UNIVERSIDADE da MADEIRA

Componente de Avaliação P1 (15%) de Arquitetura de Computadores

Ano letivo: 2019/2020

Data de entrega: 22-03-2020

Data de discussão: 25-03-2020

1. Descrição do processador

Neste primeiro trabalho de avaliação pretende-se que seja realizado um processador básico, com um conjunto mínimo de instruções, em linguagem de descrição de *hardware* (VHDL). A implementação do processador será realizada no programa ISE da Xilinx, sendo que a simulação e o teste serão realizados no ISim e em FPGA (SPARTAN 3E, SPARTAN 6 e ARTIX 7), respetivamente.

Na Figura 1 é mostrado o diagrama de blocos do processador. Este processador consegue manipular diretamente dados de 8 bits.

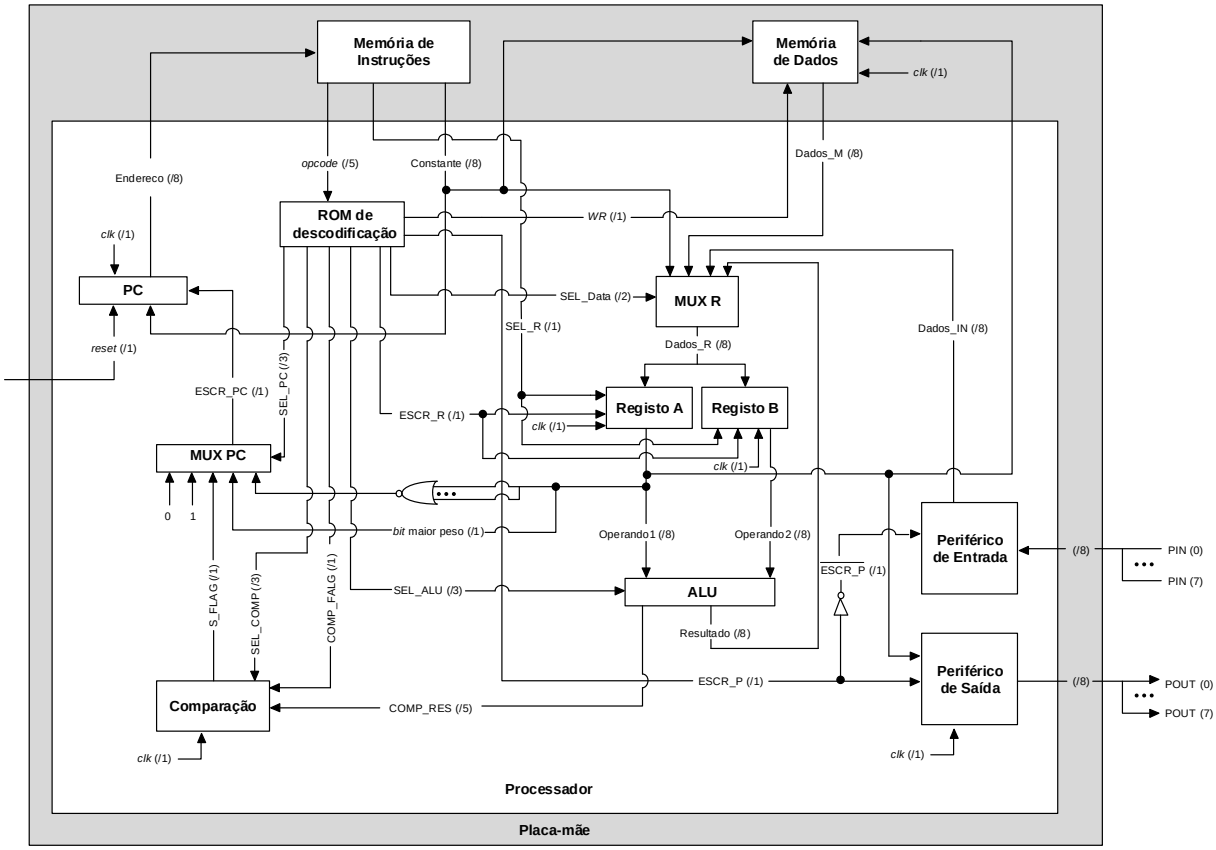

Figura 1 – Diagrama de blocos do processador a implementar.

O processador é constituído por diversos módulos que, em conjunto com a memória de instruções e a memória de dados, formam a placa-mãe. De seguida é apresentada uma breve descrição de cada bloco.

1.1. Periférico de Entrada

Os periféricos permitem a comunicação do processador com o exterior. O periférico de entrada permite ao utilizador inserir dados para posteriormente serem realizadas operações com os mesmos. Este periférico é controlado pelo sinal $\overline{ESCR\_P}$, de 1 *bit*, que quando está a '1' faz com que seja realizada uma leitura aos dados de entrada, *PIN*, de 8 *bits*, colocando-os na saída do módulo, *Dados_IN*, também de 8 *bits*.

1.2. Periférico de Saída

O periférico de saída permite que o utilizador veja os resultados dos programas executados pelo processador. Este módulo é controlado pelo sinal *ESCR_P*, de 1 *bit*, que quando está a '1', na transição ascendente do relógio (*clk*), escreve no sinal de saída, *POUT*, de 8 *bits*, o valor do sinal à entrada do módulo, *Operando1*, também de 8 *bits*.

1.3. *Multiplexer* dos Registos (Mux R)

O *multiplexer* dos registos é responsável por encaminhar um dos quatro sinais disponíveis, de 8 *bits*, à sua entrada (*Resultado*, *Dados_IN*, *Dados_M* e *Constante*) para apresentar na sua saída, *Dados_R*, de 8 *bits*. O sinal a encaminhar depende do valor na entrada de seleção, o sinal *SEL_Data*, de 2 *bits*, de acordo com a Tabela 1.

Tabela 1 – Sinal de saída do Mux R em função do sinal *SEL_Data*.

SEL_Data	***Dados_R***
00	*Resultado*
01	*Dados_IN*
10	*Dados_M*
11	*Constante*

1.4. Registos A e B

A escrita, nos registos A e B, é controlada pelo sinal *ESCR_R*, de 1 *bit*, que quando estiver a '1' permite que o valor presente no sinal de entrada, *Dados_R*, de 8 *bits*, seja guardado no registo especificado pelo sinal *SEL_R*, de 1 *bits*, como indicado na Tabela 2, quando ocorrer uma transição ascendente do sinal de relógio (*clk*).

Tabela 2 – Sinal de seleção de escrita nos registos A e B.

SEL_R	**Registo a ser escrito**
0	*Registo A*
1	*Registo B*

Além da escrita, os registos estão continuamente a efetuar leituras. A saída *Operando1*, de 8 *bits*, apresentará o valor guardado no *registo A* e a saída *Operando2*, de 8 *bits*, apresentará o valor guardado no *registo B*.

1.5. Unidade Lógica e Aritmética (ALU)

A Unidade Aritmética e Lógica de um processador é responsável pela realização das operações aritméticas e lógicas. Neste projeto pretende-se que a ALU desenvolvida seja capaz de realizar as operações apresentadas na Tabela 3, com os dois sinais de entrada de 8 *bits*, *Operando1* e *Operando2*, representando números inteiros com sinal, sendo que os sinais de saída da ALU são determinados pelo sinal de seleção *SEL_ALU* de 3 *bits*. A saída *Resultado*, de 8 *bits*, será atualizada no caso das operações de soma, subtração, AND, OR e XOR. A Saída *COMP_RES*, de 5 *bits*, será atualizada apenas quando é executada uma comparação. Cada um dos seus *bits* indica o resultado de uma das cinco comparações (como será apresentado na secção seguinte).

Tabela 3 – Operações da ALU.

SEL_ALU	**Operação**		
000	*Operando1*	+	*Operando2*
001	*Operando1*	–	*Operando2*
010	*Operando1*	**AND**	*Operando2*
011	*Operando1*	**OR**	*Operando2*
100	*Operando1*	**XOR**	*Operando2*
101	*Operando1*	>	*Operando2*
	Operando1	>=	*Operando2*
	Operando1	=	*Operando2*
	Operando1	<=	*Operando2*
	Operando1	<	*Operando2*

1.6. Comparação

O funcionamento do módulo de comparação é idêntico ao dos registos. Este guarda o sinal de entrada, *COMP_RES*, de 5 *bits*, quando o sinal *COMP_FLAG* está a '1' e na transição ascendente do sinal de relógio. Este está continuamente a efetuar leituras, e apenas um dos 5 *bits* do sinal guardado é encaminhado para a saída, *S_FLAG*, de 1 *bit*. Esse *bit* é determinado através do sinal de seleção *SEL_COMP*, de 3 *bits*, do modo apresentado na Tabela 4.

Tabela 4 – Sinal de saída do *multiplexer* de comparação em função do sinal de seleção *SEL_COMP*.

