ISTITUTO NAZIONALE DI FISICA NUCLEARE
Laboratori Nazionali di Frascati

LNF-77/20(R)
12 Maggio 1977

R. Biancastelli, V. Bidoli, A. Cavestro, G. Cioffi, M. Cordelli, G. De Luca, A. Giordana, P. Gricia, S. Salza, M. Spinetti and G. Zanella: SEM. A MULTIPROCESSOR SYSTEM FOR REAL TIME DATA PROCESSING - FIRST PROGRESS REPORT.

Servizio Documentazione
dei Laboratori Nazionali di Frascati
Cas. Postale 13 - Frascati (Roma)

Laboratori Nazionali di Frascati
Servizio Documentazione

LNF-77/20(R)
12 Maggio 1977

R. Biancastelli[1], V. Bidoli[2], A. Cavestro[3], G. Cioffi[4], M. Cordelli[1], G. De Luca[4], A. Giordana[5], P. Gricia[1], S. Salza[4], M. Spinetti and G. Zanella[3] : SEM. A MULTIPROCESSOR SYSTEM FOR REAL TIME DATA PROCESSING - FIRST PROGRESS REPORT.

ABSTRACT. -

A project is now in progress, at the Laboratori Nazionali di Frascati, for the investigation of the possibilities of the parallel processing, in the field of high energy physics experiments as well as in other fields. The interest in the study of parallel systems is also motivated by the recent availability of cheap and fast microprocessors.

The SEM system is implemented using a set of NOVA 3 processors which are used as a pool of omogeneous resources, communicating each other through a common memory.

1. - INTRODUCTION. -

The progress recently achieved in the high energy physics experimental apparata has two important consequences which are saturating the throughput of the single minicomputer on line with the experiment.

---

(1) - Istituto Superiore di Sanità, and INFN, Sezione Sanità, Roma.
(2) - INFN, Sezione di Roma.
(3) - Istituto di Fisica dell'Università di Padova, and INFN, Sezione di Padova.
(4) - Istituto di Automatica della Facoltà d'Ingegneria di Roma, and C.S.S.C.C.A. del CNR, Roma.
(5) - Istituto di Fisica Sperimentale del Politecnico di Torino, and INFN, Sezione di Pisa.

The growth of the instrumentation has made the control work heavier on the apparatus and often even limits the acquisition rate of the physical events. In fact the new particle detectors (the proportional chambers and the drift chambers) have raised the data flow in front of the on-line processing system.

It would be possible to have a data reduction by using a hardware processor[1], but this method is neither flexible nor simple to implement.

Also the use of a midi-maxi computer is not practicable because of the high cost and its problems in the operation.

In this framework the use of parallel processing and in particular of a multiprocessor system appears as a promising perspective[2].

The SEM multiprocessor has been conceived for the fast execution of the following tasks:

a) data acquisition from the experiment, through CAMAC instrumentation,
b) analysis of the data taken in the point a),
c) on-line instrumentation test and experiment control.

The SEM structure is not hierarchical therefore there is not a group of processors, usually named "front-end" (FE), which are exclusively dedicated to the data acquisition and a group of processors, named "back-end" (BE) which are dedicated to the processing. It is then possible, with this system, a better exploitation of all resources.

In fact, since the physical events are not uniformly distributed in time, the FE-BE solution is not efficient, the number of FE computers being sized on the peak load.

Instead in the SEM system there is the possibility to use all resources during the acquisition and then, between one event and the next, the same resources are all assigned to the processing.

It is also possible, in the data processing time, between an acquisition and the next, to analyse an event as a whole of parallel tasks (parallel tasking) and to overlap the processing for more events (multiprogramming).

The instrument logging is carried out by the system during the analysis time and it may be regarded as another multiprogramming level.

2. - SEM SYSTEM DESCRIPTION. -

The SEM system is the final evolution after a study series[4] on the high speed on-line data processing.

The basic concept is to have a multiprocessor system where a set of omogeneous processors are concurrent in the parallel execution

of the same program, under control of the SEM operating system which provides the scheduling of the various tasks.

The block diagram of the system is shown in Fig. 1, where M_i are the common memory blocks and MX are the bus multiplexers.

