**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ,
СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»
(СПбГУТ)**

Факультет «Информационных систем и технологий»
Кафедра «Интеллектуальных систем автоматизации и управления»

Направление подготовки: 09.03.02 - Информационные системы и технологии

Направленность (профиль): Системное и прикладное программирование информационных систем

КУРСОВОЙ ПРОЕКТ

по дисциплине:
Вычислительные машины, системы и сети

на тему:
«Разработка программы для тестирования на внимательность»

Выполнила студентка группы: ИСТ-261

Уласик В.В.
Фамилия И. О.

Руководитель ассистент каф. ИСАУ
оценка *уч. степень, уч. звание*

Шабанов А.П.
дата, подпись *Фамилия И. О.*

Члены комиссии:

Волынкин П.А.
дата, подпись *Фамилия И.О.*

дата, подпись *Фамилия И.О.*

Санкт-Петербург
2023

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

на курсовую работу по дисциплине
«Вычислительные машины, системы и сети»

Написать на языке TurboAssembler программу, которая в случайном месте выводит на экран монитора простейшие фигуры (квадрат, треугольник, ромб, круг) разных цветов (синий, красный, зеленый, желтый, фиолетовый). В рамках одного сеанса должно появиться 20 фигур. Размер фигур должен составлять 1% от ширины экрана. Пользователь должен успеть как можно быстрее нажать курсором мыши на фигуре. После верного нажатия фигура пропадает и появляется следующая фигура. По окончании нажатий программа выводит на экран и сохраняет в файле время, затраченное на нажатие на 20 фигурах. После создания программы провести исследование зависимости быстроты реакции от цвета фигур, формы фигур, времени эксперимента (утро, день, вечер, ночь). Для каждого из трех видов исследований менять только один параметр (или цвет, или форму, или время суток) при постоянстве остальных двух параметров.

Структура, объем, содержание, оформление, а также подготовка к защите и защита курсовой работы, определены в методических указаниях по курсовому проектированию: (Волынкин П.А. Методические указания по курсовому проектированию по дисциплине «Вычислительные машины, системы и сети» для студентов очной и заочной форм обучения (бакалавриат). 2023г.)

Преподаватель: _____ к.т.н. доцент кафедры ИСАУ Волынкин П.А.

Оглавление

ВВЕДЕНИЕ

Assembler — это низкоуровневый язык программирования, напрямую связанный с архитектурой компьютера. В отличие от высокоуровневых языков, таких как C++ или Java, Assembler оперирует непосредственно машинными командами процессора. Это делает его мощным инструментом для написания программ, но также более сложным и трудоемким для разработчика.

Особенности и преимущества:

- Управление: Assembler позволяет программисту полностью контролировать аппаратные ресурсы компьютера, что особенно важно в областях, требующих максимальной производительности;
- Эффективность: программы на Assembler могут быть оптимизированы для конкретной архитектуры процессора, что обеспечивает высокую производительность;
- Прямой доступ к памяти: Assembler позволяет работать с памятью напрямую, что может быть полезно при управлении большими объемами данных.

Недостатки и ограничения:

- Сложность: написание кода на Assembler требует глубокого понимания аппаратной архитектуры и инструкций процессора, что делает его менее доступным для начинающих разработчиков;
- Перенос: программы, написанные на Assembler, часто не переносятся между разными архитектурами процессоров, что затрудняет их использование на различных платформах;
- Масштабируемость: в разработке больших и сложных программ Assembler может быть неэффективным из-за своей низкоуровневости;

Код на Assembler обычно состоит из секций, таких как .data (для данных), .text (для исполняемого кода) и других, в зависимости от конкретной архитектуры и компилятора;

Команды Assembler представляют собой машинные инструкции процессора и обычно записываются в мнемонической форме, например, mov для перемещения данных, add для сложения и т. д.;

Assembler использует регистры процессора для временного хранения данных. Регистры обычно обозначаются буквами (например, eax, ebx) и могут использоваться для арифметических операций, хранения адресов памяти и других задач;

Assembler остается важным инструментом в области системного программирования и низкоуровневой оптимизации. В ходе развития технологий использование Assembler, вероятно, будет сохраняться в специализированных областях, где необходима максимальная производительность и полный контроль над аппаратурой.

ОСНОВНАЯ ЧАСТЬ

Глава 1. Алгоритм решения поставленной задачи

Целью данного проекта является разработка программы на языке TurboAssembler, решающей задачу измерения времени реакции пользователя на появление простейших геометрических фигур различных цветов на экране монитора.

Алгоритм решения:

1) Инициализация:
- Определение параметров программы, таких как размер экрана, количество фигур, и характеристики фигур (цвет, форма);
- Инициализация переменных для хранения времени реакции и других параметров.
2) Генерация и отображение фигур:
- Запуск цикла для генерации случайных цветов и форм для 20 фигур;
- Отображение каждой фигуры на экране.
3) Ожидание пользовательского ввода:
- Ожидание нажатия курсора мыши;
- Получение координат щелчка.
4) Обработка результатов:
- Проверка, попал ли пользователь внутрь текущей фигуры;
- Если пользователь попал, убрать фигуру и запомнить время реакции.
5) Сохранение результатов:
- Сохранение в файл времени, затраченного на нажатие 20 фигур.
6) Исследование зависимости:
- Проведение трех серий экспериментов, меняя поочередно цвет, форму и время суток, при этом остальные параметры остаются неизменными;
- Для каждой серии экспериментов фиксирование времени реакции.

7) Анализ результатов:

- Сравнение результатов для различных параметров.

8) Выводы и заключение:

- Суммирование результатов исследования.
- Формулирование выводов о влиянии цвета, формы и времени суток на быстроту реакции пользователя.

