HP-41 BARCODE GENERATORS:
TYPES 4,6,7 & 8
PLUS CUSTOM BARCODE

FOR THE IBM-PC™

Comments and suggestions concerning this package may be sent to:

Tony Malburg
~~1201 N. Alvernon #26~~ 2315 S. Grand Ave #1
~~Tucson, AZ 85712~~ San Pedro, CA 90731

TABLE OF CONTENTS

TABLE OF CONTENTS (cont'd)

INTRODUCTION

The routines contained in this package were written as a supplement to the HP-41 Assembler and Barcode Generator to permit full utlilization of the HP82153A optical wand. The routines run on an IBM-PC^{tm} or compatible personal computer using MS-DOS^{tm}. In addition to this, 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 7 for more information.

This manual contains a summary of the command line entries needed to process each type of barcode on the PC and a description of the syntax required by each. In addition to this, the formats of each type of barcode are given to permit experimentation using a CUSTOM barcode routine, along with an automated checksum computer. 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 8 and should questions arise about the IBM-PC^{tm} or compatible, the appropriate DOS manual should be consulted.

1. Barcode Types

Before a meaningful discussion of the specific types of barcode available for use with the HP82153A optical wand, an overview of the different types and their functions is in order.

As it turns out, there are seven different types of barcode that can be read by the HP82153A. These are listed below in Table 1.1 along with a short description of each.

Table 1.1

Type No.	Description
1	Program - Non Private
2	Program - Private
4	Direct Execution
5	Wand Paper Keyboard
6	Numeric Data
7	Alpha Replace Data
8	Alpha Append Data

The first two types, type 1 & 2, are rather involved in their construction and syntax since they incorporate most all available HP-41 instructions and usually consist of several rows. For this reason, I've written a separate software package to accomplish their printing and, with the exception of a discussion of their format in the chapter on CUSTOM barcode, no further comments will be made about them.

Since type 5 barcode is available on the wand paper keyboard that came with the wand, there is no need to generate this type. As with types 1 & 2, the only discussion on this type will be found in the chapter on CUSTOM barcode.

Out of the seven in the list, this leaves types 4, 6, 7, and 8. In the following chapters, the routines required to generate these types, using an IBM-PC or compatible and an EPSON or compatible dot matrix printer, and the necessary syntax for each will be discussed. Also it should be noted that, with the exception of chapter 5, each chapter was written to stand on its own. That is to say, all of the information needed to use a given routine, except the custom barcode routine, can be found its respective chapter. The chapter on custom barcode generation, chapter 5, however, makes references to entries in the appendix.

2. TYPE4

TYPE4 is the type 4 barcode generator routine. From Table 1.1 in chapter 1, type 4 barcode is defined as direct executing barcode. That is to say that when a type 4 barcode is scanned, a function is carried out or executed. Due to the number of instructions that can be produced and executed with this type of barcode, this is by far the most complex of the routines included with this package.

2.1 Command Format

TYPE4 [printer option]

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

Following is a list of the available printer options. If no option is entered, the printing parameters will default to the EPSON mode of bit image graphics. It should be noted that only one option is allowed and should two or more be entered, only the first one will be recognized by TYPE4.

[printer options]

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

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

An example of a typical command line entry and the initial banner display is shown below. The default drive is A and the printer option used is -s for the STAR MICRONICS printers.

```
A> TYPE4 -s
HP-41 Type 4 Barcode Generator        (C)Tony Malburg, 1986

Enter INSTRUCTION (CR to end):
```

After the banner is displayed, the program prompts for an instruction to be converted to type 4 barcode. When the last instruction has been entered, simply hit the carriage return (CR) and the printer will produce a form feed for the current sheet of paper. If no form feed is desired, a control-C (^C) will exit the program without one.

The following section will discuss the set of instructions that are available and provide examples to show the syntax for proper entry.

2.2 Instructions and Syntax

The following is a description of the syntax that TYPE4 needs to decode the instruction properly. Care was taken in laying out the format to allow the full range of instructions to be entered from the keyboard of any PC. Before getting started a few comments, definitions, and guidelines will be provided.

The first point of interest is the use of synthetic instructions in type 4 barcode. Due to the way in which the wand firmware handles this type of barcode, synthetic instructions, in general, are NOT allowed. There are some instances, such as synthetic global label XEQ or GTO's, where the use of some synthetic characters are permitted. If such synthetic characters are allowed, examples of their use will be given.

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 from A to Z. 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 set of valid type 4 instructions. First of all, the TYPE4 routine is CASE DEPENDENT and requires all instructions to be entered in UPPER CASE LETTERS. There are a few exceptions to this limitation, namely in local and global XEQ and GTO instructions where the lower case letters [a-e] are valid. Secondly, when entering the instructions at the prompt, there must be at least one white space, either a tab or space character from the space bar, between the mnemonic and the argument. When in doubt about a specific instruction, follow the examples given for the correct form.

2.2.1 SIZE

The SIZE instruction provides a means to set the number of data registers with a single pass of the wand. The most obvious use would be to include it with the program barcode to set the size to the required value before running the program.

The syntax for the SIZE instruction is the same as it is on the HP-41 and that is:

SIZE nnn

where nnn is a three digit number corresponding to the number of data registers desired. The range of valid sizes is [000-320]. A few examples will follow. (The comments that follow the instruction examples should not be entered

at the prompt unless it is desired that they be printed also. They are merely added here as an explanation of the syntax.)

Examples:

```
SIZE 010  ;set size to 10 data registers - note that 3
          ; digits are required by TYPE4
SIZE 050  ;set size to 50 data register
SIZE 320  ;set size to 320 data registers - note that with
          ; this setting there are no program registers left
```

2.2.2 ASN

The ASN instruction provides a means of assigning a resident function, an existing user program, or an application pac program to a key with one pass of the wand. When in the user mode, this key will execute the assigned function or program.

The syntax for the ASN instruction is the same as it is on the HP-41 and that is:

ASN [program/function] [keycode]

When an ASN instruction is entered on the HP-41, all the user has to do is enter the [program/function] and the [keycode] is found by pressing the desired key. With TYPE4, the keycode must be known to encode it in the barcode. To find the keycode, follow these steps:

1. Enter [SHIFT][ASN][ALPHA][ALPHA]

2. Hit the desired key and watch the display for the corresponding keycode. (If you hold the key down for about a second, 'NULL' will appear and the key will not be reassigned to itself.)

As an example of this, complete the above steps using the TAN key. The keycode that is displayed is 25. This corresponds to row 2 and column 5 of the keyboard. If a shifted key is to be assigned, a negative sign must be entered before the keycode. The proper keycode for the shifted TAN key would be -25 for example.

There are no limitations when using the ASN function with TYPE4. All keys, including the SHIFT key, can be reassigned. Before reassigning the SHIFT key, careful consideration should be given to the specific application program being run. That is, if the program requires data to be entered or manipulated, the reassignment of the SHIFT key could be very awkward while in the user mode.

Figure 2.1 below shows the value and locations of the valid keycodes. Notice there are two values within each key on the keyboard template, one being the negative of the other. The value on top, the negative value, is the keycode for the shifted key and the one on the bottom is the keycode for the unshifted key.

