PA - Processor Architecture

Course 2022/23

Processor project

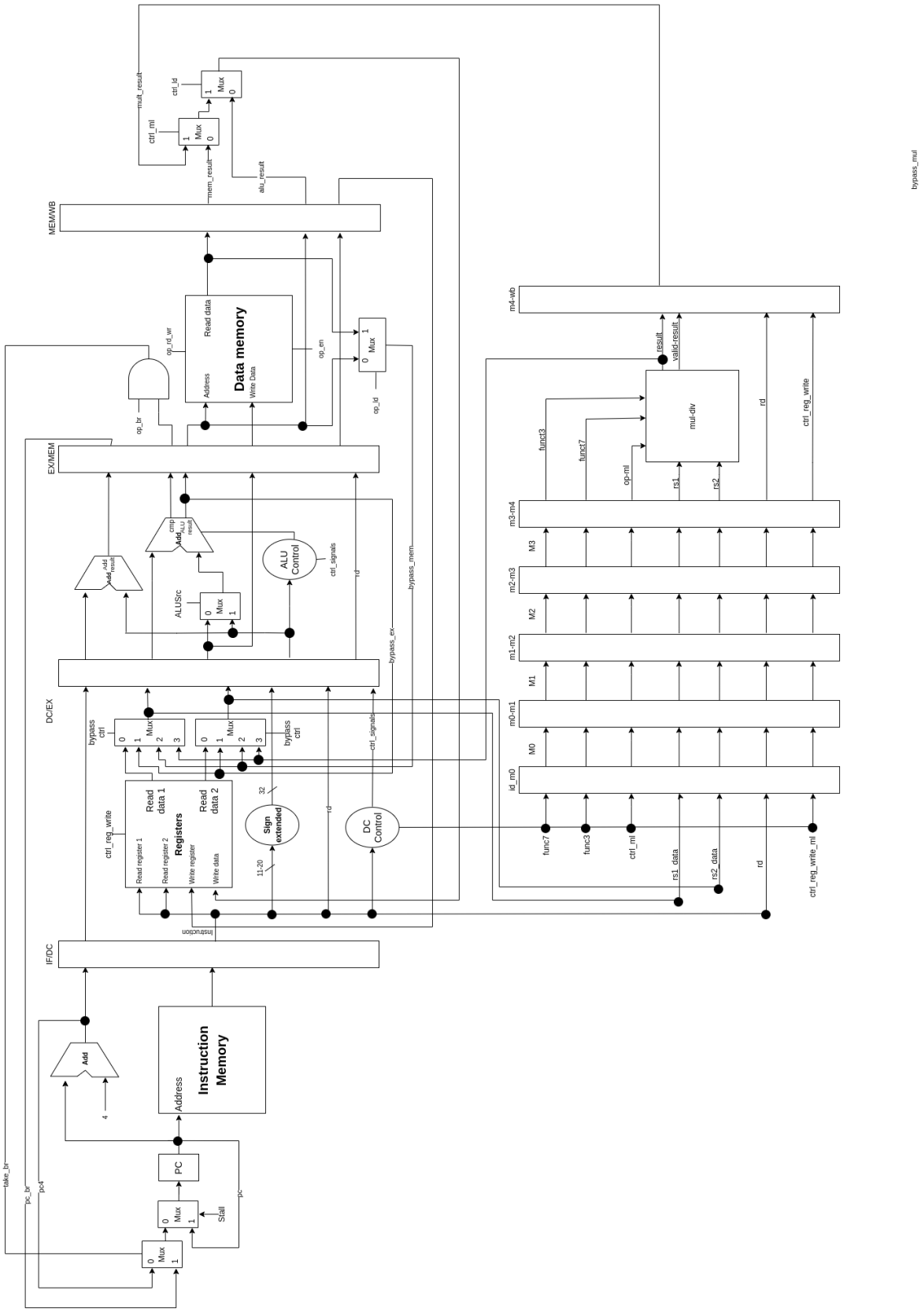
Andrea Querol

January 24, 2023

Contents

1 Block diagram

The RTL is RISC-V based.



