Prosect 2 help.
Check Branches
opis load ⇒ in DV.

Name Derek Hernandez
netID djh119
(email not long Axxxxx number)

Assignment 3 CS 3339 – Spring 2019
Due: Friday 3/1 11:55pm
40 points (late until noon 3/2 -10 points)

All submissions must be written in very neat handwriting and scanned (or typed) and submitted in PDF format to TRACS with the filename of Ax_netID.pdf. You may submit as many times as you like prior to the deadline; only the most recent submittal will be graded. All assignments must be submitted individually and reflect your own work; however, you are encouraged to work in groups and discuss the problems with your classmates.

1) [4 points] Write a solution to the following truth table using minterms. Minterms take the form of ABC or A'BC, ABC' etc and when combined with logical *or* functions give the correct output. The first slide of this presentation gives some insight, only the first slide the remainder covers optimization techniques. http://kth.s3-website-eu-west-1.amazonaws.com/ie1204_5/slides/eng/F4minimering_eng.pdf

$F(1,2,3) = \bar{X}_1 X_0 + X_1 \bar{X}_0 + X_1 X_0$

A	B	C	F(A,B,C)
0	0	0	**0**
0	0	1	**0**
0	1	0	**1**
0	1	1	**0**
1	0	0	**0**
1	0	1	**1**
1	1	0	**1**
1	1	1	**0**

→ $\bar{A}B\bar{C}$
→ $A\bar{B}C$
→ $AB\bar{C}$

F(A,B,C) = $\bar{A}B\bar{C} + A\bar{B}C + AB\bar{C}$ → (1) $B\bar{C} + A\bar{B}C$ Simplified

