Лабораторная работа №1

Вывод на экран полной кодировочной таблицы изображений ASCII кодов символов

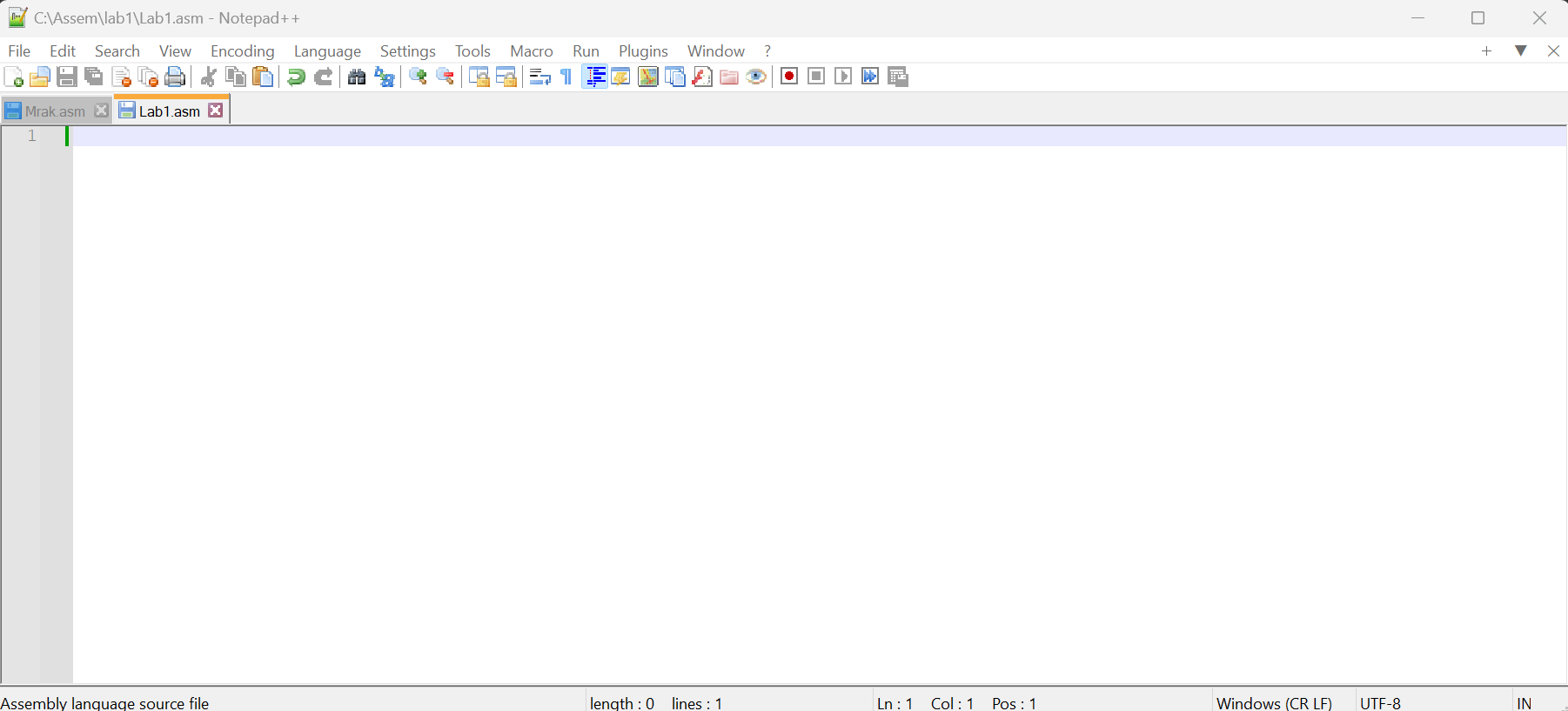
**Задание**

*«Получить программу (не приложение, не исходный текст на записи алгоритмов, не программу на языке C++), которая позволяет вывести на экран полную кодировочную таблицу изображений ASCII кодов символов в виде таблицы 16 на 16 таким образом, что на каждый символ отводится 5 позиций. 1 позиция – изображение позиции символа, 2 позиция – чёрточка, 3 позиция – шестнадцатеричное значение кода символа и далее пробел. Например, A – 41 B – 42.»*

**Ход решения**

Для выполнения данного задания основным источником была лекция К.А. Зубовича «Лекция 23. Как бороться с управляющими». Стоит заметить, что исходный код, предоставленный в файле, содержит некоторые ошибки, из-за которых программа не будет работать. Это также будет отображено в отчёте.

