Системное программирование

Лекция №3

Работа в Ассемблере TASM 5.0

Жизненный цикл программы

1. Постановка и формулировка задачи:

- изучение предметной области и сбор материала в проблемно-ориентированном контексте;
- определение назначения программы, выработка требований к ней и представление требований, если возможно, в формализованном виде;
- формулирование требований к представлению исходных данных и выходных результатов;
- определение структур входных и выходных данных;
- формирование ограничений и допущений на исходные и выходные данные.

Жизненный цикл программы

2. Этап *проектирования:*

- формирование ≪ассемблерной≫ модели задачи;
- выбор метода реализации задачи;
- разработка алгоритма реализации задачи;
- разработка структуры программы в соответствии с выбранной моделью памяти.

3. Этап *кодирования:*

- уточнение структуры входных и выходных данных и определение ассемблерного формата их представления;
- программирование задачи;
- комментирование текста и составление предварительного описания программы.

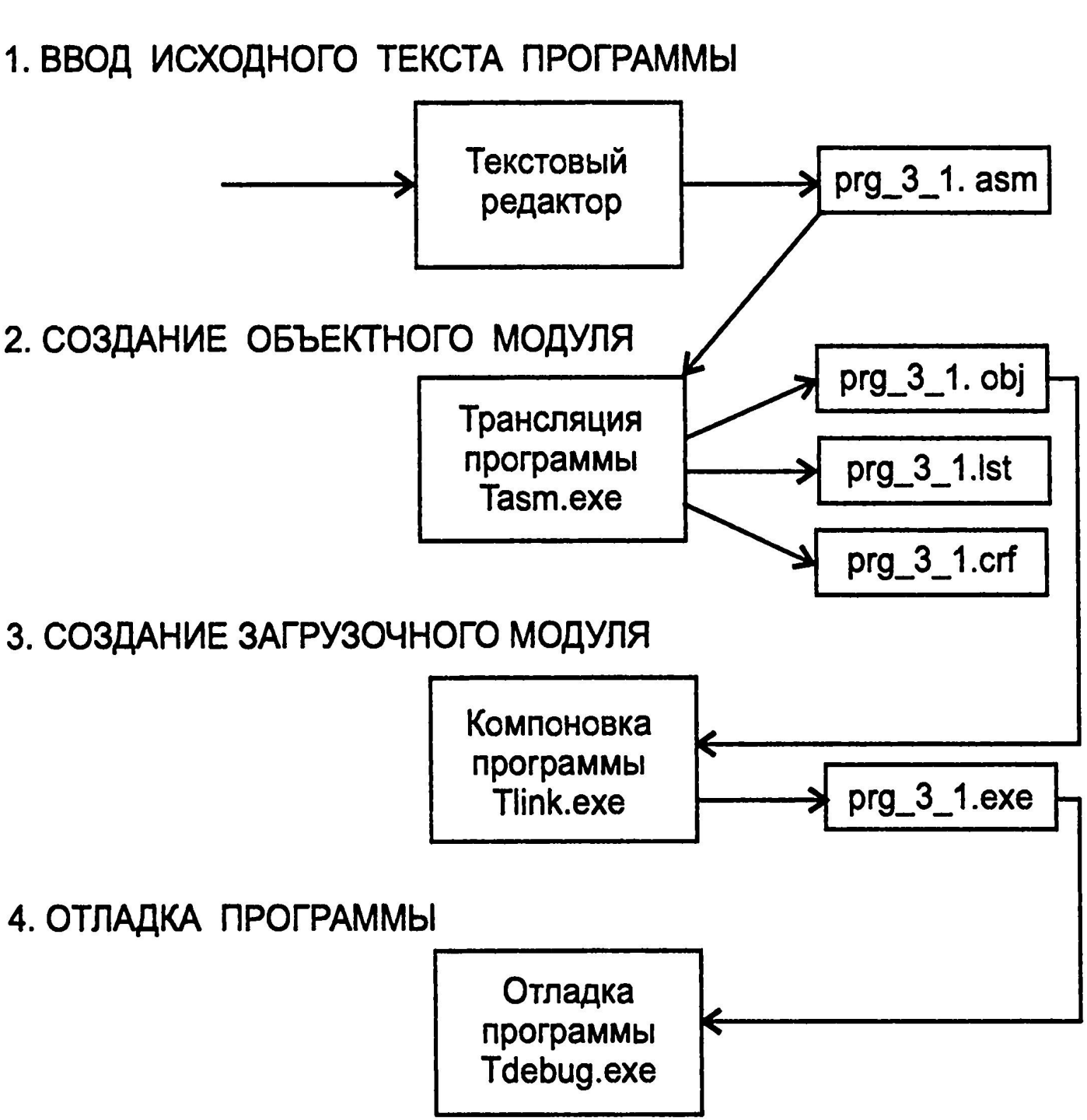
Жизненный цикл программы

4. Этап *отладки и тестирования:*

- составление тестов для проверки работоспособности программы;
- обнаружение, локализация и устранение в программе ошибок, выявленных в тестах;
- корректировка кода программы и ее описания.

