Implementation of a pipelined MIPS processor in VHDL

This laboratory work describes the design of a simplified MIPS pipelined processor. The outcome of this laboratory work will be an implementation of the simplified MIPS pipelined processor in VHDL. As the implementation will be based on the MIPS multi-cycle (sequential) implemented in VHDL in the previous laboratories, the objective of this laboratory work is to highlight the changes that have to be made on the sequential design in order to obtain the pipelined version.

1. Sequential vs pipelined instruction execution

Each instruction is divided into a series of steps:

- Instruction fetch (IF): fetch the instruction from the memory and compute the address of the next instruction.
- Instruction decode (ID): registers indicated by rs and rd are read.
- Execution (Ex): the instruction is known, so the function is executed (memory address computation, arithmetic-logical operation).
- Memory access (Mem): the memory is accessed based on the address computed before, or the result is written in the destination register.
- Write back (WB): the load operation is completed by writing the value from the memory in the register.

Each step of instruction execution is performed in a clock cycle.

For the scope of this laboratory work, only I-type and R-type will be used.

In a sequential CPU, a new instruction is fetched only after the previous has finished its execution. Fig. 1a shows the execution of instructions on a sequential CPU.

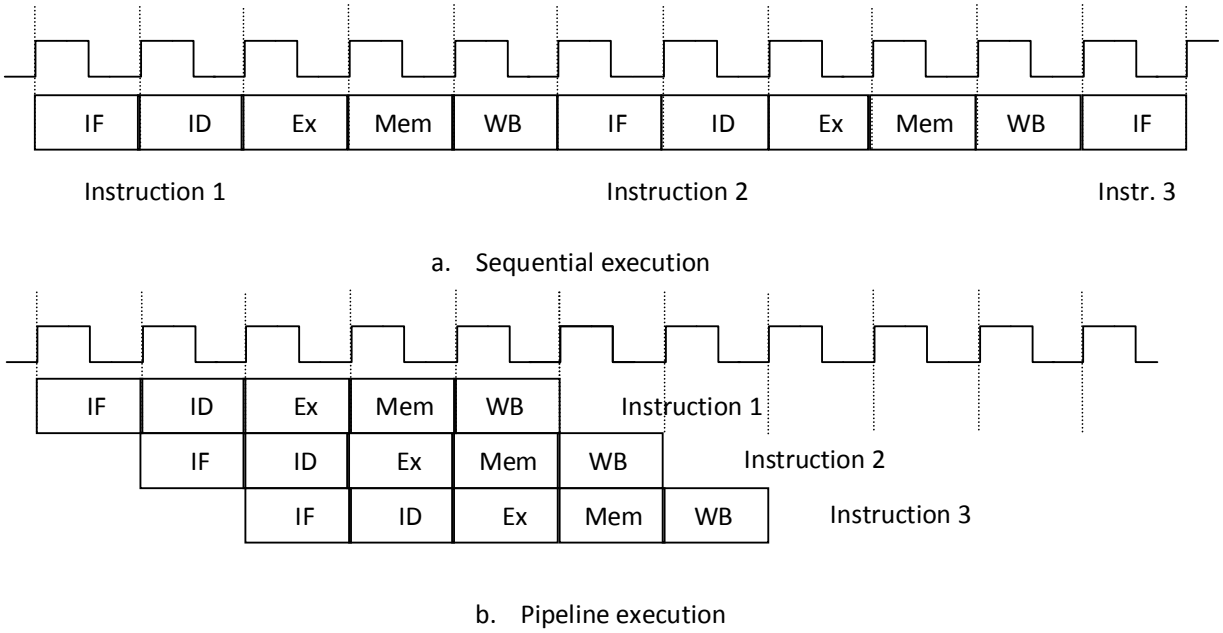

Figure 1. Instruction execution (sequential and pipeline)

A pipeline CPU executes at most five instructions in parallel. At one point, each instruction is in a different step of execution. Fig. 1b shows the execution of instructions on a pipeline CPU. The hardware units that solve the execution of each step have to be well-delimited, as there can't be any hardware used in common between these units. The signals generated at each execution step have to be memorized and forwarded to the next units the instruction will use, as in the next clock cycle the current unit will be used for the execution of another instruction.

2. Pipeline structure

The pipeline is composed of hardware units, one for each instruction execution step. These are called pipeline stages. Two consecutive pipeline stages are connected by a pipeline register. A pipeline stage receives inputs from the pipeline register placed before it and computes the outputs, which are written in the next pipeline register (see Fig.2).

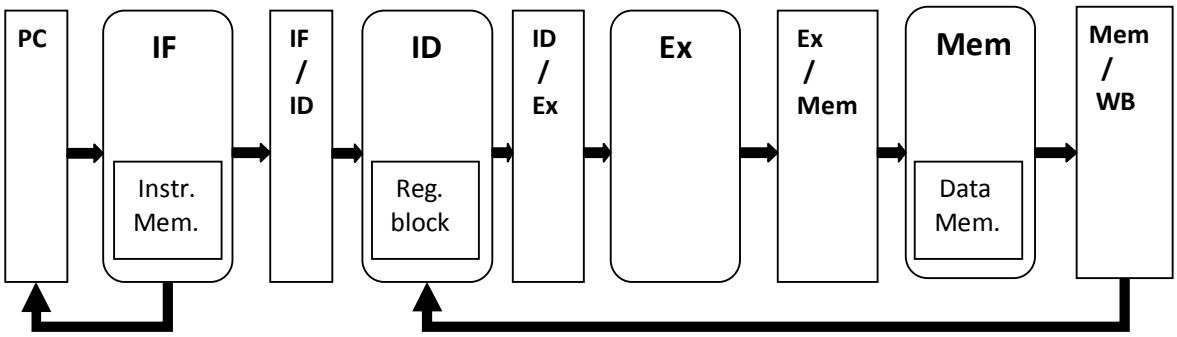

Figure 2. Pipeline structure

As the instruction fetch stage and the memory access stage of the pipeline have to be independent, there is need for separate Instruction Memory and Data Memory.

