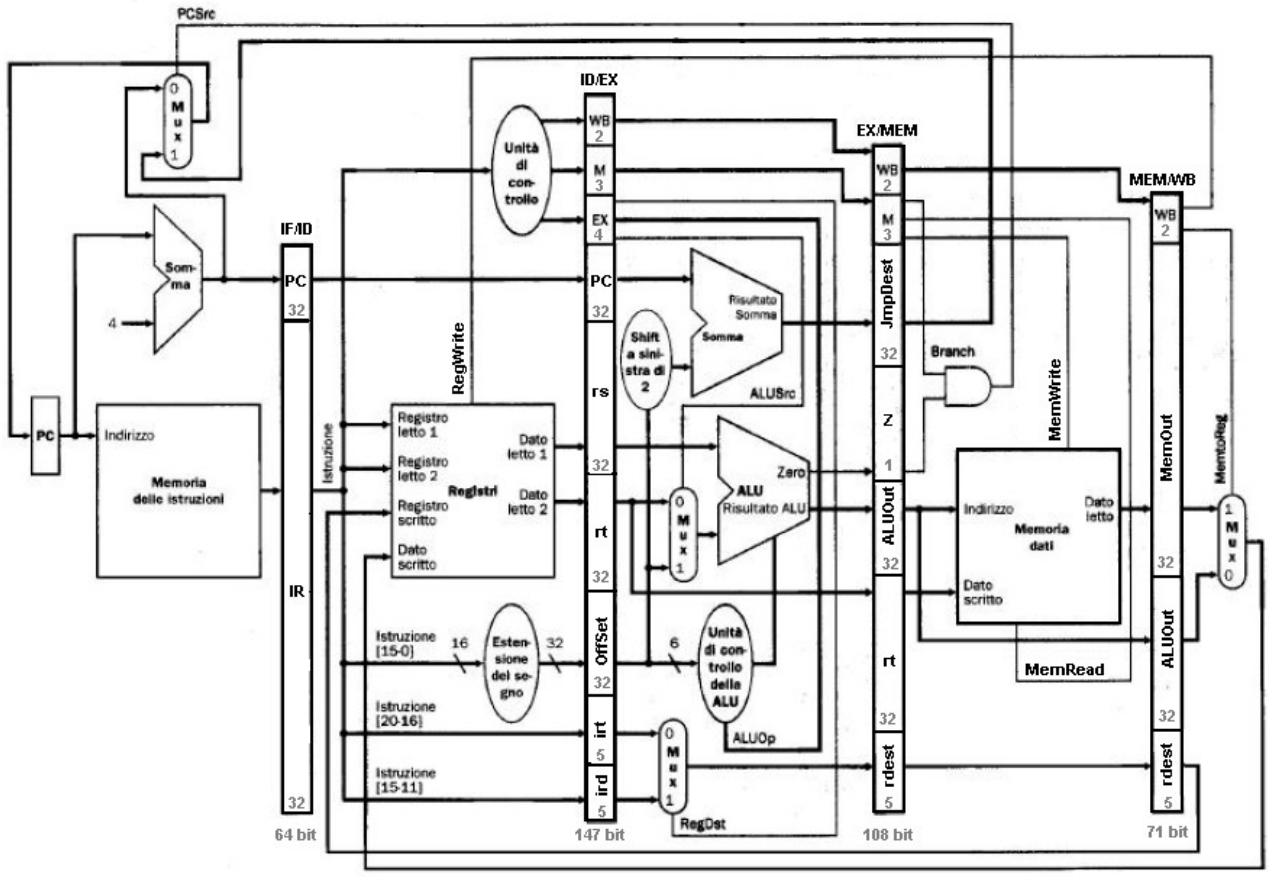
Esercitazione del 18/05/2006 – Soluzioni

1. Contenuto dei registri di pipeline in una CPU pipelined non ottimizzata

a. Esecuzione di un'istruzione di LOAD.

