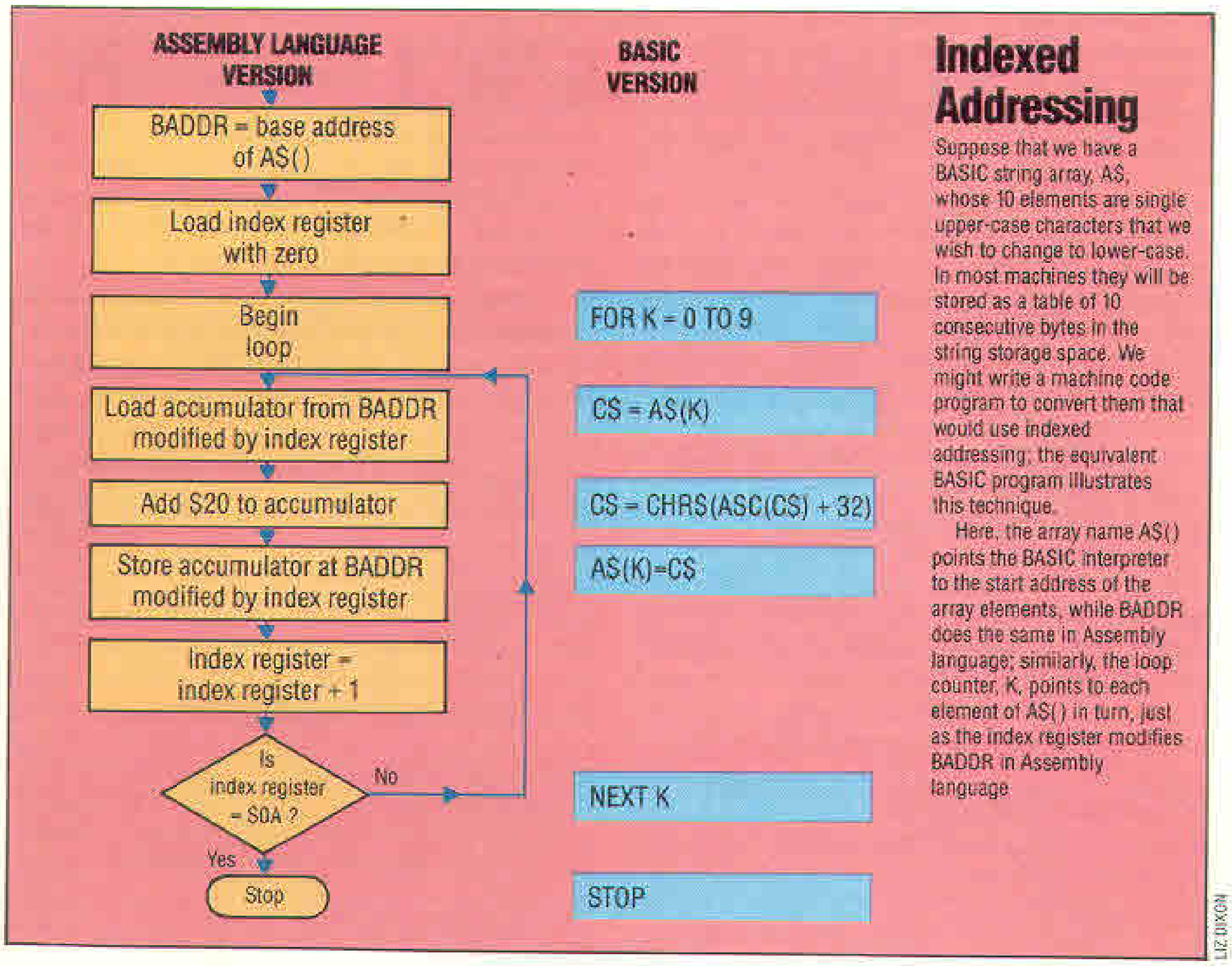
ASSEMBLY LANGUAGE VERSION
BADDR = base address of A$()
Load index register with zero
Begin loop
Load accumulator from BADDR modified by index register
Add $20 to accumulator
Store accumulator at BADDR modified by index register
Index register = index register + 1
Is index register = $0A ?
No
Yes
Stop
BASIC VERSION
FOR K = 0 TO 9
C$ = A$(K)
C$ = CHR$(ASC(C$) + 32)
A$(K)=C$
NEXT K
STOP
LIZ DIXON

Indexed Addressing

Suppose that we have a BASIC string array, A$, whose 10 elements are single upper-case characters that we wish to change to lower-case. In most machines they will be stored as a table of 10 consecutive bytes in the string storage space. We might write a machine code program to convert them that would use indexed addressing; the equivalent BASIC program illustrates this technique.

Here, the array name A$() points the BASIC interpreter to the start address of the array elements, while BADDR does the same in Assembly language; similarly, the loop counter, K, points to each element of A$() in turn, just as the index register modifies BADDR in Assembly language

Given this, we can refer to every subsequent byte in the table by its position relative to the base address, so that the first byte is in position zero, the second byte is in position one, the third in position two, and so on. A byte's position relative to the table base address is called its index, and the absolute address of any byte in the table is calculated from the sum of the base address and the byte index. If we can construct a program loop in Assembly language, and use the loop counter as an index to the base address of the table, then we can address each byte of the table in sequence, just as we might access the elements of a BASIC array using a FOR..NEXT loop.

Once again, the Z80 and 6502 Assembly languages handle indexed addressing differently. The 6502 chip contains two single-byte registers called X and Y, each of which can hold an index that modifies a base address. This limits the length of a table to 256 bytes (the largest possible single-byte number). The Z80 chip contains two two-byte registers, IX and IY, which may hold the base address itself, and can then be incremented or decremented to point to successive bytes of the table. Since they are two-byte registers, IX and IY can address any byte addressable by the CPU itself. Their contents can also be modified by a single-byte index.

INDIRECT ADDRESSING

Indirect addressing involves the use of pointer addresses, a concept which was introduced early in the course, in relation to floating boundaries in memory (see page 58). Imagine that a group of people form a cinema club and that they meet every week to watch a film chosen by the club president. The film may be showing at any one of a dozen different cinemas, so when he has chosen the film for the week, the president writes details of the time and place on a postcard which he then sticks in the window of a shop in the centre of town. Club members don't know where the film will be from week to week, but they know where the shop is, and the shop 'points' them to the correct cinema. The address of the shop is, indirectly, the address of the cinema.

In indirect addressing mode it is possible to

TRAIN INDICATOR
No 3 PLATFORM FOR TRAIN TO
No 2 PLATFORM FOR TRAIN TO
No 1 PLATFORM FOR TRAIN TO
IAN McKINNELL

Indicator Pointer
Examples of indirect addressing do not often appear in everyday life. However, in this photograph the train indicator board contains the actual data wanted by the traveller, so the sign telling him where to find the board indirectly addresses that data. Indirect addressing in an Assembly language instruction means that the address supplied in the operand contains the address of the byte where the data is stored; the operand address is a pointer