|  |  |
| --- | --- |
| 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**

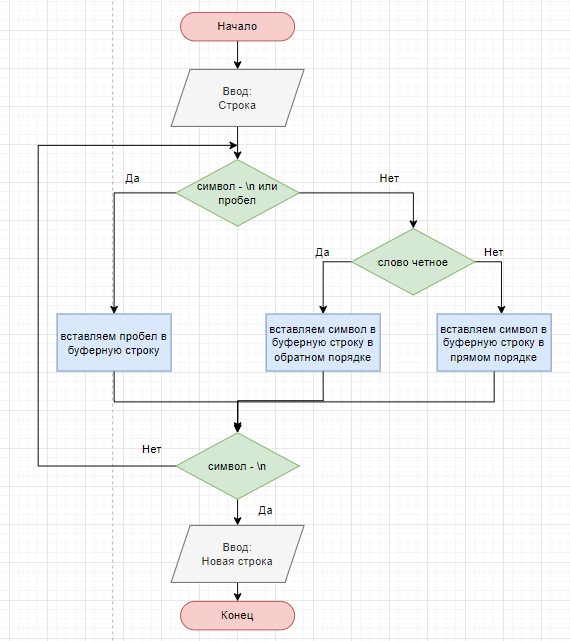
*Цель работы:*

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).



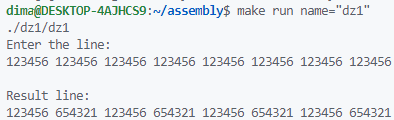
*Рисунок 1. - схема алгоритма ввода матрицы*

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. read\_string rsp, 56     ; вводим строку
64. mov rcx, 0              ; rcx - индекс символа в строке, введенной пользователем
65. mov rbx, 1
66. mov rdx, 0
67. while:
68. cmp byte [rsp + rcx], 32        ; сравниваем символ в строке с пробелом
69. je end\_of\_word
70. cmp byte [rsp + rcx], 10
71. je end\_of\_word
72. not\_space:
73. test rbx, 1
74. jz even
75. odd:
76. lea rdi, [result + rcx]
77. mov al, [rsp + 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. mov al, [rsp + rcx]
88. movsb
89. continue\_not\_space:
90. inc rdx
91. jmp continue
92. end\_of\_word:
93. lea rdi, [result + rcx]
94. mov al, [rsp + rcx]
95. movsb
97. xor rdx, rdx
98. inc rbx
99. cmp byte [rsp + rcx], 10
100. je break\_while
101. continue:
102. inc rcx                     ; переходим к следующему символу в строке
103. jmp while                   ; переходим к следующей итерации цикла
104. break\_while:
105. xor rbx, rbx
107. write\_string newLine, 1
108. write\_string OutputMsg, lenOutput
110. ; вывод строки
111. write\_string newLine, 1
112. write\_string result, 56
114. mov     rax, 60         ; системная функция 60 (exit)
115. xor     rdi, rdi        ; return code 0
116. 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. Почему в ассемблере не определены понятия «массив», «матрица»?

Массив – структурный тип данных. Поскольку ассемблер – язык низкого уровня, программисту предоставляется возможным самому описать такие типы данных, как массив и матрица, используя скалярные типы.

2. Как в ассемблере моделируются массивы?

Массив во внутреннем представлении – это последовательность элементов в памяти.

Программирование обработки выполняется с использованием адресного регистра, в котором хранится либо смещение текущего элемента относительно начала сегмента данных, либо его смещение относительно начала массива. При переходе к следующему элементу и то, и то смещение увеличивают на длину элемента. Если длина элемента отлична от единицы, то можно использовать масштаб.

3. Поясните фрагмент последовательной адресации элементов массива? Почему при этом для хранения частей адреса используют регистры?

При обработке всех элементов двумерного массива используют два цикла, при этом один вложен в другой. Регистры используются для подсчета эффективного адреса элемента, исходя из номера итерации.

4. Как в памяти компьютера размещаются элементы матриц?

Организуя матрицу в памяти, удобнее располагать ее элементы построчно, то есть после последнего элемента первой строки располагается первый элемент второй строки и так далее. Следовательно, элементы матрицы размещаются в памяти компьютера один за другим с интервалом в размер одного элемента.

5. Чем моделирование матриц отличается от моделирования массивов? В каких случаях при выполнении операций для адресации матриц используется один регистр, а в каких – два?

Моделирование массива любой размерности заключается в правильной организации хранения и обработки всех его элементов. Матрица представляет собой двумерный массив, элементы которого удобнее располагать в памяти построчно, поскольку именно в такой последовательности их, скорее всего, будет вводить пользователь.

При прохождении всех элементов матрицы без различения ее строк достаточно использовать один регистр, который будет считать смещение относительно начала массива. Если важно учесть строку или столбец при адресации, можно задействовать два регистра.

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