Лабораторная работа номер 8.

Программирование цикла. Обработка аргументов командной строки.

Сорокин Кирилл

Содержание

Список иллюстраций

1 Цель работы

Научиться писать циклы и обрабатывать информацию из командной строки с помощью языка ассемблер.

2 Задание

Изучить приведённый материал на практике и выполнить самостоятельную работу.

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. Стек имеет вершину, адрес последнего добавленного элемента, который хранится в регистре esp (указатель стека). Противоположный конец стека называется дном. Значение, помещённое в стек последним, извлекается первым. При помещении значения в стек указатель стека уменьшается, а при извлечении — увеличивается.

4 Выполнение лабораторной работы

Создадим необходимые для работы директории и файлы (рис. 4.1).

```
kvsorokin@dk3n60 ~ $ mkdir ~/work/arch-pc/lab08
kvsorokin@dk3n60 ~ $ cd ~/work/arch-pc/lab08
kvsorokin@dk3n60 ~/work/arch-pc/lab08 $ touch lab8-1.asm
kvsorokin@dk3n60 ~/work/arch-pc/lab08 $ gedit lab8-1.asm
```

Рис. 4.1: Создание файлов и директорий

Откроем файл lab8-1.asm и введём в него текст программы(рис. 4.2).

```
;-------------------------------------------------------------------
; Программа вывода значений регистра 'ecx'
;-------------------------------------------------------------------
%include 'in_out.asm'
SECTION .data
msg1 db 'Введите N: ',0h

SECTION .bss
N: resb 10
SECTION .text
global _start
_start:
; -----
        mov eax,msg1
        call sprint
; ----- Ввод 'N'
        mov ecx, N
        mov edx, 10
        call sread
; ----- Преобразование 'N' из символа в число
        mov eax,N
        call atoi
        mov [N],eax
; ------ Организация цикла
        mov ecx,[N]
; Счетчик цикла, `ecx=N`
label:
        mov [N],ecx
        mov eax,[N]
        call iprintLF
        loop label
        ; Вывод значения `N`
        ; `ecx=ecx-1` и если `ecx` не '0'
        ; переход на `label`
        call quit
```

Рис. 4.2: Текст первой программы

После компиляции файлов запустим программу и увидим не совсем требуемый результат число хоть и уменьшается, но не до нуля(рис. 4.3).

```
kvsorokin@dk3n60 ~/work/arch-pc/lab08 $ gedit lab8-1.asm
kvsorokin@dk3n60 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
kvsorokin@dk3n60 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
kvsorokin@dk3n60 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 2
2
1
```

Рис. 4.3: Первая попытка выполнить

Изменим содержимое label в тексте программы(рис. 4.4).

```
label:
        sub ecx,1
        mov [N],ecx
        mov eax,[N]
        call iprintLF
        loop label
        ; Вывод значени
        ; `ecx=ecx-1` и
        ; переход на `l
        call quit
```

Рис. 4.4: Редактирование label 1

Попробуем ещё раз выполнить программу и увидим, что программа попадает в бесконечный цикл, выводя совсем не корректные значения. Это связанно с тем, что мы вначале уменьшает ecx с помощью sub, а потом ещё раз с помощью loop, поэтому он и не принимает значения 0 и уходит в бесконечность.(рис. 4.5).

```
4294736658
4294736656
4294736654
4294736652
4294736650
4294736648
42^Z
[1]+  Остановлен    ./lab8-1
kvsorokin@dk3n60 ~/work/arch-pc/lab08 $
```

Рис. 4.5: Вторая попытка выполнить

Ещё раз изменим содержимое label так, чтобы ecx мог заходить и выходить из стека.(рис. 4.6).

```
label:
        push ecx ; доб
        sub ecx,1
        mov [N],ecx
        mov eax,[N]
        call iprintLF
        pop ecx ; извл
        loop label
        call quit
```

Рис. 4.6: Редактирование label 2

После выполнения опять увидим, что наконец достигнут желаемый результат.(рис. 4.7).

```
kvsorokin@dk3n60 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
kvsorokin@dk3n60 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
kvsorokin@dk3n60 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 5
4
3
2
1
0
```

