Национальный исследовательский университет «Высшая школа экономики»
Факультет компьютерных наук
Программная инженерия

Архитектура вычислительных систем

Отчёт по индивидуальному домашнему заданию№1
Вариант 37

Работу выполнила:
М. В. Царахова
Группа: БПИ-213
Преподаватель:
С. А. Виденин

Москва
2022

Содержание

Постановка задачи

Разработать программу, которая получает одномерный массив AN, после чего формирует из элементов массива A новый массив B по правилам, указанным в варианте, и выводит его. Память под массивы может выделяться статически, на стеке, автоматичеси по выбору разработчика. При решении задачи необходимо использовать подпрограммы для реализации ввода, вывода и формирования нового массива.

Сформировать массив B из элементов массива A в следующем порядке: элементы с индексами $i \leq \frac{N+1}{2}$ переместить на позиции с четными индексами массива B с сохранением их исходного порядка относительно друг друга, а оставшиеся элементы ($i > \frac{N+1}{2}$) разместить на позициях с нечетными индексами массива B также с сохранением их исходного порядка.

1. 4 балла

Была получена ассемблерная программа с помощью команды

```
gcc -O0 -S -masm=intel -fno-asynchronous-unwind-tables task.c
```

-O0 убирает любую оптимизацию, -S останавливает компилятор после порождения ассемблерного кода, -masm=intel устанавливает синтаксиса Intel, -fno-asynchronous-unwind-tables отвечает за удаление макросов. Далее были написаны комментарии к ассемблированному коду.

С помощью следующей команды был получен исполняемый файл из программы на си

```
gcc task.c -o c_out
```

А с помощью этой команды был получен исполняемый файл из ассемблерной программы

```
gcc -c task.s -o task.o && gcc task.o -o asm_out
```

1.1. Тестирование

1. Минимальный размер массива

   Ввод: 1 1

   Вывод ./c_out: 1

   Вывод ./asm_out: 1

2. Размер массива = 2

   Ввод: 2 1 2

   Вывод ./c_out: 1 2

   Вывод ./asm_out: 1 2

3. Размер массива = 3

   Ввод: 3 1 2 3

   Вывод ./c_out: 1 3 2

   Вывод ./asm_out: 1 3 2

4. Размер массива нечетный = 7

   Ввод: 7 1 2 3 4 5 6 7

   Вывод ./c_out: 1 5 2 6 3 7 4

   Вывод ./asm_out: 1 5 2 6 3 7 4

5. Размер массива четный = 8

   Ввод: 8 1 2 3 4 5 6 7 8

   Вывод ./c_out: 1 5 2 6 3 7 4 8

   Вывод ./asm_out: 1 5 2 6 3 7 4 8

Вывод - программы идентичны, тесты полностью соответствуют запрашеваемой логике.

2. 5 баллов

Моя программа изначально была написана с использованием локальных переменных и была разбита на функции. Также в моей ассемблерной программе изначально было написано много комментариев, включая моменты передачи параметров в функции. Параметры передаются через регистры rdi, rsi, edx. Внутри функций эти регистры переносятся на стек, начиная с QWORD PTR -24[rbp].

```
input:
        endbr64
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32                 # пролог
        mov     QWORD PTR -24[rbp], rdi  # array = rdi
        mov     DWORD PTR -28[rbp], esi  # size = esi
        mov     DWORD PTR -4[rbp], 0     # i = 0
        jmp     .L2                     # переход в проверку условия цикла
```

```
create_new_array:
        endbr64
        push    rbp
        mov     rbp, rsp                # пролог
        mov     QWORD PTR -24[rbp], rdi # array = rdi
        mov     QWORD PTR -32[rbp], rsi # new_array = rsi
        mov     DWORD PTR -36[rbp], edx # size = edx
        mov     DWORD PTR -16[rbp], 0   # i = 0
        mov     DWORD PTR -12[rbp], 0   # j = 0
        jmp     .L5
```

```
output:
        endbr64
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32                 # пролог
        mov     QWORD PTR -24[rbp], rdi # array = rdi
        mov     DWORD PTR -28[rbp], esi # size = esi
        mov     DWORD PTR -4[rbp], 0    # i = 0
        jmp     .L13                    # переход к проверке условия цикла
```

Сверху приведены три функции, и есть описание того что записывается из регистра на стек. Например в input из rdi, в который передают array, данные записываются в QWORD PTR -24[rbp].

