Computer Architecture
EECE7352
Prof. Kaeli

Homework 3:

Part A: (10 points)

For any two current embedded processors (e.g., ARM7, ARM9, Intel 8051, TI MSP430, MIPS32, RISCV E7 or S7), provide the details of the microarchitecture of the pipeline (e.g., the width of the data path, the number of pipeline stages, number of instructions issued per cycles, the number of integer units, the branch resolution unit, the names of the pipeline stages, etc.). You might not find every one of these details but provide at least 3 different pipeline characteristics that you can compare between two embedded CPUs. Make sure to cite your sources.

Part B: (40 points)

We are providing you with an instruction trace named itrace.out.gz (the file has been compressed with gzip, so you will need to decompress it before using it). The trace is from tracing the execution of gcc from the SPECint2000 benchmark suite. The file contains 16 million conditional branches. The trace file simply lists the 64-bit instruction address for each conditional branch instruction executed (addresses are in decimal), followed by an indication of either taken (T) or not taken (N). Here is a snippet of the trace:

3086632129 N
3086632133 T
3086632129 N
3086632133 T

The uncompressed trace is over 300MB in size, so keep it gzipped when not using it.

Your assignment is to write a simulator to model two simple conditional branch predictors. The first predictor will have 64 1-bit predictors, saving the action of the last branch outcome to predict the outcome of the next branch. The second predictor will use 32 2-bit counters to predict branches. Assume initially that the 2-bit counters are initialized to be weakly not-taken.

Write a simple simulator that models the 2 predictors and processes the instruction trace provided. Submit both the code for your branch predictor simulator (on Canvas) and report on the number of buffer misses (first time taken branches), and the number of correct and incorrect predictions for this trace for each predictor. Add a discussion explaining why one predictor does better than the other.

As an added step, experiment with other prediction algorithms (besides a 2-bit counter). You only have 16 entries in total for these alternative algorithms. The best performing algorithm will receive **25 extra credit points**.

Part C: (25 points)

To build a better understanding of pipelining and superscalar concepts, complete problems C.1 (parts a-b) and C.7 (parts a-b) in the textbook in Appendix C, and problems 3.1, 3.2, 3.3 and 3.18 from Chapter 3. Make sure to state any and all assumptions you make when answering these problems.

Part D: (25 points)

The following is x86 assembly. Write equivalent code in C and RISC-V assembly. (It will probably be easier for you to write the C code first, and then the RISC-V assembly equivalent.) For C code, include a mapping between the variables you use and the x86 registers; for RISC-V code, include a mapping between the RISC-V registers you use and the x86 registers. Your code should run on the BRISC-V simulator. Remember to only use the RV32I instruction subset.

```
        .text
        .globl main

main:
        movl $0x0, %ebx
        movl $0x0, %ecx
        jmp  here
tloop:  mov  %ecx, %eax
        add  %eax, %ebx
        addl $0x1, %ecx
here:   cmpl $0x63, %ecx
        jle  tloop
ret
```

Part E:
Extra credit: (30 points) – added to your quiz grade

Find a tool to generate a data address trace for the linpack.c program run on X86. Only collect data addresses for all load and store instructions. Vary the size of the matrix (the value of N in line 44 in the linpack.c code). Identify how many unique pages of memory (assume a 2KB page size) are touched by data references in your code. Plot the number of unique pages versus the value of N (careful, making N too large will cause problems for the hardware). How are you going to get this done? You could modify a simulator that can save the trace, or you can use a profiling tool that provides you with the ability to generate the traces and record the address for all loads and stores. This is your choice. Provide details on what tool you used.