



ORNL/TM-11125

OAK RIDGE  
NATIONAL  
LABORATORY

MARTIN MARIETTA

A Z8 FORTH Assembler  
for SMART HOUSE Prototyping

Reid Gryder

OAK RIDGE NATIONAL LABORATORY  
CENTRAL RESEARCH LIBRARY  
CIRCULATION SECTION  
4800 ROSS ST.  
**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.

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

This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from the Office of Scientific and Technical Information, P.O. Box 62, Oak Ridge, TN 37831; prices available from (615) 576-8401, FTS 626-8401.

Available to the public from the National Technical Information Service, U.S. Department of Commerce, 5285 Port Royal Rd., Springfield, VA 22161  
NTIS price codes—Printed Copy: A04 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

A Z8 FORTH Assembler  
for SMART HOUSE Prototyping

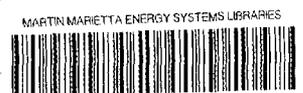
Reid Gryder

Date Published - August 1989

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



3 4456 0313809 4



## CONTENTS

	Page
1. INTRODUCTION . . . . .	1
2. THE Z8 MICROCONTROLLER HARDWARE ARCHITECTURE . . . . .	4
2.1 THE Z8 REGISTER FILE . . . . .	4
2.2 WORKING REGISTER CONCEPT . . . . .	4
2.3 Z8 ADDRESSING MODES . . . . .	5
2.4 PROCESSOR FLAGS . . . . .	6
2.5 Z8 INTERRUPT HANDLING . . . . .	7
2.6 Z8 OPERATIONS AND INSTRUCTIONS . . . . .	8
2.7 WORD INSTRUCTIONS . . . . .	8
2.8 Z8 SYSTEM REGISTERS . . . . .	8
3. THE FORTH ENVIRONMENT . . . . .	10
3.1 USE OF REGISTERS BY FORTH . . . . .	11
3.2 Z8 MEMORY ALLOCATION . . . . .	13
3.3 ASSEMBLY LANGUAGE INTERRUPT HANDLING ROUTINES . . . . .	14
4. WRITING AN ASSEMBLER PROGRAM . . . . .	17
4.1. CONDITION CODES . . . . .	18
4.2. CONDITIONAL AND FLOW CONTROL STATEMENTS . . . . .	18
4.3. USING WORKING REGISTERS . . . . .	21
5. EXAMPLES . . . . .	22
5.1 TIME OF DAY CLOCK . . . . .	22
5.2 SAVE AND RESTORE THE REGISTER POINTER . . . . .	22
5.3 REPLACEMENT FOR LOOP . . . . .	23
6. ERROR MESSAGES . . . . .	25
7. CONCLUSIONS . . . . .	26
APPENDIX I. Z8 INSTRUCTION MATRIX . . . . .	28
APPENDIX II. TABLE OF INSTRUCTIONS . . . . .	29
APPENDIX III. LISTING OF THE Z8/FA . . . . .	35
APPENDIX IV. TIME-OF-DAY CLOCK PROGRAM LISTING . . . . .	37
APPENDIX V. SAVE AND RESTORE WORKING REGISTER PROGRAM LISTING . . . . .	38



## LIST OF TABLES AND FIGURES

	<b>Page</b>
Table 1. Addressing Modes . . . . .	5
Table 2. Processor Flags . . . . .	6
Table 4. FORTH Use of Registers . . . . .	12
Table 5. FORTH System Variables . . . . .	13
Table 6. Z8 Memory Allocation . . . . .	13
Table 7. Condition Codes . . . . .	18
Table 8. Error Messages . . . . .	25
Figure 1. Assembly-Language Code for LOOPCODE . . . . .	24



## **ACKNOWLEDGEMENTS**

The author would like to acknowledge that this document was made possible by the careful tutelage of Robert Edwards, and the continuing interest and assistance of Grimes Slaughter of CONRAY, Inc.



## ABSTRACT

This report presents an extension of the Zilog Z8/FORTH microcontroller system software by defining FORTH words to implement a Z8 assembler. 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 several major corporations that provide products and services to the home building industry. The major goal of the project is to help the participating companies use advanced technology in the development of new residential products for communications, energy distribution, and appliance control.

Use of a high-performance, easily-adaptable microcontroller such as the Zilog Z8 can facilitate the testing of Smart House product prototypes. The device can be used to meet numerous sensing, tracking, and test-suite generation needs, and can provide an important function in system testing by simulating the operation of associated Smart House components.

The intent of this report is to provide the Smart House Project team with information concerning the programming techniques needed for developing and testing Smart House products. The material is technical in nature and assumes considerable experience in microcontroller technology and microcomputer programming.



## 1. INTRODUCTION

This report presents the SMART HOUSE Z8 FORTH Assembler (Z8/FA), a technique extending Zilog Z8/FORTH microcontroller system software for use in rapid prototyping by defining assembler language instructions representing native Z8 microcontroller operations as FORTH words. Previous studies provided an analysis of the rapid-prototyping potential of the Z8/FORTH microcontroller (Edwards, Feb. 1987), and techniques for optimizing the Zilog Z8/FORTH microcontroller for use in rapid-prototyping (Edwards, Sept. 1987). This report documents the work which was performed with the original Z8 hardware and FORTH System. However, at this writing, improved hardware and an optimized FORTH System are being prepared.

The techniques presented in this report were developed 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 that provide products and services to the home building industry. The Project will help the participating corporations use advanced technology in the development of 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 piping and control products into the market for homes constructed in the 1990s.

Certain parts of the designs now used for home electrical and gas distribution systems are as much as 100 years old. Utilities first piped gas to homes in the 1880s as a substitute for oil lamps and candles. A few years later, electricity became available to home owners as a replacement for gas lighting. The use of electricity for purposes other than illumination came after the turn of the century, motivated by a desire to use the excess generating capacity produced during daylight hours. Today, the amount of energy used for lighting is dwarfed by that required for heating and cooling. In spite of the radical change in use patterns and the need for sophisticated control of technologically advanced appliances, the basic energy distribution and control systems in homes have not changed in several decades. As a result, the expectations of modern homeowners are not being realized. 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 can now be accomplished only with difficulty or not at all. 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 before the circuit leading to the appliance will initiate the flow of power. After the circuit is powered, the control for the circuit requires a continuing "nominal-operation" signal from the appliance in order 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 focused on reliability, a leading Smart House design proposal calls for several distributed controllers all wired together, each with the ability to back up one another in case of failure, rather than concentrating home control in one central computer.

Oak Ridge National Laboratory (ORNL) is helping NAHB by providing technical evaluations of proposed Smart House designs, assistance with project management, advice on facilities necessary for design evaluation, and development of prototype equipment for system testing and integration. This report concerns ORNL experience with use of the native Z8 instructions in the Z8/FORTH microcontroller for the Smart House Project. The information is intended to help Smart House Project participants with the use of the microcontroller in the development and test of Smart House products.

In a report by R. G. Edwards on the Zilog/Z8 FORTH-based microcomputer, ORNL determined that the relatively slow speed of Z8 FORTH software (average execution rate of 5000 FORTH instructions/second) may be a constraint that prohibits use of the microcontroller in demanding rapid-prototyping situations (Edwards, Feb. 1987). Another report (Edwards, Sept. 1987), compared the speed of Z8 FORTH with versions of FORTH operating on other microcomputers, and presented techniques for improving the speed of

Z8 FORTH. In that report, Edwards gave examples of an optimized timing loop that used assembly-language code to implement the function. Although the optimized Z8 FORTH bench mark is very fast, achieving rates as much as four times faster than FORTH on an IBM PC, use of Z8/FORTH in this manner is not very different from simply coding in native machine instructions to accomplish the function. Use of native machine code is generally viewed with disdain because it defeats the transportability, commonality, and readability advantages of FORTH. For certain real-time operations, however, execution speed is of utmost importance and even when the original functionality is developed in FORTH, the target system must be coded in optimal machine code for speed.

