CHAPTER 20. DECOMPILER

The source code of the decompiler is in UTILITY.BLK, screens 31 to 42.

A decompiler is a program which can translate an object program in machine executable form back to the source program a human being can read. This is normally impossible because traditional compilers produce more object code than source code, showing a 'code expansion'. However, decompilation is rather easy in Forth because there is an one-to-one correspondence between the source code and object code in Forth, as a word or a command in Forth is compiled to an execution address in the object. Exception to this one-to-one relationship occurs in the control structures and some other special compiler directives. A Forth decompiler must be able to deal with these exceptions.

The final command doing the decompilation is SEE, which is used in the following fashion:

SEE <name>

where name is the name of a Forth word. The decompiled source code will be displayed on the terminal as a sequence of Forth words similar to the orginal source code.

20.1. POSITIONAL CASE DEFINING WORD

This is the simplest among the CASE control structures effecting an n-way branching. In the parameter field of the case word are a sequence of execution addresses. One address in the list is selected by the number on the stack and executed. In this version of case, additional range checking is also implemented for safety.

```
: OUT                   ( n pfa --- )   Display an error message if the index is out of range for a
                                        case word whose parameter field address pfa is on the stack.
        CR ." Subscript out of range on "       Initial error report.
        DUP BODY>                       Get the code field address first.
        >NAME                           Then the name field address.
        .ID                             Print the name of the case word.
        ."   Max is " ?                 Print the range allowed by the case word.
        ."   tried " .                  The index tried.
        QUIT                            Abort.
        ;

: MAP           ( n pfa --- addr )      Given the pfa of a case word and the index n for case
                                        selection, return the execution address selected. Abort if the
                                        index is out of range.
        2DUP @                          Fetch the range from pfa.
        U< IF                           Is the index n within range?
                2+ SWAP 2* +            Address of the execution code.
        ELSE OUT                        Abort if out of range.
        THEN ;
```

The case defining word is CASE: . It is used in the same way as a regular colon defining word. The name of the new case word follows CASE:, and then a list of regular Forth words followed by ; . A range number should be on the stack before CASE: is encountered to specify the number of branches in the case word.

n CASE: <name> <list of Forth words> ;

When the new case word <name> is executed, it uses the top item on the stack as an index to select one of the Forth words in the list and executes it.

```
: CASE:          ( n --- )      A positional case statement. The range n is used for error
                                checking. At runtime, the nth word is executed, depending
                                on the value on stack when executed.
    CONSTANT                    Compile the range n as a constant.
    HIDE                        Smudge the name field as : would do.
    ]                           Now, use the colon compiler to compile the cases.
                                Compilation will be terminated by the ; command.
    DOES>        ( index --- )  At runtime, use the index to find the execution address
                                among the compiled cases and execute it.
        MAP                     Return the address pointing to one of the cases compiled.
        PERFORM                 Execute it.
    ;
```

Because of the multitude of special compiler directives used in the F83 system, we need a big case statement to handle all the exceptions. This CASE: defining word, though very simple by borrowing facilities in the colon compiler, is extremely powerful to take care of a wide range of n-way branching structures. The limitation is that all the cases must be defined as single words. This is not a problem because it is a good practice to modularize the cases into single testable words before putting them into a big case structure.

20.2. ASSOCIATIVE DEFINING WORD

An associative word also has a list of values in its parameter field. At runtime a value on the top of the data stack is compared with the list of values in the associative word. If a match is found, the index of the matched value in the parameter field is returned. This is the inverse of an array.

```
: ASSOCIATIVE:   ( n --- )      Store the maximum range of the associative array as a
                                constant. The values will be compiled explicitly by the ,
                                (comma) command.
    CONSTANT                    Compile n as a constant.
    DOES>        ( value --- index )   Search value in the parameter field and return the
                                index if found.
    DUP @                       Get the range n.
    -ROT         ( n value pfa --- )
    DUP @                       Get another copy of n.
    0 DO                        Scan the list in parameter field.
        2+                      Next number in the list.
```

```
            2DUP @ =            Match?
            IF                  Yes.
            2DROP DROP          Clear the stack.
            I 0 0               Put on the index and flags.
            LEAVE               Quit the loop.
            THEN
        LOOP
        2DROP                   Return only the index.  If no match, return n+1.
        ;
```

Associative and case words are using to build tables to drive the decompiler.

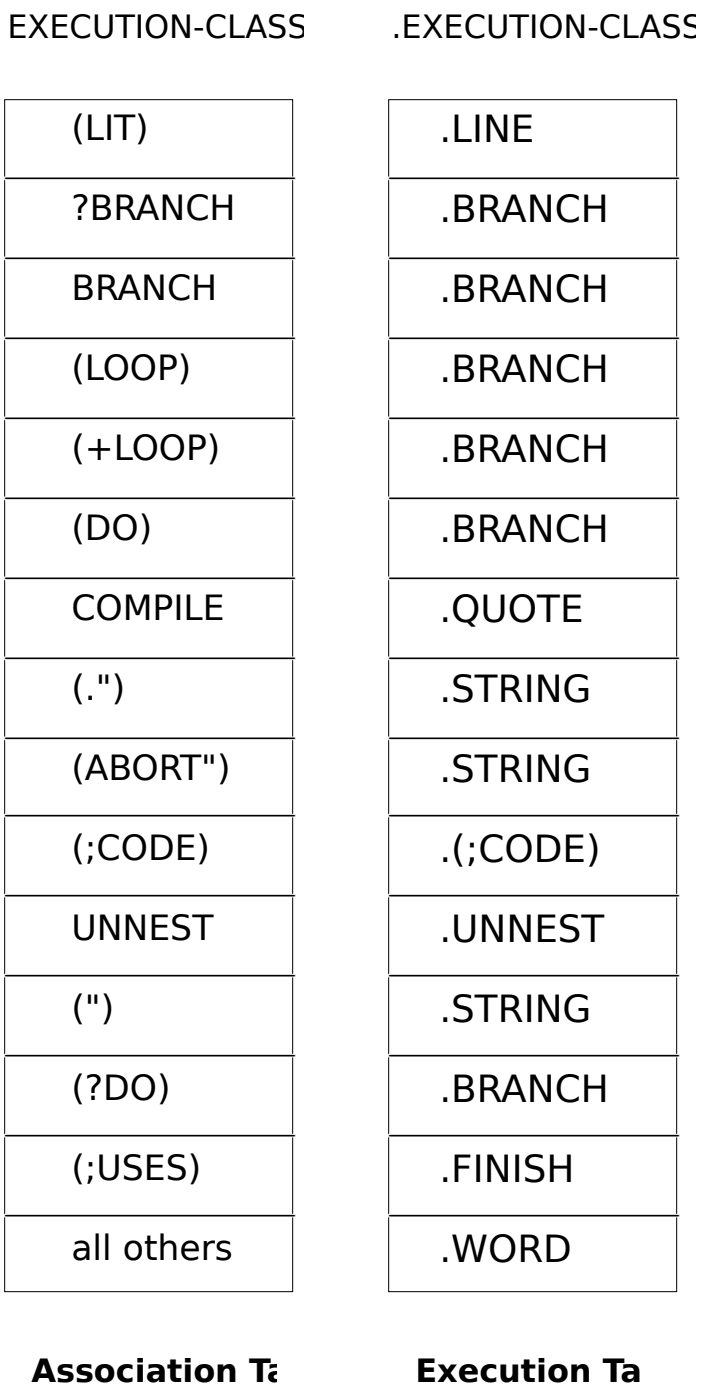
20.3. DECODING DIFFERENT CLASSES OF WORDS

There are several types or classes of words which execute differently and thus require different actions to decode them. The decompiler does not have to do much other than printing the names of the words and taking care of the additional information compiled into the object code with the word.

```
DEFER (SEE)                     A deferred word vectored to decompile deferred words.
HIDDEN DEFINITIONS              Hide all the supporting words in the HIDDEN vocabulary.

: .WORD             ( ip --- ip+2 )  Display the name of a colon word and increase the ip by 2.
        DUP @                        Execution address.
        >NAME .ID                    Print the name.
        2+      ;

: .INLINE           ( ip --- ip+4 )  Display an inline literal and its value.
        .WORD                        Print the name.
        DUP @ .                      Print the value.
        2+                           Increment ip again.
        ;

: .BRANCH           ( ip --- ip+4 )  Display a word that has an inline branch address.
        .WORD                        Print the name of the branch word.
        DUP @ OVER - .               Print the branching offset.
        2+                           Increment ip again.
        ;

: .QUOTE            ( ip --- ip+4 )  Handle the special case of COMPILE xxx .
        .WORD                        Print COMPILE.
        .WORD                        Print name of xxx.
        ;

: .STRING           ( ip --- ip' )   Display a word with inline string argument.
        .WORD                        Print name.
        COUNT 2DUP TYPE              Type out the inline string.
        SPACE
```

```
        +                       Add the string length to ip to skip over the in-line string.
        EVEN                    Align the cell boundary.
        ;

: DOES?         ( ip --- ip' f )  Increment simulated ip and return a true flag if DODOES is
                                  called as the first instruction in the parameter field.
        DUP 3 +                 Skip over the CALL DODOES code.
        SWAP @                  Get the machine code.
        DOES-OP =               Is it a CALL instruction?
        ;                       Return the flag.
```

Figure 20.1 Decoding different types of words.



```
: .(;CODE)          ( ip --- ip' )    Decompile a DOES> word.
        .WORD                         Print name.
        DOES?                         Is it a DOES> word?
        IF ." DOES> "          Yes.   Print DOES>.
        ELSE DROP FALSE               Otherwise, replace ip with a 0.
        THEN
        ;

: .UNNEST           ( ip --- 0 )      End of a colon definition.
        ." ; "                        Print ; .
        DROP 0                        Replace ip with 0.
        ;

: .FINISH           ( ip --- 0 )      Display current word and quit.
        .WORD
        DROP 0                        Replace ip with 0, indicating end of decompilation.
        ;
```

20.4. SORTING AND EXECUTION TABLES

The associative word EXECUTION-CLASS collects all the special cases that must be decompiled differently from normal Forth words like DUP, + , etc. At runtime if the address pointed to by ip matches the address of a word in this table, the corresponding index will be returned. This index will be used to select an execution address in the following case table and the appropriate decompilation function will be invoked. These two tables make up the basic mechanism of this table driven decompiler.

```
14 ASSOCIATIVE: EXECUTION-CLASS          14 classes of special compiler words are to be
                                         processed.
' (LIT) ,                                Each execution address must be compiled explicitly using , .
' ?BRANCH ,          ' BRANCH ,    ' (LOOP) ,      ' (+LOOP) ,
' (DO) ,             ' COMPILE ,          ' (.") ,          ' (ABORT") ,
' (;CODE) ,          ' UNNEST ,    ' (") ,    ' (?DO) ,
' (;USES) ,
```

```
15 CASE: .EXECUTION-CLASS     A giant case statement handles the special case decompilation.
                              Each entry corresponds to an entry in the EXECUTION-
                              CLASS assocoative table.  In case of no match, .WORD will
                              be executed, assuming a normal Forth word.
        .INLINE     .BRANCH   .BRANCH   .BRANCH   .BRANCH   .BRANCH
        .QUOTE      .STRING   .STRING   .(;CODE)  .UNNEST   .STRING
        .BRANCH     .FINISH   .WORD
        ;                     CASE: must end with a ; , because it borrows the colon
                              compiler to do the compiling.
```

20.5. DECOMPILING DIFFERENT WORD CLASSES

When the decompiler is given a word to decompile, it has to determine first which type this word is. If the word is of a simple type, like constant or variable, all the decompiler has to do is to tell the user what it is. Decompilation is only needed for the more complicated colon words. Therefore, we need another case and associative table pair to handle different types of words.

```
: .PFA          ( cfa --- )          Given the code field address of a colon word, decompile the
                                     list of execution addresses in its parameter field.
        >BODY                        Transform cfa into pfa.
        BEGIN                        Scan the parameter field.
                DUP @                Get an execution address.
                EXECUTION-CLASS      Identify which class the word belongs.
                .EXECUTION-CLASS     Decompile it.
                DUP                  Dup the ip or the flag.
                0= KEY? OR           If it is 0 or a key was pressed, terminate the loop.
        UNTIL                        Otherwise continue decompiling.
        DROP                         Last flag.
        ;

: .IMMEDIATE        ( cfa --- )      Indicate whether the current word is immediate or not.
        >NAME                        Get to the name field.
        C@                           The count byte at the beginning of the name field.
        64 AND                       Is the precedent bit set?
        IF                           Yes.
                ." IMMEDIATE"        Print that it is immediate.
        THEN ;

: .CONSTANT         ( cfa --- )      Decompile a constant and print its value.
        DUP >BODY ?                  Print its value first.
        ." CONSTANT "                Print the type.
        >NAME .ID                    And the name.
        ;

: .VARIABLE ( cfa --- )              Decompile a variable.  Print its location and value.
        DUP >BODY .                  Print its location.
        ." VARIABLE "                Type.
        DUP >NAME .ID                Name.
        ." Value = " >BODY ?         Value.
        ;

: .:                ( cfa --- )      Decompile a colon definition.
        ." : "                       Print the almighty colon.
        DUP >NAME .ID                Name.
        2 SPACES
        .PFA                         Decompile the parameter field.
        ;
```

```
: .DOES>              ( cfa ---)    Decompile a word defined by a CREATE-DOES> defining word.
        DUP >NAME .ID               Name.
        ." DOES> "                  Type.
        BODY>                       Address of the high level runtime code or the interpreter.
        .PFA                        Decompile the interpreter.
        ;


: .USER-VARIABLE  ( cfa --- )       Decompile a user variable.  Print its offset from the base of
                                    user area and its current value.
        DUP >BODY ?                 Offset.
        ." USER VARIABLE "          Type.
        DUP >NAME .ID               Name.
        ." Value = " >IS .          Value.
        ;

: .DEFER              ( cfa --- )   Tell the user that this is a deferred word and decompile its
                                    current definition.
        ." DEFERRED "               Type.
        DUP >NAME .ID               Name.
        ." IS "                     Deferred.
        >IS @ (SEE)                 Decompile the vectored word.
        ;

: .USER-DEFER         ( cfa --- )   Tell the user that it is a user deferred word and decompile its
                                    current definition.
        ." USER DEFERRED "          Type.
        DUP >NAME .ID               Name.
        ." IS "                     Deferred.
        >IS @ (SEE)                 Decompile the current definition.
        ;

: .OTHER              ( cfa --- )   Decompile words whose type is not known.
        DUP >NAME .ID               Print the name first.
        DUP @                       Contents of code field.
        OVER >BODY =                Is it pfa?
        IF                          Yes.  Must be a code definition.
                DROP
                ." is code"         Print type.
                EXIT                Quit because we have no disassembler.
        THEN
        DUP DOES? IF                Is it a 'does' word?
                DROP
                DOES> EXIT          Decompile it as a DOES> word.
        THEN
        2DROP
        ." is unknown"              Tell the truth also.
        ;
```

20.6. WORD CLASSIFICATION

Different classes of word definitions are characterized by the inner interpreters which execute the words. Words of the same class share the same inner interpreter, whose address is stored in the code field of these words. Inner interpreters are code routines in the virtual Forth machine and generally they do not have names and cannot be referred directly. However, we can find the address of an inner interpreter by looking at the code field of any word in the corresponding class.

```
6 ASSOCIATIVE: DEFINITION-CLASS     Categorize different classes of words that the
                                    decompiler will handle.  For each class defined by the same
                                    defining word, the code field is identical. Thus standard
                                    classes can be recognized.
        ' QUIT @ ,                  Colon word.
        ' 0 @ ,                     Constant.
        ' SCR @ ,                   Variable.
        ' BASE @ ,                  User variable.
        ' KEY @ ,                   Deferred word.
        ' EMIT @ ,                  User deferred word.

7 CASE: .DEFINITION-CLASS           Define a table of routines to handle decompilation of each
                                    class of definition.
        .:                          Colon word decompiler.
        .CONSTANT
         .VARIABLE
        .USER-VARIABLE
        .DEFER
        .USER-DEFER
        .OTHER                      Code and DOES> words.
        ;
```

20.7. THE DECOMPILER 'SEE'

```
: ((SEE))               ( cfa --- )     Given an arbitrary code field address, decompile it based
                                        upon its definition class.  Upon completion, indicate whether
                                        or not the word is immediate.
        CR      DUP DUP @               Get the contents of the code field.
        DEFINITION-CLASS                Determine the type of definition.
        .DEFINITION-CLASS               Decompile it.
        .IMMEDIATE                      If it is an immediate word.
        ;

' ((SEE)) IS (SEE)                      (SEE) is a deferred word so that .DEFER and .USER-DEFER
                                        can make use of it before it is actually defined. Now patch it in.
```

FORTH DEFINITIONS All the above supporting word are defined in the HIDDEN vocabulary. Now switch context back to FORTH and declare it the current vocabulary so that the decompiler word SEE will be available to the user in the FORTH vocabulary.

```
: SEE                   ( --- ) SEE <name>  decompiles the word whose name follows SEE.
        '                       Get the code field address of the word <name>.
        (SEE)                   Decompile it.
        ;
```

Figure 20.2 Decompiling different types of words.