# 프로젝트 계획서
# /결과 보고서

**※ 주　제 : 최적의 캐시메모리 설계 및 분석**

**※ 기　간 : 2015/11/25 ~ 2015/12/14**

**※ 팀　원 : 노형래 2014152009 캐시메모리 설계, 분석 및 보고서 작성**

**박재원 2014152048 캐시메모리 설계, 분석 및 PPT 작성**

# 목 차

# 1. 개요 및 목표

컴퓨터 시스템에서 가장 빠른 CPU와 CPU에 비해 느린 주기억 장치 사이의 데이터 교환 시 병목현상을 피할 수 없다. 하지만 주기억 장치에 비해 속도가 빠른 캐시메모리를 CPU와 주기억 장치 사이에 배치해 계층적 메모리를 구성하게 되면 이러한 병목 현상을 완화할 수 있다. 본 프로젝트에서는 이러한 캐시의 성능을 임의의 조건에서 분석하며 분석 결과를 기반으로 최적의 성능을 보이는 캐시를 제안할 것이다.

# 2. 설계

(1) 캐시 주소비트 구성

캐시 주소비트의 구성은 매핑 방식에 따라 달라지게 되는데 주어진 조건에서 각각의 매핑 방식에 따른 캐시 주소를 파악한다.

- 메모리 사이즈 : 4MB
- 캐시메모리 사이즈 : 64KB
- 블록 사이즈 : 8B
- 매핑방식 : Directing Mapping, Set-Associative
- Set Size : 8 blocks

(2) 캐시 적중률 분석

캐시의 적중률을 좌우하는 요소로 Block Replacement, 캐시 사이즈, Set 정책이 있으며 각각의 요소가 캐시 성능에 미치는 영향을 분석한다.

(3) 캐시 평균 접근시간 분석

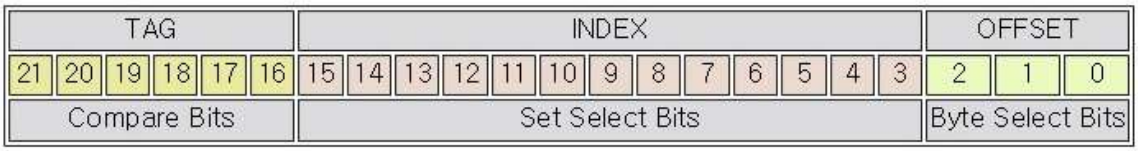
캐시의 접근시간을 좌우하는 요소로 캐시 사이즈, Associativity Sets, Block size, Write policy가 있으며 본 프로젝트에서는 특정 조건에서 64K의 캐시와 256K의 캐시의 Associativity Sets, Block size, Write policy가 캐시 성능에 미치는 영향을 분석한다.

# 3. 구현 및 테스트

(1) 캐시 주소비트 구성

주어진 조건에서 캐시의 매핑방식에 따른 주소비트는 다음과 같다.

- 메모리 사이즈 : 4MB = $2^{22}$
- 캐시메모리 사이즈 : 64KB = $2^{16}$
- 블록 사이즈 : 8B = $2^{3}$
- 매핑방식 : Directing Mapping
- Set Size : 8 blocks

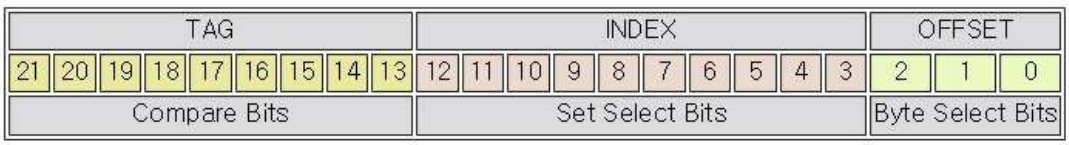



TAG = $2^{22-13-3}=2^{6}$

Index = $2^{16-3}$ = $2^{13}$

OFFSET = $2^{3}$

- 메모리 사이즈 : 4MB = $2^{22}$
- 캐시메모리 사이즈 : 64KB = $2^{16}$
- 블록 사이즈 : 8B = $2^{3}$
- 매핑방식 : Set-Associative
- Set Size : 8 blocks




