Report #03

캐시 시뮬레이터

0. 사용된 오픈소스 및 특징

https://github.com/magicole/simple-cache-simulator

현재 프로젝트에서 분석하는 오픈소스 코드는 simple-cache-simulator 라는 이름을 가진 캐시 시뮬레이터이다. 이 코드에서는 Direct, Set Associative, Fully Associative 3가지에 대한 간단한 시뮬레이팅을 지원하며, 결과는 CPI 형태로 출력된다. CPI는 (hit + miss_rate * miss_panalty)/All_inst 형식으로 나타난다.

이 오픈소스를 선택한 이유는 github 상에 등록된 다양한 캐시 시뮬레이터 관련 오픈소스 중 (1) 파이썬 언어로 작성되었고, (2) 구조가 비교적 단순하며 (3) 모든 내용을 처음부터 구현하고 있지는 않은 프로젝트 중 가장 적합하다고 판단되었기 때문이다.

현재 오픈소스가 "simple"이라는 표현을 사용한 이유는 주소를 디코딩하는 등의 과정을 생략하고 각각의 방식이 동작하는 원리만을 묘사했기 때문이다. 따라서 address_size가 명시되어 있기는 하지만, 실제 제공되는 주소는 단순히 4의 배수로 나타나는 값으로 주소를 해석하여 특정 동작을 하는 대신 해당 값을 조작하여 CPI을 구하고 있다.

1. 오픈소스 분석

현재 오픈소스에서 사용하는 글로벌 변수들은 다음과 같다.

이름	설명
addresses	테스트에 사용하는 주소 목록
address_size	주소의 크기. check~ 함수 내에서 비교하는데 사용된다.
block_size	블록의 크기. 바이트 단위로 설명되어 있다.
number_of_rows	행의 개수. 각 방식에서 절대적으로 변하지 않는다.
ways	n-way associative 방식에서 way의 개수를 나타낸다.
max_storage_bits	전체적으로 담을 수 있는 최대 비트의 수를 지정하는 상수
miss_cost	miss가 발생했을 때 비용. 18 + (3 * block_size)로 묘사되고 있다. 정확한 의도를 예측하기는 어렵지만, 3이라는 수치가 전체 읽기 + 전체 쓰기 + 메모리 쓰기 정도의 과정을 묘사하고 있는 것 아닐까 추측한다. 오픈소스 상에서는 어떠한 주석도 달려 있지 않다. 따라서 현재 프로젝트에서는 기본 block_size 에 의해 연산되는 30 이라는 비용을 메모리 접근에 필요한 상수로 가정한다.
hit_cost	hit가 발생했을 때 비용. 1로 지정되어 있다.

의미 있는 함수는 종류로는 2분류, 개수로는 6개가 존재한다.

분류 별 구분

- check~ : 현재 조건에서 구성되는 캐시의 크기가 max_storage_bit을 초과하는지 검사
- simulate~: 대응되는 캐시 매핑 방식을 시뮬레이팅

Check~ 함수들

check~ 함수에서는 캐시가 가질 수 있는 모든 데이터 크기와 최대 크기를 비교한다. set associative의 경우 태그 길이 계산 방식이 실제와 조금 다르므로, 이 부분은 따로 설명한다.

CheckDirectMap

```
line_bits = ceil(log2 number_of_rows)
word_bits = ceil(log2 block_sizes)
tag_bits = address_size – line_bits – word_bits

valid_bits = 1
row_size = tag_bits + (8 * block_size) + valid_bits
table_size = row_size * number_of_rows

if table_size > max_storage_bits
    cache is too large
else:
    cache is within size constraints
```

Direct Map에서 주소는 tag, line, word 부로 나뉜다. line 비트는 행의 개수를 log2 한 값이고, word 비트는 block의 크기를 log2한 값이며, 전체 주소 길이에서 tag의 비중은 이 두 값을 제외한 수치다.
이러한 값들을 포함하여 한 행에 필요한 비트의 크기는 태그 구분용 비트 + validation 위한 비트 + 블록 자체를 위한 비트이며, 전체 캐시의 크기는 행의 크기 * 행의 개수 로 나타난다.

CheckSetAssociative

```
set_bits = ceil(log2 number_of_rows)
word_bits = ceil(log2 block_sizes)
tag_bits = address_size – set_bits – word_bits

valid_bits = 1
LRU_bits = ceil(log2 ways)
row_size = (tag_bits + (8 * block_size) + valid_bits + LRU_bits) * ways
```

```
table_size = row_size * number_of_rows

if table_size > max_storage_bits
    cache is too large
else:
    cache is within size constraints
```

Set-Associative 방식에서 주소는 tag, set, word로 나뉜다. 이때 Set-Associative 및 Fully Associative 방식은 Direct Map 방식과는 달리 데이터가 들어갈 위치가 정해져 있지 않으므로 replacing algorithm이 필요하다. 현재 오픈소스에서는 LRU(Least Recently Used) 정책을 사용하고 있으며, 이를 표기하기 위한 비트를 두고 있다.

수업 시간에 배운 set-associative 구조에 따르면, 캐시 전체의 크기가 동일할 때 set의 개수 * way의 개수 = direct mapping 기준 라인의 개수에 해당한다. 예를 들어 direct mapping에서 32라인이 존재한다면 line bit는 5비트로 표현된다. 동일한 크기의 캐시를 2 way set-assiciative 구조로 사용하는 경우 셋은 16개, way의 개수는 각각 2개로 총 32개는 동일하다. 실제 주소에 대한 비트를 따져보더라도 set-assiciative 에서는 tag 비트에 2-way 표현을 위한 1비트가 증가, set 비트는 line bit에 비해 1비트 감소해야 한다.

