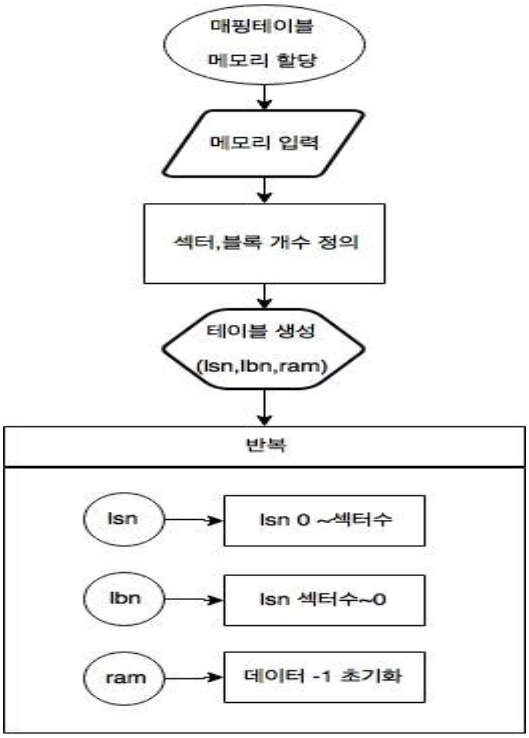
< 목 차 >

━ Sector_Mapping ━

○ 매핑테이블 생성

FlowChart



설명

●메모리 할당

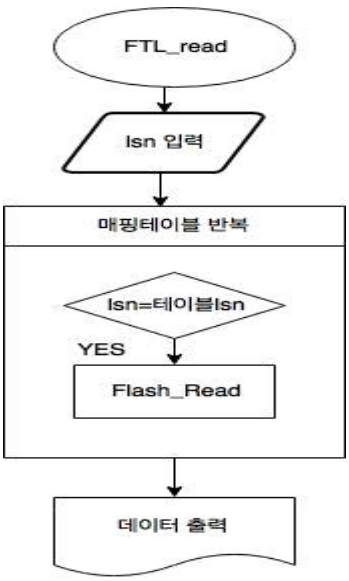
●섹터는 512byte
●블록은 32개의 섹터
●1섹터=n * 1024 * 1024 / 512

