Deliverable D2: Memory Management Unit block level design

Sistemi di Elaborazione a.a. 2018/2019

Group M

Produced by	Date of Approval	Approved by	Version
Antonino Durazzo, Giuseppe Mascolo	8 Dic. 2018	Francesco de Pertis, Edoardo Cossentino, Fernando Manna	1.3

Deliverable D2: Memory Management Unit block level design by group M

Release Information

The following changes have been made to this document.

Table 1 Change History

Date	Revision	Authors	Change Details
03 November 2018	0	Antonino Durazzo	First complete draft
5 Dicember 2018	1	Antonino Durazzo	Clock signal sent only to the memory components D-MMU, I_MMU, Cache L2, Main Memory. Update bus in input and output of buffer between L1 and L2. Define two new functional block in core MMU, Miss Definer and Buffer Stall block.
8 Dicember 2018	2	Antonino Durazzo	Structural revision
15 Dicember 2018	3	Antonino Durazzo	Review documentation

Summary

List of Tables

Deliverable D2: Memory Management Unit block level design

List of Figures

Deliverable D2: Memory Management Unit block level design

List of Acronyms

Deliverable d2: top level architecture

L1	Level 1
L2	Level 2
MEM	Memory
MM	Main Menu
MMU	Memory Management Unit
PC	Program Counter
TLB	Translation Lookaside Buffer
VIPT	Virtually Indexed Physically Tagged
FSM	Finite State Machine
VM	Virtual Memory
WB	Write Back
WT	Write Through

Preface

This preface introduces the Deliverable d2: top level architecture (Revision 1). It contains the following sections:

About this manual

The purpose of this manual is to describe a detailed design implementation based on ARM9TDMI, a 32-bit Harvard architecture with 5-stage pipeline.

The main features being implemented are:

- 2nd level Cache for both Instructions and Data
- Virtual Memory
- Memory Management Unit

A combined use of these features allows to avoid some issues regarding performances and opens to new solutions such as Speculative Execution, Instruction-level Parallelism or in-execution Forwarding.

Main purpose of this design is to keep a good trade-off among cost, complexity and performance; metrics adopted follow the idea of saving resources in absence of significant performances improvements.

Intended audience

This manual is written for experienced hardware and software engineers who might or might not have experience of ARM products.

Using this manual

The information in this manual is organized into three chapters, as described below.

Chapter 1: Funtional Block

Chapter 2 describes how work every functional block that is present in the top level design

References

ARM publications

- ARM Architecture Reference Manual (ARM DDI 0100I)
- ARM9TDMI Technical Reference Manual (ARM DDI 0180A)

Other sources

- Computer Organization and Design (3rd edition) - David A. Patterson, John L. Hennessy
- Lecture notes of the course of Sistemi di elaborazione a.y. 2018/2019 – Prof. Angelo Marcelli

Chapter 1: Functional Blocks

This chapter introduces the functional blocks specific implementation and contains the following sections:

1 Miss Handler

The Miss Handler deals with the miss that occurs when accessing the data or instruction in the first level cache.

Figure 1.1 Miss Handler

The function of this functional block is to determine where the block that raised the miss (cache L2 or main memory) is stored and to communicate it to the Replacement Block.

If the block is present in the level 2 cache (hit in L2 cache), the Miss Handler tells the Replacement Block to load the block in the level 1 cache. If the block is not present in the level 2 cache (miss in L2 cache), it follows without doubt that it is in the main memory. In this case, the Miss Handler asks the Replacement Block to load the block from the central memory, both in the first and the second level cache, since the L2 cache is inclusive, unlike the L1 cache.

Whether a miss occurs in the first level cache, the Miss Handler is expected to report it to the CPU, so that the pipeline can be properly managed.

If the miss event has been generated by the I-mmu, the miss signal is sent straight to the CPU; whereas, if the miss has been generated by the D-mmu, the miss signal is sent to the Miss definer block, which takes charge of sending it to the CPU.

Name	Bits	Function
I_Miss	1	This signal comes from the I-MMU when occurs a miss in the istruction cache.
D_Miss	1	This signal comes from the D-MMU when occurs a miss in the data cache.
L2_Miss	1	This signal comes from the cache L2 when, trying searching a block, occurs a miss.
L2_Hit	1	This signal comes from the cache L2 when, trying searching a block, occurs a hit.
End_Replace	1	This signal comes from the Replacement Block when, following a miss, the replace operation is terminated.

Table 1 Inputs of MissHandler

Name	Bits	Function
I_Miss	1	This signal is propagated to the CPU when occurs a miss in the istruction cache. The signal remains asserted until the end of the management of the miss.
D_Miss	1	This signal is propagated to the Miss Definer when occurs a miss in the data cache. The signal remains asserted until the end of the management of the miss.
L2_Read	1	This signal enables reading from the cache L2.
Replace	1	This signal enables the Replacement Block.
Found_Block	1	This signal communicates to the Replacement Block the position of the requested block. The signal is asserted if the block is in the cache L2; it is not asserted if the block is in the main memory. This signal is consistent only if the signal Replace is asserted.

