Comp4611 Tutorial 3

Pipelining and Data Hazards

Oct. 3, 5, 8 2012

Outline

1. Pipelining
2. What is Data Hazards ?
3. Data Hazard Detection
4. Data Hazard Solutions
5. Data Hazard Example

Pipelining Exercise

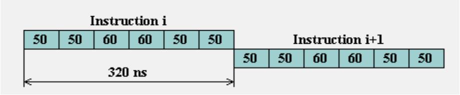
Consider a nonpipelined machine with 6 execution stages of lengths 50 ns, 50 ns, 60 ns, 60 ns, 50 ns, and 50 ns.

- Find the instruction latency on this machine.
- How much time does it take to execute 100 instructions ?

Solution:

Instruction latency = 50+50+60+60+50+50= 320 ns

Time to execute 100 instructions = 100*320 = 32000 ns



Pipelining Exercise

Suppose we introduce pipelining on this machine. Assume that when pipelining is introducing, the clock skew adds 5 ns of overhead to each execution stage.

- What is the instruction latency on the pipelined machine ?
- How much time does it take to execute 100 instructions ?

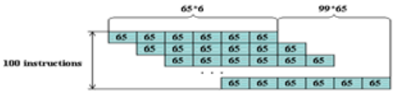
Solution:

Note: the length of the pipe stages must all be the same.

Length of pipelined stage = MAX (lengths of unpipelined stages) + overhead = 60 + 5 = 65 ns

Instruction latency = 65*6 = 390 ns

Time to execute 100 instructions = 65*6 + 65*99 = 6825 ns



Pipelining Exercise

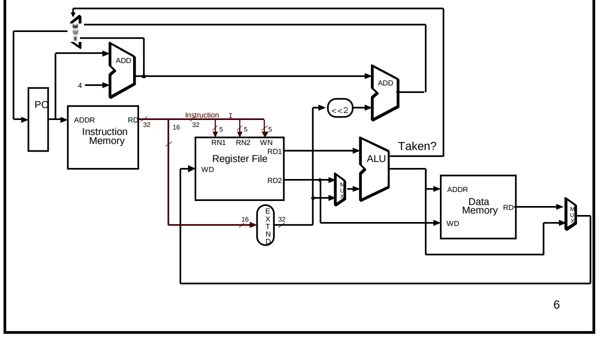
What is the speedup obtained from pipelining?

Speedup is the ratio of the average instruction time without pipelining to the average instruction time with pipelining. (here we do not consider any stalls introduced by different types of hazards which we will look at in the next section)

Solution:

Speedup = Old Execution Time / New Execution Time
= 32000 / 6825
= 4.69

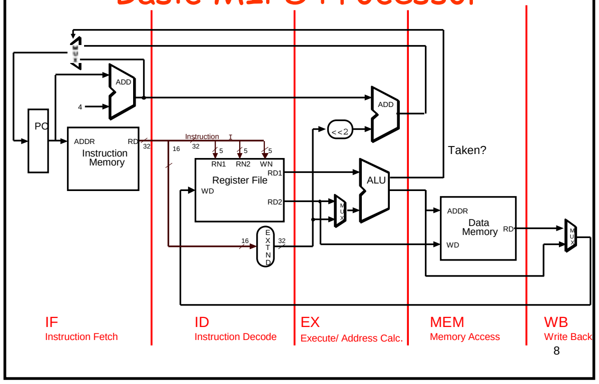
Basic MIPS Integer Datapath



Pipelining in MIPS

- **Question:** What happens if we break execution into multiple cycles?
- **Answer:** in the best case, we can start executing a new instruction on each clock cycle - this is pipelining
- Pipelining stages:
  - IF: Instruction Fetch
  - ID: Instruction Decode
  - EX: Execute / Address Calculation
  - MEM: Memory Access (read / write)
  - WB: Write Back (results into register file)

