

# ornl

**OAK RIDGE  
NATIONAL  
LABORATORY**

**MARTIN MARIETTA**

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

MARTIN MARIETTA ENERGY SYSTEMS LIBRARIES



3 4456 0262037 7

ORNL/TM-10361

## Evaluation of a Single-Board Microcontroller Suitable for Rapid Prototyping

Robert Edwards

OAK RIDGE NATIONAL LABORATORY

CENTRAL RESEARCH LIBRARY

CIRCULATION SECTION

4500N ROOM 177

**LIBRARY LOAN COPY**

DO NOT TRANSFER TO ANOTHER PERSON

If you wish someone else to see this  
report, send in name with report and  
the library will arrange a loan.

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: A03; Microfiche A01

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Energy Division

Evaluation of a Single-Board Microcontroller  
Suitable for Rapid Prototyping

Robert Edwards

Date Published - February 1987

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

NOTICE: This document contains information of  
a preliminary nature. It is subject to  
revision or correction and therefore does  
not represent a final report.

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

MARTIN MARIETTA ENERGY SYSTEMS LIBRARIES



3 4456 0262037 7



## TABLE OF CONTENTS

	Page Number
LIST OF TABLES . . . . .	v
ABSTRACT . . . . .	vii
1. INTRODUCTION . . . . .	1
2. FUNCTIONAL REQUIREMENTS FOR A PROTOTYPING MICROCONTROLLER . . . . .	4
3. DESCRIPTION OF A SINGLE-BOARD, FORTH-BASED PROTOTYPING MICROCONTROLLER . . . . .	6
3.1 The Zilog Z8 Family of Microcomputers . . . . .	6
3.2 Introduction to the Z8 Architecture . . . . .	7
3.3 Z8 Addressing Methods . . . . .	7
3.4 The Micromint Z8-Based family of Microcontroller Products . . . . .	7
3.5 Z8 FORTH Capabilities . . . . .	9
4. A FORTH EDITOR SUITABLE FOR RAPID PROTOTYPING . . . . .	12
5. USE OF MACHINE LANGUAGE SUBROUTINES WITH Z8 FORTH . . . . .	14
6. SUITABILITY OF THE Z8 FORTH MICROCOMPUTER FOR RAPID PROTOTYPING . . . . .	16
6.1 Functionality . . . . .	16
6.2 Speed . . . . .	17
7. CONCLUSIONS . . . . .	20
8. REFERENCES . . . . .	21
APPENDIX I. INTRODUCTION TO FORTH . . . . .	22
APPENDIX II. FORTH EDITOR FOR A VT52 EMULATOR . . . . .	25
APPENDIX III. UNSTRUCTURED LISTING OF THE PROTOTYPE EDITOR . . . . .	29
APPENDIX IV. AUXILIARY Z8 FORTH WORDS . . . . .	30
APPENDIX V. ACRONYMS . . . . .	31



LIST OF TABLES

	Page Number
Table 1. Words Provided by Z8 FORTH . . . . .	10
Table 1. (Continued) Words Provided by Z8 FORTH . . . . .	11
Table 2. Bench Marks of Z8 and Personal Computer FORTH . . . . .	19



## ABSTRACT

This report presents an evaluation of a single-board microcontroller suitable for rapid prototyping. 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. The Project will help manufacturers use advanced technology in the development of new products for communications, energy distribution, and appliance control. By integrating home communication, control, and energy networks, Smart House products will provide homeowners with new functions and accommodate future technological advances.

Testing Smart House hardware prototypes will require a general-purpose microcontroller, not only to help monitor test performance but also to simulate, if necessary, other Smart House devices associated with the device being tested. To provide this adaptability, a prototyping controller must offer a broad spectrum of functions that are easily used by a laboratory technician. Because it is optimized for laboratory use, a prototyping microcontroller may not be suitable for commercially-available Smart House products. The report presents a set of rapid-prototyping requirements based on past ORNL experience with laboratory testing and experimentation.

The evaluation showed that a Zilog Z8 microcomputer with a FORTH development system in internal ROM meets most of the ORNL-developed rapid-prototyping requirements. The report concludes that the Z8 FORTH microcontroller may be deficient in its complement of internally-provided FORTH words, in the clarity of control algorithms written in FORTH, and in execution speed for Z8 FORTH routines. Suggestions are provided for improving the understandability of FORTH programs and in using machine language routines to compensate for lack of execution speed.

The intent of this report is to provide Smart House Project management with information regarding the equipment needed for developing and testing Smart House products.



## 1. INTRODUCTION

This report presents an evaluation of a FORTH-based, single-board microcontroller suitable for rapid prototyping. The work was conducted by the Oak Ridge National Laboratory (ORNL) for the National Association of Home Builders Research Foundation (NAHB/RF) Smart House Project. The Project, a cooperative research and development effort involving American home builders and a number of major corporations, will incorporate current technology in the distribution and use of energy and communications services in the home. Through use of advanced electronics, improved designs for home communication, control, and energy distribution networks will provide opportunities for offering new functions and will provide a basis for accommodating future technological advances. The schedule for the Project calls for availability of Smart House cabling, electrical control devices, consumer appliances, and gas piping and control products for homes constructed from 1990 onward.

Certain parts of the designs now used for home electrical and gas distribution systems are as much as 100 years old. In the 1880s, utilities first provided gas to homes as a substitute for oil lamps and candles. A few years later, electricity was made available as a replacement for gas lighting. The use of electricity for appliances other than light fixtures came much later as a way to use the excess generating capacity produced during daylight hours. Today, the amount of energy used for illumination in homes is dwarfed by that required for heating and cooling appliances that were not even conceived of 50 years ago. In spite of the radical change in use patterns, the basic energy distribution and control systems used in homes have not changed in several decades. As a result, the needs of modern homes are not being met. By providing intelligent control and coordination services among appliances and the devices used to control their operation, the Smart House will make possible functions that are accomplished with difficulty or not at all using present wiring and control systems. For example, programmed control of heating, ventilating, and air

conditioning (HVAC) units and remote control of entertainment centers are difficult to accommodate in a conventionally designed home, but are simple to support with the Smart House concept.

The Smart House design for power distribution will also establish a new standard of safety in the home, primarily by incorporating closed-loop control of electrical appliances. With this feature, branch circuits are de-energized except when power is necessary to operate appliances. The device controlling an appliance (e.g., a switch on a vacuum cleaner) must send an electronic message to the control for the circuit leading to the appliance to initiate the flow of power. After the circuit is powered, the control for the circuit requires a continuing "nominal-operation" signal from the appliance to continue the supply of power. Closed-loop protection of circuits in the Smart House can be thought of as the electrical equivalent of the thermocouple protection device used 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 does not concentrate home control in one central computer. Rather, it calls for several distributed controllers all wired together, each with the ability to back up one another in case of failure.

