TDTS08: Lab Report

Lab 2: Instruction Pipelining

Name	PIN	Email
Alexander Yngve	930320-6651	aleyn573@student.liu.se
Pål Kastman	851212-7575	palka285@student.liu.se

Contents

1 Introduction

The purpose of this lab is to learn how instruction pipelining works and how branch prediction affects the performance of the pipeline.

2 Pipeline basics I

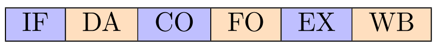

Figure 1 – Six stage pipeline.

LB instruction

IF fetch instruction in memory.

DA activate different parts in the cpu depending on the instruction.

CO calculate the address in the memory where the operand is stored.

FO fetch operand from memory.

EX not used.

WB write operand to register.

ADD instruction

IF fetch instruction in memory.

DA activate different parts in the cpu depending on the instruction.

CO calculate the address in the memory where the operand is stored.

FO fetch operand from memory.

EX compute addition.

WB write result to register.

The main difference for the two instruction is that LB doesn't need to use the instruction execute (EX) state, since its only loading data from the memory.

3 Pipeline basics II

When we have a short pipeline we get less time penalty due to that its only one step that needs redoing and therefore its detected earlier.

	1	2	3	4	5
1	IF	EX			
2		IF	EX		
3			IF	EX	
4				IF	EX

Figure 2 – Ideal pipeline operation.

	1	2	3	4	5	6
1	IF	EX				
2		IF	EX			
25			IF			
4				IF	EX	
5					IF	EX

Figure 3 – Pipeline operation during conditional jump.

4 Branch prediction

Here we analyze how the different branch prediction algorithms perform.

4.1 Desciption

For each predictor a benchmark was run according to the following command

```
sim-outorder -bpred predictor ~/TDTS08/bin/go.ss 3 8
```

The different algorithms works as following:

TAKEN	Assumes the jump is always taken.
BIMOD	Address of the branch instruction is used as a key into an associative memory where a two bit saturating counter is stored, when branch is taken the counter is incremented and vice versa. The branch will look on the msb to determine direction.
2LEV	Current branch instruction has a set of saturating counters which is choosen depending on the history of the branch, this history is stored in a shift register.
COMB	Combination of bimod and 2lev.
PERFECT	This one cheats and always chooses the correct branch.
NOTTAKEN	Assumes the jump is never taken.

4.2 Result

The performance result can be seen in figure 4 below.

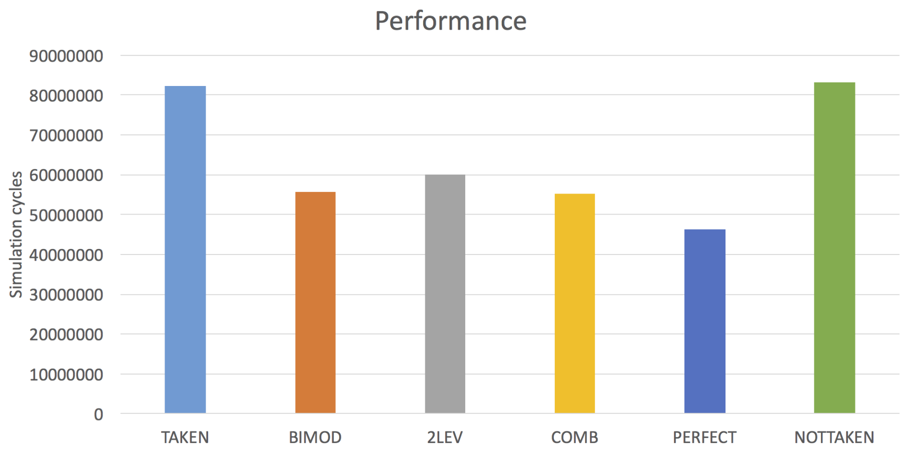

Figure 4 – Performance of the different branch prediction algorithms

In figure 4 we can see that the *not taken* algorithm performs worst. If we look at how the algorithms perform according to each other, and we originate from the *not taken* algorithm since it performs the worst, we then get a result according to figure 5. The results in figure 5 are based on the direction rate metric (bpred_dir_rate).

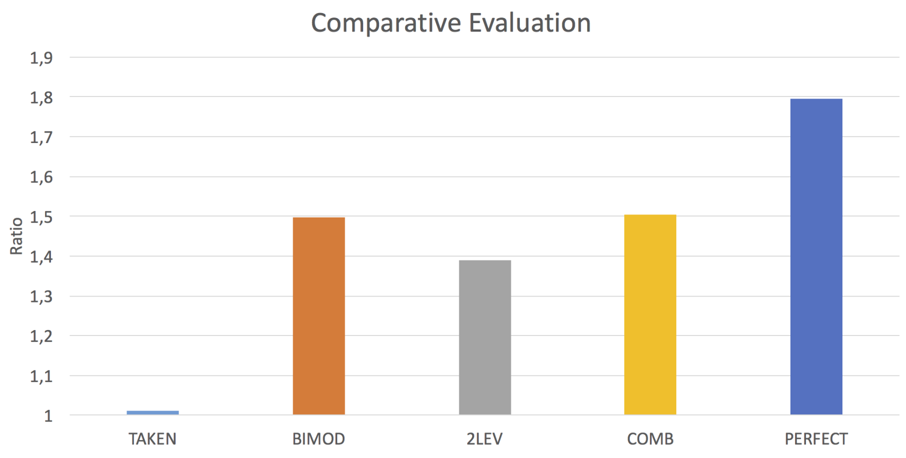

Figure 5 – Performance of the alogorithm according to the not taken algorithm who performed worst.