МИНИСТЕРСТВО ТРАНСПОРТА РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ

ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»

(РУТ (МИИТ)

Институт транспортной техники и систем управления

Кафедра «Управление и защита информации»

Отчет по лабораторной работе №3

«Команды передачи управления»

по дисциплине

«Машинно-ориентированные языки программирования»

Выполнил: студент ТКИ-341 Козлов А. Д.

Проверили: Доцент “УиЗИ” Логинова Л.Н.

Зав. лаборатории “УиЗИ” Антонов Д. А.

Москва 2024 г.

Цель работы: изучение команд передачи управления и получение навыка работы с ними.

Постановка задачи:

1. С помощью процедуры *Random* сформировать случайное целое число в диапазоне от 0 до 65535;
2. Четное число (четное с точки зрения математики) записать в ЯП (ячейку памяти), начиная с нулевого элемента массива *data*. Нечетное число (нечетное с точки зрения математики) записать в ЯП, начиная с 128 элемента массива *data*. Все числа, которые больше или равны 50000 записать в ЯП, начиная с 256 элемента массива *data*. Все числа, которые меньше 10000 записать в ЯП, начиная с 384 элемента массива *data*;
3. Повторять п.1-п.2, пока число распределенных данных по любому из признаков не превысит 127.

Код программы:

```
#include<iostream>
#include<ctime>

using namespace std;
const int DataSize = 512;

int main()
{
    srand(time(NULL));

    unsigned short data[DataSize] = { NULL };

    int32_t even_counter = 0;
    int32_t odd_counter = 0;
    int32_t more_50000_counter = 0;
    int32_t less_10000_counter = 0;

    _asm {
        PUSHAD

        main_loop :
            CALL rand
            MOV ESI, EAX
            MOV EBX, 2
            xor EDX, EDX
            DIV EBX
            CMP EDX, 0
            JE even_num_proc
            JNE odd_num_proc
            JMP main_loop_exit

        even_num_proc :
            MOV ECX, 0
            LEA EBX, even_counter
            MOV EDX, DWord Ptr[EBX]
            CALL add_num_in_data
            CALL inc_counter_proc
            JMP more_or_less_proc

        odd_num_proc :
            MOV ECX, 128
            LEA EBX, odd_counter
            MOV EDX, DWord Ptr[EBX]
            CALL add_num_in_data
            CALL inc_counter_proc
            JMP more_or_less_proc

        more_or_less_proc :
            CMP EAX, 50000
            JAE more_50000_proc
            CMP EAX, 10000
            JBE less_10000_proc
            JMP couters_reach_127_proc

        more_50000_proc :
            MOV ECX, 256
            LEA EBX, more_50000_counter
            MOV EDX, DWord Ptr[EBX]
            CALL add_num_in_data
            CALL inc_counter_proc
            JMP couters_reach_127_proc
```

```
    less_10000_proc :
        MOV ECX, 384
        LEA EBX, less_10000_counter
        MOV EDX, DWord Ptr[EBX]
        CALL add_num_in_data
        CALL inc_counter_proc
        JMP couters_reach_127_proc

    add_num_in_data
        ADD ECX, EDX
        LEA EDI, data
        MOV Word Ptr[EDI + ECX * 2], SI
        RET

    inc_counter_proc :
        INC EDX
        MOV DWord Ptr[EBX], EDX
        RET

    couters_reach_127_proc :
        LEA EBX, even_counter
        CALL current_counter_reach_127_proc
        JAE main_loop_exit

        LEA EBX, odd_counter
        CALL current_counter_reach_127_proc
        JAE main_loop_exit

        LEA EBX, more_50000_counter
        CALL current_counter_reach_127_proc
        JAE main_loop_exit

        LEA EBX, less_10000_counter
        CALL current_counter_reach_127_proc
        JAE main_loop_exit

        JMP main_loop


    current_counter_reach_127_proc :
        MOV EDX, DWord Ptr[EBX]
        CMP EDX, 127
        RET


    main_loop_exit :

    POPAD
}

int i = 0, j = i + even_counter;
cout << "Numbers [0 : 127] (count:" << even_counter << "): ";
for (i; i < j; i++) {
    cout << data[i] << " ";
}
cout << "\n" << endl;

i = 128, j = i + odd_counter;
cout << "Numbers [128 : 255] (count:" << odd_counter << "): ";
for (i; i < j; i++) {
    cout << data[i] << " ";
}
cout << "\n" << endl;
```

```
    i = 256, j = i + more_50000_counter;
    cout << "Numbers [256 : 383] (count:" << more_50000_counter << "): ";
    for (i; i < j; i++) {
        cout << data[i] << " ";
    }
    cout << "\n" << endl;

    i = 384, j = i + less_10000_counter;
    cout << "Numbers [384 : 511] (count:" << less_10000_counter << "): ";
    for (i; i < j; i++) {
        cout << data[i] << " ";
    }
    cout << "\n" << endl;

    return 0;
}
```

