Constructional Project

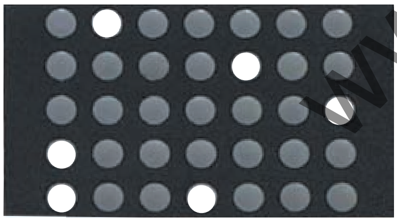
PIC-POCKET BATTLESHIPS

BART TREPAK

Become a Sea Lord with our interpretation of the age-old pen and paper game.

THE renowned game of *Battleships* is normally played by two players with pencil and paper. Its aim is for each opponent to sink the other's fleet before their own fleet is sunk. The ships are normally marked on a 10 × 10 grid of squares and each player calls out a grid reference in turn, to which the other player responds by saying whether it is a hit of a miss.

The variant of the game described here provides the excitement of the sea chase for just one player, who pits his wits against a PIC microcontroller as the other opponent. The position of the enemy (set by the PIC program!) is unknown and there are five merchant ships to be protected by the battleship. These six ship positions are shown on a 5 × 7 light emitting diode (l.e.d.) matrix display used horizontally.



This 5 x 7 matrixed l.e.d. display measures 39mm x 23mm.

PLAYING THE GAME

When the unit is first switched on, the positions of the five merchant ships are indicated by l.e.d.s that are lit continuously. The position of the battleship is represented by a flashing l.e.d., the "cursor". The enemy battleship is at the centre of the display but its position is not indicated.

The flashing cursor can be moved to any position on the display by means of four push-switches that control movement in the horizontal and vertical direction, one position at a time. Each time the cursor is moved, however, the unseen enemy ship can also move one square in the horizontal or vertical direction so that its current position changes and remains unknown. (Note that if the cursor is placed on the position occupied by one of the merchant ships, the l.e.d. will not flash).

When the player thinks the enemy is at the position of the cursor, the "fire" button may be pressed to try to sink the enemy. If the enemy ship is not in this position, the cursor will continue to flash and the game will continue. If the enemy *is* at this position then there are two possible outcomes of this engagement: either the player's battleship or that of the enemy will be sunk, and this is determined randomly!

If the enemy is sunk, the player wins the game (indicated by the cursor ceasing to flash) but if his own ship is sunk then a new one will appear at the start position with the enemy remaining at the position where the ship was sunk.

If the enemy warship moves into a position occupied by a merchant ship then that ship will be sunk immediately (i.e. the l.e.d. will go out) and the current position of the raider will be revealed. Of course as soon as the player attempts to move the cursor to this new position, the enemy may also move. If the merchant ship that is sunk is the last one, the game is lost and the cursor returns to its start position. To re-start the game, the unit must be reset by switching it off briefly.

NOT SUCH EASY PICKINGS

Although all the cards appear to be stacked in favour of the PIC, the raider is just as much in the dark about the position of the merchant ships as the player is about the position of the enemy. The PIC has no strategy other than to randomly move about the "sea" looking for ships, even to the extent of crossing and re-crossing the same squares.

If a ship is encountered then it will be sunk but, as in war, that is a matter of luck. Since the PIC has no memory of previous games or indeed even of its last move, there is no point in making the positions of the merchant ships variable or changing