TAG = $2^{22-10-3}$ = $2^{9}$

Index = $2^{16}/(2^{3}*2^{3})=2^{10}$

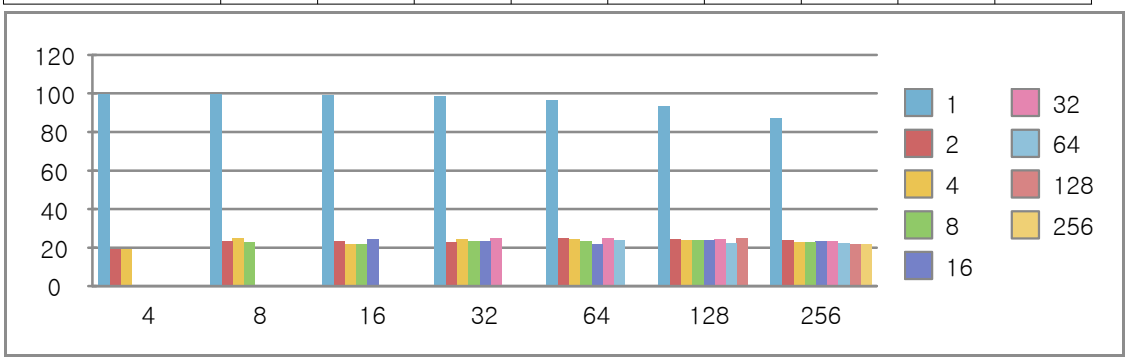
OFFSET = $2^{3}$

(2) 캐시 적중률 분석

캐시에 무작위의 데이터를 넣었을 때 Block Replacement, 캐시 사이즈, Set 정책에 따른 캐시의 적중률은 다음과 같다.

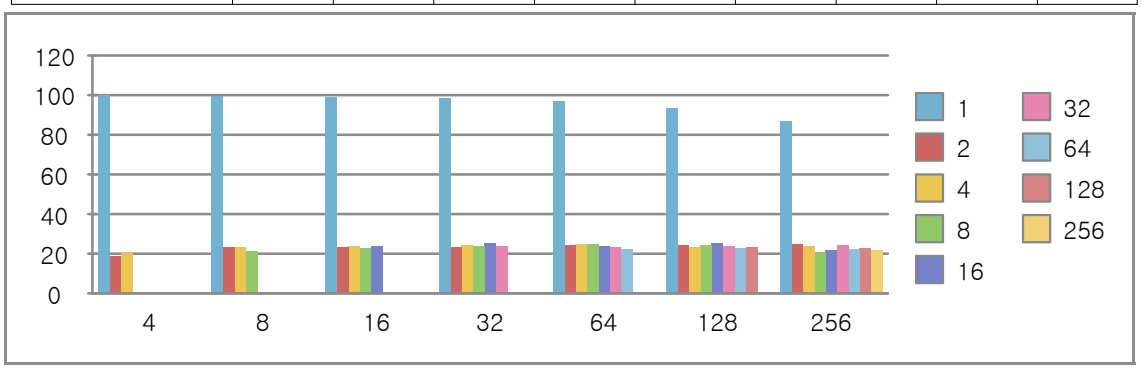
Replacement Policy : LRU

| cache size \ set | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 |
|---|---|---|---|---|---|---|---|---|---|
| 4 | 99.8 | 19.44 | 19.19 | x | x | x | x | x | x |
| 8 | 99.6 | 23.43 | 24.75 | 23.03 | x | x | x | x | x |
| 16 | 99.19 | 23.64 | 21.72 | 21.62 | 24.55 | x | x | x | x |
| 32 | 98.38 | 23.18 | 24.24 | 23.23 | 23.38 | 24.8 | x | x | x |
| 64 | 96.77 | 24.85 | 24.65 | 23.54 | 21.92 | 24.9 | 23.94 | x | x |
| 128 | 93.54 | 24.6 | 24.09 | 23.99 | 23.74 | 24.29 | 22.37 | 24.85 | x |
| 256 | 87.07 | 24.04 | 22.78 | 23.08 | 23.38 | 23.38 | 22.27 | 22.02 | 21.57 |