그러나, 현재 오픈소스에서는 이러한 모습을 보이지 않는다. 동일 조건을 맞춰주기 위해서는 사용자가 의도적으로 number_of_rows 및 ways의 값을 변경하여 설정해 줘야 하는 것으로 보인다.

CheckFullyAssociative

```
word_bits = ceil(log2 block_sizes)
tag_bits = address_size – word_bits

valid_bits = 1
LRU_bits = ceil(log2 ways)
row_size = tag_bits + (8 * block_size) + valid_bits + LRU_bits
table_size = row_size * number_of_rows

if table_size > max_storage_bits
    cache is too large
else:
    cache is within size constraints
```

Fully-Associative 매핑 방식에서 각 값은 태그 값으로만 비교한다. 따라서 Direct / Set Associative 방식과는 달리 line/set 비트가 필요하지 않다. 함수 내에서 이외의 과정은 다른 check와 동일하므로 생략한다.

simulate~ 함수의 경우 주석 형태로 pseudo code에 설명을 달았다.

simulateDirectMap

```
initialization
 tags = [ 0 for number_of_rows ] # 각 row에 대한 태그 값을 0으로 지정한다.
 valid = [ 0 for number_of_rows ] # 각 row를 not valid 상태로 지정한다.
 miss_count = 0 # miss가 발생한 횟수
 total_instructions = 0 # 전체 실행된 instruction의 개수

repeat 3 times: # 3번 반복
 for addr in address: # 미리 지정된 주소 목록들에 대해
  offset = addr % block_size # 오프셋 생성
  row = (addr // block_size) % number_of_rows # 데이터가 저장될 위치 지정
  # direct map에서 데이터가 저장되는 위치를 mod 연산으로 계산하는 것.
  tag = addr // (block_size * number_of_rows) # 태그 부분을 지정.

  if valid[row] == 0: # 행에 데이터가 설정되지 않은 경우.
   valid = 1 # 유효성 설정
   tags[row] = tag # 태그를 설정
  else if tag != tags[row]: # valid = 1이고, 태그의 값이 다른 경우(conflict miss)
    tags[row] = tag # 태그를 설정
    if not first round: # 첫번째 라운드가 아니면
     cache miss
     miss_count += 1
  else: # valid = 1이고 태그의 값이 맞은 경우(hit)
   cache hit
  if not first round:  # 첫번째 라운드가 아니면
   total_instructions += 1 # instruction 실행 횟수 증가.
 Cycle End
print row info and cpi
end
```

위 코드는 direct mapping 방식을 이용하는 시스템에서 3라운드에 걸쳐 주소에 접근할 때 발생하는 miss의 수를 구하고 있다. 현재 코드에서 첫 라운드는 캐시의 내용을 미리 채우는 부분인데, 캐시 시뮬레이터를 일반적인 상황(캐시에 내용이 차 있는 경우)을 기준으로 묘사하기 위한 방법으로 생각된다. 즉, 컴퓨터를 처음 실행했을 때 발생하는 캐시 상의 miss을 무시한다고 볼 수 있다. 고려하는 miss는 conflict miss로 해당 행이 valid 하지만 tag 상에 충돌이 발생한 경우에 해당한다. 두번째 라운드 이상에서는 해당 라인에 대해 miss가 발생했다고 횟수를 증가시킨다.

simulateSetAssociative

```
initialization
 tags = [ [-1 for ways] for number_of_rows ] # 각 row의 set에 대해 태그를 초기화한다.
 valid = [ [0 for ways] for number_of_rows ] # 각 row의 set에 대해 상태를 초기화한다.
 LRU = [deque for number_of_rows] # 각 row에 대한 LRU 값을 설정한다.
 miss_count = 0 # miss가 발생한 횟수
 total_instructions = 0 # 전체 실행된 instruction의 개수

repeat 3 times: # 총 3번 반복, 첫번째 라운드는 단순 초기화 과정으로 취급
 for addr in address: # 미리 지정된 주소 목록들에 대해
  offset = addr % block_size # 오프셋 생성
  row = (addr // block_size) % number_of_rows # 데이터가 저장될 위치 지정
  tag = addr // (block_size * number_of_rows) # 태그 부분을 지정
  flag = false # miss 발생 여부를 판단하기 위한 플래그. false이면 miss 발생.
  if (tag in tags) and (valid[row][target_idx] is valid) # 태그가 존재 + 대응되는 위치가 valid
   cache hit   #히트
   if i > 0:
    total_instruction += 1 # 첫번째 라운드 이후부터 처리
    continue # 이후 코드 무시하고 다른 주소에 대해 처리
  for ways: # 현재 set의 각 way에 대해 수행
   if valid[row][way] == 0: # 현재 way에 값이 설정된 적이 없다면
    setting tag # 태그 값 설정
    # LRU 알고리즘 동작
    if way in LRU[row]:
     LRU.remove(row) # 이전에 사용된 기록 있다면 삭제하고 다시 넣기.
    LRU.append(row) # 아니면 그냥 넣기. deque 동작으로 LRU 구현하기 위한 동작
    flag = true # 값을 설정한 것은 miss로 취급하지 않기로 함.
   if flag == false: # miss가 발생한 경우
    leastUsedWay from LRU # LRU에서 가장 사용된 지 오래된 way 찾기
    tags[row][leastUsedWay] = tag # 해당 way에 값 설정
    if i > 0: # 첫번째 라운드 이후라면
     miss_count += 1 # miss로 취급
   LRU.append(leastUsedWay) # LRU에 현재 처리한 way 넣기
   cache miss.
  if i > 0:
   total_instruction += 1
 Cycle End
print row info and cpi
end
```

