REFERENCE MANUAL

1. Introduction

IFORTH stands for Forth on Intel-compatible processors.

The ANS Forth standard allows for a Forth that does not assume an underlying model of implementation. IFORTH uses this liberty to compile directly to optimized machine code. This Forth has a cell width of 32-bits and directly addresses all of its 4 Gigabyte address space.

The ANS Forth standard, like the Forth-83 standard, distinguishes between core words and extensions. In IFORTH all of extensions are present. This means any standard-complying program will run, irrespective of the extensions it uses. Nearly all words are available directly, the freedom allowed by the standard to supply extensions in source form only is used sparingly.

IFORTH does not run on 16-bit or 8-bit Intel processors. The advantage of the large unsegmented address space would be lost on them. IFORTH programs are portable to the transputer platform (tForth on T8xx and T4xx) and to some DSP processors (dspForth on the TMS320C30), when care is taken with hardware-specific issues.

1.1. This handbook

This handbook is intended to be a comprehensive documentation of the IFORTH system; not all of it is needed to get started. IFORTH is a standard system, so you might also want to read the ANS Forth documentation as issued by the ANSI (available electronically in ASCII and HTML formats).

After reading the introductory chapter you will need to go through the installation process as described in chapter 2. Chapter 3 contains some practical guidance for using the system.

The chapters 4 and 5 are, in fact, the implementation-dependent additions to ANS Forth. Chapter 4 describes all the implementation detail that is required by the standard. Chapter 5 is a summary of the available words, grouped logically rather than lexicographically. This includes all the standard words. Also words suggested by the standard (notably “programmer conveniences”) show up here, but not the system-dependent assembly language words. Note that information here is sometimes duplicated in the glossary.

The three following chapters will show you the words to adapt the system to your wishes, make a turn key application, and do some assembler programming.

The glossary, Forth’s ultimate reference, is in a separate section. It summarizes the words that are present in this Forth. Each word is listed with the extension (if any) to which it belongs. This will prove valuable if you want to write programs that have to be ported to a less fully equipped Forth.

The IFORTH environment and the server protocol are discussed in appendix I and II.

1.2. Starting up

The ANS Forth standard is comparatively new. I have tried to identify the tricky problems and discuss them below.

1.2.1. A word to the uninitiated

As I have said before, this is a reference manual. It contains little tutorial material, and you can not learn Forth from it. Alongside it you will need the ANS Forth standard.

To get started in Forth I recommend reading *Starting Forth*, see appendix III, or search your CDROM (`/dfwforth/examples/StartingForth/`) for the electronic version. You should be aware that this classic book is not fully compatible with ANS Forth, and used to be somewhat difficult to find[1]. *Forth: The New Model* fixes both problems but is silent about the important locals, files and floating-point word sets.

1.2.2. Forth can be compiled and still be standard

For those people still thinking of Forth as a threaded language it may come as a surprise that IFORTH compiles to machine code. By carefully defining the effect of compiling words the ANS Forth committee has succeeded in portability across interpreted and compiled systems. However, there is a small problem that you have to be aware of. The traditional notion of compilation as "comma'ing into the dictionary" has become invalid. For example: the word **'** will return a so-called execution token and its use is restricted—but still sufficient for the majority of uses it has been put to. All of this is documented meticulously in the standard.

1.2.3. A word to the long time file user

There are numerous Forth-ers who use files exclusively. They probably are pleased to see that files have made it into the standard. But the ANS standard does not simply trade blocks for files. It still requires blocks. That is, if a file word set is supplied it is mandatory to also supply the block word set. A slight disadvantage is that common words like **LOAD** are thus reserved for blocks. The word **INCLUDE** and derivations thereof are used to "load" files. All words that apply to files end in **FILE** e.g. **READ-FILE** . Some of the words containing **FILE** are in the section about blocks because they are only concerned with files containing blocks. These can safely be ignored by file-users. Note that blocks need not be mapped within files. A standard system could use a single block system that directly maps onto the sectors of a hard disk.

1.2.4. A word to the user of blocks

Religious wars… So I won't tell a block user to switch. I personally prefer files over blocks because inserting comments and adding extra indentation to existing definitions is much easier. In many systems, amongst them IFORTH, you can use your favorite text editor from within Forth. Porting sources to other Forth systems is also easier, especially if these are running under a host operating system.

There is no built-in block editor in the **EDITOR** word list. A block editor is available in the file *editor.frt* in the examples/blocks directory. Load it manually.

---

[1] It used to be available through mail order from FIG, USA. Try MPE in the UK, or Amazon.com in the "used books" section.

1.2.5. A word to the Forth-83 user

ANS Forth is largely based on the Forth-83 standard. Once you have studied the notes in the ANS Forth standard about the differences with Forth-83 you are ready to go.

One of the most notable differences is the introduction of **>NUMBER ACCEPT POSTPONE** to replace **CONVERT EXPECT** and the duo **COMPILE [COMPILE]** respectively. They clean up ambiguities and unpleasant properties of the old words.

The word **NOT** is replaced by **0=** or **INVERT** , depending on whether it inverts a logic result or is intended to invert all the bits of a cell on the stack.

Related to portability towards 32-bit systems, cell-counting and alignment words have been introduced.

Furthermore some longstanding practices have been honored, such as **RECURSE** and **EVALUATE** .

The controversy between rounding towards zero or towards minus infinity has been settled by providing both: **FM/MOD SM/REM** .

The philosophy of looping has not changed. It is still possible to do a wrap around loop by providing two equal limits. The usefulness of this in a 32-bit system is, of course, questionable.

1.2.6. A word to the user of Brodie's Starting Forth

The well known introductory text *Starting Forth* of Leo Brodie is based on polyFORTH or Forth-83 depending on whether you have the first or second edition. It still can be used in combination with IFORTH as long as you take heed that **'** and **CREATE** ... **DOES>** behave differently.

1.2.7. The hardware iForth runs on

IFORTH is written modularly. All hardware dependent parts are isolated and the adaptation to other hardware is comparatively straightforward.

For special wishes with regards the MS-DOS, Linux, or Windows products you have to contact the implementor. The UNIX version comes with "C" source code for a server program. This server is able to absorb most common implementation details.

1.3. Concepts

Most of the concepts and names used by the ANSI document are taken for granted. I will draw attention to a few that have been changed or may cause confusion.

A "character string" is described by an address and a count on the stack. This is the preferred method of the standard to pass strings around. Note that for parsing etc. it is not necessary to copy characters in order to return a string.

A "counted string" is represented by a single address, where the first character represents a count.

An "execution token" is a single cell on the stack that represents a word in the

form required by **COMPILE,** and **EXECUTE** . The idea of a “code field address” is gone. An execution token need not be an address and its use is restricted in ways detailed in the standard.

The space addressable by programs is called “data space”. The **HERE** area, parameter fields of words and tables created by **,** (comma) are all allocated in data space.

Executable code is put in the “code space”. A standard program may not assume that the code space is addressable.

When working with IFORTH you will discover that it will almost always find the files it needs. The word “findable” will be used to denote that a file can be found, the exact way of which is explained in the installation chapter.

The “memory manager” is the part of IFORTH that implements the optional memory allocation word set of ANS Forth. This word set allows to free and allocate memory in random order, unlike **ALLOT**.

A “module” is a word set of IFORTH that has a **REVISION** word. It can be forgotten as a whole, and features facilities like information hiding and its own help. It is always located in a separate file. All utilities are in the form of modules. A “module name” is the same as its **REVISION** word, and is used to identify it in glossaries and such.

The “iforth directory” plays an important role in finding system files, utilities and documentation. Normally it is the directory where you installed IFORTH.

A “dictionary entry address” is the principal address of a dictionary word. It can be used to find such data as the name field, data field and execution token.

1.4. Typographical conventions

All sections are numbered in a leveled hierarchy. Headers are bold and in successive smaller fonts for the lower levels. A general problem with documenting Forth is how to make Forth words stand out in the context. This is more severe than in other languages because words such as **,** and **;** can all be language elements. As you see from this example, I have adapted the convention from the ANS Forth documents to print those words in a bold font. Also a closing full stop, brackets or other punctuation are never concatenated with a Forth word. This produces silly sentences in a number of occasions.

2. Installation

2.1. Installation on an MS-DOS system

The installation is done under program control and should proceed smoothly. Just take care to follow the instructions.

After the installation you will have an IFORTH directory on hard disk, which in general would be added to your PATH so that the executable files can be found without having to change the current directory. Note that the dos-extender, GO32, additionally needs two environment variables to set it up properly. In order to be able to use the include and help files the DOS *append* command is recommended.

Once this is set up you may develop in any directory you wish. If you do not add the IFORTH directory to your PATH you must specify the whole installation directory path when starting IFORTH. The environment variables must be set up in your *autoexec.bat* or manually before starting IFORTH. The *append* command is optional.

2.1.1. Content of the installation diskettes

Apart from an *install.bat* and some auxiliary files, the installation diskettes contain archive files that each hold a directory you are about to create on your hard disk. The tree structure generated by the install command is important for the retrieval of files and must be preserved. The *examples* directory is an exception: it may be left out. The *bin* directory contains MS-DOS executables, the IFORTH binary, and driver files and executables for the DJGPP DOS-extender. The *doc* directory contains documentation files of which the online help file is the most important one. The *include* directory contains IFORTH source code of tools that you can include in your programs or use when developing. The *examples* directory contains source code for numerous programs. Run them to get an idea of the power of IFORTH or use them as a starting point for your own development. This directory is split into several sub-directories, grouping the example programs by subject.

2.1.2. The installation process

The result of the installation process will be a new directory on your hard disk. The space needed for a complete IFORTH is about 14 megabytes. For the installation you need to:

Decide upon a name for the sub-directory on your hard disk, make sure it does not already exist (we'll assume it is `c:\iforth` from now on). You may add this directory to your PATH. Anyhow, if you run the IFORTH that is present in this directory, it can find all the files it needs by virtue of the directory tree conventions, two additional environment variables, and the append command. I refrained from having the installation procedure modify your *autoexec.bat* file automatically for this.

Insert the IFORTH floppy into a floppy drive connected to your system, change the current drive to this drive (say a:) and type: `INSTALL a: c:\iforth` (assuming you want to install in `c:\iforth`). Don't forget the "a:".

Wait until the installation is finished. Edit the automatically generated *ith.bat* file for the proper path names and options. If there were no errors in the installation process you are now ready to use IFORTH.

About *ith.bat*. Typically the file will look like shown below, when you specify i:\ as the installation directory: (The line numbers between parens are for reference only).

```
( 1 ) @echo off
( 2 ) set GO32TMP=i:\tmp
( 3 ) set GO32=driver i:\bin\TRIDENTX.grd gw 640 gh 480
( 4 ) append i:\;i:\doc;i:\include;i:\include\fonts;
( 5 ) i:\bin\i3fe2m__ include iforth.prf %1 %2 %3 %4 %5 %6 %7 %8
```

Line 2 points to a directory where the dos-extender can keep its swap file. Line 3 is optional, but if left out the dos-extender will not use the high resolution modes of your SVGA card (if you have one). Instead of TRIDENTX.grd use the name corresponding to your SVGA card (If that driver is available in the `bin` directory). Line 4 points IFORTH to its `include`, `font` and `doc` directories. Without this, the word **NEEDS**, **HELP** and **IHELP** will not work and using the function keys will generate “file not found” errors.

Line 5 starts IFORTH. The first command IFORTH executes is to **INCLUDE** iforth.prf (command line parameters are explicitly allowed).

2.2. Installation on a Linux host

2.2.1. Unpacking

IFORTH comes on a CDROM, combined with the DOS (`./dfwforth/iforth`) and Windows (`./dfwforth/ifwinnt`) versions. You can place it anywhere you want, as long as the directory structure is kept intact. The final tree will look like this:

```
                .-----iflinux
                |     |-----bin
                |     `-----doc
 [..]dfwforth--+-----include-----fonts
                |-----dataf
                |-----meta
                `-----examples------... (many directories)
```

iForths for other platforms are added to above tree as follows:

```
                :
                |-----iforth
                |     |-----bin
                :     `-----doc
```

For the example shown, all IFORTHs use the same `include, dataf, meta` and `examples` directories. A slight nuisance is the different lf/cr+lf convention (your editor might be able to hide this for you, IFORTH can read either file format).

The Forth **OPEN-FILE** routine is aware of the above tree structure. If you

change it the **NEEDS** construct fails. You can add your own paths in iforth.c's *main()*.

IFORTH is not written in C. It is generated by a metacompiler which is itself written in IFORTH and in (Forth) assembler. This metacompiler and assembler know nothing about the format of Linux object files. Luckily Linux internals are such that binaries assembled for absolute addresses will work (when the used absolute address is chosen correctly). The C-server loads the iforth.img binary in an array, then jumps to the start address of this array (for this to work, code and data space of the unix must overlap, which is the case for Linux). From the above description it is easy to see that once IFORTH is up and running, it can extend itself in any number of ways and simply save the modified memory image back to disk to make the changes permanent (See **SAVE-SYSTEM**).

The C-server program *iforth.c* must be compiled by the superuser. Check out the options in the makefile, change them when you don't have the needed libraries, then type make. Afterwards the binary must be made accessible to all users (the supplied *makefile* does this by default).

There might be a number of warnings ("initialization from incompatible pointer type"). When not compiled by the superuser IFORTH still works, but things like line editing will be a pain. Also, some of the more useful demos and utilities may have to be rewritten.

Export a shell variable called **IFORTH** that points to the start of the IFORTH directory tree, like this:

```
export IFORTH=/home/gonzo/dfwforth
```

Now you can start IFORTH from any directory you want; the include and doc files will be found without the need for an explicit pathname (See **OPEN-FILE** above).

2.2.2. Files

For compatibility reasons you may want to use *only* lower case, and limit yourself to the 8+3 filename format.

2.2.3. Graphics and Mouse

IFORTH comes with a driver for the Linux vga, vgagl and vgamouse libraries. (Unfortunately these have been made obsolete with recent Linuxes. There is a driver for X too, however.) The one, two and three bytes per pixel modes are supported. Mouse and graphics work without any additional installation when the */etc/vga/libvga.config* file is correct[2]. In graphics mode you can still use the normal commandline editor PROCED. Try the S" name" **SET-FONT** command (or execute words like **MODERN16** etc. after executing **GRAPHICS**).

2.2.4. Audio

IFORTH will drive MIDI interfaces directly, without using system libraries,

[2] I had to export SVGA_MOUSE_OVERRIDE=1 in order to have the driver really use the mouse type entry in *libvga.config* (I have a 3-button mouse but could only use 2 of them). In extreme cases you may need to recompile the svga package.

because only a simple serial byte store-fetch is needed.

Audio-CDs can be played and even sampled.

In principle, all soundcards are supported.

2.2.5. Shell

IFORTH calls the standard /bin/sh shell when you type **0. SYSTEM** or **SHELL** . Normally /bin/sh is a link to your preferred login shell. Other commands (in *os.prf*) are also accessed through /bin/sh.

2.2.6. Signals

The ISERVER catches most signals and tries to do something intelligent with them:

- SIGFPE (arithmetic error, e.g. division by zero),
- SIGSEGV (illegal memory access)
- SIGINT (terminal interrupt character ^C)
  These write a message and perform the equivalent of **ABORT** .

