Soluzioni degli esercizi del Capitolo 11

Questo documento contiene le soluzioni ad un numero selezionato di esercizi del Capitolo 11 del libro "Calcolatori Elettronici - Architettura e organizzazione", Mc-Graw Hill 2017.
Coloro che avessero sviluppato soluzioni alternative a quelle qui proposte, o soluzioni a esercizi non compresi tra quelli qui trattati, sono invitati a trasmetterle all'indirizzo sotto riportato. Serviranno a migliorare e tenere aggiornati i contenuti di questo sito.
L'autore sarà grato nei confronti di coloro che segnaleranno errori di qualunque genere, sia nella parte che segue sia nel libro menzionato.
`giacomo.bucci@unifi.it`

Aggiornato il 19 aprile 2017

11.1 L'elaborazione da compiere è `Y[0:63] = Y[0:63] + a * X[0:63]`. Indicando con X e Y le posizioni dei (primi elementi dei) vettori X e Y in memoria, il codice è il seguente. Ovviamente si è assunto che l'architettura sia del tipo registro-registro.

```
FLD     F1, a               ;F1 <- a
VLD     V1, X               ;Load X[] in V1
VLD     V2, Y               ;Load Y[] in V2
VMULSF  V3, V1, F1          ;V3 = X[]*a
VADD    V4, V2, V3          ;V4 = Y[]+X[]*a
VST     Y, V4               ;Y[] <- V4
```

Il significato delle istruzioni è ovvio; VMULSF è l'istruzione che moltiplica un vettore per uno scalare floating.

11.2 La dimensione del vettore non è multipla della dimensione dei registri. Il quoziente di 400/64 è pari a 6, il resto è pari a 16. Occorreranno quindi 7 iterazioni dell'operazione di somma vettoriale. Con il metodo *strip mining* la prima iterazione effettuerà la somma di 16 elementi, le restanti 6 la somma di 64. A tale scopo, il registro VRL deve essere inizialmente caricato con 16, in modo che le operazioni vettoriali operino su 16 elementi. Nelle iterazioni successive, VLR deve contenere 64, in modo che le operazioni vettoriali lavorino su 64 elementi.

11.3 Concettualmente l'esercizio è del tutto simile al precedente. Indichiamo con N il numero di iterazioni che servono a sommare per intero i registri (N = quoziente(200/64)= 3) e indichiamo con M il resto della divisione, (M = resto(200/64) = 8). Occorre prevedere un passo in cui si elaborano i primi M elementi dei vettori e N passi in cui si elaborano ogni volta 64 elementi. Si faccia riferimento a quanto detto per l'esercizio 11.2 per quanto riguarda la predisposizione di VLR e alla soluzione dell'esercizio 11.1 per quanto riguarda il codice.

Supponiamo di usare R1 come indice per indirizzare i vettori e di usare R2 come contatore. R1 deve essere inizializzato a 0, R2 a 200. Usiamo il registro R3 per contenere il numero di elementi da trattare ad ogni ciclo. Per ipotesi ogni elemento dei vettori è in doppia precisione e quindi occupa 8 byte. Il primo passaggio è questo

```
;R1 indice
;R2 contatore; inizialmente contiene 200
;R3 num elementi per iterazione
;
        XOR     R1,R1,R1        ;R1 <- 0
        ANDI    R3,R2,64        ;R3 <- resto (=8)   vedi nota
        LDVLR   R3              ;VLR <- resto
        FLD     F1, a           ;F1 <- a
;
        VLD     V1, X(R1)           ;Load X[] in V1
        VLD     V2, Y(R1)           ;Load Y[] in V2
        VMULSF  V3, V1, F1          ;V3 = X[]*a
        VADD    V4, V2, V3          ;V4 = Y[]+X[]*a
        VST     Y, V4               ;Y[] <- V4
```

Nota: L'istruzione `ANDI` (AND immediato) effettua l'AND tra il contenuto del registro sorgente e il numero che costituisce l'immediato. È facile convincersi che il resto di una divisione per un numero potenza del 2 (2^k) è dato dal contenuto dei k bit meno significativi del numeratore. Quindi l'istruzione `ANDI R3,R2,64` mette in R3 il resto di 200/64, cioè 8.

Le iterazioni successive richiedono che in VLR ci sia 64. Per il corpo dell'elaborazione valgono le ultime 5 istruzioni del precedente codice.

```
;R0 contiene permanentemente 0
;R1 indice
;R2 contatore; inizialmente contiene 200
;R3 num elementi per iterazione
;
        XOR     R1,R1,R1        ;R1 <- 0
        ANDI    R3,R2,64        ;R3 <- resto (=8)   vedi nota
        LDVLR   R3              ;VLR <- resto
        FLD     F1, a           ;F1 <- a
loop:
        VLD     V1, X(R1)           ;Load X[] in V1
        VLD     V2, Y(R1)           ;Load Y[] in V2
        VMULSF  V3, V1, F1          ;V3 = X[]*a
        VADD    V4, V2, V3          ;V4 = Y[]+X[]*a
        VST     Y, V4               ;Y[] <- V4
;
        SUB     R2,R2,R3        ;conteggia per il test di conclusione
        SLI     R3,R3,3         ;R3 <- R3*8  byte da saltare
        ADD     R1,R1,R3        ;R1 punta al prossimo blocco di 64 elementi
        ADDI    R3,R0,64        ;R3 <- 64
        LDVLR   R3              ;blocchi di 64 elementi
        BNEQ    R2,R0,loop
```

L'istruzione `SLI` (*shift left immediate*) fa scorrere a sinistra del numero corrispondente all'immediato. Uno scorrimento di 3 posizioni equivale a una moltiplicazione per 8. Poiché ogni elemento occupa 8 byte (doppia precisione), occorre saltare un numero di byte pari a 8 volte il numero di elementi (in VLR).
Il ciclo si conclude quando R2 arriva a zero.

11.5 Il calcolo si effettua considerando che le operazioni sui vettori richiedono 64 cicli di clock (assumendo che ogni operazione sui singoli elementi richieda un solo clock) e quindi l'effettiva conclusione dell'operazione `vadd`, emessa sull'ultimo ciclo del loop, si avrà solo dopo 64 cicli. Per il resto, con le ipotesi fatte e assumendo che il processore attui il *chaining* (Pag. 444 del testo), si tratta di conteggiare 4 volte il numero di operazioni svolte nel corpo del loop (oltre le poche che precedono).

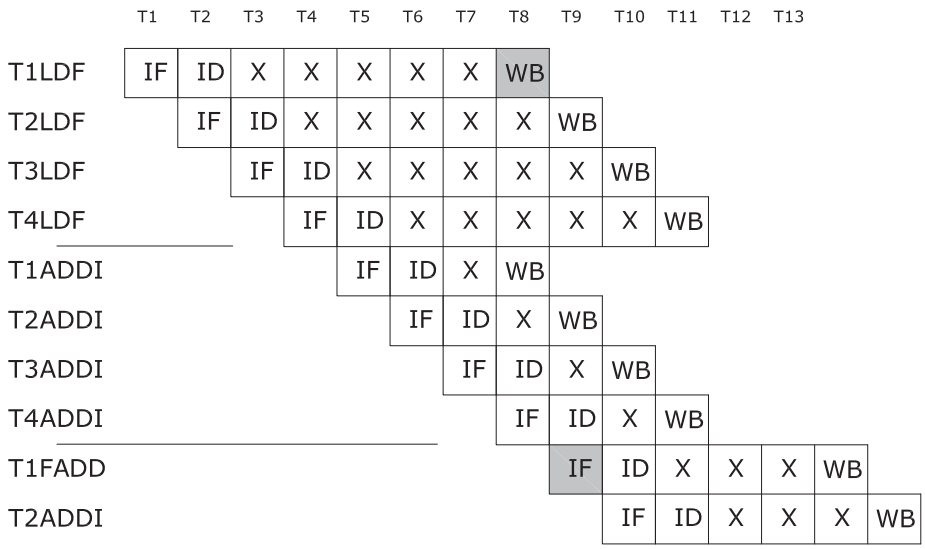
11.6 Per quanto si riferisce alla predisposizione dei bit di MASK a 1 quando il corrispondente `A[i]>1`, basta che la rete porti in MASK[i] il contenuto del bit di segno di A[i] complementato, in modo che MASK[i] diventi 1 se `A[i]>1` (ovvero se il bit di segno di A[i] è 0). Si può ipotizzare che il repertorio preveda un'istruzione che svolge tale funzione.

