Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра системного програмування
і спеціалізованих комп’ютерних системи

Лабораторна робота №1.1

з дисципліни
«Архітектура для програмістів»

Виконав:
студент групи КВ-11
Терентьєв Іван Дмитрович

Перевірив:
Молчанов О. А.

Київ – 2024

Загальне завдання

1. Реалізувати програму сортування масиву згідно із варіантом мовою С (інформація про варіанти наведена в п. 4). Результатом виконання цього пункту є лістинг програми мовою С.

2. Виконати трансляцію програми, написаної мовою С, в асемблерний код за допомогою gcc й встановити семантичну відповідність між командами мови С та командами одержаного асемблерного коду, додавши відповідні коментарі з поясненням. Результатом виконання даного пункту буде лістинг асемблерного коду програми із коментарями в коді, в яких наведено відповідний код програми, записаною мовою С (приклад. див. в п. 3.2).

3. Розібратись і вміти пояснити, що виконують ті чи інші команди мови асемблера, що будуть присутні в коді мовою асемблера, отриманого в другому пункті загального завдання; вміти пояснити зв’язок між кодом мовою С та кодом мовою асемблера.

Завдання за варіантом 23

Задано двовимірний масив (матрицю) цілих чисел A[m][n]. Відсортувати окремо кожен стовпчик масиву алгоритмом №1 методу вставки (з лінійним пошуком зліва) за незменшенням.

1. Лістинг програми мовою С

```
void Algo1(int **A, int m, int n)
{
   int Elem,j;
   // main loop
   for (int r = 0; r < n; r++)
   {// secondary loop
      for (int i = 1; i < m; i++)
      {
         Elem = A[i][r];
         j = 0;
         // third loop
         while (Elem > A[j][r])
            j = j + 1;
         //fourth loop
         for (int k = i - 1; k >= j; k--)
            A[k + 1][r] = A[k][r];
         A[j][r] = Elem;
      }
   }
}
```

