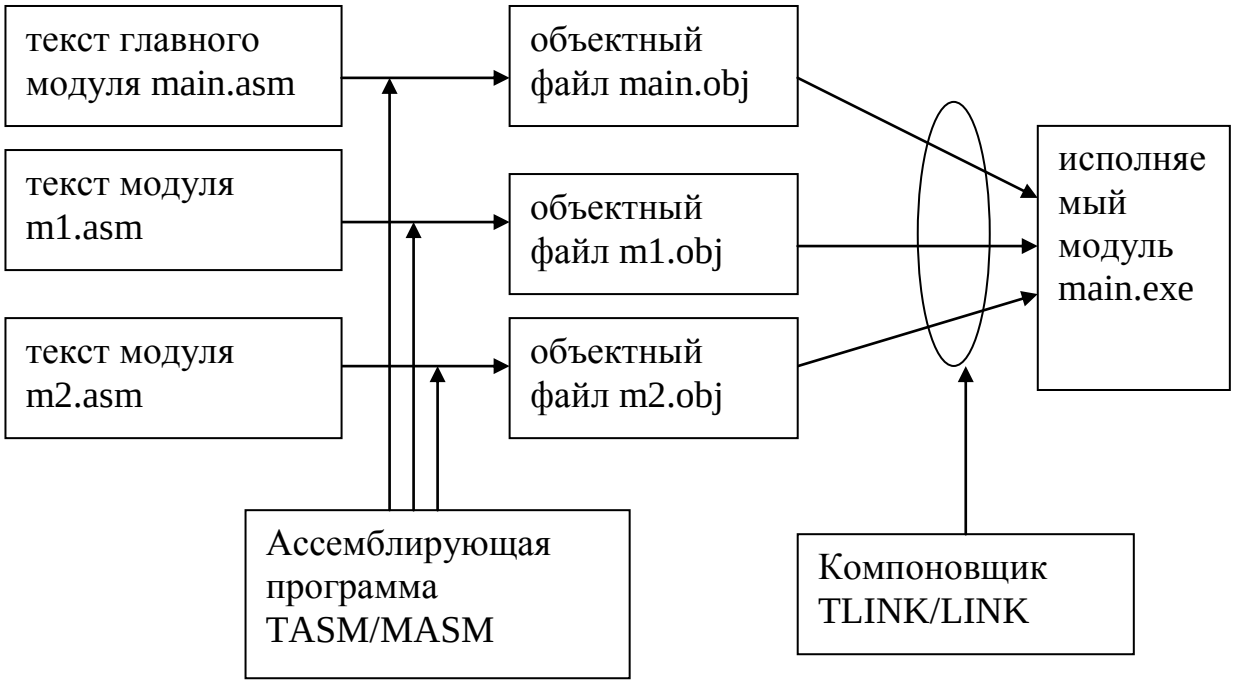
13. Многомодульные ассемблерные программы

Простейшие ассемблерные программы могут состоять только из одного модуля, содержащего описание используемых данных и необходимые команды (возможно – с использованием внутренних подпрограмм). Более сложные задачи могут потребовать разделения проекта на относительно самостоятельные единицы с возможностью независимой разработки. Модуль – это единица разработки, которая транслируется отдельно от других модулей с созданием собственного объектного модуля. В дальнейшем отдельные объектные модули объединяются программой-компоновщиком с созданием единого исполняемого модуля.



Каждый исходный модуль оформляется обычным образом, содержит описание данных и машинные команды и завершается директивой END.

Большинство имен, объявляемых в модуле, являются локальными, т.е. они используются только внутри данного модуля. Это дает возможность в разных модулях использовать одинаковые имена. Однако, практически

всегда модули должны взаимодействовать друг с другом и между ними возникают перекрестные связи. Эти связи делят на две группы:

- если в модуле используются данные, объявленные в другом модуле, то это связь по данным
- если в модуле вызываются подпрограммы, реализованные в других модулях, то это связь по управлению

Наличие перекрестных связей между модулями является основной проблемой при создании многомодульных программ. Поскольку модули транслируются независимо друг от друга, то использование “чужих” имен требует дополнительного описания в исходном тексте, чтобы ассемблирующая программа не выдавала ошибок при их обработке. Введем два важных понятия:

- внешнее имя – это имя, которое используется внутри модуля, но в нем не описывается, т.е. описывается где–то в другом модуле (импортируемые имена)
- общедоступное имя – это имя, которое описывается в данном модуле, используется в нем и может использоваться в других модулях (экспортируемые имена)

Оба эти понятия являются относительными, т.е. имеют смысл только для конкретного модуля и при переходе к другому модулю внешнее имя может стать общим, а общее внешним.

