Esercizio 2 Pipeline MIPS

Soluzione

Considerando la pipeline MIPS vista a lezione, si consideri il seguente frammento di codice:

LW	$1, 10($0)	**R1 ← mem[10+[R0]]**
ADDI	$8,$1, 1	**R8 ← [R1] + 1**
LW	$3, 0($1)	**R3 ← mem[0+[R1]]**
SUB	$3, $8, $1	**R3 ← [R8] – [R1]**
SW	$8, 0($1)	**mem[0+[R1]] ← [R8]**
ADDI	$1, $1, 16	**R1 ← [R1] + 16**
SW	$4, 20($1)	**mem[20+[R1]] ← [R4]**

a) si individuino e discutano le dipendenze dovute ai dati

b) mostrare come evolve la pipeline durante l'esecuzione del codice, assumendo:

- impossibilità di data forwarding;
- possibilità di data forwarding, così come visto a lezione per la pipeline MIPS;

Soluzione a):

DIPENDENZE	**[dipendenza dati (senza considerare limiti architettura MIPS)]** **[dipendenza dati considerando i limiti della architettura MIPS]**
R1 in ADDI $8,**$1**, 1 dipende da LW **$1**, 10($0)	**[input EX_{ADDI} ha bisogno di output da MEM_{LW}]** **[ID_{ADDI} deve legge R1 aggiornato da WB_{LW} (stesso ciclo clock o ciclo precedente)]**

R1 in LW $3,0(**$1**) dipende da LW **$1**, 10($0)	**[input EX_{LW} ha bisogno di output da MEM_{LW}]** **[ID_{LW} deve legge R1 aggiornato da WB_{LW} (stesso ciclo clock o ciclo precedente)]**
R8 in SUB $3, **$8** , $1 dipende da ADDI **$8**,$1, 1	**[input EX_{SUB} ha bisogno di output da EX_{ADDI}]** **[ID_{SUB} deve legge R8 aggiornato da WB_{ADDI} (stesso ciclo clock o ciclo precedente)]**
R1 in SUB $3, $8, **$1** dipende da LW **$1**, 10($0)	**[input EX_{SUB} ha bisogno di output da MEM_{LW}]** **[ID_{SUB} deve legge R1 aggiornato da WB_{LW} (stesso ciclo clock o ciclo precedente)]**
R1 in SW $8, 0(**$1**) dipende da LW **$1**, 10($0)	**[input EX_{SW} ha bisogno di output da MEM_{LW}]** **[ID_{SW} deve legge R1 aggiornato da WB_{LW} (stesso ciclo clock o ciclo precedente)]**
R8 in SW **$8** , 0($1) dipende da ADDI **$8**,$1, 1	**[input MEM_{SW} ha bisogno di output da EX_{ADDI}]** **[ID_{SW} deve legge R8 aggiornato da WB_{ADDI} (stesso ciclo clock o ciclo precedente)]**
R1 in ADDI $1,**$1**, 16 dipende da LW **$1**, 10($0)	**[input EX_{ADDI} ha bisogno di output da MEM_{LW}]** **[ID_{ADDI} deve legge R1 aggiornato da WB_{LW} (stesso ciclo clock o ciclo precedente)]**
R1 in SW $4, 20(**$1**) dipende da ADDI **$1**,$1, 16	**[input EX_{SW} ha bisogno di output da EX_{ADDI}]** **[ID_{SW} deve legge R1 aggiornato da WB_{ADDI} (stesso ciclo clock o ciclo precedente)]**
R1 in SW $4, 0(**$1**) dipende anche da LW **$1**, 10($0), ma R1 è già modificato da ADDI **$1**,$1,16 (che crea una dipendenza WAW)	

Soluzione b):

Evoluzione pipeline senza data forwarding

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
LW	$1, 10($0)	IF	ID	EXE	MEM	WB											
ADDI	$8,$1, 1		IF	ID	ID	ID	EX	MEM	WB								
LW	$3, 0($1)			IF	IF	IF	ID	EX	MEM	WB							

SUB	$3, $8, $1						IF	ID	ID	EX	MEM	WB					
SW	$8, 0($1)							IF	IF	ID	EX	MEM	WB				
ADDI	$1, $1, 16									IF	ID	EX	MEM	WB			
SW	$4, 20($1)										IF	ID	ID	ID	EXE	MEM	WB

Evoluzione pipeline con data forwarding

		1	2	3	4	5	6	7	8	9	10	11	12
LW	$1, 10($0)	IF	ID	EX	MEM>fw LMD	WB							
ADDI	$8,$1, 1		IF	ID	ID	da MEM_{LW}>EX	MEM>fw ALUOutput	WB					
LW	$3, 0($1)			IF	IF	ID	EX	MEM	WB				
SUB	$3, $8, $1					IF	ID	da MEM_{ADDI}>EX	MEM	WB			
SW	$8, 0($1)						IF	ID	EX	MEM	WB		
ADDI	$1, $1, 16							IF	ID	EX>fw ALUOutput	MEM	WB	
SW	$4, 20($1)								IF	ID	da EX_{ADDI}>EX	MEM	WB

Notare che

- al ciclo 4 si genera in uscita dalla memoria (la LW legge dalla memoria) il dato necessario per lo stadio esecutivo della ADDI; il dato utilizza il circuito di bypass che parte dal banco di registri MEM/WB ed arriva a uno degli ingressi della ALU.

- alla fine del ciclo 5 si genera in ALUOutput del banco di registri EX/MEM il dato necessario allo stadio esecutivo della SUB; lo stadio esecutivo della SUB, tuttavia, non può iniziare prima del ciclo 7; pertanto il dato generato dalla ADDI alla fine del suo stadio EX viene copiato, come da funzionamento ordinario, nel banco di registri MEM/WB durante il ciclo 6 e quindi durante il ciclo 7 il dato viene trasferito ai registri per la scrittura (stadio WB della ADDI) e simultaneamente si attiva il circuito di bypass, che tramite opportuno segnale di controllo ai multiplexer in ingresso alla ALU permette al dato di arrivare ad uno degli ingressi della stessa in tempo per l'inizio della fase EX della SUB.

- alla fine del ciclo 9 in ALUOutput del banco di registri EX/MEM si trova il dato che deve essere scritto nel registro $1, che successivamente deve essere letto dalla SW. Pertanto è possible utilizzare il circuito di bypass dall'output della ALU all'input alto della ALU.