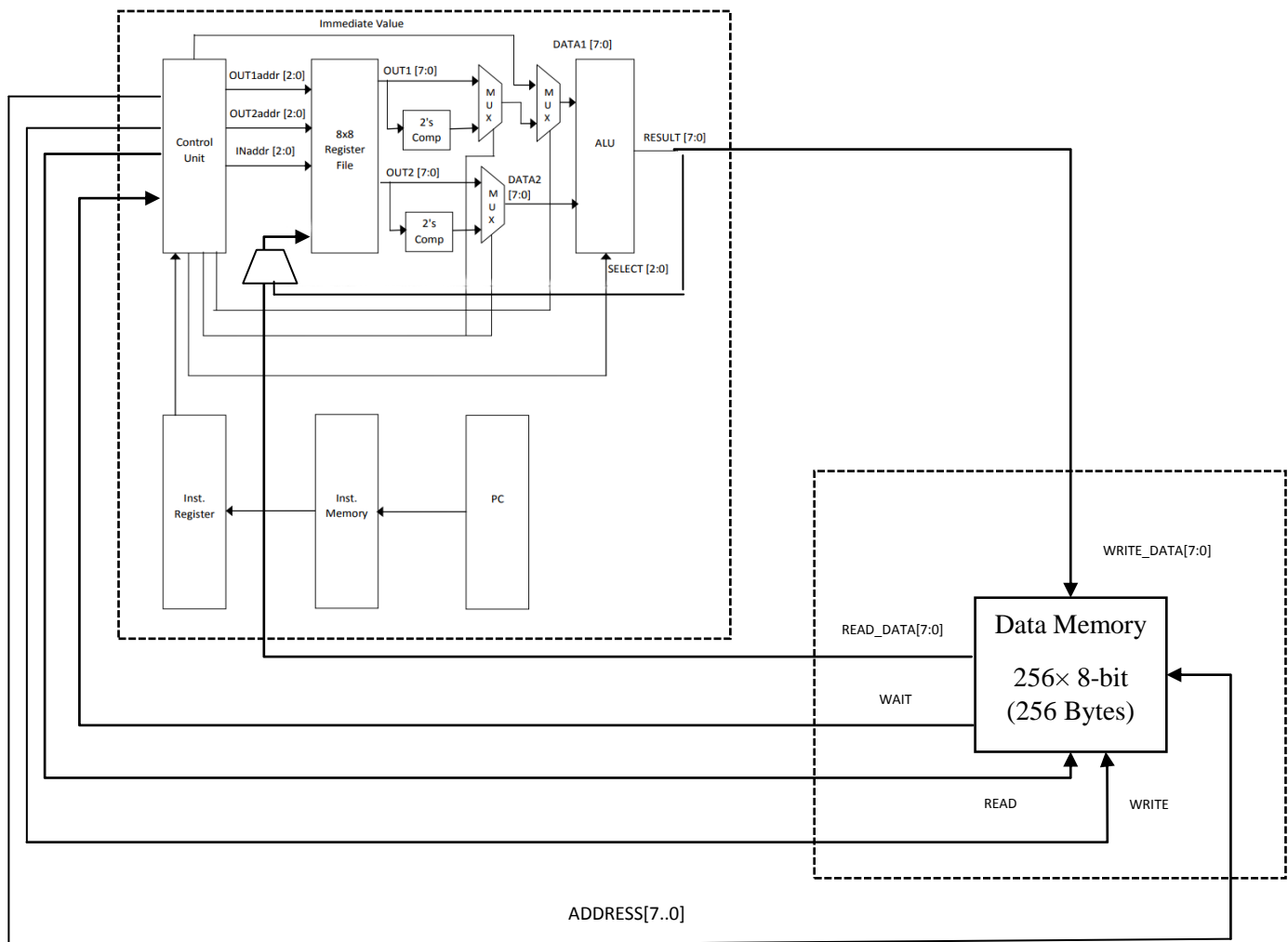


Lab5-Memory

In this lab you will add memory modules to your processor. Some systems store both instructions and data in the same memory device, while other systems use separate memory devices for instructions and memory. For our system, we will use separate memory devices. The lab will be completed in three parts. Parts 1 and 2 must be completed and demonstrated in the lab itself. Part 3 can be submitted online.

