Snow64 LAR File Details (What happens during a write into the LAR file)

- Data only:
  - metadata:
    - `tag`: constant
    - `data_type`: constant
    - `int_type_size`: constant
    - `data_offset`: constant
  - shareddata:
    - `base_addr`: constant
    - `ref_count`: constant
    - `dirty`: set to 1
    - `data`: set to `real_in_wr.non_ldst_data`
- Data and Type:
  - metadata:
    - `tag`: constant
    - `data_type`: set to `real_in_wr.data_type`
    - `int_type_size`: set to `real_in_wr.int_type_size`
    - `data_offset`: constant
  - shareddata:
    - `base_addr`: constant
    - `ref_count`: constant
    - `dirty`: set to 1
    - `data`: set to `real_in_wr.non_ldst_data`
- Loads and Stores:
  - Set our `data_type` and our `int_type_size` to that provided by the instruction. Also set our `data_offset` to `__in_wr__incoming_base_addr.data_offset`.
  - If **any** LAR has data from the requested address:
    - If **we aren't** one of those LARs:
      - Set our tag to the already allocated one.
      - Increment the reference count of the already allocated element of shared data.
      - If we were doing a `store` instruction: copy our old element of shared data's `data` field to our new element of shared data.
      - Depending on our **current** element of shared data's `ref_count`:
        - `current ref_count == 0`:
          - We do not much of interest here, and we don't have to touch memory at all.
        - `current ref_count == 1`:
          - We were the **only** reference to our old data.

                - In this situation, we need to deallocate our old tag, pushing it onto the stack of tags.
                    - Note that this is the **only** situation in which we need to actually deallocate a tag.
                - In addition to deallocating our tag, we need to set our old element of shared data to have zero references.
                - If our old data was dirty, we have to write it to memory, as all references to our data have disappeared.
                - If we are doing a `store` instruction, we don't need to actually do anything else, but if we are doing a `load` instruction, we need to write our old data out to memory.
            - `current ref_count > 1`:
                - There are other LARs besides us who have a reference to our element of shared data.
                - In this situation, no allocation or deallocation has to be performed, and we do not have to touch memory at all.
                - We need to decrement our element of shared data's `ref_count`, though.
        - If **we are** one of those LARs:
            - We do nothing really useful here in this case. This is the cheapest form of a "hit".
    - If **no** LAR has the data from the requested address
        - Depending on our **current** element of shared data's
            - `current ref_count == 0`:
                - This is from before we were allocated.
                - In this case, we do not have to perform a write to memory, as we don't have any data of our own yet.
                - Thus, we need to allocate a fresh element of shared data, and set its `base_addr` field to that of the requested address. We also need to set its `ref_count` field to 1.
                - If we are doing an `load`:
                    - No LAR currently has the data from our desired address, so we need to do a read from memory.
                    - We also need to clear our dirty flag as this is a load of fresh data from memory.
                - If we are doing an `store`:
                    - This is a weird case, as we are doing an `store` of a LAR that hasn't been allocated yet (i.e. its data is zero)!
                        - Note that this case is not likely to happen much in practice, but it does still need well-defined behavior.
                    - We also need to mark this LAR as dirty.

- `current ref_count == 1`:
    - In this case, we are the only reference to our old data, and we will next be the only reference to our new data. Also, we don't need to do any allocation or deallocation of shared data, as we can just re-use our old element of shared data.
    - Additionally, we need to set our existing element of shared data's `base_addr` field to the new base address.
    - If we were dirty:
        - Write our old data needs to go out to memory at our old address.
        - If we're performing a `load`:
            - Load from memory the data from our new address.
            - Mark our data as clean, as this is a load of fresh data.
        - If we're performing a `store`:
            - We do not have to load data from memory.
            - We were already marked as dirty, which is what a `store` would have done anyway.
    - If we were not dirty:
        - We do not have to write our old data back out to memory because we were previously marked as clean.
        - If we're performing a `load`:
            - Load from memory the data from our new address.
            - Since our element of shared data is already marked as clean, we do not need to mark ourself as clean.
        - If we're performing a `store`:
            - We do not need to touch memory at all during this case, and in fact, all we have to do is mark our element of shared data as dirty.
- `current ref_count > 1`:
    - There are other LARs that have our old data, but no LAR has the data from our new address.
    - In this case, we do not have to perform a memory write.
    - As we are no longer going to be one of the references to our old data, we need to decrement our old element of shared data's `ref_count` field.
    - Also, we need to allocate a new element of shared data for us to use.
    - Set the new element's `base_addr` field to the new base address, and set the new element's `ref_count` field to 1. We should be the only reference to this element of shared data.

- If we're performing a `load`:
  - Load from memory the data from our new address.
  - Mark our data as clean, as this is a load of fresh data.
- If we're performing a `store`:
  - We do not need to touch memory at all in this case.
  - We do, however, need to make a copy of our old element of shared data's `data` field into our new element of shared data's `data` field, and we need to mark our new element of shared data as dirty.