68-KNEWS

Volume 2, Number 1 May 1987

Hawthorne Technology, 8836 SE Stark, Portland, Oregon 97216
(503) 254-2005

# # # # #

GIANT UPDATE
on the TinyGiant 68000 Single Board Computer

We have had some people drop by to see the TinyGiant board. They are always surprised at it's small size. It does so many things but takes so little space. The board doesn't look crowded because each chip does so much that the high number of features doesn't translate into a high component count.

K-OS DOSEN'T MEAN CHAOS to us who are using it.
(Even if we do pronounce it that way.)

Many of the K-OS ONE operating system packages are going out to people who have either orphan or one of a kind 68000 systems. Others are going to people who just didn't like the operating system on the machine they have. Some people just want to see how an operating system goes together. The source code we provide makes a great study tool.

A Systems Software special interest group is continuing to meet at Hawthorne Technology on the third Tuesday of each month. The May meeting is going to be on Editors, how they work and how to write them.

If you are looking for books on the 68000, Forth, or other computer subjects, Magrathea has over 3000 titles in stock and will special order others. They also ship books on request, so if you can't stop by, call. (See advertisement below.)

- - - - - - - - - -

Bulletin Board Systems with 68000 information:

The Computer Journal magazine: 24 hrs, 1200 baud,
at: (406) 752-1038

Micro Cornucopia magazine: 24 hrs, 300-1200-2400 baud,
at: (503) 382-5060 (Conference #3 is for 68000)

Motorola BBS Freeware: (512) 440-3733
Voice: (512) 440-2714

- - - - - - - - - -

WHAT IS RPN, AND WHY WOULD YOU WANT TO USE IT?

There are three different ways to mix operators and the things they operate on: prefix, infix, and postfix. These simply mean that the operator comes before the operands, between the operands, or after the operands. Most of the common languages like BASIC, C or PASCAL are infix languages. LISP is the only common prefix language. Postfix languages, refered to as RPN (Reverse Polish Notation), are represented by FORTH, PostScript, and HTPL. The Teco editor used RPN. Adding machines all use RPN and most printing desk calculators use RPN. Each notation has its adherents. So why use RPN?

The compiler for an RPN language is smaller and simpler than the compiler for an infix algebraic language. A large portion of most compilers is a syntax analysis routine that converts the source language to an internal RPN format. If the source is RPN this step is eliminated. When a subscripted variable is referenced a lot of code needs to be generated to calculate the address to use. In RPN these calculations are explicit rather than hidden. For expressions, all an RPN compiler needs to do is push any operand on the evaluation stack and call or generate code for any operator.

In an RPN language, user created operators look the same as the built in operators. When a subroutine package is used to extend an infix language the subroutine calls are very different from the built in operators. If the extensions look the same as the built in operators they are easier to use and the whole program has a more natural look about it. It is easy to create a special set of words for graphics, statistics, mathamatics or data base programs. By the time a conventional language has been extended very far it starts to look more like LISP than whatever it started out as. An RPN language in contrast looks the same no matter how far it is extended.

An RPN language is much simpler to learn than an algebraic language. There are no rules of associativity or precedence. The operations are done in the order specified. In the C language there are 14 levels of precedence. Some associate left to right and some the other way. With RPN languages things are much simpler. If it is data it goes on the stack. If it is an action word, the action happens.

In an RPN language the programmer has more control over what kind of code is produced. The sequence of operations is given by the source code. You don't have to worry about the compiler rearranging the order to get better code. Even when using an optimizing compiler you are assured that the operations will be executed in the sequence given.

RPN languages are more flexible with the way arguments are passed to subroutines. You can pass parameters by value and parameters by reference in a single call

The items on the stack become abstract items. They can be used as values or addresses. They can be used as byte pointers, word pointers, long pointers, pointers to structures, or pointers to strings. Different numbers of parameters can be used by the called routine depending on what is found on the stack. A procedure can return a varying number of results depending on what happened. Conventional languages don't offer this kind of flexibility. The 68000 is a very good processor to use with an RPN language. All eight of the address registers can be used as stack pointers. In the other contending micros you have only a single stack pointer and that is used for return addresses. The 68000 also has a very effective set of opcodes that make for small efficient programs.

