Soluzioni del Capitolo 9

Questo documento contiene le soluzioni ad un numero selezionato di esercizi del Capitolo 9 del libro "Calcolatori Elettronici - Architettura e organizzazione", Mc-Graw Hill 2017.
Sarò grato a coloro che mi segnaleranno errori di qualunque genere. Sarò altrettanto grato a chi mi segnalerà errori nel libro.

Vorrei invitare coloro che avessero sviluppato soluzioni alternative a quelle da me proposte, o soluzioni a esercizi non compresi tra quelli qui trattati, a trasmettermele, in modo da migliorare i contenuti di questo sito.
Inviare le segnalazioni a: `giacomo.bucci@unifi.it`

Aggiornato il 19 aprile 2017

9.1 In presenza di un flusso di istruzioni senza salti o conflitti il guadagno di prestazioni della pipeline è pari al numero degli stadi. Questo suggerisce di realizzare pipeline ad alto numero di stadi. Tuttavia il flusso reale prevede i salti (e le interruzioni), che comportano lo svuotamento. La penalità pagata è tanto maggiore quanti più stadi devono essere svuotati. In una pipeline molto lunga è probabile che il numero di stadi da svuotare (cioè quelli che precedono lo stadio che rileva la condizione di svuotamento) sia alto. Un esempio chiarirà il concetto. Supponiamo che, come nella nostra pipeline, si debbano svuotare 2 stadi e assumiamo che gli svuotamenti capitino con una frequenza del 5%. Ciò equivale a una penalizzazione (perdita di clock effettivi) del 10%. Se gli stadi da svuotare fossero 4, la penalizzazione sarebbe del 20%.

Naturalmente a questo si deve aggiungere che più la pipeline è lunga e più è complessa e più componenti sono necessari per realizzarla, occupando spazio sul silicio.

9.2 Osserviamo anzitutto che per tutte le istruzioni diverse da LD/ST l'eliminazione dello stadio ME ha solo un effetto benefico in quanto per esse il passaggio attraverso lo stadio ME comporta solo una perdita di tempo[1]. Per queste istruzioni la differenza con la pipeline a 5 stadi consiste semplicemente nel far passare l'uscita della ALU direttamente sul latch EX/WB (il latch EX/ME è sparito).

Per l'istruzione LD occorre prevedere che la memoria venga letta direttamente in EX e che il dato letto venga scritto in EX/WB.MEMDat. Dunque occorre prevedere che nello stadio EX l'uscita della ALU venga direttamente presentata alla memoria (come indirizzo) e che venga asserito il comando M_Read. Per l'istruzione ST lo stadio EX non è dissimile da quello della pipeline a 5 stadi, con la differenza che ora la scrittura in memoria avviene nello stadio WB, il quale deve essere attrezzato con la logica corrispondente; in particolare il comando M_Write deve essere asserito in WB. In ogni caso è possibile che l'insieme delle operazioni che ora posson essere eseguite in EX o WB richieda un estensione della durata del periodo di clock.

9.3 Risposta ai tre quesiti.

[1] Questa affermazione richiede una precisazione. A parità di altre condizioni, l'eliminazione dello stadio ME allunga la durata del periodo di clock, in quanto lo stadio EX deve svolgere la funzione di calcolo dell'indirizzo e di accesso alla memoria. Ne risulta una pipeline a 4 stadi, ma con frequenza di clock diminuita. Resta perciò da vedere, per un dato mix di istruzioni, quale delle due soluzioni è la migliore.

1. Nel caso di macchina multiciclo non in pipeline il periodo di clock è pari al tempo richiesto dallo stadio più lento: 5 ns. Pertanto la frequenza del clock sarebbe $f = 1/(5\ 10^{-9}) = 0,2\ 10^9 = 200$ MHz (si è fatta l'ipotesi che il tempo di memorizzazione in IR fosse compreso nel tempo del primo stadio). Il tasso di esecuzione sarebbe pari a $200\ 10^6/5 = 40$ milioni di istruzioni al secondo.
2. Nel caso di pipeline il periodo di clock sarebbe pari a 6 ns, per una frequenza di clock $f = 1/6\ 10^{-9} = 166,67$ MHz. In condizioni di pipeline sempre piena (limite massimo delle prestazioni), il tasso di esecuzione sarebbe pari a 166, 67 milioni di istruzioni al secondo.
3. Il guadagno tra la soluzione del punto 2 e quella del punto 1 è pari a $166,67/40 = 4,17$.

