IST. ... BIBLIOTECA
Posiz. ARCHIVIO
E73-4

A MODEL FOR PIPELINE COMPUTERS AND ITS EFFECTIVENESS *

Gianfranco CASAGLIA
Nuovo Pignone at Centro ENI
Pisa, ITALY

Giorgio CONTI
Nuovo Pignone at Centro ENI
Pisa, ITALY

Marco VANNESCHI
Nuovo Pignone at Centro ENI
Pisa, ITALY

ABSTRACT

A model of computer organization is proposed, which is focused on the concept of pipelining of the phases that can be distinguished during the execution of a generic instruction. Four main units appear in such organization: Instruction Memory Unit (IMU), Data Memory Unit (DMU), Instruction Preparation Unit (IPU) and Instruction Execution Unit (IEU). These units operate as mutually asynchronous stages of a pipeline and exchange information by means of single-channel queues with queue discipline First-In-First-Out.

An analytical evaluation of the memory bandwidth, obtained by dividing the memory into the two parts IMU and DMU, is made.

The performance of the whole system, computed as number of instructions executed per second, is evaluated under simplified, but general enough, hypotheses; then a comparison with the performance obtainable with a stricly synchronized pipeline system is made.

Finally, a method for designing the cooperation control of a system, realized according to the proposed model, is given and the problem of the influence of such control realization on the performance of the whole system is introduced.

1.- INTRODUCTION

The need of realizing more and more fast and efficient computer systems, which are able to meet the requirements of more and more high performance, has caused, since several years, the introduction of sophisticated design techniques, made feasible by the development of technology. Two main types of such techniques can be distinguished: by the first, a system is realized as composed of a number of subsystems operating in parallel on essentially independent information-streams; according to the second, a system is realized as a cascade of subsystems simultaneously executing a different "phase" belonging to the same information-stream.

The last technique, called "pipeline", was introduced [1], [2], [3] as a way of designing more fast units, such as the execution unit or the instruction decoding unit of high-speed computing systems. As a consequence, the denomination of pipeline systems was extended to those computers, which, composed of several units operating in look-ahead or according to the concurrent-SISD or MISD organization (see Flynn's high-speed computers classification [4]), contain one or more of such units realized as a pipeline: for example, the execution units of IBM 360/91 [5] and of CDC 7600 [6]. However, systems of this type are not realized as a pipeline "on the whole".

In this paper we propose a model of computer organization in which the pipeline concept represents the basis for the design of the "whole" system: such organization don't necessarily require that one or more units are realized as a pipeline, but it is chiefly characterized by the fact that each unit is a pipeline stage realizing a different "phase" of the execution of the generic computer instruction.

For this purpose, we shall show that it is suitable to divide the first level of memory hierarchy in a part containing exclusively instructions and in another part containing exclusively data. Moreover, various units, mutually asynchronous and with processing time in general variable, exchange data by means of single-channel queues with queue-discipline First-In-First-Out.

An analytical study of the two above-men

* This work has been sponsored and supported by the Convention on digital systems for process control between Consiglio Nazionale delle Ricerche (C.N.R.) and Ente Nazionale Idrocarburi (E.N.I.).

tioned characteristics is made, aiming to show their convenience in comparison with more traditional solutions (i.e. a single memory containing instructions and data, and a strictly synchronized way of exchanging data among directly connected units).

Finally, a method for designing the cooperation control of a system realized according to the proposed model is given, by means of which it is possible to grant a correct behaviour (i.e. determinate and free from deadl... situations), taking into account some performance requirements of the whole system.

2.- THE MODEL

We consider initially the model of look-ahead computer shown in fig.1. The three main units, called Memory Unit (MU), Instruction Preparation Unit (IPU) and Instruction Execution Unit (IEU), operate simultaneously on distinct phases of instructions processing.

MU, storing instructions and data, supplies through the "look-ahead buffer" b_4 instructions to IPU, sends operands to IEU and receives instruction results from IEU. MU is therefore the system bottleneck, degrading the performance otherwise reacheable according to processing speed of IPU and IEU.

IPU receives from MU one instruction at a time, sends each decoded instruction to IEU and each computed operand address to MU. Utilization of IPU is essentially limited by the presence of jump instructions in the instruction stream; we denote by λ the jump probability in the instruction stream.

Unconditional jumps are executed by IPU and, once decoded, cause the loss of all read instructions (stored in buffer b_4) and of all instructions which MU is presently reading. Denote by λ_u the probability that a generic instruction is an unconditional jump instruction.

Decoding a conditional jump instruction causes transferring an auxiliary TEST instruction to IEU: IPU continues to decode instructions starting from the one following the jump and stores the jump-target address. When IEU receives a TEST instruction, it attempts to verify if the condition is ...; if yes, it is necessary to delete the effects of the instructions decoded by IPU, and of instructions and data read by MU, after IPU decoded the jump instruction. We denote by λ_c the probability that a generic instruction is a conditional jump instruction for which the jump condition is true, we have $\lambda = \lambda_u + \lambda_c$.

Instruction stream discontinuities, through the data path MU-IPU-IEU-MU, may be caused, besides by jump instructions and memory clashes, by instructions of the "store" type or by instructions which operate on registers or memory cells whose contents must be used by preceding instructions. We shall consider the effects of instructions of the last type in the choice of processing time distributions of the units.

For the purpose of reducing the number of memory clashes, we propose a model of computer organization, in which the memory is divided into two distinct parts: Instruction Memory Unit (IMU), containing instructions, and Data Memory Unit (DMU), containing data used by the program stored in IMU.

Moreover, if the various units are mutually asynchronous and with processing times in general variable, system performance can be improved by inserting some queues on the connections among the units; queues allow to execute predictions of conditional jumps in look-ahead mode, reducing so the performance degradation caused by instructions of this type.

The correct execution of queue operations can be obtained by means of the "producer-consumer" algorithm described in [7].

The model is shown, in its essential parts, in fig.2.

Since the results, which IEU should write in DMU, can be used by an instruction successively decoded by IPU, it may be useful to realize queue q_4 in an associative manner, so that DMU can look for an operand directly in q_4. This queue is therefore an associative buffer, into which IEU sends the results of its processing; such results are written in DMU either when q_4 is full or when the DMU load is low enough. Queue discipline of q_4 may be simply First-In-First-Out or a more sophisticated one, depending on the program structure.

A computer system realized according to the proposed model is therefore represented by a pipeline of four main units connected by queues; we call a similar structure a "pure pipeline", aiming to distinguish it from the most general case of pipeline systems represented by queueing networks.

About this model we intend:

a) to evaluate the IMU-DMU memory organization from the point of view ... performance (i.e. its bandwidth) ... point of view of its applic...

real environment;

b) to derive the queue lengths, taking into account the distributions of processing times of the stages connected by queues; and to evaluate the performance of the whole system, computed as number of instructions executed per second, and the efficiency figures of each stage, taking into account the effects of the finite lengths of queues;

c) to compare this organization with another organization, very frequently used, in which the various stages are strictly synchronized (i.e. pulsed by the same clock) and connected each other directly.

