RISC-V M (Basic Add-Shift Multiplier) (3 points):

How we will implement it:

We will implement a multiplier that supports up to a 32-bit multiplier, 32-bit multiplicand, and a 64-bit output. The multiplier will use the add-shift method to compute the product. For division and remainder support, the unit will use the subtract-shift method. As of right now, we are expecting our multiplication unit to contain full-adders, half-adders, muxes, and basic combinational logic. Furthermore, we expect that we may have to implement 64-bit registers for the support of a 64-bit product.

Where it will be placed in the datapath:

The multiplication unit will be in the execution stage of the pipeline. This is due to the fact that logical computations are done in this stage.

Potential pipeline modifications required:

Since the multiplication unit is another variation of execution units, we will have to account for the same implementations as we did when creating the ALU and comparator. This includes adding input muxes, and another data value that will be stored in our control rom, such as alu_out and cmp_out, and passed throughout the pipeline registers. We will also have to update our hazard detection unit and forwarding unit to account for stalling and data dependency as the multiplier will take multiple clock cycles.

L2 Cache + 4 Way-Set Associativity (4 points):

How we will implement it:

We will implement a lower cache level (L2 cache) that interacts directly with L1 cache and arbiter. L1 cache would directly hook up with L2 cache. On a L1 cache miss, we would check L2 cache to see if it has the required data. If not, L2 cache would go to physical memory to fetch the data and pass it back up to L1 cache. Both caches will contain the same data, however, L2 cache will support a larger size so it can hold more data. For 4-way set associativity, we would upgrade our cache ways to 4 ways by increasing the size of our LRU array, and by instantiating more ways.

Where it will be placed in the datapath:

Our new cache would be placed in between the arbiter and our current L1 cache in our datapath.

Potential pipeline modifications required:

The new cache would require muxes, combinational logic elements, data arrays, and an LRU array just to name a few required elements. No pipeline modifications will be required. Any cache/memory modification should not interfere with the pipeline as the pipeline should be unaware of these changes.

Basic Hardware Prefetching (4 points):

How we will implement it:

We will have a monitor that actively checks the address line and instruction cache of the arbiter when there is a cache miss. When the required cache line is fetched, the monitor will initiate another read for the next cache line which will get stored into cache.

Where it will be placed in the datapath:

In the memory stage. More precisely, in between the L2 cache and arbiter as it will need to check the address and response signals between these two units.

Potential pipeline modifications required:

No pipeline modifications will be required. Any cache/memory modification should not interfere with the pipeline as the pipeline should be unaware of these changes.

Eviction Write Buffer (may upgrade to victim buffer if time allows) (4 points):

How we will implement it:

The eviction write buffer will contain evicted data (1 cache line) from L1 cache. This will allow L2 cache to respond with the updated data for the cache line which will allow the pipeline to begin progressing earlier as we are not doing a read-after-write.

Where it will be placed in the datapath:

The eviction buffer will be placed in between L1 cache and L2 cache. The caches should not be aware of the existence of this buffer.

Potential pipeline modifications required:

No pipeline modifications will be required. Any cache/memory modification should not interfere with the pipeline as the pipeline should be unaware of these changes.

Parametrized Cache (# of Ways and # of Sets (3 points):

How we will implement it:

We will rewrite the cache module by defining parameters, such as number of ways and number of sets. We will then generate the correct number of data arrays, which represent ways, dirty arrays, and tag arrays according to the parameter. We will also have to calculate the LRU bits according to the new size of the cache. This will involve using the pseudo-LRU algorithm and tweaking the algorithm depending on the number of LRU bits that we have.

Where it will be placed in the datapath:

Cache will remain where it is at in the datapath currently. This is in the memory stage and is located right before the arbiter and physical memory.

Potential pipeline modifications required:
No pipeline modifications will be required. Any cache/memory modification should not interfere with the pipeline as the pipeline should be unaware of these changes.