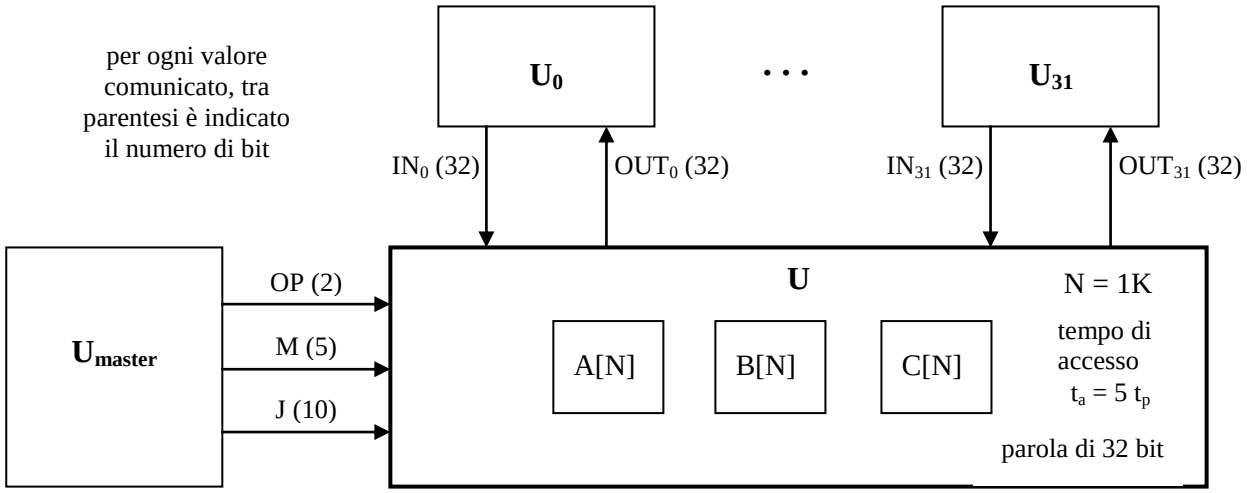
Architettura degli Elaboratori, 2007-08

Prima prova di verifica intermedia

7 novembre 2007

Domanda 1

Una unità di elaborazione U è inserita nel seguente sistema:



U contiene tre componenti logici memoria *A, B, C* con le caratteristiche indicate in figura.

U riceve da U_{master} messaggi del tipo *(OP, M, J)*, dove *OP* è il codice operativo delle operazioni esterne, *M* è un indice di una delle unità $U_0, \ldots, U_{31}$, e *J* è un indirizzo delle memorie. Nei confronti della generica unità U_M, U riceve messaggi IN_M di una parola ed invia messaggi OUT_M di una parola.

Le operazioni esterne di U sono definite come segue:

0. il valore *C[J] + max (A[J], B[J])* è scritto in *C[J]* e inviato all'unità U_M;
1. se $512 \le A[J] < 1024$ allora invia a U_M il valore di *B[J] + C[J]*, altrimenti invia a U_M il valore di *A[J] + C[J]*;
2. $\forall i = 0 .. N - 1$: *C[i]* = numero di volte che in *B* compare un valore minore del valore $IN_M + A[i]$.

Per la progettazione di U è posto il seguente vincolo: *il tempo di elaborazione sia dell'operazione esterna 0 che dell'operazione esterna 1 deve essere uguale ad un ciclo di clock.*

Si dispone di porte logiche con al più 8 ingressi e ritardo di stabilizzazione t_p, e di ALU con ritardo di stabilizzazione $5t_p$. L'impulso di clock ha durata t_p.

a) Inserendo opportune spiegazioni, scrivere il microprogramma di U, esprimere la funzione delle uscite della Parte Operativa, determinare il ciclo di clock in funzione di t_p, e valutare il numero di cicli di clock impiegati dall'operazione esterna 2.

