РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 8

дисциплина: Архитектура компьютера

Студент: Ниемек Яи Жак

Группа: НММБд-04-24

МОСКВА

2025_ г.

Цель работы

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

Задание

Напишите программу, которая находит сумму значений функции ﷺ(ﷺ) для ﷺ = ﷺ1, ﷺ2, …, ﷺﷺ, т.е. программа должна выводить значение ﷺ(ﷺ1) + ﷺ(ﷺ2) + … + ﷺ(ﷺﷺ). Значения ﷺﷺ передаются как аргументы. Вид функции ﷺ(ﷺ) выбрать из таблицы 8.1 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу на нескольких наборах ﷺ = ﷺ1, ﷺ2, …, ﷺﷺ.

Теоретическое введение

Стек — это структура данных, организованная по принципу LIFO («Last In — First Out» или «последним пришёл — первым ушёл»). Стек является частью архи□тектуры процессора и реализован на аппаратном уровне. Для работы со стеком в процессоре есть специальные регистры (ss, bp, sp) и команды. Основной функци□ей стека является функция сохранения адресов возврата и передачи аргументов при вызове процедур. Кроме того, в нём выделяется память для локальных пере□менных и могут временно храниться значения регистров. На рис. 8.1 показана схема организации стека в процессоре. Стек имеет вершину, адрес последне□го 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. 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. 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 Анало□гично команде записи в стек существует команда popa, которая восстанавливает из стека все регистры общего назначения, и команда popf для перемещения зна□чений из вершины стека в регистр флагов. Для организации циклов существуют специальные инструкции. Для всех инструкций максимальное количество про□ходов 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.

Выполнение лабораторной работы

```
nyemeckyai@fedora:~$ cd work
nyemeckyai@fedora:~/work$ cd arch-pc
nyemeckyai@fedora:~/work/arch-pc$ mkdir lab08
nyemeckyai@fedora:~/work/arch-pc$ cd lab08
nyemeckyai@fedora:~/work/arch-pc/lab08$ touch lab8-1.asm
nyemeckyai@fedora:~/work/arch-pc/lab08$ ls
lab8-1.asm
nyemeckyai@fedora:~/work/arch-pc/lab08$
```

mc [nyemeckyai@fedora]:~/work/arch-pc/lab08 — /usr/bin/mc -P /var/tmp/m
~/work/arch-pc/lab08

```
lab8-1.asm          [----]  0 L:[  1+ 0   1/ 30] *(0    / 638b) 0037 0x025
%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
```

```
nyemeckyai@fedora:~/work/arch-pc/lab08$ nano lab8-1.asm
nyemeckyai@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
nyemeckyai@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
nyemeckyai@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 6
6
5
4
3
2
1
nyemeckyai@fedora:~/work/arch-pc/lab08$
```

```
nyemeckyai@fedora:~/work/arch-pc/lab08$ touch lab8-2.asm
nyemeckyai@fedora:~/work/arch-pc/lab08$ ls
in_out.asm  lab8-1  lab8-1.asm  lab8-1.o  lab8-2.asm
nyemeckyai@fedora:~/work/arch-pc/lab08$ nano lab8-2.asm
nyemeckyai@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-2.asm
nyemeckyai@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
nyemeckyai@fedora:~/work/arch-pc/lab08$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
nyemeckyai@fedora:~/work/arch-pc/lab08$
```