This paper explores methods of increasing the speed of Z8 FORTH by using Z8 assembly language without sacrificing the advantages of the FORTH language for rapid prototyping. To provide for the fast execution of Z8 machine instructions in a FORTH programming environment, a SMART HOUSE Z8 FORTH Assembler (Z8/FA) has been written in the FORTH language. This Z8/FA defines a number of FORTH words which are somewhat like assembler OP codes. These OP codes cause the machine code of the instruction to be loaded into memory, and when the FORTH word is executed, control is passed to the machine instructions generated by the assembler.

This document is intended to give the programmer who is already fluent in assembly language, and who knows the Z8 hardware, sufficient information about the peculiarities of the Z8/FA to use it without a detailed knowledge of the FORTH language. A knowledge of assembly language programming, and of the Z8 hardware and instruction set is assumed. By using the Z8/FA, one can produce assembly language programs which can be executed in the FORTH environment.

## 2. THE Z8 MICROCONTROLLER HARDWARE ARCHITECTURE

To take advantage of the optimization techniques described in this report, it is necessary to have a working knowledge of the Z8 microcontroller hardware architecture. The definitive reference on this subject is the Zilog Z8 Microcomputer Technical Manual.

In the discussions which follow, address values, constants, and register numbers are in hexadecimal. The mnemonic operation codes all end with a comma, and when used in test, will be written in BOLD FACE. For exaple, **NOP**, is the code for the Z8 null operation which does nothing. It is assumed that the Z8 hardware consists of a Micromint SBC11 stand-alone microcontroller containing a Z8611 microcomputer with a FORTH system in 4K bytes of piggy-back ROM, a Micromint SBC14 16K byte memory expansion board at base address 4000, and a Micromint SBC33 memory expansion II board based at address 8000. Also assumed is the presence of a 2K EPROM containing the Program Editor described in a previous document (Edwards, Feb. 1987).

### 2.1 THE Z8 REGISTER FILE

The Z8 Microcomputer is a Register File Machine. Instead of having a single accumulator in which operations are performed, an entire bank of 128 memory locations can be used as accumulators. These are located in the register file, at addresses 00 to FF. Each of these registers may be the operand of any of the Z8 operation codes. Thus each register may be used as an accumulator, or used as an index register. In addition, a group of 16 registers can be allocated as "Working Registers" and thus accessed with 4-bit addresses and very fast instructions.

### 2.2 WORKING REGISTER CONCEPT

The Z8 working registers are a very powerful feature which allows the Z8 microcomputer to perform certain operations very fast. The Working Register Pointer is located at FD, and when set with an *SRP*,<sup>1</sup> instruction, allows a group of 10 (hexadecimal, 16 decimal) registers to be addressed with only four address bits. A special set of very fast

---

<sup>1</sup>The reader is reminded at this point that all Z8/FA instructions will end in a comma.

instructions are available for the use of this feature. A lower case "r" in an address mode name tells the Z8/FA to use these special instructions.

A form of indexing is available through the working register feature as well. If you use the register designation "En" in a regular register address field, it signals the hardware to use the left half of the working register pointer for the left half of the register address and the "n" of the "En" for the right half of the register address. Thus these instructions, like the working register instructions, can refer to different registers each time they are executed, according to what value is stored in the working register pointer.

The Z8/FA allows the use of the register names, "R0", "R1", to "RF" to refer to working registers in normal register instructions. Thus,

**60 SRP, R3 44 RR LD,**

loads register 63 from register 44.

### 2.3 Z8 ADDRESSING MODES

The Z8 microcomputer has several addressing modes. In coding for the Z8/FA, these modes must be explicitly stated. Thus the "RR" in the previous example refers to register-to-register addressing mode.

The complete list of addressing modes which can be used with the Z8/FA and the codes for indicating them is given in Table 1.

Table 1. Addressing Modes

CODE	Addressing Mode
R	Register addressing
IR	Indirect register addressing
rr	Working register to working register
rIr	Indirect working register to working register
RR	Register to register
RIR	Indirect register to register
RIM	Immediate to register
IRIM	Immediate to Indirect register

For example,

```
10 7F RR LD,
    loads the content of register 7F into register 10
10 7F RIM LD,
    loads the hex constant 7F into register 10
7 R INC,
    increments the contents of register 7
7 rINC,
    increments the contents of working register 7
```

(note that because no addressing mode code was defined for operating on a single working register, a special OPcode (rINC,) was created for the working register increment instruction)

An earlier version of the pseudo-assembler incorrectly assigned the mnemonic IRR for RIR. Until such time as programs developed using the early version have been corrected, both mnemonics will be accepted.

## 2.4 PROCESSOR FLAGS

The Z8 register at location FC is the flag register, and bits in this register are set and cleared by certain CPU operations to indicate the status which results from that operation. The flags are shown in table 2. Section 4.1 shows the names which the Z8/FA has given to the various conditions indicated by these flags.

Table 2. Processor Flags

<u>Description</u>	<u>Bit</u>
User Flag 1	0
User Flag 2	1
Half Carry Flag	2
Decimal-adjust Flag	3
Overflow Flag	4
Sign Flag	5
Zero Flag	6
Carry Flag	7

## 2.5 Z8 INTERRUPT HANDLING

The Z8 supports six levels of vectored interrupts, IRQ0 through IRQ5. The four Port 3 input lines (P30-P33) are used to obtain external interrupts. These external interrupts are negative edge triggered. Serial In, Serial Out, and the two Counter/Timers are available as internal interrupts.

The interrupt protocol is controlled by the Interrupt Priority Register (register file location F9, Write Only), the Interrupt Mask Register (register file location FB, Read/Write), and the Interrupt Request Register (register file location FA, Read/Write).

The Interrupt Priority Register (F9) separates the interrupts into three groups (A, B, C) of two. It sets the relative priority within the group and the relative priority of the groups.

The Interrupt Mask Register (FB) determines which interrupts are enabled. Bits 0-5 control whether the respective IRQs (set=enabled, cleared=disabled) result in transfer to an interrupt routine when the corresponding interrupt occurs. Bit 7 is a global control determining whether all interrupts are enabled (set) or disabled (reset). Bit 7 must be set with an EI, instruction and cleared with a DI, instruction.

Bits 0-5 in the Interrupt Request Register (FA, Read/Write) are set by the respective incoming interrupt signals. If interrupts are enabled (bit 7 of FB is set), and an interrupt bit is set in FA, what happens is determined by corresponding bits in FB, the Interrupt Mask Register. If the corresponding bit is set in FB, the following things happen:

- 1) the Program Counter and the Flag Register are pushed onto the stack;
- 2) the corresponding bit in FA is cleared;
- 3) bit 7 in FB is cleared, disabling interrupts; and
- 4) an indirect Jump is taken using the address at the corresponding interrupt vector location of program memory.

Memory locations 00-01, 02-03, ... 0A-0B correspond to vectors for the respective IRQs. This address is the address of the interrupt service routine. The last instruction in the interrupt service routine has to be an IRET, (return from interrupt). An IRET, causes the Flag Register and Program Counter to be popped from the stack, bit 7 of FB is set enabling interrupts, and program execution to resume where it left off.

See Section 3.3 for information on how to code interrupt handling routines.

## 2.6 Z8 OPERATIONS AND INSTRUCTIONS

Appendix A is a matrix of instructions showing the mnemonic and the address mode code for each of the instructions. Appendix B is a list of all the instructions, showing which are implemented, and giving a sample instruction together with the hex machine code generated by the instruction.

## 2.7 WORD INSTRUCTIONS

Two Z8 instructions operate on a full 16-bit word instead of an 8-bit single register. These instructions are **INCW**, which increments a pair of bytes, and **DECW**, which decrements a pair of bytes. These instructions require an even address, and modify that location and the next location. The names "W0", "W2", to "WE" have been given to the words which represent the legal working register addresses for these instructions. For example,

**W6 R DECW,**

will subtract one from the 16-bit value stored in working register pair 6 and 7.

## 2.8 Z8 SYSTEM REGISTERS

A listing of the Z8 System Registers, together with the initial value loaded into them by the Micromint Z8 FORTH System, is given in Table 3. A revision of this document will describe the changes which have been made to accommodate a new optimized FORTH System now under development.

