|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | ***«*Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ \_\_\_\_\_\_\_ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_\_\_\_\_\_КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ОТЧЕТ**

**по домашней работе № 1**

**Дисциплина:** Машинно-зависимые языки и основы компиляции

**Название лабораторной работы:** Обработка символьной информации

Студент гр. ИУ6-45Б **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_**И.А.Дулина**\_\_\_\_\_\_\_**

(Подпись, дата) (И.О. Фамилия)

Преподаватель  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_**С.С.Данилюк\_**\_\_\_\_\_**

(Подпись, дата) (И.О. Фамилия)

Москва, 2024

**Вариант 5.6**

**Цель работы**: изучение команд обработки цепочек и приемов обработки символьной информации.

**Задание**: дан текст, состоящий из 5 слов, состоящих из 7 символов. Расставить слова в соответствии с русским алфавитом.

На рисунке 1 показана схема алгоритма.



Рисунок 1 – Схема алгоритма

В листинге 1 представлен код программы.

Листинг 1 – Код программы

|  |
| --- |
| section .data      FormerMsg db "Изначальный текст: ", 10      lenFormerMsg equ $-FormerMsg      SortedMsg db "Отсортированный текст:", 10      lenSortedMsg equ $-SortedMsg      ErrorStr db "Error: Invalid input format", 10      lenError equ $-ErrorStr      ;Text db "радуйся каждому моменту разного счастья", 10      Text db "ДЛВПРНО АПРЦУПР ПРВАРРВ АРПВАПР ПРОВАПР", 10      lenText equ $-Text      Helper times 10 db " " ;для swap  section .bss  section .text  global \_start  \_start:  ;проверка на ошибку ввода текста      mov eax, lenText      mov ebx, 75      cmp eax, ebx      jne Error  ;вывод сообщения об изначальной строке      mov eax, 4 ; системная функция 4 (write)      mov ebx, 1 ; дескриптор файла stdout=1      mov ecx, FormerMsg ; адрес выводимой строки      mov edx, lenFormerMsg ; длина выводимой строки      int 80h ; вызов системной функции  ;вывод изначальной строки      call printText  ;сортировка пузырьком      mov ecx, 4 ;5-1 чтобы можно было поставить указатель перед последним элементом и просмотреть его      first\_iter:      mov eax, 5 ;всего элементов      sub eax, ecx ;индекс просматриваемого i      mov ebx, 0 ;счётчик для внутреннего цикла, 0 для установки указателя перед первым элементом      xchg eax, ebx ;меняем значения ebx и eax для удобства      push ecx      push eax      second\_iter:          mov ecx, 5 ;не несёт большого смысла, больше служит для продолжения работы внутреннего цикла, проверка и обнуление цикла в конце second\_iter          push ecx          call load ;загрузка элементов a[j] в esi и a[j+1] в edi          mov ecx, 15 ;просматриваем первые 15 байтов 7\*2 + 1 на совпадение          repe cmpsb          jl next ;если a[j]<a[j+1] идём дальше          call swap ;если a[j]>a[j+1] меняем их местами          next:          inc eax ;увеличиваем счётчик second\_iter          pop ecx          sub ecx, ebx ;считаем граничное значение второго итератора N-i          cmp eax, ecx ;сравниваем количество уже проверенных элементов с граничным значением          je exit\_second ;если достигли конца цикла выход          jmp cont          exit\_second: mov ecx, 1 ;выход осуществляем ручным обнулением ecx, при вызове loop ecx станет = 0 и будет произведен выход из цикла          cont: loop second\_iter      pop eax      pop ecx      loop first\_iter  ;вывод сообщения о сортированной строке      mov eax, 4 ; системная функция 4 (write)      mov ebx, 1 ; дескриптор файла stdout=1      mov ecx, SortedMsg ; адрес выводимой строки      mov edx, lenSortedMsg ; длина выводимой строки      int 80h ; вызов системной функции  ;вывод сортированной строки      call printText      jmp Exit  load:      push eax      push ebx      mov ebx, 15      mul ebx ;eax\*15      lea esi, [Text + eax] ;элемент j      lea edi, [Text + eax+ebx] ; элемент j+1      pop ebx      pop eax      ret  swap:      push eax      push ebx      push ecx      push esi      push edi      mov ebx, 15      mul ebx      ; k:=a[j]      mov ecx, 14      lea esi, [Text + eax]      lea edi, [Helper]      rep movsb      ; a[j]:=a[j+1]      mov ecx, 14      lea esi, [Text + eax + ebx]      lea edi, [Text+eax]      rep movsb      ; a[j+1] := k      mov ecx, 14      lea esi, [Helper]      lea edi, [Text+eax+ebx]      rep movsb      pop edi      pop esi      pop ecx      pop ebx      pop eax      ret  printText:      mov eax, 4 ; системная функция 4 (write)      mov ebx, 1 ; дескриптор файла stdout=1      mov ecx, Text ; адрес выводимой строки      mov edx, lenText ; длина выводимой строки      int 80h ; вызов системной функции      ret  Error:      mov eax, 4 ;      mov ebx, 1 ;      mov ecx, ErrorStr ;      mov edx, lenError ;      int 80h ;      jmp Exit  ; выход  Exit:  mov eax, 1 ;  xor ebx, ebx ;  int 80h ; |

В таблице 1 представлены тесты программы.

Таблица 1 – Тесты программы

|  |  |  |  |
| --- | --- | --- | --- |
| № | Исходные данные | Ожидаемый результат | Полученный результат |
| 1 | радуйся каждому моменту разного счастья | каждому моменту радуйся разного счастья | каждому моменту радуйся разного счастья |
| 2 | ДЛВПРНО АПРЦУПР ПРВАРРВ АРПВАПР ПРОВАПР | АПРЦУПР АРПВАПР ДЛВПРНО ПРВАРРВ ПРОВАПР | АПРЦУПР АРПВАПР ДЛВПРНО ПРВАРРВ ПРОВАПР |
| 3 | ДАЛЁКИЙ ЦАПНУТЬ ЗМЕЁНОК ВЕДЁРКО НАДЕЖНО | ВЕДЁРКО ДАЛЁКИЙ ЗМЕЁНОК НАДЕЖНО ЦАПНУТЬ | ВЕДЁРКО ДАЛЁКИЙ ЗМЕЁНОК НАДЕЖНО ЦАПНУТЬ |

На рисунках 2, 3, 4 показаны примеры работы тестов в консоли.

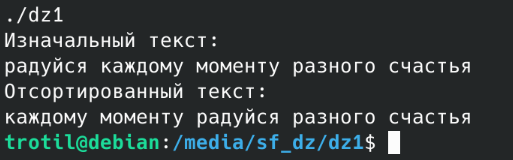


Рисунок 2 – Первый пример работы программы

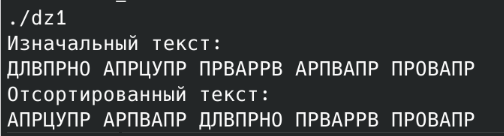


Рисунок 3 – Второй пример работы программы

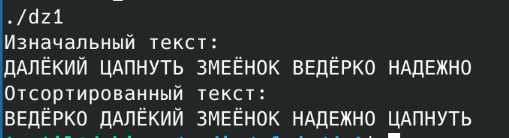


Рисунок 4 – Третий пример работы программы

**Вывод**: были изучены команды обработки цепочек и приемов обработки символьной информации.

**Контрольные вопросы**:

1. Дайте определение символьной строки.

Символьная строка – последовательность байт, находящихся в соседних ячейках памяти

1. Назовите основные команды обработки цепочек?

Основные операции-примитивы и команды, с помощью которых реализуется обработка цепочек:

* пересылка цепочки:
  + movs Адрес\_приемника,Адрес\_источника;
  + movsb;
  + movsw;
  + movsd.
* сравнение цепочек:
  + cmps Адрес\_приемника,Адрес\_источника;
  + cmpsb;
  + cmpsw;
  + cmpsd.
* сканирование цепочки:
  + scas Адрес\_приемника;
  + scasb;
  + scasw;
  + scasd.
* загрузка элемента из цепочки:
  + lods Адрес\_источника;
  + lodsb;
  + lodsw;
  + lodsd.
* сохранение элемента в цепочке:
  + stos Адрес\_приемника;
  + stosb;
  + stows;
  + stosd.

1. Какие операции выполняют строковые команды MOVS? Какие особенности характерны для этих команд?

Префикс повторения rep (REPeat). Префикс rep заставляет данные команды выполняться, пока содержимое ecx/cx не станет равным 0. При этом цепочечная команда, перед которой стоит префикс, автоматически уменьшает содержимое ecx/cx на единицу.

1. Какие операции выполняют строковые команды CMPS, SCAS? Какие особенности характерны для этих команд?

Префиксы повторения repe или repz (REPeat while Equal or Zero). Они заставляют цепочечную команду выполняться до тех пор, пока содержимое ecx/cx не равно нулю или флаг ZF равен 1. Как только одно из этих условий нарушается, управление передается следующей команде программы. Благодаря возможности анализа флага ZF, наиболее эффективно эти префиксы можно использовать для поиска отличающихся элементов цепочек.

Префиксы повторения repne или repnz (REPeat while Not Equal or Zero). Их действие на цепочечную команду несколько отличается от действий префиксов repe/repz. Префиксы repne/repnz заставляют цепочечную команду циклически выполняться до тех пор, пока содержимое ecx/cx не равно нулю или флаг ZF равен нулю. При невыполнении одного из этих условий работа команды прекращается. Данные префиксы также можно использовать для поиска совпадающих элементов цепочек.

1. Как обеспечить циклическую обработку строк?

Для обеспечения циклической обработки строк в ассемблере можно использовать инструкции сравнения строк (CMPS), сканирования строки (SCAS), а также инструкции перехода, такие как jmp, je, jne, jz, jnz, jl, jg и т. д., чтобы управлять потоком выполнения в зависимости от результатов сравнения или других условий. Также можно использовать регистры счетчиков или метки для определения начала и конца цикла.

1. Какова роль флага DF во флажковом регистре при выполнении команд обработки строк?

Флаг DF определяет знак модификации регистров, адресующих операнды.

Если DF = 0, то значение индексных регистров esi/si и edi/di будет автоматически увеличиваться (операция инкремента) цепочечными командами, то есть обработка будет осуществляться в направлении возрастания адресов.

Если DF = 1, то значение индексных регистров esi/si и edi/di будет автоматически уменьшаться (операция декремента) цепочечными командами, то есть обработка будет идти в направлении убывания адресов.

Состоянием флага DF можно управлять с помощью двух команд, не имеющих операндов: cld (Clear Direction Flag) — очистить флаг направления, команда сбрасывает флаг направления DF в 0; std (Set Direction Flag) — установить флаг направления, команда устанавливает флаг направления DF в 1.

1. Как правильно выбрать тестовые данные для проверки алгоритма обработки строки?

Для выбора тестовых данных для проверки алгоритма обработки строки необходимо учитывать различные краевые случаи и типы входных данных. Важно включить тестовые данные, содержащие пустые строки, строки разной длины, строки с разными символами, а также строки, представленные разными кодировками. Также полезно включить тестовые данные, которые вызывают срабатывание различных условий в алгоритме, таких как строки совпадающие, несовпадающие и т. д.