2. Лістинг програми байт-кодом С з поясненнями

```
	.text
	.globl	Algo1
	.type Algo1, @function
Algo1:
.LFB6:
	.cfi_startproc
	push  rbp   #
	.cfi_def_cfa_offset 16
	.cfi_offset 6, -16
	mov   rbp, rsp    #,
	.cfi_def_cfa_register 6
# associate actuals with formals
	mov   QWORD PTR -40[rbp], rdi # associate actual value with formal A
	mov   DWORD PTR -44[rbp], esi # associate actual value with formal m
	mov   DWORD PTR -48[rbp], edx # associate actual value with formal n
# function body start
# main loop start: for (int r = 0; r < n; r++)
	mov   DWORD PTR -16[rbp], 0   # initilize “counter” r: int r = 0
	jmp   .L2                     # jump to main loop condition check
.L9:
# main loop body start
# secondary loop start: for (int i = 1; i < m; i++)
	mov   DWORD PTR -12[rbp], 1   # initilize “counter” i: int i = 1
	jmp   .L3                     # jump to secondary loop condition
check
# secondary loop body start
.L8:
	mov   eax, DWORD PTR -12[rbp] # save value of i to EAX
	cdqe                          # extend value in EAX to RAX preserving
sign
	lea   rdx, 0[0+rax*8]         # calculate and save shift in A to RDX
	mov   rax, QWORD PTR -40[rbp] # save A start address to RAX
	add   rax, rdx                # create address of certain A[i] in RAX
	mov   rax, QWORD PTR [rax]    # save value of A[i] to RAX
	mov   edx, DWORD PTR -16[rbp] # save value of r to EDX
	movsx rdx, edx                # copy EDX to RDX
	sal   rdx, 2                  # left shift RDX by 2
	add   rax, rdx                # create address of certain A[i][r] in
RAX
	mov   eax, DWORD PTR [rax]    # save adress of A[i][r] to EAX
	mov   DWORD PTR -4[rbp], eax  # initialize value of Elem: Elem = A[i][r]
	mov   DWORD PTR -20[rbp], 0   # initilize “counter” j: int j = 0
	jmp   .L4   # jump to third loop condition check
# third loop start: while(Elem > A[j][r])
.L5:
# third loop body start
	add   DWORD PTR -20[rbp], 1   # increment “counter” j: j = j + 1
# third loop body end
.L4:
# third loop exit condition start
	mov   eax, DWORD PTR -20[rbp] # save value of j to EAX
	cdqe                          # extend value in EAX to RAX
preserving sign
	lea   rdx, 0[0+rax*8]         # calculate and save shift in A to RDX
	mov   rax, QWORD PTR -40[rbp] # save A start address to RAX
	add   rax, rdx                # create address of certain A[j] in RAX
```

```
      mov   rax, QWORD PTR [rax]    # save value of A[i] to RAX
      mov   edx, DWORD PTR -16[rbp] # save value of r to EDX
      movsx rdx, edx                # copy EDX to RDX
      sal   rdx, 2                        # left shift RDX by 2
      add   rax, rdx                    # create address of certain A[j][r] in
RAX
      mov   eax, DWORD PTR [rax]    # save adress of A[j][r] to EAX
      cmp   DWORD PTR -4[rbp], eax  # compare Elem with A[j][r]
      jg    .L5   # jump to third loop body start if Elem > A[j][r]
# third loop exit condition end
# third loop end
# fourth loop start: for (int k = i - 1; k >= j; k--)
      mov   eax, DWORD PTR -12[rbp] # save value of i to register EAX
      sub   eax, 1                        # subtract value of i by 1: i - 1
      mov   DWORD PTR -8[rbp], eax  # save EAX to k: k = i - 1
      jmp   .L6                           # jump to fourth loop condition
check
.L7:
# fourth loop body start
      mov   eax, DWORD PTR -8[rbp]  # save value of k to EAX
      cdqe                              # extend value in EAX to RAX preserving
sign
      lea   rdx, 0[0+rax*8]             # calculate and save shift in A to RDX
      mov   rax, QWORD PTR -40[rbp] # save A start address to RAX
      add   rax, rdx                    # create address of certain A[k] in RAX
      mov   rax, QWORD PTR [rax]    # save value of A[k] to RAX
      mov   edx, DWORD PTR -16[rbp] # save value of r to EDX
      movsx rdx, edx                # copy EDX to RDX
      sal   rdx, 2                        # left shift RDX by 2
      add   rax, rdx                    # create address of certain A[k][r] in
RAX
      mov   edx, DWORD PTR -8[rbp]  # save value of k to EDX
      movsx rdx, edx                # copy EDX to RDX
      add   rdx, 1                        # add 1 to RDX: k + 1
      lea   rcx, 0[0+rdx*8]             # calculate and save shift in A to RCX
      mov   rdx, QWORD PTR -40[rbp] # save A start address to RAX
      add   rdx, rcx                    # save adress of A[k + 1] to RDX
      mov   rdx, QWORD PTR [rdx]    # save value of A[k + 1] to RDX
      mov   ecx, DWORD PTR -16[rbp] # save value of r to EDX
      movsx rcx, ecx                # copy RCX to ECX
      sal   rcx, 2                        # left shift RCX by 2
      add   rdx, rcx                    # create address of certain A[k+1][r] in
RDX
      mov   eax, DWORD PTR [rax]    # save value of A[k][r] to EAX
      mov   DWORD PTR [rdx], eax    # save EAX to A[k+1][r]: A[k+1][r] = A[k][r]
# fourth loop body end
      sub   DWORD PTR -8[rbp], 1    # decrement “counter” k: k--
.L6:
# fourth loop exit condition start
      mov   eax, DWORD PTR -8[rbp]  # save value of k to register EAX
      cmp   eax, DWORD PTR -20[rbp] # compare k with j
      jge   .L7   # jump to fourth loop body start if k >= j
# fourth loop exit condition end
# fourth loop end
      mov   eax, DWORD PTR -20[rbp] # save value of j to register EAX
      cdqe                              # extend value in EAX to RAX preserving
sign
      lea   rdx, 0[0+rax*8]             # calculate and save shift in A to RDX
      mov   rax, QWORD PTR -40[rbp] # save A start address to RAX
```

```
        add   rax, rdx                      # create address of certain A[j] in RAX
        mov   rax, QWORD PTR [rax]    # save value of A[j] to RAX
        mov   edx, DWORD PTR -16[rbp] # save value of r to register EDX
        movsx rdx, edx                # copy EDX to RDX
        sal   rdx, 2                            # left shift RDX by 2
        add   rdx, rax                      # save adress of A[j][r] to RDX
        mov   eax, DWORD PTR -4[rbp]  # save value of Elem to EAX
        mov   DWORD PTR [rdx], eax    # initialize value of A[j][r]: A[j][r] = Elem
# secondary loop body end
        add   DWORD PTR -12[rbp], 1   # increment “counter” i: i++
.L3:
# secondary loop exit condition start
        mov   eax, DWORD PTR -12[rbp] # save value of i to register EAX
        cmp   eax, DWORD PTR -44[rbp] # compare i with m
        jl    .L8   # jump to secondary loop body start if i < m
# secondary loop exit condition end
# secondary loop end
# main loop body end
        add   DWORD PTR -16[rbp], 1   # increment “counter” r: r++
.L2:
# main loop exit condition start
        mov   eax, DWORD PTR -16[rbp] # save value of r to register EAX
        cmp   eax, DWORD PTR -48[rbp] # compare r with n
        jl    .L9   # jump to main loop body start if r < n
# main loop exit condition end
# main loop end
# function end
        nop
        nop
        pop   rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```