Five-Stage MIPS Pipeline CPU

Zhuolin YANG

June 25, 2017

Contents

1 Introductory

This report is a brief summary of my final course work of "Computer System". In this course, I implemented a CPU with five-stage MIPS pipeline which can execute some basic MIPS instructions and supports forwarding in order to reduce the data hazards.
This project is constructed in Verilog HDL.

2 Main Modules

2.1 Five stages

In the standard five-stage MIPS pipeline, there are five basic pipeline stages:

- Instruction Fetch(IF)
- Instruction Decode(ID)
- Execution(EX)
- Memory access(MEM)
- Write Back(WB)

For the consideration of convenient and regularity, all of those five stages were implemented as several modules.

2.1.1 Instruction Fetch

This module is used for controlling the program counter and sending it to the ROM to fetch the current instruction, which contains several input or output channels:
Input:

- clk: current clock signal
- rst: current reset signal
- stall : current stall signal
- pc_we : program counter write enable
- pc_write_instr

Output:

- pc_read_instr(output)

This module is implemented as if.v.

2.1.2 Instruction Decode

This module is used for decoding the instructions and obtaining the operators and the operators' type, operands and the target register of each instructions. This module also contains several inputs and outputs.
Input:

- rst: reset signal
- instr_i: input instructions
- reg1_read_instr: input data of the first operand
- reg2_read_instr: input data of the second operand
- pc_i: input data of program counter
- ex_op: the possible instruction type of the instructions forwarded by EX stage
- ex_we: the write enable of the instructions forwarded by EX stage
- ex_write_addr: the write address of the instructions forwarded by EX stage
- ex_write_instr: the write instruction of the instructions forwarded by EX stage
- mem_we: the write enable of the instructions forwarded by MEM stage
- mem_write_addr: the write address of the instructions forwarded by MEM stage
- mem_write_instr: the write instruction of the instructions forwarded by MEM stage

Output:

- instr_o: output instructions
- op: decoded operator,
- type: decoded operator type,
- reg1, reg2: two operands
- reg1_re, reg2_re: the Read Enable of two operands
- reg1_read_addr, reg2_read_addr: the Read Address of two operands
- we: Write enable signal
- write_addr: write address
- write_instr: write instruction data
- pc_we: program counter Write enable

- pc_write_instr: program counter write instruction data
- stallsignal

In order to reduce the data hazards, This module managed to fetch the latest data forwarding from the stage EX and the stage MEM. After doing this, in order to avoid the only rest data hazard, we can use the stallsignal to stall the pipeline.
This module is implemented in id.v.

2.1.3 Execution

This module is used for calculating the result of the instructions received from the stage ID.
Input:

- rst: reset signal
- instr_i: the input instruction
- op_i: the input operator
- type: the input operator type
- reg1_i, reg2_i: the input of two operands.
- we_i: the input of Write enable signal
- write_address_i: the input write address
- write_instr_i: the input write data

Output:

- instr_o: the output instruction
- op_o: the output operator
- reg1_o, reg2_o: the output two operands
- we_o: the output of Write enable signal
- write_address_i: the output write address
- write_instr_i: the output write data
- stallsignal

In fact the stallsignal is not necessary, since all kinds of calculation can be finished in a single cycle.
This module is implemented in ex.v.

2.1.4 Memory Access

This module is used for calculating the memory address and sending to the RAM to execute the load or store instructions.
Input:

- rst: reset signal
- instr: the input instructions
- op: the input operator
- reg1, reg2: the input operands
- mem_i: the input read data
- we_i: the input Write enable signal
- write_addr_i: the input write address
- write_instr_i: the input write data

Output:

- mem_re: Memory Read enable
- mem_read_addr: Memory Read address
- mem_we: Memory Write enable
- mem_write_addr: Memory Write address
- mem_write: Memory Write position mark
- mem_write_instr: Memory write data
- we_o: the output Write enable signal
- write_addr_o: the output write address
- write_instr_o: the output write data

This module is implemented in mem.v.

2.1.5 Write Back

This module is not useful in current CPU, since there's no need of register HI and LO by execute current instructions.

2.2 Latch Modules

In the five-stage MIPS pipeline, there are totally four pipeline latches which used as the buffers between adjacent pipeline stages.
These modules are implemented in:

- `if_id.v`
- `id_ex.v`
- `ex_mem.v`
- `mem_wb.v`

In fact, All of the pipeline latches transferred the data on the posedge of the clock signal.

2.2.1 ROM

Since the instructions cannot be modified at runtime, ROM uses some inputs to read the program counter and outputs the corresponding instruction.
This module is implemented in `rom.v`.

2.2.2 RAM

RAM can do memory read and memory write to read the address and outputs the corresponding data and write data.
This module is implemented in `ram.v`.

2.2.3 Register File

The register file use some inputs to get the value of corresponding register or writing a value to register.
This module is implemented in `register.v`.

2.3 Control Module

Control module is used for receiving the stall request from the pipeline and send the stall signal to the pipeline latches.
This module is implemented in `ctrl.v`.

3 Conclusion

From this project, I learned:

- A new programming language, Verilog HDL.
- A better understanding of computer architecture.

  I learned a lot by from a textbook called "How to finish your own CPU", its impressive structure helps me a lot.

Due to the short time and not quite high ability, it's not easy to fulfil a CPU full of my ideas, but just a basic CPU I learned from textbook. But I will try to improve my skills in next semester.