HTPL has very low overhead on procedure calls. In HTPL there is only a BSR or JSR to get to the procedure and an RTS to get back from the procedure. Any arguments used by the procedure are found on the evaluation stack. This means that there is no need for an explicit transfer of arguments.

Many new languages like Post Script are using RPN because as a subject gets more abstract the use of a stack to hold operands becomes more convenient. The algebraic languages were derived from math equations. When computing is less numeric in nature, it is useful to have a stack for a short term memory to hold what is being worked on.

There are not many books or articles on theory for RPN languages. In many cases this is because writers write about things that are easy to write about. If you look at any book on compilers you find good coverage of syntax and very little coverage of code generating. If you write a compiler you spend lots of time on the code generating and relatively little on the syntax.

Because FORTH is the best known of the current RPN languages many of it's quirks are assumed to be in all RPN languages. While some of these disadvantages may be true with FORTH, they are not neccessarily true about all RPN languages.

RPN and threaded code are not the same thing. RPN is a way for a programmer to describe the problem to the computer. Threaded code is a technique for generating object code. Threaded code has been popular for FORTH on microprocessors because it allows you to create a very fast interpreted instruction set. For 32 bit machines like the 68000 there is no real need to use threaded code.

Incremental compiling is also a technique that is often associated with RPN languages. This was a technique used to create an interactive environment without the slowness of a conventional interpreter. Many RPN languages are now compiled.

As you can see, RPN languages do not need to be feared. The weak points of popular RPN languages have given this method a bad name. One it does not rightly deserve. It may seem like an unnatural method at first. This is due to early mathamatics training. Anyone who has learned to use a 10 key adding machine has learned to use RPN with postfix operators. Most adding machine operators wouldn't recognize the terms, but after their first couple of weeks training, they don't even think about the order they enter the information into the machine. Ask someone you know who uses a 10 key by touch, what order they put the information into the machine. If they don't have a machine they can try it on to find out, they will have to think it through keystroke by keystroke. The actions have become automatic. It isn't so unnatural after all.

STARTING WITH HTPL

Welcome to HTPL programming. If you are familiar with Pascal, Modula or Forth then HTPL will have many parts that you already know. From FORTH we borrowed the use of RPN notation for expressions. From Pascal and Modula we borrowed a structure. HTPL is good for writing small, fast programs. It can also be extended to fit any special needs in other programming areas.

To start learning any new language it helps to see a complete example in that language. The example can then be related to the same program in a language you are more familiar with. This example is a simple but complete HTPL program that displays "Hello World!" on the console.

```
(sample program)
root
program
  "Hello World!" sprint
  13 putc  10 putc
  end
end
```

The first line is a comment. When anything is placed in parenthesis in an HTPL program it is treated as a comment and ignored. The word 'root' indicates that this is not an overlay and tells the compiler to include the runtime library with the generated object code. The word program tells where the program will start executing when it is run. The main part of the program continues until the first 'end'. The second 'end' indicates the end of the entire file being compiled. The words in the double quote marks are a string constant. When a string constant is encountered the contents of the string are saved in a data area and the address of the string is placed on the evaluation stack. The word 'sprint' is a call to a run time library routine to print the null terminated string. The 13 is the numeric value of a carriage return character. It is pushed on the stack. The word 'putc' is another library routine that prints the low 8 bits of the top of the stack as a single character. The '10 putc' sends out a linefeed character.

This is a complete HTPL program. When it is run it will display "Hello World!" on the terminal. Most of the tokens are refered to as words in HTPL just like in FORTH.

HTPL COMPILE AND RUN

To compile and run an HTPL program you first write the program using any editor. The compiler assumes that all characters have the high bit a zero. The output of the compiler is an executable binary file.

