Pipeline Simulation in C++

by Rajat Golechha and Harshit Gupta

April 15, 2023

§1 Part D : Comparison of different pipelines

§1.1 About the Pipelines

We implemented 4 different types of pipelines in this assignment which are,

1. 5 stage - without bypassing
2. 5 stage - with bypassing
3. 7-9 stage - without bypassing
4. 7-9 stage - with bypassing

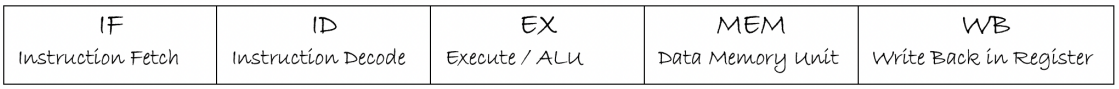

Fig : A 5 stage pipeline

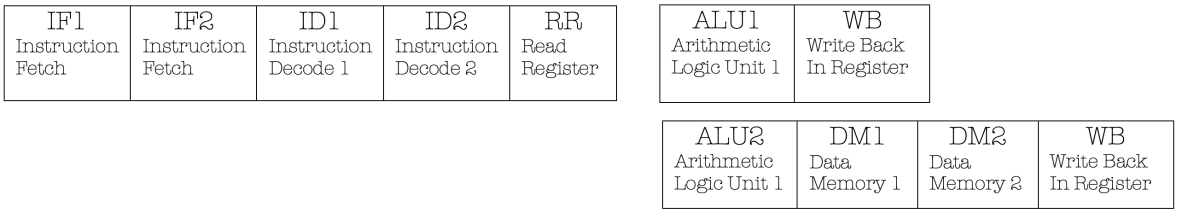

Fig : A 7-9 Stage Pipeline

Figure 1: The Pipelines

§1.2 Comparison Between Plots

§1.2.1 5 Stage Pipeline :

We designed a 5 stage pipeline simulator in part A and B of the assignment without bypassing and with bypassing. We allowed data to be forawrded from EX and MEM stages to ID stage in the latter. We made the simulators by using array of length 5 and moving instructions inside the array until it was completely empty.We implemented the bypassing by introducing flags and moving the output from latches between the two

cycles. The following graphs show the cycles taken by the two simulators on the 4 public test cases.

