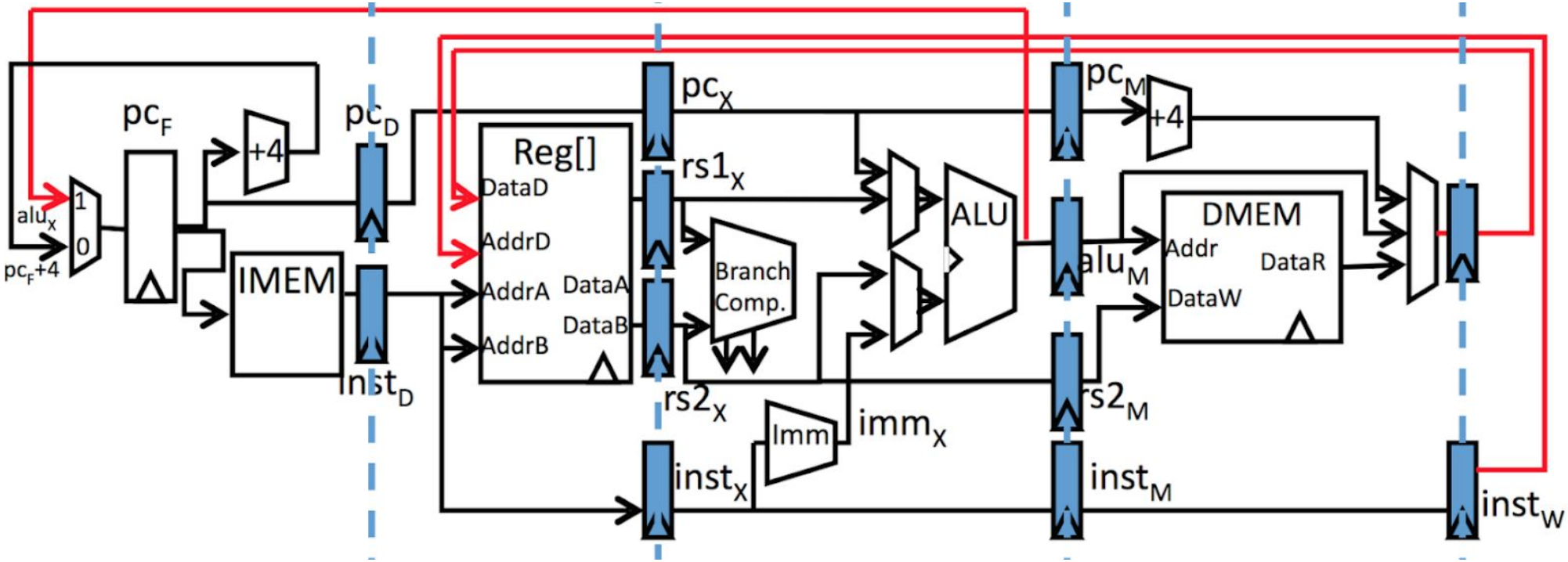
5-Stage RISC-V RV32I Datapath



Pipelined CPU Design

Now, we will optimize a single cycle CPU using pipelining. Pipelining is a powerful logic design method to reduce the clock time and improve the throughput, even though it increases the latency of an individual task and adds additional logic. In a pipelined CPU, multiple instructions are overlapped in execution. This is a good example of parallelism, which is one of the great ideas in computer architecture. To obtain a pipelined CPU, we will take the following steps.

Step 1: Pipeline Registers

Pipelining starts from adding pipelining registers by dividing a large combinational logic. We have already chopped a single cycle CPU into five stages, and thus, will add pipeline registers between two stages.

Step 2: Performance Analysis

A great advantage of pipelining is the performance improvement with a shorter clock time. We will use the same timing parameters as those in the previous discussion.

Element	Register clk-to-q	Register Setup	MUX	ALU	Mem Read	Mem Write	RegFile Read	RegFile Setup
Parameter	$t_{clk\text{-}to\text{-}q}$	t_{setup}	t_{mux}	t_{ALU}	$t_{MEMread}$	$t_{MEMwrite}$	t_{RFread}	$T_{RFsetup}$
Delay(ps)	30	20	25	200	250	200	150	20

Q1. What was the clock time and frequency of a single cycle CPU?

$t_{clk,single} >= t_{PC,\ clk\text{-}to\text{-}q} + t_{IMEMread} + t_{RFread} + t_{mux} + t_{ALU} + t_{DMEMread} + t_{mux} + t_{RFsetup}$

$= 30 + 250 + 150 + 25 + 200 + 250 + 25 + 20 = 950$ ps

$f_{clk,single} = 1/t_{clk,pipe} <= 1/ (950 \text{ ps}) = 1.05$ GHz

Q2. What is the clock time and frequency of a pipelined CPU?

