ORNL/TM-10463

ornl

OAK RIDGE NATIONAL LABORATORY

MARTIN MARIETTA

Optimizing the Zilog Z8 FORTH Microcontroller for Rapid Prototyping

Robert Edwards

OPERATED BY
MARTIN MARIETTA ENERGY SYSTEMS, INC.
FOR THE UNITED STATES
DEPARTMENT OF ENERGY

Printed in the United States of America. Available from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Road, Springfield, Virginia 22161
NTIS price codes—Printed Copy: A04; Microfiche A01

ORNL/TM-10463

Energy Division

Optimizing the Zilog Z8 FORTH Microcontroller for Rapid Prototyping

Robert Edwards

Date Published - September 1987

Prepared for the
Smart House Project
National Association of Home Builders
Research Foundation

OAK RIDGE NATIONAL LABORATORY
Oak Ridge, Tennessee 37831
operated by
MARTIN MARIETTA ENERGY SYSTEMS, INC.
for the
U.S. DEPARTMENT OF ENERGY
under Contract No. DE-AC05-84OR21400

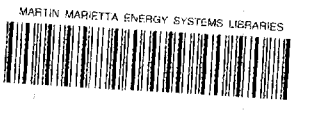

TABLE OF CONTENTS

ABSTRACT

This report presents techniques for modifying and extending Zilog Z8 FORTH microcontroller system software to improve its suitability for rapid prototyping, an increasingly popular method of developing new products and services. Rapid prototyping requires special product development methods and tools because it often mandates short term, radical changes in the concept being developed.

The work was conducted for the National Association of Home Builders Smart House Project, a cooperative research and development effort involving American home builders and a number of major corporations serving the home building industry. The major goal of the project is to help the participating organizations incorporate advanced technology in communications, energy distribution, and appliance control products for American homes.

Use of a high-performance, easily adaptable microcontroller can greatly facilitate a laboratory evaluation of Smart House product prototypes. Such a device can meet numerous sensing and testing needs, including the important function of simulating the operation of Smart House components associated with the one being tested.

The techniques described in this report improve Z8 execution rates up to 300 percent by replacing selected parts of the vendor-supplied FORTH software with routines optimized for speed rather than program size. Also, the report shows how to exploit an unusual Z8 FORTH system feature to simulate a multitasking environment (concurrent execution of several tasks).

This information is provided to project participants to help them select development and test equipment for Smart House products. The material is technical in nature and assumes considerable experience in microcontroller technology and microcomputer programming.

1. INTRODUCTION

This report presents techniques to improve the performance of the Zilog Z8/FORTH microcontroller when used as a tool for developing and testing new products and services. Product developers often rely on an expedited "build it, try it out, and change it if necessary" approach in development and testing. This technique, commonly called rapid prototyping, requires laboratory tools that permit quick changes in the concept of the product being developed. The applicability of one such tool, the Z8 FORTH single-board microcontroller, was the subject of a previous ORNL study (Edwards 1987). That study uncovered a Z8 FORTH execution rate limitation that can be easily overcome using the technique introduced in Sections 2 through 4 of this report. Section 5 shows how a multiprocessing environment can be simulated using Z8 FORTH.

The information in this report is intended to provide assistance in selecting equipment suitable for developing and testing products using rapid-prototyping techniques. The material is technical in nature and assumes considerable experience in microcontroller technology and microcomputer programming.

ORNL is developing rapid-prototyping tools for the National Association of Home Builders Smart House Project, a cooperative research and development effort involving American home builders and a number of major corporations associated with home building industry products and services. The project will help the participants incorporate advanced technology in new products for communications, energy distribution, and appliance control. One of the project's most important goals is to bring Smart House cabling, electrical control devices, consumer appliances, and gas products into the market for homes constructed in the 1990s.

The impetus for the Smart House project comes from two areas. One is the pressure of foreign consumer products, which dominate the after-sale consumer appliance market and threaten to make inroads into sales to home builders. The other is the increasing sophistication of modern electric

and gas appliances and the inability of present energy distribution and control systems to accommodate the potential these appliance offer.

The Smart House will provide intelligent control and coordination between appliances and their controlling devices. Aplpliance functions that are presently accomplished with difficulty, such as distributed control of multizone heating, ventilating, and air conditioning and multiunit remote control of entertainment components, are simple to support with the Smart House design.

This same intelligent control and coordination also makes possible a new standard of safety in the home, primarily by providing closed-loop control of electrical appliances. With closed-loop control, branch circuits are de-energized except when power is necessary to operate appliances. This feature is made possible through redesigning the function of an appliance switch, which in the Smart House is a signaling device rather than a power controller. When an appliance is turned on, a signal is sent from the appliance switch to a circuit controller, which is part of the Smart House wiring system. This device then powers the circuit that supplies energy to the appliance. Once the flow of power has been initiated, the appliance must send a recurring "nominal-operation" signal to the circuit controller to continue the power feed. If the circuit controller does not receive this signal, it assumes that a fault has occurred (e.g., the plug was pulled out of the outlet) and de-energizes the circuit. Closed-loop protection of circuits in the Smart House can be thought of as the electrical equivalent of the "proof-of-flame" protection in gas appliances.

The Smart House design must meet rigid requirements for reliability, installability, and maintainability. As an example of the attention being paid to reliability, a leading Smart House design proposal provides home control through use of several distributed controllers all wired together, each with the ability to back up one another in case of failure. This redundancy avoids a sudden loss of all home control functions that could occur if only one central controller were used.

ORNL is assisting the Smart House Project by providing technical evaluations of proposed designs, assisting NAHB project managers, and advising on facilities necessary for design evaluation and system integration.

The Zilog Z8, originally introduced in 1981, has become a popular microcomputer for single-board applications. The Z8 is available in several configurations, including single-chip BASIC and FORTH versions with internal read-only memory (ROM), versions without internal memory, and "piggy-back" versions that permit mounting 4K bytes of ROM or electronically-programmable read-only memory (EPROM) on the top of the microcontroller chip. A Z8-based single-board microcontroller and auxiliary support boards are readily available at low cost (Ciarcia 1981).

FORTH is a high-level language widely used in real-time control applications. Its most distinctive feature is the ease with which any FORTH routine (called words) can be changed. In fact, a FORTH programmer can completely redefine even basic system words if the need arises.

Charles Moore developed FORTH in the 1970s for use in telescope control. During the early 1980s, FORTH became popular with amateur programmers because it permitted using a high-level language on microcomputers with limited memory and processing capacity. Since then, FORTH has had a checkered history. Its recent decline in popularity for general-purpose programming has resulted from a growing availability of high capability, low-cost microcomputers (such as the ubiquitous IBM PC clone) and the tendency of FORTH programmers to develop unstructured code that is difficult to follow and often impossible to alter. Guidelines for averting this latter problem are presented in Leo Brodie's excellent tutorial, *Starting FORTH* (Brodie 1981).

2. Z8 FORTH PERFORMANCE CHARACTERISTICS

In a previous report on Z8 FORTH (Edwards 1987), ORNL concluded that the relatively slow speed of the Z8 FORTH system software (average execution rate of 5000 FORTH instructions/second) could prohibit its use in certain types of rapid-prototyping. The report showed methods of circumventing this problem through use of assembly-language code for critical parts of an application program. Table 1 of that report (duplicated on page 6 in this section) shows execution times for two types of test loops. Execution times of the vendor-supplied Z8 FORTH (called Z8 FORTH nonoptimized) and the improved Z8 FORTH (optimized) are compared with versions of FORTH on other microcomputers[1]. Although the optimized Z8 FORTH benchmark is very fast, achieving rates as much as four times greater than FORTH on an IBM PC, use of FORTH in this manner is not very different from simply using assembly language to accomplish the function.

The nonoptimized Z8 timing results shown in the 16-bit store column of Table 1 used repeated calls to TEST, a 16-bit store timing test[2], that is defined as

```
: TEST DO 20 I ! LOOP ;
```

As can be seen in the listing of Z8 code equivalent of the timing test (page 7), extensive use of assembly language can be a problem because it defeats the transportability, commonality, and readability advantages that are the main reasons for using FORTH in the first place.

---

[1]The last entry in the table, Novix FORTH, executes on a Novix NC4000, a microprocessor that uses FORTH words as its native instruction set.

[2]The word TEST assumes the initial and final limits for the loop are on the stack when TEST is executed. The sequence "20 I !" simply stores the constant 20 at the address specified by the value of the loop index.

Table 1. Benchmarks of Z8 FORTH vs. FORTH on other microcomputers

FORTH version	Seconds to perform 1,000,000 iterations Contents of loop Empty loop[a]	16-bit store[b]
Z8 FORTH Non-optimized	460.0	760.0
MVP-FORTH IBM-XT	107.30	325.19
MVP-FORTH IBM-AT	35.85	67.25
Z8 FORTH Optimized	8.00	24.00
Novix FORTH	2.50	3.22

[a]FORTH definition - : TEST DO LOOP ;
[b]FORTH definition - : TEST DO 20 I ! LOOP ;

Z8 Code[3]		Assembly Language[4]		Comments
		LBL	TEST	16-bit store test loop
50EC		POP	RC	Pop Top-of-Stack into working register
50ED		POP	RD	pair C and D (Character count)
50EA		POP	RA	Pop word underneath TOS in working
50EB		POP	RB	register pair A and B (Address)
9C00		LD	R8,"00"	Load a zero into working register 8
9C20		LD	R9,"20"	Load a blank into working register 9
D28A	LH	LDC	@WA,R8	Store zero in address pointed to be A&B
A0EA		INCW	WA	Point A and B to the next address
D29A		LDC	@WA,R9	Store blank in address pointed to by A&B
A0EA		INCW	WA	Point A and B to next address
80EC		DECW	WC	Decrement character count in C and D
EBF4		JR	NZ,LH	Repeat loop if character count not zero
3050		JP	@50	Exit to next FORTH word

The readability situation is made even worse when the assembly language code is converted into FORTH to permit its use in a conventional FORTH program:

```
CREATE TEST SMUDGE HERE 2 + , 50EC , 50ED , 50EA , 50EB , 8C00 , 9C20 ,
    D28A , A0EA , D29A , A0EA , 80EC , EBF4 , 3050 ,
```

This report explores methods of increasing the speed of Z8 FORTH by redefining existing FORTH system words rather than substituting Z8 assembly language for groups of application-specific words as was done for the timing test. Using the techniques presented in Section 4 of this report, it is possible to write the timing test in the form

```
: TEST DO 20 I ! LOOP ;
```

yet achieve execution speeds faster than is possible with the original vendor-supplied FORTH definitions.

---

[3]Z8 code, memory addresses, and constants are in hexadecimal throughout this report.

[4]Standard Z8 assembly language mnemonics [cf. *Z8 Microcomputer Technical Reference Manual* (Zilog Inc. 1984)] have been altered in this report to conform to conventions used with other microcontrollers.

3. INTRODUCTION TO Z8 FORTH SYSTEM SOFTWARE ARCHITECTURE

A working knowledge of Z8 FORTH system software is required to take advantage of the optimization techniques described in this report. This section provides an overview of the system software, which is presented in Appendix A and B as detailed listings. Because Micromint, the developer of Z8 FORTH, does not furnish assembly-language listings of their FORTH system, the listings were developed by reconstructing FORTH and assembly language images of the binary code contained in Z8 ROM (Appendix C).

The first third of the Z8 FORTH system software is an assembly language section consisting of five parts:

- o initialization code (addresses 00C-0A8),
- o initialization tables (0A9-0EC),
- o stack/register-file utility routines (0ED-1A2),
- o subroutine to pass control to a subsequent word (1A3-1B3), and
- o coding for auxiliary and dictionary-defined FORTH words (1B4-59A).

The remainder of the system software, which is an adaptation of the FIG-FORTH 79 (FORTH Interest Group 1980), occupies addresses 59A through FFF. The FORTH section of the system is divided as follows:

- o Execution vectors for auxiliary FORTH words (59A-5A7, 600-607)
- o Block of constants (5A8-5FF)
- o "Headless" FORTH words (FORTH words having no entry in the dictionary)
  - Error message subroutine (6BE-635)
  - Error routine (636-643)
  - BLK subroutine (644-685)
  - Threaded execution routine (686-6BD)
- o Dictionary of FORTH words consisting only of a vector to a block of code in the assembly language section of the system (6BE-6C7)
- o Dictionary of FORTH words adapted from FIG-FORTH 79 (6C8-F83)

4. OPTIMIZATION OF Z8 SYSTEM WORDS

This section presents a technique for optimizing individual FORTH system words for speed rather than size of code, as was done for the vendor-supplied FORTH in Z8 ROM. The system word LOOP is used to illustrate the method.

The function of the FORTH code for loop control (locations 3A2 through 3B0, Appendix A, p. 32) is to increment the loop control index and branch back to the beginning of the loop (the last encountered DO) if the index does not exceed the index limit. Examination of the Z8 FORTH code for loop control shows extensive use of shared code. Note the branching to snatches of code in other routines (+LOOPSUB and LOADEV) and numerous calls to utility subroutines (POPA8, PUSH8, and COPYR>S) to minimize the amount of memory necessary to accomplish the function.

Better performance in the execution of loops can be achieved by optimizing the word LOOP for speed of execution rather than memory used. This is done by providing a new assembly language code called LOOPCODE and a new definition of the FORTH word LOOP for calling the new code when LOOP is executed.[5] The new definition of LOOP is

```
: LOOP [COMPILE] COMPILE [ FIND LOOPCODE , ] RELADR ; IMMEDIATE
```

This definition of LOOP is identical to the original (APPENDIX B., p. 43), except for linkage to the optimized routine LOOPCODE rather than the code for LOOP stored in internal ROM.