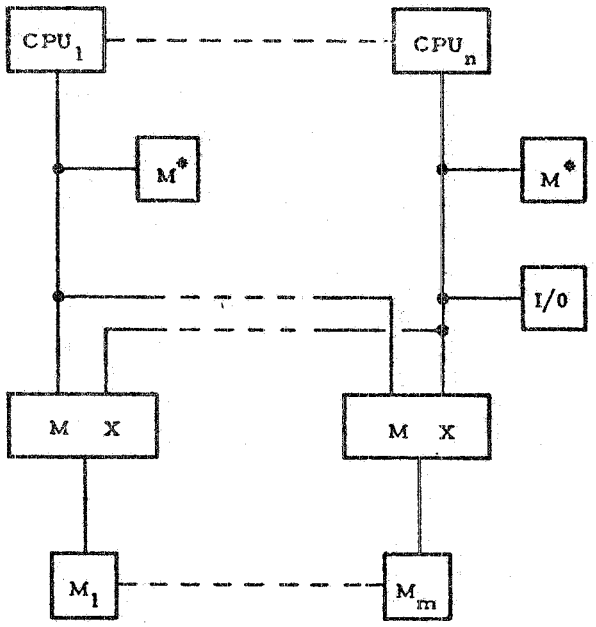


FIG. 1 - General diagram of the SEM system: CPU = processor; M^* = local memory; M_i = common memory; MX = = bus multiplexer.

The MX bus-multiplexers solve the access conflict of various processors to the same common memory block, with a dynamic reconfigurable priority, by regulating the data the address and the control signal flows.

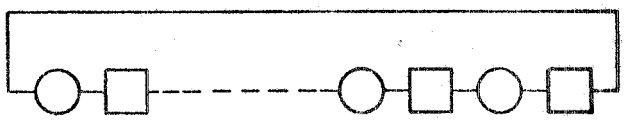
Through the common memory it is possible to have the interchange of data and of programs between the processors.

The SEM architecture is general in the sense that many different architectures (e.g. tree, star, ring, etc.) may be obtained (Fig. 2) by reconfiguring the system in the basic software and in its interconnections to the common memories.

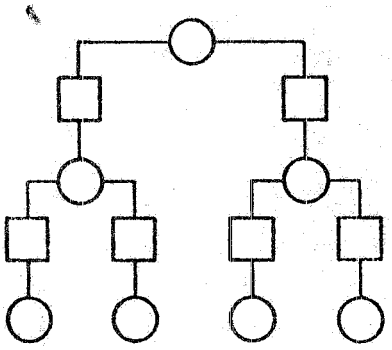
The SEM system may be a study object for many applications; in particular it can also be used for artificial intelligence researches, real time pattern recognition, and in all the fields where a high data flow or a real time bound requires high speed data processing[5].

The input-output is not common to more processors, at least in this first design, and only the processor which has the peripheral unit connected to its bus executes the corresponding I/O tasks. The interrupt system carries out the buffered I/O, in the standard manner.

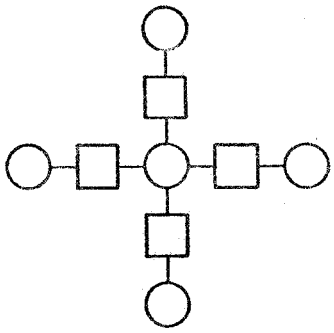
RING :



TREE :



STAR :



FULL CONNECTION :

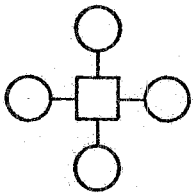


FIG. 2 - Examples of various architectures of the SEM system, with different connections between processors and common memories : ○ = processor ; □ = common memory.

However all processors may execute I/O operations in each peripheral through a common memory buffer (see par. 3.3).

3. - THE OPERATING SYSTEM. -

3.1. - Data acquisition.

As introduced in the former section, CAMAC has direct access to the I/O busses of the processors ; the CAMAC system is tipically divided in separate blocks interfaced to different processors to allow the highest degree of parallelism for the acquisition thus minimizing the time needed to perform this task.

For sake of concreteness, we will refer to the application considered : suppose we have to read the CAMAC instrumentation whenever some event occurs in our experiment, and then to analyze the data acquired.

The procedure by means of which the acquisition is carried out can be stated as follows : whenever an event occurs, an interrupt signal is sent to each processor which halts its current processing activity

and starts to execute the acquisition routine, the code of which is in its private memory.

The acquisition routine is structured in the following steps:

a) allocation of a memory area needed for the data sent by CAMAC,
b) input data storing,
c) communication to the operating system of the addresses of the memory locations where the data have been stored.

When all the processors have completed the acquisition, the supervisor activates the analysis of the event occured by putting its analysis program in the set of the programs waiting for execution. Each processor, except the last one, as soon as terminates its acquisition routine, restarts on the program that was running before the interrupt signal.

The last processor - i.e. the last in accomplishing its acquisition routine - enters a supervisive state to activate the analysis program.

A first problem which arises is that of preventing memory access conflicts during the acquisition time, as the data read from CAMAC are stored in the common memory blocks. This is done allowing each processor to operate during this task a different common memory block in which the acquisition routine of the processor directly allocates the storage area needed. This implies of course that the number of memory blocks should not be less than that of the processor employed in the CAMAC readout.

In each block the system takes care that a portion of storage is reserved for future input data. The wideness of this portion could be dimensioned according to the estimated statistical distribution of events in time, to the memory and time needed for the analysis of an event, as well as to the "cost" of the loss of an event (no acquisition possible for lack of memory).

Otherwise when an "a priori" estimate of the former factors is difficult or unfeasible it could be possible for the system to vary dinamically the amount of that storage area by a real time analysis of the behaviour of the system.

The problem of combining the whole information concerning a single event and acquired by different processors operating independently and in parallel, has been worked out creating, for each event, a table in which every processor, once it terminates the acquisition, writes control informations and addresses of data.

In particular the last processor accessing the table is responsible of the correctness (and the uncorrectness, as well) of the acquisition and collects all the information needed to start the corresponding analysis.

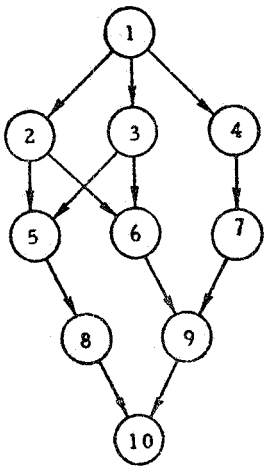
3.2. - Parallel processing.

3.2.1. - Program structure.

To fully exploit the multiprocessor architecture of SEM, the user's programs have to be structured in a special way: the main program must be organized in blocks (tasks), some of which can be executed in parallel (task parallelism).

This leads to a well known structure for the program: it can be modelled as a directed graph in which the nodes are the tasks of the program and each directed path is a precedence relation between its first and last nodes (tasks). See Fig. 3.

GRAPH



CONTROL TABLE

Task	Number of predecessors	List of Successors	Memory Pages	Instruction Field
1	ø	2, 3, 4	A, B, C, D	1øøø
2	1	5, 6	A, B, E	4576
3	1	5, 6	B, E, F	1541
4	1	7	C, H	343ø
5	2	8	E, F, G	64øø
6	2	9	E, F	377ø
7	1	9	C, H	214ø
8	1	1ø	F, G	723ø
9	2	1ø	E, F, H	51ø3
10	2	-	E, F, G	6ø14

MEMORY MAP

Logic page	A	B	C	D	E	F	G	H
Physical Page Address								

FIG. 3 - Example of a graph structured in ten tasks with corresponding control table and memory map.

In the following such structured program will be called "graph".

The condition expressed by the arrows entering the nodes of the graph is that the corresponding tasks can be activated if and only if all of the predecessors are terminated.

For instance, in the graph of Fig. 3, task T 9 can be activated iff T 6 and T 7 are terminated.

Each graph has a single initial task and a single final one, which do the job of defining univocally the inizialization and termination of the execution.

It is easily understood that the first and or the last task can sometimes be dummy tasks.

With every graph a set of logical pages P_G is associated, that represents the area of the related data; at the execution time the operating system matches every logical page with a physical one.

With every task T_i there is associated the set of logical pages $P_{T_i} \subseteq P_{G_I}$ on which T_i operates.

Finally two more sets of logical pages are to be defined: P_I and P_O (P_I, $P_O \subseteq P_G$); these are the input and output pages of the graph respectively.

The whole information about the graph structure is stored in two simple tables, the "control table" and the "memory map" of the graph; the SEM Operating System, as it will be clear from the following uses these tables (and other information structures derived from these) to manage the execution of the graph, i. e. to decide when new tasks can start their execution and to allocate the memory pages needed. Fig. 3 shows a sample control table.

3. 2. 2. - Uniprogramming in SEM Operating System.

The usual situation in the case of high processing load is the simultaneous execution of more than one graph in the system. In fact, several graphs may be active, in the sense shown, at the same time.

Nevertheless, for an easier understanding of the basic philosophy of the system, we will restrict ourselves in this section to the description of the management of one single graph.

In this case, the system handles two queues: the ready task queue (RQ), that is the queue of tasks whose execution can be started at any moment, and the completed tasks queue (CQ), that is the queue of tasks which have terminated their execution.

In the general case, more than one task is in execution in the system - each task being executed on a different processor. As soon as one processor terminates a task, it enters a supersive state, puts the completed task in CQ and collects a new one from RQ. If RQ is empty, the processor collects informations from CQ, enters the control table and operates in order to update the situation of the graph execution.

Now, if RQ remains empty - or if CQ was empty when first entered - the processor enters a waiting state: in fact, no task can be executed at this time.

We schematize the update of RQ as follows:

a) Once that RQ is empty, the processor collects one task from CQ and finds its successors in the control table.
b) In the rows corresponding to each successor, the number in the

second column (number of predecessors ; see Fig. 3) is decremented by one.

c) If this number is now zero, this means that the corresponding task can be activated - i.e. put in RQ - : as a matter of fact all of its predecessors are terminated.

The steps 1-3 are repeated until RQ is nonempty. If CQ is empty at step 1, the processor enters a waiting state.

In the remaining columns of the control table and in the memory map there are the informations needed to actually start the execution of a task.

All the starting procedures are carried out by the processor that is going to execute the task.

In the last column of the control table the processor finds the address of the first instruction of the task and in the last but one it finds the names of the logical pages used; the addresses of the corresponding physical pages are found in the memory map of the graph.

We want to note that SEM operates with dynamic allocation of memory pages: the pages are allocated during the starting procedure of the first task that uses them and then released as soon as all the tasks referring to the corresponding logical pages have been terminated.

Because of this fact, the mean number of physical pages allocated for one graph during its execution can be considerably less than the total number of pages in the memory map.

3.2.3. - Extensions to multiprogramming.

A possible situation in SEM is the parallel processing of more than one graph. In that case, the management goes in a very analogous way, with the differences that we will point out in this section.

In RQ and CQ there are now tasks belonging to different graphs: each task name has a label with the name of its graph in order to let the processor enter the right control table during the supervisive actions performed in updating the queues.

To achieve an higher multiprogramming level and thus to exploit the capacities of the hardware, it is possible to have more than one execution of one graph at the same time with different inputs.

Each of these executions - that are, we repeat, distinct and simultaneous - will be called "activation" of the graph.

Each activation will need a different memory allocation for its data; i.e. different physical pages, one for each activation, will correspond to the same logical page of the graph. From this point of view the system has to handle a distinct memory map for each activation.

Of course, the system will delete the maps of the activations that have terminated their execution.

In a similar way, it will immediately infer that, in order to manage the RQ and the CQ, the system needs one copy of the second column of the control table for each activation: we have shown before that this part of the control table is modified during the execution of the corresponding graph.

In order to avoid the presence in memory of as many copies of the code of each task as the number of expected activations of the graph, the tasks have to be written in a "reentrant" way, i.e. granting that their instruction field will not be modified during the execution.

Assuming this property for the programs, two or more processors can execute simultaneously the same task within distinct activations of the same graph.

3.2.4. - Dynamic allocation of the memory.

In the former sections we introduced the capability for SEM to allocate dynamically the memory pages for the tasks.

The reasons for that are the following

a) The difficulty, in parallel programs, of reusing for different purposes and in different parts of the program the same memory pages; this comes from the lack of a strong sequentiality in such programs: as a matter of fact, it is often unpredictable the relative order of execution of two tasks.
b) The need, once that we have permitted several parallel executions of the same graph, to have different physical pages for the same logical ones.

For pratical reasons, the part of the memory available for dynamic allocation, that is the data area, is partitioned in pages.

Each page has 256 locations, according to the characteristics of the processors that we have used to implement the system (The Data General NOVA 3).

At system level, each block of common memory contains a list of pages available for allocation (free pages).

Whenever the allocation of a new page is needed, the allocation routine looks for a free page in the lists. The routine will be able to balance the load on every memory block. This balancing has been introduced to reduce the memory access conflicts.

As soon as one page is released, its address will return in the list of its memory block.

3.3. - Input-output.

In the SEM system each processor can perform any output operation by preparing the relative record in a special buffer allocated at the beginning of the task in the common memory. At the end of a graph

a specialized task will be put in the "waiting queue", ready to be executed by the processor which have the appropriate I/O device on the bus. This task will link all the output records together and it will start the device. The conversion of the data in the right code and the output of all the characters in sequence will be carried out by the interrupt system routines, which can be loaded in the private memory of the processor with the I/O device. Also the input is acquired by the interrupt system in a memory buffer.

The transfer of the input data to the destination addresses is performed at the end of the input operation by the same processor which have the input device.

This I/O system allows all the processors to execute input/output independently from the presence of the physical device on their own I/O busses and hence it avoids us to provide, at least in this first stage of the project, a bus-multiplexing for the input-output busses. Moreover this choice minimizes the program execution delay due to the I/O operations; this fact is especially important during the data acquisition from the user's experiment.

The dynamic allocation of the I/O buffers for the tasks is very useful in a multiprocessor architecture like SEM, because it allows a better exploitation of the memory resources of the system.

4. - HARDWARE STRUCTURE. -

4.1. - General description.

The modularity is the basic concept of the hardware structure of the SEM system (Fig. 1).

It is possible in this way to increase the system capacity without logical problems.

The MX circuit must perform two tasks:

a) to do the electrical connection between a common memory and the processor which needs it;
b) to resolve the conflicts, using a priority rule, when more processors attempt to simultaneously use the same memory unit.

The priority rule is dynamically selected according to the system operation and to the particular application.

The design of the MX circuit allows normally to resolve the conflict of more processors with simultaneous requests to the same memory unit, in the chronological order of request arrival. Therefore any processor cannot engage the memory for more than one cycle in sequence, if other processors are concurrent for the access.

When the management of special periferals is necessary this priority rule is not adequate; see, for example the transfer of data

blocks from CAMAC instrumentation, by DMA.

Then a device which generates other priority rules upon request is foreseen in the MX circuit to meet these needs.

4.2. - The MX circuit.

Also the MX circuit has been designed with a modular structure, according to the stated philosophy.

In one MX circuit (Fig. 4) there is one arbiter module and more interface modules (one for each processor).

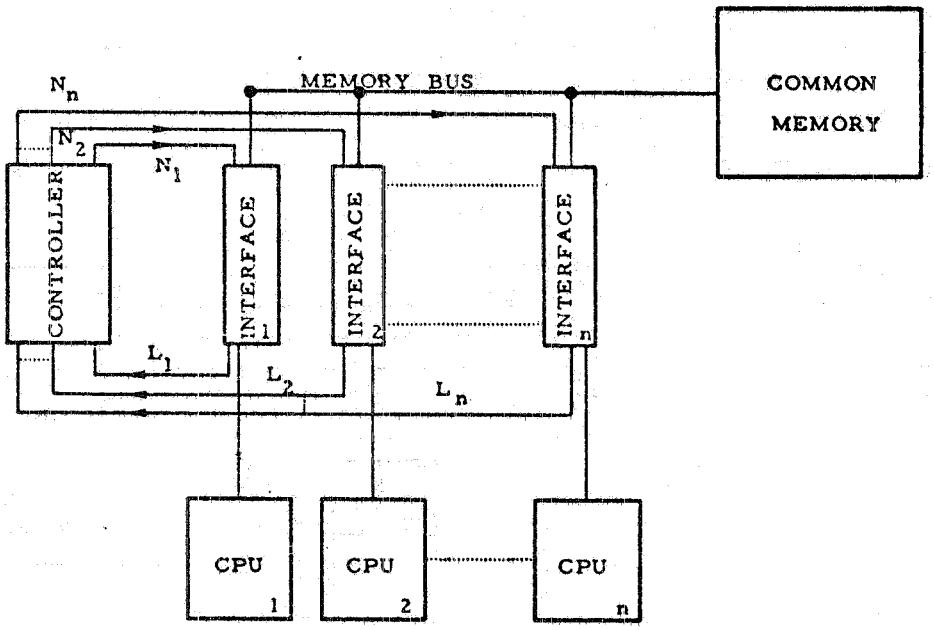


FIG. 4 - Block diagram of the MX system.

The arbiter module insures that processor requests are honored according to an assigned priority rule. The interface modules carry out the electrical connection between the corresponding processors and the common memory unit.

4.3. - The Arbiter module.

When a processor Pi requests the access to a common memory unit, it sends a signal on the L_i line (Fig. 4) of the arbiter module which replies with the acknowledged signal on the N_i line.

The N_i signal allows the processor Pi to access the memory unit.

In this way the arbiter module activates the interface modules of the various processors in sequence, until all requests have been serviced. Also the arbiter has been designed with modular structur-

ing (Fig. 5) and this meets the implementation need of various priority rules.

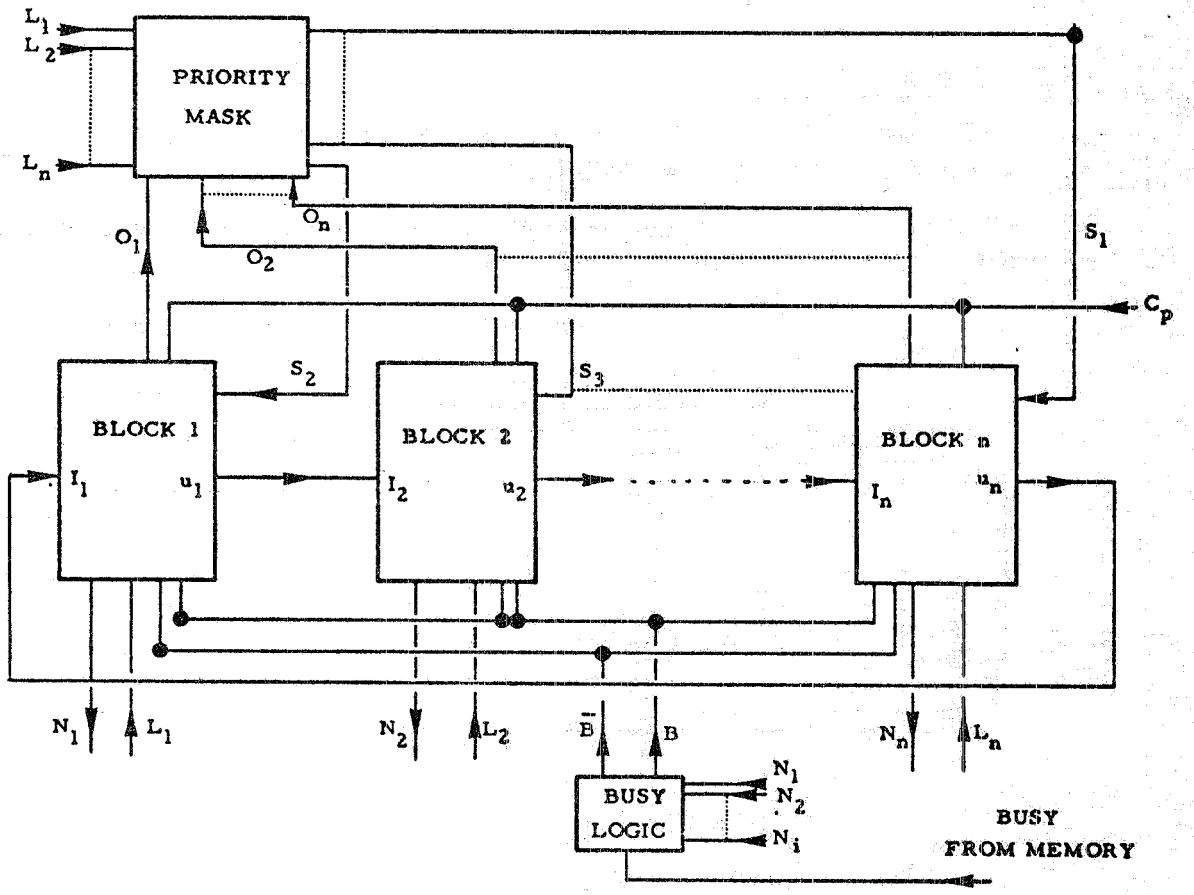


FIG. 5 - Block diagram of an arbiter submodule.

Every j-th arbiter submodule receives on the $L_i(j)$ lines the access memory requests, from some interfaces connected to it and it replies to them through the $N_i(j)$ lines (in Fig. 5 only one priority level j is shown for simplicity).

The $N_i(j)$ acknowledges are according to the "chronological" arrival order of the $L_i(j)$ signals.

Between one arbiter submodule and the other, there is instead a hierarchical structure. In fact, when a submodule (with a j level priority) has $L_i(j)$ requests to acknowledge, it honores the processor requests by inhibiting the activity of all submodules with lower priority, if it is not inhibited from a higher priority submodule.

In this way every arbiter submodule is a subsystem of the processor whole which uses the same common memory unit.

The processors in the same subsystem have no priority between them, but they have absolute priority on the processors which use

an arbiter submodule with lower priority.

In particular an arbiter module may have only one submodule (as in Fig. 5): than all the processors, which use this arbiter, have the same access priority to the common memory unit.

