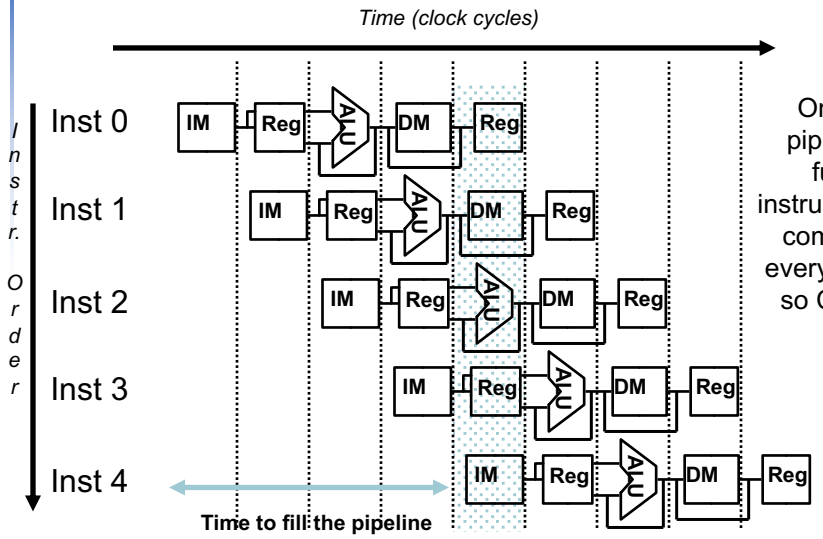
Chapter 4

The Processor
Control Hazards

Review

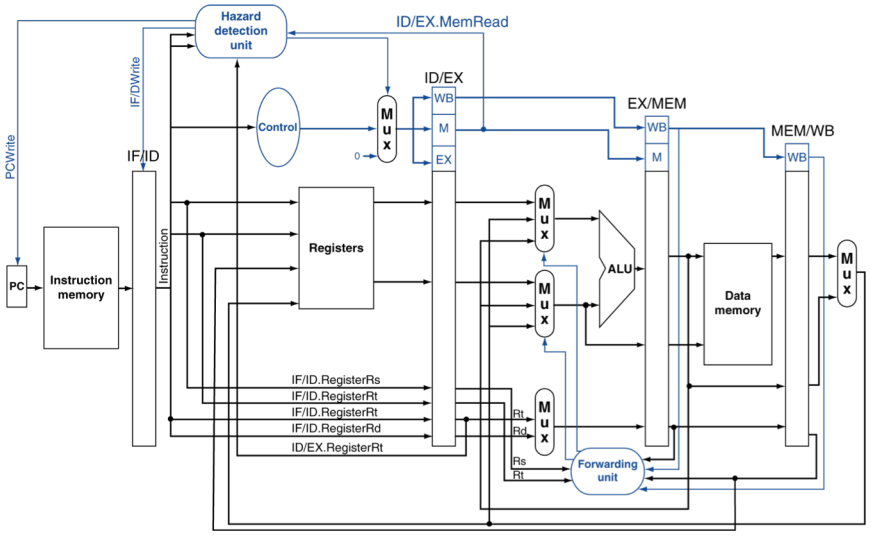
- All modern day processors use pipelining.
- Pipelining doesn't help *latency* of single task, it helps *throughput* of entire workload.
- Potential speedup: a CPI of 1 and faster Clock Cycle.
- Pipeline rate limited by *slowest* pipeline stage
  - Unbalanced pipe stages make for inefficiencies.
  - The time to "fill" pipeline and time to "drain" it can impact speedup for deep pipelines and short code runs.
- Must detect and resolve hazards
  - Data and control hazards.

