Architettura degli Elaboratori, 2006-07

Prima Esercitazione di Verifica Intermedia

(Versione aggiornata il 3 Novembre 2006)

Consegna: lezione di mercoledì 25 ottobre ottobre, ore 11

L'elaborato, da presentare in una forma leggibile agevolmente, deve contenere spiegazioni chiare ed esaurienti, utilizzando la corretta terminologia ed i concetti del corso.
Insieme a nome e cognome indicare l'anno di corso di ogni studente.

Domanda 1

a) Realizzare una rete combinatoria avente quattro variabili booleane di ingresso *a, b, x, y* e due variabili booleane di uscita *z, w*. La funzione delle uscite è così definita: se $a = b$, allora $z = x$ e $w = x \wedge y$; se $a \neq b$, allora $z = y$ e $w = x \vee y$.

Dare due realizzazioni, una ricavata dalla tabella di verità, l'altra direttamente dalla definizione in forma algoritmica. In entrambe le soluzioni ricavare il tempo di stabilizzazione.

b) Realizzare una rete sequenziale sincrona avente tre variabili booleane di ingresso *a, b, x,* una variabile booleana di uscita *z,* ed una variabile dello stato interno: quest'ultima sia indicata con *y* per lo stato presente e con *Y* per lo stato successivo. La funzione delle uscite e di transizione dello stato interno sono definite come segue: se $a = b$, allora $z = x$ e $Y = x \wedge y$; se $a \neq b$, allora $z = y$ e $Y = x \vee y$.

Dare due realizzazioni come indicato al punto *a)* e, per entrambe, ricavare il ciclo di clock della rete.

Sia per il punto *a)* che per il punto *b)* è noto il ritardo t_p di una porta logica. Al punto b), la durata dell'impulso di clock è uguale a t_p.

Domanda 2

Una unità di elaborazione U riceve dall'unità U_a messaggi (OP (2 bit), J (2 bit), X (32 bit), Y (32 bit)) e invia all'unità U_b messaggi (OUT1 (32 bit)), all'unità U_c messaggi (OUT2 (1 bit)), all'unità U_d messaggi (OUT3 (8 bit)).

U contiene una memoria A di capacità 32K parole, con parole di 32 bit. *Fisicamente* A è realizzata mediante 64 memorie ognuna di capacità 512 parole. *Logicamente* A è vista come composta da 256 *blocchi*, ognuno di 128 parole aventi indirizzi consecutivi.

Le operazioni esterne sono le seguenti:

i. invia su OUT1 il valore di X + Y[J] + Y[J+1];

ii. invia sull'uscita OUT2 un valore booleano vero se e solo se il valore di X è una potenza di 2;

iii. considerato il blocco di A identificato univocamente dal valore di Y[J], calcola il numero C di parole di tale blocco aventi valore minore o uguale a X, e invia il valore di C sull'uscita OUT3.

È noto il ritardo t_p di una porta logica con al massimo 8 ingressi. Una ALU ha ritardo uguale a $5t_p$. La durata dell'impulso di clock è uguale a t_p. Una memoria di 512 parole ha tempo di accesso uguale a $3t_p$.

a) Scrivere il microprogramma di U, spiegando le scelte più significative. Per l'operazione esterna *ii*, utilizzare un algoritmo di complessità lineare nel numero di bit della parola e *senza* usare la funzione di shift.

b) Per la rete sequenziale Parte Operativa: mostrare lo schema, ricavare la funzione delle uscite, ricavare la funzione di transizione dello stato interno relativamente alla parte di stato rappresentata dal registro che nella terza operazione esterna funge da contatore di passi; spiegare le risposte.

c) Per la rete sequenziale Parte Controllo: ricavare e realizzare la funzione di transizione dello stato interno, ricavare l'espressione logica della variabile di controllo che abilita la scrittura nel registro contatore di passi di cui al punto *b)*; spiegare le risposte.

d) Ricavare il valore del ciclo di clock in funzione di t_p, spiegando chiaramente come è stato ottenuto.

e) Ricavare il tempo medio di elaborazione delle due operazioni esterne *i*, *iii*.

Domanda 3

Si consideri una unità di elaborazione che, rispetto ad U della Domanda 2, ha solo le prime due operazioni esterne; le interfacce vanno opportunamente modificate di conseguenza, e, ovviamente, non esiste la memoria A.

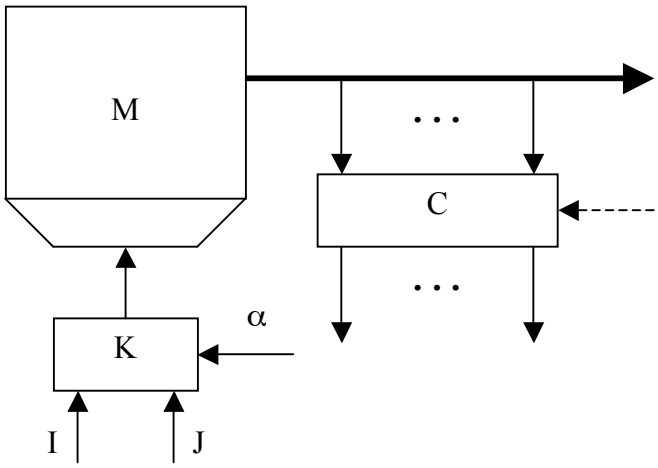
a) Spiegare se la realizzazione di tale unità può concettualmente essere sostituita da una rete combinatoria.

b) Nel caso sia concettualmente fattibile, realizzare tale rete combinatoria, e valutarne il ritardo di stabilizzazione facendo le stesse ipotesi della Domanda 1 sui ritardi di porte logiche e reti.

c) Spiegare quali vantaggi e svantaggi ha una realizzazione come rete combinatoria.

Domanda 4

Si supponga che la seguente struttura sia contenuta nella Parte Operativa di una certa unità di elaborazione.



C e le parole della memoria M sono di 32 bit, α è una variabile di controllo. Il microprogramma dell'unità contiene una microistruzione del tipo:

$$i.\ (C_{27} = 0)\ \ldots.;\ (C_{27} = 1)\ \ldots.$$

Dire se la seguente affermazione è vera, falsa oppure vera sotto determinate condizioni, e spiegare esaurientemente la risposta:

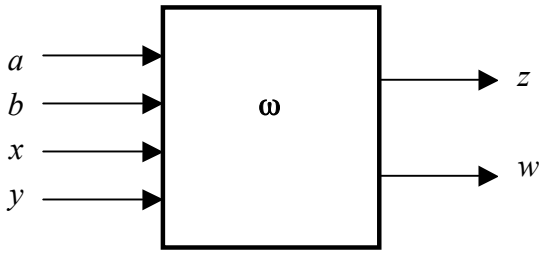
"poiché

$$C_{27} = \textbf{if } \alpha \textbf{ then } M[I]_{27} \textbf{ else } M[J]_{27}$$

la Parte Operativa di questa unità non soddisfa la condizione necessaria per la correttezza".

