ECE/CS 552: INTRODUCTION TO COMPUTER ARCHITECTURE

Project Description – Stage 3

Due on April 28/29 (at project demo), 2018

In stage 3 of this project, the task is to add a cache hierarchy to interface with the **5-stage pipeline** from phase 2.

The major work in stage 3 is the implementation of the cache modules – I cache and D cache, and the cache controller which allows interaction between the caches and the processor pipeline, and the caches with the memory.

Either Modelsim or Icarus should be used as the simulator to verify the design. **You are required to follow the Verilog rules as specified by the rules document uploaded on canvas. NOTE: the only exception to this rule is the required use of *inout* (tri-state logic) in the register file. Do not use *inout* anywhere else in your design.**

1. WISC-S18 ISA Summary

WISC-S18 contains a set of 16 instructions specified for a 16-bit data-path with load/store architecture.

The WISC-S18 memory is byte addressable, even though all accesses (instruction fetches, loads, stores) are restricted to halfword (2 byte), naturally-aligned accesses.

WISC-S18 has a register file, and a 3-bit FLAG register. The register file comprises sixteen 16-bit registers and has 2 read ports and 1 write port. Register $0 is hardwired to 0x0000. The FLAG register contains three bits: Zero (Z), Overflow (V), and Sign bit (N).

The list of instructions and their opcodes are summarized in Table 1 below. Please refer Phase 1 specs for specific details.

Table 1: Table of opcodes

Instruction	Opcode
ADD	0000
SUB	0001
RED	0010
XOR	0011
SLL	0100
SRA	0101
ROR	0110
PADDSB	0111
LW	1000
SW	1001
LHB	1010
LLB	1011
B	1100
BR	1101
PCS	1110
HLT	1111

2. Memory System

For this stage of the project, you are required to design the a) I Cache and D Cache modules, b) Cache controllers for reading and writing to the caches, c) Interfacing between the Caches and Memory, d) Interfacing the I Cache with the IF pipeline stage, e) Interfacing the D Cache with the MEM pipeline stage.

Verilog modules are provided for: a) multi-cycle main memory, b) Cache Data Array, c) Cache Meta-data Array.

You will load the main memory with the binary machine code instructions to be executed on your design (not your I Cache). You will use one set of a data array and a meta-data array for the I-Cache and another set for the D-cache.

3. Implementation

3.1 Cache/Memory Specification

a. The processor will have separate single-cycle instruction and data caches which are bye-addressable. Your caches will be 2KB in size, direct mapped, with cache blocks of 16B each. Correspondingly, the data array has 128 lines, each line being 16 Bytes wide. Further, the meta-data array has 128 lines and is 1 Byte wide. You are free to choose how/what information is stored in each line of the meta-data array – but it should contain at least the tag bits and the valid bit.

b. The cache write policy is write-through with write-allocate which means that on hits it writes to cache **and** main memory; on misses it finds the correct block in main memory and brings that block to the cache, and then re-performs the write (which will now be a cache hit – and so will parallel right to cache and memory).

c. The cache read policy is to read from the cache for a hit; on a miss, the data is brought back from main memory to the cache and then the required word (from the block) is read out.

d. The memory module is same as before, except for the longer read latency and a "data_valid" output bit .A 2-byte write to memory from the processor (write through cache) will take one cycle while 2-byte read from memory will take 4-cycles (though read requests can be placed to the memory on every cycle).

e. The cache modules will have the following interface: one 16-bit (2 Byte) data input port, one 16-bit (2-Byte) data output port, one 16-bit (2 Byte) address input port and a one-bit write-enable input port.

f. Note that interaction between the memory and caches should occur at cache block (16 Byte) granularity. Considering that the data ports are 2-Bytes wide, this would require a burst of 8 consecutive data transfers.

3.2 Cache Hits Reads/Writes

a. Cache hits take only one cycle to execute. Once the data/meta-data array line/word is identified based on the index/offset bits of the address, Data is read/written from/to the data array **and in parallel** the tag match is performed.

