**Лекция 12. «Спагетти или Лапша».**

**Замечание. В сегодняшней Лекции не будет никаких вопросов и не надо слать мне никаких ответов. Просто читаем и работаем НАД ЛАБАМИ – там вполне хватает «работы». Rfr levftnt? J<VFYEK,**

**Но прежде чем переходить непосредственно к поглощению спагетти, необходимо вспомнить, что было на прошлой лекции и что Вам нужно КОПАТЬ на лабораторных занятиях. Такое копание будете делать на примере решения одной из задач, включенной в билеты и, возможно, которая кому-то попадётся на контрольной, которая вероятнее всего будет проводиться 13.12.2023 года или 20.12.2023 – так называемый «контрольный выстрел». Эта задача предполагает получение полного имени исполнимого файла выполняемой программы. ПРИ ЭТОМ В РЕШЕНИИ ЭТОЙ ЗАДАЧИ ПРИСУТСТВУЕТ ФРАГМЕНТ, КОТОРЫЙ ВАМ ПОНАДОБИТСЯ И при решении Лабы №4. Я так понимаю, что мало кто из Вас догнал, прочитав по диагонали текст Лекции 11, как это можно добраться до имени исполнимого модуля. Ведь нужно было не Читать (буквы «Ч» нет в слове «ДУРКовать), а Думать, Учить, Рисовать, Копать. Поэтому приходится жевать чупа-чупа-чупс-чупс. Если кто уже обжевался и разобрался в том, как получено имя исполнимого файла выполняемой программы, дальнейшее описание можно пропустить до СЛОВ:**

**На прошлой лекции по сути дела был сформулирован ещё один постулат:**

**Постулат 5. Так как абсолютной защиты не существует, и так как «время-деньги», то первейшей задачей любого разработчика систем защиты данных, в основе которых лежит ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ, является ЗАДАЧА УСЛОЖНЕНИЯ для нехорошего человека, нехороших организаций процесса исследования того или иного программного продукта. А те, кому «не пошла» прошлая лекция, копаем:**

**Решение задачи нахождения имени исполнимого модуля в тексте Лекции 11 реализовано посредством такой процедуры:**

COPY\_FILE\_NAME PROC NEAR ; DS:BX - WHERE TO COPY FILE NAME

PUSH ES

MOV AX,ES:[2CH]

MOV ES,AX

MOV SI,-1

SEARCH\_01:

INC SI

MOV AL,ES:[SI]

CMP AL,0

JNE SEARCH\_01

MOV AL,ES:[SI+1]

CMP AL,1

JNE SEARCH\_01

ADD SI,2

COPY\_NAME:

INC SI

MOV AL,ES:[SI]

MOV [BX],AL

INC BX

CMP AL,0

JNE COPY\_NAME

POP ES

RET

COPY\_FILE\_NAME ENDP

**Речь идёт о том, что в формируемом при загрузке программы в память PSP (Program Segment Prefix) НАЧИНАЯ С АДРЕСА**

PROGRAM SEGMENT PREFIX

|  |  |  |  |
| --- | --- | --- | --- |
| offset | size | C O N T E N T S |  |
| 0000h | 2 bytes | int 20h |  |
| 0002h | 2 bytes | segment address, end of allocation block |  |
| 0004h | 1 byte | reserved, normally 0 |  |
| 0005h | 5 bytes | FAR call to MSDOS function dispatcher (int 21h) |  |
| 000Ah | 4 bytes | previous termination handler interrupt vector (int 22h) |  |
| 000Eh | 4 bytes | previous contents of ctrl-C interrupt vector (int 23h) |  |
| 0012h | 4 bytes | prev. critical error handler interrupt vector (int 24h) |  |
| 0016h | 22 bytes | reserved for DOS |  |
|  | \* 2 bytes | (16) parent process' PSP |  |
|  | \* 20 bytes | (18) "handle table" used for redirection of files |  |
| ***002Ch*** | ***2 bytes*** | ***address of the program's environment block*** |  |
| 002Eh | 34 bytes | reserved, DOS work area |  |
|  | \* 4 bytes | (2E) stores the calling process's stack pointer when switching to DOS's internal stack. |  |
|  |  | \* (32) DOS 3.x max open files |  |
|  | \* 2 bytes | (3A) size of handle table |these functions are in here |  |
|  | \* 4 bytes | (3C) handle table address |but reported addresses vary |  |
| 0050h | 3 bytes | int 21h, RETF instruction |  |
| 0053h | 2 bytes | reserved - unused? |  |
| 0055h | 7 bytes | reserved, or FCB#1 extension |  |
| 005Ch | 16 bytes | default unopened File Control Block #1 |  |
| 006Ch | 16 bytes | default unopened FCB #2 (overlaid if FCB #1 opened) |  |
| 0080h | 1 byte | parameter length (number of chars entered after filename) |  |
| 0081h | ... | parameters |  |
| ***00FFh*** | ***128 bytes*** | ***command tail and default Disk Transfer Area (DTA)*** |  |

