UNIVERSIDADE da MADEIRA

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

Ano letivo: 2020/2021

Data de entrega e discussão: 14-04-2020

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 e a simulação realizada no ISim.

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

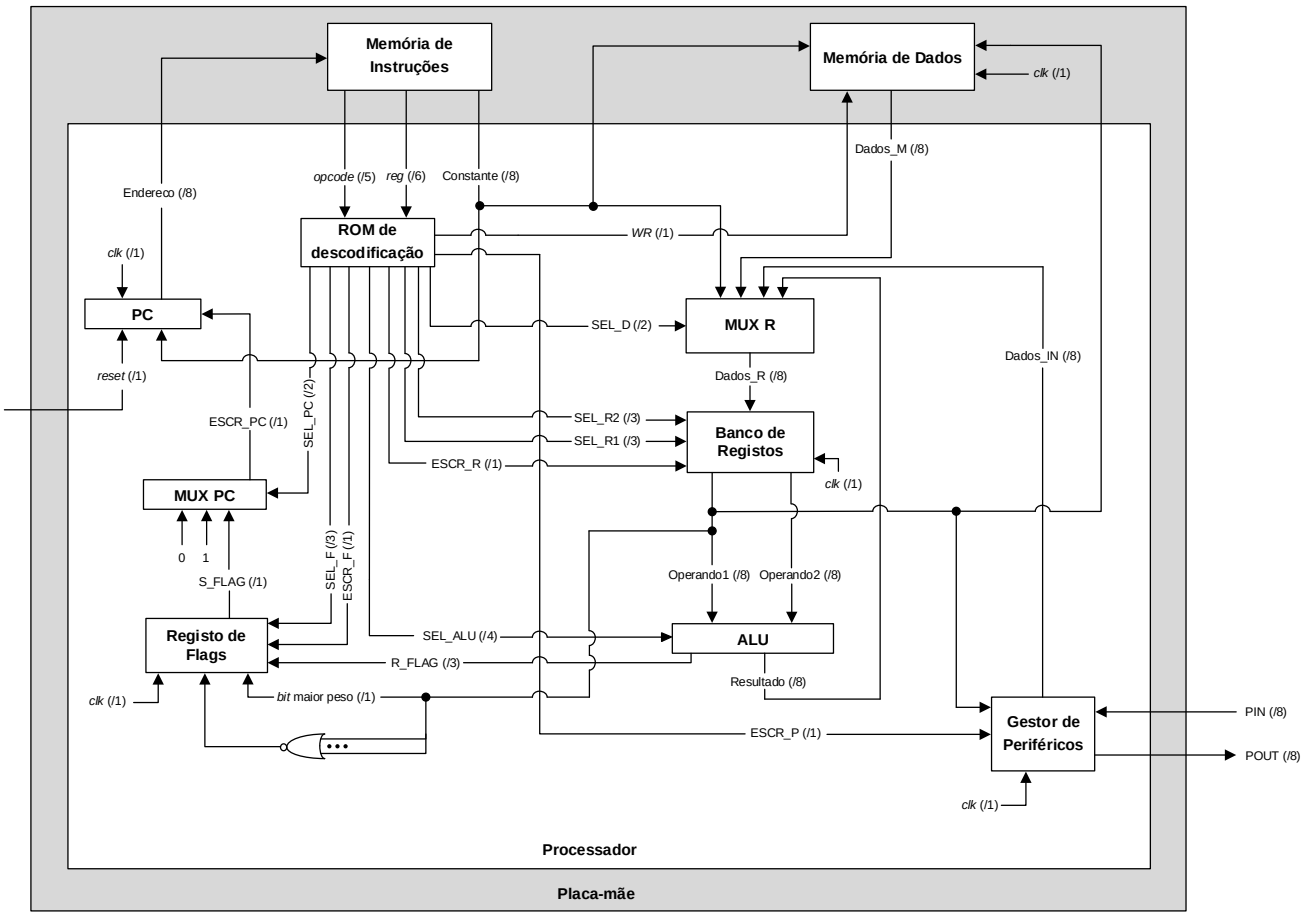

Figura 1 – Diagrama de blocos do processador a implementar.