To compile a program type HTPL at the command line prompt. After the compiler is loaded it will prompt for the name of the first input file. Next it will prompt for the name of the output file. Any extension can be given for the output file but the command processor will only try to load and execute files that have the extension '.BIN'. Next you will be prompted for options. If you enter an 'N', there will be no listing of the source code as the program is compiled. If you put an 'S', there will be no symbol table listing after the program is compiled. The options can be given in any order. The complier reads the source program and any files involved twice. The run time library hex file "HTPLRTL.HEX" must be on the default drive for the compiler to find it. An overlay doesn't include the runtime library so it is not needed. You can include as many source files as you want at compile time so each source file can be kept small to be easier to edit.

STACK NOTATION

The commands in the manual have a comment describing the stack before and after the call to the routine. This is necessary because in a stack oriented programming environment the programmer has to keep track of the stack. Errors in the size of the stack is perhaps the most common kind of error made.

The letters or words before the '--' are the contents of the stack before the call. The top of the stack is on the far right hand side. The words or letters after the '--' are the contents after returning from the routine or after the word is executed. If a word appears before and not after it has been used up. The number of items before and after the call indicate how the stack will grow or shrink when the routine is run. If a routine calls itself then this can be used to estimate how many levels of stack will be required to run the program.

EXAMPLE: a b -- c

Shows the stack change:

stack before	stack after	
b		top
a	c	
---	---	bottom

HT-FORTH on the K-OS ONE Operating System.

We have had a lot of demand for a standard FORTH to run with K-OS ONE and for use on the TinyGiant. So we decided to write a standard FORTH. If you already know FORTH, this will make the system more usable. It also means that you can transport an existing FORTH application to your 68000 system with little effort. Another advantage is that there are many books on FORTH and FORTH user groups. A lot of the books have sample programs that are very usable. Because K-OS ONE uses the same disk format as MS/DOS you can easily take advantage of any FORTH programs for the PC.

K-OS FORTH is a full featured standard FORTH that runs with the K-OS ONE operating system. A full 32 bit stack is used that allows access to all of memory. The generic arithmetic operators are all 32 bit. There are some special 16 bit arithmetic operators for special cases. All of the system calls can be done in FORTH.

The source code is compiled to inline macros, JSR, or BSR. This allows for large programs but remains position independent and is very fast. Normal system ASCII files can be loaded. The traditional FORTH screens are not supported and there is no resident FORTH style 68000 assembler.

A set of utilities to use the K-OS file system is included. These include such things as opening files, and reading and writing to the disk. All the standard console I/O uses the operating system calls so it is portable to any set of hardware. Also a set of useful screen utilities is included that can be easily customised for any terminal. String operators and definers are included so you can use Pascal like strings.

An option is included so you can create standard .BIN executable files. A file can be saved in a fixed mode that can't be changed further for turnkey applications or can be saved in a modifyable live mode for further development. As long as the end user does not have the ability to add words or change words the resulting object can be distributed without royalties.

Full source code for the system is provided. The kernal is written in 68000 assembler and can be reassembled with the standard K-OS assembler. Many of the utilities are written in FORTH. A manual describes the action of each FORTH word and the theory on how the system is written.

\# \# \# \# \#

68000 Single Board Computer with Software -- $395.00

Hawthorne Technology's HT-68K TinyGiant is the lowest priced board of its kind. The HT68K is a complete 68000 including software. All you need to add is a terminal, disk drive, and a power supply to complete the system.

Hardware features begin with an 8 Mhz 68000. The HT-86K TinyGiant comes with 128k of RAM, no wait states. RAM is expandable to 512k by adding chips. There are two EPROMs (up to 64k of ROM). The EPROMs contain a Debug-/Monitor program and a boot loader.