**Размещается *address of the program's environment block* – ПЕРЕВОД ТАКОЙ: АДРЕС (ЧИСЛО) области (блока) оперативной памяти, содержащей данные об «окружении» выполняемой программы.**

**Для того, чтобы «добраться» до этого блока записано:**

PUSH ES

MOV AX,ES:[2CH]

MOV ES,AX

**В результате в регистре ES находится АДРЕС ЭТОГО БЛОКА. Теперь по нему надо «побегать» - НАЙТИ полное имя исполнимого модуля, из которого получается программа. ДОГОВОРИЛИСЬ (то ли дядьки, то ли тётьки, то ли с Microsoft, то ли с Intel, то ли together?), ЧТО ЭТО ИМЯ НАЧИНАЕТСЯ СРАЗУ ПОСЛЕ ДВУХ СЛЕДУЮЩИХ ДРУГ ЗА ДРУГОМ БАЙТОВ, СОДЕРЖАЩИХ ЗНАЧЕНИЯ 0 и 1 – СНАЧАЛА 0, потом 1! Поэтому в тексте есть такой фрагмент, в котором делается попытка найти эти два следующих друг за другом (бабушка – 0 рядышком с дедушкой -1) байта:**

SEARCH\_01:

INC SI

MOV AL,ES:[SI]

CMP AL,0

JNE SEARCH\_01

MOV AL,ES:[SI+1]

CMP AL,1

JNE SEARCH\_01

ADD SI,2

**Получая в регистре SI смещение начала имени исполнимого файла.**

**И ЕЩЁ ДОГОВОРИЛИСЬ(то ли дядьки, то ли тётьки, то ли с Microsoft, то ли с Intel, то ли together?), ЧТО ПРИЗНАКОМ ОКОНЧАНИЯ ЛЮБОЙ ПОСЛЕДОВАТЕЛЬНОСТИ СИМВОЛОВ, ТРЕБУЮЩЕЙ ФОРМИРОВАНИЯ ИЛИ ОБРАБОТКИ, ВСЕГДА И ВСЮДУ БУДЕТ БАЙТ, СОДЕРЖАЩИЙ ЗНАЧЕНИЕ 0! НЕ ГОВОРИТЕ НИКОГДА ВМЕСТО слов «последовательность символов» СЛОВО «СТРОКА». НЕЛЬЗЯ ГОВОРИТЬ, ЧТО ЧИСЛО НОЛЬ ЯВЛЯЕТСЯ ПРИЗНАКОМ ОКОНЧАНИЯ СТРОКИ! Так как английское слово «string» НЕЛЬЗЯ ПЕРЕВОДИТЬ на русский язык как «строка»!**

**Такой перевод приводит к неверной трактовке основополагающих понятий в разработке программного обеспечения. Только не спрашивайте, как следует переводить «string». Его не нужно переводить, либо, если переводить, то брать пятый-шестой перевод. Но лучше не переводить, а использовать адекватные, приятные уху кодировщика, подходящие по контексту слова. Как в данном случае «последовательность символов, признаком конца которой является символ с кодом ноль». Поэтому, чтобы «выщемить» нужную последовательность символов в тексте, который мы жуём, записан следующий фрагмент:**

COPY\_NAME:

INC SI

MOV AL,ES:[SI]

MOV [BX],AL

INC BX

CMP AL,0

JNE COPY\_NAME

DS:BX - WHERE TO COPY FILE NAME

**Теперь к нашим ramam.**

**На прошлой лекции по сути дела был сформулирован ещё один постулат:**

**Постулат 5. Так как абсолютной защиты не существует, и так как «время-деньги», то первейшей задачей любого разработчика систем защиты данных, в основе которых лежит ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ, является ЗАДАЧА УСЛОЖНЕНИЯ для нехорошего человека, нехороших организаций процесса исследования того или иного программного продукта.**

