ARQUITETURA DE COMPUTADORES

2º Trabalho de Laboratório

Conceção e análise e um processador

Objetivo: Pretende-se que os alunos compreendam a metodologia usada na síntese, implementação e programação de um processador básico, constituído pelos 5 estágios convencionais de um MIPS: *Instruction Fetch* (IF), *Instruction Decode* (ID), *Execute* (EX), *Memory* (MEM) e *Write-Back* (WB). O trabalho terá uma duração de 2 semanas, devendo o relatório ser submetido em formato PDF no Fénix no final da segunda semana de laboratório, ao qual deverá ser anexado os ficheiros de projecto (ver secção 6). O trabalho deverá ser preparado fora do horário de laboratório, destinando-se as 3 horas de laboratório à resolução de eventuais problemas e à demonstração do trabalho realizado.

1 INTRODUÇÃO

Pretende-se projetar um processador de 32 bits, com suporte para um número reduzido de instruções. O processador tem por base a Unidade de Processamento (UP) do primeiro trabalho de laboratório, embora esta tenha sido estendida de forma a suportar operandos de 32 bits. Assim, os registos, unidades funcionais e memória de dados operam agora sobre 32 bits. Contudo a estrutura do circuito foi modificada, sendo agora composta pelos seguintes estágios:

- **IF (*Instruction Fetch*)** – Leitura da instrução indicada pelo PC (*Program Counter*) da memória de instruções;
- **ID (*Instruction Decode*)** – Descodificação da instrução e leitura dos operandos;
- **EX (*Execute*)** – Execução da instrução, i.e., cálculo do resultado (Unidade Funcional) ou alteração do fluxo de instruções (teste de condição e realização de um salto na Unidade de Controlo de Salto);
- **MEM (*Memory*)** – Leitura ou escrita de um valor na memória de dados;
- **WB (*Write-Back*)** – Escrita do resultado da instrução na Unidade de Armazenamento *(Register File)*.

A estrutura simplificada do processador encontra-se ilustrada na Figura 1. Para efeitos de representação, o *Register File* foi dividido em duas partes: leitura dos operandos (estágio de ID); e escrita do resultado (estágio de WB).

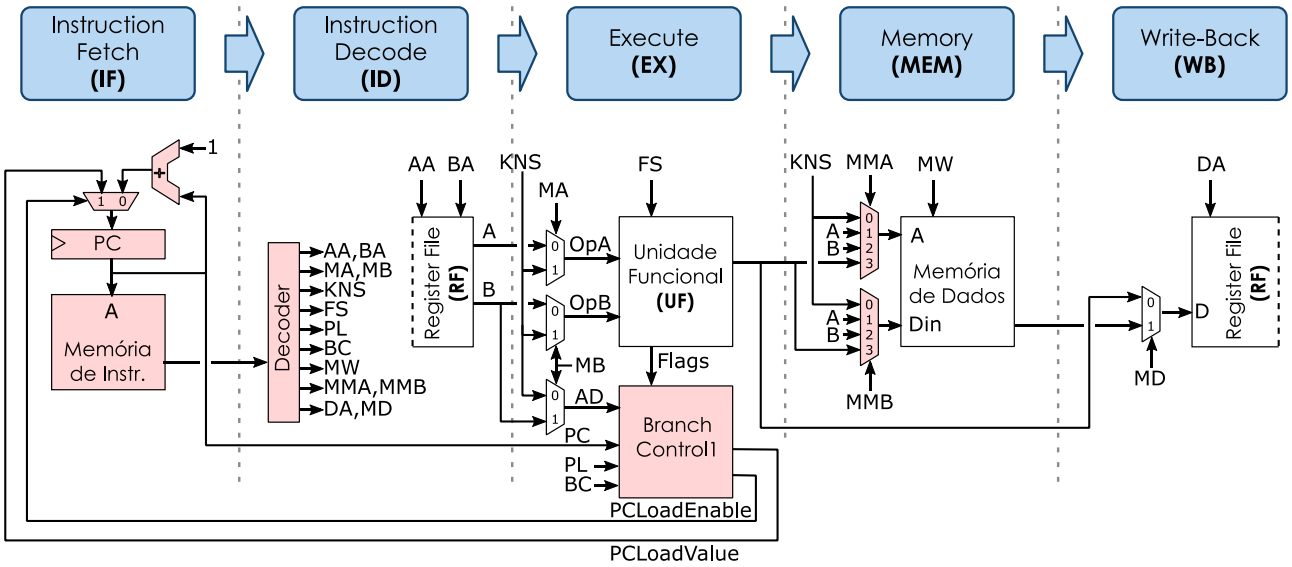

Figura 1 - Esquema simplificado do processador. A branco apresentam-se os blocos existentes na Datapath do laboratório 1 e a vermelho os blocos adicionados para o desenvolvimento do trabalho 2.

