26강_Mapping

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

▼ 스크립트

1. 네. 자 그러면은 앞서 이제 학생과 어떤 칸막이 화장실을 이용해서 3가지 Mapping방식을 쉽게 설명하려고 노력해 보았는데요. 이제 구체적으로 어떤 방식으로 Mapping을 하는지 Memory 이용해서 얘기를 해보겠습니다.
2. 그림 상에서 이제 좌측에 있는 것 이것은 뭘 얘기하냐면 Main Memory을 얘기합니다. 그다음에 이건 뭘 얘기하냐면 Cache를 의미하죠. 그리고 이 아래에 있는 건 뭐냐면요. 이런 Cache가 있었고요. 그리고 여기 보면 이제 Main Memory, Main Memory의 Data를 Access 하기 위해서 이런 Address들이 쭉 들어왔을 때 Cache가 어떤 형태로 변형되는지를 보여주는 겁니다.
3. 그래서 먼저 이것에 대해서 설명을 하면 이런 식으로 Address를 찾으려고 할 때 어떤 식으로 Cache에 Data가 다시 쓰여지는지 그런 말씀을 드릴게요.
4. 자, 먼저 여기 보시겠습니다. 이 Main Memory를 보면요. comp가 4Byte로 구성되있는, 그러니까 영어 문자 하나, 하나는 1Byte를 차지하게 되니까 4개의 문자가 되면 4Byte가 되죠. 총 32Bit가 되겠네요. 그래서 폭 좌우 폭은 32Bit가 되고요.
5. 그 다음에 여기 좌측에 있는 Data 그러니까 저장할 수 있는 공간의 개수가 몇 개가 되냐면요. 여기 보시면 이렇게 되죠. 32개에요. 32개 하나하나 다 세보면 32개가 됩니다. 그래서 이걸 토탈 세로는 32, 그 다음에 가로는 Byte단위로 표현하면 4Byte잖아요. 그래서 32 x 4를 하면 총 128Byte의 기억장치 주기억장치 Main Memory 용량을 갖게 되는 그런 Memory를 한번 그려 본 거고요. 그 Memory 안에는 문자들이 쭉 저장되있다고 가정을 했습니다.
6. 자 그 다음, 총 32개에요. 지금 실제로는 이 comp에서 c, o, m, p 이 하나 Byte 단위로 Addressing이 되어있다고 되어있는 Memory라고 가정을 할 겁니다. 그럼 총 이게 저장할 수 있는 공간을 Byte단위로 하면 어떻게 되나요. 2의 7승이 되니까 128개. 128개의 Address를 지정할 수 있거든요. 여기. 그러니까 이 c가 0번지고 o가 1번지고 m이 2번지고 p가 3번지고 이런 식으로 하게 되면 이 맨 뒤에 있는 'enda'에서 a는 총 127번지 0번지부터 시작했으니까 127번지까지 나오게 되는거죠. 그런 구조를 그렸고요.

7. 여기 지금 Main Memory Address는 총 몇Bit로 이루어져있냐면 7Bit로 이루어 있습니다. Memory 공간이 당연히 128개기 때문에 7Bit로 이루어져 있고 이거 하나를 Block으로 보겠습니다. Block. 이 Block Address입장에서 보면 Block이 32개기 때문에 Block을 지정할 수 있는 Address의 기능은 5Bit면 충분하고요. 여기 뒤에 있는 2개 Bit를 갖고 00일 경우에는 c 01경우에는 o 10일 경우엔 m 그다음에 11일 경우엔 p를 지정하는 것으로 가정을 하겠습니다.
8. 자 그러면 Main Memory를 Address 할 수 있는 주소의 길이는 총 7Bit가 되겠죠. 자 그런 구조로 되어있고요.
9. 이 Cache를 보겠습니다. Cache같은 경우에는 Tag를 저장할 수 있는 공간이 있고요. 그 다음에 실제 Data를 저장할 수 있는 공간이 있습니다. 그리고 Slot 번호가 여기 쭉 이렇게 할당이 되어 있죠. 이 구조를 보실 것 같으면 이제 여기도 역시 마찬가지로 4Byte가 저장이 되어야 되기 때문에 총 이제 32Bit 폭은 32Bit가 되겠고요. 높이는 세로는 총 8개의 Slot공간을 갖게 되죠.
10. 그래서 Cache로 Main Memory에 있는 Data를 Cache로 한번 가져 왔는지를 한번 확인을 해보면, 00001에 memo라는 단어가 있는데요, 이게 이미 일로 옮겨져 있는 상태입니다. 이렇게. 그 다음에 00110은 어딥니까 여기네요. 여기엔 copy라는 단어가 일로 이미 저장이 있는 상태죠. 이 상태입니다.
11. 자 이걸 잘 보시면요, 결국은 memo라는 Data하고 copy라는 Data가 그 전에 급해 갖고 뭐 아무래도 CPU에서 사용될 것 같아서 이미 Cache로 올라온 상태인 거죠. 이 상태로 되있습니다. 자 그 다음에 이제 CPU에서 어떤 동작을 취하냐면요. '어, 나는 CPU는 00001 00의 Data를 갖고 오고 싶어.' 라고 Main Memory에 보냅니다. 주소를.
12. 그럼 Main Memory에 보내기 전에 뭐부터 검사한다고 그랬죠? Cache부터 검사해야 되죠. Cache부터 검사하기 위해서 일단 뭘 봐야 됩니까? 화장실에 들어간 학생이 걸어 논 Tag를 봐야 되겠죠. 이름표를. 이름표를 볼 때 뭐만 보면 됩니까? 앞에 있는 5개만 보면 되겠죠.
13. 이 뒤에 꺼는 이제 볼 필요가 없습니다. 왜냐면 00001에 따라서 이 4개 중에 하나가 선택이 되는 것이기 때문에 이 5개만 보면 됩니다. 5개가 찾아가요. 어, 여기 1이 있네, 어, memo네 Data가. 그럼 memo를 가져와서 CPU로 전송을 합니다. 자 이건 뭐에요. Cache에서 Hit가 발생했죠. Hit. 자 문제가 없습니다. 잘 됐어요.
14. 그다음에 이제 또 CPU가 0001101이 Data를 요구를 합니다. Main Memory에 그러면 일단 이 00011 요거, 이거를 실제로는 Memory의 주소지만 00011은 Tag가 되겠죠. 이 Tag가 실제 Cache에 있는지 찾아봅니다. 00011은 없네요. 없죠. 없으면 어떻게 해야되죠? 여기서 다시 Main Memory로 가야 됩니다. 가서

