Лабораторное занятие 4. Получить ***ВТОРУЮ*** ПРОГРАММУ (изначально используя язык записи алгоритмов ассемблер) ДЛЯ 64-РАЗРЯДНОЙ ЭВМ, которая **может заставить ЭВМ (БЕЗ ИСПОЛЬЗОВАНИЯ ОБОЛОЧКИ DOSBOX) вывести на экран знакомое Вам стандартное окно: MessageBox, имеющее все стандартные «пимпочки», а в качестве сообщения в него выводится текст: «Hello, world», а после нажатия клавиши ввода это сообщение изменяется на «Hello, hell!» И ЭТОТ MessageBox НАЧИНАЕТ ХАОТИЧНО ДВИГАТЬСЯ (НЕ СКАЧЕТ, а хаотично двигается), причём MessageBox НЕ ДОЛЖЕН выдвигаться никаким боком за границу экрана, И он ДОЛЖЕН БЫТЬ НА ЭКРАНЕ ВСЕГДА ОДИН – без клонов в других местах экрана).**

**В качестве «свиньи» привожу текст на языке записи алгоритмов ассемблер, который после трансляции, линковки, размещения исполнимого модуля в оперативной памяти, получается программа - текст на машинном языке, который ДЕКОДИРУЕТСЯ САМОЙ ЭВМ и заставляет эту ЭВМ выполнить определенные действия – заставляет ЭВМ отображать на экране messageBox и двигать этот боксик, НО ПО СИНУСОИДЕ (а не хаотично). Ваша задача сделать ХАОС без «выезда» за границу!!! То есть найти места, где задаются скорость движения и направление.**

**Замечание. Выражение «подложить свинью» означает следующее (ИИ от google).** "Подложить свинью" означает совершить коварный, подлый поступок или намеренно причинить кому-либо неприятность, навредить, обмануть или поставить в неловкое положение. Этот фразеологизм описывает действие, направленное на то, чтобы доставить другому человеку хлопоты или нанести ущерб, будь то намеренно или случайно.

Приведение мною текста на языке записи алгоритмов ассемблер в рамках того, что называется «программированием» или «БИС», как раз и представляет собой «подкладывание свиньи», так как этим я доставляю Вам ущерб, совершаю коварный поступок, причиняю Вам неприятность, ставлю Вас в неловкое положение. Ущерб заключается в том, что не Вы сами нашли решение задачи по движению боксика, а его (решение) Вам «подложили». И в следующий раз, когда придётся решать какую-либо другую задачу, которую ещё не решили на помойке или GigaChatом, Чатом ДжиПиТи и Глубоким поиском, Вы будете не способны самостоятельно найти решение. Существует, конечно, метод обучения, основанный на «аналогиях» (типа «копирования» или «делай как я»), спроси Гугела, но это не наш метод. Наш метод ориентирован на то, чтобы «научить учиться» вне зависимости от предмета рассмотрения.

**Внимательно читаем постановку задачи, чтобы не «попасть»!!! Так как обманки никуда не делись. Они с Вами!**

**Присылать мне на почту ничего не надо – очень даже может быть, что я снова, как и в случае с «папой», не буду ничего проверять, но, вдруг, буду. Заодно вспоминайте папу, так как следующее лабораторное занятие будет «посвящено» деду, папе и отладчикам…. Отладчики весьма востребованы специалистами в области БИС.**