2 Instructions

The ISA is RISC-V.

2.1 Load instructions

The load instructions implemented are: `LW, LH, LB`.

2.1.1 Fetch

Loads do not affect the usual behaviour of the Fetch stage, which is adding 4 to the `pc`, unless a dependency is found in the Decode stage. In that case, the fetch is stalled.

2.1.2 Decode

Loads are encoded in the I-type format, so the following signals are extracted from the instruction: `opcode, func3` (width) `, rs1, imm_se` and `rd`.

Moreover, the control signals, which indicate the instruction type, are set to 0 except the `ctrl_ld`. That signal shows that the instruction is a load. Also, the signal `ctrl_reg_write` is set to 1.

`rs1` and `rs2` dependencies are checked agains older instructions in the pipeline. Both registers are compared agains the `rd` register from the instructions which are currently in the Execution, Memory and M4 stage. In case some register matches Execution `rd` register, the result from the alu is bypassed to the corresponding source register. The result bypassed from the Memory stage depends on the instruction type that is currently in that stage, if it is a load, then the read memory access data is bypassed, otherwise, the result from the ALU is bypassed. If no register `rd` match, Fetch and Decode are stalled, waiting for the dependency to be solved.

2.1.3 Execution (ALU)

In this stage, the load address is computed by adding register `rs1` and the sign-extended immediate.

2.1.4 Memory

The data memory is accessed performing a read to the address computed in the previous stage. The result sent to the Decode stage is the one from the memory access.

2.1.5 Write back

Register `rd` from the Memory stage and the data from the memory access are selected to be written into the Register File.

2.2 Store instructions

The store instructions implemented are: `SW, SH, SB`.

2.2.1 Fetch

Stores do not affect the usual behaviour of the Fetch stage, which is adding 4 to the `pc`, unless a dependency is found in the Decode stage. In that case, the fetch is stalled.

2.2.2 Decode

Stores are encoded in the S-type format. Therefore, signals extracted from the instruction are: `opcode, func3` (width) `, rs1, rs2` and `imm_se`.

Moreover, the control type signals, which indicate the instruction type, are set to 0 except the `ctrl_st`. That signal shows that the instruction is a load. Also, the signal `ctrl_reg_write` is set to 0.

`rs1` and `rs2` dependencies are checked agains older instructions in the pipeline. The dependency behaviour is the same as in subsubsection 2.1.2 (Decode in Load instructions).

2.2.3 Execution

The store address is computed by adding register `rs1` and the sign-extended immediate.

2.2.4 Memory

The data memory is accessed performing a write to the address computed in the previous stage. The result sent to the Decode stage is the one from the Execution stage.

2.2.5 Write back

No data has to be written into the Register File.

2.3 Branch instructions

The implemented instructions are: `BEQ, BNE, BLT, BGE, BLTU, BGEU`.

2.3.1 Fetch

The Fetch stage receives the decision taken if the Memory stage, which indicated if the branch has to be taken or not. It also receives the `pc` address to which branch. In case, the branch is taken, the `pc` is updated with the new one received. If a dependency is found during the Decode stage, the Fetch stage is stalled.

2.3.2 Decode

Branch instructions are encoded in the B-type format. Therefore, signals extracted from the instruction are: `opcode, func3, rs1, rs2` and `imm_se`.

The control type signal set to 1 is `ctrl_br`, the other ones are set to 0. Moreover, the signal `ctrl_reg_write` is set to 0.

`rs1` and `rs2` dependencies are checked agains older instructions in the pipeline. The dependency behaviour is the same as in subsubsection 2.1.2 (Decode in Load instructions).

If a branch is taken, the current instruction in the Decode stage is killed.

2.3.3 Execution

The condition of the branch is evaluated depending on the `func3` value. In other words, the registers are compared according to the instruction condition. In parallel, the `pc` address from the branch is calculated.

If a branch is taken, the current instruction in the Execution stage is killed.