5. Этап *эксплуатации и сопровождения:*

- настройка программы на конкретные условия использования;
- обучение пользователей работе с программой;
- организация сбора сведений о сбоях в работе программы, ошибках в выходных данных, пожеланиях по улучшению интерфейса и удобства работы с программой;
- модификация программы с целью устранения выявленных ошибок и, при необходимости, изменения ее функциональных возможностей.



Процесс разработки программы на ассемблере

Трансляция программы

Решаются задачи:

- перевод команд ассемблера в соответствующие машинные команды;
- построение таблицы символов;
- расширение макросов;
- формирование файла листинга и объектного модуля.

Программа, которая реализует эти задачи, называется *ассемблером.* Итог работы ассемблера — файл объектного модуля и файл листинга.

Объектный модуль включает в себя представление исходной программы в машинных кодах и некоторую другую информацию, необходимую для отладки и компоновки его с другими модулями.

Файл листинга содержит код ассемблера исходной программы. Для каждой команды ассемблера указываются ее машинный (объектный) код и смещение в кодовом сегменте. В конце листинга TASM формирует таблицы с информацией о метках и сегментах, используемых в программе, а также сообщения об ошибках.

Трансляция программы

Формат командной строки для запуска tasm.exe:

TASM [опции] имя_исходного_файла [, имя_объектного_файла] [,имя_файла_листинга] [, имя_перекрестных _ссылок]

- если имена объектного файла, файла листинга и файла перекрестных ссылок должны совпадать с именем исходного файла, то нужно просто поставить запятые вместо имен этих файлов:

  tasm.exe prg_6_l , , ,

- если имена объектного файла, файла листинга и/или файла перекрестных ссылок не должны совпадать с именем исходного файла, то нужно в командной строке указать имена нужных файлов в соответствующем порядке.
- если требуется выборочное создание файлов, то вместо ненужных файлов необходимо подставить параметр nul

Трансляция программы

Формат строки в файле листинга:

<глубина_вложенности> <номер_строки> <смещение>
<машинный_код> <исходный_код>

- *глубина_вложенности* — уровень вложенности включаемых файлов или макрокоманд в файле.
- *номер_строки* — номер строки в файле листинга. Номера строк листинга могут не соответствовать номерам строк в исходном файле.
- *смещение* — смещение в байтах текущей команды относительно начала сегмента кода. Это смещение называют также счетчиком адреса. Величину смещения вычисляет транслятор для адресации в сегменте кода.
- *машинный_код* — машинное представление команды ассемблера, представленной далее в этой строке полем исходный_код.
- *исходный_код* — строка кода из исходного файла.

