**Supersoft / Small Systems Services**

**Расширенный Fortran IV**

**Руководство пользователя**

**(Версия 3.0)**

Авторское право 1977, 1978, 1981 Small Systems Services, Incorporated, Урбана, Иллинойс. Все права защищены. Никакая часть этого материала не может быть перепечатана, скопирована или продана без явного письменного согласия Small Systems Services, Incorporated.

**Оплата за лицензию на время выполнения**

Сборы за распространение пакетов времени выполнения Fortran следующие:

Первые 100 экземпляров в год: без взимания платы.

101 копия или более в год:

Опция 1: Стоимость копии составляет 2% от продажной цены вашего продукта или 5,00 $ США, в зависимости от того, что больше.

Если пакет поставляется с оборудованием и не может быть определена отдельная стоимость программного обеспечения, плата составляет 5 $ США за копию (более 100 экземпляров).

Опция 2: Ежегодный лицензионный сбор в размере 2000.00 $

Опция 3: Бессрочная лицензия 7500,00 $

Для получения дополнительной информации об этих лицензионных сборах времени выполнения, или получения соглашения для распространения пакета времени выполнения FORTRAN, свяжитесь с SuperSoft.

**Содержание**

1. Введение 4

2. Язык SSS Fortran 4

2.1 Введение 4

2.2 Сравнение с WATFIV 5

2.2.1 Типы данных 5

2.2.2 Ограничения SSS Fortran 6

2.2.3. Отличие SSS Fortran от WATFIV 6

2.2.4. Расширения SSS Fortran 8

2.3. Краткий обзор типов операторов 11

2.3.1. Выподняемые операторы 11

2.3.2. Неисполняемве операторы 15

2.4. Примечания 19

3. Использование компилятора 21

3.1. Общее использование 21

3.2. Опции переключателей компилятора 22

3.3. Примечания 23

4. Загрузчик 25

4.1. Общее использование 25

4.2. Формат командной строки 25

4.3. Формат вывода 25

4.4. Библиотеки 26

4.5. Ошибки загрузчика 26

4.6. Опции переключателей 27

4.7. Примечания 27

5. Пакеты ввода-вывода 27

5.1. Общее использование 27

5.2. Вызываемые подпрограммы 27

5.3. Интерфейс пользователя 29

5.4. Примечания 29

5.5. Примеры программ 29

6. Библиотеки 30

6.1. Общее использование 30

6.2. Библиотека внутренних и встроенных функций 31

6.3. Библиотека служебных подпрограмм 33

6.4. Математическая библиотека SSS Fortran 33

6.5. Библиотеки работы со строками и динамическим распределением 36

6.5.1. Арифметические действия 37

6.5.2. Динамическое распределение памяти 37

6.5.3. Строки 37

Приложения 37

Приложение 1. Сообщения и коды ошибок 37

1 Коды ошибок компилятора 37

2 Коды ошибок форматирования 37

3. Коды ошибок математических подпрогамм 37

4. Другие коды ошибок времени выполнения 37

Приложение 2. Блок-схема обработки 37

Приложение 3. Пример обработки 37

Приложение 4. Форматы вызова и интерфейс ассемблера 37

Приложение 5. Формат чисел 37

Приложение 6. Назначение устройств ввода-вывода 37

Приложение 7. Полезная информация о средстве форматирования 37

Приложение 8. Системные имена 37

Пиложение 9. Добавление устройств пользователя 37

Приложение 10. Примеры строковых подпрограмм 37

Приложение 11. Расширенная грамматика SSS Fortran 37

# 1. Введение

SSS Fortran был разработан и реализован, чтобы предоставить для микрокомпьютерных систем полный набор языка Fortran IV и системные возможности. Язык соответствует и превосходит все стандартные функции языка Fortran ANSI X3.9-1966, и аналогичен языкам Fortran, доступным на больших компьютерах.

Системный пакет включает:

**Компилятор Fortran**: Поддерживает расширенные функции, такие как сложные арифметические и символьные переменные и функции. Компилирует со средней скоростью 600 строк в минуту.

**Связывающий загрузчик**: поддерживает полное связывание модулей языка Fortran и ассемблера. Поддерживает системные и пользовательские библиотеки смешанных модулей Fortran и ассемблера.

**Дисковый пакет ввода-вывода**: поддержка последовательного и произвольного доступа, бинарные и ASCII файлы, файлы блочного и потокового режима.

**Пакет времени выполнения**: средство форматирования Fortran, математическая библиотека с подпрограммами двойной точности и комплексными переменными, библиотека встроенных функций, библиотека динамического выделение и связи с языком ассемблера.

Ниже приведена операционная среда, необходимая для использования системы Fortran на микрокомпьютерной системе Z80.

1. Вторичная система хранения, способная поддерживать операционную систему CP/M.
2. Минимум 32 Кбайт ОЗУ (для больших программ может потребоваться больше места для компиляции), который включает в себя пространство, необходимое для работы CP/M.

# 2. Язык SSS Fortran

## 2.1 Введение

Эта глава не является полным учебником по языку Fortran, а скорее руководством по доступным функциям языка для читателя, обладающего знаниями Fortran. Существует много разновидностей Fortran. Каждый крупный производитель компьютерных систем имеет один или несколько вариантов. Хотя некоторые ссылки доступны для основного языка (ANSI Fortran), многие ссылки касаются конкретного варианта, который, скорее всего, является расширением основного языка. SSS Fortran реализует полный стандарт языка Fortran стандарта ANSI X3.9-1966, включая COMPLEX, а SSS Fortran превосходит стандарт ANSI в ряде важных областей.

Эта глава относится к реализации языка Fortran IV, который очень похож на язык SSS Fortran, а именно на язык WATFIV. Предполагается, что читатель обратиться к следующей ссылке в качестве учебника по SSS Fortran или для подробного описания использования стандартных возможностей Fortran.

Fortran IV with WATFOR and WATFIV,

Cress, Dirksen, and Graham

Prentice-Hall Series in Automatic Computation

1970.

## 2.2 Сравнение с WATFIV

### 2.2.1 Типы данных

Компиляторы Fortran, работающие на компьютере с 32-битным (4-байтным) размером слова, обычно поддерживают конструкции данных, которые занимают либо полные слова (4 байта), либо двойные слова (8 байтов), причем некоторые конструкции занимают половину слова (2 байта) или четверть слова (1 байт). Поскольку большинство микрокомпьютеров являются 8-разрядными (1 байт) машинами, компилятор Fortran для такой машины обычно поддерживает конструкции данных, которые несколько отличаются от тех, которые поддерживаются 32-разрядным компилятором Fortran. Это относится к WATFIV (32-разрядный компилятор Fortran и SSS Fortran (8-разрядный компилятор Fortran). Конструкции данных, поддерживаемые каждым, можно резюмировать следующим образом.

|  |  |  |  |
| --- | --- | --- | --- |
| Конструкции данных | WATFIV | | SSS FORTRAN |
| Short Integer | 2 байта  INTEGER\*2  Диапазон: -32768..32767 | | 1 байт  INTEGER\*1  Диапазон: -128..127 |
| Full Integer | 4 байта  INTEGER\*4 или INTEGER  Диапазон: -2147483648..2147483647 | | 2 байта  INTEGER\*2 или INTEGER  Диапазон: -32768..32767 |
| Real | То же самое для обоих  4 байта  REAL\*4 или REAL  Диапазон: примерно от 10-78 до 1076  Точность: примерно 16.2 десятичных цифр | | |
| Double Precision | То же самое для обоих  8 байтов  REAL\*8 или DOUBLE PRECISION  Диапазон: примерно от 10-78 до 1076  Точность: примерно 16.8 десятичных цифр | | |
| Complex | То же самое для обоих  8 байтов  COMPLEX  Диапазон и точность: те же, что и пара REAL | | |
| Extended Complex | 16 байтов  COMPLEX\*16  Диапазон и точность: те же, что и пара REAL\*8 | не поддерживаются | |
| Logical | 1 или 4 байта  LOGICAL\*1 или LOGICAL  Значение: .TRUE. или .FALSE | 1 байт  LOGICAL  Значение: .TRUE. или .FALSE. | |
| Character | от 1 до 255 байтов  Фиксированная длина  CHARACTER\*n  (0<n<256) | от 1 до 32767 байтов  Регулируемая длина до фиксированного максимума  CHARACTER\*n  (0<n<32768) | |
|  | где для обоих, n указывает на n символов ASCII данных CHARACTER означает то же самое что CHARACTER\*1. | | |

Обратите внимание, что, хотя в SSS Fortran тип LOGICAL\*2 недоступен, INTEGER (или INTEGER\*2) могут использоваться как LOGICAL с теми же результатами. Кроме того, в SSS Fortran, .FALSE. - это нулевое значение и .TRUE. - любое ненулевое значение.

### 2.2.2 Ограничения SSS Fortran

WATFIV включил ряд расширений в стандартный Fortran, некоторые из которых не были включены в SSS Fortran. Они кратко излагаются ниже.

**Операторы ENTRY, DOLIST, NAMELIST, PRINT, PUNCH**

Разрешены WATFIV, не разрешены SSS Fortran.

**Массивы**

WATFIV допускает до семи индексов. SSS Fortran разрешает до трех индексов.

**Оператор FORMAT**

WATFIV допускает спецификацию типа Т. SSS Fortran этого не делает. WATFIV позволяет использовать формат G с целыми числами. SSS Fortran этого не делает.

**Оператор PAUSE**

WATFIV разрешает "PAUSE сообщение". SSS Fortran - нет. (Но каждый разрешает PAUSE, а также "PAUSE целое число".)

**Оператор FORMAT- свободный ввод-вывод**

WATFIV разрешает инструкции READ и WRITE для консоли или устройств печати с принятым по умолчанию форматированием. SSS Fortran не разрешает эти формы. Но SSS Fortran включает функции для ввода свободного формата, предназначенные для свободного ввода пользовательских данных в программы Fortran, а также для вывода строк в свободном формате. (См. [Приложение 7](#_Приложение_7._Полезная).)

**Оператор RETURN**

WATFIV разрешает форму оператора RETURN с возвращаемым целым числом (с соответствующими \*-фиктивными параметрами и n аргументами). SSS Fortran не допускает этого.

**Оператор DATA**

WATFIV разрешает инициализацию данных в объявлениях типа. SSS Fortran разрешает инициализацию данных только в инструкциях DATA.

### 2.2.3. Отличие SSS Fortran от WATFIV

Существуют некоторые особенности Fortran, которые реализуются иначе в рамках системы WATFIV. Они кратко излагаются ниже.

**Шестнадцатеричные константы**

WATFIV требует, чтобы шестнадцатеричным константам предшествовала буква Z. Фортран SSS требует, чтобы они были заключены в восклицательные знаки (например, !6520!). Обратите внимание, что в SSS Fortran все шестнадцатеричные константы вводятся как числовые константы в зависимости от числа заданных шестнадцатеричных цифр.

|  |  |
| --- | --- |
| Число шестнадцатеричных цифр | Тип, присвоенный константе |
| 1 или 2  3 или 4  5 до 8  9 или более | INTEGER\*1  INTEGER\*2  REAL\*4  REAL\*8 |

Если шестнадцатеричная константа короче, чем будет предоставлен тип данных, то заполнение с нулями выполняется справа. Если предварительно указано более 16 шестнадцатеричных цифр, то происходит усечение справа.

**Оператор FORMAT**

В WATFIV, используя спецификацию Zn (hex) при выводе, если элемент данных имеет более n цифр, выводятся самые правые цифры. В SSS Fortran выводятся самые левые цифры.

**Оператор IMPLICIT**

WATFIV разрешает только один оператор IMPLICIT в (под) программе. SSS Fortran разрешает несколько, но требует, чтобы в любом операторе IMPLICIT, был объявлен только один неявный тип. Например:

**IMPLICIT REAL (A-V), INTEGER (X,Y)**

в WATFIV, в SSS Fortran должен быть записан:

**IMPLICIT REAL (A-V)**

**IMPLICIT INTEGER (X,Y)**

**Оператор READ**

WATFIV использует END=Label и ERR=Label для передачи управления в конце файла или ошибках ввода-вывода. SSS Фортран использует ENDFILE=Label и ERREXIT=Label, чтобы сделать то же самое.

**Спецификации длины**

WATFIV допускает спецификации длины отдельных элементов в декларации. SSS Fortran разрешает спецификацию длины только для явной спецификации типа. Например:

**CHARACTER\*2 A(30), B\*4, C\*5(3)**

в WATFIV, в SSS Fortran должен быть записан:

**CHARACTER\*2 A(30)**

**CHARACTER\*4 B**

**CHARACTER\*5 C(3)**

**Ввод-вывод прямого доступа**

WATFIV использует оператор DEFINE FILE, а также оператор FIND. SSS Fortran использует вызовы системных подпрограмм для реализации этих функций.

