THE HANDHELD CALCULATOR
WITH
ASW APPLICATIONS

Rex H. Shudde
Department of Operations Research
Naval Postgraduate School

MAY 1978

TABLE OF CONTENTS

INDEX TO EXAMPLES

LIST OF FIGURES

P R E R E Q U I S I T E S

The reader is expected to have read and become familiar with pages 15-107 and 118-167 of the HP-67 "*Owner's Handbook and Programming Guide*"† and to have worked through the examples contained on those pages.

---

*Herein referred to as *Owner's Handbook*.

I. INTRODUCTION

Uses of the handheld programmable calculator (HPC) are rapidly being found in all areas of endeavor. To the uninformed, the HPC appears to be nothing more than an expensive toy, but to people involved in problem solving the HPC is an indispensible tool which can perform routine or tedious analysis quickly. A major advantage of the HPC is its portability; it goes where you go and it can be used in any location with or without an auxiliary power source.

One area of application of the HPC is to provide a computational aid in solving environmental tracking and localization algorithms used by search platforms in Anti-Submarine Warfare (ASW). The Hewlett-Packard HP-67 Handheld Programmable Calculator has been adopted for this task. With the adoption of this HPC have come the additional tasks of (1) providing training in the use of the HPC in the real-time operational environment using available programs and (2) providing additional education in the use of the calculator functions in general and the techniques and methods of designing and constructing programs for the HPC in particular.

The material presented here has been developed from the author's lecture notes for a short course which has been presented by the Office of Continuing Education of the Naval Postgraduate School at the Naval Air Stations at Moffett Field,

California; Brunswick, Maine; and Jacksonville, Florida. The material has been designed to provide additional education in the use of the HPC and hopefully to remove the Black-Box mystique that generally prevails during the initial learning phases. The material does not address actual programs that are used in ASW; it is oriented toward the use of the calculator and the demonstration of techniques that are used in many of the ASW programs.

II. VECTORS

In ASW applications "vector" inputs are almost always in polar form: bearing and range, or course and speed. Almost all tactical problems require that these inputs be "added" or "subtracted" in order to achieve some end result. It has been common practice to place these quantities onto a plotting table so that these "vector" quantities can be conveniently manipulated.

Although plotting tables are a great aid to "visualizing" tactical situations, the use of such graphical aids can be quite time consuming by modern standards. In a real-time tactical encounter, speed is of the essence; and such speed is available with properly programmed computers or even with computational aids such as the HPC.

Vector operations are implemented easily on the HP-67 by using a combination of the Polar/Rectangular functions [f] [R←] and [g] [→P]; and the summation functions [Σ+], [h] [Σ-], and [RCL] [Σ+]. Each of these functions will be discussed separately and then the functions will be combined into useful vector operations.

Polar/Rectangular Function Conventions

The polar-to-rectangular [f] [R←] and rectangular-to-polar [h] [→P] functions are discussed on pages 98 through 103 of the HP-67 *Owner's Handbook*. The polar diagram on page 99 is commonly used by mathematicains and electrical engineers, but it is awkward to use for navigation and relation problems.

Throughout this text the following conventions will be used:

1. North is the direction of the positive x-axis and East is the direction of the positive y-axis.
2. Bearing angles and courses will be measured from North at 0° and will become increasingly positive in the clockwise direction from North.

The HP-67 was designed primarily to be used for mathematical and scientific applications rather than navigational problems and so some care must be used with regard to Item 2. In particular, bearings and courses *as calculated* on the HP-67 will be positive and increase in the clockwise direction from North to +180° only, and will be negative and decrease in the counterclockwise direction from North to -180°. Fortunately, this is only a minor inconvenience and is remedied by adding 360° to negative bearings and courses; the result is that bearings and courses will follow the usual navigational convention of increasing from 0° to +360° as measured in the clockwise direction from North. These conventions are illustrated in Figure 1.

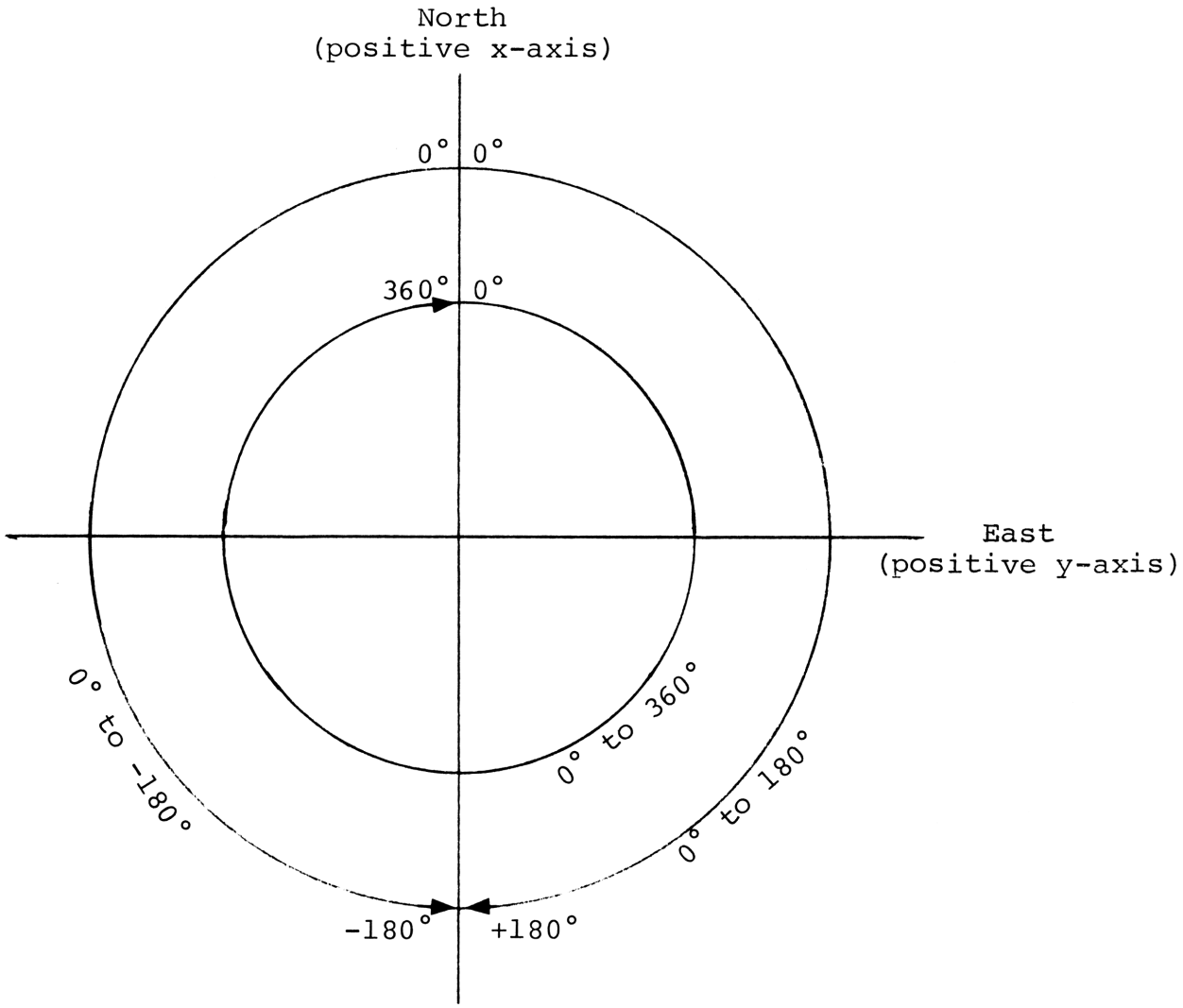

FIGURE 1. Polar Coordinate Conventions
Inner circle: Navigation Convention.
Outer circle: Mathematical Convention on HP-67.

Input and output information is almost always in the polar form of bearing and range or course and speed. Unfortunately we cannot manipulate directly with the quantities; they must first be converted to a rectangular or x,y-coordinate system. The vector operations of addition and subtraction are performed in the rectangular system; then the results are converted to the polar system.

Polar-to-Rectangular Conversion

To convert polar coordinates to rectangular coordinates in the HP-67 we must first enter the bearing angle in *decimal degrees* (if the input is in degrees and minutes, then this must be converted to decimal degrees using the [f] [H←] function discussed in pages 94 through 96 of the HP-67 *Owner's Handbook*), and then we enter the range. More explicitly, the angle is placed in the Y-register and the range is placed in the X-register. The function keys [f] [R←] are then pressed. The resulting rectangular x-coordinate is the X-register and the rectangular y-coordinate is in the Y-register.

EXAMPLE 1. Rectangular Coordinates from Bearing and Range.

A contact is at bearing 075°, and range 20,000 yards from own ship (Figure 2). What are the rectangular coordinates of the contact?

Solution:

Press	Display	Comment
075 [↑] *	[75.00]	Enter the bearing (polar angle) into the Y-register
20000 [f] [R←]	[5176.38]	The x-coordinate of the target is 5176.38 yards (North) from own ship
[h] [x ⇄ y]	[19318.52]	Exchange the X- and Y-registers. The y-coordinate of the target is 19318.52 yards (East) from own ship.

*The symbol [↑] is used to designate the function key [ENTER].

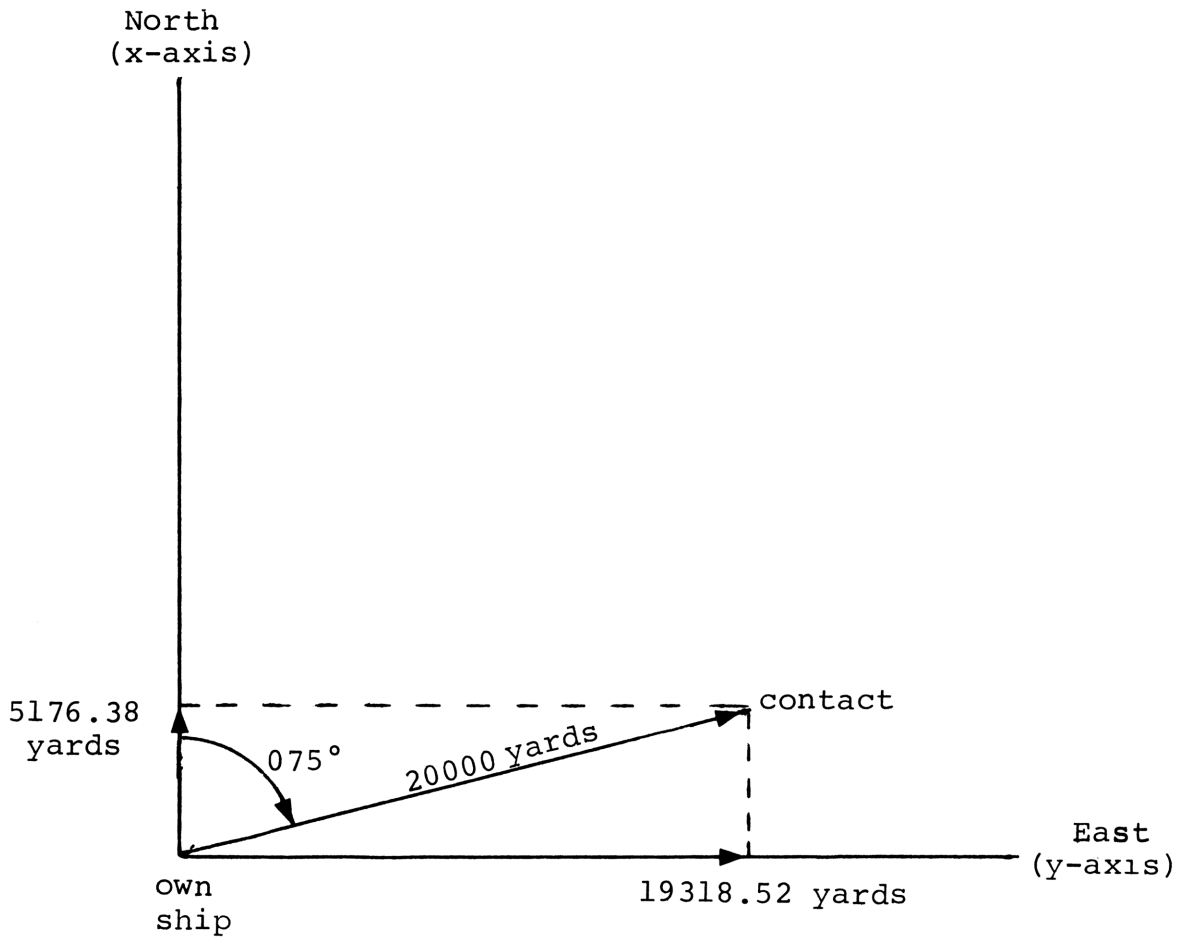

FIGURE 2. Rectangular Components of a Contact.

EXERCISES:

Convert the following bearing and range data to rectangular coordinates.

	Input		Solution	
	Bearing	Range	x-coordinate	y-coordinate
1.	050°	10,000 yds	6427.88 yds (6427.88 yds North)	7660.44 yds (7660.44 yds East)
2.	123°	4,000 yds	-2178.56 yds (2178.56 yds South)	3354.68 yds (3354.68 yds East)
3.	245°	25 n.mi	-10.57 n.mi (10.57 n.mi South)	-22.66 n.mi (22.66 n.mi West)
4.	336°	30 k.yd	27.41 k.yd (27.41 k.yd North)	-12.20 k.yd (12.20 k.yd West)

Convert the following course and speed data to rectangular coordinates.

	Course	Speed	x-component of velocity	y-component of velocity
5.	035°	20 kts	16.38 kts (16.38 kts towards the North or from the South)	11.47 kts (11.47 kts toward the East or from the West)
6.	320°	30 ft/sec	22.98 ft/sec (22.98 ft/sec towards the North or from the South)	-19.28 ft/sec (19.28 ft/sec towards the West or from the East)
7.	190°	15 kts	-14.77 kts (14.77 kts towards the South or from the North)	-2.60 kts (2.60 kts toward the West or from East)
8.	115°	25 kts	-10.57 kts (10.57 kts towards the South or from the North)	22.66 kts (22.66 kts towards the East or from the West)

These exercises illustrate the following points:

1. A positive x-coordinate is North.
2. A negative x-coordinate is South.
3. A positive y-coordinate is East.
4. A negative y-coordinate is West.
5. The range may be in any convenient measure such as yards, kiloyards, nautical miles, feet, etc.; the x- and y-coordinates are in the same measure.
6. Course and speed are treated in the HP-67 in exactly the same way as bearing and range except that the contents of the X and Y registers denote the x-component and y-component of velocity.
   a. A positive x-component is *toward* North or *from* South.
   b. A negative x-component is *twoard* South or *from* North.
   c. A positive y-component is *toward* East or *from* West.
   d. A negative y-component is *toward* West or *from* East.

Rectangular-to-Polar Conversion

The conversion of rectangular coordinates to polar coodinates is the *inverse* of the conversion of polar coordinates to rectangular coordinates. To convert rectangular coordinates to polar coordinates in the HP-67 we must first enter the y-coordinate and then we enter the x-coordinate. More explicitly, the y-coordinate or y-component is placed in the Y-register and the x-coordinate or x-component is placed in the X-register. The function keys [g] [→P] are then pressed. The resulting range or speed is in the X-register and the bearing or course is in the Y-register.

EXAMPLE 2. Bearing and Range from Rectangular Coordinates.

A contact is 6000 yards North and 8000 yards East. What is the range and bearing of the contact?

