U_0 dati di tipo intero a 32 bit, e l'unità U_9 preleva dall'unità U_0 dati dello stesso tipo. L'unità U_0 contiente una memoria di capacità 8K parole e tempo di accesso uguale a $10t_p$. U_0 implementa una coda FIFO di parole: le parole da inserire sono inviate dalle unità U_1-U_8, mentre l'unità U_9 richiede l'estrazione di una parola.



Le estrazioni hanno priorità sulle inserzioni.
Le inserzioni devono essere gestite in modo che nel caso in cui più di una unità (fra le unità U_1-U_8) voglia inserire un valore nella coda FIFO, avviene per primo l'inserimento della parola spedita dall'unità con indice più alto. In altre parole, se all'istante *t* sia U_i che U_{i+k} (k>0) vogliono inserire una parola nella coda, avviene per primo l'inserimento di quella dell'unità U_{i+k} e successivamente l'inserimento di quella dell'unità U_i.
Per l'unità U_0, si descriva lo schema, si fornisca il microprogramma e si forniscano le misure relative al ciclo di clock.

Domanda 2

Si consideri la rete di processi della figura. Il processo Filter accetta dati dai processi Client. I dati sono matrici 1Kx1K di interi a 64 bit. Sulle matrici viene effettuato un particolare calcolo, e il risultato, un vettore da 1K interi a 64 bit, viene restituito al processo Client che ha inviato la matrice. Si dettagli il codice dei processi e si fornisca la compilazione in codice assembler della porzione di codice che effettua la spedizione della matrice nei processi client.



Domanda 3

1. Per una cache associativa su insiemi di tipo write through si descrivano il metodo di accesso (in lettura e in scrittura) e la relativa implementazione.
2. Si dettagli il ruolo delle interruzioni nelle comunicazioni fra processi interni e processi esterni

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.

Bozza di soluzione

Domanda 1

Una coda FIFO implementata con una memoria richiede due puntatori (registri) gestiti modulo la lunghezza della memoria: un puntatore di inserimento (che punta alla prima posizione vuota) e uno di estrazione (che punta alla prima posizione da estrarre). L'inserimento può avvenire solo se la coda non è piena, e in tal caso si fa l'inserzione e si incrementa (modulo la lunghezza della memoria) il puntatore di inserzione (TESTA). L'estrazione può avvenire solo se la coda non è vuota, e in tal caso si preleva e si incrementa (modulo la lunghezza della memoria) il puntatore di estrazione (CODA). Possiamo usare due flag binari per rappresentare la condizone di coda vuota e coda piena (VUOTA e PIENA, da 1 bit ciascuno).

Inizialmente si pone CODA = TESTA = 0, VUOTA=true/1, PIENA=false/0. L'inserzione avviene nel seguente modo:

```
if(!PIENA) {
      MEM[TESTA] = valore;
      TESTA = (TESTA+1)%LEN;
      PIENA = (TESTA == CODA ? true : false);
}
```

mentre l'estrazione avviene in maniera duale:

```
if(!VUOTA) {
      valore = MEM[CODA];
            CODA = (CODA+1) % LEN;
      VUOTA = (CODA==TESTA ? true : false);
}
```

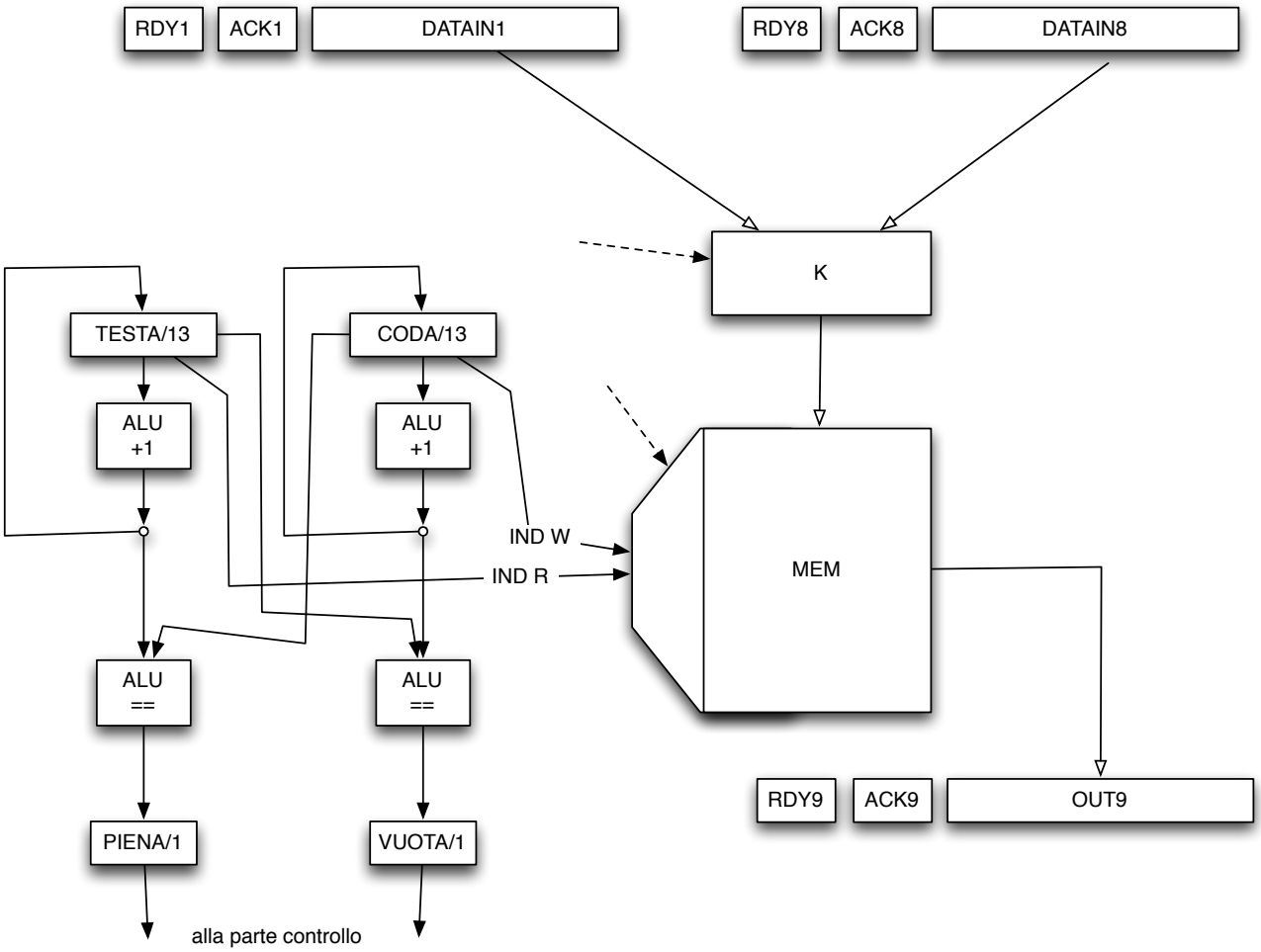
Questo significa che possiamo utilizzare, con una memoria da 8K, due registri da 13bit per TESTA e CODA, che verranno incrementati ignorando il riporto (quindi modulo 8K) e due registri da 1 bit per i flag PIENA e VUOTA.

Il microcodice (senza utilizzare il controllo residuo) potrebbe dunque essere il seguente:

0. (PIENA,VUOTA,RDY9,RDY8,...,RDY1=-01--------) // richiesta prelievo coda non vuota
   MEM[CODA] → OUT9, reset RDY9, set ACK9,
   CODA+1→CODA,(CODA+1==TESTA)→VUOTA, 0
(= -11--------) nop,0 // richiesta di prelievo da coda vuota
(= -001-------) IN8→MEM[TESTA], reset RDY8, set ACK8, // prelievo U8
   TESTA+1 → TESTA, (TESTA+1 == CODA) → PIENA, 0
(= 0-001------) IN7→MEM[TESTA], reset RDY7, set ACK7, // prelievo U7
   TESTA+1 → TESTA, (TESTA+1 == CODA) → PIENA, 0
....
(= -0000000001) IN1→MEM[TESTA], reset RDY1, set ACK1, // prelievo U1