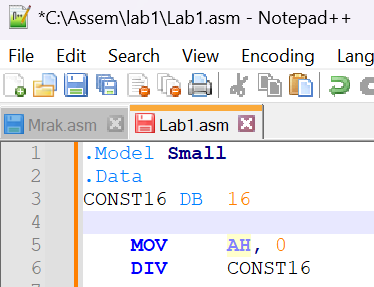
Итак, начнём с того, что создадим файл на языке ассемблер. Именно на нём нам предлагает написать исходный текст будущей программы. Файл я назвала соответствующе Lab1. Все программы на ассемблере я пишу в Notebook++.



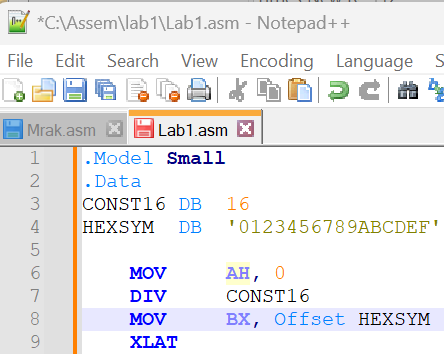
Я решила воспользоваться тем же «планом действий», как и при выполнении лабораторного занятия №2. То самое занятие, где мы вместо рамы мыли папу. А если конкретнее, то воспользоваться DosBox, а также компилятором Tasm и линкером Tlink. То есть, сначала мы получаем модуль исходного текста, а потом превращаем его в исполняемый файл.

Лирическое отступление завершено. Переходим к делу.

Как и было указано в лекции, «сначала найдем ЗНАЧЕНИЯ шестнадцатеричных цифр, разделив содержимое регистра AL на 16». Не забудем описать в сегменте данных наш CONST 16.

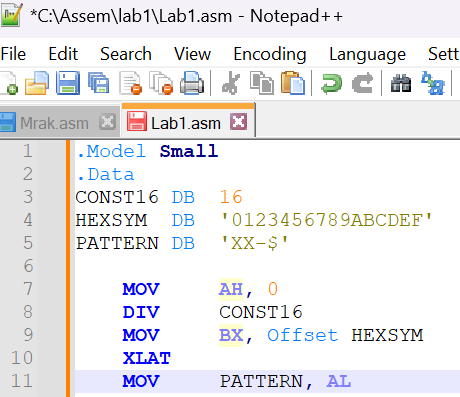


Далее мы будем использовать команду XLAT. Благодаря ей, мы сможем вывести на экран значения полученных в регистрах AH и AL шестнадцатеричных цифр в символьном виде.

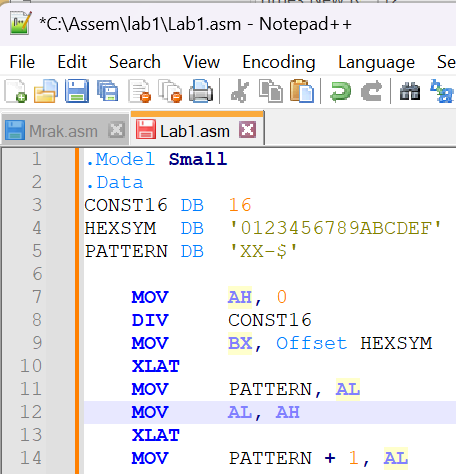


После выполнения команды XLAT в регистре AL будет находиться символ, соответствующий значению первой шестнадцатеричной цифры, полученной в результате деления кода символа на 16. Это значение представляет собой частное от этого деления.