SEL_COMP	***S_FLAG***
000	*COMP_RES (0)* (>)
001	*COMP_RES (1)* (>=)
010	*COMP_RES (2)* (=)
011	*COMP_RES (3)* (<=)
100	*COMP_RES (4)* (<)

1.7. Contador de programa (PC)

O contador de programa indica qual é a posição atual da sequência de execução de um programa. Assim, este envia, na transição ascendente de cada ciclo de relógio, a sua saída *Endereco*, de 8 *bits*, à Memória de Instruções. A sequência de execução será incrementada de um em um caso a entrada *ESCR_PC*, de 1 *bit*, esteja a '0', caso contrário (*ESCR_PC* a '1'), a saída do PC corresponderá ao valor da entrada Constante, de 8 *bits*, e ocorrerá um salto para o endereço de instrução indicado por esse valor. A entrada *Reset*, de 1 *bit*, quando ativa, permite voltar ao início do programa.

1.8. *Multiplexer* do *Program Counter* (Mux_PC)

O *multiplexer* do contador de programa tem o objetivo de indicar ao PC se é para realizar um salto ou incrementar o contador, através do sinal de saída *ESCR_PC*, de 1 *bit*, a '1' ou a '0', respetivamente. Este apresenta um sinal de seleção *SEL_PC*, de 3 *bits*, que indica qual dos valores de entrada deve passar para a saída, como indicado na Tabela 5.

Tabela 5 – Valor de saída do *MUX_PC* em função do sinal de seleção *SEL_PC*.

SEL_PC	***ESCR_PC***
000	'0'
001	'1'
010	*S_FLAG*
011	*Operando1*(7)
100	NOT (*Operando1*(7) OR *Operando1*(6) OR *Operando1*(5) OR *Operando1*(4) OR *Operando1*(3) OR *Operando1*(2) OR *Operando1*(1) OR *Operando1*(0))

1.9. ROM de descodificação (ROM)

A ROM de descodificação é responsável por fornecer aos restantes blocos os sinais de controlo. Esta recebe o sinal *opcode* da memória de instruções, de 5 *bits*, e coloca na sua saída os valores correspondentes aos seguintes sinais de controlo: *SEL_PC*, de 3 *bits*, *SEL_COMP*, de 3 *bits*, *COMP_FLAG*, de 1 *bit*, *SEL_ALU*, de 3 *bits*, *ESCR_R*, de 1 *bit*, *SEL_Data*, de 2 *bits*, *ESCR_P*, de 1 *bit*, *WR*, de 1 *bit*. Na Tabela 6 encontra-se a relação entre o sinal *opcode*, e os sinais de controlo. É também indicada em *assembly*, a instrução correspondente. O registo *Ri* corresponde ao registo indicado pelo sinal *SEL_R*.

Tabela 6 – Relação entre o sinal *opcode* e os sinais de saída da ROM.

Opcode	Instrução	SEL_ALU	ESCR_P	SEL_Data	ESCR_R	WR	SEL_PC	COMP_FLAG	SEL_COMP
Periféricos									
00000	LDP *Ri*	XXX	0	01	1	0	000	0	XXX
00001	STP *RA*	XXX	1	XX	0	0	000	0	XXX
Leitura e Escrita									
00010	LD *Ri*, constante	XXX	0	11	1	0	000	0	XXX
00011	LD *Ri*, [constante]	XXX	0	10	1	0	000	0	XXX
00100	ST [constante], *RA*	XXX	0	XX	0	1	000	0	XXX
Lógica e Aritmética									
00101	ADD *RA*, *RB*	000	0	00	1	0	000	0	XXX
00110	SUB *RA*, *RB*	001	0	00	1	0	000	0	XXX
00111	AND *RA*, *RB*	010	0	00	1	0	000	0	XXX
01000	OR *RA*, *RB*	011	0	00	1	0	000	0	XXX
01001	XOR *RA*, *RB*	100	0	00	1	0	000	0	XXX
01010	CMP *RA*, *RB*	101	0	XX	0	0	000	1	XXX
Salto									
01011	JG constante	XXX	0	XX	0	0	010	0	000
01100	JGE constante	XXX	0	XX	0	0	010	0	001
01101	JE constante	XXX	0	XX	0	0	010	0	010
01110	JLE constante	XXX	0	XX	0	0	010	0	011
01111	JL constante	XXX	0	XX	0	0	010	0	100
10000	JMP constante	XXX	0	XX	0	0	001	0	XXX
10001	JN *RA*, constante	XXX	0	XX	0	0	011	0	XXX
10010	JZ *RA*, constante	XXX	0	XX	0	0	100	0	XXX
Outros									
Outros	NOP	XXX	0	XX	0	0	000	0	XXX

A Tabela 7 apresenta a descrição de cada uma das instruções apresentadas anteriormente.

