Федеральное государственное бюджетное образовательное

учреждение высшего профессионального образования

«**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**

**МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ им. Н. П. ОГАРЁВА»**

Институт электроники и светотехники

Кафедра автоматизированных систем обработки информации и управления

**ЛАБОРАТОРНАЯ РАБОТА №6**

по курсу

«Программирование на языке Ассемблера»

**Структурное программирование**

ЛР – 02069964 – Программирование – 06 – 16

ВЫПОЛНИЛ: ПРОВЕРИЛ:

студент 341 группы зав. кафедрой АСОИУ,

к. т. н., профессор

\_\_\_\_\_\_\_\_\_\_08.12.2016 \_\_\_\_\_\_\_\_\_14.12.2016

Гагарин В. Ю. Федосин С. А.

Саранск 2016

**Лабораторная работа № 6**

**Структурное программирование**

**Задание 1.**

|  |
| --- |
| Разработайте процедуры, выполняющие операции со строками: вычисление длины строки, копирование одной строки в другую, добавление одной строки в конец другой. Напишите демонстрационную программу для разработанных процедур. |

**Код программы:**

|  |
| --- |
| TITLE exercise1  .686  .MODEL FLAT, C  .STACK 4096    printf PROTO format : PTR BYTE, args : VARARG  gets PROTO s : PTR BYTE    PUBLIC main    .CONST  sStr1 BYTE "Enter the first string: ", 0  sStr2 BYTE "Enter the second string: ", 0  sLength BYTE "String length: %d", 0Ah, 0  sCopy BYTE "After copy string: %s", 0Ah, 0  sInsert BYTE "After insert string: %s", 0Ah, 0  sNewLine BYTE " ", 0Ah, 0    .DATA?    rez DWORD ?  MaxStr = 256  str1 BYTE MaxStr DUP(?)  str2 BYTE MaxStr DUP(?)    .CODE    \_size PROC s1 : PTR BYTE, n : DWORD  cld  mov edi, [s1]  mov ecx, [n]  xor eax, eax  repnz scasb  neg ecx  sub n, 1  add ecx, [n]  mov rez, ecx  ret  \_size ENDP    \_copy PROC USES esi edi, s1: PTR BYTE, s2 : PTR BYTE  cld  mov ecx, LENGTHOF str1  mov esi, [s1]  mov edi, [s2]  rep movsb  ret  \_copy ENDP    \_insert PROC USES esi edi, s1: PTR BYTE, s2 : PTR BYTE  cld  mov ecx, LENGTHOF str1  mov esi, [s1]  INVOKE \_size, ADDR str2, MaxStr  mov edi, [s2]  add edi, rez  rep movsb  ret  \_insert ENDP    main PROC    INVOKE printf, ADDR sStr1  INVOKE gets, ADDR str1    INVOKE \_size, ADDR str1, MaxStr    INVOKE printf, ADDR sLength, ecx    INVOKE printf, ADDR sNewLine  INVOKE printf, ADDR sStr2  INVOKE gets, ADDR str2    INVOKE \_copy, ADDR str1, ADDR str2    INVOKE printf, ADDR sCopy, ADDR str2    INVOKE printf, ADDR sNewLine  INVOKE printf, ADDR sStr1  INVOKE gets, ADDR str1    INVOKE printf, ADDR sStr2  INVOKE gets, ADDR str2    INVOKE \_insert, ADDR str2, ADDR str1    INVOKE printf, ADDR sInsert, ADDR str1    mov eax, 0  ret  main ENDP  END |

Результат выполнения программы представлен на рисунке 1.

|  |
| --- |
|  |

Рисунок 1

**Задание 2.**

|  |
| --- |
| Разработайте процедуры, выполняющие сложение и умножение матриц четвертого порядка. Напишите демонстрационную программу для разработанных процедур. |