●섹터수 만큼 매핑 테이블 생성

●lsn 는 0부터 섹터수까지 입력
●psn 는 섹터수부터 0까지 입력
●ram 데이터는 -1로 초기화

○ FTL_Read

FlowChart



설명

●FTL_Read

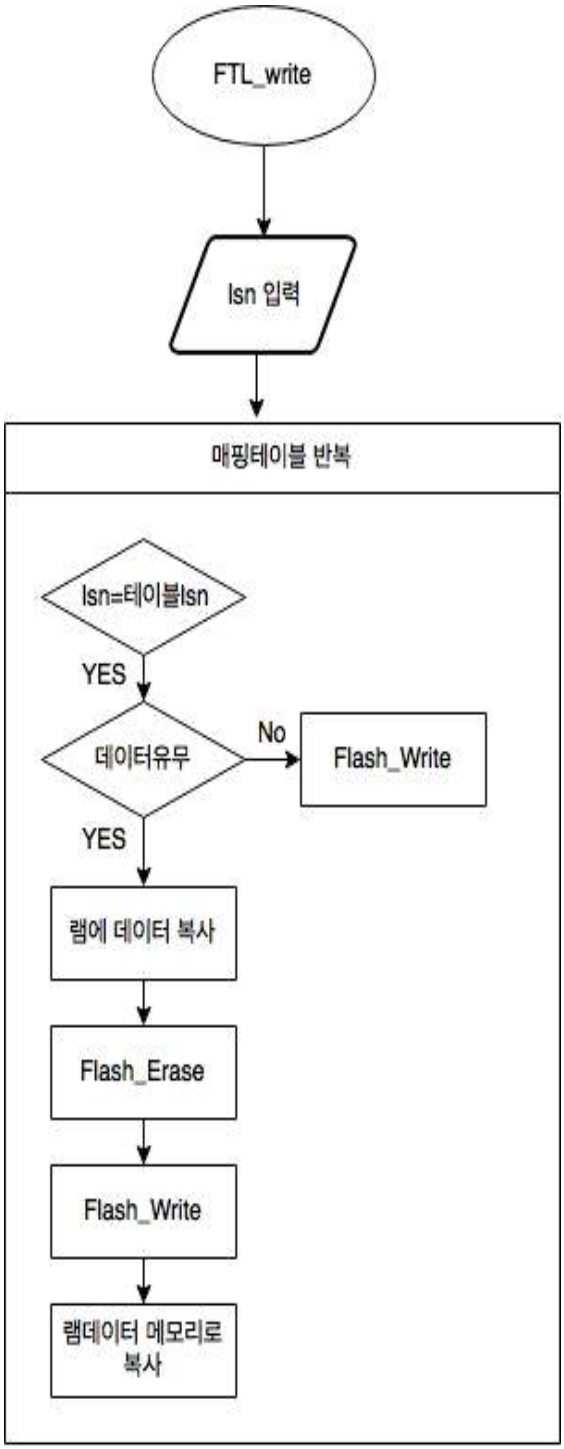
●논리 주소(lsn) 입력

●입력한 논리주소와 매핑테이블의 논리주소가 일치한 위치의 물리주소 위치를 Flash_read 한다

●데이터 출력

○ FTL_Write

FlowChart



설명

●FTL_WRITE

●논리 주소(lsn) 입력

●논리주소가 일치하는 위치의 물리주소를 매핑 테이블에서 찾는다

●물리주소에 일치하는 메모리에 데이터가 없으면 FLash_Write 연산 수행

●데이터가 있으면 먼저 메모리에 있는 데이터를 램에 복사한다

●이후에 Overwrite가 일어난 곳의 블록을 지운다

●메모리를 비웠으면 메모리에 새로운 데이터를 쓴다

●램에 저장된 나머지데이터들을 다시 메모리로 복사한다

○섹터매핑 주요 함수 기능 및 설명

○ 메모리 생성 / 초기화

```
memory = (MEMORY*)malloc(sizeof(MEMORY)*(sector)); //섹터의 크기만큼 구조체생성


for (i = 0; i < sector; i++)
{
    strcpy(memory[i].memory, "-1"); //데이터 -1로 초기화
}
```

섹터 개수만큼 메모리의 크기를 할당받고, 데이터를 -1로 초기화한다.

○ 매핑테이블 생성 / 초기화 &램 초기화

```
table = (TABLE*)malloc(sizeof(TABLE)*(sector)); //메모리와 마찬가지로 섹터수만큼

for (i = 0; i < sector; i++)
{
    table[i].lsn = i; //논리주소는 0부터~ 섹터수까지
}

for (j = 0; j <sector; j++)
{
    table[j].psn =sec-1; //물리주소는 섹터부터~0까지 (거꾸로)
    sec--;
}

for (k = 0; k < sector; k++)
{
    strcpy(table[k].ram, "-1"); //램의 데이터 -1로 초기화
}
```

메모리와 마찬가지로 섹터 개수만큼 테이블의 크기를 할당받는다.
매핑테이블의 논리주소는 0부터~섹터수 로 초기화하고
물리주소는 섹터수~0 으로 초기화한다.
추가로 램의 데이터도 -1로 초기화한다.

○ FTL_Write

```
for (i = 0; i < sector; i++)
{
    if (table[i].lsn ==lsn) //입력한 논리주소와 매핑테이블의 논리주소가 일치하는 위치를 찾고
    {
        if (!(strcmp(memory[table[i].psn].memory, "-1") == 0)) // 논리주소에 대응하는 물리주소의 메모리 섹터번호에 데이터가 있으면
        {
            Flash_copy(memory, table, table[i].psn); //램에 데이터 복사
            Flash_erase(memory, table, table[i].psn); //지우기연산 실행(지워진 데이터는 램에 저장됨)
            erase_count++;
            erase_Allcount++;
            Flash_write(memory, table,table[i].psn,data); //지우고 난 후 쓰기
            write_count++;
            write_Allcount++;

            for (j = (table[i].psn / SECTOR_SIZE) * SECTOR_SIZE; j <= ((table[i].psn / SECTOR_SIZE) * SECTOR_SIZE) + SECTOR_SIZE; j
            {
                if (strcmp(memory[j].memory, "-1") == 0) //메모리가 비어있는공간에만
                {
                    strcpy(memory[j].memory, table[j].ram); // 램데이터를 메모리로 다시 복사
                }
            }
        }
        else
        {
            Flash_write(memory, table, table[i].psn, data); //비어있지 않으면 데이터쓰기
            write_count++;
            write_Allcount++;
        }
```