3.- EVALUATION OF THE MEMORY ORGANIZATION

The organization containing two physically distinct memories, IMU and DMU, comes out, besides from the idea of obtaining a model of a pipeline computer in which information flow is more corresponding to the pipeline characteristics, also observing that instruction stream is essentially sequential (out of sequence conditions are produced by jump instructions) and that data stream is nearly random; two distinct structures (different access methods) therefore allow to optimize the memory bandwidth.

We made the following hypotheses:

* memory unit (MU) is divided into n modules;
* n_1 modules are dedicated to the Instruction Memory Unit (IMU) and n_2 modules ($n_1 + n_2 = n$) are dedicated to Data Memory Unit (DMU); IMU and DMU are interleaved n_1 ways and n_2 ways, respectively;
* all the memory modules of IMU and DMU operate synchronously, i.e. their cycles are overlapped and with identical memory cycle time T. (*)

We want to evaluate the overall memory bandwidth for an optimal partition (n_1, n_2) in terms of λ and of α ($1/n_2 \leq \alpha \leq 1$), probability of sequential accesses to data. The results are compared with those obtained by Burnett-Coffman [8] under different hypotheses. Also useful for the comparison are the results obtained by Hellerman [9] [illegible] n-ways interleaved memories in which the access to each module is random, i.e. with probability $1/n$.

<u>Hellerman's hypothesis:</u>

$$B_n = \sum_{k=1}^{n} \frac{k^2 (n-1)!}{n^k (n-k)!}$$

When $1 \leq n \leq 45$, B_n can be approximated by the expression

$$B_n \simeq n^{.56} \qquad (3.1)$$

with an error less than 4%.

<u>Burnett-Coffman's hypothesis:</u>

Memory cycles are alternatively dedicated to reading instructions and to accessing data, following the scheme depicted in fig.3. Therefore we have a bandwidth, in terms of λ, referring to instructions:

$$IB_n = \frac{1-(1-\lambda)^n}{\lambda} \qquad (3.2)$$

and a data bandwidth DB_n in terms of α and $\beta = (1-\alpha)/(n-1)$, probability that a data access is out of sequence and referred to one of the (n-1) remaining modules. Data bandwidth can be derived by a recursive enumeration procedure; the numerical results reported here are derived from [8].

The overall bandwidth can be derived by combining IB_n and DB_n taking care that statistical results carried out by IBM 360/91 [10] suggest that approximately 80% of all instructions require an operand (data) reference to memory.

B_n is therefore

$$\begin{cases} B_n = \dfrac{IB_n + DB_n}{2} & \text{if } DB_n = .8\ IB_n \\ B_n = \dfrac{IB_n + .8\ IB_n}{2} & \text{if } DB_n > .8\ IB_n \\ B_n = \dfrac{C_1 \cdot IB_n + C_2 \cdot DB_n}{[illegible]} & \text{if } DB_n < .8\ IB_n \end{cases} \qquad (3.3)$$

being $C_1 < 1$ and $C_2 > 1$, calculated as follows:

$$\begin{cases} C_2 \cdot DB_n = .8 C_1 \cdot IB_n \\ C_1 \ [illegible] \end{cases}$$

This last set of equations implies that the system must periodically add an extra data cycle in order to avoid overflowing any one unit of buffering allocated to the data request buffer.

<u>Divided memories hypothesis:</u>

Memory cycles and decoding cycles are depicted in fig.4. IMU bandwidth is [illegible]

(*) It is possible to verify that, referring to the bandwidth, this hypothesis is equivalent to that in which IMU(DMU) cycles are staggered of a $\tau_1 = T/n_1$ ($\tau_2 = T/n_2$).

by (3.2) multiplied by a coefficient K taking into account that, if the i.th ($1 \leqslant i \leqslant n$) decoded instruction is a jump, we lose not only the remaining ($n_1 - i$) instructions but also the n_1 instructions that the memory is reading when the jump instruction is decoded.

The coefficient K is derived observing that a waste of n_1 instructions for each jump is a waste of one memory cycle out of two memory cycles for each jump instruction. Being P_j the probability of finding a jump among n_1 instructions

$$P_j = \sum_{i=1}^{n_1} \lambda (1-\lambda)^{i-1} = 1 - (1-\lambda)^{n_1}$$

and ℓ the number of instruction memory cycles, we have:

$$K = \frac{1}{\ell}\left[\ell - \frac{\ell}{2}(1-(1-\lambda)^{n_1})\right] = \frac{1+(1-\lambda)^{n_1}}{2}$$

IMU bandwidth is therefore:

$$IB_{n_1} = \frac{1-(1-\lambda)^{n_1}}{\lambda} \cdot \frac{1+(1-\lambda)^{n_1}}{2} \qquad (3.4)$$

Numerical values of DMU bandwidth, DB_{n_2} are derived from [8].

The overall bandwidth can be calculated by expressions (3.5), similar to the (3.3).

$$\begin{cases} B_n = IB_{n_1} + DB_{n_2} & \text{if } DB_{n_2} = .8\, IB_{n_1} \\ B_n = IB_{n_1} + .8 IB_{n_1} & \text{if } DB_{n_2} > .8\, IB_{n_1} \\ B_n = \frac{DB_{n_2}}{.8} + DB_{n_2} & \text{if } DB_{n_2} < .8\, IB_{n_1} \end{cases} \qquad (3.5)$$

Results obtained applying (3.4) and (3.5)* are depicted in figg. 5÷7 in terms of n, assuming for λ and α the following values:

$\lambda = .125, \quad \alpha = \frac{1}{n}$ (random access to data)

$\lambda = .125, \quad \alpha = .5$

$\lambda = .2, \quad \alpha = \frac{1}{n}$ (random access to data)

Statistical results [8] allow to state that λ values between .2 and .1 are likely for a range of programs and that α values between .125 and .5 appear typical.

Diagrams of figg. 5÷7 show that a memory organization with two physically divided parts generally allows a better bandwidth.

[illegible] IMU bandwidth improvement.

[illegible] bandwidth waste can be reduced using [illegible]chronous memory and such that (a) it [illegible] possible to interrupt a read operation, without destroying the contents of the memory cell and (b) it is possible to request immediately a different read operation.

In this case a jump instruction in the i.th position ($1 \leqslant i \leqslant n_1$) implies, assuming as reference the expression (3.2), an additional waste of i instructions (fig.6).

Being ℓ the number of Instruction Memory cycles, the coefficient K' results:

$$K' = \frac{1}{\ell}\left\{\ell - \frac{\ell}{2}\left[\frac{\frac{1-(1-\lambda)^{n_1}}{\lambda}}{n_1}\right] \cdot (1-(1-\lambda)^{n_1})\right\} = 1 - \frac{1}{2}\left[\frac{(1-(1-\lambda)^{n_1})^2}{\lambda n_1}\right]$$

and

$$IB_{n_1} = K' \cdot \frac{1-(1-\lambda)^{n_1}}{\lambda}$$

3.2.- DMU bandwidth improvement

When data to be accessed are nearly random, an interleaving organization of the memory is not the most convenient; in this case in fact the bandwidth is the mean length of all the strings of data requests without collisions.

