

Short Notes

On Instruction Set Encoding

This note proposes that computer instruction encoding and decoding should take advantage of the user and kernel states of the machine so as to provide the operating system with privileged instructions inaccessible to the user.

Today most computers, including many microcomputers, execute instructions in one of two states, which we will call Kernel and User states. The purpose of this division is to provide protection for the operating system from user error or malice, and to protect processes from each other also in a multi-user or multi-tasking environment. In the Kernel state all instructions in the machine's instruction set are available, whereas in the User state only a subset can be executed; those instructions that affect peripheral equipment or the internal registers whereby the operating system effects protection and manages the machine's resources are considered illegal instructions.

Sometimes, as in the DEC PDP-11 machines, a segment of working storage address space is designated as the 'I/O page'. Access thereto is denied in the User state and there reside the peripheral control, status and data registers, as well as the internal registers used in management and protection. However, even on such machine architectures a number of Kernel privileged instructions are needed, as resort to the reference manuals will show.

The net result is that during instruction decoding in the User state attempts to execute certain instructions, the Kernel state 'privileged instructions', have to be detected and trapped.

To avoid the need to detect these special cases we make the following proposal.

Every contemporary machine architecture to which the above discussion applies also has a program status word (PSW), or an analogue thereof. A bit in this PSW determines in which state the machine is. We propose that this bit be gated as the leading (trailing?) bit of the operation code during instruction decode, thus providing for different instruction sets in the two states.

Rarely is an instruction set dense, using all the possible encodings. So those instructions that are 'no-ops' can be used as privileged instructions in the Kernel state. Alternatively, there will be User state instructions without utility in the Kernel state, e.g. floating-point operations or decimal arithmetic.

An objection will be that compilers and interpreters can afford protection against illegal instructions (except perhaps while they are themselves being debugged) and that nobody programs in assembly code anymore. Wrong. The circle of home, personal computer users widens daily, and many wish to play with machine code. It may be that if they crash their system we should regard it as a salutary lesson and not a tragedy, but there is much talk of 'user friendly' systems. Perhaps it could be facilitated at the microchip level.

H. D. BAECKER
Department of Computer Science
University of Calgary
Calgary, AB, Canada

Received June 1983

Participative Systems Design

A recent paper by Wood-Harper and Fitzgerald¹ and a subsequent note by Cookson² have suggested that participative systems design is concerned primarily or solely with implementation. This note provides a description of participative design as practised by the writer.

In recent issues of *The Computer Journal* there have been discussions of different approaches to systems analysis, and attempts have been made to classify and describe the characteristics of each approach. This is a praiseworthy effort and it is to be hoped that such discussions and explanations will continue. However, in the paper by Wood-Harper and Fitzgerald¹ and the subsequent note by Cookson² statements have been made about participative design which are not correct, in particular the point that participative design is concerned primarily or only with implementation. This does not accord with my experience of using this approach in different companies during the past thirteen years. I would describe participative design, which should be called participative sociotechnical design, as having the following features. First, the total design task is defined as embracing a number of subsystems. These are technology (hardware and software), administrative tasks (handling information, solving problems, co-ordinating activities etc.) and organizational structure (networks of groups, roles and relationships). The design boundary is therefore placed around an integrated unit or set of units such

as a department or functional area. Secondly, users are actively and jointly involved with the technical specialists in parts of this design task; especially administrative and organizational design and the design of the man-machine interface. There are two reasons for involving users in this way. One, users possess a great deal of knowledge relevant to good organizational design; they know how their department does work and can soon identify how it should work. They are, better than an outside 'expert', able to systematically and accurately analyse their own information needs and, if an expert system is being designed, to specify their key knowledge attributes. Two, practical experience in designing contributes to learning how to design, and technical knowledge can be gradually transferred to users. This is of considerable benefit when technical expertise is a scarce and overstretched resource.

The design task starts, not with the assumption that a computer-based solution will be introduced, but with the question 'do we need to change? Is our department optimally organized now to achieve its mission or could organizational and/or technical change make us more effective?' If the answer to these questions is 'yes' then the mixed user/technical specialist design group moves into the design task. This encompasses an analysis of problems that are impeding efficiency and effectiveness and an analysis of the job satisfaction needs of all staff who work in the department. Precise objectives are then set for the new system covering efficiency, effectiveness and

job satisfaction. A number of organizational administrative and technical options are next identified and examined in detail and the organizational-administrative-technical combination that matches best with the system objectives is the one that is selected. This is then designed out in detail, tested through pilots and implemented. Later the working system is carefully evaluated by the users to check that it is continuing to meet the specified system objectives.

The participative systems design process therefore covers the traditional systems design model of investigation, analysis, design and implementation shown by Cookson. Where it differs is in breadth not in length.³ Its emphasis is on good organizational and technical design, with participation a means for achieving this.

ENID MUMFORD
Manchester Business School,
Booth Street West,
Manchester, M15 6PB, UK

References

1. A. T. Wood-Harper and G. Fitzgerald, A taxonomy of current approaches to systems analysis. *The Computer Journal* 25(1), 12-16 (1982).
2. M. J. Cookson, Taxonomic Studies on current approaches to systems analysis. *The Computer Journal* 25(3), 283-284 (1983).
3. E. Mumford, *Designing Human Systems*, Manchester Business School (1983).

Received September 1983