Tutorato Architettura degli Elaboratori Modulo 2
Lezione 1

Davide Cecchini

17 Marzo 2022

1 Instruction Level Parallelism

La semplice pipeline usata per eseguire il set di istruzioni ristretto (`lw,sw,add,or,beq,slt`) del nostro processore MIPS è composta da 5 stadi:

1. `IF` : Instruction fetch (memoria istruzioni)
2. `ID` : Instruction decode e lettura registri
3. `EXE` : Esecuzione istruzioni e calcolo indirizzi
4. `MEM` : Accesso alla memoria (memoria dati)
5. `WB` : Write back (scrittura del registro risultato, calcolato in `EXE` o `MEM`)

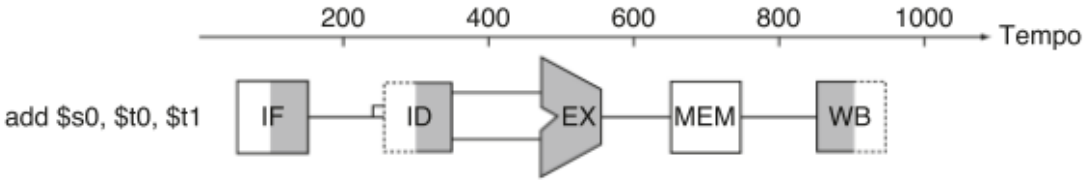


Figure 1: Rappresentazione grafica di una istruzione nella pipeline.

Dalle slide del corso:

- Istruzioni **aritmetiche**:
  - Producono il dato da scrivere nel register file al 3° ciclo (stadio EXE), durante il quale hanno bisogno dei dati corretti in ingresso nell'ALU.
- Istruzioni **load**:
  - Calcolano l'indirizzo al 3°ciclo, durante il quale hanno bisogno del dato corretto in ingresso nell'ALU.
  - Producono il dato da scrivere nel register file al 4° ciclo (stadio MEM).
- Istruzioni **branch**:
  - Producono il dato da scrivere nel PC al 2° ciclo (stadio ID), durante il quale hanno bisogno dei registri corretti in ingresso al comparatore.
- Istruzioni **store**:
  - Calcolano l'indirizzo al 3°ciclo, durante il quale hanno bisogno del dato corretto in ingresso nell'ALU.
  - Al 4° ciclo hanno bisogno del dato corretto (da memorizzare) in ingresso alla MEMORIA.

Esercizio 1

Considerare un processore con pipeline a 5 stadi senza forwarding, con un register file non ottimizzato[1]. Il processore è fornito di hazard detection unit, è quindi in grado di mettere in stallo la pipeline.

```
1 sub $3, $2, $5
2 lw $10, 4($3)
3 addi $3, $3, 8
4 add $20, $20, $10
```

Domande:

1. Determinare le dipendenze RAW tra le istruzioni del programma assembler precedente.
2. Disegnare il diagramma temporale di esecuzione.
3. Cosa succede all'8° ciclo di clock nei vari stadi?
4. Se il processore non fosse dotato di hazard detection unit, dove dovrebbero essere inserite le `nop` per evitare inconsistenze dovute alle dipendenze sui dati?

Soluzione domanda 1

Dipendenze:

- $1 \rightarrow 2$. L'istruzione 2 legge il registro \$3 precedentemente scritto dall'istruzione 1.
- $2 \rightarrow 4$. L'istruzione 4 legge il registro \$10 precedentemente scritto dall'istruzione 2.
- $(1 \rightarrow 3)$, Anche se in realtà, visto il tipo di processore, l'istruzione 3 entra in stallo a causa della dipendenza $(1 \rightarrow 2)$ e quando ne esce ha il dato disponibile senza ulteriori ritardi.

Soluzione domanda 2

		1	2	3	4	5	6	7	8	9	10	11	12	13
1	sub	IF	ID	EXE	MEM	WB								
2	lw		IF	ID				EXE	MEM	WB				
3	addi			IF				ID	EXE	MEM	WB			
4	add							IF	ID			EXE	MEM	WB

Soluzione domanda 3

All'ottavo ciclo i vari stadi eseguono quanto segue:

- `IF`: opera sull'eventuale quinta istruzione, di cui non sappiamo niente perchè non data.
- `ID`: esegue la decodifica della quarta istruzione (`add`).
- `EXE`: esegue la terza istruzione (`addi`).
- `MEM`: legge la memoria per la seconda istruzione (`lw`).
- `WB`: è in attesa (bolla).