An improvement can be obtained by associating a queue to each DMU module; this queue allows to store multiple requests to each module. Such a solution has been proposed by Ravi [11] and evaluated for random accesses. In fig.5 we depicted the bandwidth of a memory organization with two distinct parts, in which each module of DMU has its own queue; the improvement appears to be satisfactory for $n > 6$.

3.3.- Design consideration about memory organization with IMU and DMU.

The division of the memory into two physically distinct parts, IMU and DMU, allows an improvement in the overall bandwidth, but imposes serious constraints to the ratio between the dimension of the program and the dimension of the relative data. In a general purpose computer this ratio is unpredictable. A more realistic solution is obtained inserting IMU and DMU at the first level (i.e. the fastest) of a memory hierarchy. Assuming this first level paged, the problem of the ratio between program and data dimensions is transferred at the level of the activity (working set) of each set of information. In this case therefore the question is not a limited memory space for data or for programs, but an unbalancing of

* Results obtained from (3.5) and inserted in figg 5÷7 refer to an optimal partition (n_1, n_2).

the activity of each set of information and a degradation of the system performance.

We can also imagine a solution in which each program has its own optimal partition (n_1, n_2) calculated not only in terms of the first level memory bandwidth, but also in terms of the traffic between IMU and second memory level and between DMU and second memory level imposed by the partition (n_1, n_2).

These considerations remain valid in a multiprogrammed environment, since in a specific instant the actual partition (n_1, n_2) is that belonging to the program in execution.

Concluding we can assert that the proposed solution must be evaluated as inserted in a system with a memory hierarchy.

4. QUEUEING ANALYSIS OF THE MODEL

Consider the scheme of fig.9; with respect to the scheme of Sect.2, the connection between IEU and DMU by means of q_4 is not considered. Nevertheless the results of the analysis is still meaningful, first of all because q_4 can be considered as a queue of infinite length and so (as we will see) it does not substantially modify the behaviour of the IEU (and hence of all the other stages); besides because we suppose that the distribution of the requests from q_2 to DMU already takes into account the number of write-cycle requests which can arrive from q_4, involving an approximation for excess in computing the length of q_2.

With this schematization, the analysis is brought again to the case of a "pure pipeline". In the case of fig.9, IMU is the generator stage (S_a); IPU, DMU and IEU are the service stations (S_{s1}, S_{s2}, S_{s3} respectively); the queues q_1, q_2, q_3 have finite length L_1, L_2, L_3 respectively and FIFO is their queue discipline. The analysis of such a system can be made by means of the method shown in [12] and the comparison with a synchronized system can be done relating this system to the case in which all queues have length equal to zero.

The results obtained in [12] are essentially the following:

Let t_{a}, t_{s1}, t_{s2}, t_{s3} be the random variables: time interval between two succeding issues of items from stage S_a, service time of stage S_{s1}, S_{s2} and S_{s3} respectively; let these variables have known distributions, with mean T_a, T_{s1}, T_{s2}, T_{s3} respectively. These distributions are the ones initially supposed, as if queues were all of infinite length.

a) Due to finite length queues, the "actual" distribution of the variables t_a, t_{s1}, t_{s2} is different from that initially supposed; let t_{aib}, t'_{s1}, t'_{s2} be the new random variables: their means T_{aib}, T'_{s1}, T'_{s2} are increased, with respect to T_a, T_{s1}, T_{s2}, of the quantities $P_{L1} T'_{s1}$, $P_{L2} T'_{s2}$, $P_{L3} T_{s3}$ respectively, where P_{Li} is the <u>block probability</u> of generic queue q_i, i.e. the probability that an item to be put into q_i finds the queue q_i full. The distribution of t_{s3} remains on the contrary unmodified. A method for obtaining the distributions of t_{aib}, t'_{s1}, t'_{s2} is given in [12].

b) Independently from arrival and service distributions, the mean of the random variable time interval between two succeding outputs from a generic stage S_{si} is equal to the mean of the random variable time interval between two successive arrivals to the queue q_i at the input of stage S_{si}. This result is valid within the same bounds of approximation in which the probability P_{oi} that a single-channel queue q_i is empty can be expressed by [13]:

$$P_{oi} = 1 - \rho_i$$

where ρ_i is the utilization factor of the queue q_i computed as the ratio between the "actual" mean service time and the "actual" mean interarrival time:

$$\rho_i = \frac{T'_{si}}{T_{aib}}$$

with the symbols of fig.9.

A method to obtain the distribution of t_{aib} for every queue q_i is given in [12]. From the preceding result it follows that the mean time interval between two succeeding outputs from the last stage of a pure pipeline is equal to the actual mean time interval between two successive issues from S_a. Really, such a mean becomes certainly higher if one wants to take into account the "blocks" of the pipeline caused by jump instructions inside the program. When an unconditional jump occurs, it is necessary to empty the queue q_1 only; when a conditional jump occurs, and the jump condition is verified, it is necessary to empty all the queues q_1, q_2, q_3. To obtain the output from a stage of the pipeline under these conditions, consider first the system in fig.10 where an item can be taken out of the queue by server S_2 with probability λ and by server S_1 with probability $(1-\lambda)$. T_1 and [illegible]

the mean service time of S_1 and S_2 respectively.
The utilization factor of the system is given by:

$$\rho = \frac{(1-\lambda)\ T_1 + \lambda T_2}{T_{a1}}$$

If we suppose now that the stage S_2 has mean service time equal to zero, the preceding situation represents what happens in a pipeline computer when a queue must be empted (supposing instantaneous such operation). In these cases, then, it is:

$$\rho = (1-\lambda)\frac{T_{s1}}{T_{a1}} = (1-\lambda)\rho_1$$

ρ_1 being the utilization factor of the queue computed without considering the occurrence of jumps.
The mean time interval between two successive outputs from stage S_1 is computed by:

$$T_{a2} = T_{s1} + P_o\ T_{a1}$$

Since $P_o = 1-\rho$, it is:

$$T_{a2} = T_{a1} + \lambda T_{s1} \qquad (4.1)$$

In the case of the system in fig.9, the mean time interval between two successive outputs from stage S_{s3} is given by:

$$T_{OUT} = T_{aib} + \lambda_u T'_{s1} + \lambda_c (T'_{s1} + T'_{s2} + T_{s3}) \qquad (4.2)$$

In a pipeline computer, T_{OUT} is the mean time elapsed between the execution of two successive instructions, so the parameter:

$$I = 1/T_{OUT} \qquad (4.3)$$

gives the <u>mean number of instructions executed in the unit of time</u>, which is a meaningful parameter, giving a measure of system performance. Other meaningful parameters are:

- <u>efficiency of stage S_i as producer</u>:

$$\eta_{pi} = \frac{T_{ai}}{T_{aib}} \qquad (4.4)$$

- <u>efficiency of stage S_i as consumer</u>:

$$\eta_{ci} = \frac{T_{si}}{T_{aib}} \qquad (4.5)$$

- <u>service efficiency of stage</u> S_i :

$$\eta_{si} = \frac{T_{si}}{T'_{si}} \qquad (4.6)$$

The analysis of the system in fig.9 is made by evaluating the parameters (4.3), (4.4) (4.5), (4.6) for systems with queues of various lengths, including the case $L_1 = L_2 = L_3 = 0$. The evaluation of the queue parameters which appear in the formulae mentioned above was made by simulating the system with GPSS/360.
The initial distributions, i.e. the distributions of t_{a1}, t_{s1}, t_{s2}, t_{s3}, were supposed to be:

$F(t_a)$: regular
$F(t_{s1})$: Erlang-4
$F(t_{s2})$: regular
$F(t_{s3})$: Erlang-5

These hypotheses have been suggested by experience and by computations made by other authors, like Lorin [14].
Varying the mean values of the distributions $F(t_{s1})$ and $F(t_{s2})$ and λ_u and λ_c, we obtained the diagrams of figg.11÷16.
From these we can note essentially that:
a) a pipeline computer with queues is better than one without queues with regard to all the parameters evaluated, being equal the values of T_{a1}, T_{s1}, T_{s2}, T_{s3}, λ_u, λ_c.
In the analyzed cases, the parameter I becomes better of about 20%.
b) For every set of values T_{a1}, T_{s1}, T_{s2}, T_{s3}, λ_u, λ_c and with the same initial distributions of t_{a1}, t_{s1}, t_{s2}, t_{s3}, it is possible to find, once fixed the dimensions of some queues, the best sizes for the other queues, in the sense that for greater sizes the evaluated parameters do not vary considerably.
In particular, the sizes relative to the highest curves of figg.11÷16 allow to obtain the same values of the evaluated parameters which we should obtain with queues of infinite length.
About the cost of the solution with queues in comparison with the one without queues, we can say that the relative increase of the system cost due to queues is certainly more than counterbalanced by the increase of the performance. But a more considerable result is that always it is possible to design a pipeline computer with queues which has greater or equal performance than one without queues, but some or all stages of the former can nevertheless be chosen with smaller operating speed, and so cheaper.
In this connection, it is at last to be noted that the synchronization system for the exchange of data among the stages is practically the same, in the case of a pipeline computer with queues or without queues, both with regard to the hardware necessary to its implementation and with [illegible] the time spent by the stages to perf[illegible] rations requested for the synchroni[illegible] self, if the synchronisation is performed [illegible] the data; if the stages are synchronized [illegible]

the same clock, the hardware is more simple, but not so much to invalidate the results reported here.

5.- COOPERATION CONTROL

5.1.- Information exchange disciplines.

The definition of any system composed of mutually asynchronous and cooperating units is completed by the definition of the disciplines by means of which the cooperation of the units is obtained, i.e. the rules by means of which the operation parts of the units exchange data and the control parts exchange control messages, called commands.

In the preceding section we discussed the data exchange; in this section we shall make some considerations about command exchange. As an introductory example, in the proposed model of pipeline computer, commands are exchanged between IPU and IMU for the purpose of informing IMU of a variation in the instruction stream caused by a jump instruction; and for the purpose of informing IPU of the completion of the operations executed by IMU after receiving the jump command, so that IPU can restart its processing. Another command is sent by IEU to IPU, informing IPU of a true or false jump condition.

Since units are mutually asynchronous, the command exchange has to obey some rules granting the correct system behaviour, i.e. granting determinacy and absence of deadlock situations.

Two rules of command exchange seems to fit our purposes; such rules are introduced in [15] to which we refer the reader for more detailed information, particularly about correctness.

Suppose the receiving unit, while in a certain state, examines its input command lines, for the purpose of verifying the presence of certain commands. The two command exchange rules are mainly characterized by a different behaviour of the receiving unit in the case that some (or all) such commands are not present. They are:

a) the receiving unit continues its processing, behaving as if the sending unit [illegible] commands to send it;

b) the receiving unit waits for the arrival of all commands expected in this state; at this point the unit starts again its processing; for the purpose of avoiding that the receiving unit waits infinitely, the sending unit must send, with a non-zero frequency, commands to the receiving unit: this may be eventually fictitious commands whose meaning is "nothing to communicate".

One can see that, in the case a), processing speeds of the units remain completely indipendent from each other; however only in particular cases the system behaviour will be determinate, i.e. only when each unit is designed in such a way that all the input command sequences, belonging to certain subsets, give rise to the same final result.
On the contrary, in the case b) it is possible to prove that the system behaviour is always determinate, but some constraints are imposed to the average processing speeds of the units; this fact may lead, in some realizations, to unnecessary limitations.

With regard to the structure of the command lines, the most general solution is that for which commands are transmitted by means of queues [15], whose management is similar to that of the data queues. A particular case, of remarkable importance, is that called "interlocking", for which a command is sent only after the previously sent command has been used by the receiving unit.

5.2.- Effecti[illegible] of command exchange disciplines.

The selection of the command exchange rule for a specific application depends widely upon the algorithm describing the cooperation control functions, once the conditions granting behaviour correctness are satisfied. For instance, in a pipeline system controlled, with regard to the jump management algorithm, by the three types of commands mentioned in the example at the beginning of par.5.1, rule a) is certainly the most suitable; in this case, the mechanisms by which a determinate behaviour is granted are: a command which causes resetting data queue q_1, when IPU decodes an unconditional jump instruction; and a command which causes resetting data queues q_1, q_2, q_3 when IEU verifies that a conditional jump was predicted in the wrong way.

Once selected the command exchange rule, we are able to define the probabilistic distributions of command generation and utilization times; therefore it is possible to select the best command line [illegible], a problem consisting essentially in [illegible] the optimum command queue length [illegible] problem solved in sect.4 for data [illegible]. However, the problem is, in this case, [illegible] cated by the fact that, with regard to the

best performance of the whole system, command queue lengths are not independent from data queue lengths: this imposes to analyze the system as a general network of queues (instead of a pipeline of queues). The analytical solution of this problem is not yet known [16], excepted for particular cases (i.e. particular distributions and infinite length queues).
Presently, the only possibility is therefore computer simulation. Nevertheless, it is sometimes possible to simplify the system model in such a way that the solution is easily reachable. For example, in the pipeline system already mentioned, the selected jump management algorithm and the command exchange rule a) lead to use the "interlocking" technique for the command line structure, with regard to the command "I decoded a jump" (sent by IPU to IMU) and "Prediction of conditional jump is false" (sent by IEU to IPU). As shown in Sect.4, the implications of these commands on system performance can be easily evaluated.

6.- CONCLUSIONS

In this work a model of computer organization, in which the overall system is realised as a pipeline, is presented.