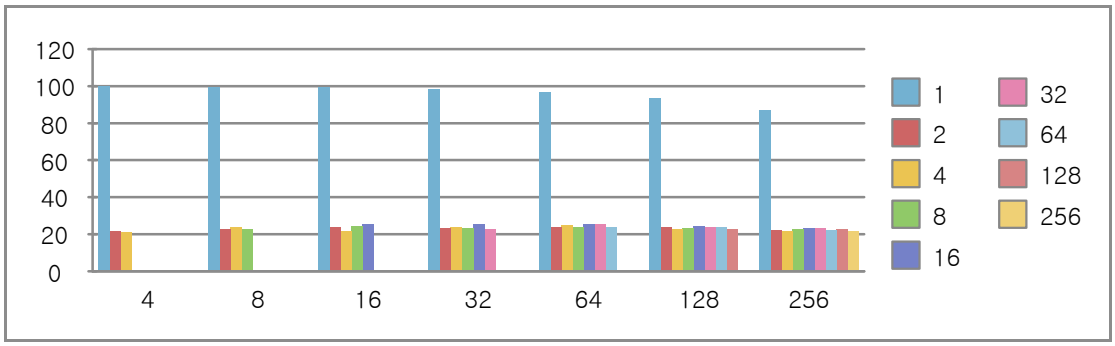

Replacement Policy : FIFO

| cache size \ set | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 |
|---|---|---|---|---|---|---|---|---|---|
| 4 | 99.8 | 18.79 | 20.4 | x | x | x | x | x | x |
| 8 | 99.6 | 23.59 | 23.13 | 21.41 | x | x | x | x | x |
| 16 | 99.19 | 23.54 | 23.69 | 22.78 | 23.69 | x | x | x | x |
| 32 | 98.38 | 23.59 | 24.49 | 24.09 | 25.56 | 24.09 | x | x | x |
| 64 | 96.77 | 24.29 | 24.8 | 24.85 | 23.94 | 23.28 | 22.47 | x | x |
| 128 | 93.54 | 24.19 | 23.13 | 24.34 | 25.45 | 24.04 | 22.68 | 23.59 | x |
| 256 | 87.07 | 25.05 | 23.99 | 20.91 | 22.02 | 24.44 | 22.37 | 22.98 | 22.02 |



Replacement Policy : Random

| cache size \ set | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 |
|---|---|---|---|---|---|---|---|---|---|
| 4 | 99.80 | 21.87 | 21.21 | x | x | x | x | x | x |
| 8 | 99.47 | 22.78 | 23.99 | 22.93 | x | x | x | x | x |
| 16 | 99.19 | 23.89 | 21.67 | 24.24 | 25.30 | x | x | x | x |
| 32 | 98.38 | 23.08 | 23.64 | 23.13 | 25.56 | 22.73 | x | x | x |
| 64 | 96.77 | 23.74 | 24.65 | 23.69 | 25.53 | 25.30 | 23.79 | x | x |
| 128 | 93.54 | 23.59 | 22.98 | 23.13 | 24.19 | 23.69 | 23.54 | 22.78 | x |
| 256 | 87.07 | 22.22 | 21.62 | 22.73 | 23.08 | 23.43 | 22.22 | 22.68 | 21.87 |




분석 결과 Replacement Policy에 따른 성능 변화는 매우 미약하나 Set이 1일 때(완전 연관 매핑) 가장 높은 캐시 적중률을 보인다. 또한 캐시 사이즈가 커질수록 적중률이 감소하는 모습을 확인할 수 있으며 Set >= 2일 때, 성능차이가 거의 없음을 확인할 수 있다.