[1] prima legge il vecchio valore di un registro e poi lo scrive al ciclo di clock successivo

Soluzione domanda 4

Se il processore non fosse dotato di hazard detection unit le `nop` andrebbero inserite come segue:

```
sub $3, $2, $5
nop
nop
nop
lw $10, 4($3)
addi $3, $3, 8
nop
nop
add $20, $20, $10
```

Notare che le `nop` inserite corrispondono alle bolle del diagramma temporale.

Esercizio 2

Si consideri il seguente programma.

```
1  ori $6, $0, 0
2  ori $7, $0, 1000
3  loop:
4      sll $5, $6, 2 #moltiplico per 4
5      add $8, $4, $5
6      lw $9, 0($8)
7      and $9, $9, $0
8      sw $9, 0($8)
9      addi $6, $6, 1
10     bne $6, $7, loop
```

Domande:

1. Assumendo che \$4 contenga l'indirizzo di un vettore di interi, cosa fa questo codice?
2. Determinare il CPI del programma nel caso in cui il processore sia multiciclo (si ipotizzi che la `lw` impieghi 5 cicli di clock, la `bne` 3 e il resto delle istruzioni 4).
3. Rispetto all'implementazione multiciclo a pipeline vista a lezione (5 stadi, forwarding e delayed branch), dove si verificano eventuali stalli? Inserire le istruzioni `nop` opportune, e ricalcolare il CPI, senza considerare i cicli persi per il riempimento della pipeline.

Soluzione domanda 1

Il programma pone a 0 i primi 1000 elementi di un vettore di interi.

Soluzione domanda 2

Per il processore multiciclo bisogna considerare che l'istruzione `bne` impiega 3 cicli, `lw` 5 cicli e tutte le altre istruzioni 4 cicli. Quindi abbiamo $2 * 4 = 8$ cicli per il preambolo del loop (2 istruzioni da 4 cicli ciascuna), e $5 * 4 + 1 * 5 + 1 * 3 = 28$ cicli per ogni iterazione del ciclo (5 istruzioni non `lw` e `bne` + 1 `lw` + 1 `bne`), per un totale di $28 * 1000 = 28000$ cicli per l'intero ciclo (clock per iterazione * N° iterazioni). In totale abbiamo quindi $8 + 28000 = 28008$ clock per l'intero programma.

Il numero di istruzioni eseguite è $IC = 2 + 7 * 1000 = 7002$ (istruzioni preambolo + istruzioni iterazione * 1000). Per cui abbiamo che:

$$CPI = \frac{Nclock}{Nistruzioni} = \frac{28008}{7002} = 4$$

Soluzione domanda 3

Dal diagramma temporale possiamo notare come ci siano due stalli, uno tra la `lw` e la `and` (dovuto alla dipendenza sul valore contenuto in \$9) e uno tra la `addi` e la `bne` (dovuto alla dipendeza sul valore contenuto in \$6). Quindi le `nop` dovute agli stalli sono due, una tra la `lw` e la `and` e una tra la `addi` e la `bne`; a queste va aggiunta la `nop` dopo la `bne`, dovuta al delayed branch. Le frecce in rosso nel diagramma mostrano il forwarding dei registri che determinano le dipendenze RAW.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
ori	if	id	exe	mem	wb											
ori		if	id	exe	mem	wb										
sll			if	id	exe	mem	wb									
add				if	id	exe	mem	wb								
lw					if	id	exe	mem	wb							
and						if	id	id	exe	mem	wb					
sw							if	if	id	exe	mem	wb				
addi									if	id	exe	mem	wb			
bne										if	id	id	exe	mem	wb	
nop											if	if	id	exe	mem	wb

Quindi il programma modificato è il seguente:

```
ori $6, $0, 0
ori $7, $0, 1000
loop:
    sll $5, $6, 2 #moltiplico per 4
    add $8, $4, $5
    lw $9, 0($8)
    nop
    and $9, $9, $0
    sw $9, 0($8)
    addi $6, $6, 1
    nop
    bne $6, $7, loop
    nop
```

Sono necessari quindi un numero di cicli pari a $2 + 10 * 1000 = 10002$ (N° cicli preamblo + N° cicli per iterazione * 1000 iterazioni). Per cui abbiamo che:

$$CPI = \frac{10002}{7002} = 1.428$$

N.B. Se non contiamo il tempo per riempire la pipeline, il numero di cicli "reali" per istruzione è 1. Questo perché ad ogni di ciclo di clock una istruzione termina l'esecuzione, avendo così, 10002 cicli totali.

