FORTRAN-80

Microsoft FORTRAN-80 является наиболее популярным в науке и технике языком программирования для микрокомпьютеров 8080/Z80. FORTRAN-80 сопоставим с компиляторами FORTRAN для больших ЭВМ и мини-компьютеров. Пакет FORTRAN-80 поддерживает полный стандарт ANSI FORTRAN X3.9-1966, за исключением типа данных COMPLEX.

**FORTRAN-80 расширяет стандарт ANSI 1966 несколькими способами:**

1. Одно байтовые переменные LOGICAL, могут быть использованы в качестве целочисленных величин в диапазоне от +127 до -127.
2. Циклы DO могут использовать переменные LOGICAL для более компактного, быстрого выполнения небольших циклов.
3. Поддерживается смешанный режим арифметических выражений.
4. Шестнадцатеричные константы.
5. Допускается использование Hollerith литералов (символов).
6. Логические операции над целыми данными. .AND., .OR.,·.NOT., .XOR. могут использоваться в 8-битных, 16-битных или 32-битных логических операциях.
7. Переход в операторах READ/WRITE при обнаружении конца файла или возникновении ошибки. Параметры END=n и ERR=n (где n является меткой оператора) могут быть включены в операторы READ или WRITE для передачи управления на оператор с меткой n при возникновении ошибки или обнаружении конца файла.
8. Операторы ENCODE/DECODE для операций FORMAT в память.
9. Оператор IMPLICIT для изменения типов переменных по умолчанию указанием типа и диапазона начальных букв.
10. Оператор INCLUDE для включения часто используемых подпрограмм, кода или определений из другого файла.
11. Переменные и константы INTEGER\*4 используют 32-бита в диапазоне от +2 147 483 647 до -2 147 483 648.
12. Поддержка CP/M версии 2.х с предоставлением доступа к максимум 65 536 записям в файле большого размера до 8 мегабайт.

**Компилятор FORTRAN-80**

FORTRAN-80 компилирует несколько сотен операторов в минуту за один проход. Это требует не более 27К байт памяти для компиляции большинства программ. Дополнительная память, если она доступна, используется для хранения таблицы символов и оптимизации. Скомпилированные программы являются перемещаемыми модулями, которые связываются между собой при компоновке и загружаются во время выполнения.

Компилятор FORTRAN-80 оптимизирует генерируемый объектный код следующим образом:

**Ликвидация общих подвыражений.**

Общие подвыражения вычисляется один раз, значение автоматически подставляется в последующие вхождения подвыражений.

**Локальная оптимизация.**

В особых случаях небольшие фрагменты кода могут быть заменены на более компактный код. Например: I=I+1 использует инструкцию INX H вместо DAD.

**Свертывание констант.**

Выражения целых констант вычисляются во время компиляции.

**Оптимизация условных переходов.**

Количество условных переходов в арифметических и логических операторах IF сведено к минимуму.

Компилятор обеспечивает диагностический вывод. Описательные сообщения об ошибках включают предыдущие двадцать символов. В конце программы, компилятор генерирует сводные сведения об ошибках. Также производится генерация полного списка символов машинного языка. Он дополняется таблицей адресов, назначенных меткам, переменным и константам.

**Библиотека подпрограмм**

FORTRAN-80 предоставляет обширную библиотеку эффективных подпрограмм. Только необходимые подпрограммы загружаются во время выполнения. Библиотека в стандарте LIB-80 включает в себя:

|  |  |  |  |
| --- | --- | --- | --- |
| ABS | DATAN | DSIN | MAX1 |
| AINT | DATAN2 | DSQRT | MIN0 |
| ALOG | DBLE | EXP | MIN1 |
| ALOG10 | DCOS | FIX | MOD |
| AMAX0 | DEXP | FLOAT | OUT |
| AMAX1 | DIM | IABS | PEEK |
| AMIN0 | DLOG | IDIM | POKE |
| AMIN1 | DLOG10 | IDINT | SIGN |
| AMOD | DSIGN | MAX0 | TANH |
| DABS |  |  |  |

Библиотека также содержит эффективные процедуры для 16-битных и 32-битных целочисленных арифметических и 32-битных и 64-битных арифметических операций с плавающей запятой.

**Ассемблер и компоновщик**

Пакет FORTRAN-80 включает в себя перемещаемый макроассемблер MACRO-80 и перемещающий связывающий загрузчик LINK-80.

Макроассемблер MACRO-80 использует примерно 19 Кбайт. Он включает в себя полностью совместимые с Intel макросредства IRP, IRPC, REPEAT, локальные переменные и EXITM. MACRO-80 также обеспечивает полный набор условных псевдокоманд, управление условным выводом листинга, блоков комментариев, входных списков переменных в восьмеричной или шестнадцатеричной системе счисления. Ассемблер принимает мнемоники Intel 8080 и Zilog Z80. Загрузчик LINK-80 разрешает внешние ссылки объектных модулей. LINK-80 выполняет поиск в библиотеках системных подпрограмм.

Он также создает карту памяти, показывающую расположение основной программы, подпрограмм и области COMMON блоков. LINK-80 требует около 10 Кбайт памяти.

**Драйверы ввода-вывода пользователя**

Пользователи могут написать нестандартные драйверы ввода/вывода для каждого логического устройства, поэтому взаимодействие с нестандартными устройствами в программе на FORTRAN не сложно.

**Система FORTRAN-80**

Версии FORTRAN-80 доступны для различных операционных систем, включая CP/M®Digital Research, TEKDOS Tektronix, ISIS II Intel™, TRSDOS™ Radio Shack Model I и Model II, RDOS GenRad и многих других. Microsoft приветствует возможность адаптировать FORTRAN-80 для OEM-систем

Периодически объявляются обновления FORTRAN-80, и они доступны за минимальную плату.

|  |  |
| --- | --- |
| FORTRAN-80 (включая документацию): | 500$ |
| Только документация: | 20$ |

OEM и дилерские соглашения предоставляются по запросу.

CP/M является зарегистрированным товарным знаком Digital Research

Intel является товарной маркой Intel Corporation

TRSDOS является товарной маркой компании Radio Shack

**Дополнение в: Справочное руководство FORTRAN-80**

**Генератор случайных чисел**

Новая функция RAN, добавлена в библиотечные процедуры FORTRAN, перечисленные в **Таблице 9-2**. Функция RAN является генератором случайных чисел, совместимым с компилятором Microsoft BASIC и интерпретатором BASIC-80.

Сгенерированное случайное число, является вещественным десятичным числом между 0 и 1.

Генератор случайных чисел вызывается оператором следующего вида:

<переменная> = RAN(x)

Если x<0, возвращается первое значение новой последовательности случайных чисел.

Если x=0, снова возвращается последнее сгенерированное случайное число.

Если x>0, создается следующее случайное число в последовательности.

**Дополнение в: Справочное руководство FORTRAN-80**

**Переменные INTEGER\*4**

Существуют следующие ограничения на использование новых расширенных целых данных типа INTEGER\*4:

1. Переменные INTEGER\*4 нельзя использовать в качестве номеров логических устройств в операторах READ или WRITE.
2. Переменные INTEGER\*4 нельзя использовать в качестве индексов массива.
3. Переменные INTEGER\*4 нельзя использовать в качестве индексов в неявных циклах DO.
4. Переменные INTEGER\*4 нельзя использовать в вычисляемых или назначаемых операторах GOTO.

**FORTRAN-80  
Руководство  
пользователя**

Информация в этом документе может изменяться без предварительного уведомления и не представляет собой обязательств со стороны Microsoft. Программное обеспечение, описанное в этом документе, оформлено под лицензионное соглашение или соглашением о неразглашении. Программное обеспечение может использоваться или копироваться только в соответствии с условиями соглашения.

© Microsoft, 1979

CP/M является зарегистрированной торговой маркой Digital Research

8202-342-02

**Дополнение к: FORTRAN-80 Руководство пользователя**

**(для операционных систем CP/M)**

**Добавить в Раздел 1.2, Формат команд**

Компилятор FORTRAN-80 теперь принимает ввод имен файлов и устройств в нижнем регистре на уровне приглашения.

**Добавить в Раздел 2.2, Сообщения об ошибках во время выполнения FORTRAN**

Неустранимая ошибка во время выполнения \*\*FN\*\* была удалена. Эта ошибка возникает, когда программа пытается читать несуществующие файлы. Новый файл DSKDRV.MAC теперь позволяет программисту использовать в операторе READ параметр "ERR=" что бы перейти к ловушке этой ошибки. Если параметр "ERR=" не используется в операторе READ, и предпринимается попытка чтения из несуществующего файла будет возникать неустранимая ошибка \*\*IO\*\*.

**Дополнение к: FORTRAN-80 Руководство пользователя**

**Раздел 4**

**TEKDOS версия 3.1**

Если вы используете TEKDOS версии 3.1 и хотите иметь вставленные символы перевода строки при печати листинга, который отправляются компилятором непосредственно на принтер, необходимо внести следующие исправления в код компилятора FORTRAN версии 3.4:

С помощью программы TEKDOS DEBUG, вставьте 00H (NOP) по адресам 2A95Н, 2A96H и 2A97H.

Эти изменения позволят пользователям принтеров без функции авто перевода каретки печатать листинг компилируемой программы должным образом.

**Microsoft FORTRAN-80**

**Руководство пользователя**

**Содержание**

Раздел 1 Компиляция программ FORTRAN 9

1.1 Запуск компилятора FORTRAN-80 9

1.2 Формат команды 9

1.2.1 Устройства 10

1.2.2 FORTRAN -80 Ключи компиляции 10

1.3 Программы в ПЗУ 11

1.4 Пример компиляции 11

Раздел 2 Сообщения об ошибках 13

2.1 Сообщения об ошибках компилятора FORTRAN 13

Раздел 3 FORTRAN-80 Файлы на диске 16

3.1 Имена файлов по умолчанию 16

3.2 Вызов OPEN 16

3.3 Длина записи 18

Раздел 4 FORTRAN-80 в TEKDOS 18

4.1 Формат команды 18

4.2 Дисковый ввод/вывод и назначение LUN 18

Раздел 1 Введение 21

РАЗДЕЛ 2 Форма программы FORTRAN 21

2.1 Набор символов FORTRAN 21

2.1.1 Буквы 21

2.1.2 Цифры 22

2.1.3 Буквенно-цифровые символы 22

2.1.4 Специальные символы 22

2.2 Формат строки FORTRAN 22

2.3 Операторы 24

2.4 Оператор INCLUDE 24

Раздел 3 Представление данных / формат хранения 25

3.1 Имена и типы данных 25

3.1.1 Имена 25

3.1.2 Типы 25

3.2 Константы 25

3.3 Переменные 27

3.4 Массивы и элементы массивов 28

3.5 Индексы 28

3.6 Выделение памяти данным 29

Раздел 4 Выражения FORTRAN 31

4.1 Арифметические выражения 31

4.2 Вычисление выражений 32

4.3 Логические выражения 32

4.3.1 Реляционные выражения 33

4.3.2 Логические операторы 33

4.4 Текстовые, литеральные и шестнадцатеричные константы в выражениях 34

Раздел 5 Операторы присваивания 35

Раздел 6 Операторы описания 36

6.1 Операторы описания 36

6.2 Объявление массива 36

6.3Операторы определения типа 37

6.4 Операторы EXTERNAL 37

6.5 Операторы DIMENSION 38

6.6 Операторы COMMON 38

6.7 Операторы EQUIVALENCE 39

6.8 Операторы инициализации данных 42

6.9 Оператор IMPLICIT 43

РАЗДЕЛ 7 Операторы управления FORTRAN 43

7.1 Операторы GOTO 44

7.1.1 Безусловный GOTO 44

7.1.2 Вычисляемый GOTO 44

7.1.3 Присваиваемый GOTO 44

7.2 Оператор ASSIGN 45

7.3 Оператор IF 45

7.3.1 Арифметический IF 45

7.3.2 Логический IF 46

7.4 Оператор цикла DO 46

7.5 Оператор CONTINUE 48

7.6 Оператор STOP 49

7.7 Оператор PAUSE 49

7.8 ОператорCALL 49

7.9 ОператорRETURN 49

7.10 ОператорEND 49

РАЗДЕЛ 8 Ввод/Вывод 50

8.1 Форматные операторы READ/WRITE 50

8.1.1 Форматный оператор READ 50

8.1.2 Форматный оператор WRITE 52

8.2 Бесформатные операторы READ и WRITE 53

8.3 Ввод/вывод в файлы на диске 53

8.3.1 Произвольный ввод/вывод на диск 53

8.3.2 Подпрограмма OPEN 54

8.4 Вспомогательные операторы ввода/вывода 54

8.5 Операторы ENCODE/DECODE 54

8.6 Спецификации списка ввода/вывода 55

8.6.1 Типы элементов списка 55

8.6.2 Специальные замечания по спецификации списка 56

8.7 Операторы FORMAT 56

8.7.1 Поля дескрипторов 57

8.7.2 Числовые преобразования 58

8.7.3 Преобразования текста 61

8.7.4 Логические преобразования 62

8.7.5 Дескриптор X 63

8.7.6 Дескриптор P 63

8.7.7 Специальные функции управления операторов FORMAT 64

8.7.8 Обработка FORMAT, список спецификаций и разграничение записей 66

8.7.9 Вывод форматных записей на печать 67

8.7.10 Спецификации формата в массивах 67

Раздел 9 Функции и подпрограммы 67

9.1 Оператор PROGRAM 69

9.2 Оператор-функция 69

9.3 Библиотечные функции 70

9.4 Подпрограммы функции 72

9.5 Конструкция подпрограмм-функций 73

9.6 Ссылки на подпрограмму-функцию 74

9.7 Подпрограммы SUBROUTINE 74

9.8 Конструкция подпрограммSUBROUTINE 75

9.9 Ссылки на подпрограммуSUBROUTINE 75

9.10 Возврат из подпрограмм FUNCTION и SUBROUTINE 76

9.11 Обработка массивов в подпрограммах 77

9.12 Подпрограмма BLOCKDATA 78

9.13 Цепочки программ 78

Приложение A Расширения и ограничения языка 79

Приложение B Интерфейс ввода-вывода 80

ПРИЛОЖЕНИЕ C Вызовы подпрограмм 85

Приложение D Коды символов ASCII 87

Приложение E Ссылки на библиотечные процедуры FORTRAN-80 88

# Раздел 1 Компиляция программ FORTRAN

## 1.1 Запуск компилятора FORTRAN-80

Команда для запуска FORTRAN-80

**F80**

Компилятор FORTRAN-80 возвращает приглашение "\*" показывая, что он готов к приему команд.

**Примечание**

Если вы используете операционную систему TEKDOS, см. [Раздел 4](#_Раздел_4_FORTRAN-80_2) для уточнения правильного формата команды.

## 1.2 Формат команды

Команда вызова FORTRAN-80 состоит из строки имен файлов с дополнительными ключами. Все имена файлов должны следовать соглашениям операционной системы для имен файлов и расширений. Расширения имен файлов по умолчанию, предоставленные программным обеспечением Microsoft, следующие:

|  |  |  |
| --- | --- | --- |
| **Файл** | **CP/M** | **ISIS-II** |
| Исходный файл FORTRAN | FOR | FOR |
| Исходный файл COBOL | COB | COB |
| Исходный файл MACRO-80 | MAC | MAC |
| Перемещаемый объектный файл | REL | REL |
| Файл листинга | PRN | LST |
| Исполняемый файл | COM |  |

Команда передает FORTRAN-80 имя исходного файла для компиляции, имена файлов которые необходимо создать и необходимые ключи. Формат команды FORTRAN-80 следующий:

**objfile,lstfile=sourcefile**

Для создания перемещаемого объектного файла с именем файла по умолчанию (источника) и расширением по умолчанию REL требуются только знак равенства и поле «sourcefile».

В противном случае объектный файл создается, только если заполнено поле «objfile», и файл листинга создается, только если заполнено поле «lstfile».

Для компиляции исходного файла, не создавая объектный файл и файл листинга, поместите только запятую слева от знака "равно". Это удобная процедура, которая позволяет проверить наличие синтаксических ошибок перед компиляцией в объектный файл.

Примеры:

|  |  |
| --- | --- |
| **\*=TEST** | Компиляция программы TEST.FOR и создание объектного файла TEST.REL |
| **\*TESTOBJ=TEST.FOR** | Компиляция программы TEST.FOR и создание объектного файла TESTOBJ.REL |
| **\*TEST,TEST=TEST** | Компиляция программы TEST.FOR, создание объектного файла TEST.REL и файла листинга TEST.PRN. (В ISIS-II файл листинга TEST.LST). |
| **\*,=TEST.FOR** | Компиляция программы TEST.FOR без создания объектного файла и файла листинга. Полезно для проверки ошибок. |

### 1.2.1 Устройства

Любое поле в командной строке FORTRAN-80 может также определить имя устройства. По умолчанию именем устройства в операционной системе CP/M является имя текущего диска. По умолчанию имя устройства в операционной системе ISIS-II это диск 0. Формат команды следующий:

**dev:objfile,dev:lstfile=dev:source file**

Могут использоваться следующие имена устройств:

|  |  |  |
| --- | --- | --- |
| **Устройство** | **CP/M** | **ISIS-II** |
| Диски | A:, B:, C:, ... | :F0:, :F1:, :F2:,... |
| Принтер | LST: | LST: |
| Консоль | TTY: | TTY: |
| Считыватель | HSR: |  |

**Примеры:**

|  |  |
| --- | --- |
| **\*,TTY:=TEST** | Компиляция исходного файла TEST.FOR и вывод списка программ на консоли. Объектный код не генерируется. Полезно для проверки ошибок. |
| **\*SMALL,TTY:=B:TEST** | Компиляция TEST.FOR (находится на диске B), создание объектного файла SMALL.REL и вывод списка программ на консоль. |

### 1.2.2 FORTRAN-80 Ключи компиляции

Ключ является буквой, которая добавляется в командной строке, после косой черты. Он указывает выполнение необязательной задачи во время компиляции. Может быть использовано более одного ключа, но каждому должна предшествовать косая черта. (В операционной системой TEKDOS, ключам предшествует запятая или пробел. См. [Раздел 4](#_Раздел_4_FORTRAN-80).) Все ключи не являются обязательными.

**Доступные** **ключи**:

|  |  |
| --- | --- |
| Ключ | Действие |
| **O** | Печать в листинге всех адресов и т.п. в восьмеричном формате. |
| **H** | Печать всех адресов и т.п. в листинге в шестнадцатеричном формате (по умолчанию). |
| **N** | Не генерировать в листинге объектный код. Выводить в листинг только исходный код FORTRAN. |
| **R** | Генерация объектного файла. |
| **L** | Генерация файла листинга. |
| **P** | Каждый ключ /P выделяет дополнительные 100 байт стекового пространства для использования во время компиляции. Используйте /P, если при компиляции возникают ошибки переполнения стека. В противном случае не требуется. |
| **M** | Указывает компилятору, что создаваемый код должен быть в форме, которая может быть загружена в ПЗУ. Если указан ключ /M, сгенерированный код будет отличаться от обычного тем, что:  1. Коды операторов FORMAT будут размещены в области программы вместе с командами "JMP", обходящими эти операторы.  2. Блоки параметров при вызовах подпрограмм (имеющих более 3 формальных параметров) будут инициализированы во временя выполнения программы, а не при компиляции. |

**Примеры:**

|  |  |
| --- | --- |
| **\*,TTY:=MYPROG/N** | Компиляция MYPROG.FOR и вывод списка файлов программ на терминал, но без сгенерированного кода. |
| **\*=TEST/L** | Компиляция TEST.FOR, создание объектного файла TEST.REL и файла листинга TEST.PRN. (в ISIS-II, файл листинга TEST.LST). |
| **\*=BIGGONE/P/P** | Компиляция файла BIGGONE.FOR и создание объектного файла BIGGONE.REL. Компилятор выделит дополнительно 200 байт стека. |

## 1.3 Программы в ПЗУ

Если программа FORTRAN предназначена для размещения в ПЗУ, программист должен иметь в виду следующее:

1. Операторы DATA не должны использоваться для инициализации ОЗУ. Такая инициализация осуществляется загрузчиком и, поэтому, не будет происходить при исполнении. Переменные и массивы могут быть инициализированы во время выполнения с помощью операторов присваивания или операции чтения.
2. Операторы FORMAT не должны быть считаны в процессе выполнения.
3. Если используется стандартная библиотека подпрограмм ввода/вывода, дисковые файлы не должны быть открыты с любым LUN, кроме 6, 7, 8, 9, 10. Если необходимы другие LUN для дисковых операций ввода/вывода, процедура $LUNTB должна быть перекомпилирована с надлежащими адресами, указывающими на подпрограмму драйвера диска.

Библиотечная процедура $INIT до выполнения программы устанавливает указателю стека значение адреса верхней части доступной памяти (как указано в операционной системе).

Соглашение о вызовах:

**LD BC,<адрес возврата>**

**JP $INIT**

Если сгенерированный код предназначен для другой машины, эту процедуру, вероятно, следует переписать. Исходный код стандартной процедуры инициализации присутствует на диске в виде файла "INIT.MAC". Только часть этой подпрограммы, устанавливающая указатель стека, должна быть изменена пользователем. Библиотека FORTRAN уже содержит стандартную процедуру инициализации.

## 1.4 Пример компиляции

A>**F80**

\***EXAMPL,TTY:=EXAMPL**

FORTRAN-80 Ver. 3.4 Copyright 1978, 79, 80 (C) By Microsoft

BYTES: 31663

Created: 19-Feb-85

1 PROGRAM EXAMPLE

\*\*\*\*\* 0000' LD BC,$$L

\*\*\*\*\* 0003' JP $INIT

2 INTEGER X

3 I = 2\*\*8 + 2\*\*9 + 2\*\*10

4 DO 1 J=1,5

\*\*\*\*\* 0006' LD HL,0700

\*\*\*\*\* 0009' LD (I),HL

5 C CIRCULAR SHIFT I LEFT 3 BITS -- RESULT IN X

6 CALL CSL3(I,X)

\*\*\*\*\* 000C' LD HL,0001

\*\*\*\*\* 000F' LD (J),HL

7 WRITE(3,10) I,X

\*\*\*\*\* 0012' LD DE,X

\*\*\*\*\* 0015' LD HL,I

\*\*\*\*\* 0018' CALL CSL3

\*\*\*\*\* 001B' LD DE,10L

\*\*\*\*\* 001E' LD HL,[03 00 00 00]

\*\*\*\*\* 0021' CALL $W2

8 1 I=X

\*\*\*\*\* 0024' LD BC,X

\*\*\*\*\* 0027' LD DE,I

\*\*\*\*\* 002A' LD HL,[01 00 00 00]

\*\*\*\*\* 002D' LD A,03

\*\*\*\*\* 002F' CALL $I0

\*\*\*\*\* 0032' CALL $ND

9 10 FORMAT(2I15)

\*\*\*\*\* 0035' LD HL,(X)

\*\*\*\*\* 0038' LD (I),HL

\*\*\*\*\* 003B' LD HL,(J)

\*\*\*\*\* 003E' INC HL

\*\*\*\*\* 003F' LD A,05

\*\*\*\*\* 0041' SUB L

\*\*\*\*\* 0042' LD A,00

\*\*\*\*\* 0044' SBC H

\*\*\*\*\* 0045' JP P,000F'

10 END

\*\*\*\*\* 0048' CALL $EX

\*\*\*\*\* 004B' 01 00 00 00

\*\*\*\*\* 004F' 03 00 00 00

Program Unit Length=0053 (83) Bytes

Data Area Length=000D (13) Bytes

Subroutines Referenced:

$I0 $INIT CSL3

$W2 $ND $EX

Variables:

X 0001" I 0003" J 0005"

Labels:

$$L 0006' 1L 0035' 10L 0007"

См. Раздел 2.9 Руководства Microsoft Утилиты для получения листинга MACRO-80 подпрограммы CSL3.

# Раздел 2 Сообщения об ошибках

## 2.1 Сообщения об ошибках компилятора FORTRAN

Компилятор FORTRAN-80 обнаруживает два вида ошибок: предупреждения и неустранимые ошибки. Когда выдается предупреждение, компиляция продолжается со следующего элемента в строке исходного кода. При обнаружении неустранимой ошибки, компилятор игнорирует остальные логические строки, включая любое продолжение строки. Предупреждающим сообщениям предшествует знак процента (%), а неустранимым ошибкам вопросительный знак (?). Дальше напечатан номер строки в редакторе, если такой имеется, или физический номер строки. Затем следует код ошибки или сообщение об ошибке и 20 последних символов отсканированных во время обнаружения ошибки.

**Пример**

**?Line 25: Mismatched Parentheses: I=(I+J))**

**%Line 16: Missing Integer Variable: I(2\*2,**

При возникновении ошибки любого типа, программа должна быть изменена так, что бы она компилировалась без ошибок. Нет гарантий, что программа, которая компилируется с ошибками, будет выполнять разумно.

**Неустранимые ошибки**

| **Номер ошибки** | **Сообщение** |  |
| --- | --- | --- |
| 100 | Illegal Statement Number | Неверный номер оператора |
| 101 | Statement Unrecognizable or Misspelled | Оператор не опознан или с ошибкой |
| 102 | Illegal Statement Completion | Неверное окончание оператора |
| 103 | Illegal DO Nesting | Неверная вложенность оператора DO |
| 104 | Illegal Data Constant | Неверные данные константы |
| 105 | Missing Name | Отсутствует имя |
| 106 | Illegal Procedure Name | Неверное имя процедуры |
| 107 | Invalid DATA Constant or Repeat Factor | Неверная константа или повторитель в DATA |
| 108 | Incorrect Number of DATA Constants | Неверное количество констант в DATA |
| 109 | Incorrect Integer Constant | Некорректная числовая константа |
| 110 | Invalid Statement Number | Неверный номер оператора |
| 111 | Not a Variable Name | Недопустимое имя |
| 112 | Illegal Logical Form Operator | Неверная форма логического оператора |
| 113 | Data Pool Overflow | Переполнение пула данных |
| 114 | Literal String Too Large | Слишком длинная литеральная строка |
| 115 | Invalid Data List Element in I/O | Неверный список элементов при вводе/выводе |
| 116 | Unbalanced DO Nest | Несбалансированный DO |
| 117 | Identifier Too Long | Слишком длинный идентификатор |
| 118 | Illegal Operator | Неверный оператор |
| 119 | Mismatched Parenthesis | Несоответствие скобок |
| 120 | Consecutive Operators | Последовательность операторов |
| 121 | Improper Subscript Syntax | Неправильный синтаксис подстроки |
| 122 | Illegal Integer Quantity | Недопустимое целое число |
| 123 | Illegal Hollerith Construction | Недопустимая текстовая конструкция |
| 124 | Backwards DO reference | Ссылка внутрь DO |
| 125 | Illegal Statement Function Name | Недопустимое имя оператора функции |
| 126 | Illegal Character for Syntax | Недопустимый синтаксис символа |
| 127 | Statement Out of Sequence | Оператор вне последовательности |
| 128 | Missing Integer Quantity | Отсутствует целое число |
| 129 | Invalid Logical Operator | Неверный логический оператор |
| 130 | Illegal Item Following INTEGER or | Неверный пункт после INTEGER или |
| 131 | Illegal Item in Type Declaration | Неверный пункт в описании типа |
| 132 | Illegal Mixed Mode Operation | Недопустимое смешение типов |
| 133 | Function Call with No Parameters | Вызов функции без параметра |
| 134 | Stack Overflow | Переполнение стека |
| 135 | Illegal Statement Following Logical IF | Недопустимый оператор за логическим IF |
| 136 | Wrong Number of Subscripts | Неверный индекс |
| 137 | File Not Found | Файл не найден |

**Предупреждения:**

|  |  |  |
| --- | --- | --- |
| 0 | Duplicate Statement Label | Повторение метки оператора |
| 1 | Illegal DO Termination | Неверное окончание оператора DO |
| 2 | Block Name = Procedure Name | Присвоение блоку данных имени процедуры |
| 3 | Array Name Misuse | Неправильное имя массива |
| 4 | COMMON Name Usage | Использование имени COMMON |
| 5 | Wrong Number of Subscripts | Неправильное число индексов |
| 6 | Array Multiply EQUIVALENCEd within a Group | Несколько EQUIVALENCE одного массива в группе |
| 7 | Multiple EQUIVALENCE of COMMON | Несколько одинаковых COMMON |
| 8 | COMMON Base Lowered | Пониженная основа COMMON |
| 9 | Non-COMMON Variable in BLOCK DATA | Не COMMON переменная в BLOCK DATA |
| 10 | Empty List for Unformatted WRITE | Пустой список бесформатного WRITE |
| 11 | Non-Integer Expression | Не целое выражение |
| 12 | Operand Mode Not Compatible with Operator | Режим операнда не совместим с оператором |
| 13 | Mixing of Operand Modes Not Allowed | Смешивание режимов операндов не допускается |
| 14 | Missing Integer Variable | Отсутствует целая переменная |
| 15 | Missing Statement Number on FORMAT | Отсутствует номер оператора в FORMAT |
| 16 | Zero Repeat Factor | Нулевой повторитель FORMAT |
| 17 | Zero Format Value | Нулевое значение FORMAT |
| 18 | Format Nest Too Deep | Вложенность FORMAT слишком глубока |
| 19 | Statement Number Not FORMAT Associated | Номер оператора не связан с FORMAT |
| 20 | Invalid Statement Number Usage | Использован неверный номер оператора |
| 21 | No Path to this Statement | Нет пути к этому оператору |
| 22 | Missing Do Termination | Отсутствует окончание DO |
| 23 | Code Output in BLOCK DATA | Код выхода в BLOCK DATA |
| 24 | Undefined Labels Have Occurred | Переход на неопределенную метку |
| 25 | RETURN in a Main Program | В главной программе оператор RETURN |
| 26 | STATUS Error on READ | Статус ошибки при READ |
| 27 | Invalid Operand Usage | Использован неверный операнд |
| 28 | Function with no Parameter | Функция без параметров |
| 29 | Hex Constant Overflow | Переполнение шестнадцатеричной константы |
| 30 | Division by Zero | Деление на 0 |
| 32 | Array Name Expected | Ожидаемое имя массива |
| 33 | Illegal Argument to ENCODE/DECODE | Недопустимый аргумент ENCODE/DECODE |

**Сообщения об ошибках во время выполнения FORTRAN**

| **Код** | **Значение** |  |
| --- | --- | --- |
| **IB** | Input Buffer Limit Exceeded | Превышен лимит входного буфера |
| **TL** | Too Many Left Parentheses in FORMAT | Слишком много левых скобок в FORMAT |
| **OB** | Output Buffer Limit Exceeded | Превышен размер выходного буфера |
| **DE** | Decimal Exponent Overflow (Number in input stream had an exponent larger than 99) | Десятичное переполнение экспоненты (у числа экспонента, больше, чем 99) |
| **IS** | Integer Size Too Large | Размер целого числа, слишком велик |
| **BE** | Binary Exponent Overflow | Переполнения двоичной экспоненты |
| **IN** | Input Record Too Long | Входная запись слишком длинная |
| **OV** | Arithmetic Overflow | Арифметическое переполнение |
| **CN** | Conversion Overflow on REAL to INTEGER Conversion | Переполнение при преобразовании вещественного числа в целое |
| **SN** | Argument to SIN Too Large | Аргумент SIN слишком большой |
| **A2** | Both Arguments of ATAN2 are 0 | Оба аргумента ATAN2 равны 0 |
| **IO** | Illegal I/O Operation | Неверная операция ввода/вывода |
| **BI** | Buffer Size Exceeded During Binary I/O | Превышен размер буфера в процессе операции двоичного ввода/вывода |
| **RC** | Negative Repeat Count in FORMAT | Отрицательное число повторений в FORMAT |
| **FW** | FORMAT Field Width is Too Small | Поле FORMAT слишком мало |

**Неустранимые ошибки:**

|  |  |  |
| --- | --- | --- |
| **ID** | Illegal FORMAT Descriptor | Незаконный дескриптор FORMAT |
| **F0** | FORMAT Field Width is Zero | Ширина поля в FORMAT равна нулю |
| **MP** | Missing Period in FORMAT | Отсутствует период в FORMAT |
| **IR** | Real Number Written to INTEGER Format Field | Вещественное число записано в поле формата INTEGER |
| **IT** | I/O Transmission Error | Ошибка ввода/вывода |
| **ML** | Missing Left Parenthesis in FORMAT | Отсутствует левая круглая скобка в FORMAT |
| **DZ** | Division by Zero, REAL or INTEGER | Деление на ноль, REAL или INTEGER |
| **LG** | Illegal Argument to LOG Function (Negative or Zero) | Недопустимый аргумент для функции LOG (отрицательное или нулевое) |
| **S** | Illegal Argument to SQRT Function (Negative) | Недопустимый аргумент для функции SQRT (минус) |
| **DT** | Data Type Does Not Agree With FORMAT Specification | Тип данных не согласован с спецификацией FORMAT |
| **EF** | EOF Encountered on READ | Оператор READ обнаружил конец файла |

Ошибки времени выполнения, обрамленные звездочками, выводятся в следующем виде:

**\*\*FW\*\* at address XXXX\*\***

с указанием места возникновения ошибки.

Неустранимые ошибки вызывают прекращение выполнения программы (управление передается операционной системе). После предупреждения выполнение программы продолжается после вывода сообщения об ошибке. Однако, после 20-го предупреждения, выполнение программы прекращается.

**Примечание**

В редких случаях FORTRAN-80 выводит вместо кода внутренней ошибки "??". Это указывает на внутреннюю неисправность во время выполнения. Если вы получаете код ошибки "??", свяжитесь с Microsoft и сообщите об условиях, при которых появилось сообщение.

# Раздел 3 FORTRAN-80 Файлы на диске

**Примечание**

Если вы используете операционную систему TEKDOS, смотрите Раздел 4.2, [Дисковые операции ввода/вывода и назначение LUN](#_4.2_Дисковый_ввод/вывод).

## 3.1 Имена файлов по умолчанию

Дисковый файл (произвольного или последовательного доступа), открываемый операторами READ или WRITE имеет имя по умолчанию, которое зависит от операционной системы и номера логического устройства (LUN). Для CP/M и ISIS-II имена файлов по умолчанию следующие:

FORT06.DAT, FORT07.DAT,..., FORT10.DAT

(Имена файлов по умолчанию, используемые в TEKDOS, см. [Раздел 4](#_Раздел_4_FORTRAN-80_1).)

В каждом случае номер логического устройства будет включен в имя файла по умолчанию.

## 3.2 Вызов OPEN

Вместо использования READ или WRITE, файл на диске может быть открыт с помощью подпрограммы OPEN (см. [Справочное руководство FORTRAN-80, Раздел 8.3.2](#_8.3.2_Подпрограмма_OPEN)). В CP/M формат вызова OPEN следующий:

**CALL OPEN (LUN, Filename, Drive)**

где:

|  |  |  |
| --- | --- | --- |
| **LUN** | - | номер логического устройства связанный с файлом (должен быть целой константой или целой переменной со значением между 1 и 10). |
| **Filename** | - | имя ASCII, которое операционная система будет ассоциировать с файлом. Имя файла должно быть текстовой или буквенной константой, или именем переменной или массива, где переменная или массив содержат ASCII имя. Имя файла должно быть заполнено пробелами точно до одиннадцати символов. |
| **Drive** | - | номер дискового устройства, на котором файл существует или будет существовать (должен быть целой константой, или целой переменной со значением допустимым в операционной системе). Если указан диск 0 - предполагается текущий диск, 1 - диск А, 2 - диск B и т.д. |

Форма вызова оператора OPEN в ISIS-II:

**CALL OPEN (LUN, Filename)**

где:

|  |  |  |
| --- | --- | --- |
| **LUN** | - | номер логического устройства связанный с файлом (должен быть целой константой или целой переменной со значением между 1 и 10). |
| **Filename** | - | имя ASCII, которое операционная система будет ассоциировать с файлом. Имя файла должно быть текстовой или буквенной константой, или именем переменной или массива, в которых содержится ASCII имя. Имя файла должно быть в форме поддерживаемой ISIS-II, то есть, имя устройства, окружённое двоеточиями, за которым следует имя до 6 символов, точка, расширение до 3 символов, и пробел (или другой не буквенно-цифровой символ). Имя файла должно заканчиваться не буквенно-цифровым символом. |

Ниже приведены примеры допустимых вызовов OPEN в ISIS-II:

CALL OPEN (6, ':F1:F00.DAT ')

CALL OPEN (1, ':F5:TESTFF.TMP ')

CALL OPEN (4, ':F3:A.B ')

## 3.3 Длина записи

Длина записи любых файлов произвольного доступа в CP/M или ISIS-II может быть 128 байт (1 сектор). Поэтому, рекомендуется, чтобы любой файл, который вы захотите считать в произвольном порядке, был создан с помощью операторов произвольного доступа FORTRAN (или Microsoft BASIC). Файлы произвольного доступа созданные таким способом (используя двоичный, или форматный операторы WRITE), всегда имеют 128-байтовые записи. Если оператор WRITE не передает достаточно данных, чтобы заполнить запись до 128 байт, то конец записи заполняется нулями (символами NULL).

# Раздел 4 FORTRAN-80 в TEKDOS

## 4.1 Формат команды

Формат команды для FORTRAN-80 в операционной системе TEKDOS слегка отличается. Компилятор FORTRAN-80 принимает только командную строку. Приглашение не отображается и интерактивные команды не принимаются. Команды имеют такой же формат, как TEKDOS команды ассемблера, т.е. три имени файла или имени устройств, а также дополнительные ключи компиляции.

**F80 [objfile] [lstfile] sourcefile [sw1] [sw2…]**

Параметры objfile и lstfile не являются обязательными. Эти файлы не будут созданы, если параметры опущены, однако сообщения об ошибках все равно будет отображаться на консоли. Доступны ключи, описаны в [Разделе 1](#Ключи) настоящего руководства. Кроме этого, ключи разделяются запятыми или пробелами вместо косой черты.

## 4.2 Дисковый ввод/вывод и назначение LUN

(См. [FORTRAN-80 Справочное руководство, Раздел 8.3.](#_8.3_Ввод/вывод_в_1))

FORTRAN-80 в TEKDOS не поддерживает произвольный доступ к файлу на диске. Поддерживаются только последовательный доступ к файлу.

Логические устройства 1-4 назначаются консоли и могут быть использованы для ввода или вывода. Логические устройства 5-10 привязаны к DSKDRV. Они по умолчанию для последовательных дисковых файлов с именами FOR05DAT, ... FOR10DAT.

Эти устройства могут быть повторно назначены любому файлу или устройству с использованием вызова OPEN. Формат вызова OPEN:

**CALL OPEN(LUN, filename)**

где LUN номер логического устройства (целая переменная или константа между 5 и 10), а также filename (имя файла) с текстовой или буквенные константой или переменной, содержащей ASCII имя файла и/или устройства. Имя файла не может иметь более 11 символов, и оно должно быть закончено пробелом или нулевым символом.

**Примеры:**

**CALL OPEN(8,'TSTFIL/1 ')**

Открыть файл TSTFIL на устройстве 1 и привязать его к LUN 8

**CALL OPEN(5,'REM0')**

Открыть LUN 5 для устройства REM0

**FORTRAN-80  
Справочное  
руководство**

Информация в этом документе может быть изменена без предварительного уведомления и не представляет собой обязательств со стороны Microsoft. Программное обеспечение, описанное в этом документе, поставляется в соответствии с лицензионным соглашением или соглашением о неразглашении. Программное обеспечение может использоваться или копироваться только в соответствии с условиями соглашения.

© Microsoft, 1979

CP/M является зарегистрированной торговой маркой Digital Research

8201-342-03

**Microsoft FORTRAN-80**

**Справочное руководство**

**Содержание**

# Раздел 1 Введение

FORTRAN является универсальным, проблемно-ориентированным языком программирования, разработанным для упрощения подготовки и выполнения компьютерных программ. Название языка FORTRAN является аббревиатурой FORmula TRANslator (транслятор формул).

Синтаксические правила использования языка являются строгими и требуют от программиста полностью определять особенности задачи с помощью точной последовательности операторов. Эти операторы, называются исходной программой. Компилятор под названием FORTRAN переводит исходную программу в объектную на машинный язык компьютера, на котором выполняется программа.

Это руководство определяет исходный язык FORTRAN для 8080 и Z-80 микрокомпьютеров. Этот язык соответствует американскому национальному стандарту языка FORTRAN, описанному в документе ANSI X3.9-1966, утвержденному 7 марта 1966 года, добавляет несколько расширений языка и имеет некоторые ограничения. Расширения и ограничения языка описаны в тексте настоящего документа и перечислены в [Приложении A](#_Приложение_A_Расширения).

**Примечание**

Этот FORTRAN отличается от стандарта, тем, что он не поддерживает тип данных COMPLEX.

В настоящем руководстве приведены примеры для иллюстрации построения и использования элементов языка. Программист должен быть знаком со всеми аспектами языка, чтобы в полной мере воспользоваться его возможностями.

[Раздел 2](#_Раздел_2_Форма) описывает формы и компоненты исходной программы FORTRAN-80. [Разделы 3](#_Раздел_3_Представление_1) и [4](#_Раздел_4_Выражения_2) - определения типов данных и их выразительные отношения. Разделы с [5](#_Раздел_5_Операторы_3) по [9](#_Раздел_9_Функции_7) описывают правила построения и использования операторов различных классов.

# Раздел 2 Форма программы FORTRAN

Исходные программы FORTRAN-80 состоят из одного программного модуля под названием основная программа, и любого количества программных модулей называемых подпрограммы. Обсуждение типов подпрограмм, методов их написания и использования находится в [Разделе 9](#_Раздел_9_Функции_6) данного руководства.

Программы и модули подпрограмм состоят из упорядоченного множества операторов, которые точно описывают методику решения задачи, и которые также определяют информацию, которая будет использоваться компилятором FORTRAN во время компиляции объектной программы. Каждый оператор записывается с использованием набора символов FORTRAN и предписанного формата строки.

## 2.1 Набор символов FORTRAN

Чтобы упростить ссылки и объяснение, набор символов FORTRAN разделен на четыре подмножества, и каждому присвоено имя.

### 2.1.1 Буквы

A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, $

Между прописными и строчными буквами нет различия. Однако для ясности и четкости, рекомендуется использовать прописные буквы.

### 2.1.2 Цифры

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Строки из цифр, представляют собой числовые величины, которые, как правило, интерпретируется как десятичные числа. Тем не менее, в некоторых операторах, используется интерпретация в шестнадцатеричной системе счисления. В этом случае буквы A, B, C, D, E, F могут быть использованы в качестве шестнадцатеричных цифр. Шестнадцатеричные цифры используются в операторах объявления, в которых допустима такая нотация.

### 2.1.3 Буквенно-цифровые символы

Подмножество символов, составленных из всех букв и всех цифр.

### 2.1.4 Специальные символы

Пробел

= Знак равенства

+ Знак "плюс"

- Знак "минус"

\* звездочка

/ Наклонная черта

( Левая круглая скобка

) Правая круглая скобка

, Запятая

. Десятичная точка

**Примечания**:

1. Строки программы FORTRAN состоят из 80 символьных позиций или столбцов, пронумерованных 1 - 80. Они разделены на четыре поля.
2. Следующие специальные символы классифицируются как арифметические операторы и имеют важное значение в операторах арифметических выражений.

+ Сложение или положительное значение

- Вычитание или отрицательное значение

\* Умножение

/ Деление

1. У других специальных символов есть определенное назначение в синтаксическом выражении языка FORTRAN и в конструкции операторов FORTRAN.
2. Любой печатаемый символ может появиться в текстовых или литеральных полях.

## 2.2 Формат строки FORTRAN

Строка программы FORTRAN состоит максимум из 80 позиций, разделенных на четыре поля:

1. Поле **метки оператора** (или число) - позиции с 1 по 5 (См. [Определение меток](#Метка)).
2. Поле символа продолжения позиция 6
3. Поле **оператора** позиции с 7 по 72
4. Поле **идентификации** позиции с 73 по 80

**Идентификационное поле** доступно для любого использования программистом и будет проигнорировано компилятором FORTRAN.

Строки оператора FORTRAN размещаются в позициях с 1 по 72 отформатированных в соответствии с типами строк. Четыре типа строк, их определение и формат следующие:

1. **Начальная строка** - первая или единственная строка каждого оператора.
2. Позиции 1-5 могут содержать метку оператора для его идентификации.
3. Позиция 6 должна быть пустой.
4. Позиции 7-72 содержат весь оператор, или его часть.
5. Начальная строка может начинаться в любом месте поля оператора.

**Пример:**

C ОПЕРАТОР НИЖЕ СОСТОИТ ИЗ НАЧАЛЬНОЙ СТРОКИ

C

A= .5\*SQRT(3-2.\*C)

1. **Строка продолжения** используется, если требуются дополнительные строки, чтобы завершать кодирование оператора, начатое в начальной строке.
2. Позиции с 1 по 5 игнорируются, если позиция 1 не содержит символ C.
3. Если позиция 1 содержит символ C, это - строка комментария.
4. Позиция 6 должна содержать любой символ кроме нуля или пробела.
5. Позиции с 7 по 72 содержат продолжение оператора.
6. Строк продолжения может быть несколько для завершения оператора.

**Пример:**

C ОПЕРАТОРЫ НИЖЕ - НАЧАЛЬНАЯ СТРОКА И

C 2 СТРОКИ ПРОДОЛЖЕНИЯ

C

63 BETA(1,2) =

1 A6BAR\*\*7-(BETA(2,2)-A5BAR\*50

2 +SQRT(BETA(2,1)))

1. **Строка комментария** используется для аннотации исходной программы для удобства программиста.
2. Позиция 1 содержит букву C.
3. Позиции 2-72 используются в любом желаемом формате, чтобы выразить комментарий, или они могут быть оставлены пустыми.
4. За строкой комментария могут следовать только начальная строка, строка с оператором END, или другая строка комментария.
5. Строки комментариев не оказывают никакого влияния на объектную программу, и игнорируются компилятором FORTRAN, за исключением отображения в листинге программы.

**Пример:**

C СТРОКИ КОММЕНТАРИЕВ ОБОЗНАЧАЮТСЯ

C СИМВОЛОМ C В ПОЗИЦИИ 1.

C ЭТО СТРОКИ КОММЕНТАРИЕВ

1. Строка с оператором END - **заключительная строка** модуля программы.
2. Позиции 1-5 могут содержать метку оператора.
3. Позиция 6 должна быть пустой.
4. Позиции 7-72 содержат оператор END.
5. Каждый модуль программы FORTRAN должен содержать строку с оператором END в качестве своей последней строки, чтобы информировать компилятор о физическом окончании программного модуля.
6. Строка с оператором END может следовать за любой строкой другого типа.

**Метка** оператора может быть размещена в позициях 1-5 в начальной строке оператора FORTRAN и используется для ссылки на нее в других операторах.

Следующие соображения регламентируют использование меток оператора:

1. Метка - целое число от 1 до 99999.
2. Метка имеет числовое значение, ведущие нули и пробелы не являются значимыми.
3. Метка должна быть уникальной в пределах программного модуля.
4. Метка в строке продолжения игнорируется компилятором FORTRAN.

**Пример:**

C ПРИМЕРЫ МЕТОК ОПЕРАТОРОВ

C

1

101

99999

763

## 2.3 Операторы

Отдельные операторы имеют дело с конкретными аспектами процедуры, описанной в программном модуле, и классифицируются как выполняемые или невыполняемые.

