TRABALHO DE LABORATÓRIO IV

CIRCUITO DE PROCESSAMENTO DE DADOS

VERSÃO 2.2

1. INTRODUÇÃO

Pretende-se com este trabalho que os alunos analisem e projetem um circuito de processamento de dados constituído por elementos básicos de memória (registos) e módulos combinatórios.

Este trabalho é considerado para **avaliação de conhecimentos**. **Na aula**, cada grupo deverá **impreterivelmente** mostrar ao docente a resposta a todas as questões referidas na folha de respostas. Recomenda-se que **realize todo o trabalho em casa, usando a aula de laboratório apenas para testar o circuito na placa de prototipagem**. A folha de respostas (em formato PDF) e o projeto arquivado[1], deverão ser colocados num *único ficheiro* (em formato **ZIP**). Este ficheiro deverá ser entregue (no Fénix) **até às 23h59m de sexta-feira, dia 2 de Dezembro.**

A preparação prévia do trabalho de laboratório está dependente da instalação (bem-sucedida) da ferramenta de Vivado WebPack. A instalação deste software pode ser realizada com a ajuda do Guia de Instalação do Vivado Design Suite WebPack, disponível na página da cadeira. Durante a realização deste trabalho, poderá ser útil consultar os seguintes slides das aulas teóricas:

1. Aula 12: Linguagens de Descrição e Simulação de Circuitos Digitais
2. Aula 14: Circuitos Sequenciais Básicos: Simbologia e Descrição em VHDL
3. Aula 16: Registos
4. Aula 17: Contadores

Para conseguir realizar a prototipagem dos circuitos implementados no laboratório, é fundamental consultar o **Guia de Implementação de Circuitos na Placa de Desenvolvimento** (*Digilent Basys 3*), disponível na página da cadeira. Recomenda-se vivamente que **tenha estes documentos consigo durante a aula**.

2. REPRESENTAÇÃO DE NÚMEROS EM VÍRGULA FLUTUANTE

As representações em vírgula flutuante (Floating-Point) permitem representar simultaneamente números muito grandes e números muito pequenos, com um número reduzido de bits. De uma forma muito simplificada, uma representação em vírgula flutuante, adequada para sistemas digitais, consiste em:

$$A = m_A * 2^{e_A}$$

onde m_A é a mantissa de A, e e_A o expoente de A. Assim, com um número reduzido de bits podem representar-se números muito maiores ou menores do que o que seria possível com a representação binária natural. Para além disso, o mesmo número pode ser representado por mais que um código em vírgula flutuante. Por exemplo, $8 * 2^0 = 4 * 2^1 = 1 * 2^3$. Na tabela seguinte mostram-se os alcances de algumas representações com 8 bits.

[1] Para arquivar o projeto no Vivado, clique *File→Archive Project*. Será aberta uma janela, onde deve indicar o nome ("*Archive name*") e localização em disco do projeto arquivado ("*Archive location*") - ex: no ambiente do trabalho. Clique *OK* para guardar o projeto em formato *zip*.

Tabela 1 – Propriedades de algumas representações com 5 bits de mantissa e 3 bits de expoente.

Mantissa	Expoente	Min.	1º Valor depois do Min.	Mais Pequeno[2]	1º Valor antes do Max.	Max
5-bit sem sinal	3-bit sem sinal	0	1	1	3840	3968
	3-bit com sinal	0	0.0625	0.0625	240	248
5-bit com sinal	**3-bit sem sinal**	**-2048**	**-1920**	**1**	**1792**	**1920**
	3-bit com sinal	-128	-120	0.0625	112	120

Neste trabalho de laboratório vamos estudar a operação de adição de números na representação de 8 bits assinalada a negrito na tabela 1, em que a mantissa tem uma representação em complemento para 2 com 5 bit, e o expoente é representado por 3 bit sem sinal.

Sendo A, um número de vírgula flutuante, com a representação "11011010", A(7:3) com o valor de "11011" (-5 em decimal) é a respetiva mantissa (mA), enquanto que A (2:0), com o valor de "010" (2 em decimal) representa o expoente (eA). Esta representação permite descrever números inteiros entre -2048 a 1920. No entanto continuam a existir apenas 256 representações diferentes, por isso o número de números diferentes representáveis é na realidade menor que na representação binária natural. O que existe é um aumento gradual, com o aumento do expoente, da diferença entre dois números representáveis consecutivos. Por exemplo, tendo o número 32 ($8*2^2$) como referência, os números representáveis mais próximos são 30 ($15*2^1$) e 36 ($9*2^2$).

A soma de dois números em vírgula flutuante é realizada através da soma das suas mantissas, depois de serem ambos os números transformados para o mesmo expoente. Para minimizar a perda de precisão, os números devem estar normalizados (como é explicado no parágrafo seguinte), e converte-se o número com menor expoente para uma representação intermédia utilizando o maior dos expoentes (dos dois operandos).

No contexto deste trabalho, um número em vírgula flutuante está normalizado quando não é possível reduzir o expoente sem que ocorra perda de informação. Por exemplo 16 pode ser representado por $8*2^1$, $4 * 2^2$, …, $1 * 2^4$, onde, $8*2^1$ é a representação normalizada.

De seguida, ilustram-se 3 alternativas para fazer a adição dos operandos 2 e 16, descrevendo-se os passos intermédios e a importância de ter as entradas normalizadas e ajustar ao maior expoente. O ajuste do expoente faz-se através da aplicação de operações de deslocamento para a direita (esquerda) à mantissa e incrementando (decrementando) o expoente.

Alternativa 1: Operandos normalizados e ajuste ao maior expoente:

$2*2^0 + 8*2^1 = 1*2^1$(SRA) + $8*2^1 = 9*2^1 =$ **18**

Alternativa 2: Operandos não normalizados e ajuste ao maior expoente:

$2*2^0 + 2*2^3 = 0*2^3$(3 SRA) + $2*2^3 = 2*2^3 =$ **16 ← Resultado Incorreto!**

Alternativa 3: Operandos normalizados mas ajuste para o menor expoente:

$2*2^0 + 8*2^1 = 2*2^0 + -16*2^0$(SLA) $= -14*2^0 =$ **-14 ← Resultado Incorreto!**

(SRA – Deslocamento para a direita aritmético da mantissa. SLA – Deslocamento para a esquerda aritmético da mantissa.)

