PROJECT

AN INTERPRETER FOR THE

Beginning with an explanation of the principles used to translate high-level languages into machine code, John Dawson continues his series by outlining the structure of a control-orientated interpreter.

IN THE SERIES so far, we have looked at the general principles of closed-loop feedback systems and the similarities between control systems in biological and computer applications.

In the first article, I showed how a radio transmitter could be used to send one form of control information from a computer to a remote device. In the second article, I presented some ideas for interfacing the signal from a radio-control receiver to a hefty DC motor.

I used a common-or-garden car windscreen wiper motor and a comparatively simple electronic circuit to show how an effect — regulating the speed of a DC motor — can be achieved with the minimum cost and time spent in construction.

The extent to which you may wish to go beyond the ideas in the article to a finished, definitively-engineered product is entirely up to you — there are many reference works on control electronics which will help you to improve, say, the pulse-width modulation circuit. You might, for example, include feedback control of the motor speed to make it independent of the load on the motor.

Exhausted by analogue electronics and metalwork, this month we shall examine the foundations of a control-orientated interpreter for the Microtan. I shall indicate which sections of the machine code are specific to the Microtan so that those with access to other machines with 6502 CPUs may modify the code accordingly.

Some form of go-between is required to cross the no man's land between the language of humans and the internal machine-code instructions obeyed by a computer. People recognise and understand the relationships between numerals, characters and symbols; the internal store of the computer, the various electronic circuits and logic elements are able to respond only to binary patterns of on or off electrical impulses.

The binary machine-code instructions used by a computer are known generally as object code, while a series of instructions written in a high-level language is known as source code.

The purpose of every language other than machine code is to create a bridge between instructions we can write and recognise and the binary machine code which will achieve the desired effect. A program written in Basic will calculate results through a series of machine-code instructions which are invisible or transparent to the user.

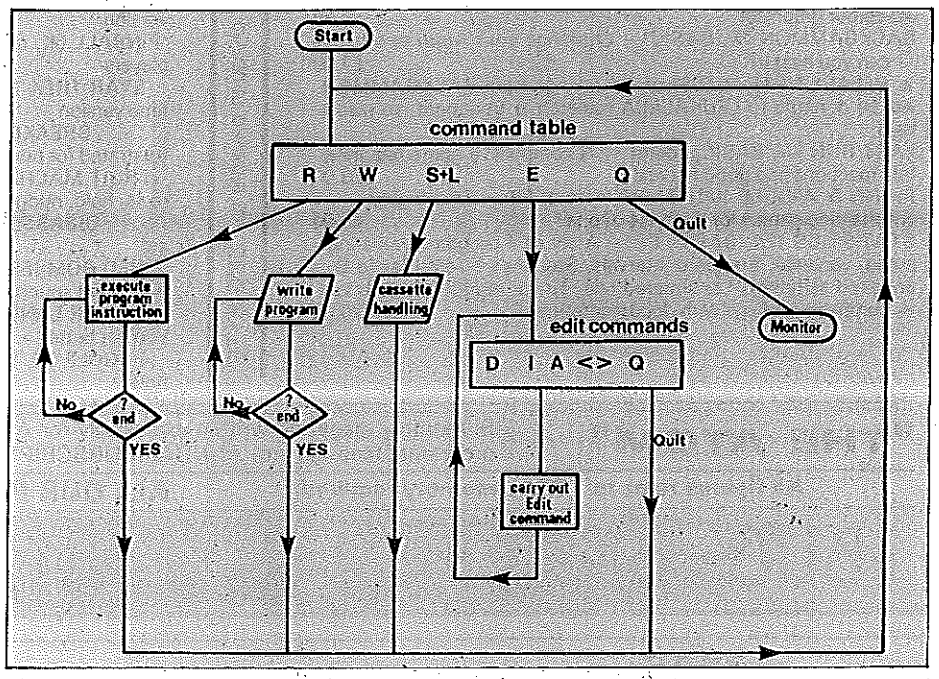


Figure 1. The minimum elements for a high-level interpreter.

There is a hierachy of computer languages ranging from fundamental machine code, in which each instruction entered by the operator produces one instruction to be executed by the computer, to high-level languages where a spoken command may initiate sustained computing activity.

For example, reports from the States already suggest that intelligent microwave ovens are a distinct possibility. The oven would contain a program to allow it to respond to the instruction sequence:

"Cook the pork casserole for 10 minutes or until the thermometer probe has reached 110 degrees for three minutes; Tell me when it's done — I'm in the sitting room".