Basic MIPS Processor



Simple MIPS Pipelined Integer Instruction Processing

	Clock Number					Time in clock cycles →			
Instruction N°	1	2	3	4	5	6	7	8	9
Instruction I	IF	ID	EX	MEM	WB				
Instruction I+1		IF	ID	EX	MEM	WB			
Instruction I+2			IF	ID	EX	MEM	WB		
Instruction I+3				IF	ID	EX	MEM	WB	
Instruction I +4					IF	ID	EX	MEM	WB

← Time to fill the pipeline →

First instruction I completed

Last instruction I+4 completed

MIPS Pipeline Stages:

IF = Instruction Fetch
ID = Instruction Decode
EX = Execution
MEM = Memory Access
WB = Write Back

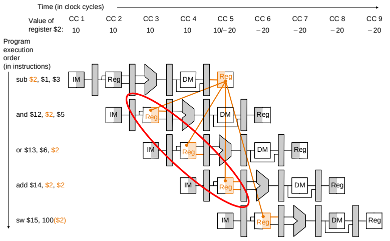
Pipelining Hazards

Hazards prevent next instruction from executing during its designated clock cycle

- Structural hazards
  - Caused by hardware resource conflicts
- Data hazards
  - Arise when an instruction depends on the results of a previous instruction
- Control hazards
  - Caused by change of control (e.g. jump)

Data Hazards

Data hazards occur when data is used before it is ready



The use of the result of the SUB instruction in the next three instructions causes a data hazard, since the register $2 is not written until after those instructions read it.

Data Hazards

Read After Write (RAW)

Instr_J tries to read operand before Instr_I writes it

Execution Order is:
Instr_I
Instr_J

```
I: add r1,r2,r3
J: sub r4,r1,r3
```

Caused by a "Dependence" (in compiler nomenclature). This hazard results from an actual need for communication.

Data Hazards

Write After Read (WAR)

Instr_J tries to write operand *before* Instr_I reads i

- Gets wrong operand

Execution Order is:
Instr_I
Instr_J

```
I: sub r4,r1,r3
J: add r1,r2,r3
K: mul r6,r1,r7
```

- Called an "Anti-Dependence" by compiler writers. This results from reuse of the name "r1".
- Can this data hazard happen in MIPS 5 stage pipeline ?

Data Hazards

Write After Write (WAW)

Instr_J tries to write operand *before* Instr_I writes it

- Leaves wrong result (Instr_I not Instr_J)

Execution Order is:
Instr_I
Instr_J

```
I: sub r1,r4,r3
J: add r1,r2,r3
K: mul r6,r1,r7
```

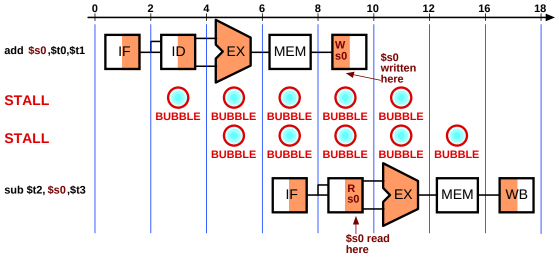
- Called an "Output-Dependence" by compiler writers. This also results from the reuse of name "r1".
- Can't happen in MIPS 5 stage pipeline because:
  - All instructions take 5 stages, and
  - Writes are always in stage 5
- Will see WAR and WAW later in more complicated pipes

Data Hazards Solutions

Solutions for Data Hazards

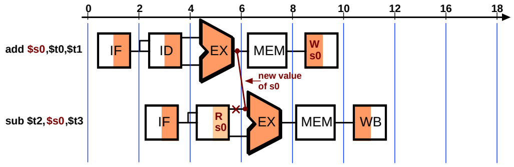
- Stalling
  - Add bubbles
- Forwarding
  - Connect new value directly to next stage
- Reordering

Data Hazard - Stalling



Data Hazards - Forwarding

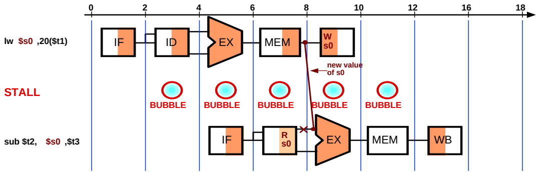
- **Key idea:** connect new value directly to next stage
- Still read s0, but ignore in favor of new result



- **Problem:** what about load instructions?

