III.

Firmware e Modello di Unità di Elaborazione

1. Struttura generale di un sistema a livello firmware

1.1 Unità di elaborazione

Ricordiamo dal Capitolo I che un *sistema di elaborazione* a livello firmware è costituito da un certo numero di *unità di elaborazione* tra loro interagenti, U_1, ..., U_n, , come mostrato genericamente in fig. 1.1 :

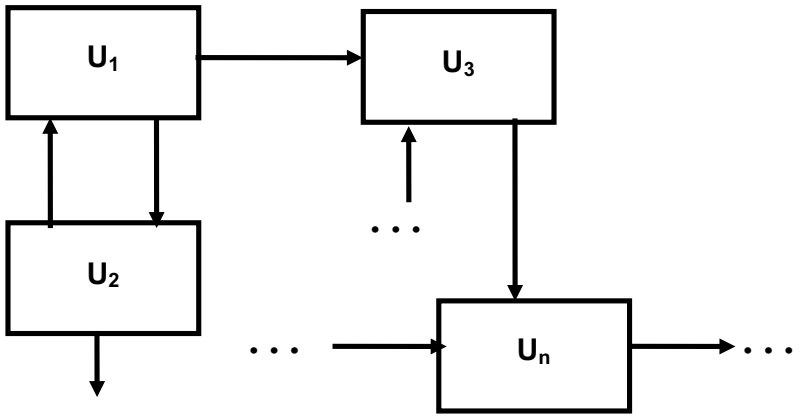


Fig. 1.1 - Sistema di elaborazione come insieme di unità interagenti

Ad ogni unità di elaborazione è affidato un certo sottoinsieme delle funzionalità dell'intero sistema. Attraverso l'interazione tra unità viene realizzato il compito che globalmente si intende affidare all'intero sistema. Ad esempio, in un calcolatore general purpose si riconoscono molte unità, tra le quali il processore centrale, la memoria principale, e le unità di ingresso-uscita includenti anche le memorie secondarie ; ogni unità è specializzata verso uno specifico compito, ma attraverso l'interazione tra le unità si realizza un sistema avente la caratteristica di *essere generale nei confronti del supporto fornito ai livelli superiori del sistema* ; in particolare il funzionamento di ogni unità e l'interazione tra le varie unità realizza *l'interpretazione* di qualunque programma espresso con il formalismo che definisce il *livello assembler* del sistema.

Una unità di elaborazione svolge il proprio specifico compito in modo :

autonomo : l'unità è capace di controllare la propria elaborazione in modo del tutto indipendente, pur ricavando informazioni sulle funzionalità da eseguire ed i valori dei dati attraverso l'interazione con altre unità ;

sequenziale : il funzionamento dell'unità è descritto da un programma sequenziale. Chiameremo **microprogramma** la descrizione del funzionamento dell'unità, e **microlinguaggio** il linguaggio di programmazione sequenziale con cui esprimere il microprogramma.

1.2 Modello Parte Controllo - Parte Operativa

Il microprogramma di una unità di elaborazione è a sua volta interpretato dal *livello hardware* sottostante. Questa interpretazione viene realizzata da due reti sequenziali LLC tra loro interagenti, dette **Parte Controllo** (PC) e **Parte Operativa** (PO), come mostrato *dal modello generale di unità di elaborazione* di fig. 1.2 :

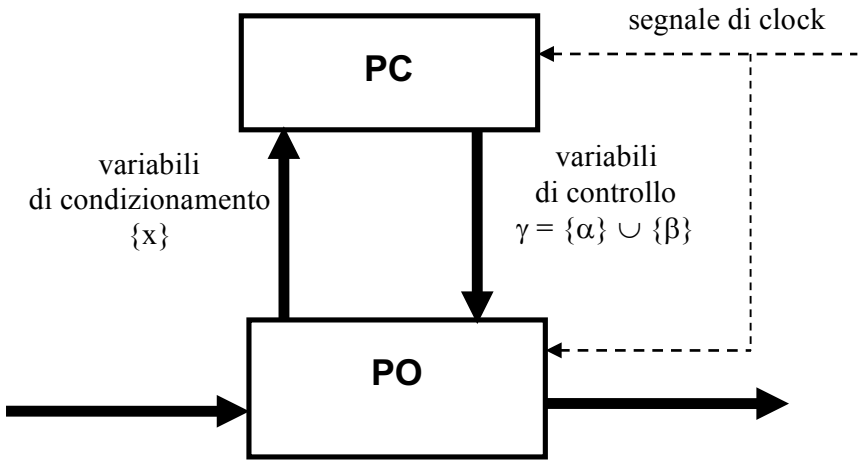


Fig. 1.2 - Modello di unità elaborazione

La PO provvede all'esecuzione delle operazioni previste dai comandi del microlinguaggio, o **microistruzioni**, allo scopo disponendo delle tipiche risorse strutturali messe a disposizione dal livello hardware :

- *commutatori,*
- *selezionatori,*
- *reti di calcolo* mono-funzione o multi-funzione (ALU),
- *registri* : tra questi si distinguono quelli *visibili anche al livello superiore* (in un calcolatore, i registri generali del linguaggio assembler) e quelli *visibili esclusivamente al livello firmware* (i registri utilizzati per le variabili del microprogramma).

PC provvede

- al *controllo della sequenzializzazione* delle microistruzioni, allo scopo utilizzando i valori delle **variabili di condizionamento** {x} relative allo stato della PO (ad esempio, in una certa microistruzione occorre effettuare il test per zero del contenuto di un registro),
- ad *ordinare alla PO l'esecuzione* di ogni microistruzione mediante i valori delle **variabili di controllo** $\gamma = \{\alpha\} \cup \{\beta\}$, , una parte delle quali $\{\beta\}$ abilita/disabilita la scrittura nei registri della PO, mentre

l'altra parte {α} fornisce gli ingressi secondari di commutatori, selezionatori e reti di calcolo (ad esempio, in una certa microistruzione si deve ordinare alla ALU di eseguire l'operazione di addizione sui contenuti di due specifici registri e di scrivere il risultato in uno specifico registro).

PC e PO sono *reti sequenziali LLC* (Level-input Level-output Clocked) impulsate dallo stesso segnale di clock, e quindi aventi lo stesso ciclo di clock : questo, detto *ciclo di clock dell'unità*, verrà determinato in modo tale da permettere la stabilizzazione di entrambe le reti per l'esecuzione di una qualsiasi microistruzione. Poiché la struttura PC - PO rappresenta l'interprete a livello hardware del microlinguaggio del livello firmware*, il modello di programmazione del livello firmware è* dunque *sincrono* : *ogni microistruzione è eseguita in un tempo costante uguale al ciclo di clock*.

Lo scopo di questa parte del corso è di studiare la progettazione di unità di elaborazione mediante un procedimento *formale* che, partendo dalla scrittura del microprogramma, permette di ricavare la struttura della PC e della PO, e quindi di determinare la struttura dell'interprete a livello hardware).

La metodologia che verrà fornita ha carattere di assoluta generalità nei confronti dei vari tipi di sistemi e nei confronti della specifica tecnologia hardware : la metodologia permette di descrivere semplici unità da utilizzare in sistemi specializzati, così come le unità di un calcolatore general purpose incluso il processore centrale. In questa prima parte la metodologia verrà esemplificata nei confronti della progettazione di unità "qualsiasi", non necessariamente facenti parte di un calcolatore general purpose. La sua applicazione allo studio dell'architettura dell'unità centrale e del sottosistema di ingresso-uscita di un calcolatore general purpose sarà oggetto della seconda e terza parte rispettivamente.

1.3 Il procedimento di progettazione delle unità

I passi essenziali nel procedimento formale di progettazione di una unità di elaborazione sono i seguenti :

0. *specifica delle operazioni esterne affidate all'unità*. Il termine **operazioni esterne** sta da indicare le funzionalità del livello superiore che devono essere interpretate dall'unità. Solo nel caso di un processore centrale le operazioni esterne corrispondono effettivamente alle istruzioni assembler ; in tutti i casi che vedremo in questa prima parte, le operazioni esterne vanno sempre pensate come le funzionalità che dall'esterno (quindi da altre unità) vengono richieste all'unità in questione. La specifica delle operazioni esterne può essere data a parole o con un formalismo di volta in volta ritenuto più adatto ;

1. *scrittura del microprogramma che interpreta le operazioni esterne*. In prima istanza, specie nei casi più complessi, è conveniente dare una descrizione dell'interpretazione delle operazioni mediante un formalismo ad alto livello, ad esempio un Pascal-like, per poi passare ad una descrizione nel *microlinguaggio eseguibile* (che verrà definito nel seguito) ;

2. *derivazione formale dello schema della rete sequenziale Parte Operativa a partire dal microprogramma* ;
3. *derivazione formale della rete sequenziale Parte Controllo a partire dal microprogramma* ;
4. *valutazione del ciclo di clock dell'unità* in funzione dei ritardi temporali delle risorse hardware adottate (applicando opportune formule analitiche oggetto della trattazione);
5. *valutazione del tempo medio di elaborazione* di ogni operazione e/o relativo all'insieme delle operazioni affidate all'unità.

È essenziale avere ben presente fin da ora che, ai passi 2 e 3, le strutture hardware della PC e della PO verranno ottenute con procedimenti *formali* a partire dal microprogramma ; in altri termini, la visione erronea secondo la quale i sistemi di elaborazione vengono realizzati mediante approcci empirici, consistenti nel "mettere insieme" in modo artigianale componenti hardware, collegamenti e clock, non appartiene alla metodologia adottata in questo corso. Diversa cosa è che , specie in casi molto complessi (come lo stesso processore centrale), *l'ottimizzazione* del progetto dell'unità (in termini di prestazioni e/o costo) potrà eventualmente richiedere l'applicazione di ulteriori *procedimenti, euristici o approssimati*, ma pur sempre tali da poter essere *ricondotti alla metodologia complessiva* ed al corredo di conoscenze ad essa associato.

2. Esempio 1

In questa sezione descriveremo in dettaglio un semplicissimo esempio di unità di elaborazione, allo scopo di introdurre la metodologia di progettazione. Questa verrà poi formalizzata nella sezione successiva.

2.1 Specifica delle operazioni esterne

Si vuole progettare una unità di elaborazione U avente la struttura esterna di fig. 1.3,

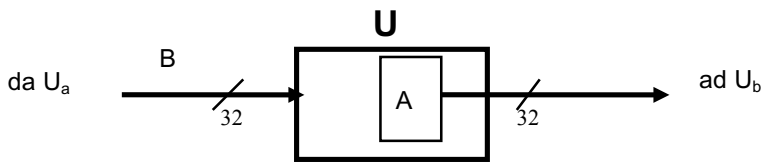


Fig. 1.3 - Visione esterna di una semplice unità di elaborazione

dove A è un registro, di 32 bit, *visibile anche al livello superiore*, e B è un ingresso esterno sul quale viene inviata ad U una sequenza di valori, ognuno rappresentato in complemento a 2 su parola di 32 bit..

L'unità è capace di eseguire una singola *operazione* : per ogni valore B della sequenza d'ingresso, ad A viene assegnato il valore assoluto della somma tra il contenuto corrente di A stesso ed il valore B. Si suppone A inizializzato a 0 : di regola, la condizione che tutti i registri assumano inizialmente il valore 0 è forzata, per ragioni fisiche, all'atto dell'accensione del sistema.

La sequenza di valori B viene inviata ad U da un'altra unità, diciamo U_a, del sistema, e la sequenza di risultati viene inviata da U ad una unità U_b, eventualmente coincidente con U_a. Occorre fare subito una precisazione : per il momento (fino alla sez. 9) non supporremo l'esistenza di alcun *meccanismo di sincronizzazione* per distinguere se, ogni volta che viene calcolata una nuova operazione, il valore B è effettivamente un *nuovo* valore inviato da U_a o è ancora il valore precedente della sequenza (ricordiamo che tutti i segnali, tranne quello per realizzare il clock, sono a *livelli*). La semplificazione sottintende l'ipotesi che, ogni volta che viene calcolata una nuova operazione, sia presente un nuovo valore di B, e che questa sincronizzazione sia sotto la responsabilità di U_a. Allo stesso modo, si suppone che U_b sia capace di distinguere l'arrivo di un nuovo risultato da U. A questa semplificazione, non realistica nella pratica, verrà posto rimedio nella sez. 9, nella quale verranno sviluppati i necessari meccanismi di comunicazione e sincronizzazione per il corretto scambio di informazioni, in ingresso ed in uscita, tra le unità.

2.2 Microprogramma

In un formalismo ad alto livello si può scrivere :

```
repeat
        A := A + B ;
        if A < 0 then A := – A
forever.
```

Come di regola, una unità di elaborazione esegue un *ciclo infinito*, all'inizio del quale ricava le informazioni necessarie, ed al termine del quale ritorna ad attendere tali informazioni. Questa strutturazione è un caso particolare di quella tipica di un interprete ; nel caso più generale di unità multi-operazione, l'interprete, dopo essersi procurato le informazioni sull'operazione esterna da eseguire, provvede ad una fase di *decodifica* dell'operazione stessa, dopo di che passa alla fase di *esecuzione vera e propria*.

Su questa base, il *microprogramma eseguibile* può essere espresso nel seguente modo :

0. $A + B \rightarrow A$, **goto** 1

1. **if** $A_0 = 1$ **then begin** $-A \rightarrow A$, **goto** 0 **end else begin** nop, **goto** 0 **end**.

