|  |  |
| --- | --- |
| 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 лабораторной работе №** | **4** |

|  |  |
| --- | --- |
| **Название:** | Обработки массивов и матриц |
| **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**

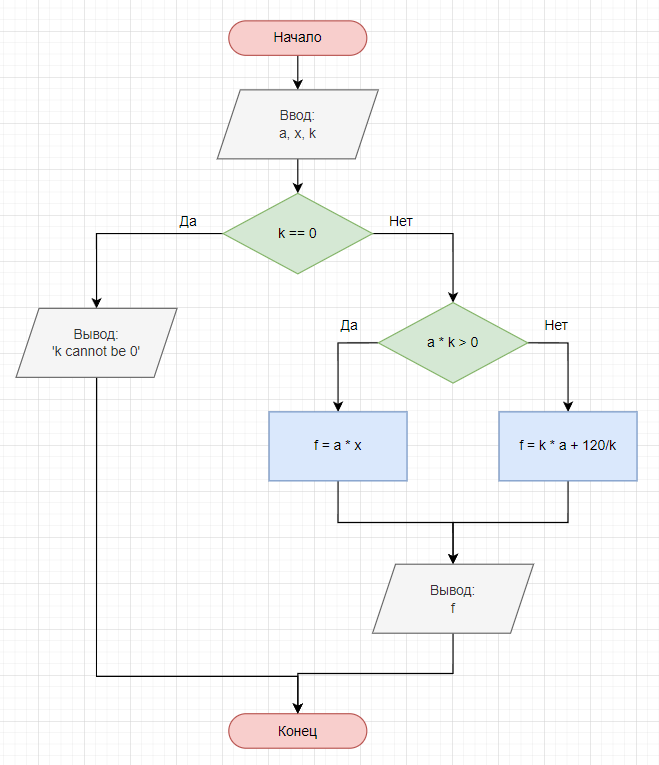
*Цель работы:*

Изучение приемов моделирования обработки массивов и матриц в языке ассемблера.

*Задание:* Дана матрица 5х5. Определить сумму положительных элементов каждой строки и поместить на место элементов главной диагонали. Организовать ввод матрицы и вывод результатов.

*Ход работы:*

Составим общую схему алгоритма решения данного задания (смотри рисунок 1).



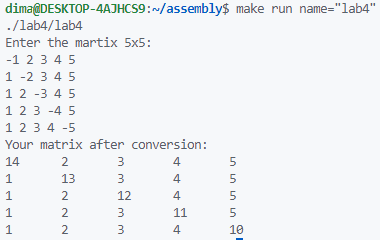
*Рисунок 1. - схема алгоритма программы*

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 martix *5x5:*", 10
44. lenInput    equ     $-InputMsg
45. OutputMsg   db      "Your matrix after conversion:", 10
46. lenOutput   equ     $-OutputMsg
47. tab         db      9
48. newLine     db      10
50. section .bss            ; сегмент неинициализированных переменных
51. InBuf   resb    10          ; буфер для вводимой строки
52. lenIn   equ     $-InBuf     ; длина буфера для вводимой строки
53. OutBuf  resb    10
54. lenOut  equ     $-OutBuf
55. matrix  resd    25          ; 5 \* 5 = 25 => резервируем 25 элементов для матрицы
56. sum     resd    1
58. section .text           ; сегмент кода
59. global \_start
61. \_start:
63. write\_string InputMsg, lenInput
65. ; ввод матрицы
66. mov rcx, 0          ; обнуляем счётчик внешнего цикла
67. cycle\_read\_matrix:
68. push rcx        ; помещаем rcx в стек
70. sub rsp, 16     ; выделяем память для буфера перевода строк в числа и счетчик (6 - буфер, 2 -счетчик)
71. sub rsp, 64     ; выделяем память для буфера ввода
73. read\_string rsp, 64
75. mov rcx, 0              ; rcx - индекс символа в строке, введенной пользователем
76. mov rax, [rsp + 80]     ; поместим в rax номер текущей строки (80 т.к. 64+16 = 80)
77. imul rax, 5             ; вычислим индекс элемента массива для записи при сквозной нумерации
78. mov [rsp + 70], ax      ; [rsp + 70] - индекс элемента массива для записи
79. mov rax, 0              ; rax - счётчик символов в буфере для перевода строк в числа
80. while:
81. cmp byte [rsp + rcx], 32        ; сравниваем символ в строке с пробелом
82. jne not\_space                   ; если не пробел, прыгаем на not\_space
83. jmp end\_of\_number               ; иначе прыгаем на end\_of\_number
84. not\_space:
85. cmp byte [rsp + rcx], 10    ; сравниваем символ в строке с enter
86. jne not\_enter               ; если не enter, прыгаем на not\_enter
87. jmp end\_of\_number           ; иначе прыгаем на end\_of\_number
88. not\_enter:
89. ; запоминаем символ в буфере
90. mov bl, [rsp + rcx]
91. mov [rsp + 64 + rax], bl    ; перенос символа из исходной строки в буфер для перевода
92. inc rax                     ; увеличиваем счётчик
93. jmp continue                ; прыгаем на continue
94. end\_of\_number:
95. mov bl, 10
96. mov [rsp + 64 + rax], bl    ; добавляем символ \n в буфер для перевода
97. lea rsi, [rsp + 64]         ; помещаем в rsi адрес буфера для перевода
98. mov rbx, 0                  ; чтобы StrToInt нормально работал
99. push rcx                    ; помещаем rcx в стек, потому что регистров не хватает, создатели ассемблера не подумали
100. mov rcx, [rsp + 78]         ; помещаем в rcx индекс элемента массива для записи
101. StrToInt [matrix + rcx\*4]   ; преобразуем буфер в число и записываем в матрицу
102. inc word [rsp + 78]         ; переходим к следующему элементу матрицы
103. pop rcx                     ; вытаскиваем rcx из стека, потому что регистров не хватало и т.п....
104. mov rax, 0                  ; обнуляем счётчик символов в буфере для перевода
105. cmp byte [rsp + rcx], 10    ; сравниваем символ в строке с enter
106. je break\_while              ; если enter, то выходим из цикла
107. continue:
108. inc rcx                     ; переходим к следующему символу в строке
109. jmp while                   ; переходим к следующей итерации цикла
110. break\_while:
112. add rsp, 80     ; вернем стек к изначальному состоянию
113. pop rcx         ; вытащим rcx из стека
114. inc rcx         ; увеличиваем счётчик строк на 1
115. cmp rcx, 5      ; если строка < 5 по счету, то переходим к следующей итерации
116. jl cycle\_read\_matrix
118. ; вычисления
119. mov rcx, 0              ; обнуляем счётчик внешнего цикла
120. cycle\_row:              ; внешний цикл для строк
121. push rcx            ; помещаем rcx в стек (номер текущей строки - 1)
122. mov rbx, 0
123. mov [sum], rbx      ; изначально сумма равна 0
124. mov rcx, 5          ; количество итерации внешнего цикла равно 5
125. cycle\_col:                      ; внутренний цикл для столбцов
126. push rcx                    ; помещаем rcx в стек (счётчик итераций)
127. push rbx                    ; помещаем rbx в стек (индекс элемента в строке)
128. mov rbx, [rsp + 16]         ; rbx = номер текущей строки - 1
129. imul rbx, 5                 ; rbx = (номер текущей строки - 1) \* 5, т.к. в строке 5 элементов
130. add rbx, [rsp]              ; rbx = rbx + индекс элемента в строке
131. mov eax, [matrix + rbx\*4]   ; eax = matrix[rbx], умножаем на 4, т.к. dword
132. pop rbx                     ; вытаскиваем rbx из стека (индекс элемента в строке)
133. inc rbx                     ; переходим к следующему элементу строки
134. pop rcx                     ; вытаскиваем rcx из стека (счётчик итераций)
135. cmp eax, 0
136. jle cont
137. add eax, [sum]              ; eax = eax + sum
138. mov [sum], eax              ; sum = eax
139. cont:
140. loop cycle\_col              ; переходим к следующей итерации внутреннего цикла
141. mov eax, [sum]              ; eax = конечная сумма строки
142. mov edx, eax                ; помещаем eax в edx
143. shr edx, 16                 ; сдвигаем edx на 2 байта, чтобы в dx лежали старшие разряды eax
144. pop rcx                     ; вытаскиваем rcx из стека (номер текущей строки - 1)
145. mov rbx, rcx                ; rbx = номер текущей строки - 1
146. imul rbx, 5                 ; rbx = (номер текущей строки - 1) \* 5
147. add rbx, rcx                ; rbx = индекс элемента матрицы, совпадающий с (5 - номер строки, в которой он находится)
148. mov [matrix + rbx\*4], eax   ; matrix[rbx] = сумма положительных чисел строки
149. mov rbx, 0                  ; обнуляем rbx
150. inc rcx                     ; увеличиваем счетчик итераций внешнего цикла
151. cmp rcx, 5                  ; сравниваем счётчик с 5
152. jl cycle\_row                ; если итераций меньше 5, прыгаем на cycle\_row
154. write\_string OutputMsg, lenOutput
156. ;вывод матрицы
157. mov rcx, 0              ; обнулим счётчик внешнего цикла (rcx = i \* 5, i = 0)
158. cycle\_print\_matrix:     ; внешний цикл для строк
159. push rcx            ; помещаем rcx в стек
160. mov rcx, 5          ; количество итераций внутреннего цикла равно 5
161. cycle\_print\_array:                      ; внутренний цикл для столбцов
162. push rcx                            ; помещаем rcx в стек, запоминаем номер итерации
163. neg rcx
164. add rcx, 5
165. add rcx, [rsp + 8]                  ; вычисляем индекс текущего элемента при сквозной нумерации (+8, т.к. до этого было 2 пуша)
166. IntToStr [matrix + rcx\*4], OutBuf   ; переводим элемент массива в строку и записываем в OutBuf
167. mov rbx, rax                        ; перекладываем длину строки в rbx, т.к. write\_string работает с rax
168. dec rbx                             ; удаляем символ перевода строки из строки
169. write\_string OutBuf, rbx            ; выводим один элемент матрицы
170. mov rbx, 0                          ; обнуляем rbx
171. write\_string tab, 1                 ; дописываем \t к строке
172. pop rcx                             ; вытаскиваем rcx из стека
173. loop cycle\_print\_array
174. write\_string newLine, 1                 ; дописываем \n к строке
176. pop rcx                     ; вытаскиваем rcx из стека, rcx = количество пройденных строк \* 5
177. add rcx, 5                  ; увеличиваем счетчик (rcx = i \* 5, i++)
178. cmp rcx, 25                 ; сравниваем счетчик для нахождения конца матрицы (25 = 5 \* 5, 5 - количество строк в матрице)
179. jl cycle\_print\_matrix       ; если счетчик меньше количества строк, переходим к следующей итерации
181. mov     rax, 60         ; системная функция 60 (exit)
182. xor     rdi, rdi        ; return code 0
183. syscall                 ; вызов системной функции

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



*Рисунок 2 – пример выполнения программы*

*Таблица 1 – результаты тестирования*

|  |  |  |
| --- | --- | --- |
| **Исходные данные** | **Ожидаемый результат** | **Полученный результат** |
| 1 2 3 4 5  1 2 3 4 5  1 2 3 4 5  1 2 3 4 5  1 2 3 4 5 | 15 2 3 4 5  1 15 3 4 5  1 2 15 4 5  1 2 3 15 5  1 2 3 4 15 | 15 2 3 4 5  1 15 3 4 5  1 2 15 4 5  1 2 3 15 5  1 2 3 4 15 |
| -1 2 3 4 5  1 -2 3 4 5  1 2 -3 4 5  1 2 3 -4 5  1 2 3 4 -5 | 14 2 3 4 5  1 13 3 4 5  1 2 12 4 5  1 2 3 11 5  1 2 3 4 10 | 14 2 3 4 5  1 13 3 4 5  1 2 12 4 5  1 2 3 11 5  1 2 3 4 10 |

*Контрольные вопросы*

1. Какие машинные команды используют при программировании ветвлений и циклов?

Ветвление осуществляется про помощи команды проверки *cmp* и команд перехода по условию: je, jnl, jge и других. Принцип их работы заключается в считывании регистра флагов и переходе по указанной метке.

1. Выделите в своей программе фрагмент, реализующий ветвление. Каково назначение каждой машинной команды фрагмента?

Ветвление реализовано на строках 95-119 листинга 1. Здесь при помощи команды *cmp* сравнивается содержимое регистра DX (регистра, в котором находятся старшие байты, полученные при вычислении операции a \* x)

1. Чем вызвана необходимость использования команд безусловной передачи управления?

Безусловная передача управления используется, чтобы перейти на команды, следующие за ветвлением, а не на те, которые следует пропустить.

1. Поясните последовательность команд, выполняющих операции ввода-вывода в вашей программе. Чем вызвана сложность преобразований данных при выполнении операций ввода-вывода?

При чтении пользовательского ввода строки сначала попадают в буферную переменную в памяти, из которой они сразу же извлекаются и преобразовываются в числа. Сложность вызвана именно тем, что для ввода и вывода используются символьные обозначения цифр, а для вычислений применяются внутренние представления чисел.

***Вывод:***в рамках лабораторной работы были изучены программирование ветвлений на языке ассемблера и их применение при решении вычислительных задач.