Domanda 1

a) Rete combinatoria



La rete è completamente definita dalle specifiche, che possono essere espresse in un formalismo algoritmico:

$z =$ **if** $(a = b)$ **then** x **else** y, $\quad w =$ **if** $(a = b)$ **then** $x \wedge y$ **else** $x \vee y$

Soluzione 1 – Espressioni logiche ricavate dalla tabella di verità

Dalla definizione si ricava la tabella di verità:

a	b	x	y	z	w
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	1	0
0	0	1	1	1	1
0	1	0	0	0	0
0	1	0	1	1	1
0	1	1	0	0	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	1	1	1
1	0	1	0	0	1
1	0	1	1	1	1
1	1	0	0	0	0
1	1	0	1	0	0
1	1	1	0	1	0
1	1	1	1	1	1

da cui si ottengono le espressioni logiche in forma SP delle variabili di uscita (ricavate indipendentemente l'una dall'altra), introducendo subito alcune semplificazioni

- in seguito all'applicazione dei teoremi della complementazione ($A + \overline{A} = 1$) e dell'unione ($A\ 1 = A$),
- osservando che $\overline{A}B + A\overline{B} + AB = A + B$:

$$z = \overline{a}\,\overline{b}x\overline{y} + \overline{a}\,\overline{b}xy + \overline{a}\,b\overline{x}y + \overline{a}\,bxy + a\overline{b}\,\overline{x}y + a\,\overline{b}xy + a\,bx\overline{y} + a\,bxy =$$
$$= \overline{a}\,\overline{b}x(\overline{y} + y) + \overline{a}\,by\,(\overline{x} + x) + a\,\overline{b}y\,(\overline{x} + x) + a\,b\,x\,(\overline{y} + y) =$$
$$= \overline{a}\,\overline{b}x + \overline{a}\,by + a\,\overline{b}\,y + a\,b\,x$$

$$w = \bar{a}\,\bar{b}xy + \bar{a}\,b\bar{x}\,y + \bar{a}\,bx\bar{y} + \bar{a}\,bxy + a\,\bar{b}\,\bar{x}\,y + a\,\bar{b}x\bar{y} + a\,\bar{b}xy + a\,b\,x\,y =$$
$$= \bar{a}\,\bar{b}\,x\,y + \bar{a}\,b\,(\bar{x}y + x\bar{y} + xy) + a\,\bar{b}\,(\bar{x}y + x\bar{y} + xy) + a\,b\,x\,y =$$
$$= \bar{a}\,\bar{b}xy + \bar{a}\,b\,(x + y) + a\,\bar{b}\,(x + y) + a\,b\,x\,y =$$
$$= \bar{a}\,\bar{b}\,x\,y + \bar{a}\,b\,x + \bar{a}\,b\,y + a\,\bar{b}\,x + a\,\bar{b}\,y + a\,b\,x\,y$$

L'implementazione in termini di porte AND, OR, NOT è immediata: si tratta di una rete *a due livelli di logica*, con un primo livello di 10 porte AND in parallelo (4 per *z*, 6 per *w*) ed un secondo livello di 2 porte OR in parallelo (una per *z*, una per *w*). Tutte le porte hanno un basso numero di ingressi. Il ritardo di stabilizzazione è quindi:

$$T_\omega = 2\,t_p$$

Si osservi che l'applicazione di altre proprietà dell'algebra di Boole, come la proprietà distributiva, introdurrebbe dei ritardi di stabilizzazione addizionali. Invece, l'implementazione SP mostrata è caratterizzata dal minimo ritardo di stabilizzazione.

In effetti, applicando la proprietà distributiva si otterrebbe lo schema della Soluzione 2.

Soluzione 2 – Schema della rete ricavato direttamente dalla descrizione algoritmica usando componenti logici standard

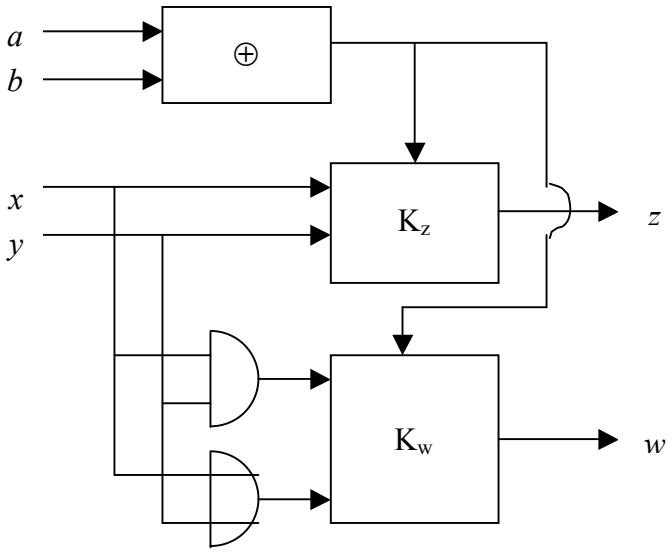
Essendo la descrizione in formalismo algoritmico molto semplice, possiamo, da essa, ricavare direttamente l'implementazione della rete in termini di *componenti logici standard:* nel nostro caso, commutatori, confrontatori, porte AND, OR.

Consideriamo la definizione algoritmica della variabile di uscita *z*:

$$z = \textbf{if}\ (a = b)\ \textbf{then}\ x\ \textbf{else}\ y$$

La presenza di un costrutto *if then else* sta a significare che esiste un *commutatore* a due ingressi; questi assumono il valore delle funzioni presenti nei rami *then* ed *else* (rispettivamente, *x* e *y*); la variabile di controllo assume il valore della funzione predicato; nel nostro caso, il predicato è implementato da un confrontatore con ingressi *a* e *b*. Analogamente si ragiona per la variabile *w*.

La rete combinatoria è quindi:



Questa soluzione mostra come, spesso, sia possibile semplificare notevolmente la sintesi di una rete combinatoria, sintesi che, partendo dalla tabella di verità, è un procedimento di *complessità esponenziale* nel

numero degli ingressi della rete. *Quando fattibile*, il procedimento che consiste nel partire dalla specifica (a parole o in un formalismo algoritmico) è, invece, di complessità *lineare* nel numero delle funzioni (rami *then, else*, o in generale rami di un *case*, e predicato): questa ridotta complessità si può avere a condizione che si abbia *una conoscenza a priori delle reti combinatorie corrispondenti a tali funzioni*, tipicamente corrispondenti a componenti *standard* (ALU, K, ⊕, AND, OR, NOT).

Per contro, il *ritardo di stabilizzazione* di una soluzione del genere è spesso maggiore di quello ottenuto ricavando le espressioni logiche SP (nel caso limite più favorevole, è uguale). In questo esempio, siamo appunto in un caso con ritardo maggiore. Infatti, sia il confrontatore che il commutatore sono reti a due livelli di logica; le porte AND, OR isolate si stabilizzano contemporaneamente al (primo livello di logica del) confrontatore; i due commutatori si stabilizzano contemporaneamente tra loro ma *dopo* che si è completamente stabilizzato il confrontatore. Di conseguenza, si tratta di una rete a 4 livelli di logica, per cui il ritardo di stabilizzazione è ora:

$$T_\omega = 4t_p$$

Quello della soluzione 2 è il tipico approccio che, nella progettazione di unità di elaborazione a livello firmware, viene adottato nella sintesi della Parte Operativa.