The assembly language routine LOOPCODE requires 43 bytes more space than the definition provided in Z8 internal ROM, but executes fewer instructions

---

[5]Compilation of the word LOOP requires inclusion of two additional FORTH words. The word [COMPILE] must be included for compilation of IMMEDIATE words such as COMPILE. RELADR, which is a headless word in the FORTH system software, is accessible only through creation of an appropriate head in the FORTH catalog (e.g., : RELADR [6BE ,] ;). Both of these techniques are discussed in *Z8 FORTH Manual* (Micromint Inc. 1984).

to accomplish the end-of-loop function. An assembly language listing of LOOPCODE followed by its FORTH word definition is shown below.

```
Z8 Code   Assembly Language        Comments
-------   ----------------------   ------------------------------------------
          LBL  LOOPCODE
3170      SRP  70                  Uses register file locations 70-74
4C70      LD   R4,"70"
C34A      LDCI @R4,@WA             Load INDEX to W0 from RSTACK
C34A      LDCI @R4,@WA
C34A      LDCI @R4,@WA             Load LIMIT to W2 from RSTACK
C23A      LDC  R3,@WA
A0E0      INCW W0                  Increment INDEX
2231      SUB  R3,R1               Subtract INDEX from LIMIT
3220      SBC  R2,R0
7B18      JR   MI,LPEXIT           Exit loop if updated INDEX exceeds LIMIT
80EA      DECW WA                  Point back to INDEX on RSTACK
80EA      DECW WA
D21A      LDC  @WA,R1              Push updated INDEX back on RSTACK
80EA      DECW WA
D20A      LDC  @WA,R0
C2DC      LDC  R0,@WC              Move IP adjustment to W0
A0EC      INCW WC
C21C      LDC  R1,@WC
80EC      DECW WC                  Point back to high byte of adjustment
02D1      ADD  R0,R1               Add adjustment to IP
12C0      ADC  RC,R0
3050      JP   @50

          LBL  LPEXIT
A0EC      INCW WC                  Skip over IP adjustment
A0EC      INCW WC
A0EA      INCW WA                  Point RSTACK back to low byte of INDEX
3050      JP   @50
```

```
CREATE LOOPCODE SMUDGE 3170 , 4C70 , C34A , C34A , C34A , C23A ,
  A0E0 , 2231 , 3220 , 7B18 , 80EA , 80EA , D21A , 80EA , D20A ,
  C2DC , A0EC , C21C , 80EC , 02D1 , 12C0 , 3050 , A0EC , A0EC ,
  A0EA , 3050 ,
```

Table 2 compares the timings of all the Z8 benchmarks presented in Table 1 with timings for the redefined word LOOP presented above. The redefined version of LOOP executes an empty loop 3.7 times faster (460 vs. 125 sec.) than the original. The version to accomplish a 16-bit store using the new version of LOOP is only 1.8 times faster because just one of the four FORTH words involved in the definition has been optimized.

Table 2. Comparison of redefined LOOP benchmarks with previous Z8 FORTH timings

	Seconds to perform 1,000,000 iterations	
	Contents of loop	
FORTH version	Empty loop[a]	16-bit store[b]
Z8 FORTH Non-optimized	460.0	760.0
Z8 FORTH with Redefined LOOP	125.0	425.0
Z8 FORTH Optimized	8.0	24.0

[a]FORTH definition - : TEST DO LOOP;

[b]FORTH definition - : TEST DO 20 I ! LOOP ;

5. MULTITASKING USING Z8 FORTH

The method employed by FORTH to exit from the code for system-defined FORTH words permits relatively easy implementation of a multitasking software environment. Examination of the assembly-language listing in Appendix A shows that each of the blocks of Z8 code used to implement a FORTH system word (EXECUTE, SWAP, DUP, @, !, etc.) exits via an indirect branch through register file location 50. When FORTH is booted-up, the address in location 50 is initialized to a pointer to the "execute next vector" (EXNV) code at address 1A3. By changing the pointer at location 50, a FORTH programmer can obtain control to accomplish a special task at the end of execution of each system word. The programmer-supplied exit routine can perform any of a variety of tasks, such as examining data on input ports or alternating execution between two or more FORTH words.

Multitasking execution of several FORTH words requires restoring system data areas (register file locations 20-3F, 76-7F) if these locations are to be used by a new routine to be executed. Figure 2 shows a multitasking routine that monitors the serial input port for a CONTROL-C break character. This routine does not use any of the system data areas, so that no save/restore is necessary. When a break character is found in the input buffer, control is passed to the FORTH word FATALERR, which prints the characters ?i on the screen and transfers control to the FORTH QUIT routine. The upper part of Figure 2 is an assembly-language listing of an EXNV substitute (NEXTCODE) to monitor the input port for a break character. The lower portion is the FORTH equivalent of the assembly-language routine.

To use the new code NEXTCODE, the vector at register file location 50 must be set to point to the new EXNV code. This can be done by executing the following sequence of FORTH words:

```
FIND NEXTCODE 50 !
```

After execution of this sequence, the input buffer will be continually checked for a CONTROL-C, the signal to interrupt execution.

```
Z8 Code     Assembly Language          Comments
---------   ------------------------   ------------------------------------
            LBL  NEXTCODE
3170        SRP  70                    Use Register file locations 70-74
C2EC        LDC  RE,@WC                Load next vector into WE (7E-7F)
A0EC        INCW WC                    Increment IP (7C-7D)
C2FC        LDC  RF,@WC
A0EC        INCW WC                    IP points to next vector at exit
            LBL  EXALT
C24E        LDC  R4,@WE                Load execution address into W4
A0EE        INCW WE                    Increment execution vector
C25E        LDC  R5,@WE
A0EE        INCW WE                    EV points to next body on exit
76FA08      CP   FA,"08"               Is input port ready?
6B19        JR   Z,EXNVEXIT            Zero means not ready
76F003      CP   F0,"03"               Is it a CONTROL-C?
EB14        JR   NZ,EXNVEXIT           Non-zero - not a CONTROL-C
56FAF7      AND  FA,"F7"               It was - set the input port empty
E65809      LD   58,"09"               Push error message "i" to stack
E65969      LD   59,"69"
D60111      CALL PUSH8
3170        SRP  70
EC06        LD   RE,"06"               Load vector to FATALERR into WE
FC36        LD   RF,"36"
8BDA        JR   EXNVALT

            LBL  EXNVEXIT
30E4        JP   @R4                   Exit via working register pair R4-R5
```

```
CREATE NEXTCODE SMUDGE 3170 , C2EC , A0EC , C2FC , A0EC , C24E ,
  A0EE , C25E , A0EE , 76 C, FA08 , 6B19 , 76 C, F003 , EB14 , 56
  C, FAF7 , E6 C, 5809 , E6 C, 5969 , D6 C, 0111 , 3170 , EC06 ,
  FC36 , 8BDA , 30E4 ,
```

Figure 1. Assembly-Language Code for NEXTCODE

6. CONCLUSIONS

This report presents two techniques for improving the performance and functionality of the Zilog Z8 FORTH microcontroller.

One technique shows how to redefine a critical FORTH system word, such as the word LOOP, to improve execution rates up to several times over those possible using only the vendor-provided system words. This technique improves execution speed while maintaining program readability.

The second technique permits implementation of a software-controlled multitasking environment using a system-provided feature that can cause special functions to be run immediately after execution of every FORTH word. This feature can be used to develop software to simulate a multiprocessing environment to enable simultaneous monitoring of input/output lines, parallel monitoring of sensor data, etc.

7. REFERENCES

Brodie, L., *Starting FORTH*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

Ciarcia S., "Build a Z8-Based Control Computer with BASIC," Parts I and II, *BYTE*, July-August 1981.

Edwards R., *Evaluation of a Single Board Microcomputer Suitable for Rapid Prototyping*, ORNL TM/10361, Oak Ridge National Laboratory, February 1987.

FIG-FORTH Installation Manual, FORTH Standards Team of the FORTH Interest Group, 1980.

Micromint Z8 FORTH Reference Manual, Micromint Inc., 1984.

Z8 Microcomputer Technical Reference Manual, Zilog Inc., 1984.