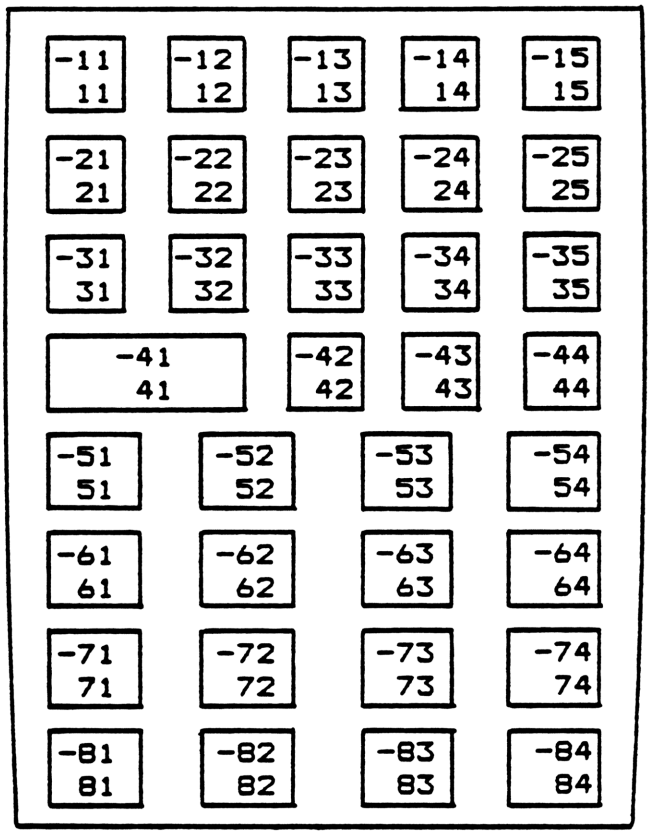


Figure 2.1 Valid HP-41 Keycode Values

Following are some examples of ASN instructions. Note that there must be at least one space separating the fields for TYPE4 to work properly. It should be mentioned here that if an attempt to assign a function or program that doesn't exist in memory somewhere, the word NONEXISTENT will be displayed and the assignment attempt will be aborted. (The comments that follow the instruction examples do not have to be entered at the prompt. They are merely added as an explanation of the syntax.)

Examples:

```
ASN MATRIX 15       ;Assigns MATH PAC routine MATRIX to
                    ; the LN key on the HP-41
ASN 15              ;Reassigns LN key to itself
ASN 10 -54          ;Assigns global LBL "10" to shift 9 key
                    ; Note that this is GLOBAL 10 not LOCAL 10
ASN a 31            ;Assigns global LBL "a" to SHIFT key
                    ; Note that this is GLOBAL a, not LOCAL a
ASN -12             ;Reassigns Y^X to itself
ASN FACT 23         ;Assigns factorial to SIN key
```

The above examples give rise to the discussion of global versus local labels. The ASN function is reserved for use with globals only and not locals. The reason is simple. The HP-41 already provides for automatic local alpha assignments to the top two rows of keys provided nothing else is assigned to the key already. Since this is the case, it would be redundant to assign these labels to a key. It should be pointed out here that for a local alpha key assignment to be active, the instruction pointer must be within the program containing the local alpha label and the calculator set to USER mode. Also note that this feature is limited to use with local alpha labels and NOT local numeric labels.

Since it is possible to have programs with synthetic labels in them, the ability to assign these labels to a key should be available to the user. With TYPE4, this ability exists and is accomplished by entering the ASCII representation of the label (in HEX) between single quotes. The only requirement is the synthetic string must contain an even count of no more than 14 HEX digits. The reason for this is each pair of HEX digits corresponds to one character in the program or function name and since a maximum of 7 characters per name is allowed, only 14 HEX digits are permitted. Note that the count must be even and for all HEX digits greater than 9, UPPER CASE letters must be used. A few examples of this follow.

Examples:

```
ASN '01' -25        ;Assigns the 'full man' label to shift TAN
ASN '222326' 81     ;Assigns LBL "#& to the division key
ASN '5B5D' -71      ;Assigns LBL [] to X>Y? key
```

Through the use of synthetics, virtually any ASN can be generated. If TYPE4 will not generate the proper ASN for the application, the CUSTOM barcode generator described in chapter 5 should be used to complete the job.

2.2.3 XEQ AND GTO

Due to the similarities in the way the XEQ and GTO instructions are generated, this section of the manual will incorporate the discussion of both. There is a difference in the way the HP-41 executes them but that topic is beyond the scope of this manual.

XEQ and GTO instructions can either be local or global in nature. Local XEQ and GTO instructions only function if the instruction pointer is inside the program containing the local label that is to be XEQ'd or GTO'd. Global XEQ and GTO instructions, on the other hand, perform a jump to the label found in CAT 1 or CAT 2 and begin execution there. Following will be a discussion of each type.

2.2.3.1 LOCAL XEQ OR GTO

Local XEQ or GTO instructions can either be direct or indirect. The valid range for direct XEQ or GTO is [00-99] for numeric and [A-J,a-e] for alpha. The proper syntax for direct local XEQ or GTO's is as follows:

1. For all local numeric XEQ or GTO instructions, a two digit number must follow the XEQ or GTO. If the number is less than 10, a leading 0 must be present or TYPE4 will generate an error.

2. For all local alpha XEQ or GTO instructions, a single alpha letter with no quotes must follow the XEQ or GTO. (Quotes indicate global label.)

Some examples are show below for clarity.

Examples:

```
XEQ 01     ;XEQ local numeric label 01 - note the leading 0
GTO a      ;GTO local alpha label a - note absence of quotes
XEQ J      ;XEQ local alpha label J
XEQ 99     ;XEQ local numeric label 99
GTO 45     ;GTO local numeric label 45
```

For indirect local XEQ or GTO's, only numeric and stack registers are allowed. As mentioned at the outset, synthetic instructions are generally not allowed and this is one of many cases that this is true. Since this is the case, the valid numeric range for indirect XEQ or GTO instructions is [00-99], provided enough memory has been allocated for data registers using the SIZE function, and the stack registers X, Y, Z, T, and L. A few examples follow.

Examples:

```
XEQ IND 00    ;XEQ indirectly through register 00
GTO IND 25    ;GTO indirectly through register 25
GTO IND X     ;GTO indirectly through stack register X
XEQ IND T     ;XEQ indirectly through stack register T
```

2.2.3.2 GLOBAL XEQ OR GTO

Global XEQ and GTO's work in conjunction with a global label somewhere else in the machine. As such they can either contain straight text or synthetic characters. This gives rise to the distinction between global text XEQ or GTO's and global synthetic XEQ or GTO's. Regardless of the type used, there are some common guidelines to follow if TYPE4 is to function properly:

1. Maximum of 7 characters allowed in the XEQ or GTO (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 name itself cannot contain a quotation mark (if quotes are desired - use synthetics)

2.2.3.2.1 GLOBAL TEXT XEQ OR GTO

Without further explanation, the following examples will be given to add some clarity to the rules.

Examples:

```
XEQ "LABEL        ;optional second quote omitted
GTO "TSTLBL"      ;second quote present
GTO "abcde12"     ;note the use of numbers
                  ; and lower case letters between [a-e]
XEQ "@#$%^%"      ;various punctuation are also allowed
```

In the last example above, 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 for use with TYPE4.

