Deliverable d0: top level architecture

Sistemi di Elaborazione a.a. 2018/2019

Group M
Revision 3

Produced by	Date of Approval	Approved by	Version
Ilaria Gigi, Edoardo Cossentino, Francesco de Pertis	24 Nov. 2018	Francesco de Pertis, Fernando Manna	1.3

Deliverable d0: top level architecture 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	Ilaria Gigi, Edoardo Cossentino, Francesco de Pertis	First complete draft
09 November 2018	1	Ilaria Gigi	Removed tables at the beginning of chapters, added an approvance table in the first page, moved technical section after title and before Contents. Restyled chapters. Added table of figures and restyled other tables.
12 November 2018	2	Fernando Manna, Ilaria Gigi	Updated top-level block diagram, and team allocation. Updated Table 2-4 Functional Units cycle times. Added ROB and RSR control signals explanation in Hazard Unit section.
24 November 2018	3	Fernando Manna, Ilaria Gigi, Giovanni Di Prisco	Updated resources timesheet and top-level block diagram. Reviewed Chapter 1: 2.1 Dynamic Branch Prediction 2.2 Forwarding Unit. Updated Table 2-2 Address/data and control bus for Forwarding Unit.

Contents

Deliverable d0: top level architecture

List of Tables

Deliverable d0: top level architecture

List of Figures

Deliverable d0: top level architecture

List of Acronyms

Deliverable d0: top level architecture

ASID	Address Space Identifier
AU	integer Arithmetic Unit
BHT	Branch History Table
BPB	Branch Prediction Buffer
BPU	Branch Prediction Unit
BTB	Branch Target Buffer
BU	Branch instruction Unit
CC	Clock Cycle
EX	Execute
FU	Functional Unit
ID	Instruction Decode
IF	Instruction Fetch
L1	Level 1
L2	Level 2
LSU	Load and Store instructions Unit
MEM	Memory
MM	Main Menu
MMU	Memory Management Unit
MU	Multiplication Unit
OOO	Out-of-Order
PC	Program Counter
ROB	Re-Order Buffer
RSR	Result Shift Register
TLB	Translation Lookaside Buffer
VIPT	Virtually Indexed Physically Tagged
VM	Virtual Memory
WB	Write Back
WT	Write Through

Preface

This preface introduces the Deliverable d0: 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:

- Branch Prediction Unit
- Forwarding Unit
- Hazard Unit
- Out-of-Order Execution with Re-Ordering Buffer
- 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: CPU

Chapter 1 describes the specific ARM implementation designed by group M.

Chapter 2: MMU

Chapter 2 describes choices and methodologies adopted by group M for designing a compatible MMU for the designed CPU.

Chapter 3: Resources

Chapter 3 describes how work has been organized between team members and the resulting total efforts in working hours

Conventions

Typographical

The typographical conventions are:

UPPERCASE	Denotes entire logical blocks
Capitalization	Denotes logical behaviours

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: CPU

This chapter introduces the ARM-based CPU implementation and contains the following sections:

1 Introduction and Reference Architecture

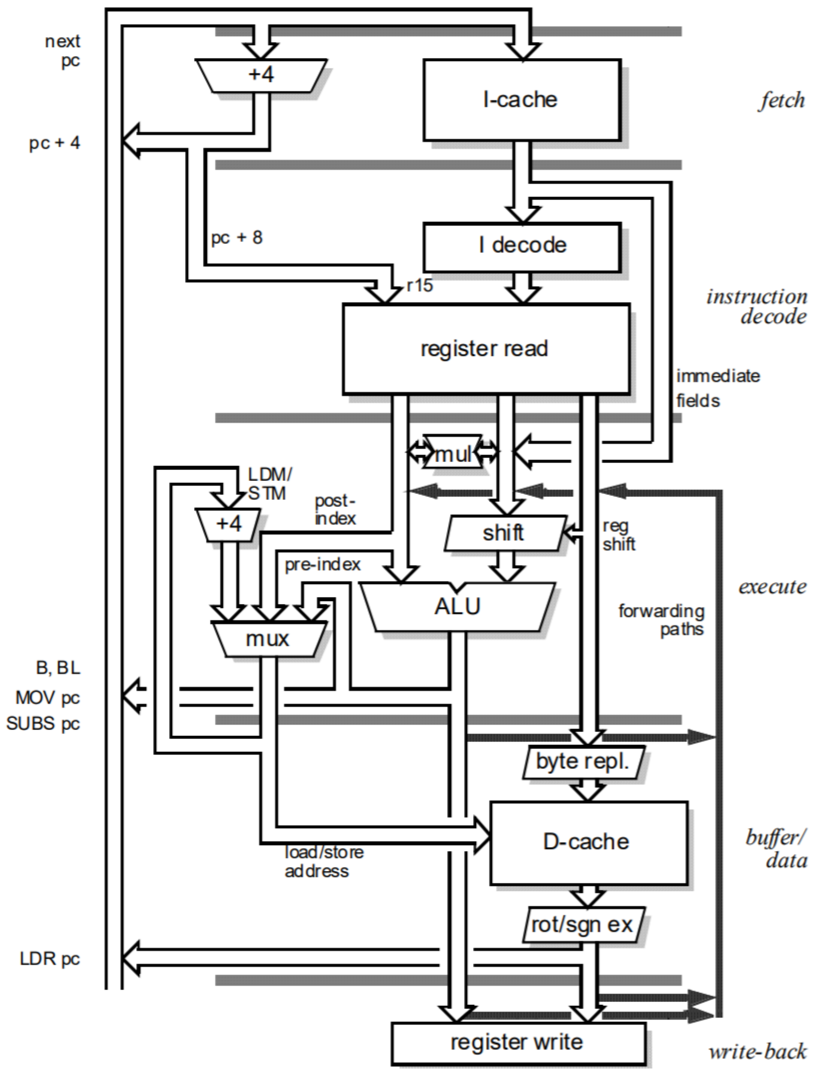

Figure 1-1 ARM9TDMI Reference Architecture design

The improvement of the ARM9TDMI processor core requires the addition of Branch Prediction, Operand Forwarding, Hazard Management and Out-Of-Order Execution modules in accordance with the initial datapath. Indeed, the original dataflow has been altered only to allow the inclusion of the above mentioned functionalities, whereas no component of the ARM9TDMI has been removed: each stage preserves the original features and all 4 inter-stage buffers are still maintained.

In particular, ID and WB stages structure remains unchanged, whereas, a makeover is necessary for the other ones, since the features to be implemented induce several changes.

In the IF stage, differently from the reference, the BPU manages the update of R15, and the MMU is responsible for accessing the Memory Hierarchy (also in the MEM stage), in case of cache miss or page fault.

The EX stage is now more complex. Indeed, to allow Instruction-level Parallelism and Re-Ordering Buffer Reservation, many changes must be applied to the reference architecture, such as the introduction of RSR and functional units, adopted to execute different instructions at the same time.

The new features are interconnected with each other but they are gathered and individually explained in this chapter.

The control signals related to pipeline stalling have already been indicated at this level of detail to clarify how stalling occurrences are handled in the proposed implementation.

All details will be discussed in the next paragraphs.