"Cook" is a reserved word in this instruction sequence and "pork casserole" a variable. "thermometer probe" is an input device and the alert at the end of the process is to be routed to an output device in the sitting room.

There are many high-level computer languages and it is possible to debate endlessly and fruitlessly which is the best language. There is no best computer language — computers exist to solve problems and a computer language is purposeless until put to some task. Certain languages are more successful for some applications than others.

Cobol, or COmmon Business-Orientated Language, was designed for general commercial use, originally under the sponsorship of the U.S. Department of Defense. Program steps in Cobol are specified by means of instructions expressed in stylised English statements which can be recognised by the compiler and translated into a sequence of machine-code instructions.

The program statements consist of reserved words which have a special significance, enabling the compiler to generate the appropriate machine instructions for the particular operation required, and identifying labels used by the programmer to refer to units of data. For example, the Cobol statement:

Add London-weighting To Salary

uses the reserved words "Add" and "To" to generate the machine coding which will perform an addition in which a quantity, London weighting, is added to another quantity, Salary.

Specialised, high-level languages have been written for many purposes such as engineering design tasks, medical information storage and retrieval, the control of British Telecom's new automatic telephone exchanges, and scientific and industrial control applications.

High-level languages fall into two groups; interpreters and compilers. Some languages may be available in both forms. I have used the word "compiler" already and there is an important distinction between the two types of high-level languages.

The way in which an interpreter works is

FOR CONTROL

Nevertheless, the work on Cogent has been valuable if it showed you that writing a language for a defined area of computing need not be difficult and can produce useful results in applications where you want to be able to arrange a series of high-level instructions to carry out a particular set of tasks.

Although I have not touched on it in the series, you might decide, for example, to write a fast special-purpose interpreter using single-letter high-level commands for handling nothing but machine-code graphics routines for animating games programs. You should also have discovered that there are many more ways of in- and outputting information than by using a simple keyboard and VDU.

The EPROM version of Forth will coexist with Basic in the language board from Microtanic. The single-sided printed-circuit board shown in the photograph measures about 4in. by 2.5in. and has sockets for five 4K EPROMS. It allows you to switch, manually or by software commands, between Forth, Microtanic two-pass assembler, or Basic. It can be used either by plugging it directly into the J2 socket on the Tanex board and allowing it to ride the board piggy-back fashion, or by bringing a 24-way cable with DIL connectors at either end from the Tanex board to the language board. The language board would then be mounted in another part of the computer system.

Intra-system bus

Microtanic Software does not give definite information about the maximum length of cable which can be used before you can expect problems from reflections up and down what is essentially an intra-system bus and piggy-back boards are never mechanically desirable although the one in my system has worked consistently and without fault from the moment of its installation.

One problem is that the language board increases the width of the Tanex to the extent that it is impossible to plug another card into the first expansion slot on the motherboard. My motherboard had four connectors rather than the full 12, and it seems that sockets for the back plane are now as scarce as buffers and memory chips were a year ago.

However, your problems do not end when you have found the socket, for you can expect problems with the paging logic. The instructions issued by Tangerine Computer did not work with the motherboard in my system. In the end I cut the board-select line which stayed stubbornly at +4V and wired the select line permanently low to ground.

The Tangerine documentation has slipped in the past few months and although the manual for the new Tanbug is available, I have not seen a copy. I hope Tangerine will be able to regain the position it had of excellent support for a good computer.

Overall, Forth is a fascinating, sophisticated, easy-to-use software tool with a multitude of possible applications. If you are prepared to invest in a book such as *Starting Forth* or to write to a Forth interest group, I think you can acquire a significant addition to your system which will allow you a new flexibility in programming. In particular, I think Forth should be a good way of approaching subjects closely related to the theme underlying this series of articles.

1982 is Information Technology Year and one of the purposes of the Health Section of the IT82 Committee is to encourage volume production and marketing of microelectronic aids for the disabled and to reduce prices to a level where disabled people will be prepared to buy the devices or tools.

Many disabled people cannot type because they have insufficient strength in their hands. Alternative Input devices have been developed, for example a suck/blow switch can be connected to a single-bit input on the computer and that single binary input can be used to control a word processor or a computer system which will control the heating, lighting, telephone and other electrical equipment in a disabled person's home.