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 ASCII character set. It, too, is included in the list of valid XEQ or GTO 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 2.2.9, the two characters [!] and [=] are used together to represent 'not equal to'. If this character is desired in a global XEQ or GTO, use synthetics.

2.2.3.2.2 GLOBAL SYNTHETIC XEQ OR GTO

On occasion, a global label may exist that contains a character that is not in the printable ASCII character set. An example of this might be the 'full man' character generated by byte 01 or the 'not equal to' sign described above. When this is the case, an XEQ or GTO instruction made up of the same characters is desired to execute the label. This can be accomplished by using a synthetic XEQ or GTO instruction.

The syntax for a synthetic string of any kind is a double quote ["] followed by a single quote ['']. As with global text XEQ or GTO's, the ending quote is optional. As described in the section on the ASN instruction, synthetic strings are entered as HEX digits with each character in the desired XEQ or GTO requiring 2 digits. Since there is a maximum of 7 characters allowed in any global label, a maximum of 14 HEX digits are allowed in any synthetic XEQ or GTO instruction and the digit count must be even.

As a reminder, HEX numbers range from [0-9,A-F]. Since the routine is CASE dependent, all HEX digits greater than 9 must be entered in UPPER CASE. Some examples follow.

Examples:

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

2.2.4 REGISTER ACCESS INSTRUCTIONS

The register access instructions supported by the HP-41 are listed on the following page. These instructions, with the exception of SIGREG, can all access numeric as well as stack 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 format. However, it can be used in both direct and indirect numeric and in indirect stack format.

As with other instructions discussed so far, no synthetic instructions are permitted. This just means that the use of the status registers is not allowed. The valid range of registers then will be the same as those used in indirect XEQ or GTO instructions, namely [00-99], X, Y, Z, T, and L. Some examples follow.

Examples:

```
RCL 35          ;recall register 35
ST* L           ;store times stack register L
ARCL 02         ;alpha recall register 2
ISG IND 00      ;increment and skip if greater
                ; indirectly through register 00
SIGREG 25       ;set first statistical register to 25
X<> 10          ;exchange register X with register 10
ST/ IND X       ;store divide indirectly through stack reg. X
```

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, TYPE4 will generate an error message.

2.2.5 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], X, Y, Z, T, and L. No synthetics are allowed with flag instructions. In the indirect format the user must insure the data contained in the indirect location 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 TYPE4 will generate an error message.

Flag setting instruction syntax:

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

Flag testing instruction syntax:

```
FS?         FC?
```

Examples:

```
SF   09        ;sets flag 9 - note leading 0
CF   29        ;clears flag 29
FS?  55        ;tests flag 55
FC?C 29        ;tests flag 29 and branches on result
               ; clears flag 29 if set
FS?C IND 99    ;tests flag indirectly through register 99
               ; clears the flag if set
SF IND 10      ;sets flag indirectly through register 10
```

2.2.6 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 above for the flag instructions. That is to say they can access [00-99], X, Y, Z, T, and L indirectly. In the direct format, the valid range is [00-09] and as stated many other times in this manual, the leading 0 must be present or TYPE4 will generate an error message.

Without further explanation, some examples will follow.

Examples:

```
FIX  02        ;fix display to 2 decimal places
ENG  09        ;display to 9 decimals in engineering format
FIX  IND 75    ;fix indirectly through register 75
SCI  04        ;scientific notation to 4 decimal places
SCI  IND X     ;scientific notation indirectly through X
```

2.2.7 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 SIMEQ is the second routine contained in this module as reflected by a CAT 2

listing of the MATH PAC routines. This means the corresponding XROM code would be XROM 01,02.

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 the 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

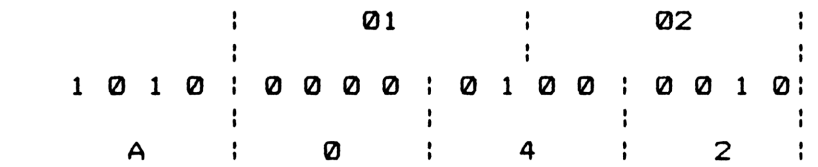
   01 XROM 'program name' or 01 'program name'

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 above example

   01 XROM 01,02

   For the curious, this equates to HEX A0 42 in machine code as follows:



With this XROM code, a program can be run in the application pac without using a long XEQ or GTO instruction. Since type 4 barcode is being used and byte savings are not critical, it really makes no difference which type is used. This option is available, however, if the need should arise.

The correct syntax for the XROM instruction is shown on the next page in a couple of examples. Note the comma shown in the HP-41 display is not entered. Entering the comma will cause TYPE4 to generate an error message.

Examples:

```
XROM  01 02     ;note space between numbers and no comma
XROM  26 28     ;code for TIME in the time module
```

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

2.2.8 TONES

Due to the fact that synthetics are not allowed in type 4 barcode, the range of tones is limited to those entered at the keyboard, namely [00-09]. These tones can be accessed either directly using a range of [00-09] or indirectly through the same registers used in flag and register access instructions, namely [00-99], X, Y, Z, T, and L. If the indirect location does not contain a number in the range from [00-09], the HP-41 will display 'DATA ERROR' when an attempt is made to execute the instruction.

Examples:

```
TONE 09       ;tone 9 - note leading 0 must be present
TONE IND X    ;tone indirectly through register X
TONE 00       ;tone 0
TONE IND 00   ;tone indirectly through register 0
```

As with other type 4 instructions, synthetic tones are not permitted. Any attempt to use synthetics will result in an error message.

2.2.9 OTHER INSTRUCTIONS

The category of other instructions encompasses all of the functions listed below in Table 2.1, shown in their correct syntactical form.

Table 2.1

+	-	*	/	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

Table 2.1 (cont'd)

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. 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 interesting item is 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, sigma minus, and clear sigma are SIG+, SIG-, and CLSIG respectively.

It should be noted at this point that the instructions in the above list also appear as type 5 barcode on the wand paper keyboard that comes with the wand. The type 4 version will execute just as well as the type 5 will but the bar is longer due to the way the two types are constructed. For more information on barcode construction read chapter 5.

3. TYPE6

TYPE6 is the type 6 barcode generator routine. From Table 1.1 in chapter 1, type 6 barcode is defined as numeric data barcode. In other words, they are used to enter numeric data into the calculator.

3.1 Command Format

TYPE6 [printer option]

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

Following is a list of the available printer options. If no option is entered, the printing parameters will default to the EPSON mode of bit image graphics. It should be noted that only one option is allowed and should two or more be entered, only the first one will be recognized by TYPE6.

[printer options]

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

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

An example of a typical command line entry and the initial banner display is shown below. The default drive is A and the printer option used is -o for the OLYMPIA NP printer.

```
A> TYPE6 -o
HP-41 Type 6 Barcode Generator        (C)Tony Malburg, 1986

Enter NUMERIC DATA (CR to end):
```

After the banner is displayed, the program prompts for an numeric data to be converted to type 6 barcode. When the last instruction has been entered, simply hit the carriage return (CR) and the printer will produce a form feed for the current sheet of paper. If no form feed is desired, a control-C (^C) will exit the program without one.

