Your COMPUTER COURSE

MAY 1986

A COMPREHENSIVE GUIDE TO PROGRAMMING – PART 3

THE PLEASURES OF PASCAL

TWO IMPLEMENTATIONS ASSESSED

TOP-DOWN GUIDE TO MACHINE CODE

INTRODUCING THE PRINCIPLES OF 68xxx M/C PROGRAMMING

SOUND ON THE AMSTRAD

CPC MUSICIANS TAKE NOTE

PAINTING BY NUMBERS

GETTING A LINE ON COMPUTER GRAPHIC TECHNIQUES

Welcome to the third part of the *Your Computer* Course. We hope that you have found the lectures so far both stimulating and refreshing, as well as informative.

This month's instalment features articles which cover all aspects of home computing. We begin by looking at computer graphics. It is one of the major growth areas in the home computer market – a wealth of high-quality software and hardware add-ons has reached the market in recent months. Those tools mean that the creation of professional-quality graphics is no longer the sole province of the graphic artist but is within grasp of every micro user.

Our guide to computer jargon continues on page 20; this month John Lettice covers letters I to L, from IBM to Loops.

Those who are interested in machine code programming will find an introduction to low-level coding on the 68xxx series of MPUs.

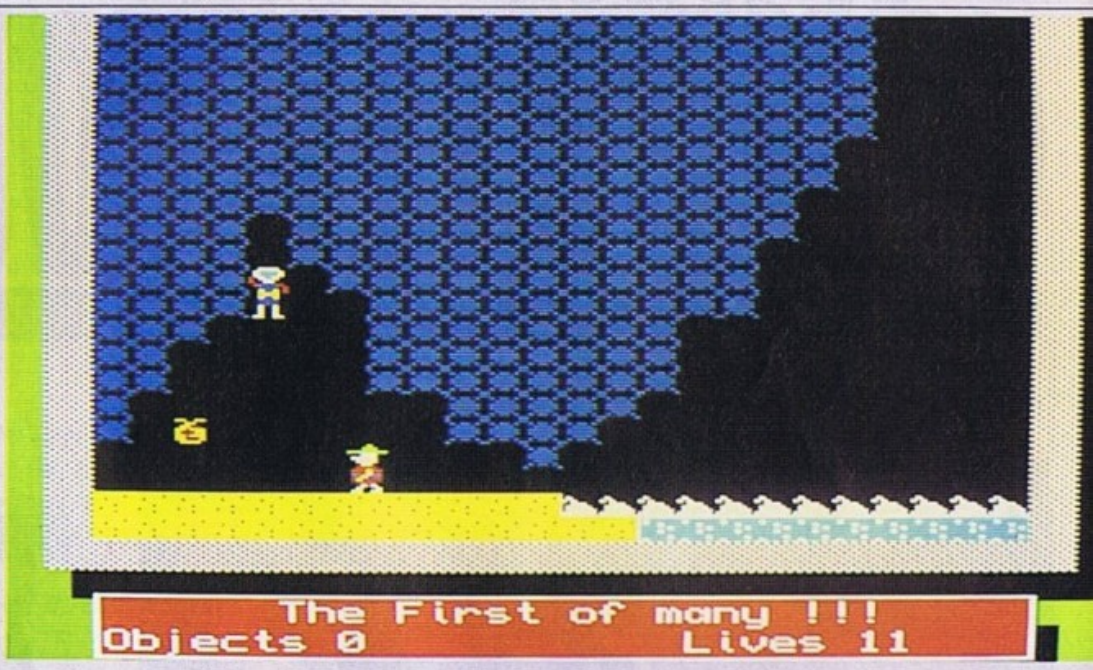
Music is another area gaining in popularity and this month we show users how to make the most of the Programmable Sound Generator which is responsible for the generation of music on the Amstrad CPC range of computers.

To round off this instalment of the course, Geof Wheelwright assesses two implementations of the Pascal programming language.

