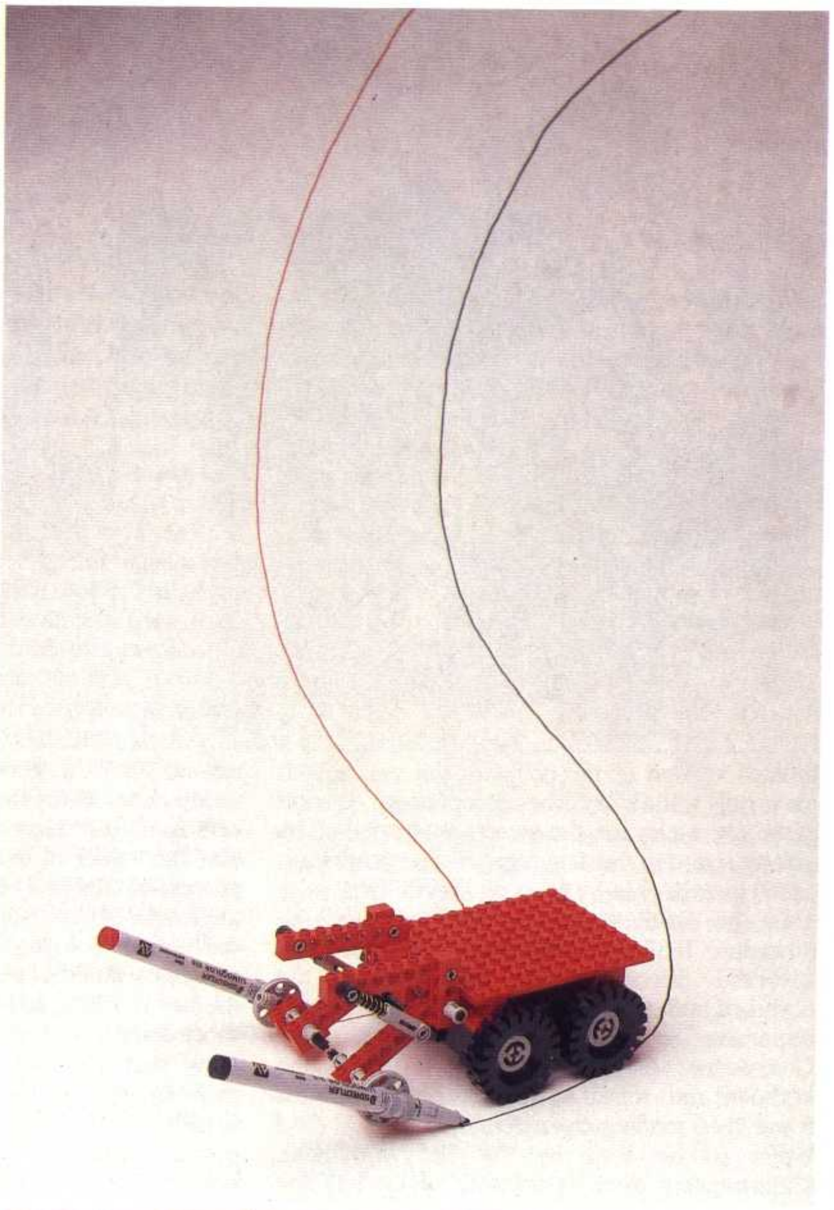
```
1120 REM REPLAY DATA
1130 FOR I=1TOC
1140 ?DATREG=DR(I,1)
1150 TIME=0
1160 REPEAT UNTIL TIME>=DR(I,2)
1170 NEXT I
1180 END
1190 :
1200 DEF PROCtest_keyboard
1210 IFA$=" " THEN ?DATREG=0
1220 IF INKEY(-36) = -1 THEN ?DATREG=5
1230 IF INKEY(-101) = -1 THEN ?DATREG=10
1240 IF INKEY(-68) = -1 THEN ?DATREG=6
1250 IF INKEY(-85) = -1 THEN ?DATREG=9
1260 PT=?DATREG
1270 IF PT<>DR(C-1,1) THEN PROCadd_data
1280 ENDPROC
1290 :
1300 DEF PROCadd_data
1310 DR(C-1,2)=TIME: REM STORE LAST TIME
1320 TIME=0: REM START NEW TIME
1330 DR(C,1)=PT: REM STORE PORT STATUS
1340 C=C+1: REM INCREMENT COUNT
1350 ENDPROC
```

COMMODORE 64

```
  10 REM CBM 64 MOVEMENT MEMORY
  15 DIMDR(100,2): REM DIRECTION ARRAY
  20 DDR=56579:DATREG=56577
  25 POKE650,128 : REM SET KEY REPEAT MODE
  30 POKEDDR,255: REM ALL OUTPUT
  35 C=1: REM INITIALISE COUNT
  40 GETA$
  50 GOSUB1000: REM TEST INPUT
  70 IF A$<>"X" THEN FOR I=1TO200:NEXT:GOTO40
  80 POKE DATREG,0: REM OFF
  85 DR(C-1,2)=TI-T: REM ENTER LAST TIME
  90 STOP: REM TYPE 'CONT' TO CONTINUE
  95 REM REPLAY DATA
 100 FOR I=1TOC
 110 POKEDATREG,DR(I,1)
 120 T=TI
 130 IF (TI-T)<DR(I,2)THEN130
 140 NEXT
 150 END
 999 :
1000 REM TEST INPUT S/R
1005 IFA$=" " THEN POKEDATREG,0
1010 IFA$="T" THEN POKEDATREG,5
1020 IFA$="B" THEN POKEDATREG,10
1030 IFA$="F" THEN POKEDATREG,6
1040 IFA$="H" THEN POKEDATREG,9
1045 PT=PEEK(DATREG)
1050 IFPT<>DR(C-1,1)THENGOSUB1500
1498 RETURN
1499 :
1500 REM ADD DATA TO ARRAY
1510 DR(C-1,2)=TI-T: REM ADD LAST TIME
1520 T=TI: REM TAKE NEW TIME
1530 DR(C,1)=PT: REM ENTER CURRENT PORT CONTENTS
1540 C=C+1: REM INCREMENT COUNT
1999 RETURN
```

This program allows the user to move the vehicle about under keyboard control. As each move is recorded as a direction and a time interval, any errors introduced in the timing of each movement will produce errors in the replay. We are entering into the difficult area of real-time computing, where program structure and execution time can become important factors.

In the next instalment of Workshop we shall take control one stage further by bringing our twin motor vehicle under the control of a joystick.



Memory Movements
It is reasonably simple to write a buggy-controlling program that accepts directions from the keyboard and drives the car accordingly. It is not much more difficult to extend the program so that it stores the operator's commands, and then replays them to the buggy, thus reproducing — in theory — the previous pattern of movement. Comparing the original with the supposed duplicate gives a measure of the software problems caused by dealing with the real world: the computer works in exact numbers and times on a simplistic model of a perfect universe, not allowing for inertia, frictional losses, irregular surfaces and low-tolerance engineering. In the light of this experience, the performance of LOGO-driven floor turtles is impressive

Exercises

Now that we can control a vehicle's movement in all directions, many possibilities arise for short programming exercises. You can probably think of many but here are a few ideas:

1) Try to calibrate your vehicle. How long does the relevant number have to be in the data register to make the vehicle move one metre forwards or backwards, or turn through 90 degrees?

2) Design an obstacle course for your vehicle and, using the programs given as a basis, write a program that allows you to 'teach' the vehicle to negotiate the course under keyboard control. Once you have guided the vehicle through its course, the program should take over, guiding the vehicle back to its starting point and retracing the course.

3) Connect up four switches to the buffer box that allow you to control the vehicle externally from the user port.