{con ***A_0*** *si indica il* ***bit più significativo*** *del registro A ; d'ora in poi ci atterremo sempre a questa convenzione, che permette, quando ritenuto opportuno, di prescindere dalla conoscenza della lunghezza di parola).*

Il microprogramma evolve attraverso una sequenza di passi, ognuno corrispondente ad una *microistruzione*. *Ogni microistruzione viene eseguita in un tempo costante uguale al valore del ciclo di clock dell'unità* : questa è infatti la condizione affinchè le reti sequenziali PC, PO si stabilizzino, assicurando la corretta esecuzione della microistruzione stessa prima dell'inizio della successiva.

Ogni microistruzione è caratterizzata da una *etichetta*, o *indirizzo* (come 0 per la prima microistruzione). Trattandosi di un formalismo che deve evidenziare un funzionamento che evolve tra passi di una sequenza, il microlinguaggio *deve* contenere, per ogni microistruzione, l'esplicita indicazione dell'indirizzo di microistruzione successiva (da cui la presenza di strutture **goto**).

Una scrittura come

$$A + B \rightarrow A$$

esprime una **operazione elementare di trasferimento tra registri**. Il significato è quello di una espressione di assegnamento, dove le variabili , trovandoci al livello firmware, corrispondono a registri: il risultato dell'operazione di addizione tra il contenuto corrente (cioè, nel ciclo di clock durante il quale ha luogo l'esecuzione della microistruzione di indirizzo 0) del registro A ed il contenuto corrente del registro B viene assegnato al registro A. Ciò significa che *prima della fine del ciclo di clock, in cui l'operazione elementare è eseguita, il valore del risultato sarà presente in forma stabile all'ingresso del registro* A *; la scrittura di tale valore nel registro* A *avverrà all'inizio del successivo ciclo di clock*, cioè in corrispondenza del fronte di discesa del successivo impulso di clock, *garantendo così che, durante tutta la durata del ciclo di clock , in cui l'operazione elementare è eseguita, il contenuto del registro* A *rimarrà inalterato.*

Il significato dell'operatore "," (come in $A + B \rightarrow A$, **goto** 1) sta ad indicare *esecuzione parallela* : l'esecuzione dell'operazione elementare ($A + B \rightarrow A$) e la determinazione dell'indirizzo successivo (**goto** 1) sono eventi indipendenti che possono avere luogo in un qualsiasi ordine e quindi *contemporaneamente* purché *all'interno dello stesso ciclo di clock*.

La scrittura *nop* indica l'*operazione elementare nulla*, cioè l'operazione che non modifica alcun registro della PO (lo stato interno della PO).

L'uscita di A_0 rappresenta una (l'unica, nell'esempio) *variabile di condizionamento*, utilizzata dalla PC per conoscere, nella microistruzione 1, il risultato dell'operazione elementare eseguita nella microistruzione 0 ($A + B \rightarrow A$).

2.3 Struttura della PO

La PO deve essere capace di eseguire le seguenti operazioni elementari :

- $A + B \rightarrow A$
- $-A \rightarrow A$
- nop

La struttura della PO può allora essere quella mostrata in fig. 1.4 :

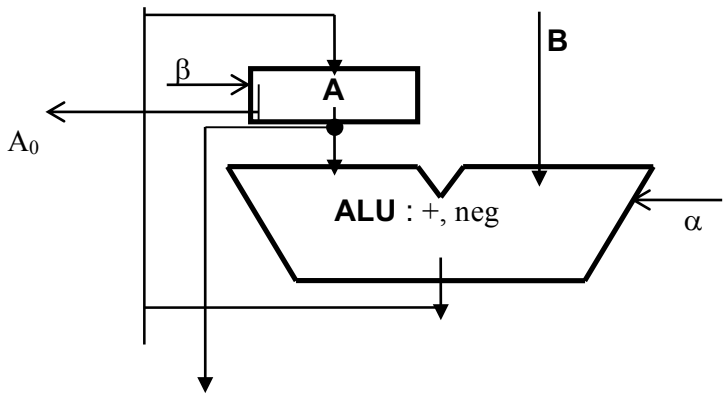


Fig. 1.4 - PO dell'esempio 1

dove si è realizzato sia l'operazione elementare di addizione che quella di negazione in complemento a 2 mediante una stessa ALU; il segnale di controllo α, inviato dalla PC, indica, ad ogni ciclo di clock, quale delle due operazioni deve eseguire la ALU.

Oltre al tipo di segnale di controllo, come α, necessario per comandare, da parte della PC, l'operazione che deve eseguire una rete di calcolo, o per comandare l'instradamento attraverso commutatori (non necessari nell'esempio, ma di regola presenti), la PO di una unità riceve dalla PC anche un altro tipo di segnale di controllo : quello *per abilitare/disabilitare la scrittura in ogni registro,* come visto nella sez. 2 del Capitolo II. Nel nostro caso si tratta del segnale β di fig. 1.4. Al riguardo, si osservi che la scrittura nel registro A non deve avvenire in ogni ciclo di clock : deve avvenire solo quelli in cui si esegue la microistruzione 0, o quelli in cui si esegue la 1 con condizione ($A_0 = 1$) vera, mentre il contenuto A deve rimanere inalterato nei cicli di clock in cui si esegue la microistruzione 1 con condizione ($A_0 = 1$) falsa.

Nei casi in cui un registro deve essere modificato (scrittura abilitata) il segnale β associato vale 1, nel caso opposto (scrittura disabilitata) il segnale β vale 0. Nella fig. 1.5 viene ricordato come, per un generico registro, il segnale β venga messo in AND con l'impulso di clock.

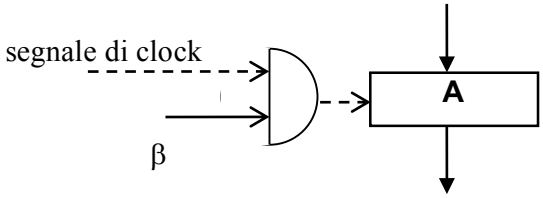


Fig. 1.5 - Meccanismo per l'abilitazione/disabilitazione della scrittura di un registro

Nella fig. 1.6 è riassunto lo schema dei collegamenti tra PC e PO.

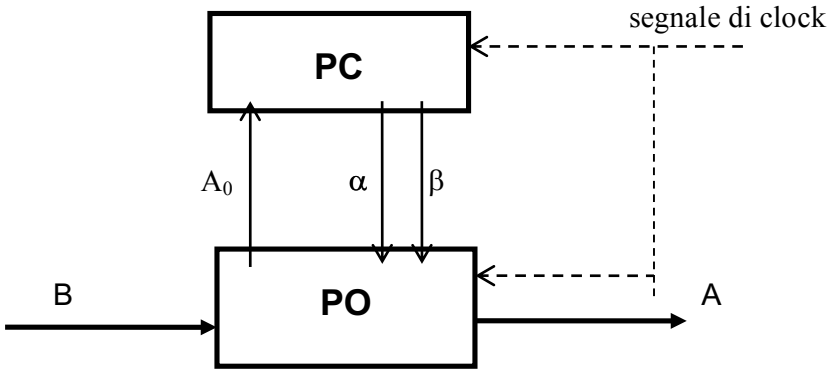


Fig. 1.6. - Modello PC - PO per l'esempio 1

La PO è una rete sequenziale avente come *ingressi* i segnali α e β, *stato interno* memorizzato nel registro A, e come *uscite* tutti i bit del registro A.

Si noti come, anche in un caso semplice come quello dell'esempio, una PO abbia un numero di stati interni molto elevato, dato da 2^m con k numero di bit complessivo di tutti i registri della PO stessa (nell'esempio, m = 32). La sintesi di una rete sequenziale con un numero così elevato di stati interni sarebbe, ingenerale, un problema troppo arduo : esso si rivela invece di bassa complessità (ordine : il prodotto del numero delle microistruzioni del microprogramma per il numero di risorse hardware della PO) grazie al procedimento formale adottato consistente nel derivare la struttura della PO direttamente dal microprogramma e nell'adottare componenti hardware standard (registri, ALU, commutatori, ecc).

Si osserva inoltre che la PO è una rete sequenziale rispondente *al modello matematico di Moore*, in quanto *ad ogni ciclo di clock* le sue uscite, sia verso l'esterno (A) che verso la PC (A_0), sono funzione esclusivamente dello stato interno (contenuto del registro A). Come nell'esempio, la *funzione delle uscite della PO*, ω_{PO}, è di regola molto semplice, spesso la funzione *identità*

La *funzione di transizione dello stato interno della PO*, σ_{PO}, è data dalla stessa implementazione delle operazioni elementari :

- $A + B \rightarrow A$
- $-A \rightarrow A$
- nop

Tale implementazione consiste infatti nel provocare, in ogni ciclo di clock, il passaggio dallo stato presente (contenuto corrente di A) allo stato successivo (valore che l'ingresso di A assumerà alla fine del ciclo di clock, e che quindi A assumerà all'inizio del successivo ciclo di clock) in funzione del valore dello stato presente stesso e del valore dello stato d'ingresso (valori di B, α, β).

2.4 Struttura della PC

Dal microprogramma di U si ricava che la PC deve avere 2 *stati interni, corrispondenti biunivocamente alle microistruzioni del microprogramma* stesso. Infatti, ogni microistruzione indica come deve comportarsi la PC dal punto di vista dell'interpretazione delle variabili di condizionamento e delle azioni da intraprendere in conseguenza. La struttura complessiva della PC è quindi del tipo di quella mostrata in fig. 1.7

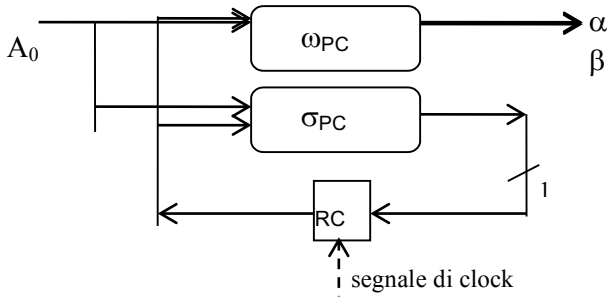


Fig. 1.7 - Struttura complessiva della PC per l'esempio 1

Si tratta ora di definire la *funzione delle uscite della PC*, ω_{PC}, e la *funzione di transizione dello stato interno della PC*, σ_{PC}, la realizzazione logica delle quali rappresenta la parte combinatoria della rete sequenziale PC. Dal microprogramma e dallo schema della PO vediamo che :

- *nello stato interno 0,* per qualunque stato d'ingresso, la PC deve generare i segnali α e β necessari ad eseguire $A + B \rightarrow A$, cioè $\alpha = 0$ e $\beta = 1$, e transire nello stato successivo 1 ;
- *nello stato interno 1,* se lo stato d'ingresso è $A_0 = 1$ allora la PC deve generare i segnali α e β necessari ad eseguire $-A \rightarrow A$, cioè $\alpha = 1$ e $\beta = 1$, e transire nello stato successivo 0 ; nello stesso stato interno 1, se lo stato d'ingresso è $A_0 = 0$, allora PC deve impedire la scrittura del registro A, quindi $\beta = 0$, mentre è indifferente il valore assunto da α in quanto il risultato della ALU, durante il relativo ciclo di clock, non verrà in alcun modo utilizzato, cioè non verrà scritto in alcun registro.

A questa descrizione a parole corrisponde la tabella di verità delle funzioni ω_{PC} e σ_{PC} della PC mostrata in Tab.1.1 :

y	A_0	α	β	Y
0	–	0	1	1
1	0	-	0	0
1	1	1	1	0

Tab. 1.1 - Tabella di verità per le funzioni della PC dell'esempio 1

Si ricavano quindi le espressioni logiche per ω_{PC} e σ_{PC} :

$$\omega_{PC} : \alpha = y \; ; \; \beta = \overline{y} + y A_0$$

$$\sigma_{PC} : Y = \overline{y}$$

Per unità "semplici", cioè con basso numero di stati interni, d'ingresso e di uscita della PC, la progettazione della PC può quindi essere ottenuta mediante il procedimento generale di sintesi delle reti sequenziali ed adottando reti combinatorie, ω_{PC} e $\sigma_{PC,}$ *a due livelli di logica*. Per unità molto complesse ciò non è realisticamente possibile, ma la sintesi può essere notevolmente agevolata mediante una tecnica alternativa che vedremo nella sez. 10.

Si osserva che la PC è una rete sequenziale rispondente al *modello matematico di Mealy*, come è evidente anche dal fatto che esiste almeno una microistruzione nella quale lo stato di uscita (la configurazione di α, β da inviare alla PO) dipende dal valore dello stato d'ingresso (variabile di condizionamento A_0).

Diremo che l'unità di elaborazione risponde, nel suo complesso, al **modello Mealy-Moore**. Questo sarà il modello che adotteremo di regola ; un cenno agli altri due modelli possibili (Moore-Moore, Moore-Mealy) sarà dato nella sez. 11.

Osservazione

In generale esistono diversi gradi di libertà nello scrivere il microprogramma. Nel nostro caso avremmo anche potuto scrivere

0. A + B $\rightarrow$ A, **goto** 1
1. **if** S = 1 **then begin** – A $\rightarrow$ A , **goto** 0 **end else begin** nop, **goto** 0 **end**

dove S denota il Flag all'uscita della ALU che fornisce il valore del *segno* dell'ultima operazione eseguita. Scegliendo questo microprogramma, nello schema della PO va quindi previsto questo Flag. Ciò che è importante notare è che il valore del Flag deve essere *memorizzato in un registro* di 1 bit, e che la variabile di condizionamento S deve quindi essere l'uscita di tale registro. Se infatti avessimo prelevato la variabile di condizionamento "segno" direttamente all'uscita della rete combinatoria ALU, il valore di tale variabile ad ogni ciclo di clock

$$segno\ (A + B)$$

sarebbe stato funzione non solo dello stato interno A, ma anche dello stato d'ingresso α ; di conseguenza, il modello matematico della PO sarebbe stato quello di Mealy. È agevole dimostrare che il modello complessivo di unità *Mealy-Mealy* non è ammissibile in quanto, in generale, non ha un comportamento determinato all'interno del ciclo di clock. Si veda anche l'esercizio 3 della sez. 3.8.

