**Question 1**

Let consider the *Hazards* that may arise in a pipelined processor. You are requested to

1. List the different types of Hazards, detailing in particular the different categories of Data Hazards
2. Describe for every type and category an example where the Hazard happens
3. Identify the corresponding data dependency (for Data Hazards, only).

Thare are different types of Hazards, Structural Hazards due to resource conflict, Data Hazards due to dependencies between instructions and Control Hazards due to Branches.

There are 2 types of dependencies that can lead to a data hazard: Data dependencies and Name Dependencies. These can be grouped into RAW, WAR and WAW

Example of Strucutral hazards is when 2 different instructions want to use the mem stage at the same time.

RAW is a Data Dependecies, it is when an instruction wants use a result of a previous instruction:

ADD r0, r1, r2

ADD r3, r4, r0

WAR is an Anti-Dependence, it is when an instruction wants to write into a register that is write in the previous instruction

ADD r1, r2, r0

ADD r0, r3, r4

WAW is an Output Dependence, it is when two instructions want to write in the same register

ADD r0, r1, r2

ADD r0, r3, r4

Example of Control Dependence is every time there is a branch.

To Solve a Name dependence is enough to use register renaming.

**Question 2**

Let consider a MIPS architecture using a *Branch History Table* (BHT) composed of 8 1-bit entries. Let assume that this architecture executes the following code, which computes the number of equal non-null elements placed in the same position in 2 vectors V1 and V2, each composed of 5 integer elements, and then writes the result in the variable Res. For ever instruction the hexadecimal address of the memory cell storing the instruction is reported.

Assuming that when the execution of the code fragment the BHT is full of null values (corresponding to the prediction Not Taken) you are asked to compute:

* The number of mispredicted branches during the execution of the fragment
* The BHT content when the execution finishes (using the table reported in the next page).

For all computation we suggest the usage of the table in the next page.

V1: .word 0, 2, 4, 6, 8 # vector V1

V2: .word 0, 1, 3, 6, 0 # vector V2

Res: .word 0 # result

…

0x0020 and $t3, $0, $0 # initialize index

0x0024 and $t4, $0, $0 # initialize counter of equal non-null elements

0x0028 lab1: lw $t0, V1($t3) # load first element from V1

0x002c lw $t1, V2($t3) # load first element from V2

0x0030 bne $t0, $t1, lab2 # compare for equal value

0x0034 beq $t0, $0, lab2 # compare for non-null value

0x0038 add $t4, $t4, 1 # increment counter

0x003c lab2: add $t3, $t3, 4 # increment index

0x0040 bne $t3, 16, lab1 # repeat

0x0044 sw $t4, Res # store result

…

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | Iteration #1 | | Iteration #2 | | Iteration #3 | | Iteration #4 | | Iteration #5 | |
|  | | prediction | result | prediction | result | prediction | result | prediction | result | prediction | result |
|  | and $t3, $0, $0 |  |  |  |  |  |  |  |  |  |  |
|  | and $t4, $0, $0 |  |  |  |  |  |  |  |  |  |  |
| lab1: | lw $t0, V1($t3) |  |  |  |  |  |  |  |  |  |  |
|  | lw $t1, V2($t3) |  |  |  |  |  |  |  |  |  |  |
|  | bne $t0, $t1, lab2 | untaken | Untak | Untake | Take | Take | Take | Take | untake |  |  |
|  | beq $t0, $0, lab2 | untaken | take |  |  |  |  | Take | untake |  |  |
|  | add $t4, $t4, 1 |  |  |  |  |  |  |  |  |  |  |
| lab2: | add $t3, $t3, 4 |  |  |  |  |  |  |  |  |  |  |
|  | bne $t3, 16, lab1 | untaken | take | Take | Take | Take | Take | Take | untake |  |  |
|  | sw $t4, Res |  |  |  |  |  |  |  |  |  |  |

**HBT  
Final content**

|  |  |
| --- | --- |
| Entry 0 | 0 |
| Entry 1 |  |
| Entry 2 |  |
| Entry 3 |  |
| Entry 4 | 0 |
| Entry 5 | 0 |
| Entry 6 |  |
| Entry 7 |  |

6