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. Gestor de Periféricos

Os periféricos permitem a comunicação do processador com o exterior. O Gestor de Periféricos permite ao utilizador inserir dados para posteriormente serem realizadas operações com os mesmos e 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*, de 8 *bits*. Quando *ESCR_P* está a '0' é realizada uma leitura aos dados de entrada, *PIN*, de 8 *bits*, colocando-os na saída do módulo, *Dados_IN*, *de* 8 *bits*.

1.2. *Multiplexer* dos Registos (Mux R)

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

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

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

1.3. Banco de Registos

O banco de registos possui 8 registos de 8 *bits* cada (R0, R1, R2, R3, R4, R5, R6 e R7). A escrita num registo 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_R1*, de 3 *bits*, como indicado na Tabela 2, quando ocorrer uma transição ascendente do sinal de relógio (*clk*).

Tabela 2 – Sinal de seleção de escrita no banco de registos.

SEL_R1	**Registo a ser escrito**
000	R0
001	R1
010	R2
011	R3
100	R4
101	R5
110	R6
111	R7

Além da escrita, o banco de registos está continuamente a efetuar leituras. A saída *Operando1*, de 8 *bits*, apresentará o valor do registo especificado pelo sinal *SEL_R1* (denominado por *Ri* na Tabela 7), e a saída *Operando2*, de 8 *bits*, apresentará o valor do registo especificado pelo sinal *SEL_R2* (denominado por *Rj* na Tabela 7), como indicado na Tabela 3.

Tabela 3 – Sinal de seleção de leitura no banco de registos.

SEL_R1	**Registo a ser lido (*Ri*)**	***SEL_R2***	**Registo a ser lido (*Rj*)**
000	R0	000	R0
001	R1	001	R1
010	R2	010	R2
011	R3	011	R3
100	R4	100	R4
101	R5	101	R5
110	R6	110	R6
111	R7	111	R7

1.4. Unidade Lógica e Aritmética (ALU)

A Unidade Lógica e Aritmética 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 4, 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 4 *bits*. A saída *Resultado*, de 8 *bits*, será atualizada no caso das operações de lógicas e aritméticas. A Saída *R_FLAG*, de 3 *bits*, será atualizada apenas quando é executada uma comparação, sendo que cada um dos seus *bits* indica o resultado de uma das três comparações (como indicado, também, na Tabela 4).

Tabela 4 – Operações da ALU.

SEL_ALU	**Operação**			**Sinal de Saída** (onde fica guardado o resultado da operação)
0000	*Operando1*	+	*Operando2*	*Resultado*
0001	*Operando1*	–	*Operando2*	*Resultado*
0010	*Operando1*	**AND**	*Operando2*	*Resultado*
0011	*Operando1*	**NAND**	*Operando2*	*Resultado*
0100	*Operando1*	**OR**	*Operando2*	*Resultado*
0101	*Operando1*	**NOR**	*Operando2*	*Resultado*
0110	*Operando1*	**XOR**	*Operando2*	*Resultado*
0111	*Operando1*	**XNOR**	*Operando2*	*Resultado*
1000	**shift_left** *(Operando1, Operando2)*			*Resultado*
1001	**shift_right** *(Operando1, Operando2)*			*Resultado*
1010	*Operando1*	<	*Operando2*	*R_FLAG(0)*
	Operando1	>	*Operando2*	*R_FLAG(1)*
	Operando1	/=	*Operando2*	*R_FLAG(2)*

1.5. Registo de Flags

O Registo de Flags tem um funcionamento idêntico a um registo. Este guarda os sinais de entrada quando o sinal *ESCR_F* está a '1' e na transição **descendente** do sinal de relógio. Este módulo está continuamente a efetuar leituras, e apenas um dos 5 *bits* guardados é encaminhado para a saída, *S_FLAG*, de 1 *bit*. Esse *bit* é determinado através do sinal de seleção *SEL_F*, de 3 *bits*, do modo apresentado na Tabela 5.