2.5 Ciclo di clock

Nella sez. 3.4 dimostreremo che la lunghezza del ciclo di clock nel modello Mealy-Moore è data da :

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

dove $T_{\omega PO}$ è il *massimo* ritardo della funzione delle uscite della PO, $T_{\omega PC}$ quella della funzione delle uscite della PC, $T_{\sigma PO}$ quello della funzione di transizione dello stato interno della PO, $T_{\sigma PC}$ quello della funzione di transizione dello stato interno della PC, δ la durata dell'impulso di clock. Per quanto riguarda i ritardi delle funzioni della PO occorre considerare *quelli massimi valutati su tutto l'insieme delle microistruzioni.*

Assumiamo i seguenti valori dei *massimi* ritardi temporali delle risorse hardware :

- singola porta logica AND, OR : 1 nsec,
- ALU : 5 nsec (ritardo dell'operazione di addizione),
- durata dell'impulso di clock : 1 nsec,
- collegamenti e porte NOT : ~ 0 (considerati inclusi nei ritardi delle porte logiche AND, OR).

Si ricava dunque che :

$T_{\omega PO} = 0$.

$T_{\sigma PO} = 5$ nsec (ritardo della ALU nella microistruzione 0 e nella 1 con condizione vera),

$T_{\omega PC} = 2$ nsec (rete a due livelli di logica per la funzione di β),

$T_{\sigma PC} = 0$ (nota : anche considerando un ritardo non nullo per la porta NOT, si avrebbe comunque che $T_{\sigma PC} < T_{\omega PC} + T_{\sigma PO}$).

e di conseguenza

$$\tau = 8 \text{ nsec}$$

Il clock ha quindi una *frequenza*

$$f = 1 / \tau = 10^6 * 10^3 / 8 = 125 \text{ MHz}$$

un valore “normale” dell’attuale tecnologia dei circuiti integrati (la tendenza dei costruttori è, già quest’anno, di passare ai 200 MHz ed oltre).

2.6 Tempo medio di elaborazione

Il **tempo medio di elaborazione** di una unità viene calcolato come

$$T = k\,\tau$$

dove *k* è il *numero medio di cicli di clock necessari ad eseguire la generica operazione del microprogramma*. Nel nostro esempio si ha semplicemente :

$$T = 2\,\tau = 16 \text{ nsec}$$

La **banda di elaborazione** è definita *come il numero medio di operazioni esterne che l’unità può eseguire nell’unità di tempo*. Espressa come numero di operazioni esterne al secondo (*ops*), la banda si calcola come :

$$B = 1 / T$$

Nel nostro esempio,

$$B = 62.5 \text{ Mops.}$$

3. Formalizzazione del procedimento di progettazione

A partire dalla specifica delle operazioni esterne, il procedimento di progettazione ha lo scopo di realizzare la struttura dell’unità di elaborazione secondo il modello PC - PO di fig. 1.2, e di valutarne le prestazioni (tempo medio di elaborazione, banda di elaborazione).

3.1 Microlinguaggio

La formalizzazione del procedimento è basata sulla scrittura del *microprogramma eseguibile*, ogni passo del quale (microistruzione) è eseguito esattamente in un ciclo di clock. Nel modello Mealy-Moore il *microlinguaggio* è detto **PS** (Phrase Structured) ; la generica microistruzione ha la seguente struttura :

i. **case** $x_{i1}\, x_{i2} \ldots x_{in}$ **of**

$00 \ldots 0 : \mu op_0$, **goto** j_0

$00 \ldots 1 : \mu op_1$, **goto** j_1

...

$11 \ldots 1 : \mu op_{2^n-1}$, **goto** j_{2^n-1}

dove

a) *i* è l'*etichetta* o *indirizzo* della microistruzione ;

b) x_{i1} x_{i2} ... x_{in} sono *variabili di condizionamento*, in numero arbitrario da zero fino al massimo della cardinalità dell'insieme {x} ;

c) i valori delle guardie del *case*, date da tutte le possibili combinazioni delle variabili di condizionamento testate x_{i1} x_{i2} ... x_{in}, sono dette **condizioni logiche** della microistruzione : C_{ih}, con h = 0 ... 2^n-1 ;

d) μop_{ih} è una **micooperazione**, che può essere :

   d1. la microoperazione nulla, *nop*,

   d2. una operazione non nulla di trasferimento tra registri,

   d3. un insieme di operazioni non nulle di trasferimento tra registri eseguite in parallelo nello stesso ciclo di clock ; vedremo questo caso nella sez. 6;

e) j_h è un *indirizzo di microistruzione successiva* ;

f) la coppia μop_{ih}, **goto** $_{jh}$ è detta **frase** ; la tripla Cih : μop_{ih}, **goto** $_{jh}$ è detta **frase condizionale**.

Invece della precedente, nel seguito useremo la notazione più compatta :

$$\mathbf{i.\ (C_{i0})\ \mu op_0,\ j_0\ ;\ (C_{i1})\ \mu op_1,\ j_1\ ;\ ...\ ;\ (Ci_{2^n-1})\ \mu op_{2^n-1},\ j_{2^n-1}}$$

con lo stesso significato e nomenclatura dei simboli.

Ad esempio :

$$3.\ (A_0,\ Z = 0\ 0)\ A + B \rightarrow A,\ 4\ ;\ (A_0,\ Z = 01)\ M - N \rightarrow M,\ 5\ ;\ (A_0\ Z = 1\ -)\ sh_{r1}\ (N) \rightarrow N,\ 0$$

o, visto che le variabili di condizionamento testate nelle condizioni logiche di una stessa microistruzione sono sempre le stesse :

$$3.\ (A_0,\ Z = 0\ 0)\ A + B \rightarrow A,\ 4\ ;\ (= 01)\ M - N \rightarrow M,\ 5\ ;\ (= 1\ -)\ sh_{r1}\ (N) \rightarrow N,\ 0$$

Si noti che, in alcune condizioni logiche, il valore di una o più variabili di condizionamento può essere *non specificato* (nell'esempio precedente A_0 Z = 1 -) : ciò significa che esistono più condizioni logiche della stessa microistruzione cui corrisponde la stessa frase.

3.2 Struttura della PO

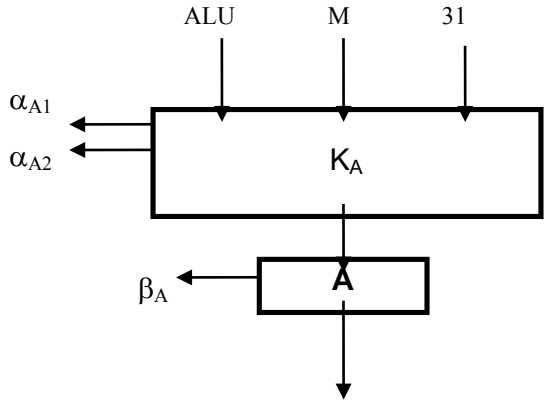
La struttura della PO, vista come *rete sequenziale di Moore*, viene ricavata formalmente dal microprogramma mediante il seguente procedimento.

a1. Si individuano tante *classi di operazioni elementari* per quanti sono i *registri* destinazione (che compaiono a destra dell'operatore $\rightarrow$). Ciò permette di individuare i possibili ingressi di ogni registro. Ad esempio se :

$$\text{classe del registro A : } A + B \rightarrow A \text{ ; } A - B \rightarrow A \text{ ; } M \rightarrow A, 31 \rightarrow A$$

gli ingressi possibili di A sono l'uscita di una ALU (indicata con ALU), l'uscita del registro M (indicata con M) e la costante 31 (da considerare "cablata" ad hardware o presente in un registro in sola lettura) ;

a2. si ricava *una sottostruttura indipendente per ogni registro*, con il proprio segnale β ; nel caso che gli ingressi possibili IN siano in numero $r > 1$, sull'ingresso del registro viene inserito un *commutatore*, comandato da $\lceil \lg_2 r \rceil$ segnali di controllo α ed i cui ingressi principali sono quelli di IN. Nell'esempio precedente :

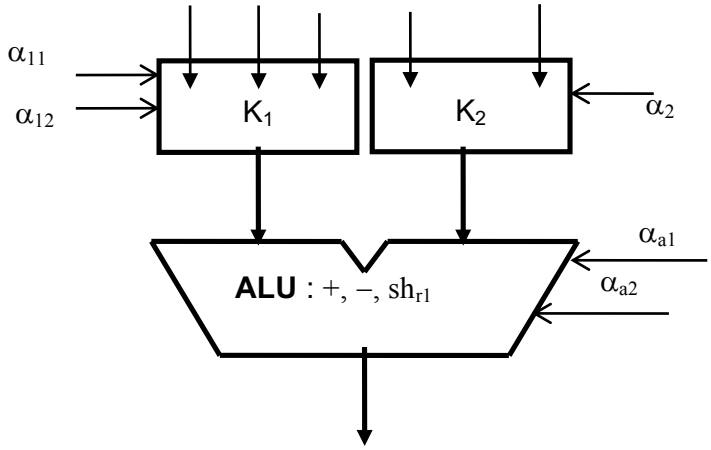


b1. Si individuano tante *classi di operazioni elementari* per quante sono le *reti logiche* dagli operatori che compaiono nelle operazioni elementari. Ciò permette di individuare le possibili sorgenti di ingresso di ogni rete logica. Ad esempio se :

$$\text{classe di una ALU : } A + B \rightarrow A \text{ ; } A - B \rightarrow A \text{ ; } M - N \rightarrow M \text{ ; } s_{r1}(N) \rightarrow N$$

le possibili sorgenti di ingresso sono A, M, N a sinistra, e B, N a destra ;

b2. si ricava *una sottostruttura indipendente per ogni rete logica* ; se si tratta di una rete *multifunzione*, come una ALU, con r funzioni, occorrono $\lceil \lg_2 r \rceil$ segnali di controllo α per comandare la scelta della funzione ; per ogni ingresso principale, se è prevista più di una sorgente, si inserisce un commutatore pilotato da opportuni segnali α. Nell'esempio precedente :



Nel caso che le microoperazioni non siano parallele, di regola è sufficiente realizzare tutte le funzioni di calcolo del microprogramma mediante una sola ALU multifunzione. Nel caso di microoperazioni parallele (sez. 6) in generale occorrono più reti di calcolo sotto forma di una o più ALU e/o reti dedicate a specifici operatori (shift, incremento, decremento, ecc).

c1. Si implementano le *variabili di condizionamento*, come funzioni di uscite di registri o come Flags di ALU memorizzati in registri di 1 bit.

Quesito

Lo studente spieghi perché nel procedimento sopra spiegato *non* viene fatto uso di *selezionatori*. In effetti, nella sintesi della PO non è necessario fare uso di selezionatori fino a che non progetteremo unità molto complesse dal punto di vista della logica interna (sez. 7, parte 2); in tali casi essi verranno introdotti solo per ragioni di ottimizzazione del costo della PO.

3.3 Struttura della PC

La struttura della PC, vista come *rete sequenziale di Mealy*, viene ricavata formalmente dal microprogramma mediante il seguente procedimento.

1. Gli *stati interni* corrispondono biunivocamente alle etichette delle microistruzioni del microprogramma. Se m è il numero degli stati interni, il registro di stato del controllo (RC) è di s bit con $s = \lceil \lg_2 m \rceil$.
2. Gli *stati di ingresso* corrispondono biunivocamente alle possibili combinazioni di variabili di condizionamento {x} per formare le condizioni logiche del microprogramma.

3. Gli *stati di uscita* corrispondono biunivocamente alle possibili combinazioni di segnali di controllo $\gamma = \{\alpha\} \cup \{\beta\}$ necessari ad eseguire tutte le microoperazioni del microprogramma.

4. La tabella di verità della *funzione delle uscite* ω_{PC} e *della funzione di transizione dello stato interno* σ_{PC} viene ricavata in base alle tre corrispondenze suddette ed alla struttura del microprogramma. Da questa tabella è possibile sintetizzare PC con parte combinatoria a due livelli di logica (o come PC microprogrammata, vedi sez. 10).

Si osservi che *tutti i segnali β devono sempre essere specificati per ogni possibile combinazione (stato d'ingresso, stato interno),* mentre i segnali α potranno eventualmente essere non specificati per qualche combinazione (stato d'ingresso, stato interno).

3.4 Ciclo di clock

A differenza di una rete sequenziale isolata, per una unità di elaborazione PC-PO occorre ricavare la *condizione di stabilizzazione di entrambe le reti sequenziali PC e PO*. Il ciclo di clock non può dunque concludersi prima che siano stabili le funzioni di transizione della PC e della PO, cioè prima che siano stabili, rispettivamente, gli ingressi del registro di stato del controllo RC e di tutti i registri della PO.

In fig. 1.8 è mostrato come ricavare la lunghezza del ciclo di clock attraverso una sequenza di eventi temporali. Le frecce indicano eventi "stabilizzazione di funzioni", gli intervalli temporali i ritardi massimi di stabilizzazione.

Poiché la PO è una rete di Moore, la sua uscita è disponibile all'inizio del ciclo di clock ; dopo un tempo $T_{\omega PO}$ sono quindi stabili le variabili di condizionamento. Solo a questo punto, poiché la PC è una rete di Mealy, può iniziare la stabilizzazione della funzione ω_{PC} e della funzione σ_{PC} in parallelo (nella figura si è assunto che la seconda abbia un ritardo maggiore della prima, ma ciò non è influente nella valutazione complessiva). Una volta che ω_{PC} si è stabilizzata, dopo un intervallo TωPC, può quindi iniziare la stabilizzazione della funzione σ_{PO} ; il ciclo di clock si può concludere quando si sono stabilizzate tanto la σ_{PC} quanto la σ_{PO}.

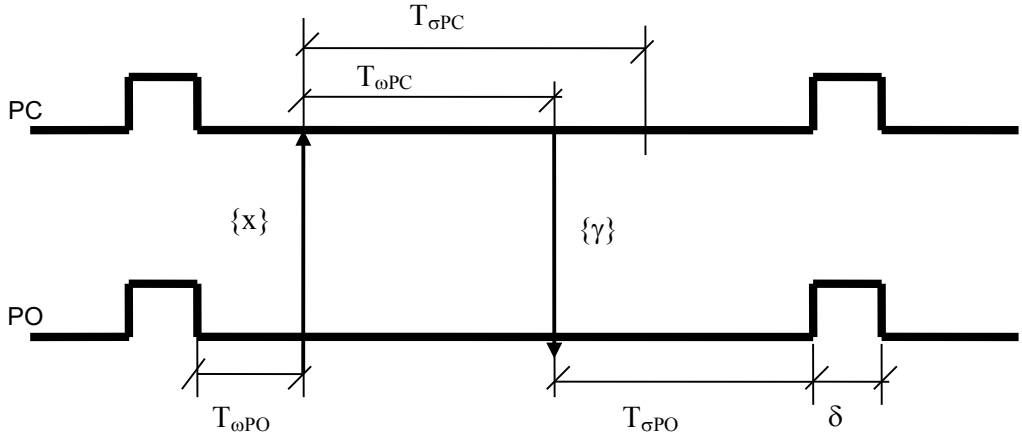


Fig. 1.8 - Procedimento grafico per ricavare la lunghezza del ciclo di clock

Di conseguenza, la lunghezza del ciclo di clock è data da :

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

con δ ampiezza dell'impulso di clock. Nella formula di τ, spesso è il termine

$$T_{\omega PC} + T_{\sigma PO}$$

a predominare, fornendo una buona approssimazione del valore di τ.

3.5 Tempo medio di elaborazione

Il tempo medio di elaborazione di una unità viene calcolato come

$$T = k\,\tau$$

dove *k* è il *numero medio di cicli di clock necessari ad eseguire la generica operazione del microprogramma*. *A partire da quando sono presenti i (nuovi) dati in ingresso*, si valuta la durata media del corpo del costrutto più esterno **repeat ... forever**.

In generale, in un microprogramma sono presenti più sottosequenze di microistruzioni eseguite con una certa probabilità p_i, dove

$$\Sigma_{(i = 0...n-1)}\, p_i = 1$$

Di ogni sottosequenza si calcola il numero di cicli di clock k_i necessario ed eseguirla e quindi si ricava il valore di T come media pesata

$$T = \tau * \Sigma_{(i = 0...n-1)}\, p_i * k_i$$

Quando non siano note le p_i, si assume che tutte le sottosequenze siano equiprobabili, calcolando perciò T come media aritmetica dei k_i.

La *banda di elaborazione* è definita *come il numero medio di operazioni esterne che l'unità può eseguire nell'unità di tempo*, nella situazione in cui, all'inizio di ogni iterazione, sia già presente un nuovo insieme di dati d'ingresso (unità *saturata*, cioè "sollecitata" al massimo). Espressa come numero di operazioni esterne al secondo (*ops*), la banda si calcola come :

$$B = 1 / T$$

3.6 Esempio 2

Come ulteriore esempio, consideriamo una unità U così definita :

a) ha un registro interno A visibile anche al livello superiore, ed un ingresso B dello stesso numero *N* di bit ;

b) esegue l'operazione esterna : conta in A il numero di "1" presenti nella configurazione binaria di B.

Nota : questa funzionalità potrebbe essere realizzata direttamente come una singola rete combinatoria. La realizzazione come unità PC-PO è certamente più lenta, ma altrettanto certamente assai meno complessa; in particolare la complessità di progettazione dell'unità è indipendente da *N*. Questa interessante dicotomia è frequente nei sistemi di elaborazione, e deve essere di volta in volta valutata in funzione del compromesso prestazioni-costo.

Microprogramma

La versione ad alto livello può essere la seguente :

```
repeat
        M := B ;
        A := 0 ;
        for i := 1 to 32 do
        begin
                M := sh_r1 (M) ;        {traslazione destra logica di M di una posizione}
                if SH = 1 then A := A +1        {SH è un registro che memorizza il bit espulso nella traslazione}
        end
forever
```

Una prima versione eseguibile è riportata di seguito.

0. $B \rightarrow M, 1$
1. $0 \rightarrow A, 2$

$31 \rightarrow I, 3$ *{I è un registro di 6 bit)*

2. $sh_{r1}\ (M) \rightarrow M, 4$
3. $(SH = 1)\ A + 1 \rightarrow A, 5\ ;\ (= 0)$ nop, 5
4. $I - 1 \rightarrow I, 6$
5. $(I_0 = 0)\ sh_{r1}\ (M) \rightarrow M, 4\ ;\ (= 1)$ nop, 0 *{test di I per la condizione > 0}*

Occorre comunque tener presente che tale versione non è ottimizzata : la versioni ottimizzata sarà vista nelle sez. 4, 5, 6.

Si noti che la prima frase della microistruzione 6 poteva anche essere espressa come “nop, 3”. In questo modo, però, avremmo perduto un ciclo di clock ad ogni iterazione del *for*, quindi complessivamente avremmo perduto 32 cicli di clock. Quella qui introdotta è una prima ottimizzazione, che generalizzeremo nella sez. 4.

Struttura della PO

Le classi per individuare le risorse della PO sono :

registro M : $B \rightarrow M$, $sh_{r1}(M) \rightarrow M$

registro A : $0 \rightarrow A$, $A + 1 \rightarrow A$

registro I : $31 \rightarrow I$, $I - 1 \rightarrow I$

ALU : funzioni : sh_{r1}, +1, −1 ; primo ingresso : M, A, I ; secondo ingresso : non significativo (tutti gli operatori usati sono ad un solo operando).

Ne consegue la struttura di Fig. 1.9.

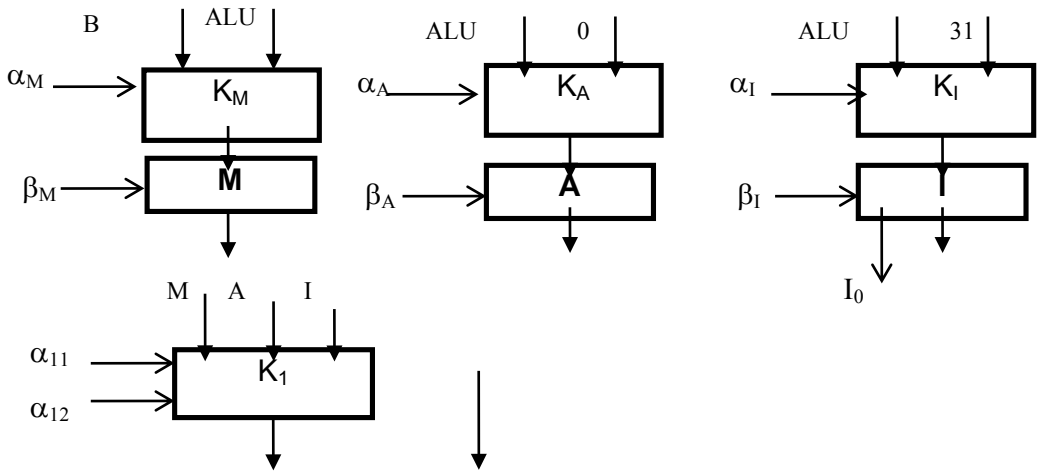
Struttura della PC

La PC ha 7 stati interni ; il registro di stato del controllo RC è quindi di 3 bit. Indichiamo con y_0, y_1, y_2 le variabili dello stato interno presente (uscite di RC), e con Y_0, Y_1, Y_2 quelle dello stato interno successivo (ingressi di RC).

Le variabili d’ingresso sono SH ed I_0.

Le variabili di uscita sono : α_M, α_A, α_I, α_{11}, α_{12}, α_{a1}, α_{a2}, β_M, β_A, β_I.

La tabella di verità delle funzioni ω_{PC} e σ_{PC} è data dalla Tab. 1. 2. La sintesi delle funzioni ω_{PC} e σ_{PC} deriva direttamente.



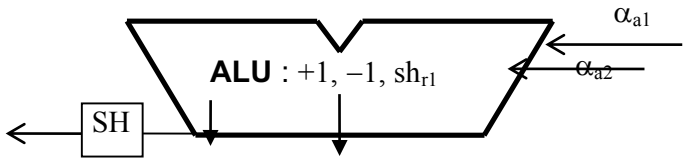


Fig. 1.9 - PO dell'esempio 2.

y_0	y_1	y_2	SH	I_0	α_M	α_A	α_I	α_{11}	α_{12}	α_{a1}	α_{a2}	β_M	β_A	β_I	Y_0	Y_1	Y_2
0	0	0	-	-	0	-	-	-	-	-	-	1	0	0	0	0	1
0	0	1	-	-	-	1	-	-	-	-	-	0	1	0	0	1	0
0	1	0	-	-	-	-	1	-	-	-	-	0	0	1	0	1	1
0	1	1	-	-	1	-	-	0	0	0	0	1	0	0	1	0	0
1	0	0	0	-	-	-	-	-	-	-	-	0	0	0	1	0	1
1	0	0	1	-	-	0	-	0	1	0	1	0	1	0	1	0	1
1	0	1	-	-	-	-	0	1	-	1	-	0	0	1	1	1	0
1	1	0	-	0	1	-	-	0	0	0	0	1	0	0	1	0	0
1	1	0	-	1	-	-	-	-	-	-	-	0	0	0	0	0	0

Tab. 1. 2 - Tabella di verità per la PC dell'esempio 2.

Ciclo di clock

Assumiamo i ritardi delle risorse di cui alla sez. 2.5. Un commutatore a due livelli di logica ha quindi un ritardo di 2 nsec

Supponiamo che le funzioni ω_{PC} e σ_{PC} abbiano lo stesso ritardo, per entrambi quello di una rete a due livelli di logica, quindi 2 nsec.

Il massimo ritardo delle operazioni elementari si incontra ogni volta che è necessario "attraversare" (vedi fig. 1.9) K1, ALU, ed uno dei commutatori di ingresso dei registri M, A, I. Tale ritardo è quindi 9 nsec.

Dunque :

$$\tau = 0 + \max(2 + 9, 2) + 1 = 12 \text{ nsec}$$

$$f = 1/\tau = 83.3 \text{ MHz}$$

Tempo medio e banda di elaborazione

La fig. 1.10 mostra, mediante un grafo, l'evoluzione del microprogramma attraverso i possibili cammini. I nodi corrispondono a microistruzioni, gli archi a transizioni tra microistruzioni. È indicato il numero delle ripetizioni dei cicli iterativi. L'arco tratteggiato non entra nella valutazione, in quanto ha il significato di rieseguire l'operazione esterna su un nuovo dato.

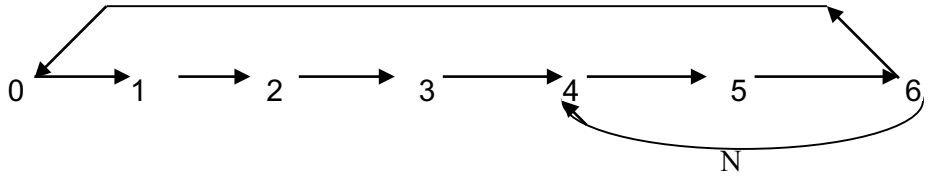


Fig. 1.10 - Grafo per la valutazione del tempo medio di elaborazione dell'esempio 2.

Il numero di cicli di clock è dato da :

$$k = 4 + 3\,N$$

I primi 4 cicli sono spesi nell'inizializzazione (microistruzioni 0, 1, 2, 3), i successivi 3N corrispondono al loop delle tre microistruzioni 4, 5, 6 ripetute N volte.

Per N = 32, k = 100, e quindi T = 100 τ = 1200 nsec = 1.2 μsec. La banda vale B = 1 /T = 0.833 Mops.

Il caso limite asintotico alle prestazioni consisterebbe nel realizzare la stessa funzionalità con una rete combinatoria a due livelli di logica, invece che con una unità PC- PO : con la stessa tecnologia hardware potremmo pensare di ottenere T = 2 nsec, B = 500 Mops. Poiché tale soluzione è praticamente irrealizzabile, si pone il problema se sia possibile ottenere un migliore compromesso tra prestazioni e costo, o meglio progettare una unità PC - PO, che è caratterizzata sempre da un procedimento di progettazione di complessità polinomiale, con prestazioni decisamente migliori.

