Indian Institute of Technology Delhi

COL216 - Computer Architecture

Assignment 2

Processor simulation with 5 and 7-9 stage pipeline and Comparative analysis of different Branch Prediction Algorithms

Garv Nagori, Aryan Sharma
2021CS10549, 2021CS10553

Contents

§1 Introduction

This assignment aims to simulate the working of processor for the MIPS Instruction Set. We have implemented the processor with various methods of pipelining namely five stage, five stage with forwarding, 7 or 9 stage, and 7 or 9 stage with forwarding. We also test various heuristic branch prediction strategies and analyze their behaviour over initializing values and the nature of the program being executed.

§2 Five Stage Pipelining

Five stage pipelining has been implemented with and without bypassing.

§2.1 Without Bypassing

(a) WB happens in the first half cycle and the rest stages happen in the second half cycle.

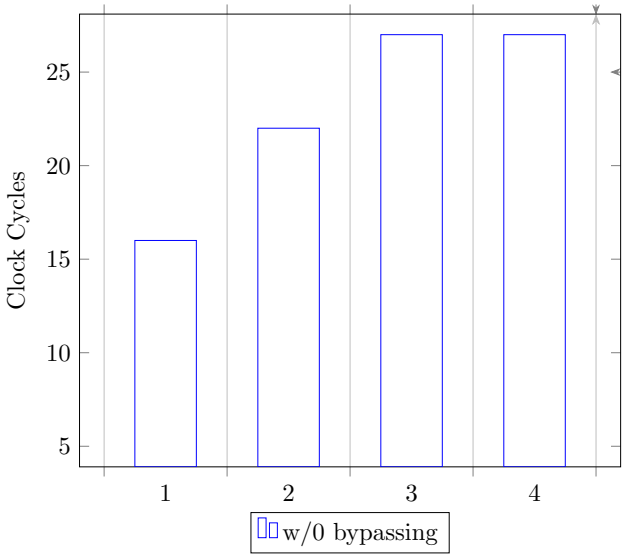
(b) 'beq' and 'bne' computations are completed in Ex stage whereas in the case of $'j'$ the computations are completed in ID stage.

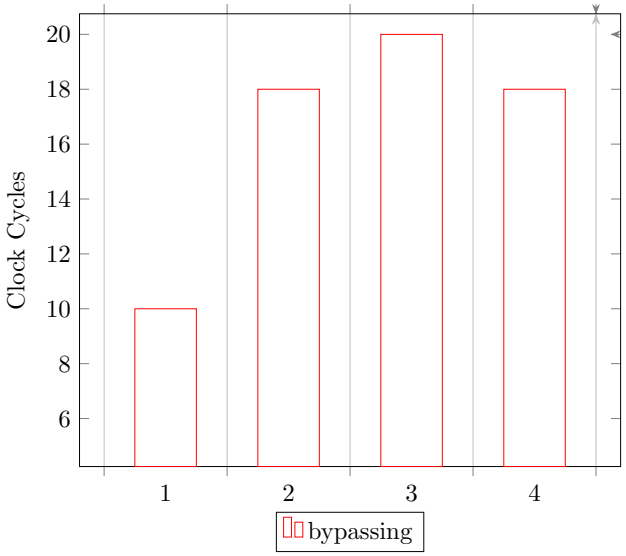
§2.2 With Bypassing

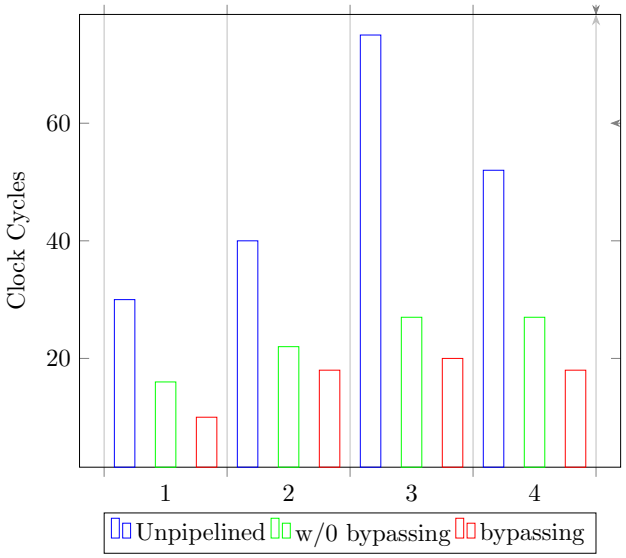
The assumptions are the same as in the case of Five Stage without bypassing.
A comparative relation between the number of cycles taken for execution between Unpipelined, Five Stage without Bypassing, and Five Stage with Bypassing.