찾아야 됩니다. 근데 찾으려고 보니까 00011에요. 00011이 어딨을까요? 여기 있네요.

15. 이게 very라는 단어에요. very 찾아갑니다. 찾아가서 일단 Miss가 됐잖아요. 이거 한번 생각해봐요. 지역성에 의해서 Miss가 되었다라는 의미는 뭐냐면 향후 빠른 시일 내에 CPU가 다시 그 Data를 사용 할 가능성이 높다는 의미에요. 그렇죠? 지금 현재 CPU가 사용 된 Data는 가까운 미래에 사용될 가능성이 높습니다.
16. 그럼 그 Data를 가져 와야 되요. 그래서 Cache에 저장을 해야 된단 말이죠. 그러니까 Miss가 나면 Main Memory로 가서 very를 CPU로 이동을 시켜요. 이동을 시키고 동시에 어떡하냐. 여기다 저장을 해야 됩니다. 저장. Cache에 다시 저장을 해야겠죠.
17. 그래서 very를 저장하고 그 다음에 뭘 해야 된다고 했죠? 칸막이 화장실을 사용하면 자기 번호를 Tag를 걸어놔야 된다고 했죠. 걸어놔야 됩니다. 뭘 걸어 놓냐면 Tag가 00011입니다. 그래서 00011을 앞에 이렇게 걸어놓게 됩니다. 그죠. 그럼 앞으로도 이 very는 사용될 가능성이 매우 높죠. 이렇게 됐습니다. 그럼 이건 Miss가 난 이후에 Cache에다가 적재를 한 경우가 되죠. Miss 후 적재
18. 자 다음. 다시 CPU가 0011010 이라는 그런 Data를 요구했습니다. 0011010을 여기서 찾습니다. 다시. 이제 이게 들어가 있는 상태로 찾아야 되는데요. 찾아요. 0011010 은 있네요. copy라고 있네요. Hit죠. 그럼 여기에 있는 copy를 CPU로 데려가고 끝이죠, 끝. Hit가 났기 때문에.
19. 다음. 0111100 자 이걸 Main Memory에 요구를 합니다. 그럼 먼저 Cache가서 찾아요. 없죠. 그럼 어떻게 해야 되요? Cache로 가서 찾았는데 없으니까 Main Memory로 가야 되죠. 갔는데 0111100이니까요. 여기네요. digi입니다. digi를 여기에다 갖고 와서 저장을 하게 되죠. 이것도 이제 Locality에 의해서 지금 현재 Miss가 났다라고 하더라도 CPU가 요구한 Data기 때문에 향후 가까운 미래에 다시 CPU가 사용할 가능성이 높다 해서 가져와서 저장을 하는거죠. 이렇게. 저장을 했습니다.
20. 이제 마지막으로 1111011이렇게 됩니다. 1111011찾아보겠습니다. 1111011이 어디 갔나요. 여깄네요. 일단 여기서 찾아야죠. 1111011없으니까 Miss가 났습니다. Miss가 났으니까 Main Memory로 가야죠. wire가 되겠네요. wire를 다시 여기 Cache로 가져와서 저장을 하고 걔의 이름표 Tag를 앞에다 딱 걸어놔야겠죠.
21. 이렇게 이제 사용하는 방식이 Fully Associative Mapping입니다. 그러니까 8개의 빈 공간에 급한 놈 와서 저장이 되는 겁니다. 사용하고 자기 이름표를 걸어놓고. 그리고 만약에 내가 Cache에 Data를 찾으려면 이 7개의 Bit Main Memory 주소 중에 앞의 5개를 딱 떼서 비교를 하는 겁니다. 있는지 없는지. 그런 식으로 하는 게 Fully Associative Mapping이고요.

22. 제가 별도로 설명을 않드리는 이유는 이 뒤에 있는거 00 이냐 01이냐 10이냐 뭐 이런 거에 따라서 여기가 Hit가 갔으니까 10을 따져보면 10이면 00110 이거죠. 이거 중에 10이면 세 번째 겁니다. 그러니까 copy 중에 p죠 이걸 가져가겠다는 얘기죠. 그러니까 이 뒤에 있는 2Bit는 크게 의미는 없어요. 이 2Bit는 이 4개중에 몇 번째 꺼인가를 선택하는 것 이외에는 크게 의미는 없습니다. 자 이런 방식으로 Fully Associative Mapping이 동작하게 됩니다.

23. 자. 여기 보시면 Main Memory의 Block이 Cache의 어떤 Slot으로든 적재를 할 수가 있다 적재 가능하다 그랬죠. 급한 놈 먼저 쓰면 되니까.

24. 그 다음에 Tag Field. Tag Field는 이 앞의 5개 그러니까 전체 7개 Main Memory 주소 폭이 7Bit인데 그 중에 5개 앞의 5개를 여기에 있는 Main Memory Block 번호로 활용을 하면 되는 거죠. 이거 하나가 Block이라고 그랬으니까, 이 Block 하나는 이 5개 갖고 지정을 할 수가 있고요. 이 뒤에 있는 2개 갖고는 이 Block 내부의 몇 번째에 있는 놈인지를 결정할 수 있다고 했으니까 Main Memory의 Block 번호가 그대로 Tag Field로 활용이 됩니다. 근데 이거는 사실 생각해보면 Tag Field의 5개의 Bit가 저장을 해야 되기 때문에 사실은 다소 비효율적이라고 보는 게 맞겠죠.

