***Министерство образования Республики Беларусь***

***Учреждение образования***

***«Брестский государственный технический университет»***

***Кафедра ИИТ***

**Лабораторная работа №7**

**По дисциплине АОИС за III семестр**

**Тема: «Кэш-память»**

**Выполнил:**

Студент группы ИИ-15 (1)

2-го курса

Волк И. А.

**Проверил:**

Михно Е. В.

Брест 2018

**Цель работы:** Изучить основные типы кэш-памяти, ознакомиться с алгоритмами размещения и восстановления данных, реализовать программную модель кэш-памяти.

**Задание:**

1. Изучить теоретический материал об основных типах кэш-памяти и алгоритмах ее функционирования.
2. Написать программу на ЯВУ, моделирующую кэш-память согласно [варианту](#_Варианты:).
3. Реализовать поиск данных по заданному физическому адресу в кэш-памяти с определением кэш-попадания либо кэш-промаха и возвратом необходимых данных.

|  |  |  |
| --- | --- | --- |
| № | Тип кэш-памяти | Размерность кэш-памяти  (количество кэш-строк на количество блоков (количество наборов)) |
| 21 | множественный ассоциативный кэш | 4 х 16 (2 набора) |

Код программы:

#include <iostream>

#include <array>

#include <vector>

#include <algorithm>

#define vector3d(T) std::vector<std::vector<std::vector<T>>>

namespace Lab7

{

const int **pow**(int x, const int deg)

{

int res = 1;

for (int i = 0; i < deg; i++)

res \*= x;

return res;

}

template<int N = 8>

class **BitWord**

{

std::array<bool, N> word;

public:

BitWord(int filler = 0)

{

for (int i = 0; i < N; i++)

{

word[N - i - 1] = (filler&pow(2, i)) >> i;

}

}

BitWord(std::initializer\_list<int> list)

{

int i = 0;

for (bool x : list)

if (i < word.size())

{

word[i] = x;

i++;

}

for (; i < word.size(); i++)

word[i] = false;

}

bool& operator[](const int id)

{

return word[id];

}

int toInt()

{

int res = 0;

for (int i = N - 1, deg = 0; i != 0; i--, deg++)

res += (word[i])?pow(2, deg):0;

return res;

}

int max()

{

return pow(2, N);

}

BitWord& operator=(const BitWord& other)

{

std::copy(other.word.begin(), other.word.end(), word.begin());

return \*this;

}

static int sizeInBitForm(int x)

{

int size = 0;

while (x != 1)

{

x /= 2;

size++;

}

return size;

}

};

template<int N>

std::ostream& **operator<<**(std::ostream& stream, BitWord<N> word)

{

for (int i = 0; i < N; i++)

stream << word[i];

return stream;

}

template<int AddressN = 8, int ValueN = 8>

class **RAM**

{

std::vector<BitWord<ValueN>> memory;

public:

RAM()

: memory(BitWord<AddressN>().max())

{}

BitWord<ValueN>& operator[](const int& id)

{

return memory[id];

}

};

template<int AddressN = 8, int ValueN = 8, int NBlocks = 2, int NLineInBlock = 4,

int NElementsInLine = 16>

class **CacheMemory**

{

typedef BitWord<AddressN> Address;

typedef BitWord<ValueN> Value;

vector3d(Value) memory;

std::array<int, NBlocks \* NLineInBlock> tags;

struct AddressInfo

{

int tag, block, intend;

AddressInfo(Address& address)

: tag(0), block(0), intend(0)

{

int sizeOfIntendId = Address::sizeInBitForm(NElementsInLine);

int sizeOfBlockId = Address::sizeInBitForm(NBlocks);

for (int i = AddressN - 1, deg = 0; i >= 0; i--, deg++)

{

if (address[i])

{

if (i >= AddressN - sizeOfIntendId)

intend += pow(2, deg);

else if (i >= AddressN - sizeOfIntendId - sizeOfBlockId)

{

if (i == AddressN - sizeOfIntendId - 1)

deg = 0;

block += pow(2, deg);

}

else

{

if (i == AddressN - sizeOfIntendId - sizeOfBlockId - 1)

deg = 0;

tag += pow(2, deg);

}

}

}

}

};

public:

CacheMemory()

{

std::fill(tags.begin(), tags.end(), -1);

memory.resize(NBlocks);

for (auto& i : memory)

{

i.resize(NLineInBlock);

for (auto& j : i)

j.resize(NElementsInLine);

}

}

bool get(Value& cell, Address& address)

{

AddressInfo info(address);

bool ok = false;

for (int i = 0; i < NLineInBlock; i++)

if (tags[info.block\*NLineInBlock + i] == info.tag)

{

cell = memory[info.block][i][info.intend];

ok = true;

break;

}

return ok;

}

void set(Value& cell, Address& address)

{

AddressInfo info(address);

bool added = false;

for(int i = 0; i < NLineInBlock; i++)

if (tags[info.block\*NLineInBlock + i] == -1)

{

memory[info.block][i][info.intend] = cell;

tags[info.block\*NLineInBlock + i] = info.tag;

added = true;

break;

}

if (!added)

{

int line = rand() % NLineInBlock;

memory[info.block][line][info.intend] = cell;

tags[info.block\*NLineInBlock + line] = info.tag;

}

}

};

template<int AddressN = 8, int ValueN = 8>

class **CacheControl**

{

typedef BitWord<AddressN> Address;

typedef BitWord<ValueN> Value;

CacheMemory<AddressN, ValueN> \* cache;

RAM<AddressN, ValueN> \* ram;

public:

CacheControl(CacheMemory<AddressN, ValueN> \* cache, RAM<AddressN, ValueN> \* ram)

: cache(cache), ram(ram)

{}

void set(Value&& cell, Address&& address)

{

ram->operator[](address.toInt()) = cell;

}

Value get(Address&& address)

{

Value result;

if (!cache->get(result, address))

{

result = ram->operator[](address.toInt());

std::cout << "Getting from RAM!" << std::endl;

cache->set(result, address);

}

else

std::cout << "Getting from cache-memory!" << std::endl;

return result;

}

};

} // Lab7

int **main**()

{

using namespace Lab7;

typedef BitWord<> Address;

typedef BitWord<> Value;

RAM<> ram;

CacheMemory<> cache;

CacheControl<> cu(&cache, &ram);

cu.set({ 0, 0, 0, 0, 0, 0, 0, 1 }, { 0, 0, 0, 0, 1, 1, 1, 1 }); // 1

cu.set({ 1, 0, 0, 0, 0, 0, 0, 0 }, { 0, 0, 1, 0, 1, 1, 0, 1 }); // 2

auto val = cu.get({ 0, 0, 0, 0, 1, 1, 1, 1 }); // 1

std::cout << val << std::endl;

val = cu.get({ 0, 0, 1, 0, 1, 1, 0, 1 }); // 2

std::cout << val << std::endl;

val = cu.get({ 0, 0, 0, 0, 1, 1, 1, 1 }); // 1

std::cout << val << std::endl;

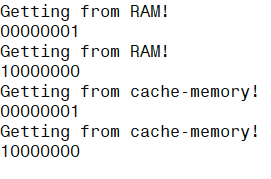
val = cu.get({ 0, 0, 1, 0, 1, 1, 0, 1 }); // 2

std::cout << val << std::endl;

std::cin.get();

}

Результат выполнения:



Вывод: Изучил основные типы кэш-памяти, ознакомиться с алгоритмами размещения и восстановления данных, реализовать программную модель кэш-памяти.