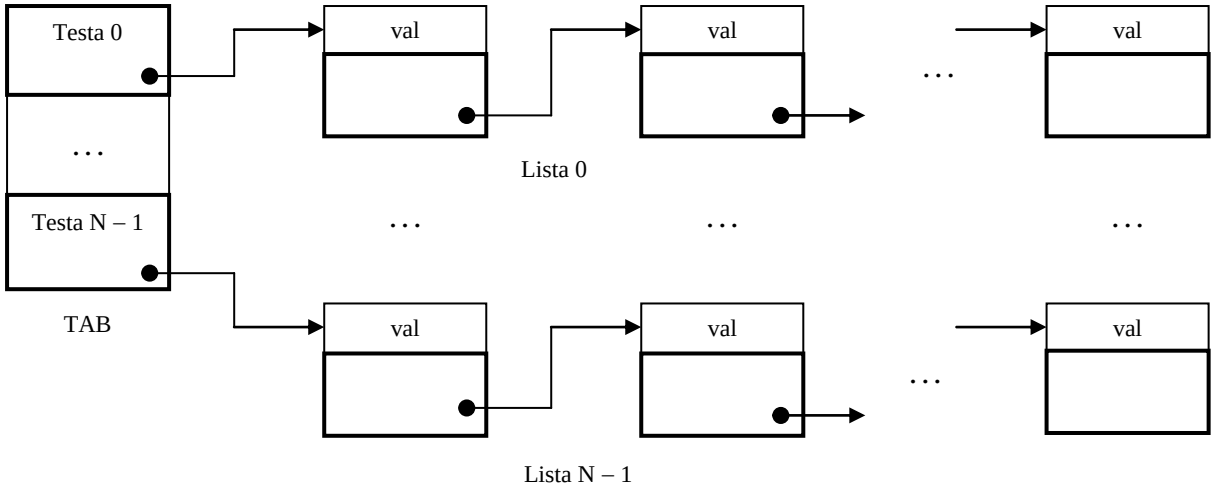
b) Si consideri il caso in cui per U siano definite *solo le operazioni esterne 0 e 1* (di conseguenza, *OP* è di 1 bit e non esistono i collegamenti IN_M). Inserendo opportune spiegazioni, dare una realizzazione di U come *singola* rete sequenziale, esprimendone la funzione delle uscite e la funzione di transizione dello stato interno (possono non essere espresse le funzioni che, nel codominio, si riferiscono agli indicatori di sincronizzazione).

Domanda 2

Un programma opera sulle seguenti strutture dati:

- *N* liste "linkate", ognuna con proprio numero di elementi fisso e noto, dove ogni elemento ha la *parte valore* di tipo intero;
- un array *TAB* di *N* elementi, dove *TAB[i]* contiene la *testa* della lista *i*-esima, cioè il puntatore al primo elemento della lista *i*-esima *e* l'indicazione se tale lista è vuota.

La seguente figura è puramente indicativa:



Per un dato valore intero *X*, il programma restituisce un array *C[N]* di interi tale che:

- $\forall i = 0 .. N - 1$: *C[i]* = numero di volte che *X* compare nella lista *i*-esima. In caso di lista vuota viene restituito *C[i] = 0*.

Inserendo opportune spiegazioni, compilare il programma in assembler D-RISC, dandogli la seguente organizzazione:

- un programma principale,
- una procedura
  - che conta il numero di volte che un dato valore intero compare in una lista "linkata" i cui elementi hanno parte valore di tipo intero,
  - i cui parametri d'ingresso corrispondono al valore intero ed alla lista, ed il cui parametro di uscita corrisponde al numero di volte che il valore compare nella lista.

Soluzione

Domanda 1

a) Scelte significative

1. Per rispettare il vincolo che le operazioni esterne 0 e 1 abbiano tempo di elaborazione di un ciclo di clock, è necessario usare *variabili di condizionamento "complesse"*, rispettivamente:

   segno (A[J] – B[J])

   cond (A[J]) = **if** (A[J][9] = 1) **and** (OR(A[J][10..31]) = 0)

   L'espressione di *cond(A[J])* esprime nel modo più diretto il predicato $512 \le A[J] < 1024$ in base a proprietà elementari dei numeri binari (si consideri che i bit della parola sono identificati in modo crescente a partire dal meno significativo).