(3) 캐시 평균 접근시간 분석

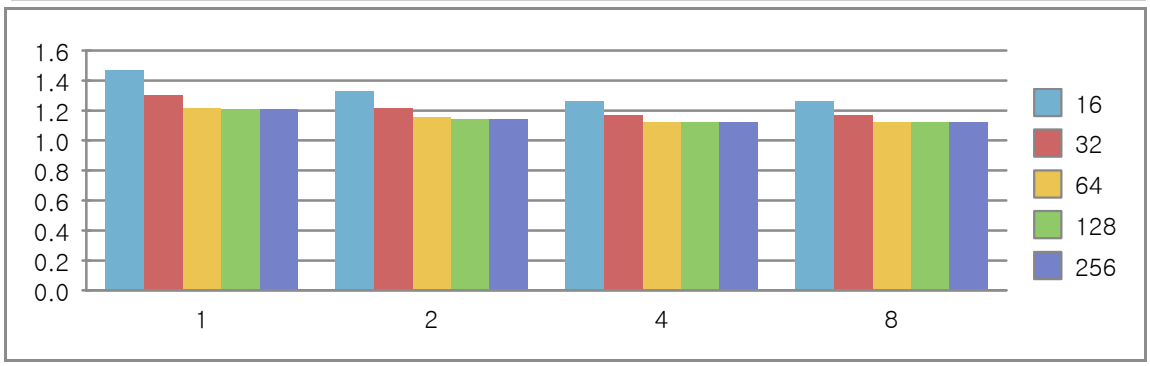
현재 사용중인 L1 캐시가 64K, L2 캐시가 256K이므로 64K와 256K의 접근시간을 분석한다. 256K 캐시와 64K 캐시의 Associativity Sets, Block size, Write policy에 따른 캐시의 평균 접근 시간은 다음과 같다.

● 분석 조건

| | |
|---|---|
| 22 | % Writes |
| 10 | % Dirty Data |
| 40 | Miss Penalty (cycles) |
| 1 | Hit Time (cycles) |
| 6 | Mem. Write (cycles) |

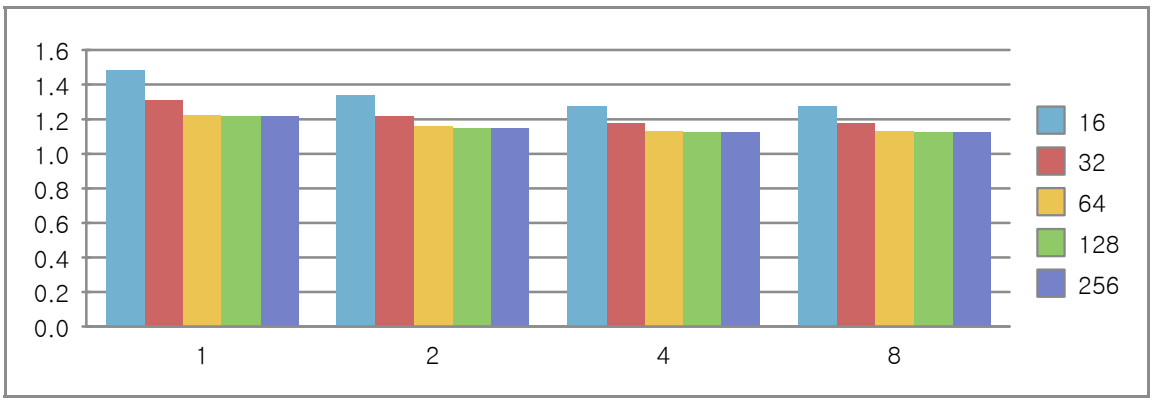
Write Back + No-Write Allocate / Cache 256K

| Sets \ block size | 16 | 32 | 64 | 128 | 256 |
|---|---|---|---|---|---|
| 1 | 1.46761 | 1.3003 | 1.21879 | 1.21021 | 1.21021 |
| 2 | 1.33033 | 1.2145 | 1.15444 | 1.14586 | 1.14586 |
| 4 | 1.26598 | 1.1716 | 1.12441 | 1.12012 | 1.12012 |
| 8 | 1.26598 | 1.1716 | 1.12441 | 1.12012 | 1.12012 |