Оба типа имен надо явно указывать в тексте модуля с помощью специальных директив EXTRN (от External) и PUBLIC. Эти директивы записываются в самом начале модуля и в качестве операндов содержат списки общих или внешних имен, причем для внешних имен дополнительно указывается их тип.

Пусть в модуле M1 требуется объявить общедоступную переменную X и общедоступную константу K. Тогда начальный фрагмент текста модуля должен иметь следующий вид:

```
PUBLIC   X, K       ;  директива объявления общих имен
```

```
X   DB   1              ; стандартное объявление переменной
K   EQU  10             ; стандартное объявление константы
```

Если эти имена надо использовать в модуле M2, то их надо просто объявить как внешние и далее использовать в командах как свои собственные:

```
EXTRN   X:BYTE,  K:ABS    ; директива объявления внешних имен
        ……
        MOV X, 0          ; использование “чужой” переменной
        MOV AX, K         ; использование “чужой” константы
```

В этом примере в директиве EXTRN использованы спецификаторы внешних имен BYTE для объявления байтовой переменной и ABS для объявления константы. Кроме них можно использовать еще спецификаторы WORD и DWORD, назначение которых вполне очевидно.

Пусть теперь в модуле M2 требуется объявить подпрограмму с именем Prog1 и сделать это имя общим, т.е. разрешить обращаться к этой подпрограмме из других модулей. Фрагмент описания модуля M2:

```
        PUBLIC   Prog1  ; объявление внешнего имени
Prog1   PROC   FAR      ;  заголовок подпрограммы
        Тело подпрограммы
Prog1   ENDP
```

В модуле M1 эту подпрограмму можно вызывать обычным образом, если имя Prog1 объявить внешним:

```
        EXTRN    Prog1:NEAR   ; объявление внешнего имени
        ……
        CALL     Prog1        ; вызов внешней подпрограммы
        ……
```

Здесь в директиве EXTRN используется спецификатор NEAR, который определяет внешнее имя как связь по управлению, в данном случае – имя внешней подпрограммы. Кроме спецификатора NEAR есть еще спецификатор FAR, оба они “пришли” из старой сегментной модели организации памяти.

Сейчас описание обоих модулей можно собрать вместе.

Модуль M1:

```
        EXTRN   Prog1:NEAR   ;
        PUBLIC  X, K         ;
        …….
X       DB   1
K       EQU   10
        …….
        CALL    Prog1
        …….
        END
```

Модуль M2:

```
        EXTRN   X:BYTE,  K:ABS
        PUBLIC   Prog1
        …….
Prog1   PROC    FAR
        Тело подпрограммы
Prog1   ENDP
        …….
        MOV X, 0            ; использование “чужой” переменной
        MOV AX, K           ; использование “чужой” константы
        …….
        END
```

Директивы описания внешних и общих имен используются при ассемблировании исходных модулей и создании выходных объектных модулей. Сохраняемая в объектном файле информация о внешних и общих именах используется компоновщиком для обработки перекрестных связей

между модулями. Только компоновщик может установить окончательное соответствие между внешними и общими именами разных модулей.

Необходимость использования внешних имен возникает при разработке ассемблерных программ для операционной системы Windows. Как известно, для доступа к ресурсам вычислительной системы Windows предоставляет огромный (более 2000) набор системных функций под общим названием Win32 API. Эти функции могут использоваться и в ассемблерных программах, и в этом случае имена всех используемых функций должны быть объявлены в начале программы как внешние с помощью директивы EXTRN и спецификатора NEAR:

```
EXTRN    имя_API:NEAR
```

Если API-функция использует входные параметры, то они передаются через стек, причем сначала в стек заносится последний параметр, потом предпоследний, и т.д., т.е. на вершине будет первый параметр. Большинство параметров – двойные слова (4 байта), поэтому работа со стеком идет именно двойными словами.

