Lecture Topics

- Today: Pipelined Processors
  (P&H 4.5-4.10)

- Next: The Memory Hierarchy
  (P&H 5.1-5.11)

Announcements

- Calendar revised
- Milestone #5 (due 3/2)
- Milestone #6 (due 3/16)
Pipeline Summary

- Pipelining improves performance by increasing instruction throughput
  - Executes multiple instructions in parallel
  - Each instruction has the same latency

- Instruction set design affects complexity of pipeline implementation

- Subject to hazards
  - structural, data, control

Handling Hazards

- Impact on ISA design:
  - each instruction uses a particular resource at most once
  - a particular resource is only used in one pipeline stage
  - resources used for only one clock cycle

- MIPS and other RISC processors designed around pipelining
Structural Hazards

- Problem: attempt to use the same resource by two or more instructions at the same time

- Solution:
  - duplicate resources (hardware elements)
  - stall when necessary (rare)

Data Hazards

- Problem: attempt to use value before it is available in the register unit

- Solution:
  - forwarding (add bypass pathways to forward value as soon as it is available)
  - stall when necessary (such as Load-Use data hazards)
Data Hazards

Program execution order (in instructions):
- sub $2, $1, $3
- and $12, $2, $5
- or $13, $5, $3
- add $14, $2, $2
- sw $15, 100($2)

Forwarding

Use data value before it is copied into register unit.
Load-Use Data Hazard

Unavoidable: stall pipeline for one cycle

Forwarding and Hazard Detection
Control Hazards

- Problem: attempt to make branch decision before branch condition is evaluated

- Solution:
  - evaluate branch condition as soon as possible
  - stall when necessary

Control Hazards

- Baseline approach:
  - assume the branch will not be taken and continue to fetch instructions sequentially
  - if the branch is taken, nullify the instruction(s) which were fetched sequentially after the branch instruction

- The *branch penalty* is the number of clock cycles which are wasted by nullified instructions
Baseline: Decision in MEM Stage

- Branch penalty is 3 clock cycles

Example

```assembly
beq $20, $21, labelA
add $1, $10, $11
sub $2, $10, $11
xor $3, $10, $11
nor $4, $10, $11
and $5, $10, $11

labelA:
addi $6, $10, 0x40
ori $7, $10, 0x40
```
Example (2)

# Branch not taken: branch penalty is 0

```
beq  $20, $21, labelA
add  $1, $10, $11
sub  $2, $10, $11
xor  $3, $10, $11
nor  $4, $10, $11
and  $5, $10, $11
.
.
```

Example (3)

# Branch taken: branch penalty is 3

```
beq  $20, $21, labelA
add  $1, $10, $11
sub  $2, $10, $11
xor  $3, $10, $11
addi $6, $10, 0x40
ori  $7, $10, 0x40
```

Three more instr have entered the pipeline before control transferred to target address, must be nullified.
Improved: Decision in ID Stage

Compute target address:

\[ PC + 4 + (\text{imm16} \ll 2) \]
Improved: Decision in ID Stage

Compare R[rs] and R[rt]:
comparator (XOR gates)

Example (4)

# Branch not taken: branch penalty is 0

```
beq $20, $21, labelA
add $1, $10, $11
sub $2, $10, $11
xor $3, $10, $11
nor $4, $10, $11
and $5, $10, $11
.. ..
```
Example (5)

# Branch taken: branch penalty is 1

beq $20, $21, labelA
add $1, $10, $11
addi $6, $10, 0x40
ori $7, $10, 0x40

One more instr has entered the pipeline before control transferred to target address, must be nullified

MIPS Approach

- Move branch decision to ID stage
- Delayed branches: when the branch is taken, do not nullify the instruction which follows the branch instruction
- Allows work to be done in the delay slot
- Use no-op when nothing useful can be placed in the delay slot
MIPS: Scheduling Instructions

Data Hazards for Branches

- Hazard if a comparison register is a destination of 2\textsuperscript{nd} or 3\textsuperscript{rd} preceding ALU instruction

