Отчет по лабораторной работе №8

Дисциплина: архитектура компьютера

Горбунова Яна Сергеевна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
	4.1 Реализация циклов в NASM	8
	4.2 Обработка аргументов командной строки	11
	4.3 Задание для самостоятельной работы	13
5	Выводы	16
6	Список литературы	17

Список иллюстраций

4.1	Создание каталога	8
4.2	Копирование программы из листинга	8
4.3	Запуск программы	9
4.4	Изменение программы	9
4.5	Запуск измененной программы	10
4.6	Добавление push и pop в цикл программы	10
4.7	Запуск измененной программы	11
		11
4.9	Запуск второй программы	11
4.10	Копирование программы из третьего листинга	12
4.11	Запуск третьей программы	12
4.12	Изменение третьей программы	13
4.13	Запуск измененной третьей программы	13
4.14	Написание программы для самостоятельной работы	14
4.15	Запуск программы для самостоятельной работы	15

Список таблиц

1 Цель работы

Приобретение навыков написания программ с использованием циклов и обработкой аргументов командной строки.

2 Задание

- 1. Реализация циклом в NASM
- 2. Обработка аргументов командной строки
- 3. Самостоятельное написание программы по материалам лабораторной работы

3 Теоретическое введение

Стек — это структура данных, организованная по принципу LIFO («Last In — First Out» или «последним пришёл — первым ушёл»). Стек является частью архитектуры процессора и реализован на аппаратном уровне. Для работы со стеком в процессоре есть специальные регистры (ss, bp, sp) и команды. Основной функцией стека является функция сохранения адресов возврата и передачи аргументов при вызове процедур. Кроме того, в нём выделяется память для локальных переменных и могут временно храниться значения регистров.

4 Выполнение лабораторной работы

4.1 Реализация циклов в NASM

Создаю каталог для программ лабораторной работы №8 (рис. 4.1).

```
ysgorbunova@dk8n60 ~ $ mkdir ~/work/arch-pc/lab08
ysgorbunova@dk8n60 ~ $ cd ~/work/arch-pc/lab08
ysgorbunova@dk8n60 ~/work/arch-pc/lab08 $ touch lab8-1.asm
```

Рис. 4.1: Создание каталога

Копирую в созданный файл программу из листинга. (рис. 4.2).

```
1 %include 'in_out.asm'
 2 SECTION .data
 3 msg1 db 'Введите N: ',0h
 4 SECTION .bss
5 N: resb 10
6 SECTION .text
7 global _start
9; ---- Вывод сообщения 'Введите N: '
10 mov eax, msg1
11 call sprint
12; ---- Ввод 'N'
13 mov ecx, N
14 mov edx, 10
15 call sread
16 ; ---- Преобразование 'N' из символа в число
17 mov eax,N
18 call atoi
19 mov [N], eax
20 : ---- Организация цикла
21 mov ecx,[N] ; Счетчик цикла, 'ecx=N'
22 label:
23 sub ecx,1 ; 'ecx=ecx-1'
24 mov [N],ecx
25 mov eax,[N]
26 call iprintLF
27 loop label
```

Рис. 4.2: Копирование программы из листинга

Запускаю программу, она показывает работу циклов в NASM (рис. 4.3).

```
ysgorbunova@dk8n60 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
ysgorbunova@dk8n60 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.c
ysgorbunova@dk8n60 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 5
5
4
3
2
1
ysgorbunova@dk8n60 ~/work/arch-pc/lab08 $ ■
```

Рис. 4.3: Запуск программы

Заменяю программу изначальную так, что в теле цикла я изменяю значение регистра есх (рис. 4.4).

```
1 %include 'in_out.asm'
 2 SECTION .data
3 msg1 db 'Введите N: ',0h
 4 SECTION .bss
 5 N: resb 10
6 SECTION .text
 7 global _start
 8 _start:
9; ---- Вывод сообщения 'Введите N: '
10 mov eax, msg1
11 call sprint
12; ---- Ввод 'N'
13 mov ecx, N
14 mov edx, 10
15 call sread
16 : ---- Преобразование 'N' из символа в число
17 mov eax, N
18 call atoi
19 mov [N], eax
20; ---- Организация цикла
21 mov ecx,[N] ; Счетчик цикла, 'ecx=N'
22 label:
23 push есх ; добавление значения есх в стек
24 sub ecx,1
25 mov [N], ecx
26 mov eax,[N]
27 call iprintLF
28 рор есх ; извлечение значения есх из стека
29 loop label
```

Рис. 4.4: Изменение программы

Из-за того, что теперь регистр есх на каждой итерации уменьшается на 2 значения, количество итераций уменьшается вдвое (рис. 4.5).

```
ysgorbunova@dk8n60 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 6
5
3
```