Processor type	Public Test	Clock Cycles
Unpipelined	1	30
	2	40
	3	75
	4	52
Five Stage(w/o bypassing)	1	16
	2	22
	3	27
	4	27
Five Stage	1	10
	2	18
	3	20
	4	18

Testing program for simulations on different programs


Clock Cycles
25
20
15
10
5
1
2
3
4
w/0 bypassing


Clock Cycles
20
18
16
14
12
10
8
6
1
2
3
4
bypassing


Clock Cycles
60
40
20
1
2
3
4
Unpipelined
w/0 bypassing
bypassing

§3 7 or 9 Stage Pipelining

7 or 9 stage pipelining has been implemented with and without bypassing.

§3.1 Without Bypassing

The procedure for implementation is the same as that for Five Stage pipelining. We have implemented the 7 or 9 stage pipeline with the following assumptions:

(a) All stages take one complete cycle in 7 or 9 pipeline.

(b) If a 7-stage instruction comes after a 9-stage instruction, we stall for two cycles, even if only one instruction uses the write port at a time.

(c) lw happens in DM_1 and sw in DM_2.

(d) Control unit processes instruction in Id_1. 'j' instruction is checked in Id_2 and also data hazards are detected in Id_2 for stalling.

(e) 7 after a 9 is also checked in Id_2.

(f) RW and RR in the same cycle with dependency will stall for one cycle.

§3.2 With Bypassing

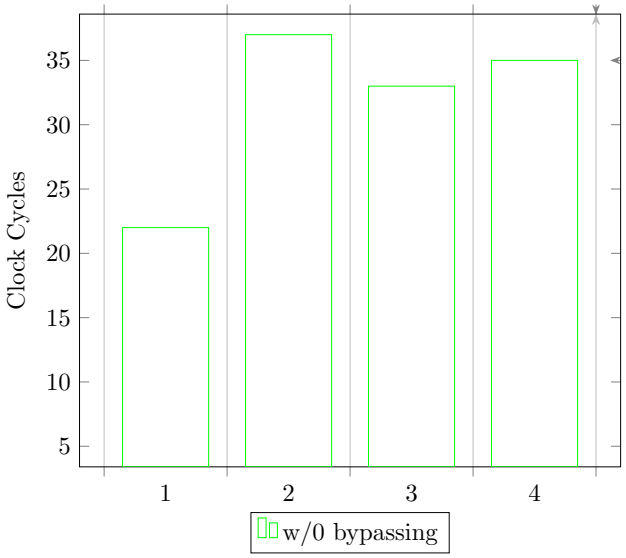
The implementation, in this case, is identical to the above except for the following extra assumptions:

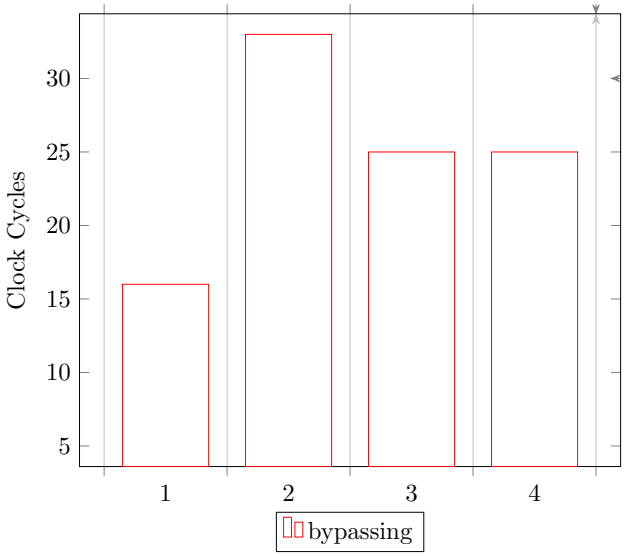
(a) In case of dependency no forwarding, RR will read the correct value because it is assumed to forward to Ex stage.

