HP-41 ASSEMBLER AND BARCODE GENERATOR: TYPES 1 & 2 BARCODE

FOR THE IBM-PC™

Comments and suggestions concerning this package may be sent to:

Tony Malburg
1201 N. Alvernon #26
Tucson, AZ 85712

TABLE OF CONTENTS

INTRODUCTION

The HP-41 Assembler and Barcode generator were written for the users of Hewlett Packard Series 41 programmable handheld calculators who own the HP 82153A optical wand and would like to translate their own programs into barcode. The routines run on an IBM-PC[tm] or compatible personal computer using MS-DOS[tm]. Also, access to a dot matrix printer is necessary to print the barcode. At the current time, three types of dot matrix printers are supported. They are EPSON, OLYMPIA, and STAR MICRONICS. Here it should be noted that any EPSON compatible should work fine. Refer to notes on printer types in chapter 6 for more information.

This manual contains a summary of command line entries needed to process the barcode on the PC and a description of the syntax required by the assembler. This manual is NOT intended as a tutorial for using the HP-41 or the IBM-PC[tm] and if questions arise, the proper reference should be consulted. Some HP-41 references are given in chapter 7 and should questions arise about the IBM-PC[tm] or compatible, the appropriate DOS manual should be consulted.

1.0 ASM41

ASM41 is the HP-41 Assembler. It reads the source file and generates the data file containing the hex machine codes.

1.1 COMMAND FORMAT

ASM41 <filename>

<filename>

The file named should contain the source file to be read. The source file must have the extension of .HPB but entry of this extension is optional at the command line. If it is not entered it is assumed to be .HPB. If the file could not be found an error message will be displayed.

The following is an example of a command line entry to process a program called DI. This program was taken from a book by W.C. Wickes (see reference section) and will be referred to throughout this manual. When run the program will activate every segment in the HP-41 display.

In addition to the command line entry, the banner information is shown along with the completion message. (Note the filename extension is omitted but is assumed to be .HPB and the default drive is A.)

Example:

```
A> ASM41 DI
HP-41 Assembler      Version 1.00      (c)Tony Malburg, 1986
No errors found.        Output file is in:  DI.DAT
```

The completion message tells there were no errors found and the output data file was written to DI.DAT. The next step is to process the data file and print the barcode using BCODE.

2.0 BCODE

BCODE is the barcode printer routine. It reads the data file generated by ASM41 and converts the hex machine codes to rows of barcode.

2.1 COMMAND FORMAT

BCODE <filename> [options]

NOTE: Minimum of one space is required between command line entries.

<filename>

The file named should contain the data file to be converted and printed. The data file will have the extension of .DAT as the default since that is the extension name ASM41 used when the output file was generated. As with ASM41, the extension is optional and is assumed to be .DAT. If the file is not found, an error message will be displayed.

Following are the explanations of the options available to BCODE. Note that if no options are entered, the default parameters will be a non-private encoding using bit image graphics (EPSON method) to produce the barcode. The maximum number of options allowed is two, on any command line entry, with one of the two being the private option. If two printer options are entered by accident, only the last one will be recognized by BCODE.

[options] parameters (enter in any order)

-p or -P private encoding of barcode (can't view the steps once the program is loaded into HP-41)

-o or -O print barcode on an OLYMPIA NP printer

-s or -S print barcode on a STAR MICRONICS printer

Following is an example of the command line entry to complete the processing on the program DI. The optional switches are included for a private encoding of the barcode using the OLYMPIA NP format to show the proper syntax for the switch entry. (Note the filename extension is omitted but is assumed to be .DAT and the default drive is A.)

```
A> BCODE DI -p -o
HP-41 Barcode Printer     Ver 1.10      (c)Tony Malburg, 1986
Enter Program Name:  DI  (display tester - private)
```

After the banner is displayed the program prompts for the program name. This provides the user a chance to enter the name and very brief description of the program which will be printed at the top of each page of barcode. The above entry is about as long as any entry should be but experimentation and personal taste will dictate the length used. No completion message will be displayed, however, since the completion will be evident when the printer stops printing the barcode and a form feed is generated.

3.0 INSTRUCTION FORMAT

The following is a general description of the syntax that ASM41 needs to decode the program instructions properly. Care was taken in laying out the format to allow any standard text editor to generate the source code. Before forging ahead into the instruction set, some definitions, explanations, and basic guidelines are provided.

Generally speaking, instructions fall into two categories: (1) direct entry, and (2) synthetic. Direct entry instructions are ones that can be keyed in directly from the HP-41 keyboard. They are by far the most commonly used instructions. Synthetic instructions, on the other hand, can not be entered directly and have to be "created" by some "synthetic" means. Hence the term synthetic programming.

This assembler not only permits the use of direct entry instructions but the full range of synthetic instructions as well. Although this is not meant to be a course in synthetic programming, many references to this type of programming are made. An excellent book on synthetic programming is listed in the reference section of this manual. It is suggested that this book be read for a broader understanding of synthetic programming.