**Вызов по ссылке**

WATFIV использует вызов по значению для передачи параметров, если это возможно, и вызов по ссылке может быть определен, заключив фиктивный параметр в символы наклонной черты. SSS Fortran по умолчанию использует более стандартную систему вызова по ссылке (передается адрес элементов) и вызовы по значению могут быть определены, заключив параметр в круглые скобки. Например:

**CALL SUB (A, (2.0) )**

В SSS Fortran A передается по ссылке (ее значение может быть изменено подпрограммой), в то время как 2.0 передаетя значением (создается копия и передается в подпрограмму). В этом случае константа 2.0 не может быть изменена вызванной программой. Если скобки убраны, и используется вызов по местоположению, можно изменить константу в вызываемой процедуре.

**Внутренний (Core-to-Core) ввод-вывод**

Эта функция позволяет SSS Fortran использовать элемент CHARACTER вместо спецификации устройства в форматном операторе READ или WRITE. Таким образом, данные могут считываться или записываться в элемент CHARACTER вместо устройства ввода-вывода, обеспечивая передачу из памяти в память или Core-to-Core. Например:

**CHARACTER\*15 A**

**DATA A / 1 2 3 /**

**READ(A,100) I,J,K**

**100 FORMAT (3I3)**

I, J и K считываются из символьной строки **A**, с заданным FORMAT. Внутренний ввод вывод в SSS Fortran совместим с функциями Core-to-Core WATFIV за следующими исключениями:

1. SSS Fortran допускает внутренние операции только в форматном вводе вывода;
2. SSS Fortran в качестве спецификаций устройств допускает только переменные или константы CHARACTER (имена массивов CHARACTER или элементы массива не допускаются).

В SSS Fortran любые сообщения об ошибках, относящиеся к внутреннему вводу вывода, указывают номер устройства 64.

### 2.2.4. Расширения SSS Fortran

Кроме того, SSS Fortran реализует некоторые расширения, которые не разрешены в WATFIV. Кратко они излагаются ниже.

**Литеральные константы**

Строка символов может быть обозначена одним из трех способов:

1. Строка может быть заключена в апострофы (одинарные кавычки).
2. Строка может быть заключена в двойные кавычки (").
3. Строке может предшествовать wH, где w - количество символов в строке (спецификация Холлерита).

Тип 1 и Типа 3 это литералальные константы в стиле ANSI, которые обычно используются в Fortran. То есть, они являются строками с фиксированной длиной, без дополнительной информации, хранящейся в строке. Строковые константы типа 2 в SSS Fortran совместимы с переменной типа CHARACTER. Текущая длина, максимальная длина и информация об указателе сохраняются отдельно от строки. Обратите внимание, что любые из вышеупомянутых типов литеральных констант могут использоваться в операторах DATA или FORMAT.

В качестве дополнительного расширения SSS Fortran позволяет использовать 1-байтные шестнадцатеричные спецификации в строках Типа 1 или Типа 2. Шестнадцатеричные 1-байтовые значения в строках ограничены левым и правым скобками ([]). Если левая скобка должна быть включена в строку Типа 1 или Типа 2, необходимо использовать две последовательные левые скобки. Например:

**INTEGER\*1 A(10)**

**DATA A / [OD]MESSAGE[OD][[ /**

Массив A инициализируется 10 байтами данных

**cr M E S S A G E cr [**

где **cr** представляет собой возврат каретки ASCII.

**Сравнения строк**

Как переменные CHARACTER, так и строки Типа 2 (двойные кавычки) могут сравниваться с помощью арифметических операторов сравнения. Сравнение использует словарный порядк ASCII с .LT. A. Две строки .EQ. если у них одинаковая длина и их символы совпадают. Например

**A < AB < B**

**Короткие целые константы**

Константа INTEGER в диапазоне -128..127 может быть записана с завершающим знаком решетки (как, например, 15#). Такая константа занимает только один байт, вместо обычных двух байт.

**Оператор вычисляемого GOTO**

Индекс оператора вычисляемого перехода может быть арифметическим выражением.

**Оператор DO**

Начальное значение, тестовое значение и приращение оператора DO могут быть произвольными арифметическими выражениями. Более того, индекс DO может быть либо именем переменной, либо элементом массива. Начальные, тестовые, инкрементные и индексные элементы должны быть одного и того же типа данных. SSS Fortran также допускает циклы DO с отрицательными значениями приращения. Индекс DO может быть REAL. Например:

**DO 10 A(I) = 50.0, 20.EO, -1.0**

Предполагая, что A - массив типа REAL, этот оператор DO допустим в SSS Fortran.

**Списки вывода**

Произвольные выражения разрешены в списках вывода. Однако заключенные в скобки простые списки, такие как

**WRITE (1) (B,C)**

не разрешены, так как они иногда неотличимы от комплексных констант и заключенных в скобки выражений. Кроме того, если какое-либо выражение списка выводп включает вызов функции, эта функция не должна выполнять ввод/вывод.

**Спецификация неявного DO**

Начальное значение, тестовое значение и инкремент в спецификации неявного DO могут быть произвольными арифметическими выражениями. Но в этом случае, в отличие от оператора DO, несмотря на то, что индекс может быть отличным от целого числа, он не может быть элементом массива.

**Несколького операторов**

В строке можно ввести несколько операторов, разделенных символом точки с запятой (;). Если в последующем операторе должен быть номер оператора, то он просто задается в начале этого оператора. В этих последующих заявлениях фактически отсутствует колонка 6. Одно исключение состоит в том, что после инструкции FORMAT в сроке не могут распологаться дополнительные операторы. следующая строка является допустимой:

**X=Y;123CONTINUE; 4567PAUSE; GOT033; 22 STOP**

**Индексы**

SSS Fortran в качестве индексов разрешает использовать произвольные арифметические выражения.

**Строки продолжения**

Нет ограничений на количество строк продолжения, разрешенных в SSS Fortran, если операторы причвоения распознаются как таковые в трех строках.

**Операторы INCREMENT и DECREMENT**

Если I является именем переменной, SSS Fortran разрешает

**I = I + 1**

записать как

**INCREMENT I**

и

**I = I - 1**

как

**DECREMENT 1**

что позволяет компилятору генерировать гораздо более эффективный код. Обратите внимание, что переменная, используемая в операторе INCREMENT или DECREMENT, должна быть простой переменной INTEGER (или INTEGER\*2) и что она не может быть параметром подпрограммы.

**Логические выражения**

Логическая операция исключающего "ИЛИ" (.XOR.) разрешен в SSS Fortran и имеет тот же уровень приоритета что и логическое сложение (.OR.). Исключающего ИЛИ имеет следующие значения:

X Y X .XOR. Y

True True False

True False True

False True True

False False False

**Маскирование выражений**

В SSS Fortran однобайтовые или двухбайтовые величины (INTEGER\*1, INTEGER\*2, одно - четырех значные шестнадцатеричные константы, значения LOGICAL), могут быть объединены в маскируемые выражения.

**.AND.** поразрядное "И"

**.OR.** поразрядное включающее "ИЛИ"

**.NOT.** Поразрядное дополнение (отрицание)

**.XOR.** поразрядное исключающее "ИЛИ"

Приоритет операторов такой же, как и с логическими выражениями. Например, следующий оператор присваивания маскирования обменивает самые правые два бита целого числа, I:

**I = (I.AND..1)\*2 .OR. (I.AND.2#) .OR. I.AND.!FFFC!**

**Оператор FORMAT**

Форматированная запись обычно завершается с помощью последовательности возврата каретки/перевода строки (CR/LF), отправляемой устройству вывода. Замена правой правой скобки оператора FORMAT на знак доллара ($) приведет к отключению передачи окончательного CR/LF. Это особенно полезно для интерактивных программ. Например, обычная последовательность:

**WRITE (TTY,100)**

**100 FORMAT ( TYPE 1 OR 2, PLEASE: )**

**READ (TTY,200) ANSR**

**200 FORMAT (Il)**

создает на терминале следующий вывод:

**TYPE 1 OR 2, PLEASE: [CR/LF]**

**[user responds here]**

Тогда как изменение FORMAT 100 на

**100 FORMAT ( TYPE 1 OR 2, PLEASE: $**

произведет

**TYPE 1 OR 2, PLEASE: [user responds here]**

**Комплексные константы**

В WATFIV константы COMPLEX должны быть определены как два значения REAL. В SSS Fortran, несмотря на то, что константа COMPLEX всегда сохраняется как два значения REAL, константа может быть определена, используя или DOUBLE PRECISION, или целые числа. Например, все следующие комплексные константы - эквивалентны в SSS Fortran.

**(1,2) (1.0,2.0) (1D0,2D0) (1,2D0)**

**Оператор логического IF**

Так как значения INTEGER\*1 и INTEGER\*2, в SSS Fortran могут использоваться для хранения логических значений, любой из этих типов данных может использоваться в операторе логического IF. Кроме того, если тип выражений ответвления не соответствует INTEGER или INTEGER\*1, оно будет преобразовано, если это возможно. Таким образом, в SSS Fortran, все следующие логические IF допустимы.

**INTEGER\*2 I**

**REAL R**

**IF (I) GOTO 10**

**IF (I+3 .AND. 7) GOTO 10**

**IF (R+1.0) GOTO 10**

## 2.3. Краткий обзор типов операторов

### 2.3.1. Выподняемые операторы

**Оператор арифметического присваивания**

**V = E**

где V имя переменной или элемент любого типа кроме CHARACTER. E - арифметическое выражение, или текстовая строка ANSI типа (строка, заключенная в одинарные кавычки или строка Холлерита).

*Примечание*: В случае, если V и E имеют различные типы, выполняется преобразование. Для процесса преобразования используются стандартные правила Fortran. Т.е. в случае оператора присваивания E преобразуется к типу V до выполнения присвоения. В арифметических выражениях для операторов отличных от присвоения, если два элемента имеют различные типы, выполняется стандартное восходящее преобразование. Это восходящее преобразование:

**INTEGER\*1 < INTEGER\*2 < REAL\*4 < REAL\*8 < COMPLEX**

Обратите внимание, что литералы стиля ANSI (строки в одинарных кавычках и строки Холлерита) могут использоваться везде, где разрешены общие выражения. В этих случаях приведение выполняется со строкой, принимающей тип элемента непосредственно с левой стороны от него, с усечением или дополнением пробелами при необходимости. Например,

REAL A

A = **'**123**'**

Поскольку A является REAL (4 байта), строка должна быть 4 байта (**'**123**'**) прежде чем присвоение будет выполнено.

**IF (I .EQ. '1234') GOTO 1**

Так как I - INTEGER (2 байта), строка преобразуется в 2 байта (**'**12**'**), прежде чем сравнение будет выполнено.

Литералы ANSI стиля, используемые в качестве параметров подпрограмм, передаются в неизменном виде (т.е. приведение не выполняется). Например:

**CALL XYZ ('123456789', 5HABCDE)**

строки не изменяются каким-либо образом. Адреса элементов **123456789** и **ABCDE** передаются подпрограмме.

**Оператор присваивания символов**

**V = E**

Где V - имя переменной или элемент массива типа CHARACTER\*n. E имеет тип CHARACTER\*n и является либо константой (строкой в двойных кавычках), либо именем переменной, либо элементом массива, либо вызовом функции.

*Примечание*: в данном типе оператора не требуется выполнить преобразование, поскольку все элементы имеют одинаковый тип.

**Оператор присваивания маски**

**V = E**

где V - имя переменной или элемент массива любого типа, кроме CHARACTER. E - это выражение маскировки.

*Примечание*. Преобразование происходит из INTEGER к типу V, как при арифметическом присваивании.

**Оператор INCREMENT**

**INCREMENT V**

где V - имя переменной INTEGER, которое не является параметром подпрограммы.

**Оператор DECREMENT**

**DECREMENT V**

где **V** совпадает с INCREMENT.

**Оператор GO TO**

**GO TO Label**

где **Label** - это метка оператора исполняемого оператора.

**Оператор вычисляемого GO TO**

**GO TO (List), E**

где **List** - разделенный запятыми список меток операторов исполняемых операторов.  
**E** - арифметическое выражение.

*Примечание*: E вычисляется и преобразуется в тип INTEGER\*2. Если результирующее значение меньше нуля или больше числа операторов в списке, то выполнение продолжается с инструкцией, следующей за вычислением.