Prg_3_1.asm

```
1     ;----Prg_3_1.asm----------------
2     ;Программа преобразования двузначного шестнадцатеричного числа
3     ;в символьном виде в двоичное представление.
4     ;Вход: исходное шестнадцатеричное число из двух цифр,
5     ;вводится с клавиатуры.
6     ;Выход: результат преобразования должен
7     ;быть в регистре al.
8     ;-----------------------------------------------
9     0000      data segment para public "data" ; сегмент данных
10    0000      82 A2 A5 A4 A8 E2 A5+ message db "Введите две шестнадцатеричные
                цифры,$"
11    20 A4 A2 A5 20 E8     A5+
12    E1 E2 AD AO A4 E6     A0+
13    E2 A5 E0 A8 E7 AD     EB+
14    A5 20 E6 A8 E4 EO     EB+
15    2C 24
16    0025      data ends
17    0000      stk   segment stack
18    0000 0100*(3F)    db 256 dup ("?") ;сегмент стека
19    0100      stк ends
20    0000      code segment para public "code" ; начало сегмента кода
21    0000      main proc      ; начало процедуры main
22    assume cs: code, ds: data, ss:stk
23    0000      B8 0000 mov       ax,data         ; адрес сегмента данных в регистр ax
24    0003 8E 08        mov       ds,ax           ;ax в ds
```

```
25   0005 B4 09          mov       ah, 9
26   0007 BA 0000        mov dx, offset messag
**Error** Prg_3_1.asm(21) Undefined symbol: MESSAG
27   000A CD 21          int 21h
28   000C 33 C0          xor       ax, ax          ; очистить регистр ax
29   000E B4 01          mov       ah,1h           ;1h в регистр ah
30   0010 CD 21          int       21h             ; генерация прерывания с номером 21h
31   0012 8A D0          mov       dl.al           ; содержимое регистра al в регистр dl
32   0014 80 EA 30       sub       dl, 30h         ; вычитание: (dl)=(dl)-30h
33   0017 80 FA 09       cmp       dl,9h           ;сравнить (dl) с 9h
34   001A 7E E4           jle      MM              ; перейти на метку M1 если dl<9h или dl=9h
**Error** Prg_3_1.asm(29) Undefined symbol: MM
35   001C 80 EA 00       sub       dl,777h         ;вычитание: (dl)=(dl)-7h
**Error** Prg_3_1.asm(30) Constant too large
36   001F      M1:               определение метки M1
37   001F      B1 04     mov       cl,4h           ;пересылка в регистр c1
38   0021      D2 E2     shl       dl, cl          ;сдвиг содержимого dl на 4 разряда
39   0023      CD 21     int       21h             ;вызов прерывания с номером 21h
40   0025      2C 30     sub       al,30h          ;вычитание: (dl)=(dl)-30h
41   0027      3C 09     cmp       al,9h           ;сравнить (a1) с 9h
42   0029      7E 02     jle       M2              ;перейти на метку M2 если a1<9Ь или a1=9Ь
43   002B      2C 07     sub       al,7h           ;вычитание: (a1)=(a1)-711
44   002D      M2:               определение метки M2
45   002D      02 D0     add       dl, al          ;сложение: (c!1 )=(c!1)+(a1)
46   002F      B8 4C00 mov       ax,4c00h        ; пересылка 4c00h в регистр ax
47   0032      CD 21     int       21h             ;вызов прерывания с номером 21h
48   0034      main      endp              конец процедуры main
49   0034      code      ends              конец сегмента кода
50   end main                    ; конец программы с точкой входа main
```

Turbo Assembler Version 4.1 02/03/98 21:23:43 Page 2
Symbol Table

Type Value Cref (defined at #)

??DATE Text "02/03/98"
??FILENAME Text "Prg_3"
??TIME Text "21:23:43"
??VERSI0N Number 040A
@CPU Text 0101H
@CURSEG Text CODE #9 #17 #20
@FILENAME Text PRG_3_1
@W0RDSIZE Text 2 #9 #17 #20
M1 Near CODE: 001F #36
M2 Near C0DE:002D 42 #44
MAIN Near CODE: 0000 #21 50
MESSAGE Byte DATA:0000 #10

Groups & Segments Bit Size Align Combine Class Cref (defined at #)
CODE 16 0034 Para Public CODE #20 22
DATA 16 0025 Para Public DATA #9 22 23
STK16 0100 Para Stack 17 22

Turbo Assembler Version 4.1 02/03/98 21:23:43 Page 3

Error Summary
Error PrgJ3_1.asm(21) Undefined symbol: MESSAG
Error Prg_3_1.asm(29) Undefined symbol: MM
Error PrgJ3_1.asm(30) Constant too large

Компоновка программы

Цель этапа — преобразование кода и данных в объектных файлах в их *перемещаемое выполняемое отображение*.

Формат командной строки для запуска компоновщика:

TLINK [опции] список_объектных_файлов [, имя_загрузочного_модуля] [,имя_файла_карты] [, имя_файла_библиотеки] [,имя_файла_определений] [, имя_ресурсного_файла]

список_объектных_файлов — список компонуемых файлов с расширением .obj. Файлы разделяются пробелами или знаком + .

имя_загрузочного_модуля — если не указано, то имя загрузочного модуля будет совпадать с первым именем в списке имен объектных файлов.

имя_файла_карты — параметр для создания специального файла с картой загрузки (перечисляются имена, адреса загрузки и размеры всех сегментов, входящих в программу).

имя_файла_библиотеки — путь к файлу библиотеки (.lib). Этот файл создается и обслуживается специальной утилитой tlib.exe

имя_файла_определений — путь к файлу определений (.def). Файл используется при компоновке Windows-приложений

имя_ресурсного_файла — путь к файлу с ресурсами Windows-приложения (.res).

Отладка программы

Цель этапа — проверка правильности функционирования как отдельных фрагментов кода, так и программы в целом в соответствии с алгоритмом. Включает в себя *тестирование* программы, т.е. проверку ее работы на «пограничных» и заведомо некорректных исходных данных. Специфика программ на ассемблере состоит в том, что они интенсивно работают с аппаратными ресурсами компьютера. Это обстоятельство заставляет программиста постоянно отслеживать содержимое определенных регистров и областей памяти.