A arquitetura do conjunto de instruções (ISA – *Instruction Set Architecture*) segue o princípio de um processador RISC (*Reduced Instruction Set Computer*). As instruções são codificadas numa palavra de dimensão fixa (32-bits), sendo estas compostas por um campo de *opcode* (identificação da operação), um registo de destino (DR) e dois registos fonte (SA e SB). Contudo, em casos excecionais a palavra de instrução pode conter ainda outros campos. A estrutura completa da arquitetura do conjunto de instruções é elaborada na secção 3.

2 ARQUITETURA DO PROCESSADOR

O circuito de processamento de dados (blocos a branco representados na figura 1) tem por base o esquema utilizado no trabalho de laboratório 1, e cuja funcionalidade se apresenta de seguida.

2.1 Circuito de processamento de dados (Datapath)

a) **Unidade de Armazenamento (UA)**:

1. Apresenta um total de 16 registos;
2. O registo R0 tem sempre o valor 0 (mesmo após uma escrita de um outro valor em R0).

b) A **Unidade Funcional (UF)** realiza as operações indicadas na tabela 2.

Tabela 1 - Funcionamento da Unidade Funcional (UF).

Tipo	*Operação (mnemónica)*	*Entradas*		*Saídas*				
		Operandos	FSUF	Dados (D)	Flags válidas			
Aritmética	ADD	A , B	0000	A + B	Z	N	C	V
	ADD+	A , B	0001	A + B + 1	Z	N	C	V
	SUB-	A , B	0010	A – B – 1	Z	N	C	V
	SUB	A , B	0011	A – B	Z	N	C	V
Lógica	AND	A , B	0100	A *(and)* B	Z	N		
	NAND	A , B	0101	A *(nand)* B	Z	N		
	OR	A , B	0110	A *(or)* B	Z	N		
	NOR	A , B	0111	A *(nor)* B	Z	N		
	XOR	A , B	1000	A *(xor)* B	Z	N		
	XNOR	A , B	1001	A *(xnor)* B	Z	N		
Deslocamento	SHL	B	1010	B(30:0),0	Z	N	C	
	SHR	B	1011	0,B(31:1)	Z	N	C	
	SHLA	B	1100	B(30:0),0	Z	N	C	V
	SHRA	B	1101	B(31),B(31:1)	Z	N	C	V
	ROL	B	1110	B(30:0),B(31)	Z	N	C	
	ROR	B	1111	B(0),B(31:1)	Z	N	C	

Flags: V – Overflow; C – Carry; N – Negative; Z – Zero

2.2 Controlo de execução de instruções

O controlo de fluxo de instruções do processador é realizado pelos seguintes blocos:

1. Registo do contador de programa (`PCReg`, estágio de IF) que guarda o endereço da instrução a ser executada.
2. Memória de instruções (`InstMem`, estágio de IF), cuja saída é a instrução indicada pelo PC.
3. Estágio de ID, responsável por gerar os sinais de controlo para a execução de cada instrução, nomeadamente:
   - Sinais para controlo do circuito de dados: AA, BA, DA, FS, MA, MB, KNS, MMA, MMB, MW, MD;
   - Sinais de controlo do fluxo de instruções: PL, BC.

   A descodificação dos sinais deverá ser feita de acordo com a secção 3 e implementada através de microprogramação da memória `decode_memory` (estágio de ID).
4. Unidade de Controlo de Salto (`UCS`, estágio de EX), que decide o endereço da próxima instrução a ser executada. Este bloco tem como operandos os seguintes sinais:
   - PC ➔ Endereço da instrução que está a ser executada.
   - PL ➔ Indica se a instrução a ser executada é de controlo (PL=1) ou de dados (PL=0).
   - BC ➔ Define a condição de salto de acordo com a Tabela 4; a condição de salto é determinada verificando o valor das flags após uma operação de subtração entre os operandos OpA e OpB.
   - AD ➔ Define o endereço relativo (offset) de salto; no caso da condição de salto ser verdadeira, o endereço da próxima instrução é determinado da seguinte forma: NextPC=PC+AD.
   - PCLoadEnable ➔ Indica que a condição de salto é verdadeira (ou que o salto é incondicional).
   - PCValueValue ➔ Endereço da instrução que deve ser carregada no registo PC caso a condição de salto seja verdadeira (PCLoad=1).