The solution of dividing the first level of the memory into two parts, the first containing only instructions, the second one containing only data, has been analytically evaluated with the purpose of determining under which conditions this solution is more convenient than others utilizing a single memory for data and instructions.

In the proposed model all the instructions, excluding few exceptions, are elaborated through a "pure pipeline" composed of four mutually asynchronous units interconnected by FIFO single-channel queues; the analytical evaluation of this solution has shown some convenience, from the point of view of performance and cost, with respect to the solution characterized by synchronized units exchanging data by direct connections.

Some problems are still open; among them we note:

1) the effect of a IMU - DMU memory organization [illegible]ed in a memory hierarchy;
2) the extension, to a generic parallel system, of the asynchronous exchange of information among units, by means of queues connecting them.

We think that the exposed results and the statement given to the model are such that they can be thought of as a first step toward modeling and evaluation of digital systems organized in parallel.

ACKNOWLEDGEMENT

The authors which to thank their collegues F.Grandoni and P.Zerbetto for the useful discussions during the initial phase of the definition of the model.

NOMENCLATURE

B_n	: memory bandwidth
DB_{n_2}	: DMU bandwidth
DMU	: Data Memory Unit
I	: mean number of instructions executed per second
IB_{n_1}	: IMU bandwidth
IEU	: Instruction Execution Unit
IMU	: Instruction Memory Unit
IPU	: Instruction Preparation Unit
L_i	: length of queue q_i
MU	: Memory Unit
n	: number of memory modules
n_1	: number of IMU modules
n_2	: number of DMU modules
q_i	: generic queue
S_a	: generator stage of a pipeline system
S_{Si}	: generic service station of a pipeline system
T_i	: mean value of random variable t_i
t_{ai}	: initial interarrival time to queue q_i
t_{aib}	: actual interarrival time to queue q_i
t_{Si}	: initial service time of service station S_{Si}
t'_{Si}	: actual service time of service station S_{Si}
α	: probability of sequential accesses to data
λ	: jump probability in the instruction stream
λ_c	: probability of conditional jumps for which the jump condition is not verified
λ_u	: unconditional jump probability
η_{Ci}	: efficiency of stage [illegible]
η_{Pi}	: efficiency of stage S_i in pro[illegible]
η_{Si}	: service efficiency of stage [illegible]

REFERENCES

1 Cotten, L.W., "Circuit Implementation of High-Speed Pipeline Systems", 1965 Fall Joint Comput. Conf., AFIPS Conf. Proc., vol.27, pp.489-504.

2 Cotten, L.W., "Maximum-Rate Pipeline Systems", 1969 Spring Joint Comput. Conf., AFIPS Conf.Proc., vol.34, pp. 581-586.

3 Hallin, T.G., and Flynn, M.J., "Pipelining of Arithmetic Functions", IEEE Trans. on Comp., Vol.C-21, N.8, pp. 880 886, Aug.1972.

4 Flynn, M.J., "Very High-Speed Computing Systems", IEEE Proc., Vol.54, N.12, pp.1901-1909, Dec.1966.

5 Anderson, S.F., Earle, J.G., Goldschmidt, R.E., and Powers, D.M., "The IBM System/360 Model 91: Floating - point Execution Unit", IBM Journal of Res. & Develop., Vol.11, N.1, pp.34-53, Jan. 1967.

6 Lonseigneur, P., "Description of the 7600 Computer System", Computer Group News, Vol.2, N.9, pp.11-15, May 1969.

7 Dijkstra, E.W., "Co-operating Sequential Processes", in Programming Languages, F.Genuys (ed.), Academic Press, New York, 1968, pp.43-112.

8 Burnett, G.J., and Coffman, E.G., "A Study of Interleaved Memory Systems", 1970 Spring Joint Comput. Conf., AFIPS Conf.Proc., Vol.36, pp.467-474.

9 Hellerman, H., Digital Computer System Principles, Mc Graw-Hill Book Company, Inc., New York, 1967, pp.228-229.

10 Anderson, D.W., Sparacio, F.J. Tomasulo, R.M., "The IBM System/360 Model 91: Machine Philosophy and Instruction-Handling", IBM Journal of Res. & Develop., Vol.11, N.1, pp.8-24, Jan.1967.

11 Ravi, C.V., "On the Bandwidth and Interference in Interleaved Memory Systems", IEEE Trans. on Comp., Vol.C-21, N.8, pp.899-901, Aug.1972.

12 Casaglia, G.F., Conti, G., Grandoni, F., and Vanneschi, M., "Queueing Analysis of a Model for Digital Pipeline Systems", submitted to IEEE Trans. on Comp.

13 Morse, P.M., Queues, Inventories and Maintenance, John Wiley & Sons, Inc., New York, 1958, p.87.

14 Lorin, H., Parallelism in Hardware and Software: Real and Apparent Concurrency, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1972.

14 Casaglia, G.F., Conti, G., and Vanneschi, M., "A Model for Parallel Systems Composed of Mutually Asynchronous Modules", International Workshop on Computer Architecture, Grenoble, France, June 26-28, 1973.

16 Baskett, F., and Muntz, R.R., "Networks of Queues", 7th Annual Princeton Conf. on Information Sciences and Systems, Princeton University, March 22 23, 1973.

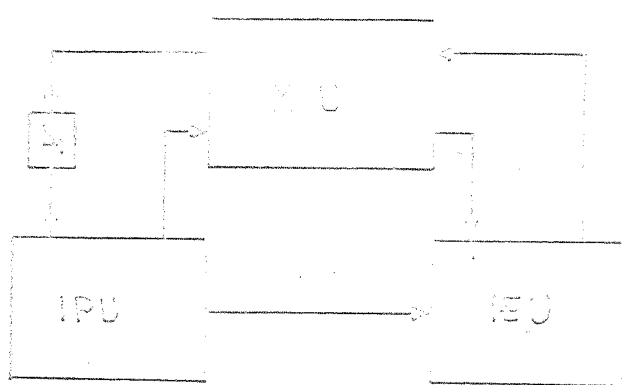

Fig.1-Model of look-ahead computer.

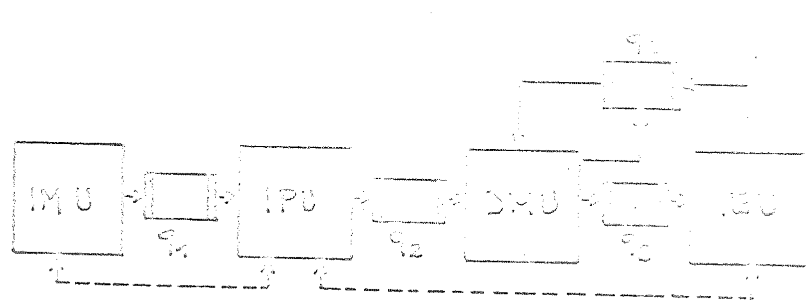

Fig.2-Model of pipeline computer.

