**Question #1**

Let consider the issue of predicting branch results.

You are requested to

1. Explain why this issue is important in pipelined architectures
2. Summarize the characteristics of Static and Dynamic Branch Prediction mechanisms, highlighting their main advantages and disadvantages
3. Describe the general architecture and behavior of a BHT
4. Detail the architecture and behavior of a 1-bit BHT.
5. Because branch in pipeline can change pc and depending on the type of branch which is conditional or unconditional the next instruction can be predictable or not. For branches that are not conditional the next Branch pc is available but we push instructions after branch to the pipeline and we need to do a flush if it is not taken.
6. Static branch predictions are managed by software and in compile time and dynamic branch prediction can be predict in runtime and by adding a piece of hardware. So if we can choose taken or not taken without looking at the branch which is quick but we may lose clock cycles. And in static predictions compiler may change instructions by loop unenrolling or changing the order of the instructions to save clock cycles.
7. In BHT we can predict that a branch will be taken or not using 3 approaches

one is the address of branch which is going backward or going forward so if it is going back we can say that it maybe a loop so it should be taken and if it is going forward it is not taken and the other approach is using previous branch states looking the history of branch and the other one is loading some instructions whether it is taken or not.

1. In 1-bit BHT we add 1 to a bit if previously a branch is taken and subtract 1 if it is not taken so by looking at the bit in next calls again we can use the history of that branch which was taken or not because of that we can avoid problems like flushing pipeline.

**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, using the table in the following page (where each column corresponds to a clock period), and determine the pipeline behavior in each clock period, as well as the total number of clock periods required to execute the fragment, reporting the result in the right column in the table below.

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

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

; v5[i] = v1[i]/v2[i] + v3[i]\*v4[i];

; }

|  |  |  |
| --- | --- | --- |
| .data | comments | Clock cycles |
| V1: .double “10 values” |  |  |
| V2: .double “10 values” |  |  |
| V3: .double “10 values”  V4: .double “10 values”  V5: .double “10 values” |  |  |
|  |  |
|  |  |
|  |  |
| .text |  |  |
| main: daddui r1,r0,0 | r1← pointer | 5 |
| daddui r2,r0,10 | r2 <= 20 | 1 |
| loop: l.d f1,v1(r1) | f1 <= v1[i] | 2 |
| l.d f2,v2(r1) | f2 <= v2[i] | 1 |
| l.d f3,v3(r1) | f3 <= v3[i] | 1 |
| l.d f4,v4(r1) | f4 <= v4[i] | 1 |
| mul.d f5,f3,f4 | f5 <= v3[i]\*v4[i] | 6 |
| div.d f6, f1, f2 | f6 <= v1[i] / v2[i] | 5 |
| add.d f7, f6, f5 | f7 <= v3[i]\*v4[i] + v1[i] / v2[i] | 4 |
| s.d f7,v4(r1) | v5[i] <= f7 | 1 |
| daddui r1,r1,8 | r1 <= r1 + 8 | 1 |
| daddi r2,r2,-1 | r2 <= r2 - 1 | 1 |
| bnez r2,loop |  | 2 |
| halt |  | 0 |
| total | 34\*10 | 340 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| daddui r1,r0,0 | f | d | e | m | w |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| daddui r2,r0,10 |  | f | d | e | m | w |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| l.d f1,v1(r1) |  |  |  | f | d | e | m | w |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| l.d f2,v2(r1) |  |  |  |  | f | d | e | m | w |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| l.d f3,v3(r1) |  |  |  |  |  | f | d | e | m | w |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| l.d f4,v4(r1) |  |  |  |  |  |  | f | d | e | m | w |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| mul.d f5,f3,f4 |  |  |  |  |  |  |  |  |  | f | d | e | e | e | e | m | w |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| div.d f6, f1, f2 |  |  |  |  |  |  |  |  |  |  | f | d | e | e | e | e | e | e | e | e | m | w |  |  |  |  |  |  |  |  |  |  |  |  |
| add.d f7, f6, f5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | f | d | e | e | m | w |  |  |  |  |  |  |  |  |
| s.d f7,v5(r1) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | f | d | s | e | m | w |  |  |  |  |  |  |  |
| daddui r1,r1,8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | f | s | d | e | m | w |  |  |  |  |  |  |
| daddi r2,r2,-1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | s | f | d | e | m | w |  |  |  |  |  |
| bnez r2,loop |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | f | s | d | e | m | w |  |  |  |
| halt |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | f |  |  |  |  |  |  |