Tabela 7 – Descrição das instruções do processador.

Instrução	Descrição
LDP *Ri*	Escreve no registo *Ri* uma cópia do valor do Periférico de entrada
STP *RA*	Escreve no periférico de saída uma cópia do registo *RA*
LD *Ri*, constante	Escreve no registo *Ri* a constante
LD *Ri*, [constante]	Escreve no registo *Ri* uma cópia da célula da RAM indicada pela constante
ST [constante], *RA*	Escreve na célula da RAM indicada por constante uma cópia do registo *RA*
ADD *RA*, *RB*	Soma o registo *RA* com o registo *RB* e escreve o resultado no registo *RA*
SUB *RA*, *RB*	Subtrai ao registo *RA* o registo *RB* e escreve o resultado no registo *RA*
AND *RA*, *RB*	Efetua a conjugação lógica *bit* a *bit* do registo *RA* com o registo *RB* e escreve o resultado no registo *RA*
OR *RA*, *RB*	Efetua a disjunção lógica *bit* a *bit* do registo *RA* com o registo *RB* e escreve o resultado no registo *RA*
XOR *RA*, *RB*	Efetua a disjunção exclusiva lógica *bit* a *bit* do registo *RA* com o registo *RB* e escreve o resultado no registo *RA*
CMP *RA*, *RB*	Compara o registo *RA* com o registo *RB* e escreve o resultado no módulo de Comparação
JG constante	Salta para a instrução indicada pela constante se o valor do módulo de Comparação referente à comparação “>” for “1”
JGE constante	Salta para a instrução indicada pela constante se o valor do módulo de Comparação referente à comparação “>=” for “1”
JE constante	Salta para a instrução indicada pela constante se o valor do módulo de Comparação referente à comparação “=” for “1”
JLE constante	Salta para a instrução indicada pela constante se o valor do módulo de Comparação referente à comparação “<=” for “1”
JL constante	Salta para a instrução indicada pela constante se o valor do módulo de Comparação referente à comparação “<” for “1”
JMP constante	Salta para a instrução indicada pela constante
JN *RA*, constante	Salta para a instrução indicada pela constante se o valor de *RA* for negativo
JZ *RA*, constante	Salta para a instrução indicada pela constante se o valor de *RA* for “0”
NOP	Não executa nenhuma operação e apenas incrementa o contador do PC

1.10. Memória de Instruções

Na memória de instruções ficam armazenadas as instruções do programa a ser executado. A sua dimensão é de 14 *bits*, sendo endereçada pelo sinal *Endereco*, de 8 *bits*, e disponibiliza à sua saída o *opcode*, de 5 *bits*, o sinal *SEL_R*, de 1 *bit* e o sinal Constante, de 8 *bits*.

1.11. Memória de Dados (RAM)

A memória de dados permite guardar os dados presentes no sinal de entrada *Operando1*, de 8 *bits*, quando o sinal *WR* estiver a '1', na transição ascendente do sinal de relógio (*clk*), na posição de memória indicada pelo sinal de entrada *Constante*, de 8 *bits*. Quando o sinal *WR* está a '0' é realizada uma leitura à posição de memória indicada pelo sinal *Constante* e esse valor é atribuído ao sinal de saída *Dados_M*, de 8 *bits*.

2. Simulação e Implementação

O processador descrito é um componente eletrónico que pode ser implementado com componentes eletrónicos lógicos discretos, num único *chip*, ou em dispositivos lógicos programáveis. Na fase de desenvolvimento de processadores é vantajoso implementar a arquitetura do processador em dispositivos lógicos, pois nessa fase por vezes é necessário fazer várias alterações para otimizar a versão final do processador. Neste trabalho prático vão ser utilizadas as placas de desenvolvimento SPARTAN 6, a SPARTAN 3E e a ARTIX 7 para implementar o processador em FPGAs, sendo necessário utilizar o sinal de relógio (*clk*) das mesmas. Para cada uma das FPGA disponíveis, os pinos de entrada e saída dos periféricos encontram-se indicados na Tabela 8.

