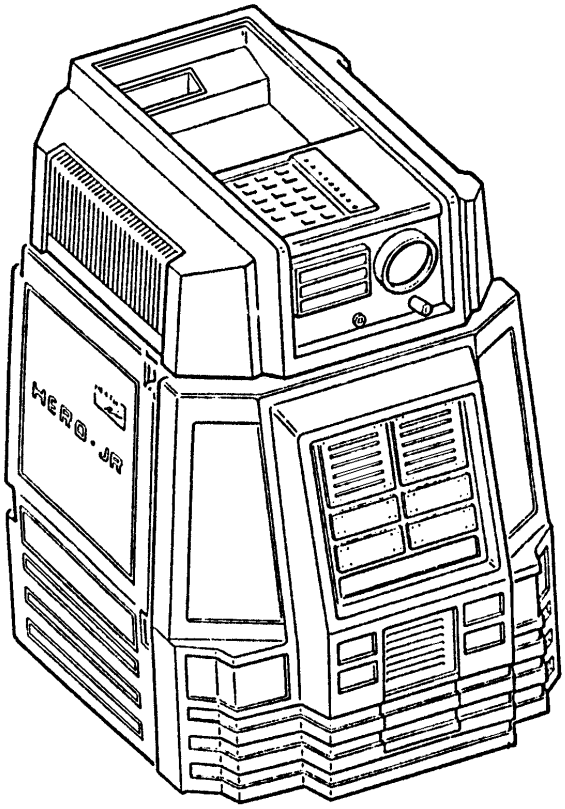

HERO JR

BASIC

RTC-1-8

Manual# 597-3801

www.robotworkshop.com
"Our robots will blow you away"™

Table of contents

OPERATING MODES

The HERO Jr Robot is a computer on wheels. HERO Jr BASIC is a program that uses the keyboard of your terminal to send information, in the form of characters, to the Robot. HERO Jr BASIC also uses the terminal screen to display computer information. If you are using a computer with terminal emulation software, your computer acts as the terminal. It is important to remember that it is the Robot that makes all of the computations.

HERO Jr BASIC allows you two modes of operation: Immediate and Running. When you are in the immediate mode, instructions are interpreted and executed as you enter them from the computer or terminal keyboard. In this mode, you will see a prompt character (>) displayed on the screen. This means that the computer or terminal is ready for a command. When you enter the Running Mode, all instructions to the computer or terminal are interpreted and executed in the ascending numerical order of their line numbers.

Turn the Robot and terminal's power on. If you are using a computer instead of a terminal, turn the computer's power on and run your Terminal Emulation program.

There are four entry points to HERO Jr BASIC: "Cold Start", "Warm Start", and "Running Start".

Key 0 -- Enter debugger first, "G" transfers to Cold Start.
Key 1 -- Cold start.
Key 2 -- Warm start.
Key 3 -- Running start.

The cold start is used when you first turn on the computer or terminal on. It clears the work area (the portion of the Robot's memory where user programs are stored), resets all variables, and displays the Immediate Mode prompt (>) on the terminal screen.

To perform a COLD START, press the 1 key on the Robot's keyboard. The following title banner is displayed on the terminal screen:

```
Wintek HERO Jr BASIC -- VERSION X.X
```

NOTE: The X.X after the "version" indicates the revision level of HERO Jr BASIC.

A WARM START will preserve the program in the work area, clear all the variables, and display the Immediate Mode prompt. Use this entry point to enter the Immediate Mode when you already have a program in the work area. To perform a WARM START, press the 2 key on the Robot's keyboard.

The screen should show the Immediate Mode prompt:

```
>
```

Next, you can type the command to display the saved program on the screen. For example, if you type LIST, on some terminals, the program in the work area will be displayed on the screen to show you it has been saved.

When you use the RUNNING START, the program already in the work area is executed. For example, a program may be written in BASIC and saved on disk via the RS-232C port. You can execute it from memory using the Robot's keyboard without having to use a terminal or computer to begin execution. To perform a RUNNING START, press the 3 key on the Robot's keyboard. NOTE: The program must have previously been loaded from the disk into the Robot's memory.