2 Proposed architecture

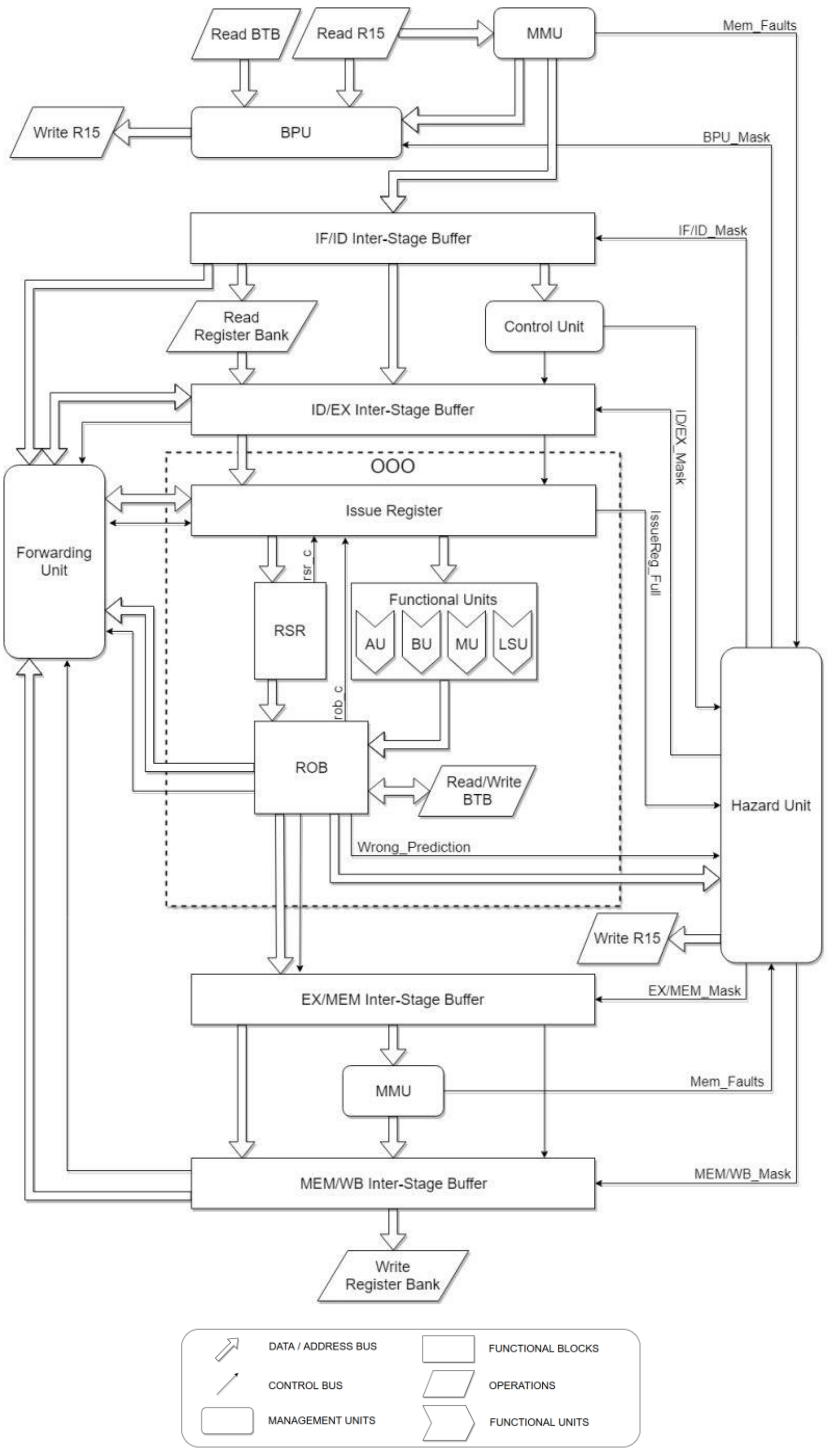

Figure 2-1 CPU top level architecture

2.1 Dynamic Branch Prediction

The implementation of dynamic branch prediction is supplied by providing a Branch Prediction Unit (BPU) based on a Branch Target Buffer (BTB).

The BPU is set up in the fetching of the instruction from the I-Cache in order to evaluate the prediction, while the BTB is used to store the result of last executions of a branch instruction. The access to the BTB is direct, the indexing is being calculated over [6:2] bits of the program counter of the branch instruction, thus the dimension of the buffer is 2^5 entries. Additional bits of the program counter are stored in the corresponding entry and are used to detect the aliasing phenomena. A validity bit is associated to each of the entries to handle the reading and the writing of the prediction bits and of the calculated branch address.

