27강_Replace Algorithm 및 Write Policy

Created	@Aug 15, 2020 4:31 PM
Tags	RE

▼ 스크립트

1. 네. 이제 그 Mapping하는 Algorithm에선 다 얘기를 했습니다. 사실 Cache에서 가장 중요한 것은 Mapping Algorithm이 가장 중요하고요. 그 다음에 이제 해야 될 게 Replace입니다. Replace.
2. Replace 라는 건 뭐냐면 교체한다는 의민데요. 교체를 어떤 걸 교체하는 거냐면 아까 그런 말씀 드렸죠. Swap-out된다고. 그죠. Direct Mapping같은 경우에는 새로운 어떤 Data를 Cache에 저장하려고 할 때 거기에 있는 이미 차있는 게 있으면 어쩔 수 없이 그걸 지우고 쓸 수밖에 없거든요.
3. 그러니까 Direct Mapping에서는 Replace를 뭘 할껀가를 고민할 필요가 없어요. 무언가 새로 바꿀게 있으면 지우고 쓰면 그만입니다. 그런데 Fully Associative Mapping이나 Set Associative Mapping같은 경우엔 도대체 기존의 Data 중에 뭐를 지워줘야 되는지를 결정을 해야 되거든요. 그래서 그런 교체를 어느 것을 해줄 거냐 라는 결정을 하는 Algorithm을 Replace Algorithm이라고 합니다.
4. 지금 보시면 Direct Mapping은 해당 사항 없고요. 자 Cache Hit Ratio를 이제 우리가 Replace할 때는 어떤 방식으로 Replace해줘야 되냐면, Cache의 Hit Ratio를 극대화 할 수 있는 방법으로 해줘야 되요. 그래서 교체할 Block을 맘대로 선택하면 안되고요. 이 Replace Algorithm의 부합하게 선택을 해줘야 됩니다.
5. 먼저 LRU라는 게 있습니다. Least Recently Used 그러니까 Least Recently Used는 최근에 사용됐다는 얘긴데요. Least가 들어가 있으니까 가장 최근에 사용되어 있지 않던 이란 의미가 되겠죠. 그래서 사용되지 않은 채로 가장 오래 있었던 Block 의미가 없잖아요. 그걸 지워버리고 Swap-out 해버리고 내 껄 넣어야죠. 그죠. 지금 저장하고 싶은 거. 그렇게 교체하는 방식이 LRU Algorithm입니다.