**Оператор ASSIGN**

**ASSIGN Label TO V**

где **Label** - это метка исполняемого оператора. **V** - это имя переменной или элемента массива типа INTEGER или INTEGER\*2.

**Оператор назначаемого GO TO**

**GO TO V, (List)**

где **V** имя переменной или элемент массива типа INTEGER. **List** - список разделенных запятой меток исполняемых операторов, один из которых был ранее присвоен V.

*Примечание*: Список должен состоять из меток операторов, определенных в других местах программы, но проверка не выполняется, чтобы определить, соответствует ли значение V любой из меток в List.

**Оператор арифметического IF**

**IF (E) Label\_1, Label\_2, Label\_3**

где **E** - арифметическое выражение. **Label\_1**, **Label\_2** и **Label\_3** являются метками исполняемых операторов.

**Оператор логического IF**

**IF (E) S**

где E - любое арифметическое выражение. S - это любой непомеченный исполняемый оператор, за исключением DO и логического IF. Результат вычисления выражения будет преобразован в целочисленное значение, и если результат равен 0 (логическое значение .FALSE.), S не будет выполняться. Если результат не равен нулю (логическое значение .TRUE.), S будет выполняться.

**Оператор DO**

**DO Label V = E1, E2, E3**

или

**DO Label V = E1, E2**

где **Label** - это метка исполняемого оператора, который является целевым или конечным оператором DO. V - переменная или элемент массива, называемая управляющей переменной. **E1**, **E2** и **E3** являются арифметическими выражениями. E1 называется начальным выражением, E2-тестовым выражением и E3-выражением приращения.

*Примечание*: E1, E2 и E3 могут иметь любой арифметический тип, в то время как V может иметь любой арифметический тип за исключением COMPLEX. Если какой-либо из E1, E2 или E3 не соответствует типу V, то они преобразуются для соответствия. Например,

**REAL R**

**INTEGER I**

**DO 10 R = 1, 4DO, I**

эквивалентно

**DO 10 R = 1.0, 4.0, FLOAT(I)**

**Оператор CONTINUE**

**CONTINUE**

**Оператор PAUSE**

**PAUSE**

или

**PAUSE integer**

**Оператор STOP**

**STOP**

или

**STOP integer**

**Оператор RETURN**

**RETURN**

**Оператор READ**

**READ (U, F, ENDFILE=Label\_1, ERREXIT=Label\_2) List**

где **U** - целое число или имя переменной (простая INTEGER переменная или простая переменная CHARACTER), которая задает номер устройства ввода, или имя символьной переменной для ввода "память-память" (см. [Приложение 6](#_Приложение_6._Назначение)). **F** - это метка оператора FORMAT, описывающая данные на форматированном устройстве ввода, или имя массива, или переменной CHARACTER, содержащая спецификацию FORMAT, описывающая данные на форматированном устройстве ввода. **Label\_1** и **Label\_2** являются метками исполняемых операторов. **List** - это список ввода-вывода.

*Примечание*: **F**, **ENDFILE=Label\_1**, **ERREXIT=Label\_2** и **List** являются необязательными.

**Оператор WRITE**

**WRITE (U, F, ENDFILE=Label\_1, ERREXIT=Label\_2) List**

где **U** - целое число или имя переменной (простая INTEGER переменная или простая переменная CHARACTER), которая задает номер устройства вывода, или имя символьной переменной для вывода "память-память" (см. [Приложение 6](#_Приложение_6._Назначение_1)). **F** - это метка оператора FORMAT, описывающая данные на форматированном устройстве вывода, или имя массива, или переменной CHARACTER, содержащая спецификацию FORMAT, описывающая данные на форматированном устройстве вывода. **Label\_1** и **Label\_2** являются метками исполняемых операторов. **List** - это список ввода-вывода.

*Примечание*: **F**, **ENDFILE=Label\_1**, **ERREXIT=Label\_2** и **List** являются необязательными.

**Оператор READ прямого доступа**

**READ (U/R, F, ENDFILE=Label\_1, ERREXIT=Label\_2) List**

где **U**, **F**, **Label-1**, **Label-2** и **List** являются такими же, как в операторе READ. **R** - это выражение, которое указывает, какая запись должна быть прочитана.

**Оператор WRITE прямого доступа**

**WRITE (U/R, F, ENDFILE=Label\_1, ERREXIT=Label\_2) List**

где **U**, **F**, **Label-1**, **Label-2** и **List** являются такими же, как в операторе WRITE. **R** - это выражение, которое указывает, какая запись должна быть записана.

**Оператор REWIND**

**REWIND U**

где **U** - целое число или имя переменной, определяющее номер устройства.

**Оператор BACKSPACE**

**BACKSPACE U**

где **U** - целое число или имя переменной, определяющее номер устройства.

**Оператор ENDFILE**

**ENDFILE U**

где **U** - целое число или имя переменной, определяющее номер устройства.

**Оператор CALL**

**CALL N**

или

**CALL N list**

где **N** - символическое имя вызываемой подпрограммы. **List** - список разделенных запятой значений выражений, имен массива, имен подпрограммы и имен функций, которые являются *аргументами* или *фактическими параметрами* для вызова подпрограммы.

### 2.3.2. Неисполняемве операторы

**Оператор явного описания типа**

**Type List**

где **Type** - один из следующих *допустимых типов*:

INTEGER\*1

INTEGER\*2

INTEGER

REAL\*4

REAL

REAL\*8

DOUBLE PRECISION

COMPLEX

LOGICAL

CHARACTER

CHARACTER\*n

CHARACTER\*32767

**List** - список разделенных запятой значений имен переменной, имен функций, имен массива и объявлений массива.

*Примечание*: для массива с размерами, заданными в операторах DIMENSION или COMMON, может быть явно указан тип, но он не может быть переопределен в объявлении типа, т. е. он может выглядеть как имя массива, но не как объявление массива.

**Оператор описания типа IMPLICIT**

**IMPLICIT Type (List)**

где **Type** - одни из допустимых типов. **List** - список разделенных запятой значений одиночных алфавитных букв и троек буква-знак минус-буква (например, P-W) представляющин первые и последние символы диапазона.

*Примечание*: В объявлениях IMPLICIT CHARACTER\*n, n не может превышать 255. Это не ограничивает максимальную длину символьных строк в целом, но действительно ограничивает максимальную длину переменной CHARACTER, тип которой объявляется с помощью IMPLICIT.

**Оператор DIMENSION**

**DIMENSION List**

где **List** - список разделенных запятой объявлений массива.

**Оператор COMMON**

**COMMON List\_1 /Name\_2/List\_2 ... /Name\_n/List\_n**

или

**COMMON /Name\_1/List\_1 /Name\_2/List\_2 ... /Name\_n/List\_n**

где **List\_1**, **List\_2**, и т.д. является списками разделенных запятой имен переменных, имен массива и объявлений массива. **Name\_1**, **Name\_2**, и т.д. является или именами *помеченных* COMMON *блоков* или является пробелом (обозначая *непомеченный* COMMON *блок*).

***Примечание 1***: в каждом операторе COMMON может иметься один, два или несколько списков COMMON блоков.

***Примечание 2***: Массив с его размерами, заданными в операторе DIMENSION или COMMON, может быть явно типизирован, но не может быть переопределен в объявлении типа, т. е. он может отображаться в операторе объявления типа как имя массива, но не как объявление массива.

***Примечание 3***: Элемент типа CHARACTER\*n хранится в COMMON без информации о длине, т. е. просто как массив символов (см. [Раздел 2.4 п.8](#r248)).

**Оператор EQUIVALENCE**

**EQUIVALENCE (List\_1), (List\_2), ... (List\_n)**

Где **List\_1**, **List\_2** и т.д. (Называемые *списками эквивалентности*) представляют собой разделенные запятыми списки двух или более имен переменных, имен массивов или элементов массива.

***Примечание***. В операторе EQUIVALENCE может быть один, два или несколько списков эквивалентности.

**Оператор EXTERNAL**

**EXTERNAL List**

где **List** - список разделенных запятой значений имен подпрограмм.

**Оператор DATA**

**DATA Vlist\_1/Dlist\_1/, Vlist\_2/Dlist\_2/, ... Vlist\_n/Dlist\_n/**

где **Vlist\_1**, **Vlist\_2** и т. д. (называемые *списки переменных*) представляют собой разделенные запятыми списки имен переменных, имен массивов или элементов массива, которые необходимо инициализировать. **Dlist\_1**, **Dlist\_2** и т. д. (называемые *списки инициализации*) - это списки, разделенные запятыми, констант инициализации. Каждой константе может предшествовать *коэффициент повторения* в форме R\*, где R - целое *число повторений*.

*Примечание*: SSS Fortran в опрераторах DATA имеет два режима инициализации. Режим 1 является стандартным режимом ANSI. Режим 2 является расширением, которое применяется только для инициализации массивов. Каждый из этих режимов объясняется ниже.

**Режим 1**: Это - обычный режим ANSI инициализации данных. В этом режиме могут быть инициализированы только простые переменные или элементы массива, т.е. имена массивоа без индексов не могут использоваться. В этом режиме применяются следующие правила.

1. Для каждой переменной, которая должна быть инициализирована, должна появиться одна константа.
2. Если константа и соответствующая переменная имеют одинаковую длину, никаких изменений в константе не происходит.
3. Если длины отличаются, усечение или дополнение (в зависимости от необходимости) делается справа. Строки дополняются пробелами и другими константами (с одним исключением) заполняются нулями.
4. Исключением является следующее. Если 2-байтовая переменная инициализируется 1-байтовой целой константой, то константа не заполняется нулями, а используется эквивалентное представление константы в 2-байтовом формате.
5. Все три типа строк (в одинарных кавычках, двойных кавычках и холлерита) в списках инициализации эквивалентны.
6. Для переменных CHARACTER\*n в списке переменных текущая длина устанавливается равной длине соответствующего элемента данных в списке инициализации до любого усечения или заполнения (см. пример ниже).

Примеры инициализации режима 1.

**REAL A,B,C**

**DATA A,B,C / 1.0, 2.0, 3.0 /**

В данном случае не требуется усечение или заполнение, поскольку все элементы REAL (4 байта).

**REAL A,B,C**

**DATA A,B,C / !0102030405!, 'ABC' , 1# /**

Так как все переменные A, B, и C типа REAL (4 байта), все константы в списке инициализации должны быть сделаны длиной 4 байта. Шестнадцатеричная строка будет усечена до !01020304!, строка дополннится до ABC[0], и константа INTEGER\*1 будет дополненав справо нулями с тем, чтобы ее шестнадцатеричное представление было !01000000!.

**CHARACTER\*8 A,B,C**

**DATA A,B,C / '12345', '12345', 5H12345 /**

Все переменные A, B и C инициализируются эквивалентом 8H12345. Обратите внимание, что так как A, B и C переменные CHARACTER\*n, текущая длина каждой из них задается оператором DATA. Поскольку длина каждой константы в списке инициализации равна 5 перед заполнением, текущая длина каждой из A, B и C равна 5.

**INTEGER A**

**LOGICAL B**

**DATA A,B / .TRUE., .FALSE. /**

Поскольку переменная B имеет тип LOGICAL, как и соответствующая константа инициализации, никакие изменения не требуются. Переменная A имеет тип INTEGER (2 байта), в то время как ее соответствующая константа инициализации является 1 байтной LOGICAL. В этом случае применяется правило d выше, и так как .TRUE. представляся как все 1, A инициализируется шестнадцатеричным !FFFF!.

**Режим 2**: Это - расширенный режим инициализации, который применяется только для инициализации массивов. В этом режиме имена массивов без индекса могут использоваться в списке переменных. Таким образом, этот режим позволяет инициализировать весь массив, не определяя в списке переменных каждый элемент отдельно. Следующие правила применяются к операторам DATA режима 2.

1. В списке инициализации необходимо указать достаточно много констант для заполнения всего массива.
2. Константы дополняются справа к кратному числу длины элемента инициализируемого массива. Усечение не происходит за одним исключением.
3. Исключением является то, что если массив имеет тип INTEGER\*1 или LOGICAL, то любые константы INTEGER\*2 усекаются слева до 1 байта. Если массив имеет тип INTEGER\*2 или INTEGER, любые константы INTEGER\*1 или LOGICAL преобразуются в 2-байтовый эквивалент.
4. Как и в режиме 1, все стили строк эквивалентны.
5. Для массивов типа CHARACTER\*n одна константа должна быть указана в списке инициализации для каждого элемента инициализированного массива. Текущая длина каждого элемента массива устанавливается до длины соответствующего элемента данных до любого дополнения.

Примеры инициализации режима 2.

**REAL A(3)**

**DATA A / 1.0, 2.0, 3.0 /**

Преобразование не требуется. A - 12 байтов, и константы - 12 байтов.