3.7 Esempio 3

Progettiamo ora una unità U capace di svolgere entrambe le funzioni dell'esempio 1 (sez. 2) e dell'esempio 2 (sez. 3.6) :

a) ha un registro interno A visibile anche al livello superiore, ed un ingresso B dello stesso numero *N* di bit ;

b) un ulteriore ingresso di 1 bit, COP, fornisce il *codice operativo* della funzione da eseguire ;

c) se COP = 0, U esegue l'operazione esterna *abs* (A – B) ; se COP = 1, U conta in A il numero di "1" presenti nella configurazione binaria di B.

Si tratta del primo esempio di *unità multi-operazione*.

Vediamo la traccia del microprogramma ; in versione ad alto livello :

repeat

if COP = 0 **then** < *esegui abs* > **else** < *esegui conta "1"* >

forever

dove il predicato su COP esprime la fase di *decodifica* dell'operazione esterna, e le due sezioni <*esegui abs*> e <*esegui conta "1"*>, definite come in sez. 2.2 e 3.6, rappresentano la fase di *esecuzione vera e propria*.

Il microprogramma eseguibile potrebbe iniziare così :

0. (COP = 0) nop, 1 ; (= 1) nop, 3
1. inizio abs
...
3. inizio conta "1"
...

o meglio, con una prima ottimizzazione :

0. (COP = 0) A + B $\rightarrow$ A, 1; (= 1) B $\rightarrow$ M, 2
1. prosegue abs
...
3. prosegue "conta "1"
...

Si lascia come esercizio il completamento della sintesi dell'unità.

3.8 Esercizi

Esercizio 1

Progettare una unità di elaborazione capace di eseguire la moltiplicazione di due numeri interi rappresentati in complemento a due, N1 e N2, ognuno di 32 bit; N1 è il moltiplicando e N2 il moltiplicatore. Il risultato della moltiplicazione va memorizzato in una coppia di registri, A e Q, ognuno di 32 bit, dove A rappresenta la metà più significativa e Q quella meno significativa del risultato stesso.

Gli ingressi esterni dell'unità sono N1 e N2, le uscite esterne sono quelle di A e Q.

L'algoritmo da adottare è il seguente :

repeat

```
M := N1 ;          {moltiplicando memorizzato nel registro M}
Q := N2 ;          {moltiplicatore memorizzato nel registro Q}
A := 0 ;           {inizializzazione della metà più significativa del risultato nel registro A}
case M0, Q0 of
      00 : H:= 0 ;
      01 : H := 1 ; Q := - Q ;
      10 : H := 1 ; M := - M ;
      11 : H := 0 ; M := - M ; Q := - Q
```

endcase ; {il registro H, di 1 bit, serve a ricordare il segno da dare al risultato alla fine dell'algoritmo; inizializzazione della metà meno significativa del risultato (Q) per applicare l'algoritmo su numeri positivi}

{il seguente comando **for** esprime l'algoritmo di moltiplicazione su numeri interi positivi}

for i := 1 **to** 32 **do**

begin

if Q_{31} = 1 **then** A := A + M ;

(A, Q) := $sh_R^{(1)}$ (A, Q) {i registri A e Q sono considerati, agli effetti dell'operazione di shift (destro di 1 posizione) come un unico registro a 64 bit; si suppone che, nella PO, questo sia possibile per tutte le operazioni monadiche (shift, negazione, ecc.)}

end ;

if H = 1 **then** (A, Q) := - (A, Q) {aggiustamento del segno del risultato su doppia parola}

forever

Per l'implementazione della Parte Operativa si dispone di una ALU con doppia uscita, *alu1* e *alu2* entrambe a 32 bit. Questa ALU è capace di eseguire le operazioni monadiche (shift, negazione, ecc.) sia su parola semplice, che su una doppia parola costituita dalla concatenazione dei due ingressi della ALU stessa, producendo il risultato sulla doppia uscita.

Esercizio 2

I seguenti microprogrammi rappresentano due implementazioni diverse della stessa funzione:

microprogramma 1::

0. $B \rightarrow A, 1$
1. $sh_L^{(1)}(A) \rightarrow C, 2$
2. $sh_R^{(1)}(A) \rightarrow A, 3$
3. $A + C \rightarrow A, 4$
4. $A + D \rightarrow A, 0$

microprogramma 2::

0. $B \rightarrow A, 1$
1. $sh_L^{(1)}(A) \rightarrow C, 2$
2. $sh_R^{(1)}(A) \rightarrow A, 3$
3. $A + C + D \rightarrow A, 0$

In entrambi i casi: *a)* mostrare lo schema della Parte Operativa, *b)* calcolare la lunghezza (in *nsec*) del ciclo di clock assumendo che il ritardo della Parte Controllo sia (in entrambi i casi) di 6 nsec, quello di un qualunque commutatore di 2 nsec, e quello di una ALU di 10 nsec (trascurare il ritardo dei collegamenti).

Confrontare quindi le due implementazioni dal punto di vista del tempo complessivo di elaborazione e, possibilmente, trarre delle conseguenze di portata generale.

Esercizio 3

Si consideri il seguente microprogramma, dove S indica il segno del risultato calcolato dalla ALU :

0. $A + B \rightarrow A, 1$
1. $(S = 0)\ sh_{L1}\ (A) \rightarrow A, 0\ ;\ (S = 1)\ sh_{R1}\ (A) \rightarrow A, 0$

a) Spiegare perché la variabile S deve essere memorizzata in un registro.

b) Dimostrare la seguente asserzione : "per la correttezza di funzionamento, *non occorre* utilizzare una variabile di controllo β per abilitare la scrittura in S ; in altri termini, il funzionamento è corretto anche se la scrittura in S è abilitata ad ogni impulso di clock".

c) Si consideri ora la seguente modifica al microprogramma :

0. $A + B \rightarrow A, 1$
1. $A \rightarrow D, 2$
2. $(S = 0)\ sh_{L}^{(1)}\ (A) \rightarrow A, 0\ ;\ (S = 1)\ sh_{R}^{(1)}\ (A) \rightarrow A, 0$

Il significato del frammento vuole essere, analogamente come nel caso precedente, quello di effettuare modifiche diverse di A a seconda del valore assunto dal segno dell'operazione A + B nella microistruzione 0.

Dimostrare la seguente asserzione : "per la correttezza di funzionamento, *è necessario* utilizzare una variabile di controllo β per abilitare la scrittura in S".

Esercizio 4

Modificare l'unità dell'Esercizio 1 nel modo seguente. Oltre ad N1, N2 (definiti come nell'Esercizio 1), l'unità ha un ulteriore ingresso di 1 bit, COP (codice operativo). Se COP = 0 deve essere calcolato in A il risultato della somma di N1 ed N2, se COP = 1 deve essere calcolato in (A, Q) il prodotto di N1 ed N2.

Nel caso che dal microprogramma derivi una struttura della Parte Operativa diversa da quella dell'Esercizio 1, scrivere anche una seconda versione del microprogramma cui corrisponda la *stessa* Parte Operativa dell'Esercizio 1. Far rilevare le differenze tra le due implementazioni.

Esercizio 5

Progettare una unità di elaborazione avente una uscita A e due ingressi : COP, di 1 bit, e *N*, numero naturale di *k* bit (k > 1).

Se COP = 0 in A viene calcolato il valore dell'*N-esimo* numero della serie di Fibonacci, se COP = 1 la somma dei primi *N* numeri naturali.

Valutare il tempo medio di elaborazione dell'unità, supponendo che le due operazioni esterne siano equiprobabili.

Esercizio 6

Si rifletta sugli esercizi finora svolti. Esistono casi in cui la stessa funzionalità poteva essere realizzata con una rete combinatoria invece che con una unità (PC, PO)?

Nei casi in cui ciò sia concettualmente possibile, studiare quali differenze esistono tra i due approcci in termini di *complessità* del procedimento di progettazione, di *costo* della struttura, di *velocità di esecuzione*, o di altri eventuali parametri di interesse.

4. Ottimizzazione dei microprogrammi : eliminazione delle *nop*

È evidente, dai microprogrammi finora scritti, che, ogni volta che si esegue una *nop,* agli effetti del tempo di elaborazione viene perduto un ciclo di clock.

Ci rendiamo però subito conto che, *scrivendo i microprogrammi con il microlinguaggio PS, è possibile eliminare tutte le nop,* tranne che nell'implementazione dei meccanismi di sincronizzazione che vedremo nella sez. 9. La tecnica di ottimizzazione, a partire dal microprogramma eseguibile in cui compaiono delle *nop*, è il seguente : ogni frase del tipo

$$F : \text{nop}, j$$

con

$$j.\ \mu op_j, h$$

viene trasformata in

$$F : \mu op_j, h$$

Nel caso che j sia una microistruzione condizionale (contenente più di una frase), le stesse condizioni sono replicate nella microistruzione in cui compare F congiungendole a quelle eventualmente già presenti.

Esempio 1a

Il microprogramma dell'esempio 1 (sez. 2) diviene :

0. $A + B \rightarrow A$, 1
1. $(A_0 = 1) - A \rightarrow A$, 0 ; $(A_0 = 0)$ $A + B \rightarrow A$, 1.

Nel caso che il risultato della somma sia positivo, è possibile iniziare la prossima elaborazione a distanza di tempo di 1 τ (purché il nuovo valore di B sia già presente : condizione di "saturazione" dell'unità). Assumendo equiprobabili gli eventi $(A_0 = 1)$ ed $(A_0 = 0)$, sia ha ora :

$$T = 1.5\ \tau = 12 \text{ nsec} , B = 83{,}3 \text{ Mops}$$

con una guadagno in prestazioni del 25%.

Esempio 2a

Il microprogramma dell'esempio 2 della sez. 3.6 diviene :

0. $B \rightarrow M$, 1
1. $0 \rightarrow A$, 2
2. $31 \rightarrow I$, 3
3. sh_{r1} (M) $\rightarrow M$, 4
4. $(SH = 1)$ $A + 1 \rightarrow A$, 5 ; $(= 0)$ $I - 1 \rightarrow I$, 6
5. $I - 1 \rightarrow I$, 6
6. $(I_0 = 0)$ sh_{r1} (M) $\rightarrow M$, 4 ; $(= 1)$ $B \rightarrow M$, 1

La fig. 1.11 mostra il grafo per la valutazione del tempo medio di elaborazione.

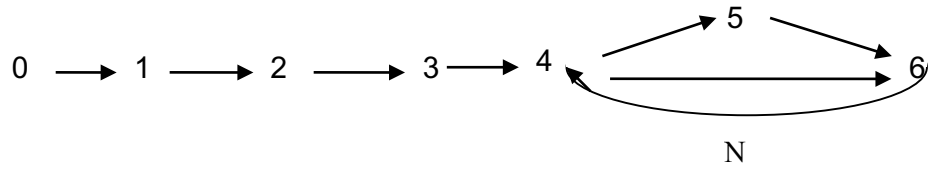


Fig. 1.11 - Grafo per la valutazione del tempo medio di elaborazione dell'esempio 2a.

Si ha ora :

$$k = 3 + 2.5 \times N$$

dove per l'inizializzazione si è trascurata la microistruzione 0 (eseguita solo all'atto dell'accensione della macchina) ; per ognuna della N iterazioni del loop si eseguono le tre microistruzioni 4, 5, 6 *oppure* le due microistruzioni 4, 6 : in media 2.5 cicli di clock per iterazione. Per N = 32, si ha k = 83, con un guadagno percentuale del 17 %.

In generale, facciamo notare che, *per applicare questa tecnica di ottimizzazione, non è necessario partire dalla versione del microprogramma con nop* : in tutte le occasioni in cui, nella scrittura del microprogramma, *verrebbe* introdotta una fase con *nop*, questa deve essere *direttamente* sostituita con la frase da eseguire successivamente.

5. Ottimizzazione dei microprogrammi : parallelismo nelle condizioni logiche

Ricordiamo che, nel caso più generale, il microlinguaggio PS permette di esprimere, in ogni microistruzione, strutture condizionali di tipo *case* che, rispetto alle più semplici strutture *if then else*, permettono un sensibile risparmio di cicli di clock.

Una sequenza di N *microistruzioni, in ognuna delle quali venga testata una sola variabile di condizionamento, può essere trasformata in una singola microistruzione costituita da un* case *ad* N *vie.*

Ad esempio, una computazione come :

```
repeat
    A := A + B ;
    if OV = 1 then A := A – B else if A < 0 then A := – A
forever
```

dove OV indica la presenza di traboccamento nell'operazione di addizione, viene scritta come :

0. $A + B \rightarrow A, 1$

1. $(OV\ A_0 = 0\ 1) - A \rightarrow A\ ,\ 0\ ;\ (=\ 0\ 0)\ A + B \rightarrow A,\ 1\ ;\ (=\ 1\ -)\ A - B \rightarrow A,\ 0.$

Per il momento, questa tecnica è vista solo come un ulteriore modo per eliminare delle *nop*. Essa è inoltre della massima utilità per poter applicare al meglio la tecnica della parallelizzazione delle microoperazioni, che presenta un grande interesse per l'ottimizzazione dei microprogrammi.

6. Ottimizzazione dei microprogrammi : parallelismo nelle microoperazioni

6.1 Condizioni per la parallelizzazione

Nel caso più generale una microoperazione consta *di più operazioni elementari eseguite in parallelo* nello stesso ciclo di clock.

