Outline
- Pipelining & Performance
- Instruction pipeline
- Speedup
- Pipelined datapath & control
- Hazards / resolution

Machine organization
- Pipelining

Performance

\[
\text{Seconds/program} = \frac{\text{Inst/program}}{\text{CPU}} \times \text{seconds/cycle}
\]

"Instruction count"

\[\uparrow\]

Software optimization

\[\uparrow\]

Levels of logic between registers

Examples of pipelining
- Laundry
- Mfg line / assembly line
- Buffet / cafeteria

- Resources to complete multiple steps in a task
- Multiple tasks at different stages
- Use every stage at once
Instruction Processing Pipeline

- appears as if instructions are executed one after another, but
- multiple instructions execute at once

Instruction every cycle.

- 5 instructions at once
speedup = due to enhancement x

\[ \frac{\text{Time (w/o x)}}{\text{Time (w/ x)}} \]

\[ \varepsilon_k = \frac{1}{n} \text{ cycles in pipeline} \]

\[ 300 \text{ K} = \frac{n}{5} \text{ stage pipeline} \]

\[ S_U = \frac{1}{300} \text{ m} = 3.33 \]

\[ T(n,k) = n + k \]

\[ S_U(n,k) = \frac{k \cdot n}{n + k} = K \left( \frac{n}{n + \frac{n}{k}} \right) \]
Pipelined Instruction Processing in detail

- don't know if instruction is a jump or branch until decode
- have already fetched the next one.

Classic: kill the "prefetched" inst following the jump / branch
- Nop

Delay Slot: execute the instruction following the jump / br then the target
- compiler tries to put something useful there, else Nop
MIPS: delayed branch
- NPC relative

When is branch resolved?

- simple conditional: zero ~ zero, eq, neq
- LT / GT take two inst

⇒ Branch condition resolved in DCD stage
- single delay slot
- compute all possible NPC and select at the last ps

\[\text{PC} + 2 \rightarrow NPC\]
\[\text{IR} \rightarrow \text{PC}\]

Data Hazard

- cannot use a value before it is produced
- real after write
- data dependence

ex: ADD R1, R2, R3
OR R4, R1, R5

Solution: Wait

- detect the hazard
- stall dependent inst.
  till hazard goes away
  (or force compiler to avoid it)
- then organization is made visible
Other data hazards

WAR

Add R1, R1, R2
Add R1, R1, R2

- Read in RF, write in WB
- Instructions issued in order complete in order
Optimization: avoid pipeline stalls

- arrange code to separate dependent instructions

```
Add R1, R2, R3
Addi R3, R1, 16
Sub R2, R2, 1
```

Bypass -

- add mux to grab values in the datapath that are valid but have not been written to the register file

```
Arith / Log: R on W
Load: only W \leq load delay slot
```

MIPS - WB in 1st half of cycle, RF in second

- additional bypass
- move mux to other side of A/B

To Page No. ___

\[\text{I}F \] the text below:

- unsigned
- signed

\[\text{IT}D\] [\text{IT}D]