2020 Computer Architecture Project 2

	B07902053 許浩鳴	**B07902133 彭道耘**	**B07902141 林庭風**
workload	report, cache controller state diagram	report, sram explanation	report, cache implementation, debug

Development Environment

- OS: Ubuntu 18.04.
- Compiler: iverilog.
- IDE: Vim.
- Method to Debug: `gtkwave` and `$display`.

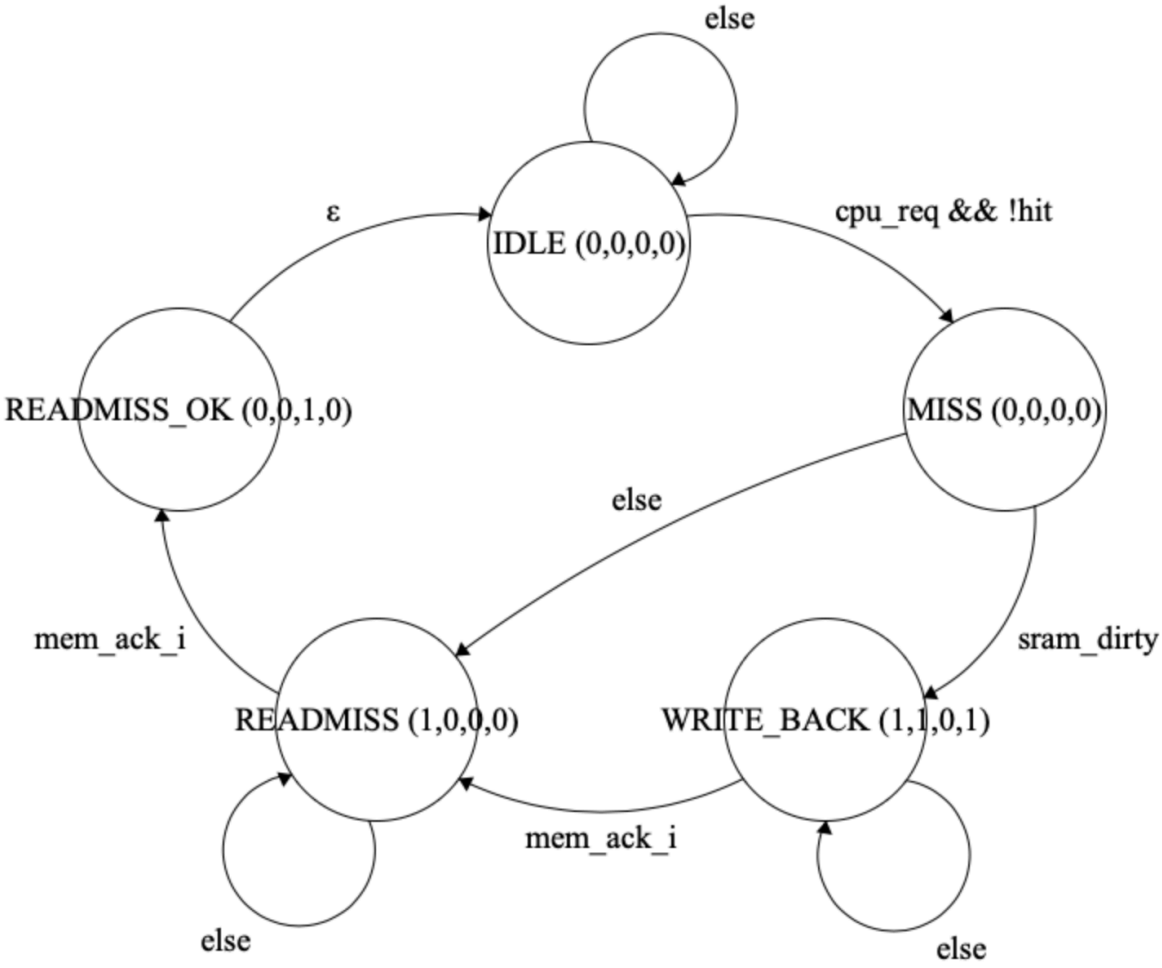
Cache

Cache Controller

There are five states of cache: *IDLE*, *MISS*, *WRITE BACK*, *READMISS*, *READMISS_OK*, and four signals *(mem_enable, mem_write, cache_write, write_back)*.

- *IDLE*: The normal state of cache, nothing happens when in this state. If the miss on cache happens, the state is transferred to *MISS*.
- *MISS*: When the cache miss just happens, we need to check whether the dirty bit is true. If it is true, we move to *WRITE BACK* state, otherwise *READMISS* state.
- *WRITE BACK*: This state means we need to write the data in the cache into Data Memory. We need to wait until *mem_ack_i* is true, and move the state to *READMISS*. The signal *mem_ack_i* may be true only in every 10 cycles.
- *READMISS*: In this state, we just wait until *mem_ack_i* becomes true, and move to *READMISS_OK*.
- *READMISS_OK*: In this state, if the signals *enable_i* is true, meaning that the data is ready and can be written to cache.

The following automata demonstrate how the states transfer, and the tuple in the states represent the signal *(mem_enable, mem_write, cache_write, write_back)*.



- Output: `cpu_data_o`, `cpu_stall_o`
  - `cpu_data_o`: data in cache being passed to CPU
  - `cpu_stall_o`: signal emitted to PC and each pipline register when cache miss, causing CPU to stall

SRAM

- input: clock signal(*clk_i*), *rst_i*, *addr_i*, *tag_i*, *data_i*, *enable_i*, *write_i*
  - *write_i*: if needed to write to the momoey.
  - *addr_i*, *tag_i*: query address and tag.
- output: *tag_o*, *data_o*, *hit_o*

We implement this module to get tag and data when memory read/write. The cache is two-way associative, with replacement policy being LRU (least recently used). When access cache with tag and data, we will check whether the tag are the same and valid bit is 1 in two sets respectively. If any set has memory hit, we can write the tag and data in the cache if the *enable_i* and *write_i* are 1.

We use the register *pos* to implement LRU (least recently used). When we get cache miss in both sets, we choose the index in *pos*, and write the data in the coressponding block in that cache, then flip the index of *pos*.

Other Difficulties Encountered

The difficulty encountered is that we found that the data in Data Memory never changed. We find the bugs for about one hour and the only bug is that *w_hit_data* should be assigned to *cpu_data_i*, but we assign it to *cpu_data*. After removing this bug, everything works and the sample instructions are passed.