Un esempio di microoperazione parallela è il seguente :

$$A + B \rightarrow A,\ C + 1 \rightarrow C$$

Le operazioni elementari $A + B \rightarrow A$ e $C + 1 \rightarrow C$ rappresentano computazioni completamente *indipendenti*, cioè l'esecuzione dei nessuna delle due dipende dal risultato dall'esecuzione dall'altra. Esse possono dunque essere eseguite in qualsiasi ordine ed, in particolare, contemporaneamente. Questo presuppone che *la PO contenga risorse sufficienti* all'esecuzione contemporanea delle due operazioni elementari, cioè che, con riferimento all'esempio precedente, abbia una struttura del tipo di fig. 1.12, in cui le operazioni

di somma e di incremento sono realizzate da due *distinte reti di calcolo*. Se invece avessimo forzato le due operazioni ad essere eseguite da una stessa ALU, il parallelismo sarebbe stato impossibile.

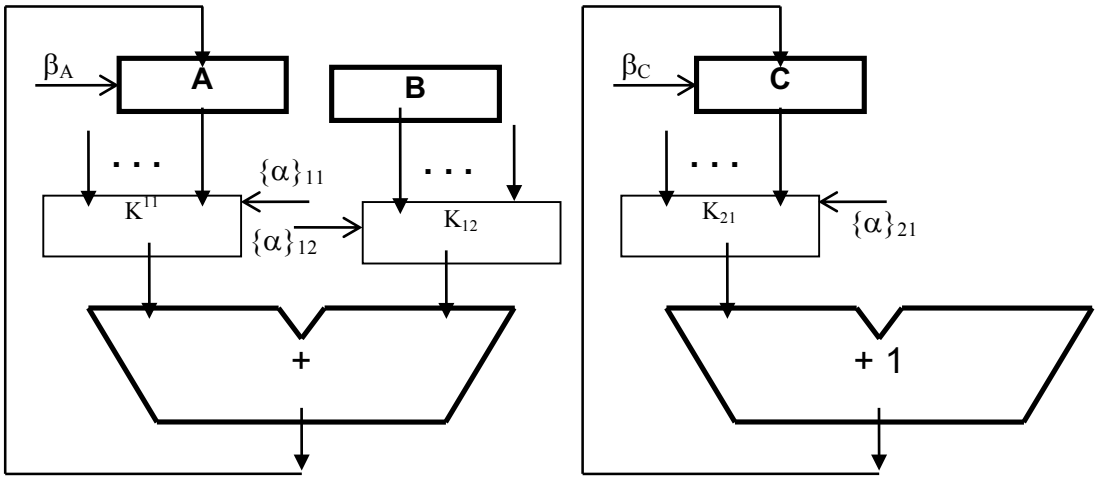


Fig. 1.12 - Risorse di PO per eseguire in parallelo A + B → A e C + 1 → C

Formalmente, l'esecuzione in parallelo di due o più operazioni elementari ha *luogo durante l'intervallo* $T_{\sigma PO}$ *del ciclo di clock* : durante tale intervallo *le reti combinatore della PO utilizzate dalle operazioni elementari eseguite in parallelo si stabilizzano contemporaneamente*.

Nella trasformazione da una computazione espressa in sequenziale alla stessa computazione espressa in parallelo, occorre ovviamente rispettare precise condizioni affinchè le due computazioni siano **equivalenti** : cioè, *per uno stesso stato interno iniziale della PO, le due computazioni portino ad uno stesso stato interno finale della PO*.

Per ricavare le **condizioni per trasformare una computazione sequenziale nella computazione parallela equivalente**, consideriamo prima i seguenti esempi, nel quali si indica con " ;" l'operatore di sequenziamento, e con "," l'operatore di esecuzione in parallelo. In tutti gli esempi supporremo che *la PO contenga sufficienti risorse* alla eventuale esecuzione in parallelo di operazioni elementari.

(1) $A + B \rightarrow A \; ; \; C \rightarrow D$ *è equivalente a* $A + B \rightarrow A \; , \; C \rightarrow D$

(2) $A + B \rightarrow A \; ; \; C \rightarrow A$ *non è equivalente a* $A + B \rightarrow A \; , \; C \rightarrow A$

il risultato sarebbe diverso, e comunque impredicibile ; ovviamente, nessuna struttura di PO può permettere la scrittura nello stesso registro di due risultati diversi nello stesso ciclo di clock ;

(3) $A + B \rightarrow A \; ; \; B \rightarrow D$ *è equivalente a* $A + B \rightarrow A \; , \; B \rightarrow D$

infatti l'uscita di uno stesso registro può essere utilizzata contemporaneamente da più operazioni (si ricordi il significato di segnali a livelli) ;

(4) $A + B \rightarrow A \; ; \; B + C \rightarrow D$ *è equivalente a* $A + B \rightarrow A \; , \; B + C \rightarrow D$

come sopra, e purché la PO disponga di due reti di calcolo indipendenti ;

(5) $A + B \rightarrow A \; ; \; C \rightarrow B$ *è equivalente a* $A + B \rightarrow A \; , \; C \rightarrow B$

infatti*, il contenuto di B, a cui si riferisce la* $A + B \rightarrow A$*, rimane stabile durante tutto il ciclo di clock* in quanto, durante l'esecuzione di $C \rightarrow B$, varia solo l'ingresso di B; il nuovo valore verrà scritto in B all'inizio del successivo ciclo di clock ;

(6) $A + B \rightarrow A \; ; \; A \rightarrow B$ *non è equivalente a* $A + B \rightarrow A \; , \; A \rightarrow B$

infatti, nella computazione sequenziale il valore di A in $A \rightarrow B$ è il risultato di $A + B \rightarrow A$, mentre nella computazione parallela il valore di A, durante il ciclo di clock, è lo stesso per le due operazioni elementari

(7) $A \rightarrow TEMP \; ; \; B \rightarrow A \; ; \; TEMP \rightarrow B$ *è equivalente a* $A \rightarrow TEMP \; , \; B \rightarrow A \; ; \; TEMP \rightarrow B$

Formalmente, valgono le seguenti **condizioni di Bernstein** per la trasformazione di equivalenza da una computazione sequenziale ad una parallela :

data la computazione sequenziale

$$f_1 : D_1 \rightarrow R_1 \; ; \; f_2 : D_2 \rightarrow R_2$$

dove f_i *sono funzioni di dominio* D_i *e rango* R_i*, la computazione parallela*

$$f_1 : D_1 \rightarrow R_1, \; f_2 : D_2 \rightarrow R_2$$

è equivalente se :

$$R_1 \cap D_2 = \varnothing \quad \text{e} \quad R_1 \cap R_2 = \varnothing$$

Inoltre, in un modello *asincrono* di computazione, nel quale non si fanno ipotesi sulla durata temporale delle operazioni e sui loro istanti di inizio, deve valere anche la terza condizione :

$$R_2 \cap D_1 = \varnothing$$

Questo *NON* è nel caso della microprogrammazione che utilizza un modello *sincrono* : di ogni operazione è noto il tempo di esecuzione (lunghezza del ciclo di clock) e l'istante di inizio. Si veda allo scopo l'esempio (5).

Occorre ribadire che le condizioni di Bernstein vanno utilizzate *per trasformare* una descrizione sequenziale del microprogramma in una descrizione parallela. Spesso, nella scrittura dei microprogrammi, è possibile "saltare" la descrizione sequenziale e relativa trasformazione, e *scrivere direttamente le microoperazioni in parallelo*.

Ad esempio ("swap di due variabili") :

(8) $A \rightarrow TEMP \; ; \; B \rightarrow A \; ; \; TEMP \rightarrow B$ *è equivalente a* $A \rightarrow B \; , \; B \rightarrow A$

questo non è un caso di trasformazione formale, bensì di una ulteriore *ottimizzazione* che permette di risparmiare cicli di clock e risorse hardware. In effetti questo caso, così come il caso (5), si

presenta molto spesso quando si scrivono *direttamente* le microistruzioni in parallelo, senza passare attraverso una fase di trasformazione a partire da una computazione sequenziale.

6.2 Esempio 2b

Consideriamo di nuovo il microprogramma per il conteggio degli "1" di B, come ricavato nella sez. 4 (esempio 2a). Una prima parallelizzazione riguarda la fase di inizializzazione :

0. $B \rightarrow M, 0 \rightarrow A, 31 \rightarrow I, 1$
1. $sh_{r1}\ (M) \rightarrow M, 2$
2. $(SH = 1)\ A + 1 \rightarrow A, 3\ ;\ (= 0)\ I - 1 \rightarrow I, 4$
3. $I - 1 \rightarrow I, 4$
4. $(I_0 = 0)\ sh_{r1}\ (M) \rightarrow M, 2\ ;\ (= 1)\ B \rightarrow M, 0 \rightarrow A, 31 \rightarrow I, 1$

Ben più significativa è la seguente ottimizzazione, che *utilizza anche la parallelizzazione delle condizioni logiche* nel caso più generale :

0. $B \rightarrow M, 0 \rightarrow A, 31 \rightarrow I, 1$
1. $sh_{r1}\ (M) \rightarrow M, I - 1 \rightarrow I, 2$
2. $(SH\ I_0 = 0\ 0)\ sh_{r1}\ (M) \rightarrow M, I - 1 \rightarrow I, 2\ ;$
   $(= 0\ 1)\ B \rightarrow M, 0 \rightarrow A, 31 \rightarrow I, 1\ ;$
   $(= 1\ 0)\ A + 1 \rightarrow A, sh_{r1}\ (M) \rightarrow M, I - 1 \rightarrow I, 2\ ;$
   $(= 11)\ A + 1 \rightarrow A, 0$

Il risultato è di passare ad un valore del numero medio di cicli di clock k = 1 + N ; per N = 32, si ha k = 33 con un guadagno relativo del 67% rispetto al caso iniziale (esempio 2, sez. 3.6).

In fig. 1.13 è mostrata la versione finale della PO dell'unità per il conteggio di "1" in B :

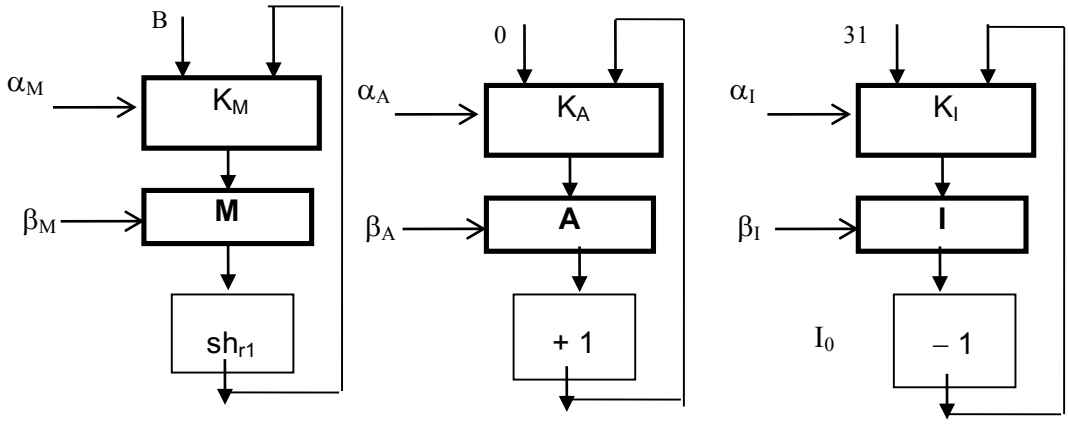


Fig. 1.13 - PO dell'esempio 2b.

Si è fatto uso di *reti di calcolo ad un solo operando*, rispettivamente per il calcolo dello *shift* di M, dell'*incremento* di A e del *decremento* di I. È interessante notare che la parallelizzazione, se da una parte ha aumentato la complessità hardware dell'unità sostituendo una singola ALU con tre reti di calcolo distinte, dall'altra parte ha ridotto tale complessità in conseguenza della riduzione nel numero di segnali α e di stati interni della PC.

Nella Tab. 1.3 è mostrata la tabella di verità della PC dell'ultima versione dell'esempio :

y_0	y_1	SH	I_0	α_M	α_A	α_I	β_M	β_A	β_I	Y_0	Y_1
0	0	-	-	0	1	1	1	1	1	0	1
0	1	-	-	1	-	0	1	0	1	1	-
1	-	0	0	-	-	0	1	0	1	1	-
1	-	0	1	0	1	1	1	1	1	0	1
1	-	1	0	1	0	0	1	1	1	1	-
1	-	1	1	-	0	-	0	1	0	0	0

Tab. 1.3 - Tabella di verità per la PC dell'esempio 2b.

6.3 Parallelismo nelle microoperazioni e nelle condizioni logiche

Partendo dalla descrizione sequenziale, o dal microprogramma ad alto livello, la tecnica ora esemplificata consiste nello *spostare* operazioni di trasferimento tra registri, rispetto all'ordinamento iniziale, in modo

a) da sostituire condizioni logiche di tipo *if then else* con condizioni logiche di tipo *case* ; per generalizzazione, in modo da sostituire condizioni logiche di tipo *case* con altre condizioni logiche di tipo *case* aventi un numero maggiore di variabili di condizionamento ;

b) comunque da aumentare il grado di parallelismo di microoperazioni.

Entrambe le tecniche a), b) per essere applicate devono comunque preservare la semantica della computazione sequenziale, e quindi rispettare le *condizioni di Bernstein* per i microprogrammi.

Ad esempio, nell'ultima versione del conteggio di "1" in B (esempio 2b, sez. 6.2), si è applicata la tecnica a) *anticipando* l'operazione "$I - 1 \rightarrow I$" rispetto al test su SH, in modo da poter testare contemporaneamente SH ed I_0.

