香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

CSC3050: Computer Architecture

Five-stage Pipelined MIPS CPU

Jiawei Lian

119030043

119030043@link.cuhk.edu.cn

The Chinese University of Hong Kong, Shenzhen

School of Data Science (SDS),
Computer Science and Engineering

May, 15, 2021

Contents

1 Big Picture Thoughts about the Five-stage Pipelined CPU

The idea of pipe-lining is to speedup the overall execution time via increasing throughputs. The throughput is increased via partition the execution of one instruction into five stages. They are:

1. IF: instruction fetch,
2. ID: instruction decode and register file read
3. EX: execution or address calculation
4. MEM: data memory access
5. WB: write back.

Each stage utilizes different resources. Thus, these five stages can be executed in one clock. The table below illustrate the process:

Table 1: An illustration of the five-stage pipelined CPU

	CLK_7	CLK_6	CLK_5	CLK_4	CLK_3	CLK_2	CLK_1
Instruction 7	IF_7						
Instruction 6	ID_6	IF_6					
Instruction 5	EX_5	ID_5	IF_5				
Instruction 4	MEM_4	EX_4	ID_4	IF_4			
Instruction 3	WB_3	MEM_3	EX_3	ID_3	IF_3		
Instruction 2		WB_2	MEM_2	EX_2	ID_2	IF_2	
Instruction 1			WB_1	MEM_1	EX_1	ID_1	IF_1

Although the five-stage pipelined CPU can make execution more efficient, it will cause hazards, which is the result of conflicts for use of a resource. This CPU faces:

1. RAW (read after write) Data hazards. R-type instruction is solved by forwarding. Load word instruction is solved by stall.
2. Control hazards: solved by stall.

2 The CPU Design

See Figure 1.

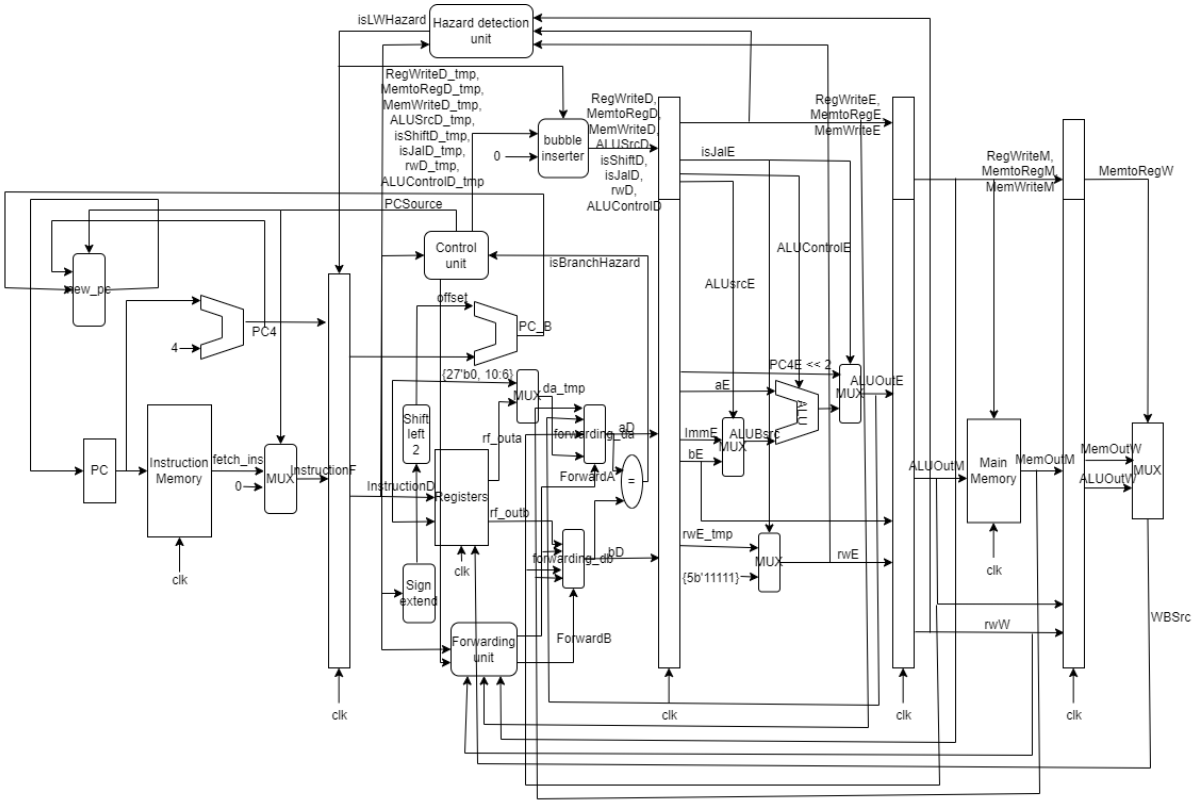

Figure 1: The CPU Design

3 High Level Implementation Ideas

The high level implementation idea is to break the CPU into 9 modules, including 5 modules for each execution stage, and 4 modules for the buffer between every two stages. The structure of the project is:

1. 00.cpu_.v: the pipelined cpu.
2. 01.if_.v: the instruction fetch stage.
3. 01a.InstructionRAM.v:the RAM for instructions.
4. 02.reg_if_id_.v: the buffer between IF and ID.
5. 03.id_.v: the instruction decode stage.
6. 03a.instruction_docoder_.v: the instruction decoder in ID stage. It inputs the instruction and outputs the instruction type.
7. 03b.control_unit_.v: the control unit inputs the instruction type and output the corresponding signals.
8. 03c.registers_.v: the register file in the ID stage.
9. 04.reg_id_ex_.v: the buffer between ID and EX.
10. 05.ex_.v: the execution stage.

11. 05a.alu_.v: the Arithmetic Logic Unit.
12. 06.reg_ex_mem_.v: the buffer between EX and MEM.
13. 07.mem_.v: the data memory access stage.
14. 07a.MainMemory.v: the main memory.
15. 08.reg_mem_wb_.v: the buffer between MEM and WB.
16. X1.mux4_32_.v: a 4 to 1 MUX.
17. X2.bubble_inserter_.v: insert a bubble in ID when J or Branch hazard occurs.
18. X3.forwarding_unit_.v: a forwarding unit to solve R-type data hazards.

4 Implementation Details

This section will briefly talk about how I counter hazards.

4.1 Data hazards

Use the forward unit to indicate which kind of hazard occurs. The forward unit contains ForwardA[1:0] and ForwardB[1:0]. ForwardA indicates the register to write conflicts with rs, while ForwardB indicates the register to write conflicts with rt.

1. R-type EX/MEM: ForwardA = 01 or ForwardB = 01
   - **RegWriteE** is True, meaning the instruction at EX stage will write into the register files.
   - **MemtoRegE** is False, meaning the instruction at EX stage is not LW.
   - **rwE = rs** or **rwE = rt**
   - **rwE** $\neq$ **0**, meaning that the destination register is not $zero
2. R-type MEM/WB: ForwardA = 10 or ForwardB = 10
   - **RegWriteM** is True, meaning the instruction at MEM stage will write into the register files.
   - **MemtoRegM** is False, meaning the instruction at MEM stage is not LW.
   - **rwM = rs** or **rwM = rt**
   - **rwE** $\neq$ **rs** or **rwE** $\neq$ **rt** respectively, meaning that EX/MEM does not happen.
   - **rwM** $\neq$ **0**
3. LW: ForwardA = 11 or ForwardB = 11
   - MemtoRegM is True, meaning that it is a LW instruction (The result from main memory will be written into the register file).
   - **rwM = rs** or **rwM = rt**
   - **rwM** $\neq$ **0**

4.2 Control hazards

The control unit detect control hazards and use PCSource[1:0] to indicate PC from which resources (PC+4, PC_JUMP, PC_BRANCH, RegisterA) to be chosen from.

1. Branch: the register file is read at the negative edge of the clk. The branch hazard is detected at ID, as long as the register file gives the value of the registers required. Thus, before the next positive edge of clk comes, the instruction can be changed, which means no need to speed another clk time to flush.
2. Jump: the bubble inserter will insert a bubble into ID via setting signals in ID to 0, and letting PC in IF remains unchanged.