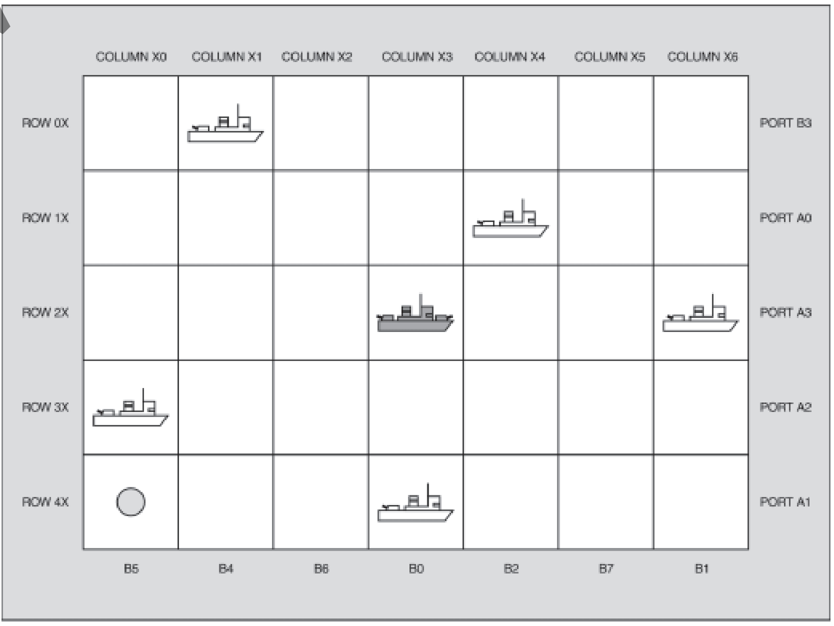


Fig.1. How the ships are positioned. The enemy battleship at the centre of the display is unseen. The circle represents your battleship and is a "moveable" flashing l.e.d.

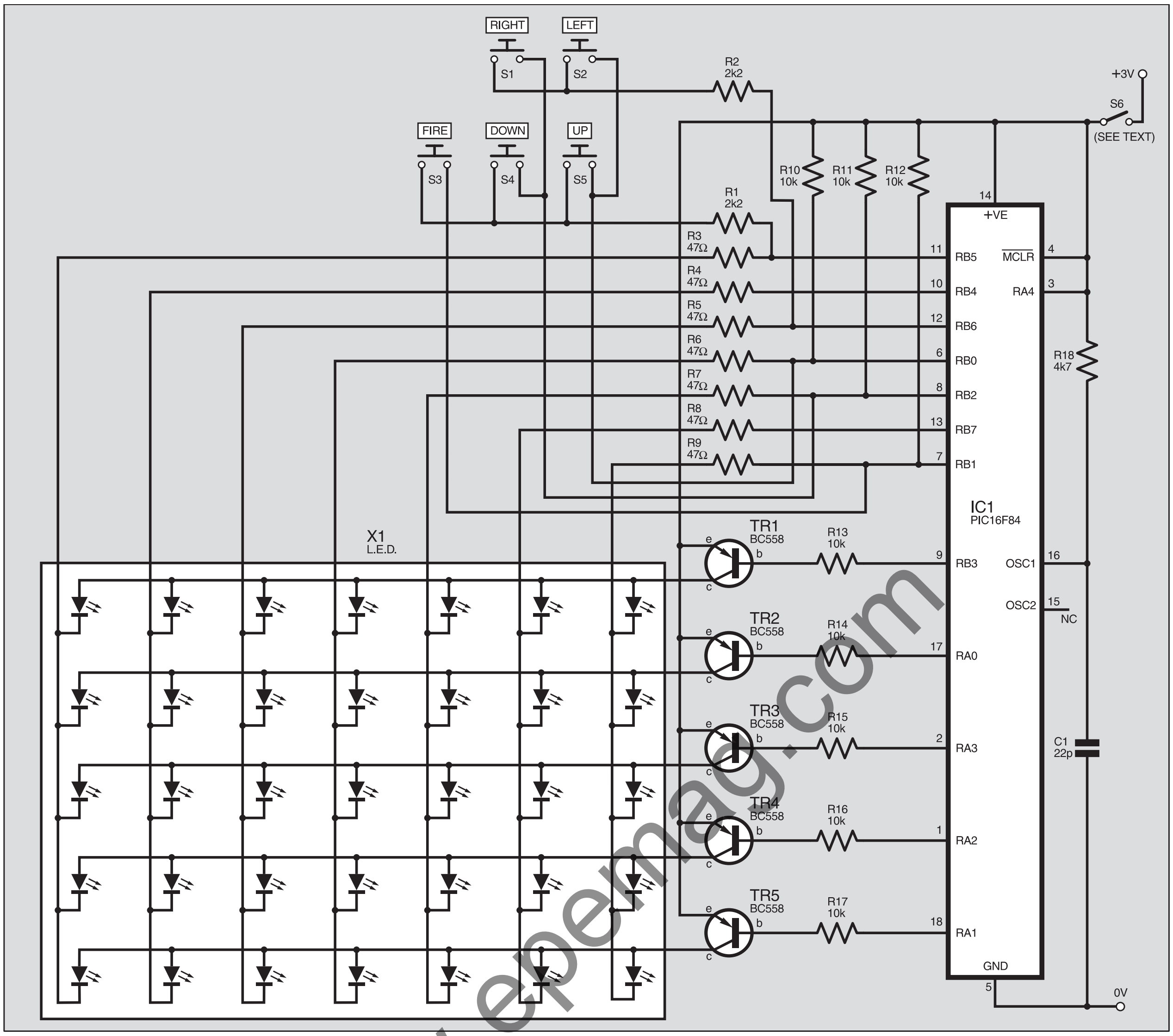


Fig.3. Complete circuit diagram for the PIC-Pocket Battleships game.

their position between or during games. These are therefore fixed by the program, as is the raider's initial position.

