МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №1

по курсу «Архитектура современных микропроцессоров и мультипроцессоров»

ОЦЕНКА ПРОИЗВОДИТЕЛЬНОСТИ МИКРОПРОЦЕССОРА НА ЗАДАННЫХ ОПЕРАЦИЯХ

Выполнил: студент 3-го курса гр. 17208

Гафиятуллин А.Р

Новосибирск, 2020

1. ЦЕЛИ РАБОТЫ:

1.1. научиться оценивать производительность микропроцессора на заданных операциях.

2. ХОД РАБОТЫ:

2.1. Написана программа, выполняющая многократно (в цикле) операцию целочисленного деления:

2.1.1. с использованием последовательности зависимых операций (оценка латентности);

2.1.2. с использованием последовательности независимых операций (оценка темпа выдачи результатов).

Листинг программы:

```
1. #include <stdio.h>
2. #include <time.h>
3. #include <x86intrin.h>
4.
5. #ifndef WARMING_UP_SECONDS
6. #define WARMING_UP_SECONDS 10
7. #endif
8.
9. #ifndef CYCLE_NUM
10.   #define CYCLE_NUM 100000000
11.   #endif
12.
13.   int main() {
14.     unsigned long long start = 0;
15.     unsigned long long end = 0;
16.
17.     unsigned x = 1 << 31;
18.     unsigned y = 2;
19.     unsigned result = 0;
20.
21.     double clocks_per_empty_iter = 0;
22.
23.     // warming up a processor
24.     time_t start_time = time(NULL);
25.     while (time(NULL) - start_time < WARMING_UP_SECONDS) {
26.       result = x / y;
27.     }
28.     printf("Processor is warmed up! result = %d\n", result);
29.
30.     // count cycle clocks
31.     int i;
32.     start = __rdtsc();
33.     for(i = 0; i < CYCLE_NUM; i++) {}
34.     end = __rdtsc();
35.     clocks_per_empty_iter = (double)(end - start) / i;
36.     printf("Amount of clocks per empty iteration: %lf\n",
   clocks_per_empty_iter);
37.
```

```
38.     //---------------------------independent operations-------------------
39.     // count clocks for cycle with divisions
40.     start = __rdtsc();
41.     for(i = 0; i < CYCLE_NUM; i++) {
42.       result = x / y;
43.     }
44.     end = __rdtsc();
45.     printf("Amount of clocks per iteration with division: %lf, result =
   %d\n\n",
46.         (double)(end - start) / i, result);
47.
48.     // real division clocks
49.     printf("Division clocks for independent operations: %lf\n\n",
50.         (double)(end - start) / i - clocks_per_empty_iter);
51.     //---------------------------Dependent operations--------------------
52.     x = 1 << 31;
53.
54.     start = __rdtsc();
55.     for(i = 0; i < CYCLE_NUM; i++) {
56.       x = x / y;
57.     }
58.     end = __rdtsc();
59.     printf("Amount of clocks per iteration with division: %lf, result =
   %d\n\n",
60.         (double)(end - start) / i, x);
61.
62.     // real division clocks
63.     printf("Division clocks for dependent operations: %lf\n\n",
64.         (double)(end - start) / i - clocks_per_empty_iter);
65.
66.     return 0;
67. }
```

Исходный код был скомпилирован с ключом -O0, далее был проанализирован ассемблерный код, генерируемый компилятором. Во всех циклах, которые были важны для оценки производительности, обращения к памяти на стеке были заменены на обращения к регистрам. Была так же произведена раскрутка циклов, чтобы уменьшить влияние накладных расходов на организацию цикла на конечные результаты оценки производительности операции целочисленного деления. Перед началом тестов был произведен «прогрев» процессора, чтобы достичь стабильной частоты.

Ассемблерные листинги данных циклов:

- цикл для оценки числа тактов на итерацию холостого цикла:

```
#------------------------------ empty cycle --------------------------------
    movl    $0, %ecx
    jmp .L5
.L6:
    addl    $1, %ecx
.L5:
```

```
    cmpl    $99999999, %ecx
    jle .L6
    movl    %ecx, -40(%rbp)
#------------------------------ empty cycle ------------------------------
```

- цикл для оценки темпа выдачи:

```
#------------------------------ independent operations ---------------------
    movl    $0, %ecx
    movl    -48(%rbp), %edi
    movl    -36(%rbp), %ebx
    jmp .L11
.L12:
    movl    %edi, %eax
    movl    $0, %edx
    divl    %ebx

    movl    %edi, %eax
    movl    $0, %edx
    divl    %ebx

    movl    %edi, %eax
    movl    $0, %edx
    divl    %ebx

    movl    %edi, %eax
    movl    $0, %edx
    divl    %ebx

    addl    $4, %ecx
.L11:
    cmpl    $2499999, %ecx
    jle .L12
    movl    %eax, -44(%rbp)
    movl    %ecx, -40(%rbp)
#------------------------------ independent operations ---------------------
```

- цикл для оценки латентности:

```
#------------------------------ dependent operations ----------------------
    movl    $0, %ecx
    movl    -48(%rbp), %eax
    movl    -36(%rbp), %ebx
    jmp .L19
.L20:
    movl    $0, %edx
    divl    %ebx

    movl    $0, %edx
    divl    %ebx

    movl    $0, %edx
    divl    %ebx

    movl    $0, %edx
    divl    %ebx

    addl    $4, %ecx
.L19:
    cmpl    $2499999, %ecx
```

```
    jle .L20
    movl    %eax, -48(%rbp)
    movl    %ecx, -40(%rbp)
#------------------------------ dependent operations ----------------------
```

2.2. Оценки производительности (в тактах):

	Intel Core i5 7200U Skylake-X (2.5 GHz, 3.1 GHz Turbo Boost)	Intel Core i7 9700F Coffee Lake (3.0 GHz, 4.7 GHz Turbo Boost)	Agner Fog (Skylake-X)
Пустой цикл	1.757505	1.314142	
Темп выдачи	5.484572	4.091744	6
Темп выдачи с поправкой на Turbo Boost	**6.800869**	**6.410398**	
Латентность	20.271018	15.632926	23
Латентность с поправкой на Turbo Boost	**25.136062**	**24.491584**	

Можно заметить, что при учитывании режима Turbo Boost полученные результаты очень близки к тем, что получил Агнер Фог.
Документация Intel не дает своих оценок для операции целочисленного деления, указывая лишь то, что при различном количестве значащих битов в операндах количество тактов, как для латентности, так и для темпа выдачи может существенно различаться.

3. ВЫВОДЫ:

3.1. научились оценивать производительность микропроцессора на заданных операциях;
3.2. узнали про отличия латентности и темпа выдачи, как характеристик микропроцессорных инструкций;
3.3. написали программу для оценки латентности и темпа выдачи для операции целочисленного деления на x86-64 процессоре;
3.4. поработали с ассемблерными листингами программы;
3.5. получили оценки латентности и темпа выдачи для операции целочисленного деления на двух компьютерах и сравнили эти результаты с другими оценками;
3.6. нужно стараться писать программы с небольшими информационными зависимостями, чтобы операции работали как можно быстрее.