2.1 – Estrutura geral do Circuito de Processamento de dados

O circuito de computação (ou processamento) estudado neste laboratório, que implementa a soma de dois números representados em vírgula flutuante assumindo a representação com 5 bit de mantissa e 3 bit de expoente descrita acima, é um circuito sequencial que lê e guarda (em registos) os dois operandos, e aplica uma sequência de operações sobre esses números de forma a obter o resultado final. Este tipo de estrutura confere alguma flexibilidade na escolha do algoritmo a usar, pois o projetista tem a possibilidade de especificar o tipo e sequência de operações que deverão ser realizadas, com vista a executar os vários passos intermédios do algoritmo a implementar.

[2] Menor valor em modulo, excluindo o "0"

O diagrama simplificado (vista de componente) do circuito de computação (float_add) considerado é apresentado na Figura 1, e mostra as suas entradas e saídas. Este circuito tem 5 entradas (A, B, START, CLK e RESET) e 3 saídas (OutA, OutB, I) com as seguintes funções:

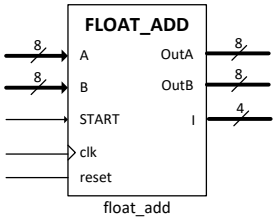

Fig.1. Circuito sequencial L4 (componente)

- **CLK** (1 bit): Sinal de relógio, síncrono com o flanco ascendente.
- **RESET** (1 bit): Reset assíncrono dos componentes de memória (ICTR, RA, RB e RZ);
- **START** (1 bit): inicia o cálculo e carrega os valores dos operandos A e B no sistema. Este sinal apenas é considerado se o sistema não estiver a efetuar cálculos intermédios do algoritmo. Até à ativação do sinal START os valores guardados na última computação em RA e RB mantêm-se fixos;
- **A** (8 bits em vírgula flutuante): valor do operando A que será guardado no início do processamento; A(7:3) é a mantissa de A (mA), enquanto que A(2:0) representa o expoente de A (eA).
- **B** (8 bits em vírgula flutuante): valor do operando B que será guardado no início do processamento; B(7:3) é a mantissa de B (mB), enquanto que B(2:0) representa o expoente de B (eB).
- **OutA** (8 bits em vírgula flutuante): Valores intermédios e resultado final da operação quando o cálculo termina;
- **OutB** (8 bits em vírgula flutuante): Valores intermédios;
- **I** (4 bits): Número da instrução a ser executada.

Na figura 2 é apresentado o sistema em detalhe, onde se vêem os componentes que compõem o circuito e alguma lógica adicional.

Os componentes que constituem o sistema são:

- **ALU1**: Unidade aritmética que efetua operações aritméticas simples sobre mantissas e expoentes, explicada em detalhe na secção 2.2.
- **RA** e **RB**: Registos com 8 bits onde são guardados números de vírgula flutuante, explicados em detalhe na secção 2.3.
- **RZ**: Registo de 1 bit (Flip Flop do tipo D com entrada de enable do relógio), explicado em detalhe no final da secção 2.3.
- **ICTR**: contador que gera uma sequência de números (4-bit) para identificar os diferentes passos do algoritmo, explicado em detalhe na secção 2.4.
- **CTRL**: circuito combinatório que define, para cada passo do algoritmo, qual a operação efetuada na ALU1 e em RA, RB e RZ, explicado em detalhe na secção 2.5.

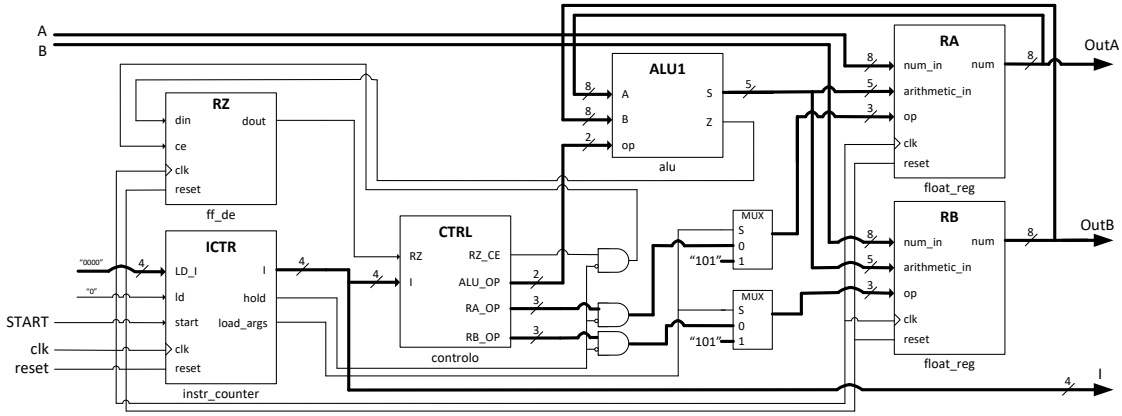

Figura 2 – Circuito de computação.

Como se pode ver na Figra2 as saídas OutA, OutB, e I, estão ligadas, respetivamente, às saídas dos componentes RA, RB e ICTR.

Como elementos de memória temos 2 registo de 8 bits desenhados especificamente para guardar números na representação de vírgula flutuante escolhida (RA e RB), e um FlipFlop do tipo D com enable (RZ). Em RA e RB são guardadas os valores dos operandos (no inicio do processamento) e valores intermédios dos cálculos (durante o processamento). O processamento dos dados é feito na ALU1 e nos registos RA e RB, que, para além de guardarem os números em vírgula flutuante (mantissa e expoente), implementam ainda algumas operações lógicas e de deslocamento.

Em cada passo do algoritmo, que corresponde a cada ciclo do sinal de relógio, os vários sinais que definem qual a operação realizada em cada um dos componentes do sistema (RA, RB, RZ e ALU1), definem uma (macro) operação ao nível do sistema. O conjunto das operações realizadas pelos vários componentes do sistema para realizar uma operação ao nível de sistema chama-se comando ou instrução. Assim, o circuito estudado implementa a soma de números em vírgula flutuante utilizando uma sequência de instruções de forma a obter o resultado esperado.

