Project #3

(Report due is 11:59:59 PM, 10/30/2022)

Project description:

In this project, we will explore various cache design strategies with Gem5 simulator. Specifically, this project will cover the tutorial of building different cache hierarchies in configuration script, modify the cache replacement policy in Gem5 simulator, and study how different cache replacement policies, cache size, and associativity can impact the computer system's performance. Finally, you are required to optimize your cache design for provided benchmarks. Whenever you face the trouble or problem on this project, please refer to Gem5 document here or contact TAs in WeChat. All the following information have been verified on Ubuntu 20.04.1 with Linux Kernel 5.15.0.

Submission:

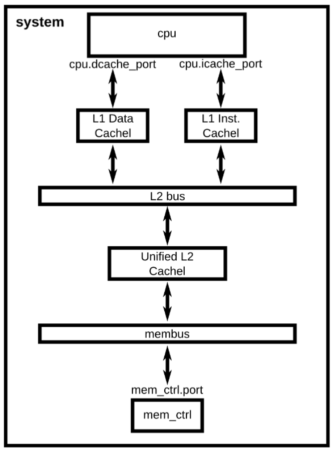
- Please submit (in a report form, both PDF and Word are acceptable) the deliverables for each part in order and clearly defined. Please give a brief description of the results in your report and DO NOT SUBMIT ANY SCRIPTS.
- Please send your report as an attachment at ca2022fall@163.com. The titles of your email AND attachment should both be Student ID_Name_Proj3 (e.g., 123456789_Nilou_Proj3).
- DO NOT PLAGIARIZE. After this project, we will select 10 students randomly and ask them to run their codes in person and answer our questions related to their codes.

Late policy:

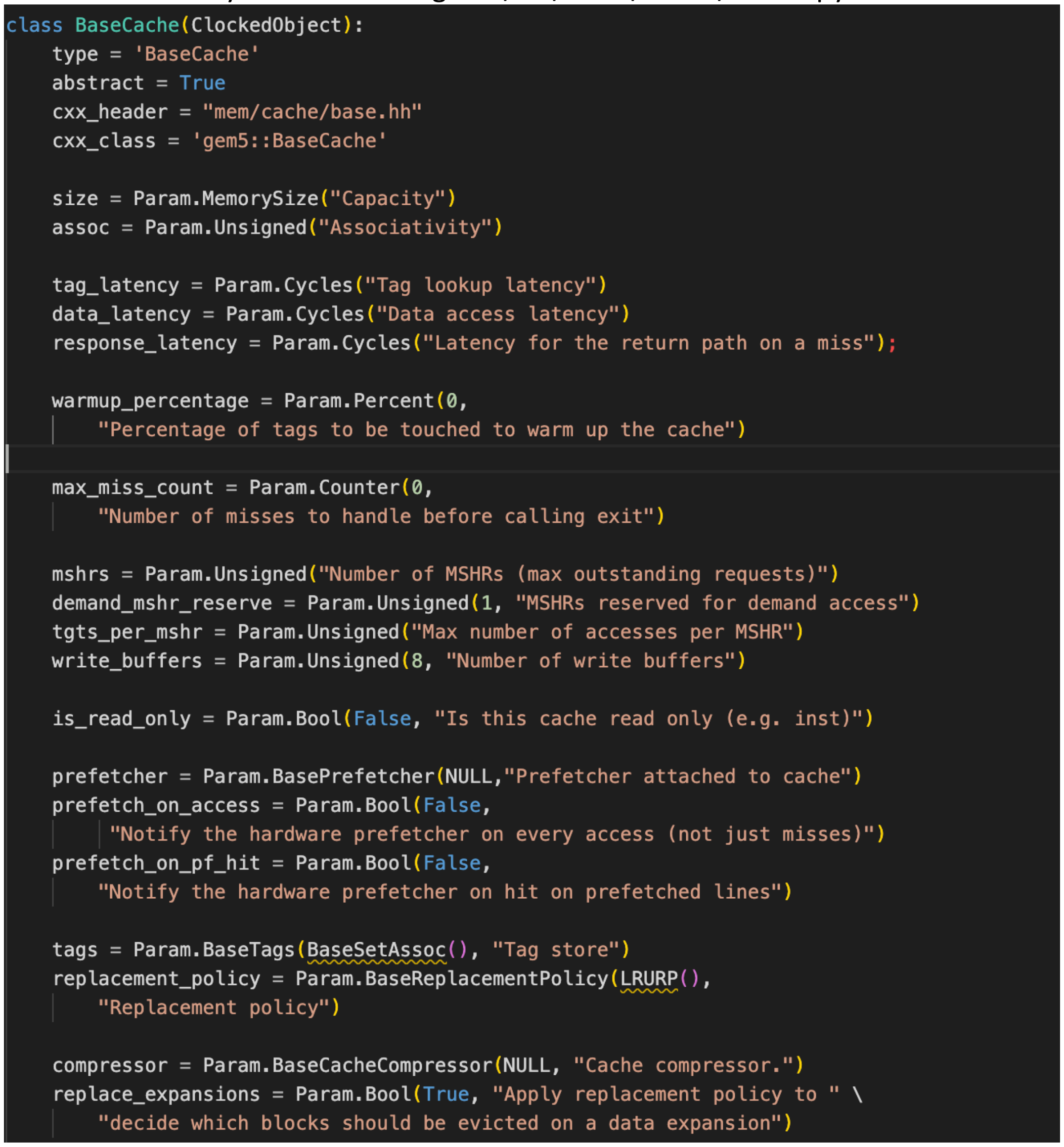
- You will be given 3 slip days (shared by all projects), which can be used to extend project deadlines, e.g., 1 project extended by 3 days or 2 projects each extended by 1 days.
- Projects are due at 23:59:59, no exceptions; 20% off per day late, 1 second late = 1 hour late = 1 day late.