**Выполняемые операторы** указывают действие, и компилятор FORTRAN для них генерирует инструкции объектной программы. Существует три типа выполняемых операторов:

1. Операторы присваивания.

2. Операторы управления.

3. Операторы ввода/вывода.

**Невыполняемые операторы** описывают компилятору природу и расположение данных и предоставляют информацию о форматах ввода/вывода и инициализации данных в объектной программе во время загрузки и выполнения программы. Существует пять типов невыполняемых операторов:

1. Операторы описания.

2. Операторы инициализации DATA.

3. Операторы FORMAT.

4. Операторы определения FUNCTION.

5. Операторы подпрограмм.

В [Разделах 5](#_Раздел_5_Операторы_2) – [9](#_Раздел_9_Функции_5) описаны правила использования и составления операторов различных типов.

## 2.4 Оператор INCLUDE

Оператор INCLUDE заставляет компилятор включить внешнюю исходную программу FORTRAN в текущую программу. Формат оператора следующий:

**INCLUDE <filename>**

Использование INCLUDE, избавляет от необходимости повторять часто используемую последовательность операторов в текущем исходном файле.

# Раздел 3 Представление данных / формат хранения

Язык FORTRAN предписывает точный способ идентификации данных, используемых в программе FORTRAN по имени и типу.

## 3.1 Имена и типы данных

### 3.1.1 Имена

1. **Константа** - явно установленные данные.
2. **Переменная** - данные, обозначенные символическим именем.
3. **Массив** - упорядоченный набор данных в 1, 2 или 3-х измерениях.
4. **Элемент массива** - один из членов набора данных в массиве.

### 3.1.2 Типы

1. **Integer** - точное представление целых чисел (положительных, отрицательных или нуля), имеющих точность до 5 цифр в диапазоне от -32 768 до +32 767 включительно   
   (-215 до 215-1).
2. **Real** - приближения вещественных чисел обычной точности (положительных, отрицательных или нуля) представленных в компьютере в 4 байтах в формате с плавающей точкой. Вещественные числа точны до 7 знаков и их величина может находиться в пределах от 10-38 до 1038 (между 2-127 и 2127).
3. **Double Precision** - приближения вещественных чисел двойной точности (положительных, отрицательных или нуля) представленных в компьютере в 8 байтах в формате с плавающей точкой. Числа двойной точности точны до 16 знаков в том же диапазоне, что и вещественные числа.
4. **Logical** - одно байтовое представление истинности имеющее значение "TRUE" или "FALSE". "FALSE" имеет внутреннее представление 0. Константа .TRUE. имеет значение -1, однако в логическом операторе IF любое ненулевое значение будет рассматриваться как .TRUE. Кроме того, логический тип данных может быть использован в качестве одно байтового целого числа в диапазоне от -128 до +127 включительно.
5. **Hollerith** - строка из любого количества символов из набора символов компьютера. Все символы, включая пробелы существенны. Текстовые данные используют один байт для хранения каждого символа в строке.
6. **Расширенное целое** - INTEGER\*4 является представлением расширенной точности с использованием 32-битного дополнение до двух (четырех 8-битных байтов) до 9 значащих цифр в диапазоне от -2147483648 до +2147483647 включительно (-231 до 231-1).

Переменные INTEGER\*4 не могут использоваться в качестве номера логического устройства (LUN), индексов массива, подразумеваемых индексов цикла типа DO, или переменной управления в вычисляемых или назначаемых операторах GO TO.

## 3.2 Константы

Константы FORTRAN определяются явно с указанием их действительного значения. Символ плюс (+) не должен обязательно предшествовать положительному значению констант.

Форматы для записи констант показаны в **Таблице 3-1**.

Таблица 3 - 1 Форматы констант

| **Тип** | **Форматы и правила использования** | **Примеры** |
| --- | --- | --- |
| **INTEGER** | 1. От 1 до 5 десятичных цифр интерпретируется как десятичное число. | -763  1  +00672 |
| 1. Предшествующий знак плюс (+) или минус (-) не является обязательным. | -32768  +32767 |
| 1. Символы десятичной точки (.) или запятой (,) не допускаются, но разрешены пробелы в исходной программе. | - 32 767 |
| 1. Диапазон значений: от -32768 до +32767  (т.е.-215 до 215-1). |  |
| **REAL** | 1. Десятичное число с точностью до 7 цифр и представлено в одном из следующих форм:   a. + или -.f+ или -i.f  b. + или -i.E+ или -e  + или -.fE+ или -e  + или -i.fE+ или -e  где i, f и e в каждой из строк, представляет собой целую часть, дробную часть, и экспоненту соответственно.   1. Символы плюс (+) и минус (-) не являются обязательными. 2. В форме, показанной в пункте 1b выше, если f представляет какую-либо из форм, предшествующих E+ или -e (т.е., fE+ или -e), значение константы интерпретируется как f умноженное на 10e, где -38≤e≤38. 3. Если константа предшествующих E+ или -е содержит больше значащих цифр, чем позволяет точность для действительных данных, происходит усечение, и будут представлены только значимые цифры. | 345.  -.345678  +345.678  +.3E3  -73E4 |
| **DOUBLE PRECISION** | Десятичное число с точностью до 16 цифр. Все форматы и правила идентичны, для вещественных констант, за исключением использования D вместо E. Отметим, что вещественная константа считается одинарной точности, если она не содержит экспоненту "D". | +345.678  +.3D3  -73D4 |
| **LOGICAL** | .TRUE., генерирует ненулевой байт (шестнадцатеричное FF) и .FALSE. генерирует байт, в котором все биты равны 0.  Если логическая переменная используются в качестве одного-байтового целого числа, ее правила использования такие же, как для типа INTEGER, за исключением того, что диапазон допустимых значений от -128 до +127 включительно. | .TRUE.  .FALSE. |
| **LITERAL** | В литеральной форме любое число символов может быть включено между одиночными кавычками. Форма выглядит следующим образом:  'X1X2X3...Xn'  где каждый xi это любой символ, кроме символа '. Для представления знака кавычек в строке могут использоваться две кавычки подряд, то есть, если символ X2 должен быть символом кавычки, строка имеет следующий вид:  'X1''X3...Xn' |  |
| **HEXADECIMAL** | 1. За символом Z или X подряд следуют до 4 шестнадцатеричных цифр (0-9 и A-F) и одинарная кавычка признается в качестве шестнадцатеричного значения. 2. Значение шестнадцатеричной константы при сохранении выравнивается по правому краю. | Z'12'  X'AB1F1'  Z'FFFF'  X'1F' |
| **INTEGER\*4** | 1. От одной до десяти десятичных цифр интерпретируется как десятичное число. | 1234567890  0 |
|  | 1. Предшествующие знаки плюс или минус не являются обязательными. | -2147483647 |
|  | 1. Знаки десятичной точки или запятой не допускается, но пробелы разрешены в исходной программе. | - 2 147 483 647 |
|  | 1. Диапазон значений: от -2 147 483 648 до 2 147 483 647 (т.е. -231до 231-1). |  |

## 3.3 Переменные

Переменные обозначаются в операторах FORTRAN символическими именами. Имена это уникальные строки, содержащие от 1 до 6 алфавитно-цифровых символов, в которых первой является буква.

**Примечание**

Имена системных переменных и подпрограмм времени выполнения отличаются от других имен переменных тем, что они начинают с символа знака доллара ($). Поэтому строго рекомендуется во избежание конфликтов, символические имена в исходных программах FORTRAN начинать с любой буквы кроме "$".

**Примеры:**

I5, TBAR, B23, ARRAY, XFM79, MAX, A1$C

Переменные подразделяются на четыре типа: INTEGER, REAL, DOUBLE PRECISION и LOGICAL.Указание типа осуществляется одним из следующих способов:

1. **Неявное объявление типа**, при котором первая буква символического имени определяет целый или вещественный тип. Если тип не объявлен явно (п. 2 ниже), символические имена, начинающиеся с I, J, K, L, M или N, представляют целые переменные и символические имена, начинающиеся с букв кроме, I, J, K, L, M или N представляют вещественные переменные.

**Целые переменные**

ITEM, J1, MODE, K123, N2

**Вещественные переменные**

BETA, H2, ZAP, AMAT, XID

1. Переменным может быть объявлен тип в явном виде. То есть, им может быть задан конкретный тип независимо от первой буквы в их имени. Переменные могут быть явно объявлены как INTEGER, REAL, DOUBLE PRECISION или LOGICAL. Конкретные операторы, используемые для явного определения типа, описаны в [Разделе 6](#_6.1_Операторы_описания).

Переменные получают свои числовые значения во время выполнения программы или, первоначально, в операторе DATA ([Раздел 6](#_6.8_Операторы_инициализации_1)).

Текстовые или литеральные данные могут быть присвоены переменной любого типа. [Подпункт 3.6](#_3.6_Выделение_памяти) содержит описание хранения текстовых данных.

## 3.4 Массивы и элементы массивов

Массив это упорядоченный набор данных, обладающий свойством размерности. Массив может иметь 1, 2 или 3 измерения, идентифицируется и получает тип по символическому имени аналогично переменной, за исключением того, что имена массивов должны быть объявлены "оператором объявления массива".

Подробное описание операторов объявления массива приводится в [Разделе 6](#_6.2_Объявление_массива_1) этого руководства. Оператор объявления массива также указывает размерность и размер массива. Элемент массива - один элемент набора данных, из которых состоит массив.

Ссылка на элемент массива в операторе FORTRAN делается путем добавления индекса к имени массива. Термин элемент массива является синонимом термина индексированная переменная, используемого в некоторых учебниках.

Начальное значение может быть присвоено любому элементу массива оператором DATA, или его значение может быть получено и определено во время выполнения программы.

## 3.5 Индексы

Индекс следует за именем массива, чтобы однозначно указать элемент массива. Индекс в операторе FORTRAN имеет то же значение, что и индекс в привычной алгебраической нотации.

Правила, определяющие использование индексов, следующие:

1. Индекс содержит 1, 2 или 3 индексных выражения в круглых скобках (см. п. 4 ниже).
2. Если есть два или три индексных выражения в скобках, они должны быть разделены запятыми.
3. Количество индексных выражений должно быть таким же, какое указанно при объявлении размерности массива за исключением вхождения имени элемента массива в операторе EQUIVALENCE ([**Раздел 6**](#_Раздел_6_Операторы_1)).
4. Выражение индекса записывается в одной из следующих форм:

K C\*V V-K

V C\*V+K C\*V-K

V+K

где C и K - целые константы, V имя целой переменной (см. [**Раздел 4**](#_Раздел_4_Выражения_1)для уточнения правил вычисления выражений).

1. Сами индексы не могут быть индексами.

**Примеры:**

X(2\*J-3,7) A(I,J,K) I(20) C(L-2) Y(I)

## 3.6 Выделение памяти данным

Для размещения данных FORTRAN выделяет некоторое число ячеек памяти. Ячейка памяти имеет размер, необходимый для размещения одного вещественного значения данных (4 байта).

**Таблица 3-2** определяет формат слова для этих трех типов данных.

Шестнадцатеричные данные могут быть сопоставлены (с помощью оператора DATA) с данными любого типа. Они размещаются в памяти так же, как связанные данные.

Текстовые или литеральные данные могут быть сопоставлены с любым типом данных при помощи операторов инициализации DATA ([Раздел 6](#_6.8_Операторы_инициализации)),

До восьми текстовых символов могут быть сопоставлены с данными DOUBLE PRECISION, до четырех с REAL или INTEGER\*4, до двух с INTEGER\*2, и один с типом LOGICAL.

Таблица 3 - 2 Выделение памяти данным различных типов

| **Тип** | **Выделение памяти** | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **INTEGER** | 2 байта / 1/2 ячейки памяти | | | | | | | |
| Бит знака | | Двоичное значение | | | |  | |
| Отрицательные числа являются дополнением до двух положительных представлений. Порядок хранения обратный. (За младшим значащим байтом следуют старший значащий байт.) | | | | | | | |
| **LOGICAL** | 1 байт / 1/4 ячейки памяти | | | | | | | |
| Ноль | (false) или | | | Не ноль | | (true) | |
| Ненулевое значение означает «истина» (логическая константа .TRUE. представлена шестнадцатеричным значением 0FFh). Нулевое значение байт, указывает «ложь» (false).  Если логические данные используется в качестве арифметического значения, они обрабатывается как целое число в диапазоне от -128 до +127. | | | | | | | |
| **REAL** | 4 байта / 1 ячейка памяти | | | | | | | |
| характеристика | | | бит знака | | мантисса (старш.) | |  |
| мантисса (средн.) | | | мантисса (младш.) | | | |  |
| Первый байт - характеристика выражается в избыточной 200 (восьмеричной) нотации, то есть, значение 200 (восьмеричное) соответствует двоичному показателю степени 0. Значения меньше 200 (восьмеричное) соответствуют отрицательным показателям степени, а значения больше 200 соответствуют положительным показателям степени. По определению, если характеристика равна нулю, то все число равно нулю.  Следующие три байта составляют мантиссу. Мантисса всегда нормализуется таким образом, что высокого уровня бит один, избавляя от необходимости фактически сохранить этот бит. Высокий бит используется вместо этого, чтобы указать знак числа. Единица указывает отрицательное число, и нуль указывает положительное число. Мантисса, как предполагается, является двоичной дробью, двоичная точка которой расположена слева от мантиссы. Формат мантиссы - "величина со знаком". Байты хранятся в обратном порядке: младший разряд мантиссы, далее средний разряд, и характеристика. | | | | | | | |
| **DOUBLE**  **PRECISION** | 8 байт / 2 ячейки памяти | | | | | | | |
| Внутренняя форма данных двойной точности идентична с представлением данных одинарной точности, за исключением того, что представление двойной точности использует 4 дополнительных байта для мантиссы. | | | | | | | |
| **INTEGER\*4** | 4 байта / 1 ячейка памяти | | | | | | | |
| Отрицательные числа представляются в форме дополнения до двух. Байты хранятся в обратном порядке, младшие значащие к старшему значащему. | | | | | | | |

Таблица 3 - 3 Эквивалентные типы данных и размеры

|  |  |
| --- | --- |
| **Типы данных** | **Размер в байтах** |
| BYTE | 1 |
| INTEGER\*1 | 1 |
| LOGICAL | 1 |
| INTEGER | 2 |
| INTEGER\*2 | 2 |
| LOGICAL\*2 | 2 |
| INTEGER\*4 | 4 |
| LOGICAL\*4 | 4 |

# Раздел 4 Выражения FORTRAN

Выражение FORTRAN состоит из единственного операнда или строки операндов, соединенных операторами. Язык FORTRAN обеспечивает два типа выражений - арифметические и логические. Операнды, операторы и правила использования обоих типов выражений описаны в следующих параграфах.

## 4.1 Арифметические выражения

Следующие правила определяют все допустимые формы арифметических выражений:

1. Константа, имя переменной, ссылка на элемент массива, или ссылка на функцию ([Раздел 9](#_Раздел_9_Функции_4)), стоящие в одиночестве являются выражением.

**Примеры:**

S(I) JOBNO 217 17.26 SQRT(A+B)

1. Если E - выражение, первый символ которого не оператор, то +E и -E называют выражениями со знаком.

**Примеры:**

S +JOBNO -217 +17.26 -SQRT(A+B)

1. Если E является выражением, то (E) означает значение вычисленного выражения E.

**Примеры:**

(-A) -(JOBNO)-(X+1) (A-SQRT(A+B))

1. Если Е выражение без знака и F это любое выражение, то: F+E, F-E, F\*E, F/E и F\*\*E являются выражениями.

**Примеры:**

-(B(I,J)+SQRT(A+B(K,L)))

1.7E-2\*\*(X+5.0)

-(B(I+3,3\*J+5)+A)

1. Вычисленное выражение может быть целым, расширенным целым, вещественным, двойной точности, или логическим. Тип определяется типами данных элементов выражения. Если элементы выражения имеют разные типы, тип выражения определяется элементом, имеющим наивысший тип.

Иерархия типов (от высшего к низшему) выглядит следующим образом:

1. DOUBLE PRECISION,

2. REAL, INTEGER\*4,

3. INTEGER,

4. LOGICAL.

1. Выражения могут содержать вложенные в скобках элементы как в следующем примере:

A\*(Z-( (Y+X)/T) )\*\*J

где Y+X это самый вложенный элемент, (Y+X)/T следующий вложенный,   
Z-(Y+X)/T), следующий. В таких выражениях, следует позаботиться, чтобы число левых скобок было равно числу правых скобок.

## 4.2 Вычисление выражений

Арифметические выражения вычисляются в соответствии со следующими правилами:

1. Элементы выражения, заключенные в круглые скобки вычисляются в первую очередь. Если элементы, заключенные в скобки, являются вложенными, первым вычисляется самый вложенный элемент, затем следующей вложенный до тех пор пока все выражения не будут вычислены.
2. В круглых скобках и/или везде, где круглые скобки не влияют на порядок вычислений, иерархия операций в порядке приоритета следующая:

1. Вычисление функции,

2. Возведение в степень,

3. Умножение и деление,

4. Сложение и вычитание.

**Пример:**

выражение

A\*(Z-((Y+R)/T))\*\*J+VAL

вычисляется в следующем порядке:

e1 = Y+R

e2 = (e1)/T

e3 = Z-e2

e4 = e3\*\*J

e5 = A\*e4

e6 = e5+VAL

1. Выражение X\*\*Y\*\*Z не допускается. Оно должно быть записано в следующем виде:

(X\*\*Y)\*\*Z или X\*\*(Y\*\*Z)

1. Использование ссылки на элемент массива требует вычисления его индекса. Индексные выражения вычисляются по тем же правилам, что и другие выражения.

## 4.3 Логические выражения

Логическим выражением является:

1. Единственная логическая константа (т.е. .TRUE. или .FALSE.), логическая переменная, логический элемент массива или ссылка на логическую функцию (см. [Раздел 9 Функции](#_Раздел_9_Функции_3)).
2. Два арифметических выражения, разделенные оператором отношения (т.е., реляционное выражение).
3. Логические операторы, реагирующие на логические константы, логические переменные, логические элементы массива, логические функции, реляционные выражения или другие логические выражения.

Значение логического выражения всегда либо .TRUE. либо.FALSE.

### 4.3.1 Реляционные выражения

Общий вид реляционного выражения выглядит следующим образом:

e1 r e2

где e1 и e2 - арифметические выражения, и r - оператор отношения. Имеются следующие шесть операторов отношения:

.LT. Меньше (<)

.LE. Меньше или равно (≤)

.EQ. Равно (=)

.NE. Не равно (≠)

.GT. Больше (>)

.GE. Больше или равно (≥)

Значение реляционного выражения .TRUE., если условие, определенное оператором, соблюдается. Иначе, значение - .FALSE.

Примеры:

A.EQ.B

(A\*\*J).GT.(ZAP\*(RHO\*TAU-ALPH))

### 4.3.2 Логические операторы

**Таблица 4-1** перечисляет логические операции. U и V обозначают логические выражения.

Таблица 4 - 1 Логические операции

|  |  |
| --- | --- |
| .NOT.U | Значение этого выражения - логическое отрицание (дополнение) U (т.е., бит равный 1 становится 0, и бит равный 0 становятся 1). |
| U.AND.V | Значение этого выражения является логическим произведением U и V (то есть, есть бит будет равен 1 только если соответствующие биты в U и V равны 1. |
| U.OR.V | Значение этого выражения логическая сумма U и V (т.е., есть, бит будет равен 1, если соответствующий бит в U или V равняется 1, или если соответствующие биты и в U и в V равняются 1. |
| U.XOR.V | Значение этого выражения - логическое исключающее ИЛИ U и V (т.е., есть 1 в результате, если соответствующие биты в U и V равняются 1 и 0 или 0 и 1 соответственно. |

**Примеры:**

Если U = 01101100 и V = 11001001, то

.NOT.U = 10010011

U.AND.V = 01001000

U .OR.V = 11101101

U.XOR.V = 10100101

Ниже приведены дополнительные сведения по построению логических выражений:

1. Любое логическое выражение может быть заключено в скобки. Тем не менее, логическое выражение, к которому применяется оператор .NOT., должно быть заключено в скобки, если оно содержит два или более элемента.
2. В иерархии операций круглые скобки могут использоваться, чтобы определить порядок вычисления выражения. В круглых скобках, и везде где круглые скобки не влияют на порядок вычислений, порядок вычисления понимают следующим образом:

1. Ссылка на функцию

2. Возведение в степень (\*\*)

3. Умножение и деление (\* и /)

4. Сложение и вычитание (+ и -)

5. .LT., .LE., .EQ., .NE., .GT., .GE.

6. .NOT.

7 . .AND.

8. .OR., .XOR.

**Примеры:**

Выражение

X .AND. Y .OR. B(3,2) .GT. Z

вычисляется как

e1 = B(3,2).GT.Z

e2 = X .AND. Y

e3 = e2 .OR. e1

Выражение

X .AND. (Y .OR. B(3,2) .GT. Z)

вычисляется как

e1 = B(3,2) .GT. Z

e2 = Y .OR. e1

e3 = X .AND. e2

1. Недопустимо иметь два смежных логических оператора, за исключением, случая, когда второй оператор .NOT. То есть,

.AND..NOT.и.OR..NOT. допустимо.

**Пример:**

A.AND..NOT.B допустимо

A.AND..OR.B недопустимо.

## 4.4 Текстовые, литеральные и шестнадцатеричные константы в выражениях

Текстовые, литеральные и шестнадцатеричные константы разрешены в выражениях вместо целых констант. Эти специальные константы всегда вычисляются к целому значению и поэтому ограничены длиной два байта. Единственным исключением из этого правила являются:

1. Длинные текстовые или литеральные константы могут быть использованы в качестве параметров подпрограммы.
2. Текстовые, литеральные и шестнадцатеричные константы могут иметь длину четыре байта в операторах DATA, когда связаны с вещественными переменными или восемь байтов, когда связаны с переменными двойной точности.

# Раздел 5 Операторы присваивания

Операторы присваивания определяют вычисления и подобным образом используются для уравнений в обычной математической нотации. Они имеют следующую форму:

**v = e**

где **v** - значение любой переменной или элемента массива и **e** - выражение.

Семантика FORTRAN определяет знак равенства (=) в смысле замены, а не обычной эквивалентности. Таким образом, инструкции объектной программы, порожденные оператором присваивания, будут при выполнении вычислять выражение справа от знака равенства и помещать результат в выделенное место в памяти для переменной или элемента массива, расположенного слева от знака равенства.

Следующие условия применяются к операторам присваивания:

1. Переменная v и знак равенства должны располагаться в одной строке. Это справедливо даже тогда, когда оператор является частью логического оператора IF ([Раздел 7](#_РАЗДЕЛ_7_Операторы)).

Пример:

C

C ОПЕРАТОР ПРИСВАИВАНИЯ '=' ДОЛЖЕН БЫТЬ В НАЧАЛЬНОЙ СТРОКЕ.

C

A(5,3) =

1 B(7,2) + SIN(C)

Строка, содержащая v= должна быть начальной строкой оператора, если оператор не является частью логического оператора IF. В этом случае v= должно находиться не позднее конца первой строки после окончания IF.

1. Если тип переменной v и выражения e будут отличаться, то значение, определенное выражением, будет преобразовано, если это возможно, чтобы соответствовать типу переменной. **Таблица 5-1** показывает, какие типы выражений могут быть преобразованы к типам переменных. Y указывает допустимое преобразование, и N указывает недопустимое преобразование. Сноски к Y указывают примечания к соответствующим преобразованиям.

Таблица 5 - 1 Преобразование типа

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Тип переменной | Типы выражений | | | | |
| Integer | Real | Logical | Double | Ext Int |
| Integer | Y | Ya | Yb | Ya | Yg |
| Real | Yc | Y | Yc | Ye | Yc |
| Logical | Yd | Ya | Y | Ya | Yd |
| Double | Yc | Y | Yc | Y | Yc |
| Ext Int | Yf | Yh | Yb,f | Yh | Y |

1. Вещественное значение выражения преобразуется в целое, усеченное при необходимости, чтобы соответствовать размеру данных целого типа.
2. Знак продлен до конца второго байта.
3. Переменной присваивается вещественное приближение значения целого выражения.
4. Переменной присваивается усеченное значение целого выражения (младший байт используется, независимо от знака).
5. Переменной присваивается округленное значение вещественного выражения.
6. Знак распространяется через третий и четвертый байт.
7. Переменной присваивается усеченное значение расширенного целого выражения.
8. Значение выражения преобразуется в расширенное целое и усекается для соответствия размера данных расширенного целого.

# Раздел 6 Операторы описания

Операторы описания - невыполнимые, не порождающие код операторы, которые определяют тип переменных и массивов, определяют размерность массива и размер, выделенный для хранения данных, или другую важную информацию для компилятора FORTRAN. Оператор инициализации DATA - невыполнимая программа, но генерирующая данные в объектную программу и устанавливающая начальные значения для переменных.

## 6.1 Операторы описания

Существует семь видов операторов описания:

Оператор IMPLICIT

Операторы определения типа, EXTERNAL и DIMENSION

Оператор COMMON

Оператор EQUIVALENCE

Оператор инициализации DATA

Все операторы описания должны быть сгруппированы в начале модуля программы и упорядочены, поскольку они появляются ранее. Операторам описания могут предшествовать только операторы FUNCTION, SUBROUTINE, PROGRAM или BLOCK DATA. Все операторы описания должны предшествовать операторам функций и первому выполняемому оператору.

## 6.2 Объявление массива

Объявление массива могут осуществлять три вида операторов описания. Эти операторы - следующее:

Операторы определения типа

DIMENSION

COMMON

Из них оператор DIMENSION выполняет единственную функцию - декларацию массивов. Два других выполняют двойные функции. Эти операторы определены в подпунктах [6.3](#_6.3_Операторы_определения), [6.5](#_6.5_Операторы_DIMENSION) и [6.6](#_6.6_Операторы_COMMON).

Объявления массива используются, чтобы определить имя, размерность и размеры массивов. В модуле программы массив может быть объявлен только один раз.

Объявление массива имеет одну из следующих форм:

ui (k)

ui (k1,k2)

ui (k1,k2,k3)

где ui - имя массива, присвоенное оператором объявления, и ki - целые константы.

Выделение места для размещения массива происходит после появления объявления массива. Элементы массива упорядочены по столбцам, самый левый индекс изменяется наиболее быстро, т.к. адреса в памяти по этому индексу расположены рядом.

Например, оператор объявления массива AMAT(3,2,2) выделяет место для этих 12 элементов в следующем порядке:

AMAT(1,1,1), AMAT(2,1,1), AMAT(3,1,1), AMAT(1,2,1),

AMAT(2,2,1), AMAT(3,2,1), AMAT(1,1,2), AMAT(2,1,2),

AMAT(3,1,2), AMAT(1,2,2), AMAT(2,2,2), AMAT(3,2,2)

## 6.3 Операторы объявления типа

Имена переменных, массивов и функций автоматически имеют целый или вещественный тип на основании «предопределенного» соглашения, если они не изменены оператором объявления типа. Например, имеют целый тип, если первая буква элемента является I, J, K, L, M или N. В противном случае - вещественный тип.

Операторы объявления типа предусматривают переопределение или подтверждение предопределенного соглашения, определяя тип элемента. Кроме того, эти операторы могут использоваться для объявления массивов.

У операторов объявления типа есть следующая общая форма:

**t v1,v2,...vn**

где t представляет собой один из терминов: INTEGER, INTEGER\*1, INTEGER\*2, INTEGER\*4, REAL, REAL\*4, REAL\*8, DOUBLE PRECISION, LOGICAL, LOGICAL\*1, LOGICAL\*2, LOGICAL\*4 или BYTE. Каждый vi - оператор объявления массива, переменная, массив или имя функции. Типы INTEGER\*1, INTEGER\*2, INTEGER\*4, REAL\*4, REAL\*8, LOGICAL\*1, LOGICAL\*2, LOGICAL\*4 используются для удобочитаемости и совместимости с другими компиляторами FORTRAN.

**Пример:**

REAL AMAT(3,3,5), BX, IETA, KLPH

Примечание

1. Для AMAT и BX тип объявлен избыточно.

2. Переменные IETA и KLPH безоговорочно объявлены вещественными.

3. AMAT(3,3,5) объявляет размер массива из 45 элементов.

**Пример:**

INTEGER M1, HT, JMP(15), FL

Примечание

Здесь для M1 тип объявлен избыточно. Для HT и FL тип изменяется с предопределенного по умолчанию до их появления в операторе INTEGER. JMP(15) объявляет размер массива. Тип элементам массива объявлен избыточно, как целый и сообщает компилятору требования для выделения памяти и размерности массива.

**Пример:**

LOGICAL LI, TEMP

**Примечание**

Все переменные, массивы или функции, которые необходимо объявить логическими должны присутствовать в операторе LOGICAL, поскольку не предусмотрены начальные буквы для определения этого типа в соответствии с соглашением по умолчанию.

## 6.4 Операторы EXTERNAL

Оператор EXTERNAL имеет следующую форму:

**EXTERNAL u1,u2,...,un**

где каждый ui - имя SUBROUTINE, BLOCK DATA или FUNCTION. Если имя подпрограммы используется в качестве аргумента вызова подпрограммы, она должна присутствовать в предшествующем операторе EXTERNAL.

Если подпрограмма BLOCK DATA должна быть включена в собранную программу, ее имя, должно быть, объявлено в операторе EXTERNAL в основном модуле программы.

Например, если SUM и AFUNC - имена подпрограмм, которые будут использоваться в качестве параметров в подпрограмме SUBR, следующие операторы должны присутствовать в модуле программы вызова:

|  |  |
| --- | --- |
| В основной программе: | В подпрограмме: |
| EXTERNAL SUM, AFUNC  .  .  CALL SUBR(SUM,AFUNC,Z,P)  . | SUBROUTINE SUBR(S,F,A,B,C)  D = S(A)  E = F(B)  C = D+E  RETURN  END |

## 6.5 Операторы DIMENSION

Оператор DIMENSION имеет следующий формат:

**DIMENSION u1,u2,u3,...,un**

где каждый ui - объявление массива.

Пример:

DIMENSION RAT(5,5), BAR(20)

Этот оператор объявляет 25 элементов в массиве RAT и 20 элементов в массиве BAR.

## 6.6 Операторы COMMON

Оператор COMMON - невыполняемый. Оператор выделяет место для переменных и массивов в области памяти под названием COMMON и предоставляет возможность различным программным модулям совместно использовать эту область памяти.

Оператор COMMON имеет следующий формат:

**COMMON /y1/a1 /y2/a2/.../yn/an**

где каждый yi это название COMMON блока и каждый ai представляет собой последовательность имен переменных, имен массивов или объявлений массивов, разделенных запятыми. Элементы ai составляют общую область с именем yi. Если какой-либо yi опущен с оставлением двух последовательных символов наклонной черты (//), то обозначенный таким способом блок называют неименованным COMMON блоком. Если первое имя блока (yi) опущено, то могут быть опущены и две наклонных черты.

**Пример:**

COMMON /AREA/A,B,C/BDATA/X,Y,Z,

X FL,ZAP(30)

В этом примере два COMMON блока выделяют ячейки - BDATA для трех переменных и ZAP для четырех переменных и 30 элементов массива.

**Пример:**

COMMON //A1,B1/CDATA/ZOT(3,3)

X //T2,Z3

В этом примере переменные A1, B1, T2 и Z3 присвоены неименованному COMMON блоку именно в таком порядке. Пару наклонных черт, предшествующих A1, можно было опустить.

COMMON блок с именем CDATA содержит девять элементов массива, таким образом, ZOT(3,3) - это объявление массива. Массив ZOT не должен быть объявлен ранее. (См. [Пункт 6.2 Объявление массива](#_6.2_Объявление_массива).)

**Дополнительные сведения:**

1. Имя COMMON блока может появляться в одном или нескольких операторах COMMON более одного раза.
2. Имя COMMON блока содержит от 1 до 6 буквенно-цифровых знаков, первый из которых должен быть буквой.
3. Имя COMMON блока должно отличаться от любых имен подпрограмм, используемых на протяжении всей программы.
4. Размер общей области может быть увеличен за счет использования операторов EQUIVALENCE. См. [Пункт 6.7 Операторы EQUIVALENCE](#_6.7_Операторы_EQUIVALENCE).
5. COMMON блоки c одинаковым именем не обязаны иметь одинаковую длину во всех программных модулях в которых они присутствуют. Тем не менее, если длины отличаются, программа, содержащая блок с наибольшей длиной должна быть загружена первой (см. LINK-80 Руководство пользователя). Длина общей области это количество ячеек памяти, необходимых для размещения переменных и массивов, объявленных в операторе (или операторах) COMMON, если она не расширена за счет использования оператора EQUIVALENCE.

## 6.7 Операторы EQUIVALENCE

Использование операторов EQUIVALENCE разрешает совместное использование ячеек памяти двумя или больше объектами. Общая форма оператора следующая:

**EQUIVALENCE (u1),(u2),...,(un)**

где каждый ui представляет собой последовательность из двух или более переменных или элементов массива, разделенных запятыми. Каждому элементу в последовательности компилятором назначаются одна и та же ячейка памяти (или ее часть). Порядок, в котором расположены элементы, не является существенным.

**Пример:**

EQUIVALENCE (A,B,C)

Переменные A, B и C совместно используют одну и ту же ячейку памяти во время выполнения объектной программы.

Если элемент массива используется в операторе EQUIVALENCE, число индексов должно соответствовать размерности, установленной в объявлении массива, или он может быть один, указывая номер соответствующего элемента массива относительно первого элемента массива.

**Пример:**

Если размерность массива Z была объявлена Z(3,3), то в операторе EQUIVALENCE Z(6) и Z(3,2) имеют одинаковый смысл.

**Дополнительные сведения:**

1. Индексы элементов массива должны быть целыми константами.
2. Элемент многомерного массива, при желании, может быть отображен одним индексом.
3. Переменные могут быть отнесены к COMMON блоку с помощью операторов EQUIVALENCE.

**Пример:**

COMMON /X/A,B,C

EQUIVALENCE (A,D)

В этом случае переменные A и D разделяют первую ячейку памяти в COMMON блоке X.

1. Операторы EQUIVALENCE могут увеличить размер блока, обозначенный оператором COMMON, добавив несколько элементов в конец блока.

**Пример:**

DIMENSION R(2,2)

COMMON /Z/W,X,Y

EQUIVALENCE (Y,R(3))

В результате COMMON блок будет иметь следующую конфигурацию:

Распределение памяти переменных

W = R(1,1) 0

X = R(2,1) 1

Y = R(1,2) 2

R(2,2) 3

Общий блок созданный оператором COMMON содержит 3 ячейки памяти. С помощью оператора EQUIVALENCE она расширяется до 4 ячеек памяти.

Размер общего блока созданного оператором COMMON может быть увеличен только вперед, начиная с последнего элемента. Не от первого его элемента назад.

Обратите внимание на то, что объявление EQUIVALENCE (X,R(3)) было бы в примере недопустимым. Оператор COMMON установил W в качестве первого элемента в общем блоке и попытка сделать X и R(3) эквивалентными будет попыткой сделать R(1) первым элементом.

1. Недопустима эквивалентность двух элементов одного и того же массива или двух элементов, принадлежащих тому же или разным COMMON блокам.

**Пример:**

DIMENSION XTABLE (20), D(5)

COMMON A,B(4)/ZAP/C,X

.

.

EQUIVALENCE (XTABLE(6),A(7),

X B(3) ,XTABLE(15) ) ,

Y (B(3),D(5))

.

.

Этот оператор EQUIVALENCE содержит следующие ошибки:

1. Он пытается объявить эквивалентными два элемента одного и того же массива XTABLE(6) и XTABLE(15).
2. Он пытается объявить эквивалентными два элемента одного и того же COMMON блока A(7) и В(3).
3. Переменная A не является массивом, (7) является недопустимой ссылкой.
4. Объявление B(3) эквивалентным D(5) расширяет COMMON назад от ее определенной начальной точки.

## 6.8 Операторы инициализации данных

Оператор инициализации данных DATA - невыполняемый оператор, который обеспечивает способ компиляции значений данных в объектную программу и присвоение этих данных переменным и элементам массива, на которые ссылаются другие операторы.

Оператор имеет следующий вид:

**DATA list/u1,u2,...,un/,list.../uk,uk+1,...uk+n/**

где "list" представляет собой список имен переменных, массивов или элементов массива, и ui является константами, количество которых равно числу элементов в списке.

Исключением из правила взаимного соответствия элементов списка является то, что имя массива может присутствовать в списке без индексов и между косыми чертами может появиться в нужном порядке требуемое количество констант необходимых для заполнения массива.

Вместо ui, допустимо написать k\*ui для того, чтобы объявить ту же константу ui k раз подряд. k должно быть положительным целым числом. Фиктивные параметры не могут появиться в списке.

**Пример:**

DIMENSION C(7)

DATA A, B, C(1) ,C(3)/14.73,

X -8.1,2\*7.5/

Это означает, что

A=14.73, B=-8.1, C(1)=7.5, C(3)=7.5

Тип каждой константы ui должен совпадать с типом соответствующего элемента в списке, за исключением текстовых или литеральных констант, которые могут быть в паре с элементом любого типа.

При использовании текстовых или литеральных констант, количество символов в строке должно не более чем в четыре раза превышать число ячеек памяти, необходимых для размещения соответствующего типа, то есть символ для логической переменной, до 2 символов для целой переменной и 4 или меньше символов для вещественной переменной.

Если текстовых или литеральных символов задано меньше, добавляются конечные пробелы, чтобы заполнить оставшуюся часть памяти. Шестнадцатеричные данные хранятся аналогичным образом. Если шестнадцатеричных символов задано меньше, добавляются необходимые ведущие нули, чтобы заполнить оставшуюся часть ячейки памяти.

Примеры, приведенные ниже, иллюстрируют многие функции оператора DATA.

DIMENSION HARY (2)

DATA HARY,B/ 4HTHIS, 4H OK.

1 ,7.86/

REAL LIT(2)

LOGICAL LT,LF

DIMENSION H4(2,2),PI3(3)

DATA Al,Bl,Kl,LT,LF,H4(l,l),H4(2,1),

1 H4(l,2),H4(2,2), PI3/5.9,2.5E-4,

2 64,.FALSE.,.TRUE.,1.75E-3,

3 0.85E-1,2\*75.0,1.,2.,3.14159/,

4 LIT(l)/'NOGO'/

## 6.9 Оператор IMPLICIT

Оператор IMPLICIT используется для переопределения типов переменных по умолчанию. Синтаксис:

**IMPLICIT type(range),type(range),...**

где type может быть одним из следующих типов: INTEGER, REAL, LOGICAL, DOUBLE PRECISION, BYTE, INTEGER\*1, INTEGER\*2, INTEGER\*4, REAL\*4, REAL\*8

и range (диапазон) - список буквенных символов, разделенных запятыми или тире.

**Примеры:**

IMPLICIT INTEGER(A,W-Z),REAL(B-V)

Все переменные (не иначе объявленные) начинающиеся с букв A, W, X, Y, Z будут иметь целый тип. Все переменные, начинающиеся с букв от B до V, будут иметь вещественный тип.

IMPLICIT INTEGER(I-N),REAL(A-H,O-Z)

Это - определение по умолчанию.

Любые операторы IMPLICIT должны располагаться в программе совместно с операторами описания типа и DIMENSION. Операторы IMPLICIT должны располагаться до любых других операторов описания. Если оператор IMPLICIT появится после операторов описания типа или DIMENSION, то уже объявленные типы переменных не будут затронуты.

# РАЗДЕЛ 7 Операторы управления FORTRAN

Операторы управления FORTRAN являются выполняемыми операторами, которые изменяют и направляют логический поток программы FORTRAN. Операторы этой категории следующие:

1. Операторы GO TO:

Безусловный GO TO

Вычисляемый GO TO

Присваиваемый GO TO

1. ASSIGN;
2. Операторы IF:

Арифметический IF;

Логический IF;

1. DO;
2. CONTINUE;
3. STOP;
4. PAUSE;
5. CALL;
6. RETURN;
7. END.

Метки, являющиеся частью оператора управления, должны быть связаны с выполняемыми операторами в том же модуле программы, в котором появляется оператор управления.

## 7.1 Операторы GOTO

### 7.1.1 Безусловный GOTO

Операторы безусловного перехода GOTO используются каждый раз, когда управление безоговорочно должно быть передано некоторому другому оператору в модуле программы.

Оператор имеет следующую форму:

**GO TO k**

где k - метка выполняемого оператора в том же модуле программы.

**Пример:**

GO TO 376

310 A(7)=VI-A(3)

.

.

376 A(2)=VECT

GO TO 310

В этой части программы, оператор с меткой 376 выполнится раньше оператора с меткой 310.

### 7.1.2 Вычисляемый GOTO

Оператор вычисляемого GOTO имеет вид:

**GO TO (k1,k2,..., kn),j**

где ki являются меткой оператора, и j является целой переменной, 1 ≤ j ≤n.

Этот оператор вызывает передачу управления на оператор помеченный kj. Если j ≤ 1 или j > n управление будет передано на оператор следующий, за оператором вычисляемого GOTO.

**Пример:**

J=3

.

.

GO TO(7, 70, 700, 7000, 70000), J

310 J=5

GO TO 325

При J = 3, вычисляемый GOTO передаст управление оператору с меткой 700. Изменение J равным 5 вызовет переход к оператору с меткой 70000. Задание J = 0 или J = 6 вызовет передачу управления оператору с меткой 310.

### 7.1.3 Присваиваемый GOTO

Присваиваемый GOTO имеют следующую форму:

**GO TO j, (k1,k2,...,kn)**

или

**GO TO j**

где j является именем целой переменной, а ki меткой выполняемого оператора. Этот оператор вызывает передачу управления оператору, метка которого равна текущему значению переменной j.

**Уточнения**

1. Оператор ASSIGN должен предшествовать оператору присваиваемого GOTO.
2. Оператор ASSIGN должен присвоить значение j, которое является меткой оператора, включенной в список kn, если список определен.

Пример:

GO TO LABEL,(80,90,100)

Только метки 80, 90 или 100 могут быть присвоены переменной LABEL.

## 7.2 Оператор ASSIGN

Этот оператор имеет следующую форму:

**ASSIGN j TO i**

где j - метка выполняемого оператора, и i - целая переменная.

Оператор используется в сочетании с каждым присваиваемым оператором GOTO, который содержит целую переменную i. Когда присваиваемый GOTO будет выполнен, управление будет передано оператору c меткой j.

**Пример:**

ASSIGN 100 TO LABEL

.

.

ASSIGN 90 TO LABEL

GO TO LABEL, (80,90,100)

## 7.3 Оператор IF

Операторы IF передают управление одной из серии операторов в зависимости от условия. Существуют два типа операторов IF:

**Арифметический IF и логический IF**

### 7.3.1 Арифметический IF

Арифметический оператор IF имеет вид:

**IF(e) m1,m2,m3**

где e - арифметическое выражение и m1, m2 и m3 - метки оператора.

Вычисление выражения e определяет одну из трех возможностей передачи управления:

|  |  |
| --- | --- |
| Если e: | Переход к: |
| < 0 | m1 |
| = 0 | m2 |
| > 0 | m3 |

**Примеры:**

|  |  |  |
| --- | --- | --- |
| Оператор | Значение e | Переход к |
| IF(A)3,4,5 | 15 | 5 |
| IF(N-1)50,73,9 | 0 | 73 |
| IF(AMTX(2,1,2))7,2,1 | -256 | 7 |

### 7.3.2 Логический IF

Логический оператор IF имеет вид:

**IF(u) s**

где u - логическое выражение, и s - любой выполняемый оператор кроме оператора DO (см. [7.4](#_7.4_Оператор_цикла)) или другого логического оператора IF. Логическое выражение u оценено как .TRUE. или .FALSE. [Раздел 4](#_4.3_Логические_выражения) содержит описание логических выражений.

**Условия управления:**

Если u - FALSE, то оператор s игнорируется, и управление переходит к следующему оператору после логического оператора IF. Однако, если выражение - TRUE, то управление, переходит к оператору s, и последующее выполнение программы происходит обычным образом.

Если s – оператор присваивания (v = e, [Раздел 5](#_Раздел_5_Операторы_1)), переменная и знак равенства (=) должны быть в той же строке, или сразу после следующего оператора IF(u) или на отдельной строке продолжения с пустой строкой следующей после оператора IF(u). Смотрите пример 5 ниже.

**Примеры:**

1. IF(I.GT.20) GO TO 115

2. IF(Q.AND.R) ASSIGN 10 TO J

3. IF(Z) CALL DECL(A,B,C)

4. IF(A.OR.B.LE.PI/2) I=J

5. IF(A.OR.B.LE.PI/2)

# I =J

## 7.4 Оператор цикла DO

Оператор DO, реализованный в FORTRAN, обеспечивает способ повторного выполнения ряда операторов. Оператор может иметь одну из двух следующих форм:

1) **DO k i = m1,m2,m3**

или

2) **DO k i = m1,m2**

где k - метка оператора, i - целая или логическая переменная, и m1, m2 и m3 являются целыми константами или целыми или логическими переменными.

Если величина шага m3 равняется 1, она может быть опущена как в форме 2 выше.

Следующие условия и ограничения определяют использование операторов DO:

1. DO и первая запятая должны быть в начальной строке.
2. Оператор с меткой k, называемый последним оператором цикла, должен быть выполняемым оператором.
3. Последний оператор цикла должен физически следовать за связанным с ним оператором DO, и исполняемые операторы, следующие за оператором DO, включая последний оператор цикла, составляют тело цикла DO.
4. Конечным оператором цикла не может быть арифметический оператор IF, операторы GOTO, RETURN, STOP, PAUSE или другой оператор DO.
5. Если конечным оператором цикла является оператор логического IF и его выражение - .FALSE., то выполнение операторов тела цикла DO повторяется.

Если выражение .TRUE., то выполняется оператор логического IF, а затем повторяется выполнение операторов тела цикла DO. Оператор логического IF не может быть оператором GOTO, арифметическим оператором IF, операторами RETURN, STOP или PAUSE.

1. Управляющая целая переменная i, называется индексом тела цикла DO. Индекс должен быть положительным и не может быть изменен любым оператором в теле цикла.
2. Если m1, m2 и m3 являются переменными Integer\*l или константами, цикл DO будет выполняться быстрее и будет короче, но диапазон будет ограничен 127 итерациями. Например, накладные расходы для цикла DO с верхней границей равной 127 и приращением цикла 1 зависят от типа индексной переменной следующим образом:

|  |  |  |
| --- | --- | --- |
| Тип переменной индекса | Издержки  микросекунд | Байт |
| INTEGER\*2 | 35.5 | 19 |
| INTEGER\*1 | 24 | 14 |

1. Во время первого выполнения тела оператора цикла DO, i равно m1, второго выполнения i=m1+m3, третьего, i=m1+2\*m3 и т.д., пока i равное наибольшему значению в этой последовательности меньше или равно m2, и затем цикл DO, говорят, удовлетворен. Операторы тела цикла DO будут выполнены всегда хотя бы один раз, даже если m1≤m2. Когда цикл DO удовлетворен, управление передается оператору, следующему за последним оператором цикла, в противном случае управление возвращается к первому выполняемому оператору после оператора DO.

**Пример:**

Следующий пример вычисляет

100 DIMENSION A(100)

.

.

.

SUM = A(1)

DO 31 I =2,100

31 SUM =SUM + A(I)

.

.

.

END

1. Тело оператора DO может быть расширено, чтобы включить все операторы, которые могут логически быть выполнены между DO и его последним оператором. Таким образом, часть операторов тела цикла DO могут быть расположены таким образом, что они физически не находятся между оператором DO и его последним оператором, но логически выполняются в диапазоне DO. Это называется расширением тела цикла.

**Пример:**

DIMENSION A(500), B(500)

. . .

DO 50 I = 10, 327, 3

IF (V7-C\*C) 20,15,31

.

30 .

50 A(I) = B(I) + C

. . .

20 C = C - .05

GO TO 50

31 C = C + .0125

GO TO 30

1. Не допускается передача управление внутрь тела цикла DO за исключением случая, когда передача управления происходит из расширения тела того же самого оператора DO.
2. В теле оператора DO, могут быть и другие операторы DO, и в этом случае циклы DO должны быть вложены. То есть, если тело одного DO содержит другой DO, то тело внутреннего DO должно быть полностью включено в тело внешнего DO. Последний оператор внутреннего цикла DO может быть также последним оператором внешнего цикла DO.

Например, если A двумерный массив из 15 строк и 15 столбцов и B одномерный из 15 элементов, следующие операторы вычисляют 15 элементов массива C по формуле:

DIMENSION A(15,15), B(15), C(15)

.

.

DO 80 K =1,15

C(K) = 0.0

DO 80 J=l,15

80 C(K) = C(K)+A(K,J)\*B(J)

.

.

## 7.5 Оператор CONTINUE

CONTINUE классифицируется как выполняемый оператор. Тем не менее, его исполнение ничего не делает. Форма оператора CONTINUE выглядит следующим образом:

**CONTINUE**

CONTINUE часто используется в качестве последнего оператора в теле цикла DO, в случае если последним оператором должен быть один из недопустимых операторов или оператор выполняющийся только условно.

**Пример:**

DO 5 K = 1,10

IF (C2) 5,6,6

6 CONTINUE

C2 = C2 +.005

5 CONTINUE

## 7.6 Оператор STOP

Оператор STOP имеет одну из следующих форм:

**STOP**

или

**STOP c**

где c любая строка, содержащая от одного до шести символов.

Если во время выполнения программы встречается оператор STOP, символы с (если имеются) отображаются на консоли оператора и выполнение программы завершается.

Оператор STOP, таким образом, представляет собой логическое завершение программы.

## 7.7 Оператор PAUSE

Оператор PAUSE имеет одну из следующих форм:

**PAUSE**

или

**PAUSE c**

где c любая строка, содержащая до шести символов.

Когда оператор PAUSE встречается во время выполнения рабочей программы, символы строки с (если имеются) отображаются на консоли оператора и выполнение программы прекращается.

Решение продолжить выполнение программы не находится под контролем программы. Если выполнение будет возобновлено посредством вмешательства оператора без иного изменения состояния процессора, нормальное выполнение программы, после паузы будет продолжено.

Выполнение может быть прекращено путем ввода "T" на консоли оператора. Ввод любого другого символа вызовет продолжение выполнения.

## 7.8 Оператор CALL

Оператор CALL выполняет передачу управления в подпрограммы SUBROUTINE и обеспечивает параметры для использования подпрограммам. Общие формы и подробное описание операторов CALL присутствуют в [Разделе 9, Функции и подпрограммы](#_Раздел_9_Функции_2).

## 7.9 Оператор RETURN

Форма, использование и интерпретация оператора RETURN описаны в [Разделе 9](#_Раздел_9_Функции_1).

## 7.10 Оператор END

Оператор END должен физически быть последним оператором любой программы FORTRAN. Он имеет следующий вид:

**END**

Оператор END - выполняемый оператор и может иметь метку. Он вызывает передачу управления служебной подпрограмме $EX выхода в систему, которая возвращает управление операционной системе.

# РАЗДЕЛ 8 Ввод-Вывод

FORTRAN обеспечивает ряд операторов, которые определяют управление, и условия передачи данных между памятью компьютера и устройствами внешней обработки данных, или массового хранения, такими как магнитная лента, диск, линейный принтер, устройство чтения перфокарт, клавиатуры, и т.д.

Эти операторы сгруппированы следующим образом:

1. Форматные операторы READ и WRITE передают отформатированную информацию между компьютером и устройствами ввода/вывода.
2. Бесформатные операторы READ и WRITE передают бесформатные двоичные данные в форме аналогичной внутренней памяти.
3. Вспомогательные операторы ввода/вывода для позиционирования и разграничения файлов.
4. Операторы ENCODE и DECODE для передачи данных между ячейками памяти.
5. Операторы FORMAT, используются в сочетании с отформатированной передачей записей, чтобы обеспечить преобразование данных и информацию редактирования между внутренним представлением данных и внешними формами символьных строк.

## 8.1 Форматные операторы READ/WRITE

### 8.1.1 Форматный оператор READ

Форматный оператор READ используется для передачи информации с устройства ввода данных на компьютер.

Существуют две формы оператора:

**READ (u, f, ERR=L1, END=L2) k**

или

**READ (u, f, ERR=L1, END=L2)**

где:

|  |  |  |
| --- | --- | --- |
| **u** | - | определяет физический и логический номер устройства |
|  | | и может быть или целым без знака или целой переменной в диапазоне 1 - 10. Если используется целая переменная, ей должно быть присвоено целое значение. |
|  | | Устройства 1, 3, 4, и 5 предварительно назначены консоли телетайпа. Устройство 2 предварительно назначено линейному принтеру (если таковой существует). Устройства 6-10 предварительно назначены файлам на диске (см. [Руководство пользователя, Раздел 3](#_Раздел_3_FORTRAN-80_4)). Эти устройства, а также устройства 11-255, могут быть повторно назначены пользователем (см. [Приложение B](#_Приложение_B_Интерфейс_2)). |
| **f** | - | метка оператора FORMAT, описывающего тип преобразования данных, которые будут использоваться во входной передаче, или имени массива, если информация о форматировании может быть введена в программу во время выполнения. (См. [Раздел 8.7.10](#_8.7.10_Спецификации_формата)). |
| **L1** | - | метка оператора FORTRAN которому процессор ввода/вывода передаст управление, если произойдет ошибка ввода/вывода. |
| **L2** | - | метка FORTRAN которому процессор ввода/вывода передаст управление, если будет обнаружен конец файла. |
| **k** | - | список имен переменных, разделенный запятыми, определяющий входные данные. |

Оператор READ(u,f)k используется для ввода переменных в количестве соответствующем списку имен k, из файла с логическим устройством u, используя оператор f FORMAT, для указания внешнего представления этих переменных (см. [Оператор FORMAT, 8.7](#_8.7_Операторы_FORMAT_2)).

Параметры ERR= и END= являются необязательными. Если они не определенны, то возникновение ошибок ввода/вывода и достижение конца файлов приведет к возникновению неустранимых ошибок во время выполнения.

Следующие примечания определяют функции оператора READ(u,f)k:

1. Каждый раз, когда начинается выполнение оператора READ, новая запись считывается из входного файла.
2. Число записей, которые будут введены одним оператором READ, определяются списком k и спецификациями формата.
3. Список k определяет число элементов, которые будут считаны из входного файла и переменные, в которые они должны быть сохранены.
4. Любое количество элементов может появиться в одном списке, и они могут иметь разные типы данных.
5. Если число входных записей превышает число элементов в списке, передаются только записи в количестве, равном по числу элементов в списке. Оставшиеся записи игнорируются.
6. Подробно спецификации списка k описаны в [п. 8.6](#_8.6_Спецификации_списка_1).

**Примеры:**

* Предположим, что четыре целые числа хранятся в последовательном файле на диске и, что значения имеют ширину поля 3, 4, 2 и 5 соответственно. Операторы

READ(5,20) K,L,M,N

20 FORMAT(I3,I4,I2,I5)

прочитают файл и присвоят входные данные переменным K, L, M и N. См. подробное описания операторов FORMAT [п. 8.7](#_8.7_Операторы_FORMAT_1).

* Ввод величин в массив (ARRY):

READ(6,21) ARRY

Только имя массива необходимо в списке (см. [п. 8.6](#_8.6_Спецификации_списка)). Все элементы массива ARRY будут считаны и сохранены с использованием соответствующего форматирования, определенного оператором FORMAT с меткой 21.

READ(u,k) может использоваться в сочетании с оператором FORMAT, чтобы считать буквенно-цифровые данные H-типа в существующее поле H-типа (см. [Преобразования текста, 8.7.3](#_8.7.3_Преобразования_текста)).

Например, операторы

READ(I,25)

.

.

25 FORMAT(10HABCDEFGHIJ)

считывают следующие 10 символов файла на устройстве ввода данных I и заменяют символы ABCDEFGHIJ в операторе FORMAT.

### 8.1.2 Форматный оператор WRITE

Форматный оператор WRITE используется для передачи информации с компьютера на устройство вывода.

Доступны две формы оператора:

**WRITE(u,f,ERR=L1,END=L2) k**

или

**WRITE(u,f,ERR=L1,END=L2)**

где:

|  |  |  |
| --- | --- | --- |
| **u** | - | определяет логический номер устройства |
| **f** | - | метка оператора FORMAT, описывающего тип преобразования данных, которые будут использоваться в выходной передаче. |
| **L1** | - | определяет переход при ошибке ввода-вывода. |
| **L2** | - | определяет переход при обнаружении конца файла. |
| **k** | - | список имен переменных, разделенный запятыми, определяющий выходные данные. |

Оператор WRITE(u,f) k используется для вывода данных, указанных в списке k в файл с логическим устройством u используя оператор f FORMAT, для указания внешнего представления этих данных (см. [Оператор FORMAT, 8.7](#_8.7_Операторы_FORMAT)).

Следующие примечания определяют действия оператора WRITE:

1. Несколько записей могут быть выведены одним оператором WRITE в количестве, определенном списком и спецификациями FORMAT.
2. Данные последовательно выводятся пока, список данных не исчерпан.
3. Если вывод производится на устройство, которое определяет записи фиксированной длины, и данные, определенные в списке, не заполняют запись полностью, остаток записи дополняется пробелами.

**Пример:**

WRITE(2,10)A,B,C,D

Данные, присвоенные переменным A, B, C и D, выводятся на устройство с логическим номером 2, отформатированные согласно оператора FORMAT с меткой 10.

ОператорWRITE(u,f) может использоваться для вывода символов, определенных в операторе FORMAT. В этом случае список переменных не требуется.

Например, вывести символы 'H CONVERSION'на устройство 1.

WRITE(1,26)

.

.

26 FORMAT ('H CONVERSION')

## 8.2 Бесформатные операторы READ и WRITE

Бесформатный ввод/вывод (т.е. без преобразования данных) осуществляется с помощью операторов:

**READ(u,ERR=L1,END=L2) k**

**WRITE(u,ERR=L1,END=L2) k**

где:

|  |  |  |
| --- | --- | --- |
| **u** | - | определяет логический номер устройства |
| **L1** | - | определяет переход при ошибке ввода/вывода. |
| **L2** | - | определяет переход при обнаружении конца файла. |
| **k** | - | список имен переменных, разделенный запятыми, определяющий входные/выходные данные. |

Следующие замечания определяют функции бесформатных операторов ввода/вывода.

1. Бесформатные операторы READ/WRITE выполняют передачу отображения данных в памяти без преобразования данных или редактирования.
2. Переданный объем данных соответствует числу переменных в списке k.

Общая длина списка имен переменных в бесформатном операторе READ не должна быть больше, чем длина записи. Если логическая длина записи и длина списка одинаковы, все записи считывается. Если длина списка короче длины логической записи непрочитанные элементы записи пропускаются.

1. Оператор WRITE(a)k записывает одну логическую запись.
2. Логическая запись может распространяться на несколько физических записей.

## 8.3 Ввод/вывод в файлы на диске

Операторы чтения (READ) или записи (WRITE) файлов на диске (LUN 6-10) автоматически открывают файл для ввода/вывода. Файл остается открытым, пока он не будет закрыт командой ENDFILE (см. [Раздел 8.4](#_8.4_Вспомогательные_операторы)), или до нормального завершения программы.

**Примечание**

Соблюдайте осторожность при выполнении вывода с последовательным доступом в файлы на дисках. Если вывод осуществляется в существующий файл, то он будет удален и заменен на новый файл с тем же именем.

### 8.3.1 Произвольный ввод/вывод на диск

См. также [Раздел 3 Microsoft FORTRAN Руководство пользователя](#_Раздел_3_FORTRAN-80_3).

Для произвольного доступа к файлу на диску номер записи определяется при помощи параметра REC=n в операторе READ или WRITE. Например:

I = 10

WRITE (6,20,REC=I,ERR=50) X, Y, Z

.

.

Этот сегмент программы записывает 10-ю запись на устройство с логическим номером 6. Если предыдущие 10 записей существуют, он перезаписывает их. Если 10 записей не существует, файл расширяется, чтобы создать ее. Любая попытка считать несуществующую запись приводит к ошибке ввода/вывода.

### 8.3.2 Подпрограмма OPEN

Кроме того, файл может быть открыт с помощью подпрограммы **OPEN**. В подпрограмме OPEN также могут быть назначены логические номера устройств (LUN) 1-5 в виде файлов на диске. Подпрограмма OPEN позволяет в программе указать имя файла и устройство, связанное с его логическим номером.

Открытие несуществующего файла создаст пустой файл с надлежащим именем. Открытие существующего файла, для вывода с последовательным доступом, удалит существующий файл. Открытие существующего файла, для ввода, предоставит доступ к текущему содержанию файла.

Форма вызова подпрограммы OPEN отличается в различных операционных системах. См. [Раздел 3 Microsoft FORTRAN Руководство пользователя](#_Раздел_3_FORTRAN-80_2).

## 8.4 Вспомогательные операторы ввода/вывода

Существуют три вспомогательных оператора ввода/вывода:

**BACKSPACE u**

**REWIND u**

**ENDFILE u**

Действия всех трех операторов зависят от номера логического устройства, с которым они используются (см. [Приложение B](#_Приложение_B_Интерфейс_1)). Если номер логического устройства связан с терминалом или построчным принтером, эти три оператора определены как no-ops (без операций).

Если номер логического устройства связан с диском, команды ENDFILE и REWIND позволяют программе в дальнейшем управлять файлами на диске. Оператор ENDFILE u закрывает файл, связанный с номером логического устройства u. Оператор REWIND u закрывает файл, связанный с номером логического устройства u, затем открывает его снова. Оператор BACKSPACE не реализован на данный момент, и поэтому вызывает ошибку при использовании.

## 8.5 Операторы ENCODE/DECODE

Операторы ENCODE и DECODE передают данные, согласно спецификациям формата, от одного участка памяти другому. Оператор DECODE преобразует данные из формата ASCII в указанный формат. Оператор ENCODE преобразует данные указанного формата в формат ASCII. Эти два оператора имеют форму:

**ENCODE(a,f) k**

**DECODE(a,f) k**

где:

**a** - имя массива

**f** - номер оператора FORMAT

**k** - список ввода/вывода

Оператор DECODE является аналогом оператора READ, так как он выполняет преобразование из ASCII во внутренний формат. Оператор ENCODE является аналогом оператора WRITE, так как он выполняет преобразование из внутреннего формата в ASCII.

**Примечание**

Следует позаботиться, чтобы массив a был достаточно большой, чтобы содержать все обработанные данные. Проверка переполнения отсутствует. Операция ENCODE, которая выходит за границы массива, вероятно, вытрет важные данные после массива. Переполнение при операции DECODE, приведет к попытке обработки данных за пределами массива.

## 8.6 Спецификации списка ввода/вывода

Большинство форм операторов ввода/вывода могут содержать упорядоченный список имен данных, который идентифицируют данные, подлежащие передаче. Порядок, в котором появляются элементы списка должен быть аналогичным порядка в котором соответствующие данные существуют (ввод), или будут существовать (вывод) на внешнем носителе ввода/вывода.

Списки ввода/вывода имеют следующий вид:

**m1,m2,..., mn**

где mi - список элементов, разделенных запятыми.

### 8.6.1 Типы элементов списка

Элемент списка может быть одиночным или многократным идентификатором данных.

Одиночный идентификатор данных - имя переменной или элемент массива.

**Примеры:**

A

C(26,1), R, K, D

B, I(10,10), S, F(1,25)

Примечание:Подсписки не реализованы.

Многократный идентификатор элементов данных может быть записан в двух формах:

1. Имя массива, появляющееся в списке без индексов, считают эквивалентным последовательному перечислению каждого элемента массива.

**Пример:**

Если B - двумерный массив, элемент списка B эквивалентен:

B(1,1),B(2,1),B(3,1),B(1,2),B(2,2)...,B(j,k)

где j и k-индексы, ограниченны максимальными значениями размера массива B.

1. Элементы неявного цикла, являющиеся списками из одного или нескольких одиночных идентификаторов данных (простых, или с индексами), либо, в свою очередь, неявных циклов разделенных символом запятой и выражение вида:

i = m1,m2,m3 или i = m1,m2

заключенное в круглые скобки.

Элементы i, m1, m2, m3 имеют тот же смысл, что и для оператора цикла DO. Неявный цикл применяется ко всем элементам списка, заключенным в круглые скобки.

**Примеры:**

| Неявные списки | Эквивалентные им списки |
| --- | --- |
| (X(I),I=1,4) | X(1), X(2), X(3), X(4) |
| (Q(J) ,R(J) ,J=1,2) | Q(1), R(1), Q(2) ,R(2) |
| (G(K),K=1,7,3) | G(1) ,G(4) ,G(7) |
| ((A(I,J),I=3,5),J=1,9,4) | A(3,1), A(4,1), A(5,1), A(3,5), A(4,5), A(5,5), A(3,9), A(4,9), A(5,9) |
| (R(M),M=I,2),I,ZAP(3) | R(1), R(2), I, ZAP(3) |
| (R(3),T(I),I=1,3) | R(3), T(1), R(3), T(2) R(3), T(3) |

Таким образом, элементы матрицы, например, могут быть переданы в порядке, отличном от того, в котором они расположены в памяти.

Массив A(3,3) в памяти расположен в порядке

A(1,1),A(2,1),A(3,1),A(1,2),A(2,2),A(3,2),A(1,3),A(2,3),A(3,3)

Указание передачи массива в списке в виде элемента неявного цикла   
((A(I,J),J=1,3),I=1,3), осуществит передачу в следующем порядке:

A(1,1),A(1,2),A(1,3),A(2,1),A(2,2),A(2,3),A(3,1),A(3,2),A(3,3)

### 8.6.2 Специальные замечания по спецификации списка

1. Упорядочение списка выполняется слева направо с повторением элементов в скобках (кроме индексов) с подстановкой в неявном цикле подразумеваемых параметров индекса.
2. Массивы передаются появлением имени массива (без индексов) в списке ввода/вывода.
3. Константы могут появиться в списке ввода/вывода только в качестве индексов или в качестве параметров индексации.
4. Для входных списков подразумеваемые элементы i, m1, m2 и m3 могут не появиться в круглых скобках как элементы списка.

**Примеры:**

1. READ(1,20) (I,J,A(I),I=1,J,2) не допускается

2. READ(1,20) I,J,(A(I),I=1,J,2) допускается

3. WRITE(1,20) (I,J,A(I),I=1,J,2) допускается

Рассмотрим следующие примеры:

DIMENSION A(25)

A(1) = 2.1

A(3) = 2.2

A(5) = 2.3

J = 5

WRITE (1,20) J,(I,A(I),I=1,J,2)

.

.

Вывод этого оператора WRITE будет следующий

5, 1, 2.1, 3, 2.2, 5, 2.3

1. В одном списке может присутствовать любое количество элементов.
2. В форматной передаче READ(u,f)k, WRITE(u,f)k каждый элемент должен иметь правильный тип, в соответствии с указанным в операторе FORMAT.

## 8.7 Операторы FORMAT

Операторы FORMAT- невыполняемые, порождающие операторы, используемые в сочетании с форматными операторами READ и WRITE. Они определяют методы преобразования и информацию о редактировании при передаче данных между ячейками памяти в компьютере и внешними носителями.

Операторам FORMAT необходима метка (f) для ссылки на них в операторах READ(u,f)k или WRITE(u,f)k.

Общая форма оператора FORMAT выглядит следующим образом:

**m FORMAT(s1,s2,...,sn/s1',s2',...,sn'/...)**

где m - метка оператора, и каждый si - поле дескриптора.

Слово FORMAT и круглые скобки должны присутствовать, как показано.

Наклонная черта (/) и запятая (,) являются символами разделителей полей и описаны в отдельном абзаце.

Поле дескриптора определяет, каким образом внешние записи заполняют один передаваемый элемент.

### 8.7.1 Поля дескрипторов

Поля дескрипторов описывают размеры полей данных и определяют тип преобразования, которое будет произведено над каждым переданным данным. У дескрипторов поля FORMAT может быть любая из следующих форм:

|  |  |
| --- | --- |
| Дескриптор | Классификация |
| rFw.d  rGw.d  rEw.d  rDw.d  rIw | Числовые преобразования |
| rLw | Логическое преобразование |
| rAw  nHh1h2...hn  'l1l2...ln' | Текстовое преобразование |
| nX | Спецификация интервала |
| mP | Коэффициент масштабирования |

где:

1. w и n положительные целые константы, определяющие ширину поля (в том числе для цифр, знаков после запятой и алгебраических знаков) во внешнем представлении данных.
2. d - целое число, определяющее число дробных цифр, появляющихся во внешнем представлении данных.
3. Символы F, G, E, D, I, A и L указывают тип преобразования, которое будет применено к элементам в списке ввода/вывода.
4. r - необязательное, ненулевое целое число, указывающее, что дескриптор будет повторен r раз.
5. hi и li символы из набора символов FORTRAN.
6. m - целая константа (положительная, отрицательная, или ноль) означающая коэффициент кратности.

### 8.7.2 Числовые преобразования

Входные операции с любым из числовых преобразований позволят данным быть представленными в "свободном формате", т.е., запятые могут использоваться, чтобы разделить поля во внешнем представлении.

**Преобразование F-типа**

Форма: **Fw.d**

Это преобразование обрабатывает данные вещественного типа или двойной точности, состоящие из w символов, из которых d считаются дробными.

**F-вывод**

Значения преобразуются и выводятся в виде знака "минус" (если отрицательные), сопровождаемого целой частью числа, десятичной точки и d-цифр дробной части числа.

Если значение не заполняет поле, оно выравнивается по правому краю, и вставляется необходимое количество предшествующих пробелов, чтобы заполнить поле. Если значение требует больше позиций, чем выделено w, выводятся первые w-1 цифры значения с предшествующими звездочками.

**Примеры F-вывода**

|  |  |  |
| --- | --- | --- |
| Дескриптор FORMAT | Внутреннее значение | Вывод (⊔=пробел) |
| F10.4 | 368.42 | ⊔⊔368.4200 |
| F7.1 | -4786.361 | -4786.4 |
| F8.4 | 8.7E-2 | ⊔⊔0.0870 |
| F6.4 | 4739.76 | \*.7600 |
| F7.3 | -5.6 | ⊔-5.600 |

\* Примечание потеря ведущих цифр в 4-й строке выше.

**F-ввод**

(См. описание E-ввода ниже.)

**Преобразование E-типа**

Форма: **Ew.d**

Это преобразование обрабатывает данные вещественного типа или двойной точности, состоящие из w символов, из которых d считаются дробными.

**E -вывод**

Значения преобразуются, округляются до d цифр, и выводятся в виде:

1. знак минус (если число отрицательное),
2. ноль и десятичная точка,
3. d десятичных цифр,
4. буква E,
5. знак экспоненты (минус или пробел),
6. две цифры экспоненты,

Значения выровнены по правому краю в поле w с предшествующими пробелами, чтобы заполнить поле при необходимости. Ширина поля w должна удовлетворять соотношению:

w ≥ d + 7

В противном случае значащие символы могут быть потеряны.

**Некоторые примеры E-вывода:**

|  |  |  |
| --- | --- | --- |
| Дескриптор FORMAT | Внутреннее значение | Вывод (⊔=пробел) |
| E12.5 | 76.573 | ⊔⊔.76573E⊔02 |
| E14.7 | -32672.354 | -⊔.3267235E⊔05 |
| E13.4 | -0.0012321 | ⊔⊔-⊔.1232E-02 |
| E8.2 | 76321.73 | ⊔.76E⊔05 |

**E –ввод**

Значения данных, которые должны быть обработаны E, F или G преобразованием, могут иметь относительно свободный формат на внешнем входном носителе. Формат идентичен для любого преобразования и обрабатывается следующим образом:

1. Ведущие пробелы (игнорируются)
2. Знак + или - (без знака ввод считается положительным)
3. Строка цифр
4. Десятичная точка
5. Вторая строка цифр
6. Символ E
7. Знак + или -
8. Десятичная экспонента

Каждый элемент в списке выше необязательный, но следующие условия должны соблюдаться:

1. Если в операторе FORMAT пункты 3 и 5 (см. выше) присутствуют, то требуется пункт 4.

2. Если в операторе FORMAT присутствует пункт 8, то требуется пункт 6 или 7 или оба.

3. Все не ведущие пробелы считаются нулями.

Входные данные могут иметь любое количество цифр в длину, и будут обработаны правильные величины, но точность будет поддерживаться только в пределах, указанных в [Разделе 3](#_Раздел_3_Представление) для вещественных данных.

Примеры E- и F- и G- ввода

|  |  |  |
| --- | --- | --- |
| Дескриптор FORMAT | Ввод (⊔=пробел) | Внутреннее значение |
| E10.3 | +0.23756+4 | +2375.60 |
| E10.3 | ⊔⊔⊔⊔⊔17631 | +17.631 |
| G8.3 | ⊔1628911 | +1628.911 |
| F12.4 | ⊔⊔⊔⊔-6321132 | -632.1132 |

Отметьте в вышеупомянутых примерах, что, если десятичная точка не задана среди входных символов, d в спецификации FORMAT устанавливает десятичную точку в сочетании с экспонентой, если задана. Если десятичная точка присутствует во входных символах, d спецификация игнорируется.

Буквы E, F и G взаимозаменяемые во входных спецификациях формата. Конечный результат - то же.

**Преобразование D-типа**

D-ввод и D-вывод идентичны E-вводу и E-выводу кроме экспоненты, которая может быть определена "D" вместо "E".

**Преобразование G-типа**

Форма: **Gw.d**

Это преобразование обрабатывает данные вещественного типа или двойной точности, состоящие из w символов, из которых d считаются значимыми.

**G –ввод**

См. описание E–ввода

**G –вывод**

Метод выходного преобразования является функцией от величины выводимого числа. Пусть n - порядок числа. Следующая таблица показывает, как число будет выведено:

|  |  |
| --- | --- |
| Величина | Эквивалентное преобразование |
| .1 ≤ n < 1 | F(w-4).d,4X |
| 1 ≤ n < 10 | F(w-4).(d-1),4X |
| . | . |
| . | . |
| . | . |
| 10d-2 ≤ n < 10d-1 | F(w-4).1,4X |
| 10d-1 ≤ n < 10d | F(w-4).0,4X |
| Иначе | Ew.d |

**Преобразование I**

Форма: **Iw**

Только целые данные могут быть преобразованы этой формой, значение w определяет ширину поля.

**I –вывод**

Значения преобразуются в целые константы. Отрицательным величинам предшествует знак "минус". Если значение не заполняет поле, оно выравнивается по правому краю, и вставляется необходимое количество предшествующих пробелов, чтобы заполнить поле. Если значение превышает ширину поля, выводятся только младшие w-1 значащих символов c предшествующей звездочкой.

**Примеры:**

|  |  |  |
| --- | --- | --- |
| Дескриптор FORMAT | Внутреннее значение | Вывод (⊔=пробел) |
| I6 | +281 | ⊔⊔⊔281 |
| I6 | -23261 | -23261 |
| I3 | 126 | 126 |
| I4 | -226 | -226 |

**I –ввод**

Вводится поле из w символов и преобразуется во внутренний целый формат. Знак "минус" может предшествовать целым цифрам. Если знак отсутствует, значение считают положительным.

Принимаются целые значения в диапазоне от -32 768 до 32 767. Не предшествующие пробелы рассматривают как нули.

**Примеры:**

|  |  |  |
| --- | --- | --- |
| Дескриптор FORMAT | Ввод (⊔=пробел) | Внутреннее значение |
| I4 | ⊔124 | 124 |
| I4 | -124 | -124 |
| I7 | ⊔⊔6732⊔ | 67320 |
| I4 | 1⊔2⊔ | 1020 |

### 8.7.3 Преобразования текста

**Преобразование A-типа**

Форма A преобразования выглядит следующим образом:

**Aw**

Этот дескриптор считывает и выводит текстовые символы из соответствующего элемента списка в неизмененном виде.

Максимальное количество фактических символов, которое может быть передано между внутренними и внешними представлениями с использованием дескриптора Aw в четыре раза превышает количество ячеек памяти выделенных соответствующему элементу в списке (т.е., 1 символ для логических элементов, 2 символов для целых, 4 символа для вещественных и 8 символов для элементов двойной точности).

**A-вывод:**

Если w больше 4n (где n количество ячеек памяти выделенной элементу из списка вывода), внешнее поле вывода будет состоять из w-4n пробелов, за которыми следуют 4n символов внутреннего представления. Если w меньше 4n, внешнее поле вывода будет состоять из расположенных слева w символов внутреннего представления.

**Примеры:**

|  |  |  |  |
| --- | --- | --- | --- |
| Дескриптор FORMAT | Внутреннее значение | Тип | Вывод (⊔=пробел) |
| A1 | Al | Integer | A |
| A2 | AB | Integer | AB |
| A3 | ABCD | Real | ABC |
| A4 | ABCD | Real | ABCD |
| A7 | ABCD | Real | ⊔⊔⊔ABCD |

**A-ввод:**

Если w больше 4n (где n количество ячеек памяти выделенной элементу из списка ввода), крайние правые 4n символов будут взяты из поля внешнего ввода. Если w меньше 4n, w символов вводятся с левого края внутреннего представления с w-4nконечными пробелами.

**Примеры:**

|  |  |  |  |
| --- | --- | --- | --- |
| Дескриптор FORMAT | Вводимые символы | Тип | Внутреннее значение (⊔=пробел) |
| A1 | A | Integer | A⊔ |
| A3 | ABC | Integer | AB |
| A4 | ABCD | Integer | AB |
| A1 | A | Real | A⊔⊔⊔ |
| A7 | ABCDEFG | Real | DEFG |

**Преобразование H-типа**

Форма H преобразования выглядит следующим образом:

**nHh1h2...hn**

**'h1h2...hn'**

Эти дескрипторы обрабатывают текстовые строки символов между дескриптором и внешним полем, где каждый hi представляет любой символ из набора символов ASCII.

**Примечание**

Требуется особое внимание, если во второй форме нужно использовать апостроф (') в литеральной строке. Апостроф в строке представляется двумя последовательными апострофы. См. Примеры ниже.

**H-вывод:**

n символов hi, размещаются во внешнем поле. В форме nHh1h2...hn количество символов в строке должно быть столько, сколько указано в n. В противном случае, символы из других дескрипторов будут рассматриваться как часть строки. В обеих формах, пробелы считаются символами.

**Примеры:**

|  |  |
| --- | --- |
| Дескриптор FORMAT | Вывод (⊔=пробел) |
| 1HA или 'A' | A |
| 8H⊔STRING⊔ или '⊔STRING⊔' | ⊔STRING⊔ |
| 11HX(2,3)=12.0 или 'X(2,3)=12.0' | X(2,3)=12.0 |
| 11HI⊔SHOULDN'T или I⊔SHOULDN''T' | I⊔SHOULDN'T |

**H-ввод:**

n символов строки hi заменяют ближайшие n символов из входной записи. Это приводит к новой строке символов в поле дескриптора.

|  |  |  |
| --- | --- | --- |
| Дескриптор FORMAT | Ввод (⊔=пробел) | Результирующий дескриптор |
| 4H1234 или '1234' | ABCD | 4HABCD или 'ABCD' |
| 7H⊔⊔FALSE или '⊔⊔FALSE' | ⊔FALSE⊔ | 7H⊔FALSE⊔ или '⊔FALSE⊔' |
| 6H⊔⊔⊔⊔⊔⊔ или '⊔⊔⊔⊔⊔⊔' | MATRIX | 6HMATRIX или 'MATRIX' |

### 8.7.4 Логические преобразования

Форма логического преобразования следующая:

**Lw**

**L-вывод**

Если значение элемента в списке вывода, соответствующее этому дескриптору, будет 0, то будет выведена буква F, иначе, будет выведена буква T. Если w больше чем 1, то перед буквами будут добавлены w-1 пробела.

**Примеры:**

|  |  |  |
| --- | --- | --- |
| Дескриптор FORMAT | Внутреннее значение | Вывод (⊔=пробел) |
| L1 | = 0 | F |
| L1 | ≠ 0 | T |
| L5 | ≠ 0 | ⊔⊔⊔⊔T |
| L7 | = 0 | ⊔⊔⊔⊔⊔⊔F |

**L-ввод**

Внешнее представление занимает w позиций. Оно состоит из необязательных пробелов с последующим символом "Т" или "F", а затем необязательных символов.

### 8.7.5 Дескриптор X

Форма X преобразований следующая:

**nX**

Этот дескриптор не выполняет преобразование, и не имеет соответствующего элемента в списке ввода/вывода. Если используется для вывода, он вставляет n пробелов в выходную запись. При вводе этот дескриптор вызывает пропуск следующих n символов входной записи. Обратите внимание на то, что необходимо писать 1X, поскольку только символ X не будет работать.

**Примеры вывода:**

|  |  |
| --- | --- |
| Оператор FORMAT | Вывод (⊔=пробел) |
| 3 FORMAT(1HA,4X,2HBC) | A⊔⊔⊔⊔BC |
| 7 FORMAT(3X,4HABCD,1X) | ⊔⊔⊔ABCD⊔ |

**Примеры ввода:**

|  |  |  |
| --- | --- | --- |
| Оператор FORMAT | Входная строка | Результат ввода |
| 10 FORMAT(F4.1,3X,F3.0) | 12.5ABC120 | 12.5, 120 |
| 5 FORMAT(7X,I3) | 1234567012 | 012 |

### 8.7.6 Дескриптор P

Дескриптор P используется для указания масштабного коэффициента для преобразований вещественных значений (F, E, D, G). Форма nP, где n является целой константой (положительной, отрицательной, или ноль).

Масштабный коэффициент автоматически обнуляется в начале каждого форматного вызова ввода/вывода (каждого оператора READ или WRITE). Если дескриптор P встречается при сканировании FORMAT, масштабный коэффициент изменяется на n. Масштабный коэффициент остается неизменным до появления другого дескриптора n, или завершения ввода/вывода.

**Эффект масштабного коэффициента при вводе**

Во время E, F или G ввода масштабный коэффициент действует, только если, отсутствует показатель степени во внешнем представлении. В этом случае внутреннее значение будет в 10n раз меньше, чем внешнее значение (число будет разделено на 10n перед сохранением).

Эффект масштабного коэффициента при выводе.

**E-вывод, D-вывод**

Коэффициент сдвигает влево на n мест положение десятичной точки, а показатель уменьшается на n (значение остается тем же).

**F-вывод**

Внешнее значение будет в 10n раз больше внутреннего значения.

**G-вывод**

Масштабный коэффициент игнорируется, если внутреннее значение достаточно мало, чтобы быть выведенным с помощью преобразования F. В противном случае, эффект такой же, как для E-вывода.

### 8.7.7 Специальные функции управления операторов FORMAT

#### 8.7.7.1 Спецификация повторения

1. Поля дескрипторов E, F, D, G, I, L и A могут быть указаны как повторяющиеся дескрипторы с помощью счетчика повторений r в виде rEw.d, rFw.d, rGw.d, rlw, rLw, rAw. Следующие пары операторов FORMAT эквивалентны:

66 FORMAT(3F8.3,F9.2)

C ЭКВИВАЛЕНТЕН:

66 FORMAT(F8.3,F8.3,F8.3,F9.2)

C

14 FORMAT(2I3,2A5,2E10.5)

C ЭКВИВАЛЕНТЕН:

14 FORMAT(I3,I3,A5,A5,E10.5,E10.5)

1. Повторение группы полей дескрипторов осуществляется путем заключения группы в скобки, перед которыми записывается счетчик повторений.

Отсутствие счетчика повторений указывает, что число повторений равно единице. Допускается до двух уровней скобок, включая скобки, требуемые оператору FORMAT.

Обратите внимание на следующие эквивалентные инструкции:

22 FORMAT (I3,4(F6.1,2X))

C ЭКВИВАЛЕНТНО:

22 FORMAT (I3,F6.1,2X,F6.1,2X,F6.1,2X,

1 F6.1,2X)

1. Повторение дескрипторов FORMAT также происходит, когда все дескрипторы оператора FORMAT были использованы, но еще остались элементы в списке ввода/вывода, которые не были обработаны.

Когда это происходит, дескрипторы FORMAT используются повторно, начиная с первой открывающей скобки в операторе FORMAT.

Счетчик повторений, предшествующий дескриптору(ам) в скобках, которые будут повторно использоваться, также активно участвует в повторном использовании. Этот тип повторного использования дескрипторов FORMAT прекращает обработку текущей записи, и инициирует обработку новой записи каждый раз, начиная повторное использование.

Разграничение записей в этих условиях совпадает с описанным в [абзаце 8.7.7.2](#_8.7.7.2_Разделители_полей) ниже.

**Пример ввода:**

DIMENSION A(100)

READ (3,13) A

.

.

13 FORMAT (5F7.3)

В этом примере первые 5 величин из каждых 20 записей вводятся и присваиваются элементам массива A.

**Пример вывода:**

WRITE (6,12)E,F,K,L,M,KK,LL,MM,K3,L3,M3

12 FORMAT (2F9.4,3(3I7,/))

В этом примере три записи. Запись 1 содержит E, F, K, L и M. Поскольку дескриптор 3I7 используется повторно два раза, запись 2 содержит KK, LL и MM и запись 3 содержит K3, L3 и M3.

#### 8.7.7.2 Разделители полей

Два соседних дескриптора должны быть разделены в операторе FORMAT запятыми, или одним или более символами наклонной черты.

**Пример:**

2H0K/F6.3 или 2H0K,F6.3

Наклонная черта не только разделяет дескрипторы полей, но она также определяет разграничение форматных записей.

Каждая наклонная черта завершает запись и устанавливает следующую запись для обработки. Остаток входной записи игнорируется. Остаток от выходной записи заполняется пробелами. Последовательные символы наклонной черты (///.../) заставляют игнорировать последующие записи на вводе и создавать последовательность пустых записей, которые будут записаны при выводе.

**Пример вывода:**

DIMENSION A(100), J(20)

WRITE (7,8) J,A

8 FORMAT (10I7/10I7/50F7.3/50F7.3)

В этом примере данные, определенные списком оператора WRITE, выводятся на устройство 7 согласно спецификациям оператора FORMAT с меткой 8. Четыре записи записываются следующим образом:

|  |  |  |  |
| --- | --- | --- | --- |
| Запись 1 | Запись 2 | Запись 3 | Запись 4 |
| J(l) | J(11) | A(1) | A(51) |
| J(2) | J(12) | A(2) | A(52) |
| . | . | . | . |
| . | . | . | . |
| J(10) | J(20) | A(50) | A(100) |

**Пример ввода:**

DIMENSION B(10)

READ(4,17) B

17 FORMAT(F10.2/F10.2///8F10.2)

В этом примере два элемента массива B(1) и B(2) получают свои значения из первых полей данных последовательных записей (остатки двух записей игнорируются). Третья и четвертая записи игнорируются, а остальные элементы массива заполняются из пятой записи.

### 8.7.8 Обработка FORMAT, список спецификаций и разграничение записей

Следует отметить следующие взаимосвязи и взаимодействия между обработкой FORMAT, списками ввода/вывода и разграничением записей:

1. Выполнение операторов форматных операторов READ или WRITE инициирует обработку FORMAT.
2. Преобразование данных осуществляется в зависимости от информации совместно представленных элементов списка ввода/вывода и дескрипторов полей в операторе FORMAT.
3. Если присутствует список ввода/вывода, по крайней мере, один дескриптор типа E, F, D, G, I или L должен быть представлен в операторе FORMAT.
4. Каждое выполнение форматного оператора READ вызывает чтение новой записи.
5. Каждый элемент в списке ввода согласуется со строкой символов в записи, и дескриптором типа E, F, D, G, I или L в операторе FORMAT.
6. Дескрипторы H и X передают информацию непосредственно между внешними записями и полями дескрипторов без привязки к списку элементов.
7. При вводе, когда в операторе FORMAT встречается косая черта или дескрипторы FORMAT были исчерпаны и повторно используются, дескрипторы инициируются, обработка текущей записи прекращается и происходит следующее:
   1. Все необработанные символы в записи игнорируются.
   2. Если необходим дальнейший ввод для удовлетворения требований списка, читается следующая запись.
8. Оператор READ завершен, когда все элементы во входном списке были удовлетворены если:
   1. Следующий дескриптор FORMAT - E, F, G, I, L или A.
   2. Обработка FORMAT достигла последней внешней правой круглой скобки оператора FORMAT.

Если входной список был удовлетворен, но следующим дескриптором FORMAT является H или X, обрабатываются дополнительные данные (с возможностью новых записей на входе) до тех пор, пока не выполняется одно из вышеперечисленных условий.

1. Если обработка FORMAT достигает последней правой круглой скобки оператора FORMAT, но существуют несколько не обработанных элементов списка, все или часть дескрипторов используются повторно. (См. [Пункт 3, Подпункт 8.7.7.1 Спецификации повторения](#_8.7.7.1_Спецификация_повторения))
2. При выполнении форматного оператора WRITE, при появлении наклонной черты в операторе FORMAT, или если обработка FORMAT достигла самой правой круглой скобки, каждый раз происходит запись. Обработка FORMATзавершается при выполнении одного из условий завершения READ, описанных в п. 8 выше. Неполные записи заполняются пробелами, чтобы обеспечить длину записи.

### 8.7.9 Вывод форматных записей на печать

Форматный ввод/вывод на консоль или принтер использует первый символ каждой записи для управления кареткой. Символ управления кареткой никогда не распечатывается. Символ управления кареткой определяет, какие действия будут выполнены, прежде чем распечатана строка. Опции следующие:

|  |  |
| --- | --- |
| Символ управления | Действия перед печатью |
| 0 | Пропуск 2 строк |
| 1 | Переход к началу следующей страницы |
| + | Нет продвижения |
| Другой | Пропуск 1 строки |

Форматный ввод/вывод на диск не требует, чтобы первый символ каждой записи был символом управления кареткой. Записи завершаются символом возврата каретки (X'0D'). В файле отсутствуют символы перевода строки (X'0A'), если не записаны в него явно.

### 8.7.10 Спецификации формата в массивах

Ссылка f в форматных операторах READ или WRITE (см. [**п. 8.1**](#_8.1_Форматные_операторы)) вместо метки может быть именем массива.

Если сделана такая ссылка, во время выполнения оператора READ/WRITE, первая часть информации, содержавшейся в массиве в естественном порядке, должна составить допустимую спецификацию FORMAT.

После правой круглой скобки, которой заканчивается спецификации FORMAT, массив может содержать информацию не относящуюся к FORMAT.

Спецификации FORMAT, которая размещена в массиве, должна иметь тот же вид, который используется оператором FORMAT (т.е., начинаться с левой скобки и заканчиваться правой скобкой).

Спецификация формата может быть вставлена в массив путем использования оператора инициализации DATA, либо путем использования оператора READ c оператором FORMAT содеращим дескриптор Aw.

**Пример:**

Предположим, спецификации FORMAT

(3F10.3,4I6)

или подобная спецификация из 12 символов должны быть сохранены в массив. Массив должен обеспечить минимум 3 ячейки памяти для данных.

Ниже показан код на языке FORTRAN, показывающий различные способы задания спецификации FORMAT, а затем ссылки на массив для форматного READ или WRITE.

C ОБЪЯВЛЕНИЕ ВЕЩЕСТВЕННОГО МАССИВА

DIMENSION A(3), B(3), M(4)

C ИНИЦИАЛИЗАЦИЯ ФОРМАТ ОПЕРАТОРОМ DATA

DATA A/'(3F1', '0.3, ','4I6) '/

.

.

C ЧТЕНИЕ ДАННЫХ ИСПОЛЬЗУЯ СПЕЦИФИКАЦИЮ

C FORMAT ИЗ МАССИВА A

READ(6,A) B, M

C ОБЪЯВЛЕНИЕ ЦЕЛОГО МАССИВА

DIMENSION IA(4), B(3), M(4)

.

.

C ЧТЕНИЕ СПЕЦИФИКАЦИИ FORMAT

READ (7,15) IA

C FORMAT ДЛЯ ВВОДА СПЕЦИФИКАЦИИ FORMAT

C

15 FORMAT(4A2)

.

.

C ЧТЕНИЕ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ

C ПРЕДЫДУЩЕГО ВВОДА СПЕЦИФИКАЦИИ FORMAT

READ (7,IA) B,M

.

.

# Раздел 9 Функции и подпрограммы

Язык FORTRAN обеспечивает средства для определения и использования часто необходимых процедур программирования, таким образом, что оператор или операторы процедур должны появиться в программе только один раз, но могут быть упомянуты и принесены в логическую последовательность выполнения программы каждый раз, тогда и так часто как это необходимо.

Этими процедурами являются:

1. Оператор функции.
2. Библиотечные функции.
3. Подпрограммы FUNCTION.
4. Подпрограммы SUBROUTINE.

У каждой из этих процедур есть свои собственные уникальные требования при определении и вызове. Эти требования описаны в последующих абзацах этого раздела.

Однако, некоторые особенности, характерны для всей группы или двух или более процедур. Эти общие особенности заключаются в следующем:

1. Каждая из этих процедур может быть вызвана ссылкой на свое имя, которое, во всех случаях, содержит от одного до шести буквенно-цифровых символов, из которых первый является буквой.
2. Первые три определяются как "функции" и подобны в этом:
   1. Они всегда однозначные (т.е., они возвращают одно значение в программный модуль, из которого к ним происходит обращение).
   2. Они упомянуты в выражении, содержащем имя функции.
   3. Они должны быть определены в операторах описания типа, если тип данных возвращаемого результата должен отличаться от обозначенного предопределенным соглашением.
3. Подпрограммы FUNCTION и SUBROUTINE считаются программными модулями.

В следующих описаниях этих процедур термин вызывающая программа означает модуль программы или процедуры, в котором сделана ссылка на процедуру, а термин вызываемая программа означает процедуру, на которую сделана ссылка.

## 9.1 Оператор PROGRAM

Оператор PROGRAM предназначен для определения имени основного модуля программы.

Форма оператора:

**PROGRAM name**

Оператор PROGRAM, если присутствует, должен появиться перед любым другим оператором в модуле программы. Параметр name (имя) состоит из 1-6 алфавитно-цифровых символов, первым из которых является буква.

Если оператор PROGRAM отсутствует в основной программе, компилятор присваивает ей имя $MAIN.

## 9.2 Оператор-функция

Оператор-функция определяется единственным арифметическим или логическим оператором присваивания, и относятся только к модулю программы, в котором она появляется.

Общая форма оператора-функции следующая:

**f(a1,a2,...an) = e**

где f - имя функции, ai фиктивные параметры, и e - арифметическое или логическое выражение.