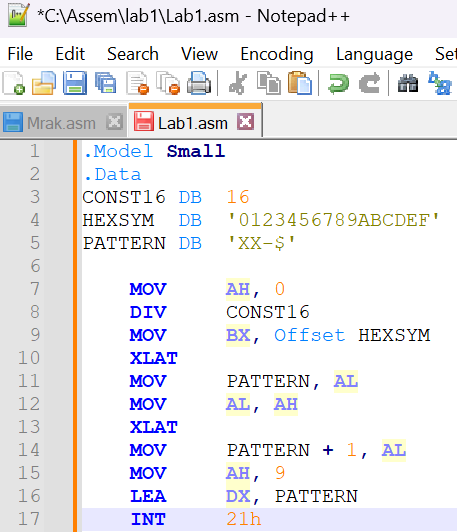
Чтобы отображать на экране цифры и черточку, создадим в сегменте данных область памяти, состоящую из трех байтов. В этой области мы разместим коды двух цифр и код черточки, не изменяя при этом значение черточки.



Затем мы выполняем аналогичное преобразование значения из регистра AH в символьное представление для второй цифры.



Чтобы завершить данный кусочек, нам осталось только вывести полученное представление с помощью 21h функции и прерывания 9.

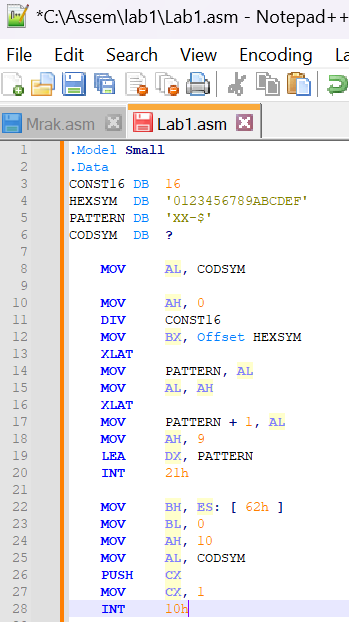


И, видимо, я наткнулась на ваши ловушки!

В ваших лекциях по команде MOV была упущена вторая часть, касающаяся регистра AL, из которого данные перемещаются в PATTERN + 1. Из-за этого программа будет нерабочей.

Также, нам нужно было скорректировать вывод символов, для чего необходимо было обратиться к области BIOS. Для этого я использовала системный регистр ES. Также нужно было получить номер активной видеостраницы, и для этого пригодилась команда MOV BH, ES:[62h]. В регистре AH хранится номер функции, отвечающей за вывод символов на экран. Кроме того, потребуется область памяти для хранения кода текущего символа, так как в AL он будет постоянно изменяться (не забываем определить CODSYM в сегменте данных).

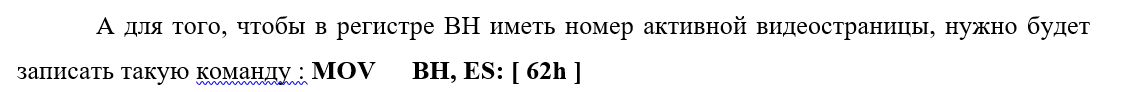
В общем, должно получиться что-то такое:



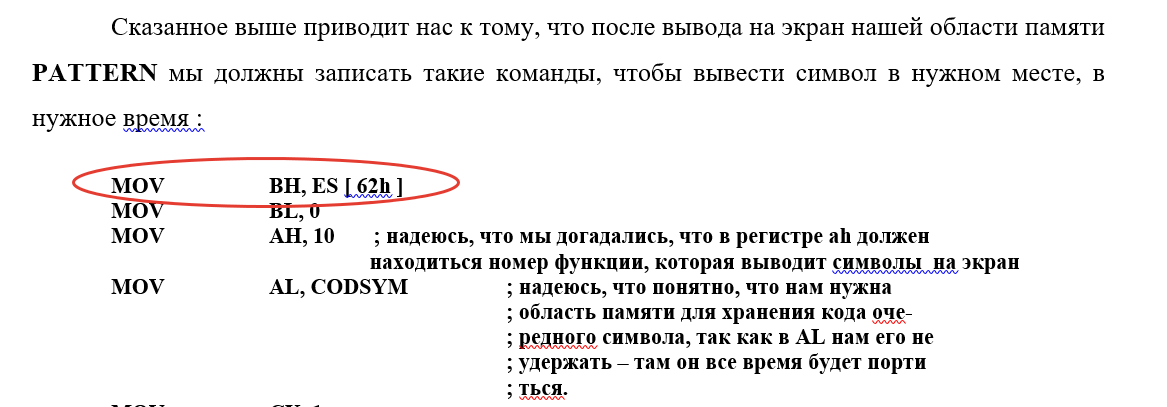
Серым я как раз выделила то, что было добавлено.

Идём дальше по ловушкам Константина Антоновича.

В самом конце лекции есть исходный текст программы, которая работать не будет.

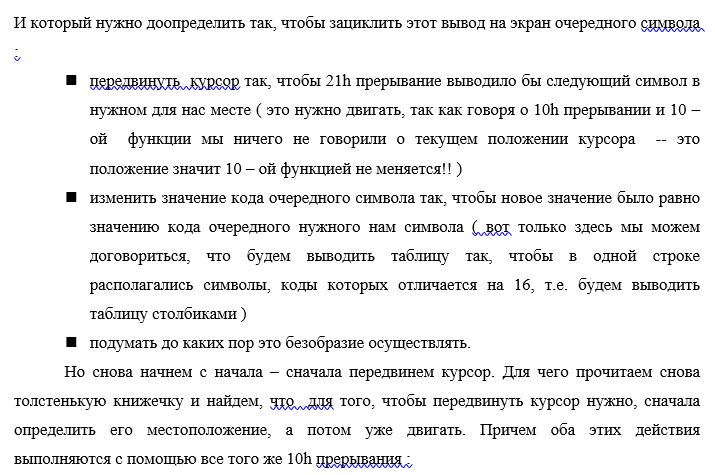


Здесь вы правильно указываете команду, но в дальнейшем, почему-то, не ставите необходимое двоеточие:

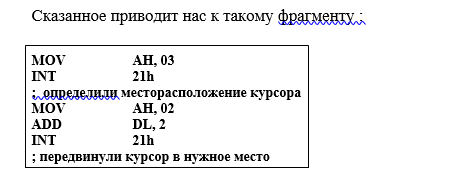


Не могу сказать точно: была это ваша ловушка или нет, но тем не менее.

Идём дальше. Второй недочёт (или ловушка).

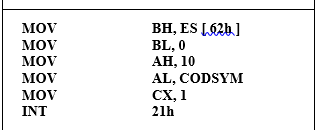


Тут вы подробно объясняете, что нам необходимо использовать прерывание 10h, но, в итоге, указываете 21h.



Этот момент был исправлен.

Третий момент. Необходимо было уменьшить значение регистра стека на величину CX, добавив строку PUSH CX перед тем, как установить значение 1 и вызвать функцию.



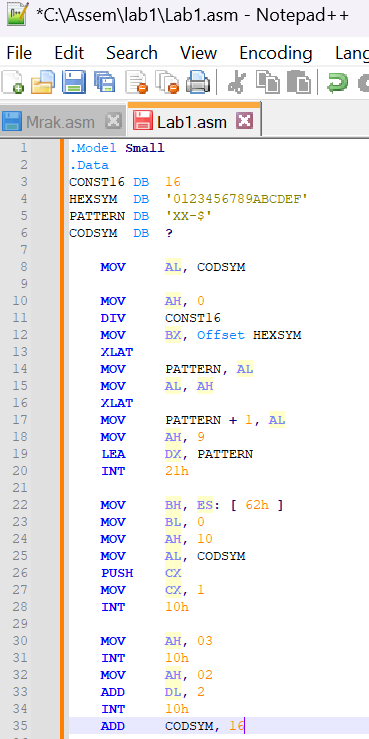
Выглядит это так:

**PUSH CX**

**MOV CX, 1**

Далее нужно было сосредоточиться на курсоре, а именно определить его текущее положение и реализовать его перемещение. После вывода предыдущего символа курсор должен был сдвинуться вправо на 2 позиции: во-первых, чтобы обойти уже выведенные символы, а во-вторых, для создания промежутка между символами. Также необходимо было внести изменения в код символа, чтобы адаптировать параметры цикла.

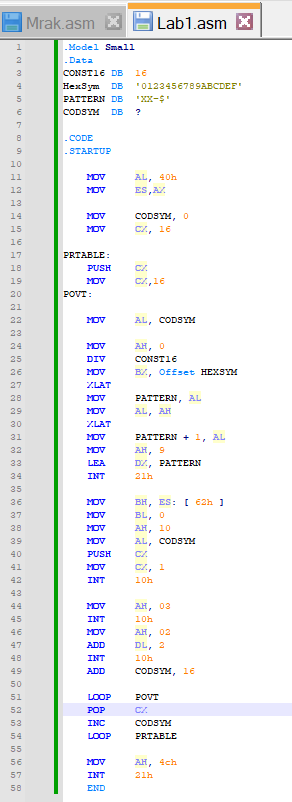
В общем, имеем мы такую красоту:



Для перемещения курсора нужно использовать прерывание 10h, а не 21h. Об этом уже говорилось немного ранее.

Кроме того, чтобы вывести 16 строк на экран, нам нужен внешний цикл, так как одного внутреннего цикла недостаточно. В противном случае в конце итерации регистр CX окажется равным 0, и выполнение программы остановится. Поэтому я добавила внешний цикл и поместила значение CX в стек, чтобы вернуть его обратно, когда это понадобится, что поможет избежать проблемы с нулем.

Теперь у нас есть такой исходный текст программы:



Видно не очень хорошо, поэтому продублирую для чёткости текстом:

.Model Small

.Data

CONST16 DB 16

HexSym DB '0123456789ABCDEF'

PATTERN DB 'XX-$'

CODSYM DB ?

.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

LOOP POVT

POP CX

INC CODSYM

LOOP PRTABLE

MOV AH, 4ch

INT 21h

END

Попробуем запустить.

Краткий план действий при запуске:

1. Открываем DosBox
2. Переходим на в папку с исходным текстом (покажу на своём примере):

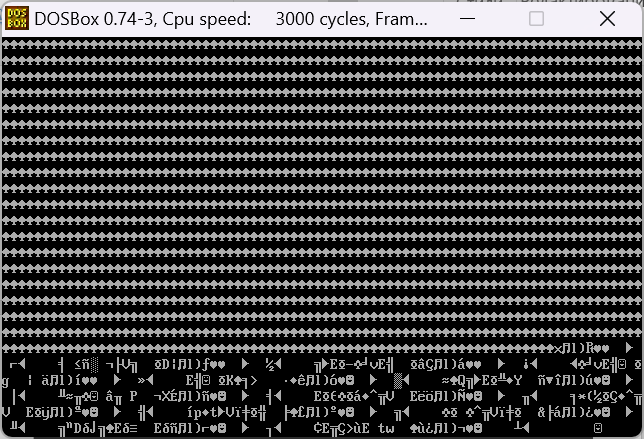
**mount c c:/assem/lab1**

1. Пишем **c:** и так переходим в нужную папку
2. Прописываем **TASM.EXE lab1.asm.** ВАЖНО: TASM у вас должен лежать в папке с исходным текстом. Как и TLINK со всеми необходимыми ему файлами.

