FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Minimal Preconditions for Timing Anomalies in WCET Calculations

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Christoph Martinek
Matrikelnummer 0425174

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer: Ao.Univ.-Prof. Dipl.-Ing. Dr. tech. Peter Puschner

Wien, 20.04.2011

(Unterschrift Verfasser/in) (Unterschrift Betreuer)

Christoph Martinek, Leobendorferstrasse 69/3, 2105 Unterrohrbach

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 20.04.2011,

Abstract

In real-time systems it has an important relevance to analyse the worst-case-execution-time (WCET). With the knowledge of the maximum execution time it is possible to predict the behaviour of time-critical systems. To get a detailed analysis of the timing behaviour, it is necessary to know about the control flow of a program in modern real-time systems, in which different cache- and pipeline-architectures are used. So called timing anomalies could get analysed with the information of the control flow and the instructions. Along with this, necessary conditions for their appearance could be determined. If timing anomalies occur, the attribute proportionality in timing behaviour is violated during an execution.

In this thesis the most important timing anomalies are characterized and techniques to calculate the WCET are presented. There are series timing anomalies, which appear on a number of different paths along an execution, and parallel timing anomalies, where the timing-relevant dynamic computer state (TRDCS) is partitioned into different hardware components.
It is necessary for a strong analysis to investigate timing anomalies in cache- and pipeline-architectures. Timing anomalies combined with the cache replacement strategies FIFO, pseudo round robin and pseudo LRU are observed. Also pipeline-architectures like simple scalar-, scalar-, superscalar in-order and superscalar out-of-order-pipelines in association with timing anomalies are explored. For timing anomalies, which occur in these architectures, preconditions are elaborated. To classify a specific processor with a potential timing anomaly a checklist is created.

Kurzfassung

Bei Echtzeitsystemen ist die Analyse der Worst-Case-Execution-Time (WCET) von entscheidender Bedeutung. Die Kenntnis über diese maximale Ausführungszeit führt zu einer notwendigen Vorhersage des Zeitverhaltens bei zeitkritischen Systemen. Um eine detaillierte Untersuchung des Zeitverhaltens eines Programms durchzuführen, ist in modernen Echtzeitsystemen, in denen unterschiedliche Cache- und Pipeline-Architekturen vorkommen, das Wissen über den Verlauf einer Ausführung von Wichtigkeit. Mit diesen Informationen können Besonderheiten, nämlich sogenannte Zeitanomalien, analysiert und notwendige Bedingungen dafür hergeleitet werden. Beim Auftreten von Zeitanomalien wird im Verlauf einer Ausführung die Eigenschaft Proportionalität im zeitlichen Verhalten verletzt.

In dieser Diplomarbeit werden die wichtigsten Zeitanomalien charakterisiert und Methoden vorgestellt, mit denen sich gewisse Anomalien bezüglich der Berechnung der WCET berechnen lassen. Darunter fallen serielle Zeitanomalien, welche über eine Anzahl von unterschiedlichen Pfaden auftreten können, und parallele Anomalien, bei denen der timing-relevant dynamic computer state (TRDCS) in unterschiedliche Hardwarekomponenten aufgeteilt wird.
Für eine aussagekräftige Analyse ist eine genaue Erforschung von Zeitanomalien in verschiedenen Cache- sowie in Pipeline-Architekturen erforderlich. Es werden hier die Cache-Ersetzungsstrategien FIFO, Pseudo Round Robin und Pseudo LRU genauer betrachtet. Bei Pipeline-Architekturen werden Simple Scalar-, Scalar-, Superscalar In-Order und Superscalar Out-of-Order-Pipelines untersucht. Es werden Vorbedingungen für Zeitanomalien ausgearbeitet, die in diesen Architekturen auftreten können. Darüber hinaus wird gewissermaßen eine Checkliste erarbeitet, mit der Prozessoren betreffend der eintretenden Zeitanomalien eingestuft werden können.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Real-time computing

In real-time systems there are deadlines in which a computation must be finished. These deadlines have to be satisfied within a specific time interval. The duration of this interval is irrelevant, but it must know a priori. That means, a real-time system does not only make a right computation, but also it must be right in the time domain. Thus there is an additional distinction in hard- respectively soft-real-time systems, based on the deadline restrictions.

Soft real-time systems

A soft real-time system can tolerate a missed deadline and admit a decreased service quality. An example of such a system is a video conference, where a video frame or an audio sequence could not be processed correctly. But those real-time systems should have an average load factor, that the majority of calculations are processed within the deadline interval.

Hard real-time systems

In hard real-time systems the deadlines must be met. If computations not satisfy the deadline, these computations are said to be failed. In this case such a failure can damage the whole system and furthermore human lives and the environment threatened. An example of a real-time system is the motor-management system of a car. In general hard real-time systems are applied in embedded systems, which react within a closely connection to its environment.
If all necessary computations are known a priori, then the system could be designed time-controlled. In this design style there are time slots for each computation. Thereby it is possible to get a almost fully system load. But the computations must be finished within the allocated time slot. To derive the duration of a computation it is necessary to assume the worst case. Hence it is essential to calculate the worst-case-execution-time (WCET) of an instruction.

1.2 Structural Organization

First we start with some basic concepts and definitions of timing anomalies in modern processor architectures in Chapter 2. Furthermore the notation is defined in the chapter and additional some composition techniques are explained.
In Chapter 3 the advantages of using pipeline architectures and the later analysed pipeline models are introduced.
The second part of the debated architectures, the caches, are presented in Chapter 4. Additional the functionality and concepts are discussed.
In Chapter 5 and 6 series timing anomalies are analysed regarding pipeline and cache architectures. Furthermore we differ between inversion and amplification timing anomalies and the minimal preconditions for their occurrence are under discussion.
In Chapter 7 the combination of the before treated architectures are in the focus of the parallel timing anomaly analysis, where an execution in the cache affects the behaviour of the pipeline execution.

CHAPTER 2

Basic Concepts and Related Work on Timing Anomalies

2.1 Overview

In this thesis the concentration is on timing anomalies occurring in the calculation of the WCET. Thus a short summary about the WCET calculation is presented first. Next the notation, which is used along the thesis, is shown. Finally timing anomalies are described and some composition techniques to compute the WCET are reported.

2.2 WCET Analysis

In modern processor architectures, where timing anomalies can appear in cache and pipeline architectures, the estimation of the WCET can only be done at substantial expense, see [19, 8, 2, 4]. For this analysis it is necessary to determine the longest possible path through an execution [6, 3]. This can be done with the control-flow analysis. With this we know a set of all possible paths along an execution and can now study them. A clever method is to use the 'divide and conquer'-concept, thus to derive the maximum duration of a statement and do this along the execution in a bottom-up manner. Unfortunately this method leads not always to an appropriate result for the WCET. This is due to the cache- and pipeline-architectures of modern processors. With these components so called timing anomalies can occur, the discussed subject of this thesis. These timing anomalies violate the notion of proportionality in the timing domain in a certain manner.

2.3 Timing Relevant Dynamic Computer State

With the timing-relevant state model, which is introduced in [5], interesting computer states can be examined. In this model several computer states are defined. The first one is the overall

computer state CS, which includes all states for different components like external data memory, external cache, external code memory, input-, output-devices and of course the processor state.
A subset of the CS is the timing-relevant computer state TRCS. In this set all elements which are relevant for the timing of an execution are combined. The next subset of the TRCS is the timing-relevant computer configuration TRCC. Within this set are timing-relevant elements, but they are constant. A graphical model of the definitions is presented with Figure 2.1. These definitions bring us finally to the timing-relevant dynamic computer state TRDCS, which contains those elements that are in the timing-relevant computer state, though they are not in the timing relevant computer configuration.
Thus in all following chapters only states in the TRDCS state are of interest. Note that timing anomalies, which are introduced in the next chapters, are trivial to produce in cases, where the whole state space or timing irrelevant states are involved. Therefore the concentration of the WCET analysis is only on the essential part of the Computer State, namely the TRDCS.

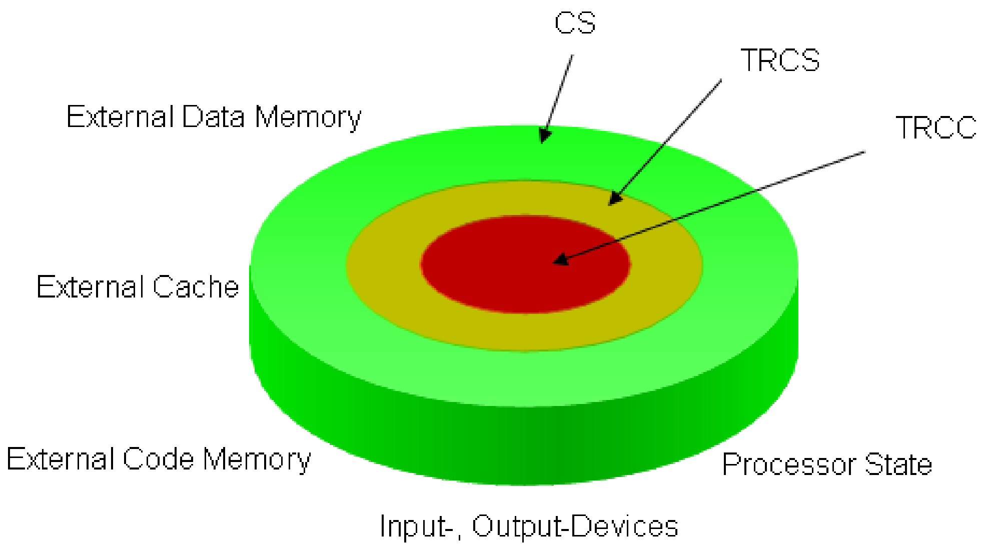


Figure 2.1: Timing Relevant Dynamic Computer State

2.4 Notation

For a detailed analysis it is necessary to get familiar with the formal definitions. The following definitions and achievements are principal obtained by [6].

Defintion 2.4.1 $T(I, s)$ *is the execution time of an instruction sequence I (which can consists of particular instruction* $I_1 \circ I_2 \circ I_3 \circ \ldots$*, where the* $\circ$ *operator connect instructions in a sequential manner) beginning from the initial state* $s \in$ *TRDCS.*

Defintion 2.4.2 $T_{max}(I, S) = max(T(I, s)|s \in S)$ *is the maximum of all execution times for instruction I with an initial state* s *of the set of potential initial states* S.

Defintion 2.4.3 $\Delta(I, s, s') = T(I, s') - T(I, s)$ *is the difference of execution times of instruction sequence I for different initial states* $s, s' \in$ *TRDCS.*

Defintion 2.4.4 $\mathbb{IN}_I$ *is the set of potential initial states of instruction I.*

Defintion 2.4.5 $\mathbb{IN}_{I,max} = \{s \in \mathbb{IN}_I \wedge \forall s' \in \mathbb{IN}_I.T(I, s') \leq T(I, s)\}$ *is the set of initial states of instruction I, where* $T(I, s)$ *is maximal.*

The following definitions apply to the partition of the TRDCS in hardware components of interest for the timing of the instruction. These definitions imagine, that two different components (like pipeline and cache) can analysed for itself and join the results for an overall analysis.

Defintion 2.4.6 $T_{hw_A}(I, a)$ *is the execution time for instruction I of hardware component* hw_A *with the initial local state* $a \in A$*, where* A *is the set of initial states for the TRDCS component* hw_A*. When another component is in progress and consume execution time, but hardware component* hw_A *is not involved, then this does not count to* $T_{hw_A}(I, a)$.

Defintion 2.4.7 $\Delta_{hw_A}(I, a, a') = T_{hw_A}(I, a') - T_{hw_A}(I, a)$ *is the difference of the execution times for the instruction I for different initial states* a *and* $a' \in A$ *for the hardware component* hw_A.

2.5 Related Work on Timing Anomalies

The first exploration of timing anomalies in association with WCET calculation is done by Lundqvist and Stenström in [9]. They introduced a concept, where a cache miss results in a decreasing execution time in an out-of-order pipeline and cache processor, namely a simplified PowerPC architecture processor. Also they assert, that in in-order pipelines no timing anomalies are appear, which was invalidated later. In this paper also a formal definition about timing anomalies are given and examples of domino effects are shown. Additionally a simple code modification method is displayed to inhibit timing anomalies.
In [1, 12] a more precise study on domino effects and the effect of an unbounded WCET. He shows some examples with the pseudo least recently used (PLRU) cache replacement strategy.
A classification in scheduling-, speculating- (also [14]) and cache-timing anomalies are given in [13]. Whereas the latter two classification classes can appear in in-order architectures.
In the paper of Wenzel et al. [18] they acquire the resource allocation criterion, which is a necessary, but not a sufficient condition for the appearance of timing anomalies. With this criterion it is possible to evaluate, if an anomaly can occur. But the resource allocation criterion limits the occurrence of anomalies in that sense, that only one instruction can change the timing. Also they provide, that timing anomalies can occur in simply models, like in-order architectures.
In [5] Kirner et al. presenting formal definitions of serial and parallel timing anomalies and categorize them in amplification and inversion classes. Also they had shown some composition

techniques for parallel timing anomalies, in particular the delta- and max-composition, and the series-composition for series timing anomalies. Furthermore parallel timing anomalies are described more in detail in case of a simultaneously appearance.

2.6 Fundamental Definition of Timing Anomalies

Lundqvist and Stenström were the first, who explored the phenomenon of timing anomalies and described them with the following words: *"Previous timing analysis methods have assumed that the worst-case instruction execution time necessarily corresponds to the worst-case behaviour. We show that this assumption is wrong in dynamically scheduled processors."* [9]. Furthermore they explained that a cache miss can provide a shorter overall execution than a cache hit.
In [13], Reineke et al. described a timing anomaly as the circumstances, where a local worst-case situation does not affect a global worst-case situation. Kirner et al. characterize a timing anomaly as a case where the persistence properties of proportionality and monotony in the timing behaviour invalidates the continuity characteristic in [6].

2.7 Timing Anomaly Definition by Lundqvist and Stenström

A first definition of timing anomalies was introduced by Lundqvist and Stenström in [9]. They described it in the following manner.

"Consider the execution of a sequence of instructions. Let us study two different cases where the latency of the first instruction is modified. In the first case, the latency is increased by i clock cycles. In the second case, the latency is decreased by d cycles. Let C be the future change in execution time resulting from the increase or decrease of the latency. Then:" ([9])

Defintion 2.7.1 *"A timing anomaly is a situation where, in the first case, $C > i$ or $C < 0$, or in the second case, $C < -d$ or $C > 0$." (Definition 1 in [9])*

That means a timing anomaly does not happen as described in [9], if the future change of the remaining execution C is in the interval $0 \leq C \leq i$ in the first case or $-d \leq C \leq 0$ in the second case. In Figure 2.2 these definitions are explained in a graphical manner.

The duration of the first instruction M is represented as thick black line and the duration of the overall execution I is indicated as a rectangle. The difference of the entire execution between $T(I, s_0)$ and $T(I, s_i)$ (with $1 \leq i \leq 4$) is labelled as C_i.
In the first case, $T(M, s_1)$ and $T(M, s_2)$ need i more clock cycles for their executions than the reference execution $T(M, s_0)$. But the difference $C_1 = T(I, s_1) - T(I, s_0)$ is greater than the difference of the first instruction $i = T(M, s_1) - T(M, s_0)$. This timing anomaly is discussed later as *amplification timing anomaly*. The difference $C_2 = T(I, s_2) - T(I, s_0) < 0$ brings us to the timing anomaly we will later call *inversion timing anomaly*.
In the second diagram a decreased duration $d = T(M, s_3) - T(M, s_0)$ of the first instruction M between the reference execution $T(M, s_0)$ and $T(M, s_3)$ (resp. $T(M, s_4)$) is presented. Hence $C_3 = T(I, s_3) - T(I, s_0)$ is smaller than $-d$. This results again in an amplification timing

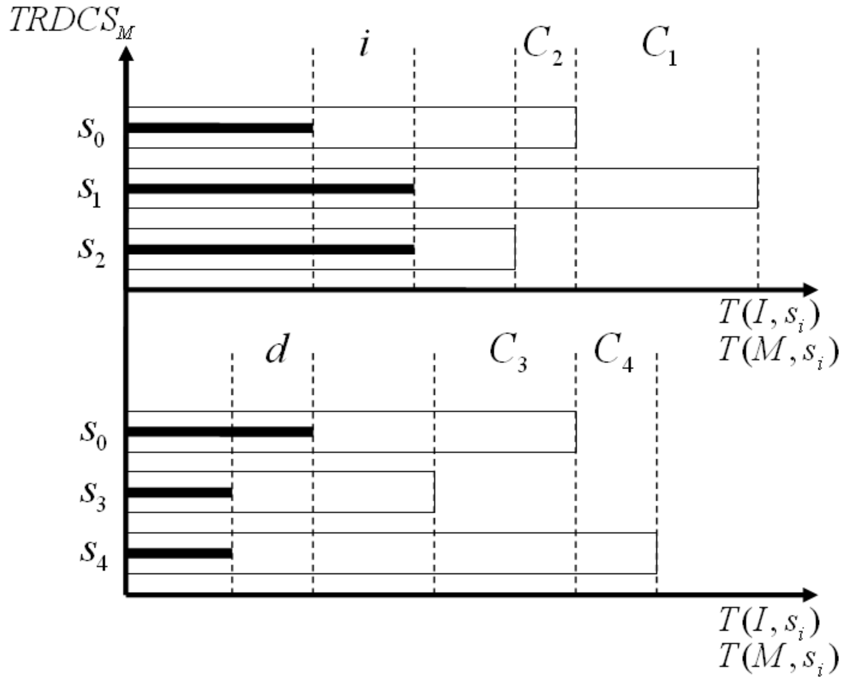


Figure 2.2: First Definition of Timing Anomalies

anomaly, but in a decreasing way. On the other hand $C_4 = T(I, s_4) - T(I, s_0) > 0$ and leads to an inversion timing anomaly.

2.8 Timing Anomaly Definition by Reineke et al.

Reineke et al. defined a program execution, where a path π is partitioned in three parts (π_{pre}, $\pi_{|l}$ and π_{post}), where l is a locally constraint for $\pi_{|l}$ in [13]. Following the non-local worst-case path leads to a global worst-case path for the execution of π. In the Figure 2.3 this concept is illustrated with a realization, where the first execution has the local worst-case path; however the last execution (with the non-local worst-case path) is the crucial path for the WCET calculation.

2.9 Series Timing Anomalies

Timing anomalies which can be handled with the series decomposition technique (described later) are labelled as 'series timing anomalies'. This category of timing anomalies is in general observed, if an instruction sequence is split into two different streams (whereas an instruction stream can be consists of only one instruction). Then the relation between the execution of the first instruction stream and the whole instruction sequence is analysed and in case of a timing anomaly, this situation is called a series timing anomaly. Within the definition of the series timing anomalies, all initial states are elements from the TRDCS. These definitions based on the

definitions of Lundqvist and Stenström [9] and extended by Kirner et al. in [6].
Series timing anomalies differ in their impact on the timing behaviour. In case of *inversion series timing anomalies (TA-S-I)* the execution time of the whole instruction sequence change in an opposite direction than the first sequence. On the other hand if the execution of the whole instruction stream follows the direction of the alteration of the first instruction sequence execution and amplifies this change, this is called an *amplification series timing anomalies (TA-S-A)*. The followed definitions are mostly obtained from [6].

Defintion 2.9.1 ***TA-S-I***
Assume an instruction sequence $I = M \circ N$, which consists of two non empty instruction sequences M and N and the initial states $s, s' \in TRDCS$. Then an inversion series timing anomaly is a situation, where:
$\exists s, s' \in \mathbb{IN}_M.$
$\Delta(M, s, s') > 0 \ \wedge \ \Delta(I, s, s') < 0$

Defintion 2.9.2 ***TA-S-A***
Assume the same instruction sequence $I = M \circ N$ as in the above definition and the initial states $s, s' \in TRDCS$. Then an amplification series timing anomaly is a situation, where:
$\exists s, s' \in \mathbb{IN}_M.$
$0 < \Delta(M, s, s') < \Delta(I, s, s')$

The timing anomaly TA-S-I is also called *'strong timing anomaly'*, because this anomaly cannot be analysed with the series decomposition without problems. The TA-S-A timing anomaly is named *'weak timing anomaly'*, since a serious calculation of the WCET can be achieved with the series decomposition technique.

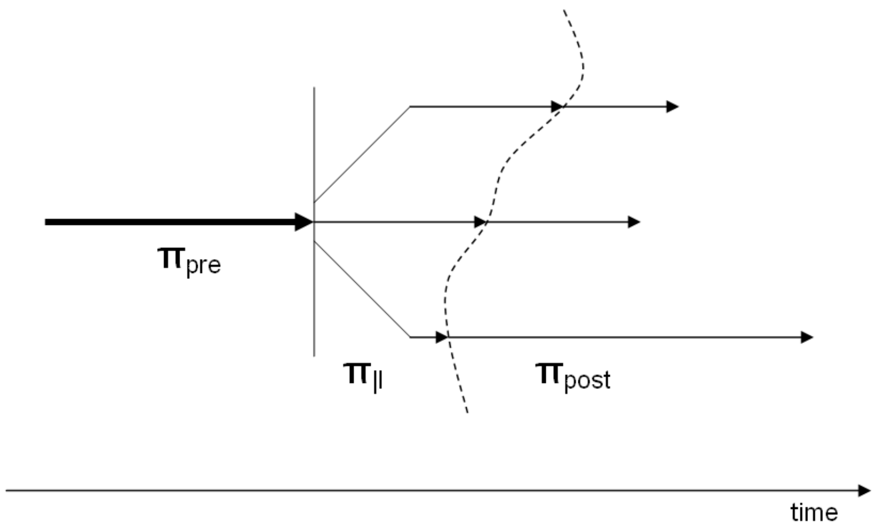


Figure 2.3: Definition of Timing Anomalies by Reineke et al.

The next definitions highlight the cases, where series timing anomalies affect the calculation of the WCET. For that reason these timing anomalies are termed *'worst-case series timing anomalies'*. Worst-case inversion anomalies of this kind are labelled as *TAW-S-I* and worst-case amplification anomalies as *TAW-S-A*.

Defintion 2.9.3 ***TAW-S-I***
Assume an instruction sequence $I = M \circ N$, which consists of two non empty instruction sequences M and N and the initial states $s, s' \in$ TRDCS. Then a worst-case inversion series timing anomaly is a situation, where:
$\exists s \in \mathbb{IN}_M, \forall s' \in \mathbb{IN}_{M,max}.$
$\Delta(M, s, s') > 0 \lor \Delta(I, s, s') < 0$

Defintion 2.9.4 ***TAW-S-A***
Assume the same instruction sequence $I = M \circ N$ as in the above definition and the initial states $s, s' \in$ TRDCS. Then an amplification series timing anomaly is a situation, where:
$\exists s \in \mathbb{IN}_M, \forall s' \in \mathbb{IN}_{M,min}.$
$0 < \Delta(M, s, s') < \Delta(I, s, s')$

The difference of the worst-case series timing anomalies and the 'normal' series timing anomalies is first the quantifier of s' changes from $\exists$ to $\forall$ and second for TAW-S-I to $\forall s' \in \mathbb{IN}_{M,max}$ and for TAW-S-A to $\forall s' \in \mathbb{IN}_{M,min}$. This causes a situation, in which the initial state s' is taken from the set $\mathbb{IN}_{M,max}$ (in case of *TAW-S-I*) and thus a more pointed setting is created. In the case of *TAW-S-A* the initial state s' is in the set $\mathbb{IN}_{M,min}$ and this results in a more precise definition of an amplification anomaly due to the WCET calculation.

Series Decomposition

To analyse series timing anomalies, the straight forward technique *series decomposition* is presented in [6]. With this technique it is possible to approximate one worst-case state for each control-flow node instead of k states for k input states and the maximum of two successive instructions is computed. This is an enormous advantage, because the number of collected states is growing exponentially with the program size. If this is done for each control-flow node across the entire execution, then the resulting maximum execution time is actual the WCET bound.

