25 July 1967

A sequel to EWD201.

In today's letter I shall give some hardware considerations, because all the time they are at the back of my mind and it may be confusing if I keep silent about them much longer.

The first observation is that inside a sequential machine an important logical function has to be attached to the notion of simultaneity. This notion is never an exact one (in the mathematical sense) it is always with respect to a certain grain of time. The faster the sequential machihe, the smaller its time grain. It is also obvious (at least to my taste, formed by theoretical physics years ago!) that the physical dimensions of such an sequential component must be small compared with the distance travelled by light during such a grain of time, i.e. the fast components must be small. One answer to this has been given by (micro)miniaturization, but I am looking for another answer, viz. the coupling of a number of such fast components.

I presume that each fast component will have its own, private clock, its "local time", and I am looking to an arrangement, where the various components can really work independent of each other, temporarily unsychronized with respect to each other.

Immediate consequences of this approach are the following two.

Firstly, the individual fast components must have their private memory, coupled with one or more processors: without a private memory they will not be able to work on their own for a considerable number of microseconds.

Secondly, we shall have to arrange the co-operation between these components, and this will minimally imply an information traffic between these components. The actions taking place in this co-operation must correspond to a much coarser grain of time.

In order to visualize thing a little bit more clearly, I have been thinking about a number of fast components grouped around a centre. This centre should have a great amount of store and it should have logic. I assume that bulk information traffic from one component to another will always take place via the centre, I have verified that a single centre can be coupled to a number of mutually asynchronized components. It is particularly simple if we conceive the components as being in one of two mutually exclusive states, either working, independently and geared to its private clock, or communicating, to a large extent geared to the clock of the centre, that probably must be able to maintain a number of communications simultaneously, say word wise merged. Information traffic from centre to component is the easiest of the two: the centre will only send it when the component is standby available to receive it and the centre should send it not too fast (or in the case of conflict at centre side) slower. Information traffic in the other direction requires a signal from the centre to the component, that the centre is ready to accept along that channel the next information unit. If the channel has such a long delay, that its capacity slows down, then I presume a private buffer at the centre end of the channel. All this can be done if necessary, the main thing to take care of is to avoid urgency situations as may arise due to causes of combinatorics, i.e. where a number of channels share the same memory and its access mechanism. (In the above picture the receiving component is faced with an urgency situation, but that is not too bad, for the component has nothing else to do and is "standing by".)

The potential advantage of such a configuration would be, that configuration extension with some more components should not be the cause of a major software revision. A consequence is that the total work load is presented to the machine in such a way, that the possibility of parallellism is given and need not be detected.

Previous experiences seem to indicate that in the mutual synchronization of sequential processes we have two distinct, and in a sense complementory, aspects. On the one hand we have the critical sections, of which we only know, that they exclude each other in time, but where we do not define any priority rules; total time spent in critical sections must be such a small fraction of real time that the queueing strategy can be regarded as irrelevant. With an implementation for critical sections and the notion of the private semaphore for each sequential process, we can implement whatever strategy we want.

This knowledge may give a hint how the system should be divided over hardware and software and what should take place in the centre and what in the components. At least a hint as to what is attractive from a logical point of view.

My present hope is the following.

All processing, also the execution of "systems program" in critical sections will be done by the components. The centre is fairly neutral (although its logic will be programmed it will be a special purpose device) but caters for mutual exclusion automatically. I hope that the mutual exclusions needed can be coupled to the accessibility claim towards a certain amount of information.

It is obviously associated with the state of embedding of a sequential process: if a process operates on its private variables only, as long as it moves in its own state space, than this process together with the variables being the coordinates in this state space, can be delegated for the time being to one of the components (having processor and store). As soon as communication with the outer world implies operating on "common variables", then also these common variables must be sent to the component, and therefore cannot be somewhere else. But these are exactly the critical sections: one of the processes claims and gets the privilege to inspect and modify in a universe it shares with others.

In our present multiprogramming system we have at bottom level the task of processor allocation; if we follow an analogous approach we should get here the task of "component allocation"; logically speaking one should expect the implementation of the component allocation in the centre. The little experience we have has not yet belied our assumption that the problem of processor allocation is relatively trivial and it is conceivable that a non-sophisticated standard solution, implemented in the centre, will indeed do the job of component allocation.

Warning. Here I must be very careful not to make too hasty an extrapolation. In our present environment processor reallocation does not imply extensive moving around of bulk information. In the case of component allocation this will no longer hold. So maybe the centre will have more refined logic (of a special component, put aside permanently, to do this scheduling job).