Contrariamente a quanto può sembrare il numero di istruzioni non è 10002 ma bensì 7002. Il CPI è una statistica che calcoliamo prima che il compilatore inserisca le `nop`. Quindi, in parole povere, non le contiamo in quanto vengono inserite a posteriori dal compilatore e non rientrano nel calcolo della statistica.

Esercizio 3

La tabella in Fig. 2 descrive la funzione usuale dei registri ed il loro nome. Nei listati i registri possono venir chiamati col numero oppure col nome.

Nome	Numero	Utilizzo	Preservato durante le chiamate
$zero	0	costante zero	*Riservato MIPS*
$at	1	riservato per l'assemblatore	*Riservato Compiler*
$v0-$v1	2-3	valori di ritorno di una procedura	No
$a0-$a3	4-7	argomenti di una procedura	No
$t0-$t7	8-15	registri temporanei (non salvati)	No
$s0-$s7	16-23	registri salvati	Si
$t8-$t9	24-25	registri temporanei (non salvati)	No
$k0-$k1	26-27	gestione delle eccezioni	*Riservato OS*
$gp	28	puntatore alla global area (dati)	Si
$sp	29	stack pointer	Si
$s8	30	registro salvato (fp)	Si
$ra	31	indirizzo di ritorno	No

Figure 2: Tabella dei registri

Nel listato seguente i registri vengono chiamati col loro nome, per cui il registro $t8 corrisponde al registro $24, $t9 al $25 etc..

Dato il seguente programma:

```
0 loop:
1     lw $t8,0($t1)
2     add $t9,$t8,$t9
```

```
3       addi $t1,$t1,4
4       sw $t9,-4($t1)
5       bne $t1,4096,loop
```

Domande:

1. Individuare le dipendenze RAW, e disegnare il diagramma di esecuzione per un processore MIPS pipeline a 5 stadi (come quello visto a lezione, con delayed branch) nei 3 casi seguenti:
   (a) Senza forwarding, con un register file non ottimizzato.
   (b) Senza forwarding, ma con un register file ottimizzato, (scrive un nuovo registro nella prima parte di un ciclo, e legge una coppia di registri nella seconda parte del ciclo).
   (c) Con forwarding e con register file ottimizzato.
2. Individuare un'ottimizzazione del codice per il caso (c) che riduce gli stalli.

Soluzione domanda 1

Dipendenze:

- $1 \rightarrow 2$ l'istruzione 2 legge il registro $t8 precedentemente scritto dall'istruzione 1
- $2 \rightarrow 4$ l'istruzione 4 legge il registro $t9 precedentemente scritto dall'istruzione 2
- $3 \rightarrow 4$ l'istruzione 4 legge il registro $t1 precedentemente scritto dall'istruzione 3
- $3 \rightarrow 5$ l'istruzione 5 legge il registro $t1 precedentemente scritto dall'istruzione 3

Soluzione domanda 2

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
01	lw	IF	ID	EXE	MEM	WB										
02	add		IF	ID				EXE	MEM	WB						
03	addi			IF				ID	EXE	MEM	WB					
04	sw							IF	ID				EXE	MEM	WB	
05	bne								IF				ID	EXE	MEM	WB
06	nop												IF	ID	...	...

Soluzione domanda 3

		1	2	3	4	5	6	7	8	9	10	11	12	13
01	lw	IF	ID	EXE	MEM	WB								
02	add		IF	ID			EXE	MEM	WB					
03	addi			IF			ID	EXE	MEM	WB				
04	sw						IF	ID			EXE	MEM	WB	
05	bne							IF			ID	EXE	MEM	WB
06	nop										IF	ID	...	...

Soluzione domanda 4

		1	2	3	4	5	6	7	8	9	10	11
01	lw	IF	ID	EXE	MEM	WB						
02	add		IF	ID		EXE	MEM	WB				
03	addi			IF		ID	EXE	MEM	WB			
04	sw					IF	ID	EXE	MEM	WB		
05	bne						IF	ID	EXE	MEM	WB	
06	nop							IF	ID	EXE	MEM	WB

Soluzione domanda 5

Il seguente codice elimina tutti gli stalli e sposta una istruzione indipendente nel delay slot.

```
loop:
    lw $t8,0($t1)
    addi $t1,$t1,4
    add $t9,$t8,$t9
    bne $t1,4096,loop
    sw $t9,-4($t1)
```

1.1 Esercizio 4

Considerare il processore pipeline MIPS a 5 stadi visto a lezione, con delayed branch, forwarding, e register file speciale, e il seguente programma MIPS, che incrementa gli elementi di un array di interi, il cui indirizzo iniziale è contenuto nel registro $20, mentre $10 contiene l'indice dell'array.

```
0 loop:
1     add $11, $20, $10
2     lw $17, 0($11)
3     addi $17, $17, 50
4     sw $17, 0($11)
5     addi $10, $10, 4
6     bne $10, $21, loop
```

Domande:

1. Determinare le dipendenze RAW, e il diagramma temporale di esecuzione delle istruzioni che appaiono nel corpo del loop, mettendo in evidenza i forwarding.
2. Forzare con delle nop gli stalli che verrebbero comunque inseriti dall'hazard detection unit, ed ottimizzare il codice.
3. Rispetto al codice non ottimizzato, determinare il diagramma temporale nel caso in cui il processore non avesse il forwarding.