Part 1: Add cache in the configuration script

In this section, we will extend the configuration script "simple.py" in project #1 by adding cache hierarchy. After finishing this section, you are able to make the following system architecture:



In project #1, "simple.py" script directly instantiates CPU and memory class defined in Gem5 and configures them with user's values. Unfortunately, L1I, L1D or L2 cache classes are not provided in Gem5 source code. So it is required to firstly define the corresponding cache class. For your information, Gem5 provides a Python abstract class, *BaseCache*, which describes basic cache framework. The Python class is in gem5/src/mem/cache/Cache.py and shown as follows:



```
class BaseCache(ClockedObject):
    type = 'BaseCache'
    abstract = True
    cxx_header = "mem/cache/base.hh"
    cxx_class = 'gem5::BaseCache'

    size = Param.MemorySize("Capacity")
    assoc = Param.Unsigned("Associativity")

    tag_latency = Param.Cycles("Tag lookup latency")
    data_latency = Param.Cycles("Data access latency")
    response_latency = Param.Cycles("Latency for the return path on a miss");

    warmup_percentage = Param.Percent(0,
        "Percentage of tags to be touched to warm up the cache")

    max_miss_count = Param.Counter(0,
        "Number of misses to handle before calling exit")

    mshrs = Param.Unsigned("Number of MSHRs (max outstanding requests)")
    demand_mshr_reserve = Param.Unsigned(1, "MSHRs reserved for demand access")
    tgts_per_mshr = Param.Unsigned("Max number of accesses per MSHR")
    write_buffers = Param.Unsigned(8, "Number of write buffers")

    is_read_only = Param.Bool(False, "Is this cache read only (e.g. inst)")

    prefetcher = Param.BasePrefetcher(NULL,"Prefetcher attached to cache")
    prefetch_on_access = Param.Bool(False,
         "Notify the hardware prefetcher on every access (not just misses)")
    prefetch_on_pf_hit = Param.Bool(False,
        "Notify the hardware prefetcher on hit on prefetched lines")

    tags = Param.BaseTags(BaseSetAssoc(), "Tag store")
    replacement_policy = Param.BaseReplacementPolicy(LRURP(),
        "Replacement policy")

    compressor = Param.BaseCacheCompressor(NULL, "Cache compressor.")
    replace_expansions = Param.Bool(True, "Apply replacement policy to " \
        "decide which blocks should be evicted on a data expansion")
```

This Python class provides multiple parameters. Among them the most important are: "size" (the cache capacity), "assoc" (the associativity of the cache), "mshr" (the number of entries in miss status handling register), "replacement_policy" (the cache replacement policy), and "tgts_per_mshr" (the number of entries in miss status handling register).

Now, we need to define L1I cache class, L1D cache class, and unified L2 cache class. Before that, let's create a new folder, declare cache class in a new file (new_cache.py), and create a new configuration script by copying the contents of "simple.py".

```
mkdir configs/proj3

cp configs/proj1/simple.py configs/proj3/two-level.py

vi configs/proj3/new_cache.py
```

Then, we should work on new_cache.py. Since new cache class should inherit "Cache" class (Cache is the subclass of BaseCache) defined by Gem5, the first step is to import Gem5's objects in new_cache.py:

```
from m5.objects import *
```

Next we can declare "L1ICache", "L1DCache", "L2Cache" class by inheriting "Cache":

```
class L1DCache(Cache):
    size = '32kB'
    assoc = 2
    tag_latency = 2
    data_latency = 2
    response_latency = 2
    mshrs = 4
    tgts_per_mshr = 20

    def __init__(self):
        super(L1DCache, self).__init__()
```

```
class L1ICache(Cache):
    size = '32kB'
    assoc = 2
    tag_latency = 2
    data_latency = 2
    response_latency = 2
    mshrs = 4
    tgts_per_mshr = 20

    def __init__(self):
        super(L1ICache, self).__init__()
```

```
class L2Cache(Cache):
    size = '256kB'

    assoc = 8

    tag_latency = 20

    data_latency = 20

    response_latency = 20

    mshrs = 20

    tgts_per_mshr = 12


    def __init__(self):
        super(L2Cache, self).__init__()
```

So far, we have specified all the necessary parameters for each cache class. Then, we need to work around the configuration script “two-level.py” we created before. To instantiate L1D cache, L1I cache, and L2 cache, we firstly need to import the object names from “new_cache.py”. This can be done by adding the following to the head of the file:

```
 from new_cache import *
```

