Branch Target Buffer 5

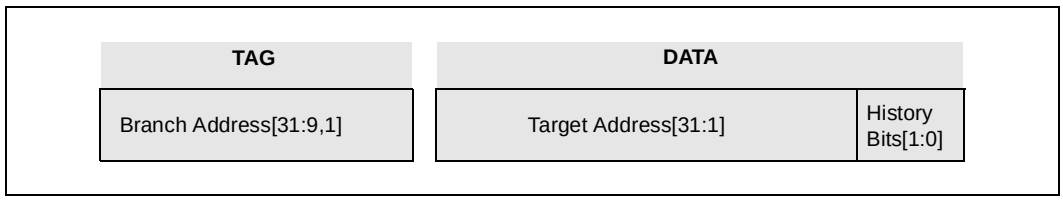
The Intel® XScale™ core uses dynamic branch prediction to reduce the penalties associated with changing the flow of program execution. The Intel® XScale™ core features a branch target buffer that provides the instruction cache with the target address of branch type instructions. The branch target buffer is implemented as a 128-entry, direct mapped cache.

This chapter is primarily for those optimizing their code for performance. An understanding of the branch target buffer is needed in this case so that code can be scheduled to best utilize the performance benefits of the branch target buffer.

5.1 Branch Target Buffer (BTB) Operation

The BTB stores the history of branches that have executed along with their targets. Figure 5-1 shows an entry in the BTB, where the tag is the instruction address of a previously executed branch and the data contains the target address of the previously executed branch along with two bits of history information.

Figure 5-1. BTB Entry



The BTB takes the current instruction address and checks to see if this address is a branch that was previously seen. It uses bits [8:2] of the current address to read out the tag and then compares this tag to bits [31:9,1] of the current instruction address. If the current instruction address matches the tag in the cache and the history bits indicate that this branch is usually taken in the past, the BTB uses the data (target address) as the next instruction address to send to the instruction cache.

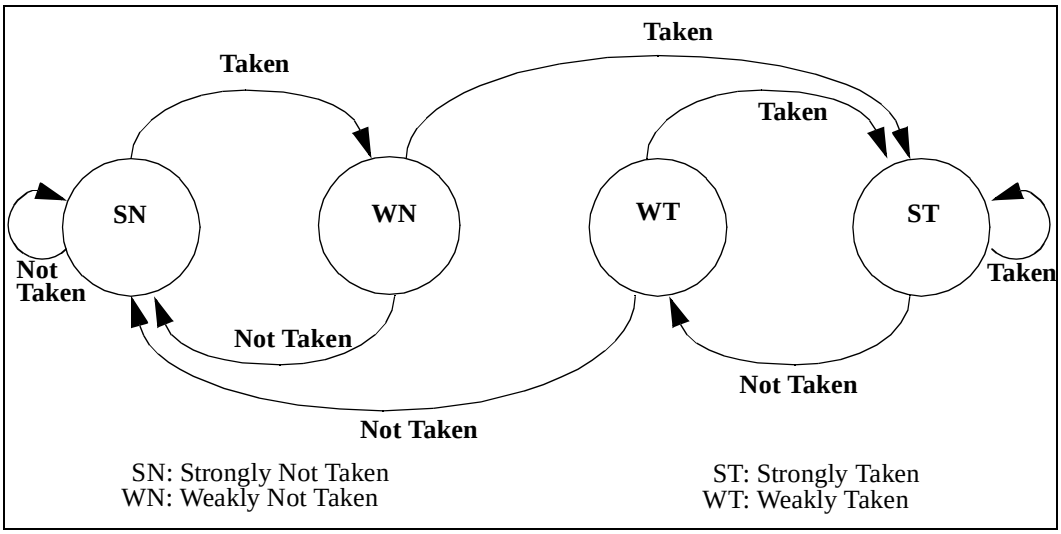
Bit[1] of the instruction address is included in the tag comparison in order to support Thumb execution. This organization means that two consecutive Thumb branch (B) instructions, with instruction address bits[8:2] the same, will contend for the same BTB entry. Thumb also requires 31 bits for the branch target address. In ARM* mode, bit[1] is zero.

The history bits represent four possible prediction states for a branch entry in the BTB. Figure 5-2, “Branch History” on page 5-2 shows these states along with the possible transitions. The initial state for branches stored in the BTB is Weakly-Taken (WT). Every time a branch that exists in the BTB is executed, the history bits are updated to reflect the latest outcome of the branch, either taken or not-taken.

Chapter 10, “Performance Considerations” describes which instructions are dynamically predicted by the BTB and the performance penalty for mispredicting a branch.

The BTB does not have to be managed explicitly by software; it is disabled by default after reset and is invalidated when the instruction cache is invalidated.

Figure 5-2. Branch History



5.1.1 Reset

After Processor Reset, the BTB is disabled and all entries are invalidated.

5.1.2 Update Policy

A new entry is stored into the BTB when the following conditions are met:

- the branch instruction has executed,
- the branch was taken
- the branch is not currently in the BTB

The entry is then marked valid and the history bits are set to WT. If another valid branch exists at the same entry in the BTB, it will be evicted by the new branch.

Once a branch is stored in the BTB, the history bits are updated upon every execution of the branch as shown in Figure 5-2.

5.2 BTB Control

5.2.1 Disabling/Enabling

The BTB is always disabled with Reset. Software can enable the BTB through a bit in a coprocessor register (see Section 7.2.2).

Before enabling or disabling the BTB, software must invalidate it (described in the following section). This action will ensure correct operation in case stale data is in the BTB. Software should not place any branch instruction between the code that invalidates the BTB and the code that enables/disables it.

5.2.2 Invalidation

There are four ways the contents of the BTB can be invalidated.

1. Reset
2. Software can directly invalidate the BTB via a CP15, register 7 function. Refer to Section 7.2.8, “Register 7: Cache Functions” on page 7-11.
3. The BTB is invalidated when the Process ID Register is written.
4. The BTB is invalidated when the instruction cache is invalidated via CP15, register 7 functions.