APPENDIX A

ZILOG Z8 FORTH SYSTEM
ASSEMBLY-LANGUAGE SECTION

```
Addr Inst   Label Oprn Operands   Comments

000C 8F           DIS            DISABLE INTERRUPTS
000D 3150         SRP  50
000F EC00         LD   RE,"00"   LOAD REG F0-FF FROM 00B5
     FCB5         LD   RF,"B5"
     DCF0         LD   RD,"F0"   START WITH RD = F0
     CC10         LD   RC,"10"   DO 10 (HEX) BYTES
0017 C3DE   L017  LDCI @RD,@WE
     CAFC         DJNZ RC,L017
     EC00         LD   RE,"00"   LOAD REG FILE 10-2F FROM 00C5
     FCC5         LD   RF,"C5"
     DC10         LD   RD,"10"
     CC20         LD   RC,"20"   DO 20 (HEX) BYTES
     C3DE   L023  LDCI @RD,@WE
     CAFC         DJNZ RC,L023
0027 3170         SRP  70        USE RF 70-7F TO FIND LIMITS OF ROM & RAM
     AC00         LD   RA,"00"   LOOK FOR END OF ROM IN 1K INCREMENTS
     BC00         LD   RB,"00"
002D 06EA04 L02D  ADD  RA,"04"   ADD 400 (HEX) TO WA (FIRST BYTE OF PAGE)
     C27A         LDC  R7,@WA    LOAD BYTE POINTED TO BY WA INTO R7
     68E7         LD   R6,R7     COPY THE BYTE INTO R6
     60E6         COM  R6        COMPLIMENT THE COPY IN R6
     D26A         LDC  @WA,R6    ATTEMPT TO STORE THE COPY TO MEMORY
     C26A         LDC  R6,@WA    LOAD IT BACK TO R6
     D27A         LDC  @WA,R7    RESTORE MEMORY FROM R7 (IN CASE IT WAS RAM)
     A267         CP   R6,R7     COMPARE R6 AGAINST R7
     6BED         JR   EQ,L02D   IF EQUAL WE ARE STILL IN ROM
0040 C8EA         LD   RC,RA     STORE BEGINNING OF RAM IN WA
     D8EB         LD   RD,RB
     C912         LD   12,RC     STORE BEGINNING OF RAM IN DICTIONARY PNTR
     D913         LD   13,RD
0048 06EA04 L048  ADD  RA,"04"   LOOK FOR END OF RAM IN 1K INCREMENTS
     C27A         LDC  R7,@WA    LOAD BYTE POINTED TO BY WA INTO R7
     68E7         LD   R6,R7     COPY THE BYTE INTO R6
     60E6         COM  R6        COMPLIMENT THE COPY IN R6
     D26A         LDC  @WA,R6    ATTEMPT TO STORE THE COPY TO MEMORY
     C26A         LDC  R6,@WA    LOAD IT BACK TO R6
     D27A         LDC  @WA,R7    RESTORE MEMORY FROM R7 (IN CASE IT WAS RAM)
     A267         CP   R6,R7     COMPARE R6 AGAINST R7
0059 EBED         JR   NE,L048   IF NOT EQUAL WE ARE STILL IN RAM
005B A922         LD   22,RA     RSTACK AT END-OF-RAM + 1
     B923         LD   23,RB
     00EA         DEC  RA        SUBTRACT 100(HEX) FROM END-OF-RAM
     A920         LD   20,RA     START DSTACK AT 100(HEX) LESS THAN RSTACK
     B921         LD   21,RB
     A91A         LD   1A,RA     SET TIB (1A-1B) TO BOTTOM OF RSTACK
     B91B         LD   1B,RB
```

```
Addr Inst  Label Oprn Operands   Comments

     80EC        DECW WC        LOAD STARTING ADDR (LAST WORD IN ROM)
     C27C        LDC  R7,@WC      INTO R6-R7
     80EC        DECW WC        IF START ADDR IS FFFF THEN USE START ADDR
     C26C        LDC  R6,@WC      AT LAST WORD OF Z8 ROM (@FFE)
     A0E6        INCW W6        TEST IF ADDR EQUALS "FFFF"
     EB04        JR   NZ,L079
     CC0F        LD   RC,"0F"   LOAD EXECUTION VECTOR WITH POINTER TO ADDRESS
     DCFE        LD   RD,"FE"     IN NUCLEUS (QUIT) WHERE EXECUTION IS TO BEGIN
     3150  L079 SRP  50          POINT TO FORTH WORD REGISTER GROUP
     8CFF        LD   R8,"FF"   LOAD W8 WITH BAUD RATE SWITCH ADDRESS "FFFD"
     9CFD        LD   R0,"FD"
     C278        LDC  R7,@WR8   LOAD R7 WITH VALUE OF BAUD RATE SWITCHES
     8D0F84      JP   0F84      INITIALIZE SHARED-BUS INTERRUPT ROUTINE
0084 8C00        LD   R8,"00"
     9CE5        LD   R9,"E5"
     0297        ADD  R9,R7
     16E800      ADC  R8,"00"
     C278        LDC  R7,@W8
     79F4        LD   F4,R7
0091 E6F103      LD   F1,"03"
     E420FE      LD   FE,20     INITIALIZE STARTING VALUE OF STACK
     E420FF      LD   FF,21
     E4227A      LD   7A,22     INITIALIZE STARTING VALUE OF RSTACK
     E4237B      LD   7B,23
00A0 E65001      LD   50,"01"   INITIALIZE VECTOR TO "EXNV" CODE
     E651A3      LD   51,"A3"
     9F          EI             ENABLE INTERRUPTS
     3050        JP   @50       GO TO NEXT CODE

                 LBL  POP8      POP STACK TO W8 (R8-R9)
00ED 3150        SRP  50
     50EE        POP  RE        POP RETURN ADDRESS TO WE
     50EF        POP  RF
     50E8        POP  R8        POP STACK TO W8
     50E9        POP  R9
     305E        JP   @5E       RETURN VIA WE

                 LBL  POPA8     POP TOP-OF-STACK TO WA (RA-RB);
                                  NEXT-TO-TOP-OF-STACK TO W8 (R8-R9)
00F9 3150        SRP  50
     50EE        POP  RE        POP RETURN ADDRESS TO WE
     50EF        POP  RF
     50EA        POP  RA        POP STACK TO WA
     50EB        POP  RB
     8BEE        JR   L0F3      GO TO POP8 TO POP NEXT WORD ON STACK INTO W8

                 LBL  POPCA8    POP TOP-OF-STACK TO WC (WC-WD);
                                  NEXT-TO-TOP-OF-STACK TO WA (RA-RB);
                                   THIRD-FROM-TOP TO W8 (R8-R9)
```

```
Addr Inst  Label Oprn Operands   Comments

0105 3150        SRP  50
     50EE        POP  RE         POP RETURN ADDRESS TO WE
     50EF        POP  RF
     50EC        POP  RC         POP STACK TO WC
     50ED        POP  RD
     8BEE        JR   LOFF       GOTO POPA8 - POP NEXT TWO WORDS TO WA & W8

                 LBL  PUSH8      PUSH W8 TO STACK
0111 3150        SRP  50
     50EE        POP  RE         POP RETURN TO WE
     50EF        POP  RF
0117 70E9   L117 PUSH E9         PUSH W8 (R9-R8) TO STACK
     70E8        PUSH E8
     305E        JP   @5E

                 LBL  PUSHA8     PUSH WA THEN W8 TO THE STACK
011D 3150        SRP  50
     50EE        POP  RE         POP RETURN TO WE
     50EF        POP  RF
     70EB   L123 PUSH RB         PUSH WA (RB-RA) TO STACK
     70EA        PUSH RA
     8BEE        JR   L117       GOTO PUSH8 TO PUSH W8 TO STACK

                 LBL  PUSHCA8    PUSH WC, THEN WA, THEN W8 TO STACK
0129 3150        SRP  50
     50EE        POP  RE         POP RETURN TO WE
     50EF        POP  RF
     70ED        PUSH RD         PUSH WC (RD-RC) TO STACK
     70EC        PUSH RC
     8BEE        JR   L123       GOTO PUSHA8 TO PUSH WA, W8 TO STACK

                 LBL  R>S        POP RSTACK - PUSH TO STACK
0135 3150        SRP  50
     50EE        POP  RE         POP RETURN ADDRESS TO WE
     50EF        POP  RF
     687A        LD   R6,7A      LOAD RSTACK POINTER INTO W6
     787B        LD   R7,7B
     C286        LDC  R8,@W6     POP RSTACK TO W8
     A0E6        INCW W6
     C296        LDC  R9,@W6
     A0E6        INCW W6
     797B        LD   7B,R7      UPDATE RSTACK POINTER
     697A        LD   7A,R6
     8BCA        JR   L117       PUSH W8 TO STACK

                 LBL  S>R        POP STACK - PUSH TO RSTACK
     3150        SRP  50
     50EE        POP  RE         POP RETURN ADDRESS TO WE
     50EF        POP  RF
```

```
Addr Inst   Label Oprn Operands   Comments

     50E8         POP  R8         POP STACK TO W8 (R8-R9)
     50E9         POP  R9
0157 687A         LD   R6,7A      RSTACK POINTER TO W6
     787B         LD   R7,7B
     80E6   L15B  DECW W6
     D296         LDC  @W6,R9     W8 (R9-R8) TO RSTACK
     80E6         DECW W6
     D286         LDC  @W6,R8
     697A         LD   7A,R6      UPDATE RSTACK POINTER
     797B         LD   7B,R7
     30EE         JP   @RE        RETURN VIA WE

                  LBL  LOADNV     LOAD NEXT VECTOR TO W8, INCREMENT IP (7C-7D)
0169 3150         SRP  50
     A87C         LD   RA,7C      INSTRUCTION POINTER TO WA
     B87D         LD   RB,7D
     C28A         LDC  R8,@WA     VECTOR AT IP TO W8
     A0EA         INCW WA
     C29A         LDC  R9,@WA
     A0EA         INCW WA
     A97C         LD   7C,RA      UPDATE INSTRUCTION POINTER
     B97D         LD   7D,RB
     AF           RTN             RETURN FROM CALL

                  LBL  SEROUT     SERIAL OUTPUT (WRITE) ROUTINE
017C 76FA10 L17C  CP   FA,"10"    IS SERIAL OUTPUT PORT READY?
     6BFB         JR   Z,L17C     ZERO MEANS NOT READY
     56FAEF       AND  FA,"EF"    AND OUT PORT READY BIT
     99F0         LD   F0,R9      PLACE THE CHARACTER IN I/O PORT
     AF           RTN             RETURN FROM CALL

                  LBL  SERIN      SERIAL INPUT (READ) ROUTINE
0187 76FA08 L187  CP   FA,"08"    IS SERIAL INPUT PORT READY?
     6BFB         JR   Z,L187     ZERO MEANS NOT READY
     56FAF7       AND  FA,"F7"    SET PORT EMPTY BIT
     98F0         LD   R9,F0      READ CHARACTER FROM I/O PORT
     AF           RTN             RETURN FROM CALL

                  LBL  COPYR      COPY WORD AT RSTACK TO STACK
0192 3150         SRP  50
     50E4         POP  R4         POP RETURN TO W4
     50E5         POP  R5
     D60135       CALL R>S        POP RSTACK TO STACK
     D6014D       CALL S>R        PUSH STACK TO RSTACK
     D60111       CALL PUSH8      PUSH W8 TO STACK
     30E4         JP   @R4        RETURN VIA R4

                  LBL  EXNV       EXECUTE NEXT VECTOR
01A3 D60169       CALL LOADNV     LOAD NEXT VECTOR
```

```
Addr Inst  Label Oprn Operands   Comments

     C2A8   L1A6 LDC  RA,@W8     LOAD CODE ADDRESS POINTED TO BY VECTOR INTO WA
     A0E8        INCW W8
     C2B8        LDC  RB,@W8
     A0E8        INCW W8
     897E        LD   7E,R8      UPDATE EXECUTION VECTOR (7E-7F)
     997F        LD   7F,R9
     30EA        JP   @RA        EXECUTE VIA VECTOR LOADED INTO WA

                 LBL  EXWOS      EXECUTE WORD ON STACK
01B4 D600ED      CALL POP8       STACK TO W8
     8BED        JR   L1A6       USE END OF EXNV

                 LBL  COLON      CODE FOR COLON - EXECUTE VECTOR LIST
01B9 707D        PUSH 7D         PUSH INSTRUCTION POINTER TO STACK
     707C        PUSH 7C
     D6014D      CALL S>R        MOVE INSTRUCTION POINTER TO RSTACK
     E47E7C      LD   7C,7E      MOVE EXECUTION VECTOR TO INSTRUCTION POINTER
     E47F7D      LD   7D,7F
     3050        JP   @50

                 LBL  RTN        CODE FOR SEMICOLON - RETURN FROM FORTH WORD
01C8 D60135      CALL R>S        RSTACK TO STACK
     507C        POP  7C         STACK TO INSTRUCTION POINTER
     507D        POP  7D
     3050        JP   @50

                 LBL  KEYSUB     CODE FOR KEY
01D1 3150        SRP  50
     D60187      CALL SERIN      READ SERIAL PORT
     B0E8        CLR  R8         ZERO HIGH BIT OF RESULT
     56E97F      AND  R9,"7F"    ELIMINATE HIGH ORDER BIT OF RESULT
     D60111      CALL PUSH8      PUSH RESULT TO STACK
     3050        JP   @50

                 LBL  EMITSUB
01E0 D600ED      CALL POP8       STACK TO W8
     D6017C      CALL SEROUT     WRITE CHARACTER IN R9 TO SERIAL PORT
     3050        JP   @50

                 LBL  !SUB       CODE FOR !
01E5 D600F9      CALL POPA8      POP ADDRESS TO WA, OBJECT TO W8
     42AA        OR   RA,RA      IS HIGH BYTE OF ADDRESS ZERO?
     6B08        JR   Z,L1F4     IF NOT ZERO STORE IN REGISTER FILE LOCATION
     D28A        LDC  @WA,R8     CODE TO STORE W8 IN RAM - HIGH BYTE TO RAM
     A0EA        INCW WA
     D29A   L1F0 LDC  @WA,R9     LOW BYTE TO RAM
     3050        JP   @50        RETURN
     F5E85B L1F4 LD   @5B,R8     CODE TO STORE R8 IN REG FILE - HIGH BYTE FIRST
     205B        INC  5B         POINT RB TO NEXT BYTE IN REGISTER FILE
```

```
Addr Inst   Label Oprn Operands   Comments

     F5E95B L1F9  LD   @5B,R9     LOW BYTE TO REG FILE
     3050         JP   @50        RETURN

                  LBL  C!SUB      CODE FOR C!
0201 D600F9       CALL POPA8      POP ADDRESS TO WA, OBJECT TO W8
     42AA         OR   R8,R8      IS HIGH BYTE OF ADDRESS ZERO?
     6BF4         JR   Z,L1F9     MAKE USE OF REG FILE PORTION OF !
     8BE9         JR   L1F3       MAKE USE OF RAM PORTION OF !

                  LBL  @SUB       CODE FOR @
020A D600ED       CALL POP8       POP ADDRESS TO W8
     4288         OR   R8,R8      TEST FOR FETCH FROM REG FILE OR RAM
     6B0C         JR   Z,L21D     IF ZERO, FETCH FROM REGISTER FILE
     C2A8         LDC  RA,@W8     FETCH FROM RAM
     A0E8         INCW W8
     C2B8   L215  LDC  RB,@W8
     70EB   L217  PUSH RB         PUSH W8 TO STACK
     70EA         PUSH RA
     3050         JP   @50        RETURN
021D E3A9   L21D  LD   RA,@R9     FETCH FROM REGISTER FILE
     9E           INC  R9
     E3B9   L220  LD   RB,@R9
     8BF3         JR   L217       USE LAST PART OF RAM FETCH TO PUSH TO STACK

                  LBL  C@SUB      CODE FOR C@
0224 D600ED       CALL POP8       POP ADDRESS TO W8
     B0EA         CLR  RA         SET HIGH BYTE OF RESULT TO ZERO
     4288         OR   R8,R8      TEST FOR FETCH FROM REG FILE OR RAM
     6BF3         JR   Z,L220     IF ZERO FETCH FROM REG FILE
     8BE6         JR   L215       USE PORTIONS OF @ FOR ONE-BYTE FETCHES

                  LBL  >RSUB      CODE FOR >R
022F D6014D       CALL S>R        MOVE STACK TO RSTACK
     3050         JP   @50        RETURN

                  LBL  R>SUB      CODE FOR R>
0234 D60135       CALL R>S        MOVE RSTACK TO STACK
     3050         JP   @50        RETURN

                  LBL  NRSUB      CODE FOR THE NUMBER RUNNER
0239 D60169       CALL LOADNV     LOAD VECTOR AT IP TO W8
     D60111       CALL PUSH8      PUSH VECTOR TO STACK
     3050         JP   @50        RETURN

                  LBL  JRSUB      JUMP RELATIVE SUBROUTINE
0241 D60169       CALL LOADNV     LOAD IP INCREMENT TO W8
     807C         DECW 7C         BACK UP IP TO THE INCREMENT
     807C         DECW 7C
     04E97D       ADD  7D,R9      ADD INCREMENT TO INSTRUCTION POINTER
```

```
Addr Inst  Label Oprn Operands   Comments

     14E87C      ADC  7C,R8
     3050        JP   @50        RETURN

                 LBL  JRZSUB     SUBROUTINE TO JUMP RELATIVE IF TOS ZERO
0250 D600ED      CALL POP8       POP STACK TO W8
     4289        OR   R8,R9      TEST WHETHER W8 IS ZERO
     6BEA        JR   Z,L241     IF TOS ZERO, USE ADDRESS TO ADJUST IP
     D60169      CALL LOADNV     OTHERWISE, SKIP OVER THE INCREMENT
     3050        JP   @50

                 LBL  SWAPSUB    CODE FOR SWAP
025C D600F9      CALL POPA8      POP ONE WORD INTO R8, THE OTHER INTO WA
     D6011D      CALL PUSHA8     PUSH WA THEN W8 ONTO THE STACK
     3050        JP   @50        RETURN

                 LBL  DUPSUB     CODE FOR DUP
0264 D600ED      CALL POP8
     D60111      CALL PUSH8
     D60111      CALL PUSH8
     3050        JP   @50

                 LBL  DROPSUB    CODE FOR DROP
026F D600ED      CALL POP8
     3050        JP   @50

                 LBL  ANDSUB     CODE FOR AND
0274 D600F9      CALL POPA8      POP ARGUMENTS INTO W8, WA
     528A        AND  R8,RA
     529B        AND  R9,RB
     D60111      CALL PUSH8      PUSH RESULT ONTO STACK
     3050        JP   @50

                 LBL  ORSUB      CODE FOR OR
0280 D600F9      CALL POPA8      POP ARGUMENTS INTO W8, WA
     428A        OR   R8,RA
     429B        OR   R9,RA
     D60111      CALL PUSH8      PUSH RESULT ONTO STACK
     3050        JP   @50

                 LBL  XORSUB     CODE FOR XOR
028C D600F9      CALL POPA8      POP ARGUMENTS INTO W8,WA
     B28A        XOR  R8,RA
     B29B        XOR  R8,RA
     D60111      CALL PUSH8      PUSH RESULT ONTO STACK
     3050        JP   @50

                 LBL  +SUB       CODE FOR +
0298 D600F9      CALL POPA8      POP ARGUMENTS INTO W8,WA
     029B        ADD  R9,RB
```

```
Addr Inst   Label Oprn Operands   Comments

     128A         ADD  R8,RA
     D60111       CALL PUSH8
     3050         JP   @50

                  LBL  NEGSUB    CODE FOR NEGATE
0298 D600ED       CALL POP8
     60E9         COM  R9
     60E8         COM  R8
     A0E8         INCW W8
     D60111       CALL PUSH8
     3050         JP   @50

                  LBL  MULSUB    CODE FOR UNSIGNED MULTIPLY
02B2 D600F9       CALL POPA8     POP ARGUMENTS INTO W8, WA
     B0EC         CLR  RC        CLEAR PRODUCT REGISTER
     B0ED         CLR  RD
     7C11         LD   R7,"11"   GO THROUGH BIT-LOOP 17 TIMES
     CF           RCF
     C0EC   L2BC  RRC  RC        ROTATE QUAD REGISTER C,D,A,B RIGHT
     C0ED         RRC  RD
     C0EA         RRC  RA
     C0EB         RRC  RB
     FB04         JR   OV,L2C9   IF BIT IN CARRY IS ZERO, ADD IN MULT
     02D9         ADD  RD,R9
     12C8         ADC  RC,R8
     7AF0   L2C9  DJNZ R7,L2BC   TEST FOR END OF BIT-LOOP
     88EC         LD   R8,RC     LOAD W8 WITH HIGH PART OF RESULT (WC)
     98ED         LD   R9,RD
     D6011D       CALL PUSHA8    PUSH BOTH WORDS OF RESULT TO STACK
     3050         JP   @50

                  LBL  DIVSUB    CODE FOR UNSIGNED DIVIDE
02D5 D60105       CALL POPCA8    POP DBL WORD QUOTIENT IN WC/WA, DIVISOR TO W8
     B0EE         CLR  RE        CLEAR DOUBLE WORD RESULT
     B0EF         CLR  RF
