**CDA155 Assignment 2**

Shalini Tangella (FSUID: st13h)

Out of order execution of instructions is implemented in this simulator. Instructions are fetched in order and the execution is allowed to be out of order while we commit the values to the destination registers in the commit phase in order.

Two instructions are fetched at once unless a halt or a stall case. Destination registers are renamed while the rob queue and the associated reservation station are free. Flags are set such that each instruction execution follow each other i.e. if the first instruction isn't allocated we do not allocate the second instruction and thus we stall the next cycle fetch.

Once the instructions are allocated the execution isn't done in order but rather is done out of order I.e., the instructions with source operands ready are executed first. Functional units are used for scheduling the instructions. These functional units save the cycle start and cycle end so that the latency is registered for each instruction. We check every cycle if any of these functional units if set have an execution time as the current cycle. If so we remove the instruction from the reservation station and place the destination value in the reorderbuffer. Beq instructions are given priority over the normal add and name instructions in order to avoid more squashing of mispredicted instructions.

We commit two instructions each cycle and an instruction can be commuted only if all its previous instructions are commuted.

In the program, 4 separate structures have been used for reservation station purpose. Variables which store sources, destination and validity are common for all the instructions. So a basic structure includes these. And an instance of this structure has been created in the remaining three structures – one for add,nand,beq instrunctions , one for load and store , one for mult and div instructions respectively. Additionally functional unit flag is employed for mult and div reservation station. And in rs\_addtype, additional variables include beqPC, opertype and the reorder buffer index for the beq. Similarly, for record\_rs additional variables include operatn\_typ, store reorder buffer index and offset. The Reorder buffer is implemented using a queue. Additionally, beq\_bit determining beq instruction and and ren\_idn are used in the reorder buffer for for data forwarding.

One set contains the information at begin of cycle and the other set gets updated as the cycle progresses, and these values are carried to the next consequent cycle. Two methods were used, namely assign\_oldtonew() and assign\_newtoold() were used to exchange of information during cycles.

**TEST CASES:**

1. Test1:

lw 0 1 six

lw 0 2 seven

add 1 1 1

add 2 2 2

add 1 2 3

add 1 1 1

done halt

six .fill 1

seven .fill 2

* 4th add stalls the fetch stage as reservation station\_add will be full. It executes when first add executes and gets free from reservation station\_add!

1. Test 2:

lw 0 1 six

lw 1 2 5

start add 1 2 1

beq 0 1 1

beq 0 0 start

done halt

six .fill 2

neg1 .fill -1

staddr .fill start

* This test case is used to test branch prediction! Initially all the values in pht are initialized to W\_NOTTAKEN. If the branch is taken we increment pht value of that branch to W\_TAKEN and thus, it predicts to be taken when it is fetched next.

Here beq 0 1 1 is mispredicted once (when condition is satisfied) and beq 0 0 start is mispredicted once (the very first time).

1. Test 3:

lw 0 1 three

lw 0 2 four

mult 1 1 2

add 0 1 4

add 2 1 6

done halt

three .fill 4

four .fill 8

* This test case is used to test broadcasting. Add and mult wait for load instructions to complete and once load is executed, value 4 is broadcasted and mult add add instructions set the corresponding source valid bits to 1.

1. Test 4:

lw 0 4 four

add 4 0 5

nand 5 5 6

beq 6 0 1

sw 0 7 0

done halt

four .fill 6

* In this test case, reservation stations and rob gets initialized again after beq executes as nand updates register 6 to 0!

1. Test 5:

lw 0 1 ten

mult 1 1 4

add 1 1 4

mult 1 0 6

done halt

ten .fill 6

* This test case shows normal execution latency of all the reservation stations.

Load execution starts at 2nd cycle and ends at 5th cycle

Add execution starts at 6th cycle and ends at 7th cycle.

Mult 1 execution starts at 6th cycle and ends at 12th cycle.

Mult 2 execution starts at 8th cycle and ends at 14th cycle