Soluzione 3 – Espressioni logiche ricavate dalla definizione algoritmica

Lo schema della Soluzione 2 può talvolta essere ottimizzato arrivando fino a ricavare le espressioni logiche in forma SP per una rete a due livelli di logica. Ciò è possibile nei casi in cui sia nota l'espressione logica in forma SP dei componenti usati: questo è nostro caso, in quanto si conoscono l'espressione logica in forma SP del confrontatore:

$$a \oplus b = \bar{a}\,b + a\,\bar{b} \qquad , \qquad \overline{a \oplus b} = a\,b + \bar{a}\,\bar{b}$$

e del commutatore:

$$z = \bar{\alpha}\,x + \alpha\,\bar{y}$$

Sviluppando le espressioni logiche di z e w della Soluzione 2:

$$z = \overline{(a \oplus b)}\,x + (a \oplus b)\,y$$

$$w = \overline{(a \oplus b)}\,(x\,y) + (a \oplus b)\,(x + y)$$

si ottiene:

$$z = (a\,b + \bar{a}\,\bar{b})x + (\bar{a}\,b + a\,\bar{b})\,y = a\,b\,x + \bar{a}\,\bar{b}\,x + \bar{a}\,b\,y + a\,\bar{b}\,y$$

$$w = (a\,b + \bar{a}\,\bar{b})(x\,y) + (\bar{a}\,b + a\,\bar{b})(x + y) = a\,b\,x\,y + \bar{a}\,\bar{b}\,x\,y + \bar{a}\,bx + \bar{a}\,by + a\,\bar{b}\,x + a\,\bar{b}y$$

che coincidono con le espressioni della Soluzione 1 (in generale, sarebbe stato sufficiente ottenere espressioni SP equivalenti).

b) Rete sequenziale

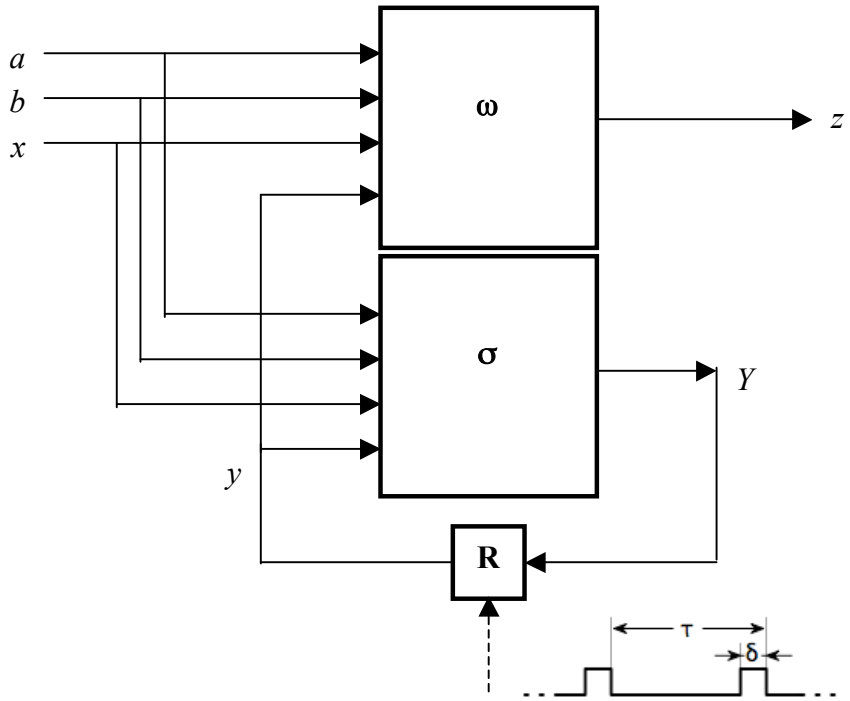
L'automa è completamente definito dall'insieme delle variabili d'ingresso *(a, b, x),* dall'insieme delle variabili di uscita *(z),* dall'insieme delle variabili dello stato interno (presente y, successivo Y), dalla **funzione delle uscite** *ad ogni istante della sequenza temporale*:

$z = \omega\,((a, b, x), y))$: $z =$ **if** $(a = b)$ **then** x **else** y

(si tratta di un *automa di Mealy*), e dalla **funzione di transizione dello stato interno** *ad ogni istante della sequenza temporale*:

$Y = \sigma\,((a, b, x), y))$: $Y =$ **if** $(a = b)$ **then** $x \wedge y$ **else** $x \vee y$La realizzazione sincrona (LLC) fa uso, nella "richiusura" dello stato interno, di un registro impulsato R (un bit in questo caso) allo scopo di

mantenere stabile il valore dello stato interno presente fino alla prossima (possibile) variazione dello stato d'ingresso.



Visto che R è standard, *la realizzazione della rete sequenziale si riduce alla realizzazione delle due reti combinatorie corrispondenti alle funzioni* ω e σ.

Sostituendo *Y* a *w*, si tratta della stessa realizzazione vista nella parte *a)*.

Il ciclo di clock τ della rete sequenziale è dato dal ritardo di stabilizzazione della "più lenta" rete tra ω e σ (nel nostro caso, sono uguali) più la durata dell'impulso. Quindi, essendo l'impulso di clock di durata t_p, nella realizzazione con ritardo di stabilizzazione minore (*Soluzioni 1 e 3*) si ha:

$$T_\omega = T_\sigma = 2\ t_p \qquad \tau = 3\ t_p$$

Nell'altra realizzazione (*Soluzione 2*):

$$T_\omega = T_\sigma = 4\ t_p \qquad \tau = 5\ t_p$$

Domanda 2

Verranno fornite soluzioni alternative che dipendono dalla seguente scelta di progetto:

- *Soluzione 1*: cercare di minimizzare il numero medio di cicli di clock,

oppure

- *Soluzione 2*: cercare di minimizzare il ciclo di clock.

In generale, nessuna delle due soluzioni è ottima a priori, essendo NP-arduo il problema di trovare l'ottimo del tempo medio di elaborazione. Tra l'altro, la soluzione di tale problema dipende anche dalla conoscenza delle probabilità delle operazioni esterne, che nel nostro caso non sono note.

Spesso si può trovare una soluzione capace di minimizzare sia il ciclo di clock che numero medio di cicli di clock, e quindi che sia ottima in assoluto. Per le ragioni di complessità sopra ricordate, è almeno richiesto che siano individuate "buone soluzioni" del tipo 1 e 2.

Discuteremo anche alcune varianti alle Soluzioni 1 e 2, una delle quali è quella ottima.

a) Scelte progettuali e microprogramma

L'unità U ha interfacce a transizione di livello con le unità U_a, U_b, U_c, U_d. Gli indicatori di interfaccia saranno indicati con l'indice della rispettiva unità; inoltre, useremo i suffissi IN e OUT a seconda dei casi. Per uniformità indicheremo OUT1 con OUT_b , OUT2 con OUT_c , OUT3 con OUT_d .

Poiché servirà completare alcune parole con gruppi di zeri nei bit più/meno significativi, useremo la notazione

cost_zero(m)

per indicare una costante di *m* zeri "cablata" nella PO da concatenare ad uscite di registri.

Seguono le **spiegazioni** significative per la scrittura del microprogramma delle tre operazioni esterne. Ulteriori spiegazioni saranno date nelle singole soluzioni.

Interprete dell'operazione esterna i)

La notazione Y[J] indica il byte J-esimo della parola contenuta in Y (J è di 2 bit). Oltre che di tipo parola, Y può infatti essere visto come un *array di 4 byte*: il valore Y[J] è quindi ottenuto con la tecnica del *controllo residuo*, mediante un commutatore a 4 ingressi (ognuno di tipo byte) con variabile di controllo J.

Per essere sommato ad una parola a 32 bit come X, il valore Y[J] deve essere rappresentato su parola intera completandolo con 24 "0" in testa.

