Complex Pipelines & Branch Prediction

- Deeper & wider pipelines
- Out-of-order execution
- Branch prediction

Today’s handouts:
- Lecture slides
Microprocessor Performance

\[
\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \cdot \frac{\text{Cycles}}{\text{Instruction}} \cdot \frac{\text{Time}}{\text{Cycle}} \cdot \frac{\text{CPI}}{t_{CK}}
\]

• Pipelining lowers \( t_{CK} \). What about CPI?

• \( \text{CPI} = \text{CPI}_{\text{ideal}} + \text{CPI}_{\text{hazard}} \)
  – \( \text{CPI}_{\text{ideal}} \): cycles per instruction if no stall

• \( \text{CPI}_{\text{hazard}} \) contributors
  – Data hazards: long operations, cache misses
  – Control hazards: branches, exceptions
5-Stage Pipelined Processors

- Advantages
  - CPI\textsubscript{ideal} = 1 (pipelining)
  - Simple and elegant
    - Still used in ARM & MIPS processors

- Shortcomings
  - Upper performance bound is CPI=1
  - High-latency instructions not handled well
    - 1 stage for accesses to large caches or multiplier
    - Long clock cycle time
  - Unnecessary stalls due to rigid pipeline
    - If one instruction stalls, anything behind it stalls
Improving 5-Stage Pipeline Performance

- Increase clock frequency: deeper pipelines
  - Overlap more instructions

- Reduce CPI_{ideal}: wider pipelines
  - Each pipeline stage processes multiple instructions

- Reduce impact of data hazards: out-of-order execution
  - Execute each instruction as soon as its source operands are available

- Reduce impact of control hazards: branch prediction
  - Predict both direction and target of branches and jumps
Deeper Pipelines

- Break up datapath into N pipeline stages
  - Ideal $t_{CK} = 1/N$ compared to non-pipelined
  - So let’s use a large N!

- Other motivation for deep pipelines
  - Not all basic operations have the same latency
    - e.g., ADD, MUL, DIV, cache access
  - Difficult to fit them in one pipeline stage
    - $t_{CK}$ must be large enough to fit the longest one
  - Break some of them into multiple pipeline stages
    - e.g., ADD in 1 stage, data cache access in 2 stages, MUL in 3 stages...
Limits to Pipeline Depth

• Each pipeline stage introduces some overhead (O)
  – Propagation delay of pipeline registers
  – Setup and hold times
  – Clock skew
  – Inequalities in work per stage
    • Cannot break up work into stages at arbitrary points

• If original $t_{CK}$ was $T$, with N stages $t_{CK}$ is $T/N+O$
  – If $N \rightarrow \infty$, speedup = $T / (T/N+O) \rightarrow T/O$
    • Assuming that CPI stays constant
  – Eventually overhead dominates and deeper pipelines have diminishing returns
Deeper Pipelines Pros & Cons

- **Advantage:** higher clock frequency
  - The workhorse behind multi-GHz processors
  - Opteron: 11, Power5: 17, Pentium4: 22/34; Nehalem: 16

- **Cost**
  - Complexity: more bypass paths & control logic

- **Disadvantages**
  - More overlapping \(\Rightarrow\) more dependencies
    - \(\text{CPI}_{\text{hazard}}\) grows due to data and control hazards
  - Clock overhead becomes increasingly important
  - Power consumption
Wider (aka Superscalar) Pipelines

- Each stage operates on up to $W$ instructions each clock cycle
  - $CPI_{ideal} = 1/W$

- Options (from simpler to harder)
  - One integer and one floating-point instruction
  - Any $W=2$ instructions
  - Any $W=4$ instructions
  - Any $W=?$ instructions
    - What are the limits?
Wider Pipelines Pros & Cons

• Advantages: lower $\text{CPI}_{\text{ideal}} (1/W)$
  – Opteron: 3, Pentium4: 3, Nehalem: 4, Power7: 8

• Cost
  – Need wider path to instruction cache
  – Need more ALUs, register file ports, ...
  – Complexity: more bypass & stall cases to check

• Disadvantages
  – Parallel execution $\Rightarrow$ more dependencies
    • $\text{CPI}_{\text{hazard}}$ grows due to data and control hazards
Resolving Hazards

• Strategy 1: Stall. Wait for the result to be available by freezing earlier pipeline stages

• Strategy 2: Bypass. Route data to the earlier pipeline stage as soon as it is calculated

• Strategy 3: Speculate
  – Guess a value and continue executing anyway
  – When actual value is available, two cases
    • Guessed correctly → do nothing
    • Guessed incorrectly → kill & restart with correct value

• Strategy 4: Find something else to do
Out-of-Order Execution

- Consider the expression $D = 3(a - b) + 7ac$

**Sequential code**

- ld a
- ld b
- sub a-b
- mul 3(a-b)
- ld c
- mul ac
- mul 7ac
- add 3(a-b)+7ac
- st d

**Dataflow graph**

Out-of-order execution runs instructions as soon as their inputs become available.
Out-of-Order Execution Example

- If `ld b` takes a few cycles (e.g., cache miss), can execute instructions that do not depend on `b`.