Правила размещения, структуры и использования оператора-функции следующие:

1. Определения оператора-функции, если оно существует в модуле программы, должно предшествовать всем исполнимым операторам в модуле и следовать за всеми операторами описания.
2. ai -различные имена переменных или элементов массива, но, будучи фиктивными переменными, они могут иметь те же имена, что и переменные того же типа появляющиеся в другом месте в программном модуле.
3. Выражение e строится согласно правилам [Раздела 4](#_Раздел_4_Выражения) и может содержать только ссылки на фиктивные параметры и нелитеральные константы, ссылки на переменные и элементы массивов, ссылки на служебные и математические функции и ссылки на ранее определенные оператор-функции.
4. Тип любого имени оператор-функции или параметра, который отличается от его типа предопределенного по умолчанию, должен быть определен оператором описания типа.
5. Отношение между f и e должно соответствовать правилам присвоения, описанным в [Разделе 5](#_Раздел_5_Операторы).
6. Оператор-функции вызывается указанием ее имени, сопровождаемого списком параметров заключенного в скобки. Выражение вычисляется, используя параметры, определенные в вызове, и ссылка заменяется результатом.
7. Каждый параметр в списке аргументов должен быть согласован, по типу с фиктивным в операторе-функции.

Приведенный ниже пример показывает, оператор-функцию и инструкции вызова.

C ОПРЕДЕЛЕНИЕ ОПЕРАТОР-ФУНКЦИИ

C

FUNC1(A,B,C,D) = ((A+B)\*\*C)/D

C

C ВЫЗОВ ОПЕРАТОР-ФУНКЦИИ

C

A12=A1-FUNC1(X,Y,Z7,C7)

## 9.3 Библиотечные функции

Библиотечные функции - группа системных и математических функций, которые "встроены" в систему FORTRAN. Их имена предопределены для компилятора, и их тип определяется автоматически.

Функции перечислены в **Таблицах 9-1** и **9-2**. В таблицах параметры обозначены как a1,a2,...,an, если требуется больше одного параметра, или как a, если требуется только один. Библиотечная функция вызывается, когда ее имя используется в арифметическом выражении. Такая ссылка принимает следующую форму:

**f(a1,a2,...an)**

где f - имя функции, и ai фактические параметры. Параметры должны быть согласованы по типу, числу и порядку следования со спецификациями, обозначенными в **Таблицах 9-1** и **9-2**.

Кроме функций, перечисленных в **Таблицах 9-1** и **9-2**, предоставляются четыре дополнительные библиотечные подпрограммы, чтобы разрешить прямой доступ к оборудованию 8080 (или Z80). Это:

PEEK, POKE, INP, OUT

В описании ниже:

b, b1 и b2 это константы или переменные типа BYTE.

i это целая (INTEGER) константа или переменная

PEEK и INP являются логическими функциями. POKE и OUT являются подпрограммами. PEEK и POKE обеспечивают прямой доступ к любой ячейке памяти. PEEK(i) возвращает содержимое ячейки памяти, указанной в i. CALL POKE(i,b) содержимое ячейки памяти, указанной в i заменяет на содержимое b. INP и OUT обеспечивают прямой доступ к портам ввода/вывода. INP(b) выполняет чтение из порта b и возвращает введенное 8-битное значение. CALL OUT(b1,b2) записывает значение b2 в порт, указанный b1.

RAN другая функция в FORTRAN библиотеки. RAN является генератором случайных чисел, который совместим с компилятором Microsoft BASIC и интерпретатором BASIC-80. Сгенерированное случайное число, является вещественным десятичным числом между 0 и 1.

Генератор случайных чисел вызывается оператором следующего вида:

**<переменная> = RAN(x)**

Если x<0, возвращается первое значение новой последовательности случайных чисел.

Если x=0, снова возвращается последнее сгенерированное случайное число.

Если x>0, создается следующее случайное число в последовательности.

Примеры использования библиотечных функций:

Al = B+FLOAT(17)

MAGNI = ABS(KBAR)

PDIF = DIM(C,D)

S3 = SIN(T12)

ROOT = (-B+SQRT(B\*\*2-4.\*A\*C))/

1 (2.\*A)

Таблица 9 - 1 Встроенные функции

| Встроенная функция | Имя функции | Определение | Тип | |
| --- | --- | --- | --- | --- |
| аргумента | функции |
| Абсолютное значение | ABS | |a| | Real | Real |
| IABS | Integer | Integer |
| DABS | Double | Double |
| Усечение | AINT | sign(a) умножается на |a| | Real | Real |
| INT | Real | Integer |
| IDINT | Double | Integer |
| Взятие остатка | AMOD | a1(mod a2) | Real | Real |
| MOD | Integer | Integer |
| Выбор наибольшего значения | AMAX0 | max(a1,a2,…) | Integer | Real |
| AMAX1 | Real | Real |
| MAX0 | Integer | Integer |
| MAX1 | Real | Integer |
| DMAX1 | Double | Double |
| Выбор наименьшего значения | AMIN0 | min(a1,a2,…) | Integer | Real |
| AMIN1 | Real | Real |
| MIN0 | Integer | Integer |
| MIN1 | Real | Integer |
| DMIN1 | Double | Double |
| Преобразование в вещественное | FLOAT | Преобразование из целого в вещественное | Real | Integer |
| Преобразование в целое | IFIX | Преобразование из вещественного в целое | Real | Integer |
| Передача знака | SIGN | sign of a2 times |a1| | Real | Real |
| ISIGN | Integer | Integer |
| DSIGN | Double | Double |
| Положительная разность | DIM | a1-min(a1,a2) | Real | Real |
| IDIM | Integer | Integer |
|  | SNGL | Получение максимальной значащей части аргумента дойной точности | Double | Real |
|  | DBLE | Преобразование вещественного аргумента к двойной точности | Real | Double |

Таблица 9 - 2 Основные внешние функции

| Внешняя функция | Имя функции | Число аргументов | Определение | Тип | |
| --- | --- | --- | --- | --- | --- |
| аргумента | функции |
| Экспоненциальная функция | EXP | 1 | ea | Real | Real |
| DEXP | 1 | Double | Double |
| Натуральный логарифм | ALOG | 1 | ln(a) | Real | Real |
| DLOG | 1 | Double | Double |
| Десятичный логарифм | ALOG10 | 1 | lg(a) | Real | Real |
| DLOG10 | 1 | Double | Double |
| Тригонометрический синус | SIN | 1 | sin(a) | Real | Real |
| DSIN | 1 | Double | Double |
| Тригонометрический косинус | COS | 1 | cos(a) | Real | Real |
| DCOS | 1 | Double | Double |
| Гиперболический тангенс | TANH | 1 | tanh(a) | Real | Real |
| Квадратный корень | SQRT | 1 | (a)\*\*1/2 | Real | Real |
| DSQRT | 1 | Double | Double |
| Арктангенс | ATAN | 1 | arctan(a) | Real | Real |
| DATAN | 1 | Double | Double |
| ATAN2 | 2 | arctan(a1/a2) | Real | Real |
| DATAN2 | 2 | Double | Double |
| Взятие остатка | DMOD | 2 | a1(mod a2) | Double | Double |

## 9.4 Подпрограммы функции

Модуль программы, который начинается с оператора FUNCTION, называется подпрограммой-функцией.

Оператор FUNCTION имеет следующую форму:

**t FUNCTION f(a1,a2,...an)**

или

**FUNCTION f(a1,a2,...an)**

где:

|  |  |  |
| --- | --- | --- |
| **t** | - | указатель типа может быть INTEGER, REAL, DOUBLE PRECISION или LOGICAL или отсутствовать, как показано во второй форме. |
| **f** | - | имя подпрограммы-функции. |
| **ai** | - | фиктивные параметры, из которых должен быть представлен, по крайней мере, один, и которые являются именами переменных, массивов или фиктивными именами SUBROUTINE или других подпрограмм FUNCTION. |

## 9.5 Конструкция подпрограмм-функций

Конструкция подпрограмм-функций должна соответствовать следующим ограничениям:

1. Оператор FUNCTION должен быть первым оператором модуля программы.
2. В подпрограмме-функции имя функции должно появиться, по крайней мере, один раз с левой стороны знака равенства оператора присваивания или как элемент во входном списке входного оператора. Эти определения значения функции, которое может быть возвращено в программу вызова.

Дополнительные значения могут быть возвращены в программу вызова через присвоение значений фиктивным параметрам.

**Пример:**

FUNCTION Z7(A,B,C)

.

.

Z7 = 5.\*(A-B) + SQRT(C)

.

.

C ПЕРЕОПРЕДЕЛЯЕТ АРГУМЕНТ

B=B+Z7

.

.

RETURN

END

1. Имена из списка фиктивных параметров не должны появляться в операторах EQUIVALENCE, COMMON или DATA внутри подпрограммы-функции.
2. Если фиктивный параметр - имя массива, то в подпрограмме-функции должен появиться оператор объявления массива с определением размеров, непротиворечащим размерам массива в программе вызова.
3. Подпрограмма-функция может содержать любые операторы, определенные в FORTRAN кроме операторов BLOCKDATA, SUBROUTINE, другого оператора FUNCTION или любого другого оператора который непосредственно или косвенно обращается к ней самой.
4. Логическим завершением подпрограммы-функции является оператор RETURN, который, по крайней мере, один раз должен присутствовать в подпрограмме.
5. Подпрограмма-функция должна физически заканчиваться оператором END.

**Пример:**

FUNCTION SUM (BARY, I, J)

DIMENSION BARY(10,20)

SUM =0.0

DO 8 K = 1, I

DO 8 M = 1, J

8 SUM = SUM + BARY(K,M)

RETURN

END

## 9.6 Ссылки на подпрограмму-функцию

Подпрограмма-функция вызывается каждый раз, когда имя функции, сопровождаемое списком параметров, используется в качестве операнда в выражении. Такие ссылки принимают следующую форму:

**f(al,a2,...,an)**

где f является именем функции и ai фактические параметры. Скобки должны присутствовать, как показано в форме.

Параметры ai должны быть согласованы по типу, порядку следования и числу с фиктивными параметрами в операторе FUNCTION вызываемой подпрограммы. Параметры не имеют преобразований типа. Должен присутствовать крайней мере один параметр. Параметры могут быть любыми из перечисленных:

1. Имя переменной.

2. Имя элемента массива.

3. Имя массива.

4. Выражение.

5. Имя подпрограммы SUBROUTINE или FUNCTION

6. Текстовая или литеральная константа.

Если ai имя подпрограммы, то это имя должно иметь отличие от обычных переменных, появившись ранее в операторе EXTERNAL, и должно соответствовать фиктивному параметру ссылке на подпрограмму в вызываемой подпрограмме-функции.

Если ai текстовая или литеральная константа, то соответствующая фиктивная переменная должна обеспечивать достаточно места необходимого для размещения константы.

При вызове подпрограммы-функции выполнение программы переходит к первому выполнимому оператору после оператора FUNCTION.

Следующие примеры показывают ссылки на подпрограммы-функции.

Z10 = FT1+Z7(D, T3, RHO)

DIMENSION DAT(5,5)

.

.

SI = TOT1 + SUM(DAT,5,5)

## 9.7 Подпрограммы SUBROUTINE

Модуль программы, который начинается с оператора SUBROUTINE, называют подпрограммой SUBROUTINE. Оператор SUBROUTINE имеет одну из следующих форм:

**SUBROUTINE s (a1,a2,...,an)**

или

**SUBROUTINE s**

где **s** - имя подпрограммы SUBROUTINE, и каждый **ai** - фиктивный параметр, который представляет имя переменной, массива, имя другой SUBROUTINE или имя подпрограммы-функции.

## 9.8 Конструкция подпрограмм SUBROUTINE

1. Оператор SUBROUTINE должен быть первым оператором модуля программы.
2. Имя подпрограммы SUBROUTINE не должно появляться ни в каком операторе, кроме начального оператора SUBROUTINE.
3. Имена из списка фиктивных параметров не должны появляться в операторах EQUIVALENCE, COMMON или DATA внутри подпрограммы SUBROUTINE.
4. Если фиктивный параметр - имя массива, то в подпрограмме-функции должен появиться оператор объявления массива с определением размеров, непротиворечащим размерам массива в программе вызова.
5. Если какой-либо из фиктивных параметров представляет значения, которые должны быть определены подпрограммой SUBROUTINE и возвращены к программе вызова, эти фиктивные параметры должны появиться в подпрограмме c левой стороне от знака равенства в операторе присваивания, во входном списке входного оператора или в качестве параметра в ссылке на подпрограмму.
6. Подпрограмма SUBROUTINE может содержать любые операторы FORTRAN кроме операторов BLOCK DATA, операторов FUNCTION, другого оператора SUBROUTINE или любого другого оператора который непосредственно или косвенно обращается к ней самой.
7. Подпрограмма SUBROUTINE может содержать любое число операторов RETURN, который, по крайней мере, один раз должен присутствовать в подпрограмме.
8. Оператор RETURN является логическим окончанием подпрограммы
9. Физическим окончанием подпрограммы SUBROUTINE служит оператор END.
10. Если фактический параметр, передает в подпрограмму SUBROUTINE из вызывающей программы имя подпрограммы SUBROUTINE или FUNCTION, соответствующий фиктивный параметр должен использоваться в вызванной подпрограмме SUBROUTINE в качестве ссылки на подпрограмму.

**Пример:**

C ПОДПРОГРАММА СЧИТАЕТ ПОЛОЖИТЕЛЬНЫЕ ЭЛЕМЕНТЫ В МАССИВЕ

C

SUBROUTINE COUNT P(ARRY,I,CNT)

DIMENSION ARRY(7)

CNT = 0

DO 9 J=1,I

IF(ARRY(J)) 9,5,5

9 CONTINUE

RETURN

5 CNT = CNT+1.0

GO TO 9

END

## 9.9 Ссылки на подпрограмму SUBROUTINE

Подпрограмму SUBROUTINE можно вызвать при помощи оператора CALL. У оператора CALL есть одна из следующих форм:

**CALL s(a1,a2,...,an)**

или

**CALL s**

где **s** - имя подпрограммы SUBROUTINE, и **ai** фактические параметры, которые будут использоваться подпрограммой. ai должен быть согласован по типу, порядку следования и числу с соответствующими фиктивными параметрами в определяющем подпрограмму операторе SUBROUTINE.

Параметры в операторе CALL должны соответствовать следующим правилам:

1. Имена FUNCTION и SUBROUTINE, появляющиеся в списке параметров должны быть ранее описаны в операторе EXTERNAL.
2. Если вызываемая подпрограмма SUBROUTINE содержит переменную объявления массива, то оператор CALL в списке параметров должен содержать подлинное имя массива и фактические спецификации размерности массива.
3. Если элемент в фиктивном списке параметров подпрограммы SUBROUTINE - массив, соответствующий элемент в списке параметров оператора CALL должен быть массивом.

При вызове подпрограмму SUBROUTINE программа передает управление первому выполнимому оператору после оператора SUBROUTINE.

**Пример:**

DIMENSION DATA(10)

.

.

C ОПЕРАТОР НИЖЕ ВЫЗЫВАЕТ ПОДПРОГРАММУ ВПРЕДЫДУЩЕМ ПАРАГРАФЕ

C

CALL COUNTP(DATA,10,CPOS)

## 9.10 Возврат из подпрограмм FUNCTION и SUBROUTINE

Логическое завершение подпрограммы FUNCTION или SUBROUTINE - оператор RETURN, который возвращает управление к программе вызова. Общая форма оператора возврата - просто слово

**RETURN**

Следующие правила регулируют использование оператора RETURN:

1. Должен быть, по крайней мере, один оператор RETURN в каждой подпрограмме SUBROUTINE или FUNCTION.
2. Возврат из подпрограммы-функции является последовательность команд программы вызова следующая за ссылкой на функцию.
3. Возврат из подпрограммы SUBROUTINE происходит к следующему выполняемому оператору в программы вызова, которая логически следует после оператора CALL.
4. По возвращении из подпрограммы-функции единственный результат подпрограммы доступен для вычисления выражения, из которого был сделан вызов функции.
5. По возвращении из подпрограммы SUBROUTINE значения, присвоенные аргументам в подпрограмме доступны для использования в вызывающей программе.

**Пример:**

|  |  |
| --- | --- |
| Вызывающая программа  .  .  CALL SUBR(Z9,B7,R1)  .  . | Вызываемая программа  SUBROUTINE SUBR(A,B,C)  READ(3,7) B  A = B\*\*C  RETURN  7 FORMAT(F9.2)  END |

В этом примере, Z9 и B7 будут доступны в вызывающей программе, когда происходит возврат.

## 9.11 Обработка массивов в подпрограммах

Если программа вызова передает имя массива в подпрограмму, подпрограмма должна содержать информацию о подходящей размерности массива. Подпрограмма должна содержать операторы объявления массива, если любой из ее формальных параметров представляют собой массивы или элементы массива.

Например, подпрограмма-функция, предназначенная для вычисления среднего значения элементов любого одномерного массива, может быть следующей:

|  |  |
| --- | --- |
| Вызывающая программа  DIMENSION Z1(50), Z2(25)  .  .  Al = AVG(Z1,50)  .  .  A2 = A1-AVG(Z2,25)  .  . | Вызываемая программа  FUNCTION AVG(ARG, I)  DIMENSION ARG(50)  SUM =0.0  DO 20 J=1,I  20 SUM - SUM + ARG(J)  AVG = SUM/FLOAT(I)  RETURN  END |

Обратите внимание на то, что фактические размеры массивов, которые будут обработаны подпрограммой-функцией, определены в программе вызова и имена массивов, и их фактические размерности переданы подпрограмме-функции ссылкой. Сама подпрограмма-функция содержит фиктивный массив и определяет размер в операторе объявления массива.

Информация о размерах может также быть передана подпрограмме в списке параметров. Например:

|  |  |
| --- | --- |
| Вызывающая программа  DIMENSION A(3,4,5)  .  .  CALL SUBR(A,3,4,5)  .  .  END | Вызываемая программа  SUBROUTINE SUBR(X,I,J,K)  DIMENSION X(I, J, K)  .  .  RETURN  END |

Использование переменных с информацией о размерностях допустимо, только если имя массива и все переменные с информацией о размерностях являются фиктивными параметрами. Переменные с информацией о размерностях должны быть числами целого типа. Недопустимо изменить значения любой из переменных с информацией о размерностях в вызванной программе.

## 9.12 Подпрограмма BLOCK DATA

Подпрограмма BLOCK DATA выполняет единственную задачу инициализации данных в COMMON блоке во время загрузки объектной программы FORTRAN. Подпрограммы BLOCK DATA начинаются с оператора BLOCK DATA следующей формы:

**BLOCK DATA [subprogram-name]**

и заканчиваются оператором END. Такие подпрограммы могут содержать только операторы описания типа, операторы EQUIVALENCE, DATA, COMMON и DIMENSION и должны отвечать следующим требованиям:

1. Если любой элемент в COMMON блоке должен быть инициализирован, все элементы блока должны быть перечислены в операторе COMMON, даже если они не могут быть все инициализированы.
2. Инициализация данных в нескольких COMMON блоках может быть выполнена в одной подпрограмме BLOCK DATA.
3. Несколько подпрограмм BLOCK DATA может быть использовано в любой момент времени.
4. Любой определенный элемент COMMON блока должен быть инициализирован только одним модулем программы.

**Пример:**

BLOCK DATA

LOGICAL Al

COMMON/BETA/B(3,3)/GAM/C(4)

COMMON/ALPHA/A1,F,E,D

DATA B/l.1,2.5,3.8,3\*4.96,

1 2\*0.52,1.1/,C/1.2E0,3\*4.0/

DATA Al/.TRUE./,E/-5.6/

## 9.13 Цепочки программ

Программы могут быть загружены и выполнены (последовательно) с помощью программы FORTRAN путем вызова FCHAIN.

Общий синтаксис:

**CALL FCHAIN ('filename')**

где filename является допустимым именем исполняемого файла в зависимости от используемой операционной системы. Точный синтаксис меняется в разных операционных системах. См. [**Раздел 3** Microsoft FORTRAN-80 Руководство пользователя](#_Раздел_3_FORTRAN-80_1).

**Правила:**

1. 'filename' должно быть допустимым в соответствии с требованиями вашей операционной системы.
2. Прицепленная программа должна быть программой "MAIN". То есть, иметь точку входа. Подпрограммы FORTRAN, COBOL и на языке ассемблера не содержат точку входа «MAIN».
3. Параметры не могут быть переданы прицепленной программе.
4. Недопустимое имя файла, неверное определение диска, отсутствие файла, нехватка памяти и ошибки чтения диска приведут к неустранимой ошибке \*\*io\*\*.

# Приложение A Расширения и ограничения языка

Язык FORTRAN-80 включает в себя следующие расширения стандарта ANSI FORTRAN (x3.9-1966):

1. Если в операторах 'STOP c' или 'PAUSE c' используется поле с, оно может содержать любые шесть символов ASCII.
2. В операторах READ и WRITE могут быть указаны обработка ошибок и обнаружение конца файла с помощью параметров ERR= и END=.
3. Стандартные подпрограммы PEEK, POKE, INP и OUT были добавлены в библиотеку FORTRAN.
4. Оператор-функции могут использовать индексные переменные.
5. Шестнадцатеричные константы могут использоваться везде, где, как правило, допустимы целые константы.
6. Литеральная форма текстовых данных (символьная строка между символами апострофа) разрешена вместо стандартной формы nH.
7. Текстовые и литеральные данные разрешены в выражениях вместо целых констант.
8. Отсутствует ограничение на число строк продолжения.
9. Допустим смешанный режим выражений и присвоений, и преобразования выполняются автоматически.

Язык FORTRAN-80 имеет следующие ограничения по отношению к стандарту FORTRAN.

1. Тип данных COMPLEX не реализован.
2. Операторы описания должны появиться в следующем порядке:
   1. PROGRAM, SUBROUTINE, FUNCTION, BLOCK DATA;
   2. Операторы описания типа, EXTERNAL, DIMENSION;
   3. COMMON;
   4. EQUIVALENCE;
   5. DATA;
   6. Оператор-функции.
3. Различный размер памяти выделяется для каждого из типов данных: целых, вещественных, двойной точности, логических.

# Приложение B Интерфейс ввода-вывода

Операции ввода/вывода таблично-диспетчеризированы в подпрограмме драйвера для надлежащего логического номера устройства (LUN). $LUNTB является таблицей переходов. Она содержит один 2-байтовый адрес драйвера для каждого возможного LUN. У нее вначале также есть однобайтовая запись, которая содержит максимальное значение LUN+1.

Первоначальный пакет времени выполнения предусматривает 10 LUN (1 - 10). Устройства 1, 3, 4, и 5 предварительно назначены для консоли (TTY). Устройство 2 предварительно назначено линейному принтеру. Устройства 6-10 предварительно назначены для файлов на диске (см. Руководство пользователя, [**Раздел 3**](#_Раздел_3_FORTRAN-80)).

Любое из них может быть переопределено пользователем простым изменением соответствующих записей в $LUNTB. Исполняющая система использует LUN 3 для вывода ошибок и других сообщений пользователя. Поэтому, логическое устройство 3 должно соответствовать консоли оператора. Исходная структура $LUNTB показана в последующих листингах этого приложения.

Также возможно добавить LUN в $LUNTB. Если Вы делаете это, измените байт MAXLUN+1 в ячейке $LUNTB и удостоверьтесь, что также изменяете значение MAXLUN в модуле DSKDRV.MAC.

Драйверы устройств также содержат местные таблицы переходов. Обратите внимание, что $LUNTB содержит один адрес для каждого устройства, но на самом деле есть семь возможных операций c устройством:

1) Форматное чтение

2) Форматная запись

3) Неформатное чтение

4) Неформатная запись

5) Rewind (перемотка)

6) Backspace

7) Endfile (конец файла)

Каждый драйвер устройства содержит до семи подпрограмм. Исходные адреса каждой из этих семи подпрограмм размещены в начале драйвера, в порядке, перечисленном выше. Вход в $LUNTB затем указывает на эту локальную таблицу, и система выполнения по индексу получает в ней соответствующий адрес для выполнения поставленной операции ввода/вывода.

Применяются следующие правила к отдельным подпрограмм ввода/вывода:

1. Расположение $BF содержит адрес буфера данных для READ и WRITE.
2. Для WRITE, число байтов, подлежащих записи находится в расположения $BL.
3. Для READ число прочитанных байтов должно быть возвращено в $BL.
4. Перед выходом все операции ввода/вывода устанавливают коды условия, указывающие состояние ошибки, условие конца файла или нормальный возврат:

|  |  |  |
| --- | --- | --- |
| CY=1, | Z=не важно | - ошибка ввода-вывода |
| CY=0, | Z=0 | - обнаружен конец файла |
| CY=0, | Z=l | - нормальное возвращение |

Исполняющая система проверяет коды состояния после вызова драйвера. Если они указывают на ненормальное состояние, управление передается на метки, указанные в параметрах "ERR=" или "END=" или, если метка не указана, произойдет неустранимая ошибка.

1. $IOERR является обшей процедурой, которая печатает сообщение "ILLEGAL I/O OPERATION" (нефатальной). Эта процедура может быть использована, если некоторые операции не могут быть выполнены на определенном устройстве (т.е. Неформатный ввода/вывода на TTY).

**Примечание**

Буфер ввода/вывода имеет фиксированную максимальную длину 132 байта. Если драйвер позволит операциям ввода писать мимо конца буфера, могут быть затронуты важные переменные времени выполнения. Последствия могут быть непредсказуемыми.

Листинг, приведенный в данном приложении, содержит пример драйвера для TTY.

Операции REWIND, BACKSPACE и ENDFILE реализованы как No-Op (нет операций) и неформатные операции ввода/вывода рассматриваются как ошибка. Это драйвер TTY, поставляемый с пакетом времени выполнения.

MAC80 1.0 PAGE 1

00100 ; TTY I/O DRIVER

00200

0000 00300 EXT $IOERR,$BL,$BF,$ERR,$TTYIN,$TTYOT

0012 00400 IRECER EQU 022 ;INPUT RECORD TOO LONG

0000 00500 ENTRY $DRV3

0000 0013 ' 00600 $DRV3: DW DRV3FR ;FORMATTED READ

0002 0042 ' 00700 DW DRV3FW ;FORMATTED WRITE

0004 0010 ' 00800 DW DRV36R ;BINARY READ

0006 0010 ' 00900 DW DRV3BW ;BINARY WRITE

0008 000E ' 01000 DW DRV3RE ;REWIND

000A 000E ' 01100 DW DRV3BA ;BACKSPACE

000C 000E ' 01200 DW DRV3EN ;ENDFILE

000E AF 01300 DRV3EN: XRA A ;THESE OPERATIONS ARE

01400 ;NO-OPS FOR TTY

000E 01500 DRV3RE EQU DRV3EN

000E 01600 DRV3BA EQU DRV3EN

000F C9 01700 RET

0010 C3 0000 \* 01800 DRV3BW: JMP $IOERR ;ILLEGAL OPERATIONS

01900 ;(PRINT ERROR AND RETURN)

0010 02000 DRV3BR EQU DRV3BW

0013 AF 02100 DRV3FR: XRA A ;READ

0014 32 0000 \* 02200 STA $BL ;ZERO BUFFER LENGTH

0017 CD 0000 \* 02300 DRV31: CALL $TTYIN ;INPUT A CHAR

001A E6 7F 02400 ANI 0177 ;AND OFF PARITY

001C FE 0A 02500 CPI 10 ;IGNORE LINE FEEDS

001E CA 0017 ' 02600 JZ DRV31

0021 F5 02700 PUSH PSW ;SAVE IT

0022 2A 0015 \* 02800 LHLD $BL ;GET CHAR POSIT IN BUFFER

0025 26 00 02900 MVI H,0 ;ONLY 1 BYTE

0027 EB 03000 XCHG

0028 2A 0000 \* 03100 LHLD $BF ;GET BUFFER ADDR

002B 19 03200 DAD D ;ADD OFFSET

002C F1 03300 POP PSW ;GET CHAR

002D 77 03400 MOV M,A ;PUT IT IN BUFFER

002E 13 03500 INX D ;INCREMENT $BL

002F EB 03600 XCHG

0030 22 0023 \* 03700 SHLD $BL ;SAVE IT

0033 FE 0D 03800 CPI 015 ;CR?

0035 C8 03900 RZ ;YES—DONE

0036 7D 04000 MOV A.L ;$BL

0037 FE 80 04100 CPI 128 ;MAX IS DECIMAL 128

0039 DA 0017 ' 04200 JC DRV31 ;GET NEXT CHAR

003C CD 0000 \* 04300 CALL $ERR

003F 12 04400 DB IRECER ;INPUT RECORD TOO LONG

0040 AF 04500 XRA A ;CLEAR FLAGS

0041 C9 04600 RET

0042 3A 0031 \* 04700 DRV3FW: LDA $BL ;BUFFER LENGTH

0045 B7 04800 ORA A

MAC80 1.0 PAGE 2

0046 C8 04900 RZ ;EMPTY BUFFER

0047 2A 0029 \* 05000 LHLD $BF ;BUFFER ADDRESS

004A 3D 05100 DCR A ;DECREMENT LENGTH

004B F5 05200 PUSH PSW ;SAVE IT

004C 3E 0D 05300 MVI A,13 ;CR

004E CD 0000 \* 05400 CALL $TTYOT ;OUTPUT IT

0051 7E 05500 MOV A,M ;GET FIRST CHAR IN BUFFER

0052 FE 2B 05600 CPI '+'

0054 CA 0079 ' 05700 JZ DR3FW2 ;NO LINE FEEDS

0057 FE 31 05800 CPI '1'

0059 C2 0064 ' 05900 JNZ DR3FW1 ;NOT FCRM FEED

005C 3E 0C 06000 MVI A,12 ;FORM FEED

005E CD 004F \* 06100 CALL $TTYDT ;OUTFUT IT

0061 C3 0079 ' 06200 JMP DR3FW2

0064 3E 0A 06300 DR3FW1: MVI A,10 ;LF

0066 CD 005F \* 06400 CALL $TTYOT

0069 7E 06500 MOV A,M ;GET CHAR BACK

006A FE 20 06600 CPI ' '

006C CA 0079 ' 06700 JZ DR3FW2 ;NO MORE LINE FEEDS

006F FE 30 06800 CPI '0'

0071 C2 0079 ' 06900 JNZ DR3FW2 ;NO MORE LINE FEEDS

0074 3E 0A 07000 MVI A,10 ;LF

0076 CD 0067 \* 07100 CALL $TTYOT

0079 F1 07200 DR3FW2: POP PSW ;GET LENGTH BACK

007A 23 07300 INX H ;INCREMENT PTR

007B C8 07400 DRV32: RZ

007C F5 07500 PUSH PSW ;SAVE CHAR COUNT

007D 7E 07600 MOV A,M ;GET NEXT CHAR

007E 23 07700 INX H ;INCREMENT PTR

007F CD 0077 \* 07800 CALL $TTYOT ;OUTPUT CHAR

0082 F1 07900 POP PSW ;GET COUNT

0083 3D 08000 DCR A ;DECREMENT IT

0084 C3 007B ' 08100 JMP DRV32 ;ONE MORE TIME

0087 08200 END

MAC80 1.0 PAGE 3

$IOERR 0011\* $BL 0043\* $BF 0048\* $ERR 003D\*

$TTYIN 0018\* $TTSOT 0080\* IRECER 0012 $DRV3 0000'

DRV3FR 0013' DRV3FW 0042' DRV3BR 0010' DRV3BW 0010'

DRV3RE 000E\* DRV3BA 000E' DRV3EN 000E' DRV31 0017'

DR3FW2 0079' DR3FW1 0064' DRV32 007B'

MAC80 1.0 PAGE 2

00100 ;COMMENT \*

00200 ; DRIVER ADDRESSES FOR LUN'S 1 THROUGH 10

00210 ;

0001 00220 LPT EQU 1 ;UNIT 2 IS LPT

0001 00230 DSK EQU 1 ;UNITS 6-10 ARE DSK

0000 00235 DTC EQU 0 ;DTC COMMUNICATIONS UNIT4

00240 ;

00300

0000 00400 ENTRY $LUNTB

0000 00500 EXT $DRV3

0000 0B 00600 $LUNTB: DB 013 ;MAX LUN + 1

0001 0000 \* 00700 DW $DRV3 ;THEY ALL POINT TO $DRV3

;FOR NOW

0003 00800 IFF LPT

00900 DW $DRV3

0003 01000 ENDIF

0003 01100 IFT LPT

0003 01200 EXT LPTDRV

0003 0000 \* 01300 DW LPTDRV

0005 01400 ENDIF

0005 0001 \* 01500 DW $DRV3

0007 01510 IFF DTC

0007 0005 \* 01600 DW $DRV3

0009 01602 ENDIF

0009 01604 IFT DTC

01605 EXT $CMDRV

01606 DW $CMDRV

0009 01608 ENDIF

0009 0007 \* 01700 IFF DSK

000B 01800 DW $DRV3

01900 DW $DRV3

02000 DW $DRV3

02100 DW $DRV3

02200 DW $DRV3

02300 DW $DRV3

000B 02400 ENDIF

000B 02500 IFT DSK

000B 02600 EXT DSKDRV

000B 0000 \* 02700 DW DSKDRV

000D 000B \* 02800 DW DSKDRV

000F 000D \* 02900 DW DSKDRV

0011 000F \* 03000 DW DSKDRV

0013 0011 \* 03100 DW DSKDRV

0015 03200 ENDIF

0015 03300 END

MAC80 1.0 PAGE 2

LPT 0001 DSK 0001 DTC 0000 $LUNTB 0000'

$DRV3 0009\* LPTDRV 0003\* DSKDRV 0013\*

# Приложение C Вызовы подпрограмм

Это приложение описывает нормальный вызов подпрограммы сгенерированный компилятором FORTRAN. Это описание служит для упрощения компоновки программам на языке FORTRAN с написанными на других языках, таких как ассемблер 8080.

Вызов подпрограммы без параметров генерирует простую инструкцию "CALL". Соответствующая подпрограмма должна возвратиться через простое, "RET". (CALL и RET являются кодами операций процессора i8080, см. Руководство по ассемблеру или справочное руководство 8080 для объяснений.)

Вызов подпрограммы с параметрами приводит к несколько более сложной вызывающей последовательности. Параметры всегда передаются ссылкой (т.е., фактически передается адрес младшего байта фактического параметра). Поэтому, параметры всегда занимают два байта каждый, независимо от типа.

Метод передачи параметров зависит от количества передаваемых параметров:

1. Если количество параметров меньше или равно 3, они передаются в регистрах. Параметр 1 будет в [HL], 2 в [DE] (если имеется), и 3 в [BC] (если имеется).
2. Если число параметров больше 3, они передаются следующим образом:
   1. Параметр 1 в регистре [HL].
   2. Параметр 2 в регистре [DE].
   3. Параметры 3 через n в непрерывном блоке данных. Регистр [BC] укажет на младший байт этого блока данных (т.е. на младший байт параметра 3).

Отметим, что в этой схеме, подпрограмма должна знать, какое число параметров ожидать, чтобы найти их. И наоборот, вызывающая программа отвечает за передачу правильного числа параметров. Ни при компиляции, ни при исполнении система не проверяет корректность числа параметров.

Если подпрограмма ожидает более 3 параметров и должна передать их в локальную область данных, существует системная подпрограмма, которая выполнит эту передачу.

Эту подпрограмму передачи параметров называют $AT и вызывают с информацией в регистре HL, указывающей на локальную область данных, в регистре [BC] указывающей на третий параметр и в регистре [A], содержащем число передаваемых параметров (т.е., общее количество параметров минус 2). Подпрограмма ответственна за сохранение первых двух параметров прежде, чем вызвать $AT. Например, если подпрограмма ожидает 5 параметров, она должна выглядеть так:

SUBR: LD (P1),HL ; Сохранение параметра 1

EX DE,HL

LD (P2),HL ; Сохранение параметра 2

LD A,3 ; Число параметров

LD HL,P3 ; Указатель на локальную область

CALL $AT ; Передача других 3 параметров

.

.

[Тело подпрограммы]

.

.

RET ; Возврат в место вызова

P1: DS 2 ; Место для параметра 1

P2: DS 2 ; Место для параметра 2

P3: DS 6 ; Место для параметров 3-5

При доступе к параметрам в подпрограмме не забывайте, что они являются указателями на фактически переданные параметры.

**Примечание**

Обязанность, следить за тем, что бы параметры в вызывающей программе соответствовали по числу, типу и длине параметрам, ожидаемым в подпрограмме, возложена на программиста. Это относится и к подпрограммам на FORTRAN, а также, написанным на ассемблере.

Функции FORTRAN ([**Раздел 9**](#_Раздел_9_Функции)) возвращают свои значения в регистрах или памяти в зависимости от типа. Логические результаты возвращаются в [A], целые в [HL]. Расширенные целые и вещественные числа и возвращают результаты в памяти в $AC. Вещественные двойной точность возвращают результаты в памяти в $DAC. $AC и $DAC - адреса младших байтов мантисс.

# Приложение D Коды символов ASCII

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Dec | Hex | CHR |  |  | Dec | Hex | CHR |  | Dec | Hex | CHR |
| 0 | 00H | NUL | ^@ |  | 43 | 2BH | + |  | 86 | 56H | V |
| 1 | 01H | SOH | ^A |  | 44 | 2CH | , |  | 87 | 57H | W |
| 2 | 02H | STX | ^B |  | 45 | 2DH | - |  | 88 | 58H | X |
| 3 | 03H | ETX | ^C |  | 46 | 2EH | . |  | 89 | 59H | Y |
| 4 | 04H | EOT | ^D |  | 47 | 2FH | / |  | 90 | 5AH | Z |
| 5 | 05H | ENO | ^E |  | 48 | 30H | 0 |  | 91 | 5BH | [ |
| 6 | 06H | ACK | ^F |  | 49 | 31H | 1 |  | 92 | 5CH | \ |
| 7 | 07H | BEL | ^G |  | 50 | 32H | 2 |  | 93 | 5DH | ] |
| 8 | 08H | BS | ^H |  | 51 | 33H | 3 |  | 94 | 5EH | ^ |
| 9 | 09H | HT | ^I |  | 52 | 34H | 4 |  | 95 | 5FH | \_ |
| 10 | 0AH | LF | ^J |  | 53 | 35H | 5 |  | 96 | 60H | ` |
| 11 | 0BH | VT | ^K |  | 54 | 36H | 6 |  | 97 | 61H | a |
| 12 | 0CH | FF | ^L |  | 55 | 37H | 7 |  | 98 | 62H | b |
| 13 | 0DH | CR | ^M |  | 56 | 38H | 8 |  | 99 | 63H | c |
| 14 | 0EH | SO | ^N |  | 57 | 39H | 9 |  | 100 | 64H | d |
| 15 | 0FH | SI | ^O |  | 58 | 3AH | : |  | 101 | 65H | e |
| 16 | 10H | DLE | ^P |  | 59 | 3BH | ; |  | 102 | 66H | f |
| 17 | 11H | DC1 | ^Q |  | 60 | 3CH | < |  | 103 | 67H | g |
| 18 | 12H | DC2 | ^R |  | 61 | 3DH | = |  | 104 | 68H | h |
| 19 | 13H | DC3 | ^S |  | 62 | 3EH | > |  | 105 | 69H | i |
| 20 | 14H | DC4 | ^T |  | 63 | 3FH | ? |  | 106 | 6AH | j |
| 21 | 15H | NAK | ^U |  | 64 | 40H | @ |  | 107 | 6BH | k |
| 22 | 16H | SYN | ^V |  | 65 | 41H | A |  | 108 | 6CH | l |
| 23 | 17H | ETB | ^W |  | 66 | 42H | B |  | 109 | 6DH | m |
| 24 | 18H | CAN | ^X |  | 67 | 43H | C |  | 110 | 6EH | n |
| 25 | 19H | EM | ^Y |  | 68 | 44H | D |  | 111 | 6FH | o |
| 26 | 1AH | SUB | ^Z |  | 69 | 45H | E |  | 112 | 70H | p |
| 27 | 1BH | ESCAPE | ^[ |  | 70 | 46H | F |  | 113 | 71H | q |
| 28 | 1CH | FS | ^\ |  | 71 | 47H | G |  | 114 | 72H | r |
| 29 | 1DH | GS | ^] |  | 72 | 48H | H |  | 115 | 73H | s |
| 30 | 1EH | RS | ^^ |  | 73 | 49H | I |  | 116 | 74H | t |
| 31 | 1FH | US | ^\_ |  | 74 | 4AH | J |  | 117 | 75H | u |
| 32 | 20H | SPACE |  |  | 75 | 4BH | K |  | 118 | 76H | v |
| 33 | 21H | ! |  |  | 76 | 4CH | L |  | 119 | 77H | w |
| 34 | 22H | " |  |  | 77 | 4DH | M |  | 120 | 78H | x |
| 35 | 23H | # |  |  | 78 | 4EH | N |  | 121 | 79H | y |
| 36 | 24H | $ |  |  | 79 | 4FH | O |  | 122 | 7AH | z |
| 37 | 25H | % |  |  | 80 | 50H | P |  | 123 | 7BH | { |
| 38 | 26H | & |  |  | 81 | 51H | Q |  | 124 | 7CH | | |
| 39 | 27H | ' |  |  | 82 | 52H | R |  | 125 | 7DH | } |
| 40 | 28H | ( |  |  | 83 | 53H | S |  | 126 | 7EH | ~ |
| 41 | 29H | ) |  |  | 84 | 54H | T |  | 127 | 7FH | DEL |
| 42 | 2AH | \* |  |  | 85 | 55H | U |  |  |  |  |

Dec=десятичный, Hex=шестнадцатеричный (H), CHR= символ.

LF= перевод строки, FF= прогон страницы, CR= возврат каретки, DEL= стереть последний символ

# Приложение E Ссылки на библиотечные процедуры FORTRAN-80

Библиотека FORTRAN-80 содержит много процедур, на которые пользователь может сослаться из программ FORTRAN или ассемблера.

**1. Ссылки на арифметические процедуры**

В следующих описаниях $AC относится к вещественному аккумулятору; $AC - адрес младшего байта мантиссы. $AC+3 - адрес экспоненты. $DAC относится к аккумулятору DOUBLE PRECISION; $DAC - адрес младшего байта мантиссы. $DAC+7 - адрес экспоненты DOUBLE PRECISION.

Все арифметические процедуры (сложение, вычитание, умножение, деление и возведение в степень) придерживаются следующих соглашений о вызовах.

1. Параметр 1 передается в регистрах:
2. целое число в [HL],вещественное в $AC, двойной точности в $DAC
3. Параметр 2 передается в регистрах или в памяти в зависимости от типа:
   1. Целые числа передаются в [HL] или [DE], если [HL] содержит параметр 1.
   2. Значения вещественные или двойной точности передаются в памяти, на которую указывает [HL]. ([HL] указывает на младший байт мантиссы.)

Библиотека содержит следующие арифметические процедуры:

| **Функция** | **Имя** | **Тип 1-го параметра** | **Тип 2-го параметра** |
| --- | --- | --- | --- |
| Сложение | $AY | Integer\*4 | Integer |
| $A1 | Integer\*4 | Integer\*4 |
| $AA | Real | Integer |
| $AE | Real | Integer\*4 |
| $AB | Real | Real |
| $AQ | Double | Integer |
| $AV | Double | Integer\*4 |
| $AR | Double | Real |
| $AU | Double | Double |
| Деление | $D9 | Integer | Integer |
| $DY | Integer\*4 | Integer |
| $D1 | Integer\*4 | Integer\*4 |
| $DA | Real | Integer |
| $DE | Real | Integer\*4 |
| $DB | Real | Real |
| $DQ | Double | Integer |
| $DV | Double | Integer\*4 |
| $DR | Double | Real |
| $DU | Double | Double |
| Возведение в степень | $E9 | Integer | Integer |
| $EY | Integer\*4 | Integer |
| $E1 | Integer\*4 | Integer\*4 |
| $EA | Real | Integer |
| $EE | Real | Integer\*4 |
| $EB | Real | Real |
| $EQ | Double | Integer |
| $EV | Double | Integer\*4 |
| $ER | Double | Real |
| $EU | Double | Double |
| Умножение | $M9 | Integer | Integer |
| $MY | Integer\*4 | Integer |
| $M1 | Integer\*4 | Integer\*4 |
| $MA | Real | Integer |
| $ME | Real | Integer\*4 |
| $MB | Real | Real |
| $MQ | Double | Integer |
| $MV | Double | Integer\*4 |
| $MR | Double | Real |
| $MU | Double | Double |
| Вычитание | $SY | Integer\*4 | Integer |
| $S1 | Integer\*4 | Integer\*4 |
| $SA | Real | Integer |
| $SE | Real | Integer\*4 |
| $SB | Real | Real |
| $SQ | Double | Integer |
| $SV | Double | Integer\*4 |
| $SR | Double | Real |
| $SU | Double | Double |

Дополнительные библиотечные процедуры обеспечивают преобразования между типами значений. Параметры всегда передаются и возвращаются этими процедурами преобразования в следующих регистрах:

логические в [A],

целые в [HL],

расширенные целые в $AC,

вещественные в $AC,

двойной точности в $DAC.

|  |  |
| --- | --- |
| Имя | Функция |
| $CD | Integer в Integer\*4 |
| $CA | Integer в Real |
| $CC | Integer в Double |
| $C4 | Integer\*4 в Integer |
| $C5 | Integer\*4 в Real |
| $C6 | Integer\*4 в Logical |
| $C7 | Integer\*4 в Double |
| $CH | Real в Integer |
| $CL | Real в Integer\*4 |
| $CJ | Real в Logical |
| $CK | Real в Double |
| $CX | Double в Integer |
| $C0 | Double в Integer\*4 |
| $CY | Double в Real |
| $CZ | Double в Logical |

**2. Ссылки на встроенные функции**

Встроенные функции передают свои параметры в [HL] и [DE]. Если есть три параметра, [BC] содержит третий параметр. Если есть больше чем три параметра, то [BC] содержит указатель на блок памяти, который содержит оставшиеся параметры. Каждый из этих параметров является указателем на параметр. (См. [Приложение B.](#_Приложение_B_Интерфейс)),

Для функций MIN или MAX, число параметров передается в [A].

**Примечание**

Ни одна из функций (кроме INP и OUT) не может использовать представление переменной в байтах в качестве параметра. Представление переменной в байтах сначала должны быть преобразованы в тип, ожидаемый функцией. Иначе, результаты будут непредсказуемы.

**3. Процедуры для форматных READ и WRITE**

Оператор READ или WRITE вызывает одну из следующих процедур:

|  |  |
| --- | --- |
| $W2 (2 параметра) | инициализация передачи ввода/вывода |
| $W5 (5 параметров) | к устройству (WRITE) |
| $R2 (2 параметра) | инициализация передачи ввода/вывода |
| $R5 (5 параметров) | от устройства (READ) |

Эти процедуры придерживаются следующих соглашений о вызовах:

1. [HL] указывает на LUN
2. [DE] указывает на начало оператора FORMAT
3. Если процедура имеет пять параметров, то [BC] указывает на блок из трех   
   параметров:

1. адрес для ERR= переход

2. адрес для EOF= переход

3. адрес для REC= значение

Подпрограммы, которые передают значения в буфер ввода-вывода:

$I0 передают целые числа

$I1 передают вещественные числа

$I2 передают логические

$I3 передают числа двойной точности

$I4 передают расширенные целые числа (4 байта)

Подпрограммы передачи придерживаются следующих соглашений о вызовах:

1. [HL] указывает на место, которое содержит число переменных в списке
2. [DE] указывает на первое значение, которое будет передано
3. [BC] указывает на второе значение, которое будет передано, если есть только два значения, которые будут переданы в этом вызове. Если есть больше чем два значения, [BC] указывает на блок, который содержит указатели на второе через n значений.
4. Регистр A содержит число параметров (включая [HL]) сгенерированное этим вызовом.

