[Question 1] [20 Points]

For the code sequence shown below:

```
loop:
    l.d    F2, 0(R4)
    mul.d  F10, F10, F2
    daddui R4, R4, -8
    bne    R4, R1, loop // R1 holds the address of the
                           last value to be operated on
```

a. Show loop unrolling so that there are four copies of the loop body. Assume ~~$f5, $f9~~ (Correction: R1, R4) (that is, the size of the array) are initially a multiple of 32, which means that the number of loop iterations is a multiple of 4. Eliminate any obviously redundant computations and do not reuse any of the registers.

```
loop: l.d    F2,0(R4)
      mul.d  F11,F11,F2
      l.d    F3,-8(R4)
      mul.d  F12,F12,F3
      l.d    F4,-16(R4)
      mul.d  F13,F13,F4
      l.d    F5,-24(R4)
      mul.d  F14,F14,F5
      daddui R4,R4,-32
      bne    R4,R1,loop
      mul.d  F8,F11,F12
      mul.d  F9,F13,F14
      mul.d  F10,F8,F9
```

b. Compute the number cycles needed for 4 iterations:

Instruction Producing Result	Instruction Using Result	Latency in Cycles
FP ALU OP	Store double	4
FP ALU OP	FP ALU OP	3
ALU OP	BNE	2
Load double	FP ALU OP	1
Load double	Store double	0

```
loop: l.d    F2,0(R4)
      stall
      mul.d  F11,F11,F2
      l.d    F3,-8(R4)
      stall
      mul.d  F12,F12,F3
      l.d    F4,-16(R4)
      stall
      mul.d  F13,F13,F4
      l.d    F5,-24(R4)
      stall
      mul.d  F14,F14,F5
      daddui R4,R4,-32
      stall
      stall
      bne    R4,R1,loop
      mul.d  F8,F11,F12
      mul.d  F9,F13,F14
      stall
      stall
      stall
      mul.d  F10,F8,F9
```

22 cycles

[Question 2] [10 Points]

In the following instruction sequence, find all hazards. Rename the registers to eliminate the anti-dependencies and output dependences:

```
div.s  F6, F2 ,F4
sub.s  F14,F12,F6
add.s  F12,F2, F8
mult.s F6, F8, F12
```

Red=WAW Hazard

Blue/**Green**=WAR Hazard

Orange=RAW Hazard

```
div.s  F16,F2,F4
sub.s  F14,F12,F16
add.s  F10,F2,F8
mult.s F6,F8,F10
```

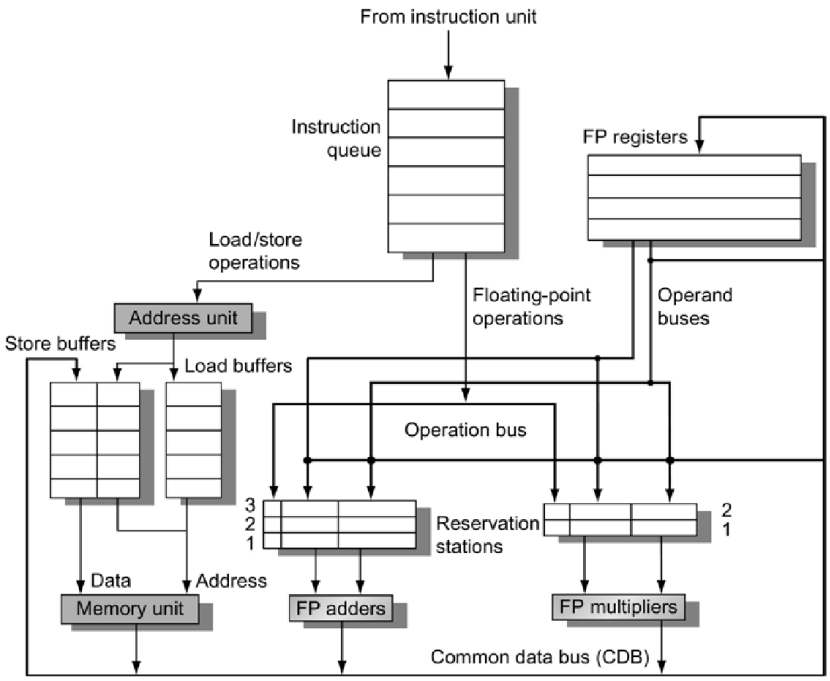
[Question 3] [20 Points]

Consider the following instruction sequence (floating point) on a processor (shown below) which uses Tomasulo's algorithm to dynamically schedule instructions (dual issue per cycle - no speculation):

```
x: ADD R1,R2,R4
y: MUL R0,R1,R2
z: ADD R2,R4,R0
```

Initial values of the registers:

Register	Initial Value
R0	4
R1	3
R2	2
R4	6



The processor has the following non-pipelined execution units:

- A 1 cycle, FP add unit
- A 2 cycle, FP multiply unit

Assume instructions can begin to execute in the same cycle as soon as its dispatched and resides in Reservation Stations. Trace the execution by showing Reservation Stations and FP Registers at the end of cycles 1, 2, 3, and 4:

Cycle 1:

	Reservation Station			
	Tag1	S1	Tag2	S2
x 1	0	2	0	6
2				
3				
	Adder			

	Reservation Station			
	Tag1	S1	Tag2	S2
y 4	1	-	0	2
5				
	Multiply/Divide			

	FP Registers		
	Busy	Tag	Data
R0	Yes	4	4
R1	Yes	1	3
R2			2
R4			6

Cycle 2:

	Reservation Station			
	Tag1	S1	Tag2	S2
1				
z 2	0	6	4	-
3				
	Adder			

	Reservation Station			
	Tag1	S1	Tag2	S2
y 4	0	8	0	2
5				
	Multiply/Divide			

	FP Registers		
	Busy	Tag	Data
R0	Yes	4	4
R1			8
R2	Yes	2	2
R4			6

Cycle 3:

	Reservation Station			
	Tag1	S1	Tag2	S2
1				
z 2	0	6	4	-
3				
	Adder			

	Reservation Station			
	Tag1	S1	Tag2	S2
y 4	0	8	0	2
5				
	Multiply/Divide			

	FP Registers		
	Busy	Tag	Data
R0	Yes	4	4
R1			8
R2	Yes	2	2
R4			6

Cycle 4:

	Reservation Station			
	Tag1	S1	Tag2	S2
1				
z 2	0	6	0	16
3				
	Adder			

	Reservation Station			
	Tag1	S1	Tag2	S2
4				
5				
	Multiply/Divide			

	FP Registers		
	Busy	Tag	Data
R0			16
R1			3
R2	Yes	2	2
R4			6