**Тогда же (в Лекции 11) был представлен один из возможных вариантов подобного усложнения – это определение того, находится ли в данный момент времени программа «под отладчиком», и удаление соответствующего ей исполнимого модуля в том случае, если такое имеет место быть.**

**Ещё один способ «запутывание редиски – нехорошего человека» предполагает сокрытие пароля непосредственно в тексте программы. А такое «сокрытие» чаще всего предполагает использование методики, которая основана на построении программы (совокупности команд, которая после декодирования её самой ЭВМ заставляет эту машину выполнить определённые действия) так, чтобы команды, формирующие пароль, были перепутаны очень похоже на то, каким образом перепутаны спагеттино-лапшовая смесь после варки и перемешивания.**

**Комбинирование двух способов: 1.«Вешания лапши на уши» нехорошему человеку и 2. «Защита от отладчика» - увеличивает вероятность того, что Ваш исполнимый модуль НЕ будет «взломан», так как: а) Очевидно, что простой взгляд на исполнимый модуль не позволит «увидеть» напрямую пароль – потребуется отладка – будь то отладка с целью «выщемить» пароль, или будь это отладка с целью обойти ветвь проверки пароля и напрямую перейти к функционалу программы; б) А как только будет сделана попытка осуществить «отладку», это приведёт к уничтожению исполнимого модуля.**

**Понятно, что среди Вас есть отличники, есть двоечники, есть супердвоечники. Для двух последних категорий далее приводится фрагмент исходного текста на языке записи алгоритмов ассемблер, в котором как раз делается попытка «спрятать» пароль. Посмотрите, может пригодится в жизни, хотя вряд ли…**

.model tiny

.data

input db 5,?,5 dup (?)

errmsg db 'FAIL',10,13,'$'

scmsg db 'SUCCESS',10,13,'$'

.code

.startup

main proc

; password

mov bx,15Fh

call todigit

push ax

mov bx,120h

call todigit

push ax

mov bx,15Ah

call todigit

push ax

mov bx,166h

call todigit

push ax

; input

mov ah,10

lea dx,input

int 21h

mov ah,4

cmp ah,input+1

jne miscompare

; comparison

mov bx,2

compsym:

pop ax

mov al,input[bx]

cmp ah,al

jne miscompare

inc bx

cmp bl,input+1

jne compsym

jmp success

miscompare:

lea dx,errmsg

jmp outmsg

success:

lea dx,scmsg

outmsg:

mov ah,9h

int 21h

…

mov ah,4ch

int 21h

ret

main endp

todigit proc

; BX byte offset

; returns symbol in AH

mov al,byte ptr [bx]

mov ah,0

mov dl,0Ah

div dl

xor ah,30h

ret

todigit endp

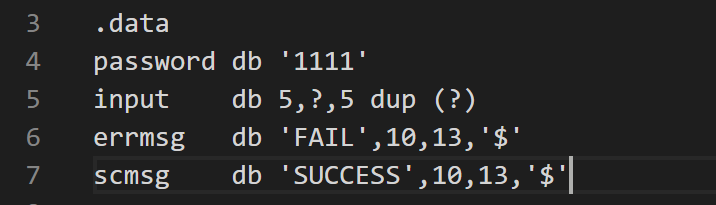
end

**Интересно, сколько Вам потребуется времени, чтобы определить пароль?**

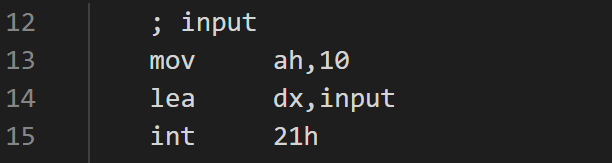
**ЗАДАНИЕ 12.1. КАК ТОЛЬКО «ВЫЩЕМИТЕ» ПАРОЛЬ, ПРИСЛАТЬ МНЕ ЕГО НА ПОЧТУ ПРОСТО В ВИДЕ СООБЩЕНИЯ: ПАРОЛЬ = \_\_\_\_\_\_\_\_\_\_\_\_\_.**