**INTEGER A(9)**

**DATA A / '12345', '678', 8HABCDEFGH /**

Поскольку массив A имеет тип INTEGER (2 байта на элемент), каждый элемент данных дополняется до составных 2 байтов: Таким образом A составляет 18 байтов и инициализируется эквивалентом 18H12345 678 ABCDEFGH.

**INTEGER A(9)**

**DATA A / 6\*0, 2\*2, -1# /**

Единственное преобразование состоит в том, что 1-байтная константа преобразуется в ее 2-байтовый эквивалент, а именно: hex !FFFF!.

***Примечание 1***: Кроме того, Режим 1 и Режим 2 могут присутствовать вместе в едном операторе DATA. Например,

**INTEGER A(2),B**

**DATA A,B / 1, 2, 3 /**

***Примечание 2***: В соответствии с требованиями ANSI многоразмерные массивы заполняются по столбцам, т. е. первый индекс, изменяется наиболее быстре, а последний индекс наименее быстро.

Например,

**INTEGER A(2,3)**

**DATA A / 1, 2, 3, 4, 5, 6 /**

инициализация имеет эффект

**A(1,1)=1; A(2,1)=2; A(1,2)=3; A(2,2)=4; A(1,3)=5; A(2,3)=6**

Вышеупомянутые правила могут сначала казаться чрезмерно сложными. Но поскольку SSS Fortran был расширен, включив в него тип данных CHARACTER\*n, 1-байтовые константы и режим 2 инициализациb массива, стандартные правила Fortran должны быть расширены разумным образом, так что логические результаты будут иметь место. Следует помнить, что стандартные результаты получаются при использовании стандартных функций. Расширенные функции могут потребовать привыкания к ним, но могут быть удобными функциями.

**Оператор END**

**END**

**Оператор BLOCK DATA**

**BLOCK DATA**

**Оператор SUBROUTINE**

**SUBROUTINE N**

или

**SUBROUTINE N list**

где **N** - символическое имя подпрограммы. **List** - список разделенных запятой значений символических имен, которые являются *фиктивными* параметрами подпрограммы.

**Оператор FUNCTION**

**FUNCTION N list**

или

**Type Function N list**

где **N** - символическое имя подпрограммы. **List** - список разделенных запятой значений символических имен, которые являются *фиктивными* параметрами подпрограммы. **Type** - один из допустимых типов.

**Оператор-функция**

**N (List) = E**

где **N** - символическое имя подпрограммы. **List** - список разделенных запятой значений символических имен, которые являются *фиктивными* параметрами подпрограммы.  
**E** - выражение.

**Оператор FORMAT**

**Label FORMAT (List)**

или

**Label FORMAT list**

где **Label** - метка оператора FORMAT. **List** - список разделенных запятой спецификаций полей,

и

Спецификация полей - любой дескриптор редактирования:

**rIw**  (INTEGER FORMAT Code)

**rFw.d**  (REAL FORMAT Code)

**rDw.d**  (DOUBLE PRECISION FORMAT Code)

**rEw.d**  (REAL FORMAT Code)

**rLw**  (LOGICAL FORMAT Code)

**rAw**  (CHARACTER FORMAT Code)

**rZw**  (Hexadecimal FORMAT Code)

**rGw.d**  (REAL FORMAT Code)

**wHXX...X** (Hollerith Literal Character FORMAT Code)

**wX**  (Spacing FORMAT Code)

(End-of-record FORMAT Code)

**nP** (масштабный коэффициент)

или групповая спецификация FORMAT:

**r list**

где

**r** (коэффициент повторения) необязательная целая константа без знака.

**w** (спецификация ширины) целая константа без знака.

**d** (дробные десятичные разряды) целая константа без знака.

**n** (масштабный коэффициент) необязательная знаковая целая константа.

***Примечание 1***: запятая может быть опущена до или после дескриптора редактирования косой черты (/).

***Примечание 2***: запятые должны быть опущены после дескриптора редактирования **nP**, а **nP** должен быть сразу же после дескриптора редактирования **D**, **E**, **F** или **G**.

***Примечание 3***: См. [Приложение 7](#_Приложение_7._Полезная_1) для ввода-вывода в свободном формате и дополнительную информацию об операции форматирования.

## 2.4. Примечания

В следующих примечаниях описывается дополнительная информация, относящаяся к языку SSS Fortran.

1. В командах и именах переменных SSS Fortran разрешены только прописные буквы. В литеральной строке может использоваться любой символ, кроме возврата каретки, который во всех случаях является зарезервированным символом.
2. Входные исходные строки могут иметь любую длину, завершаясь символами возврата каретки / перевода строки. В результате этого столбцы 72-80 не используются в информационной последовательности.
3. Код SSS Fortran повторно используемый и рекурсивный. Однако возможности автоматического хранения недоступны, за исключением динамического выделения пользователем. Все переменные Fortran являются статическими.
4. SSS Fortran имеет следующие ограничения в порядке следования операторов:
5. Операторы IMPLICIT должны предшествовать всем другим операторам в основной программе и должны непосредственно следовать за оператором FUNCTION или SUBROUTINE в подпрограмме. Операторы IMPLICIT применяются в порядке, в котором они появляются в программе. Любые конфликты в спецификациях IMPLICIT устраняются их упорядочиванием.
6. Операторы DATA для данной переменной должны следовать за операторами типа или DIMENSION для переменной.
7. За операторами определения функций должны следовать все прочие неисполняемые операторы.
8. Типы возвращаемые функциями должны соответствовать типу, заданному функции в вызывающей программе, или будут отображаться неопределенные результаты. Единственным исключением из этого правила является то, что типы INTEGER\*1, INTEGER\*2 и LOGICAL являются эквивалентными для возврата функций.
9. Иногда желательно передавать значения LOGICAL или INTEGER\*1 в подпрограммы, которые ожидают аргументы INTEGER\*2 или INTEGER. Это возможно с помощью "вызов по значению". Например:

**LOGICAL Q**

**10 CALL SUB((10#),(Q))**

**SUBROUTINE SUB(A,B)**

**INTEGER\*2 A,B**

даст ожидаемые результаты, тогда как, удаление дополнительных круглых скобок в операторе 10 выше нет, так как будет использоваться вызов по ссылке.

1. Оператор EXTERNAL может использоваться для определения переменных внешними, если они не являются параметрами. В случае символьных строк или массивов, дескриптор элемента может быть внешним, и совпадать с именем элемента (см. [Приложение 4](#_Приложение_4._Форматы) для получения дополнительной информации о дескрипторе массива). Переменная, объявленная как EXTERNAL, не может быть использована в EQUIVALENCE или инициализирована с помощью оператора DATA.
2. Переменные CHARACTER или строковые константы, используемые в COMMON блоке, имеют общие значения, а не их текущую или максимальную длину. Рассмотрим следующий пример.

CHARACTER\*12 STRING

COMMON STRING

DATA STRING /ABCDE/

CALL XXX

STOP

END

SUBROUTINE XXX

CHARACTER\*12 STRING

COMMON STRING

WRITE(1,100) STRING

100 FORMAT(1X,A0)

RETURN

END

Эта программа печатает пустую строку. Чтобы достичь того, что, вероятно, было запланировано, следует использовать следующую программу.

CHARACTER\*12 STRING

COMMON STRING, LENGTH

DATA STRING / ABCDE/

LENGTH=KLEN(STRING)

CALL XXX

STOP

END

SUBROUTINE XXX

CHARACTER\*12 STRING

COMMON STRING, LENGTH

CALL SETLEN(STRING,LENGTH)

WRITE(1,100) STRING

100 FORMAT(1X,A0)

RETURN

END

Эта реализация допускает правильное наложение строк в COMMON. Например, строка длиной 100 может быть наложена поверх двух строк длиной 50 следующим образом:

CHARACTER\*50 STR1, STR2

COMMON STR1, STR2

SUBROUTINE XXX

CHARACTER\*100 STRING

COMMON STRING

CALL SETLEN(STRING,100)

# 3. Использование компилятора

## 3.1. Общее использование

Эта глава предполагает знание операционной системы CP/M.

Компилятор является модулем в исполняемой форме с именем FOR.COM. Для компиляции программы необходимы два файла, входной и выходной файлы. Входной файл должен содержать исходный код Fortran для компиляции, тогда как выходной файл будет создан компилятором и будет содержать представление объектного кода данного исходного кода. Созданный компилятором объектный файл обычно имеет двоичную форму. Компилятор также создает листинг источника и распределение памяти на консольном устройстве или, используя переключатель ;DISK в файле на диске.

Основная форма для компиляции исходного модуля, названного, скажем, INP.FOR

**A>FOR INP.FOR OUT.REL**

Выходной файл в этом случае будет называться OUT.REL. Если расширение входного файла не задано, используется значение по умолчанию .FOR. Расширение выходного файла по умолчанию - .REL.

**A>FOR INP OUT**

Приведенного выше достаточно, если файл INP имеет расширение .FOR, а выходной файл будет назван OUT.REL. Если файл с тем же именем, что и выходной файл, уже существует при выполнении команды компиляции, то существующий файл удаляется, и открывается новый файл с таким же именем.

Если выходной файл не задан, для формирования имени выходного файла используется имя входного файла с расширением .REL. Например,

**A>FOR INP**

Компилирует INP.FOR в INP.REL. Также могут использоваться стандартные спецификации устройства, используемые в CP/M.

**A>B:FOR B:INP**

предполагает, что компилятор и входной файл находятся на устройстве B:, в то время как выходной файл с именем INP.REL будет находиться на зарегистрированном устройстве.

## 3.2. Опции переключателей компилятора

Для управления выполнением компилятора доступны несколько следующих переключателей:

**;LISTOFF**

Отключает печать исходного файла и распределения памяти, так что вся информация (сообщения об ошибках и имена идентификаторов) отправляется на консольное устройство. По умолчанию - LISTON, в этом случае вся информация поступает на устройство печати.

**;MAP**

Отключает оператор и распределение памяти переменных, которые создается по умолчанию. Этот переключатель, используемый вместе с переключателем ;LISTOFF, включает карту. Таким образом, ;LISTOFF;MAP не производит листинг источника, но создает карту распределения переменных.

**;OUTOFF**

Определяет, что для этой текущей компиляции объектный файл не создается. Значение по умолчанию - OUTON. Этот переключатель полезен для проверки синтаксиса программ.

**;PAGE**

Обычно любая информация, отправленная во время компиляции на устройство печати, разбивается на пронумерованные страницы по 66 строк в каждой, без особых попыток выравнивания информации по границам страниц. Этот переключатель указывает, что каждая подпрограмма и ее карта должны быть привязаны к новой странице, а остаток страницы заполняется пустыми строками, до 66 строк. Этот переключатель полезен для тех, у кого есть линейные принтеры, и требуется сегментированный вывод.

**;HEXOUT**

Этот флаг определяет, что компилятор производит объектный модуль для текущей компиляции в формате HEX-ASCII. Этот переключатель может быть полезен для хранения объектных модулей на перфоленте, но его основной целью является оказание помощи в обслуживании компилятора (см. [Раздел 3.3 Примечания](#_C._Примечания)). По умолчанию формат вывода двоичный.

**;DISK**

Этот флаг определяет, что компилятор выводит листинг в файл на диске (с расширением .LST), а не на устройство печати.

Переключатели компилятора, указанные выше, могут быть использованы в командной строке и должны быть указаны после задания информации о файлах. Порядок следования переключателей не имеет значения и важна только первая буква после символа ";".

Примеры:

**A>FOR TEST ;L ;M**

Указывает, что листинг не будет произведен, но будет создана карта распределения.

**A>FOR TEST ;O ;M ;D**

определяет входной файл TEST.FOR без создания выходного объектного файла или карты распределения и с выводом листинга, записываемого в файл TEST.LST.

## 3.3. Примечания

Следующие примечания относятся к компилятору и его использованию.

1. Символы табуляции (Ctrl-I) могут быть включены во входной поток компилятора. Первой позицией табуляции служит столбец 7, если за ним не следует цифра от 1 до 9. Если следует цифра, предполагается, что текущая строка является строкой продолжения с цифрой, служащей символом продолжения. Последующие символы табуляции перемещаются в позиции строки кратные 8, начиная со столбца 16, (16, 24, 32...). Все символы табуляции расширяются в исходном листинге.

