CSE/ESE 362M Computer Architecture

Assignment 4, Due Friday, Nov. 22, 2024

In this lab, you will design and code a direct-mapped instruction cache for a RISC-V processor. For extra credit, you have the option of also designing and coding a direct-mapped data cache (i.e., adding support for a write operation).

1. Pipelined RISC-V Processor

Figure 1 shows the complete pipelined processor from the textbook.

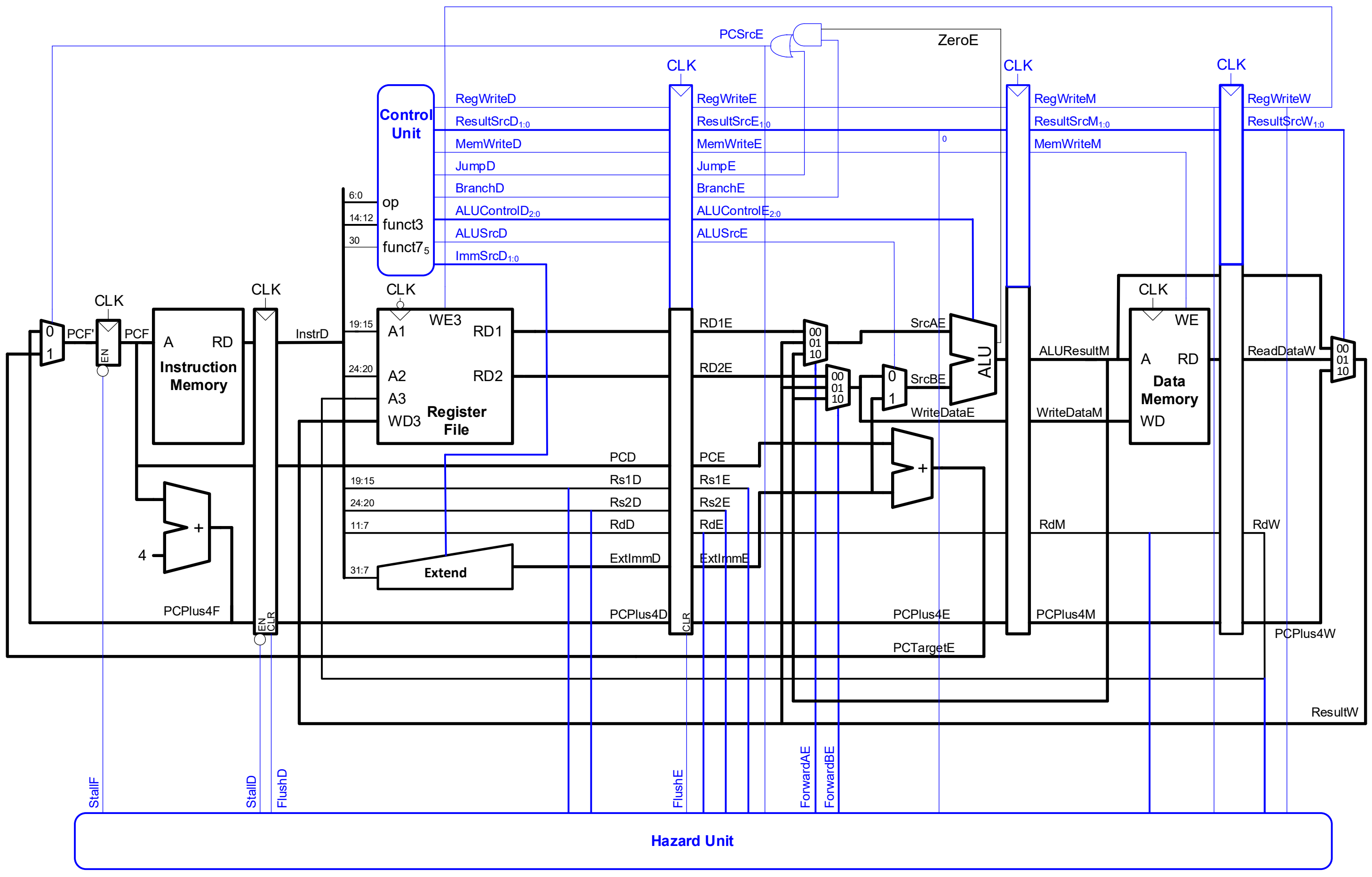


Figure 1. Pipelined RISC-V processor.