$\bar{A}B\bar{C} + A\bar{B}C + AB\bar{C}$

or'ed

$(\bar{A} + A)B\bar{C} + A\bar{B}C$

1 or 0

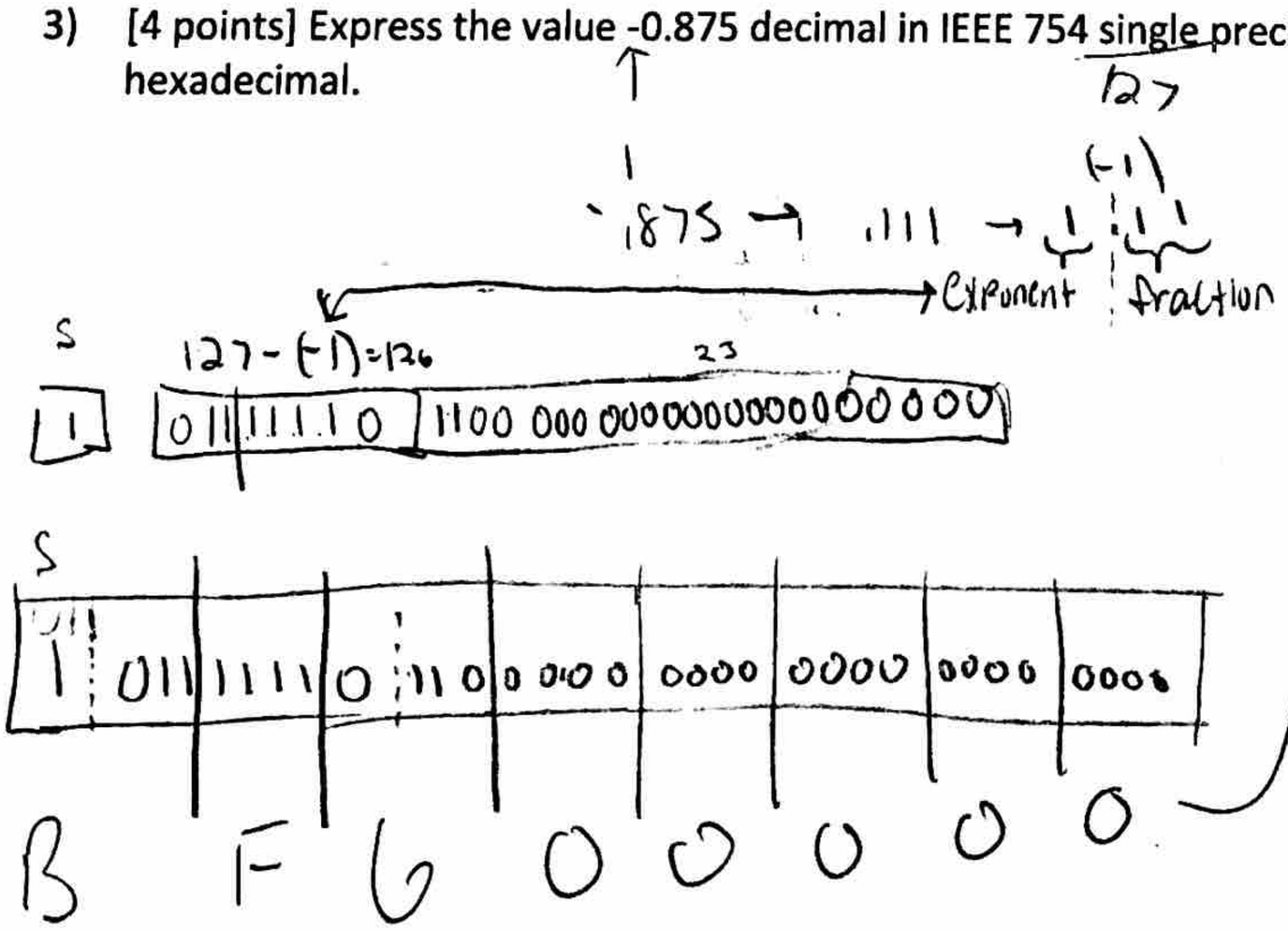
=(1) $B\bar{C} + A\bar{B}C$

2) [4 points] All other things being equal which is faster integer addition or floating-point addition? Why?

Integer addition is the fastest. Integer addition is much more simpler Compared to Floating-Point addition.

Integer addition doesn't need to: Align decimal points, or Add Significands, or Normalize result, nor round.

3) [4 points] Express the value -0.875 decimal in IEEE 754 single precision binary format and convert to hexadecimal.



Hex = 0xBF600000

Binary = 10111111 0110 0000 0000 0000 0000 0000

decimal = -.875

4) [4 points] Correct the following emulation code for the MIPS *beq* instruction.

no rd

```
/* beq */ if (regFile[rs] = regFile[rd]) { pc = 4 + uimm; }
```

```
if (regFile[rs] = regFile[rt]) {
    pc = pc + (simm << 2);
}
```

5) [8 points] Complete the code to properly set all (do not assume safe defaults) control signal values as in Project 2.

```
case 0x2b:
D(cout << "sw " << regNames[rt] << ", " << dec << simm << "(" << regNames[rs] << ")");
opIsLoad = false; opIsStore = true;
writeDest = false; destReg = false;
aluOp = ADD;
aluSrc1 = regFile[rs];
aluSrc2 = Simm;
storeData = regFile[rt];
break;
```

6) [4 points] The MIPS pipelined implementation has no structural hazards. What are the other 2 hazards that exist and what is the primary technique used to improve performance versus simply inserting bubbles.

Single Memory

Data Hazard : Instead of Bubble, Use forwarding to immediately use when computed.

Control Hazard : Branch Prediction, Predict branch operations likely to occur.

7) [4 points] If a pipelined processor has implemented all forwarding paths but no hazard detection, describe a case where the processor could the processor produce erroneous results.

- No Stall
- load-use

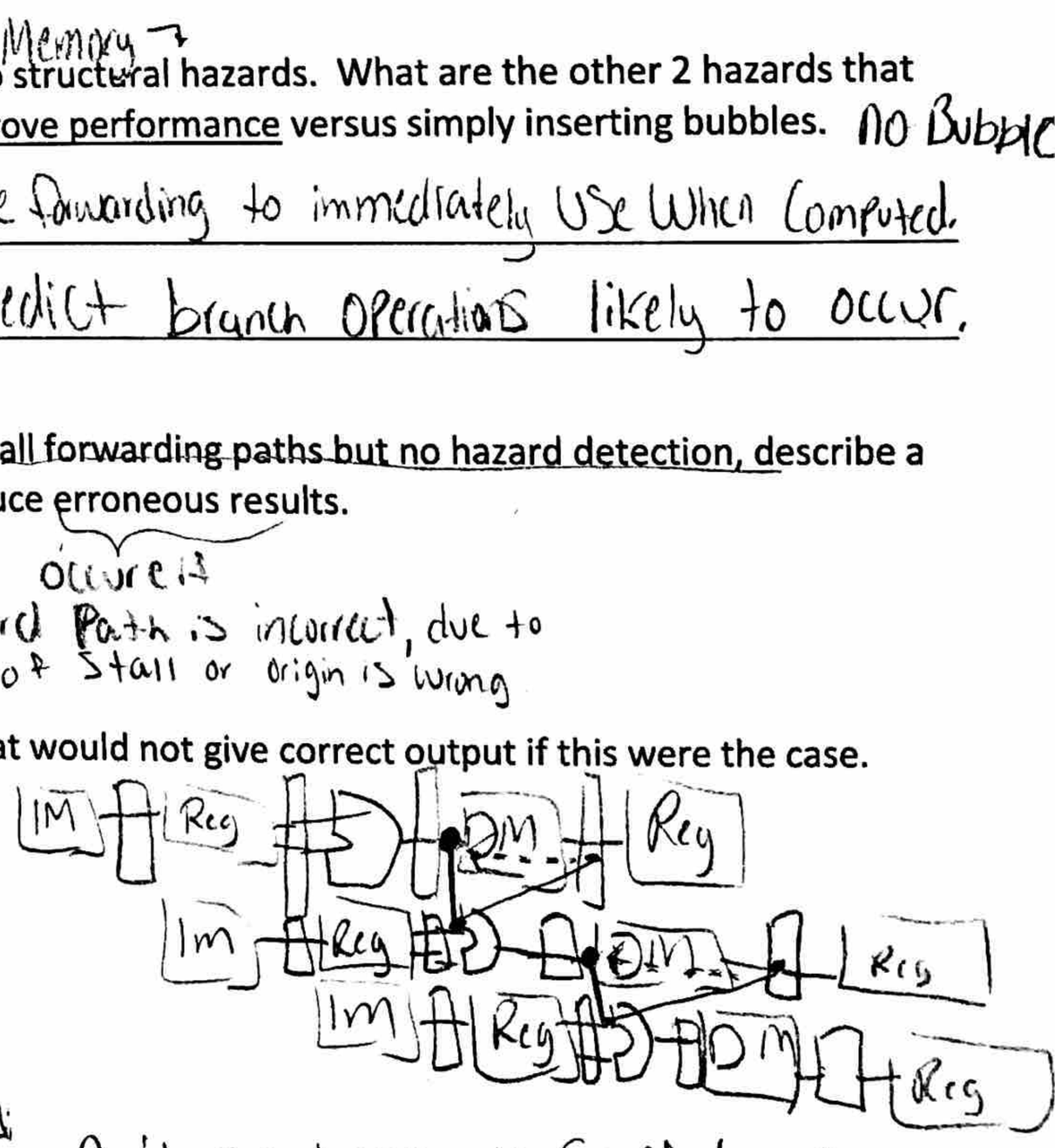
Error will occure if a forward Path is incorrect, due to a lack of Stall or origin is wrong

Write a short sequence of assembly instructions that would not give correct output if this were the case.

