MIPS Pipelined Datapath with Data Hazard Solution and 2 Level Cache Memory

Alessandro Fornasier

Capitolo 1

MIPS Instruction set

Il presente documento è redatto come semplice guida per il test del processore con architettura MIPS Pipelined sviluppato in VHDL. Le istruzioni per questo tipo di architettura sono essenzialmente tre e sono mostrate in figura 1.1.

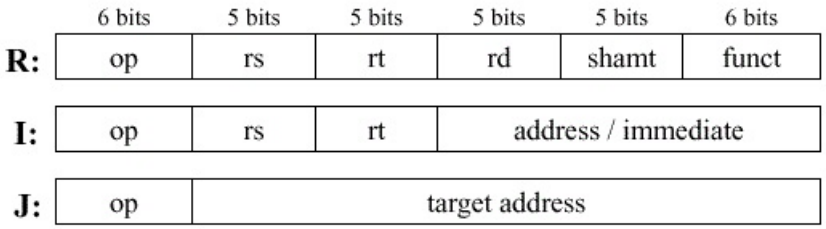

Figura 1.1: MIPS Instruction

Le tabelle 1.1 e 1.2 definiscono il significato dei campi *op* e *funct* delle varie istruzioni.

Per il test del processore sviluppato si è scritto un semplice programma *"instruction.txt"* che ne testasse il funzionamento, in particolare si sono testati i casi "critici". Il seguente elenco puntato e la tabella 1.3 descrivono il programma sviluppato.

- NOP.
- NOP.
- NOP.
- NOP.
- Salvo nel registro 0 il valore 2.

- Salvo nel registro 1, 2 + il valore contenuto nel registro 0.
- Salvo nel registro 2, 2 + il valore contenuto nel registro 1.
- Salvo nel registro 3, 2 + il valore contenuto nel registro 2.
- Salvo in memoria all'indirizzo 0 il valore 8 (contenuto nel registro 3).
- Salvo nel registro 4, 2 + il valore contenuto nel registro 3.
- Salvo nel registro 5, 2 + il valore contenuto nel registro 4.
- Salvo nel registro 6, 2 + il valore contenuto nel registro 5.
- Salvo nel registro 7, 2 + il valore contenuto nel registro 6.
- salvo nel registro 8 il valore 8 (contenuto in memoria all'indirizzo 0).
- Sommo i valori nei registri 7 e 8 (8+16 = 24) e salvo il risultato nel registro 9.
- salvo nel registro 10 il valore 4 (contenuto in memoria all'indirizzo 1).
- Salvo in memoria all'indirizzo 33 il valore 8 (contenuto nel registro 3).
- Salvo in memoria all'indirizzo 65 il valore 10 (contenuto nel registro 4).

op	**Significato**
000000	NOP
100000	R
000001	Arithmetic I (sum)
000010	Data transfer I load
000011	Data transfer I store
000100	Logical I (and)
000101	Logical I (or)
000110	Logical I (shift left)
000111	Logical I (shift right)
001000	Londitional branch J
010000	Unconditional jump J
others	None

Tabella 1.1: Significato op code

funct	**Significato**
000000	Add
000001	Subtract
000010	And
000011	Or
000100	Nor
others	None

Tabella 1.2: Significato funct code

Instruction	**Behavior**
00000000000000000000000000000000	NOP
00000000000000000000000000000000	NOP
00000000000000000000000000000000	NOP
00000000000000000000000000000000	NOP
00000110000000000000000000000010	$S1 = $S2 + 2
00000100000000001000000000000010	$S1 = $S2 + 2
00000100001000100000000000000010	$S1 = $S2 + 2
00000100010000110000000000000010	$S1 = $S2 + 2
00001111111000110000000000000000	Memory[$S2 + 0] = $S1
00000100011001000000000000000010	$S1 = $S2 + 2
00000100100001010000000000000010	$S1 = $S2 + 2
00000100101001100000000000000010	$S1 = $S2 + 2
00000100110001110000000000000010	$S1 = $S2 + 2
00001010101010000000000000000000	$S1 = Memory[$S2 + 0]
10000001000001110100100000000000	$S1 = $S2 + $S3
00001010101010100000000000000001	$S1 = Memory[$S2 + 0]
00001111111000110000000000100001	Memory[$S2 + 0] = $S1
00001111111001000000000001000001	Memory[$S2 + 0] = $S1

Tabella 1.3: Programma di test sviluppato.

