THE UNIVERSITY OF BRITISH COLUMBIA
Computer Science 313, Section 201
Quiz 1
February 10, 2017

Practice quiz 2, 45 minutes

Marks	
1.	/12
2.	/10
3.	/7
4.	/7
Total	/36

12

1) Some short answer questions.

a) Briefly describe the functional difference between a pipeline register and a register like %rax.

A general purpose register like %rax is visible to the assembler writer (application programmer) whereas a pipeline register is internal to the CPU and used strictly by the CPU for controlling its execution.

b) One of the branch prediction rules we discussed was one where forward jumps are not taken but backwards ones are. Why does such a rule make sense? (Think of how code for certain programming language structures might make use of this.)

Backward jumps are typically used at the bottom of a loop and since loops tend to take the jump back to the top it makes sense to favour the backward jump. If/the/elses use forward jumps and since the then part tends to be favoured over the else part it makes sense for forward jumps to favour the jump not taken option.

c) When a conditional jump instruction is executed on a CPU that solves all hazards by stalling some number of bubbles get introduced into the instruction stream. So, when a conditional move is executed, on this same CPU, how many bubbles are introduced? Explain.

A stall occurs on a conditional jump because the CPU needs to determine what instruction to execute after the jump and that can't be done until the conditional jump evaluates the status registers in the execute stage, hence the two bubbles. The conditional moves have no such constraint as the next instruction is always the instruction that immediately follows.

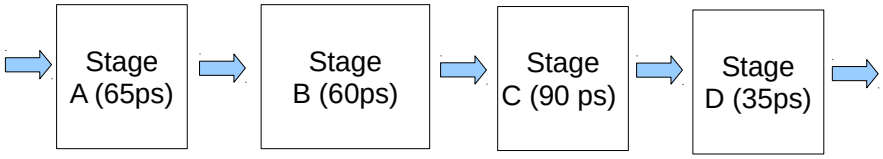
2

d) In our pipelined CPU we only care about casual dependencies and not output or alias (anti) dependencies. Carefully explain why.

In a causal dependency the output of a previous instruction is needed by a subsequent instruction and hence that instruction needs to have saved its results before subsequent instructions can use them. With anti dependencies an instruction uses a register that is subsequently modified and for the write dependency the instruction writes to a register that is subsequently overwritten. So, as long as the instructions are executed in order a subsequent instruction can't effect/change the result of executing a previous instruction so we don't need to worry about those sorts of hazards.

10

2) Consider the following picture of a the various stages in a sequential CPU. Within each stage the amount of time to execute that stage is given in pico seconds (ps) (A ps is 10^{-12} seconds, that is there are 10^{12} ps in a second and a billion is 10^9) **For all parts of this question show your work and circle your answer. Make sure your final answer is a single number and circle it.**



a) When an instruction is executed on this processor what is its latency in ps?

It is the sum of the stages 65 + 60 + 90 + 35 = 250 ps

2 marks

b) What is the throughput in billions of instructions per second?

$1/(250*10^{-12}) = 10^{12}/ / 250 = 1000 * 10^9/250 = 4 *10^9 = 4$BIPS

2 marks

c) The plan is to create a pipelined CPU by inserting stage registers between each stage. The time to load the stage registers is 10ps. How long, in ps, does a clock tick need to be for this pipelined CPU?

For each stage compute the length of time to execute that stage plus load the stage registers for that stage. In this case the longest stage is 90 + 10 = 100ps. So a clock tick needs to be 100ps

2 marks

d) For this pipelined CPU what is the latency, in ps, for an instruction?

The instruction has to go through 4 stages at 100ps a stage so the latency is 400ps

2 marks

e) What is the maximum throughput, in billions of instructions per second, of this pipelined CPU?

The calculation will be the sames as in (b) except the 250 is replaced with 100. Since 250 is 2.5 times larger than 100 the throughput rate will be 2.5 times greater or 10BIPS

$10^{12} / 100 = 1000*10^9/100 = 10 * 10^9 = 10$ BIPS 2 marks

7

3) The table below contains some Y86-64 assembly code. You are to fill in all the open (none greyed cells.) In column (a) you are to list all the data hazards for the instruction in that row in the form of the line number followed by the dependency causing the hazard. For example if the irmovq instruction of row 3 depended upon rax in line 1 and rdi in line 2 you would write it as shown in the table. If there are no dependencies write none. In column (b) write the number of stalls this instruction is subject to. If the there is more than one dependency then indicate the maximum number of stalls. In column (c) follow the same rules as for column (b) and write the maximum number of stalls for that instruction for our pipelined CPU with data forwarding.

vv

				(a) Pipelined – no forwarding hazard/dependency	(b)Number of stalls no - forwarding	(c) Pipelined – forwarding CPU Stalls
1		xorq	%rax, %rax			
2		irmovq	data, %rcx			
3		irmovq	$80, %rdi	(eg. 1 rax, 3 rdi)	14	67
4	top:	addq	%rdi, %rcx	*2-rcx, 3-rdi, **(8-rdi)***	*3*	*0*
5		mrmovq	(%rcx), %rbx	*4 - rcx*	*3 (6)*	*0*
6		irmovq	$8, %rdx	*none*	*0 (6)*	*0*
7		addq	%rbx, %rax	*5 – rbx,*	*2 (8)*	*0*
8		subq	%rdx, %rdi	*6 - rdx*	*0 (8)*	*0*
9		jg	top			
10		call	done			

In Line 4 there is a hazard on line 8 because of the jump but we won't observe it since the CPU stalls handling the branch which hides it and when there is branch prediction it will be dealt with via forwarding.

7

4) This question makes reference to the code in the previous question but concerns control hazards.

a) For the pipelined CPU with no forwarding and no branch prediction when the the jg instruction of line 9 is in the decode stage what instruction is in the fetch stage? If it is a bubble use the NOP instruction. Explain.

Since there is no forwarding, the fetch stage is being held until the jg instruction completes the execute stage. As a result the Decode stage will have a NOP

2

b) For the pipelined CPU **with branch prediction** (assume branches are taken) what instruction is in the **fetch** stage when the jg instruction is in the execute stage ? If it is a bubble use the NOP instruction. Explain.

2 mrmovq Since the CPU is predicting that branches will be taken we fetch the instructions starting at top. Since the fetch stage is 2 stages behind the execute stage we look for the 2nd instruction from top which is line 5 mrmovq

c) Observe that instruction 6 (irmovq) doesn't really need to be in the loop. Move it so that it now executes between instructions 1 and 2. For the CPU with data forwarding and branch prediction will the change result in the overall running time of the supplied code increasing, decreasing, or staying the same? (Note that overall running time means the total number of clock ticks required to execute the code.) Explain.

3 Interestingly it will take longer by 1 cycle. When we remove line 6, the line 7 dependency will result in a bubble while that instruction is held in the decode stage until instruction 7 gets to the end of the memory stage, so the number of cycles for the loop hasn't changed. However, by moving the instruction outside the loop to the indicated location an extra cycle is needed to retire that instruction. As a result it takes longer by one cycle