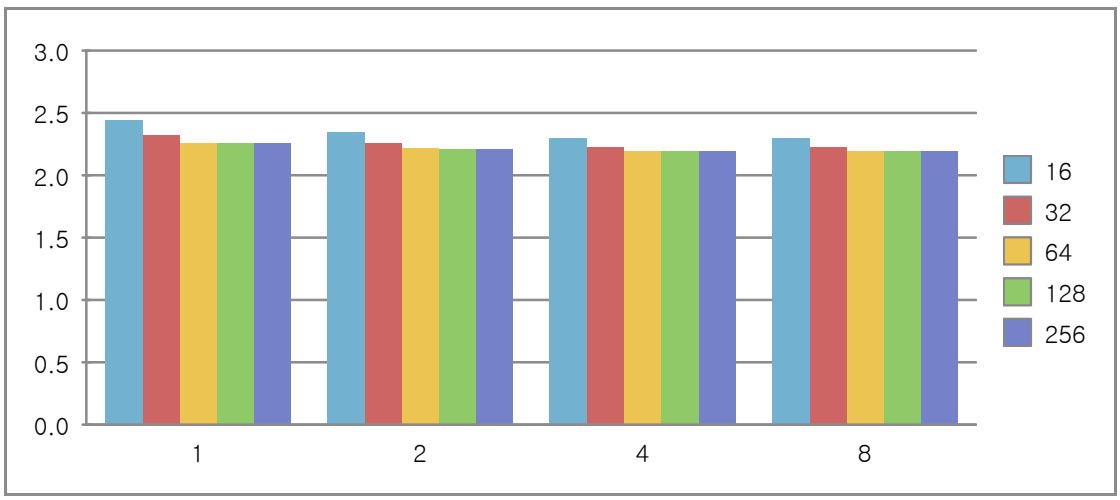
Write Back + Write Allocate / Cache 256K

| Sets \ block size | 16 | 32 | 64 | 128 | 256 |
|---|---|---|---|---|---|
| 1 | 1.4796 | 1.308 | 1.2244 | 1.2156 | 1.2156 |
| 2 | 1.3388 | 1.22 | 1.1584 | 1.1496 | 1.1496 |
| 4 | 1.2728 | 1.176 | 1.1276 | 1.1232 | 1.1232 |
| 8 | 1.2728 | 1.176 | 1.1276 | 1.1232 | 1.1232 |



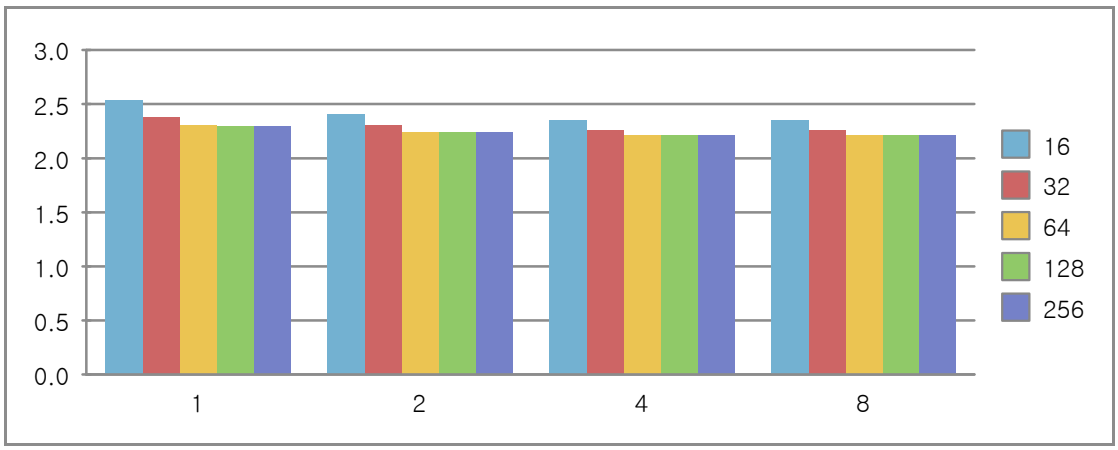

Write Through + No-Write Allocate / Cache 256K

| Sets \ block size | 16 | 32 | 64 | 128 | 256 |
|---|---|---|---|---|---|
| 1 | 2.44008 | 2.3184 | 2.25912 | 2.25288 | 2.25288 |
| 2 | 2.34024 | 2.256 | 2.21232 | 2.20608 | 2.20608 |
| 4 | 2.29344 | 2.2248 | 2.19048 | 2.18736 | 2.18736 |
| 8 | 2.29344 | 2.2248 | 2.19048 | 2.18736 | 2.18736 |