Tabela 5 – Sinal de saída do *Registo de Flags* em função do sinal de seleção *SEL_F*.

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

1.6. 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.

1.7. *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 2 *bits*, que indica qual dos valores de entrada deve passar para a saída, como indicado na Tabela 6.

Tabela 6 – Valor de saída do *MUX_PC* em função do sinal de seleção *SEL_PC*.

SEL_PC	***ESCR_PC***
00	*S_FLAG*
01	'1'
10	'0'

1.8. ROM de descodificação (ROM)

A ROM de descodificação é responsável por fornecer aos restantes blocos os sinais de controlo. Esta recebe, da memória de instruções, o sinal *opcode* de 5 *bits*, e o sinal *reg*, de 6 *bits*, e coloca na sua saída os valores correspondentes aos seguintes sinais de controlo: *SEL_PC*, de 2 *bits*, *SEL_F*, de 3 *bits*, *ESCR_F*, de 1 *bit*, *SEL_ALU*, de 4 *bits*, *ESCR_R*, de 1 *bit*, *SEL_D*, de 2 *bits*, *ESCR_P*, de 1 *bit* e *WR*, de 1 *bit*. Na Tabela 7 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_R1* e o registo *Rj* corresponde ao registo indicado pelo sinal *SEL_R2*. O sinal *SEL_R1*, de 3 *bits*, corresponde aos três *bits* mais significativos de *reg* e o sinal *SEL_R2*, de 3 *bits*, corresponde aos três *bits* menos significativos do sinal *reg*.

Tabela 7 – Relação entre o sinal *opcode* e os sinais de saída da ROM.

Opcode	**Instrução**	**SEL_ALU**	**ESCR_P**	**SEL_D**	**ESCR_R**	**WR**	**SEL_PC**	**ESCR_F**	**SEL_F**
Periféricos									
00000	LDP *Ri*	XXXX	0	10	1	0	10	0	XXX
00001	STP *Ri*	XXXX	1	XX	0	0	10	0	XXX
Leitura e Escrita									
00010	LD *Ri*, constante	XXXX	0	00	1	0	10	0	XXX
00011	LD *Ri*, [constante]	XXXX	0	01	1	0	10	0	XXX
00100	ST [constante], *Ri*	XXXX	0	XX	0	1	10	0	XXX
Lógica e Aritmética									
00101	ADD *Ri*, *Rj*	0000	0	11	1	0	10	0	XXX
00110	SUB *Ri*, *Rj*	0001	0	11	1	0	10	0	XXX
00111	AND *Ri*, *Rj*	0010	0	11	1	0	10	0	XXX
01000	NAND *Ri*, *Rj*	0011	0	11	1	0	10	0	XXX
01001	OR *Ri*, *Rj*	0100	0	11	1	0	10	0	XXX
01010	NOR *Ri*, *Rj*	0101	0	11	1	0	10	0	XXX
01011	XOR *Ri*, *Rj*	0110	0	11	1	0	10	0	XXX
01100	NXOR *Ri*, *Rj*	0111	0	11	1	0	10	0	XXX
01101	shift_left(*Ri*, *Rj*)	1000	0	11	1	0	10	0	XXX
01110	shift_right(*Ri*, *Rj*)	1001	0	11	1	0	10	0	XXX
01111	CMP *Ri*, *Rj*	1010	0	XX	0	0	10	1	XXX
Salto									
10000	JG constante	XXX	0	XX	0	0	00	0	011
10001	JNE constante	XXX	0	XX	0	0	00	0	010
10010	JL constante	XXX	0	XX	0	0	00	0	100
10011	JMP constante	XXX	0	XX	0	0	01	0	XXX
10100	JN *Ri*, constante	XXX	0	XX	0	0	00	1	001
10101	JZ *Ri*, constante	XXX	0	XX	0	0	00	1	000
Outros									
Outros	NOP	XXX	0	XX	0	0	10	0	XXX

A Tabela 8 apresenta a descrição de cada uma das instruções apresentadas anteriormente.

Tabela 8 – Descrição das instruções do processador.

Instrução	Descrição
LDP *Ri*	Escreve no registo *Ri* uma cópia do valor de *PIN*
STP *Ri*	Escreve em *POUT* uma cópia do registo *Ri*
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], *Ri*	Escreve na célula da RAM indicada por constante uma cópia do registo *Ri*
ADD *Ri, Rj*	Soma o registo *Ri* com o registo *Rj* e escreve o resultado no registo *Ri*
SUB *Ri, Rj*	Subtrai ao registo *Ri* o registo *Rj* e escreve o resultado no registo *Ri*
AND *Ri, Rj*	Efetua a conjunção lógica *bit* a *bit* do registo *Ri* com o registo *Rj* e escreve o resultado no registo *Ri*
NAND *Ri, Rj*	Efetua a negação da conjunção lógica *bit* a *bit* do registo *Ri* com o registo *Rj* e escreve o resultado no registo *Ri*
OR *Ri, Rj*	Efetua a disjunção lógica *bit* a *bit* do registo *Ri* com o registo *Rj* e escreve o resultado no registo *Ri*
NOR *Ri, Rj*	Efetua a negação da disjunção lógica *bit* a *bit* do registo *Ri* com o registo *Rj* e escreve o resultado no registo *Ri*
XOR *Ri, Rj*	Efetua a disjunção exclusiva lógica *bit* a *bit* do registo *Ri* com o registo *Rj* e escreve o resultado no registo *Ri*
NXOR *Ri, Rj*	Efetua a negação da disjunção exclusiva lógica *bit* a *bit* do registo *Ri* com o registo *Rj* e escreve o resultado no registo *Ri*
shift_left(*Ri*,*Rj*)	Efetua o deslocamento aritmético para a esquerda de *Rj* bits do sinal *Ri* e escreve o resultado no registo *Ri*
shift_right(*Ri*,*Rj*)	Efetua o deslocamento aritmético para a direita de *Rj* bits do sinal *Ri* e escreve o resultado no registo *Ri*
CMP *Ri, Rj*	Compara o registo *Ri* com o registo *Rj* e escreve o resultado no Registo de Flags
JG constante	Salta para a instrução indicada pela constante se a *flag* referente à comparação “>” for “1”
JNE constante	Salta para a instrução indicada pela constante se a *flag* referente à comparação “/=” for “1”
JL constante	Salta para a instrução indicada pela constante se a *flag* referente à comparação “<” for “1”
JMP constante	Salta para a instrução indicada pela constante
JN *Ri*, constante	Salta para a instrução indicada pela constante se o valor de *Ri* for negativo
JZ *Ri*, constante	Salta para a instrução indicada pela constante se o valor de *Ri* for zero
NOP	Não executa nenhuma operação e apenas incrementa o contador do PC

1.9. Memória de Instruções

Na memória de instruções ficam armazenadas as instruções do programa a ser executado. A sua dimensão é de 19 *bits*, sendo endereçada pelo sinal *Endereco*, de 8 *bits*, e disponibiliza à sua saída o *opcode*, de 5 *bits*, o sinal *reg*, de 6 *bit* e o sinal Constante, de 8 *bits*.

1.10. 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. 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 R0, 14	
00000001	LD R1, 26	
00000010	LD R2, -1	
00000011	LD R3, 1	
00000100	ST [0], R0	
00000101	ST [36], R1	
00000110	LDP R0	
00000111	JN R0, 22	
00001000	JZ R0, 15	
00001001	LD R1, 29	
00001010	CMP R0, R1	
00001011	JL 15	
00001100	XOR R0, R2	
00001101	ADD R0, R3	
00001110	JMP 24	
00001111	LD R4, 2	
00010000	shift_left(R0, R4)	
00010001	NAND R0, R2	
00010010	ADD R0, R3	
00010011	LD R5, [0]	
00010100	SUB R0, R5	
00010101	JMP 24	
00010110	LD R6, [36]	
00010111	ADD R0, R6	
00011000	STP R0	
00011001	JMP 25	

3. 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 deverá 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).
- 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.

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