Another subject worth discussing is memory. Anyone who has used the HP-41 extensively will know that memory is one resource that is somewhat scarce even with the now available Extended Memory modules. Since each machine register can only hold seven bytes of program machine code, it is important to make use of short forms of instructions whenever possible to optimize memory use. This assembler is optimizing and will use the smallest possible number of bytes to generate the machine code for instructions that are variable in their byte requirements. If optimizing is critical, pay close attention to the notes and comments made throughout this section. Byte counts are given for various instructions to provide the user a method of determining what type of instructions to use or avoid when optimizing.

Throughout this section of the manual, discussion of the various instructions leads to the use of a specific range of valid numeric or character entries. Whenever the brackets are used with more than one character between them, the characters enclosed in the brackets make up the desired range. For example [A-Z] means include all upper case letters A through Z (inclusive). If a single character is used between the brackets, it is merely to offset the character from the rest of the text.

Two final comments before starting the instruction set. First of all, the ASSEMBLER IS CASE DEPENDENT and requires all instructions to be entered in UPPER CASE LETTERS. There are a few exceptions to this limitation, namely in alpha labels and alpha text where the lower case letters [a-e] are valid. Secondly, when entering the instructions in the source (text) file, there must be at least one white space

between the mnemonic and the argument. When in doubt about a specific instruction, follow the examples given for the correct form.

3.1 COMMENT LINES AND USE OF WHITE SPACE IN SOURCE FILE

The use of comments can be very helpful in understanding how a program is suppose to work. While over commenting is not recommended, a few choice comments scattered throughout the program will allow a programmer to review the code at a later date and be able to understand it. While some say that "uncommented code means job security", most programmers will agree that writing the code once was hard enough and would rather not have to rewrite it to understand what was written the first time. Putting comments in the code can help eliminate such problems.

The format for comments in this implementation is similar to many other assemblers and that is all comments must be preceded by a semicolon [;]. They can appear on separate lines or follow a properly terminated instruction. (Properly terminated instructions will be discussed in section 3.11.)

Examples:

```
;This is a comment line!  (note the semicolon)
     ;This is also a comment  (note the space in front)
RCL  02   ;Recall register 2
"THIS IS TEXT"   ;This is an alpha string
```

The second example above leads into the discussion of white space. Basically, white space is defined as those spaces found between items on a line. The two most commonly used white spaces are the tab and the space (from the space bar). The method used in parsing each line in the source file nulls out any white space before any decoding begins. Since this is the case, tabs and spaces may appear anywhere in the code, except in global text labels, with no adverse effects. If tabs are used in text strings they will appear as the starburst character in the HP-41 display since the tab produces the hex byte 09 and there is no equivalent character for this code on the HP-41.

Also it should be noted that, while it is not often utilized, blank lines are permitted in the source file. A blank line is defined as a line with nothing more than a carriage return entered.

3.2 LABELS

Labels provide a method of program control or a means of executing a program or part of a program directly from the keyboard. There are two types of labels used on the HP-41: (1) local labels, and (2) global labels. At the end of this section, it should be obvious when and where to use each and the proper syntax required by ASM41.

3.2.1 LOCAL LABELS

Local labels are those that are only accessible while the instruction pointer is inside the program containing them. In other words they can not be accessed by another program. They can be numeric with a valid range of [00-99] or they can be single letter alpha with valid letters being [A-J] and [a-e]. Local alpha labels are very useful in simulating menu driven programs. While the instruction pointer is inside the program containing these local alphas and the machine is in the user mode, the top two rows of keys execute the labels [A-J] and the shift of the top row of keys execute the labels [a-e].

Examples:

```
LBL  01     ;this is a local label
LBL  J      ;and so is this but this is alpha
LBL  d
```

Note the absence of any quotes on the local alpha labels. As will be seen in the section on global labels, a quote in front of the letter signifies that the label is global and will be stored as such occupying 5 bytes of program memory instead of only 2. If the desired result is a local alpha label (i.e. - a menu driven program), DO NOT put a quote in front of the letter. Also note that any numeric label less than 10 must have the leading 0 present. This will be true for all instructions that can have an argument less than 10 (an argument being the second part of the instruction). If the leading 0 is not present, an assembly error will be generated.

Byte Counts:

```
LBL  00 thru 14                ;one byte each
LBL  15 thru 99                ;two bytes each
LBL  A thru J and a thru e     ;two bytes each
```

3.2.2 GLOBAL LABELS

Global labels are those that can be executed by any program at any time by using the appropriate XEQ or GTO instruction, hence the term global. All user global labels appear in the user program catalog CAT 1 whereas local labels do not.

This assembler allows two methods of generating global labels: (1) straight text and (2) synthetics. Regardless of the method used, there are some common guidelines to follow if ASM41 is to function properly.