Defintion 2.9.5 ***Series Composition***
Assume an instruction sequence $I = M \circ N$, which consists of two non empty instruction sequences M and N and the initial state $s \in$ TRDCS. Then the WCET of the series composition, T_{sc}, is calculated in the following manner:
$T_{sc}(M \circ N) = \underset{s \in \mathbb{IN}_{M,max}}{max} T(M \circ N, s)$

Amplification timing anomalies do not invalidate this calculation technique and an upper bound for the WCET is provided for this kind of timing anomalies with this technique.
The safeness of the series composition and a sufficient and necessary condition for a safe WCET bound is presented in the next theorem. The proof of the theorem and the theorem itself is shown in [6].

Theorem 2.9.6 ***Safeness of Series Composition***
"Assuming that the program to be analysed is decomposed into connected control-flow nodes and the set of possible subpathes (= sequences of control-flow nodes) of the program is denoted by the set $\mathbb{SP}$, then Series-Composition allows to provide a safe WCET bound on processor hardware whose timing characteristics obey the following sufficient and necessary condition:
$\forall M \circ N \in \mathbb{SP}, \forall s_1 \in \mathbb{IN}_M, \exists s_2 \in \mathbb{IN}_{M,max}.$
$\Delta(M, s_1, s_2) > 0 \rightarrow \Delta(M \circ N, s_1, s_2) \geq 0$*" ([6] Theorem 4.1 and Proof A.1)*

That means that s_2 is taken from the set $\mathbb{IN}_{M,max}$ and this ensures, that the execution time of the instruction M and so $T(M, s_2)$ is maximal. If the difference of $T(M, s_2)$ and $T(M, s_1)$ is greater than zero, then the difference of execution time of the instruction sequence of $T(M \circ N, s_2)$ and $T(M \circ N, s_1)$ must be also greater than or equal to zero.
If we take a precise look to the definition of the *TAW-S-I* timing anomaly one can notice that this is an opposite definition of the safeness condition of the series composition. Even by this reason the series composition can not ensure a serious WCET bound.

In Figure 2.4 the above definitions are explained graphically. In the first diagram the difference $\Delta(M, s_0, s_1)$ is greater than zero and the difference of the instruction sequence $\Delta(I, s_0, s_1)$, with $I = M \circ N$, is also greater than $\Delta(M, s_0, s_1)$. This results in a series amplification timing anomaly *TA-S-A*.
In the second diagram the difference $\Delta(M, s_0, s_2)$ is greater than zero (same as in the first diagram), but the difference $\Delta(I, s_0, s_2)$ is smaller than zero. This is an indication of a series inversion timing anomaly *TA-S-I*.

If we compare the definitions of Lundqvist and Stenström with the definitions of Kirner et al., then we can determine similarities. The number of increased clock cycles i in the definitions of Lundqvist and Stenström correlates with $\Delta(M, s_0, s_1)$ (resp. $\Delta(M, s_0, s_2)$) of the definitions of Kirner et al. The difference of the overall execution times C_1 and C_2 are equivalent with $\Delta(I, s_0, s_1)$ and $\Delta(I, s_0, s_2)$. Furthermore the additional term $C < -d$ or $C > 0$ of Lundqvist and Stenström has been substituted by the permutation of the variables in the Delta-Term.

2.10 Parallel Timing Anomalies

The idea of clarify *parallel timing anomalies* is to partition the TRDCS state space, which was introduced in Chapter 2.3, into two independent state spaces TRDCS $= A \cup B$, e.g. like that from caches and pipelines. Consider that each of the partition areas has a separate timing behaviour. If we observe parallel timing anomalies, then we focus on the comparison of the component latency $T_{hw_A}(I, a)$ of the instruction I of hardware component hw_A (with $a \in A$) and the overall execution time $T(I, \langle a, b \rangle)$ with $b \in B$. The state $\langle a, b \rangle$ is taken from the initial state set $A \times B$. The definitions are obtained from Kirner et al. [6].

Defintion 2.10.1 ***TA-P-I***
Assume an instruction sequence I and a TRDCS initial state space, which is partitioned in two

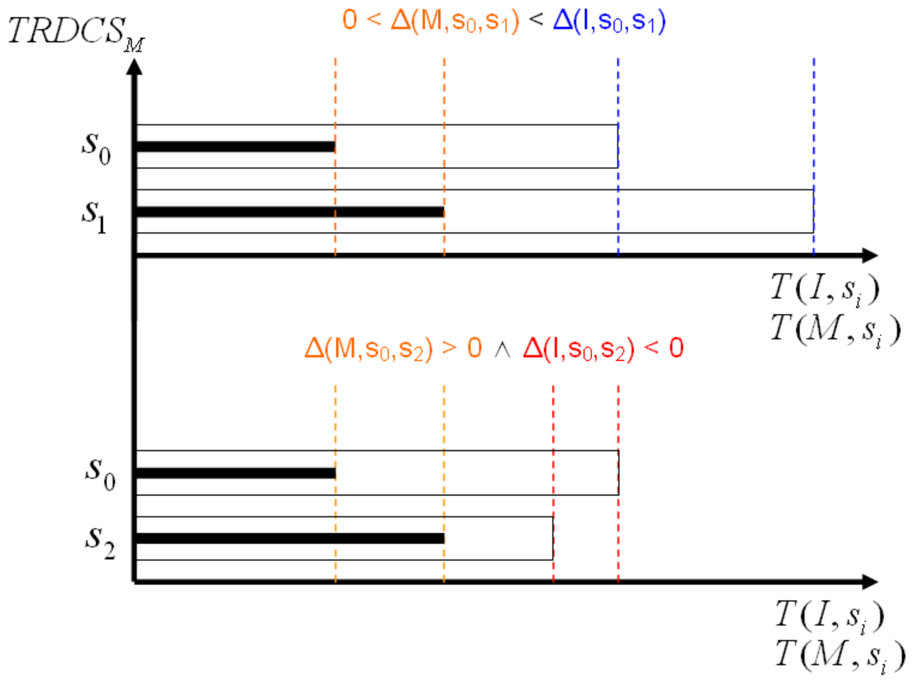

$TRDCS_M$
$0 < \Delta(M,s_0,s_1) < \Delta(I,s_0,s_1)$
s_0
s_1
$T(I,s_i)$
$T(M,s_i)$
$\Delta(M,s_0,s_2) > 0 \wedge \Delta(I,s_0,s_2) < 0$
s_0
s_2
$T(I,s_i)$
$T(M,s_i)$

Figure 2.4: Definition of Series Timing Anomalies

non-empty sets $A \cup B$ with the component latency $T_{hw_A}(I,a)$ of hardware component hw_A and the overall execution time $T(I,\langle a,b\rangle)$, with $a \in A, b \in B$. Then an inversion parallel timing anomaly is a situation, where:
$\exists a, a' \in A, \exists b \in B.$
$\Delta_{hw_A}(I,a,a') > 0 \ \wedge \ \Delta(I,\langle a,b\rangle,\langle a',b\rangle) < 0$

Defintion 2.10.2 ***TA-P-A***
Assume an instruction sequence I with the same partitioning as in the above definition and the identical execution times. Then an amplification parallel timing anomaly is a situation, where:
$\exists a, a' \in A, \exists b \in B.$
$0 < \Delta_{hw_A}(I,a,a') < \Delta(I,\langle a,b\rangle,\langle a',b\rangle)$

A more specific case for the above definitions, the *worst-case parallel timing anomalies*, are shown in the next definitions, like for the series timing anomalies. There are more specific situations indicate where the analysis with the *parallel decomposition* techniques (which will be presented below) could be difficult to handle. First, some helpful sets are introduced.

Defintion 2.10.3
$A_{min} = \{a \in A \mid \forall a' \in A.\ T_{hw_A}(I,a') \geq T_{hw_A}(I,a)\}$
$A_{max} = \{a \in A \mid \forall a' \in A.\ T_{hw_A}(I,a') \leq T_{hw_A}(I,a)\}$
$B_{A,max}(a) = \{b \in B \mid \forall b' \in B.\ T(I,\langle a,b\rangle) \geq T(I,\langle a,b'\rangle)\}$

The set A_{min} describes the set of initial states, which have a minimal execution time for instruction I for hardware component hw_A. The elements of the set A_{max} lead for instruction I and hw_A to a maximal execution time. Additionally the set $B_{A,max}(a)$ includes the initial states $b \in B$, which achieve a maximal execution time $T(I, \langle a, b \rangle)$.

Defintion 2.10.4 ***TAW-P-I***
Assume an instruction sequence I and a TRDCS initial state space, which is partitioned in two non-empty sets $A \cup B$ with the component latency $T_{hw_A}(I, a)$ of hardware component hw_A and the overall execution time $T(I, \langle a, b \rangle)$, with $a \in A, b \in B$. Then a worst-case inversion parallel timing anomaly is a situation, where:
$\exists a \in A, \exists b \in B, \forall a' \in A_{max}, \forall b' \in B_{A,max}(a').$
$\Delta_{hw_A}(I, a, a') > 0 \ \vee \ \Delta(I, \langle a, b \rangle, \langle a', b \rangle) < 0$

Defintion 2.10.5 ***TAW-P-A***
Assume an instruction sequence I with the same definitions as above. Then a amplification parallel timing anomaly is a situation, where:
$\exists a \in A, \exists b \in B, \forall a' \in A_{min}, \forall b' \in B_{A,max}(a').$
$0 < \Delta_{hw_A}(I, a', a) < \Delta(I, \langle a', b' \rangle, \langle a, b \rangle)$

Thus the worst-case inversion parallel timing anomaly originates when the difference between the executions with the initial states a and a', $a' \in A_{max}$ of instruction I at the hardware component hw_A is greater than zero and the difference of the overall execution times is less than zero. Thus the characteristic of the execution of the overall execution proceeds in the opposite direction than the execution at hw_A. The fact, that a' is taken from the set A_{max} invokes a situation in which the term $\Delta_{hw_A}(I, a, a')$ has a maximum value and the term $\Delta(I, \langle a, b \rangle, \langle a', b \rangle)$ reaches a minimal value.
For the worst-case amplification parallel timing anomaly *TAW-P-A* the term $\Delta_{hw_A}(I, a', a)$ (with $a' \in A_{min}$) depicts a value which is slightly greater than zero. And the difference of $T(I, \langle a, b \rangle)$ and $T(I, \langle a', b' \rangle)$ (which forces the maximum execution time for the initial state a') is still greater than the first term.

Illustration of Parallel Timing Anomalies

In Chapter 2.9 an illustration of timing anomalies has been introduced for series timing anomalies. In the following the visualization for parallel timing anomalies is presented. First of all, the instructions which are executed in hw_A (with initial states a_i, $0 \leq i \leq 4$) are presented as black thick lines and are followed by the executions for the other hardware component (with initial states b_i, with $0 \leq i \leq 4$) as black thick lines.
To identify timing anomalies easier, we order the executions $T(I, a)$ based on the required time in descending order, Figure 2.5.

The initial states of the executions are relabeled with a'_i. Next the execution times $T(I, \langle a'_i, b \rangle)$ for all $a'_i \in A$ and one fixed $b \in B$ are built. With this information timing anomalies can now be identified immediately. In the Figure 2.6 a *TA-P-I* example is displayed on the left. If we take a close look at the executions we can observe that $\Delta_{hw_A}(I, a'_4, a'_3) > 0$ and $\Delta(I, \langle a'_4, b \rangle, \langle a'_3, b \rangle) <$

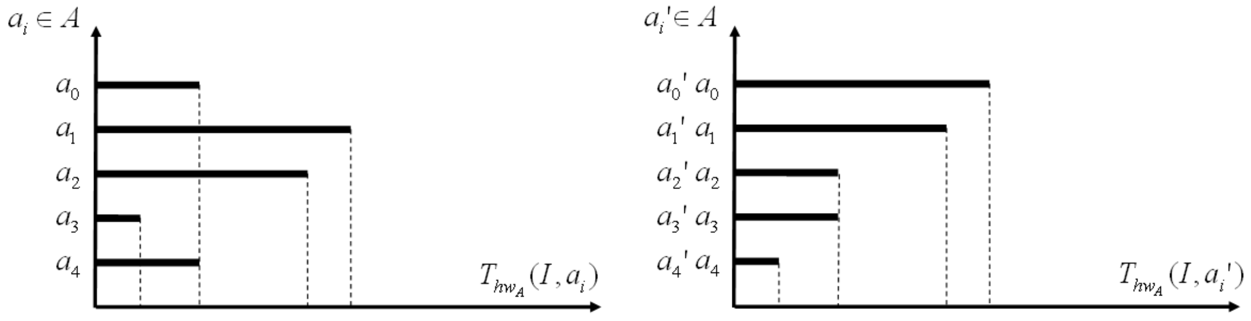


Figure 2.5: Descending order of executions by hw_A

0 and this is equivalent to the definition of the *TA-P-I*. Furthermore we can find that $\Delta(I, \langle a_3', b\rangle, \langle a_2', b\rangle) <$ 0. Is this not also an *TA-P-I*? No, because the first term $\Delta_{hw_A}(I, a_3', a_2') > 0$ is not fulfilled. If the difference of two executions at hw_A is zero, then there is no anomaly.
On the right side we notice a *TA-P-A* timing anomaly, due to the fact that $0 < \Delta_{hw_A}(I, a_1', a_0')$ and this is less than $\Delta(I, \langle a_1', b\rangle, \langle a_0', b\rangle)$.

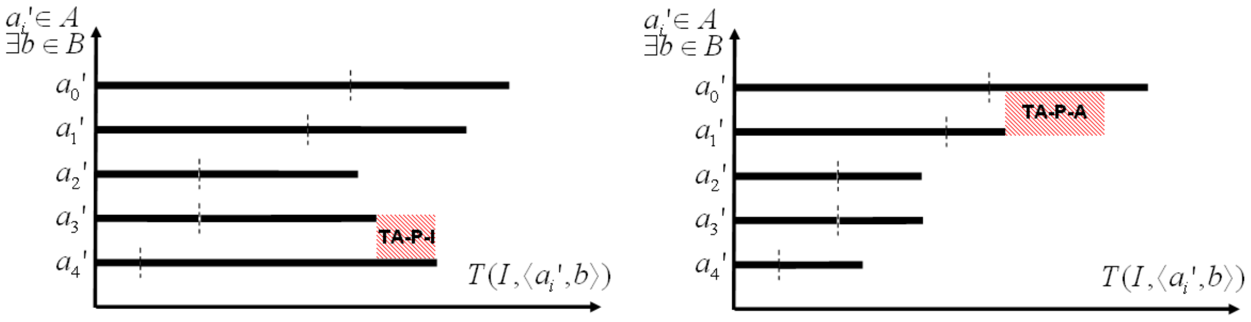


Figure 2.6: Example of *TA-P-I* and *TA-P-A*

Of course it is possible, that a *TA-P-I* and *TA-P-A* timing anomaly for the same $b \in B$ might occur. On the left side in Figure 2.7 this case is shown.
Certainly we are interested in those cases where no timing anomalies occur. When the execution times $T(I, a_i')$ are in descending order, then no timing anomaly appears, if the execution times of $T(I, \langle a_i', b\rangle)$ are also in descending order or adjacent execution times are equal. Also no timing anomaly appear, if we subtract the difference of time from the actual and the successive execution of hardware component hw_A from the actual overall execution time $T(I, \langle a_i', b\rangle)$. For executions with the same latency no timing anomaly can appear.

Composition Techniques for WCET Analysis

In this section some composition techniques are presented. There are helpful for calculating a bound for the WCET analysis. The principle of partitioning the TRDCS state set into two sets with their initial states $a \in A, b \in B$ is needed for the shown techniques. The techniques assume

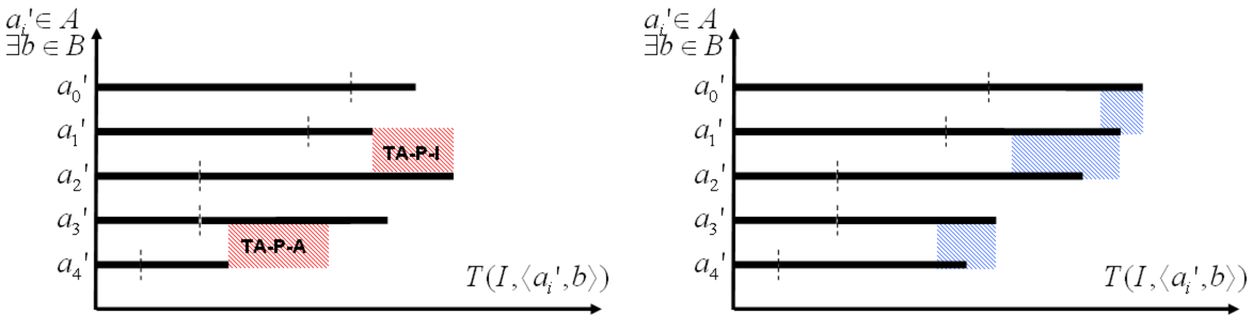


Figure 2.7: Example of *TA-P-I*, *TA-P-A* for the same $b \in B$ and possible areas for no timing anomaly

two TRDCS state sets A and B and calculate the overall execution time for the instruction I and for each pair $\langle a, b \rangle$ from the set $A \times B$.

Delta Composition

To get a WCET bound the following steps must be performed (visually represented at Figure 2.8):

1. $\Delta_{hw_A,max} = \underset{a,a' \in A}{max} |T_{hw_A}(I, a) - T_{hw_A}(I, a')|$:
   The maximum difference of execution times $T_{hw_A}(I, a)$ for hardware component hw_A is calculated.

2. $A_{min} = \{a \in A | \forall a' \in A. T_{hw_A}(I, a') \geq T_{hw_A}(I, a)\}$:
   The set of initial states for hardware component hw_A, where execution times $T_{hw_A}(I, a)$ are minimal is computed.

3. $T(I, \langle a_{hw_A,min}, b \rangle)$:
   For all states, which are within the before calculated set A_{min}, the entire execution times with fixed initial states $a_{hw_A,min}$ are derived.

4. $b_{dc,max} \in B_{A,max}(a_{hw_A,min}) = \{b \in B | \forall b' \in B. T(I, \langle a_{hw_A,min}, b \rangle) \geq T(I, \langle a_{hw_A,min}, b' \rangle)\}$:
   One element b from the set $B_{A,max}(a_{hw_A,min})$, in which are initial states of set B that produce maximum overall execution times for the initial states $a_{hw_A,min}$ of set A, is selected.

5. $T(I, \langle a_{hw_A,min}, b_{dc,max} \rangle) + \Delta_{hw_A,max}$:
   The difference of the first step is added to the overall execution times of step four.

The above presented steps can be formalized with the next definition:

Defintion 2.10.6 $T_{dc}(I) = \underset{a \in A_{min}, b \in B}{max} T(I, \langle a, b \rangle) + \Delta_{hw_A,max}$

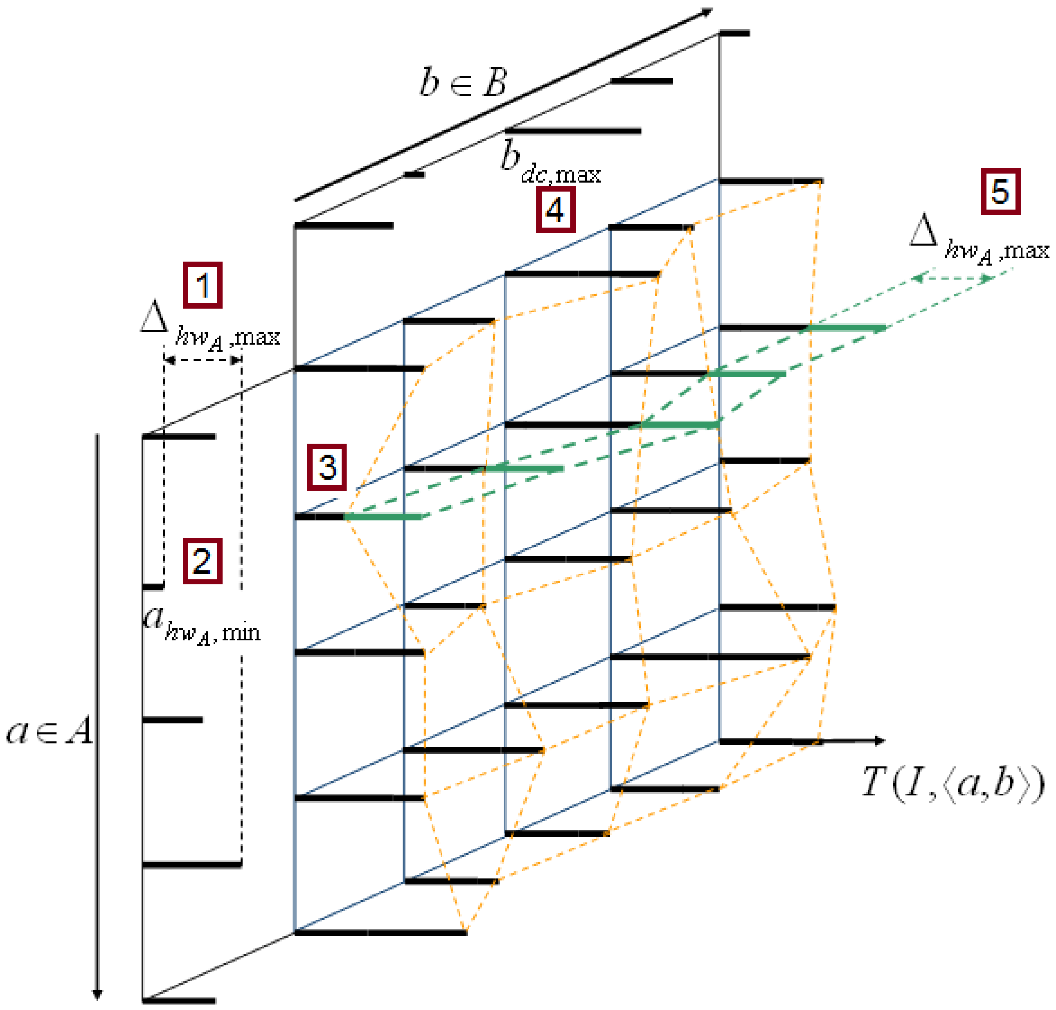


Figure 2.8: Delta Composition principle

In general the delta composition overestimates the WCET and if we analyse the steps defined above, we can detect that the worst case for the calculation of T_{dc} is when all initial states $a \in A_{min}$. But in this case there are no differences in the execution times and hence all states $a \in A_{min}$ are not part of the TRDCS. Thus the worst-case is $|A_{min}| = |A|-1$ and the calculation for the delta composition requires $O((|A| + |A_{min}| * |B|) * |I|)$. This is explained by the search of $\Delta_{hw_A,max}$ (step 1) and A_{min} (step 2) for $|A|$. The term $|A_{min}| * |B|$ can be declared by the computation of the overall execution times $T(I, \langle a_{hw_A,min}, b\rangle)$ and then multiplied by the instruction $|I|$.

Theorem 2.10.7 ***Safeness of Delta Composition***
Assume the definitions from 2.10.4 and an execution of I with a partitioned TRDCS, then the following expression is sufficient and necessary for the safeness of the delta composition tech-

nique:
$\forall a \in A, \forall b \in B, \exists a' \in A_{min}, \exists b' \in B_{A,max}(a')$.
$\Delta_{hw_A}(I, a', a) > 0 \rightarrow \Delta(I, \langle a', b' \rangle, \langle a, b \rangle) \leq \Delta_{hw_A,max}$ *([6] Theorem 5.1 and Proof A.2)*

That means, if the execution time of an instruction I for an arbitrary initial state a is greater than for a' (= $a_{hw_A,min}$), then the difference of the overall execution time between a pair of random initial states $\langle a, b \rangle$ and $\langle a', b' \rangle$ ($b' \in B_{A,max}(a')$) must be less or equal to the maximum difference of the execution times for hardware component hw_A, $\Delta_{hw_A,max}$. One can observe, that this is the opposite of the definition of *TAW-P-A*. With this steps it is possible to calculate a serious WCET bound in case of parallel inversion timing anomalies, because there is a local best case ($a_{hw_A,min}$) and the difference to another initial state a ($\Delta_{hw_A}(I, a_{hw_A,min}, a)$) must be greater or equal than $\Delta(I, \langle a_{hw_A,min}, b' \rangle, \langle a, b \rangle)$ (see Safeness of Delta Composition).
The delta composition is used to analyse TA-P-I timing anomalies.

Max Composition

In the previous section a composition technique was presented to compute a WCET bound for TA-P-I timing anomalies. In this chapter the max composition technique for analysing TA-P-A timing anomalies is shown. This can be done with the following steps (see Figure 2.9):

1. $A_{max} = \{a \in A | \forall a' \in A . T_{hw_A}(I, a') \leq T_{hw_A}(I, a)\}$:
   The set of initial states for hardware component hw_A, where execution times $T_{hw_A}(I, a)$ are maximal is computed.

2. $T(I, \langle a_{hw_A,max}, b \rangle)$:
   For all states, of A_{max}, the entire execution times $T(I, \langle a_{hw_A,max}, b \rangle)$ with fixed initial states $a_{hw_A,max}$ are derived.

3. $b_{mc,max} \in B_{A,max}(a_{hw_A,max}) = \{b \in B | \forall b' \in B . T(I, \langle a_{hw_A,max}, b \rangle) \geq T(I, \langle a_{hw_A,max}, b' \rangle)\}$:
   One element b from the set $B_{A,max}(a_{hw_A,max})$, in which are initial states of set B to produce maximum overall execution times for the initial states $a_{hw_A,max}$ of set A, is selected.

The max composition offers a precise WCET bound and is specified with the following definition.

Defintion 2.10.8 $T_{mc}(I) = \underset{a \in A_{max}, b \in B}{max} T(I, \langle a, b \rangle)$

It seems that the worst-case for the calculation of T_{mc} is that all initial states $a \in A_{max}$. But in this case there are no differences in the timing behaviour (like in the explanation of T_{dc}) and hence it is not part of the TRDCS. Therefore $|A_{max}| = |A| - 1$ is the worst-case for the calculation and it requires $O((|A| + |A_{max}| * |B|) * |I|)$. This computational cost is calculated as stated above by the search of all states from A to get A_{max} and then derive the overall execution times with the states from B multiplied with the number of instructions.

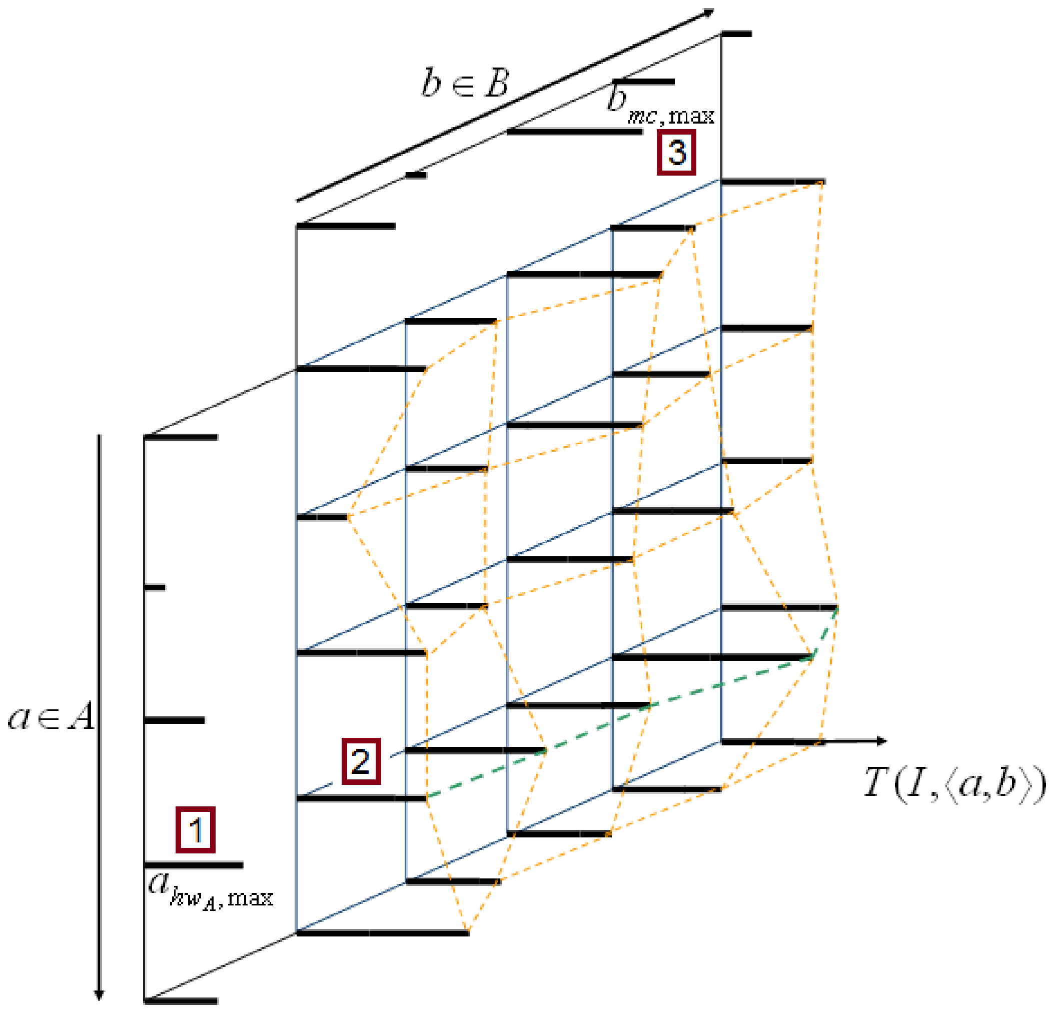


Figure 2.9: Max Composition principle

Theorem 2.10.9 ***Safeness of Max Composition***
Assume the same definitions from def:TAW-P-I and an execution of I with a partitioned TRDCS, then the following expression is necessary and sufficient for the safeness of the max composition technique:
$\forall a \in A, \forall b \in B, \exists a' \in A_{max}, \exists b' \in B_{A,max}(a').$
$\Delta_{hw_A}(I, a, a') > 0 \rightarrow \Delta(I, \langle a, b\rangle, \langle a', b'\rangle) \geq 0$ *([6] Theorem 5.3 and Proof A.2)*

That means, if the execution time of an instruction I for a' (= $a_{hw_A,max}$) is greater than for an arbitrary initial state a, then the difference of the overall execution time between $\langle a', b'\rangle$ ($b' \in B_{A,max}(a')$) and a pair of random initial states $\langle a, b\rangle$ must be greater or equal to zero. One can observe, that this is the opposite of the definition of *TAW-P-I*.
Thus the max composition technique calculates the maximum execution time for the combina-

tion of the sets A and B, the following statement is obvious.
The max composition is used to analyse TA-P-A timing anomalies.

With each of the introduced techniques it is possible to give a WCET bound without analysing the whole state space $A \times B$. If the two parallel anomaly types (TA-P-I and TA-P-A) occur in one execution for different initial states $b_1, b_2 \in B$, then also a WCET bound can be computed, even without searching the whole state space.

Defintion 2.10.10 ***TA-P-E***
Assume an instruction sequence I and a TRDCS initial state space, which is partitioned in two non-empty sets $A \cup B$ with the component latency $T_{hw_A}(I, a)$ of hardware component hw_A and the overall execution time $T(I, \langle a, b\rangle)$, with $a \in A, b \in B$. Then an exclusive parallel timing anomaly is a situation, where the next three properties are fulfilled:
$\exists a_1, a_2, a_3, a_4 \in A, \exists b_1, b_2 \in B.$
$(b_1 \neq b_2) \ \wedge \ (\Delta_{hw_A}(I, a_1, a_2) > 0 \ \wedge \ \Delta(I, \langle a_1, b_1\rangle, \langle a_2, b_1\rangle) < 0) \ \wedge$
$(0 < \Delta_{hw_A}(I, a_3, a_4) < \Delta(I, \langle a_3, b_2\rangle, \langle a_4, b_2\rangle)),$

$\forall a_1, a_2, a_3, a_4 \in A, \forall b \in B.$
$(\Delta_{hw_A}(I, a_1, a_2) > 0 \ \wedge \ \Delta(I, \langle a_1, b\rangle, \langle a_2, b\rangle) < 0 \ \wedge \ \Delta_{hw_A}(I, a_3, a_4) > 0) \rightarrow$
$(\Delta_{hw_A}(I, a_3, a_4) \geq \Delta(I, \langle a_3, b\rangle, \langle a_4, b\rangle)),$

$\forall a_1, a_2, a_3, a_4 \in A, \forall b \in B.$
$(0 < \Delta_{hw_A}(I, a_1, a_2) < \Delta(I, \langle a_1, b\rangle, \langle a_2, b\rangle) \ \wedge \ \Delta_{hw_A}(I, a_3, a_4) > 0) \rightarrow$
$(\Delta_{hw_A}(I, \langle a_3, b\rangle, \langle a_4, b\rangle) \geq 0).$

The first expression shows, that a *TA-P-I* anomaly for state b_1 and a *TA-P-A* anomaly for the state b_2 occur. And if there is a *TA-P-I* for one state b (in that case b_1), then it is excluded, that a *TA-P-A* can emerge for the same state b (second expression). In the third term the opposite case is displayed; if a *TA-P-A* occur for b, then for this state a *TA-P-I* anomaly is precluded.
In this case it is possible to provide a WCET bound, because with the delta composition TA-P-I and with the max composition TA-P-A timing anomalies could be analysed. If both techniques are used concurrently and the maximum of the results is taken (T_{dmc}), then a precise bound can be given. Observe that $T_{dmc}(I)$ overestimates the WCET, resulting from the delta composition.

Defintion 2.10.11 $T_{dmc}(I) = max(T_{dc}(I), T_{mc}(I))$

If both types of timing anomalies can occur for the same state $b \in B$. Then an efficient calculation of a safe WCET bound is not possible and the whole state space $A \times B$ must be analysed (proof is shown in [6]). This is named a coupled parallel timing anomaly (*TA-P-C*).

Defintion 2.10.12 ***TA-P-C***
Assume an instruction sequence I and a TRDCS initial state space, which is partitioned in two

non-empty sets $A \cup B$ with the component latency $T_{hw_A}(I, a)$ of hardware component hw_A and the overall execution time $T(I, \langle a, b\rangle)$, with $a \in A, b \in B$. Then coupled parallel timing anomaly occurs whene:
$\exists a_1, a_2, a_3, a_4 \in A, \exists b \in B.$
$(\Delta_{hw_A}(I, a_1, a_2) > 0 \ \wedge \ \Delta(I, \langle a_1, b\rangle, \langle a_2, b\rangle) < 0) \ \wedge$
$(0 < \Delta_{hw_A}(I, a_3, a_4) < \Delta(I, \langle a_3, b\rangle, \langle a_4, b\rangle))$

Parallel timing anomalies are studied between related hardware components, in this thesis the interaction between different pipeline and cache architectures are analysed in Chapter 7 of this thesis, where the effects of the execution of the cache influence the behaviour and timing of the pipeline timing.

CHAPTER 3

Pipeline Architectures

3.1 Overview

In early years computer processors commonly executed instructions successively usually in the following sense: fetching the instruction from the memory (*IF*), decoding it (*ID*), reading the operands for the instructions from the registers and memory (*RR*), executing the instruction (*EX*) and writing the results to the memory (*MM*) or the registers (*WB*). This strictly sequential execution is not very fast and it was improved in the way, that several instructions can now overlap. That means, the first instruction reads its operands, the second instruction can be decoded and the third instruction can be fetched in parallel. This concept increases the efficiency immense (see Figure 3.1).
The instructions pass through a number of stages. Naturally an instruction must not go through all pipeline stages. Additional there are several ways which stage an instruction can take. A stage can be occupied by only one instruction, and after a stage is entered by an instruction the appropriate step is processed and the instruction leaves the stage. However an instruction can occupy a stage for more than one clock cycle. The flow of instructions is from the upper left to the lower right corner, where the time scale is horizontal (divided in clock cycles) and the stages are displayed vertically.

3.2 Simple Scalar Pipelines

A simple scalar pipeline is the simplest form of the presented pipelines. All stages are performed in an in-order manner. The number of stages in modern processors varies between three and ten, but in general there are five to seven stages. An example of a simple scalar pipeline is shown in Figure 3.2.

An execution of an instruction can be delayed in a pipeline stage by some reasons. In case of a *structural hazard* [16] a pipeline stage is blocked by an instruction and the next instruction is stalled during this time. On the left side of Figure 3.3 two instructions are shown, where the first

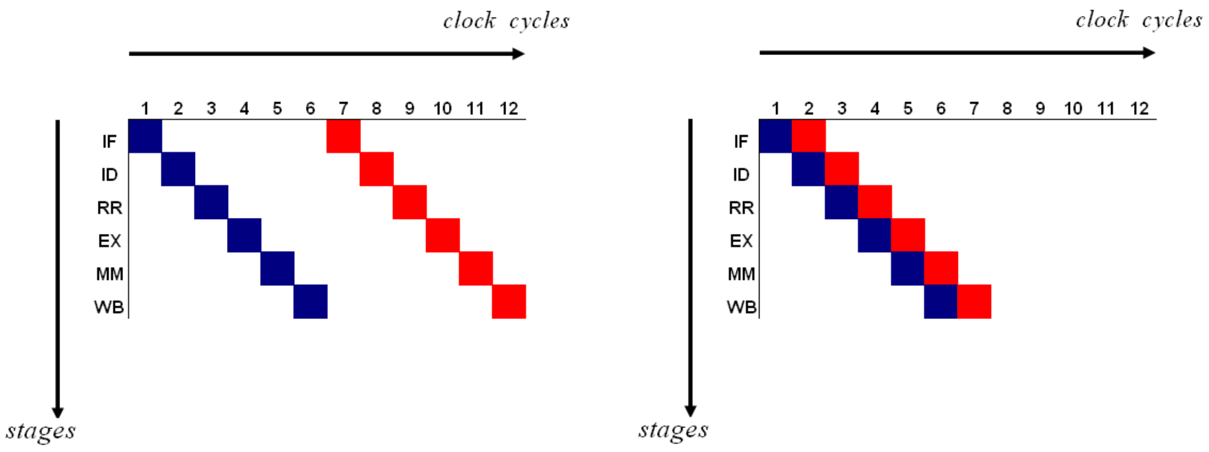


Figure 3.1: Concept of Pipelines

instruction needs two clock cycles for the decode-stage and three clock cycles for reading the registers. The instructions are executed successively. On the right side the instructions overlap and thus the first instruction stalls the second instruction, which must pause in the *IF*- and *ID-stage*.

Another kind of delay can result from data dependencies. More precisely, when a preceding instruction generates some data, which is needed by the subsequent instruction and the generation is not complete until the next instruction is in the processing stage, then this next instruction is stalled. In the Figure 3.4 an example shows an execution of two instructions within a program, whereas on the left side the instructions are executed successively and on the right side the maximal overlap is shown.
The first instruction adds the content from registers r_2 and r_3 and writes the result to register r_1.

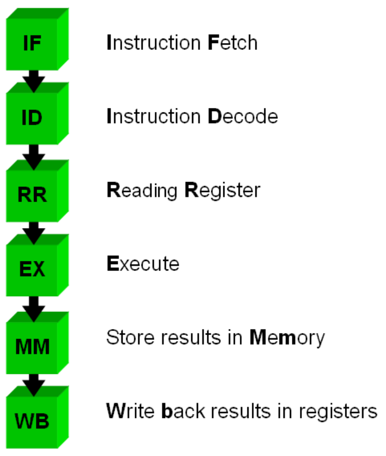


Figure 3.2: Simple Scalar Pipeline

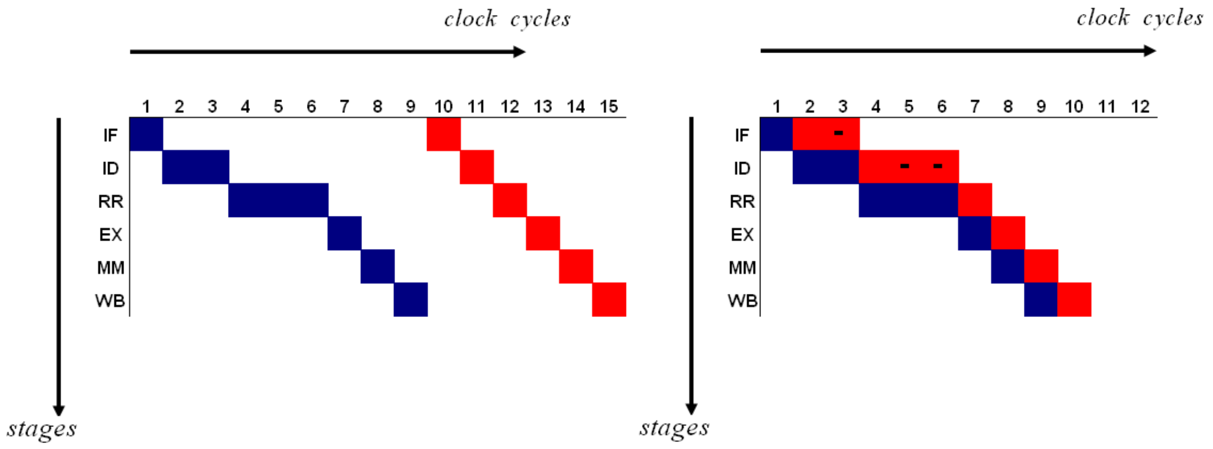


Figure 3.3: Structural Hazard

Thus this result in r_1 is only available until termination in the *WB-stage*. The second instruction needs the content of the register r_1 for a further addition, the second instruction is stalled in the RR-stage and waits until r_1 can be read. Note that the *MM-stage* is not needed, because there is no memory access.

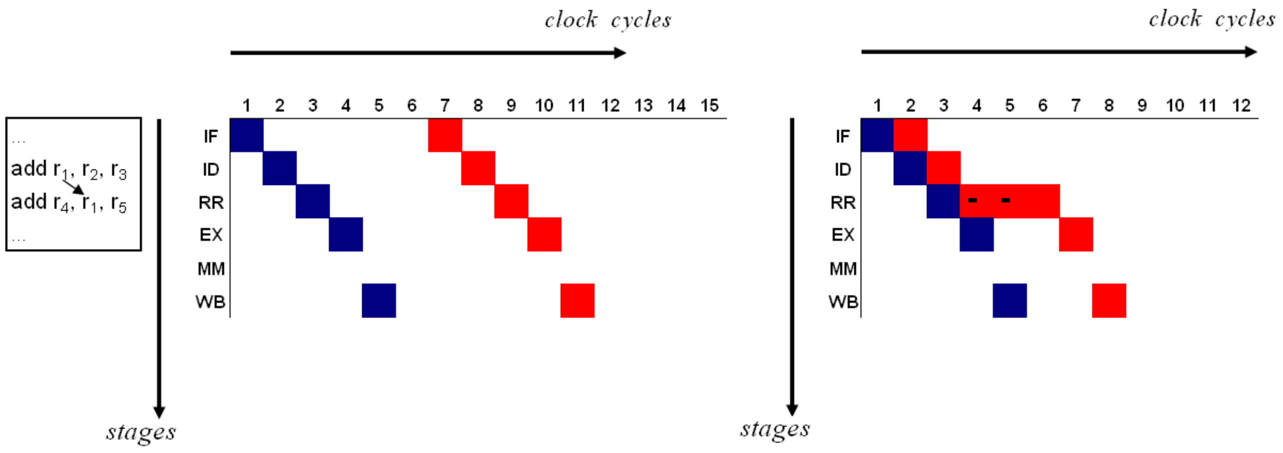


Figure 3.4: Data Hazard

In the following context of this thesis instructions can be combined into blocks. This is illustrated in Figure 3.5. Note that a single instruction could be also a block.

3.3 Scalar Pipelines

Within these pipelines the execution of an instruction can take different paths through the execution units of the processor. The decision which pipeline stages an instruction has to pass

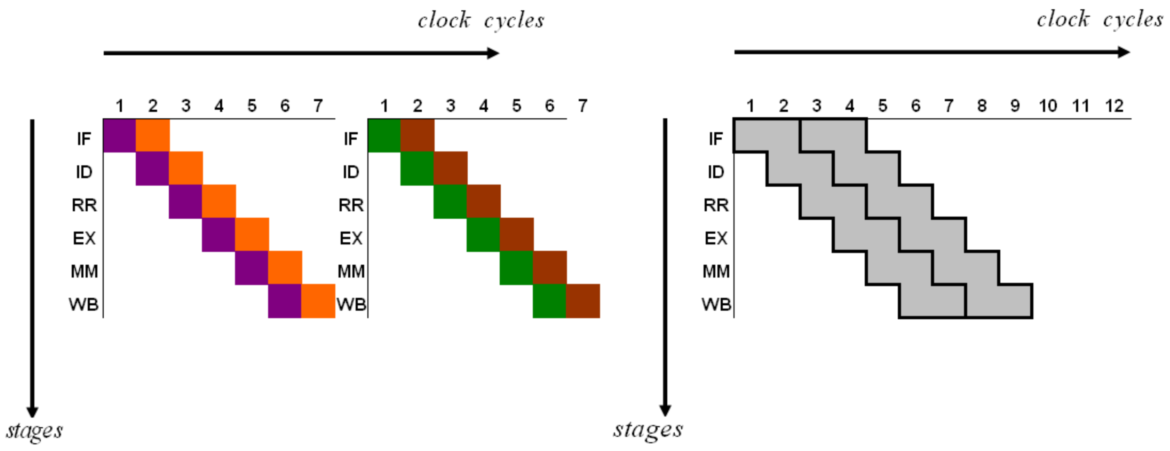


Figure 3.5: Merged Instructions illustrated as Blocks

through depends on the type of the instruction. This concept is used to increase to efficiency of the pipeline. A well known case of application is the segregation of integer and floating-point operations. In Figure 3.6 an example of a scalar pipeline is illustrated. All instructions are executed in in-order.

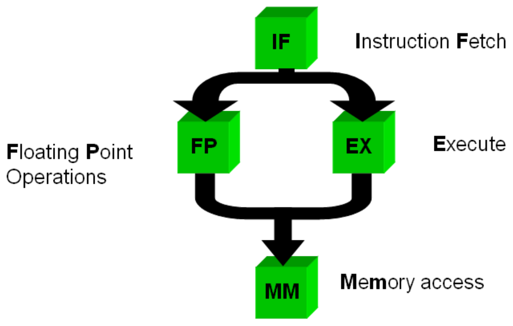


Figure 3.6: Scalar Pipeline

3.4 Superscalar In-Order Pipelines

Another more efficient pipeline is illustrated in Figure 3.7. In the *IF-stage* a number of instructions can be fetched within a clock cycle. The maximum number of instructions which are fetched in one clock cycle corresponds to the number of stages that can be executed in parallel. The *SC-stage* groups the previously fetched instructions and associates them with the subsequent stages, where the instructions are executed in-order. It is essential, that the instructions are scheduled statically, not dynamical, in the *SC-stage*.

