Отчёт по лабораторной работе №8

Дисциплина: Архитектура компьютера

София Андреевна Кудякова

Содержание

Список иллюстраций

1 Цель работы

Цель данной лабораторной работы - научиться писать программы с использованием циклов и обработкой аргументов командной строки.

2 Задания

1. Реализация циклов в NASM
2. Обработка аргументов командной строки
3. Выполнение заданий для самостоятельной 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. 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) и 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.

```
push -10 ; Поместить -10 в стек
push ebx ; Поместить значение регистра ebx в стек
push [buf] ; Поместить значение переменной buf в стек
push word [ax] ; Поместить в стек слово по адресу в ax
```

Существует ещё две команды для добавления значений в стек. Это команда

pusha, которая помещает в стек содержимое всех регистров общего назначения в следующем порядке: ax, cx, dx, bx, sp, bp, si, di. А также команда pushf, которая служит для перемещения в стек содержимого регистра флагов. Обе эти команды не имеют операндов. Команда pop извлекает значение из стека, т.е. извлекает значение из ячейки памяти, на которую указывает регистр esp, после этого уменьшает значение регистра esp на 4. У этой команды также один операнд, который может быть регистром или переменной в памяти. Нужно помнить, что извлечённый из стека элемент не стирается из памяти и остаётся как “мусор”, который будет перезаписан при записи нового значения в стек.

```
pop eax ; Поместить значение из стека в регистр eax
pop [buf] ; Поместить значение из стека в buf
pop word[si] ; Поместить значение из стека в слово по адресу в si
```

Для организации циклов существуют специальные инструкции. Для всех инструкций максимальное количество проходов задаётся в регистре ecx. Наиболее простой является инструкция loop. Она позволяет организовать безусловный цикл, типичная структура которого имеет следующий вид:

```
mov ecx, 100 ; Количество проходов
NextStep:
...
... ; тело цикла
...
loop NextStep ; Повторить `ecx` раз от метки NextStep
```

Иструкция loop выполняется в два этапа. Сначала из регистра ecx вычитается единица и его значение сравнивается с нулём. Если регистр не равен нулю, то выполняется переход к указанной метке. Иначе переход не выполняется и управление передаётся команде, которая следует сразу после команды loop.

4 Выполнение лабораторной работы

4.1 Реализация циклов в NASM

Ввожу команду mkdir, с помощью которой создаю директорию, в которой буду создавать файлы. Перехожу в нее. С помощью команды touch создаю файл lab8-1.asm. (рис. 4.1).

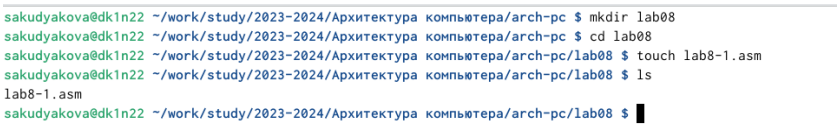

Рис. 4.1: Создание директории и файла

Открываю созданный файл в редакторе и вставляю в него программу вывода значений регистра ecx. (рис. 4.2).

```
%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
```

Рис. 4.2: Редактирование файла

Создаю исполняемый файл и запускаю его. (рис. 4.3).

```
sakudyakova@dk1n22 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ nasm -f elf lab8-1.asm
sakudyakova@dk1n22 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
sakudyakova@dk1n22 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ./lab8-1
Введите N: 4
4
3
2
1
```

Рис. 4.3: Запуск программы файла

Программа выводит числа от N до 1.

Изменяю текст программы, добавив изменение значение регистра ecx в цикле. (рис. 4.4).

```
%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:
sub ecx,1 ; `ecx=ecx-1`
mov [N],ecx
mov eax,[N]
call iprintLF
loop label
call quit
```

Рис. 4.4: Редактирование файла

Создаю исполняемый файл и запускаю его. (рис. 4.5).

```
sakudyakova@dk1n22:~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08
4294731468
4294731466
4294731464
4294731462
4294731460
4294731458
4294731456
4294731454
4294731452
4294731450
4294731448
4294731446
4294731444
4294731442
4294731440
4294731438
4294731436
4294731434
4294731432
4294731430
4294731428
4294
```

Рис. 4.5: Запуск программы файла

Число проходов цикла не соответствует значению N, введенному с клавиатуры.

Измененяю текст программы, добавив команды push и pop (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла loop.(рис. 4.6).

```
%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:
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
call quit
```

Рис. 4.6: Редактирование файла

Создаю исполняемый файл и запускаю его. (рис. 4.7).

```
sakudyakova@dk1n22 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ nasm -f elf lab8-1.asm
sakudyakova@dk1n22 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
sakudyakova@dk1n22 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ./lab8-1
Введите N: 4
3
2
1
0
sakudyakova@dk1n22 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $
```

Рис. 4.7: Запуск программы файла

В данном случае число проходов цикла соответсвует значению N, введенному с клавиатуры. Программа выводит числа от N до 0.