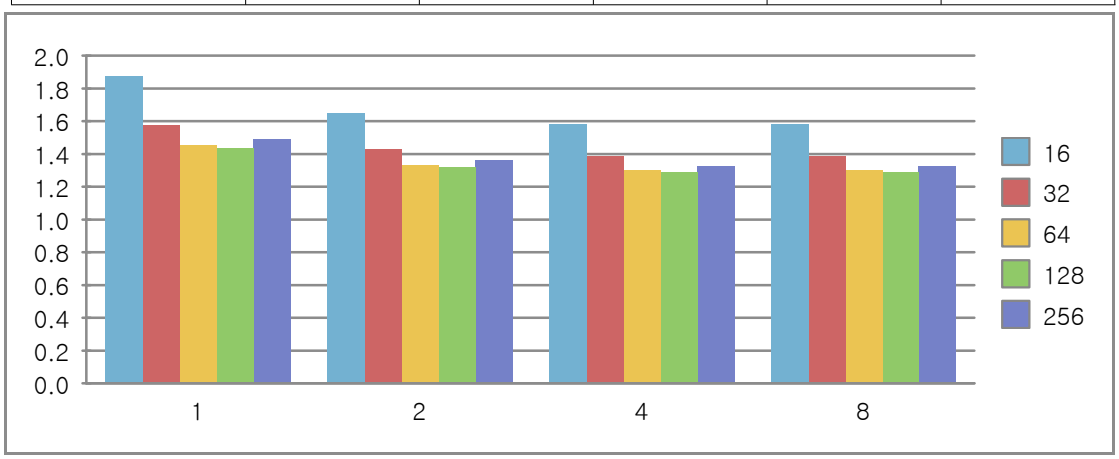
Write Through + Write Allocate / Cache 256K

| Sets \ block size | 16 | 32 | 64 | 128 | 256 |
|---|---|---|---|---|---|
| 1 | 2.536 | 2.38 | 2.304 | 2.296 | 2.296 |
| 2 | 2.408 | 2.3 | 2.244 | 2.236 | 2.236 |
| 4 | 2.348 | 2.26 | 2.216 | 2.212 | 2.212 |
| 8 | 2.348 | 2.26 | 2.216 | 2.212 | 2.212 |



