**Experiment 5**

**AIM:** Implement Direct Memory Mapping using Given set of data and link. Use UNC miniMIPS Simulator and C Program.

**THEORY:**

Direct mapping is a procedure used to assign each memory block in the main memory to a particular line in the cache. If a line is already filled with a memory block and a new block needs to be loaded, then the old block is discarded from the cache.

Direct mapping divides an address into three parts:

* t tag bits
* l line bits
* w word bits.

The word bits are the least significant bits that identify the specific word within a block of memory. The line bits are the next least significant bits that identify the line of the cache in which the block is stored. The remaining bits are stored along with the block as the tag which locates the block’s position in the main memory.

Cache line number = (Main Memory Block Address) Modulo (Number of lines in Cache)

Diagram

Description automatically generated with medium confidence

After CPU generates a memory request,

* The line number field of the address is used to access the particular line of the cache.
* The tag field of the CPU address is then compared with the tag of the line.
* If the two tags match, a **cache hit** occurs and the desired word is found in the cache.
* If the two tags do not match, a **cache miss** occurs. In case of a cache miss, the required word has to be brought from the main memory.
* It is then stored in the cache together with the new tag replacing the previous one.

Diagram, schematic

Description automatically generated

**UNC miniMIPS Assembly Language:**

A typical line of assembly code specifies a single primitive operation, called an instruction, and its operands. Instructions are specified via short mneumonics of the operation specified. A list of comma-separated operands follows each instruction's mneumonic. The assembler uses this mneumonic and operand list to generate a binary encoding of the instruction, which is stored as a word in memory. Thus, an assembly language program is merely a way of generating a sequence of binary words, that the computer interprets as either (or both) program or data.

Example line of assembly code

add $3, $4, $4 .… ("add" is the instruction mneumonic and "$3,$4,$4" is a list of 3 operands)

**CODE:**

**Trace file used:**

main: addu $t0,$0,$0

addiu $t1,$0,80

addu $t2,$0,$0

loop: lw $t3,array($t0)

addu $t2,$t2,$t3

addiu $t0,$t0,4

bne $t0,$t1,loop

\*done: beq $0,$0,done

array: .word 1,2,3,4,5,6,7,8,9,10

.word 11,12,13,14,15,16,17,18,19,20

**trace.txt:**

0x80000000

0x80000004

0x80000008

0x8000000C

0x00000020

0x80000010

0x80000014

0x80000018

0x8000000C

0x00000024

0x80000010

0x80000014

0x80000018

0x8000000C

0x00000028

0x80000010

0x80000014

0x80000018

0x8000000C

0x0000002C

0x80000010

0x80000014

0x80000018

0x8000000C

0x00000030

0x80000010

0x80000014

0x80000018

0x8000000C

0x00000034

0x80000010

0x80000014

0x80000018

0x8000000C

0x00000038

0x80000010

0x80000014

0x80000018

0x8000000C

0x0000003C

0x80000010

0x80000014

0x80000018

0x8000000C

0x00000040

0x80000010

0x80000014

0x80000018

0x8000000C

0x00000044

0x80000010

0x80000014

0x80000018

0x8000000C

0x00000048

0x80000010

0x80000014

0x80000018

0x8000000C

0x0000004C

0x80000010

0x80000014

0x80000018

0x8000000C

0x00000050

0x80000010

0x80000014

0x80000018

0x8000000C

0x00000054

0x80000010

0x80000014

0x80000018

0x8000000C

0x00000058

0x80000010

0x80000014

0x80000018

0x8000000C

0x0000005C

0x80000010

0x80000014

0x80000018

0x8000000C

0x00000060

0x80000010

0x80000014

0x80000018

0x8000000C

0x00000064

0x80000010

0x80000014

0x80000018

0x8000000C

0x00000068

0x80000010

0x80000014

0x80000018

0x8000000C

0x0000006C

0x80000010

0x80000014

0x80000018

**A) Direct Mapping(tag size=8 bits):**

#include <stdio.h>

int tag[8];

int main( )

