

A controller for computer internal communication network

Citation for published version (APA):

Sung, C. S. (1978). *A controller for computer internal communication network*. Technische Hogeschool Eindhoven.

Document status and date:

Published: 01/01/1978

Document Version:

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

TECHNISCHE HOGESCHOOL EINDHOVEN

Afdeling der Electrotechniek

Report on the project developed

by C. S. Sung in the period

November 77 till May 78.

NUFFIC adviser : Prof. Ir. A. HEETMAN

A CONTROLLER FOR COMPUTER INTERNAL
COMMUNICATION NETWORK

C. S. SUNG

This project was supervised
by Ir. M. P. J. STEVENS

May 1978

INDEX

ABSTRACT

1. INTRODUCTION
2. COMMUNICATION NETWORK
 - a. General
 - b. Functional blocks
3. MATRIX CONTROLLER
 - a. Specifications
 - b. Solutions
4. DESCRIPTION OF FUNCTIONAL BLOCKS
 - a. Line Circuit Interface
 - b. Console
 - c. Switching Matrix Interface
5. CPU DESIGN : HARDWARE
 - a. Microprocessor
 - b. Microprogram Controller
 - c. Pipeline
 - d. Clock
 - e. Internal Bus
 - f. Memory
 - g. Microprogram Store
 - h. Interrupt Control Unit
 - i. Lock/Key
 - j. Condition Selector
 - k. Mapping PROM and Vector Address
 - l. Watch Dog Timer
 - m. D-Register
6. CPU DESIGN : SOFTWARE
 - a. General System
 - b. Instruction set
 - c. Memory Organization
 - d. Microinstruction
 - e. Microprogram Store Organization
 - f. Internal Registers
7. MICROPROGRAMS

8. TESTING

9. APPENDICES

a. Microprogram Controller emulation

b. Connectors

c. Signal timing

d. Instruction codes

e. FPLA

ABSTRACT

This report deals with a fast controller for a computer internal communication network. This network should provide reliable communication among the independent processes within the computer. The controller, based on bit-slice microprocessors, besides dictating the actions to be taken by the network, must also look after the safety of the system. External error sources as well as internal misfunctionings have to be detected. The time delay to serve a request for a connection between two processes deeply affects the total computer performance, and it is supposed to be minimized.

1. INTRODUCTION

As the performance of the new integrated circuits improves ever more, it increases the advantage of spreading processing power in a number of intelligent nodes performing diverse tasks in parallel.

On the other hand, the growing volume of information to be exchanged among the nodes needs a sort of communication network to link them.

10 This collection of nodes together with its interconnecting channels can be regarded as one unique system, though complex it may be. Thus, it may be considered as one computer built up from a number of functional units which interact with one another. The performance of such a system is dependent not only upon the power
15 of each of its constituent nodes, but also upon the speed and reliability of the internal data communication between them.

It was decided at the THE in the group of Digital Techniques, to mount such an internal communication network in order to connect the various microcomputers in use and the
20 different peripherals available, so that any device has access to all the others. The solution adopted was a duplex switching matrix to which all the devices are connected like subscribers to a telephone exchange.

The design of a fast controller for this switching matrix
25 was the object of an effort contrived from November '77 to May '78. Due to the restricted time available, just part of the controller has been developed up to this date and it is exposed in this paper.

2. COMMUNICATION NETWORK (Fig. 1)

a. General

An information exchange involves not only data but commands as well. Separation between both bit streams may be in time, in which case we need insert some overhead to make the distinction. For speed's sake, spatial separation was chosen, implying in the split of the Communication Network (CNET) in two networks, one for commands and the second for data transmission. As a consequence, an extra flexibility was gained, namely the possibility of exchanging data with a partner, and, simultaneously, commands with another partner.

This split brings along an improvement in the system's reliability. A collapse in one network may result in just a decrease in performance without hindering the working of the whole. This can be obtained by using the surviving network for both commands and data transmission using a temporal division. For sure, the devices should have the capacity to recognize and deal with such a situation. Since physically both networks are similar, the difference being merely in the kind of information flowing through them, they can be equally used should a single network breakdown occur.

For one network to survive independently of the other, it is required that their controls be independent, i.e., one controller for each network. The main units in a reliable system should always be duplicated, and having only one controller would expose the system to the risk of complete paralization in case of a software error.

A major factor affecting the efficiency of the CNET is the speed with which it can process the messages sent by the devices and perform the convenient actions. Roughly it can be stated that the delay for the establishment of a connection ought not to be larger than the average connection duration time.

The man-machine interface is another task to be tended for by the CNET, allowing the operator to obtain information on the

b. Functional blocks (Fig. 2)

A block diagram of the CNET shows a clear symmetry between the Command and the Data networks. Each of them consists of 3 types of functional units: the Line Circuit (LC), the Switching Matrix (SM), and the Matrix Controller (MC). A Console shared by both halves completes the system.

Each port to the CNET possesses a LC. It is supposed to detect requests from the device, which are then signalled to the MC. Under the control of the MC, it can establish a communication between device and MC for message exchange or set a link between device and the SM. Here takes place the conversion from the 4-wire full-duplex channel with current-mode signalling to the internal signalling used in the SM and vice-versa. The LC decouples electrically the channel and the internal circuitry, for safety purposes.

Furthermore, the LC participates in the testing and disconnection of the matrix crosspoints.

The LC here proposed differs from the present version (developped for a single MC system), in that it cares only about one channel, either the Data or the Command channel, but not both. Requests for the Command network and for the Data network are sent through the respective channels, not all through the Command channel as in the present version. Thus, complete independence exists between both parts of the CNET.

The SM is a 32 x 32 full-duplex matrix which crosspoints are thyristors. Connections are accomplished by activating pairs of thyristors, while disconnection occurs when the pair of LC's associated to that particular crosspoint is commanded to disactivate its thyristors. Interchannel crosstalk poses not much of a problem since digital data tolerates a low S/N ratio without significant effects on the bit error rate.

Requests detected by the LC are served by the MC according to a priority scheme and each request is processed and all the appropriate measures are taken before the next one is served. Eventual uncoherences and errors found out during this processing

are signalled to the Console, and attempts to correct them are made. For statistics upkeep, the MC informs the Console about all requests received and the result of the actions performed.

Connections are laid across the SM's under the control of the MC which accionates the desired crosspoint establishing a 4-wire full-duplex link between a pair of ports. There is also the possibility of testing the crosspoint by sending from the MC, via one of the LC's associated to that crosspoint, a bit pattern which goes through the SM and is detected by the other LC associated to that crosspoint. This LC conveys the received bits back to the MC which then verifies the integrity of the test pattern.

Network administration, fault detection and diagnosis constitute the main tasks of the MC.

The operator-CNET interaction is provided by the Console. It monitors the system's performance thanks to a constantly updated information received from the MC concerning the network status, the start and the end of the connections, the faults detected. Besides, it transmits to the MC any operator's command demanding more information on the present network status or ordering modifications in it.

Being under a relatively light load, the Console can store the story of the CNET for further processing, aiming at statistical analysis which can help optimize the network utilization and also help discover the source of sporadic faults happening only under special or random conditions. Furthermore, preventive maintenance can also be scheduled by the Console since it can keep track of the story of the network components.

3. MATRIX CONTROLLER

a. Specifications

The 'intelligent' block of the CNET, the Matrix Controller (MC), has to satisfy a number of functional as well as safety requirements.

- . Initializing the system, testing the network and updating the internal network maps.

- . Periodic polling of the LC's to serve device requests according to a priority scheme. The selected LC is then supposed to put MC and the requesting device in communication.

- . Reception and processing of the message from the device, to check its validity.

- . If valid and possible to be attended to, the desired action is carried out and eventually a confirming reply is transmitted back to the device. Otherwise, a convenient advise is sent back.

- . If the desired partner is found already engaged in a conversation, there must exist the possibility to request a waiting connection, so that the link will be established as soon as the other device becomes available.

- . Updating of the network maps and status.

- . Testing, fault detection and correction as far as possible. The Switching Matrix crosspoints are to be tested; the integrity of the network must be protected from software errors in the MC, and if the network system goes astray, causing an excessive delay in the polling periodicity, this should be detected and self-recovery procedures take place.

- . Communication with Console to inform about the occurrences and to perform commands from the Console.

- . Waiting time for a device to have its request served should be minimum.

- . Interfaces within the functional blocks of the CNET were quite loosely defined and no requirements were set for the internal architecture of the MC.

. No demands were put forward as to the type of printed circuit boards and components to be employed. Since it is not a commercial system, no preferential component list or bus standard were imposed. Nevertheless, some practical limitations, as the excessive delay for the acquisition of certain components and a sensible commitment to the standard digital logic, did restrain the choice.

b. Solutions

To speed up the request serving, we distribute the functions.

A Line Circuit Interface (LCI) takes care of scanning the LC's and assembling the messages arriving in serial mode into the parallel format, checking the code and eventually asking for retransmission, thus saving CPU time. In case of replies to be sent to the devices, the LCI receives them from the CPU in parallel and transmits them serially, generating the redundancy bits and, if necessary, retransmitting the message. Matrix crosspoint testing and disconnection commands are also channeled through the LCI.

Message processing and map updating are assigned to a dedicated processor (CPU) with an architecture specially adapted to its functions. Frequently accessed data is stored in a fast semiconductor memory while less often used data and programs are in a slower section of the RAM memory.

Implementation of the 'waiting connection' feature allowing formation of waiting queues, was limited to only one waiting partner a for every engaged device b. Should a third device also try to communicate with b, the MC will reply with a 'device b busy' message. When the device b finishes its conversation, immediately a gets a connection with b, at the same time freeing the waiting queue for b. Unlimited queueing (up to 30 devices might wait to talk with a certain partner) would require large queue maps, and time-consuming updating and searching. There is also a risk that a low priority device might stay indefinitely in queue.

The CPU instruction set is tailored for the particular tasks most often executed but also includes standard general-purpose instructions to enable programming of less critical functions. This feature requires the possibility to microprogram, so that powerful instructions, implemented through special microprograms, deal with the critical functions. They are faster than the alternative of programming the tasks with standard instructions, which are not optimized for the specific goal and require more time since they need more accesses to the program memory to fetch the instructions.

Error detection and diagnosis are undertaken during the message processing by the CPU, which tries to verify the coherence between the memory maps and the message. For instance, a device requesting a new connection is supposed not to have a 'busy' status according to the map, but, should this not hold true, an attempt is carried out to find whether the device has committed a mistake or the map is incorrect.