Блок схема алгоритма представлена на рисунке 1

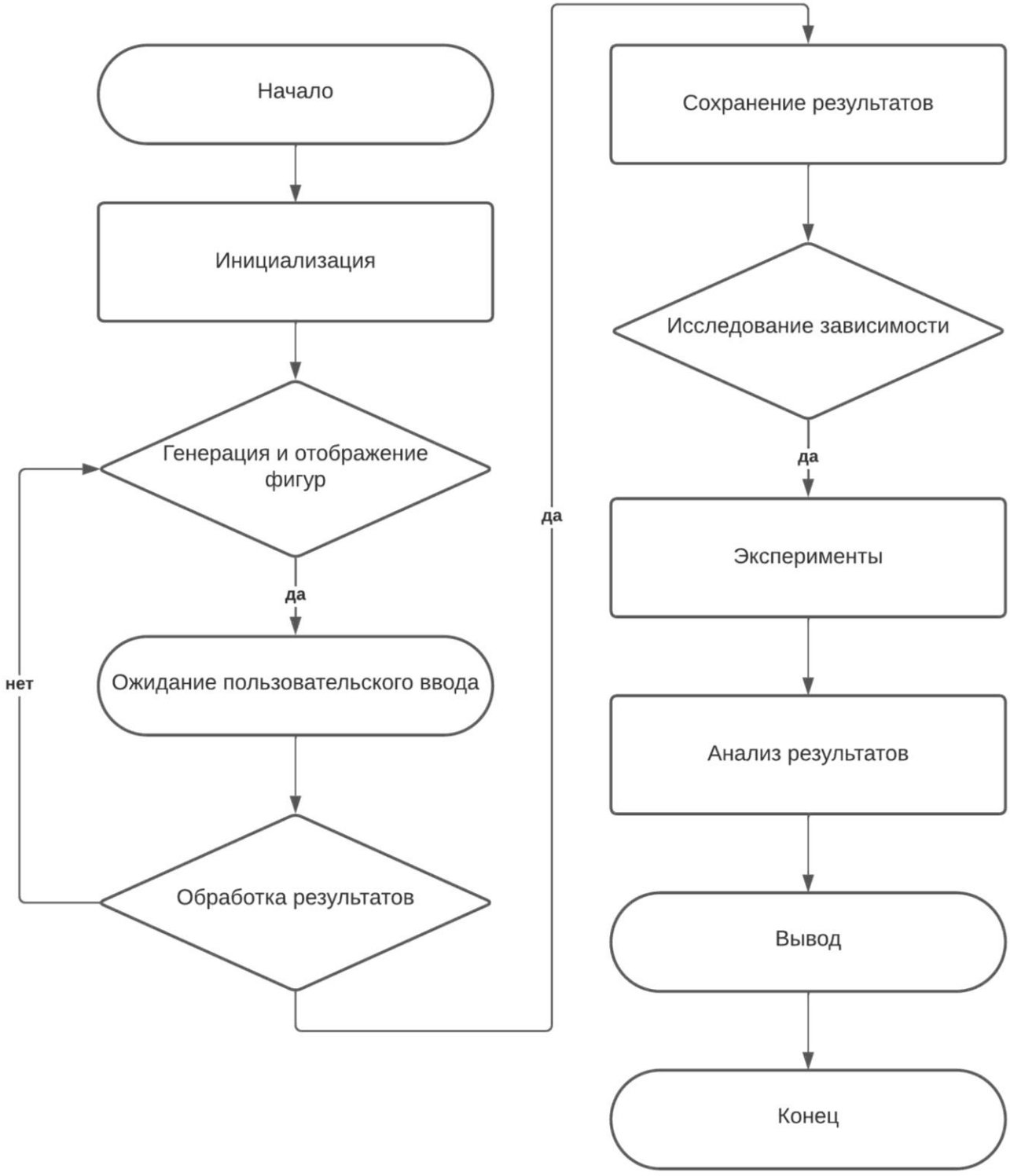

Рисунок 1 – Алгоритм программы

Глава 2. Исследование на быстроту реакции

Цель данного исследования заключается в выяснении влияния изменений цвета и формы фигур, а также времени суток на быстроту реакции участников взаимодействия с программой. Каждое исследование направлено на изучение воздействия одного конкретного параметра при постоянстве остальных двух.

Подготовка к исследованию:

- Разработана программа, способная случайным образом генерировать фигуры (квадраты, треугольники, ромбы, круги) разнообразных цветов;
- Для замера времени введен файл results.txt;
- Реализована логика отслеживания нажатий курсора мыши, проверки правильности нажатия и замены фигур после верного нажатия.

Определение параметров:

- Выбран параметр для исследования: цвет и форма фигуры;
- План исследования предусматривает изменение только одного параметра за раз, оставляя другой постоянным.

Исследование зависимости от цвета фигур:

- Запуск программы с уже изменённой формой фигуры, оставляя цвет тот же и наоборот:
- Фиксирование результатов времени (реакции).

Исследование зависимости от формы фигур:

- Зафиксированы результаты времени реакции на ромб зелёного цвета;
- Зафиксированы результаты на круг серого цвета.

Проведенные исследования позволили выделить следующие основные результаты:

1) Зависимость от цвета:

– Наблюдается различие в быстроте реакции участников в зависимости от цвета фигур. Фигуры с высоким контрастом (зелёный, красный) вызывают более быструю реакцию.

2) Зависимость от формы:

– Форма фигур также оказывает влияние на время реакции. Если фигура имеет большой размер – реакция будет быстрее.

Анализ результатов:

Замер времени происходит следующим образом:

Время реакции фиксируется в начале и конце теста, и расчет производится вычитанием начального времени из конечного, предоставляя значение в секундах. Как пример: сейчас 4 часа 23 минуты 16 секунд. Из этого - берём минуты умножаем их на 60, потому что в одной минуте 60 секунд: 23*60 и прибавляем к этому 16 секунд. Таким образом получим сколько секунд прошло с начала 4-го часа. Прошло 23*60+16=1396 секунд.

Затем повторный замер. Сейчас 4 часа 26 минут 19 секунд. 26 * 60 и прибавляю 19: 26*60+19=1579. Таким образом, число секунд, прошедших между точками, где замеряли время будет равно: 1579–1396 = 183 секунды;

Формула реакции представлена ниже:

$$\text{Временной интервал} = (\text{Часы} \times 3600) + (\text{Минуты} \times 60) + \text{Секунды}$$

Коэффициент 3600 используется, потому что в одном часе 60 минут, и в каждой минуте 60 секунд. Поэтому, чтобы преобразовать часы в секунды, умножаем количество часов на 60 (минут в часе) и затем на 60 (секунд в минуте), что дает 60 * 60 = 3600.

Для детализации исследования, рассмотрим конкретные результаты двух участников:

Процент реагирования будем считать по формуле:

$$\left(\frac{\text{Время реакции}}{\text{Максимально допустимое время}}\right) \times 100\%$$

Как пример, пусть максимально допустимое время будет равно 60 секунд. Время реакции представляет собой время, которое пользователь затрачивает на реакцию на появление фигур. Максимальное допустимое время определяет максимальное разрешенное время для завершения реакции на 20 фигур и равно 60 секундам. Результат выражается в процентах, отражая долю времени, затраченную пользователем, относительно максимально разрешенного времени. Чем ниже полученный процент к 100%, тем более эффективной была реакция пользователя в установленные временные рамки.

Если результат превышает 100%, то просто ограничиваю его до 100%.

- Пользователь 1 (Тусклые цвета, маленькие фигуры):
  - Фигура: круг;
  - Цвет: серый;
  - Время реакции: 50 секунд;
  - Процент реагирования: $\left(\frac{50}{60}\right) \times 100\%$.
- Пользователь 2 (Яркие цвета, большие фигуры):
  - Фигура: ромб;
  - Цвет: зелёный;
  - Время реакции: 28 секунд.
  - Процент реагирования: $\left(\frac{28}{60}\right) \times 100\%$.

Исходя из результатов проведенного исследования было выявлено, что пользователь 1, проходящий программу с тусклыми цветами и маленькими фигурами, демонстрировал более длительное время реакции (хуже), чем пользователь 2, который взаимодействовал с программой, используя яркие цвета и большие фигуры (лучше). Пользователь 1 затратил 50 секунд на завершение 20 фигур, в то время как пользователь 2 справился за 28 секунд.

Таким образом, процент реагирования у пользователя 1 (83.33%) выше, чем у пользователя 2 (46.67%). В данном случае - более высокий процент реагирования указывает на менее эффективную реакцию.

Результат исследования представлен на диаграмме 1

Диаграмма 1 - Результат

Проведенное исследование позволило понять влияние цвета фигур, их формы на быстроту реакции при взаимодействии с программой. Результат этого исследования имеет существенное значение для разработчиков, направленных на улучшение пользовательского восприятия и эффективности взаимодействия с программой.

Глава 3. Анализ работы приложения

Структура проекта:

- main.asm – основной файл, к которому подключаем необходимые модули.

Папка lib (модули):

- Keyboard.asm – функция waitKey
- Mouse.asm – инициализация мыши, скрытие, закрытие указателя, проверка цвета кликнутого пикселя, установка курсора, получение положения курсора
- Geometry.asm – код фигур, которые будут отображаться: ромб, квадрат, треугольник, круг, точка
- Os.asm – работа с файлами, преобразование числа в строку, генератор случайных чисел, очистка экрана(текст), очистка экрана(графика), функции завершения работы программы, печать строки, печать символов
- Res.asm – включение графического режима
- Tlink.exe – линковщик (собирает main.exe из объектного файла(main.obj))
- Tasm.exe – собирает объектный файл
- TD.exe – отладчик
- All.bat – сборка, линковка, запуск программы

main.asm

Сегмент кода (Code Segment или CS) — содержит все команды и инструкции, которые должны быть выполнены. 16-битный регистр сегмента кода или регистр CS хранит начальный адрес сегмента кода.

```
75
76  _TEXT segment ; .code
77      include geometry.asm ; отрисовка геометрических фигур
78      include os.asm       ; работа с файлами, преобразование числа в строку, генератор случайных чисел
79      include res.asm      ; фключение графического режима
80      include keyboard.asm ; тут функция - нажмите любую клавишу чтобы выйти (waitKey)
81      include mouse.asm    ; мышь, проверка кликнутого пикселя
82
83
84      ; С помощью директивы ASSUME можно сообщить транслятору, какой сегмент,
85      ; к какому сегментному регистру «привязан», или, говоря боле точно,
86      ; в каком сегментном регистре хранится адрес сегмента..
87      assume cs: _TEXT, ds: _DATA, es: _DATA, ss:_STACK
88  start: ; .startup
89      mov ax, @data        ; установка в ds адреса
90      mov ds, ax           ; Для указания сегмента данных используется регистр DS
91
92
93
94
95      ; ставим временну метку в s1
96      mov ah, 2Ch ; команда для доступа к времени после неё в cl - запишется сколько сейчас минут, в dh
97      int 21h
98      mov dl, 0
99      xchg dh, dl
100
101     xor ax, ax
102     xor bx, bx
103     mov al, cl
104     mov bl, 60
105     mul bl
106     add ax, dx
107
108     mov s1, ax
```

Рисунок 2 - main.asm

Команда assume в 87 строке, сообщает транслятору, какой сегмент, к какому сегментному регистру относится.