Ход выполнения:

1. Метка main_loop используется для создания цикла генерации псевдослучайных чисел, после этой метки мы создаем число и по условным переходам выясняем к какой группе относится число.
2. Распределение по категориям происходит в два этапа: первый этап – это выяснение четности и нечетности числа (переходы по меткам `even_num_proc` и `odd_num_proc`).
3. Затем в каждой метке происходит переход к проверке <= 11000 и => 50000 (переход к метке `more_or_less_proc`). После проверки условий переход осуществляется к одной из меток условия (`more_50000_proc` и `less_10000_proc`).
4. При выполнении условия мы должны записать число в память и увеличить счетчик категории, за это отвечает код после меток `add_num_in_data` и `inc_counter_proc`.
5. После проверки всех условий мы производим проверку достижения счетчика хотя бы в одной группе 128 элементов. Для этого после мы переходим на метку `couters_reach_127_proc` если условие выполняется, программа переходит на метку main_loop_exit, которая ведет к завершению программы. Мы выводим данные из памяти и видим следующий результат:

```
Numbers [0 : 127] (count:106): 25948 30808 31262 7226 25022 12144 28970 23544 24006 9744 7554 13908
24410 29966 16966 8402 12990 27214 3116 16178 3922 17948 9652 20598 13116 2570 31536 6432 20030 7172
 16434 24522 1004 31034 10778 4418 4382 5768 9100 8996 1874 19612 16476 24656 24046 22000 8144 3442
30586 31268 30452 15850 19332 8924 32102 20246 6220 28720 26310 21190 13282 4132 7502 29168 14100 90
30 12558 3010 26418 1194 29172 24226 16716 18936 11882 5224 15456 15054 4082 17646 22452 23904 22736
 11734 19070 3476 28382 22818 2376 19548 29672 23312 5804 18888 598 27868 8942 570 16170 27232 12392
 18646 16190 12932 10684 22850

Numbers [128 : 255] (count:106): 335 17121 21959 18159 14923 30477 13613 26451 19421 14845 29591 323
9 9159 20083 8527 28331 25273 14105 20993 12379 3189 3721 24653 4409 18811 12483 24893 30065 28629 3
2087 11549 31387 19323 3281 21627 20425 10263 3465 22567 17825 27443 10033 2545 31367 30807 10731 27
743 21577 27361 5115 12765 3879 41 18801 6615 8747 29895 2813 6065 2163 30803 25849 25945 13917 2025
 21171 31075 849 3165 1263 8681 31221 20095 21191 10289 31773 13593 27025 29299 22985 17399 24103 25
971 11263 11287 17035 17703 3287 17957 25915 1155 22573 27003 31795 8489 1909 3609 12177 3857 7621 2
0561 21499 9727 31835 3887 567

Numbers [256 : 383] (count:0):

Numbers [384 : 511] (count:127): 335 17121 7226 18159 12144 14923 9744 7554 13613 19421 14845 13908
3239 16966 9159 8402 12990 8527 3116 16178 3922 17948 9652 13116 2570 6432 14105 12379 3189 7172 164
34 3721 1004 4409 10778 4418 4382 18811 12483 5768 9100 8996 1874 11549 19612 16476 19323 3281 8144
3442 10263 3465 17825 15850 19332 8924 10033 2545 6220 10731 5115 12765 3879 41 18801 13282 4132 750
2 6615 8747 14100 2813 9030 12558 3010 6065 2163 1194 13917 2025 16716 18936 11882 5224 15456 15054
4082 17646 11734 19070 3476 849 3165 1263 8681 2376 19548 10289 13593 5804 18888 17399 598 11263 112
87 17035 17703 3287 8942 17957 570 16170 1155 12392 8489 1909 18646 3609 12177 3857 7621 16190 12932
 9727 10684 3887 567
```

Вывод:

1. В ходе выполнения лабораторной работы были изучены команды передачи управления языка ассемблера, такие как JMP, JCC, CALL, RET.
2. Были изучены команды CMP, CMPXCHG и CMPXCHG8B для сравнения данных между собой.
3. Также была изучена группа команд LOOP, которые позволяют создавать циклы со счетчиком.
4. Было продемонстрированно как с помощью команд передачи управления можно реализовать циклы, а с помощью связки команд CMP и JCC реализовать функционал, похожий на условные операторы в языках более высокого уровня.
5. Выяснили, что при работе командой CALL требуется обязательно использовать в конце команду RET, так как при ее отсутствии в регистре EPI будут храниться невалидные данные.

Заключение

Выполнение лабораторной работы расширило знание команд языка ассемблера. Мы реализовали конструкции такие, как циклы и условные переходы, необходимые для построения более сложных программ. Также научились выделять повторяющиеся части кода в отдельные процедуры и переиспользовать их. Полученные навыки будут необходимы при разработке программ со сложной логикой и ветвлением.