Keywords

Keywords are the instructions (GOTO, PRINT, etc.) that are recognized and interpreted by HERO Jr BASIC. You need only to enter the first three letters of a keyword to be recognized (REM, DIM, PRI, etc.).

Statements

A statement in HERO Jr BASIC consists of a keyword and any associated expressions or operations. If a statement is entered without a line number, BASIC automatically enters the Immediate Mode and executes the statement at once. If the statement is entered with a line number, it will be saved in the work area for execution later.

You can place multiple statements on numbered lines by separating them with a colon (:). You cannot make multiple statements in the Immediate Mode, since any statement made will automatically be executed.

Statements may be up to 71 characters long (including line numbers and spaces). If you enter a 72nd character, it is echoed, but the entire line is discarded and the Immediate Mode prompt is issued.

Line Numbers

Line numbers must be within the range of 0 to 9999. You may enter them in any order. HERO Jr BASIC will automatically put them in ascending numerical order. When writing a program in BASIC, you should number your lines in multiples of ten or so, in case you find it necessary to insert an instruction later or need the extra space.

Format HERO Jr BASIC lines as follows:

```
nnnn <stat>[:<stat>[:etc...]]
```

The line number is specified by nnnn. You must begin line numbers in the first column of the line and follow them by one or more spaces. HERO Jr BASIC "statements" are indicated by <stat>. Additional statements, as shown in side the brackets [ ], are optional.

Example:

```
10 LET X = 0:PRINT "HELLO WORLD"
20 FOR I = 1 TO 10:PRINT I:NEXT I
30 PRINT "DONE"
```

If you wish to erase the line presently being typed, press the DELETE or RUBOUT key. If you wish to erase characters on the line presently being typed, press the BACKSPACE key or CTRL-H. (NOTE: Even though the characters are still displayed on the screen, they have been erased from memory.) If you wish to erase a line previously entered, type the line number only and press RETURN.

Constants

Constants are actual numbers specified by the programmer. They may be in decimal (base 10) or hexadecimal (base 16) format. Decimal format numbers use digits 0 through 9, while hexadecimal format numbers also use the letters A through F to represent values 10 through 15. HERO Jr BASIC will accept numbers as decimal unless you precede them with a "$" symbol as shown below:

```
64075       (decimal)
$FA4B       (hexadecimal)
```

Constants must be integers within the range of +99999 (+$1869F) to -99999 (-$1869F). They cannot have decimal points, commas, or imbedded spaces. If the result of a calculation exceeds these limits, an error will be reported. Since HERO Jr BASIC deals only with integers, any fractional portion of a number is disregarded.

```
99/100 = 0        (fractional part discarded leaving 0)
101/100 = 1       (fractional part discarded leaving 1)
```

Variables

Variables are represented by the letters A-Z and may have up to two dimensioned subscripts (see DIMENSION) the value may be assigned by a constant or with the value of an expression.

Expressions

Expressions consist of constants or variables to be evaluated. They may be accompanied by arithmetic operators, (+) for addition, (-) for subtraction, (*) for multiplication, (1) for division, (^) for exponentiation, or special functions (refer to "SGN" and "ABS" under "BASIC Statements and Functions" on page 10.)

NOTE: Division by 0 is not permitted. If HERO Jr BASIC encounters such an operation, it will report an error.

Expressions are evaluated from left to right with all operators having equal precedence. Remember: When in doubt, group subexpressions with parentheses.

The following are examples of expressions:

```
15/3        (evaluates to 5)
A^B         (A raises to power B)
3+3*2       (evaluates to 12)
3+(3*2)     (evaluates to 9)
```

Relations

A relation compares the value of two expressions. The result of the comparison will be either true or false. The following table defines the relational operators:

SYMBOL	EXAMPLE	MEANING
=	A = B	A is equal to B
<	A < B	A is less than B
>	A > B	A is greater than B
<=	A <= B	A is less than or equal to B
>=	A >= B	A is greater than or equal to B
<>	A <> B	A is not equal to B

BASIC STATEMENTS AND FUNCTIONS

SCRATCH is used to clear the current user's program from the work area and reset all variables. It is normally used in the Immediate Mode, but it can also appear on a numbered line.

LIST is used to display all or some of the lines of the program in the work area. LIST has several different forms:

```
LIST              Displays the entire program
LIST nnnn         Display line number "nnnn"
LIST nnnn,        Display all lines starting at "nnnn"
LIST nnnn, X      Display X lines starting at "nnnn"
```

Although it is used primarily in the Immediate Mode, LIST may appear on a numbered line. After execution of LIST, however, the program will be suspended and control will return to the Immediate Mode.

RUN is used to execute the program in the work area. Execution always begins with the lowest numbered line.

END suspends program execution whenever it is encountered. Program control returns to the Immediate Mode.

Use the CTRL-C to suspend program execution or LIST at any time. This causes an ATTENTION INTERRUPT (Error number 99), which returns control to the Immediate Mode and causes the prompt to be issued.

CTRL-C is checked between statements and at INPUT prompts. Since some statements which involve Robot functions take a while to execute; it may take some time to be recognized.

MONITOR is used to return to the Debugger Mode of the monitor. The "G" command in the debugger will return to the ">" prompt in BASIC.

PUNCH saves the program in the work area. The program is converted to Motorola's S1/S9 format and sent to the terminal or computer for storage.

LOAD is used to receive a program that is stored in Motorola's S1/S9 format (by PUNCH) from the terminal or computer.

DIMENSION establishes the size of an array. All subscripted variables must appear in a DIMENSION statement before they are referred to by any other statement.

The format of a DIMENSION statement is shown in the following example. The variable to be dimensioned is specified by <var>. The size is specified by <constant1> and the optional <constant2>. Each of these constants must be in decimal form within the range of 0 to 98. Variables may be as large as available memory will permit. Dimensioned variables cannot be used as the index variable specified in a FOR…NEXT statement.

```
DIMENSION <var>(<constant1>[,<constant2>])
```

When more than one variable is specified in a DIMENSION statement, they must be separated by commas as shown in the following statement:

```
DIMENSION A(2,3), B(18), C(5)
```

LET is used to assign a value to a variable. The word "LET" is optional. The first line of the example below shows the format of the statement. The variable specified by <var> is assigned the value of the expression specified by <expr>. The second assigns the variable A the value 15. The third assigns the variable B the value 64.

```
LET <var> = <expr>
LET A = 3*5
B = $40
```

REMARK statement is used to put comments in your program. These statements are ignored by HERO Jr BASIC.

```
10 REM – THIS IS AN EXAMPLE OF A REMARK STATEMENT
```

IF...THEN :ELSE statement is used to perform a test and then conditionally execute a statement or transfer execution to a specified line according to the result. The form is shown in the example below where <expr1> and <expr2> are expressions, <rel op> is a relational operator, <stat> is a statement, and <nnnn> specifies a line number.

```
IF <expr1><rel op><expr2> THEN <stat or nnnn>[:ELSE<stat>]
```

The relational operator compares the values and returns the condition. If the condition is true, the statement following THEN is executed. If a line number was specified, execution is transferred to that line. If the condition returns a false condition, the statement or line number is ignored and the optional ELSE statement is executed. ELSE is executed if, and only if, the comparison is false and it appears directly behind the IF ... THEN statement on the same line separated by a colon. When found anywhere else, it is ignored. If the else statement is omitted, control transfers to the next highest numbered line.

FOR...NEXT statement creates a program loop that may be executed a specific number of times. In the example that follows, <var> is any variable A through Z without a subscript <expr1>, <expr2>, and <expr3> are expressions. <expr2> must not contain any reference to the variable specified by <var>.

Before the loop is executed, all expressions in the statement are evaluated. <var> is assigned the value of <expr1>. The program then executes until the NEXT statement is encountered. The value of <expr3> is added to <var>. If <var> does not exceed <expr2>, then the loop is repeated. Otherwise, execution continues on the line following the NEXT statement. The variable in the NEXT statement must be the same variable specified by <var> in the FOR statement.

```
10 FOR <var> = <expr2> TO <expr2> [STEP <expr3>]
20 PRINT "CURRENT COUNT IS "; <var>
30 NEXT <var>
```

The **STEP** keyword and <expr3> are optional. If they are omitted, the value of <expr3> is assumed to be +1. If the STEP value is the wrong sign (for example, stepping from 1 to ten in increments of -2), or if it is 0, the loop is executed only once.