Part 1 - Data Memory



The above diagram shows a data memory module being connected to your processor. We will use 256, 8-bit registers to store data in this memory. The data memory uses the following signals to interface with the processor:

- ADDRESS[7..0] - Control Unit provides this to the memory when it wants to access (read or write) that location.
- WRITE_DATA[7..0] - This is a data value the CPU is trying to store in the memory, at the location pointed to by ADDRESS. This value is supplied from the Register File, through the ALU.
- READ_DATA[7..0] - This is a data value the CPU is trying to read from the memory, at the location pointed to by ADDRESS. This value is then stored in the Register File.
- READ - Control Unit asserts this signal when it needs to read a value from the memory. Memory starts a read operation when it sees this signal go from low to high.
- WRITE - Control Unit asserts this signal when it needs to write a value to the memory. Memory starts a write operation when it sees this signal go from low to high.
- WAIT - Memory asserts this signal when it starts a read or write operation, and keeps it asserted while the operation is in progress. Control Unit should stall the processor while this signal is asserted. In other words, all operations including instruction fetching and register-file reading/writing should be suspended until WAIT is de-asserted by the Memory.

Study the supplied sample code for the Data Memory module and see how to connect it to your processor. Please note that a delay of 100 clock cycles is to be artificially added to the memory module in order to simulate it with realistic timing.

You need to implement two new instructions (load and store) in your processor, to make use of the new Data Memory. The new instructions will follow the same bit format as previous instructions.

OP-CODE (bits 31-24)	DESTINATION (bits 23-16)	SOURCE 2 (bits 15-8)	SOURCE 1 (bits 7-0)
-------------------------	-----------------------------	-------------------------	------------------------

In the new instructions, memory addresses should be provided as immediate values in the instruction itself. See examples below:

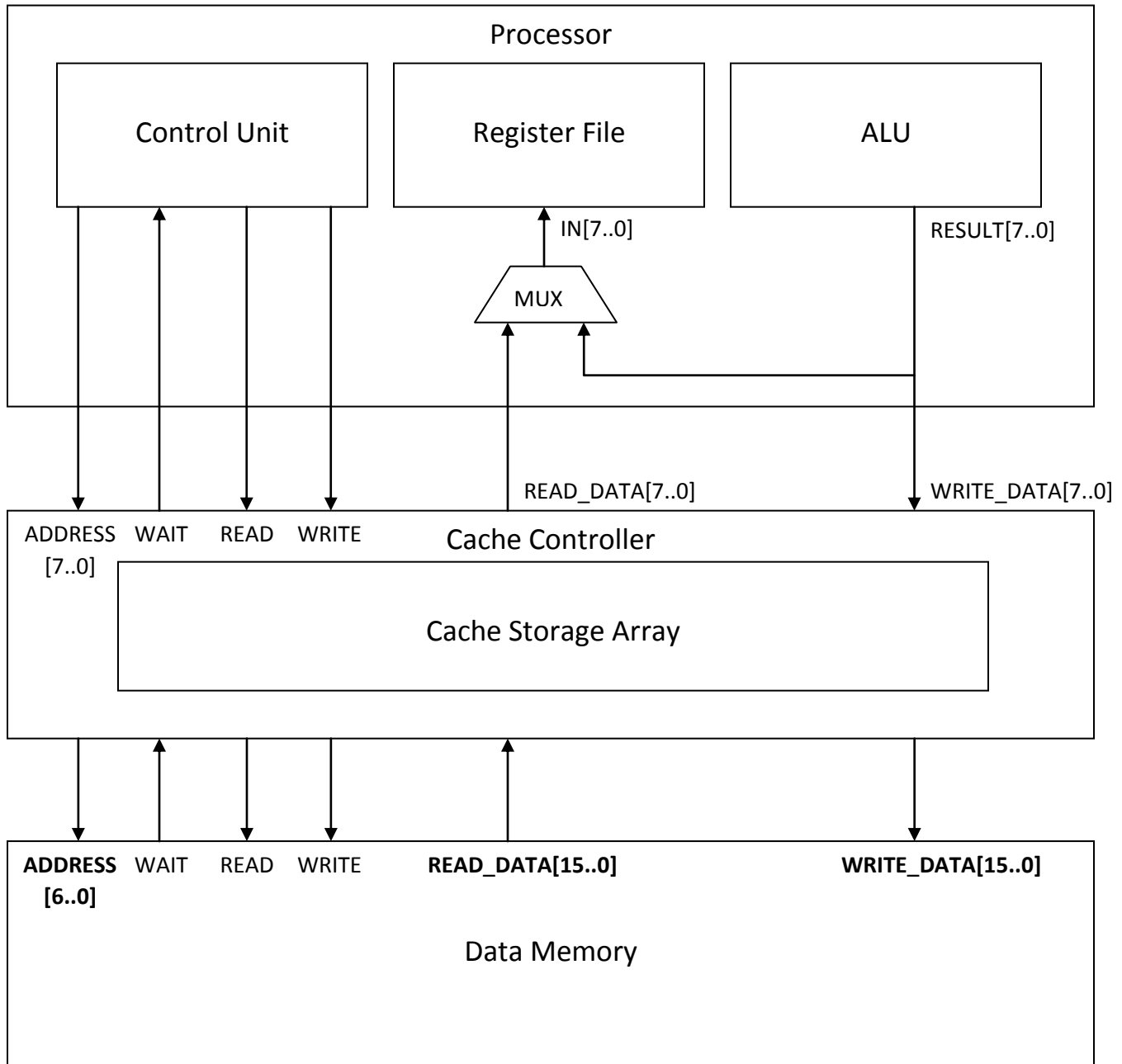
`load 4, X, 0x1F` (read value at memory address 0x1F into register 4. Ignore SOURCE 2)