1. Maximum of 7 characters allowed in a label (quotes are not included in this count)

2. The first character must be a double quote ["] (the ending quote is optional)

3. No white spaces are allowed in the name itself (if a space character is desired - use synthetics)

4. The label itself can not contain a quotation mark (if quotes are desired - use synthetics)

Byte Counts:

All global labels require 4 bytes of overhead plus one byte for each character in the label. To optimize, use single letter alphas outside the range of the local alphas or double letter alpha labels.

3.2.2.1 GLOBAL TEXT LABELS

Global text labels are the ones most commonly used by programmers as most of them contain letters and symbols that can be entered right on the HP-41 keyboard.

Examples:

```
LBL  "LABEL          ;optional second quote omitted
LBL  "KRdec"         ;note the lower case letters allowed
                     ;are between [a-e]
LBL  "abcde12"       ;numbers are allowed
LBL  "@#$%^%"        ;so are various punctuation
LBL  "TSTLBL
```

In the above example using punctuation, the [@] character and the [#] character were used. These two characters can not be generated directly on the HP-41 alpha keyboard but since they are in the standard ASCII character set, the HP-41 can display them. Rather than force the use of synthetics to generate characters like these, all of the standard ASCII printable characters have been included in this assembler. The only exception to this is limiting the lower

case alphas to the range of [a-e]. The sigma character used by the HP-41 is the tilde [~] character in the standard ASCII character set. It, too, is included in the list of valid label characters.

One character that can be generated on the HP-41 that can not be generated on a personal computer is the 'not equal to' sign. As will be seen in the section on other instructions, section 3.9, the two characters [!] and [=] are used together to represent 'not equal to'. If this character is desired in a global label, use synthetics.

3.2.2.2 GLOBAL SYNTHETIC LABELS

On occasion, a character in a label may be desired that does not have a printable ASCII equivalent. An example might be the full man character generated by the byte Ø1 or the 'not equal to' sign described above. Although these labels can not be executed directly from the keyboard without going through CAT 1, they can be paired with an XEQ or GTO of the same label within a program or in another program. That's not to say this is all that useful in day to day programming since most needs are filled with global text labels. It is interesting, however, and this capability was included for completeness.

The syntax for a synthetic string of any kind (a preview of synthetic text strings) is a double quote ["] followed by a single quote [']. As with the global text labels, the ending quote is optional. The synthetic string is entered as hex bytes and each character requires two digits. As a reminder, hex numbers range from [Ø-9,A-F]. As mentioned earlier this assembler is CASE dependent and this is one instance where this is very true. All hex digits greater than 9 must be upper case.

Examples:

```
LBL  "'5453544C424C'"     ;equivalent to LBL "TSTLBL"
LBL  "'01050406           ;note omission of second quote
LBL  "'574859204D453F     ;generates LBL "WHY ME?
                          ; including the space (20H)
LBL  "'0C0D'"             ;note upper case C and D
LBL  "'1D'"               ;generates 'not equal to' sign
```

Note that in first synthetic label above, there are more than 7 characters. The reason being that each label character requires 2 hex digits to represent it. With this in mind it is permissible to have up to 14 hex digits in a string to make up a global label. The only restriction is that the count must be even or an assembly error will occur. The quotes are not included in this count.

3.3 XEQ AND GTO

Due to the similarities in the way the XEQ and GTO instructions are generated, this discussion will incorporate both. There is a difference in the way they are executed but that is beyond the scope of this manual.

As with the labels, XEQ and GTO can be either local or global in nature. Local XEQ and GTO instructions only function if the program that they are in has a LBL with the appropriate numeric or alpha value. Global XEQ and GTO instructions perform a jump of sorts to the label found in CAT 1 or CAT 2 if the GTO or XEQ was not generated for the specific XROM routine. (XROM codes are discussed in section 3.7.)

3.3.1 LOCAL XEQ OR GTO

Local XEQ's or GTO's can either be direct or indirect. As with local labels, the valid range for direct XEQ or GTO instructions is [00-99] for numeric and [A-J,a-e] for alpha. Therefore, rather than repeat the syntax rules for local labels, if there is a question about a particular direct XEQ or GTO instruction, refer to section 3.2.1.

Indirect XEQ's or GTO's are a bit different. It is still valid to go indirectly through [00-99] provided enough memory has been allocated for data registers. Any attempt to XEQ or GTO indirectly through [A-J] will result in an error. This is because these registers do not exist. Registers [a-e] do exist even though Hewlett Packard does not provide information into their existence. They are called status registers and are used by the machine to monitor the system (i.e. - flags, return addresses for subroutines, key assignment, etc.). The only way these registers can be accessed is through synthetic programming techniques or if an assembler, such as this one, allows for their decoding. In addition to [a-e], registers [M-R] also exist and can be XEQ'd or GTO'd as can the normal stack registers X, Y, Z, T, and L. (Actually register R is not displayed on the HP-41 as register R but as the alpha append character. The letter R was chosen because personal computers can not generate the append character and because R follows Q in the alphabet and register Q <u>is</u> an actual register.)