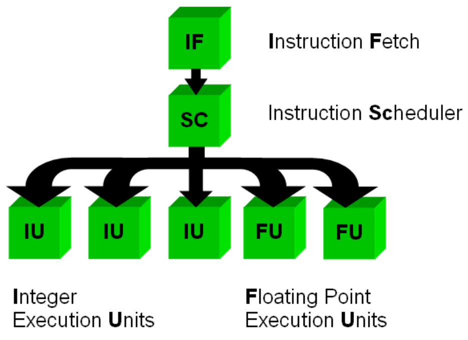


Figure 3.7: Superscalar In-Order Pipeline

3.5 Superscalar Out-Of-Order Pipelines

In a superscalar out-of-order pipeline the selection of the processing instruction is done dynamically. This makes the pipeline more efficient, because the instructions can be scheduled so that delays by data dependencies and resource conflicts are reduced. In parallel the complexity of the pipeline and the function units increase. These pipelines are used for applications, where a high average-case performance is necessary, but the performance in the worst case is very hard to predict.
There exists also a distinction for superscalar out-of-order pipelines; instructions can be fetched and results written back in-order or out-of-order. If we consider in-order issue and in-order completion (IOI/IOC) pipelines, then we can observe that these pipelines are working in the same manner as superscalar in-order pipelines. In-order issue with out-of-order completion (IOI/OOC) pipelines are able to process the in-order fetched instructions and wrote back the results out-of-order.
To fetch and execute the instructions in an out-of-order manner an instruction buffer has to be added. This ensures the selection of instructions in a way, that the available resources are optimally occupied. Also this must procure, that the program behaviour and data dependencies are not affected. Thus the superscalar pipeline is perfectly utilised. The concept of processing instructions in OOI/OOC pipelines is shown in the a resource allocation Table 3.1.

Instruction	Unit	Cycles	Data Dep.
I_0	IU	1	
I_1	IU	2	
I_2	IU	3	before I_3
I_3	FU	1	after I_2
I_4	FU	2	before I_5
I_5	FU	2	after I_4
I_6	FU	1	
I_7	IU	2	
I_8	IU	2	
I_9	FU	2	

Clock Cycle	Dispatcher		Buffer				IU	IU	FU	FU	FU	WB	WB	WB
1	I_0	I_1												
2	I_2	I_3	I_0	I_1			I_0	I_1						
3	I_4	I_5	I_2	I_3			I_2	I_1				I_0		
4	I_6	I_7	I_3	I_4	I_5		I_2		I_4			I_1		
5	I_8	I_9	I_3	I_5	I_6	I_7	I_2	I_7	I_4	I_6				
6			I_3	I_5	I_8	I_9	I_8	I_7	I_3	I_5	I_9	I_2	I_4	I_6
7							I_8			I_5	I_9	I_7	I_3	
8												I_5	I_8	I_9

Table 3.1: Superscalar Out-Of-Order Pipeline: Resource Allocation Table

CHAPTER 4

Cache Architectures

4.1 Overview

Because storage elements have a large access times, buffer elements (caches) are placed between the main storage elements and the CPU. Caches contain copies of required data, therewith it is possible to provide that data faster for future requests. If some data is needed, then an enquiry on the caches is raised. If the cache contains the requested data, then this situation is called a *cache hit* and the data is provided. If the requested data is not part of the cache storage, this situation is called a *cache miss*. The required data is fetched from the main storage element (which takes some time) and it is saved for future requests. The *cache replacement strategy* decides which cache block is being replaced if new data must be stored in the cache. In the next sections several strategies are presented and discussed. With the may and must analysis [10] [7] the knowledge of the contained cache blocks can be done. Some cache replacement policies, which are responsible for the appearance of timing anomalies, are explored in [11] by Reineke et al. For serious WCET calculations we need a prediction if a cache hit or cache miss causes timing anomalies (and which timing anomalies) or not. This issue and an analysis of different replacement strategies are explored in detail in the next chapters.

4.2 Structure of Caches

Caches are segmented into rows and each row consists of a cache line, which includes the required data memory blocks with b bytes. A *valid bit* indicates if a row contains valid data. The address is split into an index (with length of $\lceil log_2(numberofcacherows)\rceil$), which indicates the row of the main memory, an offset (with length of $\lceil log_2(numberofdatablocks)\rceil$), which shows the especially block in the cache line and an tag (with length of $totaladdresslength - indexlength - offsetlength$) and comprises the MSBs of the main memory address.
The length of a data block with b bytes times the total number of cache lines l indicates the cache size s.

4.3 Associativity

The associativity (number of cache lines within one set) k can be computed by the number of cache lines l divided by the number of sets. In *fully-associative* caches ($k = l$) there is only one set with all l cache lines in it [15]. Each data block can contain each data from the main memory. In case of a cache request it is necessary to check all tags; this makes only sense if small caches are used.
In case of *direct-mapped* caches ($k = 1$) every cache line represents a set. Thereby every cache block is directly mapped to data of the main memory and only this line has to be checked in case of a request. Thus this is the best choice if large cache sizes are available.
If $k > 1$, then a replacement strategy selects the cache line in the set, which should be updated. This should be done with the focus on decreasing the number of cache misses.

4.4 Cache Replacement Policies

FIFO

With the first-in first-out replacement strategy the cache uses only one pointer, which increases its value sequentially modulo the sum of cache pages to select a cache line for updating. The entries are saved in the order they were loaded. Thus this strategy is relative simple and easy to understand. If some data is added to the last cache line, then afterwards the pointer has the value zero and points to the first cache line. This strategy has a constant access time regardless of the cache size.

Least Recently Used

This strategy updates the least recently used cache line, if the cache is full and a cache miss appears. It requires a tracking of the *age* of cache lines. This can be done by several implementations. To abstract the cache as a linked list is one realization. A referenced page is put to the head of the list. If a new element has to be saved, this element is added on the new head of the list and the oldest page (tail of the list) is ejected.
In another method the *ages* of cache lines are tracked. In a k-way associative LRU cache the most recently used line has the age 0 and the least recently used the age $k - 1$. In a cache miss situation the line with the age $k - 1$ is ejected and then the line, which contains the new data, gets the age 0 and all other increase their ages by 1. If a line of age a is accessed, then the age of the this page is set to 0 and the lines of age 0 to $a - 1$ increase their ages by 1. An example is shown in Table 4.1 for a 4-way LRU cache. Note that least recently used caches with greater than 4-way associativity are uncommon.

Pseudo Least Recently Used

The pseudo least recently used cache replacement strategy is using a tree-based approach for the replacing within a cache set. The average-case performance is comparable with LRU, nevertheless the worst-case performance is worse. It needs a less complex update logic. The constructed tree in 4-way PLRU caches consists of inner nodes, which contain state bits t_0 to t_{k-2} (for a k-way PLRU cache). The content of the leafs L_0 to L_k of the constructed tree include the data. The content of the inner nodes are state bits t_0 to t_{k-2} (for a k-way PLRU cache), which point to the subtrees of them (in our case 0 to the left and 1 to the right subtree). Thereby it is possible to build a path from the root to every leaf. In case of a cache miss the content of the leaf, with the state bits of the tree pointing to it, is updated with the new one and this state bits are flipped. In case of a cache hit, the state bits to the accessed content are changed in that way that they are pointing away from this most recently used content. In practice 8-way PLRU is common (same update logic, but 7 state bits) and a 2-way PRLU equals a 2-way LRU. In Figure 4.1 the tree-based approach for the PLRU strategy for a 4-way PRLU is shown. An update example is shown in Table 4.2.

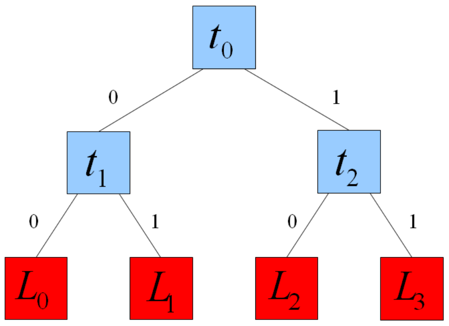


Figure 4.1: 4-way PLRU Tree

Accesses	hit/miss	Ages			
		0	1	2	3
M_0	miss	M_0			
M_1	miss	M_1	M_0		
M_2	miss	M_2	M_1	M_0	
M_3	miss	M_3	M_2	M_1	M_0
M_2	hit	M_2	M_3	M_1	M_0
M_4	miss	M_4	M_2	M_3	M_1
M_1	hit	M_1	M_4	M_2	M_3
M_2	hit	M_2	M_1	M_4	M_3
M_5	miss	M_5	M_2	M_1	M_4
M_6	miss	M_6	M_5	M_2	M_1
M_7	miss	M_7	M_6	M_5	M_2
M_2	hit	M_2	M_7	M_6	M_5

Table 4.1: 4-way LRU cache: Filling with Age Update

Accesses	**hit/miss**	**State Bits**			**Content**			
		t_0	t_1	t_2	L_0	L_1	L_2	L_3
		0	0	0				
M_0	miss	1	1	0	M_0			
M_1	miss	0	1	1	M_0		M_1	
M_2	miss	1	0	1	M_0	M_2	M_1	
M_3	miss	0	0	0	M_0	M_2	M_1	M_3
M_2	hit	1	0	0	M_0	M_2	M_1	M_3
M_4	miss	0	0	1	M_0	M_2	M_4	M_3
M_1	miss	1	1	1	M_1	M_2	M_4	M_3
M_2	hit	1	0	1	M_1	M_2	M_4	M_3
M_5	miss	0	1	0	M_1	M_2	M_4	M_5
M_6	miss	1	0	0	M_1	M_6	M_4	M_5
M_7	miss	0	0	1	M_1	M_6	M_7	M_5
M_2	miss	1	1	1	M_2	M_6	M_7	M_5

Table 4.2: 4-way PLRU cache: Filling with Tree Bits Update

CHAPTER 5

Timing Anomalies in Pipelines

5.1 Overview

In Chapter 3 some pipeline architectures have been presented. In this chapter the introduced pipelines are discussed in detail with regard to timing anomalies and the types of timing anomalies. In the next sections the complexity of the pipeline models is extended.
If a real program is assumed and this is considered as a sequence of single instructions, then the mutual influence of those instructions can be analysed. The argumentation, theorems and definitions are mainly taken from [17]. For simplicity, pipelines are considered as isolated parts of a processor regarding the analysis of series timing anomalies. As in the preceding chapters the result of an instruction is available until this instruction is performed in the associated pipeline stage. An instruction sequence can every time be analysed in a way, that the observation-window begins with the time-varying instruction I_0.

5.2 Timing Anomalies in Simple Scalar Pipelines

If we abstract the pipeline model from Section 3.2 to a model, where the *ID*-, *RR*- and *EX-stages* are summarized to an explicit *function unit (=FU)* and the *MM*- and *WB-stage* are combined to a single *WB-stage*, then we get a model which consists of three stages; namely the stage for fetching the instructions (*IF*), an explicit function unit *(*FU) and a write back stage *(*WB). This is possible (within the scope of timing anomaly analysis), because all stages are in-order units and by the strict processing-sequence this abstraction is possible (see Figure 5.1).

We know from the definition of timing anomalies that for an occurrence a dynamic behaviour is necessary. In the simple scalar pipeline model all instructions are fetched in-order in the *IF-stage* and after fetching they run in the *FU-stage*, where the processing of an instruction can take longer than one clock cycle. If the *FU-stage* is occupied, then no other instruction is fetched. After the execution in the function unit stage, the results are written back in the *WB-stage* in-order. That means for every instruction of an instruction sequence:

1. There is at most one instruction to be fetched within one clock cycle.
2. The duration of an instruction in the *FU-stage* can be greater than one cycle, but there is only one instruction inside it.
3. The results are written back in-order.

Is it possible that structural or data hazards may generate timing anomalies in this pipeline model? This is not feasible, because an instruction is processed identically and has the same duration in every execution.
We know from [17] that even in in-order resources timing anomalies appear. But in this model no partially overlapping resources exist and we know that timing anomalies only appear if a pipeline model is composed of in-order resources and at least two of them are partially overlapping.

Theorem 5.2.1 *In simple scalar pipelines no timing anomalies can occur.*

Proof The proof is trivial. All stages proceed in an in-order manner and there are no partially overlapping units. If there is a change in the execution time of the first instruction I_0, then this leads to the same variation for the whole execution $\Delta(I_0, s_0, s_1) = \Delta(I, s_0, s_1)$. Thus no timing anomaly can take place.$\square$

5.3 Timing Anomalies in Scalar Pipelines

In Section 3.3 scalar pipelines were introduced. For an accurate analysis of timing anomalies in scalar pipelines a small modification is necessary. Instructions are processed in different function units after the fetching stage, depending on the instruction type. The integer and floating-point operation units are now called function units (FU_0 and FU_1). In the *WB-stage* the results are written back to their destinations in-order. A model of these types of pipelines is shown in Figure 5.2

Scalar Pipelines without Overlapping Function Units

Assume that the existing function units FU_0 to FU_n (in this case $n = 1$) can only process instructions c, which are in the instruction sets $c \in IS_i$ (with $0 \leq i \leq n$). In scalar pipelines

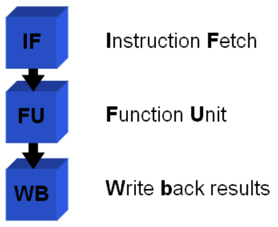


Figure 5.1: Abstraction of Simple Scalar Pipeline

without overlapping function units the instruction sets are completely disjunct. That means that $IS_j \cap IS_k = \emptyset$, with $0 \leq j, k \leq n$.
In this case the *IF-stage* fetches the instructions in-order and distributes the instructions to the appropriate function unit. Timing anomalies as a result of data dependencies are not possible, because the fetching process is stalled until the dependencies are solved. In addition the *write back-stage* guarantees that the results are delivered in program order; more precisely, if a subsequently fetched instruction is faster in a particular function unit than the previously fetched instruction, then this (subsequently fetched) instruction is stalled until the result of the other instruction is committed. That means for every stage:

1. There is at most one instruction to be fetched in an in-order manner within one clock cycle.
2. If a subsequent instruction is faster than the actual executed instruction in a function unit, then the subsequent instruction is stalled until the actual executed instruction is finished with its execution.
3. The results are written back in-order.

Theorem 5.3.1 *In scalar pipelines without overlapping instructions sets of the function units no timing anomalies can occur.*

Proof The proof is similar to the proof of the simple scalar pipeline. All stages proceed in an in-order manner and there are no partially overlapping instruction sets of the function units. If there is a local change Δ_l in the execution time (caused by a longer duration in the *FU-stage* or a structural- or data-hazard), then this leads to the same variation Δ_l in the global timing, because the *WB-stage* ensures that the results are written back in-order. If a local change Δ_l happens in a function unit and the execution of a subsequent instruction in the other function unit is faster than the actual, then the *WB-stage* stalls the faster, subsequent instruction. The only difference to simple scalar pipelines is that the execution time decreases due to the fact that two operations can be processed simultaneously and this is done in-order.$\square$

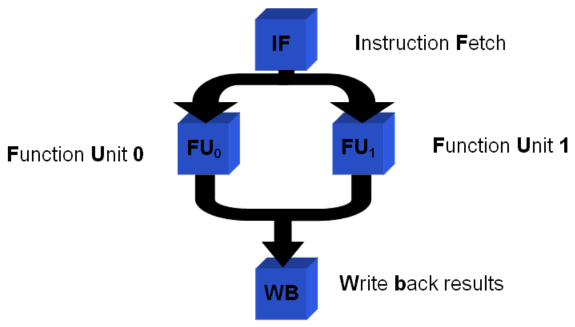


Figure 5.2: Abstraction of Scalar Pipeline

Scalar Pipelines with Overlapping Function Units

For the next explanations it is useful to introduce some definitions and abstractions. Figure 3.2 provides a diagram for the visualization of the workload of a pipeline. From now on, the visualization is lightly modified so that more complex processes are better understandable. The *IF-stage* and the *WB-stage* are excluded and only the stages that are necessary for the analysis are shown. Furthermore the numbering of clock cycles is changed. We start counting clock cycles when a function unit is occupied the first time. The preceding operations during the fetching phases are not numbered, see Figure 5.3.

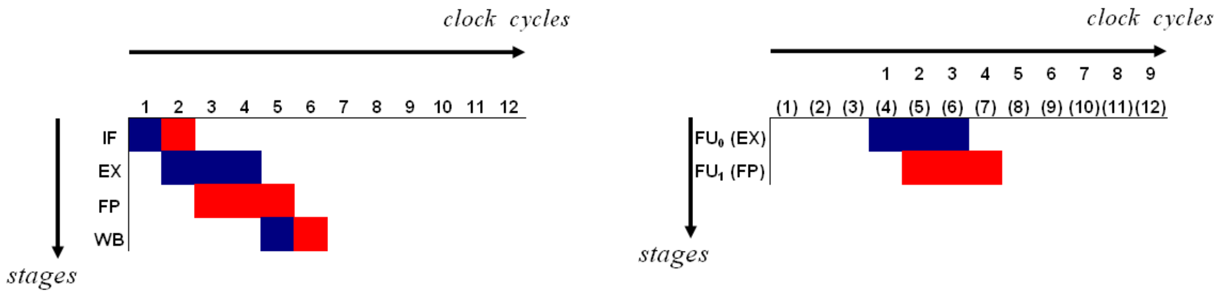


Figure 5.3: Better pipeline visualization of complex pipeline processes

Scalar pipelines could have several function units that can process the same instruction. Thus the instruction sets of these units are overlapping. That means that for n function units the instruction sets intersect each other $IS_j \cap IS_k \neq \emptyset$ (with $0 \leq j, k \leq n$) and in this thesis we assume that the instruction set IS_0 of FU_0 can execute more instructions than the instruction set IS_1 of FU_1, $|IS_0| \geq |IS_1|$. This is the minimal form of overlapping instruction sets. To simplify the analysis of these pipelines some restrictions are defined:

1. Only two function units are considered.
2. One instruction is fetched every clock cycle and in detail the instruction I_l is dispatched in the clock cycle $l + 1$.
3. The execution time of exactly one instruction (I_0) is changed.
4. If both function units are not occupied and an instruction could be dispatched to either unit, then function unit FU_0 has a higher priority than FU_1.

This minimal form of overlapping function units is called 'Minimal Overlapping FUs' in the course of this thesis.

In the next figures the instructions are listed on the left side with their duration and the function unit, which is able to perform the instruction. On the right side an example of the pipeline for an instruction sequence $I = I_0 \circ I_1 \circ I_2 \circ I_3$ is shown. With the specific durations of the instructions and the execution options, Figure 5.4 shows an example, where no timing anomaly appears. Also assume two different executions E, with initial state s_0, and F, with initial state s_1.

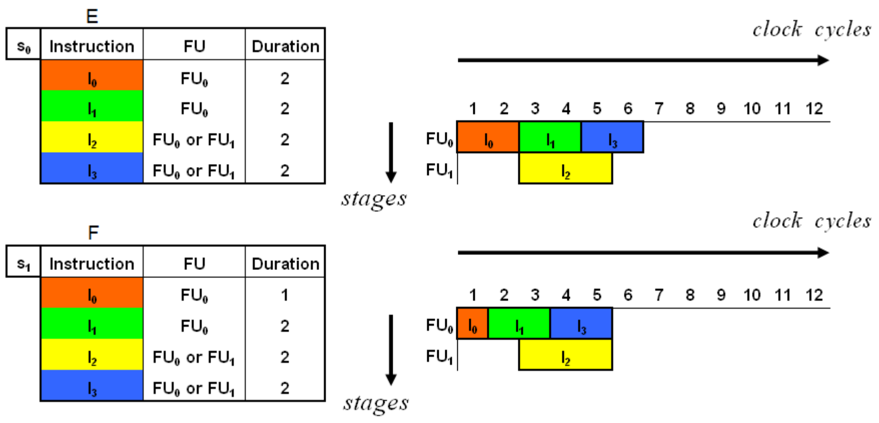
In the diagram in the upper half of Figure 5.4 instruction I_0 is first fetched and the function unit FU_0 is not busy at this time in E. Thus the instruction I_0 occupies FU_0 for its duration, namely two clock cycles. Instruction I_1 is dispatched before clock cycle 2. It stalls, because FU_0 is occupied by I_0 at this point of time. Hence I_1 using FU_0 at clock cycle 3 for further two clock cycles. At the same time I_2 is fetched and occupies function unit FU_1 until clock cycle 6, because FU_0 is reserved. The instruction I_3 is stalled until clock cycle 5 and is processed at FU_0 afterwards.
In the lower diagram of Figure 5.4 the duration of I_0 is decreased by one clock cycle and as a result the entire execution is decreased by one clock cycle also in F. Thus no timing anomaly appears ($\Delta(I_0, s_1, s_0) > 0 \wedge\ \Delta(I, s_1, s_0) = \Delta(I_0, s_1, s_0)$).

Scalar Pipelines with Overlapping Function Units Execution 4 Instructions

In Figure 5.5 a TA-S-I timing anomaly is displayed, where in the above diagram the instruction I_1 is forced to be executed at FU_0 due to the restriction that the preferred function unit is FU_0 and thereby the global execution in E is faster than in F, whereas the instruction I_0 has an increased execution time in E against in F. This leads to a series inversion timing anomaly ($\Delta(I_0, s_1, s_0) > 0 \wedge\ \Delta(I, s_1, s_0) < 0$).

Assume two different executions E, with initial state s_0, and F, with initial state s_1. A decreased execution time of I_0 in E in Figure 5.6 leads to a change of the processing function units for instruction I_1 and I_2 in F. Thus the faster instruction I_1 is processed in function unit FU_0 in place of I_2, which has a longer duration. In this way a series amplification timing anomaly is constructed, with $0 < \Delta(I_0, s_1, s_0) < \Delta(I, s_1, s_0)$.

How are these timing anomalies created? If the execution time of an instruction is changed and the succeeding instructions can be assigned to one function unit out of a set of function units, then a replacement of the executing function unit of instructions could be reached and thereby



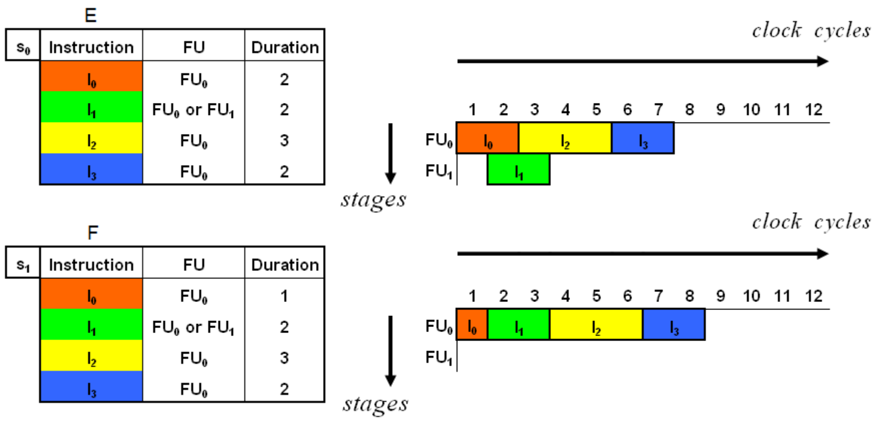
E

s_0	Instruction	FU	Duration
	I_0	FU_0	2
	I_1	FU_0	2
	I_2	FU_0 or FU_1	2
	I_3	FU_0 or FU_1	2

F

s_1	Instruction	FU	Duration
	I_0	FU_0	1
	I_1	FU_0	2
	I_2	FU_0 or FU_1	2
	I_3	FU_0 or FU_1	2

Figure 5.4: Scalar pipeline with no timing anomaly



E

s_0	Instruction	FU	Duration
	I_0	FU_0	2
	I_1	FU_0 or FU_1	2
	I_2	FU_0	3
	I_3	FU_0	2

F

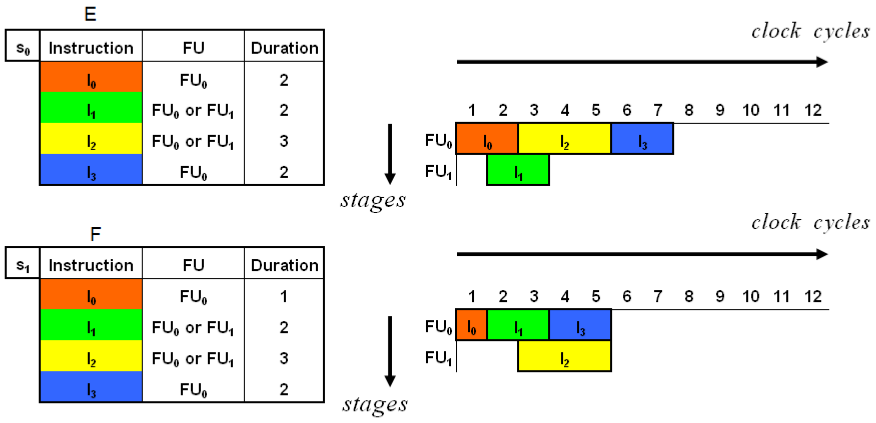
s_1	Instruction	FU	Duration
	I_0	FU_0	1
	I_1	FU_0 or FU_1	2
	I_2	FU_0	3
	I_3	FU_0	2

Figure 5.5: Scalar pipeline with TA-S-I and 4 instructions

timing anomalies occur. With the restriction from above, that a specific function unit is preferred and an instruction is fetched at this point, this unique situation is created. We can determine for every stage:

1. There is at most one instruction to be fetched in an in-order manner within one clock cycle.
2. Instructions could be executed at several function units.
3. The results are written back in-order.

Theorem 5.3.2 *In scalar pipelines with Minimal Overlapping FUs and executing 4 instructions TA-S-I and TA-S-A timing anomalies can occur.*



E

s_0	Instruction	FU	Duration
	I_0	FU_0	2
	I_1	FU_0 or FU_1	2
	I_2	FU_0 or FU_1	3
	I_3	FU_0	2

F

s_1	Instruction	FU	Duration
	I_0	FU_0	1
	I_1	FU_0 or FU_1	2
	I_2	FU_0 or FU_1	3
	I_3	FU_0	2

Figure 5.6: Scalar pipeline with TA-S-A and 4 instructions

Proof Assume two different executions E, with initial state s_0, and F, with initial state s_1. If the duration of I_0 is decreased/increased (against an reference execution) and it is processed in the function unit FU_i in E, then it is possible, that the succeeding instruction (which can be realized at more than one function unit and is processed in function unit FU_j/FU_i) is forced to be executed at FU_i/FU_j in F, this could result in a longer/shorter duration than in E (TA-S-I). If at least two succeeding instructions I_l, I_m can be executed at more than one function unit and one of these instructions I_l is processed at the same function unit in E, as instruction I_0 and $T(I_l, s) > T(I_m, s)/T(I_l, s) < T(I_m, s)$ and the instructions I_l, I_m change the function units in F, then this situation could lead to a shorter/longer duration of the overall execution (TA-S-A).□

Scalar Pipelines with Overlapping Function Units and Executing 3 Instructions

We observed one only has to construct a TA-S-I anomaly, where an instruction is 'squeezed' into the timing relevant function unit (in case if the execution time of the relevant instruction I_0 is decreased and I_0 is processed at the timing relevant function unit, seen in Figure 5.5).

In case of 3 instructions this behaviour is also simple to construct. Merely the constant factor of instruction I_3 is missing, which has no effect on the occurrence of TA-S-I anomaly, see Figure 5.7.

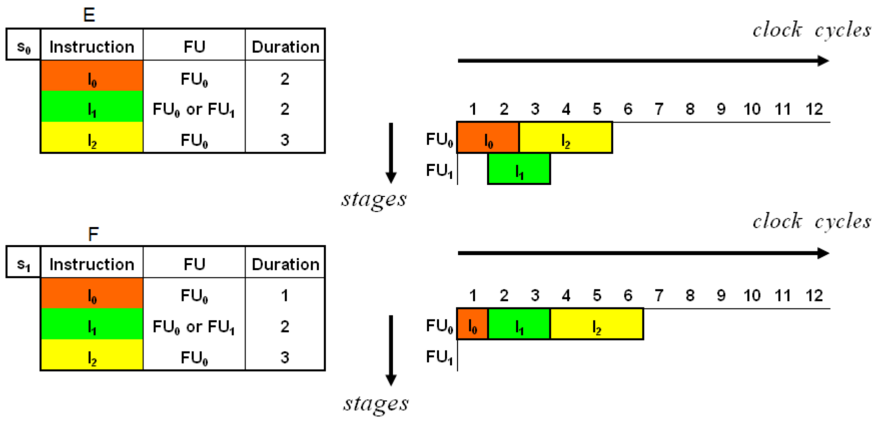


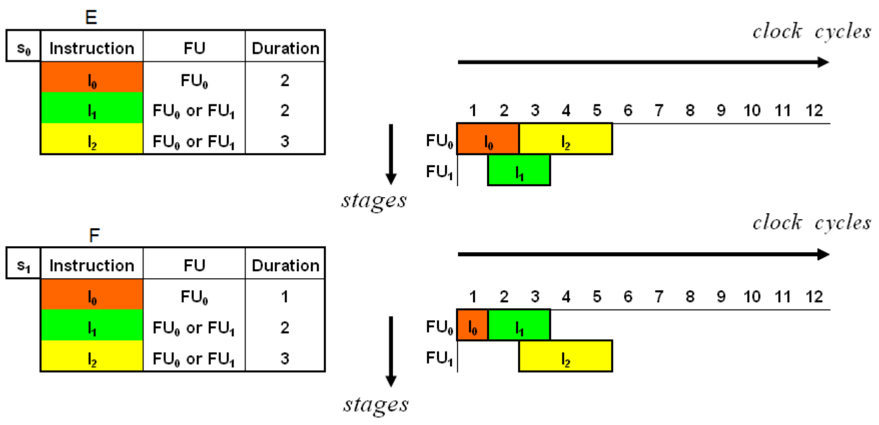
Figure 5.7: Scalar pipeline with TA-S-I and 3 instructions

Theorem 5.3.3 *In scalar pipelines with Minimal Overlapping FUs and executing 3 instructions TA-S-I anomalies can occur.*

Proof Assume two different executions E, with initial state s_0, and F, with initial state s_1. If the duration time of I_0 is decreased/increased (against an reference execution) and it is processed in the function unit FU_i in E, then it is possible, that the succeeding instruction (which can be realized at more than one function unit and is processed in another function unit FU_j / the same function unit FU_i) is forced to be executed at FU_i/FU_j in F, this could result in a longer/shorter duration time than in E (TA-S-I).□

Now the occurrence of TA-S-A anomalies in scalar pipelines with *Minimal Overlapping FUs* and 3 instructions are analysed. We have learned from the above subsection, that TA-S-A anomalies are easiest to construct, if the execution time of the reference instruction I_0 decreases/increases and two instructions I_l, I_m with $T(I_l, s) > T(I_m, s)$/$T(I_l, s) < T(I_m, s)$ and I_l is executed at the timing relevant function unit are interchanged on the function units and the instruction I_m is afterwards executed at that function unit which is relevant for the overall execution time. For simplify the explanations let us concentrate on the case, where I_0 has a decreased execution time.

To construct such a timing anomaly we have to interchange two instructions on their function units. To force one instruction (in our case I_1) to be executed at the timing relevant function unit (in this case FU_0), it is advisable to decrease the execution time of I_0 to 1 clock cycle. The length for instruction I_2 is 3 clock cycles. The instructions I_1 and I_2 can be executed at both function units. Now this case is illustrated in Figure 5.8.



s_0	Instruction	FU	Duration
	I_0	FU_0	2
	I_1	FU_0 or FU_1	2
	I_2	FU_0 or FU_1	3

s_1	Instruction	FU	Duration
	I_0	FU_0	1
	I_1	FU_0 or FU_1	2
	I_2	FU_0 or FU_1	3

Figure 5.8: Scalar pipeline trying to construct a TA-S-A with 3 instructions

This is the best case to construct a TA-S-A anomaly for that kind of pipeline and 3 instructions. Thus I_1 and I_2 are starting from the same point in E and F and hence no TA-S-A anomaly can occur. Note that a longer duration time for I_0 in E causes that a more decreased entire execution time must be realized in F.

Theorem 5.3.4 *In scalar pipelines with Minimal Overlapping FUs and executing 3 instructions TA-S-A anomalies can not occur.*

Proof Informal proof on the facts collected before:
Assume two different executions E, with initial state s_0, and F, with initial state s_1. Best preconditions for I_0 to create a TA-S-A:

- I_0 has to be executed at FU_0.
- Execution time of I_0 has to be 1 clock cycle in E, to force that I_1 is performed at FU_0.

- Execution time of I_0 has to be 2 clock cycles in E, to minimize the requirements for a TA-S-A.

To create an amplification series timing anomaly, I_1 needs a duration time of 2 clock cycles, to obtain a realization of I_2 at FU_0 in E and at FU_1 in F.
The best circumstances are created for an occurrence of TA-S-A, however the instruction I_1 (resp. I_2) and of course I_0 starts from the same point in both executions. Thus no variation in the overall execution time can be achieved and the criteria for TA-S-A anomalies can not be satisfied.□

Scalar Pipelines with Overlapping Function Units and Executing 2 Instructions

Is it possible, that in scalar pipelines with overlapping function units (with the requirements, which are defined above) timing anomalies can occur, if only 2 instructions are executed? Remember that in this kind of pipeline timing anomalies appear, if at least one instruction changes the function unit, in which it is executed. To force an execution of instruction I_1 to be realized at the function unit FU_1 in F instead at function unit FU_0 in E, the duration time of instruction I_0 has to decrease to 1 clock cycle, so that I_1, which is fetched after the first clock cycle, can change its function unit, Figure 5.9. One can observe, that no timing anomaly with two instructions at this sort of pipeline can occur.

Theorem 5.3.5 *In scalar pipelines with Minimal Overlapping FUs and executing 2 instructions TA-S-A and TA-S-I anomalies can not occur.*

Proof Informal proof on the facts collected before:
Assume two different executions E, with initial state s_0, and F, with initial state s_1. Best preconditions for I_0:

- I_0 has to be executed at FU_0.
- Execution time of I_0 has to be 1 clock cycle in s_1, to force that I_1 is performed at FU_0.

Instruction I_1 is starting from the same clock cycle in both executions E and F.
If a TA-S-A anomaly should created and the execution time of I_0 in F is decreased, then the overall duration in E has to decrease more than in F in order to obtain a TA-S-A anomaly. This is not possible, because I_1 starts from the same point in both executions.
If a TA-S-I anomaly should created and the execution time of I_0 in F is decreased, then the overall duration in E has to increase in order to obtain a TA-S-I anomaly. This is not possible, because I_1 starts from the same point in both executions and $T(I_0, s_1)$ has to be one clock cycle to achieve the change of I_1 to FU_0 in F.□

5.4 Timing Anomalies in Superscalar Pipelines

The main difference between scalar pipelines and superscalar pipelines is that in superscalar pipelines more than one instruction can be fetched within one clock cycle. Hence this results in a more packed and faster execution along the pipeline stages. Over the course of the following sections the factors that encourage timing anomalies becoming exacerbate.

Superscalar Pipelines with Dual-Issue and Dual-Complete

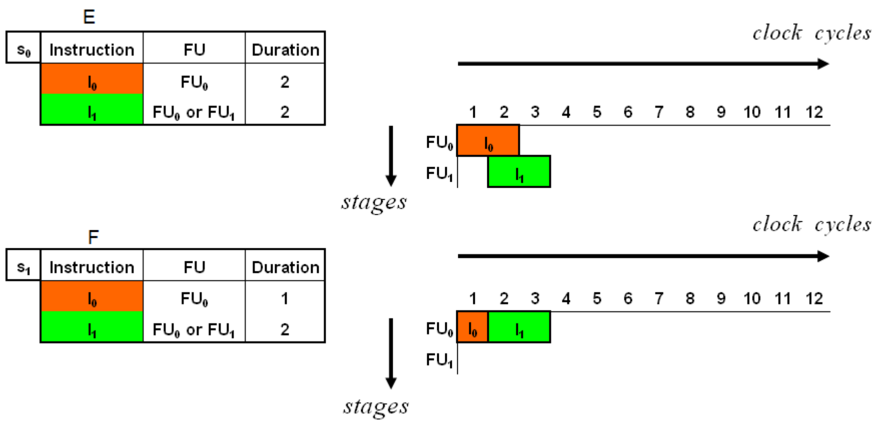
Based on scalar pipelines at most two instructions can be fetched within one clock cycle and at most two instructions can complete within one clock cycle. All stages have an in-order execution behaviour. That means for every stage of this type of superscalar pipeline:

1. There are at most two instructions to be fetched in an in-order manner within one clock cycle.

2. If a subsequent instruction is faster than the actual executed instruction in a function unit, then the subsequent instruction is stalled until the actual executed instruction is finished with its execution.

3. There are at most two results, which are written back in-order within one clock cycle. More detailed, within an instruction stream $I = I_0 \circ I_1 \circ I_2 \circ \ldots$ an instruction I_k (with $k \in \mathbb{N}$ must enter the *WB-stage* before I_{k+1}.

Theorem 5.4.1 *In superscalar pipelines without Minimal Overlapping FUs and with the property that at most two instructions can be fetched and at most two results are written back within one clock cycle no timing anomalies can occur.*



E

s_0	Instruction	FU	Duration
	I_0	FU_0	2
	I_1	FU_0 or FU_1	2

F

s_1	Instruction	FU	Duration
	I_0	FU_0	1
	I_1	FU_0 or FU_1	2

Figure 5.9: Scalar pipeline trying to construct TA-S-I and TA-S-A timing anomalies with 2 instructions

Proof The proof is similar to the proof for scalar pipelines without overlapping function units. Though that maximal two instructions can be fetched, this can not result in different execution times, because this is done in-order and there is no dynamical resource allocation. Additionally the fact that the results are written back in-order (third point of the description above) achieves that no timing anomaly can occur.□

Superscalar Pipelines with Overlapping Function Units and 5 Instructions

In this subsection the pipeline is enhanced with overlapping instruction sets as described see 5.3. Since the minimal preconditions should be analysed also only two function units are considered and therefore at most only two instructions can be fetched within one clock cycle. This implies for every stage of this type of superscalar pipeline:

1. There are at most 2 instructions to be fetched in an in-order manner within one clock cycle.
2. Several function units exist with overlapping instruction sets (defined in 5.3). Instructions could be executed at several function units.
3. There are at most 2 results within one clock cycle.

In this case that means, that I_0 and I_1 are fetched before clock cycle 1 and I_2 and I_3 are dispatched before clock cycle 2. If a function unit is not occupied and this function unit is able to execute an instruction, then the instruction is processed immediately. But if the function unit is busy, then the instruction is stalled.

To construct a TA-S-A anomaly the execution of two different instructions have to interchange the function units in different executions E and F. One way to obtain this is to vary the execution time of instruction I_0, that in one case (in E) I_2 has to be realized at FU_1, because FU_0 is busy at this point of time and in another case (in F) the instruction I_2 has to be executed at FU_0, because both function units are not required and FU_0 has a higher priority (in case that I_1 can be processed in either function units). An additional restriction to I_4 causes that this instruction has to run at FU_0. Thereby the executions of I_2 and I_3 change their function units and the faster instruction is performed at the timing relevant function unit, shown in Figure 5.10. Thence it is not necessary to choose a greater difference $\Delta(I_0, s_0, s_1)$, because this would only lead to constant shift in the anomaly criteria.

Theorem 5.4.2 *In superscalar pipelines with Minimal Overlapping FUs and executing 5 instructions TA-S-A anomalies can occur.*

Proof Assume two different executions E, with initial state s_0, and F, with initial state s_1. Because the first two instructions are fetched simultaneously, an interchange of the function units (which is need for the occurrence of a timing anomaly) of two instructions is possible, if $(T(I_0, s_0) - T(I_1, s_0)) \geq 0 \wedge (T(I_0, s_1) - T(I_1, s_1)) \leq 0 \wedge \Delta(I_0, s_1, s_0) > 0$ (this implies decreasing duration time of I_0) or $(T(I_0, s_0) - T(I_1, s_0)) \leq 0 \wedge (T(I_0, s_1) - T(I_1, s_1)) \geq$

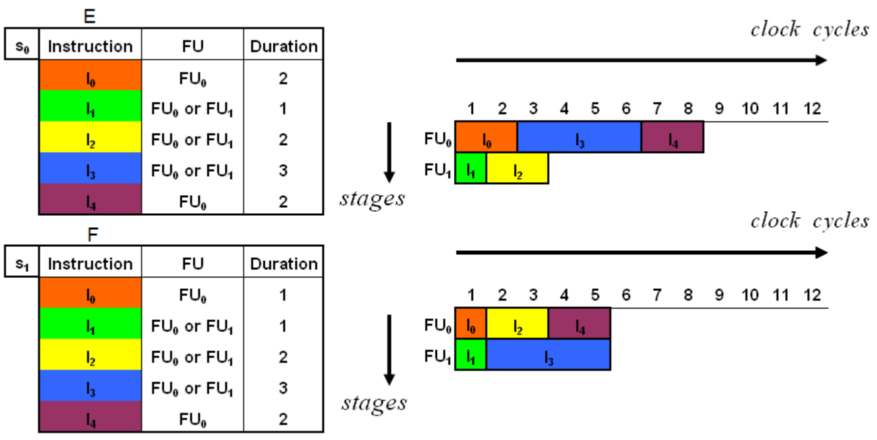
$0 \wedge \Delta(I_0, s_1, s_0) < 0$ (this implies increasing duration time of I_0), if I_1 can be executed in both function units.
If the duration of I_0 is decreased/increased (against an reference execution) and at least two succeeding instructions I_l, I_m can be realized at more than one function unit and one of these instructions I_l is executed at the same function unit in E then the faster/slower instruction I_0 and $T(I_l, s) > T(I_m, s)$/$T(I_l, s) < T(I_m, s)$ and the instructions I_l, I_m change the function units. This situation could lead to a shorter/longer execution time for the whole instruction sequence (TA-S-A).

For this scenario it is important, that the last instruction is restricted to a function unit.□

Superscalar Pipelines with Overlapping Function Units and Executing 4 Instructions

In the above subsection it was shown that TA-S-A anomalies can occur in this kind of pipeline for 5 instructions. With 4 instructions amplification series timing anomalies are not possible. This is displayed in this subsection. For that reason we have to discuss two different cases, the first, where I_1 can be executed at either function unit:
To obtain a TA-S-A anomaly we know from the above sections that an interchange of two instructions has to occur. But the difference $\Delta(I_0, s_1, s_1)$ can only achieve a shift of the beginning point in time of the instruction I_3 with the same value as the difference for the whole instruction sequence execution. This ensures only a constant offset at the TA-S-A criteria. Furthermore instruction I_2 begins at the same clock cycle in both executions and no amplification series timing anomaly can occur, see Figure 5.11.

Now we analyse the second case, where I_1 can only run at function unit FU_0:
In this situation I_1 is stalled and has to wait until I_0 has terminated. Instruction I_2 can be stalled, because it can only be scheduled earliest at the same point where I_1 is processed, because with the in-order execution property the instructions must be processed in ascending order. An



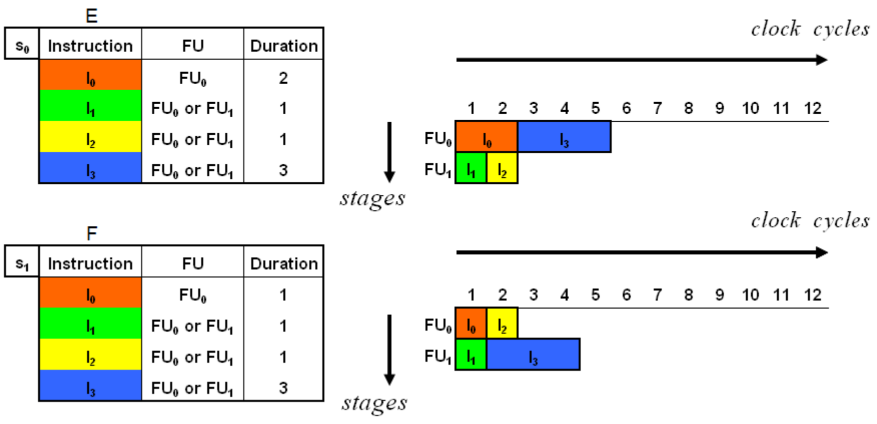
E

s_0	Instruction	FU	Duration
	I_0	FU_0	2
	I_1	FU_0 or FU_1	1
	I_2	FU_0 or FU_1	2
	I_3	FU_0 or FU_1	3
	I_4	FU_0	2

F

s_1	Instruction	FU	Duration
	I_0	FU_0	1
	I_1	FU_0 or FU_1	1
	I_2	FU_0 or FU_1	2
	I_3	FU_0 or FU_1	3
	I_4	FU_0	2

Figure 5.10: Superscalar pipeline with TA-S-A and 5 instructions



E

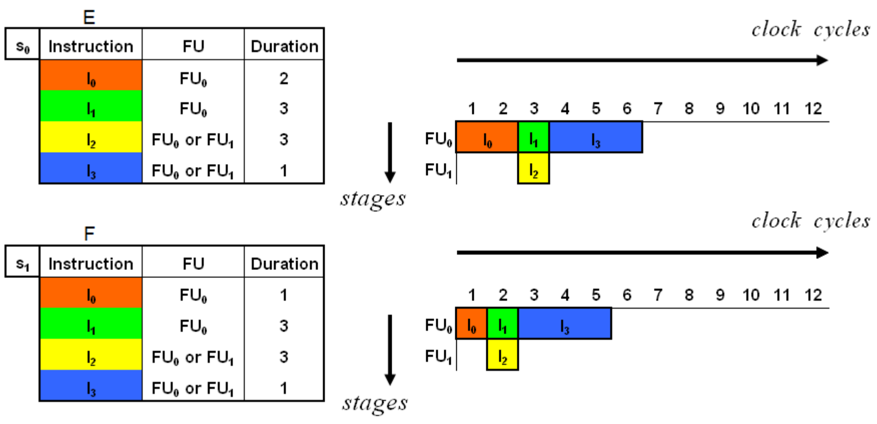
s_0	Instruction	FU	Duration
	I_0	FU_0	2
	I_1	FU_0 or FU_1	1
	I_2	FU_0 or FU_1	1
	I_3	FU_0 or FU_1	3

F

s_1	Instruction	FU	Duration
	I_0	FU_0	1
	I_1	FU_0 or FU_1	1
	I_2	FU_0 or FU_1	1
	I_3	FU_0 or FU_1	3

Figure 5.11: Superscalar pipeline trying to construct a TA-S-A with 4 instructions and I_1 can be executed at both function units

example is shown in Figure 5.12. One can see, that the starting points of the three instructions I_1, I_2 and I_3 are shifting in all executions with the same value as the difference $\Delta(I_0, s_1, s_1)$. Thus no TA-S-A anomaly can occur.



E

s_0	Instruction	FU	Duration
	I_0	FU_0	2
	I_1	FU_0	3
	I_2	FU_0 or FU_1	3
	I_3	FU_0 or FU_1	1

F

s_1	Instruction	FU	Duration
	I_0	FU_0	1
	I_1	FU_0	3
	I_2	FU_0 or FU_1	3
	I_3	FU_0 or FU_1	1

Figure 5.12: Superscalar pipeline trying to construct a TA-S-A with 4 instructions and I_1 can only be executed at FU_0

Theorem 5.4.3 *In superscalar pipelines with Minimal Overlapping FUs and executing 4 instructions TA-S-A anomalies can not occur.*

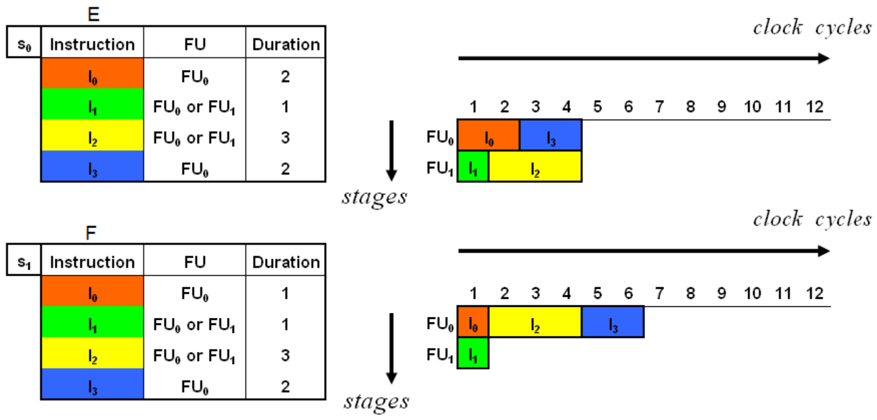
Proof If I_1 can be executed in either function unit:
Because the first two instructions are fetched simultaneously, an interchange of the function units of the executions of two instructions is possible if $(T(I_0, s_0) - T(I_1, s_0)) \geq 0 \wedge (T(I_0, s_1) - T(I_1, s_1)) \leq 0 \wedge \Delta(I_0, s_1, s_0) > 0$ (this implies decreasing the duration of I_0) or $(T(I_0, s_0) -$

$T(I_1, s_0)) \leq 0 \wedge (T(I_0, s_1) - T(I_1, s_1)) \geq 0 \wedge \Delta(I_0, s_1, s_0) < 0$ (this implies increasing the execution time of I_0) if I_1 can be executed in either function unit.
The difference $\Delta(I_0, s_1, s_1)$ can lead to a shift of the starting point of I_3 with the same value as the difference. Due to the TA-S-A criteria, this does not lead to an anomaly.
If I_1 can only be performed in FU_0:

- I_0 runs at FU_0.
- I_1 is stalled until I_0 has terminated.
- I_2 can be stalled until I_1 start its in-order execution.

The starting points of I_0, I_1 and I_2 differ by the difference $\Delta(I_0, s_1, s_1)$. Due to the TA-S-A criteria, this does not lead to an anomaly.$\square$

Now we analyse if TA-S-I anomalies for this kind of pipeline and 4 instructions can occur. For this sort of anomaly it is only necessary that one instruction changes the function unit on which it is executed. In Figure 5.13 one way to create such an timing anomaly is presented on which I_2 changes its executing function unit from FU_1 to FU_0. Along to the above defined criterion for this kind of pipeline, the execution of these two instructions interchange the function units for the durations of I_0 and I_1, this is used to force the execution of I_2 in the other function unit, but reduce the instruction set of FU_1, so that I_3 can only be realized at FU_0. This leads to an inversion series timing anomaly.



E

s_0	Instruction	FU	Duration
	I_0	FU_0	2
	I_1	FU_0 or FU_1	1
	I_2	FU_0 or FU_1	3
	I_3	FU_0	2

F

s_1	Instruction	FU	Duration
	I_0	FU_0	1
	I_1	FU_0 or FU_1	1
	I_2	FU_0 or FU_1	3
	I_3	FU_0	2

Figure 5.13: Superscalar pipeline with TA-S-I and 4 instructions

Theorem 5.4.4 *In superscalar pipelines with Minimal Overlapping FUs and executing 4 instructions TA-S-I anomalies can occur.*

Proof Assume two different executions E, with initial state s_0, and F, with initial state s_1. Because the first two instructions are fetched simultaneously, an interchange of the function units

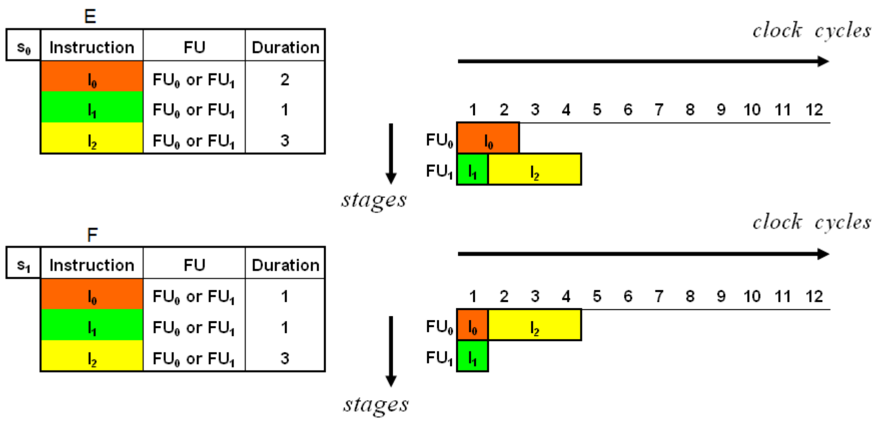
of the executions of two instructions is possible if $(T(I_0, s_0) - T(I_1, s_0)) \geq 0 \wedge (T(I_0, s_1) - T(I_1, s_1)) \leq 0 \wedge \Delta(I_0, s_1, s_0) > 0$ (this implies decreasing the duration of I_0) or $(T(I_0, s_0) - T(I_1, s_0)) \leq 0 \wedge (T(I_0, s_1) - T(I_1, s_1)) \geq 0 \wedge \Delta(I_0, s_1, s_0) < 0$ (this implies increasing the execution time of I_0) if I_1 can be executed in either function unit.
If the execution time of I_0 is decreased/increased (against a reference execution) and it is processed in the function unit FU_i in E, then it is possible that the succeeding instruction (which can be executed at either function unit and is processed in another function unit FU_j / the same function unit FU_i) is forced to be executed at FU_i/FU_j in F, this could result in a longer/shorter execution time than in E (TA-S-I).

For this scenario it is important that the last instruction is bound to a specific function unit.□

Superscalar Pipelines with Overlapping Function Units and Executing 3 Instructions

In the previous subsection it was shown that TA-S-A anomalies can not occur executing 4 instructions in this sort of pipeline, but it is possible, that TA-S-I anomalies can appear. In this subsection we analyse the situation with 3 instructions. To create a TA-S-I anomaly different resource allocation for two executions have to be done. For this purpose it is necessary to distinguish three situations. Note that in each case it is not essential, that I_0 can be processed at either function unit, because it is fetched before the first clock cycle and by the in-order execution.

First, it is feasible for each instruction to be executed at either function unit:
Instruction I_0 is dispatched before the first clock cycle and is executed at FU_0. Thereby instruction I_1 is also fetched before the first clock cycle and is performed at FU_1. Both instructions are executed in this order, regardless of the durations. Thus it is only possible that the execution of I_2 interchanges the function unit. But it is obvious, that the start point of I_2 can maximally shift by the same value as $\Delta(I_0, s_0, s_1)$, see Figure 5.14.


E

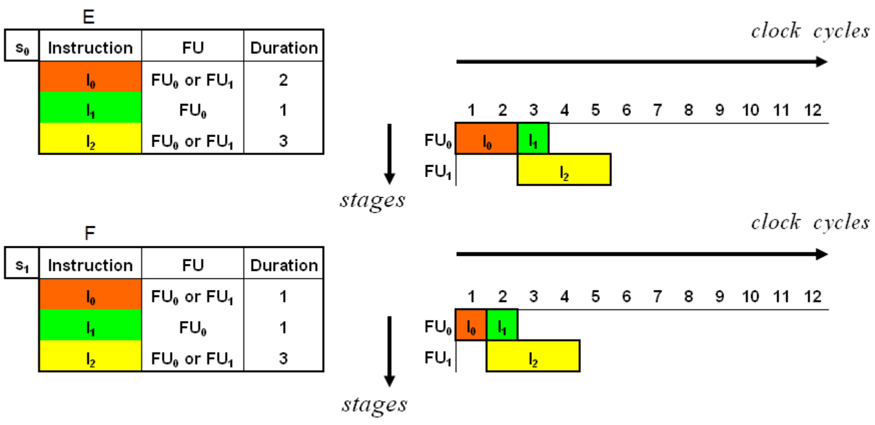
s_0	Instruction	FU	Duration
	I_0	FU_0 or FU_1	2
	I_1	FU_0 or FU_1	1
	I_2	FU_0 or FU_1	3

F

s_1	Instruction	FU	Duration
	I_0	FU_0 or FU_1	1
	I_1	FU_0 or FU_1	1
	I_2	FU_0 or FU_1	3

Figure 5.14: Superscalar pipeline trying to construct a TA-S-I with 3 instructions and I_0, I_1 and I_2 can be executed at each function unit

Second, both, I_0 and I_2, are allowed to execute on either function unit. I_1 is bound to a fixed function unit, FU_0:
Instruction I_0 runs at FU_0 in the first clock cycle. I_1 is stalled until I_0 terminates, because it has to be executed at FU_0 also (Figure 5.15). That means that instruction I_2 must be processed at FU_1 in every execution, because $T(I_0, s) + T(I_1, s) \geq 2$ and I_2 is fetched before clock cycle 2 and FU_1 cannot be busy at this point of time. The starting point of I_1 and I_2 depends on the duration of I_0. Then the overall duration varies by the same value as I_0 does.


E

s_0	Instruction	FU	Duration
	I_0	FU_0 or FU_1	2
	I_1	FU_0	1
	I_2	FU_0 or FU_1	3

F

s_1	Instruction	FU	Duration
	I_0	FU_0 or FU_1	1
	I_1	FU_0	1
	I_2	FU_0 or FU_1	3

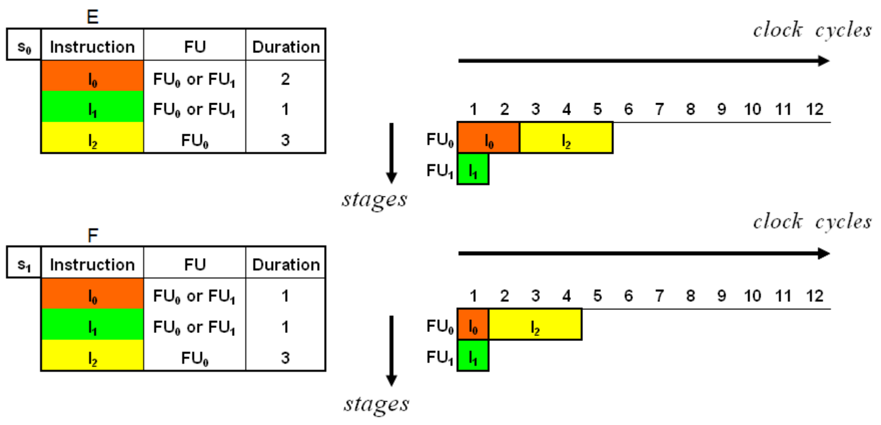
Figure 5.15: Superscalar pipeline trying to construct a TA-S-I with 3 instructions and I_0, I_2 can be executed at each function unit

Third, both, I_0 and I_1, are allowed to execute on either function unit. I_2 is bound to a fixed function unit, FU_0:
Again, instruction I_0 is performed at FU_0 in the first clock cycle. If I_1 can be realized at either function unit and it is fetched before the first clock cycle and FU_0 is occupied by I_0, instruction I_1 runs at FU_1. Regardless of their durations both instructions are executed in this order and they have the same starting point in each execution. Thereby I_2 can only be executed at FU_0 after the termination of I_0. As a result the starting point of I_2 varies also with the same value as the difference $\Delta(I_0, s_1, s_1)$ (example see at Figure 5.16).

Theorem 5.4.5 *In superscalar pipelines with Minimal Overlapping FUs and executing 3 instructions TA-S-I anomalies can not occur.*

Proof Note that in each case it is not essential, that I_0 can be processed at both function units, because it is fetched in the first clock cycle.
We have to distinguish the before introduced three cases: I_0, I_1, I_2 can be executed at either function unit:

- I_0 is performed at FU_0 in the first clock cycle in every execution.
- I_1 is performed at FU_1 in the first clock cycle in every execution.



E

s_0	Instruction	FU	Duration
	I_0	FU_0 or FU_1	2
	I_1	FU_0 or FU_1	1
	I_2	FU_0	3

F

s_1	Instruction	FU	Duration
	I_0	FU_0 or FU_1	1
	I_1	FU_0 or FU_1	1
	I_2	FU_0	3

Figure 5.16: Superscalar pipeline with trying to construct a TA-S-I with 3 instructions and I_0, I_1 can be executed at each function unit

- I_2 can be executed either at FU_0 or at FU_1 but the start point of I_2 can shift at most by the same value as $\Delta(I_0, s_1, s_1)$, hence no TA-S-I.

I_0, I_2 can be executed at each of the two function units:

- I_0 is executed at FU_0 in the first clock cycle in every execution.
- I_1 is executed next to the termination of I_0 at FU_0 in every execution.
- I_2 must be executed at FU_1, because $T(I_0, s) + T(I_1, s) \geq 2$ and I_2 is fetched before clock cycle 2 and FU_1 cannot be busy at this point of time and FU_0 is occupied. Then the overall execution time varies by the same value as I_0 does. Thus there is no TA-S-I anomaly.

I_0, I_1 can be executed at each of the two function units:

- I_0 is performed at FU_0 in the first clock cycle in every execution.
- I_1 is performed at FU_1 in the first clock cycle in every execution.
- I_2 must be executed at FU_0 after the termination of I_0 and the start point from I_2 varies with the same value as the difference $\Delta(I_0, s_1, s_1)$. Thus there is no TA-S-I anomaly.

□

5.5 Timing Anomalies in Superscalar Out-Of-Order Pipelines

In this section the occurrence of timing anomalies in out-of-order pipelines are analysed. In consideration of the definitions in Section 3.5 the stages consists of a *DI-stage* (dispatcher), the execution stage with the function units *FU-stage* and the *WB-stage* (write back), whereby the *DI-*, *FU-* and *WB-stages* could be divided into more stages of the same type.
To analyse the minimal preconditions, some restrictions are defined for each stage:

1. There is at most one instruction to be fetched within one clock cycle.
2. The instruction sets of the function units are disjunctive, but more than one function unit with the same instruction set is permitted, more precisely: Assume that the existing function units FU_0 to FU_n (in this case $n = 1$) can only process instructions c, which are in the instruction sets $c \in IS_i$ (with $0 \leq i \leq n$). That means for superscalar out-of-order pipelines $(IS_j \cap IS_k = \emptyset) \vee (IS_j = IS_k)$, with $0 \leq j, k \leq n$. Instructions could be executed out-of-order at several function units.
3. At most n results are written back out-of-order within one clock cycle.

Suppose that the *DI-stage* can contain only one instruction and as a result at most one instruction can be dispatched. Additionally assume enough write back-stages where all results can be written back in the clock cycle after they have been executed in the function units. With all this, it is possible to omit some stages to make the figures easier to understand, see Figure 5.17.

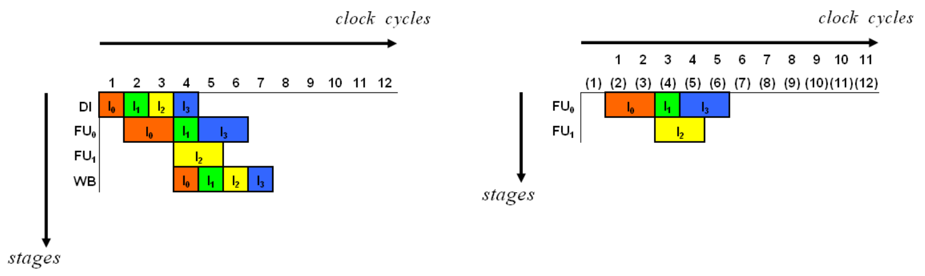
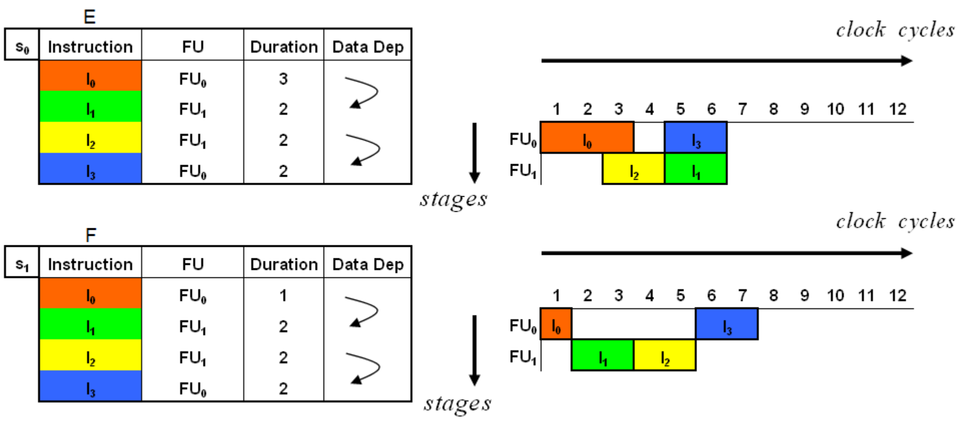


Figure 5.17: Pipeline visualization of complex out-of-order pipeline processes

Superscalar Out-Of-Order Pipelines and Execution 4 Instructions

Suppose that every clock cycle one instruction is fetched and data dependencies between instructions exist. Then the execution sequence depends on the execution time of the instructions. The potential that a timing anomaly occurs is also a subject of the execution order of the instructions. In the next diagrams data dependencies are marked by arrows, where the end point of an arrow shows the instruction that requires operands from the instruction where the arrow starts.

If we suppose that the instructions I_0 and I_3 can only be performed at the function unit FU_0 and the instructions I_1 and I_2 can only be processed at the function unit FU_1 and data dependencies exist between $I_0 \rightarrow I_1$ and $I_2 \rightarrow I_3$, then it is possible that a TA-S-I anomaly can occur, see Figure 5.18. This is due to the fact that I_1 cannot begin its execution in E at the point when it is fetched, because I_0 has not finished then and there exists a data dependency. As a result I_2 occupies FU_1 at clock cycle 3 and I_1 can be executed only after I_2 at FU_1. At the same point I_3 can start its execution at FU_0, because it was stalled due to a data dependency. In F the duration of I_0 is decreasing to 1 and thereby I_1 can start from the earliest point. In this situation I_2 is stalled and I_3 can start its execution only after the execution of I_2. Thus a TA-S-I anomaly occurs ($\Delta(I_0, s_1, s_0) > 0 \wedge \Delta(I, s_1, s_0) < 0$).



E

s₀ Instruction	FU	Duration	Data Dep
I_0	FU_0	3	
I_1	FU_1	2	
I_2	FU_1	2	
I_3	FU_0	2	

F

s₁ Instruction	FU	Duration	Data Dep
I_0	FU_0	1	
I_1	FU_1	2	
I_2	FU_1	2	
I_3	FU_0	2	

Figure 5.18: Superscalar Out-Of-Order pipeline with TA-S-I and 4 instructions

Theorem 5.5.1 *In superscalar out-of-order pipelines and executing 4 instructions TA-S-I anomalies can occur.*

Proof A TA-S-I timing anomaly with executing 4 instructions in superscalar out-of-order pipelines can be achieved in the following way: Suppose that the starting point of an execution of an instruction I_j is termed as $SP(I_j)$. The preconditions of a TA-S-I anomaly in this pipeline are:

- At least two instructions I_j, I_k changing their execution order in one function unit, more precisely: If $SP(I_j) < SP(I_k)$ in E, then in F $SP(I_j) > SP(I_k)$.
- The instruction I_j has a data dependency to another instruction $I_l \neq I_j, I_k$, in the way $I_j \rightarrow I_l$.

With these preconditions TA-S-I anomalies can occur with executing 4 instructions in this kind of pipeline.□

Is it also possible that TA-S-A anomalies occur in such a pipeline with 4 instructions? If we suppose that instruction I_0 and I_3 could only be realized at FU_0 and the instructions I_1 and I_2 could only be performed at FU_1 and data dependencies $I_0 \rightarrow I_1$, $I_1 \rightarrow I_3$ exist, then the execution of I_1 is stalled in E, because I_1 can only be processed after I_0 due to the dependency between I_0 and I_1 and I_2 can be executed before I_1 at FU_1. Then the instruction I_3 can be executed after the termination of I_1 due to the dependency in E. If the execution time of I_0 is decreased, I_1 is not stalled in F and I_2 and I_3 could begin their execution after the termination of I_1, see Figure 5.19. This leads to a TA-S-A anomaly, $0 < \Delta(I_0, s_1, s_0) < \Delta(I, s_1, s_0)$.

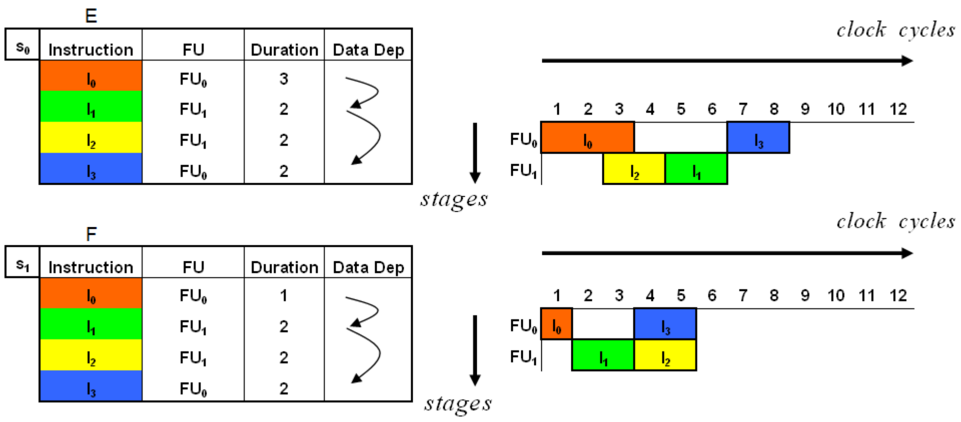


Figure 5.19: Superscalar Out-Of-Order pipeline with TA-S-A and 4 instructions

Theorem 5.5.2 *In superscalar out-of-order pipelines and executing 4 instructions TA-S-A anomalies can occur.*

Proof A TA-S-A timing anomaly with executing 4 instructions in superscalar out-of-order pipelines can be achieved in the following way: Suppose that the starting point of an execution of an instruction I_j is termed as $SP(I_j)$. The preconditions of a TA-S-A anomaly in this pipeline are:

- At least two instructions I_j, I_k changing their execution order in one function unit, more precisely: If $SP(I_j) < SP(I_k)$ in E, then in F $SP(I_j) > SP(I_k)$.
- The instruction I_k has a data dependency to another instruction $I_l \neq I_j, I_k$, in the way $I_k \to I_l$.

With these preconditions TA-S-A anomalies can occur with executing 4 instructions in this kind of pipeline.□

Superscalar Out-Of-Order Pipelines and 3 instructions

In this subsection an analysis for this kind of pipeline with 3 instructions within an execution is done.

Theorem 5.5.3 *In superscalar out-of-order pipelines and executing 3 instructions TA-S-I and TA-S-A anomalies cannot occur.*

Proof The proof from [17] can be applied.□

5.6 Summary

Table 5.1 gives an overview of the timing anomalies, that can occur on pipeline architectures. One can see, that overlapping instruction sets of the function units or/and out-of-order execution are necessary for the appearance of a timing anomaly. If a timing anomaly occurs in a specific type of pipeline with k instructions that this timing anomaly also appears with more than k instructions. Otherwise, if a timing anomaly is not possible in an execution of k instructions, then this timing anomaly is also not achievable with less than k instructions. If the reader is interested in the occurrence of timing anomalies in other pipeline architectures then the models in [17] should be consulted.