3 CONJUNTO DE INSTRUÇÕES

O conjunto de instruções que o processador deverá suportar está descrito na tabela 3. Cada instrução é codificada com um conjunto de 32 bits, sendo que os 6 bits mais significativos indicam o código da instrução, e os restantes os operandos da instrução (ver coluna 2).

Tabela 3 - Instruções a implementar

Mnemónica	Formato da instrução (bits)					Descrição
	I(31:26)	I(25:22)	I(21:18)	I(17:14)	I(13:0)	
NOP	000000	0000000h				*No Operation*
ADD	000000	DR	SA	SB	-	R[DR] ← R[SA] + R[SB]
ADDI	000001	DR	SA	SIMM18(1)		R[DR] ← R[SA] + SIMM18(1)
SUB	000010	DR	SA	SB	-	R[DR] ← R[SA] – R[SB]
SUBI	000011	DR	SA	SIMM18(1)		R[DR] ← R[SA] – SIMM18(1)
AND	000100	DR	SA	SB	-	R[DR] ← R[SA] *and* R[SB]
ANDIL	000101	DR	SA	- -,IMM16		R[DR] ← R[SA] *and* (FFFFh,I(15:0))
ANDIH	000110	DR	SA	- -,IMM16		R[DR] ← R[SA] *and* (I(15:0),FFFFh)
NAND	000111	DR	SA	SB	-	R[DR] ← R[SA] *nand* R[SB]
OR	001000	DR	SA	SB	-	R[DR] ← R[SA] *or* R[SB]
ORIL	001001	DR	SA	- -,IMM16		R[DR] ← R[SA] *or* (0000h,I(15:0))
ORIH	001010	DR	SA	- -,IMM16		R[DR] ← R[SA] *or* (I(15:0),0000h)
NOR	001011	DR	SA	SB	-	R[DR] ← R[SA] *nor* R[SB]
XOR	001100	DR	SA	SB	-	R[DR] ← R[SA] *xor* R[SB]
XNOR	001101	DR	SA	SB	-	R[DR] ← R[SA] *xnor* R[SB]
SHL	001110	DR	-	SB	-	R[DR] ← SHL R[SB]
SHR	001111	DR	-	SB	-	R[DR] ← SHR R[SB]
ROL	010000	DR	-	SB	-	R[DR] ← ROL R[SB]
ROR	010001	DR	-	SB	-	R[DR] ← ROR R[SB]
SHLA	010010	DR	-	SB	-	R[DR] ← SHLA R[SB]
SHRA	010011	DR	-	SB	-	R[DR] ← SHRA R[SB]
LD	010100	DR	SA	SB	-	R[DR] ← M[R[SA]+R[SB]]
LDI	010101	DR	SA	SIMM18(1)		R[DR] ← M[R[SA]+SIMM18]
ST	010110	SIMM18(2)	SA	SB	SIMM18(2)	M[R[SA]+SIMM18] ← R[SB]
B.*cond*	010111	BC(4)	SA	SB	SIMM14(3)	PC ← PC + SIMM14 se R[SA] *cond* R[SB] (ver Tabela 4)
BI.*cond*	011000	BC(4)	SA	SB	SIMM14(3)	PC ← PC + R[SB] se R[SA] *cond* SIMM14 (ver Tabela 4)

Nota: A notação usada na tabela é a seguinte: 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.

(1) O campo SIMM18 indica que os 18 bits extraídos da palavra de instrução, <I(17:0)>, correspondem a um número representado em complemento para 2 e que deve ser estendido para 32-bits, de forma a gerar o sinal KNS.

(2) O campo SIMM18 indica que os 18 bits extraídos da palavra de instrução, <I(25:22),I(13:0)> correspondem a um número representado em complemento para 2 e que deve ser estendido para 32-bits, de forma a gerar o sinal KNS.

(3) O campo SIMM14 indica que os 14 bits extraídos da palavra de instrução, <I(13:0)> correspondem a um número representado em complemento para 2 e que deve ser estendido para 32-bits, de forma a gerar o sinal KNS.

(4) A condição de salto é determinada de acordo com a tabela 3, tendo por base os operandos OpA=R[SA] e OpB=R[SB] para a instrução de BR, e OpA=R[SA] e OpB=SIMM14 no caso da instrução BRI.