Un esempio della tecnica b) è il seguente : la computazione

$A + B \rightarrow A$;

if $M_0 = 0$ **then** $C + 1 \rightarrow C$ **else** $C - 1 \rightarrow C$;

si può implementare mediante una sola microistruzione, *posticipando* "$A + B \rightarrow A$" nei due rami della struttura *if then else*,:

i. $(M_0 = 0)$ $A + B \rightarrow A$, $C + 1 \rightarrow C$, $i + 1$; $(M_0 = 1)$ $A + B \rightarrow A$, $C - 1 \rightarrow C$, $i + 1$

6.4 Esercizi

Rifare gli esercizi **1, 4, 5** della sez. 3 applicando tutte le ottimizzazioni dei microprogrammi.

7. Componente logico memoria

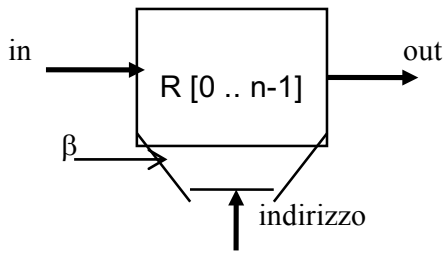
7.1 Realizzazione logica

Nella realizzazione di unità di elaborazione è spesso conveniente, o necessario, fare uso di un ulteriore componente logico "standard" : il componente memoria.

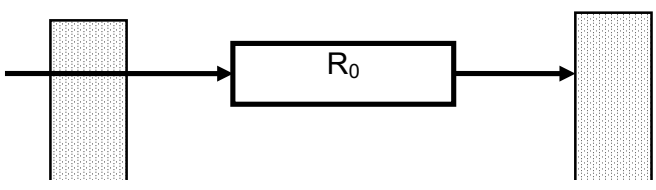
In versione *RAM*, una memoria è definita come un *array unidimensionale M di registri* su cui sono definite le seguenti operazioni fondamentali :

- *lettura* sull'uscita *out* del contenuto della cella di indirizzo i : *out* = M [i]
- *scrittura* del valore presente sull'ingresso *in* nella locazione di indirizzo i : M [i] : = *in*.

Il simbolo convenzionale è il seguente :



Lo schema di implementazione è mostrato in fig. 1.14.



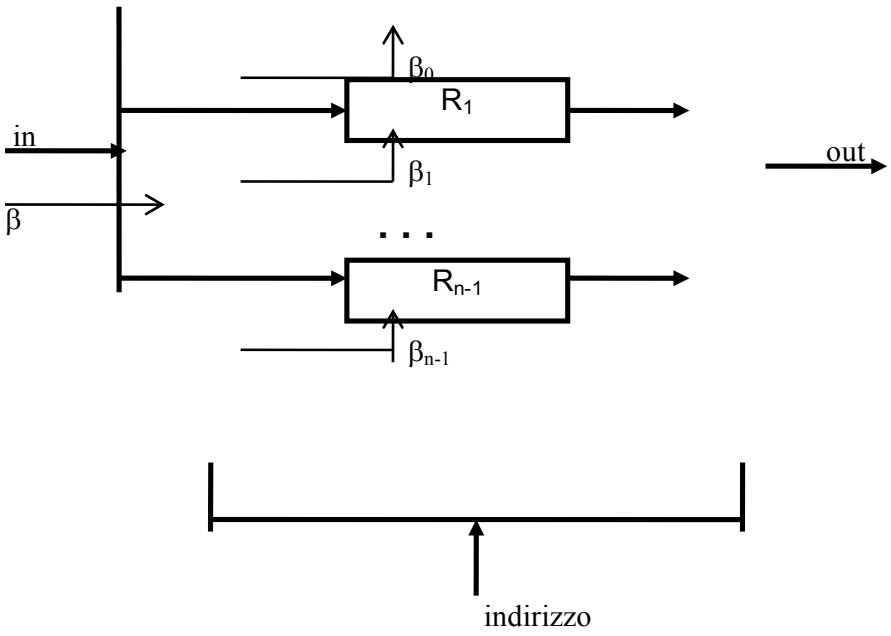


Fig. 1.14 - Realizzazione logica di una RAM

Il *commutatore* K di uscita, i cui ingressi primari sono le uscite dei registri ed i cui ingressi secondari sono i bit dell'indirizzo, realizza l'operazione di lettura.

Il *selezionatore* S di ingresso, avente anch'esso come ingressi secondari i bit dell'indirizzo, ha come ingresso il segnale di controllo β per l'abilitazione alla scrittura ; inoltre, l'ingresso *in* viene collegato a tutti gli ingressi dei registri ; di conseguenza, se β = 1, il valore di *in* viene scritto solo nel registro indirizzato.

Nel caso di una memoria *ROM*, è ovviamente presente solo il commutatore di uscita.

Si possono avere memorie RAM *a più ingressi e più uscite* (ad esempio, 1 ingresso e 2 uscite), utilizzando il numero corrispondente di selezionatori e commutatori.

7.2 Realizzazione fisica e ritardi

Questo schema di principio viene adottato sia per memorie di registri di piccola capacità (32 - 256 celle), sia per memorie, di grande capacità (10^7 - 10^9 celle) da adottare all'interno dell'unità memoria principale di un calcolatore general purpose. In tutti i casi, è improponibile una realizzazione del commutatore di uscita come rete a due livelli di logica, visto il numero troppo elevato di ingressi. La soluzione risiede nel realizzare la logica OR del secondo livello mediante una linea, di opportuna realizzazione elettronica, detta *Wired-OR*. La realizzazione di K che ne consegue è detta *bus in commutazione*.

Per ragioni analoghe, e vista la presenza di una linea che connette *in* a tutti i registri, il selezionatore di ingresso viene realizzato come *bus in selezione*.

Questi tipi di bus appartengono alla famiglia delle strutture dette **bus sincroni**. Sono possibili, nel caso più generale, bus sincroni (che useremo nella parte seconda per la progettazione del processore centrale) del tipo *bus in commutazione e selezione*, realizzati mediante la cascata di un commutatore e di un selezionatore aventi la linea (di uscita per K, di ingresso per S) in comune realizzata come Wired-OrR.

Il ritardo *d* di un bus sincrono in funzione del numero *n* di ingressi (e quindi della lunghezza fisica della linea Wired-OR) è quello tipico di un collegamento : per un numero di ingressi inferiore ad un certo limite *n1*, *d(n)* cresce lentamente, ragion per cui si può assumere il ritardo stesso praticamente costante e di valore (massimo) dato da *d(n1)* ; al di sopra di *n1*, *d* cresce linearmente con pendenza almeno uguale a quella in *n1*. Il valore di *n1* è dell'ordine di 10^1 - 10^2. Un elevato numero di ingressi consiglia allora di collegare più bus in cascata, ognuno con numero di ingressi inferiore ad *n1*, in modo da limitare il ritardo complessivo ad un multiplo piccolo di *d(n1).*

Il ritardo del bus sincrono usato nell'implementazione di una memoria, usato in commutazione e/o selezione, fornisce il valore del **tempo di accesso** della memoria stessa.

Le memorie RAM più economiche sono quelle così dette *dinamiche*, in quanto, per ragioni di potenza del segnale relativo al contenuto delle celle, ogni cella necessita di un "rinfresco" periodico" (la cella va letta e quindi riscritta con lo stesso valore letto) con cadenza dell'ordine dei msec. Tipici tempi di accesso di memorie dinamiche sono attualmente dell'ordine delle decine di nsec, con capacità dell'ordine 10 Mbyte per chip.

Le memorie *statiche* (che non necessitano di rinfresco) presentano tempi di accesso decisamente più bassi, dell'ordine di 1 - 10 nsec, ma, a parità di generazione tecnologica, sono anche assai meno "dense" delle dinamiche e quindi caratterizzate da una minore capacità per chip, dell'ordine del Mbyte.

7.3 Uso delle memorie nella realizzazione di unità di elaborazione

Il componente memoria RAM viene usato per la progettazione di unità di elaborazione che debbano manipolare strutture dati implementabili come array o mediante array. Il componente va usato come indicato dal simbolo convenzionale della sez. 7.1 ed in base alla definizione delle operazioni di lettura e scrittura.

Esempio 4

Progettiamo una unità di memoria "intelligente", avente ingressi DATAIN (dato d'ingresso da scrivere), IND (indirizzo) e COP (codice operativo), ed uscita DATAOUT (dato letto in uscita), capace di eseguire le seguenti operazioni esterne :

- COP = 0 : lettura della cella di indirizzo IND,

- COP = 1 : scrittura di DATAIN nella cella di indirizzo IND,
- COP = 2 : lettura all'indirizzo IND e scrittura di DATAIN allo *stesso* indirizzo,
- COP = 3 : incremento di uno del contenuto della cella di indirizzo IND.

Indicando con MEM il componente logico RAM, il microprogramma eseguibile è il seguente :

0. (COP = 00) MEM [IND] → DATAOUT, 0 ;
   (= 01) DATAIN → MEM [IND], 0 ;
   (= 10) DATAIN → MEM [IND], MEM [IND] → DATAOUT, 0 ;
   (= 11) MEM [IND] → TEMP, IND → IND1, 1
1. TEMP + 1 → MEM [IND1], 0

Il salvataggio di IND in IND1 è necessario in quanto non è possibile fare affidamento sul fatto che l'ingresso esterno rimanga costante per più di un ciclo di clock.

Avremmo anche potuto realizzare l'operazione esterna avente COP = 3 mediante un unico ciclo di clock, scrivendo nella microistruzione 0

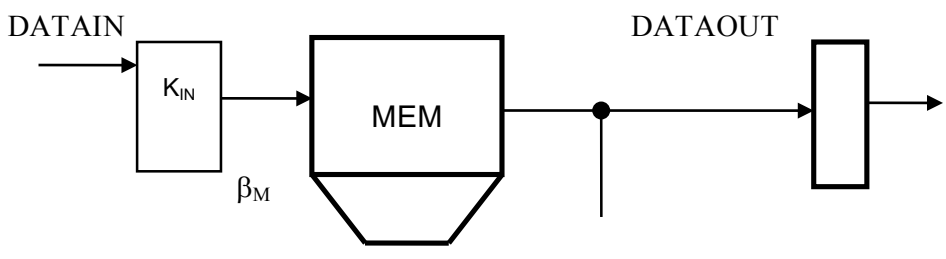
(= 11) MEM [IND] + 1→ MEM [IND], 0

Così facendo, avremmo però allungato il ciclo di clock, rispetto alla versione precedente, in quanto le *risorse* MEM e rete logica "+ 1" verrebbero usate *in cascata* durante uno stesso ciclo : si verrebbero, cioè, a sommare 3 ritardi (tempo di accesso per la lettura di MEM, incremento di 1, tempo di accesso per la scrittura di MEM). Questo maggior valore di τ renderebbe più veloce l'esecuzione dell'operazione avente COP = 3, ma rallenterebbe l'esecuzione delle altre tre operazioni. A meno di particolari valori delle probabilità di occorrenza delle operazioni esterne, il tempo medio di elaborazione risulta minore scegliendo un ciclo di clock di lunghezza minore. Per lo stesso motivo, è anche possibile che, a seconda dei ritardi delle risorse, convenga spezzare ulteriormente la microistruzione in due microistruzioni, la prima che incrementa TEMP, la seconda che scrive TEMP in memoria.

Si tratta di un problema generale, introdotto con l'esercizio 2 della sez. 3.8, che va comunque risolto caso per caso.

Lo schema della PO è mostrato in Fig. 1.15.

Si lascia come esercizio il completamento della sintesi di questa unità.



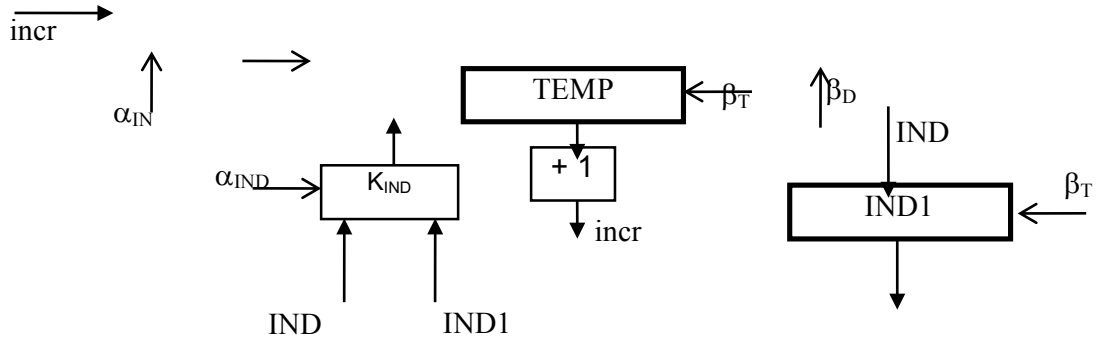


Fig. 1.15 - PO dell'esempio 4 (memoria intelligente)

7.4 Esercizi

Gli esercizi qui proposti corrispondono alla realizzazione a livello firmware di programmi didattici studiati nei corsi informatici del primo anno. Per ampliare l'insieme degli esercizi, lo studente può quindi riferirsi ai programmi suddetti.

Esercizio 7

Progettare una unità di elaborazione capace di eseguire operazioni su *array unidimensionali* di interi. L'unità contiene una memoria di registri A[0 .. 255]. Le operazioni da implementare sono:

a) caricamento di A con dati provenienti dall'esterno, uno alla volta in sequenza,

b) somma di tutti gli elementi di A (da restituire all'esterno),

c) ricerca di un valore *x* in A, restituendo all'esterno l'indirizzo dell'elemento uguale ad *x* (ci si limiti al caso della ricerca *certa* e con un solo elemento uguale ad *x*).