Once system and cpu have been instantiated (described in project #1), the L1 caches and L2 cache can be created as follows:

```
root.system.cpu.icache = L1ICache()

root.system.cpu.dcache = L1DCache()

root.system.l2cache = L2Cache()
```

Recall that we designed to connect cache ports directly to the memory bus in project #1. As we need to add L1caches, cpu cache ports should be redirected to L1caches. So you should remove the previous codes and add the following codes:

~~root.system.cpu.icache_port = root.system.membus.cpu_side_ports~~

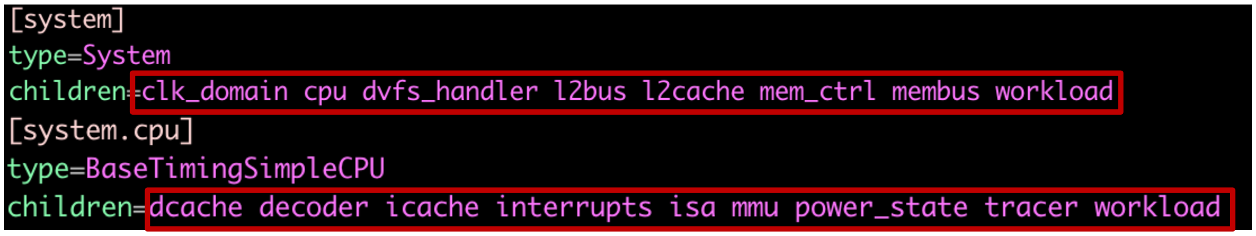
~~root.system.cpu.dcache_port = root.system.membus.cpu_side_ports~~

```
root.system.cpu.icache.cpu_side = root.system.cpu.icache_port

root.system.cpu.dcache.cpu_side = root.system.cpu.dcache_port
```

However, we cannot directly connect both L1I and L1D caches to L2 cache because the L2 cache only opens a single port for connection. The common solution for such conflict is to create a bus which connect L1I cache, L1D cache and L2 cache. Then our next step is to create a bus with the name of “l2bus”. Gem5 provides various bus class in “src/mem/XBar.py”. In this script, you should choose to instantiate “L2XBar”. After that, you need to finish cache port connection:

```
root.system.l2bus = L2XBar()

root.system.cpu.icache.mem_side = root.system.l2bus.cpu_side_ports

root.system.cpu.dcache.mem_side = root.system.l2bus.cpu_side_ports
root.system.l2cache.cpu_side = root.system.l2bus.mem_side_ports

root.system.l2cache.mem_side = root.system.membus.cpu_side_ports
```

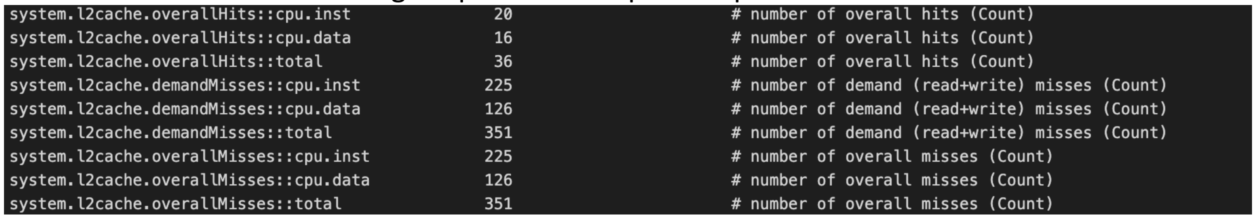
Now, you have completed the configuration of the two-level cache hierarchy. If you run the executable of "hello", you can find out icache, dcache, l2cache and l2bus have been added in config.ini.



```
[system]
type=System
children=clk_domain cpu dvfs_handler l2bus l2cache mem_ctrl membus workload
[system.cpu]
type=BaseTimingSimpleCPU
children=dcache decoder icache interrupts isa mmu power_state tracer workload
```

stats.txt:

The statistics of new components such as icache, dcache, and l2cache are automatically added in file "stats.txt". The following snapshot shows part output of l2cache:



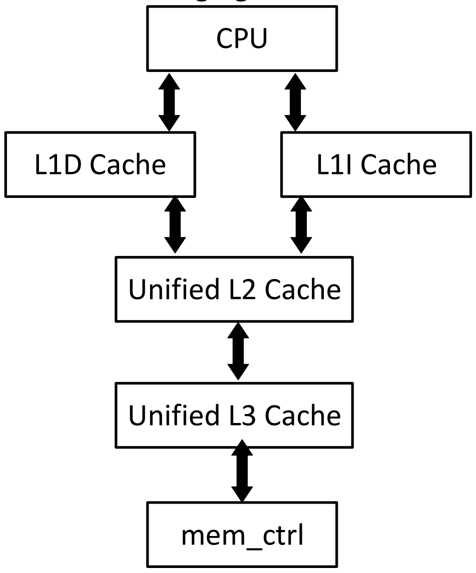
```
system.l2cache.overallHits::cpu.inst            20                   # number of overall hits (Count)
system.l2cache.overallHits::cpu.data            16                   # number of overall hits (Count)
system.l2cache.overallHits::total               36                   # number of overall hits (Count)
system.l2cache.demandMisses::cpu.inst          225                   # number of demand (read+write) misses (Count)
system.l2cache.demandMisses::cpu.data          126                   # number of demand (read+write) misses (Count)
system.l2cache.demandMisses::total             351                   # number of demand (read+write) misses (Count)
system.l2cache.overallMisses::cpu.inst         225                   # number of overall misses (Count)
system.l2cache.overallMisses::cpu.data         126                   # number of overall misses (Count)
system.l2cache.overallMisses::total            351                   # number of overall misses (Count)
```

