Team Quiz 1 - CS 250P, Fall 2019 - 4 November 2019

Q.1. beq $R1, $R2, 100
If the above condition is satisfied, in order to fetch the next instruction, to which register shall we add the offset 100?
A.1. Program Counter

—

Q.2. Why do instructions like unconditional and conditional branches introduce hazards?
A.2. These instructions disrupt the sequential flow of control. The effects of these instructions cannot be exactly determined until later in the execution. So the instruction fetch cannot continue until we take measures.

—

Q.3. How does a branch delay slot work?
A.3. Delays execution to find out which instruction is to be executed.

—

Q.4. Would you characterize the process of installing branch delay slots in code as a compiler-based approach or a hardware-based approach?
A.4. A compiler-based approach.

—

Q.5. Say you have the following sequence of instructions followed by a branch delay slot:

I1 - [a non conditional instruction]
I2 - [a branch instruction]
<a branch delay slot>

In which situation are we not able to move I1 into the branch delay slot?
A.5. If I1 is data dependent on I2.

—

Q.6. How can structural hazards be resolved?
A.6. Adding more ports to the resource (e.g. memory) being accessed.

—

Q.7. Why are WAR hazards not possible?
A.7. We want to write after read. Read will always happen earlier, as it takes place in the second cycle (Register Read).

—

Qs.8-9. Say we have a multistage pipeline executing the following instructions, in-order:

MUL R1, R2, R3
ADD R4, R5, R6
SUB R8, R9, R10

And let's say we have an exception during the MUL instruction. However the ADD and SUB instructions proceed just fine, updating registers R4, R8.
This problem is known as ____________ (**Q.8**), and it can be solved by the use of __________.(**Q.9**)
A.8. Imprecise exceptions
A.9. Reorder buffers (ROB)

—

Q.10. Explain how the solution given in **A.9.** works.
A.10. Stores the instructions in the order in which they were fetched, from which the oldest instructions are committed after those instructions have completed. Committed instructions are cleared from the ROB.

—

Qs.11-12. A 5-stage pipeline with width 1 can execute at most _____ (**Q.11**)instructions during different stages of the pipeline, at a given cycle. A 5-stage pipeline with width 2 can execute at most _____(**Q.12**)instructions during different stages of the pipeline, at a given cycle.
A.11. 5
A.12. 10

—

Q.13. Answers 11 and 12 refer to the degrees of _______ of the pipelines.
A.13. ILP

—

Qs.14-20. Identify production and consumption points of instructions in a 5 stage pipeline. (State the stage).
(**Q.14.**) P.O.C(r2) in add r1, r2, r3 **A.14.** ALU
(**Q.15.**) P.O.P(r1) in add r1, r2, r3 **A.15.** ALU
(**Q.16.**) P.O.P(r2)in lw r1, 8(r2) **A.16.** RR
(**Q.17.**) P.O.C(r2) in lw r1, 8(r2) **A.17.** ALU
(**Q.18.**) P.O.C(r3)in sw r1,8(r3) **A.18**. ALU
(**Q.19.**) P.O.C(any reg operand) in a branch instruction **A.19**. ALU
(**Q.20.**) P.O.P(r1) in sw r1,8(r3) **A.20**. DM
where, P.O.C(r) is the point of consumption of the value to be used in register r, and P.O.P(r) is the point of production of the value to be stored in register r.

—

Q.21. At least how many stalls do we need to execute the following instructions in the order given,

<FP operation storing result in r1>
st r1, 8(r3)

A.21. 2

FPALU R1	IM	RR	A1	A2	A3	A4	DM	RW	
st r1, 8(r3)		IM	RR	ALU	DM				!
st r1, 8(r3)			IM	RR	ALU	DM			!
st r1, 8(r3)				IM	RR	ALU	DM		

—

Q.22. At least how many stalls do we need to execute the following instructions in the order given,

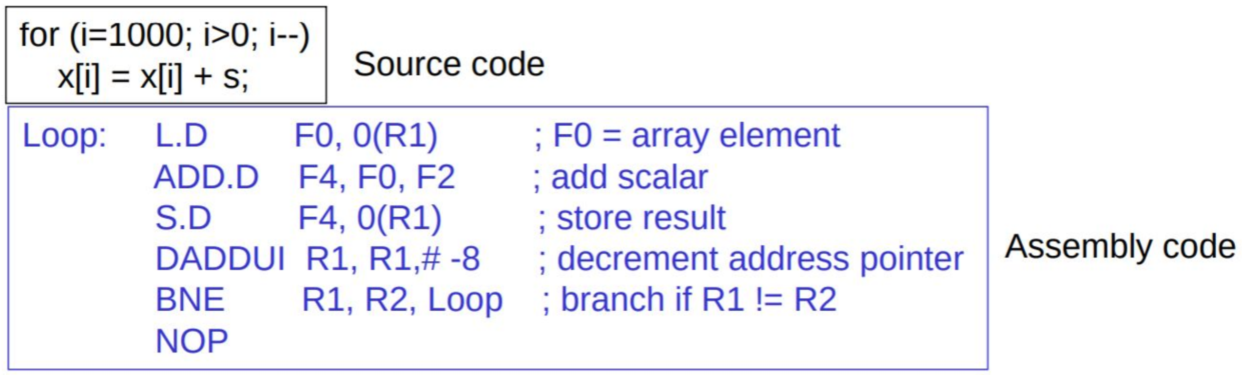
<FP operation storing result in r1>
st r3, 8(r1)

