

*ECE/CS 552: INTRODUCTION TO COMPUTER ARCHITECTURE*

## Project Description – Phase 3

Due on Monday, December 16, 2023, 11:59pm

In Phase 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 Phase 3 is the implementation of the cache modules (I-cache and D-cache) and the cache controller that allows interaction between the caches and the processor pipeline, as well as between the caches and 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-F24 ISA Summary [unchanged from Phase 1]**

WISC-F24 contains a set of 16 instructions specified for a 16-bit data-path with load/store architecture.

The WISC-F24 memory is byte addressable, even though all accesses (instruction fetches, loads, stores) are restricted to half-word (2-byte), naturally-aligned accesses.

WISC-F24 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 (N). Note that opcodes that update flags will always update flags, even if the destination register is R0. In other words, an operation like ADD, R0, R1, R2 will not change R0, and it's result should never be forwarded, but it must update the FLAG register(s).

The list of instructions and their opcodes are summarized in Table 1 below. Please refer to the Phase 1 handout for more details.

Table 1: Table of opcodes

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

## 2. Memory System [new information]

For this stage of the project, you are required to design a) I-cache and D-cache modules, b) cache controllers for reading and writing to the caches, c) interface between the caches and memory, d) interface between the I-cache and the IF pipeline stage, e) interface between the D-cache and 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 **main memory only** with the binary machine code instructions to be executed on your design (**I-cache is to be empty after reset**). You will use one instance of the data array the meta-data array for each way of each cache.

## 3. Implementation [new information]

### 3.1 Cache/Memory Specification [new information]

a. The processor will have separate single-cycle instruction and data caches, which are byte-addressable. Your caches will be 2KiB (i.e. 2048B) in size, 2-way set-associative, with cache blocks of 16B each. Correspondingly, the data array would have 128 lines in total, each being 16 bytes wide. The meta-data array would have 128 total entries composed of 64 sets with 2 ways each. Each entry in the meta-data array should contain the tag bits, the valid bit and one bit for LRU replacement.

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

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 the same as before, except for the longer read latency and a “data\_valid” output bit. A 2-byte write to memory from the processor will take one cycle, while a 2-byte read from memory will take 4 cycles. Memory is pipelined, so read requests can be issued to 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 signal.

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

### 3.2 Cache Hits Reads/Writes [new information]

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

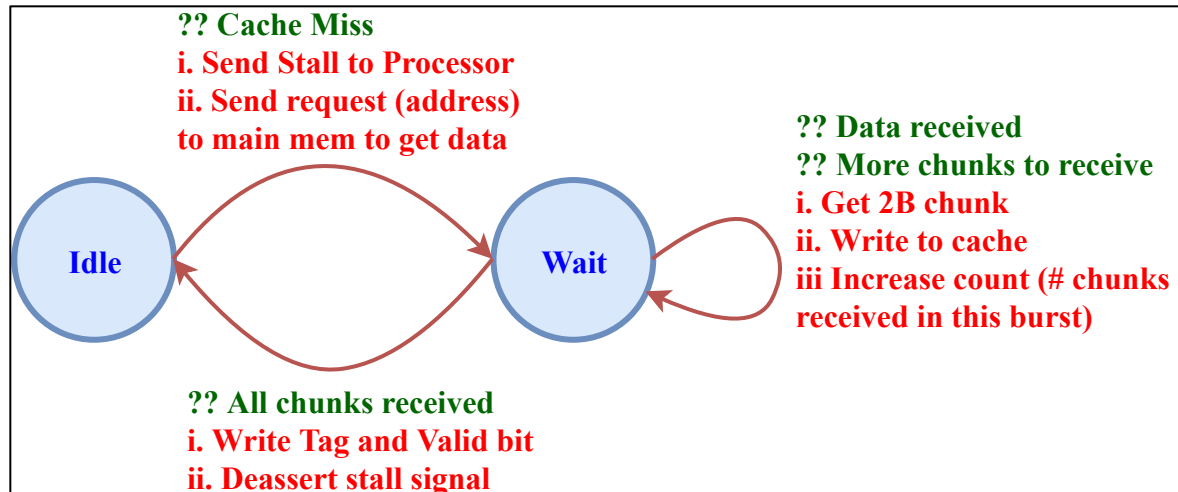
b. If the tag matches (i.e., hit), the read/write from/to the data array is valid. This data is returned via the cache data port in case of a read.

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 if it is a write hit, the memory write will complete in parallel to the cache write.

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 [new information]

The figure below shows a simple FSM for retrieving a block from memory upon a cache miss. The blue refers to the state, the green refers to the condition for changing state, and the red refers to actions performed on state transitions.



This is only a simple FSM for sequentially requesting and receiving each 2-byte chunk from memory. You are required to enhance this FSM to support **pipelined** memory requests. Since memory is pipelined, read requests can be issued to memory every cycle, and each 2-byte chunk should be received in consecutive cycles.

**Points to note:**

- The cache is **write-allocate**. On both read and write misses, you need to bring in the correct block from memory to the cache.
- The cache is **write-through**, so there is no data to be written back to memory upon an eviction.
- The cache controller stalls the processor on a miss, and only after the entire cache block is brought into the cache, the new tag is written into the meta-data array, the valid and LRU bits are set and the stall is deasserted.

### 3.4 Memory Contention on Cache Misses [new information]

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 selects 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 misses from the I-cache at one time nor will you ever have multiple misses from the D-cache at one time; at most you can have one of each type in parallel.

## 4. Interface [unchanged from Phase 2]

Your top level Verilog code should be in a file named *cpu.v*. It should have a simple 4-signal interface: *clk*, *rst\_n*, *hlt* and *pc[15:0]*.

Signal Interface of <i>cpu.v</i>		
Signal:	Direction:	Description:
clk	in	System clock
rst	in	Active high reset. A 1'b1 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 has finished processing the instruction prior to the HLT
pc[15:0]	out	PC value over the course of program execution

**5. Submission Requirements [read carefully, new information]**

1. You were provided with an assembler in Phase 1 to convert your text-level test cases into machine level instructions. You will also be provided with a global testbench and test case. The test case should be run with the testbench and the trace output (as a .txt file) should be submitted for Phase 2 evaluation. In other words, test1.list should produce a trace file called test1.txt.
  - a. Note: The trace module, wisc\_trace\_p3.v, has MIPS-like signal ports. MemRead should only be 1'b1 for LW, and MemWrite should only be 1'b1 for SW. Technically MemRead can be “dontcare” for all but LW, but it pollutes the tracefiles with unused reads.
2. You are also required to submit a zipped file containing: all the Verilog files of your design, all testbenches used and any other support files.
3. You must also submit the project final report; see the guidelines for the report on Canvas.