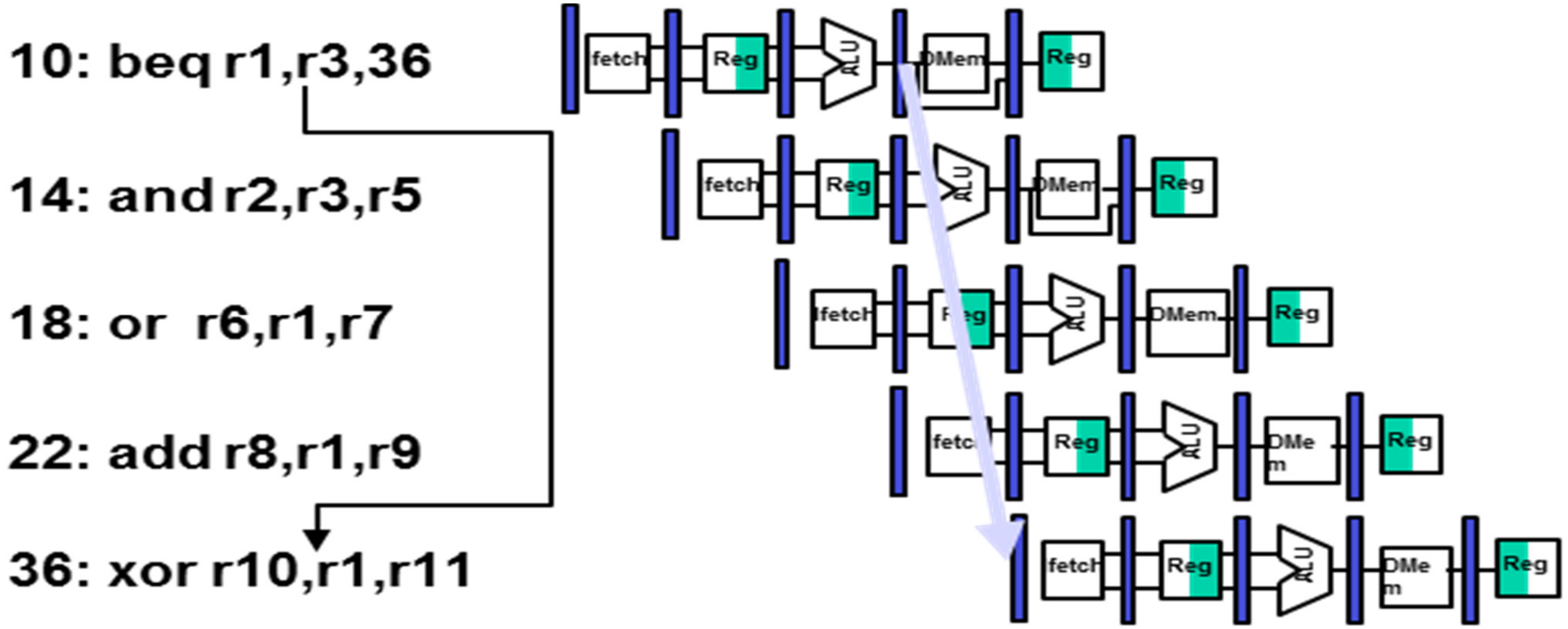
CS203
PIPELINE

Manish Kumar Bajpai

Control Hazards

- A control hazard is when we need to find the destination of a branch, and can't fetch any new instructions until we know that destination



```
10: beq r1,r3,36

14: and r2,r3,r5

18: or  r6,r1,r7

22: add r8,r1,r9

36: xor r10,r1,r11
```

Control Hazards

- If CPI = 1, 30% branch, Stall 3 cycles => new CPI = 1.9?
- (How did we get that 1.9???)
  - Two part solution to this dramatic increase:
  - Determine branch taken or not sooner, AND
  - Compute taken branch address earlier

Control Hazards

- **Five Branch Hazard Alternatives**
- **#1: Stall until branch direction is clear**
- **#2: Predict Branch Not Taken**
  - Execute successor instructions in sequence
  - “Squash” instructions in pipeline if branch actually taken
  - Advantage of late pipeline state update
  - 47% MIPS branches not taken on average
  - PC+4 already calculated, so use it to get next instruction

Control Hazards

- #3: **Predict Branch Taken**
  - 53% MIPS branches taken on average
  - But haven’t calculated branch target address in MIPS
  - MIPS still incurs 1 cycle branch penalty
  - Other machines: branch target known before outcome
- **#4: Execute Both Paths**
- **#5: Delayed Branch**
- – Define branch to take place AFTER a following instruction

Control Hazards

- **Compiler "Static" Prediction of Taken/Untaken Branches**
- **Just overlap tasks, easy if tasks are independent**
- **Two strategies**
  - Backward branch predict taken, forward branch not taken
  - Profile-based prediction: record branch behaviour, predict branch based on prior run

- **Speed Up Vs Pipeline Depth; if ideal CPI is 1, then:**

$$\text{Speedup} = \frac{\text{Pipeline Depth}}{1 + \text{Pipeline stall CPI}} \times \frac{\text{Clock Cycle Unpipelined}}{\text{Clock Cycle Pipelined}}$$

Control Hazards

- **Delayed Branch**
- Where to get instructions to fill branch delay slot?
  - Before branch instruction
  - From the target address: only valuable when branch taken
  - From fall through: only valuable when branch not taken
  - Cancelling branches allow more slots to be filled
- Compiler effectiveness for single branch delay slot:
  - Fills about 60% of branch delay slots
  - About 80% of instructions executed in branch delay slots useful in computation
  - About 50% (60% x 80%) of slots usefully filled
- **Delayed Branch downside: 7-8 stage pipelines, multiple instructions issued per clock (superscalar)**

Control Hazards

- **Evaluating Branch Alternatives**
- Pipeline Speedup $= \dfrac{Pipeline\ Depth}{1 + Branch\ Frequency}$

Pipeline Hazards impact

- Hazards limit performance on computers:
- **Structural:** Need more H/W resources
- **Data (RAW,WAR,WAW):** Need forwarding, compiler scheduling
- **Control:** Delayed branch, Prediction

What Makes Pipelining Hard?

- **Examples of interrupts**
  - Power failing,
  - Arithmetic overflow,
  - I/O device request,
  - OS call,
  - Page fault
- **There are 5 instructions executing in 5 stage pipeline when an interrupt occurs:**
  - How to stop the pipeline?
  - How to restart the pipeline?
  - Who caused the interrupt?
- **Interrupts (also known as: faults, exceptions, traps) often require**
  - surprise jump (to vectored address)
  - linking return address
  - state change (e.g., to kernel mode)

What Makes Pipelining Hard?

- **Complex Addressing Modes and Instructions**
  - Address modes: Auto increment causes register change during instruction execution
  - Interrupts? Need to restore register state
  - Adds WAR and WAW hazards since writes are no longer the last stage.
- **Memory-Memory Move Instructions**
- **Must be able to handle multiple page faults**
- **Long-lived instructions: partial state save on interrupt**
- **Condition Codes**