Examples:

```
XEQ  a          ;execute local label a
XEQ  IND  a     ;execute indirectly through register a
XEQ  03         ;execute local label 3
GTO  34         ;go to local label 34
GTO  J          ;go to local label J
GTO  IND  R     ;go indirectly through register R
GTO  IND  00    ;go indirectly through register 00
GTO  IND  O     ;go indirectly through register O
```

Note that as with any instruction that can have an argument less than 10, the leading 0 must be included. As an example of this, look at the above instruction XEQ 03. If the leading 0 is not included, an assembly error will be generated.

Byte Counts:

```
GTO  00 thru 14      ;two bytes each
All other direct GTO's and XEQ's  ;three bytes each
All indirect GTO's and XEQ's      ;two bytes each
```

3.3.2 GLOBAL XEQ OR GTO

Since global XEQ's and GTO's work in conjunction with a global label somewhere else in the machine, they follow the same syntax rules as global labels, both text and synthetics. Without further explanation some examples follow.

Examples:

```
XEQ  "TSTLBL              ;no second quote needed
GTO  "'574859204D453F'"   ;go to LBL "WHY ME?"
XEQ  "12245"              ;execute LBL "12245
```

Byte Counts:

All global XEQ and GTO instructions are two bytes plus one byte per character in the label name.

3.4 REGISTER ACCESS INSTRUCTIONS

The register access instructions supported by the HP-41 are listed below. These instructions, with the exception of SIGREG, can all access numeric as well as stack/status registers in both direct and indirect formats.

RCL STO ST+ ST- ST* ST/ X<>

ISG DSE VIEW SIGREG ASTO ARCL

Note that SIGREG is the sigma register. This instruction is used to determine the starting register for the six statistical registers. As such it can not be used in direct stack/status format. However, it can be used in both direct and indirect numeric and in indirect stack/status formats.

The set of valid registers are the same as those used in indirect XEQ or GTO instructions, namely [00-99], [M-R], [a-e], X, Y, Z, T, and L.

Examples:

```
RCL  35          ;recall register 35
ST*  X           ;store times register X
ARCL 02          ;alpha recall register 2
SIGREG 25        ;set first statistical register to
                 ; be in register 25
DSE  M           ;decrement and skip if equal to M
ISG  IND  M      ;increment and skip if greater
                 ; indirectly through register M
X<>  d           ;exchange register X with register d
ST/  IND  P      ;store indirectly through register P
```

Note that as with any instruction that can have an argument less than 10,the leading 0 must be included. As an illustration of this, look at the ARCL 02 instruction above. If the leading 0 is left off, an assembly error will be generated.

Byte Counts:

```
RCL  00 thru 15   ;one byte each
STO  00 thru 15   ;one byte each
All other register access instructions both,
direct and indirect, are two bytes each.
```

3.5 FLAG INSTRUCTIONS

Flag instructions fall into two categories: (1) flag setting, and (2) flag testing. The instruction syntax for each type is listed below. Both types can be indirect and operate through the usual [00-99], [M-R], [a-e], X, Y, Z, T, and L registers. The assembler will generate the correct machine code for these instructions but the user must insure the data contained in these indirect locations is within the range of the number of flags available for setting and/or testing. These ranges are [00-29] for setting and [00-55] for testing.

In the direct mode, only numeric instructions are permitted. For direct flag instructions, the ranges are the same as those listed above, namely [00-29], for flag setting instructions and [00-55] for flag testing instructions. As with other instructions that can have an argument less than 10, the leading 0 must be included or an error will be generated by the assembler.

Flag setting instruction syntax:

```
SF        CF        FS?C      FC?C
```

Flag testing instruction syntax:

```
FS?       FC?
```

Examples:

```
SF  09          ;this one sets flag 9
CF  29          ;this one clears flag 29
FS? 55          ;this one only tests
FC?C 29         ;note this instruction tests AND clears
FC?C IND 99     ;same as above only indirectly thru 99
FS?C 13         ;this one also tests AND clears
SF  IND 10      ;sets flag indirectly thru 10
```

Byte Counts:

All flag instructions, both direct and indirect, are two bytes.

3.6 DISPLAY FORMAT INSTRUCTIONS

There are three display format instructions available on the HP-41. They are FIX, SCI, and ENG. They control how many digits follow the decimal point and how exponents are to be displayed. They can be entered in direct or indirect format. In the indirect format they follow the same syntax rules given for flag instructions. That is to say they can access [00-99], [M-R], [a-e], X, Y, Z, T, and L indirectly. In the direct format the valid range is [00-09] and as stated several times before the leading 0 must be present or an assembly error will be generated.

Examples:

```
FIX  02
ENG  09
FIX  IND  75
SCI  04
SCI  IND  a
```

Byte Counts:

All display format instructions, both direct and indirect, are two bytes.

3.7 XROM INSTRUCTIONS

The XROM instructions incorporate all those functions that are present in ROM application modules. Each module has a specific number assigned to it and each routine within the module is numbered, starting with the number one being assigned to the first routine. As an example, the MATH PAC is numbered module 01. The program MATRIX is the first routine contained in this module as reflected by a CAT 2 listing of the MATH PAC routines. This means the correct XROM code would be XROM 01,01.