Moreover, we take the hypothesis of storing the whole 32-bit to allow to immediately withdraw the corresponding instruction, thus nullifying the branch delay; in addition, 2 bits are used to indicate the probability of exact prediction, as a trade-off between accuracy and memory usage.

The choice of the BTB implies a performance improvement in respect to other hardware solutions, such as BPB (Branch Prediction Buffer) or BHT (Branch History Table), in terms of avoiding the re-calculation of the target address in case of repeating the same branch more than once.

Table 2-1 Address/data and control bus for BPU

	Input from	**Output to**
Address/data bus	Branch Target Buffer, R15, I-Cache	R15
Control bus	Hazard Unit	-

2.2 Operand forwarding

Data and control signals coming from ID, EX, WB stages and OOO-block are forwarded through the Forwarding Unit in order to resolve dependencies

occurring in the pipeline. Dependencies may occur between instructions stored in the ID/EX inter-stage buffer, or in the Issue Register, and those stored in the subsequent inter-stage buffers, EX/MEM and MEM/WB, or in the Re-Ordering Buffer; likewise, they can occur within the OOO-block itself, between the instructions queueing in the Issue Register.

These dependencies cannot be solved by the forwarding unit until the involved instructions complete their execution, which means they reach the ROB. Provided that an instruction has properly produced the required result and it is stored in the ROB, the Propagation Unit must guarantee the correct operand whenever the next instruction is going to be issued.

Once the result of the execution has left the ROB, it must still be guaranteed to the instructions to be issued, yet after the write-back occurrence; the write-back, indeed, does not guarantee that all the instructions which are in the pipeline have the correct operands. Then, it is reasonable that some instructions, that have not been executed yet, are expected to have overcome the decode phase before the occurrence of the write-back and to still require the results previously stored in the ROB but which have already left the pipeline. In this case, a refined management policy of the Forwarding Unit will be disclosed in the block level design.

Table 2-2 Address/data and control bus for Forwarding Unit

	Input from	**Output to**
Address/data bus	OOO-block, IF/ID, ID/EX and MEM/WB inter-stage buffers	ID/EX inter-stage buffer, OOO-block
Control bus	OOO-block, ID/EX and MEM/WB inter-stage buffers	OOO-block

2.3 Hazard unit

The hazard unit supplies the management of branch prediction faults, memory faults, software interrupts, decode errors and hazards coming from the OOO-block.

This task is performed by stalling pipeline stages, depending on the nature of detected hazard, which means flushing the inter-stage buffers and restoring, accordingly, the Program Counter.

The advantage of choosing to stall single stages is to maintain the execution of the other stages, so that other stalls potentially incoming (e.g.: due to a full Issue Register or full ROB) are avoided, although no clock cycles are saved within this solution, just like in the case of stalling the entire pipeline.

Indeed, in case of a full Issue register there is no point into stalling the entire pipeline and, thus, blocking the execution since it would lead to an increasing risk of filling up the ROB; from this perspective, maintaining the execution alive implicate the outcome of releasing the Issue Register.

On the other hand, a full Re-Ordering Buffer necessarily requires the entire pipeline not to be stalled, otherwise a deadlock would occur since there is no other way of releasing the ROB than carrying on the execution of MEM and WB stages.

In order to achieve this, a proper signal is sent, from ROB, to the Issue Register, to ensure that no instruction is issued to any Functional Unit while ROB is full. The same happens to manage a busy RSR slot, or a busy Functional Unit, when the Issue Register is held by RSR, until the slot, or the unit, is available.

By performing this, the execution keeps going without and any control needed by the Hazard Unit.

Eventually, it goes without saying that the probability of these occurrences being taken place may be reduced by properly scaling the buffers.