La somma di tre valori, richiesta dell'operazione esterna, non può essere ottenuta in modo primitivo con una ALU. Le soluzioni sono due:

- effettuare la somma in due cicli di clock, con l'obiettivo di minimizzare il ciclo di clock (relativamente alla σ_{PO}),
- effettuare la somma in un solo ciclo di clock, usando due ALU in cascata, con l'obiettivo di minimizzare il numero di cicli di clock.

Interprete dell'operazione esterna ii)

Dalle specifiche deriva che occorre testare singolarmente i bit della parola X, in un certo ordine, finché non si incontra un "1"; il numero è una potenza di due se non si incontrano altri "1" nella scansione successiva. Useremo un registro P di un bit, inizializzato a 0; affinché X sia una potenza di 2 occorre che P venga messo a 1 (incontrando un "1" per la prima volta) e non rimesso a 0 (incontrando un ulteriore "1" nella scansione successiva).

Per testare il generico bit della parola, si fa ancora uso di controllo residuo mediante un commutatore a 32 ingressi, ognuno di un bit, trattando X come un *array di 32 bit*.

Il controllo del loop si effettua mediante un registro I, di 6 bit, inizializzato a 0 (tranne alcune soluzioni ottimali) e incrementato via via; nel caso peggiore, si effettuano 32 passi, tale evento essendo segnalato dalla condizione $I_0 = 1$.

Interprete dell'operazione esterna iii)

L'indirizzo IND della memoria A è di 15 bit, dove gli 8 bit più significativi identificano il blocco, i 7 meno significativi l'indirizzo all'interno del blocco. Il valore iniziale di IND è ottenuto per concatenazione dell'identificatore di blocco Y[J] e di una costante di 7 bit a "0", i successivi valori di IND mediante incremento.

Il loop è controllato da un registro H, di 8 bit, inizializzato a 0 e incrementato via via, con condizione di uscita dal loop espressa da $H_0 = 1$. Il numero di parole del blocco che soddisfano il predicato richiesto è contenuto nel registro C, di 8 bit, inizializzato a 0.

Soluzione 1.1

In questa soluzione, l'obiettivo è di *minimizzare il numero di cicli di clock*, cercando di concentrare in una stessa microistruzione il maggior numero di computazioni possibile (oltre, ovviamente, a far uso del massimo parallelismo possibile sia nelle microoperazioni che nelle condizioni logiche).

Ciò è ottenuto, a spese della durata del ciclo di clock:

- utilizzando *variabili di condizionamento "complesse"*, cioè ottenute mediante funzioni applicate alle uscite di registri della PO, e ricordando, nella costruzione della PO, di imporre che tali funzioni non abbiano tra i propri argomenti d'ingresso variabili di controllo (*condizione necessaria per la correttezza*). Questo caso è significativo nelle operazioni esterne *ii), iii)* agli effetti della verifica del predicato richiesto.

In questa soluzione, per ragioni didattiche non viene fatto uso di risorse (ALU) il cascata per eseguire la somma di tre numeri nell'operazione esterna *i).* Questa caratteristica verrà inserita nella Soluzione 1.2.

<u>*Microprogramma 1.1*</u>

Inizializzazione

Sono usati gli ulteriori registri B e N per memorizzare X (o il risultato della prima somma nella *i)*) e J dopo il primo ciclo di clock.

0. ($RDYIN_a$, OP_0, OP_1 = 0 – –) nop, 0;

   (= 1 0 0) reset $RDYIN_a$, set $ACKIN_a$, X + cost_zero(24) ° Y[J] → B, Y → M, J +1 → N, 1;

   (= 1 0 1) reset $RDYIN_a$, set $ACKIN_a$, X → B, 0 → I, 0 → P, 2;

   (= 1 1 –) reset $RDYIN_a$, set $ACKIN_a$, X → B, Y[J] ° cost_zero(7) → IND, 0 → H, 0 → C, 3

Completamento dell'esecuzione della prima operazione esterna

1. ($ACKOUT_b$ = 0) nop, 1;

   (= 1) B + cost_zero(24) ° M[N] → OUT_b, set $RDYOUT_b$, reset $ACKOUT_b$, 0;

Esecuzione della seconda operazione esterna

2. (I_0, $\mathbf{B[I_M]}$, P, $ACKOUT_c$ = 0 0 – –) I + 1 → I, 2;

   (= 0 1 0 –) I + 1 → I, 1 → P, 2;

   (= 0 1 1 0) nop, 2;

   (= 0 1 1 1) 0 → OUT_c, set $RDYOUT_c$, reset $ACKOUT_c$, 0;

   (= 1 – – 0) nop, 2;

   (= 1 – – 1) P → OUT_c, set $RDYOUT_c$, reset $ACKOUT_c$, 0;

Esecuzione della terza operazione esterna

3. (H_0, **segno (B – A[IND])**, $ACKOUT_d$ = 0 0 –) C + 1 → C, H + 1 → H, IND + 1→ IND, 3;

   (= 0 1 –) H + 1 → H, IND + 1 → IND, 3;

   (= 1 – 0) nop, 3;

   (= 1 – 1) C → OUT_d, set $RDYOUT_d$, reset $ACKOUT_d$, 0

Soluzione 1.2

Si tratta della soluzione precedente, con la modifica sull'operazione esterna *i),* che ora viene eseguita in un solo ciclo di clock effettuando la somma di tre numeri mediante due ALU in cascata e l'indirizzamento dei byte di Y mediante due commutatori, il secondo dei quali comandato da J+1 (ulteriore ALU). Ciò comporta anche il test di $ACKOUT_b$ nella microistruzione 0.

Microprogramma 1.2

Inizializzazione

0. ($RDYIN_a$, OP_0, OP_1, $ACKOUT_b$ = 0 – – –) nop, 0;

   (= 1 0 0 0) reset $RDYIN_a$, set $ACKIN_a$ X → B, Y → M, J → N, 1;

   (= 1 0 0 1) reset $RDYIN_a$, set $ACKIN_a$, **X + cost_zero(24) ° Y[J] + cost_zero(24) ° Y[J+1] → OUT_b**, set $RDYOUT_b$, reset $ACKOUT_b$, 0;

   (= 1 0 1–) reset $RDYIN_a$, set $ACKIN_a$, X → B, 0 → I, 0 → P, 2;

   (= 1 1 – –) reset $RDYIN_a$, set $ACKIN_a$, X → B, Y[J] ° cost_zero(7) → IND, 0 → H, 0 → C, 3

Completamento dell'esecuzione della prima operazione esterna.

La microistruzione 1 è inserita solo per ragioni di efficienza: se l'unità U_b non è pronta a ricevere, non viene impedito all'unità U_a di inviare un nuovo messaggio (nella microistruzione 0, in presenza di $ACKOUT_b$ = 0, viene comunque inviato l'ack a U_a).