Forth is an ideal language in which to write routines to take information from special-purpose switches — perhaps by bouncing infra-red light off the operator's eye to pick up eye movements and blinks — to control both physical devices and the operation of ordinary commercial software in a microcomputer.

If you start thinking about designing a switch or device for use by somebody who is less able than an ordinary person, a number of factors emerge which you will have to take into account. A switch should give some feedback to the user to indicate that the switching action has taken place.

For example, some keyboards require an increasing pressure on the key until the switch has occurred and then there is a sudden give to let you know that this has happened. Other keyboards provide an audible feedback by bleeping when a key has been pressed.

It is important that operating a key repeatedly should not make the operator tired and should not cause cramp if a person is using one finger or a toe or even a stick attached to their forehead to operate the keys. Remember that this is even more important to the disabled because they are less able than an ordinary person to switch tasks to another part of their body. You can open a door with your foot if your arms are full.

Consistent response

Above all, the machine must be consistent in its response for there is nothing more frustrating to someone who is absolutely dependent on a machine than to have an inconsistent and arbitrary response. Good tools are always predictable in their actions and it is this quality that allows fine control by craftsmen and skilled workers.

You cannot iron clothes with the flat of your hand — you need a tool to do the job. In the same way, a disabled person may require a special telephone or a word processor to communicate. These aids should be available at ordinary retail outlets where you might buy an electric iron.

The combination of Forth, given some additional words in the dictionary for real-time control, with some inventive thinking and clever production engineering could generate a set of tools for the disabled. For the first time, it would be possible to build machines which could be extended and adapted by a person to perform a specific job or task.

Comparatively low-cost machines could be sold for disabled people with the capacity to carry out a wide range of jobs. The user could mould the functions of the machine to suit their own purposes. ■

Listing 2.

```
: STAR 42 EMIT ; OK
: STARS 0 DO STAR LOOP ; OK
: MARGIN CR 20 SPACES ; OK
: BLIP MARGIN STAR ; OK
: BAR MARGIN 5 STARS ; OK
: F BAR BLIP BAR BLIP BLIP CR ; OK
: BLIPS MARGIN STAR 3 SPACES STAR ; OK
: EIGHT BAR BLIPS BLIPS BAR BLIPS BLIPS BAR CR ; OK
```

```
EIGHT
      *****
      *   *
      *   *
      *****
      *   *
      *   *
      *****
OK
```

```
F
      *****
      *
      *****
      *
      *
OK
```

(continued from previous page)

shows the entry point to the command table labelled "Start". The first task an interpreter should perform is to clear the screen and then display a message to tell the user that control has been passed from the system monitor to the high-level language and that the interpreter program is running.

After removing any previous information from the screen, the interpreter could then call up the subroutine set out in figure 3 which will display a message at the top of the Microtan screen.

A flowchart for the operations is set out next to the listing in figure 3. With the aid of the Microtan manual, you should be able to trace through the machine-code instructions, working out the purpose of each and its relationship to the whole.

The address of the line on the screen along which the cursor is located is stored in the zero-page locations 0A Hex and 0B Hex and the position of the cursor along that line is stored in zero-page location 03 Hex. The monitor subroutine at FE75 Hex loads a character into the next sequential position along the VDU line from the CPU register A.

The message to be displayed on the screen is stored in memory locations pointed to by locations 48 and 49 Hex plus the contents of index register Y. For example, the first message put on the screen is stored at 0F00 Hex and locations 48 and 49 contain 00 Hex and 0F Hex — remember that the low-order byte is stored first.

If Y contains 00 Hex when the subroutine is entered, that message will be displayed byte by byte until a null value — 00 Hex — is encountered. The second message in the program from which this is taken is stored at 0F20 Hex and Y must be loaded with 20 Hex before the SR is called.

List 2 is a subroutine to clear a section of memory from 1010 Hex to 2000 Hex. You can use it to set a part of the computer RAM to a known state before a new program or text is written into it.

List 3 is the start of a word-processing program called Asimov I am writing at present. Interpreters need editors, and there are marked similarities between word-processing programs and the more primitive text editors used to alter program lines and insert new instructions. You can tell that the program is still in an experimental phase from the untidy way in which reference values are loaded into zero-page locations.

