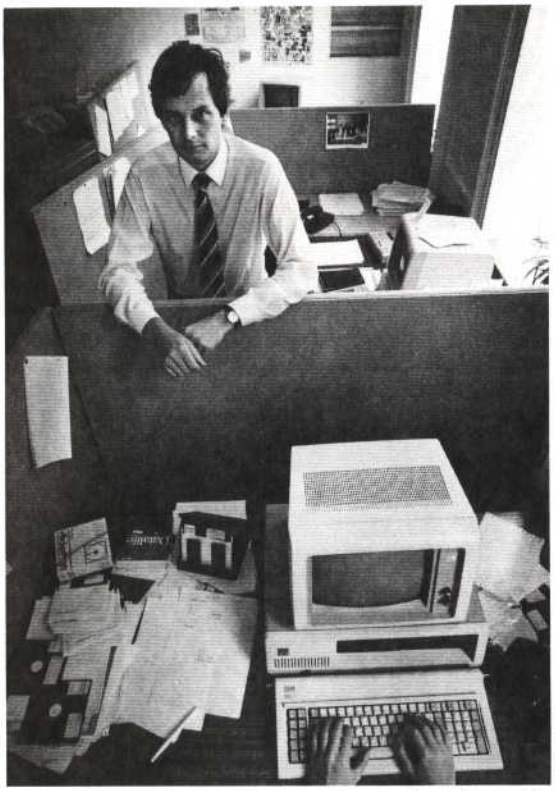
Intelligent Software
Specialising in strategy games such as chess, IS uses IBMs and Apples, with its own specially-developed interfaces, to develop software. Dividing programs into their machine-dependent and universal segments makes it easier for IS to support a range of computers and dedicated chess-playing machines

languages known as cross-compilers or cross-assemblers, which permit the work to be done on a micro that uses an 8086 processor, for example, while the programs that are produced will work on machines with Z80 processors. These cross-compilers are high-level languages (like BASIC), which makes them easy for the programmer to use, but the programs they create are written in machine code. Skilled machine code programmers scrutinise the programs that are developed, and often succeed in further optimising them.

Clearly, a development system has an enormous advantage over the home micro. A disk-based assembler, or one making use of expanded RAM space to store larger tables, will work more efficiently than an assembler that has to be wound in off tape and operates in the confines of a home micro. Debugging routines can be added into the development version of the code, with no worries about the code being too big for memory. It is also far better to work on a business computer that has a good keyboard, sharp display and disk drives.

A firm that makes use of this technique of program development is Intelligent Software (IS), founded in 1981 out of a pooling of experience between David Levy, the chess specialist, and Robert Madge's ANT Microware. The company specialises in strategy games, mostly written on contract for all the popular home micros. They also develop the software side of dedicated chess machines. Although there are no rapid combat displays in games like chess and bridge, a great deal of computation goes on behind the scenes. So, like arcade games, strategy games also need the speed of assembler-written software.

As well as using the target machines themselves for development, IS uses IBM PCs and Apples with specially developed interfaces to allow code to be exchanged across its range of machines. The company is often involved in conversion projects — transferring a chess game, say, from one computer to another — so its programmers have



TONY SLEEP

learned to write code in a form that is easily segmented. One level of segmentation that proves useful when the time comes to hand code from one processor to another is the division of the program into playing code and input/output code. The I/O code on the new machine will have different port or memory addresses, and will perhaps be strategically different too (polling replaced by interrupts, and so forth). Ingenuity may well be required to get round the hardware limitations, but there won't be a forbidding quantity of input/output. Playing code, on the other hand, will be there in abundance, but because it is isolated from the hardware (except, of course, the processor) its conversion will be straightforward.

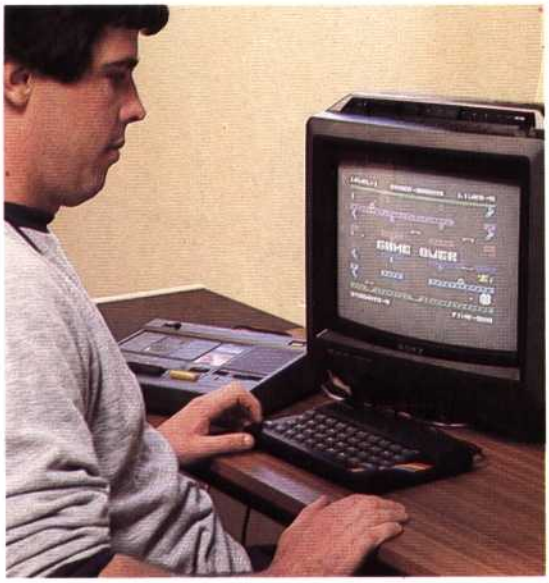
IS wants to avoid restricting programmers' inventiveness, so there are very few 'house rules' to govern the writing of code. One important point that they insist on, though, is that source code includes numerous comments, so it is always clear what the routines are doing.

Where programmers are working at home for a software house, each developing his or her own project, there is little pooling of resources. In this case, individuality is preserved at the cost of a great deal of duplicated effort, because the code for similar routines has to be reinvented by each separate programmer.

One software company, Psion, is making use of computers even larger than the IBM PC. Among British software houses writing for the home computer games market, Psion is unique in doing the bulk of its development on minicomputers. The company's hardware installation alone is worth a quarter of a million pounds.

Psion began as a company by developing software for the ZX81 — and used ZX81s to do it.

BOB BROMIDE



Visions
Programmers working from home on the target machines provide the bulk of Visions' programming effort. After the game concept and scenarios have been decided, the component routines are developed in native Assembly language (Z80 or 6502) using assemblers such as HiSoft DevPak on the Spectrum