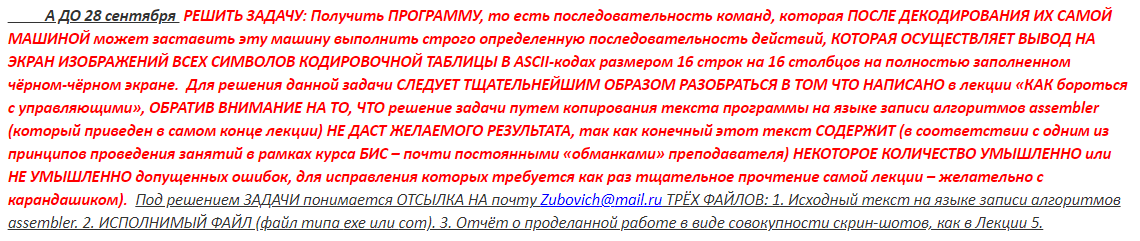
Жуковский Павел Сергеевич, 4 курс, 12 группа

Лабораторная работа №1

Пароль от архива: **041**

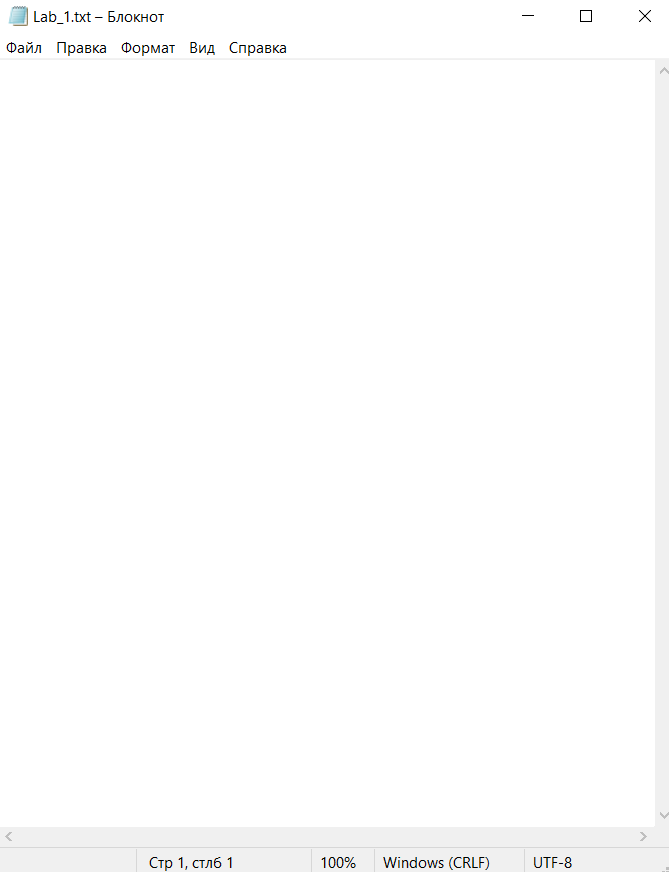
Постановка задачи:



Алгоритм решения:

Для начала я открыл 23-ю лекцию Зубовича под названием «Как бороться с управляющими».

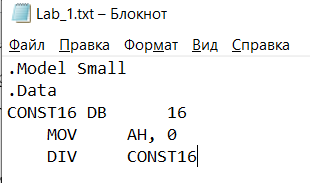
Далее, я вспомнил, что вы не любите различные оболочки для работы с Assembler, поэтому я по старинке создал блокнот с именем Lab\_1.txt, где и начинал работу над исходным текстом программы:



Моим планом было через терминал получить объектный модуль моего исходного текста (\*.obj файл), а затем слинковать его в исполняемый файл (\*.exe файл) и запустить, когда я закончу.

Итак, вернёмся к исходному тексту.

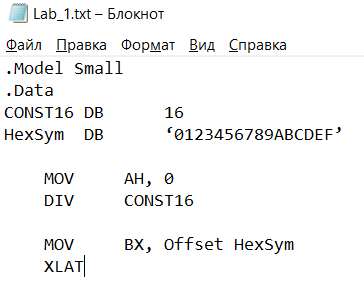
Для начала, как было сказано в лекции, я указал в сегменте данных константу CONST16, а затем с помощью неё находил значение шестнадцатеричных чисел:



На тот момент, я ещё не пытался запускать программу (и не думал).

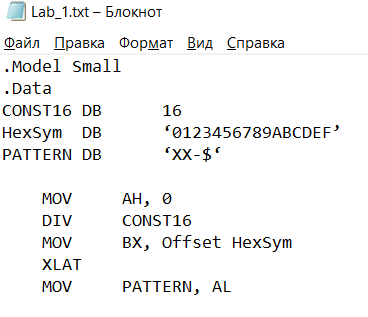
Далее, в лекциях было указано использовать команду XLAT (которая, вроде как уже должна была нам быть известна?). Я, честно, не знал, как она работает, поэтому подсмотрел во всемирной липкости и узнал, что эта команда предназначена для декодирования байта по заданой таблице. Как раз то, что нам нужно…

Таким образом, я использовал XLAT (не забыв добавить HexSym в сегмент данных):

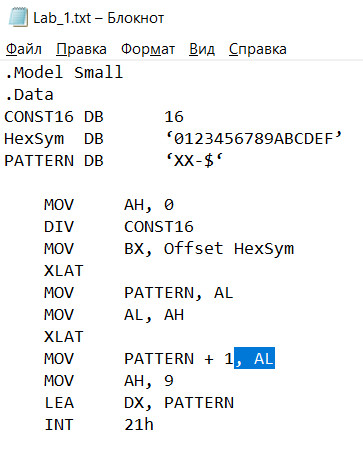


По итогу в регистре AL – символ, который совпадаем с символом, обозначающим значение шестнадцатеричной цифры, что находилась после команды деления в регистре AL (частное от деления кода символа на 16 или значение первой шестнадцатеричной цифры).

Далее, нужно было подготовить область памяти в сегменте данных под три байта (а именно для двух символов-цифр и черточки):



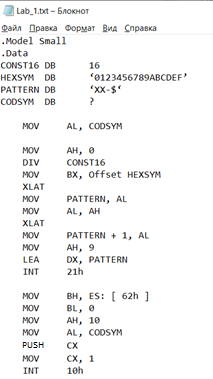
Также необходимо было сделать преобразование из значения в символьное представление для второй цифры (которая осталась в регистре AH) и полученное представление вывести на экран с помощью 21h функции прерывания 9. Так я получил основной кусок программы:



Почему-то у вас в лекциях для команды MOV была пропущена вторая часть (регистр AL), из которой мы перемещаем данные в PATTERN + 1, поэтому я дописал его туда. (в противном случае, программа бы попросту не запустилась). Я предположил, что это первая ваша ловушка.

Далее, как я прочитал в лекциях Зубовича, нам было скорректировать вывод символов, для этого необходимо было добраться до BIOS AREA, а для этого можно было воспользоваться системным регистром ES, собственно я и воспользовался. Также нужно было заполучить в регистре BH номер активной видеостраницы, моэтому также пригодилась команда MOV BH, ES: [ 62h ]. В ргеистре AH будет находиться номер функции, которая выводит символы на экран. Также понадобится область памяти для хранения кода очередного символа (ибо в AL он будет постоянно меняться). Помимо того также нужно было определить CODSYM в сегменте данных.

По итогу:

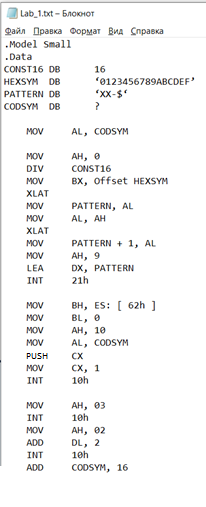


И здесь в лекциях Зубовича было ещё три ловушки:

1. Вы почему-то не поставили «:» перед [ 62h ], хотя изначально записали правильно.
2. Вы долго рассказывали, что для управления дисплеем нам нужно прерывание 10h, хотя в своём исходном тексте в лекции почему-то оставили 21h. (поэтому я исправил на 10h, как и планировалось)
3. Нужно же уменьшить значение регистра стека на размер CX, добавив строку PUSH CX перед тем, как помещать туда 1 и выполнить функцию.

Далее нужно было поработать над курсором, а именно над определением его мостоположения и последующим движением. Нам нужно было изменить положение курсора после вывода предыдущего символа вправо на 2 позиции (чтобы передвинуть курсор через выведенных символ это раз, чтобы передвинуть курсор для отделения одного символа от другого это два). К этому всему надо было также добавить фрагмент изменения значения кода символа (получив изменение параметров цикла).

По итогу:

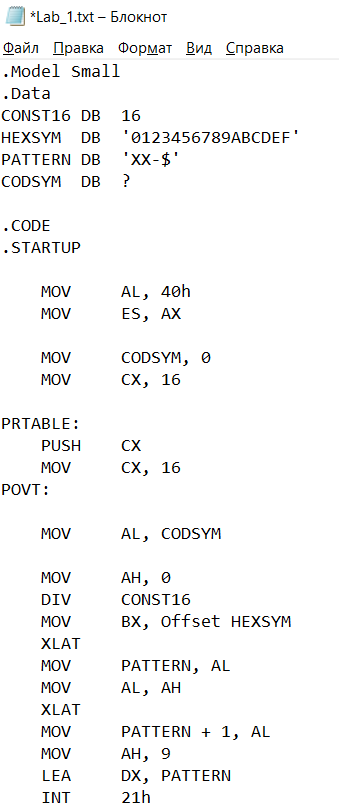


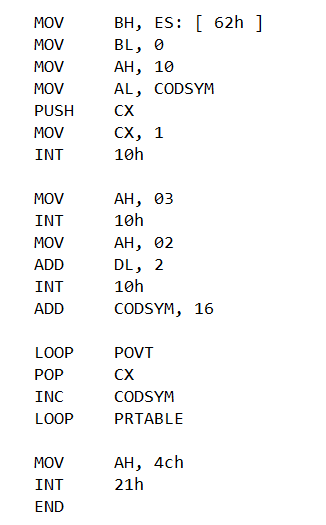
Опять таки, ещё одна ловушка Зубовича! Для перемещения курсора нам нужно прерывание 10h, а не 21h, поэтому я это исправил. (в лекциях Зубовича после INT стояли 21h)

Далее, нам необходимо было зациклить наш вывод, чтобы вывести 16 строк на экран. Но одного цикла (внутреннего нам недостаточно), т.к. в конце итерации в регистре CX остался бы 0 и всё пошло бы одному месту. Поэтому также необходимо было добавить внешний цикл, а значение CX поместить в стек, а затем вернуть обратно, когда понадобится. (так мы избавимся от нуля)

Также, в исходном тексте лекции я обнаружил небольшую ловушку. «MOV CODSYM, 0.» 🡨 здесь почему-то стояла точка в конце (это абсолютно неприемлимо для программы, поэтому я убрал её)

Итак, я попробовал запустить следующую программу:



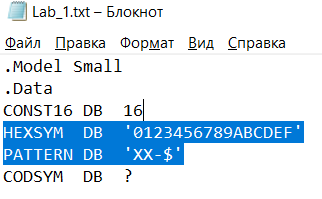


Результат:

\_28.09.2021 9:57:59 : \*\*Error\*\* polygon.asm(4) Undefined symbol: ?0123456789ABCDEF?

\_28.09.2021 9:57:59 : \*\*Error\*\* polygon.asm(5) Undefined symbol: ?XX

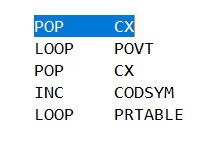
Похоже, это ещё одна ловушка Зубовича, нужно было поправить кавычки в сегменте данных:



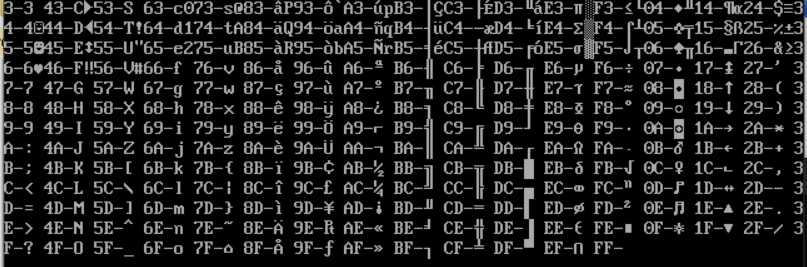
Теперь снова попробовал запустить:



Позже я пришёл к выводу, что перед конструкцией LOOP необходимо всё-таки удалить содержимое регистра CX, т.е. добавить строку:



После этого я снова попробовал запустить программу:

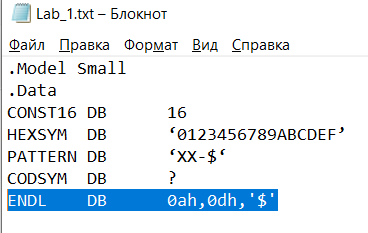


Уже было лучше, но всё равно было что-то не то.

Почему-то, с верхней частью таблицы были проблемы.

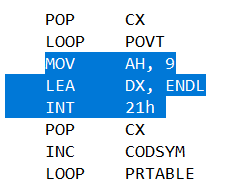
Немного подумав, я понял, что нам было бы неплохо оформить вывод строку внутри цикла…

Для этого я добавил в сегмент данных конструкцию, которая соответствует переходу на новую строку ‘\n’ и возврат картеки ‘\n’:



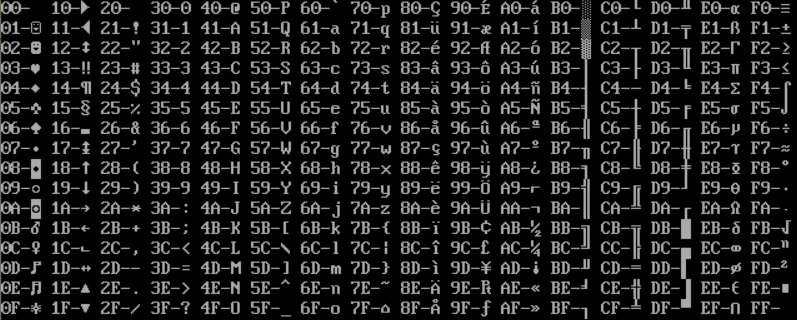
Не имеет значения, как мы его назовём. Но я назвал ENDL, потому что в языке C++ такая же конструкция переводит нас на новую строку. (мне так проще ассоциировать). Здесь 0ah – соответствует переходу на новую строку, а 0dh – закрывающей каретке (но нас она не очень интересует).

Далее я добавил в цикл следующую конструкцию:



Здесь мы помещаем в регистр AH значение 9 (почему 9, а не 8 или 10? Потому что 9 – номер функции перехода на новую строку, которая нам и нужна). Затем я даю регистру DX представление о том, что такое ENDl командой LEA (она работает примерно как MOV, только выводит не первые два байта, а всё, что нам нужно). А под конец вызываю INT 21h, чтобы вызвать функцию.

После всех мучений мне удалось получить то, что я планировал:



Ещё разок, исходный текст финальной программы:

.Model Small

.Data

CONST16 DB 16

HEXSYM DB '0123456789ABCDEF'

PATTERN DB 'XX-$'

CODSYM DB ?

ENDL DB 0ah,0dh,'$'

.CODE

.STARTUP

MOV AL, 40h

MOV ES, AX

MOV CODSYM, 0

MOV CX, 16

PRTABLE:

PUSH CX

MOV CX, 16

POVT:

MOV AL, CODSYM

MOV AH, 0

DIV CONST16

MOV BX, Offset HEXSYM

XLAT

MOV PATTERN, AL

MOV AL, AH

XLAT

MOV PATTERN + 1, AL

MOV AH, 9

LEA DX, PATTERN

INT 21h

MOV BH, ES: [ 62h ]

MOV BL, 0

MOV AH, 10

MOV AL, CODSYM

PUSH CX

MOV CX, 1

INT 10h

MOV AH, 03

INT 10h

MOV AH, 02

ADD DL, 2

INT 10h

ADD CODSYM, 16

POP CX

LOOP POVT

MOV AH, 9

LEA DX, ENDL

INT 21h

POP CX

INC CODSYM

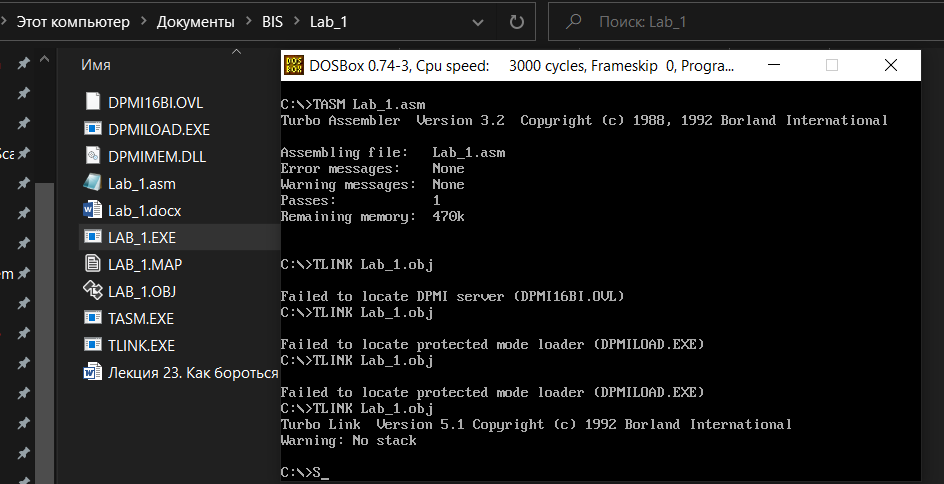
LOOP PRTABLE

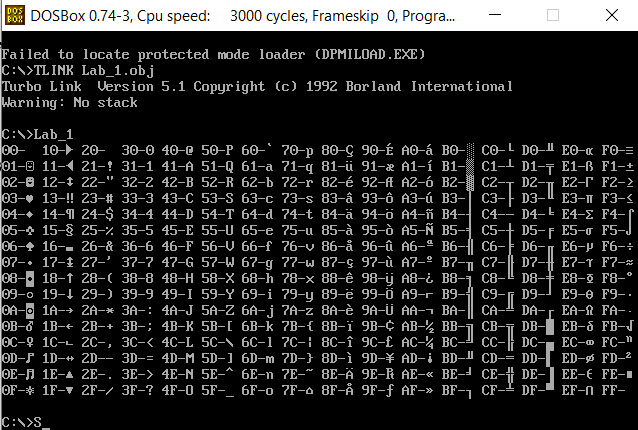
MOV AH, 4ch

INT 21h

END

В кратце покажу, как я получил исполняемый файл программы:





В письме я постараюсь прикрепить архив, который будет включать в себя всё этот отчёт, исполняемый файл и исходный текст.

Пароль от архива: **041** (на всякий случай, укажу пароль на самом верху этого отчёта тоже)