UNIVERSIDADE da MADEIRA

P1 Assessment Component (15%) of Computer Architecture

School year: 2022/2023

Delivery date and discussion: 15-03-2023

1. Processor Description

In this first evaluation work, it is intended that a basic processor be created, with a minimum set of instructions, in a hardware description language (VHDL). The implementation of the processor will be carried out in Xilinx's ISE program, and the simulation and testing will be carried out in ISim and in FPGA (SPARTAN 3E, SPARTAN 6, ARTIX 7 and NEXYS A7), respectively.

Figure 1 shows the block diagram of the processor. This processor can directly handle 8-bit data.

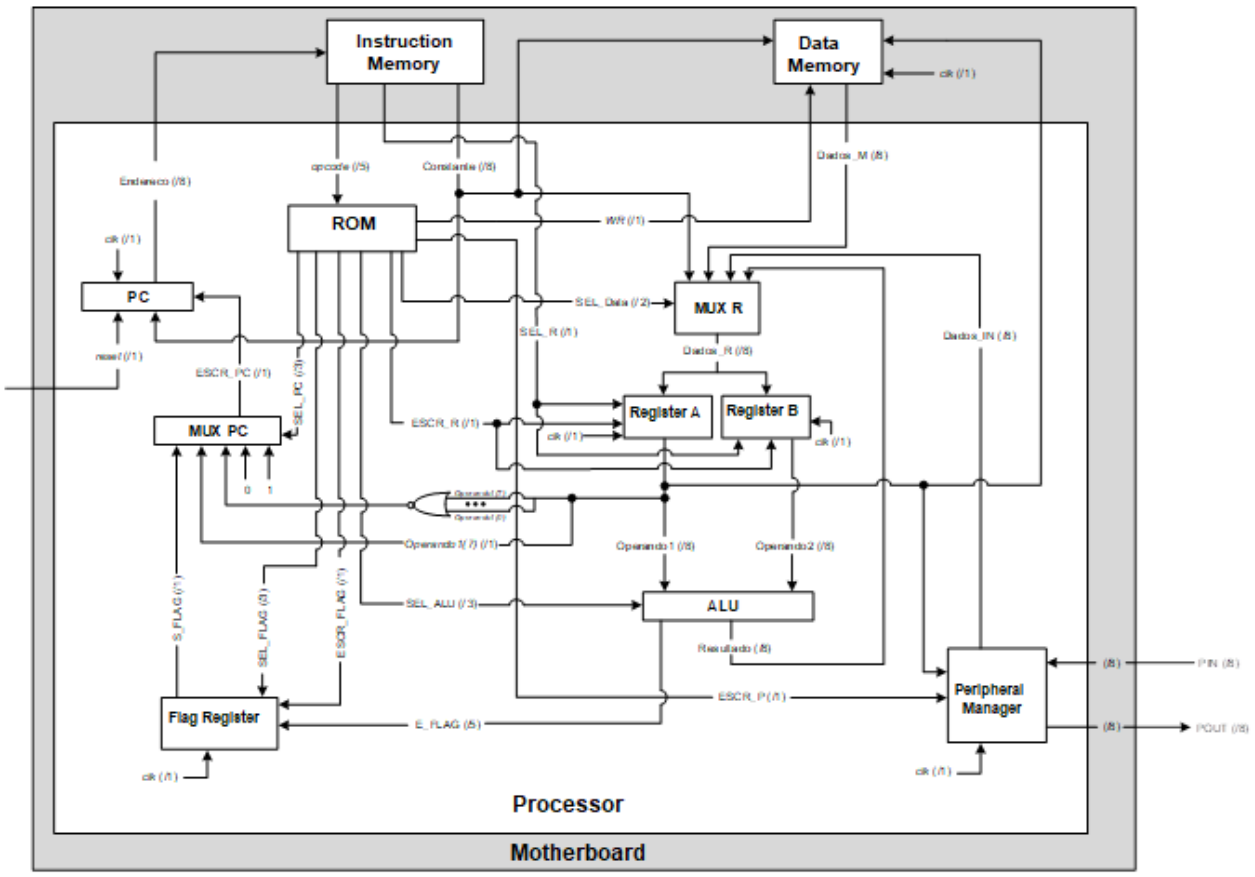

Figure 1 – Block diagram of the processor to be implemented.

The processor is made up of several modules that, together with the instruction memory and the data memory, form the motherboard. Below is a brief description of each block.

1.1. Peripheral Manager

The Peripheral Manager allows the processor to communicate with the outside world. This allows the user to insert data to later perform operations with them, and allows the user to see the results of programs executed by the processor. This module is controlled by the signal *ESCR_P*, of 1 bit, which when set to '0' causes a reading of the input data to be carried out *PIN*, of 8 bits, placing them in the output module, *Dados_IN*. On the other hand, when the signal *ESCR_P* is at '1', on the rising edge of the clock (*clk*), writes to the output signal, *POUT*, of 8 bits, the value of the input signal *Operando1*, of 8 bits.

1.2. *Multiplexer* Registers (Mux R)

Multiplexer of registers is responsible for forwarding a signal, of 8 bits, of the four available at your input (*Constante*, *Data_M*, *Data_IN* and *Resultado*) to give on your output, *Data_R*, of 8 bits. The signal to forward depends on the value in the select input, *SEL_Data*, of 2 bits, according to Table 1.

Table 1 – Mux R output signal depending on the signal *SEL_Data*.

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

1.3. A and B Registers

The writing, in *Registo A* and *B*, is controlled by the signal *ESCR_R*, of 1 bit, which when set to '1' allows the value present in the input signal, *Dados_R*, of 8 bits, to be stored in the register specified by the signal *SEL_R*, of 1 bit, as indicated in Table 2, when an upward transition of the clock signal occurs (*clk*).

Table 2 – Writing selection signal in *Registo A* and *B*.

SEL_R	**Register to be written**
0	*Registo A*
1	*Registo B*

In addition to writing, each register **is continuously taking readings**. The exit, *Operando 1*, of 8 bits, will display the value stored in the *Registo A* and the exit, *Operando 2*, of 8 bits, will display the value stored in the *Registo B*.

Note that these are two separate modules.

1.4. Logical and Arithmetic Unit (ALU)

The Arithmetic and Logic Unit of a processor is responsible for performing arithmetic and logical operations. In this project, it is intended that the developed ALU be able to perform the operations presented in Table 3, with the two input signals of 8bits, *Operando 1* and *2* , representing signed integers, where the ALU output signals are determined by the selection signal, *SEL_ALU*, of 3bits. The exit *Resultado*, of 8 bits, will be updated in the case of arithmetic and logical operations (addition, subtraction, AND, OR, NOR, XOR). The ALU also performs comparisons between the two input signals. The exit *E_FLAG*, of 5 bits, will only be updated when a comparison is performed (*SEL_ALU* = 110). Each of your bits indicates the result of each of the five comparisons, as shown in Table 3.

Table 3 – ALU operations.

SEL_ALU	**Operation**			**Affected signal**
000	*Operando1*	+	*Operando2*	*Resultado*
001	*Operando1*	–	*Operando2*	*Resultado*
010	*Operando1*	**AND**	*Operando2*	*Resultado*
011	*Operando1*	**OR**	*Operando2*	*Resultado*
100	*Operando1*	**NOR**	*Operando2*	*Resultado*
101	*Operando1*	**XOR**	*Operando2*	*Resultado*
110	*Operando1*	<	*Operando2*	*E_FLAG(0)*
	Operando1	<=	*Operando2*	*E_FLAG(1)*
	Operando1	=	*Operando2*	*E_FLAG(2)*
	Operando1	>	*Operando2*	*E_FLAG(3)*
	Operando1	>=	*Operando2*	*E_FLAG(4)*

1.5. Register of Flags

The register of *Flags* is intended to store the value of the input signal *E_FLAG*, of 5 bits, when the clock signal (*clk*) is on the rising edge and when the input signal *ESCR_FLAG*, of 1 bit, is at '1'. This register forwards one of the bits saved for the output signal *S_FLAG*, of 1 bit. That bit is determined by the *SEL_FLAG*, of 3 bits, as shown in Table 4.

Table 4 – Output signal of the *multiplexer* comparison based on the selection signal *SEL_FLAG*.

SEL_FLAG	***S_FLAG***
000	*bit* 0 of Registo (<)
001	*bit* 1 of Registo (<=)
010	*bit* 2 of Registo (=)
011	*bit* 3 of Registo (>)
100	*bit* 4 of Registo (>=)

1.6. Program Counter (PC)

The program counter indicates the current position of a program's execution sequence. It sends, on the rising edge of each clock cycle (*clk*), your output *Endereço*, of 8 bits, to the Instruction Memory. The execution sequence will be incremented by one address if the input *ESCR_PC*, of 1 bit, is at ‘0’, otherwise (*ESCR_PC* to ‘1’), the PC output will match the input value *Constante*, of 8 bits, and a jump to the instruction address indicated by this value will occur. The *Reset*, of 1 bit, when activated, allows you to go back to the beginning of the program.

1.7. *Multiplexer* of *Program Counter* (Mux_PC)

The *multiplexer* of the program counter has the purpose of indicating to the PC whether to perform a jump or increment the counter, through the output signal *ESCR_PC*, of 1bit, if it is '1' or '0', respectively. This one depends of the signal *SEL_PC*, of 3 bits, which indicates which of the input values should pass to the output, as indicated in the Table 5.

Table 5 –Output value of the Mux_PC depending on the selection signal *SEL_PC*.

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

1.8. Decoding ROM (ROM)

The decoding of the ROM is responsible for providing the remaining blocks with control signals. This one gets the signal *opcode* of instruction memory, of 5 bits, and puts in its output the values corresponding to the following control signals: *SEL_PC*, of 3 bits, *SEL_FLAG*, of 3 bits, *ESCR_FLAG*, of 1 bit, *SEL_ALU*, of 3 bits, *ESCR_R*, of 1 bit, *SEL_Data*, of 2 bits, *ESCR_P*, of 1 bit, *WR*, of 1 bit. Table 6 shows the relationship between the signal *opcode* and the control signals. It is also indicated in *assembly language*, the corresponding statement. The register *Ri* corresponds to the register indicated by the signal *SEL_R*, of 1 bit (if it is '0', *Ri* corresponds to *Registor A* and if it is '1', *Ri* corresponds to *Registor B*). The acronyms *RA* and *RB* concerns the *Registors A* and *B*, respectively.

Pay attention to the Logic and Arithmetic instructions (except for comparison). In these instructions it is intended that the result be saved in the *Registor A* (there needs to be a write to the register) and therefore the signal *SEL_R* will have to take the value shown in Table 2.

Table 6 –Relationship between the signal *opcode* and the ROM output signals.

Opcode	**Instrução**	**SEL_ALU**	**ESCR_P**	**SEL_Data**	**ESCR_R**	**WR**	**SEL_PC**	**ESCR_FLAG**	**SEL_FLAG**
Peripherals									
00000	LDP *Ri*	XXX	0	01	1	0	011	0	XXX
00001	STP *RA*	XXX	1	XX	0	0	011	0	XXX
Reading and writting									
00010	LD *Ri*, *Constante*	XXX	0	11	1	0	011	0	XXX
00011	LD *Ri*, [*Constante*]	XXX	0	10	1	0	011	0	XXX
00100	ST [*Constante*], *RA*	XXX	0	XX	0	1	011	0	XXX
Logic and Arithmetic									
00101	ADD *RA*, *RB*	000	0	00	1	0	011	0	XXX
00110	SUB *RA*, *RB*	001	0	00	1	0	011	0	XXX
00111	AND *RA*, *RB*	010	0	00	1	0	011	0	XXX
01000	OR *RA*, *RB*	011	0	00	1	0	011	0	XXX
01001	NOR *RA*, *RB*	100	0	00	1	0	011	0	XXX
01010	XOR *RA*, *RB*	101	0	00	1	0	011	0	XXX
01011	CMP *RA*, *RB*	110	0	XX	0	0	011	1	XXX
Jump									
01100	JL *Constante*	XXX	0	XX	0	0	000	0	000
01101	JLE *Constante*	XXX	0	XX	0	0	000	0	001
01110	JE *Constante*	XXX	0	XX	0	0	000	0	010
01111	JG *Constante*	XXX	0	XX	0	0	000	0	011
10000	JGE *Constante*	XXX	0	XX	0	0	000	0	100
10001	JZ *RA*, *Constante*	XXX	0	XX	0	0	001	0	XXX
10010	JN *RA*, *Constante*	XXX	0	XX	0	0	010	0	XXX
10011	JMP *Constante*	XXX	0	XX	0	0	100	0	XXX
Others									
Outros	NOP	XXX	0	XX	0	0	011	0	XXX

Table 7 presents a brief description of each of the instructions presented above.

Table 7 – Description of the processor instructions.

Instructions	Description
LDP *Ri*	write in the register *Ri* copy of the *PIN* value
STP *RA*	Writes to *POUT* a copy of the register *RA*
LD *Ri*, *Constante*	Write in the register *Ri* the value of *Constante*
LD *Ri*, [*Constante*]	Write in the register *Ri* copy of the RAM cell indicated by *Constante*
ST [*Constante*], *RA*	Writes to the RAM cell indicated by *Constante* copy of the register *RA*
ADD *RA*, *RB*	Add the register *RA* with the register *RB* and write the result to the register *RA*
SUB *RA*, *RB*	Subtract from the register *RA* the register *RB* and write the result to the register *RA*
AND *RA*, *RB*	Performs the logical conjugation bit by bit of the register *RA* with the registrar *RB* and write the result to the register *RA*
OR *RA*, *RB*	Performs the logical disjunction bit by bit of the register *RA* with the registrar *RB* and write the result to the register *RA*
NOR *RA*, *RB*	Performs the logical disjunction bit by bit of the register *RA* with the register *RB* and write the result to the register *RA*
XOR *RA*, *RB*	Perform the logical exclusive disjunction bit by bit of the register *RA* with the register *RB* and write the result to the register *RA*
CMP *RA*, *RB*	Compare the register *RA* with the register *RB* and write the result in the *Flag register*
JL *Constante*	Jumps to the instruction indicated by the *Constante* if the value of *Flag Register* referring to the comparison "<" for "1"
JLE *Constante*	Jumps to the instruction indicated by the *Constante* if the value of *Flag Register* referring to the comparison "<=" for "1"
JE *Constante*	Jumps to the instruction indicated by the *Constante* if the value of *Flag Register* referring to the comparison "=" for "1"
JG *Constante*	Jumps to the instruction indicated by the *Constante* if the value of Flag *Register* referring to the comparison ">" for "1"
JGE *Constante*	Jumps to the instruction indicated by the *Constante* if the value of *Flag Register* referring to the comparison ">=" for "1"
JZ *RA*, *Constante*	Jumps to the instruction indicated by the *Constante* if the value of *RA* is "0"
JN *RA*, *Constante*	Jumps to the instruction indicated by the *Constante* if the value of *RA* is negative
JMP *Constante*	Jumps to the instruction indicated by the *Constante*
NOP	Does not perform any operations and just increments the PC counter

1.9. Instruction Memory

The instructions of the program to be executed are stored in the instruction memory. Its size is 14 bits, being addressed by the signal *Endereço*, of 8 bits, and makes available at your exit the *opcode*, of 5 bits, the signal *SEL_R*, of 1 bit, and the signal *Constante*, of 8 bits.

1.10. Data Memory (RAM)

The data memory allows saving the data present in the input signal *Operando 1*, of 8 bits, when the signal *WR* is at '1', on the rising edge of the clock signal (*clk*), in the memory location indicated by the input signal *Constante*, of 8 bits. When the signal *WR* is at '0' a reading is carried out to the memory position indicated by the signal *Constante* and this value is assigned to the output signal *Data_M*, of 8 bits.

2. Simulation e Implementation

The described processor is an electronic component that can be implemented with discrete logic electronic components, in a single chip, or in programmable logic devices. In the processor development phase, it is advantageous to implement the processor architecture in logical devices, because in this phase, sometimes it is necessary to make several changes to optimize the final version of the processor. In this practical work, the development boards SPARTAN 3E, SPARTAN 6 and ARTIX 7 will be used to implement the processor in FPGAs, being necessary to use the clock signal (*clk*). For each of the FPGAs, the peripheral input and output pins are shown in Table 8.

Table 8 - Input and output pins for available FPGAs.

FPGA	Input									
	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	**Output**									
	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. Test

The program language *assembly* must be converted to machine code to program the instruction memory. The program in Table 9 serves to test the operation of the processor. However, they can and should try other programs.

Table 9 –Project testing instructions.

Endereço	Instruction (*assembly*)	Instruction (machine code)
00000000	LD RB, 10	
00000001	LDP RA	
00000010	JN RA, 7	
00000011	CMP RA, RB	
00000100	JG 14	
00000101	LD RA, -1	
00000110	JMP 29	
00000111	LD RB, -1	
00001000	XOR RA, RB	
00001001	LD RB, 1	
00001010	ADD RA, RB	
00001011	LD RB, 14	
00001100	SUB RA, RB	
00001101	JMP 29	
00001110	ST [10], RA	
00001111	LD RA, 0	
00010000	ST [11], RA	
00010001	LD RA, 3	
00010010	ST [5], RA	
00010011	LD RA, [11]	
00010100	LD RB, [10]	
00010101	ADD RA, RB	
00010110	ST [11], RA	
00010111	LD RA, [5]	
00011000	LD RB, 1	
00011001	SUB RA, RB	
00011010	JZ RA, 28	
00011011	JMP 18	
00011100	LD RA, [11]	
00011101	STP RA	
00011110	JMP 30	

4. Evaluation and relevant information

- The project must be carried out individually or in a group of 2 students, representing 15% of the final grade and has a minimum grade of 8 values.

- The report may have a maximum of 5 pages (not counting the appendix, cover and index), containing the description of the work carried out. Its structure should include the following topics:

    1. Introduction;
    2. Objectives;
    3. Development;
    4. Discussion of results;
    5. Conclusion;
    6. Bibliography;
    7. Appendix A: Test Instructions Table (Table 9) completed and the respective simulation (with the input and output signs in decimal and readable);
    8. Appendix B: must contain the code listing of the modules in VHDL.

- The processor should work correctly in the simulator (ISim) and in the FPGA, using its clock signal.

- The report must be delivered to the Student Support Office ("trabalhos@uma.pt"). The copy of the work implies its annulment.

- Discussion of the work is individual, and it is necessary to present how it works in the simulator and in the FPGA (the student should start the discussion with the program running on the computer and a copy of the P1 assessment). If you do not have a personal computer, you can use one of the University's computers, having to notify the professors in advance.

Bibliography

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