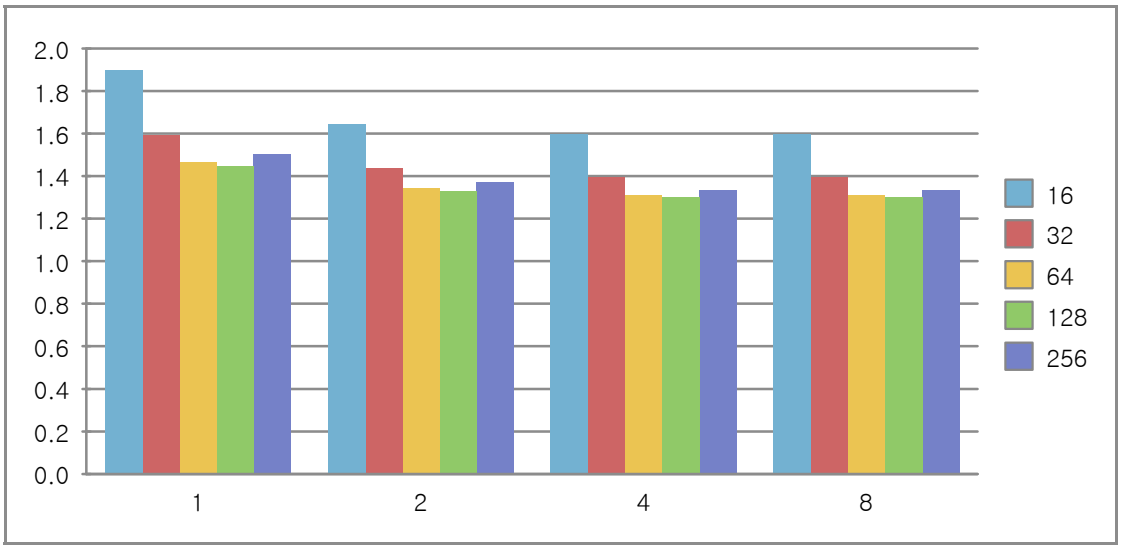

Write back + No-write Allocate / Cache 64K

| Sets \ Block size | 16 | 32 | 64 | 128 | 256 |
|---|---|---|---|---|---|
| 1 | 1.87516 | 1.57915 | 1.45474 | 1.43758 | 1.49335 |
| 2 | 1.64779 | 1.429 | 1.33462 | 1.32175 | 1.36465 |
| 4 | 1.58344 | 1.3861 | 1.3003 | 1.29172 | 1.32604 |
| 8 | 1.58344 | 1.3861 | 1.3003 | 1.29172 | 1.32604 |



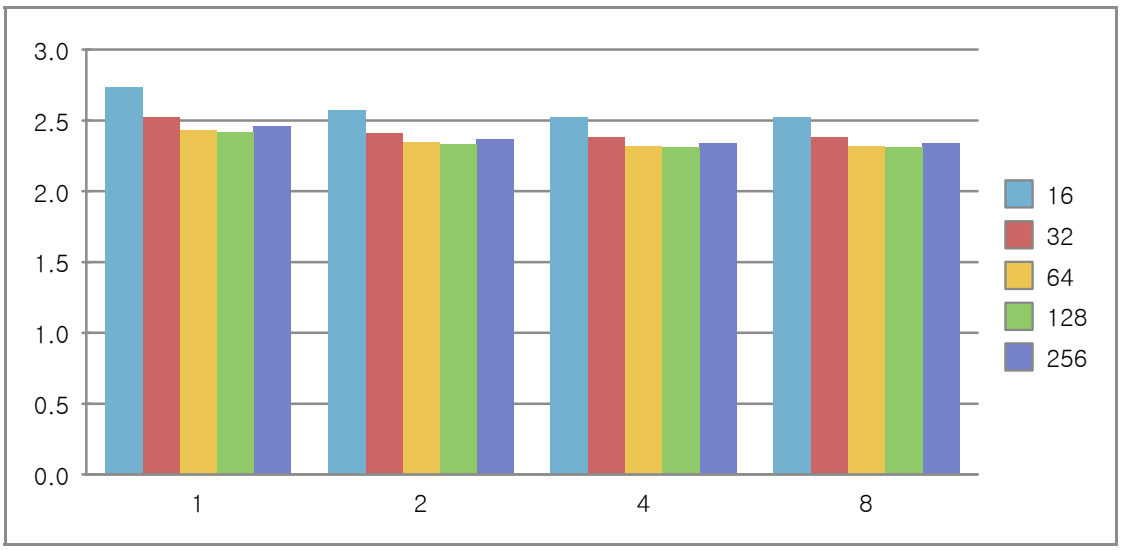
Write back + Allocate on Miss / Cache 64K

| Sets \ Block size | 16 | 32 | 64 | 128 | 256 |
|---|---|---|---|---|---|
| 1 | 1.8976 | 1.594 | 1.4664 | 1.4488 | 1.506 |
| 2 | 1.644 | 1.44 | 1.3432 | 1.33 | 1.374 |
| 4 | 1.5984 | 1.396 | 1.308 | 1.2992 | 1.3344 |
| 8 | 1.5984 | 1.396 | 1.308 | 1.2992 | 1.3344 |



