BULLETIN

TO: All KIM BASIC users.

To more specifically describe the start-up procedure for loading the KIM BASIC from cassette, we are offering the following:

(1) Have at least 16K of memory added on to your KIM-1, beginning at address 2000 HEX.

(2) We suggest you make a memory test at this point to be sure you have no bad memory bits.

(3) If you are loading via the KIM-1 hexadecimal keyboard, enter the following:

* AD 00F1 DA 00
  AD 17F9 DA 01
  + 00 + 1C + 00 + 1C + 00 + 1C
  AD 1873 G

* Start tape recorder with volume set at approximately 80% and treble on full.

* After three minutes, you should get a display of 0000.

* Flip your switch to transfer to your terminal.

* Press "RUBOUT" key on your terminal.

* On your terminal, type in "4065" (starting address for KIM 9 digit BASIC) followed by the space bar.

* Type "G".

* KIM BASIC should come up asking memory size.

* Tap the "RETURN" key. However, if you want to set aside memory at the high end, answer with the decimal value of the highest memory location KIM BASIC can use. KIM BASIC memory must be contiguous from 2000 HEX up.

* KIM BASIC should now ask for terminal width.

* Tap the "RETURN" key if you have a 72 character terminal width. If you have less than 72 characters across, or if you have more than 72 characters and plan on using the entire terminal width, type in actual terminal width.

(Cont. on Page 2)

P.O. BOX 523 Medina, Ohio 44256
(216) 725-4560
* BASIC will ask if you want SINE, COSINE, etc. We suggest you always answer "Y", for YES.

You should now be in BASIC.

(4) If you intend to transfer to the keyboard for the entire start-up, use the following procedure:

* Press "RS" on the KIM-1 hexadecimal keyboard.
* Press "RUBOUT" on your terminal keyboard.
* Type in the following on your terminal keyboard:

  00F1 (space bar) 00 .
  17F9 (space bar) 01 .
  00 . 1C . 00 . 1C . 00 . 1C .
  1873 (space bar) G

* At this point, turn on the cassette. After a successful load, the screen will display:

  KIM
  0000

* type starting address and routine as follows:

  4065 (space bar) G

* Answer questions on memory size, terminal width and transcendental functions as on page one.

You are now in BASIC.

Saving programs on cassette is accomplished as follows:

(1) Plan to save only one program per cassette. That program should be in your memory when you are ready to record. Use the following procedure:

* On your terminal keyboard, type "SAVE".
* Start your tape recorder. When leader has passed, type "G".
* A successful SAVE will be indicated by the following display on your screen:

  KIM
  0000

(2) Since 0000 is WARM START, get back to BASIC simply by typing "G".

In order to load a program from cassette back into KIM, use the following procedure:

* Enter BASIC as above and then type "LOAD".

(Cont. on Page 3)
* Tap the "RETURN" key.
* Start tape recorder with volume at approximately 80% and tone full treble.
* A successful load will be indicated by the following on the screen:

  KIM
  0000

* Since 0000 is WARM START, simply type a "C" to get back into BASIC.

Program is loaded.

Read the USAGE NOTES in the BASIC's documentation to better understand the initialization dialogue and the cassette interface.
WHERE THE 4 PRINTS TO OK TO CH/SCE AND INTEGRATE TO RS-232.

PRINTED ON A TELETYPE 4350-MC, 1799.90. OUR STOCK.

THIS COPY OF KIM BASIC LOADED INTO A KIM-1 AND RUN SUCCESSFULLY. --

11.18.78

RUN

OK

10 PRINT THE TITLE 4 PRINTS TO OK TO CH/SCE AND INTEGRATE TO RS-232.
90 PRINT "TITLE" ON A TELETYPE 4350-MC, 1799.90. OUR STOCK.
80 PRINT "THIS COPY OF KIM BASIC LOADED INTO A KIM-1 AND RUN SUCCESSFULLY"

S PRINT "THIS IS A TEST PROGRAM FOR THE MICROSOFT & DIGITAL BASIC FOR KIM-1."
NOTICE

For perfect load and dump operations using a cassette the following should be observed:

1. The small magnetic bar located on the polished surface of the recording head used in the tape recorder should be aligned to the magnetic information recorded on the tape. Typically the recording head is secured in place with two screws. One screw is spring loaded and allows the recording head to be shifted slightly for alignment with the information stored on the tape. It is possible the recording head was not aligned properly at the factory or the head has been jarred to an improper setting by rough handling of the recorder itself. If a production type tape has been recorded properly it can be used to find the proper setting of your recording head. While playing the tape the spring loaded recording head adjustment screw can be varied to obtain maximum brilliance (sharpness) of tone. Additionally, proper adjustment will allow programs to be stored on either side (track) of the cassette with no interference between the two. Recording head misalignment is one of the chief causes of improper loading from the cassette in the KIM-1 system.

2. If, after verifying proper recording head alignment, you still have problems, check the setting of VR-1 (the 5K potentiometer) located to the left of the keyboard. Connect a jumper from terminal P on the application connector to terminal L on the application connector. Next, connect a DC volt meter between terminal X on the expansion connector and ground. Then, referring to the KIM-1 User's manual Page E2 adjust the potentiometer to get a reading of +1.4V (no less than .7V and no more than 3.0V). This is a very touchy adjustment. If you have properly aligned your recorder head and the KIM-1 is properly calibrated you should have no problem dumping to your recorder and loading from your recorder. Should you still have a problem you might check to be sure that the tape you are trying to load in your KIM-1 was recorded on a properly adjusted recorder head. If this was not done, the program can be saved by readjusting your recorder head in the wrong position so that it properly lines up with the bad-cassette, loading to KIM and then readjust your head to its proper setting and re-recording the program from your KIM to a clean cassette.
Your KIM-1 Basic by Microsoft cassette has been recorded using HYPERTAPE developed by Jim Butterfield of Toronto, Canada and published in KIM-1 Users Notes, Issue 2, Pages 12, 13 and 14. This program enables you to record and play back in 1/6th of the time required using the standard KIM-1 tape routines. Playing back into your KIM-1 from a HYPERTAPE recording follows the same procedure as the standard KIM-1 tape loading routine using 1873 and GO. HYPERTAPE is more sensitive to the adjustment of the tape recorder head. If you do not get a load on the first try then there is a strong possibility that your tape recorder head has not been aligned exactly the same as ours. You can "tune in" to the tape by using some delightful programs published in KIM-1 Users Notes.

1. Before changing the setting of your tape recorder head take one of your own cassettes and generate a SYNC STREAM on your own recorder using the program from Page 11 of KIM-1 Users Notes, Volume 1, Issue 2.

2. Now, load the VUTAPE into your KIM-1. VUTAPE was written by Jim Butterfield of Toronto, Canada and published in KIM-1 Users Notes, Issue 2, Page 12.

3. After loading VUTAPE and OOOO, press GO. The last character in your display will come on in a random fashion. Now go to your KIM BASIC cassette and locate the 30 second SYNC STREAM (you can tell it by the steady sound) which we have recorded immediately following the end of the KIM-1 BASIC. This is at about 3 minutes and 15 seconds into the tape. Adjust your volume control to about 2. While playing this sync stream into your KIM-1, adjust your tape recorder head screw so that the sync pulses "lock in" on the right end of display. This adjustment should be very close to the original adjustment of your head. Once locked in, lower the volume control to about 1 or even .5 and adjust again.

4. Next you might want to check your phase lock loop (VR-1) adjustment on your KIM-1. This is easiest done by using the PLL SET program by Louis Edwards, Jr. of Trenton, N.J. and published on Page 3 of KIM-1 Users Notes Issue 5. Now go to the beginning of the KIM-1 BASIC cassette, set up your load routine. Enter Ident 01 at 17F9. Be sure 00F1 is loaded with 00. Go to 1873, set volume control at about 8 and GO. You should load in three minutes. Now refer to the enclosed documentation for BASIC operating instructions.

KIM-1 User Notes is published by Erick Rehnke, 109 Center Street, West Norriton, PA. 19401. Subscriptions are 6 issues for $5.00 in U.S.A. and Canada, $10.00 for 6 issues elsewhere. Johnson Computer will accept subscriptions on purchase orders. Individuals making checks payable to "KIM-1 Users Notes". Also, "The First Book of KIM", by ORB (Ocker, Rehnke & Butterfield) has reprinted most important programs published in issues 1 through 5 and also includes excellent text on KIM and the 6502. Price is $9.50 available through Johnson Computer, P.O. Box 523, Medina, Ohio 44256.