There are a few simple steps in finding out what the correct XROM code will be for any routine in any application pac that is available to be plugged in. (If the pac is not available, a listing from the users manual may provide this information.)

1. With HP-41 OFF, plug the module into an open port.

2. Turn HP-41 ON and switch to program mode.
   Get to the .END. by entering [SHIFT][GTO . .]

3. Enter [XEQ][ALPHA] 'program name' [ALPHA].
   If the program name was a valid application pac program name, one of the following will appear

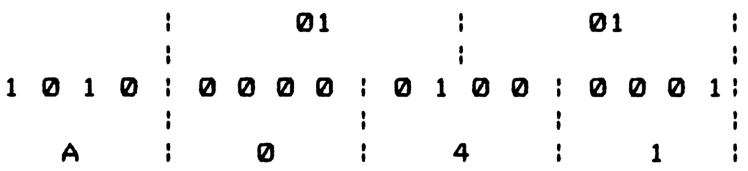
   01 XROM 'program name' or 01 'program name'

4. Turn HP-41 OFF and pull module out of the port.

5. Turn HP-41 ON and switch to program mode.
   The display will now contain the correct XROM code. As in the example above

   01 XROM 01,01

   For the curious, this equates to hex A0 41 in machine code as follows:



With this XROM code, a program can be encoded to run the routine in the application pac without using a long XEQ or GTO instruction. The correct syntax is shown below. Note the comma shown in the HP-41 display is not entered. Entering the comma will result in an assembly error.

Examples:

```
XROM  01 01   ;note space between numbers and no comma
XROM  26 28   ;code for TIME in time module
```

The maximum number allowed for the module number is 31 and the maximum number allowed for the routine number is 63.

Byte Counts:

All XROM instruction require two bytes.

3.8 TONES

The tone capabilities on the HP-41 are somewhat limited by direct entry programming. A user is limited to 10 different tones, all of the same duration. Through synthetic programming, however, 128 different tones can be generated. Actually only 6 new tone frequencies are generated but each frequency can have a variety of tone durations ranging from 0.023 seconds to 5.00 seconds. To allow for all 128 different tones with two digit entry, the correct entry syntax will be in hex. The direct range of TONE's will be [00H-7FH] and the indirect range of TONE's will be [80H-FFH]. One word of caution concerning the indirect tones. If the indirect location does not contain a number in the range from [0-9], the HP-41 will display 'DATA ERROR' when an attempt is made to execute the instruction.

Examples:

```
TONE  25      ;generates a display of TONE 7 with
              ; a duration of 0.023 seconds at a
              ; frequency of 105 Hz
TONE  1A      ;generates a display of TONE 6 with
              ; a duration of 5.00 seconds at a
              ; frequency of 394 Hz
```

Refer to W.C. Wickes for a complete listing of the tones and their durations.

Byte Counts:

All TONE instructions, both
direct and indirect, require two bytes.

3.9 OTHER INSTRUCTIONS

The category of other instructions encompasses all of the single byte functions listed below, shown in their correct syntactical form.

+	-	*	/	X<Y?
X>Y?	X<=Y?	SIG+	SIG-	HMS+
HMS-	MOD	%	%CH	P-R
R-P	LN	X^2	SQRT	Y^X
CHS	E^X	LOG	10^X	E^X-1
SIN	COS	TAN	ASIN	ACOS
ATAN	DEC	1/X	ABS	FACT
X!=0?	X>0?	LN1+X	X<0?	X=0?
INT	FRC	D-R	R-D	HMS
HR	RND	OCT	CLSIG	X<>Y
PI	CLST	R^	RDN	LASTX
CLX	X=Y?	X!=Y?	SIGN	X<=0?
MEAN	SDEV	AVIEW	CLD	DEG
RAD	GRAD	ENTER^	STOP	RTN
BEEP	CLA	ASHF	PSE	CLRG
AOFF	AON	OFF	PROMPT	ADV

As mentioned earlier, the 'not equal to' sign can not be generated on personal computers using most standard text editors. As a way around this problem the two characters [!] and [=] used together are taken to mean the same thing. Therefore in the above list, the != means 'not equal to'. The carat [^] is used to represent the up arrow on the HP-41. This means, for example, that X^2 represents 'X squared' and R^ represents 'Roll the stack up' on the HP-41. Another anomaly is in the sigma register notation. As seen in the section on register accesses, the correct syntax for the sigma register is SIGREG. In a similar fashion, sigma plus and sigma minus are SIG+ and SIG- respectively.

Examples are not necessary because these are all stand alone instructions and have only one format.

Byte Counts:

As mentioned above, all of these instructions are one byte each.

3.10 DATA

Numeric data entry falls into one of three categories: (1) integers, (2) decimal fractions, and (3) exponentials. There are only a couple of simple rules to follow when entering numeric data.

