Computer Architecture

Project 3 : Cache Memory PageTable Simulator

Peter-Huang

A. Simulator Design

1. Data Structure

大部份保留並延伸 single cycle simulator 的設計方式。並新增了，相關的 structure for Cache Memory PageTable Simulator. 例如：1. **MemorySettingStructure** for recording memory settings like memory size and cache size, etc. 2. **MemoryStructure** for recording memory Information like valid bit, 3. **HitRateStructure** for recording hit and miss count. 4. **CacheStructure** for recording cache information. 5. **PageTableStructure** for recording PageTable information. 6. **TLBStructure** for recording tlb information.

設計原則：因爲 SPEC 要求可以動態配置參數，這樣會造成實際的 memory or pageTable Block 個數，會變動。所以，我利用最大記憶體是 1K 的限制，讓每個陣列長度都是 1024, (PS.其實更準確 256 就可以了，因爲 MIPS 以 WORD 爲單位是 4BYTE), 而實際的長度，再記錄到對應的 STRUCTURE 裡，這樣，就可以達到動態調整參數的需求。

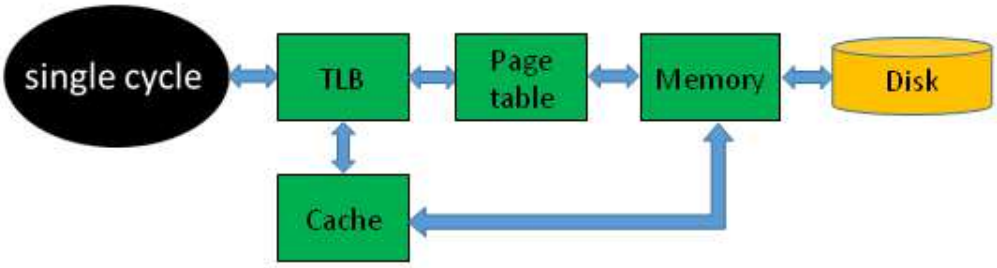
```
typedef struct MemorySettingStructure{
    int memorySize;          // 記錄 memory size
    int pageSize;            // 記錄 page size
    int cacheSize;           // 記錄 cache size
    int cacheBlockSize;      // 記錄 cache block size
    int cacheSetAssociativity;   // 記錄 cache SetAssociativity
}MemorySetting;

typedef struct MemoryStructure{
    int memLen;              // 記錄 memory 的 blocks 有幾個 ()
    int valid[1024];         // 記錄 memory 的 valid bit
    int seqNO_LRU[1024];         // 記錄的 LRU sequential NO.
    int current_SeqNO_LRU; //記錄下一個沒用過 LRU sequential No.
    int dataLen;                 // 記錄 memory 的 data 的長度
```

```
    int data[1024];                 // data 是以 int 爲單位存資料
}Memory;

typedef struct HitRateStructure{
    int hit;                        // hit count
    int miss;                       // miss count
}HitRate;

typedef struct CacheStructure{
    int cacheLen;       // recording how many cache block there are
    int valid[1024];    // cache valid bit
    int tag[1024];      // cache tag
    int seqNO_LRU[1024];    // cache LRU sequential no
    int current_SeqNO_LRU;  // next un-used LRU sequential No.
    int dataLen;            // cache data length
    int data[1024];         // cache data
    HitRate hitRate;        // cache hit rate
    int cacheBlockAddr[1024];   /** for debugging to validate if my
calculated formula for cacheBlockAddr is right. No-used in my program.
Just for validate cacheBlockAddr. **/
}Cache;

typedef struct PageTableStructure{
    int pageTableLen;   //How many pagetable entry there are.
    int content[1024];  // pageTable content to recording PA
    int valid[1024];    // valid bit
    HitRate hitRate;    // hit rate
}PageTable;

typedef struct TLBStructure{
    int tlbLen;         // how many tlb block in TLB
    int content[1024];  // recoridng PA
    int valid[1024];    // recoridng valid bit
    int tag[1024];      // recording tag
    int seqNO_LRU[1024];    // sequential LRU no
    int current_SeqNO_LRU; // next un-used LRU sequntial No.
    HitRate hitRate;
}TLB;
```