The "sea" is divided into "squares", each indicated by an l.e.d., with the columns numbered 0 to 6 while the rows are numbered as 0 to 4, as shown in Fig.1.

Each position is defined by one byte, shown in Fig.2, where the most significant nibble (highest four bits) defines the row while the other nibble defines the column. Thus the location at column 1, row 1, is represented by the hexadecimal (hex) number 00h. The raider's initial position is set at 23h as it will be in the third row down in the fourth column, while the positions of the merchant ships are stored as numbers 01h, 14h, 26h, 30h and 43h. The cursor position is defined in the same way, starting at 40h.

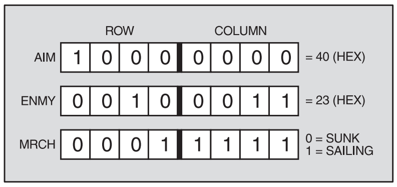

ROW
COLUMN
AIM 1 0 0 0 0 0 0 0 = 40 (HEX)
ENMY 0 0 1 0 0 0 1 1 = 23 (HEX)
MRCH 0 0 0 1 1 1 1 1
0 = SUNK
1 = SAILING

Fig.2. Arrangement of the program registers which hold the ship positions and their status.

At least five cursor moves are required to reach the raider's initial position, giving the enemy ship a chance to get away at the beginning of the game. The position of the enemy ship is stored in a register called ENMY and the cursor position in one called AIM.

The status of the merchant ships (i.e. sunk or afloat) is stored in register MRCH as five bits. These are set (binary 00011111) at the start of the game and individually reset to zero as each ship is sunk. These bits control the display so that a 0 in a particular position in this register prevents the l.e.d. for that ship from turning on, so that only the positions of the remaining merchant ships will be indicated.

When all five ships have been sunk, the game is lost and from the relative position of the cursor and the last ship sunk, the player will know how close he came to catching the enemy battleship.

CIRCUIT DIAGRAM

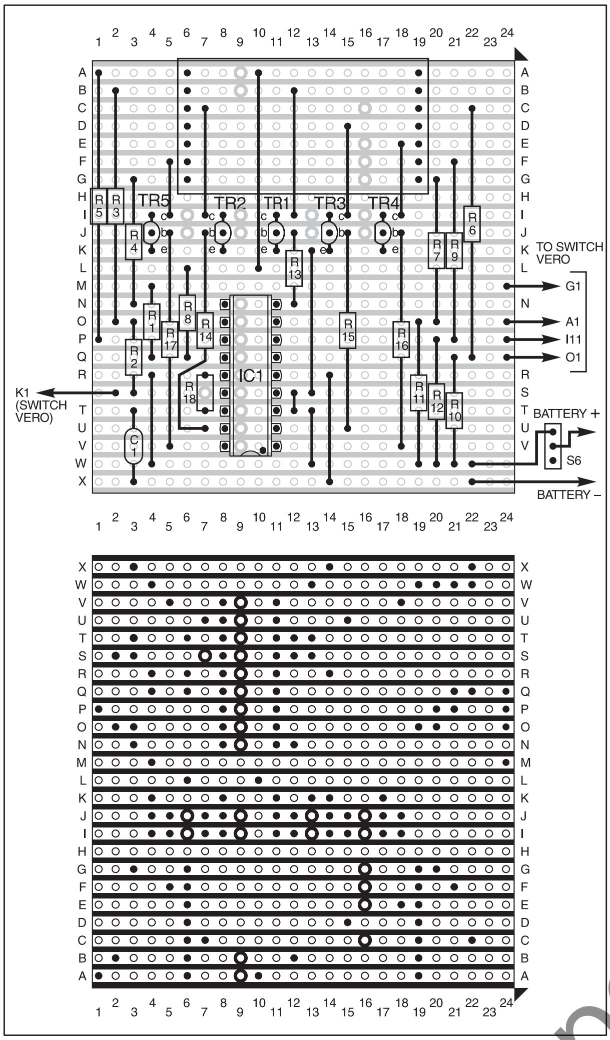
The complete circuit for PIC-Pocket Battleships is shown in Fig.3. It is based around a PIC16C54 microcontroller (one of the earlier PIC types having a UV erasable structure and window), which is operated in RC (resistor-capacitor) mode as precise timing of the software is not necessary. Resistor R18 and capacitor C1 set the PIC's clock frequency, at about 4MHz.

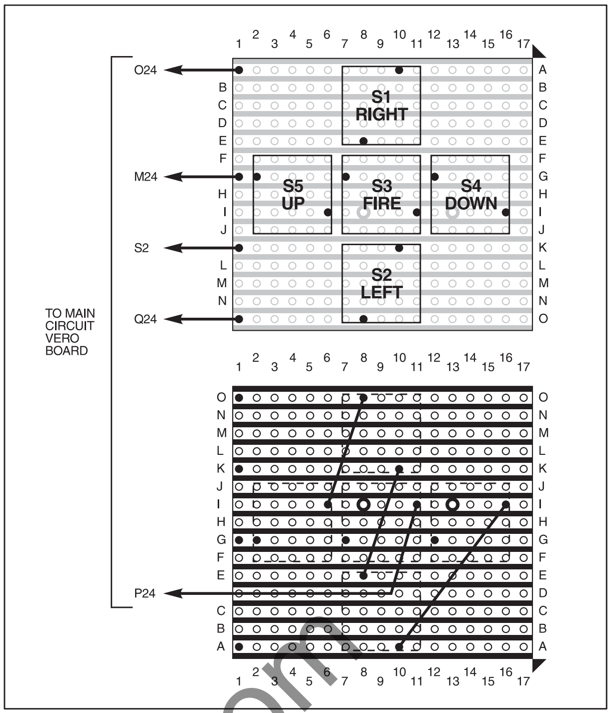
The l.e.d. display, X1, is multiplexed, which means that only one row is switched on at any one time. During this period, the appropriate column drives are activated in sequence. Only the l.e.d. at the junction of the "active" column and row is turned on.

As each row is switched on, the column drives are altered and because this is done very fast, all the "merchant ship" l.e.d.s appear to be on at once. The rows are driven via *pnp* driver transistors, TR1 to TR5, from PIC pins RA0 to RA3 plus RB3, buffered by resistors R13 to R17. To switch on a particular row, the corresponding output port goes low.

The column drives are output from the remaining lines of Port B via current limiting resistors R3 to R9. These lines also have to go low to switch on the corresponding l.e.d.

The function (game-play) selection switches S1 to S5 are also multiplexed to the lines connecting to the l.e.d. columns. They are additionally buffered by resistors R1 and R2. The PIC scans the switches to determine if a change in the cursor position is required or the fire button has been pressed. During scanning, RB7 is taken low and the three lines RB0 to RB2 are redefined as inputs and read in turn.





Component layout and stripboard track view for (Fig.4, left) the main control board, and (Fig.5, above) the optional switch board (see text).

COMPONENTS

Resistors

R1, R2	2k2 (2 off)	See SHOP TALK page
R3 to R9	47Ω (7 off)	
R10 to R17	10k (8 off)	
R18	4k7	

All 0·25W 5% carbon film.

Capacitor

C1	22p ceramic disc

Semiconductors

TR1 to TR5	BC558 *pnp* transistor (or similar)
IC1	PIC16C54 microcontroller, preprogrammed (see text)
X1	SE1110, 5 x 7 matrixed l.e.d. display, row-anode (see text)

Miscellaneous

S1 to S5	min. push-to-make switch, p.c.b. or panel mounting (see text) (5 off)
S6	min. s.p.s.t. toggle switch

Stripboard, 24 holes x 24 strips; stripboard, 17 holes x 15 strips (optional, see text); plastic case to suit (see text); battery holder/connector for 2 x AA batteries; connecting wire; solder, etc.

Approx. Cost Guidance Only **£15** *excluding battery*

Transistors TR1 to TR5 are turned off during this process. This prevents switch presses from shorting out column lines and causing erroneous displays. (Pressing more than one key at a time will still cause an erroneous display, but the game is not intended to be used in this way.) The software has been written to eliminate switch-bounce problems.

The circuit is designed to operate from a 3V d.c. supply and no voltage regulation is required. **It must not be run at a voltage greater than 6V d.c.**.

The PIC consumes very little current and since only one matrixed l.e.d. is on at any one time the current consumption of the whole unit is only about 10mA. Consequently, the circuit can be powered by two series-connected AA cells (1·5V each). It can also be operated on a 2·5V supply, so that rechargeable NiCad cells with their lower terminal voltage (1·2V) could also be used. (The PIC can be run from a voltage as low as 2V, although the l.e.d.s will be far less bright.)

RANDOMISING

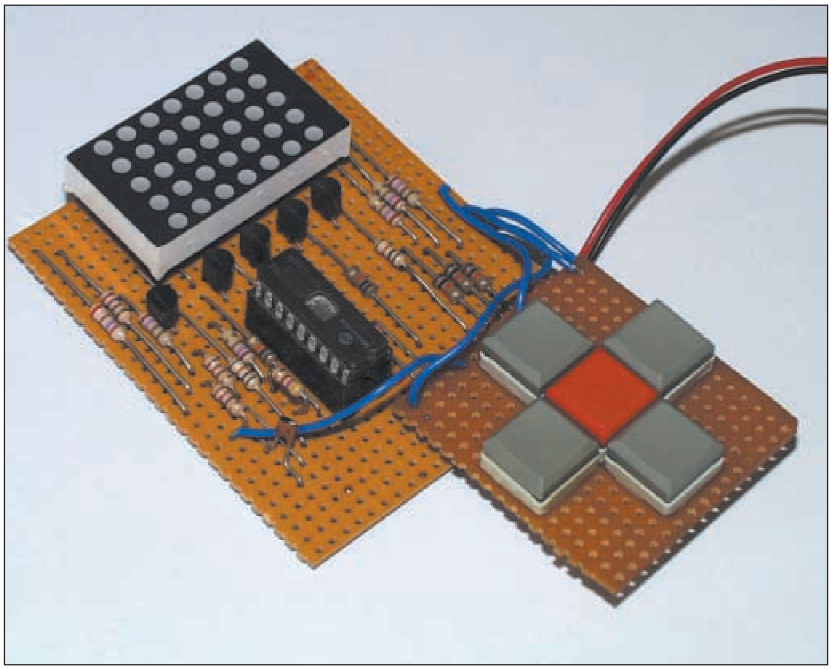
The game requires that random numbers are generated to determine the raider's next move. This is achieved by using a register which counts continuously while the program is running. The counter is read whenever one of the cursor positioning switches is pressed. Since this will occur at various time intervals, depending on the player and the fact that the count rate is very fast, the actual count reached will, to all intents, be indeterminate.

There are five possible ways that the enemy ship can move following a switch being pressed: up, down, left, right or remain in its current position. The counter is therefore programmed to count to four and when five is reached, it is reset to zero thus giving five different states. When a switch is pressed, the counter's value is read and the appropriate move is made. Bit 0 of this counter is also tested to determine the result of an encounter between the two opposing warships and thus provide an element of chance in the result.

The chances of one of these options occurring more often than the others can be increased by readers who are familiar with PIC programming. The software could be written to have more states than five and having a count of, say, one and two corresponding to the "move up" command, while three, four and five correspond to the "move left" command, for example.

Alternatively, making provision for the raider to move two squares on some of the counts could make the game more difficult. Adding or subtracting 02h instead of 01h from the enemy position register to move it horizontally, or 20h instead of 10h to move it vertically would do this.

The first part of the position controlling subroutine (EPOS) decodes the random counter (RND) and the program then proceeds, as appropriate to the decoded value, to move the enemy one square down, right, left or up, or to exit the routine without change. Adding or subtracting 10h or 01h from the current contents of the ENMY register does this and a software check is also made to ensure that the ship does not move out of the displayed area.



Thus if the enemy is at position 16h and 01h (move right) is added, the result will be position 17h which is off the screen. This is detected and 01h is subtracted again, thus leaving the enemy in position 16h. In this program, the effect of a ship trying to move out of the screen area will therefore result in a "no move" instruction and this will apply to both the enemy ship and the cursor.

The program could easily be changed so that if the above occurred, the enemy position could become 10h simply by loading ENMY with 10h when 17h is detected instead of subtracting 01h from this register. If this was made to apply only to the ENMY register and not to the AIM (cursor) register, the enemy ship would become much harder to catch. This could be done by setting or clearing a spare bit in the FLAG register (bit 4, say) and on this basis either subtracting 01h or resetting the target register to 10h as required.

