Computer Architecture - tutorial 2 [TUTOR COPY]

Context, Objectives and Organization

The goals of the quantitative exercises in this tutorial, which covers Lecture 4 (pipelining and pipeline hazards), are: to provide examples that demonstrate the principles and performance implications of pipelining (E1), and to work through the scheduling of loops for the 5-stage the MIPS pipeline (E2). The exercises are extracted from the H&P book 3rd edition.

E1: H&P (2e) 3.4 pg. 217 and Example pg. 137, groups of 2 – 20 min

Problem

Consider an unpipelined machine with a 10ns clock time. For a particular workload, instructions take on average 4.4 clock cycles to execute. Now consider a pipelined version of this machine. Due to clock skew the machine adds 1ns of overhead to the clock. Assume that this overhead is fixed and each pipeline stage is balanced and takes 10ns in the five stage pipeline. Plot the speedup of the pipelined machine versus the unpipelined machine as the number of pipeline stages is increased from five stages to 20 stages, considering only the impact of the pipelining overhead and assuming that the work can be evenly divided as the stages are increased (which is not generally true). Also plot the "perfect" speedup that would be obtained if there was no overhead.

Solution

CPU performance equation: $CPUTime = IC * CPI * ClockTime$

We know:

$ClockTime_{unpipe} = 10ns$

$CPI_{unpipe} = 4.4$ (from Example)

$CPI_{pipe} = 1$

Thus:

$$Speedup = \frac{ClockTime_{unpipe} * CPI_{unpipe}}{ClockTime_{pipe} * CPI_{pipe}}$$

but,

$$ClockTime_{ideal-pipe} = \frac{50ns}{NStages}$$

$$ClockTime_{pipe} = \frac{50ns}{NStages} + 1ns$$

where $50ns$ is the total time to execute one instruction, $NStages$ is the number of stages in a pipelined machine, and 1 ns is the overhead of pipelining. Then,

$$IdealSpeedup = \frac{44ns}{\frac{50ns}{NStages}}$$

$$Speedup = \frac{44ns}{\frac{50ns}{NStages} + 1ns}$$

Plotting $IdealSpeedup$ for $5 \leq NStages \leq 20$ we get for instance:

- $NStages = 5 \Rightarrow IdealSpeedup = 4.4$
- $NStages = 6 \Rightarrow IdealSpeedup = 5.28$
- $NStages = 10 \Rightarrow IdealSpeedup = 8.8$
- $NStages = 15 \Rightarrow IdealSpeedup = 13.2$
- $NStages = 20 \Rightarrow IdealSpeedup = 17.6$

Plotting $Speedup$ for $5 \leq NStages \leq 20$ we get for instance:

- $NStages = 5 \Rightarrow Speedup = 4$ (91% of ideal speedup)
- $NStages = 6 \Rightarrow Speedup = 4.7$ (89% of ideal speedup)
- $NStages = 10 \Rightarrow Speedup = 7.3$ (82% of ideal speedup)
- $NStages = 15 \Rightarrow Speedup = 10.2$ (77% of ideal speedup)
- $NStages = 20 \Rightarrow Speedup = 12.6$ (72% of ideal speedup)

This is an example of Amdahl's law in action as the portion of the execution that cannot be optimized (in this case the $1ns$ that cannot be equally split among the pipeline stages) is a limiting factor for greater speedups.

E2: groups of 3 – 30 min

Problem

Use the following code fragment:

```
loop: LD     R1,0(R2)
      DADDI  R1,R1,1
      SD     0(R2),R1
      DADDI  R2,R2,4
      DSUB   R4,R3,R2
      BNEZ   R4,loop
```

Assume that the initial value of R3 is R2+396. Throughout this exercise use the classic RISC five-stage integer pipeline in H&P (branch is resolved in the second stage) and assume all memory accesses take 1 clock cycle.

a. Show the timing of this instruction sequence for the RISC pipeline *without* any forwarding or bypassing hardware but assuming a register read and a write in the same clock cycle "forwards" through the register file. Assume that the branch is handled by flushing the pipeline. If all memory references take 1 cycle, how many cycles does this loop take to execute?
b. Show the timing of this instruction sequence for the RISC pipeline with normal forwarding and bypassing hardware. Assume that the branch is handled by predicting it as not taken. If all memory references take 1 cycle, how many cycles does this loop take to execute?
c. Assume the RISC pipeline with a single-cycle delayed branch and normal forwarding and bypassing hardware. Schedule the instructions in the loop including the branch delay slot. You may reorder instructions and modify the individual instructions operands, but do not undertake other loop transformations that change the number or opcode of the instructions in the loops. Show a pipeline timing diagram and compute the number of cycles needed to execute the entire loop.