   La prima variabile di condizionamento, *fatta salva la sua implementazione corretta* (vedi in seguito), comporta un ritardo di stabilizzazione uguale a

   $$t_a + T_{ALU} = 10\ t_p$$

   la seconda uguale a $3t_p$ (due livelli OR, un livello AND), da cui

   $$T_{\omega PO} = 10\ t_p$$

   A questo punto, *per il microprogramma dell'operazione esterna 2 è inutile cercare ottimizzazioni relative al ciclo dei clock agendo sul ritardo della funzione* ω_{PO}, in quanto $T_{\omega PO}$ è imposto dalle operazioni esterne 0 e 1 (inoltre sul valore di $T_{\sigma PO}$ non si potranno avere differenze significative). Quindi, anche nell'operazione esterna 2, allo scopo di *minimizzare il numero di cicli di clock e quindi il tempo di elaborazione,* verranno usate variabili di condizionamento complesse come funzioni di operazioni effettuate da ALU sul contenuto di memorie (quindi con lo stesso ritardo di $10t_p$ visto sopra);

2. le interfacce con U_0, …, U_{31} sono implementate *in modo parametrico applicando la tecnica del controllo residuo* in funzione del valore M. Ciò significa che esistono due commutatori, controllati da M, aventi in ingresso rispettivamente tutti i 32 indicatori RDYIN e tutti i 32 indicatori ACKOUT, dando luogo alle due variabili di condizionamento

   RDYIN[M] e ACKOUT[M]

   IN[M] è ottenuto anch'esso da un commutatore a 32 ingressi, su parola, controllato da M. Questi commutatori, pur non a 2 livelli di logica, hanno certamente ritardo di stabilizzazione $< 10\ t_p$. I segnali di abitazione alla scrittura per tutti gli indicatori di interfaccia, e per tutti i registri OUT, sono ottenuti da un *selezionatore* a 32 uscite, avente in ingresso una variabile β e controllato da M.

3. l'interprete dell'operazione esterna 2 è espresso da due cicli *for* annidati:

   IN[M] → D;
   for (i = 0; i < N; i++)
   { D + A[i] → E, 0 → S;
   for (h = 0; h < N; h++)
   if (B[h] < E) **then** S + 1 → S;
   S → C[i]
   }

4. I registri temporanei I, H, S dell'operazione esterna 2 sono tutti di 11 bit.

5. Si accetta che U_{master} rimanga bloccato in seguito all'attesa che U comunichi con U_M.

Microprogramma

0. (RDY_{master}, OP_0, OP_1, segno (A[J] – B[J]), cond (A[J]), RDYIN[M], ACKOUT[M] = 0 – – – – – –)
   nop, 0;

   // operazione esterna 0 //

   (= 1 0 0 – – – 0) nop, 0;

   (= 1 0 0 0 – – 1) reset RDY_{master}, set ACK_{master}, set RDYOUT[M], reset ACKOUT[M],
   C[J] + A[J] → C[J], C[J] + A[J] → OUT[M], 0;

   (= 1 0 0 1 – – 1) reset RDY_{master}, set ACK_{master}, set RDYOUT[M], reset ACKOUT[M],
   C[J] + B[J] → C[J], C[J] + B[J] → OUT[M], 0;

   // operazione esterna 1 //

   (= 1 0 1 – – – 0) nop, 0;

   (= 1 0 1 – 0 – 1) reset RDY_{master}, set ACK_{master}, set RDYOUT[M], reset ACKOUT[M],
   C[J] + A[J] → OUT[M], 0;

   (= 1 0 1 – 1 – 1) reset RDY_{master}, set ACK_{master}, set RDYOUT[M], reset ACKOUT[M],
   C[J] + B[J] → OUT[M], 0;

   // operazione esterna 2 //

   (= 1 1 – – – 0 –) nop, 0;

   (= 1 1 – – – 1 –) reset RDY_{master}, set ACK_{master}, reset RDYIN[M], set ACKIN[M],
   IN[M] → D, 0 → I, 0 → H, 1;