The remainder of this chapter will discuss the format of type 6 barcode and provide examples to show the syntax for proper entry.

3.2 Type 6 Syntax

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           ;the leading 1 can be omitted
-E-30         ; as shown is both of these cases
-.36923E75    ;negative decimal exp. (leading 0 omitted)
36.256E-55
1782489564
```

As you can see above, type 6 barcode is quite simplistic in nature. For information on how type 6 barcode is constructed, read chapter 5.

4. TYPE7 & TYPE8

TYPE7 and TYPE8 are the type 7 and type 8 barcode generator routines respectively. From Table 1.1 in chapter 1, type 7 barcode is defined as alpha replace barcode while type 8 barcode is defined as alpha append barcode. When they are scanned, type 7 barcode will replace the contents of the alpha register and type 8 barcode will append the alpha register with the new alpha data. Since the only difference in the construction of the barcode is the type number, they will both be discussed together.

4.1 Command Format

TYPE7 [printer option] or TYPE8 [printer option]

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

Following is a list of the available printer options. If no option is entered, the printing parameters will default to the EPSON mode of bit image graphics. It should be noted that only one option is allowed and should two or more be entered, only the first one will be recognized by TYPE7 or TYPE8.

[printer options]

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

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

An example of a typical command line entry and the initial banner display is shown below for the TYPE7 routine. (To use TYPE8, simply replace the 7 in the program name with 8.) The default drive is A and the since no printer option is used the default parameter will be in effect. That is to say that the barcode will be printed using EPSON bit image graphics.

```
A> TYPE7
HP-41 Type 7 Barcode Generator        (C)Tony Malburg, 1986

Enter ALPHA REPLACE STRING (CR to end):
```

After the banner is displayed, the program prompts for an alpha replace string to be converted to type 7 barcode. (Had TYPE8 been used instead, the prompt would have been for an alpha append string.) When the last instruction has been entered, simply hit the carriage return (CR) and the printer will produce a form feed for the current sheet of paper. If no form feed is desired, a control-C (^C) will exit the program without one.

The remainder of this chapter will discuss the format of type 7 and 8 barcode and provide examples to show the syntax for proper entry.

4.2 Type 7 & 8 Syntax

Since both type 7 and 8 barcodes are alpha data, the correct syntax needed by TYPE7 or TYPE8 can be straight text or synthetics. Regardless of the entry method used, there are a two basic guidelines that apply.

1. The limit for any alpha string in barcode form is 14 characters. This limitation is due to the way the barcode is constructed with 16 bytes per row being the limiting factor. The count can consist of either 14 text characters or 28 synthetic HEX digits.

2. No comments may be entered after the text. The added characters will either generate an error stating the string is too long or that there are invalid synthetic characters in the string.

The two 'rules' above are very general. Since each method of entry has its own peculiarities, there will be a separate discussion on each method below.

4.2.1 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 XEQ and GTO instructions, 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. These characters are mainly the lower case alpha letters [f-z]. These characters may be generated using the proper HEX digits in a synthetic alpha string but the HP-41 will display them as starbursts since they do not generate an assigned character code.

The following table lists both the valid and invalid characters for use in text strings. With the exception of the tilde [~] character, the characters in the list of valid characters produce its own image. That is to say that [<] produces the less than sign, ["] produces the double quote, etc. The tilde character, however, produces the sigma character on the HP-41.

Table 4.1

Valid Characters	Invalid Characters
! " # $ % & ' ( ) * + , -	f g h i j k l m n o p q
. / Ø 1 2 3 4 5 6 7 8 9 :	r 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 ~ (space)	

To enter a text string, simply type the desired characters. No double quotes are necessary since this type of barcode is nothing but text. Some examples follow. Note that the comments next to the strings are for explanation only and, recalling 'rule 2' above, would generate an error message.

Examples:

```
THIS IS TEXT        ;note that no quotes are needed and spaces
                    ; are permitted within the text string
!@#$%^&*()"'        ;punctuation characters are permitted
abcdeØ12345         ;so are lower case alpha [a-e] and digits
```

When the barcode is printed out, the string will appear inside double quotes to enable the user to see at a glance what type of barcode they are scanning. The type 8 barcode also includes a 'plus' in parenthesis (+) at the beginning to signify that the string is appending and not replacing existing alpha data.

4.2.2 Synthetic Alpha Strings

Synthetic alpha strings include all possible characters in the range from [1Ø-EF] HEX and are created using HEX digits instead of text characters. The first and last sixteen characters in the range from [ØØ-FF] are not allowed as they will cause the calculator to 'lock' up. In other words HEX bytes in the ranges [ØØ-ØF] and [FØ-FF] are not allowed.

As mentioned in 'rule 1' above, the maximum number of HEX digits allowed in a synthetic alpha string is 28. This, if you recall, corresponds to the maximum of 14 characters allowed in type 7 or 8 barcode. Note that the count must be even or an error message will be displayed.

To enter a synthetic alpha string, begin the string with a single quote ['] and follow it with the desired combination of HEX digits. The single quote at the end of the string is optional.

The following examples are the synthetic string counterparts of the above text string examples. As with those examples, the comments are added for explanatory purposes only.

Examples:

```
'5448495320495320544558 54'    ;same as THIS IS TEXT - note
                               ; use of single quotes
'2140232425 5E262A28292227     ;same as !@#$%^&*()"' - note
                               ; last single quote omitted
'6162636465303132333435'       ;same as abcde012345
```

When the barcode is printed out, the synthetic string will appear inside single quotes and the whole thing will be inside double quotes to enable the user to see at a glance what type of barcode they are scanning. The type 8 barcode also includes a 'plus' in parenthesis (+) at the beginning to signify that the string is appending and not replacing existing alpha data. For information on how these two types are constructed, read chapter 5.

5. CUSTOM

CUSTOM is a custom barcode generator routine. This routine is unique in that any type of barcode can be generated. There is no automatic checksum or type number added to the barcode, only the directional bars and the hex bytes that were entered.

5.1 Command Format

CUSTOM [printer option]

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

Following is a list of the available printer options. If no option is entered, the printing parameters will default to the EPSON mode of bit image graphics. It should be noted that only one option is allowed and should two or more be entered, only the first one will be recognized by CUSTOM.

[printer options]

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

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

An example of a typical command line entry and the initial banner display is shown below. The default drive is A and the printer option used is -o for the OLYMPIA NP printer.