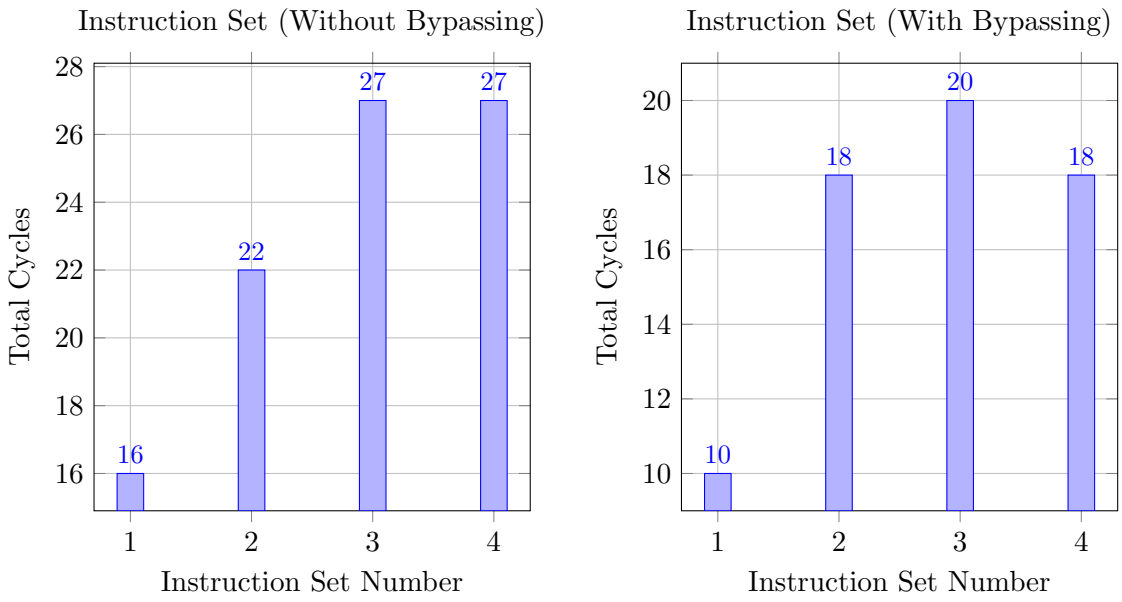

Figure 2: 5-stage pipeline

§1.2.2 7-9 Stage Pipeline :

In part C of the assignment, we implemented a 7-9 stage pipeline as described above, we did so using vectors as the pipeline was of varying size. we added an attribute of current stage to the vector and ran a while loop until the entire pipeline was empty. In the second part where we had to implement the bypassing we implemented it in the same manner as in 5 stage using flags and moving data from latches between the cycles. In this we had flags for ALU1, DM2 and WB forwarding to RR. e following graphs show the cycles taken by the two simulators on the 4 public test cases.

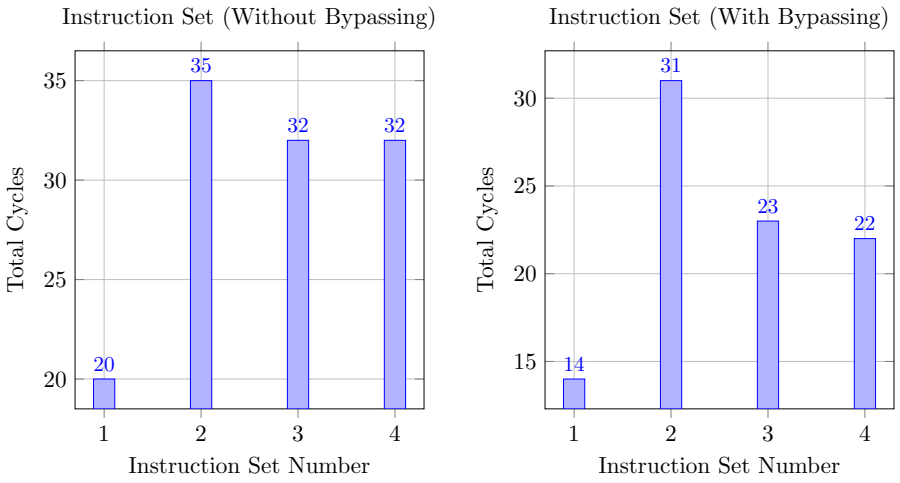

Figure 3: 7-9 stage pipeline

§1.2.3 Observations :

We observe from the following comparison plots that bypassing/forwarding significantly reduces the number of cycles required to implement the same pipeline that would have been rather stalled because the input was not loaded in the data/memory/register.