25. 자 그럼 이제 Fully Associative Mapping Cache의 하드웨어 조직을 Read하고 Write 하기 위한 하드웨어 조직을 한번 살펴보겠습니다. 자 여기 보시면 이제 Main Memory Address가 여기 나와 있습니다. 지금 앞의 예에서는 Tag Field 이게 5Bit였고요. Word가 2Bit였습니다.

26. 자 Main Memory CPU에서 Main Memory로 어떤 Data로 요구하기 위해서 7개의 Bit를 Main Memory로 전송을 하는데 전송하기 전에 Cache를 먼저 거쳐야 되겠죠. 그래서 그걸 여기 있는 것처럼 5개 2개로 분리되는 이 Path를 전송을 합니다. 그럼 5개가 내려오겠죠. 내려오면 이게 이제 각각 Cache의 Slot을 의미하는데요. 이게 1Byte씩 이에요. 이게. 1Byte, 1Byte, 1Byte 이렇게 되죠. 4Byte죠. 그리고 여기에 Tag가 저장이 되있습니다. Tag에는 5Bit짜리가 저장이 되있겠죠. 그걸 일로 다 가져옵니다. Tag를.

27. 모조리 다 병렬로 쭉 가져오죠. 가져와서 비교기를 통과시켜서 비교를 하게 됩니다. 같은 것을 찾는거죠. 이 n개 중에 이 같은 게 있을 수도 있고 없을 수도 있습니다. 만약에 같은 게 있었다라고 하면 Hit가 났다고 봐야겠죠. 만약에 또 같은 게 없다고 하면 Miss가 됩니다. Miss가 되니까 Miss가 됐을 때는 이렇게 해가지고 이 Buffer. Buffer인데요. Main Memory의 Address 전체를 이 Buffer에 잠시 저장을 했다가 Miss가 날 것 같은 경우에는 그냥 Main Memory에 보내면 됩니다. 그래서 Main Memory에서 Data를 찾아가면 그만이죠.

28. 근데 만약에 Hit가 났다. Hit가 났다는 얘기는 이 비교기를 통해서 비교를 했는데 똑같은 게 있다라는 얘기죠. 그럴 경우에는 그 똑같은 게 발생한 그 Slot. Slot에

해당하는 Word를 찾아가게 되죠. 왜냐하면 이 4개 중에 여기에 이 4Byte가 저장이 되있는데 이 4개 중에 뭔지를 알아야 되니까

29. 그래서 이 2개의 Word를 이용해서 이게 만약에 00이었다라고 하면 이거를 Main Memory로 CPU로 가져가면 되고 이게 01이었다, 그러면 이걸 가져가면 되고 10이면 이거 11이면 이거를 CPU로 가져다 놓으면 되기 때문에 여기서 이제 이 Buffer를 열어주게 됩니다. 그러면 쭉 가서 4개 중에 하나를 골라서 CPU로 전송하게 되는 그런 구조가 되겠습니다.
30. 여기서 이제 Bit 수는 t 라고 되있는게 5Bit가 되겠고요 w로 되있는 게 2Bit가 되겠죠. 장점을 한번 보겠습니다. 새로운 Block, 그러니까 Main Memory의 새로운 Block이 Cache로 적재될 때 Slot의 아무 Block이나 아무 Slot이나 적재가 될 수가 있기 때문에 선택이 매우 자유로운 장점이 있죠.
31. 두 번째 Locality가 매우 높다 뭐 지금 이건 사용하지 않은 지금 당장 사용을 CPU가 Main Memory에 어떤 Data를 요구 했는데 만약에 그게 Main Memory에 없었다 그러면 Cache에 바로 바로 적재가 되기 때문에 Locality가 높다면 Hit Ratio가 매우 높아지는 그런 장점을 가질 수가 있겠죠.
32. 자 근데 단점이 하나 있는데요. 이거의 가장 큰 단점은 바로 이겁니다. 여기 보시면 비교기를 통과할때요. 비교기를 통과할 때 만약의 Cache의 Slot의 개수가 32개였다라고 한다면 32개의 Path가 다 존재해야 됩니다. 다 모조리. 그래서 여기 나온 Tag와 비교를 해야 됩니다. 그럼 이 비교기의 복잡도가 엄청나게 증가하게 되겠죠.
33. **하드웨어적으로 상당히 그게 부담이 되거든요. 그래서 Cache Slot들이 Slot들의 Tag들을 병렬로 검사하기 위하여 매우 복잡하고 비용이 높은 회로가 필요로 하게 된다는 그런 단점이 존재하게 됩니다.**
34. 자 두 번째 방식은 이제 Direct Mapping입니다. Direct Mapping은 우리 과거에 어떤 식으로 Direct Mapping을 했는지 한 번 고민을 해볼게요. 어떻게 했었냐면요. 8개의 그룹으로 나눴어요. 32명을 8개의 그룹으로 나눠서 각 그룹당 4명씩 할당을 하기로 했어요. 그죠. 그거 기억나죠. 그런 다음에 지금 사용할 수 있는 칸막이 화장실 개수는 총 8개가 있는데요. 그 8개 중에 어떤 하나의 칸막이를 그 4명이 공유를 하게 하는 그런 형태라고 했습니다.
35. 지금 이 그림을 보시면 어떻게 되있는지 한번 보실게요. 이게 칸막이 화장실이에요. 총 8개가 있습니다. 그리고 여기 총 32명의 학생이 있습니다. 즉 여기 빨간색으로 표시되어 있는 Data들, 학생들은 첫 번째 Cache Slot, 첫 번째 칸막이 화장실만 사용할 수 있습니다. 그죠. 내가 만약에 Miss가 나서 적재를 Cache Slot을 해야 된다고 하면 반드시 여기에 적재가 되어야 됩니다. 반드시.