Процедура $ND завершает процесс ввода/вывода.

**Пример:**

EXTRN $W2,$I0,$ND

ENTRY TEST

TEST: LXI H,LUN

LXI D,FORMAT

CALL $W2

LXI H,DIMENS

LXI D,NUMBER

MVI A, 2

CALL $IO

CALL $ND

RET

LUN: DW 1

FORMAT: DB '(11H РЕЗУЛЬТАТ=,15)'

DIMENS: DW 1

NUMBER: DW 9999

END TEST

**4. Загрузка и хранение вещественного аккумулятора**

В следующих определениях $AC относится к вещественному аккумулятору, и $DAC относится к аккумулятору двойной точности.

Для загрузки вещественного числа в аккумулятор ([HL] указывает на значение, которое будет загружено.):

|  |  |
| --- | --- |
| Имя | Функция |
| $L1 | Загрузка в $AC,4 байт |
| $L3 | Загрузка в $DAC, 8 байт |

Для сохранения вещественного числа из аккумулятора([HL] указывает на значение, которое будет сохранено.):

|  |  |
| --- | --- |
| Имя | Функция |
| $T1 | Сохранение 4 байт из $AC |
| $T3 | Сохранение 8 байт из $DAC |

**Предметный указатель**

А

Арифметические выражения

Правила вычисления 32

Преобразование типа 35

Форма 31

В

Ввод/вывод

Длина записи 18

Драйвер устройства 80

Имена файлов по умолчанию 16

Неявный цикл 55

Номер логического устройства 16, 80

Символ управления кареткой 67

Списки 55

Выражение

индексное 29, 32

операнд 31, 74

Реляционное 32

Д

Дескриптор преобразования данных 57

Разграничение форматных записей 65

Счетчик повторений 64

A (текстовых) 61

D (с плавающей точкой) 59

E (с плавающей точкой) 58

F (с плавающей точкой) 58

G (с плавающей точкой) 60

H (текстовых) 62

I (целых) 60

L (логических) 62

P (масштабный коэффициент) 63

X (пропуск/вставка пробелов) 63

И

Имя

Символическое 27

Системных переменных и подпрограмм 27

К

Компилятор

Ключи 9, 10, 18

Команда запуска 9

Команда запуска в TEKDOS 18

Ограничения языка 79

Расширения языка 79

Константы 25

в выражениях 34

Литеральные 34

Логические 32

Текстовые 34

Шестнадцатеричные 34

Л

Логические выражения 32

Порядок вычисления 34

Правила построения 34

Логические операции 33

М

Массив 25, 28

Индекс 28, 39

Правила использования 28

Обработка в подпрограммах 77

Объявление 28, 36, 37

Размещение в памяти 36

Элемент массива 25, 28

Н

Набор символов

ASCII 87

FORTRAN 21

буквенно-цифровые символы 22

буквы 21

специальные символы 22

цифры 22

О

Оператор

ASSIGN 45

BACKSPACE 54

BLOCK DATA 78

CALL 49, 75

параметры 76

COMMON 36, 38

CONTINUE 48

DATA 11, 24, 28, 29, 34, 36, 42

DECODE 54

DIMENSION 38

DO 46

индекс цикла 47

неявный 55

приращение цикла 47

расширение 47

ENCODE 54

END 49, 78

ENDFILE 54

EQUIVALENCE 36, 39

EXTERNAL 37, 76

FORMAT 11, 24, 56

обработка 66

FUNCTION 24, 72

GOTO

безусловный 44

вычисляемый 44

присваиваемый 44

IF арифметический 45

IF логический 46

IMPLICIT 36, 43

INCLUDE 24

LOGICAL 37

PAUSE 49

PROGRAM 69

READ 5

бесформатный 53

параметр "END=" 50

параметр "ERR=" 5, 50

форматный 50

RETURN 49

правила использования 76

REWIND 54

STOP 49

SUBROUTINE 74

WRITE

бесформатный 53

параметр"END=" 52

параметр"ERR=" 52

форматный 52

Операторы 21

Выполняемые 24, 48, 49

Инициализации данных 42

Метка 43, 45, 50

Невыполняемые 24, 38, 42, 56

Описания 36

Определения типа 37

Отношения 33

Порядок следования 79

Присваивания 35

Управления 43

Ошибки

Код ошибки 13

Неустранимые ошибки во время выполнения 15

Неустранимые ошибки при компиляции 13

Ошибки во время выполнения 14

Предупреждения при компиляции 14

П

Переменные 25, 27

Указание типа 28

INTEGER\*4 25

Подпрограмма

Метод передачи параметров 85

BLOCK DATA 78

FCHAIN 78

OPEN 16, 18, 54

OUT 70

POKE 70

RAN 3, 70

SUBROUTINE 74

Программа

Исходная 21

заключительная строка 23

начальная строка 23, 35

строка комментария 23

строка продолжения 23

Невыполнимая 36

Основная 21, 69

Размещение в ПЗУ 11

Соглашение о вызовах 11

Цепочки программ 78

Р

Реляционные выражения 33

С

Системная подпрограмма

$AT 85

$EX 49

$INIT 11

$IOERR 81

$LUNTB 11

Т

Тип

Иерархия 31

Неявное объявление 28, 37

Явное объявление 28

Тип данных

COMPLEX 21, 79

Ф

Файл

Имя 9

Исходный 9

Листинга 9

Объектный 9

Последовательного доступа 53, 54

Произвольного доступа 18

Расширение 9

DSKDRV.MAC 5

INIT.MAC 11

Фиктивные параметры 42

Функции

Библиотечные 70

внешние 72

встроенные 71

передача параметров 88

Оператор-функция 69

Подпрограмма-функция 72

C

COMMON блок

Имя 38, 39

Неименнованный 38

Расширение 39

**Справочное руководство**

**Пакет утилит**

**для микропроцессоров 8080**

**Microsoft, Inc.**

**Microsoft Building**

**10700 Northup Way**

**Bellevue, WA 98004**

Информация в этом документе может изменяться без предварительного уведомления и не представляет собой обязательств со стороны Microsoft, Inc. Программное обеспечение, описанное в этом документе оформлено под лицензионное соглашение или соглашение о неразглашении. Программное обеспечение может использоваться или копироваться только в соответствии с условиями соглашения. Противозаконно копировать Пакет утилит на кассетную ленту, диски или любой другой носитель с любой целью, кроме персонального использования покупателем.

Copyright © Microsoft, Inc., 1981

**Ограниченная гарантия**

Microsoft, Inc. не несет никакой ответственности перед покупателем или перед любым другим физическим или юридическим лицом в отношении любой ответственности, потери или повреждения, вызванных или предположительно вызванных прямо или косвенно этим продуктом, в том числе, но не ограничиваясь любым перерывом обслуживания, потерей бизнеса или ожидаемой прибыли или косвенного ущерба, возникших в результате использования или эксплуатации данного изделия. Этот продукт будет заменяться в течение двенадцати месяцев с даты покупки в случае обнаружения дефектов изготовления, маркировки или упаковки, но за исключением такой замены на продажу, или последующего использования этой программы без гарантии или ответственности.

**Вышеупомянутое - ограниченная и единственная гарантия, данная Microsoft, inc. любые и все гарантии коммерческие или пригодности для определенной цели, как правило, исключены.**

Чтобы сообщить об ошибках в программном обеспечении или ошибках в документации, пожалуйста, заполните и отправьте отчет о проблемах, находящийся в конце данного руководства.

CP/M является зарегистрированной торговой маркой Digital Research

Пакет утилит, MACRO-80, LINK-80, CREF-80 и LIB-80 являются зарегистрированной торговой маркой Microsoft, Inc.

8401-343-04

Содержание

# Глава 1 Введение

Добро пожаловать в мир программирования Пакета утилит. В ходе этого руководства, мы узнаем, что такое Пакет утилит, для чего его можно использовать, и как его использовать.

## 1.1 Содержимое Пакета утилит

Одна дискета со следующими файлами:

**M80.COM** - Программа макроассемблера MACRO-80

**L80.COM** - Программа связывающего загрузчика LINK-80

**CREF80.COM** - Средство перекрестных ссылок CREF80.COM

**LIB.COM** - Программа менеджера библиотек (только CP/M версия)

Одно руководство

Справочное руководство Пакет утилит

**ВАЖНО**

Всегда создавайте резервные копии своих дискет перед их использованием.

## 1.2 Системные требования

Программа MACRO-80 требует около 19 Кбайт памяти, плюс около 4 Кбайт для буферов. Программа LINK-80 требует около 14 Кбайт памяти. Программа CREF-80 требует около 6 Кбайт памяти. LIB-80 требует около 5 Кбайт памяти. Операционная система обычно требует около 6 Кбайт байт памяти. Таким образом, минимальные требования к системе для Пакета утилит – 29 Кбайт байт памяти. Хотя можно запустить программы пакета утилит только с одним диском, мы настоятельно рекомендуем иметь два доступных дисковых накопителя.

## 1.3 Для кого пакет утилит?

Пакет утилит является полной системой разработки на языке ассемблера с мощными функциями, которые поддерживают передовые навыки программирования на языке ассемблера. Это руководство подробно описывает Пакет утилит, но описание предполагает, что читатель понимает программирование на языке ассемблера и имеет опыт работы с ассемблером.

Если вы никогда не программировали на ассемблере, мы предлагаем вам получить определенный опыт на более простом ассемблере.

**Книги по программированию на языке ассемблера**

Мы можем также рекомендуем следующие книги для базового обучения программированию на ассемблере:

Leventhal, Lance A. 8080A/8085 Assembly Language Programming. Berkeley: Osborne/McGraw-Hill, 1978.

Leventhal, Lance A. Z80 Assembly Language Programming. Berkeley: Osborne/McGraw-Hill, 1979.

Zaks, Rodnay. Programming the Z80. Second edition. Berkeley: Sybex, 1980.

## 1.4 Об этом руководстве

**Структура**

Перед каждой главой приводится содержание, которое дает начальные ссылки на страницы с расширенным описанием в руководстве. Глава 1 содержит вводную информацию, сведения и обзор Пакета утилит. Главы 2-8 описывают использование и работу Пакета утилит. Руководство заканчивается несколькими приложениями, которые содержат некоторую полезную справочную информацию.

**Синтаксическая нотация**

В данном руководстве в описаниях синтаксиса команд и операторов используются следующие обозначения:

|  |  |
| --- | --- |
| [ ] | Квадратные скобки указывают, что вложенная запись не является обязательной. |
| < > | Угловые скобки указывают данные вводимые пользователем. Если угловые скобки включают текст в нижнем регистре, пользователь должен ввести определенный текст, например, <filename>. Если угловые скобки включают текст в верхнем регистре, пользователь должен нажать клавишу с указанным текстом, например, <RETURN>. |
| { } | Фигурные скобки указывают, что у пользователя есть выбор между двумя или больше записями. По крайней мере одна из записей, включенных в фигурные скобки, должна быть выбрана, если записи также не заключены в квадратные скобки. |
| ... | Многоточие указывает, что запись может быть повторена несколько раз, при необходимости. |
| CAPS | Заглавные буквы указывают части операторов или команд, которые должны быть введены, точно как показано. |

Вся другая пунктуация, такая как запятые, двоеточия, знаки наклонной черты, и знаки "равно", должны быть введены точно как показано.

## 1.5 Обзор

Пакет утилит является системой программирования на языке ассемблера, подобной по дизайну и стилю программированию ассемблерам и соответствующему программному обеспечению больших компьютеров. Следовательно, дизайн и использование Пакета утилит включает в себя черты и методы, которые могут быть для вас в новинку. Как указывалось выше, мы предполагаем, что у вас есть некоторый опыт в программировании на языке ассемблера.

Ваши знания, когда и зачем использовать определенные коды операций и псевдокоманды является той базой, на которой можно строить ваши знания о Пакете утилит.

Одно слово предостережения: некоторые термины, использованные в этом руководстве, могут быть знакомы вам из других источников. Обязательно обратите внимание, особенно, как знакомые термины используются в Пакете утилит, чтобы не сбивать вас с толку или не вводить в заблуждение.

Пакет утилит основывается на двух важных программах - ассемблере и связывающем загрузчике. Чтобы разработать программу на языке ассемблера, которая работает на вашем компьютере, вы должны использовать и ассемблер и связывающий загрузчик. Весь процесс схематически изображен на следующей странице. Числа на схеме соответствуют числам в объяснениях ниже.

1. Вы создаете исходную программу на языке ассемблера, используя некоторый редактор.
2. Вы компилируете свою исходную программу, используя макроассемблер MACRO-80. Результатом является файл содержащий промежуточный объектный код. Этот промежуточный код ближе к машинному коду, чем Ваш исходный код, но не может быть выполнен.
3. Вы соединяете и загружаете отдельно собранный файл(ы) в один программный файл, используя связывающий загрузчик LINK-80. LINK-80, преобразует файл(ы) промежуточного кода в один файл истинного машинного кода, который может быть выполнен в операционной системе.



Рисунок 1.1: Разработка программ на языке ассемблера

Это только основы всего процесса. Этот двухступенчатый процесс преобразования исходного файла в исполняемую программу позволяет управлять вашими программами, чтобы сэкономить время и расширить полезность своих программ следующими способами:

**Во-первых**, вы можете разбить вашу программу на удобные части, называемые модулями. Вы можете манипулировать этими модулями по желанию. Вы можете компилировать модули по отдельности. Вы исправляете только те, которые работают не правильно и перекомпилируете их. Это экономит ваше время.

**Во-вторых**, вы можете манипулировать размещением модулей в памяти, при соблюдении определенных ограничений, или разрешить LINK-80 размещать модули для вас.

**В-третьих**, Вы можете использовать собранные модули в других программах или в изменениях исходной программы, потому что нет никакого постоянного соединения между модулями. Это экономит вам время перекодирования, если часть программы выполняет несколько полезных, часто повторяющихся задач.

Всякий раз, когда вы хотите совместить скомпилированные модули в исполняемую программу, используйте программу связывающего загрузчика LINK-80. Если вы просто сообщите LINK-80 модули, которые хотите объединить, она загружает их от начала до конца в памяти. Но у вас есть дополнительный выбор. Вы можете установить прямую связь между оператором в одном модуле и оператором внутри другого модуля. Эти прямые соединения (или "ссылки") означает, что значения (как правило, адрес программы), в одном модуле могут быть использованы в другом модуле, именно в той точке, которая требуется.

Программа LINK-80 создает связи между модулями. Вы даете LINK-80 сигналы, необходимые для создания этих ссылок. Сигналы называются символами, в частности, EXTERNAL (внешние) символы и PUBLIC (общедоступные) символы. EXTERNAL символ сигнализирует LINK-80, что вы хотите, чтобы он соединил значение из другого модуля в эту точку в программе.

Значение, которое будет соединено определено символом PUBLIC, который является сигналом, который направляет LINK-80 к корректному модулю и строке оператора. Программа LINK-80 сначала соединяет значение PUBLIC символа с EXTERNAL символом, а затем продолжает загрузку модуля с EXTERNAL символом. На диаграмме ниже приводится этот процесс.



Рисунок 1.2: PUBLIC символ связанный с EXTERNAL модулем

**В-четвертых**, модули могут быть ассемблированы в различных режимах, даже в пределах одного модуля. Эти четыре режима следующие: абсолютный, перемещаемых данных, перемещаемого кода и перемещаемых COMMON. Абсолютный режим аналогичен коду, созданному с помощью самых простых ассемблеров. Код ассемблирован с фиксированными адресами в памяти. Остальные три режима очень разные и по этой причине вы можете разместить модули в любом месте памяти. Каждый из трех относительных режимов компилирует обособленный сегмент. Адреса в каждом сегменте являются относительными адресами. Это означает, что первому байту инструкции в сегменте присваивается относительный адрес 0, второму байту присваивается относительный адрес 1 и т.д.

Когда LINK-80 загружает модуль, он изменяет относительные адреса в сегментах на фиксированные адреса в памяти. Программа LINK-80 использует смещение относительных адресов от некоторого фиксированного адреса. Для первого загруженного модуля в операционной системе CP/M, этот адрес 103H. Таким образом, смещение относительных адресов в первом модуле начинается с 103H. Второй модуль загружается в конец первого, и смещение относительных адресов считаются от последнего адреса в первом модуле. Последующие модули загружаются (и смещаются) так же. Вы можете изменить начальный адрес по умолчанию для первого модуля во время сборки. После этого относительные адреса становятся смещением от фиксированного адреса, который Вы определяете.

Одно из следствий этого метода относительной адресации заключается в том, что оператор ORG выполняет очень разные действия. Для перемещаемых сегментов, оператор ORG задает смещение, а не фиксированный адрес (как делает большинство ассемблеров - ORG задает фиксированный адрес в абсолютном сегменте). Таким образом относительный сегмент с оператором ORG будет пропускать заданное количество адресов перед началом загрузки оставшейся части кода в этот сегмент.



Вы должны внимательно прочитать описание оператора ORG в Главе 4.

Способность манипулировать размещением модулей в памяти, с некоторыми ограничениями (см. Главу 6), происходит от возможности ассемблера присваивать относительные адреса вместо абсолютных. Эта способность манипулировать размещением модулей в памяти называется перемещаемость. Модули являются перемещаемыми. Промежуточный код производимый ассемблером для трех относительных сегментов называется перемещаемым кодом. Именно поэтому скомпилированные модули создаются с расширением .REL, и эти скомпилированные файлы называются REL файлами.

Каждый режим служит для различных целей. Абсолютный режим содержит код, который Вы хотите поместить в определенные адреса памяти. Каждый относительный режим загружается в память как отдельный сегмент. Сегмент перемещаемых данных содержит элементы данных и любой код, который может часто изменяться и должен быть помещен в ОЗУ. Сегмент перемещаемого кода содержит только код, который не изменится и поэтому подходит для ПЗУ и ППЗУ. Сегмент перемещаемого COMMON содержит элементы данных, которые могут быть совместно использованы несколькими модулями.

Исходные операторы в этих режимах приобретают свойства своего режима. Символы и выражения в операторах обрабатываются ассемблером в зависимости от режима, в котором они находятся, и тип данных и другие элементы, определяют символы или части выражений. Черты режима, приписанные символу или выражению, называют, соответственно, его режимом, т.е. символы или выражение являются абсолютными, перемещаемыми данными, перемещаемым кодом, или перемещаемым COMMON. Это понятие режима важно, потому что это источник как гибкости так и сложности. Если все исходные операторы ассемблированы в абсолютном режиме, у символов и выражений всегда абсолютные значения, и использование абсолютных символов и выражений не сложно.

Проблема с абсолютном режимом состоит в том, что перемещаемость возможна только с помощью более сложных и трудоемких методов. Абсолютный режим существенно снижает возможность повторного использования кода в новой программе.

Относительные режимы (данные, код, и COMMON) являются основой перемещаемости и, следовательно, гибкости управления модулями. Сложность состоит в том, что относительные символы и относительные выражения гораздо более трудно анализировать. Фактически, ассемблер должен пройти через исходные операторы дважды, чтобы скомпилировать модуль. Во время первого прохода ассемблер анализирует операторы и разворачивает операторы макро вызовов, вычисляет объем кода, который он генерирует и создает таблицу символов, в которой всем символам и макросам присваиваются значения. Во время второго прохода ассемблер заполняет символы и значения выражений из таблицы символов, разворачивает операторы макро вызовов и порождает промежуточный код в файл REL.

Когда REL файлы обрабатываются LINK-80, сегменты соединяются друг с другом и загружаются в фиксированные адреса памяти. Относительные адреса преобразуются в абсолютные адреса. Фиксированные адреса присваиваются перемещаемым сегментам в следующем порядке: перемещаемые COMMON и перемещаемые данные, потом перемещаемый код. Перемещаемые сегменты в CP/M по умолчанию загружаются перемещением начиная с адреса 103H. (Адреса 100H-102H используются для перехода к начальному адресу первой инструкции программы, которым обычно является первый адрес после областей COMMON и данных.)

Когда LINK-80 закончил соединение модулей вместе и назначение адресов, результат может быть сохранен в файл, исполняемый в операционной системе. Выполнение программы так же просто как ввод команды в операционной системе, поэтому, эти соединенные и загруженные файлы называются командными файлами.

Этот краткий обзор должен дать Вам общее представление о работе и процессах Пакета утилит. Краткие описания всех программ Пакета утилит приведены в следующей главе. Подробные описания даны в остальной части этого руководства.

Таким образом, информация, содержащаяся в данном обзоре будет повторяться более подробно в другом месте этого руководства.

Для оказания помощи в понимании информации в следующей главе и остальных частях этого руководства, следующая страница содержит расширенную версию диаграммы, приведенной в начале этого обзора. Расширенная диаграмма показывает отношения между всеми программами Пакета утилит.



Рисунок 1.5: Отношения между программами

# Глава 2 Функции пакета утилит

Пакет утилит является системой разработки на языке ассемблера, которая собирает перемещаемый код на двух языках ассемблера, поддерживает макросредства, условное ассемблирование, и содержит несколько полезных программ, которые расширяют программы разработки.

**Что такое Пакет утилит?**

Пакет утилит более чем ассемблер. Пакет утилит представляет собой серию связанных с ним инструментов программирования для:

компиляции исходного файла на языке ассемблера,

компоновки нескольких скомпилированных модулей в одну программу,

создания файлов библиотек подпрограмм (также скомпилированных модулей),

создания листинга перекрестных ссылок программных символов,

тестирования и отладки бинарной (исполняемый машинный код) программы.

Пакет утилит компании Microsoft предоставляет версии этих инструментов, которые делают пакет утилит чрезвычайно мощным и полезным в качестве инструментов программирования. Каждый инструмент в пакете утилит подробно описан в отдельной главе.

## 2.1 Два языка ассемблера

Ассемблер в пакете утилит поддерживает два языка ассемблера. Макроассемблер Microsoft MACRO-80 поддерживает мнемоники 8080 и Z80.

## 2.2 Перемещаемость

Программа MACRO-80 может производить модули перемещаемого кода. Кроме того, как и многие ассемблеры, ассемблер MACRO-80 может производить абсолютный код. Ключевым преимуществом перемещаемости является то, что программы могут быть собраны в виде модулей. Затем, с учетом определенных ограничений, описанных в Главе 6, модули могут быть расположены практически в любом месте памяти.

Перемещаемые модули также обеспечивают преимущество более простого кодирования и более быстрого тестирования, отладки и изменения. Кроме того, можно определить сегменты собранного кода, которые позже будет загружены в ОЗУ или в ПЗУ/ППЗУ.

Перемещаемость будет обсуждена далее в Разделе 3.2, **Символы**.

## 2.3 Макросредства

Ассемблер MACRO-80 поддерживают полный, стандарт макросредств Intel. Макросредства позволяет программисту писать блоки кода для набора инструкций которые используются наиболее часто. Необходимость перекодировки этих инструкций исключается.

Программист дает имя этому блоку кода, называемого макросом. Инструкции являются макроопределением. Каждый раз, когда необходим набор инструкций, вместо того, чтобы повторно кодировать их, программист просто "вызывает" макрос. MACRO-80 разворачивает макро-вызов, компилируя блок инструкций в программу автоматически. Вызова макроса также передает параметры ассемблеру для использования в процессе расширения макроса. Использование макросов уменьшает размер исходного модуля, потому что макроопределения хранятся в файлах на диске и подключаются в модуль только в случае необходимости во время компиляции.

Макросы могут быть вложены, т.е. макрос можно вызвать из другого макроса. Вложение макросов ограничено только памятью.

## 2.4 Условное ассемблирование

Программа MACRO-80 также поддерживает условное ассемблирование. Программист может определить условие, при котором части программы ассемблируются или не ассемблируются. Условное ассемблирование улучшено полным набором условных псевдокоманд, которые включают проверку прохода ассемблирования, определения символов и параметров макросов. Условные выражения могут быть вложены до 255 уровней.

## 2.5 Программные утилиты

Три утилиты обеспечивают дополнительную поддержку, необходимую для разработки мощных и полезных программ на языке ассемблера: LINK-80 связывающий загрузчик, LIB-80 менеджер библиотек и средство построения перекрестных ссылок CREF-80.

**LINK-80 связывающий загрузчик**

Связывающий загрузчик Microsoft LINK-80 используется для преобразования скомпилированного модуля (файл .REL) в исполняемый модуль (файл .COM). Файл REL не является исполняемым файлом.

LINK-80 также может быть использован для:

загрузки, связывания и запуска одного или нескольких модулей

загрузки перемещаемых программ в указанное пользователем место

загрузки областей программ и данных в отдельные участки памяти

При выполнении этих задач, LINK-80 урегулирует внешние ссылки между модулями (то есть, любая программа, которая вызывает внешнее значение, которое определено в другой программе или модуле, будут иметь внешние ссылки, удовлетворенные во время компиляции LINK-80), и сохраняет исполняемый объект (.COM) файл на диске, так что она может быть запущена из операционной системы.

Эти умение загрузки означает, что созданная программа может быть связана с пользовательской библиотекой, чтобы добавить подпрограммы на одном из языков высокого уровня из библиотеки времени выполнения. Собранные программы могут быть связаны с программами на языке COBOL-80 и FORTRAN-80, к примеру, а также с программами MACRO-80.

**Средство построения перекрестных ссылок CREF-80**

Средство построенияперекрестных ссылок CREF-80 обрабатывает файл перекрестных ссылок, сгенерированный MACRO-80. Результат - листинг перекрестных ссылок, который может помочь в отладке вашей программы.

**LIB-80 менеджер библиотек (Только версия CP/M)**

Программа LIB-80 разработана как менеджер библиотек времени исполнения для CP/M версий Пакета утилит. LIB-80 может также использоваться для создания Вашей собственной библиотеки подпрограмм на ассемблере.

LIB-80 создает библиотеки времени выполнения из программ на языке ассемблера, которые являются подпрограммами для программ на COBOL, FORTRAN и других программам на языке ассемблера. Программы, собранные LIB-80, могут быть специальными модулями, создаваемыми программистом, или модулями из существующих библиотек. С помощью LIB-80, Вы можете создать специализированные библиотеки времени выполнения для любых своих требований.

# Глава 3 Программирование с Пакетом утилит

В этой главе описывается то, что пользователь должен знать, чтобы создать исходные файлы макроассемблера MACRO-80. Исходные файлы создаются с помощью текстового редактора, например, CP/M ED. Пакет утилит не включает в себя текстовый редактор. Исходные файлы собираются с использованием процедур, описанных в Главе 4.

## 3.1 Структура исходного файла

**Структура файла**

Исходный файл макроассемблера MACRO-80 это группа строк, написанных на языке ассемблера. Последняя строка файла должна быть оператором END. Соответствующие операторы такие, как IF...ENDIF, должны быть введены в правильной последовательности. В остальном, строки могут появиться в любом порядке по замыслу программиста.

**Формат строки оператора**

Исходные файлы обрабатываемые макроассемблером MACRO-80 состоят из строк операторов, разделенных на части или "полей".

BUF: DS 100H ; create a buffer

символ оператор аргумент комментарий

Поле символ содержит один из трех типов символов (LABEL, PUBLIC и EXTERNAL), за которым следует двоеточие, если оно не является частью операторов SET, EQU или MACRO.

Поле оператор содержит код операции, макрокоманду, псевдокоманду или выражение.

Поле аргумент содержит выражение (конкретные значения, переменные, имена регистров, операнды и операторы).

Поле комментарий содержит текст комментария, которому всегда предшествует точка с запятой.

Все поля являются необязательными. Вы можете ввести абсолютно пустую строку.

Строки оператора могут начаться в любом столбце. Несколько пробелов или знаков табуляции могут быть вставлены между полями для улучшения удобочитаемости, но по крайней мере один пробел или знак табуляции требуются между каждым полем.

**Комментарии**

Исходные строки макроассемблера MACRO-80 в основном содержат операторы и их аргументы. Поэтому, макроассемблер MACRO-80 требует, чтобы комментарий всегда начался с точки с запятой. Комментарий заканчивается возвратом каретки.

Для длинных комментариев, вы можете использовать псевдокоманду .COMMENT во избежание ввода точки с запятой в каждой строке. Подробнее см. раздел Главы 4 связанный c псевдокомандой .COMMENT.

## 3.2 Символы

Символы просто имена для определенных функций или значений. Имена символов создаются и определяются программистом.

Символы в Пакете утилит принадлежат к одному из трех типов, в зависимости от их функций. Этими тремя типами являются LABEL, PUBLIC и EXTERNAL. У всех трех типов символов есть атрибут режима, который соответствует сегменту памяти к которому относится символ. Обратитесь к разделу **Режимы** после описания типов символов.

Все три типа символов имеют следующие характеристики:

1. Символы могут быть любой длины, но количество значащих символов, передаваемых в компоновщик зависит от типа символа:

1. Для меток, значимы только первые шестнадцать символов.

2. Для символов PUBLIC и EXTERNAL, только первые шесть символов передаются компоновщику.

Дополнительные символы обрезаются.

1. В именах допустимы следующие символы:

A-Z 0-9 $ . ? §

1. Символ не может начинаться с цифры или знака подчеркивания.

Когда символ читается, нижний регистр преобразуется в верхний регистр, поэтому вы можете вводить имя, используя любой регистр или оба.

**МЕТКА:**

LABEL: контрольная точка в программном модуле для операторов, у которых присутствует метка. LABEL: присваивает символу метки значение адреса следующих за ней данных. Например, в операторе:

BUF: DS 1000H

BUF: равняется первому адресу зарезервированного пространства из 1000H байт.

После того, как метка определена она может быть использована в поле аргумента. Оператор с меткой в качестве аргумента ссылается на строку оператора с этой меткой в поле символ в котором метка определяется. Определение метки заменяет метку, используемую в поле аргумента. Например,

STA BUF

отправляет значение в аккумуляторе в область в памяти, представленной меткой BUF.

Метка может быть любым допустимым именем символа, до 16 символов в длину.

Если Вы хотите определить метку, она должна быть первым элементом в строке оператора. Метки 8080 и Z80 должны сразу сопровождаться единственным двоеточием (без промежутка), если метка не является частью оператора SET или EQU. (Если введены два двоеточия, "метка" становится PUBLIC символом. Смотрите PUBLIC символы ниже.)

**PUBLIC**

PUBLIC символ определяется так же, как метка. Разница в том, что PUBLIC символ так же доступен в качестве контрольной точки в других программных модулях.

Символ объявлен PUBLIC c:

двумя двоеточиями (::) после имени. Например,

F00:: RET

одной из псевдокоманд PUBLIC, ENTRY или GLOBAL. Например,

PUBLIC FOO

См. Разделы **Определение данных** и **Определение символов** в Главе 4 для справки об использовании этих псевдокоманд.

Результат обоих способов объявления аналогичен. Таким образом,

FOO:: RET

эквивалентно

PUBLIC FOO

FOO: RET

**EXTERNAL**

EXTERNAL символ определяется вне программного модуля в котором он присутствует. EXTERNAL символ определяется как PUBLIC символ в другом, отдельном программном модуле. Во время компоновки (при использовании LINK-80), символу EXTERNAL присваивается значение символа PUBLIC из другого программного модуля. Например:

MOD1

FOO:: DB 7 ;PUBLIC FOO = 7

MOD2

BYTE EXT FOO ;EXTERNAL FOO

Во время компоновки, LINK-80 берет адрес PUBLIC FOO и использует значение (7) для EXTERNAL FOO.

Символ считается объявленным EXTERNAL с:

1. двумя знаками решетки (##) следующими за именем символа. Например:

CALL FOO##

объявляет FOO как двухбайтовый символ, определенный в другом программном модуле.

1. одной из псевдокоманд EXT, EXTRN или EXTERNAL для двухбайтовых значений. Например:

EXT FOO

объявляет FOO как двухбайтовое значение, определенное в другом программном модуле.

1. одной из псевдокоманд BYTE EXT, BYTE EXTERN или BYTE EXTERNAL для однобайтовых значений. Например:

BYTE EXT FOO

объявляет FOO как однобайтовое значение, определенное в другом программном модуле.

См. Раздел **Определение символов** в Главе 4 для информации об использовании этих псевдокоманд.

Как и для символов PUBLIC, результаты обоих методов декларирования те же. Таким образом,

CALL F00##

эквивалентно

EXT F00

CALL F00

**Режимы**

На символ ссылаются, вводя его имя в поле аргумент в строке оператора. Когда на символ ссылаются, значение символа (полученное на основании инструкции, которая определяет символ), заменяет имя символа и используется в работе.

Значение символа вычисляется в зависимости от режима программного счетчика (PC).

Режим программного счетчика определяет, будет ли секция программы загружена в память по адресам предопределенным программистом (абсолютный режим), либо в относительных адресах, которые изменяются в зависимости от размера и количества программ (режим перемещаемого кода) и объема данных (режим перемещаемых данных), или по адресам, совместно используемым с другими программными модулями (режим COMMON). По умолчанию используется режим перемещаемого кода.

**Абсолютный режим**: Абсолютный режим ассемблирует неперемещаемый код. Программист выбирает абсолютный режим, если блок кода программы должен быть загружен каждый раз в определенные адреса, независимо от того, что еще загружается одновременно с ним.

**Режим перемещаемых данных**: режим перемещаемых данных ассемблирует код для секции программы, которая может изменяться и поэтому должна быть загружена в ОЗУ. Особенно это относится к областям данных программы. Символы в режиме перемещаемых данных перемещаемы.

**Режим перемещаемого кода**: режим перемещаемого кода (программа) ассемблирует код для секций программ, которые не будут изменяться и поэтому могут быть загружены в ПЗУ/ППЗУ. Символы в режиме перемещаемого кода перемещаемы.

**Режим COMMON**: Режим COMMON ассемблирует код, который загружается в определенную область общих данных. Это позволяет программным модулям совместно использовать блок памяти и общие значения.

Чтобы изменить режим, используйте псевдокоманды режима программного счетчика в строке оператора. Псевдокоманды режима программного счетчика следующие:

ASEG абсолютный режим

DSEG режим перемещаемых данных

CSEG режим перемещаемого кода – режим по умолчанию

COMMON режим COMMON

Эти псевдокоманды подробно описаны в разделе **Режим программного счетчика** Главы 4.

Этот потенциал режима программного счетчика в макроассемблере MACRO-80 позволяет программисту разрабатывать программы на языке ассемблера, которые могут быть перемещены. Многие программисты ассемблера, возможно, учились всегда устанавливать оператор ORG в начале каждого модуля, подпрограммы или основной программы на языке ассемблера. В MACRO-80 этот режим адресации называется абсолютным режимом, потому что вначале специально указываются жесткие (или фактические адреса) с помощью оператора ORG.

MACRO-80 доступны два других, «перемещаемых» режима адресации, называемые режим перемещаемого кода (программа) и перемещаемых данных. Сегменты кода, написанные в этих двух режимах перемещаемы. Перемещаемость означает, что программный модуль может быть загружен начиная с любого адреса в доступной памяти, используя ключи /P и /D (специальные команды) в LINK-80.

## 3.3 КОДЫ ОПЕРАЦИЙ И ПСЕВДОКОМАНДЫ

Коды операций это мнемонические имена для машинных команд. Псевдокоманды это указания ассемблеру, а не микропроцессору.

MACRO-80 поддерживает два набора команд: 8080 и Z80. Перечень кодов операций с краткими резюме своих функций приведен в **Приложении F**. Для программирования с кодами операций на разных языках, пользователь должен сначала ввести псевдокоманду, которая говорит ассемблеру, какой язык в настоящее время используется. Смотрите раздел **Выбор набора инструкций** Главы 4.

MACRO-80 также поддерживает большое разнообразие псевдокоманд, которые позволяют ассемблеру выполнять множество различных функций. Псевдокоманды подробно описаны в **Главе 4** и приведены в **Приложении Е**.

Коды операций и псевдокоманды (обычно) вводятся в поле оператор строки оператора. (У строки оператора программы обычно есть запись в поле оператор, если только строка не является комментарием. Поле оператор будет первым заполненным полем, если отсутствует метка.) Оператор может быть любой мнемоникой 8080 или Z80, или псевдокомандой макроассемблера MACRO-80, вызовом макро или выражением.

Записи поля оператор обрабатываются в следующем порядке:

1. Вызов макро
2. Коды операций/псевдокоманды
3. Выражения

MACRO-80 сравнивает запись в поле оператор c внутренним списком имен макро. Если запись найдена, макрос расширяется. Если запись не макро, MACRO-80 пытается обработать запись в качестве кода операции. Если запись не является кодом операции, MACRO-80 пытается обработать запись как псевдокоманду. Если запись не является псевдокомандой, то MACRO-80 обрабатывает запись в качестве выражения.

Если выражение введено как строка оператора без кода операции, псевдокоманды, или имени макро перед ним, макроассемблер MACRO-80 не возвращает ошибку. Ассемблер предполагает, что в выражении пропущена псевдокоманда определения байта (DB) и ассемблирует строку.

Порядок обработки имени макро, совпадающего с кодом операции, не позволяет вам использовать код операции снова, за исключением вызова макро.

Например, если вы в программе присвоите блоку кода макроса имя ADD, вы не сможете использовать ADD как код операции в этой программе.

## 3.4 АРГУМЕНТЫ: ВЫРАЖЕНИЯ

Аргументы для кодов операций и псевдокоманд обычно называют выражениями, потому что они напоминают математические выражения, такие как 5+4\*3.

Части выражения называются операнды (5, 4 и 3 в математическом выражении) и операторы (+ и \* в качестве примера). Выражения могут содержать один или более операндов. Один операнд в выражении, вероятно, наиболее часто используемая форма аргумента.

Если выражение содержит более одного операнда, операнды связаны друг с другом оператором. Например:

5+4 6-3 7\*2 8/7 9>8

и так далее. В MACRO-80, операнды числовые значения, представленные числами, символами или кодами операций 8080. Операторы могут быть арифметическими или логическими.

Вы, вероятно, знакомы с различными формами выражений, которые могут быть использованы в качестве аргументов, но вы можете просмотреть данные, приведенные ниже для характеристик, уникальных для MACRO-80.

Следующие разделы определяют формы операндов и операторов поддерживаемых ассемблером MACRO-80.

### 3.4.1 Операнды

Операнды могут быть числами, символами или кодами операций 8080.

**Числа**

По умолчанию основание для чисел десятичное. Основание может быть изменено псевдокомандой .RADIX. Может быть выбрано любое основание от 2 (двоичный файл) до 16 (шестнадцатеричный). Если основание больше, чем 10, буквы A-F используются для цифр после 9. Если первая цифра числа не числовая, числу должен предшествовать нуль.

Число всегда оценивается в текущем основании, если не используется одна из следующих специальных нотаций:

nnnnB считается двоичным

nnnnD считается десятичным

nnnnO считается восьмеричным

nnnnQ считается восьмеричным

nnnnH считается шестнадцатеричным

X'nnnn' считается шестнадцатеричным

Числа интерпретируются как беззнаковые 16-битные константы. Переполнение числа вне двух байт (16 бит - т.е. 65535 десятичных чисел) игнорируется, и результатом считаются младшие разряды 16 бит.

**Строки ASCII**

Строка состоит из нуля или более символов, окруженных кавычками. Одиночные (') или двойные (") кавычки могут использоваться в качестве разделителей строк. Если в качестве аргумента вводится строка в кавычках, значения символов хранятся в памяти один за другим Например:

DB "ABC"

сохраняет значение ASCII A в первом адресе, B во втором адресе и С в третьем.

Кавычки разделителя могут использоваться в качестве символов, если они появляются дважды для вхождения каждого требуемого символа. Например, оператор

"I am ""great"" today"

сохраняет строку

I am "great" today

Если между кавычками отсутствуют символы, строка обрабатывается как пустая строка.

**Символьные константы**

Как строки, символьные константы составлены из нуля, одного или двух символов ASCII, разграниченных кавычками. В качестве разделителей могут использоваться одиночные или двойные кавычки. Кавычки разделителя могут использоваться в качестве символов, если они появляются дважды для вхождения каждого требуемого символа.

Различия заключаются в следующем:

1. Символьная константа - только ноль, один, или два символа.

2. Символы в кавычках являются символьной константой, только если выражение имеет более одного операнда. Если символы вводятся в качестве единственного операнда, они, обрабатываются и сохраняются в виде строки. Например:

'A'+l символьная константа, а

'A' это строка.

3. Значение символьной константы вычисляется, и в результате младший байт сохраняется в первом адресе и старший байт во втором. Например:

DW 'AB'+0

преобразует в 4142H и хранит 42 в первом адресе и 41 во втором.

Символьная константа, состоявшая из одного символа, имеет значение его ASCII символа. Т.е. старший байт значения равен нулю, и младший байт содержит значение символа ASCII. Например, значение константы 'A' - 41H.

Символьная константа состоящая из двух символов имеет в качестве своего значения значение ASCII первого символа в старшем байте и значение ASCII второго символа в младшем байте. Например, значение символьной константы 'AB'+0 равняется 41H\*256+42H+0.

Десятичные и шестнадцатеричные значения символов ASCII приведены в **Приложении C**.

**Символы в выражениях**

Символ может быть использован в качестве операнда в выражении. Символ вычисляется, и символ заменяется значением. Операция выполняется с использованием значения символа.

Преимущество использования символов в качестве операндов состоит в том, что программист не должен помнить его значение каждый раз когда это необходимо, проще использовать имя символа. Имя, как правило, легче запомнить, особенно если имя символа - мнемоническое. Использование символов в качестве операндов становится более привлекательным с ростом количества символов в программе.

Правила использования EXTERNAL в выражениях:

1. Символы EXTERNAL могут быть использованы в выражениях только со следующими операторами:

+ - \* / MOD HIGH LOW

1. Если EXTERNAL символ используется в выражении, результат выражения всегда внешний.

Правила режима, влияющие на символы в выражениях:

1. В любой операции, за исключением AND, OR, или XOR операнды могут быть в любом режиме.
2. Для AND, OR, XOR, SHL, и SHR, оба операнда должны быть абсолютными и внутренними.
3. Если выражение содержит абсолютный операнд и операнд в другом режиме, результат выражения будет в другом (не абсолютном) режиме.
4. При вычитании двух операндов в различных режимах результат будет в абсолютном режиме. Иначе, результат будет в режиме операндов.
5. При добавлении символов перемещаемых данных и перемещаемого код, результат будет неизвестен, и MACRO-80 передает выражение LINK-80, как неизвестное, которое урегулирует LINK-80.

**Текущий символ счетчика команд**

Необходимо отметить один дополнительный символ поля аргумент: текущий символ счетчика команд. Текущий счетчик команд это адрес следующей инструкции, которая будет ассемблирована. Текущий счетчик команд это удобная контрольная точка для вычисления новых адресов. Вместо того, чтобы помнить или вычислять текущий адрес программы, программист использует символ, который говорит ассемблеру использовать значение текущего адреса программы.

Текущий символ счетчика команд - $.

**Коды операций 8080 в качестве операндов**

Коды операций 8080 - допустимые однобайтовые операнды в только режиме 8080. Во время ассемблирования код операции переводится к его шестнадцатеричному значению.

Чтобы использовать коды операций 8080 в качестве операндов, сначала установите псевдокоманду .8080. Посмотрите раздел псевдокоманды Выбора набора языка Главы 4 для описания того, как использовать псевдокоманду .8080.

Только первый байт является допустимым операнд. Используйте скобки, чтобы указать ассемблеру генерировать один байт для кодов операций, которые обычно генерируют больше чем один. Например:

MVI A,(JMP)

ADI (CPI)

MVI B,(RNZ)

CPI (INX H)

ACI (LXI B)

MVI C,MOV A,B

Ассемблер возвращает ошибку, если более одного байта входит в операнд (в скобках) - таких, как (CPI 5), (LXI B,LABELl), или (JMP LABEL2).

Коды операций, которые генерируют один байт, обычно могут быть использованы в качестве операндов без заключения в скобки.

### 3.4.2 Операторы

MACRO-80 допускает арифметические и логические операторы. Операторы, которые возвращают истинные или ложные условия, возвращают true (истина), если результат имеет какое-либо ненулевое значение и false (ложь), если результат равен нулю.

Следующие арифметические и логические операторы допустимы в выражениях.

| **Оператор** | **Определение** |
| --- | --- |
| NUL | Возвращает истину (true), если аргумент (параметр) является нулевым. Остальная часть строки после NUL принимается в качестве аргумента NUL. Условный  IF NUL <argument>  ложь, если первый символ argument - что-нибудь кроме точки с запятой или возврата каретки. Обратите внимание на то, что IFB и IFNB выполняют те же функции, но более просты в использовании. (См. раздел **Условное Ассемблирование** в Главе 4.) |
| TYPE | Оператор TYPE возвращает байт, который описывает две характеристики своего аргумента: 1) режим, и 2) является ли он внешним или нет. Аргументом TYPE, может быть любое выражение (строка, число, логическое). Если выражение является недопустимым, TYPE возвращает ноль.  Байт, который возвращен, сконфигурирован следующим образом:  Младшие два бита режима. Если младшие два бита равны:  0 абсолютный режим  1 режим перемещаемой программы  2 режим перемещаемых данных  3 режим перемещаемого Common  Старший (80H) бит является внешним битом. Если старший бит установлен, выражение содержит External. Если старший бит выключен, выражение локальное (не внешнее).  Определенный бит 20H. Этот бит установлен, если выражение определено локально, и он выключен, если выражение неопределенное или внешнее. Если не установлены ни один бит, выражение - недопустимое.  TYPE обычно используется в макросах, где тип аргумента, возможно, должен быть протестирован, чтобы принять решение относительно процесса выполнения программы; например, когда условный блок включен.  Пример:  F00 MACRO X  LOCAL Z  Z SET TYPE X  IF Z...  TYPE тестирует режим и тип X. в Зависимости от оценки X блок кода, который начинается с IF Z... может быть ассемблирован или опущен. |
| LOW | изолирует младшие 8 бит абсолютного 16-битного значения |
| HIGH | изолирует старшие 8 бит абсолютного 16-битного значения |
| \* | умножение |
| / | деление |
| MOD | По модулю. Делит левый операнд на правый и возвращает значение остатка (по модулю). |
| SHR | Сдвиг вправо. SHR сдвиг левого операнда вправо на число разрядов, указанное правым операндом. |
| SHL | Сдвиг влево. SHL сдвиг левого операнда влево на число разрядов, указанное правым операндом. |
| - | (Унарный минус) Указывает, что следующее значение является отрицательным, как и в отрицательном целом. |
| + | сложение |
| - | Вычитает правый операнд из левого операнда. |
| EQ | Равно. Возвращает истину, если операнды равны друг другу. |
| NE | Не равно. Возвращает истину, если операнды не равны друг другу. |
| LT | Меньше. Возвращает истину, если левый операнд меньше правого операнда. |
| LE | Меньше или равно. Возвращает истину, если левый операнд меньше или равен правому операнду. |
| GT | Больше чем. Возвращает истину, если левый операнд больше правого. |
| GE | Больше или равно. Возвращает истину, если левый операнд больше или равен правому операнду. |
| NOT | Логическое НЕ. Возвращает истину, если левый операнд истинный правый является ложным, или если правый истинный и левой ложный. Возвращает ложь, если оба являются истинными или оба ложны. |
| AND | Логическое "И". Возвращает истину, если оба оператора являются истиной. Возвращает ложь, если один из операндов является ложным или если оба ложны. Оба операнда должны быть абсолютными значениями. |
| OR | Логическое "ИЛИ". Возвращает истину, если один из операндов истина, или если оба истина. Возвращает ложь, если оба оператора ложь. Оба операнда должны быть абсолютными значениями. |
| XOR | исключающее ИЛИ. Возвращает истину, если один операнд - истина, а другой ложь. Возвращает false, если оба оператора - истина, или если оба оператора - ложь. Оба операнда должны быть абсолютными значениями. |

Приоритет операторов:

NUL, TYPE

LOW, HIGH

\*, /, MOD, SHR, SHL

Унарный минус

+ , -

EQ, NE, LT, LE, GT, GE

NOT

AND

OR, XOR

Сначала вычисляются части выражения, включающие операции, которые имеют в выражении более высокий приоритет.

Порядок очередности может быть изменен при помощи круглых скобок вокруг частей выражения, которым Вы хотите дать более высокий приоритет.

Все операторы кроме +, -, \* и / должны быть разделены от их операндов по крайней мере одним пробелом.

Операторы изоляции байта (HIGH и LOW) изолируют старшие, или младшие разряды 8 бит 16-разрядного значения.

# Глава 4 Функции ассемблера

Макроассемблер MACRO-80 функционально имеет три основных средства: однофункциональные псевдокоманды, макро-средства и средства условного ассемблирования.

## 4.1 Однофункциональные псевдокоманды

Однофункциональная псевдокоманда включает только собственную строку оператора и направляет ассемблер, чтобы выполнить только одну функцию. (Макросы и условные выражения включают больше чем одну строку кода, таким образом, они могут считаться блочными псевдокомандами.)

Однофункциональные псевдокоманды разделены на пять типов: Выбор системы команд, определение данных и определение символов, режим PC, файловые и управление листингом.

**Выбор набора инструкций**

По умолчанию режим мнемоники команд 8080. Если псевдокоманда установки мнемоники команд выбрана не корректно, то ассемблер возвратит фатальные ошибки для кодов операций, которые не допустимы в установленном режиме мнемоники команд. Т.е. .Z80 ассемблирует только коды операций Z80, .8080 ассемблирует только коды операций 8080. Поэтому, если вы написали какие-либо программы на языке ассемблера для Z80, вы должны вставить псевдокоманду .Z80 в начале программы.

Обратите внимание, что все псевдокоманды, перечисленные в этой главе будут ассемблироваться в обоих режимах.

.Z80

.Z80 не имеет аргументов. .Z80 направляет MACRO-80 ассемблировать мнемонику команд Z80.

.8080

.8080 не имеет аргументов. .8080 направляет MACRO-80 ассемблировать мнемонику команд 8080.

Все коды операций, введенные после псевдокоманды выбора мнемоники команд, будут ассемблироваться с этим типом кода, пока не встретится псевдокоманда выбора другой мнемоники команд.

Если вы введете код операции, не принадлежащий выбранной мнемоники команд, то MACRO-80 возвратит синтаксическую ошибку (буква O).

**Определение данных и символов**

Все псевдокоманды определения данных и символов поддерживаются в обоих режимах набора команд. (Существует одно исключение для SET, которая является незаконной в режиме Z80. К вашему сведению, следующие обозначения будут помещены перед определением синтаксиса псевдокоманды для индикации c каким микропроцессором они обычно ассоциируется:

\* Обозначает псевдокоманды Z80

Отсутствие звездочки указывает что псевдокоманды Intel 8080

**Определение байт**

**DB <exp>[,<exp>...]**

\* **DEFB <exp>[,<exp>...]**

**DB <string>[<string>...]**

\* **DEPM <string>[,<string>...]**

Аргументами DB могут быть выражения или строки. Аргументами DEFB - выражения. Аргументами DEFM - строки. Строки должны быть заключены в одиночные или двойные кавычки.

**Примечание:** Обозначение DB используется во всех следующих объяснениях, описывающих все псевдокоманды определения байт. DB используется для хранения значение (строки или числа) в ячейке памяти, начиная с адреса, указанного в счетчике адреса.

Выражения должны иметь один байт. (Если старший байт результата равен 0 или 255, ошибка не выдается, в противном случае, возвращается ошибка A.)

Строки из трех или более символов, не могут быть использованы в выражениях (т.е., они должны сразу сопровождаться запятой или концом строки). Символов в строках 8080 или Z80 хранятся в порядке появления, каждый как один байт с старшим битом равным нулю.

Пример:

DB 'AB'

DB 'AB' AND 0FFH

DB 'ABC'

ассемблируется в:

0000' 41 42 DB 'AB'

0002' 42 DB 'AB' AND 0FFH

0003' 41 42 43 DB 'ABC

**Определение символов**

**DC <string>**

DC хранит символы в <string> в последовательных ячейках памяти, начиная с адреса, указанного в счетчике адреса. Как и DB, символы хранятся в порядке появления, каждый как и однобайтные значения с старшим битом равным нулю. Тем не менее, DC сохраняет последний символ строки с установленным старшим битом в единицу. Возникает ошибка, если аргумент в DC является пустой строкой.

Пример:

F00: DC "ABC"

ассемблируется в:

0000' 41 42 C3 FOO: DC "ABC"

**Определение области** **памяти**

**DS <exp>[,<val>]**

\* **DEFS <exp>[,<val>]**

Псевдокоманда определения области резервирует область памяти. Значение <exp> задает количество байт, которые будут резервированы.

Чтобы инициализировать зарезервированное пространство, установите в <val> желаемое значение. Если <val> - null (опущено), зарезервированное пространство останется как есть (неинициализированным). Резервируемый блок памяти автоматически не инициализируется нулями. В качестве альтернативы присвоению значению <val> нуля, если вы хотите определить пространство блока инициализированного нулями, Вы можете использовать ключ /M во время ассемблирования. См. описание ключа /M в разделе Ключи в Главе 5 Запуск MACRO-80.

Все имена, используемые в <exp> должны быть предварительно определены (то есть, все имена известны в момент первого прохода). В противном случае генерируется ошибка V во время первого прохода, и ошибка U может быть сгенерирована на втором проходе. Если ошибка U не генерируется во время второго прохода, вероятно, будет сгенерирована ошибка фазы, потому что псевдокоманда определения пространства не сгенерировала код на первом проходе.

Пример:

DS 100H

резервирует 100H байт памяти, неинициализированную (независимо от того, какие значения были в тех ячейках памяти прежде, когда программа будет загружена, они все еще будут там). Используйте ключ /M при ассемблировании, чтобы инициализировать 100H байт к нулю, если хотите. Или используйте следующий оператор для инициализации зарезервированного место нулями или любыми другими значениями:

DS 100H,2

резервирует 100H байт, инициализируя каждый значением 2.

**Определение слова**

**DW <exp>[,<exp>...]**

\* **DEFW <exp>[,<exp>...]**

Псевдокоманда определения слова сохраняет значения выражений в последовательных ячейках памяти, начинающихся с текущего счетчика адреса. Значения выражений есть двухбайтовые величины. Значения сохраняют сначала младший, а затем старший байты.

Отличается от DB.

Пример:

F00: DW 1234H

ассемблируется в:

0000' 1234 F00: DW 1234H

Примечание: байты в листинге отображаются в порядке поступления, а не в порядке сохранения.

**Соответствие**

**<name> EQU <exp>**

EQU присваивает значение выражения <exp> символическому имени <name>, <name> может быть меткой, символом или переменной, и может использоваться впоследствии в выражениях. <name> не может сопровождаться двоеточием (двоеточиями).

Если <exp> является внешним, генерируется ошибка. Если <name> уже имеет значение отличное от <exp>, генерируется ошибка M.

Если Вы захотите переопределить <name> в программе позже, используйте псевдокоманду SET или ASET, чтобы определить <name> вместо определенного EQU.

Отличается от SET

Пример:

BUF EQU 0F3H

**Внешнее имя** (Импорт символов из других модулей)

**EXT <name>[,<name>...]**

**EXTRN <name>[,<name>...]**

\* **EXTERNAL <name>[,<name>...]**

**BYTE EXT <symbol>**

**BYTE EXTRN <symbol>**

**BYTE EXTERNAL <symbol>**

Псевдокоманда определения внешнего символа заявляет, что имя (имена) в списке являются внешними (т.е. определенными в другом модуле). Если любой элемент в списке определяет имя, которое определено в текущей программе, генерируется ошибка M. Ссылка на имя, в котором за именем сразу же следует два знака решетки (например, NAME##) также объявляет имя как внешнее.

Внешние символы могут представлять собой один или два байта. Для всех внешних имен, только первые 6 символов передаются компоновщику. Дополнительные символы обрезаются.

Пример:

EXTRN ITRAN ;tranf init rtn

MACRO-80 не генерирует код для этого оператора, во время ассемблирования. Если ITRAN используется в качестве аргумента оператора CALL, оператор CALL ITRAN генерирует код для CALL, но нулевого значения (0000\*) для ITRAN. Во время компоновки LINK-80, будет искать во всех загруженных модулях, оператор PUBLIC ITRAN, и использовать определение ITRAN, которое найдет в этом модуле для определения ITRAN в операторе CALL ITRAN.

**Внутренние имена символов** (Экспорт символов в другие модули)

**ENTRY <name>[,<name>...]**

**GLOBAL <name>[,<name>...]**

**PUBLIC <name>[,<name>...]**

Псевдокоманда определения внутреннего символа объявляет каждое имя в списке, внутренними и таким образом доступными для использования в этой и других программах, которые компонуются LINK-80 вместе с ней. Все имена в списке должны быть определены в текущей программе, иначе возникнет ошибка U. ошибка M - генерируется, если имя внешнее или имя общего (common) блока.

Для всех внутренних имен, только первые 6 символов передаются компоновщику. Дополнительные символы обрезаются.

Пример:

PUBLIC ITRAN ;tranf init rtn

.

.

.

ITRAN: LD HL,PASSA ;store addr of

;reg pass area

MACRO-80 ассемблирует оператор LD, как обычно, но не генерирует код для оператора PUBLIC ITRAN. Когда программа LINK-80 встречает EXTRN ITRAN в другом модуле, она будет продолжать поиск, пока не увидит этот оператор PUBLIC ITRAN. Значит, LINK-80 свяжет значение оператора ITRAN: LD HL,PASSA с оператором CALL ITRAN в другом модуле(ях).

**Set**

**<name> SET <exp>** (не в режиме .Z80)

\* **<name> DEFL <exp>**

**<name> ASET <exp>**

Псевдокоманда SET присваивает значение выражения <exp> символическому имени <name>. <name> может быть меткой, символом или переменной, и могут быть в последствии использовано в выражениях. <name> не может сопровождаться двоеточием (двоеточиями). Если <exp> является внешним, генерируется сообщение об ошибке.

Псевдокоманда SET, не может быть использована в .Z80 режиме, поскольку SET - это операция Z80. Псевдокоманды ASET и DEFL могут использоваться в обоих режимах мнемоник.

Используйте одну псевдокоманду SET вместо EQU для определения и переопределения <name> позже. Имя <name> определенное любой из псевдокоманд можно переопределить с помощью SET, вне зависимости какой псевдокомандой <name> было определено первоначально (однако, запрет на SET в режиме .Z80 все еще действует).

Отличается от EQU

Пример:

F00 ASET BAZ+1000H

Всякий раз, когда F00 используется как выражение (операнд), ALDS ассемблер будет оценивать BAZ+1000H и заменить значение F00. Позже, если вы хотите отображал F00 другие значения, просто введите оператор F00 ASET с другим выражением.

F00 ASET BAZ+1000H

.

.

F00 ASET 3000H

.

.

F00 DEFL 6CDEH

**Режим счетчика команд**

Многие из псевдокоманд действуют на или от текущего адреса счетчика, также известного как счетчик команд (program counter) или PC. Текущее значение счетчика команд является адресом следующего байта, который будет создан.

В MACRO-80, у счетчика команд есть режим, который задает символам и выражениям их режимы. (Обратитесь снова к **Обзору** в Главе 1 и разделу **Символы** в Главе 3, при необходимости.) Каждому режиму LINK-80 выделяет сегмент памяти для инструкций ассемблирования в каждом режиме.

Эти четыре режима – абсолютный (не перемещаемый), перемещаемых данных, перемещаемого кода, и перемещаемого COMMON.

Если режим счетчика команд абсолютный, счетчик команд имеет абсолютный адрес. Если режим счетчика команд перемещаемый, счетчик команд имеет относительный адрес и может считаться смещением от абсолютного адреса куда LINK-80, будет загружать начало перемещаемого сегмента.

Псевдокоманда режима счетчика команд используются чтобы указать, в каком режиме счетчика команд будет ассемблироваться сегмент программы.

**Абсолютный сегмент**

**ASEG**

ASEG не имеет операндов. ASEG генерирует неперемещаемый код.

ASEG устанавливает счетчик адреса для абсолютного сегмента (фактического адреса) памяти. По умолчанию ASEG получит значение равное 0, которое может заставить модуль переписать часть операционной системы. Мы рекомендуем, чтобы каждый ASEG сопровождался оператором ORG устанавливающим адрес 103H или более высокий.

**Сегмент кода**

**CSEG**

У CSEG нет операнда. Код, собранный в режиме перемещаемого кода, может быть загружен в ПЗУ/ППЗУ.

CSEG сбрасывает счетчик адреса для сегмента памяти перемещаемого кода. Значение счетчика будет продолжением предыдущего CSEG (по умолчанию 0), если не указан ORG после CSEG, чтобы изменить местоположение.

Однако следует отметить, что оператор ORG не устанавливает жесткий (абсолютный) адрес в режиме CSEG. Оператор ORG после CSEG заставит ассемблер добавить количество байт, указанных в аргументе <exp> оператора ORG к последнему загруженному адресу CSEG. Если например, задан ORG 50, MACRO-80 добавит 50 байт к текущему значению CSEG затем начнет загрузку CSEG. Эффект очистки оператора ORG после CSEG (и DSEG также) может использоваться, чтобы дать модулю смещение. Логично не позволять ORG устанавливать абсолютный адрес для CSEG чтобы сохранить CSEG перемещаемым.

Чтобы установить абсолютный адрес для CSEG, используйте ключ /P в LINK-80.

CSEG - режим по умолчанию ассемблера. Ассемблирование автоматически начинает выполняться с CSEG, и счетчик адреса в режиме перемещаемого кода, указывает на расположение 0 в сегменте памяти перемещаемого кода. Все последующие инструкции будут добавлены в сегмент памяти перемещаемого кода до выполнения псевдокоманд ASEG, DSEG или COMMON. CSEG вводится, чтобы возвратить ассемблер в режим ассемблирования перемещаемого кода со следующего свободного адреса на которую указывал счетчик адреса в сегменте перемещаемого кода.

**Сегмент данных**

**DSEG**

У псевдокоманды DSEG нет операндов. DSEG определяет сегменты ассемблированного перемещаемого кода, который будет позже загружен только в оперативную память.

DSEG устанавливает счетчик адреса в сегменте памяти перемещаемых данных.

Значение счетчика будет продолжением предыдущего DSEG (по умолчанию 0), если отсутствует ORG после DSEG, чтобы изменить местоположение.

Однако следует отметить, что оператор ORG не устанавливает жесткий (абсолютный) адрес в режиме DSEG. Оператор ORG после DSEG заставит ассемблер добавить количество байт, указанных в аргументе <exp> оператора ORG к последнему загруженному адресу DSEG. Если например, задан ORG 50, MACRO-80 добавит 50 байт к текущему значению DSEG затем начнет загрузку DSEG. Эффект очистки оператора ORG после DSEG (и CSEG также) может использоваться, чтобы дать модулю смещение. Логично не позволять ORG устанавливать абсолютный адрес для DSEG чтобы сохранить DSEG перемещаемым.

Чтобы установить абсолютный адрес для DSEG, используйте переключатель /D в LINK-80.

**Общий блок**

**COMMON /<block name>/**

Аргумент к COMMON название общей области данных. COMMON создает в программе общую область данных для каждого COMMON блока с именем. Если аргумент <block name> опущен, или состоит из пробелов, блок считается неименованным.

Операторы COMMON являются неисполнимыми, операторами выделения памяти. COMMON определяет переменные, массивы и данные в области памяти под названием COMMON. Это позволяет различным программным модулям совместно использовать одну и ту же область памяти. Операторы введенные после оператора COMMON ассемблируются в зону COMMON, обозначенную <block name>. Длина области COMMON это число байт, требуемых для размещения переменных, массивов и данных, объявленных в блоке COMMON, который заканчивается, когда встречаются другие псевдокоманды программного счетчика. COMMON блоки с одинаковым именем могут иметь различную длину. Если их длины отличаются, то программный модуль с самым длинным COMMON блоком должен быть загружен первым (т.е. должен быть первым именем модуля, введенным в командной строке LINK-80 см. Главу 6 описание LINK-80).

COMMON устанавливает счетчик на адрес выбранного блока общей памяти. Адрес всегда указывает начало области, таким образом, сохраняется совместимость с оператором COMMON в FORTRAN.

Пример:

COMMON /DATABIN/

ANVIL EQU 100H

DB 0FFH

DW 1234H

DCI 'FORGE'

CSEG

.

.

**Установка начала**

**ORG <exp>**

В любой момент, значение счетчика может быть изменено путем использования ORG. В режиме программного счетчика ASEG, счетчик устанавливается в значение <exp>, и ассемблер генерирует код, начинающийся с этого значения. В режимах CSEG, DSEG и COMMON программный счетчик сегмента увеличивается на величину <exp>, и ассемблер генерирует код, начинающийся со значением адреса этого последнего загруженного сегмента плюс значение <exp>. Все имена, используемые в <exp> должны быть известны на первом проходе, а значение должно быть абсолютным, или быть в том же диапазоне, что и счетчик адреса.

Пример:

DSEG

ORG 50

устанавливает значение счетчика адреса перемещаемых данных в 50, относительно адреса сегмента памяти перемещаемых данных. Это означает, что первые 50H адресов будут заполнены 0.

Этот метод обеспечивает перемещаемость. Оператор ORG <exp> не определяет фиксированный адрес в режиме CSEG или DSEG. Скорее, LINK-80 загружает в сегмент по подвижному адресу, соответствующие модули, которые загружаются вместе.

С другой стороны, программа которая начинается с операторов

ASEG

ORG 800H

приведет к ассемблированию только в абсолютном режиме, и будет всегда загружать начало по адресу 800H, если оператор ORG не изменен в исходном файле.

То есть, оператор ORG <exp> следующий за ASEG назначает сегменту фиксированный (т.е. абсолютный) адрес, указанный в <exp>.

Однако, та же программа, ассемблированная в режиме перемещаемого кода без оператора ORG, может быть загружена в любой указанный адрес путем добавления ключа /P:<address> в командной строке LINK-80. (Дополнительные сведения см. в Разделе 6.3, **Ключи**.)

**Перемещение перед загрузкой**

**.PHASE <exp>**

.

.

**.DEPHASE**

.PHASE позволяет коду быть расположенным в одной области, но быть выполненным в другой области с начальным адресом, определенным <exp>. <exp> должно иметь абсолютное значение. .DEPHASE используется, чтобы указать конец блока перемещаемого кода.

Режим программного счетчика в блоке .PHASE является абсолютным, так же, как и <exp> в операторе .PHASE. Код, однако, загружается в область, по адресу действовавшему на момент встречи оператора .PHASE. Код внутри блока для исполнения позднее перемещается по адресу, указанному в <exp>.

Пример:

.PHASE 100H

FOO: CALL BAZ

JMP ZOO

BAZ: RET

.DEPHASE

ZOO: JMP 5

ассемблируется в:

.PHASE 100H

0100 CD 0106 FOO: CALL BAZ

0103 C3 0007' JMP ZOO

0106 C9 BAZ: RET

.DEPHASE

0007' C3 0005 ZOO: JMP 5

END

Блоки .PHASE....DEPHASE представляют собой способ выполнения блока кода по определенному абсолютному адресу

**Псевдокоманды имеющие отношение к файлу**

Псевдокоманды имеющие отношения к файлу вставляют длинные комментарии в программу, дают модулю имя, заканчивают модуль или вставляют другие файлы в текущую программу.

**Комментарии**

**.COMMENT <delim><text><delim>**

Первый символ отличный от пробела встретившийся после .COMMENT используется в качестве разделителя. <text> после разделителя становится блоком комментария, который продолжается до следующего появления разделителя (delimiter).

Используйте псевдокоманду .COMMENT для написания длинных комментарии. Не обязательно, вводить в точку с запятой, чтобы указать комментарий. В самом деле, основная причина для использования .COMMENT является отсутствие необходимости начинать каждого строку комментария точкой с запятой. Во время сборки блоки .COMMENT игнорируются и не ассемблируются.

Пример:

.COMMENT \* любое количество текста, вводится здесь

.

.

.\* ;Возврат к нормальному ассемблированию

**Окончание программы**

**END [<exp>]**

Оператор END определяет конец модуля. Если оператор END не включен, появится предупреждающее сообщение %No END statement.

<exp> может быть меткой, символом, числом или любым другим допустимым аргументом, которые LINK-80, сможет использовать в качестве стартового адреса, который будет загружен. Если присутствует <exp>, то LINK-80 поместит инструкций 8080 JMP в 0100H с адресом <exp>. Если <exp> отсутствует, то начальный адрес не передается LINK-80 для той программы, и выполнение начинается с первого загруженного модуля. (Кроме того, если <exp> не будет определен, LINK-80, то переключатель /G не будут воздействовать на модуль.)

<exp> говорит LINK-80, что программа представляет собой основную программу. Без <exp> LINK-80 рассматривает программу на языке ассемблера в качестве подпрограммы. Если Вы собираете программы только на языке ассемблера, и ни одна из них не содержит оператор END с <exp>, то LINK-80 попросит основную программу. Если Вы собираете две или больше программ с операторами END <exp>, LINK-80, не сможет различить, которая из них должна быть основной программой.

Если вы хотите, чтобы связать два или более основных программ, используйте ключи /G:Name или /E:Name для LINK-80 (см. Раздел 6.2.2, **Ключи**). "Name" определит <exp> оператора END для программы, которую Вы хотите назначить основной программой.

Если какая-либо программа на языке высокого уровня соединяется с модулями на ассемблере, LINK-80 автоматически берет программу высокоуровневого языка в качестве основной программы. Поэтому, если Вы хотите чтобы модуль ассемблера, выполнялся перед программой высокоуровневого языка, используйте ключи /G:Name или переключатель /E:Name для LINK-80, чтобы установить модуль ассемблера в качестве начальной программы.

Как альтернатива, мы рекомендуем, чтобы Вы поместили оператор CALL или INCLUDE в начале программы на языке высокого уровня и вызвали программу на языке ассемблера для выполнения до выполнения программы на языке высокого уровня.

**Включения в текст исходной программы**

**INCLUDE <filename>**

**$INCLUDE <filename>**

**MACLIB <filename>**

Все эти псевдокоманды эквивалентны.

Псевдокоманды INCLUDE позволяют вставить исходный код из другого исходного файла на языке ассемблера в текущий исходный файл во время ассемблирования. Использование псевдокоманды INCLUDE избавляет от необходимости повторять часто используемые последовательности инструкций в текущем исходном файле.

<filename> - это правильная спецификация файла для операционной системы. Если расширение имени файла и/или обозначение устройства отличаются от значений по умолчанию, спецификации имени файла-источника должны включать их. По умолчанию расширение файла для исходных файлов .MAC. По умолчанию используется устройство текущего диска или устройства.

Включенный файл открывается и ассемблируется в текущем исходном файле сразу после оператора псевдокоманды INCLUDE, при достижении конца файла, ассемблирование возобновляется со следующего оператора после псевдокоманды INCLUDE.

Вложенные INCLUDE, не допускаются. Если они встретятся, то приведут к нежелательной синтаксической ошибке O.

Файл, указанный в поле операнда должен существовать. Если файл не будет найден, то возвращается ошибка V (ошибка значения), и включение игнорируется. Также возвращается ошибка V, если расширение включаемого файла не .MAC.

В листинге MACRO-80 печатается буква «+» между собранным кодом и строкой исходного текста в каждой строке, ассемблированного из включаемого файла. См. ниже раздел **Псевдокоманды управления листингом** для изучения описания формата файлов листинга.

**Имя модуля**

**NAME ('modname')**

NAME определяет имя модуля. Требуется круглые скобки и кавычки вокруг поля modname. Только первые шесть символов значимы в имени модуля.

Имя модуля также может быть определено в псевдокоманде TITLE. В отсутствие псевдокоманд NAME и TITLE, имя модуля создается из имени исходного файла.

**Основание системы счисления**

**.RADIX <exp>**

<exp> в операторе .RADIX всегда десятичная числовая константа, независимо от текущей системы счисления.

По умолчанию ввод системы счисления (или основания) для всех констант десятичный Псевдокоманда .RADIX позволяет изменить входную систему счисления к любому основанию в диапазоне от 2 до 16.

.RADIX не изменяет основание в листинга. Скорее, это позволяет вам вводить числовые значения в выбранной системе счисления без специальных обозначений. (Значения в других системах счисления по-прежнему требуют специальные обозначения, описанные в Разделе 3.4.1.) Значения генерируемого кода остаются в шестнадцатеричной системе счисления.

Пример:

DEC: DB 20

.RADIX 2

BIN: DB 00011110

.RADIX 16

HEX: DB 0CF

.RADIX 8

OCT: DB 73

.RADIX 10

DECI: DB 16

HEXA: DB 0CH

ассемблируется в:

0000' 14 DEC: DB 20

0002 .RADIX 2

0001' 1E BIN: DB 00011110

0010 .RADIX 16

0002' CF HEX: DB 0CF

0008 .RADIX 8

0003' 3B OCT: DB 73

000A .RADIX 10

0004' 10 DECI: DB 16

0005' 0C HEXA: DB 0CH

**Запрос**

**.REQUEST <filename>[,<filename>...]**

При запуске LINK-80, .REQUEST посылает запрос компоновщику LINK-80 на поиск неудовлетворенных внешних имен в файлах, указанных в списке. Если LINK-80 находит любые неопределенные внешние символы (внешние символы, для которых соответствующий PUBLIC символ в настоящее время не загружен), вы будете знать, что вам необходимо загрузить один или несколько дополнительных модулей для завершения компоновки.

Имена файлов в списке должны быть в форме допустимых символов. <filename> не должно включать расширение имени файла или обозначение устройства. LINK-80 предполагает расширение по умолчанию (.REL) и текущий диск.

Пример:

.

.

.REQUEST SUBR1

.

.

LINK-80 будет искать в SUBR1 внешние символы, которые не имеют соответствующих определений PUBLIC символов объявленные среди загруженных в данный момент модулей.

**Псевдокоманды листинга**

Псевдокоманды листинга выполняют две основные функции: управление форматом и управление листингом. Псевдокоманды управления форматом позволяют программисту вставлять разрывы страниц и прямые заголовки страниц. Управление листингом включать и выключать листинг для всего или части ассемблируемого файла.

**Управление форматированием**

Эти псевдокоманды позволяют вам выводить разрывы страниц, заголовки и подзаголовки в распечатках программ.

Перевод формата

\* **\*EJECT [<exp>]**

**PAGE <exp>**

**$EJECT**

Эта псевдокоманды перевода формата заставляет ассемблер начать с нового листа. Ассемблер помещает символ прогона страницы в файл листинга в конце страницы.

Если присутствует значение <exp>, определяет новый размер страницы (изменяет количество строк на странице) и должно быть в диапазоне от 10 до 255. Размер страницы по умолчанию составляет 50 строк на странице.

Пример:

.

.

\*EJECT 58

.

.

Ассемблер заставляет принтер начинать новую страницу каждый раз после печати 58 строк программы.

**Заголовок**

**TITLE <текст>**

TITLE определяет заголовок, который будет указан на первой строке каждой страницы. Если задана больше чем один псевдокоманда TITLE, происходит ошибка Q. Первые шесть символов заголовка используются в качестве имени модуля, если псевдокоманда NAME не используется. (Если псевдокоманды TITLE и NAME не используются, имя модуля присваивается из исходного имени файла. )

Пример:

TITLE PROG1

Имя модуля теперь PROG1. Модуль можно вызывать этим именем, которое будет напечатано в верхней части каждой страницы листинга.

**Подзаголовок**

**SUBTTL <текст>**

**$TITLE ('<текст>')**

Псевдокоманда SUBTTL определяет подзаголовок который будет выведен в каждом заголовке страницы в строке после заголовка. Поле <текст> усекается после 60 символов.

В программе может быть задано любое число псевдокоманд SUBTTL. Каждый раз, когда ассемблер встречается с SUBTTL, он заменяет <текст> из предыдущего SUBTTL на <текст> от следующего встреченного SUBTTL. Чтобы выключить вывод SUBTTL для части листинга, введите SUBTTL с пустой строкой <текст>.

Пример:

SUBTTL SPECIAL I/O ROUTINE

.

.

SUBTTL

.

.

Первая псевдокоманда SUBTTL вызывает подзаголовок “SPECIAL I/O ROUTINE”, который будет напечатан в верхней части каждой страницы. Второй SUBTTL выключается подзаголовок (строка подзаголовка в листинг останется пустой).

**Общее управление листингом**

**.LIST** - Печать всех строк кода

**.XLIST** - Подавить печать в листинг

.LIST - условие по умолчанию. Если Вы определите файл листинга в командной строке, то файл будет создан.

При появлении псевдокоманда .XLIST в исходном файле, исходный и объектный код не будет распечатываться. .XLIST действует, пока не встретится .LIST.

.XLIST перекрывает все другие команды управления листингом. Так, ничего не будет выведено, даже если встретится другая псевдокоманда листинга (кроме .LIST).

Пример:

.

.

.XLIST ; здесь листинг приостановлен

.

.

.LIST ; здесь листинг возобновлен

**Распечатать на терминале**

**.PRINTX <разделитель><текст><разделитель>**

Первый символ после .PRINTX, отличный от пробела, считается разделителем. Текст, идущий после разделителя, во время трансляции выводится на терминал до тех пор, пока не встретится второй разделитель - тот же самый символ. Псевдокоманда PRINTX полезна для визуализации процесса прохождения программы через транслятор или для отображения значения условных переключателей сборки.

.PRINTX выводит листинг при каждом проходе ассемблера, поэтому, если листинг нужен только при одном каком-либо проходе, то следует использовать псевдокоманды IF1 или IF2. Смотрите условные псевдокоманды IF1 и IF2.

Пример:

.PRINTX \*Половина ассемблирования выполнена\*

Ассемблер отправит это сообщение на экран терминала при появлении.

IF1

.PRINTX \*Проход 1 выполнен\* ; сообщение только на 1 проходе

END IF

IF2

.PRINTX \*Проход 2 выполнен\* ; сообщение только на 2 проходе

END IF

**Условное управление листингом**

Три псевдокоманды условного управление листингом используются, чтобы определить, хотите ли Вы, чтобы операторы, содержащиеся в условном блоке имеющем значение ложь (False), появились в листинге. См. также описание ключа /X в Главе 5.

**Подавление ложных условных выражений**

**.SFCOND**

.SFCOND подавляет часть листинга условное выражение которого содержит значение ложь.

**Вывод ложных условных выражений**

**.LFCOND**

.LFCOND распечатывает часть листинга часть листинга условное выражение которого содержит значение ложь.

**Переключение ложных условных выражений**

**.TFCOND**

.TFCOND переключает текущую настройку. Псевдокоманда .TFCOND работает независимо от .LFCOND и .SFCOND. Псевдокоманда .TFCOND переключает настройку по умолчанию, которая установлена в присутствии или в отсутствие в командной строке ассемблера ключа /X. Если ключ /X присутствует, то .TFCOND распечатывает часть листинга условное выражение которого содержит значение ложь. Если ключ /X отсутствует, то .TFCOND будет подавлять печать часть листинга условное выражение которого содержит значение ложь.

**Управление листингом макро расширений**

Псевдокоманды расширенным управления листингом строк макро и псевдокомандами повторения блоков (REPT, IRP, IRPC), могут использоваться только в макро или блоке повторения.

**Исключение строк не генерирующих код**

**.XALL**

.XALL - значение по умолчанию.

.XALL выводит исходный и объектный код, произведенный макросом, но исходные строки, которые не генерируют код, не выводятся.

**Вывод текста макро**

**.LALL**

.LALL перечисляет полный макро-текст для всех расширений, включая строки, которые не генерируют код.

**Подавление текста макро**

**.SALL**

.SALL подавляет в листинге вывод текста и объектного кода, произведенного макросами.

**Псевдокоманды управления листингом CREF**

Вам может понадобиться возможность создания листинга перекрестных ссылок для части программы, а не всей.

Чтобы управлять печатью или подавлением печати перекрестных ссылок, используйте псевдокоманды управления листингом перекрестных ссылок .CREF и .XCREF, в исходном файле для MACRO-80.

Эти две псевдокоманды могут вводиться в любом месте программы в поле операторов, как и другие псевдокоманды управления листингом .CREF и .XCREF не поддерживают параметры.

**Подавление перекрестных ссылок**

**.XCREF**

Псевдокоманда .XCREF отменяет .CREF (по умолчанию). .XCREF действует до тех пор пока MACRO-80 не встретит .CREF.

Используйте .XCREF, чтобы подавить создание перекрестных ссылок в выбранных частях файла. Поскольку .CREF и .XCREF не действуют, пока не установлен ключ /C в командной строке MACRO-80, нет необходимости использовать .XCREF, если Вы хотите создать обычный файл листинга (без перекрестных ссылок). Просто опустите ключ /C в командной строке компиляторе ассемблера.

**Печать перекрестных ссылок**

**.CREF**

.CREF - условие по умолчанию. Используйте .CREF, чтобы возобновить, создание файла перекрестных ссылок после использования псевдокоманды .XCREF. .CREF действует до тех пор пока MACRO-80 не встретит .XCREF.

Однако следует отметить, что .CREF не действует, пока ключ /C не установлен в командной строке MACRO-80.

## 4.2 Макросредства

Макросредства позволяет создавать блоки кода, которые можно повторить без перекодирования. Блоки кода, начинаются с псевдокоманды определения макроса или одной из псевдокоманд повторения и заканчиваются псевдокомандой ENDM. Все макро псевдокоманды могут быть использованы внутри блока макроса. Фактически, вложение макросов ограничено только памятью.

Макросредства макроассемблера MACRO-80 включает псевдокоманды для:

Определения макроса

MACRO

повторения:

REPT (повторение)

IRP (неопределенное повторение)

IRPC (неопределенное повторение символа)

окончания:

ENDM

EXITM

уникальных символов в макро-блоках:

LOCAL

Макросредства также поддерживает некоторые специальные макро-операторы:

&

;;

!

%

**Определения макроса**

<name> MACRO <dummy>[,<dummy>...]

.

.

ENDM

Блок операторов от строки оператора MACRO до строки оператора ENDM включает в себя тело макроса, или макроопределение.

Поле <name> походит на МЕТКУ и соответствует правилам для формирования символов. Обратите внимание, что <name> может быть любой длины, но только первые 16 символов передаются компоновщику. После того, как макрос определен, <name> используется для, вызова макроса.

<dummy> формальный параметр, который заменяется фактическим параметром один к одному текстовой заменой, когда используется блок MACRO. Каждый формальный параметр <dummy> может быть до 32 символов длиной. Число формальных параметров ограничено только длиной строки. Если Вы определяете больше чем один формальный параметр, они должны быть разделены запятыми. MACRO-80 интерпретирует все символы между запятыми как единственный формальный параметр.

**Примечание**

Формальный параметр всегда признается исключительно как формальный параметр. Даже если имя регистра (например, А или В) используют в качестве формального параметра, он будет заменен параметром при расширении.

Макро-блок не компилируется, когда он встречается. Скорее когда Вы вызываете макрос, ассемблер "разворачивает" оператор вызова, переносит и компилирует соответствующий блок макроса.

Если Вы хотите использовать псевдокоманды TITLE, SUBTTL или NAME для части Вашей программы, в которой появляется макро-блок, Вы должны быть осторожными относительно формы оператора. Например, если Вы введете SUBTTL MACRO DEFINITIONS, то MACRO-80 ассемблирует оператор как макроопределение с именем макроса SUBTTL и формальным параметром DEFINITIONS. Чтобы избежать этой проблемы, измените слово MACRO каким-либо способом, например, MACR0, MACROS, и т.д.

**Вызов макроса**

Для использования макроса введите оператор вызова макроса:

**<name> <parameter>[,<parameter>...]**

<name> является <name> макроса блока. Фактический параметр <parameter> заменяет фиктивный параметр <dummy> в соотношении один к одному. Число параметров ограничено только длиной строки. При вводе более одного параметра, они должны быть разделены запятыми. Если Вы поместите угловые скобки вокруг параметров, разделенных запятыми, то ассемблер передаст все элементы в угловых скобках как единственный параметр. Например:

F00 1,2,3,4,5

передает пять параметров макросу, но:

F00 <1,2,3,4,5>

передает только один.

Число параметров в операторе вызова макроса не должно совпасть с числом фиктивных параметров в макроопределении. Если формальных параметров больше, чем фиктивных, лишние будут проигнорированы. Если меньше, то вместо недостающих фактических параметров будут подставлены нули. Ассемблированный код блока макроса будет включен - после каждого оператора макро вызова.

Пример:

EXCHNG MACRO X,Y

PUSH X

PUSH Y

POP X

POP Y

ENDM

Если Вы введете как часть программы некоторый код и оператор вызова макроса:

LDA 2FH

MOV HL,A

LDA 3FH

MOV DE,A

EXCHNG HL,DE

Ассемблер генерирует код:

0000' 3A 002F LDA 2FH

0003' 67 MOV HL,A

0004' 3A 003F LDA 3FH

0007' 57 MOV DE,A

EXCHNG HL,DE

0008' E5 + PUSH HL

0009' D5 + PUSH DE

000A' El + POP HL

00OB' Dl + POP DE

**Псевдокоманды повторения**

Псевдокоманды в этой группе позволяет операциям в блоке быть повторенными столько раз, сколько Вы определяете. Существенными различиями между псевдокомандами повторения и псевдокомандой MACRO являются:

1. MACRO дает блоку имя, которым можно вызвать в программе везде и когда это необходимо; блок макроса может быть использован в самых разных программах, просто введом оператора вызова макроса.
2. MACRO позволяет параметры, передаваемые в блок MACRO при вызове MACRO, а значит, параметры могут быть изменены.

Параметры псевдокоманда повторения должны быть присвоены как часть блока кода. Если параметры известны заранее и не будут изменяться, и если повторение должно быть выполнено для каждого выполнения программы, то псевдокоманда повторения удобна. Псевдокоманда MACRO, Вы должны вызвать MACRO каждый раз, когда это необходимо.

Обратите внимание на то, что каждая псевдокоманда повторения должна иметь соответствующую псевдокоманда ENDM, чтобы завершить блок повторения.

**Повторение**

**REPT <exp>**

.

.

**ENDM**

Блок операторов между REPT и ENDM повторяется <exp> раз. <exp> рассматривается как 16-разрядное число без знака. Если <exp> содержит внешний символ или неопределенные операнды, генерируется ошибка.

Пример:

X SET 0

REPT 10 ; Генерирует DB 1 - DB 10

X SET X+1

DB X

ENDM

Ассемблер генерирует код:

0000 X SET 0

REPT 10 ; Генерирует DB 1 - DB 10

X SET X+1

DB X

ENDM

0000' 01 + DB X

0001' 02 + DB X

0002' 03 + DB X

0003' 04 + DB X

0004' 05 + DB X

0005' 06 + DB X

0006' 07 + DB X

0007' 08 + DB X

0008' 09 + DB X

0009' 0A + DB X

END

**Неопределенное повторение**

**IRP <dummy>,<параметры в угловых скобках>**

.

.

**ENDM**

Параметры должны быть включены в угловые скобки. Параметры могут быть любым допустимым символом, строкой, числовой, или символьной константой. Блок операторов повторяется для каждого параметра. Каждое повторение подставляет следующий параметр содержащийся в блоке <dummy>. Если параметр - пуст (т.е., <>), блок повторяется один раз с пустым параметром.

Пример:

IRP X,<1,2,3,4,5,6,7,8,9,10>

DB X

ENDM

Этот пример генерирует те же байты (DB 1 - DB 10) как пример REPT.

Когда IRP используется внутри блока макроопределения, угловые скобки вокруг параметров в операторе макро вызова удаляются до того, как параметры передаются в макроблок. Пример, генерирует тот же код что и выше, иллюстрирует удаление одного уровня скобкок из параметров:

FOO MACRO X

IRP Y,<X>

DB Y

ENDM

ENDM

Когда оператор вызова макро

FOO <1,2,3,4,5,6,7,8,9,10>

ассемблирован, расширение макроса становится:

IRP Y,<1,2,3,4,5,6,7,8,9,10>

DB Y

ENDM

Угловые скобки вокруг параметры удаляются, и все элементы передаются в качестве единственного параметра.

**Неопределенное повторение символа**

**IRPC <dummy>,<string>**

.

.

**ENDM**

Операторы в блоке повторяются один раз для каждого символа в строке. Каждое повторение заменяет следующим символом в строке для каждого возникновения <dummy> в блоке.

Пример:

IRPC X,0123456789

DB X+1

ENDM

Этот пример генерирует тот же код (DB 1 - DB 10) как два предыдущих примера.

**Окончание MACRO**

**ENDM**

ENDM говорит ассемблеру, что блок MACRO или повторения закончен.

Каждый MACRO, REPT, IRP, и IRPC должны завершаться псевдокомандой ENDM. В противном случае в конце каждого прохода генерируется сообщение 'Unterminated REPT/IRP/IRPC/MACRO' (Незакрытый REPT/IRP/IRPC/MACRO). Несопоставленный ENDM вызывает ошибку O. Если Вам необходимо выйти от MACRO или блока повторения, прежде чем расширение будет завершено, используйте EXITM.

**Завершение** MACRO

**EXITM**

Псевдокоманда EXITM используется в блоке MACRO или повторения, чтобы завершить расширение, когда некоторое условие делает оставшееся расширение ненужным или нежелательным. Обычно EXITM используется в сочетании с условными псевдокомандами.

Когда EXITM ассемблирован происходит выход из расширения. Любое остающееся расширение или повторение не генерируется. Если блок, содержащий EXITM, вложен в другом блоке, внешний уровень продолжает расширяться.

Пример:

FOO MACRO X

Y SET 0

REPT X

Y SET Y+l

IFE Y-0FFH ;test Y

EXITM ;if true, exit REPT

ENDIF

DB Y

ENDM

ENDM

**Метки типа LOCAL**

**LOCAL <dummy>[,<dummy>...]**

Псевдокоманда LOCAL допускается только внутри блока макроопределения. Когда LOCAL выполняется, ассемблер создает уникальный символ (символическое имя) для каждого <dummy> и заменяет этот символ для каждого вхождения <dummy> в расширении. Эти уникальные символы, как правило, используется для определения меток в макрос, тем самым исключая создание многократно определенных меток при неоднократных вызовах макроко манды. Символы созданются ассемблером в диапазоне от ..0001 до ..FFFF. Поэтому пользователю в своих программах следует избегать имен типа ..nnnn. Оператор LOCAL должен предшествовать всем другим операторам в макроопределении.

Пример:

FOO MACRO NUM,Y

LOCAL A,B,C,D,E

A: DB 7

B: DB 8

C: DB Y

D: DB Y+l

E: DW NUM+1

JMP A

ENDM

FOO 0C00H,0BEH

END

генерирует следующий код (обратите внимание, что MACRO-80 заменила имена меток в виде ..nnnn для экземпляров фиктивных символов):

FOO MACRO NUM,Y

LOCAL A,B,C,D,E

A: DB 7

B: DB 8

C: DB Y

D: DB Y+l

E: DW NUM+1

JMP A

ENDM

FOO 0C00H,0BEH

0000' 07 +..0000: DB 7

00011 08 +..0001: DB 8

0002' BE +..0002: DB 0BEH

0003' BF +..0003: DB 0BEH+1

0004' 0C01 +..0004: DW 0C00H+1

0006' C3 0000' + JMP ..0000

END

**Специальные макро-операторы**

Некоторые специальные операторы могут быть использованы в макроблоке, чтобы выбрать дополнительные функции ассемблирования.

|  |  |
| --- | --- |
| **&** | Знак амперсенда используется для конкатенации имен и текста в макрорасширениях. Параметр макро, являющийся строкой в кавычках, не будет заменять формальный параметр, если только тому не предшествует знак амперсенда. Для формирования имен из текста и параметров необходимо поставить между ними знак &.  Например:  ERRGEN MACRO X  ERROR&X: PUSH B  MVI B,'&X'  JMP ERROR  ENDM  Вызов ERRGEN A генерирует следующее расширение:  ERRORA: PUSH B  MVI B,'A'  JMP ERROR |
| **;;** | В блоковых командах комментарий, следующий за двумя точками с запятыми, не будет входить в листинг макрорасширения даже при команде .LALL, в отличие от комментария, следующего за одной точкой с запятой. |
| **!** | Восклицательный знак может быть введен в качестве аргумента, чтобы указать, что следующий символ включается в программу непосредственно. Таким образом, !; эквивалентно <;>. |
| **%** | Знак процента используется только в качестве аргумента макроса для преобразования выражения, которое следует за ним (обычно символ) в число в текущей системе счисления (установленного псевдокомандой .RADIX). Во время макрорасширения число, полученное из преобразования выражения, заменяет формальный параметр. Использование специального оператора % позволяет макро вызов по значению. (Обычно, макро вызов это вызов по ссылке с текстом параметра макро, полностью замещающим текст формального параметра.)  Выражение после % должно соответствовать тем же правилам что и выражения для псевдокоманды DS (Определение области памяти). Т.е. требуется допустимое выражение, которое преобразуется в абсолютную (неперемещаемую) константу.  Пример:  PRINTE MACRO MSG,N  .PRINTX \* MSG,N \*  ENDM  SYM1 EQU 100  SYM2 EQU 200  PRINTE <SYM1 + SYM2 = >,%(SYMl + SYM2)  Обычно, вызов оператор макро заставил бы строку (SYMl + SYM2) заменить формальный параметр N. Результат был бы:  .PRINTX \* SYMl + SYM2 = (SYMl + SYM2)  Когда % помещен перед параметром, ассемблер генерирует:  .PRINTX \* SYMl + SYM2 = 300 \* |

## 4.3 Команды условного ассемблирования

Псевдокоманды условного ассемблирования позволяет пользователям разрабатывать блоки кода, которые действуют в зависимости от условия.

Все условные команды имеют следующий формат:

IFxxxx [argument] COND [argument]

. .

. .

[ELSE [ELSE

. .

. .

.] .]

ENDIF ENDC

У каждого IFxxxx должен быть соответствующий ENDIF, чтобы завершить условное выражение. У каждого COND должен быть соответствующий ENDC, чтобы завершить условное выражение. Иначе, генерируется сообщение 'Unterminated conditional' (Незакрытый условный оператор) в конце каждого прохода. ENDIF без соответствующего IF или ENDC без соответствующего COND вызывает ошибку C.