It has a floppy disk controller (WD1770), that can control and select up to four 5 1/4" or 3 1/2" disk drives. The operating system reads and writes MS-DOS format diskettes. The board has a parallel printer port and two RS-232 serial ports. The serial lines are controlled by a 68681 DUART chip. The baud rates on the two serial lines are independent and programmable for rates from 50 to 38.4k baud. One of the serial ports is set up as a console device and the other is available for a modem or other serial device. System control is provided by vectored interrupt control, a timer, and a watchdog bus timer. It is possible to attach additional circuits to an HT-68k system through a full, unbuffered expansion bus.

The board is only 5.75 x 8.0 inches. This is the same size and shape as a 5 1/4 inch floppy disk drive. The board can actually mount to the side of a drive. The HT-68K requires the same voltage levels (+5VDC and +12VDC) and uses the same type of power connector as a standard 5 1/4 inch drive.

A complete software package is included. The board comes with the K-OS ONE operating system. K-OS ONE is a single user, single task operating system that reads and writes MS-DOS format disks (including the sub-directories). The package also contains a line editor, an assembler that uses standard Motorola nmeonics, and a compiler for HTPL. Complete source code is provided for the operating system, command processor, runtime library, and the BIOS debug PROMS. This package provides the tools you would need to modify the system to fit your special requirements. This software was designed to be easily customized.

Single piece price, $395.00. The operating system software is available separatly for $50.00, Qty one.

\# \# \# \# \#

The K-OS ONE Operating System

K-OS ONE was designed to be easy to implement, east to use, and low priced. It incorporates all of the features you would expect from an operating system like CP/M or MS-DOS, bringing these features to the 68000.

A simple design was used to allow implementation of the operating system on most 68000 hardware. Source code makes it possible to modify and maintain the operating system. With the operating system you can read and write ASCII files on MS-DOS format diskettes. The number of devices or open files is not restricted.

Package price: $50.00
The K-OS ONE operating system package includes:

Operating System - Object and HTPL source code
Command Processor - Object and HTPL source code
HTPL Compiler - Object code
68000 Assembler - Object code
Editor - Object code
Manual - Bound hardcopy

System Calls:

I/O Management
- Open Channel
- Create File
- Create Temporary File
- Close Channel
- Delete File
- Read
- Write
- Position
- Control Device
- Make Directory
- Get/Change Current Dir
- Delete Directory
- Start Directory Search
- Find Next File
- Lock/Unlock Block
- Get/Set File Date & Time
- Duplicate Channel
- Get/Set File Attributes
- Rename File
- Get Free Space On Disk

Non Disk
- Get/Set Control-Break Address/Action
- Get/Set Time & Date
- Get DOS ID
- Get/Release/Inquire About Memory Size

Program Control
- Get Last Terminate Code
- Get and Execute Program
- Time Delay Wait
- Terminate Program

Batch Control
- Get Next Line From .BAT
- Get .BAT Control Line
- Start .BAT File Processor

WHY WORK WITH A NEW OPERATING SYSTEM?

The small computer market is caught between two ruts today. On the small side is the PC and on the large side is Unix. The other players missed the boat by having a great (or so they thought) interface with nothing behind it to do any useful work. To be PC compatible is a dead end. The system is a kludge.

As developers try to squeeze the last bit of performance from the PC there will be problems. It is true that there are several million PCs in the world today. This doesn't mean there is a good market. Because the market is so large, it is hard (and expensive), for a small company to make themselves heard. There are public domain or low priced programs for every common application that anyone wants. These are hard to compete with. The pressure is to continue lowering prices while cutting profits. A business person needs to look at what point he can no longer afford to remain in this kind of market.

To break out of this rut a new system architecture is needed. Use the PC and clones where they fit but start to forge ahead in new directions. This doesn't mean trying to run a PC program on another machine. It is possible to emulate an 8086 on a 68000 but a full PC emulation is not worth while. In every case so far the emulation costs more than a PC clone. The interchange of disks on the other hand is very economical and easy to do. This protects the investment in data and makes it possible to add new machines without giving up the old ones.