1. **($ACKOUT_b$ = 0) nop, 1;**

   (= 1) B + cost_zero(24) ° M[N] + cost_zero(24) ° M[N+1] → OUT_b, set $RDYOUT_b$, reset $ACKOUT_b$, 0

Esecuzione della seconda operazione esterna

2. (I_0, $B[I_M]$, P, $ACKOUT_c$ = 0 0 – –) I + 1 → I, 2;

   (= 0 1 0 –) I + 1 → I, 1 → P, 2;

   (= 0 1 1 0) nop, 2;

   (= 0 1 1 1) 0 → OUT_c, set $RDYOUT_c$, reset $ACKOUT_c$, 0;

   (= 1 – – 0) nop, 2;

   (= 1 – – 1) P → OUT_c, set $RDYOUT_c$, reset $ACKOUT_c$, 0;

Esecuzione della terza operazione esterna

3. (H_0, segno (B – A[IND]), $ACKOUT_d$ = 0 0 –) C + 1 → C, H + 1 → H, IND + 1 → IND, 3;

   (= 0 1 –) H + 1 → H, IND + 1 → IND, 3;

   (= 1 – 0) nop, 3;

   (= 1 – 1) C → OUT_d, set $RDYOUT_d$, reset $ACKOUT_d$, 0

Soluzione 2.1

In questa soluzione, l'obiettivo è di *minimizzare il ciclo di clock*, agendo opportunamente sul ritardo delle funzioni ω_{PO} e σ_{PO}:

- per la minimizzazione della prima, tutte le variabili di condizionamento sono "semplici", cioè sono prelevate direttamente sull'uscita di registri, senza ulteriori trasformazioni;
- per la minimizzazione della seconda, applicheremo quanto detto nel commento all'operazione esterna *i)*, effettuando la somma di tre numeri in due cicli di clock.

Compatibilmente con questi vincoli, il microprogramma dovrà comunque essere ottimizzato come numero di cicli di clock, applicando il massimo parallelismo possibile sia nelle microoperazioni che nelle condizioni logiche (teoricamente, potrebbero anche esistere soluzioni che minimizzano il ciclo di clock a discapito del parallelismo, ma queste non avrebbero alcun interesse per i nostri scopi).

Microprogramma 2.1

Inizializzazione

0. ($RDYIN_a$, OP_0, OP_1 = 0 – –) nop, 0;

   (= 1 0 0) reset $RDYIN_a$, set $ACKIN_a$, X + cost_zero(24) ° Y[J] → B, Y → M, J +1 → N, 1;

   (= 1 0 1) reset $RDYIN_a$, set $ACKIN_a$, X → B, 0 → I, 0 → P, 2;

   (= 1 1 –) reset $RDYIN_a$, set $ACKIN_a$, X → B, Y[J] ° cost_zero(7) → IND, 0 → H, 0 → C, 4

Completamento dell'esecuzione della prima operazione esterna

1. ($ACKOUT_b$ = 0) nop, 1;

   (= 1) B + cost_zero(24) ° M[N] → OUT_b, set $RDYOUT_b$, reset $ACKOUT_b$, 0;

Esecuzione della seconda operazione esterna

Nell'operazione esterna *ii),* la variabile di condizionamento, uguale al valore del generico bit della parola B, è memorizzata in un registro E di un bit.

2. **B[I_M] → E**, 3
3. (I_0, **E**, P, $ACKOUT_c$ = 0 0 – –) I + 1 → I, 2;

   (= 0 1 0 –) I + 1 → I, 1 → P, 2;

   (= 0 1 1 0) nop, 3;

   (= 0 1 1 1) 0 → OUT_c, set $RDYOUT_c$, reset $ACKOUT_c$, 0;

   (= 1 – – 0) nop, 3;

   (= 1 – – 1) P → OUT_c, set $RDYOUT_c$, reset $ACKOUT_c$, 0;

Esecuzione della terza operazione esterna

4. (H_0, $ACKOUT_d$ = 0 –) **segno (B – A[IND]) → S**, 5;

   (= 1 0) nop, 4;

   (= 1 1) C → OUT_d, set $RDYOUT_d$, reset $ACKOUT_d$, 0

5. (**S** = 0) C + 1 → C, H + 1 → H, IND + 1 → IND, 4;

   (= 1) H + 1 → H, IND + 1 → IND, 4

Soluzione 2.2

Nella Soluzione 2.1 si è fatto uso del massimo grado di parallelismo possibile, compatibilmente con l'algoritmo adottato. È possibile una ulteriore ottimizzazione, individuando un algoritmo che permetta di eseguire i loop delle operazioni esterne *ii)* e *iii)* in un singolo ciclo di clock, invece che in due: ciò si ottiene con una *opportuna inizializzazione delle variabili di condizionamento E, S e dei contatori I, H, facendo in modo che il primo passo venga anticipato nella microistruzione 0*; *durante il loop sarà possibile effettuare in parallelo l'incremento dei contatori e la modifica della variabile di condizionamento*.

Il risultato è una soluzione che, eccetto per l'operazione esterna *i),* minimizza il tempo medio di elaborazione, ed è ritenuta la soluzione ottimale (a meno che l'operazione esterna *i)* non sia la più frequente).

Microprogramma 2.2

Inizializzazione

0. ($RDYIN_a$, OP_0, OP_1 = 0 – –) nop, 0;

   (= 1 0 0) reset $RDYIN_a$, set $ACKIN_a$, X + cost_zero(24) ° Y[J] → B, Y → M, J +1 → N, 1;

   (= 1 0 1) reset $RDYIN_a$, set $ACKIN_a$, X → B, **1 → I**, **X[0] → E**, 0 → P, 2;

   (= 1 1 –) reset $RDYIN_a$, set $ACKIN_a$, X → B, **Y[J] ° 0000001 → IND, 1 → H**,
   segno (X – (A[Y[J] ° cost_zero(7))]) → S, 0 → C, **3**

Completamento dell'esecuzione della prima operazione esterna

1. ($ACKOUT_b$ = 0) nop, 1;

   (= 1) B + cost_zero(24) ° M[N] → OUT_b, set $RDYOUT_b$, reset $ACKOUT_b$, 0;

Esecuzione della seconda operazione esterna

2. (I_0, **E**, P, $ACKOUT_c$ = 0 0 – –) I + 1 → I, **B[I_M] → E** , 2;

   (= 0 1 0 –) I + 1 → I, 1 → P, **B[I_M] → E** , 2;

   (= 0 1 1 0) nop, **2**;

   (= 0 1 1 1) 0 → OUT_c, set $RDYOUT_c$, reset $ACKOUT_c$, 0;

   (= 1 – – 0) nop, **2**;

   (= 1 0 – 1) P → OUT_c, set $RDYOUT_c$, reset $ACKOUT_c$, 0;

   (= 1 1 0 1) 1 → OUT_c, set $RDYOUT_c$, reset $ACKOUT_c$, 0;

   (= 1 1 1 1) 0 → OUT_c, set $RDYOUT_c$, reset $ACKOUT_c$, 0;

Esecuzione della terza operazione esterna

3. (H_0, **S**, $ACKOUT_d$ = 0 0 –) C + 1 → C, H + 1 → H, IND + 1 → IND, **segno (B – A[IND]) → S**, 3;

   (= 0 1 –) H + 1 → H, IND + 1 → IND, **segno (B – A[IND]) → S**, 3;

   (= 1 – 0) nop, 3;

   (= 1 1 1) C → OUT_d, set $RDYOUT_d$, reset $ACKOUT_d$, 0

   (= 1 0 1) C + 1 → OUT_d, set $RDYOUT_d$, reset $ACKOUT_d$, 0

In questa sede, le domande b), c), d), e) saranno esaminate con riferimento al caso della sola Soluzione 1.1.

b) Parte Operativa

Utilizzando la metodologia generale, dal microprogramma si ottiene lo schema mostrato nella figura a pagina seguente.

