Politecnico di Torino

III Facoltà di Ingegneria

Exercises and Homeworks for the course Integrated Systems Architecture

Master degree in Electrical Engineering

Authors: e02GQCxx

Paperoga, Gilberto de Pippis, Brigitta, Filo Sganga

February 20, 2021

Many thanks to Prof. Mariagrazia Graziano for providing us with this template.

Contents

CHAPTER 1

Datapath

We have implemented the VHDL architecture of the processor(Figure 1.1), splitted in 5 pipeline stages: Fetch, Decode, Execute, Mem, WriteBack.
Instructions and datas of the entire procssor are represented on 32 bits. As specification says, Instruc-

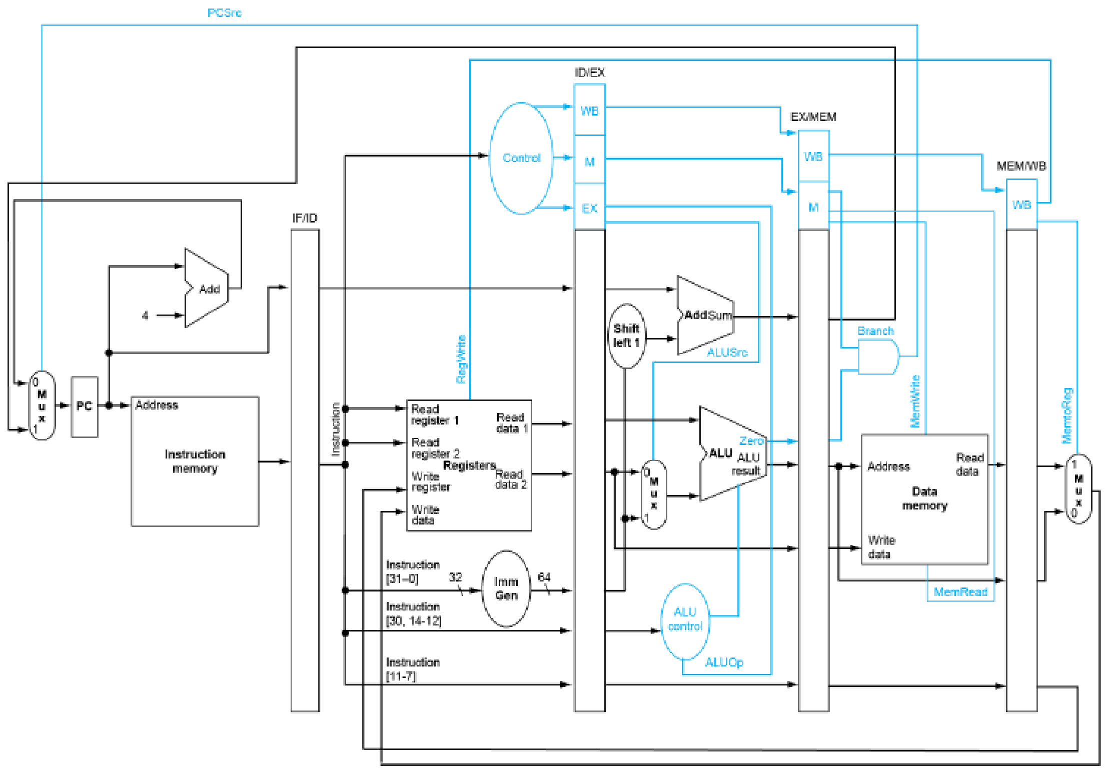

Figure 1.1: General structure

tion and Data memory was not included. We have implemented it in the testbench.
Here we have tried to impement most of the single blocks in a Structural way.
Only Register file, alu and all filpflops are implemented as Behavioral.

On Figure 1.2 it is showed how we have organized the VHDL files. Now more details on singolar block implementation.