Пример: (показывает символ табуляции)

C2345678901234567890

**10 GOTO 20**

дает:

C2345678901234567890

**10 GOTO 20**

1. Допускается одновременная компиляция нескольких сегментов программы. Т.е. несколько подпрограмм, или основная программа и несколько подпрограмм могут быть скомпилированы при однократном выполнении компилятора.
2. SSS Fortran допускает пустые строки.
3. Символ @ в 1 столбце оператора обрабатывается как C (строка комментария), за исключением того, что переключается флаг листинга компилятора, (см. Раздел III.B Опции переключателя компилятора).
4. Если флаг LIST установлен, исходный листинг включен, сообщения об ошибках смешиваются с исходным листингом и отправляются на устройство печати. Символ включается в исходный листинг, чтобы указать приблизительную позицию ошибки. Если флаг LIST выключен, вывод исходного листинга отключен, вся информация отправляется на устройство консоли.
5. Во время компиляции распознаются следующие символы прерывания, поступающие с устройства консоли.

Ctrl-S Остановка выполнения; ожидание Ctrl-Q

Ctrl-Q Возобновление выполнение после Ctrl-S

Ctrl-C Прекращение выполнение, возврат в CP/M

1. Для достижения максимальной скорости компиляции, следует отметить следующее.
2. Перемещение головки диска минимизируется, если входной файл источника и выходные объектные файлы находятся на различных устройствах.
3. Листинг исходного файла занимает время. Средство исправления: используйте функцию LISTOFF, когда это возможно.
4. Использование флага OBJECT-OFF для проверки синтаксиса ускоряет компиляцию, так как исключается большая часть времени ожидания диска.
5. Любые подозрительные ошибки, которые могут возникнуть в результате использования компилятора, должны быть задокументированы со следующей информацией.
6. Выходной листинг компилятора с созданием полного листинга и карты распределения. В листинге должен присутствовать номер версии компилятора.
7. Листинг объектного модуля в шестнадцатеричном формате, созданный при компиляции (используя флаг ;H).
8. Любой выходной файл загрузчика, имеющий важное значение.
9. Любая другая существенная информация, которая может быть предоставлена пользователем.
10. В таблице символов распределения операторов перечисляются номера операторов и их смещения в соответствующей области. Для каждой ошибки ERROR 200 (неопределенная метка), будет отмечен неопределенный номер (номера) оператора с нулевым смещением.
11. В таблице символов распределения переменных представлена следующая информация:

*Флаги типа:*

**B** LOGICAL или INTEGER\*1

**I** INTEGER\*2

**R** REAL\*4

**D** REAL\*8 или DOUBLE PRECISION

**C** COMPLEX

**S** STRING

*Флаги атрибутов:*

**A** переменная - массив

**C** переменная в COMMON

**P** переменная является параметром

**E** переменная - EXTERNAL

**V** значение переменной использовалось

**D** переменной было присвоено значение

Длина программы и области данных также приводятся в шестнадцатеричном формате для каждого сегмента.

1. Если компилятор обнаруживает ошибки, файл $$$.SUB, если он присутствует, удаляется.
2. Компилятор поддерживает функцию вставки библиотеки. Эта функция позволяет вставку исходного текста компилятору из файлов библиотеки. Вставка инициируется # в столбце 1 компилируемого исходного файла. За символом # должно следовать допустимое в CP/M имя файла, например:

**# B: FILE.EXT**

Однако имя файла может содержать только буквы и цифры (не специальные символы). При вызове компилятор будет считывать из файла библиотеки до тех пор, пока не будет обнаружен конец файла. На этом этапе компилятор вернется к исходному файлу для ввода источника. Обратите внимание, что эта функция ограничена только одной только вставкой, т. е. файл библиотеки может не иметь никаких # в столбце 1.

В спецификации оператора вставки пробелы и символы табуляции игнорируются. Если устройство не задано, используется зарегистрированное устройство. Если расширение не задано, по умолчанию используется значение FOR. Если указанный файл не найден, или если предпринят более чем один уровень вставки, выводится код ошибки 186.

# 4. Загрузчик

## 4.1. Общее использование

LOADER - связывающий загрузчик SSS Fortran, системная утилита, объединяющая независимо скомпилированные или ассемблированные модули в исполнимый бинарный образ. Его функция заключается в том, чтобы переместить все адреса в каждом модуле и разрешить внешние ссылки, выполнив поиск необходимых модулей в определенных файлах.

Ввод для LOADER служат несколько файлов, которые представляют собой конкатенированный вывод ассемблера (создающий файлы REL с поддержкой TDL или Xitan) и компилятора SSS Fortran версии 2.0 или более поздней. LOADER выводит исполняемый образ загруженных модулей, а также карту расположения модулей и таблицу символов, если это указано.

## 4.2. Формат командной строки

LOADER выполняется в операционной системе CP/M с командной строкой формы:

**A>LOADER [outfile=] infile [,infile]... [/switch]... [,libfile]...**

где outfile, infile и libfile – выходной, входной и библиотечный файлы. Входных и библиотечных файлов может быть несколько. Если устройство не указано, подразумевается зарегистрированное устройство. По умолчанию расширение имеет значение REL для входных файлов и COM для выходного файла. Если спецификация выходного файла отсутствует, используется первое имя входного файла. Например, командная строка LOADER для загрузки TEST.REL и PROG.REL и вывода в TEST.COM, будет:

**A>LOADER TEST,PROG**

Если для входных файлов, указанных выше, выходной файл должен быть TESTY.COM, можно использовать следующую команду:

**A>LOADER TESTY=TEST,PROG**

Если файл не найден или часть команды неправильно введена, так что его нельзя интерпретировать, выдается сообщение об ошибке, а оставшаяся часть команды может быть введена повторно. Например:

**A>LOADER TEST,UDPATE,PROG**

**A:UDPATE.REL FILE NOT FOUND!**

**>UPDATE,PROG**

## 4.3. Формат вывода

Вывод из LOADER - это образ памяти (COM-файл), с подпрограммой инициализации, расположенной по адресу 100H. Блок содержит инициализацию указателя стека, область патча отладки, указатель на первый свободный байт и переход к глобальному символу .MAIN. Пользователь должен определить вход .MAIN. в любую основную программу. Процедура инициализации имеет следующий формат и глобальные символы.

**100H:**

**.SYSB.: LSPD 6 ;STACK INIT**

**.DBUG.: NOP ;DEBUG PATCH LOCATIONS**

**NOP**

**NOP**

**JMP .MAIN. ;START**

**.SYMB.: .WORD. DBGDT ;DEBUG IDENT LIST**

**.FREE.: .WORD. PRGEND ;FIRST FREE BYTE ADDRESS**

Первый модуль перемещается начиная с адреса 110H. Последующие модули загружаются последовательно по более высоким адресам. LOADER использует два счетчика перемещения, один для данных и один для кода. Вывод от загрузчика может также иметь стандартный формат контрольной суммы, если используется переключатель **/HEX**. Пользователь может также загрузить и выполнить программу, не выводя образ включением переключателя **/GO**.

Если переключатель **/MAP** присутствует в команде на устройство печати выводится карта загруженных модулей и их адреса перемещения. Аналогично, **/SYM** перечислит глобальные символы в порядке определения. Символы предваряются знаком ? если повторяющийся символ присутствовал во входном потоке. Если указано повторяющееся имя идентификатора, оно пропускается. Это полезно для загрузки обновлений перед балансом программы. Загружаются обновленные модули! и старые идентификаторы с одинаковыми именами пропускаются. При обнаружении дубликатов вводятся соответствующие предупреждения.

**? DUPLICATE IDENT name**

**? DUPLICATE SYMBOL name**

## 4.4. Библиотеки

После загрузки модулей пользователей осуществляется поиск любых модулей, которые содержат записи, на которые ссылаются загруженные модули в библиотеке системных подпрограмм LIBRARYS.REL (которая должна находиться на зарегистрированном устройстве). Если найдена необходимая запись, этот модуль загружается. Пользователь может создать свои собственные библиотеки (см. [Раздел 6](#_VI._Библиотеки)) и осуществлять поиск в них при помощи переключателя команды /LIB. Все файлы после /LIB загружаются в режиме поиска в библиотеке, то есть только ранее упомянутые модули берутся из библиотеки и включаются в программу.

Если существуют неопределенные символы после поиска LIBRARYS.REL, они печатаются, и пользователь может определить их в шестнадцатеричной нотации или он может загружать другие файлы, пытаясь определить символы. Например:

**UNDEFINED SYMBOLS**

**INPORT>A**

**LSQFIT>;**

**>LSQPAK**

INPORT определяется как 000AH, а затем командный режим повторно вводится, когда символ LSQFIT отображается как неопределенный, и затем загружается файл LSQPAK.

## 4.5. Ошибки загрузчика

LOADER - это однопроходный связывающий загрузчик, использующий все ядро для своей 4K-программы, таблицы символов и место для загружаемой программы. Сообщение INSUFFICIENT MEMORY не требует пояснений. Символ LODFRE, который отображается в любой созданной карте идентификаторов, является индикатором неиспользуемого объема памяти во время процесса загрузки.

Хотя это маловероятно, сообщение INVALID INPUT выводится, когда в некоторый входной модуль имеет недопустимый формат, символ, содержащий недопустимые символы и т.д.

Ошибка может возникнуть, когда LOADER переводит образ программы в соответствующие места для двоичного вывода или при загрузке и выполнении. В этом случае будет введено соответствующее сообщение, и пользователь может воссоздать ядро образа, выведя шестнадцатеричный файл (с помощью переключателя /HEX) и запустив программу LOAD.COM. Например:

**LOADER TEST**

**? OVERWRITE USE /HEX! ?**

**LOADER TEST/H**

**LOAD TEST**

Ошибка OVERWRITE скорее всего возникает, когда абсолютное модули, расположенные в высоких адресах загружаются перед перемещаемыми модулями. Проблема устраняется путем загрузки абсолютных модулей в порядке их расположения после загрузки перемещаемых модулей. Обратите внимание, что с системой SSS Fortran предоставляется специальная программа LOAD. Эта программа LOAD должна использоваться вместо утилиты CP/M LOAD, когда вы, используя LOADER с переключателем /H, генерируете .HEX файлы.

## 4.6. Опции переключателей

Переключатели LOADER могут быть усечены до одной буквы, после чего должен следовать разделитель. Переключатели:

**/GO** или **/G** Загрузить и выполнить без создания выхода

**/HEX** или **/H** вывод файл в формате контрольной суммы с шестнадцатеричным расширением

**/LIB** или **/L** Поиск всех последующих файлов осуществляется в режиме поиска в библиотеке

**/MAP** или **/M** Вывод модуля или карты идентификаторов

**/SYM** или **/S** Вывод карты глобальных символов

## 4.7. Примечания

1. LOADER реагирует на Ctrl-S, Ctrl-Q и Ctrl-C аналогично компилятору.
2. Если происходит ошибка загрузки, то файл $$$.SUB удаляется, если он существует.

# 5. Пакеты ввода-вывода

## 5.1. Общее использование

Пакет ввода-вывода Fortran поддерживает все сопряженные с CP/M устройства и обеспечивает необходимые связи для поддержки любого сопряженного пользовательского устройства. Пакет ввода-вывода использует логические устройства с 1 по 4 для ссылки на консоль, перфоратор, считыватель и устройства печати и устройства с 5 по 20 для обращения к файлам на диске. В то время как консоль, перфоратор, считыватели и устройство печати всегда открыты и доступны по запросу, дисковые устройства должны быть явно открыты и закрыты. Доступ к файлам на диске может осуществляться последовательно, в поточном или блочном режиме, или возможен случайный доступ в поточном режиме.

Файл потокового режима представляет собой последовательный файл без добавления информации о записи. Файлы блочного режима содержат в начале каждой записи информацию о длине записи так, что они похожи на устройства на магнитной ленте. Файлы произвольного доступа - файлы потокового режима, с которыми пользователь управляет адресом чтения или записи с помощью номера записи. Существуют различные элементы в библиотеке для каждого типа доступа, поэтому загружаются только необходимые подпрограммы.

## 5.2. Вызываемые подпрограммы

Все подпрограммы доступа к файлу - функции Fortran, возвращающие .TRUE. при ошибках. Ниже приводятся пояснения по вводу-выводу на диск.

|  |  |  |
| --- | --- | --- |
|  |  | Определения |
| UNIT | = | Целое число между 5 и 20 |
| MODE | = | 0 Режим двоичного потока  1 двоисный блочный режим  2 Режим потока ASCII |
| DEV | = | Целое число диска или 0, чтобы выбрать зарегистрированное устройство. Оно может быть переопределено спецификацией файла. |
| FILE | = | Символьная переменная или литерал, (строка) формы D:FILENAME.EXT как в стандартных описаниях файлов CP/M, за исключением того, что допустимы только числа и буквы. |

Обратите внимание, что файлы ASCII представляют собой потоковые файлы, состоящие из символов ASCII, заканчивающихся буквой Z (Ctrl- Z), в качестве знака конца файла.

