Preventing Synergistic Processor Element Indefinite Stalls Resulting from Instruction Depletion in the Cell Broadband Engine Processor for CMOS SOI 90 nm

Applications Note

Version 1.0

February 26, 2007—Preliminary

© Copyright International Business Machines Corporation, Sony Computer Entertainment Incorporated, Toshiba Corporation 2007

All Rights Reserved
Printed in the United States of America February 2007

The following are trademarks of International Business Machines Corporation in the United States, or other countries, or both.
IBM
IBM Logo
ibm.com

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc.

Other company, product, and service names may be trademarks or service marks of others.

All information contained in this document is subject to change without notice. The products described in this document are NOT intended for use in applications such as implantation, life support, or other hazardous uses where malfunction could result in death, bodily injury, or catastrophic property damage. The information contained in this document does not affect or change IBM® product specifications or warranties. Nothing in this document shall operate as an express or implied license or indemnity under the intellectual property rights of IBM or third parties. All information contained in this document was obtained in specific environments, and is presented as an illustration. The results obtained in other operating environments may vary.

While the information contained herein is believed to be accurate, such information is preliminary, and should not be relied upon for accuracy or completeness, and no representations or warranties of accuracy or completeness are made.

Note: This document contains information on products in the sampling and/or initial production phases of development. This information is subject to change without notice. Verify with your IBM field applications engineer that you have the latest version of this document before finalizing a design.

THE INFORMATION CONTAINED IN THIS DOCUMENT IS PROVIDED ON AN “AS IS” BASIS. In no event will IBM be liable for damages arising directly or indirectly from any use of the information contained in this document.

IBM Systems and Technology Group
2070 Route 52, Bldg. 330
Hopewell Junction, NY 12533-6351

The IBM home page can be found at **ibm.com®**
The IBM semiconductor solutions home page can be found at **ibm.com**/chips

Version 1.0
February 26, 2007—Preliminary

Introduction

Under some circumstances, the use of a branch hint instruction can cause a Synergistic Processor Element (SPE) to stall indefinitely. SPE instruction prefetch can be interfered with in such a way that an SPE can run out of instructions to execute and make no further progress, while remaining in the RUN state.

SPE indefinite stalls are triggered by certain branch hint instructions. Instruction fetch failures occur when these hinted branches are executed while local storage is very busy servicing reads and writes from the direct memory access (DMA) unit, and program loads and stores. These instruction fetch failures lead to unrecoverable run-out conditions that result in SPE indefinite stalls. An SPE that indefinitely stalls does not corrupt its program state, or any other system resource. A stalled SPE still acknowledges asynchronous interrupts, and resumes typical execution. The SPE in all versions after the 90 nm Cell Broadband Engine™ (CBE) processor have been fixed and execute these code sequences correctly.

It is important to note that the run-out condition occurs occasionally in normally-executing programs and does not by itself lead to the indefinite stall condition. In fact, it is the instruction fetch failure that leads to a run-out condition just before the SPE indefinitely stalls.

There are two steps that should be taken to mitigate the indefinite stall condition:

- Systems incorporating the 90 nm CBE processor should be updated so that their firmware or operating system software includes a device driver that detects the SPE stall condition and restarts the stalled SPE. When the SPE is restarted it will execute normally. A stalled SPE might remain stalled for several thousand cycles before system software can react to an SPE indefinite stall and resume execution. The recovery time depends upon the system software implementation.
- Software running on a 90 nm CBE processor can be coded such that there is no known scenario in which the indefinite stall occurs. The analysis of the issue is difficult; therefore all 90 nm based systems should feature updated system software.

This applications note describes the methods of constructing a system that can recover from the indefinite run-out stall and how to construct software that is free from the currently known causes of indefinite stalls.

Recovery Support in System Software

System software such as operating system components or system firmware can program the performance monitor to detect stalled SPEs. The system can then recover a stalled SPE. Recovery can be accomplished in two ways.

- The synergistic processor unit (SPU) Run Control Register can be used to first stop the SPE and, when it becomes idle, restart it. When the SPE is restarted, it will again make forward progress. Note that this recovery cannot be accomplished through the S bit of the memory flow controller (MFC) State Register 1 (MFC_SR1). The SPE start is a synchronizing event, and clears the hint currently in effect before execution begins.
- System software can also program the thermal throttle controller to simulate a high temperature event and pause the stalled SPE for a short interval of time. When the pause is complete, the SPE clears the hint currently in effect and begins making forward progress.

