You may work in pairs for this assignment. If you choose to work with a partner, make sure only one of you submits a solution, and you paste a copy of the Partners Template that contains the names and PIDs of both students at the beginning of the file.

Prepare your answers to the following questions in a plain text file. Submit your file to the Curator system by the posted deadline for this assignment. No late submissions will be accepted. For all questions, show supporting work if you want partial credit.

You will submit your answers to the Curator System (www.cs.vt.edu/curator) under the heading `MIPS03`.

For questions 1 through 4, refer to the incomplete preliminary pipeline design, shown below, which includes the interstage buffers needed to synchronize signals and data with the instructions, and support for forwarding operands. However, this design has no support for load-use hazard detection/stall, or for properly handling `beq` instructions if the branch is taken. This datapath supports correct execution of any sequence of the following MIPS instructions: `add`, `sub`, `and`, `or`, `slt`, `lw`, and `sw`.

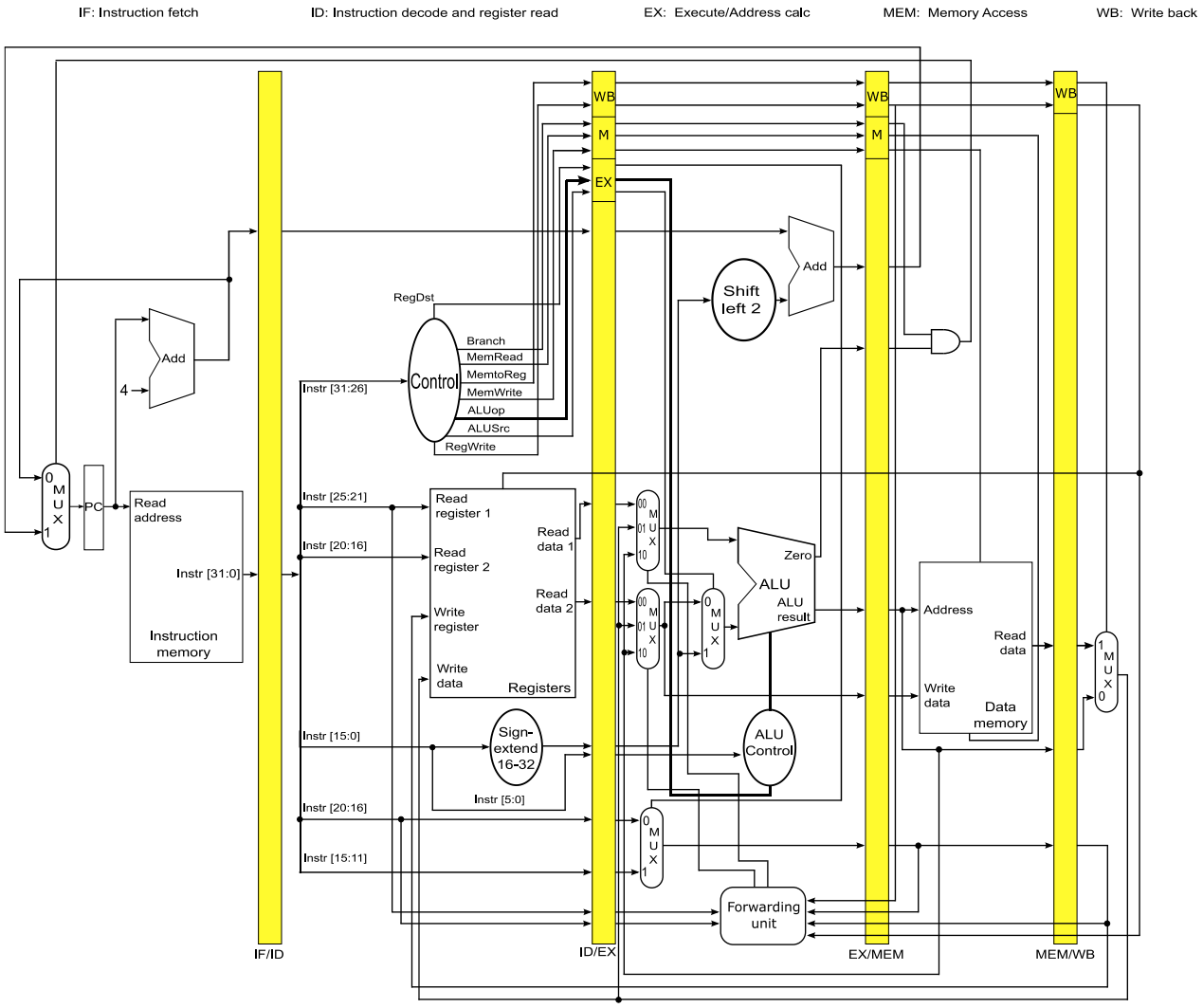

Figure 1

1. Consider the following sequence of instructions, which involves a number of data hazards, each of which will be handled by the existing forwarding logic:

```
add    $t0, $t1, $t2      # 1.1
add    $t1, $t0, $t2      # 1.2
sub    $t2, $t0, $t1      # 1.3
lw     $t1, 0($t2)        # 1.4
sw     $t2, 4($t0)        # 1.5
beq    $t0, $t2, exit     # 1.6
```

Note: there are no load-use hazards here that are not resolved by the existing forwarding logic.

a) **[16 points]** Identify all such hazards, and complete a table like the following one. The first line in the table below IS a correct response, and shows how a row in the table should be filled in.

```
Writer   Reader   Register   Source of forwarded value
-----------------------------------------------------
  #1.1     #1.2     $t0          EX/MEM buffer