The first step to a new architecture is to have a new operating system. It must be indpendent of a particular piece of hardware. This doesn't mean an operating system that can run on any processor. It means not being tied to a limited set of hardware like MS-DOS got tied to the PC hardware. The second step is to separate the application programs from the operating system itself. To use networks or multiple processors there must be a clear distinction between the logical and physical structure of the machine. To do otherwise would be to set a limit on what can be done with the operating system.

Bit map graphics and mice are good in some cases but to hobble an entire system with tricks that are not often needed or used is bad. The original use of mice was to allow people who knew little about computers to retreive information from them. They were not ones who had to put information into the computer or the more experienced users who want low cost and high performance. The operating systems like Mac and Atari are complex to the point where they hinder the development of new programs rather than helping. The windows that Microsoft has to sell are no better. Look at any stock broker, they have mulitple screens for dealing with different pieces of information at the same time, not tiny windows on a single screen.

A very promising area to look at for the future is multiple processor machines. With them, when more users are added to a system, more processing power is added also. This makes it possible to have multiple access without the slow down problems associated with trying to share a single CPU among many users. For cost sensitive or low performance users the muliple user approach can be used for lowest cost. For applications where high performance is important multiple processors can be used. If the operating system is independent of the hardware then the same program can be used in both cases.

Another area where multiple processors can be used to advantage is to split the operating system into component parts. For example the file management system can be duplicated for each disk in the system. Then when opening a file on disk A there would be no operating system overhead imposed on the system running disk B. If a disk is not involved then it would take no part in the activity. This allows large numbers of users to all access files at high speed if the load is balanced among different disks. A remote disk and file system can be like a new resource that can be easily added and integrated into a system. A company system can start small and grow to almost any size without requiring that the existing parts be replaced.

An individual workstation can have graphics and icons or not as need or tastes dictate. This will allow some users to access the system with icons but not impose that structure on other users of the system. It also means that some users could have windows and others could have more than one screen. Some users could have a local floppy disk or printer too. This approach to things opens a wide area of possible designs for working.

It is time to start planning for the future while the present generation of computers is still adaquate for today. If we don't start now we won't have the next generation when we need it. At Hawthorne Technology we are working on new ways of doing things. All of our programs are compatible with K-OS ONE at the system call level. Our hardware varies a lot. We even use PC Clones for some things. But any program that uses K-OS ONE system calls to access the hardware, and doesn't depend on special terminals, will run on any K-OS ONE system. We intend to keep this compatibility in the future for all systems whether distributed, multitask or single task. You can join us in this by using K-OS ONE or by writing applications to run with it. The number of people using K-OS ONE is increasing every day. There is a growing market for Languages and application software. Anyone interested in doing a package should contact us. We will help out in any way we can.

Q & A

QUESTIONS AND ANSWERS

These are some of the questions people have had asked in the last couple of months. I thought they might be of interest to others who have not had the time to get in touch with us.

Q: What kind of disk drive do I use with the TinyGiant?

A: We use 5 1/4", 360k floppy disk drives. This is also refered to as the PC, XT style. It is a standard style and available from many manufacturers and through dealers everywhere.

Q: Will the disk controller on the TinyGiant board control a 3 1/2" drive?

A: The TinyGiant board uses the WD1770 disk controller chip. The 1772 chip is used to control the 3 1/2" disk drive in the Atari ST. The only difference between the 1770 and the 1772 is the step rate. The 1770 is standard and the 1772 is faster. This means there should be no problems using the TinyGiant with a 3 1/2" drive. The driver and format routines would need to be modified.

Q: Where do I get connectors for power, etc. to set up my TinyGiant system?

A: There are many sources for connectors, depending on your location. If you have an electronics parts store in your area they will probably have what you need.

POWER: A standard floppy disk power connector will work for your TinyGiant power connector if you are wiring up your own supply. If you are looking for a place to buy this connector mail order, JAMECO Electronics has the parts (housing 480424, and pins 60619). They are in California at (415) 592-8097. If you are going to a distributor for the part it is an Amp connector (or equivilant), housing part # 1-480424-0 and conductor part # 60617-4 (qty 4 needed).