2. Cached Instruction Memory

Don't panic, we are going to focus just on the instruction memory. In the development of the pipelined datapath, we assume that the instruction memory returns a result in one clock cycle. In reality, this will not be the case, and we must account for the fact that (1) the number of cycles for the instruction memory to provide a result is more than one, and (2) the number of cycles for the instruction memory to provide a result is variable (i.e., takes a different number of cycles depending upon whether there is a hit or a miss in the instruction cache.

Figure 2 shows a closeup of the design around the area of the instruction memory. The original design is shown in black, and new signals are shown in blue.

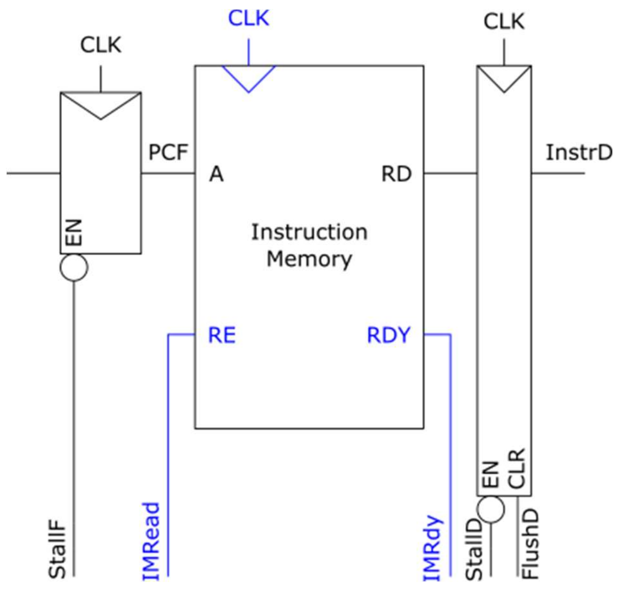


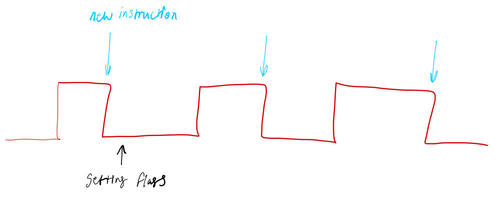
Figure 2. Closeup of instruction memory.

The original signals StallF, StallD, and FlushD all come from the Hazard Unit. A new signal (IMRead) initiates a read of the instruction memory. Another new signal comes from the instruction memory (IMRdy) that is 1 when the data from the instruction memory is ready and 0 when the data is not ready. As the cache operation will take multiple clock cycles to operate, we also provide CLK to the instruction memory.

Figure 3 looks inside the instruction memory, showing that it is actually constructed of an instruction cache (which is the target of the assignment) and a main memory (that has the same interface as the instruction memory, A (32-bit address) and RE (1-bit read enable) inputs in addition to RD (32-bit read data) and RDY (1-bit ready) outputs.

The boundary of the instruction memory of Figure 2 is shown as a dashed line in Figure 3.

Your task is to design, code (in SystemVerilog), and simulate the instruction cache of Figure 3. Testbench code is provided for the environment (generating a sequence of PCF and IMRead signals) and for the main memory. The cache should have a capacity of 128 bytes, and it should be direct mapped, with a block size of 1 word. As was the case in lecture, it is OK for the cache to only operate on words (i.e., ignore byte accesses, all addresses will end in 2b'00).