```

b) **[5 points]** How many clock cycles would be required to execute the given sequence of instructions on the pipeline design shown in Figure 1?

2. **[10 points]** Consider the following sequence of instructions, which involves one or more load-use hazards:

```
lw     $t1, 8($t0)        # 2.1
sw     $t1, 4($t0)        # 2.2
add    $s0, $t1, $t2      # 2.3
lw     $s2, 0($s0)        # 2.4
add    $s3, $s0, $s2      # 2.5
add    $s3, $s2, $s3      # 2.6
```

The sequence also may include data hazards that will be handled correctly by the forwarding logic the pipeline includes; you should ignore those data hazards in your answer.

In order for the sequence of instructions to be executed correctly on this pipeline design, one or more `nop` instructions must be inserted (so that the existing forwarding logic can do the rest). **Rewrite the sequence of instructions** to show how the sequence of instructions could be "fixed" by inserting the smallest possible number of `nop` instructions.

3. **[12 points]** Consider the following sequence of instructions, which involves both `lw` and `beq` instructions:

```
     lw     $t0, 8($t4)        # 3.1
     beq    $t0, $t1, L1       # 3.3
     lw     $t2, 4($t1)        # 3.2
L1:  sub    $t4, $t2, $t0      # 3.6
     add    $t2, $t0, $t2      # 3.4
```

The sequence also may include data hazards that will be handled correctly by the forwarding logic the pipeline includes; you should ignore those data hazards in your answer.

In order for the sequence of instructions to be executed correctly on this pipeline design, one or more `nop` instructions must be inserted (so that the existing forwarding logic can do the rest). **Rewrite the sequence of instructions** to show how the sequence of instructions could be "fixed" by inserting the smallest possible number of `nop` instructions.

4. Consider the alternate implementation of the MIPS pipeline sets the Write register # for each instruction in the ID stage, as shown in **Figure 2**:

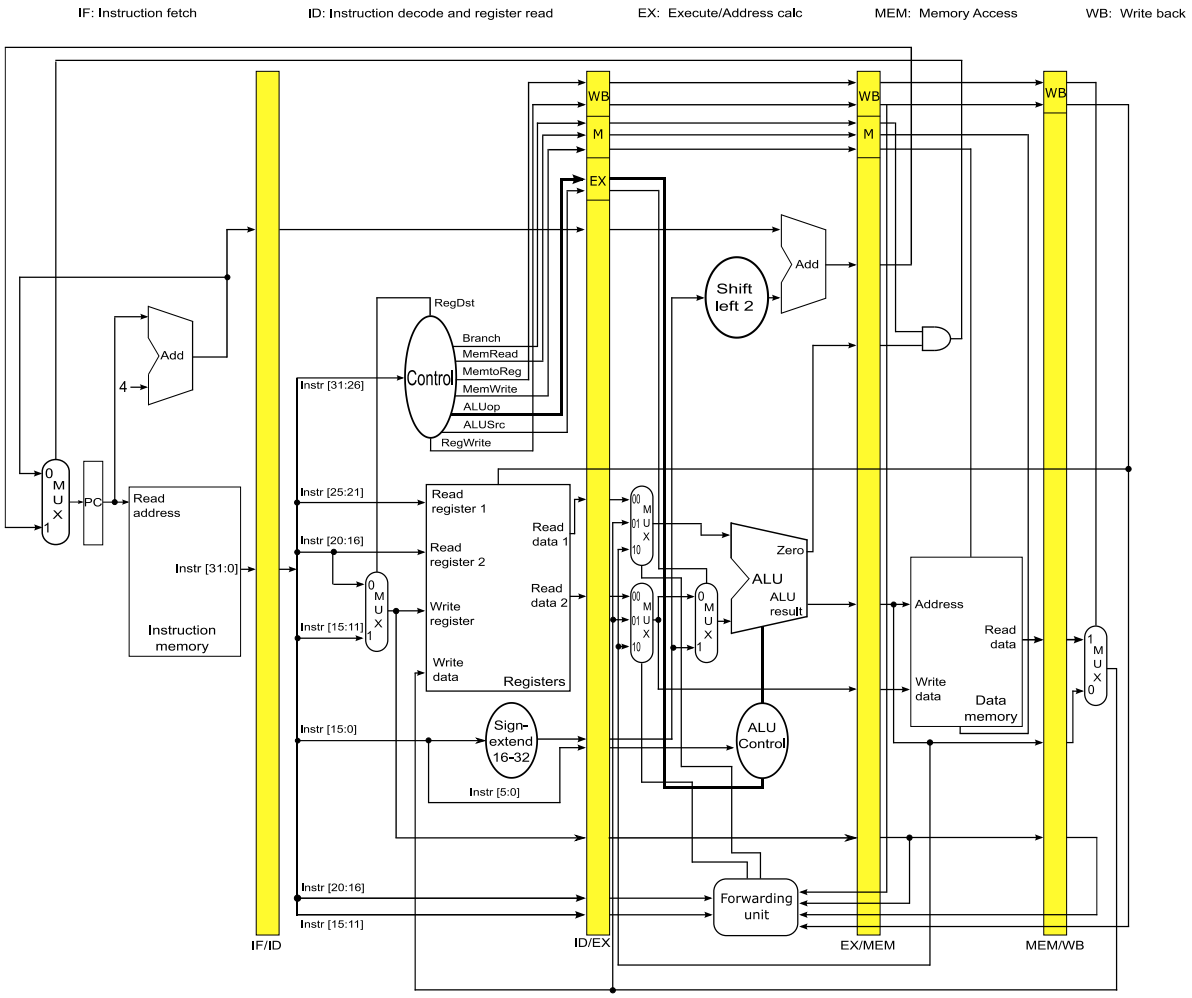

Figure 2

Suppose the following sequence of instructions is sent into the pipeline shown above, and the registers initially store the values shown in the table below:

```
add   $t1, $t2, $t3        # 4.1
add   $t5, $t1, $t2        # 4.2
add   $t2, $t5, $t1        # 4.3
add   $t3, $t3, $t1        # 4.4
add   $t1, $t5, $t1        # 4.5
```

register	initial value
$t1	1000
$t2	2000
$t3	3000
$t4	4000
$t5	5000

a) **[5 points]** Which register will instruction `#4.1` actually write its result to?

b) **[5 points]** What value will instruction `#4.4` actually read from `$t3`?

c) **[5 points]** What value will instruction `#4.2` actually send to the ALU for its first (left) operand?

For questions 5 through 7, refer to the pipeline design with forwarding and (load-use) hazard detection, shown below, which supports execution any sequence of the following MIPS instructions: `add`, `sub`, `and`, `or`, `slt`, `lw`, and `sw`.