FTL_Write는 입력한 논리주소와 매핑테이블의 논리주소가 일치하는 곳을 찾고
그 위치의 물리주소 찾아 데이터를 써야할 메모리의 섹터를 찾는다.
메모리가 데이터가 없으면 입력한 데이터를 메모리에 쓰고,
메모리에 데이터가 있으면 메모리의 데이터를 메모리에 복사하고 데이터가 들어있는 블록을 지운다. 블록이 지워지면 그 때 입력한 데이터를 메모리에 쓴다.
이후에 램에 복사된 유효한 데이터는 다시 메모리로 복사한다.

○ FTL_Read

```
    for (i = 0; i < sector; i++)
    {
        if (table[i].lsn == lsn) //논리주소에 대응하는 psn을 찾기위함
        {
            Flash_read(memory, table, table[i].psn);
        }
    }
```

논리주소에 대응하는 psn을 Flash_read로 읽는다.

○ 램으로 데이터 복사

```
void Flash_copy(MEMORY *memory, TABLE *table, int sector) //램으로 데이터 복사
{
    int i;
    for (i = (sector / SECTOR_SIZE) * SECTOR_SIZE; i <= ((sector / SECTOR_SIZE) * SECTOR_SIZE) + SECTOR_SIZE; i
    {
        strcpy(table[i].ram, memory[i].memory);//데이터를 램에저장
    }

    printf("Copy To Ram Success ! ₩n");
}
```

데이터가 있는 곳에 데이터가 쓰이면 데이터가 있는 블록을 램으로 복사한다

○ 메모리,매핑테이플,램데이터 출력

```
void print_memory(MEMORY *memory, int sector) //메모리에 저장된 데이터 출력
{
    int i;

    for (i = 0; i < sector; i++)
    {
        printf("psn%d = %s₩n", i, memory[i].memory);
    }
    printf("₩n");
}

void print_table(TABLE *table, int sector) //매핑테이블 데이터 출력
{
    int i;

    for (i = 0; i < sector; i++)
    {
        printf("lsn%d = %d ₩t psn%d = %d₩n", i, table[i].lsn, i, table[i].psn);
    }
    printf("₩n");
}

void print_ram(TABLE *table,int sector) //램 데이터 출력
{
    int i;

    for (i = 0; i <sector ; i++)
    {
        printf("Num %d : %s ₩n", i, table[i].ram);
    }
}
```

메모리,매핑테이블,램데이터를 출력한다.

○섹터매핑 FTL_Write / FTL_Read 출력결과

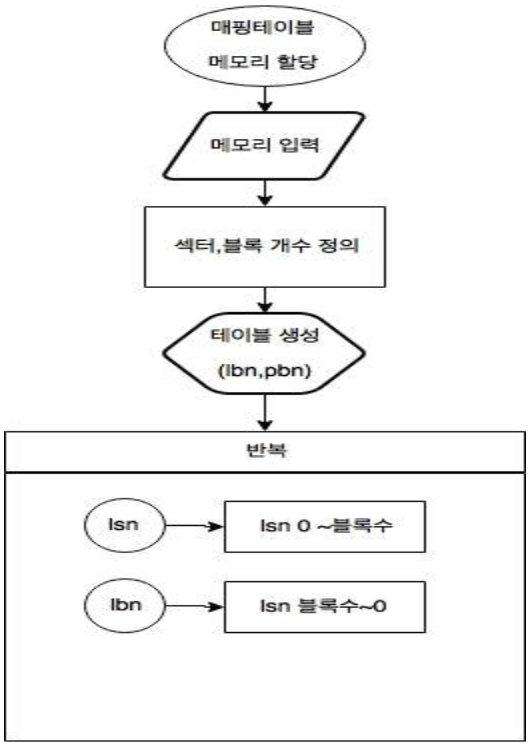
FTL_WRITE -> 메모리 데이터	
Number:1 Input lsn,data :35 A write(2012,A) Success !	psn2009 = -1 psn2010 = -1 psn2011 = -1 psn2012 = A psn2013 = -1 psn2014 = -1
Number:1 Input lsn,data :36 B write(2011,B) Success !	psn2009 = -1 psn2010 = -1 psn2011 = B psn2012 = A psn2013 = -1
Number:1 Input lsn,data :35 A' Copy To Ram Success ! Block 62 (psn1984 ~ psn2015) Erase	psn2009 = -1 psn2010 = -1 psn2011 = B psn2012 = A' psn2013 = -1

FTL_WRITE -> FTL_READ	
Number:1 Input lsn,data :35 A write(2012,A) Success !	Number:2 Input lsn:35 psn : 2012 data : A'
Number:1 Input lsn,data :36 B write(2011,B) Success !	Number:2 Input lsn:36 psn : 2011 data : B
Number:1 Input lsn,data :35 A' Copy To Ram Success ! Block 62 (psn1984 ~ psn2015) Erase	Number:2 Input lsn:35 psn : 2012 data : A'

━ Block_Mapping ━

○ 매핑테이블 생성

FlowChart



설명

●메모리 할당

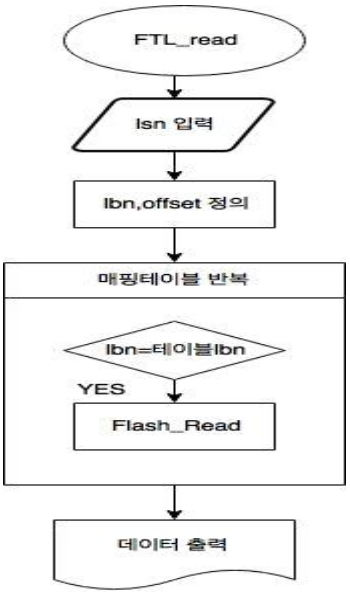
●섹터는 512byte
●블록은 32개의 섹터
●1섹터=n * 1024 * 1024 / 512

●블록수 만큼 매핑 테이블 생성

●lsn 는 0부터 섹터수까지 입력
●psn 는 섹터수부터 0까지 입력
●ram 데이터는 -1로 초기화

○ FTL_Read

FlowChart



설명

●FTL_Read

●논리 주소(lsn) 입력

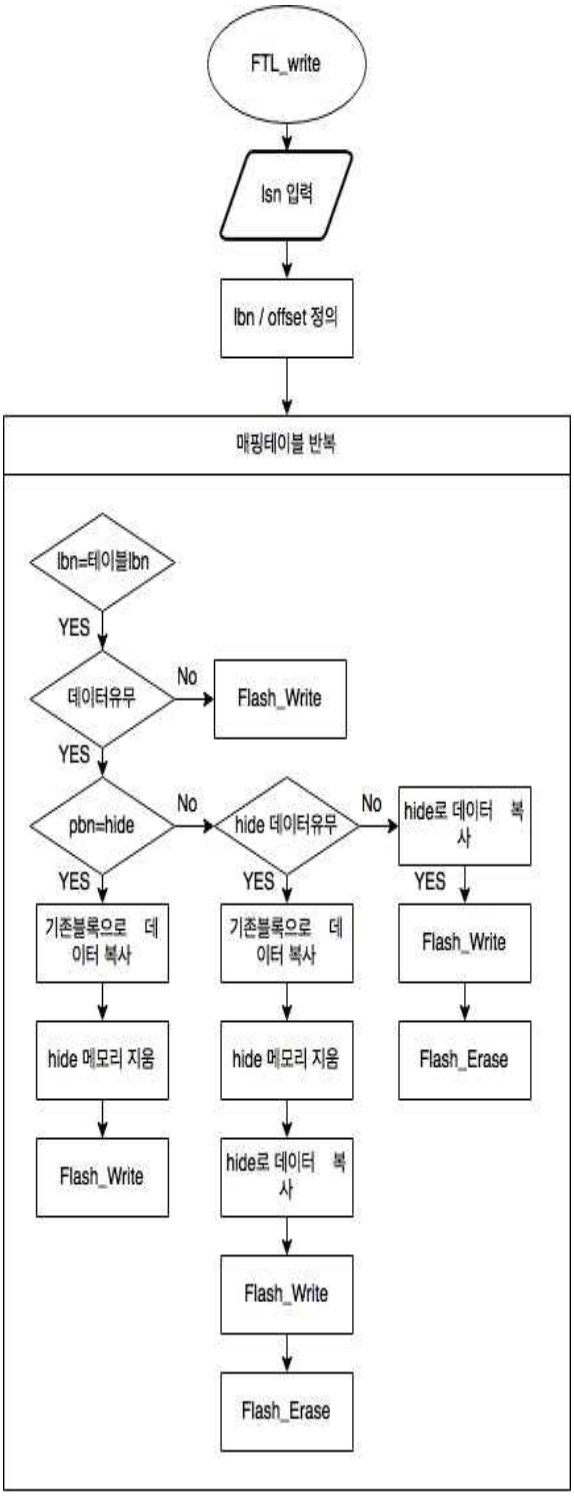
●lbn = lsn / 블록당 섹터수
●offset = lsn % 블록당 섹터수