Ассемблер сравнивает логическое выражение (условие) оператора с TRUE («истина») (который равняется FFFFH, или -1, или любое ненулевое значение), или с FALSE («ложь») (который равняется 0000H). Код в условном блоке асеемблируется, если логическое выражение (условие) зданное в условном операторе принимает значение «истина». Если логическое выражение (условие) ложно, ассемблер или игнорирует условный блок полностью, или, если условный блок содержит дополнительный оператор ELSE, ассемблирует только часть кода после ELSE.

Условные выражения могут быть вложены до 255 уровней. Любой аргумент условного выражения должен быть известен на проходе 1, чтобы избежать ошибок V и неправильной оценки. Для IF/IFT/COND и IFF/IFE выражения должны иметь значения, которые ранее были определены и выражение должны быть абсолютными. Если имя определено после IFDEF или IFNDEF, проход 1 считает что имя неопределено, но оно будет определено на проходе 2.

Каждый условный блок может дополнительно включать псевдокоманду ELSE, которая позволяет генерировать дополнительный код, если существует противоположное состояние. Допускается только один ELSE для заданного IFxxxx/COND.

ELSE во вложенных операторах отвечает ближайшему IF, не имеющему ELSE. Условный оператор, имеющий более одного ELSE, или ELSE без условного оператора вызывают ошибку C.

**Псевдокоманды условного ассемблирования**

**IF <exp>**

**IFT <exp>**

\* **COND <exp>**

Если логическое выражение <exp> оценивается как не 0, операторы внутри условного блока ассемблируются.

**IFE <exp>**

**IFF <exp>**

Если логическое выражение <exp> возвращает 0, операторы внутри условного блока ассемблируются.

**IF1** Истина при первом проходе

Если ассемблер находится на проходе 1, операторы внутри условного блока ассемблируются.

**IF2** Истина при втором проходе

Если ассемблер находится на проходе 2, операторы внутри условного блока ассемблируются.

**IFDEF <symbol>**

Если <symbol> определен или был объявлен внешним (EXTERNAL), операторы внутри условного блока ассемблируются.

**IFNDEF <symbol>**

Если <symbol> не определен или не был объявлен внешним, операторы внутри условного блока ассемблируются.

**IFB <arg>**

Обязательно требуются угловые скобки вокруг <arg>.

Если <arg> пустой (не задан ) или нуль (две угловых скобки без содержимого внутри, <>), операторы внутри условного блока ассемблируются.

**IFNB <arg>**

Обязательно требуются угловые скобки вокруг <arg>.

Если <arg> - это не пустой, операторы внутри условного блока ассемблируются. Используется для тестирования пустых параметров.

**IFIDN <argl>,<arg2>**

Обязательно требуются угловые скобки вокруг <argl> и <arg2>.

Если строка <argl> идентичена строке <arg2>, операторы внутри условного блока ассемблируются.

**IFDIF <argl>,<arg2>**

Обязательно требуются угловые скобки вокруг <argl> и <arg2>.

Если строка <argl> отличается от строки <arg2>, операторы внутри условного блока ассемблируются.

**ELSE**

Псевдокоманда ELSE позволяет генерировать альтернативный код, когда существует противоположное условие. Может использоваться с любой из условных псевдокоманд.

**END IF**

**\* ENDC**

Эти псевдокоманды завершают условные блоки. Для каждого используемой псевдокоманды условного ассемблирования должна быть дана псевдокоманда окончания блока. ENDIF должна быть согласована с псевдокомандой IFxxxx. ENDC должна быть согласованы с псевдокомандой COND.

# Глава 5 Запуск MACRO-80

После завершения создания исходного файла на ассемблере, вы готовы транслировать его. MACRO-80 транслирует операторы исходного файла, в том числе расширения макросов и псевдокоманд повторения. Результатом трансляции будет перемещаемый объектный код, который готов для компоновки в LINK-80. Перемещаемый объектный код может быть сохранен в файле на диске, которому ассемблер присваивает расширение .REL. Ассемблированный файл (REL) не является исполняемым файлом. Файл станет исполняемой программой только после того, как он будет обработан программой LINK-80.

MACRO-80 занимает приблизительно 19K памяти и имеет скорость трансляции более 1000 строк в минуту.

MACRO-80 выполняется в операционной системе CP/M.

MACRO-80 транслирует исходный файл в два прохода. Во время прохода 1, MACRO-80 оценивает операторы программы, подсчитывает размер сгенерированного кода, строит таблицу символов, где всем символам присваиваются значения, и расширяет вызовы операторов макро. Во время прохода 2, MACRO-80 присваивает символам и выражениям значения из таблицы символов, снова расширяет операторы вызова макросов, и создает перемещаемый код. MACRO-80 проверяет значения символов, выражений и макросы во время обоих проходов. Если значение во время прохода 2 отличается от значения во время прохода 1, MACRO-80 возвращает код фазовой ошибки.

Для запуска MACRO-80 дискета, которая содержит MACRO-80, должна быть вставлена в надлежащий дисковод. Дискета, на которой находится созданный Вами исходный файл также должна быть в дисководе.

## 5.1 Вызов MACRO-80

Для вызова MACRO-80, введите:

**M80**

Будет загружен файл программы M80.COM. MACRO-80 выведет на экран звездочку (\*), чтобы указать, что ассемблер готов принять командную строку.

## 5.2 Командная строка MACRO-80

Командная строка в MACRO-80 состоит из четырех полей, которые называются:

**Object,List=Source/Switch**

Командная строка может быть введена в ее собственной строке, или вместе с командой M80. (Если M80 и командная строка будут введены вместе, то MACRO-80 не выводит подсказку звездочка.) Ввод командной строки в ее собственной строке позволяет в конфигурации с одним диском использовать MACRO-80. Кроме того, вводя M80 и командную строку отдельно, Вы в состоянии компилировать другую программу, не вызывая MACRO-80 еще раз. Когда трансляция закончится MACRO-80 возвратит запрос звездочка (\*) и ожидают ввода другой командной строки. Чтобы выйти из MACRO-80, если Вы ввели M80 и отдельно командную строку, введите <Ctrl-C>.

Если Вы выполняете только одну компиляцию, ввод командной строки вместе с M80 удобен. Этот способ компактней и позволяет операции ассемблирования быть частью команды SUBMIT. Когда Вы вводите M80 и командную строку вместе, MACRO-80 автоматически выходит в операционную систему.

**Примечание**

Если Вы вводите M80 и командную строку отдельно, Вы должны вводить командную строку только в верхнем регистре. Если Вы не сделаете это MACRO-80 возвратит сообщение об ошибке ?Command. Если Вы вводите M80 и командную строку одновременно, поля могут вводится в верхнем или в нижнем регистре (или смешанном виде), потому что CP/M преобразует все поля в верхний регистр прежде, чем передать программе.

**Source (=filename)**

Чтобы компилировать Вашу исходную программу, Вы должны ввести, по крайней мере, знак равенства (=) и имя исходного файла.

=filename указывает имя исходного файла, который вы хотите компилировать. Если исходный файл не находится на текущем диске, Вы должны включать обозначение диска как часть имени файла. Если имя файла-источника вводится без расширения, MACRO-80 предполагает, что расширение .MAC. Если расширение файла не является .MAC, Вы должны добавить расширение к имени файла. Для других возможностей обозначений диска/устройства и расширений файла, см. раздел Дополнительные параметры командной строки ниже.)

Поле Source является единственно необходимым полем, кроме M80.

Простейшая команда:

**M80 =Source**

Эта команда направляет MACRO-80 для компиляции исходного файла и сохранения результата в перемещаемый объектный файл (так называемый файл REL) с тем же именем, что и исходный файл. Если исходный файл является NEIL.MAC, то командная строка:

**M80 =NEIL**

генерирует скомпилированный файл под названием NEIL.REL.

Дополнительной опцией является ввод только запятой (,) слева от знака равенства. Когда MACRO-80 видит запятую в качестве первого поля после M80, он подавляет создание всех выходных файлов (Object и List). Командная строка

**M80 ,=NEIL**

вызывает MACRO-80 для компиляции файла NEIL.MAC, но выходные файлы не создаются. Программисты используют эту командную строку для проверки синтаксиса в исходной программе перед сохранением скомпилированных программ. Поскольку файлы не генерируются, сборка заканчивается быстрее и ошибки известны раньше.

**Object (filename)**

Поле Object не обязательное. Однако, определенные обстоятельства заставят Вас заполнять поле Object.

Скомпилированная программа сохраняется на диске в объектный файл (Object). LINK-80 использует объектный файл, чтобы создать исполняемую программу. Если поля Object и List будут опущены в командной строке (как в =Source), то MACRO-80 создаст объектный файл с тем же именем, что у исходного, но с расширением по умолчанию .REL. Если Вы хотите, чтобы у Вашего объектного файла было имя, отличающееся от исходного, Вы должны ввести имя файла в поле Object. MACRO-80 добавит расширение файла. REL, если Вы не введете расширение.

Кроме того, если Вы хотите создать файлы List и REL, Вы должны ввести имя файла в поле Object, даже если хотите чтобы файл REL, имел имя исходного файла. Если Вы введете имя файла в поле List, но опустите Object, то файл REL не будет создан. Программисты используют эту функцию для того, чтобы проверить программу на ошибки перед окончательной компиляцией. Распечатка программ помогает отладке.

Имя для объектного файла может совпасть с именем исходного файла, или иметь любое другое допустимое имя, по Вашему усмотрению. Так как удобно, чтобы все файлы, относящиеся к программе имели некоторую взаимную индикацию относительно их родства, чаще всего Вы захотите присвоить Вашему объектному файлу тоже имя, что у исходного файла.

**List (,filename)**

Поле List не обязательное. Запятая требуется перед всеми полями List. Если Вам необходим листинг файлов, введите имя файла в поле List. (Есть альтернатива этому правилу. См. раздел Ключи ниже для обсуждения ключей /L.)

MACRO-80 по умолчанию присваивает расширение .PRN файлу листинга, если Вы не определяете другое расширение в поле List.

Командная строка:

**M80 ,NEIL=NEIL**

компилирует файл NEIL.MAC (исходный файл) и создает файл листинга NEIL.PRN. Объектный (REL) файл не создается.

Имя может совпасть с именем исходного файла, или иметь любое другое допустимое имя, по Вашему усмотрению. Так как удобно, чтобы все файлы, относящиеся к программе имели некоторую взаимную индикацию относительно их родства, чаще всего Вы захотите присвоить Вашему файлу листинга тоже имя что у исходного файла.

Не вводите только запятую в поле List после ввода имени файла в поле Object. Например:

**M80 NEIL,=NEIL**

MACRO-80, вероятно, проигнорирует запятую и соберет исходный файл в файл REL. Возможно, что MACRO-80 может вывести сообщение COMMAND ERROR.

Если Вы вводите только запятую в поле List, и ничего в поле Object, MACRO-80 скомпилирует исходный файл, но не будет создавать выходные файлы. Эта команда

**M80 ,=Source**

позволяет Вам проверять исходную программу на синтаксические ошибки прежде, чем сохранить скомпилированную. программу в файле на диске. Хотя MACRO-80 всегда проверяет код на наличие ошибок, эта форма команды обеспечивает гораздо более быстрее компилирование, так как выходные файлы не создаются.

В конце компиляции, MACRO-80 напечатает сообщение:

[xx][No] Fatal errors [,xx warnings]

В этом сообщении указывается количество фатальных ошибок и ошибок предупреждения которые встречаются в программе. Сообщение выводится в конце каждой компиляции на экран терминала и в файл листинга. Когда появится сообщение, ассемблер закончил компиляцию. Вывод сообщения No Fatal Errors (нет фатальных ошибок) говорит об успешном завершении ассемблирования.

**Ключи (/Ключ)**

Вы можете дать команду MACRO-80 выполнять некоторые дополнительные функции помимо компиляции и создания объектных файлов и листинга. Эти дополнительные команды задаются в MACRO-80 в поле в конце командной строки. Поле ключа заставляет MACRO-80, "включить" некоторую дополнительную или альтернативную функцию; следовательно, эти записи вызывают ключами. Ключи это буквы, которым предшествует символ наклонной черты (/). Может быть введено, любое число ключей, но каждому ключу должна предшествовать наклонная черта. Например:

**M80 ,=NEIL/L/R**

Доступные ключи MACRO-80:

| **Ключ** | **Действие** |
| --- | --- |
| **/O** | Восьмеричный листинг. MACRO-80 генерирует список адресов файла в восьмеричной системе счисления. |
| **/H** | Шестнадцатеричный листинг. MACRO-80 генерирует список адресов файла в Шестнадцатеричной системе счисления. Это - значение по умолчанию. |
| **/R** | Заставляет генерировать объектный файла с тем же именем как исходный файл. Может использоваться вместо того, чтобы задавать имя файла в поле Object командной строки.  Этот ключ удобен, когда Вы хотите файл REL, но забыли ввести имя файла в поле Object и ввели запятую и имя файла, или запятую только в поле List.  (Помните: если никаких имен файлов, или запятая не введены перед знаком равенства будет сгенерирован файл REL. )  Таким образом, если Вы введете  M80 ,NEIL=NEIL или M80 ,=NEIL  и прежде чем нажмете <ENTER> решите, что Вы хотите файл REL, просто добавьте /R. Командная строка могла бы быть:  M80 ,NEIL=NEIL/R или M80 ,=NEIL/R |
| **/L** | Заставляет генерировать файл листинга с тем же именем как исходный файл. Может использоваться вместо того, чтобы задавать имя файла в поле List командной строки.  Этот ключ удобен, когда Вы хотите файл листинга, но забыли ввести имя файла в поле List. Если Вы вводите командную строку:  M80 =NEIL  или M80 ,=NEIL  или M80 NEIL=NEIL  и прежде чем нажмете <ENTER> решите, что Вы действительно хотите файл листинга, просто добавьте /L. Командная строка могла бы быть:  M80 =NEIL/L  или M80 ,=NEIL/L  или M80 NEIL=NEIL/L |
| **/C** | вызывает MACRO-80, чтобы создать специальный файл листинга (с тем же именем, что и исходный файл) для использования с программой построения отчета о перекрестных ссылках CREF-80. Если вы хотите использовать CREF-80, вы должны компилировать свой файл с добавлением этого ключа. См. Главу 7, CREF-80 Программа построения отчета о перекрестных ссылках, для уточнения деталей. |
| **/Z** | Направляет MACRO-80, чтобы компилировать коды операций Z80. Если Ваш исходный файл содержит коды операций Z80 и если Вы не включаете псевдокоманду .Z80 в Ваш исходный файл, то Вы должны использовать ключ /Z во время компиляции, чтобы MACRO-80 собрал коды операций Z80. |
| **/I** | Направляет MACRO-80, чтобы компилировать коды операций 8080. Если Ваш исходный файл содержит коды операций 8080 и если Вы не включаете псевдокоманду .8080 в Ваш исходный файл, то Вы должны использовать ключ /I во время компиляции, чтобы MACRO-80 собрал коды операций 8080. (По умолчанию). |
| **/P** | Каждый ключ /P выделяет дополнительные 256 байт стекового пространства для использования во время компиляции. Используйте ключ /P, если возникают ошибки переполнения стека во время компиляции. В противном случае не требуется. |
| **/M** | Ключ /M инициализирует области блока данных. Если вы хотите, чтобы область, которая определяется псевдокомандой DS (Define Space) инициализировалась нулями, то вы должны использовать в командной строке ключ /M. То есть, DS автоматически не инициализирует пространство нулями, и в этом случае вы не можете знать, что хранится в пространстве DS, или как это повлияет на программу. |
| **/X** | Ключ /X устанавливает по умолчанию текущую настройку для подавления вывода блоков листинга условного ассемблирования, значение которых есть "ложь". Отсутствие ключа /X в командной строке устанавливает по умолчанию текущую настройку для подавления вывода листинга условного ассемблирования , значение которых есть "ложь". Ключ /X часто используется в сочетании с условным объявлением псевдокоманды .TFCOND. Для уточнения см. Список Псевдокоманд в Главе 4. |

**Дополнительные поля командной строки**

Каждое поле командной строки поддерживает два дополнительных типа записей - обозначение устройства и расширение файла. Эти два типа записей - фактически часть "спецификации файла". Спецификация файла включает устройство, на котором расположен файл, имя файла и расширение.

Обычно, расширения файлов и обозначения устройств обрабатываются по умолчанию - программа MACRO-80 "вставляет" эти записи, если их позиции остаются незаполненными в командной строке. Назначения по умолчанию, никоим образом не мешает вам ввести расширения файлов или обозначения устройств, включая записи, соответствующие полям по умолчанию. Программист может ввести или опустить эти дополнительные поля в любой комбинации.

Формат спецификации файла в MACRO-80:

**dev:filename.ext**

|  |  |  |
| --- | --- | --- |
| где: | **dev:** | 1-3 буквы обозначения устройства с последующим (обязательным) двоеточием. |
|  | **filename** | 1-8 букв имени файла. |
|  | **.ext** | 1-3 буквы расширения файла, которым предшествует (обязательно) точка. |

**Расширение имен файлов (**.ext**)**

Чтобы различать исходный файл, объектный файл и файл листинга, MACRO-80 добавляет расширение каждому имени файла. Расширения файла - мнемоника из трех букв, добавленная к имени файла с точкой (.) между именем файла и расширением. Расширение, которое добавляет MACRO-80, отражает тип файла. Так как расширения назначает MACRO-80, их вызывают расширениями по умолчанию. Расширения по умолчанию, которые назначает MACRO-80:

**.REL** перемещаемый объектный файл

**.PRN** файл листинга

**.COM** абсолютный (исполняемый объект) файл

Кроме того, MACRO-80 предполагает, что если расширение файла не введено, исходный файл имеет расширение .MAC.

Вы можете использовать свои собственные расширения, если считаете это необходимым или желательным. Недостаток заключается в том, что каждый раз, когда Вы обозначаете файл, должны всегда не забывать добавлять свое расширение. Кроме того, Вы должны помнить, какой это файл - перемещаемый, исходный, исполняемый, и т.д. Преимущество разрешений присваиваемых MACRO-80 по умолчанию состоит в том, что у Вас всегда есть мнемоническая индикация типа файла, и Вы можете, в большинстве случаев, обозначать имя файла без расширения.

**Обозначения устройств (**dev:**)**

Каждое из полей в командной строке (кроме вызова) также может включать в себя обозначение устройства.

Когда обозначение устройства определено в поле Source, обозначение говорит MACRO-80, где искать исходный файл. Когда обозначение устройства определено в полях Object или List, обозначение говорит MACRO-80, куда вывести объектный файл или файл листинга. Если обозначение устройства опущено в каком-либо из этих полей, MACRO-80 использует (значения по умолчанию) текущий диск. Таким образом, не обязательно указывать текущий диск или устройство.

Важно включать обозначения устройства, если несколько устройств или дисков будут использоваться во время компиляции. Например, если Ваша дискета с компилятором находится в диске A, а дискета с исходной программой находится на диске B, и Вы хотите, чтобы выходной файла REL был записан на B, необходимо ввести командную строку:

**M80 =B:NEIL**

Во время вывода файла REL, текущим диском является диск B. (Однако, когда программа MACRO-80 будет завершена, диск A снова станет текущим диском.) Напротив, если Вы сохраняли свою исходную программу на дискетах MACRO-80 в диске A и хотите, направить вывод файла REL на дискету в диске B, тогда Вы должны ввести командную строку:

**M80 B:=A:NEIL**

Как показывает опыт, если Вы не уверены, нужно ли включать обозначение устройства (особенно обозначение диска), введите обозначение. Это один из верных способов получить необходимые файлы в нужных местах.

Доступные обозначения устройства для MACRO-80:

|  |  |
| --- | --- |
| A:, B:, C:,... | Дисководы |
| LST: | Принтер |
| TTY: | Экран терминала или клавиатура |
| HSR: | Устройство высокоскоростного чтения |

**Обозначения устройств в виде имен файлов**

Как вариант, Вы можете ввести обозначение устройства только в поля командной строки вместо имени файла. Использование этой опции дает различные результаты в зависимости какое указано устройство, и в каком поле. Например:

**M80 ,TTY:=TTY:**

позволяет Вам сразу скомпилировать строку на вводе, чтобы проверить на синтаксис, или другие ошибки. Модификация этой команды (т.е. M80 ,LST:=TTY:), обеспечивает тот же результат, но распечатанный на принтере вместо экрана терминала.

Если Вы используете любую из этих команд (,TTY:=TTY: или ,LST:=TTY:), Ваша первая запись должна быть оператором END. Вы должны перевести компилятор в проход 2 прежде, чем он выдаст код. Если Вы просто начнете вводить строки операторов без ввода первоначального END, то Вы не получите ответа, пока не будет введен оператор END. Тогда Вы должны будете повторно ввести все свои операторы прежде, чем Вы увидите сгенерированный код.

В следующей таблице приведены результаты различных вариантов. Таблица призвана показать возможности, а не дать исчерпывающий перечень комбинаций.

|  |  |  |  |
| --- | --- | --- | --- |
| dev: | Object | List | =Source |
| A:, B:, C:, D: | Запись файла на указанный диск | Запись файла на указанный диск | N/A (должно быть указано имя файла) |
| HSR: | N/A (только вход) | N/A (только вход) | Чтение исходной программы с высокоскоростного считывателя |
| LST: | N/A (нечитаемый формат файла) | Вывод листинга на принтер | N/A (только выход) |
| TTY: | N/A (нечитаемый формат файла) | Вывод листинга на экран | "Чтение" исходной программы с клавиатуры |

**Рисунок 5.1: Эффекты обозначений устройства без имен файлов**

## 5.3 Формат файлов листинга МАКРО-80

Файл листинга MACRO-80 содержит две части файла в различных форматах. В одном формате выводятся на экран строки файла. Во втором формате выводится на экран список таблицы символов.

**Формат файла**

Каждая страница листинга MACRO-80 в первых двух строках печатает данные заголовка. Если в исходном файле не были указаны данные заголовка (не заданы псевдокоманды TITLE и SUBTTL), эти части строки заголовка будут пустыми.

Формат:

**[TITLE text] M80 z.zz PAGE x**

**[SUBTTL text]**

|  |  |  |
| --- | --- | --- |
| где: | **TITLE text** | это текст, заданный псевдокомандой .TITLE, если .TITLE была включена в исходный файл. Если в исходном файле псевдокоманда .TITLE отсутствует, это пространство остается пустым. |
|  | **z.zz** | является номером версии вашей программы MACRO-80. |
|  | **x** | номер страницы, и увеличивается только тогда, когда в исходном файле встречается псевдокоманда .PAGE, или каждый раз, когда заполнен текущий размер страницы. |
|  | **SUBTTL text** | это текст, заданный псевдокомандой .SUBTTL, если .SUBTTL была включена в исходный файл. Если в исходном файле псевдокоманда .SUBTTL отсутствует, это пространство остается пустым. |

За строкой заголовка следует пустая строка. Текст файла листинга начинается на следующей строке.

Формат строки листинга:

**[error] ####m xx xxxxm[w] text**

|  |  |  |
| --- | --- | --- |
| где: | **error** | Представляет код ошибки из одной буквы. Код ошибки распечатывается, только если строка содержит ошибку. Иначе, пространство остается незаполненным. |
|  | **####** | Представляет счетчик адреса. Число - 4-разрядное шестнадцатеричное или 6-разрядное восьмеричное. Основание числа счетчика адреса определяется в поле ключей /O или /H в командной строке MACRO-80. Если ключ основания не был задан, основание по умолчанию шестнадцатеричное (4-цифры). |
|  | **m** | Представляет символ индикатора перемещаемости. Возможные символы:  ' программный перемещаемый;  " данных перемещаемый;  ! COMMON перемещаемый;  <пробел> абсолютный;  \* внешний. |
|  | **xx** и **xxxx** | Представляют сгенерированный код. Поле xx представляет однобайтовое значение. Однобайтовые значения всегда сразу сопровождаются пробелом. Поле xxxx представляет двухбайтовое значение с байтом старшего разряда, распечатанным сначала (это - противоположный порядок, в котором они сохранены). Двухбайтовые значения сопровождаются одним из индикаторов режима, обсужденных выше (обозначенный вторым m). |
|  | **[w]** | Представляет собой строку, в файле MACRO-80, которая включена из другого файла с помощью псевдокоманды INCLUDE, или строки, которая является частью расширения (MACRO, REPT, IRP, IRPC). Для строки из оператора INCLUDE, после сгенерированного кода печатается символ C, для строк расширения, печатается знак плюс (+) после сгенерированного кода. В противном случае, это пространство пустое.  Текст представляет остальную часть строки, в том числе метки, код команды, аргументы и комментарии. |

**Формат таблицы символов**

Страница листинга с таблицей символических имен имеет тот же формат заголовка, что и страницы строк файла. Однако, вместо номера страницы, страница таблицы символов выводит PAGE S.

Затем в листинге выводятся в алфавитном порядке все имена макро встречающиеся в программе. Затем, все символические имена также в алфавитном порядке. За каждым символом следует символ табуляции (tab), затем значение символического имени. Каждое значение символического имени сопровождается одним из следующих символов:

|  |  |
| --- | --- |
| **I** | Имя PUBLIC |
| **U** | Неопределенное имя |
| **C** | Имя COMMON блока. Значение, показанное для имени COMMON блока, является его длиной в байтах в шестнадцатеричном или восьмеричном формате. |
| **\*** | Внешнее имя |
|  | <пробел> Абсолютное имя |
| **'** | Перемещаемый код |
| **"** | Перемещаемые данные |
| **!** | Перемещаемый COMMON |

## 5.4 Коды ошибок и сообщения

Ошибки, которые встречаются во время компиляции, заставляют MACRO-80 возвращать или код ошибки или сообщение об ошибке. Коды ошибки - флаги из одного символа, распечатанные в первом столбце файла листинга. Если файл листинга не будет распечатан на экране терминала, то строки, содержащие ошибки, все равно будут распечатаны. Сообщения об ошибках приводятся в конце файла листинга, или, если файл листинга не будет выведен на экране терминала, то любые сообщения об ошибках будут выведены на экран после строк с кодами ошибок.

|  |  |
| --- | --- |
| Код ошибки | Значение |
| **A** | **Ошибка в аргументе.**  Неправильный формат аргумента псевдокоманды или аргумент вне допустимых границ. |
| **C** | **Ошибка условного вложения.**  ELSE без IF, ENDIF без IF, два ELSE на один IF. |
| **D** | **Дважды определенное символическое имя.**  Ссылка на символическое имя, у которого есть больше чем одно определение. |
| **E** | **Ошибка при использовании EXTRN.**  Использование EXTRN недопустимо в отмеченном контексте. Например, FOO SET NAME или LXI B,2-NAME. |
| **M** | **Многократно определенное символическое имя.** |
| **N** | **Ошибка в числе.**  Ошибка в числе, обычно плохая цифра. Например, 8Q. |
| **O** | **Неверный код операции или нежелательный синтаксис.**  ENDM, LOCAL вне блока; SET, EQU или MACRO без имени, неверный синтаксис в коде команды, неверный синтаксис в выражении (например, несоответствие круглых скобок, кавычек, последовательности операторов). |
| **P** | **Ошибка фазы.**  Значение метки или имени EQU отличается проходить в течении 2 прохода от его значения в 1. |
| **Q** | **Сомнительный.**  Обычно, строка не завершена должным образом. Например, AX MOV, BX. Это - ошибка предупреждения. |
| **R** | **Перемещение.**  Недопустимое использование перемещения в выражении, таком как abs-rel. Данные, код и области COMMON перемещаемы. |
| **U** | **Неопределенное символическое имя.**  Символическое имя, на которое ссылаются в выражении, не определено. Для некоторых псевдокоманд ошибка V выводится на проходе 1, а ошибка U на проходе 2. Сравните с определением кода ошибки V ниже. |
| **V** | **Ошибка значения.**  На проходе 1 псевдокоманда, которой требуется значение (например, .RADIX, .PAGE, DS, IF, IFE), имеет неопределённое значение. Если символ, встретившийся в псевдокоманде, будет определен в программе позже, то ошибка U не будет появляться в листинге на проходе 2. |

**Сообщения об ошибках**

%No END statement

Нет оператора END, или он отсутствует, или не проанализировано, потому что находится в ложном условном выражении, незавершенном блоке IRP/IRPC/REPT или прекращенном макросе.

Unterminated conditional

По меньшей мере один условный блок не завершен до конца файла.

Unterminated REPT/IRP/IRPC/MACRO

По крайней мере один блок не завершён.

Symbol table full

При создании MACRO-80 таблицы символических имен, доступная память была исчерпана. Самая обычная причина - большое количество макро-блоков, которые также содержат операторы из большого количества строк операторов. Макро-блоки сохраняются в таблице символических имен дословно, включая комментарии, добавленные к строкам в макро-блоке. Вы должны проверить все макро-блоки в исходной программе. Чтобы исключить комментарии в макро-блоках из таблицы символических имен, поместите перед этими комментариями дважды символ точки с запятой (;;). Этот метод должен освободить достаточно места, чтобы собрать Вашу программу.

[xx][No] Fatal errors [,xx warnings]

Число фатальных ошибок и предупреждений об ошибках встречалось в программе. Сообщение выводится после каждой компиляции на экране терминала и в файле листинга. Когда сообщение появляется, компиляция закончена. Если выводится сообщение, No Fatal Errors, компиляция завершена и успешно.

# Глава 6 LINK-80 Связывающий загрузчик

Файлы .REL, которые создает MACRO-80, не являются исполняемой программой. Чтобы сделать исполняемую программу из файла REL, Вы должны загрузить и соединить файл REL с помощью программы редактора связей LINK-80. В результате будет создан исполняемый объектный файл.

Загрузка означает физическое размещение файла в память и присвоение абсолютных адресов для данных и кода вместо относительных адресов, назначенных ассемблером. Это один из необходимых шагов для преобразования перемещаемого файла (REL) в исполняемый файл (COM).

Связывание означает, что каждый загруженный файл (или модуль), который направляет выполнение программы вне себя (с помощью вызова CALL, EXTERNAL ссылки или INCLUDE), будут "привязаны" к модулю, который содержит соответствующий код.

LINK-80 может также сохранить собранную и соединенную программу как исполняемую объектную программу в файле на диске с расширением .COM. Следовательно, в любое время, когда Вы захотите выполнить свою программу, нужно только вставить диск, который содержит Ваш файл COM в надлежащий дисковод, и "вызвать" свою программу с помощью простого процесса ввода имени файла, под которым Вы раньше сохранили программу, сопровождаемого возвратом каретки.

## 6.1 Вызов LINK-80

Для вызова LINK-80 введите:

**L80**

Программный файл L80.COM будет загружен. LINK-80 выведет звездочку (\*), это означает, что связывающий загрузчик готов принять команду. Файл(ы) REL которые необходимо добавить в создаваемую программу должны находится на диске. Если у вас есть только один диск, вам придётся менять дискеты в дисководе на каждом шаге процесса загрузки.

## 6.2 Команды LINK-80

Командами LINK-80 являются имена файлов и ключи.

Вы можете вводить свои команды LINK-80 по одной за раз, или все команды (в том числе L80) в одну строку.

Командная строка, имеет гибкий формат, предлагая Вам несколько вариантов для загрузки и связывания файлов и для выполнения других операций. Основное правило для команд LINK-80 состоит в том, что файлы загружаются по порядку, в CP/M они загружаются, начиная с адрес 103H (по умолчанию). Даже если эти файлы будут загружены в произвольном порядке, вы не должны пытаться вводить файлы в порядке их исполнения. LINK-80 размещает инструкцию перехода по адресу 100H-102H, которая осуществляет переход к начальному адресу первой исполняемой инструкции, независимо от ее местоположения в памяти.

LINK-80 может выполнять около одиннадцати различных задач. Хотя можно использовать их все, вы редко будете использовать больше чем три или четыре одновременно.

Когда Вы вводите команду LINK-80, LINK-80 возвращает подсказку звездочка (\*), которая говорит Вам о возможности вводить другую команду. Например:

A>L80<RETURN>

\*/switch<RETURN>

\*filename<RETURN>

\*/switch<RETURN>

\*filename/switch<RETURN>

\*/E<RETURN> (для выхода из LINK-80)

Обратите внимание на то, что все вышеупомянутые строки могут быть введены в одной строке. Например:

L80 /switch,filename/switch,filename/switch/E<RETURN>

Это показывает дополнительную гибкость командной строки LINK-80.

Хотя ввод каждой команды в отдельной строке является медленным и утомительным, это дает преимущество, особенно если Вы плохо знакомы со связывающим загрузчиком, так как вы в любое время знаете какую функцию выполняет LINK-80.

### 6.2.1 Имена файлов

Файлы, обработанные LINK-80, являются файлами REL. Команды LINK-80 с именем файла, служат для загрузки файла с этим именем (также называемого модулем). Если какой-либо файл уже был загружен, имя файла говорит LINK-80, чтобы соединить загруженные файлы как требуется.

Обычно каждый сеанс связывания требует по крайней мере двух имен файлов. Одно имя файла с расширением REL используется LINK-80 для загрузки и связывания, другое необходимо LINK-80, чтобы сохранить исполняемый код в файле с указанным именем.

Если ввести только одно имя файла во время сессии компоновки, либо COM-файл не сохраняется (в этом случае вы, возможно, зря потратили время), или LINK-80 вернет сообщение об ошибке.

?NOTHING LOADED

Заметим, однако, что если вы введете только одно имя файла c последующим ключом /G, COM файл не будет сохранен, но после окончания компоновки программой LINK-80 программа будет запущена на выполнение. (См. описание ключей в следующем разделе).

Вы можете ввести столько файлов, сколько поместится в одной строке. Названные файлы могут быть файлами REL на различных языках (BASIC, COBOL, FORTRAN или ассемблере) или файлами REL библиотеки времени выполнения для любого из высокоуровневых языков программирования. (Для уточнения работы с файлами REL высокоуровневого языка см. Руководство продукта, поставляемое с компилятором высокоуровневого языка.)

Когда LINK-80 закончится, результаты будут сохранены в файле, заданном программистом в командной строке (имя файла, сопровождаемое ключом /N - см. ниже, Раздел 6.2.2, Ключи). LINK-80 присвоит этому файлу расширение .COM.

Команда имени файла в LINK-80 фактически означает спецификацию файла. Спецификация файла включает обозначение устройства, имя файла и его расширение. Формат спецификации файла:

**dev:filename.ext**

LINK-80 по умолчанию в качестве устройства использует устройство назначенное по умолчанию или текущий диск. По умолчанию LINK-80 для входных файлов использует расширение .REL и выходных файлов расширение .COM. Вы можете изменить назначенное устройства на любое допустимое устройство вывода, поддерживаемое MACRO-80 и/или расширение файла на любые три символа, определяя устройство или расширение файла, когда вводите команду имени файла.

### 6.2.2 Ключи

Команды ключей в LINK-80 необходимы, чтобы выполнить функции помимо загрузки и соединения. Ключи это буквы, которым предшествуют символ наклонной черты (/). Вы можете в одной командной строке поместить столько ключей, сколько Вам необходимо, но каждой букве ключа должен предшествовать символ наклонной черты (/). Например, если вы хотите скомпоновать программу под названием NEIL, сохранить ее образ на дискете, затем выполнить программу, нужно два имени и два ключа, так что вы должны ввести команду:

**NEIL,NEIL/N/G<RETURN>**

LINK-80 экономит место на диске (ключ /N), а затем запускает программу NEIL (ключ /G).

Некоторые ключи могут быть введены сами по себе (/E, /G, /R, /P, /D, /U, /M, /O, /H). Некоторые ключи должны быть добавлены к имени файла, на которое они влияют (/N, /S). Некоторые ключи работают, только если другие ключи также введены во время сеанса LINK-80 (/X, /Y). Некоторые ключи должны предшествовать любым именам файлов, которые необходимо изменить (/P, /D). Некоторые действия команды ключей, будут отложены до конца сеанса LINK-80 (/N, /X, /Y). Некоторые команды ключей, которые оказывают действие с места когда введены (/S, /R - имя файла, введенное без ключа, действуют также). Эти "правила поведения" должны быть учтены при вводе команд LINK-80. См. Описания для каждого ключа для получения полной информации о его действии.

В нижеприведенной таблице перечислены функции ключей. Полное описание функций ключей следуют за таблицей.

**Будьте осторожны**: не путайте ключи LINK-80 с ключами MACRO-80.

|  |  |  |
| --- | --- | --- |
| **Функции** | **Ключи** | **Действие** |
| Выполнение | /G  /G:Name | Выполнить .COM файл, затем выйти в операционную систему.  Назначить начальному адресу .COM файла значение Name, выполнить .COM файл, затем выйти в операционную систему. |
| Выход | /E | Выйти в операционную систему. |
| /E:Name | Назначить начальному адресу .COM файла значение Name, затем выйти в операционную систему. |
| Сохранение | /N | Сохранить все ранее загруженные программы и подпрограммы, используя имя файла, расположенное перед ключом /N. |
| /N:P | Альтернативная форма /N, сохранить только область программы. |
| Назначение адресов | /P | Назначить начальный адрес для программ и данных. Если установлены /D, /P назначить начальный адрес только программе. |
| /D | Назначить начальный адрес только области данных. |
| /R | Перегрузить LINK-80. |
| Поиск в библиотеке | /S | Поиск в библиотеке имя которой предшествует ключу /S. |
| Глобальный листинг | /U  /M | Список неопределенных глобальных символов.  Полный список глобальных символов. |
| Настройка основания | /O  /H | Восьмеричное основание.  Шестнадцатеричное основание (по умолчанию). |
| Специальные коды | /X | Сохранить файл "COM" в ASCII Intel шестнадцатеричном формат. Требуется ключ /N. Присваивает файлу "COM" расширение .HEX. |
| /Y | Создает специальный файл для использования с отладчиком SID/ZSID. Требует ключи /E и /N. Присваивает специальному файлу расширение .SYM. |

Рисунок 6.1: Таблица ключей LINK-80

По крайней мере два ключа будут, вероятно, использоваться в каждом сеансе компоновки. Эти ключи принадлежат к первым трем функциям - выполнение, выход, и сохранение.

**Выполнение**

| Ключ | Действие |
| --- | --- |
| **/G** | Ключ /G вызывает LINK-80, для загрузки имен файлов, введенных в командной строке, компоновке программы, затем выполнения собранной программы. После окончания работы программы, компьютер возвратится на уровень команд операционной системы. Например,  **L80 NEIL,NEIL/N/G**  собирает NEIL.REL, сохраняет результат в файл на диске с именем NEIL.COM, затем выходит в операционную систему.  Выполнение происходит сразу после интерпретации командной строки. Непосредственно перед выполнением, LINK-80 выводит три числа и сообщение BEGIN EXECUTION (начало выполнения). Эти три числа могут быть очень полезными для Вас в разработке будущих программ на языке ассемблера. Первое число - начальный адрес программы. Второе число - адрес следующего доступного байта, т.е. конечный адрес плюс один байт. Третье число - число 256-байтовых страниц, занимаемые программой (разница между начальным и конечным адресом, преобразованная в 256-байтовые страницы).  Если Вы не хотите сохранять .COM файл, используйте ключ /G и введите только одно имя файла в командной строке. Например:  L80 NEIL/G  Но помните, файл COM не создается (так как Вы не использовали ключ /N). Чтобы выполнить программу снова, Вам придётся выполнить LINK-80 еще раз. |
| **/G:Name** | Ключ /G:Name выполняет тоже действие, что и простой ключ /G, но с одной дополнительной функцией. <name> представляет собой глобальный символ, который был определен ранее в одном из модулей, которые в настоящее время связаны между собой и загружены. Когда программа LINK-80 видит <name>, она использует <name> как начало программы и загружает адрес в соответствии с <name> в качестве метки в инструкцию перехода в адреса 100H-102H.  Значение этого ключа (и /E:<name> ниже), указывает программе LINK-80, с какого места запустить выполнение, когда собранные модули не дают ясно понять это. Обычно это не проблема, потому что Вы соединяете модули в исполняемую программу на языке высокого уровня (из которых LINK-80 по умолчанию выбирает основную программу), или Вы соединяете только модули ассемблера, и у только одного из них есть оператор END <name>, чтобы сигнализировать LINK-80 которую из программ на языке ассемблера выполнять первой. Но если два или более модулей ассемблера содержат оператор END <name>, или если ни один из модулей ассемблера не содержит оператор END <name>, тогда ключ /G:<name> говорит LINK-80 использовать этот модуль в качестве начальной точки для выполнения.  Программисты, которые хотят выполнить модуль ассемблера перед программой высокоуровневого языка, должны использовать оператор CALL или INCLUDE в начале программы высокоуровневого языка, чтобы вызвать выполнение модуля ассемблера до выполнения программы языка высокого уровня. |
| **Выход** | |
| **/E** | Используйте ключ /E, чтобы соединить и загрузить программу и выполнить некоторые другие действия с файлами (например, чтобы сохранить на дискете), в случае, если Вы не хотите запускать программу. Когда программа LINK-80 закончит задачи, она выйдет в операционную систему.  (Ключ /G является единственным ключом, который останавливает работу LINK-80.)  При окончании компоновки, LINK-80 выдает три числа: начальный адрес, адрес следующего доступного байта, число 256-байтных страниц. |
| **/E:name** | Ключ /E:<name> работает в точности, как ключ /E, но с одной дополнительной функцией. <name> представляет собой глобальный символ, который был определен ранее в одном из модулей, которые в настоящее время связаны между собой и загружены. Когда программа LINK-80 видит <name>, она использует <name> как начало программы и загружает адрес в соответствии с <name> в качестве метки в инструкцию перехода в адреса 100H-102H.  Значение этого ключа (и /G:<name> выше), говорят программе LINK-80, с какого места запустить выполнение, если собранные модули не дают ясно понять это. Обычно это не проблема, потому что Вы соединяете модули в исполняемую программу на языке высокого уровня (из которых LINK-80 по умолчанию выбирает основную программу), или Вы соединяете только модули ассемблера, и только у одного из них есть оператор END <name>, чтобы сигнализировать LINK-80 которую из программ на языке ассемблера выполнять первой. Но если два или более модуля ассемблера содержат оператор END <name>, или если ни один из модулей ассемблера не содержит оператор END <name>, тогда ключ /G:<name> говорит LINK-80 использовать этот модуль в качестве начальной точки для выполнения.  Программисты, которые хотят выполнить модуль ассемблера перед программой высокоуровневого языка, должны использовать оператор CALL или INCLUDE в начале программы высокоуровневого языка, чтобы вызвать этот порядок выполнения. |
| **Сохранение** | |
| **/N** | Ключ /N сохраняет скомпонованную программу в файле на диске. Важно, чтобы имя файла всегда было определено для ключа /N. Если Вы не определяете расширение, то расширение по умолчанию для сохраненного файла - .COM. Именем файла COM будет имя, к которому программист добавляет ключ /N. Ключ /N должен следовать сразу за именем файла, под которым Вы хотите сохранить результаты сеанса компоновки.  Ключ /N не действует, если ключ /E или /G не следует за ним.  Наиболее распространенная ошибка программистов при использовании ключа /N, когда они забывают, что должны определить по крайней мере два имени файлов. Одно имя файла, который будет соединен и другое имя, который будет сохранен. Поэтому, командная строка как минимум должна включать:  **L80 NEIL,NEIL/N/G**  Первое имя файла NEIL является файлом, который будет загружен и соединен, второе имя файла NEIL - имя для файла COM, в который будет сохранен результат сеанса компоновки.  Конечно, можно определять имена файлов в любом порядке. Например:  **L80 NEIL/N,ASMSUB1,ASMSUB2,BASPROG/G**  В этом случае LINK-80 будет загружать и связать файлы BASPROG, ASMSUB1 и ASMSUB2, а затем сохранит результат в файл с именем NEIL.  В этих двух примерах можно видеть, что имя файла, сопровождаемое ключом /N, используется не для загрузки, это только необходимое условие для выходного файла, вы также должны всегда задать хотя бы один входной файл.  Вы будете использовать этот ключ почти каждый раз, когда соединяете файл REL, потому что нет другого способа сохранить результат сеанса компоновки, и потому что не сохранение результата означает, что Вы должны снова выполнить компоновку, чтобы запустить программу. После сохранения на диск необходимо ввести только имя COM файла в командной строке операционной системы, чтобы запустить программу. |
| **/N:P** | По умолчанию LINK-80 сохраняет области программы и данных в файле COM. Если Вы хотите сохранить только область программы, чтобы уменьшить размер файлов на диске, используйте ключ /N в форме /N:P. С этим ключом будет сохранен только код программы. |
| Два из этих ключей (/N плюс либо /G или /E), все ключи которые требуются для большинства операций LINK-80. Некоторые дополнительные функции доступны с помощью использования других ключей, которые позволяют программистам манипулировать процессом LINK-80 более тонко. Ключи, включающие эти дополнительные функции сгруппированы в категории в зависимости от типа функции. Функция каждой категории определяют название категории.  **Назначение адресов** | |
| **/P** | Ключ /P используется, чтобы одновременно назначить начало и программе и данным. Если Вы не вводите переключатель /P, LINK-80, выполнит эту задачу автоматически, используя адрес по умолчанию и для программы, и для данных. (103H для CP/M)  Формат переключателя /P:  **/P:<address>**,  Значение адреса <address> должно быть задано в текущей системе счисления. По умолчанию в шестнадцатеричной. Ключ /P разработан, чтобы позволить Вам помещать сегменты программы (или кода) в адресах отличных от значения по умолчанию. Для ключа /P значение по умолчанию 103H.  **Помните:** ключ /P вступает в силу, с момента появления и не влияет на уже загруженные файлы. Так что не забудьте разместить ключ /P **перед** любым из файлов, которые вы хотите загрузить, начиная с указанного адреса. При использовании ключи /P и /D должны быть отделены от имени файла REL запятой. Например,  **L80 /P:103,NEIL,NEIL/N/E**  Ключ /P влияет прежде всего на код CSEG в Вашей программе на языке ассемблера. Если ключ /P будет задан без /D, то области и данных и программы (CSEG и DSEG) будут загружены в /P:<address>. Сначала будут загружены DSEG (и любые области COMMON). Если присутствуют оба ключа и /P и /D, то ключ /P устанавливает только начало области CSEG. Обычно, если ваши программы все CSEG, вы будете использовать ключи /P и /D вместе.  **Отметьте особенно**, что области ASEG не затронуты ключом /P. Делайте все возможное чтобы установить адрес /P вне любых областей ASEG, если Вы не хотите, чтобы области программ или данных переписали области ASEG.  Вы можете ввести больше чем один ключ /P во время одного сеанса компоновки, чтобы поместить различные сегменты программы (кода) в области, которые не расположены вплотную. LINK-80 автоматически поместит один сегмент программы (CSEG) после следующего. Вы можете оставить пространство между модулями.  Однако существуют некоторые ограничения на размещение модулей:   1. Убедитесь, что области программ не накладываются друг на друга. LINK-80 выведет предупреждающее сообщение об ошибке, если это произойдет. 2. Убедитесь, что области программы не разделены данными или зонами COMMON. Т.е. CSEG в 200H, DSEG в 300H, и другой CSEG в 400H недопустимы. LINK-80 сообщит о фатальной ошибке в этом случае.   Когда сессии загрузки закончится, LINK-80 хочет видеть сегмент памяти загруженный данными и COMMON и другой сегмент загруженный программным кодом. Сегменты кода могут иметь разрыв между модулями, пока сегмент данных не загружен в промежутке от начала модуля первого сегмента кода и в конце модуля последнего сегмента кода, и наоборот. Так, размещение DSEG модулей в 103H-115H, 150H-165H, 170H-175H, и CSEG модулей в 200H-250H, 300H-350H, 400H-450H является приемлемым. LINK-80 покажет данные между 103H и 175H и программы между 200H и 450H.  **Обратите внимание**, что любые промежутки которые вы оставите могут содержать данные или программный код из предыдущей программы. LINK-80 не инициализирует разрывы 0 или null. Это может привести к непредсказуемым результатам. |
| **/D** | Ключ /D устанавливает источник для DSEG и зон COMMON. Если Вы не вводите ключ /D, LINK-80, выполняет эту задачу автоматически, используя адрес по умолчанию и для данных и для программы. (103H для CP/M)  Формат ключа /D:  **/D:<address>**,  Значение адреса в ключе /D должно быть задано в текущей системе счисления. По умолчанию в шестнадцатеричной.  Ключ /D разработан, чтобы позволить Вам помещать данные и сегменты COMMON в адресах отличных от значения по умолчанию. Значение по умолчанию для ключа /D есть 103H. Ключ /D должен быть отделен от имен файлов REL запятой. Например,  **L80 /D:103,NEIL,NEIL/N/E**  Если ключ /P, используется с ключом /D, данные и области COMMON загружаются, начиная с адреса заданного в ключе /D. (начало программы будет загружено, по адресу указанному в ключе /P.) Это единственный случай, когда адрес, указанный в ключе /P является начальным адресом фактического кода программы.  **Помните:** Ключ /D вступает в силу, как только LINK-80 "видит" ключ, таким образом, ключ /D не оказывает никакого эффекта на программы или данные, которые уже загружены. Поэтому, важно поместить ключ /D (а также ключ /P) перед файлами, которые Вы хотите загружать для запуска с заданного адреса.  Вы можете ввести несколько ключей /D во время одного сеанса компоновки, чтобы поместить различные сегменты программы (кода) в адреса, которые не расположены вплотную. LINK-80 автоматически поместит один сегмент данных (DSEG) после следующего. Вы можете оставить пространство между модулями.  Однако существуют некоторые ограничения на размещение модулей:   1. Убедитесь, что области данных не накладываются друг на друга. LINK-80 выведет предупреждающее сообщение об ошибке, если это произойдет. 2. Убедитесь, что области данных не разделены областями программы. Т.е. DSEG в 200H, CSEG в 300H, и другой DSEG в 400H недопустимы. LINK-80 сообщит о фатальной ошибке в этом случае.   Когда сессия загрузки закончится, LINK-80 хочет видеть сегмент памяти загруженный данными и COMMON и другой сегмент загруженный программным кодом. Сегменты данных могут иметь разрыв между модулями, пока сегмент программы не загружен в промежутке от начала модуля первого сегмента данных и в конце модуля последнего сегмента данных, и наоборот. Так, размещение DSEG модулей в 103H-115H, 150H-165H, 170H-175H, и CSEG модулей в 200H-250H, 300H-350H, 400H-450H является приемлемым. LINK-80 покажет данные между 103H и 175H и программы между 200H и 450H.  **Обратите внимание**, что любые промежутки которые Вы оставите могут содержать данные или программный код из предыдущей программы. LINK-80 не инициализирует разрывы нулем или null. Это может привести к непредсказуемым результатам. |
| **Дополнительное примечание для ключей /P и /D**  Если Ваша программа будет слишком большой для загрузчика, вы иногда будете иметь возможность загружать ее в этом случае, если используете ключи /D и /P вместе. Этим способом Вы сможете загрузить программы и данные большего суммарного размера. Хотя LINK-80 загружает и соединяет, он строит таблицу, состоящую из пяти байт для каждой программы c относительной ссылкой. Устанавливая ключи /D и /P, Вы избавляете LINK-80 от необходимости создавать эту таблицу, что освобождает некоторую дополнительную память для работы.  Чтобы установить два ключа, посмотрите в конце файла листинга. Возьмите адрес, который Вы решили использовать для ключа /D (куда Вы хотите, чтобы начал загружаться DSEG), добавьте число общего количества данных, добавьте это число к 103H, добавьте еще число 100H+1, и результат должен быть /P: <address> для запуска области программы. Ключ /D должен быть установлен в 103H или выше (D:103). | |
| **/R** | Ключ /R "сбрасывает" LINK-80 в начальное состояние. LINK-80 сканирует командную строку, до начала выполнения функций команд. Как только LINK-80 видит ключ /R, все загруженные файлы игнорируются, LINK-80 сбрасывает себя, выводит приглашение звездочку (\*), показывающее, что LINK-80 работает и ждет от Вас, ввода в командной строке. |
| **Поиск в библиотеке** | |
| **/S** | Ключ /S используется LINK-80 для поиска в файле с именем указанным перед ключом процедур, подпрограмм, определений глобальных переменных, и так далее. В командной строке, имя файла с ключом /S должно быть отделено от остальной части командной строки запятыми. Например:  **L80 NEIL/N,MYLIB/S,NEIL/G**  Ключ /S используется только для поиска файлов библиотек, в том числе библиотек которые вы построили, используя программу обслуживания библиотек LIB-80 (см. Главу 8). |
| **Глобальный листинг** | |
| **/U** | Ключ /U говорит LINK-80 перечислять все неопределенные глобальные переменные. /U работает только в командных строках, которые не включают ключи /G или /E. Обратите внимание на то, что если программа содержит какие-либо неопределенные глобальные переменные, LINK-80 перечисляет их автоматически, если командная строка не содержит ключ /S (поиск в библиотеке). В этих случаях введите только ключ /U, и будет выведен список неопределенных глобальных переменных. Используйте Ctrl-S, чтобы приостановить вывод, если Вы хотите изучить часть списка, который исчез бы с экрана. Используйте Ctrl-Q, чтобы возобновить вывод.  Различные библиотеки времени выполнения обеспечивают определения для глобальных переменных, необходимые для выполнения программы на языке высокого уровня.  В дополнение к перечислению неопределенных глобальных переменных ключ /U заставляет LINK-80, вывести начало, конец и размер областей данных и программы. Эти области приводятся как одна общая область, если ключи /P и /D не установлены. Если ключи /P и /D установлены, начало, конец, и размер обеих областей выводятся отдельно. |
| /M | Ключ /M направляет LINK-80, чтобы вывести на экран все глобальные переменные, и определенные и неопределенные. Вывод не может быть отправлен на принтер. В выводе определенные глобальные переменные сопровождаются их значениями, а неопределенные глобальные переменные сопровождаются звездочкой (\*).  В дополнение к выводу всех глобальных переменных ключ /M направляет LINK-80, чтобы вывести начало, конец и размер областей данных и программы. Эти области выводятся как одна общая область, если ключи /P и /D не установлены. Если ключи /P и /D установлены, начало, конец, и размер обеих областей выводятся отдельно. |
| **Настройка основания** | |
| **/O** | Ключ /O устанавливает текущее основание системы счисления в восьмеричную. Если у вас есть необходимость использовать восьмеричные значения в вашей программе, введите ключ /O в командной строке. Если вы не можете придумать причину, чтобы переключиться в восьмеричную систему, то нет никаких причин, чтобы использовать этот параметр. |
| **/H** | Ключ /H сбрасывает текущее основание системы счисления в шестнадцатеричную. Шестнадцатеричное основание используется по умолчанию. Вы не должны задавать этот ключ в командной строке, если ранее не использовали ключ /O, и теперь хотите вернуться к шестнадцатеричному основанию. |
| **Специальные коды** | |
| **/X** | Ключ /X сохраняет "COM" файл в Intel ASCII HEX-формате. Ключ /X требует ключ /N, добавленный к тому же имени, что и /X. например:  L80 NEIL,NEIL/X/N/E  Файлу, который сохранен с ключом /X, присваивается расширение файла .HEX.  Основное применение ключа /X для подготовки программ, которые будут записаны в ПЗУ. Шестнадцатеричный формат был изначально разработан для облегчения переноса программ с одной машины на другую. Шестнадцатеричный формат обеспечивает лучшую проверку кода, по сравнению с объектным кодом. Кроме того, HEX-файл может быть отредактирован в некоторых продвинутых строчных редакторах. |
| **/Y** | Ключ /Y сохраняет файл в специальном формате для использования с символьными отладчиками компании Digital Research, SID и ZSID. Ключ /Y требует задание ключей /N и /E (не /G) в командной строке. Например:  L80 NEIL,NEIL/Y/N/E  Файлу, который сохранен с ключом /Y, присваивается расширение файла .SYM. Также будет сохранен файл COM. Таким образом, пример командной строки выше создает и NEIL.COM и NEIL.SYM.  Файл SYM содержит имена и адреса всех глобальных переменных, который позволяет Вам использовать символьные отладчики компании Digital Research SID и ZSID с файлом SYM. |