Table 2-3 Address/data and control bus for Hazard Unit

	Input from	**Output to**
Address/data bus	OOO-block	R15
Control bus	OOO-block, MMU and Control Unit	BPU and inter-stage buffers

2.4 Out-of-order execution

The Out-Of-Order execution is being implemented through the adoption of an Issue Register, a RSR and a ROB, incorporating 4 functional units providing: integer Arithmetic Unit (AU), Multiplication Unit (MU), Branch instruction Unit (BU), Load and Store instructions Unit (LSU).

The delay introduced by the 4 functional units refers to the clock cycles described in ARM Technical Manual, in details:

Table 2-4 Functional Units cycle times

FU	**RI**	**CC**
AU	Data Op	1 to 4
MU	MUL, MLA	3 to 6
MU	SMULL, UMULL,	4 to 7

	SMLAL, UMLAL	
BU	B, BL, BX	3
LSU	LDR	1 to 5
LSU	STR	1
LSU	LDM	2 to (n + 4), n > 0
LSU	STM	2 to n, n > 0

where n means the number of words transferred in an LDM/STM/LDC/STC.

In accordance with the table, the maximum size of RSR is related to the longest clock cycles taken by the functional units, except LSU, separately managed, that is 7.

The Issue Register has been chosen with a view to obtain gain in terms of performance derived by this adoption, made by queueing instructions and issuing in sequence, which is faster than issuing a single instruction from the ID/EX, once per clock cycle, with a higher complexity price.

Table 2-5 Address/data and control bus for Out-of-order block

	Input from	**Output to**
Address/data bus	ID/EX inter-stage buffer, Forwarding Unit	Forwarding Unit, Branch Prediction Unit, Hazard Unit, EX/MEM inter-stage buffer
Control bus	ID/EX inter-stage buffer, Forwarding Unit	Hazard Unit and EX/MEM inter-stage buffer

Chapter 2: MMU

This chapter introduces the MMU specific implementation and contains the following sections:

1 Introduction

The initial assumption is that the memory system is based on a hierarchy split into three levels: a first level of set-associative caching with separate Instruction cache and Data cache, a second level of set-associative caching with unified cache and a third one including both main memory and a virtual one, involving, also, a mechanism to manage it.

The architecture follows the Harvard model, which means there are two separate caches: one for instructions and another for data.

2 Sizing

2.1 Cache L1 sizing

Both data and instruction caches of the first level of hierarchy has been chosen to be sized 16KB in order to reduce access time despite gaining a higher miss probability.

2.2 Cache L2 sizing

The second level of cache has been sized 256 KB aiming at reducing the occurrences of looking for blocks in the main memory since it would lead to a higher miss penalty.

2.3 MM sizing

The main memory can reach maximum 4 GB, since the dimension of the addresses produced by the processor is 32 bits.

2.4 Block sizing

Due to a deep analysis of benchmarks, the block size has been set to 64B. This solution allows to gain a miss probability of 2.5% approximately for first-level cache, whereas one of 0.5% for the second-level cache (information derived by benchmarks). Whether a greater size had been chosen, it would have led to a lower miss frequency but higher time delay to transfer blocks throughout the memory hierarchy (thus increasing miss penalty). Indeed, greater sized blocks would cause a reduction in the number of blocks which can be stored inside a cache, thus decreasing the number of blocks currently stored for each process in concurrent/parallel execution and increasing, for each of them, miss probability.

2.5 Page sizing

The page size has been set to 4KB, after analyzing several real systems, to gain a reasonable trade-off between a rather low transfer time of a page from main memory to virtual memory (et vice versa) and the observance of the principle of locality.

3 Cache organization

3.1 Cache L1 organization

The organization of first-level cache is 4-way set-associative cache, preserving the constraint on dimension imposed by the nature of chosen addressing methodology (see paragraph Addressing) and thus preventing aliasing phenomena.