Tabela 4 – Funcionamento esperado da Unidade de Controlo de Salto (UCS). Os operandos OpA e OpB correspondem aos sinais indicados na Figura 1 com o mesmo nome.

Mnemónica	*PL*	*BC*	*Condição*	*NextPC*	*Operação*
-	0	X	-	PC + 1	Incrementa o contador de programa
B	1	000X	-	PC + AD	Salto Incondicional
B.EQ / BI.EQ **(branch if equal)**	1	0010	OpA=OpB OpA≠OpB	PC + AD PC + 1	Salto condicionado aos operandos OpA e OpB serem iguais (teste da flag Z)
B.NE / BI.NE **(branch if not equal)**	1	0011	OpA≠OpB OpA=OpB	PC + AD PC + 1	Salto condicionado aos operandos OpA e OpB serem diferentes (teste da flag Z)
B.GT / BI.GT **(branch if greater than)**	1	0100	OpA>OpB OpA≤OpB	PC + AD PC + 1	Salto condicionado a OpA ser maior que o OpB (teste das flags Z e N)
B.GE / BI.GE **(branch if greater or equal)**	1	0101	OpA≥OpB OpA<OpB	PC + AD PC + 1	Salto condicionado a OpA ser maior ou igual a OpB (teste das flags Z e N)
B.LT / BI.LT **(branch if lower than)**	1	0110	OpA<OpB OpA≥OpB	PC + AD PC + 1	Salto condicionado a OpA ser menor que OpB (teste das flags Z e N)
B.LE / BI.LE **(branch if lower or equal)**	1	0111	OpA≤OpB OpA>OpB	PC + AD PC + 1	Salto condicionado a OpA ser menor ou igual a OpB (teste das flags Z e N)

3.1 Notação Assembly

A notação usada para a descrição Assembly das instruções (típica em processadores RISC) é a seguinte:

`<Mnemónica de operação>  <Registo de destino>,<registo A>,<registo B>`

A que corresponde uma operação do tipo:

`<Registo de destino>` ← `<registo A>  <operação> <registo B>`

Por exemplo, a instrução "`ADD R3,R4,R5`" corresponde à operação `R3←R4+R5`. As exceções a esta regra dizem respeito a:

- Instruções de deslocamento, para as quais não existe um registo de fonte A (p. ex.: a instrução "`ROL R3,R2`" corresponde a `R3←ROL R2`);
- Instruções com imediatos (constantes), nomeadamente ADDI, SUBI, ANDIH, ANDIL, ORIH, ORIL (p. ex. a instrução "`ADDI R2,R0,#-3`" corresponde a `R2←R0+(-3)`).
- Instruções com a memória com endereço indexado (i.e., determinado através da soma de dois operandos), nomeadamente LD, LDI e ST (p. ex. a instrução "`LDI R3,(R2+#3)`" corresponde a `R3←M[R2+3]`, enquanto que a instrução "`ST (R2+#3),R3`" corresponde a `M[R2+3]←R3`).
- instruções de controlo de fluxo de instruções (i.e., controlo de salto ou *branch*).

No caso das instruções de branch, a notação é:

`B.cond  <registo (operando) A>,<registo (operando) B>,<deslocamento>`

a que corresponde uma operação do tipo:

Se (`<registo A> <cond> <registo B>`) então `PC←PC+deslocamento`, caso contrário `PC←PC+1`.

ou:

`BI.cond  <registo (operando) A>,<imediato (operando B)>,<registo de deslocamento>`

a que corresponde uma operação do tipo:

Se (`<registo A> <cond> <imediato>`) então `PC←PC+<registo B>`, caso contrário `PC←PC+1`.

Por exemplo, a instrução "`B.GT R7,R0,#7`" verifica se R7 é maior ou igual a R0 (GT≡*greater than*), através da subtração entre os operandos R7 e R0. Se a condição for verdadeira, realiza a operação `PC←PC+7`. Caso contrário, realiza a operação típica sobre o registo PC, i.e., `PC←PC+1`. A tabela 4 descreve as operações de controlo de fluxo que o processador deverá suportar.

4 IMPLEMENTAÇÃO DE UM PROCESSADOR DE CICLO ÚNICO (SEMANA 1)

Pretende-se que durante a primeira semana de laboratório implemente um conjunto limitado de instruções. Para tal no Vivado deverá selecionar como ficheiro de topo de *Design Sources*, o módulo *SingleCycle*.