As manager of the Project, NAHB is responsible for the overall design of the Smart House system. The needs statement for the design is documented in the recently issued Smart House Functional Specifications [1]. The manufacturers participating in the Project are using that document as a starting point for developing conceptual designs for their products. NAHB is also responsible for providing proof-of-principle and concept prototyping for the system as a whole. As a part of this task, NAHB is establishing a laboratory environment at their National Research Center to conduct studies using actual hardware in a simulated home environment.

ORNL is contributing to the Smart House Project by providing technical advice and consulting services to NAHB in the areas of overall evaluation

and design, project management, and development of facilities for use in design evaluation and system integration. This report concerns criteria ORNL has developed for evaluating microcontrollers intended for use in rapid prototyping, the adaptation of a commercially-available microcontroller to meet these requirements, and a summary evaluation of both the prototyping requirements and the performance of the candidate microcontroller. The purpose of this report is to provide Smart House Project management with information that can be used in making decisions regarding laboratory equipment for product development and testing.

## 2. FUNCTIONAL REQUIREMENTS FOR A PROTOTYPING MICROCONTROLLER.

The laboratory evaluation of Smart House concepts will require a general-purpose microcontroller that can be efficiently adapted to explore and validate designs for proposed Smart House products. This microcontroller must meet requirements that focus on its utility in laboratory experimentation and testing, and thus may not have characteristics necessary for controllers used in commercially-offered Smart House products. Also, because of the significant adaptability required of a prototyping microcontroller, technicians using it may have to have experience with computer software such as programming interrupt drivers, and with specialized electronics such as analog signal conditioners.

ORNL proposes that a rapid-prototyping microcontroller should be able to meet the requirements listed below. The requirements are based on extensive experience in conducting laboratory testing and experimentation in areas such as the development of innovative heat pump designs, the field-test of load management systems, and the evaluation of energy-saving construction practices. The technical activities involved in these ORNL projects have been similar to those likely to be used in the evaluation of electronic devices required by Smart House products.

- o Interfaces. A microcontroller suitable for rapid prototyping must be able to provide interfaces for digital (parallel), serial, and frequency-shift keying (FSK) input/output and for analog input. (Serial and FSK input/output and analog input are intended for laboratory instrumentation purposes and may or may not be the communication methods used in commercial Smart House products.)
- o Size. To permit exploration of the packaging and installation aspects of proposed designs, the microcontroller should be sufficiently small to allow installation of the main board in the base of a small appliance, such as a floor lamp or toaster.
- o Power Requirements. The basic microcontroller board should not require more than a few hundred milliamps at 5 volts (approximately 1 to 3 watts) for usual operation. (Use of serial RS232 input/output may entail providing additional voltages to the microcontroller.) Adding auxiliary boards for FSK input/output,

additional memory, and analog input should not require more than an additional 1 amp.

- o Reliability. ORNL has found that reliability requirements are better stated in terms of the market acceptance of the product rather than as a list of specifications. Thus, a controller that has earned the trust of designers and gained wide acceptance for similar controller applications would be judged to be sufficiently reliable.
- o Cost/Availability. A basic microcontroller board, complete with power supply, should cost no more than \$150-200 for a single unit and less than \$100 in lots of 100. The controller board should be available as an off-the-shelf product. Auxiliary boards for FSK input/output or analog input should be similarly priced.
- o Adaptability. There are two aspects to this requirement. First, the control algorithms must be stated in terms easily transferable to a production controller suitable for initial Smart House products. Second, the functions performed by the controller used by the NAHB Research Foundation must be transferable to any of a number of different types of prototyping controllers used by the participating manufacturers in their product development areas.
- o Functionality. To expedite prototyping tests, a programmer should be able to alter a program within a few minutes using only the microcomputer and a conventional American National Standards Institute (ANSI) terminal. (It is expected that a personal computer, with software for emulation of an ANSI terminal, will be required for storing code and source program material between testing sessions.)
- o Speed. The microcontroller must be able to handle functions that occur in time frames as short as several milliseconds. For example, the Smart House concept of closed-loop power distribution may require disabling a power feed within a few milliseconds to provide the shock protection afforded by present ground-fault circuit breakers.

### 3. DESCRIPTION OF A SINGLE-BOARD, FORTH-BASED PROTOTYPING CONTROLLER.

ORNL used the requirements presented in Section 2 to test the rapid-prototyping potential of a currently available microcontroller. The first six requirements are addressed in this section in connection with a description of the device being evaluated. The speed and functionality characteristics of the device are presented in Section 6.

The microcontroller chosen for the evaluation uses a Zilog Z8 micro-computer, a chip that requires few additional electronic components to function as a stand-alone microcontroller. Microcomputers in the Z8 product line, which was introduced in the late 1970s, are widely used in laboratory equipment and other specialized controller applications. This wide use is a strong indicator that the device has the reliability required to withstand the abuse likely to be associated with laboratory experimentation.

#### 3.1 THE ZILOG Z8 FAMILY OF MICROCOMPUTERS.

All members of the Z8 family of microcomputers are variations of the basic Z8 microcomputer, the Z8601. The particular member of the family relevant to this evaluation is the Z8611, which has available 4K bytes of on-chip ROM. The following characteristics are common to all members of the Z8 microcomputer family:

- o 128 general-purpose 8-bit registers,
- o 16 special-purpose system-function registers,
- o 16 lines of programmable parallel input/output (ports 2 and 3),
- o 2 counter/timers with pre-scalers,
- o 6 vectored interrupts, and
- o Universal Asynchronous Receiver/Transmitter (UART) for serial input/output.

The Z8 microcomputer itself provides the parallel and serial input/output required to meet the rapid prototyping interfacing requirement. The requirements for FSK input/output and analog input must be met through use of auxiliary boards described in Section 3.4.

### 3.2. INTRODUCTION TO THE Z8 ARCHITECTURE.

All of the Z8 general-purpose registers, which include the dedicated input/output ports, can be used as accumulators. Z8 instructions may use these registers as individual accumulators, or use them in pairs as pointers for indirect addressing. Speed of execution is facilitated by a "working register area" concept that makes possible efficient reference to individual registers in 9 blocks, each block containing 16 registers. Z8 instructions are provided for operations on bits, bytes, 2-byte words, and binary coded decimal (BCD) digits.

The Z8 achieves a high throughput by using instruction pipelining. This method overlaps the execution of an instruction with the fetch of its successor. Execution speed for the Z8 is typically in the range of 250,000 instructions per second. This is about one-fourth the rate of the widely-available IBM PC personal computer.

### 3.3 Z8 ADDRESSING METHODS.

Z8 instructions can reference data by 8-bit register file location, 4-bit working register number, or 2-byte program memory address. Register file locations can be used as individual bytes of data, as 2-byte words, or as 1- or 2-byte pointers for indirect reference.

### 3.4 THE MICROMINT Z8-BASED FAMILY OF MICROCONTROLLER PRODUCTS.

The Z8 microcomputer based on the FORTH software system is offered by Micromint in an off-the-shelf line of microcomputer boards. The Micromint firm is owned by Steve Ciarcia, the author of the monthly "Circuit Cellar" column in BYTE magazine. In 1981, Ciarcia presented the antecedent of the Micromint Z8 products, a single-board microcontroller using the BASIC programming language, as a construction project in BYTE [2].

The boards in the Micromint series are described below. Each board measures 4- by 4-1/2 inches, small enough to permit use in most appliances.

- o SBC11. The key Micromint product is a stand-alone microcontroller containing a Z8611 microcomputer with a FORTH system in 4K bytes of internal ROM. The board provides sockets for 6116 (2K-byte) random-access memory (RAM) and 2716 (2K-byte) or 2732 (4K-byte) electrically programmable read-only memory (EPROM). The board provides easy access to two 8-bit ports (ports 2 and 3 of the Z8611) for parallel input/output, and a memory-mapped 8-bit port for digital input. The SBC11 requires 250 milliamps at 5 volts. Plus and minus 12 volts at 30 milliamps is required if serial input/output is used. (Quantity one price: \$149; quantity 100 price: \$94)
- o SBC33. This memory expansion and input/output board permits expansion of the basic SBC11 controller with up to 12K bytes of any combination of 6116 RAM, 6264 (8K-byte) RAM, 2716 EPROM, or 2732 EPROM. It also provides 3 parallel ports using an 8651 parallel input/output controller, and FSK input/output using an XR2211 demodulator. (Price: \$150)
- o SBC07. The Micromint EPROM board, which requires the BCC33 board for operation, will program either 2716 or 2732 EPROMs. (Price: \$145)
- o SBC14. This 16K-byte memory expansion board accommodates any mix of 2716/2732 EPROM or 6116 RAM. (Price: \$120)
- o SBC13. This analog input board uses an Analog Devices AD7581 chip to provide 8-bit resolution for as many as 8 input channels. Maximum throughput is 10,000 samples per second. (Price: \$140)
- o SBC30. This higher-capability analog-to-digital converter resolves as many as 16 channels to 12 bits of resolution. (Price: \$197)

All the tests for speed and functionality described in this paper were performed using only the SBC11 board. Development of the full-screen

editor described in Section 3.6, which entailed EPROM programming, required use of the SBC33 Memory Expansion Board and the SBC07 EPROM Board.

### 3.5 Z8 FORTH CAPABILITIES.<sup>1</sup>

The 96 FORTH words in Z8611 internal ROM are based on the 1979 FORTH Interest Group standard rather than the more current 1983 standard. Table 1 presents the complement of Z8 FORTH words grouped in the manner used by Leo Brodie in his book Starting FORTH [3]. The underlined words in Table 1 are those developed during the ORNL evaluation to adapt the Micromint-provided product to a rapid prototyping environment.

Z8 FORTH is essentially a subset of the FORTH found on a wide range of minicomputers, microcomputers, and microcontrollers. This wide availability would facilitate moving NAHB-developed algorithms to other types of controllers used by Smart House participating manufacturers. It would also allow manufacturers to develop a control program in FORTH on a commonly available system (e.g. a personal computer), and then to transfer the program to Z8 FORTH for use in laboratory testing.

---

<sup>1</sup>An introduction to the FORTH language is provided in Appendix I.

Table 1. Words Provided by Z8 FORTH

Memory:

!	(STORE STACK TO MEMORY)	C@	(FETCH CHARACTER TO STACK)
@	(FETCH FROM MEMORY TO STACK)	CMOVE	(MOVE FROM START OF BLOCK)
+	(STORE INCREMENTED)	<CMOVE	(MOVE FROM END OF BLOCK)
C!	(STORE CHARACTER TO MEMORY)	<u>READEP</u>	(READ-EPROM)
		<u>PROGRAM</u>	(PROGRAM-EPROM)

Arithmetic and Logical:

+	(PLUS)	AND	
-	(MINUS)	OR	
U*	(UNSIGNED MULTIPLY)	XOR	(EXCLUSIVE OR)
U/	(UNSIGNED DIVIDE)	1+	(INCREMENT TOP-OF-STACK)
NEGATE		S->D	(SINGLE-TO-DOUBLE)
DABS	(DOUBLE WORD ABSOLUTE VALUE)	DNEGATE	(DOUBLE WORD NEGATE)
D+	(DOUBLE WORD PLUS)		

Input/Output:

KEY	(INPUT FROM SERIAL PORT)	CR	(EMIT CARRIAGE RETURN)
EMIT	(OUTPUT TO SERIAL PORT)	SPACE	(EMIT SPACE)
EXPECT	(EXPECT CHARACTERS)	TYPE	(TYPE BLOCK OF CHARACTERS)
WORD	(SEARCH FOR WORD)	<#	(SETUP OUTPUT BLOCK)
PAD	(CREATE BUFFER)	#	(SETUP DIGIT IN BLOCK)
<u>RDBLK</u>	(READ BLOCK)	HOLD	(STORE CHARACTER IN BLOCK)
H.	(HEX OUTPUT TOP-OF-STACK)	SIGN	(INSERT SIGN IN BLOCK)
D.	(OUTPUT DOUBLE WORD)	#>	(TERMINATE OUTPUT BLOCK)
		#S	(SETUP NUMBER IN BLOCK)

Comparisons:

=	(STACK WORDS EQUAL)	MIN	(REPLACE WITH LESSER)
>=	(TOP-OF-STACK LESS)	0=	(TOP-OF-STACK ZERO)

Stack Manipulation (TOS means Top-of-Stack, 2nd means below TOS, etc.):

SWAP	(SWAP TOS WITH 2nd)	2DROP	(DROP TOS AND 2nd)
DUP	(DUPLICATE TOS)	I	(COPY LOOP INDEX TO STACK)
OVER	(DUPLICATE 2nd ABOVE TOS)	J	(COPY OUTER INDEX TO STACK)
ROT	(MOVE 3rd TO ABOVE TOS)	>R	(COPY TOS TO RETURN STACK)
DROP	(DROP TOS)	R>	(COPY RETURN STACK TO TOS)
NUMBER	(CONVERT STRING TO WORD)	DIGIT	(CONVERT WORD TO CHARACTER)
COUNT	(COUNT CHARACTERS IN STRING)		

Table 1 (Continued). Words Provided by Z8 FORTH

Structure:

IF	CASE
ELSE	OF
THEN	ENDCASE
DO	BEGIN
LOOP	UNTIL
+LOOP (INCREMENT INDEX FROM STACK)	WHILE
LEAVE	REPEAT
	AGAIN

Defining Words and Compilation:

:	(START WORD DEFINITION)	FIND
;	(STOP WORD DEFINITION)	FORGET
,	(COMMA, COMPILE TOS)	SMUDGE
C,	(COMPILE BYTE AT TOS)	LITERAL
'	(TICK, COMPILE ADDRESS)	DLITERAL
CREATE		<BUILDS
CONSTANT		DOES>
IMMEDIATE		[ (ENTER IMMEDIATE MODE)
ALLOT (ALLOT SPACE IN DICTIONARY)		] (ENTER COMPILE MODE)
COMPILE		." (DEFINE CHARACTER STRING)
0 (NULL, END-OF-BLOCK FENCE)		

FORTH System Variables:

BASE	>IN	(INTERPRETER INPUT INDEX)
BLK (INTERPRETER INPUT BLOCK)	HERE	(NEXT DICTIONARY ADDRESS)

FORTH System Commands:

BOOT	QUIT
<u>EDIT</u>	COLD
<u>DUMP</u>	EXECUTE

#### 4. A FORTH EDITOR SUITABLE FOR RAPID PROTOTYPING.

The editor supplied with the Micromint microcontroller board is not suitable for full-screen operation using a personal computer as a "smart" terminal. To test the suitability of Z8 FORTH for a non-trivial programming task, the author developed a full-screen editor under constraints similar to those that might be expected in rapid prototyping. The constraints assumed that

- o the source and the compiled form of the editor can be no longer than 2K bytes,
- o the editor must accommodate character transmission rates up to 9600 baud, and
- o the editor must have standard cursor movement and insert/delete features similar to those found in Wordstar [4].

A structured listing of the editor appears in Appendix II together with a line-by-line commentary for readers not familiar with the FORTH language. The program developed actually provides not only a full-screen editor, but also a screen initialization routine (.CLEAR), a screen loader (LSCR), a dump routine (DUMP) for viewing the hexadecimal (HEX) and American Standard Code for Information Interchange (ASCII) image of any 256-byte block of memory, and FORTH headers for EPROM programming routines. The compiled form of the editor, the auxiliary FORTH words, and the EPROM subroutines themselves occupy only 2K bytes of EPROM.

The functions and keys used to perform editing functions are listed below.

- o Numeric Pad "Arrow" Keys: Up, down, left, and right movement, one character or line at a time
- o Page Up/Page Down Keys: Delete current line/Insert a blank line
- o Back Space Key: Left one space, replacing the character at that position with a blank
- o Insert-Delete/Typeover (a toggle): Insert or delete characters on the current line at the cursor position, or revert to the default type-over mode

The rapid-prototyping editor is designed to work with an emulator that permits redefining personal computer keyboard codes, such as the VersaCom communications package from Solution Software [5]. Users of packages, such as SmartCom, that do not offer this feature will have to change the FORTH editor logic for control character recognition.

The author, who had little experience with FORTH prior to this evaluation, was able to develop the full-screen editor in about 40 hours. However, people with no prior experience working in a microcomputer machine-instruction environment may find FORTH very difficult to use.

Critics of FORTH state that FORTH programs are difficult to read. The unstructured editor listing in Appendix III, which is an example of poor FORTH programming practice, supports their contention. However, a structured listing of the same program (Appendix II) improves its readability by organizing the information to show the programmer's use of FORTH logic control words.

## 5. USE OF MACHINE LANGUAGE SUBROUTINES WITH Z8 FORTH.

Z8 FORTH makes it easy to link FORTH routines to the easily-used Z8 machine instruction code. A programmer may be forced to use this capability because the FORTH in Z8 ROM may be too slow for a particular time-critical task.

A programmer has two methods of calling a machine language subroutine in Z8 FORTH. The simplest is to place the address of a pointer to the subroutine on the stack followed by the FORTH word EXECUTE. Suppose the start of the subroutine to be called is at C186 and the word at B456 is a pointer to the subroutine (that is, B456 contains the address C186). The FORTH definition for the word SUBCALL shown below

```
: SUBCALL B456 EXECUTE ;
```

will pass control to the routine at C186 when the word SUBCALL is executed.

A slightly more sophisticated method, which results in a shorter compiled code, is to create a FORTH word for the subroutine with CREATE:

```
CREATE SUBCALL SMUDGE C186 ,
```

CREATE will create a FORTH word header followed by a single word that transfers control to the subroutine at C186. (The FORTH word SMUDGE resets a bit in the header to make the created word executable; the FORTH word ", " (comma) compiles the subroutine address into the FORTH definition for SUBCALL.)

Inserting a block of Z8 instructions into a FORTH program is also quite simple. In the example on the next page the FORTH word TEST assumes that an iteration count is on the stack and uses this number to control the times an "empty" loop is repeated.

Z8 Code	Assembly Language	Comments
50EC	TEST POP WRC	Pop Top-of-Stack into working register
50ED	POP WRD	pair C and D (Iteration count)
80EC	LH DECW WRC	Decrement count in C and D
EBFC	JR NZ,LH	Repeat loop if character count not zero
3050	JP VNEXT	Exit to next FORTH word

The FORTH words to define this subroutine are

```
CREATE TEST SMUDGE HERE 2 + , 50EC , 50ED , 80EC , EBFC , 3050 ,
```

The sequence HERE 2 + defines the execution address in the FORTH header for TEST as the next location in the dictionary. The Z8 instructions themselves are compiled by the sequence of number/comma pairs in the definition.

## 6. SUITABILITY OF THE Z8 FORTH MICROCOMPUTER FOR RAPID PROTOTYPING.

As was shown in Section 3, the Micromint microcontroller meets all rapid-prototyping requirements for interfaces, size, power requirements, reliability, cost/availability, and adaptability. The only remaining questions concern its ability to meet functionality and speed requirements.

### 6.1 FUNCTIONALITY.

Most FORTH programmers will find the complement of Z8 FORTH words too limiting and will want to add definitions usually thought to be part of a standard definition of FORTH. For example, Z8 FORTH only provides words for two comparisons, "equal" and "greater-to-or-equal." Fortunately, it is easy to define other comparison words using those provided:

FORTH Word	FORTH Definition using the words provided in Z8 FORTH
>	: > 1+ >= ;
<	: < SWAP >= ;
<=	: <= 1+ SWAP >=;

Similarly, the Z8 FORTH mixture of single-length signed and unsigned and double-length arithmetic words is limited. Z8 FORTH provides single-length arithmetic words for plus, minus, negate, increment top-of-stack, and increment memory; unsigned arithmetic words for multiply, divide, and loop control; and double-length arithmetic words for plus, absolute value, and negate. Fortunately, as was the case with the definition of additional words for condition testing, it is relatively easy to define arithmetic words in terms of existing FORTH words, or in assembly language using the technique shown in Section 6.2.

