362M Assignment 4 Lab Report

Client:
Computer and Electrical Engineering Department

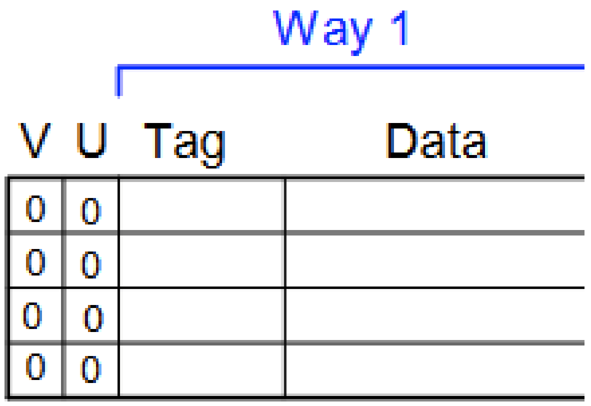
Instructor:

Roger Chamberlain[1] - roger@wustl.edu
Michael Hall[2] - mhall24@wustl.edu

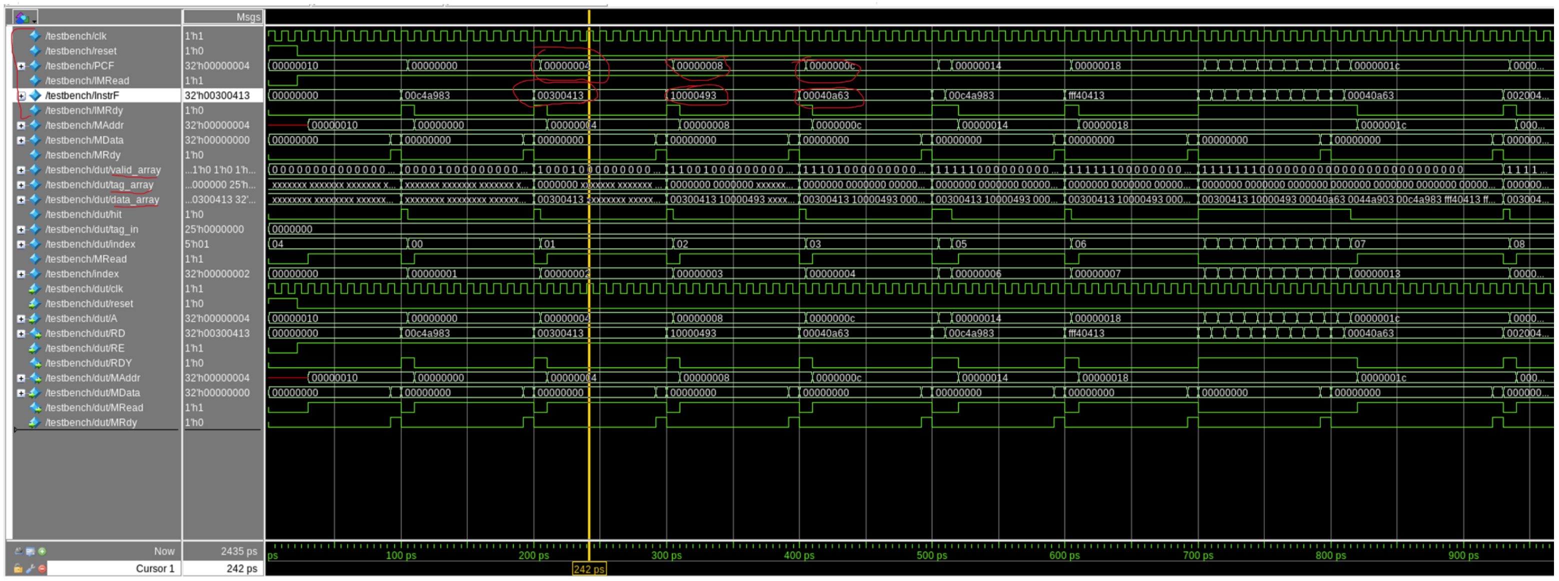
Engineers:
Edgar Sarceno[3] - e.sarceno@wustl.edu

[1] **Dr. Roger D. Chamberlain, PhD**: Lecturer, Department of Computer Science & Engineering
[2] **Dr. Michael Hall, PhD**: Lecturer, Department of Computer Science & Engineering
[3] **Edgar Sarceno**, candidate for B.S. in Computer Engineering, M.S. in Computer Engineering

1. Indicate approximately how much time you spent on the assignment
- I spent approximately 15-20 hours on this assignment over the course of two weeks, including office hours, and individual work.
2. A diagram of the internal design of your instruction cache.



- The cache implemented in the code is a direct-mapped instruction cache with a capacity of 128 bytes, consisting of 32 cache lines each storing one 32-bit word (block size of 4 bytes). It operates without a finite state machine, using simple sequential logic to handle cache hits and misses, updating cache entries upon misses by fetching data from main memory and relying on the testbench to manage read operation timing.
3. Simulation Waveform.



```
                              Msgs
/testbench/clk                1'h1
/testbench/reset              1'h0
/testbench/PCF                32'h00000004
/testbench/IMRead             1'h1
/testbench/InstrF             32'h00300413
/testbench/IMRdy              1'h0
/testbench/MAddr              32'h00000004
/testbench/MData              32'h00000000
/testbench/MRdy               1'h0
/testbench/dut/valid_array    ...1'h0 1'h0 1'h...
/testbench/dut/tag_array      ...000000 25'h...
/testbench/dut/data_array     ...0300413 32'...
/testbench/dut/hit            1'h0
/testbench/dut/tag_in         25'h0000000
/testbench/dut/index          5'h01
/testbench/MRead              1'h1
/testbench/index              32'h00000002
/testbench/dut/clk            1'h1
/testbench/dut/reset          1'h0
/testbench/dut/A              32'h00000004
/testbench/dut/RD             32'h00300413
/testbench/dut/RE             1'h1
/testbench/dut/RDY            1'h0
/testbench/dut/MAddr          32'h00000004
/testbench/dut/MData          32'h00000000
/testbench/dut/MRead          1'h1
/testbench/dut/MRdy           1'h0

Now       2435 ps
Cursor 1   242 ps
```

Figure 1. Simulation Waveform.

- Here, the circles indicate the relevant data to look at (center of Figure 1.), as well as the ordered specified signals (left of Figure 1.). The red circles indicate two signals, PCF and InstrF. Here PCF is the pc counter, and InstrF is the instruction address of the testbench assembly code that is sent as machine code. Here, the pc counter and the InstrF coincide with the program count and the specific address that we are running. Consider the first pair of red circles in Figure 1. The top circle illustrates value 32’h00000004, and the second bottom circle indicates value 32’h00300413. Given the ‘*testaddresses.txt*’ file, after the initial address of 0, the test bench proceeds to 32’h00000004. This coincides with the first instruction in the given file ‘*testcode.txt*’, which is :

    - 00000000 00300413 addi s0, zero, 3.

- Here ‘00000000 00300413’ is the machine language address of the instruction addi and its register inputs/outputs. This is exactly the address displayed in my simulation waveform through the InstrF signal. This indicates that my program reads these addresses accurately, and indicates that my system operates correctly when driven by the testbench.
- The following arrays are related to my cache:
    - *valid_array*, This array holds a valid bit for each cache line, indicating whether the data in that cache line is valid and can be used for cache hits.
    - *tag_array*, This array stores the tag portion of the memory address for each cache line, which is used to determine if an incoming address matches the cached data during a lookup.
    - data_array, This array contains the actual instruction data (32-bit words) stored in each cache line, which is returned to the processor on a cache hit.
- The program works correctly because the cache implementation accurately manages the validity and contents of each cache line using the *valid_array*, *tag_array*, and *data_array*, ensuring that instruction fetches correspond precisely to the addresses specified in the *testaddresses.txt* and *testcode.txt* files. This is evidenced by the simulation waveforms where the PCF and InstrF signals match the expected program counter values and instruction data, demonstrating that the cache correctly handles hits and misses and provides the correct instructions to the processor, confirming proper operation when driven by the testbench.

4. Github Repo:
   https://github.com/B1WAYN3/ComputerArchitecture.git
   (Note: Navigate to the Homework4 folder).