Rockwell

A65-050
AIM 65 FORTH ROMS

FORTH LANGUAGE

FORTH is a unique programming language well suited to a variety of applications. Because it was originally developed for real-time control applications, FORTH is ideal for machine and process control, energy managements, data acquisition, automatic testing, robotics and other applications where assembly language was previously the only possible language choice.

FORTH actually provides the best of two worlds. It has the looping and branching constructs of high-level languages (DO . . . LOOP, BEGIN . . . END, IF . . . THEN and IF . . . ELSE . . . THEN) and the code efficiency of machine and assembly languages. And programmers will be pleased to know that FORTH allows you to specify addresses, operands and data in hexadecimal, octal, binary or any other number base from two to 40—a distinct advantage over languages like BASIC, where all information must be in decimal.

In most time-critical applications, at least part of the program must be written in assembly language. FORTH has a built in 6502 macro assembler, and lets you drop into assembly language at almost any point in your program, without perarate assembly and load steps or awkward machine level linkage. FORTH programs typically run up to ten times faster than other interpretive languages, and can even approach the speed of machine language programs for some applications.

FEATURES

- AIM 65 Microcomputer host and target
- ROM resident for immediate operation
- Application oriented
- Extensible language
- Over 200 pre-defined functions
- Interactive compilation
- Reverse polish notation
- Compact memory usage
- Fast execution
- Easy debugging
- Stack implementation
- 16-bit words
- Built-in structured macro assembler
- Shortens software development time

PRODUCT OVERVIEW

AIM 65 FORTH, consisting of primitives, interpreter, macro assembler and input/output functions, is contained in two 4K-byte ROMs that plug into the AIM 65 Microcomputer Master Module sockets Z25 and Z26. FORTH functions are linked to AIM 65 Debug Monitor and Text Editor ROMs providing access to the AIM 65 peripherals (keyboard, single-line display and 20 column printer) as well as user-defined input/output functions. Both interactive and batch modes of operation are supported. Interactive operation interprets FORTH words upon entry for immediate execution and debugging or for compilation. In the batch mode, FORTH words can be entered into the Text Buffer then input to FORTH for interpretation. The batch mode allows an application program to be easily edited using Text Editor commands. Application programs written in FORTH can thus be developed, as well as executed for checkout or production operation, on the AIM 65 Microcomputer.

AIM 65 FORTH ROMs, when installed in an AIM 65 Microcomputer, can also be used to develop and checkout an application program written in FORTH that is to be installed in an RM 65 Single Board Computer (SBC) module for runtime operation. The developed program will run with RM 65 Run-Time FORTH ROMs installed in the RM 65 SBC module. In this configuration, all RM 65 input/output operations must be user-provided and can be tested using the AIM 65 Microcomputer as the host computer prior to being installed in the RM 65 environment.

ORDERING INFORMATION

Part No.	Description
A65-050	AIM 65 FORTH ROMs
A65-040	AIM 65 Math Package ROM
RM65-0152	RM 65 Run-Time FORTH ROM
Order No.	**Description**
265	AIM 65 FORTH User's Manual(1)
283	AIM 65 FORTH Reference Card(1, 2)
2118	AIM 65 Math Package User's Manual(2)
812	RM 65 Run-Time FORTH User's Manual(3)

NOTES:
1. Included with A65-050.
2. Included with A65-040.
3. Included with RM65-0152.

DEVELOPING FORTH PROGRAMS

FORTH is built on subroutine-like functions, called "words." These words are linked together to form a "dictionary," which is the central core of the language. Writing a program in FORTH consists of using several predefined words to define each new word. Once the new word has been added to the system dictionary, it becomes as much a part of the language as any other word that has been previously defined. In this way new features and extensions can be added by simply defining one or more new words. Adding new features to conventional languages like BASIC or Pascal requires the language system to be completely reassembled or recompiled.

FORTH is a stack-oriented language, and is programmed in Reverse Polish Notation (RPM), the notation that is used in Hewlett-Packard scientific calculators. Using a data stack is an extremely efficient way of passing variables back and forth between operations. A data stack eliminates the need to tie up memory locations with variable tables, and allows you to use only as much memory as you need.