from Slide Lectures

```
lw $2, 20($1)
and $4, $2, $5
or $8, $4, $6
```

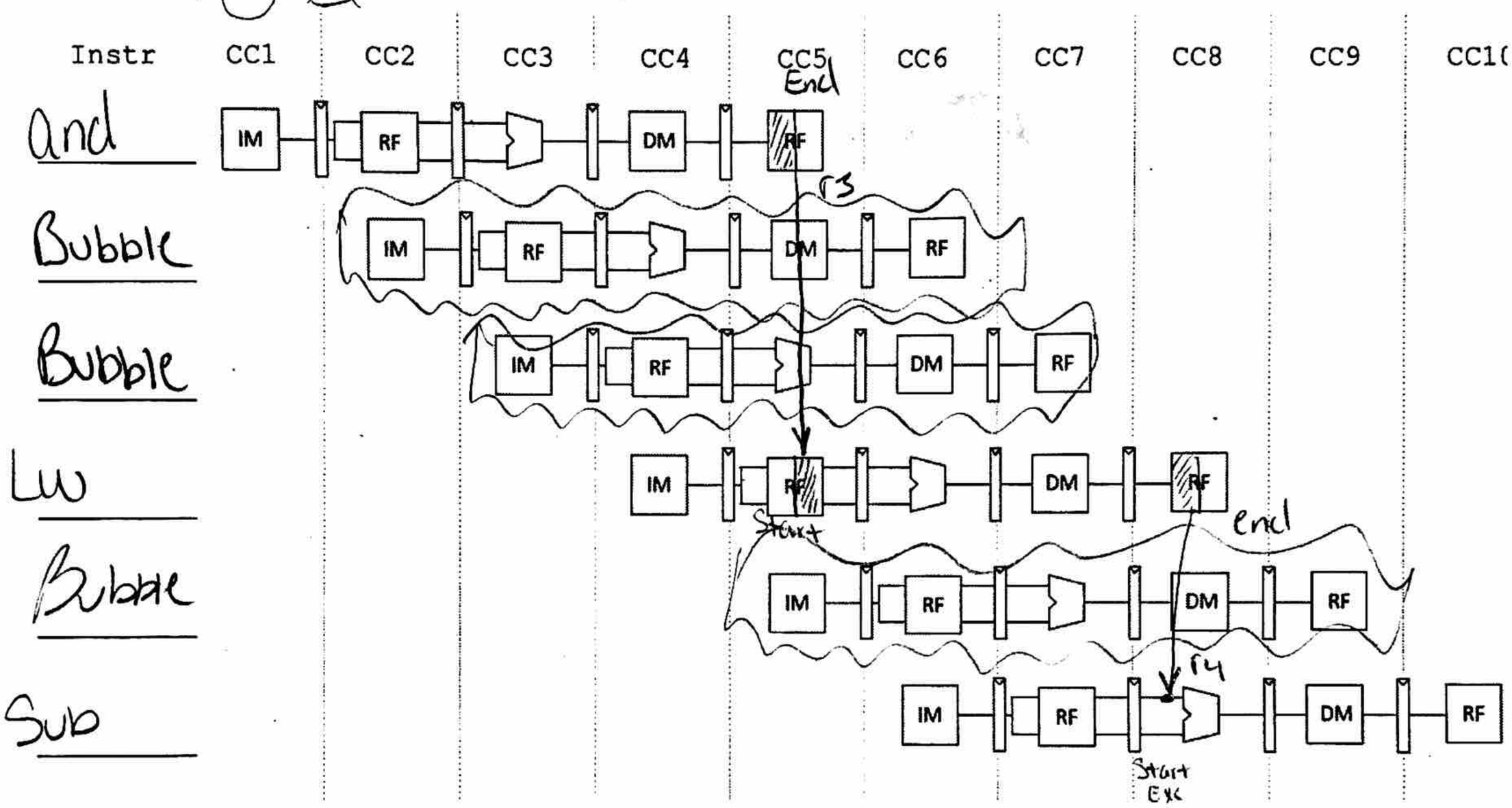
* all of these need a bubble in between, or the forwarding Path will be incorrect