In "stats.txt", you can find out the total number of hits, misses, and accesses for each cache (L1I, L1D, and L2 caches). You can easily calculate cache hit ratio by dividing cache hits ("overallHits::total") by total cache accesses ("overallHits::total" + "overallMisses::total").

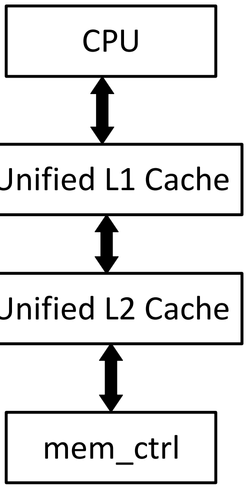
Part 1 deliverables:

Based on the study of this section, you are required to design different cache hierarchies.

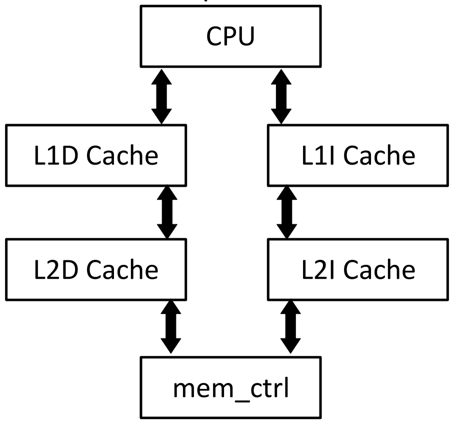
1. Please design three-level cache as the following figure:



2. Please design a new two-level cache which has a unified L1 cache:



3. Please revise two-level cache which has separate L2I and L2D caches:



Once you complete your design, please turn in a snapshot of "config.ini" file to prove your correct design and a brief explanation on how you make the implementation.

Part 2: Implement Cache Replacement Policy

Cache Replacement Policy Implementation Example

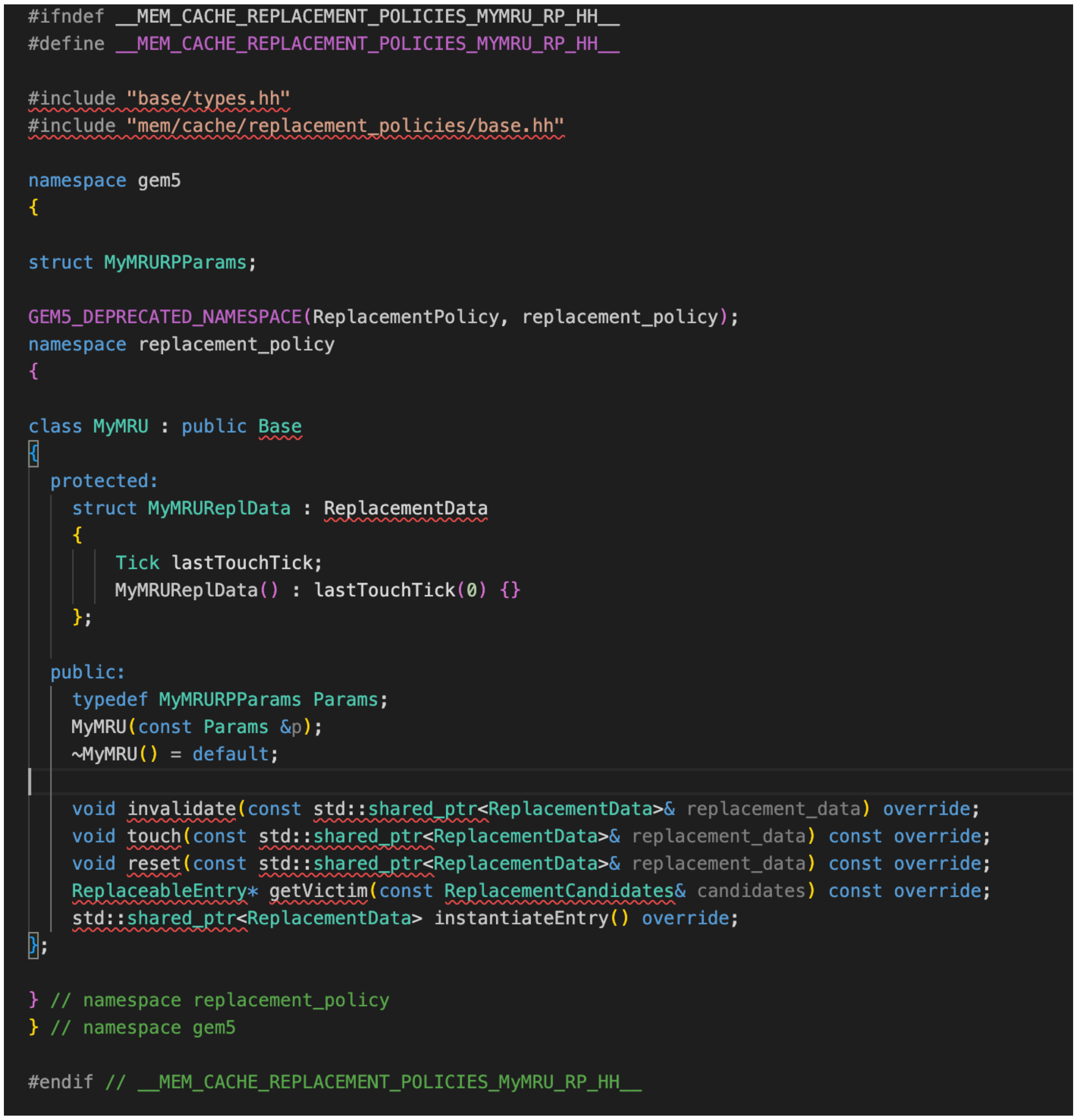
The cache hierarchy is designed to speed up memory access by leveraging data's spatial and temporal locality. In other words, an effective cache should keep frequently used data as long as possible, while evicting data which will not be used in future. Thus cache replacement policy severely impacts the cache's performance. This section will explain step by step about how to add your own cache replacement policy in Gem5. After this section, you are able to evaluate your own cache replacement scheme.

