ARQUITETURA DE COMPUTADORES

Nome:_____Nº_____
Nome:_____Nº_____
Grupo: _____ Dia:_____ Hora:_____ Sala: _____ Docente:_____

Laboratório 6 – Análise de uma Hierarquia de Memória

Este laboratório destina-se a consolidar conhecimentos de introdução a caches e hierarquias de memória. No laboratório será utilizada uma descrição em VHDL do sistema ilustrado na Figura 1. O trabalho irá consistir em analisar a arquitetura do sistema e testar o código VHDL fornecido, de acordo com os exercícios deste enunciado.

O trabalho descrito neste enunciado deve ser realizado **FORA** do horário de laboratório, destinando-se este à demonstração e avaliação do trabalho realizado. No horário de laboratório serão fornecidos exercícios adicionais que devem ser realizados durante a aula. No final da aula de laboratório deverá submeter o ficheiro de excel com as respostas às perguntas neste guia no Fénix.

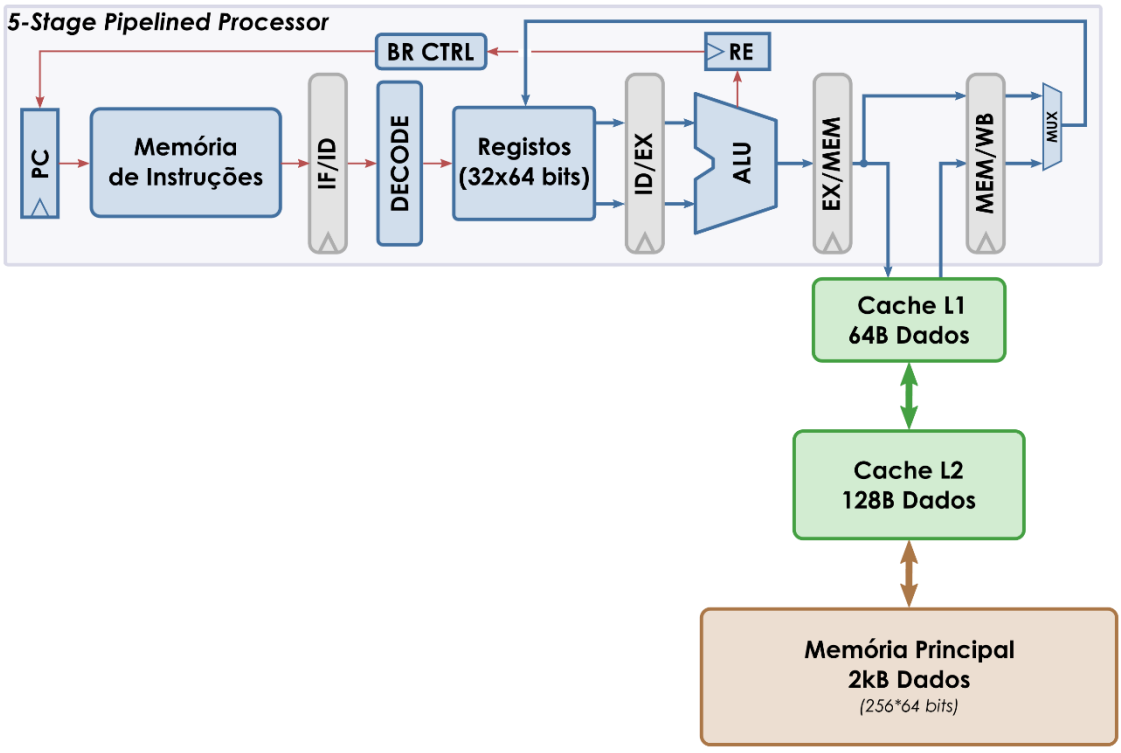

Figura 1 – Ilustração da Hierarquia de Memória do Sistema

I - Arquitetura do Processador

O processador ilustrado na Figura 1 é implementado por uma arquitetura com *pipeline* de 5 andares (IF, ID, EX, MEM e WB) e 32 bits para instruções e dados, alinhados em memória. A unidade de armazenamento (no andar ID) é composta por um banco de 32 registos de números inteiros de 32 bits (R0-R31), com LR = R30 e R31 = 0. O processador está ligado a **(1)** uma ***Memória de Instruções*** endereçada ao byte, com alinhamento a palavras de 4 bytes e com capacidade 256 x 32 bits (para instruções); e **(2)** uma ***Hierarquia de Memória*** (descrita na secção seguinte) com 2 níveis de C**ache de Dados** (L1-D e L2-D) e uma **Memória Principal**.

O processador está equipado com os seguintes mecanismos de resolução de conflitos:

- ***Predição de Salto*** – Assume **sempre** salto não tomado. Resolve casos em que o salto é tomado eliminando a instrução seguinte (substituindo por um NOP).
- ***Conflitos de Dados***
  - ***Flags do EX –*** Bypass do Registo de Estado (andar EX) diretamente para o controlo de salto (andar ID). Resolve conflitos do tipo *compare→branch*.
  - ***Conflitos RAW (todos os outros) –*** Implementação de um mecanismo de deteção de conflitos do ID para os andares seguintes. Resolução de conflitos através da introdução de *Stalls*.

A organização e descrição dos ficheiros mais relevantes fornecidos no projeto do Vivado estão apresentadas na Tabela 1. O conjunto de instruções suportado pelo processador está detalhado nas Tabelas 2 e 3 (página seguinte).

Tabela 1 - Organização e descrição dos ficheiros fornecidos no projeto para a arquitetura do processador

Implementação: ***Design Sources***	
Core-FullProcessor	Processador (core) Pipelined
↳ **InstructionFetch-IFStage**	Andar IF
↳ **PC-PCReg**	Program Counter
↳ **IMEM-InstMemory**	Memória de Instruções
↳ **IFIDRegisters-IFRegisters**	Registos IF-ID
↳ **InstructionDecode-IDStage**	Andar ID
↳ **IDU-InstDecodeUnit**	Unidade de Descodificação
↳ **HazardControl-ScoreBoard**	Controlo de Hazards
↳ **BCU-BranchControlUnit**	Unidade de Controlo de Salto
↳ **RF-RegisterFile**	Banco de Registos
↳ **R0-31-Reg32b**	32 Registos de 32 bits
↳ **IDEXRegisters-IDRegisters**	Registos ID-EX
↳ **Execute-EXStage**	Andar EX
↳ **ALU-FunctionalUnit**	ALU
↳ **AU-ArithUnit**	Unidade Aritmética
↳ **LU-LogicUnit**	Unidade Lógica
↳ **SU-Shifter**	Barrel Shifter
↳ **SReg-StateRegister**	Registo de Estado
↳ **N,Z,C,V-ffd**	4 flip-flops tipo D
↳ **EXMEMRegisters-EXRegisters**	Registos EX-MEM
↳ **MemoryAccess-DataMemory**	Andar MEM - Memória de Dados
↳ **MEMWBRegisters-MEMRegisters**	Registos MEM-WB

Tabela 2 - Conjunto de instruções suportado.

