COL216 Assignment-2 Report

1 Introduction

1.1 Part A, B, C

The aim of part A,B and C of this assignment is to simulate the pipelined MIPS processor in cpp consisting of 5 stages and 7-9 stages given that bypassing/forwarding is either allowed or not allowed.

We have assumed the first in first out (FIFO) principle in all cases when an instruction writes back. Further, we consider that for the 5-stage pipeline, WB happens in the first half cycle and the remaining stages happen in the second half cycle. However, this is not the case for the 7-9 stage pipeline in which all stages take an entire cycle. Thus, 'RW' and 'RR' cannot overlap. The branch instructions 'beq' and 'bne' are assumed to compute the output branch in 'EX' for 5-stage and 'ALU' in 79-stage whereas 'j' in 'ID' for 5-stage and 'ID2' in 79-stage respectively. The next instruction starts only after computation branch that is in the cycle after the above mentioned ones.

Let us first consider the 5-stage pipeline without bypassing. It has only data hazards and we need to stall until 'WB' and 'ID' are in same cycle if there is no bypassing. In the case of 5-stage with bypassing, stalls may come when the production and consumption stage occur in the same cycle or when consumption occurs earlier. In this case, a few stalls may be introduced. In all other cases, forwarding ensures that stalls are not needed.

In the 7-9 stage pipeline without bypassing, we need to insert stalls for data hazards so that the 'RR' stage occurs in the cycle after 'RW' and for structural hazards when a 7-stage instruction is present after an 'lw' instruction because it doesn't fit the FIFO rule mentioned above. Some stalls may also be introduced in program with bypassing but values are forwarded in most cases.

The plots for each of the above 4 cases and a combined plot can be found later in the report.

1.2 Part E

The aim of part E is this assignment is to look at and analyze the accuracies of various branch predictors namely 2-bit saturating counters and branch history registers (BHRs). We also try to use a combination of the two and observe if there is any significant improvement in the accuracy.

The table containing the various accuracies can be found later in the report.

2 Plots

The results after running the programs on the public_tests is presented in Table 2. The cycles required by each public_test given a particular program is shown below.

Program	Public_Test_Number	Clock Cycles
5stage	Public_Test1	16
	Public_Test2	22
	Public_Test3	27
	Public_Test4	27
5stage_bypass	Public_Test1	10
	Public_Test2	18
	Public_Test3	20
	Public_Test4	18
79stage	Public_Test1	21
	Public_Test2	36
	Public_Test3	32
	Public_Test4	34
79stage_bypass	Public_Test1	15
	Public_Test2	32
	Public_Test3	24
	Public_Test4	24

Table 1: Program and Testcases with corresponding Clock Cycles

The data from Table 2 is plotted in the graphs in Figure 1, 2, 3 and 4.