Solution

a. No forwarding and branch optimization

Inst.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
LD	F	D	X	M	W															
DADDI		F	D	s	s	X	M	W												
SD			F	s	s	D	s	s	X	M	W									
DADDI						F	s	s	D	X	M	W								
DSUB									F	D	s	s	X	M	W					
BNEZ										F	s	s	D	s	s	X	M	W		
LD (2)													F	s	s	F	D	X	M	W

- The first DADDI must wait until LD gets to the WB stage to obtain the value of R1.
- SD must wait until the first DADDI computes the value of R1 and reaches the WB stage (no forwarding).
- DSUB must wait until the second DADDI computes the value of R2 and reaches the WB stage.
- BNEZ must wait until DSUB computes the value of R4 and reaches the WB stage.
- LD from the next iteration is re-fetched after the branch is resolved as taken (in the branch ID stage).

The loop executes 99 iterations (396/4), iterations 0 to 98. Iteration i begins in cycle $1 + (i * 15)$ (see figure). The last iteration takes 18 cycles to complete. So, the total number of cycles for the whole loop is $(98 * 15) + 18 = 1488$ cycles.

b. Forwarding and branch predict not taken

Inst.	1	2	3	4	5	6	7	8	9	10	11	12	13	14
LD	F	D	X	M	W									
DADDI		F	D	s	X	M	W							
SD			F	s	D	X	M	W						
DADDI					F	D	X	M	W					
DSUB						F	D	X	M	W				
BNEZ							F	D	s	X	M	W		
LD (2)								F	s	F	D	X	M	W

- The first ADDI still must wait until LD gets to the WB stage to obtain the value of R1, but now stall is implemented in the EX stage through the interlocking mechanism and value is forwarded directly to EX stage.
- SD now waits until the first ADDI computes the value of R1 and forwards the value from EX to MEM.
- DSUB now gets the value of R2 forwarded by the second ADDI and does not need to wait.
- BNEZ now gets the value of R4 forwarded by sub but needs to wait till the sub reaches EX stage.
- LD from the next iteration is re-fetched after the branch is resolved as mispredicted (in the branch ID stage).

Iteration i now begins in cycle $1 + (i * 9)$ and the last iteration takes only 12 cycles to complete. So, the total number of cycles for the whole loop is $(98 * 9) + 12 = 894$ cycles.

c. Simple instruction scheduling and branch delay slot

- Optimization 1: move the second DADDI to the load delay slot.
- Optimization 2: move DSUB up, to after the second DADDI.
- Optimization 3: move SD to the branch delay slot, adjusting the offset (SD -4(R2),R1).

Inst.	1	2	3	4	5	6	7	8	9	10	11
LD R1,0(R2)	F	D	X	M	W						
DADDI R2,R2,4		F	D	X	M	W					
DSUB R4,R3,R2			F	D	X	M	W				
DADDI R1,R1,1				F	D	X	M	W			
BNEZ R4,loop					F	D	X	M	W		
SD -4(R2),R1						F	D	X	M	W	
LD (2)							F	D	X	M	W

- The first DADDI does not wait as it is now separated by two extra cycles from LD.
- The BNEZ now does not wait as it is separated by one extra cycle from DSUB.
- SD now fills the branch delay slot.
- LD from the next iteration is properly fetched as the branch is resolved in time.

Iteration i now begins in cycle $1 + (i * 6)$ and the last iteration takes only 10 cycles to complete. So, the total number of cycles for the whole loop is $(98 * 6) + 10 = 598$ cycles.

Boris Grot. Thanks to Vijay Nagarajan, Nigel Topham and Marcelo Cintra.