1. No white space is allowed within a data string between numbers or exponents.

2. Entry length is limited to 10 numeric digits plus the associated minus signs, decimal point, or the E needed for an exponential.

Valid Ranges for each data type:

Integers:	+/- 9999999999
Decimals:	+/- .0000000001
Exponentials:	E-99 to 9.9999999E99

Examples:

```
E20           ;leading 1 can be omitted to save a byte
-E-30         ;again the 1 is omitted
-.36923E75    ;negative decimal exp. (leading 0 omitted)
36.256E-55
1782489564
```

Byte Counts:

In general, the number of bytes required by numeric data equals the number of individual characters in the string. The only bytes that can be saved in numeric data is in the elimination of the leading 1 in an exponential expression where the multiplier is 1 as shown in the first two examples above. In addition to this, a byte can be saved by not entering the leading 0's in a decimal fraction entry. (i.e. - instead of entering 0.32, enter .32, etc.)

3.11 ALPHA STRINGS

As with other instructions that incorporated some type of alpha string, the entry of full line alpha strings can either be in direct text or synthetics. Regardless of the type used, there are a few guidelines that apply.

1. The first character of the string must be a double qoute ["].

2. The terminating quote is only necessary if a comment is to be added to the same line. The reason for this is, of course, that a semicolon is a valid text character and unless the text string is delimited from it in some way, the assembler assumes it is part of the string.

In the case where this comment adds enough characters to the length to exceed the limit, an assembly error will be generated.

3. The limit for any alpha string is 15 characters including the append character if one is present.

Byte Counts:

The byte count for all alpha strings, whether they are text alpha or synthetic alpha, is as follows. One byte overhead plus one byte per character in the string up to a total of 15. Quotes, single or double, are not included in the count.

These are just a few of the ground rules for alpha strings. Each entry type has its own peculiarities which are discussed below.

3.11.1 TEXT ALPHA STRINGS

This type of alpha string consists mainly of characters that can be entered directly into the HP-41 via the alpha keyboard. As with global text labels, some characters that can not be entered on the HP-41 are still allowed in this type of alpha string since these characters are in the ASCII set of printable characters and as such the HP-41 can display them. There are, however, several characters that can not be used in this type of string. The characters which the last comment was referring to were mainly the lower case letters [f-z]. These characters may be generated synthetically but the HP-41 will display them as starbursts since they do not generate an assigned character code.

The following tables list both the valid and invalid characters for use in text strings. The list for valid characters is missing two characters, the space character and the DEL character, neither of which show up when printed. However, the DEL character produces the alpha append character on the HP-41 and if the text editor being used permits this character, it is one way of producing appending alpha strings. If the text editor being used will not permit the use of the DEL character, the lower case [o] has been reserved as the append character. As one last comment about the valid characters, the tilde [~] character is used to produce the sigma character on the HP-41.

Valid Characters	Invalid Characters
! " # $ % & ' ( ) * + , -	f g h i j k l m n p q r
. / 0 1 2 3 4 5 6 7 8 9 :	s t u v w x y z { ¦ }
; < = > ? @ A B C D E F G	
H I J K L M N O P Q R S T	
U V W X Y Z [\ ] ^ _ ` a	
b c d e ~ o	

Examples:

```
"THIS IS TEXT"     ;note the closing quote since a
                   ;comment follows the text string
"oTHIS APPENDS"    ;note the lower case o used as
                   ;the append character
"abcdeABCDE"       ;only a-e are permitted in lower case
```

3.11.2 SYNTHETIC ALPHA STRINGS

Synthetic alpha strings include all possible characters in the range from [00-FF] since they are created from hex bytes. As with other instructions that contained synthetic strings, the proper syntax for the synthetic alpha strings is a double quote ["] followed by a single quote ['']. The ending quote is optional but, as with text alpha strings, must be present if a comment is to follow the string. The valid range of characters is [0-9,A-F]. Note that all entries greater than 9 must be in upper case. Also note that as with global synthetic labels, each character requires two bytes to produce it. Since a valid alpha string can contain a maximum of 15 characters, this means that a synthetic alpha string can contain 30 characters, not counting the single or double quotes. The only restriction is that the count must be even or an assembly error will be generated.

Examples: (duplicates of the text alpha string examples)

```
"'5448495320495320544558 54'"    ;"THIS IS TEXT"
"'7F5448495320415050454E4453'"   ;"oTHIS APPENDS"
"'6162636465414243444 5'"        ;"abcdeABCDE"
```

3.12 .END.

Last but certainly not least is the .END. instruction. As with the permanent .END. in the program memory, this instruction will put an end onto the last row of barcode so the HP-41 will know when the last row has been scanned. When this code is scanned by the Wand, the HP-41 will display 'WORKING' while it is compiling the scanned data.

Since the instruction is the .END., it should be located at the end of the program. The syntax should be obvious. If not, refer to chapter 5 for program examples.