SetAssociative 도 시뮬레이팅 방식은 Direct Mapping과 크게 다르지 않다. 특징적인 부분은 set associative의 구조를 구현하기 위해 각 way마다 tag, valid 정보가 존재하며, replace algorithm인 LRU에 의한 way 정보를 **각 set에 대해** 설정해 두었다는 점이다.

simulateFullyAssociative

```
initialization
 tags = [ -1 for number_of_rows ] # 각 row에 대해 태그를 초기화한다.
 valid = [ 0 for number_of_rows ] # 각 row에 대해 상태를 초기화한다.
 LRU = deque 전체 row에 대한 LRU을 설정한다.
 miss_count = 0 # miss가 발생한 횟수
 total_instructions = 0 # 전체 실행된 instruction의 개수

repeat 3 times: # 총 3번 반복, 첫번째 라운드는 단순 초기화 과정으로 취급
 for addr in address: # 미리 지정된 주소 목록들에 대해
  offset = addr % block_size # 오프셋 생성
  tag = addr // block_size # 태그 부분을 지정

  if tag in tags: # 현재 태그에 대응되는 값이 존재 -> hit
   find tag_loc
   # LRU 알고리즘 동작 위해 값 설정
   if cur_loc in LRU: # 태그에 대응되는 위치가 존재한다면
    LRU.remove(cur_loc) # 가장 최근에 사용되는 것이 될거라 삭제
   LRU.append(cur_loc) # 가장 최근에 사용된 위치라는 의미로 삽입

  else if there is invalid row: # 값이 비어 있는 row가 존재한다면
   find cur_loc # 해당 row 위치 찾기
   tags[cur_loc] = tag # 태그 설정
   valid[cur_loc] = 1 # valid로 설정
   if i > 0: # 첫 라운드 이후라면
    miss_count += 1 # miss로 취급
  # LRU 알고리즘 적용하기 위한 설정
   if cur_loc in LRU: # 값이 비어 있는 상태인데 기존에 사용했을 수가 없음
    LRU.remove(cur_loc) # 도달할 수 없는 상황. ( 기존 코드에도 명시됨 )
   LRU.append(cur_loc)
  else: # 캐시 가득 찬 상태에서 update miss
   leastUsedLoc from LRU # LRU에서 가장 사용한 지 오래된 위치 얻어오기
   tags[leastUsedLoc] = tag # 현재 태그를 해당 위치에 저장
   if i > 0: # 첫번째 라운드 이후인 경우 miss로 취급
    miss_count += 1 # miss 횟수 증가
```

```
  # 현재 작업 위치를 LRU 상에서 업데이트
  LRU.append(leastUsedLoc) # 현재 위치 LRU에 등록
  # 기존 코드에는 LRU 내에 있는지 비교하는 과정이 있으나, shift left에 의해 제거되므로 실제로 동작하지 않아 따로 명시하지 않는다.
 Cycle End
print row info and cpi
end
```

Fully Associative에서는 set Associative와는 달리 전체 위치에 대해 LRU 알고리즘을 계산한다. 실제 코드 상에서도 set Associative의 경우 LRU가 배열 형태로 나타났으나, Fully Associative에서는 단일 deque로 나타나고 있음을 볼 수 있다.

2. 오픈소스 실행

현재 오픈소스에서 글로벌 변수의 기본 값은 다음과 같다.

이름	기본값
addresses	코드 참조
address_size	16
block_size	4
number_of_rows	8
ways	1
max_storage_bits	400
miss_cost	18+(3 * block_size)= 30
hit_cost	1

Direct Map과 Fully Associative는 위 조건과 동일하게 진행하고, set associative 에서는 number_of_rows = 4, ways = 2로 변경하여 동일 캐시 크기에 대한 연산으로 수행한다.