Systems based on the 90 nm CBE processor might include preexisting software that cannot be recompiled or recoded and therefore rely on the presence of recovery support in the system software. Systems that use these workarounds might not allow access to the performance monitor or thermal throttle controller for other purposes. System software can offer a debug mode wherein it records the address of the SPE instruction where the indefinite run-out stall occurs.

Compiler Options to Enable the Generation of Code that is Free from Indefinite Stalls

Software can be constructed in such a way as to reduce the probability of an SPE indefinite stall. SPU compilers can offer compilation modes that feature varying degrees of safety and performance. See the SPU compiler reference documentation for information about the compiler options that might be available to avoid indefinite stalls.

Decisions about how much performance to trade off for a given degree of safety should be made considering the real time constraints versus the average performance requirements of the software. Software with strict real time constraints might be unable to tolerate the recovery time and might require coding to minimize the risk of an SPE indefinite stall and have known performance. Software that requires very high performance might be coded for performance and allow the system software to recover in the rare case of an indefinite stall.

Information for Assembly Writers and Compiler Implementers

When it is important to avoid indefinite stalls, some restrictions must be placed upon how the inline approaches to hinted branches and their target destinations are coded. If these rules are followed, no known scenario can result in an indefinite stall. Adherence to these rules might increase the number of instructions in the SPE software, or the average run time of the SPE software, or both.

Most often, the negative effects are small and are in code that does not perform loads and stores in bursts of eight or more cycles long. In many cases where there are long bursts of loads and stores, the worst case performance of the software is improved–even for cases where the indefinite stall does not actually occur.

Prohibition of Multiple Branch Hint Instructions

The first rule governs the inline sequence of code that leads to a hinted branch. This inline sequence must not include a hint to some other branch. If there is another branch hint, the SPE software can indefinitely stall, even though the target destinations are carefully coded as described in the following sections.

Coding Rules for Hinted Branch Target Stream

The remaining rules govern the sequence of code after the target of the hinted branch. The target must be coded such that at least one of the following conditions is true:

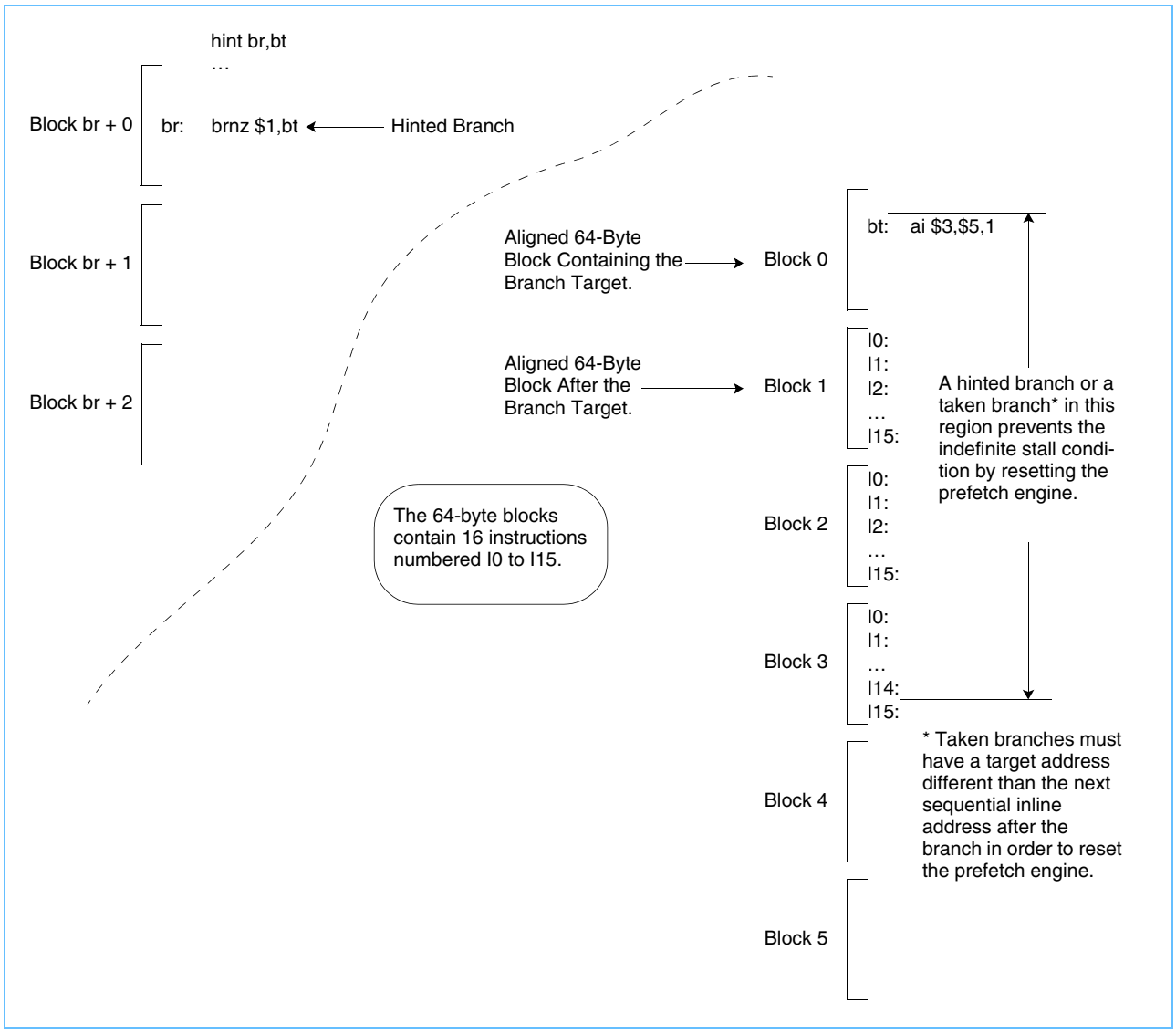
- The instruction prefetch engine is reset before the indefinite stall occurs.
- Instruction prefetch is allowed to access local storage in a timely fashion. The suggested tactic is the judicious insertion of hint prefetch (HBR.P) instructions at key points in the sequence of instructions that follow the branch target.

Prefetch Blocks

The rules presented in this document relate to the blocks of code fetched by the prefetch engine. Instructions are processed according to their position within aligned 64-byte blocks. *Figure 1* on page 5 shows how a hinted branch and the branch target code fit into 64-byte blocks with 16 instructions per block. For ease of

description, names have been assigned to the blocks. Block br+0 contains the hinted branch. Block br+1 is the 64-byte block following block br+0 in local storage. Block br+2 follows br+1 in local storage. Block 0 contains the target of the hinted branch. In the absence of certain special case branches described in *Special Case Branches* on page 6, blocks 1 through 4 follow linearly in local storage after block 0. Note that blocks belonging to the branch and its inline instruction stream (br+0, br+1, and br+2) can overlap with the blocks of the target instruction stream (blocks 0, 1, 2, ...) in an arbitrary manner.

Figure 1. A Hinted Branch and a Target that Can Indefinitely Stall



Resetting the Prefetch Engine Before an Indefinite Stall Occurs

A branch target instruction stream can trigger an indefinite stall if the SPE can execute sequentially inline into the last instruction of block 3. If instruction fetch fails, the instructions from block 4 and later might never be fetched. The SPE might then stall indefinitely, waiting for the necessary instructions to arrive unless the sequential execution ends and the prefetch engine is reset. Sequential inline execution can end in two ways:

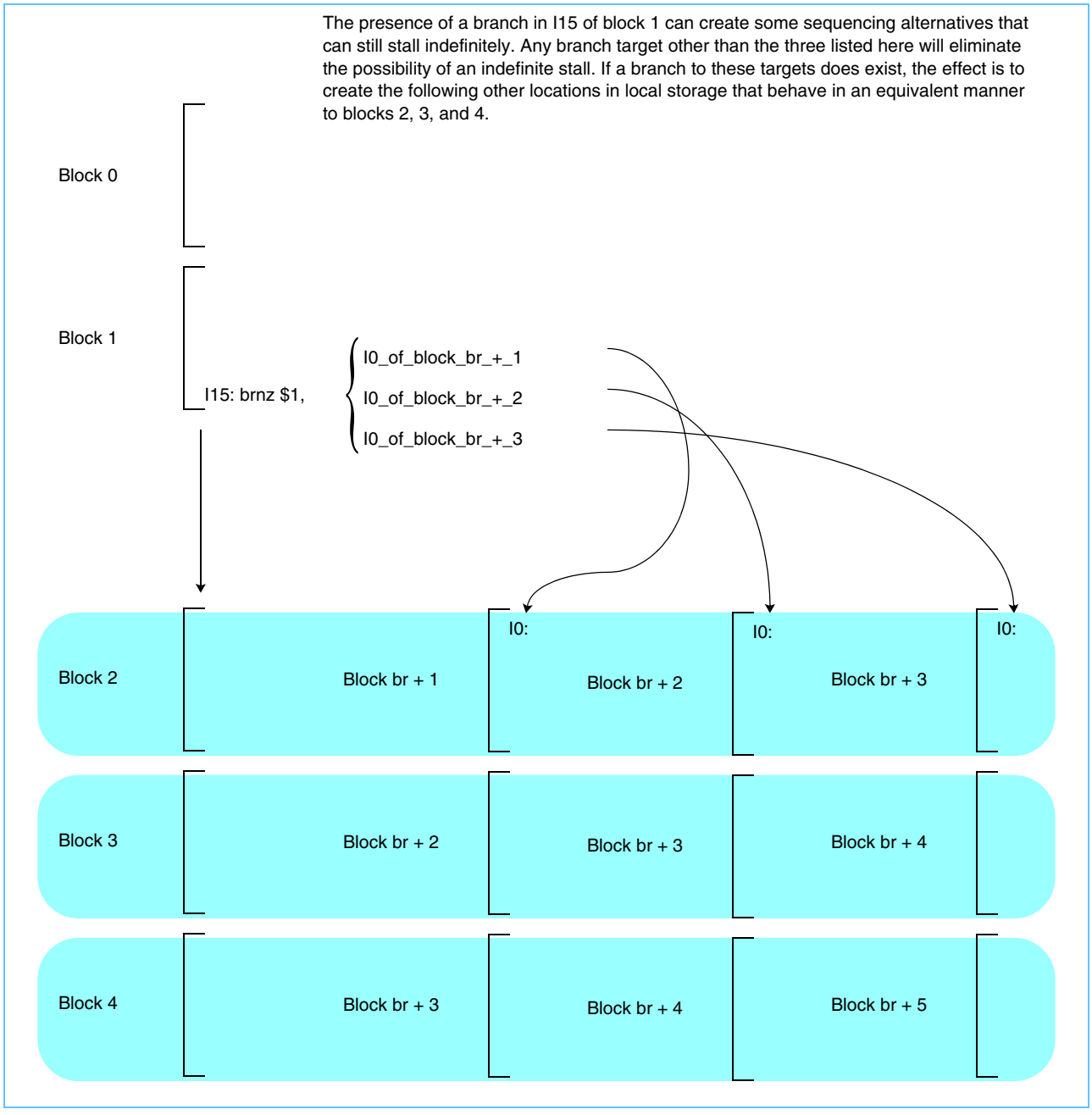
- There can be a mispredicted branch. Branches that are not hinted are speculated to go inline. Therefore, an unhinted branch that branches away from the inline path is mispredicted and causes a pipeline flush that resets the prefetch engine.
- There can be a hinted branch. Hinted branches end sequential inline execution by executing from the hinted target, rather from the inline path. Of course, the hinted target can be another opportunity for an indefinite stall.

Thus, the presence of a hinted branch or a taken branch before I15 of block 3 ensures that the SPE will not indefinitely stall in this target stream. Note that unhinted taken branches that go to the instruction after the branch are processed in the same manner as a no-op instruction, and do not typically cause a pipeline flush, and therefore do not serve as a guarantee against an indefinite stall in this target stream.

Special Case Branches

Certain unhinted but taken branches, called “special case branches”, complicate the definitions of blocks 2, 3, and 4. These branches can alter the defined locations of these blocks to other local storage addresses. *Figure 2* on page 7 shows a branch at I15 of block 1. If this branch happens to go to one of the three addresses in the inline path of the original hinted branch, the SPE can speculate to one of these addresses and avoid a pipeline flush. The three target addresses that can avoid a pipeline flush are I0 of blocks br+1, br+2, and br+3. When instructions are sequenced from these addresses, they are in the place of block 2, and the SPE can still stall indefinitely if it reaches the equivalent of block 4.

