Базовое программное обеспечение процессора NM6403

Справочное руководство

Предварительная версия

АКЦИОНЕРНОЕ ОБЩЕСТВО
НАУЧНО-ТЕХНИЧЕСКИЙ ЦЕНТР

Оглавление

Предисловие

В предисловии описывается назначение и состав документа, приводит краткий обзор разделов и глав, определяет стиль и символьные нотации, используемые в данном документе.

Примечание

В данном справочном руководстве не содержится информация об устройстве процессора. Для получения данных о структуре NM6403 обращайтесь к другим источникам, в частности к документу "ПО процессора NeuroMatrix® NM6403. Описание языка ассемблера".

О справочном руководстве

Данное руководство содержит следующую информацию:

- состав базового программного обеспечения (БПО) процессора Neuro|Matrix® NM6403;
- справочная информация по каждой компоненте базового ПО;
- соглашения о передаче параметров при вызове функций;
- описания форматов файлов, используемых в БПО.

Как организован данный документ

Данный документ разбит на четыре части, каждая из которых освещает свой блок компонент, входящих в состав базового программного обеспечения процессора NM6403. Разбиение на части осуществляется следующим образом:

Часть 1 **Справочные руководства по компонентам**

Содержит справочную информацию о каждой компоненте БПО, используемой для получения исполняемого кода для процессора NM6403. Например, соглашения по расширениям файлов, опции командной строки, дополнительные утилиты для просмотра объектных и исполняемых файлов, конвенции вызова функций и передачи параметров.

Часть 2 **Вопросы отладки прикладных программ**

Содержит справочную информацию о средствах

отладки прикладных программ, описывает символьный отладчик, его основные функции, методы работы с ним.

Часть 3 **Описание форматов файлов БПО**

Содержит справочную информацию о форматах следующих типов файлов, используемых в БПО: объектные и исполняемые файлы (формат ELF и формат отладочной информации DWARF), карты памяти (.map), файлы объектных библиотек и макросов, файлы, содержащие дизассемблированную информацию, листинги ассемблерных файлов.

Соглашения о нотациях

В данном справочном руководстве используются следующие типографические нотации:

`courier` так помечается текст, который может быть набран пользователем с клавиатуры: команды и параметры, файлы и имена программ, исходные тексты на языках Си++ и ассемблера.

`-help` то же, что и в предыдущем случае, подчеркнутая часть слова говорит о том, что в командной строке при вызове программы можно использовать короткую нотацию вместо длинной, то есть вместо -help можно использовать -h, и это сокращение не приведет к изменению поведения программы.

`Courier` отмечает текст, который должен быть заменен пользовательской информацией, например, путем к той или иной компоненте БПО.

Текст так помечается текст, на который необходимо обратить особое внимание.

Примечание

Данное примечание представляет собой пример того, как оформлены все важные замечания и комментарии, возникающие по ходу описания.

Соглашения по именам файлов

При выборе имен файлов необходимо придерживается нотации, принятой в MS-DOS, то есть не более 8-ми символов отводится на имя и не более 3-х на расширение. Расширение должно отделяться точкой. Все файлы должны иметь расширение.

Примечание *Информация о том, какие именно расширения должны быть использованы для файлов тех или иных типов, приводится в главах, описывающих соответствующие компоненты, использующие их.*

Пример имени файла, соответствующего приведенным выше соглашениям:

```
d:\mydir\my_file.cpp
```

Имена программ компонент БПО

Здесь приводятся имена файлов программ компонент БПО процессора NM6403, описываемых в данном руководстве.

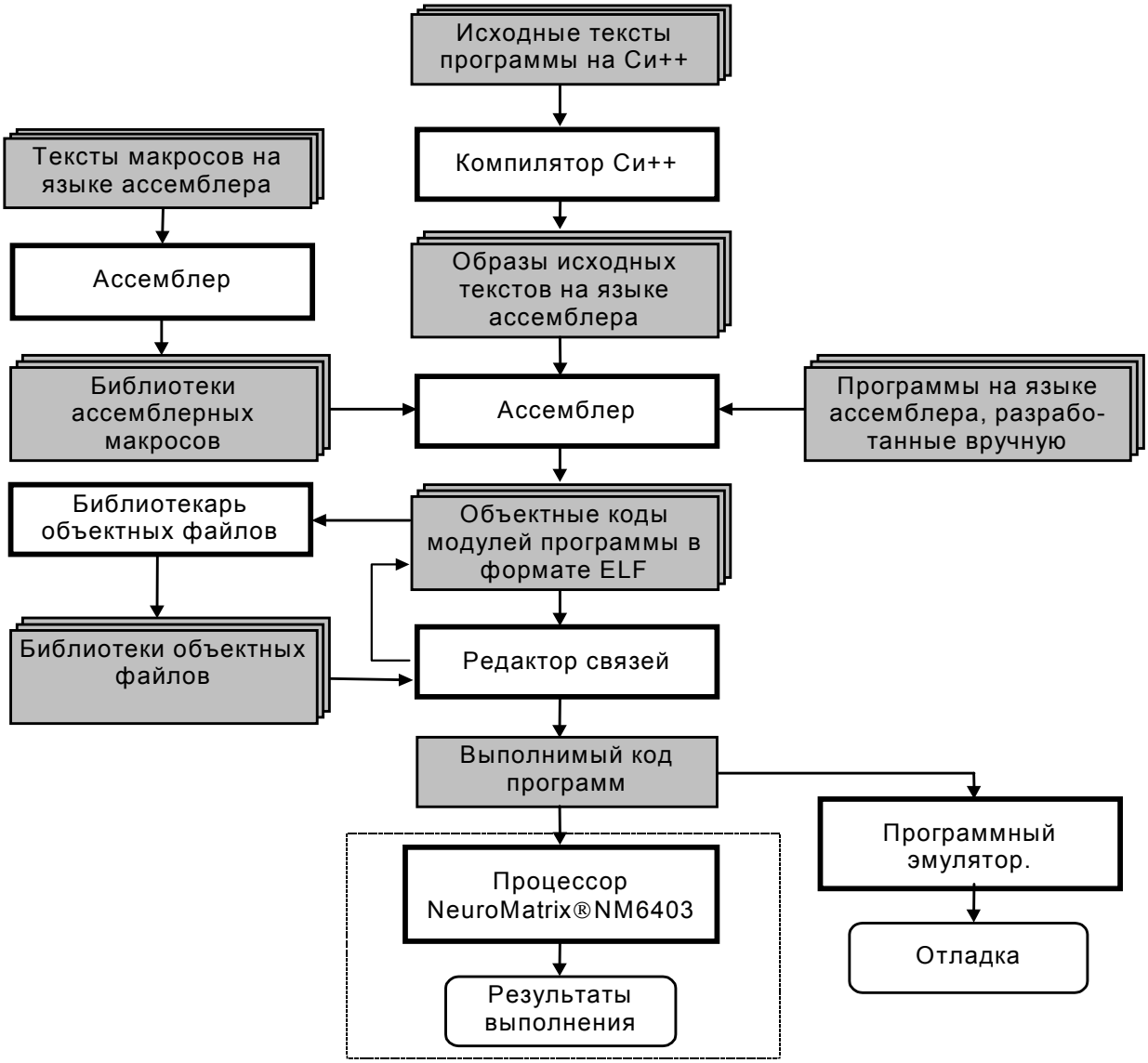
`nmcc`	Компилятор Си++ (драйвер компонент)	Глава 1
`asm`	Ассемблер	Глава 2
`linker`	Редактор связей	Глава 3
`libr`	Библиотекарь объектных файлов	Глава 4
`dump`	Декодер объектных и исполняемых файлов	Глава
`emurun/ nmrun`	Эмулятор процессора на уровне инструкций	Глава
`temu`	Точный эмулятор процессора	Глава
`emudbg`	Символьный отладчик под Windows'95	Глава

Библиотека времени выполнения Си существует либо в виде исходных текстов, либо в объектном виде. Во втором случае она называется `libc.lib` и расположена в каталоге `LIB`. Более подробное описание библиотеки, а также описание действий по ее сборке приводится в Главе XXXX.

Структура взаимодействия компонент БПО

Следующий рисунок иллюстрирует структуру взаимодействия компонент БПО процессора NeuroMatrix® NM6403.

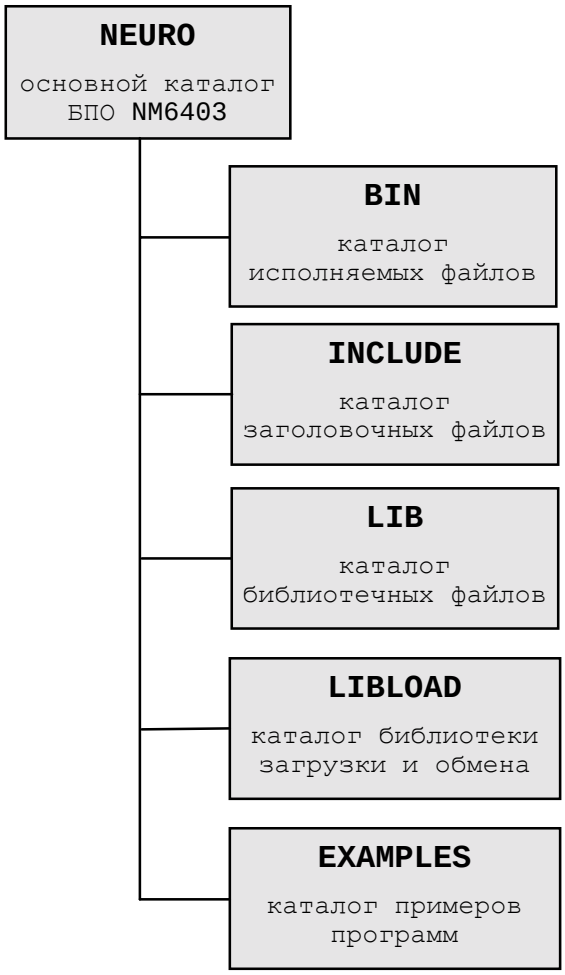
Рис. П-1-1 Схема взаимодействия компонент БПО.



Структура каталогов БПО

Для удобства работы с компонентами БПО процессора NM6403 используется предопределенная структура каталогов программного обеспечения. Она описывается следующим деревом каталогов(см. Рис. П-1-2):

Рис. П-1-2 Структура каталогов системного ПО процессора NM6403.



Содержимое каталогов зависит от комплектации поставляемого базового ПО процессора NM6403. Оно приводится в документе по инсталляции БПО.

Переменная окружения NEURO

Для того, чтобы в процессе компиляции и сборки прикладных программ компоненты БПО автоматически находили все необходимые библиотеки и заголовочные файлы, в базовое ПО процессора NM6403 введена переменная окружения `NEURO`.

Переменная окружения помещается в файл `autoexec.bat` в виде следующей строки:

`NEURO=<`*`путь_к_каталогу_БПО_процессора_NM6403`*`>;`

например:

`SET NEURO=`*`D:\NEURO`*`;`

Помимо задания переменной окружения в файл `autoexec.bat` в раздел `PATH` необходимо добавить путь к каталогу `BIN`, поскольку в

нем содержатся исполняемые программы компонент БПО процессора:

```
PATH=%PATH%;D:\NEURO\BIN;
```

Данные изменения, вносимые в файл `autoexec.bat`, добавляются туда автоматически на этапе инсталляции. Если пользователь отказался вносить изменения автоматически, впоследствии он может внести их вручную.

Как оформлять замечания и предложения

По документации

Если при работе с документацией, описывающей процессор и его базовое ПО, у Вас возникли сложности, например, вы считаете, что материал изложен недостаточно подробно для понимания, пожалуйста, присылайте свои замечания в следующем виде:

- название документа;
- персональный номер документа и номер версии документации;
- номера страниц, на которые приводится ссылка;
- краткое описание замечания;

По работе компонент БПО

Если при работе с какой либо из компонент БПО у Вас возникли сложности, пожалуйста, присылайте свои замечания в следующем виде:

- номер версии БПО процессора NM6403, которое Вы используете;
- небольшой фрагмент исходного кода, на котором проявляется ошибка;
- краткое описание ее проявления и возможные предположения о причинах ее возникновения, если таковые имеются.

1 Компилятор Си++

1.1 Введение

Данная глава содержит информацию о компиляторе Си++ для процессора NM6403. Здесь собраны данные о поддерживаемой компилятором версии языка Си++, краткая справка о составе компилятора, опциях управления компилятором, приводится пример компиляции простейшей программы.

Примечание

Данное справочное руководство не может использоваться в качестве вводного курса в по программированию на языке Си++, также оно не содержит справочной информации по языку Си++.

1.2 О компиляторе Си++ для NM6403

Компилятор Си++ для NM6403 поддерживает определение языка Си++, описанное в предварительном стандарте ANSI X3J16/95-0029.

Рассматриваемая версия компилятора Си++ предназначена для работы в среде Windows 95 или Windows NT 4.0 в качестве консольного приложения.

Компилятор имеет интерфейс командной строки и запускается на исполнение при помощи драйвера компонент `nmcc`, который представляет собой программное средство, предназначенное для объединения нескольких компонент БПО процессора NM6403 под единым пользовательским интерфейсом.

Примечание

Подпрограммы, составляющие непосредственно компилятор Си++ не предназначены для индивидуального запуска пользователем. Осуществлять компиляцию рекомендуется только через nmcc.

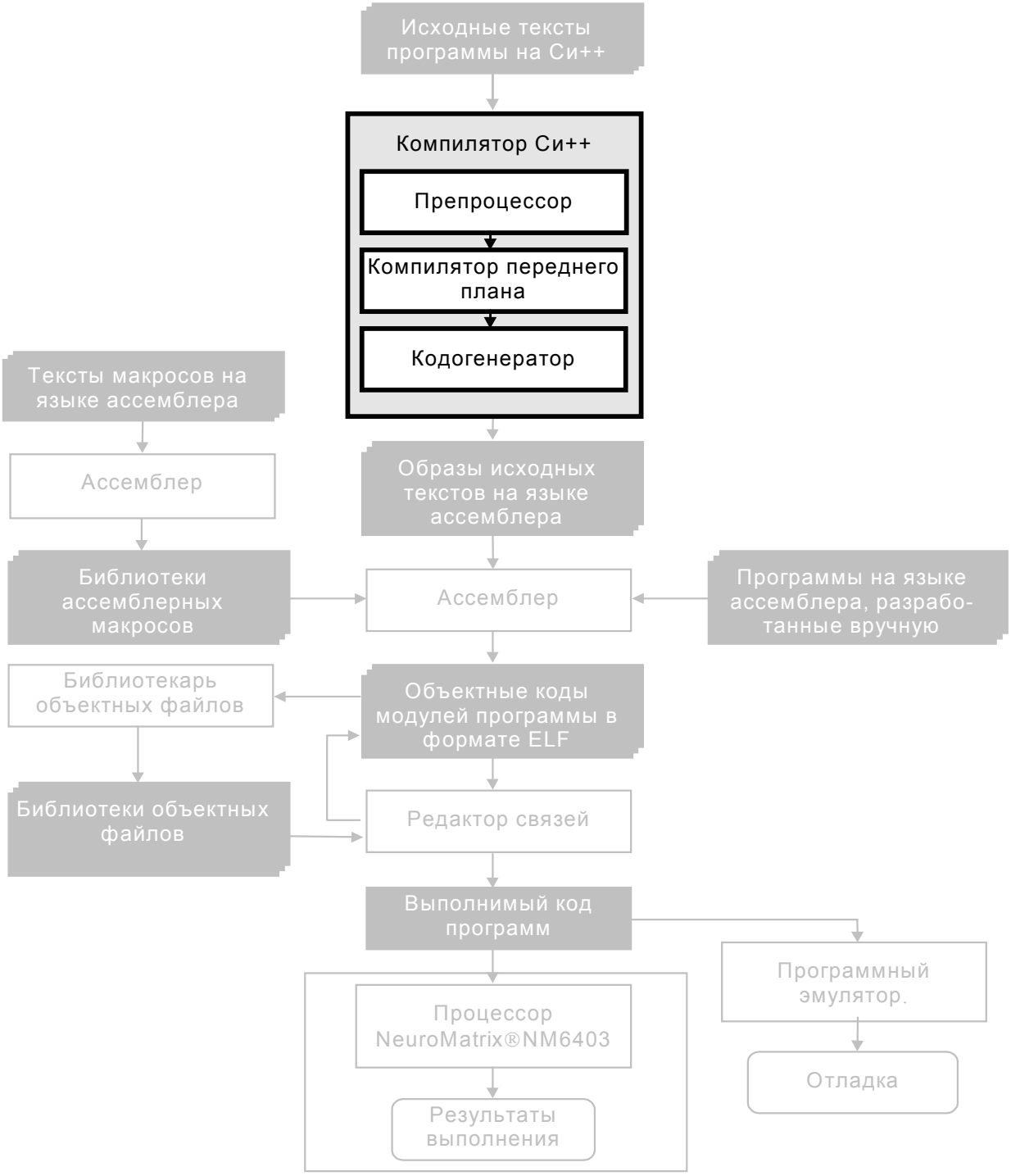
Процесс компиляции разбит на три стадии:

- предобработку, которую выполняет препроцессор;
- разбор входного файла и генерацию промежуточного файла, выполняемые компилятором переднего плана;
- порождение ассемблерного представления программы, осуществляемого генератором кода.

Стадии компиляции выполняются в приведенной выше последовательности, результатом компиляции является файл на языке ассемблера, адекватный соответствующему файлу на языке Си++.

Положение компилятора Си++ в общей структуре БПО приведено на Рис. 1-1.

Рис. 1-1 Положение компилятора Си++ в общей структуре БПО.



1.3 Драйвер компонент

БПО процессора NeuroMatrix® NM6403, предназначенное для создания пользовательских программ, состоит из следующих отдельных компонент:

- Компилятор Си++:
  - препроцессор;

  - компилятор переднего плана;
  - кодогенератор,
- Ассемблер;
- Редактор связей.

Для получения исполняемого файла из программы на языке ассемблера или Си/Си++, требуется последовательно запустить несколько компонент для каждого исходного файла проекта. Драйвер компонент позволяет автоматизировать процесс создания результирующего файла из произвольного числа исходных, путём объединения перечисленных компонент под единым интерфейсом.

Управление драйвером осуществляется путем использования входных параметров. Эти параметры во многом совпадают по нотации с теми, что используются при запуске отдельных компонент БПО. Также в качестве входных параметров рассматриваются имена исходных файлов и библиотек. Подробное описание всех входных параметров драйвера приведено в разделе 1.6. Список параметров управления компиляцией.

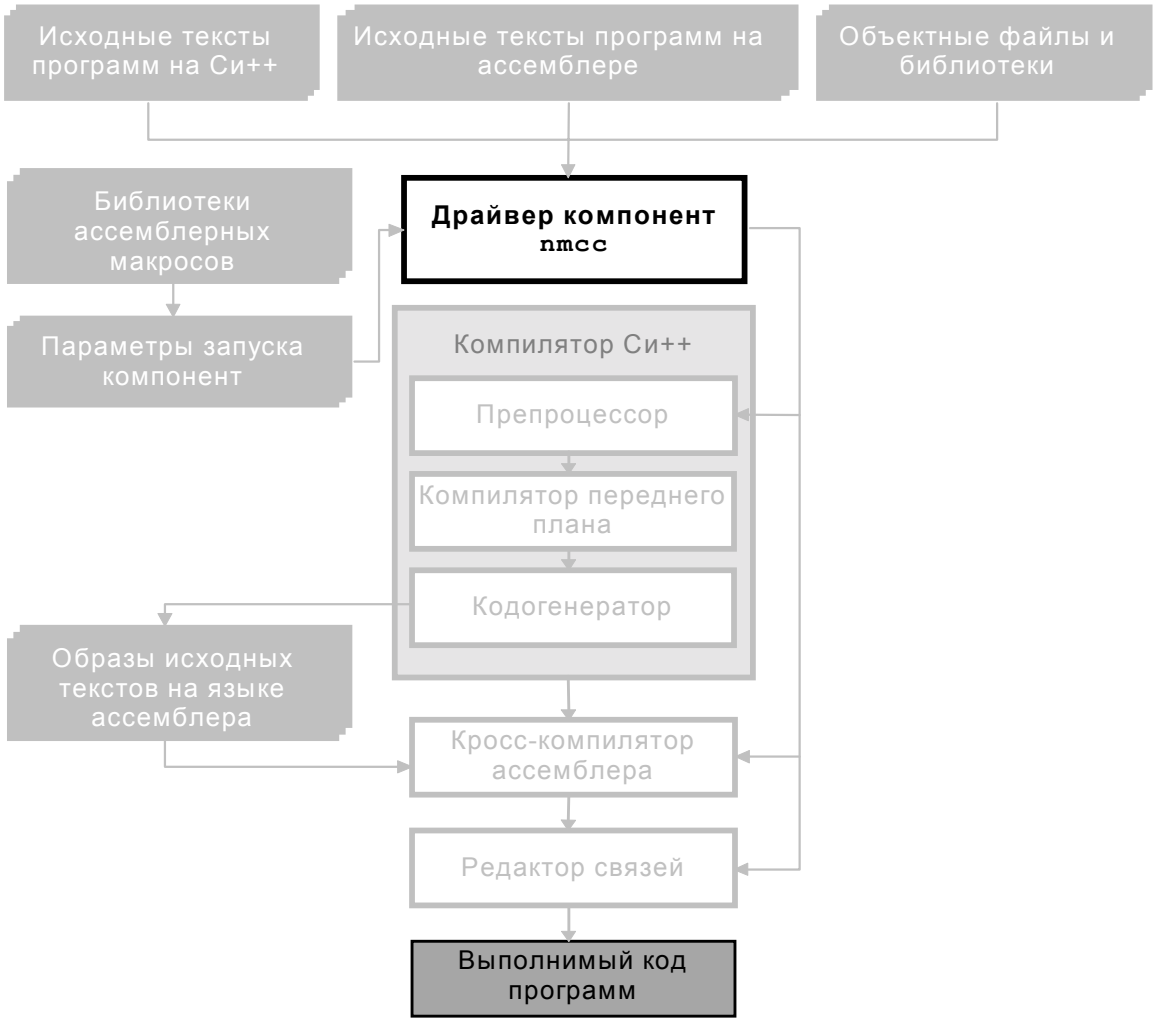
Для каждого исходного файла драйвер запускает компоненту или компоненты, предназначенные для его обработки, с теми входными параметрами, которые соответствуют данной компоненте. Например, для исходного Си/Си++ файла драйвер запустит все перечисленные выше компоненты, начиная с препроцессора.

В случае возникновения ошибки в процессе компиляции и сборки программы на экран выдается диагностической сообщение, и последующие шаги компиляции не выполняются.

1.3.1 Взаимодействие драйвера с компонентами БПО

На Рис. 1-2 показана роль драйвера компонент в структуре базового ПО процессора. Драйвер обрабатывает несколько типов входных файлов, в том числе исходные файлы на языках ассемблера и Си/Си++, объектные файлы, библиотеки. В результате работы драйвера создается абсолютный исполняемый или исполняемый перемещаемый файл, предназначенный для загрузки в память процессора или в программный эмулятор.

Рис. 1-2 Взаимодействие драйвера с компонентами БПО.



1.4 Соглашение о связи типов файлов с их расширениями

В базовом программном обеспечении процессора NM6403 используются следующие расширения для обозначения типов файлов:

Табл. 1-1 Расширения, используемые в БПО процессора NM6403.

Расширения	Описание
`.cpp, .c`	исходные тексты на языках Си++ и Си.
`.hpp, .h`	заголовочные файлы к исходным текстам на языках Си++ и Си.
`.asm`	файлы с исходными текстами на языке ассемблера для NM6403.
`.elf,.elz`	объектные файлы формата ELF, порожденные ассемблером для NM6403.
`.abs`	абсолютные исполняемые файлы формата ELF,

	порожденные редактором связей и готовые к загрузке на NM6403.
`.rel`	исполняемые перемещаемые файлы формата ELF, порожденные редактором связей и готовые к загрузке на NM6403.
`.lib`	библиотеки объектных файлов, порождаемые библиотекарем для NM6403.
`.mlb`	библиотеки макросов, порождаемые ассемблером для NM6403.
`.lst`	листинг-файл содержащий набор инструкций, порожденных ассемблером и соответствующие им команды на языке ассемблера.
`.map`	файл-карта памяти, описывающий размещение всех секций и глобальных переменных абсолютного исполняемого файла в адресном пространстве процессора.

Типы входных файлов распознаются по расширению. Следует понимать, что предназначение драйвера компонент - распределять файлы и параметры по соответствующим компонентам. Если расширение входного файла не соответствует его типу, то либо файл будет передан не подходящей компоненте, которая выдаст сообщение о несоответствии типа входного файла, либо будет отвергнут самим драйвером.

Файлы с расширением `.c` и `.cpp` подаются на вход цепочки компилятора Си++, `.asm` файлы подаются ассемблеру, а файлы `.elf`, `.elz`, `.lib` обрабатываются редактором связей.

1.5 Формат вызова драйвера компонент

```
nmcc [параметры] [файл1] ... [файлN]
```

`nmcc`	Имя файла, содержащего исполняемый код драйвера.
`Файл1...файл2`	Входные файлы исходных текстов, объектные файлы и библиотеки. Могут располагаться в произвольном порядке.
`Параметры`	Параметры драйвера компонент (с префиксом "`-`"). Могут располагаться в произвольном месте командной строки, в произвольной последовательности.(Более подробно

обсуждаются ниже).

1.6 Список параметров управления компиляцией

Для управления процессом компиляции используется набор параметров.

Все параметры начинаются с символа "-".

В названиях параметров различаются прописные и строчные буквы.

Последовательность, в которой расположены параметры, значения не имеет. Параметры разделяются между собой пробелами. Если за параметром следует аргумент, то он записывается **без пробела**, например, `-ofile.abs`. В противном случае драйвер компонент `nmcc` воспримет данное имя, как имя входного файла и, скорее всего, выдаст ошибку.

Табл. 1-2 Общие параметры.

Параметры	Описание
`-help или -?`	Вывод информации о параметрах.

Табл. 1-3 Параметры драйвера компонент.

Параметры	Описание
`-Stmp, -Skeeptemps`	Не удалять промежуточные файлы.
`-Snoexec`	Вместо запуска компонент выдать в стандартный поток вывода список сформированных команд запуска компонент.
`-Snolink`	Получить объектные файлы, но не передавать их редактору связей.
`-Ssyntax`	Проверить синтаксическую правильность Си++ программ.

