

CMU 18-447 INTRODUCTION TO COMPUTER ARCHITECTURE, SPRING 2012  
HANDOUT 9/ HW 4: PIPELINING

Prof. Onur Mutlu, Instructor  
Chris Fallin, Lavanya Subramanian, Abeer Agrawal, TAs

Given: Monday, Feb 27, 2012  
Due: **Monday, Mar 19, 2012**

## 1 Delay Slots

A machine has a five-stage pipeline consisting of fetch, decode, execute, mem and write-back stages. The machine uses delay slots to handle control dependences. Jump targets, branch targets and destinations are resolved in the execute stage.

- (a) What is the number of delay slots needed to ensure correct operation?
- (b) Which instruction(s) in the assembly sequences below would you place in the delay slot(s), assuming the number of delay slots you answered for part(a)? Clearly rewrite the code with the appropriate instruction(s) in the delay slot(s).

(a) ADD R5 <- R4, R3  
OR R3 <- R1, R2  
SUB R7 <- R5, R6  
J X

Delay Slots

LW R10 <- (R7)  
ADD R6 <- R1, R2  
X:

(b) ADD R5 <- R4, R3  
OR R3 <- R1, R2  
SUB R7 <- R5, R6  
BEQ R5 <- R7, X

Delay Slots

LW R10 <- (R7)  
ADD R6 <- R1, R2  
X:

(c) ADD R2 <- R4, R3  
OR R5 <- R1, R2  
SUB R7 <- R5, R6  
BEQ R5 <- R7, X

Delay Slots

LW R10 <- (R7)  
ADD R6 <- R1, R2  
X:

- (c) Can you modify the pipeline to reduce the number of delay slots (without introducing branch prediction)? Clearly state your solution and explain why.

## 2 Hardware vs Software Interlocking

Consider two pipelined machines A and B.

**Machine I** implements interlocking in hardware. On detection of a flow dependence, it stalls the instruction in the decode stage of the pipeline (blocking fetch/decode of subsequent instructions) until all of the instruction's sources are available. Assume internal register file forwarding (an instruction writes into a register in the first half of a cycle and another instruction can access the same register in the next half of the cycle). No other data forwarding is implemented. However, there are two execute units with adders, and independent instructions can be executed in separate execution units and written back out-of-order. There is one write-back stage per execute unit, so an instruction can write-back as soon as it finishes execution.

**Machine II** does not implement interlocking in hardware. It assumes all instructions are independent and relies on the compiler to order instructions such that there is sufficient distance between dependent instructions. The compiler either moves other independent instructions between two dependent instructions, if it can find such instructions, or otherwise, inserts NOPs. Assume internal register file forwarding (an instruction writes into a register in the first half of a cycle and another instruction can access the same register in the next half of the cycle).

Both machines have the following four pipeline stages and two adders.

- Fetch (one clock cycle)
- Decode (one clock cycle)
- Execute (ADD takes 2 clock cycles. Each ADD unit is not pipelined, but an instruction can be executed if an unused execute (ADD) unit is available.)
- Write-back (one clock cycle). There is one write-back stage per execute (ADD) unit.

Consider the following 2 code segments.

### Code segment A

```
ADD R5 <- R6, R7
ADD R3 <- R5, R4
ADD R6 <- R3, R8
ADD R9 <- R6, R3
```

### Code segment B

```
ADD R3 <- R1, R2
ADD R8 <- R9, R10
ADD R4 <- R5, R6
ADD R7 <- R1, R4
ADD R12 <- R8, R2
```

- Calculate the number of cycles it takes to execute each of these two code segments on machines I and II.
- Calculate the machine code size of each of these two code segments on machines I and II, assuming a fixed-length ISA, where each instruction is encoded as 4 bytes.
- Which machine takes a smaller number of cycles to execute each code segment A and B?
- Does the machine that takes a smaller number of cycles for code segment A also take a smaller number of cycles than the other machine for code segment B? Why or why not?
- Would you say that the machine that provides a smaller number of cycles as compared to the other machine has higher performance (taking into account all components of the Iron Law of Performance)?
- Which machine incurs lower code size for each code segment A and B?
- Does the same machine incur lower code sizes for both code segments A and B? Why or why not?

### 3 Branch Prediction

Suppose we have the following loop executing on a pipelined LC-3b machine.

```

DOIT   STW   R1 <- R6, #0
        ADD   R6 <- R6, #2
        AND   R3 <- R1, R2
        BRz   EVEN
        ADD   R1 <- R1, #3
        ADD   R5 <- R5, #-1
        BRp   DOIT
EVEN   ADD   R1 <- R1, #1
        ADD   R7 <- R7, #-1
        BRp   DOIT
    
```

Assume that before the loop starts, the registers have the following decimal values stored in them:

Register	Value
R0	0
R1	0
R2	1
R3	0
R4	0
R5	5
R6	4000
R7	5

The fetch stage takes one cycle, the decode stage also takes one cycle, the execute stage takes a variable number of cycles depending on the type of instruction (see below), and the store stage takes one cycle.

