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 management, 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 separate 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 Z28 and Z29. 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

<table>
<thead>
<tr>
<th>Part No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A85-050</td>
<td>AIM 65 FORTH ROMs</td>
</tr>
<tr>
<td>A85-040</td>
<td>AIM 65 Math Package ROM</td>
</tr>
<tr>
<td>RM65-0152</td>
<td>RM 65 Run-Time FORTH ROM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Order No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>265</td>
<td>AIM 65 FORTH User's Manual</td>
</tr>
<tr>
<td>283</td>
<td>AIM 65 FORTH Reference Card</td>
</tr>
</tbody>
</table>

NOTES:
1. Included with A85-060.
2. Included with A85-040.
3. Included with RM65-0152.

Document No. 29000D87

Data Sheet Order No. D87
Rev. 1, March 1983
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 (RPN), 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

<table>
<thead>
<tr>
<th>Address (Hex)</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0000-$DFFF</td>
<td>Math Package Program</td>
</tr>
<tr>
<td>$8000-$CFFF</td>
<td>FORTH Program</td>
</tr>
<tr>
<td>$2B0-$2FF</td>
<td>Terminal Input Buffer</td>
</tr>
<tr>
<td>$250-$27F</td>
<td>Math Package Variables</td>
</tr>
<tr>
<td>$200-$237</td>
<td>FORTH User Variables</td>
</tr>
<tr>
<td>$AB-$C4</td>
<td>Math Package Variables</td>
</tr>
<tr>
<td>$10-$5AA</td>
<td>FORTH Variables</td>
</tr>
</tbody>
</table>

MEMORY

@  Fetch value addressed by top of stack.
1  Store n1 at address n2.
C  Fetch one byte only.
?  Store one byte only.
?  Print contents of address.
+1  Add second number on stack to contents of address on top.
MOVE 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         Double, giving remainder and quotient.
/IMOD        Multiply, then divide (n1/n2n3), with double intermediate.
J            Like /IMOD, 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/IMOD       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.
e            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.
LEAVE        Place current index value on stack.
BEGIN ... UNTIL Loop back to BEGIN until true at UNTIL.
BEGIN ... WHILE Loop while true at WHILE, REPEAT loops unconditionally to BEGIN.
BEGIN ... REPEAT Unconditional loop.
BEGIN ... AGAIN 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.
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.
ALLOC          Leave a gap of n bytes in the dictionary.
TASK           A dictionary marker.
- FIND          Search dictionary for <name>.
DP              User variable containing the dictionary pointer.
C              Store byte into dictionary.
PAD            Compile a number into the dictionary.
IMMEDIATE      Pointer to temporary buffer.
INTERPRET      Force execution when compiling.
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

USBORT User variable for ABORT.
US/BUF User variable for B/BUF.
US/SCR User variable for B/SCR.
UC/L User variable for C/L.
UGE/M User variable for EMIT.
UFRST User variable for FIRST.
UK/KEY User variable for KEY.
ULK/M 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
WRITE Output n2 characters to AOD at
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.
CL/RIN Output a CTRL B.
DUMP Dump n2 words starting at address.
TYPE Type string of n1 characters starting at
?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
WORD Read one word from input stream, until
IN User variable contained within TIB.
BAUD Set BAUD rate.
BL Set a SPACE character.
C/L Number of characters/line.
TIB Pointer to terminal input buffer start
QUERY Input text from terminal.
ID. Print <name> from name # field
HANG Wait for keystroke.

OUTPUT FORMATTING (CONT’D)

# Convert next digit of double-precision
#S number and add character to output
SIGN string.
#> Convert all significant digits of double-
#E precision number to output string.
#E> Insert sign of n into output string.
#E> Insert output string (ready for
#E 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.
.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.
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 Increment buffer address.
BUFFERS 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 PRIMITIVES (CONT'D)

<table>
<thead>
<tr>
<th>PRIMITIVE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0BRANCH</td>
<td>Run-time conditional branch.</td>
</tr>
<tr>
<td>BRANCH</td>
<td>Run-time unconditional branch.</td>
</tr>
<tr>
<td>ENCLOSE</td>
<td>Text scanning primitive used by WORD.</td>
</tr>
<tr>
<td>RD</td>
<td>Location of Return Stack.</td>
</tr>
<tr>
<td>SO</td>
<td>Location of Parameter Stack.</td>
</tr>
<tr>
<td>RPI</td>
<td>Initialize Return Stack.</td>
</tr>
<tr>
<td>SPI</td>
<td>Initialize Parameter Stack.</td>
</tr>
<tr>
<td>NEXT</td>
<td>The FORTH virtual machine.</td>
</tr>
</tbody>
</table>

