* The group wrote three files (“func.h”, containing all the functions we might need to simulate the project), (“struct.h”, containing all the structs for the reservation stations and so ), (“Tomasulu.cpp”, representing the main function).
* The program reads the file called (“input.txt”, containing all the instruction separated by a new line character.)
* The group also wrote some test cases to make sure the program is working fine.
* We’ll be explaining two short programs for the sake of the report, one of which contains a loop.

First test case:

*Jalr r3*

*Sw r3, 10(r0)*

*Lw r7,10(r0)*

*Add r4,r5,r6*

*Neg r5,r6*

*Ret*

We will be explaining the program on every clock cycle until it finishes executing:

A picture containing calendar

Description automatically generated

In cc 1, the first instruction will be issued, the corresponding reservation station will be marked as busy

Calendar

Description automatically generated

In cc2 , the second will be issued marking the Rs as busy, and the first is to start execution

A picture containing calendar

Description automatically generated

In cc3, the first instruction will finishi execution and since it is a jalr instruction, the second instruction shall be flushed, marking its issued bit as -1 and the program will be fetching the value in r3, which is initialized as 3.

A picture containing graphical user interface

Description automatically generated

In cc4, the program will fetch the add instruction, leaving the second one flushed, and the jalr will be marked virtually written.

A picture containing graphical user interface

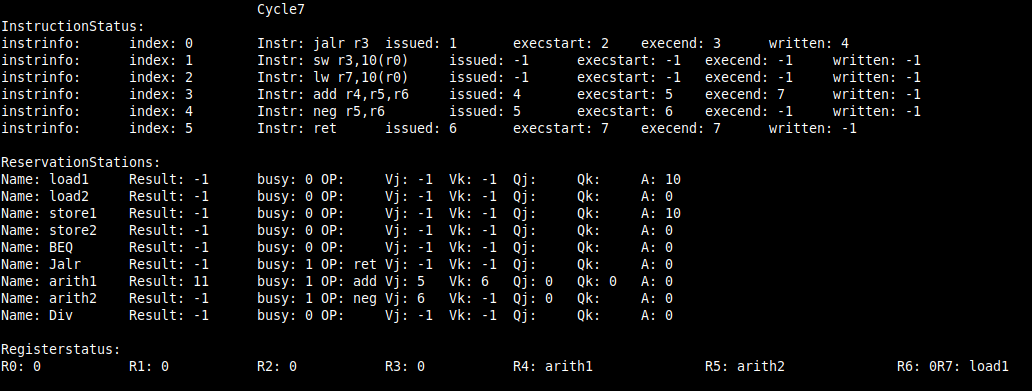
Description automatically generated

In cc5, the add instruction will proceed to execution and the neg instruction will be issued.

A picture containing graphical user interface

Description automatically generated

In cc6, the add is still in the execution and the add will proceed to execution, and the neg will proceed to execution, and the ret will be issued



In cc7, the add will complete execution; the ret will be ending execution returning back to the jalr position depending on R1;

A picture containing graphical user interface

Description automatically generated

In cc8, the instruction after the jalr will be issued, and the neg finishes execution.

A picture containing graphical user interface

Description automatically generated

In cc9, the last instruction before the function segment will be issued , and the neg will not be written as the common data path will be busy writing another instruction.

A picture containing graphical user interface

Description automatically generated

In cc10, the neg will be done writing, the sw is in the execution stage. The lw will start execution

A picture containing text, computer, computer

Description automatically generated

In cc11 , the sw will be written, and the lw will be still in the execution stage.

A picture containing graphical user interface

Description automatically generated

In cc12, only lw is in the pipeline, which now has done execution.

A picture containing graphical user interface

Description automatically generated

In cc13, the lw will be written back, freeing the reservation station, marking the end of the main program before the function.

In the previous screen shot the IPC is shown, we executed 6 inst/ 13 clk = 0.462

The program has no branches, that is why the branchencountered number= 0;

The program took 13 cycle to finish up writing all the instruciton and terminate.

Test case 2: Loop (small)

The following program targets only the concept of looping, that is why I made an infinite loop and in the main program, I specified the program to run for only 12 clock cycles

*add r5,r4,r7*

*neg r6,r4*

*beq r0,r0,-2*

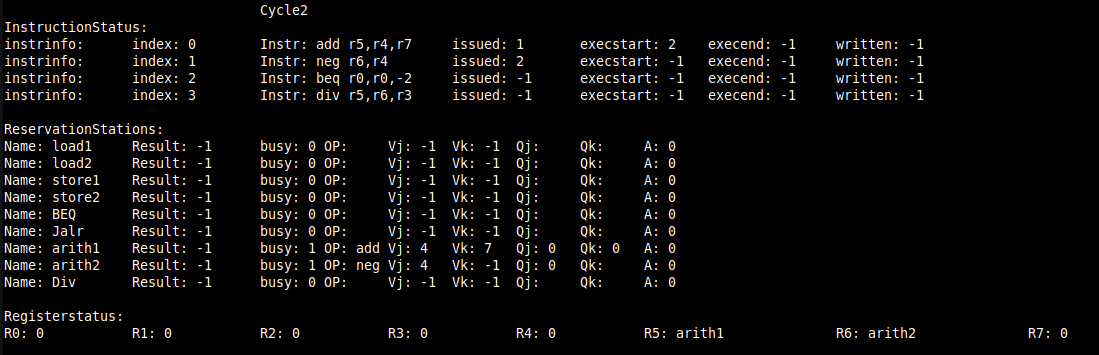
*div r5,r6,r3*

The program basically executes only two instructions over and over again:

A picture containing text, computer

Description automatically generated

In cc1, the first instruction is typically issued.



In cc2, the second will be issued, the first will move to the execution stage.

A picture containing calendar

Description automatically generated

In cc3, the neg will move to the execution and the add is still in the execution, the beq is issued.

A picture containing text

Description automatically generated

In cc4, the add will finish execution, and the neg will be still in the execution, the beq will effectively go to the execution; the instruction after the branch will be issued also but not executed until the branch is determined.

A picture containing text

Description automatically generated

In cc 5, the branch is determined to be taken, that is why the div instruction will be flushed and the add will be written, and the neg will finish execution; the miss number will increment by one.

A picture containing text, computer

Description automatically generated

In cc6, the first instruction will be issued all over again, and the neg will be completed by writing the result to the R6.

A picture containing text, computer

Description automatically generated

In cc7, the neg will be issued again, and the beq will be written virtualy to mark it as completed as instruciton. The add will start executing for the second time;

A picture containing text, computer, computer

Description automatically generated

In cc8, the beq will be issued all over again; by this time the add will still be in the execution stage;

The neg will move to the execution .

A picture containing text

Description automatically generated

In cc9, the div instruction will be issued but not executed until the branch is determined to be not taken. The add will finish execution, and be ready for writing in the next clock cycle, while the neg is is in the execution, and the beq has just started execution.

A picture containing text, computer, computer

Description automatically generated

In cc10, the branch is determined to be taken, thus, the div instruction will be flushed and the miss number will be incremented by one . the add will be written back and in the next clock cycle we will start the program all over again, and it will flow in the same pattern

A picture containing calendar

Description automatically generated

The miss ratio is 100 because I make it as an infinite loop; so it will be always taken, unlike our always not-taken prediction.

The IPC is still the same as it will be in 12 clk cycles executing the loop two times except for the last instruction which makes it 5 instruciton.

IPC=5/12= 0.614;