Solution:

Press	Display	Comment
8000 [↑]	[8000.00]	The East/West or y-component must be entered first. Use +8000 since the contact is East.
6000 [g] [→P]	[10000.00]	The North/South or x-component is placed in the X-register (+6000 since the contact is North). The range of 10,000 yards is in the display.
[h] [x ⇄ y]	[53.13]	Exchange the X- and Y-registers. The bearing angle of 53°.13 is in the display.

EXAMPLE 3. Bearing and Range from Rectangular Coordinates.

A contact is in 10 n.mi West and 15 n.mi South. What is the range and bearing of the contact?

Solution:

Press	Display	Comments
10 [CHS] [↑]	[-10.00]	The East/West or y-component is entered first. Use -10 since the contact is West.
15 [CHS]	[-15.00]	The North/South or x-component is placed in the X-register. Use -15 since the contact is South.
[g] [→P]	[18.03]	The range of the contact is 14.14 n.mi.
[h] [x ⇄ y]	[-146.31]	The bearing of the contact is 146°.31 *counterclockwise* from North.
360 [+]	[213.69]	Since the bearing angle was negative, add 360°. The bearing of the contact is 213°.69 *clockwise* from North. This is the conventional bearing angle.

EXAMPLE 4. Course and Speed from Rectangular Coordinates.

A contact is moving at 12 kts toward the North and 15 kts toward the West. What is the course and speed of the contact?

Solution:

Press	Display	Comment
15 [CHS] [↑]	[-15.00]	The East/West component of velocity is entered first. Use -15 since the contact is moving *toward* the West.
12	[12]	The North/South component is placed in the **X**-register. Use +12 since the contact is moving *toward* the North.
[g] [→P]	[19.21]	The speed of the contact is 19.21 kts.
[h] [x ⇄ y]	[-51.34]	The course is negative, so 360° must be added.
360 [+]	[308.66]	The contact's course is 308°.66. (Note: This is *decimal* degrees.)
[g] [→H.MS]	[308.39]	The contact's course is 308°39'.

EXERCISES:

Convert the following rectangular positions to bearing and range.

	Input		Solution		
	x-position	y-position	Input	Range	Bearing
1.	15 kyds North	20 kyds East	20 [↑] 15	25.00 kyds	53°.13
2.	12 kyds South	15 kyds East	15 [↑] -12	19.21 kyds	128°.66
3.	10 n.mi South	13 n.mi West	-13 [↑] -10	16.40 n.mi	-127°.57 or 232°.43
4.	8 n.mi North	12 n.mi West	-12 [↑] 8	14.42 n.mi	- 56°.31 or 303°.69

Convert the following rectangular velocity components to course and speed.

	x-component	y-component	Input	Speed	Course
5.	10 kts toward North	8 kts toward East	8 [↑] 10	12.81 kts	38°.66
6.	10 kts from North	8 kts toward East	8 [↑] -10	12.81 kts	141°.34
7.	10 kts toward South	8 kts toward East	8 [↑] -10	12.81 kts	141°.34
8.	8 kts from North	15 kts from East	-15 [↑] - 8	17.00 kts	-118°.07 or 241°.93
9.	8 kts toward South	15 kts toward West	-15 [↑] - 8	17.00 kts	-118°.07 or 241°.93

Summation Function Keys

The summation functions [Σ+], [h] [Σ-], and [RCL] [Σ+] are described with the statistical functions on pages 107 through 111 and 116 to 117. We will not be concerned with the statistical functions mean and standard deviation, but only with the operation of the three summation functions. The use of [Σ+] and [h] [Σ-] affects the contents of the secondary storage registers S4, S5, S6, S7, S8, and S9. It is important that the effect is recognized when programs that require the use of the secondary storage registers are being written. However the operation of only two of these six registers, S4 and S6, is important for vector operations.

When the [Σ+] key is pressed, the contents of the x-register are *added to* the contents of the S4 storage register and simultaneously the contents of the Y-register are *added to* the contents of the S6 storage register. It is important to note that the words "added to" are used; the contents of the X-register and Y-register are *not stored* in secondary registers S4 and S6, respectively, but they are *added to the contents* of S4 and S6, respectively. This is why the [Σ+] key is referred to as a summation (Σ is the Greek letter *sigma* and is used to denote summation).

When the [h] [Σ-] keys are pressed, the contents of the X-register are *subtracted from* the contents of the S4 storage register and simultaneously the contents of the Y-register are *subtracted from* the contents of the S6 storage register.

When the [RCL] [Σ+] keys are pressed, the contents of the X-register are *replaced by* the contents of the S4 storage register and simultaneously the contents of the Y-register are *replaced by* the contents of the S6 storage register. The contents of the S4 and the S6 registers remain unchanged during this operation.

After the [Σ+] or the [h] [Σ-] keys are pressed, the contents of the X-register are changed. Whatever number appears in the display is meaningless for our purposes and should be ignored.

A rapid method for clearing (setting to zero) the S4 and S6 storage registers is the following keystroke sequence:

[RCL] [Σ+]	Recall the contents of storage registers S4 and S6 and place these contents in the X-register and Y-register, respectively. The contents of S4 and S6 are unaltered.
[h] [Σ-]	Subtract the contents of the X-register from S4 and subtract the contents of the Y-register from S6. The result is that S4 contains 0 and S6 contains 0.

EXAMPLE 5. Vector Addition with the $\Sigma+$ Function Key.

Let $x_1 = 3$ and $y_1 = 8$. Let $x_2 = 4$ and $y_2 = -5$. Find $x_3 = x_1 + x_2$ and $y_3 = y_1 + y_2$ using the summation function keys. The y-values are to be placed in the Y-register and the x-values are to be placed in the X-register.

Solution:

Press	Display	Comment
[RCL] [$\Sigma+$] [h] [$\Sigma-$]	could be anything Ignore	This key sequence clears the secondary registers S4 and S6. If the calculator was just turned on, the contents will be zero, but if a program is being used it is safest to clear these registers.
8 [↑]	8.00	Place y_1 in the Y-register.
3	3.	Place x_1 in the X-register.
[$\Sigma+$]	Ignore	3 is added to the contents of S4 (which was zero) and 8 is added to the contents of S6 (which was zero).
5 [CHS] [↑]	-5.00	Place y_2 in the Y-register.
4	4.	Place x_2 in the X-register
[$\Sigma+$]	Ignore	Add 4 to the contents of S4 (which was 3) and add -5 to the contents of S6 (which was 8).

Press	Display	Comment
[RCL] [Σ+]	[7.00]	The X-register (display) contains the contents of S4, which is $x_1 + x_2$ = 3 + 4 = 7.
[h] [x ⇄ y]	[3.00]	Exchange the contents of the X- and Y-registers. The display shows what was in the Y-register, which are the contents of the S6 register, which $y_1 + y_2$ = 8 - 5 = 3.

Vector Operations

Vector operations are briefly described on pages 118 through 120 of the HP-67 *Owner's Handbook*. Since vector operations are of fundamental importance in tactical analyses, they will be examined at some length here.

EXAMPLE 6. Course-Made-Good with Bearing and Distance. Two Legs.

From point 0, a course of 025° is maintained for 30 n.mi to point A. At point A the course is changed to 160° and the new course is maintained for 50 n.mi to point C. What is the course-made-good from point 0 to point C? (See Figure 3.)

Discussion: The track from point 0 to point A is a vector whose polar components (025°, 30 n.mi) are known. Similarly the track from point A to point B is a vector whose polar components (160°, 50 n.mi) are known. The components of the vector $\overrightarrow{OB}$ are the course-made-good and the distance-made-good. To obtain the vector $\overrightarrow{OB}$, we need to *add* the vector $\overrightarrow{AB}$ to the vector $\overrightarrow{OA}$, or

$$\overrightarrow{OB} = \overrightarrow{OA} + \overrightarrow{AB} \quad .$$

To perform this vector addition, we need to express $\overrightarrow{OA}$ in rectangular coordinates. Then we need to express $\overrightarrow{AB}$ in rectangular coordinates. If we add the x-component of $\overrightarrow{OA}$ to the x-component of $\overrightarrow{AB}$, the result is the x-component of $\overrightarrow{OB}$. Similarly, if we add the y-component of $\overrightarrow{OA}$ to the y-component of $\overrightarrow{AB}$, the result is the y-component of $\overrightarrow{OB}$. Thus the rectangular components of $\overrightarrow{OB}$ will have been obtained. Finally, if we convert the rectangular components of $\overrightarrow{OB}$ to polar components, these polar components are the course-made-good quantities that we want.

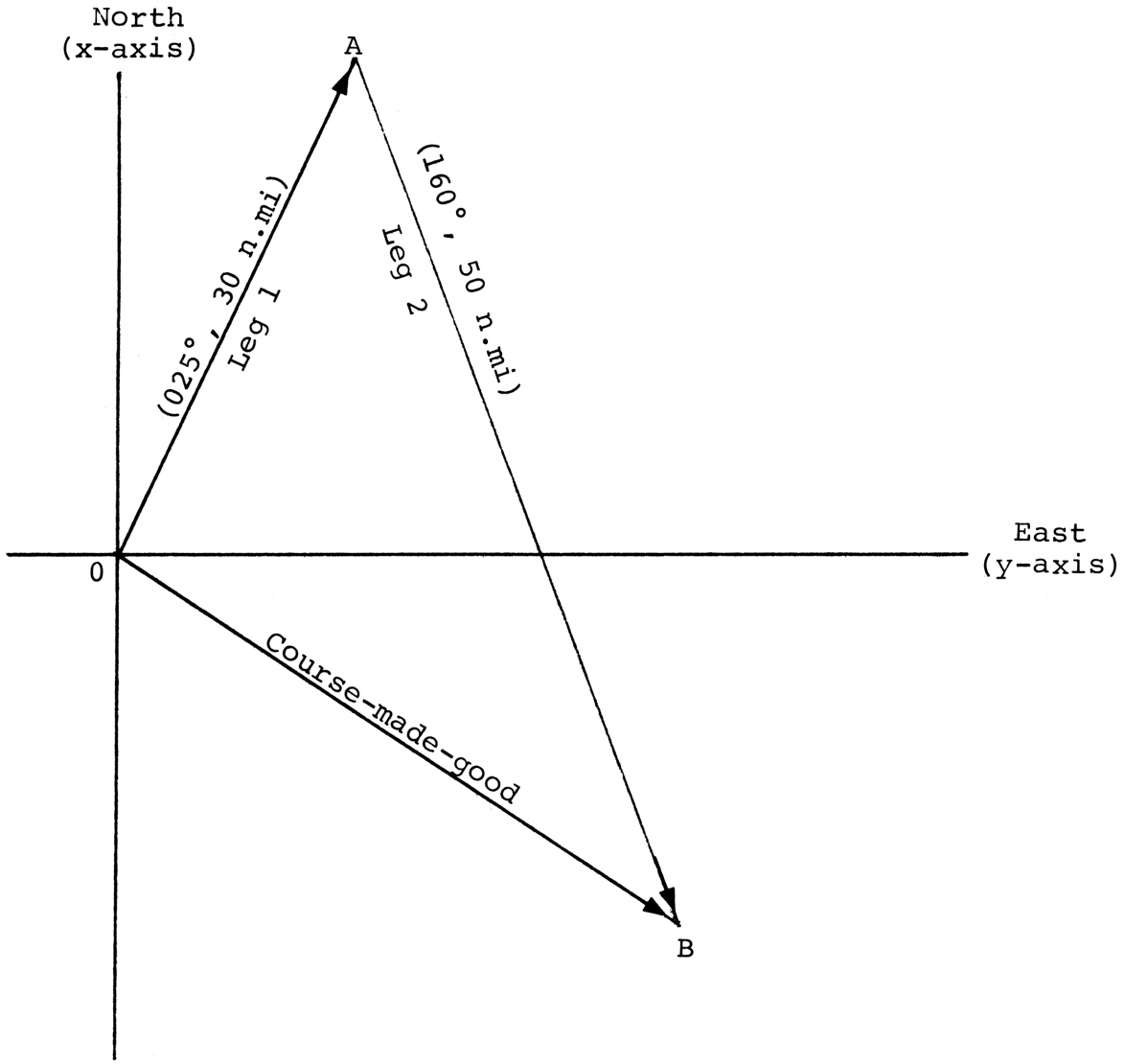

FIGURE 3. Course-Made-Good with Two Legs

Solution:

Press	Display	Comment
[RCL] [Σ+] [h] [Σ-]	[Ignore]	Clear secondary registers S4 and S6.
025 [↑] 30	[30.]	Enter the polar components of of $\overrightarrow{OA}$
[f] [R←]	[27.19]	Convert the polar components of $\overrightarrow{OA}$ to rectangular components.
[Σ+]	[Ignore]	Add the x-component of $\overrightarrow{OA}$ to S4 and add the y-component of $\overrightarrow{OA}$ to S6.
160 [↑] 50	[50.]	Enter the polar components of $\overrightarrow{AB}$.
[f] [R←]	[-46.98]	Convert the polar components of $\overrightarrow{AB}$ to rectangular components.
[Σ+]	[Ignore]	Add the x-component of $\overrightarrow{AB}$ to the x-component of $\overrightarrow{OA}$ in S4 and add the y-component of $\overrightarrow{AB}$ to the y-component of $\overrightarrow{OA}$ in S6. As a result, S4 contains the x-component of $\overrightarrow{OB}$ and S6 contains the y-component of $\overrightarrow{OB}$.
[RCL] [Σ+]	[-19.80]	The contents of S4 and S6 recalled to the X-register and Y-register, respectively. The -19.80 in the display is the x-component of $\overrightarrow{OB}$.
[g] [→P]	[35.76]	Convert the rectangular components of $\overrightarrow{OB}$ to polar coordinates. The distance made good is 35.76 n.mi.
[x ⇄ y]	[123.61]	Exchange the X- and Y-registers. The course-made-good is 123°.61.

Vector additions are not limited to just adding one vector to another. It is possible to find the course-made-good after any number of course changes. The important rule to remember is that

VECTOR ADDITION AND VECTOR SUBTRACTION MUST ALWAYS BE PERFORMED IN RECTANGULAR COORDINATES.

EXAMPLE 7. Course-Made-Good. Four Legs.

Find the course-made-good for the following flight legs (see Figure 4):

$\overrightarrow{OA}$ = (26°.5, 47 n.mi),

$\overrightarrow{AB}$ = (224°, 91 n.mi),

$\overrightarrow{BC}$ = (105°, 77 n.mi),

and $\overrightarrow{CD}$ = (305°, 63 n.mi) .

Solution:

Press	Display	Comment
[RCL] [Σ+] [h] [Σ-]	Ignore	Clear S4 and S6.
26.5 [↑] 47 [f] [R←]	42.06	Enter $\overrightarrow{OA}$ and convert to rectangular.
[Σ+]	Ignore	Accumulate x and y components in S4 and S6.
224 [↑] 91 [f] [R←]	-65.46	Enter $\overrightarrow{AB}$ and convert to rectangular.
[Σ+]	Ignore	Accumulate x and y components.
105 [↑] 77 [f] [R←]	-19.93	Enter $\overrightarrow{BC}$ and convert to rectangular.
[Σ+]	Ignore	Accumulate x and y components.
305 [↑] 63 [f] [R←]	36.14	Enter $\overrightarrow{CD}$ and convert to rectangular.
[Σ+]	Ignore	Accumulate x and y components.

Press	Display	Comment
[RCL] [Σ+]	[-7.19]	Recall rectangular components of $\overrightarrow{OD}$.
[g] [→P]	[20.76]	Convert to polar. Distance-made-good is 20.76 n.mi.
[h] [x ⇄ y]	[-110.27]	Course is negative. And 360°.
360 [+]	[249.73]	Course-made-good is 249°.73 .