     B0E6         CLR  R6
     B0E7         CLR  R7
     B0E5         CLR  R5        CLEAR LEADING BIT COUNT REGISTER
     4C20         LD   R4,"20"   EXAMINE 32 BIT OF QUOTIENT
     CF           RCF
     76EE80 L2E5  TM   RE,"80"   IS THIS THE LEADING BIT OF THE QUOTIENT?
     EB18         JR   NZ,L302   YES - GO DO THE DIVISION
     5E           DEC  R5        NO - ROTATE QUAD REG D, C, F, E, LEFT
     10ED         RLC  RD
     10RC         RLC  RC
     10RF         RLC  RF
     10RE         RLC  RE
     4AEF         DJNZ R4,L2E5   KEEP LOOKING FOR THE LEADING BIT
     ACFF         LD   RA,"FF"   DIDN'T FIND A LEADING BIT - SO LOAD DBL WORD
```

```
Addr Inst   Label Oprn Operands   Comments

     BCFF         LD   RB,"FF"     RESULT WITH -1 AND RETURN
     9CFF         LD   R9,"FF"
     8CFF         LD   R8,"FF"
     D6011D       CALL PUSHA8     PUSH DOUBLE WORD RESULT TO STACK
     3050         JP   @50
0302 5E      L302 INC  R5         INCREMENT BIT COUNT
     CF      L303 RCF
     10E7         RLC  R7
     10E6         RLC  R6
     229D         SUB  R9,RD      SUBTRACT QUAD DENOMINATOR FROM QUAD QUOTIENT
     328C         SBC  R8,RC
     32BF         SBC  RB,RF
     32AE         SBC  RA,RE
     FB0A         JR   NC,L31C
     029D         ADD  R9,RD      ADD QUAD DENOMINATOR TO QUAD QUOTIENT
     128C         ADC  R8,RD
     12BF         ADC  RB,RF
     12AE         ADC  RA,RE
     8B02         JR   L31E
031C A0E6         INCW W6
     CF           RCF
     C0EE         RRC  RE         ROTATE QUAD REG E,F,C,D RIGHT
     C0EF         RRC  RF
     C0EC         RRC  RC
     C0ED         RRC  RD
     5ADA         DJNZ R5,L303
     70E9         PUSH R9         PUSH DOUBLE WORD RESULT TO STACK
     70E8         PUSH R8
     70E7         PUSH R7
     70E6         PUSH R6
     3050         JP   @50        RETURN

                  LBL  0=SUB      CODE FOR 0=
0333 D600ED       CALL POP8
     4289         OR   R8,R9      TEST IF ARGUMENT IS ZERO
     B0E8         CLR  R8         ZERO RESULT REGISTER
     B0E9         CLR  R9
     EB01         JP   NZ,L33F    SKIP SETTING RESULT TO 1 IF ARG WAS ZERO
     9E      L33E INC  R9
     D60111  L33F CALL PUSH8      PUSH RESULT TO STACK
     3050         JP   @50

                  LBL  >=SUB      CODE FOR >=
0344 D600F9       CALL POPA8      POP LEFT ARG TO RA, RIGHT ARG TO R8
     229B         SUB  R9,RB      SUBTRACT RIGHT ARG FROM LEFT ARG
     328A         SBC  R8,RA
     B0E8         CLR  R8         ZERO RESULT REGISTER
     B0E9         CLR  R9
     EBED         JR   UGE,L33E   MAKE USE OF CODE AT END OF 0=
```

```
Addr Inst  Label Oprn Operands   Comments

     8BEC        JR   L33F

                 LBL  CONSUB    LOAD A CONSTANT ON STACK
0353 3150        SRP  50
     A87E        LD   RA,7E     INSTRUCTION POINTER TO WA
     B87F        LD   RB,7F
     C28A        LDC  R8,@WA    LOAD CONSTANT AT WA TO W8
     A0EA        INCW RA
     C29A        LDC  R9,@WA
     8BDE        JR   L33F      EXIT USING LAST PART OF 0= CODE

                 LBL  IP>S      CODE TO PUSH INSTRUCTION POINTER TO STACK
0361 707F        PUSH 7F
     707E        PUSH 7E
     3050        JP   @50

                 LBL  DOSUB     CODE FOR DO
0367 D600F9      CALL POPA8     SWAP
     D6011D      CALL PUSHA8
     D60A4D      CALL S>R       PUSH LIMIT TO STACK
     D6014D      CALL S>R       PUSH INITIAL INDEX TO STACK
     3050        JP   @50

                 LBL  +LOOPSUB  CODE FOR +LOOP (INCREMENT IS ON STACK)
0375 D60135      CALL R>S       MOVE INDEX TO STACK
     D600F9 L378 CALL POPA8     INDEX TO WA, INCREMENT TO W8
     029B        ADD  R9,RB     INDEX + INCREMENT TO W8
     128A        ADC  R8,RA
     D60111      CALL PUSH8     PUSH UPDATED INDEX TO STACK
     D60192      CALL COPYR>S   COPY LIMIT TO STACK
     D600F9      CALL POPA8     LIMIT TO WA, UPDATED INDEX TO R8
     22B9        SUB  RB,R9     SUBTRACT UPDATED INDEX FROM LIMIT
     32A8        SBC  RA,R8
     7B09        JR   MI,L397   MINUS IF UPDATED INDEX EXCEEDS LIMIT
     D60111      CALL PUSH8     PUSH UPDATED INDEX TO STACK
     D6014D      CALL S>R       UPDATED INDEX FROM STACK TO RSTACK
     8D0241      JP   L241      BACK UP TO HEAD OF LOOP
0397 D60135 L397 CALL R>S       EXIT LOOP - MOVE LIMIT FROM RSTACK TO STACK
     D600ED      CALL POP8      POP LIMIT FROM STACK
     D60169      CALL LOADEV    SKIP OVER IP ADDRESS ADJUSTMENT
     3050        JP   @50

                 LBL  LOOPSUB   CODE FOR LOOP
03A2 3150        SRP  50
     9C01        LD   R9,"01"   SET INCREMENT TO 1
     B0E8        CLA  R8
     D60111      CALL PUSH8     PUSH THE INCREMENT ONTO THE STACK
     8BC8        JR   L378      PROCESS THROUGH +LOOP
```

```
Addr Inst  Label Oprn Operands   Comments

                 LBL  ISUB       CODE FOR I
03AD D60192      CALL COPYR>S
     3050        JP   @50

                 LBL  JSUB       CODE FOR J
03B2 D60135      CALL R>S        MOVE INNER INDEX TO STACK
     D60135      CALL R>S        MOVE INNER LIMIT TO STACK
     D60192      CALL COPYR>S    MOVE OUTER INDEX TO STACK
     50E4        POP  R4         POP OUTER INDEX TO W4
     50E5        POP  R5
     D6014D      CALL S>R        MOVE INNER LIMIT BACK TO RSTACK
     D6014D      CALL S>R        MOVE INNER INDEX BACK TO RSTACK
     70E5        PUSH R5         PUSH W4 ONTO THE STACK
     70E4        PUSH R4
     3050        JP   @50

                 LBL  LEAVESUB   CODE FOR LEAVE
03CB D60135      CALL R>S        MOVE INDEX TO STACK
     D600ED      CALL POP8       PURGE INDEX FROM STACK
     D60192      CALL COPYR>S    COPY LIMIT TO STACK
     D6014D      CALL S>R        SET INDEX TO LIMIT

                 LBL  CMOVESUB   CODE FOR CMOVE
03D9 D601105     CALL POPCA8     COUNT TO WC, TARGET ADDR TO WA, SOURCE TO W8
     B0E7   L3DC CLR  R7         CLEAR SOURCE/DESTINATION FLAG
     4288        OR   R8,R8      R8 IS ZERO IF SOURCE IS IN REGFILE
     6B01        JR   Z,L3E5
     7E          INC  R7         SET LOW BIT OF R7 TO INDICATE SOURCE IN RAM
     42AA   L3E5 OR   RA,RA      RA IS ZERO IF DESTINATION IS IN REGFILE
     6B03        JR   Z,L3EA
     46E702      OR   R7,"02"    BIT 1 OF R7 IS SET FOR DESTINATION IN RAM
     4277   L3EA OR   R7,R7      R7 ZERO IF SOURCE AND DEST IN REG FILE
     6B18        JR   Z,Z3F6
     A6E701      CP   R7,"01"    R7 IS 1 FOR SOURCE IN RF, DEST IN RAM
     6B0F        JR   Z,L3F2
     A6E702      CP   R7,"02"    R7 IS 2 FOR SOURCE IN RAM, DEST IN RF
     6B06        JR   Z,L3EE
     C268        LDC  R6,@W8     BOTH SOURCE AND DESTINATION IN RAM
     D26A   L3EA LDC  @WA,R6
     8B0C        JR   L3FA       LOOP EXIT
03EE E369   L3EE LD   R6,@R9     SOURCE IN RF, DESTINATION IN RAM
     8BF8        JR   L3EA       STORE IN RAM
     C268   L3F2 LDC  R6,@W8     SOURCE IN RAM, DESTINATION IN RF
     8B02        JR   L3F8
     E369   L3F6 LD   R6,@R9     BOTH SOURCE AND DEST IN REG FILE
     F3B6   L3F8 LD   @RB,R6     STORE IN REG FILE
     A0E8   L3FA INCW R8         INCREMENT SOURCE ADDRESS
     A0EA        INCW WA         INCREMENT DESTINATION ADDRESS
     80EC        DECW RC         DECREMENT COUNT
```

```
Addr Inst  Label Oprn Operands   Comments

     EBCA        JR   NZ,L3DC   IF NOT ZERO, REPEAT THE LOOP
     3050        JP   @50

                 LBL  HERESUB   CODE FOR HERE
0414 7013        PUSH 13        PUSH HERE (REG FILE 12/13) TO STACK
     7012        PUSH 12
     3050        JP   @50

                 LBL  ,SUB      CODE FOR , (COMMA - COMPILE STACK)
041A D600ED      CALL POP8      POP WORD TO COMPILE OFF STACK
     A812        LD   RA,12     DICTIONARY POINTER TO WA
     B813        LD   RB,13
     D213        LDC  @WA,R8    STORE W8 AT END ON DICTIONARY
     A0EA        INCW WA
0425 D29A   L425 LDC  @WA,R9    STORE LOW BYTE OF W8 AT END OF DICTIONARY
     A0EA        INCW WA        SET DICTIONARY POINTER TO NEXT UNUSED LOCATION
     A912        LD   12,RA     STORE UPDATE DICTIONARY POINTER IN REGFILE 12/13
     B913        LD   13,RB
     3050        JP   @50

                 LBL  C,SUB     CODE FOR C, (C, - COMPILE BYTE TO STACK)
042F D600ED      CALL POP8
     A812        LD   RA,12     DICTIONARY POINTER TO WA
     B813        LD   RB,13
     8BED        JR   L425

                 LBL  H.SUB     CODE TO PRINT BYTE ON STACK
0438 D600ED      CALL POP8      POP WORD OFF TOP OF STACK
     A8E9        LD   RA,R9     TOP BYTE TO RA
     BC04        LD   RB,"04"   DO FOUR CHARACTERS
     98E8   L43F LD   R9,R8
     F0E9        SWAP R9
     56E90F      AND  R9,"0F"   AND OUT LEADING NIBBLE
     06E930      ADD  R9,"30"   MAKE IT A CHARACTER
     A6E93A      CP   R9,"3A"   WAS IT A-F?
     7B03        JR   ULT,L451
     06E907      ADD  R9,"07"   ADD 7 TO MAKE IT A-F
0451 D6017C L451 CALL SEROUT    WRITE THE CHARACTER TO SERIAL OUTPUT PORT
     A6EB03      CP   RB,"03"   IF COUNTER IS THREE GET LOWER BYTE
     EB04        JR   NZ,L45D
     88EA        LD   R8,RA     LOAD LOWER BYTE INTO R8
045B 8B02   L45B JR   L45F
045D F0E8   L45D SWAP R8        SWAP TO MAKE LOW BYTE THE ONE THAT'S WORKED ON
045F BADE   L45F DJNZ RB,L43F   DO ANOTHER NIBBLE
     3050        JP   @50

                 LBL  1+SUB     CODE FOR 1+
0463 D600ED      CALL POP8
     A0E8        INCW W8
```

```
Addr Inst  Label Oprn Operands   Comments

     D60111      CALL PUSH8
     3050        JP   @50

                 LBL  2DROPSUB  CODE FOR 2DROP
046D D600ED      CALL POP8      PRUNE TWO WORDS FROM STACK
     D600ED      CALL POP8
     3050        JP   @50

                 LBL  DLITSUB   CODE FOR DLITERAL
0475 D60169      CALL LOADEV    LOAD HIGH WORD OF DOUBLE WORD
     D60111      CALL PUSH8     PUSH IT ONTO THE STACK
     D60169      CALL LOADEV    LOAD LOW WORD OF DOUBLE WORD
     D60111      CALL PUSH8     PUSH IT ONTO THE STACK
     8D025C L481 JP   L25C      EXIT VIA CODE FOR SWAP

                 LBL  D+SUB     CODE FOR D+
0484 D600F9      CALL POPA8     POP ONE DOUBLE WORD ARGUMENT TO WA,W8
     50E6        POP  R6        POP THE OTHER TO W6,W4
     50E7        POP  R7
     50E4        POP  R4
     50E5        POP  R5
     0295        ADD  R9,R5     ADD W6,W4 TO WA,W8
     1284        ADC  R8,R4
     12B7        ADC  RB,R7
     12A6        ADC  RA,R6
     D6011D L497 CALL PUSHA8    PUSH WA THEN W8 ONTO STACK
     8BE5        JR   L481      EXIT VIA CODE FOR SWAP

                 LBL  DNEGSUB   CODE FOR DNEGATE
049C D600F9      CALL POPA8     POP THE DOUBLE WORD ARGUENT TO WA,W8
     60E9        COM  R9
     60E8        COM  R8
     60EB        COM  RB
     60EA        COM  RA
     06E901      ADD  R9,"01"   ADD ONE FOR TWOS COMPLEMENT
     16E800      ADC  R8,"00"
     16EB00      ADC  RB,"00"
     16EB00      ADC  RA,"00"
     8BE2        JR   L497      EXIT VIA CODE FOR D+

                 LBL  DOES_SUB  CODE FOR DOES
04B5 707D        PUSH 7D        PUSH EXECUTION VECTOR ONTO THE STACK
     707C        PUSH 7C
     D6014D      CALL S>R       PUSH EXECUTION VECTOR TO RSTACK
     3170        SRP  70
     C2CE        LD   RC,@WE    LOAD EXECUTION VECTOR @IP
     A0EE        INCW RE
     C2DE        LD   RD,@WE
     A0EE        INCW WE
```

```
Addr Inst  Label Oprn Operands   Comments

     70EF        PUSH RF        PUSH IP TO STACK
     70EE        PUSH RE
     3050        JP   @50

                 LBL  WORDAUX   SUBROUTINE FOR WORD CODE
04CC 68E8        LD   R6,R8     BUFFER ADDRESS TO W6
     78E9        LD   R7,R9
     0415E7      ADD  R7,15     ADD >IN TO BUFFER ADDRESS
     1414E6      ADC  R6,14
     C2A6        LDC  RA,@W6    FETCH CHARACTER FROM BUFFER TO RA
     42AA        OR   RA,RA     TEST WHETHER ITS ZERO
     AF          RTN

                 LBL  WORDSUB   CODE FOR WORD
04DB D600F9      CALL POPA8     FENCE TO RB, BUFFER ADDRESS TO W8
04DE D604CC L4DE CALL WORDAUX   SKIP TO FIRST OCCURRENCE OF FENCE
     6B2C        JR   Z,L50F    ZERO IF END OF BUFFER
     A2AB        CP   RA,RB     TEST CHARACTER AGAINST FENCE
04E5 EB04        JR   NZ,L4ED   NON-ZERO IF CHAR WAS NOT THE FENCE
     A014        INCW 14        INCREMENT OFFSET
04E9 8BF3        JR   L4DE
04EB C8E6        LD   RC,R6     LOAD CHARACTER ADDRESS TO WD
04ED D8E7   L4ED LD   RD,R7
04EF A014   L4EF INCW 14        INCREMENT >IN
     D60RCC      CALL WORDAUX   GET ANOTHER
     6B19        JR   Z,L50D
     A2AB        CP   RA,RB     COMPARE CHAR WITH FENCE
     EBF5        JR   NZ,L4EF
     88E6        LD   R8,R6     CHAR ADDRESS TO W8
     98E7        LD   R9,R7
     229D        SUB  R9,RD     LENGTH OF WORD TO W8
0500 328C        SBC  R8,RC
0502 A812   L502 LD   RA,12     DIRECTORY POINTER (TARGET) TO WA
     B813        LD   RB,13
     D29A        LDC  @WA,R9    LENGTH BYTE TO DICTIONARY
     A0EA        INCW WA        POINT TO DESTINATION FOR TEXT
     D60129      CALL PUSHCA8   FROM ADDR IN WC, TO IN WA, COUNT IN W8
     3050   L50D JP   @50
050F B014   L50F CLR  14        CASE WHEN BUFFER STARTS WITH ZERO
     B015        CLR  15          CLEAR >IN
     C8E6        LD   RC,R6     CHARACTER ADDRESS TO WC
     D8E7        LD   RD,R7
     B0E8        CLR  R8        SET CHARACTER COUNT TO 1
     9C01        LD   R9,"01"
051B 8BE5        JR   L502

                 LBL  FINDCC    COMPARE CHARACTERS ROUTINE (FOR FIND)
051D C2C8        LDC  RC,@W8    GET CHARACTER FROM DIRECTORY
     C2DA        LDC  RC,@WA    GET CHARACTER FROM WORD
```

```
Addr Inst   Label Oprn Operands   Comments

     52C7           AND  RC,R7     AND OUT CHARACTERS TO TEST
     52D7           AND  RD,R7
     AA2CD          CP   RC,RD
     AF             RTN

                    LBL  FINDSB    FIND STOP BIT ROUTINE (FOR FIND)
0528 C2D8           LDC  RD,@W8    GET THE CHARACTER FROM WORD
     56ED80         AND  RD,"80"   GET THE STOP BIT
     AF             RTN

052E D600F9         CALL POPA8     ADDRESS OF WORD TO RA, DICTIONARY IN R8
     4288           OR   R8,R8     DICTIONARY ADDRESS IN REG FILE?
     EB0B           JR   NZ,L54D   NZ IF DICTIONARY ADDRESS EXPLICIT
     E559EC         LD   RC,@R9    IF DICT ARG IN REG FILE ITS A POINTER
     9E             INC  R9         LOAD POINTER TO WC
     E559ED         LD   RD,@R9
     88EC           LD   R8,RC     NOW LOAD THE DICTIONARY ADDRESS
     98ED           LD   R9,RD
0540 7C1F   L540    LD   R7,"1F"   TEST EQUALITY OF WORD COUNTS
0542 D6051D L542    CALL FINDCC
     EB20           JR   NZ,L562   IF NOT EQUAL GO GET NEXT DICTIONARY ENTRY
     A0E8   L547    INCW W8        POINT TO NEXT BYTE IN DICTIONARY
     A0EA           INCW WA        POINT TO NEXT BYTE IN WORD
     7C7F           LD   R7,"7F"   SET UP TO TEST THE ASCII CHARACTERS
     D6051D         CALL FINDCC
     EB10           JR   NZ,L562   IF NOT EQUAL GO GET NEXT DICTIONARY ENTRY
     D60528         CALL FINDSB    CHECK IF THE DICTIONARY ENTRY STOP BIT IS SET
     6BF0           JR   EQ,L547   IF NOT SET GO CHECK NEXT CHAR IN DICTIONARY
     A0E8           INCW R8        DICTIONARY ENTRY MATCHES WORD - POINT TO
     A0E8           INCW R8          PARAMETER PART OF DICTIONARY ENTRY
     A0E8           INCW R8
     D60111         CALL PUSH8     PUSH ADDRESS OF PARAMETER ENTRY (OR ZERO)
     3050           JP   @50         ONTO STACK AND RETURN
0562 D60528 L562    CALL FINDSB    CHAR OF ENTRY DIDN'T MATCH - IS SEARCH AT THE
     EB04           JR   NZ,L56B     END OF THE DICTIONARY ENTRY?
     A0E8           INCW W8        IF NOT, LOOK AT THE NEXT CHAR IN THE DICTIONARY
     8BF7           JR   L562
056B A0E8           INCW W8        DICTIONARY ENTRY DIDN'T MATCH - GO TO NEXT ENTRY
     C2C8           LDC  RC,@W8    LOAD LINK TO NEXT ENTRY INTO WC
     A0E8           INCW W8
     C2D8           LDC  RD,@W8
     88EC           LD   R8,RC     LOAD ADDRESS OF DICTIONARY ENTRY TO W8
     98ED           LD   R9,RD
     42CD           OR   RC,RD     IS THE LINK ADDRESS ZERO?
     6BE2           JR   EQ,L55D
     A812           LD   RA,12     RESTORE ADDRESS OF WORD INTO WA
     B813           LD   RB,13
     8BBF           JR   L540
```

```
Addr Inst  Label Oprn Operands   Comments

                 LBL  ."SUB      CODE TO EXECUTE DOT-QUOTE
0581 3150        SRP  50
     B87D        LD   RB,7D      LOAD INSTRUCTION POINTER INTO RA
     A87C        LD   RA,7C
     C28A        LDC  R8,@RA     LOAD CHARACTER COUNT FROM TEXT
     A0EA        INCW WA
     C29A   L589 LDC  R9,@WA     LOAD SUCCEEDING CHARACTER FROM TEXT
     D6017C      CALL SEROUT     WRITE CHARACTER TO SERIAL OUTPUT PORT
     8AF7        DJNZ R8,L589
     A0EA        INCW WA
     B97D        LD   7D,RB      RESTORE IP FROM WA
     A97C        LD   7C,RA
     3050        JR   @50
```

APPENDIX B

Headless words (written as conventional FORTH definitions):

```
     ( Addr 608 )
: ERRMSG WARNING @ DUP IF EXECUTE ELSE DROP CR HERE COUNT 1F AND
     TYPE SPACE K3F EMIT EMIT CR ;

     ( Addr 636 )
: FATALERR ERRMSG BELL EMIT CR QUIT ;

     ( Addr 644 )
: MSHOOK BLK @ DUP 400 U* IF DROP EXECUTE ^; THEN SWAP ;

     ( Addr 662 )
: NUCLEUS BEGIN BEGIN BEGIN BEGIN FIND DUP IF [ ROT ROT ] DUP K2 - K1 - BEGIN
     K1 - DUP C@ K80 AND UNTIL C@ STATE @ >= IF EXECUTE [ SWAP ] AGAIN THEN ,
     AGAIN THEN DROP HERE NUMBER 10 @ 1+ IF [COMPILE] DLITERAL [ SWAP ] AGAIN
     THEN DROP [COMPILE] LITERAL AGAIN ;

     ( Addr 6BE )
: RELADR HERE - , ;
```

Interpreted FORTH words:

```
     ( Addr 73D )                ( Addr 7DA )
: - NEGATE + ;                : = - 0= ;

     ( Addr 7E6 )                ( Addr 7CE )              ( Addr 80A )
: OVER >R DUP R> SWAP ;       : ROT >R SWAP R> SWAP ;    : SPACE K20 EMIT :

     ( Addr 818 )
: EXPECT OVER + OVER DO KEY DUP 8 = IF OVER I = IF K1 - EMIT K0 ELSE R> SWAP
     DUP EMIT SPACE EMIT K2 - >R [ SWAP ] ELSE DUP KD = IF DROP I OVER OVER C!
     1+ K0 OVER C! K20 [ SWAP ] THEN LEAVE [ SWAP ] THEN DUP I C! THEN EMIT
     THEN K1 LOOP DROP ;

     ( Addr  89E )
: +! SWAP @ + SWAP ! ;

     ( Addr  8B3 )
: | [ null ]  [ 0 HERE 4 - C! ] BLK @ IF K1 BLK +! >IN @ K0 >IN ! 3FF = IF
     STATE @ K0 = IF 74C FATALERR [ ROT ROT ] THEN THEN R> DROP ; IMMEDIATE

     ( Addr 8F5 )
: CR KD EMIT KA EMIT ;
```

---

[6]FORTH words beginning with a caret (^) are addresses of routines in the assembly section of the system software. FORTH words consisting of a hex number preceded by the letter K (K1, K2, KC0) are constants loaded from the constant block (addresses 5A8-5FF).

```
    ( Addr 906 )
: WORD HERE K20 OVER C! DUP 1+ K20 CMOVE BLK @ IF MSHOOK ELSE TIB @ SWAP
    WORDSUB CMOVE HERE ;

    ( Addr 93A)
: COUNT DUP >R 1+ R> C@ ;

    ( Addr 94E )
: TYPE K1 - K0 DO I OVER + C@ EMIT LOOP DROP ;

    ( Addr 970 )
: FIND K20 WORD CURNT @ SWAP FINDSUB DUP
    IF ^; THEN DROP CONTXT @ HERE FINDSUB ;

    ( Addr 9A7 )
: COMPILE STATE @ IF R> DUP K2 + >R @ , ELSE 143 FATALERR THEN ;

    ( Addr 9CD )
: LITERAL STATE @ IF [COMPILE] COMPILE ^NR , ; IMMEDIATE

    ( Addr 9E5 )
: DLITERAL STATE @ IF COM ^DLIT , , THEN ; IMMEDIATE

    ( Addr 9FF )
: DIGIT 30 - DUP K0 >= IF 7 - DUP KA >= ELSE/IF DUP BASE @ 1+ THEN >= ELSE/IF
    DROP ^; THEN THEN K1 ;

    ( Addr A49 )
: NUMBER K0 K0 ROT DUP 1+ C@ 2DROP = DUP >R + -1 10 ! 1+ DUP >R C@ DIGIT WHILE
    SWAP BASE @ U* DROP ROT BASE @ U* D+ 10 @ 1+ IF K1 10 +! THEN AGAIN DUP C@
    K20 - WHILE DUP C@ 2E - IF R> 24E FATALERR THEN K0 REPEAT DROP R> IF
   DNEGATE THEN ;

    ( Addr AE5 )
: QUIT S0 @ FE ! K0 BLK ! K0 STATE ! R0 @ 7A ! BEGIN CR BEGIN TIB @ K0 OVER C!
    DUP 1+ 40 CMOVE TIB @ 40 EXPECT NUCLEUS STATE @ 0= [ SWAP ] UNTIL ." OK"
    AGAIN ;

    ( Addr B48 )                    ( Addr B56 )
: [ K0 STATE ! ; IMMEDIATE        : ] C0 STATE ! ; IMMEDIATE

    ( Addr B66 )
: ' FIND DUP IF K2 + LITERAL THEN;

    ( Addr B82 )                                ( Addr B9A )
: MIN OVER OVER >= IF SWAP THEN DROP ;        : ALLOT 12 +! ;

    ( Addr BAA )
: CREATE FIND IF 55 FATALERR THEN HERE DUP @ WIDTH @ MIN ALLOT A0 OVER C@ OR
    OVER C! HERE K80 OVER C@ OR SWAP C! K1 ALLOT VOCLNK @ , VOCLNK ! K0 HERE
    ! ;

    ( Addr BFE )
: : STATE @ IF 444 FATALERR THEN STKPTR @ STKBAL ! CURNT @ CONTXT ! CREATE ]
    [COMPILE] COMPILE ^: ; IMMEDIATE

    ( Addr C2C )
: SMUDGE CURNT @ @ DF OVER C@ AND SWAP C! ;
```

```
    ( Addr C4A )
: ; STKPTR @ STKBAL @ - IF 353 FATALERR THEN [COMPILE] COMPILE ^; SMUDGE
    [ ; IMMEDIATE

    ( Addr C6E )                                    ( Addr C84 )
: CONSTANT CREATE ^CONST , , SMUDGE ;          : <BUILDS K0 CONSTANT ;

    ( Addr C92 )
: DOES> CURNT @ @ BEGIN 1+ DUP C@ K80 AND UNTIL 3 + 4B5 OVER ! K2 + R> SWAP ! ;

    ( Addr CC8 )
: IMMEDIATE CURRENT @ @ 40 OVR C@ OR SWAP C! ;

    ( Addr CE6 )
: ." K1 >IN +! [COMPILE] ^." 22 WORD C@ 1+ ALLOT K1 >IN +! ; IMMEDIATE

    ( Addr D0B )            ( Addr D1D )                    ( Addr D35 )
: PAD HERE 44 + ;         : HOLD HLD OVER +! @ C! ;         : <# PAD HOLD ! ;

    ( Addr D44 )
: # BASE @ >R K0 I U/ R> SWAP >R U/ R> ROT DUP KA >= IF 7 + THEN 30 + HOLD ;

    ( Addr D7C )                                    ( Addr D93 )
: #S # OVER OVER OR 0= IF [-C SWAP ! ] ;        : #> 2DROP HLD @ PAD OVER - ;

    ( Addr DA8 )
: SIGN ROT 8000 >= IF 2DROP HOLD ELSE ;

    ( Addr DC4 )
: DABS DUP 8000 >= IF DNEGATE THEN ;

    ( Addr DDC )
: D. SWAP OVER DABS <# #S SIGN #> TYPE ;

    ( Addr DF5 )
: S-> K0 OVER 8000 >= IF K1 - THEN ;

    ( Addr E11 )
: DO [COMPILE] COMPILE DOSUB HERE ; IMMEDIATE

    ( Addr E20 )
: +LOOP [COMPILE] COMPILE ^+LOOP RELADR ; IMMEDIATE

    ( Addr E30 )
: LOOP [COMPILE] COMPILE ^LOOP RELADR ; IMMEDIATE

    ( Addr E40 )
: IF [COMPILE] COMPILE ^JRZ HERE K0 , ; IMMEDIATE

    ( Addr E53 )
: THEN HERE OVER - SWAP ! ; IMMEDIATE

    ( Addr E67 )
: ELSE [COMPILE] COMPILE ^JR HERE K0 , SWAP [COMPILE] THEN ; IMMEDIATE

    ( Addr E7F )
: BEGIN HERE ; IMMEDIATE

    ( Addr E8B )
: UNTIL [COMPILE] COMPILE ^JRZ RELADR ; IMMEDIATE
```

```
    ( Addr E9B )
: AGAIN [COMPILE] COMPILE ^JR RELADR ; IMMEDIATE

    ( Addr EAB )
: WHILE [COMPILE] IF ; IMMEDIATE

    ( Addr EB7 )
: REPEAT >R [COMPILE] AGAIN R> [COMPILE] THEN ; IMMEDIATE

    ( Addr EC9 )
: CASE CASE# @ STKPTR @ CASE# ! ; IMMEDIATE

    ( Addr EDF )
: OF COMPILE OVER COMPILE = [COMPILE] IF COMPILE DROP ; IMMEDIATE

    ( Addr EF6 )
: ENDCASE BEGIN STKPTR @ CASE# @ - WHILE [COMPILE] THEN REPEAT CASE# ! ;
    IMMEDIATE

    ( Addr F18 )
: FORGET FIND DUP IF CURNT @ CONTXT @ - ELSE/IF DROP 656 FATALERR THEN K2 - DUP
    @ CURNT @ ! K1 K0 DO BEGIN K1 - DUP C@ K80 AND UNTIL LOOP 12 ! ; IMMEDIATE
```

APPENDIX C

ZILOG Z8 FORTH SYSTEM
4K ROM MEMORY DUMP

```
Addr    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F    Comment

000  < 00 A9 00 AB 00 AD 00 AF 00 B1 00 B3 8F 31 50 EC > First instruction: 00C
010  < 00 FC B5 DC F0 CC 10 C3 DE CA FC EC 00 FC C5 DC > Initialize F0-FF
020  < 10 CC 20 C3 DE CA FC 31 70 AC 00 BC 00 06 EA 04 > Initialize 10-2F
030  < C2 7A 68 E7 60 E6 D2 6A C2 6A D2 7A A2 67 6B ED > Find limits of ROM
040  < C8 EA D8 EB C9 12 D9 13 06 EA 04 C2 7A 68 E7 60 >
050  < E6 D2 6A C2 6A D2 7A A2 67 EB ED A9 22 B9 23 00 > Find limits of RAM
060  < EA A9 20 B9 21 A9 1A B9 1B 80 EC C2 7C 80 EC C2 > Find starting address
070  < 6C A0 E6 EB 04 CC 0F DC FE 31 50 8C FF 9C FD C2 > Get baud rate switches
080  < 78 8D 0F 84 8C 00 9C E5 02 97 16 E8 00 C2 78 79 > Initialize shared-bus
090  < F4 E6 F1 03 E4 20 FE E4 21 FF E4 22 7A E4 23 7B > Initialize stack adds
0A0  < E6 50 01 E6 51 A3 9F 30 50 30 04 30 06 30 08 30 > Initialize NEXT vector
0B0  < 0A 30 0C 30 0E 20 00 F0 03 10 0F FF 41 B2 2B 10 > Constants for F0-FF
0C0  < 00 00 50 1F 00 00 00 10 00 00 00 00 00 00 10 00 > Constants for 10-2F
0D0  < 00 00 00 00 00 00 00 00 00 00 03 00 00 00 2C 00 >
0E0  < 2C 0F 6E 0F FF 80 01 02 04 08 10 AF 40 31 50 50 > Baud rate table
0F0  < EE 50 EF 50 E8 50 E9 30 5E 31 50 50 EE 50 EF 50 > Stack routines: POP8
100  < EA 50 EB 8B EE 31 50 50 EE 50 EF 50 EC 50 ED 8B >   POPA8   POPCA8
110  < EE 31 50 50 EE 50 EF 70 E9 70 E8 30 5E 31 50 50 >   PUSH8   PUSHA8
120  < EE 50 EF 70 EB 70 EA 8B EE 31 50 50 EE 50 EF 70 >   PUSHCA8
130  < ED 70 EC 8B EE 31 50 50 EE 50 EF 68 7A 78 7B C2 >   RSTACK > STACK
140  < 86 A0 E6 C2 96 A0 E6 79 7B 69 7A 8B CA 31 50 50 >   STACK  > RSTACK
150  < EE 50 EF 50 E8 50 E9 68 7A 78 7B 80 E6 D2 96 80 >
160  < E6 D2 86 69 7A 79 7B 30 EE 31 50 A8 7C B8 7D C2 > Load next vector to W8
170  < 8A A0 EA C2 9A A0 EA A9 7C B9 7D AF 76 FA 10 6B > Serial output
180  < FB 56 FA EF 99 F0 AF 76 FA 08 6B FB 56 FA F7 98 > Serial input
190  < F0 AF 31 50 50 E4 50 E5 D6 01 35 D6 01 4D D6 01 > Copy RSTACK to STACK
1A0  < 11 30 E4 D6 01 69 C2 A8 A0 E8 C2 B8 A0 E8 89 7E > Execute next vector
1B0  < 99 7F 30 EA D6 00 ED 8B ED 70 7D 70 7C D6 01 4D > Execute word on stack
1C0  < E4 7E 7C E4 7F 7D 30 50 D6 01 35 50 7C 50 7D 30 > Code for ":" and ";"
1D0  < 50 31 50 D6 01 87 B0 E8 56 E9 7F D6 01 11 30 50 > Code for KEY
1E0  < D6 00 ED D6 01 7C 30 50 D6 00 F9 42 AA 6B 08 D2 > Code for EMIT and "!"
1F0  < 8A A0 EA D2 9A 30 50 F5 E8 5B 20 5B F5 E9 5B 30 >
200  < 50 D6 00 F9 42 AA 6B F4 8B E9 D6 00 ED 42 88 6B > Code for C! and "@"
210  < 0C C2 A8 A0 E8 C2 B8 70 EB 70 EA 30 50 E3 A9 9E >
220  < E3 B9 8B F3 D6 00 ED B0 EA 42 88 6B F3 8B E6 D6 > Code for C@ and >R
230  < 01 4D 30 50 D6 01 35 30 50 D6 01 69 D6 01 11 30 > Code for R> No. runner
240  < 50 D6 01 69 80 7C 80 7C 04 E9 7D 14 E8 7C 30 50 > Jump relative
250  < D6 00 ED 42 89 6B EA D6 01 69 30 50 D6 00 F9 D6 > Cond. jump relative
260  < 01 1D 30 50 D6 00 ED D6 01 11 D6 01 11 30 50 D6 > Code for SWAP and DUP
270  < 00 ED 30 50 D6 00 F9 52 8A 52 9B D6 01 11 30 50 > Code for DROP and AND
280  < D6 00 F9 42 8A 42 9B D6 01 11 30 50 D6 00 F9 B2 > Code for OR and XOR
290  < 8A B2 9B D6 01 11 30 50 D6 00 F9 02 9B 12 8A D6 > Code for + and NEGATE
2A0  < 01 11 30 50 D6 00 ED 60 E9 60 E8 A0 E8 D6 01 11 > Code for U* (Unsigned
2B0  < 30 50 D6 00 F9 B0 EC B0 ED 7C 11 CF C0 EC C0 ED >    multiply)
```

```
Addr    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F    Comment

2C0  < C0 EA C0 EB FB 04 02 D9 12 C8 7A F0 88 EC 98 ED >
2D0  < D6 01 1D 30 50 D6 01 05 B0 EE B0 EF B0 E6 B0 E7 > Code for U/ (Unsigned
2E0  < B0 E5 4C 20 CF 76 EE 80 EB 18 5E 10 ED 10 EC 10 >    divide)
2F0  < EF 10 EE 4A EF AC FF BC FF 9C FF 8C FF D6 01 1D >
300  < 30 50 5E CF 10 E7 10 E6 22 9D 32 8C 32 BF 32 AE >
310  < FB 0A 02 9D 12 8C 12 BF 12 AE 8B 02 A0 E6 CF C0 >
320  < EE C0 EF C0 EC C0 ED 5A DA 70 E9 70 E8 70 E7 70 >
330  < E6 30 50 D6 00 ED 42 89 B0 E8 B0 E9 EB 01 9E D6 > Code for 0=
340  < 01 11 30 50 D6 00 F9 22 9B 32 8A B0 E8 B0 E9 FB > Code for >=
350  < ED 8B EC 31 50 A8 7E B8 7F C2 8A A0 EA C2 9A 8B > Load constant on stack
360  < DE 70 7F 70 7E 30 50 D6 00 F9 D6 01 1D D6 01 4D > Push IP onto stack
370  < D6 01 4D 30 50 D6 01 35 D6 00 F9 02 9B 12 8A D6 > Code for +LOOP
380  < 01 11 D6 01 92 D6 00 F9 22 B9 32 A8 7B 09 D6 01 >
390  < 11 D6 01 4D 8D 02 41 D6 01 35 D6 00 ED D6 01 69 >
3A0  < 30 50 31 50 9C 01 B0 E8 D6 01 11 8B C8 D6 01 92 > Code for LOOP and "I"
3B0  < 30 50 D6 01 35 D6 01 35 D6 01 92 50 E4 50 E5 D6 > Code for "J"
3C0  < 01 4D D6 01 4D 70 E5 70 E4 30 50 D6 01 35 D6 00 > Code for LEAVE
3D0  < ED D6 01 92 D6 01 4D 30 50 D6 01 05 B0 E7 42 88 > Code for CMOVE
3E0  < 6B 01 7E 42 AA 6B 03 46 E7 02 42 77 6B 18 A6 E7 >
3F0  < 01 6B 0F A6 E7 02 6B 06 C2 68 D2 6A 8B 0C E3 69 >
400  < 8B F8 C2 68 8B 02 E3 69 F3 B6 A0 E8 A0 EA 80 EC >
410  < EB CA 30 50 70 13 70 12 30 50 D6 00 ED A8 12 B8 > Code for HERE
420  < 13 D2 8A A0 EA D2 9A A0 EA A9 12 B9 13 30 50 D6 > Code for C,
430  < 00 ED A8 12 B8 13 8B ED D6 00 ED A8 E9 BC 04 98 > Code for H.
440  < E8 F0 E9 56 E9 0F 06 E9 30 A6 E9 3A 7B 03 06 E9 >
450  < 07 D6 01 7C A6 EB 03 EB 04 88 EA 8B 02 F0 E8 BA >
460  < DE 30 50 D6 00 ED A0 E8 D6 01 11 30 50 D6 00 ED > Code for 1+ and 2DROP
470  < D6 00 ED 30 50 D6 01 69 D6 01 11 D6 01 69 D6 01 > Code for DLITERAL
480  < 11 8D 02 5C D6 00 F9 50 E6 50 E7 50 E4 50 E5 02 > Code for D+
490  < 95 12 84 12 B7 12 A6 D6 01 1D 8B E5 D6 00 F9 60 > Code for DNEGATE
4A0  < E9 60 E8 60 EB 60 EA 06 E9 01 16 E8 00 16 EB 00 >
4B0  < 16 EA 00 8B E2 70 7D 70 7C D6 01 4D 31 70 C2 CE > Code for <DOES
4C0  < A0 EE C2 DE A0 EE 70 EF 70 EE 30 50 68 E8 78 E9 > Subroutine for WORD
4D0  < 04 15 E7 14 14 E6 C2 A6 42 AA AF D6 00 F9 D6 04 > Code for WORD
4E0  < CC 6B 2C A2 AB EB 04 A0 14 8B F3 C8 E6 D8 E7 A0 >
4F0  < 14 D6 04 CC 6B 19 A2 AB EB F5 88 E6 98 E7 22 9D >
500  < 32 8C A8 12 B8 13 D2 9A A0 EA D6 01 29 30 50 B0 >
510  < 14 B0 15 C8 E6 D8 E7 B0 E8 9C 01 8B E5 C2 C8 C2 >
520  < DA 52 C7 52 D7 A2 CD AF C2 D8 56 ED 80 AF D6 00 > Code for FIND
530  < F9 42 88 EB 0B E5 59 EC 9E E5 59 ED 88 EC 98 ED >
540  < 7C 1F D6 05 1D EB 20 A0 E8 A0 EA 7C 7F D6 05 1D >
550  < EB 10 D6 05 28 6B F0 A0 E8 A0 E8 A0 E8 D6 01 11 >
560  < 30 50 D6 05 28 EB 04 A0 E8 8B F7 A0 E8 C2 C8 A0 >
570  < E8 C2 D8 88 EC 98 ED 42 CD 6B E2 A8 12 B8 13 8B > Code for ."
580  < BF 31 50 B8 7D A8 7C C2 8A A0 EA C2 9A D6 01 7C >
590  < 8A F7 A0 EA B9 7D A9 7C 30 50 02 39 02 41 02 50 > Execution vectors
5A0  < 03 67 03 75 03 A2 05 81 03 53 00 1A 03 53 00 1C > Load constant to stack
5B0  < 03 53 00 1E 03 53 00 20 03 53 00 22 03 53 00 24 >
```

```
Addr    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F    Comment

5C0  < 03 53 00 26 03 53 00 28 03 53 00 2A 03 53 00 2C >
5D0  < 03 53 00 00 03 53 00 01 03 53 00 02 03 53 00 0A >
5E0  < 03 53 00 0D 03 53 00 0F 03 53 00 20 03 53 00 3F >
5F0  < 03 53 00 80 03 53 00 FE 03 53 00 78 03 53 00 76 >
600  < 01 C8 04 75 05 2E 04 DB 01 B9 05 C0 06 F1 07 16 > Execution vectors
610  < 05 9E 00 08 06 CE 05 9C 00 1C 07 1E 08 FA 07 83 > FORTH headless code:
620  < 09 40 05 9A 00 1F 07 26 09 54 08 10 05 EC 06 DE >  ERRMSG
630  < 06 DE 08 FA 06 00 01 B9 06 08 05 9A 00 07 06 DE >  FATALERR
640  < 08 FA 0A EB 01 B9 07 C0 06 F1 07 16 05 9A 04 00 >  MSHOOK (BLK) Rooutine
650  < 07 4A 05 9E 00 08 07 1E 06 CE 06 00 07 0E 07 1E >
660  < 06 00 01 B9 09 76 07 16 05 9E 00 34 07 16 05 D8 >  NUCLEUS
670  < 07 D2 05 D4 07 D2 05 D4 07 D2 07 16 06 F8 05 F0 >
680  < 07 26 05 9E FF F2 06 F8 05 AC 06 F1 07 5F 05 9E >
690  < 00 08 06 CE 05 9C FF CE 07 89 05 9C FF C8 07 1E >
6A0  < 07 83 0A 4F 05 9A 00 10 06 F1 07 A6 05 9E 00 08 >
6B0  < 09 EB 05 9C FF B0 07 1E 09 D3 05 9C FF A8 01 B9 >  RELADR
6C0  < 07 83 07 D2 07 89 06 00 87 45 58 C5 00 00 01 B4 >  Headed FORTH words:
6D0  < 83 4B 45 D9 06 C8 01 D1 84 45 4D C9 06 D0 01 E0 >   EXECUTE KEY EMIT
6E0  < 81 A1 06 D8 01 E8 82 43 A1 06 E0 02 01 81 C0 06 >   C!      @
6F0  < E6 02 0A 82 43 C0 06 ED 02 24 82 3E D2 06 F3 02 >   C@      >R
700  < 2F 82 52 BE 06 FA 02 34 84 53 57 C1 07 01 02 5C >   R>      SWAP
710  < 83 44 55 D0 07 08 02 64 84 44 52 CF 07 10 02 6F >   DUP     DROP
720  < 83 41 4E C4 07 18 02 74 82 4F D2 07 20 02 80 83 >   AND     OR
730  < 58 4F D2 07 28 02 8C 81 AB 07 2F 02 98 86 4E 45 >   XOR     NEG
740  < C7 07 37 02 A4 82 55 AA 07 3D 02 B2 82 55 AF 07 >   U*      U/
750  < 45 02 D5 82 30 BD 07 4C 03 33 82 3E BD 07 53 03 >   0=      >=
760  < 44 81 C9 07 5A 03 AD 81 CA 07 61 03 B2 85 4C 45 >   I       J
770  < C1 07 67 03 CB 85 43 4D CF 07 6D 03 D9 84 48 45 >   CMOVE   HERE
780  < D2 07 75 04 14 81 AC 07 7D 04 1A 82 43 AC 07 85 >   ,       C,
790  < 04 2F 82 48 AE 07 8B 04 38 84 43 4F CC 07 92 00 >   H.      COLD
7A0  < 0C 82 31 AB 07 99 04 63 85 32 44 D2 07 A1 04 6D >   1+      2DROP
7B0  < 83 3E 49 CE 07 A8 03 53 00 14 83 42 4C CB 07 B0 >   >IN     BLK
7C0  < 03 53 00 16 84 42 41 D3 07 BA 03 53 00 18 81 AD >   BASE    -
7D0  < 07 C4 01 B9 07 43 07 3B 06 00 81 BD 07 CE 01 B9 >   ;       =
7E0  < 07 D2 07 58 06 00 84 4F 56 C5 07 DA 01 B9 06 FF >   OVER
7F0  < 07 16 07 06 07 0E 06 00 83 52 4F D4 07 E6 01 B9 >   ROT
800  < 06 FF 07 0E 07 06 07 0E 06 00 85 53 50 C1 07 F8 >   SPACE
810  < 01 B9 05 E8 06 DE 06 00 86 45 58 D0 08 0A 01 B9 >   EXPECT
820  < 07 EC 07 3B 07 EC 05 A0 06 D6 07 16 05 9A 00 08 >
830  < 07 DE 05 9E 00 2E 07 EC 07 65 07 DE 05 9E 00 0E >
840  < 05 D4 07 D2 06 DE 05 D0 05 9C 00 4C 07 06 07 0E >
850  < 07 16 06 DE 08 10 06 DE 05 D8 07 D2 06 FF 05 9C >
860  < 00 34 07 16 05 E0 07 DE 05 9E 00 22 07 1E 05 E8 >
870  < 07 65 07 EC 07 EC 06 EB 07 A6 05 D0 07 EC 06 EB >
880  < 07 A6 06 EB 05 E8 07 73 05 9C 00 08 07 16 07 65 >
890  < 06 EB 06 DE 05 D4 05 A2 FF 90 07 1E 06 00 82 2B >   +!
8A0  < A1 08 18 01 B9 07 0E 07 EC 06 F1 07 3B 07 0E 06 >
8B0  < E4 06 00 C1 80 08 9E 01 B9 07 C0 06 F1 05 9E 00 >   (NULL)
```

```
Addr    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F    Comment

8C0  < 30 05 D4 07 C0 08 A3 07 B6 06 F1 05 D0 07 B6 06 >
8D0  < E4 05 9A 03 FF 07 DE 05 9E 00 1A 05 AC 06 F1 05 >
8E0  < 9A 00 C0 07 DE 05 9E 00 08 05 9A 07 4C 06 36 07 >
8F0  < 06 07 1E 06 00 82 43 D2 08 B3 01 B9 05 E0 06 DE >   CR
900  < 05 DC 06 DE 06 00 84 57 4F D2 08 F5 01 B9 07 83 >   WORD
910  < 05 E8 07 EC 06 EB 07 16 07 A6 05 E8 07 7B 07 C0 >
920  < 06 F1 05 9E 00 08 06 44 05 9C 00 06 05 A8 06 F1 >
930  < 07 0E 06 06 07 7B 07 83 06 00 85 43 4F D5 09 06 >   COUNT
940  < 01 B9 07 16 06 FF 07 A6 07 06 06 F8 06 00 84 54 >
950  < 59 D0 09 3A 01 B9 05 D4 07 D2 05 D0 05 A0 07 65 >   TYPE
960  < 07 EC 07 3B 06 F8 06 DE 05 A4 FF F4 07 1E 06 00 >
970  < 84 46 49 CE 09 4E 01 B9 05 E8 09 0C 05 C8 06 F1 >   FIND
980  < 07 0E 06 04 07 16 05 9E 00 04 06 00 07 1E 05 C4 >
990  < 06 F1 07 83 06 04 06 00 82 44 AB 09 70 04 84 87 >
9A0  < 44 4E C5 09 98 04 9C 87 43 4F CD 09 9F 01 B9 05 >   DNEGATE COMPILE
9B0  < AC 06 F1 05 9E 00 12 07 06 07 16 05 D8 07 3B 06 >
9C0  < FF 06 F1 07 89 06 00 05 9A 01 43 06 36 C7 4C 49 >   LITERAL
9D0  < D4 09 A7 01 B9 05 AC 06 F1 05 9E 00 08 09 AD 05 >
9E0  < 9A 07 89 06 00 C8 44 4C C9 09 CD 01 B9 05 AC 06 >   DLITERAL
9F0  < F1 05 9E FF F0 09 AD 06 02 07 89 07 89 06 00 85 >
A00  < 44 49 C7 09 E5 01 B9 05 9A 00 30 07 D2 07 16 05 >   DIGIT
A10  < D0 07 5F 05 9E 00 2A 07 16 05 DC 07 5F 05 9E 00 >
A20  < 12 05 9A 00 07 07 D2 07 16 05 DC 07 5F 05 9E 00 >
A30  < 10 07 16 07 CA 06 F1 07 A6 07 5F 05 9E 00 08 07 >
A40  < 1E 05 D0 06 00 05 D4 06 00 86 4E 55 CD 09 FF 01 >   NUMBER
A50  < B9 05 D0 05 D0 07 FE 07 16 07 A6 06 F8 05 9A 00 >
A60  < 2D 07 DE 07 16 06 FF 07 3B 05 9A FF FF 05 9A 00 >
A70  < 10 06 E4 07 A6 07 16 06 FF 06 F8 0A 05 05 9E 00 >
A80  < 30 07 0E 07 CA 06 F1 07 4A 07 1E 07 FE 07 CA 06 >
A90  < F1 07 4A 09 9D 05 9A 00 10 06 F1 07 A6 05 9E 00 >
AA0  < 0A 05 D4 05 9A 00 10 08 A3 07 06 05 9C FF C6 07 >
AB0  < 06 07 16 06 F8 05 E8 07 D2 05 9E 00 1E 07 16 06 >
AC0  < F8 05 9A 00 2E 07 D2 05 9E 00 0A 07 06 05 9A 02 >
AD0  < 4E 06 36 05 D0 05 9C FF 96 07 1E 07 06 05 9E 00 >
AE0  < 04 09 A5 06 00 84 51 55 C9 0A 49 01 B9 05 B4 06 >   QUIT
AF0  < F1 05 9A 00 FE 06 E4 05 D0 07 C0 06 E4 05 D0 05 >
B00  < AC 06 E4 05 B8 06 F1 05 9A 00 7A 06 E4 08 FA 05 >
B10  < D0 07 B6 06 E4 05 A8 06 F1 05 D0 07 EC 06 EB 07 >
B20  < 16 07 A6 05 9A 00 40 07 7B 05 A8 06 F1 05 9A 00 >
B30  < 40 08 1E 06 62 05 AC 06 F1 07 58 05 9E FF D0 05 >
B40  < A6 02 4F 4B 05 9C FF BD C1 DB 0A E5 01 B9 05 D0 >   [
B50  < 05 AC 06 E4 06 00 C1 DD 0B 48 01 B9 05 9A 00 C0 >   ]
B60  < 05 AC 06 E4 06 00 C1 A7 0B 56 01 B9 09 76 07 16 >   '
B70  < 05 9E 00 0A 05 D8 07 3B 09 D3 06 00 05 9A 05 57 >
B80  < 06 36 83 4D 49 CE 0B 66 01 B9 07 EC 07 EC 07 5F >   MIN
B90  < 05 9E 00 04 07 0E 07 1E 06 00 85 41 4C CC 0B 82 >   ALLOT
BA0  < 01 B9 05 9A 00 12 08 A3 06 00 86 43 52 C5 0B 9A >   CREATE
BB0  < 01 B9 09 76 05 9E 00 08 05 9A 00 55 06 08 07 83 >
BC0  < 07 16 06 F8 05 BC 06 F1 0B 88 0B A0 05 9A 00 A0 >
```

```
Addr    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F    Comment

BD0  < 07 EC 06 F8 07 2D 07 EC 06 EB 07 83 05 F0 07 EC >
BE0  < 06 F8 07 2D 07 0E 06 EB 05 D4 0B A0 05 CC 06 F1 >
BF0  < 07 89 05 CC 06 E4 05 D0 07 83 06 E4 06 00 C1 BA >   :
C00  < 0B AA 01 B9 05 AC 06 F1 05 9E 00 08 05 9A 04 44 >
C10  < 06 36 05 F4 06 F1 05 F8 06 E4 05 C8 06 F1 05 C4 >
C20  < 06 E4 0B B0 0B 5A 09 AD 01 B9 06 00 86 53 4D D5 >   SMUDGE ,
C30  < 0B FE 01 B9 05 C8 06 F1 06 F1 05 9A 00 DF 07 EC >
C40  < 06 F8 07 26 07 0E 06 EB 06 00 C1 BB 0C 2C 01 B9 >   ;
C50  < 05 F4 06 F1 05 F8 06 F1 07 D2 05 9E 00 08 05 9A >
C60  < 03 53 06 36 09 AD 06 00 0C 32 0B 4C 06 00 88 43 >
C70  < 4F CE 0C 4A 01 B9 0B B0 05 9A 03 53 07 89 07 89 >   CONSTANT
C80  < 0C 32 06 00 87 3C 42 D5 0C 6E 01 B9 05 D0 0C 74 >   <BUILDS
C90  < 06 00 85 44 4F C5 0C 84 01 B9 05 C8 06 F1 06 F1 >   DOES>
CA0  < 07 A6 07 16 06 F8 05 F0 07 26 05 9E FF F4 05 9A >
CB0  < 00 03 07 3B 05 9A 04 B5 07 EC 06 E4 05 D8 07 3B >
CC0  < 07 06 07 0E 06 E4 06 00 89 49 4D CD 0C 92 01 B9 >   IMMEDIATE
CD0  < 05 C8 06 F1 06 F1 05 9A 00 40 07 EC 06 F8 07 2D >
CE0  < 07 0E 06 EB 06 00 C2 2E A2 0C C8 01 B9 05 D4 07 >   ."
CF0  < B6 08 A3 09 AD 05 A6 05 9A 00 22 09 0C 06 F8 07 >
D00  < A6 0B A0 05 D4 07 B6 08 A3 06 00 83 50 41 C4 0C >   PAD
D10  < E6 01 B9 07 83 05 9A 00 44 07 3B 06 00 84 48 4F >   HOLD
D20  < CC 0D 0B 01 B9 05 B0 05 9A FF FF 07 EC 08 A3 06 >
D30  < F1 06 EB 06 00 82 3C A3 0D 1D 01 B9 0D 11 05 B0 >   #
D40  < 06 E4 06 00 81 A3 0D 35 01 B9 07 CA 06 F1 06 FF >   <#
D50  < 05 D0 07 65 07 51 07 06 07 0E 06 FF 07 51 07 06 >
D60  < 07 FE 07 16 05 DC 07 5F 05 9E 00 08 05 9A 00 07 >
D70  < 07 3B 05 9A 00 30 07 3B 0D 23 06 00 82 23 D3 0D >   #S
D80  < 44 01 B9 0D 48 07 EC 07 EC 07 2D 07 58 05 9E FF >
D90  < F4 06 00 82 23 BE 0D 7C 01 B9 07 AE 05 B0 06 F1 >   #>
DA0  < 0D 11 07 EC 07 D2 06 00 84 53 49 C7 0D 93 01 B9 >   SIGN
DB0  < 07 FE 05 9A 80 00 07 5F 05 9E 00 08 05 9A 00 2D >
DC0  < 0D 23 06 00 84 44 41 C2 0D A8 01 B9 07 16 05 9A >   DABS
DD0  < 80 00 07 5F 05 9E 00 04 09 A5 06 00 82 44 AE 0D >   D.
DE0  < C4 01 B9 07 0E 07 EC 0D CA 0D 3A 0D 81 0D AE 0D >
DF0  < 98 09 54 06 00 84 53 2D BE 0D DC 01 B9 05 D0 07 >   S->
E00  < EC 05 9A 80 00 07 5F 05 9E 00 06 05 D4 07 D2 06 >
E10  < 00 C2 44 CF 0D F5 01 B9 09 AD 05 A0 07 83 06 00 >   DO
E20  < C5 2B 4C CF 0E 11 01 B9 09 AD 05 A2 06 BE 06 00 >   +LOOOP
E30  < C4 4C 4F CF 0E 20 01 B9 09 AD 05 A4 06 BE 06 00 >   LOOP
E40  < C2 49 C6 0E 30 01 B9 09 AD 05 9E 07 83 05 D0 07 >   IF
E50  < 89 06 00 C4 54 48 C5 0E 40 01 B9 07 83 07 EC 07 >   THEN
E60  < D2 07 0E 06 E4 06 00 C4 45 4C D3 0E 53 01 B9 09 >   ELSE
E70  < AD 05 9C 07 83 05 D0 07 89 07 0E 0E 59 06 00 C5 >
E80  < 42 45 C7 0E 67 01 B9 07 83 06 00 C5 55 4E D4 0E >   BEGIN  UNTIL
E90  < 7F 01 B9 09 AD 05 9E 06 BE 06 00 C5 41 47 C1 0E >   AGAIN
EA0  < 8B 01 B9 09 AD 05 9C 06 BE 06 00 C5 57 48 C9 0E >   WHILE
EB0  < 9B 01 B9 0E 45 06 00 C6 52 45 D0 0E AB 01 B9 06 >   REPEAT
```

```
Addr    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F    Comment

EC0  < FF 0E A1 07 06 0E 59 06 00 C4 43 41 D3 0E B7 01 >   CASE
ED0  < B9 05 FC 06 F1 05 F4 06 F1 05 FC 06 E4 06 00 C2 >
EE0  < 4F C6 0E C9 01 B9 09 AD 07 EC 09 AD 07 DE 0E 45 >   OF
EF0  < 09 AD 07 1E 06 00 C7 45 4E C4 0E DF 01 B9 05 F4 >   ENDCASE
F00  < 06 F1 05 FC 06 F1 07 D2 05 9E 00 08 0E 59 05 9C >
F10  < FF EE 05 FC 06 E4 06 00 C6 46 4F D2 0E F6 01 B9 >   FORGET
F20  < 09 76 07 16 05 9E 00 10 05 C8 06 F1 05 C4 06 F1 >
F30  < 07 D2 05 9E 00 0A 07 1E 05 9A 06 56 06 36 05 D8 >
F40  < 07 D2 07 16 06 F1 05 C8 06 F1 06 E4 05 D4 05 D0 >
F50  < 05 A0 05 D4 07 D2 07 16 06 F8 05 F0 07 26 05 9E >
F60  < FF F2 05 A4 FF EE 05 9A 00 12 06 E4 06 00 84 42 >   BOOT
F70  < 4F CF 0F 18 01 B9 05 D0 07 B6 06 E4 07 C0 06 E4 >
F80  < 06 62 0A EB 66 E7 18 6B 11 46 03 10 E6 06 0F E6 >  Code for shared-
F90  < 07 A0 E6 F9 25 B0 FA 46 FB 02 56 E7 07 8D 00 84 >   bus interrupt
FA0  < 70 FD 31 50 70 EF 70 EE 70 ED 70 EC EC FF FC FD >
FB0  < C2 DE 56 ED 18 A6 ED 10 6B 0C EC F7 FC 00 E5 5F >
FC0  < EC D2 CE FE EB F8 E6 F8 BA 56 03 EF 76 03 08 6B >
FD0  < FB 46 03 10 E6 F8 B2 42 DD EB 0B C2 CE F5 EC 5F >
FE0  < FE 76 EF 80 6B F5 50 EC 50 ED 50 EE 50 EF 50 FD >
FF0  < BF FF FF FF FF FF FF FF FF FF FF FF FF FF 0A EB > Start address: QUIT
```