È stato usato il *numero minimo di ALU*. Tale numero (quattro) è riconoscibile dalla frase in cui c'è il massimo parallelismo nell'uso delle reti di calcolo:

(H_0, **segno (B – A[IND])**, $ACKOUT_d$ = 0 0 –) **C + 1** → C, **H + 1** → H, **IND + 1** → IND, 3;

Di queste quattro ALU, una (ALU0) deve essere dedicata solo all'operazione *segno (B – A[IND]),* allo scopo di garantire che la PO sia un automa di Moore.

Le altre possono essere così definite:

ALU1: X + Y[J], B + M[N], IND+1

ALU2 : H+1, C+1

ALU3: I+1, J+1

Poiché gli operandi di operazioni assegnate ad alcune ALU (ALU1, ALU3) hanno lunghezza (numero di bit) diversa, occorre provvedere all'*allineamento* di certi valori, completandoli con un opportuno numero di "0" cablati in testa. Ad esempio, alla ALU1 possono essere portati i valori X (32 bit), B (32 bit), IND (15 bit), Y[J] (8 bit) e M[N] (8 bit): poiché occorre una ALU a 32 bit, in testa ai valori provenienti dall'uscita di IND, di Y[J] e di M[N] vengono cablati, in corrispondenza degli ingressi dei commutatori, rispettivamente 17 bit, 24 bit e 24 bit tutti uguali a "0". Dunque anche gli allineamenti dei byte Y[J] e M[N], che nel microprogramma erano espressi esplicitamente per ragioni di correttezza semantica, rientrano in un caso più ampio quando si va a costruire lo schema della rete sequenziale PO. La situazione inversa si ha quando si tratti di scrivere il valore dell'uscita di una ALU in un registro con un numero inferiore di bit: ad esempio, del valore di uscita della ALU1 solo i 15 bit meno significativi sono portati all'ingresso del commutatore di IND.

La *funzione delle uscite* ω_{PO} è espressa, *ad ogni ciclo di clock*, come segue, per quanto riguarda solo le variabili dei condizionamento (le uscite esterne sono ovvie):

$x_0 = RDYIN_a$

$x_1 = OP_0$

$x_2 = OP_1$

$x_3 = ACKOUT_b$

$x_4 = I_0$

$\boldsymbol{x_5 = B[I_M]}$

$x_6 = P$

$x_7 = ACKOUT_c$

$x_8 = H_0$

$\boldsymbol{x_9 = segno\_ALU0\ (B - A[IND])}$

$x_{10} = ACKOUT_d$

Gli unici casi in cui tali variabili non coincidono con uscite di registri sono x_5 e x_9. In entrambi i casi, le reti combinatorie che producono tali valori non hanno ingressi che siano variabili di controllo, dunque è garantita la condizione necessaria per la correttezza della PO (automa di Moore). Si ricorda che tale condizione non può essere verificata sul microprogramma, bensì deve essere verificata facendo l'analisi della rete sequenziale PO.

La *funzione di transizione* σ_{PO} applicata alla parte di stato rappresentata dal registro contatore H è espressa come:

$\sigma_{PO/H}$: in_H = **when** $\beta_H = 1$ **do**

if not $\alpha_{KH} = 0$ **then** 0

else if $\alpha_{KALU2} = 0$ **then** H **else** C

a significare che, *ad ogni ciclo di clock*, l'espressione "**if not** $\alpha_{KH} = 0$ **then** …" fornisce il valore della parte di stato successivo, relativa ad H, solo a condizione che venga abilitata la scrittura in H.

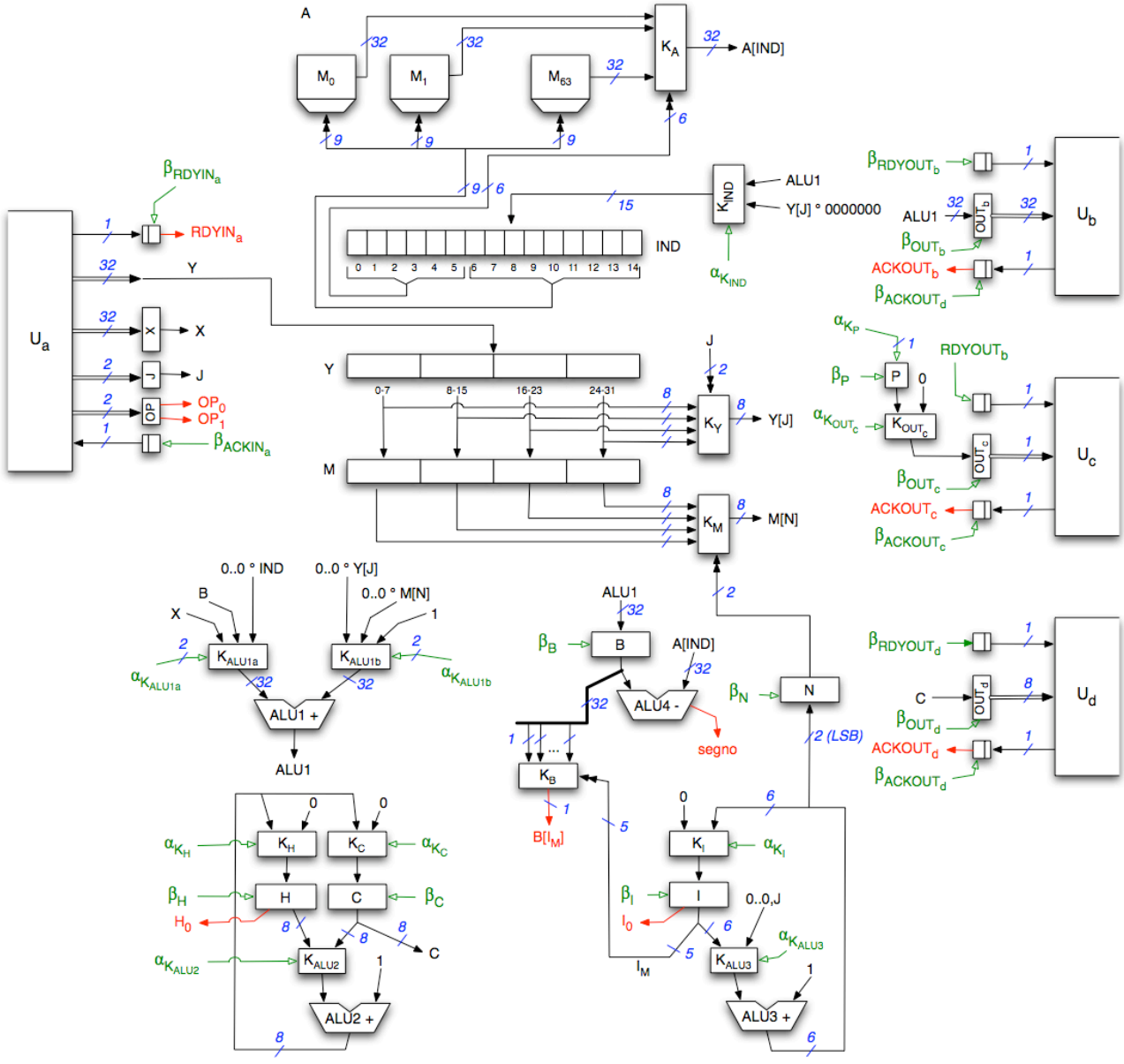


Figura 1: PO relativa al microprogramma 1.1. In rosso sono evidenziate le variabili di condizionamento, in verde le variabili di controllo. La freccia con la doppia punta indica l'utilizzo del controllo residuo.

c) Parte Controllo

La definizione completa delle funzioni delle uscite, ω_{PC}, e di transizione dello stato interno, σ_{PC}, si ottiene costruendo le rispettive tabelle di verità; tali tabelle constano di tante righe distinte per quante sono le frasi distinte del microprogramma, cioè 16, a fronte delle $2^{12} = 4096$ possibili combinazioni del complesso delle 2 variabili dello stato interno e delle 10 variabili d'ingresso.