Table 3. Z8 System Registers

Register	Register	Source	Initial	
Location	Name	Table	Value	Description
F0	Serial I/O	B5	20	Initialized to blank
F1	Timer Mode	B6	00	Disable both counters
F2	Counter/Timer1	B7	F0	Initialized to 240
F3	Prescaler 1	B8	03	Initialized to 1
F4	Counter/Timer2	B9	01	Initialized for 9600 baud
F5	Prescaler 0	BA	27	11.0592 crystal
F6	Port 2 Mode	BB	FF	Initialized as input
F7	Port 3 Mode	BC	41	Serial I/O, Port2 Pull-ups on
F8	Ports 0 & 1 Mode	BD	B2	Adr/data on 0/1, wait state on
F9	Interrupt Priority	BE	2B	3>5/4>1/2>0 A>C>B
FA	Interrupt Request	BF	10	Interrupt: serial output
FB	Interrupt Mask	C0	00	Interrupts disabled
FC	Conditions Flag	C1	00	All flags cleared
FD	Register Pointer	C2	50	Register Group 50
FE	Stack Pointer (H)	C3	1F	FORTH stack from 1F00 down
FF	Stack Pointer (L)	C5	00	

### 3. THE FORTH ENVIRONMENT

Very little of the FORTH environment needs to be understood by the assembler programmer. FORTH is a language which uses subroutines called "words." When a "word" is defined, it is placed in a "dictionary." A "word" in the "dictionary" is executed by typing its name on the keyboard, or by interpreting a program containing the word. FORTH uses a "stack" to communicate between the "words."

The first rule of programming for the FORTH Z8 Prototyping Computer is, "Never take anyone's word for anything!" It is so easy to verify the behavior of a particular word, and so difficult to adequately describe its behavior, that there is no excuse for not verifying the behavior of any definition prior to incorporating it into your own program. You will be wise to follow this rule for any code you develop, and code you get from a friend or a book, and any code you find in this document. If you have not personally verified its accuracy, don't use it!

The easiest and most common mistake to make is to leave an extra word on the stack, or to dispose of one which should have been left. The "stack effect" of a definition should be well understood prior to using it.

FORTH programs are coded on pages called "screens." A simple editor that allows modification of these "screens" is described by Edwards (Sept. 87). Before the words on a "screen" can be executed, the "screen" must be loaded (into the dictionary) with the LSCR command. The end of a program must be marked with a FORTH "screen terminator" word, ";S". All FORTH words must be preceded and followed by a space in order for the FORTH interpreter to recognize them.

The FORTH word ":" is used to define a new FORTH word. For instance, if we write a screen which says:

```
: FOO BRING COFFEE ; ;S
```

and LSCR that screen into the dictionary, we will have defined a FORTH word named "FOO" that when executed, will execute the FORTH words "BRING" and "COFFEE". The ":" begins the definition, the name of the word being defined comes next, and the definition is terminated by ";". Several such definitions could be on the screen before

the screen terminator ";S". In this example, the words "BRING" and "COFFEE" must have been previously defined or an error will occur and "FOO" will not be defined.

Leo Brodie has written an excellent book on the FORTH language, Starting Forth, which includes an example of a FORTH assembler for the 8080 processor. For this document, only the features of the Micromint Z8 FORTH system, together with the Editor described in a previous ORNL report (Edwards, Feb. 1987), are assumed.

Some knowledge about the organization of the Z8 FORTH software stored in the microcomputer's internal ROM is required to understand this subject. A previous ORNL report (Edwards, Sept. 1987) provides an overview of that structure, and provides a complete listing of the Micromint Z8 FORTH system in an Appendix.

FORTH has been described as a "write-only" language. This statement has its roots in several aspects of the language. The use of a stack to communicate between routines produces code which is not very readable, and the various manipulations of the stack which are required to place the parameters in the proper order are often even more confusing. One cannot pick up a FORTH program and start reading it from the middle. The following hints may assist you in reading programs.

First, it is absolutely essential that you know the "stack effect" of every FORTH word. That is, you must know what a FORTH word expects to find on the stack, and what values it leaves on the stack. You cannot read a program without that information. Second, the most useful tool in following the logic of a FORTH program is the "stack trace." A stack trace is a chart where the content of the stack is noted before and after execution of each word.

It is very dangerous to mess with FORTH's stack while in an assembly language program. It is possible to **PUSH**, items temporarily onto the stack and **POP**, them back off before leaving the assembly language program, but other use of the stack is not advised.

### 3.1 USE OF REGISTERS BY FORTH

When working in the FORTH environment, the assembly language programmer must be aware of the registers available for use. Some of the registers contain FORTH systems

variables, and if those registers are used, the contents must be restored to their original condition in order for FORTH to continue to operate. Tables 4 and 5 give the definitions of the register file locations for system variables used by FORTH.

Table 4. FORTH Use of Registers

<u>Register File Address</u>	<u>Allocation</u>
00 - 03	Ports 0 - 3
04 - 0F	FORTH Interrupt Vectors
04	Vector for level 0 interrupt
06	Vector for level 1 interrupt
08	Vector for level 2 interrupt
0A	Vector for level 2 interrupt
0C	Vector for level 4 interrupt
0E	Vector for level 5 interrupt (T1)
10 - 1F	FORTH System Variables
20 - 2F	FORTH System Variables
30 - 3F	16 Available Bytes
40 - 4F	16 Available Bytes
50 - 5F	FORTH Working Registers
52-53	Not used by FORTH
60 - 6F	16 Available Bytes
70 - 77	8 bytes used by FORTH Editor
78 - 7F	FORTH System Variables
80 - EF	Write Only Memory (WOM) (writing to these addresses has no effect, and reading gives you the address itself)
F0 - FF	Z8 System Registers

Table 5. FORTH System Variables

Register Location	FORTH Variable Name	Source Table	Initial Value	Definition
10	DPL	00C5	0000	Places to right of dec point
12	H	00C7	1000	Dictionary Pointer (beginning of RAM)
14	>IN	00C9	0000	Buffer Index Pointer
16	BLK	00CB	0000	Mass Storage Hook
18	BASE	00CD	0010	Current Number Base
1A	TIB	00CF	0000	Terminal Input Buffer Pointer
1C	STATE	00D1	0000	Zero if EXEC, CO if Compiling
1E	HLD	00D3	0000	Pointer to No. being converted
20	SO	00D5	0000	Initial Value of Dstack
22	RO	00D7	0000	Initial Value of Rstack
24	WIDTH	00D9	0003	Name Width in Dictionary Entry
26	WARNING	00DB	0000	Zero or address of error trap
28	CONTEXT	00DD	002C	Link to context vocabulary
2A	CURRENT	00DF	002C	Link to current vocabulary
2C	VOCLINK	00E1	0F6E	Pointer: Last Dictionary Entry
2E	FENCE	00E3	0FFF	Lowest address used in FORGET
78-79				Used for stack balancing
7A-7B				FORTH Rstack pointer
7C-7D				FORTH Instruction Pointer
7E-7F				FORTH Execution Vector

### 3.2 Z8 MEMORY ALLOCATION

The addressable memory in the system we are describing is allocated as given in Table 6.

Table 6. Z8 Memory Allocation

0000 - 00FF	Register File
0000 - 0FFF	FORTH System
1000 - 17FF	Editor
1800 - 1FFF	Stacks & Temporary RAM
2000 - 3FFF	Non-existent
4000 - B777	Dictionary
B800 - FFFF	Special Hardware

### 3.3 ASSEMBLY LANGUAGE INTERRUPT HANDLING ROUTINES

The Z8 hardware support for interrupt handling is described in section 2.5. The interrupt protocol is controlled by the Interrupt Priority Register (F9), the Interrupt Mask Register (FB), and the Interrupt Request Register (FA). The sequence of interrupt initialization is rigidly prescribed. Interrupts must be disabled with a **DI**, command and the interrupt control registers loaded in the order 1) the Interrupt Priority Register, 2) the Interrupt Mask Register, and 3) the Interrupt Request Register. Before interrupts are enabled with an **EI**, command, the address of a valid interrupt routine must be loaded into the register pair corresponding to the vector for interrupts being enabled (see Table 4).