SYNC STREAM - 0000 A0 BF 8C 43 17 A9 16 20 7A 19 D0 F9

VUTAPE - 0000 DB A9 7F BD 41 17 A9 13 85 E0 BD 42 17 20 41 1A 46 F9 05 F9 8F 09
0016 BD 40 17 C9 16 D0 E9 20 24 1A C9 2A DO F5 A9 00 BD E9 17 20 24 1A
002C 20 00 1A DO D5 A6 E0 E8 E8 E0 15 DO 02 A2 09 BD E0 3E 42 17 AA
0041 BD E7 1F BD 4D 17 DO DB

PLL SET - 1780 A9 07 BD 42 17 A9 01 BD 01 17 85 E1 A9 7F BD 41 17 A2 09 A0 07
1795 2C 42 17 30 02 A0 38 BC 40 17 8E 42 17 2C 47 17 10 FB E6 E2 30 04
17AB A9 91 DO 03 A9 93 EA 8D 44 17 A9 01 45 E1 85 E1 BD 00 17 E8 E8
17CO E0 15 DO CF F0 CB
TO: Users of KIM-1 BASIC by Microsoft

Implementation of a CONTROL C is difficult due to the nature of the KIM I/O port. Because only a bit at a time comes into the port, it is impossible to handle a character typed during computation. By the time BASIC tries to detect a CONTROL C character, several bits of data may have already passed through the port.

Exiting to the monitor and re-entering at the WARM START location will stop the program, however, the "CONT" command will not work properly. In fact, exiting to monitor can leave BASIC in a state where all variable accesses hang the machine until "RUN", "NEW", or "CLEAR" is typed or a program line is changed, however, this is very unlikely.

The patch given below will cause any character typed to be treated like a CONTROL C:

<table>
<thead>
<tr>
<th>Memory Location</th>
<th>Code</th>
<th>Mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>KB-6 2614</td>
<td>26E3</td>
<td>LDA #3</td>
</tr>
<tr>
<td>KB-9 2615</td>
<td>26E4</td>
<td>03</td>
</tr>
<tr>
<td>KB-9 2616</td>
<td>26E5</td>
<td>18</td>
</tr>
</tbody>
</table>

The CONTROL 0 facility can be handled by POKEing or using the KIM monitor to set the CONTROL 0 flag location of 0013 in KB-6, or 0014 in KB-9, to FF for no output or to 00 for output.

Some people have expressed the desire to have a NULL greater than 8. This can be increased to 240 decimal by entering F2 at 2663 for KB-6, or 2732 for KB-9.

For those of you who will find the information helpful and have the ability to use it, on page two are the page 0 locations used by both the 6 digit and the 9 digit version.

P.O. BOX 523 MEDINA, OHIO 44256
(216) 725-4560
Page 0 is configured as follows:

KB-6 uses all locations 0-D4 and FF
KB-9 uses all locations 0-DC and FF

<table>
<thead>
<tr>
<th>KB-6</th>
<th>KB-9</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>JMP to WARM START BASIC.</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>Address of routine to transfer USR argument to y,A (AYINT).</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>Address of routine to transfer (y,A) to result of USR function (GIVAYF).</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>FLAG set to FF if output is suppressed (CONTROL 0 mode). Set to 0 otherwise.</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
<td>Number of NULLS to print.</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>Current terminal column (equal to POS(0)).</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>Line Length.</td>
</tr>
<tr>
<td>17</td>
<td>18</td>
<td>Position beyond which there are no more comma fields. Equal to 14*(INT(line length/14)-1).</td>
</tr>
<tr>
<td>1A</td>
<td>1B</td>
<td>Input buffer .72 decimal bytes.</td>
</tr>
<tr>
<td>76</td>
<td>78</td>
<td>Pointer to start of program.</td>
</tr>
<tr>
<td>78</td>
<td>7A</td>
<td>Pointer to start of simple variable table.</td>
</tr>
<tr>
<td>7A</td>
<td>7C</td>
<td>Pointer to start of array table.</td>
</tr>
<tr>
<td>7C</td>
<td>7E</td>
<td>First location unused by array table.</td>
</tr>
<tr>
<td>7E</td>
<td>80</td>
<td>Lowest location used by string data.</td>
</tr>
<tr>
<td>82</td>
<td>84</td>
<td>Highest memory location in use by BASIC.</td>
</tr>
<tr>
<td>84</td>
<td>86</td>
<td>Current line number.</td>
</tr>
<tr>
<td>A9</td>
<td>AE</td>
<td>Floating accumulator</td>
</tr>
<tr>
<td>B9</td>
<td>CD</td>
<td>Routine to read a character from current program position.</td>
</tr>
<tr>
<td>D1</td>
<td>D8</td>
<td>Current random number.</td>
</tr>
<tr>
<td>D5</td>
<td>DD</td>
<td>First unused page 0 location.</td>
</tr>
<tr>
<td>FF</td>
<td>FF</td>
<td>Used by STR$ function.</td>
</tr>
</tbody>
</table>
Following is information for reading a line:

(1) The routine to input a line from the terminal is located at:

KB-6 . . . . . 2351-2386 (start at 2357)
KB-9 . . . . . 2420-2455 (start at 2426)

(2) The compare for a (line delete) is at:

KB-6 . . . . . 236C
KB-9 . . . . . 243B

(3) The compare for a (character delete) is at location:

KB-6 . . . . . 2370
KB-9 . . . . . 243F

(4) Note that codes < 20 HEX and >= 70 HEX have already been ignored by the compare above these.

If you are having any problems with your KIM-1 BASIC by Microsoft, please write to us. If at all possible, include a print-out illustrating your difficulty. Document your print-out with handwritten notes indicating the difficulty.

All future orders for KIM-1 BASIC by Microsoft will include the patches for CONTROL C and NULL.

KIM-1 9 digit BASIC by Microsoft is now available in a version adapted for the TIM monitor manufactured by MOS Technology. TIM stands for Teletype Input Monitor and is often used by people developing a CPU of their own design. The TIM is part #6530-004 and sells for $14.15. The manual for application of the TIM sells for $4.95. Both items are available from Johnson Computer.

A PROMable version of the KIM-1 9 digit BASIC by Microsoft will soon be available. This version can be furnished on cassette or paper tape. It will also be available already programmed into 2708's or 2716's, as needed, on the KM-8KRO 8K/16K PROM board designed for direct use with the KIM-4 Motherboard, or on an S-100 PROM board for the KEM or KINSI. Pricing is available on request.

In-house object code rights for use on a specific entire project are available for $750.00. This would be appropriate for a user who would otherwise have to purchase several copies of BASIC authorized for use on a specific KIM unit. As an example, using KIM in an in-house test stand and then building ten more test stands, each using a KIM.

In-house source rights to all versions of 6502 BASIC is available to customers who would like to customize the program for each KIM-1. The price is $3,000.00 for the in-house for each use of the source code.

OEM source code rights are available for making Microsoft's 6502 BASIC a part of a product being manufactured for resale. The price for source code rights to an OEM is $3,000.00 plus $35.00 for each copy up to 1,000 copies ($35,000.00) after which they then own the rights. This can be purchased outright for $21,000.00.
APPLICATION NOTE: Microsoft has advised us that the value of .1 cannot be represented exactly in binary floating point. Programs magnify this inaccuracy until it shows up on print-out. This can be handled by:

1. Always use integers which are exact and then scale. As an example, 14 x 1.23 would be (14*123)/10.

2. Use tests such as ABS(X-Z) < 1E-6 or STR$(X)$=STR$(Z)$.

3. Round out to the number of digits you want before printing.

The way to save data using Microsoft KIM BASIC is to:

1. Have it in DATA statements and use CSAVE.

2. Write your own USR function to perform this.

Microsoft will provide access to cassette data in a future release if there is enough general interest. Drop us a card if you are interested.

If BASIC is waiting for an input and you type a carriage return, you will exit your program and return to the Immediate Mode of BASIC. This is intentional. You can return to the program using CONT. If you want to prevent accidentally leaving the program in this manner, you can use POKE or the KIM Monitor to change the A5 to an A9 at location 10920 decimal (2AA8 HEX) for the 9 digit version or at location 10688 decimal (29C0 Hex) for the 6 digit version. Now, however, you will only be able to exit BASIC by using CONTROL C, encountering a STOP in the program, or if the program runs to an END.

SOFTWARE: Three KIM-1 routines have been written by Ralph Bugg, a user, which are compatible with the KIM-1 9 digit BASIC. These routines are as follows:

1. A video drive making the Kent-Moore video board catalog 60083 (32 x 16)
   compatible with KIM BASIC . . . . . . . . . . $4.00 postage paid

2. A video driver making the Kent-Moore video board catalog 6X117 (64 x 16)
   compatible with KIM BASIC . . . . . . . . . . $4.00 postage paid

3. A routine for operating KIM BASIC from a parallel
   input keyboard . . . . . . . . . . . . . . . . . . . . $4.00 postage paid

4. Routine for output to a Baudot (5 level) Teletype . $4.00 postage paid

From time to time we will make a mailing of miscellaneous information gathered on the KIM BASIC and related items. If you have any tips you would like to share with others, please send them in. At this point, there is no schedule or promise for the next issue but it will be sent out when enough information is accumulated to make it worthwhile.