36. 두 번째 노란색을 볼까요. 노란색은 아니고 좀 황색정도 되겠네요. 황색을 보시면 이 첫 번째 숫자로는 1이고요. 번째로는 2번째죠. 이 Slot. 이 Slot에는 여기있는 것처럼 memo, role, para, evol 이 4개 중에 하나만 이거를 사용할 수가 있습니다. 그죠. 4개 끼리의 경쟁이죠. 여기 전체적으로 보시면 어떻게 구분을 한거냐면요. 8개 단위로 하나씩 띄어가면서 그룹을 만들었어요. 그죠.

37. 그렇게 만든 이유는 이 Slot번호를 맞추기 위해서에요. 즉 Main Memory의 Address가 총 7개 Bit가 있는데 이런 식으로 나오게 되면 빨간색을 기준으로 할게요. 빨간색은 여기가 다 000입니다. 여기도 000 000, 000 여기도 000 그럼 이 위에 있는 Cache 번호가 000이죠. 그러니까 이걸 중간에 Main Memory Address가 000인 것은 Slot번호 000을 공유하게 되는거죠.

38. 다른 거 한번 볼까요. 보라색을 보겠습니다. 보라색은 111이거든요. 111, 111, 111, 여기 111 이거는 여기 111을 공유하게 되요. 그렇게 되면 CacheSlot이 몇 번째냐만 확인하면 Main Memory 주소 중에 중간의 3Bit를 알아낼 수 있는 거죠. 그죠.

39. 그렇게 활용하기 위해서 이렇게 띄어가면서 띄어가면서 Data를 할당하게 되는거죠. 근데 이게 그러면 연속적으로 예를 들어서 이런거죠. 여기에 들어가는 이거 4개. 이거 4개를 쓰게 하면 안되냐. 그렇게 할 수 없다는 거에요. 그렇게 해버리면 여기 주소관계가 약간 얽히기 때문에 간단하게 우리가 Slot 번호를 알아내기가 힘들죠. 그러기 위해서 이렇게 띄어가면서 할당을 한거고요.

40. 그 다음에 또 이제 빨간색을 한번 볼까요. Slot번호 000인거. 그거는 00 여기 보면은 01 여기보면은 10, 여기보면은 이제 11 이렇게 되있습니다. 그러니까 이 4놈 중에 한 놈을 구분하는 방법으로 이 Tag를 사용하게 됩니다. 00 01 10 11 그래서 여기에 뭐가 들어가느냐 할 때 들어갈 때 Data 놓고요. 거기다가 Tag를 앞에 딱 써놓습니다.

41. 그러면 첫 번째 Slot에 00이 들어갔다 하면 comp죠. 당연히. 그죠. 이것도 그래요. 두 번째 Slot 001의 10의 Tag가 들어갔다 그러면 Main Memory에서 10001을 찾아야 됩니다. 10001이 여기네요. 여기. para죠. 그렇게 저장이 되는거에요. 이제 그걸 위해서 이런 식으로 이제 건너 띄면서 이렇게 Data들을 Data 주소들을 할당하게 되는거죠.

42. 자 이것도 한번 예제를 갖고 얘기를 해볼게요. 먼저 comp하고 para가 Cache에 저장이 먼저 되있었어요. 그래서 CPU가 Main Memory의 Data를 요구하게 됩니다. 뭘 요구했냐면 00001을 요구했습니다. 00001 얘는 어떻게 하면 되냐면요. 먼저 001을 확인해야됩니다. 001의 Slot에 갑니다. 갔어요. 간 다음에 Tag가 00입니다. 00 아이고, 없네. 10이 들어가있네. Miss죠. Miss가 됐으니까 적재를 해야됩니다.

43. 왜, Miss가 됐다는 것은 앞으로 사용될 가능성이 높기 때문에 그럼 적재를 하기 위해서 이제는 Slot을 찾는 게 아니라 Main Memory를 찾아 가야 되겠죠. 그래서 00001을 찾습니다. 00001이 어디냐면요. 여기네요. 이겁니다. 그럼 memo에 해당하는 주소가 됩니다. 그래서 memo를 여기로 가져옵니다.

44. 갖고 오는데 이 memo도 역시 결정적으로 para와 동일한 이 Slot번호 001에 들어갈 수밖에 없기 때문에 이 파라를 지워 버려야되요. 어쩔수없이. 지운 다음에 memo라고 적고 여기 앞에 있는 10을 지우고 여기에 있는 00을 적어놔야 됩니다. 그렇게 하면 이렇게 바뀌게 되죠. 00의 memo 이런식으로 바뀌어야 되고요.

45. 그 다음에 CPU가 다시 Data를 요구합니다. Main Memory에. 00000을 요구합니다. 자 그러면 000번째의 Slot을 찾아가서요, 찾아갔습니다. 그 앞의 Tag가 00인지 확인합니다. 00 어 있네요. comp있네요. 그러면 CPU로 가져가게 되죠. 끝. Hit가 됐습니다.

46. 다음 00110을 확인해야 됩니다. 그럼 110 Slot을 갑니다. 여기있네요. 여긴데 거기 안의 00이 있나 확인합니다. 00. 없죠. 아무것도 없는데. 없으니까 이거는 Miss가 난거죠. 그럼 Main Memory에 가서 CPU로 Data를 옮기면서 동시에 Cache에 적재를 해야 놔야 되겠죠. 그래서 적재를 하기 위해서 00110을 찾아갑니다. 그래서 어딨냐면요 00110이. 여기네요. copy네요. copy를 이 자리로 가져와야 되겠죠.