Table 2 Outputs of Miss Handler

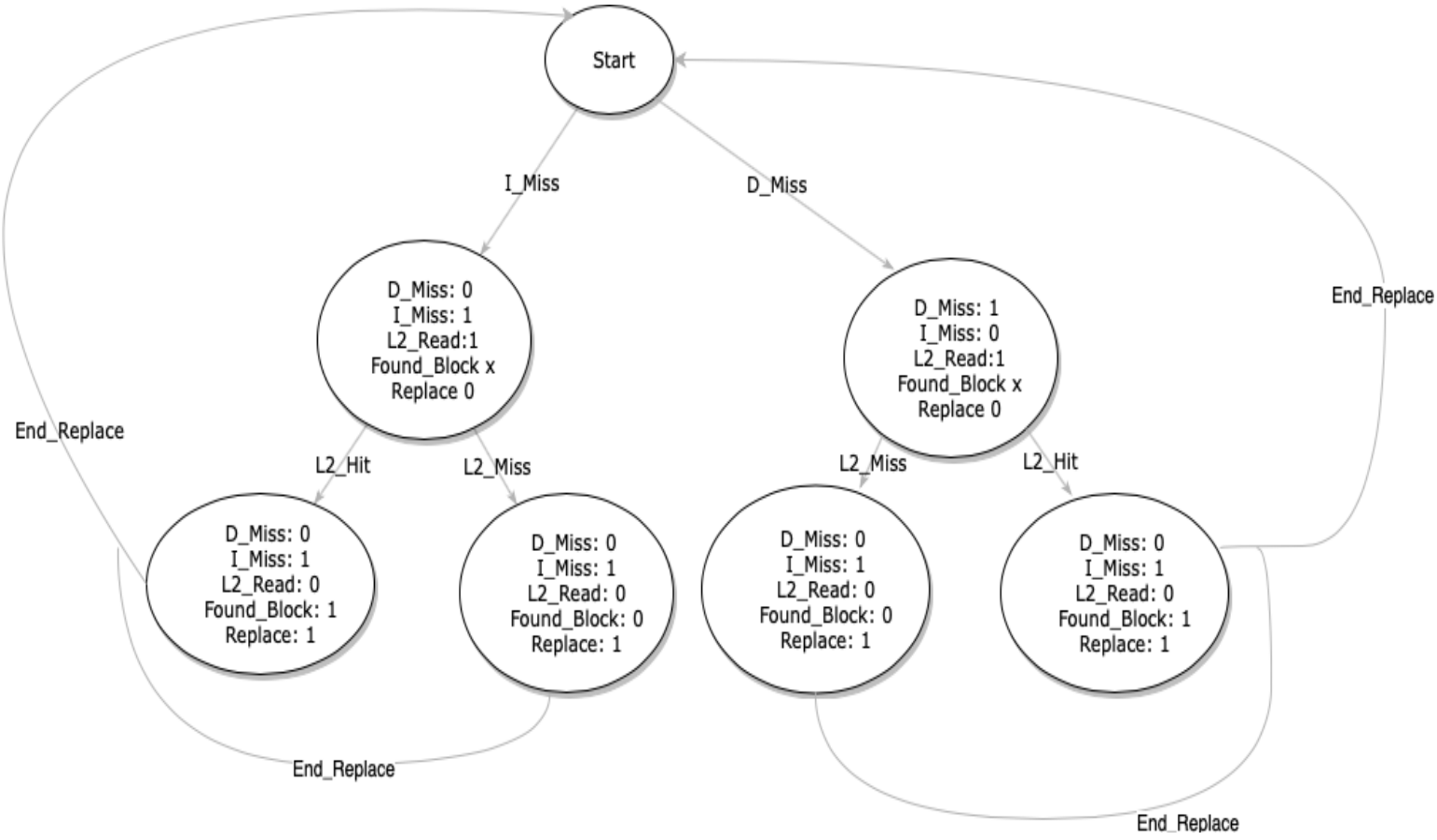

Figura 1.2 FSM Miss Handler

1.1 Miss Definer

The Miss Definer is responsible for generating the miss signal which is sent to the CPU, in case of miss in data cache.

Figura 1.3 Miss Definer

This block is needed to be included in order to properly distinguish whether the miss in data cache has occurred because of a multiple load/store or in the MEM Stage. To this purpose, it takes in input the D_Miss signal and the signal from the CPU that indicates whether the read / write request comes from a multiple load / store or not; whereas, the output signals are the D_Miss_Single and D_Miss_Multiple, which are needed to be propagated to the CPU.

If both the inputs are asserted, the combination of the output is:

- D_Miss_Single = 0 and D_Miss_Multiple = 1.

If the signal indicating a multi-memory operation is desserted, the output is:

- D_Miss_Multiple = 0 and D_Miss_Single = D_Miss.

Name	Bits	Function
D_Miss	1	This signal comes from the Miss Handler when occurs a miss in the data cache.
Is_Multiple	1	This signal comes from the CPU when the writing/reading request comes from a multiple load/store instruction.

Table 3 Inputs of Miss Definer

Name	Bits	Function
D_Miss_Single	1	This signal is propagated to the CPU when the signal D_Miss is asserted and the singal Is_Multiple is not asserted.
D_Miss_Multiple	1	This signal is propagated to the CPU when the signals D_Miss and Is_Multiple are asserted.

Table 4 Outputs of Miss Definer

2 Replacement Block

The Replacement Block manages the transfer of blocks when the first level cache (data or instructions) raises a miss.

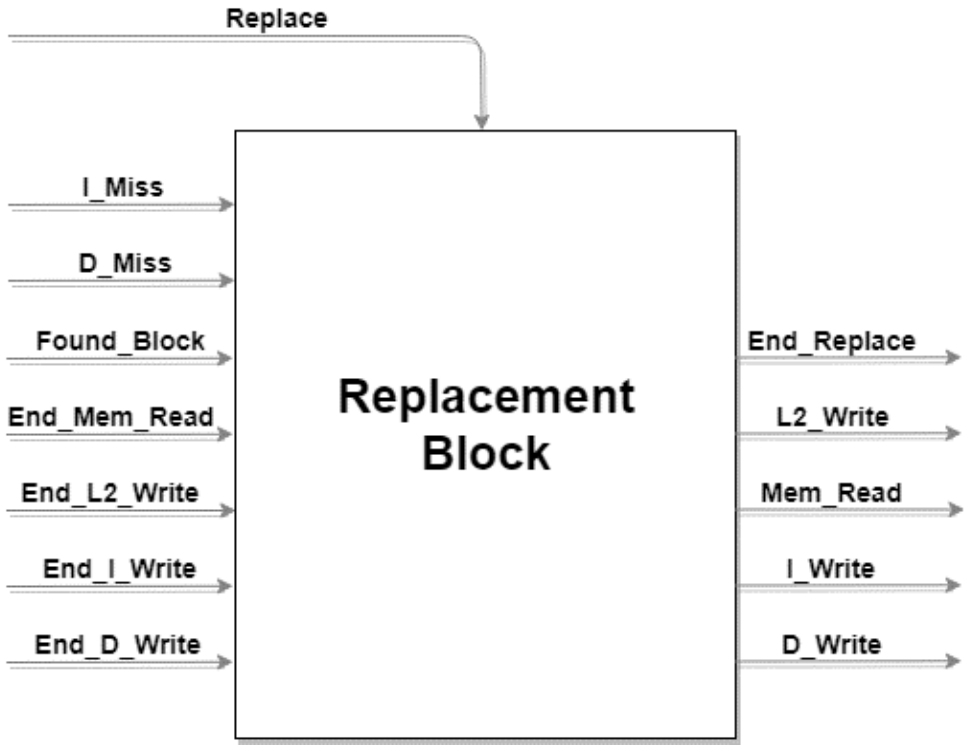

Figura 1.4 Replacement Block

The Miss Handler activates the Replacement Block when a block must be transferred between the levels of the memory hierarchy (triggered by the signal Replace) and communicates to it which level of the hierarchy the block is stored in (second level cache or main memory).

If the block is in the second level cache, the Replacement Block enables the writing of the requested block in the L2 cache.

If the block is in the main memory, the Replacement Block enables the main memory to send the data on the bus and the writing in both the two levels of the cache.

The completion of the block writing operations is notified to the Miss Handler with the signal “End_Replace”.

Name	Bits	Function
Replace	1	This signal comes from the Miss Handler in order to enable the Replacement block in case of miss in the cache L1(instruction or data).
I_Miss	1	This signal comes from the Miss Handler in case of miss in the instruction cache.
D_Miss	1	This signal comes from the Miss Handler in case of miss in the data cache.
Found_Block	1	This signal comes from the Miss Handler and is consistent only when the signal Replace is asserted. If Found_Block is asserted the requested block is in the cache L2, if is not asserted the block is in the main memory.
End_I_Write	1	This signal comes from the instruction cache and indicates the end of the block write operation.
End_D_Write	1	This signal comes from the data cache and indicates the end of the block write operation.
End_L2_Write	1	This signal comes from the cache L2 and indicates the end of the block write operation.
End_Mem_Read	1	This signal comes from the main memory and indicates the end of the read operation of the searched block.

Table 5 Inputs of Replacement Block

Name	Bits	Function
End_Replace	1	This signal informs the Miss Handler of the end of handling the replace operation.
I_Write	1	This signal enables writing to the instruction cache.
D_Write	1	This signal enables writing to the data cache.
L2_Write	1	This signal enables writing to the cache L2.
Mem_Read	1	This signal enables the main memory to present the data on the bus.