Tabela 8 - Pinos de entrada e saída para as FPGA disponíveis.

FPGA	**Entrada**									
	PIN(7)	PIN(6)	PIN(5)	PIN(4)	PIN(3)	PIN(2)	PIN(1)	PIN(0)	Reset	clk
Spartan 6 (Atlys)	E4	T5	R5	P12	P15	C14	D14	A10	F5	L15
Spartan 3E (Xilinx)	D18	V4	H13	K17	N17	H18	L14	L13	V16	C9
Spartan 3E (Nexys 2)	R17	N17	L13	L14	K17	K18	H18	G18	H13	B8
Spartan 6 (Nexys 3)	T5	V8	U8	N8	M8	V9	T9	T10	B8	V10
Artix 7 (Nexys 4/A7)	R13	U18	T18	R17	R15	M13	L16	J15	N17	E3

FPGA	**Saídas**							
	POUT(7)	POUT(6)	POUT(5)	POUT(4)	POUT(3)	POUT(2)	POUT(1)	POUT(0)
Spartan 6 (Atlys)	N12	P16	D4	M13	L14	N14	M14	U18
Spartan 3E (Xilinx)	F9	E9	D11	C11	F11	E11	E12	F12
Spartan 3E (Nexys 2)	R4	F4	P15	E17	K14	K15	J15	J14
Spartan 6 (Nexys 3)	T11	R11	N11	M11	V15	U15	V16	U16
Artix 7 (Nexys 4/A7)	U16	U17	V17	R18	N14	J13	K15	H17

3. Teste

Os programas em linguagem *assembly* têm de ser convertidos para código máquina para programar a memória de instruções. O programa da Tabela 9 serve para testar o funcionamento do processador. No entanto, podem e devem experimentar outros programas.

Tabela 9 – Instruções de teste do projeto.

Endereço	Instrução (*assembly*)	Instrução (código máquina)
00000000	LD RA, 3	
00000001	ST [0], RA	
00000010	LD RA, 20	
00000011	ST [1], RA	
00000100	LDP RA	
00000101	JN RA, 22	
00000110	LD RB, 40	
00000111	CMP RA, RB	
00001000	JGE 27	
00001001	ST [2], RA	
00001010	ST [3], RA	
00001011	LD RA, [0]	
00001100	LD RB, 1	
00001101	SUB RA, RB	
00001110	JZ RA, 20	
00001111	ST [0], RA	
00010000	LD RA, [3]	
00010001	LD RB, [2]	
00010010	ADD RA, RB	
00010011	JMP 10	
00010100	LD RA, [3]	
00010101	JMP 29	
00010110	LD RB, -1	
00010111	XOR RA, RB	
00011000	LD RB, 1	
00011001	ADD RA, RB	
00011010	JMP 29	
00011011	LD RB, [1]	
00011100	SUB RA, RB	
00011101	STP RA	
00011110	JMP 30	

4. Avaliação e informações relevantes

- O projeto deve ser realizado individualmente ou em grupo de 2 alunos;
- Este trabalho representa 15% na nota final e tem nota mínima de 8 valores.
- O relatório poderá ter no máximo 5 páginas (sem contar com os anexos, capa e índice), contendo a descrição do trabalho realizado. A sua estrutura deverá incluir os seguintes tópicos:

    1. Introdução;
    2. Objetivos;

3. Desenvolvimento;
4. Discussão de resultados;
5. Conclusão;
6. Bibliografia;
7. Anexo A: tabela das instruções de teste (Tabela 9) preenchida e a respetiva simulação (com os sinais de entrada e saída em decimal e legíveis);
8. Anexo B: deverá conter a listagem de código dos módulos em VHDL.

· O processador deverá funcionar corretamente no simulador (ISim) e na FPGA, usando o sinal de relógio da mesma.

· O relatório deverá ser entregue ao Gabinete de Apoio ao Estudante ("trabalhos@uma.pt"). A cópia do trabalho implica a anulação do mesmo.

· A discussão do trabalho é individual, sendo necessário apresentar o funcionamento do mesmo no simulador e na FPGA (o aluno deverá iniciar a discussão já com o programa a correr no computador e uma cópia do enunciado). Caso não possua computador pessoal poderá usar um dos computadores da Universidade, devendo avisar previamente os docentes.

Bibliografia

J. Delgado e C. Ribeiro, Arquitectura de Computadores, FCA, 2007

D. M. Harris and S. L. Harris, Digital Design and Computer Architecture, Elsevier MK, 2007