**Question #1**

Let consider the Branch Prediction Mechanism based on the Branch History Table (BHT).

You are requested to

1. Describe the architecture of a BHT
2. Describe the behavior of a BHT, detailing when the processor accesses it, and which information it gets from it
3. Assuming that the processor uses 32 bit addresses, each instruction is 4 byte wide, and the BHT is composed of 16 entries, identify the final content of the BHT if
   * The BHT is initially empty (i.e., full of 0s, meaning NotTaken)
   * The following instructions are executed in sequence
     + l.d r2,v1(r1) located at the address 0x00A50050
     + bnez r2,l1 located at the address 0x00A50054; the branch is taken, and the branch target address is 0x00A60050
     + addi r2, r2, #1 located at the address 0x00A60050
     + bez r2,l2 located at the address 0x00A60054; the branch is not taken
     + andi r5, r5, #1 located at the address 0x00A60058
     + bez r2,l3 located at the address 0x00A6005C; the branch is taken, and the branch target address is 0x00BB0040
     + add r1, r2, r3 located at the address 0x00BB0040
     + bez r1,l4 located at the address 0x00BB0044; the branch is taken, and the branch target address is 0x00A50050.
4. For the example in the previous point, determine which of the 4 branch instructions were correctly predicted, and which were mispredicted.

Branch History Table is a Table composed of N entries, each entry store one or two bits that indicate if the branch must be taken or untaken.

~~Every~~ Each time a branch is Decoded the log2(N) LSBs of the address of the branch(excluding the ~~first 2~~ bits related to ~~PC~~ instructions alignment) are used to access to the BHT, if the value in the entry is 1, then the branch is taken, otherwise the branch is untaken. The value is updated as soon as the is known the branch result.

At the end, all entries are with value 0 except for the entry 1 and 7.

In the example all the branches are mispredicted.

**Question #2**

Let consider a MIPS64 architecture including the following functional units (for each unit the number of clock periods to complete one instruction is reported):

* Integer ALU: 1 clock period
* Data memory: 1 clock period
* FP arithmetic unit: 2 clock periods (pipelined)
* FP multiplier unit: 4 clock periods (pipelined)
* FP divider unit: 8 clock periods (unpipelined)

You should also assume that

* The branch delay slot corresponds to 1 clock cycle, and the branch delay slot is not enabled
* Data forwarding is enabled
* The EXE phase can be completed out-of-order.

You should consider the following code fragment and, filling the following tables, determine the pipeline behavior in each clock period, as well as the total number of clock periods required to execute the fragment. The values of the constants k1 and k2 are written in f10 and f11 before the beginning of the code fragment.

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MIPS64 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; for (i = 0; i < 10; i++) {

; v5[i] = v1[i]\*k1 + v2[i]/k2;

; }

|  |  |  |
| --- | --- | --- |
| .data | Comments | Clock cycles |
| v1: .double “10 values” |  |  |
| v2: .double “10 values” |  |  |
| V3: .double “10 values” |  |  |
|  |  |
|  |  |
| .text |  |  |
| main: daddui r1,r0,0 | r1← pointer |  |
| daddui r2,r0,10 | r2 <= 20 |  |
| loop: l.d f1,v1(r1) | f1 <= v1[i] |  |
| mul.d f2, f1, f10 | f2 <= v1[i] \*k1 |  |
| l.d f3,v2(r1) | f3 <= v2[i] |  |
| div.d f4, f3, f11 | f4 <= v2[i]/k2 |  |
| add.d f5, f4, f2 | f5 <= v1[i]\*k1 + v2[i] /k2 |  |
| s.d f5,v3(r1) | v3[i] <= f5 |  |
| daddui r1,r1,8 | r1 <= r1 + 8 |  |
| daddi r2,r2,-1 | r2 <= r2 - 1 |  |
| bnez r2,loop |  |  |
| halt |  |  |
| total |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| main: daddui r1,r0,0 | F | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 5 |
| daddui r2,r0,10 |  | F | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |
| loop: l.d f1,v1(r1) |  |  | F | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |
| mul.d f2, f1, f10 |  |  |  | F | D |  | E | E | E | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 5 |
| l.d f3,v2(r1) |  |  |  |  | F |  | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 0 |
| div.d f4, f3, f11 |  |  |  |  |  |  | F | D |  | E | E | E | E | E | E | E | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 7 |
| add.d f5, f4, f2 |  |  |  |  |  |  |  | F |  | D |  |  |  |  |  |  |  | E | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  | 2 |
| s.d f5,v3(r1) |  |  |  |  |  |  |  |  |  | F |  |  |  |  |  |  |  | D | E |  | M | W |  |  |  |  |  |  |  |  |  |  |  | 1 |
| daddui r1,r1,8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F | D |  | E | M | W |  |  |  |  |  |  |  |  |  |  | 1 |
| daddi r2,r2,-1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F |  | D | E | M | W |  |  |  |  |  |  |  |  |  | 1 |
| bnez r2,loop |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F |  | D | E | M | W |  |  |  |  |  |  |  | 2 |
| halt |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |