Lecture: Pipelining Extensions

- Topics: control hazards, multi-cycle instructions, pipelining equations
Summary

• For the 5-stage pipeline, bypassing can eliminate delays between the following example pairs of instructions:
  add/sub R1, R2, R3
  add/sub/lw/sw R4, R1, R5

  lw R1, 8(R2)
  sw R1, 4(R3)

• The following pairs of instructions will have intermediate stalls:
  lw R1, 8(R2)
  add/sub/lw R3, R1, R4 or sw R3, 8(R1)

  fmul F1, F2, F3
  fadd F5, F1, F4
Problem 8

• Consider this 8-stage pipeline (RR and RW take a full cycle)

  IF  DE  RR  AL  AL  DM  DM  RW

• For the following pairs of instructions, how many stalls will the 2nd instruction experience (with and without bypassing)?

  ▪ ADD R3 ← R1+R2
    ADD R5 ← R3+R4
  ▪ LD R2 ← [R1]
    ADD R4 ← R2+R3
  ▪ LD R2 ← [R1]
    SD R3 → [R2]
  ▪ LD R2 ← [R1]
    SD R2 → [R3]
Problem 8

• Consider this 8-stage pipeline (RR and RW take a full cycle)

IF DE RR AL AL DM DM RW

• For the following pairs of instructions, how many stalls will the 2nd instruction experience (with and without bypassing)?

  ▪ ADD R3 ← R1+R2
    ADD R5 ← R3+R4
    without: 5   with: 1
  ▪ LD R2 ← [R1]
    ADD R4 ← R2+R3
    without: 5   with: 3
  ▪ LD R2 ← [R1]
    SD R3 → [R2]
    without: 5   with: 3
  ▪ LD R2 ← [R1]
    SD R2 → [R3]
    without: 5   with: 1
Control Hazards

• Simple techniques to handle control hazard stalls:
  ➢ for every branch, introduce a stall cycle (note: every 6th instruction is a branch on average!)
  ➢ assume the branch is not taken and start fetching the next instruction – if the branch is taken, need hardware to cancel the effect of the wrong-path instructions
  ➢ predict the next PC and fetch that instr – if the prediction is wrong, cancel the effect of the wrong-path instructions
  ➢ fetch the next instruction (branch delay slot) and execute it anyway – if the instruction turns out to be on the correct path, useful work was done – if the instruction turns out to be on the wrong path, hopefully program state is not lost
Branch Delay Slots

(a) From before
DADD R1, R2, R3
if R2 = 0 then
Delay slot
becomes
if R2 = 0 then
DADD R1, R2, R3

(b) From target
DSUB R4, R5, R6
DADD R1, R2, R3
if R1 = 0 then
Delay slot
becomes
DSUB R4, R5, R6

(c) From fall-through
DADD R1, R2, R3
if R1 = 0 then
Delay slot
OR R7, R8, R9
DSUB R4, R5, R6
becomes
DADD R1, R2, R3
if R1 = 0 then
DSUB R4, R5, R6
OR R7, R8, R9
DSUB R4, R5, R6
Problem 1

- Consider a branch that is taken 80% of the time. On average, how many stalls are introduced for this branch for each approach below:
  - Stall fetch until branch outcome is known
  - Assume not-taken and squash if the branch is taken
  - Assume a branch delay slot
    - You can’t find anything to put in the delay slot
    - An instr before the branch is put in the delay slot
    - An instr from the taken side is put in the delay slot
    - An instr from the not-taken side is put in the slot
Problem 1

- Consider a branch that is taken 80% of the time. On average, how many stalls are introduced for this branch for each approach below:
  - Stall fetch until branch outcome is known – 1
  - Assume not-taken and squash if the branch is taken – 0.8
  - Assume a branch delay slot
    - You can’t find anything to put in the delay slot – 1
    - An instr before the branch is put in the delay slot – 0
    - An instr from the taken side is put in the slot – 0.2
    - An instr from the not-taken side is put in the slot – 0.8
Multicycle Instructions
Effects of Multicycle Instructions

- Potentially multiple writes to the register file in a cycle
- Frequent RAW hazards
- WAW hazards (WAR hazards not possible)
- Imprecise exceptions because of o-o-o instr completion

Note: Can also increase the “width” of the processor: handle multiple instructions at the same time: for example, fetch two instructions, read registers for both, execute both, etc.
Precise Exceptions

• On an exception:
  ➢ must save PC of instruction where program must resume
  ➢ all instructions after that PC that might be in the pipeline must be converted to NOPs (other instructions continue to execute and may raise exceptions of their own)
  ➢ temporary program state not in memory (in other words, registers) has to be stored in memory
  ➢ potential problems if a later instruction has already modified memory or registers

• A processor that fulfils all the above conditions is said to provide precise exceptions (useful for debugging and of course, correctness)
Dealing with these Effects

- Multiple writes to the register file: increase the number of ports, stall one of the writers during ID, stall one of the writers during WB (the stall will propagate)

- WAW hazards: detect the hazard during ID and stall the later instruction

- Imprecise exceptions: buffer the results if they complete early or save more pipeline state so that you can return to exactly the same state that you left at
Slowdowns from Stalls

• Perfect pipelining with no hazards $\rightarrow$ an instruction completes every cycle (total cycles $\sim$ num instructions) $\rightarrow$ speedup = increase in clock speed = num pipeline stages

• With hazards and stalls, some cycles (= stall time) go by during which no instruction completes, and then the stalled instruction completes

• Total cycles = number of instructions + stall cycles

• Slowdown because of stalls = $1/ (1 + \text{stall cycles per instr})$
Assume that there is a dependence where the final result of the first instruction is required before starting the second instruction.

Gap between indep instrs: $T + T_{ovh}$
Gap between dep instrs: $T + T_{ovh}$

Gap between indep instrs: $T/3 + T_{ovh}$
Gap between dep instrs: $T + 3T_{ovh}$

Gap between indep instrs: $T/6 + T_{ovh}$
Gap between dep instrs: $T + 6T_{ovh}$

14
Problem 2

- Assume an unpipelined processor where it takes 5ns to go through the circuits and 0.1ns for the latch overhead. What is the throughput for 20-stage and 40-stage pipelines? Assume that the P.O.P and P.O.C in the unpipelined processor are separated by 2ns. Assume that half the instructions do not introduce a data hazard and half the instructions depend on their preceding instruction.
Problem 2

• Assume an unpipelined processor where it takes 5ns to go through the circuits and 0.1ns for the latch overhead. What is the throughput for 1-stage, 20-stage and 50-stage pipelines? Assume that the P.O.P and P.O.C in the unpipelined processor are separated by 2ns. Assume that half the instructions do not introduce a data hazard and half the instructions depend on their preceding instruction.

• 1-stage: 1 instr every 5.1ns
• 20-stage: first instr takes 0.35ns, the second takes 2.8ns
• 50-stage: first instr takes 0.2ns, the second takes 4ns
• Throughputs: 0.20 BIPS, 0.63 BIPS, and 0.48 BIPS
ILP

- Instruction-level parallelism: overlap among instructions: pipelining or multiple instruction execution

- What determines the degree of ILP?
  - dependences: property of the program
  - hazards: property of the pipeline
Arguments against dynamic scheduling:
- requires complex structures to identify independent instructions (scoreboards, issue queue)
  - high power consumption
  - low clock speed
  - high design and verification effort
- the compiler can “easily” compute instruction latencies and dependences – complex software is always preferred to complex hardware (?).
Title

• Bullet