Table 6 Outputs of Replacement Block

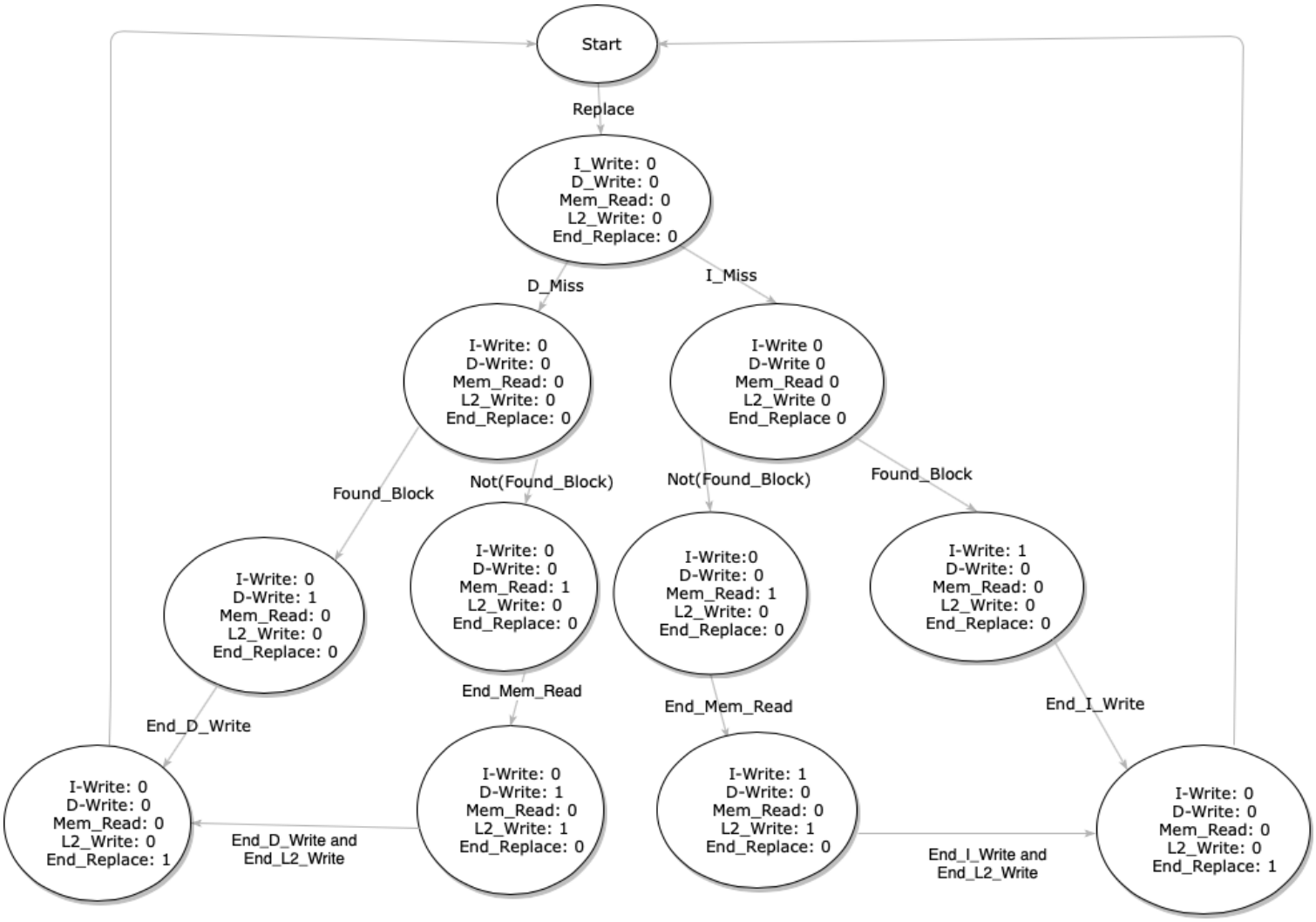

Figura 1.5 FSM Replacement Block

3 Coherence Block

The Coherence Block is expected to properly manage the coherence between the different levels of the memory hierarchy.

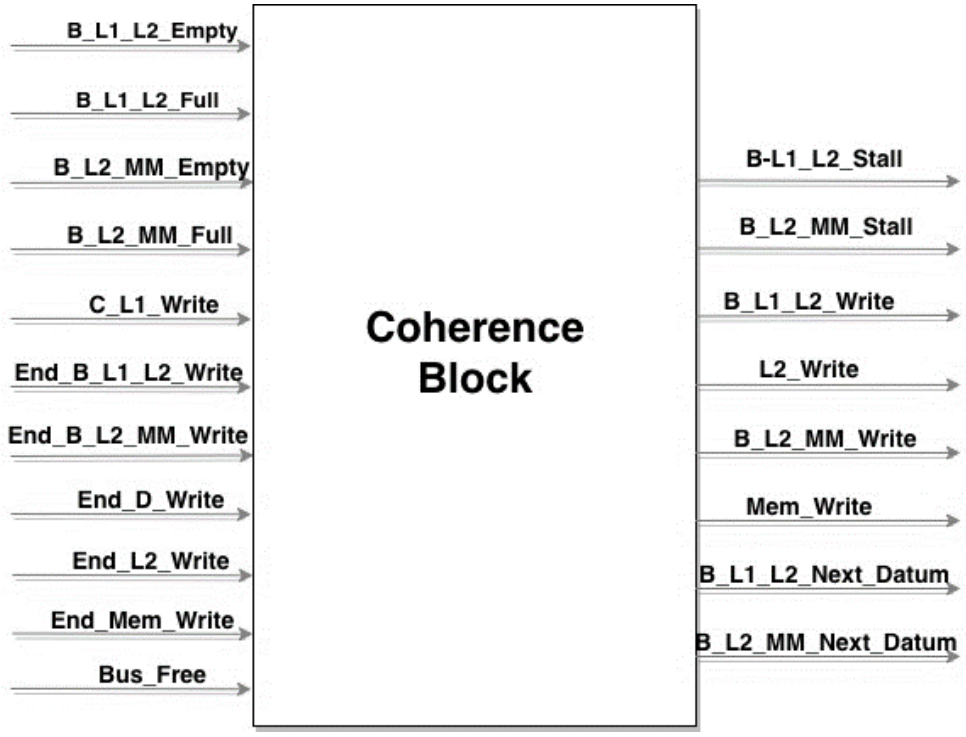

Figura 1.6 Coherence Block

In depth, this functional block manages the writing and emptying of the buffers, in order to ensure the consistency of data between cache L1(data or instruction), cache L2 and main memory.

This block is activated when the writing in data cache is required; to this purpose, the writing of the buffer between the data cache and the cache L2 is enabled, in order to write in parallel both in the cache and in the buffer. Then, when the writing in cache L2 is required, the Coherence Block enables this operaton in parallel with the writing on the buffer between the cache L2 and the main memory.

While performing the operations mentioned above, as far as it can, the Coherence Block also transfers the data contained in the buffers into the respective underlying memory components, in order to restore the consistency between the levels of the hierarchy. This operation takes place through the data bus and the address bus, used if free. This decision was taken to avoid that management operations of coherence divert resources to other components, which may have an higher priority. Furthermore, it is necessary to note that if both buffers are not empty, the highest priority is given to emptying the buffer between data cache and L2 cache.

When a buffer is full, the corresponding signal is asserted and propagated to Buffer Stall Definer, so that the unit can take charge of communicating this occurrence to the CPU, in order to properly handle the pipeline, until the buffer is completely emptied and the signal deasserted.