`store0x8C, X, 2` (write value in register 2 into the memory at address 0x8C. Ignore SOURCE 2)

1. Connect the Data Memory module to your processor as specified. Include the 100 cycle delays.
2. Implement the `load` and `store` instructions, and modify the control unit accordingly. Make sure to stall the processor when the WAIT signal is asserted by the Memory.
3. Test your processor with a hardcoded sequence of instructions (program) including `load` and `store`. Note the number of clock cycles taken to complete the program.
4. Demonstrate this part to an instructor within the lab itself.

Part 2 - Data Cache

Now that your processor can access Data Memory using load and store instructions, your next task is to implement a simple Data Cache. The goal of using a cache is to reduce time spent on accessing memory. The Data Cache will act as an intermediary between processor and Data Memory (see figure below). Your cache module should use the same signals as the memory device when interfacing with the processor.



Following are the parameters to use when designing the cache:

- Cache size should be 16 data bytes.
- Use direct-mapped block placement.
- Use a block size of 2-bytes. (Cache has eight entries, each of which can store a 2-byte data block, hence a total size of 16 bytes. Note that you should also store the address Tag for each cache entry.)
- Each cache entry should be associated with a Valid bit. This should be used to indicate "empty" and "filled" cache entries. At the very beginning, all entries are invalid.
- Each cache entry should be associated with a Dirty bit. This should be used to indicate "updated/written" and "non-updated/non-written" cache entries. When a block receives a write access, its Dirty bit should be updated.
- Use write-back policy for writes. (When a block is evicted from the cache AND its Dirty bit is set, that block should be written to the memory. Note that an eviction can happen in the event of a cache miss. If the evicted block is Dirty, it should be written back to the memory before the new block is fetched into the cache)

Your processor accesses a single data byte at a time. Cache controller should divide the address into Tag, Index and Offset sections. Then figure out which cache block/entry should be accessed based on the Index, and perform Tag comparison to decide if there is a hit. In the case of a hit, controller should determine which data byte within the block the processor is accessing, based on the Offset.