**Время «выщемления» будет определяться по времени получения мною письма. Понятно, что для «выщемления» пароля потребуется DosBox и отладчик, а посему к сообщению нужно прикрепить видео того, КАК это было.**

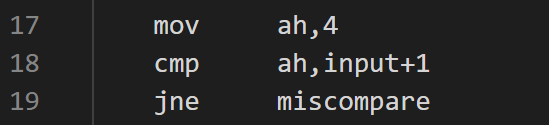
Далее представлен текст, совмещающий «упрятывание» пароля и вывод на экран полного имени файла:



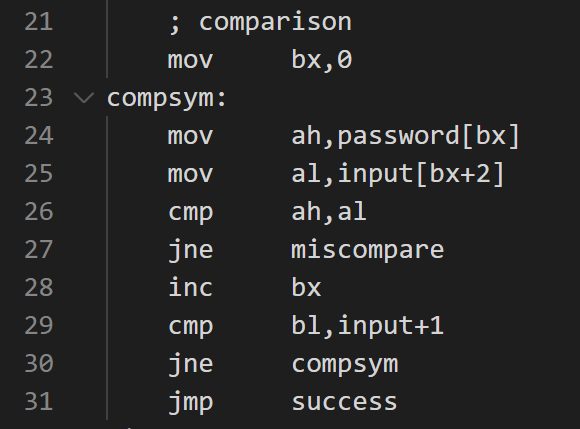
Ввод пароля пользователем:



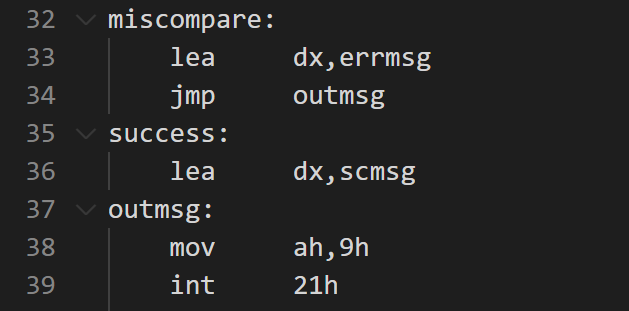
Сразу переходим к выводу сообщения об ошибке, если длина введенной строки и ожидаемая длина не совпадают:



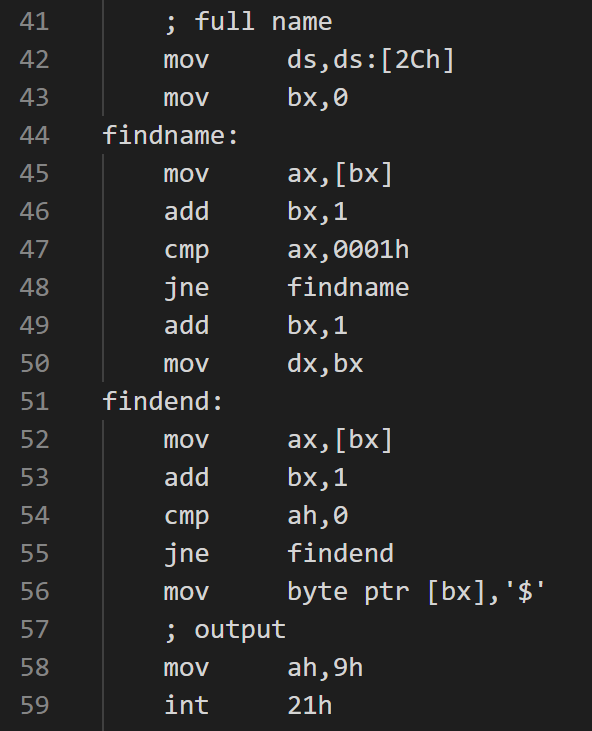
Если длины совпадают, сравниваем поочередно каждый байт. В случае ошибки переходим к выводу ошибки, иначе – к выводу сообщения об успехе:



Вывод:



Вывод полного имени:



Результат при вводе верного пароля:



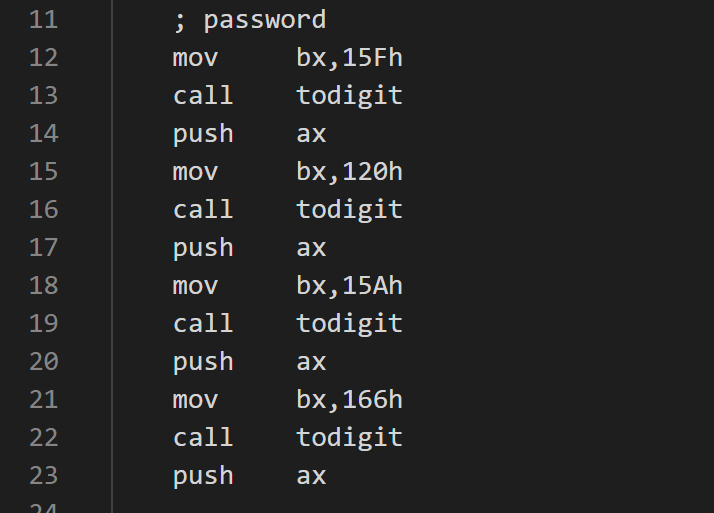


Результат при вводе неверного пароля:

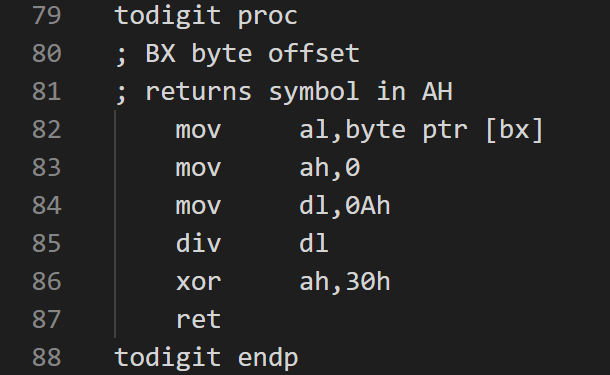




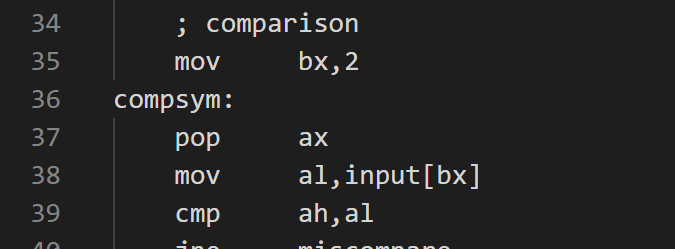
**Чтобы спрятать пароль**, в данном случае «1111», найдем 4 байта, из значений которых можно будет извлечь «1», взяв остаток от деления на 10 и добавив 30h. ***Например, это могут быть байты по адресам 15fh, 120h, 15ah, 166h.***



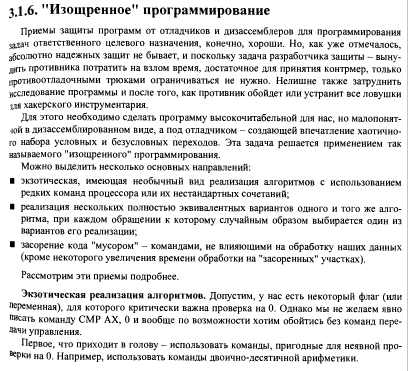
Каждый из байтов преобразуем следующим образом:

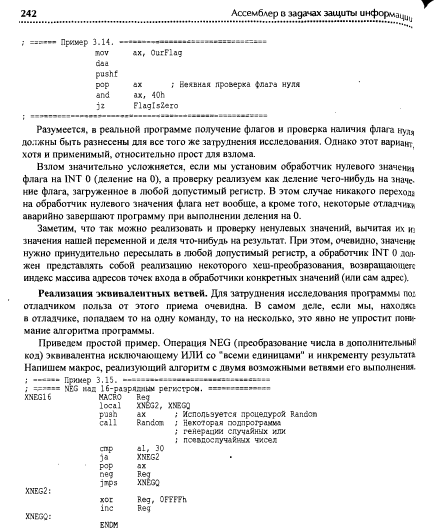


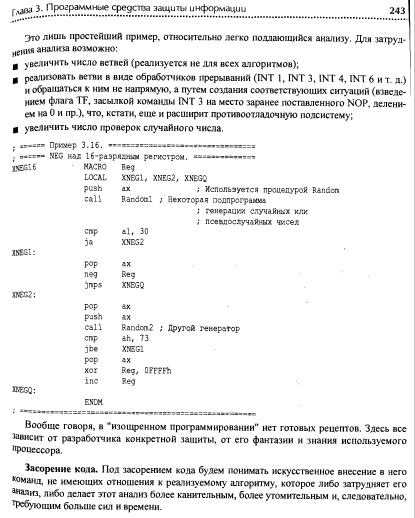
При сравнении байты будем извлекать по одному из стека (точнее, из стека достаем слова, где код символа – в верхнем байте):

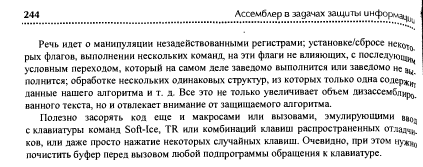


**В книге про ассемблер представлены ещё несколько вариантов «запутывания» программ. Эти варианты в любом случае потребуют от нехорошего человека попытки запустить программу под отладчиком… В том, что представлено далее, я не совсем «догоняю», поэтому «жевать» не буду, а предложу Вам «пожевать» самостоятельно. Только сильно не увлекайтесь …**









**Ещё один способ запутывания заключается в** получении **программы, которая меняет сама себя, когда кажется, что программа вроде бы должна заставить компьютер сделать некоторую операцию, а на самом деле – в процессе уже выполнения – процессор делает совсем другое.**

**Как это сделано, например, в следующем примере.**

**Изначально ТРЕБОВАЛОСЬ ПОЛУЧИТЬ ПРОГРАММУ, исполнимый модуль которой НЕ ПРЕВЫШАЕТ 36 байтов, и которая позволяет пользователю ввести одну цифру (только цифру, причём без проверки на правильность ввода) и в том случае, когда значение цифры лежит в диапазоне от 0 до 4 включительно получить на экране значение четвертой степени этой цифры, в противном случае нужно получить значение квадрата введенной цифры. Например, если введена цифра 4, на экране должно быть выведено число 256, а если цифра 9 – число 81. Исходный текст на ЯЗА ассемблер решения данной задачи выглядит следующим образом:**

.MODEL TINY

.CODE

.STARTUP

ORG 100H

XOR AX,AX

INT 16H

SUB AL,'0'

MUL AL

CMP AL,16

JG PRINT\_RESULT

MUL AL

PRINT\_RESULT:

MOV BX,2660

PRINT\_SYMBOL:

DIV BL

ADD AL,'0'

INT 29H

MOV AL,AH

MUL BH

CMP AL,0

JNE PRINT\_SYMBOL

MOV AH,4CH

INT 21H

END

**Понятно, что вряд ли среди Вас найдётся много желающих разобраться в том, что написано, действительно ли задача решена верно (то есть после трансляции, линковки и загрузки полученного исполнимого модуля размером не более 36 байтов и выполнения программы на экране будет получаться требуемый результат). А ПРИДЁТСЯ!!!**

**ДОМАШНЕЕ ЗАДАНИЕ 12.1 (РЕЗУЛЬТАТ В ВИДЕ ТРЁХ ФАЙЛОВ ПРИСЛАТЬ НА ПОЧТУ ДО 23-59 28.11.2023): Решить сформулированную задачу: набрать исходный текст, используя оболочку DosBox транслировать, исправить синтаксические ошибки (если они есть), пролинковать, получить исполнимый модуль, проверить правильность работы процессора под управлением программы ДЛЯ ВСЕХ ЦИФР (ЦИФРА – ЭТО НЕ ЧИСЛО!), удостовериться, что исполнимый модуль действительно не превосходит 36 байт (если превосходит, то добиться того, чтобы не превосходил), задокументировать процесс решения задачи.**

**На основе данной программы продемонстрируем, как можно получить программу, которая ИЗМЕНЯЕТ САМА СЕБЯ, когда кажется, если смотреть на исходный текст, что программа вроде бы должна заставить компьютер выполнить строго определенные действия, а на самом деле – в процессе уже выполнения – процессор делает совсем другое.**

**Например, пусть программа вывода на экран либо четвёртой, либо второй степени цифры изменяет себя так, что НА ЭКРАН ВСЕГДА ВЫВОДИТСЯ КВАДРАТ ВВЕДЁННОЙ ЦИФРЫ.**

