CS232 Midterm Exam 2
April 14, 2003

Name: _____

- This exam has 7 pages including the pipelined datapath diagram on the last page, which you are free to tear off.
- You have 50 minutes, so budget your time carefully!
- No written references or calculators are allowed.
- To make sure you receive credit, please write clearly and show your work.
- We will not answer questions regarding course material.

Question	Maximum	Your Score
1	35	
2	30	
3	35	
Total	100	

Question 1: Multicycle CPU implementation (35 points)

Consider extending the MIPS architecture with the instruction below, which loads two consecutive words of data from memory and stores them into two destination registers.

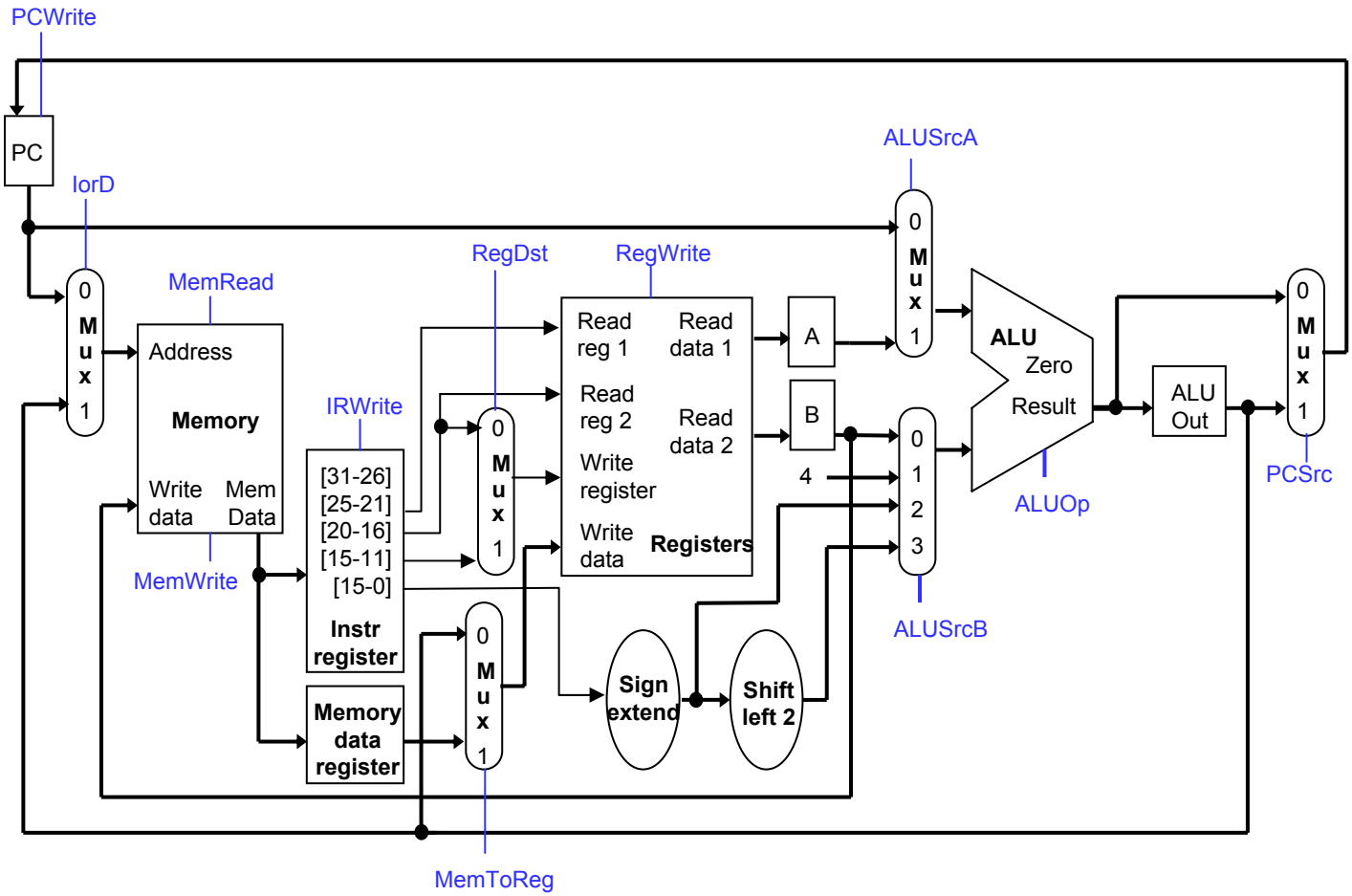
ld rt, rd, rs # rt = Mem[rs]; rd = Mem[rs + 4]