Consideriamo la seguente CPU non ottimizzata (Ignoriamo per il momento la gestione degli hazard):



e verifichiamo lo stato dei registri di pipeline durante l'esecuzione dell'istruzione

lw $t0 , 4 ($s0)

posizionata all'indirizzo di memoria istruzioni 0x40000000. Ignoriamo per semplicità le istruzioni precedenti e assumiamo che non abbiano effetto su quelle attuali.

L'istruzione ha codice operativo:

31 26	25 21	20 16	15 0
LW 100011	**base** 10000	**rt** 01000	**offset** 0000 0000 0000 0100

All'inizio del primo ciclo di clock il Program Counter (PC) contiene **0x0400.0000.** Durante il primo

ciclo di clock, il primo stadio della pipeline esegue le seguenti operazioni:

1. viene letto la locazione di memoria puntata da PC.
2. viene calcolato l'indirizzo successivo, **PC+4**, tramite un sommatore dedicato.

Alla fine del primo ciclo di clock sull'ingresso del registro **IF/ID** saranno presenti i seguenti segnali:

- **PCX** = **0x0400.0004,** l'indirizzo dell'istruzione immediatamente seguente,
- **IR = 1000 11.10 000.0 1000. 0000 0000 0000 0100 = 0x8e08.0004,** l'istruzione attuale.

In totale il registro **IF/ID** comprende **64 bit.**

Supponiamo che il multiplexer presente nel primo stadio sia settato a 0, **PCSrc=0** (non si è verificato nessun branch, negli istanti precedenti). All'ingresso del PC è presente il valore **0x0400.0004** che verrà memorizzato alla fine del ciclo. Nei successivi cicli il PC verrà incrementato in questo modo a meno di salti.

Al secondo ciclo di clock l'istruzione viene decodificata e vengono preparati alcuni dati temporanei. Nel dettaglio:

1. Viene decodificata l'istruzione dall'unità di controllo e generati tutti i controlli per i successivi stadi.
2. Vengono letti dal register file i registri corrispondenti a **rs** e **rt** in una ipotetica istruzione di tipo **R** (bit 25-21 per rs e bit 20-16 per rt).
3. L'offset viene esteso a 32 bit considerando il segno (bit 15-0).

All'interno del registro di pipeline **ID/EX** vengono memorizzati i seguenti valori:

- **WB** contiene i controlli per il register file durante la fase di WriteBack:
  - **WB.MemtoReg** = **1**, in modo da creare un path tra memoria e register file,
  - **WB.RegWrite** = **1,** per abilitare la scrittura nel register file
- **MEM** contiene i segnali per la memoria dati e il controllo per la valutazione del branch:
  - **MEM.MemRead** = **1**
  - **MEM.MemWrite** = **0**, poiché vogliamo compiere una lettura da memoria
  - **MEM.Branch** = **0**, poiché non vogliamo valutare un eventuale salto.
- **EX** contiene i segnali per la ALU:
  - **EX.ALUSrc** = **1**, usa per il secondo termine della somma l'**offset** esteso.
  - **EX.RegDest** = **0**, usa **rt** per memorizzare il dato letto da memoria.
  - **EX.ALUOp** = **00**, esegui la somma.
- **PCX** = **0x0400.0004,** l'indirizzo dell'istruzione immediatamente successiva,
- **rs,** il valore del registro **rs**, la base, ex. **0x1000.0000**
- **rt**, il valore del registro **rt**, non usato, ex. **0x1234.5678**
- **Offset = 0x0000.0004**
- **irt = 01000,** il valore dell'indice del registro **rt** in una ipotetica istruzione che lo prevede (i bit 20-16 dell'istruzione),
- **ird = 00000,** il valore dell'indice del registro **rd** (i bit 15-11 dell'istruzione).

In totale **ID/EX** risulta di 147 bit.

Nello stesso ciclo di clock l'istruzione immediatamente seguente viene estratta dalla memoria istruzioni e memorizzata nel registro **IF/ID.**

Al terzo ciclo viene eseguita l'istruzione. In questo caso viene calcolato l'indirizzo da usare per estrarre il dato corretto dalla memoria. In particolare:

- viene eseguita l'operazione impostata sulla ALU dal segnale presente sulla linea ALUOp e successivamente rifinita dalla sotto unità di controllo della ALU. In questo caso il codice **00** viene interpretato come una somma senza esaminare i bit **funct/shift**. Poiché ALUSrc è uguale a **1** la somma viene eseguita tra **rs** e l'**offset** esteso a 32 bit.
- viene calcolato l'indirizzo di un eventuale salto condizionato, non usato in quanto non è in esecuzione una istruzione di branch, sommando al valore **PCX,** l'**offset** esteso e shiftato a sinistra di 2 (moltiplicato per 4).
- viene selezionato in quale registro memorizzare eventualmente il dato letto/calcolato, in questo caso **rt** dato che **RegDest** = **0**.

Alla fine del terzo ciclo di clock, agli ingressi del registro di separazione **EX/MEM** sono persenti i seguenti valori:

- **WB** come allo stadio precedente,
- **MEM** come allo stadio precedente**,**
- **JumpDest** = **0x0400.0004 + 0x0000.0004 * 4 = 0x0400.0014,** l'indirizzo dell'istruzione di destinazione di un eventuale salto condizionato, non usato.
- **Z = 0** , poiché il risultato della ALU non è zero**,** non usato.
- **ALUOut = 0x1000.0004,**l'indirizzo da leggere dalla memoria,
- **rt**, il valore del registro **rt**, non usato, ex. **0x1234.5678**
- **rdest = 01000,** il valore dell'indice del registro di destinazione.

In totale **EX/MEM** risulta di 108 bit.

Nello stesso ciclo di clock l'istruzione immediatamente seguente legge i registri eventualmente usati dal register file e li memorizza in **ID/EX.** Se l'istruzione usa il registro **$t0** in lettura, si verifica un incoerenza sui dati, poiché il suo valore non è quello che sara estratto dalla memoria dall'istruzione di LOAD. L'istruzione che segue a distanza di due cicli di clock viene fetch-ta e memorizzata in **IF/ID.**

Al quarto ciclo di clock viene eseguita la lettura/strittura della memoria dati ed eventualmente valutato un ipotetico salto condizionato.

1. Viene eseguita la lettura della memoria dati,
2. viene valutato se occorre reimpostare il PC, non in questo caso poiché **Branch** = **0.**

Al termine del ciclo di clock nel registro di pipeline MEM/WB (parte master) saranno presenti i seguenti segnali:

- **WB** come allo stadio precedente,
- **MEMOut** , il contenuto della locazione ev. letta dalla memoria, ex. **0x8765.4321**,
- **ALUOut** come l' stadio precedente, per un eventuale scrittura nel registrer file, non usato,
- **rdest = 01000,** il valore dell'indice del registro di destinazione.

Il registro **MEM/WB** richiede 71 bit.

Nello stesso ciclo di clock l'istruzione immediatamente seguente viene eseguita nel terzo stadio. Se esiste una dipendenza con **$t0** non è possibile recuperare il dato in quanto è ancora all'interno della memoria dati. L'unico modo per risolvere il problema consiste nel mettere in stallo la CPU in attesa che il dato sia disponibile. L'istruzione che segue a distanza di due cicli di clock legge il registri dal register file e li memorizza in **ID/EX**. Anche in questo caso si può verificare un problema di inconsistenza su **$t0.** L'istruzione che segue a distanza di tre cicli di clock viene fetch-ta dalla memoria e memorizzata in **IF/ID**.

Al quinto ciclo di clock viene eseguita l'operazione di *WriteBack* nel register file, se necessaria. In questo caso il valore estratto dalla memoria viene selezionato impostando il multiplexer in maniera opportuna, **MemtoReg = 1.**

- Viene scritto il dato, **0x8765.4321** in questo caso, nel register file in posizione definita da **rdest, $t0**.

Nello stesso ciclo di clock l'istruzione che segue immediatamente può eseguire una scrittura in memoria utilizzando dati eventualmente incoerenti. L'istruzione a distanza di due cicli di clock viene eseguita. Al contrario del caso precedente, il valore corretto per **$t0** è presente nel registro **EX/WB.MemOut (parte slave).** Con un opportuno hardware questa criticità può essere risolta senza mettere in stallo la CPU. L'istruzione che segue a distanza di tre cicli di clock legge i registri dal register file. A meno di non utilizzare accorgimenti particolari anche in questo caso può insorgere incoerenza su **$t0**.

b. Risolvere le criticità sui dati in una operazione di LOAD.

Alla fine del quinto ciclo di clock l'istruzione è terminata. Così strutturato il dato **$t0** risulta inconsistente per le tre istruzioni successive alla istruzione di LOAD.

Almeno per gli ultimi due casi di inconsistenza esaminati è possibile risovere la criticità, adottando:

- Un hardware opportuno per propagare in anticipo un dato presente nella pipe ma non ancora memorizzato nel register file (Unita di propagazione).
- Utilizzando un register file che garantisca la lettura del dato corretto alla fine del ciclo di clock nel caso un registro venga letto e scritto nello stesso ciclo (scrittura nel primo semi-ciclo e lettura nel secondo semi-ciclo oppure by-pass tra dato in scrittura e datoin lettura).

Rimane la criticità rispetto alla istruzione immediatamente seguente che può essere risolta mandando in stallo la CPU al momento della sua decodifica, primo momento possibile per rilevare questa condizione.

c. Esecuzione di un'istruzione di BRANCH.

Vediamo ora la stessa sequenza per un istruzione di Branch**:**

beq $s0 , $s1 , 40

L'OpCode dell'istruzione, che supponiamo memorizzata all'indirizzo **0x0400.0000**, è (l'offset codificato nella trama è calcolato dividendo per 4 l'offset della istruzione assembly, poiché i salti saranno sempre allineati alle word):

31 26	25 21	20 16	15 0
000100	10000	10001	0000 0000 0000 1010

All'inizio del primo ciclo di clock il Program Counter (PC) contiene **0x0400.0000.** Durante il primo ciclo di clock, il primo stadio della pipeline esegue come nel caso precedente le seguenti operazioni:

1. viene letto la locazione di memoria puntata da PC.
2. viene calcolato l'indirizzo successivo, **PC+4**, tramite un sommatore dedicato.

Alla fine del primo ciclo di clock sull'ingresso del registro **IF/ID** saranno presenti i seguenti segnali:

- **PCX** = **0x0400.0004,** l'indirizzo dell'istruzione immediatamente seguente,
- **IR = 000100.10 000.10001 .00000000 00001010 = 0x1211.000a,** l'istruzione attuale.

Supponiamo che il multiplexer presente nel primo stadio sia settato a 0, **PCSrc=0** (non si è verificato nessun branch, negli istanti precedenti). All'ingresso del PC è presente il valore **0x0400.0004** che verrà memorizzato alla fine del ciclo. Nei successivi cicli il PC verrà incrementato in questo modo a meno che non si verifichi il salto.

Al secondo ciclo di clock l'istruzione viene decodificata e vengono preparati alcuni dati temporanei.

All'interno del registro di pipeline **ID/EX** vengono memorizzati i seguenti valori:

- **WB** contiene i controlli per il register file durante la fase di WriteBack:
  - **WB.MemtoReg** = **X**, non è necessario eseguire nessuna scrittura nel register file da cui questo valore è irrilevante,
  - **WB.RegWrite** = 0**,** per disabilitare qualsiasi scrittura nel register file
- **MEM** contiene i segnali per la memoria dati e il controllo per la valutazione del branch:
  - **MEM.MemRead** = **0**
  - **MEM.MemWrite** = **0**, poiché non vogliamo compiere nessuna operazione sulla memoria,
  - **MEM.Branch** = **1**, poiché vogliamo valutare il salto condizionato.
- **EX** contiene i segnali per la ALU:
  - **EX.ALUSrc** = **0**, usa per il secondo termine della somma il registro **rs**.
  - **EX.RegDest** = **X**, irrilevante poiché non dobbiamo compiere operazioni di WB.
  - **EX.ALUOp** = **01**, esegui la sottrazione, necessaria per valutare l'ugualianza.
- **PCX** = **0x0400.0004,** l'indirizzo dell'istruzione immediatamente successiva,
- **rs,** il valore del registro **rs**, la base, ex. **0x1000.0000**
- **rt**, il valore del registro **rt**, non usato, ex. **0x1000.0000**,

- **Offset = 0x0000.000a**
- **irt = 10001,** non usato,
- **ird = 00000**, non usato.

Nello stesso ciclo di clock l'istruzione immediatamente seguente viene estratta dalla memoria istruzioni e memorizzata nel registro **IF/ID.** Questa istruzione, a meno di accorgimenti opportuni, verrà comunque eseguita sia nel caso non si verifichi il salto, sequenza di istruzioni corretta, sia nel caso il salto si verifichi, nel qual caso questa istruzione non dovrebbe essere eseguita. In quest'ultimo caso occorre annullare l'istruzione nella pipe e farla diventare una **bubble.**

Al terzo ciclo viene eseguita l'istruzione. In questo caso viene calcolata la differenza tra **rs** ed **rt**, in modo da verificare l'ugualianza e quindi decidere se eseguire il salto.

Alla fine del terzo ciclo di clock, agli ingressi del registro di separazione **EX/MEM** sono persenti i seguenti valori:

- **WB** come allo stadio precedente,
- **MEM** come allo stadio precedente**,**
- **JumpDest** = **0x0400.0004 + 0x0000.000a * 4 = 0x0400.002c,** l'indirizzo dell'istruzione di destinazione del salto condizionato,
- **Z = 1** , poiché il risultato della ALU**,** la sottrazione tra **rs** ed **rt** ,uguali, da come risultato zero,
- **ALUOut = 0x0000.0000,** il risultato della sottrazione, non usato,
- **rt=0x1000 0000**, il valore del registro **rt**, non usato in seguito.
- **rdest = X, non usato**.

Nello stesso ciclo di clock l'istruzione immediatamente seguente legge i registri eventualmente usati dal register file e li memorizza in **ID/EX.** L'istruzione che segue a distanza di due cicli di clock viene fetch-ta e memorizzata in **IF/ID.** In caso di salto anche questa istruzione deve essere annullata.

Al quarto ciclo di clock viene valutato il salto condizionato. Il segnale **ID/EX.Branch=1** viene messo in AND con il segnale **ID/EX.Z=1** ed il risultato viene usato per pilotare il multiplexer presente all'ingresso del registro **PC**. Al termine del ciclo di clock il **PC** verrà riscritto con il valore presente in **ID/EX.JumpDest=0x0400.002c.**

Al termine del ciclo di clock nel registro di pipeline MEM/WB (parte master) saranno presenti i seguenti segnali:

- **WB** come allo stadio precedente,
- **MEMOut**, non usato,
- **ALUOut** come l' stadio precedente, non usato,
- **rdest** come l' stadio precedente, non usato.

Nello stesso ciclo di clock l'istruzione immediatamente seguente viene eseguita nel terzo stadio memorizzando i risultati nel registro **EX/MEM**. L'istruzione che segue a distanza di due cicli di clock legge il registri dal register file e li memorizza in **ID/EX**. L'istruzione che segue a distanza di tre cicli di clock viene fetch-ta dalla memoria e memorizzata in **IF/ID**. Anche questa istruzione andrà annullata nel caso si verifichi il salto, come nel caso dell'esempio.

Al quinto ciclo di clock l'istruzione **beq** non esegue nessuna operazione di WriteBack (all'atto pratico si è trasformata in una nop che sta uscendo dalla pipe).

Poiché si è verificato il salto, nella pipe è stata caricata l'istruzione in posizione **0x04000.002c.** Nello stesso ciclo di clock l'istruzione che segue immediatamente puo' eventualmente eseguire una scrittura in memoria dati. L'istruzione a distanza di due cicli di clock viene eseguita. L'istruzione che segue a distanza di tre cicli di clock legge i registri dal register file.

d. Risolvere le criticità sulle istruzioni in una operazione di BRANCH.

Alla fine del quinto ciclo di clock l'istruzione è terminata. Così strutturata una istruzione di BRANCH influenza le tre istruzioni seguenti che vengono comunque caricate nella pipe prima che il salto venga valutato. E' possibile anticipare la valutazione del salto modificando opportunamente l'hardware:

- Anticipando il calcolo dell'indirizzo di salto allo stadio di decode spostando il sommatore dedicato.
- Inserendo in coda al register file un hardware dedicato per la comparazione dei due valori estratti dai registri. La lettura del register file risulta molto più veloce della lettura della memoria dati ed istruzioni. Questo lascia un po' di tempo alla fine del ciclo di clock per la comparazione di uguaglianza tra i registri, che può essere ottenuta con un circuito molto più semplice e veloce del circuito implementato dalla ALU per eseguire la sottrazione.

Ottimizzando in questo modo l'hardware si può limitare la criticità sulle istruzioni alla sola istruzione seguente che andrà comunque annullata in caso di salto.

e. Esecuzione di un'istruzione di tipo R.

Vediamo ora la stessa sequenza per un istruzione di tipo R**:**

and $t3, $s0 , $t1

posizionata all'indirizzo di memoria istruzioni **0x40000000**.

L'istruzione ha codice operativo:

31	**26**	**25**	**21**	**20**	**16**	**15**	**11**	**10**	**6**	**5**	**0**
Special 000000		**rs** 10000		**rt** 01001		**rd** 01011		**shift** 0000		**function** 101000	

All'inizio del primo ciclo di clock il Program Counter (PC) contiene **0x0400.0000.**

Alla fine del primo ciclo di clock sull'ingresso del registro **IF/ID** saranno presenti i seguenti segnali:

- **PCX** = **0x0400.0004**,
- **IR = 000000.10 000.01001. 01001.000 00.101000 = 0x0209.4828.**

Supponiamo che il multiplexer presente nel primo stadio sia settato a 0, **PCSrc=0** (non si è verificato nessun branch, negli istanti precedenti). All'ingresso del PC è presente il valore **0x0400.0004** che verrà memorizzato alla fine del ciclo. Nei successivi cicli il PC verrà incrementato in questo modo a meno di salti.

Al secondo ciclo di clock l'istruzione viene decodificata.

All'interno del registro di pipeline **ID/EX** vengono memorizzati i seguenti valori:

- **WB** contiene i controlli per il register file durante la fase di WriteBack:
  - **WB.MemtoReg** = **0**, in modo da creare un path tra ALU e register file,
  - **WB.RegWrite** = **1,** per abilitare la scrittura nel register file
- **MEM** contiene i segnali per la memoria dati e il controllo per la valutazione del branch:
  - **MEM.MemRead** = **0**
  - **MEM.MemWrite** = **0**, poiché non vogliamo compiere alcuna operazione sulla memoria,
  - **MEM.Branch** = **0**, poiché non vogliamo valutare un eventuale salto.
- **EX** contiene i segnali per la ALU:
  - **EX.ALUSrc** = **0**, usa per il secondo termine della somma il registro **rt**.
  - **EX.RegDest** = **1**, usa **rd** per memorizzare il dato letto da memoria.
  - **EX.ALUOp** = **10**, valuta il campo funct per sapere quale operazione eseguire.
- **PCX** = **0x0400.0004,** l'indirizzo dell'istruzione immediatamente successiva,
- **rs,** il valore del registro **rs**, ex. **0x1111.1111**
- **rt**, il valore del registro **rt**, ex. **0x0000.ffff**
- **Offset = 0x0000.4828 ,** non usato come offset ma scomposto per valutare il campo **shift** e **funct**,
- **irt = 01001,** il valore dell'indice del registro **rt**,
- **ird = 01011,** il valore dell'indice del registro **rd**.

Nello stesso ciclo di clock l'istruzione immediatamente seguente viene estratta dalla memoria istruzioni e memorizzata nel registro **IF/ID.**

Al terzo ciclo viene eseguita l'istruzione. In questo caso la sotto-unità di controllo della ALU valuta il campo **funct** estratto dal campo **ID/EX.Offset=0x00004828,** e imposta la ALU in modo da eseguire l'operazione di **and** tra registri.

Alla fine del terzo ciclo di clock, agli ingressi del registro di separazione **EX/MEM** sono persenti i seguenti valori:

- **WB** come allo stadio precedente,
- **MEM** come allo stadio precedente**,**
- **JumpDest = 0x0400.0004 + 0x0000.4828 * 4 = 0x0401.20A4,** non usato,
- **Z = 0** , poiché il risultato della ALU non è zero**,** non usato,
- **ALUOut = 0x0000.1111,** il risultato della **and,**
- **rt = 0x0000ffff**, il valore del registro **rt**, non usato,
- **rdest = 01011,** il valore dell'indice del registro di destinazione.

Nello stesso ciclo di clock l'istruzione immediatamente seguente legge i registri eventualmente usati dal register file e li memorizza in **ID/EX.** Se l'istruzione che segue usa il registro **$t3** in lettura, si verifica un incoerenza sui dati. L'istruzione che segue a distanza di due cicli di clock viene fetch-ta e memorizzata in **IF/ID.**

Al quarto ciclo di clock l'istruzione **and** nessuna operazione. All'atto pratico l'istruzione nel quarto stadio si comporta come una **nop:** i dati calcolati nello stadio precedente vengono passati allo stadio successivo.

Al termine del ciclo di clock nel registro di pipeline MEM/WB (parte master) saranno presenti i seguenti segnali:

- **WB** come allo stadio precedente,
- **MEMOut,** non usato,
- **ALUOut=0x00001111,** come l' stadio precedente,
- **rdest = 01011,** come l' stadio precedente.

Nello stesso ciclo di clock l'istruzione immediatamente seguente viene eseguita nel terzo stadio. Se esiste una dipendenza con **$t3,** nel caso di operazione tra registri, è possibile recuperare il dato dalla pipe dal registro **EX/MEM.ALUOut (parte slave)**. L'istruzione che segue a distanza di due cicli di clock legge il registri dal register file e li memorizza in **ID/EX**. Anche in questo caso si può verificare un problema di inconsistenza su **$t3.** L'istruzione che segue a distanza di tre cicli di clock viene fetch-ta dalla memoria e memorizzata in **IF/ID**.

Al quinto ciclo di clock viene eseguita l'operazione di *WriteBack* nel register file. In questo caso il valore estratto dalla memoria viene selezionato impostando il multiplexer in maniera opportuna, **MemtoReg** = 0**.**

- Viene scritto il dato, **0x0000.1111** in questo caso, nel register file in posizione definita da **rdest, $t3**.

Nello stesso ciclo di clock l'istruzione che segue immediatamente può eseguire una scrittura in memoria utilizzando dati eventualmente incoerenti. L'istruzione a distanza di due cicli di clock viene eseguita. Il valore corretto per **$**t3, se necessario, è presente nel registro **EX/WB.MemOut (parte slave).** L'istruzione che segue a distanza di tre cicli di clock legge i registri dal register file. Anche in questo caso può insorgere incoerenza su **$t3**.

b. Risolvere le criticità sui dati in una operazione di tipo R.

Alla fine del quinto ciclo di clock l'istruzione è terminata. Così strutturato il dato **$t3** risulta inconsistente per le tre istruzioni successive alla istruzione di tipo R.

Per risolvere queste criticità si può procedere come per il caso delle istruzioni LOAD.

- Un hardware opportuno per propagare in anticipo un dato presente nella pipe ma non ancora memorizzato nel register file (Unita di propagazione).
- Utilizzando un register file che garantisca la lettura del dato corretto alla fine del ciclo di clock nel caso un registro venga letto e scritto nello stesso ciclo (scrittura nel primo semi-ciclo e lettura nel secondo semi-ciclo oppure by-pass tra dato in scrittura e datoin lettura).

In questo caso però è possibile risolvere tutte e tre le criticità senza introdurre stalli nella pipe poichè in ogni istante il dato corretto può essere recuperato da un qualche punto opportuno della pipe stessa.