1. D + $A[I_m]$ → E, 0 → S, 2

2. (I_0, H_0, segno ($B[H_m]$ – E) = 0 0 0) H + 1 → H, 2;

   (= 0 0 1) S + 1 → S, H + 1 → H, 2;

   (= 0 1 –) I + 1 → I, 0 → H, S → $C[I_m]$, 1;

   (= 1 – –) "nop", 0 // *nop* eliminabile ma inessenziale per le prestazioni, avendo impatto $1/N^2$ cicli di clock //

Relativamente alle operazioni esterne 0 e 1, si può notare che *segno (...)* e *cond (...)* potrebbero, alternativamente, essere usate come variabili di controllo residuo per comandare commutatori all'ingresso di ALU (o simili). La soluzione è equivalente a quella data e, visto il basso numero stati d'ingresso distinti della PC, non comporta un miglioramento apprezzabile della complessità di progettazione.

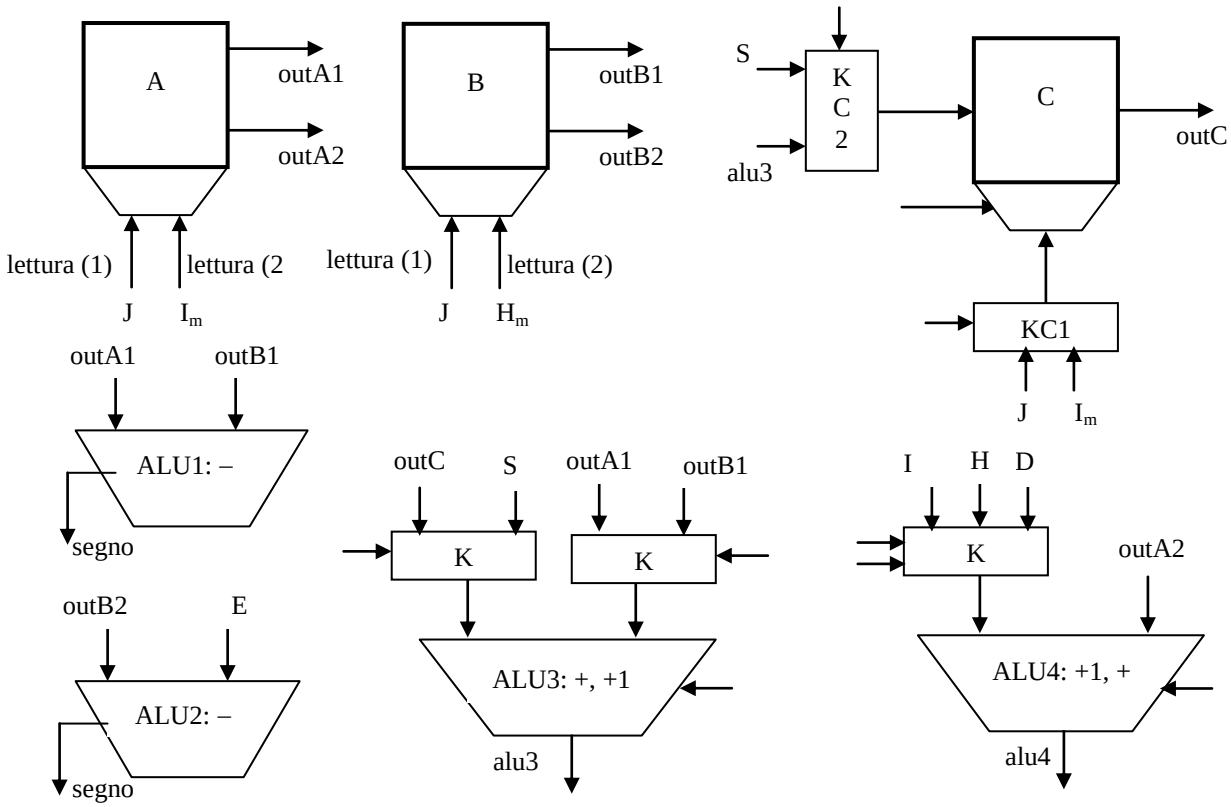
Per l'operazione esterna 2, la microistruzione 1 ha *effetto trascurabile sulle prestazioni* essendo eseguita solo nel loop più esterno; essa potrebbe essere fusa con la 0 e con la 2, con una diversa inizializzazione degli indici, ma questo avrebbe avuto come conseguenza la necessità di indirizzare la memoria A anche con la costante 0, aumentandone inutilmente il numero di indirizzamenti indipendenti.