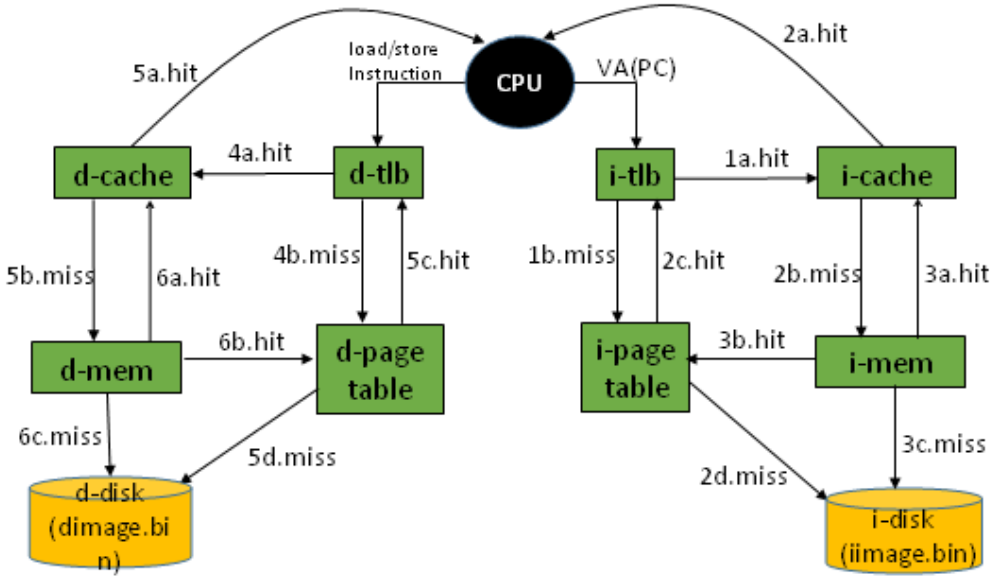
以上的設計，其實很直觀，對應參數的名命，就可以知道實際的用途，所以不多做解釋。

2、Execution Flow　(如助教投片中)

Main flow **(1) 先去 TLB 查資料，TLB HIT 去 CAHCE 查**
(2) TLB MISS，去 PAGETABLE 查，
PAGE TABLE HIT 去 CAHCE 查
(3) PAGE TABLE MISS 去 DISK 搬 data to Memory
再去 cache 查
(4) 把 miss 的資料，寫進 cache , PAGE table , TLB



Detail Flow:



Detail flow:

(1) 先透過 **virtual address** 去 **TLB** 查資料

If TLB hit, 去 **TLB content** 拿 **physical address**,
再拿 **PA** 去 **cache** 裡查資料 **(ps. cache** 又分 **cache hit and miss,** 在這段內容 **(3)**，統一說明**)**。

If TLB miss, 去 **Page table** 查資料，**(ps. Page table** 又分 **page table hit and miss.** 在這段內容 **(2)**，統一說明**)** , **Page table** 查完，回到 **TLB**，找出是 **a.** 否有 **valid bit ==0**, **b.**如果沒有 **valid==0**，利用 **LRU** 找出 **victim** 去 **replace** 這 **victim**。 之後，把 **PA** 寫回 **TLB**, 然後，再拿 **PA** 去 **CAHCE** 查資料。

(2) 透過 **virtual address** 去 **Page table** 查資料

If Page Table Hit, 代表資料在 **Page Table** 回到 **TLB**

If Page Table Miss, 代表資料在 **disk** 而不在 **memory**, 先 **move data from disk into memory**, 然後 **a.** 在 **page table** 找出是否有 **valid bit ==0**,
b.如果沒有，利用 **LRU** 找出 **victim**, 之後，把 **PA** 寫回 **Page Table** 的 **content**, **(ps.**注意如果，有 **swap Page** 時，記得，在 **page table , TLB, cache** 把相關參考對應的 **page talbe content** 的值， 都設成 **invalid** ，因爲 **Page in Page Table has been replaced**。**)** 然後回到 **TLB**,

(3) 透過 **PA** 去 **cache** 查資料 **(PA** 的取得透過 **Page Table content)**

If cache hit, **return cache block index**

If cache miss, 代表資料從 **memory** 而不在 **cache**, 先 **move data from memory into cache.** 找出是否有 **valid bit ==0**, 如果沒有，利用 **LRU** 找出 **victim**, 把資料 **replace.**

以上**(1)(2)(3)**的動作，如果有對 **cache**，**tlb**，**page table** 進行讀取的，都要變他的 **LRU sequential No.** 這樣就可以知道，哪一筆資料是最新的被參考的，哪筆是最舊的。

實作的準備

這是這**專案的核心，先與同學討論，先了解各個運算**，memory cache 間的運算方式. 及如何計算的，**了解之後，其實這 project 3 就完成了。**

Memory Sizes: 1024 = 2^10 bytes
Cache Sizes: 64 = 2^6 bytes
Block Sizes: 4 bytes

How to mapping it ➔

Address Bit Length = 10 bit.
Memory Block sizes: 1024/4 = 256
Cache Block sizes : 64/4 = 16

(1) **Directed Mapping:**
Offse =2 bits (Block sizes: 4 =2^2)
Index= 4 bits (Cache Block Size:16=2^4bytes)
Tag = 10 – 2 – 4 = 4 bits