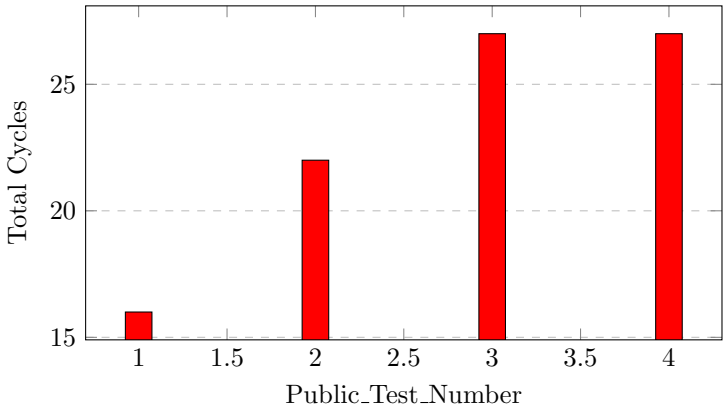

Figure 1: 5-stage Pipeline without Bypassing

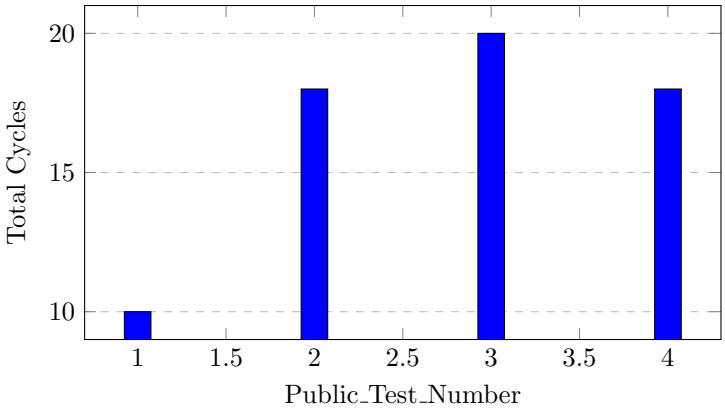

Figure 2: 5-stage Pipeline with Bypassing

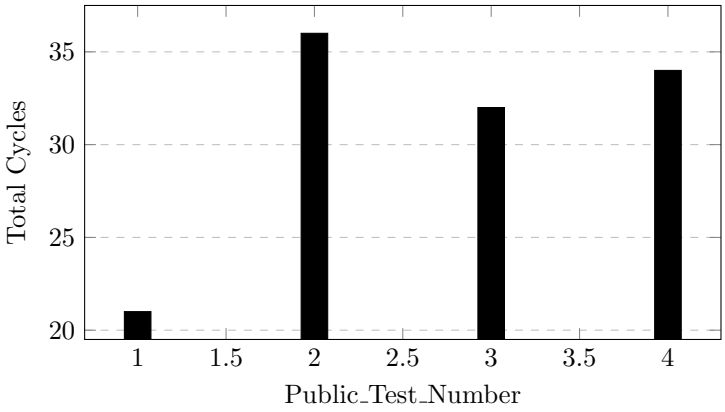

Figure 3: 7-9 stage pipeline without Bypassing

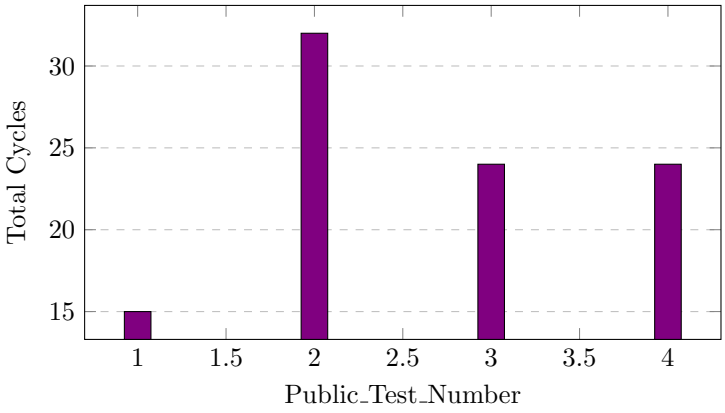

Figure 4: 7-9 stage pipeline with Bypassing

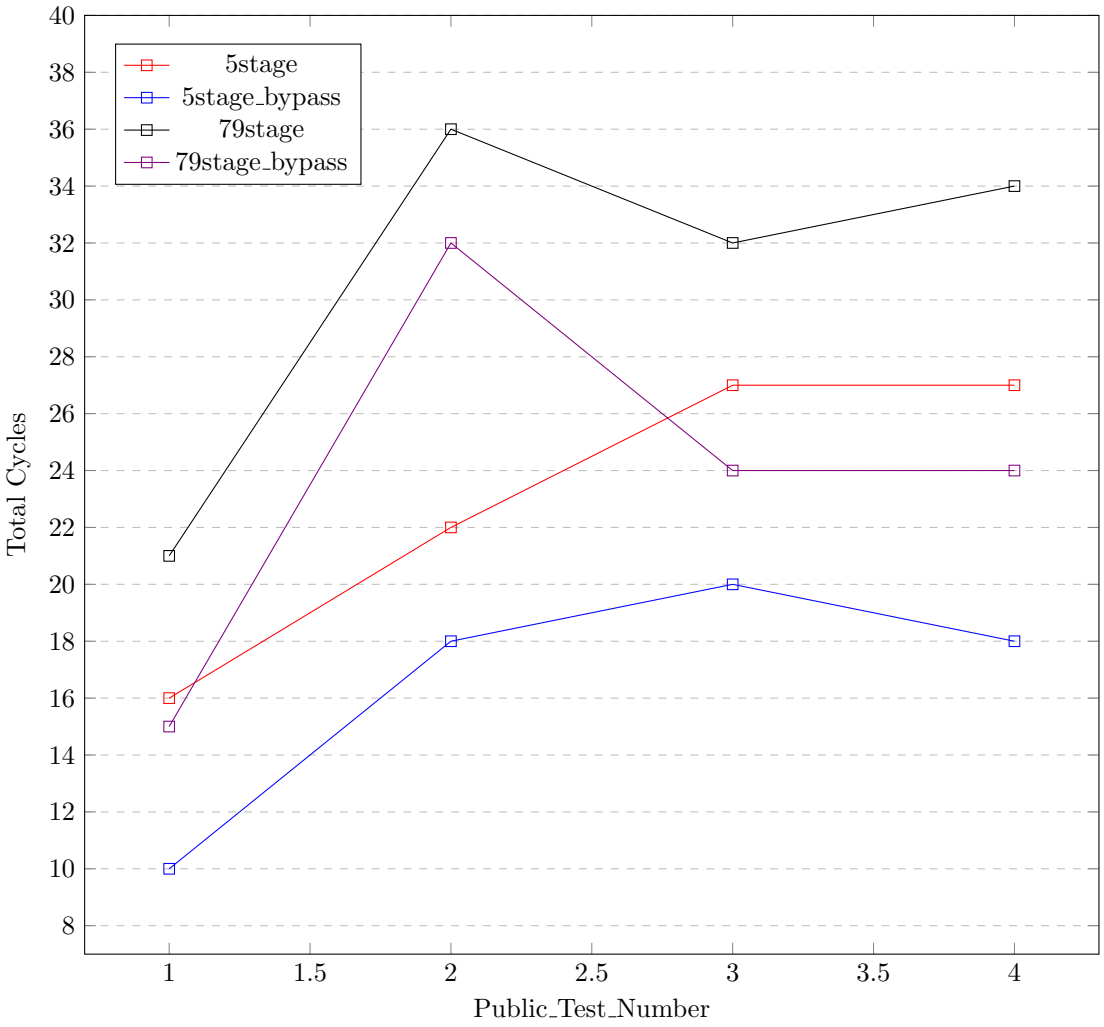

Figure 5: All Programs in same figure

3 Plot Conclusions

Let us now look at the key observations based on the above plots which corelate the stages of a pipeline, bypassing and the clock cycles. Note that the clock cycles are not directly indicative of the running time of the program. The time taken for an individual clock cycle is just as important.

1. **Forwarding**: We observe from the plots that the number of clock cycles taken in the without bypassing case is always more than that of the with bypassing case for both the 5-stage as well as 79stage pipeline. This is because we no longer need to wait for the produced values to actually be written into the register and can access them immediately after their computation.

2. **Program Runtime**: Note that if we fix the bypassing part (whether or not it is allowed) then the 79stage seems to be taking more clock cycles as compared to the 5-stage pipeline. This does not imply that the runtime of the program is higher. This is because the increase in the number of stages from 5-stage to 79stage causes a decrease in the individual clock cycle time.

4 Accuracy Table

Predictor	Initial Value	Accuracy
2-Bit Saturating Branch Predictor	0	78.83%
	1	83.75%
	2	87.73%
	3	86.49%
BHR Branch Predictor	0	71.35%
	1	72.08%
	2	72.44%
	3	72.62%
Saturating BHR Branch Predictor	0	79.01%
	1	85.76%
	2	85.21%
	3	84.12%