Many programmers are accustomed to using the "parenthesis" word that is provided in most implementations of FORTH to imbed comments within FORTH source material. The FORTH word "parenthesis" is easily defined in Z8 FORTH as follows:

```
: ( 29 WORD 1 >IN +! ; IMMEDIATE ( PERMITS COMMENTS )
```

A number of essential utility routines are missing from Z8 FORTH, such as those for reading a block of characters from the serial port at any of the allowed baud rates, checking an EPROM to verify that it is blank, and comparing an EPROM with memory. Appendix IV provides definitions for FORTH words to provide these functions. (The routines for reading and verifying an EPROM are adapted from examples given in the Z8 FORTH Reference Manual. [6])

In spite of the somewhat abbreviated extent of FORTH words in the Z8611 ROM, it is relatively easy to construct, in just a few minutes time, any additional FORTH words needed to alter a control program.

## 6.2 SPEED.

The time required to translate source text into compiled code is generally the limiting factor in making a change to a FORTH program. For example, 250 FORTH words (about as many as can be packed into a FORTH page of 1024 bytes) take about 10 seconds to compile.

Execution of a 4-word FORTH loop that initializes a block of 16K bytes of memory to blanks takes 15 seconds with Z8 FORTH, which corresponds to an execution rate of about 5000 FORTH words per second. This speed is only marginally acceptable for the performance required for Smart House prototyping. Fortunately, it is easy to "optimize" Z8 FORTH code to achieve very favorable performance. For example, the FORTH word TEST, used in establishing the bench mark just discussed

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

can be replaced with the Z8 instructions below, which will blank a block of memory 100 times faster.

```
CREATE TEST SMUDGE HERE 2 + , 50EC , 50ED , 50EA , 50EB , 9C20 ,  
D29A , A0EA , 80EC , EBF8 , 3050 ,
```

The instructions in Z8 assembly language are:

Z8 Code	Assembly Language	Comments
50EC	TEST POP WRC	Pop Top-of-Stack into working register
50ED	POP WRD	pair C and D (Character count)
50EA	POP WRA	Pop word underneath TOS in working
50EB	POP WRB	register pair A and B (Address)
9C20	LD WR9,20	Load a blank into working register 9
D29A	LH LD @@WRA,WR9	Store blank in address pointed to by A&B
A0EA	INCW WRA	Point A and B to next address
80EC	DECW WRC	Decrement character count in C and D
EBF8	JR NZ,LH	Repeat loop if character count not zero
3050	JP VNEXT	Exit to next FORTH word

When machine language routines are used in FORTH code to optimize time-critical operations, Z8 FORTH programs can be made to operate at speeds comparable to machines usually thought to be in an entirely different class. Table 2 presents comparisons of optimized and nonoptimized versions of Z8 FORTH routines with other microcomputer FORTH bench marks. The last entry in Table 2 shows the performance of the Novix NC4000 [9], a microprocessor that has FORTH words as its native instruction set.

TABLE 2.

Bench Marks of Z8 and Personal Computer FORTH.

FORTH Version	Minutes:Seconds to Perform 1,000,000 Iterations	
	-----Contents of Loop----- Empty Loop	16-Bit Store
Z8 FORTH Non-optimized	6:40.0	13:20.0
MVP-FORTH IBM-XT	1:47.30	5:25.19
MVP-FORTH IBM-AT	0:35.85	1:07.25
Z8 FORTH Optimized	0:08.00	0:16.00
Novix FORTH	0:02.50	0:03.22

## 7. CONCLUSIONS.

The ORNL-determined requirements for a microcontroller suitable for Smart House related development and testing can be met using Micromint Z8 FORTH products. However, the Micromint Z8 FORTH microcontroller should be used with caution because of potential shortcomings in three areas:

- o the need to augment the basic complement of FORTH words provided in ROM with those customarily used in writing FORTH programs,
- o the potential difficulty in reading FORTH source material, and
- o the need to use Z8 machine language subroutines to achieve the required execution speed.

## 8. REFERENCES

- [1] Smart House Functional Description, Smart House Development Venture Inc., October 1986.
- [2] S. Ciarcia, "Build a Z8-Based Control Computer with BASIC," Parts I and II, BYTE, July-August, 1981.
- [3] L. Brodie, Starting FORTH, Prentice-Hall, Englewood Cliffs, NJ, 1981.
- [4] WordStar Reference Manual Release 3.3, MicroPro International, 1983.
- [5] VersaCom Users' Guide, Second Edition, Solution Software, Tucson, Arizona, July 1985.
- [6] Z8 Microcomputer Technical Reference Manual, Zilog Inc., 1984.
- [7] Z8671 Single-Chip BASIC Interpreter - BASIC/DEBUG Software Reference Manual, Zilog Inc., June 1981.
- [8] Micromint Z8 Reference Manuals:
  - Z8 BASIC System/Controller, 1983.
  - Micromint Z8 FORTH, December 1984.
  - Z8 System Controller Expansion Board II.
  - Z8 BASIC 16K Memory Expansion Board, 1983.
  - Eight Bit Analog to Digital Converter Board, 1983.
  - Z8 EPROM Programmer.
- [9] "Fast Processor Chip Takes Its Instructions Directly From FORTH," Electronic Design, March 21, 1985.

## APPENDIX I.

### INTRODUCTION TO FORTH

FORTH was originally developed by Charles Moore in the 1970s to facilitate computer control of telescope operation. During the early 1980s it became very popular with microcomputer enthusiasts and with firms specializing in applications requiring dedicated microcomputer controllers. FORTH is an unusual language in that it is not strictly an interpreter, like most BASIC implementations, nor a source-to-object language converter, such as an assembler. Most computer professionals, when first introduced to FORTH, feel very uncomfortable because the language is so different from anything they have had experience with before.

FORTH has two modes, execution and compilation. A FORTH programmer can shift back and forth between these modes during the course of an interactive session with a computer. The following example (computer output underlined) shows this process:

```
: GREET ." Good evening " CR ;OK      [compiles a new FORTH word, GREET]
GREET Good evening                    [executes the FORTH word GREET]
OK
```

The "natural" method of stating a FORTH process is with post-fix (sometimes called reverse-Polish) notation, similar to the way calculations are done on a Hewlett-Packard calculator. Thus, adding 3 and 4 together, then 5 and 6, and finally determining the product of the two sums is thought of as

```
3 4 + 5 6 + *
```

Although FORTH programmers can reference memory in the same way as with a conventional programming language, FORTH encourages use of a utility stack to keep track of constants and operations to be performed. If a snapshot

of the FORTH stack could be taken after each step in the exercise just described, the result would be

```

-----STEP-----
  1  2  3  4  5  6  7  8  9 10 11
top of stack > 3  4  +  7  5  6  + 11  *  77
                3  4      7  5  6  7  11
                3      7  5      7
bottom of stack >

```

The use of post-fix notation and extensive use of a stack for communicating information within a FORTH program is often viewed as an inconvenience at best, and a serious obstacle to readability at worst. It is the sort of thing that is considered to be either a valuable asset or a serious flaw.