С 96-й по 108-ю строку мы записываем в секундах относительно текущего часа, время в 2-байтовую переменную s1.

Команда call setResulutionVGA40, в 111 строке, устанавливает режим отображения dosbox, за ней идёт инициализация курсора мыши: call SetCursor.

После метка draw1 генерируются случайные координаты и цвет: случайные координаты – randomX (114–117), randomY (119–122), случайный цвет - randomColor (124-128), и номер случайной фигурки – randomFigure (130-133).

```
109
110
111         call setResulutionVGA40 ;
112         call SetCursor ; Инициализировать мышь
113     draw1:
114         push 620
115         call getRandom
116         mov ax, random
117         mov randomX, ax
118
119         push 460
120         call getRandom
121         mov ax, random
122         mov randomY, ax
123
124         push 15
125         call getRandom
126         mov ax, random
127         inc ax
128         mov randomColor, ax
129
130         push 4
131         call getRandom
132         mov ax, random
133         mov randomFigure, ax
134
135         ; mov randomColor, 2
136         ; jmp f3 ; (заменяем метку на которую хотим прыгать, например, если круг, то f3)
137         cmp randomFigure, 0
138         je f0 ; прыгнуть на метку с ромбом
139
140         cmp randomFigure, 1
141         je f1 ; прыгнуть на метку с квадратом
142
143         cmp randomFigure, 2
144         je f2 ; прыгнуть на метку с треугольником
```

Рисунок 3 - main.asm

В зависимости от числа в randomFigure мы будем прыгать на соответствующую метку. Cmp – от слова compare – сравнивать. Je – означает jump equel (прыгнуть если равно). Вместе строки cmp и je образуют условие: если randomFigure равно 0, то прыгаем на метку f0 (149 строка) и т. д.

Также ниже можно заметить слово push. Это означает поместить, что-нибудь в стек. В нашем случаем мы помещаем туда параметры для функции, эти параметры внутри функции мы сможем получить из стека.

Jmp [имя метки] – означает сделать прыжок на метку без условий

```
137     cmp randomFigure, 0
138     je f0 ; прыгнуть на метку с ромбом
139
140     cmp randomFigure, 1
141     je f1 ; прыгнуть на метку с квадратом
142
143     cmp randomFigure, 2
144     je f2 ; прыгнуть на метку с треугольником
145
146     cmp randomFigure, 3
147     je f3 ; прыгнуть на метку с кргом
148
149 f0: ; ромб
150     push randomX ; x начальная точка
151     push randomY ; y начальная точка
152     push 7  ; половина диагонали по оси x
153     push randomColor  ; цвет
154     call drawThromb ; вызвать процедуру рисования ромба
155     jmp fexit ; прыгнуть без условия на fexit
156
157 f1: ; квадрат
158     push randomColor ; цвет ; https://s7a1k3r.narod.ru/4.html
159     push randomX ; x начальная точка
160     push randomY ; y начальная точка
161     push 14      ; ширина
162     push 14      ; высота
163     call drawSquare ; вызвать процедуру рисования квадрата
164     jmp fexit ; прыгнуть без условия на fexit
165
166 f2: ; треугольник
167     push randomX       ; начальная точка x
168     push randomY       ; начальная точка y
169     push 14            ; ширина
170     push randomColor   ; цвет
171     call drawTriangle  ; вызвать процедуру рисования треугольника
```

Рисунок 4 - main.asm

Дальше идёт метка DotGame (188 строка). Итак, на данный момент фигурка отобразилась. В строках 188–192 начинается цикл, который будет выполняться до тех пор, пока не нажмём левую кнопку мыши.

Jz – означает jump if zero – прыгнуть если результат вычислений равен нулю

Пример использования: Jz [имя метки]

После завершения функции в GetMouseState (190 строка) в bx запишется некоторое значение, и, если в результате «побитового и» не получится ноль, то произойдёт выход из цикла.

Далее мы записываем координаты x, y в x_mouse, y_mouse соответственно, эти значения используются в функции checkColorPixel. Функция checkColorPixel сравнивает цвет кликнутого пикселя с randomColor и если они совпадают, то в isNew записывает 1, а если нет, то прыгаем снова на метку DotGame (188-я) строка. Если цвет кликнутого пикселя совпадает с выбранным случайно, то мы прыгаем на метку ok1 (203 строка). Тут мы увеличиваем на единичку переменную счётчик уровней – game_point, с помощью команды inc.

Inc [переменная, регистр, значение в стеке] – значит увеличить на 1.

Функция hideMouse – скрываем мышь, чтобы закраска заднего фона происходила полностью

В строке 209 мы полностью очищают видеобуфер с помощью функции clearScreen. Затем показываем мышку снова - строка 211.

```
187  ; проверяем нажали ли мы клавишу или нет и записываем координаты мыши в x_mouse, y_mouse
188  DotGame:
189      mov  bx, 0              ; Проверка на нажатие левой кнопки мыши (1 для проверки право
190      call GetMouseState
191      and  bx, 1              ; Проверка первого бита (бит 0).
192      jz  DotGame             ; пока не кликнули левой кнопкой. повтор.
193
194      mov  x_mouse, cx        ; сохраняем X и Y, потому что
195      mov  y_mouse, dx        ; CX DX будут изменены
196
197      call checkColorPixel
198
199      cmp isNew, 1
200      je ok1
201      jmp DotGame
202
203  ok1:                         ; успех мы нажали на фигурку
204      inc game_point
205      mov randomColor, 0
206
207      call hideMouse
208
209      call clearScreen
210
211      call showMouse
212
213
214      mov dl, score
215      cmp game_point, dl ; тут количество фигур которое будет отображаться
216      je stop_game ; если game_point равен dl, то прыгаем на stop_game, иначе записыаем isN
217      mov isNew, 0 ; сброс чекпоинта
218
219
220      jmp draw1 ; рисуем новую фигуру
```

Рисунок 5 - main.asm

Далее в 8-ми битный регистр dl помещаем число уровней – 20.

В строках 215–216 выполняется условие: если количество пройденных уровней равно 20, то прыгаем на метку stop_game, если не равно, то идём дальше – записываем в переменную isNew 0 это нужно, чтобы в следующий раз при клике по пустоте мы не перешли в метку ok1. После прыгаем на метку draw1, которая была в самом начале.

Таким образом происходит рисование фигур, но что, если условие в (215–216) строках выполнилось, тогда прыгаем на метку stop_game в 222 строке. Тут мы получаем текущее время с помощью команды:

Mov ah, 2Ch

Int 21h

Вход:	AH	2Ch
Выход:	CH	час
	CL	минута
	DH	секунда
	DL	сотая доля секунды

Эта функция использует системный таймер, поэтому время изменяется только 18,2 раза в секунду и число в DL увеличивается сразу на 5 или 6.

```
222  stop_game:
223      mov ah, 2Ch
224      int 21h
225      mov dl, 0
226      xchg dh, dl
227      xor ax, ax
228      xor bx, bx
229      mov al, cl
230      mov bl, 60
231      mul bl
232      add ax, dx
233      mov s2, ax
234      mov ax, s2
235      sub ax, s1
236      mov si, offset numstr1
237      call num2str
238      ; открыть существующий файл
239      push offset results ; так передаём название файла
240      call openFileRW ; открыть для чтения записи
241      ; jc error
242      mov bx, ax
243      call appendToEndFile
244      push offset numstr1
245      push 5
246      push bx
247      call writeFile
248      call closeFile
249      mov dx, offset game_end_message
250      call printString
251      call waitKey ; для задержки т.е.
252      call exit
253  _TEXT ends
254  end start
```

Рисунок 6 - main.asm

В 225 строке помещаем в 8-ми битный регистр dl, ноль

В 226 строке есть команда xchg – она меняет значения двух одинакового разрядных регистров, т. е. в dl записали секунды.