47. 110이니까 이 자리. 이 자리로 가져와야 되겠죠. 갖고 오고 copy에 해당하는 Tag가 00이니까, 00을 이 앞에다 이렇게 붙여주면 그게 바로 이 결과입니다. 자 이런 방식으로 진행을 하는겁니다. 그래서 마지막 꺼는 비슷하니까 하지 않겠는데요.

48. 이런 방식으로 어느 Slot에 들어가냐에 따라서 자기가 Main Memory의 주소가 뭔지 그걸 확인할 수가 있고요. 자기가 몇 번째 Slot으로 들어가면 Data 중에 몇 번째 인지를 확인해서 여기에 있는 Tag로 활용을 하면 효율적인 Mapping을 구성을 할 수가 있습니다.

49. 자 Main Memory의 Block이 Cache의 정해진 Slot으로만 적재할 수 있다. 지금 여기 보시면 000번지는 다 중간에 있는 주소가 000이 인것만 이 000의 Slot을 점유할 수가 있게 되는거죠.

50. 그 다음에 Tag Field하고 SlotField하고 더했다. 일단 Tag Field는 여기서 보면 뭐냐면 이거죠. 이 Tag Field하고 이 Slot Field 이 두 개 이 두 개를 더한 5개의 Bit를 얘기합니다. 그게 바로 Main Memory의 Block번호가 되버린다. 그러니까 쉽게 우리가 Main Memory Block과 Cache의 Slot을 찾아갈 수가 있게 되는 것입니다.

51. 자 역시 비슷한 방식으로 Direct Mapping의 하드웨어 조직을 살펴 볼게요. 먼저 이제 Main Memory Address가 있습니다. 이게 CPU에서 Main Memory로 여

기에 있는 Data를 가져와라라고 요구를 하게 되는데요. Main Memory를 주소가 보내지기 전에 말씀드린 것처럼 Cache로 먼저 보내져서 Data가 있는지 Hit가 날 수 있는지 확인을 해야 되겠죠.

52. 그럼 먼저 어떡하냐면요. 이렇게 합니다. Slot 이 Slot 3개 Bit를 먼저 보냅니다. 지금 앞에선 이게 2고 이건 3이고 이개 2개였습니다. 그래서 이 Slot을 가서 찾아갑니다. 찾아가면 여기 있는 Tag가 있겠죠. Tag를 갖고 와서 비교를 합니다. 뭐랑 비교하죠. 지금 Main Memory Address의 Tag랑 비교하죠.

53. 그럼 아까와는 달라 아까는 병렬로 n개를 다 비교를 했어야 되는데 여기선 선택된 하나만 비교를 하게 되죠. 그러면 이 비교기의 복잡도가 굉장히 낮아지죠. 그러니까 심플하게 구성할 수 있고요, 이게 바로 Miss가 났다. 그럼 아까와 똑같은 방식으로 Main Memory로 주소 전송 해서 Data를 찾아가면 그만입니다. 그런데 만약에 Hit가 났다 라고 하면 이쪽으로 와서 원하는 이게 선택된 게 Slot이 여기니까 여기서 이 Word를 통해서 4개 중에 어느거냐를 골라 갖고 CPU로 전송해 주면 Hit가 나게 되는거죠. 이런 식으로 하트웨어를 구성할 수 있습니다.

54. 장점은 일단 첫 번째 하드웨어가 간단하다. 그래서 구현비용이 적게 든다라는 장점이 있고요. 두 번째 각 Main Memory Block이 적재될 수 있는 Cache Slot이 한 개뿐이 없다라는 거에요. 즉 아까도 말씀드렸지만 사용할 수 있는 칸막이 화장실이 하나 밖에 없기 때문에 4명이 반드시 그 화장실을 사용해야 되요. 근데 한 그룹의 4명 중에 급한 화장실이 급한 사람이 2명이었다. 그러면 큰일나는 거에요. 겹쳐질 수밖에 없어요. 그런 문제가 발생하개 되는거죠.

55. 그래서 적재될 수 있는 Cache Slot이 한 개 뿐이기 때문에 그 Slot을 Share하는 다른 Block이 적재되는 경우에는 Swap-out된다 라는 얘기는 뭐냐면 지워져 버린다는 얘기에요. 아까 지워지는 예를 우리가 위에서 보기도 했는데요. 먼저 그 4명 중에 어떤 한 놈이 화장실을 사용하고 있었는데 내가 지금 급해 죽겠어, 그럼 그 화장실 사용해야 되거든요. 그럼 먼저 있던 놈 끌어낼 수밖에 없단 얘기에요. 끌어내고 내가 사용해야 된다. 그런 경우가 Swap-out이라는 표현을 쓰는데요. 이런 문제가 발생하기 때문에 이런게 이제 그 Direct Mapping Cache의 단점이라면 단점이 되겠습니다.

56. 네 그 다음이 이제 Set Associative Mapping입니다. Set Associative Mapping은 아까 말씀드릴 때 어떻게 말씀을 드렸냐면 이제 4개의 그룹으로 나누는 거죠. 근데 그 4개의 그룹에는 8명이 들어가 있는 거에요. 지금 여기 보시면 빨간색으로 표시되있는 게 하나의 그룹입니다. 그래서 그 그룹은 어떤 걸로 구분이 되냐면 여기에 있는 이거 00 이 빨간색도 00이고요. 이것도 00 다 00 이죠. 총 8개가 있어요.

57. 그러니까 여기가 00으로 표시되 있으면 여기 Set 번호라고 되있는 이 00 이게 Slot이 2개가 있죠. 1개의 Slot 2개의 Slot 두 개 있죠. 이 두 개 중에 하나를 사

용할 수 밖에 없다고요. 아까 제가 말씀드린 것처럼 구분된 나랑 같이 화장실을 공유하는 어떤 놈이 있었는데 그놈이 가서 볼 일을 보고 있어요. 내가 급해. 그럼 그놈 끌어내지 않아도 되요. 이제는 그놈은 분명히 이걸 사용하고 있었을 테니까 난 이거 사용하면 그만입니다. 그죠. 이제 이런 식으로 두 개가 혼합된 형태를 앞서 말씀드린 두 가지 경우가 혼합된 이런 형태를 구현해 내는 거죠.

58. 그래서 역시 마찬가지로 이 화장실을 사용할 때는 Tag 자기 자신의 번호를 고지해야 되는데요. 여기 보시면 00이라고 되있는 걸 하나하나 따져보면 Tag가 000부터 시작해서 001 010 011 또 여기 보면은 101 그다음에 또 여기 00있네요 110 그 다음에 이거, 이렇게 되죠. 그러면 이게 순서대로 000 001 010 011 여기 중간에 100이 빠졌네요. 100 101 110 111 이렇게 되는 거잖아요. 그죠. 이게 자기 자신의 번호죠. 이 앞에 있는게. 자기 자신의 번호를 이 Tag Field에 저장을 해놓고 볼 일을 봐야되죠. 그렇게 구조가 되는 겁니다.

59. 지금 보시는 것처럼 일단 두 개의 Data가 들어와 있고요. 이것도 예제를 갖고 판단을 해보겠습니다. 먼저 000 01 00 이 Data를 CPU가 Main Memory한테 가져와라. 라고 요구를 했습니다.

60. 그럼 먼저 봐야되는 게 01을 봐야됩니다. 01 아 01 Set으로 가보겠습니다. 여기 있고요. 그 다음에 000이 있는지를 확인해 보겠습니다. 어 있네요. Hit네요. 그러면 memo를 CPU로 가져가면 그만입니다.

61. 다음 11을 보겠습니다. 11을 보죠. 아 여기고요. 그 다음에 앞에가 000입니다. 확인해 보겠습니다. 000 아무것도 없네요. 그럼 Miss네요. 그럼 어떻게 해야되요? Main Memory에 가서 Data를 CPU로 갖고 와야 되죠. 동시에 그 갖고 간 Data를 Cache에 저장을 해놔야 됩니다. 자 어떻게 저장하는지 보겠습니다.

62. 11, 00011이니까요. 00011을 Main Memory에서 봐야 되겠죠. 그러면 요겁니다. very very를 해야 되는데요. 여기 지금 11이니까. Set가. 지금 여기에 저장을 해야 됩니다. very 저장하고 여기다가는 뭘 저장하냐면 very 앞에 있는 Tag번호 000 자기이름표를 고지해야되죠. 000 이렇게 해야 됩니다. 그래서 이렇게 저장이 되는 겁니다. 이렇게. 그 다음에 001 10 이걸 보겠습니다. 10을 보면 여기죠. 001 있죠. Hit네요. 문제 없네요.

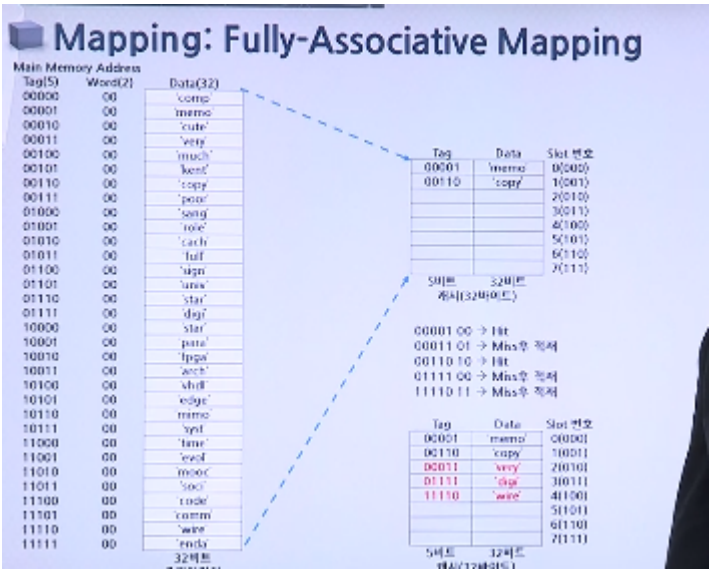
63. 다음 밑에꺼 보겠습니다. 011 11 11을 보겠습니다. 011 없네요. 그럼 이거 적재해야 되겠네요. 기존에는 이거 지워버리고 적재를 했어야 되는데 그게 아니라 이 뒤에 공간이 하나 더 있기 때문에 여기다가 갖다 놓는겁니다. 어 011 11은 어딨나요. 011 11은 여겁니다 이거. digi입니다. digi를 very 뒤에 이렇게 갖다 놓고 011을 적게되죠. 자 이런 식으로 하게 되면 효율적으로 기존에 있는 걸 Swap-out을 하지 않고 추가적으로 자기 Data까지 올려놓을 수 있게 되죠. 이런 방식을 Set Associative Mapping이 되겠습니다.