A.22. 3

FPALU R1	IM	RR	A1	A2	A3	A4	DM	R W	
st r3, 8(r1)		IM	RR	ALU	DM				!
st r3, 8(r1)			IM	RR	ALU	DM			!
st r3, 8(r1)				IM	RR	ALU	DM		!
st r3, 8(r1)					IM	RR	AL U	DM	

—

Q.23. In the assembly code generated for the C source code below, where x is an array of floating point numbers, why do we decrement R1 by 8?



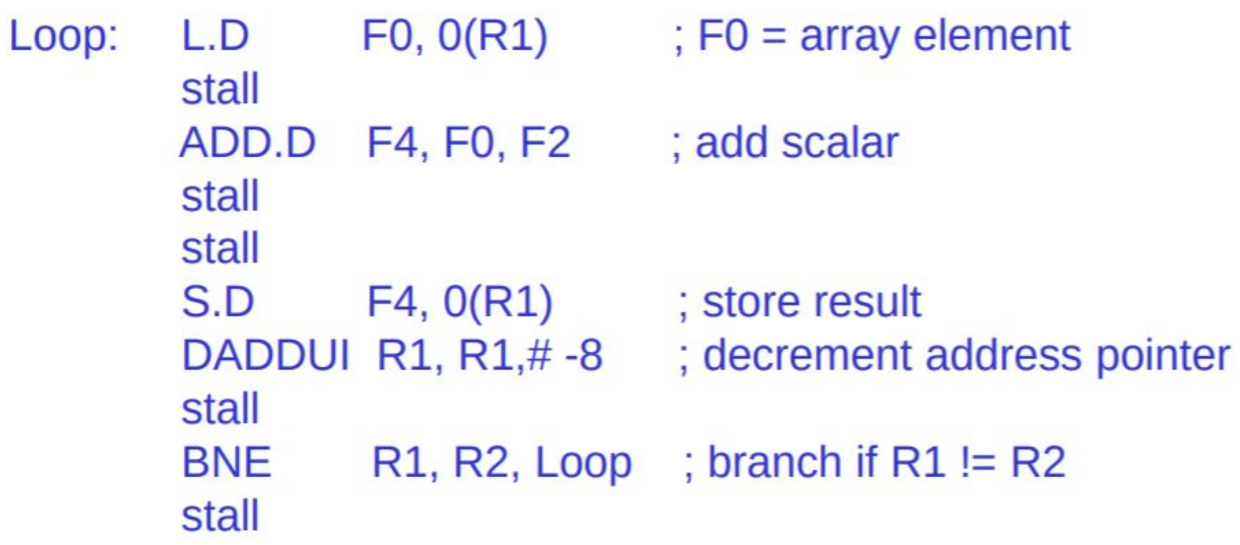
```
for (i=1000; i>0; i--)
    x[i] = x[i] + s;
```

```
Loop:   L.D       F0, 0(R1)        ; F0 = array element
        ADD.D     F4, F0, F2       ; add scalar
        S.D       F4, 0(R1)        ; store result
        DADDUI    R1, R1,# -8      ; decrement address pointer
        BNE       R1, R2, Loop     ; branch if R1 != R2
        NOP
```

A.23. A floating point array element takes 8 bytes.

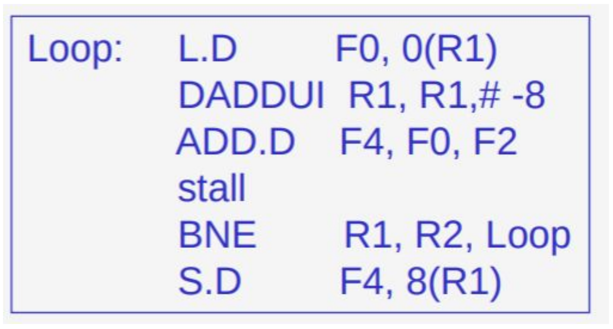
—

Q.24-25. For the above assembly code, the compiler generates the following in order to schedule the instructions in a way that avoids hazards:



```
Loop:   L.D      F0, 0(R1)      ; F0 = array element
        stall
        ADD.D    F4, F0, F2     ; add scalar
        stall
        stall
        S.D      F4, 0(R1)      ; store result
        DADDUI   R1, R1,# -8    ; decrement address pointer
        stall
        BNE      R1, R2, Loop   ; branch if R1 != R2
        stall
```

which is optimized to generate:



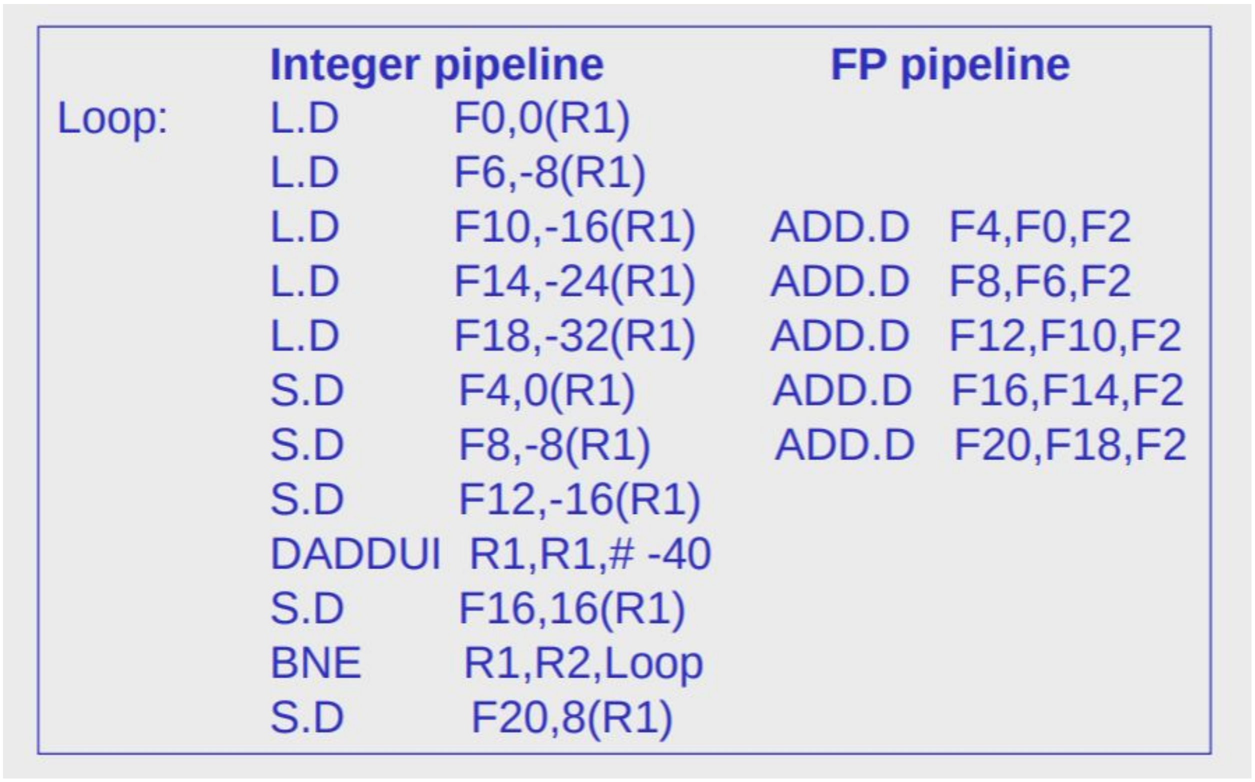
```
Loop:   L.D      F0, 0(R1)
        DADDUI   R1, R1,# -8
        ADD.D    F4, F0, F2
        stall
        BNE      R1, R2, Loop
        S.D      F4, 8(R1)
```

(Q.24.) Why is it safe to move the DADDUI instruction above the SD instruction?
A.24. We decrement by first, and then increment the offset by 8 during store, resulting in the same effect.

(Q.25.) Why is it safe to move up the BNE instruction to where it has been placed?
A.25. The store instruction was originally inside the loop. Now, although it is placed after the loop in the code, since it is immediately after the branch instruction, it is always executed. The instruction after the branch is always fetched.

—

Qs.26-28. The code in Q.23 is executed via a multi-cycle pipeline (which has an Integer pipeline and an adjacent FP pipeline) to further reduce the cycle time per iteration. The instructions are provided to the pipeline as follows, where each line of code represents operation(s) in a cycle. As can be seen the loop has been unrolled:



	Integer pipeline		FP pipeline	
Loop:	L.D	F0,0(R1)		
	L.D	F6,-8(R1)		
	L.D	F10,-16(R1)	ADD.D	F4,F0,F2
	L.D	F14,-24(R1)	ADD.D	F8,F6,F2
	L.D	F18,-32(R1)	ADD.D	F12,F10,F2
	S.D	F4,0(R1)	ADD.D	F16,F14,F2
	S.D	F8,-8(R1)	ADD.D	F20,F18,F2
	S.D	F12,-16(R1)		
	DADDUI	R1,R1,# -40		
	S.D	F16,16(R1)		
	BNE	R1,R2,Loop		
	S.D	F20,8(R1)		

(**Q.26**) How many iterations have been unrolled?
A.26. 5

(**Q.27**) What is the width of the pipeline?
A.27. 2

(Q.28) What would be the effect on the code if we unrolled the loop 4 times.
A.28. We need a 2-cycle gap between ADD.D F4, and SD F4. Thus by unrolling the loop 4 times, we would have had to put in an extract stall cycle to avoid a RAW hazard. The extra unrolling of the loop ensures we are doing useful work by doing loads.

—