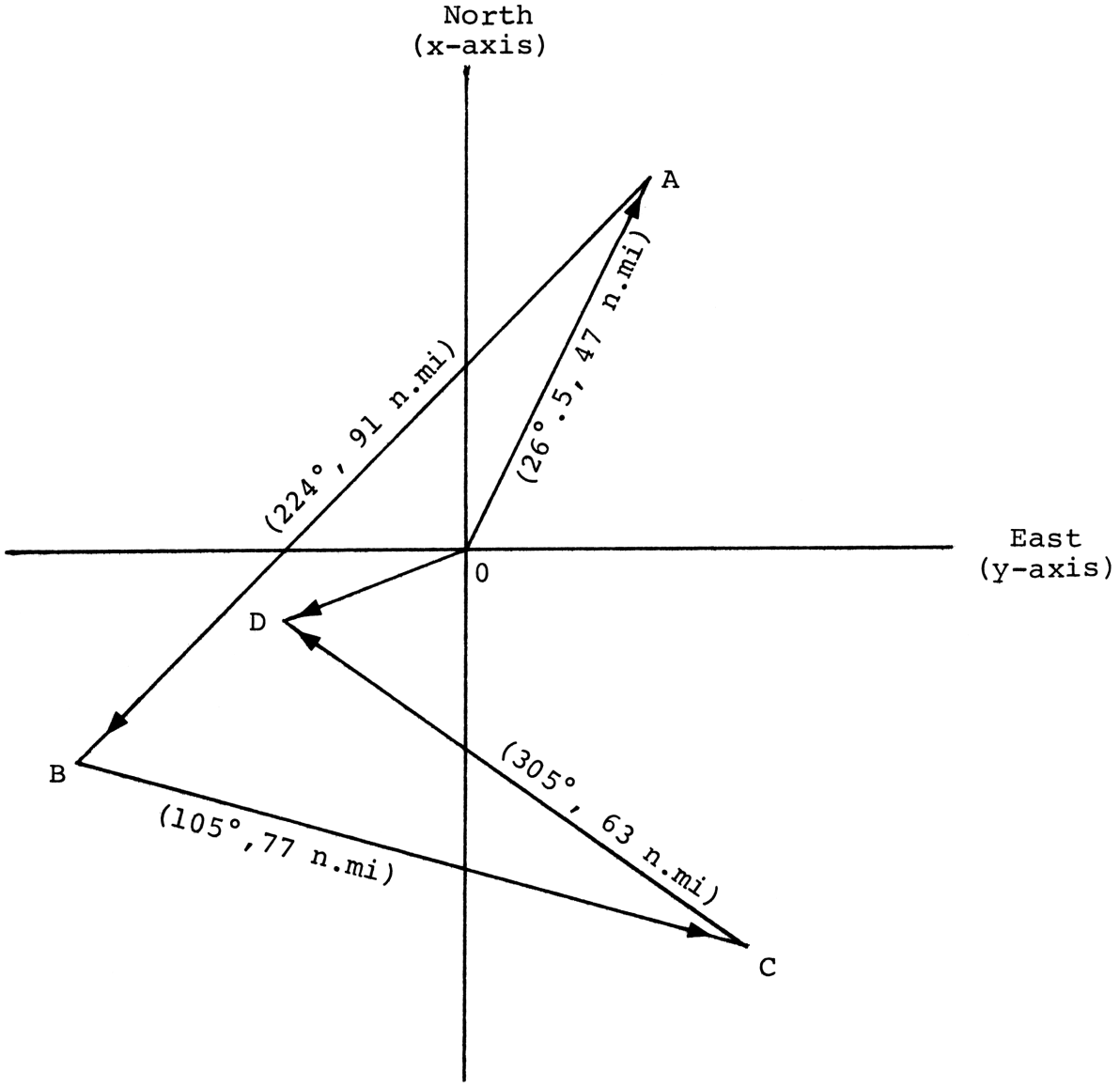

FIGURE 4. Course-Made-Good with Four Legs.

EXAMPLE 8. Course-Made-Good at Each Leg.

Using the data in the previous example, find the course-made-good at the end of each leg of the flight.

Solution:

Press	Display	Comment
RCL Σ+ h Σ-	Ignore	Clear S4 and S6.
26.5 ↑ 47 f R←	42.06	Convert $\overrightarrow{OA}$ to rectangular.
Σ+	Ignore	Accumulate x and y.
224 ↑ 91 f R←	-65.46	Convert $\overrightarrow{AB}$ to rectangular.
Σ+	Ignore	Accumulate x and y.
RCL Σ+	-23.40	Recall rectangular components of $\overrightarrow{OB}$
g →P	48.29	Distance-made-good to point B is 48.29 n.mi.
h x ⇄ y	-118.98	Course is negative. Add 360°.
360 +	241.02	Course-made-good to point B is 241°.02.
105 ↑ 77 f R←	-19.93	Convert $\overrightarrow{BC}$ to rectangular.
Σ+	Ignore	Accumulate x and y.
RCL Σ+	-43.33	Recall components of $\overrightarrow{OC}$.
g →P	53.94	Distance-made-good to point C is 53.94 n.mi.
h x ⇄ y	143.44	Course-made-good to point C is 143°.44.
305 ↑ 63 f R←	36.14	Convert $\overrightarrow{CD}$ to rectangular.

[Σ+]	[Ignore]	Accumulate x and y.
[RCL] [Σ+]	[-7.19]	Recall components of $\overrightarrow{OD}$.
[g] [→P]	[20.76]	Distance-made-good at point D is 20.76 n.mi.
[h] [x ⇄ y]	[-110.27]	Negative. Add 360°.
360 [+]	[249.73]	Course-made-good at point D is 249°.73.

This last example illustrates that it is possible to accumulate and display intermediate results without having to start the problem over to obtain the results at the end of each leg. The rectangular coordinates of the previous course-made-good are contained in secondary registers S4 and S6. This position can be updated in real time as desired.

The next example will illustrate a slight variation on the previous problem. Instead of having courses and distances as direct input we will consider course, ground speed, and time-on-leg as input data. Since time will be entered in minutes or hours and minutes, it may be necessary to review the [g] [→H.MS] and [f] [H←] functions on pages 94 through 96 of the HP-67 *Owner's Handbook*.

EXAMPLE 9. Course-Made-Good with Bearing, Speed, and Time.

Find the course-made-good at the end of the third leg of the following flight (see Figure 5):

$\overrightarrow{OA}$: 045° at 200 knots for 13 minutes.

$\overrightarrow{AB}$: 194° at 170 knots for 26 minutes.

$\overrightarrow{BC}$: 316°.5 at 185 knots for 31 minutes.

Solution:

Press	Display	Comment
[RCL] [Σ+] [h] [Σ-]	Ignore	Clear S4 and S6.
045 [↑] 200 [↑] 0.13 [f] [H←]	0.22	Enter course, speed and time. Convert time to decimal hours.
[x]	43.33	Multiply time times speed. At 200 knots for 13 minutes, 43.33 n.mi have been traveled.
[f] [R←] [Σ+]	Ignore	Convert to rectangular and accumulate x and y.
194 [↑] 170 [↑] 0.26 [f] [H←]	0.43	Enter next course, speed and time. Convert time to decimal hours.
[x]	73.67	Multiple time and speed. Distance traveled on second leg is 73.67 n.mi.
[f] [R←] [Σ+]	Ignore	Convert to rectangular and accumulate x and y.

Press	Display	Comment
316.5 [↑] 185 [↑]		
0.31 [f] [H←]	[0.52]	Enter final course, speed and time. Convert time to decimal hours.
[x]	[95.58]	Multiply time and speed. Distance traveled on third leg is 95.58 n.mi.
[f] [R←] [Σ+]	[Ignore]	Convert to rectangular and accumulate x and y.
[RCL] [Σ+] [g] [→P]	[60.15]	Recall x and y components of $\overrightarrow{OC}$. Convert to polar. Distance-made-good is 60.15 n.mi.
[h] [x ⇄ y]	[-61.72]	Angle is negative. Add 360°.
360 [+]	[298.28]	Course-made-good is 298°.28.

These examples have illustrated several variations of computing course-made-good using the rectangular/polar functions and the summation functions to perform vector addition.

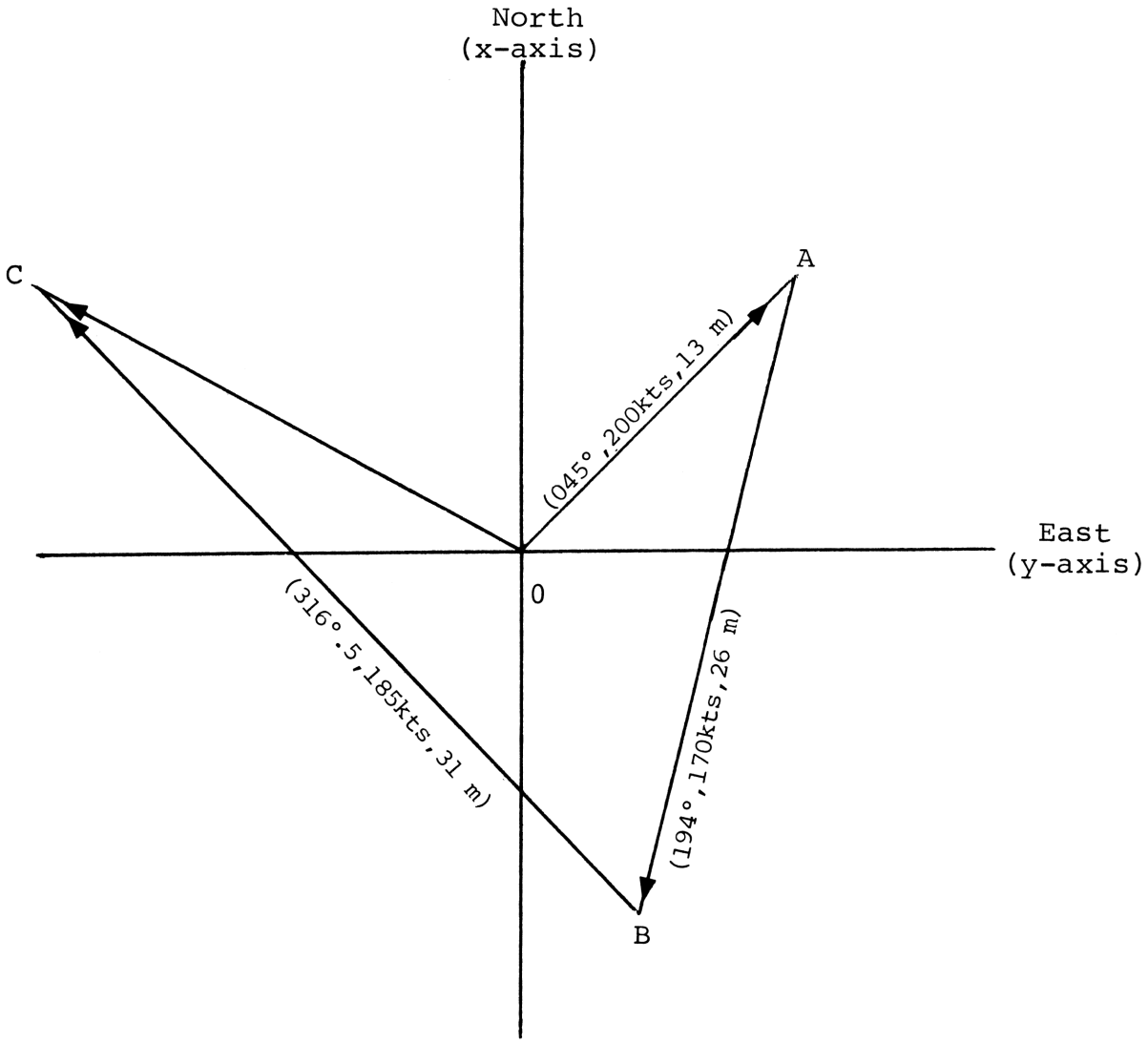

FIGURE 5. Course-Made-Good with Course, Speed and Time.

The next example illustrates the use of the rectangular/polar functions and the summation functions to perform vector subtraction.

EXAMPLE 10. Course and Speed of an Active Sonobuoy Contact.

Using an active sonobuoy, a contact at 213° and 5 n.mi was made at 1420 hours. At 1450 hours the position of the contact was 290° and 7 n.mi. Assuming that the contact has not made a course or speed change, what is the course and speed of the contact?

Discussion: There are several ways in which the solution to this problem may be structured. For this example we will take the easiest approach and assume that only the two contact positions are to be used. If active updating of the contact's course and speed is required with more than two position marks then a more sophisticated solution would allow for real-time updating.

In Figure 6 the target's track is denoted by the vector $\overrightarrow{AB}$. The two fixes are denoted by the vectors $\overrightarrow{OA}$ and $\overrightarrow{OB}$. In the Course-Made-Good with two legs problem (Figure 3), we saw that $\overrightarrow{OB} = \overrightarrow{OA} + \overrightarrow{AB}$. In that problem we knew the components of $\overrightarrow{OA}$ and $\overrightarrow{AB}$ and from those components we calculated the components of $\overrightarrow{OB}$. In Figure 6 the same vector geometry is valid and so again $\overrightarrow{OB} = \overrightarrow{OA} + \overrightarrow{AB}$.

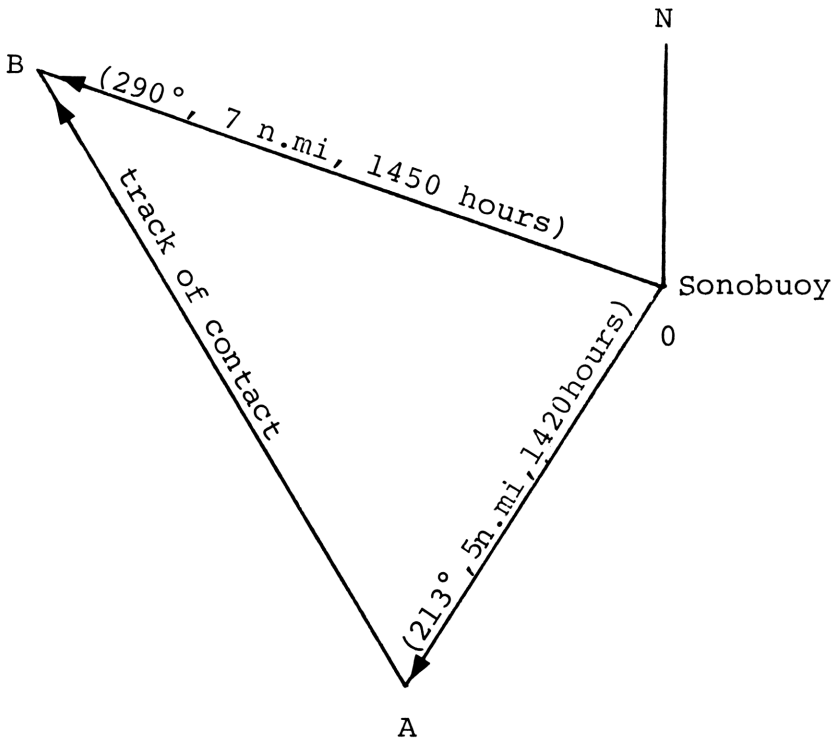

FIGURE 6. Course and Speed of an Active Contact.

However, in this problem we know the components of $\overrightarrow{OA}$ and $\overrightarrow{OB}$, and we wish to calculate the components of $\overrightarrow{AB}$. We can rewrite the vector relationship as

$$\overrightarrow{AB} = \overrightarrow{OB} - \overrightarrow{OA} .$$

To perform this vector subtraction we will express the vector $\overrightarrow{OB}$ in rectangular components. From the rectangular components of $\overrightarrow{OB}$ we will *subtract* the rectanglar components of $\overrightarrow{OA}$. The result will be the rectangular components of $\overrightarrow{AB}$. When these components are converted to polar components we will have the distance traveled by the contact and the course of the contact. We can then divide the distance that the contact has traveled by the time difference between the two fixes to obtain the speed of the contact.

Solution:

Press	Display	Comment
[RCL] [Σ+] [h] [Σ-]	[Ignore]	Clear secondary storage registers S4 and S6
290 [↑] 7	[7.]	Enter the bearing and range at the *second* contact time.
[h] [R←] [Σ+]	[Ignore]	Convert to rectangular and accumulate the x- and y-components of $\overrightarrow{OB}$ in S4 and S6, respectively.
213 [↑] 5	[5.]	Enter the bearing and range at the *first* contact time.

Press	Display	Comment
[h] [R←] [h] [Σ-]	Ignore	Convert $\overrightarrow{OA}$ to rectangular components and *subtract* the x-component of $\overrightarrow{OA}$ from S4 and the y-component of $\overrightarrow{OA}$ from S6. The x- and y-components of $\overrightarrow{AB}$ are now in S4 and S6, respectively.
[RCL] [Σ+] [g] [→P]	7.63	Recall the rectangular complements of $\overrightarrow{AB}$ and convert to polar. The target has traveled 7.63 n.mi.
14.50 [f] [H←]	14.83	Convert the time of the *second* contact to decimal hours.
14.20 [f] [H←]	14.33	Convert the time of the *first* contact to decimal hours.
[-]	0.50	The time between the first and second contacts is 0.5 hours. Note that we have used all of the operational stack. Just prior to the subtraction the polar angle (which we have not yet seen) was in the T-register; the 7.63 n.mi traveled was in the Z-register; the 14.83 was in the Y-register; and the 14.33 was in the X-register. After the subtraction the time between contacts, 0.5 hours, is in the X-register and the 7.63 n.mi traveled is in the Y-register.
[÷]	15.26	This is the contact speed of 15.26 kts; 7.63 n.mi in 0.5 hours is a speed of 15.26 knots.
[h] [x ⇄ y]	-30.33	This is the target course. Since it is negative, add 360°.
360 [+]	329.67	The target course is 329°.67.

III. PROGRAMMING

As stated in the HP-67 *Owner's Handbook* (pg. 124),

> "A *program* is nothing more than a series of calculator keystrokes that you would press to solve a problem manually. The calculator remembers these keystrokes when you key them in, then executes them in order at the press of a single key. If you want to execute a program again and again, you have only to press the single key each time."

You will be shown how programs can be written to perform the operations that were done in the section on vector arithmetic. These programs will be developed and elaborated upon a step at a time. Occasionally a method will be needed that is not in the normal sequence of the presentation in the *Owner's Handbook* if such is the case, then the need will motivate the use of the method and we shall deviate from the sequence of instruction.

Program Memory

The program memory in the HP-67 consists of 224 steps which are numbered 001 to 224, together with a top-of-memory marker which is displayed as step 000. In any HP-67 program, each program step will be equivalent to a *complete instruction*, where a complete instruction may consist of one, two, or three individual keystrokes. For example, the single keystroke [CLx] is a complete instruction which means "clear the X-register." The double keystrokes [STO] [3] and [h] [1/x] are each complete instructions which mean "store the contents of the X-register into primary storage register 3" and "take the reciprocal of the contents of the X-register and display the result in the X-register," respectively. The triple keystroke [STO] [+] [5] is a single instruction which means "add the contents of the X-register to the contents of primary storage register 5." A complete list of all of the program instructions can be found in Appendix E of the HP-67 *Owner's Handbook*.

Now, let us examine the HP-67 calculator. Turn the calculator ON (or if it is already on, turn it OFF and then ON). Slide the Program Mode Switch to the W/PRGM position. The calculator is now in the *Program Mode*. In the left-hand side of the display you will see the three digit program address, in this case the [000] is the top-of-program marker. Now, perform the following operations.

Press	Display	Comment
[SST]	001 84	The 001 denotes program step number 1. The 84 is the keycode (to be discussed later) for R/S
[SST]	002 84	The [SST] button allows you to Single Step through the program memory one program step at a time. Program step 002 also shows the 84 keycode.
[h] [BST]	001 84	The [h] [BST] instruction allows you to Backstep through the program memory. Program step 001 is in the display.
[h] [BST]	224 84	We have backstepped past the top-of-memory to the last program step 224.
[h] [BST]	223 84	Another backstep leads to program step 223.
[SST]	224 84	Single step forward.
[SST]	001 84	Single step forward to program step (often called *location*) 001. What happened to 000?
[GTO] [.] [000]	000	This keystroke sequence has positioned the *program (location) pointer* to the top-of-memory.

Press	Display	Comment
GTO · 035	035 84	This keystroke sequence has placed the program pointer at program step 035. This shows that it is possible to GO TO any desired program step without having to single step.
STO 3	036 33 03	The program pointer was at location 035. The keystrokes STO 3 comprise a *complete instruction* which has been *inserted* into program memory at location 036. The keycode for STO is 33 and the keycode for 3 is 03.

Keycodes

The two-digit keycodes are used to identify the position of the keystrokes that have been used to make a program step. Note that there are eight *rows* of keys in the keyboard and there are either four or five keys in each row. The first digit of the keycode designates the row number (the top row is row 1 and the bottom row is row 8) and the second digit designates the number of the key in the row (counting with one from the *left* side of the keyboard).

For example, 24 designates the second row of keys and the fourth key in that row; thus 24 is the keycode for the key which is labeled (i). The keycode denotes the *key position* and not the *key function*. For example, the keycode 24 could denote the function (i), x ⇄ I, or RND; the actual function depends upon the other keycodes that comprise the complete instruction code. Consider some instructions that involve the 24 key: 34 24, 35 24, and 31 24; the complete instructions involved are [RCL] [(i)], [h] [x ⇄ I], and [f] [RND], respectively. Each *instruction* is unique, and for each keycode sequence there is only one possible meaning, but that meaning must be inferred from the associated keycodes. This may sound complex, but usually the meaning is clear. The keycode 35 can designate only the function [h];

[h] is the *black* prefix key, thus 35 24 designates [h] plus the function in *black* on key 24, which is then [h] [x ⇄ I]. Similarly, 31 can designate only the *gold* prefix key [f], so that 31 24 designates [f] plus the function in *gold* on key 24, which is then [f] [RND].

The only exception to this rule is the designation of the *numbers* on the numeric key. For example, the keycode for [4] is 04. Since there is no zero row, the first digit 0 designates a numeric key, and the second digit designates the numeral on that key. There are four possible instructions involved in the [4] key:

Keycode	Meaning
04	Numeral 4
35 62	[h] [1/x]
31 62	[f] [sin]
32 62	[g] [$\sin^{-1}$]

The numeral 4 is on key 62, but the *keycode* for the numeral 4 is 04 and not 62. However the other three functions that are involved with the 62 key carry the 62 designation. To see this on your calculator, turn the calculator ON (or OFF, then ON) and place the Program Mode Switch in the W/PGRM position. Now press the following key sequences:

Key sequence	Display
[4]	001 04
[h] [1/x]	002 35 62
[f] [sin]	003 31 62
[g] [sin^{-1}]	004 32 62

Additional keycode information may be found on pages 129 and 130 of the HP-67 *Owner's Handbook*. The keycodes for every possible program instruction may be found in Appendix E of the *Owner's Handbook*.

Program Writing, Recording and Execution

In the section on vector operations there were several examples and exercises on finding the course-made-good. In this section we will discuss how to code the basic course-made-good problem. You will create the program code, record the code onto a magnetic program card, and then load the program back into the calculator from the program card. The basic program will not use any fancy program methods; we will use the basic program on a few problems to see how it works and in doing so we will probably find that there are other features that would be "nice-to-have." We will then learn how to modify or "edit" the program, make changes and add new features.

In Examples 6 and 7 we used vector addition to find a course-made-good for a two-leg flight and a four-leg flight, respectively. For each leg of the flight there was a common set of keystrokes that could form the basis for a program. In Example 8 the additional feature of updating the course-made-good at the end of each leg of the flight was introduced. Since this seems to be a desirable feature, we will program the basic set of keystrokes that were used in Example 8. The basic keystrokes are

[f] [R←] [Σ+] [RCL] [Σ+] [g] [→P]

These are not all of the keystrokes that we used, but they at least are the most used keystrokes.

Now, turn your HP-67 ON (or OFF and then ON), and slide the Program Mode Switch into the W/PRGM position and press the key sequence above. Now, review the keycodes to see that they have been properly entered. Proceed as follows:

Press	Display	Comment
[GTO] [·] [000]	000	Set the program pointer to the top-of-memory.
[SST]	001 31 72	Program step 001 is [f] [R↓].
[SST]	002 21	Program step 002 is [Σ+].
[SST]	003 34 21	Program step 003 is [RCL] [Σ+].
[SST]	004 32 72	Program step 004 is [g] [→P].
[SST]	005 84	Program step 005 is [R/S] which we did not key in. It is the end of our program.

Now, find a blank magnetic program card and insert it into the second (lower) slot on the right-hand side of the HP-67. When the blank card has been pushed almost two-thirds of the way into the slot, a slight resistance is encountered. With a slight bit more pressure on the card, the card transport motor turns on and the card automatically continues

into the slot and out of the left-hand side of the calculator. The card can now be removed from the calculator. Since the Program Mode Switch is in the W/PRGM position, the program in the calculator has been *recorded onto* the program card.

Let us experiment further. Turn the calculator OFF and back ON (leave the Program Mode Switch in the W/PRGM position). The display should show 000.

Press	Display
[SST]	001 84
[SST]	002 84
[SST]	003 84
[SST]	004 84
[SST]	005 84

The program memory is filled with the keycode 84 or [R/S] in all locations, and our program is no longer in the calculator. DO NOT RUN THE PROGRAM CARD THROUGH THE CALCULATOR. SINCE THE PROGRAM MODE SWITCH IS IN THE W/PRGM POSITION THE PROGRAM ON THE CARD WOULD BE REWRITTEN AND HENCE DESTROYED.

Slide the Program Mode Switch to the RUN position. The display should show [0.00]. Now, place the recorded program card into the lower slot on the right-hand side of the HP-67 as before. Then remove the card from the left-hand side of the HP-67. Now, slide the Program Mode Switch to the W/PRGM position. The program pointer should be at the top-of-memory 000 position.

Press	Display
[SST]	[001 31 72]
[SST]	[002 21]
[SST]	[003 34 21]
[SST]	[004 32 72]
[SST]	[005 84]

The keycodes indicate that the basic course-made-good program is in the calculator memory.

Now, let us use the program on the data in Example 8.

EXAMPLE 11. Course-Made-Good Program.

Use the program just created with the data in Example 8 (see Figure 4):

$\overrightarrow{OA}$ = (26°.5, 47 n.mi),

$\overrightarrow{AB}$ = (224°, 91 n.mi),

$\overrightarrow{BC}$ = (105°, 77 n.mi),

and $\overrightarrow{CD}$ = (305°, 63 n.mi)

Discussion: As in Example 8 we must clear secondary program register using the key sequence [RCL] [Σ+] [h] [Σ-], and we need to enter the $\overrightarrow{OA}$ data using 26°.5 [↑] 47, but then what do we do? The next keystrokes that we want to use are in the program memory, but how do we use them or get at them? Each time we use the program we must start at the top-of-memory. The keystroke sequence [GTO] [·] [0] [0] [0] will position the program pointer at program step 000 (note, this is true when the Program Mode Switch is either the W/PRGM position or in the RUN position), but this is 5 keystrokes just to get to the top-of-memory, and our entire program is only 7 keystrokes long to begin with. Already we are encountering some undesirable features and we have not yet used the program. Surely there must be some better way.

When the Program Mode Switch is in the RUN position (or, we sometimes say that the calculator is in the RUN mode), we can position the program pointer to the top-of-memory by pressing the keys [h] [RTN]. At least this is only 2 keystrokes instead of 5 needed in [GTO] [·] [0] [0] [0], and the result is the same in either case. Now, we can start the program in the program memory by pressing the [R/S] key. The R/S or RUN/STOP key performs several functions. When the calculator is in the RUN mode, pressing the R/S key will start the program execution at the current program pointer location, or, if a program is already running, pressing the R/S key will stop the execution of the program. Thus, to position the program pointer to the top-of-memory and start the program running we can use the sequence [h] [RTN], [R/S]. We can now proceed with the example.

Solution: Turn ON the calculator and in the RUN mode read in the program card.

Press	Display	Comment
[RCL] [Σ+] [h] [Σ-]	[Ignore]	Clear secondary registers S4 and S6.
26.5 [↑] 47	[47.]	Enter $\overrightarrow{OA}$ the polar components of $\overrightarrow{OA}$.
[h] [RTN] [R/S]	[47.00]	Execute the program. Since only the first leg has been entered the distance-made-good is in the display.
[h] [x ⇄ y]	[26.50]	As a check, we see that the course-made-good on the first leg is 26°.50.
224 [↑] 91	[91.]	Enter $\overrightarrow{AB}$ the polar components of $\overrightarrow{AB}$.
[h] [RTN] [R/S]	[48.29]	Execute the program. The distance-made-good to point B is 48.29 n.mi.
[h] [x ⇄ y]	[-118.98]	Course is negative. Add 360°.
360 [+]	[241.02]	The course-made-good to point B is 241°.02.
105 [↑] 77	[77.]	Enter $\overrightarrow{BC}$ the components of $\overrightarrow{BC}$.
[h] [RTN] [R/S]	[53.94]	Execute the program. The distance-made-good to point C is 53.94 n.mi.

Press	Display	Comment
[h] [x ⇄ y]	[143.44]	The course-made-good to point C is 143°.44.
305 [↑] 63	[63.]	Enter $\overrightarrow{CD}$ the components of $\overrightarrow{CD}$.
[h] [RTN] [R/S]	[20.76]	Execute the program. The distance-made-good to point D is 20.76 n.mi.
[h] [x ⇄ y]	[-110.27]	The course is negative. Add 360°.
360 [+]	[249.73]	The course-made-good to point D is 249°.73.

Although this program leaves a lot to be desired, its use is somewhat simpler than the manual solution that was used in Example 8. We shall return to this problem to modify and streamline our program, but first we will discuss program editing.

Program Editing and An Improved Course-Made-Good Program

In working through Example 11, several awkward sequences could be observed. For one, the keystroke sequence [h] [RTN] [R/S] is cumbersome. For another, the initialization sequence [RCL] [Σ+] [h] [Σ-] is bothersome. And for yet another, there must be something that can be done to convert negative courses to positive courses easily, or at least more conveniently than manually adding 360°. Most but not all of these handicaps will be overcome in this section.

The beginning and ending of a program are discussed briefly on pages 133 to 134 of the *Owner's Handbook*. The suggestion is made that a program should start with a LABEL A, B, C, D, E, a, b, c, d, or e. These labels are sometimes referred to as "User Defined Labels." There are 10 such user defined labels, and they can be created as a program step using one of the sequences:

[f] [LBL] [A]	31 25 11
[f] [LBL] [B]	31 25 12
⋮	⋮
[f] [LBL] [E]	31 25 15
[g] [LBLf] [a]	32 25 11
[g] [LBLf] [b]	32 25 12
⋮	⋮
[g] [LBLf] [e]	32 25 15

The fact that there are 10 user defined labels means that the user may have as many as 10 programs, subprograms, or program segments in the program memory at one time. (It is possible to create more than 10 user defined labels, but that is beyond the scope of the present discussion.)

