Processor Design Choices:

1)Handling and Execution of branch instruction :

There are three parts to a branch instruction.

1)Detecting that it is a branch.

2)Second calculating it's value required for the comparison.

3)Comparison of the value and updation of the PC accordingly.

In our design what we have done is we have assumed always branch not taken and using the ALU for calculating the value required for comparison and in the memory stage is when we get to know whether to take branch or not. As a consequence of this when a branch is confirmed, three invalid instructions are into the pipeline which we am flushing. So there are 3 idle cycles for each branch taken.

2)Handling of the mult-add instruction :

Since we require to access four registers for implementing it, what we did is treated it as mult instruction and simply added the calculated Hi,Lo values to the existing ones.

3)Working with stalls :

In the case where a stall is neccesarily needed we have stalled PC,IFID,IDEX register and introduced a bubble in the execution stage. As a consequence the EXMEM register recieves a NOP on that clock cycle.

Error Handling:

Following is the list of error that we have handled followed by those we have not handled.

1)Wrong Input file given.

2)Incorrect format of hex in the input file.

3)Invalid hex code.

4)Unsupported instruction.

5)Overlow in addition,subtraction.

6)Index out of bounds in load word,store word.

7)The given address is not a multiple of 4 in case of load word,store word.

8)Index out of bounds of some address in case of a MemDump command.

Errors we have not handled.

1)Invalid hex given in case of a MemDump command.

BreakPoint Handling:

Respecting the fact that we are simulating a pipelined processor, what we have done is placed the breakpoint in such a way that whenever the instruction at which break is placed is being about to load then our program stops. we acknowledge that this may not be the best way to handle these but we decided not to tinker with our pipelined design and override it.

Integrating cache with processor:

So the way we integrated it is as follows whenever in our processor design we had an memory access we simply trigger the cache simulator with the appropiate request so that it can update it's stats. Finally use its stats to print the corresponding results file.

we have handled all the cache configurations and our trace driven cache simulator is working perfectly.

In your test cases there were some instructions that were unsupported, hence we have not tested our program on these input test cases.

If we encounter any instruction that is unsupported than our program exits after informing the user about the same.

Some Assumptions we took:

1)Block size and all the parameters are assumed to be a perfect power of 2(that makes is easy to deal with)

2) Stack pointer is initialized with the highest possible memory location.

3)All registers are initialised to 0.

4) All parameters and policies of the different cache are assumed to be the same.