DEEC
DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

ADVANCED COMPUTER ARCHITECTURES

Project 1

Abstract: The first lab consists in the development of a simple pipelined processor with a RISC instruction set architecture (ISA), as described in sections 1 and 2. However, two options are given to the students, namely: (i) implementation of the processor in VHDL (see section 3); or (ii) development of a cycle-accurate processor simulator (see section 4). The work must be completed until April 3 (for demonstration in the laboratory) with a written report being submitted via Fenix until April 8 (see section 5).

1 INTRODUCTION

The objective of this work is to design a 5-stage pipelined 32-bit RISC processor, with 32-bit instructions and 16 general-purpose 32-bit registers (although register R0 is always stuck at zero). The processor should be divided into the following main modules:

- **IF (*Instruction Fetch*)** – reading of instruction memory and PC update;
- **ID (*Instruction Decode*)** – instruction decode and operand fetch;
- **EX (*Execute*)** – execution of instruction;
- **MEM (*Memory*)** – read/write of data from/to memory;
- **WB (*Write-Back*)** – writing of the result to *Register File*.

A simplified (illustrative) structure of the processor is shown in Figure 1, in its single-cycle form.

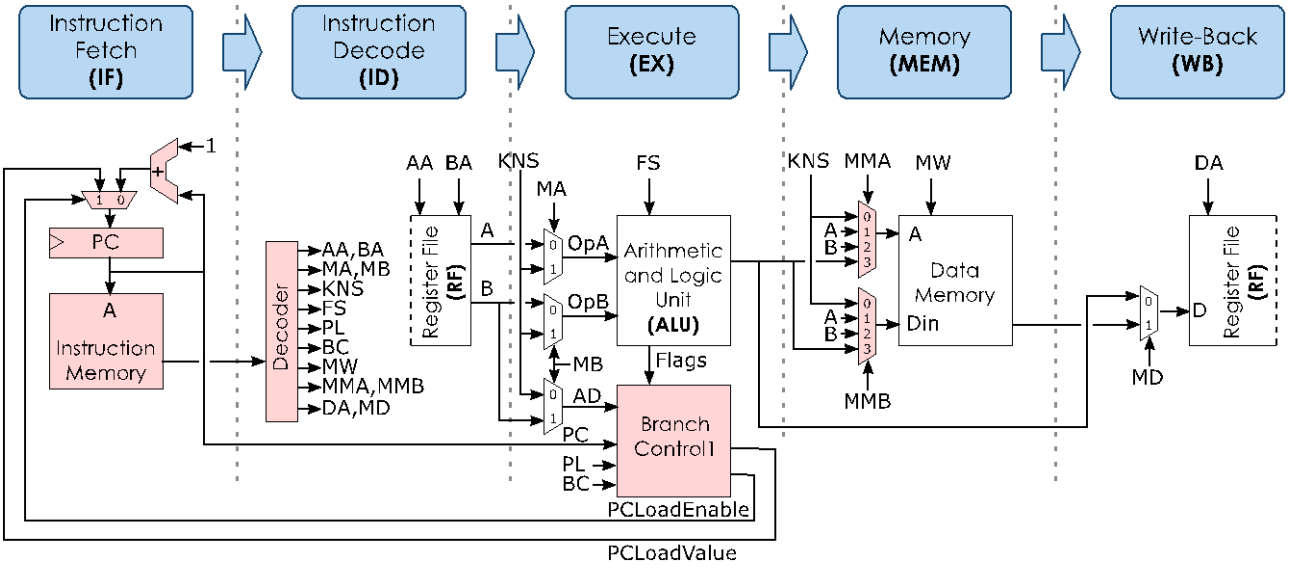

Figure 1 – Simplified diagram of the processor.

2 INSTRUCTION SET ARCHITECTURE

The processor operates on a set of 16 registers (with register R0=0), each storing 32 bit numbers. However, the addressable memory space is limited to 16 bits, each position also holding a 32-bit number.

Table 1 – Set of instructions to support.

Mnemonic	Instruction format (bits)					Description of operation
	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(1)	SA	SB	SIMM18(1)	M[R[SA]+SIMM18] ← R[SB]
B.*cond* **BL.*cond*(3)**	010111	*0,cond*(3) *1,cond*(3)	SA	SB	SIMM14(1)	If (R[SA] *cond* R[SB]) then PC ← PC + SIMM14
BI.*cond* **BIL.*cond*(3)**	011000	*0,cond*(3) *1,cond*(3)	SA	SB	SIMM14(1)	If (R[SA] *cond* SIMM14) then PC ← PC + R[SB]
J.*cond* **JL.*cond*(3)**	011001	*0,cond*(3) *1,cond*(3)	SA	--,UIMM16(2)		If (R[SA] *cond* R0) then PC ← UIMM16
JI.*cond* **JIL.*cond*(3)**	011010	*0,cond*(3) *1,cond*(3)	SA	SB	SIMM14(1)	If (R[SA] *cond* SIMM14) then PC ← R[SB]

(1) The field SIMMx indicates that the x bits extracted from the instruction word correspond to a two's complement number, which should be extended to 32-bits to generate the immediate value.