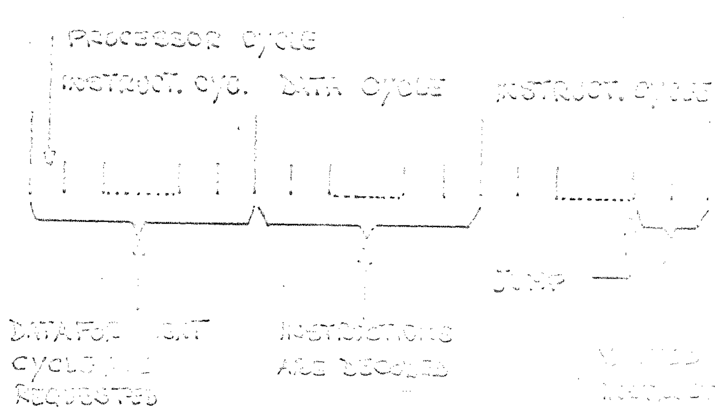

Fig.3-Memory accesses in the Burnett model.

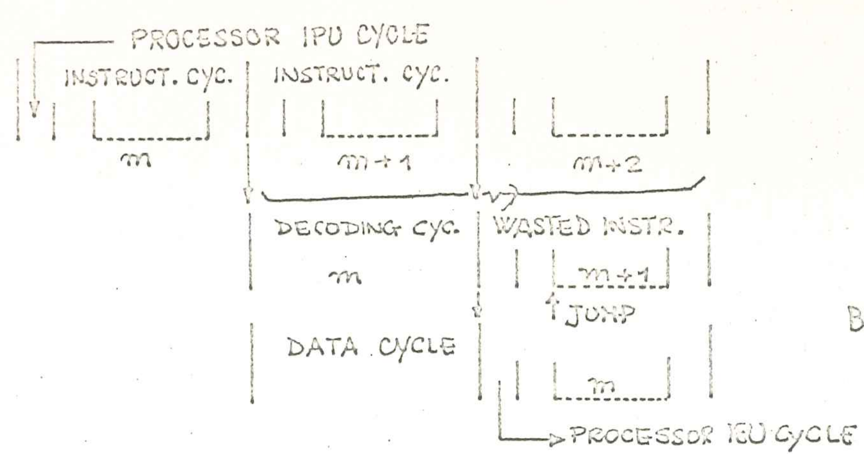

Fig.4-Memory accesses in the divided-memory model.

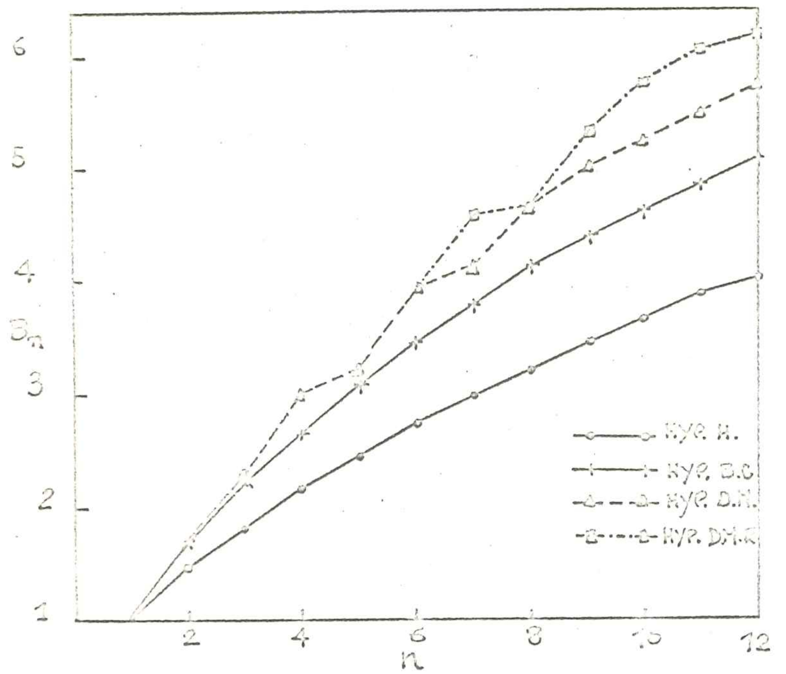

Fig.5-Memory bandwidth diagrams for $\lambda = .125$, $\alpha = 1/n$.

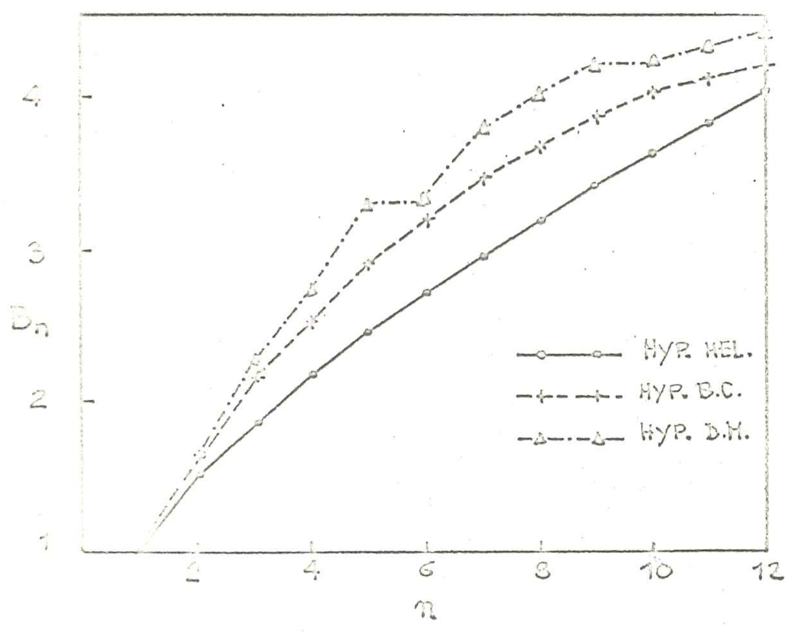

Fig.6-Memory bandwidth diagrams for $\lambda = .125$, $\alpha = .5$.

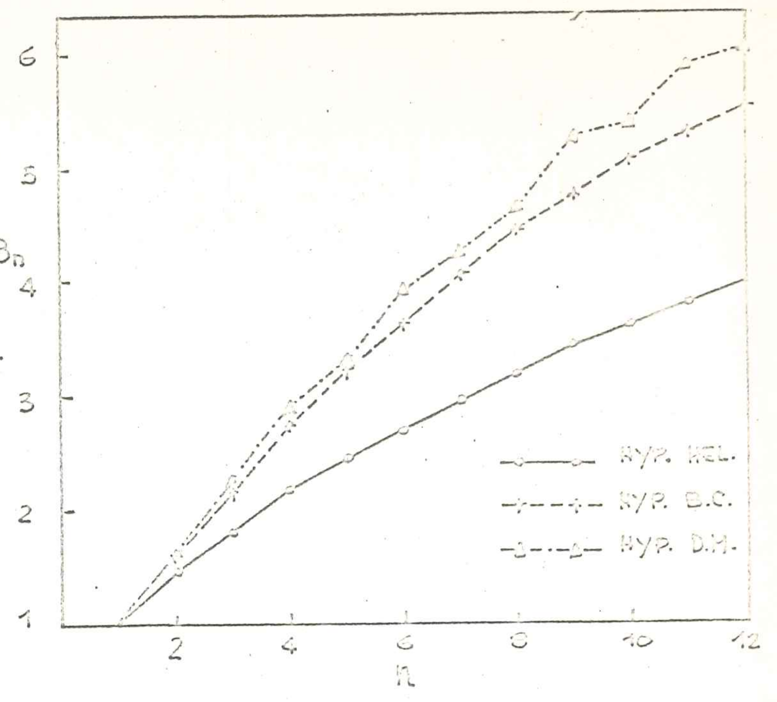

Fig.7-Memory bandwidth diagrams for $\lambda = .2$, $\alpha = 1/n$.

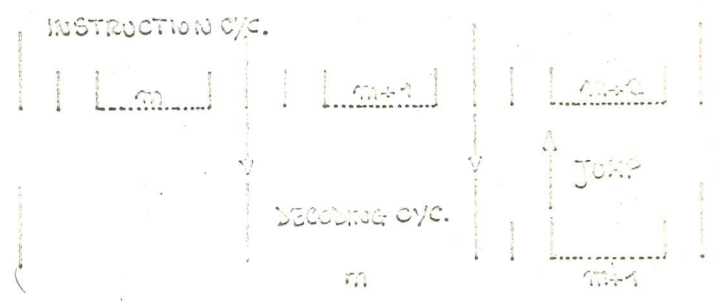

Fig.8-Memory accesses in the divided-memory model in the case of par.3.1.

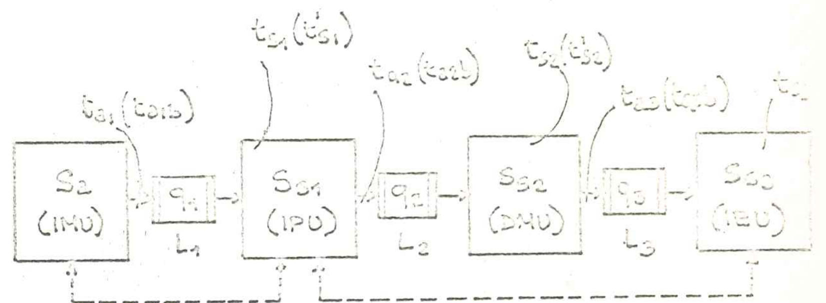

Fig.9-Simplified model of pipeline computer.

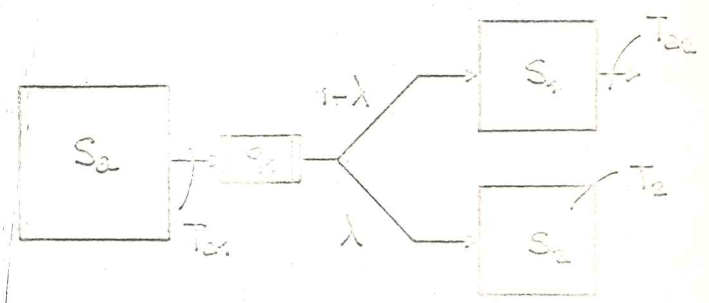

Fig.10-Schematization for evaluating the influence of jumps in a pipeline system.

