

Министерство науки и высшего образования Российской Федерации
Федерального государственное бюджетное образовательное учреждение
высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра комплексной информационной безопасности электронно-
вычислительных систем (КИБЭВС)

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ АССЕМБЛЕР

Отчет по лабораторной работе №2
по дисциплине «Системное программирование»

Выполнил

Студент гр. 738-1

_____ Вдовина И.Е.

_____.____.2022

Принял

М.н.с. ИСИБ

_____ Калинин Е. О.

_____.____.2022

1 Введение

Цель работы: познакомиться со структурой программы на языке Ассемблер, разновидностями и назначением сегментов, способами организации простых и сложных типов данных, изучить форматы и правила работы с транслятором FASM, компоновщиком и отладчиком CV, познакомиться со средой программирования RadASM, возможностями Visual Studio для работы с Ассемблером и средствами создания программ на Ассемблере для ОС Linux.

Вариант: 3.

2 краткая теоретическая информация

Ассемблеры, как правило, специфичны конкретной архитектуре, операционной системе и варианту синтаксиса языка. Вместе с тем существуют мультиплатформенные или вовсе универсальные ассемблеры, которые могут работать на разных платформах и операционных системах.

Среди последних можно также выделить группу кросс-ассемблеров, способных собирать машинный код и исполняемые модули (файлы) для других архитектур и ОС. Процесс трансляции программы на языке ассемблера в объектный код принято называть ассемблированием. В отличие от компилирования, ассемблирование - более или менее однозначный и обратимый процесс. В языке ассемблера каждой мнемонике соответствует одна машинная инструкция, в то время как в языках программирования высокого уровня за каждым выражением может скрываться большое количество различных инструкций.

3 ход работы

Создадим два файла с помощью команды «nano lab2.cpp» и «nano lab2.s», где расширение «cpp» для C++, а «s» для Ассемблера. Также пропишем команду «gcc lab.cpp -o lab2», чтобы скомпилировать файл на C++, а команда «gcc -m32 -fno-pie lab2.s -o lab2 -g» для компиляции Ассемблера.

Для начала напишем программу для Ассемблера, которая задана по варианту №3 (рисунок 2.1).

```
1 .data
2
3 print_format:
4     .string "%d\n"
5
6 array:
7     .long -50, 5, 85, -102, -17, 42, -126, -21, 103
8 array_end:
9
10 .text
11 .global main
12 .type main, @function
13
14 main:
15     movl $array, %ebx
16     movb (%ebx), %al
17     jmp bound
18
19 positive:
20     cmpl $0, (%ebx)
21     jl negative
22     btr $0, (%ebx)
23     btr $4, (%ebx)
24     push (%ebx)
25     push $print_format
26     call printf
27     jne less
28
29 negative:
30     sar $4, (%ebx)
31     push (%ebx)
32     push $print_format
33     call printf
34     jne less
35
36 less:
37     addl $4, %ebx
38     movb (%ebx), %al
39
40 bound:
41     cmpl $array_end, %ebx
42     jne positive
43
44 addl $8, %esp
45 movl $1, %eax
46 movl $0, %ebx
47 int $0x80
48
```

Рисунок 2.1 – Код программы на ассемблере

Проверим программу на работоспособность (рисунок 2.2).

```
root@vie738:/home/vdovina# gcc -m32 -fno-pie lab2.s -o lab2 -g
lab2.s: Сообщения ассемблера:
lab2.s:22: Предупреждение: не указан мнемонический суффикс инструкции и нет регистровых операндов;
используется по умолчанию для «btr»
lab2.s:23: Предупреждение: не указан мнемонический суффикс инструкции и нет регистровых операндов;
используется по умолчанию для «btr»
lab2.s:30: Предупреждение: не указан мнемонический суффикс инструкции и нет регистровых операндов;
используется по умолчанию для «sar»
/usr/bin/ld: /tmp/ccVrLBMi.o: предупреждение: перемещение в разделе только для чтения «.text»
/usr/bin/ld: предупреждение: создаётся DT_TEXTREL в PIE
root@vie738:/home/vdovina# ./lab2
-4
4
68
-7
-2
42
-8
-2
102
```

