Санкт-Петербургский государственный политехнический
университет Институт компьютерных наук и кибербезопасности

КУРСОВАЯ РАБОТА

Программирование на ассемблере

По дисциплине «Архитектура ЭВМ. Часть 1»

Выполнил студент гр. 5130904/30008 Ребдев П.А.

Руководитель

проф. д.т.н. Милицын А.В.

Санкт-Петербург
2024

Оглавление

Введение

Ассемблер - низкоуровневый машинно-ориентированный язык программирования. Реализация языка зависит от типа процессора и определяется архитектурой вычислительной системы. Ассемблер позволяет напрямую работать с аппаратурой компьютера. Программа на языке ассемблера включает в себя набор команд, которые после трансляции преобразуются в машинные команды.

Программа 1

Разработать набор процедур работы с очередью, а именно:

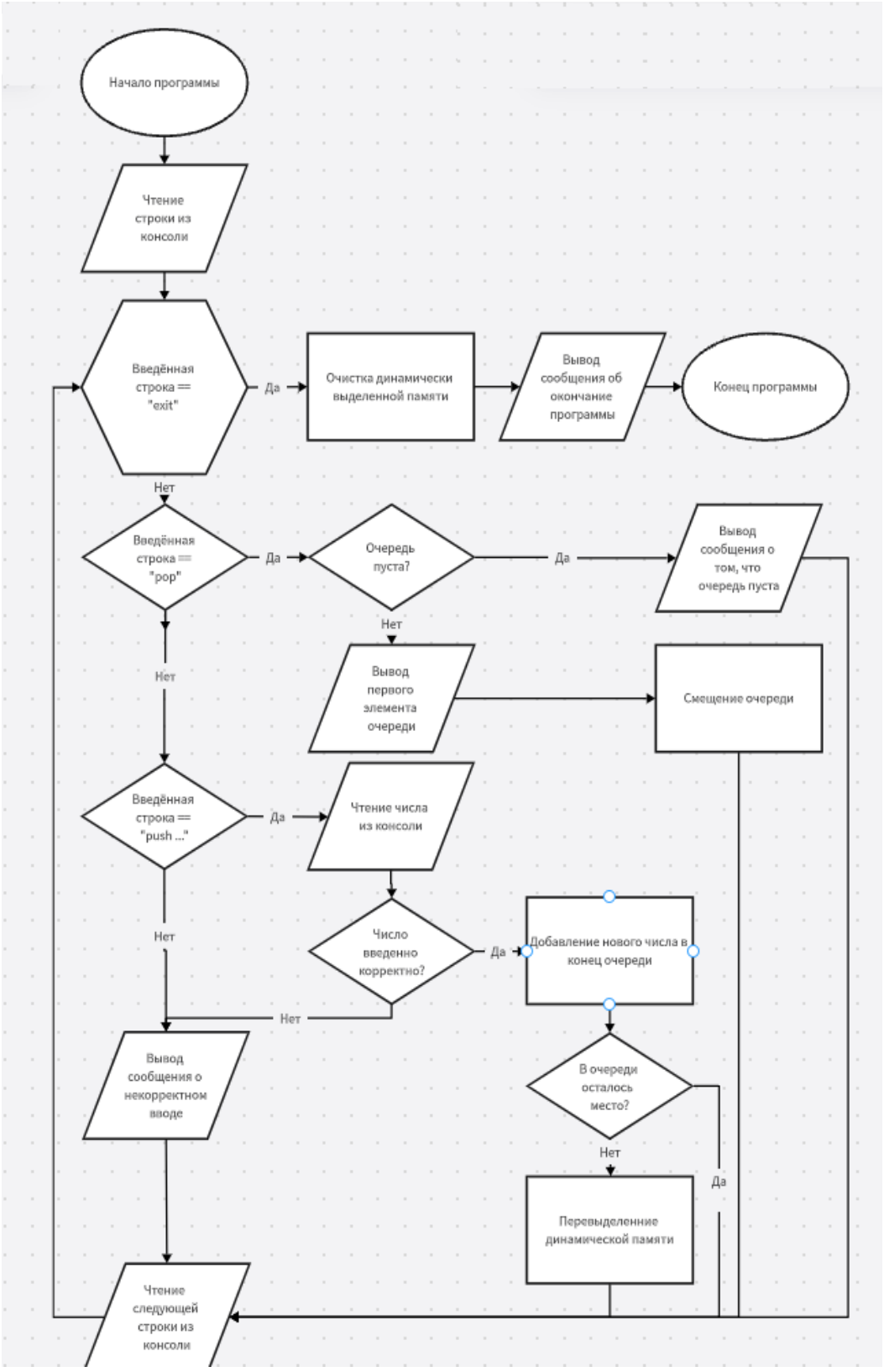
- включение нового элемента;

- выборка очередного элемента (со сдвигом очереди)

Разработать демо-программу.

Программа 2 "БудильникЗ". Задайте с клавиатуры время относительно текущего времени (например, 2 минуты). На экран выводится время, оставшееся до звонка. При наступлении заданного времени выдайте звуковой сигнал.

Программа 1
Блок-схема



Текст программы

```
#include <iostream>
#include <string>

int main()
{
  std::cout << "\033[1;33mQueue programm had started!\nYou have next options: PUSH NUM / POP\
nTo exit print \"exit\"\033[0m\n";
  std::string input;
  std::getline(std::cin, input);
  size_t size = 100;
  long long * queue = new long long[size]; //Выделение динамической памяти для очереди
  long long head = 0;

  while (input.find("exit") == std::string::npos) //Пока пользователь не введёт «exit» продолжать
работу
  {
    if (input.find("pop") != std::string::npos)
    {
      if (head == 0) //Если очередь пуста, вывести соответствующее сообщение
      {
        std::cout << "\033[1;31mQueue is empty!\033[0m\n";
      }
      else
      {
        std::cout << *queue << '\n'; //Если очередь не пуста, то выводим первый элемент
        __asm__ //Смещение очереди
        (
          "leaq %0, %%rsi\n"
          "subq $1, (%%rsi)\n"
          "leaq %1, %%rsi\n"
          "movq %0, %%rcx\n"

          "LoopMain:\n"
            "jrcxz exit\n"
            "movq 8(%%rsi), %%rax\n"
            "movq %%rax, (%%rsi)\n"
            "addq $8, %%rsi\n"
          "loop LoopMain\n"
          "exit:\n"
          ://list of output parameters
          :"m" (head), "m"(*queue)//list of input parameters
          :"rsi", "rcx", "rax"//list of using registers
        );
```

```
    }
  }
  else if(input.find("push") != std::string::npos)
  {
    try
    {
      long long in = std::stoll(input.substr(input.find("push") + 5)); //Пытаемся вычленить число из введённой строки
      __asm__ //Добавляем число в конец очереди
      (
        "leaq %0, %%rsi\n"
        "movq %1, %%rax\n"
        "movq $8, %%rdx\n"
        "mulq %%rdx\n"
        "addq %%rax, %%rsi\n"
        "movq %2, %%rax\n"
        "movq %%rax, (%%rsi)\n"
        "leaq %1, %%rsi\n"
        "addq $1, (%%rsi)\n"
        ://list of output parameters
        :"m" (*queue), "m"(head), "m"(in)//list of input parameters
        :"rsi", "rax", "rdx"//list of using registers
      );
      if (head == size) //Если в очереди максимальное количество элементов, то перевыделяем динамическую память
      {
        long long * q = new long long[size * 2];
        for (size_t i = 0; i < size; ++i)
        {
          q[i] = queue[i];
        }
        delete[] queue;
        queue = q;
        q = nullptr;
        size *= 2;
      }
      std::cout << "\033[1;32m" << in << " successfull added to queue\033[0m\n"; //Сообщение об успешном добавление элемента
    }
    catch(...)
    {
      std::cout << "\033[1;31mBad push input!\033[0m\n"; //В случае некорректного ввода, выводим ошибку
    }
  }
```

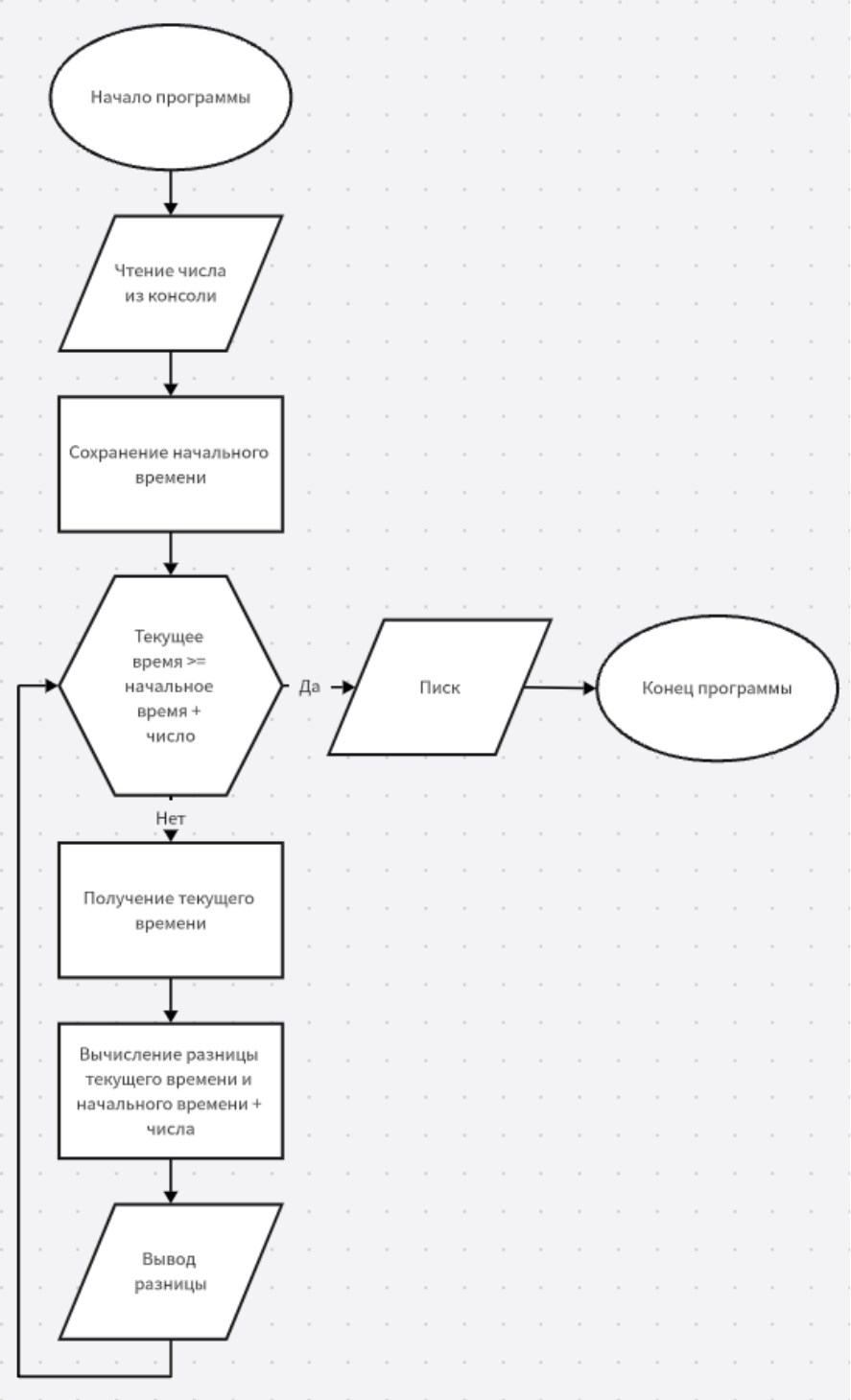
```
    else if(input.find("show") != std::string::npos)
    {
      for(size_t i = 0; i < head; ++i)
      {
        std::cout << *(queue + i) << ' ';
      }
      std::cout << '\n';
    }
    else
    {
      std::cout << "\033[1;31mBad input!\033[0m\n"; //В случае ввода неизвестной команды, выводим ошибку
    }
    std::getline(std::cin, input); //Получаем следующую строку
  }

  std::cout << "\033[1;32mProgramm is ended\033[0m\n"; //Выводим сообщение о завершение программы
  delete[] queue; //Очищаем динамическую память
  return 0;
}
```

Скриншоты

```
pavel@debian:~/Desktop/PolikekWorks/My-learning-repo/Sem3/Computer acrhitector/courseWork$ make crun n="queueWithASM"
make cbuild n=""queueWithASM""
make[1]: вход в каталог «/home/pavel/Desktop/PolikekWorks/My-learning-repo/Sem3/Computer acrhitector/courseWork»
grep -n -r "dell"
Makefile:5:     grep -n -r "dell"
g++ "queueWithASM".cpp -o "queueWithASM"
make[1]: выход из каталога «/home/pavel/Desktop/PolikekWorks/My-learning-repo/Sem3/Computer acrhitector/courseWork»
./"queueWithASM"
Queue programm had started!
You have next options: PUSH NUM / POP
To exit print "exit"
pop
Queue is empty!
push
Bad push input!
push notNumber
Bad push input!
pop
Queue is empty!
push 12
12 successfull added to queue
push 144
144 successfull added to queue
push 249
249 successfull added to queue
push -214214
-214214 successfull added to queue
pop
12
pop
144
pop
249
pop
-214214
pop
Queue is empty!
exit
Programm is ended
rm "queueWithASM"
```

Программа 2
Блок-схема



Текст программы

```
global _start

section .data ;Создаём переменные, для: записи времени, частоты писка, длительности писка
  curtime dq 0
  freq equ 500
  beep_sec equ 1
  beep_nsec equ 0

section .bss ;Выделяем память для структуры timespec (необходимо для писка)
  sleep_time: resb 16 ; for timespec

section .text
_start:
  ;Считываем строку из консоли и конвертируем её в число
  readNum:
    mov rax, 3 ; systeam read
    mov rbx, 0 ; default input
    mov rcx, number
    mov rdx, 8 ; number of bytes
    int 0x80

    ; convert string to num
    mov rsi, number
    mov rax, 0
    read_loop:
      movzx rdx, byte [rsi] ; put in rdx byte from number
      cmp rdx, 0xA ; compare rdx with '\n'
      je end_read_loop ; if (rdx == '\n') end
      sub rdx, '0'
      imul rax, 10 ; rax = ( rax * 10 ) + rdx
      add rax, rdx
      inc rsi ; go to next number byte
      jmp read_loop
    end_read_loop:
    push rax
  ;Получаем и сохраняем начальное время, добавляем к нему число введённое пользователем
  main:
    ; get current time
    mov rax, 0xc9
    mov rdi, curtime
    syscall
```

```
    mov rdx, [curtime]
    pop rax
    add rdx, rax ; add to timer number from user

    timeloop:
      mov rax, 0xc9
      mov rdi, curtime
      syscall

      mov rax, [curtime]
      ;Проверяем прошло ли с момента последней проверки секунда, если да, то выводим оставшееся время, если нет, то пропускаем этот блок
      cmp rbx, rax
      jae afterPrint ; if a second has passed since the previous check, the output occurs

      print:
        push rax
        push rbx
        push rcx
        push rdx

        mov rbx, qword [curtime] ; rbx is now times
        mov rax, rdx ; rax is start time
        sub rax, rbx ; ras is time diff
        ;Для печати двух и трёх значных чисел, требуется разбивать на цифры и последовательно печатать (142 → 1, 4, 2)
        printBigNum:
          mov rcx, 0
          decomposition:
            mov rbx, 10
            mov rdx, 0
            div rbx                 ; rax = rdx:rax / rbx; rdx = rax % rbx
            push rdx
            add rcx, 1; ++rcx
            cmp rax, 0              ; (rax > 0) ? (ZF = 1) : (ZF = 0)
            jnz decomposition       ; while (ZF != 1) call jmp decomposition
          outputLoop:
            pop rax
            add rax, '0'            ; char(9) is '\t', but char(9 + '0') is '9'
            push rcx
            basePrint:
              mov [number], rax
              mov rax, 4
              mov rbx, 1
```

```
            mov rcx, number
            mov rdx, 1
            int 0x80
           pop rcx
           loop outputLoop

        mov rax, 10 ; put '\n' in rax
        mov [number], rax
        mov rax, 4
        mov rbx, 1
        mov rcx, number
        mov rdx, 1
        int 0x80

        pop rdx
        pop rcx
        pop rbx
        pop rax

        mov rbx, rax

      afterPrint:
        ;Сравниваем, равно ли прошедшее время начальному + числу, если нет,то возвращаемся в начало цикла
        cmp qword [curtime], rdx
        jb timeloop
  ;Производим писк: открываем консоль, и с помощью KIOCSOUND для ioctl генерируем звук
  beep:
    mov rax, 2 ; open
    mov rdi, console_path ; /dev/console
    mov rsi, 0 ; O_RDONLY
    mov rdx, 0
    syscall
    mov r10, rax ; save descriptor in r10

    ; ioctl for sound generation
    mov rax, 16    ; ioctl
    mov rdi, r10    ; /dev/console
    mov rsi, 0x4B2F ; KIOCSOUND for ioctl
    mov rdx, freq   ; frequency
    syscall

    ; timespec
    mov qword [sleep_time], beep_sec
```

```
    mov qword [sleep_time + 8], beep_nsec

    ; pause
    mov rax, 35    ; syscall nanosleep
    mov rdi, sleep_time ; timespec
    mov rsi, 0
    syscall

    ; sound off
    mov rax, 16    ; ioctl
    mov rdi, r10    ; /dev/console
    mov rsi, 0x4B2F ; KIOCSOUND for ioctl
    mov rdx, 0      ; frequency adress 0
    syscall

    ; close /dev/console
    mov rax, 3    ; close
    mov rdi, r10
    syscall
  ;Завершаем программу
  exit:
    mov rdi, 0
    mov rax, 60
    syscall

section .bss
  number resd 1 ;Переменная, в которой храниться введённое пользователем число

section .data
  console_path db "/dev/console", 0 ;Путь до консоли (необходимо для писка)
```

Список использованных прерываний

INT 0x80 - вызов ядра операционной системы Linux

rax	Описание
3	Ввод данных
0xc9	Получение текущего времени
4	Вывод данных
2	Открытие файла
16	Вызов ioctl
35	Вызов задержек/режим сна
3	Закрытие файла
0	Завершение программы

Скриншоты

```
pavel@debian:~/Desktop/PolikekWorks/My-learning-repo/Sem3/Computer acrhitector/courseWork$ make abuild n="alarmClock"
nasm -f elf64 "alarmClock".asm -o "alarmClock".o
ld -o "alarmClock" "alarmClock".o
pavel@debian:~/Desktop/PolikekWorks/My-learning-repo/Sem3/Computer acrhitector/courseWork$ sudo ./alarmClock
20
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
```

Вывод

В ходе лабораторной работы были реализованы 2 программы для Debian 11 для Linux 64 bit, первая программа была реализована на c++ (g++) с ассемблерными вставками, использующими синтаксис AT & T, вторая была реализована на чистом ассемблере с использованием nasm:

1. **Очередь:**
   Программа является реализацией структуры данных «Очередь» и оболочкой по взаимодействию с этой структурой данных. Использование встроенных ассемблерных вставок позволяет ускорить работу очереди.

2. **"Будильник3":**
   Программа является демонстрацией освоения работы с системными вызовами, в частности с вызовом для получения времени, а так же с вызовом, позволяющим генерировать звуковой сигнал. Программа демонстрирует разносторонние возможности доступные при работе с языком ассемблера

Результаты:

- Изучены методы низкоуровневой работы с динамической памятью.
- Получен опыт по работе с временем и устройствами аудиовыхода на уровне ассемблера
- Подтверждена возможность реализации общепринятых и востребованных структур данных на языке ассемблера.

Заключение:

Работа продемонстрировала возможность применения языка ассемблера для решения реальных задач. Язык ассемблера позволил значительно оптимизировать программы, которые использовали бы тяжеловесные библиотеке, в случае программирования на высокоуровневых языках. Было выявлено, что использование ассемблерных вставок в программе на c++, при понимании программистом происходящего, является довольно простым и очень эффективным методом по оптимизации программы