A program segment (created in the Program Mode) with a user defined label will for example have the following structure:

```
[f] [LBL] [A]          31 25 11
     :                    :
[h] [RTN]              32 22
or   [R/S]        or      84.
```

The program segment starts with a user defined label and ends with either [h] [RTN] or [R/S]. The dots in between indicate a user defined program such as the four step program that we generated in the previous section. To execute the program section while in the RUN Mode, the user must press only the key with the user label. In the program segment above pressing [A] is equivalent to the sequence [GTO] [A] [R/S]. In other words, when [A] is pressed, the calculator will search for the program step

that contains [f] [LBL] [A], and when that program step is found, the program will start execution at the *next* program step, and execution will continue until either the instruction [h] [RTN] or the instruction [R/S] is encountered.

At the beginning of this section it was mentioned that the course-made-good program had three awkward coding sequences: the initialization sequence [RCL] [Σ+] [h] [Σ-]; the execution sequence [h] [RTN] [R/S]; and the negative course or bearing sequence 360 [+]. This suggests that we might make these three sequences into three program segments, each with a distinct user defined label. In fact, we could write the program segments as follows:

Program Instruction	Comments
[f] [LBL] [A]	User defined label "A".
[f] [R←]	Program sequence to add vectors as shown in the previous section.
[Σ+]	
[RCL] [Σ+]	
[g] [→P]	
[h] [RTN]	End of program segment "A".
[f] [LBL] [E]	User defined label "E".
[3]	A new program segment to add 360°.
[6]	
[0]	
[+]	
[h] [RTN]	End of program segment "E".

Program Instruction	Comments
[g] [LBLf] [c]	User defined label "c".
[RCL] [Σ+]	A new program segment to clear secondary registers S4 and S6.
[h] [Σ-]	
[h] [RTN]	End of program segment "c".

The skeleton outline of how to use these segments is the following:

1. Clear S4 and S6

[f] [c]

2. Enter bearing and distance

[A]

3. Distance-made-good is in the display.

4. Display the course-made-good

[h] [x ⇄ y]

5. If the course-made-good is negative, add 360°

[E] .

6. Read the course-made-good from the display.

Since the use of labels allows the program to be operated with a minimum of keystrokes, the reader may wonder why the sequence [f] [c] is used at step 1 ([g] [LBLf] [c]) instead of the shorter [C] that would be required if [f] [LBL] [C] were used instead. This is a program technique

that is used to insure that a portion of the storage, in this case secondary memory registers S4 and S6, is not accidently erased by pressing the wrong key by mistake. With the sequence [f] [c], two keystrokes are required to clear S4 and S6, and the chance of accidental erasure is much less than if the single stroke [C] had been used.

Since the proposed new program is short, we could just power-up the calculator and key it all in at one time. However, we will use the original program to illustrate some of the *editing* features of the HP-67.

A program might require editing for one of several reasons:

1. A program step is incorrect and should be *deleted.*
2. A program step has been omitted and should be *inserted.*
3. A program step is incorrect and needs to be *replaced.*

To *delete* a step, the program pointer must be pointing to the step to be deleted. Usually the program pointer is placed at the appropriate step using the [GTO] [•] [XXX] command, where XXX is the program step to be deleted. For example in our original course-made-good program, suppose at step 002 the instruction [Σ+] is to be deleted. The command [GTO] [•] [002] in either the RUN mode or the PROGRAM mode will position the pointer at step 002.

Then with the calculator in the PROGRAM mode (the Program Mode Switch is in the W/PRGM position) the keystrokes [h] [DEL] will cause program step 002 to be deleted.

Let us do this with our program. Turn the calculator ON, place the Program Mode Switch in the RUN position, and read the magnetic card with the course-made-good program into the calculator. Now let us review the program. Place the Program Mode Switch in the W/PRGM position and do the following steps:

Press	Display	Instruction
[SST]	001 31 72	[f] [R←]
[SST]	002 21	[Σ+]
[SST]	003 34 21	[RCL] [Σ+]
[SST]	004 32 72	[g] [→P]
[SST]	005 84	[R/S]

Now, if we want to delete step 002, then

Press	Display	Instruction
[GTO] [·] [002]	002 21	[Σ+]
[h] [DEL]	001 31 72	[f] [R←]
[SST]	002 34 21	[RCL] [Σ+]
[SST]	003 32 72	[g] [→P]
[SST]	004 84	[R/S]

The [GTO] [.] [002] instruction places the pointer at program step 002. The keystrokes [h] [DEL] delete program step 002 and the pointer location is decreased by one and points to step 001. The following [SST] instructions reveal that the old step 003 is now at step 002, the old step 004 is now at step 003, and the old step 005 is now at step 004. In other words, when step 002 was deleted, all of the step numbers of the instructions with step numbers larger than 002 were decreased by one so that there was no "empty" step number left where the deleted step had been.

Now, let us repair the program and put the [Σ+] instruction back at location 002. That is, we want to *insert* a [Σ+] instruction between the steps 001 and 002 that are now in the calculator. To do this, place the program pointer of the step just above the location where the instruction is to be inserted. Since we want to insert a new 002, place the program pointer of step 001, and then press the keystroke(s) that are to be inserted.

Press	Display	Instruction
[GTO] [.] [001]	001 31 72	[f] [R←]
[Σ+]	002 21	[Σ+]
[SST]	003 34 21	[RCL] [Σ+]
[SST]	004 32 72	[g] [→P]
[SST]	005 84	[R/S]

The [GTO] [·] [001] instruction placed the program pointer at location 001. The keystroke [Σ+] was placed at step 002. NOTE THAT THE PROGRAM STEP NUMBERS OF ALL OF THE FOLLOWING STEPS WERE AUTOMATICALLY INCREASED BY ONE. NO PROGRAM STEPS WERE DESTROYED BY BEING OVER-WRITTEN BY THE NEW KEYSTROKES. It is important to note that no program steps can be accidently destroyed by being overwritten for there is no way on the HP-67 to over-write another instruction; if an instruction is to be destroyed it must intentionally be deleted using the [h] [DEL] instruction.

Finally, suppose that we want to *replace* the [Σ+] instruction at step 002 with the [h] [Σ-] instruction. Since we cannot intentionally overwrite the [Σ+] instruction we must first *delete* the [Σ+] instruction and then *insert* the [h] [Σ-] instruction.

Press	Display	Instruction
[GTO] [·] [002]	002 21	Σ+
[h] [DEL]	001 31 72	[f] [R←]
[h] [Σ-]	002 35 21	[h] [Σ-]

To review

Press	Display	Instruction
GTO · 000	000	Top-of-memory
SST	001 31 72	f R←
SST	002 35 21	h Σ-
SST	003 34 21	RCL Σ+
SST	004 32 72	g →P
SST	005 84	R/S

Observe that the replacement of Σ+ with h Σ- was quite easy. The GTO · 002 instructions placed the pointer at step 002. The h DEL instruction deleted step 002 and placed the pointer at step 001, *exactly* where it should be to insert a new instruction such as h Σ- of step 002.

Let us now modify the course-made-good program by writing the program segments that we discussed earlier. One way that this can be done is shown below. First, read in the program card. Then slide the Program Mode Switch into the W/PRGM position; the program pointer should be at the top of memory. Then,

Press	Display	Instruction
f LBL A	001 31 25 11	f LBL A
SST	002 31 72	f R←
SST	003 21	Σ+
SST	004 34 21	RCL Σ+
SST	005 32 72	g →P
SST	006 84	R/S
h BST	005 32 72	g →P
h RTN	006 35 22	h RTN
f LBL E	007 31 25 15	f LBL E
3	008 03	3
6	009 06	6
0	010 00	0
+	011 61	+
h RTN	012 35 22	h RTN
g LBLf c	013 32 25 13	g LBLf c
RCL Σ+	014 34 21	RCL Σ+
h Σ-	015 35 21	h Σ-
h RTN	016 35 22	h RTN

While the calculator is in the PROGRAM Mode, record the program onto the magnetic card. If you pass the same card through the calculator, the new program will be written on top of the old program. Place the Program Mode Switch in the RUN position and use the program on the data in Examples 8 and 11.

EXAMPLE 12. Edit Course-Made-Good Program.

Use the program just created with the data in Examples 8 and 11:

$\overrightarrow{OA}$ = (26°.5, 47 n.mi)

$\overrightarrow{AB}$ = (224°, 91 n.mi)

$\overrightarrow{BC}$ = (105°, 77 n.mi)

and $\overrightarrow{CD}$ = (305°, 63 n.mi)

Solution:

Press	Display	Comment
[f] [c]	Ignore	Clear secondary registers S4 and S6.
26.5 [↑] 47	47.	Enter the polar coordinates of $\overrightarrow{OA}$.
[A]	47.00	Execute the program. Since only the first leg has been entered the distance-made-good is in the display.
224 [↑] 91	91.	Enter the polar components of $\overrightarrow{AB}$
[A]	48.29	Execute the program. The distance-made-good to point B is 48.29 n.mi.
[h] [x ⇄ y]	-118.98	Course is negative. Add 360°.
[E]	241.02	The course-made-good to point B is 241°.02.
105 [↑] 77	77.	Enter the components of $\overrightarrow{BC}$.

Press	Display	Comment
[A]	[53.94]	Execute the program. The distance-made-good to point C is 53.94 n.mi.
[h] [x ⇄ y]	[143.44]	The course-made-good to point C is 143°.44.
305 [↑] 63	[63.]	Enter the components of $\overrightarrow{CD}$.
[A]	[20.76]	Execute the program. The distance-made-good to point D is 20.76 n.mi.
[h] [x ⇄ y]	[-110.27]	The course is negative. Add 360°.
[E]	[249.73]	The course-made-good to point D is 249°.73.

This version of the course-made-good program is considerably easier to use than the earlier version, and should illustrate adequately the utility of the user defined labels. It is possible to make further improvements, but first we need to learn more of the HP-67 architecture.

EXAMPLE 13. Horizontal Range to the First Convergence Zone.

The horizontal range R_h to the first convergence zone is a function of the surface or reference sound velocity V_r. The function is [References 1 and 2]:

$$R_h = -2264 + 0.8539V_r - 7.891 \times 10^{-5} V_r^2 ,$$

where the horizontal range is in nautical miles and the sound velocity V_r is measured in feet/second. Write a program to compute R_h for any value of V_r.

Solution 1.

Since V_r must be used twice in the evaluation of R_h it should be stored in some storage register for later use. Let V_r be stored in R0 and let the user define label [A] be used both to store V_r and to start the program execution.

PROGRAM MODE:

Program Instruction	Program Step	Keycode	Comment
f LBL A	001	31 25 11	User defined key to input and run.
STO 0	002	33 00	Store V_r in primary storage register R0.
.	003	83	V_r remains in the x-register after storage. Multiply V_r by 0.8539 to evaluate the second term in the equation.
8	004	08	
5	005	05	
3	006	03	
9	007	09	
x	008	71	Multiply. $0.8539V_r$ is in the x-register.
2	009	02	Subtract 2264.
2	010	02	
6	011	06	
4	012	04	
-	013	51	
7	014	07	Set up the constant 7.891×10^{-5}
.	015	83	
8	016	08	
9	017	09	
1	018	01	
EEX	019	43	
5	020	05	
CHS	021	42	

Program Instruction	Program Step	Keycode	Comment
RCL 0	022	34 00	Recall V_r from R0.
g x^2	023	32 54	Compute V_r^2.
x	024	71	Multiply V_r^2 times 7.891×10^{-5}.
-	025	51	Subtract the result from the value in the Y-register.
DSP 1	026	23 01	Set the display for one figure after the decimal.
h RTN	027	35 22	End of program segment.

This program illustrates that constants can be made part of the calculator program by keying the digits of the constants in the same way that any other instruction is keyed. Long or many constants can rapidly increase the number of program steps since each *digit* of the constant requires a separate program step.

Now, if the sound velocity of the reference depth is 4962 feet/second, use the program to compute the range to the first CZ.

RUN MODE:

Press	Display	Comment
4962	[4962.]	Key in the reference sound velocity.
[A]	[30.2]	Compute. The range to the first CZ is 30.2 n.mi.

Solution 2.

The equation

$$R_h = -2264 + 0.8539V_r - 7.891 \times 10^{-5}V_r^2$$

is said to be a second order polynomial in V_r. Often space can be saved and accuracy increased by evaluating polynomials in a *nested* form. The equation

$$R_h = (-7.891 \times 10^{-5}V_r + 0.8539)V_r - 2264$$

is called a *nested polynomial*. This equation is the same as the original equation except that it has been rewritten. It is important to observe that the only operations that are used in nested polynomials are addition, subtraction, and multiplication; we do not have to square the quantity V_r.

For third, fourth, and higher order polynomials the saving in computational time and space can be quite dramatic; but for our second order polynomial, we only break even. This equation can be coded in the following way.

PROGRAM MODE.

Program Instruction	Program Step	Keycodes	Comments
f LBL A	001	31 25 11	User defined label.
STO 0	002	33 00	Store V_r in R0.
7	003	07	Set up the constant -7.891×10^{-5}.
.	004	83	
8	005	08	
9	006	09	
1	007	01	
CHS	008	42	
EEX	009	43	
5	010	05	
CHS	011	42	
x	012	71	Multiply by V_r in the Y-register.

Program Instruction	Program Step	Keycodes	Comments
.	013	83	Set up the constant 0.8539
8	014	08	
5	015	05	
3	016	03	
9	017	09	
+	018	61	Add.
RCL 0	019	34 00	Recall V_r and multiply.
x	020	71	
2	021	02	Set up the constant 2264.
2	022	02	
6	023	06	
4	024	04	
-	025	51	Subtract
DSP 1	026	23 01	Set the display.
h RTN	027	35 22	End of program segment.

This second program is used in exactly the same way that the first program is used.

IV. BRANCHING

The [GTO] instruction is used to perform an *unconditional branch* (or an *unconditional transfer*) to some specified location in the program memory. In effect, it changes the position of the program pointer to a new program step number (which is not generally the next sequential program step number). For example, if a program is running and the instruction [GTO] [A] is encountered, the calculator will stop executing program instructions, and will start a search for the instruction [LBL] [A] by incrementing the program pointer (program step 001 *follows* program step 224) until either the instruction [LBL] [A] is found or until the program pointer has returned to the [GTO] [A] instruction. If the instruction [LBL] [A] is found, then calculator will resume execution program instructions with the instruction immediately following the [LBL] [A] instruction; in effect, all of instructions between the [GTO] [A] instruction and the [LBL] [A] instruction are skipped. If the program pointer returns to its original position without finding a [LBL] [A] instruction then the display will show [error].

Usually the unconditional transfer is of little use by itself for seldom are programs written that jump from one portion of program memory to another. When a jump is made it is usually because of the outcome of a *conditional test*. For example, in the course-made-good problem we manually pressed [E] to jump to [LBL] [E] and execute the program segment that added 360° to a negative course. Visually we *tested* the contents of the X-register (display) for a negative value and if the value was negative we then jumped (or branched) unconditionally to [LBL] [A] to continue execution. Testing for a negative value is one of eight conditional tests that can be performed by the HP-67 calculator upon the value in the X-register. The eight tests are discussed on pages 185 through 192 of the *Owner's Handbook*.

Four of the conditional tests are used to compare the value of the number in the X-register to the value of the number in the Y-register. These tests are used to ask one of the questions:

Question	Instruction
Is x = y?	[g] [x = y]
Is x ≠ y?	[g] [x ≠ y]
Is x ≤ y?	[g] [x ≤ y]
Is x > y?	[g] [x > y]

The other four tests are used to compare the value in the X-register to zero. These tests are used to ask one of the questions:

Question	Instruction
Is x = 0?	[f] [x = 0]
Is x ≠ 0?	[f] [x ≠ 0]
Is x < 0?	[f] [x < 0]
Is x > 0?	[f] [x > 0]

These conditional tests operate in the following manner. If the answer to the question is YES, then the program pointer is incremented by *one* and the instruction at that address is executed. If the answer to the question is NO, then the program pointer is incremented by *two* and the instruction at that address is executed.