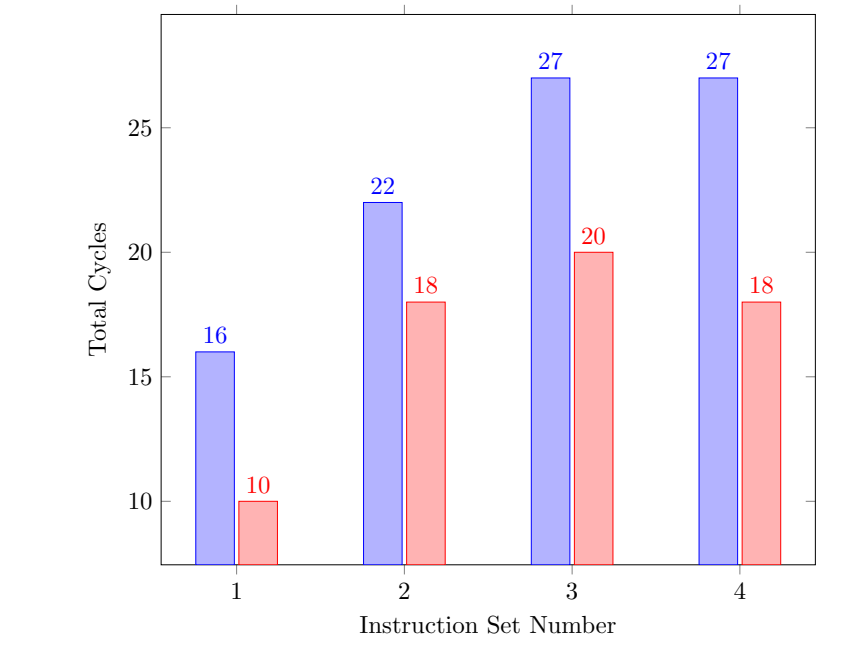

Figure 4: 5 stage pipeline

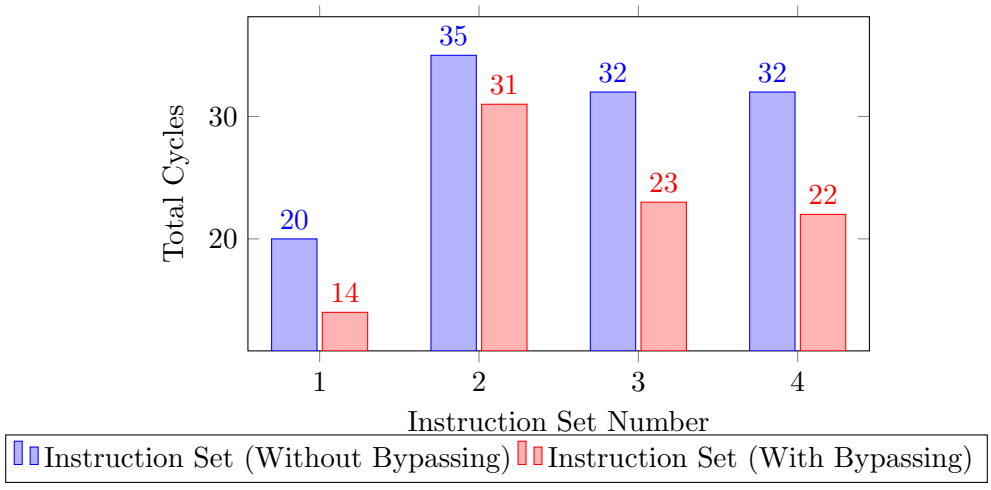

Figure 5: 7-9 stage pipeline

§2 Part E : Branch Predictor

§2.1 Saturating Branch Predictor

We implemented a Saturating Branch Predictor by making a table of size 2^{14} consisting of 2-bit counters implemented as bitset, with the states set as (00 : strongly not taken , 01 : weakly not taken , 10 : weakly taken , 11 : strongly taken). We take the branchtrace as input, consisting of a 32 bit branch address and taken/not taken. In the predictor we extracted the 14 LSB and then predicted the next state depending on the value in 2-bit counter at the address in the table and then updated the counter depending on the correctness of the prediction. We initialised the counters to each of the 4 states of the FSM to see the accuracies and we obtained the following results.

Percentage Accuracy for different Starting States			
Initial State	Correct Predictions	Mispredictions	% Percentage Accuracy
00	433	115	79.0146%
01	460	88	83.9416%
10	482	66	87.9562%
11	475	73	86.6788%

§2.2 BHR(Branch History Register) Branch Predictor

We implemeted a BHR Branch Predictor using the branch history of previous 2 commands and a table of 4 2-bit counters implemented as bitsets, with the states set as (00 : strongly not taken , 01 : weakly not taken , 10 : weakly taken , 11 : strongly taken). We take the branchtrace as input, consisting of a 32 bit branch address and taken/ not taken. In the predictor we have the history of the previous two commands and depending on the value of the same we access the counter in the table corresponding to it and predict a value based on it, after which we update the counter based on the correctness of the prediction. We initialised the counters to each of the 4 states of the FSM and the BHR bitset to see the accuracies and we obtained the following results.