Figure 2. Special Case Branch at I15 of Block 1



Block 3 can also end with a special case branch. Again the block 3 special case branch can go to its target without a pipeline flush and allows the SPE to make forward progress in a manner where an indefinite stall is still possible. Block 3 special case branches go to I0 of blocks br+1, br+2, and br+3. Similarly to block 3, block 5 can end with a special case branch. However, block 5 special case branches must go to I0 of block 2.

Block 5 special case branches are not allowed when method 2, presented later, is used to prevent indefinite stalls. Note that the existence of a block 1 special case branch does not prevent the existence of block 3 or block 5 special case branches. In summary there are three important special case branches:

1. An unhinted branch at I15 of block 1 that targets I0 of block br + 1, br + 2, or br + 3
2. An unhinted branch at I15 of block 3 that targets I0 of block br + 1, br + 2, or br + 3
3. An unhinted branch at I15 of block 5 that targets I0 of block 2

Facilitating Timely Prefetches by Prefetch Hint (HBR.P) Instruction Insertion

Instruction prefetch has the lowest priority of access to the local storage. It waits until there is an available cycle when no DMA, program load or store, or hint instruction is scheduled. If prefetch is prevented for long periods of time, the SPE can eventually run out of instructions and pause in a run-out state, until the pipeline drains and prefetch can access local storage. In general, branch target streams stall when the instruction prefetch unit cannot access local storage in a timely fashion. However, a small number of scenarios exists where the prefetch unit is blocked from the local storage, triggering a prefetch-engine failure, leading to an indefinite stall.

In order to prevent the instruction-fetch failure, it is important to allow the queued instruction prefetches to access local storage before certain instructions are issued. Instruction prefetch can be aided by the execution of HBR.P instructions. HBR.P instructions are an implementation-specific prefetch timing hint. HBR.P instructions are intended to help software avoid the loss of performance associated with instruction run-out in code that has long bursts of local storage activity. In the 90 nm SPE, HBR.P instructions stall until local storage is available to accept a prefetch request. When the HBR.P instruction reaches the instruction prefetch unit, the highest priority prefetch is sent to local storage. DMA local storage access obeys certain timing rules, but is essentially asynchronous to instruction execution and can use an almost arbitrary set of cycles during the execution of the program. The SPE issues instructions that do not use local storage while the DMA is using local storage. Thus, the only easy method to guarantee that instruction fetch can access local storage before a certain instruction is issued is to insert HBR.P instructions. See the *Synergistic Processor Unit Instruction Set Architecture* for information about the HBR instruction and the prefetch (P) bit.