9.4 In Figura 9.1 viene mostrato lo schema del JMP. Come al solito vengono mostrati solo i segnali rilevanti. In Figura 9.2 viene mostrato lo schema dell'istruzione JR,mentre lo schema dell'istruzione JAL è in Figura 9.3. Per quest'ultima si noti che l'indirizzo di ritorno viene passato attraverso ALU, in modo che in WB venga prelevato dal campo ME/WB.ALUOut per essere copiato nel registro di destinazione

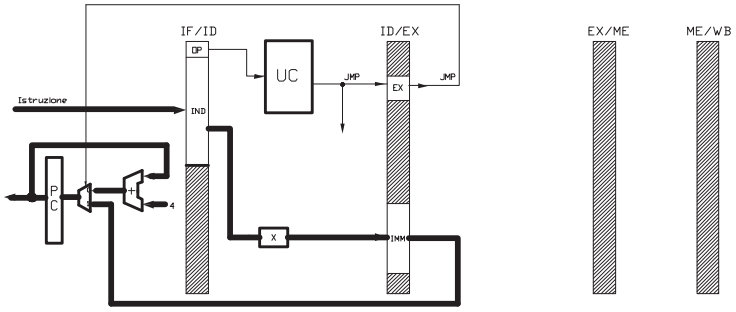

Figura 9.1 (Esercizio 9.4) Schema del trattamento dell'istruzione JMP.

9.5 Lo schema per selezionare l'ingresso a PC è in Figura 9.4.

9.6 La pipeline richiede queste modifiche:

- in ID ed EX il flusso dei dati è del tutto analogo a quello dell'istruzione ST, con il contenuto del registro RS1 (R2 nel caso specifico) presentato in ALU.A e con il contenuto del campo OFFSET trasferito in ID/EX.IMM e da qui presentato in ALU.B.
- L'uscita della ALU viene trasferita verso EX/ME.ALUOut, come se si trattasse di una operazione aritmetica, in modo che il risultato possa propagarsi attraverso lo stadio ME verso lo stadio WB, esattamente come con le operazioni aritmetiche.

9.7 Per rendere possibile l'esecuzione di un'istruzione come `ADDM Rx,Mem` si può seguire la soluzione di rendere accessibile la memoria dallo stadio EX, in modo da far sì che l'operazione aritmetica possa essere eseguita in EX; nel seguito l'istruzione può procedere come una normale operazione aritmetica.

Se l'istruzione fosse del tipo `ADDM Mem,rx`, ovvero, se essa prevedesse l'esecuzione della somma e la memorizzazione del risultato il memoria l'istruzione richiederebbe due

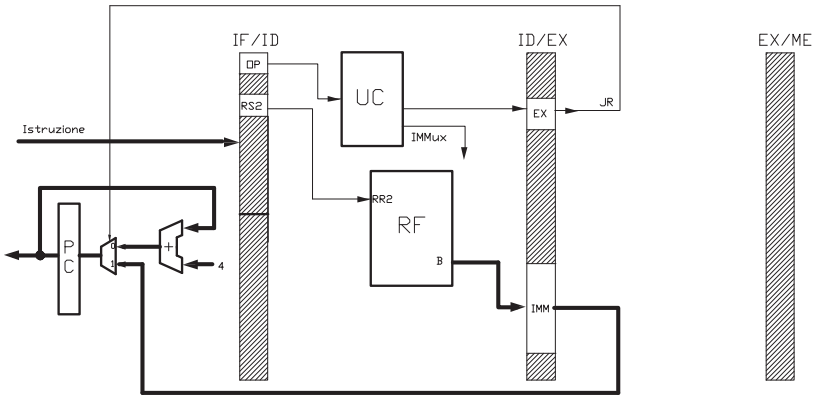

Figura 9.2 (Esercizio 9.4) Schema del trattamento dell'istruzione JR.