Рисунок 2.3 – Результат программы на Ассемблере

Напишем программу по заданному варианту на языке С (рисунок 2.3).

```
1 #include <stdio.h>
2
3 int main()
4 {
5     signed char mask = 0xEE;
6     char i;
7     signed char array[9] = { -50, 5, 85, -102, -17, 42, -126, -21, 103 };
8
9     for (i = 0; i < 9; ++i)
10    {
11        if (array[i] < 0)
12        {
13            array[i] = array[i] >> 4;
14        }
15        else
16        {
17            array[i] &= mask;
18        }
19    }
20
21    for (i = 0; i < 9; ++i)
22    {
23        printf("%d", array[i]);
24        printf(" ");
25    }
26
27    return 0;
28 }
29 }
```

Рисунок 2.3 – Код программы на С++

Проверим программу на работоспособность (рисунок 2.4).

```

root@vie738:/home/vdovina# gcc lab2.cpp -o lab2
root@vie738:/home/vdovina# ./lab2
-4 4 68 -7 -2 42 -8 -2 102 root@vie738:/home/vdovina#

```

Рисунок 2.4 – Результат программы на C++

Так же по заданию нужно было провести анализ двух написанных программ. Далее сравним две программы. Пропишем команду «**du --bytes name**», чтобы узнать их дисковое пространство (рисунок 2.5) и можно увидеть, что код на Ассемблере меньше, чем код на C++.

```

root@vie738:/home/vdovina# du --bytes lab2
16248 lab2
root@vie738:/home/vdovina# du --bytes lab22
16664 lab22

```

Рисунок 2.5 – Вес программного кода

Напишем команду «**time ./name**», чтобы узнать скорость выполнения программ. Можем заметить, что по скорости выполнения кода на C выполняется быстрее, чем на Ассемблере (рисунок 2.6).

```

root@vie738:/home/vdovina# time ./lab2
-4
4
68
-7
-2
42
-8
-2
102

real    0m0,002s
user    0m0,002s
sys     0m0,000s
root@vie738:/home/vdovina# time ./lab22
-4 4 68 -7 -2 42 -8 -2 102

real    0m0,003s
user    0m0,002s
sys     0m0,000s

```

Рисунок 2.7 – Скорость выполнения программ

Далее проведем дизассемблирование программ с помощью команды «objdump -D name» (рисунок 2.8 – 2.9).

```

00001030 <printf@plt>:
    1030:      ff a3 0c 00 00 00      jmp     *0xc(%ebx)
    1036:      68 00 00 00 00      push   $0x0
    103b:      e9 e0 ff ff ff      jmp     1020 <.plt>

00001040 <__libc_start_main@plt>:
    1040:      ff a3 10 00 00 00      jmp     *0x10(%ebx)
    1046:      68 08 00 00 00      push   $0x8
    104b:      e9 d0 ff ff ff      jmp     1020 <.plt>

Дизассемблирование раздела .plt.got:

00001050 <__cxa_finalize@plt>:
    1050:      ff a3 f0 ff ff ff      jmp     *-0x10(%ebx)
    1056:      66 90                  xchg    %ax,%ax

Дизассемблирование раздела .text:

00001060 <_start>:
    1060:      31 ed                  xor     %ebp,%ebp
    1062:      5e                    pop     %esi
    1063:      89 e1                  mov     %esp,%ecx
    1065:      83 e4 f0              and     $0xfffffffff0,%esp
    1068:      50                    push    %eax
    1069:      54                    push    %esp
    106a:      52                    push    %edx
    106b:      e8 22 00 00 00      call    1092 <_start+0x32>
    1070:      81 c3 90 2f 00 00      add     $0x2f90,%ebx
    1076:      8d 83 50 d2 ff ff      lea     -0x2db0(%ebx),%eax
    107c:      50                    push    %eax
    107d:      8d 83 f0 d1 ff ff      lea     -0x2e10(%ebx),%eax
    1083:      50                    push    %eax
    1084:      51                    push    %ecx