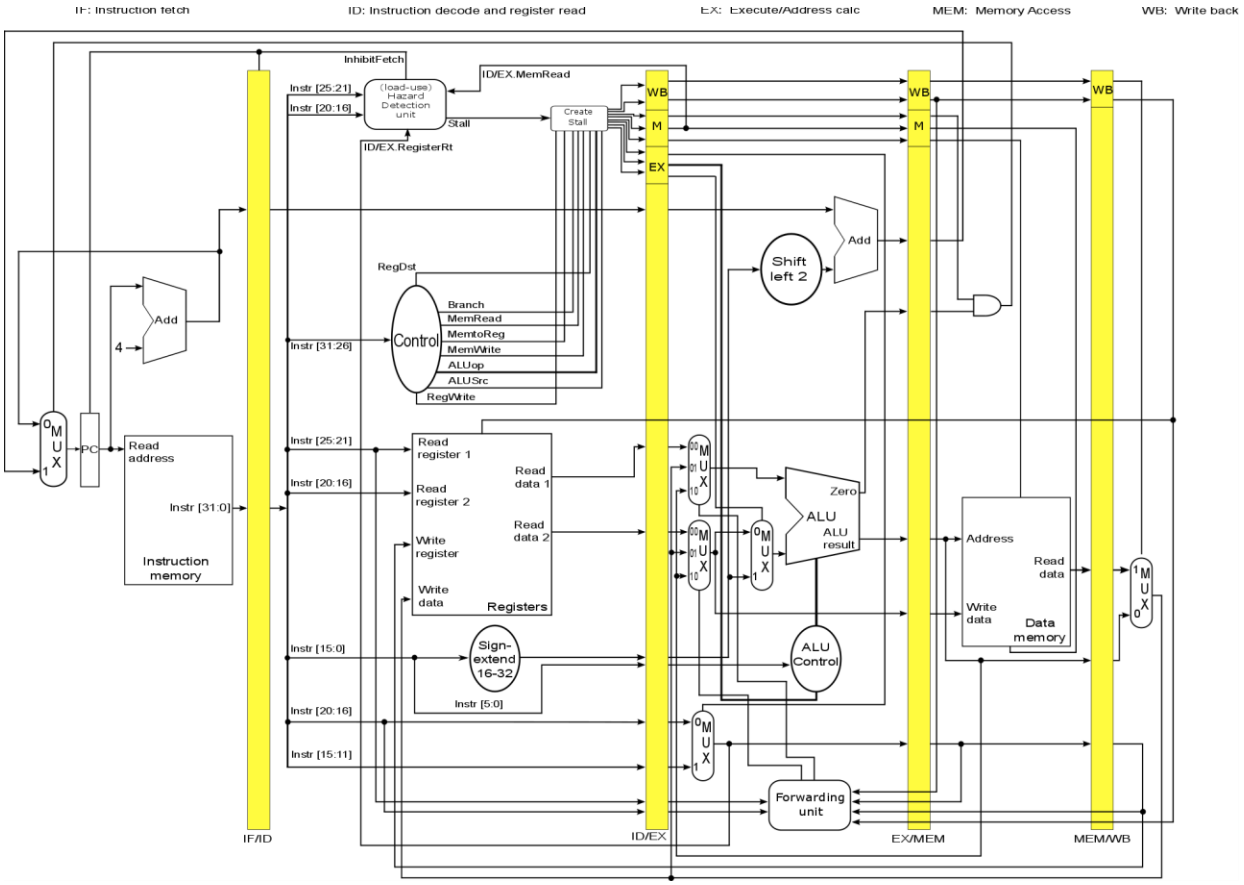

Figure 3

5. [15 points] Suppose that, due to a manufacturing defect, the InhibitFetch control signal from the load-use Hazard Detection unit suffers a *stuck-at-0 error*. That is, the InhibitFetch control signal is initialized to 0 until the first instruction enters the ID stage, and the Hazard Detection unit then always sets InhibitFetch to 0, no matter what the inputs to the Hazard Detection unit are. Assume that the rest of the hardware operates as designed.