4.0 ERROR MESSAGES

During the assembly process, should ASM41 find any errors in the instruction syntax, an error message will be printed on the screen following the instruction that was at fault. An error does not terminate the assembly process, however. The assembler continues checking the rest of the source file to determine if there are any more errors. If any errors were found during the assembly process, the total number of errors will be printed along with the message saying that no output file was generated. The general format of these messages and a listing of the different error messages available to ASM41 is shown below.

When an error occurs, the following will be printed:

<instruction>
Syntax Error in line <line number> - <error type>

The instruction is the line of code that could not be assembled for one reason or another. The line number is a count of the number of carriage returns that have been encountered at the time the error occurred. This count includes blank lines and comment lines since each of these lines is terminated with a carriage return. The count starts at 1 with the first line in the source file. The error type will be listed if the assembler started to decode the instruction and found discrepancies in the syntax.

Error Messages (error type):

invalid local label

invalid indirect XEQ

invalid local alpha XEQ

invalid local XEQ

invalid indirect GTO

invalid local GTO

invalid local alpha GTO

label too long!

invalid synthetic label!

invalid global label!

invalid GTO label!

invalid XEQ label!

invalid synthetic characters in string!

invalid characters in alpha string!

text string too long!

synthetic string too long!

synthetic string has odd count!

exponent not valid!

exponent too big!

number is not valid!

too many digits in number!

need more that just a '.'!

need more than just a '-.'!

invalid XROM instruction!

invalid stack/status access!

invalid indirect stack/status access!

There are two error messages that act as a catch all for general instructions. These messages will have the following form:

invalid <instruction>

invalid indirect <instruction>

If the assembler has no idea where to start decoding the instruction, the error type will be:

can't decode instruction!

The error messages listed above are pretty much self explanatory or should be by now. If, for some reason, an instruction does not function as you feel it should, let me know. I've tried to debug this program for all possible cases but after reading this manual, I'm sure you'll agree that with the number of instructions and the variations of each, it would be a monumental task to check everything.

5.0 PROGRAM EXAMPLES

Following will be two sample programs. The first will be the program mentioned at the outset of this manual, DI, and the second will be another program from W.C. Wickes book, DC. Comments are added for clarity.

PROGRAM #1

```
;Program  DI  (display tester)
;  tests HP-41 display segments while
;  demonstrating mass flag control via
;  register d.
;
;  to execute this program simply enter
;  [XEQ][ALPHA] DI [ALPHA]
;
     LBL  "DI
     "'F80000100021E8'"   ;mass flag control number
     RCL  M
     "'803A803A803A'"     ;three sets of starburst/colons
     ASTO Y               ;store this alpha string in Y
     ARCL Y               ;recall Y three times to fill
     ARCL Y               ; the alpha display with the
     ARCL Y               ; starburst characters
     AVIEW                ;view the alpha register
     X<>  d               ;perform the mass flag control
     .END.                ;end of program
```

PROGRAM #2

```
;Program DC  (Decimal to Character Conversion)
;
;   takes a decimal integer from the X register
;   and converts it into the equivalent HP-41
;   display character
;
;   To use:
;       enter number in X register
;       enter [XEQ][ALPHA] DC [ALPHA]
;
;   the character will be displayed in the X register
;
     LBL  "DC
     OCT            ;convert entry to octal from decimal
     E3             ; to be used to set flags
     /              ; in register d
     10
     +
     X<>  d         ;set the flags according to entry
     FS?C 19        ;do a series of flag checks
     SF   20        ;and sets
```

```
FS?C 18
SF   19
FS?C 17
SF   18
FS?C 15
SF   17
FS?C 14
SF   16
X<>  d          ;restore original flags
STO  M          ;store character information in M
"'7F0000'"      ;append two nulls
CLX             ;clear register X
STO  N          ;store register X in register N
"oA"            ;append the letter A
X<>  N          ;swap register X and register N
CLA             ;clear alpha register
X<>  M          ;swap register X and register M
AVIEW           ;view the alpha register
.END.           ;end of program
```

6.0 PRINTER TYPES

This section will discuss the printer types permitted for use with the program BCODE in generating the barcode. Information regarding the method used in producing the barcode on each type of printer will be included to permit the user to determine if their printer will work properly.

6.1 EPSON PRINTERS

As was seen in the options section of BCODE, the default parameters used include the EPSON double density bit image graphics mode to print the barcode. In terms of escape sequences, this means:

```
ESC 'L' n1 n2
```

If the printer you are using supports this print mode continue reading this section for other codes used to insure compatibility. If not, chances are your printer will not work with this program.

Other codes used from the EPSON family of printers include:

```
ESC 'l' n      ;sets left margin to column n

ESC 'J' n      ;one time line feed of n/216 inches

ESC '@'        ;master reset

ESC 'G'        ;select double strike mode
```

Of course the carriage return <CR>, line feed <LF>, and horizontal tab <HT> codes of 13, 10, and 9 respectively are also used but are generally standard on all printers.