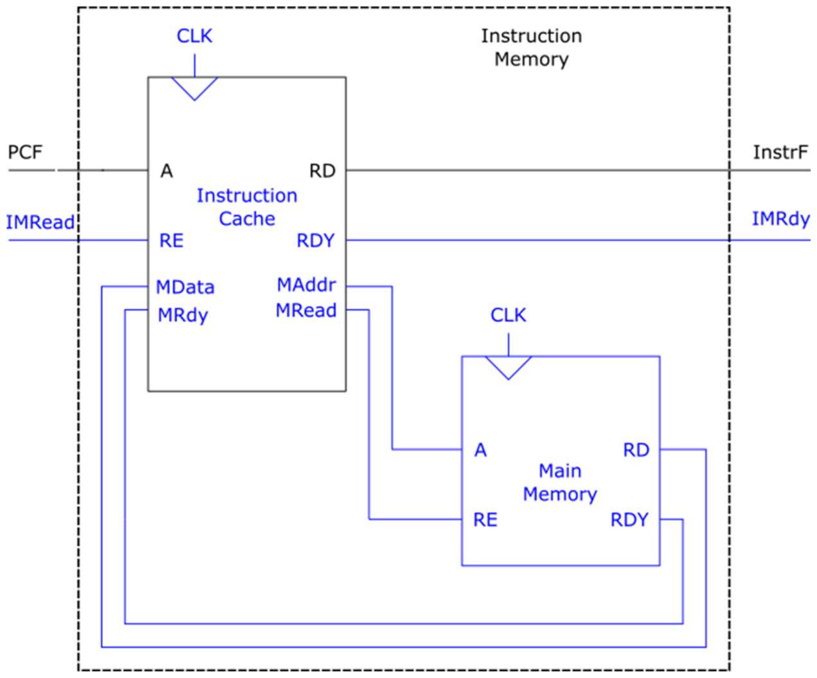


Figure 3. Internals of instruction memory.

What to Turn In

Generate a written report that includes the following elements:

1. Indicate approximately how much time you spent on the assignment
2. A diagram of the internal design of your instruction cache.
3. Simulation waveforms (in the order listed here: clk, reset, PCF, IMRead, InstrF, IMRdy, MAddr, MRead, MData, and MRdy – all 32-bit values displayed in hexadecimal for ease of reading). Also show appropriate internal signals (e.g., valid bits, cache contents, etc.). You need not show all of the internal signals, just regions of interest. Does your system operate correctly when driven by the testbench? Circle or highlight the waves showing that the correct value is output on InstrF, and make sure it is legible.

Please indicate any bugs you found in this assignment writeup, or any suggestions you would have to improve it.