Suppose that, initially, the registers used below are initialized as shown below. Suppose that all the memory words are initialized to be `8000`. Consider the execution of the following code in this buggy pipeline. Determine the final values of the `$t1`, `$t3`, and `$t4` registers after all the instructions leave the pipeline (no other instructions modify any of these registers).

```
add  $t1, $t2, $t3      # 5.1
lw   $t3, 0($t1)        # 5.2
add  $t4, $t2, $t3      # 5.3
add  $t3, $t1, $t2      # 5.4
```

register	initial value
`$t1`	1000
`$t2`	2000
`$t3`	3000
`$t4`	4000
`$t5`	5000

6. Suppose the following instructions are in the pipeline shown in **Figure 3**, in the stages indicated:

```
lw   $t1, 0($t3)     # 6.1 is in the EX stage
add  $t2, $t1, $t5   # 6.2 is in the ID stage
```

The load-use Hazard Detection Unit receives three register numbers as input:

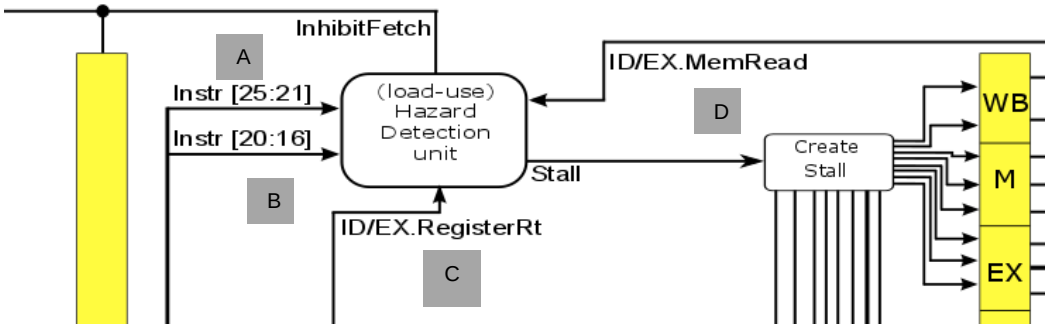

Figure 4

a) **[5 points]** What register number will the Hazard Detection unit receive for input A?

b) **[5 points]** What register number will the Hazard Detection unit receive for input B?

c) **[5 points]** What register number will the Hazard Detection unit receive for input C?

7. **[12 points]** Review the discussions of the Forwarding Unit and the load-use Hazard Detection Unit. Suppose that the branch target address computation and the register comparison (for `beq` instructions) are moved into the ID stage, as discussed in class. We have seen that if there is a read-after-write data hazard, where the reading instruction is `beq`, then it may be necessary to stall the `beq` instruction, for one or two cycles. For each instruction sequence shown below, state whether `beq` would need to be stalled 0, 1 or 2 cycles, and state the name(s) of the register(s) involved in the hazard that leads to the need to stall. (We assume that the datapath design has been modified to include forwarding hardware so that operands can be substituted into the equality unit that compares registers in the ID stage.)

a)
```
add   $t2, $t0, $t1
add   $s5, $s3, $s2
beq   $t2, $s5, L1
```

b)
```
add   $s5, $s3, $s2
add   $t2, $t0, $t1
beq   $t2, $s5, L1
```

c)
```
lw    $s5, ($s3)
lw    $t2, ($t1)
beq   $t2, $s5, L1
```

d)
```
lw    $t2, ($t1)
add   $s5, $s3, $s2
beq   $t2, $s5, L1
```

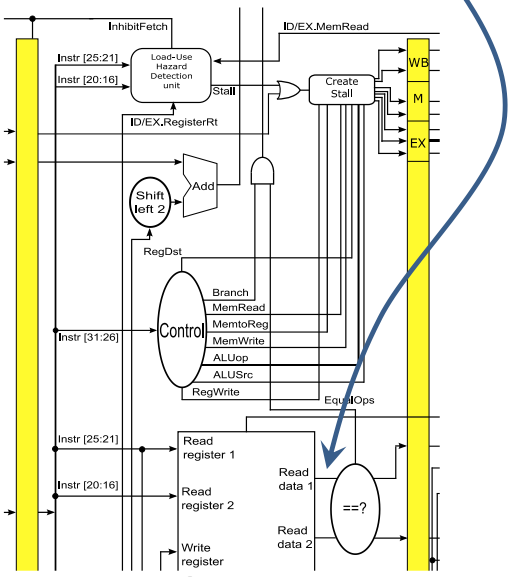

Figure 5