Now in the SEM system, there is the arbiter with only one submodule. The Fig. 5 displays the block diagram of an arbiter submodule.

The principle of operation is the following:

a) Dynamical operation.

Every block has an I_i input, which is activated by the preceding block, and a u_i output, which enables the next block. A block may be activated only once. The enabling signal goes from one block to the next at every pulse of the C_p clock. The logical operations performed by a block of the arbiter are described in the state diagram of Fig. 6.

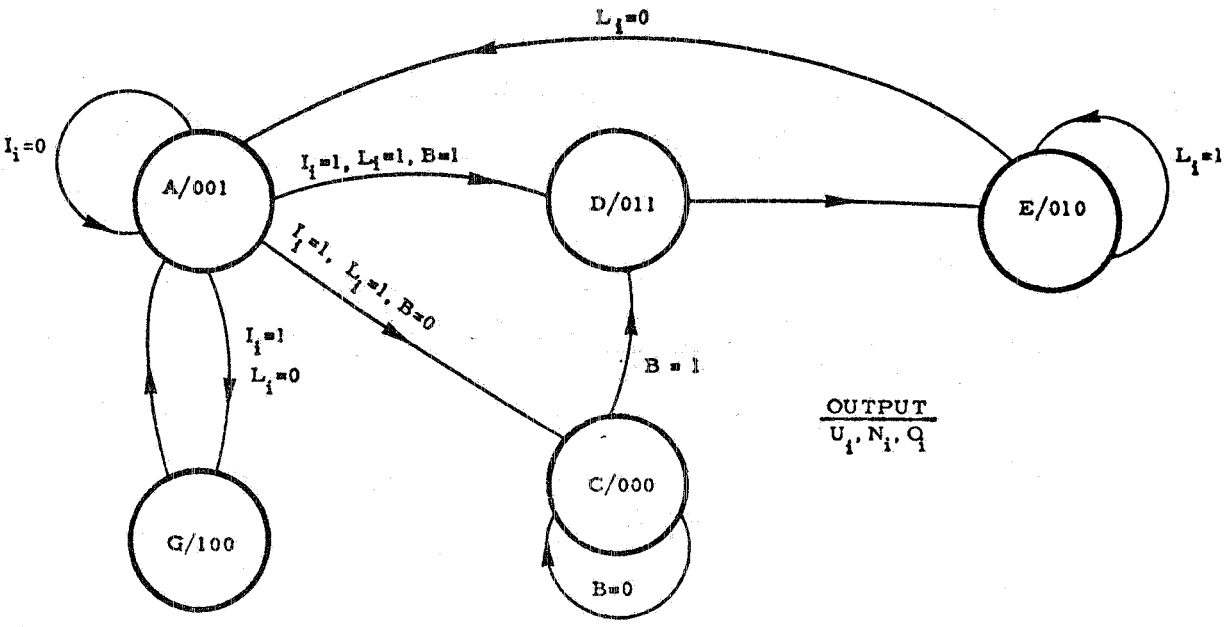


FIG. 6 - Flow graph of the i-th arbiter submodule block.

b) Idle state (without access requests to the memory).

The "priority mask" circuit (Fig. 5) is activated when all the blocks are in the A state. This is pointed out by the O_i lines.

In this state, a L_i request (from one interface module) enables the output preceding the "i" block which is then activated. If more L_i requests arrive at the same time only one enables the "i" block, because of the circuit logic.

The arbiter submodule goes in the idle state if it does a full cycle without finding L_i requests, in order to avoid the idle times if the L_i lines scanning continues indefinitely.

In fact, if the i-th interface requests the memory access, when the enabling signal is on the k-th block and B is equal 1 (idle memory), it would wait uselessly for some clock cycles before to allocate the memory.

Instead the "priority mask" system permits to allocate the memory instantaneously to the first interface which asks to access them, when the arbiter submodule is in the idle state.

In this way, when the memory is idle, the MX system introduces a delay of only 100 ns.

However it is expedient to know that the common memory access time of the SEM system may be reduced by adopting commercial memories with a faster cycle.

4. 4. - The Interface module.

The Fig. 7 displays the block diagram of an interface module.