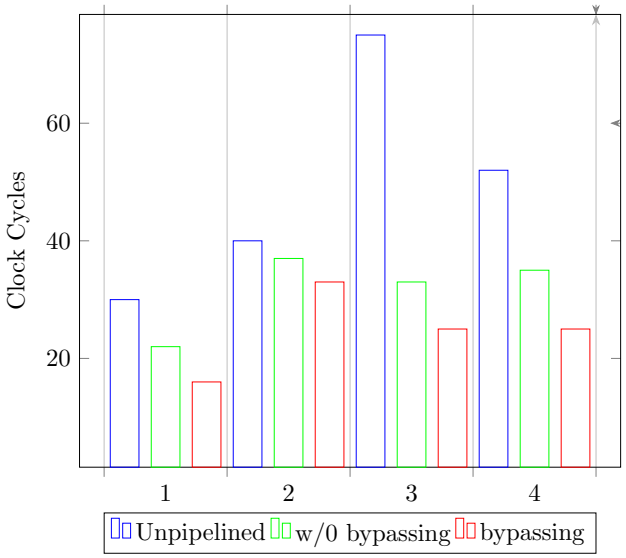
(b) The register between DM_1 and DM_2 cannot forward data or receive forwarded data. So forwarding only happens to the register between Ex and DM_1.
A comparative relation between the number of cycles taken for execution between Unpipelined, 7 or 9 Stage without Bypassing, 7 or 9 Stage with Bypassing.

Processor type	Public Test	Clock Cycles
Unpipelined	1	30
	2	40
	3	75
	4	52
7 or 9 Stage(w/o bypassing)	1	22
	2	37
	3	33
	4	35
7 or 9 Stage	1	16
	2	33
	3	25
	4	25

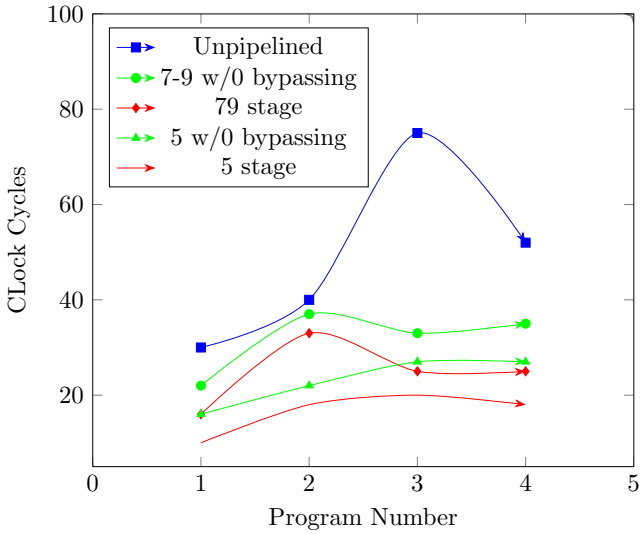
Testing program for simulations on different programs


Clock Cycles
35
30
25
20
15
10
5
1
2
3
4
w/0 bypassing


Clock Cycles
30
25
20
15
10
5
1
2
3
4
bypassing


Clock Cycles
60
40
20
1
2
3
4
Unpipelined
w/0 bypassing
bypassing

In conclusion, we present a comparative graph between all programs in a single plot.