In the case of a cache miss, the controller should first check the Dirty bit of the mismatching 2-byte block and write it back to the memory if Dirty. Then load the missing 2-byte block from the memory into the cache. To do this load, the cache needs to use a 7-bit block-address (this block-address is composed with the Tag and Index portions of the original byte-address, ignoring the Offset). After loading the cache entry with the new data, you must update the Tag, Valid and Dirty bits for that entry.

Note that the interface between the Data Cache and the Data Memory transfers data as 2-byte blocks. Therefore, the cache controller should provide 7-bit block-addresses to the memory. **You must modify the Data Memory** module accordingly to use 7-bit block addresses and 16-bit WRITE_DATA and READ_DATA signals.

1. Implement a cache module as specified, modify the Data Memory accordingly and connect them.
2. Test your processor with the hardcoded sequence of instructions. Compare the number of clock cycles taken with Part 1.
3. Demonstrate this part to an instructor within the lab itself.

Part 3 - Instruction Memory and Cache

Once you have the Data Cache and Data Memory working properly, use the same concepts to add an Instruction Memory module and an Instruction Cache module for your processor.

It should be noted that your instructions are 32-bits wide (let's call this a "word"). Therefore, your Instruction Cache and Instruction Memory should store 32-bit word entries. Instruction Memory should contain 256 such entries, each of which is 32-bits size (Total size of Instruction Memory is $256 \times 32\text{-bits} = 1024\text{ Bytes}$). Its interface should use the signals ADDRESS[7..0], READ_INST[31..0], WAIT and READ. Processor's Control Unit provides the ADDRESS based on the PC (Program Counter) value, and asserts the READ signal when it's ready to fetch a new instruction (note that the address points to a 32-bit word, not a byte). Instruction Memory keeps looking for a READ signal, and asserts the WAIT signal when it starts a new read operation, and keeps it asserted while the operation is in progress. Control Unit should stall the processor while this WAIT signal is asserted. Instruction memory should use an artificial delay of 100 cycles, provide the requested instruction word in READ_INST signal, and de-assert the WAIT signal. Control Unit needs to read the instruction from the READ_INST signal and store it in the IR (Instruction Register).

You may hardcode your program (sequence of instructions) into this memory.

1. Implement the Instruction Memory as specified, test your program and note the number of clock cycles taken.

Following are the parameters to use when designing the Instruction Cache:

- Cache size should be 16 words (64-bytes).
- Use direct-mapped block placement.
- Use a block size of 4-words (16-bytes). (Cache has four entries, each of which can store a 16-byte data block, hence a total size of 64 bytes. Note that you should also store the address Tag for each cache entry.)
- Each cache entry should be associated with a Valid bit. This should be used to indicate "empty" and "used" cache entries. At the very beginning, all entries are invalid.
- No Dirty bit is needed, as we don't write to the instruction memory.
- No write-policy is needed, as we don't write to the instruction memory.

Your processor fetches a single instruction at a time. Cache controller should divide the address into Tag, Index and Offset sections. Then figure out which cache block/entry should be accessed based on the Index, and perform Tag comparison to decide if there is a hit. In the case of a hit, controller should determine which instruction (32-bit word) within the block the processor is accessing, based on the Offset.

In the case of a cache miss, the controller should load the missing 4-word (16-byte) block from the Instruction Memory into the cache. To do this load, the cache needs to use a **6-bit block-address** (this block-address is composed with the Tag and Index portions of the original byte-address, ignoring the Offset). After loading the cache entry with the new data, you must update the Tag and Valid bits for that entry.

Note that the interface between the Instruction Cache and the Instruction Memory transfers data as 4-word (16-byte) blocks. Therefore, the cache controller should provide 6-bit block-addresses to the memory. **You must modify your Instruction Memory** module accordingly to use 6-bit block addresses and 128-bit READ_INST signals.

1. Implement an Instruction Cache module as specified, modify the Instruction Memory accordingly and connect them.
2. Test your processor with the hardcoded sequence of instructions. Compare the number of clock cycles taken with Part 3-1 above.
3. Submit your entire system in the course web-page. You will get additional time for this submission.