**LOGICAL FUNCTION IOREAD (UNIT, MODE, DEV, FILE)**

*Открывает* указанный файл на обозначенном логическом устройстве. Возвращает .TRUE., если такой файл не существует или устройство уже открыто.

**LOGICAL FUNCTION IOWRIT (UNIT, MODE, DEV, FILE)**

*Удаляет*, затем *создает* указанный файл на обозначенном устройстве. Возвращает .TRUE., если отсутствует место в каталоге или файл уже открыт.

**LOGICAL FUNCTION IORAND (BLKSIZ, RECSIZ, UNIT, DEV, FILE)**

Инициализирует устройство для ввода-вывода произвольного доступа к указанному файлу. BLKSIZ - размер блока в байтах, который содержит BLKFAC записей каждая размером RECSIZ. Ввод-вывод оптимизирован, если BLKSIZ равняется 128, размеру дискового блока или кратному этому числу. Каждая операция ввода-вывода вычисляет адрес записи:

**BLKFAC = BLKSIZ/RECSIZ**

**BLOCK = (RECORD-1)/BLKFAC**

**REC = IMOD(RECORD-1,BLKFAC)**

**ADR = BLOCK\*BLKSIZ + REC\*RECSIZ**

Ввод-вывод продолжается от расчетного ADR в файле в потоковом режиме. Возвращает .TRUE., если состояние ошибки. Обратите внимание на то, что RECORD=1 - первая запись в файле.

**LOGICAL FUNCTION IOAPND (UNIT, MODE, DEV, FILE)**

Создает файл, если он не существует или добавляет к следующему свободному блоку, если он уже существует. Последний блок файла обнуляется IOCLOS, что приведет к обнулению, если файл считывается в потоковом режиме. Возвращает .TRUE., если отсутствует место в каталоге или устройство уже открыто.

**LOGICAL FUNCTION IOCLOS(UNIT)**

Сбрасывает соответствие устройство-файл. Если файл является последовательным, IOCLOS обнуляет остаток дискового блока и обновляет каталог. Если файл является устройством произвольного доступа, IOCLOS обновляет все блоки управления файлом. Возвращает .TRUE., если устройство не открыто.

**LOGICAL FUNCTION IOREW (UNIT)**

Сбрасывает устройство для чтения из, или записи в начало файла. Эквивалентен IOCLOS, затем IOWRIT или IOREAD. Возвращает .TRUE. если устройство не открыто.

**LOGICAL FUNCTION IODEL (DEV, FILE)**

Удаляет указанный файл или возвращает .TRUE., если файл не существует.

**LOGICAL FUNCTION IOREN (DEV, FILE1, FILE2)**

Переименовывает FILE1 в FILE2. Возвращает .TRUE., если FILE1 не существует, или FILE2 уже существует.

**LOGICAL FUNCTION IOLOOK (DEV, FILE)**

Возвращает .TRUE., если файл существует.

## 5.3. Интерфейс пользователя

Пользователь может написать контроллеры устройств и взаимодействовать с ними следующим образом. Предоставленная пользователем подпрограмма USRINZ, вызывается для инициализации устройства пользователя перед операцией ввода-вывода. Следующие расположения содержат параметры чтения/записи.

**IORDWT** не нулевой байт если запись;

**IOUNIT** байт с номером устройства;

**IOFFLG** ненулевой байт если форматный ввод-вывод;

**IORFLG** ненулевой байт если произвольный ввод-вывод;

**IORECD** номер записи или 0.

Пользователь должен поместить адрес своей сервисной программы своего устройства для указанного устройства в IOXADR и адрес процедуры завершения передачи своего устройства в IOEADR. Все операции ввода-вывода используют IOBYTE для чтения/записи байта. Сервисные процедуры пользователя должны вводить или выводить байты в IOBTYE при каждом вызове. В конце файла служебная программа должна выполнить ПЕРЕХОД на IOENDX. При ошибках устройства сервисные процедуры должнывыполнить ПЕРЕХОД на IOERRX с соответствующим кодом ошибки в регистре C.

Процедура USRINZ сохраняет адрес процедуры завершения передачи устройств в IOEADR, чтобы можно было выполнить любое управление очисткой или устройством для прекращения передачи. Однако этого не происходит, если подпрограммы пользователей перешли к IOENDX или IOERRX. В [Приложении 9](#_Пиложение_9._Добавление) приведен пример добавления пользовательского устройства.

Операторы управления устройством Fortran в настоящее время не реализуются пакетом системного ввода-вывода, а зарезервированы для ленточных устройств. Эти инструкции (REWIND, BACKSPACE, ENDFILE и PAUSE) генерируют вызовы REWIND, BACKSPACE, ENDFILE и PAUSE с номером устройства в регистровой паре HL. В настоящее время пользователь может использовать эти вызовы для надлежащего управления своими пользовательскими сопряженными устройствами, но такое использование не рекомендуется.

## 5.4. Примечания

1. Элементам типа CHARACTER, выводимым на устройства больше 4 и меньше 51, предшествует длина строки (только для режима двоичного потока). При вводе данных с этих устройств предполагается, что строки будут содержать их длину. Строки, вводимые с устройств менее 5 или более 50, заканчиваются возвратом каретки (ASCII 13).
2. Пакет ввода-вывода распределяет буферы динамически. Если память превышена, выводится соответствующее сообщение. Примерно 160 байт используются для каждого последовательного файла и 420 байтов для каждого файла произвольного доступа.
3. Ошибки устройства или форматера, которые не попадают в ловушку, печатаются с сообщением:

**RUNTIME ERROR = XX ON LAST UNIT = YY**

Список ошибок времени выполнения содержится в [Приложении 1](#_Приложение_1._Сообщения). Список назначений устройств ввода/вывода, приведен в [Приложении 6](#_Приложение_6._Назначение_2).

## 5.5. Примеры программ

Следующие программы - примеры последовательного и бесформатного дискового ввода-вывода произвольного доступа.

**C EXAMPLE OF AN UNFORMATTED RANDOM ACCESS ROUTINE**

**CHARACTER\*8 RANFIL**

**DATA RANFIL /'TEST.DAT'/**

**C OPEN THE FILE**

**IF (IORAND(2,2,5,0,RANFIL)) GOTO 300**

**C WRITE EVERY 2\*\*N RECORD**

**DO 10 I=0,12**

**10 WRITE(5/ISHIFT(1,I)) I**

**IF (IOCLOS(5)) GOTO 300**

**C TEST IF DATA WRITTEN CORRECTLY**

**IF (IORAND(2,2,5,0,RANFIL)) GOTO 300**

**DO 20 I=0,12**

**READ(5/ISHIFT(1,I)) J**

**20 IF (J .NE. I) GOTO 300**

**WRITE(1) 'ALL OK'**

**STOP**

**300 WRITE(1) '?ERROR?'**

**STOP**

**END**

**C EXAMPLE OF SEQUENTIAL I/0 ROUTINE**

**CHARACTER\*15 FILE**

**REAL\*8 BUF(16)**

**10 WRITE(1) FILE = 'TEST.TXT'**

**READ(1) FILE**

**IF (IOREAD(5,0,0,FILE)) GOTO 40**

**IF (IOWRIT(6,0,0,XXX.XXX)) GOTO 50**

**20 READ(5,ENDFILE=30) BUF**

**WRITE(6) BUF**

**GOTO 20**

**30 IF (IOCLOS(6) .OR. IODEL(0,FILE)) GOTO 60**

**IF (IOREN(0, XXX.XXX,FILE)) GOTO 60**

**WRITE(1) DONE**

**STOP**

**40 WRITE(1) 'NO SUCH FILE'**

**GOTO 10**

**50 WRITE(1) 'CANNOT CREATE'**

**STOP**

**60 WRITE(1) 'RENAME ERROR'**

**STOP**

**END**

# 6. Библиотеки

## 6.1. Общее использование

Пакет SSS Fortran поставляется с системной библиотекой Fortran (LIBRARYS.REL) и математической библиотекой Fortran (MLIB.REL). LIBRARYS.REL состоит из следующих модулей.

1. Библиотека форматеров Fortran
2. Библиотека пакетов ввода-вывода Fortran
3. Библиотека обработки строк Fortran
4. Библиотека динамической размещения Fortran
5. Различные подпрограммы, используемые системой Fortran, и процедуры связывания языка ассемблер.

**MLIB.REL** состоит из следующих модулей.

1. Внутренние функция Fortran и библиотека встроенных функций
2. Математическая библиотека одинарной точности
3. Математическая библиотека двойной точности
4. Математическая библиотека комплексных вычислений.

LOADER всегда осуществляет поиск в **LIBRARYS.REL**. Однако, если программа требует какой-либо части MLIB.REL, то для поиска в MLIB.REL необходимо использовать опцию LOADER /LIB (см. [Раздел 4](#_IV._ЗАГРУЗЧИК)).

В следующих разделах этой главы будет рассмотрена соответствующая информация, касающаяся системной библиотеки SSS Fortran, которая ранее не была подробно описана в данном руководстве.

Разумеется, пользователь может создать собственные библиотеки, объединив вместе несколько созданных объектных модулей. Этот процесс может быть выполнен с помощью команды CP/M PIP. Следует отметить, что порядок модулей в библиотеке имеет важное значение. Модули должны быть помещены в последовательности таким образом, чтобы модули, которые должны быть включены из-за включения другого модуля, появлялись в библиотеке после того, как первый модуль будет включен. Например, если пользователь имеет два модуля A и B, которые должны быть помещены в библиотеку, если модуль A должен вызываться непосредственно программой для загрузки или ссылки на ранее загруженную программу, и если модуль B должен быть включен из-за наличия модуля A, то модуль A должен появиться перед модулем B в библиотеке. Рассмотрим следующую последовательность команд.

**A>FOR A**

**A>FOR B**

**A>PIP LIB1.REL=A.REL,B.REL**

**A>LOADER TEST/L,LIB1**

Файл A.FOR компилируется для получения A.REL и файл B.FOR компилируется для получения B.REL. Затем эти два модуля объединяются, используя утилиту PIP, чтобы получить один модуль под названием LIB1.REL. Обратите внимание, что порядок в библиотеке модулей A, затем B. В последней команде загружается объектный модуль TEST.REL и затем выполняется поиск в библиотеке LIB1.REL. То есть модули A или B будут загружены только если на них ссылается TEST. Обратите внимание, что поиск в системной библиотеке LIBRARYS.REL выполняется после того, как все первоначально указанные загрузки завершены. В этом случае после загрузки TEST и поиска в LIB1.

Обратите внимание, что модуль, проверяемый в данный момент загрузчиком, не может вызвать ранее отсканированный модуль. Если это ситуация происходит, останутся неопределенные символы будут, и будет необходим второй проход через ту же библиотеку загрузчиком. Необходимо проявлять осторожность при определении порядка модулей в библиотеке в сложных ситуациях, связанных с ссылками.

Пользователь не должен заботиться о деталях поиска и упорядочения модулей в библиотеке, если необходимо работать только с предоставленными библиотечными процедурами Fortran. Загрузчик автоматически осуществляет поиск в системной библиотеке Fortran.

## 6.2. Библиотека внутренних и встроенных функций

Ниже представлен список процедур, обеспеченных в математической библиотеке SSS Fortran (**MLIB.REL**), которые рассматривают как внутренние или встроенные функции. Подробности работы этих процедур можно найти в любом справочнике по стандартному Fortran.

| **Имя** | **К-во аргументов** | **Тип аргумента** | **Возвращаемый тип** |
| --- | --- | --- | --- |
| INT | 1 | REAL | INTEGER |
| AMOD | 2 | REAL | REAL |
| ABS | 1 | REAL | REAL |
| IABS | 1 | INTEGER | INTEGER |
| DABS | 1 | DOUBLE | DOUBLE |
| FLOAT | 1 | INTEGER | REAL |
| IFIX | 1 | REAL | INTEGER |
| SIGN | 2 | REAL | REAL |
| ISIGN | 1 | INTEGER | INTEGER |
| MOD | 2 | INTEGER | INTEGER |
| AIMAG | 1 | COMPLEX | REAL |
| DBLE | 1 | REAL | DOUBLE |
| DSIGN | 2 | DOUBLE | DOUBLE |
| DIM | 2 | REAL | REAL |
| IDIM | 2 | INTEGER | INTEGER |
| SNGL | 1 | DOUBLE | REAL |
| REAL | 1 | COMPLEX | REAL |
| CMPLX | 2 | REAL | COMPLEX |
| CONJG | 1 | COMPLEX | COMPLEX |
| CABS | 1 | COMPLEX | COMPLEX |
| DMOD | 2 | DOUBLE | DOUBLE |
| IDINT | 1 | DOUBLE | INTEGER |
| AINT | 1 | REAL | REAL |
| DINT | 1 | DOUBLE | DOUBLE |
| DMAX1 | \* | DOUBLE | DOUBLE |
| DMIN1 | \* | DOUBLE | DOUBLE |
| AMAX0 | \* | REAL | INTEGER |
| MAX0 | \* | INTEGER | INTEGER |
| AMIN0 | \* | REAL | INTEGER |
| MIN0 | \* | INTEGER | INTEGER |
| AMAX1 | \* | REAL | REAL |
| MAXI | \* | INTEGER | REAL |
| AMIN1 | \* | REAL | REAL |
| MIN1 | \* | INTEGER | REAL |
| INT1 | 1 | INTEGER\*2 | INTEGER\*1 |
| MAX2 | \* | INTEGER\*1 | INTEGER\*2 |
| MIN2 | \* | INTEGER\*1 | INTEGER\*2 |
| AMAX2 | \* | INTEGER\*1 | REAL |
| AMIN2 | \* | INTEGER\*1 | REAL |

*Примечание 1*: INTEGER означает 2-байтовое целое число, REAL означает 4-байтовое число плавающей точкой, DOUBLE означает 8-байтовое число с плавающей точкой.