In case a failure leads to improper behaviour of the CPU, a lock-key safety feature limits the access of a program to only certain parts of the system. Thus, eventually, havoc in the network or in the memory can be avoided. An attempt to violate a forbidden section causes a branch to a re-initialization procedure that tries to put the system back to the rails.

A Watch Dog Timer (WDT) looks after that the LCI is polled periodically. An excessive delay probably means a software error due to some particular conditions, that resulted in undefined loops. When too large a span of time elapses since the last polling, the Console gets a warning and then it ought to command the MC to enter a reinitialization procedure, and it is hoped that the special circumstances leading to that error are not going to repeat. Afterwards, the Console may try to diagnose the error.

Due to the loose definition of the interfaces between the constituent blocks of the CNET, it was thought better to leave the LCI and the Console for a further development, when the Console functions and the LC are clearly settled, lest eventual modifications in these definitions invalidate the design made.

Nevertheless, for the design of the CPU, many assumptions were taken about the LCI and the Console, and they are presented in the next chapter.

4. DESCRIPTION OF FUNCTIONAL BLOCKS

The CPU was developed based upon certain premises on the characteristics of the Line Circuit Interface, Console and Switching Matrix Interface. Due to the lack of time, these blocks were left aside untackled. For sure, they should be connectable to the Internal Bus, described in the following chapter.

a. Line Circuit Interface (LCI) (Fig. 4)

On one side, it interfaces with the CPU and on the other, with up to 32 LC's, which are under its command.

The LCI is connected to the Internal Bus as a peripheral with fixed address 0 (established for ease of programming). It communicates only with the CPU, and since it is often contacted by the CPU, it should have fast buffers for this message exchange. Any outgoing data, either a device message or a LCI message, is pushed into the outgoing buffer. It works as a stack that pops one 12-bit word, writing it on the Internal Bus lines, every time the LCI is polled. Polling occurs when the CPU addresses the LCI in a read operation. Commands issued by the CPU to the LCI are sent on the Internal Bus by means of a write operation and they should be stacked into the incoming buffer. Speed requirements for these buffers are given in Appendix c.

The LCI is a finite-state machine initially in the 'idle' state, ignoring any requests from the LC's.

A normal sequence of events for the LCI is started by the arrival of a 'Scan' command from the CPU, making it go to the 'scan' state. Then, it has to scan the request lines of the LC's which correspond to unmasked bits in a Priority Mask. Masked LC's are not served.

After a device is chosen to be served, the LCI selects the corresponding LC (which is supposed to lay a two-way connection between device and the LCI as long as it is selected), acknowledges the request and expects the device to send the message, which is serial-to-parallel converted and stored in the outgoing buffer.

The LCI holds the LC selected while it waits for the CPU to poll, upon what the LCI pops the data of the outgoing buffer onto the internal data bus.

Now, one of four possible commands should come from the CPU:

- . 'Break Xpoint' - the LCI must order the pair of LC's specified in the command to break the crosspoint. After this, LCI goes to the 'idle' state.
- . 'Send message to device' - in which case the message is supposed to come in the next word sent on the data bus, and it will be transmitted after parallel-to-serial conversion with redundancy code generation is performed by the LCI. Then LCI goes to the 'idle' state.
- . 'Test' - LCI must test crosspoint specified by the pair of addresses in the second word of the command and give the result with either a 'Test OK' or a 'Test failed' message, before going to 'idle' state.
- . 'Stop' - and LCI returns to 'idle' state.

After fulfilling the tasks, the LCI always liberates the selected LC, before going 'idle'.

Should a polling come when no requests at all have been detected or before a message has been completely received, then the LCI answers likewise by writing the contents of the outgoing buffer on the internal data bus, but, since its contents are always cleared after the last message is transmitted, this is understood by the CPU as 'No message'.

The CPU has at its disposal two other commands,

- . 'Connection to device a' - only accepted by LCI when 'idle'. Then a is selected and the message contained in the second word of the command is forwarded to this device
- . 'New Priority Mask' - only accepted by an 'idle' LCI, the next words supply the new Priority Mask.

The state diagram proposed in fig. 4 shows the allowed combinations, and it would be safe to have the LCI ask for interruption in case a violation of the permitted sequences is detected. This interruption should start a CPU procedure to

investigate the source of the error.

When its corresponding LC is selected, a device with physical address a may send to the MC one of the following messages:

- . 'Connect a to b' - where b is the logical address of the desired partner.
- . 'Disconnect a and b' - which may mean that either there is a connection between both which must now be undone, or there is a waiting connection requested by a (and the desired partner b was engaged in a connection) but now a does not want to wait any longer.
- . 'Inoperative' - device a is going out of operation and it is disabled for any connection from now on. Any remaining connection is broken.
- . 'Operative' - device a is operating normally now.
- . 'LOG = a' - answer to the command 'Your logical address?' from the MC.

When the LCI receives a device message, it must add the physical address a of the requesting device, which is an information needed by the CPU. Thus, the internal format of a device message consists of two words, the first for the message code and the second for the data - physical address a and logical address b, which are read by the CPU in two consecutive read operations.

Besides, there are also messages originated in the LCI itself to the CPU. Two of them concern the result of a crosspoint test. The test consists of ordering the pair of LC's associated to that crosspoint to go into the 'test' state, in which they provide a duplex link from LCI to the Switching Matrix. So, the test pattern generated by the LCI goes, via one LC, across the SM and, via the other LC, it is received by the LCI, which then checks its integrity. Simultaneously, the inverse path is tested.

The LCI may send to the CPU:

- . 'Test OK' - correct 2-way communication through the crosspoint.
 - . 'Test failed' - at least one of the ways does not operate properly.
-

- . 'No message' - when the outgoing buffer contains no message.
- . 'LC a crazy' - a 2-word message, with the LC address a in the second one. It signals something wrong with that LC.

By means of a 'Send message to device' or a 'Connection to device a' command, the MC may send to the selected device the following messages:

- . 'You are inoperative' - tells the device that it is disabled to receive or request connections. It is used as an acknowledgement to an 'Inoperative' message from the device or as a notice that due to a command from the Console, it is being put inoperative now. While in this state, it cannot send a 'Connect a to b' message, though its requests continue being served.
- . 'You are operative' - now it is enabled to receive and request connections. It is used as an acknowledgement to an 'Operative' message from the device.
- . 'Xpoint defective' - answer to 'Connect a to b' message from the device, informing a that the tests made on the cross-point (a,b) have failed and it should temporarily quit trying to contact b directly.
- . 'Device b inoperative' - reply to a 'Connect a to b' message in case b is inoperative.
- . 'Device b busy' - same as above, only that now b is engaged in a connection, and besides, there is already one device waiting to talk with b. So a should repeat later the attempt to contact b.
- . 'Connections disabled' - also a reply to 'Connect a to b', when the MC is disabled to lay any new connection across the network.
- . 'You are crazy' - reply to any device message that is deemed by the MC to be improper for the present situation.
- . 'Your logical address?' - question issued by the MC to update its internal maps.

These sets of messages above exposed are considered sufficient to deal with all the situations. In case of error, either due to an invalid command code or an incorrect sequence of commands, the LCI resorts to an interrupt request as a means to

warn the CPU.

A proposed set of codes for the messages is exposed in the Appendix d.

b. Console

It is connected to the Internal Bus as a peripheral with fixed address F_{16} (for ease of programming). Like the LCI it must be equipped with fast stack buffers to talk with the MC.

Console commands are polled by the MC in read operations and information about the network arrives in write operations, as well as by "bugging" the Internal Bus to overhear the messages sent by the LCI. Should something go awry and the Console not be polled after the due interval, it can resort to an interrupt request to force the acceptance of its command.

A tentative definition of the Console assigns to it two main statuses, dictated by some key on the Console panel: Disabled and Enabled.

- Disabled - the Console just receives information about the CNET, not being allowed to interfere. All messages issued by the LCI are to be received by the Console, which detects when the LCI is being polled by the CPU, and reads the message written by the LCI on the internal data bus.

Besides, the MC transmits in write operations to the Console, the results of the message processing - actions taken and eventual errors found. All the data collected can be processed by the Console to analyze the CNET traffic to optimize its performance. Failure may be diagnosed by studying the events' sequence preceding the error.

The set of one-word messages issued by the MC comprises:

- . 'Initializing' - sent at the start of the Initialization program to advise Console that the Watch Dog Timer must be ignored till the system enters the Network Operation program.
 - . 'Reinitializing' - same as above, only that it is sent at the start of the Reinitialization program.
-

- . 'Disconnection failed' - the test made on the crosspoint shows that the disconnection command was not successfully carried out.
- . 'Connection failed' - idem, but now the connection command was unsuccessful.
- . 'Waiting connection' - the last connection request is on the waiting queue.
- . 'Send Status Word' - when (Re)Initialization reaches the last stage, Console is requested to forward a new Status Word. At the same time, it is informed that after satisfying the request, the system is ready to go.
- . 'LCI crazy' - results of the test on the crosspoint, performed by the LCI, are neither OK nor failed.
- . 'Check LCI' - an interrupt request from the LCI implies that it has detected some fault.
- . 'Check SM' - an interrupt request from the SMI implies that some fault was found in the Switching Matrix.
- . 'Error messages' - bits 7,8,9,10 of the message indicate the number of the error found during a message processing.

See Appendix d for all the proposed codes.

When polled by the MC, the Console in Disabled status answers with a 'No commands' message.

- Enabled - Aside from performing the tasks described in the Disabled status, the Console is able to issue commands to the MC, which are stacked into the outgoing buffer. The proposed Console command set consists of:

- . 'Load Memory' - Console wishes to load new data into the MC Memory.
 - . 'Dump Memory' - Console asks for the contents of parts of the MC Memory. Useful in case of a thorough diagnosis of a system failure.
 - . 'Dump registers' - same as above, now the MC internal registers are dumped.
 - . 'Go to n' - MC must start executing instruction in Memory address n.
 - . 'New Priority Mask' - A new Priority Mask for the LCI is
-

given.

- . 'New Status Word' - MC must adopt this new Status Word.
- . 'Device a off' - order for the MC to consider the device a as inoperative. Same effect as a device message 'Inoperative'
- . 'Device a on' - orders the MC to put device a operative.
- . 'No commands' - there are commands presently.

In the Appendix d, codes for the commands above are proposed.

c. Switching Matrix Interface (SMI)

The SMI hangs onto the Internal Bus as a peripheral with a fixed address 1 (for ease of addressing it in a microprogram).