HYPERTAPE TIP: For those of you who are unaware of Hypertape, this is a program written by Jim Butterfield of Toronto, Canada and published in Eric Rehke's KIM-1 User Notes, Volume I, Issue 2. Hypertape allows you to record on cassette in 1/6 the time of standard KIM dump routine. Ralph Bugg suggests you place Hypertape at 0300 and enter the following changes:

035F 60
275C 20 00 03
From the Authors of KIM-1 BASIC

Before a computer can perform any useful function, it must be "told" what to do. Unfortunately, at this time, computers are not capable of understanding English or any other "human" language. This is primarily because our languages are rich with ambiguities and implied meanings. The computer must be told precise instructions and the exact sequence of operations to be performed in order to accomplish any specific task. Therefore, in order to facilitate human communication with a computer, programming languages have been developed.

KIM-1 BASIC* is a programming language both easily understood and simple to use. It serves as an excellent "tool" for applications in areas such as business, science and education. With only a few hours of using BASIC, you will find that you can already write programs with an ease that few other computer languages can duplicate.

Originally developed at Dartmouth University, BASIC language has found wide acceptance in the computer field. Although it is one of the simplest computer languages to use, it is very powerful. BASIC uses a small set of common English words as its "commands". Designed specifically as an "interactive" language, you can give a command such as "PRINT 2+2", and KIM-1 BASIC will immediately reply with "4". It isn't necessary to submit a card deck with your program on it and then wait hours for the results. Instead the full power of the KIM-1 is "at your fingertips".

Generally, if the computer does not solve a particular problem the way you expected it to, there is a "Bug" or error in your program, or else there is an error in the data which the program used to calculate its answer. If you encounter any errors in BASIC itself, please let us know and we'll see that it's corrected. Write a letter to us containing the following information:

1) System Configuration
2) Version of BASIC
3) A detailed description of the error. Include all pertinent information such as a listing of the program in which the error occurred, the data placed into the program and BASIC printout.

All of the information listed above will be necessary in order to properly evaluate the problem and correct it as quickly as possible. We wish to maintain as high a level of quality as possible with all of our KIM-1 software.

NOTE: BASIC is available under license or purchase agreements. Copying or otherwise distributing Microsoft software outside the terms of such an agreement may be a violation of copyright laws or the agreement itself.

If any immediate problems with Microsoft software are encountered, feel free to give us a call at 216/725-4560. The joint authors of the KIM-1 BASIC Interpreter, Bill Gates, Paul Allen and Monte Davidoff, will be glad to assist you.

We hope that you enjoy KIM-1 BASIC, and are successful in using it to solve all of your programming needs.

* KIM-1 is a registered Trademark of MOS TECHNOLOGY
  BASIC is a registered trademark of Dartmouth University
We recommend that you try each example in this section as it is presented. This will enhance your "feel" for BASIC and how it is used.

Once your I/O device has typed "OK", you are ready to use KIM-1 BASIC.

**NOTE:** All commands to KIM-1 BASIC should end with a carriage return. The carriage return tells BASIC that you have finished typing the command. If you make a typing error, type a back-arrow (←), usually shift/0 or an underline, to eliminate the last character. Repeated use of "←→" will eliminate previous characters. An at-sign (@) will eliminate the entire line that you are typing.

Now, try typing in the following:

```
PRINT 10:4 (end with carriage return)
```

KIM-1 BASIC will immediately print:

```
6
OK
```

The print statement you typed in was executed as soon as you hit the carriage return key. BASIC evaluated the formula after the "PRINT" and then typed out its value, in this case 6.

Now try typing in this:

```
PRINT 1/2.3*10
```

(meaning "divide by 2.3 multiply by 10")

BASIC will print:

```
.5 30
```

As you can see, KIM-1 BASIC can do division and multiplication as well as subtraction. Note how a "", (comma) was used in the print command to print two values instead of just one. The comma divides the 72 character line into 5 columns, each 14 characters wide. The last two of the positions on the line are not used. The result is a "", causes BASIC to skip to the next 14 column field on the terminal, where the value 30 was printed.

Commands such as the "PRINT" statements you have just typed in are called Direct Commands. There is another type of command called an Indirect Command. Every Indirect command begins with a Line Number. A Line Number is any integer from 0 to 64000.

Try typing in the following lines:

```
10 PRINT 2*3
20 PRINT 2-3
```

A sequence of Indirect Commands is called a "Program". Instead of executing indirect statements immediately, KIM-1 BASIC saves Indirect Commands in the KIM-1’s memory. When you type in RUN, BASIC will first execute the lowest numbered indirect statement that has been typed in, then the next highest, etc. for as many as were typed in.

Suppose we type in RUN now:

```
RUN
```

KIM-1 BASIC will type out:

```
5
1
OK
```
In the example above, we typed in line 10 first and line 20 second. However, it makes no difference in what order you type in indirect statements. BASIC always puts them into correct numerical order according to the Line Number.

If we want a listing of the complete program currently in memory, we type in LIST. Type this in:

```
LIST
```

KIM-1 BASIC will reply with:

```
10 PRINT 2+3
20 PRINT 2-3
OK
```

Sometimes it is desirable to delete a line of a program altogether. This is accomplished by typing the Line Number of the line we wish to delete, followed only by a carriage return.

Type in the following:

```
10
LIST
```

KIM-1 BASIC will reply with:

```
20 PRINT 2-3
OK
```

We have now deleted line 10 from the program. There is only one way to get it back. To insert a new line 10, type in 10 followed by the statement we want BASIC to execute.

Type in the following:

```
10 PRINT 2*3
LIST
```

KIM-1 BASIC will reply with:

```
10 PRINT 2*3
20 PRINT 2-3
OK
```

There is an easier way to replace line 10 than deleting it and then inserting a new line. You can do this by just typing the new line 10 and hitting the carriage return. BASIC throws away the old line 10 and replaces it with the new one.

Type in the following:

```
10 PRINT 3-3
LIST
```

KIM-1 BASIC will reply with:

```
10 PRINT 3-3
20 PRINT 2-3
OK
```

It is not recommended that lines be numbered consecutively by increments of one (e.g., 1, 2, 3...). It may become necessary to insert a new line between two existing lines. An increment of 10 between line numbers is generally sufficient.

If you want to erase the complete program currently stored in memory, type in "NEW". If you are finished running one program and are about to read in a new one, be sure to type in "NEW" first. This should be done in order to prevent a mixture of the old and new programs.

Type in the following:

```
NEW
```

KIM-1 BASIC will reply with:

```
OK
```

Now type in:

```
LIST
```

KIM-1 BASIC will reply with:

```
OK
```
# Companion to the Schaum's Outline Series' Programming with BASIC