(2) **4-Way :**
Offset = 2 bit
Index = 2 bit (Cache Block sizes / 4-way = 16/4 = 4 =2^2)
Tag = 10-2-2 = 6 Bits

(3) **Fully Associativity:**
Offset = 2 bits
Tag = 10 -2 =8 bits

實作的設計

A. LRU 的設計:

1. 給一個流水號，從 0 開始，每次加 1，當要找 victim 去 replace 時，只要看 seqNO_LRU 選，裡面最小的數字，就是 least LRU index。

2. 例外處理，因爲數字可能會超過，一定值後，會變負值，所以判斷方式會因些改變，因爲 sequence NO 順序就不同了。所以寫這

shiftLRUSeqNoWhenBeyondLimit() ，只要 **LRU seqNO** 超過我定的最大值，就找出目前裡面所有最小的值，然後，每一個 **LRU seqNo** 都扣掉那個最小值。
Ex: 9999, 9988, 9977, ➔ 都扣掉 9977 ➔ 22,11,0

B. 設計 Memory, TLB, PAGE TABLE, Cache 查詢

因爲 D-memory 與 I-memory 的查詢方式一樣，所以，可以用相同的程式。透過，**lookupTLB(), lookupPageTable(), lookupCache();**

lookupTLB() 主要去實作 **Detail flow (1)** 的部份。
lookupPageTable () 主要去實作 **Detail flow (2)** 的部份。
lookupCache 主要去實作 **Detail flow (3)** 的部份。

而程式的流程，其實就是 Detail Flow 的實作。而裡面細部的就是，**各個的位址的計算了，其實作方式是參考課本的方式**。以計算 cache 爲例，因 cache 比較複雜。

(1) 首先，不管是哪個 byte address，一定要轉成相對應的 Block Address.
這步，是爲了知道，查到每個 unique 的 ID ,透過這 ID 了解到，我們到底存了什麼資料，在哪裡。

如：以 cacheBlockAddr，位址：0-1023, 存在 16 個的 block size 的 cache，爲例。

這樣我們哪任何一數在０-1023 裡去除１６, cacheBlockAddr＝(0~1023) /16 都會是一個 unique 的 ID, 我們就可以知道，到底現在的資料是存哪個 byte 位址的 cahce 資料。

(2) 分成 a. direct map, b. fully associativity c. n-way associativity ，爲了知道，要實際存哪個 cache index 上。

a. Direct map：

```
int index = cacheBlockAddr % (*cache).cacheLen;
int tag =    cacheBlockAddr /   (*cache).cacheLen;
```

b. Fully associativity:

Fully associativity ,index will always to be zero, because cacheLen=1, then
(cacheBlockAddr % 1) =0
tag = (cacheBlockAddr /1) = cacheBlockAddr

c. N-way associativity:

```
int oneCacheSetBlocks =
        ( (*cache).cacheLen /   (*memorySetting).cacheSetAssociativity );
int   whichCacheSet =   cacheBlockAddr % oneCacheSetBlocks;
int   tag =   cacheBlockAddr / oneCacheSetBlocks;
```

計算出 tag 後，我們要去對應的 cache set 找出，valid =0 的 index , 放入資料。
而對應的 cache set 的位址計算，如下：

```
int s = (whichCacheSet* (*memorySetting).cacheSetAssociativity);
int e = s+ (*memorySetting).cacheSetAssociativity;
```

在 s-e 就是對應的 cache set index

(3) 但，之後，在查詢 cache 時，我只知道目前在哪個 cache index , 及 cahce tag 是什麼，我並不知道，原本的 cacheBlockAddr, 所以我必需依造他的 cache index 及 tag 還有 set Associativity 去還原出他 cacheBlockAddr. 所以我實作了，getCacheBlockAddrByCacheTagIndex().

a. Direct map：

```
int tag = (*cache).tag[cacheIndex];
int cacheBlockAddr = (tag * (*cache).cacheLen ) +cacheIndex ;
```

b. Fully associativity:

```
int cacheBlockAddr = (*cache).tag[cacheIndex];
```

c. N-way associativity

```
int oneCacheSetBlocks =
        ( (*cache).cacheLen /   (*memorySetting).cacheSetAssociativity );

int whichCacheSet =
```

```
        cacheIndex /    (*memorySetting).cacheSetAssociativity;

int tag = (*cache).tag[cacheIndex];
int cacheBlockAddr = (tag * oneCacheSetBlocks )+ whichCacheSet;
```

C. Write Back And Write Allocate

Write Back: 當 **cache hit** 時，若 CPU **要寫入資料到某一位址時,** 會先將資料寫入 cache 中，然後再將同一位址的資料整批一起寫入主記憶體中. **(實作方式**：write-back 是將資料量儲存到一定的量之後,會依據同區塊的資料一次整批寫回去.裡面有提到 dirty ,他是在記憶體裡面 cache 的一個 bit 用來指示這筆資料已經被 CPU 修改過但是尚未回寫到儲存裝置中.)