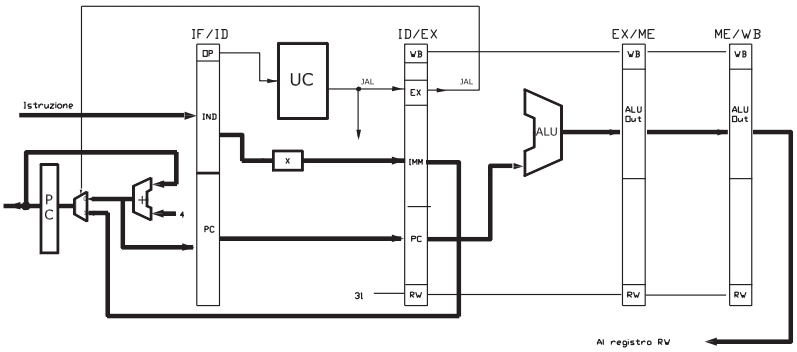

Figura 9.3 (Esercizio 9.4) Schema del trattamento dell'istruzione JAL.

accessi alla memoria: in EX (per leggere il primo addendo ed effettuare la somma) e in ME (per scrivere il risultato).

In ambedue i casi, se l'istruzione che precede fosse una LD/ST (che perciò accede alla memoria dallo stadio ME) ovvero un'altra `ADDM Mem,rx`, si avrebbe una situazione di conflitto a causa del tentativo di accesso concomitante alla memoria dallo stadio EX e dallo stadio ME.

9.8 Ambedue le sequenze A e B prevedono l'impiego del registro R1 come registro sorgente nella seconda istruzione.
Risposta alla domanda a)
Se il conflitto viene risolto con lo stallo è necessario che la seconda istruzione venga tenuta in stallo in ID fino a che non è stato scritto R1. Ne consegue che la seconda istruzione

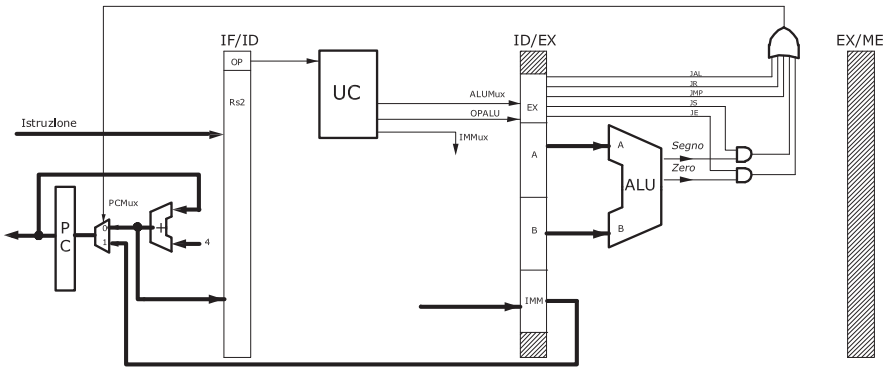

Figura 9.4 (Esercizio 9.5) Rete di selezione ingresso PC.

deve essere tenuta bloccata per 3 clock (corrispondenti al passaggio attraverso EX, ME e WB della prima istruzione).
Risposta alla domanda b)
Le relazioni che danno le condizioni di stallo sono quelle di pagina 249 del testo, con l'aggiunta delle due condizioni riportate qui di seguito e non presenti nel testo dove è stata fatta l'assunzione della sovrapposizione tra le fasi ID e WB.

Il conflitto fra l'istruzione in ID e quella in WB si verifica se è vera almeno una delle seguenti funzioni logiche:

$$\text{ConflWa} = (\text{IF/ID.Rs1} \equiv \text{ME/WB.RW}) \cdot \text{S1} \cdot \text{ME/WB.RWrite}$$

$$\text{ConflWb} = (\text{IF/ID.Rs2} \equiv \text{ME/WB.RW}) \cdot \text{S2} \cdot \text{ME/WB.RWrite}$$

La condizione di conflitto risulta dunque:

$$\text{Confl} = \text{ConflXa} + \text{ConflXb} + \text{ConflMa} + \text{ConflMb} + \text{ConflWa} + \text{ConflWb}$$

dove ConflXa, ConflXb, ConflMa e ConflMb hanno il significato riportato nel testo.

Tracciare la rete che dà Confl è un banale esercizio manuale.

Per quanto si riferisce all'uso di una rete di by-pass, si osservi che le due sequenze danno luogo a soluzioni differenti.

Nel caso della sequenza A si tratta di un conflitto tra l'istruzione ST che usa il dato (R1) in ME e l'istruzione LD che produce il dato in ME. Dunque quando l'istruzione ST è in ME, il dato è già stato prodotto e si trova su ME/WB; pertanto il forward è da ME/WB.MEDat alla memoria.

Nel caso della sequenza B l'istruzione ST usa in EX il dato prodotto dalla LD (in ME). In questo caso lo stallo è inevitabile e il forward è da ME/WB.MEDat a EX. La relativa rete di bypass è mostrata In Figura 9.5.