FOR...NEXT loops may be nested as deep as the stack will permit. Nesting means that a FOR...NEXT loop is contained completely within another FOR...NEXT loop. Each nested loop must have a unique variable. In the example below, note that the entire loop specified by <var2> is within the boundaries of the loop specified by <var1>.

```
10 FOR <var1> = <expr1> TO <expr2>
20 FOR <var2> = <expr3> TO <expr4>
30 ...
40 ...
50 NEXT <var2>
60 NEXT <var1>
```

GOTO is used to change the flow of execution. Execution is transferred to the line number specified by <expr>.

```
nnnn GOTO <expr>

5 INPUT X
10 IF X=1 THEN 30
20 GOTO 100
30 REMARK COME HERE IF X=1
40 ...
...
100 REMARK THIS IS THE END OF THE PROGRAM
110 END
```

GOSUB is used to call a subroutine. Execution is transferred to the line number specified by <expr> and continues until the RETURN statement is encountered. The RETURN statement transfers execution to the statement immediately following the GOSUB statement whether on the same line separated by a colon, or on the next line in the program. GOSUB must appear on a numbered line.

```
nnnn GOSUB <expr>

10 GOSUB 100
20 ...
...
...
100 REMARK THIS IS A SUBROUTINE
110 PRINT "THIS IS A SUBROUTINE"
120 RETURN
```

NOTE: Subroutines may call other subroutines and may be nested as deep as the stack will allow.

ON...GOTO and **ON...GOSUB** are used to transfer execution to one of up to nine specified line numbers, according to the value of an expression. This value should be between 1 and 9.

In the following example, <expr> is an expression and nnnn1 through nnnn9 are the line numbers in their assigned order. For the ON ...GOSUB statement, the line numbers must be subroutines.

```
nnnn ON <exp> GOTO nnnn1, nnnn2, nnnn3,....nnnn9

10 INPUT N
20 ON N GOTO 200, 250, 300, 375
...
100 ON E GOSUB 1000, 1100, 1300, 1560
```

INPUT is used to assign values to variables with data from the terminal or computer. The prompt string is printed on the screen along with a question mark that indicates that HERO Jr BASIC is ready for data.

NOTE: Whenever HERO Jr BASIC prints letters to the screen, they will appear in uppercase.

```
20 INPUT "PROMPT STRING" <var>
```

You may specify more than one variable in an INPUT statement. In this case, the prompt string is printed once along with the question mark. Enter the value of each variable in the order specified, separated by commas. The first line of the example below shows the format of the INPUT statement. The second is a sample.

```
INPUT "PROMPT STRING" <var1>[, <var2>[, etc.]]

10 INPUT "ENTER THE VALUE OF X, Y, AND Z" X,Y,Z
```

DATA statements contain a list of values entered on the same line and separated by commas. These values may be expressions or constants and are assigned to variables in READ statements. Any expressions are evaluated when the READ is encountered. Whenever a DATA statement is encountered, it becomes the current DATA statement. This allows you to change the values to be assigned while a program is running.

When the computer encounters line 10 of the following example, it becomes the current DATA statement. Then, on line 20, the variable W is assigned the value of 1. Similarly, the variables X, Y, and Z are assigned the values 2, 3, and 4 respectively. When line 30 is encountered, it becomes the current DATA statement. Line 40 assigns the value 5 to variable A. On line 50, the variables W, X, and Y are assigned the values 5, 6, and 7. Variable Z is assigned the value of variable A (5) + 4, or 9.

```
10 DATA 1, 2, 3, 4
20 READ W, X, Y, Z
30 DATA 5, 6, 7, A+4
40 LET A = 5
50 READ W, X, Y, Z
```

The **RESTORE** statement causes the "data pointer", which points to the next value to be assigned, to point to the first value on the current DATA statement. On line 29 of the following example, variables X and Y are assigned the values 2 and 4. The RESTORE statement on line 30 resets the "data pointer" so that, on line 40, the variable Z is assigned the value 2.

```
10 DATA 2, 4, 6, 8
20 READ X, Y
30 RESTORE
40 READ Z
```

Use the **PRINT** statement to display a list of values and/or text on the screen. Strings of text must be enclosed in quotation marks. Punctuation used in the PRINT statement determines the actual position on the screen to which each item in the print list will be printed. The following functions are also available to format the output of print statements: **TAB**, **SPC**, and **CHR**.