MNEM	Formato de instrução (32 bits)						Descrição	Flags geradas	Descrição Assembly
		I(31:26)	I(25:21)	I(20:16)	I(15:11)	I(10:0)			
NOP	-	000 000	-				*No Operation*	*nenhum*	**NOP**
ADD	*A*	010 000	DR	SA	SB	-	R[DR] ←R[SA] + R[SB]	N,Z,C,V	**ADD** *RD,RA,RB*
SUB	*A*	010 001	DR	SA	SB	-	R[DR] ←R[SA] – R[SB]	N,Z,C,V	**SUB** *RD,RA,RB*
AND	*A*	010 010	DR	SA	SB	-	R[DR] ←R[SA] *and* R[SB]	N,Z	**AND** *RD,RA,RB*
ORR	*A*	010 011	DR	SA	SB	-	R[DR] ←R[SA] *or* R[SB]	N,Z	**ORR** *RD,RA,RB*
CMP	*A*	000 001	-	SA	SB	-	Flags ←R[SA] – R[SB]	N,Z,C,V	**CMP** *RA,RB*
EOR	*A*	010 100	DR	SA	SB	-	R[DR] ←R[SA] *xor* R[SB]	N,Z	**EOR** *RD,RA,RB*
ADDI	*B*	110 000	DR	SA	IMM16		R[DR] ←R[SA] + s.ext(IMM16)	N,Z,C,V	**ADDI** *RD,RA,imm16*
SUBI	*B*	110 001	DR	SA	IMM16		R[DR] ←R[SA] – s.ext(IMM16)	N,Z,C,V	**SUBI** *RD,RA,imm16*
ANDI	*B*	110 010	DR	SA	IMM16		R[DR] ←R[SA] *and* s.ext(IMM16)	N,Z	**ANDI** *RD,RA,imm16*
ORRI	*B*	110 011	DR	SA	IMM16		R[DR] ←R[SA] or s.ext(IMM16)	N,Z	**ORRI** *RD,RA,imm16*
EORI	*B*	110 100	DR	SA	IMM16		R[DR] ←R[SA] xor s.ext(IMM16)	N,Z	**EORI** *RD,RA,imm16*
LSL	*B*	101 101	DR	SA	-	UIMM6	R[DR] ←R[SA] <<< UIMM6	N,Z,C	**LSL** *RD,RA,uimm6*
LSR	*B*	101 110	DR	SA	-	UIMM6	R[DR] ←R[SA] >>> UIMM6	N,Z,C	**LSR** *RD,RA,uimm6*
ASR	*B*	101 111	DR	SA	-	UIMM6	R[DR] ←R[SA] >>> UIMM6	N,Z,C,V	**ASR** *RD,RA,uimm6*
B	*C*	100 000	Offset				PC ←NextPC + LSL Offset, 2	*nenhum*	**B** *Offset*
B.*cond*	*C*	100 ***cnd***	Offset				PC ←NextPC + LSL Offset, 2 (*se cond válida*)	*nenhum*	**B.*cond*** *Offset*
BL	*C*	100 111	Offset				PC ←NextPC + LSL Offset, 2 R[LR] ←NextPC	*nenhum*	**BL** *Offset*
BR	*C*	000 111	-		SB	-	PC ←R[SB]	*nenhum*	**BR** *RB*
LDR	*A*	011 100	DR	SA	SB	-	R[DR] ←M[R[SA] + R[SB]]	*nenhum*	**LDR** *RD,[RA+RB]*
STR	*A*	011 101	DR	SA	SB	-	M[R[SA] + R[SB]] ←R[DR]	*nenhum*	**STR** *RD,[RA+RB]*
LDRI	*B*	111 100	DR	SA	IMM16		R[DR] ←M[R[SA] + IMM16]	*nenhum*	**LDRI** *RD,[RA+imm16]*
STRI	*B*	111 101	DR	SA	IMM16		M[R[SA] + IMM16] ←R[DR]	*nenhum*	**STRI** *RD,[RA+imm16]*
MOV	*A*	011 000	DR	-	SB	-	R[DR] ← R[SB]	*nenhum*	**MOV** *RD,RB*
MOVI	*B*	111 000	DR	-	IMM16		R[DR] ← s.ext(IMM16)	*nenhum*	**MOVI** *RD,imm16*

Notação: O símbolo R[x] representa o valor do registo indicado por x; M[R[x]] denota o conteúdo da memória cujo endereço é dado pelo valor proveniente do registo Rx; RA,RB,RD correspondem ao símbolo do registo usado (i.e., R0-R31). No formato B, *IMM16* representa um inteiro com sinal de 16 bits e *UIMM6* representa um inteiro sem sinal de 6 bits. No formato C, a constante *Offset* está representada em complemento para 2.