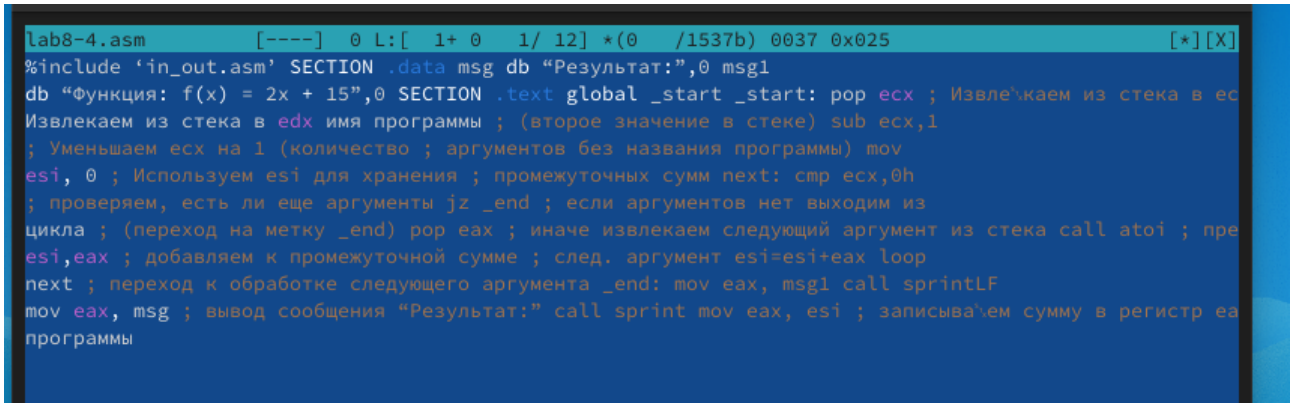
```
lab8-2.asm          [----]  0 L:[  1+20  21/ 21] *(944 / 944b) <EOF>
%include 'in_out.asm'
SECTION .text
global _start
_start:
pop ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в `edx` имя программы
; (второе значение в стеке)
sub 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
```

```
nyemeckyai@fedora:~/work/arch-pc/lab08$ nano touch lab8-3.asm
nyemeckyai@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
nyemeckyai@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
nyemeckyai@fedora:~/work/arch-pc/lab08$ ./lab8-2 12 13 7 10 5
12
13
7
10
5
nyemeckyai@fedora:~/work/arch-pc/lab08$ ./lab8-3 12 13 7 10 5
Результат: 47
nyemeckyai@fedora:~/work/arch-pc/lab08$
```

mc [nyemeckyai@fedora]:~/work/arch-pc/lab08 — /usr/bin/mc -P /var/tmp/
~/work/arch-pc/lab08

```
lab8-3.asm          [----]  0 L:[  1+ 0   1/ 31] *(0    /1430b) 0037 0x0
%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
```

```
nyemeckyai@fedora:~/work/arch-pc/lab08$ nano lab8-4.asm
nyemeckyai@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-4.asm
```



Выводы

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

1. Опишите работу команды `loop`.

Команда `loop` в ассемблере используется для организации цикла с учётом декремента счётчика. Она выполняет следующее:

- Понижает значение регистра **ECX** (счётчик).
- Если ECX не равен нулю, то выполняется переход на метку, указанную в команде `loop`. Если значение ECX равно нулю, выполнение программы продолжается с инструкции, следующей за командой `loop`.

Пример:

```
mov ecx, 5      ; Инициализируем счётчик цикла (5 итераций)
loop_start:
; Тело цикла
dec ecx         ; Уменьшаем ECX на 1
loop loop_start ; Если ECX != 0, повторить цикл
```

В данном примере цикл будет повторяться 5 раз. Команда `loop` работает, пока значение регистра ECX не станет равным нулю.

2. Как организовать цикл с помощью команд условных переходов, не прибегая к специальным командам управления циклами?

Цикл можно организовать, используя команды условных переходов, такие как `cmp`, `jne` (не равно), `je` (равно) и другие, в сочетании с командой для уменьшения счётчика (например, `dec` или `sub`). Пример:

```
mov ecx, 5           ; Инициализация счётчика цикла
loop_start:
; Тело цикла
dec ecx              ; Уменьшаем счётчик
cmp ecx, 0           ; Сравниваем с нулём
jne loop_start       ; Если ECX != 0, повторить цикл
```

В данном примере мы явно контролируем цикл через условный переход `jne`, который продолжает цикл, пока значение ECX не станет равным нулю.

3. Дайте определение понятия «стек».

Стек — это структура данных, которая работает по принципу **LIFO** (Last In, First Out), то есть последний элемент, который был добавлен в стек, первым извлекается. Стек используется для временного хранения данных, таких как адреса возврата из подпрограмм, локальные переменные и промежуточные значения.

Основные операции стека:

- **Push**: добавление элемента в стек.
- **Pop**: извлечение элемента из стека.
- **Peek**: просмотр верхнего элемента без его извлечения.

Стек используется в большинстве программных систем для управления вызовами функций и рекурсией.

4. Как осуществляется порядок выборки содержащихся в стеке данных?

Порядок выборки данных из стека осуществляется по принципу **LIFO** (Last In, First Out):

- **Последний добавленный элемент** извлекается первым.
- Когда команда `push` добавляет элемент в стек, он помещается на верх стека.
- Когда команда `pop` извлекает элемент, она удаляет верхний элемент из стека и возвращает его.

Например:

```
push eax       ; Помещает значение регистра EAX в стек
pop ebx        ; Извлекает верхний элемент стека в регистр EBX
```

Таким образом, порядок извлечения данных строго обратен порядку их добавления, что и делает стек полезным для сохранения состояния программ (например, при вызове подпрограмм).