**Код программы:**

|  |
| --- |
| TITLE exercise2  .686  .MODEL FLAT, C  .STACK 4096    printf PROTO format : PTR BYTE, args : VARARG  scanf PROTO format : PTR BYTE, args : VARARG    PUBLIC main    .CONST    sPrint BYTE "%d ", 0  sMatrA BYTE "Matrix A:", 0Ah, 0  sMatrB BYTE "Matrix B:", 0Ah, 0  sADD BYTE "A + B:", 0Ah, 0  sMult BYTE "A \* B:", 0Ah, 0  sPrintARR BYTE "%d %d %d %d", 0Ah, 0  sNewLine BYTE " ", 0Ah, 0    .DATA    a SDWORD 1, 5, 9, 13  SDWORD 2, 6, 10, 14  SDWORD 3, 7, 11, 15  SDWORD 4, 8, 12, 16    b SDWORD 16, 12, 8, 4  SDWORD 15, 11, 7, 3  SDWORD 14, 10, 6, 2  SDWORD 13, 9, 5, 1    column SDWORD 4    .DATA?    res SDWORD 4 DUP(?)  SDWORD 4 DUP(? )  SDWORD 4 DUP(? )  SDWORD 4 DUP(? )    \_column SDWORD ?  i DWORD ?  j DWORD ?  buf DWORD ?    .CODE    addMatr PROC USES eax ebx ecx esi edi, A1 : PTR SDWORD, B1 : PTR SDWORD, RES1 : PTR SDWORD    mov edi, RES1  mov esi, A1  mov ebx, B1  mov ecx, 16  Tag1:  mov eax, SDWORD PTR[esi]  add eax, SDWORD PTR[ebx]  mov SDWORD PTR[edi], eax  add esi, 4  add ebx, 4  add edi, 4  loop Tag1    ret  addMatr ENDP    multMatr PROC USES eax ebx ecx edx esi edi, A1 : PTR SDWORD, B1 : PTR SDWORD, RES1 : PTR SDWORD    xor ecx, ecx    Multiply :  mov eax, ecx    and eax, 03h  mov edx, 4  mul edx  mov edi, [B1]  add edi, eax    mov eax, ecx  and eax, 0Ch  mul edx  mov esi, [A1]  add esi, eax    xor ebx, ebx  mov eax, SDWORD PTR[esi]  imul DWORD PTR[edi]  add ebx, eax  add esi, 4  mov eax, SDWORD PTR[esi]  add edi, 16  imul DWORD PTR[edi]  add ebx, eax  add esi, 4  mov eax, SDWORD PTR[esi]  add edi, 16  imul DWORD PTR[edi]  add ebx, eax  add esi, 4  mov eax, SDWORD PTR[esi]  add edi, 16  imul DWORD PTR[edi]  add ebx, eax  mov eax, ecx  mov edx, 4  mul edx  mov edx, [RES1]  add edx, eax  mov SDWORD PTR[edx], ebx  inc ecx  cmp ecx, 16  jb Multiply    ret  multMatr ENDP        main PROC  INVOKE printf, ADDR sMatrA  INVOKE printf, ADDR sPrintARR, a, a + 4, a + 8, a + 12  INVOKE printf, ADDR sPrintARR, a + 16, a + 20, a + 24, a + 28  INVOKE printf, ADDR sPrintARR, a + 32, a + 36, a + 40, a + 44  INVOKE printf, ADDR sPrintARR, a + 48, a + 52, a + 56, a + 60    INVOKE printf, ADDR sNewLine  INVOKE printf, ADDR sMatrB  INVOKE printf, ADDR sPrintARR, b, b + 4, b + 8, b + 12  INVOKE printf, ADDR sPrintARR, b + 16, b + 20, b + 24, b + 28  INVOKE printf, ADDR sPrintARR, b + 32, b + 36, b + 40, b + 44  INVOKE printf, ADDR sPrintARR, b + 48, b + 52, b + 56, b + 60    INVOKE addMatr, ADDR a, ADDR b, ADDR res    INVOKE printf, ADDR sNewLine  INVOKE printf, ADDR sADD  INVOKE printf, ADDR sPrintARR, res, res + 4, res + 8, res + 12  INVOKE printf, ADDR sPrintARR, res + 16, res + 20, res + 24, res + 28  INVOKE printf, ADDR sPrintARR, res + 32, res + 36, res + 40, res + 44  INVOKE printf, ADDR sPrintARR, res + 48, res + 52, res + 56, res + 60    INVOKE printf, ADDR sNewLine  INVOKE multMatr, ADDR a, ADDR b, ADDR res    INVOKE printf, ADDR sMult  INVOKE printf, ADDR sPrintARR, res, res + 4, res + 8, res + 12  INVOKE printf, ADDR sPrintARR, res + 16, res + 20, res + 24, res + 28  INVOKE printf, ADDR sPrintARR, res + 32, res + 36, res + 40, res + 44  INVOKE printf, ADDR sPrintARR, res + 48, res + 52, res + 56, res + 60    xor eax, eax  ret  main ENDP  END |

