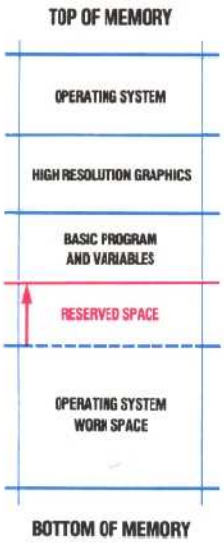
STARTING ORDERS

Where To Locate Machine Code Instructions

BBC Micro

In direct mode enter:
PRINT~PAGE
this gives the hex address of the start of your reserved space. Then enter:
PAGE=PAGE+N
where N is the decimal number of bytes you wish to reserve



When a program has been written in its Assembly language form, the machine code programmer must provide directives for the assembler at the beginning of the assembly. We look at several of these 'assembler directives' to see what functions they perform. These instructions may be used with both processors.

In the last instalment of the course, we wrote a simple machine code program that added two numbers into the accumulator and stored the result in memory. There was nothing very startling about what the program did, but, in writing it, we covered many points of significance to the machine code programmer. Let's look at the program again, with location addresses included, as if it were to be loaded at $0000, and the accumulated result to be stored at address $0009. (This is purely for example's sake: any attempt to use these particular locations would almost certainly result in an unrecoverable crash). The two versions of the program are:

Location Address	Machine Code	Assembly Language
Z80		
0000	A7	AND A
0001	3E 42	LD A,$42
0003	CE 07	ADC A,$07
0005	32 09 00	LD BYTE1,A
0008	C9	RET
6502		
0000	18	CLC
0001	A9 42	LDA #$42
0003	69 07	ADC #$07
0005	8D 09 00	STA BYTE1
0008	60	RTS

Note that the fourth instruction (which stores the accumulator contents at $0009) in both programs does not specify a destination address in the Assembly language column. Instead, it uses a symbolic address, BYTE1. In the machine code version of the instruction, however, we see the op-code for 'transfer the accumulator contents' followed by 09 00, the two-byte lo-hi form of the address $0009.

This is another aspect of the translation (or assembly) process from Assembly language to machine code. Just as we use reasonably meaningful instruction mnemonics (STA and RET, for example, instead of hex codes such as 8D and C9) because they make the programs easier for us to read and write, so we will often use symbols such as BYTE1 instead of unfriendly hex addresses or numbers like $0009. This is no different from initialising variables with constants in a BASIC program, and the reasoning is exactly the same in both cases — the program is made more readable, the possibility of errors occurring when writing such numbers is reduced, and the program is made more easily manipulable. For example, changing the statement in which the constant value is assigned to the variable in the first place will cause the new value to be used throughout the program automatically, needing no further editing on the programmer's part.

This is easy to understand when talking about BASIC programming, but where in our Assembly language program is the equivalent of the BASIC statement LET BYTE1=$0009? At present there isn't any such instruction. When we actually come to assemble the Assembly language into machine code we must remember to do this ourselves. If, however, we were using an assembler program to do the assembly for us, then we could make such an assignment statement at the beginning of the program (we give the Z80 version of the program here, although these assembler directives may be used with both processors):

Z80		
0000		BYTE1 EQU $0009
0000	A7	AND A
0001	3E 42	LD A,$42
0003	CE 07	ADC A,$07
0005	32 09 00	LD BYTE1,A
0008	C9	RET

BYTE1 is placed in a column of its own, known as the label field, which we will say more about later. In the op-code field, a new mnemonic (EQU, standing for 'equate' or 'is to be set equal to...') is used; and, in the operand field, the value that is to be assigned to BYTE1 is given (in this case, $0009).

It is important to note that although EQU appears in the op-code field, and looks like a mnemonic, it isn't an Assembly language mnemonic and doesn't belong in either the Z80 or 6502 instruction sets. Such a mnemonic is called a *pseudo-op* or an *assembler directive*. EQU tells the assembler program that 'whenever it finds the preceding alphanumeric symbol (BYTE1 in this case), it must replace it by the value that follows the directive ($0009 here)'. Remember that when we use an assembler program we write only the Assembly language program, either as a tape/disk file or directly at the keyboard, and then call the assembler program to turn it into a machine code program. The output of an assembler is usually a full Assembly listing like those we've been producing, plus the machine code program consisting simply of a string of hex