Direct Map

```
Cache is within size constraints: 352/400
A cache miss will cost you: 30 cycles
Address: 4, tag: 0, row: 1, offset: 0    placing item
Address: 8, tag: 0, row: 2, offset: 0    placing item
Address: 20, tag: 0, row: 5, offset: 0   placing item
Address: 24, tag: 0, row: 6, offset: 0   placing item
Address: 28, tag: 0, row: 7, offset: 0   placing item
Address: 36, tag: 1, row: 1, offset: 0   Cache Miss - updating row 1
Address: 44, tag: 1, row: 3, offset: 0   placing item
Address: 20, tag: 0, row: 5, offset: 0   Cache Hit on row 5
Address: 24, tag: 0, row: 6, offset: 0   Cache Hit on row 6
Address: 28, tag: 0, row: 7, offset: 0   Cache Hit on row 7
Address: 36, tag: 1, row: 1, offset: 0   Cache Hit on row 1
```

```
Address: 40, tag: 1, row: 2, offset: 0   Cache Miss - updating row 2
Address: 44, tag: 1, row: 3, offset: 0   Cache Hit on row 3
Address: 68, tag: 2, row: 1, offset: 0   Cache Miss - updating row 1
Address: 72, tag: 2, row: 2, offset: 0   Cache Miss - updating row 2
Address: 92, tag: 2, row: 7, offset: 0   Cache Miss - updating row 7
Address: 96, tag: 3, row: 0, offset: 0   placing item
Address: 100, tag: 3, row: 1, offset: 0 Cache Miss - updating row 1
Address: 104, tag: 3, row: 2, offset: 0 Cache Miss - updating row 2
Address: 108, tag: 3, row: 3, offset: 0 Cache Miss - updating row 3
Address: 112, tag: 3, row: 4, offset: 0 placing item
Address: 100, tag: 3, row: 1, offset: 0 Cache Hit on row 1
Address: 112, tag: 3, row: 4, offset: 0 Cache Hit on row 4
Address: 116, tag: 3, row: 5, offset: 0 Cache Miss - updating row 5
Address: 120, tag: 3, row: 6, offset: 0 Cache Miss - updating row 6
Address: 128, tag: 4, row: 0, offset: 0 Cache Miss - updating row 0
Address: 140, tag: 4, row: 3, offset: 0 Cache Miss - updating row 3
END OF CYCLE 0

Address: 4, tag: 0, row: 1, offset: 0     Cache Miss - updating row 1
Address: 8, tag: 0, row: 2, offset: 0     Cache Miss - updating row 2
Address: 20, tag: 0, row: 5, offset: 0   Cache Miss - updating row 5
Address: 24, tag: 0, row: 6, offset: 0   Cache Miss - updating row 6
Address: 28, tag: 0, row: 7, offset: 0   Cache Miss - updating row 7
Address: 36, tag: 1, row: 1, offset: 0   Cache Miss - updating row 1
Address: 44, tag: 1, row: 3, offset: 0   Cache Miss - updating row 3
Address: 20, tag: 0, row: 5, offset: 0   Cache Hit on row 5
Address: 24, tag: 0, row: 6, offset: 0   Cache Hit on row 6
Address: 28, tag: 0, row: 7, offset: 0   Cache Hit on row 7
Address: 36, tag: 1, row: 1, offset: 0   Cache Hit on row 1
Address: 40, tag: 1, row: 2, offset: 0   Cache Miss - updating row 2
Address: 44, tag: 1, row: 3, offset: 0   Cache Hit on row 3
Address: 68, tag: 2, row: 1, offset: 0   Cache Miss - updating row 1
Address: 72, tag: 2, row: 2, offset: 0   Cache Miss - updating row 2
Address: 92, tag: 2, row: 7, offset: 0   Cache Miss - updating row 7
Address: 96, tag: 3, row: 0, offset: 0   Cache Miss - updating row 0
Address: 100, tag: 3, row: 1, offset: 0 Cache Miss - updating row 1
Address: 104, tag: 3, row: 2, offset: 0 Cache Miss - updating row 2
Address: 108, tag: 3, row: 3, offset: 0 Cache Miss - updating row 3
Address: 112, tag: 3, row: 4, offset: 0 Cache Hit on row 4
Address: 100, tag: 3, row: 1, offset: 0 Cache Hit on row 1
Address: 112, tag: 3, row: 4, offset: 0 Cache Hit on row 4
Address: 116, tag: 3, row: 5, offset: 0 Cache Miss - updating row 5
Address: 120, tag: 3, row: 6, offset: 0 Cache Miss - updating row 6
Address: 128, tag: 4, row: 0, offset: 0 Cache Miss - updating row 0
Address: 140, tag: 4, row: 3, offset: 0 Cache Miss - updating row 3
END OF CYCLE 1

Address: 4, tag: 0, row: 1, offset: 0     Cache Miss - updating row 1
Address: 8, tag: 0, row: 2, offset: 0     Cache Miss - updating row 2
```

```
Address: 20, tag: 0, row: 5, offset: 0   Cache Miss - updating row 5
Address: 24, tag: 0, row: 6, offset: 0   Cache Miss - updating row 6
Address: 28, tag: 0, row: 7, offset: 0   Cache Miss - updating row 7
Address: 36, tag: 1, row: 1, offset: 0   Cache Miss - updating row 1
Address: 44, tag: 1, row: 3, offset: 0   Cache Miss - updating row 3
Address: 20, tag: 0, row: 5, offset: 0   Cache Hit on row 5
Address: 24, tag: 0, row: 6, offset: 0   Cache Hit on row 6
Address: 28, tag: 0, row: 7, offset: 0   Cache Hit on row 7
Address: 36, tag: 1, row: 1, offset: 0   Cache Hit on row 1
Address: 40, tag: 1, row: 2, offset: 0   Cache Miss - updating row 2
Address: 44, tag: 1, row: 3, offset: 0   Cache Hit on row 3
Address: 68, tag: 2, row: 1, offset: 0   Cache Miss - updating row 1
Address: 72, tag: 2, row: 2, offset: 0   Cache Miss - updating row 2
Address: 92, tag: 2, row: 7, offset: 0   Cache Miss - updating row 7
Address: 96, tag: 3, row: 0, offset: 0   Cache Miss - updating row 0
Address: 100, tag: 3, row: 1, offset: 0 Cache Miss - updating row 1
Address: 104, tag: 3, row: 2, offset: 0 Cache Miss - updating row 2
Address: 108, tag: 3, row: 3, offset: 0 Cache Miss - updating row 3
Address: 112, tag: 3, row: 4, offset: 0 Cache Hit on row 4
Address: 100, tag: 3, row: 1, offset: 0 Cache Hit on row 1
Address: 112, tag: 3, row: 4, offset: 0 Cache Hit on row 4
Address: 116, tag: 3, row: 5, offset: 0 Cache Miss - updating row 5
Address: 120, tag: 3, row: 6, offset: 0 Cache Miss - updating row 6
Address: 128, tag: 4, row: 0, offset: 0 Cache Miss - updating row 0
Address: 140, tag: 4, row: 3, offset: 0 Cache Miss - updating row 3
END OF CYCLE 2

row     valid   tag
0       1       4
1       1       3
2       1       3
3       1       4
4       1       3
5       1       3
6       1       3
7       1       2
CPI: 21.40740740740741
Simulation complete
```

CPI는 21.41이 나왔다.