Типы отладчиков:

- *интегрированные* отладчики, реализованные в виде интегрированной среды, напоминающей среду для языков высокого уровня (Turbo Pascal, Visual C и т. д.);
- *автономные* отладчики, представляющие собой отдельные программы.

Turbo Debugger

Позволяет решить две главные задачи:

- определить место логической ошибки;
- определить причину логической ошибки.

Возможности TD:

- трассировка программы в прямом направлении, т.е. последовательное выполнение программы, при котором за один шаг выполняется одна машинная инструкция;
- трассировка программы в обратном направлении, т.е. выполнение программы по одной команде за один шаг, но в обратном направлении;
- просмотр и изменение состояния аппаратных ресурсов процессора во время трассировки.

Формат командной строки для запуска отладчика:

TD имя_исполняемого_модуля

Turbo Debugger

Правильная организация процесса получения исполняемого модуля, пригодного для отладки на уровне исходного текста.

В исходной программе обязательно должна быть определена метка для первой команды, с которой начнется выполнение программы. Такая метка может быть собственно меткой или именем процедуры. Имя этой метки обязательно должно быть указано в конце программы в качестве операнда директивы END: end имя_метки

Исходный модуль должен быть оттранслирован с ключом /zi: tasm /zi имя_исходного_модуля , , , Ключ разрешает транслятору сохранить связь символических имен в программе с их смещениями в сегменте кода, что позволяет выполнять отладку, используя оригинальные имена.

Редактирование модуля должно быть осуществлено с ключом /v: tlink /v имя_объектного_модуля Ключ указывает на необходимость сохранения отладочной информации в исполняемом файле.

Turbo Debugger

Режимы запуска программы в отладчике:

- безусловного выполнения;
- выполнения по шагам;
- выполнения до текущего положения курсора;
- выполнения с установкой точек прерывания.

Режим безусловного выполнения программы целесообразно применять, когда требуется посмотреть на общее поведение программы.

Режим выполнения программы до текущего положения курсора целесообразно использовать в том случае, если интерес представляет только правильность функционирования некоторого участка программы.

В *режиме выполнения программы с установкой точек прерывания* программа после запуска будет останавливаться в строго определенных *точках прерывания* (breakpoints). Перед выполнением программы необходимо установить эти точки.

Режим выполнения программы по шагам применяется для детального изучения ее работы. В этом режиме выполнение программы прерывается на каждой машинной (ассемблерной) команде. Для активизации режима нужно нажать клавишу F7 (Run > Trace into) или F8 (Run > Step over).

Turbo Debugger

Окно CPU - отражает состояние процессора и состоит из пяти подчиненных окон.

- В окне с исходной программой в дизассемблированном виде представлена та же самая программа, что и в окне Module, но уже в машинных кодах. Пошаговую отладку можно производить прямо в этом окне; строка с текущей командой подсвечивается.
- В окне регистров процессора (Registers) отражается текущее содержимое регистров.
- В окне флагов (Flags) отражается текущее состояние флагов процессора в соответствии с их мнемоническими названиями.
- В окне стека (Stack) отражается содержимое памяти, выделенной для стека. Адрес области стека определяется содержимым регистров SS и SP.
- Окно дампа оперативной памяти (Dump) отражает содержимое области памяти по адресу, который формируется из компонентов, указанных в левой части окна. В окне можно увидеть содержимое произвольной области памяти. Для этого нужно в контекстном меню выбрать нужную команду.

Ассемблер MASM корпорации Microsoft

Основные программы:

- masm.exe — ассемблер;
- ml.exe — ассемблер и компоновщик (Masm and Link);
- link.exe — компоновщик;
- cv.exe — отладчик (CodeView)

Командная строка ml.exe:

ml [ключи] исх_файл_1 [[ключи] исх_файл_2] . . . [/link ключи_link]

Ключи командной строки для ml.exe чувствительны к регистру.

Командная строка masm.exe:

masm [ключи] исх_файл [, [объектный_файл] [, [файл_листинга] [, [файл_перекрестных_ссылок]]]]

Компоновщик компонует (объединяет) объектные файлы и библиотеки в исполняемый файл или динамически компонуемую библиотеку (DLL).

Командная строка Link.exe:

link [ключи] объект_файлы [,[исполн_файл][,[файл_карты][,[файлы_6иблиотек] [, [def _ файл]]]]] [;]