Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

**Лабораторная работа №1 (вариант 11)**

по дисциплине: «Ассемблеры и их применение».

Выполнил:

студент 3 курса, гр. ИВТАПбд-31

Кондратьев Павел Сергеевич.

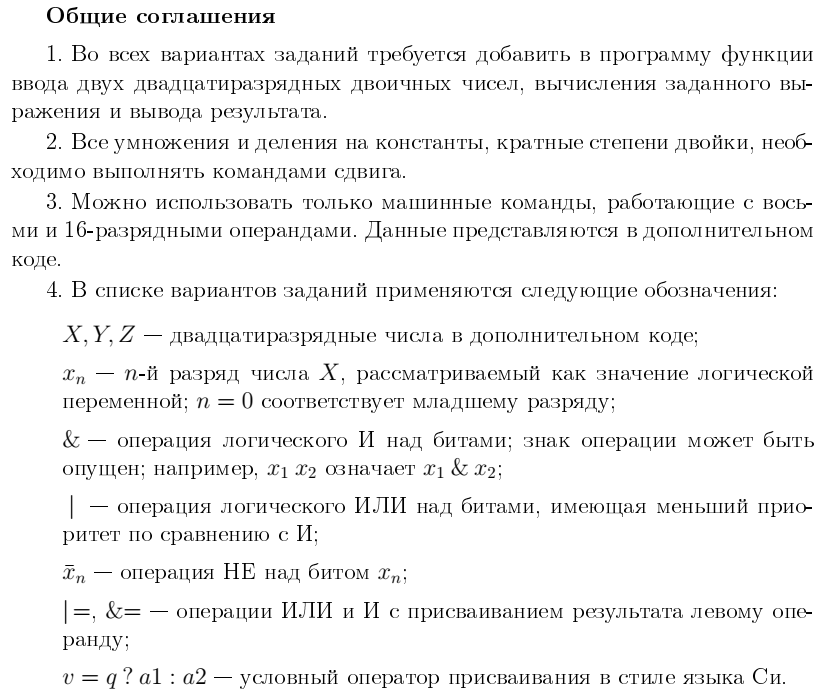
Проверил:

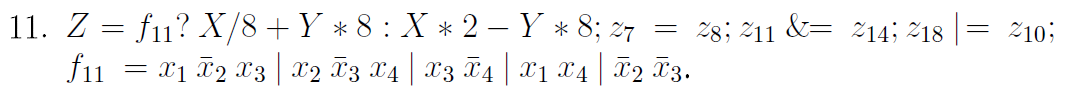
преподаватель кафедры ВТ

Лылова Анна Вячеславовна.

г. Ульяновск, 2019

**1) Задание:**





**2) Описание операций циклического процесса:**

Разработка программы включает несколько этапов:

* подготовка (изменение) исходного текста программы,
* ассемблирование программы (получение объектного кода),
* компоновка программы (получение исполняемого файла программы),
* запуск программы,
* отладка программы.

Обычно эти этапы циклически повторяются, потому что при нахождении ошибок при ассемблировании, компоновки или отладке приходится вновь возвращаться к первому этапу и изменять текст программы для устранения ошибок.

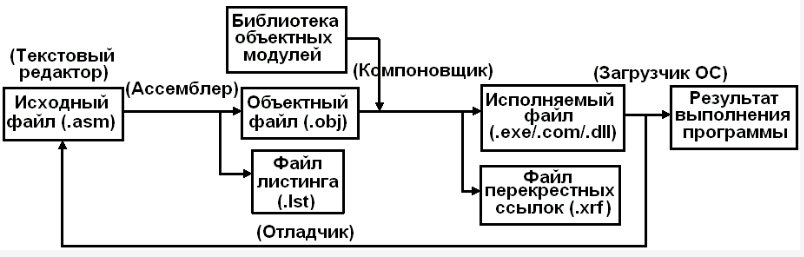


Рис. 1.1. Этапы разработки программ на ассемблере

**Подготовка текста программы**

Текст программы записывается в один или несколько текстовых файлов. Имена файлов могут быть любыми, но для файлов, содержащих текст программы, принято расширение .asm, а для файлов с определением констант и новых типов – расширение .inc. Эти файлы являются текстовыми, их можно подготовить с помощью стандартных редакторов текста или с использованием интегрированных сред разработки программ.