However, two of the subroutines I have mentioned are called in the first few lines and the program continues to fetch a character from the Microtan keyboard, 0D33, and then seeks for a match between the character and a number of stored command letters. If a match is found, the program jumps to one of the functions set out in the command table of figure 1; if not, the program is re-started.

Finally, you may find the chart illustrated in figure 4 useful for keeping track of zero-page locations when developing your own special-purpose interpreter or any other low-level program. The best book I have found on 6502 assembler programming is the extraordinarily lucid, well-written *6502 Software Design* by Leo Scanlon. ■

```
0540 48     PHA
0541 8A     TXA
0542 48     PHA
0543 98     TYA
0544 48     PHA
0545 A50A   LDA $000A
0547 48     PHA
0548 A50B   LDA $000B
054A 48     PHA
054B A920   LDA ##0020
054D 850A   STA $000A
054F A900   LDA ##0000
0551 8503   STA $0003
0553 A902   LDA ##0002

0555 850B   STA $000B
0557 B148   LDA (#0048),Y
0559 C900   CMP ##0000
055B F006   BEQ $0563
055D 2075FE JSR $FE75
0560 C8     INY
0561 D0F4   BNE $0557
0563 A920   LDA ##0020
0565 A403   LDY $0003
0567 910A   STA (#000A),Y
0569 A900   LDA ##0000
056B 8503   STA #0003
056D 68     PLA
056E 850B   STA $000B

0570 68     PLA
0571 850A   STA $000A
0573 68     PLA
0574 A8     TAY
0575 68     PLA
0576 AA     TAX
0577 68     PLA
0578 60     RTS
```

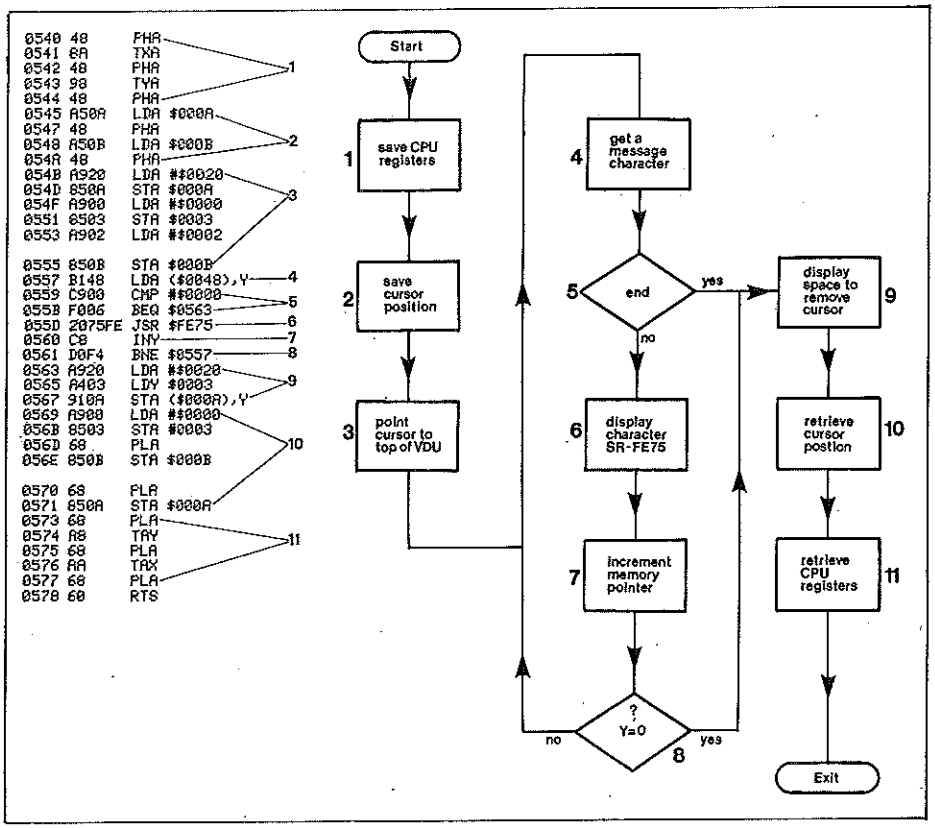
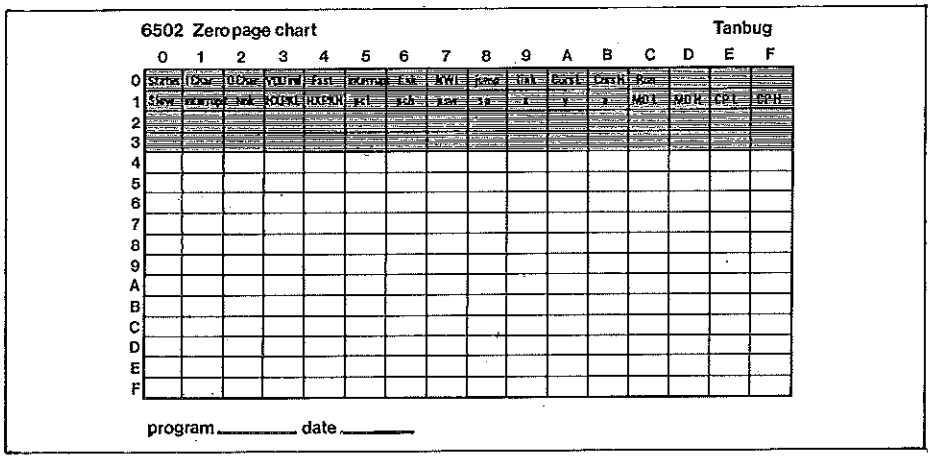