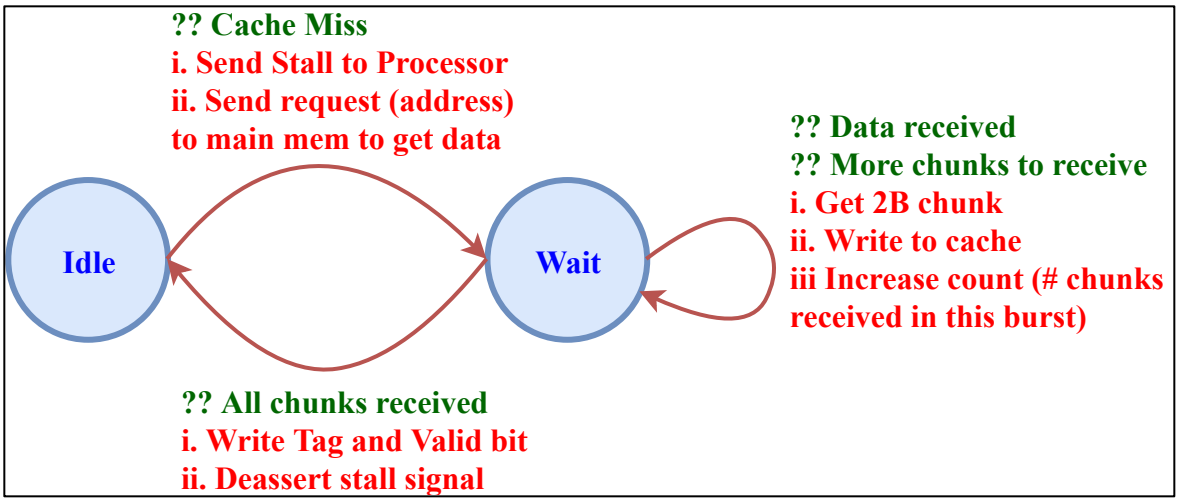
b. If the tag matches (i.e. hit) the read/write from/to the data array is valid. This data is returned via the cache data port in case of a read.

c. Being a write-through cache, all writes are written to main memory as well. As mentioned earlier, memory writes take only 1 cycle, so they will complete as and when a cache write completes (in case of a write hit).

d. In case of a tag mismatch (i.e. miss), the miss handler is triggered and the pipeline is stalled (if the data cache misses, all upstream instructions must stall) or nops are inserted (if it is an instruction cache miss).

3.3 Cache Miss Handler FSM

The sequence of events to be performed by the cache controller on a cache miss are shown in figure below. The blue refers the state, the green refers to the condition for changing state and the red refers to actions performed on state transitions.



Points to note:

a. On both read and write misses, you need to bring in the correct block from the memory to the cache.

b. A single data transfer between cache and memory is of an entire cache block and is thus 16 bytes but the data transfer granularity (i.e. width of data ports in cache/memory and wires between them) is only 2 byte words. So you will need to sequentially grab 8 chunks of data sequentially from memory and insert them word by word into the cache block.

c. Cache is write-through, so there is no data to be written back from cache to memory during explicitly during a miss that needs to be handled by the miss controller.

d. The cache controller stalls the processor on a miss, and only after entire cache block is brought into the cache, the new tag is written into the tag array, valid bit is set and the stall is deasserted.

3.4 Memory Contention on Cache Misses

The mechanism described above for handling misses is independent for the I-Cache and D-Cache. If requests from both caches are sent to memory only one can be handled at a time. You are required to implement an arbitration mechanism that select either one of the competing requests and gives it a grant to go through to memory. The other request stalls for an extended period. Note that you will never have multiple I-Cache and/or D-Cache misses in unison. At most you can have one of each type in parallel.

4. Interface

Your top level verilog should be file called ***cpu.v***. It should have a simple 4 signal interface: ***clk***, ***rst_n***, ***hlt*** and ***pc[15:0]***.

Signal Interface of ***cpu.v***		
Signal:	**Direction:**	**Description:**
clk	in	System clock
rst_n	in	Active low reset. A low on this signal resets the processor and causes execution to start at address 0x0000
hlt	out	When your processor encounters the HLT instruction it will assert this signal once it is finished processing the instruction prior to the HLT.

pc[15:0]	Out	PC value over the course of program execution.

5. Submission Requirements

1. You are provided with an assembler to convert your text-level test cases into machine level instructions. You will also be provided with test bench and test cases. The test cases should be run with the test bench and demonstrated at the demo on Apr 28/29.

2. You are also required to submit a zipped file containing: all the Verilog files of your design, all test benches used and any other support files.