Politecnico di Torino

III Facoltà di Ingegneria

Laboratory 3
RISCV Processor

Master degree in Electrical Engineering

Authors: Group 21

Dilillo Nicola S284963
Moncalvo Stefano S290315
Carrano Lorenzo S281565

February 15, 2022

Contents

CHAPTER 1

Introduction

1.1 Introduction

CHAPTER 2

Datapath

2.1 Fetch Stage

During fetch stage, an instruction is read by the instruction register accordingly to the content of the Program Counter, a 64-bit register that store the address in memory of next instruction to be fetched. Fetched instruction is then passed to the next stage. Assuming the case of the execution of concurrent instructions, from one instruction to the next one the program counter is incremented by 4. This is because each instruction is encoded on 4 bytes and the instruction memory is byte-addressed, thus we have an alignment of 4 bytes. In case of branch instructions or jumps, the value of the program counter can be modified in other ways using various offsets and rules depending on the case, as it will discussed later.

2.2 Decode Stage

In Decode stage the instruction that has been previously fetched is decoded by means of the bit-fields that compose it. Operands are decoded, be them a content of a register or an immediate value, and the control bits for the other components eventually employed in requested calculation (e.g. multiplexers or the ALU) are prepared to be sent to next stage. Decode stage is crucial since some hazard can be detected in this stage only, by comparing the output of decoding with previous ones (currently in next stages), as in the case of data hazards and forwarding mechanism, as it is discussed in the dedicated section.

2.2.1 Register File

The register file stores internal runtime values used during calculations. Some registers have very special purposes, such as the stack pointer, the thread pointer, the return address or the register r0, the only one that can't be overwritten and that always stores a value of all zeros, others can be used during program execution, and are said to be **general purpose registers**. The register file is synchronous with the clock signal and it is composed by a total of 32 registers, indexed by means of 5 address lines, each storing a word of 4 bytes. It has been decided to keep the register file always able to provide data in reading, without any enable signal, while a dedicated signal is needed to allow writing operations. An active-high reset signal is present, in order to clear the entire content of the registers.

2.3 Execute Stage

During execute stage, operands are sent to the ALU in order to perform the required computation. In case of a Data Hazard, a forwarding mechanism has been implemented.

2.3.1 ALU

The Arithmetic Logic Unit is the computational core of the overall architecure. Since we chose to support 32 bits of parallelism for registers and data management, the ALU has been built with a parallelism of 32 bits too. The Arithmetic Logic Unit has been developed by means of standard arithmetic operations implemented by VHDL language in order to let to the compiler a certain freedom during optimization. To better manage constants, sizes and opcodes, all meaningful values have been delcared in the file *myTypes.vhd*. The ALU is basically implemented using a process, in which inputs are employed into the proper computation by means of a control signal on 3 bits. Since the control unit is an hardwired one, this control signal is directly derived from original opcode of the decoded instruction. The inputs of the ALU have been multiplexed using two 3-to-1 multiplexers, in order to correctly manage Data Hazards by eventually perform a forwarding.

2.3.2 Forwarding Unit

As mentioned in previous section, the Forwarding Unit is a component with a very special purpose: eventually forward a previously computed result to one of the ALU inputs before it is effectively wrote in memory, in order to resolve a possible Data Hazard, avoiding a stall. The need to forward a certain value in one of the stages following the Execution one is detected by means of comparison between the encodings of actual source registers that are passed from Decode stage to the Execute Stage, and the encodings that are actually kept in Memory and Write-back stages. If an equivalence is found, this means that the instruction that has been decoded in previous clock cycle is asking as operand a result that is not actually available in register file, but that it has already been computed, thus it is possible to directly wire it to the proper ALU input via additional interconnections and a couple of multiplexer, as shown in figure.

2.4 Memory Stage

Memory stage is the step of the pipeline in which results are loaded/stored in main memory. Main memory is byte addressed and characterized to be Little Endian, i.e. the least significant byte is stored at a least address than the most significant one. Main memory is involved in load and store operations, that directly invoke it. It has a crucial role during when composite data need to be stored to guarantee the program correctness. Since RISCV does not require stored words to be aligned in memory, their management is simpler and requires less hardware resources.

2.5 Write Back Stage

During Write-back stage, the register file is accessed in order to store the result of the previously performed calculation inside the destination register previously determined during the past decoding phase of the instruction that has generated this value. As previously clarified, the register file needs in this case that the relative write-enable signal is set in order to be able to accept a new value inside one of its locations, and it is in charge of the control unit to properly drive that enable signal.

CHAPTER 3

Control Unit

3.1 Introduction

CHAPTER 4

Hazards