Table 2: Predictors and Initial Values with Accuracies

1. **2-Bit Saturating**: The 2-bit saturating counter works on the idea that the chance of a particular branch repeating depends on whether or not the same branch was taken earlier in the program.
2. **BHR**: The BHR predictor works on the idea that the chances of taking a branch depends on whether or not branches were being taken recently in the program flow.
3. **Combination**: The combined predictor works by combining both the ideas mentioned above that is a particular branch has a chance associated with being taken or not depending upon whether or not the recent say n (2 in our case) branches were taken. Let us look into the implementation used for the above accuracies. We used a vector 'combination' of size 2^{16} in which the number (0-3) associated with a branch and the previous 2 branches (taken or not) can be found at say index i which consists of 16 bits. The first two bits of index i come from the 2 branches whereas the remaining come from the 14 bit address of the branch.
4. **Initial Values**: The initial values of the saturating counter/register affects the accuracies as shown above. For the 2-bit saturating counter, the accuracy is maximum at 2 (we can see that the percentages are greater for 1 and 2 as they allow quick change). For the BHR the percentages are more or less same for all initial values because it affects only first two jumps.

5 Contributions

The test folder containing the cpp, hpp, Makefile and report PDF files has been zipped and submitted in the specified format. The files can be run by using the Makefile. The Team consists of:

- Santhosh Rishi Deshineni(2021CS10564)
- Pothamsetty Chethan Manogna Sai(2021CS10561)

Contributions	**Santhosh**	**Chethan**
Total	**50%**	**50%**

Table 3: Contributions