6. 두 번째 이제 First In First Out FIFO인데요. FIFO. Cache에 적재 된지가 가장 오래된 Block 그거를 가장 먼저 교체하는 방식이 FIFO 방식입니다. 그러니까 최근에 들어온 거 최근에 예전에 저장 되있던 것은 CPU가 다시 사용할 가능성이 굉장히 낮잖아요. 그러니까 빨리빨리 주어버리는 게 좋죠. 최근에 들어왔던 최근에 저장됐던 거는 CPU가 다시 사용할 확률이 높기 때문에 그건 좀 오랫동안 남겨 두고, 그렇죠? 그러니까 먼저 들어온 것을 먼저 지우는 방식 그게 FIFO Algorithm이었고요.
7. 세 번째 Least Frequently Used 이건 뭐냐면 Frequently라는 게 이제 자주 사용되던 이라는 뜻이에요. 그런데 Least 가 들어가 있기 때문에 '가장 자주 사용되지 않던'이 되는 겁니다. 그래서 참조되었던 횟수가 가장 적은 Block을 먼저 교체하는 방식 그걸 지워 버리고 새로운 Data가 들어오면 그걸 가지고 Cache에 적재하는 그런 방식이 LFU 방식이 되겠습니다.
8. 자 Replace Algorithm의 예를 한번 들어보겠습니다. 예를 들어보면 LRU Replace Algorithm을 사용하는 Set Associative Mapping이라고 했어요. 그러니까 그림을 잘 이해하셔야 되는 게요. 이게 이 2개가 각각 Slot을 의미하고요. 그리고 이 묶음 하나가 Set입니다. 그 상황이에요. 그래서 이건 뭐냐면 0번지가 들어왔다 1번지가 들어왔다 뭐 이런 의미입니다. 2번지가 들어왔다. 그죠. 그런 의미입니다.
9. 그러니까 CPU가 Main Memory한테 첫 번째 0번째에 있는 Data를 요구를 한겁니다. 여기 1이라고 써놓으면 CPU가 Main Memory한테 1번지에 있는 Data를 요구를 한겁니다. 이제 그런 의민데요. 그런 상태에서 LRU Algorithm을 사용한다. 그 다음에 아래와 같은 Block들이 연속적으로 들어온다고 할 때 이게 Block이죠 Block의 주소들이죠. 주소들이 연속적으로 들어온다고 할 때 각 Slot에 적재되는 Block을 표시하고 Hit Ratio를 구해라 라고 되있습니다.
10. 근데 이제 문제는 이렇게 되있는데요. 전 답까지 표시된 상태에서 이걸 어떻게 된 건지 설명을 하도록 할거고요. 우리 이제 뒤에 퀴즈 쪽에 가보면 이거에 대해서 추가적으로 문제가 나와 있으니까 거기서 여러분들이 연습을 하시면 될 것 같습니다.
11. 아.. 지금 여기 보시면요. 이거와 같은 경우에는 하나에 Set에 Slot이 2개가 들어가 있고 이건 하나의 Set에 Slot이 3개가 들어가 있습니다. 자 그럼 순서대로 한번 진행을 해보겠습니다.
12. 자 0번째 Data를 요구를 했습니다. 여긴 기존에 비어 있었습니다. 근데 비어있기 때문에 Cache에서는 가져갈 Data가 없겠죠. 그럼 0번지 저장해 가야죠. 그렇게 된 겁니다. 그 다음에 1번지의 Data를 Main Memory에 요구를 했습니다. 그럼 Cache에 먼저 찾아봐야 되겠죠. 1번지에 여기 저장된 게 없었죠. 그러면 1을 저장을 해놓고 이렇게 형태가 바뀌게 됩니다.

13. 자 그 다음에 이제 2번지의 Data를 요구를 합니다. Main Memory에. 그럼 Cache에 01이 있었는데 2번지에 있는지 확인하려고 하는데 없죠. 그럼 이 둘 중에 뭔가 하나를 지우고 2번지의 Data를 저장을 해놔야 되는데 여러분 같은 경우에는 뭘 지우겠냐고요. 당연히 0은 사용되지 않고 오랫동안 남아있었죠. 그죠. 사용되지 않고 남아있으니 이건 최근에 들어왔잖아요. 그러니까 이거 지워야 되지. 이거 지우고 2가 여기 채워지는 거죠. 그 얘기를 하는 겁니다.

14. 자 그럼 4가 들어왔습니다. 4. 4가 들어왔으면 어떻게 되느냐. 없어요. Cache에요. Hit가 아니라 Miss가 났어요. 그러면 뭔가 둘 중에 하나를 지워야 되는데 어느 걸 지워야 될까요. 당연히 이걸 지워야지. 이게 오랫동안 사용하지 않고 남아있었으니까. 저게 다시 사용될 가능성이 낮으니까. 그래서 4를 여기에 채워 넣은 겁니다.

15. 자 그 다음에 2는 2번지에 있는 Data 또 요구를 했습니다. Main Memory에. Main Memory에 찾기 전에 Cache를 찾아보니까 아 2가 있네요. Hit. 한번 Hit가 났습니다. 다음 3이 들어옵니다. 3. 3이 들어오는데 3이 있는지를 찾습니다. 2 4 에요. 없어요. 없으면 Miss죠.

