Отчет по лабораторной работе №8

Дисциплина архитектура компьютера

Ахатов Эмиль Эрнстович

Содержание

Список иллюстраций

1 Цель работы

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

2 Задание

1. Программа вывода значений регистра ecx
2. Программа выводящая на экран аргументы командной строки
3. Программа вычисления суммы аргументов командной строки
4. 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 ре- гистре esp (указатель стека). Противоположный конец стека называется дном. Значение, помещённое в стек последним, извлекается первым. При помещении значения в стек указа- тель стека уменьшается, а при извлечении — увеличивается. Для стека существует две основные операции: • добавление элемента в вершину стека (push); • извлечение элемента из вершины стека (pop). Добавление элемента в стек. 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. Примеры:

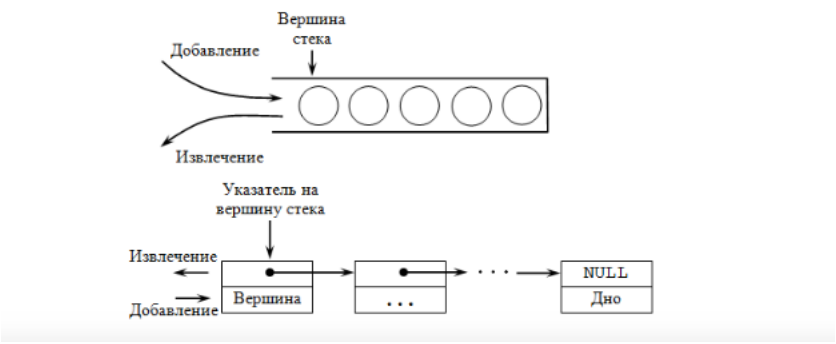

Рис. 3.1: Организация стека в процессоре

```
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.

Извлечение элемента из стека. 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. Примеры:

```
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
```

Аналогично команде записи в стек существует команда popa, которая восстанавливает из стека все регистры общего назначения, и команда popf для перемещения значений из вершины стека в регистр флагов.

Инструкции организации циклов Для организации циклов существуют специальные инструкции. Для всех инструкций максимальное количество проходов задаётся в регистре ecx. Наиболее простой является ин- струкция 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 NASM

Создаю каталог lab8, перехожу в него,создаю файл lab8-1.asm. Я скопировал внешний файл в созданный ката- лог, ввёл текст программы.

```
%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`
```

Рис. 4.1: Редактирование файла

Проверил программу на корректную работу,программа работает некоррект- но,выводит бесконечное колличество значений

Рис. 4.2: Запуск исполняемого файла

Изменяю текст программы добавив изменение значения регистра ecx в цикле

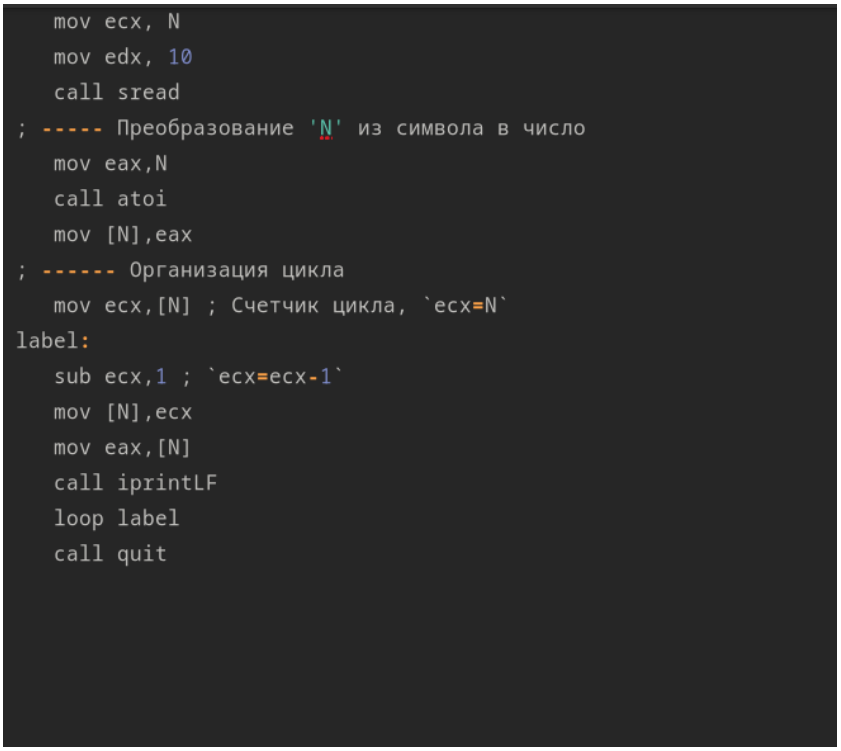

Рис. 4.3: Редактирование файла

запускаю программу,теперь она работает верно

Рис. 4.4: Запуск исполняемого файла

В данной программе регистр ecx принимает значения от N−1N−1 до 00, где NN — это значение, введённое пользователем с клавиатуры. ecx устанавливается равным введённому значению N, т.е. ecx = N. В каждом проходе цикла ecx уменьшается на 1 с помощью sub ecx,1. Команда loop label проверяет значение ecx. Если ecx не равно нулю, то выполняется следующий проход цикла. Когда ecx достигает нуля, выполнение цикла завершается. Нет, число проходов цикла не соответствует значению N, введённому с клавиатуры, так как команда loop повторяет цикл ровно N/2 раз, начиная с N и заканчивая 0. Вношу изменения в текст программы добавив команды push и pop для сохранения значения счетчика цикла loop