Set Associative (number_of_rows = 4, ways = 2)

```
Cache is within size constraints: 368/400
A cache miss will cost you: 30 cycles
Address: 4, tag: 0, row: 1, offset: 0     added item for the first time
Address: 8, tag: 0, row: 2, offset: 0     added item for the first time
Address: 20, tag: 1, row: 1, offset: 0    added item for the first time
Address: 24, tag: 1, row: 2, offset: 0    added item for the first time
Address: 28, tag: 1, row: 3, offset: 0    added item for the first time
Address: 36, tag: 2, row: 1, offset: 0    Cache Miss - updating entry
Address: 44, tag: 2, row: 3, offset: 0    added item for the first time
Address: 20, tag: 1, row: 1, offset: 0    Cache Hit
Address: 24, tag: 1, row: 2, offset: 0    Cache Hit
Address: 28, tag: 1, row: 3, offset: 0    Cache Hit
Address: 36, tag: 2, row: 1, offset: 0    Cache Hit
Address: 40, tag: 2, row: 2, offset: 0    Cache Miss - updating entry
Address: 44, tag: 2, row: 3, offset: 0    Cache Hit
Address: 68, tag: 4, row: 1, offset: 0    Cache Miss - updating entry
Address: 72, tag: 4, row: 2, offset: 0    Cache Miss - updating entry
Address: 92, tag: 5, row: 3, offset: 0    Cache Miss - updating entry
Address: 96, tag: 6, row: 0, offset: 0    added item for the first time
Address: 100, tag: 6, row: 1, offset: 0 Cache Miss - updating entry
Address: 104, tag: 6, row: 2, offset: 0 Cache Miss - updating entry
Address: 108, tag: 6, row: 3, offset: 0 Cache Miss - updating entry
Address: 112, tag: 7, row: 0, offset: 0 added item for the first time
Address: 100, tag: 6, row: 1, offset: 0 Cache Hit
Address: 112, tag: 7, row: 0, offset: 0 Cache Hit
Address: 116, tag: 7, row: 1, offset: 0 Cache Miss - updating entry
Address: 120, tag: 7, row: 2, offset: 0 Cache Miss - updating entry
Address: 128, tag: 8, row: 0, offset: 0 Cache Miss - updating entry
Address: 140, tag: 8, row: 3, offset: 0 Cache Miss - updating entry
END OF CYCLE: 0

Address: 4, tag: 0, row: 1, offset: 0     Cache Miss - updating entry
Address: 8, tag: 0, row: 2, offset: 0     Cache Miss - updating entry
Address: 20, tag: 1, row: 1, offset: 0    Cache Miss - updating entry
Address: 24, tag: 1, row: 2, offset: 0    Cache Miss - updating entry
Address: 28, tag: 1, row: 3, offset: 0    Cache Miss - updating entry
Address: 36, tag: 2, row: 1, offset: 0    Cache Miss - updating entry
Address: 44, tag: 2, row: 3, offset: 0    Cache Miss - updating entry
Address: 20, tag: 1, row: 1, offset: 0    Cache Hit
Address: 24, tag: 1, row: 2, offset: 0    Cache Hit
Address: 28, tag: 1, row: 3, offset: 0    Cache Hit
Address: 36, tag: 2, row: 1, offset: 0    Cache Hit
Address: 40, tag: 2, row: 2, offset: 0    Cache Miss - updating entry
Address: 44, tag: 2, row: 3, offset: 0    Cache Hit
Address: 68, tag: 4, row: 1, offset: 0    Cache Miss - updating entry
Address: 72, tag: 4, row: 2, offset: 0    Cache Miss - updating entry
Address: 92, tag: 5, row: 3, offset: 0    Cache Miss - updating entry
Address: 96, tag: 6, row: 0, offset: 0    Cache Miss - updating entry
```

```
Address: 100, tag: 6, row: 1, offset: 0 Cache Miss - updating entry
Address: 104, tag: 6, row: 2, offset: 0 Cache Miss - updating entry
Address: 108, tag: 6, row: 3, offset: 0 Cache Miss - updating entry
Address: 112, tag: 7, row: 0, offset: 0 Cache Miss - updating entry
Address: 100, tag: 6, row: 1, offset: 0 Cache Hit
Address: 112, tag: 7, row: 0, offset: 0 Cache Hit
Address: 116, tag: 7, row: 1, offset: 0 Cache Miss - updating entry
Address: 120, tag: 7, row: 2, offset: 0 Cache Miss - updating entry
Address: 128, tag: 8, row: 0, offset: 0 Cache Miss - updating entry
Address: 140, tag: 8, row: 3, offset: 0 Cache Miss - updating entry
END OF CYCLE: 1

Address: 4, tag: 0, row: 1, offset: 0    Cache Miss - updating entry
Address: 8, tag: 0, row: 2, offset: 0    Cache Miss - updating entry
Address: 20, tag: 1, row: 1, offset: 0   Cache Miss - updating entry
Address: 24, tag: 1, row: 2, offset: 0   Cache Miss - updating entry
Address: 28, tag: 1, row: 3, offset: 0   Cache Miss - updating entry
Address: 36, tag: 2, row: 1, offset: 0   Cache Miss - updating entry
Address: 44, tag: 2, row: 3, offset: 0   Cache Miss - updating entry
Address: 20, tag: 1, row: 1, offset: 0   Cache Hit
Address: 24, tag: 1, row: 2, offset: 0   Cache Hit
Address: 28, tag: 1, row: 3, offset: 0   Cache Hit
Address: 36, tag: 2, row: 1, offset: 0   Cache Hit
Address: 40, tag: 2, row: 2, offset: 0   Cache Miss - updating entry
Address: 44, tag: 2, row: 3, offset: 0   Cache Hit
Address: 68, tag: 4, row: 1, offset: 0   Cache Miss - updating entry
Address: 72, tag: 4, row: 2, offset: 0   Cache Miss - updating entry
Address: 92, tag: 5, row: 3, offset: 0   Cache Miss - updating entry
Address: 96, tag: 6, row: 0, offset: 0   Cache Miss - updating entry
Address: 100, tag: 6, row: 1, offset: 0 Cache Miss - updating entry
Address: 104, tag: 6, row: 2, offset: 0 Cache Miss - updating entry
Address: 108, tag: 6, row: 3, offset: 0 Cache Miss - updating entry
Address: 112, tag: 7, row: 0, offset: 0 Cache Miss - updating entry
Address: 100, tag: 6, row: 1, offset: 0 Cache Hit
Address: 112, tag: 7, row: 0, offset: 0 Cache Hit
Address: 116, tag: 7, row: 1, offset: 0 Cache Miss - updating entry
Address: 120, tag: 7, row: 2, offset: 0 Cache Miss - updating entry
Address: 128, tag: 8, row: 0, offset: 0 Cache Miss - updating entry
Address: 140, tag: 8, row: 3, offset: 0 Cache Miss - updating entry
END OF CYCLE: 2

row      valid    tag      |       valid    tag
0                 1        8       |       1        7       |
1                 1        6       |       1        7       |
2                 1        6       |       1        7       |
3                 1        8       |       1        6       |
CPI: 22.48148148148148
```