Type of Pipeline	Instructions	TA-S-I	TA-S-A
Simple Scalar Pipeline		no	no
Scalar Pipelines without overlapping Function Units		no	no
Scalar Pipelines with overlapping Function Units	4	yes	yes
Scalar Pipelines with overlapping Function Units	3	yes	no
Scalar Pipelines with overlapping Function Units	2	no	no
Superscalar Pipelines with Dual-Issue and Dual-Complete		no	no
Superscalar Pipelines with overlapping Function Units	5	yes	yes
Superscalar Pipelines with overlapping Function Units	4	yes	no
Superscalar Pipelines with overlapping Function Units	3	no	no
Superscalar Pipelines with overlapping Function Units	2	no	no
Superscalar Out-Of-Order Pipelines	4	yes	yes
Superscalar Out-Of-Order Pipelines	3	no	no

Table 5.1: Summary of Timing Anomalies on Pipelines

CHAPTER 6

Timing Anomalies in Caches

6.1 Overview

For simplicity we consider caches as isolated parts of a processor in this chapter. If we assume a real program which consists of a sequence of instructions then each instruction is able to require some data blocks (usually between zero to two) of the main memory. Before execution these data blocks are searched in the cache to accelerate the data access. Thus it is possible to abstract a real program to a series of cache accesses of data blocks. In the course of this thesis this abstraction is used. If we consider two different executions and a cache/memory access is required, then a cache hit at an instruction in the first execution leads in contrast to a cache miss at the same instruction to a variation of the execution duration of this instruction or/and an instruction sequence.
Furthermore series timing anomalies that can occur in cache architectures are discussed and analysed in this chapter.
The structure and the functionality of caches have been presented in Chapter 4. Now some cache replacement strategies are discussed with regard to timing anomalies. Some new definitions concerning timing anomalies in caches are presented. Next, the influence of replacement strategies on the occurrence of timing anomalies is analysed.

6.2 Definition of Timing Anomalies on Caches

To analyse timing anomalies on caches a precise definition for anomalies is needed. In Chapter 5, in which timing anomalies on pipelines are studied, the behaviour of an initial timing situation (the execution of the first instruction in pipeline architectures) is compared with a timing situation after the execution of some instructions in a pipeline.
If this concept is applied to caches, the timing of an initial cache situation must be matched with a timing situation, where some cache accesses of several instructions are processed. If we assume two different executions, then in an initial timing situation the execution time on the first

same cache access is changing in both executions. More precisely if the first cache access in one execution results in a cache hit/miss then in the second execution a cache miss/hit occur.
If a timing situation after some instructions or cache accesses can be observed then the focus is on the number of hits/misses during the execution of these instructions. From the number of hits/misses the execution time can be derived. This leads to the definition of timing anomalies for caches:

Defintion 6.2.1 $\Delta(MB, s, s') = T(MB, s') - T(MB, s)$ *is the difference of execution times (respective difference between cache hits and cache misses) for a cache access for memory block* MB *for different initial states* $s, s' \in$ *TRDCS, where a cache hit corresponds to zero clock cycles and a cache miss amounts* k *clock cycle. For simplicity we assume that* $k = 1$ *in this thesis.*

Defintion 6.2.2 ***TA-S-I***
Assume an instruction sequence I*, with cache accesses* $CA = M_0$ *to* M_n *and the initial states* $s, s' \in TRDCS$*. Then an inversion series timing anomaly is a situation, where:*
$\exists s, s' \in TRDCS.\Delta(M_0, s, s') > 0 \wedge \Delta(CA, s, s') < 0$

Assume two different executions E with initial state s_0 and F with initial state s_1. That means, that for an inversion series timing anomaly it is necessary that at least three hits happen in one execution while at the same cache accesses three misses appear in another execution. Because if the initial cache access in E is a hit and in F this access leads to a miss, then in E at least two more misses than in F should happen so, that the overall duration of E becomes longer than the duration of F. There are the minimum preconditions that such an anomaly occurs. A pair

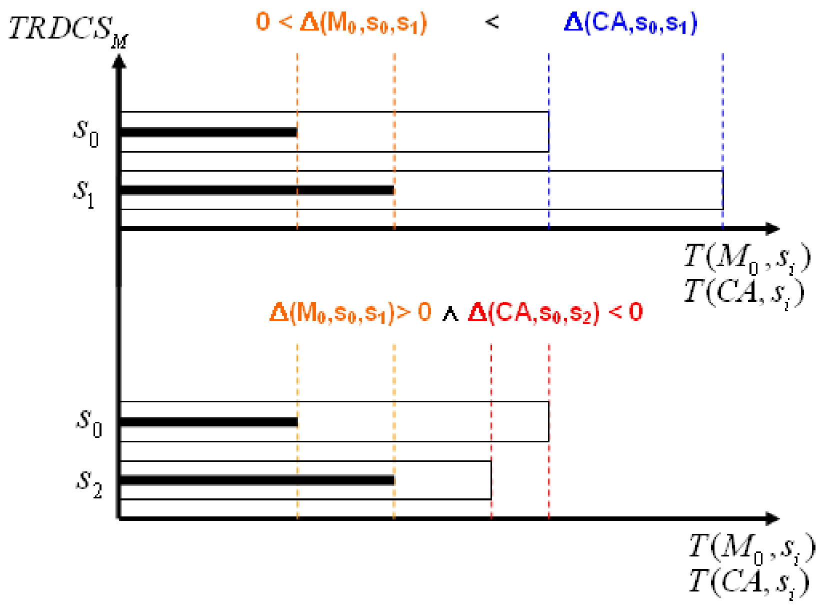


Figure 6.1: Timing Anomalies of Cache Architectures

of hit/miss shall be understood as a hit in one execution and a miss in another execution for the same cache access. If a hit (or miss) occurs in an execution and a hit (or miss) appears in another execution for the same memory access, this does not lead to a change in the difference of the execution times.

Defintion 6.2.3 ***TA-S-A***
Assume an instruction sequence I, with cache accesses $CA = M_0$ to M_n and the initial states $s, s' \in TRDCS$. Then an amplification series timing anomaly is a situation, where:
$\exists s, s' \in TRDCS.0 < \Delta(M_0, s, s') < \Delta(CA, s, s')$

Let us assume two different executions, E with initial state s_0 and F with initial state s_1. For the occurrence of an amplification series timing anomaly only two pairs of hits/misses are essential. Because if initially a cache hit happens in E and a cache miss appears initially in F, then only one additional hit must occur in E for another miss in F to achieve such an anomaly. If this situation appears, the overall duration of E is shorter than the execution time of F and the difference of the overall durations is greater than the difference at the initial cache access. These are the minimal preconditions of such a timing anomaly.

In Figure 6.1 a graphical representation of these definitions is illustrated.

If we assume two executions, E with the initial state s_0 and F with the initial state s_1, then these initial states can only differ in the initial cache content. Note that also the state bits are included in the initial cache state in case of the PLRU replacement strategy. The order of the memory accesses to the cache is the same in both executions, because the memory accesses arise from instructions, which need data from the memory for the executions. Note that in PRLU caches also the state bits of the virtual tree are important for the occurrence of timing anomalies. In the next figures cache blocks that are accessed are named with a number-index, whereas cache blocks that are not needed during the regarding execution are labelled with a character-index.

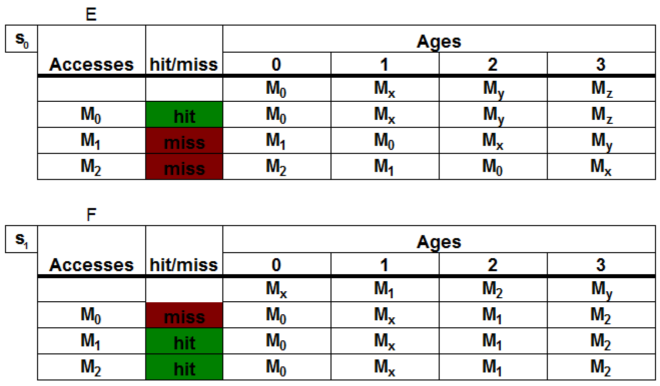
6.3 Timing Anomalies in FIFO Caches

In Section 4.4 the FIFO replacement strategy was explained. Now timing anomalies with the before explained definition are analysed for this kind of cache architecture. If a data block is referenced and it is not part of the cache content, then the memory block is taken from the main memory and is saved in the first cache line and the last cache line (related to the associativity) is removed. If a cache hit happens, then the cache content remains unchanged.

4-way associative FIFO Caches

Theorem 6.3.1 *In 4-way associative FIFO caches TA-S-I timing anomalies can occur.*

Proof For a TA-S-I anomaly the number of misses must be greater in E than in F, although in E a cache hit takes place instead of a cache miss in F. In this kind of caches it is trivial to construct such an anomaly. One possibility is that the first needed block is in the initial state s_0, however the initial state s_1 does not contain this required block. And two other subsequent



E

s_0			Ages			
Accesses	hit/miss	0	1	2	3
		M_0	M_x	M_y	M_z
M_0	hit	M_0	M_x	M_y	M_z
M_1	miss	M_1	M_0	M_x	M_y
M_2	miss	M_2	M_1	M_0	M_x

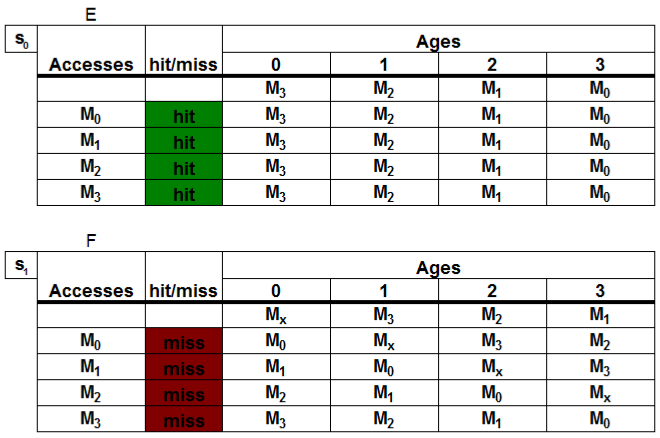
F

s_1			Ages			
Accesses	hit/miss	0	1	2	3
		M_x	M_1	M_2	M_y
M_0	miss	M_0	M_x	M_1	M_2
M_1	hit	M_0	M_x	M_1	M_2
M_2	hit	M_0	M_x	M_1	M_2

Figure 6.2: 4-way associative FIFO, TA-S-I cache timing anomaly

accessed blocks are in the initial state s_1, instead these blocks are not in cache in s_0. An example of such a TA-S-I cache timing anomaly is shown in Figure 6.2. □

Theorem 6.3.2 *In 4-way associative FIFO caches TA-S-A timing anomalies can occur.*



E

s_0			Ages			
Accesses	hit/miss	0	1	2	3
		M_3	M_2	M_1	M_0
M_0	hit	M_3	M_2	M_1	M_0
M_1	hit	M_3	M_2	M_1	M_0
M_2	hit	M_3	M_2	M_1	M_0
M_3	hit	M_3	M_2	M_1	M_0

F

s_1			Ages			
Accesses	hit/miss	0	1	2	3
		M_x	M_3	M_2	M_1
M_0	miss	M_0	M_x	M_3	M_2
M_1	miss	M_1	M_0	M_x	M_3
M_2	miss	M_2	M_1	M_0	M_x
M_3	miss	M_3	M_2	M_1	M_0

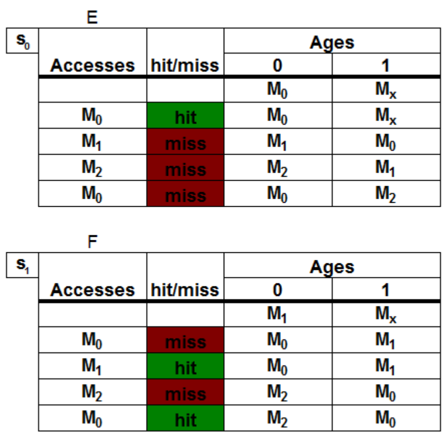
Figure 6.3: 4-way associative FIFO, TA-S-A cache timing anomaly

Proof For a serial amplification timing anomaly it is essential to receive more misses in F than in E and initially in E a cache hit happens as opposed to F, where a miss happens. This can be achieved, if the initial state s_0 contains at least two blocks, which are accessed initially and these are not in the initial state s_1 or they are overwritten. Thus the definition of a TA-S-A cache timing anomaly is fulfilled, with an example in Figure 6.3. □

2-way associative FIFO Caches

There is no difference in the replacement strategy for 2-way than in 4-way associative FIFO caches, except that of course two cache lines can be chosen. In this subsection also the occurrence of series inversion and amplification timing anomalies for this kind of caches are analysed.

Theorem 6.3.3 *In 2-way associative FIFO caches TA-S-I timing anomalies can occur.*



E

s_0

		Ages	
Accesses	hit/miss	0	1
		M_0	M_x
M_0	hit	M_0	M_x
M_1	miss	M_1	M_0
M_2	miss	M_2	M_1
M_0	miss	M_0	M_2

F

s_1

		Ages	
Accesses	hit/miss	0	1
		M_1	M_x
M_0	miss	M_0	M_1
M_1	hit	M_0	M_1
M_2	miss	M_2	M_0
M_0	hit	M_2	M_0

Figure 6.4: 2-way associative FIFO, TA-S-I cache timing anomaly

Proof Assume two different executions, E with initial state s_0 and F with initial state s_1. To construct a serial inversion timing anomaly it is necessary that in the execution where initially a hit takes place the overall execution time is longer than in the execution where initially a miss happens. An example to construct such an anomaly is that in E the first required data block is initially in the cache, while in F this data block is not in s_1 and a miss occurs. If the next cache query leads to a miss in E in contrast to a cache hit in F, then the caches have the same content. When the next access is a miss in both executions and the cache line with age 1 in E is different from the cache line with age 1 in F, then an additional cache request to the content of the cache line with age 1 in F leads to a hit and a miss in E. Thus the conditions of a TA-S-I anomaly are fulfilled. An example is shown in Figure 6.4. □

Theorem 6.3.4 *In 2-way associative FIFO caches TA-S-A timing anomalies can occur.*

Proof For a serial amplification timing anomaly it is essential to get more misses in F than in E and initially in E a cache hit happens as opposed in F, where a miss happens. If a second required data block is in the cache content in E, but this block is not in the cache content in F, this produces a second hit in E, whereas the second access is a miss in F and the definition of a TA-S-A cache timing anomaly is fulfilled, see Figure 6.5. □

6.4 Timing Anomalies in LRU Caches

In Section 4.4 the functionality of LRU caches is presented. If a cache miss happens, then the new cache element is put in the cache line with age 0 and all other cache lines increase their ages by 1, except the cache line with age $k-1$, because the content of this line is ejected, in k-way associative LRU caches.
In case of a cache hit the age of the needed cache line (say M_i with age a) is set to 0, all cache lines with age 0 to $a-1$ increase their age by 1.

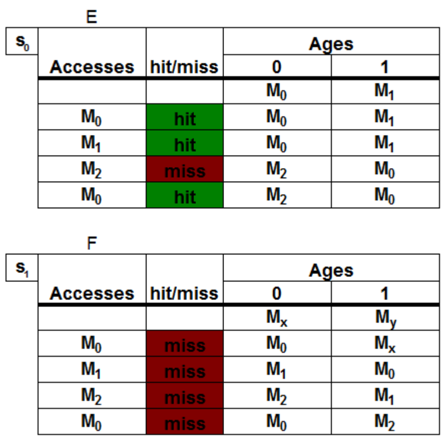
4-way associative LRU Caches

Theorem 6.4.1 *In 4-way associative LRU caches TA-S-I timing anomalies can occur.*

Proof Assume two different executions, E with initial state s_0 and F with initial state s_1. A TA-S-I anomaly can be reached, if the required data of the first cache query is contained in the initial state in E, but it is not hold in the initial state in F. But the next at least two data blocks are not covered in the initial state s_0, however these blocks are present in the initial state s_1. An example of such a TA-S-I cache timing anomaly is shown in Figure 6.6. □

Theorem 6.4.2 *In 4-way associative LRU caches TA-S-A timing anomalies can occur.*

Proof Assume two different executions, E with initial state s_0 and F with initial state s_1. Then this kind of an anomaly can be reached in 4-way associative LRU caches, if at least two hits happen in E (e.g. the first two needed data elements are in the initial state s_0) and at the same accesses two misses occur in F (e.g. the required data blocks are not in the initial state s_1). Thus the definition of a TA-S-A cache timing anomaly is fulfilled, with an example in Figure 6.7. □

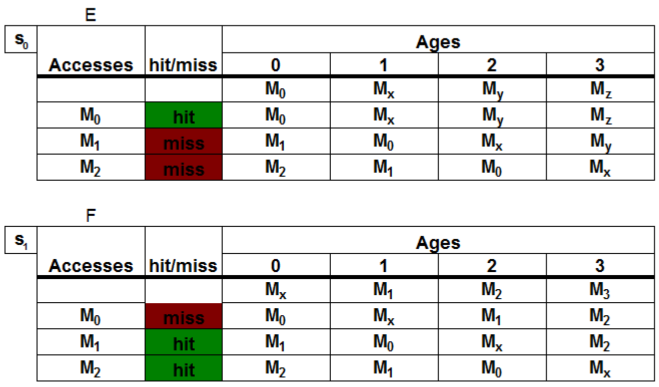

E

s_0			Ages	
	Accesses	hit/miss	0	1
			M_0	M_1
	M_0	hit	M_0	M_1
	M_1	hit	M_0	M_1
	M_2	miss	M_2	M_0
	M_0	hit	M_2	M_0

F

s_1			Ages	
	Accesses	hit/miss	0	1
			M_x	M_y
	M_0	miss	M_0	M_x
	M_1	miss	M_1	M_0
	M_2	miss	M_2	M_1
	M_0	miss	M_0	M_2

Figure 6.5: 2-way associative FIFO, TA-S-A cache timing anomaly



E

s_0

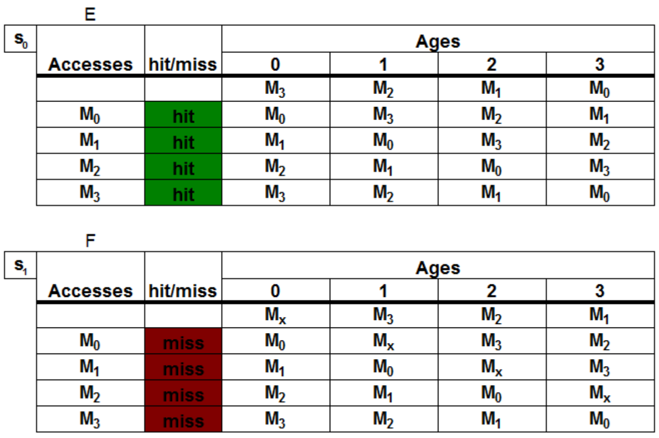
Accesses	hit/miss	Ages			
		0	1	2	3
		M_0	M_x	M_y	M_z
M_0	hit	M_0	M_x	M_y	M_z
M_1	miss	M_1	M_0	M_x	M_y
M_2	miss	M_2	M_1	M_0	M_x

F

s_1

Accesses	hit/miss	Ages			
		0	1	2	3
		M_x	M_1	M_2	M_3
M_0	miss	M_0	M_x	M_1	M_2
M_1	hit	M_1	M_0	M_x	M_2
M_2	hit	M_2	M_1	M_0	M_x

Figure 6.6: 4-way associative LRU, TA-S-I cache timing anomaly



E

s_0

Accesses	hit/miss	Ages			
		0	1	2	3
		M_3	M_2	M_1	M_0
M_0	hit	M_0	M_3	M_2	M_1
M_1	hit	M_1	M_0	M_3	M_2
M_2	hit	M_2	M_1	M_0	M_3
M_3	hit	M_3	M_2	M_1	M_0

F

s_1

Accesses	hit/miss	Ages			
		0	1	2	3
		M_x	M_3	M_2	M_1
M_0	miss	M_0	M_x	M_3	M_2
M_1	miss	M_1	M_0	M_x	M_3
M_2	miss	M_2	M_1	M_0	M_x
M_3	miss	M_3	M_2	M_1	M_0

Figure 6.7: 4-way associative LRU, TA-S-A cache timing anomaly

2-way associative LRU Caches

Theorem 6.4.3 *In 2-way associative LRU caches TA-S-I timing anomalies can not occur.*

Proof Assume two different executions E, with initial state s_0 and F with initial state s_1. To construct a serial inversion timing anomaly it is necessary that in the execution, where initially a hit takes place, say E with initial state s_0, the overall execution time is longer than in the execution, where an initial miss happens, say F with the initial state s_1. We know from above, that such an anomaly can only occur if at least three hit/miss-pairs can occur. In this proof it will be shown, that this is impossible for 2-way LRU caches.
After the first cache access the cache line with age 0 has the same content in every execution. This is shown with the next four possibilities for two executions:

- hit in E and F: the required data is situated in the cache line with age 0 after the hit in both executions.
- miss in E and F: the required block is contained in the cache line with age 0 after the miss in both executions.
- hit in E and miss in F: the required block was in the cache in E and is now in the cache line with age 0 similar to F after the miss.
- miss in E and hit in F: the required block was in the cache in F and is now in the cache line with age 0 similar as in E after the miss.

For the next cache request also four cases are possible: (Note that after the first access the cache line with age 0 is the same in every execution.)

- hit in E and F: If the required data is in the line with age 0 in E, then we know from above that this data is also in the line with age 0 in F. The state of the cache is the same as before the hit in both executions. If the required data is in the line with age 1 in E, then this data have to be also contained in the line with age 1 in F and after the hit the cache state is equal in both executions.
- miss in E and F: the new data is put into line 0 in both executions. The data that had age 0 before the miss must be equal in both execution (fact from above) and increase the age by one. Thus the cache states are equal in every execution.
- hit in E and miss in F: in E the required data is put in the line with age 0 and the data element that had age 0 before (must be equal with data element with age 0 in F) increases the age to 1. In F the new data causes a miss and is put into line 0 and the data with age 0 increases the age to 1. Thus the resulting cache states are equal in every execution after two cache accesses.

E

s_0			Ages	
	Accesses	hit/miss	0	1
			M_0	M_x
	M_0	hit	M_0	M_x
	M_1	miss	M_1	M_0
	M_2	miss	M_2	M_1
	M_0	miss	M_0	M_2

F

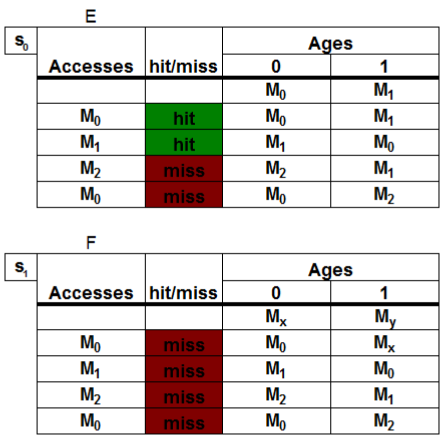
s_1			Ages	
	Accesses	hit/miss	0	1
			M_1	M_x
	M_0	miss	M_0	M_1
	M_1	hit	M_1	M_0
	M_2	miss	M_2	M_1
	M_0	miss	M_0	M_2

Figure 6.8: 2-way associative LRU, best try to achieve a TA-S-I cache timing anomaly

- miss in E and hit in F: in F the required data is put in the line with age 0 and the data element that had age 0 before (must be equal with data element with age 0 in F) increases the age to 1. In E the new data causes a miss and is put into line 0 and the data with age 0 increases the age to 1. Thus the resulting cache states are equal in every execution after two cache accesses.