**ВСЕ ПОМНЯТ, ЧТО ТАКОЕ «ПРОГРАММА»? ВСЕ ЗНАЮТ, ЧТО КОМПЬЮТЕРЫ Фон-Неймановского типа ПРЕДПОЛАГАЮТ, что КОМАНДЫ и ДАННЫЕ располагаются в оперативной памяти? В связи с чем, ЛЮБАЯ КОМАНДА НАХОДИТСЯ В ОПЕРАТИВНОЙ ПАМЯТИ И ИМЕЕТ СТРОГО ОПРЕДЕЛЕННЫЙ АДРЕС, равный содержимому сегментного регистра плюс смещение, И ЛЮБАЯ КОМАНДА ИМЕЕТ СТРОГО ОПРЕДЕЛЕННЫЙ ФОРМАТ, ВКЛЮЧАЯ КОД ОПЕРАЦИИ И АДРЕСА ОПЕРАНДОВ. А СИЕ ОЗНАЧАЕТ, ЧТО, ЗНАЯ АДРЕС КОМАНДЫ И ЕЁ ФОРМАТ (А ИХ МОЖНО ПОСМОТРЕТЬ В ОТЛАДЧИКЕ), В ПРОЦЕССЕ ВЫПОЛНЕНИЯ САМОЙ ПРОГРАММЫ МОЖНО ЛЕГКО ИЗМЕНИТЬ ЛЮБУЮ ИЗ КОМАНД, КОТОРАЯ БУДЕТ ВЫПОЛНЕНА ВПОСЛЕДСТВИИ. Например,** для изменения самой себя, можно, записав предварительно необходимые команды, обратиться к памяти по адресу изменяемой команды и изменить «содержимое команды» таким образом, чтобы представление второй команды mul ПРЕВРАТИТЬ В КОМАНДУ NOP, чтобы при выполнении программы при любом значении цифры получить на выходе квадрат):

.MODEL TINY

.CODE

.STARTUP

ORG 100H

mov ax, 00h

mov [cs:0113h], ax

XOR AX,AX

INT 16H

SUB AL,'0'

MUL AL

CMP AL,16

JG PRINT\_RESULT

MUL AL

PRINT\_RESULT:

MOV BX,2660

PRINT\_SYMBOL:

DIV BL

ADD AL,'0'

INT 29H

MOV AL,AH

MUL BH

CMP AL,0

JNE PRINT\_SYMBOL

MOV AH,4CH

INT 21H

END

ПОЛУЧИЛОСЬ? Обратите внимание НА КОМАНДУ

MOV BX, 2660

(что за БСК!!!!! Разберитесь пожалуйста! Для этого нужно нарисовать содержимое РЕГИСТРА BX в шестнадцатеричном виде и ВНИМАТЕЛЬНО ПОСМОТРЕТЬ НА СОДЕРЖИМОЕ КАЖДОГО ИЗ РЕГИСТРОВ, ИЗ КОТОРЫХ СОСТОИТ BX – на содержимое регистров BL и BH, и на ТО, КАК ЭТО ИСПОЛЬЗУЕТСЯ В ЦИКЛЕ PRINT\_SYMBOL.

.STARTUP – ‘nj lbhtrnbdf nhfyckznjhf, rjnjhfz ujdjhbn tve? Xnj lfktt ,elen cktljdfnm rjvfyls bp ctuvtynf rjljd b xnj njxrjq d[jlf ghjuhfvvs ,eltn cktle.ofz pf lbhtrnbdjq rjvfylf/

Домашнее задание 12.2. ПРИДУМАТЬ **СВОЙ** ВАРИАНТ ПРОГРАММЫ, КОТОРАЯ ИЗМЕНЯЕТ САМА СЕБЯ НА ОСНОВЕ ЛЮБОЙ ИЗ РАССМОТРЕННЫХ В РАМКАХ КУРСА ЛЕКЦИЙ ПРОГРАММ (например, про папу, маму, про кодировочную таблицу, про цифры, про вируса, про шпиона, про самоубийцу и т.д… То есть нужно приготовить спагетти или из вируса, или шпиона, или папы, или таблицы, или самоубийцы или из их частей…) . **(РЕЗУЛЬТАТ В ВИДЕ ТРЁХ ФАЙЛОВ ПРИСЛАТЬ НА ПОЧТУ ДО 23-59 5.12.2023 и с подробным описанием алгоритма «самоизменения»).**