This will use the same format as R-type instructions, shown here for reference (shamt and func are not used).

Field	op	rs	rt	rd	shamt	func
Bits	31-26	25-21	20-16	15-11	10-6	5-0

Part (a)

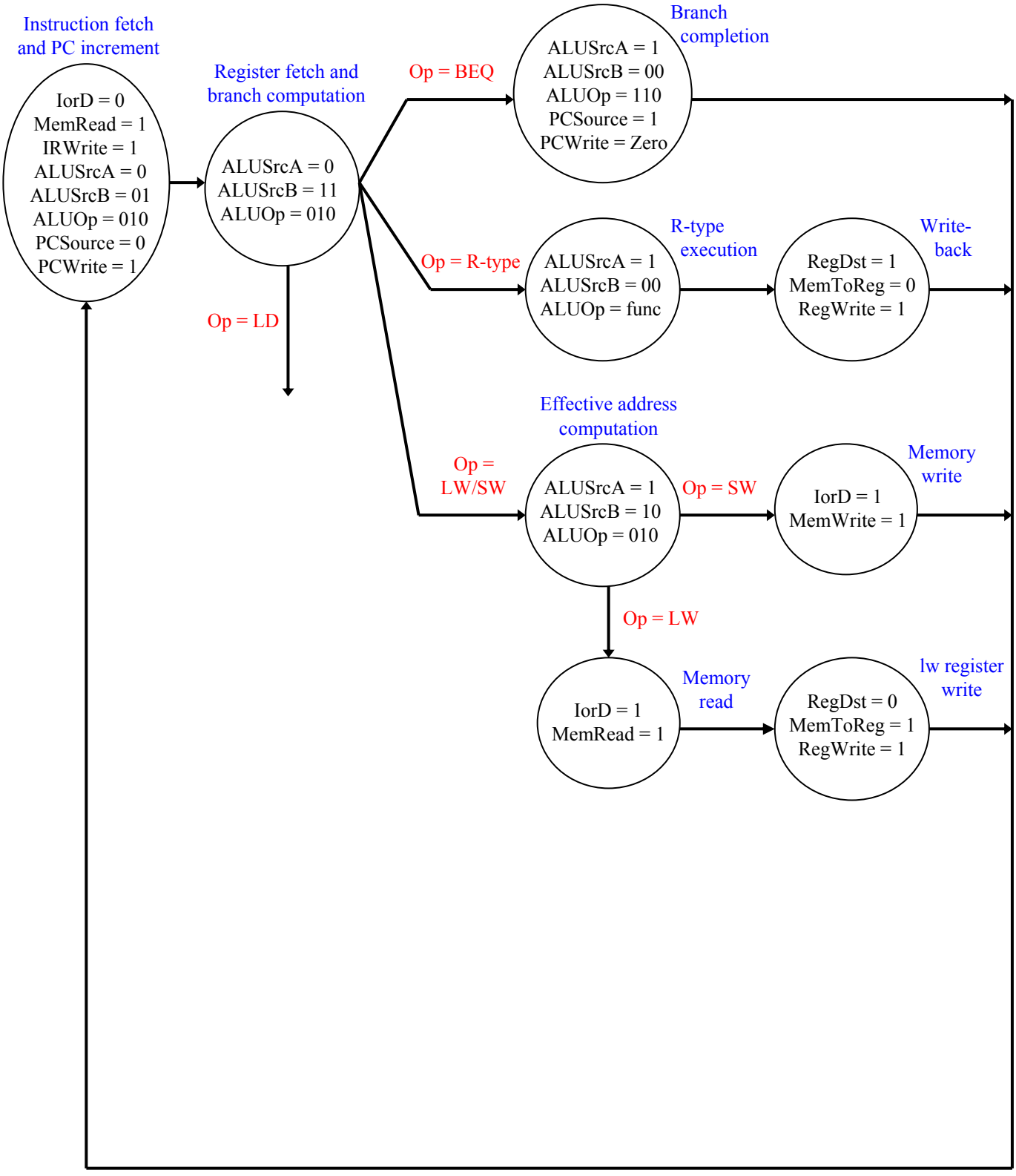
The multicycle datapath from lecture appears below. Show what changes are needed to support *ld*. You should not modify the main functional units themselves (the memory, register file and ALU), but you can make any other changes or additions necessary. Try to keep your diagram neat! (10 points)



