Problem 1

Consider the following C structure:

```c
struct mylist {
    double value;
    struct mylist *next;
    struct mylist *prev;
}
```

(a) 40 points Assuming that `cur` is a pointer to some node in a circular doubly linked list (with more than 1 node) convert the following C procedure to the MIPS assembly one. Assume that the value of `cur` is passed in register `$a0`.

```c
void myfunc (struct mylist *cur) {
    cur->prev->next = cur->next;
    cur->next->prev = cur->prev;
    return;
}
```

```assembly
myfunct:
    lw $t0, 8($a0) # load to $t0 pointer $cur->next
    lw $t1, 12($a0) # load to $t1 pointer $cur->prev
    sw $t0, 8($t1) # $cur->prev->next = $cur->next (i.e. $t0);
    sw $t1, 12($t0) # $cur->next->prev = $cur->prev (i.e. $t1);
    jr $ra # jump out of the procedure myfunct
```

(b) 10 points Briefly describe in English what the code does

The function deletes element `cur` from the doubly linked list by redirecting `next` and `prev` pointers from the previous element and the next element, respectively.
Problem #2

For the MIPS datapath shown below, several lines are marked with “X”.

For each one (total three cases):

a)  (3 points × 3)  Describe in words the negative consequence of cutting this line relative to the working, unmodified processor (use extra page if needed).

b)  (6 points × 3)  Provide a snippet of code (1-3 lines) that will fail

c)  (6 points × 3)  Provide a snippet of code (1-3 lines) that will still work

<table>
<thead>
<tr>
<th>(a1) Cannot write to register file. This means that R-type and any instruction with write back register file will fail</th>
<th>(a2) Forwarding of the first operand fails.</th>
<th>(a3) Jumping to a branch target does not work.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(b1) add $s1, $s2, $s3</td>
<td>(b2) add $s1, $t0, $t1 add $s1, $s1, $s1</td>
<td>(b3) addi $s1, $0, 2 add $s2, $0, 2 beq $s1, $s2, exit</td>
</tr>
<tr>
<td>(c1) sw $s1, 0($s2)</td>
<td>(c2) add $s1, $t0, $t1 add $s1, $t2, $s1</td>
<td>(c3) addi $s1, $0, 10 add $s2, $0, 20 beq $s1, $s2, exit</td>
</tr>
</tbody>
</table>
Problem #3

Problem 3: Given the code below show how it will be processed in a pipeline highlighting the forwarding operations with arrows between corresponding stages and denoting stalls by X. Calculate the total number of cycles it takes the code to execute.

instruction 1: sw $t0, 0($a1)
instruction 2: lw $a1, 0($t0)
instruction 3: add $a1, $t0, $t1
instruction 4: sll $t4, $t4, 1
instruction 5: xor $a2, $a1, $t4

Solution

<table>
<thead>
<tr>
<th>C0</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>C5</th>
<th>C6</th>
<th>C7</th>
<th>C8</th>
<th>C9</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF_1</td>
<td>ID_1</td>
<td>EX_1</td>
<td>MEM_1</td>
<td>WB_1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF_2</td>
<td>ID_2</td>
<td>EX_2</td>
<td>MEM_2</td>
<td>WB_2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF_3</td>
<td>ID_3</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF_4</td>
<td>ID_3</td>
<td>EXE_3</td>
<td>MEM_3</td>
<td>WB_3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF_4</td>
<td>ID_4</td>
<td>EXE_4</td>
<td>MEM_4</td>
<td>WB_4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF_5</td>
<td>ID_5</td>
<td>EXE_5</td>
<td>MEM_5</td>
<td>WB_5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cycle 3: Load use data hazard is detected and the execution of third instruction is stalled. More specifically, as a result of hazard write enable signal is de-asserted for pipeline registers PC and IF/ID and the context of these registers is not changed from C3 to C4. Simultaneously, the control signal portion of the ID/EXE register is set to 0 turning the instruction to NOP (denoted as X) which can be safely propagated down the pipeline without changing memory or register file.

Cycle 4: Forwarding unit bypasses value of register a1 (which has been just read from memory by instruction 2 but not yet updated in register file) from MEM state to EXE stage

Cycle 7: Forwarding units bypass values of register $t4 and of register $a1 from EXE and MEM stages, correspondingly, to the EXE stage

The total number of cycles is 10.