16. 그럼 이 둘 중에 뭔가 하나를 지우고 3을 갖다 거기다 적어 놔야 되는데 여러분 같으면 뭐를 적어 놓겠냐고요. 저거를. 지금 이게 기분 같아서는 3인데 이게 2가 오랫동안 있었으니까 3을 써야 될 것 같지만 그렇지 않죠. 왜냐면 여기서 Hit가 났거든요. Hit가 났으면 앞으로도 Hit가 날 가능성이 높거든요. 그러니까 이걸 지우면 안됩니다. 4를 지워야 됩니다. 그래서 4를 지우고 3을 여기다 갖다 놓는거죠.

17. **자 그 다음에 7이 들어왔습니다. 7은 아직까지 여기 Cache에 없죠. 그러니까 뭔갈 지우고 써야 되는데 2하고 3중에 뭐를 지우겠냐는 거에요. 2는 물론 여기서 Hit가 났지만 3이 그 다음에 들어왔거든요. 2를 지워야죠. 7을 여기다 넣어야죠. 그죠. 이런 식으로 진행을 쭉 한거죠. 하다보면 총 이게 11번 1번 2번 3번 4번 해서 11번 Main Memory Access 한 것 중에 Cache에서 2번 Hit가 나게 됩니다. 그럼 Hit Ratio는 2/11가 되겠죠.**

18. 그러니까 LRU Algorithm, LRU Replace Algorithm을 이용해서 여러분들이 이걸 하나, 하나 고민을 해보면 이런 결과를 얻을 수가 있고요. 이걸 머릿속에 여러분들이 한번 LRU Algorithm에 연습해 보는 계기가 될 수 있거든요. Slot이 3개인 경우에 대해서는 각자 한번 해보세요. 어차피 답이 다 나와 있으니까 이것과 똑같은 방식으로 진행을 하게 되면 3/11라는 Hit Ratio를 얻게 될 수 있거든요. 한 번 진행해 보시기 바랍니다.

19. 자 그 다음에 Write Policy입니다. 자 이거 아까 그 말씀드린 건데요. Cache에 일부 Block이 변경이 됐어요. 그럴 수 있습니다. 왜냐면 Main Memory에서 어떤 프로그램에서 뭘 수행하다가 Cache에 다시 뭔가를 write를 했는데 그게 기존의

값이 수정이 된거죠. 그런 경우에 그 내용을 Main Memory에 어떻게 업데이트를 하겠느냐, 그 시기와 방법의 결정이 필요하다는 거죠.

20. 사실 Write Policy는 학부 수준에서 얘기하기엔 조금 어려운 얘기고요. 지금 여기서 얘기하고 싶은 건 그냥 이런 게 있다 정도만 알아 두시면 될 것 같아요.

21. 첫 번째 이런 거 한번 보죠. 이게 어셈블리어인데요. LOAD를 합니다. LOAD를 어떻게 하냐면 ACC에 accumulator에 0x 16진수로 1번지에 있는 Data를 ACC에 LOAD를 합니다. 그러면 Memory에 1번지에 있는 Data가 ACC accumulator로 저장이 됩니다. 그걸 10이라고 가정할께요. 10. 자 그다음에 ADD를 합니다. ADD를 ACC의 값에 상수 5를 더하게 됩니다. 상수 5를. 그러면 ACC에 저장된 값은 10이 아니라 15로 바뀌겠죠.

22. 자 그런 다음에, ACC의 있는 값을 이제 1번지에 있는 Memory에 Memory의 1번지로 다시 저장을 합니다. 그러면은 이게 어떻게 되냐면 처음에 Memory가 있으면 여기 1번지가 있습니다. 1번지에 원래 저장된 게 얼마였냐면 10이었습니다. 10을 가져왔으니까요. 근데 여기서 연산이 돼서 이게 15로 바뀐다는 거죠. 15로 바뀐 겁니다. Cache에서 Hit가 됐기 때문에, 이건 Cache에요 Cache.

23. 근데 Main Memory는 어떻게 돼있을까요? Main Memory는 10이 그대로 유지가 되고 있겠죠. 그죠. 여기서 분명히 STOR를 통해서 accumulator의 있는 값을 1번지로 저장하게 되면 이게 바뀐다는 거죠. 이게 바뀐다는 의미는 아니에요. 그죠. 근데 이게 제대로 consistency, 일관성이 유지가 될려면 이것도 바꿔야 되거든요. 10도 15로 바꿔야 되거든요. Cache에 있는 것만 바꿔야 되는 게 아니라 Main Memory에 있는 것도 바꿔야 되거든요. 이걸 어떤 방식으로 바꿔야 되냐는 거에요.

24. 이제 그거에 대한 얘기가 Write Policy라는 겁니다. 먼저 첫 번째 방식은 write through입니다. 그러니까 모든 write 동작들이 Cache뿐만 아니라 Main Memory로도 동시에 수행을 하겠다. 그림을 보면 CPU에서 아까 10이라는 값을 15로 변형을 했어요. 그럼 Cache에 있는 값이 먼저 바뀌겠죠. Cache에 있던 값이었으니까, 동시에 Main Memory도 바꿔놔야 된다는 거죠. 이런 방식이 write through로 가겠고요.

25. 이거의 장점은 뭐냐면 Cache에 적재된 Block의 내용과 여기 있는 내용과 여기 있는 내용이 언제나 항상 균일하게 유지하겠죠. 그죠. 일관성이 항상 유지가 되죠. 근데 이거의 단점은 뭐냐. 여기만 쓰면 되는데 여기에도 써놔야 된다는 거죠. 근데 이게 시간이 많이 걸려요. Main Memory가. Cache는 금방 바꿀 수가 있는데, 시간이 적게 걸리고 많이 걸리고, 이게 시간이 너무 길어진다는 거에요. write하는 시간이. 이게 문제가 있을 수가 있습니다. 이게 write through방식이고요.

26. 두 번째 방식은 write back이라는 방식이 있어요. write back 방식은 뭐가 변경이 되면 이게 아까 15로 바뀌었다 그랬죠. Cache만 바꿔놓자. 이건 바꿔놓지 말자. '어, 그러면 큰일나지 않습니까? 이거랑 이거랑 Data가 일관성이 유지가 되지 않지 않습니까?' 라고 질문을 할 수가 있죠.

27. 근데 이걸 언제 바꾸냐, 결국 Main Memory의 값을 바꿔야 되거든요. 언제 바꾸면 될까요. 네, 여기서 이게 지워질 때 Swap-out될 때 그때 이걸 바꾸자는 거죠. 왜냐면 이걸 10으로 바꿨다가 20으로 바뀔 수도 있고 30으로 바뀔 수도 있거든요. 그때마다 계속 Main Memory를 같이 바꿀 수는 없어요.

28. 최종으로 지워지기 전에 그때 가서 여기를 이걸 15 혹은 20 이런 식으로 최종 결과 값으로 바꿔주게 되면 Main Memory를 바꾸는 횟수가 적죠. 그러니까 write time이 적게 든다는 거죠. 그런 방식을 이용하는 게 write back방식입니다. 그래서 Cache에서 Data가 변경되어도 Main Memory에는 업데이트 되지 않는 방식이고요. 장점은 write 동작의 횟수가 최소화되기 때문에 write time이 짧아진다는 장점이 있는데,

29. 단점은 이겁니다. Block을 교체할 때는 Cache 상태를 확인하는 거죠. 그러니까 이 얘기에요. Cache에 이게 변형이 됐는지 안됐는지 몰라요 사실은. 이게 그러니까 기존의 10의 값을 유지 할 수 있다고요. 15로 바뀔 수도 있고. 15로 바뀌지 않고 그냥 10이었는데 구지 여기로 다시 쓸 필요가 없거든요. 괜히 write time을 낭비하는 거거든.

30. 그럼 어떻게 하냐면 flag Bit를 하나 두는 거죠. 각 Slot 마다 플래그비트를 하나 둬서 10에서 15로 바뀌게 되면 그 플래그를 1로 세팅해놓는 거죠. 그러면 Swap-out 될 때 1로 세팅되는 부분이 있다면 그걸 다시 Memory에 write를 해주 게 되는거죠. 그래야만 일관성이 유지가 되니까 그래서 Block을 교체할 때는 Cache의 상태를 확인하여 Main Memory를 업데이트 하는 동작이 선행되어야 한다, 그를 위해서 각 Cache Slot이 상태 비트를 가지고 있어야 된다는 얘기죠. 이런 단점이 있게 됩니다.

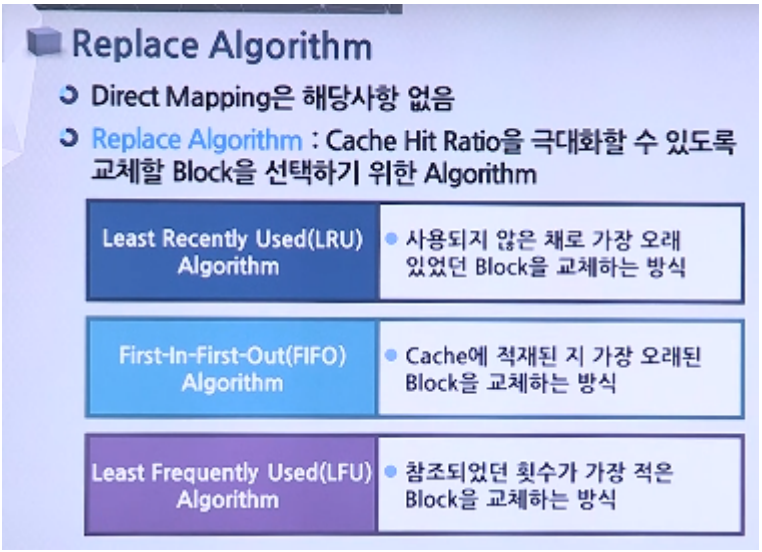
31. 자 Write Policy에서 Multiple Processor 우리가 Processor를 여러 가지 사용하는 경우 이런 경우에서도 Data의 일관성이 깨지는 경우가 빈번히 발생할 수가 있습니다. 어떤 경우가 있는지 현상만 살펴보겠습니다. 그거에 대한 해결책은 학부수준을 벗어난다고 보여지고요. 현상만 보도록 할께요.

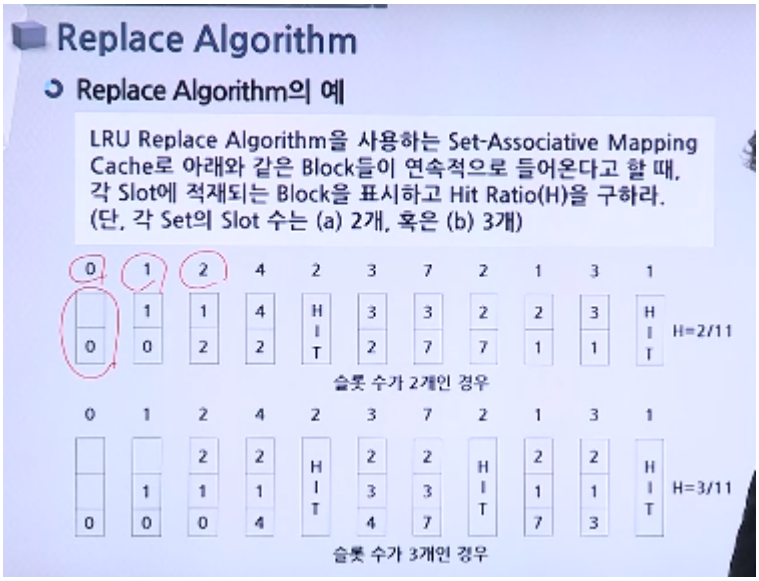
32. 지금 이게 보시면 이제 이건 write back입니다. 이건 Main Memory이고요, 이건 Processor 1번이고요, Processor 2번, Processor 3번이에요. 이건 Cache 1번 Cache, 2번 Cache, 3번 그러니까 이 Processor는 이 Cache를 사용하는 거죠.