Question 1 continued

Part (b)

Complete this finite state machine diagram for the *ld* instruction. Control values not shown in each stage are assumed to be 0. Remember to account for any control signals that you added or modified in the previous part of the question! (25 points)



Question 2: Pipelining performance (30 points)

Here is a short MIPS assembly language loop. (This is a simpler version of a very common operation in scientific applications.) Assume that we run this code on the pipelined datapath shown on the last page.

```
Elvis:
     lw     $t0, 0($a1)
     mul    $t0, $t0, $a2
     lw     $t1, 0($a0)
     add    $t0, $t0, $t1
     sw     $t0, 0($a0)
     sub    $a3, $a3, 1
     addi   $a0, $a0, 4
     addi   $a1, $a1, 4
     bne    $a3, $0, Elvis
```

Part (a)
Find the number of clock cycles needed to execute this code, accounting for all possible stalls and flushes. Assume that $a3 is initially set to 100. (20 points)

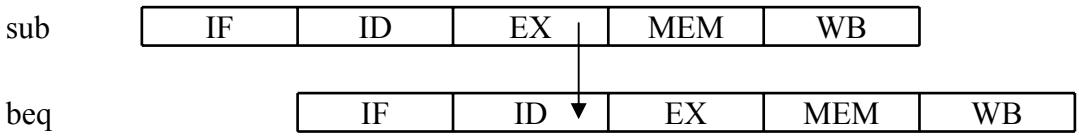
Part (b)
Show how you can re-arrange the instructions in the code above to eliminate as many stall cycles as possible. You do *not* need to reduce the number of flushes. (10 points)

Question 3: Forwarding (35 points)

Here is a sequence of two instructions, with a dependency between them.

```
sub   $s1, $s2, $s3
beq   $s1, $0, Pooh
```