```

Рисунок 2.8 – Дизассемблирование программы на Ассемблере

1140:	e9 7b ff ff ff	jmpq	10c0 <register_tm_clones>
0000000000001145 <main>:			
1145:	55	push	%rbp
1146:	48 89 e5	mov	%rsp,%rbp
1149:	48 83 ec 10	sub	\$0x10,%rsp
114d:	c6 45 fe ee	movb	\$0xee,-0x2(%rbp)
1151:	48 b8 ce 05 55 9a ef	movabs	\$0xeb822aef9a5505ce,%rax
1158:	2a 82 eb		
115b:	48 89 45 f5	mov	%rax,-0xb(%rbp)
115f:	c6 45 fd 67	movb	\$0x67,-0x3(%rbp)
1163:	c6 45 ff 00	movb	\$0x0,-0x1(%rbp)
1167:	80 7d ff 08	cmpb	\$0x8,-0x1(%rbp)
116b:	7f 50	jg	11bd <main+0x78>
116d:	0f be 45 ff	movsbl	-0x1(%rbp),%eax
1171:	48 98	cltq	
1173:	0f b6 44 05 f5	movzbl	-0xb(%rbp,%rax,1),%eax
1178:	84 c0	test	%al,%al
117a:	79 1a	jns	1196 <main+0x51>
117c:	0f be 45 ff	movsbl	-0x1(%rbp),%eax
1180:	48 98	cltq	
1182:	0f b6 54 05 f5	movzbl	-0xb(%rbp,%rax,1),%edx
1187:	0f be 45 ff	movsbl	-0x1(%rbp),%eax
118b:	c0 fa 04	sar	\$0x4,%dl
118e:	48 98	cltq	
1190:	88 54 05 f5	mov	%dl,-0xb(%rbp,%rax,1)
1194:	eb 1b	jmp	11b1 <main+0x6c>
1196:	0f be 45 ff	movsbl	-0x1(%rbp),%eax
119a:	48 98	cltq	
119c:	0f b6 44 05 f5	movzbl	-0xb(%rbp,%rax,1),%eax
11a1:	0f be 4d ff	movsbl	-0x1(%rbp),%ecx
11a5:	22 45 fe	and	-0x2(%rbp),%al
11a8:	89 c2	mov	%eax,%edx
11aa:	48 63 c1	movslq	%ecx,%rax

Рисунок 2.9 – Дизассемблирование программы на C++

Сравним вес полученных файлов (рисунок 2.10) и можно заметить, что после дизассемблирования исполняемый файл на ассемблер стал больше весить.

```

root@vie738:/home/vdovina# objdump -D lab2 > lab2_dump
root@vie738:/home/vdovina#
root@vie738:/home/vdovina# objdump -D lab22 > lab22_dump
root@vie738:/home/vdovina# du --bytes lab2_dump
42980   lab2_dump
root@vie738:/home/vdovina# du --bytes lab22_dump
41100   lab22_dump

```

Рисунок 2.10 – Сравнение программного кода после дизассемблирования

Был собран и запущен образ на основе файла «Dockerfile» (рисунок 2.11 – 2.12).


```
1 FROM debian
2 RUN apt-get update
3 RUN apt install gcc gdb gcc-multilib nano -y
4 COPY lab2.s .
5 RUN gcc -m32 lab2.s -o lab2
6 CMD ./lab2 |
```

Рисунок 2.11 – «Dockerfile»

```
root@vie738:/home/vdovina# docker run -it lab2
-4
4
68
-7
-2
42
-8
-2
102
```

Рисунок 2.12 – Запуск образа на основе «Dockerfile»

Загрузим все файлы в репозиторий GitHub.

4 заключение

В результате выполнения лабораторной работы были получены навыки программирования на языке ассемблера, а также выполнен ряд практических задач.