La variabili di uscita della PC sono 23, corrispondenti a 2^{23} = 8M stati di uscita, dei quali solo 12 distinti (microoperazioni distinte).

Le codifiche dei quattro stati interni della PC si ottengono mediante le variabili dello stato interno y_0, y_1 nel modo più diretto (0 = 00, 1 = 01, 2 = 10, 3 = 11). Siano Y_0, Y_1 le variabili dello stato successivo.

Per rispondere alle domande non è affatto necessario costruire per intero le tabelle di verità; è molto più semplice osservare il microprogramma, sostituendo alle etichette delle microistruzioni le codifiche dello stato interno presente e successivo, e considerando solo le situazioni in cui interessa una certa variabile di controllo. Si ottiene:

σ_{PC}:

$$Y_0 = \overline{y_0}\ \overline{y_1}\ RDYIN_a\ OP_1 + y_0\ \overline{y_1}\ \overline{I_0}\ \overline{B[I]} + y_0\ \overline{y_1}\ \overline{I_0}\ B[I]\ \overline{P} + y_0\ \overline{y_1}\ \overline{I_0}\ B[I]\ P\ ACKOUT_c + y_0\ \overline{y_1}\ I_0\ \overline{ACKOUT_c} + y_0\ y_1\ \overline{H_0} + y_0\ y_1\ H_0\ \overline{ACKOUT_d}$$

$$Y_1 = \overline{y_0}\ \overline{y_1}\ RDYIN_a\ \overline{OP_0} + \overline{y_0}\ y_1\ ACKOUT_b + y_0\ y_1\ \overline{H_0} + y_0\ y_1\ H_0\ \overline{ACKOUT_d}$$

ω_{PC}:

$$\beta_H = \overline{y_0}\ \overline{y_1}\ RDYIN_a\ OP_0 + y_0\ y_1\ \overline{H_0}$$

c) Ciclo di clock

I massimi ritardi di stabilizzazione delle quattro funzioni di PC e PO si valutano come segue.

ω_{PO}:

Solo x_5 e x_9 hanno ritardo non nullo.

x_5 è uscita di un commutatore a quattro ingressi primari e due variabili di controllo. Le porte AND hanno 6 ingressi, la porta OR finale 4 ingressi. Il commutatore è quindi a due livelli di logica, con ritardo $2\ t_p$.

x_9 è l'uscita di una ALU con un ingresso dato dall'uscita della memoria A; quindi:

$$T_9 = T_{ALU} + T_A$$

Quest'ultima è formata da 64 memorie ognuna di capacità 512 parole e tempo di accesso $3t_p$. Ognuna delle 64 memorie è indirizzata con i 6 bit più significativi dell'indirizzo IND, ogni memoria al suo interno è indirizzata dai rimanenti 9 bit meno significativi di IND. Le uscite delle 64 memorie entrano in un commutatore con le 6 variabili di controllo suddette; dunque, le porte AND hanno 7 ingressi e sono realizzate su un solo livello di logica, mentre la funzione OR finale (a 64 ingressi) va realizzata con un albero di arietà 8 e profondità 2, quindi con due livelli di logica. Complessivamente, questo commutatore ha ritardo $3t_p$. La memoria A ha quindi tempo di accesso $6t_p$.

Complessivamente, $T_9 = 11\ t_p$. Dunque:

$$T_{\omega PO} = 11\ t_p$$

ω_{PC}, σ_{PC}:

Il massimo numero di variabili di condizionamento per frase è 4: nel caso peggiore un termine AND di queste funzioni contiene 6 variabili.

Il numero di frasi è 16, per cui teoricamente potrebbero esserci 16 ingressi per la funzione OR; in realtà, non c'è nessun caso in cui una stessa variabile, di controllo o dello stato interno successivo, valga "1" per un numero di volte superiore a 8.

Dunque, entrambe le reti sono a due livelli di logica:

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

σ_{PO}:

Si tratta di esaminare il massimo ritardo di stabilizzazione tra quelli di *tutte* le operazioni elementari di trasferimento tra registri. Quelle di ritardo non nullo fanno tutte uso di una ALU e di due commutatori a due livelli di logica, dunque:

$$T_{\omega PO} = T_{ALU} + 2\ T_K = 9\ t_p$$

In conclusione:

$$\tau = T_{\omega PO} + max\ (T_{\omega PC} + T_{\sigma PO},\ T_{\sigma PC}) + \delta = 23\ t_p$$

È importante osservare che questo ritardo non è calcolato come il ritardo della "microistruzione più lenta". Potrebbe non esistere alcuna microistruzione per la quale i ritardi di tutte e quattro funzioni assumono i rispettivi valori massimi. Per ragioni di riduzione della complessità di progettazione (e senza commettere un errore per eccesso sensibile) la valutazione viene invece data nel caso peggiore valutando singolarmente, e in maniera indipendente, i quattro ritardi massimi senza fare riferimento ad una specifica microistruzione.

e) Tempo medio di elaborazione

Per l'operazione esterna *ii)* non può essere data una valutazione significativa, in quanto essa dipende strettamente dai valori dei dati e dallo stesso risultato numerico dell'operazione.

L'operazione esterna *i)* è completata in due cicli di clock:

$$T_i = 2\ \tau$$

L'operazione esterna *iii)* comporta l'esecuzione delle seguenti microistruzioni:

- microistruzione 0: una volta
- microistruzione 3: 128 (loop) + 1 (uscita dal loop) volte

Quindi:

$$T_{iii} = 130\ \tau$$

Domanda 3

a) Entrambe le operazioni esterne sono *funzioni* ("pure"), quindi *concettualmente*, a livello hardware, la loro implementazione *è* una rete combinatoria.

Quindi, *se tale rete è realmente fattibile*, l'unità U può essere sostituita da una rete combinatoria, data dall'unione delle due reti combinatorie per realizzare le funzioni richieste.

Pur con questa realizzazione, il modello (PC, PO) è ancora valido, ma implementato in maniera particolare:

- la PO è data dalle interfacce di ingresso e di uscita collegate dalla rete combinatoria suddetta,
- la PC è una rete combinatoria, il cui scopo è semplicemente quello di scegliere l'interfaccia di uscita in funzione di OP. Infatti, il microprogramma sarebbe del tipo:

  0. (RDYIN, OP, $ACKOUT_b$, $ACKOUT_c$ = 0 – – –) nop, 0; (= 1 0 0 –) nop, 0; (= 1 1 – 0) nop, 0;

     (= 1 0 1 –) reset $RDYIN_a$, set $ACKIN_a$, *F* (X, J, J) → OUT_b, set $RDYOUT_b$, reset $ACKOUT_b$, 0;

     (= 1 1 – 1) reset $RDYIN_a$, set $ACKIN_a$, *G* (X) → OUT_c, set $RDYOUT_c$, reset $ACKOUT_c$, 0

  dove *F* è la funzione corrispondente alla prima operazione esterna e *G* alla seconda.