В 227-228 строках применяется команда xor(побитовое исключающие или) в данном случае она используется чтобы обнулить регистры ax и bx.

В 229 строке мы перемещаем секунды в регистр al (нужно, для умножения).

В 230 строке помещаем 60 в bl (нужно, для умножения).

В 231 строке mul bl – это переводится, как умножаем al на bl, результат такой операции запишется в регистр ax.

В 232 строке мы добавляем к секундам, прошедшим с начала последнего часа(ax), секунды, прошедшие с последней минуты(dx). После этой операции сложения ax содержит все секунды, прошедшие с начала часа.

В 233 строке записываем эти секунды в s2.

В 234 строке помещаем эти секунды снова в ax и в 235 строке вычитаем из s2 s1. Результат сохраняется в ax. Далее в 236 строке в регистр si записываем адрес строки, которая хранит в себе название строки, в которую будет записано число. В 237 строке, конвертируем число в строку, результат записывается numstr1.

В 239-й строке помещаем в стек адрес строки, содержащей название файла(results.txt). В 240-й строке вызываем функцию открытия файла для чтения и записи. В 241-й строке передаём дескриптор файла из регистра ax в регистр bx.

В 242-й строке вызываем функцию appendToEndFile для перемещения указателя в конец файла.

С 243-245-ю строку кладём параметры (адрес строки, которую хотим записать, длина строки и дескриптор файла) для функции записи в стек.

В 246-й строке записываем время прохождения теста в конец файла results.txt.

В 247-й вызываем функцию закрытия файла closeFile.

В 248-й строке мы помещаем в стек адрес сообщения Finish.

В 249-й строке печатаем это сообщение с помощью функции printString.

В 250-й строке вызываем функцию waitKey.

В 251-й строке выходим из программы с помощью функции exit.

Глава 4. Руководство для работы с программой

Как запустить программу:

1. Создать на рабочем столе папку dos;
2. Распаковать содержимое архива в папку dos (C:\Users\%username%\Desktop\dos\tasm (заменить %username% на своё));
3. Открыть dosbox option;
4. Вставить в самом конце:

KEYB RU

Mount c: C:\Users\%username%\Desktop\dos\

C:

Cd tasm

All main

5. Запуск dosbox

В данный момент, программа будет выводить фигуры разной формы и разного цвета. Для того, чтобы вывести, например, только круг синего цвета – раскомментируйте строки 130 и 131.

Отрывок листинга main.asm представлен на рисунке 7

```
134
135     ; mov randomColor, 2
136     ; jmp f3 ; (заменяем метку на которую хотим прыгать, например, если круг, то f3)
137     cmp randomFigure, 0
138     je f0 ; прыгнуть на метку с ромбом
139
140     cmp randomFigure, 1
141     je f1 ; прыгнуть на метку с квадратом
142
143     cmp randomFigure, 2
144     je f2 ; прыгнуть на метку с треугольником
145
146     cmp randomFigure, 3
147     je f3 ; прыгнуть на метку с кргом
148
149 f0: ; ромб
150     push randomX ; x начальная точка
151     push randomY ; y начальная точка
152     push 7  ; половина диагонали по оси x
153     push randomColor  ; цвет
154     call drawThromb ; вызвать процедуру рисования ромба
155     jmp fexit ; прыгнуть без условия на fexit
156
157 f1: ; квадрат
158     push randomColor ; цвет ; https://s7a1k3r.narod.ru/4.html
159     push randomX ; x начальная точка
160     push randomY ; y начальная точка
161     push 14      ; ширина
162     push 14      ; высота
163     call drawSquare ; вызвать процедуру рисования квадрата
164     jmp fexit ; прыгнуть без условия на fexit
165
166 f2: ; треугольник
167     push randomX       ; начальная точка x
168     push randomY       ; начальная точка y
```

Рисунок 7 - main.asm

Конкретная фигура:

1) В 136-й строке мы прыгаем на метку f3, jmp – значит прыгнуть без условий. После слова jmp идёт название метки, т. е. что будет отображаться (ромб, треугольник, квадрат или круг);
2) Затем перезапустить программу.

Конкретный цвет:

1. В 135 строке мы, помещаем в переменную randomColor число, которое соответствует определённому цвету (список цветов: https://s7a1k3r.narod.ru/4.html) или на следующей странице. Команду mov – сокращение от слова move перемещать двигать, затем после mov идет, то куда будем помещать значение, затем идёт значение, которое собираемся помещать в переменную;
2. Затем перезапустить программу.

Список цветов стандартной цветовой палитры EVGA представлен на рисунке 9

Код цвета	Цвет	Вид
0	Черный	
1	Синий	
2	Зеленый	
3	Бирюзовый	
4	Красный	
5	Фиолетовый	
6	Коричневый	
7	Белый	
8	Серый	
9	Голубой	
10	Салатовый	
11	Светло-бирюзовый	
12	Розовый	
13	Светло-фиолетовый	
14	Желтый	
15	Ярко-белый	

Рисунок 8 - EVGA

Всего есть 4 метки (f0, f1, f2, f3).

Где:

– f0 – ромб;

– f1 – квадрат;

– f2 – треугольник;

– f3 – круг.

Листинг фигур представлен на рисунке 10

```
145  f0: ; ромб
146      push randomX ; x
147      push randomY ; y
148      push 5  ; половина диагонали по оси y
149      push randomColor  ; color
150      call drawThromb ; вызвать процедуру рисования ромба
151      jmp fexit ; прыгнуть без условия на fexit
152
153  f1: ; квадрат
154      push randomColor ; color ; https://s7a1k3r.narod.ru/4.html
155      push randomX ; x
156      push randomY ; y
157      push 10 ; width
158      push 10; height
159      call drawSquare ; вызвать процедуру рисования квадрата
160      jmp fexit ; прыгнуть без условия на fexit
161
162  f2: ; треугольник
163      push randomX ; start point x
164      push randomY ; start point y
165      push 10  ; width
166      push randomColor   ; color
167      call drawTriangle ; вызвать процедуру рисования треугольника
168      jmp fexit ; прыгнуть без условия на fexit
169
170  f3: ; круг
171      mov radius, 5 ; Радиус нашего круга.
172      mov ax, randomX
173      mov xx0, ax    ; Номер строки, в котором будет находится центр круга
174      mov ax, randomY
175      mov yy0, ax    ; Номер столбца, в котором будет находится центр круга
176      push randomColor
177      call DrawCircle3  ; вызвать процедуру рисования круга
```

Рисунок 9 - Фигуры

Случайный режим (main.asm):

```
; mov randomColor, 2
; jmp f0 ; (заменяем метку на которую хотим прыгать, например, если круг, то f3)
```

Результаты попыток представлены на рисунках 10, 11, 12 (инвертированы цвета для печати)

Рисунок 10 - Первая попытка

Рисунок 11 - Вторая попытка

Рисунок 12 - Третья попытка

Включаем только круг пурпурного цвета (main.asm):

```
; mov randomColor, 13
; jmp f3 ; (заменяем метку на которую хотим прыгать, например, если круг, то f3)
```

Результаты попыток представлены на рисунках 13, 14, 15 (инвертированы цвета для печати)

Рисунок 13 - Первая попытка

Рисунок 14 - Вторая попытка

Рисунок 15 - Третья попытка

Включаем только зелёного цвета ромбы (main.asm):

```
; mov randomColor, 2
; jmp f0 ; (заменяем метку на которую хотим прыгать, например, если круг, то f3)
```

Результаты попыток представлены на рисунках 16, 17, 18 (инвертированы цвета для печати)

Рисунок 16 - Первая попытка

Рисунок 17 - Вторая попытка

Рисунок 18 - Третья попытка

Как вывести результаты в dosbox:

1) Набираем команду type results.txt

Результат представлен на рисунке 20 (инвертированы цвета для печати)

Рисунок 19 - Results.txt

2) Нажимаем ENTER

Результат представлен на рисунке 21 (инвертированы цвета для печати)

```
22
28
25
29
26
28
23
22
28
23
24
23
23
25
26
31
24
27
25
24
23
22
22

C:\TASM>_
```

Рисунок 20 - Результат

Для более наглядного представления особенностей был записан видеоролик работы программы:

Рисунок 21 - QR-код с работоспособностью программы

ЗАКЛЮЧЕНИЕ

В рамках курсового проекта была разработана программа на языке TurboAssembler, предназначенная для измерения времени реакции пользователя на появление простых геометрических фигур различных цветов на экране монитора. Программа успешно выполнила поставленную задачу, создавая 20 фигур случайного цвета и формы в различных местах экрана, с последующим ожиданием пользовательского ввода в виде нажатия курсором мыши.

Основные этапы алгоритма работы программы включают в себя инициализацию, генерацию и отображение фигур, ожидание пользовательского ввода, обработку результатов, сохранение времени реакции в файле, а также исследование зависимости быстроты реакции от различных параметров.

Проведенные эксперименты для исследования, включающие изменение цвета, формы фигур и времени суток, позволили выявить влияние этих параметров на быстроту реакции пользователя. Полученные результаты дают право сделать выводы о том, какие из них оказывают большее влияние на реакцию.

Проведенное исследование позволило понять влияние цвета фигур, их формы на быстроту реакции при взаимодействии с программой. Результат этого исследования имеет существенное значение для разработчиков, направленных на улучшение пользовательского восприятия и эффективности взаимодействия с программой.

В заключении хотелось бы отметить, что в проекте успешно реализованы поставленные цели и задачи, предоставлены результаты исследования зависимости времени реакции от различных параметров.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Методические указания к выполнению курсового проекта по дисциплине вычислительные машины, системы и сети.
2. Аблязов Р.З. - Программирование на ассемблере на платформе x86-64 - Издательство "ДМК Пресс" - 2011 - 304с. - ISBN: 978-5-94074-676-8 - Текст электронный
3. Зубков С. В. - Assembler. Для DOS, Windows и Unix - Издательство "ДМК Пресс" - 2008 - 640с. - ISBN: 5-94074-259-9 – Учебное пособие
4. Бунаков П.Ю. - Машинно-ориентированные языки программирования. Введение в ассемблер – Издательство «Лань» - 2023–144 с. – ISBN: 9785507454914 – Учебное пособие
5. Тюгашев А.А. - Языки программирования. – Издательство «Питер» - 2014–336 с. – ISBN: 978-5-4461-9407-0 – Учебное пособие

ПРИЛОЖЕНИЕ

main.asm

```
.model small ; модель памяти
; https://devotes.narod.ru/Books/3/ch03_03b.htm - модели памяти

; SMALL — код размещается в одном сегменте,
; а данные и стек — в другом (для их описания могут применяться разные сегменты, но объединенные в одну группу).
; Эту модель памяти также удобно использовать для создания программ на ассемблере;

; эта упрощенная директива создания сегмента стэка
.stack 200 ; указываем размер стэка

; Стандартные директивы сегментации
; Стандартно сегменты на языке Assembler описываются с помощью директивы SEGMENT.
; Синтаксическое описание сегмента представляет собой следующую конструкцию
; <имя сегмента> SEGMENT [тип выравнивания] [тип комбинирования]
;          [класс сегмента] [тип размера сегмента]
; <тело сегмента>
; <имя сегмента>  ENDS

; Стандартные директивы сегментации
_STACK segment para public 'stack'
     db 1024 dup(?) ; выделяет 200 байт для системного стека.
_STACK ends

; Стандартные директивы сегментации
_DATA segment; .data, DATASEG
     buffer db 200 DUP(0), '$'          ; буффер на 200 символов
     buffer_len equ $-buffer
     results db 'results.txt', 0

     ; круг
     eror dw ?
     xx dw ?
     yy dw ?
     xx0 dw ?
     yy0 dw ?
     delta dw ?
     radius dw ?

     ; мышь
     XCords1 dw 160         ; где будет находится
     YCords1 dw 100         ; курсор при запуске
```

```
          x_mouse dw ?           ; сюда запишем координату X клика
мыши. ; ? - означает что переменная не инициализорованна
          y_mouse dw ?           ; сюда запишем координату Y клика
мыши. ; dw - размер переменной 2 байта

          numstr1 db 3 DUP (' '), 0Dh, 0Ah  ; строка состоящая из 3
пробелов и символов перевода строки

          random dw ?
          randomX dw ?
          randomY dw ?
          randomColor dw ?
          randomFigure dw ?

          isNew db ?

          game_point db 0 ; счётчик пройденных уровней
          game_end_message db "Finish $"
          score db 20 ; максимальное число попыток

          s1 dw 0 ; певая запись времени
          s2 dw 0 ; вторая запись времени

          x1 dw 0 ; эти переменные используются для закрашивания
окружности ; первая точка
          y1 dw 0

          x2 dw 0 ; для закрашивания
          y2 dw 0

     _DATA ends

     ; Стандартные директивы сегментации
     _TEXT segment ; .code - это упрощенная директива сегментации
          include geometry.asm ; отрисовка геометрических фигур
          include os.asm        ; работа с файлами, преобразование
числа в строку, генератор случайных чисел
          include res.asm      ; фключение графического режима
          include keyboard.asm ; тут функция - нажмите любую клавишу
чтобы выйти (waitKey)
          include mouse.asm    ; мышь, проверка кликнутого пикселя
          ; С помощью директивы ASSUME можно сообщить транслятору,
какой сегмент,
          ; к какому сегментному регистру «привязан», или, говоря
боле точно,
          ; в каком сегментном регистре хранится адрес сегмента.
          assume cs:_TEXT,ds:_DATA,es:_DATA,ss:_STACK
     start: ; .startup
          mov ax, @data        ; установка в ds адреса
```

```
          mov ds, ax                ; Для указания сегмента данных
используется регистр DS

          ; ставим временную метку в s1
         mov ah, 2Ch ; команда для доступа к времени после неё в cl
- запишется сколько сейчас минут, в dh - сколько сейчас секунд
         int 21h
         mov dl, 0
         xchg dh, dl

         xor ax, ax
         xor bx, bx
         mov al, cl
         mov bl, 60
         mul bl
         add ax, dx

         mov s1, ax

          call setResulutionVGA40 ;
          call SetCursor ; Инициализировать мышь
     draw1:
          push 620
          call getRandom
          mov ax, random
          mov randomX, ax

          push 460
          call getRandom
          mov ax, random
          mov randomY, ax

          push 15
          call getRandom
          mov ax, random
          inc ax
          mov randomColor, ax

          push 4
          call getRandom
          mov ax, random
          mov randomFigure, ax

          ; mov randomColor, 2
          ; jmp f0 ; (заменяем метку на которую хотим прыгать,
например, если круг, то f3)
          cmp randomFigure, 0
          je f0 ; прыгнуть на метку с ромбом
```

```
        cmp randomFigure, 1
        je f1 ; прыгнуть на метку с квадратом

        cmp randomFigure, 2
        je f2 ; прыгнуть на метку с треугольником

        cmp randomFigure, 3
        je f3 ; прыгнуть на метку с кргом

    f0: ; ромб
        push randomX ; x начальная точка
        push randomY ; y начальная точка
        push 7  ; половина диагонали по оси x
        push randomColor  ; цвет
        call drawThromb ; вызвать процедуру рисования ромба
        jmp fexit ; прыгнуть без условия на fexit

    f1: ; квадрат
        push randomColor ; цвет ; https://s7a1k3r.narod.ru/4.html
        push randomX ; x начальная точка
        push randomY ; y начальная точка
        push 14      ; ширина
        push 14      ; высота
        call drawSquare ; вызвать процедуру рисования квадрата
        jmp fexit ; прыгнуть без условия на fexit

    f2: ; треугольник
        push randomX       ; начальная точка x
        push randomY       ; начальная точка y
        push 14            ; ширина
        push randomColor   ; цвет
        call  drawTriangle     ;  вызвать  процедуру  рисования
треугольника
        jmp fexit          ; прыгнуть без условия на fexit

    f3: ; круг
        mov radius, 7     ; Радиус нашего круга.
        mov ax, randomX
        mov xx0, ax       ; Номер строки, в котором будет находится
центр круга
        mov ax, randomY
        mov yy0, ax      ; Номер столбца, в котором будет находится
центр круга
        push randomColor  ; цвет
        call DrawCircle3  ; вызвать процедуру рисования круга

    fexit:
```

```
     ; проверяем нажали ли мы клавишу или нет и записываем координаты
мыши в x_mouse, y_mouse
     DotGame:
          mov  bx, 0              ; Проверка на нажатие левой кнопки
мыши (1 для проверки правой кноп).
          call GetMouseState
          and  bx, 1              ; Проверка первого бита (бит 0).
          jz  DotGame             ; пока не кликнули левой кнопкой.
повтор.

          mov  x_mouse, cx        ; сохраняем X и Y, потому что
          mov  y_mouse, dx        ; CX DX будут изменены

          call checkColorPixel

          cmp isNew, 1
          je ok1
          jmp DotGame

     ok1:                         ; успех мы нажали на фигурку
          inc game_point
          mov randomColor, 0

          call hideMouse ; скрыть мышь

          call clearScreen ; очистить экран

          call showMouse ; показать мышь

          mov dl, score
          cmp game_point, dl ; тут количество фигур которое будет
отображаться
          je stop_game ; если game_point равен dl, то прыгаем на
stop_game, иначе записыаем isNew 0 и отрисовываем новую картинку
          mov isNew, 0 ; сброс чекпоинта

          jmp draw1 ; рисуем новую фигуру

     stop_game:
         mov ah, 2Ch
         int 21h
         mov dl, 0
         xchg dh, dl
         xor ax, ax
         xor bx, bx
         mov al, cl
         mov bl, 60
         mul bl
         add ax, dx
```

```
    mov s2, ax
    mov ax, s2
    sub ax, s1
    mov si, offset numstr1
    call num2str
     ; открыть существующий файл
     push offset results ; так передаём название файла
     call openFileRW ; открыть для чтения записи
     mov bx, ax
     call appendToEndFile
     push offset numstr1
     push 5
     push bx
     call writeFile
     call closeFile
     mov dx, offset game_end_message
     call printString
     call waitKey ; для задержки т.е.
     call exit

_TEXT ends
end start
```