**Ассемблирование программы**

Подготовленный текст является исходными данными для специальных программ, называемых ассемблерами. Задача ассемблеров – преобразовать текст программы в форму двоичных команд, которые могут быть выполнены микропроцессором. Если обнаружены синтаксические ошибки, то результирующий код создан не будет. Процесс создания исполняемого файла происходит в две стадии:

.asm -> .obj -> .exe/.dll/.com

На первой стадии (.asm -> .obj) из ассемблерного файла путем компиляции получаются файлы промежуточного объектного кода, имеющего расширение .obj (при этом могут использоваться дополнительные inc-файлы). Файл с расширением .obj содержит машинный код при условии, что не встретились синтаксические и семантические ошибки. Если в исходном файле с программой на языке ассемблера обнаруживаются ошибки, то программисту выдается список обнаруженных ошибок, в котором ошибки указываются с номером строки, в которой они обнаружены. Программист циклически выполняет действия по редактированию и компиляции до тех пор, пока не будут устранены все ошибки в исходном файле. На этом этапе уже возможно получение готовой программы, но чаще всего в ней не хватает некоторых компонентов. Если компилятор по какой-либо причине (неверно прописан путь к такому файлу или файл отсутствует) не может найти inc-файл, то выдается предупреждение и obj-файл получен не будет.

Ассемблирование, как правило, проходит в два приема. При первом проходе переводятся мнемонические команды, десятеричные числа и символы в соответствующие машинные коды, подсчитывается, сколько какая команда занимает места, обнаруженные имена, введенные пользователем (константы, метки, переменные) их тип и числовое значение записывается в таблицу. В эту же таблицу записывается, с каких адресов начинаются процедуры, адреса меток, адреса начала/конца сегментов и т. д., при втором проходе подставляются адреса начала процедур, заменяются названия меток на адреса.

В результате ассемблирования получается так называемый «объектный файл». В качестве дополнительной возможности ассемблер может создать файл листинга программы.

Обычно для получения файлов объектного кода необходимо выполнить соответствующую программу ассемблера (программы MASM.EXE и ML.EXE фирмы Microsoft и TASM.EXE или TASM32.EXE фирмы Borland), указав в командной строке имя файла с текстом программы.

**Компоновка программы**

Следующая стадия (.obj -> .exe/.dll/.com) называется линковкой или компоновкой и служит для замещения символьных имен, используемых программистом, на реальные адреса.

Сравните шестнадцатеричное содержимое OBJ и EXE файла, который у вас получился. В EXE-файле присутствует та же последовательность байтов, что и в OBJ-файле. Но помимо этого еще присутствует: имя ассемблированного файла, версия ассемблера, «имя собственное» сегмента и так далее.

Это «служебная» информация, предназначенная для тех случаев, когда ваш исполнимый файл вы хотите собрать из нескольких. При разработке больших приложений исходный текст состоит, как правило, из нескольких модулей (файлов с исходными текстами), потому что хранить все тексты в одном файле неудобно – в них сложно ориентироваться. Каждый модуль по раздельности компилируется в отдельный файл с объектным кодом. В каждом из этих файлов прописаны свои сегменты кода/данных/стека, которые затем надо объединить в одно целое. А исполнимый файл нам нужно получить только один – с единым сегментом кода/данных/стека. Именно это LINK и делает: завершает определение адресных ссылок и объединяет, если это требуется, несколько программных модулей в один. И этот один у нас и является исполнимым.

Процесс объединения объектных модулей в один файл осуществляется специальной программой-компоновщиком или сборщиком (программа LINK.EXE фирмы Microsoft и TLINK.EXE фирмы Borland), которая выполняет связывание объектных модулей и машинного кода стандартных функций, находя их в библиотеках, и формирует на выходе работоспособное приложение – исполнимый код для конкретной платформы.

**Загрузка программы**

Компонент операционной системы, называемый загрузчиком (loader), считывает данные из исполняемого файла, загружает программу в память и передает управление по адресу точки входа. В результате программа начинает выполняться.