  SIGPIPE (write to pipe with no readers, see /include/pipes.frt)

  Ignored as this is a common signal when using less etc.
- SIGWINCH (terminal window size changed)
  Caught and used to update **C/L** and **L/SCR** .
- SIGCONT (continue stopped process)
  Caught and used to reset the terminal to IFORTH's requirements, as the BASH job control doesn't do this automatically. By the way, you can interrupt IFORTH with ^Z and then continue later on with fg %<jobnumber>.
- SIGQUIT (terminal quit character ^\)
  IFORTH terminates when ^\ is typed.
- SIGTSTP (terminal stop character ^Z)
  IFORTH yields to the shell. Restart with the proper shell command (fg %1 for bash).
- SIGALRM (time out)
  This one is actually used by the timer read and write functions.
- SIGUSR1 (user signals)
- SIGUSR2
  SVGALIB uses these.

Other signals print a message and exit or ignore the signal. There is no guarantee that *all* Linux signals are caught. For instance, SIGINFO and SIGEMT don't exist.

2.2.7. Terminals

The relevant key and control strings are provided by termcap. Ncurses (i.e terminfo) is very large (200,000 bytes) and didn't work better than termcap, therefore I do not use it anymore.

The type-ahead buffer and ESC-sequence recognizer are organized such that full

terminal screens can be cut and pasted into IFORTH (using the mouse).

You should make sure that auto-wraparound is enabled for Xterms. If not, PROCED, IFORTH's command line editor, will behave strangely. (Open an Xterminal with the -aw option. Check the initialization file of your X-window manager, e.g. */etc/X11/fvwm/system.fvwm*).

iForth accesses virtual console screen memory through the */dev/vcs0* device. It is possible that this device does not exist. You can create it using the following script (see also the *makescrn* script):

```
#!/bin/sh
# Enables iForth to read/write the screen memory of the current
# virtual console.
# Used by the VSAVE VRESTORE VFREE @AT and !AT words.
# Must be done as root.
mknod -m 644 /dev/vcs0   c 7 0;
mknod -m 644 /dev/vcsa0  c 7 128;
chown root.tty /dev/vcs0
```

2.2.8. I/O permissions

Because IFORTH allows its users I/O-port access it can become a Trojan horse.

When *iForth.c* is not compiled by the superuser and/or when no `chmod +s` *iforth* is done, the OS will not grant I/O permissions. In this case some electives and programs won't work, but all kernel words are okay.

2.2.9. Technical note: addressing

Linux 1.2.14 or higher works such that the load address of `iforth[]` is always the same. So compile the server and run it, get the start address of `iforth[]` and metacompile IFORTH for the address found (or ask for a relocated binary when you don't have source).

Please note! It is neither necessary nor wanted to put IFORTH at the start of `iforth[]` (See below).

The offset at which the server stuffs the **.img* file in the array is purposefully chosen much too high. In this way additional code can be added to the iServer without needing a relocated IFORTH. The disadvantage is that some memory is wasted.

2.2.10. Panic

Because IFORTH turns off terminal echoing, things are not so nice when you crash directly to the OS, bypassing the iServer atexit procedures. In this case you are forced to "type blind", and may even have to switch to a different virtual console. For these situations I have added the following aliases to my shell (bash) :

```
alias xe='stty sane'
alias xt='stm 100x40'
```

I must admit to not having used xt for a long, long time, but xe is useful with

the SVGALIB that came with pre-2.0.0 Linuxes.

2.2.11. Preferences

Check the *.*prf* files in the `./include` directory, you will want to change them.

2.3. Installation on Windows

2.3.1. Unpacking

The IFORTH for Windows is closely modeled after the Linux version. It is commandline oriented, not GUI-based. Although it is a so-called console application, mouse-based copy and paste are supported. There is built-in support for the clipboard and for printing graphics and text. The main difference between Windows and Linux is that no server source code needs to be included (The Windows C-development system for IFORTH is about 5 times as expensive as IFORTH itself).

IFORTH comes on CDROM. You have to manually set two environment variables: My Computer→Control Panel→System→Environment, IFORTH = c:\dfwforth and IFORTHBIN = C:\dfwforth\ifwinnt\bin\iforth.img .

It is recommended to put a shortcut to IFORTH on the desktop, setup to start IFORTH in a 120x30 console with a 120x2500 screenbuffer. Enable QuickEdit.

You can place IFORTH anywhere you want, as long as the directory structure is kept intact. The final tree will look like this:

```
                .-----ifwinnt
                |     |-----bin
                |     `-----doc
 [..]dfwforth--+-----include-----fonts
                |-----dataf
                |-----meta
                `-----examples------... (many directories)
```

iForths for other platforms are added to above tree as follows:

```
                :
                |-----iflinux
                |     |-----bin
                :     `-----doc
```

For the example shown, all IFORTH's use the same `include, dataf, meta` and `examples` directories.

The Forth **OPEN-FILE** routine is aware of the above tree structure. If you change it the **NEEDS** construct fails.

Just like in the Linux case, IFORTH for Windows is generated by a metacompiler which is itself written in IFORTH and in (Forth) assembler. This metacompiler and assembler know nothing about the format of Windows object files. Windows internals are such that binaries assembled for absolute addresses will work when the used absolute address is chosen correctly. The C-server loads the

iforth.img binary in an array, then jumps to the start address of this array. Once IFORTH is up and running it can extend itself in any number of ways and simply save the modified memory image back to disk to make the changes permanent (See **SAVE-SYSTEM**).

When the environment variable **IFORTH** is defined as described above, IFORTH can be started from any directory; the include and doc files will be found without the need for an explicit pathname (See **OPEN-FILE** above).

2.3.2. Files

For compatibility reasons (with plain DOS) you may want to use *only* lower case, and limit yourself to the 8+3 filename format.

2.3.3. Graphics and Mouse

IFORTH automatically makes use of the Windows graphics libraries, assuming a four byte per pixel mode. Graphics are output to an additional window. There are three modes: text, graphics with text going to the console, and graphics with text going to the graphics console (default when the graphics console is open). In full graphics mode you can still use the normal commandline editor PROCED. Neither the console nor the graphics window have a conventional message loop. One result is that it is not necessary for the graphics window to have the focus in order to process keystrokes. A big drawback is that the graphics window will not redraw automatically when it is overwritten by another application's output. IFORTH can test when its graphic window is contaminated, but never redraws the screen automatically.

2.3.4. Audio

IFORTH uses the WIN32 MIDI and PCM libraries. It is possible to use up to four sequencer and two wave devices at the same time.

Audio-CD's can be played and sampled using the MCI library.

2.3.5. Shell

IFORTH calls the standard cmd shell when you type **0. SYSTEM** or **SHELL** . Most commands in *os.prf* are accessed through cmd.

2.3.6. Signals

The ISERVER catches most signals and tries to do something intelligent with them. See the Linux implementation in paragraph 2.2.6.

2.3.7. Terminal

The relevant key and control strings are provided and work for international keyboard layouts.

The type-ahead buffer is organized such that full terminal screens can be cut and pasted into IFORTH (using the mouse or the clipboard). When PROCED is active <TAB> provides a history mechanism, <^V> inserts clipboard text and <PgDn> completes filenames.

2.3.8. I/O permissions

On a *correctly* configured Windows system the IFORTH I/O-port access words do

nothing. Free-ware device drivers are available for NT to “unlock” ranges of I/O addresses, see the `ifwinnt` directory.

2.3.9. Preferences

Check the *.*prf* files in the `./include` directory, you will want to change them.

2.4. General directions

2.4.1. Configuration files in general

After installation, the iforth directory will contain a batch file *ith.bat* that is to be executed whenever you want to run IFORTH. *ith.bat* will configure the dos-extender and start IFORTH in such a way that it includes *iforth.prf* by default. (With Linux the shell command file is called *ith*, under Windows you have a shortcut to iserver.exe as recommended above. There is no need to start “extender” programs now, but *iforth.prf* is still passed as the first argument). The file *iforth.prf* contains IFORTH commands that are automatically executed whenever IFORTH is started. I recommend to not place source code in *iforth.prf*. Instead, place the source code in the include directory and **INCLUDE** the file. The way *proced.frt* is handled may serve as an example.

2.5. Customizing IFORTH

IFORTH can be customized by walking through the *.prf* files in both the `iforth` directory and the `include` directory, and disabling and enabling features as you go, guided by the description of the features present in the files. The settings such as given upon installation are appropriate for development, but require a comparatively large amount of extra space. You must expect to go back to this procedure as you learn more of IFORTH and discover possibilities you did not appreciate in the beginning.

Unless you are using block files, you may want to install your favorite text file editor before anything else. This is done by changing the file *os.prf*. Just replace the “editor” command by the filename of your favorite editor.

The commands you will need most are already present, but I am sure that you will want to add some more. Be warned that some of the more sophisticated tools assume that the “iforth directory” is in your path. In particular this applies to the facility in *os.frt* to start up programs using function keys.

2.6. The installed system

In this section I will describe the installed system in somewhat more detail. It is split in sections concerning the contents of the `iforth` directory itself and of sub-directories containing the binary files and the source for some of the utilities.

The examples are numerous, hence they are split over several directories. Their description is intended as an overview. You will find many more examples than are described here. When I think an example is interesting and sufficiently self-contained and documented, it may get added even after the current manual is completed.

You are entitled to incorporate sources partly or wholly into your programs,

however you yourself are responsible for the applicability thereof and the correctness of the resulting program.

2.6.1. Content of the IFORTH directory

The IFORTH directory contains the batch file *ith.bat* (or the shell script *ith*, or the shortcut to iserver.exe) that allows you to run IFORTH. It also contains the configuration file *iforth.prf* that is mentioned above.

2.6.2. The binary files executable by IFORTH

The binary system files all reside in the bin subdirectory. For Linux and Windows it is simple, there is only one binary (double precision floating-point) and there are no options.

Under MS-DOS the files i3__.exe contain the executable IFORTH system for the 386/486/Pentium with native floating-point arithmetic ('__' can be 'fe', 'fd' or 'fs' or ''). The binary i3.exe doesn't use floating point (meant for i386 systems without a co-processor, not on the distribution disks but available on request).

MS-DOS executable IFORTH systems have a filename constructed of:

- the letter 'i';
- 1 character identifying the type of processor this system is intended to run on: 3 for '386 type processors, 4 for '486 type, 5 for Pentium type.
- 1 character identifying the type of floating-point support present in the system: e for emulated floating-point, f for native floating-point or '_' for no floating-point at all.
- 1 character identifying the precision of the floating-point numbers: s for IEEE single precision floating-point numbers, d for IEEE double precision, e for IEEE extended precision floating-point numbers. The character is an '_' when there is no floating-point support.
- 4 '_' characters are added as placeholders for possible future configuration switches; 2 of these are used by iForth to indicate how much dynamic memory will be maximally used ('1M', '2M', '5M', '8M' etc, see note 2.)
- The file extension .exe .

Note1: As in practice everybody seems to use i3fe.exe, I have decided to save space on the distribution disks and leave out the other possibilities. Mail the author if you still want them (the single precision float version is about 10% faster than i3fe.exe)

Note2: In addition to the co-processor option, we also have to deal with memory size. Because of technical problems with GO32 I need to specify the room available to **ALLOCATE** at compile time as 100 Kbytes. In *iforth.prf* this is then changed to several MBytes.

2.6.3. The utilities

A number of useful utilities are present in the `include` subdirectory of the `iforth` directory. Some of these utilities are already included during startup by the file *iforth.prf*, as explained in earlier sections of this chapter. Other ones you will find yourself incorporating in projects.

The file *miscutil.frt* contains miscellaneous utilities. It is used by almost all of the other utilities and contains many goodies that may be useful in general Forth programming.

The file *terminal.frt* provides functions to control specific parts of your display.

The file *proced.frt* contains a command line editor à la ced.com or bash. This is a utility that you will typically include in your *iforth.prf*. It features command history. This means that you can recall previous commands with the "up arrow" and "down arrow" keys and edit them before issuing them again. The TAB key completes a partially typed line to the best matching line in the history buffer (it also substitutes files like a bash user might expect, try the PgDn key). With ^V the contents of the clipboard are inserted on the command line. ESC clears the line.

For heavy-duty floating-point code use the FSL matrix words from *fsl_util.frt*.

The file *backtrac.frt* can be useful with difficult debugging problems. It prints a stack trace, i.e. an overview of the calls via which you have arrived at the current position. Some people swear by it.

The file *see.frt* contains a source code viewer, such as specified in the ANS Forth **SEE** command. Because of the space required for this definition it is delivered in source form.

The file *graphics.frt* contains (not so-) basic facilities for graphics.

The file *os.frt* contains several utilities to use the operating system on which the server is running.

The file *needs.frt* makes it extremely simple to automatically load all the modules a project needs. Modules will be loaded only when they are not already present.

2.6.4. The helpfiles

The helpfiles can be found in the subdirectory "doc" of the IFORTH directory. There is a general helpfile, called forth.hlp for all built-in words and a helpfile for the utilities mentioned in the previous section. The latest addition is a file called i4thhelp.htm that is automatically generated from the ASCII *.hlp files. This hyperlinked file can be viewed with an HTML aware editor or browser (Lynx, Mosaic, Microsoft's Internet Explorer, or Netscape).

2.6.5. The benchmarks

The benchmarks can be found in the subdirectory *examples/benchmar* of your IFORTH directory. Part of the benchmarks have been adapted from existing C-benchmarks. It may seem dubious whether straightforwardly recoding C-programs into Forth is anywhere near appropriate. But anyway, the performance of the resulting programs is always comparable with that of the C-originals. The compilation time is somewhat faster than the time consumed by current real-mode C-development tools. All programs have a **.HELP** command that show you some information about the benchmark.

ackerman.frt is a classical ADA and Modula II benchmark, testing recursion.

dhryston.frt is a Forth version of the well known DHRYSTONE benchmark. It contains a typical (for C or FORTRAN that is) mix of instructions, but no floating-point instructions.

mathtest.frt contains a simple test for floating-point functions, that can easily be interpreted without need for much study.

savage.frt is also a very well known floating-point test. It gives an indication of the speed and precision attained by the floating-point package.

The file *sieve.frt* contains, apart from the classical BYTE sieve benchmark, a version of the sieving program that makes full use of the facilities in IFORTH. It has become faster and more readable.

speedtes.frt tests and compares the speed of the two counted loop mechanisms available in IFORTH.

thread.frt is another classical Forth benchmark, testing the speed of nesting.

whetston.frt is a Forth version of the well known WHETSTONE benchmark. It contains a typical (for C or FORTRAN that is) mix of instructions, in particular floating-point instructions. It gives an indication of the speed and precision of floating-point.

2.6.6. The strict ANS Forth examples

The directory *examples/ansi* contains only pure ANS Forth programs, that should run without modification on any ANS Forth platform. They show techniques to use specific IFORTH capabilities without destroying portability.

The file *horst.frt* contains an iterative version of the HORST algorithm for the factoring of numbers. It can handle very large number as far as space is concerned, but running it will wear out your patience.

towers.frt contains an almost classical programming example, the towers of Hanoi. This version employs character graphics to get some nice graphical effects within the constraints of ANS Forth.

2.6.7. Showing off IFORTH: the graphics examples

In the directory *examples/graphics* you will find some graphics examples. They all use the graphics toolkit provided in *graphics.frt*.

The program *fractals.frt* shows some Mandelbrot pictures, especially nice on a high resolution color screen.

fern.frt contains a program that draws ferns. It is also fractal based, and you may experiment to get different ferns by specifying different parameters.

2.6.8.The miscellaneous examples

The directory examples/misc contains examples that do not fit well in one of the foregoing categories.

abacus.frt is a calculator, comparable to the traditional desk calculator that comes with some unix systems.

myclock.frt presents a clock on the screen. It is another demonstration of

character graphics.

The file *roots.frt* is a numerical example. It contains methods to find roots of real functions, i.a. bisection and newton.