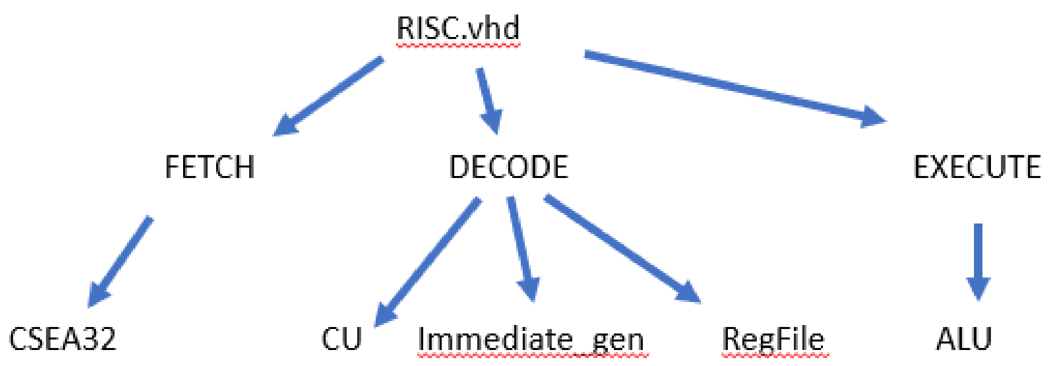

Figure 1.2: General VHDL structure

1.1 Fetch

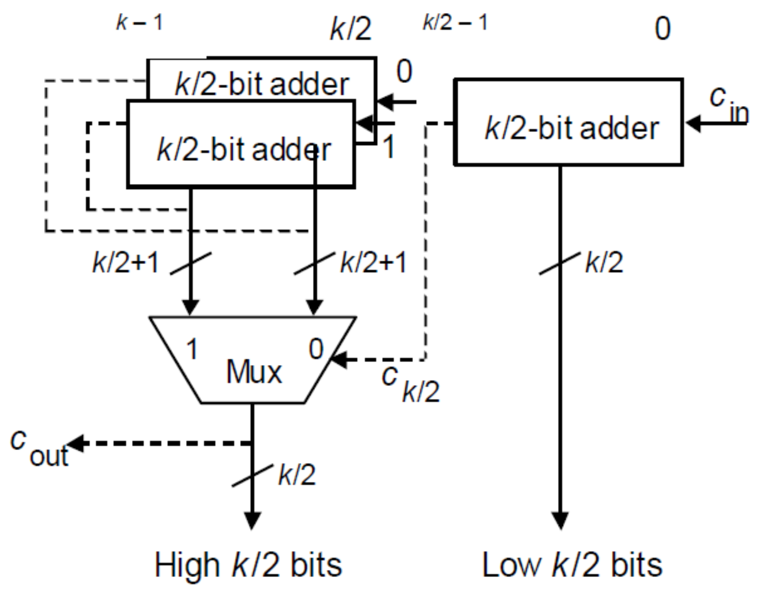

Figure 1.3: CSEA structure

To update the PC we have implemented a Carry Select Adder.
Each singolar blocks are compose of a 8 bit Ripple carry adder. In this way we have reduced the delay.

1.2 Decode

The register file is behavioral. It's composed of 32 registers on 32 bits.
At each rising edge of the clock, read two value given the addresses Register_read1 and 2.
If the write enable is 1, we write in the register location. It have this port:

- Write enable port;
- Register read1;
- Register read2;
- Write register;
- Write data;
- Read Data1;
- Read Data1;
- Clock;
- Reset;

The Immediate_generate block read the instruction and generate the Immediate value. So based

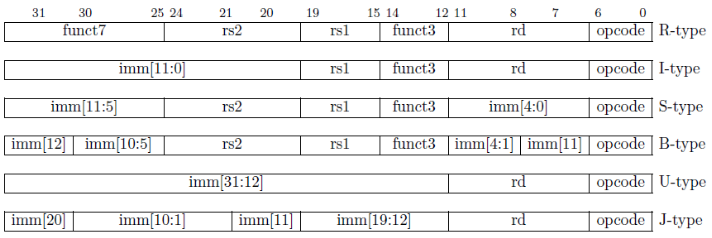
31 30 25 24 21 20 19 15 14 12 11 8 7 6 0
funct7 | rs2 | rs1 | funct3 | rd | opcode | R-type
imm[11:0] | rs1 | funct3 | rd | opcode | I-type
imm[11:5] | rs2 | rs1 | funct3 | imm[4:0] | opcode | S-type
imm[12] | imm[10:5] | rs2 | rs1 | funct3 | imm[4:1] | imm[11] | opcode | B-type
imm[31:12] | rd | opcode | U-type
imm[20] | imm[10:1] | imm[11] | imm[19:12] | rd | opcode | J-type

Figure 1.4: Instruction organization

on which instruction we are decoding, organize a 32 bits Immediate value.

1.3 Execute

The adder that create the new program counter is again implemented with a 32 bits CSEA. Then we have the Alu that receive two inputs value and the Opcode specify which instruction implement.

1.4 Memory

This module is implemented in the RISC top entity. Contain only the AND port that generate the selection signal for the multiplexer on the fetch phase.

1.5 Write back

This module is implemented in the RISC top entity. Contain a multiplexer of 4 inputs and decide between:

- Data memory read;
- Alu result;
- PC + 4 taken from fetch unit;

- PC where we need to jump;

The last two inputs are considered for the instruction auipc and jal. For the JAL stores the address of the instruction following the jump (pc+4) into register rd.
So we take the updated PC in the fetch phase and we bring it in the Write Back phase.
AUIPC forms a 32-bit offset from the 20-bit U-immediate, adds this offset to the address of the AUIPC instruction.
Then places the result in register rd. We take the result in the Execute phase, where with the CSEA we create the branch PC.