FORTH is somewhat unique among programming languages in that any word in the FORTH system itself can be redefined. For example, consider the FORTH word KEY, which is used to obtain a character from the serial input port. The standard FORTH convention is to provide only the low seven bits of an eight-bit word because those are the only bits defined in ASCII standard. However, if an application requires availability of all eight bits of the input byte, the programmer simply redefines KEY, recompiles the FORTH word definitions dependent on the word KEY, and continues right along.

Three groups of program structure words, IF-ELSE-THEN, DO-LEAVE-LOOP, and CASE-OF-THEN-ENDCASE, are significant aids in writing FORTH programs and contribute to readability of the final results. The IF-ELSE-THEN group permits writing conditional clauses in a way similar to PASCAL. For example, suppose the FORTH word TEST is to read a character from the serial input port and type the message "valid" if the character X was typed, "invalid" otherwise. Using the IF-ELSE-THEN construct, TEST is

```
: TEST KEY 58 = IF ." valid " ELSE ." invalid " THEN ;
```

The value 58 (in HEX) is equivalent to the character X. Notice that the post-fix convention sets up the test ahead of the "=" word, which in turn precedes IF.

The DO-LEAVE-LOOP group facilitates writing index-controlled iterations. The example below shows how the phrase "Good Evening" can be repeated 8 times by a program loop.

```
: LTEST CR 8 1 DO ." Good Evening" CR LOOP ;
```

The FORTH word CR sends a carriage return code to the serial output port. The word DO assumes that the stack has been set up with the final and initial values of the loop index. The loop index can be obtained through the FORTH word I. Loops can be nested to any level. The word LEAVE within a loop can be used to accomplish a conditional loop exit.

The CASE-OF-THEN-ENDCASE group can be used to advantage to accomplish different tasks corresponding to each of several values of a given quantity. In the example given below, a character is obtained from the serial port and one of three messages is printed depending on whether the character is an X (HEX 58), Y (HEX 59), or Z (HEX 5A). If the character is none of these, a fourth message indicating this fact is displayed.

```
: CTEST KEY
  CASE 58 OF ." X was entered " THEN
  ELSE 59 OF ." Y was entered " THEN
  ELSE 5A OF ." Z was entered " THEN
  ELSE DROP ." Neither X, Y, nor Z was entered "
  ENDCASE ;
```

## APPENDIX II.

STRUCTURED SOURCE LISTING  
FORTH EDITOR FOR A VT52 EMULATOR

```

1> : SCR 74 ; : S@ 74 @ ; : P@ 71 C@ ; : L@ 70 C@ ;
2> : G! 1B EMIT EMIT ; : B0 7 EMIT 0 ; : G> 59 G! 21 + EMIT 24 + EMIT ;
3> : C* 40 U* DROP + S@ + ;
4> : ;S BLK @ IF R> DROP THEN ; IMMEDIATE
5> : LSCR >IN @ 70 ! 0 >IN ! S@ 40 U* BLK ! DROP 662 EXECUTE
6> 70 @ >IN ! 0 BLK ! ;
7> : .CLEAR 20 S@ C! S@ S@ 1+ 3FF CMOVE 3B53 S@ 3FD + ! ;
8> : LIST DO I 0 <# # # #> TYPE ." <|" 40 I C*
9> 3F 0 DO 1 - DUP C@ 20 - IF LEAVE THEN LOOP
10> DUP FFCO AND DO I C@ EMIT LOOP 40 I G> ." |" CR LOOP ;
11> : ZL C* DUP 20 ROT C! DUP 1+ 3F CMOVE -4 L@ G> 4A G!
12> F L@ LIST 0 L@ G> 0 71 ! 0 ;
13> : MP 0 L@ C* DUP 40 + DUP 40 F C* SWAP - ;
14> : T = 72 @ OR ; : W 0= 72 @ OR ; : Q DUP 7F >= OVER 1F SWAP >= + ;
15> : DUMP 70 ! 48 G! 4A G! F 0 DO CR I 10 U* DROP 70 @ + H. ." <"
16> F 0 DO SPACE I J 10 U* DROP + 70 @ + C@ 0 <# # # #> TYPE LOOP
17> SPACE ." >>" F 0 DO I J 10 U* DROP + 70 @ + C@ 7F AND Q
18> IF 20 EMIT DROP ELSE EMIT THEN LOOP 3E EMIT LOOP CR ;
19> : R 8 EMIT 20 EMIT 8 EMIT -1 ;
20> : EDIT 0 72 ! 0 70 ! 48 G! 4A G! S@ 0 D.
21> F 0 CR LIST 0 0 G>
22> BEGIN KEY DROP FO C@ DUP 1B - WHILE Q IF
23> CASE CB OF P@ W IF B0 ELSE 8 EMIT -1 THEN
24> ELSE O8 OF P@ W IF B0 ELSE 20 P@ 1 - L@ C* C! R THEN
25> ELSE CD OF P@ 40 T IF B0 ELSE 43 G! 1 THEN
26> ELSE DO OF L@ F T IF B0 ELSE 42 G! 100 70 +! 0 THEN
27> ELSE C8 OF L@ W IF B0 ELSE 41 G! -100 70 +! 0 THEN
28> ELSE OD OF L@ F T IF B0 ELSE 0 71 C! 100 70 +! 0 L@ G> 0 THEN
29> ELSE D2 OF 72 @ 1 XOR DUP 72 ! DUP IF 4B G!
30> ELSE -4 L@ G> L@ L@ LIST THEN 30 -1 G>
31> IF ." INSERT" ELSE ." TYPOVR" THEN P@ L@ G> 0
32> ELSE 7F OF 72 @ 0= P@ 0= + IF B0 ELSE P@ 3F -
33> IF P@ L@ C* DUP 1 - OVER 40 L@ C* SWAP - CMOVE THEN
34> 20 3F L@ C* C! R THEN
35> ELSE C9 OF L@ F T IF B0 ELSE MP ROT SWAP CMOVE 0 F ZL THEN
36> ELSE D1 OF L@ F T IF B0 ELSE MP <CMOVE 0 L@ ZL THEN
37> ELSE DROP B0 ENDCASE
38> ELSE P@ 40 - IF DROP B0 ELSE
39> 3E P@ >= 72 @ AND
40> IF P@ L@ C* DUP 1+ DUP 40 L@ C* SWAP - <CMOVE THEN
41> DUP EMIT P@ L@ C* C! 1 THEN THEN
42> 70 +! REPEAT DROP -4 10 G> ; ;S

```

Line-by-Line Commentary  
Source Listing - FORTH Editor for VT52 Emulator

- Line  
No.
- 1 SCR 74 defines the variable SCR as locations 74 and 75 in the Z8 register file. The word S@ places the data at locations 74/75 on the stack. P@ and L@ place a character on the stack; one from register file location 71, the other from register file location 70.
  - 2 G! assumes the second byte of a VT52 control string is on the stack. G! outputs an ESC character (HEX 1B) then the control character on the stack. BO sends a BEL code to the serial port and leaves a zero on the stack that causes the line position processing in line 41 to leave L unchanged (L is the position-in-line variable stored in register file location 70).  
  
