МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образование

учреждение высшего образования

«Ижевский государственный технический университет имени М. Т. Калашникова»

(ФГБОУ ВО «ИжГТУ имени М. Т. Калашникова»)

Институт непрерывного профессионального образования

Лабораторная работа №2

по дисциплине «Архитектура компьютерных систем»

(Вариант 2)

|  |  |
| --- | --- |
|  | Выполнил студент группы: 5-19-1 од  Тепляков Георгий Константинович  Принял: преподаватель  Ситников Василий Вадимович |

Ижевск

2018

**Цель работы**

Написать программу на языке ассемблера, скомпилировать ее с помощью компилятора FASM и отладить программу, используя отладчик Turbo Debugger.

**Задание**

Создать массив из N чисел. Определить, сколько нулей находится до минимального значения в массиве (если несколько минимальных значений, то минимальным считать последнее значение, если минимальное значение находится в начале массива, то вывести сообщение об этом и прекратить обработку) и среднее арифметическое положительных чисел, находящихся после минимального значения (если положительных чисел нет, то вывести сообщение об этом)

1 Краткие теоретические сведения и понятия

**1.1 Описание языков Ассемблера**

Все языки программирования разделяются по уровням: низкий и высокий. Любой из синтаксической системы «семейки» Ассемблера отличается тем, что объединяет сразу некоторые достоинства наиболее распространенных и современных языков. С другими их роднит и то, что в полной мере можно использовать систему компьютера.

Отличительной особенностью компилятора является простота в использовании. Этим он отличается от тех, которые работают лишь с высокими уровнями. Если взять во внимание любой такой язык программирования, Ассемблер функционирует вдвое быстрее и лучше. Для того чтобы написать в нем легкую программу, не понадобится слишком много времени.

**1.2 Отладчик Turbo Debugger**

Отладчик Turbo Debugger представляет собой набор инструментальных средств, позволяющий отлаживать программы на уровне исходного текста и предназначенный для программистов, использующих семейство компиляторов Borland. В пакет отладчика входят набор выполняемых файлов, утилит, справочных текстовых файлов и примеров программ.

Turbo Debugger позволяет вам отлаживать программы для Microsoft Windows, Windows 32s, Windows NT и DOS. Многочисленные перекрывающие друг друга окна, а также сочетание спускающихся и раскрывающихся меню обеспечивают быстрый, интерактивный пользовательский интерфейс. Интерактивная, контекстно-зависимая справочная система обеспечит вас подсказкой на всех стадиях работы. Кроме того, Turbo Debugger полный набор средств отладки:

* вычисление любых выражений языка Си, C++, Pascal и Assemb ler;
* полное управление выполнением программы, включая программ ную анимацию;
* доступ на нижнем уровне к регистрам процессора и системной памяти;
* полные средства проверки данных;
* развитые возможности задания точек останова и регистрации;
* трассировка сообщений Windows, включая точки останова по сообщениям;
* обратное выполнение;
* поддержка удаленной отладки, в том числе для Windows;
* полная поддержка объектно-ориентированного программирования, включая просмотр классов и проверку объектов;
* макрокоманды в виде последовательности нажатий клавиш, ускоряющие выполнение команд;
* копирование и вставка между окнами и диалогами;
* контекстно-зависимые меню;
* возможность отладки больших программ;
* диалоговые окна, позволяющие вам настроить параметры отладчика;
* возможность отладчик 16- и 32-разрядных программ Windows (для 32-разрядной отладки имеется отладчик TD32);
* обработка исключительных ситуаций операционной системы, а также С и С++;
* сохранение сеанса;
* поддержка нитей для мультинитевого программирования Win dows NT;
* возможность подключения готовых к выполнению в Windows процессов;
* возможность выбора для элементов, выводимых в Turbo Debug ger, национального порядка сортировки.