Tabela 5 – Instruções a implementar por turno da semana

Segunda-feira	Terça-feira	Quarta-feira	Quinta-feira	Sexta-feira
ADD, ADDI, XOR, SHL B, B.GE, B.LT	SUB, SUBI, ADD, ROL, B, B.EQ, B.NE	ADD, SUBI, LDI, ST, B, B.EQ, B.NE	ORIL, AND, ADD, SHR, LD, B.NE, B.EQ, B	ADD, ADDI, AND, XOR, LD, SHRA, BI.NE, B

4.1 Descodificador de Instruções e unidade de controlo de salto

(a) **(2 Val.)** Nas *design sources* do Vivado, abra o ficheiro *InstructionDecode.vhd*, o qual contem a descodificação das instruções (a hierarquia para este ficheiro é SingleCycle→ID). Tendo em consideração o circuito disponibilizado, analise o funcionamento do estágio de ID e identifique a função de todos os sinais à saída da memória de descodificação (`decode_memory`).

(b) **(3 Val.)** De acordo com o seu turno de laboratório, e para cada operação indicada na tabela 5, indique o valor dos sinais de controlo que deverá colocar na memória de descodificação. Para tal sugere-se que preencha uma tabela como se ilustra de seguida:

OpCode	Mnemónica	PL	dAA	dBA	dDA	FS	KNSSel	MASel	MBSel	MMA	MMB	MW	MDSel
…	…	…	…	…	…	…	…	…	…	…	…	…	…
5	ANDIL	0	X	X	X	0100b	101b	00b	01b	X	X	0	00b
6	ANDIH	0	X	X	X	0100b	111b	00b	01b	X	X	0	00b
…	…	…	…	…	…	…	…	…	…	…	…	…	…

Note que os sinais dAA, dBA e dDA correspondem a saidas da memória de descodificação e não aos sinais AA, BA e DA apresentados na Figura 1. Tenha ainda em atenção que as instruções de NOP, ST e B.*cond* não devem escrever em nenhum dos registos R1-R15.

Altere o conteúdo da memória de descodificação (edite o ficheiro *InstructionDecode.vhd*, linhas 65 a 109, descomentando as linhas que achar convenientes) de forma a garantir a correta execução das instruções indicadas na tabela 5. No ficheiro VHDL não utilize os valores 'x' ou '-' para indicar indiferenças. Substitua as indiferenças pelo valor lógico 0 ou 1. Contudo, deverá indicar explicitamente estas indiferenças no relatório e justificar o valor atribuído a cada sinal de controlo.

4.2 Unidade de Controlo de Salto

(3 Val.) Projete a Unidade de Controlo de Salto (UCS) de acordo com a Tabela 4. Implemente a unidade editando o ficheiro *BranchControl.vhd* (nas *design sources* do Vivado, a hierarquia para este ficheiro é SingleCycle→EX→UCS). Faça todas as simulações que achar convenientes de forma a garantir o seu correto funcionamento. Para tal deverá criar um novo ficheiro de simulação que contenha apenas o módulo *BranchControl*, e que teste cada uma das combinações dos sinais PL, BC e Flags (o teste não tem de ser exaustivo, mas deverá ser representativo). Selecione este ficheiro de simulação como topo para simulação e verifique que a sua implementação está correta. No relatório descreva sucintamente o processo de síntese e de validação do funcionamento.

4.3 Teste das instruções

(2 Val.) Valide a correta implementação das instruções executando o programa de teste indicado na tabela 6. Este código já se encontra microprogramado (embora em comentário) na memória de instruções (ficheiro

InstructionMemory.vhd, hierarquia IFetch→InstMem). Assim, deverá comentar a linha 14 e descomentar o código correspondente ao seu turno de laboratório editando as linhas 16 a 121. **Para executar o programa de teste seleccione o ficheiro de simulação *TestSingleCycle.vhd* como ficheiro de topo na simulação**.

Determine ainda o número de ciclos necessários à correta execução do troço de código indicado até à execução (inclusive) da instrução "`B #0`".

Nota: se preferir usar a configuração das formas de onda disponibilizada (opcional), na janela *Sources,* seguindo a hierarquia *Simulation Sources→sim_1→Waveform Configuration File*, ative (Enable File) o ficheiro testSingleCycle_behav.wcfg.

Tabela 6 – Código de teste por dia da semana (pré-implementados na memória de instruções).