2.6.9. Other files

Many other files or even sub-directories of examples were added to IFORTH after this manual was printed. It is left as an exercise to the user to find out what these files do and whether they are useful.

2.7. Configuration per project or per user

Professional developers will want to divide their activities into projects with as little interference between them as possible. Projects may be executed by different persons with different preferences. The way to do this in IFORTH is to create a project directory to place all relevant sources. Put a copy of *iforth.prf* and other preference files in that directory. If you start IFORTH from this directory, either by specifying the full path of *ith.bat* or with the IFORTH directory added to your path, the local files will override those from the IFORTH directory. A modified version of a utility from the include directory will override the original utility.

3. Working with IFORTH

This chapter is describes working with IFORTH. We will walk through a setup, such that you can start productive work as soon as possible. But some feeling for the inner workings is useful.

3.1. Some internals

IFORTH uses a "virtual server" concept in its I/O architecture. There is no real server present with IFORTH for MS-DOS, but there is one for Linux, Windows, and other DFW products (e.g. tForth). This server concept allows us to write all DFW products in a modular fashion, minimizing the differences between the Forths to a limited number of primitives.

IFORTH behaves as if it has no direct access to the keyboard, screen and disk. The "server" is a subprogram that accepts commands, executes them and passes the answer back to its so-called client. (The protocol is detailed in appendix II). Basically, commands that should not be executed by IFORTH (because they involve direct hardware manipulation) are passed to the server that interprets and executes them. The server is written in C and linked with IFORTH, or it is implemented as a device driver. In this way IFORTH can be ported to any system that has a C-compiler installed (if you have the source code).

IFORTH has no problem whatsoever in manipulating any of the hardware available in a PC. However, the kernel is written to minimize hardware dependency as much as possible, to enable easy ports to future operating systems or different hardware. The technique has been pioneered with the tForth product, with very rewarding results. tForth and IFORTH have almost the same look and feel, only differing in the parallel programming tForth offers. The same source code will run on both Forths.

The present MS-DOS server is not written in C, but in a mixture of assembler and Forth. The C-approach is used for Windows and Linux.

3.1.1. Processor startup

For Linux and Windows not much needs to be done.

For MS-DOS, DJ Delorie's 32-bit GO32 DOS-extender is prepended to the IFORTH executable. GO32 places the processor in protected mode and loads graphic and ANSI drivers, then runs IFORTH. When IFORTH takes control it parses the command line arguments and adjusts the memory map, when asked to. After that the numeric co-processor is initialized. If no numeric co-processor is available IFORTH can not be used (unless you ordered the special i3.exe binary).

3.1.2. Capabilities of the server

The (virtual) server program is built into IFORTH. The behavior of this "server" can be controlled with command line options. The most important option is that all that is on the remainder of the command line invoking the server (i3__.exe for MS-DOS), is in principle passed to IFORTH and executed by it. One of the commands you can pass to IFORTH is **INCLUDE** *iforth.prf* to pull in your preferences.

3.2. Command line options

The syntax for invoking IFORTH with command line arguments is:

```
ith [cmds]
```

The square brackets denote that a part of the command is optional. Most of the time, once the system has been set up, you will just type ith.

The part called *cmds* can be any string. It contains Forth commands and is just passed to IFORTH transparently. It is executed as if this text where typed in from the terminal.

3.3. The edit-compile cycle

The edit-compile cycle for IFORTH. It is recommended to go to the directory where you want to keep your source files. Once in IFORTH you can load a file by typing

INCLUDE `filename`

You can edit a file by typing

EDIT `filename`

Here I assume that you have installed your favorite text file editor according to the instructions in the previous chapter. Using the editor is based on the important and versatile **SYSTEM** command, explained in a separate section.

The (non-standard) word **INCLUDE** does some preparatory work that culminates in calling **INCLUDE-FILE** , which is the standard word that expects the descriptor of an open file and interprets all the lines of the file. **INCLUDE** also closes the input file.

This is basically all you need for the edit-compile cycle. IFORTH has still one feature to make life easier for the programmer. That is the word **REVISION** . It is used as follows:

```
REVISION -miscutil “--- Several utilities Version 2.01 ---“
```

It creates a word that can be used to forget the application, customary the filename preceded with ‘-‘ (dash). It serves the same purpose as the standard word **MARKER**, but it has a string attached to it that identifies the application. **REVISION** also inspects whether a word with the name already exists, and in that case executes it, i.e. it removes the previous instance of the application. That is where its name comes from. It is explained in more detail in a separate section, together with some complementing facility words.

There you are. You now know enough to start programming. The programmer conveniences all behave as you will expect, but of course a study of the “implementation overview” chapter will help you to get the most out of it.

3.4. Doing what Forth cannot do

When working with IFORTH you have three facilities working for you at once. These are IFORTH itself, the operating system and the server. It may be that you want something done that cannot be done by IFORTH itself. That is where the notion of an escape comes in. By default you are “talking” to IFORTH but you may escape to the operating system or the server by special commands that are called

escapes. For both the operating system and the server escapes have been built in.

3.4.1. Escaping to the Operating System

The (non-standard) word **SYSTEM** makes the operating system execute the commands contained in a counted string. After completion of the commands it returns you to IFORTH immediately. Words related to this are found in the utility *include/os.frt*.

This command is at the heart of the **OS-IMPORT** facility described in the installation chapter. The **OS-IMPORT** , as its name suggest, makes an operating system command available from within IFORTH.

```
$" COPY" OS-IMPORT COPY
```

will create a word called **COPY** that passes the string "COPY" to **SYSTEM** . It also passes the remainder of the line, e.g.

```
COPY D:\FORTH\F?.TXT B:*.BAK
```

is a Forth line that does the same as in MS-DOS / Windows.

The **OS-IMPORT** command is also used to put the operating system escape in a syntactically more convenient form. In the default setup, the command **OS**, itself defined with **OS-IMPORT**, will simply pass what is typed on the remainder of the line to the operating system, e.g. for Linux:

```
OS mv /DOS/forth/examples/* /dev/fd0/examples/.
```

With additional parameters you can request **OS-IMPORT** to transform arguments before the operating system is called. It can accomplish such things as starting an editor with the cursor at the position where an error occurred during including. See *doc/os.hlp* for more information. *os.prf* is an accompanying file with preferences. With that as a guide you should be able to use the **OS-IMPORT** facility to your advantage, without too much work.

3.4.2. Escapes supplied by the server

The server could in principle offer a possibility to talk to it directly. Useful commands that it can execute are the resetting of IFORTH or a return to the operating system. However, this has not been implemented for the current MS-DOS and Linux IFORTH (with tForth you can sometimes call the Forth that the server is written in).

3.5. How IFORTH finds its files

Files needed by an application, whether they are source files, data files or block files, are first searched for in the current directory. Thus you may override each system file. If a file is not present in the current directory the IFORTH-directory is searched and its sub-directories `doc`, `include` and `bin`. The IFORTH-directory is the directory where the *ith.bat* or *ith* file is located. Under MS-DOS this feature is implemented with the DOS `append` command. The Linux and other C-servers maintain a compiled-in path array.

3.6. Modularity

In IFORTH you have available all words defined by ANS Forth. As Forth is sometimes described as a tool to make application specific languages, it is useful to have language extensions available. The facilities I have added to IFORTH are subdivided into “modules”, one per file. Each module has its own “revision word” or “module name”, conventionally derived from the file name with a leading ‘-‘ (dash). So “-strings” is the revision word for the module in *strings.frt*. If the file is loaded again, after you revised it, **REVISION** will make sure that the old code is forgotten.

Modularity is increased by the **NEEDS** facility. By specifying **NEEDS** <module name> in a source, the required facility is loaded if it was not already present. It may need other modules itself, but the only thing you have to do is check out from which modules you need a specific word.

IFORTH would be a monster if all the auxiliary words present in modules were visible. This “name space clutter” problem is present in all large interpretive systems. Wordlists have done very little to relieve this. The reason is that if you need a single word, you get the complete wordlist. Therefore, some people feel that wordlists are a mere inconvenience. In IFORTH this problem is addressed by the **PRIVATES DEPRIVE** mechanism. All words in a module that are present between **PRIVATES** and **DEPRIVE** and are marked by **PRIVATE** , are not visible once the module is loaded. For instance, the only word visible from the help facility is **HELP** itself.

The modules loaded in memory can be listed by typing **.MODULES** . It is also possible to add extensive help to a module by the word **:ABOUT** . That information can be inspected by either **.HELP** for the current module, i.e. the last loaded module with a revision word in place, or with **.ABOUT** module. In header lines of glossaries (such as .hlp files) the name of the ANS word group is placed at the extreme right. For glossary entries referring to a word from a module, the module name is placed in that position. They are easily identified because of the leading dash.

If you have a large application, revisions save considerable recompilation. After editing a module, load the application again. All the low level utilities are still in memory and will not reload.

Tools that are permanently loaded are best supported by a permanent addition to the general help file forth.hlp. Help files are ordinary text files[3]. The format of help files is described in *help.frt*. It amounts to glossary entries separated by lines containing “#####”. If you are faithful to this format you can edit *forth.hlp* directly.

3.7. Old (fashioned) hands are on their own...

If you insist on using block files, your text editor cannot handle your Forth

[3] Some minimal formatting has been added since the addition of the HTML converter utilities. Check the file header for details.

sources and all the facilities described above are wasted on you. You probably don't mind because you have your own facilities, so I did not bother too much.

What I did do is make a block editor available so that you can get started. The **EDITOR** wordlist is present already. If you have the system properly installed, you will find a file *editor.frt* in the subdirectory `examples/blocks`, containing an ANS Forth block editor.

Blocks are allocated in operating system files.

4. Implementation overview

This section details what is left to the implementation by the standard. It describes concepts and facilities that IFORTH provides, up to and beyond the standard.

4.1. General

In IFORTH, **1 CELLS** is equal to the word size of the underlying hardware.

IFORTH is a byte addressing, 32-bit Forth. This means, that the smallest element that can be addressed is a byte (8 bits). Incrementing an address by one makes it point to the next byte. The cell size is 32-bits, i.e. the data handled by all common Forth words like **DUP @** and the parameter and return stack are all 32 bits wide. Consequently a cell occupies 4 address units. A character occupies a single byte. Double precision words are implemented, and handle 64 bits at a time. An extended precision floating-point number takes 12 bytes in memory and three cells on the fp-stack (IFORTH uses only 10 bytes of these).

A good standard program will run irrespective of the cell size, except when you really need those 32 bits, be it for memory addressing or precision.

IFORTH comes in different configurations. You may check the configuration using **ENVIRONMENT?** . For each configuration option provided, the environment enquiries are included in this document. In IFORTH some information about the configuration can be made visible by **.SIGNON** , or you can generate an extensive report by running the utility *whatenv.frt*.

4.2. The stacks

IFORTH has five stacks: the data stack, the return stack, the system stack, the local data stack and the floating-point stack. Calculation data is on the data stack, except for floating-point data. Note that the fact that floating point numbers reside on a separate stack has a profound influence on any calculation that uses floating-point.

The return stack is used for subroutine nesting and for the storage of loop variables.

The local variables are stored on a separate stack, called the local data stack, that is used for nothing else. The standard allows local data to be allocated on the return stack. iForth's implementation of locals has none of the restrictions that would result.

ANS Forth prescribes that control data and jump addresses are stored on a so-called control stack. IFORTH stores all these on the return stack.

The system stack is often used as an extra stack for data manipulation in applications. It has the advantage over the return stack that data need not be pushed and popped in the same word, and that mistakes are not immediately fatal. The other usages of the system stack are such that it effectively may be considered as a miscellaneous multi-purpose stack.

4.3. Floating point

IFORTH uses the built-in floating-point processor but since the hardware does not provide all of the necessary algorithms, some of the transcendental functions had to be emulated.

The result is a full IEEE 752 implementation of single, double and extended precision floating-point. The default, implemented by the **F**… words (**F+, F*** etc.), is double-precision, 64-bit.

Single or extended precision is available by specifying them with **SF**… or **XF**.., e.g. **SF+ SF*** or **XF+ XF*** etc. . The FP stack is wide enough to hold an **XFLOAT** with full precision (80 bits).

4.4. Numbers and their ranges

IFORTH has built-in floating-point. The floating-point can be either single, double or extended precision, or absent. So all in all there are four configurations with respect to floating-point alone. The single precision floating-point numbers are stored on a separate stack and are 32 bits wide, IEEE standard single precision floating-point numbers.

The extended precision floating-point numbers are 80 bits wide, IEEE standard extended precision floating-point numbers. It turns out that there is a slight speed advantage, approximately 15%, in using double or single precision

type	*range (hex)*
n	-8000,0000 .. 7FFF,FFFF
u	0 .. FFFF,FFFF
+n	0 .. 7FFF,FFFF
d	-8000,0000,0000,0000 .. +7FFF,FFFF,FFFF,FFFF
ud	0 .. FFFF,FFFF,FFFF,FFFF
+d	0 .. 7FFF,FFFF,FFFF,FFFF
f	1.18E-38 .. 3.4E34 signed, 6.9 digits (1)
f	2.23E-308 .. 1.79E308 signed, 15.6 digits (2)
f	3.37E-4932 .. 1.18E4932 signed, 18 digits (3)

1 Number ranges

numbers. Of course there is a significant space advantage. IFORTH provides the standard words **SF! DF!** (floating store single or double) and **SF@ DF@** (floating fetch single or double) to be able to use single or double precision float variables where it counts. Ranges for numbers can be found in table 1. The hexadecimal base is used because this eases interpretation of the ranges, except for the single and double precision floating-point numbers.

There are essentially three version of floating-point possible. If floating point is present (this may be seen from the environment variable **FLOATING**), precision is either (1), (2) or (3) depending on whether the single, double or extended precision version of the package is included. This can be tested with the environment variables **DOUBLE-PRECISION** and **EXTENDED-**

PRECISION .

4.5. System limitations

IFORTH will run any ANS Forth program provided none of the maxima listed in appendix I are surpassed.

Some of the maxima are no restrictions of IFORTH per se, but are caused by the host system.

IFORTH itself does not restrict file names in length or with respect to the characters it may contain. It is recommended that for portable programs no case sensitivity is assumed, and that the length be restricted to 8 characters plus an extension of 3 (similar to MS-DOS). But see the section on the host system.

Since not all of these values are fixed, the utility program *whatenv.frt* is provided that can list the values of almost all environment variables.

4.5.1. File handling for MS-DOS/Windows/Linux hosts

Some of the restrictions and possibilities that apply to IFORTH are imposed by the host operating system rather than IFORTH itself.

File names are (voluntarily) restricted to 8 characters plus 3 characters, separated by a dot. These are the customary MS-DOS/Windows path names, drive letter, colon, path constituents separated by slashes, e.g.:

```
c:/iforth/include/miscutil.frt
```

Please note that all file names used in IFORTH can be written with either '/' (slash) or '\' (backslash) as the path separator, but also note that this might not be true for other programs such as those called by the **SYSTEM** or **OS** command. All file access errors that are detected by the host OS are also recognized by IFORTH.

4.5.2.Terminal interaction

The system prompt is composed of the name of the topmost wordlist to be searched, plus a '>' sign. While you are typing a definition that spans several lines, the system prompt is placed between square brackets ('[' and ']'). As a matter of concession to Charles Moore, and because ANS requires it, 'ok' is still printed after execution of a word.

The ANS Forth standard requires two words for terminal interaction: **EXPECT** and **ACCEPT** . Both are present in IFORTH with the editing provided by the server. The <BACKSPACE> is accepted as a "rubout character". If it is typed it is not put in the input buffer. Instead the last character present in the input buffer is erased. The display is updated to reflect the actual content of the buffer. Specifically, IFORTH will try to erase the faulty character by typing it over with a blank.

The word **EMIT** actually sends non-graphic characters too.

4.5.3. Terminal interaction for PC hardware

The <BACKSPACE> and <ENTER> key of the PC keyboard description serve as <RUBOUT> and <ENTER> keys for IFORTH. This just means that they behave as is

usual under MS-DOS/Windows/Linux.

4.6. Programmer conveniences

The standard word **ENVIRONMENT?** is provided by IFORTH. It allows the programmer access to knowledge about the Forth system as it is running on the moment of invocation. In appendix I a table is presented of the values returned for all possible enquiries. Refer to that section for an overview of what enquiries are possible.

Apart from the words required or mentioned as optional in the standard there are some IFORTH specific words. The environment variable **IFORTH** is set in a IFORTH system. This allows you to use IFORTH specific possibilities by conditionally compiling code that is automatically ignored on other systems.

The standard word **DUMP** is provided by IFORTH. It shows the addresses and content in hexadecimal and also as characters, when possible.

The standard word **BYE** disconnects the server code from IFORTH. The system command interpreter gains control (more precisely the program that started IFORTH in the first place).

The standard word **SEE** is only available after loading the file *see.frt*. It displays the Forth word as a sequence of machine code instructions. Most simple words (like e.g. **DUP**) are not recognizable in the output. The optimizing compiler has translated the glue words into machine instructions directly. However, a colon definition that is called shows up as a subroutine call with a symbolic address.

The standard word **.S** prints the data stack. But it will also print the system stack and the floating-point stack, and will use the current **BASE** (however, floating-point numbers are always printed in decimal). Furthermore, numbers on the data stack are printed unsigned unless the base is decimal.

In addition IFORTH supplies the non standard words **.DATA .SYSTEM .FLOAT** to print the data, system and floating-point stacks separately.

The standard word **WORDS** prints all the wordnames in the current dictionary, sssllloooowwwlllyyy. Its scrolling display can be stopped by pressing any key. It will resume after pressing some other key, and abort by pressing <ESC>. So **WORDS** can be terminated by pressing <ESC> once. Please try out **WORDS: WORDS?** and **doWORDS** (you can get help on these by typing `help <word>` at the Forth prompt).

The standard (but obsolescent) word **FORGET** takes a string argument from the input stream and deletes from the dictionary all the words after the one specified by that string. It is a somewhat dangerous word and has to be used with care. Do not **FORGET** words that are executing in some parallel (multi-tasking) process. Also do not forget executable code that is called via forward reference. Please note that **FORGET** does not return the memory allocated with **ALLOCATE** , only what has been allocated in the dictionary by using **ALLOT** . Forget fields can be used to free blocks of memory when the reference to that memory is forgotten.

The standard word **MARKER** creates a word that will forget itself and all the

words after it. It is intended to be much safer than **FORGET** and not subject to all of its restrictions. In IFORTH, **MARKER** uses **FORGET** repeatedly. **FORGET** is much safer than on most conventional systems to date, especially when forget fields are used. With properly initialized forget fields the ANS Forth restrictions on **FORGET** and **MARKER** do not necessarily apply to IFORTH.

The non standard word **HELP** is a great help. It shows the glossary information of a word as available in this manual. When the file *help.frt* is loaded it can be used as: **HELP** *item* .

Information on any item in any include file can be displayed with **IHELP** *item*.

4.7. Introducing the assembler

For details about the assembler see chapter 8.

No Forth system is complete without an assembler. The assembler is supplied in the form of a wordlist called **ASSEMBLER** and an assembler word is initiated by the compiling word **CODE** . **CODE** will put **ASSEMBLER** in the first place of the search order.

No details concerning a processor instruction set are to be expected in a Forth document. Here too, I have to refer to the *ix86 Microprocessor Programmer's Reference Manual* (appendix III). Apart from that I will restrict myself to a few remarks. The instructions as to be found in the *ix86 Microprocessor Programmer's Reference Manual* are used, but in a Forth fashion. Each instruction is placed after the data and a comma (',') is appended to each instruction word. In general no labels are used, instead the structured assembler words **BEGIN, WHILE, REPEAT, IF, ELSE, ENDIF,** are provided. They are used in a conventional way, and behave identically to the corresponding Forth structuring words. **END-CODE** saves the new word and eliminates **ASSEMBLER** from the search order.

The control information needed by these words is placed on the data stack. The combination **:** ... **CREATE** ... **;CODE** ... **END-CODE** may be used to make defining words whose run time action is defined in assembler.

5. The Language

In this chapter the words available in IFORTH are explained, with the exception of some special subjects like assembly language programming. These will be treated in separate chapters.

The main partition is along the line of the wordlists in IFORTH. The words you would normally use in applications are in the **FORTH** wordlist. This comprises effectively all of the ANS Standard Forth words, but there is a lot of "luxury" above and beyond the standard. Underneath all this are the system words, the nuts and bolts that keep IFORTH together. Extending the system itself is one of the most powerful properties of Forth. So I do not want to withhold these internals from you. But it is wise to keep them separated from the commonly used words. This is done by putting them in a separate section. None of the ANS Forth words are to be found in this wordlist, of course.

The words are grouped together in such a way that you will be able to find out whether a certain word is present, without the need to know its spelling. In this language section the stress is on understanding the relationship between words, an issue often neglected in Forth documentation. For the situation where you need to know the precise action of a word, you are referred to the glossary chapter.

The glossary also shows whether a word is in the core set, in which extension, and if it is in the standard at all.

In this chapter you will only see whether a word is in the standard or not. All words that are in the standard are implemented in IFORTH.

5.1. We are not trying to teach you Forth

Descriptions are only given for non standard extensions. If you do not know the meaning of a standard word, you will have to look it up in the glossary or in the standard. Even if a word is not standard, conciseness is favored, but completeness is never sacrificed. Where possible detailed descriptions are omitted, especially for those words whose meaning is obvious or analogous to documented words.

Use the glossary whenever you need precise knowledge about a word's behavior.

5.2. The words

As announced, the available words are presented in logical groups. (If you want to look up a word that you know already, look in the glossary.) Even for the standard words this grouping probably is a valuable addition to the standard. For example if you wonder whether there is a word that duplicates the top four items on the stack, you try to look it up in the stack manipulation group. There is nothing there with such a meaning, so it probably does not exist.

Also it helps you out if you know a word with a certain effect exists, but you have forgotten how it is spelled.

5.2.1. The stack manipulation group

IFORTH provides the following standard words for manipulating the data stack: **2DROP 2DUP 2OVER 2ROT 2SWAP ?DUP DEPTH DROP DUP NIP OVER ROT PICK ROLL SWAP TUCK** .

Also are provided **-ROT** equivalent to **ROT ROT** , and **3DUP** similar to **2DUP** but copying the top three stack items.

IFORTH provides the following standard words for manipulations involving both return stack and standard data stack: **R@ 2>R 2R> 2R@ >R R>** .

IFORTH provides the following standard words for manipulations involving the floating-point stack: **FDEPTH FDROP FDUP FOVER FROT FSWAP** .

In addition the following words are provided with obvious meaning: **F2DUP FNIP FOVER** .

Then there is **-FROT** equivalent to **FROT FROT** .

It may be seen from the environment enquiry **FLOATING-STACK** that floating point numbers are on a separate stack for IFORTH. Unfortunately it is not easy to write code that is transparent with respect to whether a floating point stack is used, or floating-point numbers are on the arithmetic stack. Consider for example the case where a float is fetched from an address on the data stack. Certainly words that swap floats with integers etc. are justifiably absent.

Because use of the return stack is restricted and inherently dangerous, a third stack, called system stack, is provided by IFORTH. The words **>S S> S** can be used instead of **>R R> R@** . The words **SDEPTH SPICK** for the system stack are similar to **DEPTH PICK** for the standard data stack.

5.2.2. The integer and address arithmetic group

IFORTH provides the following standard words for integer annex address manipulation: **- + 1+ 1- ABS D+ D- DABS DMAX DMIN DNEGATE M+ MAX MIN NEGATE** and also the logical operations **AND INVERT OR XOR** .

In addition the following words are provided with obvious meaning: **2+ 2- M-UM+** .

Integers and addresses are indistinguishable in Forth. IFORTH provides the following standard words for address arithmetic: **ALIGNED FALIGNED DFALIGNED SFALIGNED CELL+ CELL/MOD CELLS CHAR+ CHARS FLOATS FLOAT+ DFLOATS DFLOAT+ SFLOATS SFLOAT+** .

In addition the following words are provided, similar to **CELL+ CHAR+** , but decrementing the address: **CELL- CHAR-** .

The IFORTH words **[]CELL CELL[] []DOUBLE DOUBLE[] []FLOAT** and **FLOAT[]** help in addressing arrays of cells, doubles and floats. Each accept the index followed by the address of the array (or in reverse order) and return the address of the element. Using them enhances readability as well as portability.

IFORTH provides the following standard words for integer division and multiplication: *** */ */MOD / /MOD 2* 2/ D2* D2/ FM/MOD M* M*/ MOD SM/REM UM* UM/MOD** . The words **2* 2/** are shift operators that could be

used for division and multiplication of unsigned numbers. See the section about shifting operators.

In IFORTH symmetric division is the standard. This means that all division operators are compatible with **SM/REM** except, of course, **FM/MOD** .

There is a way to switch the system to floored division. See the chapter about customization.

5.2.3. The integer comparison group

IFORTH provides the following standard words for integer comparison: 0< 0<> 0> 0= < <> = > D0< D0= D< D= DU< U< U> WITHIN .

In addition the following words are provided with obvious meaning: **<= >= D>** , thus providing a more complete set of operators. Note that the use of **<>** is more readable than the customary use of **-** , and it is safer because a pure boolean flag is sometimes required.

5.2.4. Shift and rotate operators

Shift and rotate operators work on integers that are considered as bit sets. The shift operators available in ANS Forth are **RSHIFT LSHIFT 2/ 2*** and all of them work on unsigned numbers, except for **2/** . **RSHIFT LSHIFT** perform left and right logical shifts over a variable number of bits. **2*** performs a single bit shift left. **2/** performs a arithmetical single bit shift right.

IFORTH also provides double precision version of these operators: **DRSHIFT DLSHIFT** .

The rotate instruction **ROR** also shifts to the right, but it reinserts on the left the bits that are shifted out. Rotate left is available with the word **ROL.** A multi-bit arithmetic shift right is performed by **ASHR** . With this instruction the sign bit is preserved when shifting.

5.2.5. The floating-point arithmetic group

IFORTH provides the following standard words for floating-point arithmetic: **F* F** F+ F- F/ FABS FMAX FMIN FNEGATE FROUND FLOOR FSQRT** . The following in the way of standard transcendental functions are provided: **FACOS FALOG FASIN FATAN FATAN2 FCOS FEXP FEXPM1 FLN FLNP1 FLOG FSIN FSINCOS FTAN** . In addition the following transcendental functions are provided: **FACOSH FASINH FATANH FCOSH FSINH FTANH** . They are the hyperbolic versions of the standard functions without the suffix ‘H’.

In addition the following words are provided: **FSQR FDEG FRAD FSPLIT F+! REDUCE.2PI REDUCE.PI** .

FDEG converts an angle in radians to degrees and **FRAD** converts the other way around.

FSPLIT splits the number on the stack in a mantissa on the floating-point stack and an exponent with respect to base 2 on the data stack. **F+!** behaves similar to **+!** but beware that the address and data are on different stacks.

Two words reduce the number on the stack to a given range, i.e. multiples of the range are subtracted or added until the number fits in the interval -range/2 to

+range/2. **REDUCE.2PI** reduces with respect to the range **PI*2** and **REDUCE.PI** with respect to **PI** .

5.2.6. The floating-point comparison group

IFORTH provides the following standard words for floating-point comparison: **F~ F< F>** . In addition the following words are provided with obvious meaning: **F0< F0<> F0= F0>** .

(NOTFIN) tests whether the number on stack is a regular floating-point number, i.e. it should not be infinite or not a number. **(ISNAN)** returns a flag whether the number on stack is not a number.

5.2.7. Integer and floating constants

IFORTH provides the following standard words that put an integer constant on the stack: **BL FALSE TRUE** .

In addition IFORTH provides the following floating-point constants: **PI PI*2 PI/2 PI/4 LN2 LN10** , where the latter constants are the natural logarithm of 2 and 10.

Also present are the following IEEE floating-point "constants" **+INF +NAN -INF -NAN** meaning "plus infinity", "positive not a number", "minus infinity" and "negative not a number". These originate whenever a floating-point operation cannot generate a valid result, for instance because it is mathematically undefined, or the mathematical correct result cannot be represented within the available range. None of these constants are in the ANS standard. They are only "floating constants" by virtue of the fact that IFORTH handles them as possible values for a definition by **FCONSTANT FVARIABLE** . When these constants are passed to one of the floating-point formatter words they are handled as special cases, e.g. **-NAN FE.** prints the text "-NAN".

5.2.8. The conversions

IFORTH provides the following standard words for (partially) converting arithmetic values among themselves and from strings: **>NUMBER >FLOAT D>F D>S F>D S>D** .

Conversion from numbers to strings is called formatting and is treated in separate sections.

The special word **NUMBER?** tries to convert a string to a single, double or floating-point number. It also returns type information. In addition the following words are provided with obvious meaning: **F>S S>F** . The word **U>D** converts a single precision unsigned to a double number.

5.2.9. Fetch and store

IFORTH provides the following standard words for fetching and storing of different integer types: **@ 2@ C@ ! 2! C! +! COUNT** .

Also the words **C+! D+!** are provided with meanings similar to **+!** but working on the content of a character respectively double rather than a cell. The words **D@ D!** function similar to **@** and **!** but on doubles. They were in dpANS-3 but were removed later on (by accident).

The words **ON OFF** are used to store respectively **TRUE FALSE** into cells. They accept the address of the flag on the stack and leave nothing.

The word **TO** may be used for storing data in "values" as well as local and register variables. The IFORTH implementation is elaborate and warrants a description in a separate section.

Whenever **COUNT** is used as a character fetch with auto-increment, I recommend to use **C@+** instead. A full set of these words is provided with a meaning similar to **C@+** : **@+ @- C@+ C@-** .

Also provided are the following standard words for fetching and storing floating-point types: **F@ F!** .

A word with a meaning similar to **C@+** is provided for floating-point numbers: **F@+** . In addition IFORTH supplies **F0!** that stores floating constant 0 at the address on the data stack.

Also the standard words for storing as IEEE floats are provided: **DF! DF@ SF! SF@** . No ambiguities can arise during conversion because the processor uses the IEEE floats internally. The reduction of precision is done with rounding and a number that is too large will be converted to infinity first.

5.2.10. The TO-concept as an object-like paradigm

The TO-concept in IFORTH is best understood in terms of an object oriented approach. The words **FROM TO 'OF CLEAR +TO 0TO SIZEOF /OF** generate messages that are understood by words of certain types that I will call TO-objects. So in general you will see a message followed by a TO-object that accepts and destroys that message. The messages have the meaning shown in Textbox 2.

Keyword	Description
FROM	Put your content on the stack
TO	Store stack data into yourself
0TO or **CLEAR**	Initialize yourself
+TO	Add the top of the stack to yourself
'OF	Put address of content on stack
SIZEOF	Put size of content on stack
/OF	Put number of elements on stack

2 TO messages

These messages are vague because they are so general. See the separate subsections about the different objects for more specific information.

5.2.11. Terminal output of strings

The basic output word **TYPE** can be used to print a string to the screen. If the string length is still hidden in the first character of the string, **COUNT TYPE** can be used to print to the screen, which may be abbreviated to **.$** .

5.2.12. Formatting and terminal output of integers

Note that all words described in this section adhere to the convention that if they contain a "." (full stop) they perform terminal output. The only exceptions are the words within parentheses like **(D.)** that write to a string.

IFORTH provides the following standard words for formatting integer types: **#S #**

<# #> ? . .R BASE D. D.R DECIMAL HEX HOLD SIGN U. U.R .

In addition to the standard words, the words **OCTAL** and **BINARY** switch the input and output to the octal and binary number systems.

H. prints the number on the stack in an unsigned hexadecimal format (8 digits with leading zeroes and preceded by '$' (dollar sign)), regardless of the current number base in effect.

DEC. UDEC. print the number on the stack in decimal, respectively unsigned decimal (without a leading '#') regardless of the current number base in effect. It uses a variable length format. **UD. UD.R** are the unsigned equivalents of **D. D.R** respectively.

The word **(D.)** leaves the address of a counted string, that contains what would have been printed by **D.**

5.2.13. Formatting floating-point numbers

Note that all words described in this section adhere to the convention that if they contain a dot they perform terminal output. The only exceptions are the words within parentheses like **(E.)** that write to a string. IFORTH provides the following standard words for formatting floating-point numbers: **FE. FS. F. REPRESENT PRECISION SET-PRECISION** .

The dot-words recognize the special IEEE number representations and print them.

In addition the following words are provided: **(E.) (F.) E. F. E.R F.R** . These words are similar to **D.** and **D.R** in that the number is right adjusted in a field with a width such as is specified in the top of the data stack. The 'E' words are similar to **FS.** , the 'F' words try to write the number without using an exponent (e.g. 10000 instead of 1E4). Variant words like **(F.R)** leave a temporary string instead of typing it directly.

Five user variables determine the formatting of reals.

The user variable **FMSIGN** determines the mantissa sign. Its default value is 0, which means that it suppresses a positive mantissa sign. If **FMSIGN** contains a 1, the sign for a positive mantissa is printed as a '+'. If it is 2, the sign for positive mantissa is printed as a blank. The sign of a negative mantissa is always printed, using a '-' character.

The user variable **FESIGN** determines the representation of the sign of the exponent of a floating-point number. Its default value is 0, which means that it suppresses the printing of the sign for positive numbers. In all other cases the sign is printed as a '+'. The sign of a negative exponent is always printed, using a '-' character.

The user variable **FDEC** contains the character that is used as a floating point. Default it is '.' (full stop). **>FLOAT** will interpret this character correctly on input.

The user variable **FECHAR** contains the character that separates the mantissa from the exponent. Default it is 'E'. **>FLOAT** will interpret this character

correctly on input.

The user variable **FELEN** determines the minimum number of digits used in the exponent. Its value is 2 by default.

The set of user variables **PRECISION FECHAR FDEC FELEN FESIGN FMSIGN** is saved on the stack using **SAVE-FFORMAT** and restored from the stack using **RESTORE-FFORMAT** .

FSIGN is similar to **SIGN** but adds the sign of a floating-point number to the result string, subject to customization by **FMSIGN** . The input of floating point numbers by the default system handler **NUMBER?** is affected by setting any of these customization variables.

Note again that the words containing a dot perform terminal output, except those within parentheses. Also note that **FDEC FECHAR** are to be used with care because you can produce numbers that cannot be easily read by other standard systems or programs.

5.2.14. Parsing the input stream

IFORTH provides the following standard words for parsing the input stream: **>IN #TIB TIB SOURCE \ (TIB PARSE QUERY ACCEPT REFILL WORD** .

5.2.15. Compiling numbers, chars and strings

IFORTH provides the following standard words for compiling double precision, floating-point and single precision numbers once they are on the stack: **2LITERAL FLITERAL LITERAL** . The generation of these numbers from a string is in a separate section about formatting.

Conventional Forth compiles the number as a literal but, depending on the type, prefixes it with one of the following (non-standard) words: **(2LITERAL) (FLITERAL)** or **(LITERAL)** . This is done in such a way that during execution the correct literal is put on the stack. In IFORTH the extra call to **(LITERAL)** or **(2LITERAL)** is unnecessary as machine code can be compiled which pushes integers on the data stack directly. Unfortunately this is not possible for floating-point literals. An instruction to push a general floating-point constant on the internal stack is missing, and for in-line floating-point constants the word **(FLITERAL)** is needed.

The words **ILITERAL ALITERAL** are equivalent to **LITERAL** . However, they have certain advantages with respect to optimization and relocatable code. More information is available in the glossary. An example of their use can be found in the utility *arrays.frt* in the *include* directory.

The following standard words for compiling or interpreting strings are also provided: **C" S" SLITERAL** . During interpretation, the strings generated are allocated in 4 static buffers that are cyclically reused. As a consequence no more than 4 strings can be "pending". If you need more, you will have to store in buffers of your own.

The word **,"** is similar to **S"** but it also places the string in the dictionary. It parses characters delimited by "" (double quote) and allocates the packed string at **HERE** . The dictionary pointer is adjusted to point just after the last

character.

The following standard words for compiling or interpreting characters are also provided: **[CHAR] CHAR** .

IFORTH also provides the similar words for control characters: **[CTRL] CTRL** . This generates a byte with the value ^X where X is the character following in the input stream. Again **[CTRL]** is used while compiling, **CTRL** while interpreting.

5.2.16. Building data structures

IFORTH provides the following standard words for building data structures of a certain type: **CONSTANT 2CONSTANT FCONSTANT VARIABLE 2VARIABLE FVARIABLE VALUE DVALUE FVALUE** . The data structure gets its name from the input stream. The value types have a richer repertoire than the standard requires, so they are described in separate sections.

The following standard words are also provided: **CREATE DOES> ;CODE** . They are general tools to extend IFORTH with new data structures.

The following standard words that help to generate data structures in the dictionary are also provided: **, ALIGN ALIGNED FALIGN FALIGNED SFALIGN SFALIGNED DFALIGN DFALIGNED ALLOT C, HERE UNUSED** .

In addition the following word is provided that compiles a floating-point constant in the dictionary: **F,** .

5.2.17. Local data structures

ANS Forth fixes a particular syntax for local data structures with the word **LOCALS|** . This is necessary because providing only the building block type word **(LOCAL)** may lead to as many syntaxes for locals as there are Forth users. Unfortunately, **LOCALS|** has arbitrary restrictions that are not necessary in IFORTH. That is why I provide the more flexible **LOCAL** and **DLOCAL** words. To deal with floating-point, a similar **(FLOCAL) FLOCAL** mechanism is available (there are no floating-point type locals in ANS Forth).

Few of the restrictions that are mentioned in the ANS Forth document apply. In particular it is not necessary to use **END-LOCAL** . Also it is allowed to execute code between invocations of **LOCAL** in the same definition. See also the section on LOCAL objects.

5.2.18. Program structures

IFORTH has built-in compiler security. So you will have to adhere strictly to what is prescribed here, unless you turn the compiler security off. In particular a programming construct can be substituted within some other construct. If the outer construct is closed with the inner construct not completed, the compiler will flag an error.

Program structure words in general have no run time stack effect, except for the words **IF WHILE REPEAT** that consume a flag intended for influencing program control flow. Flags are supposed to be true if the cell on the stack is non-zero. As the branching and security information is kept on the data stack,

program structure words affect this stack during compile time. In addition to the data stack, sometimes the system and local stacks are needed during compilation. Although this may seem terribly complicated, I guarantee that the compiler extension examples given in the ANS document work, be it that compiler security must be switched off for most of them (with **SECURITY OFF**). Program structures can only be used within a colon definition; this implies that **IF ELSE THEN** can not be used to direct the compilation process to handle e.g. the absence of a floating-point co-processor. This useful feature is called conditional compilation and is handled in a separate section.

The words **IF ELSE THEN** (in that order) bracket a conditional structure, such as meant in the standard. In addition **ENDIF** is provided as an alias for **THEN** .

The words **BEGIN WHILE REPEAT** (in that order) bracket a mid-conditioned loop, such as meant in the standard. Furthermore in IFORTH it is also allowed to replicate the **WHILE** with the same effect, i.e. all of them will test a flag and will transfer control to after the **REPEATED** (instead of **REPEAT**) if the flag is not true.

The words **BEGIN AGAIN** (in that order) bracket a infinite loop such as meant in the standard.

The words **BEGIN UNTIL** (in that order) bracket an end-conditioned loop such as meant in the standard. IFORTH allows a **BEGIN WHILE WHILE** ... **?REPEATED** structure. This construct can be built with standard words (**BEGIN WHILE WHILE** ... **UNTIL THEN THEN**) but then lacks compiler security.

The words **DO LOOP** or **DO +LOOP** (in that order) bracket a counted loop such as meant in the standard. In these loops **DO** may be replaced with **?DO** that will loop zero times for equal bounds instead of the wrap-around looping of **DO** . With the words **I J** access is provided to the loop counter values. In addition to the standard, **K** is provided to access the outermost loop parameter in a triply nested loop.

The words **CASE OF ENDOF ENDCASE** can be used to select one from a number of actions. It looks like

```
CASE
     A OF <ACTION-A> ENDOF
     B OF <ACTION-B> ENDOF
        ...
        ( DUP) ABORT" unknown case"
ENDCASE
```

If the top stack item is A, <ACTION-A> is executed. If the top stack item is B, <ACTION-B> is executed. **ENDCASE** drops the item from the stack. The number of **OF** ... **ENDOF** lines is unlimited.

The words **FOR NEXT** bracket a simple down counting loop. IFORTH provides **UNNEXT** as an immediate exit from such a loop (cf. **UNLOOP**), and **I@** that leaves the back counting index of such a loop on the stack (cf. **I**). To prevent

looping once when a zero argument is supplied, use the construct **FOR AFT** ... **THEN NEXT** or **?FOR** ... **NEXT.**

The standard describes control words in terms of a control stack containing addresses where to jump to or where destinations of jumps have to be filled in.

The following standard words for direct manipulation of this control stack are also provided: **CS-PICK CS-ROLL** . You may use these words to define your own program control structures. However, in doing so you may have to turn off compiler security or extend it to your new structures.

5.2.19. Conditional compilation

The words **[IF] [ELSE] [THEN]** (in that order) bracket a conditional compilation, such as meant in the standard. This is very useful for situations where a program has to run in different configurations. Note that **[IF]** uses a flag at compile time, influencing the compilation of a word. So you will often see it in colon definitions preceded by a word between square brackets (e.g. **[LOCALS?]**) to put this condition on stack.

5.2.20. Word-lists (vocabularies)

IFORTH provides the following standard words concerning word-lists, formerly called vocabularies: **EDITOR ONLY ORDER ALSO ASSEMBLER DEFINITIONS FORTH FORTH-WORDLIST GET-CURRENT GET-ORDER PREVIOUS SEARCH-WORDLIST SET-CURRENT SET-ORDER WORDLIST** . As no standard word exists to create a named **WORDLIST** , IFORTH provides **VOCABULARY** . A **VOCABULARY**-type word will change the search order when it is executed.

5.2.21. Colon definitions and execution tokens

The colon definition is Forth's way of defining new programs in terms of existing ones. A colon definition may be characterized in two ways, by its name (a string) and, more compactly, by its execution token.

IFORTH provides the following standard words to create a colon definition: **: ; IMMEDIATE :NONAME** .

The following standard words to transfer program control from within a colon definition to outside it are also provided: **ABORT ABORT" CATCH EXIT QUIT RECURSE THROW** .

The following standard words to handle execution tokens are also provided: **' ['] >BODY EXECUTE FIND** .

In addition to the standard the word **BODY>** is provided, that is the inverse operation of **>BODY** .

The following standard words are felt to belong in this group, but it is hard to characterize them: **] [EVALUATE POSTPONE STATE** .

In addition to the standard the following words are provided: **HEAD' ?DEF ?UNDEF HIDE** .

An other advanced header manipulation is to add a forget action to a named definition. **IS-FORGET** accepts an execution token and parses name delimited

by blanks. It tries to find the dictionary header pertinent to the name, issues an error message if it was not found, otherwise fills the forget field of dictionary header with the execution token on the stack. When forgetting the word the data field address is put on the stack prior to executing the forget action.

5.2.22. Smart data structures

A particular type of colon definition is used to extend IFORTH with smart data structures. To this purpose, IFORTH provides the following words: **CREATE DOES> FORGET>** . They are used in the colon definition of a new defining word, where the invocation of this new word generates an instance of a smart data structure, featuring data as well as function.

It is defined as follows:

```
: <generic-word>
   <anything>
        CREATE    <builds-part>
        FORGET>   <forget-part>
        DOES>     <does-part>
   ;
```

Everything from the body except the **DOES>** line is optional. It is used as: <generic-word> <NEW-WORD> . **CREATE** generates a new header; it is responsible for parsing the name <NEW-WORD>, when <generic-word> is executed. The <builds-part> is supposed to allocate some data structure in the dictionary. The <does-part> is some code that is executed by invoking <NEW-WORD>; it gets the address of the data structure that was allocated to start with. The <forget-part> is executed when <NEW-WORD> is forgotten.

In the <builds-part> typically also header modifying words like **IMMEDIATE** and **PRIVATE** find their place.

The <does-part> may also be defined in assembler by replacing **DOES>** <does-part> **;** by **;CODE** <does-part> **END-CODE** .

5.2.23. Terminal I/O

IFORTH provides the following standard words for terminal output: **." .(CR EMIT EMIT? SPACE SPACES PAGE TYPE** . The following standard words for terminal input are also provided: **ACCEPT AT-XY KEY KEY?** .

The standard words **EKEY EKEY?** are implemented in the following way: **EKEY** will correctly report any key that can be input in the environment (OS dependent). This encompasses all straight keys combined with at most one of CTRL, ALT or SHIFT pressed at the same time. This is much more than the 95 keys in the ISO set required by ANS Forth. In fact, it is even much more than can be reported in a byte. The so-called “extended” keys of the PC (see any book on PC hardware) are reported as a 16-bit entity where the lower byte is always zero. For example the function key F1 is reported as $3B00.

IFORTH does not implement the complicated decoupling between **KEY KEY? EKEY** and **EKEY?** given as an example in the ANS document. If **KEY?** tests the keyboard and an extended key is pressed (e.g. F1), **KEY?** can be used to reports

this. If one subsequently uses **KEY** to fetch the character, it will return an ASCII NUL. If **EKEY** is used (which is not a very logical thing to do, given the previous **KEY?**) it returns $3B00.

Note that **EKEY>CHAR** can be used to filter the output from **EKEY** so that it becomes equivalent to **KEY** .

An example application of the word **WAIT?** is enabling the user to interrupt the execution of **WORDS** . It is intended to be called in a loop, and returns **TRUE** if the user wants to stop, **FALSE** otherwise. The user signals stopping by pressing <ESC>. If the user presses any key other than <ESC>, **WAIT?** waits for the next key press before returning.

BREAK? is slightly simpler. It waits for a key and returns **TRUE** if the user presses <ESC> to signal stopping, otherwise **FALSE** . The word **CLS** clears the terminal screen.

5.2.24. Strings and characters

IFORTH provides the following standard words for string handling: **-TRAILING /STRING BLANK CMOVE CMOVE> COMPARE FILL PAD SEARCH SLITERAL** .

Note the difference between strings of characters and address units.

In addition the following word is provided: **PACK** . It accepts a string in the format <addres,count>, and the address of a place where a counted string can be stored. It appends the constant string to it and returns the address of the counted string. If it did not leave the address of the counted string, it could be considered a string store operation, similar to **!** .

>UPC converts the character on the stack to upper case. If it was not lower case, it is not changed.

>GRAPHIC converts the character on the stack to a printable character. If it was a non-printable character, it is changed into '.' (full stop).

5.2.25. File handling and input/output

IFORTH provides the following standard words for operations on files identified by a character string: **CREATE-FILE DELETE-FILE OPEN-FILE RENAME-FILE** .

The following standard words for operations on files identified by a "fileid" such as meant in the standard are also provided: **CLOSE-FILE FILE-POSITION FILE-SIZE FILE-STATUS REPOSITION-FILE RESIZE-FILE** .

The following standard words for read and write operations on files identified by a fileid are also provided: **FLUSH-FILE READ-FILE READ-LINE WRITE-FILE WRITE-LINE** .

The following standard words for specification of access methods are also provided: **R/O R/W W/O BIN UNBUFFERED**. If the **BIN** access word is missing, the file is opened as a text file. This means that IFORTH guarantees that when **WRITE-LINE** is used the resulting file is a text file in the sense of the host operating system and will be compatible with host text editors and such.

The following standard words for nested compilation from files are also provided: **INCLUDE-FILE INCLUDED RESTORE-INPUT SOURCE-ID SAVE-INPUT** .

Note that some file manipulations are only interesting for the users of blocks. Those are not treated here.

5.2.26. Memory management

IFORTH provides the following standard words for memory management: **UNUSED ALLOCATE FREE RESIZE**. A useful non-standard word is **AVAILABLE**. The word **?ALLOCATE** is a general error handler that acts on the error codes returned by **ALLOCATE FREE** and **RESIZE** .

5.2.27.Vectored execution

An ANS Forth does not have any words that refer to vectored execution. The addition of vectored execution to Forth is, of course, simple enough. It is available in the source *miscutil.frt* in the include directory.

5.2.28. IFORTH sets

In IFORTH a very simple set mechanism is present. A set is a number of cells, the first one containing the cardinality (number of elements) of the set, followed by cells containing the elements. The dictionary mechanism is sufficient to generate a set, e.g. by **CREATE** SOME-SET 3 **,** A **,** B **,** C **,** . The word **INSET?** accepts an element and a set and looks up the element. It returns a flag whether the element is present in the set.

5.2.29. Programmer conveniences

To keep the system under control, IFORTH provides the following standard words to help the programmer: **BYE COLD DUMP ENVIRONMENT? SEE** .

These words are system-dependent and need a more detailed explanation, so they were explained earlier in chapter 4.

Please note that **SEE** is only available after loading *see.frt*, so it is a loadable extension.

5.2.30. Miscellaneous

IFORTH provides the following standard words that cannot be well categorized: **ERASE** and **MOVE** .

5.2.31. Time and date

The standard words **MS** and **TIME&DATE** are provided.

In addition the words **TIME DATE** provide the time respectively date information only, in the same stack order.

Moreover **'TIME$ 'DATE$** make available the information printed by those commands, as a counted string, in the formats 23:13:00 and January 3, 2001. Note that these strings are stored in a string pool that is also used for other purposes, so they have a limited life span. If you copy the data immediately there is no problem.

Finally **.TIME .DATE** and **.TIME$** print the strings immediately.

5.2.32. Trespassing into the BLOCK world

IFORTH provides the following standard words that handle the classical Forth block editing system: **BLK BLOCK BUFFER EMPTY-BUFFERS FLUSH LIST LOAD SAVE-BUFFERS SCR THRU UPDATE** . IFORTH also provides the non-standard word **BLOCK-FID** .

In addition IFORTH provides the words **USE USE-BLOCKS** . They allow to specify the file in which the blocks reside. **USE** accepts the name as an inline string; **USE-BLOCKS** accepts the name as a counted string on the stack. These words are similar to **INCLUDE INCLUDE-FILE** .

A simple public domain block-editor is provided in the file *editor.frt* in the block subdirectory. This is supplied as is and without warranty. IFORTH fully implements both the BLOCK Word Set and the BLOCK EXT Word Set. This does not mean their use is encouraged.

There is a problem with blocks for systems with hardware multi-tasking, and that is the guaranteed time a buffer or block stays available to a process. Conventional Forth systems implement multi-tasking by having a round-robin loop of tasks. When a task gets control, it can only be descheduled when it executes certain clearly defined Forth words. This word set has now been formally described in the Standard. The set contains all words that can be expected to cause an I/O operation to take place, e.g. **EMIT** and **TYPE** .

Given this model, check out the following code for for a multi-programmed Forth environment:

```
   : DUMP-BLOCK ( n -- )
( 1 )    BLOCK DUP           \ get address of block n
( 2 )    512 TYPE CR         \ type upper 512 bytes
( 3 )    512 + 512 TYPE CR   \ type lower 512 bytes
     ;
```

This doesn't work because the word **TYPE** causes descheduling. It is possible that the block address obtained at (1) is invalid at (3). A simple solution is to again execute **BLOCK** at line (3).

When hardware descheduling is fully automatic, as for the transputer, the conventional model breaks down. Descheduling can even take place in the words **DUP** and in the code generated for the literal 512.

I rejected the option to design IFORTH in a way that would have made hardware descheduling impossible. An possible solution is to provide for only one buffer and give the user an explicit semaphore for it. The example above then reads:

```
   : DUMP-BLOCK ( n -- )
     buf GETSEMAPHORE
         BLOCK DUP           \ get address of block n
         512 TYPE CR         \ type upper 512 bytes
         512 + 512 TYPE CR   \ type lower 512 bytes

     buf FREESEMAPHORE ;
```

Now the buffer belongs to the process executing **DUMP-BLOCK** and all is well. Except for the fact that during the **TYPE** 's other processes could have made good use of the buffer. Copying the block to a local buffer solves this remaining inefficiency:

```
: DUMP-BLOCK ( n -- )
   buf GETSEMAPHORE
   BLOCK PAD 1024 CMOVE      \ copy block to a local buffer
   buf FREESEMAPHORE         \ release it for others.

   PAD 512 TYPE CR           \ type upper 512 bytes
   PAD 512 + 512 TYPE CR     \ type lower 512 bytes
   ;
```

For one buffer the complexity is manageable, but for more buffers difficulties arise.

As the ANS Forth documents states that "Manipulation of semaphores for Standard words such as **BLOCK** must be incorporated automatically in the driver-level code" I cannot give you the buf semaphore above. (Or the semaphores when more buffers are to be supported). I finally decided to implement the scheme above, with only one buffer, and to hide the semaphore in the system words.

In iForth's implementation, **BUFFER** requests the buffer semaphore. All the words specified by the standard release it. Note that we need a special semaphore to do this, as a process that simply types a character will now try to release a semaphore for a buffer it has not requested first.

Furthermore the **ABORT** code of all processes contains action to release the buffer semaphore, in order to prevent a system crash when a single process hangs.

The set-up described works well in practice. A clear disadvantage is that it is relatively hard to extend the concept to multiple buffers. However, I am sure my users can at least port some their old code over to IFORTH.

5.2.33. The obsolescent words

IFORTH provides the following obsolescent standard words: **[COMPILE] EXPECT SPAN CONVERT TIB #TIB** .

Use of these words is never needed because the standard provides some alternative way to handle their functionality.

5.3. The TO-objects

Because TO-objects play an important role, this whole subsection of the current chapter is devoted to different TO-objects. In the following the effects of all possible TO-messages to all the TO-objects are described. A minus sign in the tables indicate a message that is not allowed for the object.

The word **FROM** may always be omitted, an ANS Forth requirement.

Sometimes there is a conflict about which object is the target of a message. This can be resolved by using ((and)) .

5.3.1. General properties of TO-objects

For each type of TO-object (for example a TO-variable generated by **VALUE**) it remains to be specified exactly what the effect of the message is. So probably a TO-object of type TO-string would interpret the message **+TO** as an instruction to append a string (specified via the stack) to itself. (TO-strings are not implemented in IFORTH itself, but it is present in *strings.frt* in the `include` directory). For a **VALUE** the meaning of the messages is pretty obvious, except maybe for **/OF** . This message returns the number of discrete data elements occupied by the structure. For a simple value or dvalue this returns one, but for a TO-array it returns the number of elements in the array.

The TO-approach has two advantages. The messages may be used for several types, which limits the proliferation of words. The ANS Forth standard uses this to have the same words handle TO-variables and local variables. Also it may be much safer to use than plain fetch and store. Compare for example **FROM** A **TO** B with A **@** B **!** . Because in the TO-case B knows where to store, the catastrophes of random stores are prevented. Storing of a wrong value at a location where storing is allowed is seldom as fatal. If B turns out to be a TO-constant the first example would simply result in an error message.

ANS Forth prescribes that **FROM** may be omitted; in fact it does not even recognize it. Each time a message has been accepted a standard Forth has to generate a **FROM** message to serve as a default for the next TO-action. So **FROM** A **TO** B **FROM** C **TO** D may be abbreviated to A **TO** B C **TO** D . If **FROM** were obligatory **FROM** A **TO** B would be equivalent to **TO FROM** A B provided the messages where stacked. The "**FROM** is default" rule prevents this.

In some cases (notably arrays of values if the index needed is itself a TO-variable) this stacking can be necessary. IFORTH provides (( )) for this situation. So **TO** ((**FROM** A)) B works as intended in the example above. The **FROM** in front of A may be omitted, but the brackets are essential. The system stack is used, so beware!

Keyword	Description
FROM	Put the object on the stack
TO	Store top of stack into object
0TO or **CLEAR**	Store 0 into object
+TO	Add top of the stack to object
'OF	Address where object is stored
SIZEOF	1 Cells
/OF	1 Element

3 VALUE messages

IMPORTANT! IFORTH itself uses an even more advanced version of the TO-concept. All TO-words and the messages are immediate. Because the behavior wanted at run time is already known at compile time, the compiler can anticipate it and do much more optimization than would be possible otherwise. In object parlance IFORTH uses early binding, where more classical Forths use late binding.

The immediate and classical version of the TO-concept can even be mixed, but this is inadvisable. If you want to write your own TO-objects, you are advised to use a private message variable, rather than hook into the existing system with **%VAR**.

Keyword	Description
FROM	Put the object on the stack
TO	Store double into object
0TO or **CLEAR**	Store 0. into object
+TO	Add the double to object
'OF	Address where object is stored
SIZEOF	2 Cells
/OF	1 Element

4 DVALUE messages

5.3.2. The VALUE object

The objects generated by **VALUE** are single precision integers. IFORTH initializes the object by sending a **CLEAR** message to it, but a standard program should not rely on this. The object reacts according to table 3.

5.3.3. The DVALUE object

The objects generated by **DVALUE** are double precision integers. IFORTH initializes the object by sending a **CLEAR** message to it, but a standard program should not rely on this. They react according to table 4.

5.3.4. The FVALUE object

The objects generated by **FVALUE** are floating-point numbers. IFORTH initializes the object by sending a **CLEAR** message to it, but a standard program should not rely on this. They react according to table 5.

Keyword	Description
FROM	Put the object on the floating-point stack
TO	Store float into object
0TO or **CLEAR**	Store 0E into object
+TO	Add float to object
'OF	Address where object is stored
SIZEOF	1 Floats
/OF	1 Element

5 FVALUE messages

5.3.5. The LOCAL object

See also the section about local data structures.

The objects generated by **LOCAL** are single precision integers. The dictionary entry created is temporary. **LOCAL** can only be used inside a definition. The name of the object remains in the dictionary until the current definition is finished with **; ;CODE DOES>** .

Contrary to the other objects, local objects are created at run time. IFORTH initializes the object with the integer that is on the top of the stack at the moment of creation. This works even for recursive invocations.

The **LOCAL** objects react according to table 6.

Keyword	Description
FROM	Put the object on the stack
TO	Store integer on top of stack into object
0TO or **CLEAR**	Store 0 into object
+TO	Add the integer on top of the stack to object
'OF	-
SIZEOF	1 Cell
/OF	1 Element

6 LOCAL messages

5.3.6. The FLOCAL object

See also the section about 'local data structures'.

The objects generated by **FLOCAL** are single precision floating-point variables. The dictionary entry created is temporary. **FLOCAL** can only be used inside a definition. The name of the object remains in the dictionary until the current definition is finished with one of **; ;CODE DOES>** . Contrary to the other objects, local objects are created at run time. IFORTH initializes the object with the floating-point number that is on the top of the floating point stack at the moment of creation. This works even for recursive invocations. The **FLOCAL** objects react according to table 7.

Keyword	Description
FROM	Put the object on the floating-point stack
TO	Store float into object
0TO or **CLEAR**	Store 0E into object
+TO	Add float to object
'OF	-
SIZEOF	1 Floats
/OF	1 Element

7 FLOCAL messages

5.3.7. The REGISTER object

The objects generated by **REGISTER** are single precision integers. Contrary to all other objects the user has to specify a number from 0 to 15 to specify in which high speed variable (called "register") the object is created. This number is expected on the stack when invoking **REGISTER** .

The IFORTH system itself, nor any of the utilities in the include directory uses any of the registers.

Keyword	Description
FROM	Put the object on the stack
TO	Store integer into object
0TO or **CLEAR**	Store 0 into object
+TO	Add integer to object
'OF	Put address of object on the stack
SIZEOF	1 Cells
/OF	1 Element

8 REGISTER messages

It may seem strange but getting the address of a "register" is perfectly possible and valid. A "register" is not initialized. It reacts according to table 8.

5.4. The system words

Although it may seem a bit strange to divide Forth programming in application and system programming, some words are more "system" than others. I grouped a number of words in this sub-section because I feel that many applications do not need them.

5.4.1. System vectors

The IFORTH system uses vectored execution for almost all of the terminal interaction. As explained earlier, a vector is a variable that contains an execution token. This is useful to accomplish many things. For example by changing some execution tokens, the output of a large program can be forced to a file, without the need to change the program itself. This is possible even after the program has been compiled. Note that ANS Forth does not require the use of vectors.

The execution tokens in the vectors **'TYPE 'ACCEPT 'AT-XY 'EMIT 'EMIT? 'KEY 'KEY? 'PAGE** are executed if the corresponding unticked definitions are executed.

All input of the keyboard goes through the definitions **'KEY? 'KEY** . By installing proper definitions you can temporarily feed the IFORTH interpreter from a file.

All output to the terminal goes through the vectors **'EMIT? 'EMIT 'TYPE** . By installing proper definitions you can temporarily redirect the IFORTH output to a file.

The vector in **'ACCEPT** is also used by **EXPECT** . This is used by one of the utilities provided (*include/proced.frt*) to add a fancy history mechanism to the IFORTH command interpreter.

'PROMPT contains the execution token of the definition that is to be executed when the command line interpreter wants to have input. A classical Forth system has no prompt, as Charles Moore disliked the implication of "hurry up dummy". I thought it useful to have the first word list in the search order displayed, but you can easily suppress this by executing 0 **'PROMPT !** . The default prompt also shows that you are in compilation mode by displaying the name of the current word list surrounded by square brackets.

'TAIL contains the execution token of the definition that is to be executed when the command interpreter has finished a line. It is customary to print the text *ok*. IFORTH is no exception to this. Unlike the prompt, if a command ends by **QUIT ABORT** this message is suppressed: the command is not finished but broken off.

'BOOT contains the execution token of the definition that is executed by **COLD** after all initializations are done. A turnkey application can be build by storing the execution token of the word that starts the application in **'BOOT.** Then execute the phrase **SAVE-SYSTEM** filename.GNU to save a new executable IFORTH system. There is a *makesys.bat* file on the distribution diskettes that converts a *.GNU file to *.EXE (it assumes that you have the relevant DJGPP files in your path).

'NUMBER contains the execution token of the text interpreter **NUMBER?** that converts a string to a number. It is executed whenever IFORTH cannot find a word in the vocabulary. Normally, (if **NUMBER?** is installed) it recognizes integers, doubles and floating-point numbers. It is this word that understands all the different notations allowed in IFORTH. If a double number was converted, the user variable **DPL** contains the location of the decimal point, i.e. the number of digits after the decimal point, or -1 of there was none. In principle you could replace the content of **'NUMBER** with a different action than **NUMBER?** , however you should take care to obey its stack behavior. Be warned that this mechanism may change in future versions without notice.

5.4.2. Workspace registers and stack pointers.

IFORTH has reserved 16 memory cells in its current workspace. They can be addressed somewhat faster than a general memory location. IFORTH tries to make optimal use of these workspace registers, as I will call them throughout this section.

The words **RP@ SP@ FSP@ SSP@** give the values of stack pointers for the return stack, data stack, floating stack and system stack respectively. The value is put on the data stack, and the data stack pointer is considered before this pushing takes place.

At the addresses **RP0 SP0 FSP0** and **SSP0** the values of the stack pointers for empty stacks are saved. Executing them puts the address on the data stack.

The words **RP! SP! FSP! SSP!** reset the respective stacks to a value that is popped off the data stack. For example, **SP0 @ SP!** empties the data stack.

All stacks grow downwards and are one cell wide, except for the floating point stack if **DOUBLE-PRECISION** is **TRUE** in the environment, in that case the width is two cells. If **EXTENDED-PRECISION** is **TRUE** , the width is three cells.

5.5. Internals of the TO-object mechanism

The IFORTH TO-object mechanism consists of TO-messages, TO-building words and TO-objects. With each TO-building word (e.g. **VALUE**) you can build TO-objects of a certain class, in this case the TO-variables. Contrary to older implementations IFORTH selects the action at compile-time, and not at run-time. The consequence of this behavior is that both the TO-messages as well as the objects are immediate words.

Keyword	Value of %VAR
FROM	0
TO	1
0TO or CLEAR	2
+TO	-1
'OF	3
SIZEOF	4
/OF	5

9 TO message numbers

Messages are global and unspecific. They are accepted, or ignored, by any TO-object executed. A message is generated by storing a certain value in the variable **%VAR** . Table 9 presents an overview of the message words with the content of the message.

In the **DOES>** part of the compiling word, a **CASE** statement interprets the

messages, preferably in accordance with their informal meaning. You may of course add your own messages.

As a consequence of **FROM** being a default, any TO-object should reset **%VAR** to 0.

5.5.1. Writing your own server

Although the MS-DOS version of IFORTH does not in effect use a separate server program, newer releases (Windows) use one. Whatever the actual situation, IFORTH is written in such a way that you can always assume the existence of such a server. In that way it is possible to be completely decoupled from the hardware. Connected to this server idea is the concept of a "boot link", a single communication channel to access the server. This boot link becomes a shared resource once IFORTH multi-tasks. Multi-tasking is officially supported, all kernel code is "boot link aware" and uses semaphores and descheduling where appropriate.

The words described in this section can be used to write your own server, or an extension to the existing server. At the lowest level there is a handshaking protocol that should be of no concern to you, if you use the facilities provided here.

The elementary instructions to access the boot link are **_RX _TX {{ }}** .

{{ ... **}}** marks an area where you have exclusive access to the boot link. You should use the elementary access commands only within such a region.

_RX returns a byte from the boot link as soon as the server has one available. **_TX** sends a byte to the server.

!TERMINAL performs all communication with the host computer according to the protocol in discussed in the appendix II. It accepts a variable number of parameters followed by the integer representing the command and returns a variable number of parameters. If you extend the protocol, you may use **!TERMINAL** and resort only to the elementary boot link access for the new commands.

6. Normal customization

IFORTH allows you to extend the system itself, without having to load your environment every time you start up. This capability is provided with **SAVE-SYSTEM** <name>. (Under MS-DOS a copy of D.J. Delorie's GO32.EXE is needed).

6.1. Building a fatter Forth system

The file *iforth.prf* will be loaded automatically upon start up. It contains the utilities that you elected to have available.

If you want access to operating system commands transparently, use the import facility (see also chapter 3.4.1). You may then add the commands that you want to be imported to the file *os.prf*. You have to include the file *os.prf*.

If you want the help system available, include the file *help.frt*.
If you want the disassembler available, include the file *see.frt*.

6.2. Building a leaner Forth system

At present there is no provision to specify that you do not need e.g. the floating-point facilities, such that they are not loaded and occupy no space. On the other hand you can easily build a turn key system. Simply put all commands that are allowed in this system in some wordlist and leave the other wordlists out of the search order. Use **PRIVATES** ... **DEPRIVE** to keep all internal words of your package invisible. The use of individual words can be screened off by **HIDE** .

These measures will result in a system that has a very clean appearance towards the user. But such a system can not be seen as a safe system since with some effort the user still has access to all system facilities.

6.3. Turnkey systems

It is occasionally useful to keep the command interpreter available for the user, so he/she may add his own colon definitions and operating systems facilities. If you want to keep all Forth-like features away, you can use **'BOOT** . The word whose execution token is in **'BOOT** will be executed on startup. This can be a command interpreter that replaces the Forth command interpreter. To build a turn key system under MS-DOS a copy of the GO32.EXE file is needed.

6.4. Little is impossible...

If you have special wishes that you feel are possible with IFORTH but that you cannot accomplish with the facilities provided, I will be glad to advise and, most probably, help out.

7. Programming in assembler

Using assembly language gives ultimate control over a processor at the expense of safety and convenience. IFORTH supplies you with a full protected-mode assembler for the Intel 386/486/586 and 387 processors.

An assembler instruction in IFORTH is equal to the mnemonic as mentioned in the Intel documentation, in lower case and with a comma appended. Throughout, the Forth convention is obeyed that the instructions that assemble end in a comma. This convention extends to all auxiliary instructions that add to the code that is generated, or modify it. Furthermore, there is the convention that operands are parameters to these instructions. Operands are put on the stack in the customary <source> <destination> <count> order. However, the overal effect is that the IFORTH assembler source code looks exactly opposite to the standard Intel notation:

```
Intel               IFORTH
mov  eax, 2         2 b# -> eax  mov,
```

7.1. The low level programming model

Of course IFORTH is mapped as directly as possible on the hardware. The biggest mismatch is that a Forth needs multiple stacks, whereas Intel processors support only one stack. This problem was solved by keeping every stack pointer in a processor register. Access to a stack is done by temporarily exchanging the proper register with the hardware stack pointer.

The five IFORTH stacks are shown in Table 10.

As you can see, the floating-point stack pointer is *not* in a processor register.

A pointer to the **USER** area is memory. The **USER** area is organized like an array.

The segment registers CS, DS, ES, SS, FS and GS should be handled with care as IFORTH runs in 32-bit protected mode. This means a segment register can only contain valid selectors (see the Intel documentation). We conclude that only the EAX, EBX, EBC, EDX and EDI register are free for general use.

stack	ptr register
data	ESP
return	EBP
system	kept in user area
local	ESI
float	kept in user area
user area base	kept in memory

10 Register usage

IFORTH is subroutine-threaded and compiles to machine code, meaning that every Forth word is just a conventional subroutine. As the Forth data stack is implemented on the processor hardware stack, making a subroutine call (executing the next Forth word in the definition) puts a return address on the Forth data stack, thus obscuring the parameters for the next Forth word. This problem is handled differently in **CODE** words and colon definitions.

- On entry of a **CODE** word, the top of the hardware stack (the return address) is popped to EBX by a “hidden” prelude. The user code then has two options:

1) save EBX, preferably on the Forth return stack with the *rpush,* macro, do something, perform the *rpop,* macro followed by *ebx jmp,*
2) simply don't use EBX until it is time to return, then do *ebx jmp,*

- On entry to a colon definition a hidden prelude performs *ebx pop*, *rpush,* . The high level **NEXT** code is simply *rpop, ebx jmp,* .
- The run-time code for **DOES>** is nothing (the proper pfa is on the data stack already).

A disadvantage of this technique is that it makes subroutine calls slower because the return address must be explicitly saved in the prelude for every high level Forth word. Despite this disadvantage I decided to use the call mechanism as the threading method for IFORTH. The decrease in speed becomes unimportant once the time taken for these extra instructions is negligible compared to the time the high-level Forth word needs to execute. This can be assured by compiling selected words in-line instead of calling them. The trade-off here is the resulting size of the compiled code. This technique is used extensively in the IFORTH kernel.

Note that IFORTH optimizes the code whenever it is given the opportunity. One of the things it does is expanding simple words like **DUP** or **OVER** inline while checking if pushes and pops of the data stack can be eliminated. The strategies followed are not obvious and using **SEE** on the result may be quite confusing. Another feature of IFORTH is tail-recursion removal. This means that

```
: bar  'A' EMIT ;
: foo  ..... bar ;
```

is not translated to

```
....

bar d# call,
rpop,
ebx jmp,
```

but to

```
....
bar d# jmp,
```

Certain popular Forth tricks like

CREATE jumptable] EMIT TYPE PAGE READ-LINE [\ addresses (??)

won't work as expected with IFORTH. (The ANS Forth standard does not allow it anyway). For this special, quite useful, case the words **EXEC: EXEC;** are provided in /include/miscutil.

Finally, do never use something like

```
7 CONSTANT fubar
: RATS    fubar EMIT ;
RATS  CHAR ! fubar >BODY ! RATS
```

IFORTH will optimize the access to fubar in RATS, compiling an in-line literal. Your changing of fubar will not work. Why not change fubar to a **VALUE** if you need the special behavior?

The optimization IFORTH performs would not have been possible with previous Forth standards. But now it is.

7.2. Customizing assembly

The behavior of the assembler can be adjusted in a number of ways, apart from the adjustments made to the compiler in general. **AWARNING** is a flag that, when true, gives warning when assembler instructions used are not available on the current target cpu as stored in the variable **#CPU** . This variable is initialized with the device number of the processor you are currently running on. If you are using smart macro's, optimal code will be generated for that cpu. In this way you can write largely CPU-independent assembler code. When cross assembling you must properly fill in this variable.

Internally the assembler uses a number of tables to look up capabilities of different processors. These are organized as predefined sets, compatible with the **INSET?** lookup word.

7.3. How to learn about the hardware

It is neither possible nor necessary to give a complete list of the instructions that are accepted by the IFORTH assembler. You need the *ix86 Microprocessor Programmer's Manual*.

When very low-level access to BIOS or DOS is wanted you must realize that IFORTH is running in protected mode, with the aid of the D.J. Delorie's Dos Extender (djgpp, go32.exe). This dos extender does not allow arbitrary call's to be made. Especially BIOS call's that (temporarily) modify segment registers will halt the machine. Furthermore, it was found that the extender program sometimes uses the EBP register for its own purposes.

You may be a bit disappointed about the slow file, screen and graphics I/O of IFORTH for DOS. This is caused by non-optimal performance of GO32. The advantage of using only standard features was that IFORTH could easily be ported to 32-bit operating systems like Linux and Windows.

7.4. Invoking the assembler

The assembler is invoked by the **CODE** ... **END-CODE** sequence. The assembler instructions that may be filled in at the ellipses add code to the code space, which is in fact the same space as is used by the **COMPILE,** instruction. Assembler instructions are, in Forth fashion, interpreted, i.e. **STATE** is **FALSE** while assembling. That means that the data for such instructions that accept an address or data can be generated by the full power of IFORTH. For example you may invoke a Forth word to calculate the address as follows: 3 78 + d# eax mov, . This is also the reason that in the glossary assembler words have an assembly and an execution behavior, whereas compiling words have a compilation and an execution behavior. With the execution behavior of an assembler word we mean the effect of the code that is added to the word that is being defined. The

assembly time behavior is the addition of the code to the code space.

The assembler contains very many words. They are put away into a separate word-list, called **ASSEMBLER** . **CODE** automatically adds this word-list in front of the word-lists to be searched, and **END-CODE** removes this word-list from the search order. **CODE** also adds a few bytes of code that are always needed, as explained in section 7.1 and 7.6.1.

7.5. The memory model

A designated amount of memory is taken over by Forth. About 7 MB is reserved to load the kernel and stacks and to allow new definitions to be added. The rest is used as a heap from which chunks can be allocated.

7.6. The Forth assembler interface

Any IFORTH word, thus also words generated by the assembler, are invoked by the call instruction.

7.6.1. Calling convention

In calling a word all registers are lost.

Because IFORTH uses the hardware stack as its data stack, an assembler word has to start with ebx pop, in order to expose the passed parameters. This instruction code is assembled by **CODE** automatically.

IFORTH words do not require that registers are restored at return.

IFORTH name	Meaning
FFSP	offset for floating-point stack pointer
FTL	offset for task link pointer
FCODE	offset for base of code area
FNAME	offset for base of data area
pr0	scratch register 0
pr1	scratch register 1
pr2	scratch register 2
pr3	scratch register 3

11 Stack pointers and registers

Important: Where the user code starts, the EBX register contains the return address, so a ebx jmp, instruction suffices to return. Of course, if the content of register EBX does not remain valid, it has to be saved. By convention this is done with *rpush,* .

7.6.2. Using the workspace from assembly

The important IFORTH registers such as stack pointers are located at a favorable place in the workspace or in registers.

For all variables symbolic names are provided. The actual value of the offsets may vary with the version of IFORTH you are using.

Table 11 shows which offsets are used in the workspace (among others).

See also the section “The system words” of chapter 5.

7.7. Macro’s and macro groups

A macro is a Forth definition that groups instructions that occur together often.

It is also possible that a macro accomplishes things that are more involved, especially if it has arguments, for instance a macro can include repetitions or decisions, or do some processing on the arguments.

A clever use of macro's can make a program much more readable and also much safer.

7.7.1. Comparison macro's

=, <>, test for "equal" and "not equal" respectively.

<=, <, >=, >, check the signed comparisons "less or equal", "strictly less than", "greater or equal" and "strictly greater than" respectively.

u<=, u<, u>=, u>, check the unsigned comparisons "less or equal", "strictly less than", "greater or equal" and "strictly greater than" respectively.

0<, 0>=, 0<>, 0=, check the comparison "signed less than zero", "signed greater or equal to zero", "unequal to zero" and "equal to zero", respectively.

ov, nov, check whether the last operation overflowed or not, respectively.

7.7.2. Mobility macro's

Mobility macro's perform data transfers between the internal registers and the IFORTH stacks. This is more convenient than using the stack pointers that are available to assembly processes.

Mobility macro's follow a naming convention. They consists of an action, the quantity and the stack involved. Sometimes a number is appended to the stack name part.

The action is one of **pop, push, get** and **put** . Popping and pushing means popping from and pushing onto some IFORTH stack. *get* means that the data is duplicated from the IFORTH stack.

The quantity is the number of items handled. A one is default and hence omitted.

The stack is indicated by one of the prefixes: 'd' for data stack, 'l' for locals stack, 'r' for return stack, 's' for system stack and 'f' for floating point stack. Items popped from the floating-point stack are put onto the internal floating-point register stack. Items popped from all other stacks are put into the EBX register.

Not all words within this naming convention are available and most of them have side-effects you should be aware of, so they are described separately.

dpop, transfers a number from the IFORTH data stack to the EBX register. **dpush,** transfers a number to the IFORTH data stack from the EBX register.

put, replaces the topmost item of the IFORTH data stack. Its new content is the EBX register. **get,** duplicates the topmost item from the IFORTH data stack to the EBX register.

rpop, rpush, transfer a number from, respectively to, the IFORTH return stack.

lpop, lpush, transfer a number from, respectively to, the IFORTH locals stack.

spop, spush, transfer a number from, respectively to, the IFORTH system stack.

fpop, fpush, transfer a floating-point number from, respectively to, the IFORTH floating-point stack. As a side effect the content of the EAX register is lost.

fput, replaces the topmost item of the IFORTH floating-point. Its new content is popped off the internal floating-point register stack. **fget,** duplicates the topmost item from the IFORTH floating-point stack to the internal floating-point register stack.

7.7.3. Macro groups for structured program control

The macro's for structured program control have a non-trivial assembly time behavior. They use the data stack for bookkeeping. So this can interfere with user defined macro's.

The combination **BEGIN,** ... **AGAIN,** will have the effect that the code filled in at the ellipses is repeated indefinitely.

The combination **BEGIN,** ... **UNTIL,** will have the effect that the code filled in at the ellipses is repeated until **UNTIL,** detects the zero flag is set.

BEGIN, ... **WHILE,** ... **REPEAT,** is very similar to the Forth **BEGIN** ... **WHILE** ... **REPEAT** construct. The part between **BEGIN, WHILE,** is always executed. **WHILE,** will inspect the zero flag and if it is zero, control is transferred to the statement after **REPEAT,** . Otherwise the part between **WHILE, REPEAT,** is executed and control is transferred to the statement after **BEGIN,** .

IF, ... **ELSE,** ... **THEN,** or **IF,** ... **THEN,** are also similar to the Forth constructs. **IF,** will react on any of the flags set by a comparison macro and transfers control to the statement after the **ELSE,** or **THEN,** . **ELSE,** will merely transfer control to after the **THEN,** . **ENDIF,** is an alias for **THEN,** .

AHEAD, can be used to generate constructions of your own. See the detailed explanation in the glossary. You will have to understand the internal working of the other words of this section to cooperate.

Future IFORTH compilers may cause descheduling of a process when it executes an unconditional jump. This can be prevented by replacing it by an artificial conditional jump. **AHEAD, ELSE, REPEAT,** assemble unconditional jumps and hence mark points where the process could be descheduled.

This can be prevented by using conditional jumps. The words to use are **AHEAD-D, ELSE-D,** and **REPEAT-D,** . The suffix means "No Descheduling".

Last but not least I show an example that uses macro's of all of the groups. It throws away all items from the stack up till, but not including, a zero that is used as a marker:

```
BEGIN, get, 0<>, WHILE, dpop, REPEAT,
```

It probably will convince you that pre-defined macro's can make life much easier for the assembly language programmer.

I. Forth Environment

The standard word **ENVIRONMENT?** is extended to understand more strings than the standard requires. All the strings specified in the standard are included.

The following table contains the standard strings. The value type column contains the values that are common for all IFORTH implementations. (Note that this may not be correct if yours is a customized version).

Currently there is just one version of IFORTH, which runs on the i386+387 combination or on the i486 / Pentium.

The following table contains specific data:

String	Value type	Value	Interpretation
ALIGN	n	4	alignment granularity, in address units, of an aligned address.
/CHAR	n	1	size of a character in address units
/COUNTED-STRING	n	255	maximum number of characters in a counted string
/HOLD	n	80	maximum size of a pictured numeric output string in characters
/PAD	n	256	size of the scratch area pointed to by PAD, in characters
/TIB	n	256	size of the text input buffer in characters
ADDRESS-UNIT-BITS	n	8	Size of one address unit in bits
CORE	flag	true	core word set present
CORE-EXT	flag	true	core extension word set present
FACILITY	flag	true	facility word set present
FACILITY EXT	flag	true	facility extension word set present
FULL	flag	true	full compliance (i.e., not a subset)
MAX-CHAR	u	$ff	the maximum value of any character in the implementation-defined character set
MAX-D	u	$7fffffffffffffff	largest usable signed double number
MAX-N	n	$7fffffff	largest usable signed integer
MAX-U	u	$ffffffff	largest usable unsigned integer
MAX-UD	ud	$ffffffffffffffff	largest usable unsigned double number
RETURN-STACK-CELLS	n	256	maximum size of the return

			stack in cells
STACK-CELLS	n	256	maximum size of the data stack in cells
BLOCK	flag	true	block extension word set present
BLOCK-EXT	flag	true	block word set present
DOUBLE	flag	true	double number word set present
DOUBLE-EXT	flag	true	double number extension word set present
ERROR-HANDLING	flag	true	error handling word set present
ERROR-HANDLING-EXT	flag	true	error handling extension word set present
FILE	flag	true	file word set present
FILE-EXT	flag	true	file extension word set present
FLOATING	flag	true	floating-point word set present
FLOATING-EXT	flag	true	floating-point extension word set present
FLOATING-STACK	n	64	n is the maximum depth of the separate floating-point stack.
MAX-FLOAT	float	1.797E308 or ... or ...	largest usable floating-point number. maximum depends on floating point model used.
#LOCALS	n	256	maximum number of local variables in a definition
LOCALS	flag	true	locals word set present
LOCALS-EXT	flag	true	locals extension word set present
MEMORY-ALLOC	flag	true	memory-allocation word set present
MEMORY-ALLOC-EXT	flag	true	memory-allocation extension word set present
SEARCH-ORDER	flag	true	search order word set present
SEARCH-ORDER-EXT	flag	true	search order extension word set present
WORDLISTS	n	16	maximum number of word lists usable in the search order
TOOLS	flag	true	programming tools word set present
TOOLS-EXT	flag	true	programming tools extension word set present
STRING	flag	true	string word set present
STRING-EXT	flag	true	string extension word set present

The following table is specific for IFORTH:

String	Value type	Value	Interpretation
/DATA-SPACE	n	-	the maximum number of cells that an application program may attempt to **ALLOT**
DOUBLE-PRECISION	flag	-	returns the precision of the floating-point package
EMULATED	flag	-	true if the floating-point is emulated, false if hardware is used
SERVER	char	'F'[4]	returns a character that identifies the server being used.
SYSTEM-STACK-CELLS	n	256	Maximum size of the system stack in cells
IFORTH	flag	true	true if this is a iForth system
VER	n	$107	the version number of iForth

[4] The standard server identifies itself as 'C' because it is written in the C programming language to be highly portable. Other servers are:

'F': a server written in Forth which can be used for interactive testing of the server by the development team;

'N': no server at all, all functions are performed by direct system calls;

'T': a server written in **iForth** that can be used to connect a neighbour transputer to the current transputer.

II. OS Interfacing

Interfacing with the host operating system is done using the boot link. If you play by the rules, it is possible to extend the messages that are recognized by the communication mechanism over the boot link with messages of your own. This means you have to extend IFORTH itself as well as the communication program on the host, the so called server. (Under MS-DOS “boot link”, “host” and “server” are mere software abstractions and can be bypassed using assembly language definitions. See the documentation on INT10() and GO32).

To issue a command via the server you need to send the server a message. A message consists of a series of bytes starting with the byte 255 (or $FF). Next comes a byte to select the command category followed by a byte to select the actual command. A number of arguments may then follow the command. After sending the command IFORTH waits for the answer given by the server.

All other bytes sent are also commands in their own right.

There are currently 3 types of arguments:

- a byte (B);
- a 4 byte word (W);
- a counted string which is prepended by its length as a word (S).

The tables below gives all commands currently available on the standard server. All commands not in this list are reserved for future use, except where noted.

Category: 0, STOP

There are no separate commands in this category. Upon receiving this command header the server cleans up and exits.

Category: 1, EXE

Num-ber	Explanation Arguments
0	Open an existing file IN S Filename IN W Open mode OUT W File number OUT W Error number
1	Execute a host OS command IN S Command text
2	Execute an interactive shell on the host machine
3	Delete a file IN S Filename
4	Truncate a file IN W File number IN W argument is ignored, must be 0 IN W File size
5	Flush buffers of one file IN W File number OUT W Error number
6	Create a new file IN S Filename IN W Open mode OUT W File number OUT W Error number
7	Close a file IN W File number OUT W Error number
8	Read a line from a file IN W File number IN W Buffer size OUT S The read line OUT W End of file flag OUT W Error number
9	Read the current time OUT W Seconds OUT W Minutes OUT W Hours
10	Read the current date OUT W Day in month OUT W Month

Num-ber	Explanation Arguments
	OUT W Year
11	Rename one file IN S Original name IN S New name OUT W Error number
12	Move the filepointer of a file IN W File number IN W Type of movement IN W Location IN W argument is ignored, must be 0 OUT W New file pointer position OUT W always 0 OUT W Error number
13	Read a fixed length string from a file IN W File number IN W requested size OUT S Read data OUT W Error number
14	Write a string into a file IN W File number IN S Data to be written OUT W Actually written OUT W Error number
15	Get server information OUT W A single character indicating the type of server used.
16	Write a line of text into a file IN W File number IN S Line to be written (without newline) OUT W Actually written OUT W Error number A newline character (or sequence) is appended to the text written.
17	Get processor info OUT W The number of this processor OUT W The number of processors in this network.
18	Read the current time and date OUT W Day in month OUT W Month OUT W Year OUT W Seconds OUT W Minutes

Number	Explanation Arguments
	OUT W Hours
19	Change the current directory IN S The new directory name OUT W Error number Under MS-DOS a single drive name given as the directory name changes the current drive.
20	Get directory name of current drive OUT W Error number OUT S Directory name
25	Transform error number to text IN W Buffer address IN W Buffer size IN W Error number OUT W Address OUT W string count OUT W Error number
26	Transform file number to file name IN W Buffer address IN W Buffer size IN W File number OUT W Buffer address OUT W string count OUT W Error number

Category: 2, TERM

Number	Explanation Arguments
0	Read one character from the keyboard OUT B The character (or 0 when none available)
1	Set the cursor to a location on the screen IN W x-position IN W y-position
2	Ask the location of the cursor on the screen OUT W x-position OUT W y-position
3	Set text attributes for the screen IN W new attributes IN W new background color IN W new foreground color

	Any combination of the following attribute bits may be used (but they are not guaranteed to work!): Bit Function 1 High intensity 2 Low intensity 3 Italics 4 Underline 5 Blinking 6 Rapid blinking 7 Reverse video 8 Invisible
4	Ask the screen width OUT W The number of columns on the screen
5	Ask the screen height OUT W The number of lines on the screen
6	Ask the number of characters on the screen OUT W The number of characters on the screen
7	No longer supported
8	No longer supported
9	No longer supported
10	No longer supported
11	No longer supported
12	No longer supported
13	Clear the screen from the current location to the end of the screen
14	Clear the screen from the current location to the end of the line
15	No longer supported
16	No longer supported
17	Test quickly whether a character is available OUT B Flag
18	Ask the text screen mode OUT B graphics, color or mono flag OUT B cursor available flag OUT B always 0 OUT B always 0
19	No longer supported
20	No longer supported
21	Write a text string to the screen IN S String
22	Open the log file (Forth.log)

23	Close the log file (Forth.log)
26	Calculate index for a given r,g,b value set. IN W red value IN W green value IN W blue value OUT W index in colour lookup table
27	Wait specified number of milliseconds IN W ticks of 1 millisecond
28	Return number of milliseconds since initialization OUT W ticks of 1 millisecond
29	Set cursor shape to OFF, CURSOR_OVERWRITE or CURSOR_INSERT IN W 0 is off, 1 = overwrite, 2 is insert

Category: 3, USER

Num-ber	Explanation Arguments
0	Draw a line of pixels IN W y start location IN W x start location IN W number of pixels IN B... Colors of the pixels
1	Start graphics mode
2	No longer supported
3	No longer supported
4	No longer supported
5	Switch to text mode
6	Plot one point IN W color IN W y position IN W x position
7	Draw a line IN W color IN W y1 location IN W x1 location IN W y2 location IN W x2 location
8	Select next graphics mode (max resolution or max colors) IN W flag
9	Discontinued

10	Set up a new palette on the host graphics card IN W Address of new palette IN W Count of colour entries in palette
11	Get the current palette of the host graphics card IN W Buffer address IN W Count of entries expected
12	Request a text font for use in graphics mode IN S name of font

Category: 4, FORTH

There are no separate commands in this category. This command has a single string as an argument. This string is executed on a Forth interpreter running on the host computer. If the server is not equipped to do so, the complete command is ignored.

Category: 5, DATA

There are no separate commands in this category. This command has a single string as an argument. This string contains data with no special encoding. This packet format can be used to transport data over links that usually only carry the server protocol. Its intended use is to concentrate data on the root (transputer) so it can be displayed or written to a file. The PC server simply ignores the data. Transputer servers may be configured to pass the data to the next transputer or to operate on the data.

The categories 6 to 127 are reserved for future use by the DFW.
The categories 128 to 254 are free for use.
You may extend the terminal interface by using these categories, provided you also modify the server on the other side. Contact the DFW for a source license of the server.

Category: 255, FF

Print the byte 255 to the screen. This character must be send as a command since it will otherwise be mistaken for a regular command.

All single byte characters are printed to the screen as characters except the following bytes:

Command byte	Explanation
7, \a or BEL	Ring the terminal bell, does not move the cursor.
8, \b or BS	Erase one character on the screen, does not go past the left of the screen, the cursor is moved one position to the left.
10, \n or NL	Move the cursor to the next line, scroll when the cursor is on the last line of the screen (column position does not change).
12, \f or FF	Clear the screen, move the cursor to the topleft position.
13, \r or CR	Move the cursor to the beginning of this line

III. Literature

A. Hendrix, Marcel;
"F-4TH reference Manual";
Weert, Holland, 1989

B. Ruzinsky, Steven A.;
"A simple Minimax Algorithm";

in "Dr. Dobbs Journal", volume 9, number 93, 1984

C. ANS X3J14 committee;
"Draft Proposed American National Standard: Programming Language FORTH dpANS-5"

American National Standards Institute, BSR center,

11 W. 42nd St. 13 Floor NY, NY 10036 Manhattan Beach CA, USA, 1992

D. Forth Standards Team;
"Forth-83 Standard";
Mountain View CA, USA, August 1983

E. intel, Osborne McGraw-Hill;
"i486 Microprocessor, Programmer's Reference Manual";
Intel Corporation, 1990

F. Woehr, Jack
"Forth: The New Model, A Programmer's Handbook"
M&T Publishing, 1992

G. Brodie, Leo;
"Starting FORTH";
Prentice-Hall International, Inc., London, 1981

H. Jyrki Yli-Nokari;
"Local Variables and Arguments";
Forth Dimensions Vol XI, page 13.

IV. Addresses

The author of iForth can be reached as follows:

Marcel Hendrix
Bressele-dijk 2
6024 CA Budel-Dorplein
The Netherlands
mhx@iae.nl (Marcel Hendrix)

Contact the Dutch Forth Workshop for Transputer (T8xx, T4xx) related Forths. These parallel Forths are source code compatible with iForth (when no transputer specific extensions are used). DFW also distributes the public domain CHForth (16-bit 8088-80x86, source-code compatible with iForth) and an ANS Forth for 8051-like processors.

For more information about the DFW write or call:

Dutch Forth Workshop
Boulevard Heuvelink 126
6828 KW Arnhem
The Netherlands
Tel: +31-26-4431305
BBS: +31-26-4422164