Результат выполнения программы представлен на рисунке 2.

|  |
| --- |
|  |

Рисунок 2

**Задание 3.**

|  |
| --- |
| Разработайте процедуру, выполняющую вычисление периметра произвольного прямоугольника. Для хранения вершин прямоугольника используйте массив структур Point2. |

**Код программы:**

|  |
| --- |
| TITLE exercise3  .686  .MODEL FLAT, C  .STACK 4096    printf PROTO format : PTR BYTE, args : VARARG  scanf PROTO format : PTR BYTE, args : VARARG    PUBLIC main    Point2 STRUCT  X SDWORD ?  Y SDWORD ?  Point2 ENDS    Rect2 STRUCT  leftUpPoint Point2 <>  rightDownPoint Point2 <>  Rect2 ENDS    .CONST  sInput BYTE "%d %d", 0  sEnterLeftUpPoint BYTE "Enter left down point(X, Y): ", 0  sEnterRightDownPoint BYTE "Enter right up point(X, Y): ", 0  sPerimetr BYTE "Perimetr = %d", 0Ah, 0    .DATA  rectangle Rect2 <>    .CODE    perimetr PROC USES edx esi, rect: PTR Rect2  mov esi, [rect]    mov eax, (Rect2 PTR[esi]).rightDownPoint.X  sub eax, (Rect2 PTR[esi]).leftUpPoint.X  mov edx, 2  mul edx    mov ebx, eax    mov eax, (Rect2 PTR[esi]).rightDownPoint.Y  sub eax, (Rect2 PTR[esi]).leftUpPoint.Y  mov edx, 2  mul edx    add eax, ebx    ret  perimetr ENDP    main PROC    INVOKE printf, ADDR sEnterLeftUpPoint  INVOKE scanf, ADDR sInput, ADDR rectangle.leftUpPoint.X, ADDR rectangle.leftUpPoint.Y    INVOKE printf, ADDR sEnterRightDownPoint  INVOKE scanf, ADDR sInput, ADDR rectangle.rightDownPoint.X, ADDR rectangle.rightDownPoint.Y    INVOKE perimetr, ADDR rectangle    INVOKE printf, ADDR sPerimetr, eax    mov eax, 0  ret  main ENDP  END |

Результат выполнения программы представлен на рисунке 3.

|  |
| --- |
|  |

Рисунок 3

**Задание 4.**

|  |
| --- |
| Напишите на языке C функцию, выполняющую обмен двух целочисленных переменных, переданных ей в виде аргументов. Такая функция должна иметь прототип void swap(int \*, int \*). Используя окно дизассемблера, исследуйте ассемблерный код этой функции. Где хранятся аргументы функции и как осуществляется к ним доступ? Где хранятся локальные переменные функции и как осуществляется к ним доступ? Как выглядит пролог и эпилог этой функции? Сравните ассемблерный код, полученный для пролога и эпилога функции, написанной на языке C с пологом и эпилогом функции, написанной на языке ассемблера с использованием директив PROC со списком параметров, USES и LOCAL. |

**Код программы:**

**На языке высокого уровня C:**

|  |
| --- |
| #include <stdio.h>    void swap(int\* a, int\* b)  {  int tmp = \*a;  \*a = \*b;  \*b = tmp;  }    int main()  {  int a = 1;  int b = 0;    printf\_s("Before:\na = %d\nb = %d\n", a, b);  swap(&a, &b);  printf\_s("After:\na = %d\nb = %d\n", a, b);    return 0;  } |

