Course Name:	Advanced Computer Architecture	Course Code:	EE502
Program:	MS (Computer Science)	Semester:	Fall2020
Duration:	90 Minutes	Total Marks:	30
Paper Date:	24-11-2020	Weight	15
Exam Type:	Midterm II Solution	Page(s):	4

Student : Name:_____ Roll No._____

Instruction/Notes:
1. Attempt all question in the provided space. You can use rough sheets but it should not be attached.
2. If you think some information is missing, write your assumption and solve accordingly.

Question 1 [3+3+4]

a) What is the effect of pipelining on latency of a single instruction and overall program throughput? Explain for each term, either it increases or decreases, provide reasoning for your answer!

Pipelining does not effect the latency of a single instruction but it increases the overall throughput of program.

For example if we have 3 instructions with latency = 50s and they are executing without pipelining then after every 50s on instruction will be completed so throughput will be 1/50 s.

But if they are running in pipelining then

10s	10s	10s	10s	10s	10s	10s
IF	ID	Ex	M	WB		
	IF	ID	Ex	M	WB	
		IF	ID	Ex	M	WB

3

So after every 10s, an instruction is completing so throughput will be 1/10s... but latency will remain 50s

b) Why we need loop unrolling? Suppose a loop has 20 iterations and we want to unroll it by degree 3. In this scenario how many iterations of new loop will be required and how the count of 20 will be completed? Explain your answer with details.

- We need loop unrolling because it reduce our clock-cycles becz in Loop-2 extra instructions are executed in each iteration 3 so, when we do loop unrolling we write these instructions after every 2, 3, ... iterations due to which our clock cycles reduce & our program take less time for completing their execution.

- When we do loop unrolling of degree 3 & want to execute given loop for 20 iterations then we execute the unrolled loop for 6 times & after going out of this loop we make same loop for single iteration so, the remaining 2 iterations are completed by that loop & execute program almost 3-times fast

c) Why Tomasulo is better than scoreboarding? How it provides the extra functionality? What is the purpose of reorder buffers in Tomasulo approach? Explain with details!

Tomasulo is better than scoreboarding because it implements the concept of register renaming to remove the false dependencies (WAR, WAW). On the other hand scoreboarding does not provide solution for false dependencies.

Tomasulo uses Reservation Stations and Buffers to store the values and immediately provide to the Reservation stations that need the value through a common bus.

Scoreboard does not store the values of ~~registers~~ results but tomasulo does.

4

Purpose of Reorder Buffers:

Pipelining does not cause an issue when precise interupts or exceptions occur. But there is an issue in case of imprecise. Because system has to record what causes exception when.

For example:

Exception

Instruction 1 : IF ID Ex M WB

Instruction 2 : IF ID Ex M WB

Exception.

Here Instruction 1 should raise exception first but instruct 2 will raise exception first. So, to order the instructions in Tomasulo, we use reorder buffers. Instructions commits in RO when they and preceeding instruction have commited. So, reorde Buff with the help of Header, Identifies the instruction sequence and

if exception occurs, it will flush the instructions after header pointer.

Question 2 [4+2+2+2]

a. While writing in memory, contents can be brought to cache or not depending on the policy being used. Suppose we use **Allocate on write** policy, if following addresses are accessed in the given sequence then for each address tell either it would be hit or miss. Assume cache is initially empty and block size is 16 bytes.

Address	Hit/ miss?
Load 0x1200	**miss**
Store 0x1000	**miss**
Load 0x1208	**Hit**
Load 0x1012	**Hit**
Load 0x1000	**Hit**
Store 0x1216	**miss**

b. If the cache is 4 way set associative with 512 sets and 16 bytes block size, then what is the size of the cache?

Number of blocks = 512 x 4 = 2048

Size of cache = number of blocks x block size

= 2048 x 16

= 32768 bytes

c. If the total address bits are 16 then divide it into tag, index and offset bits?

offset bits = $\log_2$(block size) = $\log_2(16)$ = 4 bit

index bits = $\log_2$(number of sets) = $\log_2(512)$

= 9 bits

tag bits = 16 - (4 + 9)

= 3 bits

d. What is the total size of the memory in bytes?

Memory size = 2^{16}

= 65536 bytes

Question 3[10]

Code	Instructions

FOR: ld R1,100(R6)
ld R2,200(R6)
add R5,R2,R1
ld R4,0(R5)
addi R6,R6,4
and R4,R5,R4
st R4,0(R5)
beq R6,R7, FOR
	o ALU instructions take 1 cycle in each stage except Mem. No cycle is consumed in memory.
	o Beq takes one cycle in each stage and calculation are performed in decode stage.
	o Memory type instructions take 2 cycles in memory and one in all other stages.
	o In case of false dependencies, write back should be in order.
	o Instructions are fetched in order however decode, execution, memory and writeback can be out of order if no dependency.
	o Forwarding is implemented so values can be forwarded as soon as they are produced.

Show the execution of the above code in the following table having two slots in each stage (2-way superscalar). A slot can be filled in fetch and decode as soon as it is empty.

Cycle No.	IF		ID		EX		MEM		WB	
1	1	2								
2	3	4	1	2						
3	5	4	3		1	2				
4	6	4	5				1	2		
5	7	8	6	4	5		1	2		
6			7	8	3				1	2
7						4			3	5
8						8		4		
9								4		
10					6			8	4	
11					7				6	8
12								7		
13								7		
14									7	
15										
.										
.										
.										
.										