Write Allocate: 當 cache miss 時, **將資料從主記憶體中載入到 cache.**

這 project 3 裡，我實作了，write Allocate 及 write through, 而沒實作 write back.。其原因，是因爲如同學討論完後的結果，我們發現，project 3 只會輸出 cache,tlb, pagetable hit/mss rate, 及個 register's content. 他並沒有列出實際 memory's data or memory hit / miss rate, 所以，如果，直接用 write through 取代 write back 其實，運算結果會是相同的，且程式的改動很少，且出錯率會很低。

3． Simulator Elaboration

原則就是，**1、make common simple 2、抽象化 each function，讓功能能簡單，專注在自己的功能要用的事上**。 Follow 這原則，設計 cache memory pagetable 的相關功能：

例如：

1、透過統一的 MemorySetting, Memory , Cache, PageTable, TLB 去存對應的 I / D memory 的資料。

2、統一的 Lookup Memory , Lookup PageTable , Look Cache, 透過不同的 memory , pageTable, cache 寫，對應的位址找尋及討算方式，如果 hit or miss 出錯，我也只要去對應的 function 改我的程式就可以了。

4． Test Case Design

1、 How to implement your test case in C code or assembly codes.

我是直接寫組語，然後，透過，同學享的 assembler 去轉成 binary code.

2、 What corner case are you targeting?

這部份，如果原本的計算就算對，其實就不太可能會出錯了。我的想法，是多增加一些執行的 **lw & sw memory！！去提高同學程式出錯的可能**。

```
addi $2 , $0 , 1000        ＃$2=1000
beq   $3 , $2 , 5          #flag=A $3=0 ，爲了大範圍的存取 meory
lw     $4, 0($3)           #從 data memory[0] 讀進值到$4
sw     $4, 0($3)           #把$4 寫入 data memory[0]
sw     $4, 8($3)           #從 data memory[0]+8 = datamemory[3]
addi $3 , $3 , 4           # $3=$3+4 ($3 每次加 4)
j      0x00000001          #跳回 flag=A，直到$3=1000
add   $3, $3 , $3          # $3=$3+$3
addi $1, $1, 8             # $1=0+8
addi $10, $10, 800         #$10= 800，爲了大範圍的存取 meory
lb     $2 , 0($1)          # flag=B. load byte form 0+$1 in $2
lb     $3 , 4($1)          # load byte form 4+$1 in $3
lb     $4 , 8($1)          # load byte form 8+$1 in $4
sb     $2 , 0($1)          # store $2   to   0+$1
lb     $5 , 12($1)         # load byte form 12+$1 in $5
sb     $4 , 8($1)          # store $4   to   8+$1
lb     $6 , 16($1)         # load byte form 16+$1 in $6
sb     $3 , 4($1)          # store $3   to   4+$1
lb     $7 , 20($1)         # load byte form 20+$1 in $7
addi $1 , $1 , 4           # $1 =$1+4 ($1 每次加 4)
beq   $1 , $10 , 1         #比對$1, $10, 如果一樣跳到下 2 行。
j      0x00000010          #跳回 flag=B，直到$1=$10=800
add   $2, $2 , $2
lw     $1 , 4($0)          #下面是混合各種 lw, lhu , lh , lbu ,lb
lhu   $1 , 4($0)           #及 sw, sh, sb 及運算去增加 hit & miss 的
lh     $1 , 4($0)          #計算
```

```
lbu    $1 , 4($0)
lb     $1 , 4($0)
lui    $1 , 0
andi $1 , $1 , 33
ori    $8 , $1 , -44
sw     $8 , 0($0)
lw     $9 , 0($0)
ori    $8 , $0 , 33
sh     $8 , 0($0)
lw     $9 , 0($0)
ori    $8 , $0 , 44
sb     $8 , 0($0)
lw     $9 , 0($0)
ori    $8 , $0 , -33
sll    $8 , $8 , 16
ori    $8 , $8 , -44
sw     $8 , 4($0)
lw     $9 , 4($0)
lh     $9 , 4($0)
lhu    $9 , 4($0)
lb     $9 , 4($0)
lbu    $9 , 4($0)
sh     $10, 510($0)
lb     $11, 511($0)
sb     $11, 513($0)
lhu    $10, 538($0)
lh     $10, 542($0)
lb     $11, 600($0)
lh     $10, 612($0)
sw     $31, 12($0)
sw     $4 , 8($0)
sw     $5 , 4($0)
sw     $6 , 0($0)
add    $2, $6 , $6
halt
```