geometry.asm

```
drawSquare proc ; квадрат
     push bp
     mov bp, sp

     mov cx, [bp+10]  ; получаем из буфера точку x, записываем
в cx
     mov dx, [bp+8]   ; получаем из буфера точку y, записываем
в dx
     mov al, [bp+12]  ; в al запишем цвет который отправили
     mov ah, 0ch      ; функция рисования пикселя

     mov si, [bp+6]  ; в si запишем ширину
     add si, [bp+10] ; добавим к ширине точку старта: width +
xstart
     mov di, [bp+4]  ; в si запишем высоту
     add di, [bp+8]      ;  добавим  к  высоте  точку  старта:
height + ystart

     colcount:
     inc cx ; увеличиваем на 1 точку старта

     int 10h ; нарисовать
     cmp cx, si    ;  пока точка старта не станет равна точке
конца
     JNE colcount ; пока не равно выполняется

     mov cx, [bp+10]  ; сбросить cx в началоо т.е. x = точке
старта x
     inc dx          ; увеличиваем y ; плоскость в компьютерах
идет: слева на прова, и сверху в низ
     cmp dx, di   ; сравниваем y и вторую точку, которая конец
     JNE colcount ; если не равно то продолжаем цикл

     mov sp, bp
     pop bp
     ret 10 ; размер каждого параметра в ms-dos равен 2 байтам:
16 битам
     ; в эту функцию передали 5 параметров, значит всего нужно
вернуть 10 байт
drawSquare endp

drawTriangle proc ; треугольник
     push bp
     mov bp, sp

     mov cx, [bp+10]    ; x начало
     mov di, cx         ; сохраняем значение x начала в di
     mov si, cx         ; сохраняем значение x начала в si
```

```
          add si, [bp+6]      ; x точка старта + ширина = конечная
точка по оси x; тут мы записали число до которого будем рисовать
пиксели по оси x
          mov dx, [bp+8]      ; y точка начала
          mov al, [bp+4]      ; в al запишем цвет
          mov ah, 0ch         ; функция нарисовать пиксель
          mov bx, 0           ; в bx записали ноль
     looph:
          int 10h   ; нарисовать пиксель
          inc cx  ; увеличить координату x
          cmp cx, si ; пока x меньше конечной точки
          JB looph

          push ax     ; кладём в стэк значение регистра ax ; потому
что этот регистр хранит функцию которая рисует пиксель и цвет пикселя
          mov ax, dx ; запишем в ax координату y
          mov bl, 2  ; запишем в bl 2
          div bl     ; ah содержит остаток от деления ax на bl
          ; TEST dx, 4
          cmp ah, 0
          JZ evn ; если остаток от деления 0
          JNZ odd ; не ноль
     evn:
          add di, 1  ; увеличиваем на 1 di который хранит самую левую
точку равностороннего треугольник x
          dec si     ; уменьшаем si на 1 который хранит самую правую
точку равностороннего треугольник x
     odd:
          dec dx     ; уменьшаем y - поднимаемся вверх
          mov cx, di ; изменяем стартовую точку x

          pop ax ; достать последнее записанное в стэк число и
записать его в ax

          cmp si, cx ; если стартовая точка равна конечной точке,
или мы дошли до вершины треугольника то выходим
          JE quit

          jmp looph ; иначе начинаем цикл заного


     quit:
          mov sp, bp
          pop bp
          ret 8
     drawTriangle endp

     drawThromb proc ; робм
          push bp
```

```
     mov bp, sp

     mov cx, [bp+10] ; координата центра x
     mov dx, [bp+8]  ; координата центра y
     mov si, 0
     mov di, [bp+6]  ; половина диаметра по оси x

     push 0
     push di
     push si

     mov al, [bp+4]  ; цвет помещам в регистр al
     mov ah, 0ch     ; нарисовать  пиксель


@@right_and_left:
     push cx
     sub cx, si
     int 10h
     pop cx

     push cx
     add cx, si
     int 10h
     pop cx

     inc si
     cmp si, di
     jne @@right_and_left


     cmp [bp-2], 0
     je @@decrease
     jne @@increase

@@decrease:
     dec dx
     jmp @@next_step
@@increase:
     inc dx
     jmp @@next_step

@@next_step:
     push ax
     push bx
     mov ax, dx
     mov bl, 2
     div bl
     cmp ah, 0
```

```
     jz @@evn
     jnz @@odd
@@evn:
     dec di
@@odd:
     mov si, [bp+14]
     mov [bp+14], si
     pop bx
     pop ax

     cmp si, di
     je @@quit

     jmp @@right_and_left

@@quit:
     cmp [bp-2], 1
     jne @@two
     je @@stop
@@two:
     mov [bp-2], 1
     mov si, [bp-6]
     mov di, [bp-4]
     mov dx, [bp+8]
     jmp @@right_and_left

@@stop:
     mov sp, bp
     pop bp
     RET 10
drawThromb endp

Plot proc ; точка
     push bp
     mov bp, sp

     mov Ah, 0Ch         ; Функция отрисовки точки
     mov al, [bp+4]  ; Цвет
     int 10h             ; Нарисовать точку

     mov sp, bp
     pop bp
     ret 2
Plot endp

drawCircle3 proc ; круг
     push bp
     mov bp, sp
```

```
        mov xx, 0 ; в xx поместить 0
         mov ax, radius ; в ax поместить radius
        mov yy, ax ; в yy поместить ax
        mov delta, 2 ; в delta поместить 2
         mov ax, 2 ;    в ax поместить 2
         mov dx, 0 ; в dx поместить 0
         imul yy ; dx:ax = ax * yy ; imul умножение со знаком
         sub delta, ax ; delta = delta - ax
         mov eror, 0 ; в error помещаем 0
         jmp ccicle ; прыгаем без условий на метку ccicle

    finally:
         ; завершение программы
         mov sp, bp
         pop bp
         ret 2
    ccicle:
         mov ax, yy ; в ax помещаем yy
         cmp ax, 0 ; сравниваем ax с нулём
         jl   finally ; если ax меньше 0 то прыгаем на метку
    finally

         ; первая точка сверху справа окружности
        mov cx, xx0 ; поместить в cx, xx0
         add cx, xx  ; прибавить к cx xx; cx = cx + xx
         mov x1, cx  ; поместить в x1, cx
         mov dx, yy0 ; поместить в dx, yy0
         sub dx, yy  ; вычесть из dx, yy
         mov y1, dx  ; поместить в y1 dx
         push  [bp+4]  ; поместить в стек цвет для отрисовки -
параметр функции
         call Plot   ; вызвать отрисовку пикселей

         ; первая точка снизу справа окружности
         mov cx, xx0 ; поместить в xx0
         add cx, xx  ; добавить к cx, xx
         mov x2, cx  ; поместить в x2, cx
         mov dx, yy0 ; поместить в dx, yy0
         add dx, yy  ; прибавить к dx, yy
         mov y2, dx  ; поместить в y2, dx
         push  [bp+4]  ; поместить в стек цвет для отрисовки -
параметр функции
         call Plot   ; вызвать отрисовку пикселей

    qright1:          ; тут происходит заполнение окружности цветом
между точкми x1, y1 и x2, y2 т.е.заполняется правая половина от
центра вправо
         inc y1
         mov cx, x1
```

```
        mov dx, y1
        push [bp+4]
        call Plot

        cmp dx, y2
        jae qright2    ; если dx больше или равен y2 то завершаем
цикл и прыгаем на qright2
        jmp qright1
    qright2: ; то есть когда мы встаём на эту метку мы заполнили
только одну линию вертикальную

        ; первая точка сверху справа окружности
        mov cx, xx0
        sub cx, xx
        mov x1, cx
        mov dx, yy0
        sub dx, yy
        mov y1, dx
        push [bp+4]
        call Plot

        ; первая точка снизу справа окружности
        mov cx, xx0
        sub cx, xx
        mov x2, cx
        mov dx, yy0
        add dx, yy
        mov y2, dx
        push [bp+4]
        call Plot

    qleft1: ; в этом цикле мы заполняем растояние между этими двумя
точками но только от центра влево
        inc y1
        mov cx, x1
        mov dx, y1
        push [bp+4]
        call Plot

        cmp dx, y2
        jae qleft2
        jmp qleft1

    qleft2: ; то есть когда мы встаём на эту метку мы заполнили
только одну линию вертикальную

        mov ax, delta ; помещаем в ax delta
        mov eror, ax  ; помещаем в eror ax
        mov ax, yy    ; помещаем в ax yy
```

```
          add eror, ax  ; прибавляем к eror ax
          mov ax, eror  ; в ax помещаем eror
          mov dx, 0     ; в dx помещаем 0
          mov bx, 2     ; в bx помещаем 2
          imul bx       ; знаковое умножение ax на bx
          sub ax, 1     ; уменьшаем ax на 1
          mov eror, ax  ; помещаем в eror ax
          cmp delta, 0  ; сравниваем delta и 0
          jg sstep       ; если delta больше 0 прыгаем на sstep в
отличии от ja может работать с отрицательными числами
          je sstep      ; если delta равен 0 прыгаем на sstep
          cmp eror, 0   ; сравниваем eror, 0
          jg  sstep     ; если eror больше 0 то прыгаем на sstep
          inc xx        ; увеличиваем xx на 1
          mov ax, 2     ; помещаем в ax 2
          mov dx, 0     ; помещаем в dx 0
          imul xx       ; знаковое умножение. умножаем ax на xx
          add ax, 1     ; в dx:ax будет результат ax*xx
          add delta, ax ; к delta прибавляем ax
         jmp ccicle    ; прыгаем в самое начало возле метки finally
     sstep:
          mov ax, delta ; поместиь в ax delta
          sub ax, xx    ; вычесть из ax xx
          mov bx, 2     ; поместиь в bx 2
          mov dx, 0     ; поместиь в dx 0
          imul bx         ; знаковое умножение. ax * bx результат
запишется в dx:ax
          sub ax, 1     ; вычесть из ax 1
          mov eror, ax  ; поместиь в eror ax
          cmp delta, 0  ; сравниваем delta и 0
          jg tstep       ; если delta больше 0 то прыгаем на tstep.
JG может работать с отрицательными числами
          cmp eror, 0   ; сравниваем eror, 0
          jg tstep      ; если eror больше 0 то прыгаем на tstep. JG
может работать с отрицательными числами
          inc xx        ; увеличиваем xx на 1
          mov ax, xx    ; поместить в ax xx
          sub ax, yy    ; вычесть из ax yy
          mov bx, 2     ; поместить в bx 2
          mov dx, 0     ; поместить в dx 0
          imul bx       ; dx:ax = ax*bx ;знаковое умножение
          add delta, ax ; прибавить к delta ax
         dec yy        ; уменьшить на 1 yy
          jmp ccicle     ; прыгнуть на метку с самого начала после
метки finally
     tstep:
          dec yy        ; уменьшить yy
         mov ax, 2     ; поместить в ax 2
          mov dx, 0     ; поместить в dx 0
```

```
          imul yy          ; умножить ax на yy результат в dx:ax ;
знаковое умножение
          mov bx, 1      ; поместить в bx 1
          sub bx, ax     ; вычесть из bx ax
          add delta, bx ; добавить к delta bx
          jmp ccicle      ; прыгаем в самое начало на метку ccicle
после метки finally
     drawCircle3 endp
```