Capitolo 2

Simulation

Il risultato relativo alla simulazione del processore, rappresentato in figura 2.1, ci permette, insieme al listato del programma ed alla sua spiegazione sopra riportata, di mostrare al lettore il corretto funzionamento del processore implementato. Dopo le prime quattro istruzioni di NOP, introdotte per fare in modo che il meccanismo del pipeline sia a regime alla prima istruzione "utile", si può notare come per ogni istruzione che richiede come operando il risultato dell'istruzione precedente (tipico caso di generazione di Data Hazard) venga correttamente eseguita, inoltre, nel caso in cui successivamente ad un istruzione di Load un istruzione richieda come operando il valore appena caricato, il sistema di risoluzione dei Data Hazard blocca il Program Counter ed i registri di pipeline ed introduce una Bubble (evidenziato in giallo in figura) al fine di rendere risolvibile questa particolare tipologia di Data Hazard. Per quanto riguarda La memoria Cache a due livelli questa è stata progettata ed implementata secondo l'architettura riportata in figura 2.2 si è inoltre implementato, ai fini di simulare una casistica reale, un ritardo di scrittura e lettura pari a 13 ns per la memoria Cache L1 e 201 ns per la memoria Cache L2 (cerchiati rispettivamente in rosso e in arancione in figura tra i due è riportato inoltre il caso di lettura con Cache Miss, in questo caso prima di avere il dato correttamente scritto nella Cache è necessario attendere 13 + 13 + 201 = 227 ns). Come si può vedere dalla simulazione, in figura 2.1, le operazioni di scrittura e di lettura sono correttamente implementate sia nel caso di Cache Hit sia nel caso di Cache Miss (risultati corretti cerchiati in verde)