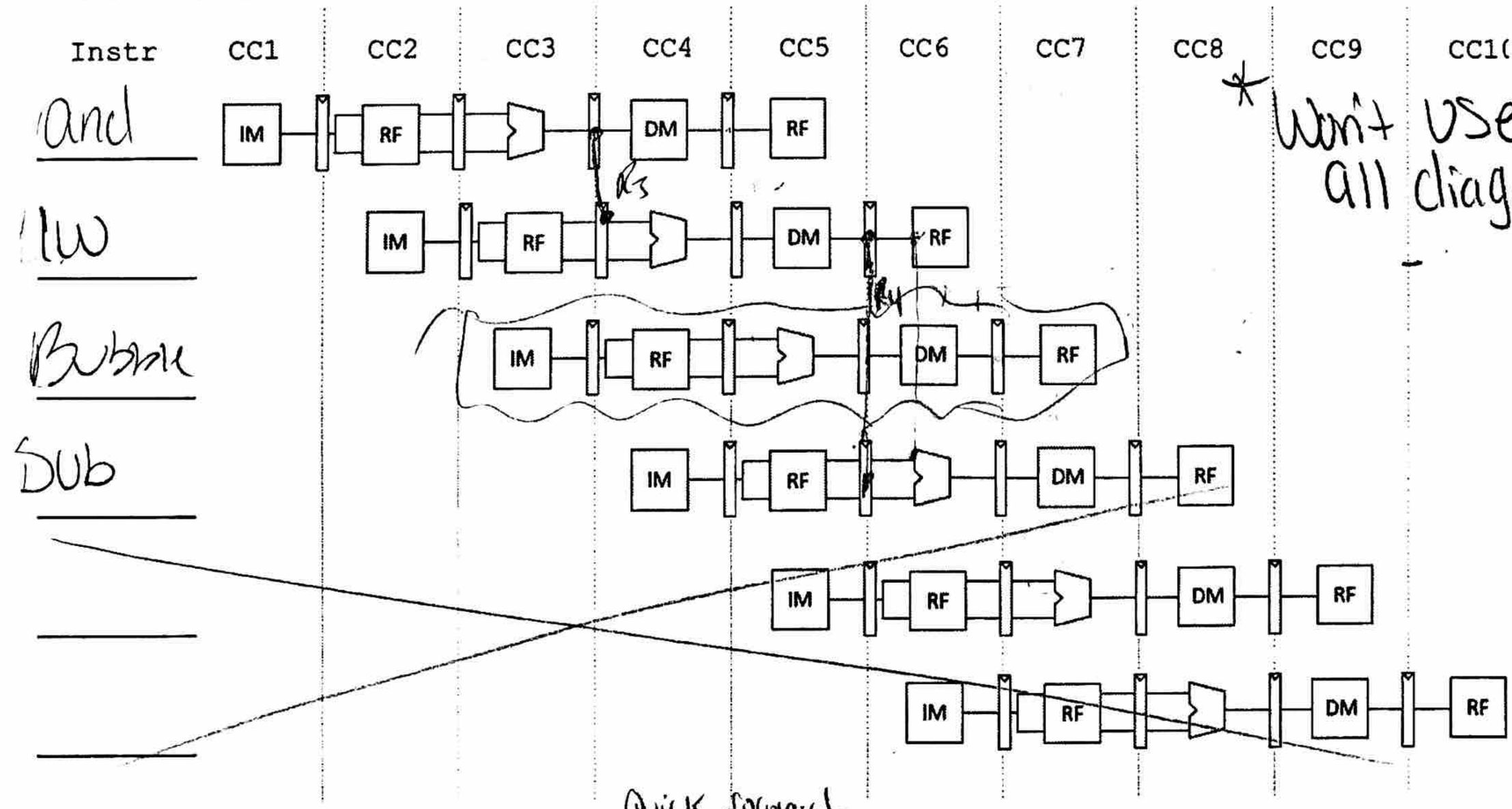
Can't go backwards, so if the forward Path is fixed before ready a Strang result

8) [4 points] For the following instruction sequence show the bubbles required to ensure correct output without reordering and without forwarding paths. Draw arrows representing the flow of r3 and r4. [See figures 4.53 and 4.58]

```
and r3, r1, r2
lw r4, 10(r3)
sub r5, r4, r1
```



[4 points] Repeat the exercise, but this time use all possible forwarding paths to speed up execution.



Quick forward

and Exc ▯ Dm R type Im-Exc.
lw Dm ▯ Rf I type Im-Rf
Sub Exc ▯ DM R type Im-Exc