Si noti che Confl verifica la condizione

$$\text{Confl} = (\text{IF/ID.Rs1} \equiv \text{ID/EX.RW}) \cdot \text{S1} \cdot \text{ID/EX.RWrite}$$

per la quale si ha conflitto tra l'istruzione in ID e quella in EX sulla via A. Lo schema della figura mostra che Confl viene memorizzato su ID/EX, in modo che al clock successivo venga attuato il by-pass del contenuto di ID/EX.A con il contenuto di EX.ME.ALUOut. Più precisamente il funzionamento è il seguente:

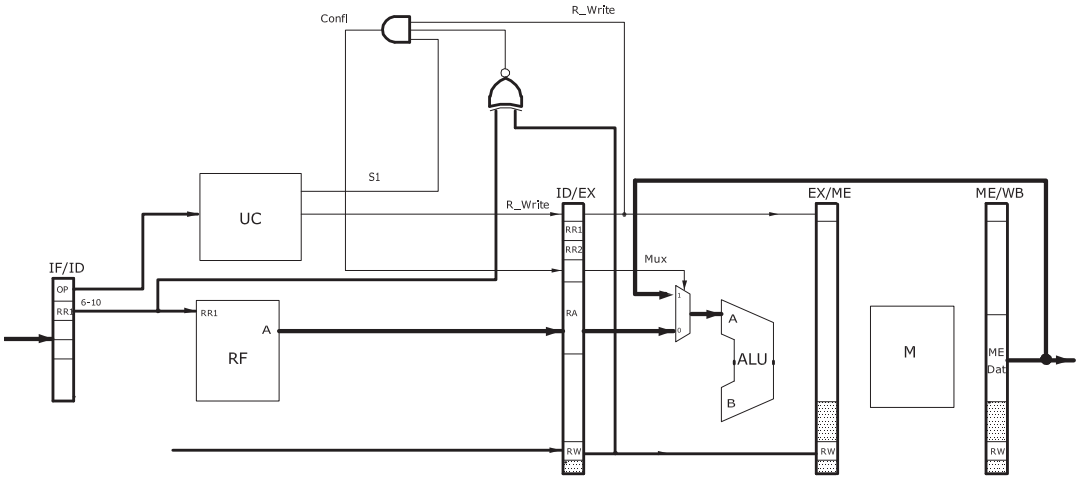

Figura 9.5 (Esercizio 9.8) By-pass sulla via A.

- quando l'istruzione ST è in ID viene rilevata la condizione di conflitto; ciò porta a scrivere il selettore Mux che farà scegliere il risultato che nello stesso periodo di clock viene calcolato per la prima istruzione (che si trova in EX);
- Al clock successivo l'istruzione ST si trova in EX e, per via del selettore in ingresso alla porta A di ALU viene prelevato il risultato calcolato in precedenza come primo operando.

9.11 Valgono le seguenti considerazioni.

- Tra la prima e la seconda istruzione non ci sono conflitti.
- Tra la seconda e la terza c'è un conflitto per R6. Il numero degli eventuali stalli e la struttura dell'eventuale by-pass sono identici a quelli visti all'esercizio 8.9.
- Quando la quarta istruzione è in ID (effettua il fetch del contenuto dei registri) la prima istruzione è in WB. Dunque quando la quarta istruzione legge i registri questi non sono ancora stati scritti. Piuttosto che usare una rete di by-pass conviene sovrapporre le fasi ID e WB (clock halving).

9.12 Rispetto alla nostra pipeline, qui la condizione di salto è determinata dall'istruzione che precede il salto; la condizione è contenuta nella necessaria parola di stato, PSW, del processore.

- Se il calcolo dell'indirizzo di destinazione viene effettuato in EX, la modifica del PC non può avvenire che al termine del ciclo di clock in cui l'istruzione `Jc OFFSET` è in EX, ciò comporta l'introduzione di due bolle in modo da eliminare sia l'istruzione che si trova in ID sia quella che si trova in IF.
- Se, come nella nostra pipeline, il calcolo dell'indirizzo di destinazione e l'aggiornamento del PC viene effettuato in ID, allora si richiede una solo bolla, quella necessaria all'eliminazione dell'istruzione di cui viene fatto il fetch (cioè l'istruzione immediatamente seguente a `Jc OFFSET`).

