



BROWN

Computer System Design Lecture 8: Pipeline Hazards

Prof. R. Iris Bahar
EN164
February 12, 2007

Reading: Appendix A, sections A.1-A.3



BROWN

Pipelining

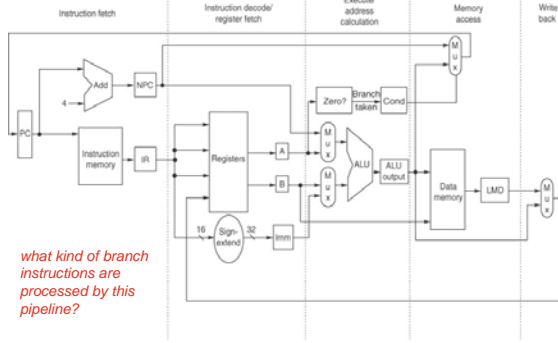
- What makes it easy (specifically for implementing RISC)
 - all instructions are the same length
 - just a few instruction formats
 - memory operands appear only in loads and stores
 - aligned data: one memory access for one data item
- Hazards make it hard
 - next instruction cannot execute in the following clock cycle
 - structural, control and data hazards
- What makes it really hard
 - exception handling
 - trying to improve performance with out-of-order execution, etc.

EN164
Lecture 8-2



BROWN

Instruction Steps in Detail



BROWN

Pipelined Datapath

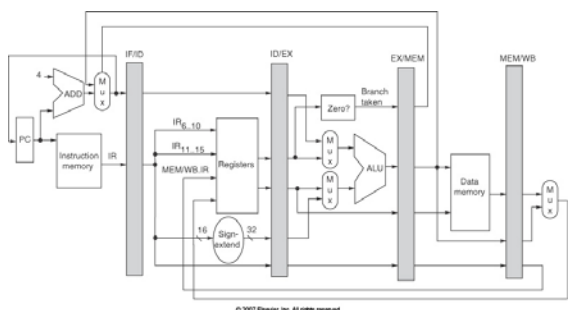
- Reuse of functional units in every clock cycle
- Additional hardware
 - Separation of pipeline stages by pipeline registers
 - Latching data between stages into registers increases latency per instruction
 - Multiple functional units if used by several instructions at the same time (for removing structural hazards)
 - *What HW unit is duplicated in our MIPS 5-stage pipeline?*
- Extended control
 - Strict “sequentialization” of instruction
 - every instruction goes through all stages
 - Check for hazards
 - Introduce stalls to remove hazards

EN164
Lecture 8-5



BROWN

Pipelined Datapath



EN164
Lecture 8-7



BROWN

Some Potential Hazards

- Usually data moves from left to right (conceptually)
 - ➔ Data moving from right to left affects later instructions
- Write back into the register file can lead to data hazards
- Selection of the next value of the PC leads to control hazards
- A simple way to deal with hazards is to **STALL** the pipeline, leading to inefficiencies in throughput

EN164
Lecture 8-8

Potential Hazards

- Structural Hazards:**
 - Resource conflicts where HW cannot support all possible combinations of instructions through the pipeline
 - Examples: Single memory port, R/W register file in same cycle
 - How does our 5-stage pipeline deal with these hazards?
 - What would happen if our pipeline had only 1 memory port?

	1	2	3	4	5	6	7	8	9	10
load	IF	ID	EX	MEM	WB					
add		IF	ID	EX	MEM	WB				
sub			IF	ID	EX	MEM	WB			
shift				stall	IF	ID	EX	MEM	WB	
store						IF	EX	MEM	WB	

EN164
Lecture 8-10

Potential Hazards

- Data Hazards:**
 - Instruction depends on results of previous instruction not yet available
 - Where are the data dependencies in the following code sequence?

```
ADD R1, R2, R3 // R1 ← R2 + R3
SUB R4, R1, R5
AND R6, R1, R7
OR R8, R1, R9
XOR R10, R1, R11
```
 - How does our register file design help avoid data hazards?
 - Do we always have to wait until the data is written to the register file in order to use results?

EN164
Lecture 8-11

Examples of Data Hazards

Time (in clock cycles) →

CC1 CC2 CC3 CC4 CC5 CC6

add r1, r2, r3

sub r4, r1, r5

and r6, r1, r7

or r8, r1, r9

How many cycles do we need to stall to handle the data hazards?

EN164
Lecture 8-15

Data Hazards

- Stalling to avoid data hazards**

	1	2	3	4	5	6	7	8	9	10
ADD	IF	ID	EX	MEM	WB					
SUB		IF	ID	stall	stall	EX	MEM	WB		
AND			IF	stall	stall	ID	EX	MEM	WB	
OR				stall	stall	IF	ID	EX	MEM	WB
XOR							IF	ID	EX	MEM

Why do I stall the SUB starting in the ID stage and not the IF stage?

Are any stalls strictly necessary here?

EN164
Lecture 8-16