Percentage Accuracy for different Starting States			
Initial State	Correct Predictions	Mispredictions	% Percentage Accuracy
00	392	156	71.5328%
01	396	152	72.2628%
10	398	150	72.6277%
11	399	149	72.8102%

§2.3 Fusion Branch Predictor

We implemented a Fusion Branch Predictor by combining the Saturating and the BHR , in this we used the branch history of the previous two registers along with the PC to obtain a combination table of size 2^{16} that took both as an input and produced a result.

Percentage Accuracy for different Starting States			
Initial State	Correct Predictions	Mispredictions	% Percentage Accuracy
00	416	132	75.9124%
01	445	103	81.2044%
10	479	69	87.4088%
11	469	79	85.5839%

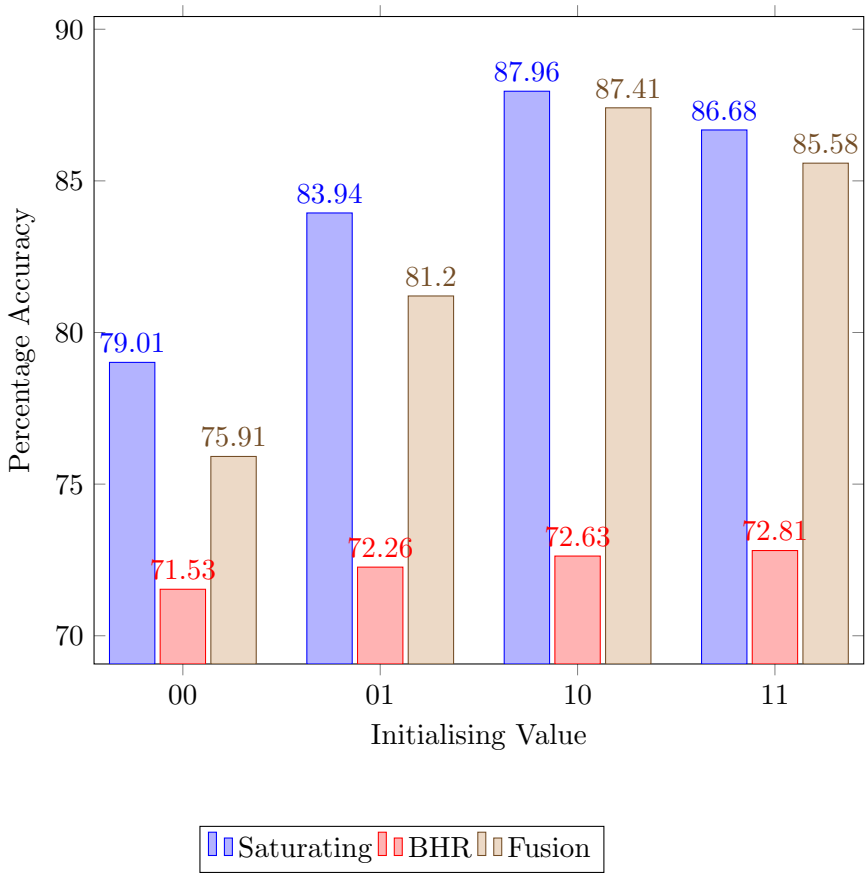

Figure 6: Percentage Accuracy of Different Predictors

§2.4 Observations :

We note that the fusion predictor has accuracy comparable to the Saturation Branch Predictor in nearly all cases, this is possibly due to the fact that our test set has only 548 value. When run on large testcases with more inputs the Fusion predictor will perform better. We also note that the accuracy of a Saturating Branch Predictor is better than the BHR Branch Predictor by approximately 10%.

§3 Conclusion :

The work split in this assignment has been:
Rajat Golechha : 2021CS10082 : 50%
Harshit Gupta : 2021CS10552 : 50%