Рис. 4.5: Запуск измененной программы

Добавляю команды push и pop в программу (рис. 4.6).

```
1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",∅
4 SECTION .text
5 global _start
6 _start:
7 рор есх ; Извлекаем из стека в есх количество
8; аргументов (первое значение в стеке)
9 pop edx ; Извлекаем из стека в edx имя программы
10; (второе значение в стеке)
11 sub ecx,1 ; Уменьшаем есх на 1 (количество
12; аргументов без названия программы)
13 mov esi, 1 ; Инициализируем esi значением 1 для произведения
14 next:
15 cmp ecx,0h ; проверяем, есть ли еще аргументы
16 jz _end ; если аргументов нет выходим из цикла
17 ; (переход на метку '_end')
18 рор еах ; иначе извлекаем следующий аргумент из стека
19 call atoi ; преобразуем символ в число
20 imul esi,eax ; умножаем промежуточное произведение
21 ; на текущий аргумент esi=esi*eax
22 loop next ; переход к обработке следующего аргумента
23 _end:
24 mov eax, msg ; вывод сообщения "Результат: "
25 call sprint
26 mov eax, esi ; записываем произведение в регистр eax
27 call iprintLF ; печать результата
28 call quit ; завершение программы
```

Рис. 4.6: Добавление push и pop в цикл программы

Теперь количество итераций совпадает введенному N, но произошло смещение выводимых чисел на -1 (рис. 4.7).

```
ysgorbunova@dk8n60 ~/work/arch-pc/lab08 $ nasm -f elf -l lab8-1 lab8-1.asm
ysgorbunova@dk8n60 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
ysgorbunova@dk8n60 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 7
6
5
4
3
2
1
0
```

Рис. 4.7: Запуск измененной программы

4.2 Обработка аргументов командной строки

Создаю новый файл для программы и копирую в него код из следующего листинга (рис. 4.8).

```
ysgorbunova@dk8n60 ~/work/arch-pc/lab08 $ touch lab8-3.asm
ysgorbunova@dk8n60 ~/work/arch-pc/lab08 $ ■
```

Рис. 4.8: Копирование программы из листинга

Компилирую программу и запускаю, указав аргументы. Программой было обратоно то же количество аргументов, что и было введено (рис. 4.9).

```
ysgorbunova@dk8n60 ~/work/arch-pc/lab08 $ nasm -f elf -l lab8-2 lab8-2.asm
ysgorbunova@dk8n60 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-2 lab8-2.o
ysgorbunova@dk8n60 ~/work/arch-pc/lab08 $ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
```

Рис. 4.9: Запуск второй программы

Создаю новый файл для программы и копирую в него код из третьего листинга (рис. 4.10).

```
1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",∅
4 SECTION .text
5 global _start
6_start:
7 рор есх ; Извлекаем из стека в есх количество
8; аргументов (первое значение в стеке)
9 pop edx ; Извлекаем из стека в edx имя программы
10; (второе значение в стеке)
11 sub ecx,1 ; Уменьшаем есх на 1 (количество
12; аргументов без названия программы)
13 mov esi, 1 ; Инициализируем esi значением 1 для произведения
15 cmp ecx,0h ; проверяем, есть ли еще аргументы
16 jz _end ; если аргументов нет выходим из цикла
17; (переход на метку '_end')
18 рор еах ; иначе извлекаем следующий аргумент из стека
19 call atoi ; преобразуем символ в число
20 imul esi,eax ; умножаем промежуточное произведение
21 ; на текущий аргумент esi=esi*eax
22 loop next ; переход к обработке следующего аргумента
23 _end:
24 mov eax, msg ; вывод сообщения "Результат: "
25 call sprint
26 mov eax, esi ; записываем произведение в регистр eax
27 call iprintLF ; печать результата
28 call quit ; завершение программы
```

Рис. 4.10: Копирование программы из третьего листинга

Компилирую программу и запускаю, указав в качестве аргументов некоторые числа, программа их складывает (рис. 4.11).

```
ysgorbunova@dk8n60 ~/work/arch-pc/lab08 $ ./lab8-3 12 13 7 10 5
Результат: 47
ysgorbunova@dk8n60 ~/work/arch-pc/lab08 $ █
```