Tabela 3 - Codificação de Saltos Condicionais.

cnd - I(28:26)	Instrução	Op.	Flags
001	**B.EQ (EQual)**	**=**	Z=1
010	**B.NE (Not Equal)**	**≠**	Z=0
011	**B.LT (Less Than)**	**<**	N!=V
100	**B.LE (Less than or Equal)**	**≤**	!(Z=0 & N=V)
101	**B.GT (Greater Than)**	**>**	(Z=0 & N=V)
110	**B.GE (Greater than or Equal)**	**≥**	N=V

II - Hierarquia de Memória

O sistema ilustrado na Figura 1 é composto por um processador ligado a uma ***Hierarquia de Memória de Dados*** composta pelos elementos de memória (endereçadas ao byte):

- ***Cache de Dados L1 (Nível 1) –*** Atende a pedidos de dados (32-bits) do processador;
- ***Cache de Dados L2 (Nível 2) –*** Atende a pedidos de dados (64-bits) da Cache L1;
- ***Memória Principal –*** Atende a pedidos de dados (64-bits) da Cache L2.

A organização e descrição dos ficheiros mais relevantes fornecidos no projeto do Vivado estão apresentadas na Tabela 4. A configuração e organização das memórias do sistema está apresentada na Tabela 5.

Tabela 4 - Organização e descrição dos ficheiros fornecidos no projeto

Implementação: ***Design Sources***	
`CPU`	Sistema de Processamento
↳ **`Core-FullProcessor`**	Processador (core) Pipelined
↳ **`L1Cache-WriteBackCache`**	Cache de Dados L1
↳ **`CacheMemory`**	Wrapper Genérico para Mem. Cache
↳ **`DirectMappedMemory`**	Memória Cache Mapeamento Direto
↳ **`L2Cache-L2WriteBackCache`**	Cache de Dados L2
↳ **`CacheMemory`**	Wrapper Genérico para Mem. Cache
↳ **`TwoWaySetAssociativeMemory`**	Memória Cache 2 Vias Associatividade
↳ **`MainMemory-MemoryController`**	Memória Principal

Tabela 5 – Configuração da Hierarquia de Memória.

CACHE L1		CACHE L2		MEMÓRA PRINCIPAL	
MAPEAMENTO DE DADOS					
Mapeamento Direto		2 Vias de Associatividade Política de substituição: - *Least-Recently Used* (LRU)			
PROTOCOLO					
Write Back, Write Allocate		Write Back, Write Allocate			
CAPACIDADE					
Capacidade (dados) 64B		Capacidade (dados) 128B		Capacidade (dados) 2KB	
Linhas de dados 8B		Linhas de dados 8B		Endereçado ao byte, mas organizado em pal. de 8B	
26-bit Tag + Valid (V) + Dirty (D)		26-bit Tag + Valid (V) + Dirty (D)			
INTERFACES DE COMUNICAÇÃO					
INCOMING: Core - 32-bits		**INCOMING**: Cache L1 - 64-bits		**INCOMING**: Cache L2 - 64-bits	
OUTGOING: Cache L2 - 64-bits		**OUTGOING**: Memória - 64-bits			
LATÊNCIA (Ciclos de Relógio)					
	LAT_{L1}		LAT_{L2}		LAT_{MEM}
Read Hit	2	Read Hit	2	Read	8
Read Miss(1)	$3+LAT_{L2}$	Read Miss	$3+LAT_{MEM}$	Write	8
Write Hit	3	Write Hit	3		
Write Miss(1)	$3+LAT_{L2}$	Write Miss	$3+LAT_{MEM}$		

(1) Caso a linha substituída seja válida (V=1) e tiver sido modificada (D=1) – *modified line eviction* –, é necessário somar a latência da escrita no nível de cache/memória inferior.