Il codice delle operazioni, ed i valori degli eventuali parametri, sono indicati tramite gli ingressi esterni.

Esercizio 8

Progettare una unità di elaborazione che implementi una *pila* (struttura *LIFO*: Last In First Out) di al più 64 elementi di tipo intero. Gli ingressi esterni rappresentano l'operazione (implementare almeno Inserzione ed Estrazione) e l'eventuale elemento da inserire. L'uscita esterna rappresenta l'eventuale valore estratto.

Nota: si supponga (per il momento) che non possano verificarsi i casi di richiesta di inserzione quando la pila è piena e di richiesta di estrazione quando la pila è vuota. Questi casi dovranno essere trattati quando si farà uso della *sincronizzazione* nelle comunicazioni tra unità.

Esercizio 9

Progettare una unità di elaborazione che implementi una *coda FIFO* (First In First Out) di al più 64 elementi di tipo intero. Gli ingressi esterni rappresentano l'operazione (implementare almeno Inserzione ed Estrazione) e l'eventuale elemento da inserire. L'uscita esterna rappresenta l'eventuale valore estratto. Vale una **Nota** analoga a quella dell'Esercizio 9.

7.5 Controllo residuo

Come sappiamo, ogni risorsa hardware (rete combinatoria o registro) ha in ingresso un certo numero di variabili di controllo. Spesso queste sono date da esplicite variabili $\{\alpha, \beta\}$ emesse dalla PC, ma esistono diversi casi interessanti in cui esse sono generate come funzioni di contenuti di registri della PO : in questo secondo caso si parla di *controllo residuo*.

Un caso tipico è l'indirizzamento di una memoria, interna ad una unità, mediante registri o ingressi esterni dell'unità stessa.. Altri casi notevoli verranno visti nella seconda parte a proposito del progetto del processore centrale.

Il controllo residuo permette una *drastica riduzione della complessità di progettazione e di struttura della PC* : basti pensare a cosa andremmo incontro se volessimo indirizzare una memoria solo con variabili $\{\alpha\}$ dalla PC.

Esercizio 10

Chiarire l'ultima affermazione. Riferendosi ad un caso specifico, come l'esempio 4 della sez. 7.3, studiare le differenze rispetto al caso in cui la memoria venga indirizzata senza applicare alcuna forma di controllo residuo.

7.6 Uso di memorie per la realizzazione di reti combinatorie

Memorie ROM possono essere utilizzate, con vantaggio dal punto di vista del costo e della complessità di progettazione, per la realizzazione di funzioni combinatorie.

Nel caso più semplice, una rete con ingressi X e uscite Z può essere implementata direttamente, cioè senza passare attraverso il procedimento di sintesi, mediante una memoria MEM indirizzata da X e tale che MEM $[X_i] = Z_i$, per ogni *i*.

8. Tecnologie integrate

La caratteristica delle tecnologie integrate è quella di cercare di realizzare su una singola piastrina di pochi centimetri di lato, o *chip*, il maggior numero possibile di componenti di una unità o di un intero sottosistema.

L'evoluzione tecnologica è, a partire dalla metà degli anni 80, improntata all'integrazione su larghissima scala, *VLSI* (Very Large Scale Integration) ottenuta con procedimento CMOS. Una unità PC-PO, con parola di 32 - 64 bit, incluso un tipico processore centrale, con ciclo di clock dell'ordine di 5 - 10 nsec, è oggi integrabile su singolo chip, allo stesso modo di una memoria RAM da 10 - 100 Mbit con tempo di accesso dell'ordine di 5 - 50 nsec. Il caso della *memoria* è particolarmente significativo : trattandosi di una struttura estremamente regolare, e dunque predisposta ad un "impaccamento" molto denso dei componenti, il numero di bit che può essere integrato su un chip di memoria è una misura attendibile del massimo numero di componenti logici che, in un dato momento tecnologico, può essere integrato su uno stesso chip.

Per comprendere l'evoluzione di questa tecnologia, è significativa la Tab. 1.4 :

Anno	*N. transistori per chip*	*Lunghezza porta logica (in μm)*	*Area del chip (in mm^2)*	*Ciclo di clock o ritardo (in nsec)*
1986	2.5×10^5	1.4	80	50
1991	1.5×10^6	0.8	150	25
1994	10^7	0.3	400	10
2000	10^8	0.05	2000	1

Tab. 1.4.- Evoluzione della tecnologia VLSI

È stato appurato che, almeno per i prossimi 15 anni, la tendenza sopra mostrata continuerà a sussistere : di conseguenza, per avere i valori validi nei prossimi anni è sufficiente fare una semplice estrapolazione dei dati in tabella.

Da un punto di vista logico, il problema dell'integrazione consiste nella ricerca del miglior *compromesso tra area e tempo di elaborazione*.

L'area del chip è proporzionale al numero di elementi da integrare, il perimetro del chip è proporzionale al numero di bit presenti all'interfaccia (*pin*) dell'unità o del sottosistema integrato sul chip stesso ; attualmente, 200 - 300 pin è la dimensione più attendibile.

Aumentando l'area del chip aumenta il perimetro e quindi *la capacità di interfacciamento*. Ragionando in termini di *lunghezza di parola*, si può essere indotti a pensare che l'interesse principale del costruttore stia nell'aumentare la capacità di interfacciamento. Occorre però fare attenzione al fatto che *al corrispondente aumento di area deve effettivamente corrispondere anche un aumento proporzionale nel numero dei componenti da integrare* : se cosi non fosse, cioè se l'impaccamento dei componenti non fosse almeno proporzionalmente più denso, l'aumento di area si tradurrebbe in un aumento dei *ritardi* interni, ed in particolare in un aumento dei ritardi dei *collegamenti*, con conseguente aumento del ciclo di clock. Dunque, il costruttore non decide di aumentare la dimensione del chip, e quindi non passa ad un sistema con lunghezza di parola maggiore, finchè non è sicuro di poter integrare un maggior numero di componenti, e quindi finchè non è sicuro di poter integrare un numero molto maggiore di funzionalità.

9. Parte controllo microprogrammata

Quando il numero di microistruzioni dell'unità sia relativamente alto, come avviene nel caso di processori centrali o coprocessori particolarmente complessi, la progettazione della PC rappresenta un problema di complessità notevole.

La realizzazione così detta *microprogrammata* [BTV, cap. 5] permette di ridurre grandemente la complessità di progettazione e, al tempo stesso, di rendere la struttura della PC più regolare e quindi predisposta all'integrazione VLSI.

Il principio è quello di memorizzare gran parte del microprogramma in una memoria, detta *Memoria di Controllo* (MC) [1], mediante la quale implementare in gran parte le funzioni di transizione delle uscite λ_{PC} e di transizione dello stato interno δ_{PC}. La rimanente parte di tali funzioni viene delegata ad una rete combinatoria (anch'essa realizzabile con una memoria, se ritenuto opportuno), detta *Rete di Condizionamento*.

Nel caso della **PC di Mealy** con microlinguaggio PS, si tratta di memorizzare in MC tutte le coppie ("frasi")

microoperazione, indirizzo successivo

[1] È sufficiente che MC sia del tipo ROM, ma non necessario.

MC è indirizzata dall'uscita della Rete di Condizionamento, i cui ingressi sono l'uscita del registro di stato RC e le variabili di condizionamento. Lo schema è mostrato in fig. 1.18.

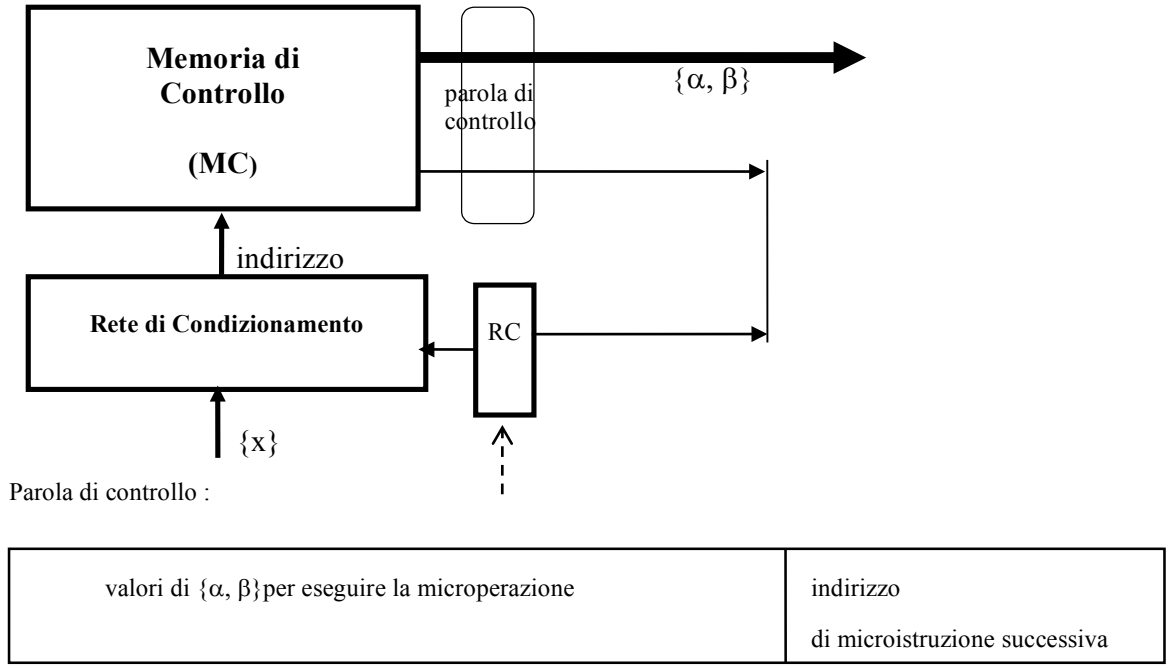


Fig. 1.18 - Schema di PC microprogrammata, modello di Mealy

Ogni *parola di controllo* contiene, nel primo campo, la configurazione dei valori delle variabili di controllo {α, β} necessari ad eseguire la microoperazione, e nel secondo campo l'indirizzo di microistruzione successiva che viene inviato all'ingresso del registro di stato del controllo RC (K_{RC}).

L'implementazione microprogrammata risolve brillantemente i problemi di complessità della progettazione e della struttura di PC, al prezzo di una minore velocità : il *ritardo* di PC è ora più grande rispetto al caso di una realizzazione con rete combinatoria "classica" a due livelli di logica, e quindi il ciclo di clock è più lungo.

Si può ovviare utilizzando una tecnologia più avanzata, e quindi più costosa, che riduca fortemente il tempo di accesso della memoria MC.

10. Cenno ai modelli Moore-Moore e Moore-Mealy

L'ottimizzazione delle prestazioni passa anche attraverso la scelta dell'opportuno modello di microprogrammazione [BTV, cap. 5.2.4]. Oltre al modello Mealy-Moore, considerato finora, è interessante il modello *Moore-Moore* che, pur dando luogo ad un maggior numero di cicli di clock per uno stesso algoritmo, è caratterizzato da una minore lunghezza del ciclo di clock in quanto è assai

maggiore il grado di sovrapposizione tra gli intervalli di stabilizzazione delle funzioni della PC e quelle della PO. Come scegliere il modello con minore tempo di elaborazione è indicato nel suddetto capitolo di [BTV].

Al modello Moore-Moore corrisponde il microlinguaggio *TS* (Transfer Structured), avente la seguente struttura :

etichetta. microperazione
case condizione logica **of**
valore 1 : indirizzo successivo 1 ;
. . .
valore k : indirizzo successivo k

Ad esempio, il microprogramma TS per l'esempio 1 di sez. 2 è il seguente :

0. $A + B \rightarrow A, 1$
1. nop $(A_0 = 0)$ 2; $(A_0 = 1)$ 3
2. $A + B \rightarrow A, 1$
3. $- A \rightarrow A, 0$

Come si vede, questa unità impiega un maggior numero di cicli di clock rispetto al modello Mealy-Moore ($T = 2.5\ \tau_1$), in quanto alcune *nop* sono ineliminabili nel modello Moore-Moore (oltre a quelle introdotte dai meccanismi di sincronizzazione). D'altra parte, la minore lunghezza del ciclo di clock contribuisce ad attenuare la differenza, o addirittura, in alcuni casi, a rendere complessivamente minore il tempo di elaborazione nel caso Moore-Moore.

Con una PC di Moore, e quindi con una microlinguaggio TS, è anche possibile avere una PO di Mealy : il modo più semplice di vedere una PO di Mealy è quello di anticipare il prelievo delle variabili di condizionamento all'ingresso dei registri ("Moore anticipato"). Indicando con a_0 l'ingresso del registro A_0, nel *modello Moore-Mealy* il microprogramma dell'esempio precedente diviene :

0. $A + B \rightarrow A$ $(a_0 = 0)$ 1; $(a_0 = 1)$ 2
1. $A + B \rightarrow A, 1$
2. $- A \rightarrow A, 0$

Come si vede, si ottiene lo stesso numero di cicli di clock del caso Mealy-Moore (le *nop* sono sempre eliminabili nel modello Moore-Mealy, a parte quelle introdotte dai meccanismi di sincronizzazione). Si può anche dimostrare che la lunghezza del ciclo di clock è uguale a quella del modello Mealy-Moore : i modelli Mealy-Moore e Moore-Mealy sono dunque completamente equivalenti ed interscambiabili.

È lasciata come esercizio la dimostrazione che il modello *Mealy-Mealy* è improponibile in quanto caratterizzato da un funzionamento non determinato a livello di ciclo di clock.