ARQUITETURA DE COMPUTADORES

Nome:_____Nº_____

Nome:_____Nº_____

Grupo: _____ Dia:_____ Hora:_____ Sala: _____ Docente:_____

Laboratório 5 – Análise de um Processador Pipelined

Este laboratório destina-se a consolidar conhecimentos de introdução à arquitetura pipeline. No laboratório será utilizada uma descrição em VHDL do processador ilustrado na Figura 1. O trabalho irá analisar a arquitetura e testar o código VHDL fornecido, de acordo com os exercícios deste guia de laboratório.

O trabalho descrito neste guia 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 no Fénix um ficheiro zip com: um ficheiro Word com as repostas às perguntas do guia e projeto do Vivado. O ficheiro com o projeto Vivado deve ser criado utilizando o comando File→Project→Archive do menu.

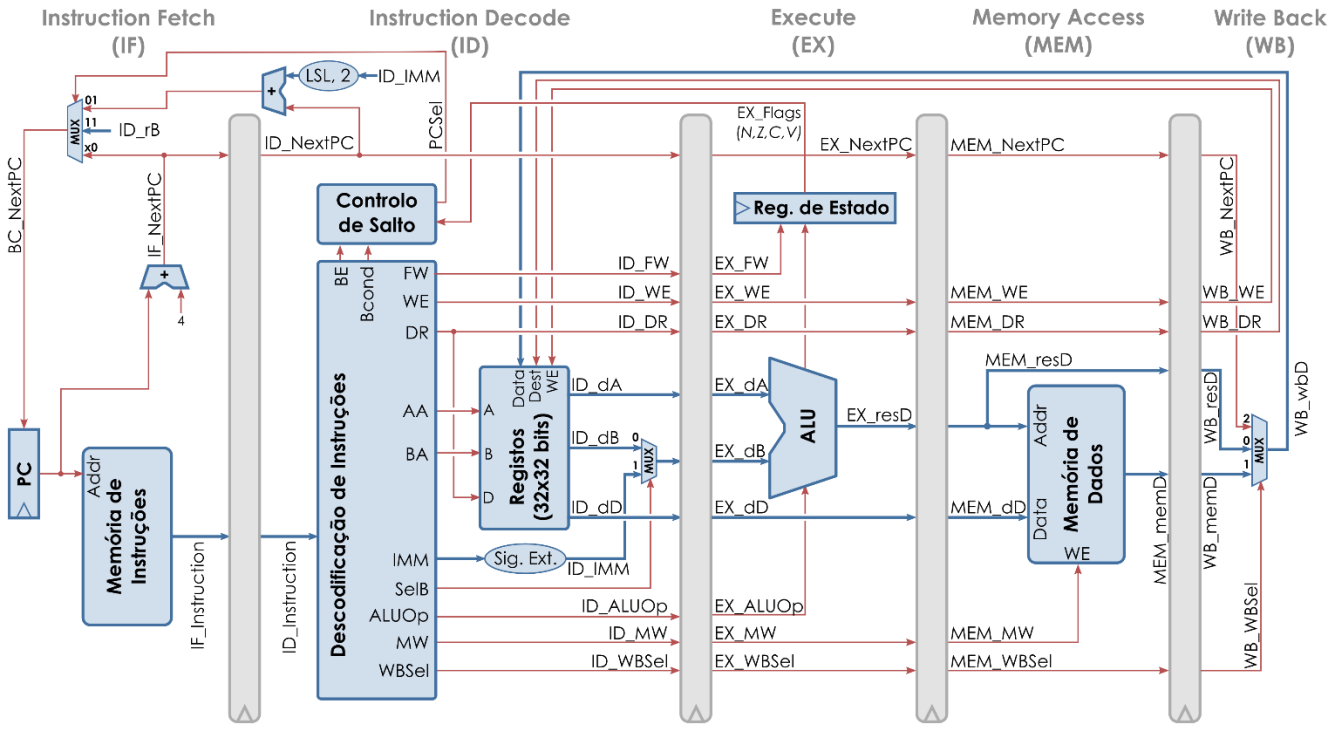

Figura 1 - Arquitetura do Processador

Arquitetura do Processador Pipelined

Pretende-se analisar a arquitetura do processador ilustrada na Figura 1. O processador é implementado por uma arquitetura com *pipeline* de 5 andares (IF, ID, EX, MEM e WB) e 32 bits para instruções e dados. A unidade de armazenamento (no andar ID) é composta por um banco de 32 registos de inteiros de 32 bits (R0-R31), com LR = R30 e R31 = 0. O processador está ligado a duas memórias endereçadas ao byte, com alinhamento a palavras de 4 bytes:

- ***Memória de Instruções* –** capacidade 256 x 32 bits (para instruções);
- ***Memória de Dados* –** capacidade 256 x 32 bits (para dados).

