from m5.params import \*

from m5.proxy import \*

from MemObject import MemObject

from Prefetcher import BasePrefetcher

from Tags import \*

class BaseCache(MemObject):

type = 'BaseCache'

abstract = True

cxx\_header = "mem/cache/base.hh"

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");

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)")

tags = Param.BaseTags(LRU(), "Tag store (replacement policy)")

sequential\_access = Param.Bool(False,

"Whether to access tags and data sequentially")

cpu\_side = SlavePort("Upstream port closer to the CPU and/or device")

mem\_side = MasterPort("Downstream port closer to memory")

addr\_ranges = VectorParam.AddrRange([AllMemory],

"Address range for the CPU-side port (to allow striping)")

system = Param.System(Parent.any, "System we belong to")

# Enum for cache clusivity, currently mostly inclusive or mostly

# exclusive.

class Clusivity(Enum): vals = ['mostly\_incl', 'mostly\_excl']

class Cache(BaseCache):

type = 'Cache'

cxx\_header = 'mem/cache/cache.hh'

clusivity = Param.Clusivity('mostly\_incl',

"Clusivity with upstream cache")

writeback\_clean = Param.Bool(False, "Writeback clean lines")