N.B. (i) Assumed for simplicity that hours worked not zero.
(ii) Intermediates not calculated assumed to be zero.

Key to symbols used:

$=\!\!\rightarrow$	derived from
$+\!\!\rightarrow$	together with
$\longrightarrow$	followed by
n	not applicable
/	the remainder of the range within straight brackets
x / *ry*	round *y* and above up to nearest *x*.

Notes: Systematics describes the information model at three levels of detail.

Level 1. *The model itself*

A model is restricted to a major and separate operation performed upon one class only. At this level a broad statement of the purpose of the model is given identifying the information class concerned. This is followed by the dictionary. The dictionary lists and classifies every item of information used in the model and defines its variability. Permanence is indicated by P = permanent, U = up-dated, O = originated. Subscripted "a" indicates a sub-class. On its own it indicates the principal of that sub-class, i.e. the item of information that uniquely identifies a particular member. In brackets it indicates a subordinate of that sub-class. O, I and R show whether the item is output, input or held as a record, or what combination of these apply.

Level 2. *Routines*

A separate routine is provided for each up-dated item and for each originated item. It states which other items are required to derive the item concerned and under what conditions the routine is performed.

It also states the "intermediate" steps taken on the way to the final derivation, and how these are related to each other through the connectives. References are provided for each expression (capital letters) and each connective (small letters).

Level 3. *Expressions and connectives*

The derivation for each routine is stated in final detail at this level.

Correspondence

To the Editor,
The Computer Journal.

Sir,

The asides, as it were, in J. P. Penny's analysis of a time-shared computer system (this *Journal*, Vol. 9, No. 1, p. 53) raise issues that are by no means settled regarding the preferred or even useful storage organization of such systems.

Towards the end of Section 5 of the paper we have the assertion that the storage requirements of a program can be found exactly when it is compiled. This is certainly not true of programs written in ALGOL, CPL, PL/1, or similar languages, the similarity being that they allow dynamic storage allocation. That this is not an exotic frill only of interest to the far out fringe I can testify after nine years' connexion with commercial and industrial applications of computers. There, there are very many practical jobs that could be greatly simplified, completed more speedily, and made cheaper to run, given full dynamic storage allocation.

In Section 2 we have the argument that the simplicity and inherent efficiency of single level storage is attractive for time-shared systems. But this assumes that it is convenient, let alone economic, to have all of a program, and all of its data space, available in primary storage at the same time. Firstly, most programs above the level of student exercises may have 50 or more of their instructions written only to deal with exceptional circumstances or errors. On any particular run of the program one hopes that most of this necessary lumber will not be called upon. It is therefore not economic to use single level storage if much of it is not called into use. A similar situation arises with, for instance, periodic analyses appended to a regularly run program. In an ideal world we would have two debugged versions of the program, with and without analysis segments, to run as required. But in practice there are too few programmers to do the tidying up, and multiplying programs harms the administration of the computing facility.

Secondly, most programs have several distinct phases, initialization, close up, main processing cycle, and so on. Although they may share common subroutines only one phase need be available for execution at a time. Thus the logical structure in the way we happen to write programs would seem to be amenable to the reasonably efficient use of multi-level storage systems, as long as the source language programmer is only faced with a virtual single level store of suitably vast size.

Reverting now to time-sharing, it would seem that we can only make efficient use of the large high speed primary storage required for this if we also incorporate in our system very much larger secondary storage to buffer the fluctuating storage requirements of the programs concurrently in the system.

Yours faithfully,

H. D. Baecker

Imperial College,
London, S.W.7.
12 May 1966