|  |  |
| --- | --- |
| 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 (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

**Отчет**

|  |  |
| --- | --- |
| **по домашнему заданию №** | **1** |

|  |  |
| --- | --- |
| **Название:** | Обработка символьной информации |
| **Дисциплина:** | Машинно-зависимые языки и основы компиляции |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-42Б |  |  | Д.В. Сулейманов |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | М.В. Широкова |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2022

**Вариант 2.24**

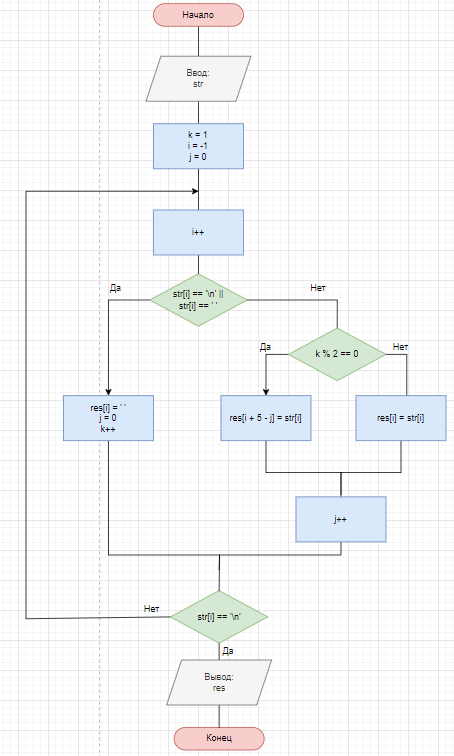
*Цель работы:*

Изучение команд обработки цепочек и приёмов обработки символьной информации

*Задание:* Дан текст 8 слов по 6 символов. В словах с четным номером изменить порядок букв на обратный.

*Ход работы:*

Составим схему алгоритма программы с учётом поставленной задачи. (смотри рисунок 1).



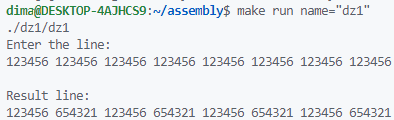
*Рисунок 1. - схема алгоритма ввода матрицы*

Реализуем схему алгоритма на ассемблере (смотри листинг 1.1).

*Листинг 1.1 – текст программы*

1. %include "./lib64.asm"
3. %macro write\_string 2
4. ; вывод
5. ; 1 - адрес строки, 2 - длина строки
6. mov     rax, 1          ; системная функция 1 (write)
7. mov     rdi, 1          ; дескриптор файла stdout=1
8. mov     rsi, %1         ; адрес выводимой строки
9. mov     rdx, %2         ; длина строки
10. syscall                 ; вызов системной функции
11. %endmacro
13. %macro read\_string 2
14. ; ввод
15. ; 1 - буфер ввода, 2 - длина буфера ввода
16. mov     rax, 0          ; системная функция 0 (read)
17. mov     rdi, 0          ; дескриптор файла stdin=0
18. mov     rsi, %1         ; адрес вводимой строки
19. mov     rdx, %2         ; длина строки
20. syscall                 ; вызов системной функции
21. %endmacro
23. %macro StrToInt 1
24. ; перевод string в integer
25. ; rsi должен содержать адрес строки для преобразования
26. call    StrToInt64          ; вызов процедуры
27. cmp     rbx, 0              ; сравнение кода возврата
28. jne     StrToInt64.Error    ; обработка ошибки
29. mov     %1, eax
30. %endmacro
32. %macro IntToStr 2
33. ; перевод integer в string
34. mov     rsi, %2
35. mov     eax, %1             ; получение числа из памяти
36. cwde
37. call    IntToStr64          ; вызов процедуры
38. cmp     rbx, 0              ; сравнение кода возврата
39. jne     StrToInt64.Error    ; обработка ошибки
40. %endmacro
42. section .data           ; сегмент инициализированных переменных
43. InputMsg    db      "Enter the line:", 10
44. lenInput    equ     $-InputMsg
45. OutputMsg   db      "Result line:"
46. lenOutput   equ     $-OutputMsg
47. newLine     db      10
49. section .bss            ; сегмент неинициализированных переменных
50. InBuf       resb    10          ; буфер для вводимой строки
51. lenIn       equ     $-InBuf     ; длина буфера для вводимой строки
52. result      resb    56
53. lenRes      equ     $-result
55. section .text           ; сегмент кода
56. global \_start
58. \_start:
60. write\_string InputMsg, lenInput
62. sub rsp, 56
63. read\_string rsp, 56     ; вводим строку
65. mov rcx, 0              ; rcx - индекс символа в строке, введенной пользователем
66. mov rbx, 1
67. mov rdx, 0
68. while:
69. cmp byte [rsp + rcx], 32        ; сравниваем символ в строке с пробелом
70. je end\_of\_word
71. cmp byte [rsp + rcx], 10
72. je end\_of\_word
73. not\_space:
74. test rbx, 1
75. jz even
76. odd:
77. lea rdi, [result + rcx]
78. movsb
79. jmp continue\_not\_space
80. even:
81. xor rax, rax
82. mov rax, rcx
83. sub rax, rdx
84. add rax, 5
85. sub rax, rdx
86. lea rdi, [result + rax]
87. movsb
88. continue\_not\_space:
89. inc rdx
90. jmp continue
91. end\_of\_word:
92. lea rdi, [result + rcx]
93. mov al, [rsp + rcx]
94. movsb
96. xor rdx, rdx
97. inc rbx
98. cmp byte [rsp + rcx], 10
99. je break\_while
100. continue:
101. inc rcx                     ; переходим к следующему символу в строке
102. jmp while                   ; переходим к следующей итерации цикла
103. break\_while:
104. xor rbx, rbx
106. write\_string newLine, 1
107. write\_string OutputMsg, lenOutput
109. ; вывод строки
110. write\_string newLine, 1
111. write\_string result, 56
113. mov     rax, 60         ; системная функция 60 (exit)
114. xor     rdi, rdi        ; return code 0
115. syscall                 ; вызов системной функции

Проверим программу на произвольных данных. Пример выполнения программы показан на рисунке 2. Результаты тестирования программы представлены в таблице 1.



*Рисунок 4 – пример выполнения программы*

*Таблица 1 – результаты тестирования*

|  |  |  |
| --- | --- | --- |
| **Исходные данные** | **Ожидаемый результат** | **Полученный результат** |
| 123456 123456 123456 123456 123456 123456 123456 123456 | 123456 654321 123456 654321 123456 654321 123456 654321 | 123456 654321 123456 654321 123456 654321 123456 654321 |
| abcdef abcdef abcdef abcdef abcdef abcdef abcdef abcdef | abcdef fedcba abcdef fedcba abcdef fedcba abcdef fedcba | abcdef fedcba abcdef fedcba abcdef fedcba abcdef fedcba |

*Контрольные вопросы*

1. Дайте определение символьной строки.

Символьная строка – массив байт. Предполагается, что каждый байт может быть расшифрован в символ согласно таблице ASCII.

1. Назовите основные команды обработки цепочек?

Основные команды обработки цепочек:

* команды пересылки цепочек movs,
* сравнения cmps,
* сканирования scas,
* загрузки элементов цепочки в регистр lods,
* сохранения элементов в цепочке stos.

В конце каждой ставится суффикс b, w или d в зависимости от размера элементов цепочки.

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

Команда movs пересылает один элемент цепочки а также модифицирует значения регистров rsi (адресует массив-источник) и rdi (адресует массив-приемник). Размер пересылаемых элементов определяется суффиксом команды.

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

Команда cmps производит сравнение элементов в цепочке-источнике с элементами цепочки-приемника.

Команда scas сравнивает элемент в цепочке-источнике с содержимым регистра al.

Обе команды реализуют вычитание, вследствие чего изменяется состояние регистра флагов, что может быть использовано для дальнейшей логики.

1. Как обеспечить циклическую обработку строк?

Циклическая обработка производится при помощи префиксов повторения rep, repz или repnz. Количество итераций помещается в регистр rcx и уменьшается на 1 при каждом проходе цикла.

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

Флаг DF определяет направление обработки. Если он равен нулю, то направление идёт в сторону старших адресов, то есть содержимое регистров rsi и rdi будет увеличено на размер элемента, если же он равен единице, то направление идёт в сторону младших, то есть содержимое регистров rsi и rdi будет уменьшено на размер элемента.

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

Тестовые данные должны удовлетворять условию поставленной задачи, а также учитывать возможность ввода в верхнем и нижнем регистре, отсутствие введенных значений и т.п.

***Вывод:***в рамках лабораторной работы был изучен процесс обработки символьной информации с помощью языка ассемблера.