(2) The field UIMMx indicates that the x bits extracted from the instruction word correspond to an unsigned number, which should be extended to 32-bits to generate the immediate value.

(3) Check Table 2. Link instructions (BL, BIL, JL, JIL) also set R15←PC+1 when the condition is true.

Table 2 – condition codes.

Mnemonic	*cond*	*Condition*	*Description*
.false	000	False	Never jumps (NOP)
.true	001	true	Always jumps (unconditional)
.eq	010	OpA=OpB	Jumps if OpA = OpB
.neq	011	OpA≠OpB	Jumps if OpA ≠ OpB
.gt	100	OpA>OpB	Jumps if OpA > OpB
.ge	101	OpA≥OpB	Jumps if OpA ≥ OpB
.lt	110	OpA<OpB	Jumps if OpA < OpB
.leq	111	OpA≤OpB	Jumps if OpA ≤ OpB

3 OPTION I – LOW-LEVEL ANALYSIS AND IMPLEMENTATION OF A PIPELINED ARCHITECTURE

The objective of this work is to implement (using VHDL) a pipelined version of the previously referred processor. To simplify the development of the work, two main modules are provided to the students: a single cycle version and a pipelined version, which differs from the former by introducing registers between stages. Both versions share the same inner modules, namely IF, ID, EX, MEM and WB.

To the successful accomplishment of this project, the fulfilment of the following steps and objectives are highly recommended. **However, the actual implementation of the schemes for resolution of data and control hazards is not mandatory, except for grades over 17 values.**

3.1 Datapath (week 1)

A. By analysing the VHDL, identify the function of all signals at the output of the ID stage.
B. Explain the operations performed at the ALU in function of the control signal FS.
C. Verify if the architecture directly supports all instructions stated in Table 1 without any changes to the EX, MEM and WB stages. For the unsupported instructions, propose changes to the architecture such as to allow their execution.

3.2 Instruction Decode (week 2)

D. Program the decode stage (file *InstructionDecode.vhd*) to guarantee the correct execution of the instructions. Also make the necessary changes to the Branch Control unit (file *branchcontrol.vhd*) to guarantee the correct execution of all branch and jump instructions.
E. Validate the correct execution of all instructions, by using the single cycle version of the processor.

3.3 Pipelined execution (week 3)

F. By considering the pipelined version of the processor, identify all possible sources data and control hazards. Illustrate the hazards with example Assembly codes.
G. Explain all the changes (including changes to the signals and the addition of new signals) that are required to solve all conflicts. Include all the figures, diagrams and schematics that you feel necessary to deeply describe all the changes. For grades over 17, the implementation of the proposed changes is mandatory.
H. Comment on the impact of such changes to the performance of the processor.

4 OPTION II – DEVELOPMENT OF A CYCLE-ACCURATE PROCESSOR SIMULATOR

The objective of this work is to implement a cycle-accurate processor simulator (although not mandatory, the use of C programming language is recommended). To the successful accomplishment of this project, the fulfilment of the following steps and objectives are highly recommended. **However, the actual implementation of the pipelined execution is not mandatory, except for grades over 17 values.**

4.1 Instruction Fetch and Decode (week 1)

A. Consider that the input program is written in ASCII mode in a text file, where each line corresponds to a different 32-bit memory position. Hence, the first line corresponds to the value on memory address 0, the second line to address 1, etc. Write a program that implements the Instruction fetch stage (i.e., at each simulated clock cycle, it reads one instruction and increments the PC).
B. Implement the instruction decode stage. The decode stage should output three different structures, corresponding to the operation to be performed at the EX, MEM and WB stages. Additionally, it is also

responsible to read the operands from the register file (whenever necessary) and generating the input values to the following stages.

4.2 Execute, Memory and Write-Back (week 2)

C. By considering that the processor is operating at single-cycle mode, implement the Execute, Memory and Write-Back stages and validate the proper execution of all instructions.

4.3 Pipelined Execution (week 3)

D. By considering the pipelined version of the processor, identify all possible sources data and control hazards. Illustrate the hazards with example Assembly codes.

I. Explain all the changes to the code that are required to solve all conflicts. Include all the figures, diagrams and schematics that you feel necessary to deeply describe all the changes. For grades over 17, the implementation of a pipelined execution is mandatory.

5 PROJECT DUE DATE AND REPORT FORMAT

The project should be completed until April 3, for demonstration in the laboratory. An up to 6 page report should be submitted online (via Fenix) until April 8. The report should follow the template for the IEEE Computer society journals:

https://www.ieee.org/publications_standards/publications/authors/author_templates.html