The Z8 supports six levels of vectored interrupts, IRQ0 through IRQ5. The interrupts are in three groups (A, B, C) of two interrupts each. Group A is IRQ3 and IRQ5, group B is IRQ0 and IRQ2, and group C is IRQ1 and IRQ4. The Interrupt Priority Register sets the relative priority within the group and the relative priority of the groups.

For example, in order to have  $IRQ0 > IRQ2$ ,  $IRQ3 > IRQ5$ ,  $IRQ4 > IRQ1$ , and Group priority  $B > A > C$ , register F9 must be loaded with the value 3E.

The Interrupt Mask Register (FB) determines which interrupts are active. Bits 0-5 control whether the respective IRQs (set=active, reset=nonactive) result in transfer to an interrupt routine when an interrupt occurs. Bit 7 is a global control for all interrupts. Note, however, bit 7 is not to be set or cleared by writing directly to register FB! Interrupts must be enabled with an **EI**, command or disabled with a **DI**, command. **EI**, and **DI**, do other things in addition to setting or resetting Bit 7.

For example, if you are using only IRQ0, the value 01 should be set in register FB, and interrupts enabled with an **EI**, command. Register FB will then contain the value 81.

Bits 0-5 in the Interrupt Request Register (FA) are set by the respective incoming interrupt signals. If interrupts are globally enabled, what happens when an interrupt bit is set in FA is determined by corresponding bits in the Interrupt Mask Register (FB). If the corresponding bit is set in FB, the hardware detects the interrupt and vectors to the

appropriate interrupt service routine. If the corresponding bit in FA is cleared, the hardware takes no action.

Note well that bit 4 in FA, the Serial Output Complete interrupt, must never be cleared or the system will enter a tight loop waiting for it to be set, but it never will be. FA is loaded with 10 Hex in the FORTH system initiation procedure.

The Interrupt Request Register (FA) permits using either polled or software interrupts. To accomplish a software interrupt, a program sets a bit in (FA), and if the corresponding bit is set in the Interrupt Mask Register (FB), the hardware detects the interrupt and vectors to the appropriate interrupt service routine.

For polled interrupts, the corresponding bits in the Interrupt Mask Register (FB) are cleared to prevent the hardware from transferring to the interrupt service routine, and the Interrupt Request Register (FA) is interrogated using a test-under-mask (TM,) instruction. The software may take any action deemed appropriate.

Program memory locations 00-01, 02-03, ... 0A-0B in the FORTH system EPROM correspond to vectors for the respective IRQs. When an enabled interrupt request occurs, the Program Counter and the Flag Register are pushed onto the stack. The corresponding IRQ bit is reset in (FA), interrupts are disabled (Bit 7 in FB is cleared), and an indirect Jump is taken using the address for the corresponding interrupt in the EPROM memory. This address is the address of the interrupt service routine.

In order to be able to dynamically change the address of the interrupt handling routines, the FORTH system EPROM has placed the address of special interrupt service routines in the read-only program memory locations corresponding to the interrupt vector. These special routines are nothing more than indirect jumps through a register file address. The register file locations, which are easily modified, are used to vector to the starting address of the actual interrupt service routine.

Since the register file addresses 00 to 03 correspond to the Z8 hardware ports, these cannot be used for the service routine addresses. Thus, the register pair 04-05 was selected to contain the address of the IRQ0 interrupt service routine. Registers 06-07

contain the address of the IRQ1 service routine, and so on to register 0E-0F for the IRQ5 routine address.

The last instruction in the interrupt service routine must be an IRET, (return from interrupt). An IRET, causes the Flag Register and Program Counter to be popped from the stack, and program execution to resume where it left off. IRET, also re-enables interrupts by setting bit 7 of the Interrupt Mask Register (FB).

Section 5.1 gives an example of a time-of-day clock using an internal timer and an interrupt handling routine.

Interrupt handling routines gain control when a hardware interrupt occurs, which may happen while a FORTH word is executing. Thus an interrupt routine must take care either to not disturb the environment, or to save it and restore it prior to exiting. Section 3.1 gives information about which registers are used by the Micromint Z8 FORTH System.

Execution speed is usually very important in interrupt routines. Either the process causing the interrupt must be handled promptly, or the event occurs so frequently that it must be completely handled in a very short time in order to have any time for other processing. Thus, interrupt routines are usually written in *optimized assembly language*.

Since execution speed in interrupt handling is important, one should take advantage of the machine instructions which are fastest. On the Z8, this should include the working register instructions, which require that the register pointer be set to a block of working registers. Unfortunately, FORTH uses certain registers which the interrupt routine must either save and restore, or not use (See Table 4). Also unfortunate is the fact that the Set Register Pointer instruction is a unique instruction for each block of registers, and does not take an argument. This results in the situation that either the instruction must be modified in line (a poor programming practice, especially in read only memories) or the setting must be saved and tested to determine which of a set of instructions should be executed in order to restore the previous condition. An example of the latter technique is included in Section 5.2.

#### 4. WRITING AN ASSEMBLER PROGRAM

The implementation of the Z8/FA defines the FORTH word ":", (colon comma) to be "define an assembly language program". Note that we have chosen to use a familiar term ":" (colon) with a "," (comma) following it. This convention will be used throughout the Z8/FA. For example, "LD," will be a FORTH word which behaves like the "LD" machine OP code. The screen

```
:, FOO 30 SRP, 60 6F RR LD, NOP, EXIT, ;S
```

defines the FORTH word "FOO" to be the instructions

```
3130 ( SRP 30 )
E4606F ( LD 60,6F )
FF ( NOP )
3050 ( JP 50 )
```

in machine code. Note that all Z8/FA instructions have a "," (comma) after the assembler OP codes, and also note that the parameter or argument to the OP code precedes the OP code in the FORTH version rather than the conventional method of following the OP code with the parameters. These conventions are followed throughout the Z8/FA.

Also note in the previous example the "EXIT," instruction, which for the Micromint Z8 FORTH System translates into a "JP 50" instruction. This is absolutely essential for any assembler program to be executed as a FORTH word, because it is the way to re-enter the FORTH environment. Forget to terminate your assembly language definition with that command, and you can never get back to FORTH!

When an OP code has more than one argument, the destination (the one whose value is changed) is always listed first. For example,

```
60 6F RR LD,
```

is the instruction to load the contents of register 6F into register 60. Only the content of the first Z8/FA operand is changed by any Z8 instruction.

Finally, notice that the address mode (if necessary) immediately precedes the OP code. Appendix I shows the allowable address modes for each OP code.

## 4.1. CONDITION CODES

Logic and conditional jump instructions are controlled by the C, Z, S, and V flags. The four-bit Condition Code names which have been defined to test for these flags are shown in Table 7. Note that condition codes are always in lower case.

Table 7. Condition Codes

<u>CODE</u>	<u>Meaning</u>
carry	carry
nc	no carry
z	zero
nz	non zero
pl	plus
mi	minus
ov	overflow
nov	no overflow
eq	equal
ge	greater than or equal
lt	less than
gt	greater than
le	less than or equal
uge	unsigned greater than or equal
ult	unsigned less than
ugt	unsigned greater than
ule	unsigned less than or equal
ne	not equal
always	always true

## 4.2. CONDITIONAL AND FLOW CONTROL STATEMENTS

The Z8/FA does not have any way of defining a label which can be used in an assembler instruction. Consequently, the Jump instructions, both to absolute and relative addresses, have no meaning. There are, however, some FORTH-like assembler words which compile to these instructions. Among these are the

```
DO, LOOP,  
IF, ELSE, THEN,  
BEGIN, UNTIL,  
BEGIN, WHILE, REPEAT, and  
BEGIN, AGAIN,
```

logic structures.

Structured Logic Control Commands are implemented in the Z8/FA in a manner very similar to the way FORTH implements branching and nesting. These logic commands allow the user to create loops and conditional branches in a fully structured way without having to use jump instructions.