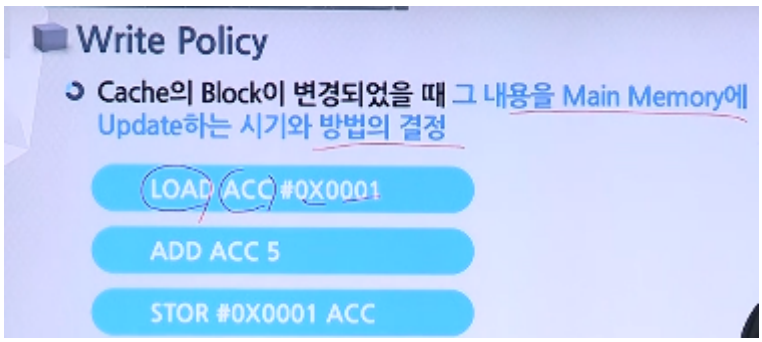
33. On chip Cache기 때문에 이 Processor는 그냥 이 Cache를 사용해야만 합니다. 처음에 x라는 것을 일로도 가져오고 일로도 가져왔습니다. Data 사용을 했습니다. 그런 중에 1번 Processor가 여기 있는 Cache를 수정을 했어요. 그래서 x가 아니라 x'로 바뀌었습니다. 그럼 이거를 write back을 할 것 같으면 나중에 바꿀 거기 때문에 이건 바뀌지 않죠. 그대로 이것만 바꾼 상태로 유지가 되죠.

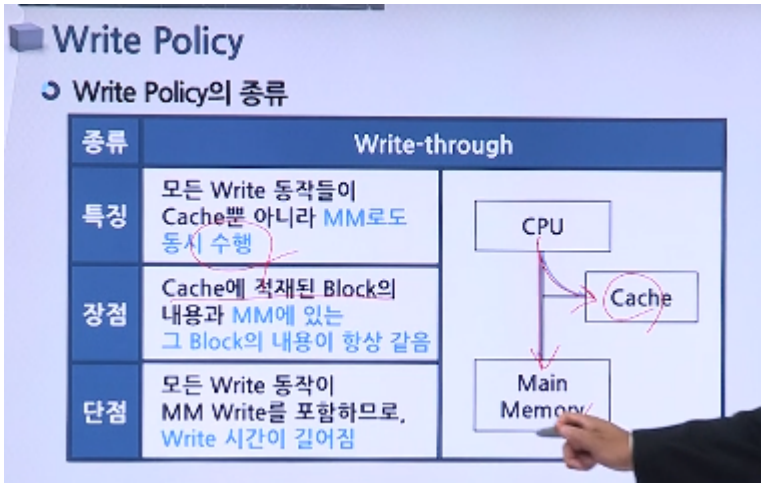
34. 근데 write through를 한다고 생각해보세요. write through를 하게 되면 이게 바뀌었으면 이것도 바꿔야 되거든요. 이렇게 바꿔야 되겠죠. x' 두 경우 어느 경우나 각 Data 일관성은 다 깨집니다. 이게 Multiple Processor 시스템에서는 이런 문제가 있기 때문에 이런 경우는 어떻게 Data 일관성을 유지하겠느냐는 문제가 여러분들이 향후에 컴퓨터 Architecture에 더 공부를 하게 되면 이런 것들에 대한 Algorithm을 개발하는 게 굉장히 중요한 문제가 될 겁니다.

