**CSE205 F2 REPORT**

**PROJECT TITLE: - CACHE TIME ANALYSIS**

**Team members:-**

1. Ch. Saiteja -13BCE1032
2. D. Shashank -13BCE1036
3. K. Gourav -13BCE1068
4. M. V. Shashank -13BCE1081

In this project we are going to show how to calculate the “average memory access time” by using the values of cache time analysis. This project can be implemented through simulator as well as the “C++” programming code. It involves some calculation based on the values of cache time analysis.

**Hit**: data or instructions appears in some block in the cache.

**Hit Rate**: the fraction of memory accesses found in the cache.

**Hit Time**: Time to access the cache which consists of cache access time + Time to determine hit/miss.

**Miss**:- data needs to be retrieved from a block in the main memory.

**Miss Rate**:- 1 - Hit Rate.

**Miss Penalty**:- Time to replace a block in the cache + time to deliver the block to the processor.

Hit Time << Miss Penalty

The average memory access time can be calculated by the summation of

1. Read Hit contribution
2. Read Miss contribution
3. Write Hit contribution
4. Write Miss contribution

For cache time analysis, we require

1. Cache size
2. Associativity
3. Block size
4. Hit time
5. Miss penalty(Cycles)
6. Mem. write(Cycles)

C++ code:-

#include<iostream>

using namespace std;

#include<conio.h>

class s

{

public:

float a,b,c,e,f,g,h,i,k,l,z;

int d;

s(float a,float b)

{

this->a=a;

this->b=b;

}

void cacheinp()

{

cout<<"Cache Specification"<<endl;

cout<<"Cache Size= "<<a<<"KB"<<endl;

cout<<"Associativity = "<<b<<endl;

}

};

class t:public s

{

public:

t(float a,float b,float c,int d):s(a,b)

{

this->c=c;

this->d=d;

}

void cacheinp1()

{

cout<<"Block Size = "<<c<<endl;

if(d==1)

{

cout<<"Write Back"<<endl;

}

else

{

cout<<"Write Through"<<endl;

}

}

};

class u:public t

{

public:

u(float a,float b,float c,int d,float e,float f):t(a,b,c,d)

{

this->e=e;

this->f=f;

}

void cacheinp2()

{

t::cacheinp1();

cout<<"%writes="<<e<<endl;

cout<<"%Reads ="<<100-e<<endl;

cout<<"%Dirty ="<<f<<endl;

}

};

class v:public u

{

public:

v(float a,float b,float c,int d,float e,float f,float g,float h,float i):u(a,b,c,d,e,f)

{

this->g=g;

this->h=h;

this->i=i;

}

void cacheinp3()

{

u::cacheinp2();

cout<<"Hit Time= "<<h<<endl;

cout<<"Miss peanlty= "<<g<<endl;

}

};

class w:public v

{

public:

w(float a,float b,float c,int d,float e,float f,float g,float h,float i):v(a,b,c,d,e,f,g,h,i)

{}

void disp()

{

v::cacheinp3();

k=0.936,l=0.064;

cout<<"Hit Rate= "<<k<<endl;

cout<<"Miss Rate= "<<l<<endl;

if(d==1)

{

cout<<"Read Hit Contributition: "<<(((100-e)/100)\*k\*h)<<endl;

cout<<"Read Miss Contributition: "<<(((100-e)/100)\*l\*((g+h)+((f/100)\*g)))<<endl;

cout<<"Write Hit Contributition: "<<((e/100)\*k\*h)<<endl;

cout<<"Write Miss Contributition: "<<((e/100)\*l\*g)<<endl;

z=(((100-e)/100)\*k\*h)+(((100-e)/100)\*l\*((g+h)+((f/100)\*g)))+((e/100)\*k\*h)+((e/100)\*l\*g);

cout<<"Total Average Memory Access Time(Avg.Clocks Per Memory Access)= "<<z<<endl;

}

else

{

cout<<"Read Hit Contributition: "<<(((100-e)/100)\*k\*h)<<endl;

cout<<"Read Miss Contributition: "<<(((100-e)/100)\*l\*(g+h))<<endl;

cout<<"Write Hit Contributition: "<<((e/100)\*k\*i)<<endl;

cout<<"Write Miss Contributition: "<<((e/100)\*l\*i)<<endl;

z=(((100-e)/100)\*k\*h)+(((100-e)/100)\*l\*(g+h))+((e/100)\*k\*i)+((e/100)\*l\*i);

cout<<"Total Average Memory Access Time(Avg.Clocks Per Memory Access)= "<<z<<endl;