Let's get started with an example of most-recent used policy (MyMRU). The basic implementation is creating an MyMRU queue and tracking the most recently used block by moving the latest accessed block to the head of MRU queue. On a replacement, the head of

MRU queue is selected to evict, which is most recently used blocks. To implement MyMRU SimObject, we firstly add a new Python class in the file "src/mem/cache/replacement_policies/ReplacementPolicies.py", which is associated to myMRU SimObject. The Python class inherits all the parameters from class "BaseSetAssoc":

```
class MyMRURP(BaseReplacementPolicy):

    type = 'MyMRURP'

    cxx_class = 'gem5::replacement_policy::MyMRU'

    cxx_header = "mem/cache/replacement_policies/mymru_rp.hh"
```

The next step is to create mymru_rp.hh and mymru_rp.cc which contains the codes of MyMRU replacement policy implementation. Please note that Gem5 provides uniform interfaces to add new cache replacement policy. For your convenience, you can directly copy and paste the function declaration of other replacement policies such as MRU ("mru_rp.hh") to your "mymru_rp.hh". The contents of mymru_rp.hh should be as follows:

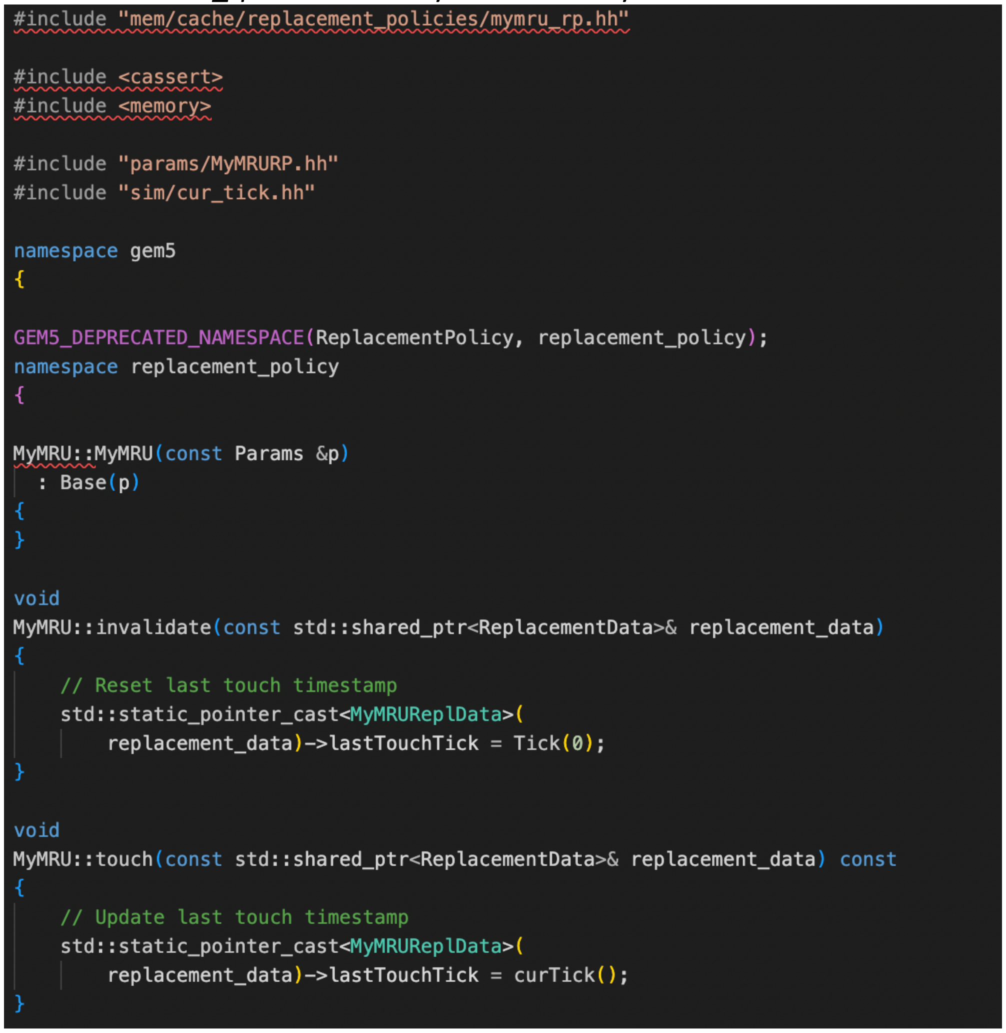


```
#ifndef __MEM_CACHE_REPLACEMENT_POLICIES_MYMRU_RP_HH__
#define __MEM_CACHE_REPLACEMENT_POLICIES_MYMRU_RP_HH__

#include "base/types.hh"
#include "mem/cache/replacement_policies/base.hh"

namespace gem5
{

struct MyMRURPParams;

GEM5_DEPRECATED_NAMESPACE(ReplacementPolicy, replacement_policy);
namespace replacement_policy
{

class MyMRU : public Base
{
  protected:
    struct MyMRUReplData : ReplacementData
    {
        Tick lastTouchTick;
        MyMRUReplData() : lastTouchTick(0) {}
    };

  public:
    typedef MyMRURPParams Params;
    MyMRU(const Params &p);
    ~MyMRU() = default;

    void invalidate(const std::shared_ptr<ReplacementData>& replacement_data) override;
    void touch(const std::shared_ptr<ReplacementData>& replacement_data) const override;
    void reset(const std::shared_ptr<ReplacementData>& replacement_data) const override;
    ReplaceableEntry* getVictim(const ReplacementCandidates& candidates) const override;
    std::shared_ptr<ReplacementData> instantiateEntry() override;
};

} // namespace replacement_policy
} // namespace gem5

#endif // __MEM_CACHE_REPLACEMENT_POLICIES_MyMRU_RP_HH__
```