Assuming immediate generator doesn't take any time:

the Memory stage takes the longest time:

$t_{clk\text{-}to\text{-}q} + t_{memread} + t_{mux} + t_{setup} = 325$ ps

$f_{clk,pipe} = 1/t_{clk,pipe} <= 1/ (325 \text{ ps}) = 3.07$ GHz

Q3. What is the speed-up? Why is it less than five?

Speed-up = $t_{clk,single} / t_{clk,pipe} = f_{clk,pipe} / f_{clk,single} = 2.9$.

This is because pipeline stages are not balanced evenly and there is overhead from pipeline registers ($t_{clk\text{-}to\text{-}q}$, t_{setup}). Moreover, this does not include the delays from the additional logic for hazard resolution.

Step 3: Pipeline Hazard

The performance improvement comes at a cost. Pipelining introduces pipeline hazards we have to overcome.

Structural Hazard

Structural hazards occur when more than one instruction use the same resource at the same time.

- **Register File:** One instruction reads from the register file while another writes to it. We can solve this by having separate read and write ports and writing to the register file at the falling edge of the clock.
- **Memory:** The memory is accessed not only for the instruction but also for the data. Separate caches for instructions and data solve this hazard.

Data Hazard and Forwarding

Data hazards occur due to data dependencies among instructions. Forwarding can solve many data hazards.

Q1. Spot the data dependencies in the code below and figure out how forwarding can resolve data hazards.

Instruction	C0	C1	C2	C3	C4	C5	C6
addi t0, s0, -1	IF	REG	EX	MEM	WB		
and s2, t0, a0		IF	REG	EX	MEM	WB	
sw s0, 100(t0)			IF	REG	EX	MEM	WB

The REG step for instructions 2 and 3 depend on data in the registers only available after the WB step of instruction 1. We can forward the ALU output of the first instruction to the EX stages of future instructions

Q2. In general, under what conditions will an EX stage need to take in forwarded inputs from previous instructions? Where should those inputs come from in regards to the current cycle? Assume you have the signals ALUout(n), rs2(n), rs1(n), regWrite(n), and regDst(n), where n is 0 for the signal of the current instruction being executed by the EX stage, -1 for the previous, etc.

Forward ALUout(-1) if (rs2(0) == regDst(-1) || rs1(0) == regDst(-1)) && regWrite(-1)
Forward ALUout(-2) if (rs2(0) == regDst(-2) || rs1(0) == regDst(-2)) && regWrite(-2)
Forward ALUout(-3) if (rs2(0) == regDst(-3) || rs1(0) == regDst(-3)) && regWrite(-3)

Data Hazard and Stall

Q1. Spot the data dependencies in the code below and figure out why forwarding cannot resolve this hazard.

Instruction	C0	C1	C2	C3	C4	C5
lw t0, 20(s0)	IF	REG	EX	MEM	WB	
addiu t1, t0, t0		IF	REG	EX	MEM	WB

The add instruction needs the value of t0 in the beginning of C3, but it is ready at the end of C3.

Q2. What can we do to solve this data hazard?

Instruction	C0	C1	C2	C3	C4	C5	C6
lw t0, 20(s0)	IF	REG	EX	MEM	WB		
nop/instruction		IF	REG	EX	MEM	WB	
addiu t1, t0, t0			IF	REG	EX	MEM	WB

We can insert a nop into the load delay slot as shown above, or even better we can reorder instructions and fill up the load delay slot with an instruction to avoid performance loss.

Control Hazard and Prediction

Control hazards occur due to jumps and branches. We could stall the pipeline, but this decreases performance.

Q1. What can we do to increase performance when we encounter a control hazard?

The naïve way is to flush the pipeline if the branch is taken. The better solution is to try to predict the branch taken. If our prediction is wrong, we have to flush the pipeline, else everything can continue with no performance loss.

Putting It All Together

Instruction	C0	C1	C2	C3	C4	C5	C6			
1. addi t0, s0, -1	IF	REG	EX	MEM	WB					
2. and s2, t0, a0		IF	REG	EX	MEM	WB				
3. sw s2, 100(t0)			IF	REG	EX	MEM	WB			
4. beq s0, s3, label				IF	REG	EX	MEM	WB		
5. addi t2, x0, x0					IF	REG	EX	MEM	WB	

Given the RISC-V code above and a pipelined CPU with no forwarding, how many hazards would there be? What types are each hazard? Consider all possible hazards from all pairs of instructions.

hazards from 1-2, 1-3, 2-3, 4-5

4-5 is a control hazard, all others are data hazard.

How many stalls would there need to be in order to fix the data hazard(s)? What about the control hazard(s)?

Assuming concurrent reads and writes to registers are possible, two stalls for instruction 2 are needed for register t0 between 1 and 2, two stalls are needed for the register s2 between 2 and 3.

Either flush the pipeline or use branch prediction to deal with the control hazard.