- Nel secondo caso non è strettamente necessario memorizzare la condizione `c`. Infatti, mentre `Jc OFFSET`) si trova in ID e produce l'indirizzo di destinazione del salto, l'istruzione precedente si trova in EX e quindi, al termine del ciclo di clock è disponibile sia l'indirizzo di destinazione sia il valore della condizione. Tramite quest'ultimo è possibile comandare il caricamento del PC con l'indirizzo di destinazione appena calcolato e, al tempo stesso, inserire una bolla per l'istruzione che verrebbe copiata su IF/D (ovvero forzare un NOP nel campo OP di IF/ID).

9.14 Prima del tratto che parte da `altre` dovrebbero essere riportati al valore dovuto solo i registri che vengono usati come sorgente e sono stati aggiornati una volta in più, quindi non R7 che non viene usato come sorgente.

9.15 Si osservi che la prima `ADD`, in ID, richiede che R2 sia stato aggiornato dalla precedente istruzione, ovvero che questa sia passata dello stadio di WB. Si hanno pertanto 3 stalli. Per la seconda `ADD` vale un discorso analogo relativamente al registro R3. Aggiungendo i due cicli persi per i NOP, in totale si ha una perdita di 8 cicli di clock.

Proviamo ora a ristrutturare il codice riempiendo i *dalay slot* come qui di seguito, eliminando ovviamente i 2 `NOP`:

```
LD      R1, 220
LD      R2, 100
JMP     QUALCHE_PARTE
ADD     R3, R1, R2
ADD     R4, R3, R1
```

Il codice precedente ha un problema: Le due istruzioni `ADD` riempiono i delay slot, ma la prima `ADD` ha comunque una dipendenza dal secondo `LD` che implica 2 stalli (il `JMP` interposto ha ridotto di uno il numero di eventuali stalli, ma, ovviamente, non li ha annullati). Si tenga però conto che il salto modifica PC dopo due *delay slot*, quindi modificherebbe PC al termine del primo stallo, con ciò facendo seguire il fetch dell'istruzione di destinazione. In altre parole, è necessario che nel ciclo in cui `JMP` modifica PC venga fatto il fetch della seconda ADD. Ovvero, è necessario che i fetch delle ultime tre istruzioni vengano eseguiti esattamente in tre clock consecutivi.

È escluso che si possano mettere 2 stalli nel corpo dell'istruzione di salto (prima della fase EX), perché sarebbe come dire che un conflitto dati tra due istruzioni, che scrivono/leggono un dato, viene risolto intervenendo su una terza di tutt'altra natura[2]; parimenti è escluso che si possano inserire dei `NOP` tra il `JMP` e le seguenti. L'unica soluzione è introdurre due `NOP` prima del `JMP`. La sequenza corretta diventa quindi

```
LD      R1, 220
LD      R2, 100
NOP
NOP
JMP     QUALCHE_PARTE
ADD     R3, R1, R2
```

[2]Ovviamente si potrebbe trovare una soluzione ad hoc per questo caso, ma essa sarebbe difficilmente generalizzabile.

```
ADD     R4, R3, R1
```

L'ultima istruzione subisce 3 stalli. Complessivamente il numero di cicli persi risulta pertanto 5. Si è passati da 8 a 5.

9.16 Notiamo anzitutto che l'istruzione `MUL` viene eseguita 5 volte (con R2 = 5, 4, 3, 2,1); la successiva istruzione `SUB` viene pure eseguita 5 volte, come pure l'istruzione `JNE` (con R2 = 4, 3, 2, 1, 0). I valori prodotti in R5 sono, ordinatamente, 5, 20, 60, 120, 120.

Ai primi quattro passaggi la diramazione ha luogo, nel senso che nella logica del programma a `JNE` segue `MUL`; al quinto la diramazione non ha luogo; nella logica del programma si passa all'istruzione successiva a `JNE` (`indicata come :::`).

Siccome l'istruzione `JNE` modifica il PC in EX, nel caso in cui essa determini il salto all'indietro, la logica di CPU deve eliminare dalla pipeline le due istruzioni che la seguono nel testo del programma, ovvero devono essere inserite due bolle. In conclusione il codice dell'esercizio introduce 8 bolle sul complesso delle 5 ripetizioni del loop.

9.20 Il fatto che le cache siano separate elimina automaticamente ogni possibile conflitto tra la fase di fetch delle istruzioni e quella di load/store dei dati. Peraltro, consente anche di trattare le operazioni di load/store, esse stesse, in pipeline, come verrà illustrato al Capitolo 10.