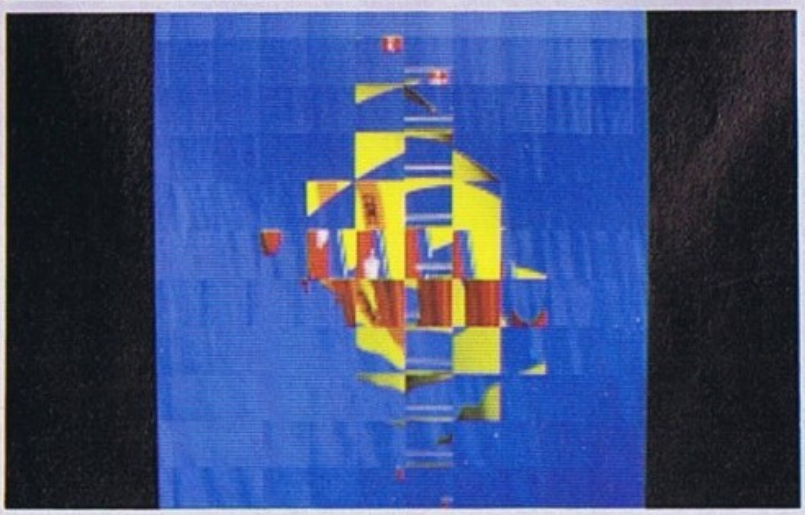
INDEX

Computer Graphics . 18
Glossary (I-L) 20
68xxx programming 22
Amstrad music 23
Pascal in action 24

Contributors:
David Janda
John Lettice
Geof Wheelwright



Computer generated



graphics

Of all the applications which can be set for a micro, few are immediately as rewarding as the generation of on-screen pictures or graphics. With the advent of modern paint software and sophisticated input devices such as the mouse, the touch tablet and the light pen, developing graphics is much easier than it used to be when all you had to hand was a grid sheet and a table of available screen colours.

You can, of course, still draw pictures on-screen by using programming methods, although many pictures used by commercial software programmers these days are either digitised using a video camera and an interface to a micro or painted using some form of sophisticated computer paint software.

On this page you will see some examples of both methods. Figures one and two show short programs for the Commodore 64 and Spectrum respectively which allow you to generate your own very rudimentary pictures using the computer's existing character set and the keyboard, while the photos show the kind of drawing program – in this case it is *QL Paint* for the Sinclair QL – which make painting your masterpieces a much easier job.

Painting attractive-looking pictures, however, is not likely to be the only application you will have for the graphics capabilities of your micro. With the high resolution available on many of the new machines – i.e., the 640 by 400 pixel resolution on the new Atari 520STM – and even the medium resolution on older machines you can consider various forms of real-world modelling.

Dedicated programs to accomplish it are often found only on more expensive machines – often under the name CAD/CAM, Computer Aided Design/Computer Aided Manufacturing – and relate largely to the design of three-dimensional shaded widgets in colour on-screen.

Again, however, such software is more often an applications solution rather than a programming solution to the problems of real-world modelling. You are simply given the tools for design, just as a draughtsman might be given an easel and a set of rulers and pencils, and you go ahead and draw.

A far more interesting idea has been developed in recent years with the discovery of a mathematical model of the real world known as a fractal. Fractals work on much the same principles as calculus; by chopping sections of a picture into its smallest possible components you will be able to obtain a series of straight lines from which it can be built.

Fractal drawings can be used to develop a highly-realistic image. The kind of work that can be done on bigger machines using fractal modelling techniques is helping designers to replace the conventional drawing board with sophisticated design and simulation systems.

Geof Wheelwright.

Figure 1 – Commodore listing

```
10 PRINT CHR$(147)
12 POKE 53280,0:POKE 53281,0
15 Q=1484:C=55756
40 S=1
50 GET B$: IF B$="" THEN 50
60 IF B$=CHR$(17) THEN Q=Q+40:C=C+40
70 IF B$=CHR$(145) THEN Q=Q-40:C=C-40
80 IF B$=CHR$(157) THEN Q=Q-1:C=C-1
90 IF B$=CHR$(29) THEN Q=Q+1:C=C+1
95 IF Q>2023 THEN Q=Q-40:C=C-40:GOTO 50
97 IF Q<1024 THEN Q=Q+40:C=C+40:GOTO 50
100 POKE Q,27
110 POKE C,S
120 S=INT(RND(1)*15)
130 GOTO 50
```