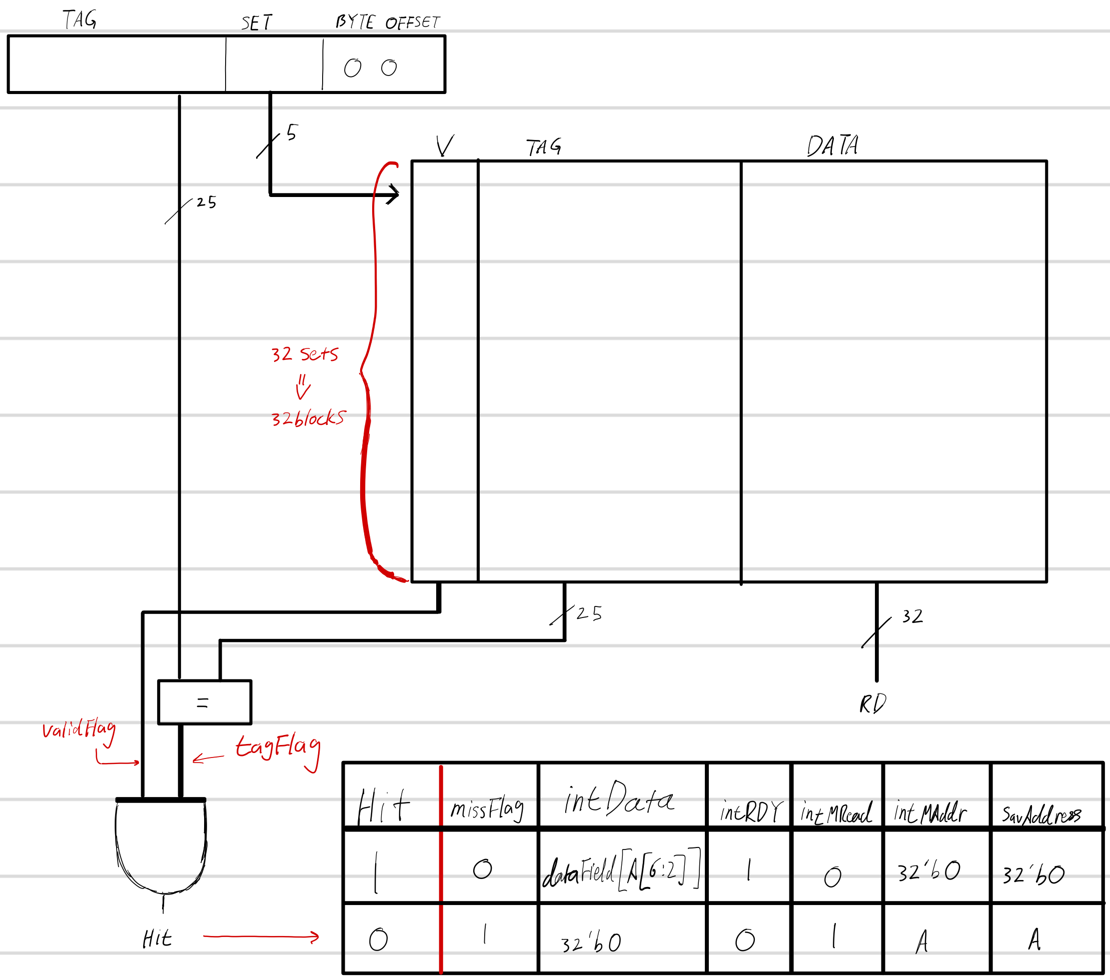
Extra Credit Option

If you would like to earn extra credit, first make sure that your instruction cache design is operating correctly.

You can earn extra credit by designing, coding, and testing a data cache, following the same approach as the instruction cache above. The data cache works similarly to the instruction cache except that now it needs to support both loads and stores. Writes to the data cache can be write-through, no allocate. I.e., a write that is a miss in the cache simply writes to main memory and doesn’t impact the cache. A write that hits in the cache updates both the data value in the cache and the value stored in main memory.

#2

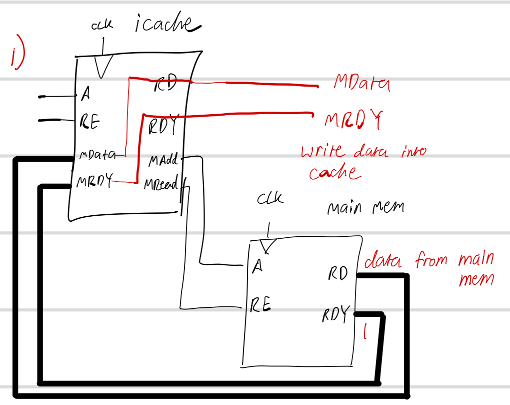
Combinational Logic



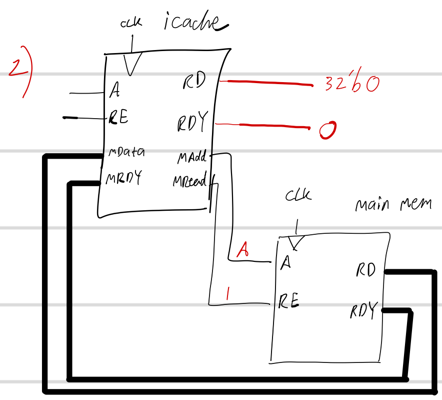
Hit	missFlag	intData	intRDY	intMRead	intMAddr	SavAddress
1	0	dataField[A[6:2]]	1	0	32'b0	32'b0
0	1	32'b0	0	1	A	A

Output to the Sequential block or the main memory.

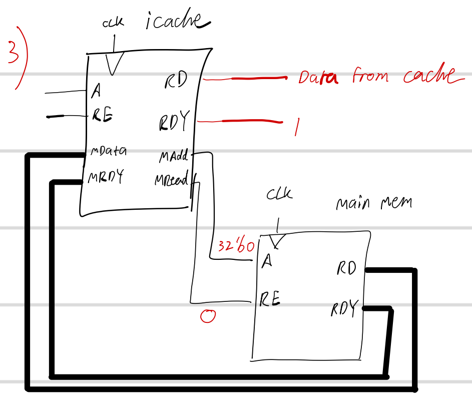
Sequential logic : 3 possibilities : 1) Data from main mem, 2) Miss, and 3) cache has the data.



1)
clk
icache
A
RE
mData
MRDY
RD
RDY
MAdd
MRead
MData
MRDY
write data into cache
clk
main mem
A
RE
RD
RDY
data from main mem
1



2)
clk
icache
A
RE
mData
MRDY
RD
RDY
MAdd
MRead
32'b0
0
clk
main mem
A
1
A
RE
RD
RDY



3)
clk
icache
A
RE
mData
MRDY
RD
RDY
MAdd
MRead
Data from cache
1
clk
main mem
32'b0
0
A
RE
RD
RDY