<table>
<thead>
<tr>
<th>SECTION</th>
<th>PAGE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2</td>
<td>4</td>
<td>Skip to page 6, just after example 1.4. TimeShare and KIM BASIC share the interactive capability.</td>
</tr>
<tr>
<td>1.3</td>
<td>5</td>
<td>Rule 1: KIM allows multiple statements per line if each statement is separated from any previous statement of the same line by a colon (:).</td>
</tr>
<tr>
<td>2.3</td>
<td>14</td>
<td>For KIM, each NUMERIC VARIABLE must consist of a letter, a letter followed by an integer, or a letter followed by a letter. Similar conditions apply to a STRING VARIABLE with all STRING VARIABLES preceded by a dollar sign ($). FNS is a specific letter-letter-$ that is not a usable string variable name on KIM.</td>
</tr>
<tr>
<td>2.7</td>
<td>17</td>
<td>Rule 3, Ex. 2.9 — Raising a number to the 1/2 or .5 power will result in the square root of that number. Similarly, to obtain the cube root of a number, raise it to the 1/3 or .333333 power. Again, the fourth root of a number is the same as that number raised to the 1/4 or .25 power.</td>
</tr>
<tr>
<td>2.8</td>
<td>18</td>
<td>Ex. 2.13 — The first, third and fourth statements are not valid with KIM. Multiple assignments are not valid. NOTE: On KIM, the word LET is optional.</td>
</tr>
<tr>
<td>2.9</td>
<td>19</td>
<td>Rule 4 — Also strings containing &quot;$&quot;.</td>
</tr>
<tr>
<td>2.11</td>
<td>22</td>
<td>The END statement is optional with KIM. If there is an END statement it does not have to be at the end of the program but can be embedded in the program.</td>
</tr>
<tr>
<td>2.17</td>
<td>24</td>
<td>Ex. 2.28 — Invalid on KIM. Instead use: 40 LET X1=1+(B+1)(2<em>A):REM CALCULATE FIRST ROOT 50 LET X2=1+(B+1)(2</em>A):REM CALCULATE SECOND ROOT Where the word LET is optional.</td>
</tr>
<tr>
<td>3.2</td>
<td>38</td>
<td>Does not pertain to KIM BASIC.</td>
</tr>
<tr>
<td>3.3</td>
<td>39</td>
<td>KIM has the prompting word &quot;OK&quot; rather than the symbol &quot;.&quot;.</td>
</tr>
<tr>
<td>3.4</td>
<td>40</td>
<td>To delete characters with KIM type an underscore(_I or back going arrow_I and one previous character is deleted. Type it once and one previous character is deleted. Type it twice and two previous characters are deleted, and so on.</td>
</tr>
<tr>
<td>3.5</td>
<td>41</td>
<td>See this manual for KIM system commands and how to use them.</td>
</tr>
<tr>
<td>3.6</td>
<td>43</td>
<td>Does not pertain to KIM.</td>
</tr>
<tr>
<td>4.1</td>
<td>53</td>
<td>See paragraph preceding Ex. 4.2 — KIM BASIC does not ignore trailing blanks.</td>
</tr>
<tr>
<td>4.2</td>
<td>54</td>
<td>See paragraph 2 — KIM BASIC also allows GOTO rather then THEN.</td>
</tr>
<tr>
<td>4.3</td>
<td>56</td>
<td>See Ex. 4.8 — THEN cannot replace GOTO in ON-GOTO statement.</td>
</tr>
<tr>
<td>4.4</td>
<td>58</td>
<td>Again, END is optional for KIM and can appear anywhere else than the physical end of the program.</td>
</tr>
<tr>
<td>4.5</td>
<td>63</td>
<td>Only STEP with KIM, not BY.</td>
</tr>
<tr>
<td>4.6</td>
<td>63</td>
<td>Running variable after NEXT is optional when obvious.</td>
</tr>
<tr>
<td>4.6</td>
<td>64</td>
<td>KIM loops will execute once under conditions 3(1a), 3(1b), and 3(1c).</td>
</tr>
<tr>
<td>6.1</td>
<td>113</td>
<td>For KIM: Functions must appear in the program after the line on which they are used. They cannot be grouped at the end of a program but should be grouped near the beginning. Also, no string functions, no multiple line functions, no multiple argument functions, and no function without an argument (Use a place holder variable e.g. 10 DEF FNY=CT<em>2+B becomes 10 DEF FNXI=CT</em>2*B where the argument of FNX is a variable that does not appear in the definition).</td>
</tr>
</tbody>
</table>
Does not pertain to KIM BASIC.

Chapter 7 does not pertain to KIM.

Chapter 8 does not pertain to KIM.
AND 2 IF A) 5 AND B) 2 THEN 7. If expression 1
(A) 5 AND expression 2 (B) 2 are both true, then branch
to statement 7.

ASC(X) Returns the ASCII numeric value of the first
character of the string expression X. An FC error will occur if
X is not a string.

ATN(X) Gives the arctangent of the argument X. The
result is returned in radians and ranges from -PI/2 to PI/2.

ABS(X) Gives the absolute value of the expression X.

CHR$(I) Returns a one character string whose single char-
acter is the character whose value is I. The result
must be > 0 and < 255.

CLEAR Clears all variables, resets FOR & GOSUB state
and STORE data.

CONT Continues program execution after a control or a
STOP statement is executed. You cannot con-
tinue after any error, after nullifying your program, or
before your program has been run. One of the main
purposes of CONT is debugging. Suppose at some point after running
your program, you want to test some program lines and
the program is performing some time consuming calculation, but it
will be because you have taken into an "infinite loop". An infinite loop is a
series of BASIC statements from which there is no escape. The KIM-1 will
keep executing the "infinite loop", store some of the values of
your variables. After examining these values you may become satisfied that your program is func-
tioning correctly. You should then type in CONT to continue executing your program where it left off, or type a direct GOTO statement to resume execution of the program at a different line. You could also use ASCII (LET) command to set some of your variables to different values. Remember, if you control a program and expect to con-
tinue in it later, you must not set any errors or type in any new
variables, because your program will be lost and you will get an "ERR" (continue-not error). It is impossible to con-
tinue a correct command. CONT always resumes execution at the
next statement to be executed in your program when called.

COS(X) Gives the cosine of the expression X. X is inter-
preted as being in radians.

DATA Specifies data, read from left to right. Information
positioned in data statements in the same order as it will be read in the
program.

DEF 100 DEF FN(A)=V/B+B/C+D Use the function names like the built-in function (SQRT, SGN, ABS, etc.)
through the use of the DEF statement. The name of the
function is "FNA" followed by any valid variable name, for example:
FNK, FNJ, FNK, FNM, FN2, FN2. User defined functions are
restricted to one line. A definition may be defined to be
any expression, but may only have one argument, in the
example B C are variables that are used in the program.
Executing the DEF statement defines the function. User
defined functions can be referenced by executing another
DEF statement for the same function. User defined string
functions are not allowed. "V" is called the dummy variable.

102 2 FN(A) Execution of this statement following the above
would cause Z to be set to 3/B+C, but the value of V
would be unchanged.

DIM 112 DIM (A), (B) Allocates space for arrays.

114 DIM (A), (B) Arrays can have more than

one dimension. Up to 255 dimensions are allowed, but due
to the restriction of 12 characters per line the practical
maximum is about 34 dimensions. Arrays can be dimensioned
with the exception that in the program execution. If an array is not
explicitly dimensioned with a DIM statement, it is assumed to be
a single dimensioned matrix of whose single subscript may
range from 0 to 10 (eleven elements). 117 A(1)4 If this

statement was encountered before a DIM statement for A
was found in the program, it would be an array of 16 dimen-
sions. The last dimension of an array is fixed, so A(1)4 = 117.

EXP(X) Gives the constant 'E' (2.71828) raised to the
power X. X is an integer that can be passed to EXP with overflow occurring in 87.3365.

FOR 300 FOR V=1 TO 3 STEP 5 Use NEXT state-
ment V is set equal to the value of the expression following
the equal sign, in this case I. This value is called the initial
value. Then the statements between FOR and NEXT are
executed. The final value is the value of the expression
following the TO. The step is the value of the expression
following the STEP. When the NEXT statement is encountered
the step is added to the variable. 310 FOR V=1 TO 3
Step 5 would specify, it is assumed to be one. If the step is
positive and the new value of the variable is (+ final value
(9.3 in this example)), or the step value is negative and the
new value of the variable is < the final value, then the
first statement following the FOR statement is executed.
Otherwise, the statement following the NEXT statement is
executed. All FOR loops execute the statements between the
FOR and the NEXT at least once, even in cases like FOR
V=1 TO 9. 320 FOR V=10 TO 3 A=45 STEP GSR (IT) Note
that expressions (formula) may be used for the initial, final
to and step values in a FOR loop. The values of the ex-
pression are calculated before the loop begins. The FOR
NEXT loop is executed. 320 FOR V=9 TO 1 STEP
-1, when the statement after the NEXT is executed, the
loop variable is never equal to the final value, but is equal to
whatever value caused the FOR,... NEXT loop to terminate. The
statements between the FOR and its corresponding
NEXT in both examples above are executed 10 times.
330 FOR V=1 TO 10; FOR W=1 TO NEXT V; W=4 Error: the values of V and W have the same
index variable. FOR loop nesting is limited only by the
available memory.

FREE (DX) or FREED (X) 270 PRINT FREE(x) Gives the num-
ber of memory bytes currently unused by BASIC. Memory
allocated for STRING space is not included in the count
returned by FREE.

GOTO Branches to the specified statement.

GOSUB Branches to the specified statement until a RE-
TURN is encountered; when a branch is then made to the
statement after the GOSUB. GOSUB nesting is limited only by
the available memory.

IF, . . . , THEN, except that if IF . . . , GOSUB must be followed by a value number.

104 IF, THEN IF X) 120 THEN X Branches to specified
statement if the relation is True. 20 IF X 7 THEN PRINT "A LESS THAN 0"
Execute all of the line's characters starting from the
remainder of the line after the THEN if the relation is True.

INPUT Requests data from the terminal to be typed in.
Each value must be separated from the preceding value
by a comma. The last value typed should be followed by a
carriage return. A "RETURN" is typed as a prompt character. If
the value typed is a number, "A" is printed on the screen, if a
"A" is printed after a string, then the rest of the string will
be ignored. Strings must be input in the

usual format as they are specified in DATA statements.

INPUT "(VALUE); ": Optionoly type a prompt string
("VALUE") before requesting data from the terminal. If a
string typed to an input statement, then the INPUT command
returns to command mode. Tyipig CONT after an INPUT command
has been interrupted will cause the execution to resume at
the INPUT statement.