Under the command of the CPU, the SMI takes the measures necessary to lay the desired connections across the Switching Matrix. The commands arrive in CPU write operations in which the data sent on the data bus consists of a pair of 5-bit addresses specifying the two ports to be linked. Appendix d describes the command format.

Aside from this function, we propose that it should request an interruption to the CPU in case any problem happens to be detected, as for example, the absence of the Switching Matrix.

5. CPU DESIGN : HARDWARE (Fig. 4)

A microprocessor-based design was immediately assumed, in order to provide the processing power required.

a. Microprocessor (Fig. 5)

Three microprocessors were taken in consideration during the selection phase.

10 Departing from the requirement of microprogrammability, the range of choice became restricted to two types of bit-slice microprocessors easily available: Advanced Micro Devices Am 2901 (4-bit slice) and Intel 3002 (2-bit slice).

15 In our application, the dedicated CPU has few arithmetical tasks, but on the other side, much map accessing, and message reception and transmission must be performed. Mainly, it is desired to manipulate bits. The Am 2901 has a more powerful arithmetical capability than the the 3002, while the latter has 3 inputs and 2 outputs as compared to only 1 input and 1 output for the Am 2901.
20 Due to the stress on the demand for high throughput, the Intel chip offers a substantial advantage over the other. Address and data buses need not be multiplexed as it would happen with the Am 2901, saving microinstruction cycles in read/write operations. Besides, one of the inputs to the 3002 serves as a mask, facilitating bit
25 manipulation.

The microinstruction execution time is about the same for both chips. The Am 2901 has 16 internal registers, but the 11 registers at disposal in the 3002 were deemed about enough for our purpose.

30 The third microprocessor considered in the study was the Signetics 8X300, a device with high speed - 250 ns instruction cycle time - , designed for high throughput and endowed with good bit handling facilities. Though not microprogrammable, its instruction set and speed suit the necessities of the application.

The 8X300 has a 8-bit wide data bus. Since up to 32 devices may be connected to the CNET, each requires a 5-bit address. As often enough a device message involves a pair of partners, 10 bits are needed to specify both of them. Thus, two 8-bit data words
5 would have to be fetched, causing a loss of processing speed. Furthermore, the need for more data words goes against the reduced number - eight - of internal registers making it certain that we would run short of registers during a device message processing. Since part of the data employed cannot be stored internally, more
10 accesses to external memory would take place.

Adding the points, preference fell upon the Intel 3002. The elected data word length is 12 bits, with room for two 6-bit addresses, in case of CNET expansion up to 64 devices. Since data and instruction are not separate, also the same length applies for
15 instruction. An array of six Intel 3002's in parallel, accompanied by a carry look-ahead generator, the Intel 3003, constitute the core of the CPU. As it can be seen in the Appendix d, this word length is about the ideal to accommodate the proposed instruction set.

20 The 12 D-outputs of the 3002 array were connected to the 12 D-BUS lines. The 11 least significant A-outputs of the array traverse an Address Logic before they are entitled to control the 11 A-BUS lines. The most significant A-output is left unused.

Normally the Address Logic just inverts the A-outputs
25 before writing them on the corresponding A-BUS lines. However, when the Network Map is accessed, inversion occurs only for the 5 LSB, while the 5 MSB are forced LOW and the middle bit selects between Logical and Physical addresses, as explained further on.

The M-inputs to the 3002's associate one-to-one with the
30 outputs of the D-Register, i.e., with the last data word read into this register. Into the I-inputs, a mix of D-Register outputs and Mask bits from the microinstruction word come, serving special purposes described later. Finally, the K-inputs (used for masking) are connected to the Mask field of the microinstruction.

35 Bit testing possibilities with the 3002 require an external logic to AND the partial results from each of the bit-slice

processors, thus generating the CO signal which is HI if the bits tested resulted to be all LOW.

b. Microprogram Controller (Fig. 6)

To control the microprogram sequence, between the Intel 3001 and the Am 2910, the latter was reckoned better, due to its higher flexibility and ease of use. It was not available yet, but, using a pair of Am 2909's (microprogram sequencers), a PROM to decode the Am 2910 instruction code and a counter, the needed features of the Am 2910 were emulated. Four 74S08 buffers provide the capability to drive the Microprogram Store.

The limited microprogram address (only 9 bits), the reduced event counter capacity (only a 4-bit counter), the elimination of some microprogram sequence control instructions, simplified the complexity of the emulation circuit, and it was considered of little harm for our application. Compatibility remains, as soon as the Am 2910 takes its place, the old programs need suffer no modification, and then, its extra features can be used in the new microprograms.

Appendix a details the emulation circuit.

c. Pipeline (Fig. 8)

The pipeline consists of a bank of registers that, at the start of every machine cycle, store the microinstruction to be executed. It possibilitates overlapping the execution of a microinstruction with the fetch of the next one, since the Microprogram Controller may already calculate the next address and access the Microprogram Store while the rest of the machine is still executing the present microinstruction. Thus, the following one will be ready at the input of the Pipeline soon enough.

With a pipelined configuration, shorter machine cycles can be obtained.

d. Clock (Fig. 10)

It consists of a counter, and a NAND buffer that supplies

the driving capability needed for the CK signal. The A0 signal also traverses a NAND buffer to ensure a better synchronism between both clock phases.

There two clock cycles : a slow one (450 ns), in case the choice of the next microinstruction depends on conditions generated by the one present presently under execution, and a fast cycle (300 ns), when we do not need to wait for conditions being now calculated in order to determine the next microprogram address. Additionally, the fast and slow cycles are used to provide the necessary timing for Memory, LCI, SMI and Console read/write operations. These blocks have a definite specification concerning read and write cycle times, so a determined sequence of fast and slow cycles applies for every contact the CPU wishes to establish with them. In Appendix c, the timing diagrams used for the calculation of the cycle times and read/write operations are detailed.

Selection of fast/slow machine cycle is governed by the F bit in the microinstruction. The heavy load driven by the CK signal required the use of a buffer.

Nominal oscillator period is 150 ns, corresponding to 6.66 MHz.

e. Internal Bus

For internal communication, it was adopted a bus structure composed of three parts:

- A-BUS : 11 address lines commanded by the Address Logic of the CPU. Since no more than 16 peripherals are considered necessary, only the 4 LSB of the A-BUS contain a valid information when addressing peripherals.
- D-BUS : 12 bidirectional data lines in inverted logic, onto which tri-state logic outputs are connected.
- C-BUS : there are 9 control lines,
 - . ~~A-BUS VALID~~ - when the CPU writes LOW on this line, a valid address is present on the A-BUS.
 - . D-OUT CK - in a CPU write operation, the D-BUS

contents should be stored by the receiving end at the positive-going edge of this signal.

- . D-IN CK - in a CPU read operation, D-BUS is read into the D-Register at the positive-going edge of this signal. Also used by the Console to overhear the LCI polling.
- . P/M - when HI, it selects a peripheral, implying that only the 4 LSb of the A-BUS are meaningful. When LOW, memory is chosen, and all 11 A-BUS lines contain information.
- . W/R - a write operation occurs when it is HI, and a read operation when LOW.
- . INTREQ - the Interrupt Control Unit sets it LOW if an interrupt request with higher priority than the present interrupt level is present.
- . ACK - for peripherals with undetermined access delay this line is used to signal back to the CPU when they are ready.
- . WDT - when Watch Dog Timer detects a long time without LCI polling, WDT goes LOW.
- . LAL - when power is turned ON or the Reset button is pressed, LAL goes LOW. It is a initialization signal for all units.

The first three control signal are inhibited if an incorrect read/write attempt happens. The logics for the generation of these signals is concentrated on a FPLA. See Appendix e.

f. Memory (Fig. 9)

Constantly demanded data and programs use a fast semiconductor memory, going from Memory address 0 to 255, while a slower, and hence cheaper, semiconductor memory stores seldom accessed information, starting from address 256 onwards. It is expected that in normal operating mode the CPU will utilize but the fast portion. Initialization and Reinitialization may fit into this section, but diagnosis and special-purpose procedures will occupy

the other portion.

With 11 A-BUS lines, up to 2K words can be reached, using memory words of 12 bits with inverted outputs to the D-BUS line . For specifications concerning memory access and write cycle times, go to Appendix c.

g. Microprogram Store (Fig. 7)

All microprograms are supposed to fit into a 512 X 36-bit PROM, called the Microprogram Store. The word length of 36 bits comes from the size of each microinstruction, as explained in the next chapter.

It comprehends two sections of 256 36-bit words, which do not interfere with each other, i.e., there are no cross references from one section to the other. In the lower half, occupying positions 0 to 255, one finds the microprograms corresponding to the 38 standard instruction. The upper half, from 256 to 511, comprises the microprograms that treat the special instructions tailored for the message processing and switching functions. Though fewer in number, these instructions require lengthier execution.

As it concerns the speed, our timing calculations assumed a maximum access delay time of 70 ns for the PROM's. The next address comes from the Microprogram Controller early enough for the next microinstruction to be accessed in the Microprogram Store and to settle down at the inputs to the Pipeline. When another cycle starts, the Pipeline stores the data present at its inputs.

h. Interrupt Control Unit (ICU) (Fig. 11)

Based on an Intel 3214 chip, it serves up to 8 interrupt request lines. The CPU polls the ICU by a write operation, sending the contents of the Status Word. In fact, the ICU cares only about bits 0,1,2,6 of the D-BUS, where the present interrupt level and the interrupt enable bit are. In the next clock cycle, the INTREQ line is tested because only in this cycle it will be valid. Then, at anytime, the new interrupt level may be read on D-BUS lines 0,1,2 by means of a read operation.

$\overline{\text{INTREQ}}$ goes LOW just in case interruptions are enabled and an interrupt request with higher priority than the present interrupt level is present.

To address the ICU, we set in the microinstruction, NEXT ADDRESS field equal to C_{16} (Load Counter and Continue) and CCSEL non-zero. This method of addressing takes advantage of the unfrequent use of these fields , so that no extra machine cycles or microinstruction bits are specially dedicated to control the ICU.

For a read operation, $\overline{\text{D-REG}}$ is made LOW, while a write requires $\overline{\text{D-ENABLE}}$ to be LOW. The FPLA takes care of decoding the ICU read, making $\overline{\text{ICUR}}$ LOW. In a ICU write operation, the FPLA makes $\overline{\text{ICUW}}$ go LOW. See Appendix e for more explanations.