#### 4.2.1 DO, and LOOP, Instructions

**DO**, and **LOOP**, are assembly language commands which allow you to create a set of instructions which is executed repeatedly. The primary difference between these assembly language commands and the corresponding FORTH commands is that the assembler requires that you set up your loop index, counter and comparison logic yourself. By executing an instruction which causes the Z8 flags to be set, you simulate the effects of a FORTH DO loop. For example,

```
R0 7 LDIM, DO, . . . R0 R DEC, LOOP, . . .
```

will execute the instructions with R0 containing first a seven, then a six, and so on. The last pass through the loop will occur when R0 contains a one, because when R0 is decremented to zero, the **LOOP**, instruction will not branch back for another loop.

**DO**, and **LOOP**, are normally used with a counter of some variety. However the instructions between the **DO**, and the **LOOP**, will continue to be executed until the ZERO flag in the flags register is set when the **LOOP**, is encountered.

#### 4.2.2 cc IF, [ELSE,] THEN, Instructions

The **IF**, **ELSE**, **THEN**, construct differs from FORTH in that you must specify a condition code prior to the **IF**, command, and this condition code, together with the status of the flags register determine which branch is taken.

The **IF**, instruction must be preceded by a condition. The **IF**, instruction looks at the flags register. If the condition specified before the **IF**, is true, the instructions following the **IF**, statement are executed next. If the condition is not true, then the instructions following the corresponding **ELSE**, (if any) are executed. In either case, the next instructions are taken from behind the corresponding **THEN**, statement.

#### 4.2.3 BEGIN, cc UNTIL, Instructions

The **BEGIN, . . . cc UNTIL**, instruction pair is used to bracket a group of instructions which will be executed repeatedly until the condition prescribed by the condition code preceding the **UNTIL**, is true when the **UNTIL**, is encountered. As long as R0 and R1 have the same value, the instruction in the following loop will continue to be executed.  
**BEGIN, ( . . ) R0 R1 rr CP, ne UNTIL,**

#### 4.2.4 BEGIN, AGAIN, Instructions

This pair of instructions defines a loop which continues forever, unless you have provided for some other exit.

**BEGIN, ( . . ) AGAIN,**  
executes the instructions between the **BEGIN**, and the **AGAIN**, repeatedly.

#### 4.2.5 BEGIN, cc WHILE, REPEAT, Instructions

These instructions define a loop which is executed repeatedly until the specified condition code is false when the **WHILE**, is encountered. For example,

**BEGIN, ( . . ) F0 1B RIM CP, ne WHILE, ( . . ) REPEAT, ( . . )**  
executes the code after **BEGIN**,. If the content of F0 (the serial input buffer) is not 1B (escape), the code between **WHILE**, and **REPEAT**, is executed and it branches back to **BEGIN**,; if the serial input buffer contains an escape, execution resumes with the code after **REPEAT**,.

#### 4.2.6 Additional Logic Instructions

All of the above instructions are implemented as lower-case OP-codes as well as those described above (e.g., **begin**, **again**, are functionally the same as **BEGIN**, **AGAIN**,). The effect of the lower-case version is identical with the upper-case version, with the exception that a Jump-relative (ccB) instruction is generated instead of a Jump-direct (ccD) instruction. These instructions take less space and execute faster than their upper-case equivalent.

Since all the lower-case logic instructions use a Jump-Relative (ccB) instruction, it is important to be sure that the distance between the jump instruction and the point it is supposed to reach is less than 127 bytes. The Z8/FA does not warn you if this limit is exceeded. It happily generates code to branch to some very unexpected location.

#### 4.2.7 EXIT, Instruction

The EXIT, instruction is provided as a means of returning from an assembly language subroutine to the FORTH program which called it. In this version of the assembler, EXIT, is equivalent to an indirect jump to the address in register pair 50-51, which is where the Micromint version of the FORTH system keeps the address of the routine to process the next word (Micromint 1984).

### 4.3 USING WORKING REGISTERS

An assembly language routine may have several reasons for setting the register pointer. First, if very fast access to the Z8 ports is required, the **00 SRP**, instruction can be used to provide access to the ports with working register instructions.

Other likely values for the register pointer are 30, 40, and 60 because at each of these locations is a block of 10 hexadecimal (16 decimal) registers which are not used by the FORTH system.

Finally, it should be possible to use the instruction **"F0 SRP,"** to set the register pointer to access the system registers stored at F0-FF. This would enable very fast initialization of the Z8 system registers. Note that any value between 80 and E0 would not make sense for a register pointer, because these register file locations are not implemented for the Z8.

The FORTH system uses working registers when interpreting FORTH words. Thus any assembly language routine which uses working registers must restore the register pointer to the value it was upon entry before exiting to FORTH or dismissing the interrupt. A routine to perform this function is described in section 5.2.

## 5. EXAMPLES

### 5.1 TIME OF DAY CLOCK

An example of a time-of-day clock running on Timer T1 and IRQ5 is given in the appendix. This example keeps the hours in register 33, minutes in 32 and seconds in 31. Register 30 is used to count up to one second. Two FORTH words are defined by this example. `SETTIME` takes three parameters following the word. The parameters are Hours, Minutes and Seconds in decimal, and the clock is set to that time. The word `TIME` requires no arguments, and prints the current value of the clock.

```
SETTIME 8 15 0    sets the clock to 8:15. Five seconds later, the command
TIME              causes the printout
```

```
Time is 8:15:5
```

The listing of the programs is given in appendix IV.

### 5.2 SAVE AND RESTORE THE REGISTER POINTER

The instruction `50 SRP`, is used to set the register pointer to access the registers 50 to 5F. FORTH uses 50-5F as working registers in almost all cases. When the FORTH `<BUILD DOES>` structure is used, working registers are set to 70-7F. Thus any assembly language routine which uses working registers must restore the register pointer to the value it was upon entry before exiting to FORTH or dismissing the interrupt. Following is an example of how you invoke a routine to perform this function. The routine requires that you push the current value of the working register pointer onto the stack. When you wish the register pointer restored to its previous value, pop the old value into register 6E and call the routine.

```
:, SAVE_REGISTER_POINTER_EXAMPLE
  FD R PUSH, nn SRP,

  " Use Working Registers nn Here "

6E R POP, ' RST CALL, EXIT, (or IRET,)
```

This example also demonstrates the use of the `CALL`, and `RET`, instructions. Appendix V contains a listing of the RST routine.

### 5.3 REPLACEMENT FOR LOOP

Edwards (Sept. 1987) examined the code used by Z8 FORTH for loop control, and showed the criticality of the auxiliary assembly-language routine used to implement the FORTH word LOOP. In that document, Edwards proposed a replacement FORTH word LOOPCODE for the assembly language code required to execute the FORTH word LOOP. Edwards constructed a new FORTH definition to link to the new assembly language routine LOOPCODE. This definition of LOOP was identical to the original except for linkage to the optimized routine LOOPCODE rather than the code for LOOP stored in internal ROM. Edwards found the redefined word LOOP to be 3.8 times faster, but it occupies about 30 bytes more than the definition provided in Z8 FORTH internal ROM. The Z8/FA code for Edwards' LOOPCODE is shown in Figure 1.