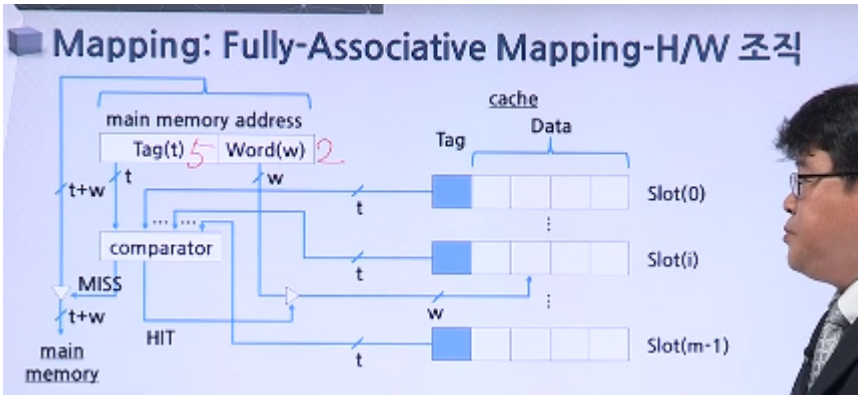
64. Set Associative Mapping Cache는 Direct Mapping하고 Fully Associative Mapping을 혼합한 형태고요. 그 다음에 Main Memory Block이 Cache의 정해진 Slot, 정해진 Slot으로만 적재를 할 수가 있습니다. 마찬가지로. 정해진 Set과 Slot으로 적재를 할 수가 있고요. 그 다음에 주소는 Tag Field하고 Slot Field 이 두 개를 이용해서 Main 메모리 Block 번호를 특정할 수 있게 되는 그런 방식이 되겠습니다.

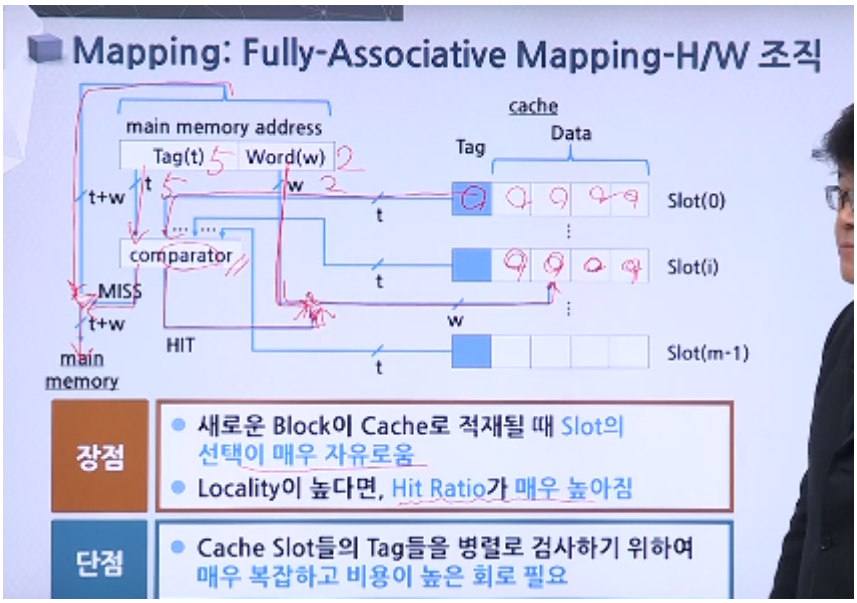
65. 마지막으로 이제 이 Set Associative Mapping을 하드웨어를 얘기할텐데 그 전에 이게 그럼 왜 Associative냐, Associative의 의미가 뭐냐. 그거는 뭐냐면 Tag끼리 비교하면 됩니다. Tag끼리 비교해서 연관성 있는 걸 찾아내겠다. 그걸 이제 Associative라는 말을 쓰게 되는거죠. 그런데 이거 같은경우에는 Set의 형태로 된 상태에서 연관성이 있는 걸 찾아가겠다 그런 의미죠. 그렇게 됩니다.

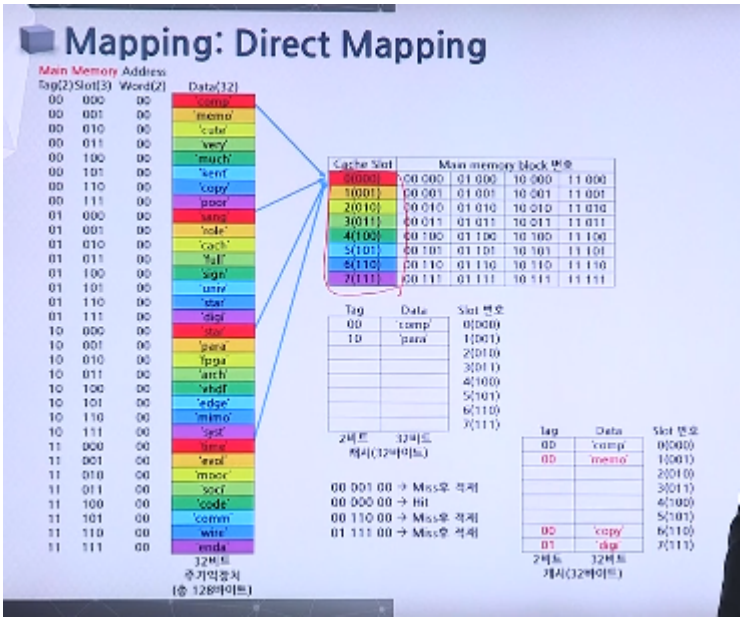
66. 하드웨어를 보겠습니다. 하드웨어를 보시면 먼저 Main Memory Address가 쭉 내려오고요. 먼저 Set을 찾아 가겠죠. 찾아갑니다. 근데 Set 아래에는 Slot이 두 개가 있습니다. 두 개가 있죠, 보시는 것처럼. 그러면 이 두 개의 Tag를 다 갖고 오죠. 여기도 갖고 오고요. 그 다음에 여기도 이렇게 갖고 옵니다. 그런 다음에 기존에 내려오는 Tag와 비교를 합니다. 같은 게 존재하나요? 존재하면 Hit. Hit이면 둘 중에 하나 선택을 하겠죠. 선택한 것을 CPU로 보내면 되죠.