```
   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. 4.5: Редактирование файла

Теперь программа работает верно, делает ровно N проходов

```
Введите N: 20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
```

Рис. 4.6: Запуск исполняемого файла

4.2 Обработка аргументов командной строки

Создаю файл lab8-2.asm и ввожу в него текст программы из листинга 8.2

```
%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. 4.7: Редактирование файла

Создал исполняемый файл и запустил его, указав аргументы,программой был обработан каждый аргумент

```
emil@fedora:~/study_2024-2025_arhpc/lab08$ nasm -f elf lab8-2.asm
emil@fedora:~/study_2024-2025_arhpc/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
emil@fedora:~/study_2024-2025_arhpc/lab08$ ./lab8-2 1 2 3
1
2
3
```

Рис. 4.8: Запуск исполняемого файла

Создал файл lab8-3.asm и ввожу в него текст программы из листинга 8.3

```
   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. 4.9: Редактирование файла

Создал исполняемый файл и запустил его, указав аргументы,программой был обработан каждый аргумент

```
emil@fedora:~/study_2024-2025_arhpc/lab08$ nasm -f elf lab8-3.asm
emil@fedora:~/study_2024-2025_arhpc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
emil@fedora:~/study_2024-2025_arhpc/lab08$ ./lab8-3 12 13 7 10 5
Результат: 47
emil@fedora:~/study_2024-2025_arhpc/lab08$
```

Рис. 4.10: Запуск исполняемого файла

Изменил текст программы из листинга 8.3 для вычисления произведения аргументов командной строки

```
%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
SECTION .text
global _start
_start:
   pop ecx            ; Извлекаем количество аргументов в `ecx`
   pop edx            ; Извлекаем имя программы из стека в `edx`
   sub ecx, 1         ; Уменьшаем `ecx` на 1 (чтобы не учитывать имя программы)
   mov esi, 1         ; Инициализируем `esi` значением 1 для умножения

next:
   cmp ecx, 0         ; Проверяем, есть ли ещё аргументы
   jz _end            ; Если аргументов нет, выходим из цикла (переход к `_end`)
   pop eax            ; Извлекаем следующий аргумент из стека
   call atoi          ; Преобразуем символ в число
   imul esi, eax      ; Умножаем промежуточное произведение `esi = esi * eax`
   loop next          ; Переход к обработке следующего аргумента

_end:
   mov eax, msg       ; Выводим сообщение "Результат: "
```

Рис. 4.11: Редактирование файла

Рис. 4.12: Запуск исполняемого файла

Программа работает верно

5 Выполнение заданий для самостоятельной работы

Создаю файл lab8-4.asm,пишу программу соответственно своему варианту - 12.

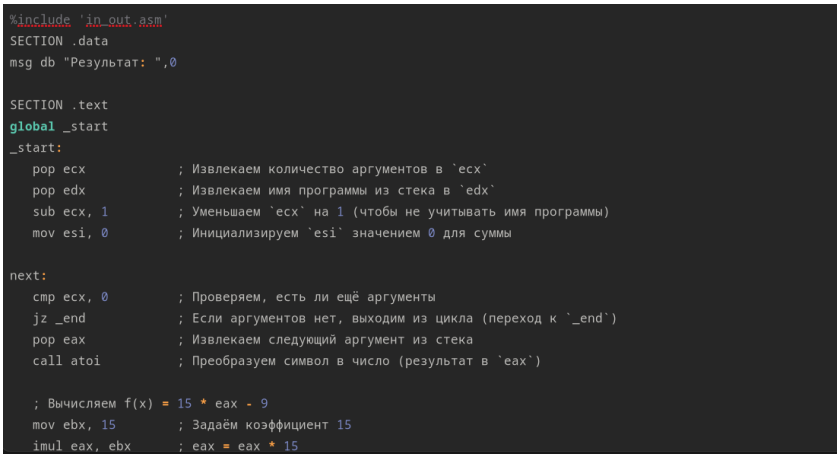

Рис. 5.1: Редактирование файла

Рис. 5.2: Запуск исполняемого файла

Программа работает корректно

6 Выводы

Я приобрел навыки написания программ с использованием циклов и обработкой аргументов командной строки.