6.2 OLYMPIA PRINTERS

The program BCODE was originally developed for use with an OLYMPIA model NP printer. The method used to generate the barcode includes the use of download characters and the high density proportional print mode. If the OLYMPIA option is selected at the command line, it is critical that DIP SWITCH 1-4 (i.e.- switch bank one, switch number 4) is turned OFF to allow the buffer area to be used as the area for the download characters. If this switch is turned ON and the OLYMPIA option is selected, the barcode will, with the exception of the row identifier strings and page heading, consist of only 0's, 1's, 2's, and 3's.

The control codes used by the OLYMPIA NP printer in this mode include:

```
ESC '&' 0 n m a p1...p11  ;define download chars.

ESC '%' 0 0   ;select internal character set

ESC '%' 1 0   ;select download character set

ESC 'J' n     ;one time line feed of n/216 inches

ESC 'l' n     ;set left margin to column n

ESC 'n'       ;select high density proportional print

ESC '@'       ;master reset
```

A rather unique situation exists with the OLYMPIA NP printer and that is it is EPSON compatible as well. If desired, the EPSON mode can be used with fair results. If the EPSON mode is used, DIP SWITCH 1-4 can be turned ON to facilitate a speedier printing since the inline buffer will then be utilized. However, for a higher resolution barcode, the OLYMPIA option should be used. It is unknown if other OLYMPIA printers will work as the NP, using download characters, but they are most likely EPSON compatible.

6.3 STAR MICRONICS PRINTERS

The two STAR MICRONICS printers that should work properly with the -S option are the RADIX and GEMINI series. Like the EPSON, these printers support the double density bit image graphics mode but other codes make these printers different enough to warrant the use of a separate option.

The control codes used by these printers include the following:

```
ESC 'L' n1 n2  ;select double density bit image

ESC 'M' n      ;set left margin to column n

ESC 'J' n      ;one time line feed of n/144 inches

ESC '@'        ;master reset

ESC 'G'        ;select double strike mode
```

Others models will most likely work provided they use the same escape sequences shown above.

If the printer you are using does not correspond to one of the above descriptions and you would like to see your printer included in future updates, please let me know. In order to include it I will need, at the very least, a listing of the available escape sequences that the printer uses. I would like to eventually include a laser printer among those supported in order to produce the best possible barcode. Unfortunately, at the present time, the cost of a laser printer is somewhat prohibitive to the average PC owner.

7.0 REFERENCES

Wickes, W.C. SYNTHETIC PROGRAMMING ON THE HP-41C
Corvallis, OR.: Larken Publications, 1982.

OWNER'S HANDBOOK AND PROGRAMMING GUIDE HP-41C/41CV
Hewlett Packard Company, 1980.

Journal staff. "HP BARCODES - HOW ARE THEY CODED?"
PPC Calculator Journal, VOL. 7, NO. 5, ppg 27-33,
JUNE 1980.

Journal staff. "KEYING HP-41 SYNTHETIC INSTRUCTIONS"
PPC Calculator Journal, VOL. 8, NO. 6, ppg 79-80
AUG - DEC 1981.

NOTES ON BARCODE QUALITY

The quality of the barcode will depend on the type of printer used and the age of the ribbon. The sharpness of the barcode will depend on the pins on the dot matrix print head. The flatter the pin head, the sharper the barcode. It should be noted that on newly printed barcode, the paper will exhibit a slightly rippled effect due to the pressure applied by these pins. This rippled effect decreases with time and has no effect on the quality of copies that may be made.

Although the sharpness is beyond user control, short of buying a new printer, the darkness of the barcode isn't. It depends only on the age of the ribbon. While a newer ribbon produces a darker barcode, it also has a greater tendency for smearing since the total area of ribbon ink per page is so great. A ribbon that is slightly used seems to give the best results.

It is suggested that the original barcode be xeroxed before being scanned. The reason for this is because the printer ribbon image is sometimes difficult for the wand to pick up due to its reflectivity. Today's dry toner copiers make this image much more readable. Also, the life of a ribbon used to print barcode can be extended since a xerox copy of a weak original can be made darker and thus produce a good quality barcode. As always, to protect the barcode from excessive wear, use the clear protective sheeting provided with the wand.

APPENDIX

On the next two pages are samples of barcode generated from the two program examples in chapter 5. Again, these routines can be found in W.C. Wickes book listed in the reference section of this manual. Note the second line tells how many registers are needed to load the program into the HP-41. This figure is not exact as it is computed from the number of rows of barcode and not the actual total byte count. The number listed will always be greater than or equal to the actual number of registers needed.

PROGRAM NAME: DI (Display Tester) PAGE 1 OF 1
PROGRAM REGISTERS NEEDED: 6

ROW 1 (1 - 2)

ROW 2 (2 - 6)

ROW 3 (6 - 11)

PROGRAM NAME: DC (Decimal to Character) PAGE 1 OF 1
PROGRAM REGISTERS NEEDED: 10

ROW 1 (1 - 5)

ROW 2 (5 - 12)

ROW 3 (12 - 18)

ROW 4 (19 - 24)

ROW 5 (24 - 28)

NOTES