Федеральное государственное бюджетное образовательное учреждение высшего образования.
«Национально исследовательский университет «Московский энергетический институт»
Кафедра ВМСС

Лабораторная работа №6

ИССЛЕДОВАНИЕ ПРИНЦИПОВ РАБОТЫ КЭШ-ПАМЯТИ

Курс: Вычислительные системы

Группа: А-08-19

Выполнил: Балашов С.А.

Проверил: Карпов А.В.

Москва 2023 г.

Домашняя подготовка

Цель работы: изучение основных принципов работы кэш-памяти, знакомство с алгоритмами замещения строк кэш-памяти, рассмотрение понятия ассоциативности кэш-памяти.

Таблица 1

Вариант для выполнения домашней подготовки

№ Бригады	Разрядность системы	Размер кэш-строки (в байтах)	Кол-во кэш-строк	Степень ассоциативности	Последовательность байт	Размер ОЗУ
4	8	64	4	4	CCC, 1234, 2, 2000, 79A, AAB	8 КБ

1. Изучить соответствующие разделы лекционного курса и описания лабораторных работ.

2. Исследование кэша прямого отображения. **Степень ассоциативности** для данного пункта **игнорировать**. Рассчитать **полезный** размер кэш-памяти исходя из данных, представленных в таблице 1. Схематично представить структуру заданной кэш-памяти (на схеме должны быть отражены: **размер** и **количество кэш-строк**, **тег**, соответствующий каждой кэш-строке). Определить возможное расположение заданной в таблице 1 последовательности байт в кэш-памяти при прямом отображении ОЗУ в кэш-память. Рассчитать вероятность коллизии в случае обращения к ячейкам ОЗУ, адреса которых отличаются на N, где N – полезный размер кэш-памяти.

Полезный размер кэш памяти: $S_{\text{п}} = C * L = 4 * 64 = 256$ (байт).

Теги строк представим в виде таблицы 2.

№ кэш-строки	Тег	
	Десятичная СС	Шестнадцатеричная СС
1	0	00
2	64	40
3	128	80
4	192	C0

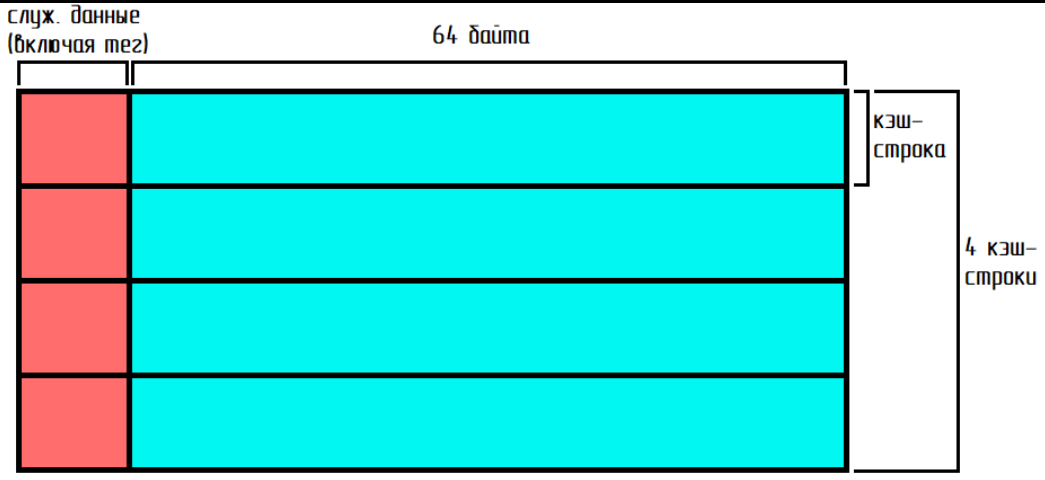

Рис. 1. Структура кэш-памяти

Определим индексы кэш-строк для заданной последовательности байт (см. таблицу 1): $\boldsymbol{i = \frac{ADR_6}{L} mod\ C}$.

Результаты вычислений занесем в таблицу 3.

Таблица 3

Индексы кэш-строк для последовательности байт

Индекс кэш-строки	Адрес байта в ОЗУ	
	Шестнадцатеричная СС	Десятичная СС
3	CCC	3 276
0	1234	4660
0	2	2
0	2000	8192
2	79A	1946
2	AAB	2731

Рассчитаем вероятность коллизии: $\boldsymbol{P}_{\text{к}} = \mathbf{1} - (\boldsymbol{N}_{\text{д}}/\boldsymbol{N}_{\text{отб}}) = \mathbf{1} - \left(\frac{\mathbf{1}}{\mathbf{64}}\right) = 0{,}984.$

3. Исследование понятия ассоциативности. Рассчитать **полезный** размер кэш-памяти исходя из данных, представленных в таблице 1 **с учётом числа банков**. Схематично представить структуру заданной кэш-памяти (на схеме должны быть отражены: **количество банков** и **полезный размер** кэш-памяти в банке). Рассчитать **вероятность коллизии** в случае обращения к ячейкам ОЗУ, **адреса** которых **отличаются на N**, где N – полезный размер кэш-памяти. Сравнить с результатом, полученным в пункте 2.

Полезный размер кэш памяти с учетом числа банков: $S_{\text{п}} = C * L * N = 4 * 64 * 4 = 1024$ (байт).

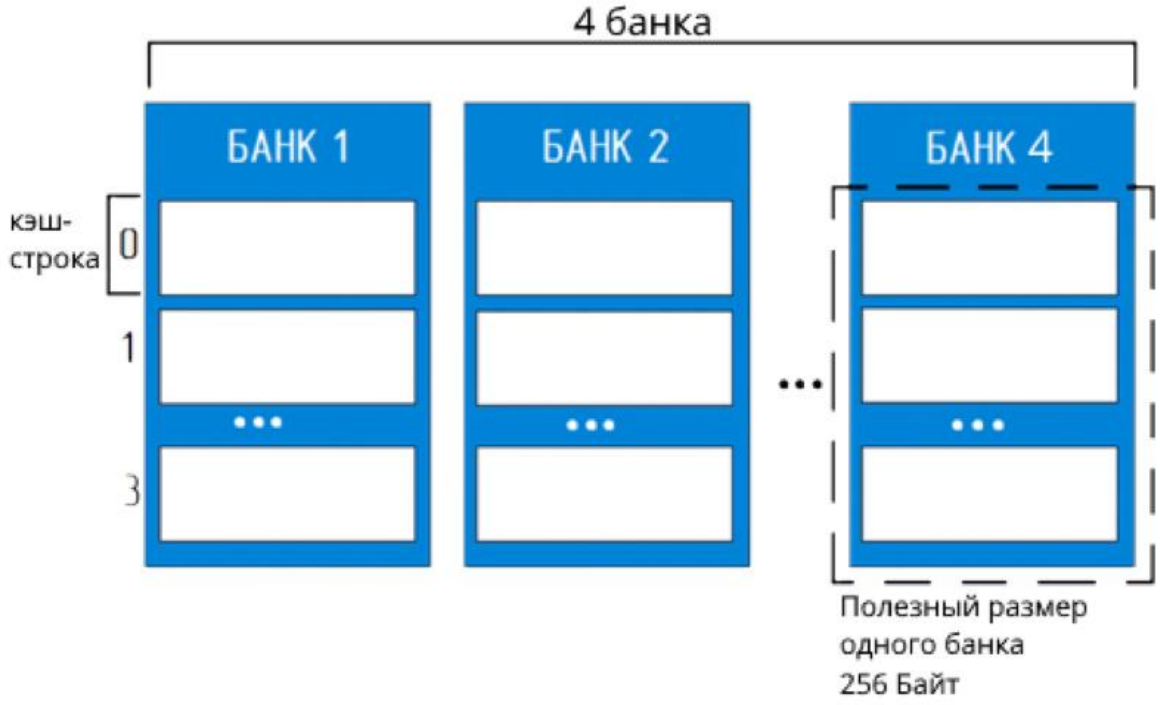

Рис. 2. Структура кэш-памяти

Рассчитаем вероятность коллизии: $\boldsymbol{P}_{\text{к}} = \mathbf{1} - (\boldsymbol{N}_{\text{д}}/\boldsymbol{N}_{\text{отб}}) = \mathbf{1} - \left(\frac{\mathbf{8}}{\mathbf{64}}\right) = 0{,}875.$

Вывод: с увеличением степени ассоциативности кэш-памяти увеличивается её полезный размер, также снижается вероятность коллизии (за счет увеличения числа строк кэш-памяти, в которые может быть отражены несколько претендующих на это строк из ОЗУ).

4. Описать процесс замещения строк кэш-памяти при последовательном обращении к набору байт из пункта 2 в случае, когда кэш-память уже заполнена данными (лежащими в ОЗУ по адресу от 0 до полезного размера кэш-памяти, учитывая степень ассоциативности). **Для группы А-08** алгоритм **MFU** (Most Frequently Used) замещения строк кэш-памяти.

Представим начальное состояние кэш-памяти в форме таблицы 4.

Таблица 4

Начальное состояние кэш-памяти

Индекс кэш-строки	**Банк**							
	1		**2**		**3**		**4**	
0	0	1	256	5	512	3	768	1
1	64	2	320	2	576	1	832	1
2	128	3	384	1	640	2	896	5
3	192	4	448	1	704	4	960	2

Алгоритм MFU заключается в том, что, при необходимости замещения, замещается та кэш-строка, к которой было совершено наибольшее кол-во обращений за последнее время. В таком случае, при обращении к байту **3 276** (**CCC** в 16-ричной СС), будет заменена **3я строка 4го банка**, так как из всех строк с индексом 2 к ней обращались наибольшее кол-во раз (5). Строка с байтом **16000** (**3E80**) заменит **2ю строку 7го банка**, **15872** (**3E00**) – **0ю строку 5го банка**, при обращении к байту **62** (**3E**) никакая из кэш-строк не будет замещена (уже лежит в кэш-памяти), **15312** (**3BD0**) – **3ю строку 5го банка**, **255** (**FF**) – кэш-строка замещена не будет.

Таблица 4

Конечное состояние кэш-памяти

Индекс кэш-строки	**Банк**							
	1		**2**		**3**		**4**	
0	0	2	4608	1	8192	1	768	1
1	64	2	320	2	576	1	832	1
2	2688	1	384	1	640	2	1920	1
3	3200	1	448	1	704	4	960	2

Лабораторное задание

1. Написать программу на языке С, позволяющую отследить изменение в скорости отклика получения данных при явлении **thrashing**'а (**пробуксовке кэша**).

2. На основе данных, полученных в пункте 1, **построить график**, **определить** по нему **размер** кэша **L1** и **L2**.

Проанализировать полученные результаты и **сделать вывод**.

Программа:

```
#include	<stdio.h>
#include	<stdlib.h>
#include	<string.h>
#include	<time.h>

#define		T	char
#define		MAX_S	0x1000000
#define		L	101

volatile T A[MAX_S];
int m_rand[0xFFFFFF];

int trash (){
	static struct timespec t1, t2;

	memset ((void*)A, 0, sizeof (A));

	srand(time(NULL));

	int v, M;
	register int i, j, k, m, x;

	for (k = 1024; k < MAX_S;) {
		M = k / L;

		printf("%g\t", (k+M*4)/(1024.*1024));

		for (i = 0; i < M; i++) m_rand[i] = L * i;
		for (i = 0; i < M/4; i++)	{
			j = rand() % M;
			x = rand() % M;

			m = m_rand[j];
			m_rand[j] = m_rand[i];
			m_rand[i] = m;

		}

		if (k < 100*1024) j = 1024;
		else if (k < 300*1024) j = 128;
		else j = 32;

		clock_gettime (CLOCK_PROCESS_CPUTIME_ID, &t1);
		for (i = 0; i < j; i++) {

			for (m = 0; m < L; m++) {
				for (x = 0; x < M; x++){
					v = A[ m_rand[x] + m ];
				}
			}

		}
		clock_gettime (CLOCK_PROCESS_CPUTIME_ID, &t2);

		printf ("%g\n",1000000000. * (((t2.tv_sec + t2.tv_nsec * 1.e-9) - (t1.tv_sec + t1.tv_nsec * 1.e-
9)))/(double)(L*M*j));

		if (k > 100*1024)	k += k/16;
		else			k += 4*1024;
	}
	return 0;
}

int main(){
	printf("Counting...\n");
    trash();
    printf("Done!\n");
    return 0;
}
```

```
#include    <stdio.h>
#include    <stdlib.h>
#include    <string.h>
#include    <time.h>

#define     T   char
#define     MAX_S   0x1000000
#define     L   101

volatile T A[MAX_S];
int m_rand[0xFFFFFF];

int trash (){
    static struct timespec t1, t2;

    memset ((void*)A, 0, sizeof (A));

    srand(time(NULL));

    int v, M;
    int i, j, k, m, x;

    for (k = 1024; k < MAX_S;) {
        M = k / L;

        printf("%g\t", (k*M*4)/(1024.*1024));

        for (i = 0; i < M; i++) m_rand[i] = L * i;
        for (i = 0; i < M/4; i++)   {
            j = rand() % M;
            x = rand() % M;
```

```
milord@World:~$ cd WORK
milord@World:~/WORK$ g++ forLinux.cpp -o forLinux
milord@World:~/WORK$ ./forlinux
-bash: ./forlinux: No such file or directory
milord@World:~/WORK$ ./forLinux
Counting...
0.00101471      1.61094
0.00507355      1.83733
0.0091362       2.14643
0.013195        2.04325
0.0172577       2.03865
0.0213165       2.0406
0.0253792       2.01763
0.0294418       2.01671
0.0335007       2.0095
0.0375633       2.01414
0.0416222       2.02026
0.0456848       2.0259
0.0497437       2.04064
0.0538063       2.03527
0.0578651       2.03378
0.0619278       2.0363
0.0659904       2.03434
0.0700493       2.03629
0.0741119       2.03557
0.0781708       2.038
0.0822334       2.03695
0.0862923       2.03678
0.0903549       2.03797
0.0944138       2.03798
0.0984764       2.04057
0.102539        2.0418
0.106948        2.03255
0.115757        2.03788
0.122991        2.03598
0.130678        2.03877
0.138843        2.03816
0.147521        2.03943
0.15674 2.03823
```

Рис. 3. Результат работы программы

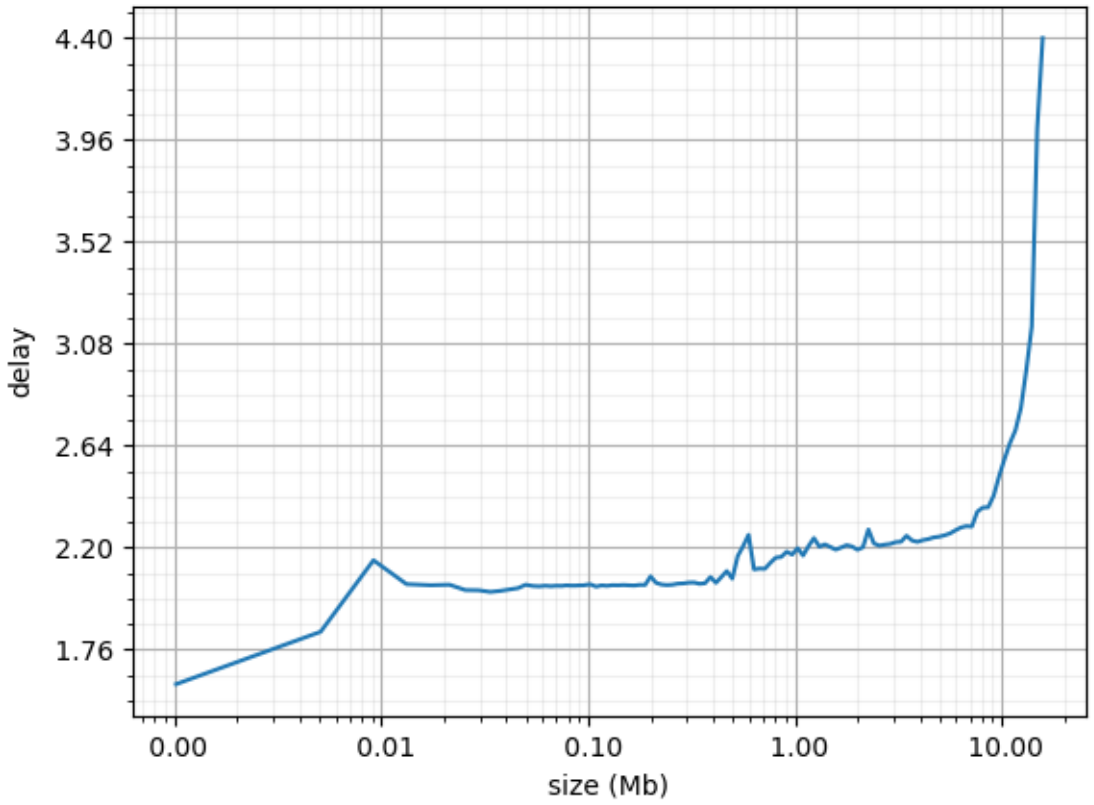

Рис. 4. График по результатам работы программы (в логарифмическом масштабе)