3.2 Cache L2 organization

The second-level cache is inclusive, in terms of including data already present in L1 cache, since this choice leads to ease coherence problems between the two levels of caches in the hierarchy. The organization of second-level cache is 8-way set-associative, to reduce miss frequency, notwithstanding a higher hit time delay, since the basic goal of an L2 cache is to reduce the miss penalty.

4 Addressing

4.1 Cache L1 addressing

The addressing methodology for the L1 cache has been chosen to be VIPT (Virtually Indexed Physically Tagged) to reduce research time of data inside the cache. It is clear that this choice implies a constraint on cache size since it cannot exceed the size of the page multiplied the number of the sets (that is, in this case, 4KB x 4 = 16KB).

4.2 Cache L2 addressing

Physical addressing has been adopted for the L2 cache to avoid constraints on size imposed by aliasing phenomena.

5 TLB

Two TLBs have been adopted, one for data cache and another for instructions cache to reduce access time. Typically, TLBs' dimensions range from 16 to 512 entries. Having two distinct TLBs it is convenient to have small TLBs, in order to reduce the hit time as much as possible. Moreover, by emptying the TLB to every context switch, all entries refer to the task being executed and the TLB is exploited to the maximum. For this reason, two TLBs of 16 entries each are used.

Each TLB is completely associative. This solution leads to reasonable performances in terms of a trade-off between a higher research time, due to have chosen a completely associative memory, and to have opted for a small size of the TLB.

6 Replacement Strategies

6.1 Cache L1-L2

In both first-level cache and second-level cache it has been chosen LRU (Least Recently Used) strategy to replace the blocks, implemented by hardware with the intent of minimizing miss penalty, whereas software replacement algorithm would impose a bigger penalty.

6.2 TLB

A usage bit has been associated to each entry of the TLB which turns out to be useful for the operating system to understand which pages have been used most recently. This bit is periodically reset by the operating system itself.

For each context switch an emptying of the TLB occurs. The alternative solution would be to keep track for each entry of a task identifier (ASID), but the identifier of the current task should be recorded in an ad hoc register. A hit in the TLB would, therefore, require a match also between the ASID contained in the register and the one in the selected entry.

Since the aim is not to introduce additional hardware except for a significant improvement in performances, it was decided to adopt the emptying of the TLB.

The management of miss faults in the TLB in embedded systems is managed via software to avoid costly management via hardware. An algorithm with a low computational complexity is required in order to have a very fast management. It has been decided to adopt an algorithm that replaces a random block among those that have the usage bit to 0.

6.3 MM

A miss in the main memory involves a very high reduction of performances. For this reason, this kind of miss has been decided to be managed by software, to use sophisticated algorithms for reducing the miss penalty.

7 Write Strategies

7.1 L1 - L2, L2 - MM

To manage the coherence between first-level cache and second-level cache and between second level cache and main memory it has been adopted the write-through strategy with a write buffer (this concerns only data cache since instructions cache is read-only).

This is the least expensive write strategy to implement and the buffer allows not having stalls until the buffer is full.

The write buffer is advantageous itself because the latency for accesses is very small and the update of the underlying memory level is realized when the address and data buses are free, and this is reasonable if the system is assumed not to be in saturation.

7.2 MM - VM

In the last level of hierarchy (main memory - virtual memory) the write-back strategy has been adopted since it involves less write operations, in respect to the write-through strategy, and accounting for that writing in virtual memory means a high cost in terms of clock cycles.

7.3 TLB

In addition to the usage bit, each entry of the TLB also has a dirty bit to indicate if the page has been modified. This bit is present only in the data TLB because the instructions are read-only. The dirty bit (if present) and the usage bit might change in TLB, thus it is necessary to define a coherence policy between TLB and page table.

It goes without saying that to use a write-back write policy is worthwhile since it is expected that the frequency of misses for the TLB is very low, working with embedded systems. Indeed, these systems are designed to repeatedly run low-cost applications.