More details on the timing appear in Appendix c.

i. Lock/Key (Fig. 10)

Seven possible locks exist, each protecting determinate parts of the CPU from an eventual invalid access. Valid operations are tabulated below:

Lock	Network Map	Rest of Memory	LCI SMI	Application
1	R,W	-	R,W	Message processing
2	R	-	-	Network Map dump
3	R,W	-	-	Network Map check
4	-	R	-	Instruction fetch
5	R,W	R,W	R,W	God's programs
6	-	R,W	-	Non-switching program
7	R,W	R,W	-	Diagnosis programs

R = read allowed

W = write allowed

Peripherals may always be read and written.

The Lock Register updates its contents whenever the micro-

instruction contains NEXT ADDRESS equal to E_{16} (Continue) and CCSEL non-zero. Then the CCSEL field is taken as the new value for the lock. Some critical microprograms immediately load an appropriate lock to prevent mishaps. When there is an attempt to perform a disallowed operation, the operation is inhibited so that no damage occurs, the I/O ERROR bit goes HI forcing a jump to microprogram address xFF (the MSb is not altered), where Re-initialization procedures start.

A FPLA performs the logics to survey whether such a violation has been attempted, and it is detailed in Appendix e.

j. Condition Selector (Fig. 10)

It comprises a clocked register to prevent the asynchronous conditions from being admitted at improper moments, and a multiplexer, controlled by the CCSEL field in the microinstruction, to select the desired condition.

CCSEL	Condition	Origin
0	I/O ERROR	Lock/Key
1	CO	3002 array
2	WDT	Watch Dog Timer
3	ACK	Asynchronous peripherals
4	TRUE	(For unconditional branching)
5	-	-
6	EV.CNT	(Goes HI when Event Counter = 0)
7	<u>INTREQ</u>	Interrupt Control Unit

k. Mapping PROM and Vector Address (Fig. 6 and 10)

Every instruction arriving from the Memory via the D-BUS is clocked into the D-Register. The 8 MSb constituting the operation code of the instruction serve as an address to the Mapping PROM which should output the 9-bit address in the Microprogram Store where the corresponding microprogram starts. Messages, commands received from LCI and Console suffer the same decoding, thus being converted to a 9-bit address pointing to the beginning of the

corresponding procedure. Invalid codes cause a branch to a micro-routine that fetches the next instruction, so that they behave just like 'No Operation' instructions.

The 4 LSb in a standard instruction specify one of the internal registers, if it is a register addressing instruction. Otherwise, they are all LOW.

Register addressing instructions need first branch to the appropriate routine that loads the working register with the contents of the desired register and then, at the end, stores back the result. The addresses of these routines come from the Vector Address, which in fact consists only of a 74125 tri-state buffer, since the addresses are directly derived from the 4 LSb of an instruction code, and from the EV.CNT signal which should be LOW at the beginning of the instruction and HI at the end.

l. Watch Dog Timer (Fig. 11)

Whenever the LCI undergoes a polling from the CPU, a pulse retriggers the Watch Dog Timer. It is basically a monostable multivibrator with a time setting around 50 to 100 μ s. The polling interval, during normal operation of the system, must not take longer than this setting or the WDT signal goes LOW and the Console is warned that the CPU has something wrong. By means of an interrupt request, the Console should command the CPU to re-initialize.

m. D-Register (Fig. 10)

In every read operation, the data present on the D-BUS is clocked into the D-Register by the positive-going edge of the D-IN CK signal. We can also load this register with a word written on the D-BUS by the 3002 array, in order to use it either as input to the Vector Address (when a branch to a register addressing routine is desired), or to shift the 5 MSb of the word into the 5 MSb position (due to the way the D-Register outputs connect with the I-inputs of the 3002 array).

6. CPU DESIGN : SOFTWARE

a. System (Fig. 14)

There exist four classes of programs: Initialization, Re-initialization, Network Operation and Off-Line programs.

. Initialization program (Fig. 15 and 16)

When power is turned on, or the Reset button is pressed, the CPU is forced to execute the Initialization procedures starting at position 0 in the Microprogram Store. This base microprogram fetches from the Memory the first instruction of the Initialization program. The Console may also trigger execution of this program by an appropriate 'Go to n' command.

The Initialization program, first sends the Console an 'Initializing' message and then polls the 3 highest priority interrupt lines, which should normally be HI. A LOW indicates a major error is being signalled by the unit associated to that interrupt line, which may be the Console, the LCI or the SMI. In this case, the Console receives a warning and the MC waits till an ACK comes from the Console, meaning that it should try again.

After none of the three interrupt requests remains, then the SM is cleared, i.e., the MC orders the breaking of all crosspoints, to assure no old connections are left over.

Every device possesses a characteristic address, named the logical address (LOG), by which it is known to the other. Due to the device-CNET interface standardization, a device may use any port of the CNET, implying its physical address (PHY) in the network, given by the physical position of the port it uses, independes of its LOG.

So, next in the Initialization program, through every port a 'Your logical address?' message is transmitted. In case a device exists that uses that port, it must answer specifying its LOG with a 'LOG = a' message. If, after a certain delay, no reply comes, the MC supposes the port is not being used and it goes on to repeat

the procedure with the next port. With this information, the Network Map is initialized.

At this point, the Console is demanded by the MC to 'Send Status Word'. After meeting this requirement, the Initialization is completed and the MC stays in a Console polling loop, waiting for commands. For instance, modifications in the Network Map or in the Priority Mask can be ordered before a command to go to the Network Operation program finally allows the system to enter its normal operating mode. Any of the commands described in Chapter 4, may be issued by the Console.

. Reinitialization program (Fig. 17)

In case of I/O ERROR going HI, the microprogram control is handed over to the microinstruction at position OFF_{16} (for standard instructions attempting to violate the LOCK) or position $1FF_{16}$ (for special instruction conflicting with the LOCK). At these positions starts the reinitialization procedure which loads a proper LOCK and orders fetch and execution of the first instruction of the Reinitialization program.

Reinitialization may also be provoked by an appropriate 'Go to n' command from the Console.

Differing from the Initialization, here the SM is not cleared and neither is the Network Map updated. Supposedly no damage has been inflicted upon the network, and we let it continue working and ignoring that a software error in the MC has happened.

Any error message still stored in the MC is forwarded to the Console before the latter receives a 'Reinitializing' message. The three highest priority interrupts are checked, if any is LOW, the Console is warned and a wait for ACK loop is executed. When everything is cleared up, a 'Stop' command is issued to the LCI and a 'Send Status Word' message to the Console. After this last message is answered, Reinitialization ends and the system stays in a Console polling loop, expecting for a command.

. Network Operation program (Fig. 18)

The Network Operation program comprehends a few instructions

stored in the fast portion of the Memory. Basically, it is a loop that successively polls the LCI and the Console, looking for messages and commands to be processed. In normal operation, the system should stay in this loop, though it may temporarily deviate to some external procedures to serve some more complex Console commands. Only when explicitly commanded by the Console, or forced by special circumstances like I/O ERROR going HI and Reset button being pressed, it may leave the Network Operation program.

Since the performance of the CNET relies heavily on the efficiency of the Network Operation program, special instructions, corresponding to powerful dedicated microprograms, are implemented to optimize the speed.

During the LCI polling, either device messages or LCI messages may appear, since both types are stacked into the LCI outgoing buffer.

In normal operation, the valid messages that a requesting device with physical address a may forward to the LCI are:

- . 'Connect a to b'
- . 'Disconnect a and b'
- . 'Inoperative'
- . 'Operative'

and the messages that may be originated in the LCI are:

- . 'No message'
- . 'Test OK' occur after a test is performed on a
- . 'Test failed' Xpoint. So they don't appear in the
- . 'LC a crazy' 1st polling made by LCI MESS instr.

Different procedures are in charge of processing each message, confronting it against the Network Map to verify any logical discrepancy. Should it be the case, an appropriate error number is written in the Status Word, and at the end of the processing, the Console is informed. The Network Map always undergoes the necessary updatings.

The message processing may generate one of the following commands, issued by the CPU to the LCI:

- . 'Test'
 - . 'Break Xpoint'
-

. 'Send message to device', which is followed by the proper reply to the device:

- . 'You are inoperative'
- . 'You are operative'
- . 'Xpoint defective'
- . 'Device b inoperative'
- . 'Device b busy'
- . 'Connections disabled'
- . 'You are crazy'

Besides, the Console gets a warning on the outcome of the device message processing in case not all goes smoothly. Otherwise, no message is generated by the MC, and it is assumed that the operation was carried out without problem. Since the Console overhears the messages transmitted by the LCI when it is polled by the CPU, no need to inform about the operation or the devices involved. The possible warnings are:

- . 'Disconnection failed'
- . 'Connection failed'
- . 'Waiting connection'
- . 'LCI crazy'
- . 'Error message n', where $n = 1, 2, \dots, 16$

After the message processing is over, in the next instruction, the Console undergoes a polling and any of the nine possible Console commands may appear.

Aside from the 'Go to n' command, all the other commands lead to convenient procedures which, after performing their due tasks, finally provoke a return to the Network Operation program.

. Off-Line programs

Off-Line programs may include diagnosis and administration programs. Due to their non-critical character, they occupy the slower part of Memory, and execution only occurs through a 'Go to n' command from the Console, which should be aware that the network stays paralyzed as long as the Off-Line programs are running.

Diagnosis procedures might simulate various events and

analyze the behaviour of the system, in an attempt to debug it.

b. Instruction set

We can distinguish two classes of instructions: standard and special instructions.

. Standard instructions

They are meant for non-critical tasks as the Off-Line programs, and seldom used functions as the Initialization program.

The addressing modes available :

- . accumulator addressing (1- or 2-word instruction) - the internal register ACC is taken as the operand.
- . register addressing (1-word instruction) - one of ten internal registers is the operand, according to the value assumed by the 4 LSb of the instruction.
- . immediate addressing (2-word instruction) - the operand is in the second word of the instruction.
- . absolute addressing (2-word instruction) - if no indirection or indexation are indicated, the second word contains the operand address. Indirection is indicated by the MSb of the second word. Due to the dedicated purpose of the MC, only one-level deep indirection was considered sufficient.

Indexation is made possible by inserting the INDX instruction just before an absolute addressing instruction, which will then have an absolute indexed addressing. The index is given by the 6 LSb of the internal register specified in the INDX instruction.

This solution, though requiring one more instruction every time indexation is desired, enabled using any register as index. Besides, if we dedicated one bit in the operation code to signal indexation, it would be hard to fit all codes needed. For simplicity, the index takes only the 6 LSb of the addressed register, allowing a value ranging from 0 to 63, which is enough for our applications.

Post-indexation is the rule for an absolute indirect indexed addressing mode.

Mnemonics	Addr. mode	Operation	No. words	Comments
LOAD	R	$R' \leftarrow R$	1	Loads register R (or ACC) with contents of the specified operand.
	A	$ACC \leftarrow M$	2	
	I	$ACC \leftarrow I$	2	
STORE	A	$M \leftarrow ACC$	2	Stores ACC into operand.
INP, n	ACC	$ACC \leftarrow \#n$	1	Peripheral n is read and the data stored in the operand.
	A	$M \leftarrow \#n$	2	
OUTP, n	ACC	$\#n \leftarrow ACC$	1	Peripheral n is written with contents of the operand.
	A	$\#n \leftarrow M$	2	
XCH	R	$ACC \leftarrow R$	1	Exchanges contents of ACC and the operand.
	A	$ACC \leftarrow M$	2	
AND	R	$ACC \leftarrow ACC \odot R$	1	Performs logical AND, OR, Exclusive-OR between the specified operands.
IOR	A	$ACC \leftarrow ACC \oplus M$	2	
XOR	I	$ACC \leftarrow ACC \oplus I$	2	
ANDM	R	$ACC \leftarrow ACC \odot R$	2	Same as above, but operation affects only the positions corresponding to HI mask bits.
IORM				
XORM				
ADD	R	$ACC \leftarrow ACC + R$	1	Sum of the specified operands is stored in register ACC.
	A	$ACC \leftarrow ACC + M$	2	
	I	$ACC \leftarrow ACC + I$	2	
ISZ	R	$R \leftarrow R + 1$	1	Increments/decrements R, skips
DSZ	R	$R \leftarrow R - 1$	1	if result is zero.
CLEAR	R	$R \leftarrow 0$	1	Register R is cleared
SET	R	$R \leftarrow \text{all } 1\text{'s}$	1	Register R is set all HI
CMPL	R	$R \leftarrow \overline{R}$	1	Register R is complemented
RTR, n	R	$R \leftarrow R \text{ rot.}$	1	R is rotated right n positions
JSBC, c	A	$PC \leftarrow Ad$	2	Condit. branch to subroutine
JMPC, c	A	$PC \leftarrow Ad$	2	Condit. branch to address Ad
RETC, c	R	$PC \leftarrow Ret.$	1	Condit. return from subroutine
INDX	R	$Index \leftarrow R$	1	6 LSb of R taken as index.
LOCK, n	-	$LOCK \leftarrow n$	1	LOCK gets new value = n
HALT	-	$PC \leftarrow PC$	1	Halt till ACK = LOW comes

R, R' = one of ten internal registers - ACC, R1, ..., R9
 PC = Program Counter register
 M = operand in the Memory
 I = immediate operand
 Ad = address calculated, after indirection, indexation
 Ret = subroutine return address
 #n = peripheral n

In the Addr. mode column, ACC = accumulator addressing
 R = register addressing
 A = absolute addressing
 I = immediate addressing

The condition tested by the JMPc, JSBC, RETC instructions depends on the value of c:

c	condition
0	TRUE
1	ACC = 0
2	ACC \neq 0
3	ACC > 0
4	ACC < 0
5	ACK = LOW
6	WDT = LOW

When the condition is TRUE, branch occurs. Otherwise, next instruction is executed.

If a branch to subroutine takes place, Program Counter and Status Word are saved in the Stack, as the Stack Pointer increases by two. In a return from subroutine, Program Counter and Status Word are restored, Stack Pointer is decreased by two.

ANDM, IORM, XORM are meant to endow bit handling capability, because the second word of the instruction is the mask. The logical AND, OR, XOR operation is performed only at the bit positions corresponding to HI bits in the mask. Bit positions corresponding to LOW bits in the mask suffer no modification.

In RTR,n instruction, the number of positions rotated varies from 1 to 8. In LOCK,n, the value of n goes from 1 to 7, the existing lock values.

When using registers, care must be taken with those used for special purposes, as detailed further on.

See Appendix d for proposed instruction codes.

. Special instructions

Designed to optimize speed in the critical section of the system, namely the Network Operation program, the special instructions have very specific tasks.

Inspecting the Network Operation block diagram (fig. 18), two clearcut functions appear: LCI polling followed by the message processing, and then the Console polling with the command processing. Thus, two special instructions exist, each tackling one of these functions:

. LCI MESS - checks the Status Word to see whether scanning is allowed. If so, it polls the LCI and processes any of the four legal device messages or the 'No message' message. It orders the necessary actions and verifies coherence between Network Map and the message. It updates the Network Map, informs the Console about any misgoing. Then next instruction is executed, unless an interrupt request causes a branch to an interrupt subroutine.

. CONS COMM - Polls Console and carries out any valid command. Next instruction follows on, except if 'Go to n' was commanded. Also an interrupt request may deviate the program from the normal sequence.

Appendix d shows instruction codes.

The microprograms performing these special instructions are rather long but still they can execute the function faster than a program consisting of standard instructions.

c. Memory organization (Fig. 19)

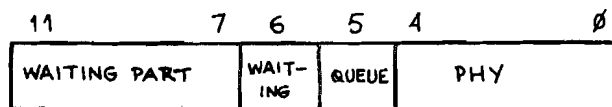
Within the fast portion of Memory (address 0 to 255), the following divisions exist:

Addresses 000 to 063 - Network Map

064 to 095 - Subroutine Stack
 096 to 103 - Interrupt Vectors 0 to 7
 104 - I/O Error Vector
 105 - Initial Vector
 106 to 255 - Initialization +
 Reinitialization programs

When a device requests a connection, it specifies the LOG address of the desired partner, but the MC needs the corresponding PHY address to know to which port the called partner is connected. Thus, a LOG-to-PHY conversion table occupies the second half of the Network Map, positions 032 to 063, corresponding to LOG addresses 00 to 31 respectively. Thus, by adding 32 to the LOG address we have the address into the LOG-to-PHY table. Besides the 5-bit PHY address, each word contains information about the present state of this device:

WAITING (1 bit) - if HI, this device has requested a connection that has not been completed, but it is on the waiting queue of the desired partner.
 QUEUE (1 bit) - if HI, there is a device in its waiting queue wishing to talk with it as soon as it becomes free.
 WAITING PART (5 bits) - if QUEUE = HI, this field contains the LOG address of the partner which is waiting in its queue.



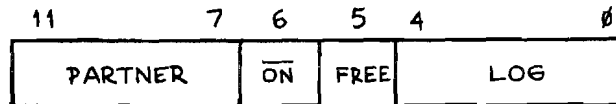
Conversely, when the MC knows PHY, a PHY-to-LOG table provides the inverse conversion. This table goes from position 0 to 31 of the Network Map and the PHY address is directly used as the entry point to this table. Aside from the LOG address, the word has some information on the device's present state:

\overline{ON} (1 bit) - if LOW, the device is operative. Otherwise, it is inoperative.

FREE (1 bit) - if HI, it is not engaged presently in any connection. If LOW, it is either already talking or it is

in the waiting queue to talk with someone.

PARTNER (5 bits) - if FREE = LOW, it contains the LOG address of its partner. In case this device is in the waiting queue of some other, this field has the LOG of the desired partner.



The Subroutine Stack permits up to 16 levels of subroutine nesting. The Stack Pointer (SP) in the CPU always points to the next free position. A subroutine branch saves in the first free word the Status Word and in the next cell the Program Counter (PC). Thus 2 stack words go for every level of nesting. A subroutine return restores the PC and the Status Word, the SP is decreased by two.

Each interrupt level has a corresponding Interrupt Vector, which content is the address of the appropriate interrupt subroutine. Similarly, the I/O Error Vector supplies the address of the Reinitialization program and the Initial Vector is the starting address of the Initialization program.

Initialization and Reinitialization programs are expected to fit within the allotted space. Otherwise, the slower Memory may be used.

d. Microinstruction

The length of a microinstruction may vary widely. Aside from the compulsorily fixed fields, some other may vary in length or may even be omitted. The larger the number of bits, the easier is the control over all the parts of the CPU. But, on the other hand, more memory goes to store the microprograms.

Only one microinstruction format was adopted, because of the impossibility to multiplex fields which are not simultaneously used.

Compulsory fields (always needed) :

- . FUNCTION (7 bits) - it is the instruction for the 3002's.
- . A-ENABLE (1 bit) - enables the A-outputs of the 3002's.
- . D-ENABLE (1 bit) - enables the D-outputs of the 3002's to write on the D-BUS.
- . CI (1 bit) - carry-in for the 3002's, connected to the CI-input of the least significant 3002.
- . NEXT ADDRESS (4 bits) - instruction for the Microprogram Controller.
- . MASK (5 bits) - connected to the K-inputs of the 3002's. Though 12 bits would endow more flexibility, a 5-bit MASK fulfills the needs.
- . F (1 bit) - dictates whether the clock cycle is slow (F = LOW) or fast (F = HI).
- . D-REG (1 bit) - enables loading the D-Register with the contents of the D-BUS.

Non-compulsory fields (needed in some microinstructions) :

- . PL ADDRESS (8 bits) - only necessary to specify branch address or the value to be loaded into the event counter.
- . CCSEL (3 bits) - required in conditional branching microinstructions to select the condition. Also used as the new value for LOCK in a load lock operation, and to address the ICU. Normally it must be all LOW.
- . MAP (1 bit) - normally LOW, it goes HI only when we address the Network Map, the LCI and the SMI. It zeroes the 5 MSb of the A-BUS and makes bit 5 follow the L/P bit of the microinstruction, in order to facilitate addressing the Map. It also indicates to the Lock/Key unit when LCI, SMI and Network Map are being addressed.
- . M (1 bit) - in read/write operations, together with the F bit, it determines the pattern for the C-BUS signal's involved in the operation.
- . L/P (1bit) - by controlling bit 5 of the A-BUS, it selects between the LOG address (L/P = HI) and the PHY address (L/P = LOW) when the Network Map is accessed. Also intended to facilitate accessing the Map. Normally it must stay LOW.

- P/M (1 bit) - for read/write operations, distinction is made between Memory (P/M = LOW) and a peripheral (P/M = HI). Controls directly the P/M line of the C-BUS.

