Bachelor of Science in Computer Technology

BCT 2408 Computer Architecture-II

Lab 2

Jeff Gicharu – SCT212-0053/2021

Computer Architecture - tutorial 2

Context, Objectives and Organization

The goal of the quantitative exercise in this tutorial, which covers Lecture 4 (pipelining and pipeline hazards), is to work through the scheduling of loops for the 5-stage MIPS pipeline. The exercise is extracted from the H&P book 3rd edition.

E1: groups of 2: 50 minutes

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. Specifically, assume that: 1) branches are resolved in the second stage of the pipeline; 2) there are separate instruction and data memories; 3) 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?

Assumptions: 5-stage pipeline (IF, ID, EX, MEM, WB). No forwarding hardware (EX->EX, MEM->EX, etc.). The *only* implicit forwarding is the register file's ability to read a value in the second half of a cycle that was written in the first half (WB->ID path). Branches are resolved in ID and flush IF/ID stages if taken.

Dependency Analysis & Stalls:

- DADDI R1, R1, 1 (I2) needs R1 written by LD R1, 0(R2) (I1). I1 writes R1 at the end of its WB stage (cycle 5). I2 reads registers in its ID stage (cycle 3). I2 must stall until cycle 6 to read the value written in cycle 5. Stalls = 3 cycles.
- SD 0(R2), R1 (I3) needs R1 written by I2. I2 writes R1 at the end of its WB stage (cycle 6+3 = 9). I3 reads R1 in its ID stage (cycle 4, but stalled). I3 must stall until cycle 10.
- SD 0(R2), R1 (I3) needs R2 for address calculation, read from I1. R2 is read by I1 in cycle 2, needed by I3 in EX (cycle 10+). Available after WB of I1 (cycle 5). No stall for this.

- DADDI R2, R2, 4 (I4) needs R2 read from I1. R2 is available after WB of I1 (cycle 5). I4 reads in ID (cycle 5, but stalled). I4 stalls until cycle 6.
- DSUB R4, R3, R2 (I5) needs R2 written by I4. I4 writes R2 at end of WB (cycle 6+4=10). I5 reads in ID (cycle 6, but stalled). I5 must stall until cycle 11.
- BNEZ R4, loop (I6) needs R4 written by I5. I5 writes R4 at end of WB (cycle 11+3=14). I6 reads in ID (cycle 7, but stalled). I6 must stall until cycle 15. Branch resolves in ID (cycle 15) as *taken*.

Inst.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
LD R1	IF	ID	EX	ME	WB												
DADDI R1		IF	ID	--	--	--	EX	ME	WB								
SD R1			IF	ID	--	--	--	--	--	EX	ME	WB					
DADDI R2				IF	ID	--	EX	ME	WB								
DSUB R4					IF	ID	--	--	--	--	EX	ME	WB				
BNEZ R4							IF	ID	--	--	--	--	--	EX	!!	!!	
LD (Tgt)											IF	ID	..	..			

!! indicates flushed stage and - - indicates a stall.

Execution Cycles: The BNEZ resolves as taken in its ID stage, which occurs in cycle 15. It flushes the instruction fetch from cycle 15 and the instruction decode from cycle 14 (which was also stalled). The target instruction (LD) is fetched in cycle 16. The first LD started in cycle 1. The loop takes **15 cycles** per iteration.

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?

Assumptions: 5-stage pipeline, full forwarding (EX->EX, MEM->EX, WB->EX not needed if others exist). Branch resolved in ID, predicted not taken. Misprediction penalty is 1 cycle (flush IF/ID).

Dependency Analysis & Stalls:

- DADDI R1, R1, 1 (I2) needs R1 from LD R1, 0(R2) (I1). LD has result after MEM (cycle 4). DADDI needs it for EX (cycle 4). Requires 1 stall cycle (load-use hazard).
- SD 0(R2), R1 (I3) needs R1 from I2. I2 has result after EX (cycle 6). SD needs it for MEM (cycle 7). Forwarding EX->MEM works. OK.
- SD 0(R2), R1 (I3) needs R2 from I1 for address. I1 has R2 after ID (cycle 2). SD needs it for EX (cycle 7). Available. OK.
- DADDI R2, R2, 4 (I4) needs R2 from I1. I1 has R2 after ID (cycle 2). I4 needs it for EX (cycle 8). Available. OK.
- DSUB R4, R3, R2 (I5) needs R2 from I4. I4 has result after EX (cycle 8). I5 needs it for EX (cycle 9). Forwarding EX->EX works. OK.
- BNEZ R4, loop (I6) needs R4 from I5. I5 has result after EX (cycle 9). BNEZ needs it for ID (cycle 9). Forwarding EX->ID works. OK.
- Branch resolves in ID (cycle 9) as *taken*. Prediction was *not taken*. Misprediction occurs. Flush IF and ID stages. 1 cycle penalty.