}

}

};

int main()

{

float a,b,c,e,f,g,h,i;

int d;

cout<<"Cache Time Analysis \n";

cout<<"Enter the cache size (2^0-2^8)kbytes \n";

cin>>a;

cout<<"Enter the Associativity(2^0-2^5)#sets \n";

cin>>b;

cout<<"Enter the Block Size (2^4-2^8)bytes \n";

cin>>c;

cout<<"Write Policy"<<endl;

cout<<"1)Write Back\n2)Write Through \n";

cin>>d;

cout<<"Enter no. of percentage Writes\n";

cin>>e;

cout<<"Enter no. of percentage Dirty Data\n";

cin>>f;

cout<<"Enter Miss Penalty(cycles)(range 1-99)\n";

cin>>g;

cout<<"Enter Hit Time(cycles)\n";

cin>>h;

cout<<"Enter Memory Write(cycles)\n";

cin>>i;

w w1(a,b,c,d,e,f,g,h,i);

w1.disp();

getch();

}

Input:-

Cache Size: =2

Associativity =4

Block Size=32

Write policy:-Write Back

% Writes=22

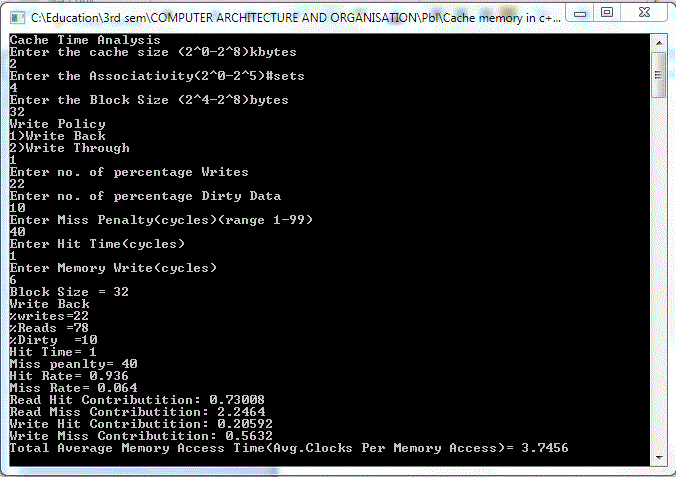
% Dirty Data=10

Miss Penalty (cycles) =40

 Hit Time (cycles) =1

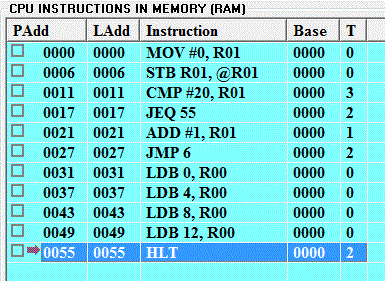
Mem. Write (cycles) =6

Output:-



**Using Simulator:-**

**Input(Instructions)**:-



The base address starts with 0000 and ends with 0055.

The instruction MOV does the operation of moving the number 0 into the register R01.

The instruction STB (Store a Byte) does the operation of storing the register value inside the memory.

The instruction CMP (Compare) compares the value 20 with the value inside the register R01.

The instruction JEQ (Jumps if equal to) jumps to the instruction 55 mentioned if the previous condition stated is satisfied.

The instruction ADD adds a value 1 to the value inside the register R01.

The instruction JMP jumps to the instruction 6 mentioned.

The instruction LDB (Loads a byte into the memory) does the operation of transferring the data present in data address 0 into the register R00.

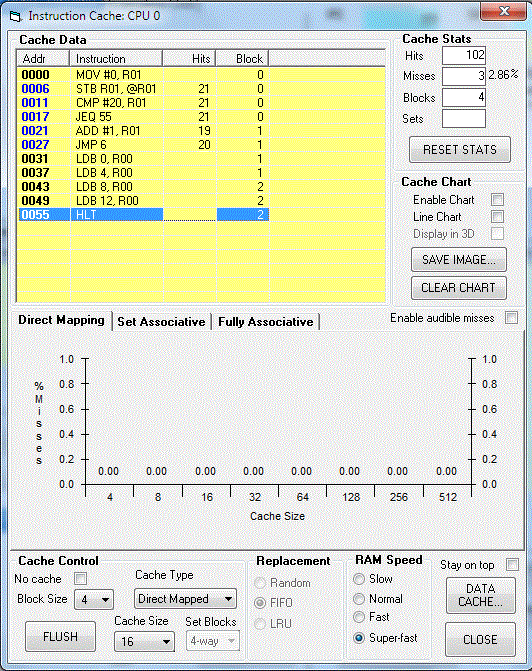
The instruction LDB (Loads a byte into the memory) does the operation of transferring the data present in data address 4 into the register R00.