All execution units (including the load/store unit) are fully pipelined and the following instructions that use these units take the indicated number of cycles:

Instruction	Number of Cycles
STW	3
ADD	3
AND	2
BR	1

Data forwarding is used wherever possible. Instructions that are dependent on the previous instructions can make use of the results produced right after the previous instruction finishes the execute stage.

The target instruction after a branch can be fetched when the BR instruction is in ST stage. For example, the execution of an ADD instruction followed by a BR would look like:

```

ADD      F | D | E1 | E2 | E3 | ST
BR       F | D | - | - | E1 | ST
TARGET  F | D
    
```

A scoreboard mechanism is used.

Answer the following questions:

- (a) How many cycles does the above loop take to execute if no branch prediction is used (the pipeline stalls on fetching a branch instruction, until it is resolved)?
- (b) How many cycles does the above loop take to execute if all branches are predicted with 100% accuracy?
- (c) How many cycles does the above loop take to execute if a static BTFN (backward taken-forward not taken) branch prediction scheme is used to predict branch directions? What is the overall branch prediction accuracy? What is the prediction accuracy for each branch?

## 4 Scoreboarding

Conventional scoreboarding as discussed in class sometimes introduces stalls when they might not actually be necessary. Consider the following code sequence:

```
ADD R2 <- R1, R3
ADD R2 <- R4, R5
```

Answer the following questions:

- (a) Why would the second instruction in this sequence stall in a conventional scoreboarding processor? (Hint: what can't the scoreboarding table track?)
- (b) Propose a scoreboard-based dependency check mechanism that addresses this issue. Show the structure of the new scoreboard table clearly, show the conditions for stalling an instruction (and in which stage this stall occurs) due to dependences, and show how the table is updated by instructions as they pass through the pipeline stages.

## 5 Interference in Two-Level Branch Predictors

Assume a two-level global predictor with a global history register and a single pattern history table shared by all branches (call this "predictor A").

- (a) We call the notion of different branches mapping to the same locations in a branch predictor "branch interference". Where do different branches interfere with each other in these structures?
- (b) Compared to a two-level global predictor with a global history register and a separate pattern history table for each branch (call this "predictor B"),
  - (i) When does predictor A yield lower prediction accuracy than predictor B? Explain. Give a concrete example. If you wish, you can write source code to demonstrate a case where predictor A has lower accuracy than predictor B.
  - (ii) Could predictor A yield higher prediction accuracy than predictor B? Explain how. Give a concrete example. If you wish, you can write source code to demonstrate this case.
  - (iii) Is there a case where branch interference in predictor structures does not impact prediction accuracy? Explain. Give a concrete example. If you wish, you can write source code to demonstrate this case as well.

## 6 Branch Prediction vs Predication

Consider two machines A and B with 17-stage pipelines with the following stages.

- Fetch (one stage)
- Decode (nine stages)
- Execute (six stages).

- Write-back (one stage).

Both machines do full data forwarding on flow dependences. Flow dependences are detected in the last stage of decode and instructions are stalled in the last stage of decode on detection of a flow dependence.

Machine A has a branch predictor that has a prediction accuracy of P%. The branch direction/target is resolved in the last stage of execute.

Machine B employs predicated execution, similar to what we saw in lecture.

- (a) Consider the following code segment executing on Machine A:

```
add r3 <- r1, r2
sub r5 <- r6, r7
beq r3, r5, X
addi r10 <- r1, 5
add r12 <- r7, r2
add r1 <- r11, r9
X: addi r15 <- r2, 10
.....
```

When converted to predicated code on machine B, it looks like this:

```
add r3 <- r1, r2
sub r5 <- r6, r7
cmp r3, r5
addi.ne r10 <- r1, 5
add.ne r12 <- r7, r2
add.ne r14 <- r11, r9
addi r15 <- r2, 10
.....
```

(Assume that the condition codes are set by the “cmp” instruction and used by each predicated “.ne” instruction. Condition codes are evaluated in the last stage of execute and can be forwarded like any other data value.)

This segment is repeated several hundreds of times in the code. The branch is taken 50% of the time and not taken 50% of the time. On an average, for what range of P would you expect machine A to have a higher instruction throughput than machine B?

- (b) Consider another code segment executing on Machine A:

```
add r3 <- r1, r2
sub r5 <- r6, r7
beq r3, r5, X
addi r10 <- r1, 5
add r12 <- r10, r2
add r14 <- r12, r9
X: addi r15 <- r14, 10
.....
```

When converted to predicated code on machine B, it looks like this:

```
add r3 <- r1, r2
sub r5 <- r6, r7
```

```
cmp r3, r5
addi.ne r10 <- r1, 5
add.ne r12 <- r10, r2
add.ne r14 <- r12, r9
addi r15 <- r14, 10
.....
```

(Assume that the condition codes are set by the “cmp” instruction and used by each predicated “.ne” instruction. Condition codes are evaluated in the last stage of execute and can be forwarded like any other data value.)

This segment is repeated several hundreds of times in the code. The branch is taken 50% of the time and not taken 50% of the time. On an average, for what range of P would you expect machine A to have a higher instruction throughput than machine B?