Computer Design — Pipelining Hazards

Computer Science (2nd year B.Sc.) — Seminar #7

1. Pre-check

This section is designed as a check to help you determine whether you understand the concepts covered in class. Please answer "true/false" to the following questions and include an explanation.

1.1. Introducing “pipeline” registers to the Datapath induces higher latency and throughput in the execution of instructions.

1.2. "Data hazards" typically result in three pipeline stalls if we don’t use the forwarding unit.

1.3. All "data hazards" can be resolved using the “forwarding unit”.

1.4. Suspending the pipeline is the only way to resolve the "branch hazards."

2. Pipeline Registers

2.1. To convert a single-cycle processor into a pipelined processor, registers are introduced between the different stages of the data path. What is the role of these registers?

2.2. Why do we need to save the “control signals” multiple times in the pipeline?

3. Performance Analysis

Before continuing with the questions in this tutorial, please have a look at the document "**Timing Constraints for digital circuits**" (available on the course website) for a quick note on the time keywords (i.e. t_{cpq}, t_{pd}, …) used here.

Let's assume the following time durations for our pipelined CPU:

t_{cpq} register :	30 ps ,	t_{setup} register :	20 ps
t_{pd} Mux:	25 ps ,	t_{pd} ALU:	200 ps
t_{cpq} Mem:	250 ps ,	t_{setup} MEM:	200 ps
t_{cpq} RegFile:	150 ps ,	t_{setup} RegFile:	20 ps

3.1. With the delays shown above for each of the Datapath components, what would be the fastest possible clock time for a single-cycle Datapath?

3.2. What would be the fastest possible clock frequency for a pipelined data path?

3.3. What is the speedup after converting the single-cycle Datapath to a pipelined Datapath? Why is the speedup less than 5?

4. Execution hazards

Converting a single-cycle data path to a pipelined version introduces three types of execution hazards: structural hazards, data hazards, and branch hazards.

Structural hazards

They occur when multiple instructions need to use the same resource in the data path at the same time. There are two main causes of structural hazards:

RegFile Access: The "registers file" is accessed both during the ID stage, when it is read, and during the WB stage, when it is modified (i.e. written to). This problem is solved by having separate read and write ports. In the case of simultaneous reading and writing to the same register, one writes to the register during the first half of the clock cycle and reads from it during the second half. This is also known as "double pumping."

Memory Access: Memory holds both the instructions of the program to be executed (accessed during the IF stage) and the data to be processed (required during the MEM stage). Having a separate instruction memory (IMEM) and data memory (DMEM) resolve the problem of simultaneous access to this resource by different instructions in the pipeline. This design is based on an old architecture called "Harvard architecture" and is implemented today by using separate memory caches for instructions and data (cf. the lecture on memory caches).

☞ **Structural hazards can always be solved by adding more hardware components.**

Data hazards

These hazards are caused by data dependencies between instructions. In particular, a data hazard occurs when an instruction reads from a register before a previous instruction has finished writing to that register.

Forwarding Unit

Most data hazards can be resolved by “forwarding”, i.e. when the output of the EX stage or the MEM stage is forwarded to the EX stage of the subsequent instruction.

4.1. Point out the data hazards in the code below and indicate how data forwarding could be used to resolve them.

Instructions \ Cycles	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

4.2. How many instructions after an `addi` statement could be affected by data hazards related to that statement?

Pipeline suspension

4.3. Identify the data hazards in the code below. Why can't one of these hazards be resolved with the forwarding unit? What can we do to resolve this issue?

Instructions \ Cycles	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, $0`				IF	ID	EX	MEM	WB

4.4. In a team of engineers, you are working on the design of a compiler for a MIPS processor (remember, it is the compiler that produces assembly language files). How could this compiler reorganise the instructions in Exercise 4.3 to minimise data hazards while ensuring the same result.

Data Hazard Detection

Suppose we have the signals `rs`, `rt`, `RegWEn`, and `rd` for two instructions at times t and $t + 1$. We want to check if there is a data hazard between these statements. In this sense, we can verify whether the `rd` of the instruction at time t matches `rs` or `rt` of the instruction at time $t + 1$, thereby indicating a data hazard. We could then use this detection to decide which data to forward (i.e. from `EX/MEM` or `MEM/WB`) or the hold period (if any) to apply to ensure correct execution of the instructions. In pseudocode, this gives:

```
if (rs(n + 1) == rd(n) || rt(n + 1) == rd(n) && RegWEn(n) == 1) {
    /* forward the output of the ALU of inst. n to inst. n+1 */
}
```

Branch hazards

Branch hazards are caused by branch instructions. Indeed, in a sequential execution without branching, the next instruction to execute is located at the address given by PC + 4. In the case of branch instructions, the next PC is not PC + 4, but will be the result of the calculation performed during the EX stage. A possible solution to this problem is to suspend the pipeline during a branch hazard, but this will negatively impact performance.

4.5. Besides suspending the pipeline, what can we do to resolve branch hazards?

4.6. How many execution hazards would there be in the MIPS code below if it were executed in a pipelined processor **without** a forwarding unit? What is the type of each "execution hazard"? (Examine all possible instruction pairs for possible hazards).

How many times should the pipeline be stalled to resolve possible data hazards? What about branch hazards?

Cycles / Instructions	C1	C2	C3	C4	C5	C6	C7	C8	C9
`1. sub $t1, $s0, 1`	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. beq $s0, $s2, L1`				IF	ID	EX	MEM	WB	
`5. add $t2, $0, $0`					IF	ID	EX	MEM	WB