III – Programa de Teste

Este trabalho tem como base a análise e teste da execução de um programa no sistema de processamento ilustrado na Figura 1. Para o efeito, estude o funcionamento do programa apresentado abaixo (em C e na linguagem *Assembly* do processador) que implementa a transposta de uma matriz A. A matriz A está inicializada na ***Memória Principal*** e organizada por linhas (*row-major order*) com os valores:

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

A operação transposta gera a matriz

$$\text{B = Transposta de } A = \begin{bmatrix} 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \\ 4 & 8 & 12 & 16 \end{bmatrix}$$

Código 1 – Transposta de uma Matriz em C

```
int i, j, N = 4;
int A[4][4] = ...;
int B[4][4] = ...;

for(i = 0; i < N; i++)
    for(j = 0; j < N; j++)
        B[j][i] = A[i][j];
```

Código 2 – Transposta de uma Matriz em *Assembly*

```
0:          MOVI   R0, #4        ; R0 X R0 Matrix
1:          MOVI   R1, #320      ; Matrix A pointer
2:          MOVI   R2, #640      ; Matrix B pointer
3:          LSL    R3, R0, #2    ; Matriz line/column size in bytes
3:          MOVI   R4, #0        ; Matrix A line
4:          MOVI   R5, #0        ; Matrix A column
6:          ADD    R6, R2, R31   ; Matrix B column pointer
7:    LOOP  LDR    R7, [R1, R31]
8:          STR    R7, [R2, R31]
9:          ADDI   R5, R5, #1    ; Next column of A
10:         ADDI   R1, R1, #4
11:         ADD    R2, R2, R3    ; Next line of B
12:         CMP    R5, R0
13:         B.LT   -7            ; LOOP
14:         ADDI   R6, R6, R4    ; Next Column of B
15:         ADD    R2, R6, R31   ; Set B pointer to Next Column
16:         ADDI   R4, R4, #1    ; Next Line of A
17:         MOVI   R5, #0        ; Reset Column of A
18:         CMP    R4, R0
19:         B.LT   -13           ; LOOP
```

Exercício 1

Simule manualmente a execução do programa assembler apresentado na secção de código 2, e preencha o ficheiro de Excel fornecido como anexo a este guia que contém uma tabela semipreenchida como a apresentada em baixo. Este ficheiro deve ser submetido no final do laboratório e destina-se a ser processado automaticamente pelo que apenas deve preencher as zonas a isso destinadas e não alterar o resto do ficheiro. De seguida apresenta-se uma descrição formato da tabela no ficheiro.

		Level 1										Cache																
												Level 2																
												Via 0								Via 1								
	R/W	**L0**	**L1**	**L2**	**L3**	**L4**	**L5**	**L6**	**L7**			**L0**	**L1**	**L2**	**L3**	**L4**	**L5**	**L6**	**L7**	**L0**	**L1**	**L2**	**L3**	**L4**	**L5**	**L6**	**L7**	**R**	**W**
A0	R	A0								M		A0																M	
B0	W	B0								M										B0								M	
A1	R	A0								M	E	A0								B0								H	H
B4	W			B4						M				B4														M	
A2	R		A2							M			A2															M	
B8	W					B8				M						B8												M	
A3	R		A2							H																			
B12	W							B12		M								B12										M	
A4	R			A4						M	E			B4								A4						M	H
B1	W	B0								M										B0								H	
A5																													
B5																													
A6																													
B9																													
A7																													
B13																													
A8																													
B2																													
A9																													
B6																													
A10																													
B10																													
A11																													
B14																													
A12																													
B3																													
A13																													
B7																													
A14																													
B11																													
A15																													
B15																													

- **Primeira coluna**: Cada umas das linhas da tabela corresponde a um acesso à memória. Nesta coluna descreve qual a matriz (A ou B) e qual o elemento da matriz que está a ser acedido. Assim A0 será o primeiro elemento da matriz A, A1 o segundo elemento da matriz A, A4 o quinto elemento da matriz A (ou o primeiro elemento da segunda linha da matriz A, já que a matriz está organizada por linhas. B0 será o primeiro elemento da matriz B, B1 o segundo

elemento da matriz B, etc. É simples determinar qual a linha da cache que está a ser acedida apenas pela posição do elemento de cada uma das matrizes.