Табл. 1-4 Параметры компонент БПО.

Параметры	Описание
`-g`	Создать и сохранять отладочную информацию.
`-I<имя_каталога>`	Путь к каталогу заголовочных файлов Си++ и библиотекам ассемблерных макросов.
`-L<имя_каталога>`	Путь к каталогу библиотечных файлов.

-X<опция>	Установить опции компилятора переднего плана.
-D<имя_символа>	Определить define символ.
-U<имя_символа>	Отменить определение define символа.
-T	Различать идентификаторы Си++ программ по первым восьми символам.
-C	Не удалять исходные комментарии при компиляции Си++ файлов.
-O	Включить оптимизацию.
-l<имя_файла>	Создать файл листинга ассемблерного файла.
-x<имя_файла>	Создать файл перекрёстных ссылок ассемблерного файла.
-o<имя_файла>	Задать имя выходного файла.
-c<имя_файла>	Задать имя файла конфигурации памяти для редактора связей.
-m<имя_файла>	Создать карту памяти абсолютного исполняемого файла.
-@<имя_файла>	Передать указанный файл редактору связей в качестве командного файла.
-asm	Отключить инициализацию статических глобальных переменных в языке Си++.

Кроме перечисленных драйвер распознаёт следующие, передаваемые без изменения, специальные параметры редактора связей:

Табл. 1-5 Параметры редактора связей.

-heap	-addr
-heap1	-abs
-stack	-rel
-start	-elf

Более подробное описание параметров редактора связей, приведенных в Табл. 1-5, содержится в разделе 3.5 Список параметров управления редактированием связей на стр. 3-8.

Драйвер компонент распознаёт только перечисленные параметры. При попытке использования других конструкций выдаётся сообщение и работа завершается.

1.7 Выдача справочной информации (ключи -help или -?)

При запуске драйверà с единственным параметром `-help` или `-?` на экран монитора выводится справочная информация обо всех параметрах, определяющих поведение драйвера и результат его работы. На экране появится следующая информация:

Рис. 1-3 Копия экрана в момент выдачи справочной информации.

```
NeuroMatrix NM6403 C++ Compiler v1.50

List of available options:
Miscellaneous options:
   -help,-h,-?          print out this help.
   -Stmp,-Skeeptemps    do not delete the temporary files,
   -Snoexec             do not execute constructed commands
                        but rather print it to stdout,
   -Snolink             only compile to object files,
   -Ssyntax             check for syntax errors only
                        (do not create any files).
Code Generation options:
   -g                   generate and keep debug info,
Preprocessor options:
   -I<>                 specify include directory,
   -L<>                 specify libraries path,
   -X<>                 set frontend option,
   -D<>                 define macro,
   -U<>                 undefine macro,
   -T                   differ identifiers by 8 first symbols,
   -C                   do not delete comments,
Output files:
   -l<>                 specify assembler listing file,
   -x<>                 specify assembler cross-reference file,
   -o<>                 specify output file name,
   -c<>                 specify linker configuration file,
   -m<>                 generate map file,
   -@<>                 specify linker command file.
Explanation of the other permitted options:
 -heap -heap1 -stack -start -addr -asm -abs -rel -elf,
```

После выдачи этого сообщения работа драйвера завершается. Использование ключей `-help` и `-?` совместно с другими ключами является ошибочным и порождает соответствующее сообщение.

При запуске драйвера без входных параметров на экран выводится информация о формате запуска программы `nmcc`.

1.8 Параметры драйвера (префикс -S)

Все параметры драйвера начинаются с префикса `-S`.

1.8.1 Запрет на удаление промежуточных файлов (ключи -Skeeptemps и -Stmp)

Параметры `-Stmp` и `-Skeeptemps` является синонимами. Они сообщают драйверу компонент о том, что промежуточные файлы, возникающие при компиляции программ, должны быть сохранены. Промежуточными файлами являются:

- `.cc` - файлы, являющиеся результатом работы препроцессора;
- `.ic` - файлы, являющиеся результатом работы компилятора переднего плана;
- `.asm` - файлы, порождённые генератором кода из `.ic` файлов.

По умолчанию, когда данный параметр не установлен, все промежуточные файлы, полученные в процессе компиляции и сборки программы, удаляются.

Примечание

При компиляции удаляются только те `.asm` файлы, которые были созданы в результате компиляции `.cpp` файлов. Критерием при этом служит совпадение имен файлов без расширения; например, при компиляции файла `prog.cpp` будет созданы `prog.cc`, `prog.ic`, `prog.asm` файлы. В отсутствие параметра `-Stmp` они будут удалены.

Следует отметить, что объектные `.elf` файлы, также как и исполняемые, являются целью компиляции и **никогда** драйвером не удаляются.

1.8.2 Выдача команд запуска компонент (ключ -Snoexec)

Параметр `-Snoexec` указывает драйверу компонент, чтобы он **вместо** реального запуска компонент для компиляции, **сформировал** и вывел список команд запуска компонент с подставленными параметрами. Команды выдаются в стандартный поток вывода безо всякой дополнительной информации.

Данный ключ может быть полезен, когда пользователь захочет знать, какие именно параметры были поданы на вход каждой из запускаемых компонент.

Пример:

```
D:\TMP>nmcc mirror.asm test1.cpp -Snoexec
preproc -F -DNM6403 -ID:\NEURO\INCLUDE test1.cpp test1.cc
c0 -o test1.ic test1.cc
codegen -q test1.ic -otest1.asm
asm -q -ID:\NEURO\INCLUDE mirror.asm -omirror.elf
asm -q -ID:\NEURO\INCLUDE test1.asm -otest1.elf
linker -q -lD:\NEURO\LIB mirror.elf test1.elf libc.lib
```

Из примера видно, что при запуске драйвера компонент `nmcc` с параметром `-Snoexec` на экран выводится последовательность вызова компонент БПО с теми параметрами, которые сформированы драйвером на основании информации, переданной на вход `nmcc`.

1.8.3 Компиляция в объектные файлы (ключ -Snolink)

Параметр `-Snolink` позволяет отключить фазу редактирования связей объектных модулей и получения исполняемой программы. В этом случае компиляция останавливается после получения `.elf` файлов.

1.8.4 Проверка синтаксиса Си/Си++ программ (ключ -Ssyntax)

Параметр `-Ssyntax` отключает запуск всех компонент, кроме препроцессора и компилятора переднего плана.

1.9 Параметры компонент

Большинство параметров драйвера `nmcc` на самом деле являются ключами для одной или нескольких компонент, причем имена и форматы записи ключей в основном совпадают с используемыми в компонентах. Все исключения в дальнейшем изложении будут особо оговорены. Подробные описания параметров компонент приведены в соответствующих разделах данного справочного руководства.

Далее следует краткое описание ключей. Для каждого ключа приводится краткое описание, форма записи и список компонент, в строке запуска которых данный ключ может присутствовать.

1.9.1 Создание отладочной информации (ключ -g)

Параметр `-g` служит для создания и сохранения отладочной информации.

Без дополнительных аргументов.

Используется при вызовах:

- компилятора переднего плана;

- генератора кода;
- ассемблера.

При наличии в командной строке драйвера компонент параметра `-g` на вход редактора связей подается ключ `-d1`.

1.9.2 Указание каталогов дополнительных файлов (ключи -I и -L)

Ключ `-I` используется для задания каталога, в котором будет осуществляться поиск:

- заголовочных файлов для программ на Си++;
- библиотек ассемблерных макросов.

Имя каталога должно следовать за ключом без пробела, например:

```
-ID:\NEURO\INCLUDE
```

Используется при вызовах **препроцессора** и **ассемблера**.

Ключ `-L` используется для задания каталога поиска объектных библиотек.

Имя каталога должно следовать за ключом без пробела, например:

```
-LD:\NEURO\LIB
```

Используется при вызове **редактора связей**.

1.9.3 Установка опций компилятора переднего плана (ключ -X)

Ключ `-X` позволяет установить различные параметры компилятора переднего плана.

Табл. 1-6 Параметры компилятора переднего плана.

Параметры	Описание
`-Xinline=n`	Это параметр компилятора переднего плана влияет на обработку определенных пользователем `inline`-функций. Если n не равно 0, компилятор Си++ будет порождать для этих функций встроенный код, а не вызов функции. В противном случае вызовы `inline`-функций будут обрабатываться, как обычные вызовы. Значение по умолчанию: `-Xinline=1`.
`-Xexcep=n`	Этот параметр компилятора переднего плана влияет на порождение кода исключительных ситуаций языка Си++. Если n не равно 0, компилятор Си++ порождает код для поддержки исключений. В противном случае код поддержки исключений не порождается. Использование данного ключа имеет смысл лишь в том случае, когда

	программа пользователя не использует исключений языка Си++. В таком случае не будет порождаться дополнительный код для свертки стека (`stack unwinding`). Если в параметре задано `n`, равное нулю, но в исходном тексте программы есть хотя бы одно `throw`-выражение, компилятор выдаст сообщение об ошибке. Значение по умолчанию: `-Xexcep=1`.
`-Xrtti=n`	Этот параметр компилятора переднего плана влияет на порождение структур данных для поддержки механизма идентификации типов языка Си++ на этапе выполнения (`RTTI`). Если `n` не равно 0, компилятор Си++ порождает такие структуры. В противном случае никаких данных для поддержки `RTTI` не порождается. Использование данного ключа имеет смысл лишь в том случае, когда пользовательская программа использует `RTTI`-особенности Си++. Необходимо заметить, что не все выражения `typeid` и `dynamic_cast` требуют порождения данных `RTTI`. Если значение данного ключа равно нулю, но при этом `RTTI`-данные всё равно используются, компилятор выдаст сообщение об ошибке. Значение по умолчанию: `-Xrtti=0`.
`-Xold=n`	Этот параметр компилятора переднего плана влияет на проверку ошибок. Если `n` не равно 0, компилятор Си ++ будет поддерживать некоторые старые правила ANSI C, отмененные в ANSI Си++. На данный момент этот ключ разрешает неявное преобразование от арифметических типов к перечислимым (`enum type`) и неявное преобразование указателя на `void` к произвольному типу указателя. Ключ полезен при компиляции кода ANSI C компилятором ANSI Си++. Значение по умолчанию: `-Xold = 0`.

Данные параметры используются при вызове **компилятора переднего плана**.

1.9.4 Параметры препроцессора (ключи -D, -U, -T, -C)

Параметры препроцессора используются только при вызове **препроцессора**.

1.9.4.1 Макро-символы Си++ (ключи -D и -U)

Параметры `-D` и `-U` используются соответственно для определения и уничтожения определения макро-символов языка Си++.

Имя макроса должно следовать непосредственно за используемым ключом. В случае определения можно задавать желаемое значение символа:

```
-DDEBUG
-UDEBUG
-DVER=100
```

1.9.4.2 Установка длины различаемых идентификаторов (ключ -T)

Параметр `-T` устанавливает режим различения идентификаторов Си++ по первым восьми символам.

Без дополнительных аргументов.

1.9.4.3 Сохранение комментариев (ключ -C)

Параметр `-C` позволяет сохранить комментарии при обработке Си++ файлов при помощи препроцессора.

Без дополнительных аргументов.

1.9.5 Включение режима оптимизации (ключ -O)

Использование параметра `-O` позволяет включить режим порождения оптимизированного кода.

Без дополнительных аргументов.

Передаётся **генератору кода**.

1.9.6 Порождение файлов листинга и перекрёстных ссылок (ключи -l и -x)

Ключи `-l` и `-x` позволяют создавать файлы листингов и файлы перекрёстных ссылок для ассемблерных модулей.

По умолчанию создаваемые файлы имеют имена, образованные из имён исходных ассемблерных файлов заменой расширения на `.lst` и `.crf` соответственно. Для создания этих файлов с другими именами можно указать желаемое имя непосредственно за используемым ключом: `-l`<имя_файла>. Однако изменение имени по умолчанию на произвольное разрешено только при компиляции одиночного `.cpp` или `.asm` файла. Указание аргументов ключей `-l` и `-x` при запуске драйвера со многими исходными файлами порождает сообщение об ошибке и завершение работы драйвера.

Ключи используются при запуске **ассемблера**.

1.9.7 Указание имени выходного файла (ключ -o)

Выходным файлом является файл, собранный редактором связей из скомпилированных другими компонентами модулей. По умолчанию в качестве имени выходного файла используется первое имя из списка входных файлов. При этом, расширение нового файла будет зависеть от того, каков его тип:

Табл. 1-7 Расширения для различных типов объектного файла.

Расширение	Описание
.abs	для абсолютных исполняемых файлов.
.rel	для исполняемых перемещаемых файлов.
.elz	для объектных файлов.

Произвольное имя выходного файла задаётся с помощью ключа:

-o<имя_файла>, где <имя_файла>- обязательный аргумент ключа.

Указанный ключ передаётся **редактору связей**.

1.9.8 Задание файла конфигурации редактора связей (ключ -c)

С помощью ключа -c задаётся файл конфигурации для редактора связей, в котором описывается схема распределения в памяти секций программы. Имя файла конфигурации должно следовать за ключом без пробела: -c<имя_файла>. Подробное описание файла конфигурации приведено в разделе 3.11 Файл конфигурации на стр. 3-27.

Используется при запуске **редактора связей**.

1.9.9 Создание карты памяти (ключ -m)

С помощью ключа -m<имя_файла> редактору связей даётся указание создать файл карты памяти для выходного файла.

Более подробное описание данного параметра содержится в параграфе 3.8.9 Создание карты памяти (ключ -m<имя_каталога>) на стр. 3-22.

Передаётся **редактору связей**.

1.9.10 Командный файл для редактора связей(ключ -@)

Командный файл является альтернативным способом задания параметров редактора связей. В этом файле записываются параметры вызова редактора связей в том же виде, в каком они используются непосредственно в командной строке. Командный файл - хорошее место для размещения редко меняющихся параметров.

Имя файла команд должно идти непосредственно за ключом:

-@<имя_файла>.

Форма записи этого ключа отличается от используемой для вызова редактора связей: -@ для драйвера, в отличие от просто @ для редактора.

Более подробное описание данного параметра содержится в параграфе 3.6.6 Командный файл (@<имя_файла>) на стр. 3-12

Используется при вызове **редактора связей**.

1.9.11 Отключение инициализации статических глобальных объектов (-asm)

Параметр `-asm` позволяет не использовать средств, необходимых для инициализации статических глобальных объектов в Си++.

Если пользователь использует свою точку входа, а его программа написана на ассемблере и не требует досрочной инициализации статических полей, он может использовать данный ключ для удаления из исполняемого файла специальных секций `.init` и `.fini`.

Более подробное описание данного параметра содержится в параграфе 3.8.7 Отключение инициализации статических глобальных объектов (ключ -asm) на стр. 3-20

Передаётся **редактору связей**.

1.10 Параметры вызова компонент по умолчанию

1.10.1 Списки параметров для каждой компоненты по умолчанию

При запуске компонент по умолчанию используются следующие параметры вызова:

- `-F, -DNM6403` для препроцессора (отметим, что по умолчанию определяется имя макро-символа языка Си++ `NM6403`, определяющее тип процессора),
- `-q` для генератора кода, ассемблера и редактора связей,
- `libc.lib` для редактора связей.

В случае, если установлена переменная среды `NEURO`, в строки вызова компонент автоматически добавляются пути к заголовочным файлам, к библиотекам макросов и библиотекам объектных файлов, входящих в стандартный комплект БПО процессора NM6403.

1.10.2 Имя выходного файла по умолчанию

В случае, когда осуществляется компиляция и сборка прикладной задачи, состоящей из нескольких файлов, и при этом не указано имя выходного файла, по умолчанию действует следующее соглашение:

Имя выходного файла совпадает с именем первого в списке компилируемых файлов, а расширение соответствует типу выходного файла, например, результатом успешной компиляции:

```
nmcc aaa.cpp bbb.cpp ccc.cpp
```

будет файл `aaa.abs`.

По умолчанию результатом компиляции является абсолютный исполняемый файл (расширение `.abs`).

Все описываемые в данном пункте соглашения, используемые по умолчанию, определяются редактором связей, поскольку именно он завершает этап компиляции. Более подробная информация об умалчиваемых значениях содержится в разделе 3.9 Параметры по умолчанию на стр. 3-24.

1.11 Пример работы с драйвером компонент

Данный пример показывает процесс сборки абсолютного файла из двух исходных файлов:

```
>nmcc t2.cpp templ.cpp -otempl.abs
"templ.cpp", line 5: (warning): return type for 'main'
                                changed to integer type
>
```

Первым исходным файлом поставлен файл `t2.cpp`, однако выходной файл с помощью ключа `-o` именуется по имени главного файла проекта: `templ.abs`. Для сборки выходного файла используется библиотека времени выполнения Си++, которая подключается автоматически при условии, что установлена переменная среды `NEURO`.

Предупреждение было выдано компилятором переднего плана, т.к. тип функции `main` в файле `templ.cpp` был описан как `void`, что не совпадает требованием нового стандарта языка Си++.

Можно посмотреть какие компоненты были запущены драйвером:

```
>nmcc t2.cpp templ.cpp -otempl.abs -Snoexec
preproc -F -DNM6403 -ID:\NEURO\INCLUDE t2.cpp t2.cc
c0 t2.cc -o t2.ic
preproc -F -DNM6403 -ID:\NEURO\INCLUDE templ.cpp templ.cc
c0 templ.cc -o templ.ic
codegen -q t2.ic -ot2.asm
codegen -q templ.ic -otempl.asm
asm -q -ID:\NEURO\INCLUDE t2.asm -ot2.elf
asm -q -ID:\NEURO\INCLUDE templ.asm -otempl.elf
linker -q -otempl.abs -lD:\NEURO\LIB t2.elf templ.elf libc.lib
>
```

Видно, что драйвер произвел подстановку переменной среды `NEURO`; стандартные файлы заголовков (`.h`) и стандартные макробиблиотеки находятся на диске в каталогах, подставляемых автоматически. Также видно, какие промежуточные файлы создаются в процессе работы компонент:

```
t2.cc t2.ic t2.asm templ.cc templ.ic templ.asm
```

Так как â командной строке не был задан параметр `-Stmp`, все эти файлы драйвер по окончании работы удалил. Остались лишь объектные и абсолютный файлы (редактор связей по умолчанию создаёт абсолютные исполняемые файлы):

```
t2.elf, templ.elf и templ.abs
```

1.12 Сообщение об ошибках при работе драйвера компонент

В время работы драйвера `nmcc` могут возникать ошибочные ситуации. Если ошибка возникла в процессе работы какой либо компоненты, сообщение об этой ошибке выдаёт сама компонента. В этом случае драйвер прекращает дальнейшую работу и завершается с результатом 1.

Сообщение об ошибке может быть порождено самим драйвером. Сообщения драйвера, как правило, являются следствием ошибки в параметрах его вызова. Все сообщения драйвера начинаются с префикса `"cc :"`

Драйвер порождает следующие сообщения:

```
"Unrecognized option '<использовавшееся_имя>'"
```

Драйвер был вызван с неизвестным ему ключом (все параметры драйвера начинаются с “`-`“).

```
"Unrecognized input file <использовавшееся_имя>'"
```

Драйвер был вызван с файлом неизвестного ему типа (известные типы файлов: `.c .cpp .asm .elf .elz .lib`).

```
"Multiple source files forbids implicit specification
of list file name."
```

Было указано имя файла листинга одновременно с указанием нескольких исходных `.c`, `.cpp` и `.asm` файлов. При запуске со многими исходными файлами использовать ключ `-l` с указанием имени запрещено. Однако ключ `-l` без дополнительного аргумента использовать можно, при этом для каждого ассемблерного файла (исходного или созданного в результате компиляции Си++ файла) создаётся файл листинга с именем, образованным из его имени заменой расширения на `.lst`.

```
"Multiple source files forbids implicit specification
of crossref file name."
```

Причина та же, что у предыдущей ошибки. Можно использовать -x

без дополнительного аргумента. Для создающихся в этом случае файлов перекрёстных ссылок используется расширение `.crf`.

```
"No such environment variable: <использовавшееся_имя>"
```

В командной строке вызова драйвера была использована несуществующая переменная окружения. Использовавшееся имя переменной выводится.

```
"Unexpected environment variable substitution error"
```

Произошла непонятная ошибка во время подстановки значений переменных окружения. При нормальной работе драйвера и нормальном состоянии системы эта ошибка возникнуть не может. В случае возникновения следует обратиться к разработчикам с подробным описанием условий, при которых произошла данная ошибка.

1.13 Детали реализации компилятора Си++

В данном разделе содержится информация о тех аспектах реализации компилятора и системной библиотеки поддержки языка Си++, которые определены в стандарте языка Си++, как определяемые реализацией.

1.13.1 Стандартные типы элементов данных

В силу архитектуры процессора NM6403 и набора поддерживаемых команд доступа к памяти, все стандартные элементы данных выражаются 32-х или 64-х битными словами в зависимости от количества битов, требуемых для их представления.

В процессоре минимально доступным является 32-х разрядное слово, поэтому независимо от того, что тип char или short может быть представлено 8-ю или 16-ю битами соответственно, в памяти они занимают 32 бита.

В таблице представлены базовые типы данных языка Си++, их размер в 32-разрядных словах, и выравнивание типа данных в памяти:

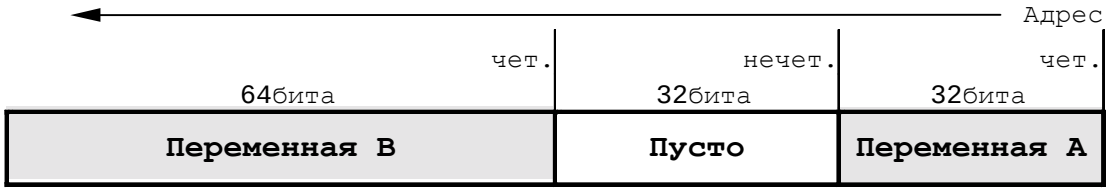
Табл. 1-8 Представление базовых типов данных Си++.

Базовые типы данных	Размер в словах процессора	Количество значимых бит	Выравнивание типа данных в памяти
char	1	8	1
short	1	16	1
int	1	32	1

long	2	64	2
указатели	1	32	1
float	1	32	1
double	2	64	2
long double	2	64	2

Понятие "Выравнивание типа данных в памяти" используется для того, чтобы подчеркнуть, что переменные, занимающие 64 бита, **не могут** располагаться по нечетным адресам памяти (см. Рис. 1-4). Поэтому если рядом расположены две переменные, одна из которых имеет размер 32 бита и лежит в памяти по четному адресу, а идущая следом за ней переменная имеет размер 64 бита, то между этими двумя переменными в памяти расположено неиспользуемое пространство, размером в 32 бита.

Рис. 1-4 Выравнивание типов данных в памяти.



1.13.2 Наборы символов и идентификаторы

Идентификатор может иметь любую длину, однако компилятор Си++ игнорирует символы после 256 символа (стандарт требует не менее 31 символа). Все символы до 256-го являются значимыми.

Набор символов, встречаемых в исходной программе, представляет собой 7-ми битный ASCII набор. В комментариях могут встречаться все символы из 8-ми битной ASCII таблицы.

Прописные и строчные буквы различаются для всех внутренних и внешних идентификаторов.

1.13.3 Диапазоны типов данных (limits.h и float.h)

В стандарте языка ANSI C++ определены два заголовочных файла `limits.h` и `float.h`, которые описывают диапазоны значений констант, определяемые типами данных.

Стандарт также определяет минимальные размеры и наличие/отсутствие знакового бита для описываемых типов данных.

Минимальный размер переменной, не являющейся битовым полем, равен 32 бита:

```
CHAR_BIT 32
```

Максимальное количество байтов в мультибайтовом символе:

```
MB_LEN_MAX 1
```

Для следующих целочисленных числовых диапазонов средняя колонка содержит числовое представление определяемого предела, а правая отображает его битовое представление в шестнадцатеричном виде.

Табл. 1-9 Предельные значения констант и переменных в реализации Си++.

Символьное обозначение	Предельное значение	Побитовое представление
CHAR_MAX	255	0xFF
CHAR_MIN	0	0x00
SCHAR_MAX	127	0x7F
SCHAR_MIN	-128	0x80
UCHAR_MAX	255	0xFF
SHRT_MAX	32767	0x7FFF
SHRT_MIN	-32768	0x8000
USHRT_MAX	65535	0xFFFF
INT_MAX	2147483647	0x7FFFFFFF
INT_MIN	-2147483648	0x80000000
LONG_MAX	9223372036854775807	0x7FFFFFFFFFFFFFFF
LONG_MIN	–9223372036854775808	0x8000000000000000
ULONG_MIN	18446744073709551616	0xFFFFFFFFFFFFFFFF

2 Ассемблер

2.1 Введение

Ассемблер для NM6403 - это однопроходный компилятор, переводящий программу, написанную на языке ассемблера в объектный файл формата ELF. Ассемблер не разрешает внутренние ссылки, не определяет неопределенные внешние символы. Главная его задача - построить таблицу символов и произвести преобразование ассемблерных строк в инструкции процессора.

Ассемблер позволяет создавать библиотеки макросов, а также добавлять в уже существующие библиотеки новые макросы.

2.2 Положение ассемблера в структуре БПО

Ассемблер обрабатывает файлы на языке ассемблера, полученные из трех источников:

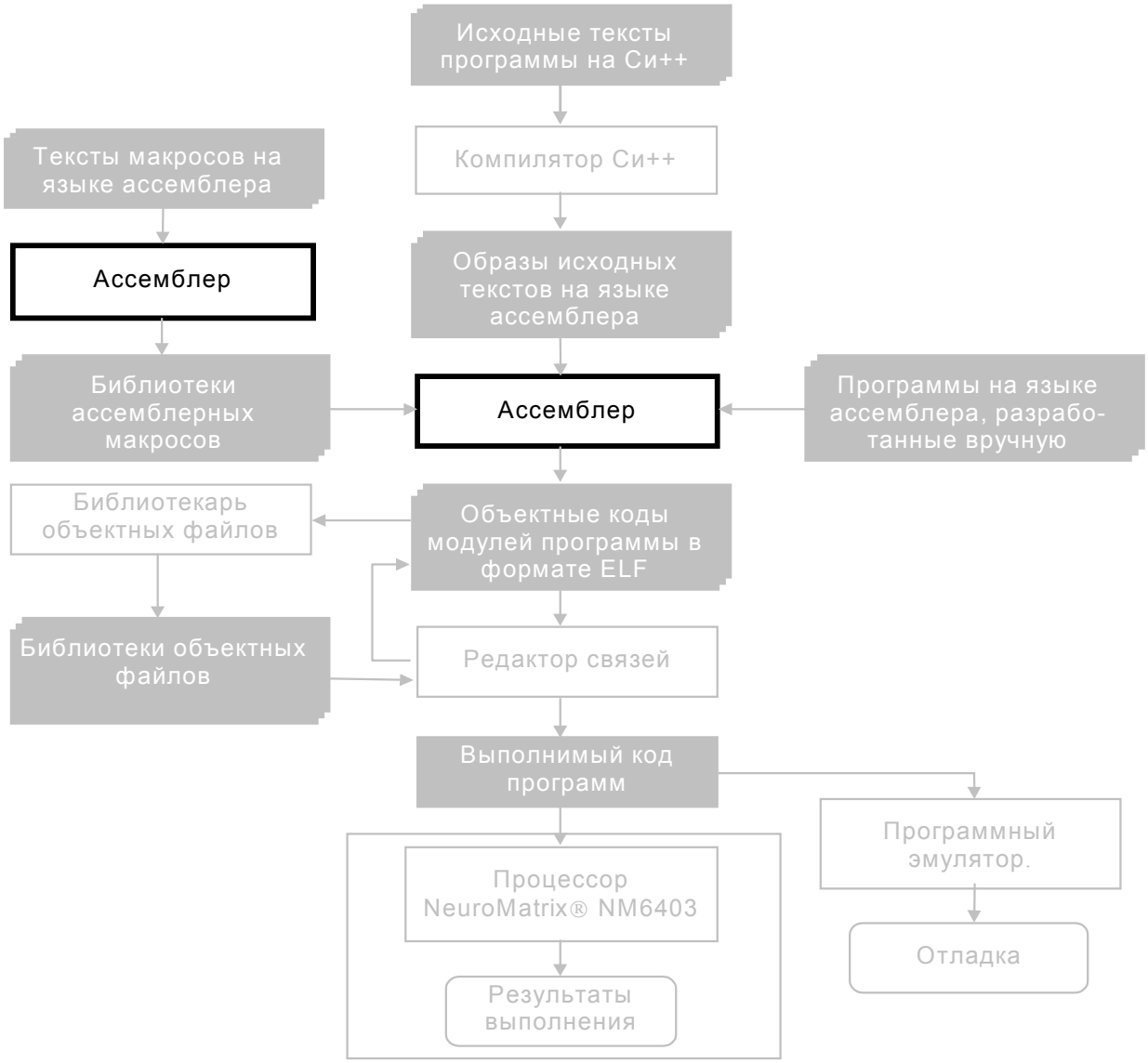
- файлы, порожденные компилятором Си++;
- файлы на языке ассемблера, разработанные вручную;
- макробиблиотеки, порожденные самим же ассемблером во время предыдущих сеансов работы.

Результатом работы ассемблера может быть либо объектный файл, либо библиотека макросов.

Приведенные выше входные и выходные данные ассемблера определяют его место в структуре БПО процессора NM6403.

На показано положение ассемблера в общем пути получения исполняемой программы из различных типов входных данных.

Рис. 2-1 Положение ассемблера в структуре БПО.



2.3 Формат вызова ассемблера

```
asm [параметры ] [ входной файл ]
```

`asm`	Имя файла, содержащего исполняемый код ассемблера для NM6403.
`Входной файл`	Входной файл, содержащий программу на языке ассемблера.
`Параметры`	Параметры ассемблера (с префиксом "-"). Могут располагаться в произвольном месте командной строки, в произвольной последовательности. (Более подробно обсуждаются ниже).

В квадратные скобки заключены необязательные параметры.

Если не указано имя входного файла или не задан ни один параметр, ассемблер не произведет никаких действий, выдавая на экран только сообщение о собственном названии, версии и авторских правах.

В конце работы в качестве кода возврата ассемблер возвращает общее количество ошибок в тексте исходной программы, если таковые имеются, или 0 в случае их отсутствия.

2.4 Список параметров управления ассемблером

Для управления работой ассемблера используется набор параметров (ключей).

Все параметры предваряются префиксом "-".

Они должны быть указаны в командной строке.
Последовательность, в которой расположены параметры, значения не имеет. Параметры разделяются между собой пробелами.

Ниже приводятся таблицы параметров ассемблера:

Табл. 2-1 Общие параметры.

Параметры	Описание
-q	Ассемблер не выдает никаких сообщений о ходе работы кроме сообщений об ошибках.
-i	Ассемблер не выводит никаких сообщений о своей работе.
-h или -?	Ассемблер выводит на экран краткую информацию о параметрах вызова.
-t	Ассемблер выводит свой заголовок, определяющий его полное название, версию и авторские права.
-p	Ассемблер в начале работы выводит свое полное имя, содержащее путь к каталогу, в котором он расположен.
-I<macrolib>	Указывает каталог, в котором следует искать библиотеки макросов.

Параметры -p, -t -h, -? являются справочными, и поэтому не могут использоваться одновременно. При одновременном их появлении в командной строке ассемблер выдает ошибку, сообщая что только один из параметров может быть использован в качестве входного.

Например, при задании параметра `-t`:

```
asm -t
```

выдает следующее сообщение:

```
Assembler for NM6403 v1.23. (c) RC Module Inc. 1996-
1998. All rights reserved.
```

Тогда как при одновременном появлении в командной строке рядом с ним любого другого параметра.

Пример

При вызове ассемблера с параметрами:

`asm -t -h,` или `asm -t -q`

будет выдано:

```
Invoke error ASM702: -p,-h,-? or -t must stand alone
```

Табл. 2-2 Параметры управления выходными файлами.

Параметры	Описание
`-o<out_file>`	Задаётся имя выходного файла компоненты. Если опция не задана, то имя выходного файла совпадает с именем входного. Если опция задана, то выходной файл имеет имя: `out_file`.
`-l`	Порождает файл с листингом программы.
`-x`	Создает файл со списком перекрестных ссылок.

Табл. 2-3 Параметры работы в режиме макро-библиотекаря.

Параметры	Описание
`-m[<macrolib>]`	При появлении данного параметра в командной строке ассемблер переходит в режим работы макро-библиотекаря. Имя `macrolib` указывает имя библиотеки. По умолчанию оно совпадает с именем входного файла, к которому добавляется расширение `.mlb`.
`-a`	Данный параметр обеспечивает добавление макросов, содержащихся во входном файле в макробиблиотеку, имя которой определяется предыдущим ключом.

Примечание

В случае, если среди ключей командной строки, подаваемой на вход ассемблеру, есть параметры перехода в режим макро-библиотекаря, то объектный файл не порождается.

Табл. 2-4 Параметры управления сообщениями пользователю.

Параметры	Описание
`-W[+|-]<num>`	Данный параметр включает/выключает возможность вывода отдельного сообщения на экран по его номеру.
`-W[+|-]<group>`	Данный параметр в сочетании с аббревиатурой, определяющей различные группы сообщений, позволяет запретить/разрешить вывод сообщений заданной группы на экран.

2.5 Параметры общего назначения

К параметрам общего назначения относятся параметры, имеющие одинаковый смысл, и выполняющие одни и те же действия для большинства компонент БПО процессора NM6403.

2.5.1 Выдача справочной информации (ключи -h, -?)

При запуске ассемблера с параметром `-h`, или `-?` на экран монитора выводится справочная информация обо всех ключах, определяющих поведение ассемблера и результат его работы. На экране появится следующая информация:

Рис. 2-2 Содержимое экрана при выдаче справочной информации по ассемблеру.

```
Ассемблер для NM6403 1.30. (c) ЗАО НТЦ Модуль 1996-1998. Все права защищены.

Формат запуска: asm [-[i|q]plxda] [-I<>] [-o<>] [-m<>] [-W<>] имя_файла

 имя_файла - входной ассемблерный файл.

Общие ключи:
 -q           - запретить выдачу заголовка программы.
 -i           - запретить вывод любых сообщений.
 -I<путь>     - указать путь (каталог) к макробиблиотекам. По-умолчанию
                макробиблиотеки ищутся а текущем каталоге.
 -p           - вывести место расположения ассемблера. Не может быть
                использован вместе с другими ключами.
 -t           - вывести заголовок ассемблера.
 -h,-?        - вывести эту страницу.
 Каждый из ключей -p,-t,-h,-? исключает использование всех других ключей.

Ключи вывода:
 -o<файл>     - по-умолчанию имя выходного elf файла получается из входного
                заменой расширения на '.elf'.
                Ключ -o позволяет указать желаемое имя выходного файла.
 -l           - создать файл листинга; имя файла создаётся из имени выходного
                файла заменой расширения '.lst'.
 -x           - создать таблицу перекрестных ссылок; имя файла создаётся из
                имени выходного файла заменой расширения '.crf'.
 Ключи вывода исключают использование ключей макробиблиотекаря.

Ключи макробиблиотекаря:
 -m[<файл>]   - создать макробиблиотеку из макросов входного файла;
                если <файл> не указан, для создания имени библиотеки
используется
                расширение '.mlb'.
                Elf файл не создается.
 -a           - добавить к библиотеке. Библиотечный файл должен существовать.
 Ключи макробиблиотекаря исключают использование ключей вывода.

Отладочные ключи:
 -d           - включить отображение процесса синтаксического разбора.

Ключи управления предупреждающими сообщениями:
 -W[+|-]<номер>  - разрешить/запретить выдачу предупреждения <номер>.
                   Отсутствие знака означает отключение.
 -W[+|-]<группа> - разрешить/запретить выдачу группы предупреждений;
                   допустимые группы:
                      all       - все предупреждения,
                      debug     - предупреждения отладочных команд,
                      object    - предупреждения создания объектного кода,
                      compile   - предупреждения компиляции,
                      librarian - предупреждения библиотекаря.
```

При появлении в командной строке ключа -h или -? работа ассемблера завершается, выходной файл не создается.

2.5.2 Режимы молчания (ключи -q и -i)

В зависимости от параметра ассемблер выдает информацию следующим образом:

- в случае отсутствия ключей `-q` или `-i` ассемблер выдает пользователю всю порождаемую информацию, а именно титульный заголовок, предупреждения, сообщения об ошибках;
- в случае `-q` пользователь получает только информацию об ошибках и предупреждения. Данный режим часто используется при пакетном запуске ассемблера или при вызове из интегрированной среды;
- в случае `-i` ассемблер не выдает ничего, то есть при работе ассемблера в данном режиме подавляются даже сообщения об ошибках.

2.5.3 Выдача титульного сообщения ("шапки") (ключ -t)

Параметр `-t` носит исключительно информационный характер. Она используется для получения сведений о названии данной компоненты БПО, о номере версии и об авторских правах. При появлении данного ключа в командной строке, ассемблер выдает информационной сообщение:

```
Ассемблер для NM6403 1.30.(c)НТЦ Модуль 1996-1998. Все права защищены
```

и завершает работу. Работа ассемблера завершается без создания выходного файла.

2.5.4 Выдача сообщения о местоположении (ключ -p)

Параметр `-p` носит чисто информационный характер. Она используется для нахождения местоположения вызываемого ассемблера. Часто возникает ситуация, когда пользователь не знает, откуда именно вызывается программа. Известно только, что это один из каталогов, приведенных в списке общедоступных каталогов в файле `autoexec.bat` (`PATH=...`). В этой ситуации можно использовать ключ `-p`. Ассемблер, встретив в командной строке данный ключ выдаст на экран сообщение:

```
D:\NEURO\BIN\ASM.EXE,
```

и завершает работу. Выходной файл не создается.

2.6 Параметры управления порождением выходных файлов

Ассемблер в зависимости от ключей, используемых в качестве входных параметров, порождает следующие типы выходных файлов:

- `.elf` - объектный файл;
- `.lst` - файл - листинг исходного текста;
- `.crf` - файл со списком перекрестных ссылок;
- `.mlb` - макро-библиотека.

2.6.1 Задание выходного файла (ключ -o<имя_файла>)

Результатом работы ассемблера в режиме трансляции является выходной объектный файл. Его имя определяется путем задания в командной строке ключа `-o<имя_файла>`. Между ключом `-o` и именем файла не должно быть пробела. Под именем файла подразумевается полное имя, то есть полный путь к файлу. Если задано только имя, а путь опущен, файл будет создан в текущем каталоге. Если файл с таким именем в данном каталоге уже существовал, он будет замещен новым файлом без каких либо дополнительных предупреждений.

Если ассемблер не встречает в командной строке ключа, определяющего название выходного файла, то в качестве имени выходного файла используется имя файла, к которому добавляется расширение `.elf`.

2.6.2 Выдача листинга программы (ключ -l)

При появлении среди входных параметров ассемблера ключа `-l` порождается выходной файл, содержащий листинг программы. Он имеет расширение `.lst`.

Файл листинга создается дополнительно к объектному файлу и хранит справочную информацию следующего характера:

- смещение относительно начала сегмента;
- кодовый образ каждой команды;
- строки исходного текста программы.

Пример

В качестве примера приведем небольшой фрагмент листинга программы:

Рис. 2-3 Фрагмент листинга программы.

```
                          * begin ".text"
                          * <_Mirror>
                          *
00000000 4e000000 ffffffff *       nb1 = 0FFFFFFFFh; // Neuron boundaries
00000002 4e400000 ffffffff *       sb  = 0FFFFFFFFh; // Sinaps boundaries
                          *
00000004 3fe40000          *       push ar4, gr4;
00000005 3fe10000          *       push ar1, gr1;
00000006 3fe20000          *       push ar2, gr2;
                          *
00000007 50100000
00000008 47d40000 fffffff8 *       ar4 = ar7 - 8;
```

смещение | кодовые образы команд | строки исходного текста

2.6.3 Выдача перекрестных ссылок (ключ -x)

При добавлении в командную строку вызова ассемблера параметра `-õ` формируется файл с перекрестными ссылками. Он имеет

расширение `.crf` и порождается дополнительно к объектному файлу.

В файле перекрестных ссылок содержится список символов и их определений в следующем формате:

- имя символа;
- его значение;
- номер строки в файле с исходным текстом, где он определен;
- номера строк исходного текста, в которых есть ссылки на этот символ.

Пример

В качестве примера приведем небольшой фрагмент файла перекрестных ссылок:

Рис. 2-4 Фрагмент файла перекрестных ссылок.

```
****** List of sections ******
Section Name              Size
.values                   200
.bss.values               0
.text                     70

****** List of objects ******
Object Name               Type    Bind    Defined in    Offset
_Mirror                   LABEL   Global  .text         0
Matrix0                   VAR     Local   .values       0
Matrix1                   VAR     Local   .values       100
```

2.7 Работа в режиме макро-библиотекаря

Ассемблер помимо своей основной функции трансляции ассемблерных программ в объектные файлы позволяет работать в режиме макро-библиотекаря. В этом случае он вместо объектного файла порождает макробиблиотеку, куда из входной ассемблерной программы собраны все встречавшиеся в ней макросы.

2.7.1 Переход в режим макро-библиотекаря (ключ -m[<macrolib>])

Параметр `-m` переводит ассемблер в режим макро-библиотекаря. При этом входной файл проходит синтаксическую проверку, после чего содержащиеся в нем макросы переводятся в специальное промежуточное представление и записываются выходной файл.

Параметр `-m` может использоваться как с дополнительным параметром, так и без него. Дополнительный параметр содержит имя файла, в который будет записана макробиблиотека. Между `-m` и именем файла не должно быть пробела.

Если имя файла отсутствует, то в качестве макробиблиотеки будет использован файл, имя которого совпадает с именем входного ассемблерного файла, а расширение - `.mlb`.

Если параметр -m используется без дополнительного параметра `-a` (см. п. 2.7.2), то создается новый файл. Если файл с таким именем уже существовал, то он будет замещен новым без дополнительных предупреждающих сообщений.

2.7.2 Добавление макросов в макробиблиотеку (ключ -a)

Параметр `–a` используется для того, чтобы показать, что макросы, найденные во входном файле, должны быть добавлены в уже существующую макробиблиотеку.

Таким образом, если используется только ключ `-m`, то макробиблиотека создается заново независимо от того, существовал ли уже файл с данным именем. Если используется пара ключей `-m -a` , то макросы будут добавлены в существующую макробиблиотеку.

Примечание

В случае, если используется комбинация ключей `-m -a`, но при этом макробиблиотека не существует, то будет выдана ошибка. Это делается для того, чтобы избежать порождения ошибочных библиотек. Обычно пользователь, используя ключ `-a` подразумевает, что библиотека уже хранится на диске. Если она не найдена, то скорее всего был неверно указан путь.

2.8 Параметры управления предупреждениями

Ассемблер имеет развитые возможности управления предупреждениями. Все предупреждения разбиты на несколько групп, относящимся к разным режимам работы или типам обрабатываемых данных. Разбиение на группы приводится ниже в пункте 2.8.2 Управление предупреждениями по группам (ключ -W[+|-]<group>) на стр. 2-13.

2.8.1 Управление предупреждениями по их номерам (ключ -W[+|-]<num>)

Параметр `-W`, после которого идет номер предупреждения, позволяет включать/выключать предупреждение с заданным номером.

Если установлен знак "+", то предупреждение в случае возникновения появляется на экране.

При установленном знаке "`-`" предупреждение игнорируется и на экран не выдается.

Пример

В программе на языке ассемблера есть ошибка, возможно не приводящая к неправильной работе программы:

```
aaa: word = 8080808080808080h;
```

Ошибка заключается в том, что переменной, объявленной как 32-х битное слово, присваивается 64-х битная константа.

При компиляции ассемблерного файла, содержащего данную строку, будет выдано следующее сообщение:

```
>asm mirror.asm
Ассемблер для NM6403 1.30. (c) ЗАО НТЦ Модуль 1996-1998. Все права защищены.
"mirror.asm",19 : Warning ASM003: Initializer too big, truncated
```

То же самое будет, если в командную строку будет добавлен параметр `-W+3`. Однако, если в качестве входного будет использован параметр `-W-3`, то сообщение будет проигнорировано и не появится на экране.

По умолчанию все предупреждения находятся в активном состоянии (+) кроме предупреждения с номером `ASM200`, для которого установлено `-W-200`.

2.8.2 Управление предупреждениями по группам (ключ -W[+|-]<group>)

Параметр `-W` в сочетании с аббревиатурой, определяющей различные группы сообщений, позволяет управлять выводом предупреждений на экран.

Все предупреждения, выводимые на экран в процессе работы ассемблера, могут быть разделены на следующие группы:

- `all` - все предупреждения ассемблера;
- `debug` - все предупреждения, порождаемые при обработке отладочной информации;
- `object` - все предупреждения, порождаемые при создании объектов ассемблера, таких как неиспользуемые метки;
- `compile` - предупреждения, возникающие в процессе генерации кода;
- `librarian` - предупреждения, порождаемые ассемблером в режиме работы макро-библиотекарем.

Описание того, какие сообщения составляют те или иные группы, приводится в разделе 2.9 Сообщения об ошибках на стр. 2-14.

Пример:

Отключить все предупреждения кроме одного можно следующим образом:

```
asm.exe mytest.asm -W-<all> -W+3
```

Примечание

Действия параметров -W зависят от очередности их появления в командной строке, то есть последующий параметр отменяет предыдущий. Например, последовательность `-W-<all> -W+3` *отключает все предупреждения кроме 3-го, тогда как последовательность* `-W+3 -W-<all>` *отключает все, в том числе и предупреждение с номером 3.*

2.9 Сообщения об ошибках

В процессе работы ассемблер может выдавать три типа сообщений:

- предупреждения;
- ошибки;
- внутренние и фатальные ошибки.

При появлении предупреждений работа ассемблера не прекращается, поскольку возникающие в этой ситуации проблемы все же позволяют породить объектный код, хотя, возможно, и неправильно работающий.

В случае возникновения любого типа ошибки выходной файл не порождается, поскольку либо нет возможности для порождения кода, либо порожденный код будет заведомо неправильным и не сможет быть исполнен. Ошибки делятся на обычные, связанные с неправильным написанием программы, и фатальные, когда, даже в случае отсутствия ошибок в самой программе она не может быть откомпилирована, поскольку, например, отсутствует файл с макробиблиотеками, или обнаруживается нехватка дискового пространства.

Для выдачи сообщений на экран ассемблер использует форматированную строку. Ее формат является единым для всего комплекса БПО и имеет следующий вид:

"имя_файла", [номер_строки] тип_ошибки номер_ошибки: сообщение_об_ошибке.

Номер строки не указывается для внутренних и фатальных ошибок.

Под номер ошибки отведено три цифры, то есть он лежит в интервале от 0 до 999. Между типами ошибок номера распределены следующим образом:

Табл. 2-5 Диапазоны номеров для различных типов сообщений.

Номера ошибок	Описание
`0 - 399`	Предупреждения.
`400 - 799`	Ошибки.

800 - 949	Внутренние ошибки.
950 - 999	Фатальные ошибки.

Ниже приводится список сообщений, выдаваемых ассемблером при появлении различных типов ошибочных ситуаций.

2.9.1 Предупреждения

Сообщения данного типа указывают на возможность наличия в обрабатываемой программе логических или семантических неточностей. Вывод данного сообщения не влияет на правильность формирования ассемблером выходных файлов.

Сообщения компиляции (группа `compile`).

ASM001 `"Имя секции в конструкции end отличается от имени при открытии :"`

Секция в ассемблере всегда заключена в специальные скобки. Открывающие скобки задаются псевдокомандами: .begin, .data, .nobits. После открывающей скобки расположено название секции. В качестве скобки, закрывающей секцию, используется псевдокоманда .end. После нее также должно располагаться имя секции. Оно должно совпадать с именем при открытии. Если имена секции у открывающей и закрывающей скобок не совпадают, возникает данное предупреждение (дисциплинарная конструкция). В случае несовпадения имя секции у закрывающей скобки игнорируется.

ASM002 `"Используйте псевдофункции 'float' или 'double'"`

Переменные типа `float` или `double` представлены в памяти процессора в формате IEEE 754. Ассемблер автоматически переводит константы типа 1.5 или 2.7E-3 в необходимый вид. Однако для того, чтобы такой перевод был корректно осуществлен, необходимо использовать псевдофункции `float` или `double`, распознав которые, ассемблер сделает соответствующее преобразование. То есть, правильная запись констант с плавающей точкой имеет вид: `float(1.5)` или `double(2.7E-3)`. В противном случае константы данного типа будут **заменены нулевыми** константами.

ASM003 `"Значение константы инициализации слишком`

	велико, усечено"

Такое сообщение обычно возникает при попытке инициализировать 32-х разрядную переменную 64-х битной константой. После усечения остается только младшая часть константы, старшая игнорируется, например: `0123456789ABCDEFhl -> 89ABCDEFh`.

ASM004	"Значение константы инициализации слишком мало, расширено"

Такое сообщение обычно возникает при попытке инициализировать 64-х разрядную переменную 32-х битной константой. Расширение происходит за счет прописывания нулями старшей части константы, например: `12345678h -> 0000000012345678hl`.

ASM005	"Псевдофункции loword/hiword не могут применяться к адресным выражениям"

Разрядность адресных выражений не превышает 32 бит, тогда как псевдофункции `loword`/`hiword` предназначены для доступа к младшей/старшей части 64-х разрядного слова соответственно.

ASM006	"Имя макроса в конструкции end отличается от имени при открытии :"

Макрос в ассемблере всегда заключен в специальные скобки. Открывающие скобки задаются псевдокомандой: .macro. После открывающей скобки расположено название макроса. В качестве скобки, закрывающей секцию, используется псевдокоманда .end. После нее также должно располагаться имя макроса. Оно должно совпадать с именем при открытии. Если имена макроса у открывающей и закрывающей скобок не совпадают, возникает данное предупреждение (дисциплинарная конструкция). В случае несовпадения имя макроса у закрывающей скобки игнорируется.

ASM100	"Для получения правильного результата регистр результата не должен совпадать с регистрами множителя и множимого"

Для операции умножения, выполняемой аппаратно, существует требование, согласно которому результат операции не может быть записан ни в регистр, в котором

располагалось множимое, ни в регистр, где хранился множитель(см. Описание языка ассемблера для NM6403). Сообщение предупреждает, что умножение будет выполнено неправильно.

ASM101 `"Операция умножения при выполнении использует регистр результата, по умолчанию подставляем gr0"`

Если не был указан регистр результата при умножении, например, в строке ассемблерной программы записано: `gr2*:gr7`, то по умолчанию результат будет сохранен в регистре `gr0`.

ASM102 `"Допустима только 1, используем 1"`

В языке ассемблера есть некоторые конструкции, в которых может быть использована только единица, например: `gr2 = gr0 - 1`. В скалярных или векторных арифметических операциях константа не может принимать участие. Запись же типа `gr0 - 1` рассматривается не как операция с константой, а как унарная арифметическая операция модификации регистра общего назначения, и соответствует определенной инструкции процессора. Если в такую конструкцию по ошибке подставлено другое число, то оно автоматически заменяется на 1.

ASM103 `"Только регистр gr7 может быть множителем, используем gr7"`

При операции умножения, выполняемой аппаратно, существует требование, согласно которому множитель всегда должен находиться в регистре `gr7`. Если в обрабатываемой команде вместо `gr7` стоит какой-либо другой регистр, от будет заменен на `gr7`.

ASM104 `"Допустим только 0, используем 0"`

В языке ассемблера есть некоторые конструкции, в которых может быть использован только 0, например: векторные инструкции, где 0 выступает в качестве нулевого операнда: `with 0 + data`. В данном случае ноль является признаком того, что в процессоре автоматически порождается нулевой вектор, который позволяет пропустить `data` в `afifo` без изменений. Если в такую конструкцию по ошибке подставлено

другое число, то оно автоматически заменяется на 0.

Сообщения отладочной информации (группа debug).

ASM200 `"Второй аргумент не используется"`

По умолчанию данное сообщение отключено. Оно может выдаваться при анализе отладочной информации формата DWARF, когда вместо одного требуемого параметра псевдокоманде подается два. При этом второй параметр игнорируется.

ASM201 `"Индекс cie должен совпадать с порядковым номером данной '.debug_frame_cie'"`

Сообщение возникает при несовпадении индекса `cie` с порядковым номером отладочной псевдокоманды. `.debug_frame_cie`.

ASM202 `"Индекс директории должен совпадать с порядковым номером команды '.debug_source_directory'"`

Сообщение возникает при несовпадении индекса каталога с порядковым номером отладочной псевдокоманды `.debug_source_directory`.