The instruction LDB (Loads a byte into the memory) does the operation of transferring the data present in data address 8 into the register R00.

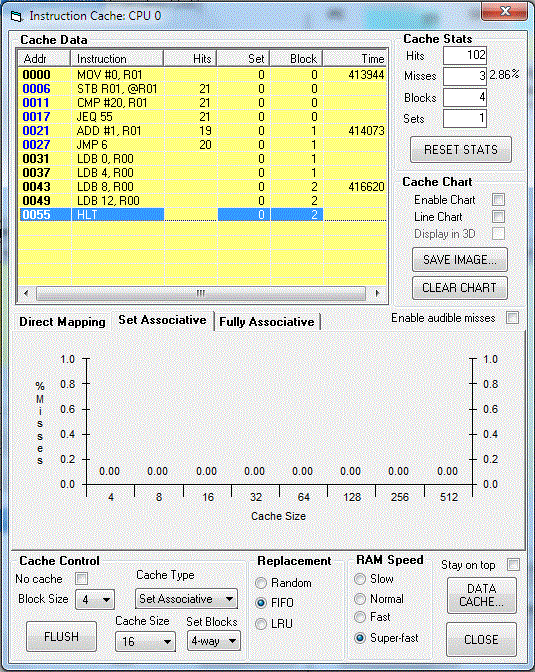
The instruction LDB (Loads a byte into the memory) does the operation of transferring the data present in data address 12 into the register R00.

The instruction HLT halts the program.

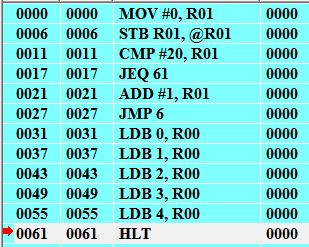
**Output(Direct mapping)**:-



**Output(4-Way Set associative mapping)**:-



Suppose the input is:



The addresses of instructions starts from 0000 and ends with 0061.

The instruction MOV does the operation of moving the number 0 into the register R01.

The instruction STB (Store a Byte) does the operation of storing the register value inside the memory.

The instruction CMP (Compare) compares the value 20 with the value inside the register R01.

The instruction JEQ (Jumps if equal to) jumps to the instruction 61 mentioned if the previous condition stated is satisfied.

The instruction ADD adds a value 1 to the value inside the register R01.

The instruction JMP jumps to the instruction 6 mentioned.

The instruction LDB (Loads a byte into the memory) does the operation of transferring the data present in data address 0 into the register R00.

The instruction LDB (Loads a byte into the memory) does the operation of transferring the data present in data address 1 into the register R00.

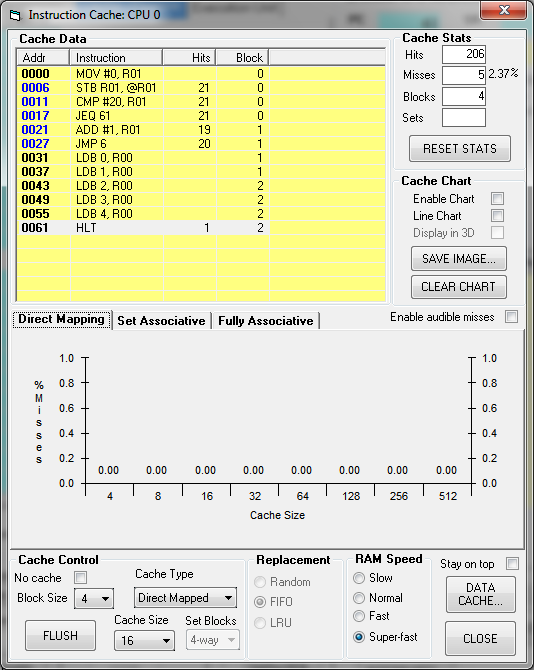
The instruction LDB (Loads a byte into the memory) does the operation of transferring the data present in data address 2 into the register R00.