The $\overline{\text{MASK}}$ is so connected to the K-inputs of the 3002 array that,

$\overline{\text{MASK}}_4$	masks the 5 MSb of the data
$\overline{\text{MASK}}_3$	" just the 6 th MSb of the data
$\overline{\text{MASK}}_2$	" " " 7 th " " " "
$\overline{\text{MASK}}_1$	" " " 8 th " " " "
$\overline{\text{MASK}}_0$	" the 4 LSB of the data

Besides, the 5 MASK lines also go the 5 MSb of the I-inputs to the 3002 array in order to generate the warning messages sent to the Console if a mistake happens to appear during a message processing.

As shown already, the Microprogram Store is divided in two halves of 256 positions, such that a microprogram in one section needs not to access the other half. Thus, an 8-bit PL ADDRESS suffices for any jump within a microprogram, while the MSb (bit 8) of the microprogram address remains unchanged.

The CCSEL field is zero when the next microinstruction address does not depend on any condition. This arises from the fact that CCSEL \neq 0 and NEXT ADDRESS = C₁₆ or E₁₆ provoke ICU addressing or Lock Register loading, respectively.

The fields occupy the following positions within the microinstruction:

Field	Bit positions
$\overline{\text{MASK}}_0$ to 4	00 to 04
FUNCTION 0 to 6	05 to 11
$\overline{\text{CI}}$	12
CCSEL 0 to 2	13 to 15
$\overline{\text{D-ENABLE}}$	16
$\overline{\text{A-ENABLE}}$	17
$\overline{\text{D-REG}}$	18
P/M	19

L/P	20
MAP	21
M	22
F	23
NEXT ADDRESS 0 to 3	24 to 27
PL ADDRESS 0 to 7	28 to 35

Field values in various circumstances:

	A-ENABLE	D-ENABLE	D-REG	Comments
Read operation	0	1	0	MAP, L/P, P/M, F, M assume the convenient values.
Write operation	0	0	1	
D-Reg. ← Working register AC	1	0	0	M = LOW

	MAP	L/P	P/M	M	Comments
LCI/SMI address.	1	0	1	0	4 LSb of A-BUS = 0/1.
Console address.	0	0	1	0	4 LSb of A-BUS = F_{16}
Netw. Map addr.	1	-	0	0	L/P, A-BUS assume desired value.
Rest of Memory	0	0	0	-	A-BUS contains 11-bit address.

	CCSEL	NEXT ADDR.	D-ENABLE	D-REG	Comments
Lock loading	n	E_{16}	x	x	$LOCK \leftarrow n$ ($n=1, \dots, 7$)
ICU write	$\neq 0$	C_{16}	0	1	A-ENABLE = HI
ICU read	$\neq 0$	C_{16}	1	0	

e. Microprogram Store Organization (Fig. 19)

It comprises two sections, one for the microprograms corresponding to the standard instructions and the other for the special instructions. The initialization base microprogram starts at address 0. It must load a proper lock and next, fetch the first Initialization program instruction from the Memory and start executing it.

When I/O ERROR goes HI, a branch occurs to the microinstruction address $0FF_{16}$ (for standard instructions trying to

violate the Lock/Key scheme) or $1FF_{16}$ (for special instruction conflicting with the Lock). Thus, these positions are supposed to contain the start of the reinitialization microprocedure, that similarly hands over the control to the first instruction of the Reinitialization program.

Furthermore, addresses $0E0_{16}$ to $0F5_{16}$ contain the routines dealing with the register addressing instruction. They load the internal working register AC with the contents of the addressed register, at the beginning of the instruction execution. At the end, if necessary, these routines store the result back into the desired register.

Excepting these special positions, the microprograms may occupy any other position.

f. Internal registers (Fig. 19)

The Intel 3002 offers 11 internal registers: AC, T, R1, R2, ..., R8, R9.

AC commands the D-outputs and it is an implicit operation in many 3002 microinstructions. So, it cannot be used to store information, and it is our internal working register, not directly accessible by the instructions.

T is the accumulator ACC as far as the instructions are concerned.

R9 contains the Status Word and the Error Number.

Bits 0 to 3	-	Present Interrupt Level	} Status Word
4	-	<u>Scan Enable</u>	
5	-	<u>Conn Enable</u>	
6	-	<u>Int. Enable</u>	
7 to 11	-	Error Number	

Present Interr. Level occupies 4 bits foreseeing a possible expansion to accommodate 16 interrupt lines. Higher priority corresponds to higher level

Scan Enable, when LOW, LCI may be polled. Conn Enable when LOW, connection requests from devices may be carried out. If Int. Enable is LOW, ICU attends interrupt requests.

Error number equal to 0 means no error has been detected.

R8 is used as the 11-bit Program Counter (PC). The MSb is not used.

R7 contains the 11-bit Stack Pointer (SP). MSb not used.

5 R6 saves the last message received from the LCI during the Network Operation program.

R5 has in bits: 0 to 5 - Index (ranging from 0 to 63)

6 - Condition: HI = TRUE

LOW = FALSE

10 7 to 11 - all LOW = no indexation

all HI = Index must be added
to the absolute addr.

15 An INDX instruction provokes the Index field in R5 to be loaded with the 6 LSb of the register addressed while bits 7 to 11 in R5 are set HI. After an absolute addressing instruction appears, these latter bits are cleared.

Finally, there is an internal register MAR which only use is to command the A-outputs of the 3002. So, it should contain the desired address in a read/write operation.

7. MICROPROGRAMS

Attention was focused on the microprogram performing the special instruction LCI MESS.

LCI messages have the message code in the first word and the addresses of the devices involved, in the second.

If Scan Enable is HI, a branch to the microroutine Next Instruction occurs. If LOW, LCI is polled and the first word suffers decoding by the Mapping PROM, generating the address in the Microprogram Store where the convenient procedure for that message begins.

The LOG-to-PHY conversion word of the requesting device a, indicated as (LOG-a), is loaded into register R5. The PHY-to-LOG conversion word, (PHY-a), goes into R4. If another device b is involved, (LOG-b) goes into R3, and R2 gets (PHY-b). Thus, during processing, all data needed is available internally, saving accesses to the Network Map.

The philosophy of the consistency verifications made on message and Map is that, in case of doubt, it is better not to connect. Otherwise, as the errors accumulate, we may end up with a congested network, full of wrong connections that stay there permanently. Whenever doubtful connections may exist, disconnection is ordered. Thus, degradation of the CNET due to an eventual series of errors is avoided.

At the end of the procedure, Network Map is updated, and the Next Instruction microroutine is executed. In this routine, an eventual error message is forwarded to the Console, if there is an error number stored in register R9. Interrupt requests are polled and if no branch to an interrupt subroutine is forced, then the next instruction is fetched.

The convention for the flow diagrams in fig. 20 to 28, is:

LOG-x = LOG address of device x

PHY-x = PHY address of device x

a = requesting device

b = device with whom a is talking or wants to talk. LOG-b

is given either in the device message or by the PARTNER

field of (PHY-a).

c = device that is waiting to be connected to a. LOG-c is given by the WAITING PART field of (LOG-a).

The CONS COMM special instruction was not developed since the Console is yet unclearly defined and modifications may invalidate most of the effort.

The standard instructions, due to the want of time, were only partly developed. Three phases exist in an instruction: fetch, addressing and execution.

The instruction fetch microroutine, which brings to the D-Register the instruction pointed at by the PC, and the addressing microroutine, fetches the desired operand at the start of the instruction and stores the result at the end (if necessary), are detailed in fig. 29. Both routines are common to most of the instructions.

Most of the execution microprograms are more or less straightforward, requiring few microinstructions. There is a kind of standardization in that the working register AC contains the needed operand at the start of the execution phase, and the result of this phase is left also in AC.

8. TESTING (Fig. 30,31,32)

As stated before, only the CPU came to be implemented.

Some difficulties were met in trying to get the components, and in fitting the odd-sized wire-wrap boards available in the group EB into a double Euroformat rack.

The CPU was wirewrapped instead of soldered because of the limited time available. Though the high crosstalk found in wirewrapped circuits prevents a high frequency clock to be employed, nevertheless, the idea can be tested. A printed circuit board version may be developed in case the CPU is found to perform satisfactorily.

Furthermore, all PROM's were substituted by 2708 UV-erasable PROM's with 450 ns. access time, far more than the 70 ns. access time specified for the MMI 6301 which should be employed in the final version. Since speed does not constitute the present goal, the advantage of reprogrammability is invaluable in the prototype development.

To help debugging, a panel was added, where the main signals can be visualized by means of LED's. A burning LED means the corresponding bit is '1', in positive logic. Due to the absence of a CPU internal oscillator, an external oscillator input as well as a single step switch are provided. Data may be input through the switches directly connected to the D-BUS. See fig. 12 and 13.

Up to date, the hardware units have been tested separately, but the CPU as a whole has not been submitted to test. It is expected that the CPU, viewed from the hardware point of view will not present major troubles. Also the microprograms corresponding to the standard instructions are rather straightforward.

Microprogram allocation in the Microprogram Store presents no difficulty since the Microprogram Controller offers much flexibility.

The main task consists in implementing the lengthy microprograms that perform the special instructions. After this, the next step is to develop the Initialization, Reinitialization

programs and then, some diagnosis programs. The Network Operation program is reduced to about three instructions, since the load is assumed by the microprograms.

To whoever, if anyone, comes to continue the project, I wish much success.

APPENDIX a: Am 2910 Microprogram Controller Emulation (Fig. 6)

The PROM used for decoding the Am2910 instructions has as inputs:

NEXT ADDRESS, CC, I/O ERROR connected to the address lines.

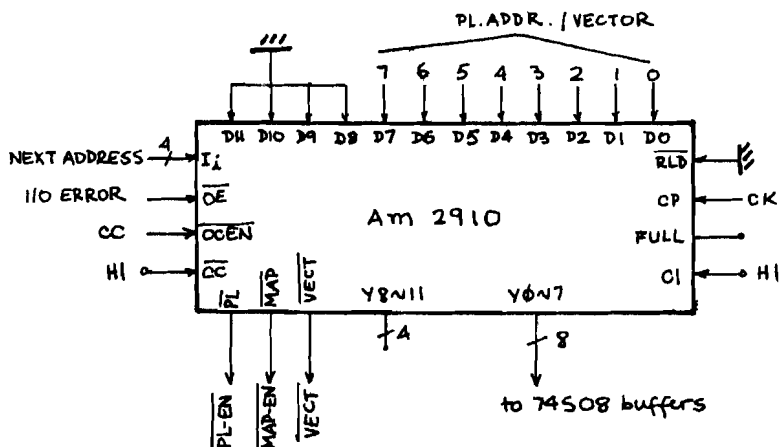
Initially a Harris HPROM 0512 with 64x8-bit words was supposed to be used, but due to lack of a proper programmer, an Intel 2708 took its place and only 64 of the 1K words contain information. The contents of the PROM are in the table next page.