CPI는 22.48이 나왔다.

Fully Associative

```
Cache is within size constraints: 400/400
A cache miss will cost you: 30 cycles
Address: 4, tag: 1, offset: 0     Cache Miss - adding to empty row
Address: 8, tag: 2, offset: 0     Cache Miss - adding to empty row
Address: 20, tag: 5, offset: 0    Cache Miss - adding to empty row
Address: 24, tag: 6, offset: 0    Cache Miss - adding to empty row
Address: 28, tag: 7, offset: 0    Cache Miss - adding to empty row
Address: 36, tag: 9, offset: 0    Cache Miss - adding to empty row
Address: 44, tag: 11, offset: 0 Cache Miss - adding to empty row
Address: 20, tag: 5, offset: 0    Cache Hit
Address: 24, tag: 6, offset: 0    Cache Hit
Address: 28, tag: 7, offset: 0    Cache Hit
Address: 36, tag: 9, offset: 0    Cache Hit
Address: 40, tag: 10, offset: 0 Cache Miss - adding to empty row
Address: 44, tag: 11, offset: 0 Cache Hit
Address: 68, tag: 17, offset: 0 Cache Miss - replacing row
Address: 72, tag: 18, offset: 0 Cache Miss - replacing row
Address: 92, tag: 23, offset: 0 Cache Miss - replacing row
Address: 96, tag: 24, offset: 0 Cache Miss - replacing row
Address: 100, tag: 25, offset: 0          Cache Miss - replacing row
Address: 104, tag: 26, offset: 0          Cache Miss - replacing row
Address: 108, tag: 27, offset: 0          Cache Miss - replacing row
Address: 112, tag: 28, offset: 0          Cache Miss - replacing row
Address: 100, tag: 25, offset: 0          Cache Hit
Address: 112, tag: 28, offset: 0          Cache Hit
Address: 116, tag: 29, offset: 0          Cache Miss - replacing row
Address: 120, tag: 30, offset: 0          Cache Miss - replacing row
Address: 128, tag: 32, offset: 0          Cache Miss - replacing row
Address: 140, tag: 35, offset: 0          Cache Miss - replacing row
Address: 4, tag: 1, offset: 0     Cache Miss - replacing row
Address: 8, tag: 2, offset: 0     Cache Miss - replacing row
Address: 20, tag: 5, offset: 0    Cache Miss - replacing row
Address: 24, tag: 6, offset: 0    Cache Miss - replacing row
Address: 28, tag: 7, offset: 0    Cache Miss - replacing row
Address: 36, tag: 9, offset: 0    Cache Miss - replacing row
Address: 44, tag: 11, offset: 0 Cache Miss - replacing row
Address: 20, tag: 5, offset: 0    Cache Hit
Address: 24, tag: 6, offset: 0    Cache Hit
Address: 28, tag: 7, offset: 0    Cache Hit
Address: 36, tag: 9, offset: 0    Cache Hit
Address: 40, tag: 10, offset: 0 Cache Miss - replacing row
Address: 44, tag: 11, offset: 0 Cache Hit
Address: 68, tag: 17, offset: 0 Cache Miss - replacing row
Address: 72, tag: 18, offset: 0 Cache Miss - replacing row
Address: 92, tag: 23, offset: 0 Cache Miss - replacing row
Address: 96, tag: 24, offset: 0 Cache Miss - replacing row
Address: 100, tag: 25, offset: 0          Cache Miss - replacing row
Address: 104, tag: 26, offset: 0          Cache Miss - replacing row
```

```
Address: 108, tag: 27, offset: 0          Cache Miss - replacing row
Address: 112, tag: 28, offset: 0          Cache Miss - replacing row
Address: 100, tag: 25, offset: 0          Cache Hit
Address: 112, tag: 28, offset: 0          Cache Hit
Address: 116, tag: 29, offset: 0          Cache Miss - replacing row
Address: 120, tag: 30, offset: 0          Cache Miss - replacing row
Address: 128, tag: 32, offset: 0          Cache Miss - replacing row
Address: 140, tag: 35, offset: 0          Cache Miss - replacing row
Address: 4, tag: 1, offset: 0     Cache Miss - replacing row
Address: 8, tag: 2, offset: 0     Cache Miss - replacing row
Address: 20, tag: 5, offset: 0    Cache Miss - replacing row
Address: 24, tag: 6, offset: 0    Cache Miss - replacing row
Address: 28, tag: 7, offset: 0    Cache Miss - replacing row
Address: 36, tag: 9, offset: 0    Cache Miss - replacing row
Address: 44, tag: 11, offset: 0 Cache Miss - replacing row