{

int addr;

int i, j, t;

int hits, accesses;

FILE \*fp;

fp = fopen("trace.txt", "r");

hits = 0;

accesses = 0;

printf("Direct Mapping Program (tag size = 8 bits):\n\n");

while (fscanf(fp, "%x", &addr) > 0) {

/\* simulate a direct-mapped cache with 8 words \*/

accesses += 1;

printf("%3d:0x%08x ", accesses, addr);

printf("\n")

i = (addr >> 2) & 7;

printf(" Cache Location:%d ",i);

t = addr | 0x1f;

printf(" Instruction Address:0x%08x ",t);

printf("tag: 0x%08x ",tag[i]);

if (tag[i] == t) {

hits += 1;

printf(" 'Hit' at %d ", i);

} else {

/\* allocate entry \*/

printf(" 'Miss' ");

tag[i] = t;

}

printf("\n Cache Status:");

for (i = 0; i < 8; i++)

printf("0x%08x ",tag[i]);

printf("\n\n");

}

printf("\n\n");

printf("Hits = %d, Accesses = %d, Hit ratio = %f\n", hits, accesses, ((float)hits)/accesses);

close(fp);

return 0;

}

**B) Direct Mapping (tag size=16 bits):**

#include <stdio.h>

int tag[16];

int main( )

{

int addr;

int i, j, t;

int hits, accesses;

FILE \*fp;

fp = fopen("trace.txt", "r");

hits = 0;

accesses = 0;

printf("Direct Mapping Program:\n\n");

while (fscanf(fp, "%x", &addr) > 0) {

/\* simulate a direct-mapped cache with 8 words \*/

accesses += 1;

printf("%4d:0x%08x ", accesses, addr);

printf("\n");

i = (addr >> 2) & 15;

printf(" Cache Location:%d ",i);

t = addr | 0x1f;

printf("\t Instruction Address:0x%08x ",t);

printf("\t tag: 0x%08x ",tag[i]);

if (tag[i] == t) {

hits += 1;

printf("\t 'Hit' at %d ", i);

} else {

/\* allocate entry \*/

printf("\t 'Miss' ");

tag[i] = t;

}

printf("\n Cache Status:");

for (i = 0; i < 16; i++)

printf("0x%08x ",tag[i]);

printf("\n\n");

}

printf("\n\n");

printf("Hits = %d, Accesses = %d, Hit ratio = %f\n", hits, accesses, ((float)hits)/accesses);

close(fp);

return 0;

}

**Output:**

**A) Direct Mapping (tag size = 8 bits):**

Text

Description automatically generated with low confidence

Text

Description automatically generated with medium confidence

Text

Description automatically generated

**B) Direct Mapping (tag size = 16 bits):**

Table

Description automatically generated

Table

Description automatically generated

Table

Description automatically generated

**ANALYSIS:**

In Direct Mapping, basic requirement is that it Needs only one comparison because of using direct formula to get the effective cache address. Main Memory Address is has 3 fields : TAG, BLOCK & WORD. The BLOCK & WORD together make an index. The least significant TAG bits is used to identify a unique word within a BLOCK of Main Memory. In the cache organization there is aa unique address for each block in the main memory which can be interpreted from the 3 fields. If the processor needs to access same memory location from 2 different main memory pages frequently, cache hit ratio decreases. Search time is less here because there is one possible location in the cache organization for each block from main memory.Thus, Direct mapping is simple and easy to implement, fast performance, and takes less time to detect a cache hit and get data from the cache.

**CONCLUSION:**

In this Experiment I implemented Direct Mapping in Cache memory in C programming language using UNC miniMIPS Simulator. First we generated a trace.txt file and observe the number of hits and misses occurred if the tag matches or not with the instruction address. Finally, Hit Ratio is computed, which is greater than 50% for both cases Direct mapping with tag size =8 bits and 16 bits, thus, the number of hits occurred were greater than the number of miss. For 16 bits tag has a larger hit ratio for direct mapping than the 8 bits tag size for the same number of accesses because of the larger cache size.