Figure 2 – Spectrum listing

```
10 CLS
11 INK 7
12 PAPER 0
13 BORDER 0
14 CLS
15 LET X=11:LET Y=16
20 PRINT AT X,Y;"!"
30 LET B$=INKEY$: IF B$="" THEN GOTO 30
40 IF B$="A" THEN LET X=X-1:PRINT AT X,Y;"!"
42 IF B$="S" THEN LET X=X-1:PRINT AT X,Y;" "
45 IF B$="Z" THEN LET X=X+1:PRINT AT X,Y;"!"
52 IF B$="X" THEN LET X=X+1:PRINT AT X,Y;" "
60 IF B$="N" THEN LET Y=Y-1:PRINT AT X,Y;"!"
62 IF B$="J" THEN LET Y=Y-1:PRINT AT X,Y;" "
70 IF B$="M" THEN LET Y=Y+1:PRINT AT X,Y,"!"
72 IF B$="K" THEN LET Y=Y+1:PRINT AT X,Y;" "
80 GOTO 30
```

Your COMPUTE

The jargon which pervades the comp language than a tool to effective com know. By following John Lettic can be fluent in the hi-tech pa

■ **IBM:** International Business Machines is what the world's largest microcomputer manufacturer would claim IBM stands for. Other suggestions are Incredibly Boring Machines, Imperialism By Marketing and Infinite Bucks Made.

■ **IC:** Integrated circuit, which is a collection of transistors, capacitors and so on built on to one small silicon chip. Also Investors' Chronicle.

■ **IF . . . THEN:** One of the basic constructions of structured programming, telling the computer that IF a certain condition exists THEN it should do something specified. Most Basics have an ELSE addition to the construct, allowing the computer to potter on without showing the slightest bit of initiative itself. Easy life.

■ **Impact printer:** A printer designed specially to survive being thrown out of windows, or other comparable impacts. Alternatively, a printer which transfers the image to paper by hitting an inked ribbon.

■ **Increment:** To increase a sum by adding a number is known as an increment. No connection to annual increment, which is a sum added to your salary infrequently and under protest by your employer.

■ **Initialise:** Initialisation is the process undergone when a bank cashier notices you have put the wrong date on your cheque. It can also mean setting the value of a variable some time before its use.

■ **Input:** The raw material, or garbage, sent to the computer through the keyboard or other peripheral device, to be acted on. See **Error message.**

■ **Instruction:** A part of the computer manual referring obscurely to a part of the computer which has been fitted/has been modified since the manual was printed/has fallen off. Also a command in a Basic listing. See **Error message.**

■ **Instruction set:** A manual which has not been collated by the manufacturer, or the complete set of instructions a given processor can understand.

■ **Integer:** A whole number, whether negative, positive or zero.

■ **Integer variable:** A variable which can only be a whole number, but as it is variable, one you can't remember.

■ **Intel:** A company responsible for the CPUs in the best-selling IBM PC series. Despite this, Intel is a highly successful manufacturer.

■ **Intelligent device:** Any device which has processing power of its own, as opposed to a dumb terminal, which can only send and receive data from elsewhere. By this definition, the merest simpleton or status quo fan/Arsenal supporter is "intelligent".

■ **Interface:** Varying degrees of assault, ranging from joystick interface – three months – through to printer interface – life imprisonment, no remission. Also the link between two devices, for example allowing a micro to drive a printer.

■ **Interpret:** To change a high-level language like Basic into machine language understandable by the computer.

■ **Interrupt:** A way of stopping the execution of a program for a small amount of time while another action is performed, giving the impression that two things are being done at the same time.

■ **Inversion:** In binary, turning a number upside down so that all the ones become zeros and the zeros ones. You can't easily do this in decimal, unless you have a dodgy accountant.

■ **I/O:** An expression first used by Walt Disney – "I/O, I/O, it's off to work we go . . ." – but now used to mean input/output. The various sockets allowing a computer to communicate with the outside world are known as I/O ports.

■ **Iteration:** An iteration is one cycle of a loop, while the process of iteration means repeating cycles over and over again. You probably know someone who does this, i.e., an irritant, and iterant is a corruption of this. An itinerant is someone who is so irritative/iterative that they are forced to move house a good deal.

■ **Joystick:** A device used for input by people whose fingers, through thumping the cursor keys too hard, have grown too dumpy and calloused for them to use the keyboard.

■ **Jump:** The same as branch, i.e., a point in a program where it can go in two directions. Jump sounds much faster and is therefore used if the program is appallingly slow.

■ **Junction:** A point where railways branch away from one another, or where electrical lines do likewise. The computer-related meaning is far too esoteric to worry about but there aren't many Js.