Name	Bits	Function
B_L1-L2_Empty	1	This signal communicates that the buffer between cache L1 and cache L2 is empty.
B_L1-L2_Full	1	This signal communicates that the buffer between cache L1 and cache L2 is full.
B_L2-MM_Empty	1	This signal communicates that the buffer between cache L2 and main memory is empty.
B_L2-MM_Full	1	This signal communicates that the buffer between cache L2 and main memory is full.
C_L1_Write	1	This signal indicates a request to write to data cache.
End_ B_L1-L2_Write	1	This signal communicates the end of the write operation in the buffer between cache L1 and cache L2.
End_ B_L2-MM_Write	1	This signal communicates the end of the write operation in the buffer between cache L2 and main memory.
End_D_Write	1	This signal communicates the end of the write operation of a block in data cache.
End_L2_Write	1	This signal communicates the end of the write operation of a block in cache L2.
End_Mem_Write	1	This signal communicates the end of the write operation of a block in main memory.
Bus_Free	1	This signal communicates that the bus is unused.

Table 7 Inputs of Coherence Block

Name	Bits	Function
B_L1-L2_Stall		This signal is propagated to the Buffer Stall Definer when the buffer between cache L1 and cache L2 is full. The signal is deasserted when the flush of the buffer is completed.
B_L2-MM_Stall		This signal is propagated to the Buffer Stall Definer when the buffer between cache L2 and main memory is full. The signal is deasserted when the flush of the buffer is completed.
B_L1-L2_Write	1	This signal enables the buffer between the data cache and the cache L2 to read the data on the input data bus.
B_L2-MM_Write	1	This signal enables the buffer between the cache L2 and the main memory to read the data on the input data bus.
L2_Write	1	This signal enables writing to the cache L2.
MM_Write	1	This signal enables writing to the main memory.
B_L1-L2_Next_Datum	1	This signal enables the buffer between cache L1 and cache L2 to present the next block on the bus.

B_L2-MM_ Next_Datum	1	This signal enables the buffer between cache L2 and main memory to present the next block on the bus.

Table 8 Outputs of Coherence Block

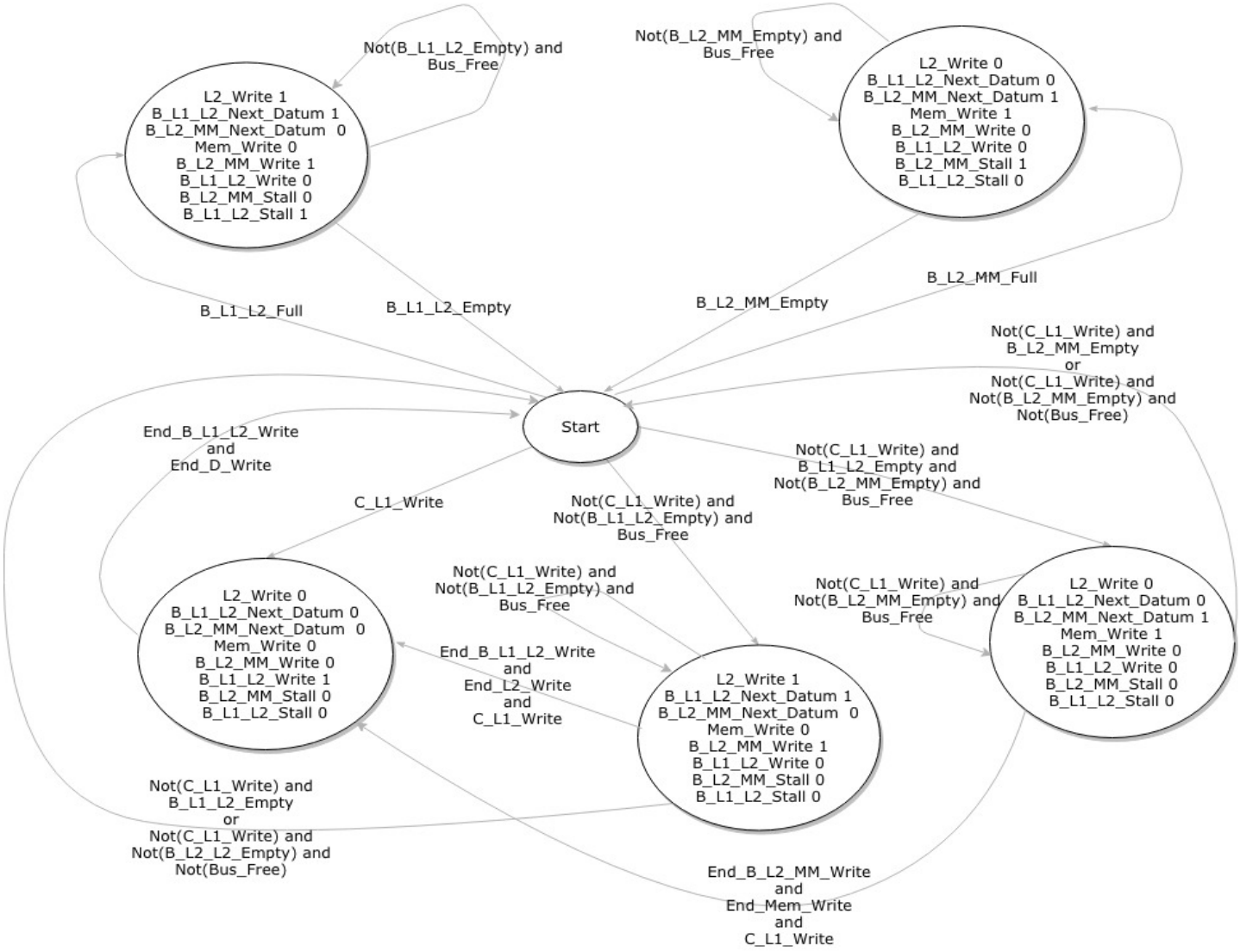

Figura 1.7 FSM Coherence Block

3.1 Buffer Stall Definer

This functional block defines the signals that stalls the CPU in the case the buffer is full.

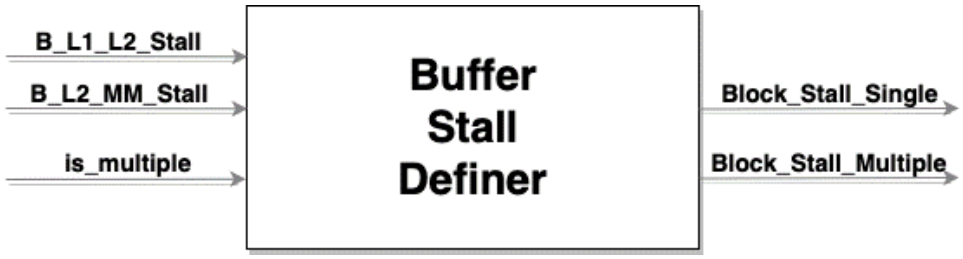

Figura 1.8 Buffer Stall Block

It was necessary to define this block to distinguish whether the buffer saturation occurs during a multiple load / store or during the MEM Stage.

The block receives in input the signal that identifies if the MMU is serving a write/read request coming from a write/read multiple and the signals that identify the saturation of the buffer placed between the data cache and the l2 cache or the buffer saturation placed between the L2 cache and the main memory.

In output, the block generates the Block_Stall_Multiple signal, which indicates whether the buffer saturation occurred during a single write, and the Block_Stall_Multiple signal, which indicates whether the buffer saturation occurred during a multiple write / read operation.

Both signals stall the pipeline stages that can not continue processing until they are asserted.

If one of the signals related to the saturation of the buffers is asserted, the combination of the outputs will be as follows:

- Is_Multiple = 1, Block_Stall_Multiple = 1 Block_Stall_Single = 0
- Is_Multiple = 0, Block_Stall_Multiple = 0, Block_Stall_Single = 1

Name	Bits	Function
Buffer_Stall_Single	1	This signal is propagated to the CPU when the signal L1_L2_Buffer_Stall or the signal L2_MM_Buffer_Stall is asserted and the signal Is_Multiple is not asserted.
Buffer_Stall_Multiple	1	This signal is propagated to the CPU when the signal L1_L2_Buffer_Stall or the signal L2_MM_Buffer_Stall is asserted and the signal Is_Multiple is asserted.

Table 9 Inputs of Buffer Stall Definer

Name	Bits	Function
L1_L2_Buffer_Stall	1	This signal comes from the Coherence Block when the buffer between cache L1 and cache L2 is full.
L2_MM_Buffer_Stall	1	This signal comes from the Coherence Block when the buffer between cache L2 and main memory is full.
Is_Multiple	1	This signal comes from the CPU when the writing/reading request comes from a multiple load/store instruction.

Table 10 Outputs of Buffer Stall Definer

3.2 Buffer Sizing

The size of the buffer placed between the level 1 cache and the level 2 cache has been set to 4 entries, after analyzing several real systems, to gain a reasonable trade-off between buffer's dimension and risk for buffer's saturation.

The size of the buffer placed between the level 2 cache and main memory has been set to 8 entries considering that, for the architecture presented, when the buffer placed between L1 cache and L2 cache it's full its content is written both in the level 2 cache and both buffer placed between the L2 cache and the main memory, thus, if the size would be the same when the first level buffer becomes full then the second level buffer will become full.

The entries of the two buffers have the same dimension.

Each entry can contain up to 32 bits of data and a 32-bit address.

4 Translation Block

This block runs the translation of virtual addresses, generated by the CPU, in physical addresses. The architecture submitted owns two primary level cache virtually addressed and physically tagged: one for data and one for instruction.

The virtual addressing provides the use of the physical address to determine if occur an hit, so the access to TLB must be the fastest possible. For this reason was chosen to have two TLB: data TLB, instruction TLB. Therefore two translation block have been allocated (data and instruction).

When a datum, sent by CPU, must be translated, in the event that the requested entry it's not present in the TLB, the Translation Block asserts the miss appropriate signal and propagates it to the CPU, so that is started via software management of the relative exception. If, instead, the entry is presente and it has validity bit set to 1, the physical address translated of the datum in cache is presented on physical address bus. Can be happen that the entry is present in the TLB, but correspondent validity bit is set to zero, in this case page fault signal is asserted and is propagated to CPU so that software management of the related exception is started.

The Translation Block flushes the TLB unsetting all the validity bit, in every context.

The architecture submitted provides for a system to the addressing space protection, in order to allow, to different process, to share informations, and at same time to avoid any illegal access. For this reason was introduced, for every page's table entry, the writing access bit. Such bit, if asserted, means that the page can be only read but non written. If a process try to write in a data cache a page with access bit in writing equals to 1, the Data Translation Block asserts the illegal access signal and propagate it to CPU so that is started via software management of the relative exception. The event just described cannot be happen in instruction cache because you access it only in reading modality unlike the data cache to which you can be access also in writing modality.

It was necessary distinguish the Instruction Translation Block and Data Translation Block.

4.1 Data Translation Block

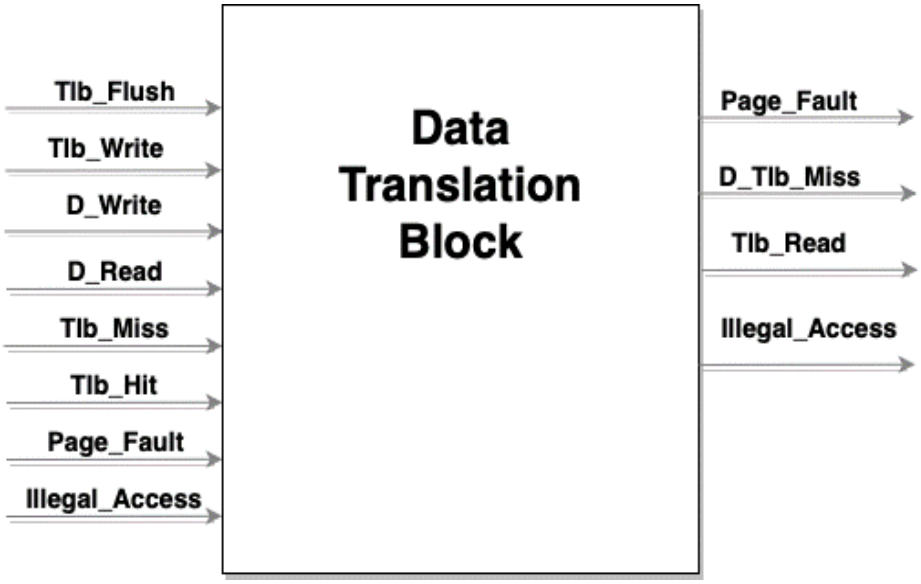

Figura 1.9 Data Translation Block

Name	Type	Bits	Function
TLB_Flush	Control	1	This signal is asserted when it is necessary to flush the data-TLB in case of context switch.
TLB_Write	Control	1	This signal enables writing to the data-TLB.
D_Write	Control	1	This signal comes from the CPU to enable the translation when there is a writing request.
D_Read	Control	1	This signal comes from the CPU to enable the translation when there is a reading request.
TLB_Miss	Control	1	This signal comes from the data-TLB when occurs a miss.
TLB_Hit	Control	1	This signal comes from the data-TLB when occurs a hit.
Page_Fault	Control	1	This signal comes from the data-TLB when occurs a page fault (the validity bit of the entry is set to 0).

Table 11 Inputs of Data Translation block

Name	Type	Bits	Function
Page_Fault	Control	1	This signal is propagated to the CPU when there is a page fault in order to manage the exception.
D_TLB_Miss	Control	1	This signal is propagated to the CPU when occurs a miss in the data-TLB in order to stall the pipeline.
TLB_Read	Control	1	This signal enables reading from the data-TLB in order to search the requested entry.
Illegal_Access	Control	1	This signal is propagated to the CPU when there is an illegal access to writing in order to manage the exception.

Table 12 Outputs of Data Translation Block

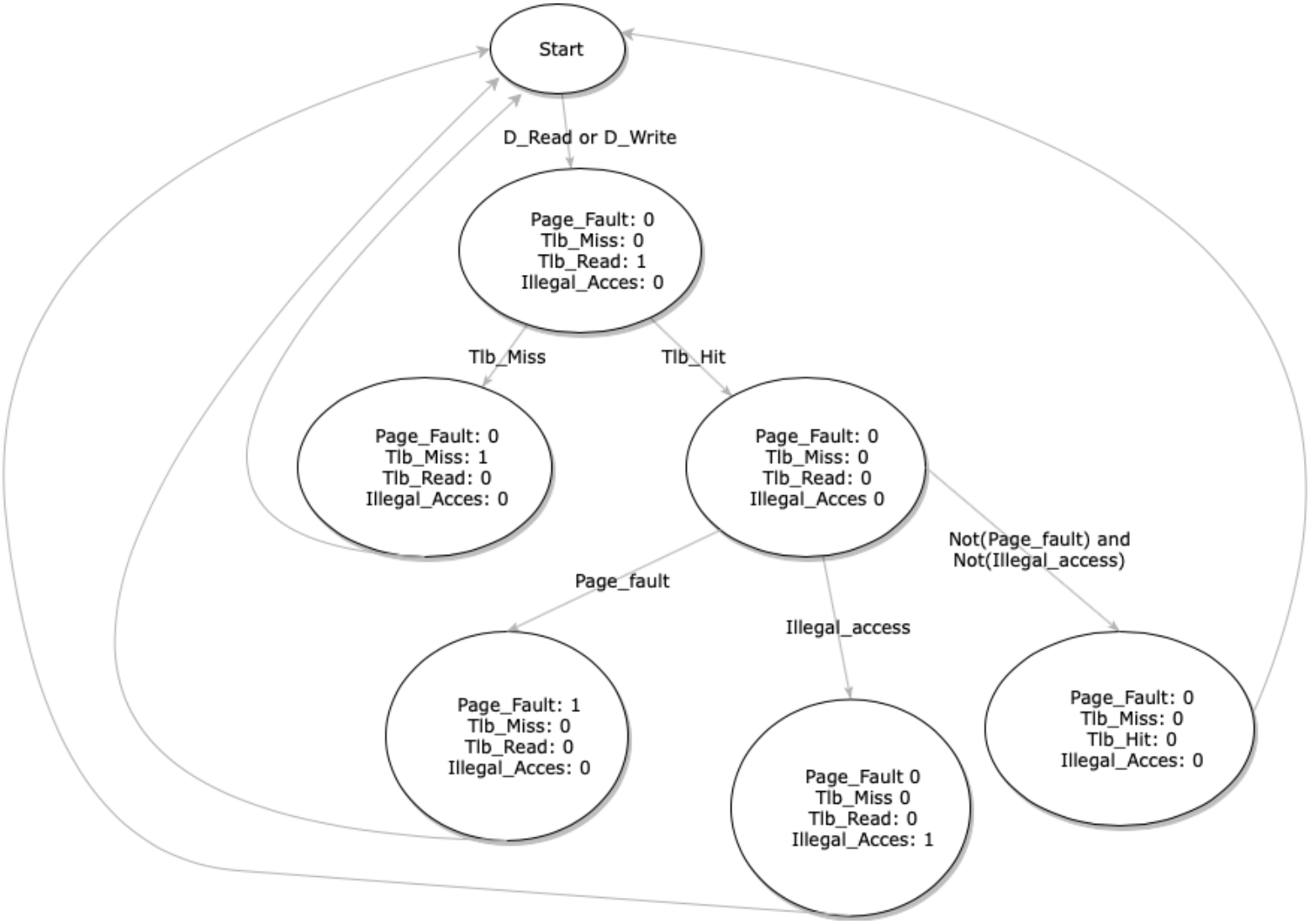

Figura 1.10 FSM Data Translation Block

4.2 Instruction Translation Block

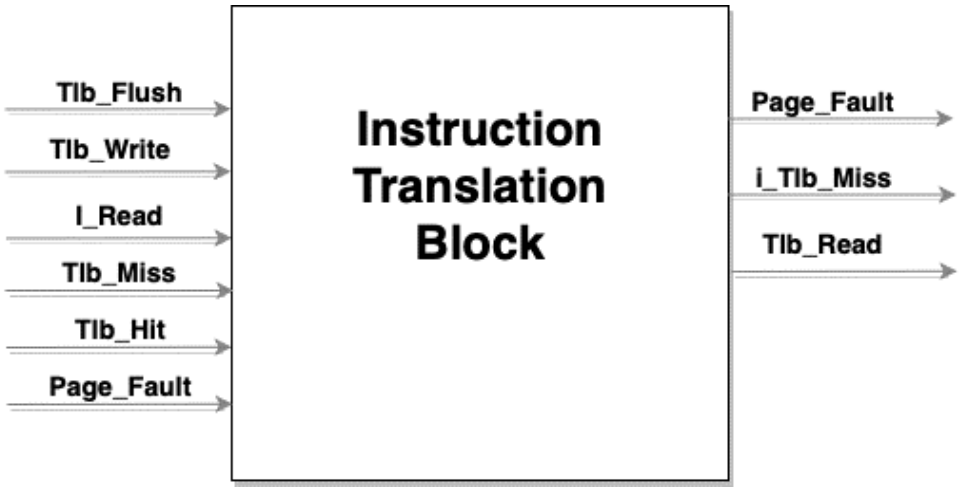

Figura 1 Instruction Translation Block

Name	Type	Bits	Function
TLB_Flush	Control	1	This signal is asserted when it is necessary to flush the istruction-TLB in case of context switch.
TLB_Write	Control	1	This signal enables writing to the istruction-TLB.
I_Read	Control	1	This signal comes from the CPU to enable the translation when there is a reading request.
TLB_Miss	Control	1	This signal comes from the instruction-TLB when occurs a miss.
TLB_Hit	Control	1	This signal comes from the instruction-TLB when occurs a hit.
Page_Fault	Control	1	This signal comes from the instruction-TLB when occurs a page fault (the validity bit of the entry is set to 0).

Table 13 Inputs of Instruction Translation Block

Name	Type	Bits	Function
Page_Fault	Control	1	This signal is propagated to the CPU when there is a page fault in order to manage the exception.
I_TLB_Miss	Control	1	This signal is propagated to the CPU when occurs a miss in the instruction-TLB in order to stall the pipeline.
TLB_Read	Control	1	This signal enables reading from the instruction-TLB in order to search the requested entry.

Table 14 Outputs of Data Translation Block

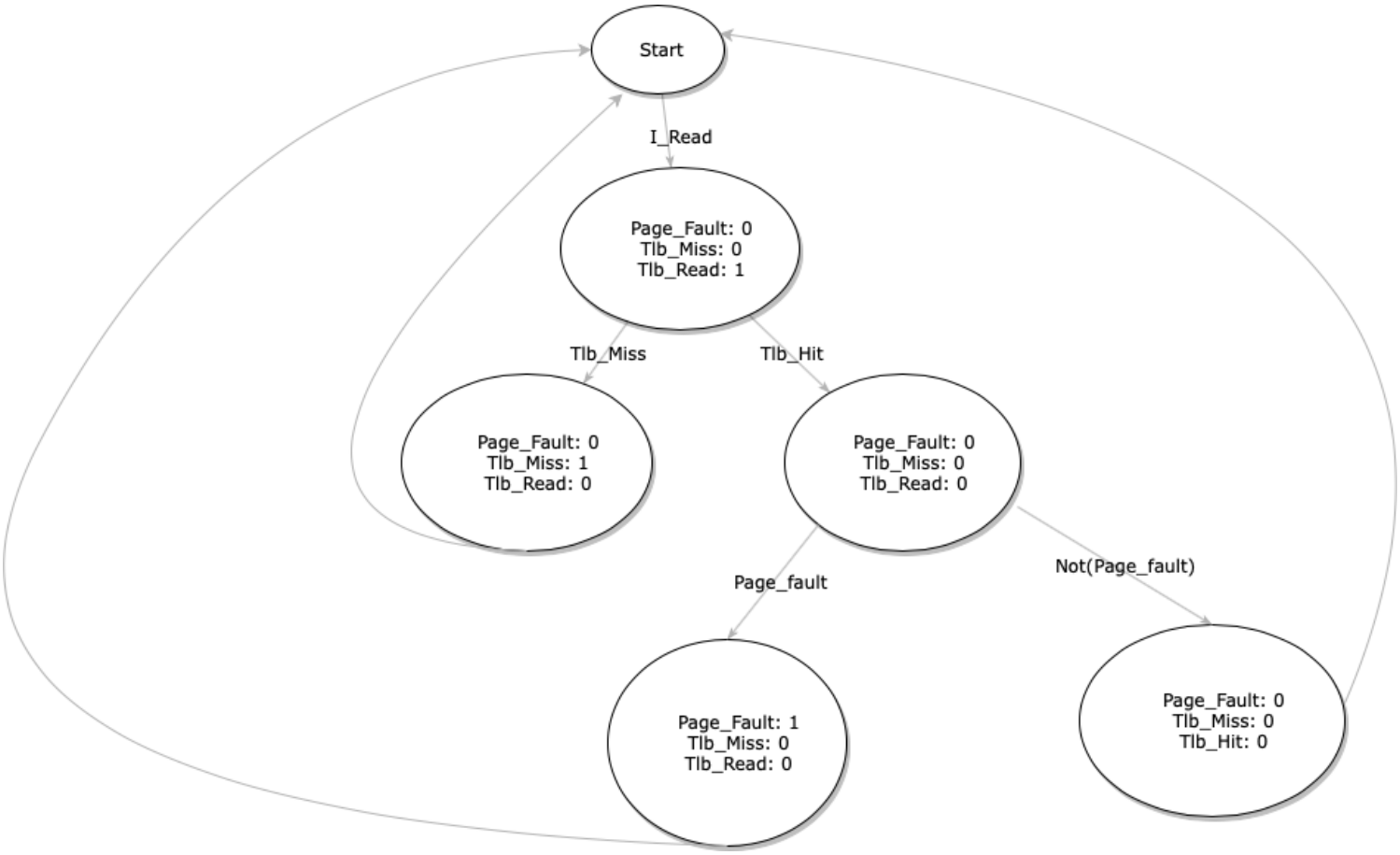

Figura 1.12 FSM Instruction Translation Block

5 Memory System

In this section will be shown the top level design with all the explicited signals and then will be made a focus on all the MMU components.

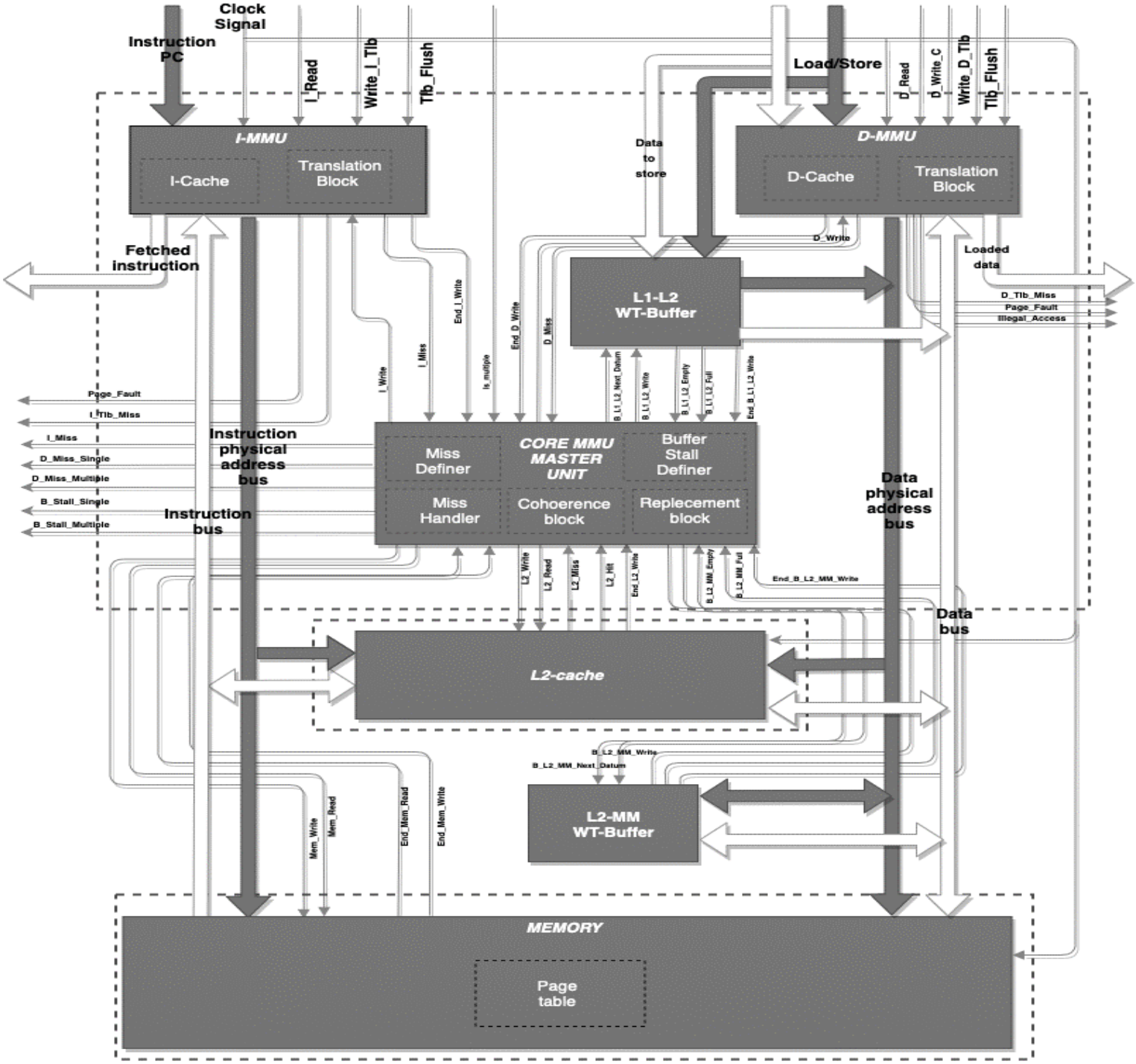

Figura 1.13 Complete Design

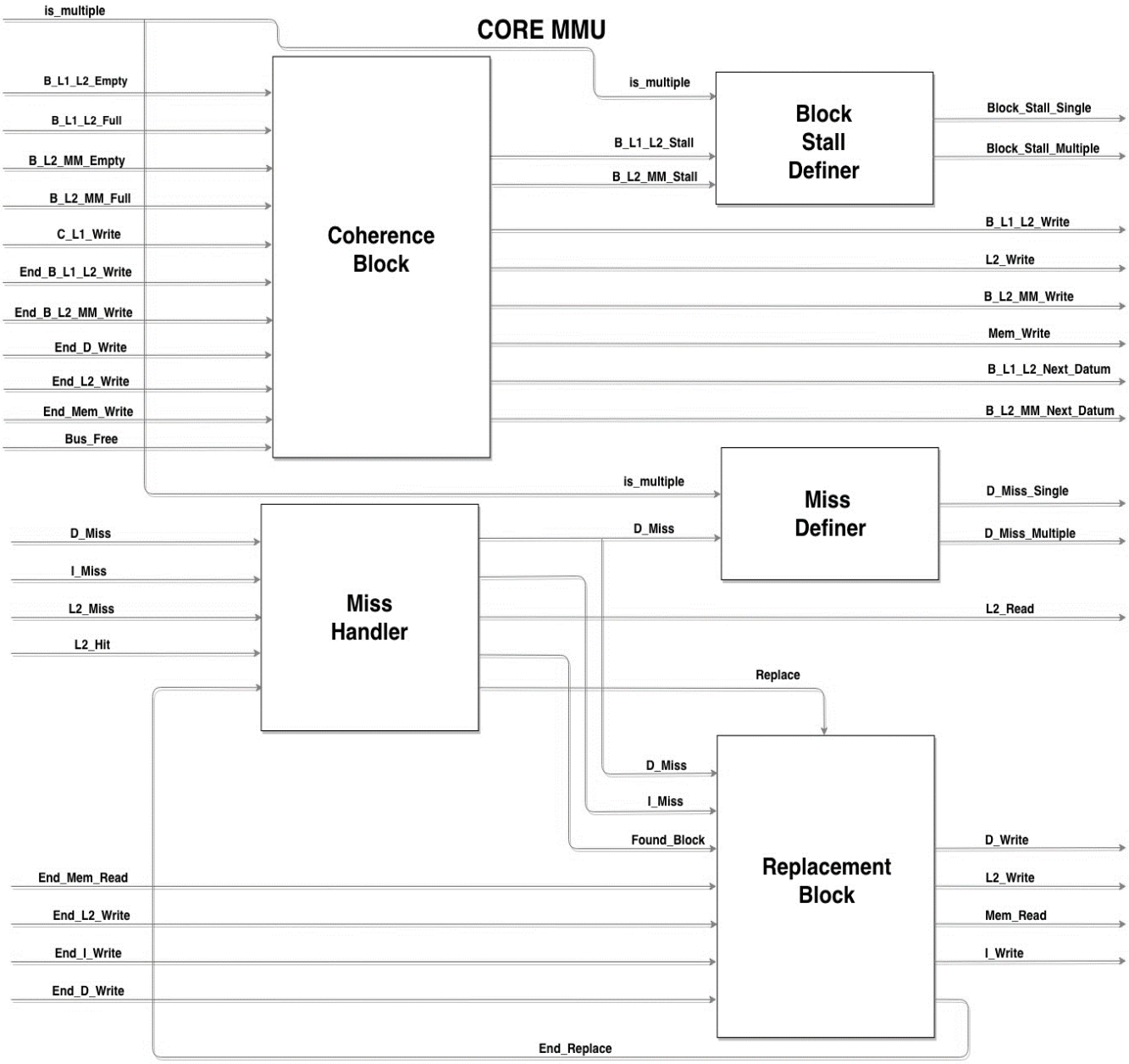

Figura 1.14 Core MMU

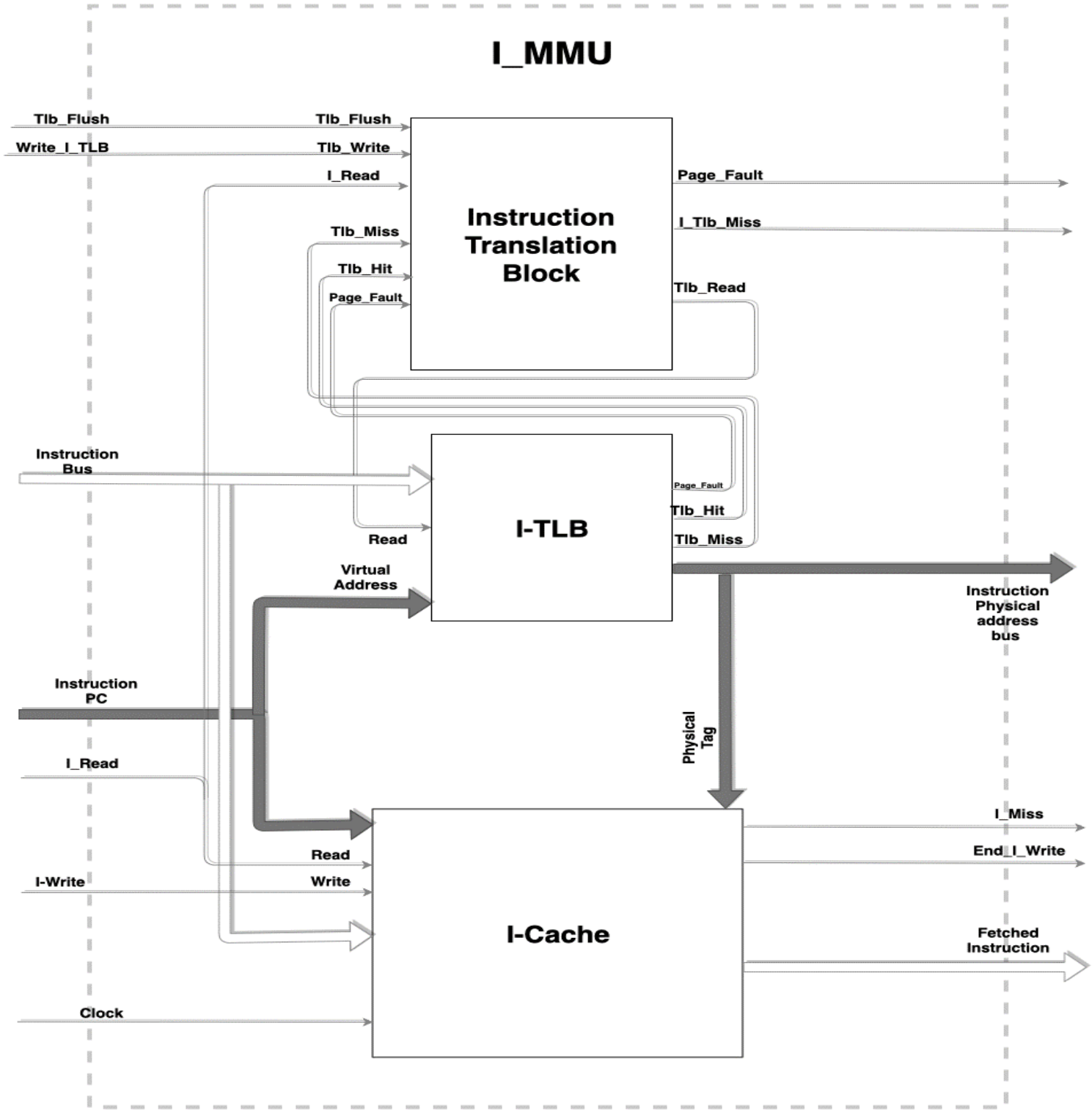

Figura 1.16 I-MMU

Resources allocation

Resources per leadership

Leadership	Member
Team Leader	Fernando Manna
CPU Leader	Giovanni Di Prisco
MMU Leader	Edoardo Cossentino
Documentation Leader	Francesco de Pertis

Resources timesheet

Sub-team	Team members	Tot. work hours d1	Tot. work hours d1 review
MMU	Eduardo Cossentino	28	-
	Antonio Coppola	27	-
	Andrea Maione	27	-
	Giuseppe Mascolo	25	-
Documentation	Antonino Durazzo	30	1