**Дизассемблирование программы с языка C:**

|  |
| --- |
| void swap(int\* a, int\* b)  {  001C14A0 push ebp  001C14A1 mov ebp,esp  001C14A3 sub esp,0CCh  001C14A9 push ebx  001C14AA push esi  001C14AB push edi  001C14AC lea edi,[ebp-0CCh]  001C14B2 mov ecx,33h  001C14B7 mov eax,0CCCCCCCCh  001C14BC rep stos dword ptr es:[edi]  int tmp = \*a;  001C14BE mov eax,dword ptr [a]  001C14C1 mov ecx,dword ptr [eax]  001C14C3 mov dword ptr [tmp],ecx  \*a = \*b;  001C14C6 mov eax,dword ptr [a]  001C14C9 mov ecx,dword ptr [b]  001C14CC mov edx,dword ptr [ecx]  001C14CE mov dword ptr [eax],edx  \*b = tmp;  001C14D0 mov eax,dword ptr [b]  001C14D3 mov ecx,dword ptr [tmp]  001C14D6 mov dword ptr [eax],ecx  }  001C14D8 pop edi  001C14D9 pop esi  001C14DA pop ebx  001C14DB mov esp,ebp  001C14DD pop ebp  001C14DE ret |

**На языке Ассемблера**

|  |
| --- |
| TITLE exercise4  .686  .MODEL FLAT, C  .STACK 4096    printf PROTO format : PTR BYTE, args : VARARG  scanf PROTO format : PTR BYTE, args : VARARG    PUBLIC main    .CONST  sPref BYTE "%d", 0  sBefore BYTE "Before: a = %d, b = %d", 0Ah, 0  sAfter  BYTE "After: a = %d, b = %d", 0Ah, 0    .DATA  a SDWORD 0  b SDWORD 1    .CODE  swap PROC, pa : DWORD, pb : DWORD  LOCAL tmp : DWORD  xor eax, eax  xor ebx, ebx  mov eax, pa  mov tmp, eax  mov eax, pb  mov pa, eax  mov eax, tmp  mov pb, eax  mov eax, pa  mov ebx, pb  ret  swap ENDP    main PROC    INVOKE printf, ADDR sBefore, a, b  INVOKE swap, a, b  mov a, eax  mov b, ebx  INVOKE printf, ADDR sAfter, a, b  ret  main ENDP  END |

**Дизассемблирование с языка Ассемблера:**

|  |
| --- |
| swap PROC, pa : DWORD, pb : DWORD  00DEB140 push ebp  00DEB141 mov ebp,esp  00DEB143 add esp,0FFFFFFFCh  LOCAL tmp : DWORD  xor eax, eax  00DEB146 xor eax,eax  xor ebx, ebx  00DEB148 xor ebx,ebx  mov eax, pa  00DEB14A mov eax,dword ptr [pa]  mov tmp, eax  00DEB14D mov dword ptr [tmp],eax  mov eax, pb  00DEB150 mov eax,dword ptr [pb]  mov pa, eax  00DEB153 mov dword ptr [pa],eax  mov eax, tmp  00DEB156 mov eax,dword ptr [tmp]  mov pb, eax  00DEB159 mov dword ptr [pb],eax  mov eax, pa  00DEB15C mov eax,dword ptr [pa]  mov ebx, pb  00DEB15F mov ebx,dword ptr [pb]  ret  00DEB162 leave  00DEB163 ret |

Результат выполнения программы представлен на рисунке 4.

|  |
| --- |
|  |

Рисунок 4

После осуществления дизассемблирования функции swap(), написанной на языке C и Assembler, проанализировав результат, можно сказать: Пролог и эпилог этой функции, в языке C, получился больше по количеству кода, чем в Assembler.

В языке C, в прологе мы заталкиваем в стек различные регистры, а в эпилоге их выталкиваем. В языке Ассемблера такого нет.