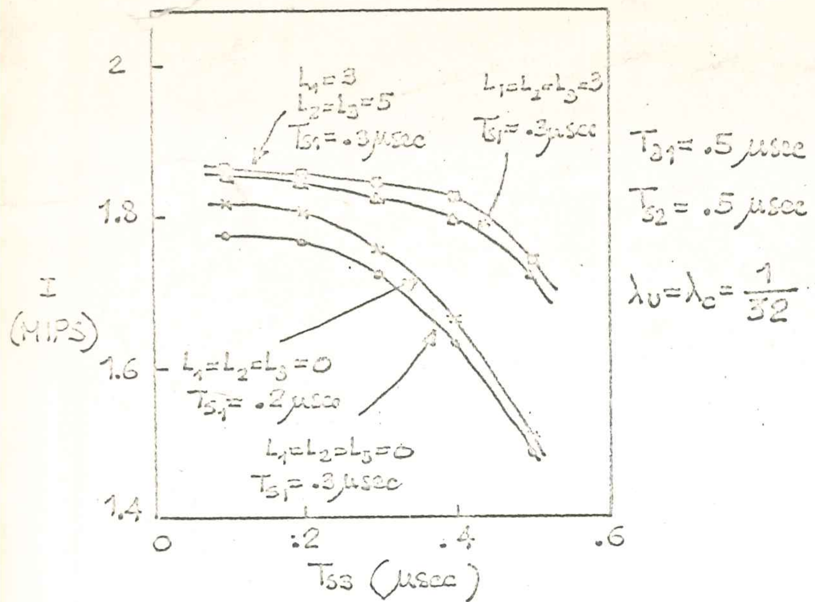

Fig.11-System performance diagram for λ=1/16

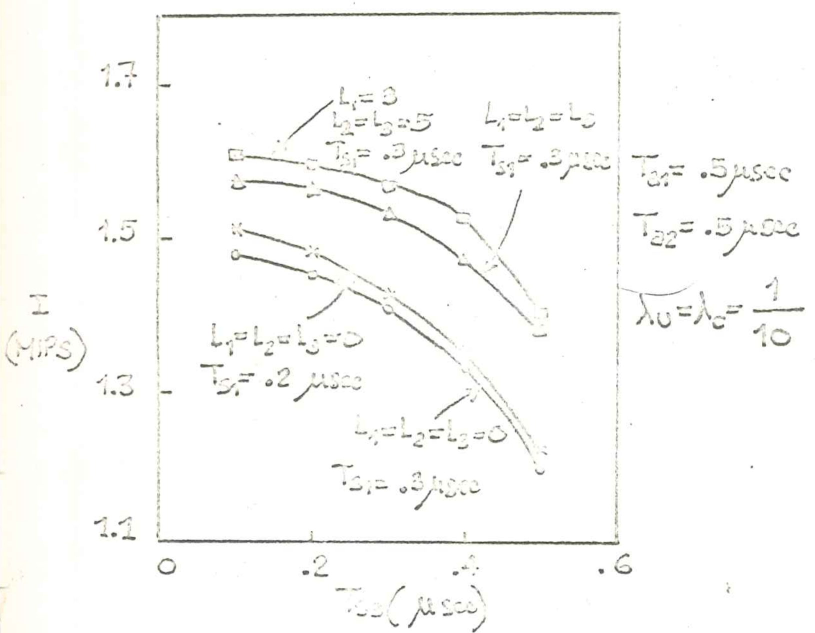

Fig.12-System performance diagram for λ=1/5

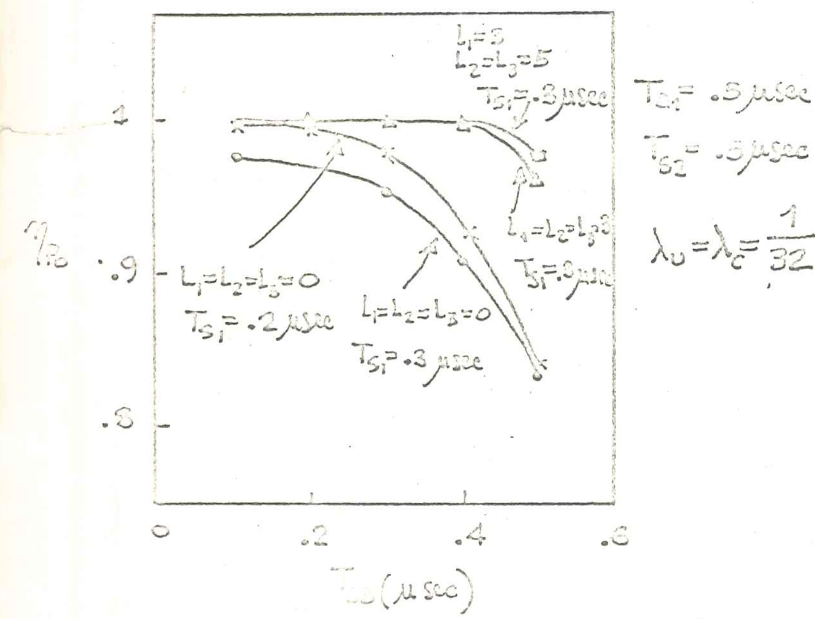

Fig.13-Efficiency figure of IMU stage as producer.

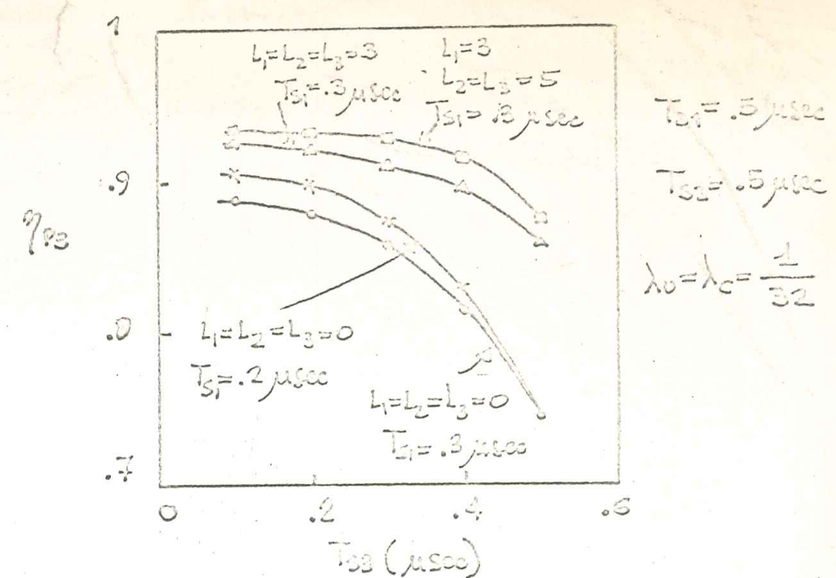

Fig.14-Efficiency figure of IMU stage as producer.

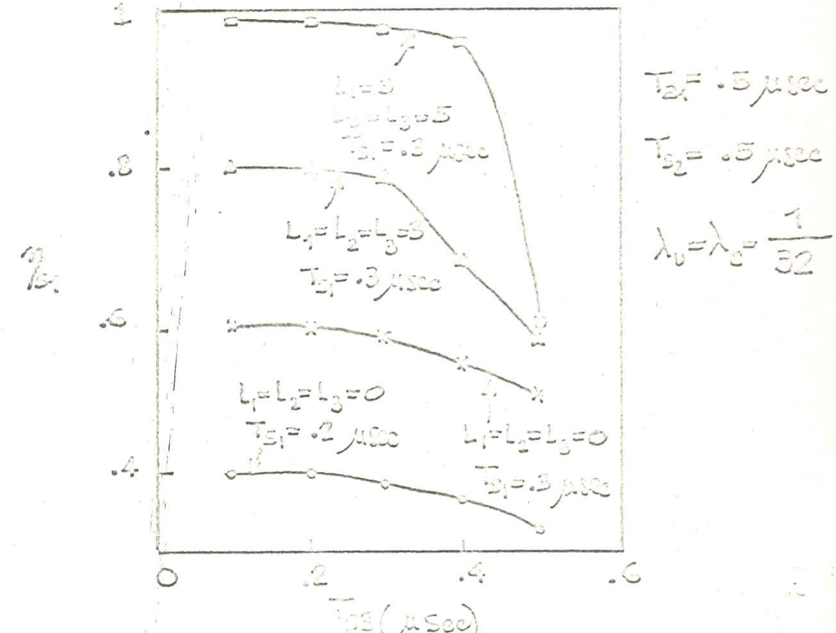

Fig.15-Service efficiency figure of IPU stage.

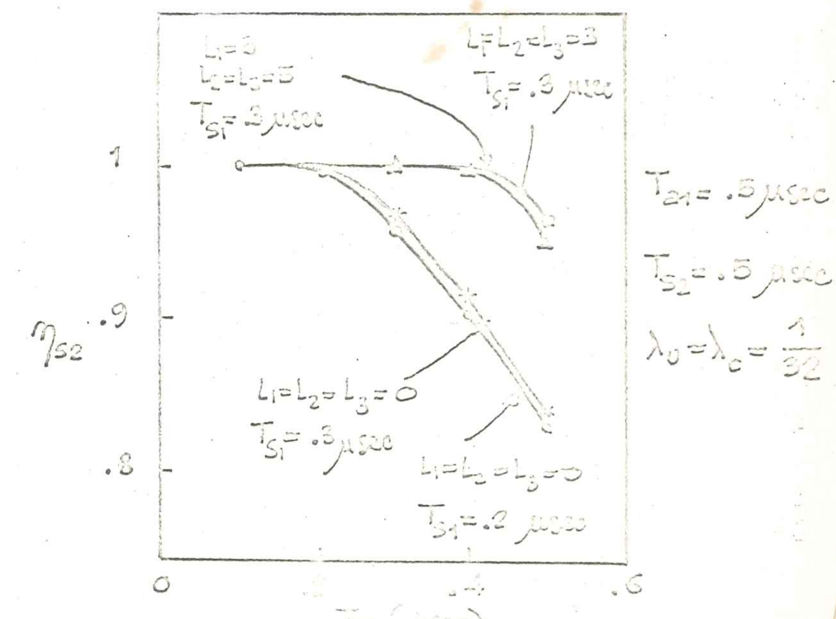

Fig.16-Service efficiency figure of DMU stage.