■ **Justify:** Having to explain the entry above to the editor. Also a technical term used by manufacturers when their machines are late. See hook. See line. See Sinclair. Of printed text, it means having the text aligned on both sides. Left-justified means aligned on the left, right-justified aligned on the right. There aren't many Js, are there?

■ **K:** Short for kilobyte, from the Greek kilo, which means 1,000. Computers, however, understand it as meaning 1,024. See **Arithmetic.**

■ **Key:** You used to be able to tell the difference between com-

R COURSE

r industry seems more like a foreign
nications for those who are not in the
uide to computer-speak you, too,
nce spoken by computer people.

puter keys and door keys by the way you couldn't lose the former. Then Sinclair launched the Spectrum Plus.

■ **Keyboard:** A device used to trap fluff, dandruff, cigarette ash and biscuit crumbs which would otherwise make a mess of the desk.

■ **Keyboard scan:** The act of looking for the biscuit crumb which makes the scrunching noise whenever you hit the Return key. Alternatively, the scanning of the keyboard by the CPU at regular and very small intervals to see if you have pressed a key. If you think about it, typing, "Oi, I've pressed a key!" wouldn't help.

■ **Keystroke:** A press of a key on the keyboard, or a heart attack suffered while searching for your key. See **Key**.

■ **Keyword:** A command the computer recognises as needing a response defined in the computer operating language, or an expletive used when looking for your keys.

■ **Kill:** Synonomous with delete and erase, invented by a programmer who read many Marvel comics. He tried Kapow! as well, but it didn't catch on.

■ **Kludge:** An improvised or botched 'quick fix' to hardware, or the sound made when you hit your dealer with the said botched hardware.

■ **Label:** A number of letters or numbers used to mark a part of a program, often used in assembly listings.

■ **Language:** A collection of words, symbols or unintelligible squiggles which, when used in the correct order, are understood by the computer. As the language is defined by the programmer, there is no logical reason why a computer language should make sense to real people. That's why they don't.

■ **Large scale integration (LSI):** See bus. see bussing. Also a small but dense integrated circuit. The progression from LSI is VLSI, or Very Large Scale Integration.

■ **LCD:** Liquid crystal display, a kind of screen display which is light, draws little power and is therefore used on portable computers.

■ **Leader:** The non-recording section at the beginning of a tape.

■ **Least significant bit:** The key on your computer with "}" and "]" on it, or in a numbering system the bit – in binary the rightmost – which has least effect on the number's total value.

■ **Least significant character:** A junior programmer, or the character at the far right of a number or word.

■ **Least significant digit:** Your little finger, or the part of a number, usually the right-most, which makes least difference to its value.

■ **LED:** Light emitting diode, one of those dinky little lights which tells you your battery is going flat.

■ **Library:** A collection of subroutines and programs commonly used by a computer. Computers can't afford their own books.

■ **LIFO:** Last in, first out. The usual way for a computer stack to operate – the last information placed on the stack is the first to be operated on. Hospitals don't work this way.

■ **Light pen:** A small pen-like device used by amateur Darth Vaders to zap information on to a CRT line driver. See **Junction**. Also an amplifier used to increase a signal between devices when they are far enough apart for the signal to degrade on the way.

■ **Line feed:** A character sent to a printer – see production editor – or display to make it move to the next line down.

■ **Line noise:** Roaring sound heard in zoos when the lions' feed is late. Also crackling on telephones or other data lines which interferes with the signal carried.

■ **Line printer:** A printer which produces a whole line of text at a time. Not to be confused with a lying printer, who produces no text at all but claims he has an excuse.

■ **Lisp:** A language uthed in artifithial intelligence applicathionth.

■ **List:** A printout or listing of a program, the command used to obtain a listing, or what happens to your desk when you put an Incredibly Big Machine (see **IBM**) on it.

■ **Listing:** You haven't moved that IBM yet, have you?

■ **Load:** Technical term meaning to put an IBM somewhere, or to enter data into the computer from a disc or tape.

■ **Loader:** A short program used to load other programs, often machine code data, into memory. Also an IBM warehouseman.

■ **Loading error:** You should have moved that IBM when I told you to do so. Also an error caused by whatever you are loading the program from not getting through to the computer, or vice versa.

■ **Local variable:** A variable used only in one part of a program, and invalid elsewhere.

■ **Location:** A cell of memory used for data storage. If you think of a character set as a collection of actors filming on location you will get completely confused.