Address: 20, tag: 5, offset: 0    Cache Hit
Address: 24, tag: 6, offset: 0    Cache Hit
Address: 28, tag: 7, offset: 0    Cache Hit
Address: 36, tag: 9, offset: 0    Cache Hit
Address: 40, tag: 10, offset: 0 Cache Miss - replacing row
Address: 44, tag: 11, offset: 0 Cache Hit
Address: 68, tag: 17, offset: 0 Cache Miss - replacing row
Address: 72, tag: 18, offset: 0 Cache Miss - replacing row
Address: 92, tag: 23, offset: 0 Cache Miss - replacing row
Address: 96, tag: 24, offset: 0 Cache Miss - replacing row
Address: 100, tag: 25, offset: 0          Cache Miss - replacing row
Address: 104, tag: 26, offset: 0          Cache Miss - replacing row
Address: 108, tag: 27, offset: 0          Cache Miss - replacing row
Address: 112, tag: 28, offset: 0          Cache Miss - replacing row
Address: 100, tag: 25, offset: 0          Cache Hit
Address: 112, tag: 28, offset: 0          Cache Hit
Address: 116, tag: 29, offset: 0          Cache Miss - replacing row
Address: 120, tag: 30, offset: 0          Cache Miss - replacing row
Address: 128, tag: 32, offset: 0          Cache Miss - replacing row
Address: 140, tag: 35, offset: 0          Cache Miss - replacing row
row     valid   tag
0       1       32
1       1       29
2       1       35
3       1       30
4       1       28
5       1       25
6       1       27
7       1       26
CPI: 22.48148148148148
SIMULATION COMPLETE
```

CPI = 22.48이 나왔다.

3. 오픈소스 개선

현재 오픈소스는 level 1을 기준으로 하고 있다. 이를 개선하기 위해 L2 캐시 계층을 도입하여 성능을 개선하고자 한다. 이때 L2 캐시를 접근하는 경우는 L1 캐시 내에 데이터가 없는 경우이다.

L2 캐시는 Fully Associative로 구현했으며, 라인의 개수는 L1 캐시의 2배에 해당하도록 설정했다. 작성한 코드는 다음과 같다.

```
from collections import deque

class L2Cache:
    """
    fully associative 로 운영되는 L2 캐시. 크기는 L1 캐시의 2 배.
    """

    def __init__(self, number_of_rows: int):
        self.tags = [0 for _ in range(number_of_rows * 2)]
        self.valid = self.tags.copy()  # 태그의 모습과 동일하게 정의된다.
        self.LRU = deque()  # L2 캐시에서도 LRU 사용

    def write(self, tag):
        """
        데이터를 쓰는 동작. 첫번째 라운드 이후라면 miss 발생할 수 있다.
        만약 이미 존재한다면 hit
        아니면 miss
        그러나, L1 캐시 입장에서는 L2 안에 존재하는지 여부만 중요하다.
        """

        # 대응되는 태그가 존재하는 경우 -> 데이터가 L2 캐시 내에 있는 경우
        if tag in self.tags:
            location = self.tags.index(tag)
            if location in self.LRU:
                self.LRU.remove(location)  # 기존에 사용되었던 기록 삭제하고
            self.LRU.append(location)  # 최근에 사용된 것이라고 바꿈.
            # 읽기 과정에서 검증하므로 이 코드가 실행될 일은 없어야 함.

        # L2 캐시에서 비어있는 공간이 있는 경우 -> 빈 공간에 쓰기
        elif 0 in self.valid:
            location = self.valid.index(0)
            self.valid[location] = 1  # 해당 공간 valid 처리
            self.tags[location] = tag  # 해당 공간 태그 넣기
            self.LRU.append(location)  # 최근에 사용된 위치라고 LRU 에 추가
        # 태그는 없지만 모든 공간이 valid. L2 캐시가 꽉 찬 경우
        else:
            leastUsedLoc = self.LRU.popleft()  # 사용한지 가장 오래된 값을 제거
            self.tags[leastUsedLoc] = tag  # 해당 위치의 태그 값을 수정
            # 오래된 위치 제거했으니까, LRU 배열 상에 없음. 그냥 삽입만 하면 됨.
```

```
            self.LRU.append(leastUsedLoc)  # LRU 배열 상에 삽입.

    def act(self, tag, round):
        """
        데이터를 읽되, 없으면 쓰는 동작.
        miss로 처리해야 하면 true, 아니면 false을 반환한다.
        miss로 처리되는 조건
        1. 첫번째 라운드 이후에
        2. hit가 아닌 경우
        """
        if tag in self.tags:  # hit 한 경우. 무조건 false 반환
            location = self.tags.index(tag)
            if location in self.LRU:
                self.LRU.remove(location)  # 기존에 사용되었던 기록 삭제하고
            self.LRU.append(location)  # 최근에 사용된 것이라고 바꿈.
            return False  # 초기화 과정이든 아니든 무조건 false 반환하게 된다.

        # miss인 경우. round > 0이라면 true 반환.
        else:
            self.write(tag)
            if round > 0:
                return True  # miss로 처리되는 경우
```