O controlo do sistema utiliza um contador de instruções (ICTR), que gera uma sequência de números que identificam os diferentes passos do algoritmo (instruções), e um controlador (ou descodificador de instruções) (CTRL) que para cada um dos passos do algoritmo calcula os valores dos sinais que definem as operações em cada um dos componentes (RA, RB, RZ e ALU1) de forma a implementar a instrução pretendida nesse passo do algoritmo.

O circuito de computação fornecido já implementa a soma de números em vírgula flutuante, assumindo que as entradas A e B estão normalizadas e que em A está o número com maior expoente, e a soma é feita com o seguinte algoritmo:

- Ler o valor nas entradas A, B.
- Calcular a diferença entre os expoentes para saber quantos deslocamentos são necessários efetuar na mantissa de B.
- Ajustar a mantissa de B, deslocando para a direita o número de vezes correspondente à diferença entre expoentes eA e eB.
- Somar as mantissas mA + mB, colocando o resultado em mA.

O código fonte VHDL deste circuito está disponível na página da disciplina (lab4.zip). Comece por criar um novo projeto no VIVADO (*Ver instruções no enunciado do L3 e no Guia da Basys3*), e adicione as implementações e testbenches. Os ficheiros da placa devem ser adicionados apenas no ponto 2.7. (*Ver instruções no enunciado do L3*). Uma descrição sumária dos ficheiros do projeto está específicada na tabela seguinte.

Tabela 2 – Sumário dos ficheiros fornecidos.

Componente	Implementação	Test Bench
Circuito Completo	**float_add.vhd**	**tb_float_add.vhd**
↳ **ALU**	alu.vhd	tb_alu.vhd
↳ Somador5	somador5.vhd	tb_somador5.vhd
↳ Full Adder	full_adder.vhd	tb_full_adder.vhd
↳ **Registos (RA/RB)**	float_reg.vhd	tb_float_reg.vhd
↳ **Registo (RZ)**	ff_de.vhd	
↳ **Contador de Instruções (ICNTR)**	instr_counter.vhd	tb_instr_counter.vhd
↳ Contador (CTR)	ctr_16.vhd	tb_ctr_16.vhd
↳ Counter Control CTR_CONTROL	ctr_control.vhd	tb_ctr_control.vhd
↳ **Controlador (CDOR)**	controlador.vhd	tb_controlador.vhd

2.2 – ALU (Unidade Aritmética e Lógica)

A Unidade Aritmética e Lógica (ALU) que foi implementada para este circuito utiliza um somador de 5 bits, e permite operações básicas sobre as mantissas ou expoentes dos operandos presentes nas suas entradas A e B (ambas de 8 bits). Recorde que A(7:3) e B(7:3) são as mantissas de A e B respetivamente (mA e mB), enquanto que A (2:0) e B(2:0) representam os expoentes (eA e eB).

A operação da ALU é controlada com um sinal de entrada da ALU de 2 bits, o sinal op(1:0). Tal como indicado na Tabela 3, cada combinação deste sinal permite realizar uma operação diferente. A ALU apresenta duas saídas: S(4:0), de 5 bits, com o resultado da operação e Z, um bit, que sinaliza com o valor '1' que o resultado da operação, S(4:0), é igual a zero ("00000"), e com o valor '0' que S(4:0) é diferente de "00000".

Tabela 3 – Operações implementadas pela ALU fornecida.

OP(1:0)	ALU Operação	Descrição
00	Sa = mA + mB	Soma Mantissas (ADDm)
01	Sa = eA - eB	Subtrai Expoentes (SUBe)
10	Sa = eB – 1	Decrementa o expoente de B (DECe)
11	Sa = eA + 1	Incrementa o expoente de A (INCe)

A sua estrutura está representada na Figura 3 (à esquerda tem-se o esquema do componente, com as respetivas entradas e saídas; à direita apresenta-se o esquema interno).

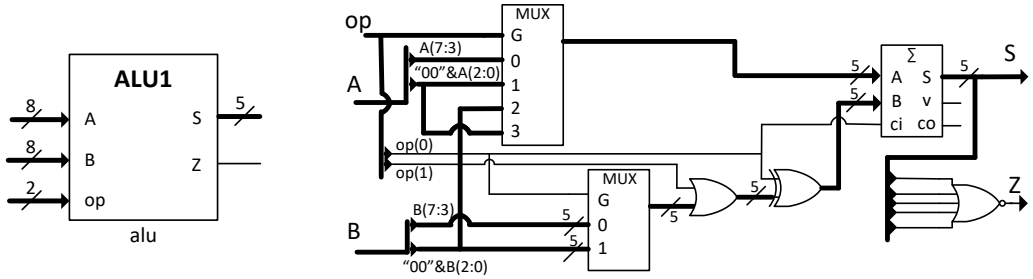

Figura 3 – Unidade aritmética e lógica (ALU).

2.3 – Registos

Tal como indicado na Figura 2, existem dois registos para alojar os operandos com representação em vírgula flutuante, RA e RB, e um Flip Flop do tipo D com enable (RZ) que guarda a saída Z da ALU.

Os registos RA e RB (float_reg.vhd) de 8 bit, ilustrados na Figura 4 ao nível de componente, e cujo diagrama detalhado é apresentado na Figura 5, permitem controlar separadamente as operações de escrita sobre o expoente e/ou mantissa, e implementam ainda operações de deslocamento sobre a mantissa.

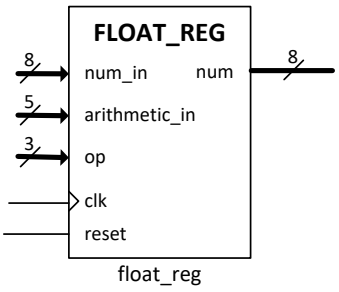

Figura 4 –Registo para números em vírgula flutuante (float_reg)

As suas entradas e saídas, além do CLK e RESET, são:

- **num_in (7:0)**: uma entrada de 8 bits ligada diretamente à entrada externa A (ou B) no circuito de topo;
- **arithmetic_in (4:0)**: uma entrada de 5 bits ligada à saída S da ALU1;
- **op(2:0)**: uma entrada de controlo de 3 bits ligada ao CTRL;
- **num(7:0)**: uma saída de 8 bits com o número guardado ligado a entrada A (ou B) da ALU1;

Recorde que num(7:3) e num_in(7:3) são mantissas, enquanto que num(2:0) e num_in(2:0) são expoentes. O significado da entrada aritmetic_in, varia com a operação que está a ser realizada no registo.

A operação a realizar pelo registo é controlada pelo sinal de entrada op(2:0). As operações realizadas no registo estão descritas na tabela seguinte.

Tabela 4 – Descrição das operações implementadas pelo registo para números em vírgula flutuante.

OP	Descrição da Operação
000	**Hold –** mantém o valor anterior.
001	**Load M –** carrega o valor da entrada arithmetic_in na mantissa e mantém o valor anterior do expoente
100	**Load E –** carrega o valor da entrada arithmetic_in no expoente (3 bits menos sinificativos) e mantém o valor anterior da mantissa.
101	**Load –** carrega o valor da entrada num_in.
111	**SRA M; Ld E–** carrega o valor da entrada arithmetic_in no expoente e faz o deslocamento para a direita aritmético da mantissa.

Tabela 5 –Valores nas operações implementadas pelo registo para números em vírgula flutuante.

OP	Operação	Valor antes do flanco			Valor após o flanco
		arithmetic_in(4:0)	num_in(7:0)	num(7:0)	num(7:0)
000	**Hold**	**A(4:0)**	**B(7:0)**	**N(7:0)**	**N(7:0)**
001	**Load M**	**A(4:0)**	**B(7:0)**	**N(7:0)**	**A(4:0) & N(2:0)**
100	**Load E**	**A(4:0)**	**B(7:0)**	**N(7:0)**	**N(7:3) & A(2:0)**
101	**Load**	**A(4:0)**	**B(7:0)**	**N(7:0)**	**B(7:0)**
111	**SRA M; Ld E**	**A(4:0)**	**B(7:0)**	**N(7:0)**	**N(7) & N(7:4) & A(2:0)**

& é o operador de concatenação

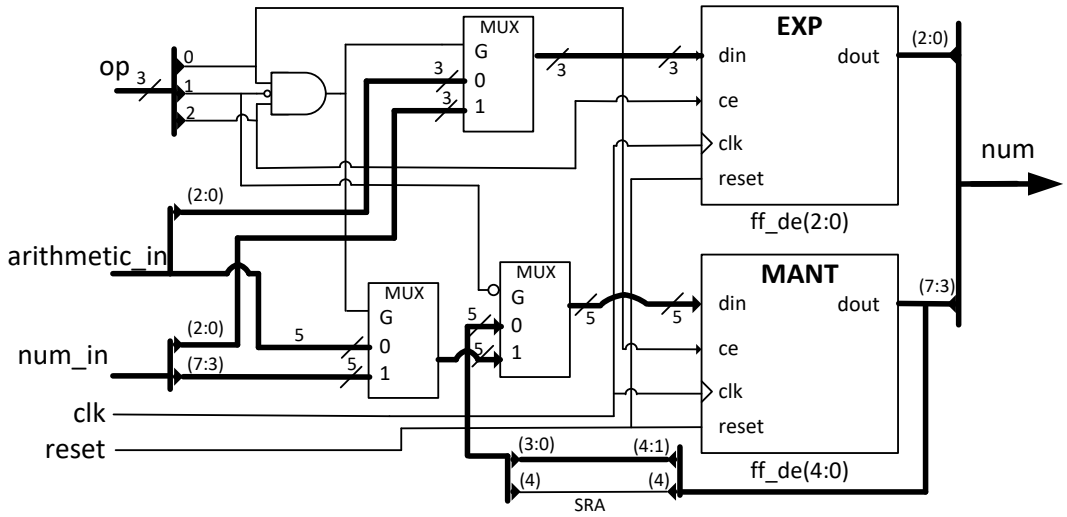

Figura 5 –Registo para números em vírgula flutuante (float_reg)

Responda à Pergunta 1 na folha de respostas.

Existe ainda o flip-flop do tipo D (ff_de.vdh) com entrada de enable onde é guardada a saída Z proveniente da ALU, que é usado como entrada no controlador para definir a execução (ou não) de determinadas operações. Tal como indicado na Figura 6, o registo é caracterizado por uma entrada din e por uma saída dout, e uma entrada de enable CE, além das entradas CLK e RESET. A escrita é

controlada pela entrada CE (1bit): CE=1 permite a escrita (e leitura) no registo aquando o flanco do relógio, enquanto CE=0 permite apenas a leitura do registo.

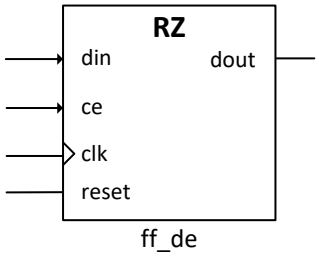

Figura 6– Registo de 3bits.

Tabela 6 – Sinal do controlo CE

CE	Operação	Antes do flanco		Após o flanco
		din	dout	dout
0	**Leitura**	A	Q	**Q**
1	**Escrita**	A	Q	**A**

2.4 – Contador de Instruções

O contador de instruções ICTR (instr_counter.vhd) é o bloco responsável pela geração da sequência de números que depois de descodificados (no controlador) faz avançar a execução dos passos do algoritmo. Este componente também implementa o controlo relativo à entrada externa START, iniciando a sequência de cálculo quando o START é pressionado, desde que não esteja ainda a processar um pedido anterior. O circuito interno do contador de instruções ICTR está representado na Figura 7.

As suas entradas e saídas, além do CLK e RESET, são:

- **LD_I(3:0)**: entrada de 4 bits de dados para carregamento paralelo, ligada inicialmente a "0000" no circuito de topo;
- **ld**: entrada de 1 bit para sinalizar a ordem de carregamento paralelo, ligada inicialmente a '0' no circuito de topo;
- **start**: entrada de 1 bit que inicia o cálculo e gera a ordem (load_args='1') para carregar os valores dos operandos A e B no sistema.
- **I**: saída de 4 bits com valor da contagem, que identifica o passo do algoritmo a executar.
- **hold**: saída de 1 bit que sinaliza que o sistema está à espera de um novo pedido, este sinal é utilizado no circuito de topo para desabilitar a escrita nos registos RA, RB e RZ.
- **load_args**: saída de 1 bit que sinaliza que um novo pedido foi detetado e é usado no sistema de topo para carregar os operandos nos registos RA e RB;