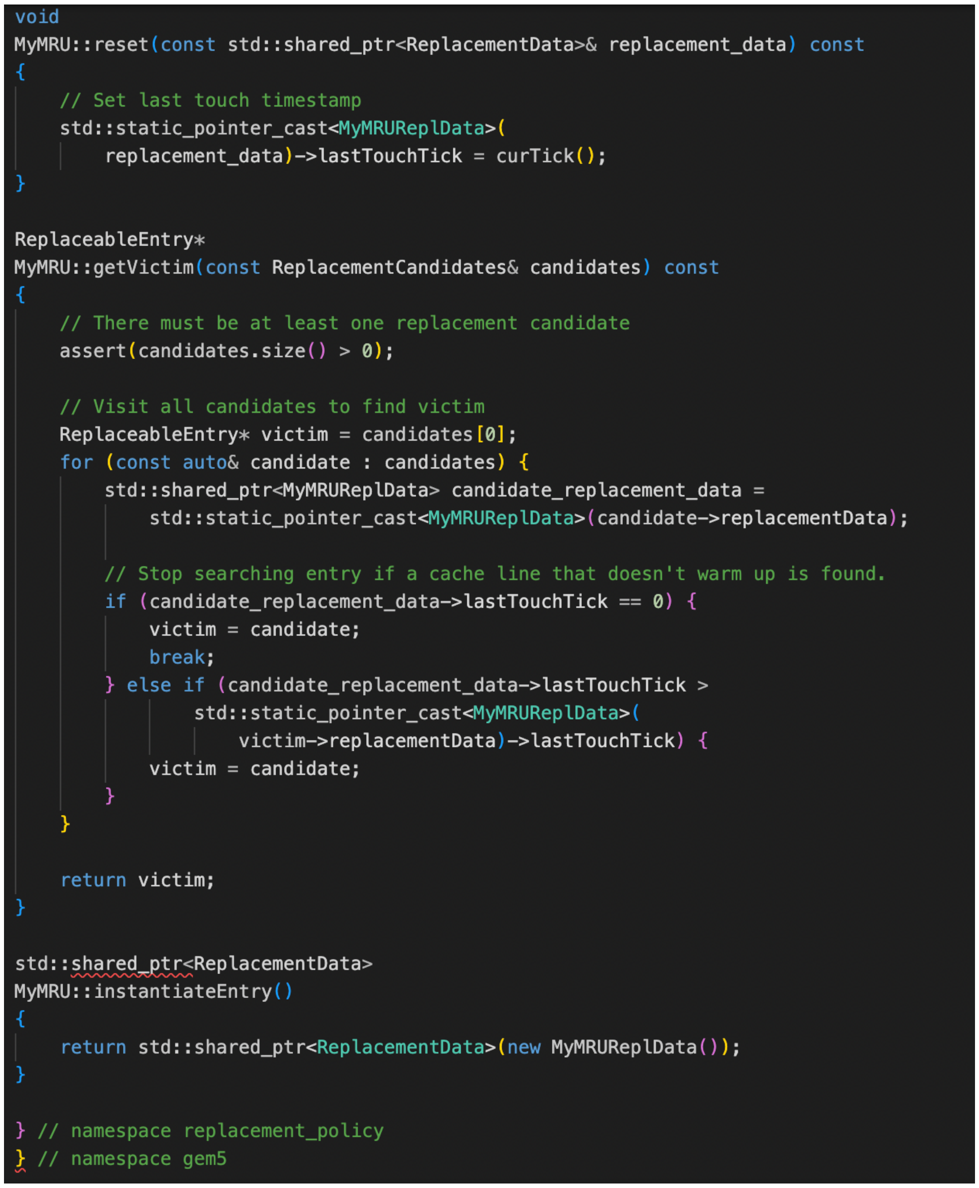
A replacement policy consists of a reset(), touch(), invalidate() and getVictim() methods. Each of which handles the replacement data differently.

- reset() is used to initialize a replacement data (i.e., validate). It should be called only on entry insertion, and must not be called again until invalidation. The first touch to an entry must always be a reset().
- touch() is used on accesses to the replacement data, and as such should be called on entry accesses. It updates the replacement data.
- invalidate() is called whenever an entry is invalidated, possibly due to coherence handling. It makes the entry as likely to be evicted as possible on the next victim search. An entry does not need to be invalidated before a reset() is done. When the simulation starts all entries are invalid.
- getVictim() is called when there is a miss, and an eviction must be done. It searches among all replacement candidates for an entry with the worst replacement data, generally prioritizing the eviction of invalid entries.

Then you need to implement each function in mru.cc. For your convenience, you can copy and paste the codes from *mru_rp.cc* and modify MRU with MyMRU.

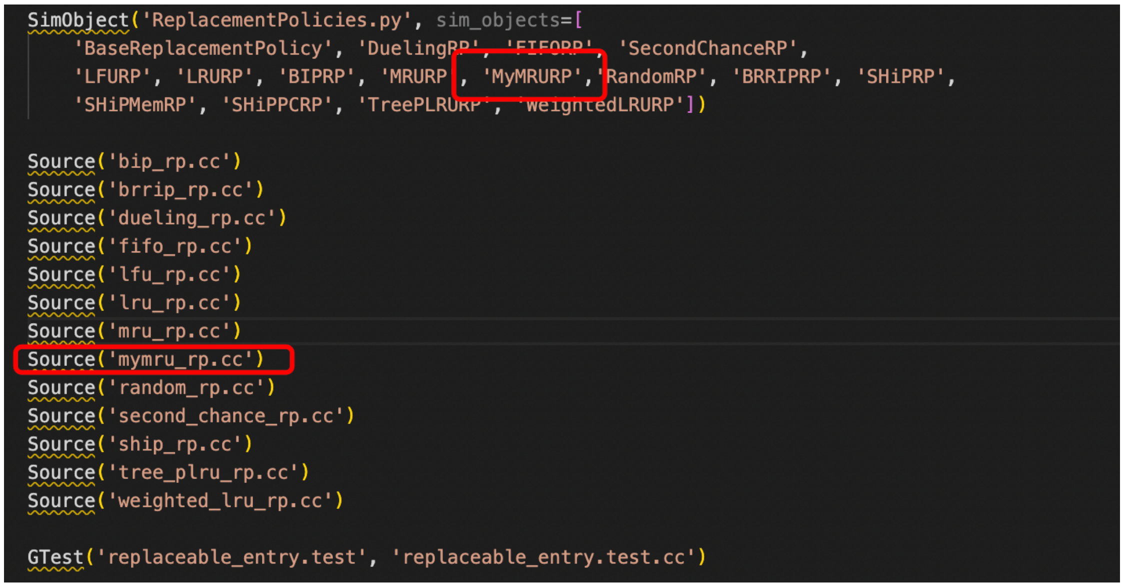


```
#include "mem/cache/replacement_policies/mymru_rp.hh"

#include <cassert>
#include <memory>

#include "params/MyMRURP.hh"
#include "sim/cur_tick.hh"

namespace gem5
{

GEM5_DEPRECATED_NAMESPACE(ReplacementPolicy, replacement_policy);
namespace replacement_policy
{

MyMRU::MyMRU(const Params &p)
  : Base(p)
{
}

void
MyMRU::invalidate(const std::shared_ptr<ReplacementData>& replacement_data)
{
    // Reset last touch timestamp
    std::static_pointer_cast<MyMRUReplData>(
        replacement_data)->lastTouchTick = Tick(0);
}

void
MyMRU::touch(const std::shared_ptr<ReplacementData>& replacement_data) const
{
    // Update last touch timestamp
    std::static_pointer_cast<MyMRUReplData>(
        replacement_data)->lastTouchTick = curTick();
}
```



```
void
MyMRU::reset(const std::shared_ptr<ReplacementData>& replacement_data) const
{
    // Set last touch timestamp
    std::static_pointer_cast<MyMRUReplData>(
        replacement_data)->lastTouchTick = curTick();
}

ReplaceableEntry*
MyMRU::getVictim(const ReplacementCandidates& candidates) const
{
    // There must be at least one replacement candidate
    assert(candidates.size() > 0);

    // Visit all candidates to find victim
    ReplaceableEntry* victim = candidates[0];
    for (const auto& candidate : candidates) {
        std::shared_ptr<MyMRUReplData> candidate_replacement_data =
            std::static_pointer_cast<MyMRUReplData>(candidate->replacementData);

        // Stop searching entry if a cache line that doesn't warm up is found.
        if (candidate_replacement_data->lastTouchTick == 0) {
            victim = candidate;
            break;
        } else if (candidate_replacement_data->lastTouchTick >
                std::static_pointer_cast<MyMRUReplData>(
                    victim->replacementData)->lastTouchTick) {
            victim = candidate;
        }
    }

    return victim;
}

std::shared_ptr<ReplacementData>
MyMRU::instantiateEntry()
{
    return std::shared_ptr<ReplacementData>(new MyMRUReplData());
}

} // namespace replacement_policy
} // namespace gem5
```

So far, you have finished the implementation of new cache replacement policy and created new SimObject. Next, you need to register them to SCons for compilation. As described before, Gem5 uses SCons as its build environment. Specifically, you need to add the following contents in src/mem/cache/replacement_policies/SConscript:



```
SimObject('ReplacementPolicies.py', sim_objects=[
    'BaseReplacementPolicy', 'DuelingRP', 'FIFORP', 'SecondChanceRP',
    'LFURP', 'LRURP', 'BIPRP', 'MRURP', 'MyMRURP', 'RandomRP', 'BRRIPRP', 'SHiPRP',
    'SHiPMemRP', 'SHiPPCRP', 'TreePLRURP', 'WeightedLRURP'])

Source('bip_rp.cc')
Source('brrip_rp.cc')
Source('dueling_rp.cc')
Source('fifo_rp.cc')
Source('lfu_rp.cc')
Source('lru_rp.cc')
Source('mru_rp.cc')
Source('mymru_rp.cc')
Source('random_rp.cc')
Source('second_chance_rp.cc')
Source('ship_rp.cc')
Source('tree_plru_rp.cc')
Source('weighted_lru_rp.cc')

GTest('replaceable_entry.test', 'replaceable_entry.test.cc')
```

Next, you need to add this cache replacement policy to L1I, L1D, and L2 cache in the configuration script “two-level.py”:

```
root.system.cpu.icache.replacement_policy = MyMRURP()

root.system.cpu.dcache.replacement_policy = MyMRURP()

root.system.l2cache.replacement_policy = MyMRURP()
```

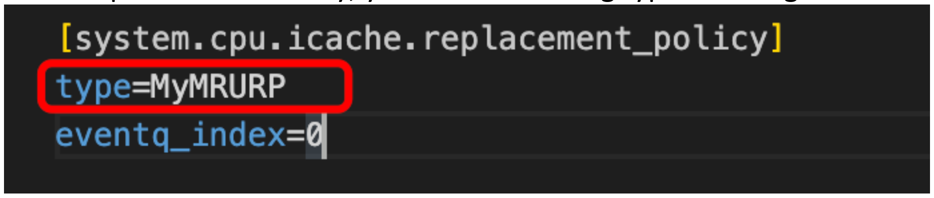
Finally, you need to recompile Gem5 with the following command:

```
scons -j8 build/ARM/gem5.opt
```

And run Gem5 with the following command:

```
build/ARM/gem5.opt configs/proj3/two-level.py
```

If the execution completes successfully, you will see the tag type in config.ini file has changed:



```
[system.cpu.icache.replacement_policy]
type=MyMRURP
eventq_index=0
```

Introduction of Clock Replacement Policy (ClockRP)

The clock algorithm keeps a circular list of pages in memory, with the "hand" (iterator) pointing to the last examined page frame in the list. When a page fault occurs and no empty frames exist, then the R (referenced) bit is inspected at the hand's location. If R is 0, the new page is put in place of the page the "hand" points to, and the hand is advanced one position. Otherwise, the R bit is cleared, then the clock hand is incremented and the process is repeated until a page is replaced. Note that we need to set the R bit of the new inserted page to 1 and move “hand” to next position. A detailed example can be found here: *https://blog.csdn.net/Gu_fCSDN/article/details/103979067*

Part 2 deliverables:

1. You are required to implement ClockRP in Gem5. To verify your correct implementation, you need to compare the performance of your implementation with other replacement policies implemented in Gem5 such as random, LRU, and MRU. We provide four different workloads for this experiment in project #1 package. Please do Gem5 simulation on each workload and report their IPC (instructions per cycle) value and L1/L2 cache miss rate. You can turn in your analysis of performance behaviors with bar graphs.

Please make sure you are using two-level cache described in Part 1 manuscript.

Part 3: Appendix A

```
process.cmd = ['test_bench/2MM/2mm_base']

process.cmd = ['test_bench/BFS/bfs','-f','test_bench/BFS/USA-road-d.NY.gr']

process.cmd = ['test_bench/bzip2/bzip2_base.amd64-m64-gcc42-nn','test_bench/bzip2/inp
ut.source','280']

process.cmd = ['test_bench/mcf/mcf_base.amd64-m64-gcc42-nn','test_bench/mcf/inp.in']
```