There are 10 print zones per printed line. These zones are eight columns wide and start in columns 1, 9, 17, 25, 33, 41, 49, 57, 65, and 73 of the printed line. In the following example, the items in the print list are separated by commas. <item1> is displayed in the first print zone (column 1 of the printed line). <item2> is printed in the next available print zone. If a string of text extends beyond its print zone, the next item in the list will be displayed in the next available print zone.

```
PRINT <item1>, <item2>
```

To skip print zones, insert on comma for each zone to be skipped. In the example below, <item1> is printed in the second print zone starting in column 9. <item2> is printed in the fifth print zone starting in column 33.

```
PRINT ,<item1>,,,<item2>
```

You can print items with no spacing between them by using semicolons (;) instead of commas. If a semicolon is placed at the end of the print list, the list will be printed, but no line feed will be issued at the end of the line. This allows items from PRINT statements on different lines of the program to be displayed on the same physical line on the screen. In the example below, the message "HERO JR BASIC" is printed on one line of the screen.

```
10 PRINT "HERO JR ";
20 PRINT "BASIC"
```

You can print an item starting in a specified column by using the **TAB** function. The TAB function cannot specify a column number lower than or equal to one that already has been printed. If this occurs, the TAB will be ignored.

In the example below, <expr2> must be greater than <expr1>

```
PRINT TAB(<expr1>); <item1>; TAB(<expr1>); <item2> etc ...
```

The **SPC** function prints a specified number of spaces between or in front of items in the print list.

```
PRINT SPC(<expr1>); <item>
```

The **CHR** function prints a specified ASCII character. The character is determined by the absolute value modulo 256 of an expression. If the value is greater than or equal to 128, the character is printed but the column counter is not advanced. (Refer to the "ASCII to Decimal and Hexadecimal Conversion Chart" on Page 14.)

```
PRINT CHR(<expr>)
```

The **PEEK** function reads values from the Robot's memory or I/O ports. The address is specified by an expression. Remember: PEEK only returns a value. If you want the value to be output to the screen, the PEEK function must appear in a PRINT statement: PRINT PEEK(<expr>).

```
PEEK(<expr>)
```

The **POKE** statement writes values to the Robot's memory or I/O ports. The values may be within the range of 0 to 255 (0 to $FF hexadecimal). In the example below, <expr1> specifies the address and <expr2> specifies the value to be written.

```
POKE <expr1>, <expr2>
```

USER calls an assembly language subroutine already residing in the Robot's memory. (A subroutine can be entered by using the HERO Jr debugger. See Page 16.)

You must place the address of the subroutine at the user vector with the high order byte of the address at 0800 and the low order byte at 0801. The user subroutine must not reside between addresses $0800 to $5FFF. The following example shows how to set up this vector. The example shows both the assembly language subroutine for reading and speaking the date and the BASIC program to read, speak, and display the date on a terminal.

```
                    ***  4K basic USR function to read/speak
                    *    the time
EDE0                .RDCLK    SET  $EDE0
CBF2                SPKTIM    SET  $CBF2
0734                          ORG   $734
0734                BUFFER    RMB  7
073B CE 0734                  USR  LDX   #BUFFER
073E BD EDED                  JSR  .RDCLK
0741 86 34C6                  LDAD  #BUFFER
0744 07

0745 36 37                    PSHD
0747 86 01                    LDAA #$01
0749 36                       PSHA
074A BD CBF2                  JSR  SPKTIM
074D 39                       RTS
                    *    Set vector to user function
0800                          ORG  $800
0800 07 3B                    FDB  USR
0802                          END

10 REM FIRST CALL "USER" TO READ/SPEAK TIME
20 USER
30 REM NOW EXTRACT HH/MM/SS
40 H = PEEK($736)
50 IF H > 12 THE H = H - 128
60 M = PEEK($735)
70 S = PEEK($734)
80 PRINT "THE CURRENT TIME "; H;".";M;".";S
```

The **SGN** function returns the sign of the value of an expression. For expressions that have a value greater than zero, the value +1 is returned. If the expression is less than zero, the value -1 is returned. If the value of the expression equals zero, a 0 is returned.

```
SGN(<expr>)
```

The **ABS** function returns the absolute value of an expression.

```
ABS(<expr>)
```

RND generates a random number in the range of 0 to 99.