Internal Distribution

1. F. P. Baxter
2. K. R. Carr
3. J. E. Christian
4-13. R. G. Edwards
14. P. D. Fairchild
15. W. Fulkerson
16. M. B. Gettings
17. R. L. Goeltz
18. I. G. Harrison
19. R. B. Honea
20. A. F. Huntley
21. H. L. Hwang
22. M. R. Ives
23. R. O. Johnson
24. R. S. Loffman
25. Sharon McConathy
26. H. Perez-Blanco
27. C. H. Petrich
28. P. H. Shipp
29. R. S. Solanki
30. S. S. Stevens
31. Central Research Library
32-33. Document Reference Section
34. Laboratory Records
35. Laboratory Records - RC
36-37. ORNL Patent Office

External Distribution

38. Office of Assistant Manager for Energy Research and Development, Department of Energy, Oak Ridge Operations Office, Oak Ridge, TN 37831

39. Jaime G. Carbonell, Associate Professor of Computer Science, Carnegie-Mellon University, Pittsburgh, PA 15213

40. S. Malcolm Gillis, Dean, Graduate School, Duke University, 4875 Duke Station, Durham, NC 27706

41. Peter Hogarth, P. O. Box 1235, Fairfield, IA 52556

42. Fritz Kalhammer, Vice President, Electric Power Research Institute, P. O. Box 10412, Palo Alto, CA 94303

43. Roger E. Kasperson, Professor of Government and Geography, Graduate School of Geography, Clark University, Worcester, MA 01610

44. Martin Lessen, Consulting Engineer, 12 Country Club Drive, Rochester, NY 14618

45-65. David MacFadyen, NAHB National Research Center, 400 Prince Georges Blvd., Upper Marlboro, MD 20772-8731

66-67. Technical Information Center, P. O. Box 62, Oak Ridge, Tn. 37831