Lab Report 10

Computer Architecture Lab

201651015 Dipansh Khandelwal

201651008 Aman Singh

15th March 2019

Branch prediction rate versus CPI

	Taken		Bimod		Comb		Perfect	
Benchmark	**Branch pred. Rate**	**CPI**	**Branch pred. Rate**	**CPI**	**Branch pred. Rate**	**CPI**	**Branch pred. Rate**	**CPI**
Anagram	0.3520	2.1562	0.9539	1.4251	0.9818	1.3912	1.3688	0.3520
Go	0.3214	2.5928	0.8432	2.1907	0.8509	2.1841	2.0713	0.3214
Compress	0.1172	2.0910	0.9744	1.7234	0.9790	1.7215	N/A	1.7102
Applu	0.3375	1.5854	0.8050	1.3580	0.9334	1.3371	N/A	1.3250

For the four branch prediction schemes ' taken|perfect|bimod|comb ', describe the predictor.

- **What information the predictor stores (if any)?**
  - In the architecture the predictor stores all the last predictions made and the corresponding output after making those predictions. Using this data it further decides how and what prediction to make in any situation it faces from then.
- **How the prediction is made?**
  - The prediction is made using the information stored about all the last predictions made and the corresponding output after making those predictions. It goes with the specific choice which have the best throughput in that situation referring to the information stored before.
- **What the relative accuracy of the predictor is compared to the others.**
  - Shown by the table above.

Choosing a new branch strategy

Suppose you are choosing a new branch strategy for a processor. Your design choices are:

1. predict branches taken with a branch penalty of 2 cycles and a 1200 MHz clock-rate
2. predict branches taken with a branch penalty of 3 cycles and a 1300 MHz clock-rate
3. predict branches using bimod with a branch penalty of 4 cycles and a 900 MHz clock-rate
4. predict branches using bimod with a branch penalty of 4 cycles and a 1000 MHz clock-rate and half the L1 cache size.

Benchmark		**Alternative 1**	**Alternative 2**	**Alternative 3**	**Alternaive 4**
Anagram	**Sim_Cycle**	12898607	12898607	12898607	12898607
	CPI	0.6935	0.6935	0.6935	0.6935
	Exe_Time	14908.6399	20642.7322	39756.3731	35780.7358
Go	**Sim_cycle**	44263366	44263366	44263366	44263366
	CPI	1.4104	1.4104	1.4104	1.4104
	Exe_Time	104018.91	144026.183	277383.76	249716.206

Execution time = (Sim_cycle * CPI / frequency) * branch-penalty

Execution time is in Milliseconds

Questions:

- **What would be your choice for your benchmark? Why?**
    - Our choice of the benchmark will be Anagram because its overall execution time is quite less compared to go benchmark.

- **How much do you have to be able to increase the clock frequency in order to gain performance when allowing a branch miss-prediction latency of 3 cycles instead of 2 when using the taken predictor?**

    - For branch miss-prediction latency of 3
        - IPC: 0.4768, CPI: 2.0973
        - Total no. of instructions: 26647438
        - Simuation time in cycles: 39008444
        - Total clock cycles needed = Total no. of instructions/IPC
          = 26647438 / 0.4768 = 55,888,083.1
        - Frequency = Total Clock cycles / Simulation time
          = 55,888,083.1 / 39008444 =1.43272

    - For branch miss-prediction latency of 2
        - IPC: 0.5154, CPI: 1.9403
        - Total no. of instructions: 26632696
        - Simuation time in cycles: 36087816
        - Total clock cycles needed = Total no. of instructions/IPC
          = 26632696/ 0.5154 = 51,673,837.8
        - Frequency = Total Clock cycles / Simulation time
          = 51,673,837.8 / 36087816 = 1.43089

    - To gain the same performance we have to make IPC's same. Therefore, for miss-penalty of 3 clock cycles,

      Total clock cycles needed = Total no. of instructions / IPC of miss penalty 2

      = 26647438 / 0.5154 = 51,702,440.8

      Therefore, new frequency = 51,702,440.8 / 39008444 = 1.3254

In-order vs out-of-order issue

Experiment 1.1

Experiment with in-order and out-of-order issue, and the width of the pipeline, by running the simulation with the following combinations of parameters. Measure CPI and total no of cycles.

Benchmark		**Pipeline width = 1**		**Pipeline width = 4**		**Pipeline width = 8**	
		Sim_cycle	**CPI**	**Sim_cycle**	**CPI**	**Sim_cycle**	**CPI**
Anagram	**In-order**	25459061	1.3688	22477029	1.2085	22307980	1.1994
	Out-of-order	25094502	0.7412	12898607	0.6935	8157056	0.4386
Go	**In-order**	65004957	2.0713	60519289	1.9284	58668083	1.8694
	Out-of-order	61258936	1.9520	44263366	1.4104	39385817	1.2550

Questions:

- **What is the impact on CPI of the increased pipeline width?**
  - With an increase in pipeline CPI decreases for both in-order and out-of-order execution.
- **Explain the impact and their difference for both in-order and out-of-order issue.**
  - As shown in the table, CPI of In-order execution is greater than CPI of out-of-order execution for both the benchmarks in all the pipeline width. This is self-explanatory as for in-order execution the pipeline has to wait till the execution of previous instruction is completed or not. And as the width of pipeline increases the CPI decreases.

Experiment 1.2

Run the sim-outorder simulator varying the number of memory ports available: 1,2 and 4. Use a pipeline width of 4.

Bench mark		**Memory port=1**		**Memory port=2**		**Memory port=4**	
		Sim_cycle	**CPI**	**Sim_cycle**	**CPI**	**Sim_cycle**	**CPI**
Anagram	**In-order**	22307980	1.1994	22307980	1.1994	22307980	1.1994
	Out-of-order	9218080	0.4956	8157056	0.4386	7987681	0.4295
Go	**In-order**	58668083	1.8694	58668083	1.8694	58668083	1.8694
	Out-of-order	40214541	1.2814	39385817	1.2550	39270778	1.2513

Questions:

- **What is the impact on CPI of the increase in available memory ports?**
  - There is no change in CPI for In-order execution by changing the number of memory ports but for out-of-order execution the CPI decreases that is processing becomes faster.