●입력받은 lbn과 매핑테이블의 lbn이 일치하는곳의 pbn을 찾아서 데이터를 읽음

●데이터 출력

○ FTL_Write

FlowChart



설명

●FTL_WRITE

●논리 주소(lsn) 입력

●lbn = lsn / 블록당 섹터수
●offset = lsn % 블록당 섹터수

●논리주소가 일치하는 위치의 물리주소를 매핑 테이블에서 찾는다

●물리주소에 일치하는 메모리에 데이터가 없으면 FLash_Write 연산 수행

●hide(여분블록) 은 메모리의 마지막 블록으로 지정한다.

●데이터가 있을 때 pbn이 가르키는 곳이 여분영역일 때와 pbn이 가르키는 곳이 여분블록이 아닐 때 여분블록의 데이터의 유무에 따라 나눠서 실행한다.

○블록매핑 주요 함수 기능 및 설명

○ 메모리 생성 / 초기화

```
memory = (MEMORY*)malloc(sizeof(MEMORY)*(sector)); //섹터의 크기만큼 구조체생성


for (i = 0; i < sector; i++)
{
    strcpy(memory[i].memory, "-1"); //데이터 -1로 초기화
}
```

섹터 개수만큼 메모리의 크기를 할당받고, 데이터를 -1로 초기화한다.

○ 매핑테이블 생성 / 초기화

```
table = (TABLE*)malloc(sizeof(TABLE)*(block)); //블록수만큼 생성

for (i = 0; i < block; i++)
{
    table[i].lbn = i; //논리주소는 0부터~ 블록수까지
}

for (j = 0; j <block; j++)
{
    table[j].pbn = blk - 1; //물리주소는 블록부터~0까지 (거꾸로)
    blk--;
}

return table;
```

블록매핑이므로 메모리와는 다르게 블록수만큼 테이블을 생성한다.
테이블의 데이터는 lbn은 0부터~블록수 까지 넣고 pbn은 블록수~0 까지 넣는다.

○ FTL_Write

```
else
{
    if (count != 0) // 하이드공간에 데이터가 있을 때
    {
        Flash_Copy_Original(memory, table, temp, offset, hide);//하이드의 데이터를 전에 오버r
        table[temp2].pbn = temp / SECTOR_SIZE;
        Flash_erase_Hide(memory, table, hide); //하이드공간을 지우고
        erase_count++;
        erase_Allcount++;
        Flash_Copy_Hide(memory, table, table[i].pbn * SECTOR_SIZE, offset, hide); //하이드에 
        Flash_write(memory, table, hide + offset, data); //하이드에 새로운 데이터 쓰기
        write_count++;
        write_Allcount++;
        Flash_erase(memory, table, table[i].pbn*SECTOR_SIZE); //원래 있던 데이터 블록 지우기
        erase_count++;
        erase_Allcount++;
        temp = table[i].pbn * SECTOR_SIZE;
        temp2 = table[i].lbn;
        table[i].pbn = hide / 32; //매핑테이블의 물리블록을 하이드공간으로

        break;
    }
    else // 하이드공간에 데이터가 없을 때
    {
        Flash_Copy_Hide(memory, table, table[i].pbn * SECTOR_SIZE, offset, hide); //하이드에 
        Flash_write(memory, table, hide + offset, data); //하이드에 새로운 데이터 쓰기
        write_count++;
        write_Allcount++;
        Flash_erase(memory, table, table[i].pbn*SECTOR_SIZE); //원래 있던 데이터 블록 지우기
        erase_count++;
        erase_Allcount++;
        temp = table[i].pbn * SECTOR_SIZE;
        temp2 = table[i].lbn;
        table[i].pbn = hide / 32; //매핑테이블의 물리블록을 하이드공간으로
        count++;
```

위의 FlowChart에서 'hide 데이터 유무' 부분을 코드로 나타냈다.
즉,데이터가 있는공간에 데이터를 썻을 때 실행되는 코드이다.

하이드 공간에 데이터가 있으면 하이드 공간의 데이터를 전에 overwrite가 나타났던 공간으로 데이터를 복사하고(Flash_Copy_Original) 하이드공간을 지운다.
그 후에 하이드공간에 현재 overwrite가 나타난 공간에 데이터를 복사하고 (Flash_Copy_Hide) , 하이드공간에 새로운 데이터를 쓴다.
새로운 데이터가 써지면 원래 블록은 지워지고 매핑테이블의 물리블록이 하이드공간을 가르키게된다.

하이드 공간에 데이터가 없으면 하이드공간에 데이터를 복사하고 하이드공간에 새로운 데이터를 쓴다. 이후에 원래 있던 데이터 블록을 지우고 비어있는 데이터 블록을 나중에 Overwrite가 나타났을 때 기억할 수 있도록 temp에 저장해둔다.
위와마찬가지로 데이터가 하이드 공간에 써졌으므로 물리블록은 하이드공간을 가르킨다.

○ FTL_Read

```
map_lbn = lsn / SECTOR_SIZE; //입력받은값을 블록당 섹터 개수 나눈값으로 매핑테이블의 논리블록을 구함
offset = lsn % SECTOR_SIZE; //오프셋은 블록당 섹터 개수로 나눈 나머지 (오프셋으로 섹터의 번호를 찾음)

for (i = 1; i < block; i++)
{
    if (table[i].lbn == map_lbn) //논리주소에 대응하는 psn을 찾기위함
    {
        Flash_read(memory, table, table[i].pbn*SECTOR_SIZE+offset);
    }
}
```

입력받은 lsn을 블록당 섹터 개수로 나누어서 매핑테이블의 lbn을 찾는다.
lbn에 대응하는 pbn을 찾아서 pbn * 블록당 섹터 개수 + offset을 하면 정확한 메모리 위치를 찾아서 데이터를 읽을 수 있다.

○ 여분블록으로 데이터복사 / 기존블록으로 데이터복사

```
void Flash_Copy_Hide(MEMORY *memory, TABLE *table, int sector,int offset,int hide) //하이
{
    int i;

    for (i = sector; i < sector + SECTOR_SIZE; i++)
    {
        if (i == sector + offset)  // overwrite가 일어난 자리의 데이터를 제외하고 복사함
        {
            strcpy(memory[hide].memory, "-1");
            hide++;
            continue;
        }
        strcpy(memory[hide].memory, memory[i].memory);
        hide++;
```

```
void Flash_Copy_Original(MEMORY *memory,TABLE *table,int sector,int offset,int hide)
{
    int i;

    for (i = hide; i < hide + SECTOR_SIZE; i++)
    {
        if (i == hide + offset)
        {
            strcpy(memory[sector].memory, "-1");
            sector++;
            continue;

        }
        strcpy(memory[sector].memory, memory[i].memory);
        sector++;
```

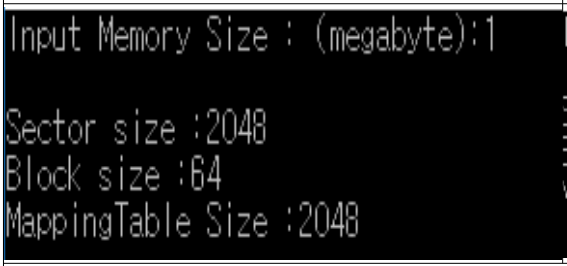
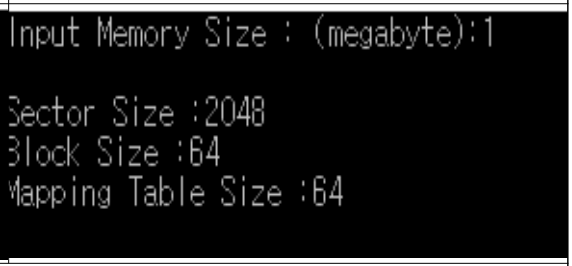
위의 FTL_Write 함수 설명에서 언급된 데이터 복사 함수이다.