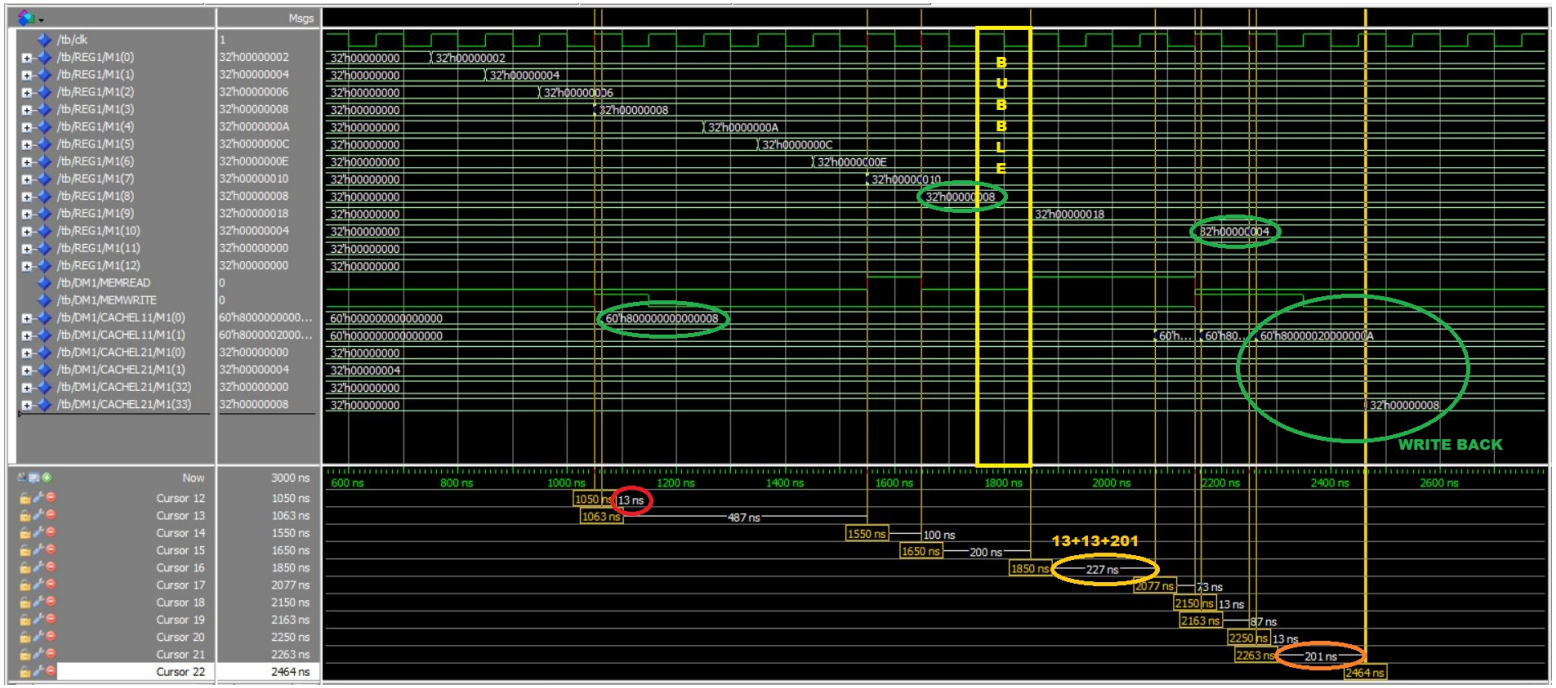

Figura 2.1: Risultati ottenuti dal test del processore

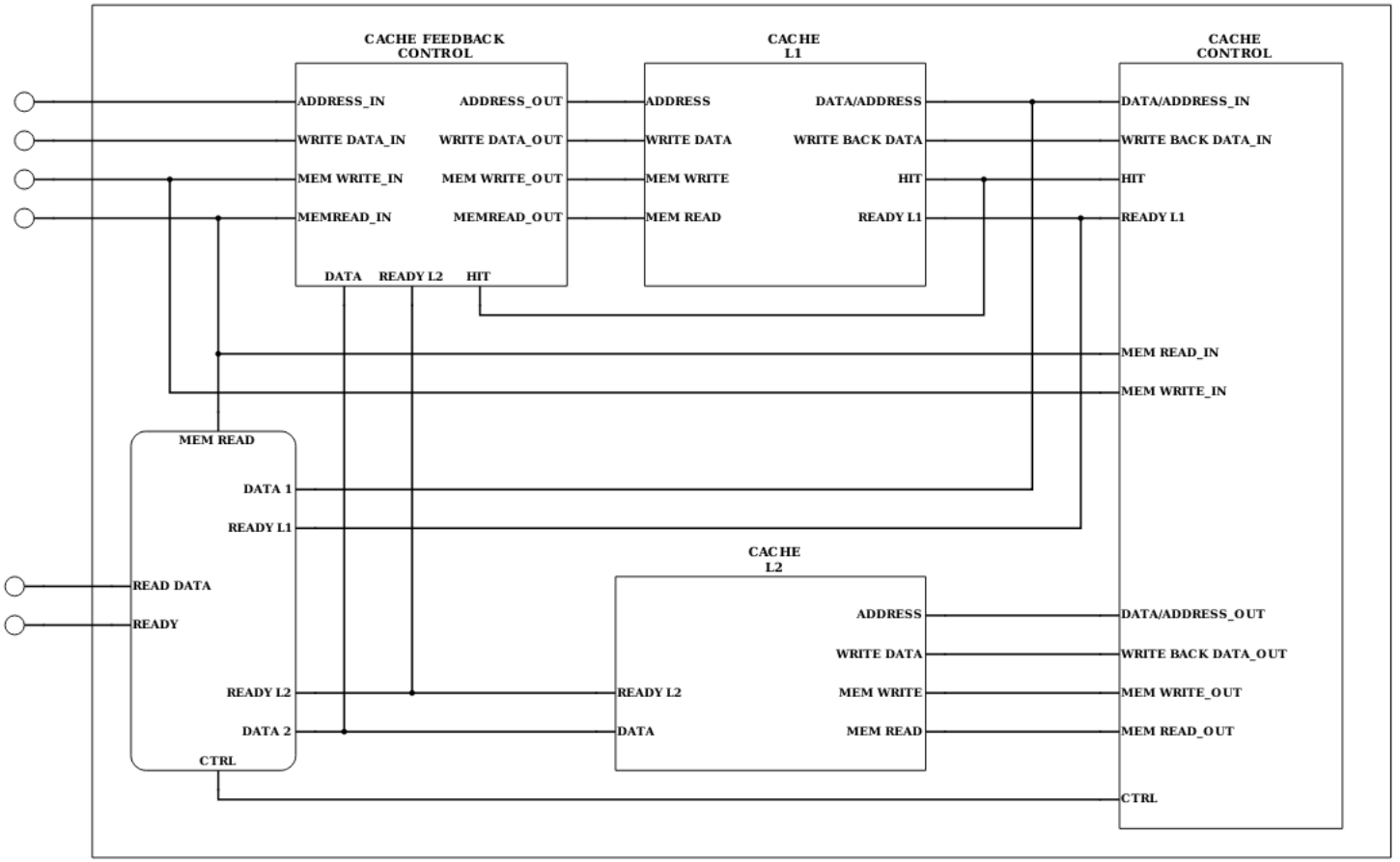

Figura 2.2: Architettura progettata ed implementata per la memoria Cache a due livelli