Рис. 4.11: Запуск третьей программы

Изменяю поведение программы так, чтобы указанные аргументы она умножала, а не складывала (рис. 4.12).

```
1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
 4 SECTION .text
 5 global _start
 6_start:
 7 рор есх ; Извлекаем из стека в 'есх' количество
 8; аргументов (первое значение в стеке)
9 pop edx; Извлекаем из стека в 'edx' имя программы
10; (второе значение в стеке)
11 sub ecx,1; Уменьшаем 'ecx' на 1 (количество
12 ; аргументов без названия программы)
13 mov esi, 0 ; Используем 'esi' для хранения
14; промежуточных сумм
15 next:
16 cmp ecx,0h ; проверяем, есть ли еще аргументы
17 jz _end ; если аргументов нет выходим из цикла
18 ; (переход на метку `_end`)
19 рор eax ; иначе извлекаем следующий аргумент из стека 20 call atoi ; преобразуем символ в число
21 add esi,eax ; добавляем к промежуточной сумме
22 ; след. аргумент 'esi=esi+eax'
23 loop next ; переход к обработке следующего аргумента
25 mov eax, msg ; вывод сообщения "Результат: "
26 call sprint
27 mov eax, esi ; записываем сумму в регистр 'eax'
                                                                        Mark = 111 ----- 0 = 1-20 C-122 IN
```

Рис. 4.12: Изменение третьей программы

Программа действительно теперь умножает данные на вход числа (рис. 4.13).

```
ysgorbunova@dk3n55 ~/work/arch-pc/lab08 $ nasm -f elf -l lab8-3 lab8-3.asm ysgorbunova@dk3n55 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o ysgorbunova@dk3n55 ~/work/arch-pc/lab08 $ ./lab8-3 5 7 15 Результат: 525
```

Рис. 4.13: Запуск измененной третьей программы

4.3 Задание для самостоятельной работы

Пишу программму, которая будет находить сумма значений для функции f(x) = 10x-4, которая совпадает с моим девытым варинтом (рис. 4.14).

```
1 %include 'in_out.asm'
 3 SECTION .data 4 msg_func db "Функция: f(x) = 10(x - 1)", 0 5 msg_result db "Результат: ", 0
 7 SECTION .text
  8 GLOBAL _start
10 _start:
11 mov
         mov eax, msg_func
11 mo
12 ca
13
14 po
15 po
16 su
17 mo
18
19 next:
          call sprintLF
                                ; Получаем количество входных данных
; Сохраняем какую-то дополнительную информацию
; Уменьшаем количество на ↑ (х - ↑)
; Инициализация суммы результатов
          pop ecx
          pop edx
         sub ecx, 1
mov esi, 0
          стр есх, ∅ ; Проверка, закончились ли входные данные
20
21
22
23
24
25
26
27
28
          рор eax ; Получаем следующее значение x вперёд call atoi ; Преобразуем строку в число
          sub eax, 1 ; Вычисление (x - 1) mov ebx, 10 ; Константа для умножения (10)
          mul ebx ; Умножение на 10 add esi, eax ; Добавляем результат к сумме
          loon_next : Перехол_к_слелующему_элементу, если_есть
Mattah ▼ Illungua табилации 2 ▼ In 39 Col 43
```

Рис. 4.14: Написание программы для самостоятельной работы

Код программы:

%include 'in_out.asm' SECTION .data msg_func db "Функция: f(x) = 10(x - 1)", 0 msg_result db "Результат: ", 0 SECTION .text GLOBAL _start _start: mov eax, msg_func call sprintLF pop ecx ; Получаем количество входных данных pop edx ; Сохраняем какую-то дополнительную информацию

```
sub ecx, 1 ; Уменьшаем количество на 1 (x - 1)
   mov esi, 0 ; Инициализация суммы результатов
next:
   cmp ecx, 0
               ; Проверка, закончились ли входные данные
   jz _end
              ; Получаем следующее значение х вперёд
   pop eax
   call atoi ; Преобразуем строку в число
   \operatorname{sub} eax, 1 ; Вычисление (x - 1)
   mov ebx, 10 ; Константа для умножения (10)
   mul ebx
                  ; Умножение на 10
   add esi, eax
                  ; Добавляем результат к сумме
   loop next
                  ; Переход к следующему элементу, если есть
end:
   mov eax, msg_result
   call sprint ; Печатаем сообщение с результатом
   mov eax, esi ; Перемещаем общую сумму в еах
   call iprintLF ; Печатаем результат
   call quit
                   ; Завершаем программу
```

Проверяю работу программы, указав в качестве аргумента несколько чисел (рис. 4.15).

```
ysgorbunova@dk3n55 ~/work/arch-pc/lab08 $ ./lab8-3 12 13 7 10 5
Результат: 54600
ysgorbunova@dk3n55 ~/work/arch-pc/lab08 $
```

Рис. 4.15: Запуск программы для самостоятельной работы

5 Выводы

В результате выполнения данной лабораторной работы я приобрела навыки написания программ с использованием циклов а также научилась обрабатывать аргументы командной строки.

6 Список литературы

- 1. Курс на ТУИС
- 2. Лабораторная работа №8
- 3. Программирование на языке ассемблера NASM Столяров А. В.