*Примечание 2*: Функции, которые возвращают результаты DOUBLE PRECISION или COMPLEX, должны быть объявлены с соответствующим типом в вызывающей программе. Например:

**DOUBLE PRECISION DSIGN**

**REAL\*8 DBLE,Q**

**A = DSIGN(DBLE(B),Q)**

*Примечание 3:* \* в столбце количество аргументов выше указывает, что количество аргументов должно быть по крайней мере 1, но не больше, чем 85.

*Примечание 4*: DINT является эквивалентном AINT выше с двойной точностью.

*Примечание 5*: INT1, MAX2, MIN2, AMAX2 и AMIN2 являются INTEGER\*1 эквивалентами IFIX, MAX1, MIN1, AMAX1 и AMIN1 соответственно.

## 6.3. Библиотека служебных подпрограмм

Следующие подпрограммы предоставляют пользователю возможность читать из и записать непосредственно в порты ввода-вывода Z80, и также непосредственно изменить или читать ячейки памяти.

**INTEGER\*1 FUNCTION INPT(PORT)**

Функция, которая вводит текущее 1-байтовое значение указанного порта.

**SUBROUTINE OUTPT(PORT,VALUE)**

Подпрограмма, которая выводит 1 байт в указанный порт.

**INTEGER\*2 FUNCTION IPEEK(ADDR)**

Функция, которая возвращает значение слова памяти по адресу **ADDR**.

**SUBROUTINE POKE1(ADDR,VALUE)**

Подпрограмма, которая вставляет (вводит 1 байт по абсолютному адресу) в память по адресу **ADDR**.

**SUBROUTINE POKE2(ADDR,VALUE)**

То же, что и **POKE1**, за исключением двух байтов, вставленных в память.

## 6.4. Математическая библиотека SSS Fortran

Ниже приводится краткое описание математических функций одинарной, двойной точности и комплексных, доступные в SSS Fortran. Обратите внимание, что функции, возвращающие результаты DOUBLE PRECISION или COMPLEX, должны быть объявлены с тем же типом, как в программе в которой они вызываются.

ATAN(A) REAL\*4 FUNCTION REAL\*4 ARGUMENT

DEFINITION: ARCTAN A

ERROR CODE: Нет

INPUT: Любое число REAL\*4

OUTPUT: -PI/2 <= ATAN(A) <= PI/2

DATAN(DA) REAL\*8 FUNCTION REAL\*8 ARGUMENT

DEFINITION: ARCTAN DA

ERROR CODE: Нет

INPUT: ANY REAL\*8 NUMBER

OUTPUT: -PI/2 <= DATAN(DA) <= PI/2

ATAN2(Y,X) REAL\*4 FUNCTION REAL\*4 ARGUMENT

DEFINITION: ARCTAN(Y,X)

ERROR CODE: 23 BOTH ARGUMENTS = 0

INPUT: ANY PAIR OF REAL\*4 NUMBERS

OUTPUT: -PI <= ATAN2 <= PI

DATAN2(DY,DX) REAL\*8 FUNCTION REAL\*8 ARGUMENT

DEFINITION: ARCTAN(DY,DX)

ERROR CODE: 23 BOTH ARGUMENTS = 0

INPUT: ANY PAIR OF REAL\*8 NUMBERS

OUTPUT: -PI <= DATAN2 <= PI

ALOG(A) REAL\*4 FUNCTION REAL@@ ARGUMENT

DEFINITION: LOG BASE e OF A

ERROR CODE: 24 INPUT < 0

INPUT: ANY REAL\*4 NUMBER > 0.0

OUTPUT: APPROX -172. TO +172.

DLOG(DA) REAL\*8 FUNCTION REAL\*8 ARGUMENT

DEFINITION: LOG BASE e OF DA

ERROR CODE: 24 INPUT < 0

INPUT: ANY REAL\*8 NUMBER > 0.0

OUTPUT: APPROX -172. TO +172.

ALOGIO(A) REAL\*4 FUNCTION REAL\*4 ARGUMENT

DEFINITION: LOG BASE 10 OF A

ERROR CODE: 24 INPUT < 0

INPUT: ANY REAL\*4 NUMBER > 0.0

OUTPUT: APPROX -79. TO +77.

DLOG10(DA) REAL\*8 FUNCTION REAL\*8 ARGUMENT

DEFINITION: LOG BASE 10 OF DA

ERROR CODE: 24 INPUT < 0

INPUT: ANY REAL\*8 NUMBER > 0.0

OUTPUT: APPROX -79. TO +77.

TANH(A) REAL\*4 FUNCTION REAL\*4 ARGUMENT

DEFINITION: TANH A

ERROR CODE: 21 INPUT RANGE EXCEEDED

INPUT: -171. < A < 171.

OUTPUT: -1. < TANH < 1.

DTANH(DA) REAL\*8 FUNCTION REAL\*8 ARGUMENT

DEFINITION: TANH DA

ERROR CODE: 21 INPUT RANGE EXCEEDED

INPUT: -171. < DA < 171.

OUTPUT: -1. < DTANH < 1.

SIN(A) REAL\*4 FUNCTION REAL\*4 ARGUMENT

DEFINITION: SIN A, A IN RADIANS

ERROR CODE: 22 INPUT RANGE EXCEEDED

INPUT: -102937. < A < 102937

OUTPUT: -1. <= SIN <= 1.

DSIN(DA) REAL\*8 FUNCTION REAL\*8 ARGUMENT

DEFINITION: SIN DA, DA IN RADIANS

ERROR CODE: 22 INPUT RANGE EXCEEDED

INPUT: -102937. < DA < 102937.

OUTPUT: -1. <= DSIN <= 1.

COS(A) REAL\*4 FUNCTION REAL\*4 ARGUMENT

DEFINITION: COS A, A IN RADIANS

ERROR CODE: 22 INPUT RANGE EXCEEDED

INPUT: -102935. <= A <= 102935.

OUTPUT: -1. <= COS <= 1.

DCOS(DA) REAL\*8 FUNCTION REAL\*8 ARGUMENT

DEFINITION: COS DA, DA IN RADIANS

ERROR CODE: 22 INPUT RANGE EXCEEDED

INPUT: -102935. <= DA <= 102935.

OUTPUT: -1. <= DCOS <= 1.

EXP(A) REAL\*4 FUNCTION REAL\*4 ARGUMENT

DEFINITION: E\*\*A

ERROR CODE: 21 INPUT RANGE EXCEEDED

INPUT: -171. <= A <= 171.

OUTPUT: E\*\*-171. <= EXP <= E\*\*171.

DEXP(DA) REAL\*8 FUNCTION REAL\*8 ARGUMENT

DEFINITION: E\*\*DA

ERROR CODE: 21 INPUT RANGE EXCEEDED

INPUT: -171. <= DA <@ 171.

OUTPUT: E\*\*-171. <= DEXP <= E\*\*171.

SQRT(A) REAL\*4 FUNCTION REAL\*4 ARGUMENT

DEFINITION: SQUARE ROOT OF A

ERROR CODE: 25 INPUT NEGATIVE

INPUT: ANY REAL\*4 NUMBER >= 0.0

OUTPUT: 0 TO APPROX 1.E38

DSQRT(DA) REAL\*8 FUNCTION REAL\*8 ARGUMENT

DEFINITION: SQUARE ROOT OF DA

ERROR CODE: 25 INPUT NEGATIVE

INPUT: ANY REAL\*8 NUMBER >= 0.0

OUTPUT: 0 TO APPROX 1.E38

*Примечание 1*: Для вышеуказанных функций при ошибке возвращается ноль.

*Примечание 2*: INPUT и OUTPUT относятся к допустимым диапазонам.

CSIN(A) COMPLEX FUNCTION COMPLEX ARGUMENT

DEFINITION: SINE CA, CA IN RADIANS

ERROR CODES: 21 IMAGINARY INPUT RANGE EXCEEDED

22 REAL INPUT RANGE EXCEEDED

INPUT: REAL PART -102935. TO +102935.

IMAGINARY PART -171. TO +171.

OUTPUT: REAL PART -(E\*\*171./2.) TO (E\*\*171./2.)

IMAGINARY PART (E\*\*171./2.) TO (E\*\*171./2.)

CCOS(CA) COMPLEX FUNCTION COMPLEX ARGUMENT

DEFINITION: COS CA, CA IN RADIANS

ERROR CODES: 21 IMAGINARY INPUT RANGE EXCEEDED

22 REAL INPUT RANGE EXCEEDED

INPUT: REAL PART -102935. TO +102935.

IMAGINARY PART -171. TO +171.

OUTPUT: REAL PART -(E\*\*171./2.) TO -(E\*\*171./2.)

IMAGINARY PART (E\*\*171./2.) TO (E\*\*171./2.)

CEXP(CA) COMPLEX FUNCTION COMPLEX ARGUMENT

DEFINITION: E\*\*CA

ERROR CODES: 21 REAL INPUT RANGE EXCEEDED

22 IMAGINARY INPUT RANGE EXCEEDED

INPUT: REAL PART -171. TO +171.

IMAGINARY PART -102935. TO +102935.

OUTPUT: REAL PART -E\*\*171. TO E\*\*171.

IMAGINARY PART -E\*\*171. TO E\*\*171.

CLOG(CA) COMPLEX FUNCTION COMPLEX ARGUMENT

DEFINITION: LOG BASE e CA

ERROR CODES: 23, 24 REAL AND IMAGINARY PARTS BOTH= 0

INPUT: ANY COMPLEX NUMBER

OUTPUT: REAL PART APPROX -172. TO +172.

IMAGINARY PART APPROX -PI TO PI

CSQRT(CA) COMPLEX FUNCTION COMPLEX ARGUMENT

DEFINITION: SQUARE ROOT OF CA

ERROR CODES: NONE

INPUT: ANY COMPLEX NUMBER

OUTPUT: ANY COMPLEX NUMBER

*Примечание*. Для всех функций COMPLEX условия ошибки могут давать неопределенные результаты.

## 6.5. Библиотеки работы со строками и динамическим распределением

Подпрограммы в этой библиотеке могут быть сгруппированы как: специальная арифметика, динамическое выделение и строки.

Специальные арифметические подпрограммы обеспечивают быстрые сдвиги аргументов INTEGER, REAL и DOUBLE PRECISION. Они также включают подпрограммы, возврашающие INTEGER двоичный логарифм аргумента и установки двоичной экспоненты. Эти подпрограммы используются для сокращения диапазона с помощью процедур в математической библиотеке.

Процедуры динамического распределения были включены для смягчения одного из основных недостатков Fortran, а именно отсутствия указателей для размещения связанных списков. ALLOC ...

С этого места в источнике отсутствует весь последующий текст, включая Приложения.

### 6.5.1. Арифметические действия

### 6.5.2. Динамическое распределение памяти

### 6.5.3. Строки

# Приложения

## Приложение 1. Сообщения и коды ошибок

### 1 Коды ошибок компилятора

### 2 Коды ошибок форматирования

### 3. Коды ошибок математических подпрогамм

### 4. Другие коды ошибок времени выполнения

## Приложение 2. Блок-схема обработки

## Приложение 3. Пример обработки

## Приложение 4. Форматы вызова и интерфейс ассемблера

## Приложение 5. Формат чисел

## Приложение 6. Назначение устройств ввода-вывода

## Приложение 7. Полезная информация о средстве форматирования

## Приложение 8. Системные имена

## Пиложение 9. Добавление устройств пользователя

## Приложение 10. Примеры строковых подпрограмм

## Приложение 11. Расширенная грамматика SSS Fortran

**Документация PATCHREL**

**PATCHREL** - это служебная программа, которая работает с файлами REL и имеет две цели:

1. Для сжатия библиотек, которые были сформированы путем объединения или копирования файлов REL вместе в библиотеку. (нежелательные нули удаляются.)
2. Обновить библиотеку путем замены выбранных модулей.

