Longhand/ Shorthand

A machine code program can take on several different forms. It is usually written by the programmer in the form of Assembly language, which uses mnemonics for opcodes and labels for operands, thus:

```
LDA WEIGHT
ADC FUEL
STA WEIGHT
```

We must however specify the addresses of those labels. For example:

```
FUEL = S03EE
WEIGHT = S031F
```

An assembler package would transform this into a hex dump, using a disk drive. 'Pseuco assembly language', as shown below, is less easy to read, but can often be entered into a package called a 'spot assembler', which doesn't need disks.

```
LDA S031F
ADC S03EE
STA S031F
```

A hex dump consists of a starting address (at the left) and the sequence of two-digit hex values as they will appear in memory. Note that an operand like S031F is stored in reverse order (1F 03) and that opcodes have been replaced by the appropriate hex value:

```
19C4 AD 1F 03 6D EE 03 8D 1F 03
```

obliterated by BASIC statements such as NEW.

Most home computers have some BASIC command to tell the machine to stop executing BASIC and begin executing the machine code program that starts at a specific location. One form of this command is SYS 4096 (RETURN), meaning 'transfer control to the system starting at decimal location 4096'; another is CALL $E651, meaning 'call the machine code routine starting at hex location E651'.

The machine code subroutine or program will then execute this system or routine (it may or may not produce any visible results, depending on the nature of the program). If it is correctly written and incorporates the proper terminating procedure, control will be passed back to BASIC. This means, incidentally, that it is possible to call machine code subroutines from several places in the operation of a BASIC program, whenever a function needs to be performed at high speed.

One of the difficulties of programming in machine code is that if you have made a mistake in your code, the computer won't come back with a nice helpful SYNTAX ERROR. It will more than likely 'crash' instead: the machine won't respond to anything you type. This isn't harmful to the computer, but you will have to reset it (or switch the machine off and then on again), and that usually means having to enter the program again from scratch. That's why you can't experiment in machine code as you can in BASIC — the operation of the program must be thoroughly checked on paper before it is entered into the computer.

However, a software device that can assist greatly in the entering and checking of machine code is the 'machine code monitor' (which has nothing to do with a monitor screen). This is built into the ROMs of a few computers but is generally purchased as a cassette or cartridge-based package. A machine code monitor is a simple operating system that will display on the screen the contents of any requested section of memory. These (hex) values can simply be altered or written over, so a monitor is by far the fastest way of entering a hex dump. Moreover, it usually allows you to load and save machine code programs directly onto cassette, without the need for the BASIC loader program. The most advanced machine code utility programs (the machine code equivalent to BASIC tool kits — see page 444) show the contents of each of the processor's internal registers.

Hex dumps are a convenient way of expressing machine code, but they aren't easy to read. Unless you happen to remember the hexadecimal equivalent of all the various opcodes, it's almost impossible to distinguish the opcodes from the operands. So programs are usually written using the three-letter mnemonics that we introduced in the previous article (page 449), and these are then translated into hex using a table of codes from the microprocessor's handbook.

However, a more sophisticated form of machine code monitor will allow you to type in the program in mnemonics, performing the conversions automatically. This is called a 'spot assembler' because it will assemble the mnemonics into numbers on the spot.

This leads us on to the final form in which machine code can be expressed — Assembly language — which not only makes use of mnemonics for the opcodes but can handle names (or labels) instead of hex numbers for the operands. Thus, if location $07B2 contains the current number of missiles fired in a game, we can load this into the accumulator with the instruction:

```
LDA MISSIL
```

At the start of the program we will have to specify the location of MISSIL=$07B2, and that this location should initially contain the value of $09 (nine missiles).

When we have finished developing this program in Assembly language (called the 'source code' of the program), we run a utility program called an assembler. This works through the code, replacing mnemonics and any labels with their hex equivalent, thereby creating a new version called the 'object code'. This code can then be entered into the computer's memory and run. The process is not dissimilar to compiling (see page 84), though in this case there is a one-to-one correspondence between the source and object code.

Assembly language, being a higher-level language than machine code, is considerably easier to write, but there is no loss in performance. However, assembler packages will usually only work with a disk drive, and so are not available to all home computer users.

Opcodes

Here are some more opcodes that a typical microprocessor would feature

JSR

Jump SubRoutine

This function is equivalent to BASIC's GOSUB. JSR $354D will change the contents of the program counter (PC) register so that it executes the code from $354D onwards

RTS

ReTurn from Subroutine

On encountering RTS, the processor will jump back to the location from which the subroutine was called (i.e. equivalent to RETURN in BASIC). RTS has no operand because the return address will have automatically been stored in a special area of memory called the Stack

BMI

Branch if Minus

This is one of several forms of conditional branching in machine code (in BASIC, IF . . . THEN GOTO is a conditional branch). If the result of the last operation resulted in a negative value in the accumulator, program execution will jump to a specified address BPL specifies Branch if PLus

LDX

LoaD X register

X is another single byte register within the processor, and while it cannot perform arithmetic in the same way as the accumulator, it is used for 'indexed addressing' (see panel). LDX loads a value into X, and STX (STore X) will store it back in memory

INX

INcrement X

By adding 1 to the value of X (DEX — DEcrement X — will subtract 1), and using indexed addressing, it is possible to step through a number of locations in memory, performing the same process on each