Risorse più significative della PO agli effetti della verifica della correttezza e delle prestazioni

La figura seguente mostra un frammento di struttura della PO contenente solo le risorse che interessano, sia per la verifica della condizione necessaria per la correttezza, che per ricavare il massimo ritardo della funzione σ_{PO}.

Le altre risorse hanno una struttura ovvia o facilmente derivabile dalle risorse mostrate.

Le interfacce parametriche sono state descritte a parole in precedenza.



Funzione delle uscite della PO

Allo scopo di soddisfare la condizione necessaria per la correttezza (PO automa di Moore), le variabili di condizionamento *segno (A[J] – B[J]), cond (A[J])* e *segno (B[H_m] – E)* sono implementate usando ALU dedicate (ALU1, ALU2 per la prima e la terza) e memorie con indirizzamento multiplo.

La funzione delle uscite si ottiene *dall'analisi* della struttura della PO, ed è espressa come segue:

variabili di condizionamento:

$x_0 = RDY_{master}$

$x_1 = OP_0$

$x_2 = OP_1$

x_3 = segno (A[J] – B[J])

x_4 = cond (A[J]) = **if** (A[J][9] = 1) **and** (OR(A[J][10..31]) = 0)

$x_5 = RDYIN[M]$

$x_6 = ACKOUT[M]$

$x_7 = I_0$

$x_8 = H_0$

x_9 = segno (B[H_m] – E)

uscite esterne:

$\forall i = 0 .. 31:$ $out_i = OUT[i]$

senza considerare gli indicatori di sincronizzazione delle interfacce

Ciclo di clock

Il massimo ritardo della funzione delle uscite della PO è già stato valutato:

$$T_{\omega PO} = 10\ t_p$$

Per quanto riguarda il ritardo di stabilizzazione della σ_{PO}, dominano tutte le operazioni elementari che comportano accessi alle memorie e operazioni nella ALU3, rispetto a quelle che operano su I, H, S.

La scrittura nella memoria C è completamente sovrapposta alle letture dalle memoria C ed A (o C e B) e al calcolo nella ALU3 (in quanto il selezionatore in scrittura agisce solo sul segnale di abilitazione alla scrittura, e non sul dato che invece è direttamente in ingresso a tutte le celle). Inoltre, sia l'indirizzo che il dato della memoria C devono attraversare commutatori a due livelli di logica (KC1, KC2).

Per la scrittura in OUT[M] si ha, rispetto alla scrittura in C, un commutatore in meno da attraversare, mentre la selezione del segnale di abilitazione alla scrittura è, anche in questo caso, completamente sovrapposta.

In conclusione, il ritardo massimo è quello delle operazioni elementari

$$C[J] + B[J] \rightarrow C[J] \qquad \text{e} \qquad C[J] + A[J] \rightarrow C[J]$$

da cui:

$$T_{\sigma PO} = t_a + T_{ALU} + 3\,T_{K\text{-}2livelli} = 16\,t_p$$

Per quanto riguarda le funzioni della PC, premesso che lo stato interno è codificato da due variabili (y_0, y_1), l'analisi della struttura del microprogramma mostra che il massimo numero di variabili specificate nei termini AND è uguale a 7, quindi tali termini sono implementabili su un singolo livello di logica. Inoltre, il numero di frasi nelle quali le variabili Y_0, Y_1 e quelle di tipo β assumono il valore "1" è limitato superiormente da 8, quindi anche i termini OR sono implementabili su un singolo livello di logica. Dunque:

$$T_{\omega PC} = T_{\sigma PC} = 2\,t_p$$

Il ciclo di clock vale:

$$\tau = T_{\omega PO} + T_{\omega PC} + T_{\sigma PO} + \delta = 29\,t_p$$

Numero di cicli di clock dell'operazione esterna 2

La microistruzione 2 (loop più interno) è eseguita N^2 volte, mentre la 1 (inizio del loop più esterno) solo N volte, quindi, dato l'elevato valore di N:

$$T_2 \sim N^2\,\tau$$

b) Se sono definite solo le operazioni esterne 0 e 1, U è realizzabile come singola rete sequenziale in quanto, essendo il microprogramma costituito da una sola microistruzione, la Parte Controllo si riduce ad una rete combinatoria che viene ad essere "fusa" con le funzioni ω_{PO} e σ_{PO} per dar luogo alle funzioni ω e σ dell'unica rete.

Microprogramma

0. (RDY_{master}, OP, segno (A[J] – B[J]), cond (A[J]), ACKOUT[M] = 0 – – – –) nop, 0;

   // operazione esterna 0 //

   (= 1 0 – – 0) nop, 0;

   (= 1 0 0 – 1) reset RDY_{master}, set ACK_{master}, set RDYOUT[M], reset ACKOUT[M], C[J] + A[J] → C[J], C[J] + A[J] → OUT[M], 0;

   (= 1 0 1 – 1) reset RDY_{master}, set ACK_{master}, set RDYOUT[M], reset ACKOUT[M], C[J] + B[J] → C[J], C[J] + B[J] → OUT[M], 0;