8 Proposed architecture

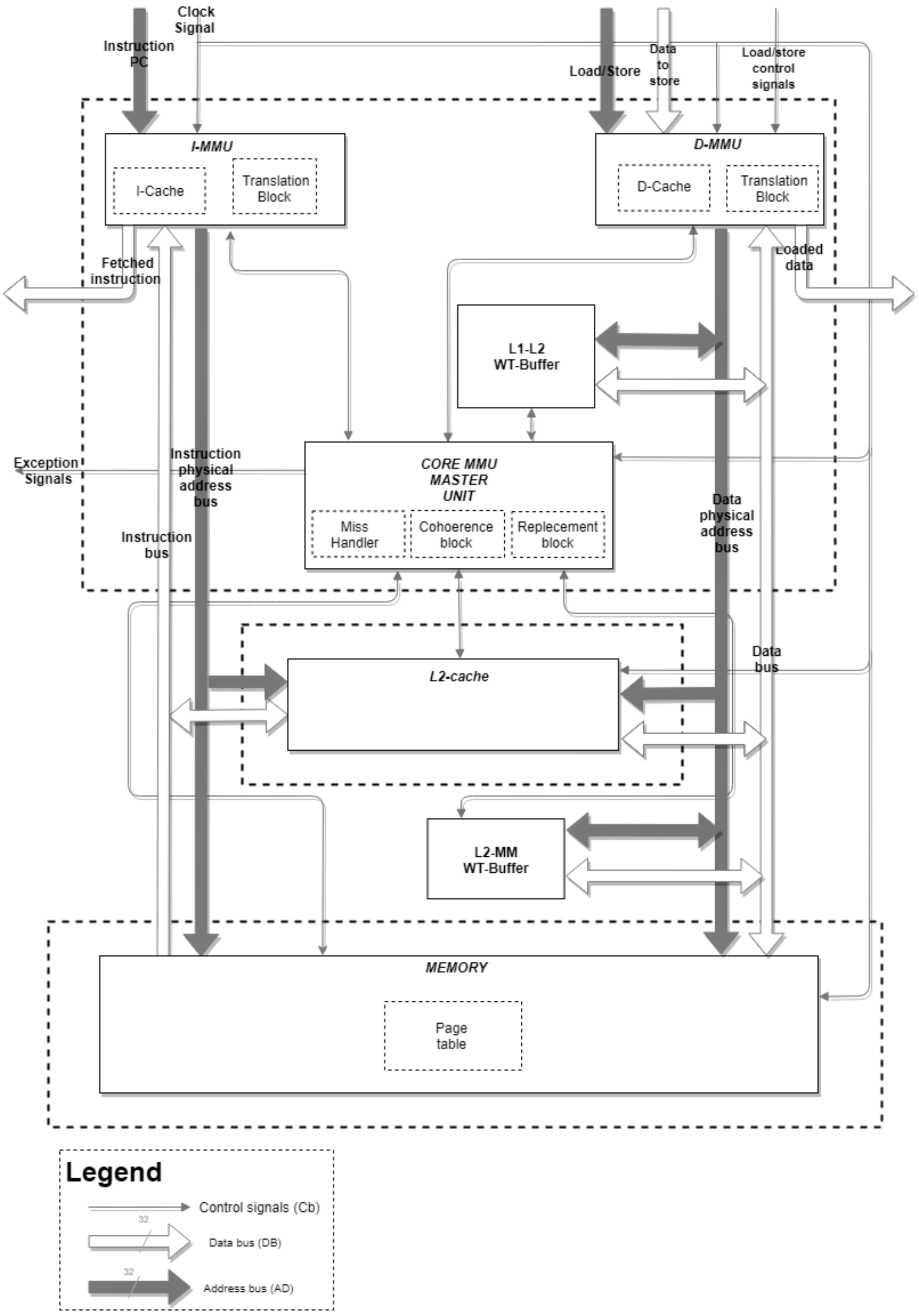

Figure 8-1 MMU top level architecture

8.1 Core MMU

This block executes three distinct activities through three different functional blocks:

1) Miss Handler finds the missing block in the memory hierarchy when a miss occurs.
2) Replacement functional block transfers the found block, that was missing, through the levels of the memory hierarchy.
3) Coherence functional block manages the coherence of the memory hierarchy and, when the bus is free, enables the buffer to write to the lower hierarchy level.