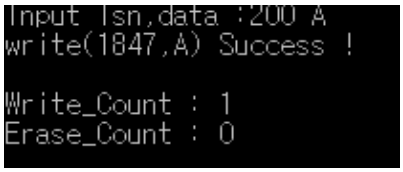
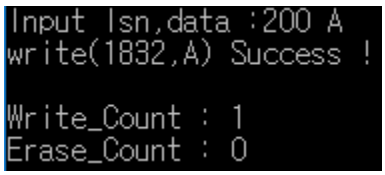
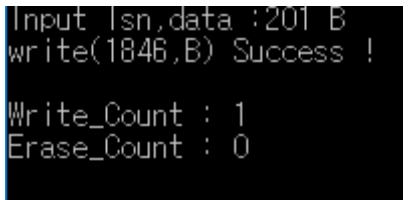
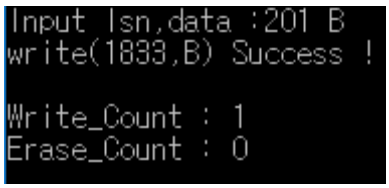
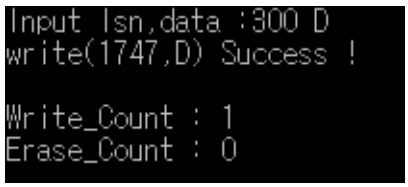
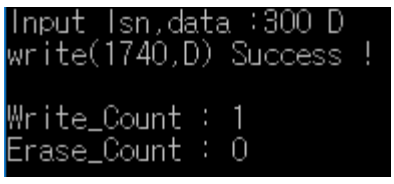
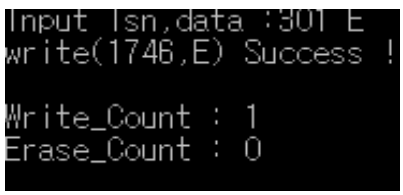
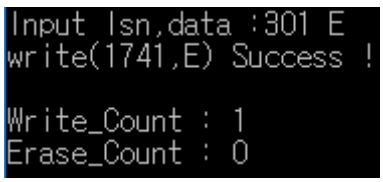
○블록매핑 FTL_Write / FTL_Read 출력결과