INTEXT Returns the largest integer less than or equal to its
argument X. For example: INT (23/3), INT (7/7), INT
LET

LST

LOAD

LOG

MIDS

NEXT

NOT

NULL

ON

OR

PEEK

POKE

PRINT

READ

REM

RESTORE

RETURN

SAVE

SIGN

SIN

SPACE

STOP

STR

TAB

(1-1) INT (2-2) -2, INT (1-1) = 1.

The following would round X to D decimal places:

INT (X×10D) + QUIET 10

LEFTS(X, I) Gives the leftmost I characters of the string expression X as characters (bytes). Non-printing characters and blanks are counted at all length.

LEN(X) Gives the length of the string expression X in characters (bytes). Non-printing characters and blanks are counted at all length.

LET

LST

LOAD

LOG

MIDS

NEXT

NOT

NULL

ON

OR

PEEK

POKE

PRINT

READ

REM

RESTORE

RETURN

SAVE

SIGN

SIN

SPACE

STOP

STR

TAB

(1-1) INT (2-2) -2, INT (1-1) = 1.

The following would round X to D decimal places:

INT (X×10D) + QUIET 10

LEFTS(X, I) Gives the leftmost I characters of the string expression X as characters (bytes). Non-printing characters and blanks are counted at all length.

LEN(X) Gives the length of the string expression X in characters (bytes). Non-printing characters and blanks are counted at all length.

LET

LST

LOAD

LOG

MIDS

NEXT

NOT

NULL

ON

OR

PEEK

POKE

PRINT

READ

REM

RESTORE

RETURN

SAVE

SIGN

SIN

SPACE

STOP

STR

TAB

(1-1) INT (2-2) -2, INT (1-1) = 1.

The following would round X to D decimal places:

INT (X×10D) + QUIET 10

LEFTS(X, I) Gives the leftmost I characters of the string expression X as characters (bytes). Non-printing characters and blanks are counted at all length.

LEN(X) Gives the length of the string expression X in characters (bytes). Non-printing characters and blanks are counted at all length.

LET

LST

LOAD

LOG

MIDS

NEXT

NOT

NULL

ON

OR

PEEK

POKE

PRINT

READ

REM

RESTORE

RETURN

SAVE

SIGN

SIN

SPACE

STOP

STR

TAB

(1-1) INT (2-2) -2, INT (1-1) = 1.

The following would round X to D decimal places:

INT (X×10D) + QUIET 10

LEFTS(X, I) Gives the leftmost I characters of the string expression X as characters (bytes). Non-printing characters and blanks are counted at all length.

LEN(X) Gives the length of the string expression X in characters (bytes). Non-printing characters and blanks are counted at all length.

LET

LST

LOAD

LOG

MIDS

NEXT

NOT

NULL

ON

OR

PEEK

POKE

PRINT

READ

REM

RESTORE

RETURN

SAVE

SIGN

SIN

SPACE

STOP

STR

TAB

(1-1) INT (2-2) -2, INT (1-1) = 1.

The following would round X to D decimal places:

INT (X×10D) + QUIET 10

LEFTS(X, I) Gives the leftmost I characters of the string expression X as characters (bytes). Non-printing characters and blanks are counted at all length.

LEN(X) Gives the length of the string expression X in characters (bytes). Non-printing characters and blanks are counted at all length.
position (column) on the terminal. May be used only in PRINT statements. Zero is the leftmost column on the terminal. If the carriage is beyond position 1, then no printing is done. I must be = 0 and =555.

TAN(X) Gives the tangent of the expression X. X is interpreted as being in radians.

USHER (Y) = Table, ZEROS. Calls the user's machine language subroutine with the argument Y. See CODE, PEAK and USR discussion.

VAL(S) Returns the string expression S converted to a number. For instance, VAL("3.1") = 3.1. If the first non-zero character of the string is not a plus (+) or minus (-) sign, a digit or a decimal point . then zero will be returned.

WAIT 805 WAIT 1, 2, 3, 4. This statement reads the status of location 1, exclusive OR's K with the status, and then AND's the result with J until a non-zero result is obtained. Execution of the program continues at the statement following the WAIT statement. If the WAIT statement only has two arguments, K is assumed to be zero. If you are waiting for a character, R, S, or T in the corresponding position of 1, 2, 3, and 4 must be = 0 or =555.

SYMBOLS AND SPECIAL KEYS
- Assign a value to a variable. The LET is optional.
- Negation. Note that == is subtraction, while - is negative.
  
  | (usually a shift/l) Exponentiation 0 TO 127 or any other power <= 0, A [B, with A negative and B not an integer gives an FC error.

- Multiplication
- Division

- Addition. String concatenation. The resulting string must be less than 156 characters in length or an LS error will occur.

- Subtraction

  ) ( ) ( ) ( ) String comparison operators. Comparison is made on the basis of ASCII codes, a character at a time until a difference is found. If the comparison of two characters is made and both are a space, the comparison is considered smaller. Note that "A" is greater than "a" since trailing spaces are significant.

- Erases current line being typed, and types a carriage return/line feed. An "H" is usually a shift/l.

  (backspace or underline) Erases last character typed, if no more characters are left on the line, types a carriage return/line feed. "H" is usually a shift/l.

CARRIAGE RETURN A carriage return must end every line typed in. Returns print head or CRT cursor to the last position (leftmost) on line. A line feed is always executed after a carriage return.

CONTROL/C Interrupts execution of a program or a list command. CONTROL/C has effect when a statement finishes execution, or in the case of interrupting a LIST command, when a complete line has finished printing. In both cases a return is made to BASIC's command level and OK is typed.

Points "BREAK IN LINE XXX" where XXX is the line number of the next statement to be executed.

: (colon) A colon is used to separate statements on a line. Colons may be used to direct and indirect statements. The only limit on the number of statements per line is the line length, which is not limited to GOTO or GOSUB to the middle of a line.

CONTROL/O Typing a Control/O once causes BASIC to suppress all output until a return is made to command level, an input statement is encountered, another control/O is typed, or an error occurs.

? Question marks are equivalent to PRINT. For instance, ? 2+2 is equivalent to PRINT 2+2. Question marks can also be used in indirect statements. 10 ? X, when listed will be typed as 10 PRINT X.

ERROR MESSAGES
After an error occurs, BASIC returns to level and types OK. Variable values and the program text remain intact, but the program may not be continued and all GOSUB and FOR content is lost.

When an error occurs in a direct statement, no line number is printed.

Format of error messages:
Direct Statement 7XX ERROR
Indirect Statement 7XX ERROR IN YYYY

In both of the above examples, "7XX" will be the error code. The "YYYY" will be the line number where the error occurred for the indirect statement.

The following are the possible error codes and their meanings:
10 Bad Subscript. An attempt was made in reference to an array element which is outside the dimensions of the array. This error can occur if the wrong number of dimensions are used in a matrix reference; for instance, LET A(1,1) = Z when A has been dimensioned DIM A(2,2).
DD Double Dimension. After an array was dimensioned, another dimension statement for the same array was encountered. This error often occurs if an array has been given the default dimension 10 because a statement like A(6,6) = 6 is encountered and then later in the program a DIM AC(10) is found.
FC Function Call error. The parameter passed to a math or string function was out of range. FC errors can occur due to: a) a negative array subscript (LET AC(-1) = 123); b) an out-of-range large array subscript (A(32767)); c) an incorrect character string function (a string function cannot be passed into a math function); d) a call to GET or an illegal character string function (string function that has been patched into GET).
DF Illegal Direct. You cannot use an INPUT or DEFIN statement as a direct command.
NF NEXT without FOR. The variable in a NEXT statement corresponds to no previously executed FOR statement.
OD Out of Data. A READ statement was executed but none of the DATA statements in the program have already been read. The program tried to read too much data or insufficient data was included in the program.
OM Out of memory. Program too large, too many variables, too many FOR loops, too many GOSUB's, too complicated an expression or any combination of the above.
OV Overflow. The result of a calculation was too large to be represented in BASIC's number format. If an underflow occurs, zero is given as the result and execution continues without any error message being printed.
SN Syntax error. Missing parenthesis in an expression, illegal character in a line, incorrect punctuation, etc.
RG RETURN without GOSUB. A RETURN statement was encountered without a previous GOSUB statement being executed.
US Undefined Statement. An attempt was made to GOTO, GOSUB or THEN to a statement which does not exist.
/D Division by Zero.
CN  Continue error. Attempt to continue a program when none exists, an error occurred, or after a new line was typed into the program.
LS  Long String. Attempt was made by use of the concatenation operator to create a string more than 255 characters long.
ST  String Temporaries. A string expression was too complex, break it into two or more shorter ones.
TM  Type mismatch. The left hand side of an assignment statement was a numeric variable and the right hand side was a string, or vice versa; or, a function which expected a string argument was given a numeric one or vice versa.
UF  Undefined Function. Reference was made to a user defined function which had never been defined.