ROBOT STATEMENTS AND FUNCTIONS

With the **SPEAK** statement, the Robot can use its voice synthesizer to simulate human speech. This statement has two forms. With the first form, the Robot will speak a preprogrammed phrase at the address specified by the value of the expression. The first line of the example below shows the format. The second and third lines are samples.

```
SPEAK <expr>
SPEAK $AD48
SPEAK 44342
```

With the second form, the Robot translates the phoneme mnemonics contained in <phoneme string>. These mnemonics are illustrated in depth in the "Robot Voice Dictionary." The "Alphanumeric Listing of Phoneme Mnemonics" reference table in the back lists the mnemonics in alphanumeric order.

```
SPEAK <phoneme string>
SPEAK "THV I S PA0 I1 Z UH1 F O W N EH1 M PH0 S T R I1 NG PA1"
```

Remember: For proper operation, always put a PA0, PA1, or STOP phoneme at the end of a phoneme string.

You can select four levels of inflection by placing 1, 2, 3, or 4 in front of a phoneme in the string, with 1 being the lowest level and 4 the highest. These may be placed anywhere within the string and will remain in effect until the next level is encountered, or until the end of the string. If no level is specified, the default value is level 1.

```
10 SPEAK "1W UH N PA1"
20 SPEAK "2T IU U W PA1"
30 SPEAK "3 TH R E1 Y PA1"
40 SPEAK "4F 01 ER PA1"
50 SPEAK "4F 01 ER PA1 3 TH R E1 Y PA1 2T IU U W PA1 1W UH M PA1"
```

Four commands govern the movement of the Robot's base. Movement is specified in inches **FWD** (forward) or **BWD** (backward) or in degrees **LEFT** or **RIGHT**. FWD and BWD move the Robot along a straight line. LEFT and RIGHT spin the Robot on its axis.

```
<direction> <expr>

10 FWD 4
20 LEFT 6
30 BWD 8
```

To compensate for variations in the Robot's environment, two special variables have been provided to calibrate the Robot's base movements. **LCF** (Linear Calibration Factor) is used to adjust the length the Robot moves during FWD and BWD commands. **TCF** (Turn Calibration Factor) adjusts how far the Robot spins during LEFT and RIGHT commands.

Both of these variables default to a value of 100%, but may be set to values from 0 to 233%. For consistency of movement, they should appear before any base movement command is encountered.

```
<LCF or TCF> = <expr>
```

Data from each of the Robot's sensors may also be addressed as a variable.

Variable	Range	Units	Comments
EYE	0 to 255		
EAR	0 to 255		
SONAR	0 to 157	Inches	
MOTION	0 or 1		1 = Motion

The **KEYIN** function waits for and returns the value of a key on the Robot's keyboard.

```
<var> = KEYIN
```

NOTE: The CTRL-C command will not respond during a KEYIN function. Use the RESET key on the Robot's keyboard; then press the 2 key for a warm start.

The **RADIO** function returns the number of a key pressed on the Remote Control. If no key is pressed it will just return 0.

```
<var> = RADIO

10 X = RADIO
20 PRINT X
```

ERROR CODES

NUMBER	MEANING
10	Unrecognized keyword
12	Error while loading program
14	Illegal variable
16	No line number reference by GOTO or GOSUB
18	Equal sign missing in assignment statement
20	Expression syntax error or unbalanced parenthesis
21	Expression expected but not found
22	Division by zero
23	Arithmetic overflow
24	Expression too complex
31	Syntax error in PRINT statement
32	Missing closing quote in printed string
40	Error in DIM statement
45	Syntax error in INPUT statement
51	Syntax error in READ statement
62	Syntax error in IF statement
73	RETURN without GOSUB
81	FOR/NEXT error
90	Memory overflow
99	CTRL-C (attention interrupt)