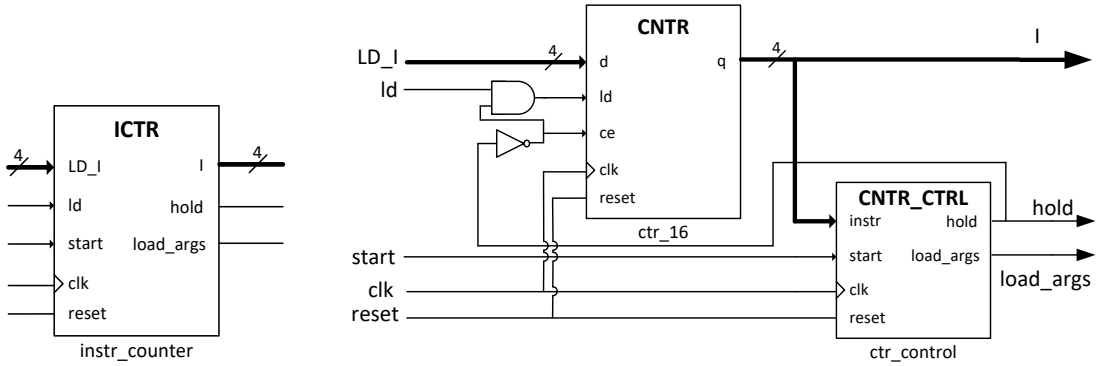

Figura 7 – Estrutura interna do contador de instruções.

O contador de instruções é composto por 2 componentes: um contador de 4 bits (CNTR) com carregamento paralelo e um bloco de controlo (CNTR_CTRL). O bloco de controlo mantem o contador (CNTR) parado em "0000" até ser detetado o sinal start = '1', desbloqueando a contagem do contador que continua a partir dai até voltar ao estado zero. Quando a contagem é desbloqueada, os valores dos registos RA e RB são também atualizados com o valor dos operandos

vindos das entradas externas A e B (este comportamento é conseguido com as saídas hold e load_args que controlam a escrita nos registos RA, RB e RZ através da lógica adicional mostrada na Figura 2). Note-se que sendo o contador de 4 bits, e estando a instrução "0000" reservada para manter o contador à espera do sinal START e carregar os valores das entradas A e B, o circuito de computação executa, no máximo, 15 instruções.

Responda à Pergunta 2 na folha de respostas.

2.5 – Controlador

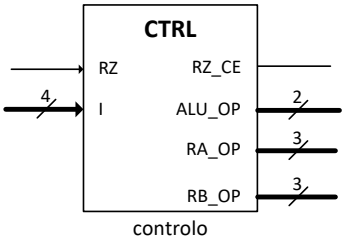

Figura 8 –Controlo (componente)

O controlador (CTRL) é um bloco combinatório que recebe como entradas o número correspondente à instrução a executar, proveniente do contador de instruções (I(3:0)), e a Flag Z da ALU registada (RZ), e devolve nas saídas os sinais de controlo correspondentes aos vários componentes do circuito de computação, nomeadamente a saída RZ_CE que está ligada à entrada CE do registo da RZ, a operação dos registos RA e RB (saídas RA_OP(2:0), RB_OP(2:0)) e a operação a executar pela ALU através da a saída ALU_OP(1:0).

A tabela 7 mostra duas formas de apresentar os valores pretendidos nas saídas (linhas da tabela de verdade do controlador) correspondentes à descodificação de uma instrução. Como a instrução não depende da entrada RZ, apenas do código de instrução I, podemos apresentar alternativamente a entrada RZ com X e reduzir o número de linhas necessárias para descrever as funções binárias de 5 entradas a implementar pelo controlador. A instrução deste exemplo verifica se mA = -mB, i.e. se o resultado da soma das duas mantissas é zero, deve configurar-se a ALU para calcular a soma das mantissas (operação "00" (ADDm) descrita em 2.2), manter o valor dos registos RA e RB (operação "000" (HOLD) descrita em 2.3), e guardar o valor da flag Z em RZ (operação '1' (escrita no registo) descrita em 2.3).

Tabela 7 – Tabela do controlo de uma instrução.

I(3:0)	RZ	RZ_CE	RA_OP(2:0)	RB_OP(2:0)	ALU_OP(1:0)	Instrução	Resultado Parcial
I0	0	1	000	000	00	TEST A=-B	RZ<='1' when (mA + mB) = '0' else '0'
	1	1	000	000	00		RZ<='1' when (mA + mB) = '0' else '0'
I0	X	1	000	000	00	TEST A=-B	RZ<='1' when (mA + mB) = '0' else '0'

A estrutura do controlador vai depender da sequência de instruções pretendida. No caso do algoritmo da soma de números de vírgula flutuante, assume-se que em RA está o operando que tem o maior expoente, e ajusta-se a mantissa do operando com menor expoente, RB. Após ser feito o ajuste da mantissa de RB, as duas mantissas são somadas e o resultado colocado no registo RA. A operação descrita pode ser implementada usado as instruções listadas na tabela 8.

Tabela 8 – Instruções requeridas para implementar a soma de dois números em vírgula flutuante.

Instrução	Descrição
NOP	NOP ou "No Operation" é uma instrução que não tem impacto no resultado. Neste caso, pode ser executado qualquer instrução na ALU, desde que o resultado não seja aguardado nos registos.
SUBe	Subtrai os expoentes e guarda o resultado em eB e a flag Z em RZ. Permitindo guardar em eB o número de deslocamentos a necessários. `{eB <= (eA-eB); RZ <= Z(eA-eB)}`
nZ_SRAmDECe	Se o valor da RZ for 0, executa duas operações simultaneamente: faz um deslocamento aritmético para a direita da mantissa de RB, e decrementa o seu expoente e atualiza RZ. Se o valor de RZ for 1 não tem impacto no resultado. Esta instrução é repetida para deslocar o número necessário de vezes a mantissa mB. `{mB <= (SRA mB); eB <= (eB-1); RZ <= Z(eA-eB)} when Z='1'else NOP`
ADDm	Soma as mantissas e guarda o resultado em RA e a flag Z em RZ. `{mA <= (mA + mB); RZ <= Z(eA-eB)}`

Onde Z(X) = '1' se X for zero ou '0' se X for diferente de zero.