keyboard.asm

```
waitKey proc
     mov ah, 0
     int 16h                    ; ждать нажатия клавиши
     ; mov ax, 3
     ; int 10h                  ; режим 3
     ; mov ah, 4ch
     ; int 21h                 ; Завершить программу после нажатия
любой клавиши

     ret                        ; Вернуть управление
waitKey endp
```

mouse.asm

```
;---------------------------------------
SetCursor proc
     ; Инициализация мыши
     mov ax, 0h
     int 33h
     ; Показать мышь
     mov ax, 1h
     int 33h
     ; получить положением мыши и статус
     ; mov ax, 3h
     ; int 33h
     ret
SetCursor endp


;---------------------------------------
; Полуить положение курсора
; возврат : BX : бит 0 = 0 : нажата левая кнопка мыши.
;                      = 1 : отпущена левая кнопка мыши.
;           :   бит 1 = 0 : нажата правая кнопка мыши..
;                      = 1 : отпущена правая кнопка мыши.
;         CX = x.
;         DX = y.
GetMouseState proc
     mov  ax, 3         ;Эта функция позволяет определить где
пользователь кликнул мышкой. Определить состояние мыши
     int  33h
     ret
GetMouseState endp

checkColorPixel proc
     push bp
     mov bp, sp

     mov ah, 0Dh ; функция получения цвета по координатам
     mov cx, [x_mouse]
     mov dx, [y_mouse]
     int 10H ; AL = цвет
     cmp al, byte ptr randomColor ; такой приём позволяет
сравнить 8бит регистр с 16 битной переменной, т.е. мы как бы
преобразуем randomColor в 8 бит
     je noblack
     jmp next
noblack:
     mov isNew, 1
     jmp next
next:
     mov sp, bp
```

```
     pop bp
     ret

checkColorPixel endp

hideMouse proc
     mov AX, 2
     INT 33h                    ; скрываем мышку
     ret
hideMouse endp

showMouse proc
     mov AX, 1
     INT 33h                    ; показываем мышку
     ret
showMouse endp
```