FORTH programs are developed using "top-down/bottom-up" techniques. That is, the programmer begins by defining the program in very general terms, then systematically breaks these definitions down into more and more detailed sub-modules. When the lowest levels of sub-modules have been defined, he starts coding, in FORTH, at those levels, working back up toward the top of the program, in pyramid fashion. Each sub-module is a stand-alone component of the program, and can be completely debugged without having the complete program in the system. This type of software development is difficult, if not impossible, to do with most other high-level languages.

FLOATING POINT FUNCTIONS

AIM 65 FORTH contains both a single- (16-bit) and double- (32-bit) precision integer arithmetic capability. In AIM 65 applications where floating point arithmetic is desired, the AIM 65 Math Package may be used in conjunction with the FORTH ROMs. These floating point functions may be called using FORTH words included in the math package ROM. When this ROM is installed in socket Z24 on the AIM 65 Microcomputer, the floating point math words can be automatically linked to the FORTH dictionary during FORTH initialization. The AIM 65 Math Package ROM can also be installed in either an RM 65 SBC or PROM/ROM module.

MEMORY MAP

Address (Hex)	Contents
$D000-$DFFF	Math Package Program
$B000-$CFFF	FORTH Program
$280-$2FF	Terminal Input Buffer
$25C-$27F	Math Package Variables
$200-$257	FORTH User Variables
$AB-$C4	Math Package Variables
$10-$AA	FORTH Variables

FORTH WORDS

STACK MANIPULATION

DUP	Duplicate top of stack.
2DUP	Duplicate top two stack items.
DROP	Delete top of stack.
2DROP	Delete top two stack items.
SWAP	Exchange top two stack items.
OVER	Copy second item to top.
ROT	Rotate third item on top.
- DUP	Duplicate only if non-zero
>R	Move top item to return stack.
R>	Retrieve item from return stack.
R	Copy top of return stack onto stack.
PICK	Copy the nth item to top.
SP@	Return address of stack position
RP@	Return address of return stack pointer.
BOUNDS	Convert "address count" to "end-address start-address."
.S	Print contents of stack.

DEFINING WORDS

:<name>	Begin colon definition of <name>.
;	End colon definition.
VARIABLE <namd>	Create a variable <name> with initial value n; returns address when executed.
CONSTANT <name>	Create a constant <name> with value n; returns value when executed.
CODE <name>	Begin definition of assembly-language primitive operation <name>.
;CODE	Used to create a new defining word, with execution-time "code routine" for this data type in assembly.
<BUILDS . . . DOES>	Used to create a new defining word, with execution-time routine for this data type in higher-level FORTH.
USER	Create a user variable.

MEMORY

@	Fetch value addressed by top of stack.
!	Store n1 at address n2.
C@	Fetch one byte only.
C!	Store one byte only.
?	Print contents of address.
+!	Add second number on stack to contents of address on top.
CMOVE	Move n3 bytes starting at address n1 to area starting at address n2.
FILL	Put byte n3 into n2 bytes starting at address n1.
ERASE	Fill n2 bytes in memory with zeroes, beginning at address n1.
BLANKS	Fill n2 bytes in memory with blanks, beginning at address n1.
TOGGLE	Mask memory with bit pattern.

NUMERIC REPRESENTATION

DECIMAL	Set decimal base.
HEX	Set hexadecimal base.
BASE	Set number base.
DIGIT	Convert ASCII to binary.
0	The number zero.
1	The number one.
2	The number two.
3	The number three.

FORTH WORDS (CONT'D)

ARITHMETIC AND LOGICAL

+	Add.
D+	Add double-precision numbers.
–	Subtract (n1 – n2)
*	Multiply.
/	Divide (n1/n2).
MOD	Modulo (i.e., remainder from division).
/MOD	Divide, giving remainder and quotient.
*/MOD	Multiply, then divide (n1*n2/n3), with double intermediate.
*/	Like */MOD, but give quotient only.
U*	Unsigned multiply leaving double product.
U/	Unsigned divide.
M*	Signed multiplication leaving double product.
M/	Signed remainder and quotient from double dividend.
M/MOD	Unsigned divide leaving double quotient and remainder from double dividend and single divisor.
MAX	Maximum.
MIN	Minimum.
+ –	Set sign.
D+ –	Set sign of double-precision number.
ABS	Absolute value.
DABS	Absolute value of double-precision number.
NEGATE	Change sign.
DNEGATE	Change sign of double-precision number.
S– >D	Sign extend to double-precision number.
1 +	Increment value on top of stack by 1.
2 +	Increment value on top of stack by 2.
1 –	Decrement value on top of stack by 1.
2 –	Decrement value on top of stack by 2.
AND	Logical AND (bitwise).
OR	Logical OR (bitwise)
XOR	Logical exclusive OR (bitwise).