Figure 3. Subroutine to display a message on screen.

Figure 4. Chart for zero-page locations.



Listing 1.

```
0500 48     PHA
0501 8A     TXA
0502 48     PHA
0503 A920   LDA ##0020
0505 A200   LDX ##0000
0507 9D0002 STA $0200,X
050A 9D0003 STA $0300,X
050D CA     DEX
050E D0F7   BNE $0507
0510 EA     NOP
0511 68     PLA
0512 AA     TAX
0513 68     PLA
0514 60     RTS
```

Listing 2.

```
0670 48     PHA
0671 8A     TXA
0672 48     PHA
0673 98     TYA
0674 48     PHA
0675 A910   LDA ##0010
0677 854E   STA $004E
0679 A910   LDA ##0010
067B 854F   STA $004F
067D A000   LDY ##0000
067F 98     TYA
0680 914E   STA (#004E),Y
0682 C8     INY
0683 D0FB   BNE $0680

0685 E64F   INC $004F
0687 A920   LDA ##0020
0689 C54F   CMP $004F
068B D0F2   BNE $067F
068D 68     PLA
068E A8     TAY
068F 68     PLA
0690 AA     TAX
0691 68     PLA
0692 60     RTS
```

Listing 3.

```
IDOO
ODOO D8     CLD
OD01 A2FF   LDX ##00FF
0D03 9A     TXS
OD04 200005 JSR$500
OD07 A90F   LDA ##000F
OD09 8549   STA $0049
ODOB A900   LDA ##0000
ODOD 8548   STA $0048
ODOF A000   LDY ##0000
OD11 204005 JSR $0540
OD14 A900   LDA ##0000
OD16 854A   STA $004A
OD18 8544   STA $0044
ODIA 8545   STA $0045

0D1C 8560   STA $0060
ODIE A90D   LDA ##000D
OD20 8561   STA $0061
OD22 A220   LDX ##0020
OD24 8651   STX $0051
OD26 CA     DEX
OD27 8652   STX $0052
0D29 A910   LDA ##0010
OD2B 8540   STA $0040
OD2D 8541   STA $0041
OD2F A90E   LDA ##000E
OD31 854B   STA $004B
0D33 20FAFD JSR $FDFA
OD36 A501   LDA $0001

OD38 C957   CMP ##0057
OD3A D003   BNE $0D3F
OD3C 4C5304 JMP $0453
OD3F C945   CMP ##0045
OD41 D003   BNE $OD46
OD43 4CE004 JMP $04E0
OD46 C953   CMP ##0053
OD48 D003   BNE $OD4D
OD4A 4C000D JMP $OD00
OD4D C952   CMP ##0052
OD4F D003   BNE $OD54
OD51 4C000D JMP $OD00
OD54 C950   CMP ##0050
OD56 D003   BNE $OD5B

OD58 4C6004 JMP $0460
OD5B C958   CMP ##0058
OD5D D003   BNE $0D62
OD5F 4C800D JMP $OD80
OD62 4C000D JMP $OD00
OD65 00     BRK
OD66 00     BRK
OD67 00     BRK
OD68 00     BRK
OD69 00     BRK
OD6A 00     BRK
OD6B 00     BRK
OD6C 00     BRK
OD6D 00     BRK
```