os.asm

```
; создание процедур; занимает меньше времени и больше памяти
; в регистр dx должна быть помещена строка, примерно так: mov
dx, offset [название строки]
printString PROC ; напечатать строку
    MOV ah, 09h
    int 21h

    ; перевод на новую строку
     MOV dl, 10
     MOV ah, 02h
     INT 21h
     MOV dl, 13
     MOV ah, 02h
     INT 21h
    RET
printString ENDP

printSymbol proc ; напечатать символ
     mov ah, 02h
     int 21h
    ; перевод на новую строку
     MOV dl, 10
     MOV ah, 02h
     INT 21h
     MOV dl, 13
     MOV ah, 02h
     INT 21h
     RET
printSymbol endp

createFile proc ; создание файла
     push bp
     mov bp, sp

    mov ah, 3Ch
    mov al, 0           ; если файл не создаётся: нужно перед
вызовом сделать mov cx, 0
    mov dx, [bp+4]    ; название файла
    int 21h

    mov sp, bp
    pop bp

    RET 2
createFile endp

; открыть для чтения
```

```
    openFileR proc    ; дескриптор файла при открытии вернется в
регистр AX
          push bp
          mov bp, sp

          mov ah, 3Dh
          mov al, 0     ; чтение
          mov dx, [bp+4]

          int 21h

          mov sp, bp
          pop bp
          RET 2        ; вернуть управление в точку запуска
    openFileR endp

    ; открыть для чтения записи
    openFileRW proc    ; дескриптор файла при открытии вернется в
регистр AX
          push bp
          mov bp, sp

          mov ah, 3Dh
          mov al, 2      ; чтение и запись
          mov dx, [bp+4]

          int 21h

          mov sp, bp
          pop bp
          RET 2        ; вернуть управление в точку запуска
    openFileRW endp

    ; прочитать файл
    readFile proc    ; Код ошибки если CF установлен к CY; если
ошибок не было то в AX будет количество прочитанных байт
          push bp
          mov bp, sp
          mov buffer[buffer_len-1], '$'
          mov ah, 3Fh
          mov bx, [bp+4]
          xor cx, cx
          mov cx, buffer_len
          lea dx, buffer
          int 21h
          call printString
          call clearBuffer
          mov sp, bp
          pop bp
```

```
          RET  2
     readFile endp
     ; записать в файл
     writeFile proc
          push bp       ; Кладём bp в стэк. Для сохранения указателя
на стек используется регистр bp,
          mov bp, sp     ; bp записываем указатель на стек

          mov ah, 40h    ; команда записи в файл
          mov bx, [bp+4] ; дескриптор файла тоже из стэка, куда
записывать данные из буфера
          mov cx, [bp+6] ; берем из стэка количество символов,
которые нужно заполнить
          mov dx, [bp+8] ; то что будем записывать в файл

          int 21h        ; записать

          mov sp, bp
          pop bp
          RET 6
     writeFile endp

     ; добавить в конец файла
     appendToEndFile proc
          ; bx должен содержать в себе дескриптор файла
          mov ah, 42h  ; "lseek"
          mov al, 2    ; позиция относительно конца файла
          ; 0 = смещение относительно начала файла
          ; 1 = смещение относительно текущей позиции файла (cx:dx
назначен)
          ; 2 = смещение относительно конца файла (cx:dx назначен)
          mov cx, 0    ; offset MSW
          mov dx, 0    ; offset LSW
          int 21h
          ret
     appendToEndFile endp

     ; добавить в начала файла
     appendToStartFile proc
          ; Начинает с начала файла перезаписывая всё на своём пути
          mov ah, 42h   ; "lseek" Для изменения текущей позиции
чтения-записи используется системный вызов lseek().
          mov al, 0    ; смещение относительно начала файла
          ; 0 = смещение относительно начала файла
          ; 1 = смещение относительно текущей позиции файла (cx:dx
назначен)
          ; 2 = смещение относительно конца файла (cx:dx назначен)
          mov cx, 0    ; смещение относительно верхнего слова
          mov dx, 0    ; смещение относительно нижнего слова
```

```
          int 21h
          ret
     appendToStartFile endp


     ; закрыть файл
     ; в BX дескриптор файла, т.е то что было при открытии нужно
вернуть в bx -> mov bx, ax
     closeFile proc
          mov ah, 3Eh ; функция закрытия файла
          int 21h
          RET
     closeFile endp

     ; сгенерировать случайное число
     getRandom proc
         push bp
         mov bp, sp
         push ax
         push bx
         push cx
         push dx


         xor ax, ax ; обнуление ax
         xor ah, ah ; обнуление ah
         mov es, ax
         mov ax, es:[46Ch] ; системный таймер

         cmp [bp+4], 4
         je get_remainder

         cmp [bp+4], 15 ; условие для псевдорандомности - запутывание
чисел таймера
         je get_remainder
         jne confuse
     confuse:
         ; rol ax, 2
         ; ror ax, 2
         mul ax
         mov al, ah
         mov ah, dl

     get_remainder:
         xor dx, dx
         mov bx, [bp+4]
         div bx

         cmp [bp+4], 4
```

```
        je get_end

        cmp [bp+4], 15
        je get_end
        jne dx_below_30
    dx_below_30:
        cmp dx, 30
        jbe below
        jnbe get_end

    below:
        add dx, 30

    get_end:
        mov random, dx

        pop dx
        pop cx
        pop bx
        pop ax
        mov sp, bp
        pop bp
        ret 2
    getRandom endp
    ;-----------------------------------------
    ; Конвертируем число в строку
    ; Алгоритм: разбить число на цифры, сохранить из в стэк, затем перевернуть их
    ; чтобы записать в одну строку.
    ; параметры: AX = число которое конвертируем.
    ;                 SI = указатель на строку, в которую запишем конвертируемое число, т.е. в эту строку мы запишем символы бывшего числа.
    ; регистры которые изменятся: AX, BX, CX, DX, SI.

    num2str proc ; то число которое мы будем преобразовывать в строку уже записано в ax тут мы будем делить ax на bx
         mov  bx, 10  ; чтобы преобразовать число в строку мы должны разобрать его по числам, для это го будем число делить на 10
         mov  cx, 0   ; счётчик чисел в числе пример: 635 6,3,5 = 3 цифры в числе 635
    _cycle1:
         mov  dx, 0   ; так как результат запишется по адресу DX:AX то мы обнуляем dx
         div  bx      ; DX:AX / 10 = AX:целая часть DX:остаток.
         ; В остатке как раз и будет одна цифра из числа 635: 635%10=5 635/10=63 -> в dx=5, ax=63;
         push dx     ; сохраняем цифру 5 в стэк чтобы потом записать его в строку.
```

```
          inc  cx     ; увеличиваем счётчик в cx. Этот счётчик нужен
именно для цикла loop.
          cmp  ax, 0   ; если регист ax
          jne  _cycle1  ; не равен нули цикл продолжается

     ; здесь мы берем цифры из стэка которые записали выше
     _cycle2:  ; это цикл работает пока cx не станет равен нулю. Сам
цикл loop подразумевает что cx с каждой итерацией уменьшается
          pop  dx ; достайм из стэка последнюю цифру которую туда
записывали
          add  dl, 48   ; это такой способ конвертировать цифру в
символ: 2 -> '2'
          mov  [ si ], dl ; как вы помните перед вызовом этой функции
в si мы поместили адрес первого символа строки.
          ; Теперь на этот адрес помещаем символ
          inc  si ; смещаемя на второй адрес строки
          loop _cycle2  ; пока cx не станет нулю

          ret ; возвращаем управление в точку вызова
     num2str endp

     exit proc
          mov ah, 04ch ; функция DOS выхода из программы
          mov al, 0h     ; код возврата
          int 21h ; Вызов DOS остановка программы
     exit endp

     clearBuffer proc ; очистка экрана в текстовом режиме
          push ax
          push bx
          push cx
          push dx

          mov cx, buffer_len
          mov bx, offset buffer
     l1:
          mov al, [bx]
          mov al, 0
          mov [bx], al
          inc bx
          loop l1

          pop dx
          pop cx
          pop bx
          pop ax
          ret
     clearBuffer endp
```

```
clearScreen proc ; очистка экрана в графическом режиме
    mov ax, 0B800h
    mov ax, 0A000h
    mov es, ax
    xor di, di              ; ES:0  это начало фрэймбуфера
    xor ax, ax
    mov cx, 32000d
    cld
    rep stosw
    ret
clearScreen endp
```

res.asm

```
; 4-битные режимы (16 цветов):
; VGA
; установить разрешение экрана в 640 на 480
setResulutionVGA40 proc
     ; 012h: 640x480 (64 Кб)
     mov ah, 4fh
     mov al, 02h
     mov bh, 0h
     mov bl, 12h
     int 10h

     ret
setResulutionVGA40 endp
```