The instruction LDB (Loads a byte into the memory) does the operation of transferring the data present in data address 3 into the register R00.

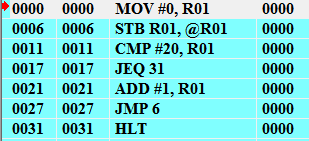
The instruction LDB (Loads a byte into the memory) does the operation of transferring the data present in data address 4 into the register R00.

The instruction HLT halts the program.

**Output:**



If the input is:



The addresses of instructions starts from 0000 and ends with 0031.

The instruction MOV does the operation of moving the number 0 into the register R01.

The instruction STB (Store a Byte) does the operation of storing the register value inside the memory.

The instruction CMP (Compare) compares the value 20 with the value inside the register R01.

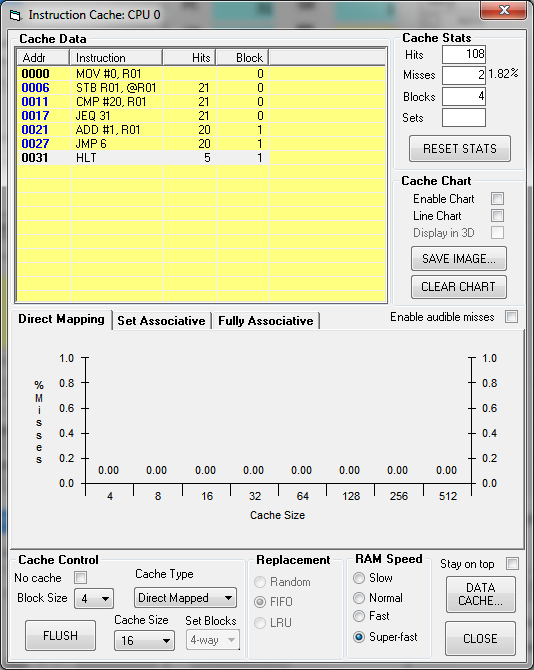
The instruction JEQ (Jumps if equal to) jumps to the instruction 31 mentioned if the previous condition stated is satisfied.

The instruction ADD adds a value 1 to the value inside the register R01.

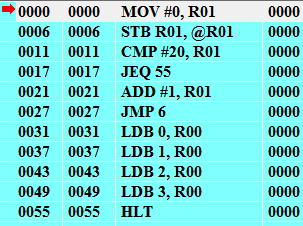
The instruction JMP jumps to the instruction 6 mentioned.

The instruction HLT halts the program.

**Output:**



If the input is:



The addresses of instructions starts from 0000 and ends with 0055.

The instruction MOV does the operation of moving the number 0 into the register R01.

The instruction STB (Store a Byte) does the operation of storing the register value inside the memory.

The instruction CMP (Compare) compares the value 20 with the value inside the register R01.

The instruction JEQ (Jumps if equal to) jumps to the instruction 55 mentioned if the previous condition stated is satisfied.

The instruction ADD adds a value 1 to the value inside the register R01.

The instruction JMP jumps to the instruction 6 mentioned.

The instruction LDB (Loads a byte into the memory) does the operation of transferring the data present in data address 0 into the register R00.

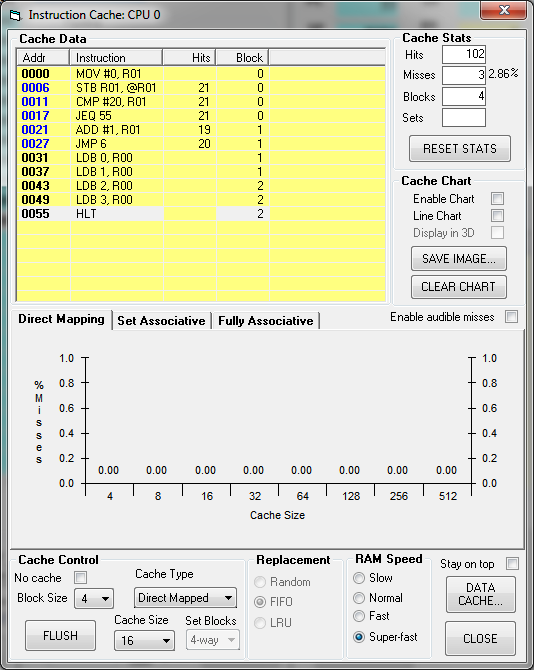
The instruction LDB (Loads a byte into the memory) does the operation of transferring the data present in data address 1 into the register R00.