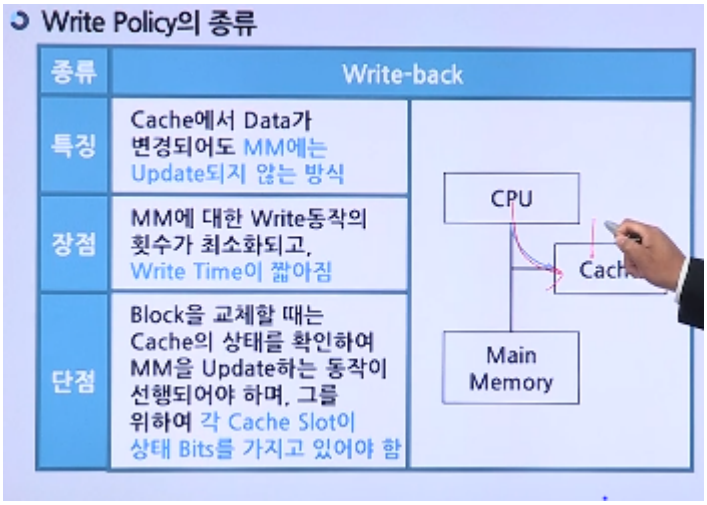
35.

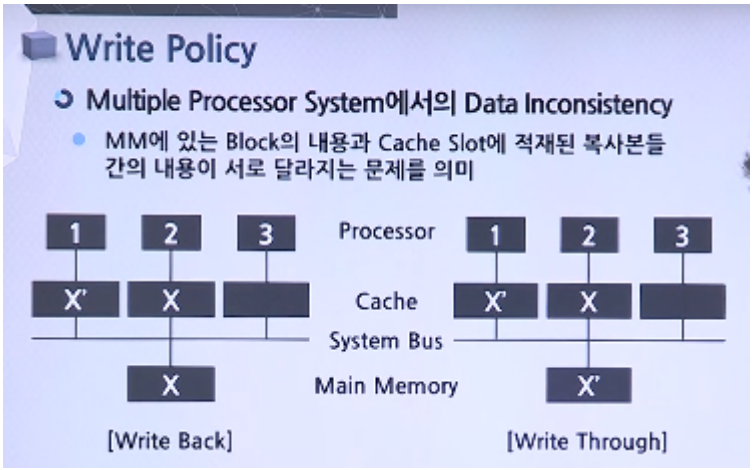



Replace Algorithm
Replace Algorithm의 예
LRU Replace Algorithm을 사용하는 Set-Associative Mapping Cache로 아래와 같은 Block들이 연속적으로 들어온다고 할 때, 각 Slot에 적재되는 Block을 표시하고 Hit Ratio(H)을 구하라. (단, 각 Set의 Slot 수는 (a) 2개, 혹은 (b) 3개)
0 1 2 4 2 3 7 2 1 3 1
H=2/11
슬롯 수가 2개인 경우
0 1 2 4 2 3 7 2 1 3 1
H=3/11
슬롯 수가 3개인 경우


Write Policy
Cache의 Block이 변경되었을 때 그 내용을 Main Memory에 Update하는 시기와 방법의 결정
LOAD ACC #0X0001
ADD ACC 5
STOR #0X0001 ACC


Write Policy
Write Policy의 종류
종류
Write-through
특징
모든 Write 동작들이 Cache뿐 아니라 MM로도 동시 수행
장점
Cache에 적재된 Block의 내용과 MM에 있는 그 Block의 내용이 항상 같음
단점
모든 Write 동작이 MM Write를 포함하므로, Write 시간이 길어짐
CPU
Cache
Main Memory


Write Policy의 종류
종류
Write-back
특징
Cache에서 Data가 변경되어도 MM에는 Update되지 않는 방식
장점
MM에 대한 Write동작의 횟수가 최소화되고, Write Time이 짧아짐
단점
Block을 교체할 때는 Cache의 상태를 확인하여 MM을 Update하는 동작이 선행되어야 하며, 그를 위하여 각 Cache Slot이 상태 Bits를 가지고 있어야 함
CPU
Cach
Main Memory


Write Policy
Multiple Processor System에서의 Data Inconsistency
MM에 있는 Block의 내용과 Cache Slot에 적재된 복사본들 간의 내용이 서로 달라지는 문제를 의미
1
2
3
Processor
X'
X
Cache
System Bus
X
Main Memory
X'
[Write Back]
[Write Through]