Снизу приведен кусочек из .L21, в который мы переходим из функции main, в котором вызываются все описанные выше функции. Везде написаны комментарии что это именно и как оно передается в вызов через регистры.

```
        mov     QWORD PTR -64[rbp], rax # new_array[size]
        mov     edx, DWORD PTR -92[rbp] # edx = size
        mov     rax, QWORD PTR -80[rbp] # rax = array
        mov     esi, edx                # esi = size
        mov     rdi, rax                # rdi = array
        call    input                   # call input with rdi = array, rsi =
    size
        mov     edx, DWORD PTR -92[rbp] # edx = size
        mov     rcx, QWORD PTR -64[rbp] # rcx = new_array
        mov     rax, QWORD PTR -80[rbp] # rax = array
        mov     rsi, rcx                # rsi = new_array
        mov     rdi, rax                # rdi = array
        call    create_new_array        # call create_new_array with rdi = array
    , rsi = new_array, edx = size
```

```
        mov     edx, DWORD PTR -92[rbp] # edx = size
        mov     rax, QWORD PTR -64[rbp] # rax = new_array
        mov     esi, edx                # esi = size
        mov     rdi, rax                # rdi = new_array
        call    output                  # call output with rdi = new_array, rsi
    = size
        mov     eax, 0
```

3. 6 баллов

Для того чтобы уменьшить рефакторинг кода было принято решение заменить массивы на стеке на динамическое выделение массива.

Было:

```
int main() {
    int size;
    scanf("%d", &size);
    int array[size];
    int new_array[size];

    input(array, size);
    create_new_array(array, new_array, size);
    output(new_array, size);
    return 0;
}
```

Стало:

```
int main() {
    int size;
    scanf("%d", &size);
    int *array = malloc(size * sizeof(int));
    int *new_array = malloc(size * sizeof(int));

    input(array, size);
    create_new_array(array, new_array, size);
    output(new_array, size);
    free(array);
    free(new_array);
    return 0;
}
```

Была получена ассемблерная программа с помощью команды

```
    gcc -O0 -S -masm=intel -fno-asynchronous-unwind-tables task2.c
```

Сначала наверх были вынесены все переменные

```
  .section  .rodata
.input:
  .string "%d"
output1:
  .string "%d "
output2:
  .string "%d\n"
```

Функция input, вместо сохранения параметров на стеке были использованы регистры r13, r12, вместо локальной переменной i на стеке теперь используется rbx. Такие же манипуляции были проведены с output и create_new_array.

```
    input:
  push  rbp
```

```
  mov     rbp, rsp
  push  r13             # сохраняем r13, r12, rbx
  push  r12
  push  rbx
  sub     rsp, 8          # выравниваем стек
  mov     r13, rdi        # r13 = array
  mov     r12d, esi       # r12d = size
  mov     ebx, 0          # ebx = i = 0
  jmp     .L2
.L3:
    mov     rsi, r13
    lea     rdi, .input[rip]
    mov     eax, 0
    call    __isoc99_scanf@PLT    # call scanf with rdi = "%d", rsi = array + 4
   * i
    add     ebx, 1          # i++
    add     r13, 4          # сдвигаем array на 4
.L2:
  cmp     ebx, r12d               # if i < size -> continue
  jl      .L3
  add     rsp, 8
  pop     rbx
  pop     r12
  pop     r13
  pop     rbp
  ret
  .size input, .-input
```

Функция main, пушим регистры r13, r12, rbx чтобы записать туда r13 = size, r12 = array, rbx = new_array.

Переменную size необходимо предварительно всё таки положить на стек, т.к. необходимо проводить считывание по адресу, после этого она переносится в регистр, чтобы не обращаться часто к стеку. А затем проводятся вызовы функций передавая данные через регистры.

```

    .globl  main
    .type   main, @function
main:
    push    rbp
    mov rbp, rsp
    push    r13
    push    r12
    push    rbx
    sub     rsp, 24
    mov     rax, QWORD PTR fs:40
    mov     QWORD PTR -40[rbp], rax
    xor     eax, eax

    lea     rax, -44[rbp]
    mov     rsi, rax                # rax = n - хранится на стеке по адресу rbp
   - 44
    lea     rdi, .LC0[rip]          # rdi = "%d"
    mov     eax, 0
    call    __isoc99_scanf@PLT  # call scanf with rdi = "%d", rsi = n

    mov     r13d, DWORD PTR -44[rbp]    # r13d  = size
    movsx   rax, r13d
    sal     rax, 2
    mov     rdi, rax                # rdi = size * 4
    call    malloc@PLT              # call malloc with rdi = size * 4
    mov     r12, rax                # r12 = array = result of malloc
```

```
    movsx   rax, r13d
    sal     rax, 2
    mov     rdi, rax
    call    malloc@PLT              # call malloc with rdi = size * 4
    mov     rbx, rax                # rbx = array = result of malloc

    mov     esi, r13d
    mov     rdi, r12
    call    input                   # call input with rdi = array, esi = size

    mov     edx, r13d
    mov     rsi, rbx
    mov     rdi, r12
    call    create_new_array        # call input with rdi = array, rsi =
   new_array, edx = size

    mov     esi, r13d
    mov     rdi, rbx
    call    output                  # call output with rdi = new_array, esi =
   size

    mov     rdi, r12
    call    free@PLT                # free array

    mov     rdi, rbx
    call    free@PLT                # free new_array

    mov     eax, 0
    mov     rcx, QWORD PTR -40[rbp]
    xor     rcx, QWORD PTR fs:40
    je      .L17
    call    __stack_chk_fail@PLT
.L17:
    add     rsp, 24
    pop     rbx
    pop     r12
    pop     r13
    pop     rbp
    ret
    .size   main, .-main
```