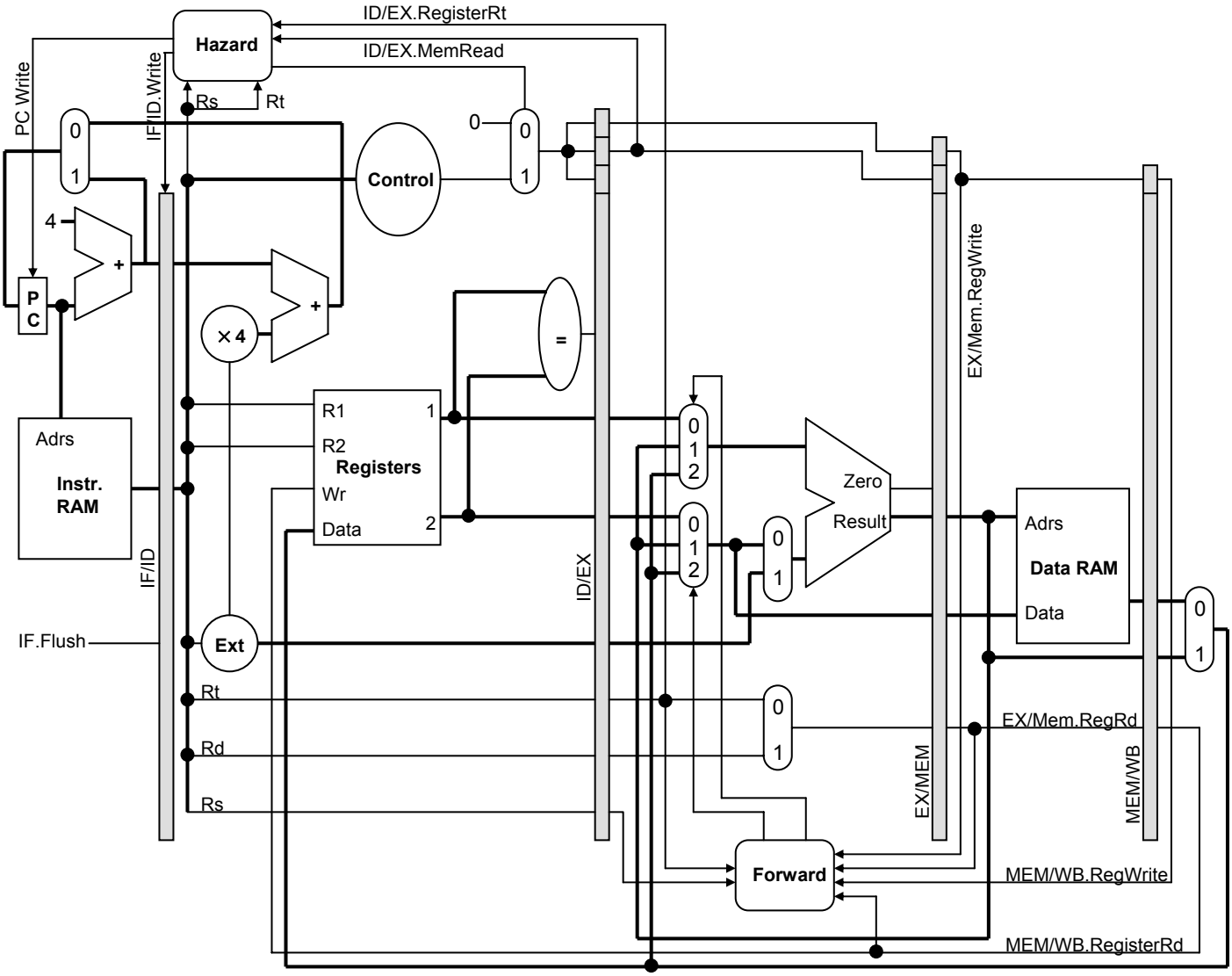
If we execute this code using the datapath on page 7, where branches are determined in the ID stage, then the new value of $s1 will be needed in the *same* cycle in which it is produced, as shown in the diagram below.



Our datapath doesn't handle this situation properly because it doesn't permit forwarding to the ID stage, so let's try to fix that. Show how multiplexers and a forwarding unit can be added to the ID stage of the datapath on the next page (it's the same as the one on page 7), and give forwarding equations to explain how the mux selection is made.

You can assume the register file and ALU have the same latency while muxes have negligible delays, so you can forward the ALU output directly to the branch comparator. Also, you only need to consider dependencies between R-type arithmetic instructions and branches as in the example above.