Thereby only a hit in E and F after the first request where the needed data is contained in line with age 0 leads to a state, that is the same as before the hit. But it is obvious, that maximal two hit/miss-pairs are necessary to achieve the same cache state. At this point it is impossible to reach another hit/miss-pair. Thus $\Delta(CA, s, s') \leq 2$ and inversion series timing anomalies are not possible in 2-way LRU caches, see the best try in Figure 6.8. □

Theorem 6.4.4 *In 2-way associative LRU caches TA-S-A timing anomalies can occur.*



E

s_0			Ages	
	Accesses	hit/miss	0	1
			M_0	M_1
	M_0	hit	M_0	M_1
	M_1	hit	M_1	M_0
	M_2	miss	M_2	M_1
	M_0	miss	M_0	M_2

F

s_1			Ages	
	Accesses	hit/miss	0	1
			M_x	M_y
	M_0	miss	M_0	M_x
	M_1	miss	M_1	M_0
	M_2	miss	M_2	M_1
	M_0	miss	M_0	M_2

Figure 6.9: 2-way associative LRU, TA-S-A cache timing anomaly

Proof We know from the proof above that maximal two hit/miss-pairs are possible in these kind of caches. For an amplification series timing anomaly this is sufficient if two hits happen in one execution while at the same cache accesses two misses appear in another execution. An example is illustrated in Figure 6.9. □

6.5 Timing Anomalies in PLRU Caches

The functionality of PRLU caches is described in Section 4.4. In case of a cache update or a cache hit the tree state bits are modified after the cache request. The state bits, that are on the path to the accessed cache line are all set to point away from the path to the current cache line. In this kind of caches the state bits of the virtual tree belong to the initial state.

4-way associative PLRU Caches

In 4-way associative PLRU caches the virtual tree consists of three state bits. The state bits affect the decision which cache line is updated.

Theorem 6.5.1 *In 4-way associative PLRU caches TA-S-I timing anomalies can occur.*

E

s_0		State Bits			Content			
Accesses	hit/miss	t_0	t_1	t_2	L_0	L_1	L_2	L_3
		0	0	0	M_0	M_3	M_2	M_x
M_0	hit	1	1	0	M_0	M_3	M_2	M_x
M_1	miss	0	1	1	M_0	M_3	M_1	M_x
M_2	miss	1	0	1	M_0	M_2	M_1	M_x

F

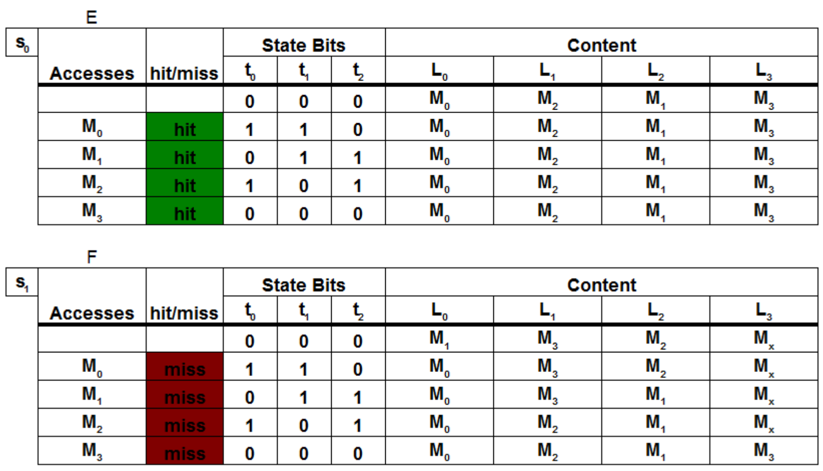
s_1		State Bits			Content			
Accesses	hit/miss	t_0	t_1	t_2	L_0	L_1	L_2	L_3
		0	0	0	M_x	M_2	M_1	M_3
M_0	miss	1	1	0	M_0	M_2	M_1	M_3
M_1	hit	0	1	1	M_0	M_2	M_1	M_3
M_2	hit	1	0	1	M_0	M_2	M_1	M_3

Figure 6.10: 4-way associative PLRU, TA-S-I cache timing anomaly

Proof In contrast to 2-way caches, in 4-way PLRU caches TA-S-I anomalies are possible. It is easy to obtain such an anomaly; assume two different executions, E with initial state s_0 and F with initial state s_1. Then let the first data access resulting in a cache hit in E, but the next two required data blocks are not included in the initial state s_0 of E or they are overwritten before the request. However the first needed data is instead contained in the initial cache state s_1, but the next two required data blocks are not in cache in s_1. Due to this, the execution F is faster than the execution E, whereas initially a miss occurs in F as opposed to an initial hit in E. Thus the conditions of a TA-S-I anomaly are fulfilled. An example is shown in Figure 6.10. □

Theorem 6.5.2 *In 4-way associative PLRU caches TA-S-A timing anomalies can occur.*

Proof Assume two different executions, E with initial state s_0 and F with initial state s_1. An anomaly is simple to generate, if two required cache blocks are in cache in the initial state s_0 (this results in two hits in E), but the same needed cache blocks are not in cache in s_1 or are overwritten before they were accessed and this leads to two misses in F. Thus the conditions of a TA-S-A timing anomaly are fulfilled. An example for a serial amplification timing anomaly is shown in Figure 6.11 □



E

s_0		State Bits			Content			
Accesses	hit/miss	t_0	t_1	t_2	L_0	L_1	L_2	L_3
		0	0	0	M_0	M_2	M_1	M_3
M_0	hit	1	1	0	M_0	M_2	M_1	M_3
M_1	hit	0	1	1	M_0	M_2	M_1	M_3
M_2	hit	1	0	1	M_0	M_2	M_1	M_3
M_3	hit	0	0	0	M_0	M_2	M_1	M_3

F

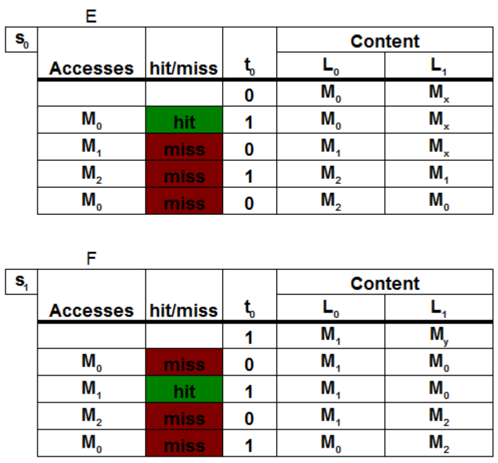
s_1		State Bits			Content			
Accesses	hit/miss	t_0	t_1	t_2	L_0	L_1	L_2	L_3
		0	0	0	M_1	M_3	M_2	M_x
M_0	miss	1	1	0	M_0	M_3	M_2	M_x
M_1	miss	0	1	1	M_0	M_3	M_1	M_x
M_2	miss	1	0	1	M_0	M_2	M_1	M_x
M_3	miss	0	0	0	M_0	M_2	M_1	M_3

Figure 6.11: 4-way associative PLRU, TA-S-A cache timing anomaly

2-way associative PLRU Caches

Theorem 6.5.3 *In 2-way associative PLRU caches TA-S-I timing anomalies can not occur.*

Proof Almost the same proof as for the non-existence of TA-S-I anomalies in 2-way LRU caches can be used. Assume two different executions, E with initial state s_0 and F with initial state s_1. To construct a serial inversion timing anomaly it is necessary that in the execution, where initially a hit takes place, say E with initial state s_0, the overall execution time is longer than of the execution, where initially a miss happens, say F with the initial state s_1. We know from above, that such an anomaly can only occur if at least at three cache accesses three hits



E

s_0			Content	
Accesses	hit/miss	t_0	L_0	L_1
		0	M_0	M_x
M_0	hit	1	M_0	M_x
M_1	miss	0	M_1	M_x
M_2	miss	1	M_2	M_1
M_0	miss	0	M_2	M_0

F

s_1			Content	
Accesses	hit/miss	t_0	L_0	L_1
		1	M_1	M_y
M_0	miss	0	M_1	M_0
M_1	hit	1	M_1	M_0
M_2	miss	0	M_1	M_2
M_0	miss	1	M_0	M_2

Figure 6.12: 2-way associative PLRU, best try to achieve a TA-S-I cache timing anomaly

appear in one execution in contrast to another execution where three miss appear. In this proof it will be shown, that this is impossible for 2-way PLRU caches.
After the first cache access at least one cache line has the same content and the state bit pointing away from that cache line in every execution. This is shown with the next four possibilities for two executions:

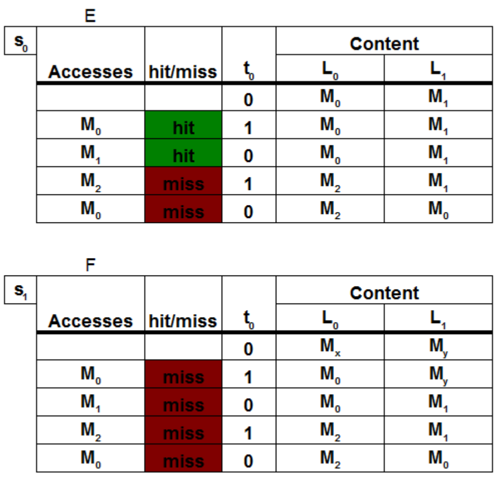
- hit in E and F: the required data is found in cache in both executions and according to the PLRU strategy the state bit is pointing away from that cache line after the access.
- miss in E and F: the needed data is not contained in cache in both executions. This data is put into a cache line, pointed by the bit, and after the access the state bit points away to the other line of the associated set.
- hit in E and miss in F: the required block was in the cache in E and is inserted in a cache line in F. In both executions the state bit points away from that data block after the cache access.
- miss in E and hit in F: the required block was in the cache in F and is inserted in a cache line in E. In both executions the state bit points away from that data block after the memory access.

Note that after the first cache query one cache line in E and F is equal. Assume that these equal cache lines are named CL_{E_E} in E and CL_{E_F} in F. The cache lines that are potentially not equal are called CL_{PNE_E} in E and CL_{PNE_F} in F. Also the state bit points away from CL_{E_E} (CL_{E_F}) to CL_{PNE_E} (CL_{PNE_E}). For the next requests also four cases are possible:

- hit in E and F: there are also two cases; first this access requires data from CL_{E_E} and CL_{E_F}. Then in both executions the content of the caches do not change and the state bits refer to CL_{PNE_E} and CL_{PNE_F}. Second this cache access does not make use of the same data as the first access. In this case the cache contents CL_{PNE_E} and CL_{PNE_F} are equal in both executions and after the final memory access the state bit points to the same element in both executions.
- miss in E and F: after this access the required data is included in CL_{PNE_E} and CL_{PNE_F} and both state bits refer to CL_{E_E} and CL_{E_F}. Thus the cache content is similar in both executions and both state bits refer to the same line.
- hit in E and miss in F: the new data must be put into CL_{PNE_F} in F and it was contained in CL_{PNE_E} in E. Then also both executions have the same content and both state bits pointing to CL_{E_E} respectively CL_{E_F} after this access.
- miss in E and hit in F: the new data must be put into CL_{PNE_E} in E and it was contained in CL_{PNE_F} in F. Then also both executions have the same content and both state bits pointing to CL_{E_E} respectively CL_{E_F} after this access.

Thereby only a hit in E and F after the first request, where the needed data is contained in CL_{E_E} and CL_{E_F} leads to a state that is the same as before the hit. But it is obvious, that maximal two hits at two cache accesses in one execution in contrast to two misses at the same two accesses in another execution leads to the same cache state afterwards. If the cache content is equal and the state bit points to the same cache line in two executions at a specific point in time then it is impossible to observe a different access behaviour related to cache hits or misses. Thus inversion series timing anomalies are not possible in 2-way PLRU caches, see the best try in Figure 6.12. □

Theorem 6.5.4 *In 2-way associative PLRU caches TA-S-A timing anomalies can occur.*


E

s_0				Content	
	Accesses	hit/miss	t_0	L_0	L_1
			0	M_0	M_1
	M_0	hit	1	M_0	M_1
	M_1	hit	0	M_0	M_1
	M_2	miss	1	M_2	M_1
	M_0	miss	0	M_2	M_0

F

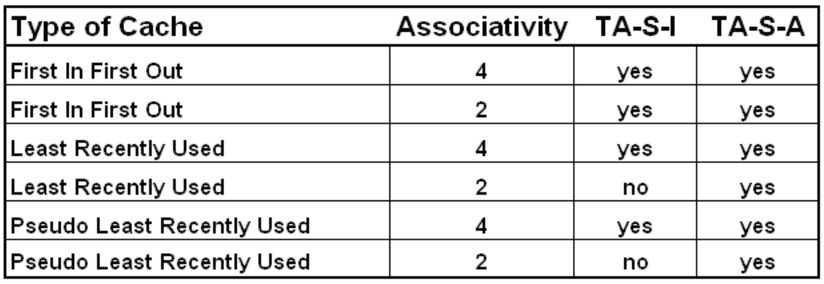
s_1				Content	
	Accesses	hit/miss	t_0	L_0	L_1
			0	M_x	M_y
	M_0	miss	1	M_0	M_y
	M_1	miss	0	M_0	M_1
	M_2	miss	1	M_2	M_1
	M_0	miss	0	M_2	M_0

Figure 6.13: 2-way associative PLRU, TA-S-A cache timing anomaly

Proof For amplification series timing anomalies at most two hits at two cache accesses in one execution in contrast to two misses at the same two accesses in another execution are possible. Assume two different executions, E with initial state s_0 and F with initial state s_1. Then a TA-S-A anomaly can be achieved, if in the initial state s_0 the data blocks that are required in the first two memory accesses are in the cache, but these are not in the cache content of s_1. Due to the fact that in F two more misses happen than in E, a TA-S-A timing anomaly is observed, see Figure 6.13. □

6.6 Summary

In Table 6.1 an overview of the timing anomalies, that can occur on cache architectures, is given. After the analysis of cache timing anomalies, one can observe that in 2-way LRU and PLRU inversion series timing anomalies can not occur, because after two two hits at two cache accesses in one execution in contrast to two misses at the same two accesses in another execution the same cache state is reached in every execution. Additional one can see that timing anomalies in 8-way associative caches can appear if they are possible in 4-way associative caches.



Type of Cache	Associativity	TA-S-I	TA-S-A
First In First Out	4	yes	yes
First In First Out	2	yes	yes
Least Recently Used	4	yes	yes
Least Recently Used	2	no	yes
Pseudo Least Recently Used	4	yes	yes
Pseudo Least Recently Used	2	no	yes

Table 6.1: Summary of Timing Anomalies on Caches

CHAPTER 7

Parallel Timing Anomalies

7.1 Overview

In the last two chapters series timing anomalies both at pipelines and caches have been analysed. In this chapter a compound of these architectures and the resulting parallel timing anomalies are treated. In Section 2.10 these kind of anomalies have been presented and defined. In the analysis of parallel anomalies a sequence of instructions is no longer split into individual instructions, but the sum of whole instruction sequences are discussed. Parallel timing anomalies are assessed on the difference of the execution times of an instruction sequence at a hardware component hw_A and the difference of the sum of the overall execution times for an instruction sequence for two hardware components.

7.2 Analysis of Parallel Timing Anomalies

Remember first the definitions of parallel timing anomalies:

- **TA-P-I:**
  $\exists a, a' \in A, \exists b \in B.$
  $\Delta_{hw_A}(I, a, a') > 0 \ \wedge \ \Delta(I, \langle a, b\rangle, \langle a', b\rangle) < 0$
- **TA-P-A:**
  $\exists a, a' \in A, \exists b \in B.$
  $0 < \Delta_{hw_A}(I, a, a') < \Delta(I, \langle a, b\rangle, \langle a', b\rangle)$

In case of parallel timing anomalies it is necessary to observe the interaction between two hardware components, the mutual interference and the effects. In the analysis two different initial states a, a' of hardware component hw_A and only one initial state b of hw_B, which includes the impact of the previous execution at hw_A, are consulted.

If we assume that the cache of a processor represents the hardware component hw_A and the specific pipeline expressed the hardware component hw_B, then it is useful to observe the interference from the cache behaviour to the pipeline. That means that differences of cache accesses in a particular instruction can result in a variation of the duration of this instruction in the pipeline (achieved by additional misses in the cache). Thus the analysis of parallel timing anomalies with cache and pipeline architectures requires the knowledge of the possibility that the cache behaviour in two different executions can lead to a situation where the timing of an instruction in the pipeline permits a timing anomaly.

Theorem 7.2.1 *Parallel timing anomalies, where hw_A represents the cache of a processor and hw_B the pipeline, can occur in combination of 2- and 4-way associative FIFO, LRU and PLRU replacement strategies with the same pipeline architectures, in which series anomalies can appear.*

Proof From Chapter 6 it is evident, that in every presented cache replacement strategy at least two hits can happen in one execution while at the same cache accesses two misses can appear in another execution and thereby at least two additional misses are feasible. Additionally we know from Chapter 5 that if a series timing anomaly in a pipeline can occur, then this is achievable with at most a difference of two clock cycles in the first instruction (in this thesis I_0). As a result of these two facts we can summarize, that a parallel timing anomaly can occur, if it also can appear in the series anomaly perspective at pipelines combined with every presented cache replacement strategy. $\square$

Thus the Table 5.1 of series timing anomalies at pipelines can also be applied for parallel timing anomalies, where the combination of caches and pipelines are observed. Note that this result originates from a completely different approach, nevertheless this leads to the same consequence than in the study of series timing anomalies.

CHAPTER 8

Conclusion

The presence of timing anomalies is very important for the Worst-Case Execution Time calculation. Thus it is essential to perform a precise analysis of timing anomalies. The subject of this thesis was to find the minimal preconditions for the occurrence of series respectively parallel timing anomalies and to investigate them. A short characterisation about RTS has been done. Also some basic concepts, few definitions and important notations have been presented, which are needed for the analysis of timing anomalies. Also the techniques 'series composition' (for series timing anomalies), 'delta composition' and 'max composition' (for parallel timing anomalies) have been displayed, which are mainly taken from [6]. Then the basic concept of pipeline architecture have been demonstrated and the differences and characterization of simple scalar-, scalar-, superscalar in-order- and superscalar out-of-order-pipelines have been done. Next the principle of cache architectures and the functionality of the cache replacement strategies FIFO, least recently used and pseudo least recently used have been explained.
Then pipeline architectures have been analysed with respect to series timing anomalies. The definitions and findings are mainly taken from [17], but in that thesis the segmentation between series and parallel timing anomalies is not elaborated. The *minimal preconditions* for the occurrence in the *presented pipeline architectures of series timing anomalies are minimal overlapping function units or/and out-of-order execution.*
Some adapted definitions of series timing anomalies in cache architectures have been implemented. Based on that definitions a detailed analysis was possible and thus the existence of *series inversion timing anomalies in 2-way associative LRU respectively PLRU cache architectures can be excluded.* In *all other presented cache architectures the existence of inversion and amplification series timing anomalies has been demonstrated.*
In the analysis of parallel timing anomalies the knowledge of the occurrence of timing anomalies in the presented pipeline architectures are linked with the knowledge of the occurrence of timing anomalies in the presented cache architectures. *The result* of this analysis *for parallel timing anomalies is the same as for series anomalies in pipeline architectures in combination with all presented cache architectures.*
Thus the analysis of the occurrence of timing anomalies in the presented pipeline and cache

architectures is finalised. With the result of this thesis it is possible to categorize a specific processor by reference to the potential series and parallel timing anomalies and a WCET bound can be given.

Future Research

The occurrence of timing anomalies in other pipeline and cache architectures can be analysed. Also more precise techniques for calculating a WCET bound can explored.
The investigation of replacement strategies for caches that the occurrence of timing anomalies forbid can be helpful, because in all presented replacement strategies timing anomalies can appear in 4-way (and of course in 8-way) associative caches.
Maybe it is achievable to program tools that provide information about the potential timing anomalies if specific instructions are given.
Finally discussions of developing processors that have predictable timing behaviour have to be done.

Keywords

Series Timing Anomalies, Parallel Timing Anomalies, Worst-Case Execution Time, WCET Analysis, Pipeline Architectures, Cache Architectures

Bibliography

[1] Christoph Berg. Plru cache domino effects. In Frank Mueller, editor, *6th Intl. Workshop on Worst-Case Execution Time (WCET) Analysis*, Dagstuhl, Germany, 2006. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[2] Christopher A. Healy. Integrating the timing analysis of pipelining and instruction caching. In *In IEEE Real-Time Systems Symposium*, pages 288–297, 1995.

[3] Christopher A. Healy, Robert D. Arnold, Frank Mueller, David B. Whalley, and Marion G. Harmon. Bounding pipeline and instruction cache performance. *IEEE Transactions on Computers*, 48:53–70, 1999.

[4] Reinhold Heckmann and Christian Ferdinand. Worst-case execution time prediction by static program analysis. In *In 18th International Parallel and Distributed Processing Symposium (IPDPS 2004*, pages 26–30. IEEE Computer Society.

[5] Raimund Kirner, Albrecht Kadlec, and Peter Puschner. Precise worst-case execution time analysis for processors with timing anomalies. In *ECRTS '09: Proceedings of the 2009 21st Euromicro Conference on Real-Time Systems*, pages 119–128, Washington, DC, USA, 2009. IEEE Computer Society.

[6] Raimund Kirner, Albrecht Kadlec, and Peter Puschner. Worst-case execution time analysis for processors showing timing anomalies. Research Report 01/2009, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2009.

[7] Benjamin Lesage, Damien Hardy, and Isabelle Puaut. Wcet analysis of multi-level set-associative data caches. In Niklas Holsti, editor, *9th Intl. Workshop on Worst-Case Execution Time (WCET) Analysis*, Dagstuhl, Germany, 2009. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Germany. also published in print by Austrian Computer Society (OCG) with ISBN 978-3-85403-252-6.

[8] Thomas Lundqvist. A wcet analysis method for pipelined microprocessors with cache memories. Technical report, 2002.

[9] Thomas Lundqvist and Per Stenström. Timing anomalies in dynamically scheduled microprocessors. In *Proceedings of the 20th IEEE Real-Time Systems Symposium*, RTSS '99, pages 12–, Washington, DC, USA, 1999. IEEE Computer Society.

[10] Jan Reineke. *Caches in WCET Analysis*. PhD thesis, Universität des Saarlandes, November 2008.

[11] Jan Reineke, Daniel Grund, Christoph Berg, and Reinhard Wilhelm. Predictability of cache replacement policies. In *Reports of SFB/TR 14 AVACS 9, SFB/TR 14 AVACS*, 2006.

[12] Jan Reineke and Rathijit Sen. Sound and efficient wcet analysis in the presence of timing anomalies. In Niklas Holsti, editor, *9th Intl. Workshop on Worst-Case Execution Time (WCET) Analysis*, Dagstuhl, Germany, 2009. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Germany. also published in print by Austrian Computer Society (OCG) with ISBN 978-3-85403-252-6.

[13] Jan Reineke, Björn Wachter, Stephan Thesing, Reinhard Wilhelm, Ilia Polian, Jochen Eisinger, and Bernd Becker. A definition and classification of timing anomalies. In *6th Intl. Workshop on Worst-Case Execution Time (WCET) Analysis, Dagstuhl, Germany, 2006. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl*.

[14] Christine Rochange and Pascal Sainrat. Difficulties in Computing the WCET for Processors with Speculative Execution . In *2nd Intl. Workshop on Worst Case Execution Time Analysis , Vienne, 18/06/02*, pages 68–71. University of York, juin 2002.

[15] Henrik Theiling, Christian Ferdinand, and Reinhard Wilhelm. Fast and precise wcet prediction by separated cache and path analyses. *Real-Time Syst.*, 18:157–179, May 2000.

[16] Stephan Thesing. *Safe and Precise WCET Determination by Abstract Interpretation of Pipeline Models*. PhD thesis, Universität des Saarlandes, 2004.

[17] Ingomar Wenzel. Principles of timing anomalies in superscalar processors. Master's thesis, Technische Universität Wien, Institut für Technische Informatik, 2003.

[18] Ingomar Wenzel, Raimund Kirner, Peter P. Puschner, and Bernhard Rieder. Principles of timing anomalies in superscalar processors. In *QSIC*, pages 295–306, 2005.

[19] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Muller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS)*, April 2008.