**extrn MessageBoxA: PROC**

**extrn ExitProcess:PROC**

**extrn CreateThread:PROC**

**extrn FindWindowA:PROC**

**extrn MoveWindow:PROC**

**extrn Sleep:PROC**

**.data**

**caption db '64-bit hello', 0**

**message db 'Hello, World!', 0**

**HWINDOW dword ?**

**SLEEP\_DURATION dword 10**

**CURRENT\_X dword 450**

**CURRENT\_Y dword 350**

**new\_x dword 0**

**new\_y dword 0**

**MSGBOX\_WIDTH dword 133**

**MSGBOX\_HEIGHT dword 140**

**MAX\_X dword 1250**

**MAX\_Y dword 600**

**SPEED\_Y dword 1**

**SPEED\_X dword 1**

**TWENTY dword 500**

**const\_X dword 1000**

**const\_Y dword 600**

**flag\_x dword 0**

**flag\_y dword 0**

**const\_sleep dword 5**

**.code**

**Start proc**

**sub rsp,28h**

**mov rcx, 0**

**lea rdx, message**

**lea r8, caption**

**mov r9d, 0**

**call MessageBoxA**

**add RSP,28h**

**;CreateThread(0,0,Thread\_Func, 0, 0, 0)**

**mov RCX, 0**

**mov RDX, 0**

**mov R8, Move**

**mov R9, 0**

**sub RSP, 12**

**mov qword ptr [RSP], 0**

**mov dword ptr [RSP + 8], 0**

**call CreateThread**

**add RSP, 12**

**sub rsp,28h**

**mov message+7,'h'**

**mov message+8, 'e'**

**mov message+9,'l'**

**mov message+10,'l'**

**mov message+11,'!'**

**mov caption+11,'!'**

**mov rcx, 0**

**lea rdx, message**

**lea r8, caption**

**mov r9d, 0**

**call MessageBoxA**

**add rsp,28h**

**Start endp**

**Move proc**

**;Find\_WIndow(0,caption) return handler in EAX**

**FIND:**

**mov RCX, 0**

**lea RDX, [caption]**

**call FindWindowA**

**cmp RAX, 0**

**je FIND**

**mov [HWINDOW], EAX**

**new\_coord:**

**mov flag\_x,0**

**mov flag\_y,0**

**mov edx, 0**

**rdrand AX**

**div const\_X**

**mov eax, edx**

**mov new\_x , eax**

**mov SPEED\_X, 1**

**mov edx, 0**

**rdrand AX**

**div const\_Y**

**mov eax, edx**

**mov new\_y, eax**

**mov SPEED\_Y, 1**

**mov edx, 0**

**rdrand AX**

**div const\_sleep**

**mov eax, edx**

**add eax, 1**

**mov SLEEP\_DURATION, eax**

**LP\_MOVE:**

**;Sleep(SLEEP\_DURATION)**

**mov ECX, [SLEEP\_DURATION]**

**call Sleep**

**mov EAX, SPEED\_X**

**add CURRENT\_X, eax**

**mov eax,flag\_x**

**cmp eax,1**

**je continue**

**mov EAX, CURRENT\_X**

**cmp eax, new\_x**

**je flag\_x\_cmp**

**cmp eax, new\_x**

**jg CHANGE\_X\_g**

**cmp eax, new\_x**

**jl CHANGE\_X\_l**

**continue:**

**mov eax,flag\_y**

**cmp eax,1**

**je MOOVE\_WINDOW**

**mov EAX, SPEED\_Y**

**add CURRENT\_Y, EAX**

**mov EAX, CURRENT\_Y**

**cmp EAX, new\_y**

**je flag\_y\_cmp**

**cmp EAX, new\_y**

**jg CHANGE\_Y\_g**

**cmp EAX, new\_y**

**jl CHANGE\_Y\_l**

**jmp MOOVE\_WINDOW**

**CHANGE\_X\_g:**

**cmp SPEED\_X,0**

**jg neg\_speed\_x**

**jmp continue**

**CHANGE\_Y\_g:**

**cmp SPEED\_Y,0**

**jg neg\_speed\_y**

**jmp MOOVE\_WINDOW**

**CHANGE\_X\_l:**

**cmp SPEED\_X,0**

**jl neg\_speed\_x**

**jmp continue**

**CHANGE\_Y\_l:**

**cmp SPEED\_Y,0**

**jl neg\_speed\_y**

**jmp MOOVE\_WINDOW**

**MOOVE\_WINDOW:**

**;MoveWindow(HWINDOW,Current\_x,Current\_Y,MSGBOX\_Width, MSGBOX\_Height, 1)**

**mov ECX, [HWINDOW]**

**mov EDX, CURRENT\_X**

**mov R8D, CURRENT\_Y**

**mov R9D, [MSGBOX\_WIDTH]**

**mov EAX, [MSGBOX\_HEIGHT]**

**mov dword ptr [RSP+20h], EAX**

**mov dword ptr [RSP+28h], 1**

**call MoveWindow**

**cmp EAX, 0**

**jne LP\_MOVE**

**mov RCX, 0**

**jmp exit\_p**

**neg\_speed\_x:**

**neg SPEED\_X**

**jmp MOOVE\_WINDOW**

**neg\_speed\_y:**

**neg SPEED\_Y**

**jmp MOOVE\_WINDOW**

**flag\_x\_cmp:**

**mov eax,flag\_y**

**cmp eax,1**

**je new\_coord**

**mov flag\_x,1**

**jmp LP\_MOVE**

**flag\_y\_cmp:**

**mov eax,flag\_x**

**cmp eax,1**

**je new\_coord**

**mov flag\_y,1**

**jmp LP\_MOVE**

**exit\_p:**

**call ExitProcess**

**Move endp**

**End**