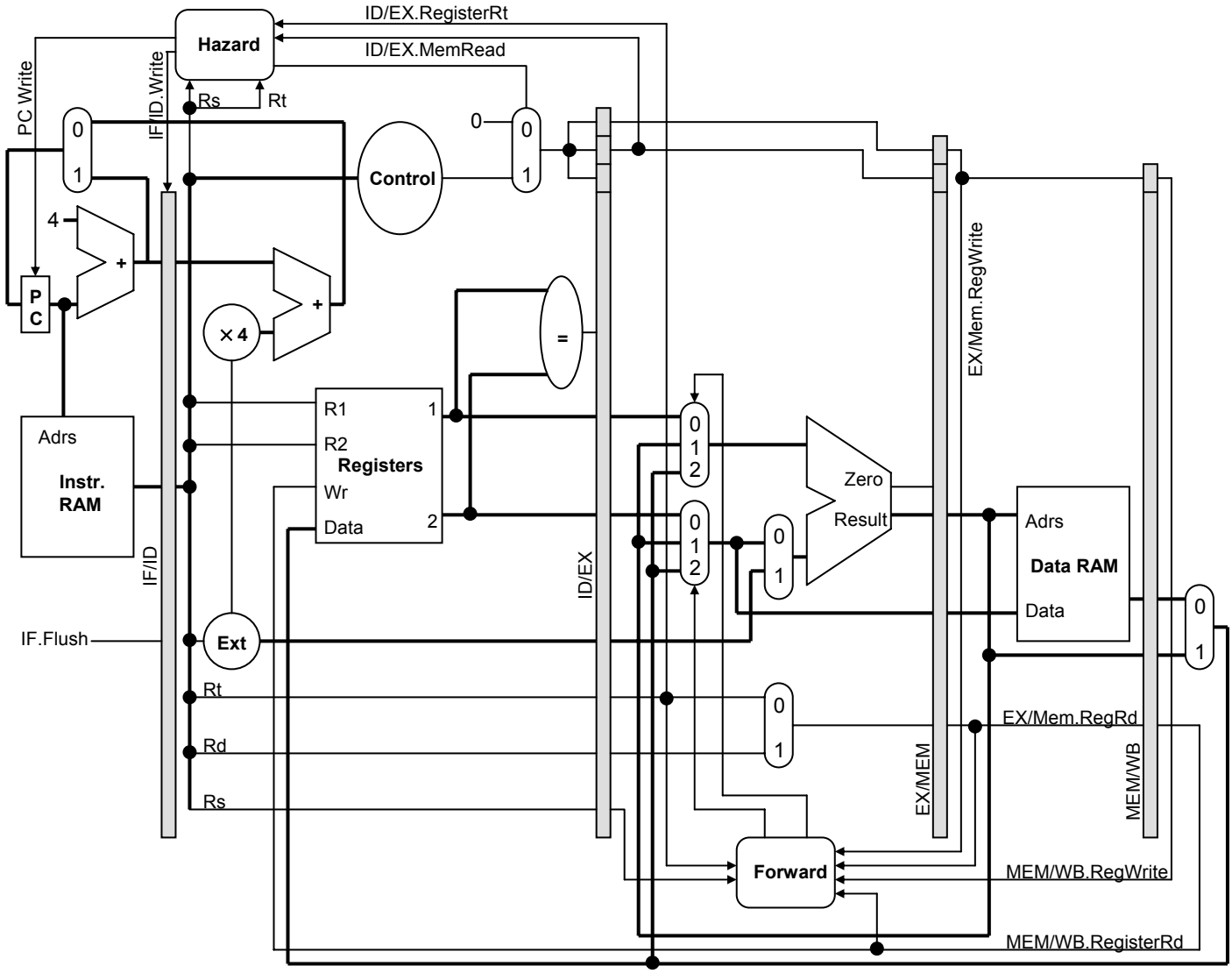
Question 3 continued



Pipelined Datapath

Here is the final datapath that we discussed in class.

- Forwarding for arithmetic operations is done from the EX/MEM and MEM/WB stages to the ALU.
- A hazard detection unit can insert stalls for lw instructions.
- Branches are assumed to be not taken, and branch determination is done in the ID stage.
- Equations for the ForwardA mux are given below; the ForwardB mux is similar.



if (MEM/WB.RegWrite = 1
and MEM/WB.RegisterRd = ID/EX.RegisterRs
and EX/MEM.RegisterRd ≠ ID/EX.RegisterRs)
then ForwardA = 1

if (EX/MEM.RegWrite = 1
and EX/MEM.RegisterRd = ID/EX.RegisterRs)
then ForwardA = 2