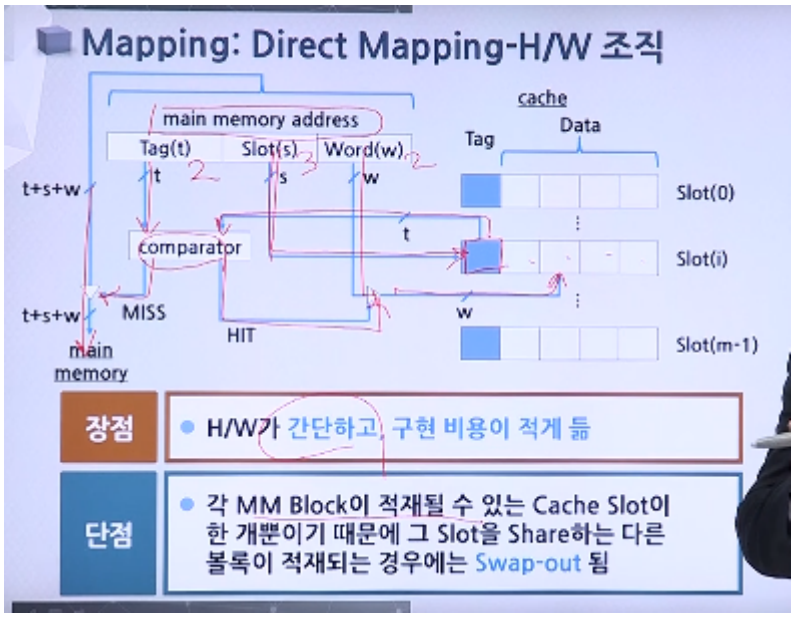
67. 이런 방식을 구현을 하게 되고 이 마찬가지로 Comparator 비교해서 같은 게 없으면 Miss죠. 그럼 Main Memory로 주소를 다시 보내야겠죠. 이런 식으로 구현을 하면 이제 Set Associative Mapping의 하드웨어 조직이 구현이 되고 이게 장점은 이제 Fully AssociativeMapping과 Direct Mapping방식의 절충 방식이다 라는 게 하나의 장점으로 볼 수가 있을 것 같습니다.

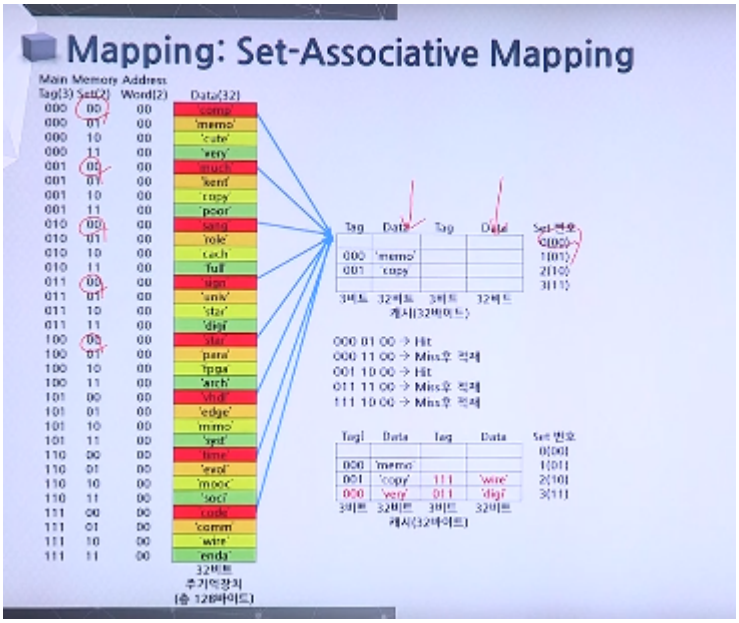
68. 


Mapping: Fully-Associative Mapping-H/W 조직
main memory address
Tag(t) 5
Word(w) 2
t+w
t
w
cache
Tag
Data
Slot(0)
Slot(i)
Slot(m-1)
comparator
MISS
t+w
main memory
HIT


Mapping: Fully-Associative Mapping-H/W 조직
main memory address
Tag(t) 5
Word(w) 2
t+w
cache
Tag
Data
Slot(0)
Slot(i)
Slot(m-1)
comparator
MISS
main memory
HIT
장점
새로운 Block이 Cache로 적재될 때 Slot의 선택이 매우 자유로움
Locality이 높다면, Hit Ratio가 매우 높아짐
단점
Cache Slot들의 Tag들을 병렬로 검사하기 위하여 매우 복잡하고 비용이 높은 회로 필요


Mapping: Direct Mapping
Main Memory Address
Tag(2) Slot(3) Word(2)
Data(32)
Cache Slot
Main memory block 번호
Tag
Data
Slot 번호
00 001 00 → Miss후 적재
00 000 00 → Hit
00 110 00 → Miss후 적재
01 111 00 → Miss후 적재


Mapping: Direct Mapping-H/W 조직
main memory address
Tag(t)
Slot(s)
Word(w)
cache
Tag
Data
t+s+w
comparator
MISS
HIT
main memory
Slot(0)
Slot(i)
Slot(m-1)
장점
H/W가 간단하고, 구현 비용이 적게 듦
단점
각 MM Block이 적재될 수 있는 Cache Slot이 한 개뿐이기 때문에 그 Slot을 Share하는 다른 블록이 적재되는 경우에는 Swap-out 됨


Mapping: Set-Associative Mapping
Main Memory Address
Tag(3) Set(2) Word(2)
Data(32)
32비트
주기억장치
(총 128바이트)
Tag
Data
Tag
Data
Set 번호
3비트
32비트
3비트
32비트
캐시(32바이트)
000 01 00 → Hit
000 11 00 → Miss후 적재
001 10 00 → Hit
011 11 00 → Miss후 적재
111 10 00 → Miss후 적재
Tag
Data
Tag
Data
Set 번호
3비트
32비트
3비트
32비트
캐시(32바이트)