In other words, if the answer is YES, the *next* instruction is executed, but if the answer is NO, the *next* instruction is skipped and the one following that is executed.

The conditional test together with the unconditional transfer add a tremendous amount of power and capability to the HP-67. The usefulness of this feature can be most readily illustrated with the course-made-good problem. In addition we will introduce the [f] [-x-] instruction (*Owner's Handbook*, page 172) which will be used to display the distance-made-good result for five seconds. The program is listed below:

Instruction	Step	Keycode
f LBL A	001	31 25 11
f R↓	002	31 72
Σ+	003	21
RCL Σ+	004	34 21
g →P	005	32 72
f -x-	006	31 84
h x ⇄ y	007	35 52
f x < 0	008	31 71
GTO E	009	22 15
h RTN	010	35 22

Instruction	Step	Keycode
f LBL E	011	31 25 15
3	012	03
6	013	06
0	014	00
+	015	61
h RTN	016	35 22
g LBLf c	017	32 25 13
RCL Σ+	018	34 21
h Σ+	019	35 21
h RTN	020	35 22

This program may be generated by editing the earlier course-made-good program. When the keycodes have been verified the new program should be recorded onto a magnetic card. The use of this program is illustrated in Example 14.

EXAMPLE 14. Course-Made-Good Program Revised.

Use the program just created with the data in Examples 8, 11, and 12:

$\overrightarrow{OA}$ = (26°.5, 47 n.mi)

$\overrightarrow{AB}$ = (224°, 91 n.mi)

$\overrightarrow{BC}$ = (105°, 77 n.mi)

and $\overrightarrow{CD}$ = (305°, 63 n.mi).

Solution.

Press	Display	Comment
[f] [c]	[Ignore]	Clear secondary registers S4 and S6.
26.5 [↑] 47	[47.]	Enter the polar coordinates $\overrightarrow{OA}$.
[A]	[47.00]	Execute the program. Since only the first leg has been entered, the distance-made-good (47 n.mi) flashes in the display for five seconds. Then the course-made-good (26°.5) remains in the display.
	[26.50]	
224 [↑] 91	[91.]	Enter the polar components of $\overrightarrow{AB}$.
[A]	[48.29]	Execute the program. The distance-made-good (48.29 n.mi) is flashed and the course-made-good (241°.02) remains in the display.
	[241.02]	
105 [↑] 77	[77.]	Enter the components of $\overrightarrow{BC}$.

Press	Display	Comment
[A]	[53.94]	The distance-made-good is 53.94 n.mi.
	[143.44]	The course-made-good is 143°.44.
305 [↑] 63	[63.]	Enter the components of $\overrightarrow{CD}$.
[A]	[20.76]	The distance-made-good is 29.76 n.mi.
	[249.73]	The course-made-good is 249°.73.

There is very little that can be done to improve this version of the course-made-good program. The only problem that might arise is that in real time operation the user's attention could be momentarily diverted and the flashing distance-made-good display could be missed. In this particular program the distance-made-good is in the Y-register while the course-made-good is in the X-register (display). Consequently the distance-made-good can be recovered by pressing [h] [x ⇄ y] after the course-made-good has been recorded. A second alternative is to replace the instruction [f] [-x-] at step 006 with the instruction [R/S]. If this is done then the program will stop execution of step 006 with the distance-made-good in the display. To resume execution the key [R/S] is pressed manually; the execution then stops a second time with the course-made-good in the display. This use of the [R/S] key is discussed on pages 169 to 172 of the *Owner's Handbook*.

EXAMPLE 15. Width of First Convergence Zone Annulus.

The width of the first convergence zone annulus is a function of the depth excess. A formula for annulus width is [Reference 2]

$$w_a = 0.003\ d_e + 1.25 \quad \text{for} \quad 0 \leq d_e \leq 500\ ,$$

and

$$w_a = 0.006\ d_e - 0.25 \quad \text{for} \quad 500 \leq d_e$$

where d_e is the depth excess measured in fathoms and w_a is the annulus width in nautical miles. Write a program which will use the proper formula for any value of $d_e \geq 0$ and will display [error] for a value of $d_e < 0$.

Discussion: There are several conditional tests that we must perform. First, if $d_e < 0$, then generate an error message, but if $d_2 \geq 0$, then calculate the annulus width. In an earlier section we observed that if we use [GTO] (a label) and if the calculator cannot find the address of that label, then an error message is generated. This is the quickest way to generate an error message: [GTO] an undefined label. If $d_e \geq 0$, then the next test is to determine if d_e is larger or smaller than 500. This can be done by using one of the conditional tests in which the value of

the quantity in the X-register is compared to the value of the quantity in the Y-register. Observe that both formulas give the same value of w_a when $d_e = 500$ so either formula can be used in that case.

Solution.

PROGRAM MODE.

Program Instruction	Program Step	Keycodes	Comment
f LBL A	001	31 25 11	User defined input and run label.
f x < 0	002	31 71	Is $d_e < 0$?
GTO 0	003	22 00	If YES, generate an error message.
5	004	05	If NO, then $d_3 \geq 0$. Enter the constant 500.
0	005	00	
0	006	00	
h x ≷ y	007	35 52	Put 500 in the Y-register and d_e in the X-register.
g x ≤ y	008	32 71	Is $d_e \leq 500$?
GTO 9	009	22 09	If YES, go to [LBL] [9].
.	010	83	If NO, then $d_e > 500$. Enter the constant 0.006.
0	011	00	
0	012	00	
6	013	06	

Program Instruction	Program Step	Keycodes	Comment
x	014	71	Multiply 0.006 and d_e.
.	015	83	Subtact 0.25
2	016	02	
5	017	05	
-	018	51	
h RTN	019	35 22	End of program segment. Display w_a.
f LBL 9	020	31 25 09	Segment to evaluate w_a when $d_e \leq 500$.
.	021	83	Enter the constant 0.003.
0	022	00	
0	023	00	
3	024	03	
x	025	71	Multiply 0.003 and d_e.
1	026	01	Add 1.25
.	027	83	
2	028	02	
5	029	05	
+	030	61	
h RTN	031	35 22	End of program segment. Display w_a.

Additional Comments.

At step 002 the value of d_e in the X-register is tested. If $d_e < 0$ then step 003 is executed and a transfer is made to [LBL] [0]. Since [LBL] [0] is not defined in the program the calculator cannot find a program step containing [LBL] [0] and so an error message is generated.

At step 007 d_e is placed in the X-register and 500 is placed in the Y-register. The test could have been done without interchanging the contents of the X- and Y-registers, but this would have left 500 in the X-register after the test was performed. Consequently an instruction would be required to get d_e back into the X-register in each leg of the branch, and this would require one more [h] [x ⇄ y] instruction than was needed by making the exchange before the test.

Sample Problems and Solutions

1. d_e = 200 fathoms. w_a = 1.9 mi.
2. d_e = 400 fathoms. w_a = 2.5 mi.
3. d_e = 500 fathoms. w_a = 2.8 mi.
4. d_e = 1000 fathoms. w_a = 5.8 mi.
5. d_e = -200 fathoms. Error.

V. SUBROUTINES

In writing programs we occasionally have the need to execute a fixed series of instructions at two or more different portions of the program. For example, if we were constructing a program in which estimates of target bearing, target course, and course-to-close are calculated, we would need to use the "add 360°-routine" in three separate places in our program. It would be convenient and would save program steps if we could use the *same* "add 360°-routine" at three different locations in the program and then continue with other calculations. The way to accomplish this is by using a subroutine (*Owner's Handbook*, pages 197-211).

A subroutine is a portion of code that *starts* with a *label* and *ends* with a *return*. When a subroutine is used, a special type of a branching instruction (GSB) will move the program pointer to the subroutine label, and at the same time a second (return) pointer is positioned at the program step immediately following the location of the GSB instruction. The program pointer will then move sequentially through the program steps and execute the instructions following the subroutine label (provided no additional branching instructions are encountered) until the return (h RTN) instruction is encountered. When the return instruction is found, the program pointer is repositioned to

coincide with the position of the return pointer. Execution then continues in the usual manner.

A subroutine starts with one of the 20 labels:

[f] [LBL] [A], ... , [f] [LBL] [E],
[g] [LBLf] [a], ... , [g] [LBLf] [e],
[f] [LBL] [0], ... , [f] [LBL] [9]

and ends with

[h] [RTN].

The "branch to the subroutine and return" instruction is the GSB instruction. We can cause a subroutine to be executed by using one of the instructions

[f] [GSB] [A], ... , [f] [GSB] [E],
[g] [GSBf] [a], ... , [g] [GSBf] [e],
[f] [GSB] [0], ... , [f] [GSB] [9],

where the letter or number corresponds to the subroutine which we wish to execute.

GSB

GSBf

Just before Example 14 is a list of instructions for the course-made-good program. If the instruction at step 009 (GTO E 22 15) is replaced by the instruction f GSB E (31 22 15), then the program at LBL E will be executed as a subroutine. You will not notice any difference in the execution of the program, but there is indeed a difference.

When course and distance information have been input and key [A] has been pressed, the program will pause at step 006 with the distance-made-good flashing in the display. Execution will continue then at step 007 in which the contents of the X-register and Y-register are exchanged. The course-made-good is now in the display, but execution immediately proceeds to step 008 where the value of the contents of the X-register is compared to zero. If the contents of the X-register are positive or zero. step 009 is skipped and the program stops at step 010 with the value of the course-made-good in the display. If the contents of the X-register are negative when the conditional test at step 008 is made, then the instruction at step 009 is executed. Since this is a GSB instruction, a *return pointer* is set to the next program step (010) and the *program pointer* is moved until label E is found at

step 011. Execution then starts at step 012 where 360 is formed in the X-register and is added at step 015. At step 016 an h RTN instruction is found; the *return pointer* is examined and is found to be pointing at step 010, so the *program pointer* is then set to step 010 and that instruction is executed, which stops the program execution.

In the earlier program, the RTN at step 016 stopped execution, but in the program above, the RTN at step 016 caused a branch back to step 010 where execution stopped. This multiple use of the RTN instruction should be reviewed on page 134 and pages 197 to 198 of the *Owner's Handbook*.

Even though there is little apparent difference in the way the last two course-made-good programs operated, there is indeed a fundamental difference. The first program terminated at step 010 if the course-made-good was positive or zero and it terminated at step 016 if the course-made-good was negative. Had there been additional computations between steps 010 and 011, then it would have been awkward to execute them if the halt occurred at step 016. The second program terminates at step 010 whether the course-made-good was positive, zero, or negative and so additional computation could immediately follow step 016. This feature will become more obvious in the next section.

VI. PACKING THE OUTPUT DISPLAY

For some purposes it is convenient to display two computed quantities simultaneously rather than sequentially so that a user can read two pieces of information at a glance. In the course-made-good problem, for example, the distance-made-good was flashed for about five seconds before the course-made-good was displayed. A fatigued operator might miss the information in the flashing display and have no way to recover it conveniently, whereas if the course-made-good and the distance-made-good were displayed simultaneously, nothing would be lost and considerable convenience might be gained.

We will display the distance and course in *packed* form. The format that we will use will be ddd.ccc, where the d's denote distance information and the c's denote course information; these two pieces of information are separated by a decimal point. The distance information might require one to five digits (if distance were in yards or in miles) but courses always require only three digits, so the decision has been made to display the course information *after* the decimal point where we can control the digits in a consistent form. It is assumed that we are only interested in course to the nearest degree and the distance to the nearest unit (yard or mile).

The special functions that we will need to perform the packing operation are display control DSP (*Owner's Handbook*, pages 42-49) and the round RND function (pages 85-86). Conceptually, the packing operation is quite simple, we will convert the bearing to a fraction by dividing by 1000, and then we will add this fractionally formatted bearing to the distance. Before the addition can take place we must guarantee that the fractional part of the range is converted to zeros. This conversion is most easily accomplished using the INT function (page 86), but unfortunately for this application the INT function truncates without rounding. For example, key in 23.85 and press f INT; the result is 23.00 and not 24.00. Once more key in 23.85 and then DSP 0; the result is 24. in the display. Now press f INT and the truncated result is 23. With the display still set at DSP 0, key in 23.85 once more and then press the ENTER key; the result in the display is 24. Now, press the keys f RND and observe that the display is unchanged. Change the display by pressing DSP 2 and observe that the contents of the display are now 24.00. The RND function will leave the contents of the numbers visible in the display unchanged, but will replace all of the *undisplayed* numbers by zeros.

We are now able to pack the distance and the bearing into a single display. We proceed as follows:

1. With the distance in the display, change the display mode to DSP 0.
2. Round (RND) the distance to the nearest unit.
3. Place the rounded distance in the Y-register and place the course in the X-register.
4. Divide the course by 1000. (We use the keys [EEX] [3] to obtain 1000 since it is two keystrokes less than [1] [0] [0] [0].)
5. Set the display to DSP 3 to see the three digits of the course.
6. Add [+] the fractionally formatted course to the rounded range. The result is a packed display.

EXAMPLE 16. Packing Output Data.

The course is 032°.45 and the distance is 45.8 n.mi. Pack the distance and the course into a single display.

Solution:

32.45 [↑] 45.8 [DSP] [0] [f] [RND] [h] [x ⇄ y] [EEX] [3] [÷] [DSP] [3] [+] .

The display contains 46.032.

EXAMPLE 17. Course-Made-Good Program with Packed Output.

Modify the course-made-good (Example 14) program so that the course and distance in the final display are in packed form.

Solution:

Instruction	Step	Keycodes
f LBL A	001	31 25 11
f R↓	002	31 72
Σ+	003	21
RCL Σ+	004	34 21
g →P	005	32 72
DSP 0	006	23 00
f RND	007	31 24
h x ≷ y	008	35 52
f x < 0	009	31 71
f GSB 0	010	31 22 00
EEX	011	43
3	012	03
÷	013	81
DSP 3	014	23 03
+	015	61
h RTN	016	35 22

Instruction	Step	Keycodes
f LBL 0	017	31 25 00
3	018	03
6	019	06
0	020	00
+	021	61
h RTN	022	35 22
g LBLf c	023	32 25 13
RCL Σ+	024	34 21
h Σ-	025	35 21
h RTN	026	35 22

Discussion: The following points should be noted:

1. Since the distance is in the X-register immediately following the execution of step 005, the rounding was performed immediately (steps 006 and 007) so that extra steps ($x \rightleftarrows y$) will not be required later.
2. The label on the "add 360°" routine was changed from E (in previous versions) to 0 (steps 010 and 017). Since we do not need to access the "add 360°" routine from the keyboard, we have freed the user defined label E for possible future use.

3. The course is converted to fractional format at steps 011, 012, 013, and 014. The packing is finalized at step 015.

4. Note the advantage of having the "add 360°" routine available as a subroutine. If step 010 was a GTO 0 rather than GSB 0 then we would somehow have to get from step 022 to step 011. It could be done but it would be inconvenient.

Sample Problem. Use the data from Example 14.

Leg	Course-made-good ddd.ccc
1. $\overrightarrow{OA}$ = (26°.5, 47 n.mi)	47.027
2. $\overrightarrow{AB}$ = (224°, 91 n.mi)	48.241
3. $\overrightarrow{BC}$ = (105°, 77 n.mi)	54.143
4. $\overrightarrow{CD}$ = (305°, 63 n.mi)	21.250

VII. UNPACKING

If sufficient program steps are available, we may wish to enter our data in a *packed* form. The HP-67 has automatic packing and unpacking of HH.MMSS (hours-minutes-seconds) in the built-in-conversion to and from hours and fractional hours. *Unpacking* is the inverse operation of packing.