Usando as instruções definidas acima, a tabela 9 descreve a sequência de controlo do circuito e os valores esperados nas saídas do controlador. (X indica "don't care"). Na coluna da direita mostram-se os resultados parciais da execução do algoritmo para o cálculo da soma de $8*2^3 + 8*2^1$.

Tabela 9 – Tabela do controlo.

I(3:0)	RZ	RZ_CE	RA_OP	RB_OP	ALU_OP	Instrução	Resultado Parcial (após o flanco)
0000	X	X	X	X	X	Not Used	RA=01000_011; RB = 01000_001;RZ=X
0001	X	1	000	100	01	SUBe	RA=01000_011; RB = 01000_010;RZ=0
0010	0	1	000	111	10	nZ_SRAmDECe	RA=01000_011; RB = 00100_001;RZ=0
	1	0	000	000	X		
0011	0	1	000	111	10	nZ_SRAmDECe	RA=01000_011; RB = 00010_000;RZ=1
	1	0	000	000	X		
0100	0	1	000	111	10	nZ_SRAmDECe	RA=01000_011; RB = 00010_000;RZ=1
	1	0	000	000	X		
0101	0	1	000	111	10	nZ_SRAmDECe	RA=01000_011; RB = 00010_000;RZ=1
	1	0	000	000	X		
0110	0	1	000	111	10	nZ_SRAmDECe	RA=01000_011; RB = 00010_000;RZ=1
	1	0	000	000	X		
0111	X	1	001	000	000	ADDm	RA=01010_011; RB = 00010_000;RZ=0
1000	X	0	000	000	X	NOP	RA=01010_011; RB = 00010_000;RZ=0
1001	X	0	000	000	X	NOP	RA=01010_011; RB = 00010_000;RZ=0
1010	X	0	000	000	X	NOP	RA=01010_011; RB = 00010_000;RZ=0
1011	X	0	000	000	X	NOP	RA=01010_011; RB = 00010_000;RZ=0
1100	X	0	000	000	X	NOP	RA=01010_011; RB = 00010_000;RZ=0
1101	X	0	000	000	X	NOP	RA=01010_011; RB = 00010_000;RZ=0
1110	X	0	000	000	X	NOP	RA=01010_011; RB = 00010_000;RZ=0
1111	X	0	000	000	X	NOP	RA=01010_011; RB = 00010_000;RZ=0

Ou seja, os diferentes bits das saídas do controlador RZ_CE, RA_OP, RB_OP, ALU_OP são funções binárias de 5 entradas (os quatro bits de I e RZ). Tendo em conta a tabela 9, uma implementação não otimizada correspondente ao controlador é apresentada na figura 9:

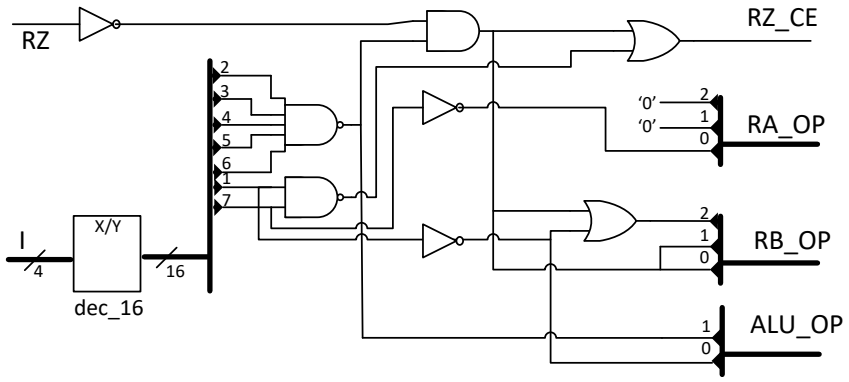

Figura 9 –Controlador (CTRL).

2.6 – Análise do circuito completo: Float Add

Concatene os 3 dígitos de menor peso do menor dos números dos alunos do grupo, com os dois dígitos de maior peso do maior número de aluno. Converta para binário o resultado da concatenação (considere apenas os 16 bits de menor peso) e use os 8 bits de menor peso como X, e os bits (15:11) como mantissa de Y. O expoente de Y é igual ao expoente de X, exceto no segundo bit, que é negado, normalize. Atribua à entrada A o número com o maior expoente e a B o número com o menor expoente.

Exemplo para os números 74640 e 75689

75 & 640 = 0010 0111 0111 1000 > X = 0111 1000, Y = 0010 0010,

Normalizando: X = 0111 1000, e Y = 0100 0001 <> A = 0100 0001, B = 0111 1000

Calcule o resultado esperado da soma de A + B, aplicando manualmente o algoritmo implementado pelo controlador. Confirme esse valor em simulação. Para isso altere o testbench do circuito global (tb_float_add.vhd) para que as entradas correspondam ao pretendido e execute a simulação.

Responda às Perguntas 3 e 4 na folha de respostas.

2.7 – IMPLEMENTAÇÃO NA PLACA DE DESENVOLVIMENTO

Nota importante: Antes de iniciar o teste do circuito é **fundamental consultar (em casa)** o **Guia de Implementação de Circuitos na Placa de Desenvolvimento** (Digilent Basys 3), disponível na página da cadeira.

Figura 10. Placa de prototipagem Basys 3.

Para realizar o teste do circuito projetado utilizando a placa de prototipagem (*Digilent Basys 3*, equipada com a FPGA *Artix-7* com referência XC7A35T-CPG236, da Xilinx – ver Figura 14), foi disponibilizado um conjunto de ficheiros na pasta ***placa*** (veja no guia da placa a descrição dos componentes), que deverá utilizar nesta parte do trabalho:

- **sd.vhd** – descrição do circuito principal (da placa)
- **Basys3_Master.xdc** – configuração dos portos (da placa)
- **clkdiv.vhd** – divisor de frequência (especificação)
- **disp7.vhd** – bloco do controlo do display de 7 segmentos (especificação).

Não modifique os nomes destes ficheiros!

1) Adicione os ficheiros `sd.vhd`, `clkdiv.vhd` e `disp7.vhd` ao projeto, fazendo *Add Sources→Add or create **design** sources*.
2) Adicione o ficheiro `Basys3_Master.xdc` ao projeto, fazendo *Add Sources→Add or create **constraints***.
3) Verifique se o ficheiro **sd.vhd** está definido como módulo de topo (faça clique direito no ficheiro e selecione a opção "*Set as Top*"). Verifique também se a hierarquia do projeto inclui os componentes `clkdiv`, `disp7` e `Basys3_Master.xdc`, conforme indicado na Figura 15. A inclusão destes componentes é obrigatória e deve ser sempre verificada (**a não inclusão do ficheiro Basys3_Master.xdc, pode DESTRUIR o dispositivo, e caso isso aconteça ser-lhe-ão pedidas responsabilidades**).

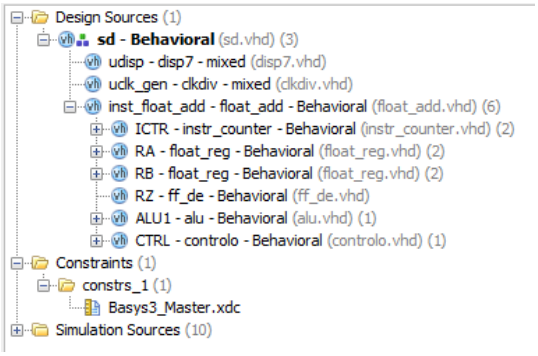

Figura 11. Hierarquia do projeto incluindo `sd`, `clkdiv`, `disp7` e `Basys3_Master.xdc`

4) Abra o módulo `sd`, clicando duas vezes em cima do ficheiro `sd.vhd`. Este projeto não é mais do que uma interface da placa que é disponibilizada ao aluno: as entradas e saídas *já estão TODAS configuradas* de acordo com o modelo do dispositivo utilizado na placa de desenvolvimento. Por conseguinte, este módulo funciona como uma placa de prototipagem virtual. **Nota**: *Não altere o conteúdo do código neste ficheiro a não ser que tal lhe seja pedido pelo docente*
5) As seguintes ligações foram estabelecidas de forma a possibilitar a correta interação do utilizador com o circuito:
    a. O sinal de relógio ***clk*** está ligado ao sinal **`clk_slow`** (este sinal tem uma frequência fixa de 1,5 Hz);
    b. A entrada ***reset*** está ligada ao buffer do botão de pressão **`BTN(0)`**, i.e., o botão superior;
    c. O sinal de entrada ***start*** está ligado ao buffer do botão de pressão **`BTN(3)`**, i.e., o botão inferior;
    d. O operando ***A*** está ligado aos interruptores **`SW(15:8)`**, i.e., os 8 interruptores (*switches*) do lado esquerdo da placa;
    e. O operando ***B*** está ligado aos interruptores **`SW(7:0)`**, i.e., os 8 interruptores (*switches*) do lado direito da placa;
    f. A instrução a ser executada, ***INST***, é apresentada no dígito 3 do display de 7 segmentos em formato hexadecimal, i.e., **`disp3`**;
    g. O valor de RA, ***OutA***, é apresentado nos dígitos 2, 1 e 0 do display de 7 segmentos em formato hexadecimal, i.e., **`disp2`** e **`disp1`** mostram a mantissa; **`disp0`** mostra o expoente.
    h. O valor de RB, ***OutB***, é apresentado tal como o valor de RA, nos dígitos 2, 1 e 0 do display de 7 segmentos em formato hexadecimal, mas apenas se o botão de pressão **`BTN(1)`**, i.e., o botão da esquerda, estiver pressionado.

6) Implemente o circuito na placa de desenvolvimento. Para tal, siga as instruções disponibilizadas no "**Guia de Implementação de Circuitos na Placa de Desenvolvimento**". Note que o interruptor ON/OFF da placa deve estar na posição ON.
   Nota: *durante a síntese do circuito, a ferramenta poderá indicar um conjunto de avisos (warnings) e erros. Os erros deverão ser **todos** corrigidos; os warnings podem, em geral, ser ignorados, sendo que alguns são originados pelo facto de ter entradas/saídas no ar*.
7) Verifique o correto funcionamento do circuito. Mostre-o ao docente. Comente.

Responda à Pergunta 5 na folha de respostas.

3. PROJETO DE UM CIRCUITO CONTROLADOR (SÍNTESE)

Nesta parte do trabalho, deverá executar as várias tarefas de projeto enquadradas no circuito analisado na secção 2.

Sempre que forem pedidas alterações a blocos existentes, aconselha-se que seja criada uma entidade nova (copiando o ficheiro e alterando o nome da entidade) em vez de alterar os blocos fornecidos. Por exemplo, se forem pedidas alterações no controlador (controlo.vhd), deve criar uma cópia do ficheiro, por exemplo controlo_v2.vhd, e renomear a entidade para controlo_v2.

3.1 – PROJETO DE OPERAÇÕES: float_reg

Para permitir a utilização de algoritmos mais complexos na implementação da soma de números em vírgula flutuante estudada neste trabalho, nomeadamente a normalização da saída, o funcionamento dos registos para números com representação em vírgula flutuante (float_reg.vhd) deve ser modificado de modo a poder executar o conjunto de operações indicado na tabela em baixo (a negrito estão assinaladas as novas operações).

Projete as alterações necessárias ao circuito original de forma a cumprir as novas especificações. Implemente essas modificações num novo módulo (float_reg_v2) em VHDL e crie o testbench correspondente para testar as funcionalidades do novo circuito. Confirme o correto funcionamento através de simulação.

Tabela 10 – Operações a implementar pelo Registo.

OP	Operação	Valor antes do flanco			Valor após o flanco
		arithmetic_in(4:0)	num_in(7:0)	num(7:0)	num(7:0)
000	Hold	A(4:0)	B(7:0)	N(7:0)	N(7:0)
001	Load M	A(4:0)	B(7:0)	N(7:0)	A(4:0) & N(2:0)
010	**SLA M**	**A(4:0)**	**B(7:0)**	**N(7:0)**	**N(6:3) & '0' & N(2:0)**
011	**SRA M**	**A(4:0)**	**B(7:0)**	**N(7:0)**	**N(7) & N(7:4) & N(2:0)**
100	Load E	A(4:0)	B(7:0)	N(7:0)	N(7:3) & A(2:0)
101	Load	A(4:0)	B(7:0)	N(7:0)	B(7:0)
110	**SLA M; Ld E**	**A(4:0)**	**B(7:0)**	**N(7:0)**	**N(6:3) & '0' & A(2:0)**
111	SRA M; Ld E	A(4:0)	B(7:0)	N(7:0)	N(7) & N(7:4) & A(2:0)

Responda às perguntas 6 e 7 na folha de respostas.