생성자 함수는 number_of_rows을 받는다. 해당 값은 L1 캐시의 row의 수로, L2 캐시의 크기를 L1 캐시의 2배로 만들기 위해 요구된다.

write 메서드는 제시된 태그 값을 L2 캐시에 쓰기 위해 사용된다. L1 캐시에서 해당 메서드를 접근할 일은 없으며, 일반적인 Fully Associative처럼 빈 공간이 있는 경우나 모든 공간이 꽉 차 있는 경우 등을 구분하여 동작한다.

action 메서드는 외부에서 L2 캐시 동작을 위해 접근하는 메서드이다. 만약 요청한 태그가 L2 캐시 내에 존재한다면 miss가 아니라는 정보를 반환한다. 아니라면 해당 데이터를 L2 캐시에 쓴 후 round 값을 비교하여 현재 라운드가 첫 라운드 이후라면 true을, 아니면 false을 반환한다. true값을 반환한 경우 L2 캐시 miss가 발생했음을 의미한다.

기존 소스코드 상에서는 다음과 같은 코드 추가 및 수정이 존재한다.

1. L2 캐시의 추가

```
    l2cache = L2Cache(number_of_rows)
```

각 simulate 함수에 L2 캐시를 추가한다

2. l1_miss_count 추가

```
l1_miss_count = 0  # l1 없는데 l2는 있을 때
```

각 simulate 함수에 l1_miss_count를 추가한다. 이 변수는 l1 miss 및 l2 hit가 발생한 경우를 의미하는 값이다.

3. l1 hit 조건이 아닌 경우 l2 캐시를 찾는 동작 추가

```
elif tag != tags[int(row)]:
                # 만약 L2 캐시 뒤져서 있다면 사용. 없다면 기존 그대로.
                l2miss = l2cache.act(tag, i)
```

l1 hit 조건이 아닌 경우에 대해 l2cache.act 메서드를 실행하여 l2 캐시에 해당 데이터가 존재하는지 검사하는 과정을 거친다. 언급한 알고리즘에 의해 miss 결과가 나온다.

4. cpi 수정

```
cpi = (hit_cost * (total_instructions - l1_miss_count - miss_count)  # l1 에서 hit
           + l2hit_cost * l1_miss_count  # l2 에서 hit
           + miss_cost * miss_count) / total_instructions  # 전체 miss
```

cpi 연산 방식을 수정했다. 기존에는 hit + miss * miss_panalty 형식이었으나, 현재는 hit_cost * l1_hit + l2hit_cost * l2hit + miss_cost * miss 꼴로 나타냈다.

L2 캐시 추가에 의한 결과는 다음과 같다. 결과의 윗부분은 생략하고 CPI 부분만을 보인다.

Direct Map

row	valid	tag
0	1	4
1	1	3
2	1	3
3	1	4
4	1	3
5	1	3
6	1	3
7	1	2

CPI: 4.518518518518518

L2 캐시를 추가했을 때 CPI가 기존에 비해 크게 감소했다.

Set Associative (number_of_rows = 4, ways = 2)

row	valid	tag	|	valid	tag	
0		1	8	|	1	7 |
1		1	6	|	1	7 |
2		1	6	|	1	7 |
3		1	8	|	1	6 |

CPI: 4.703703703703703

Simulation Complete

L2 캐시를 추가했을 때 CPI가 기존에 비해 크게 감소했다.

Fully Associative

row	valid	tag
0	1	32
1	1	29
2	1	35

```
3       1       30
4       1       28
5       1       25
6       1       27
7       1       26
CPI: 22.48148148148148
```

Fully Associative 에서는 L2 캐시의 효과를 제대로 보지 못하고 있다. 현재 주어진 조건에 따른 값들 자체가 그리 많지 않고, 반복될 필요 없이 이미 L1 캐시 수준에서 처리되어 L2 캐시가 hit 하는 경우가 거의 없었다. 대신 L2 miss가 발생하여 전체 miss로 처리되므로 사실상 기존과 달라진 점이 보이지 않는다.

4. 추가 정보

현재 프로젝트에 사용된 코드는 다음 위치에서 볼 수 있다.

기존 코드: https://github.com/magicole/simple-cache-simulator
수정한 코드: https://github.com/blaxsior/simple-2level-cache-simulator