Cache simulator

typedef struct \_CACHE{

    BOOL valid;

    BOOL dirty;

    int tag;

    DATA data[64]; // max 1block = 64words

} CACHE;

typedef struct \_MEMORY{

    BOOL valid;

    DATA idx;

    DATA tag;

    DATA data[64];

} MEMORY;

Cache로 사용하기 위한 CACHE구조체 선언하여 배열형태로 사용

Data는 한 블록이 최대 64word가 되는 경우까지 고려

Write-back policy를 구현하기위한 MEMORY 구조체 선언

Valid,idx,tag를 통해 유효한 데이터인지, 찾는 데이터가 맞는지 확인

void init\_cache(){

    int i,j;

    addr\_offset = log2(block\_size);

    addr\_wordoffset = addr\_offset -2 ;

    cache\_idx = cache\_size/block\_size;

    addr\_idx = log2(cache\_idx);

    addr\_tag = 32 - (addr\_offset + addr\_idx);

    for(i=0; i<cache\_idx; i++)

    {

        cache[i].valid = 0;

        cache[i].dirty = 0;

        cache[i].tag = -1;

        for(j=0; j<block\_size/4; j++)

        {

            cache[i].data[j] = 0;

        }

    }

}

입력받은 block\_size와 cache\_size를 통해서 offset, word offset, cache index, index비트수, tag비트 수를 계산한다.

전체 cache를 기본값으로 초기화 해준다.

void init\_memory(){

    int i,j;

    for(i=0; i<270000; i++)

    {

        mem[i].valid = 0;

        mem[i].idx = -1;

        mem[i].tag = -1;

        for(j=0; j<block\_size/4; j++)

        {

            mem[i].data[j] = 0;

        }

    }

}

Memory 또한 초기화 시켜준다.

BOOL isHit(MEMREQUEST \*mem\_request){

    int i,j;

    int mask = 1;

    this\_tag = mem\_request->addr;

    for(i=0; i<32-addr\_tag; i++)

        this\_tag = this\_tag >> 1;

    //this\_tag = this\_tag >> (32-addr\_tag) ; //this request tag

    //determin this request idx

    if (addr\_idx != 0)

    {

        this\_idx = mem\_request->addr;

        for (i = 31; i >= 32 - addr\_tag; i--)

        {

            mask = mask << i;

            this\_idx &= ~(mask);

            mask = 1;

        }

        this\_idx = this\_idx >> addr\_offset;

    }

    else

    {

        this\_idx = 0;

    }

    if (addr\_wordoffset != 0)

    {

        this\_word = mem\_request->addr;

        mask = 1;

        for (i = 31; i >= 32 - (addr\_tag+addr\_idx); i--)

        {

            mask = mask << i;

            this\_word &= ~(mask);

            mask = 1;

        }

        this\_word = this\_word >> 2;

    }

    else

    {

        this\_word = 0;

    }

//printf("0\n");

    if (cache[this\_idx].valid == 1)//valid check

    {

        if (cache[this\_idx].tag == this\_tag)//hit

        {

            hit\_cnt++;

            return TRUE;

        }

        else //miss, not my tag

        {

            //printf("1\n");

            miss\_cnt++;

            return FALSE;

        }

    }

    else //miss

    {

        //printf("2\n");

        //printf("miss index: , tag:%lu\n",this\_tag);

        miss\_cnt++;

        return FALSE;

    }

}

isHit 함수를 통해서 이번에 읽은 address의 tag, index, word offset값을 각 변수에 저장한다.

이때 각 값은 쉬프트 연산자를 통해 구해낸다.

그 변수의 index와 tag를 토대로 현재 캐시에 찾고자 하는 데이터 블록이 존재하는지 검사한다.

void insert\_to\_cache(MEMREQUEST \*mem\_request){

    int i;

    if( cache[this\_idx].valid == 1) //cache에 데이터가 있을때 evict

    {

        if(cache[this\_idx].dirty == 1) //write-back 해줘야하면

        {

            dirty\_block\_num--;

            //insert to memory

            mem[mem\_idx].valid = 1;

            mem[mem\_idx].idx = this\_idx;

            mem[mem\_idx].tag = cache[this\_idx].tag;

            //printf("evict addr: %lx, idx: %lu, tag: %lu\n",mem\_request->addr,this\_idx,this\_tag);

            for (i = 0; i < block\_size / 4; i++)

            {

                //printf("%ld ",cache[this\_idx].data[i]);

                mem[mem\_idx].data[i] = cache[this\_idx].data[i];

            }

            mem\_idx++;

        }

        cache[this\_idx].tag = this\_tag;

        cache[this\_idx].dirty = 0;

        if(search\_mem() == TRUE)

        {

            printf("true addr: %lx, idx: %lu, tag: %lu\n",mem\_request->addr,this\_idx,this\_tag);

            for (i = 0; i < block\_size / 4; i++)

            {

                printf("%ld ",bufdata[i]);

                cache[this\_idx].data[i] = bufdata[i];

            }

            puts("");

        }

        else

        {

            for (i = 0; i < block\_size / 4; i++)

                cache[this\_idx].data[i] = 0;

        }

    }

    else //빈 cache 일때

    {

        cache[this\_idx].valid = 1;

        cache[this\_idx].tag = this\_tag;

        cache[this\_idx].dirty = 0;

        if(search\_mem() == TRUE)

        {

            for (i = 0; i < block\_size / 4; i++)

                cache[this\_idx].data[i] = bufdata[i];

        }

        else

        {

            for (i = 0; i < block\_size / 4; i++)

                cache[this\_idx].data[i] = 0;

        }

    }

}

Miss 인경우 insert\_to\_cache 함수를 통해 해당 index의 캐시에 데이터가 저장돼있었는지 여부에 따라 경우를 나눠서 캐시에 로드한다.

이때, 데이터가 있었는 경우 evict를 하면서 memroy배열에 저장해준다.

또, 새로 캐시에 로드하는 address에 이전에 쓴적이 있어서 memory에 저장한 적이 있으면 그 데이터를 캐시에 로드한다.

이와 같은 과정을 토대로 주어진 범위내의 input을 토대로 cache를 simluate할 수 있다.