WORKING NOTE D6

Alan Creak
1 September 1986

INTERPRETIVE PROGRAMME EXECUTION

This note was left as unfinished business in 1986, and forgotten. I have now (2002) reconstructed it from an old MacWrite file, incorporating this notice, some amendments I found on a printed copy from 1986, and a note in hindsight towards the end. The content is otherwise unchanged. It is likely to be of historical interest only, if at all, but it's nice to have the series complete.

THE PROBLEM.

One of the continuing difficulties associated with disabled people's use of computers is their reasonable desire to use the same software as able people. Many can use a standard keyboard slowly or not at all, though they are able to communicate with computers using a variety of "adaptive devices", such as switches, paddles, joysticks, optical pointers, and so on. Unfortunately, most ordinary microcomputer software is not easily adapted to non-standard peripheral devices.

WAYS AND MEANS.

That is not to say that the task is impossible, or has never been accomplished; indeed, several approaches have been used successfully, and some are outlined below. In general, though, there is no easy answer, as the only resources available for use in solving the problem are already in use by the running programme. It is therefore necessary either to find new ways of sharing the machine's resources between the activities which require attention, or to add some more.

It is sometimes possible to find an area of the computer's memory which remains unused by a package, and to store in that area some code which can translate the signals from the adaptive devices to standard keyboard input codes, and also to handle any special requirements for displaying the output (such as, for example, superimposing selection patterns on the normal screen picture). Even if such an area is available, though – and many packages try hard to use the whole of memory for their operation – it is not necessarily easy to link the interface routines with the package software. Source code for proprietary packages is frequently either hard or expensive to come by, and without such information it is difficult to know just what needs to be done.

An alternative approach through hardware is illustrated by the Adaptive Peripherals Inc. "Adaptive Firmware Card" (2). This card is designed to run with Apple computers. It is attached as a standard circuit card to the Apple system, and is in turn connected to such adaptive devices as are required. It runs as a *terminal emulator*, receiving input from switches or other peripherals and translating what it receives into keyboard signals; it operates (I think) by waiting for appropriate keyboard read and screen write signals to appear on the Apple system buses, then interrupts the processor and handles the data transmission as if it were a terminal.

The Adaptive Firmware Card can only be used with Apple systems. A more elaborate, more powerful, and – of course – more expensive approach is a separate terminal emulator. This device is designed to plug into a computer as if it were an ordinary terminal, but it is in fact a microprocessor-based unit which, like the Adaptive Firmware Card, translates signals from adaptive devices to the corresponding terminal codes and vice versa. Its great advantage is that it can be used in any system simply by plugging into the terminal connection.

ASSESSMENT.

Of these three solutions to the problem, the first is comparatively cheap (it requires no additional hardware), but successful only sometimes; even when it can be used, it is likely to need skilled work to bend an existing programme by adding the required routines, and it is always subject to the need for revision as new versions of the software become available.

The Adaptive Firmware Card is not cheap, though its cost need only be met once. It is claimed to work successfully with the great majority of Apple programmes without any amendment being necessary. It will, of course, only work with Apple machines.

The separate terminal emulator, being a self-contained unit, is even more expensive, though potentially more versatile. It will always work with any programme and with any computer which uses a standard terminal, and is likely to be very reliable.

None of the three methods in any way *expands* the capabilities of the machine it works on; they are designed simply to make it possible to use the machines with special peripheral devices.

THE PROPOSAL.

There is another possible approach. Like the three outlined, it has disadvantages, but it has three characteristics which are noteworthy :

o It is a software solution : no additional hardware is required.

o It will cope with any programme without alteration, provided only that the programme is not critically dependent on processor timing.

o It can, in principle, execute programmes written for ANY type of microcomputer whatever type of microcomputer is actually used.

The essence of the proposed method is very simple, and certainly not new. It is to execute the programme using a *simulated* processor rather than the microcomputer's actual *hardware* processor. The programme to be run is given as "data" to a simulator, which is just another programme which mimics the actions of some appropriate real processor. The simulator inspects each machine instruction in turn, analyses it, and performs the same actions as the hardware processor would have done, but using ordinary memory areas to represent the machine's registers. The programme is therefore executed just as if it had been executed directly by hardware – though considerably more slowly (the single obvious disadvantage of the method). As against this, the simulator need not be the same size as the actual machine, so other software, to handle the translation between conventional terminal signals and those required by the special peripherals, can be accommodated in the computer's memory. The simulator will, of course, recognise the instructions in the programme which call for communication with the terminal, and handle them appropriately. Clearly, as the simulator is a programme, it need not simulate the "host" machine – so in principle a programme written for one sort of microprocessor can by this means be executed on quite a different machine.

The disadvantage is the reduction in speed, which may be considerable – a factor of 10 would perhaps be a realistic mean figure. There are two reasons, though, why that might not be very important. First, the factor applies only to the processing; and many programmes – such as editors, word processors, some simple spreadsheet programmes, and so on – spend comparatively little of their time processing, and much more either executing input or output operations, or waiting for the next instruction from the terminal, so the effect of the expansion factor on the overall time is comparatively small. Second, it is in any case likely to take longer to enter data through adaptive devices, so, while this is not presented as any sort of excuse, the slowing down is likely to make a smaller proportional contribution to the total time than would be the case for a fully able operator.

Another factor worth bearing in mind is that the microprocessors themselves are getting faster. It is therefore quite possible that a new microprocessor executing a programme interpretively could run at a speed quite comparable with that of an older machine executing the programme directly.

REASSURING HISTORICAL APPENDIX.

I know it is all possible, because many years ago I constructed a computer simulator with many parallels to this proposal (1). As in this case, the requirement was to run machine language programmes as far as possible unhindered in a computer, but at the same time to make space for other software which could monitor the performance of the running programme in various ways (none of which had anything to do with speed !), and protect the system from possible unfortunate consequences of programming errors. As the aim was for a system to teach assembly language programming, such errors were expected to be frequent; but the simulator was able to execute the programmes quite aceptably quickly, and to perform a number of ancillary tasks as well.

WHAT NOW ?

The next step is to try it. I propose to begin by simulating an Apple on a BBC microcomputer. The choice of systems is dictated by circumstances : I have a BBC microcomputer, and have access to software for disabled people which now runs on Apple systems. It happens that the two machines use (essentially) the same microprocessor; this is quite coincidental, and has little bearing on the way in which the simulator is written. In any case, I don't think that the particular systems chosen are important, as the same factors will appear whatever machines are used.

I never did that. I think I made a start, but was overtaken by circumstances.

REFERENCES

(1) G.A. Creak : "*Assembler with source programme in core – programming manual*" (Auckland University Computer Centre, 1977).

(2) "*Adaptive firmware card version 2e operators' manual*" (Adaptive Peripherals Inc., 1984).