

Architetture dei Sistemi di Elaborazione: Guida agli esercizi

Pietro Pantani

Sommario

Introduzione.....	1
1 Pipeline.....	2
1.1 Lettura della traccia.....	2
1.2 Procedimento.....	2
1.3 Stalli	3
1.4 Forwarding	3
1.5 Calcolo finale	3
1.6 Possibili domande di teoria	3
1.6.1 Hazard.....	4
1.6.2 Cammini di forwarding	5
1.6.3 Introduzione unità Pipelined	6
1.6.4 Architettura Harvard.....	6
1.6.5 Missed Prediction Ratio	6
1.6.6 Branch Delay Slot.....	6
1.6.7 Eccezione	6
1.7 Esempi.....	7
1.7.1 Esame 2024-02-08, turno 1	7
1.7.2 Esame 2023-02-06, turno 3.....	8
2 Tomasulo.....	10
2.1 Lettura della traccia.....	10
2.2 Procedimento.....	10
2.3 Vincoli.....	10
2.3.1 Dati non disponibili	10
2.3.2 Più del massimo di commit contemporaneamente	11
2.3.3 Più del massimo di scritture nel CDB contemporaneamente	11
2.3.4 Utilizzo della stessa unità funzionale contemporaneamente	11
2.3.5 Unità funzionale non pipelined.....	11
2.4 Domande di teoria	11
2.4.1 Dimensione ROB.....	11
2.4.2 Singolo CDB.....	11
2.4.3 Duplicazione unità funzionali	11
2.4.4 Introduzione unità Pipelined	11
2.5 Esempi.....	12
2.5.1 Esempio visto a lezione	12
2.5.2 Esame 2023-02-06, turno 3.....	13
2.5.3 Esame 2023-02-06, Turno 4	15
3 BPU con BHT.....	17
3.1 Lettura della traccia.....	17
3.2 Procedimento.....	17
3.3 Calcolo finale	17
3.4 Domande di teoria	17
3.5 Esempi.....	17

4	<i>BPU con Shift Register</i>	18
4.1	Procedimento	18
4.2	Calcolo finale	18
4.3	Domande di teoria	18
4.4	Esempi	18
5	<i>Ottimizzazione</i>	19
5.1	Procedimento	19
5.2	Speedup	19
5.1	Domande di teoria	19
5.2	Esempi	19
6	<i>Cache</i>	20
6.1	Esempi	20

Introduzione

In questo documento sono spiegate le sei tipologie di esercizi possibili per l'esame di Architetture dei Sistemi di elaborazione.

L'esame presenta due esercizi delle tipologie spiegate all'interno del documento, ognuno accompagnato da una domanda di teoria inerente all'esercizio.

Ogni esercizio vale 10 punti, ogni domanda vale 5 punti.

Generalmente l'esame ha come primo esercizio la pipeline, e come secondo il più frequente è Tomasulo. In linea generale, al primo appello di gennaio/febbraio esce sempre Tomasulo come secondo esercizio, mentre negli altri è abbastanza casuale. Quando non esce Tomasulo generalmente si alternano le tipologie rimanenti.

L'esercizio sulla Cache non è mai uscito, ma l'esame ha presentato variazioni alcune volte (una volta, ad esempio, il primo esercizio non era la pipeline), per cui meglio prepararsi.

Ogni tipologia di esercizio è accompagnata da:

- Esempi svolti, presi da lezioni o da temi d'esame, e commentati. Non posso garantire che siano tutti corretti, per cui vi chiedo di farmi notare eventuali errori.
- Possibili domande di teoria, prese da temi d'esami precedenti, con risposte e riferimenti alla teoria.

Questo documento non è da intendersi come un riferimento affidabile al 100% per la teoria e per gli esercizi, ma come un insieme di linee guida. La terminologia utilizzata potrebbe essere impropria. Qualora venissero riscontrati errori nella teoria e nella risoluzione degli esercizi, non esitate contattarmi.

1 Pipeline

La Pipeline è la tipologia di esercizio più frequente, in tutti i temi d'esame è capitato solo una volta che non uscisse, per cui è l'esercizio da imparare a fare assolutamente.

1.1 Lettura della traccia

La traccia si apre con una serie di caratteristiche del processore. Le caratteristiche sono:

- Durata di operazioni ALU su Integer (quasi sicuramente 1 ciclo di clock)
- Durata di operazioni sulla memoria (quasi sicuramente 1 ciclo di clock)
- Durata di moltiplicazioni floating point (variabile) e se la moltiplicazione tra floating point è pipelined o meno
- Durata di divisioni floating point (variabile) e se la divisione tra floating point è pipelined o meno
- Durata di operazioni aritmetiche floating point (variabile)
- Branch Delay Slot (generalmente disabilitato)
- Forwarding (quasi sicuramente abilitato)
- Possibilità di completare le operazioni non in ordine (quasi sicuramente possibile)

Dopo queste informazioni, è presente il codice in C. Da questo frammento di codice bisogna solo vedere il numero di esecuzioni del ciclo, così da poter effettuare il calcolo finale.

Infine, abbiamo il codice in assembly in un'apposita tabella, dove poter compilare la pipeline.

1.2 Procedimento

Ogni istruzione è composta da 5 fasi:

- F – Fetch
- D - Decode
- E - Execute
- M - Memory
- W – Write Back

Per ogni istruzione va scritta la durata di ognuna di queste fasi, indicandone ognuna con la relativa lettera.

Bisogna poi segnare quanti cicli di clock sono necessari per ogni istruzione (ossia il numero di cicli necessari per terminare l'istruzione rispetto a quella precedente), e calcolare il numero di cicli necessari all'esecuzione dell'intero programma.

1.3 Stalli

Per motivi vari, è possibile che le istruzioni vadano stallate, interrompendone l'esecuzione per uno o più cicli di clock.

Esistono due tipi di stalli:

- Stallo dovuto all'attesa di dati, indicato con una S maiuscola.
- Stallo strutturale, indicato con una s minuscola.

Uno stallo si propaga su tutte le istruzioni sottostanti, ritardando quindi l'esecuzione di tutte, a meno che lo stallo non avvenga in una fase non condivisibile.

Abbiamo uno stallo strutturale quando due istruzioni cercano di eseguire una stessa istruzione contemporaneamente, per cui quella di sotto stalla. I casi in cui avviene sono generalmente nella fase M o nella fase E di unità non pipelined, siccome unità aritmetiche pipelined o unità aritmetiche diverse possono essere nella stessa fase contemporaneamente.

Le operazioni aritmetiche, per via della mancanza dei dati, stallano PRIMA dell'inizio della fase E.

Le operazioni store (s.d) stallano DOPO la fase E, siccome nella fase E controllano semplicemente gli indirizzi di memoria. La scrittura dei dati avviene successivamente.

I salti condizionati verificano la condizione durante la fase D, per cui stallano dopo la fase D fino alla presenza del dato da verificare.

Il caso più frequente in cui uno stallo non si propaga è il seguente:

Una store, se sta attendendo il risultato di un'operazione, stalla tra la fase E e la fase M. Un'operazione successiva, quindi, può tranquillamente entrare in fase E.

1.4 Forwarding

Senza il forwarding, il dato sarebbe presente nel registro dove va memorizzato solo dopo la fase WB. Con il forwarding è possibile "inoltrare" il dato direttamente all'operazione aritmetica successiva.

Il forwarding è possibile:

- Dalla fase E di operazioni aritmetiche di qualunque tipo alla fase E di:
 - Altre operazioni aritmetiche
 - Calcolo di differenza di indirizzi (esempio: s.d. $f1, v1(r1)$, se l'operazione prima è un'operazione aritmetica su $r1$, il risultato viene inoltrato direttamente a quest'istruzione, senza causare ritardi).
- Dalla fase M di operazioni aritmetiche alla fase M di un'operazione Store.
- Dalla fase M di operazioni Load alla fase E di operazioni aritmetiche

1.5 Calcolo finale

Una volta calcolato il numero di cicli di clock necessari per un'esecuzione, bisogna sommare il numero di cicli, moltiplicarli per il numero di esecuzioni del ciclo for, e sommare i cicli necessari per l'esecuzione delle istruzioni esterne al for, quindi eseguite una sola volta.