3.1. Тестирование

1. Минимальный размер массива

   Ввод: 1 1

   Вывод ./c_ou2t: 1

   Вывод ./asm_out2: 1

2. Размер массива = 2

   Ввод: 2 1 2

   Вывод ./c_out2: 1 2

   Вывод ./asm_out2: 1 2

3. Размер массива = 3

   Ввод: 3 1 2 3

Вывод ./c_out2: 1 3 2

Вывод ./asm_out2: 1 3 2

4. Размер массива нечетный = 7

   Ввод: 7 1 2 3 4 5 6 7

   Вывод ./c_out2: 1 5 2 6 3 7 4

   Вывод ./asm_out2: 1 5 2 6 3 7 4

5. Размер массива четный = 8

   Ввод: 8 1 2 3 4 5 6 7 8

   Вывод ./c_out2: 1 5 2 6 3 7 4 8

   Вывод ./asm_out2: 1 5 2 6 3 7 4 8

Все тесты совпадают результатом с предыдущими программами, значит наш рефакторинг был корректным.

4. 7 баллов

Была переписана программа на C, с добавлением ввода из файлом через задание агрументов командной строки, также поддерживается ввод из stdin. Первым аргументом передается название файла ввода, вторым название файла вывода. Если название файла stdin, то ввод происходит через консоль.

Программа на C:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void input(int *array, int size, FILE *input_) {
    for (int i = 0; i < size; ++i) {
        fscanf(input_, "%d", &array[i]);
    }
}

// void create_new_array - не изменился

void output(int *array, int size, FILE *output_) {
    for (int i = 0; i < size - 1; i++) {
        fprintf(output_, "%d ", array[i]);
    }
    fprintf(output_, "%d\n", array[size - 1]);
}

int main(int argc, char *argv[]) {
    if (argc != 3) {
        printf("Please give input and output file");
        return 0;
    }
    FILE *input_;
    FILE *output_;
    if (strcmp(argv[1], "stdin") == 0) {
        input_ = stdin;
    } else {
        input_ = fopen(argv[1], "r");
```

```
    }

    if (strcmp(argv[2], "stdout") == 0) {
        output_ = stdout;
    } else {
        output_ = fopen(argv[2], "w");
    }

    int size;
    fscanf(input_, "%d", &size);
    int size_ = size;
    int *array = malloc(size_ * sizeof(int));
    int *new_array = malloc(size_ * sizeof(int));

    input(array, size_, input_);
    create_new_array(array, new_array, size_);
    output(new_array, size_, output_);
    free(array);
    free(new_array);
    if (strcmp(argv[1], "stdin")) {
        fclose(input_);
    }
    if (strcmp(argv[2], "stdout")) {
        fclose(output_);
    }
    return 0;
}
```

Из этого кода был получен ассемблерный, и проставлялись комментарии в соответствии с предыдущей версией. Т.е. была взята предыдущая версия и были изменены новые моменты, также учитывая критерии на 6 были использованы регистры по максимуму.

Функции были вынесены в файл func.s, а main в файле main.s . Также теперь все функции из файла func.s сделаны глобальными, чтобы можно было собрать исполняемый файл.

Чтобы собрать программу необходимо выполнить следующую команду

```
gcc main.s func.s -o asm_out3
```

1. Минимальный размер массива

   Аргументы: stdin stdout

   stdin: 1 1

   stdout: 1

2. Размер массива нечетный = 7

   Аргументы: stdin output.txt

   stdin: 7 1 2 3 4 5 6 7

   stdout: 1 5 2 6 3 7 4

3. Размер массива четный = 8

   Аргументы: input.txt output.txt

   input.txt: 8 1 2 3 4 5 6 7 8

   output.txt: 1 5 2 6 3 7 4 8

Эти результаты верные, соответствуют прошлым версиям и показывают что программа корректно работает со всеми типами ввода и вывода (за исключением случая если программе отдаётся несуществующий файл, этот случай также можно было дополнительно обработать).