```
A> CUSTOM -o
HP-41 Custom Barcode Generator        (C)Tony Malburg, 1986

Enter HEX BYTES (CR to end):
```

After the banner is displayed, the program prompts for an entry of hex bytes. These hex bytes will be converted into barcode bytes and printed according to the printer option entered. When the last instruction has been entered, simply hit the carriage return (CR) and the printer will produce a form feed for the current sheet of paper. If no form feed is desired, a control-C (^C) will exit the program without one.

The remainder of this chapter will discuss the format of the different types of barcode and how to generate each using CUSTOM.

5.2 Type 1 & 2 Barcode Format

As pointed out in chapter 1, type 1 & 2 barcodes are not covered in this manual. For completeness of this package, however, the format of the bars will be discussed so anyone with knowledge of the HP-41 instruction set could conceivably generate their own program barcode - that is, if they are willing to do a lot of tedious work. For those who are not familiar with the HP-41 instruction set but would like to be, see the appendix for a list of the best sources of information. One word of caution before proceeding. At the very minimum, it is assumed the reader knows basic computer terminology. Bytes, nybbles, bits, binary, and hex are just a few of the terms that will be used.

The best way to explain type 1 & 2 barcode is to start with a sample bar. The bar found in figure 5.1 below was generated from the example program found in section 5.2.4.

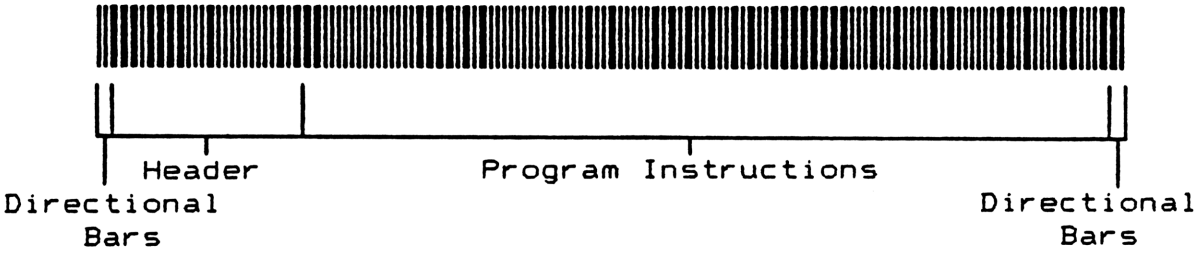


Figure 5.1 Type 1 Barcode Sample

In looking at the bar, note that it is made up entirely of skinny and fat vertical bars. The wand reads these images as binary data bits as follows: a skinny bar is read as a Ø bit and the fat bar is read as a 1 bit. With that information squared away, we are now ready to discuss the construction of the barcode in some detail.

From figure 5.1, we can see that each row of type 1 & 2 barcode is made up of three basic sections. They are the directional bars, the header, and the program instructions. Each one of these sections is described below.

5.2.1 Directional Bars

The first point of interest is the directional bars on the ends of the barcode. They simply tell the wand which way you are scanning the row so it can load its buffers appropriately. When using CUSTOM to generate any type of barcode, the directional bars are automatically added to the beginning and end of every row so unless you try to draw barcode by hand, these bars are of no concern.

5.2.2 Header

In addition to the directional bars, each row of type 1 & 2 barcode has a three byte header. These three bytes tell the wand five critical pieces of information. It is these bytes that make type 1 & 2 barcode so tedious to construct. A description of the header bytes can be found in Table 5.1 below and a detailed explanation of each will follow.

Table 5.1

Header Byte #1	- Running Checksum	8 bits
Header Byte #2	- Type Number: 1 (non private) 2 (private)	4 bits
	- Local Sequence Number Mod 16 of Row number	4 bits
Header Byte #3	- Number of leading bytes of function split between two rows of barcode	4 bits
	- Number of trailing bytes of function split between two rows of barcode	4 bits
	TOTAL	3 bytes

5.2.2.1 Running Checksum

The running checksum can best be described as the wrap around carry sum of all the bytes in the row of barcode plus the existing checksum. As such it must be determined last, after all of the other bytes in the row are known. If this checksum does not match the running checksum computed by the wand firmware, a checksum error will result. To add to the complexity, the checksum byte of row 4, for example, is the running checksum of rows 1, 2, and 3 plus the total of the bytes in row 4. The way this works will be explained in an example after the rest of the sections of the sample bar in figure 5.1 are explained.

5.2.2.2 Type Number

The type number for type 1 & 2 barcode is represented in 4 bits as follows:

$1 - 0001_2$
$2 - 0010_2$

This number must be encoded in the barcode so the wand knows whether or not the barcode is private or non private.

Private barcode simply means that after the program has been loaded, the instruction steps can not be viewed. Instead, the HP-41 will display 'PRIVATE' when switched to program mode.

5.2.2.3 Local Sequence Number

The local sequence number is modulo 16 of the row number starting with 0000_2 for the first row of barcode. This number is used by the wand as a method of keeping you honest in scanning the rows in order. The number ranges from 0000_2 for row 1 to 1111_2 for row 16 and then back to 0000_2 for row 17, etc.

5.2.2.4 Number of Leading Bytes

The number of leading bytes is the count of the bytes that have overflowed from the previous row into the current row. The wand needs to know this information when it compiles the scanned data so it combines the bytes correctly. If it was to simply combine the bytes without knowing this, the result would be nothing but garbage in program memory. This will be clarified later with an example.

5.2.2.5 Number of Trailing Bytes

The number of trailing bytes is the count of the bytes left on the current row that must be combined with the leading bytes on the beginning of the next row. As with the number of leading bytes, the wand needs to know this information if it is to combine the program bytes properly. This will also be clarified later with an example.

5.2.3 Program Instructions

The last section of the row consists of program instructions. This section can be up to 13 bytes long since the maximum length of any barcode row is 16 bytes.

To begin this section, the HP-41 instruction set must be known. Access to the HP-41 combined hex/decimal table at this point is almost essential. If the table is not available, another way to learn how different instructions are put together is to dissect other barcodes for which the program instructions are known.

5.2.4 Program Example

Following will be an example program that will be transformed into barcode bytes to illustrate the process involved. The program is very simplistic in nature and only functions to display some alpha text on the HP-41. Hopefully it will also serve as an introduction to generating this type of barcode.

Program Instruction	Number of Bytes in Barcode
01 LBL "BARS"	8
02 "HP41 BARCODE"	13
03 AVIEW	1
04 .END.	3
TOTAL	25

Recalling the above discussion, since the byte count totals 25, we can immediately determine that two rows of barcode will be needed (i.e. - 25 divided by 13 is 1 with 12 bytes left over, thereby requiring 2 rows of barcode.) Before continuing, something needs to be said about where the byte counts came from and how global labels, text strings and the .END. instructions are constructed.

5.2.4.1 Global Label Construction

Each global label requires 4 bytes of overhead plus one byte per character in the name. The breakdown of the overhead is as follows:

Byte #1:	C0	'C' designates ALPHA LABEL or END.
Byte #2:	00	Used by HP-41 for label search address.
Byte #3:	Fn	Where n is the number of characters in the global label name plus 1.
Byte #4:	XX	Key assignment of label. XX is a two nybble keycode corresponding to key being assigned. (See appendix for list.) This byte is usually 00 meaning no assignment is being made.

5.2.4.2 Text String Construction

The format for text strings is as straight forward as that for the global label. The format is one byte of overhead plus one byte per character in the string. The overhead byte is similar to byte #3 in the global label except n is the number of characters instead of the number of characters + 1. The maximum number of characters in the string is 15, which is represented in HEX as the letter F.

5.2.4.3 .END. Construction

The .END. instruction is simply a means for the wand to determine that it has scanned the last row and for it to compile the scanned data. As such, it is found at the end of the last row of barcode. The following three bytes are most often used for this but other combinations will also work: C0 00 2F.

5.2.5 Program Example (cont'd)

Continuing with the program analysis, then, the program bytes would be as follows (the instruction numbers correspond to the instructions above):

