## CSE 2021 Computer Organization

## Chapter 4 Part 3

## The Processor - Pipelining

## Pipeline Summary

Pipelining improves performance by increasing instruction throughput

- Executes multiple instructions in parallel
- Each instruction has the same latency
- Subject to hazards
- Structure, data, control

Instruction set design affects complexity of pipeline implementation

## MIPS Pipelined Datapath



Chapter 4 - The Processor - 3

## Pipeline registers

## Need registers between stages

- To hold information produced in previous cycle


Chapter 4 - The Processor - 4

## Pipeline Operation

Cycle-by-cycle flow of instructions through the pipelined datapath

- "Single-clock-cycle" pipeline diagram

Shows pipeline usage in a single cycle

- Highlight resources used
- c.f. "multi-clock-cycle" diagram

Graph of operation over time
We'll look at "single-clock-cycle" diagrams for load \& store

## Single-Clock-Cycle Diagram

## IF for Load and Store



Chapter 4 - The Processor - 6

## ID for Load, Store, ...



Chapter 4 - The Processor - 7

## EX for Load



Chapter 4 - The Processor - 8

## MEM for Load



Chapter 4 - The Processor - 9

## WB for Load



Chapter 4 - The Processor - 10

## Corrected Datapath for Load



Chapter 4 - The Processor - 11

## EX for Store



Chapter 4 - The Processor - 12

## MEM for Store



Chapter 4 - The Processor - 13

## WB for Store



Chapter 4 - The Processor - 14

## Multi-Cycle Pipeline Diagram

## Form showing resource usage

```
\begin{tabular}{llllllll} 
CC 1 & CC 2 & CC 3 & CC 4 & CC 5 & CC 6 & CC 7 & CC 8
\end{tabular} CC 9
```

Program
execution
order
(in instructions)
Iw \$10, 20(\$1)

sub \$11, \$2, \$3
add \$12, \$3, \$4

Iw \$13, 24(\$1)
add \$14, \$5, \$6

## Multi-Cycle Pipeline Diagram

## Traditional form

Program
execution
order
(in instructions)

Iw \$10, 20(\$1)
sub \$11, \$2, \$3
add \$12, \$3, \$4
Iw \$13, 24(\$1)
add \$14, \$5, \$6
$\begin{array}{lllllllll}\text { Time (in clock cycles) } \\ \mathrm{CC} 1 & \mathrm{CC} 2 & \mathrm{CC} 3 & \mathrm{CC} 4 & \mathrm{CC} 5 & \mathrm{CC} 6 & \mathrm{CC} 7 & \mathrm{CC} 8 & \mathrm{CC} 9\end{array}$


## Single-Cycle Pipeline Diagram

## State of pipeline in a given cycle

| add $\$ 14, \$ 5, \$ 6$ | Iw $\$ 13,24(\$ 1)$ | add $\$ 12, \$ 3, \$ 4$ | sub $\$ 11, \$ 2, \$ 3$ | Iw $\$ 10,20(\$ 1)$ |
| :---: | :---: | :---: | :---: | :---: |
| Instruction fetch | Instruction decode | Execution | Memory | Write-back |



Chapter 4 - The Processor - 17

## Pipelined Control (Simplified)



Chapter 4 - The Processor - 18

## Pipelined Control

Control signals derived from instruction

- As in single-cycle implementation



## Pipelined Control



Chapter 4 - The Processor - 20

## Data Hazards in ALU Instructions

Consider this sequence:

```
sub $2, $1,$3
and $12,$2,$5
or $13,$6,$2
add $14,$2,$2
sw $15,100($2)
```

We can resolve hazards with forwarding

- How do we detect when to forward?


## Dependencies \& Forwarding

| Time (in clock cycles) | CC 2 | CC 3 | CC 4 | CC 5 | CC 6 | CC 7 | CC 8 | CC 9 |  |
| :--- | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Value of | CC 1 | C |  |  |  |  |  |  |  |
| register $\$ 2:$ | 10 | 10 | 10 | 10 | $10 /-20$ | -20 | -20 | -20 | -20 |

Program
execution
order
(in instructions)
sub $\$ 2, \$ 1, \$ 3$
and \$12, \$2, \$5
or \$13, \$6, \$2
add \$14, \$2,\$2
sw \$15, 100(\$2)
Assume that $\$ 2$ : initial value $=10$, value after $\operatorname{sub}=-20$
(nstructions)

