HW 2 – Return Address Predictor

沈昱宏, 110705013

***Abstract*—In this homework assignment, I analyze the branch prediction unit (BPU) in the Aquila RISC-V core and its effects on CoreMark benchmark performance. I explored the impact of disabling the BPU and modifying the Branch History Table (BHT) size, examining changes in CoreMark scores. Additionally, branch prediction statistics, such as hit and miss rates for different branch types, are evaluated. Finally, a return address prediction architecture is proposed to improve accuracy.**

Keywords—Aquila, CoreMark, BPU

I. DISABLE BPU

A. *Disabling BPU*

I disabled the BPU by fixing the branch_hit_o (the output signal of BPU) to 0. Before disabling the BPU, the total cycle count was 2558316, and after disabling the BPU, the total cycle count increased to 2887697, showing a 12% increase in execution time.

B. *Changing Entry Num*

I changed the EntryNum from 64 to 256. The result shows that the total cycle count further reduced to 2547614, which means that using more memory did help the total cycle count to reduce.

II. BRAHCH STATISTICS

A. *Statistics*

I count the total number on hit and miss of different functions under the original setting. The hit rate is calculated by number of hit / total amount. The result shows that using a 4x memory can improve hit rate. Since the jal command is not often, the BHT is more likely to store address of branch commands, causing a lower hit rate.

TABLE I. STATISTICS

Function		**Metric**		
		Hit	***Miss***	***Hit Rate***
EntryNum = 64	jal	1200	39127	3 %
	branch	58586	253512	18.8 %
EntryNum = 256	jal	3216	37111	8 %
	branch	63420	248675	20.3 %

B. *Counting mispredictions*

I counted the number of mispredictions, which is the output from the execution stage. Under EntryNum = 64, the total number of mispredictions is 23882, which is 40% of table total hit, indicating that the branch prediction mechanism can be improved (the one implemented now is simply one-level branch predictor).

III. RAP DESIGN

In this section, I will briefly introduce how I design my return address prediction unit.

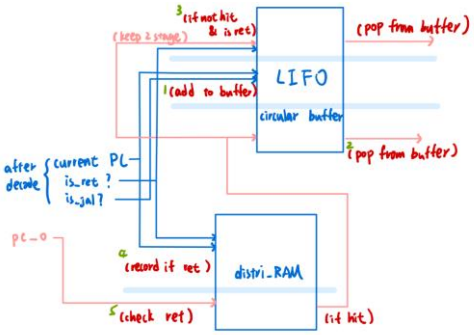

Fig 1. Design diagram

A. *I/O*

As you can see from the graph, I will use pc_o, pc / is_ret / is_jal after decoding as input. For output, I will use the return_address, and whether it is ret as output.

B. *Overall design*

I will implement my return address prediction unit with a LIFO circular buffer, which stores the return addresses. Additionally, I store the address of jal instructions into a distri_RAM. When a jal instruction is decoded, we will add the current PC+4 into the buffer (marked 1 in the graph). When a ret instruction is decoded (marked 3) or the pc_o is found to be a ret instruction (marked 2), the buffer will be popped, and the address will be sent to the PC. If a new ret instruction is found, it will be stored in the distri_RAM (marked 4). If the address of pc_o is found in the distri_RAM (table hit), it will pop the last element in the buffer and use it as output. I will use two signals as the output of RAP.