В тех случаях, когда при написании новой программы на языке ассемблера требуются лишь незначительные изменения машинных кодов, иногда быстрее и удобнее внести изменения непосредственно в объектный файл, а не проходить всю цепочку редактирования исходной программы и осуществлять ее повторную трансляцию с внесенными изменениями. Для этого существуют специальные шестнадцатеричные редакторы (типа Hacker Viewer), которые позволяют рассматривать файлы с бинарным (машинным) кодом в виде последовательности ассемблерных команд. Эту же технологию применяют в тех случаях, если исходный текст программы не доступен (взлом программы).

**Отладка программы**

До тех пор, пока Вы не наберетесь достаточного опыта в программировании на языке ассемблера, за исключением учебных, тривиальных программ, в отладке будет нуждаться любая ваша программа. Для этого используются различные отладчики (CodeView фирмы Microsoft, Turbo Debugger фирмы Borland, SoftIce фирмы NuMega, OllyDebug написанный Olech Yuschuk), позволяющие в процессе выполнения программы контролировать значения регистров или переменных, при необходимости изменять их. Можно просматривать содержимое различных участков памяти. Можно выполнять программы по шагам или расставить точки останова (BREAK POINTS), которые вызовут прекращение работы программы и переход управления к отладчику.

**3) Анализ исходных модулей:**

1 модуль: mlab1.asm – является основным файлом. Данный файл осуществляет запрос данных у пользователя, а именно ФИО. Затем при нажатии кнопки Enter, появляется сообщение в виде титульник, а также сообщение об увеличении или уменьшение скорости вывода данного сообщения или выходе из программы. В данном файле подключаются модуль 2 и модуль 3.

2 модуль: mlab1.inc – в данном файле содержатся все константы, который используются в модули 1, 3, 4. Данные константы содержат, как какие-то данные о числах, так и прерывания.

3 модуль: mlab1.mac – в нем находятся макросы. Если оценить 1 модуль, то можно заметить, что половина файла mlab1.asm состоит из вызова данных макросов. В основном в данном файле функция всех макросов заключается в выводе строк, а также завершения программы.

4 модуль: mlab1l.asm – содержит процедуры. К примеру, процедура Getch, отвечающая за ввод символа с клавиатуры. Так же есть подпрограммы вывода символа – это нужно во время уменьшения скорости вывода титульника. Есть подпрограммы вывода строк. Присутствуют подпрограммы преобразования <EDX,EAX> в беззнаковое десятичное, размещаемое по адресу DI.

**4) Выполнение макрообработки:**

Макроопределение – специальным образом оформленная группа команд, к которой в тексте программы можно обратиться с помощью некоторого символического имени – макрокоманды. (макроопределение может иметь формальные параметры, тогда макрокоманда должна иметь фактические).

Макрорасширение – некоторая последовательность инструкций ассемблера, которую формирует макроассемблер на этапе трансляции, путем замены формальных параметров макроопределения фактическими параметрами макрокоманды. Полученное макрорасширение вставляется в текст программы вместо макрокоманды.

Повторяющиеся действия (фрагменты) в программе можно описать и как процедуру, и как макроопределение. При этом в обоих случаях повторяющийся участок кода описан только один раз, а обращаемся к нему с помощью одной команды (вызов процедуры или макрокоманда).

После трансляции процедура так и останется описанной один раз, а тело макроопределения подставится во все места вызова и тем самым увеличит размер программы.

Вывод 1. Применение процедур делает код более компактным, т.е. экономим память

Но при обращении к процедуре

а) выполняется засылка параметров в регистры или стек,

б) запоминается адрес возврата

в) осуществляется переход,

г) по окончании работы процедуры восстанавливается адрес возврата,

д) очищаются регистры или стек и т.п.

Итак, при работе процедуры тратится время на переходы и передачу параметров во время выполнения программы.

При замене макрокоманд на макрорасширения тоже тратится время, но это происходит на этапе трансляции, а не во время выполнения программы.

Вывод 2. Применение макросредств экономит время выполнения программы.

Поэтому в программах критических по времени следует применять макросредства, а если необходимо экономить память следует применять процедуры.

Если в повторяющемся участке кода много команд (т.е. большой фрагмент) лучше описать его как процедуру. Если же небольшую группу команд описать процедурой, то число вспомогательных команд по ее вызову и передаче параметров станет сравнимым с числом команд самой процедуры, ее время выполнения станет на много больше.