3.2 – PROJETO DE INSTRUÇÕES

Dependendo do seu horário de laboratório, considere as instruções indicadas na tabela abaixo. Indique na folha de respostas qual a o valor dos sinais (ra_op, rb_op, rz_ce, alu_op) que definem a operação que deve ser realizada nos registos (RA, RB, e RZ) e na ALU1, respetivamente, de forma a implementar essas instruções.

Turno	Instrução 1	Instrução 2
Segunda	$mB \leftarrow SLA\ mB;\ A = eA + 1$	$when(RZ = 1)\ mA \leftarrow Ma\ SRA$
Quarta	$mA \leftarrow SLA\ mA;\ eA = eA + 1$	$when(RZ = 0)\ mB \leftarrow mB\ SRA$
Quinta	$mB \leftarrow SRA\ mA;\ eA \leftarrow eA + 1$	$when(RZ = 0)\ eA = eB - 1$
Sexta	$mA \leftarrow SRA\ mA; eA \leftarrow eA + 1$	$when(RZ = 1)\ eB = eA + 1$

Responda à Pergunta 8 na folha de respostas.

3.3 – PROJETO DE UM CIRCUITO DE CONTROLO

No algoritmo de controlo fornecido (ver Tabela 4), a instrução nZ_SRAmDECe é repetida 5 vezes. Este número pode ser reduzido para 1 se o circuito do Controlador (CTRL) controlar também a próxima instrução a ser executada, i.e. se puder fazer também o carregamento (LOAD) do contador de instruções (ICTR). Desta forma, considerando as saídas LD_I(3:0) e CTR_LD adicionais do controlador (ligadas ao contador), é possível implementar uma instrução que define qual a próxima instrução a executar. Neste caso a ideia é: repetir a instrução nZ_SRAmDECe as vezes que forem necessárias, i.e., enquanto o valor de eB for diferente de zero. Com esta alteração, é possível reduzir o número de instruções na implementação do algoritmo de soma de números de vírgula flutuante.

Na tabela abaixo está indicada uma sequência alternativa de instruções, incluindo o carregamento do valor '2' no contador (nZ_LOAD_2), que permite reutilizar a instrução 2 (nZ_SRAmDECe) várias vezes, enquanto a RZ estiver a '0'.

Tabela 11 – Tabela do controlo.

I(3:0)	RZ	CTR_LD	LD_I(3:0)	RZ_CE	RA_OP(2:0)	RB_OP(2:0)	ALU_OP(1:0)	Descrição
0000	**X**	?	?	X	X	X	X	Reserved
0001	**X**	?	?	1	000	100	01	SUBe
0010	**0**	?	?	1	000	111	10	nZ_SRAmDECe
	1	?	?	0	000	000	X	
0011	**0**	?	?	?	?	?	?	nZ_LOAD_2
	1	?	?	?	?	?	?	
0100	**X**	?	?	1	001	000	00	ADD (mA, mB)
0101	**X**	?	?	0	000	000	X	NOP
...	**X**	?	?	0	000	000	X	NOP
1111	**X**	?	?	0	000	000	X	NOP

Responda à Pergunta 9 na folha de respostas.

Projete um controlador, utilizando o mínimo de lógica, que permite a execução desta nova sequência de operações, e implemente-o em VHDL, adicionando as saídas CTR_LD, e LD_I(3:0) ao controlador e ligando-as ao ICTR. Simule o circuito obtido e crie também o ficheiro de programação da placa (seguindo novamente as indicações do ponto 2.7)

Responda às Pergunta 10, 11, 12 na folha de respostas.

3.4 – Funcionalidades avançadas

Apesar do circuito implementado possuir já um nível de complexidade razoável, há ainda alguns aspetos que o algoritmo não comtempla. Um deles prende-se com a soma de números em vírgula flutuante em que o overflow na soma das mantissas não dá origem a um overflow na representação. Por exemplo, na soma de 16 + 16, i.e., ($8*2^1 + 8*2^1$ na representação de vírgula flutuante utilizada) o resultado obtido pelo circuito implementado será $-16*2^1$, em vez de $8*2^2$. Ora sendo esta última representação possível, o resultado gerado deveria representar o resultado desta soma corretamente.

Este problema pode ser resolvido com o hardware já implementado, apenas considerando um registo RV adicional para guardar um sinal de overflow vindo da ALU controlado pelo mesmo sinal de enable que RZ e um novo controlador utilizando as funcionalidades novas do registo de números de vírgula flutuante (float_reg_v2), bem como uma instrução de LOAD. O novo controlador deve

testar o overflow na soma das mantissas e, nesse caso, deslocar ambas as mantissas para a direita e incrementar o expoente da saída antes de voltar a calcular a sua soma, se não houver overflow deve executar logo a soma.

Sem alterar o código VHDL, indique qual a sequência de operações e a respetivas saídas do circuito de controlo na tabela 2 da folha de respostas.

Responda à Pergunta 13 na folha de respostas.

ALTERAÇÕES À VERSÃO 2.0

Enunciado:

- Página 5, nova Figura 3.
- Página 6, Primeira frase: Modificação de "num(2:3)" para "num(2:0)";
- Página 6, Tabela 5: Na operação 001 modificação de "A(7:3)" para "A(4:0)";
- Página 6, Figura 5: Corrigida discrepância entre logigrama e descrição em vhdl no 2º MUX dos registos da mantissa;
- Página 8, Tabela 8: O operador ">>" para o deslocamento para a direita aritmético foi substituído por SRA;
- Página 12, Secção 3.2: O operador ">>" para o deslocamento para a direita aritmético foi substituída por SRA, e o operador "<<" para o deslocamento para a esquerda aritmético foi substituído por SLA;

ALTERAÇÕES À VERSÃO 2.1

Enunciado:

- Página 8, Tabela 8: Corrigida a legenda para "Onde Z(X) = '1' se X for zero ou '0' se X for diferente de zero."
- Página 12, Tabela 10: Na operação 001 modificação de "A(7:3)" para "A(4:0)";
- Página 13, Tabela 13: Substituído o "don't care" na saída RZ_CE por valores definidos;
- Página 13, Tabela 13: Substituído o "0" nas saídas RA_OP e RB_OP por "000";