SERIAL: Because the Transmit, Receive and Ground signals are all in a row on one side of the connector, you can get away with a 3 pin single row .1" connector. You do have to be a little more careful installing it. (Straight ribbon cable won't work because of the signal arraingement on the pins of the connector.)

Editor Toolkit for K-OS ONE

A full screen editor is something that many K-OS ONE users have asked for. A simple line editor was included because line editors don't have to be customised to a particular terminal to be useful. For developing software a screen editor is much more productive. We are providing the source code for both the line editor and a screen editor so that either one can be easily customized for any terminal. This package supplies the tools you will need to create YOUR editor. You can change the default settings or the commands to be what you like. You can add commands to do special functions

You get the source code for a full screen editor and source code for the line editor that you got with your K-OS ONE Distribution Package. Both are written in HTPL and are easy to modify. Binary copies are also included. A text formatter like nrof is also included in source and object form. The manual describes how the editors work and gives helpful information on modifying them.

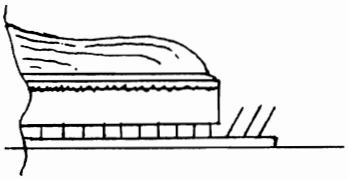
COMMAND SET FOR SCREEN EDITOR:

--------- FOR HELP SCREEN ^J or Line Feed ----------	
^E - line up	^G - del char
^X - line down	^T - del word
^R - page up	^L - repeat last find
^C - page down	^Y - delete line
^D - char right	^A - word left
^S - char left	^F - word right
----------	**----------**
^QB - to start of block	^KB - mark start of block
^QK - to end of block	^KK - mark end of block
^QS - start of line	^KR - read file
^QD - end of line	^KW - write file
^QR - top of file	^KD - exit editor
^QC - bottom of file	^KC - copy block
^QA - find and replace	^KV - move block
^QF - find	^KY - delete block
^QL - restore line	** block markers are
^QY - del to end of line	always column 1

EDITOR TOOLKIT $50.00

From: Hawthorne Technology (503) 254-2005
8836 SE Stark, Portland, OR 97216

PRINTER: The cable for a printer is a standard centronics style cable. The board connector should be a female, .1", dual row header like a ribbon cable connector. The board has a full 40 pin header. If you use a 34 pin cable, you can cut off the last 6 pins on your board, or bend them away so your cable will fit.



EXPANSION: My recommendation for expansion design would be to use a solder in female header on the expansion board so it plugs directly on to the TinyGiant. Samtec makes an Elevated Socket strip that is good for this. It comes in a 2x36 pin strip part # ESW-136-23-T-D.

A New 68K System

KNOCK YOUR SOCKS OFF KYSO TECHNOLOGY

At Editor Bartel's request, here is a brief description of my 68k system which I call Bolo. I used to call it the Homebrew Mac, but because of another company's aggressive tendency to defend its use of fruit names as product names, as well as the letters M, A, and C, I changed the name to something innocuous (I hope).

Bolo includes the following:

10 MHz 68000 processor	Serial I/O
512k RAM	Mouse input
16k ROM	Dual Floppy Drives
PC-style keyboard	8 interface slots
1024x480 video, 128K video RAM	

Bolo currently resides on a 13x14 inch PC board. It is built with standard parts ordered from the back of BYTE magazine and Radio-Electronics. Current software tools include a ROM monitor and a cross-assembler written in C for the PC. A C compiler would be a welcome addition to the tool set. With Hawthorne Tech's help, I expect to have K-OS running on it soon.

The idea behind Bolo is to have a computer completely "-able" by a user. By "-able" I mean that the computer is:

affordable	extendable
buildable	usable
repairable	understandable

Obviously, this is not a personal computer "for the rest of us"; a user must have some technical background in order to be an "-able" person. Are there are others in the world who can meet the specs of this machine? I don't know. That is why I am writing this article - the survey enclosed might help to make Bolo more "-able" to its users. If I can find the time and money, I would like to make Bolo available to others through a kit or an assembled and tested board.