G> places the VT52 gotoXY code on the stack and executes G! to issue the ESC gotoXY string, then outputs the row and column codes of the desired screen position.
  - 3 C\* assumes row and column numbers are on the stack and computes an address in the screen block corresponding to them.
  - 4 ;S is an immediate word (a word that can be executed during compilation rather than compiled) that terminates obtaining input from program memory, thus reverting the system to taking FORTH system input from the serial input port.
  - 5 LSCR begins a compilation from the screen defined by SCR. Compilation terminates on encountering a ;S word.
  - 7 .CLEAR places blanks throughout the screen defined by SCR then places the compilation terminator word ;S at the very end of the screen.
  - 8 LIST assumes the beginning and ending line number to be listed are on the stack and lists the indicated lines of the current screen (as indicated by the variable SCR) on the display terminal.
  - 11 ZL assumes the row index of a line to be set to blanks is at top-of-stack and a zero at next-to-top-of-stack (indicating the row is to be zeroed starting from column position zero). ZL zeros the indicated line then lists the page starting from the current line index (L, located at register file location 70). Finally the routine sets the cursor to the first column of the current line.

- 13 MP computes the parameters necessary for the CMOVE and <CMOVE words that are used to move blocks of characters.
- 14 T and W are used in the EDIT CASE logic to determine whether the edit is taking place in insert or type-over mode. The arrow keys and backspace key are not recognized in insert mode. Q tests whether a character received from the keyboard is a control character or not.
- 15 DUMP assumes that the address is on the stack. A 256-byte block of memory is printed in HEX and ASCII, 16 bytes to a line.
- 19 R backs up the screen cursor and erases the byte at that location.
- 20 EDIT presents the screen image of the FORTH page indicated by the address stored at SCR (register file location 74). The prologue to EDIT accomplishes the following:
- o row and column indices are set to zero,
  - o the mode is set to type-over,
  - o the screen is cleared,
  - o the beginning address of the FORTH page being edited is displayed, and
  - o the page is listed on the screen.
- 21 The BEGIN on line 21 pairs with the REPEAT in line 42. This loop repeatedly obtains characters from the serial input line until an ESC (ASCII code 1B) is encountered.
- 22 Because the FORTH word KEY only reads 7 bits of the character being input, it is re-read by the sequence FO @ (register file location FO is the serial I/O buffer). It is then checked to see if it is an ESC character (which terminates the edit). If it is not, the word Q checks the character to see if it lies outside the range of normal text characters.
- 23 The word CASE on line 23 pairs with the ENDCASE in line 37. CASE assumes that a control character is on the stack (This occurred at the DUP in line 22). The constants CB, 08, CD, etc. are the HEX equivalents of the control codes for character left, character right, character up, character down, carriage return, insert, back space, page up, and page down. With each control character except for insert, the row or column position is checked to determine whether the control operation would be meaningful. For example, for a move-left-one-character (code CB) it is not meaningful to attempt to move to the left of line position 0.

- 23 Each one of the CASE options leaves a number on the stack for incrementing the column index. The adjustment of the column index occurs at the very bottom of the processing (line 42). The processing of the code for move-left-one-character (code CB) calls for the column index to be decremented by one.
- 24 The processing for the backspace code (HEX 08) stores a blank (HEX 20) in the FORTH page, then backs up the cursor and stores a blank at that position on the screen (execution of the FORTH word R).
- 26 The processing for the move-cursor-up-one-line (HEX code C8) increments the line index by adding 1 to the high-order byte in the word stored at register file locations 70 and 71.
- 29 The processing for the insert code (HEX 72) first complements the code in location 72 (the insert is a toggle) then, according to whether the current mode is type-over or insert:
- Insert: Erases from the present cursor position to the end of the line (even at 4800 baud the character transmission rate to refresh a line is too slow to be acceptable) and places the message INSERT at the top of the screen.
- Type over: Lists from the present cursor position to the end of the line and places the message TYPOVR at the top of the screen.
- 32 The coding for the delete key (control code 7F) is similar to that for the backspace except that the remainder of the line to the right of the cursor is moved left one space.
- 36 The routine <CMOVE moves a block of characters from one block to another, moving first the last character, then the next to last, and so on. This permits moving a block of characters even when the target area overlaps the end of the source area.
- 38 This section (through line 41) either overlays or inserts a character in the page.
- 42 This line updates the character-in-line index (the byte at location 71), and goes back to line 22 for another character. An exit caused by typing an ESC code (HEX 1B) drops the code from the stack and exits after setting the cursor below the display of the page being edited.

APPENDIX III.