4.2 Обработка аргументов командной строки

Создаю новый файл lab8-2.asm в каталоге ~/work/arch-pc/lab08. (рис. 4.8).

```
sakudyakova@dk1n22 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ touch lab8-2.asm
sakudyakova@dk1n22 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $
```

Рис. 4.8: Создание файла

Ввожу в файл текст программы, выводящая на экран аргументы командной строки. (рис. 4.9).

```
%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
```

Рис. 4.9: Редактирование файла

Создаю исполняемый файл и запускаю его, указав аргументы. (рис. 4.10).

```
sakudyakova@dk1n22 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $  nasm -f elf lab8-2.asm
sakudyakova@dk1n22 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ld -m elf_i386 -o lab8-2 lab8-2.o
sakudyakova@dk1n22 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
```

Рис. 4.10: Запуск программы файла

Программа обработала 4 аргумента.

Создаю новый файл lab8-3.asm в каталоге ~/work/arch-pc/lab08. (рис. 4.11).

```
sakudyakova@dk1n22 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ touch lab8-3.asm
sakudyakova@dk1n22 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $
```

Рис. 4.11: Создание файла

Ввожу в файл текст программы вычисления суммы аргументов командной строки. (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, 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
```

Рис. 4.12: Редактирование файла

Создаю исполняемый файл и запускаю его, указав аргументы.(рис. 4.13).

```
sakudyakova@dk1n22 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $  nasm -f elf lab8-3.asm
sakudyakova@dk1n22 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
sakudyakova@dk1n22 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ./lab8-3 12 13 7 10 5
Результат: 47
```

Рис. 4.13: Запуск программы файла

Программа отработала корректно.

Изменяю текст программы для вычисления произведения аргументов командной строки. (рис. 4.14).

```
%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
mov 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. 4.14: Редактирование файла

Создаю исполняемый файл и запускаю его, указав аргументы. (рис. 4.15).

```
sakudyakova@dk1n22 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $  nasm -f elf lab8-3.asm
sakudyakova@dk1n22 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
sakudyakova@dk1n22 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ./lab8-3 12 13 7 10 5
Результат: 54600
```

Рис. 4.15: Запуск программы файла

Программа верно посчитала произведение аргументов.

Листинг 4.2.1. Программы для вычисления произведения аргументов командной строки.

```
%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
mov 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
```

4.3 Выполнение заданий для самостоятельной работы

Создаю файл lab8-4.asm, в котором буду писать программу. (рис. 4.16).

```
sakudyakova@dk1n22 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ touch lab8-4.asm
sakudyakova@dk1n22 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $
```

Рис. 4.16: Создание файла

Пишу программу которая находит сумму значений функции ⊠(⊠) для⊠ = ⊠1, ⊠2, …, ⊠⊠, т.е. программа должна выводить значение f(x1) + f(x2) + … + f(xn). Значения xi передаются как аргументы. Вид функции f(x) выбрать в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Мой вариант 14 - 7(x+1).(рис. 4.17).

```
%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
SECTION .text
global _start
_start:
pop ecx
pop edx
sub ecx,1
mov esi, 0
mov edi,7
next:
cmp ecx,0h
jz _end
pop eax
call atoi
add eax,1
mul edi
add esi,eax
loop next
_end:
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit

```

Рис. 4.17: Написание программы

Создаю исполняемый файл и запускаю его. (рис. 4.18).

```
sakudyakova@dk1n22 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $  nasm -f elf lab8-4.asm
sakudyakova@dk1n22 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ld -m elf_i386 -o lab8-4 lab8-4.o
sakudyakova@dk1n22 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ./lab8-4 1 2 3
Результат: 63
sakudyakova@dk1n22 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ./lab8-4 5 7 9 4
Результат: 203
sakudyakova@dk1n22 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ./lab8-4 9 8 7 6
Результат: 238
sakudyakova@dk1n22 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $
```

Рис. 4.18: Исполнение файла

Программа отработала верно.

Листинг 4.2.2. Программа для нахождения суммы значений функций

```
%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
SECTION .text
global _start
_start:
pop ecx
pop edx
sub ecx,1
mov esi, 0
mov edi,7
next:
cmp ecx,0h
jz _end
pop eax
call atoi
add eax,1
mul edi
add esi,eax
loop next
_end:
```

```
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit
```

5 Выводы

В ходе данной лабораторной работы я научилась писать программы с использованием циклов и обработкой аргументов командной строки.

Список литературы

Архитектура ЭВМ