Segunda-feira	Terça-feira	Quarta-feira	Quinta-feira	Sexta-feira
ADDI R7,R0,#5	SUBI R3,R0,#1	SUBI R1,R0,#-1	ORIL R2,R0,#15	ADDI R1,R0,#-2876
ADDI R3,R0,#1	SUBI R6,R0,#-1	ST (R0+1),R1	ORIL R1,R0,#FE6Ch	ADDI R7,R0,#-8
ADDI R4,R0,#1	SUB R1,R3,R6	ST (R0+2),R1	ORIL R3,R0,#0	ADDI R2,R0,#15
ADD R2,R3,R4	SUBI R4,R0,#3	ADD R2,R1,R1	B.EQ R1,R0,#9	XOR R3,R0,R0
ADDI R1,R0,#2	B.EQ R4,R0,#7	SUBI R3,R0,#-2	AND R4,R1,R2	XOR R4,R1,R2
B.GE R1,R7,#7	SUB R2,R3,R6	SUBI R4,R0,#-5	LD R5,(R4+R0)	AND R4,R4,R2
ADD R5,R3,R4	SUB R3,R0,R6	B.LE R4,R3,#7	ADD R3,R3,R5	LD R5,(R4+R0)
XOR R3,R4,R0	SUB R6,R0,R2	LDI R5,(R3+#-1)	SHR R1,R1	SHRA R1,R1
XOR R4,R5,R0	SUBI R4,R4,#-1	ADD R1,R1,R5	SHR R1,R1	SHRA R1,R1
ADD R2,R2,R5	ADD R1,R1,R2	ADD R2,R2,R1	SHR R1,R1	SHRA R1,R1
ADDI R1,R1,#1	B.NE R4,R0,#-5	ST (R3+#1),R1	SHR R1,R1	SHRA R1,R1
B.LT R1,R7,#-5	SUB R2,R0,R1	SUBI R3,R3,#-1	B.NE R1,R0,#-7	ADD R3,R3,R5
B #0	B #0	B.GT R4,R3,#-5	B #0	BI.NE R1,#-1,R7
		B #0		B #0

5 ANÁLISE DE UM PROCESSADOR *PIPELINED* (SEMANA 2)

5.1 Cálculo teórico do desempenho do processador de ciclo único

(2 Val.) Considere os seguintes tempos de propagação para os 5 estágios do processador:

- IF (Read PC from register→Memory→Instruction): 35ns
- IF (PC →Adder→MUX→Write on PC Register): 15ns
- ID (Instruction→Decoder→RF→A,B): 30ns
- EX (A,B→UF→Data): 30ns
- EX (A,B→UF→Branch Control→PCLoadEnable,PCLoadValue→MUX→Write on PC Register): 40ns
- MEM (A,B,D→Write to Memory): 30ns
- MEM (A,B,D→Read from Memory): 40ns
- WB (Data→Write to register file): 15ns

Determine a frequência máxima de relógio para o processador. Justifique.

5.2 Frequência de relógio de um processador *pipelined*

(3 Val.) Considere que introduz registos entre os andares de pipeline, tal como ilustrado na Figura 2. Determine a nova frequência de relógio, considerando que os flip-flops entre estágios do processador têm como característica: T_{Setup}=$T_{Propagação}$=1ns. Estime ainda o aumento teórico de desempenho (aceleração ou *speedup*), o qual é dado por:

$$Speedup = \frac{T_{Ciclo\ Único}}{T_{Pipelined}}$$

5.3 Execução de um troço de código num processador *pipelined*

No Vivado selecione o ficheiro *FiveStagePipeline.vhd* e coloque-o como ficheiro de topo para síntese. Se ativou o ficheiro *testSingleCycle_behav.wcfg* no ponto 4.3, desative-o (em alternativa pode agora ativar o ficheiro testFiveStagePipeline_behav.wcfg). Selecione de seguida o ficheiro *testFiveStagePipeline.vhd* e coloque-o como ficheiro de topo para simulação. Simule o funcionamento do processador com o código usado na alínea 4.3.

1. **(2 Val.)** Identifique todos os erros na execução do troço de código indicado e justifique a sua ocorrência.
2. **(2 Val.)** Proponha e implemente todas as alterações que achar adequadas de forma a garantir a correta execução do troço de código. Embora não seja necessário, pode introduzir novas instruções ao processador, devendo nesse caso microprogramá-las corretamente (preencha a memória de descodificação de instruções). Pode também propor alterações ao processador de forma a corrigir os erros identificados (não recomendado).
   Nota: todas as alterações ao código deverão respeitar a funcionalidade do algoritmo implementado.
3. **(1 Val.)** Determine o aumento real de desempenho, tendo em consideração o número total (real) de ciclos para a execução do troço de código.