Soluzione domanda 1

Dipendenze:

- $1 \rightarrow 2$, l'istruzione 2 legge il registro $11 precedentemente scritto dall'istruzione 1
- $1 \rightarrow 4$, l'istruzione 4 legge il registro $11 precedentemente scritto dall'istruzione 1
- $2 \rightarrow 3$, l'istruzione 3 legge il registro $17 precedentemente scritto dall'istruzione 2
- $3 \rightarrow 4$, l'istruzione 4 legge il registro $17 precedentemente scritto dall'istruzione 3
- $5 \rightarrow 6$, l'istruzione 6 legge il registro $10 precedentemente scritto dall'istruzione 5

	1	2	3	4	5	6	7	8	9	10	11	12	13
add	If	id	exe	mem	wb								
lw		if	id	exe	mem	wb							
addi			if	id	id	exe	mem	wb					
sw				if	if	id	exe	mem	wb				
addi						if	id	exe	mem	wb			
bne							if	id	id	exe	mem	wb	
nop								if	if	id	exe	mem	wb

Si noti che la dipendenza $1 \rightarrow 4$ non necessita di forwarding, in quanto lo stadio `WB` dell'istruzione 1 si verifica al 5° ciclo, mentre lo stadio `ID` dell'istruzione 4 avviene al 6° ciclo. In questo modo quando l'istruzione 4 leggerà il valore dei registri durante lo stadio ID, essi avranno i valori aggiornati.

Soluzione domanda 2

Le uniche dipendenze che non sono risolte dal forwarding (o dal register file speciale) sono quelle tra la `lw` e l'istruzione successiva ($2 \rightarrow 3$) e quella tra `addi` e `bne`($5 \rightarrow 6$). Un'altra `nop` bisogna inserirla esplicitamente a causa del delay branch. Abbiamo quindi:

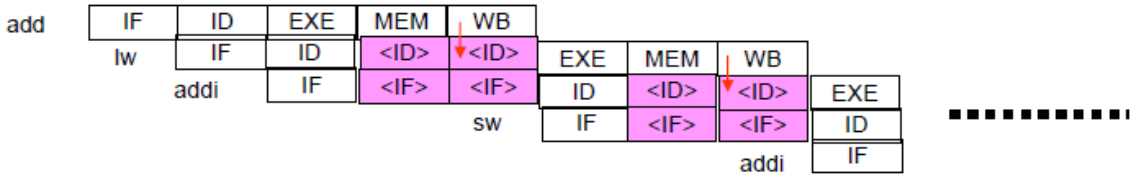
```
loop:
    add $11, $20, $10
    lw $17, 0($11)
    nop
    addi $17, $17, 50
    sw $17, 0($11)
    addi $10, $10, 4
    nop
    bne $10, $21, loop
    nop
```

Possiamo ottimizzare il codice, spostando indietro l'istruzione 5 (`addi`) dopo la `lw`, in modo da eliminare sia il primo stallo ($2 \rightarrow 3$) che il secondo ($5 \rightarrow 6$), e l'istruzione 4 (`sw`) in avanti nel branch delay slot:

```
loop:
    add $11, $20, $10
    lw $17, 0($11)
    addi $10, $10, 4
    addi $17, $17, 50
    bne $10, $21, loop
    sw $17, 0($11)
```

Soluzione domanda 3

Il diagramma relativo al codice non ottimizzato, nel caso in cui il forwarding non fosse attivo, è illustrato (solo parzialmente) di seguito:



Si noti come, grazie al register file speciale (frecce rosse), che scrive un nuovo registro nella prima parte del ciclo, e legge una coppia di registri nella seconda parte dello stesso ciclo, si risparmia un ciclo di stallo.

Risorse

- Struttura e progetto dei calcolatori - David A. Paterson, John L. Hennessy, Capitolo 4.

Risorse WTF

TIL che qualcuno ha costruito una ALU-16 bit su Minecraft.