Рис. 4.7: Успешный запуск цикла

Создадим файл lab8-2.asm и введём в него текст программы(рис. 4.8).

```
%include 'in_out.asm'
SECTION .text
global _start
_start:
pop ecx
pop edx
sub ecx, 1
        ; Извлекаем из стека в `ecx` количество
        ; аргументов (первое значение в стеке)
        ; Извлекаем из стека в `edx` имя программы
        ; (второе значение в стеке)
        ; Уменьшаем `ecx` на 1 (количество
        ; аргументов без названия программы)
next:
        cmp ecx, 0; проверяем, есть ли еще аргументы
        jz _end; если аргументов нет выходим из цикла
        pop eax ; (переход на метку `_end`)
                ; иначе извлекаем аргумент из стека
        call sprintLF; вызываем функцию печати
        loop next ; переход к обработке следующего
                ; аргумента (переход на метку `next`)
_end:
        call quit
```

Рис. 4.8: Программы lab8-2

Выполним и увидим, что программа считает за аргументы каждый данный ей элемент разделённый пробелом. (их 4) (рис. 4.9).

```
kvsorokin@dk3n60 ~/work/arch-pc/lab08 $ touch lab8-2.asm
kvsorokin@dk3n60 ~/work/arch-pc/lab08 $ gedit lab8-2.asm
kvsorokin@dk3n60 ~/work/arch-pc/lab08 $ nasm -f elf lab8-2.asm
kvsorokin@dk3n60 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-2 lab8-2.o
kvsorokin@dk3n60 ~/work/arch-pc/lab08 $ ./lab8-2 аргумент1 аргумент 2 'ap
гумент 3'
аргумент1
аргумент
2
аргумент 3
kvsorokin@dk3n60 ~/work/arch-pc/lab08 $
```

Рис. 4.9: Раюота с агрументами

Создадим файл lab8-3.asm и запишем в него программу для суммы введённых чисел (рис. 4.10).

```
%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
SECTION .text
global _start
_start:
pop ecx ; Извлекаем из стека в `ecx` количество
        ; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в `edx` имя программы
        ; (второе значение в стеке)
sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
        ; аргументов без названия программы)
mov esi, 0 ; Используем `esi` для хранения
        ; промежуточных сумм
next:
        cmp ecx,0h      ; проверяем, есть ли еще аргументы
        jz _end ; если аргументов нет выходим из цикла
        pop eax
                ; (переход на метку `_end`)
                ; иначе извлекаем следующий аргумент из стека
        call atoi       ; преобразуем символ в число
        add esi,eax ; добавляем к промежуточной сумме
                ; след. аргумент `esi=esi+eax`
        loop next ; переход к обработке следующего аргумента
_end:
        mov eax, msg ; вывод сообщения "Результат: "
        call sprint
        mov eax, esi ; записываем сумму в регистр `eax`
        call iprintLF; печать результата
        call quit       ; завершение программы
```

Рис. 4.10: Содержимое файла lab8-3.asm

Проверим её на нескольких наборах данных. (рис. 4.11).

```
kvsorokin@dk3n60 ~/work/arch-pc/lab08 $ ./lab8-3 1 2 3 4 90
Результат: 100
kvsorokin@dk3n60 ~/work/arch-pc/lab08 $ ./lab8-3 56 84 29
Результат: 169
kvsorokin@dk3n60 ~/work/arch-pc/lab08 $
```

Рис. 4.11: Сложение чисел

Изменим текст программы, чтобы она умножала числа(рис. 4.12).

```
%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
SECTION .text
global _start
_start:
pop ecx ; Извлекаем из стека в `ecx` количество
        ; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в `edx` имя программы
        ; (второе значение в стеке)
sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
        ; аргументов без названия программы)
mov esi, 1 ; Используем `esi` для хранения
        ; промежуточных сумм
next:
        cmp ecx,0h      ; проверяем, есть ли еще аргументы
        jz _end ; если аргументов нет выходим из цикла
        pop eax
                ; (переход на метку `_end`)
                ; иначе извлекаем следующий аргумент из стека
        call atoi       ; преобразуем символ в число
        mov edx, eax
        mov eax, esi
        mul edx ; добавляем к промежуточной сумме
                ; след. аргумент `esi=esi+eax`
        mov esi, eax
        loop next ; переход к обработке следующего аргумента
_end:
        mov eax, msg ; вывод сообщения "Результат: "
        call sprint
        mov eax, esi ; записываем сумму в регистр `eax`
        call iprintLF; печать результата
        call quit       ; завершение программы
```

Рис. 4.12: Изменённый текст lab8-3

Здесь видны основные отличия от предыдущей версии программы. Однако необходимо также изменить начально значение esi на 1, чтобы умножение выполнялось корректно. (рис. 4.13).

```
mov edx, eax
mov eax, esi
mul edx ; добавляем к промежуточной сумме
        ; след. аргумент `esi=esi+eax`
mov esi, eax
loop next ; переход к обработке следующего аргумента
```

Рис. 4.13: Основные изменения lab8-3

Убедимся в корректности умножения чисел (рис. **??**).

```
kvsorokin@dk3n60 ~/work/arch-pc/lab08 $ gedit lab8-3.asm
kvsorokin@dk3n60 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
kvsorokin@dk3n60 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
kvsorokin@dk3n60 ~/work/arch-pc/lab08 $ ./lab8-3 1 2 3
Результат: 6
kvsorokin@dk3n60 ~/work/arch-pc/lab08 $ ./lab8-3 2 4 5
Результат: 40
kvsorokin@dk3n60 ~/work/arch-pc/lab08 $
```

4.1 Самостоятельная работа

Так как наш вариант 1, напишем программу для суммы значений функции f(x)=2x+15 от введённых значений (рис. 4.14).

```
%include 'in_out.asm'
SECTION .data
fun db "Функция: f(x)=2x+15", 0
msg db "Результат: ",0
SECTION .text
global _start
_start:
pop ecx ; Извлекаем из стека в `ecx` количество
        ; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в `edx` имя программы
        ; (второе значение в стеке)
sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
        ; аргументов без названия программы)
mov esi, 0 ; Используем `esi` для хранения
        ; промежуточных сумм
next:
        cmp ecx,0h      ; проверяем, есть ли еще аргументы
        jz _end ; если аргументов нет выходим из цикла
        pop eax
                ; (переход на метку `_end`)
                ; иначе извлекаем следующий аргумент из стека
        call atoi       ; преобразуем символ в число
        mov edx, 2
        mul edx
        add esi,eax ; добавляем к промежуточной сумме
                ; след. аргумент `esi=esi+eax`
        mov edx, 15
        add esi, edx
        loop next ; переход к обработке следующего аргумента
_end:
        mov eax, fun
        call sprintLF
        mov eax, msg ; вывод сообщения "Результат: "
        call sprint
        mov eax, esi ; записываем сумму в регистр `eax`
        call iprintLF; печать результата
        call quit       ; завершение программы
```

Рис. 4.14: Текст программы

Основными изменениями от программы складывающей аргументы будет в необходимости домножать аргумент на 2 и прибавлять к нему 15. Однако необходимо также не забыть о том, что нужно ещё вывести функцию на экран (3 строки разбросаны по тексту программы, поэтому они не отображены на этом рисунке) (рис. 4.15).

```
mov edx, 2
mul edx
add esi,eax ; добавляем к промежуточной сумме
        ; след. аргумент `esi=esi+eax`
mov edx, 15
add esi, edx
```

Рис. 4.15: Основные изменения

Скомпилировав файл, убедимся, что он работает корректно (рис. 4.16).

```
kvsorokin@dk3n60 ~/work/arch-pc/lab08 $ gedit lab8-s.asm
kvsorokin@dk3n60 ~/work/arch-pc/lab08 $ nasm -f elf lab8-s.asm
kvsorokin@dk3n60 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-s lab8-s.o
kvsorokin@dk3n60 ~/work/arch-pc/lab08 $ ./lab8-s 2 4 5
Функция: f(x)=2x+15
Результат: 67
```

Рис. 4.16: Работа самостоятельной работы

5 Выводы

Мы научились использовать писать циклы на языке ассемблера, а также получать информацию из командной строки.

Список литературы

1. GDB: The GNU Project Debugger. — URL: https://www.gnu.org/software/gdb/.
2. GNU Bash Manual. — 2016. — URL: https://www.gnu.org/software/bash/manual/.
3. Midnight Commander Development Center. — 2021. — URL: https://midnightcommander.org/.
4. NASM Assembly Language Tutorials. — 2021. — URL: https://asmtutor.com/.
5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: http://www.amazon.com Learning-bash-Shell-Programming-Nutshell/dp/0596009658.
6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. The NASM documentation. — 2021. — URL: https://www.nasm.us/docs.php.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. :Солон-Пресс, 2017.
11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
12. Расширенный ассемблер: NASM. — 2021. — URL: https://www.opennet.ru/docs/RUS/nasm/.
13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВ- Петербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
14. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-е изд. — М. : МАКС Пресс, 2011. — URL: http://www.stolyarov.info/books/asm_unix.
15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. -

874 с. — (Классика Computer Science).
16. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. -СПб. : Питер,
17. — 1120 с. — (Классика Computer Science)