Indice:

-Introduzione: cosa è il btb e a cosa serve

-Architettura: come è stato concepito e la sua struttura interna + scenari

-Interfacciamento con il dlx: quali segnali usiamo e come cambia la logica della pipeline

-Risultati sperimentali: testbench del btb + simulazioni

-Codice

# Introduzione

Il progetto realizzato si basa sull’implementazione nel linguaggio VHDL di un Branch Target Buffer (BTB) da utilizzare con un processore DLX operante con una pipeline costituita da cinque stadi: Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory (MEM), Write Back (WB).

[todo: inserire immagine della pipeline]

Il componente realizzato dovrebbe fornire un incremento delle prestazioni della pipeline, in particolare dello throughput, in quanto permette di evitare stalli dovuti alle istruzioni di branch nel sistema DLX predicendo le stesse, grazie ad un algoritmo di predizione, nello stadio di IF. Il sistema verifica sempre la predizione nello stadio di EX in cui è valutata la condizione del branch. Nel caso di valutazione contrastante con la predizione è necessario inserire degli stalli all’interno della pipeline e riprendere l’esecuzione dall’istruzione corretta. Nel caso di valutazione coerente non è necessario apportare alcuna modifica al flusso di esecuzione poiché è stato già eseguito il fetch dell’istruzione corretta.

# Architettura

Il BTB è realizzato come una cache i cui TAG sono costituiti dai Program Counter (PC) corrispondenti a istruzioni che in precedenza sono state individuate, dallo stadio di EX, come branch. La prima volta che s’incontra un’istruzione di branch, non è presente una linea relativa all’interno del BTB; pertanto l’esecuzione procede normalmente fino allo stadio di EX, il quale valuta il branch e procede all’aggiornamento del BTB.

## Struttura interna

Nel caso specifico, l’unità è stata realizzata mediante una cache set-associative a due vie da 64 slot.

[todo: inserire immagine btb table]

Ogni via segue la seguente modellazione:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TAG | DESTINAZIONE | PREDIZIONE | RIMPIAZZAMENTO | STATO |

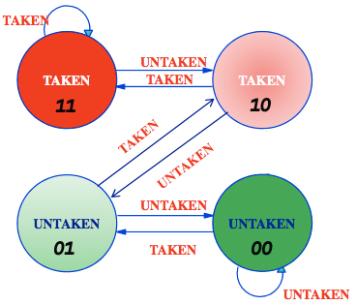
* TAG: 24 bit (30 relativi al PC a cui vengono sottratti i 6 che formano l’Index).  
  Anche se l’architettura RISC del DLX prevede indirizzi a 32 bit, poiché le istruzioni sono di lunghezza fissa (32 bit) e quindi allineate, gli ultimi due bit assumono sempre il valore zero ed è possibile non considerarli. È necessario un Index a 6 bit per l’identificazione univoca dei 64 slot (log2 64 = 6)
* DESTINAZIONE: 30 bit che individuano l’indirizzo di destinazione
* PREDIZIONE: 2 bit (vedi Algoritmo di predizione)
* RIMPIAZZAMENTO: 1 bit (vedi Politica di rimpiazzamento)
* STATO: 1 bit che indica la validità della linea

Quindi ogni via è composta da 58 bit e di conseguenza ogni slot da 116 bit. La dimensione totale della cache è di 116\*64/8 = 928 Byte.

## Algoritmo di predizione

Gli algoritmi di predizione determinano il successo o meno di un BTB poiché sono fondamentali per l’incremento prestazionale della pipeline. Esistono diversi algoritmi, alcuni protetti da segreto industriale.

Nel progetto è stato implementato un algoritmo semplice che prevede l’utilizzo di due bit per la codifica di quattro stati al fine di memorizzare la storia relativa all’istruzione.  
Basandosi sulla correttezza della predizione comunicata dal mondo esterno al BTB è possibile eseguire una transizione di stato seguendo la logica rappresentata nella figura sottostante.



I quattro stati garantiscono una maggiore robustezza nei casi di predizione errata. Infatti sono necessarie due “misprediction” consecutive per cambiare la predizione. Inoltre è possibile invertire repentinamente la predizione qualora si sbagliasse una terza volta.

Con questo algoritmo di predizione è attesa un’accuratezza superiore all’80%.

## Politica di rimpiazzamento

Quando il sistema si trova a regime, a causa della limitata dimensione della cache, a seguito della richiesta d’inserimento di una linea non presente è possibile che occorra rimpiazzare una delle linee appartenenti a un determinato slot. Tra le varie politiche ne è stata scelta una di tipo Least Recently Used (LRU), la quale prevede la sostituzione della linea utilizzata meno recentemente.

Nel caso di una cache set-associative a due vie è necessario un bit per ogni linea di ogni slot. In realtà sarebbe sufficiente un solo bit per ogni slot che indichi la linea da rimpiazzare, ma è stato scelto di utilizzare due campi dedicati per ogni slot per fornire una maggiore scalabilità al sistema.

Le situazioni che portano all’applicazione della politica di rimpiazzamento sono due: HIT in lettura e MISS in scrittura.