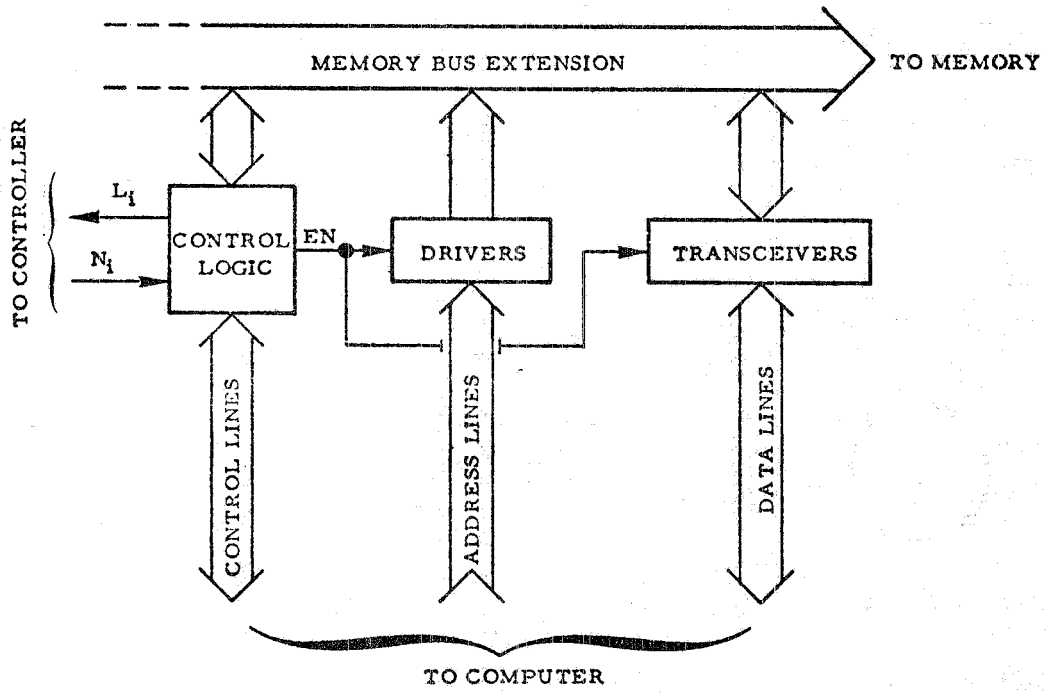


FIG. 7 - Block diagram of the interface module.

The control logic answers to the processor control lines asyn cronously. In particular, after an access memory request, the interface control logic keeps the processor in a waiting state, until it is activated to allocate the memory.

The interface module also manages the L_i line by which it sends the request to the arbiter module and it receives the acknowledge N_i.

The interface module may also keep the request signal on the L_i line, to allocate the memory for an indeterminate time. In this way the interface module allows to transfer a data block by means of DMA.

However this option is not yet inserted in the first prototype.

A control logic with a register accessible by the processor bus and with a comunication of more individual L_i and N_i lines is now in progress to build the dynamical reconfiguration in subsystems, already discussed in the preceding paragraph.

The drivers on the control, addressing and data lines are activated by the control logic and they have built to have a transit time lower than 10 ns.

The whole system already built has an arbiter module with four submodules and two interface modules.

ACKNOWLEDGEMENTS. -

The authors wish to thank C. Bosio, A. Mulachié, G. Piano Mortari and A. Vallone, who all contributed to this work. Many thanks are also due to Prof. G. Bellettini and Prof. S. Tazzari who made available the facilities of the Laboratori Nazionali di Frascati for this work.

REFERENCES. -

(1) - C. Verkerk, Special Purpose Processors, Lectures presented at the 3rd CERN School of Computing (Bergen, 1974).

(2) - G. A. Anderson and E. D. Jensen, Computer Interconnection Structures: Taxonomy, Characteristics and Examples, Computing Surveys 7, 197 (1975).

(3) - J. V. Levy, Computing with Multiprocessors, SLAC-161 (1973); see also IFIP (Nice, 1975) and Euromicro Symp. (Venice, 1976).

(4) - R. Biancastelli, Sistema modulare per una rapida elaborazione di dati in linea, Report ISS-73/4 (1973); R. Biancastelli et al., Proposta di realizzazione di un sistema modulare di elaborazione parallela rapida, Report ISS-74/12 (1974).

(5) - R. Biancastelli et al., Studio per la realizzazione di un sistema modulare di elaborazione parallela rapida, Proc. 2ª Conf. Interdisciplinare dell'INFN (Bari, 1975); C. G. Bell and P. Freeman, C. ai, A Computer Architecture for AI research, Proc. Fall Joint Computer Conf. (1972); W. A. Wulf and C. G. Bell, C. mmp, A multimini-processor, Proc. Fall Joint Computer Conf. (1972).