После запуска мы видим следующее:

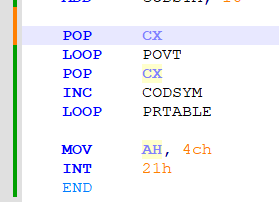


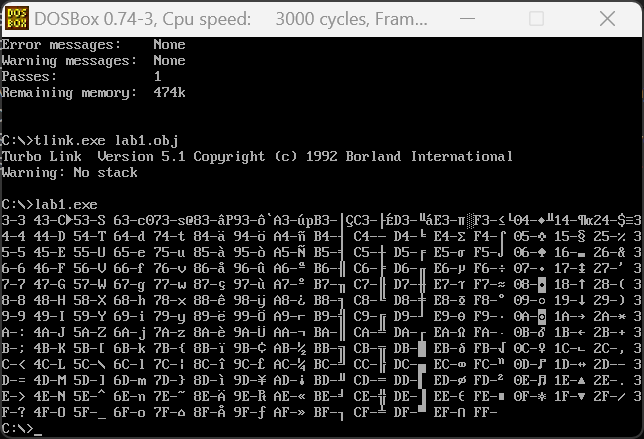
Это значит, что ошибок нет. Можем линковать. Делается это так же просто, как и компилируется. Просто пишем TLINK.EXE lab1.obj и всё. А после этого можно запускать нашу программу.



Получилось что-то страшное. Сейчас будем разбираться, в чём ошибка.

Попробуем удалять содержимое регистра CX. Просто будем добавлять строку.

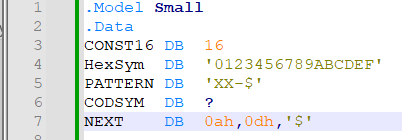




Красиво, но всё ещё не то, что нам надо.

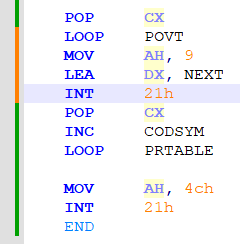
Дабы исправить это недоразумение, можно попробовать добавить в сегмент данных конструкцию, которая соответствует переходу на новую строку ‘и возврату каретки. Что за каретка? Это команда, которая перемещает курсор в начало строки.

Выглядеть это будет примерно так:



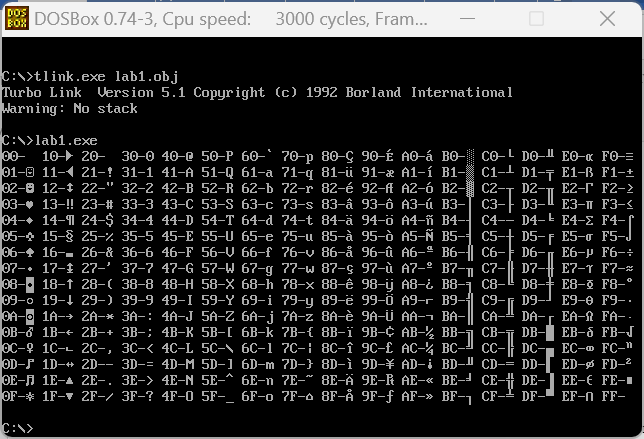
Не имеет значения, как мы это назовем. В этом случае 0ah соответствует переходу на новую строку, а 0dh — закрывающей каретке.

Теперь, нам нужно добавить ещё вот такой цикл:



Здесь мы загружаем в регистр AH значение 9, поскольку это номер функции, отвечающей за переход на новую строку. Почему именно 9? Потому что 9 — это нужная нам функция. Затем с помощью команды LEA мы передаем регистру DX адрес строки, которая будет выведена. В завершение вызываем прерывание INT 21h, чтобы активировать эту функцию.

Момент истины. Пробуем запустить программу ещё раз.



Ура, получилось! Можно смело идти гулять!

Ещё раз дублирую полностью исходный текст программы:

.Model Small

.Data

CONST16 DB 16

HexSym DB '0123456789ABCDEF'

PATTERN DB 'XX-$'

CODSYM DB ?

NEXT 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, NEXT

INT 21h

POP CX

INC CODSYM

LOOP PRTABLE

MOV AH, 4ch

INT 21h

END

Файлы, которые понадобятся при компиляции и линковке программы:

* DPMI16BI.OVL
* DPMILOAD.EXE
* DPMIMEM.DLL
* TASM.EXE
* TLINK.EXE