Review: Five Instruction Sequence



Once the pipeline is full, one instruction is completed every cycle, so CPI = 1

Review: Datapath with Hazard Detection

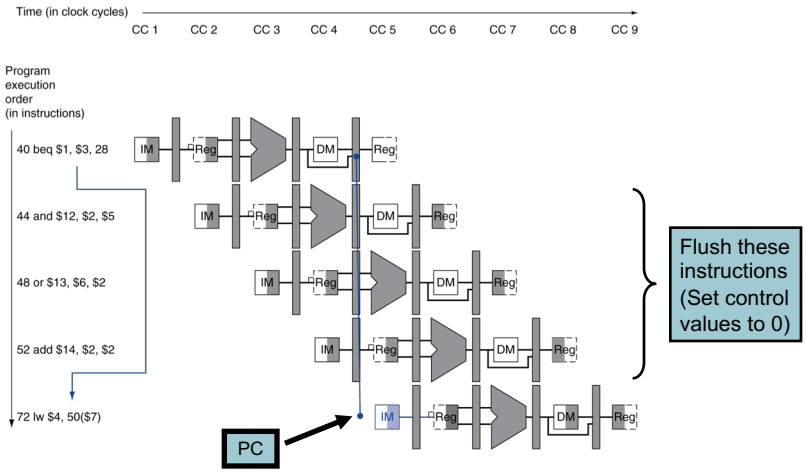


Control Hazards

- Control hazards occur when the flow of instruction addresses is not sequential
  - Unconditional branches (`j, jal, jr`)
  - Conditional branches (`beq`, `bne`)
  - Exceptions and interrupts
- Possible approaches
  - Stall (impacts CPI).
  - Move decision point as early in the pipeline as possible, thereby reducing the number of stall cycles.
  - Delay decision (requires compiler support).
  - Predict and hope for the best!
- Control hazards occur less frequently than data hazards, but there is *nothing* as effective against control hazards as forwarding is for data hazards.

Branch Hazards

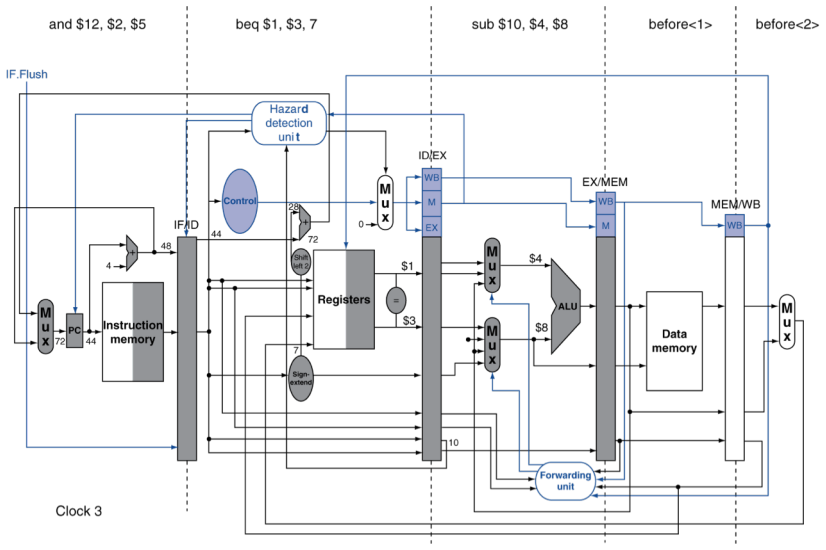
- If branch outcome determined in MEM:



Reducing the Delay of Branches

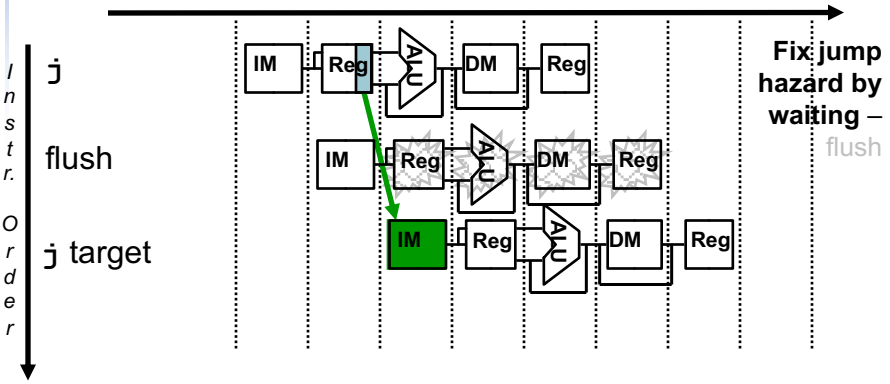
- Move the branch decision hardware back to the EX stage
  - Reduces the number of stall (flush) cycles to ***two.***
- Add hardware to compute the branch target address and evaluate the branch decision in the ID stage
  - Reduces the number of stall (flush) cycles to ***one***
    - But now need to add ***forwarding hardware*** in ID stage.
  - Computing branch target address can be done in parallel with RegFile read (done for all instructions – only used when needed).
  - Comparing the registers can't be done until after RegFile read, so comparing and updating the PC adds a mux and a compator to the ID timing path.
- For deeper pipelines, branch decision points occur even *later* in the pipeline, incurring more stalls.

Example: Branch Taken



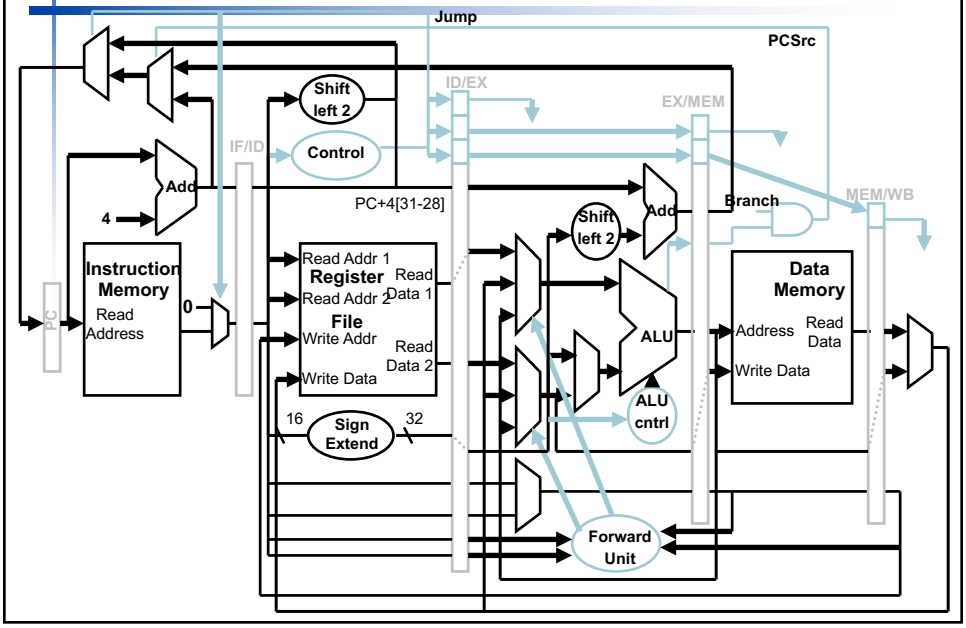
Jumps Incur One Stall

- Jumps target not decoded until ID, so one flush is needed
  - To flush, zero the instruction field of the IF/ID pipeline register (turning it into a `nop`).



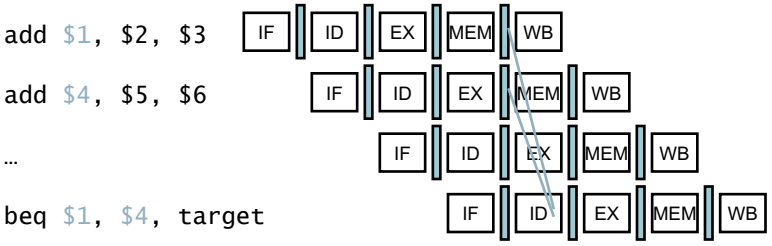
- Fortunately, jumps are very infrequent – only 3% of the SPECint instruction mix.