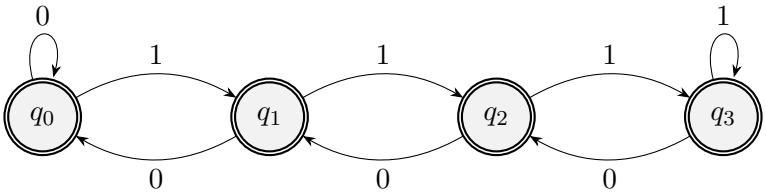


§4 Branch Prediction

We have mainly examined three different heuristic methods, as follows, these strategies take various different aspects of how programs are usually written to improve on accuracy.

§4.1 Saturation Counter

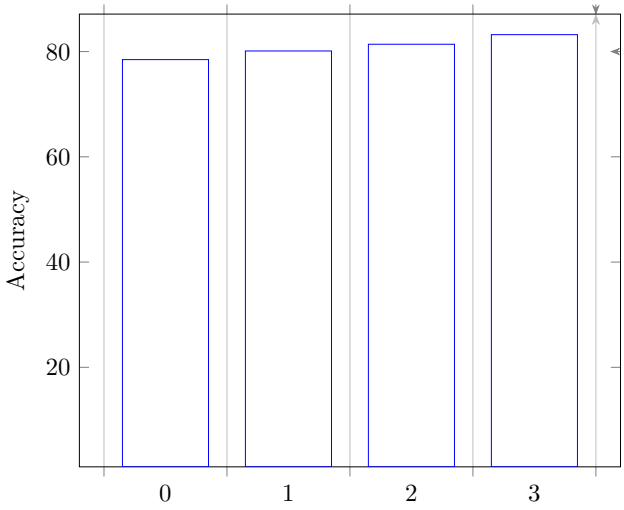
In this algorithm, we create a 2-bit program counter corresponding to each of the 14 least significant bits of all possible 32-bit addresses. The transition diagram for the counter is as shown below



This gives us quite good accuracy on the sample program provided with address and branch result **("branchtrace.txt")**. We also have the liberty of deciding what initial state to start with. The accuracy of this depends on the particular order of the program instructions. In case of the input file **"branchtrace.txt"** the accuracy are as follows:

Initial State	Accuracy(%)
q_0	78.47
q_1	80.11
q_2	81.39
q_3	83.21

Saturation Counter Accuracy

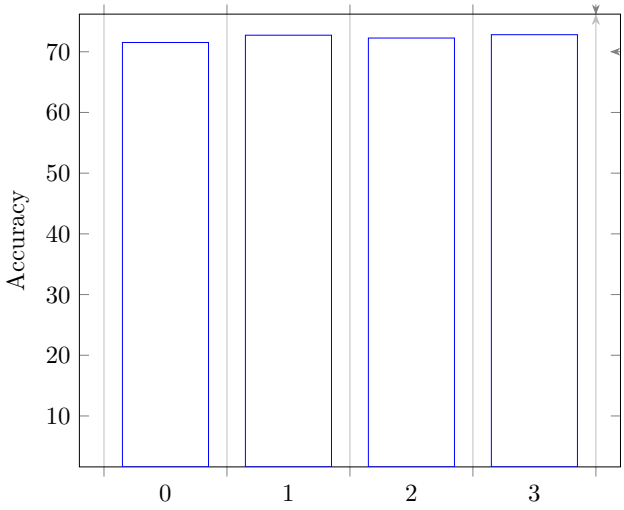


§4.2 Branch History Register(BHR)

This method focuses more on global repetitive patterns for the branch being taken or not taken, we keep account of four different two-bit saturation counters and one two-bit history register, each state of the history register corresponds to a saturation counter, and the accuracy with this algorithm is relatively less. In case of the input file **"branchtrace.txt"** the accuracy are as follows:

Initial State	Accuracy(%)
q_0	71.53
q_1	72.63
q_2	72.26
q_3	72.81

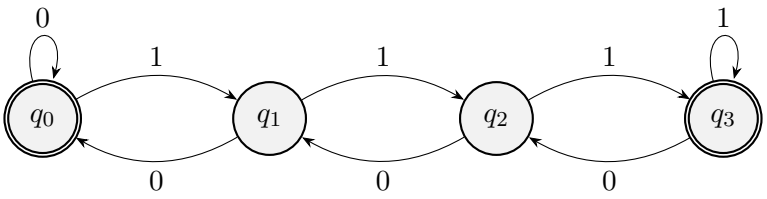
BHR Accuracy



§4.3 Branch History Register along with Saturation Counter

In order to improve upon the accuracy of the previous two strategies, we came up with a scheme which takes into account ideas from both of the systems. We make a new data structure(namely combination), which is an array of size 2^16 this takes input as two bits of BHR register and fourteen **LSB(Least Significant Bit)** of address in the instruction. We change the DFA for Saturation

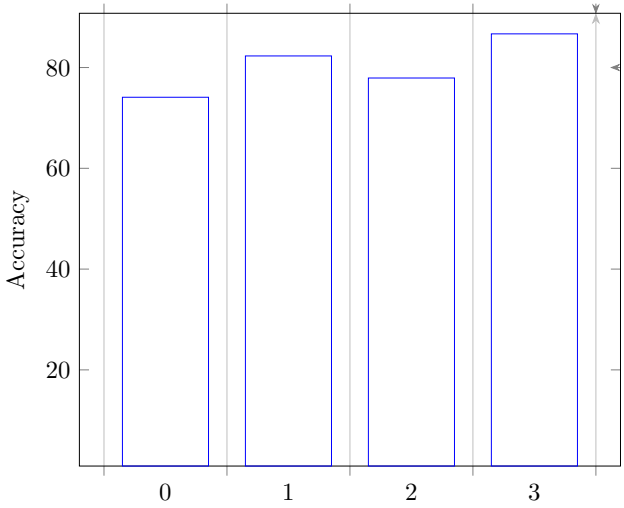
counter by making only q_0 and q_3 as accepting, which are strongly not taken and strongly taken in saturation counter.



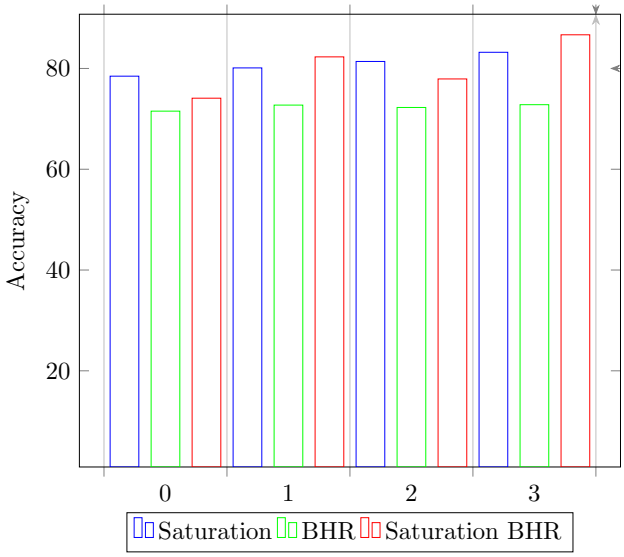
If we are at a non-accepting state, then we check the global patterns using BHR(our modified BHR has saturation counters as shown above), if the state in BHR is also non-accepting then we look up in combination and return the corresponding answer.

Initial State	Accuracy(%)
q_0	74.10
q_1	82.30
q_2	77.92
q_3	86.68

BHR Saturation Counter Accuracy



We show a comparative representation between the accuracies of all three in the plot below



§5 Token Distribution

Regarding the distribution of tokens, both of us have worked equally on the assignment. Even in quantitative measures, we have contributed almost equally to the point distribution for the different parts and checkpoints. So, we have decided to distribute 20 tokens uniformly (10 each). The list of the aspects each member has worked on is not strict since most of the time, we were working collaboratively, but if we still had to distribute the elements worked on, we would do according to the significant contributions as follows:

Name	Tokens	Aspect worked on
Aryan Sharma	10	Branch Prediction, Five stage without forwarding, LaTeX report
Garv Nagori	10	Five stage, 7 or 9 stage

Token Distribution

§6 Acknowledgements

We have used the style file from here[1] to produce this document.

[1] https://github.com/vEnhance/dotfiles/blob/main/texmf/tex/latex/evan/evan.sty