Pressing the "move right" switch when the cursor is at position 16h, however, will still result in the enemy warship moving in accordance with the contents of the RND register at that instant, although the cursor will not move.

CONSTRUCTION

The PIC-Pocket Battleships' control circuit is assembled on a piece of stripboard, 24 strips long by 24 holes wide. This accommodates all of the components except the switches and the battery, which are connected to the board by flying leads. The component layout and track-cut details are shown in Fig.4.

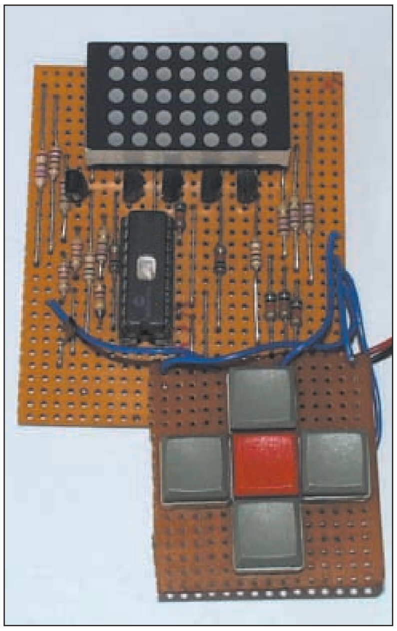
First make the 24 required breaks in the tracks, using a 2·5mm diameter drill bit, or the special tool available for this purpose. Next solder in all the link wires, noting that some go under the l.e.d. matrix position. Then insert and solder the components in any preferred order. A socket **must** be used for the PIC.

Although the transistors are specified as BC558 types, virtually any small signal *pnp* type will be suitable. Care should be taken if other types are used however, as their pinouts may vary.

The l.e.d. matrix type used in the prototype measures 39mm × 23mm, although 17mm and 50mm wide types could be used, provided that they are specified as "row-anode".

In the prototype, the display was mounted on the board by means of two 7-pin sockets obtained by cutting a standard 14-pin d.i.l. i.c. socket in half. It is positioned with its identity writing side to the right as viewed in Fig.4.

The prototype was not built into a box and the switches were mounted on a separate piece of stripboard whose assembly details are shown in Fig.5. However, the circuit could be fitted into a small handheld case which also has a battery compartment. Holes should be drilled for the switches. They should be standard push-to-make



types and connected to the board by flying leads.

When the circuit is complete, and fully checked for errors and bad soldering, the preprogrammed PIC can be fitted into its socket. Ensure that this is fitted the correct way around. See this month's *Shoptalk* page for details of obtaining the software and preprogrammed PICs.

The circuit should work correctly when power is switched on, provided it has been wired correctly. There are no adjustments to be made.

PROGRAM VARIATIONS

The use of a microcontroller enables various features to be added to the game to make it more interesting and these are limited only by the programmer's imagination, especially as no extra components are required.

Some of these possibilities have already been mentioned. One promising additional idea is to limit the quantity of ammunition carried by your own warship to, say, ten firings before the ship has to return to port (position 40h) to replenish its supply. A new register defined to count the number of times the fire button is pressed could be used to control this and the register could be reset to ten (or some other value) each time the cursor went to position 40h.

A similar idea would be to limit the range of your warship to say 20 moves. When this total expired, the ship could automatically "return" to port by loading 40h into the AIM register, or perhaps need to make its way back before its "fuel" ran out. A ship that could not return to port would be lost and a new one could appear at the home port. In this case, the number of warships could be limited to say three, so that if these were lost, the enemy would win the game.

This game could also be modified so that each time the enemy ship entered the port (i.e. location 40h) one of the ships there would be sunk. In this way, some of the player's ships could be sunk before they even left port. In this version, the port l.e.d. could remain illuminated to inform the player that new ships were still available.

ENEMY HOME PORT

The idea of a home port could also be easily extended to a base for the enemy (location 06h for example). Here, if the enemy returned to port, the player would loose the game so that as well as trying to protect his ships, the player would be forced to patrol near the enemy base to prevent the raider from returning home.

This option could appear only after all of the merchant ships had been sunk and here the movement options of the raider could be limited to move up, move right or stay still so that it would naturally tend to head for its base at the top right hand corner of the display when no more merchant ships remain afloat.

Modifications to the software to develop other scenarios to make the game harder would form an excellent basis for a science project to give budding programmers an opportunity to exercise their programming skills! □