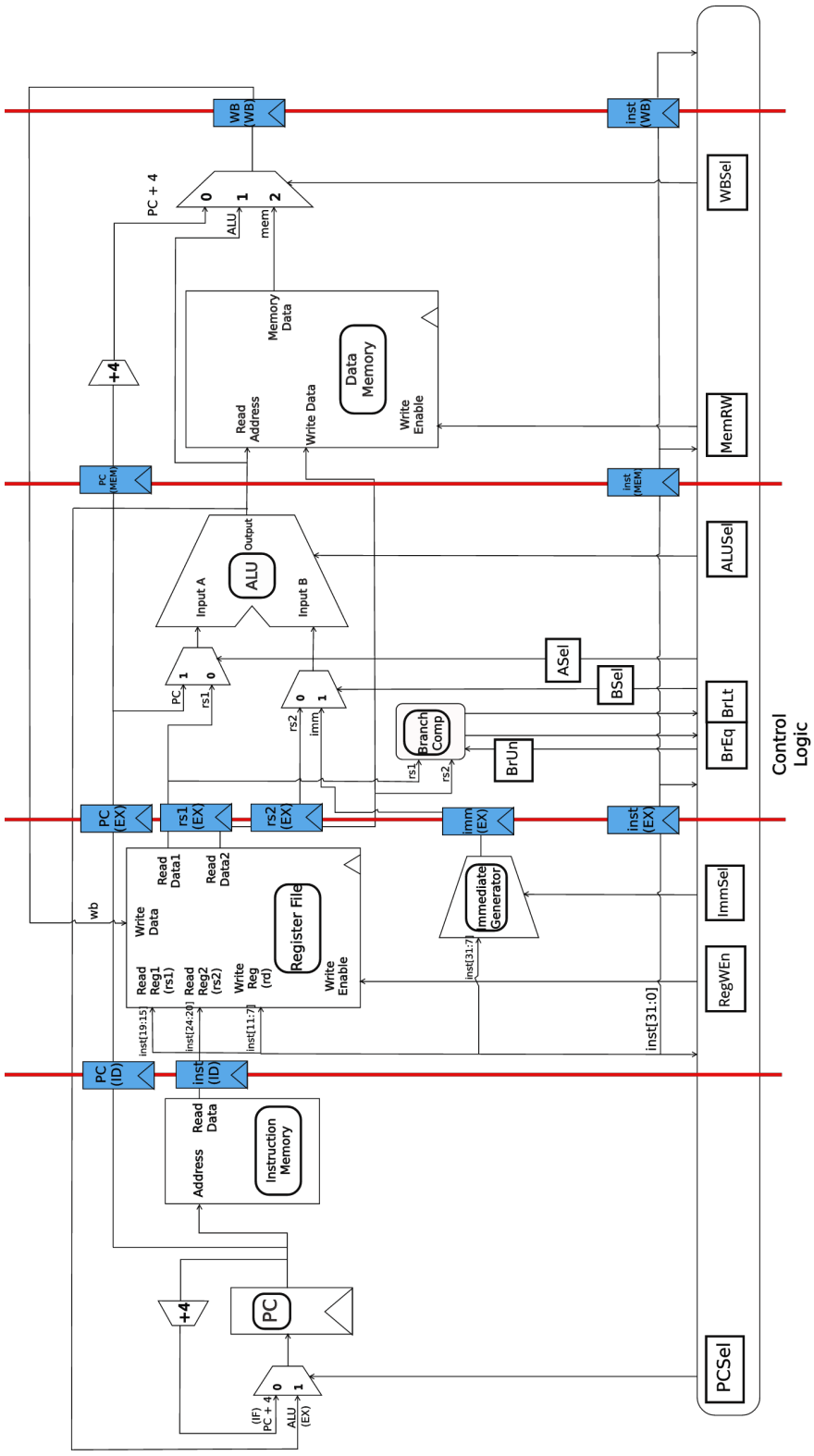
CS61C Summer 2018 Discussion 6 – Single Cycle Datapath

Pipelining Registers

In order to pipeline, we add registers between the five datapath stages. Label each of the five stages (IF, ID, EX, MEM, and WB) on the diagram below:



1. Why do we add +4 to the PC again in the memory stage?
   We add +4 to the PC again in the memory stage so we don't need to pass both PC and PC + 4 along the whole pipeline

2. Why do we need to save the instruction in a register multiple times?
   We need to save the instruction in a register multiple times because each pipeline stage needs to receive the right control signals for the instruction currently in that stage.

Performance Analysis

Element	Register clk-to-q	Register Setup	MUX	Branch Comp	ImmGen
Delay (ps)	30	20	25	75	75

Element	ALU	MemRead	MemWrite	RegFile Read	RegFile Setup
Delay (ps)	200	250	200	150	20

1. With the delays provided above for each of the datapath components, what would be the fastest possible clock time for a single cycle datapath?
   t_clk $\geq$ t_PCclk-to-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
   1/950 ps = 1.05 GHz

2. What is the fastest possible clock time for a pipelined datapath?
   IF: t_PCclk-to-q + t_IMEMread + t_RegSetup = 30 + 250 + 20 = 300 ps
   ID: t_Regclk-to-q + t_RFread + t_RegSetup = 200 ps
   EX: t_Regclk-to-q + t_mux + t_ALU + t_RegSetup = 25 + 200 = 275 ps
   MEM: t_Regclk-to-q + t_DMEMread + t_mux + t_RegSetup = 325 ps
   WB: t_Regclk-to-q + t_RFsetup = 30 + 20 = 50 ps

   Max(IF, ID, EX, MEM, WB) = 325 ps = pipelined cycle time

3. What is the speedup from the single cycle datapath to the pipelined datapath? Why is the speedup less than 5?
   950 ps / 325 ps = 2.9x speedup. The speedup is less than 5 because of (1) the necessity of adding pipeline registers, which have clk-to-q and setup times, and (2) the need to set the clock to the maximum of the five stages, which take different amounts of time. Note: because of hazards, which require additional logic to resolve, the actual speedup would likely be even less than 2.9x.

Hazards

One of the costs of pipelining is that it introduces three types of pipeline hazards: structural hazards, data hazards, and control hazards.

Structural Hazards

Structural hazards occur when more than one instruction needs to use the same datapath resource at the same time. There are two main causes of structural hazards:

- **Register File:** The register file is accessed both during ID, when it is read, and during WB, when it is written to. We can solve this by having separate read and write ports. To account for reads and writes

to the same register, processors usually write to the register during the first half of the clock cycle, and read from it during in the second half.

- **Memory:** Memory is accessed for both instructions and data. Having a separate instruction memory (abbreviated IMEM) and data memory (abbreviated DMEM) solves this hazard.

Something to remember about structural hazards is that they can always be resolved by adding more hardware.

Data Hazards

Data hazards are caused by data dependencies between instructions. In CS 61C, where we will always assume that instructions are always going through the processor in order, we see data hazards when an instruction **reads** a register before a previous instruction has finished **writing** to that register.

Forwarding

Most data hazards can be resolved by forwarding, which is when the result of the EX or MEM stage is sent back to the beginning of the EX stage for a following instruction to use.

1. Look for data hazards in the code below, and figure out how forwarding could be used to solve them.

Instruction	C1	C2	C3	C4	C5	C6	C7
1. addi t0, a0, -1	IF	ID	EX	MEM	WB		
2. and s2, t0, a0		IF	ID	EX	MEM	WB	
3. sltiu a0, t0, 5			IF	ID	EX	MEM	WB

   There are two data hazards, between instructions 1 and 2, and between instructions 1 and 3. The first could be resolved by forwarding the result of the EX stage in C3 to the beginning of the EX stage in C4, and the second could be resolved by forwarding the result of the EX stage in C3 to the beginning of the EX stage in C5.

2. Imagine you are a hardware designer working on a CPU's forwarding control logic. How many instructions after instruction 1 in the code above could be affected a potential data hazard created by this instruction?
   Three instructions. For example, with the addi instruction, any instruction that uses t0 that has its ID stage in C3, C4, or C5 will not have the result of addi's writeback in C5.

3. You have the signals rs1, rs2, RegWEn, and rd for two instructions, instruction n and the instruction right after it, instruction n + 1. Write a condition you can check to see if there is a data hazard between the two instructions, in terms of these signals.
   Forward ALU output of n IF (rs1(n + 1) == rd(n) OR rs2(n + 1) == rd(n)) AND RegWEn(n) == 1

Stalls

1. Look for data hazards in the code below. One of them cannot be solved with forwarding–why?

Instruction	C1	C2	C3	C4	C5	C6	C7	C8
1. addi s0, s0, 1	IF	ID	EX	MEM	WB			
2. addi t0, t0, 4		IF	ID	EX	MEM	WB		
3. lw t1, 0(t0)			IF	ID	EX	MEM	WB	
4. add t2, t1, x0				IF	ID	EX	MEM	WB

There are two data hazards in the code. The first hazard is between instructions 2 and 3, from t0, and the second is between instructions 3 and 4, from t1. The hazard between instructions 2 and 3 can be resolved with forwarding, but the hazard between instructions 3 and 4 cannot be resolved with forwarding. This is because even with forwarding, instruction 4 needs the result of instruction 3 at the beginning of C6, and it won't be ready until the end of C6.

2. What could we do instead of forwarding to solve this data hazard?
   There are two solutions. The first is to insert a nop (no-operation) between instructions 3 and 4. The second is to reorder the instructions 2-3-1-4, because instruction 1 has no dependencies.

Control Hazards

Control hazards are caused by **jump and branch instructions**, because for all jumps and some branches, the next PC is not PC + 4, but the result of the computation completed in the EX stage. We could stall the pipeline for control hazards, but this decreases performance.

1. Besides stalling, what can we do to resolve control hazards?
   We can predict which way branches will go, and when this prediction is incorrect, "flush" the pipeline and continue with the correct instruction. (The most naive prediction method is to simply predict that branches are always not taken).

Putting it All Together

Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9
1. sub t1, s0, s1	IF	ID	EX	MEM	WB				
2. or s0, t0, t1		IF	ID	EX	MEM	WB			
3. sw s1, 100(s0)			IF	ID	EX	MEM	WB		
4. bgeu s0, s2, label				IF	ID	EX	MEM	WB	
5. add t2, x0, x0					IF	ID	EX	MEM	WB

1. 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.
   There are four hazards: between instructions 1 and 2 (data hazard from t1), instructions 2 and 3 (data hazard from s0), instructions 2 and 4 (from s0), and instructions 4 and 5 (a control hazard).

2. How many stalls would there need to be in order to fix the data hazard(s)? What about the control hazard(s)?
   Assuming that we can read and write to the RegFile on the same cycle, two stalls are needed between instructions 1 and 2, and two stalls are needed between instructions 2 and 3. No stalls are needed for the control hazard, because it can be handled with branch prediction/flushing the pipeline.