Supporting ID Stage Jumps



Data Hazards for Branches

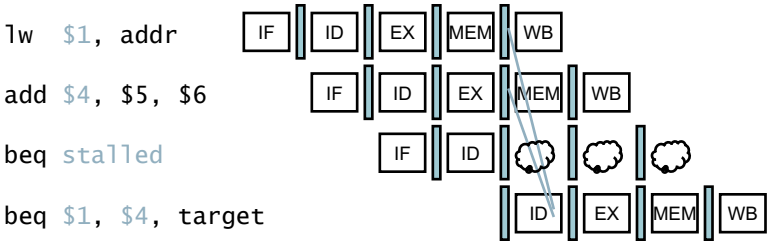
- If a comparison register is a destination of 2nd or 3rd preceding ALU instruction.



- Can resolve using forwarding.

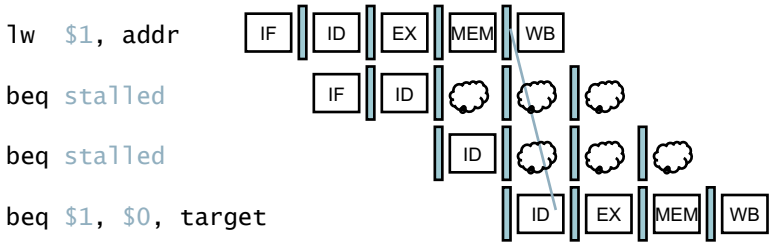
Data Hazards for Branches

- If a comparison register is a destination of preceding ALU instruction or 2nd preceding load instruction
  - Need 1 stall cycle.



Data Hazards for Branches

- If a comparison register is a destination of immediately preceding load instruction
  - Need 2 stall cycles.



Branch Prediction

- Static branch prediction
  - Based on typical branch behavior.
  - Example: loop and if-statement branches
    - Predict backward branches taken.
    - Predict forward branches not taken.
- Dynamic branch prediction
  - Hardware measures actual branch behavior
    - Records recent history of each branch instruction.
  - Assume future behavior will continue the trend
    - When wrong, stall while re-fetching, and update history.

Static Branch Prediction

- **Predict *not taken*** – always predict branches will not be taken, continue to fetch from the sequential instruction stream, only when branch *is* taken does the pipeline stall
  - If taken, flush instructions started after the branch.
  - Ensure that those flushed instructions haven't changed the machine state – automatic in the MIPS pipeline since machine state changing operations are at the tail end of the pipeline.
  - Restart the pipeline at the branch destination.

Static Branch Prediction - Continued

- **Predict taken** – predict branches will always be taken
  - Predict taken *always* incurs one stall cycle (if branch destination hardware has been moved to the ID stage) because of the need to calculate the target address.
- For deeper pipelines, branch penalty increases and a simple static prediction scheme will hurt performance. With more hardware, it is possible to try to predict branch behavior dynamically during program execution.

Dynamic Branch Prediction

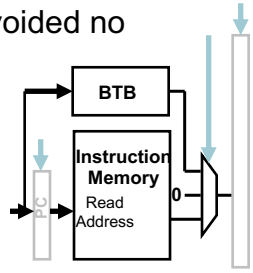
- Branch prediction buffer (aka branch history table)
  - Indexed by recent branch instruction addresses.
  - Stores outcome (taken/not taken).
- To execute a branch
  - Check table, expect the same outcome as before.
  - Start fetching from fall-through or target.
  - If wrong, flush pipeline and flip prediction.

Dynamic Branch Prediction

- A ***Branch History Table*** (**BHT**) in the IF stage addressed by the lower bits of the PC, contains bit(s) that tells whether the branch was taken the last time it was executed.
  - Prediction bit may predict incorrectly (may be a wrong prediction for this branch on this iteration, or may be from a different branch with the same low order PC bits) but this doesn't affect **correctness**, just **performance**
    - Branch decision occurs in the ID stage after determining that the fetched instruction is a branch and checking the prediction bit(s).
- A 4096-bit *Branch History Table* varies from 1% misprediction (nasa7, tomcatv) to 18% (eqntott).

Branch Target Buffer

- The BHT predicts **when** a branch is taken, but does not tell ***where*** its taken to.
  - A ***Branch Target Buffer*** **(BTB)** in the IF stage caches the branch target address, so if the branch is taken we have the address of where it branched to last time.
  - But we also need to fetch the next sequential instruction in case the branch is not taken. The prediction bit in the branch history table selects which "next" instruction will be loaded at the next clock edge
    - Or, the BTB can cache the "branch taken" instruction while the instruction memory is fetching the next sequential instruction.
- If the prediction is correct, stalls can be avoided no matter which direction the branch takes.



1-Bit Dynamic Predictor

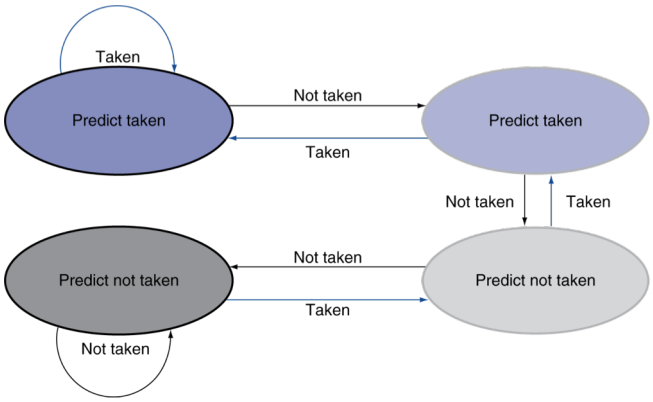
- Inner loop branches mis-predicted twice

```
outer: …
       …
inner: …
       …
       beq …, …, inner
       …
       beq …, …, outer
```

  - Mis-predict as taken on last iteration of inner loop.
  - Then mis-predict as not taken on first iteration of inner loop next time around.

2-Bit Dynamic Predictor

- Only change prediction on two successive mis-predictions.

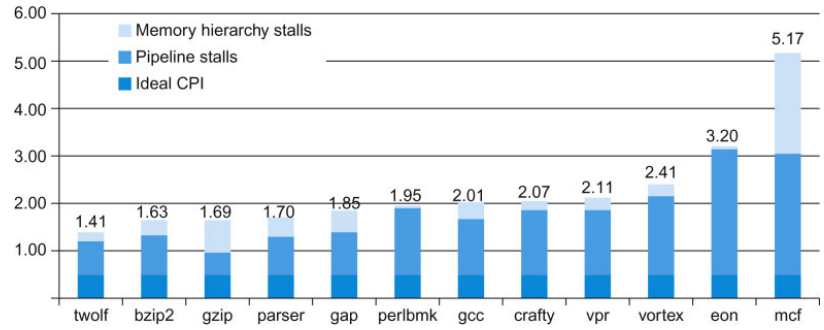


Processor Specifications

Processor	ARM A8	Intel Core i7 920
Market	Personal Mobile Device	Server, Cloud
Thermal design power	2 Watts	130 Watts
Clock rate	1 GHz	2.66 GHz
Cores/Chip	1	4
Floating point?	No	Yes
Multiple Issue?	Dynamic	Dynamic
Peak instructions/clock cycle	2	4
Pipeline Stages	14	14
Pipeline schedule	Static In-order	Dynamic Out-of-order with Speculation
Branch prediction	2-level	2-level
1st level caches / core	32 KiB I, 32 KiB D	32 KiB I, 32 KiB D
2nd level cache / core	128–1024 KiB	256 KiB
3rd level cache (shared)	–	2–8 MiB

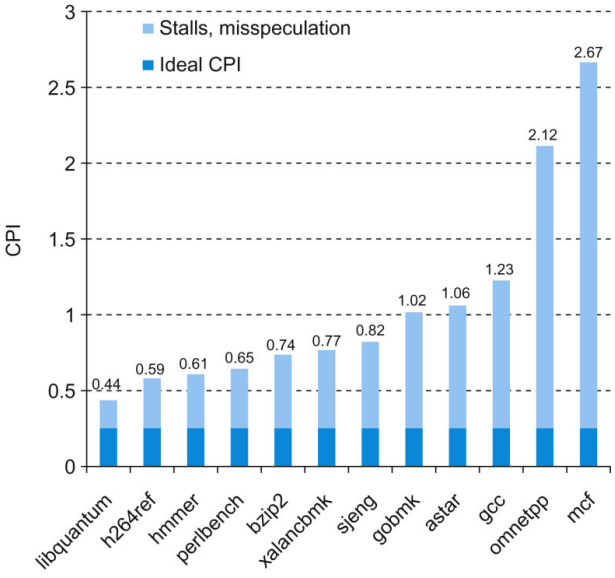
- Specification of the ARM Cortex-A8 and the Intel Core i7 920.

ARM Cortex A8 Benchmarks



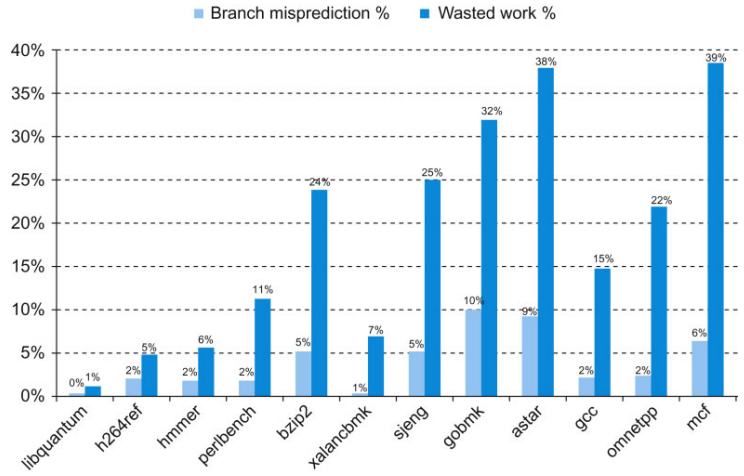
- CPI on ARM Cortex A8 for the Minnespec benchmarks, which are small versions of the SPEC2000 benchmarks. These benchmarks use much smaller inputs to reduce running time by several orders of magnitude. The smaller size significantly *underestimates* the CPI impact of the memory hierarchy (See Chapter 5).

Core I7 CPI



CPI of Intel Core i7 920 running SPEC2006 integer benchmarks.

Core I7 Branch Mis-predictions



- Percentage of branch mis-predictions and wasted work due to unfruitful speculation of Intel Core i7 920 running SPEC2006 integer benchmarks.

Summary

- All modern day processors use pipelining for performance (a CPI of 1 and a fast clock cycle).
- Pipeline clock rate limited by **slowest** pipeline stage – so designing a balanced pipeline is important.
- Must detect and resolve hazards
  - Structural hazards – resolved by designing the pipeline correctly.
  - Data hazards
    - Stall if necessary - impacts CPI.
    - Forwarding - requires hardware support.
  - Control hazards – make branch decision as early as possible
    - Stall - impacts CPI.
    - Delay decision - requires compiler support.
    - **Static** and **dynamic prediction** - requires hardware support.