Z8 Code	Z8/FA Assembly Language	Comments
-----	-----	-----
	:, LOOPCODE	
3170	70 SRP,	Use Register file locations 70-74
4C70	R4 70 LDIM,	
C34A	R4 WA LDCI,	Load index to W0 from RSTACK
C34A	R4 WA LDCI,	
C34A	R4 WA LDCI,	Load limit to W2 from RSTACK
C23A	R3 WA LDCI,	
A0E0	W0 R INCW,	Increment index
2231	R3 R1 rr SUB,	Subtract index from LIMIT
3220	R2 R0 rr SBC,	
7B1B	ge if,	Exit loop if updated INDEX exceeds LIMIT
80EA	WA R DECW,	Point back to INDEX on RSTACK
80EA	WA R DECW,	
D21A	WA R1 STC,	Push updated INDEX back on RSTACK
80EA	WA R DECW,	
D20A	WA R0 STC,	
C20C	R0 WC LDC,	Move IP adjustment to W0
A0EC	WC R INCW,	
C21C	R1 WC LOC,	
80EC	WC R DECW,	Point back to high byte of adjustment
02D1	RD R1 rr ADD,	Add adjustment to IP
12C0	RC R0 rr ADC,	
3050	EXIT,	
	then,	LOOP EXIT
A0EC	WC R INCW,	Skip over IP adjustment
A0EC	WC R INCW,	
A0EA	WA R INCW,	Point RSTACK back to low byte of INDEX
3050	EXIT,	

Figure 1. Assembly-Language Code for LOOPCODE

For comparison, Edwards' definition of the FORTH word LOOPCODE was:

```
CREATE LOOPCODE SMUDGE 3170 , 4C7D , C34A , C34A , C34A , C23A , A0E0 , 2231 , 3220  
, 7B18 , 80EA , 80EA , D21A , 80EA , D20A , C20C , A0EC , C21C , 80EC , 02D1 , 12C0  
, 3050 , A0EC , A0EC , A0EA , 3050 ,
```

## 6. ERROR MESSAGES

The Z8/FA has only one error message,

ERROR # <hex number>

The meaning of the hex number is shown in Table 8:

Table 8. Error Messages

<u>VALUE</u>	<u>MEANING</u>
4	Invalid address mode code used with an instruction which requires an R or IR address mode.
5	Invalid address mode code for an instruction referencing two registers.
7	Invalid address mode code for an LD, instruction.

## 7. CONCLUSIONS

This report has presented a FORTH Assembler for the Z8 microcomputer to facilitate program development for SMART HOUSE rapid prototyping. The Z8 FORTH system provides the capability to develop features and immediately see the results of the development. If the results are unsatisfactory, another approach may be tried without the penalty of a delay imposed by a "batch mode" system. This means that many different options may be tried, and only those which are desirable are kept. This is the essence of rapid prototyping.

In applications like those present in the SMART HOUSE, it is frequently desirable to improve the performance of FORTH words or interrupt service routines by coding them in the native machine code of the processor being used. This can be accomplished by maintaining the actual binary codes for the instructions, but that is a difficult process. Lack of a good "binary editor" makes the process, for all practical purposes, impossible.

Code written in a mnemonic assembly language is easier to understand and modify than code written in binary or hexadecimal. By defining mnemonic FORTH words to represent the features of the Z8 processor, one can make the features of the processor available to a programmer through a text file which can be maintained with any good text editor.

The Z8 FORTH Assembler presented here has proven useful in improving the performance and functionality of prototypes of SMART HOUSE functions. It makes possible extremely rapid execution rates through replacement of entire groups of FORTH words with optimized machine code, and it enables the programmer to substantially improve execution speed of interrupt service routines while maintaining the readability of the program.

## REFERENCES

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

Edwards, R., *Optimizing the Zilog FORTH Microcontroller for Rapid Prototyping*, ORNL/TM-10463, Oak Ridge National Laboratory, September 1987.

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

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

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

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	DEC R	DEC IR	ADD rr	ADD rIr	ADD RR	ADD RIR	ADD RIM	ADD IRIM	LDWR	STWR	DJNZ	JPRA	LDIM	JPDA	INC	
1	RLC R	RLC IR	ADC rr	ADC rIr	ADC RR	ADC RIR	ADC RIM	ADC IRIM								
2	INC R	INC IR	SUB rr	SUB rIr	SUB RR	SUB RIR	SUB RIM	SUB IRIM								
3	JP	SRP	SBC rr	SBC rIr	SBC RR	SBC RIR	SBC RIM	SBC IRIM								
4	DA R	DA IR	OR rr	OR rIr	OR RR	OR RIR	OR RIM	OR IRIM								
5	POP R	POP IR	AND rr	AND rIr	AND RR	AND RIR	AND RIM	AND IRIM								
6	COM R	COM IR	TCM rr	TCM rIr	TCM RR	TCM RIR	TCM RIM	TCM IRIM								
7	PUSH R	PUSH IR	TM rr	TM rIr	TM RR	TM RIR	TM RIM	TM IRIM								
8	DECW R	DECW IR														DI
9	RL R	RL IR														EI
A	INCW R	INCW IR	CP rr	CP rIr	CP RR	CP RIR	CP RIM	CP IRIM								RET
B	CLR R	CLR IR	XOR rr	XOR rIr	XOR RR	XOR RIR	XOR RIM	XOR IRIM								IRET
C	RRC R	RRC IR	LDC	LDCI				LDX								RCF
D	SRA R	SRA IR	STC	STCI	CALL IRR		CALL DA	STX								SCF
E	RR R	RR IR		LD rIr	LD RR	LD RIR	LD RIM	LD IRIM								CCF
F	SWAP R	SWAP IR		STR Irr		STR IRR										NOP

APPENDIX I. Z8 INSTRUCTION MATRIX

APPENDICES

## APPENDIX II. TABLE OF INSTRUCTIONS

Hex Code	Desti- nation	Source	Sample Instruction	Machine Code
00	R		60 R DEC,	00 60
01	IR		61 IR DEC,	01 61
02	r	r	R2 R3 rr ADD,	02 23
03	r	Ir	R4 R5 rIr ADD,	03 45
04	R	R	66 67 RR ADD,	04 67 66
05	R	IR	68 69 RIR ADD,	05 69 68
06	R	IM	6A 01 RIM ADD,	06 6A 01
07	IR	IM	6B 02 IRIM ADD,	07 6B 02
10	R		60 R RLC,	10 60
11	IR		61 IR RLC,	11 61
12	r	r	R2 R3 rr ADC,	12 23
13	r	Ir	R4 R5 rIr ADC,	13 45
14	R	R	66 67 RR ADC,	14 67 66
15	R	IR	68 69 RIR ADC,	15 69 68
16	R	IM	6A 01 RIM ADC,	16 6A 01
17	IR	IM	6B 02 IRIM ADC,	17 6B 02
20	R		60 R INC,	20 60
21	IR		61 IR INC,	21 61
22	r	r	R2 R3 rr SUB,	22 23
23	r	Ir	R4 R5 rIr SUB,	23 45
24	R	R	66 67 RR SUB,	24 67 66
25	R	IR	68 69 RIR SUB,	25 69 68
26	R	IM	6A 01 RIM SUB,	26 6A 01
27	IR	IM	6B 02 IRIM SUB,	27 6B 02
30		R	50 JP,	30 50
31	(FD)	R	60 SRP,	31 60
32	r	r	R2 R3 rr SBC,	32 23
33	r	Ir	R4 R5 rIr SBC,	33 45
34	R	R	66 67 RR SBC,	34 67 66
35	R	IR	68 69 RIR SBC,	35 69 68
36	R	IM	6A 01 RIM SBC,	36 6A 01
37	IR	IM	6B 02 IRIM SBC,	37 6B 02

Hex Code	Destination	Source	Sample Instruction	Machine Code
40	R		60 R DA,	40 60
41	IR		61 IR DA,	41 61
42	r	r	R2 R3 rr OR,	42 23
43	r	r	R4 R5 rIr OR,	43 45
44	R	R	66 67 RR OR,	44 67 66
45	R	IR	68 69 RIR OR,	45 69 68
46	R	IM	6A 01 RIM OR,	46 6A 01
47	IR	IM	6B 02 IRIM OR,	47 6B 02
50	R	(stack)	60 R POP,	50 60
51	IR	(stack)	61 IR POP,	51 61
52	r	r	R2 R3 rr AND,	52 23
53	r	r	R4 R5 rIr AND,	53 45
54	R	R	66 67 RR AND,	54 67 66
55	R	IR	68 69 RIR AND,	55 69 68
56	R	IM	6A 01 RIM AND,	56 6A 01
57	IR	IM	6B 02 IRIM AND,	57 6B 02
60	R		60 R COM,	60 60
61	IR		61 IR COM,	61 61
62	r	r	R2 R3 rr TCM,	62 23
63	r	r	R4 R5 rIr TCM,	63 45
64	R	R	66 67 RR TCM,	64 67 66
65	R	IR	68 69 RIR TCM,	65 69 68
66	R	IM	6A 01 RIM TCM,	66 6A 01
67	IR	IM	6B 02 IRIM TCM,	67 6B 02
70	(stack)	R	60 R PUSH,	70 60
71	(stack)	IR	61 IR PUSH,	71 61
72	r	r	R2 R3 rr TM,	72 23
73	r	r	R4 R5 rIr TM,	73 45
74	R	R	66 67 RR TM,	74 67 66
75	R	IR	68 69 RIR TM,	75 69 68
76	R	IM	6A 01 RIM TM,	76 6A 01
77	IR	IM	6B 02 IRIM TM,	77 6B 02