- **Segunda coluna (R/W)**: Indica se o acesso à memória é de leitura ou de escrita. Os acessos à matriz A são sempre de leitura e os acessos à matriz B são sempre de escrita.
- **Colunas 3 a 10 (Level 1, L0 a L7)**: Indicam a evolução do conteúdo das linhas (linha 0 à linha 7) da cache de nível 1 ao longo do tempo. Um espaço vazio indica que o conteúdo não se alterou nesse acesso á memória. Quando não estão vazias, indicam o primeiro elemento da matriz A ou B que é armazenado nessa linha. Note que cada linha guarda dois valores de 32 bits, logo dois elementos da matriz. Apenas devem ser preenchidas quando há um acesso a essa linha da cache.
- **Coluna 11:** Indicam um miss ou um hit na cache. M – miss. H – Hit.
- **Coluna 12:** Um "E" nesta coluna indica que houve uma modified line Eviction no cache. Ou seja, que ocorreu um miss e a linha da cache correspondente continha dados que haviam sido modificados (valid bit = 1 e dirty bit = 1), pelo que é necessário escrever essa linha no nível seguinte da hierarquia de memória antes de carregar o novo valor. Um modified line Eviction resulta num write e num read no nível seguinte da hierarquia da memória.
- **Colunas 13 a 28: (Level , Via 0 e Via 1 L0 a L7):** Tem o mesmo significado que as colunas 3 a 10 mas neste caso para as duas vias da cache de nível 2.
- **Coluna 29 (R):** Indica um miss ou um hit numa operação de leitura na cache de nível 2. Note que um acesso á memória pode resultar num read ou num write e num read na cache de nível 2 (o último caso corresponde a uma eviction), por isso a necessidade desta e da próxima coluna. M – miss. H – Hit.
- **Coluna 20 (W):** Indica um miss ou um hit numa operação de escrita na cache de nível 2. M – miss. H – Hit.

Por baixo da tabela, no mesmo ficheiro, caracterize o desempenho das caches L1 e L2 em *Número de Acessos, Read Hits, Read Misses, Write Hits, Write Misses e Modified Line Evictions*, preenchendo as células à esquerda do campo correspondente por baixo da coluna manual.

	Manual	Simulação
Acessos		
Read Hits		
Read Misses		
Write Hits		
Write Hits		
Evictions		

Exercício 2

Pretende-se agora verificar o funcionamento do programa executando-o no sistema de processamento. No projeto Vivado fornecido em conjunto com este guia de laboratório, o ficheiro `InstMemory.vhd` contém a **Memória de Instruções** (a hierarquia para este ficheiro é `CPU`→`Core`→`Instruction Fetch`→`IMEM`) e foi pré preenchido de acordo com o código máquina do programa assembly que faz a transposta de uma matriz.

Teste o funcionamento do sistema. Faça uma simulação do *testbench* **tbCPU** (situado em *Simulation Sources*→`sim_1`), utilizando o ficheiro de configuração `tbCPU_behav.wcfg` (situado em *Simulation Sources*→`sim_1`→*Waveform Configuration Files*). Aumente o tempo simulação para 30us. No ficheiro excel preencha agora as colunas "Simulação" da tabela com os hits miss e evictions com os resultados da simulação do Vivado para as caches de nível 1 e nível 2. Os sinais com estes valores são os últimos do conjunto de sinais apresentados na simulação.

	Manual	Simulação
Acessos		
Read Hits		
Read Misses		
Write Hits		
Write Hits		
Evictions		

Confirme que os resultados são iguais aos que que determinou através da simulação manual da execução do código. Caso os resultados não coincidam, deve analisar o resultado da simulação e comparar com o esperado.

Exercício 3

Proponha alterações à arquitetura das ***Caches L1 e L2*** (sem aumentar a sua capacidade total) de modo a reduzir o número total de Misses e aumentar o desempenho do sistema.