TESTA+1 → TESTA, (TESTA+1 == CODA) → PIENA, 0
(=-1---------) nop, 0 // eventuale richiesta di prelievo da coda vuota o nessun richiesta

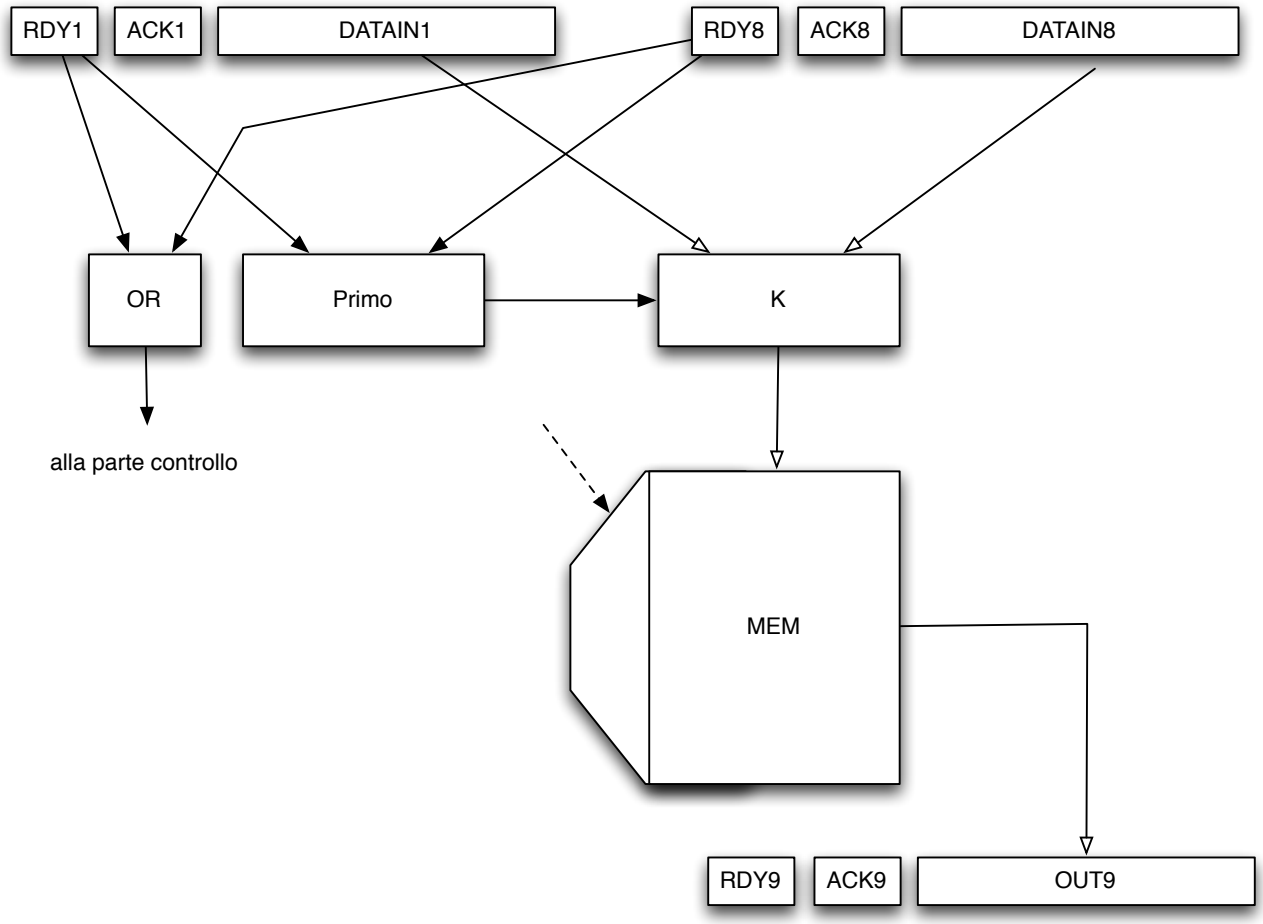
Lo schema relativo (trascurando i β sui registri) è