## 6.3 Сообщения об ошибках

Об ошибках, обнаруженных во время выполнения LINK-80 будет возвращать сообщения, которым предшествует символ ? или символа %. Нет возвращаемых кодов ошибок, поэтому, как только вы поймете смысл сообщения, распознавание ошибки должно быть легким.

?No Start Address

Нет стартового адреса. Был задан ключ /G, но была загружена не основная программа.

?Loading Error

Ошибка загрузки. Последний заданный для компоновки файл не был создан в правильном для LINK-80 формате.

?Out of Memory

Не хватает памяти. Для загрузки программы недостаточно памяти.

?Command Error

Ошибка в команде. Программа LINK-80 не может опознать команду.

?<filename> Not Found

Файл не найден. Имя файла <filename>, заданное в командной строке, не существует.

?Start Symbol - <name> - Undefined

Не определено имя начала программы. После ключа /E:Name или /G:Name символ Name, заданный в ключе, отсутствует в программе.

?Nothing Loaded

Было задано <filename> в /S или /E или /G, но ни один объектный файл не был загружен. Т.е. была предпринята попытка, поиска в библиотеке, завершения LINK-80, или выполнения программы, когда фактически ничто не было загружено. Например:

TEST/N/E

В результате "?Nothing Loaded", потому что TEST/N называет TEST.COM, но не загружен TEST.REL.

Чтобы загрузить файл, введите имя файла. Чтобы сохранить файл, введите имя файла, сопровождаемое ключом /N и ключом /E или /G.

Например, любой из следующих наборов команд должен работать:

L80 NEIL,NEIL/N/E

или

L80

\*NEIL

\*NEIL/N/E

или

L80 NEIL/N,NEIL/E

?Can't Save Object File

Произошла ошибка при сохранении файла на диск. Как правило, это означает, что диск переполнен или защищен от записи.

%2nd COMMON larger /XXXXXX/

При загрузке модулей, которые включают COMMON блоки, LINK-80, использует размер первого загруженного COMMON блока, чтобы установить необходимый объем памяти, прежде, чем будет загружен код программы. Если последующий модуль содержит COMMON блок, большего размера, чем первый загруженный, LINK-80 выведет это сообщение об ошибке. Это означает, что первое определение /XXXXXX/ блока COMMON встреченное в загруженных модулях не было самым большим блоком, названным этим именем. Переупорядочьте последовательность загрузки модулей, или измените COMMON блоки так, чтобы все блоки были одного размера.

%Mult. Def. Global YYYYYY

У Вас есть одно глобальное (PUBLIC) имя YYYYYY более чем с одним определением. Обычно, два или более из загруженных модулей объявили одинаковое имя как PUBLIC.

%Overlaying Program Area ,Start = xxxx

,Public = <symbol name> (xxxx)

,External = <symbol name> (xxxx)

Обычно это происходит, когда установлены ключи /D или /P в адрес в области, использованной LINK-80. Вы должны сбросить адрес ключа выше 102H. Это может также произойти, если вы установите адреса для загрузки программ после того как первоначально были загружены некоторые программы и адреса не были достаточно большими. Например, если MYPROG больше, чем 147 байтов, и Вы вводите команду:

MYPROG,/P:15 0,SUBR1,FUNNY/N/E

вы получите сообщение об ошибке %Overlaying Program Area.

%Overlaying Data Area ,Start = xxxx

,Public = <symbol name> (xxxx)

,External = <symbol name> (xxxx)

Ключи /D и /P были установлены слишком близко. Например, если /D дали более высокий адрес, чем /P, но не достаточно высоко чтобы быть вне области программы, когда программа будет загружена, ее верхний край будет расположен в области данных. Или, если /D ниже, чем /P, /P не был достаточно высок, чтобы препятствовать размещению начала программы в области, уже загруженной данными.

?Intersecting Program Area

или

?Intersecting Data Area

Области программ и данных пересекаются и адреса или цепочки внешних входов находится в этом пересечении. Окончательное значение не может быть преобразовано в текущее значение, поскольку оно находится в области пересечения.

Origin Above Loader Memory/ Move Anyway (Y or N)?

или

Origin Below Loader Memory, Move Anyway (Y or N)?

Это сообщение появится только после того, как LINK-80 была дана команда ключа /E или /G. Если LINK-80 не хватает памяти, чтобы загрузить модуль, но /E или /G не были введены, вы получите сообщение “?Out of Memory”.

LINK-80 может загружать модули только между первым адресом в памяти, и верхнюю часть доступной памяти. Если программа является слишком большой для этого пространства, или если Вы установите ключи /D и/или /P слишком высоко для размера вашей программы, LINK-80 не хватит памяти и она выведет приведенное выше сообщение “Origin Above Loader Memory”.

Если вы установили ключи /D и/или /P ниже первого адреса LINK-80 (100H для CP/M), LINK-80 выведет сообщение “Origin Below Loader Memory”. Это предотвратит загрузку вашей программы в память, предназначенную для операционной системы.

Если будет дан ответ Y<CR>, то LINK-80 будет перемещать область и продолжит работу. Если будет введено что-либо другое, то LINK-80 закончит работу. В любом случае, если был задан ключ /N, образ будет уже сохранен.

# Глава 7 CREF-80 Программа построения отчета о перекрестных ссылках

Программа построения отчета о перекрестных ссылках обрабатывает специально собранный файл листинга, чтобы перечислить все межмодульные ссылки и их определения. Результат - список перекрестных ссылок. Этот список перекрестных ссылок может быть использован для оказания помощи в отладке программы.

Программа построения отчета о перекрестных ссылках CREF-80 позволяет программисту обрабатывать файл перекрестных ссылок, сгенерированный MACRO-80. Этот файл перекрестных ссылок содержит встроенные управляющие символы, установленные во время ассемблирования MACRO-80. CREF-80 интерпретирует управляющие символы и создает файл, который содержит листинг перекрестных ссылок между переменными.

CREF-80 генерирует листинг, напоминающий PRN листинг MACRO-80, с двумя дополнительными возможностями:

1. Каждый исходный оператор пронумерован числом перекрестной ссылки.
2. В конце листинга имена переменных появляются в алфавитном порядке. Каждое имя сопровождается номером строки, в которой определена переменная (отмечена с помощью символа #) сопровождаемая номерами других строк, в которых на эту переменную ссылаются.

Файла листинга CREF заменяет файл листинга PRN MACRO-80 и получает расширение файла .LST вместо .PRN.

## 7.1 Создание листинга перекрестных ссылок

Создание листинга CREF включает два шага: (1) создание файла перекрестных ссылок (.CRF), и (2) генерация списка перекрестных ссылок (.LST). Первый шаг происходит в макроассемблере MACRO-80, второй в программе построения отчета о перекрестных ссылках CREF-80.

**Создание файла перекрестных ссылок**

Чтобы создать файл перекрестных ссылок, установите переключатель /C в командной строке MACRO-80. Например:

**M80 =NEIL/C**

Эта командная строка компилирует файл NEIL.MAC, генерируя выходные файлы NEIL.REL (объектный файл) и NEIL.CRF (файл перекрестных ссылок).

**Создание листинга перекрестных ссылок**

Листинг перекрестных ссылок создается путем обработки файла .CRF с помощью программы CREF-80. Чтобы вызвать программу построения отчета о перекрестных ссылках, введите:

**CREF80**

CREF-80 вернет приглашение звездочку (\*). Для создания файла листинга перекрестных ссылок, введите:

**=filename**

где filename - имя вашего .CRF файла. Например:

**CREF80 =NEIL**

будет генерировать .LST файл (NEIL.LST), содержащий информацию о перекрестных ссылках.

Этот .LST файл можно распечатать или отправить на экран терминала с помощью команд операционной системы. Кроме того, CREF-80 поддерживает те же обозначения устройств вывода, что и MACRO-80. Просто введите обозначение устройства перед именем файла. Например:

**CREF80 LST:=NEIL**

посылает листинг .LST только на принтер (файл на диске не создается).

**CREF80 TTY:=NEIL**

посылает листинг .LST только на CRT (файл на диске не создается).

Вам нужно указать диск, если вы хотите, чтобы файл .LST, был сохранен в месте отличном, от текущего диска (на котором находится файл .CRF). Например:

**CREF80 B:=A:NEIL**

Сохранит NEIL.LST на диске B.

При завершении, CREF-80 выведет подсказку со звездочкой. Вы можете ввести другое =filename, или выйти из CREF-80 в операционную систему.

Для выхода из CREF-80, введите:

**CTRL-C**

Если Вы хотите дать файлу .LST имя отличное от имени по умолчанию (имя.CRF и расширение .LST), введите имя перед знаком равенства. Например:

**CREF80 NEIL.CRF=NEIL** или

**CREF80 NEILCREF=NEIL**

Первая командная строка сгенерирует файл листинга перекрестных ссылок под названием NEIL.CRL, последняя сгенерирует файл под названием NEILCREF.LST.

Посмотрите на расширение файла, чтобы отличить, файл листинга перекрестных ссылок от файла листинга который обычно генерирует MACRO-80. По умолчанию MACRO-80 обычно генерирует файл листинга (без установленного в командной строке ключа /C), с расширением .PRN. Файл списка перекрестных ссылок, сгенерированный CREF-80, по умолчанию имеет расширение .LST.

## 7.2 Управление листингом CREF с помощью псевдо-операторов

Вы можете задать опцию создания листинга перекрестных ссылок для части программы. Чтобы управлять листингом или подавлением перекрестных ссылок, используйте псевдо-команды управления листингом перекрестных ссылок .CREF и .XCREF, в исходном файле MACRO-80. Эти две псевдо-команды могут быть введены в любом месте в программе в поле оператор. Как любая псевдо-команда управления листингом .CREF и .XCREF не имеют параметров.

| Псевдо-команда | Определение |
| --- | --- |
| **.CREF** | Создание перекрестных ссылок.  .CREF условие по умолчанию. Используйте .CREF, чтобы разрешить создание файла перекрестных ссылок после использования псевдо-команды .XCREF. .CREF остается в силе до тех пор пока MACRO-80 не встретит .XCREF. Заметим, однако, что .CREF не оказывает никакого эффекта, если не установлен ключ /C в командной строке MACRO-80. |
| **.XCREF** | Подавление перекрестных ссылок.  .XCREF отменяет действие псевдо-команды .CREF (значение по умолчанию). .XCREF, действует, пока MACRO-80 не встретится с .CREF. Используйте .XCREF, чтобы отменить создание перекрестных ссылок в выбранных частях файла. Поскольку ни .CREF, ни .XCREF не действуют, пока не установлен ключ /C в командной строке MACRO-80, нет никакой необходимости использовать .XCREF, если нужен обычный файл листинга (без перекрестных ссылок), просто опустите ключ /C в командной строке MACRO-80. |

# Глава 8 LIB-80 Обслуживание библиотек

**Предупреждение**

Прочитайте эту главу внимательно и сделайте резервную копию библиотеки перед использованием LIB-80. Программа LIB-80 очень мощная и, таким образом, может быть очень разрушительной. С помощью LIB-80 очень просто уничтожить библиотеку.

Программа LIB-80 разработана для обслуживания библиотек Microsoft FORTRAN-80 и COBOL-80 в CP/M. LIB-80 также может быть использована для создания собственной библиотеки подпрограмм на языке ассемблера.

LIB-80 создает библиотеки времени выполнения из программ на языке ассемблера, которые являются подпрограммами COBOL, FORTRAN и других программ на языке ассемблера. Программы, собранные LIB-80, могут быть специальными модулями, создаваемыми программистом или модулями из существующей библиотеки (например, FORLIB). С помощью LIB-80, вы можете создать специализированные библиотеки времени выполнения для любых программ, которые Вы разрабатываете.

Ценностью создания библиотеки является то, что все подпрограммы, необходимые для выполняемой программы могут быть связаны с ней в исполняемый файл (COM), вводом имени библиотеки, затем ключа /S в командной строке LINK-80. Например:

**L80 MAIN,NEWLIB/S,NEIL/N/G**

Это намного более удобно, чем раздельный ввод необходимых подпрограмм, особенно если существует много модулей. С файлом библиотеки Вы можете быть уверены, что все необходимые модули будут скомпонованы в файл COM, плюс отсутствует опасность исчерпать пространство командной строки LINK-80. Кроме того, библиотека делает этот набор специальных подпрограмм доступным для простого включения в любую программу.

## 8.1 Демонстрация сеанса LIB-80

Существует два наиболее распространенных применения LIB-80 - создание библиотеки и распечатка содержимого библиотеки. Следующие примеры иллюстрируют основные команды для этих двух целей.

**Построение библиотеки:**

A>**LIB**

\***TRANLIB=SIN,COS,TAN,ATAN,ACOG**

\***EXP**

\***/E**

A>

В этом демонстрационном сеансе LIB вызывает программу LIB-80, которая возвращает подсказку звездочка (\*). TRANLIB - имя создаваемой библиотеки. SIN, COS, TAN, ATAN, ACOG - имена файлов, которые будут объединены в TRANLIB. EXP - имя файла, который будет добавлен в TRANLIB. (Файл EXP мог быть перечислен в предыдущей командной строке, этот пример показывает что файлы могут добавляться по одному или сразу несколько.) Ключ /E заставляет LIB-80 переименовать TRANLIB.LIB в TRANLIB.REL и затем вернуться в CP/M.

**Распечатка содержимого библиотеки:**

A>**LIB**

\***TRANLIB.LIB/U**

\***TRANLIB.LIB/L**

.

.

(Список символов в TRANLIB.LIB)

.

.

\***CTRL-C**

A>

В этом демонстрационном сеансе LIB вызывает LIB-80. TRANLIB.LIB/U говорит LIB-80 искать в TRANLIB.LIB любые межмодульные ссылки которые не будут определены за один проход через библиотеку (т.е., любые "отсутствующие" символьные ссылки).

TRANLIB.LIB/L просит LIB-80 перечислить модули в TRANLIB.LIB и определения символов, которые содержат модули. Ctrl-C возвращает в CP/M не изменяя ни каких файлов.

**Предупреждение**

Ключ /E уничтожит Вашу текущую библиотеку, если не создается новая библиотека. Это представляет особую опасность для библиотеки FORTRAN времени выполнения FORLIB.REL. **Если Вы только распечатываете содержимое библиотеки и не пересматриваете ее, для выхода из LIB-80 используйте Ctrl-C**.

## 8.2 Команды LIB-80

**Вызов LIB-80**

Чтобы вызвать LIB-80, введите:

**LIB**

LIB-80 возвратит подсказку звездочка (\*), указывая о готовности принимать команды. Каждая команда в LIB-80 добавляет модули в библиотеку в процессе создания.

Команды LIB-80 состоят из необязательного поля Destination (файл-приемник), поля Source (файл-источник) и необязательного поля Switch (ключ).

Формат команды LIB-80:

**Destination=Source/Switch**

Каждое поле описано ниже. Общий формат для каждого поля показан в круглых скобках после имени поля.

**Поле Destination (**filename=**)**

Это дополнительное поле. Знак равенства требуется, если какая-либо запись сделана в этом поле.

Введите в это поле имя файла (с расширением если необходимо) для файла библиотеки, которую Вы хотите создать.

Если это поле опущено, LIB-80 по умолчанию в качестве имени файла использует FORLIB. По умолчанию расширение файла - REL.

**Предупреждение**

**Не путайте это имя файла по умолчанию FORLIB.LIB с библиотекой времени выполнения FORLIB.REL, поставляемой с FORTRAN-80**. Эти две библиотеки не будут тем же, если Вы дадите команду LIB-80 скопировать все файлы из библиотеки времени выполнения FORTRAN в новую библиотеку. Кроме того, когда Вы выходите из LIB-80, по умолчанию, библиотеке будет присвоено расширение файла .REL, что означает, что она заменит FORLIB.REL предоставленную с FORTRAN-80. Поэтому, если Вы не хотите уничтожить свою библиотеку времени выполнения FORTRAN-80, мы решительно рекомендуем, чтобы Вы всегда задавали имя файла назначения при создании новой библиотеки.

**Поле Source (filename<module>)**

В этом поле требуется некоторая запись. Все исходные файлы должны быть файлами REL.

Содержимое поля Source говорит LIB-80, какие файлы или части файлов (модули), необходимо добавить в файл библиотеки назначения. У вас есть два варианта записи:

1. Только имя файла(ов)

2. Любая комбинация имени файла(ов) и имени модуля(ей).

Используются следующие правила синтаксиса:

1. Если команда состоит только из имен файлов, записи разделяются только запятыми. Например:

**FILE1,FILE2,FILE3**

1. Если команда состоит из имен файлов и имен модулей, имена модулей должны быть включены в угловые скобки (<>). Модули следуют за именем файла, в котором они находятся. Каждая комбинация имя файла <имя модуля> отделяется от других записей командной строки запятыми. Например:

**FILE1,FILE2<MODZ>,FILE3<MODR>,FILE4**

1. Если указывается больше чем один модуль из одного файла, имена модулей, включенные в угловые скобки (<>), должны быть разделены друг от друга запятыми.

Например:

**FILE1,FILE2<M0DZ,MODR>,FILE3**

См. дополнительные сведения об исходных модулях, вариант 2, ниже.

Файлы и модули, как правило, являются подпрограммами FORTRAN или COBOL, или основными программами, или ALDS программами на языке ассемблера, которые содержат операторы ENTRY, GLOBAL или PUBLIC. (Эти операторы называются точками входа.) LIB-80 распознает модуль по имени программы, которое может быть именем файла, или именем заданным в псевдо-командах .TITLE или NAME в MACRO-80. Все исходные файлы должны быть файлами REL.

LIB-80 объединяет файлы и модули REL в файлы REL, то есть LIB-80 помещает один файл или модуль за другим.

Таким образом, нет никакого различия между командой по правилу 2 синтаксиса выше и

**FILE1**

**FILE2<MODZ>**

**FILE3<MODR>**

**FILE4**

Кроме того, поскольку файл библиотеки строится путем добавления, важно, разместить модули так, чтобы все межмодульные ссылки были "впереди". То есть модуль, содержащий внешнюю ссылку должен физически появляются ранее модуля, содержащего точку входа (определение). В противном случае, когда Вы направляете LINK-80 к поиску в библиотеке, LINK-80 не сможет удовлетворить все ссылки за один проход библиотеки.

**Дополнительные сведения об исходных модулях**

Для извлечения модулей из предыдущих библиотек и других файлов REL, LIB-80 использует мощный синтаксис, чтобы указывать диапазоны модулей в файле REL.

Эти диапазоны могут быть от одного модуля до всего файла (в этом случае параметры модуля не указываются).

Основной принцип определения диапазона модулей, обычно, что любой модуль, заданный в команде, будет включен в библиотеку. (Исключение составляют случаи, при указании относительного смещения диапазона, пункт 6 ниже.)

Параметры для указания модулей:

1. Только один модуль

Введите имя модуля. Например:

**FILE1<MODZ>**

включает только модуль MODZ из FILE1.

1. Несколько несмежных модулей из одного файла

Введите имена модулей, разделенных запятыми. Например:

**FILE1<MODZ,MODR,MODK>**

включает в себя модули MODZ, MODR и MODK. Обратите внимание на то, что эти модули могут быть заданы для объединения в любом порядке, который необходим для надлежащего поиска в один проход, независимо от их порядка в исходном файле.

1. От первого модуля по названный модуль

Введите две точки (..) и имя последнего модуля. Например:

**FILE1<..MODK>**

включает в себя все модули, от первого модуля в файле FILE1 по модуль MODK.

1. От названного модуля до последнего модуля

Введите имя модуля, с которого начинается диапазон, за которым следуют две точки (..). Например:

**FILE1<MODR..>**

включает в себя все модули, начиная с модуля MODR, до последнего модуля в FILE1.

1. От одного заданного имени модуля до другого заданного имени модуля

Введите имя модуля, который начинает диапазон вместе с двумя точками (..), а затем имя модуля, который заканчивает диапазон.

Например:

**FILE1<MODZ..MODK>**

включает в себя все модули, начиная с модуля MODZ, до модуля MODK.

1. Относительное смещение диапазона

Введите имя модуля, сопровождаемое + или - и число модулей, которые будут включены. + означает после заданного модуля. - предшествующие заданному модулю. Заданный модуль не включается в библиотеку. Число смещения должно быть целым числом в диапазоне 1 - 255. Например:

**FILE1<MODZ+2>**

включает в себя два модуля сразу после MODZ. В то время как

**FILE1<MODK-3>**

включает эти три модуля, непосредственно предшествующие модулю MODK.

Кроме того, диапазоны и смещения могут быть использованы вместе. Например:

**FILE1<MODR+1..MODK-1>**

включает все модули между модулем MODR и модулем MODK (но ни MODR, ни MODK не включены).

1. Все модули в файле

Введите только имя файла. Например:

**FILE1**

включает в себя весь файл (все модули в файл FILE1).

**Поле Switch (/ключ)**

Запись команд в поле ключа LIB-80 служит для выполнения дополнительных функций. Поле ключа содержит букву которой предшествует косая черта (/).

**Предупреждение**

Ключ /E уничтожит вашу текущую библиотеку, если не создается новая библиотека, в стадии строительства. Это - особая опасность для Вашей FORTRAN библиотеки времени выполнения FORLIB.REL, потому что FORLIB - имя файла по умолчанию, используемое, если вы не определяете имя файла назначения. Поэтому, если вы не хотите удалить свою полную библиотеку времени выполнения FORTRAN, задайте LIB-80 имя файла назначения для новой библиотеки. Если вы только распечатываете содержимое библиотеки и не пересматриваете ее, выходите из LIB-80 используя Ctrl-C.

| **Ключ** | **Действие** |
| --- | --- |
| /E | Выход в CP/M. Если вы не создаете новую библиотеку или пересматриваете существующую библиотеку, используйте Ctrl-C вместо /E.  Библиотека, в стадии строительства (.LIB), переименовывается в .REL, и любая предыдущая копия файла библиотеки удаляется. Вот почему ключ /E так опасен и не должен использоваться, если вы не создаете новую библиотеку. Мы снова решительно рекомендуем, чтобы вы всегда задавали имя файла в поле Destination в командной строке LIB-80. |
| /R | Создаваемая библиотека, изначально имеющая расширение (.LIB), переименовывается и получает расширение .REL. К ключу /R применимы те же предупреждения и предостережения, которые применяются к ключу /E.  Предыдущая библиотека с таким же именем стирается. Используйте /R, только если вы создаете новую библиотеку. Ключ /R выполняет те же функции что и /E, но не выходит в CP/M при завершении. Используйте /R вместо /E, если нужно выйти из текущей библиотеки, но продолжить использовать программу LIB-80 для корректировки другой библиотеки. |
| /L | Вывести список всех заданных модулей и содержащихся в них определений символических имен. Содержание файла выводится в формате перекрестных ссылок.  Списки в настоящее время всегда отправляются на терминал. Используйте Ctrl-P перед запуском LIB-80, чтобы отправить список на принтер. |
| /U | Используйте ключ /U для вывода списка символов, которые могут быть неопределенными за один проход чтения библиотеки. Если символ в библиотеке модуля расположен "позже" (к предыдущему модулю), то /U распечатает этот символ. |
| /C | Используйте ключ /C, чтобы очистить команды LIB-80, не выходя из программы LIB-80. Библиотека, в стадии строительства удаляется и сеанс LIB-80, запускается. Снова появится подсказка звездочка (\*).  Используйте /C, если вы определили неправильный модуль (и)или неправильный порядок, и хотите выполнять новые команды LIB-80. |
| /O | Используйте /O, чтобы установить режим вывода в восьмеричной системе счисления. /O вводится вместе с ключом /L команды LIB-80 вывода списка. Помните: когда ключи задаются рядом, косая черта должна предшествовать каждому ключу. Например:  NEWLIB/L/O |
| /H | Используйте /H, чтобы установить режим вывода в шестнадцатеричной системе счисления. Шестнадцатеричная система счисления используется по умолчанию. |

# Приложение A Совместимость с другими ассемблерами

Средства управления $EJECT и $TITLE предусмотрены для совместимости с ассемблером ISIS Intel. Знак доллара должен появиться в столбце 1, только если пробелы или вкладки разделяют знак доллара от управляющего слова. Управляющее слово $EJECT совпадает с псевдокомандой PAGE в MACRO-80. Управляющее слово

$TITLE('text')

совпадает с псевдокомандой SUBTTL <text> в MACRO-80.

Операнды Intel PAGE и INPAGE генерируют ошибку Q, когда используется с псевдокомандами MACRO-80 CSEG или DSEG. Эти ошибки – предупреждения, ассемблер игнорирует операнды.

По умолчанию в MACRO-80 используется счетчик команд в режиме перемещаемого кода со значением 0. В ассемблере Intel ISIS по умолчанию действует абсолютный режим со значением 0.

В MACRO-80, знак доллара ($) является определенной константой, которая указывает значение адреса счетчика команд в начале оператора. Другие ассемблеры могут использовать десятичную точку или звездочку. В MACRO-80 определены некоторые константы, имеющие следующие значения:

A=7 B=0 C=l D=2 E=3

H=4 L=5 M=6 SP=6 PSW=6

# Приложение B Пакет утилит в TEKDOS

В операционной системе TEKDOS форматы команд MACRO-80, LINK-80, и CREF-80 отличаются незначительно.

## B.1 Командные файлы TEKDOS

Файлы M80, L80 и C80 являются командными файлами TEKDOS для ассемблера, компоновщика, и программы построения перекрестных ссылок соответственно. Эти командные файлы устанавливают режим эмуляции 0 и выбирают ассемблерный процессор Z-80 (см. документацию TEKDOS), затем выполняют соответствующий файл программы. Обратите внимание, что все эти командные файлы настроены на выполнение программ Microsoft с диска #1. Программа LINK-80 будет также искать библиотеку на диске #1. Если вы хотите выполнить любую из этих программ с диска #0, командный файл должен быть отредактирован. Затем программе LINK-80, должна быть явно задана директива поиска библиотек, таких как MYLIB-S. См. раздел **Ключи** в Главе 6, LINK-80 Связывающий загрузчик.

Имена файлов в TEKDOS не используют назначенные по умолчанию расширения файлов Пакета утилит.

## B.2 MACRO-80

Ассемблер MACRO-80 понимает только командные строки (команда вызова, M80, все имена файлов и ключи должны быть в одной строке). Приглашение не выводится на экран и интерактивные команды (,TTY:=TTY: и ,LPT:=TTY:) не доступны. Команды имеют формат аналогичный команде ассемблера TEKDOS, то есть, содержат до трех имен файлов или имен устройств, а также необязательные ключи.

**M80 [object] [list] source [switch [switch [...]]]**

Поля для имен файлов object и list являются необязательными. Эти файлы не будут созданы, если параметры опущены. Сообщения об ошибках по-прежнему будут выводиться на консоль. Доступные ключи подробно описаны в Главе 5 настоящего руководства. Все поля командной строки могут быть разделены запятыми или пробелами. Если вы не хотите создавать объектный файл, необходимо ввести <пробел запятая пробел > между M80 и полем имени файла list. Например:

**M80 , LIST SOURCE**

## B.3 CREF-80

Форма команд CREF-80:

**C80 list source**

Требуются оба имени файлов. Имя исходного файла всегда имя CREF-80 файла создаваемого во время ассемблирования с ключом C.

Пример:

Чтобы создать файл CREF-80 из источника TSTMAC используя MACRO-80, введите:

**M80 , TSTCRF TSTMAC C**

Чтобы создать список перекрестных ссылок из файла CREF-80 TSTCRF, введите:

**C80 TSTLST TSTCRF**

## B.4 LINK-80

В TEKDOS компоновщик LINK-80 принимает только интерактивные команды. Командные строки не поддерживаются.

Когда программа LINK-80 вызывается, и ожидает ввода, она выводит приглашение звездочку. Команды представляют собой списки из имен файлов и/или устройств, разделенных запятыми или пробелами и, при необходимости, вперемешку с ключами. Входные файлы LINK-80 должны иметь объектный перемещаемый код Microsoft (не такой, как формат загрузчика TEKDOS).

Ключи в TEKDOS для LINK-80 разделяются дефисом, вместо косой черты. Поддерживаются все ключи LINK-80 (описанные в Главе 6), за исключением -G и -N, которые не реализованы в настоящее время.

Пример:

1. Ассемблирование с помощью MACRO-80 программы с именем XTEST, создание объектного файла под названием XREL и файла листинга с именем XLST:

>**M80 XREL XLST XTEST**

Загрузка XTEST и сохранение загруженного модуля:

>**L80**

\***XREL-E**

[04AD 22B8]

\*DOS\*ERROR 46

L80 **TERMINATED**

>**M XMOD 400 22B8 04AD**

Обратите внимание, что ключ -E вызывает выход с сообщением об ошибке в связи с исполнением инструкцией Halt. Образ памяти не поврежден, однако, может использоваться команда Module TEKDOS для его сохранения. После того, как программа будет сохранена в формате модуля, она затем может быть выполнена непосредственно без повторного прохождения через LINK-80.

Цифры в скобках напечатанные LINK-80 перед выходом являются начальным и конечным адресами загрузки соответственно. Загрузчик по умолчанию начинает загрузку с 400H. Однако, загрузчик также помещает команду перехода к начальному адресу программы в ячейке памяти 0, что позволяет выполнять программу с адреса 0. Память между адресами 0003 и 0400h зарезервирована для SRB и буферов ввода/вывода во время выполнения.

# Приложение D Формат LINK совместимых объектных файлов

Это приложение содержит справочное пособие для пользователей, которые хотят знать формат загрузки перемещаемых объектных файлов LINK-80. Ни один пункт из этого пособия не требуется для работы ALDS. В материале, представленном здесь, нет ничего чем может управлять пользователь. Материал сугубо технический, и не содержит учебных материалов.

Совместимые с LINK объектные файлы состоят из потока битов. Отдельные поля в битовом потоке не выровнены на границах байтов, за исключением указанных ниже. Использование битового потока для перемещаемых объектных файлов помогает свести размер объектных файлов к минимуму, что уменьшает количество операций чтения/записи.

Существуют два основных типа элементов загрузки: абсолютный и перемещаемый. Первый бит элемента указывает один из этих двух типов. Если первый бит - 0, следующие 8 битов загружаются в качестве абсолютного байта (в соответствии со значением места расположения счетчика). Если первый бит - 1, следующие 2 бита используются, чтобы указать один из четырех типов перемещаемых элементов:

00 Специальный элемент LINK (см. ниже).

01 Перемещаемая программа. Загрузить следующие 16 бит по адресу равному значению текущего счетчика программы плюс смещение.

10 Перемещаемые данные. Загрузить следующие 16 бит по адресу равному значению текущего счетчика данных плюс смещение.

11 Перемещаемый COMMON. Загрузить следующие 16 бит по адресу равному значению текущего счетчика данных Common плюс смещение.

Специальные элементы LINK состоят из последовательности битов 100 (один-ноль-ноль), а затем:

четырех битов поля управления

дополнительное поле A, состоит из двух битов определяющих типа адреса, которое совпадает с двух битовым полем, описанным выше, кроме 00, которое определяет абсолютный адрес, и шестнадцати битного поля адреса.

необязательное поле B, состоит из 3 битов, которые задают длину имени символа и поля имени содержащее по 8 бит для каждой буквы имени символа.

Общее представление специального элемента LINK:



где:

xxxx - поле управления четыре бита (0-15 ниже)

yy - поле типа адреса (два бита)

nn - поле со значением адреса (шестнадцать бит)

zzz - поле длины имени символа (три бита)

Следующие специальные типы используются только для поля B:

0 Вхождение символа (имя для поиска).

*Символ, обозначенный в поле имени, определен в этом модуле (объявлен PUBLIC).*

1 Выбор COMMON блока

*Инструктирует LINK-80 использовать счетчик адреса, связанный с COMMON блоком, обозначенным в поле имени для последующих перемещаемых общих элементов.*

2 Название программы (*Имя перемещаемого модуля*)

3 Запрос поиска в библиотеке

4 Элемент расширения LINK (см. ниже)

Следующие специальные элементы LINK используются в полях A и B:

5 Определяет размер COMMON

*Поле значения определяет объем памяти, зарезервированный для common блока, описанного в поле имени. Первый размер, выделенный данному блоку, должен быть больше или равным, чем любым последующим определениям для того же блока в других соединяемых модулях.*

6 Внешняя цепочка (A – адрес заголовка цепочки , B - имя внешнего символа).

*Поле значения содержит адрес заголовка цепочки, которая заканчивается абсолютным 0. Каждый элемент цепочки заменяется значением внешнего символа, описанного в поле имени. [используемый EXTRN и т.д.]*

7 Определяет точку входа (A - адрес, B - имя)

*Значение символа в поле имени определяется значением поля. [PUBLIC* *определение]*

Следующие специальные элементы LINK используются только для поля А:

8 Внешний - смещение. Используется для внешних JMP и CALL

9 Внешний + смещение. Значение A будет добавлено к двум стартовым байтам, в текущем счетчике адреса сразу перед выполнением.

10 Определяет размер области данных (A это размер)

*Значение поля содержит число байт в сегменте данных текущего модуля.*

11 Устанавливает счетчик адреса в А.

*Устанавливает счетчик адреса в значение, определенное значением поля.*

12 Адрес цепочки. А - голова цепочки. Заменить все записи в цепочке текущим местоположением счетчика. Последняя запись цепочки в поле адреса содержит абсолютный ноль.

*Значение поля содержит голову цепочки, которая заканчивается абсолютным 0. Каждый элемент цепочки заменяется текущим значением положения счетчика.*

13 Определяет размер программы (A это размер).

*Значение в поле содержит число байтов в сегменте программы текущего модуля.*

14 Конец программы (выравнивает к границе байта)

*Определяет конец текущего модуля. Если значение поля содержит значение отличное от абсолютного 0, оно используется в качестве начального адреса для соединяемого модуля. Т.е. текущий модуль - основной модуль. Следующий элемент в файле начинается на следующей границе байта.*

Следующие специальный элемент LINK не имеет ни поля А, ни поля B:

15 Конец файла

*Этот элемент следует за последним элементом модуля в конце файла.*

Элемент расширения LINK использует только общий формат поля B специального элемента LINK, но содержание поля B не является именем символа. Вместо этого область символа содержит одну букву, чтобы идентифицировать тип элемента расширения LINK, сопровождаемого от 1 до 7 символов дополнительной информации.

Таким образом каждый элемент расширения LINK имеет формат:

1 00 0100 lll s bbbbbb

где:

lll 3 бита содержащие длину поля bbbbbb (0 означает 1, так как F80 порождает 0 для неименованного COMMON),

s - это восьми битный элемент расширения LINK суб-идентификатор типа, и

bbbbbb содержит от 1 до 6 байт дополнительной информации. Если поле B используется для имени, bbbbbb может состоять только из 6 символов.

Существующие подтипы элемента расширения LINK:

5 X'35' сигнальная метка оверлейного сегмента COBOL

A X'41' арифметическая адресная привязка (арифметический оператор)

B X '42' арифметическая адресная привязка (внешняя ссылка)

C X '43' арифметическая адресная привязка (Базовый адрес + смещение)

**Описания подтипов**

**Подтип 5**

Когда оверлей LINK встречает сегмент оверлея, lll получает значение 010 (бинарное), и текущему номеру сегмента оверлея присваивается значение b+49.

Если ранее существующий номер сегмента не равен нулю и действует ключ /N, область данных записывается в файл на диск, с текущим именем программы и расширением Vnn, где nn это две шестнадцатеричных цифры представляющие число b+49 (десятичное).

**Подтипы A, B, C**

Подтипы A, B и C позволяют обрабатывать арифметический польский текст. Элементы должны обрабатываться как выражения в обратной польской записи. Одно или более значений (подтипы B или C), с последующими одним или более арифметическими операторами (подтип A), и в конце хранит результат арифметического оператора (B.STBT или B.STWD).

Все элементы помещаются в таблицу адресных привязок (Fixup) после того, как любые записи смещения были преобразованы в конечные адреса. Польское выражение выполняется из Таблицы Fixup в конце компоновки. Результат сохраняется в счетчике команд (PC), когда элементы прочитаны.

# Приложение E Список псевдокоманд MACRO-80

Обозначения: \* означает псевдокоманды Z80

отсутствие звездочки означает псевдокоманды 8080

**Однофункциональные псевдокоманды**

Выбор набора инструкций

.Z80

.8080

Определение данных и символов

<name> ASET <exp>

BYTE EXT <symbol>

BYTE EXTRN <symbol>

BYTE EXTERNAL <symbol>

DB <exp>[,<exp>...]

DB <string>[<string>...]

DC <string>

DDB <exp>[,<exp>...]

\* DEFB <exp>[,<exp>...]

\* <name> DEFL <exp>

\* DEFM <string>[,<string>...]

\* DEFS <exp>[,<val>]

\* DEFW <exp>[,<exp>...]

DS <exp>[,<val>]

DW <exp>t,<exp>...]

ENTRY <name>[,<name>...]

<name> EQU <exp>

EXT <name>[,<name>...]

EXTRN <name>[,<name>...]

\* EXTERNAL <name>[,<name>...]

GLOBAL <name>[,<name>.-.]

PUBLIC <name>[,<name>...]

<name> SET <exp> (not in . Z80 mode)

Режим счетчика команд

ASEG

CSEG

DSEG

COMMON /<block name>/

ORG <exp>

.PHASE <exp>/

.DEPHASE

Псевдокоманды имеющие отношение к файлу

.COMMENT <delim><text><delim>

END [<exp>]

INCLUDE <filename>

$INCLUDE <filename>

MACLIB <filename>

.RADIX <exp>

.REQUEST <filename>[,<filename>...]

**Псевдокоманды листинга**

Псевдокоманды управления форматированием

\* \*EJECT [<exp>] (одна звездочка является частью \*EJECT)

PAGE <exp>

SUBTTL <text>

TITLE <text>

$TITLE

Псевдокоманды общего управления листингом

.LIST

.XLIST

.PRINTX <delira><text><delim>

Псевдокоманды условного управления листингом

.SFCOND

.LFCOND

.TFCOND

Псевдокоманды управления листингом макро расширений

.LALL

.SALL

.XALL

Псевдокоманды управления листингом перекрестных ссылок

.XCREF

.CREF

**Псевдокоманды макросредств**

Псевдокоманды определения макросов

<name> MACRO <parameter>[,<parameter>...]

ENDM

EXITM

Псевдокоманды повторения

REPT <exp>

IRP <dummy>,<параметры в угловых скобках>

IRPC <dummy>,string

Псевдокоманды условного ассемблирования

\* COND <exp>

ELSE

\* ENDC

END IF

IF <exp>

IFB <arg>

IFDEF <symbol>

IFDIF <argl>,<arg2>

IFE <exp>

IFF <exp>

IFIDN <argl>,<arg2>

IFNB <arg>

IFNDEF <symbol>

IFT <exp>

IF1

IF2