Suppose we wish to *enter* distance and course in a packed form such as ddd.ccc where the d's denote the distance and the c's denote the course. Before any computations could be done on either the course or the distance, these two quantities must be separated. Unpacking is accomplished using the INT and FRAC functions (*Owner's Handbook*, pages 86 and 87). The LASTx function (pages 67 and 68) can also be useful. We proceed as follows:

1. Key in the distance and course in packed form ddd.ccc.
2. The distance is recovered by pressing [f] [INT]. The result is ddd.000.
3. The packed data is recovered by pressing [h] [LSTx] to obtain ddd.ccc.
4. The fractionally formatted course can be recovered by pressing [g] [FRAC] to obtain 000.ccc.

5. The course can then be obtained by multiplying by 1000 (EEX 3).
6. If the coordinates are to be changed to rectangular, then we must use [h] [x ≷ y] to get the course in the Y-register and the distance in the X-register.

EXAMPLE 18. Unpacking Input Data. Method 1.

Unpack 1527.033.

Solution:

Key in 1527.033. Then

[f] [INT] [h] [LSTx] [g] [FRAC] [EEX] [3] [x] .

(Optional: [h] [x ≷ y] .)

EXAMPLE 19. Unpacking Input Data. Method 2.

Unpack 1527.033 some other way.

Solution:

1527.033 [↑] [g] [FRAC] [-] [h] [LSTx] [EEX] [3] [x] .

(Optional: [h] [x ≷ y] .)

Morale: There is always another way to solve a problem. It may or may not be a shorter way.

EXAMPLE 20: Course-Made-Good Program with Packed or Unpacked Input.

Modify the last version of the course-made-good program so that it will accept *either* unpacked data (via label A) *or* packed data (via label E).

Solution:

Instruction	Step	Keycodes
f LBL E	001	31 25 15
↑	002	41
g FRAC	003	32 83
-	004	51
h LSTx	005	35 82
EEX	006	43
3	007	03
x	008	71
h x ⇄ y	009	35 52
f LBL A	010	31 25 11
f R↓	011	31 72
Σ+	012	21
RCL Σ+	013	34 21
g →P	014	32 72
DSP 0	015	23 00

Instruction	Step	Keycodes
f RND	016	31 24
h x ⇄ y	017	35 52
f x < 0	018	31 71
f GSB 0	019	31 22 00
EEX	020	43
3	021	03
÷	022	81
DSP 3	023	23 03
+	024	61
h RTN	025	35 22
f LBL 0	026	31 25 00
3	027	03
6	028	06
0	029	00
+	030	61
h RTN	031	35 22
g LBLf c	032	32 25 13
RCL Σ+	033	34 21
h Σ-	034	35 21
h RTN	035	35 22

Discussion: The new program steps to unpack the data entry (001-009) have been inserted just above label A. When packed data is entered and key E is pressed, the unpacking takes place in program steps 002 through 009. At step 010 LBL A is encountered and no action takes place. However the contents of register X and Y have been set in the unpacked format that is entered via key A. Execution then continuous with step 011 in the usual manner.

VIII. FLAGS

Flags are primarily used to control and direct the logical flow of an executing program. A flag might be thought of as a mini-storage register that can contain a single bit of information, that is, a 1 or a 0. The "1 or 0" can be interpreted to mean "yes or no," "true or false," "on or off," "up or down," "set or clear," and "flying or not flying," to name but a few of the commonly used terms. The four flags, F0, F1, F2, and F3 in the HP-67 can be *command set* (i.e., set to the value 1) using the [h] [SF] [n] instruction where n = 0,1,2, or 3. The four flags can be *command cleared* (i.e., cleared to the value 0) using the [h] [CF] [n] instruction. Also, the condition of each flag may be tested using the [h] [F?] [n] instruction. If the flag tested is in a *set* (1) condition, then the next sequential program step is executed; if the flag tested is in a *clear* (0) condition, then the next sequential program step is bypassed and the following program step is executed (see illustration on page 255 of the *Owner's Handbook*). The operation of the flags is discussed on pages 255 through 269 of the *Owner's Handbook*.

In addition to clearing the flags by command, flags 2 and 3 are also *test-cleared* flags. That is, whenever flag 2 or flag 3 is tested using the [h] [F?] [n] instruction (where n = 2 or 3), the program instruction pointer is advanced by 1 or 2 depending on whether the flag is on or off, and then the flag is automatically *cleared* before program execution continues at the current program instruction designated by the instruction pointer. An additional feature of flag 3 is that it is automatically set each time a data entry is made; this feature will not be discussed here.

EXAMPLE 21. Output Control with Flags.

Suppose that we wish to calculate distance (d) when speed (s) and time of travel (t) is specified. Further, speed will always be given in knots and time will be in the HH.MMSS format. The distance, given by $d = st$, is to be computed and displayed in either nautical miles or in yards as an option. When the speed is in knots and the time is in hours, the product is distance in nautical miles. To convert nautical miles to yards, we must multiply by 2025 (for more precision, multiply by 2025.372). For input, we use [s] [↑] [t], and then press [A] for output in nautical miles or press [E] for output in yards.

Solution.

Instruction	Step	Keycodes
f LBL A	001	31 25 11
h SF 0	002	35 51 00
GTO 0	003	22 00
f LBL E	004	31 25 15
h CF 0	005	35 61 00

Instruction	Step	Keycodes
f LBL 0	006	31 25 00
f H←	007	31 74
x	008	71
h F? 0	009	35 71 00
h RTN	010	35 22
2	011	02
0	012	00
2	013	02
5	014	05
x	015	71
h RTN	016	35 22

Sample Problems.

1. Speed is 5 knots and time is 30 minutes. Compute the distance traveled in nautical miles.

   Solution: 5 [↑] 0.3 [A] 2.50 n.mi.

2. Speed is 5 knots and time is 30 minutes. Compute the distance traveled in yards.

   Solution: 5 [↑] 0.3 [E] 5062.50 yards.

Discussion.

Problem 1.

When the [A] key is pressed, the program pointer is moved to step 001 and execution begins. At step 002, flag 0 is placed in the *set* mode and then transfer is made to label 0 at step 006. At step 007 the time in HH.MMSS format is convered to time in hours (see pages 94 to 96 of the *Owner's Handbook*). At step 008, the speed in the Y-register is multiplied by the time in the X-register, the result is the distance in nautical miles in the X-register. At step 009 the status of flag 0 is tested. Since flag 0 was *set* at program step 2, the next step (step 010) is executed, thus halting the program at step 010 with the distance in nautical miles in the display (X-register).

Problem 2.

When the [E] key is pressed, the program pointer is moved to step 004 and execution begins. At step 005, flag 0 is placed in the *clear* mode. Execution continues through the label 0 at step 006, and then the operation is the same as described above until flag 0 is tested at step 009. Since flag 0 is *clear*, the program pointer

advances from step 009 to step 011 (omitting step 010). At steps 011 through 014 the constant 2025 is generated in the X-register. At step 015 the distance in nautical miles in the Y-register is multiplied by the conversion factor 2025 in the X-register. The result is the distance in yards in the display. Execution is halted at step 016.

In an earlier section we learned how to record a program onto a magnetic card. Every time a program is recorded onto a magnetic card, the current setting of the display (DSP; FIX, SCI, or ENG), the current setting of the trigonometric mode (DEG, RAD, or GRD), and the current status of the four flags are also recorded onto the program card (*Owner's Handbook*, pages 272 to 274). The next example will illustrate a way in which the test cleared flag F2 can be preset when a program is recorded so that each time the program is used, one portion of the coding is executed once and once only. This feature can be used to preset constants or initialize data so that the program execution time is faster for the second and subsequent data entries than it is on the first data entry.

EXAMPLE 22. Leroy's Equation.

The sound velocity in water can be calculated from Leroy's Equation [Ref. 4]. An abbreviated form of Leroy's equation is

$$V = 4755.2 + 15.067T - 0.1765T^2 + 0.00085T^3$$

where T is the water temperature in degrees Celsius and V is the sound velocity in feet per second. (A more detailed form of Leroy's equation will be presented later. For now we will assume that we measure or convert water temperature to the Celsius scale.)

This program is to be written so that Leroy's equation is evaluated as a nested polynomial (see Example 13, Solution 2):

$$V = ((0.00085T - 0.1765)T + 15.067)T + 4755.2 .$$

Further, and as a time saving device, the constants in Leroy's equation are to be generated and stored in primary registers R1 through R4 when the first value of T is input, but the portion of the program that stores the constants is to be bypassed if additional values of T are to be converted to sound velocity.

Solution:

Instruction	Step	Keycodes
f LBL A	001	31 25 11
STO A	002	33 11
h F? 2	003	35 71 02
f GSB 9	004	31 22 09
RCL 4	005	34 04
RCL A	006	34 11
x	007	71
RCL 3	008	34 03
+	009	61
RCL A	010	34 11
x	011	71
RCL 2	012	34 02
+	013	61
RCL A	014	34 11
x	015	71
RCL 1	016	34 01
+	017	61
h RTN	018	35 22
f LBL 9	019	31 25 09
4	020	04
7	021	07
5	022	05
5	023	05

Instruction	Step	Keycodes
.	024	83
2	025	02
STO 1	026	33 01
1	027	01
5	028	05
.	029	83
0	030	00
6	031	06
7	032	07
STO 2	033	33 02
.	034	83
1	035	01
7	036	07
6	037	06
5	038	05
CHS	039	42
STO 3	040	33 03
8	041	08
5	042	05
EEX	043	43
5	044	05
CHS	045	42
STO 4	046	33 04
h RTN	047	35 22

Discussion: The program card can be prepared in one of two ways:

1. a. With the Program Mode Switch in the W/PRGM position, key in the program.
   b. Slide the Program Mode Switch to the RUN position.
   c. Press the keys [h] [SF] [2] [DSP] [0].
   d. Slide the Program Mode Switch to the W/PRGM position.
   e. Pass a blank magnetic card through the card reader to record the program *and* the Flag 2 SET status onto the program card. The display status is also recorded onto the program card.

2. a. With the Program Mode Switch in the RUN position, press the keys [h] [SF] [2] [DSP] [0].
   b. Slide the Program Mode Switch to the W/PRGM position.
   c. Key in the program.
   d. Pass a blank magnetic card through the card reader to record the program *and* the Flag 2 SET status onto the program card. The display status is also recorded.

When the program card is passed through the card reader (Program Mode Switch in the RUN position), the program is transferred to the program memory *and* Flag 2 is SET. When the *first* temperature is keyed in and [A] is pressed, the temperature is stored in register A at step 002. At step 003, the status of Flag 2 is tested. The process of testing Flag 2 automatically CLEARS Flag 2, but since Flag 2 was SET the instruction at step 004 is executed and a branch is made to the subroutine at f LBL 9 (step 019). The subroutine stores the four constants in registers Rl through R4, and then at step 045 the RTN instruction returns control to program step 005. From steps 005 through 018 Leroy's equation is evaluated and execution of the program is halted at step 018 with the sound velocity in the display.

When a *second* temperature is keyed in and [A] is pressed, the temperature is stored in register A at step 002. At step 003, the status of Flag 2 is tested. Since Flag 2 is now in the CLEAR condition step 004 is skipped and execution continues from program step 005 and Leroy's equation is evaluated. Since the constants are stored in registers Rl through R4, a duplicate execution of subroutine LBL 9 is bypassed.

Sample Problem.

Find the sound velocity for the following water temperatures:

a. 25°C (5035.)

b. 20°C (4993.)

c. 15°C (4944.)

d. 10°C (4889.)

If Flag 2 has not been properly set and recorded then the answers will not be correct. Note how much more rapid the computations become at the second and subsequent temperature inputs than they were for the first input.

Example 22 has been designed to illustrate the use of the test cleared Flag 2 and of the recording of the flag status onto a program card. (If Flag 3 had been used instead of Flag 2, then subroutine LBL 9 would be executed each time [A] is pressed. Why?) We will write this program more efficiently in later Examples.

IX. CARD READER OPERATIONS

In earlier sections we have shown how to record and read program cards. We have also shown how a flag can be preset and this information recorded onto a card. There are other card reader operations that are discussed in the *Owner's Handbook* in Section 14 on pages 271 to 297. Pages 271 to 274 and pages 278 to 286 should be studied carefully; the remaining portions of Section 14 contain other interesting but rarely used operations that can be studied at a later time.

The programs that have been presented here are short, 112 program steps or less, and so they have required only one side of a card, or one pass through the card reader.

Let us create a small "do nothing" program that illustrates the card reader operation of a long program. Turn the calculator on and place the Program Mode Switch in the W/PRGM position. Press the keys [f] [LBL] [A]; position the program step pointer to step 113 by pressing [GTO] [.] [1] [1] [3]; then press the keys [h] [RTN]. Now record the "program" onto a card by passing a blank card through the card reader while the Program Mode Switch is in the W/PRGM position. When the card has passed through

the card reader the display shows [Crd], which is the prompt which requests you to pass the *second* side of the card through the card reader. When this is done the display shows [114 35 22] which is the [h] [RTN] that we keyed in.

What we have done is to place information other than the "power up" [R/S] or [84] code in the *lower half* (steps 001-112) of program memory, namely [f] [LBL] [A], and we have placed [h] [RTN] in the *upper half* (steps 113-224) of program memory. When the first side of the card is passed through the card reader, program steps 001-112, the display status, the flag status, the program mode status, and the prompt to read the second side of the card are written on the one half of the card. The prompt [Crd] shows in the display to indicate that there are additional program steps to be recorded on the second half of the card. Now, slide the Program Mode Switch to the RUN position, turn the calculator OFF, then ON, and pass one side of the program card through the card reader. The display will then show [Crd] in the display to indicate that the *other* side of the card is to be passed through the card reader (if the *same* side is passed through, the display will still show [Crd] indicating that you have not fooled the calculator; it is patiently waiting for

the other side). When the second side is passed through, the display returns to normal (or to the pre-recorded display status) indicating that the card has been successfully read.

In addition to recording program information onto a card, it is also possible to record the contents of all primary and secondary registers onto a card. All of the primary register (R1-R9, RA-RE, and RI) contents are written onto one side of a *data card*, and the secondary register (S0-S9) contents are written onto the second side of a data card. To illustrate this operation, press [1] [STO] [1] with the Program Mode Switch in the RUN position. Then press the keys [f] [W/DATA]; the display will show the [Crd] prompt indicating that a blank card is to be passed through the card reader. When this is done, the display returns to normal. Now turn the calculator OFF, then ON, then pass the recorded side of the card through the card reader and press [RCL] [1]; [1.00] should be shown in the display. In this example only a single side of a card has been required to record the data (non-zero data) that was stored in register 1.

To illustrate recording both sides of a data card, press [1] [STO] [1] as before to store a 1 in storage register R1. Now press [Σ+]. If the 1.00 was still in the display, it will be added to the contents of register S4 and simultaneously the contents of register S9 will be increased by 1. What we have done is to introduce non-zero information into the secondary storage registers. Now press [f] [W/DATA]; the display will show [Crd] indicating that one side of a blank card is to be passed through the card reader. When this is done, all of the data in the primary storage registers are recorded onto one-half of the card. The display will still show [Crd] indicating that the second half of the card should be passed through the card reader to record the secondary storage register contents onto the second half of the card. When this is done, the display returns to normal.

To illustrate reading the card, turn the calculator OFF, then ON. Pass either side of the card through the card reader (the calculator must be in the RUN mode); the display will show [Crd] indicating that the other side of the card is to be passed through the card reader so that remaining information will be transferred to the proper storage locations.