Выполнить:

**A>PATCHREL**

(Обратите внимание, что в командной строке не указаны файлы.)

PATCHREL запросит три файла:

**INPUT FILE, OUTPUT FILE, PATCH FILE**

каждый файл должен быть указан как file.ext, расширение по умолчанию не предполагается.

По сути, **PATCHREL** читает входной файл и выводит его в сжатом виде в выходной файл. Если файл исправления не пуст, любые модули во входном файле, соответствующие файлам в файле исправления, заменяются версией исправления.

Чтобы сжать файл библиотеки без перестановки, вы должны создать нулевой файл исправления и указать его как файл исправления. (Это можно сделать с помощью редактора: открыть файл и выйти без вставки).

Для исправления библиотеки вам понадобится файл исправления. Обновления S.S.S. FORTRAN IV в большинстве случаев поставляется в виде исправлений, поэтому сохраните этот файл.

**Документация LISTREL.COM**

**A> LISTREL; ;ВЫПОЛНИТЬ ПРОГРАММУ**

Когда она запрашивает имя входного файла, введите имя файла ".rel", который вы хотите указать. (Программа не использует по умолчанию расширение .REL)

**LISTREL** напечатает на строчном принтере имена модулей в файле .REL. В первой строке будет имя модуля. Вторая строка модуля - это точки входа. (Подпрограммы FORTRAN имеют идентичные имена точек входа и модулей, поэтому отдельные имена не указаны.)

**Формат перемещаемого объектного модуля Technical Design Labs**

Нейл Колвин {Neil Colvin}

Technical Design Labs Research Park Bldg H 1101 State Rd Princeton NJ 08540

**Определения**

**Объектный модуль**: вывод из языкового процессора. Объектные модули могут быть загружены в память для выполнения в фиксированных адресах.

**Перемещаемый объектный модуль**: объектный модуль, содержащий информацию, которая позволяет загрузчику поместить его в любом месте адресного пространства памяти.

**Внутренний символ**: символ, расположение которого доступно другим модулям помимо того, в котором он определен.

**Внешний символ**: символ, который используется в модуле, но определен как внутренний символ в некотором другом модуле.

**Точка входа**: внутренний символ в модуле, используемый для выбора модуля для загрузки в результате ссылки на него в другом модуле в качестве внешнего символа.

**Связываемый объектный модуль**: объектный модуль, содержащий информацию, идентифицирующую внешние, внутренние символы и точки входа, которые могут быть "соединены" загрузчиком с другими подобными модулями.

**Основание перемещения**: Внешний символ, адрес которого - основание для перемещения объектного модуля. Внешний символ может представлять программу, данные или общую зону памяти.

**Определение формата объектного модуля**

Формат объектного модуля является расширением формата Intel "шестнадцатеричный файл", но не совместимый с ним. Модуль состоит из файла последовательных символов ASCII, представляющих двоичные данные, символы и управляющую информацию, необходимые для создания заключительной программы из модуля. Все двоичные байты в этой структуре представлены как два символа ASCII, соответствующие шестнадцатеричному значению байта (например: 11001001 -> C9). Все значения ASCII представлены соответствующим символом ASCII (например: A -> A).

Каждая из разных записей в модуле обозначена при помощи символа подсказки как первого символа записи (в формате Intel, это ":"). Допустимые символы подсказки:

**Символ Значение**

**!** Идентификационная запись модуля

**@** Запись точки входа

**#** Запись внутреннего символа

**\** Запись внешнего символа и основания перемещения

**&** Запись таблицы символов

**;** Запись данных или программы или конца файла

Каждая запись в модуле завершается однобайтовой двоичной контрольной суммой всех предыдущих байтов в записи за исключением символа подсказки. Контрольная сумма - поразрядное дополнение до двух суммы предыдущих байтов. Оба выходных формата (два символа двоичных или один символ ASCII) по-прежнему рассчитывают только как один байт в контрольной сумме (т.е.: перед преобразованием для вывода).

Кроме того, каждой записи предшествуют последовательность возврат каретки и перевода строки, чтобы упростить листинг модуля на внешнем устройстве.

**Идентификационная запись модуля ("!")**

Номер байта Описание

1-2 CR/LF

3 Подсказка восклицательный знак (!).

4-9 ASCII имя модуля.

10-11 Контрольная сумма.

**Запись точки входа ("@")**

Номер байта Описание

1-2 CR/LF

3 Подсказка знак коммерческого at (@).

4-5 Количество точек входа в этой записи.

6-?? ASCII имена точек входа, шесть байтов на имя. Имена выровнены по левому краю и заполнены пробелами.

?? Контрольная сумма.

**Запись внутреннего символа ("#')**

Номер байта Описание

1-2 CR/LF

3 Подсказка знак решетки (#)

4-5 Количество внутренних символов в этой записи.

6-11 ASCII имена внутренних символов, выровненных по левому краю и заполненых пробелами.

12-13 Основание перемещения для символа. Значение этого символа относительно определенного основания перемещения.

14-17 Значение символа (16 бит).

\*\*\*\* Вышеупомянутые три поля повторяются для каждого внутреннего символа в записи.

?? Контрольная сумма.

**Запись внешнего символа и основания перемещения ("\")**

Номер байта Описание

1-2 CR/LF

3 Приглашение знак наклонной черты влево (\).

4-5 Количество внешних или перемещаемых символов в этой записи.

6-11 ASCII имя символа, выровненное по левому краю и заполненого пробелами.

12-13 Число перемещения, присвоенный этому символу в этом модуле. Это число уникально для каждого символа. Оно начинается с единицы и последовательно увеличивается для каждого последующего внешнего или символа основания перемещения.

14-17 Размер перемещаемого сегмента или флаг внешней ссылки. Если это значение - нуль, оно представляет ссылку на символ, определенный внешним по отношению к этому модулю (обычно подпрограмма или глобальный элемент данных). Если но ненулевое, то значение - размер перемещаемого сегмента, определеного в этом объектном модуле. Этот сегмент может содержать или код или данные, и может загрузчиком быть расположен где угодно в памяти, независимо от любого другого сегмента.

\*\*\*\* Вышеупомянутые три поля повторяются для каждого символа, содержавшегося в этой записи.

?? Контрольная сумма.

**Symbol Table Record ("&")**

Номер байта Описание

1-2 CR/LF

3 Подсказка знак амперсанд (&).

4-?? Остаток от этой записи идентичен записи внутреннего символа. Все символы, определенные в этом модуле, содержатся в этих записях.

**Data/Program Record (";")**

Номер байта Описание

1-2 CR/LF

3 Приглашение символ точка с запятой (;).

4-5 Число байтов двоичных данных в этой записи. Максимум - 32 двоичных байта (64 байта в представлении ASCII). Если это значение - ноль, эта запись - запись конца файла, описанной ниже.

6-9 Адрес загрузки данных относительно указанной основы перемещения.

10-11 Основа перемещения для всего перемещения в этой записи. Все перемещаемые значения в этой записи добавляются к текущему значению указанной основы перемещения до того, как будут помещены в память.

12-13 Управляющий байт перемещения. Этот байт управляет перемещением следующих восьми байтов в записи (если многие остаются в соответствии с полем Count). Биты используются слева направо. Биты имеют следующие значения:

0: один абсолютный байт подразумевает немодифицированную загрузку.

10: двухбайтовое перемещаемое значение, младший значащий байт подразумевает сначала, добавляют 16 разрядных значений к текущей основе перемещения и результат загружается сначала младший значащий байт.

110: трехбайтовая ссылка на другую основу перемещения. Первый байт - основание системы счисления перемещения, и два последуюшие - 16 разрядное значение, сначала младший значащий байт. Это подразумевает, добавляет указанную основу перемещения к 16 разрядному значению и загрузить первым наименее значимый байт результата.

Обратите внимание, что комбинация из двух или трех байтов никогда не прерывается на границе записи.

14-29 Байт данных контролируется, как указано выше.

30-?? Вышеуказанные сочетания байтов управления и данных повторяются, как указано счетчиком.

?? Контрольная сумма.

**Запись конца файла (";")**

Номер байта Описание

1-2 CR/LF

3 Точка с запятой (;) подсказка.

4-5 Ноль, чтобы обозначить запись конца файла.

6-9 Начальный адрес для модуля относительно указанной основы перемещения. Этот адрес необязательно генерируется языковым процессором и может быть нулем.

10-11 Основа перемещения для начального адреса.

12-13 Контрольная сумма.

**Основания перемещения**

Одной из важных возможностей этого формата объектного модуля является возможность указать несколько оснований перемещения для содержимого модуля. Эти основания перемещений могут представлять ПЗУ по сравнению с пользовательскими программируемыми общими областями памяти, специальными областями памяти, такими как области обновления видео и т.д. В рамках модуля каждое из этих оснований перемещений присваивается имя и неявно последовательное число. Основаниям перемещения фактически присваиваются значения во время загрузки, но все ссылки на память в модуле выполняются относительно одного из этих оснований.

Четыре из оснований перемещения (от 0 до 3) имеют предопределенные имена и значения и обрабатываются по-разному во время загрузки, чем остальные основания. Основание 0 представляет собой абсолютные ячейки памяти (т.е.: всегда имеет значение 0). Основание 1 имеет имя ".PROG." и представляет собой область программы (может быть ПЗУ или ППЗУ). Большинство кода програмы генерируется относительно этого основания перемещения. Основа 2 имеет имя ".DATA." и представляет локальные области данных для каждого модуля. Большинство локальных данных определено относительно этого основания. Основание 3 имеет название ".BLNK." и представляет собой глобальный "пустой общий блок". Этому основанию перемещения всегда присваивается значение первого свободного адреса в памяти после локального хранилища данных (.DATA.) и других сегментов перемещаемых данных. Поскольку оно всегда является последним, модули, ссылающиеся на эту область, могут быть загружены в любом порядке, независимо от объема используемой области.

Перемещаемые сегменты относительно оснований 1 и 2 (.PROG. и .DATA.) всегда загружаются аддитивно. (т.е. После загрузки каждого модуля значение базы перемещений увеличивается на размер сегмента.) Предполагается, что все остальные основы перемещений имеют постоянные значения во время процесса загрузки и могут быть выделены загрузчиком.

**Документация CHESS.DOC для CHESS.COM**

(исходный код написан на FORTRAN)

25 декабря 1979 г.

CHESS.COM использует алгебраическую нотацию и отображает доску аналогично MICROCHESS:

A B C D E F G H

8 BR BN BB BQ BK BB BN BR 8

7 BP BP BP BP BP BP BP BP 7

6 -- :: -- :: -- :: -- :: 6

5 :: -- :: -- :: -- :: -- 5

4 -- :: -- :: -- :: -- :: 4

3 :: -- :: -- :: -- :: -- 3

2 WP WP WP WP WP WP WP WP 2

1 WR WN WB WQ WK WB WN WR 1

A B C D E F G H

Есть еще две команды, которые отображают доску и спрашивают компьютер, примет ли он ничью. это "BOARD" (ДОСКА) и "DRAW" (НИЧЬЯ) соответственно.

Чтобы рокироваться в сторону короля, введите O-O

Для рокировки в сторону королевы введите в O-OO

Если вы попробуете O-O-O, появится сообщение "ILLEGAL ATTEMPT TO CASTLE" (НЕЗАКОННАЯ ПОПЫТКА РОКИРОВКИ).

Всегда используйте заглавные буквы, fortran-chess не принимает строчные буквы.

Я провел несколько игр, сравнивая программу fortran-chess (fc) с программой **Chess Challenger 7** (cc7) (c) 1978 fidelity electronics. С fc на уровне 0 (у него только 0 и 1) и cc7 на уровне 1 (cc7 имеет 6 уровней игры) побеждает fc. С fc на уровне 0 и cc7 на уровне 2 победа cc7. Время отклика fc на уровне 0 варьируется, но никогда не превышает 90 секунд. Я еще не проверял его на уровне 1, но он имеет значительно большее время отклика (по крайней мере, несколько минут - оценка).

Получите удовольствие!

Джим Миллс

C.A.C.H.E. ЧЛЕН