- Can resolve using forwarding

```
add $1, $2, $3
add $4, $5, $6
beq $1, $4, target
```
Data Hazards for Branches

- If a comparison register is a destination of preceding ALU instruction or 2\textsuperscript{nd} preceding load instruction, need to stall one cycle

\begin{align*}
lw & \quad \text{\$1, addr} \\
\text{ID} & \quad \text{IF} \\
\text{EX} & \quad \text{ID} \\
\text{MEM} & \quad \text{EX} \\
\text{WB} & \quad \text{MEM} \\
\text{beq} & \quad \text{stalled} \\
\text{IF} & \quad \text{ID} \\
\text{EX} & \quad \text{ID} \\
\text{MEM} & \quad \text{ID} \\
\text{WB} & \quad \text{EX} \\
\text{add} & \quad \text{\$4, \$5, \$6} \\
\text{IF} & \quad \text{ID} \\
\text{ID} & \quad \text{EX} \\
\text{EX} & \quad \text{MEM} \\
\text{MEM} & \quad \text{WB} \\
\text{beq} & \quad \text{\$1, \$4, target} \\
\text{IF} & \quad \text{ID} \\
\text{EX} & \quad \text{ID} \\
\text{MEM} & \quad \text{ID} \\
\text{WB} & \quad \text{EX} \\
\text{beq} & \quad \text{stalled} \\
\text{IF} & \quad \text{ID} \\
\text{ID} & \quad \text{EX} \\
\text{EX} & \quad \text{MEM} \\
\text{MEM} & \quad \text{WB} \\
\text{beq} & \quad \text{stalled} \\
\text{IF} & \quad \text{ID} \\
\text{ID} & \quad \text{EX} \\
\text{EX} & \quad \text{MEM} \\
\text{MEM} & \quad \text{WB} \\
\text{beq} & \quad \text{\$1, \$0, target} \\
\text{IF} & \quad \text{ID} \\
\text{EX} & \quad \text{ID} \\
\text{MEM} & \quad \text{ID} \\
\text{WB} & \quad \text{EX} \\ 
\end{align*}
Branch Prediction

- Reduce the branch penalty by predicting whether the branch will be taken or not
  - predict a particular outcome of the branch decision, fetch instructions along that control flow pathway
  - if prediction was wrong, nullify the instruction(s) which were mistakenly fetched

- MIPS predicts *branch not taken*

Branch Prediction

- General rules of thumb:
  - Frequency of branch instructions: about 15%
  - About 75% of all branches are forward branches
  - About 60% of forward branches are taken
  - About 80% of backward branches are taken
  - About 67% of all branches are taken
Dynamic Branch Prediction

- Simple approach: maintain a *branch prediction buffer* (history table)
- One bit states whether or not the branch was taken the last time that instruction was executed
- If bit says that the branch was taken, speculatively fetch instructions from that address (if prediction wrong, nullify)

Example

- Assume the branch instruction for a loop is taken 9 times in a row, and then is not taken (the loop condition is true 9 times)
  - The initial prediction is "not taken" (wrong)
  - The next 8 guesses are "taken" (right)
  - The last guess is "taken" (wrong)

- Overall accuracy: 80%
Improvement: 2 bits of history

- Prediction must be wrong twice before the prediction is changed

Summary: Branch Prediction

- Attempt to reduce the branch penalty be prediction whether or not a branch will be taken (thus allowing instructions to be fetched along the correct flow pathway)

- Simple schemes are adequate for 5-stage pipelines

- Deeper pipelines require more complex schemes (branch penalty is greater)
Evolution

- Deeper pipelines
- Superscalar (multiple pipelines)
- Multiple cores on same chip

MIPS R4400: 8 stages
Superscalar: multiple pipelines

Figure 1-3 4-Way Superscalar Pipeline

Figure 1-4 Superscalar Pipeline Architecture in the R10000