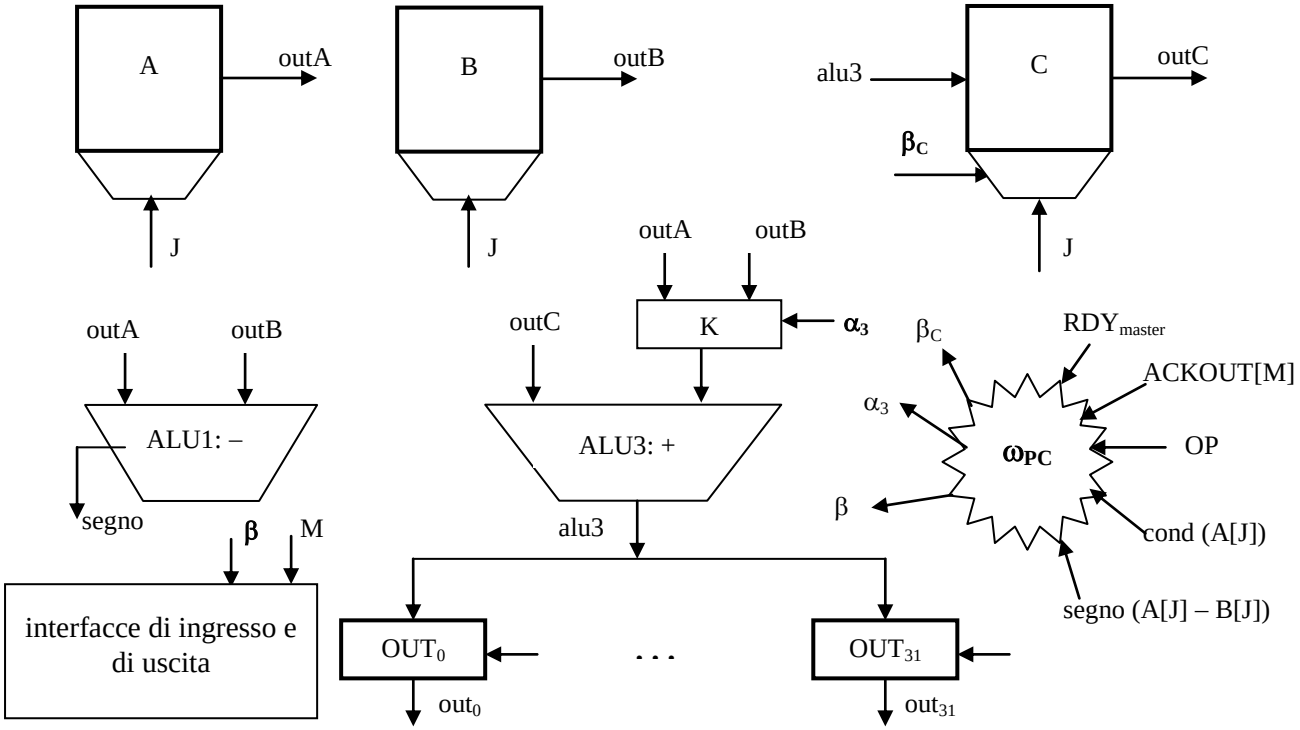
   // operazione esterna 1 //

   (= 1 1 – – 0) nop, 0;

   (= 1 1 – 0 1) reset RDY_{master}, set ACK_{master}, set RDYOUT[M], reset ACKOUT[M], C[J] + A[J] → OUT[M], 0;

   (= 1 1 – 1 1) reset RDY_{master}, set ACK_{master}, set RDYOUT[M], reset ACKOUT[M], C[J] + B[J] → OUT[M], 0;

Parte più significativa della struttura della rete sequenziale secondo il modello di Moore



La struttura è ottenuta direttamente dal microprogramma secondo l'usuale metodo formale. Dall'*analisi* della struttura si ricavano le funzioni delle uscite e di transizione dello stato interno:

ω: $\forall\, i = 0 \,..\, 31:$ $out_i = OUT[i]$

σ:
- $\forall\, p = 0 \,..\, 1023:$ $in_{C[p]}$ = **when** ($\beta_C = 1$) **do if** (p = J) **then** alu3
- $\forall\, i = 0 \,..\, 31:$ $in_{OUT[i]}$ = **when** ($\beta = 1$) **do if** (i = M) **then** alu3
- in_{OP} = **when** clock **do** op // ingressi da U_{master} indicati con *p, m, j* //
- in_M = **when** clock **do** m
- in_J = **when** clock **do** j

dove

alu3 = C[J] + **if not** α_3 **then** A[J] **else** B[J]

e le espressioni booleane:

$$\beta_C = RDY_{master}\ \overline{OP}\ ACKOUT[M]$$

$$\beta = RDY_{master}\ ACKOUT[M]$$

$$\alpha_3 = RDY_{master}\ \overline{OP}\ segno\ ACKOUT[M] + RDY_{master}\ OP\ cond\ ACKOUT[M]$$

definiscono la funzione ω_{PC}, che è parte integrante della σ.

Anche se non richiesto, mostriamo anche come potrebbe esser realizzata la rete sequenziale equivalente secondo il modello di **Mealy**. Essa può essere facilmente ottenuta per trasformazione dalla rete di Moore nel caso particolare, ma significativo, del modello "Moore anticipato". Per far questo:

- vengono lasciati inalterati l'insieme degli stati interni e la funzione di transizione dello stato interno,
- viene derivata la funzione delle uscite dalla funzione di transizione dello stato interno della rete di Moore per quanto riguarda i soli registri OUT:

ω: $\forall\, i = 0 \,..\, 31:$ out_i = **if** ((RDY_{master} = 1) **and** (ACKOUT[M] = 1) **and** (i = M))
then alu3 **else** OUT[i]

σ:
- $\forall\, p = 0 \,..\, 1023:$ $in_{C[p]}$ = **when** (β_C = 1) **do if** (p = J) **then** alu3
- $\forall\, i = 0 \,..\, 31:$ $in_{OUT[i]}$ = **when** (β = 1) **do if** (i = M) **then** alu3
- in_{OP} = **when** clock **do** op // ingressi da U_{master} indicati con *p, m, j* //
- in_M = **when** clock **do** m
- in_J = **when** clock **do** j

Nella funzione ω l'espressione

if ((RDY_{master} = 1) **and** (ACKOUT[M] = 1) **and** (i = M))

esprime lo stesso valore della condizione **when** (β = 1) **if** (i = M) presente nella funzione σ; mentre nella funzione σ interessa evidenziare che, se tale condizione non è verificata, non ha luogo transizione di stato, nella funzione ω viene esplicitata anche l'azione in seguito alla condizione non verificata (ramo *else*).

L'implementazione della funzione σ è quella mostrata nella figura precedente della rete di Moore.

L'implementazione della funzione ω si ricava formalmente dalla definizione della funzione stessa: si tratta di 32 commutatori, l'*i*-esimo dei quali ha in ingresso *alu3* e *OUT[i]* ed è controllato dall'*i*-esima uscita del selezionatore, cioè dalla stessa variabile che abilita la scrittura in *OUT[i]*, vera per (RDY_{master} = 1) **and** (ACKOUT[M] = 1) **and** (i = M) .

Domanda 2

Il programma principale scandisce tutto il vettore mediante un *for*: al passo *i*-esimo, chiama la procedura passando come parametro il puntatore al primo elemento della lista *i*-esima, se questa non è vuota, altrimenti pone *C[i] = 0*. Al ritorno da procedura assegna il risultato a *C[i]*.

Ogni elemento di lista consta di una *parte valore*, costituita da una parola, e da una *parte puntatore*, costituita da una coppia di parole, esattamente come ogni elemento dell'array TAB: la prima parola contiene l'indicatore di fine lista, la seconda l'eventuale indirizzo logico dell'elemento successivo.

La procedura adotta uno schema *do while* con guardia associata alla condizione di fine lista.

Il compilatore inizializza i registri *Rtab*, *Ri* (indice del for del programma principale), *RN*, *Rproc*, *RX*, quest'ultimo utilizzato direttamente come parametro costante della procedura.

Il puntatore viene passato alla procedura attraverso il registro generale *Rpun*, non inizializzato.

Vengono inoltre allocati i registri, non inizializzati, *Rret*, *Rcount* (risultato della procedura), *Rval* (parte valore dell'elemento di lista), *Rfine1* (indicatore di fine lista usato dal programma principale), *Rfine2* (indicatore di fine lista usato dalla procedura).

Compilazione del programma principale

```
LOOP1:  LOAD   Rtab, 0, Rfine1
        IF ≠ 0 Rfine1, THEN1
        LOAD   Rtab, 1, Rpun
        CALL   Rproc, Rret
        STORE  RC, Ri, Rcount
        GOTO   CONT1
THEN1:  STORE  RC, Ri, 0
CONT1:  ADD    Rtab, 2, Rtab
        INCR   Ri
        IF <   Ri, RN, LOOP1
        END
```

compilazione *if then else*

compilazione *for* per la scansione di TAB

scansione dell'array modificando la base e utilizzando, invece, l'indice all'interno di ogni elemento dell'array

Compilazione della procedura

```
        CLEAR  Rcount
LOOP2:  LOAD   Rpun, 0, Rval
        IF ≠   RX, Rval, CONT2
        INCR   Rcount
CONT2:  LOAD   Rpun, 1, Rfine2
        LOAD   Rpun, 2, Rpun
        IF = 0 Rfine2, LOOP2
        GOTO   Rret
```

compilazione *if then*

compilazione *do while* per la scansione della lista

Si osservi che, per indirizzare l'array TAB, i cui elementi sono di due parole, non si è usato l'indice *i* (fatto variare da *0* a *N – 1*, come indica il programma sorgente), bensì si è incrementata la base (Rtab) ad ogni passo, utilizzando invece *i* normalmente per il controllo del *for* e per indicizzare l'array C.

Nella memoria virtuale del processo il compilatore inizializza l'array TAB, le *N* liste, ovviamente tutto il codice del programma principale e della procedura, la parte del PCB relativo all'immagine di IC e dei registri generali inizializzati, ed alcune informazioni di servizio collegate al processo; alloca, ma non inizializza, l'array C e la Tabella di Rilocazione del processo.