Here is a listing of the error messages of the 9 digit KIM-I BASIC cross-indexed to the error messages of the 6 digit version:

<table>
<thead>
<tr>
<th>KB:6</th>
<th>KB:9</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAD SUBSCRPT</td>
<td>BS</td>
</tr>
<tr>
<td>REORDERED ARRAY</td>
<td>CC</td>
</tr>
<tr>
<td>ILLEGAL QUANTITY</td>
<td>ID</td>
</tr>
<tr>
<td>ILLEGAL DIRECT</td>
<td>NF</td>
</tr>
<tr>
<td>NEXT WITHOUT FOR</td>
<td>OD</td>
</tr>
<tr>
<td>OUT OF DATA</td>
<td>OM</td>
</tr>
<tr>
<td>OUT OF MEMORY</td>
<td>OV</td>
</tr>
<tr>
<td>OVERFLOW</td>
<td>SN</td>
</tr>
<tr>
<td>RETURN WITHOUT GOSUB</td>
<td>RG</td>
</tr>
<tr>
<td>UNDEFINED STATEMENT</td>
<td>US</td>
</tr>
<tr>
<td>DIVISION BY ZERO</td>
<td>UG</td>
</tr>
<tr>
<td>CAN'T CONTINUE</td>
<td>CN</td>
</tr>
<tr>
<td>STRING TOO LONG</td>
<td>LS</td>
</tr>
<tr>
<td>FORMULA TOO COMPLEX</td>
<td>ST</td>
</tr>
<tr>
<td>TYPE MISMATCH</td>
<td>TM</td>
</tr>
<tr>
<td>UNDEFINED FUNCTION</td>
<td>UF</td>
</tr>
</tbody>
</table>

---

**ASCII CHARACTER CODES**

<table>
<thead>
<tr>
<th>DECIMAL</th>
<th>CHAR _</th>
<th>DECIMAL</th>
<th>CHAR _</th>
<th>DECIMAL</th>
<th>CHAR _</th>
<th>DECIMAL</th>
<th>CHAR _</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>NUL</td>
<td>033</td>
<td>!</td>
<td>065</td>
<td>A</td>
<td>097</td>
<td>a</td>
</tr>
<tr>
<td>001</td>
<td>SOH</td>
<td>034</td>
<td>&quot;</td>
<td>066</td>
<td>B</td>
<td>098</td>
<td>b</td>
</tr>
<tr>
<td>002</td>
<td>STX</td>
<td>035</td>
<td>#</td>
<td>067</td>
<td>C</td>
<td>099</td>
<td>c</td>
</tr>
<tr>
<td>003</td>
<td>ETX</td>
<td>036</td>
<td>$</td>
<td>068</td>
<td>D</td>
<td>100</td>
<td>d</td>
</tr>
<tr>
<td>004</td>
<td>EOT</td>
<td>037</td>
<td>%</td>
<td>069</td>
<td>E</td>
<td>101</td>
<td>e</td>
</tr>
<tr>
<td>005</td>
<td>ENQ</td>
<td>038</td>
<td>&amp;</td>
<td>070</td>
<td>F</td>
<td>102</td>
<td>f</td>
</tr>
<tr>
<td>006</td>
<td>ACK</td>
<td>039</td>
<td>(</td>
<td>071</td>
<td>G</td>
<td>103</td>
<td>g</td>
</tr>
<tr>
<td>007</td>
<td>BEL</td>
<td>040</td>
<td>)</td>
<td>072</td>
<td>H</td>
<td>104</td>
<td>h</td>
</tr>
<tr>
<td>008</td>
<td>BS</td>
<td>041</td>
<td>1</td>
<td>073</td>
<td>I</td>
<td>105</td>
<td>i</td>
</tr>
<tr>
<td>009</td>
<td>HT</td>
<td>042</td>
<td>*</td>
<td>074</td>
<td>J</td>
<td>106</td>
<td>j</td>
</tr>
<tr>
<td>010</td>
<td>LF</td>
<td>043</td>
<td>+</td>
<td>075</td>
<td>K</td>
<td>107</td>
<td>k</td>
</tr>
<tr>
<td>011</td>
<td>VT</td>
<td>044</td>
<td>,</td>
<td>076</td>
<td>L</td>
<td>108</td>
<td>l</td>
</tr>
<tr>
<td>012</td>
<td>FF</td>
<td>045</td>
<td>.</td>
<td>077</td>
<td>M</td>
<td>109</td>
<td>m</td>
</tr>
<tr>
<td>013</td>
<td>CR</td>
<td>046</td>
<td>/</td>
<td>078</td>
<td>N</td>
<td>110</td>
<td>n</td>
</tr>
<tr>
<td>014</td>
<td>SO</td>
<td>047</td>
<td>0</td>
<td>079</td>
<td>O</td>
<td>111</td>
<td>o</td>
</tr>
<tr>
<td>015</td>
<td>SI</td>
<td>048</td>
<td>1</td>
<td>080</td>
<td>P</td>
<td>112</td>
<td>p</td>
</tr>
<tr>
<td>016</td>
<td>DLE</td>
<td>049</td>
<td>2</td>
<td>081</td>
<td>Q</td>
<td>113</td>
<td>q</td>
</tr>
<tr>
<td>017</td>
<td>DC1</td>
<td>050</td>
<td>3</td>
<td>082</td>
<td>R</td>
<td>114</td>
<td>r</td>
</tr>
<tr>
<td>018</td>
<td>DC2</td>
<td>051</td>
<td>4</td>
<td>083</td>
<td>S</td>
<td>115</td>
<td>s</td>
</tr>
<tr>
<td>019</td>
<td>DC3</td>
<td>052</td>
<td>5</td>
<td>084</td>
<td>T</td>
<td>116</td>
<td>t</td>
</tr>
<tr>
<td>020</td>
<td>DCA</td>
<td>053</td>
<td>6</td>
<td>085</td>
<td>U</td>
<td>117</td>
<td>u</td>
</tr>
<tr>
<td>021</td>
<td>NAK</td>
<td>054</td>
<td>7</td>
<td>086</td>
<td>V</td>
<td>118</td>
<td>v</td>
</tr>
<tr>
<td>022</td>
<td>SYM</td>
<td>055</td>
<td>8</td>
<td>087</td>
<td>W</td>
<td>119</td>
<td>w</td>
</tr>
<tr>
<td>023</td>
<td>ETB</td>
<td>056</td>
<td>9</td>
<td>088</td>
<td>X</td>
<td>120</td>
<td>x</td>
</tr>
<tr>
<td>024</td>
<td>CAN</td>
<td>057</td>
<td>;</td>
<td>089</td>
<td>Y</td>
<td>121</td>
<td>y</td>
</tr>
<tr>
<td>025</td>
<td>EM</td>
<td>058</td>
<td>:</td>
<td>090</td>
<td>Z</td>
<td>122</td>
<td>z</td>
</tr>
<tr>
<td>026</td>
<td>SUB</td>
<td>059</td>
<td>;</td>
<td>091</td>
<td>[</td>
<td>123</td>
<td>[</td>
</tr>
<tr>
<td>027</td>
<td>ESCAPE</td>
<td>060</td>
<td>)</td>
<td>092</td>
<td>\</td>
<td>124</td>
<td>\</td>
</tr>
<tr>
<td>028</td>
<td>FS</td>
<td>061</td>
<td>=</td>
<td>093</td>
<td>]</td>
<td>125</td>
<td>]</td>
</tr>
<tr>
<td>029</td>
<td>GS</td>
<td>062</td>
<td>)</td>
<td>094</td>
<td>^</td>
<td>126</td>
<td>^</td>
</tr>
<tr>
<td>030</td>
<td>RS</td>
<td>063</td>
<td>?</td>
<td>095</td>
<td>_</td>
<td>127</td>
<td>_</td>
</tr>
<tr>
<td>031</td>
<td>US</td>
<td>064</td>
<td>@</td>
<td>096</td>
<td>DEL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LF = Line Feed**  **FF = Form Feed**  **CR = Carriage Return**  **DEL = Rubout**
USAGE NOTES

INITIALIZATION DIALOG

STARTING BASIC After you have loaded BASIC, it will respond:

MEMORY SIZE? If you type a carriage return to MEMORY SIZE?, BASIC will use all the contiguous memory upwards from location 8192 that it can find. BASIC was not searching when it finds one byte of KIM or nonexistent memory.

If you wish to allocate only part of the memory to BASIC, type the number of bytes of memory you wish to allocate in decimal. This might be done, for instance, if you were using part of the memory for a machine language subroutine. There are 4096 bytes of memory in a 4K system, and 8192 bytes in an 8K system.

BASIC will then ask:

TERMINAL WIDTH? This is to set the output line width for PRINT statements only. Type in the number of characters for the line width for the particular terminal or other output device you are using. This may be any number from 1 to 255, depending on the terminal. If no answer is given (i.e., if a carriage return is input), the line width is set to 72 characters.

Now KIM-1 BASIC will enter a dialog which will allow you to delete some of the intrinsic functions. Deleting these functions will give you more memory space to store your programs and variables. However, you will not be able to call the functions you delete. Attempting to do so will result in an FC error. The only way to restore a function that has been deleted is to reload BASIC.

The following is the dialog which occurs:

WANT SINCOS-TAN-ATN? Answer "Y" to retain all four of the functions, "N" to delete all four, or "A" to delete ATN only.

Now BASIC will type out:

XXXXX BYTES FREE "XXXXX" is the number of bytes available for program, variables, array storage and the stack. It does not include string area.

It will then print out the BASIC version and COPYRIGHT MICROSOFT CO., with the year of the copyright, and finally OK.

You will now be ready to begin using KIM-1 BASIC

USING THE CASSETTE INTERFACE

To save a program on tape prepare the cassette just as though it were a dump command where about to be issued to the KIM-1 monitor. Then type M:TAPE. A tone of XXXX" is issued for all BASIC files, so only one BASIC program can be saved per tape. After completion of the "SAVE-

command and control will be returned to the KIM-1 monitor. Resetter BASIC at the "RELOAD" location specified for your version using the "RELOAD-

command.

To load a program from tape prepare the cassette just as though it were a "LOAD" command where about to be issued to the KIM-1 monitor. Then type "LOAD". Any previously program or variable values will be lost. Control will return to the KIM-1 monitor on completion of the load. If the load is successful "OK" will be displayed and simply typing "G" will return control to BASIC. Otherwise, patch locations 0661 HEX and 0662 HEX to contain their previous values (what they were before the load). HEX for K8-6 and HEX for K8-9, and do a "G" to the load address (BDLOAD) specified for your version of BASIC.

The current version of BASIC does not support loading and saving of data files.

K8.9 INTEGER VARIABLES

Integer variables are allowed in the 9 start version of KIM-1 BASIC. They must be followed by % wherever they are used. Note that an integer variable is distinct from a floating point variable of the same name. Integer arrays are also allowed. Each integer array requires 2 bytes of storage whereas floating point values require 5 bytes. Non-integer values passed to an integer variable will be truncated. Integer variables cannot be used in user defined functions or "IFN" loops. Integer variables should be used to conserve memory space. They do not save time. In fact, they are usually slower to use than floating point values.

Basic/machine Language Interface

There are four steps required to use a machine language routine.

1) Set aside memory. The KIM versions of Microsoft 6592 BASIC and RAM starts at 2000 hex. Contiguous memory above BASIC is used for program storage. The highest location used is determined by the response to the "MEM?" question. If a location value is input that will be the highest location used. Otherwise BASIC will search for the highest contiguous RAM address by storing and reading values from memory.

A machine language routine must not be in a memory area used by BASIC, so it must be

1) Below 2000 hex but above 200 hex or

2) Above the decimal address typed into "memory size" or

3) Non-contiguous with the RAM at 2000 hex.

2) Store the routine into memory. This can be done either before or after BASIC is loaded. The KIM cassette load, an assembler, keying into memory or BASIC'S POKED command may be used.

3) Notify BASIC of the location of the routine. USRLOC which is 2040 hex in all versions of KIM BASIC must be POKED to contain the address of the "USR" machine language routine. 2040 hex, 8246 hex, or whatever must be given low 8 bits of the address and 3737 hex.

4) The machine language routine must be called. The "USR" function is used for this purpose. A single numeric value must be given as the argument to USR. BASIC will dispatch to the address contained in USRLOC. The user routine may modify all of the registers. Any RTS should be performed when the routine completes.

Data can be passed to the machine language routine in two ways:
MORE ON PEEK AND POKE

POKE can be used to set up your machine language routine in high memory. BASIC does not restrict which addresses you can POKE patches which a user wishes to include in his BASIC can also be made using POKE.

MORE ON PEEK AND POKE can be used to store byte-oriented information. When you initialize BASIC, answer the following question with the amount of memory in your KIM-1 minus the amount of memory you wish to use as storage for byte formatted data.

You are now free to use the memory in the top of memory in your KIM-1 as byte storage.

RULES FOR EVALUATING EXPRESSIONS

1) Operations of higher precedence are performed before operations of lower precedence. This means the multiplication and divisions are performed before additions and subtractions. As an example, 2+10/5 equals 4, not 2.4. When operations of equal precedence are found in a formula, the left hand one is executed first: 6-3+5-8, not -2.

2) The order in which operations are performed can always be specified explicitly through the use of parentheses. For instance, to add 5 to 3 and then divide by 4, we would use (5+3)/4, which equals 2. If instead we had used 5+3/4, we would get 5.75 as a result (5 div 3/4).

The precedence of operators used in evaluating expressions is as follows, in order beginning with the highest precedence:

(Note: Operators listed on the same line have the same precedence.)

1) FORMULAS ENCLOSED IN PARENTHESES ARE ALWAYS EVALUATED FIRST: 2) ! (NOT) 3) NEGATION: 4) * / (MULTIPLICATION) 5) + - (ADDITION & SUBTRACTION) 6) > < (RELATIONAL OPERATORS: equal precedence for all six) = = ( ) ( ) ( ) ; 7) NOT: 8) AND, 9) OR

A relational expression can be used as part of any expression.

Relational Operator expressions will always have a value of True (1) or a value of False (0). Therefore, (5=5)=0, (5=5)=1, (4) 5)=0, (4) 5)=1.

The THEN clause of an IF statement is executed whenever the formula after the IF is not equal to 0. That is to say, IF X THEN...is equivalent to IF X 0 THEN...and can be used for manipulation, and for performing boolean operations.

These three operators convert their arguments to sixteen bit, signed tw's complement integers in the range -32768 to 32767. They then perform the specified logical operation on them and return a result within the same range. If the arguments are not in this range, an "FC" error results.

63 AND 16+16 Since 63 equals binary 111111 and 16 equals binary 10000, the result of the AND is binary 10000 or 16.

4 OR 2+6 Binary 100 OR's with binary 10 equals binary 110, or a decimal.

NOT X NOT X is equal to -(X+1). This is because to form the sixteen bit two's complement of the number, you take the bit (one's) complement and add one.

The operations are performed in bitwise fashion, this means that each bit of the result is obtained by examining the bit in the same position for each argument.

A typical use of the bitwise operations is to test bits sets in the KIM-1's two ports which reflect the state of some external device. The position is the most significant bit of a byte, while position 0 is the least significant.

SPACE HINTS

In order to make your program smaller and save space, the following hints may be useful.

1) Use multiple statements per line. There is a small amount of overhead (2 bytes) associated with each line in the program. Two of these five bytes contain the line number of the line in binary. This means that no matter how many digits you have in your line number (minimum line number is 0, maximum is 64000), it saves the same number of bytes. (Putting as many statements as possible on one line will cut down on the number of bytes used by your program).

2) Delete all REM statements and unnecessary spaces from your program.

3) Use variables instead of constants. Suppose you use the constant 3.14159 ten times in your program. If you insert a statement 10 P=3.14159 in the program, and use P instead of 3.14159 each time it is needed, you will save 40 bytes. This will also result in speed improvement.

4) A program need not end with an END; in an END statement at the end of a program may be deleted.

5) Resease the same variables. If you have a variable T which is used to hold a temporary result in one part of the program and you need a temporary variable later in your program, use it again. Or, if you are asking the terminal user to give a YES or NO answer to two different questions at two different times during the execution of the program, use the same temporary variable AS to store the reply.

6) Use GOSUB's to execute sections of program statements that perform identical actions.

7) Use the zero elements of arrays: for instance. A(0) A(0:0)

STORAGE ALLOCATION INFORMATION

Simple (non-array) numeric variable size V = 6 bytes; 2 for the variable name, 4 for the value. Simple non-array string variables also use 6 bytes; 2 for the variable name, 2 for the length, and 2 for a pointer.

Array variables use a minimum of 12 bytes. Two bytes are used for the variable name, two for the size of the array, two for the number of dimensions and two for each dimension alone with four bytes for each of the array elements.
String variables also use one byte of string space for each character in the string. This is true whether the string variable is a simple string variable like A$, or an element of an array such as QA[$].

When a DEF statement is executed, bytes are used to store the definition.

Reserved words such as FOR, GOTO or NEXT, and the names of the intrinsic functions such as COS, INT and SQR take up only one byte of program space. All other characters in programs use one byte of program space each.