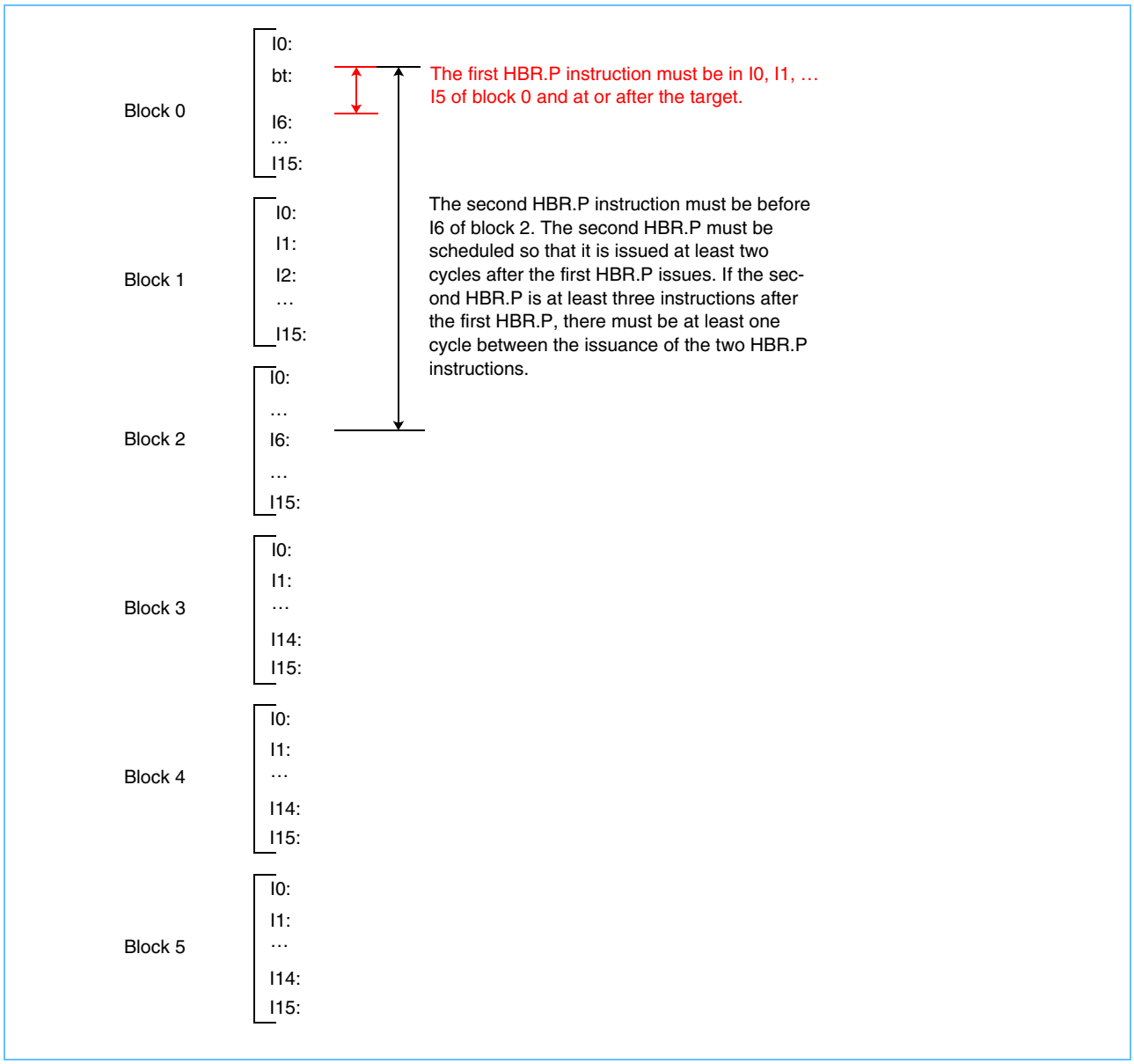
In the 90 nm SPE, instruction prefetch requires that at least two cycles be free on the 128-byte read out path of local storage. This requirement means that HBR.P instructions must be scheduled so that there is at least one cycle between the cycle they issue and the cycle that another HBR.P or branch hint instruction issues, to ensure that instruction fetch can occur. If there is not a cycle of separation, the instruction prefetch logic will ignore the HBR.P instruction and wait for another opportunity. If the HBR.P instruction is ignored, it does not help avoid any possible indefinite stall.

A branch target stream can be coded with HBR.P instructions in one of three ways to eliminate the known scenarios of indefinite stalls. If none of these methods are possible, a branch target stream is still safe if none of the branches that target it are hinted.

Method 1

This method of preventing an SPE indefinite runout stall requires two HBR.P instructions after the branch target. The first of these HBR.P instructions must be before I6 of block 0, and the second HBR.P must be before I6 of block 2. If the branch target is in I6 or later of block 0, this method cannot be used. Note that in method 1 the prefetch of blocks 2 and 3 are completed correctly, so a special case branch at I15 of block 1 will cause the pipeline to flush. Special case branches in blocks 3 and 5 also cause pipeline flushes if method 1 is used. See *Figure 3* on page 9 to visualize the positioning of the HBR.P instructions.

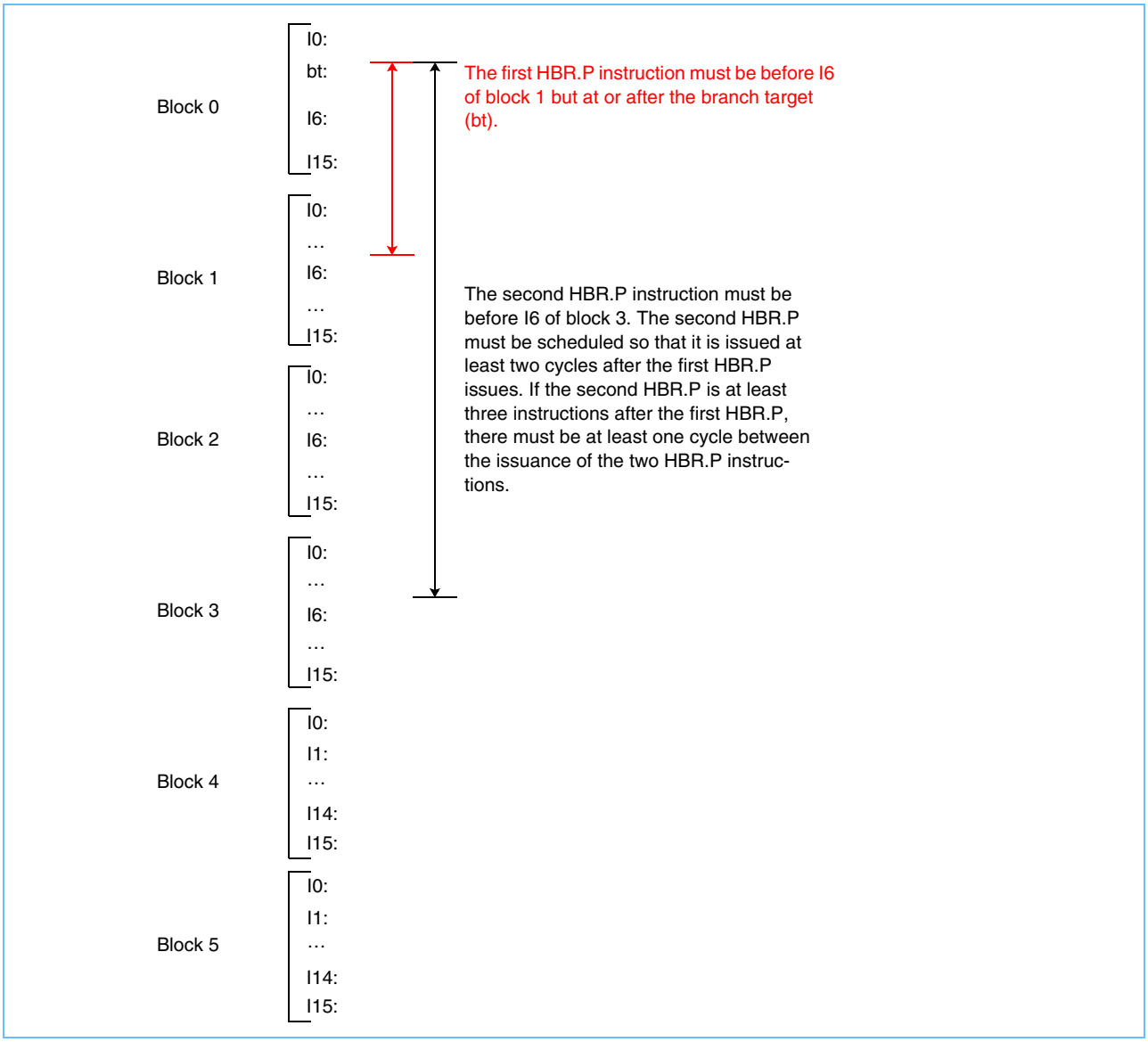
Figure 3. Method 1



Method 2

This method of preventing an SPE indefinite runout stall requires two HBR.P instructions after the branch target and no special case branch at I15 of blocks 1, 3, or 5. The first HBR.P instructions must be before I6 of block 1. The second HBR.P must be before I6 of block 3. See *Figure 4* to visualize these requirements.