ASM203 `"Индекс файла должен совпадать с порядковым номером команды '.debug_source_file'"`

Сообщение возникает при несовпадении индекса файла с порядковым номером отладочной псевдокоманды `.debug_source_file`.

ASM204 `"В данной версии команда '.debug_macro_def' не реализована, пропускаем"`

Поддержка отладочной информации для макросов в данной версии БПО не осуществляется.

ASM205 `"В данной версии команда '.debug_macro_undef' не реализована, пропускаем"`

Поддержка отладочной информации для макросов в данной версии БПО не осуществляется.

ASM206	`"В данной версии команда '.debug_macro_start_file' не реализована, пропускаем"`

Поддержка отладочной информации для макросов в данной версии БПО не осуществляется.

ASM207	`"В данной версии команда '.debug_macro_end_file' не реализована, пропускаем"`

Поддержка отладочной информации для макросов в данной версии БПО не осуществляется.

ASM208	`"Идентификатор атрибута должен быть числом"`

В качестве идентификатора атрибута используется символьная информация.

ASM209	`"Значение атрибута должно быть адресом"`

Параметр отладочной псевдокоманды имеет неверный тип.

ASM210	`"Абстрактная команда должна состоять из тройки элементов"`

Неправильно задана структура псевдокоманды.

ASM211	`"Может быть только одна '.debug_root_die'"`

В тексте программы встречено более одной псевдокоманды `.debug_root_die`. Вторая и последующие псевдокоманды `.debug_root_die` игнорируются.

ASM212	`"Последовательность команд '.debug_*die*' должна начинаться с '.debug_root_die'"`

Отсутствует псевдокоманда `.debug_root_die`, определяющая начало последовательности команд, определенных схемой `.debug_*die*`.

ASM213	`"Тег die должен быть числом"`

Тег `die` возможно определен, как символьная константа.

ASM214	`"Первый аргумент операции должен быть адресом"`

Первый аргумент псевдокоманды имеет тип, отличный от адреса.

ASM215	`"Оба аргумента должны быть числами"`

Один или оба аргумента псевдокоманды определены как символьные константы.

ASM216	`"Код операции должен быть числом"`

В качестве кода операции использовано символьное выражение.

ASM217	`"И код операции и оба аргумента должны быть числами"`

В качестве кода операции и/или аргумента использовано символьное выражение.

ASM218	`"Неверное количество аргументов"`

В строке аргументов их количество превышает необходимое или недостает до необходимого.

ASM219	`"Неверный тип аргументов"`

В качестве аргумента поданы данные неверного типа.

ASM220	`"Последовательности '.debug_line' не могут быть вложенными"`

Последовательности псевдоопераций, описывающие номера строк исходного текста не могут быть вложенными.

ASM221	`"В первой '.debug_line' должен быть указан индекс исходного файла"`

Первая псевдооперация `.debug_line`, описывающая номер строки, должна хранить индекс исходного файла, в котором происходит

ASM222	`"Пропущена команда '.debug_start_sequence'"`

Встречены псевдокоманды обработки последовательности, хотя начальная псевдокоманда `.debug_start_sequence` не была задана.

ASM223	`"Байты block'a должны быть числами"`

Возможно в структуре блока были обнаружены байты, заданные символьными константами.

Сообщения формирования объектного файла (группа `object`)

ASM300	`"Неиспользованная метка"`

В программе на языке ассемблера была декларирована метка, однако не найдена ни одна ссылка на нее, поэтому она будет проигнорирована.

ASM301	`"Генерация отладочной информации невозможна"`

Данное предупреждение появляется в том месте, где найдена первая ошибка при разборе отладочной информации. Начиная с этого места продолжится генерация объектного кода, однако вся отладочная информация будет проигнорирована.

Сообщения макро-библиотекаря (группа `librarian`)

ASM350	`"Макрос уже есть, пропускаем :"`

В макробиблиотеке уже существует макрос с данным именем, поэтому вновь встреченный макрос будет пропущен.

2.9.2 Ошибки

Данный тип сообщений указывает на наличие в обрабатываемом файле некоторой ошибки. Данная ошибка не вызывает немедленного прекращения работы ассемблера, но в случае

появления сообщений данной категории выходные файлы не порождаются.

ASM400 `"Слишком длинная константа"`

Данная ошибка возникает в случае, когда объявленная константа не помещается в 64 бита.

ASM401 `"Неправильный формат десятичного числа"`

Под неправильным форматом десятичного числа подразумевается, что в его состав входят символы, не принадлежащие цифровому ряду 0-9.

ASM402 `"Неправильный формат двоичного числа"`

Под неправильным форматом двоичного числа подразумевается, что в его состав входят символы, отличные от 0 и 1.

ASM403 `"Неправильный формат восьмеричного числа"`

Под неправильным форматом восьмеричного числа подразумевается, что в его состав входят символы, не принадлежащие цифровому ряду 0-7.

ASM404 `"Неправильный формат шестнадцатеричного числа"`

Под неправильным форматом шестнадцатеричного числа подразумевается, что в его состав входят символы, не принадлежащие цифровому ряду 0-9, и не соответствующие буквам A, B, C, D, E, F.

ASM405 `"Недопустимый символ"`

Под недопустимым символом в языке ассемблера понимается символ, не принадлежащий следующему подмножеству символов: строчные и прописные латинские буквы, десятичные цифры, а также следующие символы:

`. , / \ : ; " ' ` [ ] - + = & ! < > ( ) * { } _`.

ASM406 `"Недопустимое условие"`

Под недопустимым условием, приводящим к данной ошибке, в языке ассемблера понимается следующее:

- если после символов u> без пробела стоит любой символ, кроме =. То есть должно быть: u>=.
- если после символов <> без пробела стоит любой символ, кроме 0. То есть, должно быть: <>0.

ASM407 `"После операции квотирования должен быть идентификатор"`

Символ ``` `` ``` (обратный1 апостроф) в языке ассемблера может использоваться только в операциях квотирования, то есть когда пользователь желает использовать в качестве идентификатора служебное слово, оно должно в обязательном порядке предваряться символом ``` `` ```. Если между данным символом и идентификатором стоит пробел, или если идентификатор отсутствует, возникает данная ошибка.

ASM408 `"Не совпадают завершители строки"`

Когда некоторая строка заключается в двойные кавычки, ассемблер вправе ожидать, что если был встречен знак открывающих кавычек ", то до конца строки должны появиться закрывающие двойные кавычки. Если закрывающие кавычки не найдены, или вместо них встречены одиночные, возникает данная ошибка.

ASM409 `"Неправильное объявление переменной"`

Данная ошибка возникает, если переменная объявлена вне какой-либо программной секции, и при этом она не является переменной типа `common` или `extern`.

ASM410 `"Неправильная инициализация"`

Данная ошибка возникает, когда ассемблер обнаруживает попытку инициализации переменной типа `common` или `extern`. Переменные данных типов в силу своей природы не могут быть инициализированы.

ASM411 `"Неверный тип переменной - предполагается WORD"`

.

ASM412 `"Неправильное объявление структуры"`

Данная ошибка возникает, когда имя структуры, определенное рядом с открывающей скобкой `struct` не совпадает с именем при закрывающей скобке `end`, например:

```
struct AAA
  A: word;
  B: long;
end AA;
```

ASM413 `"Поле структуры не найдено"`

Данная ошибка возникает при попытке обращения к несуществующему полю используемой структуры, например:

```
ar0 = offset(AAA, C);,
```

хотя у структуры AAA есть только поля A и B.

ASM414 `"Неизвестное имя типа"`

ASM415 `"Неправильное использование слова OWN"`

Данная ошибка возникает при использовании ключевого слова own вне тела макроса, что недопустимо, поскольку это слово используется для меток, объявленных только внутри макроса.

ASM416 `"Неправильное имя библиотеки"`

Данная ошибка возникает в том случае, когда после ключевого слова `import` не задано имя макробиблиотеки, или задано имя несуществующей библиотеки.

ASM417 `"Неверный размер массива"`

Причиной возникновения данной ошибки является неверное указание размера массива, когда он определен равным нулю или отрицательному числу.

ASM418	`"Неправильный счетчик в дубликаторе"`

Данная ошибка возникает, когда аргументом дубликатора `dup`, определяющим количество повторений шаблона, является нуль, или отрицательное число, например: `80808080h dup -10`.

ASM419	`"Неправильный индекс массива"`

Данная ошибка возникает при попытке обратиться за границы массива. Ошибка может быть обнаружена только при явном задании индекса элемента массива. Например, если в отмеченной строке задано обращение к элементу массива с индексом -1 или 100, притом, что корректные значения индексов лежат в диапазоне от 0 до 99.

ASM420	`"Объект не найден"`

Внутренняя ошибка.

ASM421	`"Неверный объект"`

Внутренняя ошибка.

ASM422	`"Неправильная регистровая пара"`

Когда используется регистровая пара, например, для чтения из памяти 64-х разрядного слова, адресный регистр и регистр общего назначения должны иметь одинаковый индекс: `ar0, gr0` или `[ar3+=gr3]`.

ASM423	`"Неправильное константное выражение"`

Нарушены правила формирования константного выражения. Ассемблер не может вычислить его, поэтому выдает ошибку.

ASM424	`"Использование '.endif' без '.if'"`

Псевдокоманды `'.endif'` и `'.if'` всегда должны встречаться парами. Отсутствие `'.if'` при наличии `'.endif'` вызывает ошибочную ситуацию.

ASM425	`"Рекурсивное использование '.repeat'"`

Ассемблер не допускает вложенных псевдокоманд `.repeat`.

ASM426	`"Неправильный метод адресации"`

Был использован несуществующий метод адресации. Список используемых методов адресации приведен в документе **Программное обеспечение NeuroMatrix® NM6403. Описание языка ассемблера.**

ASM427	`"Адресные регистры из разных групп"`

В данной ассемблерной команде не могут быть использованы адресные регистры из разных групп. Регистры `ar0 - ar3` составляют одну группу, регистры `ar4 - ar7` вторую. Более подробно см. **Программное обеспечение NeuroMatrix® NM6403. Описание языка ассемблера.**

ASM428	`"Регистр назначения должен быть адресным"`

Данная ошибка возникает при попытке использовать адресный регистр для выполнения арифметических операций, например: `gr4 = ar5+gr5`. Такая операция невозможна, вместо нее может быть использована операция модификации адресного регистра: `ar4 = ar5+gr5`.

ASM429	`"Неправильный операнд команды сдвига"`

Данная ошибка возникает, когда неправильно задан оператор сдвига, например, в операциях циклического сдвига в качестве величины сдвига используется число, отличное от `1`.

ASM430	`"Сдвиг через carry должен выполняться только на единицу"`

Данная ошибка возникает, когда неправильно задан размер сдвига в операциях сдвига через флаг `carry`, где в качестве величины сдвига используется число, отличное от `1`.

ASM431 `"Должно быть число"`

В счетчике повторений, определяемом псевдокомандой `.repeat`. количество повторов задается константным цифровым значением, например, `.repeat 32` или `.repeat 100`. То же относится к счетчику повторений, используемому в векторных командах, диапазон значений которого лежит в пределах `1-32`.

ASM432 `"Счетчик не находится в пределах (1-32)"`

Счетчик повторений векторной команды должен лежать в пределах от 1 до 32. Этот диапазон определяется глубиной очередей `FIFO` и внутренней структурой векторного процессора.

ASM433 `"Синтаксическая ошибка"`

Самая распространенная ошибка, сообщающая, что в строке, где она произошла, находится синтаксическая конструкция, которая не может быть обработана ассемблером.

ASM434 `"Переопределение метки"`

Метка, определенная в данной строке, была определена раньше в другом месте, например, выше по тексту. Эта ошибка может возникнуть так же, когда в макросе используется метка, перед которой отсутствует ключевое слово `own`.

ASM435 `"Рекурсивное использование макроса"`

Ассемблер допускает вложенность макросов, когда в теле одного макроса содержится вызов другого, и так далее. Однако, если в этой цепочке вызовов встретится имя макроса, уже упоминавшегося в ней ранее, то будет порождена ошибка, поскольку в настоящей версии ассемблер не поддерживает рекуррентный вызов макросов.

ASM436 `"Количество параметров не совпадает в вызове макроса"`

При разработке макроса программист определяет количество передаваемых ему параметров. Если при вызове макроса количество параметров не соответствует тому, что было определено при разработке, вызывается данная ошибка.

ASM437 `"Повторное определение"`

В данной строке встречается повторное определение идентификатора.

ASM438 `"Неправильный тип поля"`

Внутренняя ошибка.

ASM439 `"Неправильный тип переменной"`

Внутренняя ошибка.

ASM440 `"Неопределенная метка"`

Если в программе встречен переход на метку, которая хотя и объявлена, но осталась неопределенной, то порождается данная ошибка. Синтаксически определение метки выглядит следующим образом: <`MyLabel`>.

ASM441 `"Инициализация переменной в секции .nobits"`

Секция `.nobits` введена в ассемблер для того, чтобы хранить неинициализированные переменные. Если в секции данного типа встречается инициализация, возникает ошибка. Необходимо перенести инициализируемую переменную в секцию соответствующего типа.

ASM442 `"Повторное использование идентификатора"`

Данная ошибка возникает при попытке объявления локальной переменной, имеющей имя, которое совпадает с именем объявленной ранее глобальной переменной.

ASM443 `"Неправильное определение поля"`

.

ASM444 `"Переопределение макроса"`

В данной строке определяется макрос, имя которого совпадает с уже определенным ранее макросом.

ASM445 `"Неопределенный макрос"`

В данной строке вызывается макрос, место определения которого не найдено. Возможно макрос определен в другом файле. Используйте команду `import`<имя_файла>, в которую подставляется имя файла, в котором определен искомый макрос.

ASM446 `"Неопределенная константа"`

В данной строке используется константа, которая не была определена заранее.

ASM447 `"Неопределенная переменная"`

В данной строке используется переменная, которая не была определена заранее.

ASM448 `"Метка не должна иметь тип 'common'"`

Ошибка возникает в том случае, когда перед меткой обнаружено ключевое слово `common`, которое не может использоваться с метками, а только с переменными.

ASM449 `"Неправильный формат числа с плавающей точкой"`

Поддерживаются только следующие форматы записи вещественных чисел:

`float(5.6)`- вещественное число одинарной точности,

`float(123.00e12)`- число с плавающей точкой (32 бита),

`double(123.00e12)`- длинное число с плавающей точкой (64 бита).

ASM450	`"Неопределённая die :"`

Ошибка связана с тем, что `die`, используемая при записи команд отладочной информации, не была заранее определена.

ASM451	`"Потеряна ';' после .endrepeat"` `"Потеряна ';' после .endif"`

Часто символ ';' теряется после псевдокоманд `.endrepeat`, или `.endif`. Данная ошибка является напоминанием.

2.9.3 Внутренние и фатальные ошибки.

Сообщение о внутренней ошибке свидетельствует о внутреннем сбое ассемблера и вслед за ним следует немедленное прекращение работы программы. При возникновении такой ошибки следует обратится к разработчикам ассемблера (см. в "Предисловии" раздел "Как оформлять замечания и предложения").

Сообщение о фатальной ошибке сигнализирует о серьезной ошибке, возникшей в ходе работы ассемблера. После вывода такого сообщения ассемблер аварийно завершает работу. Даже в случае, когда компилируемый файл не содержит синтаксических и семантических ошибок, может возникнуть фатальная ошибка. Она может быть, например, связана с нехваткой места на диске или в памяти.

ASM950	`"Не могу открыть файл :"`

Возможно файл имеет установленный флажок `read-only`, или используется другим приложением.

ASM951	`"Ошибка стека лексического анализатора"`

Стек лексического анализатора оказался переполнен из-за нехватки места в оперативной памяти в процессе компиляции. Необходимо увеличить размер оперативной памяти.

ASM952	`"Нехватка памяти"`

Недостаточно памяти на диске и в оперативной памяти, необходимого для компиляции данного файла.

ASM952	`"Внутренняя ошибка"`

При возникновении данной ошибки необходимо обратиться к разработчикам БПО (см. в "Предисловии" раздел "Как оформлять замечания и предложения").

3 Редактор связей

3.1 Введение

Данная глава описывает интерфейс редактора связей, его входные управляющие параметры и различные режимы работы. Приводятся подробная информация о каждом ключе, набор ключей по умолчанию при запуске редактора связей, организация работы при помощи командного файла.

Глава содержит описание структуры файла конфигурации, позволяющего управлять размещением различных частей исполняемой программы в памяти процессора при произвольных вариантах конфигурации физической памяти.

Приводится полный список ошибок, возникающих на экране в процессе работы редактора связей, причины их возникновения и способы устранения.

Помимо этого, глава содержит примеры работы с редактором связей, начиная от создания командного файла и файла конфигурации и заканчивая анализом полученного файла-карты памяти.

3.1.1 Функциональные возможности реактора связей

Редактор связей при обработке входных объектных файлов выполняет следующие функции:

- объединяет секции с одинаковыми именами и создает для них собственные таблицы перемещений, необходимые для перенастройки ссылок на конкретную конфигурацию памяти вычислительного устройства;
- в процессе построения исполняемых файлов с настройкой на конкретную конфигурацию вычислительного устройства вычисляет адреса символов и секций, настраивает все ссылки, хранящиеся в таблицах перемещений;
- объединяет загружаемые секции в программные сегменты для ускорения и упрощения загрузки программы в память вычислительного устройства;
- разрешает неопределенные внешние ссылки между входными файлами;
- удаляет из выходного файла неиспользуемые программой секции и символы, а также отладочную информацию;
- выдает информацию о найденных в процессе редактирования связей ошибках.

3.1.2 Настройка на различные варианты конфигурации памяти

Редактор связей поддерживает различные варианты конфигурации памяти вычислительного устройства. Для этих целей разработан Си-подобный язык, с помощью которого в специальном файле, называемом файлом конфигурации, описываются диапазоны рабочих адресов, доступных процессору, задаются адреса загрузки программных сегментов, распределение загружаемых секций по сегментам, их взаимное расположение. Этот язык содержит три основных директивы MEMORY, SEGMENTS и SECTIONS, которые позволяют сформировать информацию для редактора связей, позволяющую правильно настроить адреса и ссылки в исполняемом файле. Более подробно см. раздел 3.11 Файл конфигурации на стр. 3-27.

3.1.3 Типы порождаемых файлов

Редактор связей позволяет создавать три типа выходных файлов формата ELF:

- **абсолютный исполняемый файл** - представляет собой набор программных сегментов. Каждый сегмент является образом определенного участка памяти процессора. Он имеет адрес загрузки, размер отображаемого участка и последовательность кодов, которая должна быть помещена в данную область памяти. При этом все ссылки на символы заменены на абсолютные адреса этих символов;
- **исполняемый перемещаемый файл** - отличается от абсолютного исполняемого тем, что для него не заданы заранее адреса расположения в памяти процессора. Адрес каждой секции определяется лишь на этапе загрузки программы, поэтому в рассматриваемом файле отсутствует понятие загружаемого сегмента. Каждая загружаемая секция имеет свою таблицу перемещений, а все ссылки на символы и участки кода разрешаются в момент загрузки в процессор, когда для секции уже определен адрес размещения в памяти. Поэтому в перемещаемый исполняемый файл в отличии от абсолютного входят таблица символов и таблицы перемещений;
- **объектный файл** - является результатом объединения входных файлов. Он содержит объединенную таблицу символов, обобщенные секции, пересчитанные таблицы перемещений. Объектный файл в отличии от исполняемого может содержать в себе неопределенные символы.

3.1.4 Результаты работы редактора связей

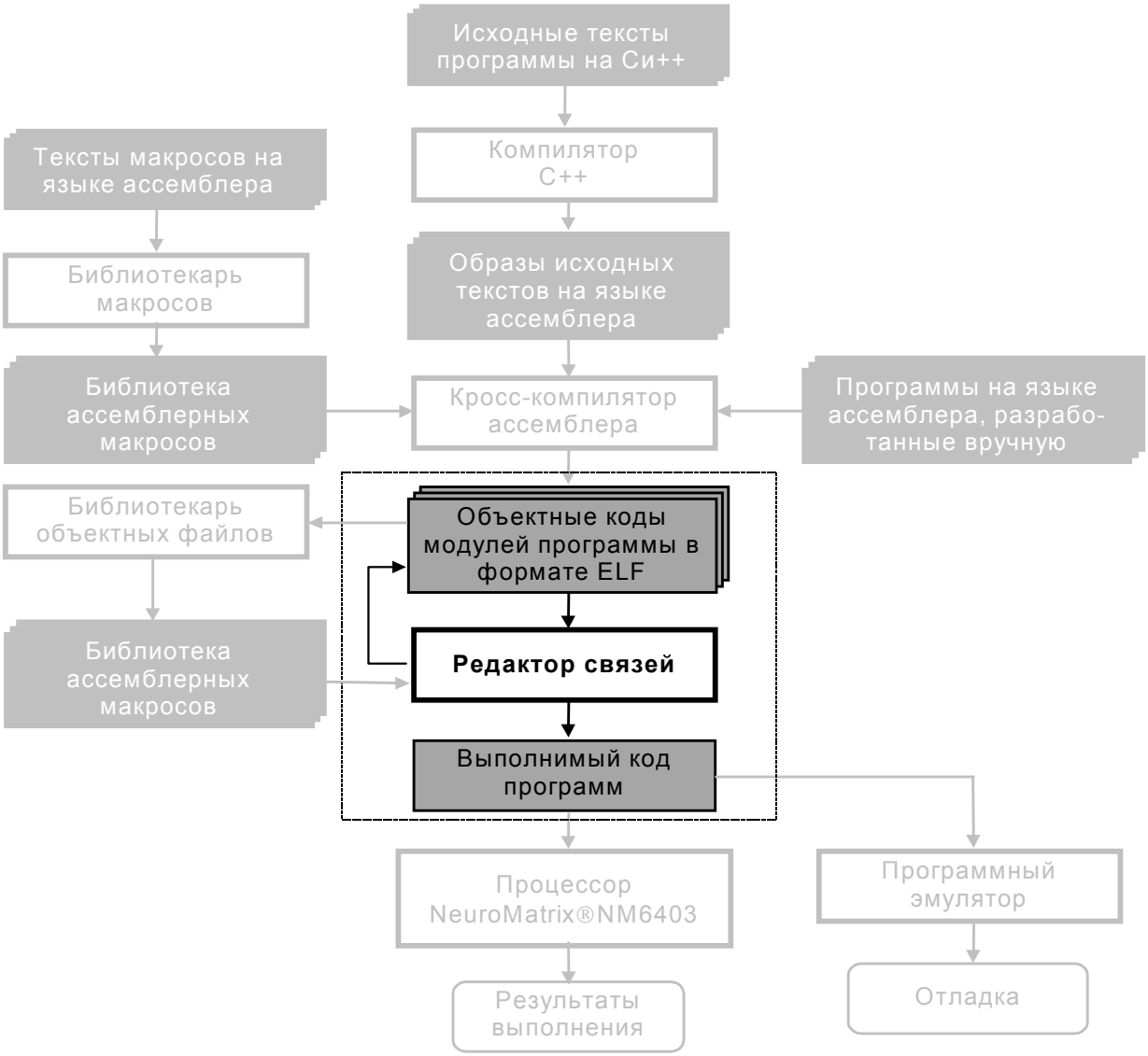
Редактор связей, в случае удачного окончания работы, выдает соответствующее сообщение и возвращает значение 0. Если в процессе работы редактора произошла ошибка, то на экране

появляется сообщение об ошибке, а работа редактора завершается аварийно с возвращаемым значением 1 или более, в зависимости от количества найденных ошибок.

3.2 Положение редактора связей в структуре базового ПО

На Рис. 3-1 показана роль редактора связей в процессе разработки прикладных программ для процессора NeuroMatrix® NM6403. Редактор связей обрабатывает несколько типов входных файлов, в том числе объектные файлы, библиотеки, командный файл, файл конфигурации. Редактор связей создает абсолютный исполняемый или исполняемый перемещаемый файл, предназначенный для загрузки в память процессора или в программный эмулятор.

Рис. 3-1 Положение редактора связей среди других компонентов БПО.



3.3 Характеристики редактора связей

Редактор связей представляет собой консольное приложение Windows95/NT.

Редактор связей ориентирован на работу только с файлами формата ELF. Он полностью поддерживает данный формат за исключением создания и обработки динамических библиотек. Однако редактор связей предназначен для обработки объектных файлов только для процессора NeuroMatrix® NM6403, так как формат ELF не специфицирует алгоритмы вычисления перемещений. Этот алгоритм является машинозависимым, то есть специально определяется для каждого типа процессора и зависит от его размерности данных и от поддерживаемых им способов адресации.

Редактор связей обладает некоторыми оптимизирующими возможностями. Его оптимизация относится только к оптимизации размера объектных файлов. Специальный алгоритм, заложенный в него, позволяет удалять неиспользуемые программой секции и символы, заменять ссылки на локальные символы ссылками на начало секций, в которых они определены. Это позволяет сократить объем кода программы, а значит и время ее обработки.

В редактор связей встроены средства самоконтроля. При возникновении сбойной ситуации он выдает диагностику на экран компьютера, или в файл, переданный в качестве параметра. Наряду с пользовательскими ошибками, редактор информирует программиста и о внутренних ошибках, которые могли бы произойти во время его работы. Если возникает внутренняя ошибка, необходимо обратиться к разработчикам редактора связей и передать им те файлы, при работе с которыми произошла данная ошибка и информацию о комбинации входных параметров.

3.4 Формат вызова редактора связей

Формат вызова редактора связей:

```
linker [параметры] [файл1] ... [файлN] [параметры]
```

`linker` Имя файла, содержащего исполняемый код редактора связей.

`Файл1...файл2` Входные объектные файлы и библиотеки. Могут располагаться в произвольном порядке. Расширение в имени файла значения не имеет. Редактор определяет объектные файлы по магическому числу, содержащемуся в заголовке файла. По этому же числу одиночные объектные файлы отличаются от

библиотек.

`Параметры` параметры управления редактором (с префиксом "`-`"). Могут располагаться в произвольном месте командной строки, в произвольной последовательности. (Более подробно обсуждаются ниже.)

Существует два способа запуска редактора связей:

- Запуск редактора с заданием параметров и имен файлов в командной строке. Следующий пример содержит вызов редактора для редактирования связей двух объектных файлов `file1.elf` и `file2.elf`. В результате работы редактора будет получен абсолютный исполняемый файл. Параметр `-o` устанавливает имя выходного файла: `result.abs`.

  ```
  linker file1.obj file2.obj -oresult.abs;
  ```

- Запуск редактора связей с указанием командного файла, содержащего параметры и имена входных объектных файлов. Командный файл представляет собой текстовый файл, каждая строка которого содержит одну или несколько параметров редактора связей. Для указания редактору, что параметры необходимо брать из командного файла, в командную строку добавляется параметр "`@file`", где `file` - имя командного файла. Командный файл может выглядеть так:

  ```
  -oresult.abs
  file1.elf
  file2.elf
  ```

  При этом запуск редактора связей с командным файлом в качестве входного параметра будет выглядеть следующим образом:

  ```
  linker @file.cmd
  ```

  Командный файл является удобной формой записи входных параметров редактора связей. Он особенно необходим, когда редактору необходимо собрать воедино большое количество объектных файлов, а командная строка не вмещает всю входную информацию. Однако, редактор связей позволяет комбинировать в командной строке как обращение к командному файлу, так и задание отдельных параметров. Например, командный файл может выглядеть так:

  ```
  file1.elf file2.elf
  ```

  При этом запуск редактора связей будет выглядеть следующим образом:

  ```
  linker @file.cmd -oresult.abs, или
  ```

```
linker -oresult.abs @file.cmd
```

3.5 Список параметров управления редактированием связей

Для управления работой редактора связей используется набор параметров (ключей).

Все параметры начинаются с символа "-".

В названиях параметров различаются прописные и строчные буквы.

Они могут быть указаны в командной строке или в командном файле. Последовательность, в которой расположены параметры, значения не имеет. Параметры разделяются между собой пробелами. Если за параметром следует аргумент, то он записывается без пробела, например, `-ofile.abs`. В противном случае редактор связей воспримет данное имя, как имя входного объектного файла, а не обнаружив его или попытавшись открыть, выдаст ошибку. При этом работа редактора завершится аварийно.

Ниже приводится сводная таблица параметров управления редактором связей:

Табл. 3-1 Параметры общего назначения.

`-h или -?`	Вывод справочной информации
`-q[n][=имя_файла]`	Режим молчания. n = 0..2. В зависимости от n в поток выдается вся информация, только сообщения об ошибках, или ничего. Если указано имя файла, то информация будет перенаправлена в него.
`-t`	Вывод заголовка редактора связей, информации о версии.
`-p`	Вывод полного собственного пути.
`-o<имя_файла>`	Задание имени выходного файла (если расширение отсутствует, оно определяется в соответствии с типом выходного файла).
`@<имя_файла>`	Задание имени командного файла.

Табл. 3-2 Тип выходного файла.

`-abs или -a`	Абсолютный исполняемый файл.
`-rel или -r`	Исполняемый перемещаемый файл.
`-elf или -e`	Объектный файл.

Табл. 3-3 Специфичные параметры редактора связей.

-c<имя_файла>	Назначение файла конфигурации
-m<имя_файла>	Создание карты памяти абсолютного исполняемого файла.
-l<имя_каталога>	Путь к каталогу библиотечных файлов.
-d0	Запрещение удаления неиспользуемых секций
-d1..3	Сохранение отладочной информации в режиме удаления неиспользуемых секций.
-d4	Удаление всей отладочной информации и неиспользуемых секций.
-heap=<размер>	Размер кучи в локальной памяти (в килословах).
-heap1=<размер>	Размер кучи в глобальной памяти (в килословах).
-stack=<размер>	Размер стека (в килословах).
-start=<имя_метки>	Точка входа в программу.
-asm	Отключение инициализации статических глобальных переменных в языке Си++.
-addr=<адрес>	Адрес сегмента по умолчанию, создаваемого в отсутствие файла конфигурации.

3.6 Параметры общего назначения

К параметрам общего назначения относятся ключи, имеющие одинаковый смысл, и выполняющие одни и те же действия для большинства компонент БПО процессора NeuroMatrix® NM6403.

3.6.1 Выдача справочной информации (ключи -h, -?)

При запуске редактора связей без входных параметров, или с параметром -h, или -? на экран монитора выводится справочная информация обо всех ключах, определяющих поведение редактора связей и результат его работы. На экране появится следующая информация:

```
Редактор связей для NeuroMartix® NM6403 * v1.7 * (c) 1996,98 * НТЦ Модуль.
Формат:  linker [флаги] <файл1> ... <файлN> [флаги]
Параметры:
-t                - выдать титульное сообщение редактора связей
-p                - выдать местоположение редактора связей
-abs (-a)         - создание абсолютного исполняемого файла
-rel (-r)         - создание исполняемого перемещаемого файла
-elf (-e)         - создание объектного файла
```

```
-o<имя_файла>       - имя выходного файла
-c<имя_файла>       - имя файла конфигурации
-m<имя_файла>       - имя файла-карты памяти
-l<имя_каталога>    - путь к каталогу, хранящему библиотеки
-start=<имя_метки> - имя точки входа в программу
-stack=<размер>     - размер стека
-heap=<размер>      - размер локальной кучи
-heap1=<размер>     - размер глобальной кучи
-d<n>               - 'степень удаления неиспользуемых секций'; n = 0..4
                      0 - ничего не удалять
                      1 - помечать DEBUG-секции до общего анализа
                      2 - анализ неиспользуемых секций
                      3 - помечать DEBUG-секции после общего анализа
                      4 - помечать только используемые DEBUG-секции
                      Когда <n> пропущено, предполагается 4
                      По умолчанию также 4 (полная очистка)
-q[n][=имя_файла]   - установка режима диагностики; n = 0..2
                      0 - вся информация выдается на экран
                      1 - выдаются только сообщения об ошибках
                      2 - ничего не выдается
                     если задано <имя_файла>, выход перенаправляется в него
-asm                - отключение инициализации глобальных статических
                      объектов Си++
-addr=<адрес>       - адрес сегмента по умолчанию
```

При появлении в командной строке ключа `-h` или `-?` работа редактора связей завершается до редактирования независимо от наличия других параметров. При этом выходной файл не создается.

3.6.2 Режимы молчания (ключ -q[n][=имя_файла])

В зависимости от параметра n редактор связей выдает информацию следующим образом:

- в случае `n = 0` редактор связей выдает пользователю всю порождаемую информацию, а именно титульный заголовок, предупреждения, сообщения об ошибках,
- в случае `n = 1` пользователь получает только информацию об ошибках и предупреждения. Данный режим часто используется при пакетном запуске редактора связей или при вызове из интегрированной среды,

- в случае `n = 2` редактор связей не выдает ничего, то есть при работе редактора связей в данном режиме подавляются даже сообщения об ошибках.

Случай, когда `n` отсутствует эквивалентен `n = 1`, то есть запись параметра `-q` или `-q1` приводит к одинаковому результату.

Если после ключа `-q[n]` через знак "равно" без пробела задано имя файла, то редактор связей создает файл с данным именем и всю информацию направляет в него. Если файл с этим именем уже существует, новый файл будет записан поверх старого без дополнительного предупреждения.

3.6.3 Выдача титульного сообщения ("шапки") (ключ -t)

Параметр `-t` носит исключительно информационный характер. Она используется для получения сведений о названии данной компоненты БПО, о номере версии и об авторских правах. При появлении данного ключа в командной строке, редактор связей выдает информационной сообщение:

```
Редактор связей для NeuroMatrix® NM6403 *v1.0* (c)1996,98 * НТЦ Модуль,
```

и завершает работу. Работа редактора завершится независимо от того, есть ли в командной строке другие параметры. Выходной файл не создается.

3.6.4 Выдача сообщения о местоположении (ключ -p)

Параметр `-p` носит чисто информационный характер. Она используется для нахождения местоположения вызываемого редактора. Часто возникает ситуация, когда пользователь не знает, откуда именно вызывается редактор. Известно только, что это один из каталогов, приведенных в списке общедоступных каталогов в файле `autoexec.bat` (`PATH=...`). В этой ситуации можно использовать ключ `-p`. Редактор, встретив в командной строке данный ключ выдаст на экран сообщение:

```
Полное имя редактора связей: ...имя файла вместе с каталогом,
```

и завершает работу. Работа редактора завершится независимо от того, есть ли в командной строке другие параметры. Выходной файл не создается.

3.6.5 Задание выходного файла (ключ -o<имя_файла>)

Результатом работы редактора связей является выходной файл. Его имя определяется путем задания в командной строке ключа `-o<имя файла>`. Между ключом `-o` и именем файла не должно быть пробела. Под именем файла подразумевается полное имя, то есть полный путь к файлу. Если задано только имя, а путь опущен, файл будет создан в текущем каталоге. Если файл с таким именем в

данном каталоге уже существовал, он будет замещен новым файлом без каких либо дополнительных предупреждений..

Если редактор связей не встречает в командной строке ключа, определяющего название выходного файла, то в качестве имени выходного файла используется имя первого файла из списка входных файлов. При этом, расширение нового файла будет зависеть от того, каков его тип:

Табл. 3-4 Возможные расширения выходных файлов редактора связей.

Расширение	Описание
`.abs`	Для абсолютных исполняемых файлов.
`.rel`	Для исполняемых перемещаемых файлов.
`.elz`	Для объектных файлов.

В базовом ПО для процессора NM6403 для объектных файлов основным расширением является "`.elf`". Однако редактор связей выходному объектному файлу присваивает расширение "`.elz`". Это делается во избежание замещения старого (входного) объектного файла.

3.6.6 Командный файл (@<имя_файла>)

Командный файл является альтернативным способом задания параметров редактору связей. Когда в командной строке приходится перечислять большое количество ключей и имен файлов, часто возникает ситуация переполнения, так как длина командной строки ограничена 128-ю символами. Поэтому удобнее расположить все параметры в командном файле, а в качестве параметра командной стоки передать редактору связей его имя, предваренное символом @. Между знаком @ и именем командного файла не должно быть пробела. В этом случае вид командной строки заметно упростится. Она будет выглядеть примерно так:

```
linker @cmdfile.cmd
```

В командном файле все параметры редактора связей могут располагаться как на отдельных строках, так и по несколько на одной строке. Например, ни одна из приведенных ниже записей не будет ошибочной:

```
-r
-d2 -stack=128
test1.elf -oresfile.rel
test2.elf test3.elf
```

В командный файл удобно помещать константную часть командной строки, а остальные, часто изменяемые параметры задавать

непосредственно в командной строке. Например, если в качестве входного редактору необходимо передать следующий набор параметров:

```
-a -stack=4 -heap=1024 -heap1=1024 -d4 main.elf
test1a.elf test1b.elf rtl.lib,
```

и при этом проследить результат работы редактора для различных типов выходного файла и для разных алгоритмов удаления неиспользуемых секций. Работу с таким набором параметров можно организовать следующим образом:

- В командный файл `myfile.cmd` поместить все ключи, не изменяемые при тестировании. Тогда командный файл может выглядеть так:

  ```
  -stack=4 -heap=1024 -heap1=1024
  main.elf test1a.elf
  test1b.elf rtl.lib
  ```

- Изменяемые параметры записывать в командной строке наряду с командным файлом. Тогда процесс запуска редактора связей сведется к следующему:

  ```
  linker -a @myfile.cmd -d4
  ```

При этом не придется каждый раз переписывать длинную командную строку и беспокоиться о нехватке в ней места для параметров, которые могут дополнить общий список.

3.7 Типы выходного файла

Редактор связей работает с объектными файлами формата ELF, предназначенными для выполнения на процессоре NM6403. В заголовке таких файлов расположено поле, определяющее тип инструментальной машины, для которой они были созданы. В данном случае, это поле имеет специальное значение, соответствующее процессору. При любых других значениях данного поля редактор выдаст соответствующую ошибку и прекратит работу.

Редактор связей порождает три типа выходных файлов:

- **абсолютный исполняемый файл** предназначен для выполнения на процессоре NM6403. Назван абсолютным, потому что для каждой структуры, входящей в его состав известен абсолютный (точный) адрес расположения в памяти, доступной процессору. Его внутренние данные структурированы таким образом, чтобы максимально ускорить и упростить процесс загрузки в память вычислительного устройства. Файл разбит на специальные области, называемые сегментами, каждый из которых представляют собой дамп определенной области памяти

вычислительного устройства. Загрузка такого файла сводится к копированию сегментов по адресам, заданным в их заголовках.

- **исполняемый перемещаемый файл** предназначен для выполнения на процессоре NM6403. В отличии от предыдущего типа файла адреса загрузки его структур заранее неизвестны, и определяются лишь в момент загрузки. Поэтому файл данного типа содержит таблицу символов и таблицы перемещений для тех секций, в которых встречаются ссылки на символы. Все символы из символьной таблицы должны быть определены. Это означает, что каждый символ должен содержать информацию о секции, в которой под него выделено место. Размещение файла данного типа в памяти вычислительного устройства осуществляется специальным загрузчиком, который должен назначать адреса размещения секций, вычислять адреса символов и разрешать все ссылки, хранящиеся в таблицах перемещения секций.
- **объектный файл** не предназначен для выполнения на процессоре, является промежуточной ступенью в создании выполняемой программы. Редактор связей позволяет из нескольких объектных файлов сделать один, объединяющий в себе всю входную информацию. Такой файл предназначен для последующей сборки с другими объектными файлами. Редактор связей осуществляет набор внутренних преобразований объектных файлов, поэтому объектные файлы, полученные из под кросс-компилятора ассемблера будут изменены и упрощены, хотя и останутся объектными файлами. Эти изменения связаны с разрешением отдельных типов ссылок в пределах секций и с заменой ссылок на все локальные символы ссылками на начало секций, в которых они определены.

В процессе создания исполняемых файлов редактор связей по желанию пользователя может включать алгоритм удаления неиспользуемых секций (см. 3.8.1Удаление неиспользуемых секций и отладочной информации (ключ -d4) на стр. 3-16). В режиме создания объектного файла такой алгоритм не предусмотрен, о чем выдается соответствующее предупреждение.

3.7.1 Создание абсолютного исполняемого файла (ключ -abs или -a)

Параметр `-a` указывает редактору связей на то, что выходным файлом должен быть абсолютный исполняемый файл. Данный ключ установлен по умолчанию, поэтому если редактор не встретит входного параметра, определяющего тип выходного файла, он создаст абсолютный файл. Абсолютный файл в обязательном порядке должен содержать точку входа. По умолчанию точка входа называется `"start"`. Обычно она содержится в стартовом коде (startup code) и представляет собой глобальную метку. Пользователь

может ввести свое имя точки входа. (см. 3.8.6 Определение имени точки входа (ключ -start=<имя_метки>) на стр. 3-19)

Следующие примеры связывают воедино файлы `file1.elf` и `file2.elf`, и создают абсолютный исполняемый файл:

```
linker -abs file1.elf file2.elf -oresult.abs
linker -a file1.elf file2.elf -oresult.abs
linker file1.elf file2.elf -oresult.abs
```

3.7.2 Создание исполняемого перемещаемого файла (ключ -rel или -r)

Параметр `-r` указывает редактору связей на то, что выходным файлом должен быть исполняемый перемещаемый файл. Для размещения данного файла в памяти нейровычислителя должен использоваться специальный загрузчик, который получает от пользователя карту памяти вычислительного устройства и адрес каждой загружаемой секции. Загрузчику необходимо вычислить адреса всех символов и разрешить все ссылки, хранящиеся в файле данного типа, определить точку входа. Для этого загрузчик обрабатывает символьную таблицу, и таблицы перемещений соответствующих секций. В отличии от абсолютного файла, в котором загружаемые секции объединены в сегменты, и загружаются в память одновременно, процесс загрузки исполняемого перемещаемого файла требует отдельной обработки каждой секции. Однако, перемещаемые файлы позволяют организовывать процесс загрузки более гибко, в зависимости от контекстных условий работы программы.

Следующие примеры связывают воедино файлы `file1.elf` и `file2.elf`, и создают исполняемый перемещаемый файл:

```
linker -rel file1.elf file2.elf -oresult.rel
linker -r file1.elf file2.elf
```

3.7.3 Создание объектного файла (ключ -elf или -e)

Параметр `-e` указывает редактору связей на то, что выходным файлом должен быть объектный файл. Порождаемый редактором объектный файл обобщает и хранит информацию из входных файлов. Он производит следующие преобразования:

- *склеивает секции с одинаковыми именами.* Под склеиванием понимается добавление очередной входной секции в конец одноименной выходной, после чего такие секции воспринимаются, как единое целое. Речь идет не обо всех секциях, содержащихся в объектных файлах, а только о секциях, хранящих инициализированные и неинициализированные данные, коды программы и отладочную информацию. Помимо этих секций в объектных файлах хранится служебная информация о символах, о ссылках на символы и об именах

символов. Механизмы обработки такой информации отличаются от склеивания. Они будут описаны в следующих пунктах,

- *создает сводную таблицу символов.* При этом частично разрешаются ссылки на неопределенные внешние символы. В отличии от двух предыдущих типов файлов объектные файлы могут содержать неразрешенные внешние ссылки,
- *строит таблицы перемещений* для тех выходных секций, в которых содержатся ссылки на символы. Разрешает часть ссылок, а именно, относительные ссылки из секций на символы, которые также определены в этих секциях (такие ссылки часто возникают при использовании команды skip в программах на языке ассемблера),
- *заменяет ссылки на локальные символы* ссылками на начало секций, в которых эти локальные символы были определены. Это позволяет разгрузить символьную таблицу, исключить из нее большое количество неиспользуемых локальных символов, что сказывается и на размере выходного файла.

В режиме создания объектного файла редактор связей отключает режим удаления неиспользуемых секций.

Следует отметить, что механизм слияния секций инициализированных и неинициализированных данных имеет некоторые особенности. Если в разных объектных файлах, собираемых вместе, встречаются одноименные секции разных типов, то есть одна секция содержит инициализированные данные, а вторая неинициализированные, то при их слиянии образуется секция инициализированных данных. Та часть, которая раньше не была инициализирована, заполняется нулями.

Следующие примеры связывают воедино файлы `file1.elf` и `file2.elf`, и создают объектный файл:

```
linker -elf file1.elf file2.elf -oresult.elf
linker -e file1.elf file2.elf -oresult.elf
```

3.8 Специфичные параметры редактора связей

К специфичным параметрам редактора связей относится набор ключей, не встречающихся в других компонентах БПО или имеющих там другой смысл.

3.8.1 Удаление неиспользуемых секций и отладочной информации (ключ -d4)

Редактор связей предоставляет пользователю возможность при создании выходного файла не включать в него неиспользуемую информацию. К такой информации относятся отладочная

информация и секции, на которые нет ссылок из других секций (исключая специальные служебные секции, которые обрабатываются отдельно).

Данный режим используется редактором связей по умолчанию при создании исполняемого файла. Другая эквивалентная запись этого режима: `-d`, где цифра не указана.

Полное удаление ненужной информации возможно только при создании абсолютного или перемещаемого исполняемого файла. При попытке указать данный режим при создании объектного файла, редактор связей игнорирует ключ `-d4` и выдает соответствующее предупреждение.

Следующий пример связывает воедино файлы `file1.elf` и `file2.elf`, и создает абсолютный исполняемый файл, из которого удалены все неиспользуемые секции и символы, включая отладочную информацию:

```
linker -d4 file1.elf file2.elf -oresult.abs
linker -d file1.elf file2.elf -oresult.abs
```

3.8.2 Сохранение отладочной информации (ключи -d{1..3})

Режимы частичного сохранения отладочной информации служат минимизации объема выходного файла при необходимости сохранения отладочной информации. Во всех этих режимах частично удаляются (по крайней мере, должны удаляться) неиспользуемые секции и привязанные к ним секции с отладочной информацией.

Эти три режима отличаются алгоритмами и появились, в основном, из-за отсутствия полной ясности относительно взаимосвязей секций данных и секций с отладочной информацией. В последующих версиях вероятно останется только один режим из трех.

Как и полное удаление, частичное удаление ненужной информации возможно только при создании абсолютного или перемещаемого исполняемого файла. При попытке указать данный режим при создании объектного файла, редактор связей игнорирует ключ `-d1..3` и выдает предупреждение.

Следующий пример создает абсолютный исполняемый файл `alone.abs`, из которого удалены все неиспользуемые секции данных, а из секций с отладочной информацией сохранены только те, которые ссылаются на оставшиеся секции данных:

```
linker alone.elf -d3
```

3.8.3 Запрещение удаления неиспользуемых секций (ключ -d0)

Параметр `-d0` используется тогда, когда пользователь желает сохранить в выходном файле всю информацию, содержавшуюся во

входных файлах, в том числе и ту, которая, возможно, не задействована в программе.

Следующий пример связывает воедино файлы `file1.elf` и `file2.elf`, и создают абсолютный исполняемый файл, сохраняя всю входную информацию:

```
linker -d0 file1.elf file2.elf -oresult.abs
```

При создании объектных файлов данный ключ является ключом по умолчанию, а кроме того единственно возможным среди ключей `-d{0..4}` , поскольку на этом этапе отсутствует информация о том, какие данные и код будут использованы в дальнейшем, а какие нет.

При создании исполняемых файлов **ключ `-d0`** в некоторых случаях **может помешать** получению выходного файла. Это произойдет в ситуации, когда в одном или в нескольких входных модулях, подаваемых в качестве параметров редактору связей, хранятся неопределенные глобальные символы, на которые нет ни одной ссылки.

Например, рассмотрим файл на языке Си++, в котором имеется следующее объявление функции:

```
extern int MyUndefFunc();
```

Однако на нее нет ни одной ссылки. Такая конструкция может вызвать предупреждение компилятора Си++, однако попадет в ассемблерный файл, а через него и в объектный. Когда редактор связей будет объединять данный файл с другими для создания исполняемого файла, он обнаружит, что данная метка не определена, но ссылки на нее отсутствуют. Если использовать любой из ключей `-d{1..4}`, то рассматриваемый символ будет проигнорирован, и не попадет в выходной файл. Однако ключ `-d0` предусматривает, что все символы и секции входных объектных файлов перейдут в выходной. Главным свойством исполняемого файла является отсутствие неразрешенных внешних ссылок. Поэтому редактор связей выдаст ошибку "неопределенный глобальный символ", и аварийно завершит работу.

3.8.4 Динамическое выделение памяти (ключи -heap и -heap1)

Компилятор Си++ для процессора NM6403 использует секции неинициализированных данных `.heap` (локальная куча) и `.heap1` (глобальная куча) для создания динамической кучи времени выполнения, используемой функцией `malloc()`. При помощи ключей `-heap` и `-heap1` можно определить размер кучи в глобальной и локальной памяти вычислительного устройства. Величина кучи задается после знака "равно", и измеряется в 32-разрядных словах:

```
linker -heap=0x40000 -heap1=0x100000,
```

под локальную кучу выделено 256K слов, а под глобальную 1Mb слов.

Между ключами `-heap` и `-heap1`, знаком "=" и размером кучи не должно быть пробелов.

По умолчанию размеры локальной и глобальной куч равны 1K слов. Если размер кучи не установлен пользователем, редактор связей выдает напоминание, что под кучу выделено всего 1K слов.

Редактор связей создает секции `.heap` и `.heap1` только в том случае, если они имеются во входных файлах, в частности, в библиотеке работы с динамической памятью. Во всех других случаях данные ключи игнорируются.

Редактор связей создает также глобальные символы `__HEAP_SIZE` и `__HEAP1_SIZE`. Им присваиваются значения, равные размерам локальной и глобальной куч памяти времени выполнения.

3.8.5 Определение размера стека (ключ -stack=[размер])

Для работы программ на процессоре NM6403, вызова функций, сохранения регистров и адресов возврата используется стек. Стек представляет собой секцию неинициализированных данных с именем `.stack`, размер которой задается на этапе создания исполняемого файла, то есть во время редактирования связей. Размер стека задается при помощи параметра `-stack=[размер]` и измеряется в 32-разрядных словах.

Следующий пример демонстрирует процесс определения стека (секции `.stack`) размером 4K слов:

```
linker -stack=0x1000
```

По умолчанию размер стека равен 1K слов.

Между ключом `-stack`, знаком "=" и размером стека не должно быть пробелов.

Если в файле конфигурации не указано обратное, секция `.stack` выделяется в памяти среди прочих секций. Однако для увеличения быстродействия рекомендуется данную секцию помещать в наиболее быструю из доступных память процессора. Механизм размещения секций по заранее определенным адресам приводится в описании файла конфигурации (см. 3.11.3 Секция SECTIONS на стр. 3-32).

3.8.6 Определение имени точки входа (ключ -start=<имя_метки>)

Адрес памяти, с которого программа начинает выполняться, называется **точкой входа**. Когда загрузчик помещает программу в память вычислительного устройства, счетчик команд должен быть инициализирован, в него необходимо записать адрес начала программы.

По умолчанию, когда в командной строке не задано имя точки входа, редактор связей полагает, что она имеет имя **`start`**. Если глобальный символ с данным именем в таблице символов не обнаружен, или обнаружен, но не определен, то пользователю выдается ошибка " точка входа `start` не определена ".

Ключ `-start=<имя_метки>` позволяет задать свое имя метки, с которой будет запущена программа. Метка обязательно должна быть определена в одной из секций, и иметь глобальный тип связывания. Редактор связей вычисляет ее адрес и сохраняет его, как адрес точки входа.

Следующий пример демонстрирует процесс определения пользовательской точки входа, которая в программе помечена меткой `BEGIN`:

```
linker file1.elf file2.elf -start=BEGIN
```

Между ключом `-start`, знаком "=" и именем метки не должно быть пробела.

Обычно точка входа с именем `start` определена в стартовом коде, который хранится в библиотеке времени выполнения. При подсоединении данной библиотеки в процессе редактирования связей стартовый код автоматически добавляется к пользовательской программе.

Примечание

Если использовать стандартную библиотеку, и при этом определить собственную точку входа, то стандартный стартовый код не будет использован, а при дополнительном задании ключа `-d{1..4}` *можно и вовсе удалить его из исполняемого файла. Тогда пользователь берет на себя ответственность по правильной установке регистра стека* `ar7` *при запуске программы, и по обработке возврата из программы.*

3.8.7 Отключение инициализации статических глобальных объектов (ключ -asm)

Параметр `-asm` позволяет редактору связей, необходимых для инициализации статических глобальных объектов в Си++.

Следующий пример связывает воедино файлы `file1.elf` и `file2.elf`, и создают абсолютный исполняемый файл, отключая обработку и создание специализированных секций `.init` и `.fini`:

```
linker file1.elf file2.elf -asm -oresult.abs
```

Если пользователь не использует стандартный стартовый код для запуска программы, то он может удалить упомянутые секции из исполняемого файла. Поскольку секции `.init` и `.fini` обрабатываются специальным образом, они не могут быть удалены

путем использования ключа `-d{1..4}`, для этого и введен специальный ключ `-asm`.

Стартовый код содержит два вызова функций:

```
call ctor;
call dtor;
```

Они вызываются соответственно до и после функции `__main` - главной функции пользовательской программы. Метки `ctor` и `dtor` определены в начале секций `.init` и `.fini`. Если в программе встречались статические глобальные объекты, поля которых необходимо инициализировать, то в секции `.init` и `.fini` добавляются вызовы соответствующих функций инициализации и удаления этих объектов. Если же полей, которые необходимо инициализировать до функции `__main` нет, то функции `ctor` и `dtor` содержат только команды возврата.

Если пользователь использует свою точку входа, а его программа написана на ассемблере и не требует досрочной инициализации статических полей, он может использовать данный ключ, и удалить из исполняемого файла секции `.init` и `.fini`.

3.8.8 Определение пути к каталогу библиотечных файлов (ключ -l (строчная "L"))

Параметр `-l` задает путь к каталогу библиотечных файлов. Если редактор связей при сборке исполняемой программы обнаружил неопределенный глобальный символ, на который существует ссылка, то для определения этого символа он просматривает библиотеки, которые приведены в командной строке и находятся в каталоге, указанном пользователем при помощи ключа `-l`.

Если пользователь хочет подключить библиотеку, которая расположена не в текущем каталоге, то он может добавить ее полное имя в командную строку, или указать при помощи ключа `-l` каталог, в котором она расположена.

Следующий пример связывает воедино файлы `file1.elf` и `file2.elf`, и осуществляет поиск неопределенных глобальных символов в библиотеке `mylib.lib`, расположенной в каталоге `c:\lib`:

```
linker file1.elf file2.elf mylib.lib -lc:\lib -oresult.abs
```

Между ключом `-l` и путем к каталогу библиотек не должно быть пробела.

Редактор связей осуществляет поиск библиотек в следующем порядке:

- рассматривает библиотеки, имена которых заданы в командной строке;
- ищет библиотеки с заданными именами в текущем каталоге;

- просматривает каталоги, заданные ключом `-l` в той последовательности, в которой они перечислены в командной строке или в командном файле.

Если пользователь желает задать несколько каталогов для поиска библиотек, он каждый каталог должен вводить параметр `-l`:

```
linker file1.elf file2.elf mylib.lib -lc:\rtl -lc:\lib -oresult.abs
```

Примечание

Каждая библиотека состоит из определенного числа объектных файлов. Если искомый символ определен в данной библиотеке, это значит, что он определен в одном из объектных файлов, входящих в состав библиотеки. Поэтому только этот файл будет загружен редактором связей для дальнейшей обработки. Если в нем содержится много посторонней информации, не относящейся к определению рассматриваемого символа, она при наличии ключа -d{1..4}будет удалена из конечного исполняемого файла.

3.8.9 Создание карты памяти (ключ -m<имя_каталога>)

Параметр `-m` задает имя файла, в который редактор связей запишет информацию о карте распределения памяти для данного выходного файла. Карта памяти создается только для абсолютных исполняемых файлов. Она описывает:

- распределение памяти процессора NM6403 (адреса и размеры банков),
- положение сегментов в банках памяти,
- распределение выходных секций по сегментам,
- положение входных секций в выходных,
- абсолютные адреса всех глобальных символов.

В файле-карте содержится имя выходного файла и точка входа. Кроме того в него включены следующие таблицы:

- таблица, описывающая адреса и размеры банков памяти процессора, размеры страницы памяти каждого банка,
- таблица, описывающая распределение сегментов по банкам памяти. Каждый сегмент имеет несколько атрибутов: принадлежность банку памяти, абсолютный адрес начала и размер,
- таблица, описывающая распределение секций по сегментам. Каждая секция имеет несколько атрибутов таких, как принадлежность сегменту, абсолютный адрес начала и размер.

Помимо этого в таблице хранится информация о входных секциях, составляющих выходную, а именно: из какого входного файла данная секция и ее размер,

- таблица, описывающая имена и адреса всех глобальных символов программы.

Следующий пример связывает воедино файлы `file1.elf` и `file2.elf`, создает абсолютный исполняемый файл и строит для него карту памяти, сохраняя ее в файле `mapfile.map`:

```
linker file1.elf file2.elf -mmapfile.map -oresult.abs
```

Между ключом `-m` и именем файла-карты памяти не должно быть пробела.

Имя файла-карты памяти ставится в том случае, если оно отлично от имени выходного файла. По умолчанию имя карты памяти совпадает с именем выходного файла, но имеет расширение `".map"`. Поэтому тот же пример, но с умалчиваемым значением карты памяти будет выглядеть так:

```
linker file1.elf file2.elf -m -oresult.abs
```

,при этом полученный файл-карта памяти будет иметь имя `result.map`.

3.8.10 Задание файла конфигурации (ключ -c<имя_файла>)

Параметр `-c` определяет имя файла конфигурации для создания абсолютного исполняемого файла. Файл конфигурации содержит всю информацию, необходимую для правильного размещения программы в памяти процессора. Более подробно файл конфигурации описывается в разделе 3.11 Файл конфигурации на стр. 3-27.

Следующий пример связывает воедино файлы `file1.elf` и `file2.elf`, создает абсолютный исполняемый файл и использует для этого файл конфигурации `cfgfile.cfg`:

```
linker file1.elf file2.elf -ccfgfile.cfg -oresult.abs
```

Между ключом `-c` и именем файла конфигурации не должно быть пробела. Файл конфигурации используется только для создания абсолютного исполняемого файла.

Если при создании абсолютного файла ключ `-c` с именем файла конфигурации не установлен, то редактор связей создает выходной файл с единственным программным сегментом, адрес начала которого равен `0x00000000`, а размер неограничен. Неограниченность размера сегмента означает, что он вмещает в себя все загружаемые секции с учетом требований по их выравниванию.

Для изменения адреса сегмента необходимо использовать ключ `-addr`, описанный ниже.

3.8.11 Задание адреса сегмента по умолчанию (ключ -addr=<адрес>)

В режиме создания абсолютного исполняемого файла, когда не задан файл конфигурации, редактор связей создает единый сегмент данных и кода, в который входят все загружаемые секции, встреченные редактором во входных файлах. Для задания адреса размещения этого сегмента в памяти процессора используется ключ `-addr=<адрес>`. В поле "адрес" заносится абсолютный адрес начала сегмента, записанный в шестнадцатиричном виде, например:

```
linker file1.elf file2.elf -addr=0x00000080;
linker file1.elf file2.elf -addr=0x80000080;
```

3.9 Параметры по умолчанию

По умолчанию, при запуске редактора связей с набором входных объектных файлов устанавливаются следующие входные параметры:

Табл. 3-5 Параметры редактора связей по умолчанию.

Параметр	Состояние	Примечания
`-a`	Установлен	В результате работы редактора связей будет создан абсолютный исполняемый файл.
`-d4`	Установлен	Удаляется вся отладочная информация и неиспользуемые секции и символы.
`-heap=0x400`	Установлен / Не установлен	Начальный размер кучи в локальной памяти равен 1K. Параметр устанавливается по умолчанию в случае включения в список входных файлов библиотеки управления динамической памятью
`-heap1=0x400`	Установлен / Не установлен	Начальный размер кучи в глобальной памяти равен 1K. Параметр устанавливается по умолчанию в случае включения в список входных файлов библиотеки управления динамической памятью
`-stack=0x400`	Установлен	Начальный размер стека равен 1K
`start=start`	Установлен	По умолчанию точкой входа считается метка start с глобальным типом

		связывания.
-asm	Не установлен	Включена поддержка начальной инициализации глобальных переменных в Си++
-l""	Установлен	При нахождении неопределенных глобальных символов в рабочем каталоге осуществляется поиск библиотек, в которых данные символы могли бы быть определены.
-addr=0x00000000	Установлен	В отсутствие файла конфигурации создается один загружаемый сегмент с адресом загрузки 0x00000000

3.10 Допустимые и недопустимые сочетания параметров

Часть параметров редактора связей, называемых информационными (-h, -?, -t, -p), не могут быть использованы совместно с другими, не входящими в данную группу. Основной целью данных параметров служит предоставление пользователю информации о порядке запуска редактора, используемых входных параметрах, версии, местоположении. При этом, встретив информационный параметр, редактор связей прекращает работу. Например, встретив комбинацию входных параметров

```
-q=file.log test.elf -h
```

редактор связей не запустит процесс обработки входного объектного файла, а только выдаст в file.log справочную информацию со списком ключей управления.

Для каждого типа выходного файла существует свой набор входных параметров. Далее перечислены наборы входных параметров для каждого типа выходного файла, порождаемого редактором связей.

Табл. 3-6 Набор входных параметров абсолютного исполняемого файла.

Параметры	Примечание
-a	абсолютный исполняемый файл.
-c<имя_файла>	назначение файла конфигурации.
-heap=[размер]	размер кучи в локальной памяти (в килобайтах).
-heap1=[размер]	размер кучи в глобальной памяти (в килобайтах).
-stack=[размер]	размер стека (в килобайтах).
-start=<имя_метки>	точка входа в программу.
-asm	отключение поддержки инициализации

	глобальных статических объектов в языке Си++.
-l<имя_каталога>	Путь к каталогу библиотечных файлов.
-d0	Запрещение удаления неиспользуемых секций
-d1..3	Сохранение отладочной информации в режиме удаления неиспользуемых секций.
-d4	Удаление всей отладочной информации и неиспользуемых секций.
-m<имя_файла>	создание карты памяти абсолютного исполняемого файла.

Табл. 3-7 Набор входных параметров исполняемого перемещаемого файла.

Параметры	Примечание
-r	Исполняемый перемещаемый файл.
-heap=[размер]	Размер кучи в локальной памяти (в килобайтах).
-heap1=[размер]	Размер кучи в глобальной памяти (в килобайтах).
-stack=[размер]	Размер стека (в килобайтах).
-start=<имя_метки>	Точка входа в программу.
-asm	Отключение поддержки инициализации глобальных статических объектов в языке Си++.
-l<имя_каталога>	Путь к каталогу библиотечных файлов.
-d0	Запрещение удаления неиспользуемых секций
-d1..3	Сохранение отладочной информации в режиме удаления неиспользуемых секций.
-d4	Удаление всей отладочной информации и неиспользуемых секций.

Табл. 3-8 Набор входных параметров объектного файла.

Параметр	Примечание
-e	Объектный файл.

Попытки комбинировать входные параметры, входящие в разные наборы, может в лучшем случае привести к игнорированию части параметров с выдачей предупреждений, в худшем - процесс создания выходного файла будет остановлен с выдачей сообщения

об ошибке. Действительно, попытка одновременного задания типа выходного файла (`-e`) и точки входа (`-start=<имя_метки>`) должна привести к игнорированию второго параметра, а одновременное использование ключей `-e` и `-a` к ошибке.

3.11 Файл конфигурации

Файл конфигурации используется редактором связей **только при создании абсолютных исполняемых файлов**. Он содержит информацию:

- о диапазонах доступных физических адресов и аппаратных характеристиках банков памяти процессора NM6403 (физическая конфигурация памяти);
- о расположении сегментов прикладной программы в памяти и их размерах (логическая конфигурация памяти);
- о принадлежности секций кода, данных и тд. программы тем или иным сегментам;
- о выравнивании секций и сегментов по границе страниц памяти или по иным величинам.

Файл конфигурации представляет собой текстовый файл, написанный на некотором формальном Си-подобном языке описания.

Файл состоит из произвольного количества блоков описаний трех типов:

- описаний памяти (`MEMORY`),
- описаний сегментов (`SEGMENTS`),
- описаний секций (`SECTIONS`).

Пример файла конфигурации:

```
MEMORY /* Физическая конфигурация памяти */
{
 LOCAL0 at 0x00000000, len = 0x00100000, page = 0x10000;
 LOCAL1 none;

 GLOBAL0 at 0x80000000, len = 0x00100000;
 GLOBAL1 at 0x80100000, len = 0x00100000;
}

SEGMENTS /* Логическая конфигурация памяти (сегменты) */
{
  name1 in GLOBAL0, length = 0x1000;
  name2 in GLOBAL1, len = 1024;
  name3 at 0x30000000, l = 2048;
  name4 in LOCAL0;
```

```
}

SECTIONS /* Размещение секций по сегментам */
{
  .text   in name2, align page;
  .data   in name1;
  .text1  in name1, align page;
  .nobits in name3;
  .heap   at 0x10020000;
  .stack  at 0x00010000;
}
```

Если секции MEMORY и SECTIONS отсутствуют, то редактор связей предполагает, что используется модель памяти по умолчанию. В этом случае все секции объектного файла размещаются по адресам, начиная с 0x00000000, в порядке поступления.

Численные значения могут быть записаны в десятичном и шестнадцатеричном форматах так же, как в языках Си/Си++: десятичное значение состоит из десятичных цифр и начинается не с нуля, шестнадцатеричные значения имеют префикс 0x, за которым следует последовательность шестнадцатеричных цифр 0..9, A..F, a..f.

3.11.1 Секция MEMORY

В секции MEMORY описывается физическая конфигурация памяти, доступной процессору вычислительного устройства. Вся память разбивается по областям адресов, называемым банками.

Для каждого банка памяти выставляются следующие параметры:

LOCAL0, LOCAL1, GLOBAL0, GLOBAL1 - зарезервированные имена банков памяти;

len - размер банка памяти

page - размер страницы банка памяти.

3.11.1.1 Зарезервированные имена банков памяти.

Банки памяти могут иметь только одно из четырех имен: LOCAL0, LOCAL1, GLOBAL0, GLOBAL1.

В секции MEMORY должны содержаться характеристики каждого банка памяти. Если в данной конфигурации процессора один, или несколько банков отсутствуют, это должно быть отражено путем использования напротив имени соответствующего банка ключевого слова none.

Для каждого банка может быть задан свой размер страницы. Пользователь может управлять только размером страницы банка

памяти. Если значение этого поля не проставлено пользователем, ему присваивается умалчиваемое значение, соответствующее максимально возможному размеру страницы. Все остальные параметры секции определяются физической конфигурацией вычислительного устройства.

Пример секции MEMORY:

```
MEMORY /* Физическая конфигурация памяти */
{
 LOCAL0 at 0x00000000, len = 0x00100000, page = 0x10000;
 LOCAL1 none;/* Банк 1 локальной памяти отсутствует. */

 GLOBAL0 at 0x80000000, len = 0x00100000;
 GLOBAL1 at 0x80100000, len = 0x00100000;
}
```

3.11.1.2 Модель памяти по умолчанию

Когда файл конфигурации не задан, или в нем отсутствует секция MEMORY, редактор связей использует модель памяти по умолчанию. Эта модель основана на архитектуре процессора NM6403, и предполагает, что пользователю доступно все 32-х битное адресное пространство. Начальный адрес размещения исполняемого файла в памяти процессора полагается равным 0x00000000. Для загрузки абсолютных исполняемых файлов создается один безразмерный сегмент. Секции укладываются в сегмент в порядке поступления, однако существует две очереди размещения. В первую очередь попадают секции инициализированных данных, во вторую неинициализированных. Такой подход позволяет размещать в сегменте сначала инициализированные секции, а затем неинициализированные.

3.11.2 Секция SEGMENTS

В секции SEGMENTS задается расположение сегментов прикладной программы по банкам памяти.

Каждый сегмент имеет следующие атрибуты:

`Имя` - имя сегмента, не более 32 символов. Для задания имени могут использоваться символы: A-Z, a-z, ., _. Не допускаются пробелы внутри имен. Имя сегмента не входит ни в одну таблицу имен, задается только для удобства чтения файла конфигурации и для выдачи информации об ошибках.

`Length` или `len` или `l` - размер сегмента. Может быть опущен. Тогда сегмент имеет возможность раздуваться до пределов банка, в котором он определен. Размер сегмента ограничивает его максимальный объем. Реально сегмент может иметь и меньший размер, поскольку этот размер определяется размерами входящих в него

секций.

Сегмент не может переходить через границу банка памяти, то есть он всегда принадлежит только одному банку. Во избежание путаницы нельзя использовать одинаковые имена для разных сегментов.

Принадлежность сегмента тому или иному банку определяется использованием ключевого слова `in`. Если в банке расположено несколько сегментов, то они выстраиваются друг за другом, в порядке, определенном в файле конфигурации. При этом адрес каждого сегмента вычисляется только после размещения всех сегментов, однако он лежит в пределах границ адресов банка, которому принадлежит.

Сегменту можно назначать точный адрес в памяти при помощи ключевого слова `at`. Он должен принадлежать диапазону доступных физических адресов процессора. Таким образом, адрес данного сегмента задается заранее, а распределение остальных сегментов осуществляется с учетом этого факта.

Пример секции `SEGMENTS`:

```
SEGMENTS /* Логическая конфигурация памяти (сегменты) */
{
  name1 in GLOBAL0, length = 0x1000;
  name2 in GLOBAL1, len = 1024; // сегмент с ограниченным сверху размером
  name3 at 0x30000000, l = 2048;// сегмент с определенным адресом
  name4 in LOCAL0;              // сегмент с плавающим размером
}
```

Сегменту можно не задавать размер, тогда редактор связей сам вычислит его. Размер сегмента определяется, как сумма размеров входящих в него секций с учетом их выравнивания. Реально сегмент может содержать как секции, определенные в файле конфигурации, так и такие, чье расположение в памяти там не регламентируется, хотя они являются загружаемыми. Плавающий размер сегмента подразумевает, что в него могут добавляться секции, не описанные в файле конфигурации, без каких- либо ограничений в пределах данного банка памяти.

Выравнивание сегмента равно наибольшему выравниванию входящих в него секций. То есть, если сегмент состоит из трех секций, при этом одна из них выровнена по границе страницы памяти, а остальные по границе 64-х разрядного слова, то весь сегмент будет выровнен по границе страницы памяти.

3.11.2.1 Распределение сегментов в пределах банка памяти

Существует несколько вариантов распределения сегментов в пределах банка памяти:

- распределение сегментов заданных размеров. В этом случае редактор связей располагает сегменты в порядке их следования, приведенном в файле конфигурации. Сегменты ложатся один за другим с учетом констант выравнивания, заданных для каждого сегмента. Если места в банке окажется недостаточно, редактор связей выдаст соответствующую ошибку. Реальные адреса сегментов будут вычислены после того, как процесс размещения будет окончен,
- распределение сегментов, когда один из них имеет заранее определенный адрес и размер. Если этот сегмент не первый в списке, то банк разбивается на две части. В первую часть добавляются сегменты в порядке, определенном в файле конфигурации. Сегмент с определенным адресом пропускается. Если для очередного сегмента не хватает места, он помещается после сегмента с определенным адресом. Образовавшееся свободное пространство остается незаполненным,
- распределение сегментов, когда один из них имеет плавающий размер. Независимо ни от чего, сначала в банке размещаются все сегменты, размеры которых определяются только секциями, описанными в файле конфигурации. Далее в сегменты добавляются загружаемые секции, не упомянутые в файле конфигурации. Если первым в списке стоит сегмент с плавающим размером, то все эти секции добавляются в него. При этом осуществляется проверка на превышение размеров банка памяти. Если первым стоит сегмент заданного размера, то в него добавляются секции, пока его реальный не превысит заданного порога. Тогда остальные секции будут добавляться в следующие сегменты по той же схеме,
- распределение сегментов, когда два из них имеют плавающий размер. Ситуация почти не отличается от предыдущего пункта. Все дополнительные секции будут добавлены в первый сегмент плавающего размера, поэтому наличие плавающего размера у второго сегмента роли не играет, он реально ничем не будет отличаться от сегментов с ограниченным максимальным размером,
- распределение сегментов, когда сегмент с плавающим размером располагается перед сегментом с определенным адресом. В этом случае первый сегмент может увеличиваться до определенной границы, то есть его размер ограничивается адресным пространством от начала банка, до начала сегмента с определенным адресом. Дальнейший механизм добавления секций остается без изменений.

3.11.3 Секция SECTIONS

В SECTIONS содержится описание загружаемых секций, входящих в состав объектного файла. Каждая секция может иметь следующие атрибуты:

Имя - имя секции. Для задания имени могут использоваться символы: A-Z, a-z, _. Не допускаются пробелы внутри имен. Как и в языке ассемблера, имя секции не должно предваряться точкой, то есть имена секций в ассемблере и в файле конфигурации совпадают.

Align page - выравнивание по началу страницы памяти. Это означает, что секция должна начинаться с адреса, кратного размеру страницы памяти данного банка, с который она попадает (по умолчанию все загружаемые секции выравниваются по границе 64-х разрядного слова).

Секция всегда входит в состав того или иного сегмента (речь идет только об абсолютных исполняемых файлах). Если несколько секций принадлежат одному и тому же сегменту, то они размещаются в порядке, определенном в файле конфигурации. Секции, не упомянутые в файле конфигурации, но являющиеся загружаемыми также добавляются в сегменты по правилам, описанным в подпункте 3.11.2.1.

Задание абсолютного адреса секции вручную: at 0x80000000; адрес, начиная с которого размещается описываемая секция. В этом случае для данной секции заводится собственный сегмент с определенным адресом и размером, равным размеру секции.

Приведем пример секции SECTIONS:

```
SECTIONS /* Размещение секций по сегментам */
{
  text   in name2;
  data   in name1;
  text1  in name1, align page;
  nobits in name2, align page;
  heap   at 0x10020000;
  stack  at 0x00010000;
}
```

Последовательность секций в сегменте определяется несколькими факторами:

- общий порядок размещения секций в сегменте, когда сначала располагаются секции инициализированных, а затем неинициализированных данных,

- сначала в сегменте расположены секции, приведенные в файле конфигурации, а затем остальные с флагом загрузки.

Следующий пример показывает, как редактор связей расположит секции в сегменте.

- объектный файл содержит следующие загружаемые секции:

```
textProc;        (секция инициализированных данных)
bss.dataVector;  (секция неинициализированных данных)
dataVector;      (секция инициализированных данных)
textMain;        (секция инициализированных данных)
dataProc;        (секция инициализированных данных)
bss.dataProc;    (секция неинициализированных данных)
textProc1;       (секция инициализированных данных)
```

- файл конфигурации задает расположение некоторых из них в сегменте VECTORS:

```
SECTIONS
{
  dataVector     in VECTORS;
  bss.dataVector in VECTORS;
  textMain       in VECTORS, align page;
  textProc1      in VECTORS;
}
```

- реальное расположение секций в сегменте, установленное редактором связей:

```
//   секции инициализированных данных.
dataVector;  // из файла конфигурации.
textProc;    // не отмеченная в файле конфигурации.
textMain;    // из файла конфигурации.
textProc1;   // из файла конфигурации.
dataProc;    // не отмеченная в файле конфигурации.
// секции неинициализированных данных.
bss.dataVector; // из файла конфигурации.
bss.dataProc1;  // не отмеченная в файле конфигурации.
```

Однако реальный порядок секций в сегменте может быть отличным от приведенного выше. А повлиять на него способно выравнивание секций по границе страницы памяти. Например, если в результате выравнивания секций, между секциями `dataVector` и `textMain` образуется неиспользуемый участок памяти, то редактор связей может поместить в него одну из секций, не упомянутых в файле конфигурации. Это зависит от размера вставляемой секции и от требования по ее выравниванию в памяти. Если секция `textProc` с учетом выравнивания имеет размер меньший размера неиспользуемого участка, то окончательное расположение секций в сегменте будет следующим:

```
//   секции инициализированных данных.
dataVector;  // из файла конфигурации.
textProc;    // не отмеченная в файле конфигурации.
textMain;    // из файла конфигурации.
textProc1;   // из файла конфигурации.
dataProc;    // не отмеченная в файле конфигурации.
```

3.11.3.1 Особенности в названиях секций данных

При записи в файл конфигурации имен секций данных следует иметь в виду следующее обстоятельство: для каждой секции инициализированных данных (определенной в ассемблере вводным словом `data`) кросс-ассемблер создает парную секцию неинициализированных данных, в которую переносятся все неинициализированные данные, объявленные в секции инициализированных данных. (более подробную информацию см. в документе: **Программное обеспечение NeuroMatrix® NM6403. Описание языка ассемблера**).

Если секция неинициализированных данных возникла, как пара соответствующей инициализированной секции, то ее имя формируется путем добавления в начало имени парной секции приставки "`bss.`". То есть, если имя секции было "`dataVector`", то имя парной секции неинициализированных данных: "`bss.dataVector`". Этот факт необходимо учитывать при формировании файла конфигурации.

Одна из такой пары секций может оказаться пустой, и даже если они обе упоминаются в файле конфигурации, пользователь, управляя редактором связей решает, что делать с пустой секцией, оставлять ее в выходном файле или удалять, используя ключ `-d[n]` (см. пп. 3.8.1, 3.8.2, 3.8.3).

3.12 Пример работы с редактором связей

Данный пример показывает процесс сборки объектных файлов `demo.elf`, `matrix.elf` и нескольких используемых ими библиотек, в единую программу - абсолютный исполняемый файл. Точка входа `start` определена в файле `startup.elf`.

Предположим, что конфигурация физической памяти процессора NM6403 такова:

Глобальная память:

- Банк 0 с адреса `0x80000000` по адрес `0x8003FFFF`
- Банк 1 с адреса `0x80040000` по адрес `0x8007FFFF`

Локальная память

- Банк 0 с адреса `0x00000000` по адрес `0x0003FFFF`
- Банк 0 с адреса `0x00040000` по адрес `0x0007FFFF`

Выходные секции образуются из следующих входных секций:

- пары секций MatrixArray и bss.MatrixArray из matrix.elf должны располагаться в локальной памяти, а секции Matrix1Array и bss.Matrix1Array в глобальной,
- исполняемый код, содержащийся в секциях .text файлов demo.elf и matrix.elf, собирается в одну выходную секцию, которую необходимо расположить (в силу внутренних причин задачи) в локальной памяти по адресу 0x00000200,
- из секции .text вызывается функция UDIV32, осуществляющая беззнаковое деление 32-х разрядных чисел. Тело функции хранится в библиотечном модуле div.elf, который в составе библиотеки времени выполнения libc.lib расположен в каталоге d:\neuro\lib,
- из секции .text вызывается функция MulVects, осуществляющая скалярное умножение матриц. Тело функции хранится в библиотеке векторно-матричных вычислений matvect.lib, расположенной в текущем каталоге,
- программа использует работу с динамически выделенными массивами. Для этого подключаются функции библиотеки времени выполнения libc.lib, расположенные в библиотечных модулях malloc.elf, calloc.elf и free.elf.

Файл конфигурации для редактора связей demo.cfg выглядит так:

```
MEMORY /* физическая память нейропроцессора */
{
    LOCAL0  at 0x00000000, len = 0x40000, page = 0x4000;
    LOCAL1  at 0x00040000, len = 0x40000;
    GLOBAL0 at 0x80000000, len = 0x40000;
    GLOBAL1 at 0x80040000, len = 0x40000;
}
SEGMENTS
{
    GlobalSeg in GLOBAL0;
    LocalSeg  in LOCAL0;
}
SECTIONS
{
    text at 0x00000200;
    MatrixArray      in LocalSeg;
    bss.MatrixArray  in LocalSeg, align page;
    Matrix1Array     in GlobalSeg;
    bss.Matrix1Array in GlobalSeg, align page;
    stack in LocalSeg, align page;
    heap  in LocalSeg, align page;
    heap1 in GlobalSeg, align page;
}
```

Командный файл demo.cmd хранит все ключи запуска редактора связей, имена входных объектных файлов, файла конфигурации:

```
-a -oOUTFILE.ABS -cdemo.cmd -ld:\system\libs -mdemo.map
-stack=4096
-heap=65536
-heap1=65536
demo.elf
matrix.elf
```

Командная строка запуска редактора связей выглядит так:

```
linker.exe @file.cmd
```

Информацию о расположении программы в памяти процессора можно получить, просматривая содержимое файла карты памяти `demo.map`:

```
*************************************************************************
Редактор связей для NeuroMatrix® NM6403 * v1.0 * (c)1996,97* НТЦ Модуль
*************************************************************************
ИМЯ АБСОЛЮТНОГО ИСПОЛНЯЕМОГО ФАЙЛА: demo.abs
ТОЧКА ВХОДА: "start" по адресу 0x00000274
=========================================================================
ДОСТУПНЫЕ ФИЗИЧЕСКИЕ ДИАПАЗОНЫ АДРЕСОВ:
LOCAL0  at 0x00000000, len = 0x40000, page = 0x400;
LOCAL1  at 0x00040000, len = 0x40000;
GLOBAL0 at 0x80000000, len = 0x40000;
GLOBAL1 at 0x80040000, len = 0x40000;
=========================================================================
РАСПОЛОЖЕНИЕ СЕГМЕНТОВ И СОСТАВЛЯЮЩИХ ИХ СЕКЦИЙ:
СЕГМЕНТ: text, АДРЕС 0x00000200, РАЗМЕР 0x424
-------------------------------------------------------------------------
выходная секция  адрес       размер       флаги/входные секции
---------------  ----------- -----------  ----------------------------
.text             0x00000200   0x00000424
                  0x00000200   0x00000058   demo.elf (.text)
                  0x00000258   0x0000001B   matrix.elf(.text)
                  0x00000274   0x00000010   startup.elf(.text), libc.lib
                  0x00000284   0x0000003F   malloc.elf(.text), libc.lib
                  0x000002C4   0x00000053   calloc.elf(.text), lib.lib
                  0x00000318   0x0000002D   free.elf(.text), libc.lib
                  0x00000346   0x00000063   div.elf(.text), libc.lib
                  0x000003AA   0x0000007A   matrix.elf(.text),matvect.lib

СЕГМЕНТ: LocalSeg, АДРЕС 0x00000400, РАЗМЕР 0x12400
-------------------------------------------------------------------------
выходная секция  адрес       размер       флаги/входные секции
---------------  ----------- -----------  ----------------------------
MatrixArray      0x00000400   0x00000800
                  0x00000400   0x00000800   matrix.elf(MatrixArray)
.init            0x00000CB0   0x00000008   НЕИНИЦИАЛИЗИРОВАНА
                  0x00000CB0   0x00000008   startup.elf(init), libc.lib
.fini            0x00000CB8   0x00000008   НЕИНИЦИАЛИЗИРОВАНА
                  0x00000CB8   0x00000008   startup.elf(init), libc.lib
bss.MatrixArray  0x00001000   0x000000A0   НЕИНИЦИАЛИЗИРОВАНА
                  0x00001000   0x000000A0   matrix.elf(bss.MatrixArray)
stack            0x00001400   0x00001000   НЕИНИЦИАЛИЗИРОВАНА
```

```
                    0x00001400   0x00001000   startup.elf(stack), libc.lib
heap                0x00002400   0x00010000   НЕИНИЦИАЛИЗИРОВАНА
                    0x00002400   0x00010000   memory.elf(heap), libc.lib

СЕГМЕНТ: GlobalSeg, АДРЕС 0x80000000, РАЗМЕР 0x10C00
-------------------------------------------------------------------------
выходная секция  адрес       размер       флаги/входные секции
---------------  ----------- -----------  --------------------------
Matrix1Array     0x80000000   0x00000800
                    0x80000000   0x00000800   matrix.elf(Matrix1Array)
bss.Matrix1Array 0x80000800   0x800000A0   НЕИНИЦИАЛИЗИРОВАНА
                    0x00000800   0x000000A0   matrix.elf(bss.Matrix1Array)
heap1            0x00000C00   0x00010000   НЕИНИЦИАЛИЗИРОВАНА
                    0x00000C00   0x00010000   memory.elf(heap), libc.lib
=========================================================================
ГЛОБАЛЬНЫЕ СИМВОЛЫ:
имя символа                     адрес символа
-----------                     -------------
start                           0x00000274
__main                          0x00000200
Matrix                          0x00000400
Matrix1                         0x80000000
UDIV32                          0x00000346
MulVects                        0x00000258
__MatrixConv                    0x000003AA
__calloc                        0x000002C4
__malloc                        0x00000284
__free                          0x00000318

ВСЕГО: 10 символов
------------------
```

3.13 Сообщения об ошибках редактора связей

Редактор связей в процессе работы может столкнуться с некорректными данными. В этом случае он выдает сообщение не экран. Для выдачи сообщений на экран редактор связей использует форматированную строку. Ее формат является единым для всего комплекса БПО и имеет следующий вид:

“[имя_файла]”: тип_ошибки номер_ошибки: сообщение_об_ошибке.

где:

имя_файла - имя файла, при разборе внутренней структуры которого произошла ошибка. Поле имя_файла может отсутствовать, когда ошибка произошла вне блока разбора входных файлов.

тип_ошибки - один из типов ошибок, приведенных ниже в этом разделе.

код_ошибки - символьное обозначение ошибки. Представляет собой буквенно-цифровой код, три первых символа которого являются сокращением названия компоненты, породившей

сообщение об ошибке, а остальные ее номер. Каждая компонента БПО имеет автономную нумерацию ошибок. Пример кода ошибки: `LNK412`.

сообщение_об_ошибке - краткая справочная информация о причине возникновения ошибочной ситуации в ходе работы данной компоненты БПО.

Пример:

```
"file.elf" ОШИБКА LNK412: "Несколько символьных таблиц в одном файле!"
```

Все некорректные ситуации редактор связей делит на четыре группы, которым соответствуют четыре типа ошибок :

- **Предупреждения.** Предупреждение появляется, когда возникшая некорректная ситуация может повлиять на конечный результат работы библиотекаря, однако библиотекарю хватает информации для создания правильного выходного файла. Например, когда одна из входных директив не может быть использована в данном режиме работы библиотекаря, редактор связей выдает соответствующие предупреждения. При этом работа по разбору остальных входных параметров продолжается.
- **Ошибки.** Ошибка возникает, когда из-за некорректных входных данных библиотекарь не в состоянии создать правильный выходной файл. Например, ... В такой ситуации на экран выдается сообщение об ошибке, а библиотекарь прекращает работу и возвращает ненулевое значение.
- **Внутренние ошибки.** Внутренняя ошибка может возникнуть из-за неправильной работы самого библиотекаря. При возникновении такой ситуации все исходные данные желательно передать разработчикам для устранения дефектов. В такой ситуации на экран выдается сообщение о внутренней ошибке, а библиотекарь прекращает работу и возвращает ненулевое значение.
- **Фатальные ошибки.** Фатальная ошибка возникает в случае нехватки памяти для работы библиотекаря. Она указывает на то, что в сложившейся ситуации работа библиотекаря с данными входными параметрами не может быть продолжена. Необходимо изменить параметры системы.

Под номер ошибки отведено три цифры, то есть он лежит в интервале от 0 до 999. Между типами ошибок номера были распределены следующим образом:

0 - 399	Предупреждения.
400 - 799	Ошибки.

800 - 949	Внутренние ошибки.
950 - 999	Фатальные ошибки.

3.13.1 Предупреждения

LNK001	"Секция "..." не должна иметь таблицу перемещений. Таблица игнорируется."
	- секция, которая не должна иметь таблицу перемещений, имеет ее. В такой ситуации таблица перемещений игнорируется. Ошибка произошла в процессе формирования данного объектного файла. Если редактору связей в дальнейшем не удастся создать выходной файл, возникнет соответствующая ошибка. На данном же этапе просто выдается предупреждение, не приводящее к остановке работы редактора.
LNK003	"По умолчанию размер кучи "..." равен 1K слов (32-битных)"
	- предупреждение о том, что пользователь не задал размер соответствующей кучи динамической памяти, хотя использовал в своей программе функции работы с динамической памяти. Поскольку библиотека времени выполнения не отслеживает выход за пределы кучи, пользователь должен самостоятельно заботиться об этом. Поэтому предупреждение напоминает, что пользователь не установил новый размер кучи, а ее умалчиваемый размер 1K слов.
LNK004	"Неверный параметр "...". Игнорируется."
	- предупреждение о том, что пользователь задал неправильный параметр в командной строке или в командном файле. Этот параметр никак не влияет на ход выполнения редактором своих работ.
LNK005	"Повторное задание ключа "...". Игнорируется."
	- предупреждение о том, что пользователь повторно задал параметр в командной строке или в командном файле. Этот параметр игнорируется.
LNK006	"Во избежание удаления входного файла выходной имеет расширение ".elz"."
	- предупреждение о том, что пользователь может запортить содержимое входного объектного файла, так как его имя совпадает с именем выходного. Такое сообщение выдается в режиме сборки объектного файла (ключ `-elf` или `-e`) при незаданном имени выходного файла. Поскольку, по умолчанию все объектные файлы имеют расширение "`.elf`", существует опасность потерять входной файл. Для устранения данной проблемы редактор связей создаст выходной файл с расширением "`.elz`".

LNK007	"В командном файле встречен вызов другого командного файла. Игнорируется."
	- предупреждение о том, что пользователь пытается осуществить рекурсивный вызов командного файла. То есть, в командном файле не может содержаться вызов другого командного файла. Редактор связей игнорирует данную директиву.
LNK008	"Не могу открыть командный файл "...". Он игнорируется."
	- предупреждение о том, что редактор связей не может открыть командный файл, указанный в качестве параметра командной строки. Данная команда игнорируется.
LNK010	"При создании объектного файла ключ "-d" игнорируется."
	- предупреждение о том, что в режиме создания объектного файла ключ -d, обозначающий удаление неиспользуемых секций, игнорируется.

3.13.2 Ошибки

LNK401	"Ссылка на неопределенный локальный символ "..." из секции "..."."
	- данная ошибка может появиться вследствие ошибки во входном объектном файле, название которого приведено в начале форматированной строки сообщения.
LNK402	"Секция "..." имеет разные флаги в разных файлах."
	- это означает, что в одном из входных объектных файлов заголовок данной секции содержит флаг о ее загрузке в память вычислительного устройства, а в другом файле секция не содержит этого флага, то есть не является загружаемой. Ошибка может быть порождена неправильной работой компилятора ассемблера или сбоем в объектном файле.
LNK403	"Недостаточно места в сегменте "..." для размещения секций."
	- данная ошибка возникает, когда в файле конфигурации записано, что сегмент "..." имеет наперед заданный размер и содержит некоторый набор секций, а в ходе вычисления размеров секций оказывается, что для них в сегменте не хватает места. Ошибка может быть исправлена путем увеличения размера сегмента или перемещением одной или нескольких секций в другой сегмент.
LNK404	"Неизвестный тип ссылки в секции "..."."
	- данная ошибка возникает, когда в соответствующем поле объектного файла хранится значение, отличное от ожидаемого. Всего существует три типа ссылки на символ: абсолютная, относительная и байтовая.

Если в поле, определяющем тип ссылки, записано другое значение, это приводит к ошибке. Ошибка может быть порождена неправильной работой кросс-ассемблера или сбоем в объектном файле.

LNK405	"Не задана точка входа "..."."

- данная ошибка возникает, когда редактор связей не может определить адрес глобальной метки, определяющей точку входа. По умолчанию точка входа именуется как "start". Поэтому редактор связей в качестве точки входа определяет адрес символа "start". При изменении имени точки входа редактор использует адрес нового глобального символа. Если символ не определен, возникает данная ошибка. (см. параграф 3.8.6 Определение имени точки входа (ключ -start=<имя_метки>) на стр. 3-19)

LNK406	"Структура данного файла не соответствует формату ELF."

- данная ошибка возникает при попытке открытия редактором связей файла, который не является объектным, или его формат не соответствует формату, принятому в рамках базового ПО. Необходимо проверить, тот ли файл подается на вход редактору.

LNK407	"Данный тип процессора не поддерживается."

- данная ошибка возникает при попытке подать на вход редактору связей объектный файл формата ELF, предназначенный для типа процессора, не поддерживаемого данным базовым ПО (для любого кроме процессора NM6403). При возникновении данной ошибки необходимо перекомпилировать исходный текст программы, из которого был получен данный объектный файл.

LNK408	"Данный файл создан при помощи устаревшей версии библиотеки доступа к ELF."

- при возникновении данной ошибки необходимо перекомпилировать исходный текст программы, из которого был получен данный объектный файл. Возможно, новая версия редактора порождает объектные файлы, которые оказываются несовместимыми по внутренней структуре с файлами, полученными при помощи предыдущей версии редактора связей.

LNK409	"Несколько символьных таблиц в одном файле не поддерживаются."

- данная ошибка может возникнуть, когда объектный файл, полученный редактором связей в качестве входного параметра, содержит несколько символьных таблиц. Данная конфигурация не запрещена форматом ELF, однако базовое ПО процессора NM6403 рассчитано на то, что в объектных файлах, порождаемых в его рамках, хранится только одна таблица символов. При возникновении данной ошибки необходимо перекомпилировать исходный текст программы, из которого был получен данный объектный файл.

LNK410	"Неизвестный тип секции ("...")."

- данная ошибка может возникнуть, когда в структуре входного объектного файла была встречена секция, тип которой редактор связей определить не смог. Всего в объектных файлах формата ELF встречается пять типов секций: секции инициализированных и неинициализированных данных, таблица символов, таблица строк и таблица ссылок. Если в поле заголовка секции записано значение, которое не соответствует ни одному типу секции, то возникнет данная ошибка. При возникновении данной ошибки необходимо перекомпилировать исходный текст программы, из которого был получен данный объектный файл.

LNK411	"Не найдена таблица символов."

- данная ошибка возникает, когда редактор связей не может найти во входном объектном файле таблицу символов. Символьная таблица должна существовать даже в случае, когда исходная программа не содержит ни одного символа. В этом случае она состоит из одного нулевого элемента. При возникновении данной ошибки необходимо перекомпилировать исходный текст программы, из которого был получен данный объектный файл.

LNK412	"Не найдена таблица имен символов."

- данная ошибка возникает, когда редактор связей не может найти во входном объектном файле таблицу имен символов. Таблица имен символов должна существовать даже в случае, когда исходная программа не содержит ни одного символа. В этом случае она состоит из одного нулевого элемента. При возникновении данной ошибки необходимо перекомпилировать исходный текст программы, из которого был получен данный объектный файл.

LNK413	"Неверное поле в заголовке таблицы символов."

- данная ошибка возникает, когда поле, хранящее размер элемента таблицы символов равно нулю. Ошибка может быть порождена неправильной работой компилятора ассемблера или сбоем в объектном файле. При ее возникновении необходимо перекомпилировать исходный текст программы, из которого был получен данный объектный файл.

LNK414	"Неверное поле в заголовке таблицы перемещений "..."."

- данная ошибка возникает, когда поле, хранящее размер элемента таблицы ссылок равно нулю. Сообщение содержит информацию о том, в какой именно таблице ссылок произошел сбой. Ошибка может быть порождена неправильной работой компилятора ассемблера или сбоем в объектном файле. При ее возникновении необходимо перекомпилировать исходный текст программы, из которого был получен данный объектный файл.

LNK415	"Символ "..." имеет разные типы в разных файлах."

- данная ошибка возникает, когда в одном из входных объектных файлов символ "..." объявлен как объект данных, а в другом файле на него ссылаются, как на метку (и наоборот, объявлен как метка, а ссылаются на него, как на объект данных). Для устранения ошибки программисту необходимо упорядочить работу с данным символом, привести в соответствие его тип в разных исходных файлах на языке ассемблера или на Си++.

LNK416	"Переопределение глобального символа "..."."

- данная ошибка возникает, когда глобальный символ определен в нескольких объектных файлах. Обычно глобальные символы определяются только в одном файле, а в других файлах объявляют его, как внешний. Задача редактора связей отождествить все ссылки на данный глобальный символ с ним самим и правильно вычислить его адрес в памяти вычислительного устройства. Если же глобальный символ объявлен в нескольких местах, то редактор не в состоянии определить его адрес. Для исправления ошибки необходимо оставить объявление символа только в одном файле (с использованием слова `global`), а в остальных файлах объявить его как внешний (с использованием ключевых слов `weak` или `extern`).

LNK417	"Глобальный символ "..." не определен."

- данная ошибка возникает, когда во всех входных объектных файлах данный глобальный символ объявлен, как внешний (с использованием ключевых слов `weak` или `extern`), и нет места, где бы он был объявлен, как глобальный и под него было бы выделено место. Ключевое слово `global` приводит к выделению места для глобального символа, тогда как `extern` только объявляет символ, как внешний, без выделения для него места. Для исправления ошибки необходимо в одном из файлов исходных текстов объявить данный символ, как глобальный.

LNK418	"Не задано ни одного входного файла."

- данная ошибка возникает, когда в командной строке или в командном файле редактора связей нет ни одного входного объектного файла.

LNK419	"Абсолютный файл не может быть входным параметром."

- данная ошибка возникает, когда в командной строке или в командном файле редактора связей в качестве входного объектного файла передается абсолютный исполняемый файл.

LNK420	"Данный тип объектного файла формата ELF не поддерживается."

- данная ошибка возникает, когда на вход редактора связей подается объектный файл, созданный вне базового ПО процессора NM6403. При ее возникновении необходимо перекомпилировать исходный текст

программы, из которого был получен данный объектный файл.

LNK421	"common-символ "..." имеет тип "метка"."

- данная ошибка возникает, когда в программе на языке ассемблера объявляется common-символ типа "метка" (пример: `common AAA: label;`). Common-символ может иметь только тип "данные". Если возникла такая ошибка, необходимо внести исправление в файл на языке ассемблера.

LNK422	"Символ "..." хранит неверный индекс секции."

- ошибка во входном объектном файле. Каждый символ таблицы символов помимо информации о файле, в котором он определен, содержит информацию о входной секции, в которой под него было выделено место. Эта информация хранится в виде индекса в таблице секций входного файла. Если данный индекс указывает на секцию, не являющуюся секцией данных, то возникает данная ошибка (функция, выдающая ошибку - `CalcSymAddr`). Возможно, ошибка произошла вследствие возникшего сбоя при создании входного объектного файла. Необходимо повторно создать объектный файл при помощи ассемблера. Если это не приведет к устранению ошибки, то необходимо с данной проблемой обратиться к разработчикам компилятора ассемблера.

LNK423	"Ссылка из секции "..." на символ, определенный в служебной секции."

- ошибка во входном объектном файле. В таблице перемещений указанной секции данных встретилась ссылка на символ, определенный в служебной секции, например в таблице имен символов.

LNK424	"Количество неопределенных символов: ... "

- данная информационная строка подводит итог по поиску определений неопределенных глобальных символов в библиотечных файлах. Возникает только в случае, когда неопределенным остался хотя бы один глобальный символ.

3.13.3 Внутренние ошибки

LNK801	"Указатель на выходную секцию "..." не существует."

- данная ошибка возникает, когда редактор связей пытается добавить данные, хранящиеся во входной секции в выходную с тем же именем (функция `AddData`), и при этом указатель на выходную секцию, для данной входной равен `NULL`. Если восстановление из архивной копии не даст результата, то объектные файлы, при редактировании связей в которых встретилась данная ошибка, необходимо передать разработчикам редактора связей.

LNK802	"Секция ".common" не найдена."

- данная ошибка возникает, когда редактор связей пытается вычислить адрес common-символа, а соответствующая секция, где он может быть определен, еще не создана (функция `CalcSymAddr`). Если восстановление из архивной копии не даст результата, то объектные файлы, при редактировании связей, в которых встретилась данная ошибка, необходимо передать разработчикам редактора связей.

LNK803	"Неизвестный тип выходного файла."

- данная ошибка возникает при попытке редактора связей создать выходной файл, тип которого отличен от абсолютного исполняемого, исполняемого перемещаемого и объектного. Ошибка возникает в функции `StoreOutFile`. Если восстановление из архивной копии не даст результата, то объектные файлы, при редактировании связей в которых встретилась данная ошибка, необходимо передать разработчикам редактора связей.

LNK804	"Переопределение специального символа "..." "

- данная ошибка возникает, когда объявление одного из символов: `__HEAP_SIZE`, `__HEAP1_SIZE`, `ctor`, `dtor` встречается дважды. Данные символы не объявляются в прикладной программе, а определяются редактором связей (функция `CreateAbsSymbols`). Данная ошибка возникнет также, когда произошла ошибка в определении точки входа (функция `FitEntryPointSymbol`). Если восстановление из архивной копии не даст результата, то объектные файлы, при редактировании связей, в которых встретилась данная ошибка, необходимо передать разработчикам редактора связей.

LNK805	"Не могу добавить данные в секцию "...". (...)"

- данная ошибка возникает при попытке добавить в выходную секцию данные, из одноименной входной секции. Ошибка происходит на низком уровне (в модуле `libelf.cpp`) при работе функции `AddData`. В скобках выдается оригинальное диагностическое сообщение библиотеки `libelf`. Если восстановление из архивной копии не даст результата, то объектные файлы, при редактировании связей, в которых встретилась данная ошибка, необходимо передать разработчикам редактора связей.

LNK806	"Не могу склеить данные секции "...". (...)"

- данная ошибка возникает при попытке слияния одноименных входных секций данных в одну выходную. Ошибка происходит на низком уровне (в модуле `libelf.cpp`) при работе функции `MergeSects`. В скобках выдается оригинальное диагностическое сообщение библиотеки `libelf`. Если восстановление из архивной копии не даст результата, то объектные файлы, при редактировании связей в которых встретилась данная ошибка, необходимо передать

разработчикам редактора связей.

LNK807	"Не могу создать выходные секции. (...)"

- данная ошибка возникает при попытке создания выходной секции. Ошибка происходит на низком уровне (в модуле `libelf.cpp`) при работе функции `CreateElfScn`. В скобках выдается оригинальное диагностическое сообщение библиотеки `libelf`. Если восстановление из архивной копии не даст результата, то объектные файлы, при редактировании связей, в которых встретилась данная ошибка, необходимо передать разработчикам редактора связей.

LNK808	"Не могу разрешить ссылки. (...)"

- данная ошибка возникает при попытке разрешения ссылок, хранящихся в таблицах ссылок секций инициализированных данных. Ошибка происходит на низком уровне (в модуле `libelf.cpp`) при работе функции `CreateAbsAddr`. В скобках выдается оригинальное диагностическое сообщение библиотеки `libelf`. Если восстановление из архивной копии не даст результата, то объектные файлы, при редактировании связей, в которых встретилась данная ошибка, необходимо передать разработчикам редактора связей.

LNK809	"Не могу пересчитать выходные таблицы перемещений. (...)"

- данная ошибка возникает в процессе создания выходных таблиц перемещений для секций инициализированных данных (в режиме `-elf`). Ошибка происходит на низком уровне (в модуле `libelf.cpp`) при работе функции `FillOutRelTabs`. В скобках выдается оригинальное диагностическое сообщение библиотеки `libelf`. Если восстановление из архивной копии не даст результата, то объектные файлы, при редактировании связей, в которых встретилась данная ошибка, необходимо передать разработчикам редактора связей.

LNK810	"Не могу сохранить выходные таблицы перемещений. (...)"

- данная ошибка возникает в процессе сохранения в выходном файле таблиц перемещений для секций инициализированных данных (в режиме `-elf`). Ошибка происходит на низком уровне (в модуле `libelf.cpp`) при работе функции `StoreRelTabs`. В скобках выдается оригинальное диагностическое сообщение библиотеки `libelf`. Если восстановление из архивной копии не даст результата, то объектные файлы, при редактировании связей, в которых встретилась данная ошибка, необходимо передать разработчикам редактора связей.

LNK811	"Не могу сохранить выходной файл. (...)"

- данная ошибка возникает в процессе сохранения выходного файла. Ошибка происходит на низком уровне (в модуле `libelf.cpp`) при вызове функции `StoreOutFile`. В скобках выдается оригинальное сообщение библиотеки `libelf`. Если восстановление из архивной

копии не даст результата, то объектные файлы, при редактировании связей, в которых встретилась данная ошибка, необходимо передать разработчикам редактора связей..

LNK812	"Ссылка на "неиспользуемый" символ "..." из секции "..."."

- данная ошибка возникает, когда редактор связей в режиме удаления неиспользуемых секций и символов помечает данный символ, как неиспользуемый, тогда как на него существует ссылка. Ошибка возникает во время работы функции `CreateAbsAddr`. Объектные файлы, при редактировании связей, в которых встретилась данная ошибка, необходимо передать разработчикам редактора связей.

LNK813	"Входная секция имеет неверную ссылку на выходную секцию."

- ошибка во взаимодействии внутренних структур редактора связей. Каждая входная секция данных хранит ссылку на одноименную выходную. Этой ссылкой является индекс в таблице выходных секций. Если этот индекс лежит вне диапазона индексов таблицы выходных секций, то возникает данная ошибка (функция `CalcSymAddr`). Если восстановление из архивной копии не даст результата, то объектные файлы, при редактировании связей, в которых встретилась данная ошибка, необходимо передать разработчикам редактора связей.

LNK814	"Файловый индекс символа "..." вне заданного диапазона."

- ошибка во взаимодействии внутренних структур редактора связей. Каждый символ таблицы символов содержит информацию о файле, в котором он был определен. Эта информация хранится в виде индекса в таблице входных файлов редактора связей. Ошибка может возникнуть во время работы функции `CalcSymAddr`. Возможно, ошибка произошла вследствие возникшего сбоя в редакторе связей. Если восстановление из архивной копии не даст результата, то объектные файлы, при редактировании связей, в которых встретилась данная ошибка, необходимо передать разработчикам редактора связей.

LNK815	"Секционный индекс символа "..." вне заданного диапазона."

- ошибка во взаимодействии внутренних структур редактора связей. Каждый символ таблицы символов помимо информации о файле, в котором он определен, содержит информацию о входной секции, в которой под него было выделено место. Эта информация хранится в виде индекса в таблице секций входного файла. Ошибка может возникнуть во время работы функции `CalcSymAddr`. Возможно, ошибка произошла вследствие возникшего сбоя в редакторе связей. Если восстановление из архивной копии не даст результата, то объектные файлы, при редактировании связей, в которых встретилась данная ошибка, необходимо передать разработчикам редактора связей.

LNK816	"Ошибка чтения таблицы символов (...)."
	- данная ошибка может возникнуть в процессе чтения символьной таблицы. Причиной может послужить ее неправильная структура или ошибка в процессе чтения из файла. Ошибка происходит на низком уровне (в модуле `libelf.cpp`) при работе функции `ReadSymTab`. В скобках выдается оригинальное диагностическое сообщение библиотеки `libelf`. Если восстановление из архивной копии не даст результата, то объектные файлы, при редактировании связей, в которых встретилась данная ошибка, необходимо передать разработчикам редактора связей.
LNK817	"Ошибка чтения таблицы перемещений "...". ("...")"
	- данная ошибка может возникнуть в процессе чтения таблиц перемещений. Причиной может послужить их неправильная структура или ошибка в процессе чтения из файла. Ошибка происходит на низком уровне (в модуле `libelf.cpp`) при работе функции `CreateRefTab`. В скобках выдается оригинальное диагностическое сообщение библиотеки `libelf`. Если восстановление из архивной копии не даст результата, то объектные файлы, при редактировании связей, в которых встретилась данная ошибка, необходимо передать разработчикам редактора связей.
LNK818	"Ошибка при загрузке файла (...)."
	- данная ошибка может возникнуть в процессе загрузки и разбора структур входного объектного файла. Причиной может послужить его неправильная структура или ошибка в процессе чтения с диска. Ошибка происходит на низком уровне (в модуле `libelf.cpp`) при работе функции `Load`. В скобках выдается оригинальное диагностическое сообщение библиотеки `libelf`. Если восстановление из архивной копии не даст результата, то объектные файлы, при редактировании связей, в которых встретилась данная ошибка, необходимо передать разработчикам редактора связей.
LNK819	"В данный момент очередь символов не должна быть пустой."
	- данная ошибка может возникнуть в процессе поиска определений неопределенных символов в библиотеках. Ошибка происходит на низком уровне (в модуле `lib.cpp`) при работе функции `LinkLibsWithoutClearance` или `LinkLibsWithClearance`. Если восстановление редактора связей из архивной копии не даст результата, то объектные файлы, при редактировании связей, в которых встретилась данная ошибка, необходимо передать разработчикам редактора связей.
LNK820	"В очереди неопределенных символов неожиданно появился определенный символ."

- данная ошибка может возникнуть в процессе поиска определений неопределенных символов в библиотеках. Ошибка происходит на низком уровне (в модуле `lib.cpp`) при работе функции `LinkLibsWithoutClearance` или `LinkLibsWithClearance`. Если восстановление редактора связей из архивной копии не даст результата, то объектные файлы, при редактировании связей, в которых встретилась данная ошибка, необходимо передать разработчикам редактора связей.

LNK821	"Символ "..." неожиданно потерян."

- данная ошибка может возникнуть в процессе поиска определений неопределенных символов в библиотеках. Ошибка происходит на низком уровне (в модуле `lib.cpp`) при работе функции `LinkLibsWithClearance`. Если восстановление редактора связей из архивной копии не даст результата, то объектные файлы, при редактировании связей, в которых встретилась данная ошибка, необходимо передать разработчикам редактора связей.

LNK822	"Символ "..." имеет тип данных, не встречающийся в библиотеках."

- данная ошибка может возникнуть в процессе поиска определений неопределенных символов в библиотеках. Ошибка происходит на низком уровне (в модуле `lib.cpp`) при работе функции `LinkLibsWithClearance`. Если восстановление редактора связей из архивной копии не даст результата, то объектные файлы, при редактировании связей, в которых встретилась данная ошибка, необходимо передать разработчикам редактора связей.

LNK823	"Секция "..." содержит неверный индекс выходной секции."

- данная ошибка может возникнуть в процессе установления связей между входными и выходными секциями. Ошибка происходит на низком уровне (в модуле `concat.cpp`) при работе функции `FillOutRelTabs`. Если восстановление редактора связей из архивной копии не даст результата, то объектные файлы, при редактировании связей, в которых встретилась данная ошибка, необходимо передать разработчикам редактора связей.

3.14 Фатальные ошибки

LNK951	"Ошибка при запросе ... байтов памяти."

- данная ошибка возникает, когда недостаточно свободного пространства для работы редактора связей. Попытка выделения в области динамической памяти очередного массива для размещения внутренних структур оканчивается неудачей. В результате работа не может быть продолжена. Для исправления данной ситуации необходимо каким-либо образом освободить динамическую память.

Например, выгрузить некоторые резидентные программы, или увеличить свободное место на диске, на котором создается системный временный файл.

LNK952	"Не могу создать выходной файл: "..."."

- данная ошибка возникает из-за каких-либо отказов операционной системы при создании файла. Например, к данной ошибке может привести нехватка места на диске, или отсутствие каталога, в который пользователь хочет записать файл, или запрет записи в определенные каталоги, налагаемый операционной системой. Прежде, чем продолжить работу с редактором связей, необходимо разобраться с причиной ошибки, или выбрать другое место на диске, куда файл должен быть записан.

LNK953	"Не могу открыть файл "..."."

- данная ошибка возникает из-за проблем, порожденных операционной системой. Например, при попытке открыть файл, используемый в это же время другими приложениями. Прежде, чем продолжить работу с редактором связей, необходимо понять, что явилось ее причиной.

LNK954	"Файл "..." не найден."

- данная ошибка возникает из-за отсутствия входного файла с данным именем.

4 Библиотекарь объектных файлов

4.1 Введение

В данной главе описан интерфейс, управляющие параметры и режимы работы программы - библиотекаря объектных файлов, входящей в состав базового ПО процессора NeuroNatrix® NM6403. Дано полное описание возможностей библиотекаря и способов работы с ним, включая информацию о всех ключах и использовании командных файлов. Здесь также приведен список ошибок, обнаруживаемых или диагностируемых библиотекарем. Для более лучшего понимания приведены примеры работы с библиотекарем объектных файлов.

4.2 Функциональное назначение библиотекаря

Библиотекарь объектных файлов (далее просто "библиотекарь") является вспомогательным средством в комплексе БПО процессора NeuroNatrix® NM6403. Данная программа предназначена для работы с библиотеками объектных файлов формата ELF, получаемых с помощью ассемблера и редактора связей БПО NeuroNatrix® NM6403.

Библиотекарь не предназначен для работы с файлами формата ELF, созданными другими программными средствами. Программа позволяет:

- создавать библиотеки из наборов объектных файлов,
- просматривать содержимое библиотек,
- добавлять, удалять и замещать файлы в библиотеках,
- извлекать объектные файлы из библиотек.

4.3 Характеристики библиотекаря

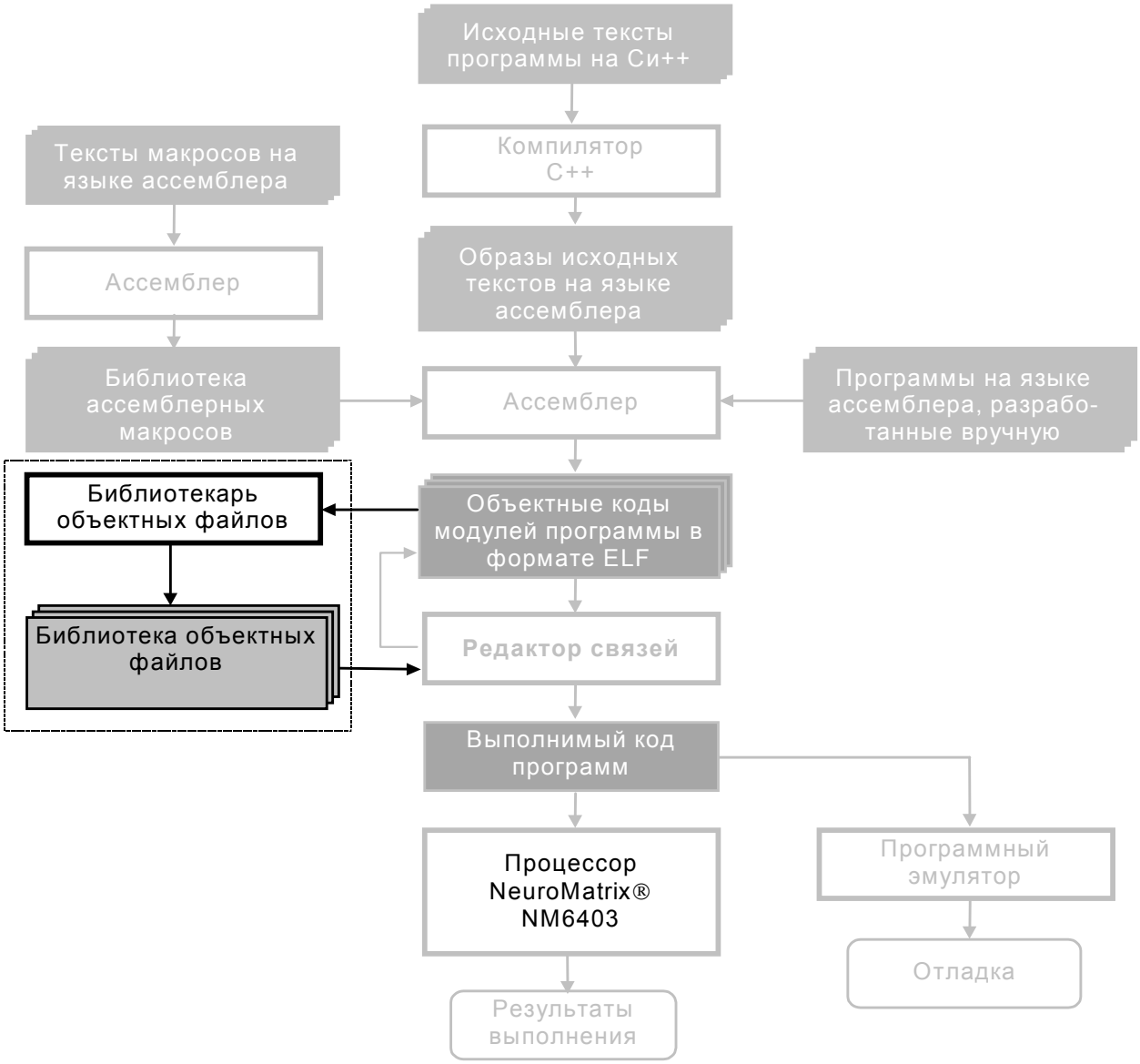
Библиотекарь объектных файлов представляет собой консольное приложение Windows95/NT.

Библиотекарь объектных файлов ориентирован на работу только с файлами формата ELF. Он полностью поддерживает данный формат за исключением создания и обработки динамических библиотек.

4.4 Положение библиотекаря в структуре БПО

На Рис. 4-1 показано положение библиотекаря в структуре БПО, а также его место в процессе разработки прикладных программ для процессора NeuroMatrix® NM6403.

Рис. 4-1 Положение библиотекаря объектных файлов в структуре БПО.



4.5 Формат запуска библиотекаря

Формат запуска библиотекаря:

```
libr [параметр] [имя_архива  [список_файлов]]
```

где:

`libr` - Имя исполняемого файла библиотекаря.

`Параметр` - Параметр, управляющий режимом работы библиотекаря.

`имя_архива` - Имя библиотеки, с которой работает библиотекарь.

`список_файлов` - Список объектных файлов, добавляемых или извлекаемых из библиотеки.

4.6 Параметры управления программой

Для управления работой библиотекаря используется набор параметров (ключей).

Все параметры предваряются префиксом "-".

Они могут быть указаны в командной строке или в командном файле (см. раздел 4.7 Командный файл на стр. 4-6).

В Табл. 4-1 приводится перечень параметров библиотекаря.

Табл. 4-1 Список параметров управления библиотекарем.

Наименование параметра	Назначение
`-c [create]`	Создание библиотеки объектных файлов.
`-a [add]`	Добавление файлов в библиотеку объектных файлов.
`-r [replace]`	Замещение файлов в библиотеке объектных файлов.
`-d [delete]`	Удаление файлов из библиотеки объектных файлов.
`-e [extract]`	Извлечение файлов из библиотеки объектных файлов.
`-l [list]`	Просмотр содержимого библиотеки объектных файлов.
`-h [help]/-?`	Краткая справка об использовании программы.

4.6.1 Создание библиотеки (ключ -c [create])

При данном режиме должны быть указаны имя создаваемой библиотеки и имена включаемых объектных файлов. При указании имен объектных файлов допускается использование `wildcards`: '*' и '?'. Если в качестве имени библиотеки используется имя существующего файла, то этот файл будет предварительно удален.

4.6.2 Добавление файлов в библиотеку (ключ -a [add])

Режим аналогичен предыдущему, за исключением того, что в случае существования библиотеки с указанным именем, эта библиотека не будет предварительно уничтожена, указанные объектные файлы будут добавлены в библиотеку. Если при попытке добавить файл в библиотеку обнаруживается, что файл с таким именем в библиотеке уже существует, то добавление не будет произведено.

4.6.3 Замещение файлов в библиотеке (ключ -r [replace])

Режим аналогичен режиму `add`, за исключением того, что указанные в командной строке объектные файлы будут добавлены в библиотеку вне зависимости от того, есть ли одноименные файлы уже в библиотеке или нет.

4.6.4 Удаление файлов из библиотеки (ключ -d [delete])

В режиме удаления в командной строке обязательно должны быть указаны имя библиотеки и удаляемые файлы. Если файла с указанным именем в библиотеке нет, библиотекарь выдает предупреждение и продолжает работу.

4.6.5 Извлечение файлов из библиотеки (ключ -e [extract])

Данный режим позволяет извлечь из библиотеки модули и поместить их в отдельные объектные файлы. Библиотека при этом не изменяется. При извлечении в командной строке должно быть указано имя библиотеки. Если не указаны имена объектных файлов, то из библиотеки будут извлечены все модули. При извлечении создаваемые файлы получают текущее время!

4.6.6 Просмотр содержимого библиотеки (ключ -l [list])

Выводится список элементов библиотеки с указанием имени, времени и размера. Сам ключ в командной строке можно не указывать, например:

```
libr library.lib
```

4.6.7 Краткая справка об использовании программы (ключ -h [help]/-?)

Краткая справка выдается также при запуске программы без параметров или с ключами `-h`, `-?`. Ее вид:

```
Библиотекарь объектных файлов * v1.02 * (c) 1997-99 * НТЦ Модуль
Формат запуска:
libr [команда] [имя_библиотеки] [список_файлов]
   команда         - команда, задающая режим работы библиотекаря
   имя_библиотеки - имя библиотеки
   список_файлов   - перечень объектных файлов
Список команд:
   -h или -?   - краткая справка (этот текст)
   -l          - просмотреть список файлов библиотеки
   -c          - создать библиотеку
   -a          - добавить файлы в библиотеку
   -d          - удалить файлы из библиотеки
   -e          - извлечь файлы из библиотеки (библиотека не изменяется)
   -r          - добавить файлы в библиотеку с замещением существующих
Примечания:
   1. Если команда не указана, выполняется '-l'.
   2. '-c' перед созданием библиотеки уничтожает файл с именем имя_библиотеки.
   3. Если при извлечении список файлов не указан, извлекаются все файлы.
   4. Извлекаемые файлы перекрывают существующие.
   5. При удалении и замещении создается временная копия библиотеки.
```

4.7 Командный файл

Часть параметров библиотекаря может быть помещена в командный файл. В этом случае командная строка имеет вид:

```
libr @имя_командного_файла.
```

Командных файлов может быть несколько:

```
libr @командный_файл1 @командный_файл2.
```

Кроме того, в командный файл может быть занесена только часть командной строки, например:

```
libr  @имя_командного_файла.
```

Не допускаются вложенные командные файлы.

Командный файл представляет собой текстовый файл, содержащий допустимые параметры, разделенные произвольным количеством разделителей. В качестве разделителей можно использовать пробелы, символы табуляции и перевода строки.

Пример командного файла:

```
-c mylib.lib
file1.elf file2.elf
file3.elf file4.elf
```

При вызове библиотекаря с данным командным файлом будет создана библиотека `mylib.lib`, в состав которой войдут перечисленные выше объектные файлы.

4.8 Использование wildcards (масок)

Использование масок в качестве имен объектных файлов допускается в командах создания (`-c`), добавления (`-a`) и замещения (`-r`). В команде извлечения использование масок не допускается, но имена объектных файлов могут быть опущены, в этом случае извлекаются все файлы. В остальных случаях использование масок не допустимо.

4.9 Варианты запуска библиотекаря

В Табл. 4-2 приведены различные варианты запуска библиотекаря объектных файлов.

Табл. 4-2 Варианты вызова библиотекаря.

Командная строка	Описание
libr	Краткая справка об использовании программы.
libr -h	Краткая справка об использовании программы.
libr -?	Краткая справка об использовании программы.
libr libname [filelist]	Просмотр списка файлов библиотеки.
libr -l libname [filelist]	Просмотр списка файлов библиотеки.

libr -c libname [filelist]	Создание библиотеки.
libr -a libname [filelist]	Добавление файлов в библиотеку.
libr -d libname [filelist]	Удаление файлов из библиотеки.
libr -e libname [filelist]	Извлечение файлов из библиотеки.
libr -r libname [filelist]	Замена файлов в библиотеке.

4.10 Примеры работы с библиотекарем

4.10.1 Создание библиотеки

```
libr -c first.lib *.elf
```

Эта команда создаст библиотеку с именем `first.lib` и включит в нее все файлы из текущего каталога, имеющие расширение `elf`. Если файл с именем `first.lib` уже существует в текущем каталоге, он будет предварительно удален.

Приведенную команду можно также записать в виде:

```
libr -c first *.elf,
```

поскольку расширение `.lib` используется библиотекарем для библиотек по умолчанию. Если необходимо создать библиотеку, не имеющую расширения, необходимо ставить точку после имени библиотеки:

```
libr -c first. *.elf
```

Эта команда создаст библиотеку с именем `first.`

4.10.2 Добавление объектных файлов в библиотеку

```
libr -a first.lib *.elf
```

Если в текущем каталоге есть файл `first.lib`, то он будет проверен на предмет того, является ли он библиотекой установленного формата, и затем в него будут добавлены все объектные файлы формата `Elf` из текущего каталога, имеющие соответствующее расширение.

Если в текущем каталоге нет файла `first.lib`, то действие данной команды ничем не отличается от рассмотренной в п. 4.10.1.

4.10.3 Извлечение объектных файлов из библиотеки

```
libr -e library.lib single.elf
```

В результате действия этой команды из библиотеки `library.lib` будет извлечен файл с именем `single.elf`.

Библиотека при этой операции не изменяется.

Извлечение из библиотеки всех файлов

```
libr -e library.elf
```

В результате действия этой команды из библиотеки `library.lib` будут извлечены все файлы.

Библиотека при этой операции не изменяется.

Несмотря на то, что файлы могут помещаться в библиотеку с запоминанием относительного или абсолютного пути, при извлечении путь отсекается от имени файла, и все файлы помещаются в текущий каталог. Если в библиотеке содержались два файла с одинаковым именем (с разными путями), то при одновременном извлечении этих двух файлов, второй затрет первый. В данной ситуации файлы необходимо извлекать по одному, указывая полное имя файла (включая путь).

4.10.4 Замена файла в библиотеке

```
libr -r lib a1.elf
```

В результате действия этой команды, файл `a1.elf`, передаваемый в качестве входного параметра, заменит файл с тем же названием в библиотеке `lib`. Если в библиотеке нет указанного файла, то будет выдано соответствующее предупреждение, и файл будет добавлен в библиотеку.

4.10.5 Удаление файла из библиотеки

```
libr -d lib a1.elf
```

В результате действия этой команды, файл `a1.elf`, передаваемый в качестве входного параметра, будет удален из библиотеки `lib`. Если в библиотеке нет указанного файла, то будет выдано соответствующее предупреждение, и выходной файл с именем `a1.elf` не будет создан. Для того, чтобы удалить из библиотеки несколько файлов, необходимо их все перечислить в командной строке.

4.11 Сообщения об ошибках

Библиотекарь в процессе работы может столкнуться с некорректными данными. В этом случае он выдает сообщение не экран. Для выдачи сообщений на экран библиотекарь использует форматированную строку. Ее формат является единым для всего комплекса БПО и имеет следующий вид:

“[имя_файла]”: тип_ошибки номер_ошибки: сообщение_об_ошибке.

где:

имя_файла - имя файла, при разборе внутренней структуры которого произошла ошибка. Поле имя_файла может отсутствовать, когда ошибка произошла вне блока разбора входных файлов.

тип_ошибки - один из типов ошибок, приведенных ниже в этом разделе.

код_ошибки - символьное обозначение ошибки. Представляет собой буквенно-цифровой код, три первых символа которого являются сокращением названия компоненты, породившей сообщение об ошибке, а остальные ее номер. Каждая компонента БПО имеет автономную нумерацию ошибок. Пример кода ошибки: LBR412.

сообщение_об_ошибке - краткая справочная информация о причине возникновения ошибочной ситуации в ходе работы данной компоненты БПО.

Например:

```
ОШИБКА LBR407: Библиотека "AAA.ELF" не найдена.
```

Все некорректные ситуации библиотекарь делит на четыре группы, которым соответствуют четыре типа ошибок :

- **предупреждения.** Предупреждение появляется, когда возникшая некорректная ситуация может повлиять на конечный результат работы библиотекаря, однако библиотекарю хватает информации для создания правильного выходного файла. Например, предупреждение появляется, когда одна из входных директив не может быть использована в данном режиме работы библиотекаря. Работа при этом не прекращается.
- **ошибки.** Ошибка возникает, когда из-за некорректных входных данных библиотекарь не в состоянии создать правильный выходной файл. Например, ... В такой ситуации на экран выдается сообщение об ошибке, а библиотекарь прекращает работу и возвращает ненулевое значение,
- **внутренние ошибки.** Внутренняя ошибка может возникнуть из-за неправильной работы самого библиотекаря. При возникновении такой ситуации все исходные данные желательно передать разработчикам для устранения дефектов. В такой ситуации на экран выдается сообщение о внутренней ошибке, а библиотекарь прекращает работу и возвращает ненулевое значение,
- **фатальные ошибки.** Фатальная ошибка возникает в случае нехватки памяти для работы библиотекаря. Она указывает на то, что в сложившейся ситуации работа библиотекаря с данными входными параметрами не может быть продолжена. Необходимо изменить параметры системы.

4.11.1 Предупреждения

LBR001 "Ошибка при получении времени создания файла ..., устанавливается текущее время."

- не удается выяснить время создания файла. При записи в библиотеку ставится текущее время.

4.11.2 Ошибки

LBR401	`"Отсутствует управляющий входной параметр."`
	- отсутствует команда управления библиотекарем (первый параметр).
LBR401	`"Имя библиотеки не определено."`
	- отсутствует имя библиотеки.
LBR401	`"Не могу прочитать входной параметр номер: ..."`
	- отсутствует параметр. Нумерация параметров ведется с команды управления библиотекарем, которая определена, как первый параметр библиотекаря.
LBR402	`"Неизвестная команда ..."`
	- неверная команда.
LBR403	`"Файл ... не является библиотекой."`
	- неверный формат библиотеки.
LBR404	`"Файл ... не является библиотекой (...)."`
	- неверный формат библиотеки; В отличии от LBR403 в данном случае выводится оригинальное сообщение библиотеки LIBELF.
LBR405	`"Файл ... уже существует в библиотеке."`
	- файл с данным именем уже находится в библиотеке.
LBR406	`"Символ ... (файл ...") уже определен в библиотеке."`
	- невозможно добавить файл в библиотеку, т.к. файл содержит глобальный символ с именем, уже присутствующей в таблице символов библиотеки. Указывается имя символа и имя файла, которые не удалось добавить в библиотеку.
LBR407	`"Библиотека ... не найдена."`
	- библиотека с заданным именем не найдена в указаном каталоге.

4.11.3 Внутренние ошибки

LBR801	`"Неизвестный рабочий режим: ..."`
	- неверный режим работы (наведенная ошибка в результате сбоя программы или системы).

LBR802	"Не могу придумать имя временного файла."

- неудачная попытка создать имя временного файла или наведенная ошибка.

LBR803	"Нулевой элемент в списке файлов."

- неверный элемент в списке файлов.

4.11.4 Фатальные ошибки

LBR950	"Ошибка при запросе ... байтов памяти"

- данная ошибка возникает, когда недостаточно памяти для работы библиотекаря. Попытка выделения очередного массива в области динамической памяти для размещения внутренних структур оканчивается неудачей. В результате работа не может быть продолжена. Для исправления данной ситуации необходимо освободить динамическую память. Например, выгрузить некоторые резидентные программы, или увеличить свободное место на диске, на котором создается временный системный файл.

АКЦИОНЕРНОЕ ОБЩЕСТВО
НАУЧНО-ТЕХНИЧЕСКИЙ ЦЕНТР

Научно-технический центр Модуль
АЯ 166, Москва, 125190, Россия
Тел: +7 (095) 152-9335
Факс: +7 (095) 152-4661
E-Mail: postmast@module.ru
WWW: http://www.module.ru

Напечатано в России Дата издания: 02.06.99