COMPARISON OPERATORS

<	True if n1 less than n2.
>	True if n1 greater than n2.
=	True if top two numbers are equal.
0<	True if top number negative.
0=	True if top number zero.
U<	True if u1 less than u2.
NOT	Same as 0=.

MISCELLANEOUS AND SYSTEM

(<comment>)	Begin comment (terminate by right parentheses on same line).
CFA	Alter PFA to CFA.
NFA	Alter PFA to NFA.
PFA	Alter NFA to PFA.
LFA	Alter PFA to LFA.
LIMIT	Top of memory.
QUIT	Clear return stack and return to terminal.

CONTROL STRUCTURES

DO . . . LOOP	Set up loop, given index range.
DO . . . +LOOP	Like DO . . . LOOP, but adds stack value to index.
I	Place current index value on stack.
LEAVE	Terminate loop at next LOOP or +LOOP.
BEGIN . . . UNTIL	Loop back to BEGIN until true at UNTIL.
BEGIN . . . WHILE . . . REPEAT	Loop while true at WHILE, REPEAT loops unconditionally to BEGIN.
BEGIN . . . AGAIN	Unconditional loop.
IF . . . THEN	If top of stack true, execute following clause THEN continue; otherwise continue at THEN.
IF . . . ELSE . . . THEN	If top of stack true, execute ELSE clause THEN continue; otherwise execute following clause, THEN continue.
END	Alias for UNTIL.
ENDIF	Alias for THEN.

COMPILER-TEXT INTERPRETER

[COMPILE]	Force compilation of IMMEDIATE word.
COMPILE	Compile following <name> into dictionary.
LITERAL	Compile a number into a literal.
DLITERAL	Compile a double-precision number into a literal.
EXECUTE	Execute the definition on top of stack.
[	Suspend compilation, enter execution.
]	Resume compilation.

DICTIONARY CONTROL

CREATE	Create a dictionary header.
FORGET	FORGET all definitions from <name> on.
HERE	Returns address of next unused byte in the dictionary.
ALLOT	Leave a gap of n bytes in the dictionary.
TASK	A dictionary marker.
'	Find the address of <name> in the dictionary.
- FIND	Search dictionary for <name>.
DP	User variable containing the dictionary pointer.
C,	Store byte into dictionary.
,	Compile a number into the dictionary.
PAD	Pointer to temporary buffer.
IMMEDIATE	Force execution when compiling.
INTERPRET	The Text Interpreter executes or compiles.
LATEST	Leave name field address (NFA) of top word in CURRENT.
LIT	Place 16-bit literal on the stack.
CLIT	Place byte literal on the stack.
LITERAL	Compile a 16-bit literal.
SMUDGE	Toggle name SMUDGE bit.
STATE	User variable containing compilation state.

FORTH WORDS (CONT'D)

USER VARIABLES

UABORT	User variable for ABORT.
UB/BUF	User variable for B/BUF.
UB/SCR	User variable for B/SCR.
UC/L	User variable for C/L.
UEMIT	User variable for EMIT.
UFIRST	User variable for FIRST.
UKEY	User variable for KEY.
ULIMIT	User variable for LIMIT.

MONITOR & CASSETTE I/O

COLD	AIM 65 FORTH cold start.
MON	Exit to AIM 65 Monitor.
?TTY	Switch; true = TTY; false = KB
CHAIN	Chain tape file.
CLOSE	Close tape file.
?IN	Set to active input device (AID).
?OUT	Set to active output device (AOD).
GET	Input a character from the AID.
PUT	Output a character to the AOD.
READ	Input n2 characters from AID to address n1.
WRITE	Output n2 characters to AOD at address n1.
SOURCE	Compile from the AID.
FINIS	Terminate complete from SOURCE.

INPUT-OUTPUT

- CR	Output CR to printer only.
CR	Carriage return.
SPACE	Type one space.
SPACES	Type n spaces.
CLRLINE	Output a CTRL B.
"	Print text string (terminated by ").
DUMP	Dump n2 words starting at address.
TYPE	Type string of n1 characters starting at address n2.
?TERMINAL	True if terminal break request present.
KEY	Read key, put ASCII value on stack.
EMIT	Output ASCII value from stack.
EXPECT	Read n1 characters from input to address n2.
WORD	Read one word from input stream, until delimiter.
IN	User variable contained within TIB.
BAUD	Set BAUD rate.
BL	Output a SPACE character.
C/L	Number of characters/line.
TIB	Pointer to terminal input buffer start address.
QUERY	Input text from terminal.
ID.	Print <name> from name # field address (nfa).
HANG	Wait for keystroke.

OUTPUT FORMATTING

NUMBER	Convert string at address to double-precision number.
<#	Start output string.
#	Convert next digit of double-precision number and add character to output string.
#S	Convert all significant digits of double-precision number to output string.
SIGN	Insert sign of n into output string.
#>	Terminate output string (ready for TYPE).
HOLD	Insert ASCII character into output string.
HDL	Hold pointer, user variable.
- TRAILING	Suppress trailing blanks.
.LINE	Display line of text from mass storage.
COUNT	Change length of byte string to type form.
	Print number on top of stack.
.R	Print number n1 right justified n2 places.
D.	Print double-precision number n2 n2.
D.R	Print double-precision number n2 n1 right justified n3 places.
DPL	Number of digits to the right of decimal point.

VOCABULARIES

CONTEXT	Returns address of pointer to CONTEXT vocabulary.
CURRENT	Returns address of pointer to CURRENT vocabulary.
FORTH	Main FORTH vocabulary.
ASSEMBLER	Assembler vocabulary.
DEFINITIONS	Set CURRENT vocabulary to CONTEXT.
VOCABULARY <name>	Create new vocabulary.
VLIST	Print names of all words in CONTEXT vocabulary.
VOC-LINK	Most recently defined vocabulary.

VIRTUAL STORAGE

LOAD	Load mass storage screen (compile or execute).
BLOCK	Read mass storage block to memory address.
B/BUF	System constant giving mass storage block size in bytes.
B/SCR	Number of blocks/editing screen.
BLK	System variable containing current block number.
SCR	System variable containing current screen number.
UPDATE	Mark last buffer accessed as updated.
FLUSH	Write all updated buffers to mass storage.
EMPTY-BUFFERS	Erase all buffers.
+BUF	Increment buffer address.
BUFFER	Fetch next memory buffer.
RW	User read write linkage.
USE	Variable containing address of next buffer.
PREV	Variable containing address of latest buffer.
FIRST	Leaves address of first block buffer.
OFFSET	User variable block offset to mass storage.
- ->	Interpret next screen.
;S	Stop interpretation.

FORTH WORDS (CONT'D)

PRIMITIVES

0BRANCH	Run-time conditional branch.
BRANCH	Run-time unconditional branch.
ENCLOSE	Text scanning primitive used by WORD.
R0	Location of Return Stack.
S0	Location of Parameter Stack.
RP!	Initialize Return Stack.
SP!	Initialize Parameter Stack.
NEXT	The FORTH virtual machine.

SECURITY

!CSP	Store stack position in check stack pointer.
?COMP	Error if not compiling.
?CSP	Check stack position.
?ERROR	Output error message.
?EXEC	Not executing error.
?PAIRS	Conditional not paired error.
?STACK	Stack out of bounds error.
ABORT	Error; operation terminates.
ERROR	Execute error notification and restart system.
MESSAGE	Displays message.
WARNING	Pointer to message routine.
FENCE	Prevents forgetting below this point.
WIDTH	Controls significant characters of <name>.