UNSTRUCTURED SOURCE LISTING OF THE PROTOTYPING EDITOR

(INCLUDING HEADERS FOR AUXILIARY FORTH WORDS)

```

CREATE <CMOVE SMUDGE 179B , CREATE READEP SMUDGE 17B5 , CREATE
PROGRAM SMUDGE 1700 , CREATE RDBLK SMUDGE HERE 2 + , 50EC , 50ED ,
50EA , 50EB , D6 C, 187 , D29A , A0EA , 80EC , EBF5 , 3050 , : SCR
74 ; : S@ 74 @ ; : P@ 71 C@ ; : L@ 70 C@ ; : G! 1B EMIT EMIT ; :
B0 7 EMIT 0 ; : G> 59 G! 21 + EMIT 24 + EMIT ; : C* 40 U* DROP +
S@ + ; : ;S BLK @ IF R> DROP THEN ; IMMEDIATE : LSCR >IN @ 70 ! 0
>IN ! S@ 40 U* BLK ! DROP 662 EXECUTE 70 @ >IN ! 0 BLK ! ; :
.CLEAR 20 S@ C! S@ S@ 1+ 3FF CMOVE 3B53 S@ 3FD + ! ; : LIST DO I 0
<# # # #> TYPE ." <|" 40 I C* 3F 0 DO 1 - DUP C@ 20 - IF LEAVE
THEN LOOP DUP FFC0 AND DO I C@ EMIT LOOP 40 I G> ." |" CR LOOP ; :
ZL C* DUP 20 ROT C! DUP 1+ 3F CMOVE -4 L@ G> 4A G! F L@ LIST 0 L@
G> 0 71 ! 0 ; : MP 0 L@ C* DUP 40 + DUP 40 F C* SWAP - ; : T = 72
@ OR ; : W 0= 72 @ OR ; : Q DUP 7F >= OVER 1F SWAP >= + ; : DUMP
70 ! 48 G! 4A G! F 0 DO CR I 10 U* DROP 70 @ + H. ." <" F 0 DO
SPACE I J 10 U* DROP + 70 @ + C@ 0 <# # # #> TYPE LOOP SPACE ."
><" F 0 DO I J 10 U* DROP + 70 @ + C@ 7F AND Q IF 20 EMIT DROP
ELSE EMIT THEN LOOP 3E EMIT LOOP CR ; : R 8 EMIT 20 EMIT 8 EMIT -1
; : EDIT 0 72 ! 0 70 ! 48 G! 4A G! S@ 0 D. F 0 CR LIST 0 0 G>
BEGIN KEY DROP FO C@ DUP 1B - WHILE Q IF CASE CB OF P@ W IF B0
ELSE 8 EMIT -1 THEN ELSE 8 OF P@ W IF B0 ELSE 20 P@ 1 - L@ C* C! R
THEN ELSE CD OF P@ 40 T IF B0 ELSE 43 G! 1 THEN ELSE DO OF L@ F T
IF B0 ELSE 42 G! 100 70 +! 0 THEN ELSE C8 OF L@ W IF B0 ELSE 41 G!
-100 70 +! 0 THEN ELSE D OF L@ F T IF B0 ELSE 0 71 C! 100 70 +! 0
L@ G> 0 THEN ELSE D2 OF 72 @ 1 XOR DUP 72 ! DUP IF 4B G! ELSE -4
L@ G> L@ L@ LIST THEN 30 -1 G> IF ." INSERT" ELSE ." TYPOVR" THEN
P@ L@ G> 0 ELSE 7F OF 72 @ 0= P@ 0= + IF B0 ELSE P@ 3F - IF P@ L@
C* DUP 1 - OVER 40 L@ C* SWAP - CMOVE THEN 20 3F L@ C* C! R THEN
ELSE C9 OF L@ F T IF B0 ELSE MP ROT SWAP CMOVE 0 F ZL THEN ELSE D1
OF L@ F T IF B0 ELSE MP <CMOVE 0 L@ ZL THEN ELSE DROP B0 ENDCASE
ELSE P@ 40 = IF DROP B0 ELSE 3E P@ >= 72 @ AND IF P@ L@ C* DUP 1+
DUP 40 L@ C* SWAP - <CMOVE THEN DUP EMIT P@ L@ C* C! 1 THEN THEN
70 +! REPEAT DROP -4 10 G> ; ;S

```

APPENDIX IV.

AUXILIARY Z8 FORTH WORDS

?BLANK - Determine if EPROM is Blank

Arguments: Top-of-Stack is 1 for 2716 EPROMs, 0 for 2732s.

```
?BLANK IF 7FF ELSE FFF THEN 0
DO I READEP FF - IF ." NOT " LEAVE THEN LOOP ." BLANK" CR ;
```

?COMPARE - Compare EPROM with Memory

Arguments: Top-of-Stack is 1 for 2716 EPROMs, 0 for 2732s.  
Second from top is the address of the block in memory.

```
?COMPARE IF 7FF ELSE FFF THEN 0
DO I READEP OVER I + C@ -
IF ." NOT " LEAVE THEN LOOP DROP ." EQUAL CR ;
```

RDBLK - Read Characters into a Block of Memory

Arguments: Top-of-Stack is the length of the block  
Second from top is the address of the block.

50EC	RDBLK	POP	WRC	Pop Top-of-Stack into working register
50ED		POP	WRD	pair C/D (Character count)
50EA		POP	WRA	Pop word underneath TOS into working
50EB		POP	WRB	register pair A/B (Address of block)
D60187	LHE	CALL	CHARIN	Get a character from serial input port
D29A		LD	@@WRA,WR9	Store the character using A/B as pointer
AOEA		INCW	WRA	Point A/B to next address
80EC		DECW	WRC	Decrement character count in C/D
EBF5		JR	NZ,LHE	Repeat loop if character count not zero
3050		JP	VNEXT	Exit to next FORTH word

```
CREATE RDBLK SMUDGE HERE 2 + , 50EC , 50ED , 50EA , 50EB , D6 C ,
0187 , D29A , AOE A , 80EC , EBF5 , 3050 ,
```

## APPENDIX V.

### ACRONYMS

ASCII	American Standard Code for Information Interchange
BASIC	Elementary compiler used with microcomputers
BAUD	Bits per second
EPROM	Electrically programmable, UV-light erasable read-only memory
FSK	Frequency Shift Keying - A method of sending digital data by means of audio tones
HEX	Hexidecimal, or base 16 (0,1,2, ... 9,A,B,C,D,E,F)
K	Thousand
NAHB/RF	National Association of Home Builders Research Foundation
RAM	Random access memory
ROM	Read-only memory
RS232	An Electronic Industries Assn. standard for serial communication
VT52	Digital Equipment Corp. video display terminal Model 52
UART	Universal asynchronous receiver/transmitter



**Internal Distribution**

- |                     |                                 |
|---------------------|---------------------------------|
| 1. V. D. Baxter     | 24. J. O. Kolb                  |
| 2. K. R. Carr       | 25. R. S. Loffman               |
| 3. J. E. Christian  | 26. Sharon McConathy            |
| 4-13. R. G. Edwards | 27. H. Perez-Blanco             |
| 14. P. D. Fairchild | 28. C. H. Petrich               |
| 15. W. Fulkerson    | 29. P. H. Shipp                 |
| 16. M. B. Gettings  | 30. R. S. Solanki               |
| 17. R. L. Goeltz    | 31. S. S. Stevens               |
| 18. I. G. Harrison  | 32-33. Central Research Library |
| 19. R. B. Honea     | 34. Document Reference Section  |
| 20. A. F. Huntley   | 35-36. Laboratory Records       |
| 21. H. L. Hwang     | 37. Laboratory Records - RC     |
| 22. M. R. Ives      | 38. ORNL Patent Office          |
| 23. R. O. Johnson   |                                 |

**External Distribution**

39. Office of Assistant Manager for Energy Research and Development, Department of Energy, Oak Ridge Operations Office, Oak Ridge, TN 37831
40. Jaime G. Carbonell, Associate Professor of Computer Science, Carnegie-Mellon University, Pittsburgh, PA 15213
41. S. Malcolm Gillis, Dean, Graduate School, Duke University, 4875 Duke Station, Durham, NC 27706
42. Peter Hogarth, P. O. Box 1235, Fairfield, IA 52556
43. Fritz Kalhammer, Vice President, Electric Power Research Institute, P. O. Box 10412, Palo Alto, CA 94303
44. Roger E. Kasperson, Professor of Government and Geography, Graduate School of Geography, Clark University, Worcester, MA 01610
45. Martin Lessen, Consulting Engineer, 12 Country Club Drive, Rochester, NY 14618
- 46-85. David MacFadyen, NAHB National Research Center, 400 Prince Georges Blvd., Upper Marlboro, MD 20772-8731
- 86-115. Technical Information Center, P. O. Box 62, Oak Ridge, TN 37831