1.6 Possibili domande di teoria

In questa sezione sono state raccolte le tipologie di domande di teoria capitate in passato.

1.6.1 Hazard

Gli hazard sono proprietà dell'organizzazione della pipeline, a differenza delle dipendenze che sono proprietà del programma.

Gli stalli dipendono dal programma e dalla pipeline: una dipendenza può causare un hazard, un hazard può causare uno stallo (non necessariamente).

Gli hazard possibili sono i seguenti:

- **RAW (Read After Write)**

- Sono gli hazard più comuni, e corrispondono ad una dipendenza tra i dati. Si hanno quando un'operazione cerca di leggere in un registro prima che il suo valore sia disponibile.

Questo tipo di hazard viene risolto tramite l'introduzione di stalli, e tramite forwarding.

Esempio:

```
DADD R1, R2, R3
```

```
DSUB R4, R5, R1
```

In questo frammento di codice DSUB cerca di leggere R1 prima che DADD vi scriva il risultato.

- **WAW (Write After Write)**

- Sono possibili se:
 - Le istruzioni scrivono in più fasi
Oppure
 - Un'istruzione può procedere anche se un'istruzione precedente è stata stallata o sta venendo processata in una stessa fase per più di un colpo di clock

Esempio:

```
DADDUI R4, R5, R6
```

```
MUL R4, R7, R9
```

In questo frammento di codice sia DADDUI che MUL scrivono su R4. Qualora l'esecuzione della DADDUI richiedesse più tempo dell'esecuzione della MUL, al termine del programma R4 conterrebbe il risultato della DADDUI e non della MUL.

- **WAR (Write After Read)**

- Sono molto rari, e sono possibili se ci sono istruzioni che leggono presto nella pipeline, e altre che leggono tardi.

Questo tipo di hazard può essere risolto tramite il Register Renaming.

Nell'architettura di Tomasulo, questa soluzione è implementata a livello hardware.

Esempio:

```
DADDUI R5, R6, R7
```

```
SUB R6, R9, R10
```

In questo frammento di codice sia SUB scrive sul registro R6, che sta venendo letto dalla DADDUI. Ciò potrebbe causare problemi se la SUB dovesse essere eseguita prima della DADDUI, siccome l'operazione verrebbe effettuata con dati errati.

Generalmente viene chiesto che tipo di hazard viene creato da una particolare coppia di istruzioni.

Esempi di domande

- 1) Considerando la coppia di istruzioni:

div.d f3,f1,f2

l.d f1,v3(r1)

che tipo di hazard crea l'utilizzo di f1 e come viene risolto? motivare la risposta.

L'hazard creato è di tipo WAR, siccome l.d scrive sul registro f1 dopo la sua lettura in div.d. Può essere risolto tramite la tecnica di register renaming.

- 2) Considerando la coppia di istruzioni:

add.d f4,f1,f2

div.d f4,f4,f3

che tipo di hazard crea l'utilizzo di f4 e come viene risolto? motivare la risposta.

L'hazard creato è di tipo RAW, siccome add.d scrive il suo risultato nel registro f4, letto immediatamente dopo da div.d. Si risolve introducendo stalli tra la fase D e la fase E della div.d, e grazie al forwarding viene ridotto il numero di stalli richiesti.

1.6.2 Cammini di forwarding

In questo tipo di domanda viene chiesto quali cammini di forwarding partecipano all'esecuzione di una determinata istruzione.

Esempi di domande

- 1)

mul.d f4,f1,f2
l.d f3,v3(r1)
mul.d f4,f4,f3

Considerando il programma precedente, e in particolare la istruzione: mul.d f4,f4,f3 quali potrebbero essere i cammini di forwarding che potrebbero partecipare alla sua esecuzione? Perché dovrebbero essere attivati? motivare la risposta.

Il percorso di forwarding utilizzato riguarda quello tra la fase di EX/MEM, ovvero quello riguardante i risultati della fase di esecuzione. Questo perché prima abbiamo l'istruzione mul.d f4,f1,f2 che calcola il valore di f4 (caso di WAW e RAW). Se non fosse attivo questo percorso di forwarding bisognerebbe attendere che il valore di f4 sia prima scritto, quindi attendere la fase di write di mul.d f4,f1,f2. Ciò costerebbe 2 c.c. in più (per via del RAW hazard).

- 2) Considerando il programma precedente, e in particolare la coppia di istruzioni

l.d f2,v2(r1)

mul.d f6,f2,f3

come viene attivato e qual è il cammino di forwarding che partecipa alla loro esecuzione? motivare la risposta.

Il cammino di forwarding che viene attivato è quello fra la fase di MEM e la fase di EXE del moltiplicatore, e viene rilevato durante la fase di decodifica dell'istruzione di moltiplicazione.

1.6.3 Introduzione unità Pipelined

1.6.4 Architettura Harvard

L'architettura Harvard fa riferimento all'esistenza di due memorie separate tra dati e codice, andando direttamente in contrasto con quella che era l'architettura di Von Neumann.

Generalmente viene chiesto quali coppie di istruzioni beneficino dell'architettura Harvard.

Per rispondere bisogna selezionare le coppie di istruzioni tali che:

- 1) la prima sia una load o una store
- 2) durante la fase M della prima avvenga la F della seconda.

1.6.5 Missed Prediction Ratio

1.6.6 Branch Delay Slot

1.6.7 Eccezione

Considerando il programma precedente, si consideri che durante la prima iterazione del codice sia scatenata una eccezione da parte della prima delle istruzioni di divisione. In particolare, l'eccezione viene intercettata dal sistema all'ultimo clock cycle della fase di esecuzione (EXE) della divisione. Che conseguenze si osserverebbero sull'esecuzione del codice? motivare la risposta

- 1) una trap viene caricata (fetch stage) nel ciclo successivo
- 2) all'istruzione che ha generato la exception e quelle successive viene impedito di scrivere in memoria
- 3) l'handler effettua delle operazioni, prima tra tutte salva il PC dell'istruzione che ha generato l'exception. Alla fine può o terminare l'esecuzione o riprendere dall'istruzione

Esempi di domande

- 1) Considerando il programma precedente, quale sarebbe il tempo di esecuzione del programma se il processore avesse abilitato il Branch Delay slot? motivare la risposta.

Viene eseguita una sola iterazione del programma, in quanto l'istruzione di Halt viene eseguita completamente e il programma si ferma alla fine della prima iterazione. Il nuovo tempo di esecuzione del programma è di (cicli per una sola iterazione) cicli di clock.

1.7.1 Esame 2024-02-08, turno 1

- Integer ALU: 1 clock cycle
- Data memory: 1 clock cycle
- FP multiplier unit: pipelined 6 stages
- FP divider unit: not pipelined unit that requires 6 clock cycles
- FP arithmetic unit: pipelined 4 stages
- branch delay slot: 1 clock cycle, and the branch delay slot disabled
- forwarding enabled
- it is possible to complete instruction EXE stage in an out-of-order fashion.

```

;   for (i = 0; i < 100; i++) {
;       v4[i] = v1[i]/v2[i] * v3[i];
;   }

```

Tutte le istruzioni fino a $\text{l.d } f2, v2(r1)$ vengono eseguite normalmente.

In l.d f1, v3(r1) abbiamo solo lo stallo propagato dalla div subito dopo la fase F.

7

In s.d f3, v4(r1), abbiamo 4 stalli dopo la F dovuti agli stalli della moltiplicazione precedente, e degli stalli dopo la fase E siccome, dovendo memorizzare il risultato della moltiplicazione, bisogna attendere la fine di questa.

La daddui e la daddi proseguono normalmente tranne che per gli stalli propagati dalla s.d.

In bnez r2, loop abbiamo uno stallo dopo la fase D, siccome si verifica che il dato diverso da zero sia r2, che sta venendo calcolato nell'istruzione precedente.

L'ultima istruzione, la HALT, ha solo uno stallo propagato dalla bnez.

Domanda di teoria:

Considerando il programma precedente, e in particolare la coppia di istruzioni:

```
div.d f3, f1, f2
l.d f1, v3(r1)
```

che tipo di hazard crea l'utilizzo di f1 e come viene risolto? motivare la risposta.