Table 8-1 Address/data and control bus for Core MMU

Input from	Output to
Clock Signals, Control Signals from I-MMU, D-MMU, L2-Cache, WT-Buffer, Memory	Exception Signals to CPU, Control Signals to I-MMU, D-MMU, L2-Cache, WT-Buffer, Memory

8.2 I-MMU:

This block acts as an interface between instruction cache - processor and between instruction cache - core MMU.

Inside this block there are two functional blocks:

1) Translation block manages the address translation.
2) I-Cache implements the accesses in cache detecting a cache miss and forwarding it.

Table 8-2 Address/data and control bus for I-MMU

Input from	Output to
Instruction PC (bus from CPU), Instruction (bus from L2-cache, Memory), Clock Signals from CPU, Control Signals from CORE-MMU	Fetched Instruction (bus to CPU), Instruction physical address (bus to Memory, L2-cache), Control Signals to CORE-MMU

8.3 D-MMU:

This block interfaces the data cache with the processor and the main-MMU. It involves two functional blocks:

1) Translation block manages the address translation.

2) D-Cache implements the accesses in cache detecting a cache miss and forwarding it.

Table 8-3 Address/data and control bus for D-MMU

Input from	**Output to**
Load or Store address (bus from CPU), Load or Store Control Signals from CPU, Clock Signals from CPU, Control Signals from CORE-MMU, Data (bus from L2-cache, Memory)	Loaded data (bus to CPU), Data instruction physical address (bus to WT-buffer, Memory), Control Signals to CORE-MMU, Data (bus to L2-cache, Memory)

Chapter 3: Resources

This chapter introduces the resources involved in the design and contains the following sections:

- *Resources allocation* on page 18
- *Resources timesheet* on page 19

Resources allocation

Resources per macro-task

Macro-task	Team members
Memory Management Unit	Edoardo Cossentino Antonio Coppola Andrea Maione Giuseppe Mascolo Giuseppe Urciuoli
CPU / Hazard and Forwarding	Ilaria Gigi Giovanni Di Prisco
CPU / Branch Prediction	Marco Schettini Ascanio Guglielmelli
CPU / Out-Of-Order execution	Fernando Manna Michele Rescigno Mario Mupo Romeo Rinaldi
Documentation	Francesco de Pertis Antonino Durazzo Giuseppe Cirillo

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

Macro-task	Team members	Tot. work hours d0	Tot. work hours d0 review
Memory Management Unit	Edoardo Cossentino	40	4
	Antonio Coppola	40	4
	Andrea Maione	40	4
	Giuseppe Mascolo	40	4
CPU / Hazard and Forwarding	Ilaria Gigi	40	6
	Giovanni Di Prisco	40	6
CPU / Branch Prediction	Marco Schettini	40	4
	Ascanio Guglielmelli	40	4
CPU / Out-Of-Order execution	Fernando Manna	40	6
	Michele Rescigno	40	4
	Mario Mupo	40	4
	Romeo Rinaldi	40	4
Documentation	Francesco de Pertis	40	4
	Antonino Durazzo	40	4
	Giuseppe Cirillo	40	4

Report over resulting total effort deliver d0

On the deliverable d0, each team member averaged 4 working hours per day, for a 40 hours total, calculated on 10 working days (excluding Sunday).

As this result may seem unrealistic and not aligned with total effort estimated for the project (80 hours/person total), initial difficulties caused little slowdowns (team is not experienced), and some additional time has been spent trying to recover those difficulties, and also to anticipate some work for the next deliverables.