MATH PACKAGE FORTH WORDS (A65-040)*

FLOATING POINT ARITHMETIC

F+	Adds two floating point numbers.
F−	Subtracts one floating point number from another floating point number.
F*	Multiplies two floating point numbers.
F/	Divides one floating point number by another floating point number.

UTILITY, SIGN AND COMPARISONS

FABS	Takes the absolute value of a floating point number.
INT	Truncates a floating point number to an integer.
SGN	Converts the sign of a floating point number to a floating point number.
FSIGN	Gets a value corresponding to the sign of a floating point number.
FCOMP	Compares the value of a compacted number in memory to a floating point number.

POLYNOMIAL

POLY	Evaluates a polynomial with consecutive exponents.
POLYODD	Evaluates a polynomial with odd exponents.

EXPONENTIAL AND LOGARITHMIC

SQR	Takes the square root of a floating point number.
>	Raises one floating point number to the power of another floating point number.
EXP	Raises the transcendental number e to the power of a floating point number.
LOG	Computes the logarithm to the base 10 (i.e., common log) of a floating point number.
LN	Computes the logarithm to the base e (i.e., natural log) of a floating point number.

*Requires AIM 65 FORTH or RM 65 FORTH be resident.

USER VARIABLE

MIN-WIDTH	Specifies the minimum field width to be output.
DEC-LENGTH	Specifies the number of places to the right of the decimal point to be output.

ASCII/FLOATING POINT CONVERSIONS

FIN	Converts a number in memory from ASCII to floating point format.
FOUT	Converts a number from floating point to ASCII.

FORMAT CONVERSION AND DATA MOVING

M>F	Unpacks the compacted number in memory to floating point.
F>M	Packs the floating point number to compacted format and stores the result in memory.
M>A	Unpacks the floating point number in memory.
S>A	Converts an integer to floating point format.
S>F	Converts an integer to floating point format.
F>S	Converts a number from floating point to an integer.

TRIGONOMETRIC AND UNITS CONVERSION

SIN	Calculates the sine of a floating point number (in radians).
COS	Calculates the cosine of a floating point number (in radians).
TAN	Calculates the tangent of a floating point number (in radians).
ARCTAN	Calculates the arc tangent of a floating point number.
DEGREES	Converts a floating point number from radians to degrees.
RADIANS	Converts a floating point number from degrees to radians.

ELECTRONIC DEVICES DIVISION REGIONAL ROCKWELL SALES OFFICES

HOME OFFICE
Electronic Devices Division
Rockwell International
4311 Jamboree Road
Newport Beach, California 92660

Mailing Address:
P.O. Box C
Newport Beach, California 92660
Mail Code 501-300
Tel: 714-833-4700
TWX: 910 591-1698

UNITED STATES
Electronic Devices Division
Rockwell International
1842 Reynolds
Irvine, California 92714
(714) 833-4655
ESL 62108710
TWX: 910 595-2518

Electronic Devices Division
Rockwell International
921 Bowser Road
Richardson, Texas 75080
(214) 996-6500
Telex: 73-307

Electronic Devices Division
Rockwell International
10700 West Higgins Rd., Suite 102
Rosemont, Illinois 60018
(312) 297-8862
TWX: 910 233-0179 (RI MED ROSM)

Electronic Devices Division
Rockwell International
5001B Greentree
Executive Campus, Rt. 73
Marlton, New Jersey 08053
(609) 596-0090
TWX: 710 940-1377

FAR EAST
Electronic Devices Division
Rockwell International Overseas Corp.
Itohpia Hirakawa-cho Bldg.
7-6, 2-chome, Hirakawa-cho
Chiyoda-ku, Tokyo 102, Japan
(03) 265-8806
Telex: J22198

EUROPE
Electronic Devices Division
Rockwell International GmbH
Fraunhoferstrasse 11
D-8033 Munchen-Martinsried
West Germany
(089) 857-6016
Telex: 0521/2650 rimd d

Electronic Devices Division
Rockwell International
Heathrow House, Bath Rd.
Cranford, Hounslow,
Middlesex, England
(01) 759-9911
Telex: 851-25463

Electronic Devices
Rockwell Collins
Via Boccaccio, 23
20123 Milano, Italy
498.74.79
Telex: 202/82

YOUR LOCAL REPRESENTATIVE

10/82