### SECURITY PRIMITIVES

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

### MATH PACKAGE FORTH WORDS (A65-040)*

#### FLOATING POINT ARITHMETIC

<table>
<thead>
<tr>
<th>PRIMITIVE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>F+</td>
<td>Adds two floating point numbers.</td>
</tr>
<tr>
<td>F–</td>
<td>Subtracts one floating point number from another floating point number.</td>
</tr>
<tr>
<td>F*</td>
<td>Multiplies two floating point numbers.</td>
</tr>
<tr>
<td>F/</td>
<td>Divides one floating point number by another floating point number.</td>
</tr>
</tbody>
</table>

#### UTILITY, SIGN AND COMPARISONS

<table>
<thead>
<tr>
<th>PRIMITIVE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>FABS</td>
<td>Takes the absolute value of a floating point number.</td>
</tr>
<tr>
<td>INT</td>
<td>Truncates a floating point number to an integer.</td>
</tr>
<tr>
<td>SGN</td>
<td>Converts the sign of a floating point number to a floating point number.</td>
</tr>
<tr>
<td>FSIGN</td>
<td>Gets a value corresponding to the sign of a floating point number.</td>
</tr>
<tr>
<td>FCOMP</td>
<td>Compares the value of a compacted number in memory to a floating point number.</td>
</tr>
</tbody>
</table>

#### POLYNOMIAL

<table>
<thead>
<tr>
<th>PRIMITIVE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>POLY</td>
<td>Evaluates a polynomial with consecutive exponents.</td>
</tr>
<tr>
<td>POLYODD</td>
<td>Evaluates a polynomial with odd exponents.</td>
</tr>
</tbody>
</table>

### EXPONENTIAL AND LOGARITHMIC

<table>
<thead>
<tr>
<th>PRIMITIVE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQRT</td>
<td>Takes the square root of a floating point number.</td>
</tr>
<tr>
<td>&gt;</td>
<td>Raises one floating point number to the power of another floating point number.</td>
</tr>
<tr>
<td>EXP</td>
<td>Raises the transcendental number e to the power of a floating point number.</td>
</tr>
<tr>
<td>LOG</td>
<td>Computes the logarithm to the base 10 (i.e., common log) of a floating point number.</td>
</tr>
<tr>
<td>LN</td>
<td>Computes the logarithm to the base e (i.e., natural log) of a floating point number.</td>
</tr>
</tbody>
</table>

### ASCI/FLOATING POINT CONVERSIONS

<table>
<thead>
<tr>
<th>PRIMITIVE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIN</td>
<td>Converts a number in memory from ASCII to floating point format.</td>
</tr>
<tr>
<td>FOUT</td>
<td>Converts a number from floating point to ASCII.</td>
</tr>
</tbody>
</table>

### FORMAT CONVERSION AND DATA MOVING

<table>
<thead>
<tr>
<th>PRIMITIVE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>M&gt;F</td>
<td>Unpacks the packed number in memory to floating point.</td>
</tr>
<tr>
<td>F&gt;M</td>
<td>Packs the floating point number to packed format and stores the result in memory.</td>
</tr>
<tr>
<td>M&gt;A</td>
<td>Unpacks the floating point number in memory.</td>
</tr>
<tr>
<td>S&gt;A</td>
<td>Converts an integer to floating point format.</td>
</tr>
<tr>
<td>S&gt;F</td>
<td>Converts an integer to floating point format.</td>
</tr>
<tr>
<td>F&gt;S</td>
<td>Converts a number from floating point to an integer.</td>
</tr>
</tbody>
</table>

### TRIGONOMETRIC AND UNITS CONVERSION

<table>
<thead>
<tr>
<th>PRIMITIVE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIN</td>
<td>Calculates the sine of a floating point number (in radians).</td>
</tr>
<tr>
<td>COS</td>
<td>Calculates the cosine of a floating point number (in radians).</td>
</tr>
<tr>
<td>TAN</td>
<td>Calculates the tangent of a floating point number (in radians).</td>
</tr>
<tr>
<td>ARCTAN</td>
<td>Calculates the arc tangent of a floating point number.</td>
</tr>
<tr>
<td>DEGREES</td>
<td>Converts a floating point number from radians to degrees.</td>
</tr>
<tr>
<td>RADIANS</td>
<td>Converts a floating point number from degrees to radians.</td>
</tr>
</tbody>
</table>

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