Вывод 3. Большие участки кода рекомендуется описывать как процедуры, а маленькие - как макроопределения.

Еще одно отличие использования макросредств и процедур заключается в том, что параметрами процедур могут быть только операнды команд, а параметрами макрокоманд могут быть любые последовательности символов, в том числе и сами команды.

**5) Ассемблирование mlab1.asm:**

В результате ассемблирования получается так называемый «объектный файл». В качестве дополнительной возможности ассемблер может создать файл листинга программы.

Ассемблируем с помощью команды: tasm название\_файла.asm. В нашем случае пишем: mlab1.asm

Если синтаксических ошибок в файле mlab1.asm, нет, тогда будет создан файл mlab1.obj.

Если открыть файл mlab1.obj можно заметить непонятные символы, которые будет невозможно прочитать.

Что из себя представляет файл mlab1.asm – к данному файлу подключаются 2 файла, в которых находятся макросы и константы. После чего идет сегмент данных data, где описаны переменные, которые хранят строки для вывода титульного листа. Далее сегмент code начинается с описания макроса заполнения строки LINE, после начинается основной этап, где происходит запрос у пользователя входной строки в виде: ФИО. Затем вызываются макросы, как раз для вывода всех строк заставки. В конце выводится сообщение о количестве тактов и том как увеличить или уменьшить скорость вывода заставки, либо завершении программы. При нажатии соответствующих кнопок программа начнет переход по меткам, в которых начинается запуск макросов, либо переход по меткам, либо выход из программы по метке @@E.

**6) Трансляция модуля библиотечных функций mlab1l.asm**

После применения команды, т.е. ассемблирования данного файла получим файл mlab1l.obj

В файле mlab1l.asm – описаны процедуры. Процедура PUTSS отвечает за вывода на экран строки, адресуемой SI, с задержкой времени между символами в <CX,DX> mcs. Подпрограмма PUTC выводил символ в консоль. GETCH – за счет данной процедуры будет возможен ввод символа в консоли. Подпрограмма DILAY осуществляет задержки выполнения программы на <CX,DX> микросекунд. Подпрограмма SLEN подсчитывает число символов в строке, адресуемой SI. GETS - подпрограмма ввода строки в буфер, адресуемый DX. UTOA10 - Подпрограмма преобразования <EDX,EAX> в беззнаковое десятичное, размещаемое по адресу DI.

**7) Компоновка модулей mlab1.obj и mlab1l.obj**

Компоновка служит для замещения символьных имен, используемых программистом, на реальные адреса.

Процесс объединения объектных модулей в один файл осуществляется специальной программой-компоновщиком или сборщиком (программа LINK.EXE фирмы Microsoft и TLINK.EXE фирмы Borland), которая выполняет связывание объектных модулей и машинного кода стандартных функций, находя их в библиотеках, и формирует на выходе работоспособное приложение – исполнимый код для конкретной платформы.

Исполнимый код – это законченная программа с расширением COM, DLL или EXE, которую можно запустить на компьютере с установленной операционной системой, для которой эта программа создавалась. Имя исполняемого файла задастся именем первого .OBJ файла

Как говорится выше, имя исполняемого файла будет задаваться по имени первому файлу. Прописав команду tlink mlab1.obj+mlab1l.obj мы получим файл типа exe, а именно: mlab1.exe. Теперь мы можем запустить данный файл на своей машине.

**8) Запуск программы**

При запуске программы видим следующее:

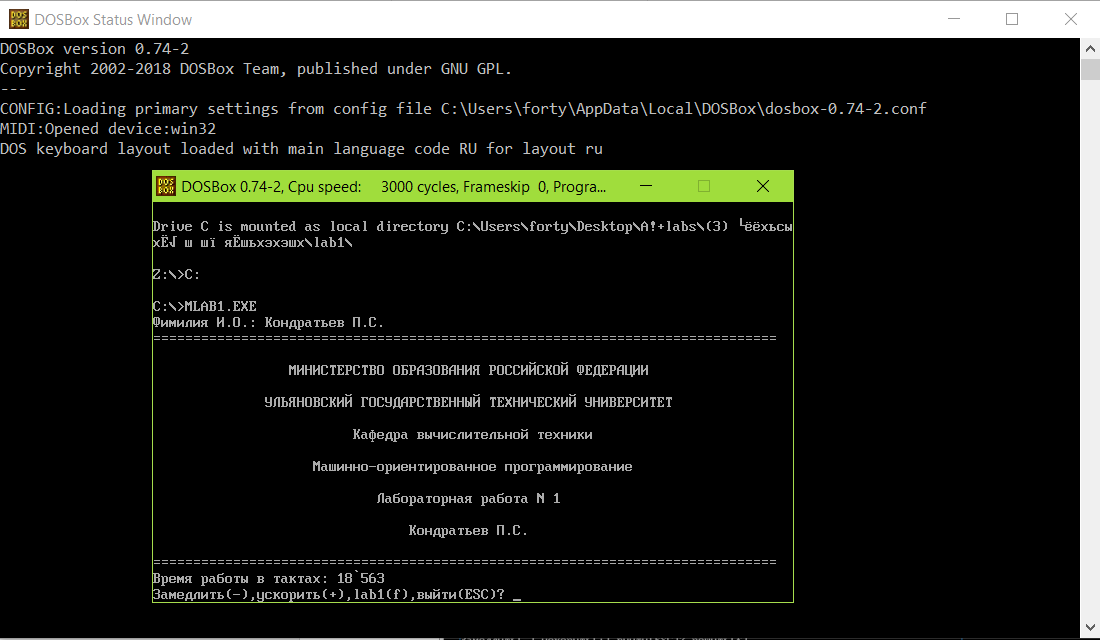
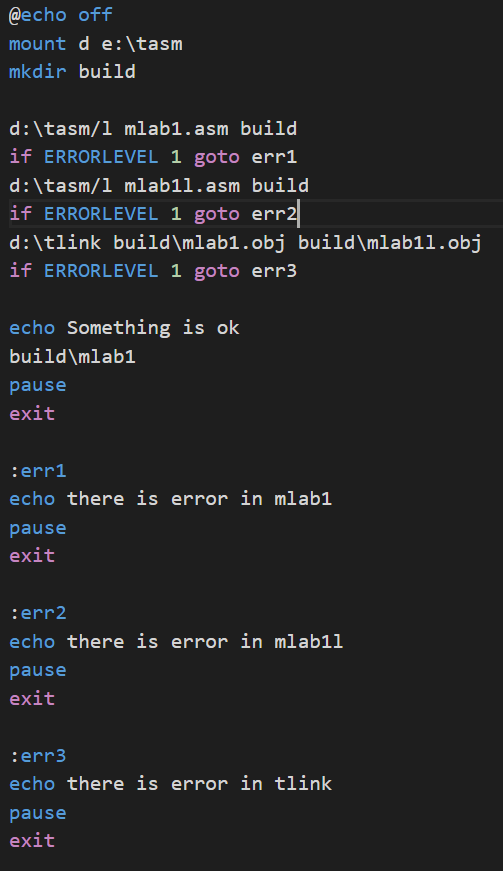


Рис. 7.1. Работа программы

**9) Разработка bat-файла mlab1.bat**

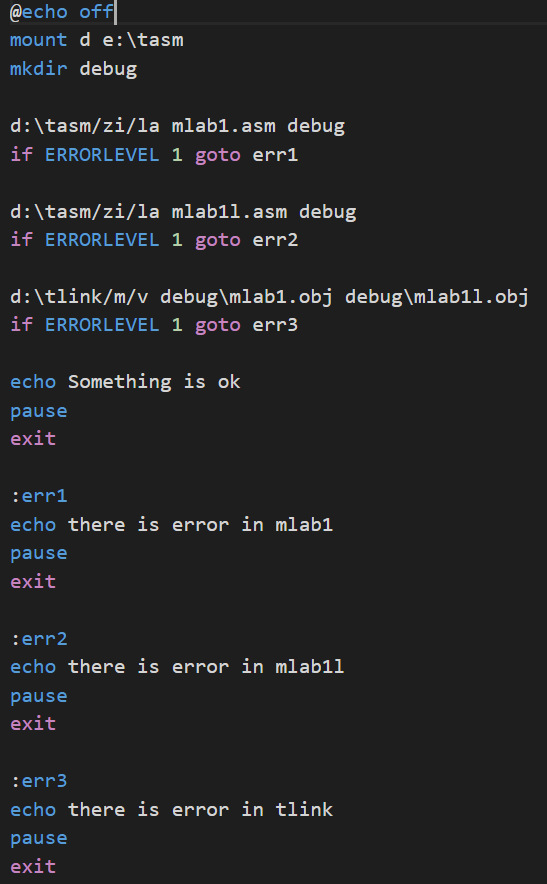
Для разработки bat – файла было создана 2 отдельных файла с расширением bat, в первом были прописаны следующие команды:

Build.bat



Во втором:

Debug.bat



В итоге у нас пройдет весь алгоритм, который мы писали вручную.

