Лабораторная работа №8

Программирование цикла. Обработка аргументов командной строки

Соловьев Богдан Михайлович

Содержание

1 Цель работы

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

2 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.

Для стека существует две основные операции:

• добавление элемента в вершину стека (push);

• извлечение элемента из вершины стека (pop).

Команда push размещает значение в стеке, т.е. помещает значение в ячейку памяти, на которую указывает регистр esp, после этого значение регистра esp увеличивается на 4. Данная команда имеет один операнд — значение, которое необходимо поместить в стек.

3 Выполнение лабораторной работы

Создаю файл lab8-1.asm (рис. 1).

```
[bmsolovjev@fedora report]$ mkdir ~/work/arch-pc/lab08
mkdir: невозможно создать каталог «/home/bmsolovjev/work/arch-pc/lab08»: Файл существует
[bmsolovjev@fedora report]$ cd ~/work/arch-pc/lab08
[bmsolovjev@fedora lab08]$ touch lab8-1.asm
[bmsolovjev@fedora lab08]$
```

Figure 1: Создание файла

Записываю в созданный файл следующий код (рис. 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:
; ----- Вывод сообщения 'Введите N: '
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 ; Вывод значения `N`
loop label ; `ecx=ecx-1` и если `ecx` не '0'
; переход на `label`
call quit
```

Figure 2: Код программы

Проверка работы созданного файла (рис. 3).

```
[bmsolovjev@fedora lab08]$ nasm -f elf lab8-1.asm
[bmsolovjev@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[bmsolovjev@fedora lab08]$ ./lab8-1
Введите N: 5
5
4
3
2
1
[bmsolovjev@fedora lab08]$
```

Figure 3: Выполнение программы

Изменяю код программы, теперь значение в регистре ecx не запоминается, и количество циклов становится бесконечным (рис. 4).

```
;-------------------------------------------------------------------
; 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:
; ----- Вывод сообщения 'Введите N: '
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
label:
sub ecx,1 ; `ecx=ecx-1`
mov [N],ecx
mov eax,[N]
call iprintLF

loop label ; `ecx=ecx-1` и если `ecx` не '0'
; переход на `label`
call quit
```

Figure 4: Неправильный код программы

Файл создаёт бесконечный цикл (рис. 5).

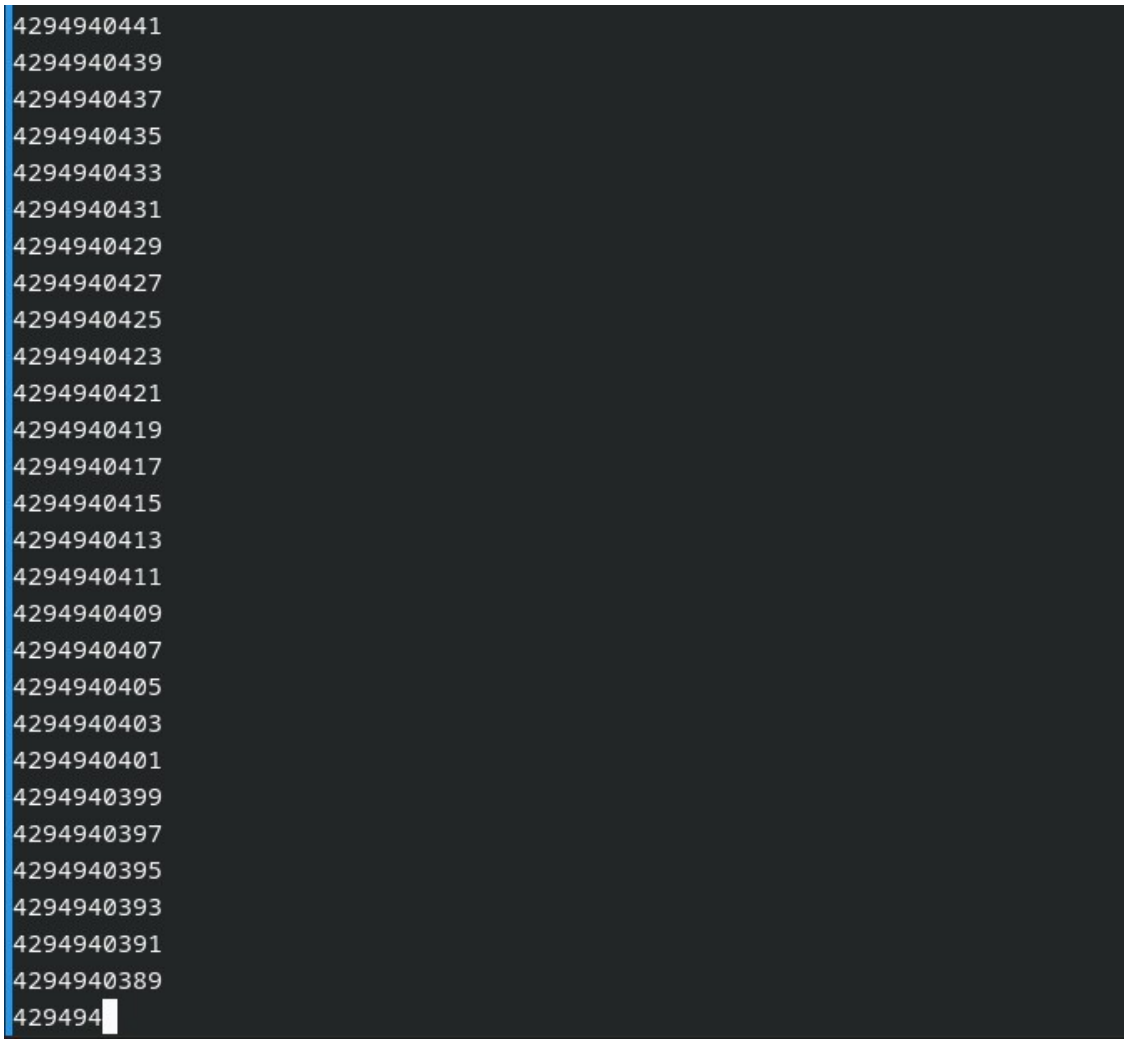

Figure 5: Бесконечный цикл

Изменяю код программы таким образом, чтобы в заносить и вытаскивать из стека значение счётчика цикла (рис. 6).

```
;-------------------------------------------------------------------
; 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:
; ----- Вывод сообщения 'Введите N: '
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
label:
push ecx ; добавление значения ecx в стек
sub ecx,1
mov [N],ecx
mov eax,[N]
call iprintLF
pop ecx ; извлечение значения ecx из стека
loop label
; переход на `label`
call quit
```

Figure 6: Поправленный код

Теперь программа снова работает корректно, и количество циклов соответствует N (рис. 7).

```
[bmsolovjev@fedora ~]$ cd /home/bmsolovjev/work/arch-pc/lab08
[bmsolovjev@fedora lab08]$ nasm -f elf lab8-1.asm
[bmsolovjev@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[bmsolovjev@fedora lab08]$ ./lab8-1
Введите N: 5
4
3
2
1
0
[bmsolovjev@fedora lab08]$
```

Figure 7: Программа работает

Создаю новый файл и ввожу в него новый текст программы (рис. 8).

```
%include 'in_out.asm'
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)
next:
cmp ecx, 0 ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
; (переход на метку `_end`)
pop eax ; иначе извлекаем аргумент из стека
call sprintLF ; вызываем функцию печати
loop next ; переход к обработке следующего
; аргумента (переход на метку `next`)
_end:
call quit
```

Figure 8: Текст файла lab8-2.asm

Проверка работы нового файла. При таком запросе программа обработала 4 аргумента (рис. 9).

```
[bmsolovjev@fedora lab08]$ ld -m elf_i386 -o lab8-2 lab8-2.o
[bmsolovjev@fedora lab08]$ ./lab8-2
[bmsolovjev@fedora lab08]$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
[bmsolovjev@fedora lab08]$
```

Figure 9: Программа принимает аргументы

Создаю новую программу, которая вычисляет сумму аргументов командной строки (рис. 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
; (переход на метку `_end`)
pop eax ; 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
```

```
[bmsolovjev@fedora lab08]$ nasm -f elf lab8-3.asm
[bmsolovjev@fedora lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[bmsolovjev@fedora lab08]$ ./lab8-3 12 13 7 10 5
Результат: 47
[bmsolovjev@fedora lab08]$ ./lab8-3 100 3 50
Результат: 153
[bmsolovjev@fedora lab08]$
```

Figure 10: Программа считает сумму

Теперь меняю текст программы таким образом, чтобы она вычисляла произведение (рис. 11).

```
%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
; (переход на метку `_end`)
pop eax ; иначе извлекаем следующий аргумент из стека
call atoi ; преобразуем символ в число
mul esi ; добавляем к промежуточной сумме
mov esi, eax
; 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
```

Figure 11: Программа считает произведие

Проверка работы нового файла. (рис. 12).

```
[bmsolovjev@fedora lab08]$ nasm -f elf lab8-3.asm
[bmsolovjev@fedora lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[bmsolovjev@fedora lab08]$ ./lab8-3 100 3 50
Результат: 15000
[bmsolovjev@fedora lab08]$
```

Figure 12: Программа считает произведение

4 Самостоятельная работа

Текст программы, считающей сумму значений функции f(x) = 10x -5 (рис. 13).

```
%include 'in_out.asm'
SECTION .data
msg0 db "f(x) = 10x - 5",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
; (переход на метку `_end`)
pop eax ; иначе извлекаем следующий аргумент из стека
call atoi ; преобразуем символ в число
mov ebx, 10
mul ebx
add eax, -5
add esi, eax
; след. аргумент `esi=esi+eax`
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg0
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
```

```
%include 'in_out.asm'
SECTION .data
msg0 db "f(x) = 10x - 5",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
; (переход на метку `_end`)
pop eax ; иначе извлекаем следующий аргумент из стека
call atoi ; преобразуем символ в число
mov ebx, 10
mul ebx
add eax, -5
add esi, eax
; след. аргумент `esi=esi+eax`
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg0
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
```

Figure 13: Код моей программы

Проверка работы программы (рис. 14).

```
[bmsolovjev@fedora lab08]$ nasm -f elf zadanie1.asm
[bmsolovjev@fedora lab08]$ ld -m elf_i386 -o zadanie1 zadanie1.o
[bmsolovjev@fedora lab08]$ ./zadanie1 1 2 3 4
f(x) = 10x - 5
Результат: 80
[bmsolovjev@fedora lab08]$
```

Figure 14: Проверка работы моей программы

5 Выводы

Выполнив эту лабораторную работу, я приобрёл навыки написания программ с использованием циклов и обработкой аргументов командной строки.

Список литературы