L'utilizzo di f1 causa un hazard di tipo WAR, Write After Read. Il problema sorge perché l'istruzione l.d potrebbe sovrascrivere il valore del registro f1 che viene utilizzato nella div, per cui la divisione potrebbe dare un risultato errato.

Potrebbe venire risolto dal compilatore con la tecnica del Register Renaming, che prevede la sostituzione del registro f1 con un registro inutilizzato a partire dalla l.d e per tutte le istruzioni che lo utilizzano.

1.7.2 Esame 2023-02-06, turno 3

Considerando il processore MIPS64 e l'architettura descritta in seguito:

- Integer ALU: 1 clock cycle
- Data memory: 1 clock cycle
- FP multiplier unit: not pipelined unit that requires 6 stages
- FP divider unit: not pipelined unit that requires 6 clock cycles
- FP arithmetic unit: pipelined 4 stages
- branch delay slot: 1 clock cycle, and the branch delay slot disabled
- forwarding enabled
- it is possible to complete instruction EXE stage in an out-of-order fashion.

Usando il frammento di codice riportato, si calcoli il tempo di esecuzione dell'intero programma in colpi di clock e si completi la seguente tabella.

```
; for (i = 0; i < 100; i++) {
;     v4[i] = v1[i]*v2[i]* v3[i];
; }
```


2 Tomasulo

2.1 Lettura della traccia

Nella traccia la prima cosa che leggeremo è il numero di iterazioni che bisognerà calcolare, dopodiché avremo i dettagli sull'architettura del processore. In particolare:

- Quante istruzioni per ciclo di clock vengono issued (tipicamente 2)
- Quante issue sono richieste per un salto (tipicamente 1)
- Quanti commit possono avvenire contemporaneamente (tipicamente 2)
- Durata in cicli di clock nella fase EXE di:
 - Accesso alla memoria
 - Operazioni ALU su interi
 - Salti
 - Moltiplicazione tra Float
 - Divisione tra Float
 - Altre operazioni tra Float
- Se i branch sono predetti correttamente o meno (tipicamente sono sempre tutti predetti correttamente)
- Se ci sono cache misses (tipicamente no)
- Il numero di Common Data Bus (CDB) (tipicamente 2)

2.2 Procedimento

Per ogni istruzione dovremmo scrivere:

- Colpo di clock in cui vengono issued
- Colpo d'inizio dell'esecuzione
- Colpo d'inizio della fase MEM
- Colpo in cui il risultato viene scritto nel CDB
- Colpo in cui avviene il commit

Non tutte le operazioni fanno tutte le fasi, in particolare:

- Operazioni Load: Tutti gli step
- Operazioni aritmetiche: Tutti gli step tranne MEM
- Branch e Store: Solo EXE e Commit

Per ogni istruzione sappiamo la durata dalla traccia.

L'issue è la sezione più semplice: se, ad esempio, il massimo di issue per colpo di clock è 2, scriviamo tutti i numeri partendo da 1 per due volte, fino ad arrivare al salto. Una volta arrivati al salto, il numero salta a quello successivo.

2.3 Vincoli

Di seguito sono illustrate le circostanze in cui alcune istruzioni vanno stallate.

2.3.1 Dati non disponibili

Le operazioni hanno lo stesso cammino dell'architettura pipelined: le operazioni aritmetiche e i salti condizionali non possono iniziare finché non hanno i dati, mentre le load e le store

possono sempre iniziare la fase EXE e devono aspettare che siano disponibili solo per il commit.

Le operazioni aritmetiche cominciano la EXE dopo la scrittura nel CDB delle load dei dati che sfruttano, così come i branch.

2.3.2 Più del massimo di commit contemporaneamente

Nella traccia viene specificato il numero di commit possibili in un solo ciclo di clock. Una volta raggiunto il massimo, bisogna effettuare il commit al colpo di clock successivo.

2.3.3 Più del massimo di scritture nel CDB contemporaneamente

Nella traccia viene specificato il numero di scritture nel CDB possibili in un solo ciclo di clock. Una volta raggiunto il massimo, bisogna scrivere nel CDB al colpo di clock successivo.

2.3.4 Utilizzo della stessa unità funzionale contemporaneamente

Un'unità funzionale non può svolgere più operazioni contemporaneamente.

È buona abitudine scrivere quale unità funzionale si sta sfruttando in fase EXE, siccome la stessa unità non può effettuare la stessa operazione nello stesso colpo di clock, ma diverse unità possono.

2.3.5 Unità funzionale non pipelined

Un'unità funzionale pipelined può eseguire più operazioni contemporaneamente, con un solo colpo di clock di differenza, ma un'unità non pipelined deve attendere il termine dell'istruzione precedente.

2.4 Domande di teoria

2.4.1 Dimensione ROB

La struttura della domanda è: considerando il programma precedente, se assumiamo che il ROB ha una dimensione di n elementi, quale sarebbe la prima istruzione che dovrebbe stallare all'esecuzione del programma?

Per rispondere, bisogna vedere l' n -esima istruzione, e vedere il ciclo di clock x in cui avviene l'issue.

A questo punto, bisogna vedere se arrivati a quel colpo di clock si sono liberate delle istruzioni. Se arrivati al ciclo di clock x sono già avvenuti dei commit, abbiamo tanti slot liberi quanti sono i commit avvenuti.

La prima istruzione che stalla è quella che avviene quando tutti gli slot del ROB sono pieni.

Se il commit e l'issue avvengono nello stesso colpo di clock, lo slot conta come occupato.

2.4.2 Singolo CDB

2.4.3 Duplicazione unità funzionali

2.4.4 Introduzione unità Pipelined

2.5 Esempi

Ad ogni esempio è stato allegato a destra un commento, non presente nel tema d'esame, per chiarire determinati passaggi.

2.5.1 Esempio visto a lezione

# iteration		Issue	EXE	MEM	CDB x2	COMMIT x2	Commento
1	LD R2, 0(R1)	1	2 ma	3	4	5	Tutto normale
1	DADDIU R2, R2, #1	1	5 i		6	7	Comincia al colpo di clock 5 siccome R2 è disponibile dopo il 4° colpo di clock
1	SD R2, 0(R1)	2	3 ma			7	Tutto normale
1	DADDIU R1, R1, #4	2	3 i		4	8	Tutto normale
1	BNE R2, R3, LOOP	3	7 j			8	La fase EXE attende R2, disponibile dopo il 6° ciclo di clock
2	LD R2, 0(R1)	4	5 ma	6	7	9	Tutto normale
2	DADDIU R2, R2, #1	4	8 i		9	10	Comincia al colpo di clock 8 siccome R2 è disponibile dopo il 7° colpo di clock
2	SD R2, 0(R1)	5	6 ma			10	Tutto normale
2	DADDIU R1, R1, #4	5	6 i		7	11	Tutto normale
2	BNE R2, R3, LOOP	6	10 j			11	La fase EXE attende R2, disponibile dopo il 9° ciclo di clock
3	LD R2, 0(R1)	7	8 ma	9	10	12	Tutto normale
3	DADDIU R2, R2, #1	7	11 i		12	13	Comincia al colpo di clock 5 siccome R2 è disponibile dopo il 10° colpo di clock
3	SD R2, 0(R1)	8	9 ma			13	Tutto normale
3	DADDIU R1, R1, #4	8	9 i		10	14	Tutto normale
3	BNE R2, R3, LOOP	9	13 j			14	La fase EXE attende R2, disponibile dopo il 12° ciclo di clock

2.5.2 Esame 2023-02-06, turno 3

Considerando il programma precedente e l'architettura del processore superscalare descritto in seguito; completare la tabella relativa alle prime 3 iterazioni.

Processor architecture:

- Issue 2 instructions per clock cycle
- jump instructions require 1 issue
- handle 2 instructions commit per clock cycle
- timing facts for the following separate functional units:
 - i. 1 Memory address 1 clock cycle
 - ii. 1 Integer ALU 1 clock cycle
 - iii. 1 Jump unit 1 clock cycle
 - iv. 1 FP multiplier unit, which is not pipelined: 6 stages
 - v. 1 FP divider unit, which is not pipelined: 6 clock cycles
 - vi. 1 FP Arithmetic unit, which is pipelined: 4 stages
- Branch prediction is always correct
- There are no cache misses
- There are 2 CDB (Common Data Bus).

# iteration		Issue	EXE	MEM	CDB x2	COMMIT x2	Commento
1	l.d f1,v1(r1)	1	2 ma	3	4	5	Tutto normale
1	l.d f2,v2(r1)	1	3 ma	4	5	6	EXE stalla e inizia al ciclo 3
1	mul.d f4,f1,f2	2	6 m		12	13	Comincia al ciclo 6 siccome bisogna attendere che f1 e f2 siano caricati, in questo caso che il valore di f2 venga scritto nel CDB.
1	l.d f3,v3(r1)	2	4 ma	5	6	13	Comincia al ciclo 4 siccome il ciclo 3 è già occupato per la memory access
1	mul.d f4,f4,f3	3	13 m		19	20	Comincia al ciclo 13 perché f4 è disponibile al termine della moltiplicazione precedente, al ciclo 12
1	s.d f4,v4(r1)	3	5 ma			20	Comincia al ciclo 5 siccome il ciclo 4 è già occupato per la memory access
1	daddi r2,r2,-1	4	5 i		6	21	Nessuna difficoltà
1	daddui r1,r1,8	4	6 i		7	21	Comincia al ciclo 6 siccome il ciclo 5 è già occupato per la somma di integer

1	bnez r2,loop	5	6 j			22	Nessuna difficoltà
2	l.d f1,v1(r1)	6	8 ma	9	10	22	Comincia al ciclo 8 siccome r1 diventa disponibile dalla daddui nel ciclo 7
2	l.d f2,v2(r1)	6	9 ma	10	11	23	EXE stalla e inizia al ciclo 9
2	mul.d f4,f1,f2	7	20 m		26	27	
2	l.d f3,v3(r1)	7	10 ma	11	12	27	
2	mul.d f4,f4,f3	8	27 m		33	34	
2	s.d f4,v4(r1)	8	11 ma			34	
2	daddi r2,r2,-1	9	10 i		11	35	
2	daddui r1,r1,8	9	11 i		13	35	
2	bnez r2,loop	10	11 j			36	
3	l.d f1,v1(r1)	11	14 ma	15	16	36	
3	l.d f2,v2(r1)	11	15 ma	16	17	37	
3	mul.d f4,f1,f2	12	34 m		40	41	
3	l.d f3,v3(r1)	12	16 ma	17	18	41	
3	mul.d f4,f4,f3	13	41 m		47	48	
3	s.d f4,v4(r1)	13	17 ma			48	
3	daddi r2,r2,-1	14	15 i	17	18	49	
3	daddui r1,r1,8	14	16 i	18	19	49	
3	bnez r2,loop	15	16 j			50	

2.5.3 Esame 2023-02-06, Turno 4

Considerando il programma precedente e l'architettura del processore superscalare descritto in seguito; completare la tabella relativa alle prime 2 iterazioni.

Processor architecture:

- Issue 2 instructions per clock cycle
- jump instructions require 1 issue
- handle 2 instructions commit per clock cycle
- timing facts for the following separate functional units:
 - i. 1 Memory address 1 clock cycle
 - ii. 1 Integer ALU 1 clock cycle
 - iii. 1 Jump unit 1 clock cycle
 - iv. 1 FP multiplier unit, which is pipelined: 8 stages
 - v. 1 FP divider unit, which is not pipelined: 8 clock cycles
 - vi. 1 FP Arithmetic unit, which is pipelined: 4 stages
- Branch prediction is always correct
- There are no cache misses
- There are 2 CDB (Common Data Bus).

# iteration		Issue	EXE	MEM	CDB x2	COMMIT x2	Commento
1	l.d f1,v1(r1)	1	2 ma	3	4	5	
1	l.d f2,v2(r1)	1	3 ma	4	5	6	
1	div.d f4,f1,f2	2	6 d		14	15	
1	l.d f3,v3(r1)	2	4 ma	5	6	15	
1	add.d f5,f4,f3	3	15 a		19	20	
1	l.d f4,v4(r1)	3	5 ma	6	7	20	
1	add.d f5,f4,f5	4	20 a		24	25	
1	s.d f5,v5(r1)	4	6 ma			25	
1	daddui r1,r1,8	5	6 i		7	26	
1	daddi r2,r2,-1	5	7 i		8	26	
1	bnez r2,loop	6	9 j			27	
2	l.d f1,v1(r1)	7	8 ma	9	10	27	
2	l.d f2,v2(r1)	7	9 ma	10	11	28	
2	div.d f4,f1,f2	8	14 d		22	28	
2	l.d f3,v3(r1)	8	10 ma	11	12	29	

2	add.d f5,f4,f3	9	23 a		27	29	
2	l.d f4,v4(r1)	9	11 ma	12	13	30	
2	add.d f5,f4,f5	10	28 a		32	33	
2	s.d f5,v5(r1)	10	12 ma			33	
2	daddui r1,r1,8	11	12 i		13	34	
2	daddi r2,r2,-1	11	13 i		14	34	
2	bnez r2,loop	12	15 j			35	

3 BPU con BHT

3.1 Lettura della traccia

3.2 Procedimento

Per ogni branch:

- 1) Si controlla lo stato della BHT. A seconda del numero si prevede se il salto verrà preso o meno:
 - Valore 0 o 1: salto non preso
 - Valore 2 o 3: salto preso
- 2) Si valuta se il salto viene preso o meno.
- 3) Verifica della correttezza della predizione:
 - La predizione corretta: non cambia nulla
 - La predizione errata: si incrementa il Missed Predictions Counter relativo a quell'operazione
- 4) Aggiornamento del valore della BHT:
 - Salto preso: il valore incrementa (fino ad un massimo di 3)
 - Salto non preso: il valore decrementa (fino ad un minimo di 0)

Si ricomincia da 1 fino al termine del numero delle esecuzioni.

3.3 Calcolo finale

$$\text{Misprediction ratio} = \frac{\text{Number of mispredicted branches}}{\text{Total branches}}$$

3.4 Domande di teoria

3.5 Esempi

4 BPU con Shift Register

4.1 Procedimento

Per ogni branch

- 1) Si controlla il valore dello Shift Register (comune a tutte le istruzioni)
- 2) Si controlla lo stato del predittore corrispondente al valore indicato sullo Shift Register.
A seconda del numero si prevede se il salto verrà preso o meno:
 - Valore 0 o 1: salto non preso
 - Valore 2 o 3: salto preso
- 3) Si valuta se il salto viene preso o meno.
- 4) Verifica della correttezza della predizione:
 - La predizione corretta: non cambia nulla
 - La predizione errata: si incrementa il Missed Predictions Counter relativo a quell'operazione
- 5) Aggiornamento del valore dello Shift Register:
 - Salto preso: il valore incrementa (fino ad un massimo di 3) e si pusha 1 nello shift register da destra a sinistra (00 → 01, 10 → 01, 01 → 11, 11 → 11)
 - Salto non preso: il valore decrementa (fino ad un minimo di 0) e si pusha 0 nello shift register da destra a sinistra (00 → 00, 10 → 00, 01 → 10, 11 → 10)

Si ricomincia da 1 fino al termine del numero delle esecuzioni.

4.2 Calcolo finale

Misprediton ratio = Number of mispredicted branches / Total branches

4.3 Domande di teoria

4.4 Esempi

5 Ottimizzazione

5.1 Procedimento

5.2 Speedup

Le formule per calcolare gli speedup sono le seguenti:

Speedup enhanced = somma dei CC delle istruzioni che hanno variato i loro CC nell'esecuzione precedente (singolo ciclo) / somma di tutti i CC di tutte le stesse istruzioni nell'esecuzione speeduppata.

Fraction Enhanced = somma di tutti i CC di tutte le istruzioni che hanno variato i loro CC(nell'istruzione precedente) *N istruzioni loop/ CC totali del programma precedente

$$\text{SPEEDUP COMPUTATION} = \frac{1}{(1 - \text{FRACtion ENHANCED}) + \left(\frac{\text{FRACtion ENHANCED}}{\text{SPEEDUP ENHANCED}} \right)}$$

$$\text{SPEEDUP OBSERVATION} = \frac{\text{CC totali attuali}}{\text{CC totali precedenti}}$$

5.1 Domande di teoria

5.2 Esempi

6 Cache

6.1 Esempi