Figure 4. Method 2

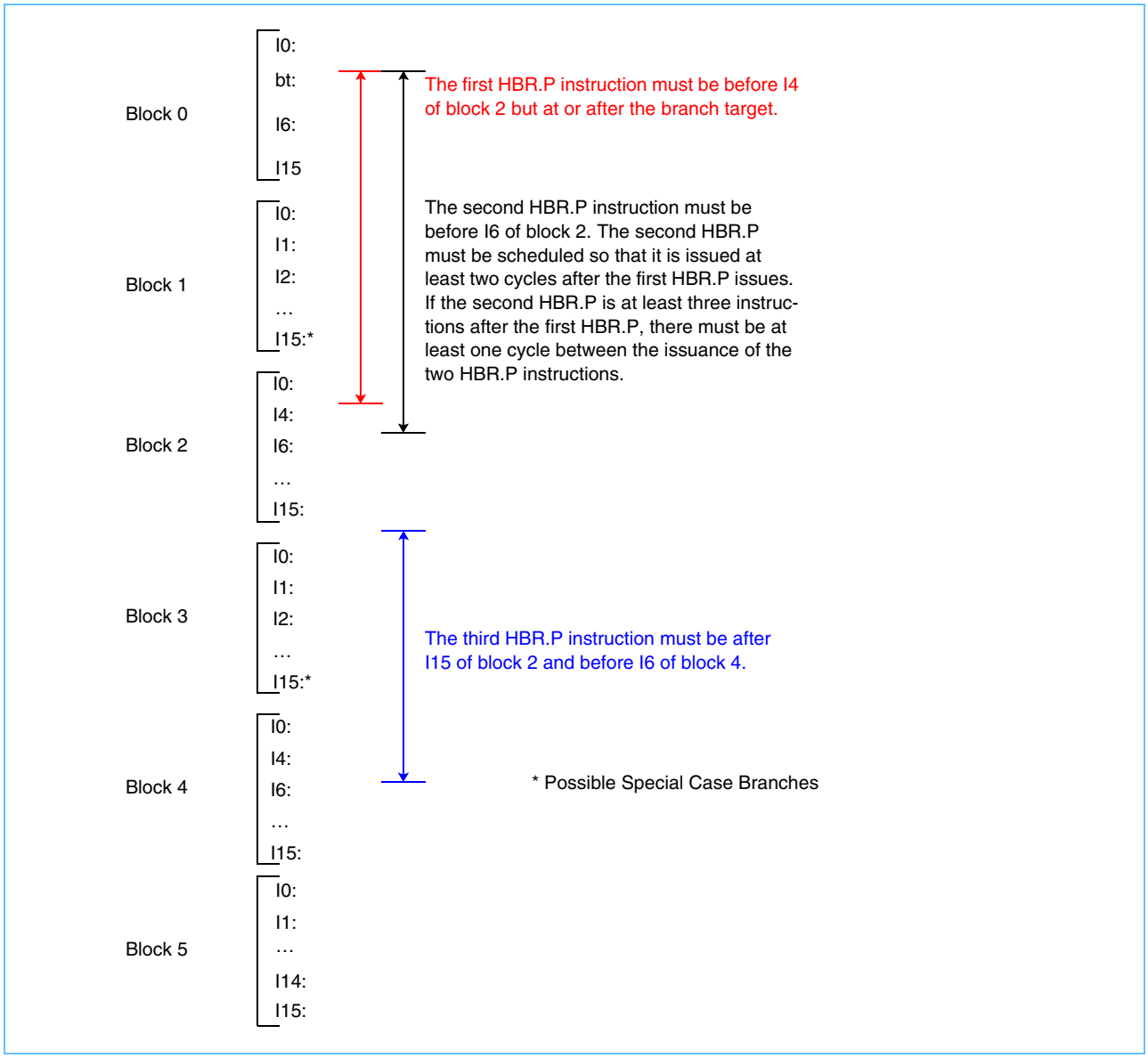


Method 3

This method of preventing an SPE indefinite stall requires there to be three HBR.P instructions after the branch target. The first HBR.P instructions must be before I4 of block 2. The second HBR.P must be before I6 of block 2. The third HBR.P must be after I15 of block 2 but before I6 of block 4.

There can be important special case branches at I15 of blocks 1 and 3 that alter where and how many HBR.P instructions must be inserted. A special case branch at the end of block 5 causes a pipeline flush. See *Figure 5* on page 11 to visualize these positional requirements. For example, if such a branch is present at I15 of block 1 and the first HBR.P has not been placed before I15 of block 1, the first HBR.P must be placed in I0 through I3 of the block of the special case target (the alias for block 2 shown in *Figure 2* on page 7) and also, in the case that the branch is conditional, I0 through I3 of block 2 from the inline path of the special case branch. Special case branches in block 3 alter the block 4 HBR.P placement in a similar manner.

Figure 5. Method 3



Summary of Coding Methods

Table 1 on page 12 summarizes the cases in which each of the three target coding methods is applicable. Note that method 1 is preferable to method 2, if it is applicable, because it features prefetch hints earlier in the instruction schedule where they reduce the probability of performance loss due to instruction run-out. Method 2 requires fewer prefetch hints than method 3. Method 3 also features more alternative sequencing options that can make analysis much more complex.

Table 1. Target Method Applicability

Method	Number of HBR.P	Can be used if target is I6 or later of block 0	Can be used if there are special case branches
1	2	No	Yes
2	2	Yes	No
3	3	Yes	Yes

Revision Log

Revision Date	Version	Contents of Modification
February 19, 2007	1.0	• First revision.