Мерочкин Илья БПИ 218

Индивидуальное домашнее задание по архитектуре вычислительных систем №1.
Вариант 6

6. Сформировать массив B, состоящи из элементов массива A, значения которых кратны введенному числу X.

4 балла

Код на C, лежит в файле asm.c:

```
#include <stdio.h>

int main() {
    unsigned int sz;
    scanf("%u", &sz);
    int a[sz];
    for (int i = 0; i < sz; ++i) {
        scanf("%d", &a[i]);
    }
    int x;
    scanf("%d", &x);
    int b_size = 0;
    for (int i = 0; i < sz; ++i) {
        if (a[i] % x == 0) {
            ++b_size;
        }
    }
    int idx = 0;
    int b[b_size];
    for (int i = 0; i < sz; ++i) {
        if (a[i] % x == 0) {
            b[idx] = a[i];
            ++idx;
        }
    }
    for (int i = 0; i < b_size; ++i) {
        printf("%d ", b[i]);
    }
```

```
    printf("\n");
    return 0;
}
```

Ассемблерная программа, лежит в файле asm.s (некоторые комментарии не влезли в одну строчку, поэтому расположены на двух):

```
        .file   "main.c"
        .intel_syntax noprefix
        .text
        .globl  a
        .bss
        .align 32
        .type   a, @object
        .size   a, 400000
a:
        .zero   400000
        .globl  b
        .align 32
        .type   b, @object
        .size   b, 400000
b:
        .zero   400000
        .section        .rodata
.LC0:
        .string "%d "
        .text
        .globl  main
        .type   main, @function
main:
        endbr64
        push    rbp
        mov     rbp, rsp
        sub     rsp, 48                         # сдвиагем регистр rsp на 48
        mov     DWORD PTR -36[rbp], edi         # записываем edi в аргумент argc
        mov     QWORD PTR -48[rbp], rsi         # записываем rsi в аргумент argv
        mov     eax, DWORD PTR -36[rbp]         # записываем в eax значение argc
        sub     eax, 2                          # вычитаем из регистра eax 2
        mov     DWORD PTR -16[rbp], eax         # присваиваем переменной a_sz значение
регистра eax
        mov     eax, DWORD PTR -36[rbp]
        cdqe
        sal     rax, 3
```

```
        lea     rdx, -8[rax]
        mov     rax, QWORD PTR -48[rbp]
        add     rax, rdx
        mov     rax, QWORD PTR [rax]
        mov     rdi, rax
        call    atoi@PLT                        # вызываем функцию atoi
        mov     DWORD PTR -20[rbp], eax
        mov     DWORD PTR -4[rbp], 0    # записываем в локальную переменную b_sz 0
        mov     DWORD PTR -8[rbp], 0     # записываем в локальную переменную idx 0
        mov     DWORD PTR -12[rbp], 0    # записываем в перменную i 0
        jmp     .L2
.L3:
        mov     eax, DWORD PTR -12[rbp]
        cdqe
        add     rax, 1
        lea     rdx, 0[0+rax*8]
        mov     rax, QWORD PTR -48[rbp]
        add     rax, rdx
        mov     rax, QWORD PTR [rax]
        mov     rdi, rax
        call    atoi@PLT
        mov     edx, DWORD PTR -12[rbp]
        movsx   rdx, edx
        lea     rcx, 0[0+rdx*4]
        lea     rdx, a[rip]
        mov     DWORD PTR [rcx+rdx], eax            # a[i] = atoi(argv[i + 1]) (строка9).
Записываем в a[i] значение из argv[i + 1]
        add     DWORD PTR -12[rbp], 1
.L2:                            # внутренность цикла for (i = 0; i < a_sz; ++i) (8 строка)
        mov     eax, DWORD PTR -12[rbp]             # кладем в eax переменную i
        cmp     eax, DWORD PTR -16[rbp]             # сравниваем регистр eax и a_sz
        jl      .L3
        mov     DWORD PTR -12[rbp], 0               # приравниваем переменную i к 0