```
01  C0 00 F5 00 42 41 52 53
02  FC 48 50 34 31 20 42 41 52 43 4F 44 45
03  7E
04  C0 00 2F
```

From the above breakdown, we can determine the splits between row 1 and 2. Counting 13 bytes from the start of instruction 1 continuing into the second instruction we find the split occurs between 31 and 20 of the second instruction. From this we can count how many bytes of the second instruction will remain in row 1 and how many will 'spill' over into row 2. This will give the leading and trailing byte information needed for overhead byte #3 of this type of barcode. The count shows 5 trailing bytes in row 1 and 8 leading bytes in row 2. Of course, it should be obvious that the leading bytes in row 1 is 0 and likewise with the trailing bytes in row 2.

At this point, with the exception of the checksum, we are ready to put together the rows of barcode. The contents of the two rows are shown below. CS is used to hold a place in the barcode for the running ckecksum that has yet to be computed.

Row #1

```
CS 10 05 C0 00 F5 00 42 41 52 53 FD 48 50 34 31
```

Row #2

```
CS 11 80 20 42 41 52 43 4F 44 45 7E C0 00 2F
```

To compute the running checksum, it is best to view the bytes as binary 1's and 0's. The checksum for any row, as you recall, is the wrap around carry sum of the bytes in the row plus the existing checksum. Since the starting checksum is 00, the checksum for the first row is just the wrap

around carry sum of the last 15 bytes in the row. It is computed below to see exactly what is meant by 'wrap around carry'.

```
  0 0 0 1 | 0 0 0 0₂        Byte #2 - 10
+ 0 0 0 0 | 0 1 0 1₂        Byte #3 - 05
  ----------------
  0 0 0 1 | 0 1 0 1₂
  1 1 0 0 | 0 0 0 0₂        Byte #4 - C0
  ----------------
  1 1 0 1 | 0 1 0 1₂
+ 0 0 0 0 | 0 0 0 0₂        Byte #5 - 00
  ----------------
  1 1 0 1 | 0 1 0 1₂
+ 1 1 1 1 | 0 1 0 1₂        Byte #6 - F5
  ----------------
1 1 1 0 0 | 1 0 1 0₂
  ------------->  1         Wrap around carry
  ----------------
  1 1 0 0 | 1 0 1 1₂
+ 0 0 0 0 | 0 0 0 0₂        Byte #7 - 00
  ----------------
  1 1 0 0 | 1 0 1 1₂
+ 0 1 0 0 | 0 0 1 0₂        Byte #8 - 42
  ----------------
1 0 0 0 0 | 1 1 0 1₂
  ------------->  1         Wrap around carry
  ----------------
  0 0 0 0 | 1 1 1 0₂
+ 0 1 0 0 | 0 0 0 1₂        Byte #9 - 41
  ----------------
  0 1 0 0 | 1 1 1 1₂
+ 0 1 0 1 | 0 0 1 0₂        Byte #10 - 52
  ----------------
  1 0 1 0 | 0 0 0 1₂
+ 0 1 0 1 | 0 0 1 1₂        Byte #11 - 53
  ----------------
  1 1 1 1 | 0 1 0 0₂
+ 1 1 1 1 | 1 1 0 0₂        Byte #12 - FC
  ----------------
1 1 1 1 1 | 0 0 0 0₂
  ------------->  1         Wrap around carry
  ----------------
  1 1 1 1 | 0 0 0 1₂
+ 0 1 0 0 | 1 0 0 0₂        Byte #13 - 48
  ----------------
1 0 0 1 1 | 1 0 0 1₂
  ------------->  1         Wrap around carry
  ----------------
  0 0 1 1 | 1 0 1 0₂
+ 0 1 0 1 | 0 0 0 0₂        Byte #14 - 50
  ----------------
```

```
  1 0 0 0 ¦ 1 0 1 0₂
+ 0 0 1 1 ¦ 0 1 0 0₂      Byte #15 - 34
  ----------------
  1 0 1 1 ¦ 1 1 1 0₂
+ 0 0 1 1 ¦ 0 0 0 1₂      Byte #16 - 31
  ----------------
  1 1 1 0 ¦ 1 1 1 1₂      Byte #1  - EF  (Row 1 CS)
```

The checksum for the second row is determined the same way except the starting checksum is not 00 but is EF. To convince you that this is very tedious, it is suggested that you try to compute it using the above method. If you are already convinced, then use the CHECKSUM routine described in the next chapter to verify that the checksum for the second row is 02.

Now that all the pertinent information is known, CUSTOM can be used to generate the barcode for this program. There will not be any fancy headers stating how many registers are needed for the program or row numbers, just HEX bytes. The correct entries for CUSTOM are shown below with spaces added for separation purposes only. These spaces should not be entered in the actual CUSTOM prompt or an error message, stating that there are invalid hex digits in the string, will be displayed.

EF 10 05 C0 00 F5 00 42 41 52 53 FC 48 50 34 31

02 11 80 20 42 41 52 43 4F 44 45 7E C0 00 2F

5.2.6 Final Comments about Type 1 & 2 Barcode

Hopefully, the above example provided enough background information to get started with type 1 & 2 program barcode generation. A lot can be done with these types of barcode including the full range of synthetic instructions. If you would like a quicker way of generating type 1 & 2 barcode, it is suggested that you purchase the HP-41 ASSEMBLER AND BARCODE GENERATOR. This package was written specifically to generate these two types of barcode and is as simple to use as an assembler.

5.3 Type 4 Barcode Format

Since the type 4 barcode generator routine is included with this package, only a couple of simple examples and a description of the basic format will be discussed. The first example will be an ASN instruction that assigns the example program above to the SHIFT key and the second will be a SIZE instruction to set the size to 50 data registers. If information on other instructions is desired, simply print the barcode for that instruction and dissect it to

determine its makeup. If the type 4 routine won't print the instruction you want to test, consult the HP-41 combined hex/decimal byte table for the proper bytes.

The basic format of type 4 barcode consists of a two byte header followed by a maximum of twelve instruction bytes. The header bytes are shown below in Table 5.2 and a brief description of each will follow.

Table 5.2

Header Byte #1	- Running Checksum	8 bits
Header Byte #2	- Type Number: 4 (0100_2)	4 bits
	- Unused (0000_2)	4 bits
	TOTAL	2 bytes

As seen in the section on type 1 & 2 barcode, the checksum is a wrap around carry sum of the bytes in the row. Since there is only one row, the checksum computation is very straight forward. Even so, the easiest way to find this checksum is to use the CHECKSUM routine described in chapter 6.

The type number of header byte #2 simply informs the wand that type 4 barcode is being scanned. Without this information, the wand would probably generate an error message. I say probably because, depending on the bytes used, the wand may or may not be able to interpret a valid instruction from the bars. If not, it will produce an error.

Having covered the basic format of type 4 barcode, the two examples, described above, will now be discussed and the bytes needed by CUSTOM to produce the barcode for each will be shown.

5.3.1 Example 1: ASN BARS 31

In addition to the two byte header described above, the ASN instruction may contain up to nine instruction bytes. These bytes, starting with the third byte in the barcode, are described below in Table 5.3.

Table 5.3

Instruction Byte #3 - ASN byte ($0F_H$)	1 Byte
Instruction Byte #4 - Keycode (Row,Col)	1 Byte
Instruction Bytes #5-12 - Alpha String	7 Bytes max
TOTAL	9 Bytes max

As you can see, the format for the ASN instruction is slightly rearranged from the way it is entered on the keyboard. That is to say that the keycode actually appears between the ASN instruction byte and the text string. Since our goal is to construct the barcode for ASN BARS 31, we will need to break the instruction down into its parts to find the bytes needed before the checksum can be computed. This is done below.