■ **Logarithm:** A musical system used by lumberjacks working in unison, or a way of keeping track of numbers which are far apart. Log tables are made from trees with square roots.

■ **Logic:** The parts of hardware which govern logic operations in a computer, or the system by which philosophers avoid gainful employment.

■ **Log-on:** The act of signing on to an electronic mail service or database. Also a technical term used by lumberjacks.

■ **Look-up table:** A table of values which will be used in a program directly rather than being calculated by the computer first.

■ **Loop:** A sequence of instructions to the computer which is repeated for a set number of times, or forever, depending on how good a programmer you are.

Machine code sub-routines

Writing machine code programs in small sections has a number of advantages. David Janda introduces the techniques involved when using sub-routines within machine code programs.

Programs of any considerable size are usually made up of smaller sections. In Basic, GOSUB or PROC is used to transfer program control to a section of the program which does a specific task. A large program, then, is often made up of smaller subprograms which are called routines.

Writing a machine code program of only 1K in length is a real task, so it is a good idea to break the program into smaller routines. Some of those routines may be applicable only for the program but others may be useful in other programs, so if written correctly it is possible to build a library of routines for future use.

The structure of a machine code routine can be described as a piece of self-contained code which performs a certain task. The routine should have one entry and exit point. The exception to this rule is when the routine. The diagram – figure one – shows the program flow for a self-contained subroutine and one which in turn calls another.

It is considered good programming practice that if subroutine A calls B and that B in turn calls C, that at the end of C control should then pass back to the next instruction in B, and so on. In other words, try to ensure

One of the best ways to learn machine code is to write small routines and the best way to use them is from within a Basic program.

that program flow does not skip past any preceding subroutines on the way back to the main program. That will ensure modularity and portability.

Writing machine code programs in smaller portions has a number of advantages. First, it makes the whole program easier to understand. It also makes the program easier to debug, as problems can be tracked down to a specific routine which can then be checked independently of the main program. Time in developing other programs can also be saved by writing small routines which may be used in more than one program.

One of the best ways to learn machine code is to write small routines and one of the best ways to use them is from Basic – that is, writing a Basic program and calling a machine code routine when necessary.

Basic is a general-purpose language and lacks speed. Writing a program in Basic and machine code is useful if, for example, there is a task which needs to be carried-out very quickly – graphics being one example. This combination of two languages – one high-level and the other low-level – is often referred to as hybrid programming.

Basic has a number of built-in commands and functions which facilitate that. PEEK and POKE, which are used to read and write to memory locations directly, can be used to pass values – results and so on – to and from machine code routines. On the Commodore 64, SYS n is used to transfer program flow to a machine code program starting at memory location n. The BBC micro has a more powerful command, CALL. That not only transfers control to a machine code program but also enables parameters to be passed to the A, X and Y registers of the 6502 microprocessor. A function common to practically all Basics is USR. It can be used in two ways.

```
LET V=USR n
or
PRINT USR n
```

where v is a numeric variable and n is the address of the machine code program to be called. The difference between USR and SYS or CALL is that it allows one parameter to be transferred from the machine code program into a Basic variable, or printed on the screen. The parameter is passed from one or more of the CPU registers depending on what machine it is.

When calling a machine code routine it is best to assume nothing. That is, you should not expect any of the CPU registers to contain a specific value unless you put it there. That can be something of a problem if, for example, routine A needs to call routine B without having any of the registers altered. One method round that is to store the contents of all the registers temporarily at the start of a routine and then restore the registers to their original contents at the end.

The stack is used for the operation and can be considered as a temporary storage area. To preserve the registers and restore them in Z-80 code, the instructions shown in figure two would have to be done.

In 6502, things are a little simpler as figure three illustrates.

There are various ways in which a machine code routine can be called from another machine code routine. In Z-80, the simplest manner is to use CALL a. That is an unconditional jump to the subroutine at the address a. The routine which was called should end with RET – short for return – which transfers program control to the next instruction after the CALL.

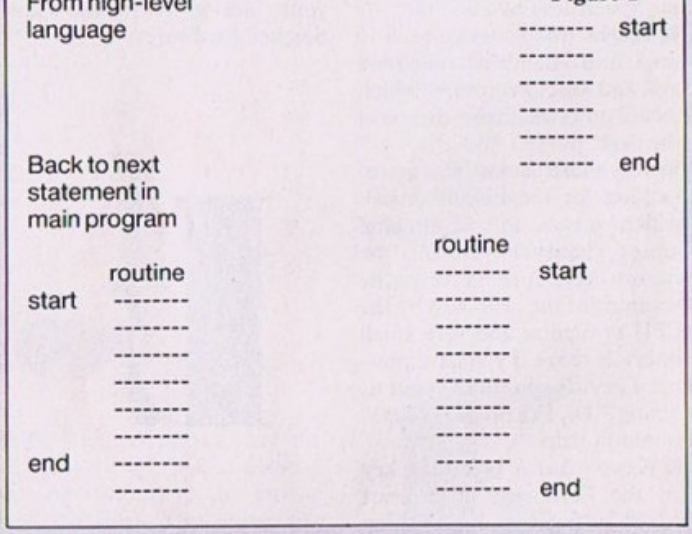
There are several variants of the CALL and TER instructions. They include CALLs and RETs which will be executed, depending on the state of various bits in the status register.

The 6502 uses the JSR – Jump SubRoutine – and RTS – ReTurn Subroutine – in place of the CALL and RET in Z-80. There are no variants of those two instructions.

Both the Z-80 and 6502 microprocessors have, as part of their instruction sets, instructions which are similar to Basic GOTO. They can be used in one of two ways, absolute or relative. In the absolute mode a jump is made into a specific address and in the relative mode control is passed to an instruction x number of bytes from the relative jump – no more than 256 bytes in either direction.

It is a good idea to keep machine code routines as portable as possible, that is, write them so they can be used at different places in memory. To

Figure 1



do that it is necessary to try to use relative jumping as much as possible rather than use absolute addressing. This idea is similar to Basic GOSUB. GOSUB requires a line number – it is absolute. Because of that the subroutine must always be at that line number. BBC Basic provides PROC, which is followed by the name of the subroutine; therefore the subroutine can be anywhere in the program. So PROC is a relative method of calling a subroutine.

INSTRUCTION	FUNCTION
PUSH HL	Save contents of H and L
PUSH DE	Save contents of D and E
PUSH BC	Save contents of B and C
PUSH AF	Save contents of A and F
CALL SUB	Call subroutine
POP AF	Restore original contents to A and F
POP BC	Restore original contents to B and C
POP DE	Restore original contents to D and E
POP HL	Restore original contents to H and L
PUSH	stores the contents of a register on to the stack while
POP	removes it

Figure 2

INSTRUCTION	
PHA	Save contents of A
PHP	Save contents of status
SUB	Call subroutine
PLP	Restore status
PLA	Restore A

Figure 3

Z-80 EXAMPLE

ASSEMBLER	COMMENTS
LD HL,FROM	Load HL with source address
LD DE,DEST	Load DE with destination address
LD BD,AMNT	Load BC with amount of bytes to be transferred
LDIR	Transfer data
RET	Finished

The example is a general-purpose copy routine which can transfer a section of memory. Registered pairs HL, DE and BC are loaded with the start address of the data to be copied, the start address where it is to be copied to, and the amount of data in bytes to be copied. The LDIR instruction is one of the most powerful in the Z-80 instruction set. The contents of the memory location pointed to by HL are transferred to the location pointed to by DE. The contents of HL and DE are then incremented by one of the contents of BC decremented by one. If BC does not equal one, the set of instructions is repeated until it does.

6502 EXAMPLE

LDA #$LB1	Load accumulator with low byte of first number
CLC	Clear carry flag before adding
ADC #$LB2	Add low byte of second number to accumulator
STA $200	Store result at memory address 200
LDA #$HB1	Load accumulator with high byte of first number
ADC #HB2	Add high byte of second number to accumulator
STA #201	Store result at memory address 201

This routine will add two two-byte numbers and store the results at memory locations $200 and $201, with the low byte at $200. The routine works by performing two one-byte additions. – *David Janda.*

Amstrad sound generator

Almost all applications can benefit from the use of sound, whether it be zaps and bangs in games or warning bleeps and buzes in application packages. In a small Basic program, handling sound should not be difficult but if you are short on memory, making music by machine code may be the answer.

Sound on the Amstrad is achieved through the AY8912 programmable sound generator. The device is capable of producing a variety of sounds with its three independently-controllable sound channels, A, B and C which produce pure done. That can be done by counting the drames or by using one of the timers.

While the sound is playing, execute another part of the program. At the end of a subtones, or white noise.

Once a sound has been set up the sound chip will take over, leaving you to do other tasks. That does not mean that you can program the chip with Beethoven's Fifth and leave it to its own devices. Only one sound per channel may be programmed at any time. To play a complete tune and do something else, some basic steps need to be followed:

Define a format for your second data which is to be read and placed into the appropriate registers. Each piece of data should include the number of the register to be loaded, together with the duration for which the note should be played.

Set up or amend the appropriate register(s).

Activate the appropriate channel, noting the time it was

routine, check to see if the desired duration of the sound has expired. This is a very rough algorithm and needs finer details before trying it.

Programming the sound generator chip directly as described is possible and useful at times but is not encouraged by Amstrad. Instead, the official entry points, of which there are 11, are recommended.

PSG REGISTERS		SOUND ENTRY POINTS	
R0	High notes value Ch A	RESET	BCA7, 1E68
R1	Low notes value Ch A	QUEUE	BCAA, 1F9F
R2	High notes value CH B	CHECK	BCAD, 206C
R3	Low notes value Ch B	ARM EVENT	BCB0, 2089
R4	High notes value Ch C	HOLD	BCB6, 1ECB
R5	Low notes value Ch C	CONTINUE	BCB9, 1EE6
R6	Noise value	RELEASE	BCB3, 204A
R7	Tone enable/disable	AMPL ENVELOPE	BCBC, 233B
R8	Ch A volume	TUNE ENVELOPE	BCBF, 233D
R9	Ch B volume	A ADDRESS	BCC2, 2349
R10	Ch C volume	T ADDRESS	BCC5, 234E

Pascal – two popular implementations

Geof Wheelwright reviews Pascal compilers from DR and Metacomco.

Most programmers go through the quest for the perfect language. After moving from Basic, most will sample a little Forth, take a trip down memory lane with Fortran and even list along with Prolog.

For most, howerver, there will inevitably be a prolonged if not permanent stop when they reach Pascal. It has long been considered a powered, structured high-level language which had much to offer the experienced programmer – long, in this case, being a little more than 12 years. The definitive work on the language was published in 1974 under the name *Pascal User Manual and Report* and outlined how the language would operate.

Since then, and particularly since the advent of the home micro, Pascal has become a popular portable language for writing software which would move easily from one machine to another. To give some idea of how that has translated into real-live software, we offer the following examples of Pascal software on two popular machines; the Amstrad 8256 and the Sinclair QL.

The Digital Research Pascal/MT+ for the Amstrad 8256 is one of the most recent versions of the language. It claims to be a full implementation of Pascal as set down in the International Standards Organisation standard DPS/7195 but also includes several extensions which enhance I/O operations, allow additional data types, offer access to the run-time system and offer modules and overlays.

The system includes a compiler, linker and programming tools and has been developed to run on a variety of operating systems and processors, the Joyce version of CP/M and its Z-80 processor being among them. DR claims a high degree of portability for this Pascal as it is consistent among its implementations, thus permitting you to switch easily between processors and operating systems. Unlike many such systems, Pascal/MT+ will let you generate software for use in a ROM-based environment, to operate with or without an operating system.

Expressions and assignments can handle input respectively from and to I/O ports; long and short integer data types are supported; and interrupt, external and assembly language procedures are included.

QL Pascal by Metacomco was one of the first to develop an extensive line of programming languages for the Sinclair QL and subsequently has made use of that 68000 expertise to develop for the Atari ST series of machines.

Its Pascal was particularly popular as it arrived at a time when powerful development languages for the machine were still relatively thin on the ground. Like the Digital Research offering for the Amstrad, QL Pascal is a high-specification implementation of Pascal designed to meet ISO 7185 standard, while taking full advantage of the target machine; in this case that means direct access to the QL Qdos windows, graphics, traps and file-handling operations.

The built-in compiler is a fast, single-pass job which produces native 68000 code. Metacomco claims compiled programs are compact and efficient, easy to use with assembler and do not require special run-time environments.

QL Pascal can address all the memory space available on the QL – up to 512K on an expanded machine – and will take variable names of any reasonable length, as well as 32-bit integers. Error-handling information is also reasonable, with more than 150 errors recognisable to the compiler and more than 30 English-language error messages generated by the run-time system.