        jmp     .L4
.L6:
        mov     eax, DWORD PTR -12[rbp]
        cdqe
        lea     rdx, 0[0+rax*4]
        lea     rax, a[rip]
        mov     eax, DWORD PTR [rdx+rax]
        cdq
        idiv    DWORD PTR -20[rbp]
        mov     eax, edx
        test    eax, eax
```

```
        jne     .L5
        add     DWORD PTR -4[rbp], 1     # увеличиваем значение переменной b_size на 1
.L5:
        add     DWORD PTR -12[rbp], 1
.L4:
        mov     eax, DWORD PTR -12[rbp]          # кладем в eax значение переменной i
        cmp     eax, DWORD PTR -16[rbp]          # сравниваем eax и переменную a_sz
        jl      .L6
        mov     DWORD PTR -12[rbp], 0            # присваиваем перменной i значение 0
        jmp     .L7
.L9:
        mov     eax, DWORD PTR -12[rbp]
        cdqe
        lea     rdx, 0[0+rax*4]
        lea     rax, a[rip]
        mov     eax, DWORD PTR [rdx+rax]
        cdq
        idiv    DWORD PTR -20[rbp]
        mov     eax, edx
        test    eax, eax
        jne     .L8
        mov     eax, DWORD PTR -12[rbp]
        cdqe
        lea     rdx, 0[0+rax*4]
        lea     rax, a[rip]
        mov     eax, DWORD PTR [rdx+rax]
        mov     edx, DWORD PTR -8[rbp]
        movsx   rdx, edx
        lea     rcx, 0[0+rdx*4]
        lea     rdx, b[rip]
        mov     DWORD PTR [rcx+rdx], eax      # присваиваем b[idx] значение eax (строка
18)
        add     DWORD PTR -8[rbp], 1          # увеличиваем idx на 1
.L8:
        add     DWORD PTR -12[rbp], 1
.L7:
        mov     eax, DWORD PTR -12[rbp]
        cmp     eax, DWORD PTR -16[rbp]
        jl      .L9
        mov     DWORD PTR -12[rbp], 0
        jmp     .L10
.L11:                                 # внутренность цикла for (i = 0; i < b_size; ++i) (22 строка)
        mov     eax, DWORD PTR -12[rbp]
        cdqe
```

```
        lea     rdx, 0[0+rax*4]
        lea     rax, b[rip]
        mov     eax, DWORD PTR [rdx+rax]
        mov     esi, eax
        lea     rax, .LC0[rip]
        mov     rdi, rax
        mov     eax, 0
        call    printf@PLT                      # вызов функции printf
        add     DWORD PTR -12[rbp], 1
.L10:
        mov     eax, DWORD PTR -12[rbp]
        cmp     eax, DWORD PTR -4[rbp]
        jl      .L11
        mov     edi, 10
        call    putchar@PLT
        mov     eax, 0                  # кладем в eax 0
        leave                           # выход из функции
        ret                             # выход из функции
```

Ассемблерную программу, откомпилированную без оптимизирующих и отладочных опций, добавлены комментарии, поясняющие эквивалентное представление переменных в программе на C.
Команда, которой это было сделано: gcc -O0 -Wall -masm=intel -S asm.c -o asm.s

Из ассемблерной программы убраны лишние макросы за счет использования соответствующих аргументов командной строки и за счет ручного редактирования исходного текста ассемблерной программы.
Команда, которой это было сделано: gcc -masm=intel -fno-asynchronous-unwind-tables -fno-jump-tables -fno-stack-protector -fno-exceptions asm.c -S -o asm.s

Тестовые прогоны для обеих команд:

Тест 1
Входные данные:
1 2 3 4 5 6 7 8
2
Выходные данные программы на C:
2 4 6 8
Выходные данные программы на ассемблере:
2 4 6 8

Тест 2
Входные данные:
9 3 2 1 7 12 5
3
Выходные данные программы на C:
9 3 12
Выходные данные программы на ассемблере:
9 3 12

Тест 3:
1 2 3
4
Выходные данные программы на C:

Выходные данные программы на ассемблере:

Поведение обеих команд эквивалентно на всех тестовых наборах.