ECEN 5593 Advanced Computer Architecture

Evaluation Report

October 2021

Team Members: Pranav Bharadwaj, Malcolm McKellips

**Project Evaluation:**

This report shall highlight the details regarding the tools used for our implementation, the methodology of evaluation (branch prediction performance) and the metrics for the same.

**Tools:**

Visual studio code is the primary platform of choice for evaluation. We use the PlatformIO extension to talk to the softcore synthesized on the board. This allows for viewing register files, counter values, memory and stack values. The extension also allows us to step over individual assembly instructions, viewing change in registers in real-time.

The FPGA board communicates via UART over USB, which is a half-duplex connection. This allows for interaction of either hardware-level control or software at any given time. Once the .bit file for the softcore image is downloaded to the board via Vivado, we switch over to VSCode for software evaluation.

**Methodology:**

The softcore we are uses implements a version of GSHARE, a 2-bit global correlating predictor. Thus, it has a branch history for every branch instruction encountered (within the table size). When the processor image is downloaded to the board for the very first time, the branch history is non-existent, and the branch prediction is always 50% accurate on the first run. Our methodology is to track the position of the program counter and stack pointer values as the instruction fetch unit encounters a branch instruction. We use a basic assembly routine with a loop containing a heterogenous set of instructions for benchmarking. The following steps follow:

1. Store the program counter value for the start of the assembly routine
2. Store the program counter value before executing the branch instruction in a register
3. Track the program counter value after pushing the branch instruction into the pipeline
4. If the program counter value matches the start of the loop subroutine, a successful prediction has occurred. If the value matches the register, the program counter has reverted back, indicating a misprediction.

**Benchmark structure:**

The assembly subroutine employed would be a mix of arithmetic and branch instructions in a loop. This allows for additional instructions to be fed into the pipeline following a prediction, which would either be executed or flushed from the pipeline depending on whether the prediction was successful or not.

A sample of the assembly routine would be as follows:

.globl main

main:

# Register t3 is also called register 28 (x28)

li t3, 0x0                  # t3 = 0

REPEAT:

addi t3, t3, 6          # t3 = t3 + 6

addi t5, t3, -1         # t3 = t3 - 1

andi t6, t3, 3          # t3 = t3 AND 3

beq  zero, zero, REPEAT # Repeat the loop

addi t7, t3, 4

addi t4, t6, 4

  nop

.end

The arithmetic instructions would be loaded into the pipeline following the branch instruction, which would then be executed or flushed depending on the prediction result. For evaluation metrics, we view the branch predictor table by dumping it onto the terminal. We then vary the number of loops to compare the performance based on the branch history size.

**Project progress:**

Currently, we have the processor image flashed onto the FPGA board, communicating with it via VSCode and the PlatformIO extension to view register and memory contents when one-stepping over instructions. We would currently rate this at 65% completion. The tasks remaining as follows:

1. Finalize the assembly subroutine to be used as the benchmark code
2. Print and tabulate program counter and register values to terminal for observing the branch predictions in real-time
3. Vary the number of runs of the assembly subroutine and compare the performance between different branch history sizes and predictor flavors.