```
01 CS   (checksum)
02 40   Type number
03 0F   ASN byte
04 31   Keycode for SHIFT key
05 42   ASCII B
06 41     "   A
07 52     "   R
08 53     "   S
```

Having found the instruction bytes, the checksum needs to be computed. Using the CHECKSUM routine of chapter 6 for this yields a checksum of A9. With this information we are ready to enter the bytes at the CUSTOM prompt to produce the barcode. These bytes are shown below with spaces entered between the bytes for separation purposes only. These spaces should not be entered in the actual CUSTOM prompt or an error message, stating that there are invalid hex digits in the string, will be displayed.

```
A9 40 0F 31 42 41 52 53
```

5.3.2 Example 2: SIZE 050

In addition to the header bytes described in Table 5.2, the SIZE instruction requires three bytes. This makes the barcode for this instruction a constant 5 bytes long regardless of the number of data registers desired. These three bytes, starting with the third byte in the row, are described below in Table 5.4.

Table 5.4

Instruction Byte #3 - SIZE byte (06_H)	1 Byte
Instruction Bytes #4 & 5 - Number of desired data registers expressed in Hex	2 Bytes
TOTAL	3 Bytes

Before the checksum can be computed, we will need to convert the number 50 to hex so bytes 4 & 5 will be known. This can be accomplished by the following method if no other method or routine is available for the conversion. Before beginning the conversion process, though, it should be noted that there are only 320 registers to allocate in the HP-41 so the first nybble of byte 4 will <u>always</u> be 0 since 16^3 is 4,096. This means the maximum number of iterations needed to arrive at the correct hex number will be three.

The conversion process involves division by the weighted value of each digit to determine the integer that belongs in that nybble. This should be clear after the conversion, using standard HP-41 functions and notation, is completed.

Find second nybble of byte 4:

50 [ENTER] 256 [/][INT] ---> 0 Byte 4, Nybble 2

Find first nybble of byte 5:

50 [ENTER] 256 [MOD] ---> 50 --> to arrows -->
------> 50 [ENTER] 16 [/][INT] ---> 3 Byte 5, Nybble 1

Find second nybble of byte 5:

------> 50 [ENTER] 16 [MOD] ---> 2 Byte 5, Nybble 2

With bytes 4 & 5 known, we are ready to find the checksum to complete the instruction. Before doing this, let's summarize the bytes determined so far.

```
01 CS   (checksum)
02 40   Type number
03 06   SIZE byte
04 00
05 32   50 in hex
```

With the CHECKSUM routine, or by simple inspection in this case, the checksum byte is determined to be 78. Having determined all of the bytes needed for this instruction, we are now ready to enter them at the CUSTOM prompt to produce the barcode. These bytes are shown below with spaces inserted for separation purposes only. These spaces should not be entered in the actual CUSTOM prompt or an error message, stating that there are invalid hex digits in the string, will be displayed.

78 40 06 00 32

5.4 Type 5 Barcode Format

Since all of the type 5 barcodes are available on the paper keyboard and the labels that come with the wand, it would be a waste of time to cover them in much detail here. They are all made up of one or two bytes and have no type number associated with them. The format of each size, one or two bytes, is listed in Table 5.5 below.

Table 5.5

ONE BYTE		
	Mirror image of function	4 bits
	Function	4 bits
	TOTAL	1 byte
TWO BYTES		
	Wrap around carry checksum	4 bits
	Function	12 bits
	TOTAL	2 bytes

If more information about a specific instruction is desired, dissect the barcode given on the paper keyboard or on the label for its byte makeup. To duplicate the barcode, enter these bytes at the CUSTOM barcode prompt.

5.5 Type 6 Barcode Format

As mentioned in chapter 3, type 6 barcode is rather simplistic in nature. As with the other barcode types, with the exception of type 5, type 6 barcode has a header that consists of a checksum byte and the type number. The format of type 6 barcode is shown below in Table 5.6.

Table 5.6

Header	
Running Checksum	1 byte
Type Number: 6 (0110_2)	.5 bytes
Digits	7.5 bytes
TOTAL	9 bytes

The checksum is just as it is in the other types of barcode, (i.e. - a wrap around carry sum of the other bytes in the row). Notice that the type number only occupies half a byte and there is no unused nybble as there is in type 4 barcode. This nybble is used by the digits in the number which are entered in BCD. The valid BCD digits are shown below in Table 5.7 along with their representation in type 6 barcode.

Table 5.7

BCD Representation	Barcode Representation
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	NULL
B	.
C	+
D	-
E	EEX

The above list is pretty straight forward with the exception of the NULL digit. This digit is used as a filler when the count of digits is even. The following example should clarify how this is used. Suppose we wanted to encode the number:

-6.256E-55

Counting the characters in the number gives 10. Since each character is entered as a BCD digit, each byte in the barcode requires two characters. Since the count is even,

producing 5 barcode bytes, and the header only uses one and a half barcode bytes, we will need to use the NULL digit to 'fill' the row out to an integer number of bytes. This NULL can be at the beginning or the end of the row. With this in mind, let's summarize the bytes needed for this encoding.

```
01 CS   (checksum)
02 6D   Type 6 : -
03 6B   6 : .
04 25   2 : 5
05 6E   6 : EEX
06 D5   - : 5
07 5A   5 : NULL
```

All that is left to do is compute the checksum for byte 1 and print the barcode using CUSTOM. Using the CHECKSUM routine of chapter 6, the checksum was found to be 9C. With that, the hex bytes to enter at the CUSTOM prompt are shown below with spaces entered between the bytes for separation purposes only. These spaces should not be entered in the actual CUSTOM prompt or an error message, stating that there are invalid hex digits in the string, will be displayed.

```
9C 6D 6B 25 6E D5 5A
```

With the above information, any type 6 barcode can be constructed. As with other barcode types, dissection of barcodes of known functions, or in this case data, can prove to be quite useful in obtaining further knowledge of barcode construction.

5.6 Type 7 & 8 Barcode Format

Chapter 4 mentioned that the only difference between type 7 and type 8 barcode is the type number. By now, if this chapter has been read in order, this statement should have significant meaning. The format for type 7 and 8 barcode is shown below in Table 5.8.

Table 5.8

Header	
Running Checksum	8 bits
Type Number: 7 (0111_2) or 8 (1000_2)	4 bits
Character Count	4 bits
Characters	14 bytes
TOTAL	16 bytes

In looking at the header bytes, the checksum is just as it is in the other types of barcode, (i.e. - a wrap around carry sum of the other bytes in the row). The type number used, as mentioned in chapter 4, determines whether the desired alpha barcode is to append or replace the contents of the alpha register. Recall that type 7 replaces the contents of the alpha register and type 8 appends the alpha register. The character count is a hex representation of the number of ASCII characters that will be in the row. The valid range for this is [1-9,A-E].

With the header squared away, let's look at an example to illustrate how the string is constructed. Suppose we want to construct the type 7 string "HP-41 BARCODE". Counting the characters in the string yields 13 (the quotes are not counted). Knowing this and the ASCII codes for the characters in the string, we can summarize the bytes needed for the string. This is done below.