Inst.	1	2	3	4	5	6	7	8	9	10	11	12
LD R1	IF	ID	EX	ME	WB							
DADDI R1		IF	ID	--	EX	ME	WB					
SD R1			IF	ID	EX	ME	WB					
DADDI R2				IF	ID	EX	ME	WB				
DSUB R4					IF	ID	EX	ME	WB			
BNEZ R4						IF	ID	!!	..	..		
LD (Tgt)							IF	ID	EX	..		

Execution Cycles: The BNEZ resolves as taken in its ID stage (cycle 9). Since it was predicted not taken, the instructions fetched in cycle 8 (next sequential LD) and cycle 9 (instruction after that) are flushed. The target LD is fetched in cycle 10. The first LD started in cycle 1. The loop takes **9 cycles** per iteration.

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.

You may use the following pipeline representation diagrams to develop your solution:

Assumptions: 5-stage pipeline, full forwarding, single-cycle delayed branch. Branch resolved in ID.

Scheduling: The goal is to move an instruction into the slot *after* BNEZ that can always execute, regardless of the branch outcome, without affecting correctness. We also want to reduce stalls. The main stall is the load-use between LD R1 and DADDI R1.

- Candidates to move into delay slot: LD R1 (no), DADDI R1 (no, needed by SD), SD (possible, uses R1/R2), DADDI R2 (possible, changes R2 needed by loop's LD/SD), DSUB (no, produces branch condition).

- Moving DADDI R2, R2, 4 seems best for reducing stalls if handled correctly. If moved to the delay slot, the DSUB *before* the branch must use the old R2 value. The SD *before* the branch must also use the old R2 value. The LD at the *start* of the loop must also use the old R2 value before it's incremented in the delay slot. This is feasible.

Optimized Schedule:

```
loop:  LD      R1, 0(R2)      ; Load R1
       DADDI   R1, R1, 1      ; Increment R1 (stalls 1 cycle)
       DSUB    R4, R3, R2     ; Compute R4 based on OLD R2 value
       SD      0(R2), R1      ; Store incremented R1 using OLD R2 address
       BNEZ    R4, loop       ; Branch based on R4
       DADDI   R2, R2, 4      ; Increment R2 (in delay slot) - safe
```

Inst.	1	2	3	4	5	6	7	8	9	10	11	12
LD R1	IF	ID	EX	ME	WB							
DADDI R1		IF	ID	--	EX	ME	WB					
DSUB R4			IF	ID	EX	ME	WB					
SD R1			IF	ID	EX	ME	WB					
BNEZ R4				IF	ID	EX	ME	WB	(Branch T/NT determined in ID @ C7)			
DADDI R2 (DS)					IF	ID	EX	ME	WB			
LD R1 (Tgt)						IF	ID	EX	ME	WB		

Detailed Cycle Breakdown:

1. IF(LD R1)

2. ID(LD R1) IF(DADDI R1)
3. EX(LD R1) ID(DADDI R1) IF(DSUB) - DADDI R1 stalls (Load-Use)
4. ME(LD R1) ID(DADDI R1) IF(DSUB) - Stall
5. WB(LD R1) ID(DADDI R1) IF(DSUB) - DADDI R1 resumes ID
6. EX(DADDI R1) ID(DSUB) IF(SD) - DSUB resumes ID (gets forwarded R2)
7. ME(DADDI R1) EX(DSUB) ID(SD) IF(BNEZ) - SD resumes ID (gets forwarded R1/R2). BNEZ resolves taken in ID. Fetch target LD.
8. WB(DADDI R1) ME(DSUB) EX(SD) ID(BNEZ) IF(DADDI R2 delay) - Fetch target LD
9. WB(DSUB) ME(SD) EX(BNEZ) ID(DADDI R2 delay) IF(Target LD)
10. WB(SD) ME(BNEZ) EX(DADDI R2 delay) ID(Target LD) IF(...)
11. WB(BNEZ) ME(DADDI R2 delay) EX(Target LD) ID(...) IF(...)
12. WB(DADDI R2 delay) ME(Target LD) EX(...) ID(...) IF(...)

Execution Cycles: The BNEZ resolves in cycle 7. The target LD is fetched in cycle 8 (after the delay slot instruction is fetched in cycle 7). The first LD started in cycle 1. The loop takes **7 cycles** per iteration.