Data Hazards - Forwarding

- STALL still required for load - data avail. after MEM
- MIPS architecture calls this delayed load, initial implementations required compiler to deal with this



Data Hazards - Reordering

- Assuming we have data forwarding, what are the hazards in this code ?

```
lw $t0, 0($t1)
lw $t2, 4($t1)
add $s3, $t2, $s4
add $s5, $t0, $s6
```

- Reorder instructions to remove hazard:

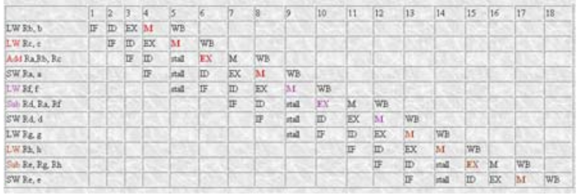
```
lw $t0, 0($t1)
lw $t2, 4($t1)
add $s5, $t0, $s6
add $s3, $t2, $s4
```

Data Hazards Example

For the following sequence of statements:

a = b + c   d = a – f   e = g - h

One solution would look like this:



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
LW Rb, b	IF	ID	EX	M	WB													
LW Rc, c		IF	ID	EX	M	WB												
Add Ra,Rb, Rc			IF	ID	stall	EX	M	WB										
SW Ra, a				IF	stall	ID	EX	M	WB									
LW Rf, f					stall	IF	ID	EX	M	WB								
Sub Rd, Ra, Rf							IF	ID	stall	EX	M	WB						
SW Rd, d								IF	stall	ID	EX	M	WB					
LW Rg, g									stall	IF	ID	EX	M	WB				
LW Rh, h											IF	ID	EX	M	WB			
Sub Re, Rg, Rh												IF	ID	stall	EX	M	WB	
SW Re, e													IF	stall	ID	EX	M	WB

Data Hazards Example

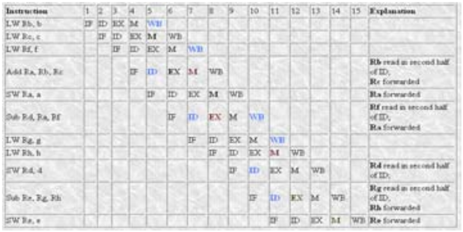
Observation:

- In time steps 4, 5, and 6, there are two forwards from the Data memory unit to the ALU in the EX stage of the Add instruction.
- So also the case in time steps 13, 14, and 15.
- The hardware to implement this forwarding will need two Load Memory Data places to store the output of data memory.
- Note that for the SW instructions, the register value is needed at the input of Data memory.

Data Hazards Example

The better solution with compiler assist is given below:

(Rather than just allow the pipeline to stall, the compiler could avoid these stalls by rearranging the code sequence to eliminate the hazards.)



Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	**Explanation**
LW Rb, b	IF	ID	EX	M	WB											
LW Rc, c		IF	ID	EX	M	WB										
LW Rf, f			IF	ID	EX	M	WB									
Add Ra, Rb, Rc				IF	ID	EX	M	WB								**Rb** read in second half of ID; **Rc** forwarded
SW Ra, a					IF	ID	EX	M	WB							**Ra** forwarded
Sub Rd, Ra, Rf						IF	ID	EX	M	WB						**Rf** read in second half of ID; **Ra** forwarded
LW Rg, g							IF	ID	EX	M	WB					
LW Rh, h								IF	ID	EX	M	WB				
SW Rd, 4									IF	ID	EX	M	WB			**Rd** read in second half of ID;
Sub Re, Rg, Rh										IF	ID	EX	M	WB		**Rg** read in second half of ID; **Rh** forwarded
SW Re, e											IF	ID	EX	M	WB	**Re** forwarded