В качестве примера рассмотрим простейшую консольную программу, которая лишь выводит текст в заголовок консольного окна. Для ее реализации необходимы 3 API-функции:

- AllocConsole для создания консоли (параметров нет)
- SetConsoleTitleA для вывода текста в заголовок окна консоли; имеет один параметр - адрес нуль-терминированной строки с текстом
- ExitProcess для завершения процесса; имеет один параметр – код завершения (0 если нормальное завершение)

Текст программы:

```
EXTRN    AllocConsole: NEAR
EXTRN    SetConsoleTitleA: NEAR
EXTRN    ExitProcess: NEAR
.DATA        ; начало данных с помощью специальной директивы
```

```
Title   DB    ‘Текст в заголовке’, 0  ; строка заканчивается нулевым байтом
        . CODE          ; начало кодового фрагмента
Main    PROC    NEAR          ; точка входа в программу
        CALL    AllocConsole
        LEA   EAX, Title   ; адрес строки – в EAX
        PUSH    EAX         ;  а потом – в стек
        CALL    SetConsoleTitleA
        PUSH    0       ;  в стек – нулевой код завершения
        CALL    ExitProcess
Main    ENDP
        END     Main
```

Больше возможностей по работе с консолью дают функции ReadConsole и WriteConsole. Первая обеспечивает простейший ввод текста с клавиатуры без анализа управляющих клавиш с занесением результата во внутренний буфер программы. Вторая функция выводит текст на экран.

Для использования этих функций надо сначала получить 2 дескриптора: стандартный дескриптор ввода и стандартный дескриптор вывода. Оба дескриптора получаются с помощью функции GetStdHandle с параметром-константой (-10) (для ввода) и (-11) (для вывода). Для сохранения дескрипторов надо зарезервировать память (два двойных слова).

Параметры функции ReadConsole:

- дескриптор ввода
- адрес буфера для сохранения вводимой строки
- размер буфера в байтах
- адрес области памяти для сохранения количества реально введенных символов
- резервный параметр со значением 0

Параметры функции WriteConsole:

- дескриптор вывода
- адрес буфера для хранения выводимой строки

- размер буфера в байтах
- адрес области памяти для сохранения количества реально выводимых символов
- резервный параметр со значением 0

Структура программы:

1. Объявление всех используемых функций как внешних
2. Объявление используемых данных

```
Bufer       DB   50   DUP (?)      ; буфер на 50 байтов
HandIn      DD    ?        ; здесь будет дескриптор входного потока
HandOut     DD   ?        ; а здесь – выходного
Len         DD   ?        ; здесь будет реальная длина введенной строки
```

3. Команды

```
PUSH   -10
CALL    GetStdHandle
MOV     HandIn,  EAX     ; сохраним дескриптор входного потока
PUSH    -11
CALL    GetStdHandle
MOV     HandOut, EAX     ; сохраним дескриптор выходного потока
PUSH    0 ;занесение в стек входных параметров функции ReadConsole
PUSH    OFFSET    Len      ; адрес области Len в стек
PUSH    50
PUSH    OFFSET    Bufer   ; адрес области Bufer в стек
PUSH    HandIn
CALL    ReadConsole
CMP     EAX , 0          ; проверить код завершения
JZ      Exit             ;  переходим на обработку неудачного ввода
PUSH    0
PUSH    OFFSET    Len
PUSH    Len
```

```
PUSH    OFFSET    Bufer
PUSH    HandOut
CALL    WriteConsole
```

Кроме трех перечисленных, для консольного ввода/вывода можно использовать еще около 50 функций, например:

- SetConsoleCursorPosition: установка позиции курсора для ввода или вывода
- SetConsoleTextAttribute: установка цветовых атрибутов текста и его фона
- ReadConsoleInput: низкоуровневый ввод с консоли с возможностью обработки управляющих клавиш и событий от мыши
- WriteConsoleOutput: низкоуровневый вывод в экранный буфер с возможностью использования нескольких буферов (окон)

Практические задания к теме №13.

Задание 1. Оформить выполненные ранее программы с подпрограммами в виде двухмодульных, вынеся во вспомогательный модуль текст подпрограммы.

Задание2. Выполнить простейшие консольные Windows-программы с помощью утилит tasm32, tlink32 и td32.