**10) Сравнительная таблица**

|  |  |  |
| --- | --- | --- |
| Название файла | Без ключей | С ключами |
| mlab1l.obj | 1 кб. | 2 кб. |
| mlab1.obj | 3 кб. | 4 кб. |
| mlab1.exe | 3 кб. | 5 кб. |

Из таблицы видно, что при запуске bat-файла объем файлов mlab1.obj и mlab1.exe увеличивается, хотя количество строк кода не изменяется, изменения лишь касаются в способе запуска данного проекта. При ассемблировании файла mlab1l.asm объем файла mlab1l.obj тоже увеличился.

**11) Модификация программы:**

Изначально была добавлена проверка с помощью операции cmp, если нажата клавиша “f”, то переходить по соответствующей метке, которая отвечает за вывод 20-битного двоичного числа, расчёта F, расчёта Z.

.MODEL SMALL

.STACK 200h

.386

INCLUDE MLAB1.INC

INCLUDE MLAB1.MAC

.DATA

f11 db "f11 = x1 !x2 x3 | x2 !x3 x4 | x3 !x4 | x1 x4 | !x2 !x3.", 0FFh

z db "Z = f11? X / 8 + Y \* 8: X \* 2 - Y \* 8; z7 = z8; z11 &= z14; z18 |= z10;", 0FFh

SpecialForX dd ?

SpecialForY dd ?

InX db "X: ",0FFh

InY db "Y: ",0FFh

f11\_1 db "f11 = 1",0FFh

f11\_0 db "f11 = 0",0FFh

beforeZ11 DB "Z11 before: ",0FFh

afterZ11 DB "Z11 after: ",0FFh

SLINE DB 78 DUP (CHSEP), 0

REQ DB "”Ё¬Ё«Ёп €.Ћ.: ",0FFh

MINIS DB "Њ€Ќ€‘’…ђ‘’‚Ћ ЋЃђЂ‡Ћ‚ЂЌ€џ ђЋ‘‘€‰‘ЉЋ‰ ”…„…ђЂ–€€",0

ULSTU DB "“‹њџЌЋ‚‘Љ€‰ ѓЋ‘“„Ђђ‘’‚…ЌЌ›‰ ’…•Ќ€—…‘Љ€‰ “Ќ€‚…ђ‘€’…’",0

DEPT DB "Љ дҐ¤а  ўлзЁб«ЁвҐ«м­®© вҐе­ЁЄЁ",0

MOP DB "Њ иЁ­­®-®аЁҐ­вЁа®ў ­­®Ґ Їа®Ја ¬¬Ёа®ў ­ЁҐ",0

LABR DB "‹ Ў®а в®а­ п а Ў®в  N 1",0

REQ1 DB "‡ ¬Ґ¤«Ёвм(-),гбЄ®аЁвм(+),lab1(f),ўл©вЁ(ESC)? ",0FFh

TACTS DB "‚аҐ¬п а Ў®вл ў в Єв е: ",0FFh

EMPTYS DB 0

BUFLEN = 70

BUF DB BUFLEN

LENS DB ?

SNAME DB BUFLEN DUP (0)

PAUSE DW 0, 0 ; ¬« ¤иҐҐ Ё бв аиҐҐ б«®ў  § ¤Ґа¦ЄЁ ЇаЁ ўлў®¤Ґ бва®ЄЁ

TI DB LENNUM+LENNUM/2 DUP(?), 0 ; бва®Є  ўлў®¤  зЁб«  в Єв®ў

; § Ї б ¤«п а §¤Ґ«ЁвҐ«м­ле "`"

.CODE

BEGIN LABEL NEAR

MOV AX, @DATA

MOV DS, AX

MOV PAUSE, PAUSE\_L

MOV PAUSE+2,PAUSE\_H

PUTLS REQ

LEA DX, BUF

CALL GETS

@@L:

FIXTIME

PUTL EMPTYS

PUTL SLINE

PUTL EMPTYS

PUTLSC MINIS

PUTL EMPTYS

PUTLSC ULSTU

PUTL EMPTYS

PUTLSC DEPT

PUTL EMPTYS

PUTLSC MOP

PUTL EMPTYS

PUTLSC LABR

PUTL EMPTYS

PUTLSC SNAME

PUTL EMPTYS

PUTL SLINE

DURAT

LEA DI, TI

CALL UTOA10

PUTL TACTS

PUTL TI

PUTL REQ1

CALL GETCH

CMP AL, '-

JNE CMINUS

INC PAUSE+2

JMP @@L

CMINUS: CMP AL, '+

JNE Lab

CMP WORD PTR PAUSE+2, 0

JE BACK

DEC PAUSE+2

Lab: CMP AL, 'f'

JNE CEXIT

PUTL EMPTYS

PUTL EMPTYS

PUTLS f11

PUTL EMPTYS

PUTLS z

PUTL EMPTYS

PUTL EMPTYS

PUTLS InX

NEXTX:

MOV AH, 01H

INT 21H

CMP AL, 0Dh

JE GETOUTX

SUB AL, 30H

CMP AL, 0

JE LX

ADD BL, 1

LX:

SHL EBX, 1

JMP NEXTX

GETOUTX:

SHR EBX, 1

PUTL EMPTYS

PUTLS InY

NEXTY:

MOV AH, 01H

INT 21H

CMP AL, 0Dh

JE GETOUTY

SUB AL, 30H

CMP AL, 0

JE LY

ADD CL, 1

LY:

SHL ECX, 1

JMP NEXTY

GETOUTY:

SHR ECX, 1

f11part1:

mov eax, 1b

and eax, ebx

cmp eax, 1

jne f11part2

mov eax, 10b

and eax, ebx

cmp eax, 0

jne f11part2

mov eax, 100b

and eax, ebx

cmp eax, 1

je ifOne

f11part2:

mov eax, 10b

and eax, ebx

cmp eax, 1

jne f11part3

mov eax, 100b

and eax, ebx

cmp eax, 0

jne f11part3

mov eax, 1000b

and eax, ebx

cmp eax, 1

je ifOne

f11part3:

mov eax, 100b

and eax, ebx

cmp eax, 1

jne f11part4

mov eax, 1000b

and eax, ebx

cmp eax, 0

je ifOne

f11part4:

mov eax, 1b

and eax, ebx

cmp eax, 1

jne f11part5

mov eax, 1000b

and eax, ebx

cmp eax, 1

je ifOne

f11part5:

mov eax, 10b

and eax, ebx

cmp eax, 0

jne f11Zero

mov eax, 100b

and eax, ebx

cmp eax, 0

je ifOne

f11Zero:

push ECX

PUTL EMPTYS

PUTLS f11\_0

PUTL EMPTYS

pop ECX

SHL EBX, 1

SHL ECX, 3

SUB EBX, ECX

JMP out0

ifOne:

push ECX

PUTL EMPTYS

PUTLS f11\_1

PUTL EMPTYS

pop ECX

SHR EBX, 3

SHL ECX, 3

ADD EBX, ECX

out0:

push ECX

PUTL EMPTYS

PUTLS beforeZ11

pop ECX

push eax

push ebx

mov cl, 12

shl ebx, cl

mov cx, 20

cycle1:

shl ebx, 1

jc l1

mov dl, '0'

jmp l2

l1:

mov dl, '1'

l2:

mov ah, 02h

int 21h

dec cx

cmp cx, 0

jne cycle1

pop ebx

pop eax

mov eax, 10000000b

and eax, ebx

cmp eax, 0

jne K1

btr ebx, 6

jmp KNEXT1

K1:

bts ebx, 6

KNEXT1:

mov eax, 10000000000000b

and eax, ebx

cmp eax, 1

jne K2

mov eax, 10000000000b

and eax, ebx

cmp eax, 1

jne K2

bts ebx, 11

jmp KNEXT2

K2:

btr ebx, 11

KNEXT2:

mov eax, 1000000000b

and eax, ebx

cmp eax, 0

jne K3

mov eax, 100000000000000000b

and eax, ebx

cmp eax, 0

jne K3

btr ebx, 18

jmp KOUT

K3:

bts ebx, 18

KOUT:

push ECX

PUTL EMPTYS

PUTL afterZ11

pop ECX

shl ebx, 12

mov cx, 20

cycle2:

shl ebx, 1

jc l3

mov dl, '0'

jmp l4

l3:

mov dl, '1'

l4:

mov ah, 02h

int 21h

dec cx

cmp cx, 0

jne cycle2

PUTL EMPTYS

BACK: JMP @@L

CEXIT: CMP AL, CHESC

JE @@E

TEST AL, AL

JNE BACK

CALL GETCH

JMP @@L

; ‚ле®¤ Ё§ Їа®Ја ¬¬л

@@E: EXIT

EXTRN PUTSS: NEAR

EXTRN PUTC: NEAR

EXTRN GETCH: NEAR

EXTRN GETS: NEAR

EXTRN SLEN: NEAR

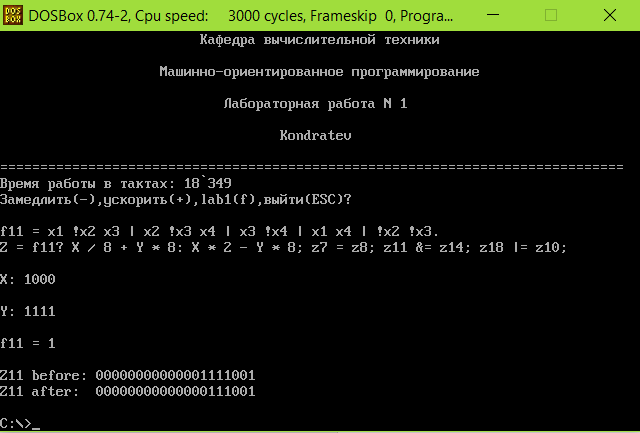
EXTRN UTOA10: NEAR

EXTRN Lab1: NEAR

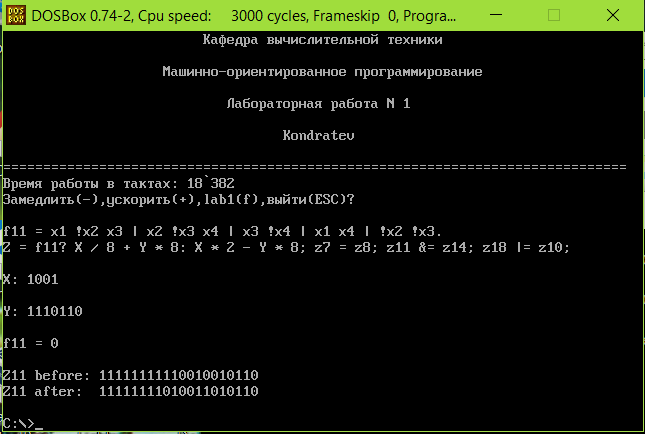
END BEGIN

**12) Примеры работы программы:**

При значениях x = 1000, y = 1111



При значениях x = 1001, y = 1110110



**Вывод:**

В ходе выполнения данной лабораторной работы, мне удалось освоить базовую разработку программ на assembler. Я научился создавать bat-файлы, с помощью которых можно сократить время работы при ручном написании команд в DOSbox. Получил базовые навыки работы со строками. Разобрался с отличиями макросов от процедур. Научился встраивать свой код в чужую программу.

**Список литературы:**

* Вострикова З. П. Программирование на языке ассемблера ЕС ЭВМ. М.: Наука, 1985.
* Калашников О. А. Ассемблер? Это просто! Учимся программировать. — [БХВ-Петербург](https://ru.wikipedia.org/w/index.php?title=%D0%91%D0%A5%D0%92-%D0%9F%D0%B5%D1%82%D0%B5%D1%80%D0%B1%D1%83%D1%80%D0%B3&action=edit&redlink=1), 2011.
* Юров В., Хорошенко С. Assembler: учебный курс. — СПб.: [Питер](https://ru.wikipedia.org/wiki/%D0%9F%D0%B8%D1%82%D0%B5%D1%80_(%D0%B8%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE))