PART ONE. FORTH SYSTEMS

I. METACOMPILING F83 FOR ROM BASED SYSTEMS

'Metamorphosis: Change of form, structure, or substance, as transformation by magic or witchcraft.'

The American College Dictionary

1. INTRODUCTION

Metacompilation is the process by which a new Forth system is generated by an existing Forth system for a target computer which may or may not be the same as the one hosting the existing Forth system. It is the higest form of Forth programming activity. It is also the very last subject a Forth programmer has to learn. After mastering metacompilation, there is nothing more to learn, as far as Forth being a language or an operating system.

Metacompilation was a very integral part of Forth since its very beginning. After Chuck Moore coined the word 'FORTH' for his new found toy, the most interesting thing he found with it was that it could recompile itself. Thus he was able to port Forth to every computer in sight. Thereafter, all Forth system were derived from another Forth and he did not need any other programming tool to put Forth on a new computer. All the products developed by Forth, Inc. were results of meta-compilation. This certainly was of great advantage to Forth, Inc. when it had to release source code to their customers, because all the source code were in Forth. Either by striping the comments intentionally or by the absence of comments as code was developed, Forth, Inc. was able to distribute Forth with source code without revealing any proprietory information for many years.

The scence changed drastically when Forth Interest Group released the fig-FORTH models. Although Bill Ragsdale apparently had mastered the art of metacompilation, as the fig-FORTH Installation Manual contained the 6502 Forth model written in Forth, he didn't release his metacompiler. Instead, all the fig-Forth implementations were published in native machine assembly code. Therefore, to bring up a fig-FORTH system, one had to use an operating system and its associated assembler. Although many vendors supplied metacompilers for various microcomputers, metacompilation was generally perceived as difficult and not natural for Forth. Metacompilation became the status of Forth priesthood. Mastering this black art accords the privilage to look down upon one's following

misfortuned Forthians.

The implementors of F83 model needed the power of metacompilation and tapped it in porting F83 over four host systems, CP/M 8080, CP/M 8086, CP/M 68000, and IBM PC-DOS. The F83 systems were released with the metacompilers used to generate themselves, giving all users equal access to all the tools needed to modify the system for specific applications. It is now very easy to delete the bells and whistles in F83 which are not used in an application, and add the application words to build a customized Forth system. This feature undoubtly contributes to the popularity of F83 amoung serious Forth practitioners.

One weakness of F83 is that it was designed only for a RAM based computer with CP/M-like environment. Code in the dictionary are contaminated by data structures which might change when Forth is running. These impure data structures are variables, user variables, deferred words, and vocabulary thread tables. It is fortunate that the stacks, disk buffers, and terminal input buffers were separated from the dictionary. Because of the commingling of data with code in the dictionary, F83 as is cannot be put into ROM without efforts in cleansing the code. This set of modified F83 metacompiler allows the generation of pure Forth code, which can be used to implement a ROM based 8080 system.

2. ROMMABLE F83 METACOMPILER

The first application of this metacompiler was to generate a Forth target system to be used in the MA2000 Macrocomponent Microprocessor system manufactured by National Semiconductor. Major features of this target system include:

a. 16 Kbytes of dictionary in ROM and 16 Kbytes of RAM for dictionary expansion, stacks, and buffers.
b. 32 Kbytes of RAM or ROM, which may be used as a disk for source code.
c. Complete F83 kernel except BIOS/BDOS interfaces for terminal and disk I/O.
d. F83 8080 assembler.
e. Starting Forth line editor.
f. F83 four-way threaded vocabulary structure.
g. Application can be added to ROM dictionary through metacompilation or compiled from the 32 Kbyte ROM disk.

The following memory map shows how the target system is configured:

0	Reset and interrupt vectors
100	Forth Kernel in 12 Kbytes of ROM
3000	4 Kbytes of ROM for expansion
4000	User variables and variables
4100	RAM for dictionary and stack expansion

7DE6	Terminal input buffer and return stack
7000	4 Kbytes of disk buffers
8000	32 Kbytes of RAM/ROM disk
FFFF	End of ROM/RAM disk

This metacompiler consists of three files: M40.BLK, K40.BLK, and U40.BLK. The M40.BLK is the meta-compiler, which isderived from the META80.BLK in F83/8080 system. K40.BLK is derived from KERNEL80.BLK, containing the Forth kernel for the MA2000 microprocessor. U40.BLK contains the 8080 assembler, the Starting Forth editor, and a few useful utility from F83's UTILITY.BLK. Modifications necessary to make the target ROMmable are:

a. VARIABLE in the target compiles a pointer to a RAM location between 4030H and 40FFH.
b. User VARIABLE compiles a pointer to a location in the User area in RAM between 4000H and 402FH.
c. Vocabulary thread tables are moved to RAM to allow new words to be defined and linked in the target system.
d. View fields are eliminated.
e. A variable DP-R is defined in the metacompiler to kept track of the variables.
f. Assembler defining words <1MI> to <5MI> are defined in the metacompiler.
g. All variables are collected together.
h. Input words are not converted to upper case. All name comparisons are case sensitive.
i. BIOS I/O calls are substituted by machine code interfacing directly to the RS-232 port in MA2000 system.
j. LIMIT is set to 8000H. The disk buffers, stacks and terminal input buffers are moved downward accordingly.
k. Four vocabularies, FORTH, EDITOR, ASSEMBLER, and USER are defined in the target. The USER vocabulary is empty.
l. BLOCK returns a memory address between 7000H and FC00H, allowing 36 blocks to be accessed.
m. All file and virtual memory management words are eliminated.
n. COLD initializes all I/O ports and the variables.
o. A table INIT-RAM contains all the values to initialize user variables, variables and vocabulary thread tables.
p. 8080 Assembler is included.
q. Starting Forth line editor is included.
r. Ultility words \ , THRU, LIST, TRIAD, INDEX, and WORDS are added.
s. A much simplified version of ONLY and ALSO is added.

For many target system which does not need on-site programming services, the assembler and the editor do not have to be meta-compiled. In this case, U80.BLK can be substituted by a file containing the application source code.

3. MTECOMPILING PROCEEDURE

The sequence of events to carry out a metacompilation process will be shown here, assuming that the U40.BLK file contains debugged application source code. Boot up the computer and load F83/8080 from drive B, with the metacompiler diskette in drive A:

```
A>B:F83 M40.BLK
```

After the F83 sign-on messages appear on the screen, type:

```
1 LOAD
```

to load the metacompiler. After it is loaded, type:

```
OPEN K40.BLK 1 LOAD
```

A series of screen numbers will then scroll through the CRT. After an 'ok' is finally displayed, type:

```
0 THERE HERE-T 256 +
ONLY FORTH ALSO DOS
SAVE B:TARGET.COM
```

Assuming that there are enough space on the diskette in B drive to store the target system image. This object files can be used to program a set of EPROM's in the target system. The area between memory 0 and OFFH are filled with garbage. You must at least put a JMP 100H instruction at memory 0-2 for reset. The rest of this area can be used to implement interrupts if necessary.

If your need to be assured that there will be a high probability that the target system would work, you may try to metacompile a target system which can be tested and debugged on your CP/M host computer. This can be done by substituting the phrase

```
45 46 THRU
```

in Screen 1, line 4 of K40.BLK, after openning K40.BLK by

```
95 96 THRU
```

Screens 95 and 96 contain console I/O routines calling CP/M BIOS. Using these I/O routines, you will be able to bring up the target Forth system under CP/M. However, the command sequence is slightly different

```
OPEN K80.BLK
1 EDIT
D 45 46 THRU
I 95 96 THRU
DONE FLUSH
1 LOAD
```

After the kernel and the utility are loaded, type:

```
256 THERE HERE-T
ONLY FORTH ALSO DOS
SAVE B:TEST.COM
BYE
A>B:TEST
```

If there is no problem in the kernel and the application code, TEST should be loaded and start executing. Whatever the sign-on message you programmed into the COLD routine will be displayed on the CRT screen and the Forth interpreter should respond to keyboard input. If not, well, happy debugging. You can use your favorate debugger to step through the initialization process:

```
A>DEB B:TEST.COM
```

From this point on, you are on your own. It's like a mother giving birth to her child. The labor is agony and pain. However, all the sufferring is compensated fully with the first sight of the new-born. The joy and ecstasy in being able to create one's own Forth systems accord Forthians the sense of superiority and righteousness, keeping the fever high forever.

4. POSSIBLE PROBLEMS

Philosophy aside, here are some hints on what might go wrong in meta-compilation:

a. Hardware memory configuration does not match the memory map.
b. The stack point is not initialize correctly. It should at least point to a RAM location before Forth moves it to the final location.
c. The console I/O port is not initialized correctly.
d. The CRT terminal does not match the console characteristics.
e. The CPU interrupt structure is not initialized or disabled.
f. Entries in INIT-RAM do not match the variables.
g. Vocabulary thread tables are not properly aligned.
h. Reset vector at memory 0 does not point to 100H.

Other causes of problem should be eliminated:

i. Power cord not plugged in.
j. RS-232 cable mismatch.
k. EPROM chips installed backward. Don't laugh. It happened to me many, many times.
l. EPROM pins are bent out of the sockets.
m. Not enough disk space to save TARGET.COM.

```
      0
 0 Meta-compiler for ROM based systems
 1
 2 This meta-compiler is based on that in F83 system by Henry
 3 Laxen and Mike Perry.
 4 Modifications are added so that the target code can be
 5 programmed into EPROM's.
 6 An 8080 assembler and a line editor are added to aid program
 7 development in a standalone environment.
 8
 9 Dr. C. H. Ting
10 Offete Enterprises, Incorporated.
11 1306 South B street
12 San Mateo, California  94402
13 (415) 574-8250
14
15
```

```
      1
 0 \ Load Screen for Pre-Compile
 1 ONLY FORTH ALSO DEFINITIONS
 2
 3 FENCE OFF   FORGET OUT
 4 WARNING OFF
 5 : NLOAD   CR .S  (LOAD) ;   ' NLOAD IS LOAD
 6  3 22 THRU   ( The Meta Compiler )
 7 ONLY FORTH DEFINITIONS ALSO
 8 CR .( Meta Compiler Loaded )
 9 EXIT
10 FROM KB0.BLK 1 LOAD
11
12
13
14
15
```

```
      3
 0 \ Vocabulary Helpers
 1 ONLY FORTH ALSO
 2 VOCABULARY META   META ALSO       META DEFINITIONS
 3 VARIABLE DP-T
 4 VARIABLE DP-R
 5 : [FORTH]        FORTH   ; IMMEDIATE
 6 : [META]         META    ; IMMEDIATE
 7 : [ASSEMBLER]    ASSEMBLER    ; IMMEDIATE
 8 : SWITCH   (S -- )
 9    NOOP ( Context )   NOOP ( Current )
10    DOES>
11       DUP @  CONTEXT @   SWAP CONTEXT !   OVER !   2+
12       DUP @  CURRENT @   SWAP CURRENT !   SWAP !   ;
13    SWITCH   ( Redefine itself )
14
15
```

```
      4
 \ Memory Access Words                                    13Jun85cht
 0 CONSTANT TARGET-ORIGIN
 : THERE   (S taddr -- addr )   TARGET-ORIGIN +   ;
 : C@-T    (S taddr -- char )   THERE C@ ;
 : @-T     (S taddr -- n )      THERE @  ;
 : C!-T    (S char taddr -- )   THERE C! ;
 : !-T     (S n taddr -- )      THERE !  ;
 : HERE-T  (S -- taddr )   DP-T @   ;
 : ALLOT-T (S n -- )       DP-T +!   ;
 : C,-T    (S char -- )   HERE-T C!-T   1 ALLOT-T   ;
 : ,-T     (S n -- )      HERE-T  !-T   2 ALLOT-T   ;
 : S,-T    (S addr len -- )
    0 ?DO   COUNT C,-T   LOOP   DROP   ;
 : HERE-R   DP-R @   ;
 : ALLOT-R   DP-R +!   ;
```

```
      5
05aug86cht \ Define Symbol Table Vocabularies                 21Dec83mas
           VOCABULARY TARGET
           VOCABULARY TRANSITION
           VOCABULARY FORWARD
           VOCABULARY USER
           ONLY DEFINITIONS FORTH ALSO META ALSO
           : META META ;
           : TARGET TARGET ;
           : TRANSITION TRANSITION ;
           : FORWARD FORWARD ;
           : USER USER   ;
           : ASSEMBLER ASSEMBLER ;
           ONLY FORTH ALSO META ALSO DEFINITIONS
```

```
      7
13Jun85cht \ 8080 Meta Assembler                              01AUG83HH
           : ?>MARK      (S -- f addr )   TRUE   HERE-T   0 ,-T   ;
           : ?>RESOLVE   (S f addr -- )   HERE-T SWAP !-T   ?CONDITION   ;
           : ?<MARK      (S -- f addr )   TRUE   HERE-T   ;
           : ?<RESOLVE   (S f addr -- )   ,-T   ?CONDITION   ;

           ALSO ASSEMBLER
           META ' C,-T        ASSEMBLER IS C,
           META ' ,-T         ASSEMBLER IS ,
           META ' ?>MARK      ASSEMBLER IS ?>MARK
           META ' ?>RESOLVE   ASSEMBLER IS ?>RESOLVE
           META ' ?<MARK      ASSEMBLER IS ?<MARK
           META ' ?<RESOLVE   ASSEMBLER IS ?<RESOLVE

           ONLY FORTH ALSO META ALSO DEFINITIONS
```

Listing 1. Metacompiler for ROM based systems

```
      8
 0 \ Meta Compiler Vocabulary Manipulators
 1 : MAKE-CODE   (S PFA -- )
 2    @ ,-T  ;
 3 : LABEL   (S -- )
 4    ASSEMBLER DEFINITIONS   HERE-T CONSTANT   ;
 5 : IN-TARGET   (S -- )
 6    ONLY TARGET DEFINITIONS   ;
 7 : IN-TRANSITION   (S -- )
 8    ONLY FORWARD ALSO TARGET DEFINITIONS ALSO TRANSITION   ;
 9 : IN-META     (S -- )
10    ONLY FORTH ALSO META DEFINITIONS ALSO   ;
11 : IN-FORWARD   (S -- )
12    FORWARD DEFINITIONS   :
13
14
15
```

```
      9
 0 \ Meta Compiler Forward Reference Linking
 1 : LINK-BACKWARDS   (S PFA -- )
 2    HERE-T OVER @ ,-T   SWAP !   ;
 3 : RESOLVED?   (S pfa -- f )
 4    2+ @   ;
 5 : FORWARD-CODE   (S pfa -- )
 6    DUP RESOLVED? IF  MAKE-CODE  ELSE  LINK-BACKWARDS  THEN :
 7 : FORWARD:    (S -- )
 8    SWITCH   FORWARD DEFINITIONS   CREATE SWITCH  0 , 0 ,
 9    DOES>   FORWARD-CODE   :
10
11
12
13
14
15
```

```
     10
 0 \ Create Headers in Target Image
 1 VARIABLE WIDTH     31 WIDTH !
 2 VARIABLE LAST-T
 3 VARIABLE CONTEXT-T
 4 VARIABLE CURRENT-T
 5 : HASH   (S str-addr voc-addr -- thread )
 6    SWAP 1+ C@ 3 AND 2* +   ;
 7 : HEADER   (S -- )
 8    BL WORD C@ 1+ WIDTH @ MIN   ?DUP IF
 9 \       ALIGN   BLK @ 4096 + ,-T   ( Lay down view field )
10       HERE CURRENT-T @ HASH DUP @-T ,-T
11       HERE-T 2- SWAP !-T
12       HERE-T HERE ROT 5,-T   ALIGN   DUP LAST-T !
13       128 SWAP THERE CSET   128 HERE-T 1- THERE CSET
14    THEN   :
15
```

```
     11
04Apr84map \ Meta Compiler Create Target Image                04Apr84map
 : TARGET-CREATE   (S -- )
    >IN @ HEADER >IN !  IN-TARGET CREATE IN-META  HERE-T , TRUE ,
    DOES>   MAKE-CODE   ;
 : RECREATE   (S -- )
    >IN @   TARGET-CREATE   >IN !   ;
 : CODE   (S -- )
    TARGET-CREATE   HERE-T 2+ ,-T  ASSEMBLER  !CSP  :
 ASSEMBLER ALSO DEFINITIONS
 : END-CODE
    IN-META   ?CSP   ;
 : C;   END-CODE   ;
 META IN-META
```

```
     12
02Nov83map \ Force compilation of target & forward words      07SEP83HH
 : 'T   (S -- cfa )
    CONTEXT @   TARGET DEFINED   ROT CONTEXT !
    0= ?MISSING   ;
 : [TARGET]   (S -- )
    'T , ;   IMMEDIATE
 : 'F   (S -- cfa )
    CONTEXT @   FORWARD DEFINED   ROT CONTEXT !
    0= ?MISSING   ;
 : [FORWARD]   (S -- )
    'F , ;   IMMEDIATE
```

```
     13
02aug86cht \ Meta Compiler Branching & Defining Words         07SEP83HH
 : T;   (S -- )
    SWITCH   TRANSITION DEFINITIONS   CREATE   SWITCH   ]
    DOES>   >R   ;
 : T;   (S -- )
    SWITCH   TRANSITION DEFINITIONS   [COMPILE] ;   SWITCH   ;
    IMMEDIATE
 : DIGIT?   (S CHAR -- F )
    BASE @ DIGIT NIP   ;
 : PUNCT?   (S CHAR -- F )
    ASCII . OVER = SWAP   ASCII - OVER = SWAP
    ASCII / OVER = SWAP   DROP OR OR ;
 : NUMERIC?   (S ADDR LEN -- F )
    DUP 1 = IF   DROP C@ DIGIT?   EXIT   THEN
    1 -ROT   0 ?DO   DUP C@   DUP DIGIT? SWAP PUNCT? OR
      ROT AND SWAP 1+   LOOP   DROP   ;
```

Listing 1. Metacompiler for ROM based systems (cont'd)

```
    14
 0 \ Meta Compiler Transition Words
 1 T: (   [COMPILE] (   T;
 2 T: (S  [COMPILE] (S  T;
 3 T: \   [COMPILE] \   T;
 4 : STRING,-T   (S -- )
 5    ASCII " PARSE  DUP C,-T  S,-T  ALIGN  ;
 6 FORWARD: (,")
 7 T: ."
 8    [FORWARD] (,")   STRING,-T   T;
 9 FORWARD: ("")
10 T: "      [FORWARD] ("")   STRING,-T   T;
11 FORWARD: (ABORT")
12 T: ABORT"
13    [FORWARD] (ABORT")    STRING,-T   T;
14
15
```

```
    15
 0 \ Meta Compiler Defining Words
 1 FORWARD: (CREATE)
 2 : CREATE
 3    RECREATE    [FORWARD] (CREATE)   HERE-T CONSTANT   ;
 4 FORWARD: (VARIABLE)
 5 : VARIABLE   (S -- )
 6    RECREATE    [FORWARD] (VARIABLE)   HERE-R CONSTANT
 7    HERE-R ,-T   2 ALLOT-R   ;
 8 FORWARD: (DEFER)
 9 : DEFER   (S -- )
10    TARGET-CREATE   [FORWARD] (DEFER)   0 ,-T   ;
11
12
13
14
15
```

```
    16
 0 \ Meta Compiler Defining Words
 1 FORTH VARIABLE $USER-T META
 2 ALSO USER DEFINITIONS
 3 : ALLOT       (S n -- )
 4    $USER-T +!   ;
 5 FORWARD: (USER-VARIABLE)
 6 : VARIABLE    (S -- )
 7    SWITCH   RECREATE   [FORWARD] (USER-VARIABLE)   $USER-T @
 8    DUP ,-T   2 ALLOT   META DEFINITIONS   CONSTANT   SWITCH   ;
 9 FORWARD: (USER-DEFER)
10 : DEFER       (S -- )
11    SWITCH   TARGET-CREATE   [FORWARD] (USER-DEFER)   SWITCH
12    $USER-T @ ,-T   2 ALLOT   ;
13 ONLY FORTH ALSO META ALSO DEFINITIONS
14
15
```

```
    17
11Mar84map \ Meta Compiler Transition Words                    14Jun85cht
          FORTH VARIABLE VOC-LINK-T META
          FORWARD: (VOCABULARY)
          : VOCABULARY   (S -- )
             RECREATE    [FORWARD] (VOCABULARY)
             HERE-R ,-T   10 ALLOT-R
             HERE-T   #THREADS 0 DO  0 ,-T  LOOP
             HERE-T VOC-LINK-T @ ,-T   VOC-LINK-T !
             CONSTANT   DOES> @ CONTEXT-T !   ;
          : IMMEDIATE   (S -- )   WIDTH @ IF ( Headers present? )
             64 ( Precedence Bit )   LAST-T @ THERE  CSET   THEN   ;
```

```
    18
13Jun85cht \ Meta Compiler Transition Words                    04Apr84map
          FORWARD: ((;USES))
          FORTH VARIABLE STATE-T   META
          T: ;USES   (S -- )
             [FORWARD] ((;USES))   IN-META ASSEMBLER
             !CSP   STATE-T OFF   T;
          T: [COMPILE]
             'T EXECUTE    T;
          FORWARD: ((IS))
          T: IS      [FORWARD] ((IS))    T;
          : IS    'T >BODY @ >BODY !-T    ;
          T: ALIGN   T;
          T: EVEN    T;
```

```
    19
07SEP83HHL \ Display an unformatted Symbol Table              13Jun85cht
          : .SYMBOLS    (S -- )
             TARGET   CONTEXT @ HERE #THREADS 2* CMOVE
             BEGIN   HERE 4 LARGEST  DUP
             WHILE   ?CR   ." [[ "   L>NAME DUP .ID
               DUP NAME> >BODY @ U.    ." ]] "   N>LINK @ SWAP !
               KEY? IF   EXIT   THEN
             REPEAT   2DROP   IN-META   ;
```

Listing 1. Metacompiler for ROM based systems (cont'd)

```
     20                                                    0
 0 \ Meta Compiler Resolve Forward References          07Jan84map Meta-compiler for ROM based systems
 1 : .UNRESOLVED   (S -- )
 2   FORWARD   CONTEXT @ HERE #THREADS 2* CMOVE    BEGIN          This meta-compiler is based on that in F83 system by Henry
 3      HERE #THREADS LARGEST   DUP WHILE                        Laxen and Mike Perry.
 4         ?CR DUP L>NAME NAME> >BODY                            Modifications are added so that the target code can be
 5         RESOLVED? 0= IF   DUP L>NAME .ID   THEN               programmed into EPROM's.
 6      @  SWAP !   REPEAT  2DROP  IN-META ;                     An 8080 assembler and a line editor are added to aid program
 7 : FIND-UNRESOLVED   (S -- cfa f )                            development in a standalone environment.
 8    'F    DUP  >BODY RESOLVED?     ;
 9 : RESOLVE   (S taddr cfa -- )                                Dr. C. H. Ting
10   >BODY   2DUP   TRUE OVER 2+ !   @ BEGIN   DUP WHILE       Offete Enterprises, Incorporated.
11      2DUP @-T   -ROT SWAP !-T   REPEAT   2DROP  !   ;       1306 South B street
12 : RESOLVES   (S taddr -- )                                  San Mateo, California  94402
13   FIND-UNRESOLVED IF   >NAME .ID ." Already Resolved"   DROP  (415) 574-8250
14   ELSE   RESOLVE   THEN   ;
15

     21                                                    0
 0 \ Interpretive words for Meta                       07SEP83HHL Meta-compiler for ROM based systems
 1 : H:   [COMPILE] :   ;
 2 H: '   'T >BODY @   ;                                        This meta-compiler is based on that in F83 system by Henry
 3 H: ,   ,-T ;                                                 Laxen and Mike Perry.
 4 H: C,  C,-T ;                                                Modifications are added so that the target code can be
 5 H: HERE HERE-T ;                                             programmed into EPROM's.
 6 H: ALLOT   ALLOT-T   ;                                       An 8080 assembler and a line editor are added to aid program
 7 H: DEFINITIONS  DEFINITIONS   CONTEXT-T @ CURRENT-T !   ;    development in a standalone environment.
 8
 9                                                             Dr. C. H. Ting
10                                                             Offete Enterprises, Incorporated.
11                                                             1306 South B street
12                                                             San Mateo, California  94402
13                                                             (415) 574-8250
14
15

     22                                                    0
 0 \ Assembler Defining words                          05aug86cht Meta-compiler for ROM based systems
 1 FORWARD: <1MI>
 2 FORWARD: <2MI>                                               This meta-compiler is based on that in F83 system by Henry
 3 FORWARD: <3MI>                                               Laxen and Mike Perry.
 4 FORWARD: <4MI>                                               Modifications are added so that the target code can be
 5 FORWARD: <5MI>                                               programmed into EPROM's.
 6 : 1MI   RECREATE [FORWARD] <1MI>   DUP C,-T   CONSTANT ;     An 8080 assembler and a line editor are added to aid program
 7 : 2MI   RECREATE [FORWARD] <2MI>   DUP C,-T   CONSTANT ;     development in a standalone environment.
 8 : 3MI   RECREATE [FORWARD] <3MI>   DUP C,-T   CONSTANT ;
 9 : 4MI   RECREATE [FORWARD] <4MI>   DUP C,-T   CONSTANT ;     Dr. C. H. Ting
10 : 5MI   RECREATE [FORWARD] <5MI>   DUP C,-T   CONSTANT ;     Offete Enterprises, Incorporated.
11                                                             1306 South B street
12                                                             San Mateo, California  94402
13                                                             (415) 574-8250
14
15
```

Listing 1. Metacompiler for ROM based systems (cont'd)

```
      0
 0 \ Kernel of ROM target system
 1
 2 This file contains the Kernal source code for a ROM based
 3 F83 target system.  Most screens are identical to the
 4 original F83/8080 screens.  Only those which were modified
 5 are commented in the corresponding shadow screens.
 6
 7 Consult Mike Perry and Henry Laxen's F83 source code or
 8 'Inside F83'.
 9
10
11
12
13
14
15
```

```
      1
 0 \ Target System Setup                              20JAN87CHT
 1 WARNING OFF
 2 ONLY FORTH META ALSO FORTH      16384 48 + DP-R !
 3 256 DP-T !  HERE 16000 + ' TARGET-ORIGIN >BODY !    IN-META
 4 2 44 THRU   45 46 THRU   47 50 THRU   58 88 THRU
 5 USER DEFINITIONS FROM OCU4.BLK 1 LOAD FORTH DEFINITIONS
 6 FROM U40.BLK 1 LOAD   89 92 THRU   EXIT
 7 ( System Source Screens )   WARNING ON
 8 CR .( Unresolved references: ) CR   .UNRESOLVED
 9 CR .(     Statistics: )  CR .( Last Host Address:          )
10 [FORTH] HERE U.          CR .( First Target Code Address:  )
11 META 256 THERE U.        CR .( Last Target Code Address:   )
12 META HERE-T THERE U.     CR CR
13 META  256 THERE HERE-T
14    ONLY FORTH ALSO DOS SAVE TARGET.COM   FORTH
15
```

```
      2
 0 \ Declare the Forward References  and Version #      04Apr84map
 1 : ]]   ]   ;
 2 : [[   [COMPILE] [   ; FORTH IMMEDIATE META
 3
 4 FORWARD: DEFINITIONS
 5 FORWARD: [
 6
 7
 8
 9
10
11
12
13
14
15
```

```
      3
 \ Boot up Vectors and NEXT Interpreter                 03aug86ch
 ASSEMBLER LABEL ORIGIN
 NOP   -1 JMP   ( Low Level COLD Entry point )
 NOP   -1 JMP   ( Low Level WARM Entry point )
 LABEL DPUSH   D PUSH     LABEL HPUSH   H PUSH
 LABEL >NEXT
       IP LDAX   IP INX    A L MOV   IP LDAX   IP INX   A H MOV
 LABEL >NEXT1
       M E MOV   H INX   M D MOV   XCHG   PCHL
 FORTH ASSEMBLER DEFINITIONS META
    H: NEXT   >NEXT JMP   ;
    H: IP>HL  B H MOV  C L MOV   ;  IN-META
 HERE-T DUP 100 + CURRENT-T !   ( harmless )
 VOCABULARY FORTH   FORTH DEFINITIONS
 0 OVER ( 2+ ) !-T ( link )
 DUP ( 2+ ) SWAP 16 + !-T ( thread )  IN-META
```

```
      4
 \ Run Time Code for Defining Words                     13Apr84map
 VARIABLE RP      ( Not enough registers on an 8080 )
 ASSEMBLER LABEL NEST
    RP LHLD   H DCX  B M MOV  H DCX  C M MOV  RP SHLD
    D INX  E C MOV  D B MOV  NEXT
 CODE EXIT     (S -- )
    RP LHLD   M C MOV   H INX   M B MOV   H INX   RP SHLD
    NEXT     END-CODE
 CODE UNNEST   ' EXIT @-T ' UNNEST !-T   END-CODE
 ASSEMBLER LABEL DODOES
    RP LHLD   H DCX B M MOV H DCX C M MOV
    RP SHLD   B POP   D INX   D PUSH   NEXT
 LABEL DOCREATE
    D INX  D PUSH   NEXT
```

```
      5
 \ Run Time Code for Defining Words                     09MAR83HH
 VARIABLE UP
 ASSEMBLER LABEL @USER   ( in: DE   out: DE  uses: HL )
    UP LHLD D DAD   M E MOV H INX M D MOV RET
 LABEL !USER   ( in: DE=off HL=value   out: none )
    H PUSH   UP LHLD D DAD   D POP
    E M MOV H INX D M MOV   RET
 LABEL DOCONSTANT
    D INX  XCHG  M E MOV  H INX  M D MOV  D PUSH  NEXT
 LABEL DOUSER-VARIABLE
    D INX  XCHG  M E MOV  H INX  M D MOV
    UP LHLD  D DAD  H PUSH  NEXT
 CODE (LIT)   (S -- n )
      IP LDAX  IP INX  A L MOV  IP LDAX  IP INX  A H MOV
      HPUSH JMP  END-CODE
```

Listing 2. Kernel of ROM target systems

```
      6
 0 \ Meta Defining Words
 1 T: LITERAL   (S n -- )
 2    [TARGET] (LIT)   ,-T   T;
 3 T: DLITERAL  (S d -- )
 4    [TARGET] (LIT) ,-T   [TARGET] (LIT) ,-T   T;
 5 T: ASCII     (S -- )
 6    [COMPILE] ASCII   [[ TRANSITION ]] LITERAL [META]  T;
 7 T: ['']   (S -- )
 8    'T >BODY @   [[ TRANSITION ]] LITERAL  [META]   T;
 9 : CONSTANT   (S n -- )
10    RECREATE   [[ ASSEMBLER DOCONSTANT ]] LITERAL ,-T
11    DUP ,-T   CONSTANT   ;
12
13
14
15
```

```
      7
 0 \ Identify numbers and forward References
 1 FORWARD: <<;CODE>>
 2 T: DOES>     (S -- )
 3    [FORWARD] <<;CODE>>   HERE-T
 4    DOES-OF C,-T   [[ ASSEMBLER DODOES ]] LITERAL ,-T   T;
 5 : NUMERIC   (S -- )
 6    [FORTH] HERE [META] NUMBER   DPL @ 1+ IF
 7       [[ TRANSITION ]] DLITERAL [META]
 8    ELSE   DROP   [[ TRANSITION ]] LITERAL [META]   THEN  ;
 9 : UNDEFINED   (S -- )
10    HERE-T   0 ,-T
11    IN-FORWARD  [FORTH] CREATE [META] TRANSITION
12    [FORTH] ,   FALSE ,   [META]
13    DOES>  FORWARD-CODE   ;
14
15
```

```
      8
 0 \ Meta Compiler Compiling Loop
 1 [FORTH] VARIABLE T-IN      META
 2 : ]   (S -- )
 3    STATE-T ON   IN-TRANSITION   BEGIN  >IN @ T-IN !
 4    DEFINED IF   EXECUTE   ELSE
 5       COUNT NUMERIC? IF   NUMERIC   ELSE
 6          T-IN @ >IN !   UNDEFINED   THEN THEN
 7    STATE-T @ 0= UNTIL   ;
 8 T: [   (S -- )
 9    IN-META   STATE-T OFF   T;
10 T: ;   (S -- )
11    [TARGET] UNNEST   [[ TRANSITION ]] [   T;
12 : :   (S -- )
13    TARGET-CREATE   [[ ASSEMBLER NEST ]] LITERAL ,-T   ]   ;
14
15
```

```
      9
07SEP83HHL \ Run Time Code for Control Structures                04MAR83hh
        CODE BRANCH   (S -- )
             IP>HL   M C MOV H INX M B MOV   NEXT   END-CODE
        CODE ?BRANCH  (S f -- )
             H POP   L A MOV   H ORA   ' BRANCH @-T JZ
             IP INX   IP INX     NEXT   END-CODE
```

```
      10
04Apr84map \ Meta Compiler Branching Words                       01AUG83Hh
        T: BEGIN   ?<MARK   T;
        T: AGAIN   [TARGET] BRANCH   ?<RESOLVE   T;
        T: UNTIL   [TARGET] ?BRANCH  ?<RESOLVE   T;
        T: IF      [TARGET] ?BRANCH  ?>MARK      T;
        T: THEN    ?>RESOLVE    T;
        T: ELSE
             [TARGET] BRANCH    ?>MARK   2SWAP ?>RESOLVE   T;
        T: WHILE   [[ TRANSITION ]] IF   T;
        T: REPEAT
           2SWAP   [[ TRANSITION ]] AGAIN   THEN   T;
```

```
      11
04MAR83HHL \ Run Time Code for Control Structures                07JUL83HHL
        ASSEMBLER LABEL LOOP-EXIT
             RP LHLD  6 D LXI   D DAD   RP SHLD
             IP INX IP INX   NEXT
        CODE (LOOP)   (S -- )
          RP LHLD   M INR 0= IF   H INX M INR LOOP-EXIT JZ
          THEN   ' BRANCH @-T JMP   END-CODE
        LABEL LOOP-BRANCH
             XCHG RP LHLD E M MOV H INX D M MOV ' BRANCH @-T JMP
        CODE (+LOOP)   (S n -- )
          RP LHLD   M E MOV H INX M D MOV
          H POP  H A MOV A ORA 0< NOT IF
             D DAD LOOP-EXIT JC  LOOP-BRANCH JMP   THEN
          D DAD  LOOP-BRANCH JC  LOOP-EXIT JMP  END-CODE
```

Listing 2. Kernel of ROM target systems (cont'd)

```
     12
 0 \ Run Time Code for Control Structures
 1 : (DO)   (S n1 n2 -- )
 2    R> DUP @ >R 2+ -ROT    SWAP DUP >R   - >R   >R   ;
 3 : (?DO)   (S n1 n2 -- )
 4    2DUP = IF   2DROP R> @ >R
 5    ELSE    R> DUP @ >R 2+ -ROT
 6            SWAP DUP >R   - >R   >R   THEN ;
 7 : BOUNDS   (S adr len -- lim first )
 8    OVER + SWAP   ;
 9
10
11
12
13
14
15
```

```
     13
 0 \ Meta compiler Branching & Looping
 1 T: ?DO
 2    [TARGET] (?DO)   ?>MARK   T;
 3 T: DO
 4    [TARGET] (DO)    ?>MARK   T;
 5 T: LOOP
 6    [TARGET] (LOOP)    2DUP 2+   ?<RESOLVE   ?>RESOLVE   T;
 7 T: +LOOP
 8    [TARGET] (+LOOP)   2DUP 2+   ?<RESOLVE   ?>RESOLVE   T;
 9
10
11
12
13
14
15
```

```
     14
 0 \ Execution Control
 1 ASSEMBLER >NEXT META CONSTANT >NEXT
 2 CODE EXECUTE   (S cfa -- )
 3    H POP      >NEXT1 JMP   END-CODE
 4 CODE PERFORM   (S addr-of-cfa -- )
 5    H POP  M E MOV  H INX  M D MOV  XCHG  >NEXT1 JMP  END-CODE
 6 LABEL DODEFER   (S -- )
 7    D INX  XCHG  ' PERFORM @-T 1+ JMP
 8 LABEL DOUSER-DEFER
 9    D INX  XCHG  M E MOV  H INX  M D MOV
10    @USER CALL  XCHG  >NEXT1 JMP
11 CODE GO       (S addr -- )
12    RET   END-CODE
13 CODE NOOP   NEXT   END-CODE
14 CODE PAUSE  NEXT   END-CODE
15
```

```
     15
02MAR83HHL \ Execution Control                                  01Oct83mac
CODE I   (S -- n )
   RP LHLD   M E MOV H INX M D MOV
   H INX M A MOV H INX M H MOV   A L MOV   D DAD
   HPUSH JMP   END-CODE
CODE J   (S -- n )
   RP LHLD   6 D LXI   D DAD   ' I @-T 3 + JMP   END-CODE
CODE (LEAVE)   (S -- )
   RP LHLD   H INX H INX H INX H INX
   M C MOV H INX M B MOV H INX   RP SHLD   NEXT   END-CODE
CODE (?LEAVE)   (S f -- )
   H POP   H A MOV L ORA ' (LEAVE) @-T JNZ   NEXT   END-CODE
T: LEAVE    [TARGET] (LEAVE)   T;
T: ?LEAVE   [TARGET] (?LEAVE)  T;
```

```
     16
01AUG83HHL \ 16 and 8 bit Memory Operations                     24FEB83HHL
CODE @     (S addr -- n )
   H POP   M E MOV   H INX   M D MOV   D PUSH   NEXT  END-CODE
CODE !     (S n addr -- )
   H POP   D POP   E M MOV   H INX   D M MOV   NEXT  END-CODE
CODE C@     (S addr -- char )
   H POP   M L MOV   0 H MVI   HPUSH JMP  END-CODE
CODE C!     (S char addr -- )
   H POP   D POP   E M MOV   NEXT  END-CODE
```

```
     17
07SEP83HHL \ Block Move Memory Operations                       24FEB83HHL
CODE CMOVE      (S  from to count -- )
   IP>HL   B POP  D POP  XTHL ( STACK=IP BC=len DE=to HL=from )
   BEGIN   B A MOV  C ORA  0= NOT WHILE
      M A MOV  H INX  D STAX  D INX  B DCX
   REPEAT  B POP  NEXT  END-CODE
CODE CMOVE>   (S from to count -- )
   IP>HL   B POP  D POP  XTHL ( STACK=IP BC=len DE=to HL=from )
   B DAD H DCX   XCHG   B DAD H DCX    XCHG
   BEGIN   B A MOV   C ORA   0= NOT WHILE
      M A MOV H DCX D STAX   D DCX B DCX
   REPEAT   B POP   NEXT   END-CODE
```

Listing 2. Kernel of ROM target systems (cont'd)

```
      18
 0 \ 16 bit Stack Operations
 1 CODE SP@     (S -- n )
 2      0 H LXI   SP DAD   HPUSH JMP   END-CODE
 3 CODE SP!     (S n -- )
 4      H POP    SPHL   NEXT   END-CODE
 5 CODE RP@     (S -- addr )
 6      RP LHLD   HPUSH JMP   END-CODE
 7 CODE RP!     (S n -- )
 8      H POP   RP SHLD   NEXT   END-CODE
 9
10
11
12
13
14
15
```

```
      19
 0 \ 16 bit Stack Operations
 1 CODE DROP    (S n1 -- )
 2      H POP   NEXT   END-CODE
 3 CODE DUP     (S n1 -- n1 n1 )
 4    H POP   H PUSH   HPUSH JMP    END-CODE
 5 CODE SWAP    (S n1 n2 -- n2 n1 )
 6    H POP   XTHL   HPUSH JMP    END-CODE
 7 CODE OVER    (S n1 n2 -- n1 n2 n1 )
 8    D POP   H POP   H PUSH   DPUSH JMP   END-CODE
 9
10
11
12
13
14
15
```

```
      20
 0 \ 16 bit Stack Operations
 1 CODE TUCK    (S n1 n2 -- n2 n1 n2 )
 2    H POP   D POP   H PUSH   DPUSH JMP   END-CODE
 3 CODE NIP     (S n1 n2 -- n2 )
 4    H POP  D POP  HPUSH JMP   END-CODE
 5 CODE ROT   (S n1 n2 n3 --- n2 n3 n1 )
 6    D POP H POP XTHL DPUSH JMP END-CODE
 7 CODE -ROT   (S n1 n2 n3 --- n3 n1 n2 )
 8    H POP D POP XTHL XCHG DPUSH JMP END-CODE
 9 CODE FLIP   (S n -- n )
10    D POP   E H MOV  D L MOV  HPUSH JMP   END-CODE
11 : ?DUP      (S n -- [n] n )
12    DUP IF   DUP   THEN   ;
13
14
15
```

```
      21
24FEB83HHL \ 16 bit Stack Operations                          24FEB83HHL
          CODE R>     (S -- n )
             RP LHLD   M E MOV   H INX   M D MOV   H INX
             RP SHLD   D PUSH   NEXT   END-CODE
          CODE >R     (S n -- )
             D POP   RP LHLD   H DCX   H DCX   RP SHLD
             E M MOV   H INX   D M MOV   NEXT   END-CODE
          CODE R@
             RP LHLD   M E MOV   H INX   M D MOV   D PUSH   NEXT END-CODE
          CODE PICK    (S n# ... n2 n1 k -- n# ... n2 n1 nk )
             H POP H DAD SP DAD   M E MOV H INX M D MOV
             D PUSH NEXT END-CODE
          : ROLL   (S n1 n2 .. nk n -- wierd )
             >R R@ PICK   SP@ DUP 2+   R> 1+ 2* CMOVE>  DROP  ;
```

```
      22
24FEB83HHL \ 16 bit Logical Operations                        13Apr84##
          CODE AND     (S n1 n2 -- n3 )
             D POP   H POP   E A MOV   L ANA   A L MOV
             D A MOV   H ANA   A H MOV   HPUSH JMP  END-CODE
          CODE OR      (S n1 n2 -- n3 )
             D POP   H POP   E A MOV   L ORA   A L MOV
             D A MOV   H ORA   A H MOV   HPUSH JMP  END-CODE
          CODE XOR     (S n1 n2 -- n3 )
             D POP   H POP   E A MOV   L XRA   A L MOV
             D A MOV   H XRA   A H MOV   HPUSH JMP  END-CODE
          CODE NOT     (S n -- n' )
            H POP  L A MOV  CMA  A L MOV  H A MOV  CMA  A H MOV
            HPUSH JMP  END-CODE
          -1 CONSTANT TRUE   0 CONSTANT FALSE
          ASSEMBLER      LABEL YES       TRUE H LXI   HPUSH JMP
                         LABEL NO       FALSE H LXI   HPUSH JMP
```

```
      23
11MAR83HHL \ Logical Operations                               16Oct83##
          CODE CSET   (S b addr -- )
            H POP D POP   M A MOV  E ORA  A M MOV   NEXT   END-CODE
          CODE CRESET   (S b addr -- )
            H POP D POP   E A MOV CMA A E MOV
                          M A MOV  E ANA  A M MOV   NEXT   END-CODE
          CODE CTOGGLE  (S b addr -- )
            H POP   D POP   M A MOV   E XRA   A M MOV   NEXT   END-CODE
          CODE ON   (S addr -- )
            TRUE  H LXI XTHL H PUSH ' ! @-T JMP   END-CODE
          CODE OFF   (S addr -- )
            FALSE H LXI XTHL H PUSH ' ! @-T JMP   END-CODE
```

Listing 2. Kernel of ROM target systems (cont'd)

```
    24
 0 \ 16 bit Arithmetic Operations                        13Apr84map
 1 CODE +   (S n1 n2 -- sum )
 2    D POP   H POP   D DAD   HPUSH JMP   END-CODE
 3 CODE NEGATE   (S n -- n' )
 4    H POP  H DCX  H PUSH  ' NOT @-T JMP  END-CODE
 5 CODE -       (S n1 n2 -- n1-n2 )
 6    D POP H POP   D A MOV CMA A D MOV   E A MOV CMA A E MOV
 7                  D INX    D DAD   HPUSH JMP   END-CODE
 8 CODE ABS   (S n -- n )
 9    H POP H PUSH   H A MOV A ORA '  NEGATE @-T JM  NEXT END-CODE
10 CODE +!   (S n addr -- )
11    H POP   D POP   M A MOV   E ADD   A M MOV
12            H INX   M A MOV   D ADC   A M MOV   NEXT   END-CODE
13 0 CONSTANT 0      1 CONSTANT 1
14 2 CONSTANT 2      3 CONSTANT 3
15
```

```
    25
 0 \ 16 bit Arithmetic Operations                        26Sep83map
 1 CODE 2*   (S n -- 2*n )
 2    H POP   H DAD   HPUSH JMP   END-CODE
 3 CODE 2/   (S n -- n/2 )
 4    H POP   H A MOV   RLC RRC RAR   A H MOV
 5    L A MOV   RAR   A L MOV   HPUSH JMP   END-CODE
 6 CODE U2/   (S u -- u/2 )
 7    H POP   A ORA   H A MOV   RAR   A H MOV
 8    L A MOV   RAR   A L MOV   HPUSH JMP   END-CODE
 9 CODE 8*   (S n -- 8*n )
10    H POP   H DAD   H DAD   H DAD   HPUSH JMP   END-CODE
11 CODE 1+      H POP H INX HPUSH JMP END-CODE
12 CODE 2+      H POP H INX H INX HPUSH JMP END-CODE
13 CODE 1-      H POP H DCX HPUSH JMP END-CODE
14 CODE 2-      H POP H DCX H DCX HPUSH JMP END-CODE
15
```

```
    26
 0 \ 16 bit Arithmetic Operations  Unsigned Multiply   26Sep83map
 1  ASSEMBLER
 2 LABEL MPYX   0 H LXI   ( 0=Partial Product )
 3              4 C MVI   ( Loop Counter )
 4       BEGIN  H DAD ( Shift AHL left by 24 bits )
 5              RAL CS IF   D DAD   0 ACI   THEN
 6              H DAD   RAL   CS IF   D DAD   0 ACI   THEN
 7       C DCR  0= UNTIL   RET
 8
 9 CODE UM*      (S n1 n2 -- d )
10      D POP      H POP     B PUSH   H B MOV   L A MOV   MPYX CALL
11      H PUSH   A H MOV   B A MOV    H B MOV            MPYX CALL
12      D POP    D C MOV     B DAD      0 ACI   L D MOV    H L MOV
13    A H MOV      B POP     DPUSH JMP END-CODE
14 : U*D   (S n1 n2 -- d )   UM* ;
15
```

```
    27
13Apr84map \ 16 bit Arithmetic Operations  Division subroutines 25FEB83HHL
ASSEMBLER
LABEL USL0
   A E MOV H A MOV C SUB   A H MOV E A MOV B SBB
   CS IF
      H A MOV C ADD   A H MOV E A MOV D DCR RZ
LABEL USLA
      H DAD RAL   USL0 JNC
      A E MOV H A MOV C SUB   A H MOV E A MOV B SBB
   THEN
   L INR D DCR   USLA JNZ   RET
LABEL USBAD
    -1 H LXI   B POP   H PUSH   HPUSH JMP
```

```
    28
26Sep83map \ 16 bit Arithmetic Operations  Unsigned Divide      25FEB83HHL
CODE UM/MOD   (S d1 n1 -- Remainder Quotient )
   IP>HL    B POP D POP XTHL XCHG
   ( HLDE = Numerator  BC = Denominator )
   L A MOV C SUB   H A MOV B SBB   USBAD JNC
   H A MOV L H MOV D L MOV B D MVI D PUSH
   USLA CALL
   D POP H PUSH E L MOV
   USLA CALL
   A D MOV H E MOV B POP C H MOV B POP
   D PUSH HPUSH JMP END-CODE
```

```
    29
13Apr84map \ 16 bit Comparison Operations
CODE 0=      (S n -- f )
   H POP   L A MOV   H ORA   YES JZ   NO JMP  END-CODE
CODE 0<      (S n -- f )
   H POP   H DAD   YES JC   NO JMP   END-CODE
CODE 0>   (S n -- f )
   H POP H A MOV A ORA  NO JM   L ORA YES JNZ   NO JMP END-CODE
CODE 0<>  (S n -- f )
   H POP   L A MOV   H ORA   YES JNZ  NO JMP  END-CODE
CODE =       (S n1 n2 -- f )
   H POP D POP   L A MOV E CMP   NO JNZ
                 H A MOV D CMP   NO JNZ   YES JMP   END-CODE
: <>     (S n1 n2 -- f )   = NOT   ;
: ?NEGATE    (S n1 n2 -- n3 )   0< IF    NEGATE   THEN   ;
```

Listing 2. Kernel of ROM target systems (cont'd)

```
    30
 0 \ 16 bit Comparison Operations                    13Apr84map
 1 CODE U<   (S n1 n2 -- f )   H POP  D POP
 2 LABEL U<1    H A MOV
 3 LABEL U<2            D CMP   NO JC   YES JNZ
 4    L A MOV  E CMP   NO JC   YES JNZ   NO JMP  END-CODE
 5 CODE U>   (S n1 n2 -- f )   D POP  H POP  U<1 JMP  END-CODE
 6 CODE <   (S n1 n2 -- f )   H POP  D POP
 7 LABEL <1   D A MOV 128 XRI A D MOV  H A MOV 128 XRI
 8    U<2 JMP END-CODE
 9 CODE >   (S n1 n2 -- f )   D POP  H POP  <1 JMP   END-CODE
10 : MIN   (S n1 n2 -- n3 )   2DUP > IF   SWAP   THEN   DROP   ;
11 : MAX   (S n1 n2 -- n3 )   2DUP < IF   SWAP   THEN   DROP   ;
12 : BETWEEN   (S n1 min max -- f )
13    >R  OVER > SWAP R> > OR NOT  ;
14 : WITHIN   (S n1 min max -- f )   1- BETWEEN  ;
15
```

```
    31
 0 \ 32 bit Memory Operations                        09MAR83HHL
 1 CODE 2@     (S addr -- d )
 2    H POP   2 D LXI   D DAD   M E MOV   H INX   M D MOV   D PUSH
 3           -3 D LXI   D DAD   M E MOV   H INX   M D MOV   D PUSH
 4           NEXT   END-CODE
 5 CODE 2!     (S d addr -- )
 6    H POP   D POP   E M MOV   H INX   D M MOV   H INX
 7           D POP   E M MOV   H INX   D M MOV   NEXT  END-CODE
 8
 9
10
11
12
13
14
15
```

```
    32
 0 \ 32 bit Memory and Stack Operations              13Apr84map
 1 CODE 2DROP     (S d -- )
 2    H POP H POP   NEXT   END-CODE
 3 CODE 2DUP     (S d -- d d )
 4    H POP D POP   D PUSH H PUSH   DPUSH JMP   END-CODE
 5 CODE 2SWAP     (S d1 d2 -- d2 d1 )
 6    H POP D POP XTHL H PUSH
 7    5 M LXI SP DAD M A MOV D M MOV A D MOV
 8    H DCX M A MOV E M MOV A E MOV  H POP DPUSH JMP END-CODE
 9 CODE 2OVER      (S d2 d2 -- d1 d2 d1 )
10    7 M LXI SP DAD M D MOV H DCX M E MOV D PUSH
11           H DCX M D MOV H DCX M E MOV D PUSH   NEXT  END-CODE
12 : 3DUP  (S a b c -- a b c a b c )        DUP 2OVER ROT   ;
13 : 4DUP  (S a b c d -- a b c d a b c d )  2OVER 2OVER   ;
14 : 2ROT  (S a b c d e f --- c d e f a b )  5 ROLL 5 ROLL ;
15
```

```
    33
 0 \ 32 bit Arithmetic Operations                    13Apr84mac
 1 CODE D+  (S d1 d2 -- dsum )
 2    6 M LXI  SP DAD  M E MOV C M MOV  H INX
 3    M D MOV  B M MOV  B POP  H POP  D DAD  XCHG
 4    H POP  L A MOV  C ADC  A L MOV  H A MOV  B ADC
 5    A H MOV  B POP  DPUSH JMP   END-CODE
 6 CODE DNEGATE  (S d# -- d#' )
 7    H POP  D POP  A SUB  E SUB  A E MOV  0 A MVI  D SBB
 8    A D MOV  0 A MVI     L SBB  A L MOV  0 A MVI  H SBB
 9    A H MOV  DPUSH JMP  END-CODE
10 CODE  S>D      (S n -- d )
11    D POP 0 H LXI D A MOV 128 ANI 0= NOT IF   H DCX   THEN
12    DPUSH JMP   END-CODE
13 CODE DABS   (S d# -- d# )
14    H POP H PUSH   H A MOV A ORA ' DNEGATE 0<> JM   NEXT END-CODE
15
```

```
    34
 0 \ 32 bit Arithmetic Operations                    06Apr84mac
 1 CODE D2#   (S d -- d#2 )
 2    H POP   D POP
 3    E A MOV   RAL   A E MOV   D A MOV   RAL   A D MOV
 4    L A MOV   RAL   A L MOV   H A MOV   RAL   A H MOV
 5    DPUSH JMP   END-CODE
 6 CODE D2/   (S d -- d/2 )
 7    H POP   D POP
 8    H A MOV   RLC RRC RAL   A H MOV   L A MOV   RAL   A L MOV
 9    D A MOV   RAL           A D MOV   E A MOV   RAL   A E MOV
10    DPUSH JMP   END-CODE
11 : D-    (S d1 d2 -- d3 )   DNEGATE D+   ;
12 : ?DNEGATE  (S d1 n -- d2 )     0< IF   DNEGATE   THEN   ;
13
14
15
```

```
    35
 0 \ 32 bit Comparison Operations                    05Oct83mac
 1 : D0=   (S d -- f )        0= 0= ;
 2 : D=    (S d1 d2 -- f )    D-  D0=  ;
 3 : DU<   (S ud1 ud2 -- f )   ROT SWAP 2DUP U<
 4    IF   2DROP 2DROP TRUE
 5    ELSE  <> IF   2DROP FALSE  ELSE  U<  THEN
 6    THEN  ;
 7 : D<    (S d1 d2 -- f )   2 PICK OVER =
 8    IF   DU<   ELSE  NIP ROT DROP <  THEN  ;
 9 : D>    (S d1 d2 -- f )    2SWAP D<   ;
10 : DMIN  (S d1 d2 -- d3 )   4DUP D> IF   2SWAP   THEN   2DROP ;
11 : DMAX  (S d1 d2 -- d3 )   4DUP D< IF   2SWAP   THEN   2DROP ;
12
13
14
15
```

Listing 2. Kernel of ROM target systems (cont'd)

```
    36
 0 \ Mixed Mode Arithmetic                              01Oct83map
 1 : M*   (S n1 n2 -- d# )
 2   2DUP  XOR  >R  ABS  SWAP  ABS  UM*  R>  ?DNEGATE  ;
 3 : M/MOD   (S d# n1 -- rem quot )
 4   ?DUP
 5   IF  DUP >R  2DUP XOR >R  >R DABS R@ ABS  UM/MOD
 6     SWAP R> ?NEGATE
 7     SWAP R> 0< IF  NEGATE DUP ,   1- R@ ROT - SWAP  THEN THEN
 8     R> DROP
 9   THEN  ;
10 : MU/MOD  (S d# n1 -- rem d#quot )
11   >R  0  R@  UM/MOD  R>  SWAP  >R  UM/MOD  R>  ;
12
13
14
15
```

```
    37
 0 \ 16 bit multiply and divide                         27Sep83map
 1 : *   (S n1 n2 -- n3 )   UM* DROP  ;
 2 : /MOD  (S n1 n2 -- rem quot )   >R  S>D  R>  M/MOD  ;
 3 : /     (S n1 n2 -- quot )   /MOD  NIP  ;
 4 : MOD   (S n1 n2 -- rem )    /MOD  DROP  ;
 5 : */MOD  (S n1 n2 n3 -- rem quot )
 6   >R  M*  R>  M/MOD  ;
 7 : */    (S n1 n2 n3 -- n1*n2/n3 )     */MOD  NIP  ;
 8
 9
10
11
12
13
14
15
```

```
    38
 0 \ Task Dependant USER Variables                      24Mar84map
 1 USER DEFINITIONS
 2 VARIABLE  TOS         ( TOP OF STACK )
 3 VARIABLE  ENTRY       ( ENTRY POINT, CONTAINS MACHINE CODE )
 4 VARIABLE  LINK        ( LINK TO NEXT TASK )
 5 VARIABLE  SP0         ( INITIAL PARAMETER STACK )
 6 VARIABLE  RP0         ( INITIAL RETURN STACK )
 7 VARIABLE  DP          ( DICTIONARY POINTER )
 8 VARIABLE  #OUT        ( NUMBER OF CHARACTERS EMITTED )
 9 VARIABLE  #LINE       ( THE NUMBER OF LINES SENT SO FAR )
10 VARIABLE  OFFSET      ( RELATIVE TO ABSOLUTE DISK BLOCK 0 )
11 VARIABLE  BASE        ( FOR NUMERIC INPUT AND OUTPUT )
12 VARIABLE  HLD         ( POINTS TO LAST CHARACTER HELD IN PAD )
13 VARIABLE  FILE        ( POINTS TO FCB OF CURRENTLY OPEN FILE )
14 VARIABLE  IN-FILE     ( POINTS TO FCB OF CURRENTLY OPEN FILE )
15 VARIABLE  PRINTING
```

```
    39
 0 \ System VARIABLEs                                   14Jun85cht
 1   DEFER EMIT   DEFER KEY   DEFER KEY?
 2   META DEFINITIONS
 3   VARIABLE  SCR       ( SCREEN LAST [illegible] OR EDITED )
 4   VARIABLE  PRIOR     ( USED F[illegible]R   TIONARY SEARCHES )
 5   VARIABLE  STATE     ( COMPILATION OR INTERPRETATION )
 6   VARIABLE  WARNING   ( GIVE USER DUPLICATE WARNINGS IF ON )
 7   VARIABLE  DPL       ( NUMERIC INPUT PUNCTUATION )
 8   VARIABLE  R#        ( EDITING CURSOR POSITION )
 9   VARIABLE  LAST      ( POINTS TO NFA OF LATEST DEFINITION )
10   VARIABLE  CSP       ( HOLDS STACK POINTER FOR ERROR CHECKING )
11   VARIABLE  CURRENT   ( VOCABULARY WHICH GETS DEFINITIONS )
12   8 CONSTANT #VOCS    ( THE NUMBER OF VOCABULARIES TO SEARCH )
13   VARIABLE  CONTEXT   ( VOCABULARY SEARCHED FIRST )
14     #VOCS 2* ALLOT-R
15
```

```
    40
 0 \ System Variables                                   24JAN87CHT
 1   VARIABLE  'TIB      ( ADDRESS OF TERMINAL INPUT BUFFER )
 2   VARIABLE  WIDTH     ( WIDTH OF NAME FIELD )
 3   VARIABLE  VOC-LINK  ( POINTS TO NEWEST VOCABULARY )
 4   VARIABLE  BLK       ( BLOCK NUMBER TO INTERPRET )
 5   VARIABLE  >IN       ( OFFSET INTO INPUT STREAM )
 6   VARIABLE  SPAN      ( NUMBER OF CHARACTERS EXPECTED )
 7   VARIABLE  #TIB      ( NUMBER OF CHARACTERS TO INTERPRET )
 8   VARIABLE  END?      ( TRUE IF INPUT STREAM EXHAUSTED )
 9   VARIABLE CAPS       ( Block 41)
10   VARIABLE CC         ( Block 48)
11   VARIABLE FENCE      ( Block 72)
12   VARIABLE AVOC       ( Block 80)
13   VARIABLE #USER      ( Block 81)
14   VARIABLE FOUND      ( EDITOR)
15
```

```
    41
 0 \ Devices                    Strings                 23Jun85cht
 1   32 CONSTANT BL      8 CONSTANT BS        7 CONSTANT BELL
 2   \ VARIABLE CAPS
 3   CODE FILL         ( start-addr count char -- )
 4     IP>HL  D POP  B POP  XTHL  XCHG
 5     BEGIN  B A MOV  C ORA  0= NOT WHILE
 6       L A MOV  D STAX  D INX  B DCX
 7     REPEAT  B POP  NEXT  END-CODE
 8   : ERASE     (S addr len -- )  0 FILL  ;
 9   : BLANK     (S addr len -- )  BL FILL  ;
10   CODE COUNT  (S addr -- addr+1 len )
11     H POP  M E MOV 0 D MVI  H INX  XCHG  DPUSH JMP END-CODE
12   : MOVE  ( from to len -- )
13     -ROT  2DUP U< IF  ROT CMOVE>  ELSE  ROT CMOVE  THEN ;
14
15
```

Listing 2. Kernel of ROM target systems (cont'd)

```
    42
 0 \ Devices                    Strings              21FEB85CHT
 1 \ ASSEMBLER LABEL >UPPER
 2 \    ASCII a CPI  RC  ASCII z 1+ CPI  RNC  BL SUI  RET
 3 \ CODE UPC   (S char -- char' )
 4 \   H POP  L A MOV  >UPPER CALL  A L MOV  H PUSH  NEXT  END-CODE
 5 \ CODE UPPER  (S addr len -- )
 6 \    D POP H POP   BEGIN      D A MOV E ORA 0= NOT WHILE
 7 \       M A MOV  >UPPER CALL   A M MOV
 8 \    H INX D DCX REPEAT   NEXT   END-CODE
 9 : HERE   (S -- addr )   DP @  ;
10 : PAD    (S -- addr )   HERE 80 +  ;
11 : -TRAILING   (S addr len -- addr len' )
12    DUP 0 ?DO   2DUP + 1- C@   BL <> ?LEAVE   1-   LOOP   ;
13
14
15
```

```
    43
 0 \ Devices                    Strings              06JUL86CHT
 1 CODE COMPARE   (S addr1 addr2 len -- -1 | 0 | 1 )
 2    C L MOV  B H MOV  B POP  D POP  XTHL
 3    ( Stack=IP  BC=len  DE=addr2  HL=addr1 )
 4    BEGIN   B A MOV  C ORA  0= NOT WHILE
 5       M A MOV  XCHG  M CMP  XCHG
 6       0= IF   D INX  H INX  B DCX
 7          ELSE    0< IF   1 H LXI   ELSE   -1 H LXI   THEN
 8                  B POP   HPUSH JMP   THEN
 9    REPEAT   0 H LXI   B POP   HPUSH JMP   END-CODE
10
11
12
13
14
15
```

```
    44
 0 \S Devices                   Strings              21FEB85CHT
 1 CODE CAPS-COMP   (S addr1 addr2 len -- -1 | 0 | 1 )
 2    C L MOV  B H MOV  B POP  D POP  XTHL
 3    ( Stack=IP  BC=len  DE=addr2  HL=addr1 )
 4    BEGIN   B A MOV  C ORA  0= NOT WHILE
 5          M A MOV  >UPPER CALL  B PUSH  A C MOV  XCHG
 6          M A MOV  >UPPER CALL  C CMP   B POP    XCHG
 7       0= IF   D INX  H INX  B DCX
 8          ELSE    0< IF   1 H LXI   ELSE   -1 H LXI   THEN
 9                  B POP   HPUSH JMP   THEN
10    REPEAT   0 H LXI   B POP   HPUSH JMP   END-CODE
11
12 : COMPARE   (S addr1 addr2 len -- -1 | 0 | 1 )
13    CAPS @ IF  CAPS-COMP  ELSE  COMP  THEN  ;
14
15
```

```
    45
 0 \ MA2232 I/O                                          27aug86cht
 1 CODE (INIT) HEX   DI A XRA 7 OUT 22 OUT CMA 4 OUT 5 OUT 6 OUT
 2    0 OUT 1 OUT 2 OUT   1D A MVI   22 OUT   99 A MVI   22 OUT
 3    1 A MVI   24 OUT   NEXT   END-CODE
 4 CODE (KEY)
 5    BEGIN   23 IN   1 ANI   0<> UNTIL
 6    21 IN   7F ANI   0 H MVI   A L MOV
 7    HPUSH JMP   END-CODE
 8 CODE (EMIT)
 9    BEGIN   23 IN   80 ANI   0<> UNTIL
10    H POP   L A MOV   20 OUT   NEXT   END-CODE
11 CODE (KEY?)
12    23 IN   1 ANI   0<> IF   OFF A MVI   ELSE A XRA   THEN
13    A H MOV   A L MOV   HPUSH JMP   END-CODE
14 : (CONSOLE)    EMIT 1 #OUT +!   ;
15 DECIMAL
```

```
    46
 0 \ Devices                    Terminal Input and Output   10jan87cht
 1 \ DEFER KEY?
 2 \ DEFER KEY
 3 DEFER CR
 4 : CRLF   (S -- )   13 EMIT   10 EMIT   #OUT OFF   1 #LINE +! ;
 5 : TYPE   (S addr len -- )   0 ?DO   COUNT EMIT   LOOP   DROP   ;
 6 : SPACE   (S -- )     BL EMIT   ;
 7 : SPACES (S n -- )   0 MAX   0 ?DO   SPACE   LOOP   ;
 8 : BACKSPACES   (S n -- )     0 ?DO   BS EMIT   LOOP   ;
 9 : BEEP   (S -- )     BELL EMIT   ;
10 CODE BDOS   H POP   D POP   B PUSH   L C MOV   5 CALL
11    0 H MVI   A L MOV   B POP   HPUSH JMP   END-CODE
12
13
14
15
```

```
    47
 0 \ Devices   System Dependent Control Characters       02Apr84cht
 1 : BS-IN   (S n c -- 0 | n-1 )
 2    DROP DUP IF   1-   BS   ELSE   BELL   THEN   EMIT   ;
 3 : (DEL-IN)   (S n c -- 0 | n-1 )
 4    DROP DUP IF  1-  BS EMIT SPACE BS  ELSE  BELL  THEN  EMIT  ;
 5 : BACK-UP (S n c -- 0 )
 6    DROP   DUP BACKSPACES   DUP SPACES   BACKSPACES   0   ;
 7 : RES-IN   (S c -- )
 8    FORTH   TRUE ABORT" Reset"  ;
 9 : P-IN  (S c -- )
10    DROP   PRINTING @ NOT PRINTING !  ;
11
12
13
14
15
```

Listing 2. Kernel of ROM target systems (cont'd)

```
    48
 0 \ Devices                    Terminal Input          14Jun85cht
 1 : CR-IN (S a n c -- a n )
 2   DROP   SPAN !   OVER   BL EMIT   ;
 3 : (CHAR)   (S a n char -- a n+1 )
 4   3DUP EMIT + C!   1+   ;
 5 DEFER CHAR
 6 DEFER DEL-IN
 7
 8 \ VARIABLE CC
 9 CREATE CC-FORTH
10  ] CHAR    CHAR   CHAR   RES-IN CHAR   CHAR    CHAR   CHAR
11   BS-IN   CHAR   CHAR   CHAR   CHAR   CR-IN   CHAR   CHAR
12   P-IN    CHAR   CHAR   CHAR   CHAR   BACK-UP CHAR   CHAR
13   BACK-UP CHAR   CHAR   CHAR   CHAR   CHAR    CHAR   CHAR [
14
15
```

```
    49
 0 \ Devices                    Terminal Input          29Sep83map
 1 : EXPECT   (S adr len -- )
 2   DUP SPAN !   SWAP 0   ( len adr 0 )
 3   BEGIN   2 PICK OVER - ( len adr #so-far #left )
 4   WHILE   KEY DUP BL <
 5     IF   DUP 2# CC @ + PERFORM
 6     ELSE DUP 127 = IF   DEL-IN   ELSE   CHAR   THEN
 7     THEN REPEAT   2DROP DROP   ;
 8
 9 : TIB     (S -- adr )   'TIB @   ;
10 : QUERY   (S -- )
11   TIB 80 EXPECT  SPAN @ #TIB !   BLK OFF   >IN OFF   ;
12
13
14
15
```

```
    50
 0 \ Devices                    BLOCK I/O               24JAN87CHT
 1    4 CONSTANT #BUFFERS
 2 1024 CONSTANT B/BUF
 3  128 CONSTANT B/REC
 4    8 CONSTANT REC/BLK
 5   42 CONSTANT B/FCB
 6 HEX 8000 CONSTANT LIMIT  DECIMAL
 7       #BUFFERS 1+ 8 * 2+ CONSTANT >SIZE
 8 LIMIT B/BUF #BUFFERS * -  CONSTANT FIRST
 9 FIRST >SIZE - CONSTANT INIT-R0
10 VOCABULARY EDITOR  VOCABULARY ASSEMBLER   VOCABULARY USER
11 : BLOCK ( n --- addr )   B/BUF * FIRST + ;
12 : (LOAD) ( n --- )   BLK @ >R   >IN @ >R   >IN OFF   BLK !
13   RUN   R> >IN !   R> BLK !   ;
14 DEFER LOAD
15
```

```
    51
 0 \ Devices                    BLOCK I/O               13Apr84mac
 1 DEFER READ-BLOCK    (S buffer-header -- )
 2 DEFER WRITE-BLOCK   (S buffer-header -- )
 3 : .FILE   (S adr -- )
 4   COUNT ?DUP IF  ASCII @ + EMIT ." :"  THEN
 5   8 2DUP -TRAILING TYPE + ." ." 3 TYPE SPACE   ;
 6 : FILE?   (S -- )   FILE @ .FILE   ;
 7 : SWITCH   (S -- )   FILE @ IN-FILE @ FILE ! IN-FILE !   ;
 8
 9 VOCABULARY DOS   DOS DEFINITIONS
10 : !FILES   (S fcb -- )   DUP FILE !  IN-FILE !   ;
11 : DISK-ABORT   (S fcb a n -- )
12   TYPE ." in "  .FILE  ABORT   ;
13 : ?DISK-ERROR  (S fcb n -- )
14   DUP DISK-ERROR !
15   IF  " Disk error" DISK-ABORT  ELSE  DROP  THEN   ;
```

```
    52
 0 \ Devices                    BLOCK I/O               04Apr84mac
 1 CREATE FCB1   B/FCB ALLOT
 2 : CLR-FCB    (S fcb -- )    DUP  B/FCB ERASE 1+ 11 BLANK ;
 3 : SET-DMA    (S adr -- )   26 BDOS  DROP ;
 4 : RECORD#    (S fcb -- adr )   33 + ;
 5 : MAXREC#    (S fcb -- adr )   38 + ;
 6 : IN-RANGE   (S fcb -- fcb )
 7   DUP MAXREC# @ OVER RECORD# @ U<  DUP DISK-ERROR !
 8   IF  1 BUFFER# ON  " Out of Range" DISK-ABORT  THEN   ;
 9 : REC-READ   (S fcb -- )
10   DUP IN-RANGE  33 BDOS  ?DISK-ERROR ;
11 : REC-WRITE  (S fcb -- )
12   DUP IN-RANGE  34 BDOS  ?DISK-ERROR ;
13
14
15
```

```
    53
 0 \ Devices                    BLOCK I/O               29Mar84mac
 1 : SET-IO       (S buf-header -- file buffer rec/blk 0 )
 2   DUP 2@ REC/BLK # OVER RECORD# !
 3   SWAP 4 + @ ( buf-addr )   REC/BLK 0   ;
 4 : FILE-READ   (S buffer-header -- )
 5   SET-IO
 6   DO   2DUP SET-DMA  DUP REC-READ   1 SWAP RECORD# +!  B/REC +
 7   LOOP  2DROP   ;
 8 : FILE-WRITE   (S buffer-header -- )
 9   SET-IO
10   DO   2DUP SET-DMA  DUP REC-WRITE  1 SWAP RECORD# +!  B/REC +
11   LOOP  2DROP   ;
12 : FILE-IO   (S -- )
13   ['] FILE-READ IS READ-BLOCK  ['] FILE-WRITE IS WRITE-BLOCK ;
14
15
```

Listing 2. Kernel of ROM target systems (cont'd)

```
     54
 0 \ Devices                    BLOCK I/O              29Mar84map
 1 FORTH DEFINITIONS
 2 : CAPACITY   (S -- n )
 3   [ DOS ]   FILE @ MAXREC# @ 1+ 0 8 UM/MOD NIP ;
 4 : LATEST?   (S n fcb -- fcb n ! a f )
 5   DISK-ERROR OFF
 6   SWAP OFFSET @ + 2DUP   1 BUFFER# 2@   D=
 7   IF   2DROP   1 BUFFER# 4 + @   FALSE   R> DROP  THEN  ;
 8 : ABSENT?   (S n fcb -- a f )
 9   LATEST?  FALSE #BUFFERS 1+ 2
10   DO  DROP 2DUP I BUFFER# 2@ D=
11     IF  2DROP I LEAVE  ELSE  FALSE  THEN
12   LOOP  ?DUP
13   IF  BUFFER# DUP >BUFFERS 8 CMOVE   >R  >BUFFERS DUP 8 +
14     OVER R> SWAP  -  CMOVE>     1 BUFFER# 4 + @ FALSE
15   ELSE  >BUFFERS 2! TRUE  THEN  ;

     55
 0 \ Devices                    BLOCK I/O              01Apr84map
 1 : UPDATE   (S -- )   >UPDATE ON   ;
 2 : DISCARD  (S -- )   1 >UPDATE ! ( 1 BUFFER# ON ) ;
 3 : MISSING   (S -- )
 4   >END 2- @ 0< IF  >END 2- OFF  >END 8 - WRITE-BLOCK  THEN
 5   >END 4 - @  >BUFFERS 4 + ! ( buffer )  1 >BUFFERS 6 + !
 6   >BUFFERS DUP 8 + #BUFFERS 8* CMOVE>   ;
 7 : (BUFFER)   (S n fcb -- a )   PAUSE  ABSENT?
 8   IF  MISSING  1 BUFFER#   4 + @  THEN  ;
 9 : BUFFER   (S n -- a )   FILE @ (BUFFER)  ;
10 : (BLOCK)    (S n fcb -- a )
11   (BUFFER)  >UPDATE @ 0>
12   IF  1 BUFFER#  DUP READ-BLOCK  6 + OFF  THEN  ;
13 : BLOCK     (S n -- a )   FILE @ (BLOCK)  ;
14 : IN-BLOCK  (S n -- a )   IN-FILE @ (BLOCK)  ;
15

     56
 0 \ Devices                    BLOCK I/O              01APR84MAP
 1 : EMPTY-BUFFERS   (S -- )
 2   FIRST LIMIT OVER - ERASE
 3   >BUFFERS #BUFFERS 1+ 8* ERASE
 4   FIRST 1 BUFFER#   #BUFFERS 0
 5   DO   DUP ON  4 +  2DUP !   SWAP B/BUF + SWAP  4 +
 6   LOOP   2DROP   ;
 7 : SAVE-BUFFERS   (S -- )
 8   1 BUFFER#   #BUFFERS 0
 9   DO   DUP @ 1+
10     IF  DUP 6 + @ 0< IF  DUP WRITE-BLOCK  DUP 6 + OFF  THEN
11       8 + THEN   LOOP   DROP   ;
12 : FLUSH   (S -- )
13   SAVE-BUFFERS  0 BLOCK DROP  EMPTY-BUFFERS  ;
14 : VIEW#    (S -- addr )    FILE @ 40 +   ;
15

     57
 0 \ Devices                    BLOCK I/O              04Apr84map
 1 DOS DEFINITIONS
 2 : FILE-SIZE   (S fcb -- n )   DUP 35 BDOS  DROP  RECORD# @ ;
 3 : DOS-ERR?    (S -- f )   255 =    ;
 4 : OPEN-FILE   (S -- )   IN-FILE @ DUP 15 BDOS DOS-ERR?
 5   IF  " Open error" DISK-ABORT  THEN
 6   DUP FILE-SIZE 1-  SWAP MAXREC# !  ;
 7 HEX 5C CONSTANT DOS-FCB   DECIMAL
 8 FORTH DEFINITIONS
 9 : DEFAULT    (S -- )   [ DOS ]   FCB1 DUP IN-FILE !  DUP FILE !
10   CLR-FCB   DOS-FCB 1+ C@ BL <>
11   IF   DOS-FCB FCB1 12 CMOVE  OPEN-FILE   THEN   ;
12 : (LOAD)     (S n -- )   FILE @ >R   BLK @ >R   >IN @ >R
13   >IN OFF  BLK !   IN-FILE @ FILE !   RUN   R> >IN !   R> BLK
14   R> !FILES  ;
15 DEFER LOAD

     58
 0 \ Interactive Layer          Number Input           04Apr84map
 1 CODE DIGIT   (S char base -- n true ! char false )
 2   H POP   D POP   D PUSH   E A MOV   ASCII 0 SUI   NO JM
 3   10 CPI   0< NOT IF   7 SUI   10 CPI     NO JM  THEN
 4   L CMP NO JP   A E MOV    H POP D PUSH YES JMP END-CODE
 5 : DOUBLE?   (S -- f )      DPL @ 1+   0<> ;
 6 : CONVERT   (S +d1 adr1 -- +d2 adr2 )
 7   BEGIN  1+  DUP >R  C@  BASE @  DIGIT
 8   WHILE  SWAP  BASE @ UM*  DROP  ROT  BASE @ UM*  D+
 9     DOUBLE?  IF  1 DPL +!  THEN  R>
10   REPEAT  DROP  R>  ;
11
12
13
14
15

     59
 0 \ Interactive Layer          Number Input           06Oct83map
 1 : (NUMBER?)   (S adr -- d flag )
 2   0 0  ROT  DUP 1+  C@  ASCII -  =  DUP  >R  -  -1 DPL !
 3   BEGIN   CONVERT  DUP C@  ASCII , ASCII / BETWEEN
 4   WHILE   0 DPL !
 5   REPEAT  -ROT  R> IF  DNEGATE  THEN   ROT C@ BL =  ;
 6 : NUMBER?   (S adr -- d flag )
 7   FALSE  OVER COUNT BOUNDS
 8   ?DO  I C@ BASE @ DIGIT NIP IF  DROP TRUE LEAVE THEN  LOOP
 9   IF  (NUMBER?)  ELSE  DROP  0 0 FALSE  THEN  ;
10 : (NUMBER)   (S adr -- d# )
11   NUMBER? NOT ?MISSING  ;
12 DEFER NUMBER
13
14
15
```

Listing 2. Kernel of ROM target systems (cont'd)

```
 60
 0 \ Interactive Laver          Number Output          03Apr84map
 1 : HOLD   (S char -- )   -1 HLD +!   HLD @ C!   ;
 2 : <#     (S -- )     PAD  HLD  !  ;
 3 : #>     (S d# -- addr len )    2DROP  HLD  @  PAD  OVER  -  ;
 4 : SIGN   (S n1 -- )  0< IF  ASCII -  HOLD  THEN  ;
 5 : #      (S -- )
 6   BASE @ MU/MOD ROT 9 OVER < IF  7 + THEN ASCII 0  +  HOLD  ;
 7 : #S     (S -- )     BEGIN  #  2DUP  OR  0=  UNTIL  ;
 8
 9 : HEX        (S -- )   16 BASE !  ;
10 : DECIMAL    (S -- )   10 BASE !  ;
11 : OCTAL      (S -- )    8 BASE !  ;
12
13
14
15
```

```
 61
 0 \ Interactive Laver          Number Output          02aug86cht
 1 : (U.)  (S u -- a l )   0   <# #S #>   ;
 2 : U.    (S u -- )       (U.)   TYPE SPACE   ;
 3 : U.R   (S u l -- )     >R   (U.)   R> OVER - SPACES   TYPE   ;
 4 : (.)   (S n -- a l )   DUP ABS 0   <# #S   ROT SIGN   #>   ;
 5 : .     (S n -- )       (.)   TYPE SPACE   ;
 6 : .R    (S n l -- )     >R   (.)   R> OVER - SPACES   TYPE   ;
 7 : (UD.) (S ud -- a l )  <# #S #>   ;
 8 : UD.   (S ud -- )      (UD.)   TYPE SPACE   ;
 9 : UD.R  (S ud l -- )    >R   (UD.)   R> OVER - SPACES   TYPE   ;
10 : (D.)  (S d -- a l )   TUCK DABS   <# #S   ROT SIGN   #>   ;
11 : D.    (S d -- )       (D.)   TYPE SPACE   ;
12 : D.R   (S d l -- )     >R   (D.)   R> OVER - SPACES   TYPE   ;
13 : (R.)  (S b -- a l )   0 <# # # #>   ;
14 : B.    (S b -- )       (B.)   TYPE SPACE   ;
15 : ?     (S a -- )       @ .  ;
```

```
 62
 0 \ Interactive Laver          Parsing          30Sep83map
 1 LABEL $DONE   B POP   H PUSH   D PUSH   NEXT   END-CODE
 2 CODE  SKIP   (S addr len char -- addr' len' )
 3    IP>HL   B POP D POP XTHL
 4    ( C=char DE=length  HL=addr )
 5    BEGIN   D A MOV  E ORA  0<>
 6    WHILE   M A MOV  C CMP   $DONE JNZ    H INX  D DCX
 7    REPEAT   $DONE JMP  END-CODE
 8 CODE  SCAN   (S addr len char -- addr' len' )
 9    IP>HL   B POP  D POP  XTHL
10    ( C=char DE=length HL=addr )
11    BEGIN   D A MOV  E ORA  0<>
12    WHILE   M A MOV   C CMP   $DONE JZ    H INX  D DCX
13    REPEAT   $DONE JMP  END-CODE
14
15
```

```
 63
 0 \ Interactive Laver          Parsing          02Apr84map
 1 : /STRING   (S addr len n -- addr' len' )
 2    OVER MIN   ROT OVER +   -ROT -   ;
 3 : PLACE     (S str-addr len to -- )
 4    3DUP  1+ SWAP MOVE  C! DROP  ;
 5 : (SOURCE)    (S -- addr len )
 6    BLK @ ?DUP IF   BLOCK B/BUF   ELSE   TIB #TIB @   THEN  ;
 7 DEFER SOURCE
 8 : PARSE-WORD   (S char -- addr len )
 9    >R  SOURCE TUCK  >IN @ /STRING  R@ SKIP  OVER SWAP R> SCAN
10    >R OVER -  ROT R>  DUP 0<> + - >IN !  ;
11 : PARSE   (S char -- addr len )
12    >R   SOURCE >IN @ /STRING   OVER SWAP R> SCAN
13    >R OVER -  DUP R>  0<> -  >IN +!  ;
14
15
```

```
 64
 0 \ Interactive Laver          Parsing          23Jun85cht
 1 : 'WORD   (S -- adr )
 2    HERE  ;
 3 : WORD    (S char -- addr )
 4    PARSE-WORD  'WORD PLACE
 5    'WORD DUP COUNT + BL SWAP C!   ( Stick Blank at end )   ;
 6 : >TYPE   (S adr len -- )
 7    TUCK PAD SWAP CMOVE   PAD SWAP TYPE  ;
 8 : .(   (S -- )   ASCII ) PARSE >TYPE  ; IMMEDIATE
 9 : (    (S -- )   ASCII ) PARSE 2DROP  ; IMMEDIATE
10 EXIT
11 : \S   (S -- )   END? ON ;  IMMEDIATE
12
13
14
15
```

```
 65
 0 \ Interactive Laver          Dictionary          03aug86cht
 1 CODE TRAVERSE (S addr direction -- addr' )
 2    D POP H POP   127 A MVI
 3    BEGIN   D DAD  M CMP  0< UNTIL   HPUSH JMP   END-CODE
 4 : DONE?   (S n -- f )
 5    STATE @ <>   END? @ OR   END? OFF   ;
 6
 7
 8
 9
10
11
12
13
14
15
```

Listing 2. Kernel of ROM target systems (cont'd)

```
 66
 0 \ Interactive Layer                Dictionary
 1 : N>LINK     2-   ;
 2 : L>NAME     2+   ;
 3 : BODY>      2-   ;
 4 : NAME>      1 TRAVERSE   1+   ;
 5 : LINK>      L>NAME   NAME>   ;
 6 : >BODY      2+   ;
 7 : >NAME      1- -1 TRAVERSE   ;
 8 : >LINK      >NAME   N>LINK   ;
 9
10
11
12
13
14
15
```

```
 67
 0 \ Interactive Layer                Dictionary
 1 CODE HASH   (S str-addr voc-ptr -- thread )
 2    D POP H POP    H INX M A MOV 3 ANI
 3    A L MOV 0 H MVI H DAD D DAD HPUSH JMP   END-CODE
 4 CODE (FIND)     (S here nfa -- here false ! cfa flag )
 5  H POP H A MOV L ORA   NO JZ
 6  BEGIN   D POP  D PUSH  H PUSH  H INX  H INX
 7     D LDAX  M XRA  63 ANI  0= IF
 8        BEGIN D INX H INX D LDAX M XRA A ADD 0= IF 2SWAP CS UNTIL
 9        H INX D POP XTHL XCHG
10        H INX H INX  M A MOV  64 ANI YES JZ  1 H LXI HPUSH JMP
11     THEN THEN
12     H POP  M E MOV  H INX  M D MOV
13     XCHG   H A MOV   L ORA
14  0= UNTIL NO JMP END-CODE
15
```

```
 68
 0 \ Interactive Layer                Dictionary
 1 4 CONSTANT #THREADS
 2 : FIND   (S addr -- cfa flag ! addr false )
 3    DUP C@ IF   PRIOR OFF   FALSE   #VOCS 0
 4       DO   DROP CONTEXT I 2* + @ DUP
 5         IF   DUP PRIOR @ OVER PRIOR !   =
 6           IF   DROP FALSE
 7           ELSE   OVER SWAP HASH @ (FIND)  DUP ?LEAVE
 8         THEN THEN   LOOP
 9    ELSE  DROP END? ON  ['] NOOP 1  THEN  ;
10 \ : ?UPPERCASE   (S adr -- adr )
11 \    CAPS @ IF  DUP COUNT UPPER   THEN  ;
12 : DEFINED   (S -- here 0 ! cfa [ -1 ! 1 ] )
13    BL WORD  ( ?UPPERCASE )  FIND   ;
14
15
```

```
 69
03aug86cht \ Interactive Layer          Interpreter              27Sep85cht
          : ?STACK  (S -- )   ( System dependant )
             SP@ SP0 @ SWAP U<   ABORT" Stack Underflow"
             SP@ PAD U<   ABORT" Stack Overflow"   ;
          DEFER STATUS  (S -- )
          : INTERPRET   (S -- )
             BEGIN   ?STACK  DEFINED
               IF     EXECUTE
               ELSE   NUMBER  DOUBLE? NOT IF  DROP  THEN
               THEN   FALSE DONE?
             UNTIL   ;
```

```
 70
27AU683HHL \ Extensible Layer          Compiler                 02aug86cht
          : ALLOT  (S n -- )      DP +!   ;
          : ,      (S n -- )   HERE !   2 ALLOT   ;
          : C,     (S char -- )   HERE C!   1 ALLOT ;
          : ALIGN  ( HERE 1 AND IF  BL C,  THEN )  ; IMMEDIATE
          : EVEN   ( DUP 1 AND + ) ;  IMMEDIATE
          : COMPILE   (S -- )   R> DUP 2+ >R   @ ,   ;
          : IMMEDIATE (S -- )   64 ( Precedence bit ) LAST @  CSET  ;
          : LITERAL   (S n -- )    COMPILE (LIT)   ,   ;   IMMEDIATE
          : DLITERAL    (S d# -- )
               SWAP   [COMPILE] LITERAL  [COMPILE] LITERAL  ; IMMEDIATE
          : ASCII     (S -- n )   BL WORD   1+ C@
             STATE @ IF   [COMPILE] LITERAL   THEN   ; IMMEDIATE
          : CONTROL   (S -- n )   BL WORD   1+ C@  31 AND
             STATE @ IF   [COMPILE] LITERAL   THEN   ; IMMEDIATE
```

```
 71
23Jun85cht \ Extensible Layer          Compiler                 225Jun85cht
          : CRASH   (S -- )
             TRUE ABORT" Uninitialized execution vector."  ;
          : ?MISSING   (S f -- )
             IF   'WORD COUNT TYPE   TRUE ABORT"  ?"   THEN   ;
          : '   (S -- cfa )   DEFINED 0= ?MISSING   ;
          : ['] (S -- )        ' [COMPILE] LITERAL   ; IMMEDIATE
          : [COMPILE]   (S -- )   ' ,   ; IMMEDIATE
          : (")    (S -- addr len )   R> COUNT 2DUP + EVEN >R  ;
          : (.")   (S -- )            R> COUNT 2DUP + EVEN >R   TYPE   ;
          : ."   (S -- )
             ASCII " PARSE  TUCK 'WORD PLACE  1+ ALLOT ALIGN  ;
          : ."   (S -- )   COMPILE (.")   ,"   ;   IMMEDIATE
          : "    (S -- )   COMPILE (")    ,"   ;   IMMEDIATE
```

Listing 2. Kernel of ROM target systems (cont'd)

```
 72
 0 \ Interactive Layer          Dictionary          03aug86cht
 1 \ VARIABLE FENCE
 2 : TRIM  (S faddr voc-addr -- )
 3   #THREADS 0 DO  2DUP @ BEGIN  2DUP U> NOT WHILE  @ REPEAT
 4      NIP OVER !  2+  LOOP  2DROP  ;
 5 : (FORGET)  (S addr -- )
 6   DUP FENCE @ U< ABORT" Below fence"
 7   DUP VOC-LINK @ BEGIN  2DUP U< WHILE  @ REPEAT
 8   DUP VOC-LINK !  NIP
 9   BEGIN  DUP WHILE  2DUP #THREADS 2* - TRIM  @  REPEAT
10   DROP  DP !  ;
11 : FORGET  (S -- )
12   BL WORD DUP CURRENT @ HASH @ (FIND) 0= ?MISSING
13   >LINK (FORGET)  ;
14
15
```

```
 73
 0 \ Extensible Layer          Compiler          25Jun85cht
 1 DEFER WHERE
 2 DEFER ?ERROR
 3 : (?ERROR)  (S adr len f -- )
 4   IF  >R >R  SP0 @ SP!  PRINTING OFF
 5      BLK @ IF  >IN @ BLK @ WHERE  THEN
 6      R> R> SPACE TYPE SPACE  QUIT
 7   ELSE  2DROP  THEN  ;
 8 : (ABORT")  (S f -- )
 9   R@ COUNT ROT ?ERROR  R> COUNT + EVEN >R  ;
10 : ABORT"  (S -- )
11    COMPILE (ABORT")  ." :  IMMEDIATE
12 : ABORT  (S -- )
13   TRUE ABORT" "  ;
14
15
```

```
 74
 0 \ Extensible Layer          Structures          03Apr84map
 1 : ?CONDITION  (S f -- )
 2   NOT ABORT" Conditionals Wrong"  ;
 3 : >MARK      (S -- addr )   HERE 0 ,  ;
 4 : >RESOLVE   (S addr -- )   HERE SWAP !  ;
 5 : <MARK      (S -- addr )   HERE  ;
 6 : <RESOLVE   (S addr -- )   ,  ;
 7
 8 : ?>MARK     (S -- f addr )  TRUE >MARK  ;
 9 : ?>RESOLVE  (S f addr -- )  SWAP ?CONDITION >RESOLVE  ;
10 : ?<MARK     (S -- f addr )  TRUE  <MARK  ;
11 : ?<RESOLVE  (S f addr -- )  SWAP ?CONDITION <RESOLVE  ;
12
13 : LEAVE  COMPILE (LEAVE)  ; IMMEDIATE
14 : ?LEAVE  COMPILE (?LEAVE)  ; IMMEDIATE
15
```

```
 75
 0 \ Extensible Layer          Structures          01Oct83aac
 1 : BEGIN  ?<MARK                                  ; IMMEDIATE
 2 : THEN  ?>RESOLVE                                ; IMMEDIATE
 3 : DO  COMPILE (DO)  ?>MARK                       ; IMMEDIATE
 4 : ?DO  COMPILE (?DO)  ?>MARK                     ; IMMEDIATE
 5 : LOOP
 6    COMPILE (LOOP)  2DUP 2+ ?<RESOLVE ?>RESOLVE   ; IMMEDIATE
 7 : +LOOP
 8    COMPILE (+LOOP) 2DUP 2+ ?<RESOLVE ?>RESOLVE   ; IMMEDIATE
 9 : UNTIL  COMPILE ?BRANCH  ?<RESOLVE              ; IMMEDIATE
10 : AGAIN  COMPILE  BRANCH  ?<RESOLVE              ; IMMEDIATE
11 : REPEAT  2SWAP [COMPILE] AGAIN  [COMPILE] THEN  ; IMMEDIATE
12 : IF  COMPILE  ?BRANCH  ?>MARK                   ; IMMEDIATE
13 : ELSE  COMPILE  BRANCH ?>MARK  2SWAP ?>RESOLVE  ; IMMEDIATE
14 : WHILE  [COMPILE] IF                            ; IMMEDIATE
15
```

```
 76
 0 \ Extensible Layer          Defining Words          02aug86cht
 1 \ : ,VIEW  (S -- )  BLK @ ,  ;
 2 : "CREATE  (S str -- )  COUNT HERE EVEN 2+ PLACE
 3    ALIGN ( ,VIEW )  HERE 0 , ( reserve link )
 4    HERE LAST ! ( remember nfa )  HERE  ( lfa nfa )  WARNING @
 5    IF  FIND
 6      IF  HERE COUNT TYPE ." isn't unique " THEN  DROP HERE
 7    THEN  ( lfa nfa )  CURRENT @ HASH DUP @ ( lfa tha prev )
 8    HERE 2- ROT !  ( lfa prev )  SWAP !  ( Resolve link field)
 9    HERE  DUP  C@  WIDTH @  MIN  1+  ALLOT  ALIGN
10    128 SWAP CSET  128 HERE 1- CSET  ( delimiter Bits )
11    COMPILE [ [FORTH] ASSEMBLER DOCREATE , META ]  ;
12 : CREATE  (S -- )
13    BL WORD  ( ?UPPERCASE )  "CREATE  ;
14
15
```

```
 77
 0 \ Extensible Layer          Defining Words          02aug86cht
 1 : !CSP  (S -- )  SP@ CSP !  ;
 2 : ?CSP  (S -- )  SP@ CSP @ <> ABORT" Stack Changed"  ;
 3 : HIDE  (S -- )
 4    LAST @  DUP N>LINK @  SWAP CURRENT @ HASH !  ;
 5 : REVEAL  (S -- )
 6    LAST @ DUP N>LINK  SWAP CURRENT @ HASH !  ;
 7 : (;USES)  (S -- )  R> @  LAST @ NAME> !  ;
 8 \ VOCABULARY ASSEMBLER
 9 : ;USES  (S -- )  ?CSP  COMPILE  (;USES)
10    [COMPILE] [  REVEAL  ASSEMBLER  ; IMMEDIATE
11 : (;CODE)  (S -- )  R>  LAST @ NAME> !  ;
12 : ;CODE  (S -- )  ?CSP  COMPILE  (;CODE)
13    [COMPILE] [  REVEAL  ASSEMBLER  ; IMMEDIATE
14 : DOES>  (S -- )  COMPILE (;CODE)  205 ( CALL ) C,
15   [  [ASSEMBLER] DODOES META ] LITERAL , ; IMMEDIATE
```

Listing 2. Kernel of ROM target systems (cont'd)

```
     78
 0 \ Extensible Layer            Defining Words
 1 : [   (S -- )   STATE OFF   ;   IMMEDIATE
 2 : ]   (S -- )
 3    STATE ON   BEGIN   ?STACK   DEFINED DUP
 4    IF      0> IF    EXECUTE   ELSE   ,   THEN
 5    ELSE   DROP   NUMBER  DOUBLE?
 6       IF          [COMPILE] DLITERAL
 7       ELSE DROP   [COMPILE] LITERAL   THEN
 8    THEN   TRUE DONE? UNTIL   ;
 9 : :   (S -- )
10    !CSP   CURRENT @ CONTEXT !   CREATE HIDE    ]
11    ;USES   NEST ,
12 : ;   (S -- )
13    ?CSP   COMPILE UNNEST   REVEAL   [COMPILE] [
14    ;   IMMEDIATE
15
```

```
     79
 0 \ Extensible Layer            Defining Words
 1 : RECURSIVE (S -- )   REVEAL ;   IMMEDIATE
 2 : CONSTANT   (S n -- )
 3    CREATE ,   ;USES DOCONSTANT ,
 4 : VARIABLE  (S -- )
 5    CREATE 0 ,   ;USES DOCREATE ,
 6 : DEFER   (S -- )
 7    CREATE   ['] CRASH ,  ;USES   DODEFER ,
 8 \    DODEFER RESOLVES <DEFER>
 9 : VOCABULARY   (S -- )
10    CREATE   #THREADS 0 DO   0 ,  LOOP
11       HERE  VOC-LINK @ ,  VOC-LINK !
12    DOES>   CONTEXT !  ;                DROP
13 : DEFINITIONS   (S -- )
14    CONTEXT @ CURRENT !   ;
15
```

```
     80
 0 \ Extensible Layer            Defining Words
 1 : 2CONSTANT
 2    CREATE   , ,     (S d# -- )
 3    DOES>   2@   ;   (S -- d# )   DROP
 4 : 2VARIABLE
 5    0 0 2CONSTANT   (S -- )
 6    DOES>        ;  (S -- addr )   DROP
 7
 8 \ VARIABLE AVOC
 9 : CODE   (S -- )      CREATE  HIDE   HERE DUP 2- !
10    CONTEXT @ AVOC !   ASSEMBLER   ;
11 ASSEMBLER DEFINITIONS
12 : END-CODE   AVOC @ CONTEXT !   REVEAL   ;
13 FORTH DEFINITIONS   META IN-META
14
15
```

```
     81
27Sep83map \S Extensible Layer            Defining Words        06aug86cht
           \ VARIABLE #USER
           \ VOCABULARY USER   USER DEFINITIONS
           USER DEFINITIONS
           : ALLOT   (S n -- )
              #USER +!   ;
           ' CREATE  ( avoid recursion: leave address for , in CREATE )
           : CREATE  (S -- )
              [ , ]     #USER @ ,   ;USES  DOUSER-VARIABLE ,
           : VARIABLE     (S -- )
              CREATE   2 ALLOT   ;
           : DEFER   (S -- )
              VARIABLE   ;USES   DOUSER-DEFER  ,
           FORTH DEFINITIONS   META IN-META
```

```
     82
02aug86cht \ Extensible Layer            ReDefining Words        23Jun85cht
           : >IS   (S cfa -- data-address )
              DUP @
              DUP [  [ASSEMBLER] DOUSER-VARIABLE META ] LITERAL = SWAP
              DUP [  [ASSEMBLER] DOUSER-DEFER    META ] LITERAL = SWAP
              DROP   OR IF   >BODY @ UP @ +   ELSE    >BODY   THEN   ;
           : (IS)      (S cfa --- )
              R@ @  >IS !   R> 2+ >R   ;
           : IS   (S cfa --- )
              STATE @ IF  COMPILE (IS)  ELSE  ' >IS !  THEN ; IMMEDIATE
```

```
     83
14Jun85cht \ Initialization             High Level              21FEB85CHT
           : [   (S -- )   STATE OFF   ;   IMMEDIATE
           : RUN   (S -- )
              INTERPRET ;
           : QUIT   (S -- )
              SP0 @ 'TIB !    BLK OFF    [COMPILE] [
              BEGIN RP0 @ RP! STATUS QUERY  INTERPRET
                 STATE @ NOT IF   ." ok"   THEN   AGAIN  ;
           DEFER BOOT
           : WARM   (S -- )
              TRUE ABORT" Warm Start"   ;
           : COLD   (S -- )   10 DO LOOP
              INIT-RAM 16384 256 CMOVE   INIT CR ." OCU V.4.3"
              CR ." 5-10-1987"
              CR   BOOT QUIT   ;
```

Listing 2. Kernel of ROM target systems (cont'd)

```
 84
 0 \ Initialization              High Level
 1 \ 1 CONSTANT INITIAL
 2 \ : OK   (S -- )   INITIAL LOAD   ;
 3 : BYE   ( -- )
 4 \    CR   HERE 0 256 UM/MOD NIP 1+   DECIMAL U.   ." Pages"
 5   0 0 BDOS  ;
 6
 7
 8
 9
10
11
12
13
14
15
```

```
 85
 0 \ Initialization              Low Level
 1 HEX [FORTH] ASSEMBLER
 2 HERE ORIGIN 6 + !-T  ( WARM ENTRY POINT )
 3   ' WARM H LXI   >NEXT1 JMP
 4 HERE ORIGIN 2 + !-T  ( COLD ENTRY POINT )
 5   4000 H LXI   UP SHLD
 6   INIT-R0 H LXI   RP SHLD
 7   INIT-R0 100 - H LXI   SPHL
 8   ' COLD H LXI   >NEXT1 JMP
 9 DECIMAL
10
11
12
13
14
15
```

```
 86
 0 \ System variable boot up table
 1 CREATE INIT-RAM
 2 HEX   0 , 0 , 0 ,   INIT-R0 100 - ,  INIT-R0 ,
 3 4100 , 0 , 0 , 0 ,   0A , 0 , 0 , 0 , 0 ,
 4 ' (EMIT) , ' (KEY) , ' (KEY?) , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
 5 0 , 0 , 0 , 0 , 0 , ( FORTH VOC)
 6 INIT-R0 , 4000 , 0 , 0 , 0 , 0 , 0 , 0 ,
 7 0 ( LAST) , 0 , 4030 ( FORTH) , 4030 ( FORTH) ,
 8 4092 ( USER) , 4030 , 0 , 0 , 0 , 0 , 0 , 0 ,   ( context)
 9 INIT-R0 100 - , 1F , 409A ( VOC-LINK) , 0 , 0 , 0 , 0 ,
10 0 , 0 , 1238 ( CC) , 4100 ( FENCE) , 0 , 1E , 0 , ( FOUND)
11 0 , 0 , 0 , 0 , 4038 , ( EDITOR)
12 0 , 0 , 0 , 0 , 4086 , ( ASSEMBLER)
13 0 , 0 , 0 , 0 , 4090 , ( USER)
14 C300 , 80 , ( RSTA) C300 , C0 , ( RSTB) C900 , 0 , ( RSTC)
15 C300 , 40 , ( INT) C900 , 0 , ( NMI) DECIMAL
```

```
 87
23Jun85cht \ Resident Tools                                    27Jun85cht
: DEPTH      (S -- n )   SP@ SP0 @ SWAP - 2/   ;
: .S         (S -- )
   DEPTH ?DUP
   IF  0 DO  DEPTH I - 1- PICK  7 U.R SPACE  KEY? ?LEAVE  LOOP
   ELSE  ." Empty "  THEN  ;
: .ID     (S nfa -- )
   DUP 1+ DUP C@ ROT C@ 31 AND 0
   ?DO DUP 127 AND EMIT   128 AND
     IF   ASCII _ 128 OR   ELSE  1+ DUP C@  THEN
   LOOP 2DROP SPACE ;
: DUMP    (S addr len -- )
   0 DO   CR ( DUP 6 .R SPACE )  16 0 DO   DUP C@ B. 1+   LOOP
   16 +LOOP   DROP   ;
: SUM (S addr len -- )
   0 SWAP   0 DO   OVER I + C@ +   LOOP   NIP   ;
```

```
 88
25dec86cht \ For Completeness                                  03aug86cht
: RECURSE   (S -- )
   LAST @ NAME> ,  ;  IMMEDIATE
: START
   SP0 @ SP!      RUN   QUIT ;
HEX
LABEL DOVOC
   D INX   XCHG   M E MOV   H INX   M D MOV
   XCHG CONTEXT SHLD   NEXT
DECIMAL
```

```
 89
24JAN87CHT \ Resolve Forward References                        02aug86cht

' (.") RESOLVES <(.")>   ' (") RESOLVES <(")>
' (;CODE) RESOLVES <(;CODE)>
' (;USES) RESOLVES <(;USES)>   ' (IS) RESOLVES <(IS)>
' (ABORT") RESOLVES <(ABORT")>
[ASSEMBLER] DODEFER META RESOLVES <DEFER>
[ASSEMBLER] DOCREATE META RESOLVES <CREATE>
[ASSEMBLER] DOCONSTANT META RESOLVES <VARIABLE>
[ASSEMBLER] DOUSER-DEFER META RESOLVES <USER-DEFER>
[ASSEMBLER] DOUSER-VARIABLE META RESOLVES <USER-VARIABLE>
[ASSEMBLER] DOVOC META RESOLVES <VOCABULARY>
```

Listing 2. Kernel of ROM target systems (cont'd)

```
 90                                                          0
 0 \ Resolve Forward References                 03aug86cht \ Kernel of ROM target system
 1 ' R> RESOLVES R>              ' DUP RESOLVES DUP
 2 ' @ RESOLVES @                ' >R RESOLVES >R
 3 ' -ROT RESOLVES -ROT          ' SWAP RESOLVES SWAP
 4 ' - RESOLVES -                ' = RESOLVES =
 5 ' 2DROP RESOLVES 2DROP
 6 ' + RESOLVES +                ' OVER RESOLVES OVER
 7 ' DEFINITIONS RESOLVES DEFINITIONS
 8 ' [ RESOLVES [                ' 2+ RESOLVES 2+
 9 ' 1+ RESOLVES 1+              ' 2* RESOLVES 2*
10 ' 2DUP RESOLVES 2DUP          ' ?MISSING RESOLVES ?MISSING
11 ' QUIT RESOLVES QUIT          ' RUN RESOLVES RUN
12 ' INIT-RAM RESOLVES INIT-RAM
13 ' (INIT) RESOLVES INIT
14
15
```

This file contains the Kernal source code for a ROM based F83 target system. Most screens are identical to the original F83/8080 screens. Only those which were modified are commented in the corresponding shadow screens.

Consult Mike Perry and Henry Laxen's F83 source code or 'Inside F83'.

```
 91                                                          0
 0 \ Initialize DEFER words                     29APR87CHT \ Kernel of ROM target system
 1    ' (LOAD) IS LOAD
 2    ' CRLF IS CR
 3    ' NOOP IS WHERE             ' CR IS STATUS
 4    ' (SOURCE) IS SOURCE
 5    ' NORMAL IS BOOT
 6    ' (NUMBER) IS NUMBER
 7    ' (CHAR) IS CHAR            ' (DEL-IN) IS DEL-IN
 8    ' (?ERROR) IS ?ERROR
 9
10
11
12
13
14
15
```

This file contains the Kernal source code for a ROM based F83 target system. Most screens are identical to the original F83/8080 screens. Only those which were modified are commented in the corresponding shadow screens.

Consult Mike Perry and Henry Laxen's F83 source code or 'Inside F83'.

```
 92                                                          0
 0 \ Initialize vocabulary                      10jan87cht \ Kernel of ROM target system
 1 HEX
 2 ' FORTH 4 + THERE   ' INIT-RAM 2+ 30 + THERE  8 CMOVE
 3 ' EDITOR 4 + THERE  ' INIT-RAM 2+ 7E + THERE  8 CMOVE
 4 ' ASSEMBLER 4 + THERE  ' INIT-RAM 2+ 88 + THERE  8 CMOVE
 5 ' USER 4 + THERE    ' INIT-RAM 2+ 92 + THERE  8 CMOVE
 6 LAST-T @    ' INIT-RAM 2+ 4A + THERE !
 7 ' CC-FORTH 2+   ' INIT-RAM 2+ 74 + THERE !
 8 DECIMAL
 9
10
11
12
13
14
15
```

This file contains the Kernal source code for a ROM based F83 target system. Most screens are identical to the original F83/8080 screens. Only those which were modified are commented in the corresponding shadow screens.

Consult Mike Perry and Henry Laxen's F83 source code or 'Inside F83'.

Listing 2. Kernel of ROM target systems (cont'd)

```
      0
 0 Utility                                              06aug86cht
 1 This file includes many important utilities needed for doing
 2 serious Forth programming.  Including these functions in a
 3 ROM-based Forth kernel would allow the system to be a
 4 complete and useful software development system.
 5
 6 Dr. C. H. Ting
 7 Offete Enterprises, Inc.
 8 1306 S. B Street
 9 San Mateo, CA  94402
10
11
12
13
14
15
```

```
      1
 0 \ Utiltiy Loading
 1 16 19 THRU ( MISCELLANEOUS )
 2 8 13 THRU ( EDITOR )
 3 2 LOAD 4 LOAD 3 LOAD 5 LOAD ( ASSEMBLER )
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
      2
 0 \ 8080 Assembler Defining Words & Registers
 1 : LABEL   CREATE   ASSEMBLER   ;
 2 205 CONSTANT DOES-OP
 3 ASSEMBLER DEFINITIONS
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
      3
06aug86cht \ 8080 Assembler Defining Words & Registers          05aug86cht
    >NEXT 1-  CONSTANT HPUSH     >NEXT 2-  CONSTANT DPUSH
    7 CONSTANT A      DPUSH CONSTANT WPUSH
    0 CONSTANT B   1 CONSTANT C      2 CONSTANT D   3 CONSTANT E
    0 CONSTANT I   1 CONSTANT I'     2 CONSTANT W   3 CONSTANT W'
    0 CONSTANT IP  1 CONSTANT IP'    4 CONSTANT H   5 CONSTANT L
    6 CONSTANT M   6 CONSTANT PSW    6 CONSTANT SP  6 CONSTANT S
    : 1MI CREATE C, DOES> C@ C, ;   RESOLVES <1MI>
    : 2MI CREATE C, DOES> C@ + C, ;   RESOLVES <2MI>
    : 3MI CREATE C, DOES> C@ SWAP 8* + C, ;   RESOLVES <3MI>
    : 4MI CREATE C, DOES> C@ C, C, ;   RESOLVES <4MI>
    : 5MI CREATE C, DOES> C@ C, , ;   RESOLVES <5MI>
```

```
      4
10feb87cht \ 8080 Assembler mnemonics                    09MAR83HHL
    HEX
    00 1MI NOP   76 1MI HLT  F3 1MI DI   FB 1MI EI    07 1MI RLC
    0F 1MI RRC   17 1MI RAL  1F 1MI RAR  E9 1MI PCHL  EB 1MI XCHG
    C9 1MI RET   C0 1MI RNZ  C8 1MI RZ   D0 1MI RNC   D8 1MI RC
    2F 1MI CMA   37 1MI STC  3F 1MI CMC  F9 1MI SPHL  E3 1MI XTHL
    E0 1MI RPO   E8 1MI RPE  F8 1MI RM   27 1MI DAA
    80 2MI ADD   88 2MI ADC  90 2MI SUB  98 2MI SBB   A0 2MI ANA
    A8 2MI XRA   B0 2MI ORA  B8 2MI CMP  02 3MI STAX  04 3MI INR
    03 3MI INX   09 3MI DAD  0B 3MI DCX  C1 3MI POP   C5 3MI PUSH
    C7 3MI RST   05 3MI DCR  0A 3MI LDAX D3 4MI OUT   DB 4MI IN
    C6 4MI ADI   CE 4MI ACI  D6 4MI SUI  DE 4MI SBI   E6 4MI ANI
    EE 4MI XRI   F6 4MI ORI  FE 4MI CPI  22 5MI SHLD  CD 5MI CALL
    2A 5MI LHLD  32 5MI STA  3A 5MI LDA  C3 5MI JMP
    C2 5MI JNZ   CA 5MI JZ   D2 5MI JNC  DA 5MI JC    E2 5MI JPO
    EA 5MI JPE   F2 5MI JP   FA 5MI JM
```

```
      5
060ct83aap \ 8080 Assembler Branches                     21FEB85CHT
    D2 CONSTANT C0=  DA CONSTANT C0<>  D2 CONSTANT CS
    C2 CONSTANT 0=   CA CONSTANT 0<>   E2 CONSTANT PE
    F2 CONSTANT 0<   FA CONSTANT 0>=  : NOT   8 [ FORTH ] XOR   ;
    : NEXT   >NEXT JMP   ;
    : MOV   8* 40 + + C, ;
    : MVI   8* 6 + C, C, ;     : LXI   8* 1+ C, , ;
    : IF     C,   ?>MARK   ;
    : THEN        ?>RESOLVE   ;
    : ELSE   C3 C, ?>MARK   2SWAP   ?>RESOLVE   ;
    : BEGIN        ?<MARK   ;
    : UNTIL      C, ?<RESOLVE   ;
    : AGAIN   C3 ( JMP ) C, ?<RESOLVE   ;
    : WHILE   C, ?>MARK   ;
    : REPEAT    2SWAP C3 C, ?<RESOLVE   ?>RESOLVE   ;
    FORTH DEFINITIONS   DECIMAL   ( source destination MOV )
```

Listing 3. Utility

```
 8
 0 \ String operators
 1 EDITOR DEFINITIONS
 2 : INSERT   (S string length buffer size -- )
 3    ROT OVER MIN >R  R@ - ( left over )
 4    OVER DUP R@ +  ROT CMOVE>   R> CMOVE  ;
 5 : REPLACE   (S string length buffer size -- )  ROT MIN CMOVE ;
 6 : DELETE   (S buffer size count -- )
 7    OVER MIN >R  R@ - ( left over )  DUP 0>
 8    IF  2DUP SWAP DUP R@ + -ROT SWAP CMOVE  THEN  + R> BLANK ;
 9 \ VARIABLE FOUND
10 : SEARCH   ( sadr slen badr blen -- n f )
11    FOUND OFF  OVER >R  ROT TUCK 2DUP U<
12    IF  2DROP  ELSE  - 1+ 0
13      ?DO   3DUP COMPARE 0= IF  FOUND ON  LEAVE  THEN
14         SWAP 1+ SWAP   LOOP
15    THEN  DROP NIP R> -  FOUND @  ;
```

```
 9
 0 \ Move the Editor's cursor around
 1 : TOP          (S -- )      R# OFF ;
 2 : C            (S n -- )    R# @ + B/BUF MOD R# ! ;
 3 : CURSOR       (S -- n )    R# @ ;
 4 : LINE#        (S -- n )    CURSOR  C/L  /  ;
 5 : COL#         (S -- n )    CURSOR  C/L  MOD  ;
 6 : 'START       (S -- adr )  SCR @ BLOCK ;
 7 : 'CURSOR      (S -- adr )  'START  CURSOR  + ;
 8 : 'LINE        (S -- adr )  'CURSOR  COL# -  ;
 9 : #AFTER       (S -- n )    C/L COL# -  ;
10 : #REMAINING   (S -- n )    B/BUF CURSOR - ;
11 : #END         (S -- n )    #REMAINING COL# +  ;
12 : T            (S n -- )    TOP  C/L *  C
13    CR 2 SPACES   'CURSOR C/L TYPE ;
14 : +T           (S n -- )    LINE# + T   ;
15
```

```
 10
 0 \ buffers
 1 ASCII ^ CONSTANT EOS
 2 : ?TEXT   (S adr -- adr+1 n )   >R   EOS WORD  C@
 3    IF  R@ C/L 1+ BLANK  HERE COUNT R@ PLACE  THEN  R> COUNT ;
 4 84 CONSTANT C/PAD
 5 : 'INSERT   (S -- insert-buffer )   PAD     C/PAD + ;
 6 : 'FIND     (S -- find-buffer )     'INSERT C/PAD + ;
 7 : ?MISS (S n f -- n ; )
 8    0= IF  DROP 'FIND COUNT TYPE
 9    ." not found " QUIT THEN ;
10
11
12
13
14
15
```

```
 11
02aug86cht \ buffers                                          16Oct83ac
          : KEEP   (S -- )   'LINE C/L 'INSERT  PLACE  ;
          : K   (S -- )   'FIND PAD  C/PAD CMOVE
             'INSERT 'FIND  C/PAD CMOVE   PAD 'INSERT  C/PAD CMOVE  ;
          : 'C#A   (S -- 'cursor #after )   'CURSOR #AFTER  ;
          : (I)  (S -- len 'insert len 'cursor #after )
             'INSERT ?TEXT  TUCK 'C#A  ;
          : (TILL)  (S -- n )   'FIND ?TEXT 'C#A SEARCH ?MISS ;
          : 'F+   (S n1 -- n2 )  'FIND C@ +  ;
```

```
 12
21FEB85CHT \ line editing                                     03aug86cht
          : I   (S -- )   (I)  INSERT  C ;
          : P   (S -- )   'INSERT ?TEXT DROP 'LINE C/L CMOVE  ;
          : U   (S -- )   C/L C 'LINE C/L OVER #END INSERT  P ;
          : X   (S -- )   KEEP  'LINE #END C/L  DELETE  ;
          : G   (S  screen line -- )
            R# @ >R  SCR @ >R   T  SCR !  KEEP
            R> SCR !  R> R# !  C/L NEGATE C  U  C/L C ;
          FORTH DEFINITIONS
          : L   SCR @ LIST ;
          : N   1 SCR +!   L ;
          : B   -1 SCR +!   L ;
          EDITOR DEFINITIONS
```

```
 13
03aug86cht \ find and replace                                 03aug86cht
          : FIND?  (S - n f ) 'FIND ?TEXT  'CURSOR #REMAINING  SEARCH ;
          : F   (S -- )   FIND? ?MISS   'F+ C ;
          : S   (S n - )   FIND?
            IF  'F+ C  EXIT  THEN  DROP  FALSE OVER SCR @
            DO   N TOP  'FIND COUNT 'CURSOR #REMAINING SEARCH
              IF  'F+ C DROP TRUE LEAVE  ELSE  DROP  THEN
             KEY? ABORT" Break!"
            LOOP  ?MISS  ;
          : E   (S -- )   'FIND C@  DUP NEGATE C  'C#A ROT DELETE ;
          : D   (S -- )   F E ;
          : R   (S -- )   E I ;
          : TILL    (S -- )   'C#A (TILL)  'F+  DELETE ;
          FORTH DEFINITIONS
```

Listing 3. Utility (cont'd)

```
    16
 0 ( Commenting and Loading Words                    16Oct83map
 1 )
 2 64 CONSTANT C/L
 3 : \   ( -- )     >IN @ NEGATE  C/L MOD  >IN +!  ; IMMEDIATE
 4 : THRU   (S n1 n2 -- )
 5   1+ SWAP ?DO   I DUP . LOAD   LOOP   ;
 6
 7
 8
 9
10
11
12
13
14
15
```

```
    17
 0 \ Managing Source Screens                         24JAN87CHT
 1 : LIST   (S n -- )
 2   CR  DUP SCR !   ." Scr # " DUP .   16 0
 3   DO   CR  I 3 .R SPACE
 4     DUP BLOCK  I C/L * + C/L -TRAILING TYPE   KEY? ?LEAVE
 5   LOOP  DROP CR   EDITOR ;
 6 : TRIAD   (S n -- )
 7   12 EMIT ( form feed ) 3 / 3 * 3 BOUNDS DO  I LIST  LOOP  ;
 8 : INDEX   (S n1 n2 -- )
 9   1+ SWAP
10   DO   CR   I 3 .R SPACE   I BLOCK  C/L -TRAILING TYPE
11   KEY? ?LEAVE   LOOP  CR  ;
12
13
14
15
```

```
    18
 0 \ Display the WORDS in the Context Vocabulary     02aug86cht
 1 : ?LINE   (S n -- )
 2   #OUT @ +  70 > IF  CR  THEN   ;
 3 : LARGEST (S addr n -- addr' val )
 4   OVER 0 SWAP ROT 0
 5   DO   2DUP @ U< IF   -ROT 2DROP    DUP @ OVER   THEN  2+
 6   LOOP   DROP   ;
 7 : WORDS   (S -- )
 8   CR   CONTEXT @ HERE #THREADS 2* CMOVE
 9   BEGIN   HERE #THREADS LARGEST   DUP
10   WHILE   DUP L>NAME DUP C@ 31 AND  ?LINE
11     .ID SPACE SPACE   @ SWAP !   KEY? IF  EXIT  THEN
12   REPEAT   2DROP   ;
13
14
15
```

```
    19
 0 \ Vocabulary management                           06aug86cht
 1 HEX
 2 : ALSO   CONTEXT DUP 2+ 8 CMOVE> ;
 3 : ONLY   CONTEXT 2+ 8 ERASE
 4   4030 CONTEXT !   ALSO ;
 5 DECIMAL
 6
 7
 8
 9
10
11
12
13
14
15
```

```
    0
 0 Utility                                           06aug86cht
 1 This file includes many important utilities needed for doing
 2 serious Forth programming.  Including these functions in a
 3 ROM-based Forth kernel would allow the system to be a
 4 complete and useful software development system.
 5
 6 Dr. C. H. Ting
 7 Offete Enterprises, Inc.
 8 1306 S. B Street
 9 San Mateo, CA  94402
10
11
12
13
14
15
```

```
    0
 0 Utility                                           06aug86cht
 1 This file includes many important utilities needed for doing
 2 serious Forth programming.  Including these functions in a
 3 ROM-based Forth kernel would allow the system to be a
 4 complete and useful software development system.
 5
 6 Dr. C. H. Ting
 7 Offete Enterprises, Inc.
 8 1306 S. B Street
 9 San Mateo, CA  94402
10
11
12
13
14
15
```

Listing 3. Utility (cont'd)

II. FORTH FOR THE HARRIS 80 COMPUTER

1. INTRODUCTION

Harris Corp. produced a series of minicomputers based upon 24 bit data and instruction format. With an extensive set of instructions for arithmetic operations, these computers are suited for scientific applications to process large amount of numeric data in real time. With the larger instruction words, memory references can be included in the instructions and the memory accessing overhead can be significantly reduced. The 24-bit data format also allows very efficient representation of floating point numbers. Most Harris computers has the scientific arithmetics process option which can perform extensive mathematical operations on single precision (24-bits) and double precision (48-bits) floating point numbers . Harris Model 80 Computer is the smallest member in this family. However, it executes the same instruction set as the other bigger members. The operating system provided by Harris is their VOS Operating System, supporting all the conventional programming languages, including FORTRAN, PASCAL, BASIC, and the native assembler. Although most of the languages run fairly fast, and compiling/linking/loading processes can be done using macro commands. It is still advantageous to build a FORTH environment in which program development can be carried out interactively.

This implementation of FORTH on the Harris 80 Computer is basically a transcription of the fig-FORTH for NOVA Computer by myself(1), released by the FORTH Interest Group in March, 1981. The NOVA implementation was chosen as the model for Harris due to the similarity in the architectures of these two CPU's, with roughly the same types of registers and similar addressing schemes. The design goal was set to follow the fig-FORTH model as closely as possible so that programs developed using fig-FORTH can be transported into the Harris computer with the minimal modifications. However, the Harris computer, being a 24 bit machine, is not quite compatible with the fig-FORTH model(2), which assumes a byte addressible machine. Most of the deviations from the fig-FORTH model are in the area of byte addressing instructions.

2. THE IMPLEMENTATION

The source program consists of about 1600 lines of assembly code. The dictionary after assembly occupies 3 Kwords of memory. The source file, named FORTH, can be assembled and executed by the following commands:

```
VA  FORTH

VX
```

If disk blocks are to be used in the FORTH system, a data file must be first assigned to logic file 64. This data file, if not present, can be generated by the following commands:

```
GE  DAT  U  G=10  M=200

AS 64=DAT
```

The DAT file is thus generated in unblocked form, with a capacity of 60 1024 byte blocks, which should be enough for most programming tasks. If the FORTH program was already assembled and vulcanized into the XE file, the command:

```
XE
```

will start the FORTH program with DAT file opened.

The FORTH registers are mapped to the Harris registers in the following order:

FORTH	Harris	Function
RP	I	Return stack pointer
SP	J	Data stack pointer
W	K	Current word pointer
IP	---	Instruction pointer
UP	---	User area pointer
---	A,E	Accumulators

The memory map referring to the origin of the FORTH program is:

Label	Memory	Function
ORIGIN	0	
NEXT	55	Code interpreter
LIT	63	Start of dictionary. Nucleus.
COLON	1021	Defining instructions
DOCOL	1036	Address interpreter
ONEP	1377	Level 1 instructions
COLD	3214	Cold start. Level 2 instructions
IO	4023	Low level I/O instructions
TICK	4170	Level 3 instructions
TASK	5062	End of dictionary
XDP	5067	Free dictionary space
XSO	14727	Top of data stack
XR#	15073	Top of return stack
XUP	15074	User area
IP	15240	Interpreter pointer

UP	15240	User area pointer
SAFE	15242	Register storage
DSKBUF	15250	First disk buffer
XUSE	17252	Second disk buffer
IOPAR	21254	I/O Parameter list
XTIB	21257	Terminal input buffer
TOB	21402	Terminal output buffer
END	21522	End of memory

3. IMPLEMENTATIONON PECULIARITIES

NEXT -- THE CODE INTERPRETER

NEXT moves the contents of IP into W register, increments IP, and then jumps indirectly through W register. IP points to the code field of the instruction to be executed. Since Harris does not allow a negative offset in the indirect jump BUC* , the W register will be pointing to the code field after NEXT. User has to increment it so that W can be used to address the parameter field.

For the convenience of debugging the program, all the code ending routines: PUSH, PUT, BINAR, POP2, and POP, terminate at NEXT. The only exception is EXECUTE, which jumps to the routine being executed directly.

THE NAME FIELD

The name fields of all instructions are created by a macro, HEAD. Its length is fixed at 2 words or 6 bytes. The format of name field is:

/ PS counts / char 1 / char 2 /

/ char 3 / char 4 / char 5 /

Up to five characters in the name are saved in the name field with the character count in the leftmost byte of the first word. If the name is less than five characters, the rest of the name field is blank filled. The most significant bit in the count byte is the precedence bit P, and the next bit is the smudge bit S.

5-character names are sufficient for most applications. The Harris computer can not handle bytes very efficiently. Putting names in two words does have the advantage that they can be manipulated as double integers and it does simplify the dictionary searching routine considerably.

BODY

BODY is the macro which generates the body of high level colon instructions. For some reason, the Harris Assembler would not accept label at the beginning of BODY. It is therefore necessary to put in labels whenever a branch address is needed, using the data statement DAC .

(FIND)

(FIND) searches through the dictionary for a matching name in the word buffer. It treats the name as a double precision integer and uses the double word subtract SMO to do the comparison. Before comparison, the precedence bit is masked out. The searching cycle is very tight and the searching process is rather fast because of the double integer comparison technique.

TERMINAL INPUT-OUTPUT

This FORTH system is designed to operate under the VOS operating system. All the I/O activities must go through the regular I/O channel calls in the VOS I/O service package. Therefore, the terminal I/O has to conform to the VOS I/O protocol. VOS terminal I/O assumes that the data transfers are processed by lines, not characters. Thus the primitive terminal I/O instruction in FORTH must be shifted to TYPE and EXPECT, from KEY and EMIT as prescribed in the FIG-FORTH model. Two buffers are dedicated to terminal I/O: TOB for terminal output and XTIB for terminal input. Both buffers are placed at the end of the memory map so that they will not have any adverse effect on the other data storage buffers in case they overflow.

EMIT puts one character into the TOB buffer and increments the output character count OUT. TYPE puts a character string into the TOB buffer and increments OUT by the character count. The actual output occurs when CR is executed. CR calls the code routine I/O to send the contents of TOB out to the logic device 3, dedicated to the user terminal.

EXPECT accepts a whole line of characters from the top of the Harris terminal display and stores them in the terminal input buffer XTIB. The XTIB buffer is 80 words or 240 bytes long. If the input line does not fill up the TIB buffer, the rest of the buffer is blank filled. However, there is no terminating or trailing zero appended to the string. This causes many problems for the text interpreter because flg- FORTH text interpreter assumes that there will be a zero at the end of an input line. Since I put a zero word at the end of XTIB, the text interpreter works fine because it does not mind the intervening blanks. But the user has to be aware of this peculiarity if he wanted to process the input stream himself, for example, to implement a text editor. The output buffer TOB is at the very end of the memory map, and is of 80 words or 240 bytes long. Harris outputs

characters in 3 byte chunks. The first byte in the buffer is used for printer carriage control. After CR, the OUT counter is set to 1 and the first byte is set to blank to take care of this abnormality.

?TERMINAL is not functional because Harris does not monitor the status of the keyboard. It always returns a false flag.

BYTE ADDRESSING AND WORD ADDRESSING

This is the biggest problem in transcribing the fig-FORTH model to Harris. The way Harris CPU addresses a byte in its memory is the following: the lower 16 bit field contains the word address and the most significant two bits, bit 22 and 23, are used to designate the byte location within a word. The way bytes are addressed is:

Bits 23 and 22	Byte selection
01	Leftmost byte (bits 23-16)
10	Middle byte (bits 15-8)
11	Rightmost byte (bits 7-0)
00	Rightmost byte on read. No action on write.

There is really no good way of handling the bytes in a word, as in the byte addressible PDP-11. The method used here to address individual bytes is to create a virtual byte memory space, in which bytes are serially arranged. Dividing the byte address by 3 and one gets the word address. The remainder of the division is incremented by one and put into the byte addressing field (bits 22 and 23). To find the virtual byte address from the Harris byte address, the byte addressing field is shifted into E register and the word address is multiplied by 3. The number in E register is decremented by one to get the true byte offset. If the byte offset is -1, meaning that the original byte field is 0, the byte offset is set to 0 to address the leftmost byte. Then the byte offset is added to the virtual byte address of the word, 3 times the word address, to form the virtual byte address of the byte.

The instruction converting a Harris byte address to the virtual byte address is BYTE, and the instruction converting the virtual byte address back to a Harris byte address is CELL. Any byte address calculation must be done in the virtual byte address space first by BYTE. After this calculation is done, revert to Harris address space by CELL. The sequence of instructions to access a byte in the memory is:

```
( byte-offset word-addr -- )  BYTE + CELL  ( -- byte-addr )
```

BYTE AND WORD ADDRESSING COMMANDS

Because of the special requirements to address bytes in this Harris computer, the memory addressing instructions are divided into two groups: the byte addressing instructions and the word addressing instructions. Each instruction must be used according to the addressing scheme assigned to it. If the schemes are not followed, one will either get the wrong information or the system will trap you and kick you back to VOS with a memory out of bound error message.

Byte Addressing Instructions

```
ENCLOSE          ( b-addr  c  --  b-addr  offset  end  next,
                   offset, end, and next are all byte addresses)
CMOVE            ( b-addr1 b-addr2 count -- )
C@               ( b-addr -- c)
TYPE             ( b-addr count  --  )
-TRAILING        ( b-addr count1 -- b-addr count2)
EXPECT           ( b-addr count -- )
FILL             ( b-addr count c -- )
ERASE            ( b-addr count -- )
BLANKS           ( b-addr count -- )
BLOCK            ( block# -- b-addr)
(LINE)           ( line# block#  -- b-addr  count)
#>               ( --  b-addr  count)
```

The following list of instructions use word addressing scheme:

```
CELL EXECUTE (FIND) MOVE +!  TOGGLE  @  !  user-variables  +ORIGIN
HERE  LATEST  LFA  NFA  PFA  CFA  PAD  (NUMBER) NUMBER  ID.  I/O  R/W
READB  WRITB  SECTR  XTYPE  XEXPE  '
```

The following instructions convert addressing modes:

```
BYTE             ( w-addr  --  virtual-b-addr)
CELL             ( virtual-b-addr  --  b-addr)
COUNT            ( w-addr --- b-addr count)
```

This problem in using the fig-FORTH model in word addressing computers is common to all the computers which cannot directly address bytes in the memory. The virtual byte address space seems to be a logical method to deal with this problem.

DOUBLE PRECISION INTEGERS

Harris has many machine instructions to handle double precision 48-bit integers, which are convenient for extended precision arithmetic operations. However, the double integer has a small twist in its format. When the A and E registers are used together as a double integer D register, the most significant bit in A must be zero. Thus the A register preserves only 23 bits of information instead of 24 bits. With a double integer on the stack, one cannot simply DROP the upper half of the number and expects that the lower

24 bits are preserved in the lower half of the double number, as assumed in fig-FORTH model. To retain 24 lower bits in a double integer on the stack, the proper instruction to use is D->S, not DROP. This instruction was used in INTERPRET and . so that signs of single precision integer are handled correctly.

VIRTUAL MEMORY

This is about the virtual memory in FORTH to use disk files, not the virtual memory demand paging in Harris, which presumably is handled by the VOS operating system.

FORTH accesses a logic file whose logical file number is 64 . This file must be generated in unblocked mode and associated to logic file 64 before entering into FORTH. It is recommended that 200 sectors be allocated to this file, but it can be made larger or smaller depending upon the application. The sector size is 112 words, which is rather inconvenient for a decent FORTH system, because 3 sectors have 1008 bytes, 16 bytes short of a FORTH block. What I did was to extend the disk buffers to 3 blocks or 1024 words, which is a triad or one page of FORTH texts. One buffer is then mapped to 10 sectors on the disk. This way 96 bytes will be wasted per block.

Every time a block of data is accessed from the file, all adjacent 3 blocks are read into a disk buffer. If one arranges texts according to the TRIAD format, in which related texts are arranged in pages of 3 blocks, this FORTH system will do very well. Nevertheless, the instructions BLOCK, LIST, and LOAD work as specified in fig-FORTH, just as UPDATE, FLUSH, and EMPTY-BUFFERS. BUFFER was not implemented.

Because three blocks of data reside together in one buffer, one has to break the loading sequence in the first two blocks explicitly by EXIT instruction at the end of texts. Otherwise, loading will continue into the next block, which might be useful as a default --> instruction. At the end of the 3 block buffer, there is a zero word to serve as the stopper.

The two buffers are pointed to by PREV and USE in the ping-pong fashion. When a block is requested, PREV buffer is checked first to see if the requested block is in this buffer. If it is not in PREV buffer, the contents of PREV and USE are exchanged and the new PREV buffer is checked, if the requested block is still not in this buffer, then this PREV buffer will be used to access disk file to obtain the requested block, flushing the contents of PREV buffer if any block in it was UPDATEd. This method assures that the PREV buffer is always the most recently accessed buffer.

4. SAMPLE LISTINGS

A listing generated by the command VLIST is shown in Fig. 1, showing all the commands implemented in this FORTH system. The headers of

most low level inner interpreter commands like LIT, (DO), (LOOP), etc., were removed from the system. It is not expected that users will ever use these commands in normal programming. These commands, if executed in the interpreting mode, will surely crash the system. It is better to leave these commands hidden.

A text editor was implemented using this FORTH system as a tool for writing FORTH programs. This editor is a full implementation of the editor discussed in Leo Brodie's "Starting FORTH"(3). One should note the use of the byte addressing and word addressing commands in handling text and string data. This editor and an assembler to build code definitions of Harris 80 machine instructions are shown in Listing 4. This FORTH system was used to control a Floating Point Array Processor AP-120B made by Floating Point System. It enables a user to access directly the hardware facilities provided in the Array Processor which now can be controlled interactively. The detailed description of this application(4) is also included in this book.

REFERENCES

1. C. H. Ting, 'fig-FORTH for NOVA Computer', FORTH Interest Group, 1981.

2. W. F. Ragsdale, 'fig-FORTH Model and Installation Manual', FORTH Interest Group. 1978.

3. L. Brodie, 'Starting FORTH', Prentice-Hall, 1981.

4. C. H. T!ing, 'An Interactive Operating System for AP-120B Array Processor Based on FORTH Language', 1984 ARRAY Conference Proceedings, Society of FPS Array Processor Users.

Figure 1. Forth Commands in Harris-FORTH

K	R	F	S	D	I	E	B	N	TILL	FOUND
SEARC	TOP	MATCH	-TEXT	L	COPY	M	U	X	P	T
KEEP	LINE	>TYPE	INSER	<CMOV	DELET	#LAGO	#LAG	TEXT	FBUF	IBUF
2SWAP	2DUP	2DROP	TASK	<EXPE	<TYPE	FORTH	;CODE	VLIST	TRIAD	INDEX
LIST	U.	?	.	D.	.R	D.R	#S	#	SIGN	#>
<#	SPACE	WHILE	ELSE	IF	REPEA	AGAIN	UNTIL	+LOOP	LOOP	DO
THEN	BEGIN	BACK	FORGE	'	CLOSE	OPEN	SECTR	WRITB	READB	R/W
I/O	-->	LOAD	MESSA	.LINE	'LINE	BLOCK	DR1	DRO	FLUSH	(FLUS
EMPTY	UPDAT	M/MOD	*/	*/MOD	MOD	/	/MOD	*	M/	M*
MAX	MIN	DABS	ABS	S->D	D->S	COLD	ABORT	QUIT	(	DEFIN
VOCAB	IMMED	INTER	?STAC	U<	DLITE	LITER	[COMP	CREAT	ID.	ERROR
(ABOR	-FIND	NUMBE	(NUMB	WORD	PAD	HOLD	BLANK	ERASE	FILL	
QUERY	EXPEC	."	(.")	-TRAI	TYPE	COUNT	DOES>	<BUIL	(;COD	OCTAL
DECIM	HEX	SMUDG	]	[	COMPI	?LOAD	?CSP	?PAIR	?EXEC	?COMP
?ERRO	!CSP	PFA	NFA	CFA	LFA	LATES	?DUP	SPACE	ROT	>
<	=	-	,	ALLOT	HERE	2+	1+	PREV	USE	HLD
R#	CSP	FLD	DPL	BASE	STATE	CURRE	CONTE	OFFSE	SCR	OUT
>IN	BLK	LIMIT	FIRST	VOC-L	DP	FENCE	WARNI	WIDTH	TIB	RO
SO	+ORIG	B/SCR	B/BUF	C/L	BL	3	2	1	0	USER
VARIA	CONST	;	:	CELL	BYTE	C!	!	C@	@	TOGGL
+!	DUP	SWAP	DROP	OVER	DNEGA	NEGAT	D+	+	0<	0=
R	R>	>R	LEAVE	EXIT	RP!	SP!	SP@	XOR	OR	AND
U/	U*	MOVE	CMOVE	CR	?TERM	KEY	EMIT	ENCLO	(FIND	DIGIT
I	(DO)	(+LOO	(LOOP	OBRAN	BRANC	EXECU	LIT			

(260 words)

```
SCR#  3
  Ø ( MINI-EDITOR, CHT, 7-OCT-83)
  1 : P    ( line --- )
  2         94 WORD HERE COUNT DROP    SWAP SCR @ (LINE) CMOVE ;
  3 : L    SCR @ LIST ;
  4 EXIT
  5
  6
  7
  8
  9
 1Ø
 11
 12
 13
 14
 15


SCR#  4
  Ø ( Error Messages)
  1 stack empty
  2 dictionary full
  3
  4 isn't unique
  5
  6
  7
  8
  9
 1Ø
 11
 12
 13
 14
 15 C. H. Ting                                          Harris 8Ø


SCR#  5
  Ø ( Error Message, Cont'd)
  1 compilation only
  2 execution only
  3 structure not paired
  4 definition not finished
  5 protected dictionary
  6 loading only
  7
  8
  9
 1Ø
 11
 12
 13 vector underflow
 14 declare vocabulary
 15

C. H. Ting                                              Harris 8Ø
```

Listing 4. Editor and Assembler for Harris-FORTH

```
SCR#  6
  Ø ( EDITOR LOAD BLOCK, CHT, 7-OCT-83)
  1 : 2DROP    DROP DROP ;
  2 : 2DUP     OVER OVER ;
  3 : 2SWAP    ROT >R ROT R> ;
  4 : 2OVER    >R >R   2DUP   R> R>   2SWAP    ;
  5 VOCABULARY EDITOR IMMEDIATE
  6 EDITOR DEFINITIONS
  7 7 LOAD ( TOOLS)
  8 8 LOAD ( LINE EDITOR)
  9 9 LOAD ( MATCH)
 1Ø 1Ø LOAD ( STRINGS)
 11 11 LOAD ( SRTING EDITOR)
 12 FORTH DEFINITIONS
 13 : L     EDITOR L     [COMPILE] EDITOR ;
 14 : THRU   1+ SWAP DO   CR I . I LOAD LOOP    ;
 15 EXIT


SCR#  7
  Ø ( EDITING TOOLS, CHT, 7-OCT,83)
  1 : IBUF PAD 3Ø + BYTE CELL ;    : FBUF PAD 6Ø + BYTE CELL ;
  2 : TEXT    ( badd --- baddr+1)   >R 94 WORD   HERE COUNT 64 MIN
  3     DUP HERE BYTE CELL C!
  4     -TRAILING   IF R 68 BLANKS   HERE BYTE CELL R 65 CMOVE   THEN
  5     DROP   R> BYTE 1+ CELL   ;
  6 : #LAG   SCR @ BLOCK BYTE R# @ + CELL   64 R# @ 63 AND - ;
  7 : #LAGØ ( --- baddr 64)   R# @ -64 AND R# !   #LAG ;
  8 : DELETE ( d d# s s# --- )   OVER MIN >R R -   OVER OVER OVER
  9     BYTE R + CELL ROT ROT CMOVE   SWAP BYTE + CELL R> BLANKS ;
 1Ø : <CMOVE   >R BYTE 1 - SWAP BYTE 1 - R + SWAP R> OVER + DO
 11     DUP CELL C@ I CELL C!   1 - -1 +LOOP   DROP ;
 12 : INSERT ( s s# d d# ---)   ROT OVER MIN >R R -   OVER DUP
 13     BYTE R + CELL ROT <CMOVE   R> CMOVE ;
 14 : >TYPE ( badd n ---)   >R PAD BYTE CELL R CMOVE
 15     PAD BYTE CELL R> TYPE ;          EXIT

SCR#  8
  Ø ( LINE EDITOR, CHT, 7-OCT-83)
  1 : LINE    CR SPACE   #LAG 64 SWAP -   >R BYTE R - CELL R> TYPE
  2     ." _"   #LAG >TYPE   R# @ 64 / .   ;
  3 : KEEP   #LAG IBUF OVER OVER C!   BYTE 1+ CELL SWAP CMOVE ;
  4 : T    15 AND 64 * R# !   LINE ;
  5 : P    IBUF TEXT   #LAGØ CMOVE   UPDATE ;
  6 : X    KEEP   #LAGØ 1Ø24 R# @ -   SWAP DELETE UPDATE ;
  7 : U     IBUF TEXT   64 R# +!   #LAGØ SWAP 1Ø24 R# @ - INSERT
  8     UPDATE ;
  9 : M    KEEP R# @ 64 + >R   SCR @ >R   64 * R# ! SCR !
 1Ø     U   R> SCR !   R> R# ! ;
 11 : COPY   SWAP BLOCK SWAP BLOCK 1Ø24 CMOVE UPDATE ;
 12 : L   SCR @ LIST ;
 13 EXIT
 14
 15
```

C. H. Ting Harris 8Ø

Listing 4. Editor and Assembler for Harris-FORTH (cont'd)

```
SCR#  9
  Ø ( -TEXT, MATCH, CHT, 7-OCT-83)
  1 : -TEXT    ( s s# d --- f )
  2     SWAP ?DUP IF
  3        ROT BYTE ROT BYTE ROT   OVER + SWAP DO
  4           DUP CELL C@   I CELL C@   -
  5           IF Ø= LEAVE   ELSE 1+ THEN
  6        LOOP
  7   ELSE   DROP Ø=   THEN   ;
  8 : MATCH    ( s s# d d# --- f d' )
  9     >R >R  2DUP R> R>   2SWAP   SWAP BYTE SWAP OVER + SWAP DO
 1Ø        2DUP I CELL -TEXT
 11        IF    I + >R   2DROP DROP   Ø R> Ø Ø LEAVE THEN
 12     LOOP   2DROP   OVER BYTE + CELL   ;
 13 EXIT
 14
 15


SCR#  1Ø
  Ø ( STRING EDITOR TOOLS, CHT, 7-OCT-83)
  1 : TOP   Ø R# ! ;
  2 : SEARCH   (   ---   )
  3     FBUF TEXT DROP   BEGIN
  4        #LAG OVER >R   FBUF COUNT   MATCH   BYTE R> BYTE -
  5        R# +!   R# @ 1Ø23 >   OVER Ø=   + Ø=   WHILE DROP
  6     REPEAT   ;
  7 : FOUND   SEARCH   IF TOP   FBUF COUNT TYPE
  8     ." ?"   QUIT   THEN   ;
  9 : TILL   R# @   FOUND   R# @ SWAP   DUP R# !
 1Ø     -   #LAG ROT   DELETE   LINE   UPDATE   ;
 11 EXIT
 12
 13
 14
 15


SCR#  11
  Ø ( STRING EDITOR, CHT, 7-OCT-83)
  1 : N   1 SCR +! ;
  2 : B   -1 SCR +! ;
  3 : E   FBUF COUNT SWAP DROP   DUP NEGATE R# +!
  4     #LAG ROT DELETE   LINE UPDATE   ;
  5 : I   IBUF TEXT   IBUF COUNT SWAP DROP   #LAG INSERT
  6     IBUF COUNT R# +! DROP   LINE UPDATE   ;
  7 : D   FOUND E  ;
  8 : S   ( block# --- )
  9     DUP SCR @ DO   SEARCH   IF N TOP
 1Ø     ELSE LINE SCR ? LEAVE   DUP   THEN   LOOP   DROP ;
 11 : F   FOUND LINE UPDATE ;
 12 : R   E I ;
 13 : K   IBUF PAD 6Ø MOVE   PAD FBUF 3Ø MOVE   ;
 14 EXIT
 15

C. H. Ting                                          Harris 8Ø
```

Listing 4. Editor and Assembler for Harris-FORTH (cont'd)

```
SCR#  12
  Ø ( ASSEMBLER LOAD BLOCK, CHT, 2Ø,OCT-83)
  1 VOCABULARY ASSEMBLER IMMEDIATE
  2 : CODE   CREATE [COMPILE] ASSEMBLER   ;
  3 OCTAL   : NEXT   21ØØØØ53 , SMUDGE   ;
  4 DECIMAL
  5 ASSEMBLER DEFINITIONS
  6 13 17 THRU
  7 FORTH DEFINITIONS
  8 EXIT
  9
 1Ø
 11
 12
 13
 14
 15


SCR#  13
  Ø ( PREASSEMBLER, CHT, 24-OCT-83)
  1 : ØOP    CONSTANT   DOES>   @  ,   ;
  2 : 1OP    CONSTANT   DOES>   @ + ,   ;
  3 : 1RO    CONSTANT   DOES>   @ SWAP 32768 * + + ,   ;
  4 : 1R     CONSTANT   DOES>   @ SWAP 32768 * + ,   ;
  5 : 2R     CONSTANT   DOES>   @ + SWAP 64 * + ,   ;
  6 : BIT    CONSTANT   DOES>   @ + SWAP 256 * + ,   ;
  7 : MEM    CONSTANT   DOES>   @ SWAP 262144 * + SWAP 32768 * +
  8          + ,   ;
  9 : #      131Ø72 HERE 1 - +!   ;
 1Ø
 11
 12
 13
 14
 15


SCR#  14
  Ø ( REGISTER DEFINITIONS, CHT, 24-OCT-83)
  1 1 CONSTANT I   2 CONSTANT J   3 CONSTANT K   4 CONSTANT E
  2 5 CONSTANT A   6 CONSTANT T   6 CONSTANT D   7 CONSTANT B
  3 8 CONSTANT R
  4 OCTAL
  5 1 CONSTANT RP   2 CONSTANT SP   4 CONSTANT W   1Ø CONSTANT ER
  6 2Ø CONSTANT AR   4Ø CONSTANT TR   3Ø CONSTANT DR
  7 DECIMAL
  8
  9
 1Ø
 11
 12
 13
 14
 15

C. H. Ting                                          Harris 8Ø
```

Listing 4. Editor and Assembler for Harris-FORTH (cont'd)

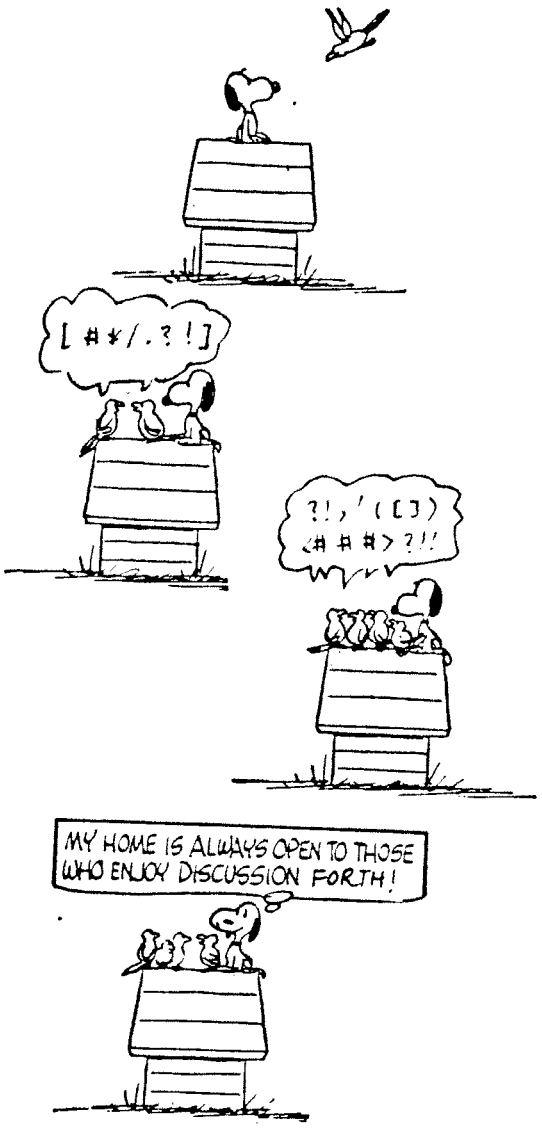
```
SCR#  15
  Ø ( MEMORY  INSTRUCTIONS, CHT, 24-OCT-83)
  1 OCTAL
  2 36ØØØØØØ MEM AMX    43ØØØØØØ MEM AXM
  3 46ØØØØØØ MEM SMX    26ØØØØØØ MEM CMX    63ØØØØØØ MEM IMX
  4 Ø MEM TMX    1ØØØØØØØ MEM TXM
  5 64ØØØØØØ 1RO AOR    65ØØØØØØ 1RO SOR    62ØØØØØØ 1RO TOR
  6 63ØØØØØØ 1RO TNR    61ØØØØØØ 1RO DVR    6ØØØØØØØ 1RO MYR
  7 23ØØØØØØ 1RO BWX    24ØØØØØØ 1RO BLX
  8 2ØØØØØ 2R ARR    22ØØØØ 2R NRR    23ØØØØ 2R PRR    75ØØØØ 2R RRR
  9 21ØØØØ 2R SRR    24ØØØØ 2R CRR    25ØØØØ 2R KRR    26ØØØØ 2R DRR
 1Ø 3ØØØØØ 2R ORR    27ØØØØ 2R XRR    35ØØØØ 2R IRR    3ØØØØØ 2R TRR
 11 734ØØØ 2R IAW    72ØØØØ 2R IDW    734Ø4Ø 2R IPW    73ØØØØ 2R ISW
 12 714ØØØ 2R OAW    7ØØØØØ 2R OCW    71ØØØØ 2R ODW
 13 DECIMAL
 14
 15


SCR#  16
  Ø ( ASSEMBLER MNEMONICS, CHT, 25-OCT-83)
  1 OCTAL
  2 61ØØØØØØ 1OP DVO    6ØØØØØØØ 1OP MYO    ØØ13ØØØØ 1OP SOB
  3 6ØØ7ØØØØ 1OP BBI    617ØØØØØ 1OP BBJ    ØØ67ØØØØ 1OP BLU
  4 225ØØØØØ 1OP BNN    224ØØØØØ 1OP BNO    227ØØØØØ 1OP BNP
  5 226ØØØØØ 1OP BNZ    221ØØØØØ 1OP BON    22ØØØØØØ 1OP BOO
  6 223ØØØØØ 1OP BOP    222ØØØØØ 1OP BOZ    21ØØØØØØ MEM BUC
  7 ØØ14ØØØØ 1OP COB    ØØ15ØØØØ 1OP KOB    ØØ16ØØØØ 1OP DOB
  8 ØØØ4ØØØØ 1OP OOB    ØØ17ØØØØ 1OP XOB
  9 ØØ4ØØØØØ 1OP LAA    ØØ46ØØØØ 1OP LAD    ØØ42ØØØØ 1OP LLA
 1Ø ØØ5ØØØØØ 1OP LLD    ØØ44ØØØØ 1OP LRA    ØØ52ØØØØ 1OP LRD
 11 ØØ41ØØØØ 1OP RAA    ØØ47ØØØØ 1OP RAD    ØØ43ØØØØ 1OP RLA
 12 ØØ51ØØØØ 1OP RLD    ØØ45ØØØØ 1OP RRA    ØØ53ØØØØ 1OP RRD
 13 DECIMAL
 14
 15


SCR#  17
  Ø ( ASSEMBLER MNEMONICS, CHT, 25-OCT-83)
  1 OCTAL
  2 31ØØØØØØ 1 OP EMB    27ØØØØØØ 1OP RBM    ØØØ3ØØØØ 1OP TOB
  3 ØØ36ØØØØ 1 OP TOC    236ØØØØØ 1OP TLO    46ØØØØØØ 1OP TFM
  4 66ØØØØØØ 1 OP TZM    ØØ11ØØØØ 1OP QBB
  5 ØØ24ØØØØ 1 OP CZR    31ØØØØØØ 1OP TSR    ØØ3ØØØØØ 1OP TZR
  6 775ØØØØØ BIT DMH    7752ØØØØ BIT DNH    77542ØØØ BIT OMH
  7 7756ØØØØ BIT XMH    7762ØØØØ BIT XNH    7764ØØØØ BIT TMH
  8 7764ØØØØ BIT TMH    7766ØØØØ BIT QBM    777ØØØØØ BIT THM
  9 7772ØØØØ BIT FBM    7774ØØØØ BIT ZBM
 1Ø DECIMAL
 11
 12
 13
 14
 15

C. H. Ting                                          Harris 8Ø
```

Listing 4. Editor and Assembler for Harris-FORTH (cont'd)

```
SCR#  18
  Ø ( DUMP, CHT, 1-NOV-83)
  1 : U.R    ( N COL --- )
  2         OVER Ø< IF 1 ELSE Ø THEN SWAP    D.R    ;
  3 : DLINE     ( addr ---)
  4         8 OVER + SWAP DO    I @ 8 U.R    LOOP    ;
  5 : DUMP    ( addr count --- )
  6         OVER + SWAP DO    CR I 8 U.R 2 SPACES
  7         I DLINE    8 +LOOP    ;
  8 : 2*    DUP + ;
  9 : EXPONENT    ( byte --- n )    DUP 127 AND SWAP 128 AND
 1Ø         IF -128 ELSE Ø THEN OR ;
 11 : F.     ( F --- )     SWAP EXPONENT >R    8 .R
 12         R>    ." E" 3 .R    ;
 13 EXIT
 14
 15
```

Listing 4. Editor and Assembler for Harris-FORTH (cont'd)

FORTH INTEREST GROUP



III. VAX AS A 32 BIT FORTH MACHINE

1. CISC IS NOT A DIRTY WORD

Currently, CISC (Complicated Instruction Set Computer) seems to be a dirty word while RISC (Reduced Instruction Set Computer) becomes fashionable and popular among the computer scientists. This wind is so strong that many Forth enthusiasts labelled the Novix NC4000 as a RISC computer in order to emphasize the fact that NC4000 executes most of its instructions in a single clock cycle.

It is not the purpose of this paper to argue the merits or demerits of either RISC or CISC. The purpose is to point out that Forth has a extremely strong CISC flavor, when viewed from the angle of virtual machine and instruction set. In a Forth application, the instruction set is as complicated as the application, because the instruction set is the collection of all the words making up the application. Even the Forth kernel, which forms the platform supporting the application, is rather large, containing about 200 words.

The most fundamental property of Forth is its extensibility, that the user can create new instructions and adds them to his Forth system to solve his own problem. Extensibility is not a property of RISC structure. CISC structure can be built to include most Forth kernel words and to form a very efficient Forth engine, as in Phil Koopman's WISC (Writable Instruction Set Computer). The remaining argument is how many instructions are to be implemented in hardware and how many are best left to the software.

2. MATCHING CISC MACHINES TO FORTH

We can examine available CISC computers and microprocessors and match their instruction sets against the Forth instruction set as defined in Forth-83. Different machines can thus be ranked as to the ease in implementing a Forth on them. A partial ranking list is shown in Figure 2:

Figure 2. Ranking of CISC Machines

Machine	Comments
VAX	32 bit, many instructions match high level Forth words.
NC4000	16 bit optimized Forth engine.
680x0	32 bit PDP-11 clone.
80386	32 bit ding-dong.
PDP-11	16 bit stack oriented machine.
80x86/8088	16 bit ding-dong.
Z80	8 bit enhanced ding-dong.
8080/8085	8 bit ding-dong.
6502/6800	8 bit first generation micro.

One might be surprise at that VAX is ranked above NC4000. It is true that NC4000 has the architecture best suited for Forth. However, only the elementary stack operations, the ALU operations, and the subroutine call/return, looping, conditional jump and unconditional jump instructions are implemented as machine instructions. Other types of more complicated Forth words have to be synthesized from these elementary operations, just like all the other microprocessors.

VAX inherited and expanded on many of the nice features in PDP-11, like byte addressiblity, multiple stack pointers, single instruction multiply and divide, and all the logic and arithmetic operations. In addition, VAX has a very strong instruction set to handle strings, which is very useful in implementing the Forth interpreter and compiler. It also handles many different types of data, bits, bytes, 16 bit words, 32 bit long words, 64 bit quad words, 32 bit floating point, and 64 bit floating point. The 32/64 bit integer and floating point capability makes VAX more like a mainframe than a micro- or mini-computer.

In going through the VAX instruction set, the impression I got was that VAX is not only a good virtual machine for the bare-bone Forth as spelled out in the various Forth standards, it is also a very good virtual machine for F83, which is much more extensive than the bared Forth-83 Standard. In the next section, let see how the VAX instruction set matches with the F83 kernel instruction set.

3. VAX AS A F83 ENGINE

VAX is a 32 bit machine, because all the registers are 32 bit wide, and it has an address space of 2**31 bytes. Since it can access data and operate on them as 16 bit quantities, it

is possible to implement the 16 bit F83 faithfully on VAX. However, as the world is moving in the direction of 32 bit machines, we should take the advantage of 32 bit architecture and extend the F83 model to 32 bits. Here we shall assume that all numbers and addresses are 32 bit quantities unless otherwise specified explicitly.

Listing 5 shows most of the F83 kernel words implemented in the assembly code of VAX. Most of them were code words in F83, but still a large portion were implemented in high level because the operations are quite complicated. Many of these high level kernel words can be reduced to very simple code words in VAX, due to the vast and complicated instruction set of VAX.

Figure 3 shows a list of F83 high level kernel words which have corresponding VAX machine instructions or very close approximations. This list is the basis of my claim that VAX is the best host for 32 bit F83.

Figure 3. F83 Kernel Words and Their VAX Counterparts.

VAX Machine Code	F83 Words
MOVL	Most Stack Words
MOVZ	S>D
CMPL/CMPD	=, >, <, ><, D=, D>, D<
TSTL/TSTD	0=, 0>, 0<, D0=, D0>, D0<
ADDL/ADDD	+, D+
SUBL/SUBD	-, D-
INCL/DECL	1+, 1-
EMUL	UM*, D*
EDIV	UM/MOD, /MOD, MOD
DIVL2	/
MULL2	*
BICL2	AND, CRESET
BISL2	OR, CSET
XORL2	XOR, CTOGGLE
ASHL/ASHD	2*, 2/, D2*, D2/
BEQL	ZBRANCH
BR	BRAMCH
ACBLEQ	+LOOP
AOBLEQ	LOOP
CASE	CASE:
MOVC3	CMOVE
MOVTC	FILL, ERASE, BLANK UPPER, DIGIT,
MOVTUC	PARSE, WORD
CMPC3	COMP, COMPARE
LOCC	SCAN
SKPC	SKIP
MATCHC	SEARCH

The F83 words in Figure 3 can thus be greatly simplified in VAX code, as shown in Listing 5. There are still some overhead in moving data between the data stack and the CPU registers, which is unavoidable because of the inconsistency between a true stack machine and a register based machine.

Most of the F83 words in VAX code assume that the widths of numbers and addresses are 32 bits. Double precision words are 64 bit wide. As the addresses in VAX are byte addresses, VAX is ideally suited to host figForth and its derivatives, including F83, which all assume byte addressability.

Another issue is indirectly threaded code vs. directly threaded code vs. subroutine threaded code. If we take the subroutine threaded approach, many of the Forth words can be most conveniently coded as in-line machine code. It is also possible to use the flags in the status register to cause conditional jump or branch, thus reducing IF, ELSE, UNTIL, etc., to single machine instructions. These approaches are interesting possibilities for us to explore.

4. CONCLUDING REMARKS

It is interesting to observe that many of the complex instructions in VAX are very useful instructions required by Forth as its primitive instructions. These instructions are implemented in VAX because of necessity, which are common to all operating systems and languages. It is nice to see the VAX instruction set underscores many F83 words, confirming that the basic design of F83 is well thought out.

F83 is also capable of being extended towards 32 bit host computers. The understanding is that numbers and addresses are now 32 bits wide instead 16 bits wide, and that double numbers mean 64 bit numbers. It can also serve as a testbed in extending Forth standard to the 32 bit world while retain compatibility with the 16 bit models used in old standards.

Forth is a very flexible structure, which can take full advantage of the host processor to build an interactive programming environment for the end user. It will be more efficient if the host processor can provide more service. In this sense, CISC computers can be put to good use and we should not hesitate to exploit them.

```
                .default        displacement,long
;               rp=r11          ;return stack pointer
;               ip=r10          ;instruction pointer
;               wp=r9           ;word pointer
;               up=r8           ;user area pointer
;               dp=r7           ;double word pointer
;               qp=r6           ;quad word pointer
                .psect  vax32,wrt,exe,long
                .entry  vax32,^m<r6,r7,r8,r9,r10,r11>
;               .macro  next
;               movl    (r10)+,r9
;               jmp     (r9)+
;               .endm

                .macro  next
                jmp     xnext
                .endm

push2:          pushl   r2
push1:          pushl   r1
push0:          pushl   r0
xnext:          movl    (r10)+,r9
                jmp     (r9)+

nest:           movl    r10,-(r11)
                movl    r9,r10
                next
unnest:         movl    (r11)+,r10
                next
dodoes:         movl    r10,-(r11)
                movl    (sp)+,r10
docreate:       pushl   r9
                next
doconstant:     pushl   (r9)
                next
douservariable: addl3   (r9),r8,-(sp)
                next
xlit:           pushl   (r10)+
                next
branch:         movl    (r10),r10
                next
zbranch:        tstl    (sp)+
                beql    branch
                tstl    (r10)+  ;increment r10
                next
xloop:          aobleq  4(r11),(r11),branch
ploop:          cmpl    (r11)+,(r11)+
                cmpl    (r11)+,(r10)+   ; increment
                next
xploop:         acbl    4(r11),(sp)+,(r11),branch
                jmp     ploop
xdo:            movl    (r10)+,-(r11)
                popl    r0
                popl    -(r11)
                movl    r0,-(r11)
                next
xqdo:           cmpl    (sp),4(sp)
                bgeq    xdo
                jmp     branch
execute:        popl    r9
```

Listing 5. Forth kernel in VAX machine code

```
                jmp     (r9)+
perform:        popl    r9
dodefer:        movl    (r9),r9
                jmp     (r9)+
douserdefer:    addl3   (r9),r8,r9
                jmp     dodefer
go:             ret
noop:           next
pause:          next
xi:             movl    (r11),-(sp)
                next
xj:             movl    12(r11),-(sp)
                next
xleave:         cmpl    (r11)+,(r11)+   ;pop return stack
                movl    (r11)+,r10      ;exit address
                next
xqleave:        tstl    (sp)+
                bneq    xleave
                next
fetch:          movl    (sp),r0
                movl    (r0),(sp)
                next
store:          movl    (sp)+,r0
                movl    (sp)+,(r0)
                next
cfetch:         movl    (sp),r0
                movb    (r0),(sp)
                next
cstore:         movl    (sp)+,r0
                movb    (sp),(r0)
                tstl    (sp)+
                next
cmove:          popr    #^m<r0,r1,r2>
xcmove:         movb    (r2)+,(r1)+
                sobgeq  r0,xcmove
                next
cmover:         popl    r0
                addl3   (sp)+,r0,r1
                addl3   (sp)+,r0,r2
xcmover:        movb    -(r2),-(r1)
                sobgeq  r0,xcmover
                next
spfetch:        movl    sp,-(sp)
                next
spstore:        movl    (sp),sp
                next
rpfetch:        pushl   r11
                next
rpstore:        movl    (sp)+,r11
                next
drop:           tstl    (sp)+
                next
dup:            movl    (sp),-(sp)
                next
swap:           popl    r1
                popl    r0
                jmp     push1
over:           movl    4(sp),-(sp)
                next
tuck:           popl    r0
```

Listing 5. Forth kernel in VAX machine code (cont'd)

```
                popl    r1
                movl    r0,r2
                jmp     push2
nip:            movl    (sp)+,(sp)
                next
rot:            popl    r1
                popl    r2
                popl    r0
                jmp     push2
nrot:           popl    r0
                popl    r1
                popl    r2
                next
flip:           movb    (sp),r0         ;flip lower two bytes
                movb    1(sp),r1
                movb    r0,1(sp)
                movb    r1,(sp)
                next
tor:            movl    (sp)+,-(r11)
                next
rfrom:          pushl   (r11)+
                next
rfetch:         pushl   (r11)
                next
pick:           movl    (sp)+,r0
                pushl   (sp)[r0]
                next
roll:           movl    (sp)+,r0
                movl    (sp)[r0],-(sp)
                incl    r0
                addl3   sp,r0,r1
                addl3   #1,r1,r2
proll:          movl    -(r1),-(r2)
                sobgeq  r0,proll
                tstl    (sp)+
                next
xand:           bicl2   (sp)+,(sp)
                next
oor:            bisl2   (sp)+,(sp)
                next
xxor:           xorl2   (sp)+,(sp)
                next
xnot:           mcoml   (sp),(sp)
                next
cset:           movl    (sp)+,r0
                movl    (sp)+,r1
                bisb2   r1,(r0)
                next
creset:         movl    (sp)+,r0
                mcoml   (sp)+,r1
                bicb2   r1,(r0)
                next
ctoggle:        movl    (sp)+,r0
                movl    (sp)+,r1
                xorb2   r1,(r0)
                next
xon:            movl    #-1,(sp)+
                next
xoff:           clrl    (sp)+
                next
```

Listing 5. Forth kernel in VAX machine code (cont'd)

```
plus:           addl2   (sp)+,(sp)
                next
minus:          movl    (sp)+,r0
                subl2   (sp),r0
                jmp     push0
negate:         mnegl   (sp),(sp)
                next
abs:            tstl    (sp)
                blss    negate
                next
plusstore:      movl    (sp)+,r0
                addl2   (sp)+,(r0)
                next
zero:           clrl    -(sp)
                next
one:            pushl   #1
                next
two:            pushl   #2
                next
three:          pushl   #3
                next
twostar:        ashl    #1,(sp),(sp)
                next
twoslash:       ashl    #-1,(sp),(sp)
                next
utwoslash:      bicl2   #1,(sp)
                rotl    #-1,(sp),(sp)
                next
fourstar:       ashl    #2,(sp),(sp)
                next
eightstar:      ashl    #3,(sp),(sp)
                next
twoplus:        incl    (sp)
oneplus:        incl    (sp)
                next
twominus:       decl    (sp)
oneminus:       decl    (sp)
                next
umstar:         emul    (sp)+,(sp)+,#0,r0
                pushl   r0
                pushl   r1
                next
umslashmod:     movl    (sp)+,r0
                movq    (sp)+,r1
                ediv    r0,r1,r3,-(sp)
                pushl   r3
                next
zeroequal:      tstl    (sp)+
                beql    xtrue
                jmp     xfalse
zeroless:       tstl    (sp)+
                blss    xtrue
                jmp     xfalse
zerogreater:    tstl    (sp)+
                bgtr    xtrue
                jmp     xfalse
zeronotequal:   tstl    (sp)+
                bneq    xtrue
                jmp     xfalse
xequal:         cmpl    (sp)+,(sp)+
```

Listing 5. Forth kernel in VAX machine code (cont'd)

```
                    beql    xtrue
                    jmp     xfalse
xnotequal:          cmpl    (sp)+,(sp)+
                    bneq    xtrue
                    jmp     xfalse
qnegate:            tstl    (sp)+
                    beql    xnegate
                    mnegl   (sp),(sp)
xnegate:            next
uless:              cmpl    (sp)+,(sp)+
                    blssu   xtrue
                    jmp     xfalse
ugreater:           cmpl    (sp)+,(sp)+
                    bgtru   xtrue
                    jmp     xfalse
less:               cmpl    (sp)+,(sp)+
                    blss    xtrue
                    jmp     xfalse
greater:            cmpl    (sp)+,(sp)+
                    bgtr    xtrue
                    jmp     xfalse
true:
xtrue:              xorl2   -(sp),(sp)
                    next
false:
xfalse:             clrl    -(sp)
                    next
min:                movl    (sp)+,r0
                    cmpl    (sp),r0
                    blss    xmin
                    movl    r0,(sp)
xmin:               next
max:                movl    (sp)+,r0
                    cmpl    (sp),r0
                    bgtr    xmax
                    movl    r0,(sp)
xmax:               next
failure:            clrl    (sp)
                    next
between:            movl    (sp)+,r0
                    cmpl    (sp)+,r0
                    blss    failure
                    cmpl    (sp)+,r0
                    bgeq    xtrue
                    jmp     xfalse
within:             movl    (sp)+,r0
                    cmpl    (sp)+,r0
                    bleq    failure
                    cmpl    (sp)+,r0
                    bneq    xtrue
                    jmp     xfalse
twofetch:           movl    (sp),r0
                    movl    4(r0),(sp)
                    movl    (r0),-(sp)
                    next
twostore:           movl    (sp)+,r0
                    movl    (sp)+,(r0)+
                    movl    (sp)+,(r0)
                    next
twodrop:            cmpl    (sp)+,(sp)+
```

Listing 5. Forth kernel in VAX machine code (cont'd)

```
                next
twodup:         movl    4(sp),-(sp)
                movl    4(sp),-(sp)
                next
twoswap:        popr    #^m<r0,r1,r2,r3>
                pushr   #^m<r0,r1>
                pushr   #^m<r2,r3>
                next
fourdup:        movl    12(sp),-(sp)
                movl    12(sp),-(sp)
twoover:        movl    12(sp),-(sp)
                movl    12(sp),-(sp)
                next
threedup:       movl    8(sp),-(sp)
                movl    8(sp),-(sp)
                next
tworot:         popr    #^m<r0,r1,r2,r3,r4,r5>
                pushr   #^m<r0,r1,r2,r3>
                pushr   #^m<r4,r5>
                next
dplus:          popr    #^m<r0,r1>
                addl2   r1,4(sp)
                adwc    r0,(sp)
                next
dnegate:        mnegl   (sp)+,r0
                mnegl   (sp)+,r1
                incl    r1
                adwc    #0,r0
                jmp     push1
stod:           tstl    (sp)
                blss    stod1
                jmp     xfalse
stod1:          jmp     xtrue
dabs:           tstl    (sp)
                blss    dnegate
                next
dtwostar:       popl    r1
                popl    r0
                ashq    #1,r0,r0
xdtwostar:      pushl   r0
                pushl   r1
                next
dtwoslash:      popl    r1
                popl    r0
                ashq    #-1,r0,r0
                jmp     xdtwostar
dminus:         popr    #^m<r0,r1>
                subl2   r1,4(sp)
                sbwc    r0,(sp)
                next
qdnegate:       tstl    (sp)+
                blss    dnegate
                next
dzeroequal:     bisl3   (sp)+,(sp)+,r0
                bneq    yfalse
ytrue:          jmp     xtrue
yfalse:         jmp     xfalse
dequal:         cmpd    (sp)+,(sp)+
                beql    ytrue
                jmp     xfalse
```

Listing 5. Forth kernel in VAX machine code (cont'd)

```
duless:         cmpc3   #8,7(sp),15(sp)
                blss    ytrue
                jmp     xfalse
dless:          jmp     duless  ;cheating
dgreater:       cmpc3   #8,7(sp),15(sp)
                bgtr    ytrue
                jmp     xfalse
dmin:           cmpl    (sp)+,(sp)+
                next            ;cheating
dmax:           cmpl    (sp)+,(sp)+
                next            ;cheating
stard:          emul    (sp)+,(sp)+,#0,r0
                pushl   r0
                pushl   r1
                next
mslashmod:      popr    #^m<r0,r1,r2>
                movl    r1,r3
                ediv    r0,r2,r1,-(sp)
                pushl   r1
                next
muslashmod:     jmp     mslashmod
star:           mull2   (sp)+,(sp)
                next
slashmod:       popr    #^m<r0,r1>
                clrl    r2
                tstl    r1
                bgeq    xslashmod
                decl    r2
xslashmod:      ediv    r0,r1,r3,-(sp)
                pushl   r3
                next
slash:          divl2   (sp)+,(sp)
                next
mod:            popr    #^m<r0,r1>
                clrl    r2
                tstl    r1
                bgeq    xmod
                decl    r2
xmod:           ediv    r0,r1,r3,-(sp)
                next
starslashmod:   popr    #^m<r0,r1,r2>
                clrq    r3
                emul    r1,r2,r3,r3
                ediv    r0,r3,r1,-(sp)
                pushl   r1
                next
starslash:      popr    #^m<r0,r1,r2>
                clrq    r3
                emul    r1,r2,r3,r3
                ediv    r0,r3,-(sp),r1
                next
fill:           movc5   4(sp),8(sp),(sp)+,(sp)+,(sp)
                next
erase:          movc5   (sp),4(sp),#0,(sp)+,(sp)+
                next
blank:          movc5   (sp),4(sp),#32,(sp)+,(sp)+
                next
count:          movb    (sp),r0
                incl    sp
                jmp     push0
```

Listing 5. Forth kernel in VAX machine code (cont'd)

```
length:      movl    (sp),r0
             addl2   #4,sp
             jmp     push0
upcase:      .byte   0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
             .byte   16,17,18,19,20,21,22,23,24,25,26,27,28
             .byte   29,30,31,32,33,34,35,36,37,38,39,40
             .byte   41,42,43,44,45,46,47,48,49,50,51,52
             .byte   53,54,55,56,57,58,59,60,61,62,63,64
             .byte   65,66,67,68,69,70,71,72,73,74,75,76
             .byte   77,78,79,80,81,82,83,84,85,86,87,88
             .byte   89,90,91,92,93,94,95,96,65,66,67,68
             .byte   69,70,71,72,72,74,75,76,77,78,79,80
             .byte   81,82,83,84,85,86,87,88,89,90
             .byte   121,122,123,124,125,126,127
             .byte   255,255,255,255,255,255,255,255,255,255
             .byte   255,255,255,255,255,255,255,255,255,255
             .byte   255,255,255,255,255,255,255,255,255,255
             .byte   255,255,255,255,255,255,255,255,255,255
             .byte   255,255,255,255,255,255,255,255,255,255
             .byte   255,255,255,255,255,255,255,255,255,255
             .byte   255,255,255,255,255,255,255,255,255,255
             .byte   255,255,255,255,255,255,255,255,255,255
xdigit:      .byte   255,255,255,255,255,255,255,255,255,255
             .byte   255,255,255,255,255,255,255,255,255,255
             .byte   255,255,255,255,255,255,255,255,255,255
             .byte   255,255,255,255,255,255,255,255,255,255
             .byte   255,255,255,255,255,255,255,255
             .byte   0,1,2,3,4,5,6,7,8,9,255,255,255,255,255
             .byte   255,255,10,11,12,13,14,15,16,17,18,19,20
             .byte   21,22,23,24,25,26,27,28
             .byte   29,30,31,32,33,34,35,36,37,38,39,40
             .byte   41,42,43,44,45,46,47,48,49,50,51,52
             .byte   53,54,55,56,57,58,59,60,61,62,63,64
             .byte   65,66,67,68,69,70,71,72,73
             .byte   255,255,255,255,255,255,255,255,255,255
             .byte   255,255,255,255,255,255,255,255,255,255
             .byte   255,255,255,255,255,255,255,255,255,255
             .byte   255,255,255,255,255,255,255,255,255,255
             .byte   255,255,255,255,255,255,255,255,255,255
             .byte   255,255,255,255,255,255,255,255,255,255
             .byte   255,255,255,255,255,255,255,255,255,255
             .byte   255,255,255,255,255,255,255,255,255,255
             .byte   255,255,255,255,255,255,255,255,255,255
             .byte   255,255,255,255,255,255,255,255,255,255
             .byte   255,255,255,255,255,255,255,255,255,255
             .byte   255,255,255,255,255,255,255,255,255,255
             .byte   255,255,255,255,255,255,255,255
upper:       movtc   (sp),4(sp),#255,upcase,(sp)+,(sp)+
             next
digit:       movtc   #1,4(sp),#0,xdigit,#1,4(sp)
             cmpl    (sp)+,(sp)
             bgeq    ztrue
             jmp     xfalse
dtrailing:   addl3   4(sp),(sp),r0
xtrailing:   cmpb    -(r0),#32
             bneq    ytrailing
             sobgtr  (sp),xtrailing
ytrailing:   next
comp:        cmpc3   (sp)+,(sp)+,(sp)+
             beql    zfalse
```

Listing 5.   Forth kernel in VAX machine code (cont'd)

```
                blss    ztrue
                pushl   #1
                next
compare:        jmp     comp    ;case sensitive
skip:           skpc    (sp)+,(sp)+,(sp)+
                jmp     pushl
scan:           locc    (sp)+,(sp)+,(sp)+
                jmp     pushl
search:         movl    (sp),r4
                matchc  (sp)+,(sp)+,(sp)+,(sp)+
                bneq    search1
                subl3   r3,r4,-(sp)
ztrue:          jmp     xtrue
search1:        pushl   r3
zfalse:         jmp     xfalse
                .end    vax32
```

Listing 5. Forth kernel in VAX machine code (cont'd)

ASSEMBLER

IV. 8086 DISASSEMBLER

1. INTRODUCTION

Disassemblers are for the programming thieves, and those reverse-engineers who try to steal other peoples' work. I have to admit some tendency to both. The reason of doing this 8086 disassembler was to look at one of the BIOS rom's in my PC clone. Why did I want to look at it? I was interested in target compiling a Forth system to be run on the clone. The program worked somewhat, but it still needed the BIOS rom to bring it up. Since my original purpose was to replace the BIOS rom with the Forth rom, it was a puzzle to me why the Forth rom did not work by itself. I was able to identify that my Forth rom was caught in a loop waiting for the floppy disk drive to be initialized. I tried to following exactly what the IBM BIOS does in initializing all the peripheral devices, but of little avail. Since I did have this clone with a different BIOS in it, a detailed analysis of this rom might be helpful.

At that time, I had an early polyForth II system running on the clone which was all the tools I got one this machine. I was able to dump the entire rom code, but looking at the hex dump was not something to be enjoyed. Since I had seen a few disassemblers in Forth for a number of 8 bit microprocessors, I thought that it should not be a big job to write a disassembler for 8086. Writing the assembler I did. I also disassembled the entire clone BIOS. However, I never did found what was wrong with my Forth rom, which added to the array of my unfinished projects.

Among the reasons why this Forth rom project was abandoned, one important one was my conversion to F83 from polyForth. PolyForth II for IBM-PC is a very nice system. It is very fast. It has many good features, such as direct disk access, extended memory addressing, a true multitasker, and a target compiler. However, because it is a proprietary system and the disk format does not allow me to exchange programs and other information with other Forth users, I felt isolated using it. F83 is in the public domain, and I can freely discuss its internal structures without the fear that somebody's going after my tail if I talked too much. Since F83 seems to fill all my needs in serious and entertaining programming, I decided to use it exclusively. PolyForth thus went to storage, and this disassembler was the last major projects using it.

2. THE DESIGN OF 8086 DISASSEMBLER

Intel Corp. published a very handy reference guide for 8086 assembly language in 1978, the title is 'MCS-86 Assembly Language Reference Manual.' It measures 3.5" by 6". In 20 small pages, it contains every thing you need to know about the 8086 assembler, although sometimes you need a magnifying glass to read it. The last two pages have a table titled '8086 Instruction Set Matrix', which is reproduced in Figure 4. It summarizes the 8086 instructions in a matrix of 16 by 16 entries. A closer examination shows that the 16 columns can be grouped into two panels, and there are basically 32 major types of instructions. The disassembler is organized accordingly.

Unfortunately, Intel did not really believe in this matrix. There are all kinds of exceptions and 'enhancements,' making it difficult for the disassembler. The worst disease in 8086 architecture is the different addressing modes and types of a single instruction, which can be from one byte to five bytes long. Thus to disassemble all of them correctly is not a trivial task.

However, in Forth everything's possible. It Intel can build a chip using this instruction set, I shall disassemble it. All it takes are little case statements nested in bigger case statements, and so forth. A concise and efficient case structure is required. My favorite case structure is the 'positional case structure' CASE: as defined in F83 and shown in Screen 5. It is compiled exactly like a colon definition. When a case word is executed, it takes the top element on the stack, indexes into the word array in the parameter field, and executes the word selected. This case structure is used extensively in the disassembling process.

The most significant 5 bits in an 8086 instruction is used to determine the instruction type, and they are used to select and execute one among the 32 <ntype> words inside the case word <DECODE> in Screen 23. Many of these <nTYPE> words in turn are case words and continue the selection process until the name and the argument list are printed. The final word DECODE behaves like DUMP. It takes an address and a byte count, and disassemble that many bytes into a list of assembler instructions.

3. THE DISASSEMBLER AND EXAMPLE

The source code of the disassembler is show in the Listing 6 and a sample of its output is shown in Figure 5. The source code has quite detailed comments in the shadow screens, and these comments will not be repeated here. Only a few special features of this disassembler will be noted.

In Screen 2, the defining world FIELD is used to generate all the references to different fields in the 8086 instruction byte and the associated memory addressing byte. To define a field, a field mask and a LSB bit position are needed. When a field word is executed, it pulls the 8086 code out of the DECODER area, shifts it down till the defined field is at the least significant end, and masks off all the bits out side of the field. The field word thus will return the contents of a field.

The word NAME is another defining word which is used to define words which compile strings packed with register names of two ASCII characters. When a word defined by NAME is executed with a string number on the stack, a two character string is unpacked from the superstring and printed on the console. They are used to define words which print the names of 16 bit registers, 8 bit registers, and segment registers.

There are two defining words in Screen 5. CASE: defines a case structure. It is very similar to a colon definition, in that it compiles a sequence of Forth words terminated by a semi-colon. When a case word thus define is executed, it uses the top item on the stack to select one of the compiled Forth words in its parameter field and executes it. This case structure is used very heavily in this disassembler, to direct the disassembler to search through many levels of multiple choice branches to find the right way to display the mnemonics and argument list of an 8086 instruction.

MESS simply associates a print string with a name. Its function can be accomplished by a ." xxxx" string in a colon definition. However, MESS is much cleaner and efficient for lots of short messages.

Another interesting defining word is in Screen 8. SELF defines a word which prints its own name. This is very handy in defining 8086 instructions which do not have arguments. Words defined by SELF have three character names, very common among 8086 mnemonics. For 8086 instructions with four character names, 4SELF in Screen 9 is the proper defining word.

NAMES in Screen 17 is very similar to NAME. The only difference is that words defined by NAMES print strings of four characters, allowing the extended instruction set to be printed conveniently.

In Screen 23, the giant case word <DECODE> is a case structure with 32 branches. It selects one of the 32 nTYPE word for execution, serving as the primary decoder of 8086 instructions. Each nTYPE word is a smaller case structure with 8 branches, which then determine how the instruction byte has to be treated. After that, the disassembler picks up the rest of the instruction and prints out the argument list, if any.

4. CONCLUSION

Disassembler is not a very difficult job. However, Intel did not make this job any easier, with the convoluted 8086 instruction set, which tried to shoe-horne a 16 bit processor into the mold of an 8 bit machine.

The most significant feature of this disassembler is the use of a two level case structure to decode the 8086 instruction byte. The position case structure is very convenient and efficient for this application.

8086 INSTRUCTION SET MATRIX

Hi \ Lo	0	1	2	3	4	5	6	7
0	ADD b,f,r/m	ADD w,f,r/m	ADD b,f,r/m	ADD w,f,r/m	ADD b,ia	ADD w,ia	PUSH ES	POP ES
1	ADC b,f,r/m	ADC w,f,r/m	ADC b,f,r/m	ADC w,f,r/m	ADC b,i	ADC w,i	PUSH SS	POP SS
2	AND b,f,r/m	AND w,f,r/m	AND b,f,r/m	AND w,f,r/m	AND b,i	AND w,i	SEG ES	DAA
3	XOR b,f,r/m	XOR w,f,r/m	XOR b,f,r/m	XOR w,f,r/m	XOR b,i	XOR w,i	SEG SS	AAA
4	INC AX	INC CX	INC DX	INC BX	INC SP	INC BP	INC SI	INC DI
5	PUSH AX	PUSH CX	PUSH DX	PUSH BX	PUSH SP	PUSH BP	PUSH SI	PUSH DI
6								
7	JO	JNO	JB/ JNAE	JNB/ JAE	JE/ JZ	JNE/ JNZ	JBE/ JNA	JNBE/ JA
8	Immed b,r/m	Immed w,r/m	Immed b,r/m	Immed is,r/m	TEST b,r/m	TEST w,r/m	XCHG b,r/m	XCHG w,r/m
9	NOP	XCHG CX	XCHG DX	XCHG BX	XCHG SP	XCHG BP	XCHG SI	XCHG DI
A	MOV m → AL	MOV m → AX	MOV AL → m	MOV AX → m	MOVS b	MOVS w	CMPS b	CMPS w
B	MOV r → AL	MOV r → CL	MOV r → DL	MOV r → BL	MOV r → AH	MOV r → CH	MOV r → DH	MOV r → BH
C			RET (i+SP)	RET	LES	LDS	MOV b,r/m	MOV w,r/m
D	Shift b	Shift w	Shift b,v	Shift w,v	AAM	AAD		XLAT
E	LOOPNZ/ LOOPNE	LOOPZ LOOPE	LOOP	JCXZ	IN b	IN w	OUT b	OUT w
F	LOCK		REP	REP z	HLT	CMC	Grp 1 b,r/m	Grp 1 w,r/m

Hi \ Lo	8	9	A	B	C	D	E	F
0	OR b,f,r/m	OR w,f,r/m	OR b,f,r/m	OR w,f,r/m	OR b,i	OR w,i	PUSH CS	
1	SBB b,f,r/m	SBB w,f,r/m	SBB b,f,r/m	SBB w,f,r/m	SBB b,i	SBB w,i	PUSH DS	POP DS
2	SUB b,f,r/m	SUB w,f,r/m	SUB b,f,r/m	SUB w,f,r/m	SUB b,i	SUB w,i	SEG CS	DAS
3	CMP b,f,r/m	CMP w,f,r/m	CMP b,f,r/m	CMP w,f,r/m	CMP b,i	CMP w,i	SEG DS	AAS
4	DEC AX	DEC CX	DEC DX	DEC BX	DEC SP	DEC BP	DEC SI	DEC DI
5	POP AX	POP CX	POP DX	POP BX	POP SP	POP BP	POP SI	POP DI
6								
7	JS	JNS	JP/ JPE	JNP/ JPO	JL/ JNGE	JNL/ JGE	JLE/ JNG	JNLE/ JG
8	MOV b,f,r/m	MOV w,f,r/m	MOV b,f,r/m	MOV w,f,r/m	MOV sr,f,r/m	LEA	MOV sr,f,r/m	POP r/m
9	CBW	CWD	CALL l,d	WAIT	PUSHF	POPF	SAHF	LAHF
A	TEST b,i	TEST w,i	STOS b	STOS w	LODS b	LODS w	SCAS b	SCAS w
B	MOV r → AX	MOV r → CX	MOV r → DX	MOV r → BX	MOV r → SP	MOV r → BP	MOV r → SI	MOV r → DI
C			RET i,(i+SP)	RET i	INT Type 3	INT Any	INTO	IRET
D	ESC 0	ESC 1	ESC 2	ESC 3	ESC 4	ESC 5	ESC 6	ESC 7
E	CALL d	JMP d	JMP l,d	JMP sl,d	IN v,b	IN v,w	OUT v,b	OUT v,w
F	CLC	STC	CLI	STI	CLD	STD	Grp 2 b,r/m	Grp 2 w,r/m

b = byte operation
d = direct
f = from CPU reg
i = immediate
ia = immed. to accum.
id = indirect
is = immed. byte, sign ext.
l = long (ie. intersegment

m = memory
r/m = EA is second byte
sl = short intrasegment
sr = segment register
t = to CPU reg
v = variable
w = word operation
z = zero

where

mod ☐ r/m	000	001	010	011	100	101	110	111
Immed	ADD	OR	ADC	SBB	AND	SUB	XOR	CMP
Shift	ROL	ROR	RCL	RCR	SHL SAL	SHR	—	SAR
Grp 1	TEST	—	NOT	NEG	MUL	IMUL	DIV	IDIV
Grp 2	INC	DEC	CALL id	CALL l,id	JMP id	JMP l,id	PUSH	—

Figure 4. 8086 instruction set matrix

```
7000    E0      LOOPNZ    7035
7002    C9      ILL
7003    EC      IN      AL,DX
7004    A       OR      AL,AL
7006    78      JS        7015
7008    E2      LOOP      7003
700A    FE      DEC       BH
700C    75      JNE       7001
700E    C       OR      AL,1
7010    24      AND       AL,F9
7012    EB      JMP       7025
7014    90      XCHG      AX
7015    42      INC       DX
7016    B0      MOV       AL,D
7018    EE      OUT       AL,DX
7019    B0      MOV       AL,C
701B    EE      OUT       AL,DX
701C    4A      DEC       DX
701D    EB      JMP       7022
701F    8A      MOV       AH,AL
7021    42      INC       DX
7022    EC      IN      AL,DX
7023    24      AND       AL,F8
7025    34      XOR       AL,48
7027    86      XCHG      AL,AH
7029    EB      JMP       6FF4
702B    8A      MOV       AH,AL
702D    42      INC       DX
702E    42      INC       DX
702F    B0      MOV       AL,8
7031    EE      OUT       AL,DX
7032    B9      MOV       CX,5DC
7035    E2      LOOP      7035
7037    EB      JMP       7019
7039    FF      ILL
703A    FF      ILL
703B    FF      ILL
703C    FF      ILL
703D    FF      ILL
703E    FF      ILL
703F    FF      ILL

7065    FB      STI
7066    FC      CLD
7067    55      PUSH      BP
7068    6       PUSH      ES
7069    1E      PUSH      DS
706A    56      PUSH      SI
706B    57      PUSH      DI
706C    52      PUSH      DX
706D    51      PUSH      CX
706E    53      PUSH      BX
706F    50      PUSH      AX
7070    BB      MOV       BX,40
7073    8E      MOV       DS,BX
7075    8A      MOV       BL,[10 ]
7079    80      AND       BL,30
707C    80      CMP       BL,30
707F    BB      MOV       BX,-4800
7082    75      JNE       7087
7084    BB      MOV       BX,-5000
7087    53      PUSH      BX
7088    8B      MOV       BP,SP
708A    E8      CALL      7104
708D    5E      POP       SI
708E    58      POP       AX
708F    5B      POP       BX
7090    59      POP       CX
7091    5A      POP       DX
7092    5F      POP       DI
7093    5E      POP       SI
7094    1F      POP       DS
7095    7       POP     ES
7096    5D      POP       BP
7097    CF      IRET
7098    FF      ILL
7099    FF      ILL
709A    FF      ILL
709B    FF      ILL
709C    FF      ILL
709D    FF      ILL
709E    FF      ILL
709F    FF      ILL
70A0    FF      ILL
70A1    FF      ILL
70A2    FF      ILL
70A3    FF      ILL

70A4    38      CMP       [BX+SI],CH
70A6    2D      SUB       AX,1F0A
70A9    6       PUSH      ES
70AA    19      SBB       [SI],BX
70AC    2       ADD     AL,[BX]
70AE    6       PUSH      ES
70AF    7       POP     ES
70B0    0       ADD       [BX+SI],AL
70B2    0       ADD       [BX+SI],AL
70B4    71      JNO       7106
70B6    5A      POP       DX
70B7    A       OR      BL,[BX]
70B9    6       PUSH      ES
70BA    19      SBB       [SI],BX
70BC    2       ADD     AL,[BX]
70BE    6       PUSH      ES
70BF    7       POP     ES
70C0    0       ADD       [BX+SI],AL
70C2    0       ADD       [BX+SI],AL
70C4    38      CMP       [BX+SI],CH
70C6    2D      SUB       AX,7F0A
70C9    6       PUSH      ES
70CA    64      ILL
70CB    70      JO        70CF
70CD    1       ADD     [7 ],AX
70D1    0       ADD       [BX+SI],AL
70D3    0       ADD     [BX+DI]50 ,AH
70D6    52      PUSH      DX
70D7    F       POP     CS
70D8    19      SBB     [1919 ],AX
70DC    2       ADD     CL,[DI]
70DE    B       OR      CX,[SI]
70E0    0       ADD       [BX+SI],AL
70E2    0       ADD       [BX+SI],AL
70E4    0       ADD       [BX+SI],CL
70E6    0       ADD       [BX+SI],DL
70E8    0       ADD     [BX+SI]0 ,AL
70EB    40      INC       AX
70EC    28      SUB       [BX+SI],CH
70EE    50      PUSH      AX
70EF    50      PUSH      AX
70F0    28      SUB       [BX+SI],CH
70F2    50      PUSH      AX
70F3    50      PUSH      AX
70F4    2C      SUB     AL,28
70F6    2D      SUB     AX,2A29
70F9    2E      SEG     CS
70FA    1E      PUSH      DS
70FB    29      SUB       [BX+SI],AX
70FD    0       ADD     [BX+SI],DL
70FF    10      ADC       [BX+SI],AH
```

Figure 5. Disassembled 8086 machine code

```
      0
 0 8086 Disassembler
 1 C. H. Ting
 2
 3 Based on polyForth II from Forth Inc.
 4
 5 Command syntax:
 6
 7    address #bytes  DECODE
 8
 9 The instructions from address will be displayed in Intel
10 mnemonics till #bytes are disassembled.
11
12
13
14
15
```

```
      1
 0 ( 8086 DISASSEMBLER, CHT, 9-FEB-85)
 1 2 23 THRU
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
      2
 0 (   INSTRUCTION DECODE, CHT, 9-FEB-85)
 1 VARIABLE PC
 2 ^ VARIABLE DECODER   VARIABLE PC
 3 : @PC   PC @ @   DECODER !   ;
 4 : FIELD   CREATE , ( MASK)   , ( BIT LOCATION )
 5    DOES>   2@ DECODER @ SWAP     ( CODE MASK BIT --- )
 6    ?DUP IF   0 DO 2/ LOOP THEN   AND ;
 7 1 0 FIELD W2   1 1 FIELD DIR    7 0 FIELD R1   1 3 FIELD W1
 8 15 4 FIELD MAIN   7 8 FIELD R/M   7 11 FIELD R2
 9 3 3 FIELD S1   3 11 FIELD S2   3 14 FIELD MODE
10 : BYTES ( N --- )   PC +!   ;
11 : BD   PC @ C@   1 BYTES   ;
12 : WD   PC @ @   2 BYTES   ;
13 : LWD   PC @ 2@   4 BYTES   ;
14 : [A]   ." [" WD 0 D. ." ]"   ;
15
```

```
      93
03oct87cht 8086 Disassembler                                    03oct87cht

References:
  MSC-80 Assembly Language Reference Manual, Intel Corp. 1978
  The 8086 Book, Russel Rector and George Alexy,
    Osborne/McGraw Hill, Berkeley
```

```
      94
03oct87cht ( 8086 DISASSEMBLER, CHT, 9-FEB-85)            02:38
PolyForth II is fairly close to Forth-79 Standard, and not much
different from Forth-83.  Most of the words used in this
program behave identically across these dialects.  Some special
pol.Forth words to be watched are:

        Def__ nex_ wor_ wit_ ful_ nam_, not jus_ usi__ the
        fir__ thr__ cha_____ and a cou__.  For__, Inc. is
        ver_ rel_____ in aba_____ thi_ fea____.
```

```
      95
(   INSTRUCTION DECODE, CHT, 9-FEB-85)          00:21
PC        Program counter pointing to instruction to be decoded
DECODER   Current instruction being decoded.
@PC       Get an instruction and store in DECODER.
FIELD     To define various fields in an 8086 instruction.
          It takes two parameters: field width and LSB position.
MAIN      Instruction type.
MODE, R2, R/M   Fields in addressing mode byte.
S1, S2, R1, W1, W2   Fields in Instruction byte.

BYTES     Add offset to PC to get next byte in instruction.
BD        Get one byte and move PC pointer.
WD        Get one word and move PC pointer.
LWD       Get a long word and move PC.
[A]       Print an address in square brackets.
```

Listing 6. 8086 disassembler

```
      3
 0 (   REGISTER NAMES, CHT, 9-FEB-85)
 1 : NAME   CREATE 34 WORD   C@ 3 + ALLOT
 2   DOES>  SWAP 2* + 3 +   2 TYPE   ;
 3 NAME REG16 AXCXDXBXSPBPSIDI"
 4 NAME REG8 ALCLDLBLAHCHDHBH"
 5 NAME .SEG ESCSSSDS"
 6 NAME .REG XSXDPSPDSIDIBPBX"
 7 NAME .JUMP O NOB NBE NENAA S NSP NPL NLNGG "
 8 : TAB  5 SPACES   ;
 9 : COMMA  44 EMIT  ;
10 : EXTEND   ( B --- W.   DUP 128 AND IF 256 - THEN   ;
11 : DISP   BD EXTEND PC @   + .   ;
12 : DISP16   WD PC @ + 0 D.   ;
13 : JCC   .' J"   DECODER C@ 15 AND .JUMP   TAB   DISP ;
14
15
```

```
      4
 0 (   REGISTER INSTRUCTIONS, CHT, 9-FEB-85)
 1 : INC   ." INC"   TAB   R1 REG16   ;
 2 : DEC   ." DEC"   TAB   R1 REG16   ;
 3 : PUSH  ." PUSH"  TAB   R1 REG16   ;
 4 : POP   ." POP"   TAB   R1 REG16   ;
 5 : XCHG  ." XCHG"  TAB   R1 REG16   ;
 6 ^ : MVIB   ." MOV"   TAB   R1 REG8   COMMA   BD .   ;
 7 ^ : MVIW   ." MOV"   TAB   R1 REG16  COMMA   WD .   ;
 8 : 8TYPE   INC   ;                : 9TYPE   DEC   ;
 9 : 10TYPE  PUSH  ;                : 11TYPE  POP   ;
10 : 18TYPE  XCHG  ;
11 : 22TYPE  MVIB  ;                : 23TYPE  MVIW  ;
12 : 14TYPE  JCC   ;                : 15TYPE  JCC   ;
13
14
15
```

```
      5
 0 (   REGISTER MODES, CHT, 9-FEB-85)
 1 : CASE:   CREATE ] SMUDGE
 2   DOES>   SWAP 2* + @ 2+   EXECUTE   ;
 3 : MESS   CREATE 34 WORD C@ 3 + ALLOT
 4   DOES>   2+ COUNT TYPE   ;
 5 MESS 0RM [BX+SI]"   MESS 1RM [BX+DI]"   MESS 2RM [BP+SI]"
 6 MESS 3RM [BP+DI]"   MESS 4RM [SI]"   MESS 5RM [DI]"
 7 MESS 6RM [BP]"   MESS 7RM [BX]"
 8 CASE: .R/M   0RM 1RM 2RM 3RM 4RM 5RM 6RM 7RM   ;
 9
10
11
12
13
14
15
```

```
     96
(   REGISTER NAMES, CHT, 9-FEB-85)            00:28
NAME    Define register names as a super-string.
        Print a register name from the string at run time.
REG16   Names of 16 bit registers.
REG8    Names of 8 bit registers.
.SEG    Names of segment registers.
.REG    Names of special regsiter.
.JUMP   Names of jump instructions.
TAB     5 SPACES
COMMA   Print a comma.
EXTEND  Sign extension of a byte.
DISP    Print a bytes displacement from PC.
DISP16  Print a 16 bit displacement from PC.
JCC     Print a conditional jump instruction with displacement.
```

```
     97
(   REGISTER INSTRUCTIONS, CHT, 9-FEB-85)      00:35
8TYPE   INC   Increment
9TYPE   DEC   Decrement
10TYPE  PUSH  Push register
11TYPE  POP   Pop register
14TYPE  JCC   Conditional jump
15TYPE  JCC   Conditional jump
18TYPE  XCHG  Exchange registers or memory
22TYPE  MVIB  Move byte immediate
23TYPE  MVIW  Move word immediate
```

```
     98
(   REGISTER MODES, CHT, 9-FEB-85)            00:43
CASE:     My favorite case structure.
          Positional case statement.
MESS      Create string messages and give them names.

0RM       [BX+SI]
1RM       [BX+DI]
2RM       [BP+SI]
3RM       [BP+DI]
4RM       [SI]
5RM       [DI]
6RM       [BP]
7RM       [BX]
.R/M      Print register deferred mode specification.
```

Listing 6. 8086 disassembler (cont'd)

```
      5
 0 (   MODE BYTE, CHT, 9-FEB-85)
 1 : 0MOD   R/M 6 = IF   [A]   ELSE R/M .R/M   THEN ;
 2 : 1MOD   R/M .R/M   BD EXTEND .   ;
 3 : 2MOD   R/M .R/M   WD .   ;
 4 : 3MOD   W2 IF   R/M REG16   ELSE R/M REG8   THEN   ;
 5 CASE: <.RM>   0MOD 1MOD 2MOD 3MOD   ;
 6 : .RM   1 BYTES   MODE <.RM>   ;
 7 : .R2   W2 IF   R2 REG16   ELSE R2 REG8   THEN   ;
 8 : REGS   DIR IF   .R2 COMMA   .RM
 9   ELSE .RM   COMMA .R2   THEN   ;
10 : BI   ." AL,"   BD .   ;
11 : WI   ." AX,"   WD .   ;
12
13
14
15
```

```
      99
(   MODE BYTE, CHT, 9-FEB-85)                    00:52
0MOD      Memory addressing mode.
1MOD      Memory addressing mode, one displacement byte.
2MOD      Memory addressing mode, two displacement bytes.
3MOD      Register addressing mode.
<.RM>     Select one of above.
.RM       Print the address argument.
.R2       Print 16 bit or 8 bit register name.
REGS      Print the source and destination register/memory.

BI        8 bit accumulator.
WI        16 bit accumulator.
```

```
      7
 0 (   MATH INSTRUCTIONS, CHT, 9-FEB-85)
 1 CASE: <OPERAND>   REGS REGS REGS REGS BI WI   ;
 2 : ADD   ." ADD" TAB   R1 <OPERAND>   ;
 3 : .OR   ." OR" TAB   R1 <OPERAND>   ;
 4 : ADC   ." ADC" TAB   R1 <OPERAND>   ;
 5 : SBB   ." SBB" TAB   R1 <OPERAND>   ;
 6 : .AND   ." AND" TAB   R1 <OPERAND>   ;
 7 : SUB   ." SUB" TAB   R1 <OPERAND>   ;
 8 : .XOR   ." XOR" TAB   R1 <OPERAND>   ;
 9 : CMP   ." CMP" TAB   R1 <OPERAND>   ;
10 : PUSHSEG   ." PUSH" TAB   S1 .SEG   ;
11 : POPSEG   ." POP" TAB   S1 .SEG   ;
12 : SEG   ." SEG" TAB   S1 .SEG   ;
13
14
15
```

```
      100
(   MATH INSTRUCTIONS, CHT, 9-FEB-85)          00:58
<OPERAND>   Select argument options.
ADD
.OR
ADC
SBB
.AND
SUB
.XOR
CMP
PUSHSEG
POPSEG
SEG
```

```
      8
 0 (   SINGLE INSTRUCTIONS, CHT, 9-FEB-85)
 1 : SELF   CREATE
 2   DOES>   5 - 3 TYPE   ;
 3 SELF DAA   SELF AAA   SELF DAS   SELF AAS   SELF NOP
 4 SELF REP   SELF HLT
 5 SELF CMC   SELF CBW   SELF CWD   SELF CLC   SELF STC
 6 SELF CLI   SELF STI   SELF CLD   SELF STD
 7 SELF ILL
 8
 9
10
11
12
13
14
15
```

```
      101
(   SINGLE INSTRUCTIONS, CHT, 9-FEB-85)        01:02
SELF       Define instructions which will print its own names.
           Define 3 character name instructions without
           arguments.
DAA   AAA   DAS   AAS   NOP
REP   HLT
CMC   CBW   CWD   CLC   STC
CLI   STI   CLD   STD
ILL        Illegal instruction. Very important in disassembler.
```

Listing 6. 8086 disassembler (cont'd)

```
 9
 0 (   4 CHARACTER INSTRUCTIONS, CHT, 9-FEB-85)
 1 : 4SELF   ^ CREATE
 2   DOES>  6 - 4 TYPE   ;
 3 4SELF LOCK   4SELF REPZ   4SELF XLAT
 4 4SELF WAIT   4SELF PSHF   4SELF POPF   4SELF SAHF   4SELF LAHF
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
 102
(   4 CHARACTER INSTRUCTIONS, CHT, 9-FEB-85)    01:05
4SELF       Define 4 character name instructions which do not
            require arguments.
            Instruction so defined prints its own name at run
            time.
LOCK   REPZ   XLAT
WAIT   PSHF   POPF
SAHF   LAHF
```

```
 10
 0 (    8 TYPES, CHT, 9-FEB-85)
 1 : ?SEG   R1 6 =   IF PUSHSEG   ELSE POPSEG THEN   ;
 2 : 0TYPE   R1 6 <   IF ADD   ELSE ?SEG   THEN ;
 3 : 1TYPE   R1 6 <   IF .OR   ELSE ?SEG   THEN ;
 4 : 2TYPE   R1 6 <   IF ADC   ELSE ?SEG   THEN ;
 5 : 3TYPE   R1 6 <   IF SBB   ELSE ?SEG   THEN ;
 6 : 4TYPE   R1 6 <   IF .AND   ELSE   R1 6 =
 7   IF SEG   ELSE DAA   THEN THEN   ;
 8 : 5TYPE   R1 6 <   IF SUB   ELSE   R1 6 =
 9   IF SEG   ELSE DAS   THEN THEN   ;
10 : 6TYPE   R1 6 <   IF .XOR   ELSE   R1 6 =
11   IF SEG   ELSE AAA   THEN THEN   ;
12 : 7TYPE   R1 6 <   IF CMP   ELSE   R1 6 =
13   IF SEG   ELSE AAS   THEN THEN   ;
14
15
```

```
 103
(    8 TYPES, CHT, 9-FEB-85)                    01:12
?SEG      Print PUSH/POP segment register instructions.
0TYPE     ADD
1TYPE     .OR
2TYPE     ADC
3TYPE     SBB
4TYPE     .AND DAA

5TYPE     SUB  DAS

6TYPE     .XOR AAA

7TYPE     CMP  AAS
```

```
 11
 0 (   TYPE 17, CHT, 10-FEB-85)
 1 : SR-RM   DIR IF   S2 .SEG COMMA R/M REG16
 2   ELSE   R/M REG16   COMMA S2 .SEG   THEN   1 BYTES :
 3 : PRM   ." POP" TAB   .RM   ;
 4 : LEA   ." LEA" TAB   .RM   ;
 5 : LES   ." LES" TAB   .RM   ;
 6 : LDS   ." LDS" TAB   .RM   ;
 7 CASE: (17)   REGS REGS REGS REGS SR-RM ILL SR-RM ILL   ;
 8 : 17TYPE   R1 5 = IF LEA   ELSE   R1 7 = IF PRM
 9   ELSE ." MOV" TAB   R1 (17)   THEN THEN   ;
10
11
12
13
14
15
```

```
 104
(   TYPE 17, CHT, 10-FEB-85)                    01:38
SR-RM     Print source and destination register/memory for
          type 17 instructions..SEG   THEN   1 BYTES ;
PRM       POP
LEA       LEA
LES       LES
LDS       LDS
(17)      Select options for type 17 instructions.  :
17TYPE    Type 17 instruction processing.
```

Listing 6. 8086 disassembler (cont'd)

```
     12
 0 (   TYPE 19, CHT, 10-FEB-85)
 1 : .LD   WD 0   W2 16 M#    D+ D.   ;
 2 : CALLLD   ." CALL" TAB   .LD   ;
 3 CASE: <19>   CBW CWD CALLLD WAIT PSHF POPF SAHF LAHF   ;
 4 : 19TYPE   R1 <19>   ;
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
     13
 0 (   TYPE 21, CHT, 10-FEB-85)
 1 : B/W   W2 IF   ." AX"   ELSE   ." AL"   THEN   ;
 2 : BYWD   W2 IF ." WORD"   ELSE ." BYTE"   THEN ;
 3 : .TESTI   ." TEST" TAB - W2 IF WI   ELSE BI THEN ;
 4 : STOS   ." STOS" TAB   BYWD   ;
 5 : LODS   ." LODS" TAB   BYWD   ;
 6 : SCAS   ." SCAS" TAB   BYWD   ;
 7 CASE: <21>   .TESTI .TESTI STOS STOS LODS LODS SCAS SCAS ;
 8 : 21TYPE   R1 <21>   ;
 9
10
11
12
13
14
15
```

```
     14
 0 (   TYPES 24-25, CHT, 10-FEB-85)
 1 4SELF IRET   SELF RET   4SELF INTO   4SELF RETL
 2 : INT   ." INT" TAB   3 .   ;
 3 : RETLI   ." RETL"   TAB WD .   ;
 4 : RETI   ." RET"   TAB WD .   ;
 5 : INTI   ." INT"   TAB BD .   ;
 6 CASE: <25>   ILL ILL RETLI RETL INT INTI INTO IRET   ;
 7 : 25TYPE   R1 <25>   ;
 8 : MIM   ." MOV" TAB   .RM   COMMA
 9    W2 IF   WD   ELSE   BD   THEN .   ;
10 CASE: <24>   ILL ILL RETI RET LES LDS MIM MIM   ;
11 : 24TYPE   R1 <24>   ;
12
13
14
15
```

```
  105
(   TYPE 19, CHT, 10-FEB-85)                    01:4(
.LD       Print 32 bit long address.
CALLLD    Process long call instruction.
<19>      Select type 19 options.
19TYPE    Process type 19 instruction.
```

```
  106
(   TYPE 21, CHT, 10-FEB-85)                    01:4E
B/W       Print AX or AL depending on the word size bit.
BYWD      Print WORD or BYTE.
.TESTI    TEST
STOS      STOS
LODS      LODS
SCAS      SCAS
<21>      Select type 21 options.
21TYPE    Type 21 instruction processing.
```

```
  107
(   TYPES 24-25, CHT, 10-FEB-85)                01:5E
IRET   RET   INTO   RETL    Simple instructions.
INT       Interrpt type 3.
RETLI     Return and add immediate to SP.
RETI      Intersegment return.
INTI      Regular interrpt.
<25>      Select interrpts and returns.
25TYPE    Type 25 instruction processing.
MIM       Move immediate to memory/register.

<24>      Select type 24 instruction.
24TYPE    Process type 24 instructions.
```

Listing 6. 8086 disassembler (cont'd)

```
     15
 0 (   TYPE 28, CNT, 10-FEB-85)
 1 : .LOOP   ." LOOP" TAB   DISP   ;
 2 : LOOPZ   ." LOOPZ" TAB   DISP   ;
 3 : LOOPNZ   ." LOOPNZ" TAB   DISP   ;
 4 : JCXZ   ." JCXZ" TAB   DISP   ;
 5 : PORT   B/W   COMMA   BD .   ;
 6 : IN   ." IN" TAB   PORT   ;
 7 : OUT   ." OUT" TAB   PORT   ;
 8 CASE: <28>   LOOPNZ LOOPZ .LOOP JCXZ IN IN OUT OUT   ;
 9 : 28TYPE   R1 <28   ;
10
11
12
13
14
15
```

```
     16
 0 (   TYPE 29, CNT, 10-FEB-85)
 1 : CALLD   ." CALL" TAB   DISP16   ;
 2 : JMPI   ." JMP" TAB   DISP16   ;
 3 : JMPLI   ." JMP" TAB   .LI   ;
 4 : JMPSID   ." JMP" TAB   DISP   ;
 5 : VDX   B/W   COMMA ." DX"   ;
 6 : INV   ." IN" TAB   VDX   ;
 7 : OUTV   ." OUT" TAB   VDX   ;
 8 CASE: <29>   CALLD JMPI JMPLI JMPSID INV IN OUTV OUTV   ;
 9 : 29TYPE   R1 <29   ;
10
11
12
13
14
15
```

```
     17
 0 (    EXTENDED INSTRUCTION NAMES, CNT, 10-FEB-85)
 1 : NAMES   CREATE 34 WORD   C@ 1 + ALLOT
 2   DOES>   SWAP 4 * + 3 +   4 TYPE   TAB   ;
 3 NAMES IMMED ADD OR  ADC ABB AND SUB XOR CMP "
 4 NAMES SHIFT ROL ROR RCL RCR SHL SHR ILL SAR "
 5 NAMES .16RP TESTILL NOT NEG MUL IMULDIV IDIV"
 6 NAMES .26RP INC DEC CALLCALLJMP JMP PUSHILL "
 7
 8
 9
10
11
12
13
14
15
```

```
     108
(   TYPE 28, CNT, 10-FEB-85)                    02:03
.LOOP
LOOPZ
LOOPNZ
JCXZ
PORT        Print port number.
IN
OUT
<28>        Select type 28 instruction.UT OUT   ;
28TYPE      Process type 28 instructions.
```

```
     109
(   TYPE 29, CNT, 10-FEB-85)                    02:04
CALLD       Call with 16 bit address.
JMPI        Jump with 16 bit address.
JMPLI       Jump with 32 bit address.
JMPSID      Jump with 16 bit displacement
VDX         I/O through DX register.
INV         Input with port address in DX.
OUTV        Output with port address in DX.
<29         Select type 29 instruction.
29TYPE      Process type 29 instructions.
```

```
     110
(   EXTENDED INSTRUCTION NAMES, CNT, 10-FEB-85)02:07
NAMES       Define 8086 extended instruction sets.

IMMED       Immediate ALU instructions.
SHIFT       Extended shift instructions.
.16RP       Extended multiply/divide instructions.
.26RP       Extended call and jump[ instructions.
```

Listing 6. 8086 disassembler (cont'd)

```
     18
 0 (   TYPE 16, CHT, 10-FEB-85)
 1 : .IMMED   R2 IMMED   .RM COMMA   W2 IF
 2   DIR IF   BD EXTEND  ELSE   WD THEN   ELSE   BD THEN   . ;
 3 : TESTIR   ." TEST" TAB   .RM COMMA
 4   W2 IF WI   ELSE BI THEN   ;
 5 : XCHGR   ." XCHG" TAB   REGS   ;
 6 CASE: <16>   .IMMED .IMMED .IMMED .IMMED TESTIR TESTIR
 7   XCHGR XCHGF   ;
 8 : 16TYPE   R1 <16>   ;
 9
10
11
12
13
14
15
```

```
  111
(   TYPE 16, CHT, 10-FEB-85)                    02:10
.IMMED      Print immediate instructions.

TESTIR      Print test instructions.

XCHGR       Exchange instructions.
<16>        Select type 16 instructions.

16TYPE      Process type 16 instructions.
```

```
     19
 0 (   TYPE 26, CHT, 10-FEB-85)
 1 : .SHIFT   R2 SHIFT   .RM   DIR IF COMMA ." CL" THEN   ;
 2 : AAM   ." AAM"   1 BYTES   ;
 3 : AAD   ." AAD"   1 BYTES   ;
 4 CASE: <26>   .SHIFT .SHIFT .SHIFT .SHIFT AAM AAD ILL XLAT   ;
 5 : 26TYPE   R1 <26>   ;
 6
 7
 8
 9
10
11
12
13
14
15
```

```
  112
(   TYPE 26, CHT, 10-FEB-85)                    02:12
.SHIFT      Extended shift instructions.." CL" THEN   ;
AAM         Ascii adjust for multiply.
AAD         Ascii adjust for divide.
<26         Select type 26 instruction. AAD ILL XLAT   ;
26TYPE      Process type 26 instruction.
```

```
     20
 0 (   TYPE 30, CHT, 11-FEB-85)
 1 : 1GRP   R2 .1GRP   .RM   ;
 2 CASE: <30>   LOCK ILL REP REPZ HLT CMC 1GRP 1GRP   ;
 3 : 30TYPE   R1 <30>   ;
 4
 5
 6
 7
 8
 9
10
11
12
13
14
```

```
  113
(   TYPE 30, CHT, 11-FEB-85)                    02:13
1GRP        Group 1 extended instructions.
<30>        Select type 30 instruction.
30TYPE      Process type 30 instructions.
```

Listing 6. 8086 disassembler (cont'd)

```
      21
 0 (   TYPE 31, CMT, 11-FEB-85)
 1 : CALLID   ." CALL" TAB   .RM   ;
 2 : CALLLID  ." CALL" TAB   .LD   ;
 3 : JMPID    ." JMP" TAB   .RM   ;
 4 : JMPLID   ." JMP" TAB   .LD   ;
 5 : PUSHRM   ." PUSH" TAB   .RM   ;
 6 : INCRM    ." INC" TAB   .RM   ;
 7 : DECRM    ." DEC" TAB   .RM   ;
 8 CASE: <26RP>   INCRM DECRM CALLID CALLLID JMPID JMPLID PUSHRM
 9    ILL   ;
10 : 26RP   R2 <26RP>   ;
11 CASE: <31>   CLC STC CLI STI CLD STD 26RP 26RP   ;
12 : 31TYPE   R1 <31>   ;
13
14
15
```

```
      22
 0 (   TYPE 20  CMT, 11-FEB-85)
 1 : LDA   ." MOV" TAB   B/W   COMMA   [A]   ;
 2 : STA   ." MOV" TAB   [A] COMMA   B/W   ;
 3 ^ : MOVS   ." MOVS" TAB   B/W/D   ;
 4 ^ : CMPS   ." CMPS" TAB   B/W/Z   ;
 5 CASE: <20>   LDA LDA STA STA MOVS MOVS CMPS CMPS   ;
 6 : 20TYPE   R1 <20>   ;
 7 : 27TYPE   ." ESC" TAB   .RM   ;
 8
 9
10
11
12
13
14
15
```

```
      23
 0 (   BIG CASE, CMT, 11-FEB-85)                    02:30
 1 CASE: <DECODE>
 2    0TYPE   1TYPE   2TYPE   3TYPE   4TYPE   5TYPE   6TYPE
 3    7TYPE   8TYPE   9TYPE  10TYPE  11TYPE   ILL     ILL
 4   14TYPE  15TYPE  16TYPE  17TYPE  18TYPE  19TYPE  20TYPE
 5   21TYPE  22TYPE  23TYPE  24TYPE  25TYPE  26TYPE  27TYPE
 6   28TYPE  29TYPE  30TYPE  31TYPE   ;
 7 : (DECODE)   DR PC ?   @PC   TAB DECODER C@ .   TAB
 8    1 BYTES   DECODER C@ 5 / <DECODE>   ;
 9 : DECODE   ( ADDR # --- )
10    OVER PC !   +   BEGIN   (DECODE)   PC @ OVER = UNTIL   ;
11 : F0 <DECODE> TAB PC ?   ;
12
13
14
15
```

```
      114
(   TYPE 31, CMT, 11-FEB-85)                    02:19
CALLID     Indirect call within segment.
CALLLID    Intersegment indirect call.
JMPID      Indirect jump within segment.
JMPLID     Intersegment indirect jump.
PUSHRM     Push register/memory.
INCRM      Increment register/memory.
DECRM      Decrement register/memory.
<26RP>     Select group 2 extended instruction.

26RP       Process group 2 extended instructions.
<31>       Select type 31 instruction.RF   ;
31TYPE     Process type 31 instructions.
```

```
      115
(   TYPE 20, CMT, 11-FEB-85)                    02:22
LDA        Load accumulator.
STA        Store accumulator.
MOVS       Move string.
CMPS       Compare strings.
<20>       Select type 20 instruction.
20TYPE     Process type 20 instructions.
27TYPE     Process type 27 instructions.
```

```
      116
(   BIG CASE, CMT, 11-FEB-85)                    02:30
<DECODE>   This is the big case structure which does the
           decoding of an 8086 instruction using the most
           significant 5 bits.
           Given a number between 0 and 31 inclusive, it
           will decode this instruction by executing the
           corresponding nTYPE word.
(DECODE)   Grab a byte pointed to by PC into DECODER, decode
           this instruction and move PC passing the arguments.
DECODE     The final, user friendly application word.  Given
           an address and a byte count, DECODE will display
           all the instructions in this memory range.
F0         Single step.  Decode one instruction at a time.
           PolyForth automatically assigns this function to
           the function key  F10 .
```

Listing 6. 8086 disassembler (cont'd)

V. 68000 DISASSEMBLER

This 68000 disassembler was developed so that I could get a closer look at the Forth kernel which came with the CAT 1600 imaging system discussed elsewhere in this book. The Forth kernel was implemented by Ergonomic Research Group for a 68000 based microcomputer in a S-100 card cage. I was able to use this kernel to develop all the code necessary to control the CAT 1600 Imaging System without having to understand the internal structures in the kernel. However, it presented a tempting challenge to me and invited me to break into it. There were many features in the kernel which especially intriqued me, such as the Winchester disk and cartridge tape controlling words, mostly done in machine code. I used this disassembler to examine these words so that I could gain direct access to these peripheral devices.

The disassembler was not really finished, because I was pulled away from this computer for other more urgent tasks. Now that the ERG 68000 computer was junked, it is impossible to continue the work. As I digged through my code on the 8086 disassembler, several copies of this 68000 disassembler were also found and the two make a nice pair, compliment each other in many respects. It is very agonizing to decide what to do with this one, as it is still in a form not quite suitable for publishing. Without some amount of comments, it is doubtful that it can be of much use to people. After much tossing around, I decided that I will offer this 68000 disassembler in its currently form as an appendix to the 8086 disassembler, without comments. Those who need such disassembler are mostly likely well versed in Forth and in 68000 machine code to navigate through the source code.

68000 is a messed-up PDP-11 clone. It's got the look-and-feel of the much cleaner PDP-11 code and addressing scheme. However, by adding so many extensions, the machine code is well randomized so that it is a good size job to disassemble it. This disassembler classifies the 68000 code into 16 different types, according to the most significant nibble in a machine instruction. Each type is further decoded, and all the exceptions have to be processed accordingly. The aim is to print the machine code in the Motorola assembly mnemonic style.

The Forth underneath is based on the fig-FORTH model with 32 bit extensions. All addresses and stack items are 32 bit in length. @ and ! also uses 32 bit addresses and 32 bit data. For 16 bit quantities, W@ and W! are used. C@ and C! are used to deal with bytes, as usual. For defining words, it used the <BUILDS ... DOES> structure instead of the modern CREATE from any other Forth systems.

The last word in Screen 921 is SEARCH. This word is used to decode all the code words in the context vocabulary. Part of the dictionary thus disassembled is shown after the source code.

Figure 6. Index of 68000 disassembler

```
OK
OK

 0 ( LOAD SCREEN FOR 68000 DISSAMBLER, CHT, 18-JAN-85)
 1 ( DISSAMBLER, CASE, CHT, 18-JAN-85)
 2 ( FIELD EXTRACTION, CHT, 18-JAN-85)
 3 : .P ." A0A1A2A3WPIPRPSP" ;
 4 ( EFFECTIVE ADDRESS DECODING, CHT, 21-JAN-85)
 5 ( EFFECTIVE ADDRESS, CHT, 21-JAN-85)
 6 ( EFFECTIVE ADDRESS, CHT, 21-JAN-85)
 7 ( TYPE 0 CODES, CHT, 25-JAN-85)
 8 ( MOVE CODES, CHT, 21-JAN-85)
 9 ( MISCELLANEOUS CODES, CHT, 22-JAN-85)
10 ( MISCELLANEOUS CODES, CHT, 22-JAN-85)
11 ( SELECT MISCELLANEOUS CODES, CHT, 22-JAN-85)
12 ( TYPE 5 CODES, CHT, 22-JAN-85)
13 ( TYPE 6 CODES, CHT, 22-JAN-85)
14 ( TYPE 8 CODES, CHT, 22-JAN-85)
15 ( TYPE 9 AND 11 CODES, CHT, 23-JAN-85)
16 ( TYPE 12 CODES, CHT, 23-JAN-85)
17 ( TYPE 12 CODES, CHT, 23-JAN-85)
18 ( TYPE 13 CODES, CHT, 23-JAN-85)
19 ( TYPE 14 CODES, CHT, 23-JAN-85)
20 ( DISASSEMBLER, CHT, 23-JAN-85)
21 ( SEARCH AND DECODE, CHT, 25-JAN-85)
```

```
(FIND)
  6064    41203  CLR.L    D3
  6066    41204  CLR.L    D4
  6070    41207  CLR.L    D7
  6072    33037  MOVE.W   A7@+,D3
  6074    34037  MOVE.W   A7@+,D4
  6076    23103  MOVE.L   D3,A3
  6100    24104  MOVE.L   D4,A4
  6102    13033  MOVE.B   A3@+,D3
  6104    17003  MOVE.B   D3,D7
  6106     1103  ANDI     #77 ,D3
  6112    63472  BEQ      6206
  6114    36003  MOVE.W   D3,D6
  6116     1106  ANDI     #37 ,D6
  6122   133034  CMP.B    A4@+,D3
  6124    63042  BNE      6170
  6126    15023  MOVE.B   A3@,D5
  6130     1005  ANDI     #177 ,D5
  6134   135034  CMP.B    A4@+,D5
  6136    63030  BNE      6170
  6140    51506  SUBQ.W   #106 ,D6
  6142     4033  BTST     #7 ,A3@+
  6146    63756  BEQ      6126
  6150   153306  ADD.L    D6,D3
  6152    54113  ADDQ.W   #113 ,A3
  6154    37413  MOVE.W   A3,A7-@
  6156    37407  MOVE.W   D7,A7-@
  6160    37474  MOVE.W   #1 ,A7-@
  6164    60000  BRA      4540
  6170   153306  ADD.L    D6,D3
  6172    45123  TST.W    A3@
  6174    63410  BEQ      6206
  6176    33023  MOVE.W   A3@,D3
  6200    23103  MOVE.L   D3,A3
  6202    24104  MOVE.L   D4,A4
  6204    60274  BRA      6102
  6206    41147  CLR.W    A7-@
  6210    60000  BRA      4540
  6214   103505  OR.W     D3,D5
J
  6040    37456  MOVE.W   A6@4 ,A7-@
  6044    60000  BRA      4540
  6050      206  ORI      #5021444516 ,D6
I
  6024    37426  MOVE.W   A6@,A7-@
  6026    60000  BRA      4540
  6032   100712  DIVS     A2,D0
(DO)
  6002    36457  MOVE.W   A7@2 ,A6-@
  6006    36427  MOVE.W   A7@,A6-@
  6010    54117  ADDQ.W   #117 ,A7
  6012    60000  BRA      4540
  6016   100711  DIVS     A1,D0
```

Figure 7. Examples of disassembled 68000 machine code

```
(+LOOP)
  5724    34027   MOVE.W   A7@,D4
  5726   154526   ADD.W    D4,A6@
  5730    45137   TST.W    A7@+
  5732    65416   BMI      5752
  5734    34056   MOVE.W   A6@2 ,D4
  5740   114126   SUB.W    A6@,D4
  5742    67734   BLE      5700
  5744   155325   ADD.L    A5@,D5
  5746    60000   BRA      4540
  5752    34026   MOVE.W   A6@,D4
  5754   114156   SUB.W    A6@2 ,D4
  5760    67716   BLE      5700
  5762   155325   ADD.L    A5@,D5
  5764    60000   BRA      4540
  5770      204   ORI      #5021047651 ,D4
(LOOP)
  5656    51126   ADDQ.W   #126 ,A6@
  5660    34026   MOVE.W   A6@,D4
  5662   134156   CMP.W    A6@2 ,D4
  5666    63410   BEQ      5700
  5670    61006   BHI      5700
  5672   155325   ADD.L    A5@,D5
  5674    60000   BRA      4540
  5700    54116   ADDQ.W   #116 ,A6
  5702    52115   ADDQ.W   #115 ,A5
  5704    60000   BRA      4540
  5710   103450   OR.B     D3,A0@25514
0BRANCH
  5622    45137   TST.W    A7@+
  5624    63006   BNE      5634
  5626   155325   ADD.L    A5@,D5
  5630    60000   BRA      4540
  5634    52115   ADDQ.W   #115 ,A5
  5636    60000   BRA      4540
  5642      206   ORI      #5023047517 ,D6
BRANCH
  5600   155325   ADD.L    A5@,D5
  5602    60000   BRA      4540
  5606   103460   OR.B     D3,A0@122 D4.L
EXECUTE
  5552    41204   CLR.L    D4
  5554    34037   MOVE.W   A7@+,D4
  5556    24104   MOVE.L   D4,A4
  5560    60000   BRA      4546
  5564      206   ORI      #10224440516 ,D6
LIT
  5530    37435   MOVE.W   A5@+,A7-@
  5532    60000   BRA      4540
  5536   103505   OR.W     D3,D5
BYE
  5514    41100   CLR.W    D0
  5516    47102   47102    ILLEGAL CODE
  5520   101514   OR.W     D1,A4
```

Figure 7. Examples of disassembled 68000 machine code (cont'd)

```
      Scr #900
  0 ( LOAD SCREEN FOR 68000 DISSAMBLER, CHT, 18-JAN-85)
  1 901 921 THRU
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```
C. H. TING 6-FEB-85

```
      Scr #901
  0 ( DISSAMBLER, CASE, CHT, 18-JAN-85)
  1 : CASE:   <BUILDS   [COMPILE] ]   SMUDGE
  2    DOES>   SWAP 2* + @   EXECUTE   ;
  3 : COMMA   44 EMIT ;
  4 : PERIOD   46 EMIT   ;
  5 : POUND   35 EMIT   ;
  6 : TAB   9 EMIT   ;
  7 EXIT
  8 CASE: TEST 0 1 2 ;
  9 0 TEST .
 10 1 TEST .
 11 2 TEST .
 12
 13 ALL WORDS IN THE CASE DEFINITION MUST BE PREDEFINED.
 14 NO NUMBERS.
 15
```
C. H. TING 6-FEB-85

```
      Scr #902
  0 ( FIELD EXTRACTION, CHT, 18-JAN-85)
  1 : FIELD   ( MASK OFFSET N --- N' )
  2    ROT SWAP ?DUP IF   0 DO 2/ LOOP   THEN   AND   ;
  3 : SOURCE   ( N --- SOURCE-REG )   7 AND   ;
  4 : DEST     ( N --- DEST-REG   )   7 9 FIELD   ;
  5 : SIZE     ( N --- SIZE )         3 6 FIELD   ;
  6 : S-MODE   ( N --- SOURCE-MODE)   7 3 FIELD   ;
  7 : D-MODE   ( N --- DEST-MODE  )   7 6 FIELD   ;
  8 : EA       ( N --- EFF-ADDR   )   63 AND   ;
  9 : OP-MODE  ( N --- OP-MODE )      D-MODE   ;
 10 : R/M      ( N --- R/M )         1 3 FIELD   ;
 11 : CONDITION ( N --- CONDITION )   15 8 FIELD   ;
 12 : ID       ( N --- ID-FIELD )    1 8 FIELD   ;
  3 : SZ       ( N --- SZ-FIELD )    1 6 FIELD   ;
 14 : U.R   0 SWAP D.R   ;
 15
```

Listing 7. 68000 disassembler

```
      Scr #903
  0 : .P ." A0A1A2A3WPIPRPSP" ;
  1 ' .P 3 + CONSTANT .PC
  2 : .AN ( N --- )   2* .PC + 2 TYPE   ;
  3 : .DN   ( N --- )   68 EMIT   48 + EMIT   ;
  4 : .AN@  ( N --- )   .AN   64 EMIT   ;
  5 : .AN@+   ( N --- ) .AN@   43 EMIT   ;
  6 : .AN-@   ( N --- ) .AN   45 EMIT   64 EMIT   ;
  7 : .DATA ( N --- )   255 AND DUP 128 AND IF 256 - THEN .   ;
  8 : .D/A   ( REG F --- )   IF .AN ELSE .DN THEN   ;
  9 : .W/L   ( F --- )   PERIOD   IF 87 ELSE 76 THEN EMIT   ;
 10 : .INDEX   ( INDEX --- )   DUP .DATA
 11    DUP 7 12 FIELD OVER 0<   .D/A
 12    2048 AND .W/L   ;
 13
 14
 15
C. H. TING                                          6-FEB-85

      Scr #904
  0 ( EFFECTIVE ADDRESS DECODING, CHT, 21-JAN-85)
  1 VARIABLE IP#
  2 6 ALLOT   IP# 2+ IP# !   IP# 2+ 6 -1 FILL
  3 : @IP   ( --- N )   IP# @ @   ;
  4 : @NEXT   ( --- N )   IP# @ 2+   DUP IP# !   @   ;
  5 : MODE5 ( REG --- )   .AN@   @NEXT   . ;
  6 : MODE6   ( REG --- )   .AN@   @NEXT .INDEX   ;
  7 : ABS-SHORT   ( --- )   @NEXT .   0 .W/L   ;
  8 : ABS-LONG   ( --- )   @NEXT @NEXT SWAP D.   1 .W/L   ;
  9 : PC+D   ( --- )   ." PC@"   @NEXT .   ;
 10 : PC+X   ( --- )   ." PC@"   @NEXT .INDEX   ;
 11 : (IMMD)   ( --- )   @IP SIZE 1   > IF @NEXT ELSE 0 THEN
 12       POUND   @NEXT   SWAP D.   ;
 13 : IMMD   ( --- )   @IP 15 12 FIELD   ?DUP IF
 14    2 = IF @NEXT ELSE 0 THEN POUND @NEXT SWAP D.
 15    ELSE (IMMD) THEN   ;
C. H. TING                                          6-FEB-85

      Scr #905
  0 ( EFFECTIVE ADDRESS, CHT, 21-JAN-85)
  1 : .ILL   ."  ILLEGAL CODE"   ;
  2 CASE: SPECIAL ( 0-4)   ABS-SHORT   ABS-LONG   PC+D   PC+X
  3    IMMD   .ILL   .ILL   .ILL   ;
  4 CASE: EFF-ADD   ( REG MODE --- )
  5    .DN   .AN   .AN@   .AN@+   .AN-@   MODE5   MODE6   SPECIAL ;
  6 : (.EA) ( N --- )   DUP SOURCE SWAP S-MODE   EFF-ADD   ;
  7 : .EA ( --- )   @IP (.EA)   ;
  8 : .SIZE  ( SIZE --- )
  9    ?DUP IF 1 = .W/L   ELSE PERIOD 66 EMIT THEN   ;
 10 : .NAME ( --- , MUST BE THE 1ST WORD IN A .NAME DEFINITION.)
 11    R@ 7 -   -1 TRAVERSE COUNT 31 AND TYPE   ;
 12
  3
 14
 15
```

Listing 7. 68000 disassembler (cont'd)

```
      Scr #906
  0 ( EFFECTIVE ADDRESS, CHT, 21-JAN-85)
  1 : .SIZ-EA ( --- )   @IP SIZE .SIZE   TAB .EA    ;
  2 : .R-EA ( --- )   TAB @IP DEST .DN   COMMA    .EA    ;
  3 : .-EA ( --- )   TAB .EA    ;
  4 : .STATIC ( --- )   TAB @IP   POUND @NEXT U.    COMMA (.EA) ;
  5 : (.IMM) ( --- )     @IP @NEXT   OVER SIZE 1 >
  6    IF @NEXT SWAP   ELSE 0   THEN
  7    TAB POUND D. COMMA   (.EA)   ;
  8
  9
 10
 11
 12
 13
 14
 15
C. H. TING                                              6-FEB-85

      Scr #907
  0 ( TYPE 0 CODES, CHT, 25-JAN-85)
  1 : ORI   .NAME (.IMM)    ;                : ANDI  .NAME (.IMM)    ;
  2 : SUBI  .NAME (.IMM)    ;                : ADDI  .NAME (.IMM)    ;
  3 : EORI  .NAME (.IMM)    ;                : CMPI  .NAME (.IMM)    ;
  4 : .BIT    @IP ID    IF .R-EA ELSE .STATIC THEN   ;
  5 : BTST    .NAME    .BIT    ;              : BCHG .NAME .BIT    ;
  6 : BCLR    .NAME    .BIT    ;              : BSET .NAME .BIT    ;
  7 CASE: .DYN    BTST BCHG BCLR BSET    ;
  8 : <DYN>    @IP SIZE .DYN    ;
  9 CASE: .IMM    ORI ANDI SUBI ADDI <DYN> EORI CMPI .ILL    ;
 10 : MOVEP .NAME    @IP DUP 2DUP     SZ 0= .W/L TAB
 11     SIZE 2 = IF DEST .DN    COMMA     SOURCE MODE5
 12     ELSE SOURCE MODE5    COMMA    DEST .DN THEN    ;
 13 : TYPE0   @IP DUP ID    IF
 14     S-MODE 1 =    IF MOVEP    ELSE @IP SIZE .DYN THEN
 15     ELSE DEST .IMM THEN    ;
C. H. TING                                              6-FEB-85

      Scr #908
  0 ( MOVE CODES, CHT, 21-JAN-85)
  1 : .EAS    TAB    @IP >R    R@ SOURCE R@ S-MODE EFF-ADD
  2       COMMA    R@ DEST R> D-MODE EFF-ADD    ;
  3 : MOVE.B .NAME .EAS    ;
  4 : MOVE.L .NAME .EAS    ;
  5 : MOVE.W .NAME .EAS    ;
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

Listing 7. 68000 disassembler (cont'd)

```
      Scr #909
  0 ( MISCELLANEOUS CODES, CHT, 22-JAN-85)
  1 : .SELF   <BUILDS DOES>   7 - -1 TRAVERSE COUNT 31 AND TYPE ;
  2 .SELF RESET   .SELF NOP     .SELF RTE
  3 .SELF RTS     .SELF TRAPV   .SELF RTR
  4 : STOP .NAME TAB POUND @NEXT U.   ;
  5 : .REG   TAB   @IP SOURCE .AN   ;
  6 : UNLK .NAME .REG   ;           : LINK .NAME .REG   ;
  7 : MOVE>USP .NAME .REG   ;       : MOVE<USP .NAME .REG   ;
  8 : EXT   .NAME @IP DUP   SZ 0= .W/L   TAB   SOURCE .DN   ;
  9 : .SWAP ." SWAP" TAB   @IP SOURCE .DN   ;
 10 : TRAP .NAME   TAB   @IP 15 AND .   ;
 11 : NEGX .NAME .SIZ-EA   ;       : .CLR ." CLR" .SIZ-EA   ;
 12 : NEG .NAME .SIZ-EA   ;        : .NOT ." NOT" .SIZ-EA   ;
 13 : TST .NAME .SIZ-EA   ;
 14
 15
C. H. TING                                          6-FEB-85

      Scr #910
  0 ( MISCELLANEOUS CODES, CHT, 22-JAN-85)
  1 : MOVE<SR .NAME .-EA   ;        : MOVE>CCR .NAME .-EA   ;
  2 : MOVE>SR .NAME .-EA   ;        : NBCD .NAME .-EA   ;
  3 : .PEA ." PEA" .-EA   ;         : TAS .NAME .-EA   ;
  4 : JSR .NAME .-EA   ;            : JMP .NAME .-EA   ;
  5 : MOVEM   @IP S-MODE 0= IF EXT EXIT THEN   ." MOVEM"
  6     @IP SZ 0= .W/L   TAB   @IP DUP 1 10 FIELD
  7     @NEXT SWAP IF U. COMMA (.EA) ELSE SWAP (.EA) COMMA U. THEN ;
  8 : CHK .NAME .R-EA   ;           : LEA .NAME .R-EA   ;
  9 CASE: .RESET   RESET NOP STOP RTE .ILL RTS TRAPV RTR   ;
 10 : <RESET>   @IP SOURCE .RESET   ;
 11 CASE: .TRAP TRAP TRAP LINK UNLK MOVE>USP MOVE<USP <RESET> .ILL ;
 12   : <TRAP>   @IP S-MODE .TRAP   ;
 13 CASE: .JSR   .ILL <TRAP>   JSR JMP   ;
 14 : <JSR>   @IP SIZE .JSR   ;
 15
C. H. TING                                          6-FEB-85

      Scr #911
  0 ( SELECT MISCELLANEOUS CODES, CHT, 22-JAN-85)
  1 : PEA   @IP S-MODE IF .PEA ELSE .SWAP THEN   ;
  2 CASE: .MOVE   MOVE<SR .ILL MOVE>CCR MOVE>SR EXT
  3    TST MOVEM <JSR>   ;
  4 : <MOVE>   @IP DEST .MOVE   ;
  5 CASE: .NBCD   NBCD PEA MOVEM MOVEM ;
  6 : <NBCD>   @IP SIZE .NBCD   ;
  7 CASE: .NEGX   NEGX .CLR NEG .NOT <NBCD> TST MOVEM <TRAP>   ;
  8 : <NEGX>   @IP DEST .NEGX   ;
  9 CASE: .CHK   CHK LEA   ;
 10 : <CHK>   @IP SZ .CHK   ;
 11 CASE: .MISC   <NEGX> <NEGX> <NEGX> <MOVE>   ;
 12 : <MISC>   @IP SIZE .MISC   ;
  3 CASE: <TYPE4>   <MISC> <CHK>   ;
 14 : TYPE4   @IP ID <TYPE4>   ;
 15
```

Listing 7. 68000 disassembler (cont'd)

```
      Scr #912
  0 ( TYPE 5 CODES, CHT, 22-JAN-85)
  1 : .QUICK   @IP DUP SIZE .SIZE   TAB   POUND
  2    .DATA   COMMA   .EA   ;
  3 : ADDQ .NAME .QUICK   ;          : SUBQ .NAME .QUICK   ;
  4 CASE: .ADDQ   ADDQ SUBQ   ;
  5 : <ADDQ>   @IP ID .ADDQ   ;
  6 : CC ." RARAHILSCCCSNEEQVCVSPLMIGELTGTLE" ;
  7 ' CC 3 + CONSTANT 'CC
  8 : .CONDITION   ( N --- )   2* 'CC + 2 TYPE   ;
  9 : .DBCC   @IP  DUP ." DB"   CONDITION .CONDITION
 10    TAB   SOURCE .DN   COMMA   @NEXT @IP + 2+   U.   ;
 11 : .SCC   63 EMIT   @IP   CONDITION .CONDITION
 12    TAB   .EA   ;
 13
 14
 15
C. H. TING                                          6-FEB-85

      Scr #913
  0 ( TYPE 6 CODES, CHT, 22-JAN-85)
  1 : <DBCC>   @IP S-MODE 1 = IF .DBCC ELSE .SCC THEN   ;
  2 : TYPE5   @IP SIZE 3 = IF <DBCC> ELSE <ADDQ> THEN   ;
  3 : TYPE6   @IP DUP   66 EMIT CONDITION
  4    DUP 1 =    IF DROP ." SR"   ELSE .CONDITION   THEN
  5    TAB   255 AND ?DUP   IF
  6       DUP 128 AND IF 256 - THEN
  7    ELSE @NEXT THEN      IP# @ + 2+   U.   ;
  8 : TYPE7   ." MOVEQ" TAB POUND   @IP DUP    .DATA
  9    COMMA   DEST .DN   ;
 10
 11
 12
 13
 14
 15
C. H. TING                                          6-FEB-85

      Scr #914
  0 ( TYPE 8 CODES, CHT, 22-JAN-85)
  1 : .EA>D ( N --- )    .EA   COMMA DEST .DN   ;
  2 : .D>EA ( N --- )    DEST .DN   COMMA   .EA   ;
  3 : DIVU   .NAME TAB   @IP .EA>D   ;
  4 : DIVS   .NAME TAB   @IP .EA>D   ;
  5 : <OR>    @IP DUP   SIZE .SIZE TAB    DUP ID
  6    IF .D>EA ELSE .EA>D THEN   ;
  7 : .OR   ." OR"   <OR>   ;
  8 : .RXY   @IP DUP DEST OVER SOURCE   ROT R/M
  9    IF .AN-@ COMMA .AN-@   ELSE .DN COMMA .DN   THEN   ;
 10 : SBCD   .NAME TAB   .RXY   ;
 11 : TYPE8   @IP D-MODE   DUP 3 = IF DIVU
  2    ELSE DUP 7 =    IF DIVS
 13    ELSE DUP 4 =   @IP S-MODE   2 < AND IF SBCD ELSE .OR THEN
 .4    THEN THEN   DROP   ;
 15
```

Listing 7. 68000 disassembler (cont'd)

```
      Scr #915
  0 ( TYPE 9 AND 11 CODES, CHT, 23-JAN-85)
  1 : SUB    .NAME <OR>    ;
  2 : SUBX   .NAME @IP SIZE .SIZE    TAB    .RXY    ;
  3 : TYPE9    @IP $130 AND $100 =    IF SUBX ELSE SUB THEN    ;
  4 : CMP    .NAME <OR>    ;
  5 : CMPM   .NAME @IP SIZE .SIZE    TAB    .RXY    ;
  6 : EOR    .NAME @IP SIZE .SIZE    TAB    @IP DEST .DN
  7     44 EMIT    .EA    ;
  8 : TYPE11    @IP ID IF    @IP S-MODE 1 =    IF CMPM ELSE EOR THEN
  9     ELSE CMP THEN    ;
 10
 11
 12
 13
 14
 15
C. H. TING                                          6-FEB-85

      Scr #916
  0 ( TYPE 12 CODES, CHT, 23-JAN-85)
  1 : .AND    ." AND" <OR>    ;
  2 : MULU    .NAME TAB @IP .EA>D    ;
  3 : MULS    .NAME TAB @IP .EA>D    ;
  4 : ABCD    .NAME TAB    .RXY    ;
  5 : EXGD    .NAME TAB    @IP DUP DEST .DN
  6     COMMA    SOURCE .DN    ;
  7 : EXGA    .NAME TAB    @IP DUP DEST .AN
  8     COMMA    SOURCE .AN    ;
  9 : EXGM    .NAME TAB    @IP DUP DEST .DN
 10     COMMA    SOURCE .AN    ;
 11 : <TYPE12>    @IP D-MODE    DUP 3 = IF MULU
 12     ELSE DUP 7 =    IF MULS
 13     ELSE DUP 4 =    @IP S-MODE 2 <    AND IF ABCD ELSE .AND THEN
 14     THEN THEN    DROP    ;
 15
C. H. TING                                          6-FEB-85

      Scr #917
  0 ( TYPE 12 CODES, CHT, 23-JAN-85)
  1 : TYPE12    @IP 63 3 FIELD
  2     DUP 40 = IF EXGD DROP EXIT THEN
  3     DUP 41 = IF EXGA DROP EXIT THEN
  4         49 = IF EXGM DROP EXIT THEN
  5     <TYPE12>    ;
  6
  7
  8
  9
 10
 11
 12
  3
 14
 15
```

Listing 7. 68000 disassembler (cont'd)

```
      Scr #918
  0 ( TYPE 13 CODES, CHT, 23-JAN-85)
  1 : ADD    .NAME <OR>    ;
  2 : ADDX   .NAME @IP SIZE .SIZE    TAB    .RXY    ;
  3 : TYPE13    @IP $130 AND $100 =    IF ADDX ELSE ADD THEN    ;
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
C. H. TING                                                6-FEB-85

      Scr #919
  0 ( TYPE 14 CODES, CHT, 23-JAN-85)
  1 : AS .NAME ;    : LS .NAME ;    : ROX .NAME ;    : RO .NAME ;
  2 CASE: .SHRT    AS LS ROX RO    ;
  3 : .DIR    @IP ID IF ." L"    ELSE ." R"   THEN    ;
  4 : .MSHRT    @IP 3 9 FIELD .SHRT    .DIR TAB    .EA    ;
  5 : .RSHRT    @IP DUP 3 3 FIELD .SHRT    .DIR
  6    DUP SIZE .SIZE    TAB    DUP 1 5 FIELD
  7    IF DUP DEST .DN ELSE ." #" DUP DEST . THEN    44 EMIT
  8    SOURCE .AN    ;
  9 : TYPE14    @IP SIZE 3 =    IF .MSHRT ELSE .RSHRT THEN    ;
 10
 11
 12
 13
 14
 15
C. H. TING                                                6-FEB-85

      Scr #920
  0 ( DISASSEMBLER, CHT, 23-JAN-85)
  1 CASE: OPCODE    TYPE0    MOVE.B    MOVE.L    MOVE.W
  2    TYPE4    TYPE5    TYPE6    TYPE7    TYPE8    TYPE9    .ILL
  3    TYPE11    TYPE12    TYPE13    TYPE14    .ILL    ;
  4 : <DECODE>    CR    IP# @ 6 U.R    2 SPACES    @IP 6 U.R    TAB
  5       @IP 15 12 FIELD    OPCODE    2 IP# +!    ;
  6 : DECODE    ( ADDR N --- )
  7    OVER IP# !    +   ( LAST ADDR )
  8    BEGIN    <DECODE>    IP# @ OVER > UNTIL    DROP    ;
  9 : // <DECODE> ;
 10
 11
 12
 13
 14
 15
```

Listing 7. 68000 disassembler (cont'd)

```
      Scr #921
  0 (  SEARCH AND DECODE, CHT, 25-JAN-85)
  1 :  .WORD    ( OLD-NFA NFA LFA --- NFA NEXT-NFA )
  2     SWAP DUP >R     CR ID.    KEY DUP 81 = IF QUIT THEN    89 = IF
  3     DUP @ SWAP ( NEXT-NFA LFA )    4 + ROT OVER - 2- ( LENGTH)
  4     DECODE    ELSE @ SWAP DROP ( OLD-NFA ) THEN
  5     R> SWAP    ;
  6 :  ?CODE    ( NFA --- LFA FLAG )
  7     1 TRAVERSE 1+ ( LFA )    DUP 2+ @ ( CODE )
  8     OVER 4 +    =    ;
  9 :  SEARCH    ( --- , GO THRU CONTEXT VOCABULARY)
 10     CONTEXT @ @    DUP ?CODE DROP @    ( OLD-NFA NFA )
 11     BEGIN    DUP ?CODE    OVER @ WHILE
 12     IF .WORD    ELSE ROT DROP @ THEN    REPEAT
 13     2DROP    ;
 14
 15
C. H. TING                                          6-FEB-85

      Scr #922
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
C. H. TING                                          6-FEB-85

      Scr #923
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
  3
 14
 15
```

Listing 7. 68000 disassembler (cont'd)