2.3.4 Memory

To check if the branch has to be taken, the signal `ctrl_br` and the signal with the comparison result are checked. If both signals are HIGH, then the branch is taken.

2.3.5 Write back

No data has to be written into the Register File.

2.4 Arithmetic instructions

The arithmetic instructions implemented are: `ADD, ADDI, SUB`.

2.4.1 Fetch

Arithmetic instructions do not affect the usual behaviour of the Fetch stage, which is adding 4 to the `pc`, unless a dependency is found in the Decode stage. In that case, the fetch is stalled.

2.4.2 Decode

Arithmetic instructions are encoded in the R-type format for `ADD` and `sub`, and in I-type format for `ADDI`.

In case of the R-type instructions, the signals extracted from the instruction are: `opcode, funct7, func3, rs1, rs2` and `rd`. The only control type signal set to 1 is `ctrl_op`, which shows that is an R-type arithmetic instruction.

For the I-type instructions, the following signals are extracted from the instruction: `opcode, funct7, func3, rs1, rd, imm_se`. Therefore, the control type signal set to 1 is `ctrl_im`, to indicate an I-type arithmetic instruction.

In both cases, the signal `ctrl_reg_write` is set to 1.

`rs1` and `rs2` dependencies are checked agains older instructions in the pipeline. The dependency behaviour is the same as in subsubsection 2.1.2 (Decode in Load instructions).

2.4.3 Execution

The corresponding arithmetic operation is performed: a register-register addition or substaction, or the register-immediate addition.

2.4.4 Memory

The result from the ALU is pipelined to the next stage and bypassed if necessary to Decode.

The memory is not accessed, no operation is enabled to access memory.

2.4.5 Write back

Register `rd` from the Memory stage and the data from the ALU are selected to be written into the Register File.

2.5 M RISC-V extension instructions

The RISC-V M extension has been followed. The arithmetic instructions implemented are: `MUL, MULH, MULHSU, MULHU, DIV, DIVU, REM, REMU`.

2.5.1 Fetch

M-extension instructions do not affect the usual behaviour of the Fetch stage, which is adding 4 to the `pc`, unless a dependency is found in the Decode stage. In that case, the fetch is stalled.

2.5.2 Decode

M-extension instructions are encoded in the R-type format. The signals extracted from the instruction are: `opcode, funct7, func3, rs1, rs2` and `rd`. The control type signals, which indicate the instruction type, are set to 0 except the `ctrl_ml`. That signal shows that the instruction is a multiplication, division or remainder. The signal `ctrl_reg_write_ml` is set to 1, and `ctrl_reg_write` to 0.

`rs1` and `rs2` dependencies are checked agains older instructions in the pipeline. The dependency behaviour is the same as in subsubsection 2.1.2 (Decode in Load instructions). However, when a M-extension instruction is being executed in its exclusive stages and a non M-extension instruction, such as `ADD`, is decoded, there is a problem. The execution time of M-extension instructions is longer than the other ones, so a shorter younger intruction can write its `rd` result in the Register File before the M-extension older one. Or can even try to write to the Register File at the same time, which is not possible in my implementation. To solve that problem, a conservative approach has been considered: stall the shorter younger instruction in Decode stage until the M-instruction(s) graduate.

2.5.3 Execution and Memory

Those stages are not used by M-extension instructions.

2.5.4 M-extension stages

There are 5 M-extension-exclusive stages: M0, M1, M2, M3, M4, M5. The necessary signals are pipelined from M0 to M5. Therefore, there can be 5 non-dependent multiplications in flight at the same time.

M5 stage is where the result is being computed, depending on the operation required. `MUL` instruction calculates the multiplication of `rs1` by `rs2` and takes the lower 32 bits. `MULH[[S]U]` instructions perform the signed and/or unsigned multiplication, but gather the upper 32 bits. `DIV[U]` instructions compute the signed/unsigned division, and `REM[U]` the remainder.

2.5.5 Write back

In this stage, the multiplication result is sent to the Register File.

2.6 Exception instructions

There are no exception implemented.

3 Performance results