In this survey, use 1 thru 5 as responses to the questions. One indicates a low amount of enthusiasm, desirablility, usefulness, etc., while a five indicates You Gotta Have It! Make copies for your friends. Anybody who sends me a stamped, self-addressed envelope can get the results of the survey directly. The results will also be published here in KNEWS.

Send your responses to Tony Ozrelic, PO Box 5246, Bend, OR 97708. Thanks in advance for taking the time to fill out the survey form.

I. Hardware

* * * What kind of processor do you like?

__ 8MHz 68000 __ 10MHz 68000 __ 10MHz 68020
__ 10MHz 68030 __ Other:

* * * Video Displays - Monochrome

__ 512x480 __ 1024x480 __ Other:

* * * Video Displays - Color

__ 512 x 480 x 32768 colors/pixel
__ 1024 x 480 x 16 colors/pixel __ Other:

* * * I/O

__ 3-1/2" floppy __ 5-1/4" floppy __ 8" floppy
__ 20Mb Hard disk __ Optical Disk __ Other:
__ Mouse Interface __ Serial Interface __ Parallel
__ Sound (simple) __ Sound (complex) __ MIDI i/f
__ LAN i/f __ Other: __ Other:

* * * Interface Slots

__ 8 bits Only __ 16 bits Only __ Both 8 & 16 bits
__ interrupts on slot __ DMA capability
__ Memory Expansion __ Simple interface-like Apple II
__ Complex i/f (NuBus,VME,etc.) __ VME compatible
__ PC compatible __ MAC II compatible
__ Other:

II. Software

* * * Tools

__ Assembler __ Editor __ Other:

* * * Languages

__ C compiler __ PASCAL __ FORTH __ BASIC
__ LISP __ PROLOG __ HTPL __ Other:

* * * Operating Systems

__ UNIX-like __ MS-DOS-like __ Macintosh-like
__ K-OS ONE __ Other:

* * * Applications

__ Spreadsheets __ Word Processing __ Desktop Publish
__ CAD __ Paint program __ Other:

III. If you had a Bolo, what would you use it for?

__ Software Development __ Hardware Development
__ Word Processing __ Desktop Publishing
__ Business/Financial __ Real-Time/Industrial
__ Hobby/Hacking __ Educational
__ Other:

IV. The Bottom Line: How much should it cost?

	100->199	200->	300->	400->	500-1000
Bare board with manual,software:					
Assembled & Tested with software:					
A&T w/o software, (OEM) qty 100 ea.:					
Entire system, case,PS,kbd,display:					

- --- - --- - --- - --- - --- - cut - --- - --- - --- - --- - --- - --- - --- -

READER RESPONSE / ORDER FORM

Please send me the following items:

Quantity	Description	Price
_____	_____	_____
_____	_____	_____
_____	_____	_____

Ship To:

Name _____

Street _____

City/State/Zip _____

____ VISA / MC ____ UPS C.O.D. ____ Pre Paid

Card # _____

Signature _____

Hawthorne Technology

8836 S.E. Stark, Portland, Or 97216

(503) 254-2005

_____ Please keep me on the mailing list for 68-KNEWS.
(We want to hear from you. If you haven't contacted us, and wish to continue to receive 68-KNEWS, let us know.)

#
#
##

HIGHLIGHTS:

HTPL -- Techniques

K-OS ONE -- Forth

K-OS ONE -- Full Screen Editor

#
#
##

68-KNEWS

Hawthorne Technology

8836 S.E. Stark
Portland, Or 97216

BULK RATE
US POSTAGE
PAID
PORTLAND, OR
PERMIT NO. 1116

Gem World Compu Systems Co., Inc.
1537 A Fourth Street
San Rafael, CA 94901