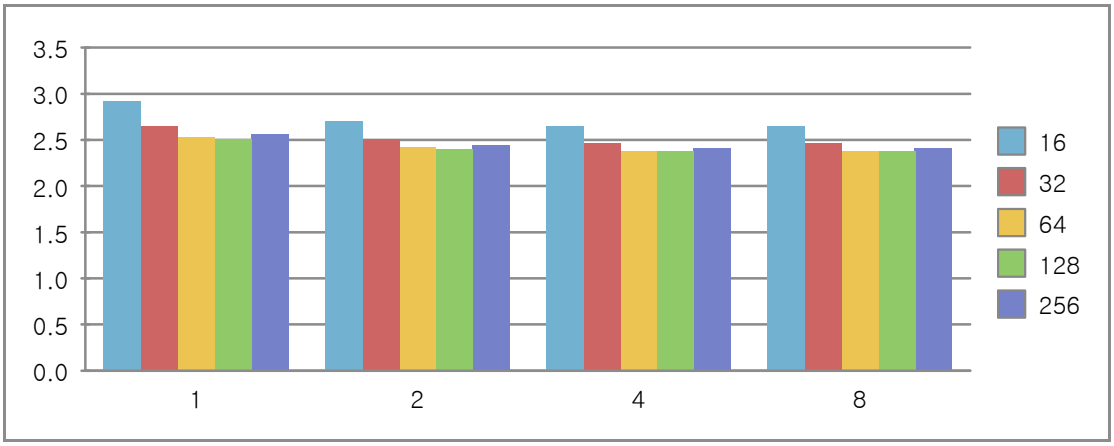

Write Through + No-Write Allocate / Cache 64K

| Sets \ Block size | 16 | 32 | 64 | 128 | 256 |
|---|---|---|---|---|---|
| 1 | 2.73648 | 2.5212 | 2.43072 | 2.41824 | 2.4588 |
| 2 | 2.57112 | 2.412 | 2.34336 | 2.334 | 2.3652 |
| 4 | 2.52432 | 2.3808 | 2.3184 | 2.31216 | 2.33712 |
| 8 | 2.52432 | 2.3808 | 2.3184 | 2.31216 | 2.33712 |



Write Through + Allocate on Miss / Cache 64K

| Sets \ Block size | 16 | 32 | 64 | 128 | 256 |
|---|---|---|---|---|---|
| 1 | 2.916 | 2.64 | 2.524 | 2.508 | 2.56 |
| 2 | 2.704 | 2.5 | 2.412 | 2.4 | 2.44 |
| 4 | 2.644 | 2.46 | 2.38 | 2.372 | 2.404 |
| 8 | 2.644 | 2.46 | 2.38 | 2.372 | 2.404 |




분석 결과를 보면 Block Size가 클수록, 집합의 수가 많아질수록 CPU의 평균 접근시간은 감소하며 Write Back(모아쓰기)이 Write Through(연속쓰기)보다 빠른 접근 시간을 가지는 것을 확인할 수 있다. 이 때 Write Back(모아쓰기)의 경우 No-Write Allocate(쓰기 비할당)와 Allocate on Miss(쓰기 할당)의 차이가 미약하나 Write Through(연속쓰기)의 경우 Allocate on Miss(쓰기 할당)보다 No-Write Allocate(쓰기 비할당)의 평균 접근 시간이 감소하는 것을 확인 할 수 있다. 일반적으로 Write-Back + Allocate on Miss와 Write Through + No-Write Allocate가 많이 사용된다. 또한 마지막으로 캐시의 평균 접근 시간은 캐시의 사이즈가 클수록 감소하는 것을 확인할 수 있다.

## 4. 결과 및 결론

캐시의 매핑방식에 따른 주소비트와 Block Replacement, 캐시 사이즈, Set 정책에 따른 캐시의 적중률, 캐시 사이즈, Associativity Sets, Block size, Write policy에 따른 캐시의 평균 접근 시간을 모두 고려하여 캐시를 설계할 때 캐시의 적중률과 캐시의 평균 접근 시간에 동시에 상반된 영향을 끼치는 캐시 사이즈는 하드웨어 가격 등을 고려하여 적당한 사이즈의 캐시를 사용하는 것이 현명하다. 또한 Set(집합의 수)은 증가할수록 캐시 평균 접근 시간을 감소시키는 이점이 있지만 Set이 1개일 때(완전 연관 매핑) 굉장한 적중률을 보이므로 하드웨어 비용이 비싸도 Set을 1개로 설정하여 최대의 적중률을 확보하거나 하드웨어 가격을

낮추는 대신 적중률을 포기하고 Set의 수를 증가시켜 캐시 평균 접근 시간을 감소시키는 방법을 고려할 수 있다. 마지막으로 가능하면 Block Size를 크게하여 캐시의 평균 접근 시간을 감소시키고 Write Back + Allocate on Miss의 Write Policy를 적용하면 최대의 캐시 성능을 얻어 낼 수 있다.