O processador está totalmente implementado e encontra-se no ficheiro `PipelinedProcessor.zip` disponibilizado na página da cadeira. **Para analisar o processador deve estudar o código VHDL fornecido** (**abra e estude todos os ficheiros de código fonte).** A organização e descrição dos ficheiros mais relevantes fornecidos no projeto estão apresentadas na Tabela 1.

Tabela 1 - Organização e descrição dos ficheiros fornecidos no projeto

Implementação: ***Design Sources***	
SingleCycle	Processador (core) de Ciclo Único
↳ **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
↳ **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

Finalmente, o conjunto de instruções suportado pelo processador está detalhado nas Tabelas 2 e 3 (página seguinte).

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

Exercício 1

Analise o segmento de código *Assembly* apresentado abaixo. Tendo em consideração a arquitetura do processador da Figura 1 e a informação das Tabelas 2 e 3, identifique todos os conflitos de controlo e de dados. Nos conflitos de dados indique a instrução (número da linha e OPCODE) que escreve e a instrução que lê e qual o operando que provoca o conflito. Nos conflitos de controlo indique qual a instrução que provoca o conflito.

```
0:  MOVI R0, #0
1:  MOVI R1, #0
2:  MOVI R2, #7
3:  MOVI R3, #64
4:  LDR  R4,[R31,R1]
5:  CMP  R4, R2
6:  B.LT +1
7:  EOR  R0, R0, R4
8:  ADDI R1,R1,#4
9:  CMP  R1,R3
10: B.LT -7
11: STR  R0,[R31,R1]
```

Resposta:

Exercício 2

Reescreva o código do exercício anterior de modo a resolver os conflitos de controlo e dados que identificou por software (introduzindo NOPs e/ou reordenando instruções). Apresente a codificação para cada uma das instruções no programa de acordo com a informação das Tabelas 2 e 3 (indique explicitamente indiferenças com "X").

		Codificação da Instrução				
	Instrução	**I(31:26)**	**I(25:21)**	**I(20:16)**	**I(15:11)**	**I(10:0)**
0:						
1:						
2:						
3:						
4:						
5:						
6:						
7:						
8:						
9:						
10:						
11:						
12:						
13:						
14:						
15:						
16:						
17:						
18:						
19:						
20:						
21:						
22:						
23:						
24:						
25:						
26:						
27:						
28:						
29:						
30:						
31:						

Exercício 3

Calcule o número de ciclos de relógio necessários para executar o código que escreveu no exercício anterior (apresente todos os cálculos que efetuar, devidamente justificados).

Resposta:

Pretende-se agora verificar o funcionamento do programa executando-o no processador *pipelined*. Nas *Design Sources* do Vivado, abra o ficheiro `InstMemory.vhd`, o qual contém a **Memória de Instruções** (a hierarquia para este ficheiro é `PipelinedProcessor`→`InstructionFetch`→ `IMEM`). Preencha a **Memória de Instruções** de acordo com a tabela que preencheu no exercício anterior. Este código deve ser introduzido, editando o espaço entre as linhas 50 e 54. No ficheiro VHDL não utilize o valor 'X' para indicar indiferenças, substitua-o pelo valor lógico 0 ou 1.

Teste o funcionamento do processador *pipelined*. Faça uma simulação do *testbench* **tbPipelinedProcessor** (situado em *Simulation Sources*→`sim_1`), utilizando o ficheiro de configuração `tbPipelinedProcessor_behav.wcfg` (situado em *Simulation Sources*→ `sim_1`→*Waveform Configuration Files*). **Analise** com cuidado a simulação verificando a evolução dos sinais ao longo do tempo e **verifique** que tudo decorre como esperado. Tenha em atenção que os sinais apresentados são os mesmos que os indicados no diagrama do processador no início do guia de laboratório. Observe por exemplo a variação do PC. A partir do valor deste pode determinar qual é a instrução que está a ser executada num determinado instante e verificar se o resultado é o esperado. **Apresente** a simulação do processador completo ao docente durante o horário de laboratório.

Indique o instante de tempo em que termina a execução do programa.

t_{ns} =	

Compare o número de ciclos de relógio que calculou acima com o tempo que obteve através da simulação. Para o efeito, assuma que a frequência de operação do processador na simulação é de 20 MHz.

Resposta:

Exercício 4

De acordo com o esquema do processador da Figura 1, indique todas as alterações à arquitetura necessárias para resolver os conflitos de controlo e de dados por hardware. Identifique qualquer conflito que não seja possível resolver sem aumentar significativamente o caminho crítico do processador. Para indicar alterações à arquitetura use a seguinte notação: forward de sinal_de_origem para sinal_de_destino, indicando todos os sinais necessários.

Resposta: