|  |  |
| --- | --- |
| 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 №** | **2** |

|  |  |
| --- | --- |
| **Название:** | Программирование целочисленных вычислений |
| **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**

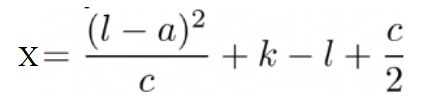
*Цель работы:*

Изучение форматов машинных команд, команд

целочисленной арифметики ассемблера и программирование целочисленных

вычислений.

*Задание:* вычислить целочисленное выражение:



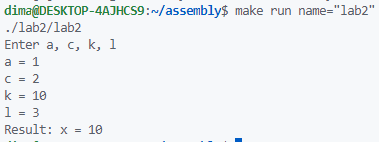
*Ход работы:*

Выполним поставленную задачу с помощью следующей программы 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     ax, %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
43. InputMsg db "Enter a, c, k, l", 10
44. lenInput equ $-InputMsg
45. aInput db "a = "
46. aLen equ $-aInput
47. cInput db "c = "
48. cLen equ $-cInput
49. kInput db "k = "
50. kLen equ $-kInput
51. lInput db "l = "
52. lLen equ $-lInput
53. OutputMsg db "Result: x = "
54. lenOutput equ $-OutputMsg
55. ErrorMsg db "c cannot be 0", 10
56. lenError equ $-ErrorMsg
58. section .bss
59. InBuf resb 10
60. lenIn equ $-InBuf
61. OutBuf resb 10
62. a resw 1
63. c resw 1
64. k resw 1
65. l resw 1
66. x resw 1
68. section .text
69. global \_start
71. \_start:
72. ; ввод
74. write\_string InputMsg, lenInput
76. write\_string aInput, aLen
77. read\_string InBuf, lenIn
78. StrToInt [a]
80. write\_string cInput, cLen
81. read\_string InBuf, lenIn
82. StrToInt [c]
84. write\_string kInput, kLen
85. read\_string InBuf, lenIn
86. StrToInt [k]
88. write\_string lInput, lLen
89. read\_string InBuf, lenIn
90. StrToInt [l]
92. mov ax, [c]
93. cmp ax, 0
94. je catch
96. ; вычисления
98. mov AX, [l]  ; ax = l;
99. sub AX, [a]  ; ax -= a;
100. imul AX      ; ax \*= ax; // dx:ax = ax\*ax;
101. mov BX, [c]  ; bx = c;
102. idiv BX      ; ax = dx:ax / bx;
103. add AX, [k]  ; ax += k;
104. sub AX, [l]  ; ax -= l;
105. mov BX, [c]  ; bx = c;
106. sar BX, 1    ; bx >> 1;
107. add AX, BX   ; ax += bx;
108. mov [x], AX  ; x = ax
110. ; вывод
112. xor rbx, rbx
113. write\_string OutputMsg, lenOutput
114. IntToStr [x], OutBuf
115. mov RBX, RAX
116. write\_string OutBuf, RBX
117. jmp exit
119. catch:
120. write\_string ErrorMsg, lenError
122. exit:
123. mov     rax, 60         ; системная функция 60 (exit)
124. xor     rdi, rdi        ; return code 0
125. syscall                 ; вызов системной функции

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



*Рисунок 1 – пример выполнения программы*

*Таблица 1 – результаты тестирования*

|  |  |  |
| --- | --- | --- |
| **Исходные данные** | **Ожидаемый результат** | **Полученный результат** |
| a = 1, c = 2, k = 10, l = 3 | x = 10 | x = 10 |
| a = -1, c = 3, k = -5, l = 1 | x = -4 | x = -4 |
| a = -10, c = 10, k = 10, l = 10 | x = 45 | x = 45 |
| a = 1, c = 0, k = 1, l = 1 | c cannot be 0 | c cannot be 0 |

Рассмотрим в отладчике несколько команд *mov*, представленных в программе. В таблице 2 приведены некоторые операции из кода программы, их машинные коды и интерпретации.

*Таблица 2 – расшифровки команд процессора*

|  |  |
| --- | --- |
| Команда | КОП и его расшифровка |
| 98.  mov ax, [l] | 66 8b 04 25 56 20 40 00  66 – префикс размера операнда  8b16 = 100010112  100010 – код операции mov  1 – бит d. Направление обработки – в регистр.  1 – бит w. Используются двойные слова.  0416 = 000001002  00 – mod. Смещение отсутствует.  000 – reg. Регистр EAX.  100 – r/m. Ожидается байт Sib.  2516 = 001001012  00 – SS. 1 байт.  100 – index. ESP  101 – base. Disp32+ss\*index  56 20 40 0016 – смещение на 40205616 |
| 101.  mov bx, [c] | 66 8b 1c 25 52 20 40 00  6616 – префикс размера операнда  8b16 = 100010112  100010 – код операции mov  1 – бит d. Направление обработки – в регистр.  1 – бит w. Используются двойные слова.  1c16 = 000111002  00 – mod. Смещение отсутствует.  011 – reg. Регистр EBX.  100 – r/m. Ожидается байт Sib.  2516 = 001001012  00 – SS. 1 байт.  100 – index. ESP  101 – base. Disp32+ss\*index  52 20 40 0016 – смещение на 40205216 |
| 108.  mov [x], ax | 66 89 04 25 58 20 40 00  66 – префикс размера операнда  8916 = 100010012  100010 – код операции mov  0 – бит d. Направление обработки – из регистра.  1 – бит w. Используются двойные слова.  0416 = 000001002  00 – mod. Смещение отсутствует.  000 – reg. Регистр EAX.  100 – r/m. Ожидается байт Sib.  2516 = 001001012  00 – SS. 1 байт.  100 – index. ESP  001 – base. Disp32+ss\*index  58 20 40 0016 – смещение на 40205816 |

*Контрольные вопросы*

1. Что такое машинная команда? Какие форматы имеют машинные ко-

манды процессора IA32? Чем различаются эти форматы?

Машинная команда – это элементарная инструкция машине для выполнения. Размер машинной команды процессора IA-32 колеблется от 1 до 15 байт. Машинные команды ассемблера имеют следующий формат:

**[Метка :] Код операции [Список операндов] [; Комментарии].**

1. Назовите мнемоники основных команд целочисленной арифметики.

Какие форматы для них можно использовать?

Основные команды целочисленной арифметики: add, sub, div, idiv, mul imul.

1. add приемник, источник
2. sub приемник, источник
3. mul/imul [приемник], источник
4. div/idiv источник

В mul, div, imul, idiv - источник регистры AL/AX/EAX в зависимости от размера источника.

*Таблица 2 - расположение в регистрах при умножении (mul/imul)*

|  |  |  |
| --- | --- | --- |
| **Размер операнда** | **Множитель** | **Результат** |
| 1 байт | AL | AX |
| 2 байта | AX | DX:AX |
| 4 байта | EAX | EDX:EAX |

*Таблица 3 - расположение в регистрах при делении (div/idiv)*

|  |  |  |  |
| --- | --- | --- | --- |
| **Размер операнда (делителя)** | **Делимое** | **Частное** | **Остаток** |
| 1 байт | AX | AL | AH |
| 2 байта | DX:AX | AX | DX |
| 4 байта | EDX:EAX | EAX | EDX |

3. Сформулируйте основные правила построения линейной программы

вычисления заданного выражения.

– следует определить порядок действий в выражении

– при написании программы лучше придерживаться логики порядка действий и выполнения вычислений

– следует предположить, какие значения могут вводиться пользователем и получатся в результате выполнения операций

4. Почему ввод-вывод на языке ассемблера не программируют с использованием соответствующих машинных команд? Какая библиотека используется для организации ввода вывода в данной лабораторной?

Потому что использование библиотеки для обработки операций ввода и вывода значительно сокращает объем программы. В данной лабораторной работе используется библиотека, скачанная с предложенного источника.

5. Расскажите, какие операции используют при организации ввода-

вывода.

Для организации ввода и вывода соответствующие режимам коды сообщают регистрам RAX и RDI, в RSI кладут адрес вводимой (выводимой) строки, в RDX – ее длину.

***Вывод:***в рамках лабораторной работы были изучены программирование арифметики на языке ассемблера и формат машинных команд.