I/O ERROR = HI sets HI all the outputs of the Am 2909's and provokes an unconditional JUMP PL. This feature can only be implemented in the Am2910 by connecting I/O ERROR to its \overline{OE} input. When the signal goes HI, all the Am 2910 outputs go to a high impedance state, which causes the 74S08 buffers to go all HI, thus having a similar final effect, i.e., to force execution of microinstruction at address xFF_{16} .

The Am 2910 features not present in the emulator are :

- . THREE-WAY BRANCH instruction (code F_{16}) was in fact substituted by a REPEAT LOOP, $\text{CNTR} \neq 0$.
- . JMP ZERO does not clear internal stack because the Am2909 possesses no such facility.
- . In COND JSB R/PL and COND JUMP R/PL, if Register is selected, it always gives 00 address.
- . The 4-bit event counter employed has much smaller counting capability than the 12-bit counter present in the Am2910.

The Am 2910 is supposed to be connected as below:



ADDRESS INPUTS					OUTPUTS						INSTRUCTION for			
I/O	CC	NEXT ADDR.				LD	EN	PL-	MAP-	PUP	FE	S1	S0	MICROPROGRAM CONTROLLER
ERROR		3	2	1	0	CNT	CNT	EN	EN					
0	0	0	0	0	0	1	1	0	1	1	1	0	1	JUMP ZERO
0	0	0	0	0	1	1	1	0	1	1	1	0	0	COND JSB PL
0	0	0	0	1	0	1	1	1	0	1	1	1	1	JUMP MAP
0	0	0	0	1	1	1	1	0	1	1	1	0	0	COND JUMP PL
0	0	0	1	0	0	1	1	0	1	1	0	0	0	PUSH/COND LD CNTR
0	0	0	1	0	1	1	1	0	1	1	0	0	1	COND JSB R/PL
0	0	0	1	1	0	1	1	1	1	1	1	0	0	COND JUMP VECTOR
0	0	0	1	1	1	1	1	0	1	1	1	0	1	COND JUMP R/PL
0	0	1	0	0	0	1	0	0	1	1	1	1	0	REPEAT LOOP,CNT≠0
0	0	1	0	0	1	1	0	0	1	1	1	1	1	REPEAT PL, CNT≠0
0	0	1	0	1	0	1	1	0	1	1	1	0	0	COND RTN
0	0	1	0	1	1	1	1	0	1	1	1	0	0	COND JUMP PL&POP
0	0	1	1	0	0	0	0	0	1	1	1	0	0	LD CNTR&CONTINUE
0	0	1	1	0	1	1	1	0	1	1	1	1	0	TEST END LOOP
0	0	1	1	1	0	1	1	0	1	1	1	0	0	CONTINUE
0	0	1	1	1	1	1	0	0	1	1	1	1	0	REPEAT LOOP,CNT≠0
0	1	0	0	0	0	1	1	0	1	1	1	0	1	JUMP ZERO
0	1	0	0	0	1	1	1	0	1	1	0	1	1	COND JSB PL
0	1	0	0	1	0	1	1	1	0	1	1	1	1	JUMP MAP
0	1	0	0	1	1	1	1	0	1	1	1	1	1	COND JUMP PL
0	1	0	1	0	0	0	0	0	1	1	0	0	0	PUSH/COND LD CNTR
0	1	0	1	0	1	1	1	0	1	1	0	1	1	COND JSB R/PL
0	1	0	1	1	0	1	1	1	1	1	1	1	1	COND JUMP VECTOR
0	1	0	1	1	1	1	1	0	1	1	1	1	1	COND JUMP R/PL
0	1	1	0	0	0	1	1	0	1	0	0	0	0	REPEAT LOOP,CNT≠0
0	1	1	0	0	1	1	1	0	1	1	1	0	0	REPEAT PL,CNT≠0
0	1	1	0	1	0	1	1	0	1	0	0	1	0	COND RTN
0	1	1	0	1	1	1	1	0	1	0	0	1	1	COND JUMP PL&POP
0	1	1	1	0	0	0	0	0	1	1	1	0	0	LD CNTR&CONTINUE
0	1	1	1	0	1	1	1	0	1	0	0	0	0	TEST END LOOP
0	1	1	1	1	0	1	1	0	1	1	1	0	0	CONTINUE
0	1	1	1	1	1	1	1	0	1	0	0	0	0	REPEAT LOOP,CNT≠0
1	x	x	x	x	x	1	1	0	1	1	1	1	1	I/O ERROR

APPENDIX b: Connectors

The connections between the wirewrapped boards go through two 70-pin connectors. As we look at the components' side of the board, we have the Left connector and the Right connector. Each connector possesses two rows of 35 pins: row a on the connections' face and row b on the components' face. Pins are numbered 1 to 35 starting from the right end of the connector as we face the components' side of the board.

The table on the next page shows the pin assignment.

PIN	LEFT a	LEFT b	RIGHT a	RIGHT b
1	CK	CK	-	+12 V
2	RESET	WDT	-	+12 V
3	$\overline{\text{FRESH}}$	$\overline{\text{ICUW}}$	$\overline{\text{MASK}} 0$	FUNCTION 0
4	$\overline{\text{CK}}$	$\overline{\text{ICUR}}$	$\overline{\text{MASK}} 1$	FUNCTION 1
5	-	-	$\overline{\text{MASK}} 2$	FUNCTION 2
6	$\overline{\text{DB}} 0$	$\overline{\text{DB}} 6$	$\overline{\text{MASK}} 3$	FUNCTION 3
7	$\overline{\text{DB}} 1$	$\overline{\text{DB}} 7$	$\overline{\text{MASK}} 4$	FUNCTION 4
8	$\overline{\text{DB}} 2$	$\overline{\text{DB}} 8$	$\overline{\text{CI}}$	FUNCTION 5
9	$\overline{\text{DB}} 3$	$\overline{\text{DB}} 9$	$\overline{\text{D-ENABLE}}$	FUNCTION 6
10	$\overline{\text{DB}} 4$	$\overline{\text{DB}} 10$	$\overline{\text{A-ENABLE}}$	L/P
11	$\overline{\text{DB}} 5$	$\overline{\text{DB}} 11$	-	MAP
12	-	-	-	-5 V
13	LAL	LAL	$\overline{\text{D-BUS}} 0$	$\overline{\text{D-BUS}} 6$
14	-	-	$\overline{\text{D-BUS}} 1$	$\overline{\text{D-BUS}} 7$
15	INT.LINE 0	INT.LINE 4	$\overline{\text{D-BUS}} 2$	$\overline{\text{D-BUS}} 8$
16	INT.LINE 1	INT.LINE 5	$\overline{\text{D-BUS}} 3$	$\overline{\text{D-BUS}} 9$
17	INT.LINE 2	INT.LINE 6	$\overline{\text{D-BUS}} 4$	$\overline{\text{D-BUS}} 10$
18	INT.LINE 3	INT.LINE 7	$\overline{\text{D-BUS}} 5$	$\overline{\text{D-BUS}} 11$
19	-	-	-	-
20	OSC	-	R/W	$\overline{\text{INTREQ}}$
21	-	-	D-OUT CK	CO
22	$\mu\text{A} 0$	$\mu\text{A} 4$	$\overline{\text{A-BUS}} \text{ VALID}$	D-IN CK
23	$\mu\text{A} 1$	$\mu\text{A} 5$	ACK	-
24	$\mu\text{A} 2$	$\mu\text{A} 6$	A-BUS 0	A-BUS 6
25	$\mu\text{A} 3$	$\mu\text{A} 7$	A-BUS 1	A-BUS 7
26	-	$\mu\text{A} 8$	A-BUS 2	A-BUS 8
27	-	-	A-BUS 3	A-BUS 9
28	-	-	A-BUS 4	A-BUS 10
29	-	-	A-BUS 5	P/M
30	-	-	-	-
31	-	-	LAL	LAL
32	-	-	CC	I/O ERROR
33	-	-	-	-
34	+ 5 V	GND	-	-
35	+ 5 V	GND	-	-

APPENDIX c: Signal timing

. Clock cycle

Depending on the microinstruction, different routes are followed by the signals and diverse delays are involved.

The emulation circuit delays were used in the calculations, since specifications on the Am2910 are yet unknown. It is expected that the Am2910 will improve the speed, allowing shorter cycles.

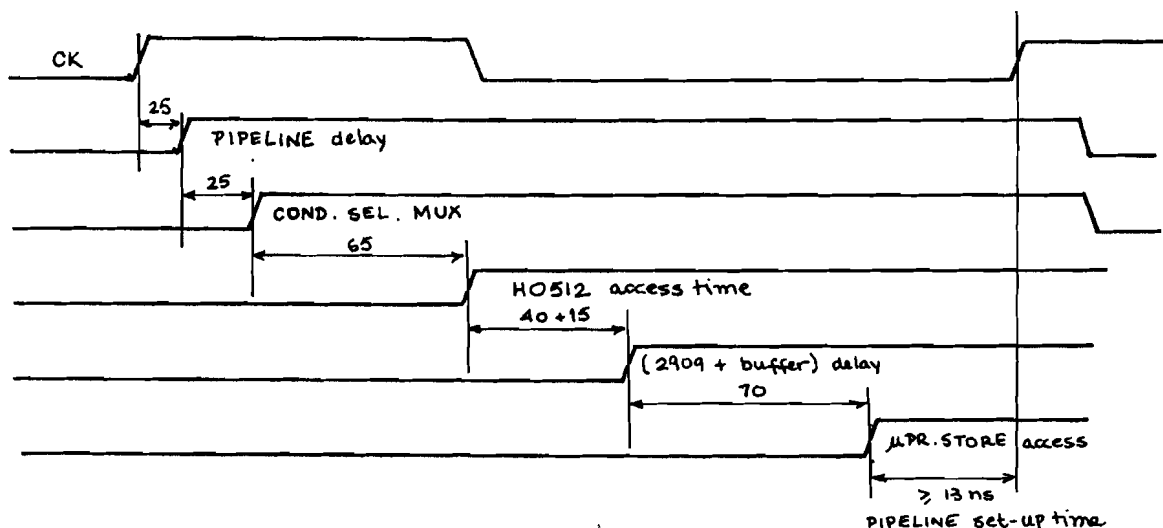
For the clock cycle time determination, the worst cases must be taken into account. The delays employed in the calculations consider the case if the desired PROM's were in the circuit in place of the 2708 EPROM's. When data available in the manuals does not take in account power supply and temperature variations, as it is the case with the 74 TTL series manuals, the times used in the diagrams are at least 50% larger than the maximum specified for nominal conditions ($V_{CC} = +5\text{ V}$, Temp. = 25°C).