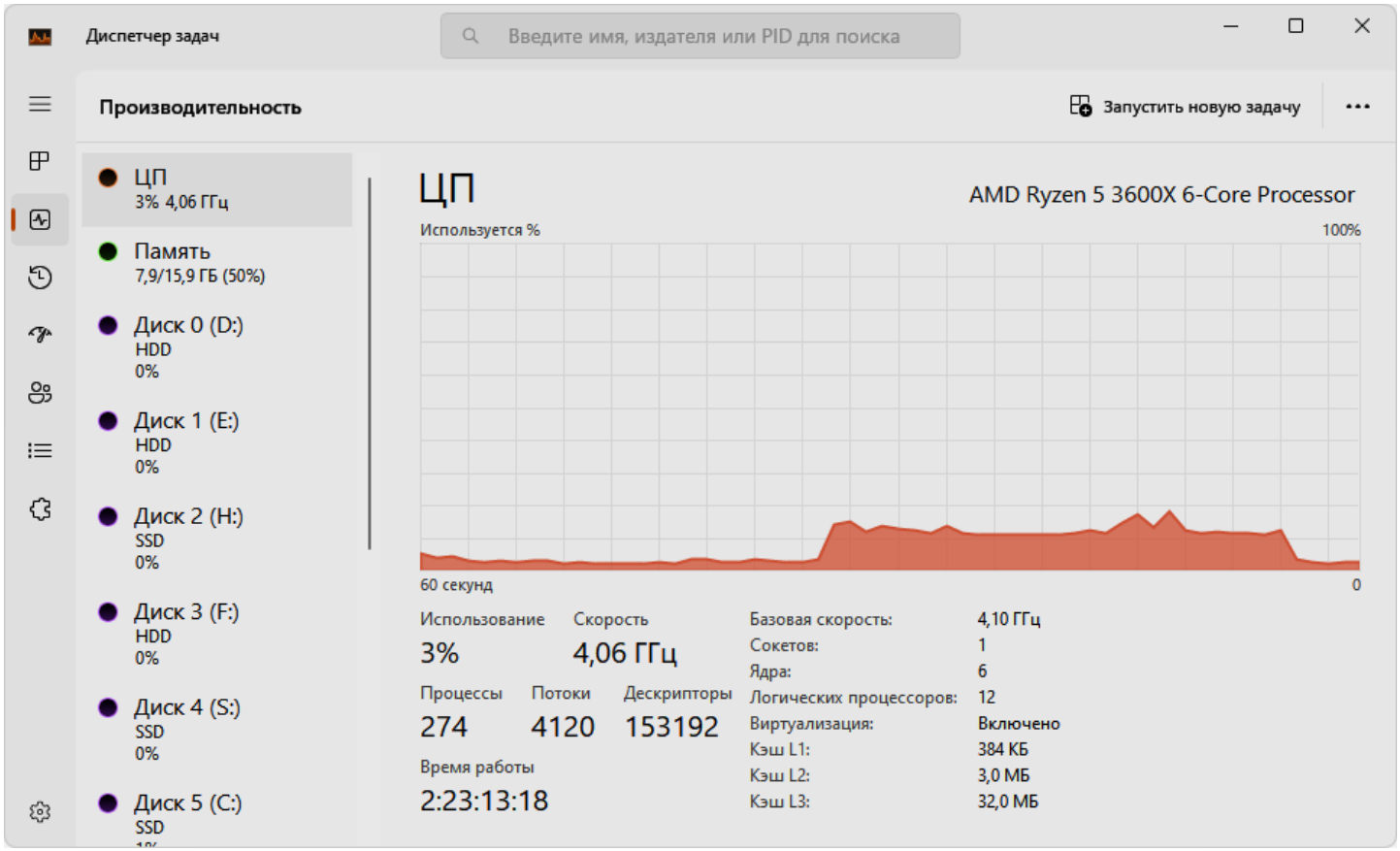

Рис. 5. Реальные размеры кэша L1 и L2

Вывод: скачки на графике иллюстрируют переход от кэша более низкого уровня к кэшу более высокого уровня. Небольшие переходы на графике означают хорошую оптимизацию кэша у данного процессора. Первый пик можно считать шумом других процессов при старте программы. По графику видно, что первый скачок происходит примерно при 0,04 МБ или 40 КБ. Умножим эту величину на количество ядер (см. Рис. 5) и получим размер кэша L1: 40*6=240 КБ. Следующий скачок наблюдается при 0,5 МБ. Также умножим на кол-во ядер и получим размер кэша L2: 0,5*6=3 МБ. Сравнивая результаты с реальными размерами кэша, можно заметить достаточно высокую точность (совпадение размерностей значений) программного нахождения объема кэш-памяти.