**Sequential code**
```
ld a
ld b
sub a-b
mul 3(a-b)
ld c
mul ac
mul 7ac
add 3(a-b)+7ac
st d
```

**Dataflow graph**
- Completed
- Executing
- Not ready
A Modern Out-of-Order Superscalar Processor

Reconstruct dataflow graph

Execute each instruction as soon as it source operands are available

Write back results in program order

Why is this needed?
Control Flow Penalty

- Modern processors have >10 pipeline stages between next PC calculation and branch resolution!

- How much work is lost every time pipeline does not follow correct instruction flow?

  Loop length x Pipeline width

- One branch every 5-20 instructions... performance impact?
Beta Branches and Jumps

• Each instruction fetch depends on information from the preceding instruction:
  1) Is the preceding instruction a taken branch or jump?
  2) If so, what is the target address?

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Taken known?</th>
<th>Target known?</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMP</td>
<td>After Inst. Decode</td>
<td>After Inst. Decode</td>
</tr>
<tr>
<td>BEQ/BNE</td>
<td>After Reg. Read*</td>
<td>After Inst. Decode</td>
</tr>
</tbody>
</table>

* In ISAs with complex branches, taken known after execute
Resolving Hazards

• Strategy 1: Stall. Wait for the result to be available by freezing earlier pipeline stages

• Strategy 2: Bypass. Route data to the earlier pipeline stage as soon as it is calculated

• Strategy 3: Speculate
  – Guess a value and continue executing anyway
  – When actual value is available, two cases
    • Guessed correctly → do nothing
    • Guessed incorrectly → kill & restart with correct value

• Strategy 4: Find something else to do
Static Branch Prediction

• Probability a branch is taken is ~60-70%, but:

- ISA can attach preferred direction hints to branches, e.g., Motorola MC88110
  - bne0 *(preferred taken)* beq0 *(not taken)*

- Achieves ~80% accuracy
Dynamic Branch Prediction

*learning from past behavior*

- **Temporal correlation**
  - The way a branch resolves may be a good predictor of the way it will resolve at the next execution

- **Spatial correlation**
  - Several branches may resolve in a highly correlated manner (a preferred path of execution)
One-Bit Branch History Table (BHT)

- $2^m \times 1$ bit table, indexed by $m$ PC bits
- Predict taken if bit is 1, not-taken if 0
- Once branch condition is known, update table

- How many mispredictions per loop?
  - One on loop exit
  - Another one on first iteration

\[
\text{loop: } \text{SUBC}(R1, 1, R1) \\
\text{BNE}(R1, \text{loop})
\]
Two-Bit Predictor
Smith 1981

- Assume 2 bits per BHT entry
- Use saturating counter:

<table>
<thead>
<tr>
<th></th>
<th>On taken</th>
<th>On not-taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1</td>
<td>Strongly taken</td>
<td></td>
</tr>
<tr>
<td>1 0</td>
<td>Weakly taken</td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td>Weakly not-taken</td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td>Strongly not-taken</td>
<td></td>
</tr>
</tbody>
</table>

- Direction prediction changes only after two wrong predictions

- How many mispredictions per loop? 1
Exploiting Spatial Correlation
Yeh and Patt 1992

if (x[i] < 7)
    y = y + 1;
if (x[i] < 5)
    c = c - 4;

• If first condition is false, second condition is also false

• History register: Records direction of the last N branches executed
  – Predictor uses this information to predict next branch
Two-Level, Global History Predictor

- Pentium Pro uses the result from the last two branches to select one of the four sets of BHT bits
  - ~95% accurate
Limitations of Branch Direction Prediction

- If we only predict branch direction, we cannot redirect PC until after branch target is determined
  - Common cases still incur large penalties
  - Example: UltraSPARC-III pipeline
Predicting the Target Address: Branch Target Buffer (BTB)

- BTB is a cache for targets: Remembers last target PC for taken branches and jumps
  - If hit, use stored target as predicted next PC
  - If miss, use PC+4 as predicted next PC
  - After target is known, update if prediction is wrong
BTB Reduces Target-Related Penalties

- Need next PC immediately
  - PC
  - BTB

- Inst. type & branch target known
  - Fetch
  - Decode

- Branch direction & jump target known
  - RegRead
  - Execute
  - WriteBack

A misprediction is corrected as soon as it is detected
Multiple Predictors: BTB + Branch Direction

Need next PC immediately

Inst. type & branch target known

Branch direction & jump target known

BTB

Fetch

Decode

RegRead

Execute

WriteBack

Tight loop

What happens if BTB is right but branch dir predictor is wrong?
In Practice, Multiple Specialized Predictors

- Need next PC immediately
- Inst. type & branch target known
- Branch direction & jump target known

Fetch → Decode → RegRead → Execute → WriteBack

BTB

Branch dir predictor
Return addr predictor
Loop predictor

Correct mispred

Best predictors reflect program behavior
Summary

• Modern processors rely on a handful of techniques
  – Deep pipelines $\rightarrow$ Multi-GHz frequency
  – Wide (superscalar) pipelines $\rightarrow$ Multiple instructions/cycle
  – Out-of-order execution $\rightarrow$ Reduce impact of data hazards
  – Branch prediction $\rightarrow$ Reduce impact of control hazards

• Main objective is high sequential performance
  – High costs (area, power)
  – Requires high memory bandwidth and low latency
  – Simple to use (but knowing these techniques makes it easy to write high-performance programs!)

• Next lecture: Multicore processors