The pipeline registers hold data only for a clock cycle. Data that are needed for the subsequent stages are forwarded to the next pipeline register.

Table 1 shows the data fields held by each pipeline register.

Table 1. Data fields in the pipeline registers

Pipeline register	Data fileds
IF/ID	Instruction and PC
ID/Ex	PC, Reg[rs], Reg[rt], sign extended offset, rd, rt
Ex/Mem	Zero, ALU result, Reg[rt], Destination register address (rt or rd)
Mem/WB	ALU Result, Data from memory, Destination register address

There are two options for the pipeline control:

- Determine control signal in ID stage and pass the needed signals along the pipeline
- Pass instruction code parts along the pipeline and determine control signals inside each stage

The first option is used for most of real systems.

A general description of the MIPS pipeline datapath is presented in Fig. 3.

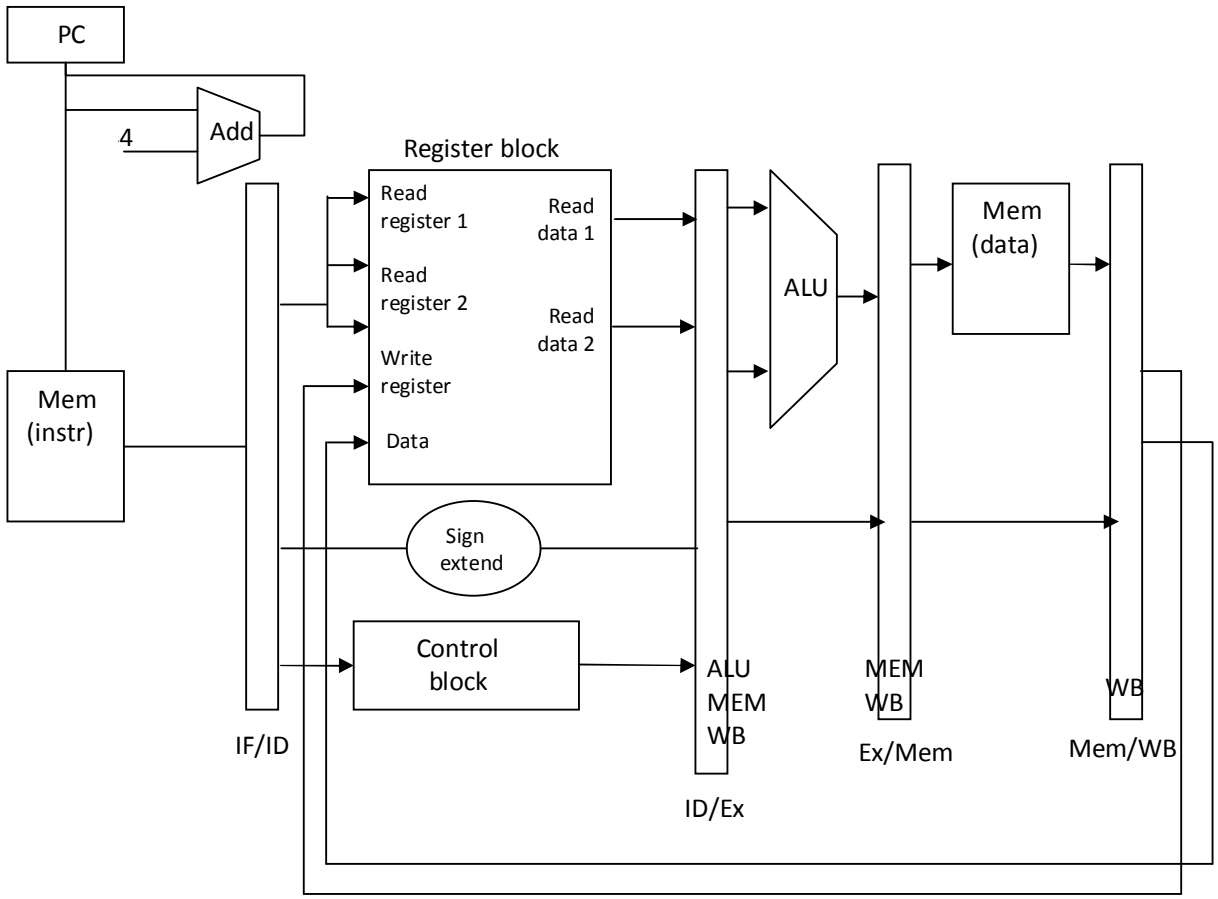

Figure 3. MIPS pipeline datapath

3. Exercises

1. Implement the MIPS pipeline and test it by simulation.
2. Detect a RAW data hazard on the pipeline MIPS.

Bibliography

[1] SCS lecture notes