## Detecting the Need to Forward

Pass register numbers along pipeline

- e.g., ID/EX.RegisterRs = register number for Rs sitting in ID/EX pipeline register
ALU operand register numbers in EX stage are given by
- ID/EX.RegisterRs, ID/EX.RegisterRt

Data hazards when
1a. EX/MEM.RegisterRd = ID/EX.RegisterRs
1b. EX/MEM.RegisterRd = ID/EX.RegisterRt
2a. MEM/WB.RegisterRd = ID/EX.RegisterRs
2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

Fwd from EX/MEM pipeline reg

Fwd from MEM/WB pipeline reg
*Recall for R-type: add rd, rs, rt, i.e. ALU uses values of rs and rt registers for calculation.

## Detecting the Need to Forward

But only if forwarding instruction will write to a register!

- EX/MEM.RegWrite, MEM/WB.RegWrite And only if Rd for that instruction is not \$zero
- EX/MEM.RegisterRd $=0$, MEM/WB.RegisterRd $=0$


## Forwarding Paths


b. With forwarding

## Forwarding Conditions

EX hazard

- if (EX/MEM.RegWrite and (EX/MEM.RegisterRd $=0$ ) and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) ForwardA = 10
- if (EX/MEM.RegWrite and (EX/MEM.RegisterRd $=0$ ) and (EX/MEM.RegisterRd = ID/EX.RegisterRt)) ForwardB = 10


## MEM hazard

- if (MEM/WB.RegWrite and (MEM/WB.RegisterRd $=0$ ) and (MEM/WB.RegisterRd = ID/EX.RegisterRs)) ForwardA = 01
- if (MEM/WB.RegWrite and (MEM/WB.RegisterRd $=0$ ) and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
ForwardB $=01$


## Datapath with Forwarding



Chapter 4 - The Processor - 27

## Load-Use Data Hazard

Program
execution
order
(in instructions)

Iw \$2, 20(\$1)
and $\$ 4, \$ 2, \$ 5$
or $\$ 8, \$ 2, \$ 6$
add \$9, \$4, \$2
slt \$1, \$6, \$7


Chapter 4 - The Processor - 28

## Load-Use Hazard Detection

Check when using instruction is decoded in ID stage
ALU operand register numbers in ID stage are given by

- IF/ID.RegisterRs, IF/ID.RegisterRt

Load-use hazard when

- ID/EX.MemRead and
((ID/EX.RegisterRt = IF/ID.RegisterRs) or
(ID/EX.RegisterRt = IF/ID.RegisterRt))
If detected, stall and insert bubble


## How to Stall the Pipeline

Force control values in ID/EX register to 0

- EX, MEM and WB do nop (no-operation)

Prevent update of PC and IF/ID register

- Using instruction is decoded again
- Following instruction is fetched again
- 1-cycle stall allows MEM to read data for 1w

Can subsequently forward to EX stage

## Stall/Bubble in the Pipeline



Program
execution
order
(in instructions)

Iw \$2, 20(\$1)
and becomes nop

and $\$ 4, \$ 2, \$ 5$ stalled in ID
or \$8, \$2, \$6 stalled in IF
add $\$ 9, \$ 4, \$ 2$


## Datapath with Hazard Detection



Chapter 4 - The Processor - 32

## Stalls and Performance

Stalls reduce performance

- But are required to get correct results

Compiler can arrange code to avoid hazards and stalls

- Requires knowledge of the pipeline structure


## Branch Hazards

If branch outcome determined in MEM

|  |  |  |  |  |  |  |  |  |  |
| :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- |
| Time (in clock cycles) |  |  |  |  |  |  |  |  |  |
|  | CC 1 | CC 2 | CC 3 | CC 4 | CC 5 | CC 6 | CC 7 | CC 8 | CC 9 |



Chapter 4 - The Processor - 34

## Solution to Control Hazard

Example: branch taken
36: sub $\$ 10, \$ 4, \$ 8$
40: beq $\$ 1, \$ 3,7$
44: and \$12, \$2, \$5
48: or $\$ 13, \$ 2, \$ 6$
52: add \$14, \$4, \$2
56: s7t \$15, \$6, \$7
72: 1w \$4, 50(\$7)
Assume additional hardware to determine outcome of branch in ID stage

- Target address adder: PC+4+4*7=72
- Register comparator: e.g. if \$1=\$3


## Example: Branch Taken



Chapter 4 - The Processor - 36

## Example: Branch Taken



Chapter 4 - The Processor - 37