Quindi, complessivamente, U è una rete combinatoria che collega interfacce d'ingresso con interfacce di uscita.

b) A questo punto si tratta di verificare *la reale fattibilità delle reti combinatorie*. Come sempre, data la complessità esponenziale del procedimento di sintesi in funzione del numero di ingressi, se si decide di percorrere la strada della definizione mediante tabella di verità, è perché il numero di variabili booleane d'ingresso è, in pratica, notevolmente basso; questo non è il nostro caso (*F* ha 72 variabili d'ingresso, *G* ne ha 32).

La strada alternativa, a bassa complessità, è quella che viene usata anche nel procedimento di sintesi di una PO: usare *combinazioni di operatori standard* partendo da una *specifica mediante un formalismo algoritmico*. Entrambe le operazioni *F* e *G* si prestano a questo tipo di soluzione.

La funzione *F* è descritta da:

$$somma = X + Y[J] + Y[J+1]$$

L'implementazione è ottenuta formalmente come segue:

- il risultato è l'uscita di un addizionatore:

$$alu1 = alu2 + Y[J+1] \qquad (1)$$

- il primo addendo è ottenuto come uscita di un secondo addizionatore:

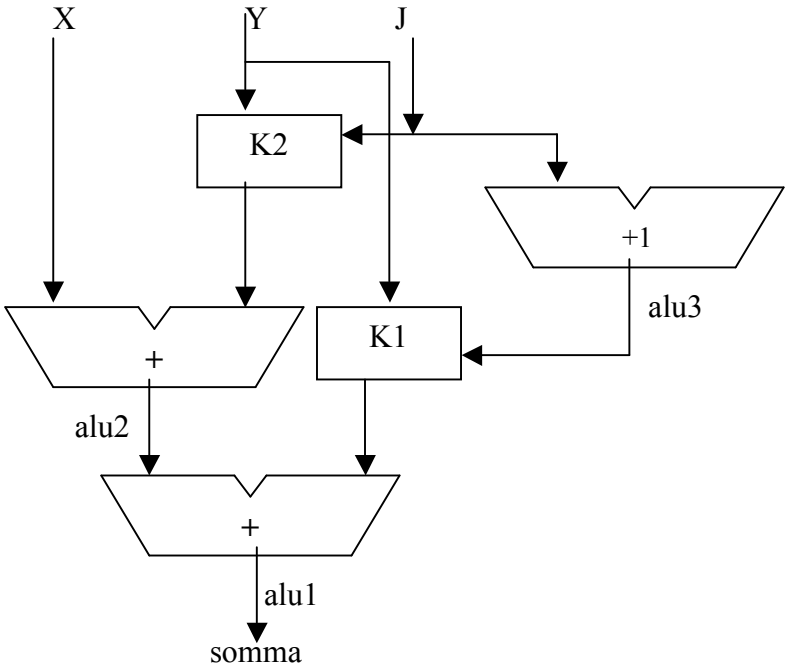
$$alu2 = X + Y[J] \qquad (2)$$

- il secondo addendo della (1) è dato dal byte di Y avente identificatore J+1, quindi è dato dall'uscita di un commutatore K1 comandato dall'uscita di un terzo addizionatore (alu3) con ingresso J;
- il secondo addendo della (2) è dato dal byte di Y avente identificatore J, quindi è dato dall'uscita di un commutatore K2 comandato da J.

Lo schema è mostrato a pagina seguente.

Il ritardo di stabilizzazione è dato dalla somma del ritardo di due ALU e di un commutatore: K2, ALU2, ALU1, oppure ALU3, K1, ALU1:

$$12\ t_p$$



La funzione *G* restituisce vero se e solo se il numero naturale X è una potenza di 2. Formalmente si può definire così:

potenza = **if** (X = 000…01) **or** (X = 000…10) **or** … **or** (X = 001 …00) **or** (X = 010 …00) **or** (X = 100...00)

Si tratta di un predicato costituito da 32 termini tra loro in OR, dove ogni termine verifica che in X ci sia un solo bit uguale a "1" e tutti gli altri a "0". L'espressione booleana, in forma *canonica* SP, è semplicemente la stessa descrizione espressa con un diverso zucchero sintattico:

$$\text{potenza} = \overline{X_0}\,\overline{X_1}\,\overline{X_2}\ldots\overline{X_{30}}\,X_{31} + \overline{X_0}\,\overline{X_1}\,\overline{X_2}\ldots X_{30}\,\overline{X_{31}} + \ldots +$$

$$+ \overline{X_0}\,\overline{X_1}\,X_2\ldots\overline{X_{30}}\,\overline{X_{31}} + \overline{X_0}\,X_1\,\overline{X_2}\ldots\overline{X_{30}}\,\overline{X_{31}} + X_0\,\overline{X_1}\,\overline{X_2}\ldots\overline{X_{30}}\,\overline{X_{31}}$$

La rete combinatoria consta di 32 AND in parallelo, ognuno a 32 ingressi, e di un OR finale a 32 ingressi. A causa delle ipotesi sul massimo numero di ingressi per porta (8), la generica funzione AND e la funzione OR devono essere implementate ognuna come un albero di porte di arietà 8 e profondità 2. Quindi, il ritardo di stabilizzazione è dato da

$$4\ t_p$$

c) Quando tutte le operazioni esterne siano funzioni "pure", la realizzazione come rete combinatoria, rispetto a quella come unità costituita da due reti sequenziali (reali), ha il vantaggio di un minor tempo di elaborazione. Questo è evidente soprattutto nel caso di una operazione esterna come la seconda (potenza di 2), dove la realizzazione a microprogramma mediante un algoritmo iterativo ha un tempo di elaborazione multiplo della lunghezza di parola, mentre la realizzazione con la rete combinatoria di cui sopra ha tempo di elaborazione di un solo ciclo di clock (si tratta poi di confrontare i cicli di clock nei due casi).

In generale, lo svantaggio della soluzione come rete combinatoria è la potenziale maggiore complessità del procedimento, che, come detto più volte, varia esponenzialmente con il numero di ingressi delle reti, **a meno di tutti quei casi in cui la realizzazione, di bassa complessità, è ottenuta direttamente a partire dalla specifica senza passare attraverso la tabella di verità, come nella soluzione data sopra:** in essa, infatti, le reti combinatorie "somma" e "potenza" sono state ottenute partendo da un formalismo algoritmico, nel primo caso usando combinazioni di componenti standard, nel secondo ricavando l'espressione logica direttamente da tale formalismo.

Domanda 4

L'affermazione è falsa, in quanto è falsa la premessa:

$$\text{poiché } C_{27} = \textbf{if } \alpha \textbf{ then } M[I]_{27} \textbf{ else } M[J]_{27}$$

Infatti, detta *x* la variabile di condizionamento usata nella microistruzione *i*, la sua definizione (*funzione delle uscite*) è:

$$x = C_{27} \quad \text{ad ogni ciclo di clock}$$

dove C_{27} denota l'uscita del bit 27-esimo del registro *C*, e quindi una porzione dello stato interno della PO.

Dunque, per quanto riguarda la parte oggetto della domanda, lo *stato di uscita della PO* dipende, *ad ogni ciclo di clock*, solo dallo stato interno della PO stessa, ed è verificata la condizione necessaria per la correttezza.

L'espressione

$$\textbf{if } \alpha \textbf{ then } M[I]_{27} \textbf{ else } M[J]_{27}$$

denota, invece, l'ingresso del bit 27-esimo del registro C (si noti che, in questo esempio, è sempre $\beta_C = 1$), e dunque tale espressione esprime una parte della funzione di *transizione dello stato interno della PO*.