ALPHANUMERIC LISTING OF PHONEME MNEMONICS

MNEMONIC	DURATION	EXAMPLE
A	185 ms	dAY
A1	103 ms	mAde
A2	71 ms	mAde
AE	185 ms	dAd
AE1	103 ms	After
AH	250 ms	mOp
AH1	146 ms	fAther
AH2	71 ms	hOnest
AW	250 ms	cAll
AW1	146 ms	lAWful
AW2	30 ms	sAlty
AY	21 ms	dAY
B	71 ms	Bag
CH	71 ms	CHop
D	55 ms	faDe
DT	47 ms	buTTer
E	185 ms	mEEt
E1	121 ms	bE
EH	185 ms	gEt
EH1	121 ms	hEAvy
EH2	71 ms	Enlist

MNEMONIC	DURATION	EXAMPLE
EH3	59 ms	jackEt
ER	146 ms	bIRd
F	103 ms	Fast
G	71 ms	Get
H	71 ms	Hello
I	185 ms	pIn
I1	121 ms	inhIbit
I2	80 ms	Inhibit
I3	55 ms	inhibIt
IU	59 ms	yOU
J	47 ms	JuDGe
K	80 ms	triCK
L	103 ms	Land
M	103 ms	Mat
N	80 ms	suN
NG	121 ms	riNG
O	185 ms	cOld
O1	121 ms	abOArd
O2	80 ms	fOr
OO	185 ms	bOOk
OO1	103 ms	lOOking
P	103 ms	Past
R	90 ms	Red
S	90 ms	paSS
SH	121 ms	SHop
T	71 ms	Tap
TH	71 ms	THin
THV	80 ms	THe
U	185 ms	mOve
U1	90 ms	yOU
UH	185 ms	cUp
UH1	103 ms	Uncle
UH2	71 ms	About
UH3	47 ms	missIOn
V	71 ms	Van
W	80 ms	Win
Y	103 ms	anY
Y1	80 ms	Yard
Z	71 ms	Zoo
ZH	90 ms	aZure
PA0	47 ms	silent
PA1	185 ms	silent
STOP	47 ms	silent

ASCII TO DECIMAL AND HEXIDECIMAL CONVERSION CHART

ASCII	DECIMAL	HEX
NUL	0	$00
SOH	1	$01
STX	2	$02
ETX	3	$03
EOT	4	$04
ENQ	5	$05
ACK	6	$06
BEL	7	$07
BS	8	$08
HT	9	$09
LF	10	$0A
VT	11	$0B
FF	12	$0C
CR	13	$0D
SO	14	$0E
SI	15	$0F
DLE	16	$10
DC1	17	$11
DC2	18	$12
DC3	19	$13
DC4	20	$14
NAK	21	$15
SYN	22	$16
ETB	23	$17
CAN	24	$18
EM	25	$19
SUB	26	$1A
ESC	27	$1B
FS	28	$1C
GS	29	$1D
RS	30	$1E
US	31	$1F
SP	32	$20
!	33	$21
“	34	$22
#	35	$23
$	36	$24
%	37	$25
&	38	$26
‘	39	$27
(	40	$28
)	41	$29
*	42	$2A
+	43	$2B
,	44	$2C
-	45	$2D
.	46	$2E
/	47	$2F
0	48	$30

ASCII	DECIMAL	HEX
1	49	$31
2	50	$32
3	51	$33
4	52	$34
5	53	$35
6	54	$36
7	55	$37
8	56	$38
9	57	$39
:	58	$3A
;	59	$3B
<	60	$3C
=	61	$3D
>	62	$3E
?	63	$3F
@	64	$40
A	65	$41
B	66	$42
C	67	$43
D	68	$44
E	69	$45
F	70	$46
G	71	$47
H	72	$48
I	73	$49
J	74	$4A
K	75	$4B
L	76	$4C
M	77	$4D
N	78	$4E
O	79	$4F
P	80	$50
Q	81	$51
R	82	$52
S	83	$53
T	84	$54
U	85	$55
V	86	$56
W	87	$57
X	88	$58
Y	89	$59
Z	90	$5A
[	91	$5B
\	92	$5C
]	93	$5D
^	94	$5E
_	95	$5F
`	96	$60
a	97	$61
b	98	$62
c	99	$63
d	100	$64

ASCII	DECIMAL	HEX
e	101	$65
f	102	$66
g	103	$67
h	104	$68
i	105	$69
j	106	$6A
k	107	$6B
l	108	$6C
m	109	$6D
n	110	$6E
o	111	$6F
p	112	$70
q	113	$71
r	114	$72
s	115	$73
t	116	$74
u	117	$75
v	118	$76
w	119	$77
x	120	$78
y	121	$79
z	122	$7A
{	123	$7B
|	124	$7C
}	125	$7D
~	126	$7E
DEL	127	$7F

HERO JR DEBUGGER

The following paragraphs list the debugger commands and explain some of their uses. For further debugger information, refer to the Heath ET-3400 educational course.

DEBUGGER COMMANDS

KEY	COMMAND
M	Display/Change Memory Bytes
I	Display/Change Instruction
R	Display MPU Registers
CNTL-A	Display/Change Accumulator A
CNTL-B	Display/Change Accumulator B
CNTL-C	Display/Change Condition Codes Register
CNTL-P	Display/Change Program Codes Register
CNTL-X	Display Change Index Register
S	Single Step User Program
G	Go to User Program
D	Dump Memory
L	Load Memory

The HERO Jr Debugger also supports:

- free-format hexadecimal input
- insertion and deletion of breakpoints
- mnemonic form instruction display
- lower case input

Free-format means that you may omit leading zeros when you enter hexadecimal memory addresses, data, or opcodes. Also, you can imbed spaces between the digits, and you can use upper or lower case. For example, to enter address "00A0", the following are equivalent: "00A0", "0A0", "A0", "0 a0". To enter the data byte "02", you may type "02" or "2".

MEMORY ACCUMULATOR OR REGISTER DISPLAY/CHANGE

To display the content of any memory location, type "M" followed by the hexadecimal address of that memory location. For example, to display the content of memory location 0001, start by typing "M" (the debugger responds by typing a space); then type "0001 ", or "001 ", or "01 ", or "1" followed by RETURN. The debugger responds by printing the memory location, followed by a space, followed by the content, e.g. "0001 03". To display the content of the next memory location, hit RETURN. The debugger will respond with "0002 7E" (assuming 7E is the content of 0002). To display successive memory locations, keep hitting RETURN. To exit the display mode and return to the command mode, type "!" (escape). The debugger will respond with "d>", indicating it is ready for another command.

To change the content of any memory location, first display that memory location and then type the replacement value followed by RETURN. To return to the command mode, type "!".

To display the content of the C, B, A, X, or P register, type "C", "B", "A", "X", or "P" while you hold down the "CTRL" key.

INSTRUCTION DISPLAY/CHANGE

The instruction display/change mode is similar to the memory display/change mode except that the debugger prints one instruction per line rather than one byte per line. An instruction can be one, two, or three bytes; that is, an opcode alone or an opcode followed by a one- or two-byte operand. A RETURN causes the debugger to advance to the next instruction. An "!" returns the debugger to the command mode.

GO TO USER PROGRAM

The "G" command causes the debugger to begin execution of a user program at the point specified by the value in the program counter register. Execution continues until the user program returns control to the debugger, the program issues an "ssr exit", or until you press the RESET switch.

SINGLE STEP USER PROGRAM

The "S" command is similar to the "C" command except that only a single instruction is executed in the user program. Control is returned to the debugger which prints the user registers after the instruction has been executed.

GO WITH BREAKPOINTS

The "G" command can be used to execute the user program until a specific instruction is reached. To use this feature, type "G" followed by the hexadecimal address of the first "breakpoint", followed by "," followed by the hexadecimal address of the second "breakpoint", etc., followed by two periods. The debugger begins execution of the user program at the address specified by the counter and continues execution until one of the breakpoints is encountered, whereupon the debugger regains control, and displays the user registers.

NOTE: Breakpoints can be placed only in RAM and only at addresses containing an opcode.

DUMP MEMORY

To display an area of memory, type "D" followed by the address of the first memory location to be displayed, followed by a comma, followed by the address of the last memory location to be displayed, then hit RETURN. The debugger responds by displaying the contents of this range of memory locations in hexadecimal, separated by spaces with 16 bytes per line. Each line starts with the address of the first byte on that line.