In questo caso il ciclo di clock può essere calcolato come segue:
$T_{\omega PO}$ = 0 (solo lettura da registri)
$T_{\omega PC}$ = 3 tp (più di otto ingressi)
$T_{\sigma PC}$ = 0 (c'e' uno stato solo ...)
$T_{\sigma PO}$ = 12 tp (K + ALU, le due ALU in cascata per i flag pesano meno)

per un totale di 3tp + 12tp + tp = 16 tp

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 segue (trascuriamo la parte relativa ai registri TESTA, CODA, PIENA e VUOTA, in quanto risulta uguale alla precedente):



si può scrivere il microprogramma usando come variabili di condizionamento

PIENA,VUOTA,RDY9,OR(RDY1,...,RDY8)

riducendo sensibilmente il numero delle variabili di condizionamento. In questo caso, le prima frasi della microistruzione del microprogramma precedente rimarrebbero invariate (prelievo su coda non vuota e prelievo su coda vuota, ma cambierebbero le frasi relative alla inserzione):

0. (PIENA,VUOTA,RDY9,or(RDY8,...,RDY1)=-01-) // richiesta prelievo coda non vuota
MEM[CODA] → OUT9, reset RDY9, set ACK9,
CODA+1→CODA,(CODA+1==TESTA)→VUOTA, 0
(= -11--------) nop,0 // richiesta di prelievo da coda vuota
(= -001) IN[primo()]→MEM[TESTA], reset RDY [primo()], set ACK [primo()],
TESTA+1 → TESTA, (TESTA+1 == CODA) → PIENA, 0
(=-1--) nop, 0 // eventuale richiesta di prelievo da coda vuota o nessun richiesta

In questo caso, abbiamo una variabile di condizianamento risultato di una operazione logica su otto ingressi e quindi

$T_{\omega PO} = tp$
$T_{\omega PC} = 2\ tp$

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.

$T_{\sigma PC} = 0$ (c'e' uno stato solo ...)
$T_{\sigma PO} = 12$ tp

per un totale di tp + 2tp + 12tp + tp = 16 tp (si guadagna sulla PC e si perde sulla parte seriale della PO).

In entrambi i casi, visto che la PC ha un solo stato, l'unità può essere realizzata come una rete sequenziale. La rete sequenziale è quella che ha i β e gli α generati direttamente nella PO a partire dalle variabili di condizionamento.

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. È irrealistico pensare di trasmettere l'intera matrice con un singolo invio, visto le dimensioni), e assumendo di avere una coppia di canali distinti fra ogni client e il server, 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++) {  // invio della matrice a blocchi
      send(toServer, matrice[i][]);
   }
   ...
   receive(fromServer,vettoreRisposa); // attesa della risposta
   ...

Server ::
   channel out toClient[nClient],
   channel in  fromClients[nClients] (nClients);
   ...
   int P = ...;
   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
          }
         or priority P, receive(fromClient[3], matr[0][] do {
              // stesso codice ... per tutti i canali client
          }
         ...

         or priority P,receive(fromClient[nClients],matr[0][] do {
              // stesso codice ... per tutti i canali client
          }

      }
   }
   ...
```

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 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 client, potrebbe essere il seguente:

```
     ADD R0,R0,Ri          ; preparazione indice
     MOV RbaseM, Rbuf      ; preparazione base blocco da spedire
     MOV #1K, RmsgLenSend ; inizializzazione registro lungh msg
     LD RtabChan,Rchan,RchanSend ; iniz. indirizzo canale
loop:MOV Rbuf, RbufSend   ; indirizzo del messaggio
     CALL Rsend, Rret     ; spedizione
     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 risveglio del processo mittente, in caso di comunicazione sincrona.