```
01 CS   (checksum)
02 7D   Type 7 : 13 characters (hex D)
03 48   ASCII H
04 50     "   P
05 2D     "   -
06 34     "   4
07 31     "   1
08 20     "   (space)
09 42     "   B
10 41     "   A
11 52     "   R
12 43     "   C
13 4F     "   O
14 44     "   D
15 45     "   E
```

Having determined the bytes needed for the string, the only thing left to determine is the checksum. Using the CHECKSUM routine of chapter 6, the checksum was found to be BA. To summarize, the bytes that should be entered at the CUSTOM prompt are shown below with spaces between them for separation purposes only. These spaces should not be entered in the actual CUSTOM prompt or an error message, stating that there are invalid hex digits in the string, will be displayed.

```
BA 7D 48 50 2D 34 31 20 42 41 52 43 4F 44 45
```

Obviously, the same string could have been done in type 8 format by changing the 7 in byte 7D to 8 and refiguring the checksum.

As a final comment about type 7 and 8 custom barcode, it should be noted that, as stated in chapter 4, certain synthetic characters are not valid. For those who wish to experiment with these characters, like the 'full man' characters and other characters from row 1 of the byte table, be warned that these characters will cause the calculator to 'lock' up when scanned. To 'unlock' the calculator, simply remove and replace the battery pack.

5.7 Final Comments on CUSTOM

As should be obvious by now, CUSTOM can be used to create any type of barcode. With the given formats, any limitations posed by the specific barcode type routine can be overcome with CUSTOM since it has no syntax requirement other than straight hex digits. With this routine available, anyone owning the HP82153A optical wand will be able to fully utilize all of its capabilities.

6. CHECKSUM

CHECKSUM is a utility program that was written specifically to compute the wrap around carry checksum required by HP-41 barcodes. It is included with the package as an 'extra' to make using CUSTOM less tedious and more enjoyable.

6.1 Command Format

CHECKSUM

When the program is run, a screen resembling that shown in figure 6.1 below will appear.

```
+-----------------------------------------------+
|                                               |
|        HP-41 Barcode Checksum Computer        |
|                                               |
|                  Version 1.10                 |
|                                               |
|             (C) 1986 Tony Malburg             |
|                                               |
+-----------------------------------------------+


        B01: --                    B09: --
        B02: --                    B10: --
        B03: --                    B11: --
        B04: --                    B12: --
        B05: --                    B13: --
        B06: --                    B14: --
        B07: --                    B15: --
        B08: --                    B16: --
```

Figure 6.1 CHECKSUM Program Screen

As seen above, the display will show locations for 16 different hex bytes to be entered, B01 - B16. The cursor will be located beside B01 waiting for an entry. Since byte 1 is the checksum byte, you have two options at this point. Either enter 00 as the initial checksum or if you are working on type 1 or 2 barcode and have an existing checksum from the previous row, enter that number instead. From there, after pressing RETURN or ENTER, the cursor moves to B02, B03, etc., updating the checksum with each valid byte, until the last byte is entered.

If the row of barcode is to contain less than 16 bytes, simply press RETURN twice after the last entry. At that

time, a prompt will display at the bottom of the table asking if you are sure you want to quit. If the response is yes, hit [Y] or RETURN and the program will exit leaving the data on the screen. If the response if no, hit [N] and the cursor will return to the table for the next byte of data.

Like other routines in this package, this routine is CASE dependent and requires that all hex digits greater than 9 be entered in UPPER CASE. If a lower case letter is entered accidentally, if the letter is out of the range of valid hex digits, or if more than two digits are entered, an error tone will sound and a message will display at the bottom of the screen reminding you of your mistake. It should be noted that when a mistake is made, the checksum is NOT updated. Only when valid hex bytes are entered will the checksum be computed and updated.

7. PRINTER TYPES

This section will discuss the printer types permitted for use with the programs included with this package. 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.

7.1 EPSON PRINTERS

As was seen in the options section of each type of barcode generator, 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 these programs.

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> and line feed <LF> codes of 13 and 10 respectively are also used but are generally standard on all printers.

7.2 OLYMPIA PRINTERS

The programs contained in this package were 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.

7.3 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.

8. 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

Sources of Information on HP-41 Instruction Set

Of the sources I've seen, the best one is a book by William C. Wickes. The complete instruction set is covered including synthetics. For the complete reference for this book, see chapter 7. The book can be purchased from a California based mail order company who deals in the HP-41, its peripherals and many informational books. The address of this company is shown below along their phone number.

EduCALC MAIL STORE
27953 Cabot Road
Laguna Niguel, CA 92677
PH: (714) 582-2637

In addition to this, there are several users groups around the country that have periodical newsletters that contain a lot of useful information concerning new developments in the field. The two groups that I am most familiar with are listed below with an address to write to for information on membership.

PPC
P.O. Box 90579
Long Beach, CA 90809

CHHU
P.O. Box 10758
Santa Ana, CA 92711-0758

APPENDIX

HEX Keycodes for Key Assignments in Global Labels

The following table provides the keycodes for both the shifted and unshifted keys on the HP-41 keyboard. The keys listed under KEY, with the exception of 42, 43, 44, and 45, correspond to those found in figure 1.1 on page 6 of this manual. The reason for the discrepancy is the double width ENTER key which actually incorporates both 41 and 42. This shifts the count by one when dealing with this row of keys. (i.e. - 43 below corresponds to 42 on the HP-41, etc.)

KEY	Unshifted	Shifted
11	01	09
12	11	19
13	21	29
14	31	39
15	41	49
21	02	1A
22	12	2A
23	22	3A
24	32	4A
25	42	5A
31	03	1B
32	13	2B
33	23	3B
34	33	4B
35	43	5B
41	04	1C
42	14	2C
43	24	3C
44	34	4C
45	44	5C

KEY	Unshifted	Shifted
51	05	1D
52	15	2D
53	25	3D
54	35	4D
61	06	1E
62	16	2E
63	26	3E
64	36	4E
71	07	1F
72	17	2F
73	27	3F
74	37	4F
81	08	16
82	18	24
83	28	36
84	38	44

APPENDIX

BARCODE SAMPLES

Type 4 Barcode

XEQ "BARS"

ASN BARS 31

GTO "BARS"

ASN 31

SIZE 050

Type 6 Barcode

65536

-156.2347E-99

E-27

.375

BARCODE SAMPLES (cont'd)

Type 7 Barcode

"HP-41 BARCODE"

"NONEXISTENT"

" (C) 1986"

"!@#$%^&*:"'"

Type 8 Barcode

(+) "HP-41 BARCODE"

(+) "NONEXISTENT"

(+) "(C) 1986"

(+) "!@#$%^&*:"'"

NOTES