Per quanto si riferisce alla copiatura di B[i] in A[i] con mascheramento, assumendo che questa venga effettuata con un'istruzione tipo `VLM RVa,RVb`, si tratta di prevedere una porta, per ciascun elemento, che non faccia effettuare il trasferimento (ad esempio annullando il clock all'elemento i di RVa) se MASK[i] è 0.

11.10 L'istruzione FLD richiede 8 clock. La figura che segue mostra quale sarebbe il contenuto della pipeline per effetto delle prime due istruzioni; la gestione a grana fine farebbe il fetch ed emetterebbe prima FLD del thread T1 e successivamente FLD del thread T2 (nella figura e in quella che segue si è adottata la convenzione di indicare con Tixx, l'istruzione xx del thread i).

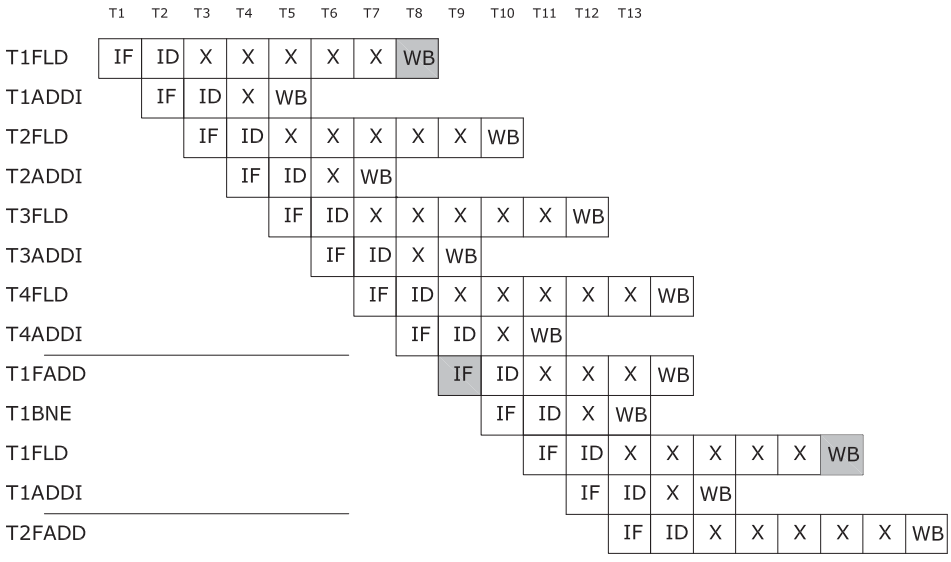
	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
T1FLD	IF	ID	X	X	X	X	X	X	WB	
T2FLD		IF	ID	X	X	X	X	X	X	WB

Si deve stabilire quanti thread danno luogo alla massima utilizzazione del processore. A tale scopo, con le ipotesi fatte circa il funzionamento della macchina, è necessario che l'istruzione FADD, che usa F2 come sorgente, abbia la fase di ID dopo la fase WB di FLD (dello stesso thread). Ciò si ottiene facendo in modo che il fetch dell'istruzione FADD venga ritardato quanto serve a riempire tale intervallo. Tenuto conto di come avviene la gestione, l'istruzione FADD di T1 potrà avvenire solo dopo la ADDI del task n. Ciò conduce allo schema di della figura che segue.



Come si vede vengono eseguite le prime due istruzioni di ciascun thread. L'istruzione FADD di T1 legge la sorgente F1 in fase ID, essendo F1 stato scritto dalla FLD di T1 due clock prima. In conclusione, il numero n cercato è 4. Si noti che con 3 thread l'ID della FADD di T1 coinciderebbe con il WB di FLD dello stesso thread.

11.11 L'esercizio si risolve tenendo conto che se il processore resta sullo stesso thread fino a che non si manifesta un conflitto. Inizialmente possono essere emesse le prime due istruzioni di T1, successivamente verranno emesse le prime due istruzioni di T2. Per arrivare all'esecuzione di FADD di T1 occorre anche in questo caso che i thread siano 4. Dopo che viene eseguita l'istruzione FADD ogni thread esegue tutte le altre 3 istruzioni del ciclo, come schematizzato nella figura che segue.



11.12 Se l'operazione FLD ha un solo ciclo X, due thread sono sufficienti a mantenere il processore completamente impegnato. Si può verificare che il fetch della FADD di T1 avviene subito dopo il WB della FLD del medesimo T1.