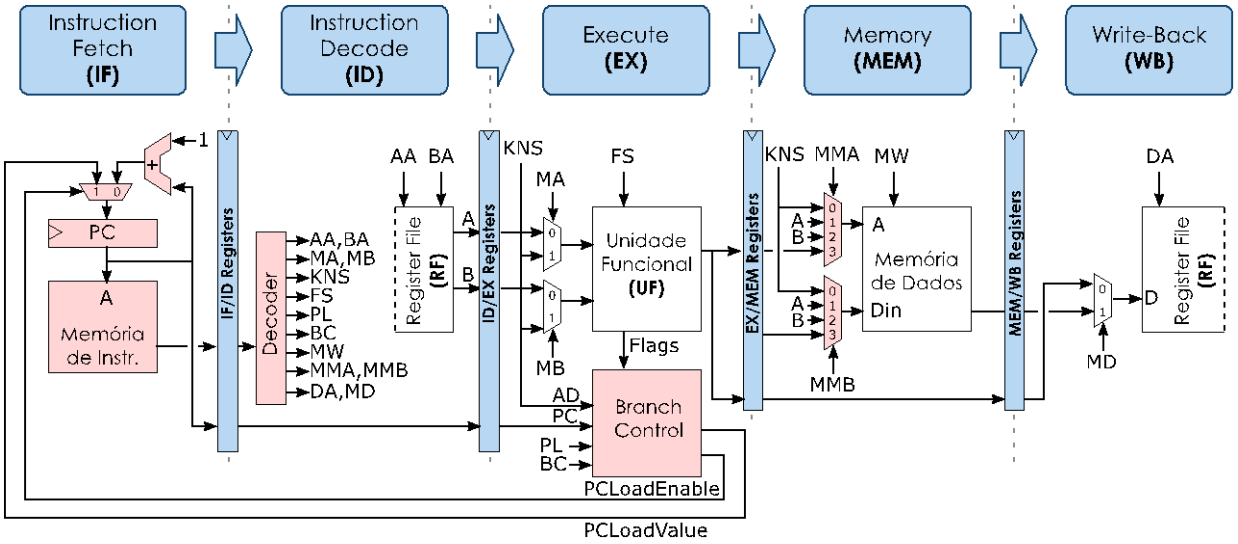

Figura 2 - Esquema simplificado do processador pipelined. A diferença relativamente à figura 1, encontra-se na introdução de registos entre os estágios do processador. Com excepção dos sinais à saida da unidade *BranchControl* (*PCLoadEnable* e *PCLoadValue),* todos os sinais que atravessam vários estágios de pipeline, têm de passar pelos respectivos registos entre estágios. Por exemplo, o sinal MD que sai do descodificador no estágio de ID (identificado por *ID_MD* no ficheiro FiveSagePipeline.vhd) passa necessáriamento pelos registos ID/EX, EX/MEM e MEM/WB, de forma a formar os sinais EX_MD (estágio de EX), MEM_MD (estágio de MEM) e WB_MD (estágio de MD).

PC	Palavra de Instrução	Instrução em Assembly	R1	R2	R3	R4	R5	
0 ↓ **1**	04800001h	ADDI R2,R0,#1	0	0 ↓ **1**	0	0	0	← Início da execução da instrução
1 ↓ **2**	04C00003h	ADDI R3,R0,#3	0	1	0 ↓ **3**	0	0	← Valores actualizados pela instrução após o flanco de relógio
2 ↓ **3**	0108C000h	ADD R4,R2,R3	0	1	3	0 ↓ **4**	0	
2 ↓ **9**	5D500007h	B.GE R4,R7,#7	0	1	3	4	0	
...	...	...	...	...	...	...	...	

Figura 3 – Tabela com a execução esperada de um troço de código exemplo.

Sugestão: Para a realização dos pontos 1 e 2, sugere-se que comece por construir uma tabela (p. ex. em papel, ou no MS Excel) com a execução esperada do seu troço de código (ver exemplo da figura 3). De seguida, verifique em cada ciclo de relógio se os registos são corretamente atualizados. Sempre que encontrar um valor que é incorretamente atualizado, verifique no estágio de ID se os operandos lidos do *Register File* estão de acordo com o esperado. Se não estiverem, verifique em que estágio se encontra a instrução que atualiza o valor esperado (lembre-se que numa

arquitetura com N estágios de *pipeline*, uma instrução só atualiza o valor dos registos após a execução dos N estágios, i.e., após a execução do estágio de WB – *write back*).