The instruction LDB (Loads a byte into the memory) does the operation of transferring the data present in data address 2 into the register R00.

The instruction LDB (Loads a byte into the memory) does the operation of transferring the data present in data address 3 into the register R00.

The instruction HLT halts the program.

**Output:**



**Assembly code for Cache time Analysis:**

0000 MOV #2, R00 //Entering Cache size as 2 KB

0006 STB R00,@R00 //Storing the value of register inside the memory.

0011 MOV #4, R01 //Entering Cache Associativity in sets as 4

0017 STB R01,@R01 //Storing the value of register inside the memory.

0022 MOV #32, R02 //Entering Cache Block size as 32 Bytes

0028 STB R02,@R02 //Storing the value of register inside the memory.

0033 MOV #22, R04 //Entering %writes as 22

0039 STB R04,@R04 //Storing the value of register inside the memory.

0044 MOV #10, R05 //Entering %Dirty Data as 10

0050 STB R05,@R05 //Storing the value of register inside the memory.

0055 MOV #40, R06 //Entering Miss Penalty as 40

0061 STB R06,@R06 //Storing the value of register inside the memory.

0066 MOV #1, R07 //Entering Hit Time

0072 STB R07,@R07 //Storing the value of register inside the memory.

0077 MOV #6, R08 //Entering Memory Writes

0083 STB R08,@R08 //Storing the value of register inside the memory.

0088 MOV #0.936, R09 //Entering Hit Rate

0094 STB R09,@R09 //Storing the value of register inside the memory.

0099 MOV #0.064, R10 //Entering Miss Rate

0105 STB R10,@R10 //Storing the value of register inside the memory.

0110 MOV #100, R11 //Moving 100 into R11

0116 STB R11,@R11 //Storing the value of register inside the memory.

0121 SUB R04, R11 //R11=R11-R04

0126 DIV #100, R11 //R11=R11/100

0132 MOV R11, R12

0137 MUL R07, R11 //R11=R11\*R07

0142 MUL R09, R11 //R11=R11\*R09

0147 MOV R11, R13 //Read hit Contribution = R13

0152 MOV R05, R14 //R14=R05

0157 DIV #100, R14 // R14=R14/100

0163 MUL R06, R14 // R14=R14\*R06

0168 MOV R06, R15 // R15=R06

0173 ADD R07, R15 // R15 = R15+R07

0178 ADD R14, R15 // R15 = R15+R14

0183 MUL R10, R15 // R15 = R15\*R10

0188 MUL R12, R15 // R15 = R15\*R12, Read miss contribution = R15

0193 MOV R04, R16 // R16 = R04

0198 DIV #100, R16 // R16 = R16/100

0204 MOV R16, R17 // R17 = R16

0209 MOV R17, R22 // R22 = R17

0214 MOV R22, R23 // R23 = R22

0219 MUL R07, R16 // R16 = R16\*R07

0224 MUL R09, R16 // R16 = R16\*R09, Write hit contribution

0229 MUL R10, R17 // R17 = R17\*R10

0234 MUL R06, R17 // R17 = R17\*R06, Write miss contribution

0239 MOV R13, R18 // R18 = R13

0244 ADD R15, R18 // R18 = R18+R15

0249 ADD R16, R18 // R18 = R18+R16

0254 ADD R17, R18 // R18 = R18+R17, Total average access time for write back

0259 MOV R13, R19 // R19 = R13, Read hit contribution = R19

0264 MOV R12, R20 // R20 = R12

0269 MOV R07, R21 // R21 = R07

0274 ADD R06, R21 // R21 = R21+R06

0279 MUL R20, R21 // R21 = R21\*R20

0284 MUL R10, R21 // R21 = R21\*R10, Read miss contribution

0289 MUL R08, R22 // R22 = R22\*R08

0294 MUL R09, R22 // R22 = R22\*R09, Write hit contribution

0299 MUL R08, R23 // R23 = R23\*R08

0304 MUL R10, R23 // R23 = R23\*R10, Write miss contribution

0309 MOV R19, R24 // R24 = R19

0314 ADD R21, R24 // R24 = R24+R21

0319 ADD R22, R24 // R24 = R24+R22

0324 ADD R23, R24 // R24 = R24+R23, Total Memory Average Access time for Write Through

0329 MOV R18, R25 // R25 = R18

0334 HLT // Halts the program

Cache Analysis:

