Correspondence

To the Editor,

The Computer Journal.

Sir,

Paging and segmentation

After attending the meeting at Imperial College on Virtual Address Spaces, I get the impression that a clear statement in your pages as to the difference between paging and segmentation would not come amiss. The distinction is very simple, but owing to the pragmatics of the situation, every attempt to go into detail is apt to obscure the simplicity. In brief, then:

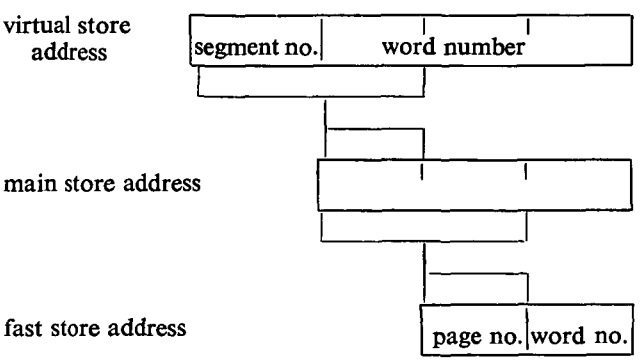
> *Paging* is an engineering device of which the programmer need be in no way aware;
>
> *Segmentation* is a programming situation in which addresses $p1, p2, p3, \ldots$ can be indexed using $i1, i2, i3, \ldots$ and if $pj \neq pk$ then no choice of values for ij, ik can ever make $pj + ij$ equal to $pk + ik$.

The definition of segmentation is equivalent to stating that every address is an ordered pair of integers with no theoretical upper limit on either member of the pair. Of course even unsegmented addresses, which are single integers with no theoretical upper limit, must have an upper bound in any actual machine, and we are all familiar with the difference this creates between the abstract program and the real one whenever the requirements of the program approach this bound. When addresses are segmented, flexible upper bounds to each segment and to the number of segments used must ensure that the total store called for does not exceed that of the machine.

Now for some of the puckishly obscurantist pragmatics. (1) A programmer who knows the relative times taken by addition, multiplication and division can write more efficient programs than one who does not. In the same way, it may be to a programmer's *advantage* to be aware of the paging in a given system. This must not obscure the fact that there is never any *necessity* for him to be aware of it. (2) Equally, of course, a programmer can ignore any possibility of segmentation which he may be offered, but this is trivial—he can equally ignore the existence of a multiplication order and write his own software to replace it if he wishes. (3) A paging system is probably the most efficient way of implementing segmentation. This is possibly the main cause of the confusion which still exists in some places and which it is the object of this letter to dispel. Let us consider a specific example.

Imagine a machine whose instruction format contains a 20-bit address field but whose actual store is only 2^{15} words of a rather slow core store. Initially, the programmers put up with all this—use only the first 2^{15} addresses and accept the slowness of access. Then the engineers get to work. First they install 2^{10} words of fast core store. This is not used to increase storage capacity but to increase speed; each block of 2^5 words in it is a copy of some corresponding block in the main store which is currently being heavily accessed. The programmer knows nothing about this (except to rejoice in the increased speed), choice of which "pages" to keep in fast store at any moment being a matter decided by the system, which has to keep a record of which pages are currently in the fast store and where, and to divert accesses from main store to fast store when possible. The machine now has paging; unknown to the programmer a limited part of the main store is mapped into the fast store. But now the engineers take a second step. They arrange for a precisely similar mechanism to map a part of the virtual (i.e. addressable) store of 2^{20} words on to the real main store of 2^{15} words. This means that the programmer effectively has 2^5 virtual store areas each of 2^{15} words maximum which he can use in any way he likes subject to a total aggregate in all areas of 2^{15} words. If he uses each area reasonably (and this would include keeping several stacks, each one in a separate area), he knows that no area can overflow into the next, since "physical store full" is bound to be signalled before any single virtual area becomes full. The programmer now has a simple form of segmentation, and, far from being unconscious of it, he will probably be quick to exploit it.

The moral of this story is that the segmentation could be provided even though the fast store were not. The mapping which it demands must, to be economic, be done in units longer than a word, and thus, in a sense, in pages. But with the figures from the previous paragraph it is quite likely that these "pages" would be 2^{10} words long (as opposed to the 2^5 of the fast store paging) in order that both mappings could be from a 10-bit into a 5-bit field, and the same "design" (hardware and software combined) could be used for both. The figure illustrates this:



Other arrangements are possible; where the maximum size of a segment is not equal to the main store size, the independence of segments can be secured by a trap on carry from word-number field to segment-number field when indexing.

Yours faithfully,

BRYAN HIGMAN

University of London
Institute of Computer Science,
44, Gordon Square,
London W.C.1.
19 April 1967.

To the Editor,

The Computer Journal.

Sir,

I am rather a tyro as far as analogue computers are concerned, and what I have to say may well have been said before, though I have not found any record of it as yet.

Coming from a background of digital computers, it appears to me that analogue computers are at present in the same state as regards programming as were their digital counterparts twenty years ago, when the programmer had to translate his thoughts into a binary code punched on teleprinter tape. The user of an analogue computer must translate his logical model into a patch-board, which he has to wire up himself.

It would surely be greatly to the advantage of analogue computer users (and manufacturers) if they could specify their problems in a high-level language, which could then be translated automatically into a patch-board diagram, and the patch-board could then be automatically wired up. They would then be in at least as happy a position as Fortran users, perhaps happier if they could agree on a standard language and learn from the mistakes made with digital computers. This state of affairs could be brought about by stages, some of these performed in parallel, and each stage being useful by itself or in conjunction with past stages.

1. A patch-board wirer, accepting a punched paper tape in whatever code is most convenient, and automatically interconnecting the required terminals. The logic of this piece of hardware would be similar to that of a telephone exchange, though it is to be hoped that the service would be a bit more reliable, and the difficulty would be to keep the price down. I believe there exist already devices which set the potentiometers to the required values, thus completing the hardware necessary to set up the computer automatically.

2. A high-level language, together with a digital computer program to compile this into an intermediate language. The intermediate language being an abbreviated form of patching instructions, such as "B4O–P3U", meaning "connect the output of amplifier B4 to the upper end of potentiometer P3".

3. A digital computer program to compile the intermediate language into the code required for the patch-board wirer.

4. When once the previous stages have been completed, a system of automatic re-scaling is performed by a multi-access digital computer, to which the analogue computer is linked. Up to this point the compiler of stage 2 would require scaling factors as input, but at this stage it would become possible to manipulate these automatically until the analogue computer is working at maximum accuracy.

No doubt, being a novice in this area, I have left out something vital, but I should be interested in the comments of those more versed than I in the use of analogue computers.

Yours faithfully,

I. D. CRADDOCK

8, Wardie Road,
Edinburgh 5.
22 May 1967

Chromatic numbers and timetabling problems

To the Editor,

The Computer Journal.

Sir,

In a recent paper Welsh and Powell give a method for finding a bound to the chromatic number of a graph and indicate an application to the examination timetabling problem, referring to a paper of mine. Unfortunately this reference is misleading to the casual reader and possibly to the authors. In my paper I obtained a solution to an example in 14 periods and pointed out that by inspection of the incompatibility table it could be deduced that this was the best possible solution. Welsh and Powell apply their method to the data in Table 2 (the incompatibility table) in my paper, and by coincidence find that their bound for the problem is 14 periods. This bound, however, is a bound to a different problem, namely a case in which there is only one paper per subject.

The fact that some subjects have more than one paper increases the number of nodes in the graphical representation and also the number of edges joining incompatible nodes. In my example using the number of papers per subject which can be extracted from Table 3, the resulting chromatic number bound increases to 23 which although a significant improvement on the cruder bound max $(d_i) \equiv 32$ is still poor by comparison with the true minimum of 14.

This is not an isolated case. In two recent timetables produced at St. Andrews the corresponding figures are

	MINIMUM	ACTUAL SOLUTION	CHROMATIC BOUND	MAX (d_i)
Case 1	7	8	14	28
Case 2	not known	18	31	41

It should also be noted that these bounds take no cognizance of additional conditions such as subject A must precede subject B, subject C must be in a certain period, papers of subject D must not be consecutive and so on.

Yours faithfully,

Computing Laboratory,
University of St. Andrews
19 June 1967.

A. J. COLE
(Dr. A. J. COLE)

References

WELSH, D. J. A., and POWELL, M. B. "An upper bound for the chromatic number of a graph and its application to timetabling problems", *The Computer Journal*, Vol. 10, pp. 85–86.

COLE, A. J. (1964). "The preparation of examination timetables using a small store computer", *The Computer Journal*, Vol. 7, pp. 117–121.