**1.3 Команды языка**

Команды языка с их описанием:

* точка с запятой — это символ комментария;
* *«***use16***»* сообщает FASM’у, что нужно генерировать 16-битный код;
* «**org 100h**» объясняет FASM’у, что следующие команды и данные будут располагаться в памяти, начиная с адреса 100h. Дело в том, что при загрузке нашей программы в память, DOS размещает в первых 256 байтах (с адресов 0000h — 00FFh) свои служебные данные;
* **AX, BX, CX и DX** используются для хранения данных и выполнения различных арифметических и логических операций. Кроме того, каждый из этих регистров поделён на 2 части по 8-бит, с которыми можно работать как с 8-битными регистрами (AH, AL, BH, BL, CH, CL, DH, DL). Младшие части регистров имеют в названии букву L (от слова Low), а старшие H (от слова High). Некоторые команды неявно используют определённый регистр, например, CX может выполнять роль счетчика цикла;
* **SI**  и **DI** Индексные регистры.SI (Source Index) содержит индекс источника, DI (Destination Index) — индекс приёмника, хотя можно использовать и как регистры общего назначения;
* **BP** и **SP** это регистры-указатели используются для работы со стеком. BP (Base Pointer) позволяет работать с переменными в стеке. Его также можно использовать в других целях. SP (Stack Pointer) указывает на вершину стека. Он используется командами, которые работают со стеком;
* **mov** приёмник, источник – команда пересылки данных. Копирует содержимое источника в приёмник, источник не изменяется. Операнды команды mov могут быть как регистрами, так и переменными, но одновременно оба операнда не могут быть переменными;
* **lea** загружает указатель на объект, который вы адресуете. Цель LEA - дать возможность выполнить нетривиальный расчет адреса и сохранить результат для последующего использования;
* **xor** приёмник, источник – логическое исключающее ИЛИ. Выполняет побитое логическое исключающее ИЛИ над приёмником и источником, результат заносится в приёмник. Часто используется для обнуления регистров. Этой командой следует пользоваться для обнуления регистров;
* **cmp** операнд1, операнд2 – по сути вычитание операнда2 из операнда1, только операнды не меняется (команда меняет только флаги). [С помощью этой команды обычно выполняются условные переходы](http://osinavi.ru/asm/CMP.html);
* **test** используется для сравнения значения регистра с нулем. После этой операции флаги нуля, знака и чётности либо устанавливаются, либо сбрасываются;
* **IP** (*Instruction Pointer*) указатель команд**,** содержит адрес команды (в сегменте кода). При выполнении обычных команд значение IP увеличивается на размер выполненной команды. Существуют также команды передачи управления, которые изменяют значение IP для осуществления переходов внутри программы;
* [**add**](http://asmworld.ru/spravochnik-komand/add/) предназначена команда  для сложения двух чисел. Она работает как с числами со знаком, так и с числами без знака;
* [**sub**](http://asmworld.ru/spravochnik-komand/sub/) выполняется вычитание с помощью команды . Результат также помещается на место первого операнда и выставляются флаги;
* [**mul**](http://asmworld.ru/spravochnik-komand/mul/) предназначена для умножения чисел без знака;
* [**cwd**](http://asmworld.ru/spravochnik-komand/cwd) (*Convert Word to Double word* — преобразовать слово в двойное слово). У этих команд нет явных операндов. Команда [CWD](http://asmworld.ru/spravochnik-komand/cwd) преобразует слово, находящееся в регистре AX, в двойное слово в регистрах DX:AX;
* [**loop**](http://asmworld.ru/spravochnik-komand/loop) предназначена для организации цикла.  У этой команды один операнд — имя метки, на которую осуществляется переход;
* **метка** представляет собой символическое имя, вместо которого компилятор подставляет адрес. Обычно метки используются для организации переходов, циклов или каких-то манипуляций с данными. Метка объявляется очень просто: достаточно в начале строки написать имя и поставить двоеточие;
* **анонимная метка** это метка с именем *@@.* В программе можно объявлять сколько угодно анонимных меток, но обратиться получится только к ближайшей. Для этого существуют специальные имена: вместо *@b* (или *@r*) FASM подставляет адрес предыдущей анонимной метки, а вместо *@f* — адрес следующей анонимной метки. Этого, как правило, достаточно, чтобы реализовать простой цикл, переход или проверку условия. Таким образом можно избавиться от большого количества «неанонимных» меток;
* **push** добавление элемента на вершину стека;
* **pop** извлечение элемента с вершины стека;
* [**call**](http://asmworld.ru/spravochnik-komand/call) предназначены для работы с процедурами. С помощью команды [CALL](http://asmworld.ru/spravochnik-komand/call) выполняется вызовпроцедуры. Эта команда работает почти также, как команда безусловного перехода но с одним отличием — одновременно в стек сохраняется текущее значение регистра IP. Это позволяет потом вернуться к тому месту в коде, откуда была вызвана процедура;
* **idiv** производит целочисленное деление с остатком [знаковых](http://www.club155.ru/x86data-common#integer) целочисленных операндов;
* **ret** команда "возврат из подпрограммы" последовательно выгружает старший и младший байты счетчика команд из стека, уменьшая указатель стека на 2;
* **neg** изменяет знак числа на противоположный, то есть положительное число делает отрицательным, а отрицательное - положительным. При этом число преобразуется в двоичный дополнительный код. NEG изменяет флаги CF, ZF, SF, OF, AF, PF. Значения [флагов](http://av-assembler.ru/asm/afd/asm-flags-register.htm) зависят от результата операции отрицания;
* **inc** команда "инкремент" выполняет прибавление "1" к указанной переменной и влияет на флаги. Начальное значение 0FFH перейдет в 00Н;
* **условный переход** (jnz, jg, jnp, jae, jng, jns…) осуществляется, если выполняется определённое условие, заданное флагами процессора. Состояние флагов изменяется после выполнения арифметических, логических и некоторых других команд. Если условие не выполняется, то управление переходит к следующей команде.

2 Листинг программы с комментариями

use16 ;16-битный режим

org 100h ;смещение програмы в памяти (com-файл)

jmp start ;Безусловный переход на метку start

;----------------------------------------------------------

randSeed dw 0

pmin dw mas

msg1 db 'Array: $' ;массив:

msg2 db 13,10,'Min: $' ;минималный:

msg3 db 13,10,'Min at start$' ;мин в начале:

msg4 db 13,10,'Zeros before min: $' ;нули до мин:

msg5 db 13,10,'No positives after min$' ;без положительных результатов после мин.

msg6 db 13,10,'Avg of positives after min = $' ;средние положительные значения

после мин.

;----------------------------------------------------------

start:

;----------------------------------------------------------

N equ 20 ;N кол-во элементов в массиве

mas dw N dup(0) ;исходный массив, размер каждого элемента 2 байта;

masL dw 10 ;значение в массиве слева (со знаком -)

masR dw 20 ;значение в массиве справа (со знаком +), с вычетом левого(masL)

;masR-masL

;----------------------------------------------------------

mov bx,[masR] ;значение в массиве справа

mov ah,9 ;функция вывода сообщения на экран

mov dx,msg1 ;выводимое сообщение

int 21h ;выводим на экран

call randomize ;инициализация рандома

mov si,mas ;начало масива

mov cx,N ;количество элементов массива

;инициализация рандомными значениями и вывод массива

outmas: call rand

sub ax,[masL] ;значение в массиве слева

mov [si],ax ;берем элемент массива

call printdec ;преобразовываем число в строку и выводим

mov ah,2 ;выводим символ

mov dl,' ' ;разделительный пробел

int 21h ;выводим

add si,2 ;переходим к следующему элементу массива

loop outmas ;продолжаем вывод

mov si,mas ;начало масива

mov cx,N ;количество элементов массива

;получение случайного числа

;bx - предел числа (0..bx-1)

; линейный конгруэнтный метод - один из методов генерации псевдослучайных чисел, применяется в простых случаях

rand:

push cx ;сохраняем регистры

mov ax,[randSeed]

mov cx,4e6dh ;коэффициенты генератора

mul cx

add ax,3031h ;коэффициенты генератора ;(3031h - 12345)

xor dx,dx ;обнулить старшую часть делимого

mov [randSeed],ax ;сохраниь предыдущее случайное

idiv bx ;ограничить случайную величину по модулю bx

mov ax,dx ;взять остаток от деления

pop cx ;восстанавливаем регистры

ret ;выход из подпрограммы

getTime: ;берет число тиков из области BIOS

;Возвращается в ax

push ds ;сохраняем регистры

xor ax,ax

mov ds,ax

mov ax,word [ds:46ch] ;BIOS timer tick counter ;(чтобы рандом при каждом

;запуске был разным)

pop ds ;восстанавливаем регистры

ret ;выход из подпрограммы

randomize:

call getTime

mov [randSeed],ax

ret ;выход из подпрограммы

;преобразование числа из ах в десятичную строку и вывод на экран

;ax - число

printdec:

push cx ;сохраняем регистры

push dx ;сохраняем регистры

push bx ;сохраняем регистры

push si ;сохраняем регистры

mov si,ax ;запоминаем число

mov bx,10 ;основание системы

xor cx,cx ;в сх будет количество цифр в десятичном числе

test ax,ax ;если число положительное

jns @@m1a ;то пропускаем

neg ax ;если отрицательное, то меняем знак

@@m1a: xor dx,dx ;очистить старшую часть делимого

div bx ;делим число на степени 10

push dx ;и сохраняем остаток от деления (коэффициенты при степенях)

;в стек

inc cx ;увеличиваем количество десятичных цифр числа

test ax,ax ;после деления остался 0?

jnz @@m1a ;если нет, продолжаем

test si,si ;число отрицательное?

jns @@m3a ;если нет, пропустить

push -3 ;иначе дописать перед числом минус

inc cx ;увеличить количество цифр числа

@@m3a: mov ah,2 ;ф-я вывода символа

@@m2a: pop dx ;взять из стека цифру числа

add dl,'0' ;преобразовываем цифру в ASCII символ

int 21h ;вывести на экран

loop @@m2a ;все цифры

pop si ;восстанавливаем регистры

pop bx ;восстанавливаем регистры

pop dx ;восстанавливаем регистры

pop cx ;восстанавливаем регистры

ret ;выход из подпрограммы

mov ax,4C00h ;\

int 21h ;/ Завершение программы

3 Результат выполнения программы

Первый исход выполнения программы, представлен на (рис.1). В данном случае сгенерировался рандомный массив размером 20 состоящий из знаковых и беззнаковых чисел, в котором минимальный элемент -8. В данном случае до минимального значения нет ни одного 0. Также в этом случае, мы можем посчитать среднее арифметическое положительных чисел, находящихся после минимального.

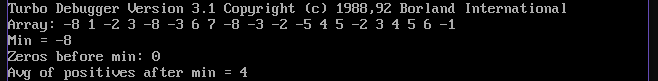


Рис.1 – Вид программы в отладчике Turbo Debugger в консоли (первый исход)

Второй исход выполнения программы, представлен на (рис.2). В данном случае сгенерировался рандомный массив размером 20 состоящий из знаковых и беззнаковых чисел, в котором минимальный элемент -9. В данном случае до минимального значения встречаются два 0. Также в этом случае, мы можем посчитать среднее арифметическое положительных чисел, находящихся после минимального.

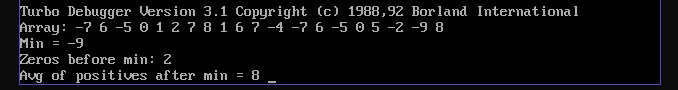


Рис.2 – Вид программы в отладчике Turbo Debugger в консоли (второй исход)

Для того чтобы проверить сообщения выводимые программой изменил размер массива с 20 до 5, N equ 20 на N equ 5.

Третий исход выполнения программы, представленный на (рис.3). В данном случае сгенерировался рандомный массив размером 5 состоящий из знаковых и беззнаковых чисел, в котором минимальный элемент -10. В данном случае до минимального значения нет ни одного 0. Также мы можем наблюдать, что после минимального значения нет положительных чисел и программа, выводит сообщение об этом.

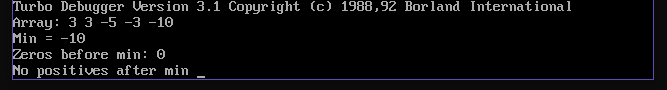


Рис.3 – Вид программы в отладчике Turbo Debugger в консоли (третий исход)

Четвертый исход выполнения программы, представленный на (рис.4). В данном случае сгенерировался рандомный массив размером 5 состоящий из знаковых и беззнаковых чисел, в котором минимальный элемент -5. Также мы можем наблюдать, что минимальное значения находится в начале массива и программа, выводит сообщение об этом и прекращает обработку.

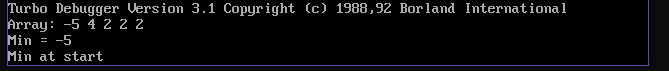


Рис.4 – Вид программы в отладчике Turbo Debugger в консоли (четвертый исход)

**Вывод**

В ходе выполнения данной лабораторной работы была написана программа, которая создает массив из N чисел. Определяет, сколько нулей находится до минимального значения в массиве и находит среднее арифметическое положительных чисел, находящихся после минимального значения.