De forma a facilitar o processo, todos os sinais do componente principal do processador (ficheiro *FiveStagePipeline.vhd*) são precedidos da identificação do estágio a que o sinal pertence. Por exemplo, o sinal `IF_PC` corresponde ao valor do PC correspondente à instrução atualmente no estágio de *Instruction Fetch*; o sinal `WB_PC` corresponde ao valor do PC correspondente à instrução atualmente no estágio de *write-back*.

Para resolver os problemas relacionados com a execução do código, sugere-se que reordene as instruções, ou que insira instruções de NOP sempre que achar conveniente (resolução de conflitos por *software*). Sugere-se ainda que tente resolver cada um dos conflitos encontrados individualmente e sequencialmente. Adicionalmente, chama-se à atenção para a execução das instruções de salto condicional (`B.cond`) que podem gerar dois tipos de conflitos: i) conflitos de dados (os operandos lidos pela instrução no estágio de ID estão incorretos); e ii) conflitos de controlo (o estágio de IF lê uma ou mais instruções erradas da memória), já que a execução da instrução é realizada no estágio de EX, mas o seu resultado é necessário logo no estágio de IF. Devido a estes dois conflitos, a execução da instrução de salto condicional pode estar errada, mas a operação sobre o registo de PC estar correta (apenas por acaso).

Observação: Nos processadores de uso geral típicos (i.e., excluindo micro-controladores e alguns processadores desenhados para ter um consumo de potência excecionalmente baixo) são usadas diversas técnicas mais avançadas para explorar melhor o paralelismo ao nível das instruções, o qual é exposto com a introdução de pipelines mais longos (mais estágios). Entre estas, encontram-se técnicas para execução simultânea de múltiplas instruções; para resolver de forma eficiente e automática os conflitos de dados (introduzindo NOPs, ou bolhas, sempre que necessário, ou alterando a ordem de execução das instruções de forma automática e transparente para o programador); e para predição do resultado dos conflitos de controlo (execução especulativa). Estas técnicas são elaboradas com detalhe no âmbito de unidades curriculares mais avançadas.

6 RELATÓRIO

No fim da segunda semana de laboratório deverá submeter o relatório, o qual deverá ser sucinto (dispensam-se introduções teóricas), mas que terá obrigatoriamente de conter:

- Objetivos
- Descrição breve do funcionamento do processador e indicação da função de cada um dos sinais de controlo à saída da memória de descodificação.
- Descrição das alterações à unidade de descodificação e à unidade de controlo de salto (*BranchControl*).
- Metodologia usada para verificação dos conflitos encontrados no código, quando executado no processador *pipelined* e proposta justificada de alterações.
- Conclusões e comentários aos resultados obtidos.

O relatório completo não deverá exceder as **10 páginas.** Na submissão do relatório (em formato PDF) deverá ainda anexar os ficheiros de projeto com a sua solução final. Para tal, no Vivado, deverá ir a Ficheiro→Archive Project. Não selecione as opções "Include configuration settings" e "include run results".

7 AVALIAÇÃO

A avaliação do trabalho será realizada, ao longo das duas aulas de laboratório ponderando a resposta às questões indicadas no enunciado e realizadas pelo docente na aula de laboratório (peso de 85%), e pela qualidade do relatório (peso de 15%). Valoriza-se, em particular: (1) a participação e empenho dos alunos nas aulas; (2) a originalidade da solução e dos testes realizados; (3) o grau de detalhe da solução; (4) a descrição do processo de síntese e justificação das diferentes opções de projeto; (5) a boa estruturação, a escrita sucinta e objetiva e, ainda, a boa apresentação do relatório.

8 BIBLIOGRAFIA

[1] N. Horta, "Arquitecturas de Computadores – Unidade de Controlo", Aulas Teóricas, 2017.
[2] M. Morris Mano, Charles R. Kime, "Logic and Computer Design Fundamentals", 4thEdition Updated, Prentice-Hall International, 2014.
[4] G. Arroz, J. Monteiro, A. Oliveira, "Arquitectura de Computadores: dos Sistemas Digitais aos Microprocessadores", IST Press, 2007.