When a program is being executed, space is dynamically allocated on the stack as follows:

1. Each active FOR...NEXT loop uses 22 bytes.
2. Each active DO Sub (one that has not returned) uses 6 bytes.
3. Each parenthesis encountered in an expression uses 4 bytes and each temporary result calculated in an expression uses 12 bytes.

SPEED HINTS

The hints below should improve the execution time of your BASIC program. Note that some of these hints are the same as those used to decrease the space used by your programs. This means that in many cases you can increase the efficiency of both the speed and size of your programs at the same time.

1. Delete all unnecessary spaces and REM's from the program. This may cause a small decrease in execution time because BASIC would otherwise have to ignore or skip over spaces and REM statements.
2. This is probably the most important speed hint of all. Use integers instead of constants. It takes more time to convert a constant to its floating point representation than it does to reach the value of a simple or array variable. This is especially important within FOR...NEXT loops or other code that is executed repeatedly.
3. Variables which are encountered first during the execution of a BASIC program are allocated at the start of the variable table. This means that a statement such as A$=B$ or A$=C$ or A$=C$ will place A first, B second, and C third in the symbol table (assuming line 5 is the first statement executed in the program). Later in the program, when BASIC finds a reference to the variable A$, it will search only one entry in the symbol table to find A, two entries to find B and three entries to find C, etc.
4. NEXT statements without the index variable, NEXT is somewhat faster than NEXT I because no check is made to see if the variable specified in the NEXT is the same as the variable in the most recent FOR statement.

CONVERTING BASIC PROGRAMS NOT WRITTEN FOR THE KIM-1

Though implementations of BASIC on different computers are in many ways similar, there are some incompatibilities which you should watch for if you are planning to convert some BASIC programs that were not written for the KIM-1.

1. Array subscripts. Some BASICs use "I" and "J" to denote array subscripts. KIM BASIC uses "I" and "J".

2. Strings. A number of BASICs force you to dimension (declare) the length of strings before you use them. You should remove all dimension statements of this type from the program. In some of these BASICs, a declaration of the form DIM A$(J) declares a string matrix of J elements each of which has a length I. Convert DIM statements of this type to equivalent ones in KIM-1 BASIC. DIM A$(J).

KIM-1 BASIC uses " " for string concatenation of " " or " ".

KIM-1 BASIC uses LEFTS, RIGHTS and MID$ to take substrings of strings. Other BASICs use A$(J) to access the Jth character of the string A$, and A$(J) to take a substring of A$ from character position I to character position J. Convert as follows:

OLD NEW
A$(I) MID$(A$(I),I)
A$(I,J) MID$(A$(I,J+1)

This assumes that the reference to a substring of A$ is in an expression or is on the right side of an assignment. If the reference to A$ is on the left-hand side of an assignment, and $X is the string expression used to replace characters in A$, convert as follows:

OLD NEW
A$(I,J) $X A$='LEFT$(A$(I,J),I)$X$=$MID$(A$(I,J))
A$(I,J) $X A$='LEFT$(A$(I,J),I)$X$=$MID$(A$(I,J))

3. Multiple assignments. Some BASICs allow statements of the form: $000 LET B=C=E. This statement would set the variables B, C, and E to zero.

In KIM-1 BASIC this has an entirely different effect. All the "$=E" to the right of the first one would be interpreted as logical comparison operators. This would set the variable B to 1 if C equaled 0. If C did not equal 0, it would be set to 0. The easiest way to convert statements like this one is to rewrite them as follows:

$000 C=0$B=C

4. Some BASICs use "$=" instead of "=": "=" to delimit multiple statements per line. Change the "=" to "":"=" in the program.

5. Paper tapes punched by other BASICs may have no nulls at the end of each line, instead of the three per line recommended for use with KIM-1 BASIC.

To get around this, try to use the tape feed control on the Teletype to stop the tape from reading as soon as KIM-1 BASIC types a carriage return at the end of the line. Wait a second, and then continue feeding the tape.

When you have finished reading in the paper tape of the program, be sure to punch a new tape in KIM-1 BASIC's format. This will save you from having to repeat this process a second time.

6. Programs which use the MAT functions available in some BASICs will have to be rewritten using FOR...NEXT loops to perform the appropriate operations.

DERIVED FUNCTIONS

The following functions, while not intrinsic to KIM-1 BASIC, can be calculated using the existing BASIC functions.

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>FUNCTION EXPRESSED IN TERMS OF BASIC FUNCTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECANT</td>
<td>sec(x) = 1/cos(x)</td>
</tr>
<tr>
<td>COSECANT</td>
<td>csc(x) = 1/sin(x)</td>
</tr>
<tr>
<td>COTANGENT</td>
<td>cot(x) = 1/tan(x)</td>
</tr>
<tr>
<td>INVERSE SINE</td>
<td>arcsin(x) = atan(x/sqrt(1-x^2))</td>
</tr>
</tbody>
</table>
**Inverse Cosine**

\[
\text{ARCCOS}(x) = -\text{ATN}(x / \text{SQRT}(x^2 + 1)) \times 1.5708
\]

**Inverse Secant**

\[
\text{ARCSEC}(x) = \text{ATN}(\text{SQRT}(x^2 - 1)) + \text{SGN}(x) \times 1.5708
\]

**Inverse Cosecant**

\[
\text{ARCSC}(x) = \text{ATN}(\text{SQRT}(x^2 - 1)) + \text{SGN}(x) \times 1.5708
\]

**Inverse Cotangent**

\[
\text{ARCCOT}(x) = -\text{ATN}(x) \times 1.5708
\]

**Hyperbolic Sine**

\[
\text{SINH}(x) = (\text{EXP}(x) - \text{EXP}(-x)) / 2
\]

**Hyperbolic Cosine**

\[
\text{COSH}(x) = (\text{EXP}(x) + \text{EXP}(-x)) / 2
\]

**Hyperbolic Tangent**

\[
\text{TANH}(x) = \text{EXP}(-x) / (\text{EXP}(x) + \text{EXP}(-x)) \times 1.5708
\]

**Hyperbolic Secant**

\[
\text{SECH}(x) = 2 / (\text{EXP}(x) + \text{EXP}(-x))
\]

**Hyperbolic Cosecant**

\[
\text{CSCH}(x) = 2 / (\text{EXP}(x) - \text{EXP}(-x))
\]

**Inverse Hyperbolic Sine**

\[
\text{ARGSINH}(x) = \text{LOG}(\text{SQRT}(x^2 + 1))
\]

**Inverse Hyperbolic Cosine**

\[
\text{ARCCOSH}(x) = \text{LOG}(x + \text{SQRT}(x^2 - 1))
\]

**Inverse Hyperbolic Tangent**

\[
\text{ARGTANH}(x) = \text{LOG}(1 + x / (1 - x)) / 2
\]

**Inverse Hyperbolic Secant**

\[
\text{ARGSECH}(x) = \text{LOG}(\text{SQRT}(x^2 + 1) + 1) / x
\]

**Inverse Hyperbolic Cosecant**

\[
\text{ARCCH}(x) = \text{LOG}(\text{SQRT}(x^2 + 1) + 1) / x
\]

**Inverse Hyperbolic Cotangent**

\[
\text{ARGCOTH}(x) = \text{LOG}(x + 1) / (x - 1)
\]

**Important Locations (Hex)**

<table>
<thead>
<tr>
<th>Location</th>
<th>KB-6</th>
<th>KB-9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowest Location</td>
<td>2000</td>
<td>2000</td>
</tr>
<tr>
<td>Highest Location</td>
<td>4040</td>
<td>4260</td>
</tr>
<tr>
<td>INIT (start)</td>
<td>1E44</td>
<td>4065</td>
</tr>
<tr>
<td>GLOAD (return from good load)</td>
<td>0000</td>
<td>0009</td>
</tr>
<tr>
<td>BDLOAD (return from bad load—also patch locations 1 &amp; 2)</td>
<td>2454</td>
<td>2523</td>
</tr>
<tr>
<td>RETSAV (return from save)</td>
<td>2690</td>
<td>275F</td>
</tr>
<tr>
<td>USRLOC (for user address)</td>
<td>2040</td>
<td>2040</td>
</tr>
<tr>
<td>AYINT (pass A,Y to machine code USR routine)</td>
<td>——</td>
<td>2FC2</td>
</tr>
<tr>
<td>GIVAYF (return A,Y from machine code USR routine)</td>
<td>——</td>
<td>3195</td>
</tr>
<tr>
<td>ISLNC (check for control/C)</td>
<td>260B</td>
<td>26DA</td>
</tr>
<tr>
<td>Location of call to KIM-1 input routine</td>
<td>2387</td>
<td>2456</td>
</tr>
<tr>
<td>Location of call to KIM-1 output routine</td>
<td>296C</td>
<td>2A51</td>
</tr>
</tbody>
</table>