Hex Code	Destination	Source	Sample Instruction	Machine Code
80	R		60 R DECW,	80 60
81	IR		61 IR DECW,	81 61
82	(Not Implemented)			
83	(Not Implemented)			
84	(Invalid)			
85	(Invalid)			
86	(Invalid)			
87	(Invalid)			
90	R		60 R RL,	90 60
91	IR		61 IR RL,	91 61
92	(Not Implemented)			
93	(Not Implemented)			
94	(Invalid)			
95	(Invalid)			
96	(Invalid)			
97	(Invalid)			
A0	R		60 R INCW,	A0 60
A1	IR		61 IR INCW,	A1 61
A2	r	r	R2 R3 rr CP,	A2 23
A3	r	Ir	R4 R5 rIr CP,	A3 45
A4	R	R	66 67 RR CP,	A4 67 66
A5	R	IR	68 69 RIR CP,	A5 69 68
A6	R	IM	6A 01 RIM CP,	A6 6A 01
A7	IR	IM	6B 02 IRIM CP,	A7 6B 02
B0	R		60 R CLR,	B0 60
B1	IR		61 IR CLR,	B1 61
B2	r	r	R2 R3 rr XOR,	B2 23
B3	r	Ir	R4 R5 rIr XOR,	B3 45
B4	R	R	66 67 RR XOR,	B4 67 66
B5	R	IR	68 69 RIR XOR,	B5 69 68
B6	R	IM	6A 01 RIM XOR,	B6 6A 01
B7	IR	IM	6B 02 IRIM XOR,	B7 6B 02

Hex Code	Destination	Source	Sample Instruction	Machine Code
C0	R		60 R RRC,	C0 60
C1	IR		61 IR RRC,	C1 61
C2	r	Irr	R2 W4 LDC,	C2 24
C3	Ir	Irr	R3 W6 LDCI,	C3 36
C4			(Invalid)	
C5			(Invalid)	
C6			(Invalid)	
C7	r	Rx	R8 60 R9 LDX,	C7 89 60
			(R9 is the index register in this example)	
D0	R		60 R SRA,	D0 60
D1	IR		61 IR SRA,	D1 61
D2	r	Irr	R2 W4 STC,	D2 42
D3	Ir	Irr	R3 W6 STCI,	D3 63
D4		IW	67 CALL0,	D4 67
D5			(Invalid)	
D6		A(IM)	' NAME CALL,	D6 xx xx
			(xx xx represents the address of NAME)	
D7	Rx	r	60 R8 R9 STX,	D7 89 60
E0	R		60 R RR,	E0 60
E1	IR		61 IR RR,	E1 61
E2			(Invalid)	
E3	r	Ir	R4 R5 rIr LD,	E3 45
E4	R	R	66 67 RR LD,	E4 67 66
E5	R	IR	68 69 IRR LD,	E5 69 68
E6	R	IM	6A 01 RIM LD,	E6 6A 01
E7	IR	IM	6B 02 IRIM LD,	E7 6B 02
F0	R		60 R SWAP,	F0 60
F1	IR		61 IR SWAP,	F1 61
F2			(Invalid)	
F3	Ir	r	R2 R3 Irr STR,	F3 23
F4			(Invalid)	
F5	IR	R	64 65 IRR STR,	F5 65 64
F6			(Invalid)	
F7			(Invalid)	

Hex Code	Destination	Source	Sample Instruction	Machine Code
r8	r	R	RC 6D	LDWR, C8 6D
r9	R	r	6E RF	STWR, F9 6E
rA	(Not implemented)			
rB	(See Logic Codes)			
rC	r	IM	RD FF	LDIM, 0C FF
rD	(See Logic Codes)			
rE	r		RD	rINC, 0E
0F	(Invalid)			
1F	(Invalid)			
2F	(Invalid)			
3F	(Invalid)			
4F	(Invalid)			
5F	(Invalid)			
6F	(Invalid)			
7F	(Invalid)			
8F	(FB)			DI, 8F
9F	(FB)			EI, 9F
AF		(rStack)		RET, AF
BF		(rStack)		IRET, BF
CF	(FC)			RCF, CF
DF	(FC)			SCF, DF
EF	(FC)			CCF, EF
FF				NOP, FF

Explanation of codes used in source and destination field.

Code      Meaning

r	Working Register
R	Register
IM	Immediate
Ir	Indirect reference through a working register
IR	Indirect reference through a register
IW	Indirect to external memory through a register pair
Irr	Indirect to external memory through a working register pair
xR	Refers to a register displaced from register R by the content of working register x
(Fn)	Is used for an instruction which is intended to modify bits of a system register. Many instructions result in the bits of the flag register being modified.
xx xx	Refers to an address which is compiled into the code.
(stack)	Refers to the stack.
(rStack)	Refers to the return address stack.

The content of the source field of an instructions is never changed. Those instructions which have a destination field have the capability to change the content of the destination field.

Some instructions can modify the content of the register used for indirect addressing (e.g. LDCI, STCI,). Any instruction for which a stack is mentioned modifies that stack.

### APPENDIX III. LISTING OF THE Z8/FA

```

: ERR ." ERROR # " H. QUIT ;
: ?ERR SWAP IF ERR ELSE DROP THEN ;
: CC, C, C, ;
\ Defining Words
: IN SWAP 10 U* DROP FO AND SWAP OF AND OR ;
: MO <BUILDS C, DOES> C@ C, ;
: M1 <BUILDS C, DOES> C@ OVER DUP 7 > OVER 2 < OR 5 ?ERR OR C, DUP 5 >      IF
DROP SWAP CC, ELSE 3 > IF CC, ELSE IN C, THEN THEN ;
: N1 <BUILDS C, DOES> C@ OVER 1 > 4 ?ERR OR CC, ;
: O1 <BUILDS C, DOES> C@ C, IN C, ;
: o1 <BUILDS C, DOES> C@ C, SWAP IN C, ;
: P1 <BUILDS C, DOES> C@ CC, ;
\ Address Mode Names
0 KON R 1 KON IR 2 KON rr 3 KON rIr 3 KON Irr
4 KON RR 5 KON IRR 5 KON RIR 6 KON RIM 7 KON IRIM
\ Register Names
E0 KON R0 E1 KON R1 E2 KON R2 E3 KON R3 E4 KON R4 E5 KON R5
E6 KON R6 E7 KON R7 E8 KON R8 E9 KON R9 EA KON RA EB KON RB
EC KON RC ED KON RD EE KON RE EF KON RF E0 KON W0 E2 KON W2
E4 KON W4 E6 KON W6 E8 KON W8 EA KON WA EC KON WC EE KON WE
\ OP Codes
8F MO DI, 9F MO EI, AF MO RET, BF MO IRET, CF MO RCF, DF MO SCF, EF MO
CCF, FF MO NOP, 30 P1 JP, 31 P1 SRP, 00 M1 ADD, 10 M1 ADC, 20 M1 SUB, 30
M1 SBC, 40 M1 OR, 50 M1 AND, 60 M1 TCM, 70 M1 TM, A0 M1 CP, B0 M1 XOR,
00 N1 DEC, 10 N1 RLC, 20 N1 INC, 40 N1 DA, 50 N1 POP, 60 N1 COM, 70 N1
PUSH, 80 N1 DECW, 90 N1 RL, A0 N1 INCW, B0 N1 CLR, C0 N1 RRC, D0 N1 SRA, E0
N1 RR, F0 N1 SWAP, C2 O1 LDC, D2 o1 STC, C3 O1 LDCI, D3 o1 STCI,
: LD, DUP 7 > OVER 3 < OR 7 ?ERR DUP E0 OR C, DUP 5 > IF DROP
SWAP CC, ELSE 3 > IF CC, ELSE IN C, THEN THEN ;
: LDWR, SWAP 08 IN CC, ;
: STWR, 09 IN CC, ;
: LDIM, SWAP 0C IN CC, ;
: rINC, 0E IN C, ;
: CALL, D6 C, , ;
: CALL@, D4 CC, ;
: LDX, C7 C, ROT SWAP IN CC, ;
: STX, D7 C, IN CC, ;
\ Condition Code Names
F KON carry 7 KON nc E KON z 6 KON nz 5 KON pl D KON mi
C KON ov 4 KON nov E KON eq 1 KON ge 9 KON lt 2 KON gt
A KON le 7 KON uge F KON ult 3 KON ugt B KON ule 6 KON ne
8 KON never 0 KON always
: CHO C, HERE 0 ;
: :, CREATE SMUDGE HERE 2 + , ;
: CH1 C, HERE 1+ - C, ;

```

```

/ Logic and Flow Control
: do,    HERE ;
: begin, HERE ;
: DO,   HERE ;
: BEGIN, HERE ;
: loop,  EB CH1      ;
: LOOP, ED C, , ;
: if,    OB IN CHO   C, ;
: IF,   OD IN CHO   , ;
: then,  HERE OVER 1+ - SWAP C! ;
: THEN, HERE SWAP ! ;
: else,  8B CHO      C, SWAP then, ;
: ELSE,  8D CHO      , SWAP THEN, ;
: until, OB IN CH1   ;
: UNTIL, OD IN C, , ;
: while, if, ;
: WHILE, IF, ;
: repeat, 8B C, SWAP HERE 1+ - C, then, ;
: REPEAT, 8D C, SWAP , HERE SWAP ! ;
: again,  8B CH1     ;
: AGAIN, 8D C, , ;
: EXIT, 3050 , ;

```

## APPENDIX IV. TIME-OF-DAY CLOCK PROGRAM LISTING

```

:, TIME-OF-DAY          \ If the timer has interrupted
68 R INC,              \ increment the fractions of a second
68 55 RIM CP,         \ and see if it has reached the limit.
eq IF,                \ If so,
    68 0 RIM LD,      \ clear it and
    69 R INC,         \ increment seconds. Did the number
    69 3C RIM CP,     \ of seconds reach 60?
eq IF,                \ If so,
    69 0 RIM LD,     \ clear seconds and
    6A R INC,        \ increment minutes. Did the number
    6A 3C RIM CP,    \ of minutes reach 60?
eq IF,                \ If so,
    6A 0 RIM LD,    \ clear minutes and
    6B R INC,        \ increment hours.
    THEN,
    THEN,
    THEN,
IRET,                  \ Return to the point of interruption.

: DECIMAL A 18 ! ; : HEX 10 18 ! ;
: #D DECIMAL 20 WORD NUMBER DROP HEX ; : SD. S->D D. ;
:, DI DI, EXIT, :, EI FA 10 RIM AND, EI, EXIT,

: T-INIT              \ Timer Initialization
    DI                \ Disable Interrupts
    FF F3 C!         \ Load maximum value in T1 Prescaler
    00 F2 C!         \ Clear T1 timer register
    F1 C@ 0C OR F1 C! \ Load T1 and Enable Count
    ' TIME-OF-DAY E ! \ Put address of routine in IRQ5 vector
    00 68 ! 00 6A ! \ Clear registers used for clock
    1E F9 C!         \ Set Interrupt Priorities
    FB C@ 20 OR FB C! \ Set enable bit for IRQ5
;                    \ But don't enable interrupts yet. . .

: SETTIME
    T-INIT           \ Initialize Timer
    #D 6B C!        \ Input Hours
    #D 6A C!        \ Input Minutes
    #D 69 C!        \ Input Seconds
    0 68 C!         \ Clear fractions of a second
    EI ;           \ Enable interrupts to start the clock

: TIME              \ Display current time
    DEC SPACE 6B C@ SD. ." : " 6A C@ SD. ." : " 69 C@ SD. CR HEX ;

```

## APPENDIX V. SAVE AND RESTORE WORKING REGISTER PROGRAM LISTING

```

:, RST          \ Restore Working Register Pointer
6E F0 RIM AND,  \ to Value Saved in Register 6E
6E 00 RIM CP, eq IF, 00 SRP, THEN,
6E 10 RIM CP, eq IF, 10 SRP, THEN,
6E 20 RIM CP, eq IF, 20 SRP, THEN,
6E 30 RIM CP, eq IF, 30 SRP, THEN,
6E 40 RIM CP, eq IF, 40 SRP, THEN,
6E 50 RIM CP, eq IF, 50 SRP, THEN,
6E 60 RIM CP, eq IF, 60 SRP, THEN,
6E 70 RIM CP, eq IF, 70 SRP, THEN,
6E F0 RIM CP, eq IF, F0 SRP, THEN, RET,

```

### Linkage Example

```

( . . . )      \ Original register pointer has unknown value.
FD R PUSH,     \ Save original register pointer value.
  xx SRP,      \ You may change the register pointer here,
  ( . . . )    \ but the stack must remain balanced.
6E R POP,      \ Put saved value in register 6E.
' RST CALL,    \ Call routine to restore register pointer.
( . . . )      \ Register pointer is restored to original value.

```

## INTERNAL DISTRIBUTION LIST

- |     |                  |        |                            |
|-----|------------------|--------|----------------------------|
| 1.  | W. Fulkerson     | 15.    | Central Research Library   |
| 2.  | R. T. Goeltz     | 16.    | Document Reference Section |
| 3.  | J. O. Kolb       | 17-19. | Laboratory Records         |
| 4.  | Russell Lee      | 20.    | Laboratory Record-RC       |
| 5.  | V. C. Mei        | 21.    | R. B. Honea                |
| 6.  | George T. Privon | 22.    | ORNL Patent Office         |
| 7.  | P. M. Spears     | 23.    | R. B. Shelton              |
| 8.  | D. P. Vogt       | 24.    | Reid Gryder                |
| 9.  | R. G. Edwards    |        |                            |
| 10. | Steve Wallace    |        |                            |
| 11. | T. J. Wilbanks   |        |                            |
| 12. | Teresa G. Yow    |        |                            |
| 13. | D. E. Reichle    |        |                            |
| 14. | Eleanor Rogers   |        |                            |

## EXTERNAL DISTRIBUTION LIST

25. Office of Assistant Manager for Energy Research and Development, Department of Energy, Oak Ridge Operations Office, Oak Ridge, TN 37831.
- 26-35. Office of Scientific and Technical Information, P. O. Box 62, Oak Ridge, TN 37831.
36. R. L. Perrine, Professor, Engineering & Applied Sciences, Civil Engineering Department, Engineering I. Room 2066, University of California, Los Angeles, Ca 90024.
37. D. E. Morrison, Professor of Sociology, Michigan State University, 201 Berkey Hall, East Lansing, MI 48824-1111.
38. Bruce Buchanan, Department of Computer Science, Alumni Hall, Rm. 318, University of Pittsburgh, Pittsburgh, PA 15260.
39. J. J. Cuttica, Vice President of Research & Development, Gas Research and Development, Mawr Avenue, Chicago, IL 60631.
40. Grimes Slaughter, Conray, Inc., 240 N. Purdue Ave., Apt. 211, Oak Ridge, TN 37830.
- 41-46. Charlie Adkins, Smart House Development Venture, Inc., 400 Prince Georges Center Blvd. Upper Marlboro, MD 20772-8731.
47. Martin Williams, Professor, Department of Economics, Northern Illinois University, Dekalb, IL 60115.