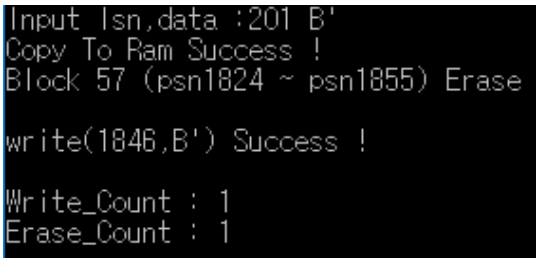
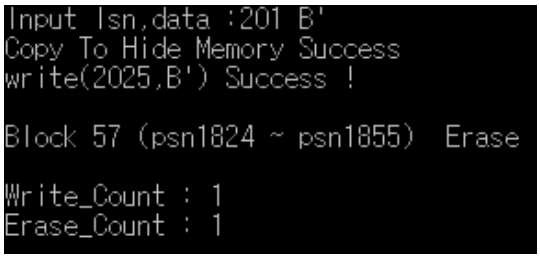
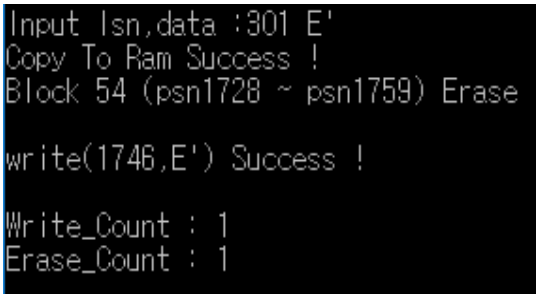
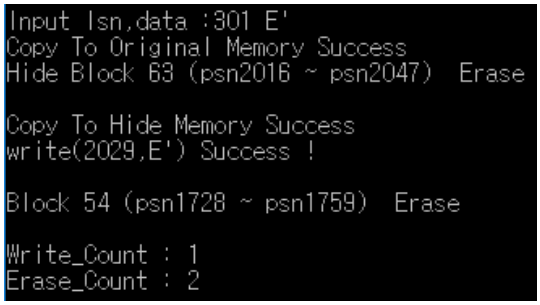
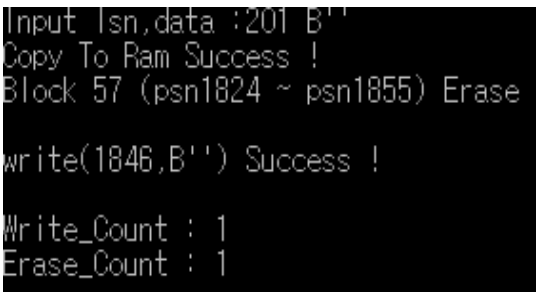
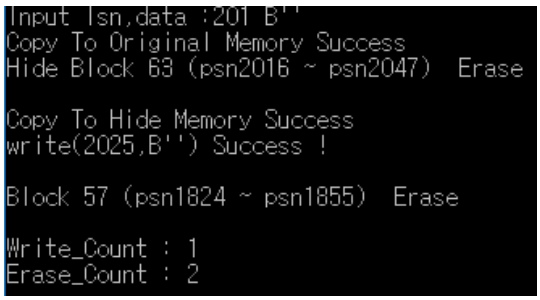
FTL_WRITE ->	메모리데이터
Number:1 Input lsn,data :35 A write(1987,A) Success !	psn1986 = -1 psn1987 = A psn1988 = -1 psn1989 = -1 psn1990 = -1
Number:1 Input lsn,data :36 B write(1988,B) Success !	psn1985 = -1 psn1986 = -1 psn1987 = A psn1988 = B psn1989 = -1
Input lsn,data :35 A' Copy To Hide Memory Success write(2019,A') Success ! Block 62 (psn1984 ~ psn2015) Erase	psn2018 = -1 psn2019 = A' psn2020 = B psn2021 = -1 psn2022 = -1 psn1996 = -1 psn1997 = -1 psn1998 = -1 psn1999 = -1 psn2000 = -1

FTL_WRITE ->	FTL_READ
Number:1 Input lsn,data :35 A write(1987,A) Success !	Number:2 Input lsn:35 psn : 1987 data : A
Number:1 Input lsn,data :36 B write(1988,B) Success !	Number:2 Input lsn:36 psn : 1988 data : B
Input lsn,data :35 A' Copy To Hide Memory Success write(2019,A') Success ! Block 62 (psn1984 ~ psn2015) Erase	Number:2 Input lsn:35 psn : 2019 data : A'

◉비교 분석

○ 생성되는 메모리 비교	
SECTOR_MAPPING	BLOCK_MAPPING
	
1MB를 입력했을 때 섹터매핑의 매핑테이블은 2048개이고 블록매핑은 64개 이다. 메모리 부분에서는 블록매핑이 섹터매핑보다 우수하다.	

○ FLASH_Erase 횟수 비교	
SECTOR_MAPPING	BLOCK_MAPPING
	
	
	
	

<table>
<tr>
<td> 
</td>
<td> 
</td>
</tr>
<tr>
<td> 
</td>
<td> 
</td>
</tr>
<tr>
<td> 
</td>
<td> 
</td>
</tr>
<tr>
<td> 
</td>
<td> 
</td>
</tr>
<tr>
<td colspan="2">지우기 연산은 읽기, 쓰기 연산에 비해 더 많은 메모리가 필요하기 때문에 연산이 많이 일어날수록 성능이 저하된다.
따라서 성능은 지우기 연산이 더 적은 섹터매핑이 블록매핑보다 우수하다.

성능개선)
블록매핑에서는 섹터매핑에서 사용한 데이터공간이 램이아닌 할당받은 메모리 공간중 한블록을 여분블록으로 사용하기 때문에 지우기연산이 많아지는 경우가 생긴다.
이 여분블록의 위치를 정적으로 하지말고 모든 블록에서 동적으로 사용할 수 있도록 하면 지우기 연산이 줄어들 것이다.</td>
</tr>
</table>