If a short program (112 program steps or less) is written which also uses some fixed data (primary registers only), then it is possible to place the program on one side of a magnetic card and the data on the other side. To illustrate this, turn the calculator on and place it in the RUN mode. Key in [1] [STO] [2], then [f] [W/DATA]; when [Crd] appears, pass one side of a magnetic card through the card reader to record the data. Next place the calculator in the W/PRGM mode and key in any thing except R/S (for example, [f] [LBL] [A]). Then pass the other side of the magnetic card through the card reader to record the "program." To demonstrate that the program/data card has been properly written, place the calculator in the RUN mode and turn it OFF, then ON. Pass one side and then the other side of the program/data card through the card reader. Observe that there was no [Crd] prompt given; the calculator did not "know" that a part program, part data card had been written. Press [RCL] [2] and the 1.00 from storage register 1 will appear. Slide the Program Mode Switch to W/PRGM; 000 should appear. Press [SST] and observe [001 31 25 11] in the display (f LBL A).

The only undesirable feature of producing a program/data card as indicated above is the fact that we do not obtain the very convenient [Crd] prompt. With a little trickery we can produce a card which does give us a prompt from one side of the card. Proceed with the previous example as follows. With the calculator in the RUN mode press [1] [STO] [2] [Σ+]; the quantity 1.00 is stored in primary register 2, and the Σ+ has placed non-zero "garbage" in the secondary statistical registers (which are not to be recorded anyway). Place the calculator in the W/PRGM mode and key in any keystrokes except R/S (for example, [f] [LBL] [A]). Next return the calculator to the RUN mode and press [f] [W/DATA] when the [Crd] prompt appears, pass side 1 of a magnetic card through the calculator. The [Crd] prompt will reappear, indicating that there is data in the secondary registers to be recorded. Instead of recording the secondary register data, press [CLx]; the display should return to normal. Place the calculator in the W/PRGM mode and pass the second side of the card through the card reader. The program/data card has been prepared with a prompt on side 1. This procedure is perhaps a little lengthy, but it is usually well worth the additional effort required to add the prompt.

Test your card for proper operation by turning the calculator OFF, then ON. In the RUN mode pass Side 1 of the card through the card reader; Crd should show in the display. Pass Side 2 of the card through the card reader and the display should return to normal. Note that the prompt is *not* obtained if Side 2 is passed through the card reader before Side 1.

EXAMPLE 23. Leroy's Equation. Program/Data Card.

Rewrite the Leroy equation program (Example 22) so that the data generated and stored by subroutine 9 is instead placed on the data side of a program/data card. The Flag 2 and subroutine 9 usage can be removed from the program which should be placed on the program side of the program/data card. Place a [Crd] prompt on side 1 of the program/data card.

Recall that the nested form of Leroy's Equation is

$$V = ((0.00085T - 0.1765)T + 15.067)T + 4755.2.$$

Solution:

In the RUN mode, key in:

a)	4755.2	[STO] [1]	
b)	15.067	[STO] [2]	
c)	-0.1765	[STO] [3]	
d)	0.00085	[STO] [4]	
e)		[Σ+]	Place nonzero information in the secondary registers.
f)		[DSP] [0]	
g)		[f] [W/DATA]	Pass side 1 of a magnetic card. When the prompt [Crd] appears, press [CLx].

Place the Program Mode Switch in the W/PRGM position.

a) Key in:

Instruction	Step	Keycodes
f LBL A	001	31 25 11
STO A	002	33 11
RCL 4	003	34 04
x	004	71
RCL 3	005	34 03
+	006	61
RCL A	007	34 11
x	008	71
RCL 2	009	34 02
+	010	61
RCL A	011	34 11
x	012	71
RCL 1	013	34 01
+	014	61
h RTN	015	35 22

b) Pass Side 2 of the magnetic card through the card reader, then place the Program Mode Switch in the RUN position.

The program is now ready to use. To test it, turn the calculator OFF, then ON. Pass Side 1 of the program card through the card reader, this will place the constants of Leroy's Equation into Primary Registers R1, R2, R3, and R4. The display should show the [Crd] prompt. Pass Side 2 of the program card through the card reader. The display should show [0.]. Now try the sample problem

a.	25°C	[A]	5035. ft/sec
b.	20°C	[A]	4993. ft/sec
c.	15°C	[A]	4944. ft/sec
d.	10°C	[A]	4889. ft/sec

Occasionally a program is written which cannot be shortened to fewer than 224 program steps. In such cases *multiple card programs* can be written. Usually, as much of the program as possible is placed in the 224 available program steps. This program segment should end with an [h] [RTN] or a [R/S] instruction. If the stack is not being used a 2. immediately preceding the [h] [RTN] or [R/S] can be used to place [2.] in the display to prompt the user to read in a second program card which contains

the continuation of the program. When the new program card is read, the contents of the stack registers and all primary and secondary registers are unaffected. Thus the new program steps *overlay* the old program steps, which are no longer needed. The computations are resumed by pressing some user defined key in the second program card. This process may be continued for as many program cards as required.

The additional card reader features of program merging and data merging are discussed on pages 274 to 278 and pages 286 to 297 of the *Owner's Handbook*. These features are used so rarely that they will not be discussed here.

X. INDIRECT CONTROL (PANDORA'S BOX)

Sections 11 and 12 (pages 213-253) of the *Owner's Handbook* are devoted to "Controlling the I-Register" and "Using the I-Register for Indirect Control," respectively. The uses of the indirect control features are quite varied and are limited only by the creativity of the programmer. One exotic use is to control the display in conjunction with the printer on the HP-97 to plot graphs. This graphic technique has been used to plot propagation loss profiles on the HP-97 with ranges shown in the exponent field. Here we will only describe the indirect functions briefly and we will present a final version of the Leroy Equation program to illustrate one application of indirect control.

The I-register and its contents are involved in all indirect control applications. Any number in the display can be stored in the I-register using the [h] [STI] keystrokes. Similarly the contents of the I-register can be recalled using [h] [RCI]. The contents of the x-register and the I-register can be exchanged using [h] [x ≷ I]. The contents of the I-register can be increased by one using the *Increment and Skip on Zero* keystrokes [f] [ISZ], and the contents of the I-register can be decreased by one using

the *Decrement and Skip on Zero* keystrokes [f] [DSZ]. In the Program Mode, the [f] [ISZ] and [f] [DSZ] instructions operate as conditional branches. For example, the [f] [DSZ] instruction operates as follows: first the contents of the I-register is decreased by 1, then the contents of the I-register is tested. If the integer part of the number in the I-register is not zero, then the program instruction pointer is increased by 1 and the instruction at that program step is executed; if the integer part of the number in the I-register is equal to zero, then the program instruction pointer is increased by 2 (hence the *Skip on Zero*) and the instruction at that program step is executed.

EXAMPLE 24. Use of DSZ Instruction.

Write a program to compute the sum of the numbers 1, 2, ... , N by adding these integers together. (Do not use the formula SUM = N(N + 1)/2.) The program is to be initiated by placing N in the x-register and then pressing [A].

Solution.

Instruction	Step	Keycodes
f LBL A	001	31 25 11
h STI	002	35 33
0	003	00
f LBL 0	004	31 25 00
h RCI	005	35 34
+	006	61
f DSZ	007	31 33
GTO 0	008	22 00
R/S	009	84

Discussion: At step 002, the number in the display is stored in the I-register. At step 003 the contents of the X-register are set to zero. At step 005 the contents of the I-register are recalled, and at step 006 they are added to what was in the X-register. At step 007, the contents of the I-register are decreased by 1. If the new content of the I-register is zero, then the program instruction counter is increased by 2 to 009 and the R/S instruction of step 009 is executed, thus halting the computation with the sum in the display. If the new contents of the I-register are not zero, then the program instruction counter is increased by 1 to 008 and the GTO 0 instruction at step 008 is executed. Control branches unconditionally to step 004. At step 005 the new contents of the I-register are recalled, and at step 006 they are added to what was in the X-register. The process will continue until the contents of the I-register are reduced to zero and execution terminated at step 009.

Sample Problems.

N	SUM
1	1.00
2	3.00
3	6.00
4	10.00
5	15.00
10	55.00
20	210.00
50	1275.00
100	5050.00

The instruction [f] [ISZ] operates in a manner similar to the [f] [DSZ] except that the contents of the I-register are increased instead of decreased before testing. [f] [ISZ] will not be discussed here.

The five instructions discussed thus far involve the I-register in a direct manner. Indirect control involves using the contents of the I-register to control some other function. For example, store a 5 in the I-register, and then press the keys [DSP] [(i)] and you will see 5.00000 in the display, just as you would if you had pressed the keys [DSP] [5]. The [(i)] key designates that *if at all possible* the contents of the I-register are to be used to perform the designated operation. If the contents of the I-register conflict with the designated operation, then the operation will not be performed and an [Error] indication will be given.

Try [9.99] [h] [STI] [DSP] [(i)]. The result 9.990000000 is shown in the display. Here the integer part of the contents of the I-register was used as if [DSP] [9] had been keyed. Now try [-9.99] [h] [STI] [DSP] [(i)]. The result is -9.99000000 in the display showing that the absolute value of the integer part of the contents of the I-register was used to set the display. However [10.] [h] [STI] [DSP] [(i)] leads to an [Error] display since it is not possible to display 10 figures after the decimal following any operation.

EXAMPLE 25. Display Control with I-Register.

This example illustrates both indirect control of the display and direct control of the I-register. Key in the following program:

Instruction	Step	Keycodes
f LBL A	001	31 25 11
9	002	09
h STI	003	35 33
f LBL 0	004	31 25 00
h RCI	005	35 34
DSP(i)	006	23 24
h PAUSE	007	35 72
f DSZ	008	31 33
GTO 0	009	22 00
R/S	010	84

Now press [A]. The following number will be displayed with a brief pause for each number:

9.000000000

8.00000000

7.0000000

6.000000

5.00000

4.0000

3.000

2.00

1.0

Discussion: When [A] is pressed, a 9 is stored in the I-register. At step 005 the contents of the I-register are recalled to the X-register for display. At step 006 the display format is controlled indirectly by the number in the I-register. At step 007 execution is halted for about one second so that the contents of the X-register may be viewed. At step 008 the contents of the I-register are decremented by one and then the contents of the I-register are tested. If the contents of the I-register are not zero, then the program instruction counter is increased by 1 and step 009 is executed thus branching unconditionally

to LABEL 0 at step 004. If the contents of the I-register are zero after the decrement at step 008, then the program instruction counter is increased by 2 and step 010 is executed halting the program.

Indirect control of the display is only one of the many indirect functions. A complete list of the indirect control functions with a brief description of each is given on pages 223-224 of *Owner's Handbook*. We conclude with a final version of the Leroy Equation program to illustrate some of the saving that can be obtained with indirect control.

EXAMPLE 26. Leroy's Equation with Indirect Control.

In Example 23 the Leroy Equation program was written so that the data was stored on one side of a magnetic card and the program was on the other side. The same data storage allocation will be used here. With a minor modification, the program instructions from Example 23 are repeated below for ease of discussion. They are:

```
f LBL A
STO A
RCL 4
-----------
| RCL A   |
| x       |
| RCL 3   |
| +       |
---------
| RCL A   |
| x       |
| RCL 2   |
| +       |
---------
| RCL A   |
| x       |
| RCL 1   |
| +       |
---------
h RTN
```

Observe that three blocks of the program sequence are identical with the exception of recalling a constant from a different storage location. With indirect control we could replace each of these blocks with the sequence:

```
RCL A
x
RCL (i)
+
```

Then the value of the I-register can be controlled using the DSZ instruction.

Solution:

Instruction	Step	Keycodes
f LBL A	001	31 25 11
STO A	002	33 11
3	003	03
h STI	004	35 33
RCL 4	005	34 04
f LBL 5	006	31 25 05

Instruction	Step	Keycodes
RCL A	007	34 11
x	008	71
RCL (i)	009	34 24
+	010	61
f DSZ	011	31 33
GTO 5	012	22 05
h RTN	013	35 22

Discussion: Although the step saving in this example is not dramatic it does indicate what can be done with indirect control. If Leroy's Equation had one or two more constants in the temperature series, the step saving would become quite dramatic. In this program, the temperature in °C is stored in register-A (step 002). At steps 003 and 004 the constant 3 is stored in the I-register for indirect control. At step 005 the constant 0.00085 is brought into the X-register. At steps 007 and 008 the temperature is recalled and multiplied by the constant

from R4. At step 009 the contents of R3 are recalled indirectly (a 3 is in the I-register), and at step 010 the constant is added. At step 011 the contents of the I-register are decreased by 1 (to 2) and at step 012 a branch is made to LABEL 5 at step 006 and the block is repeated using the constant in R2. The total process is continued until the program execution is stopped at step 013.

XI. ADDITIONAL EXERCISES

1. The surface duct cutoff frquency f is a function of the sonic layer depth d and the sound velocity c. The relationship is

$$f = \frac{cd^{-1.5}}{4.7 \times 10^{-3}}$$

where f is measured in Hertz, d is measured in feet, and c is measured in feet per second. Write a program to calculate f for various input values of c and d. An additional embellishment is to provide a default sound velocity of 5000 feet per second in the program.

[For $c = 5000$ ft/sec and $d = 260$ ft, $f = 253.8$ Hz;
for $c = 4980$ ft/sec and $d = 500$ ft, $f = 94.8$Hz.]

2. The velocity of sound in water can be calculated from Leroy's equation. A more elaborate form of Leroy's equation than that given in Examples 22, 23, and 25 is:

$$V = 4755.2 + 15.067T - 0.1765T^2 + 0.00085T^3 + 3.9(S - 35) + D/61$$

where

T is the water temperature in °Celsius,

S is the salinity in parts per thousand,

D is depth in feet, and

V is the sound velocity in feet per second.

Until the United States "goes metric," temperatures will commonly be measured in °Fahrenheit. The relationship between the two temperature scales is:

$$T = (t - 32)/1.8$$

where

t is the temperature in °Fahrenheit and

T is the temperature in °Celsius.

2a. Write a program to compute V from various values of t, S, and D. Then use your program to find V when $t = 76\,^{\circ}F$, S = 33 ppt, and D = 330 feet
(Ans: 5028 feet/second.)

b. Provide options to input the temperature in either degrees Fahrenheit or degrees Celsius, and to input depth in either feet or meters. (1 foot = 0.3048 meters.)

3. If a Lloyd Mirror pattern is observed and if the water conditions are isovelocity, then source depth d_s can be computed from range R, sound velocity c, hydrophone depth d_h and frequency separation F. The formula is

$$d_s = \frac{\lambda}{2} \sqrt{1 - \frac{R^2}{(\frac{\lambda}{2})^2 - d_h^2}}$$

where $\lambda = c/F$. R, d_s, and d_h are in feet, c is in feet/second, and F is in Hertz. Write a program to compute d_s for various values of R, d_h, c and F. [For c = 5000 ft/sec, R = 3000 ft, F = 35 Hz, and d_h = 300 ft we find that d_s = 739 ft.]

REFERENCES

1. LT. R. P. Huff, "On-Station Update of Oceanographic Information," COMPATWINGSPAC TAC MEMO 160-24-74, 22 October 1974.

2. R. H. Shudde, "On-Station Update of Oceanographic Information with Programs for the HP-67 Calculator," Naval Postgraduate School Technical Report NPS55-77-41, October 1977.

3. Robert J. Urick, Principles of Underwater Sound, 2nd Edition McGraw-Hill Book Company, 1975.

4. C. C. Leroy, "Development of Simple Equations for Accurate and More Realistic Calculations of the Speed of Sound in Sea Water," J. Acoust. Soc. Am. 46, 216 (1969).