The clock cycle time consists of an initial HI semicycle lasting t_{HI} and followed by a LOW semicycle t_{LOW} . The total cycle is indicated by t_{CY} .

Considering the most critical cases, we find the maximum clock frequency possible in the circuit.

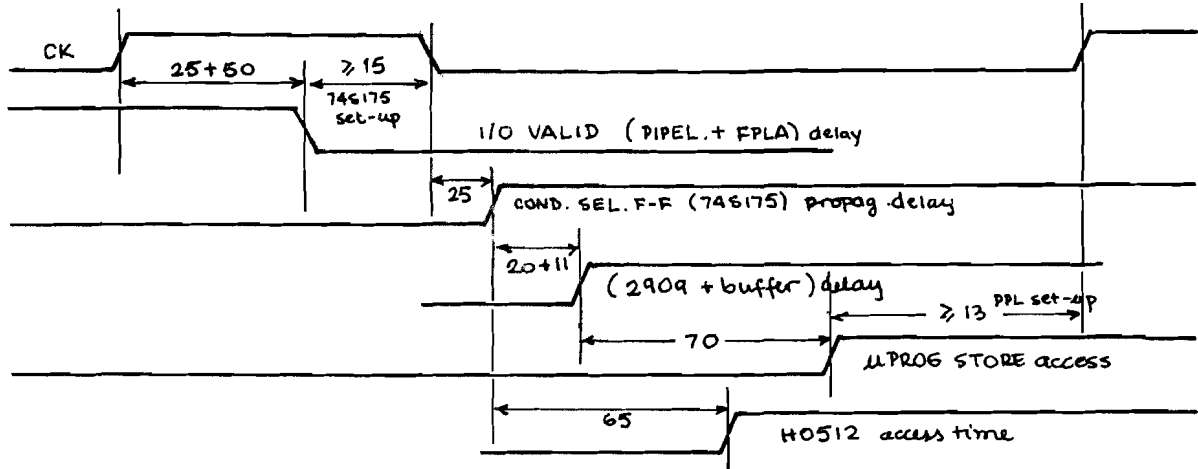
. Non read/write operation, next address not depending on condition now being generated:

$$t_{CY} > 253\text{ ns.}$$



- read/write operation violating lock:

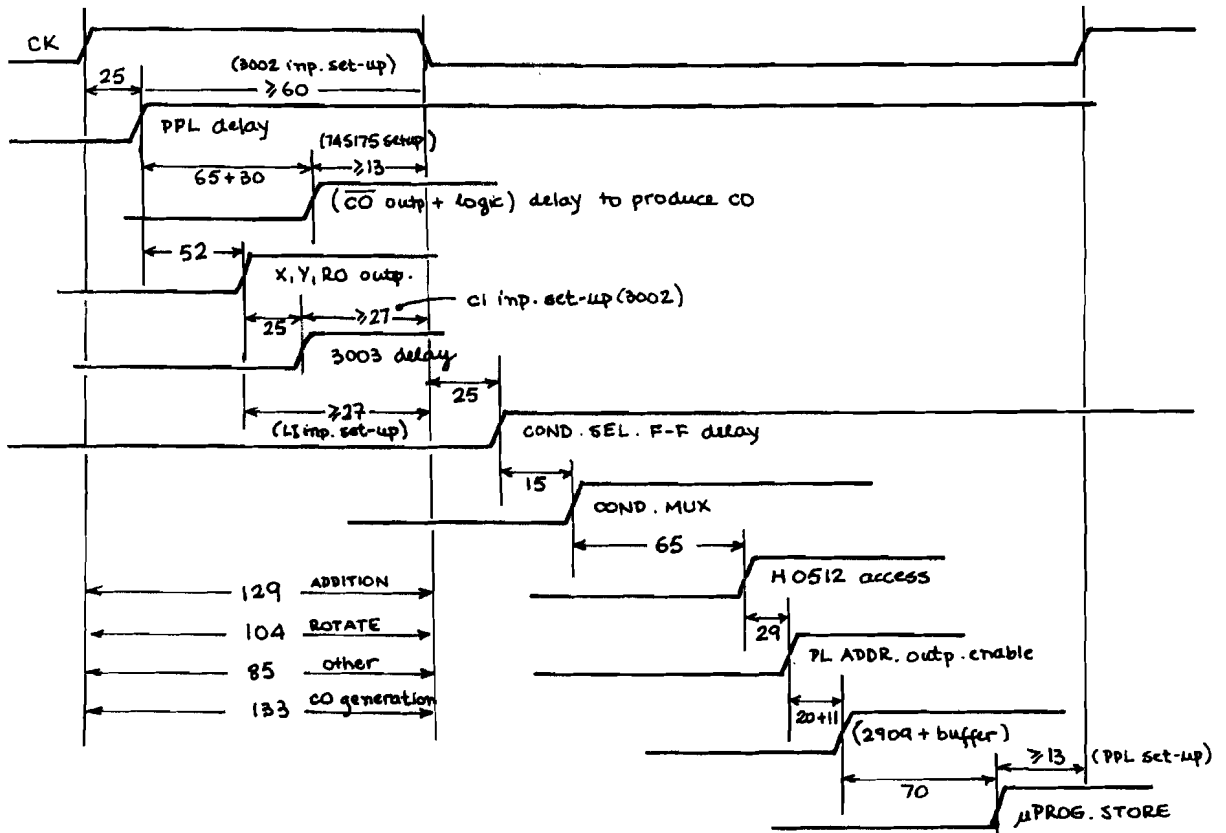
$$t_{HI} > 90 \text{ ns} \quad t_{LOW} > 139 \text{ ns}$$



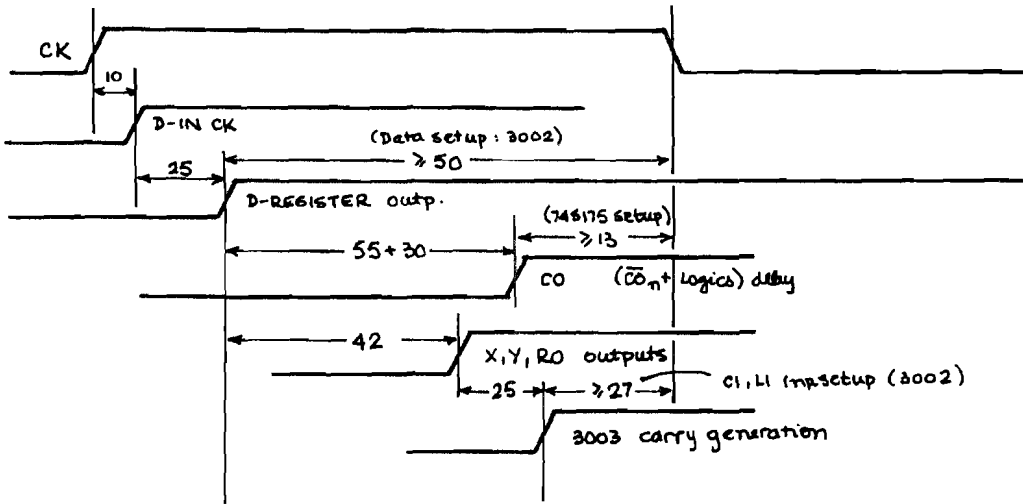
- arithmetic operation, condit. branch depending on CO:

$$t_{LOW} > 268 \text{ ns} \quad t_{HI} > 129 \text{ ns (addition)}$$

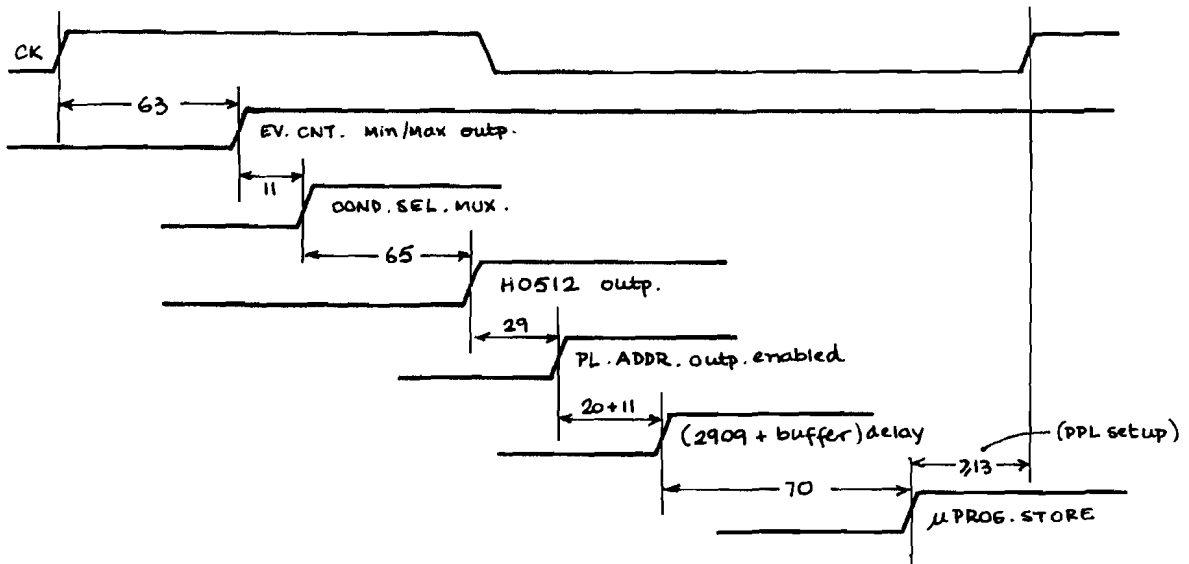
$$t_{HI} > 133 \text{ ns (for CO generation before end of HI semicycle)}$$



- arithmetic operation on data just read into D-Register
 $t_{HI} > 129$ ns (addition)
 $t_{HI} > 133$ ns (for CO generation before end of HI semicycle)



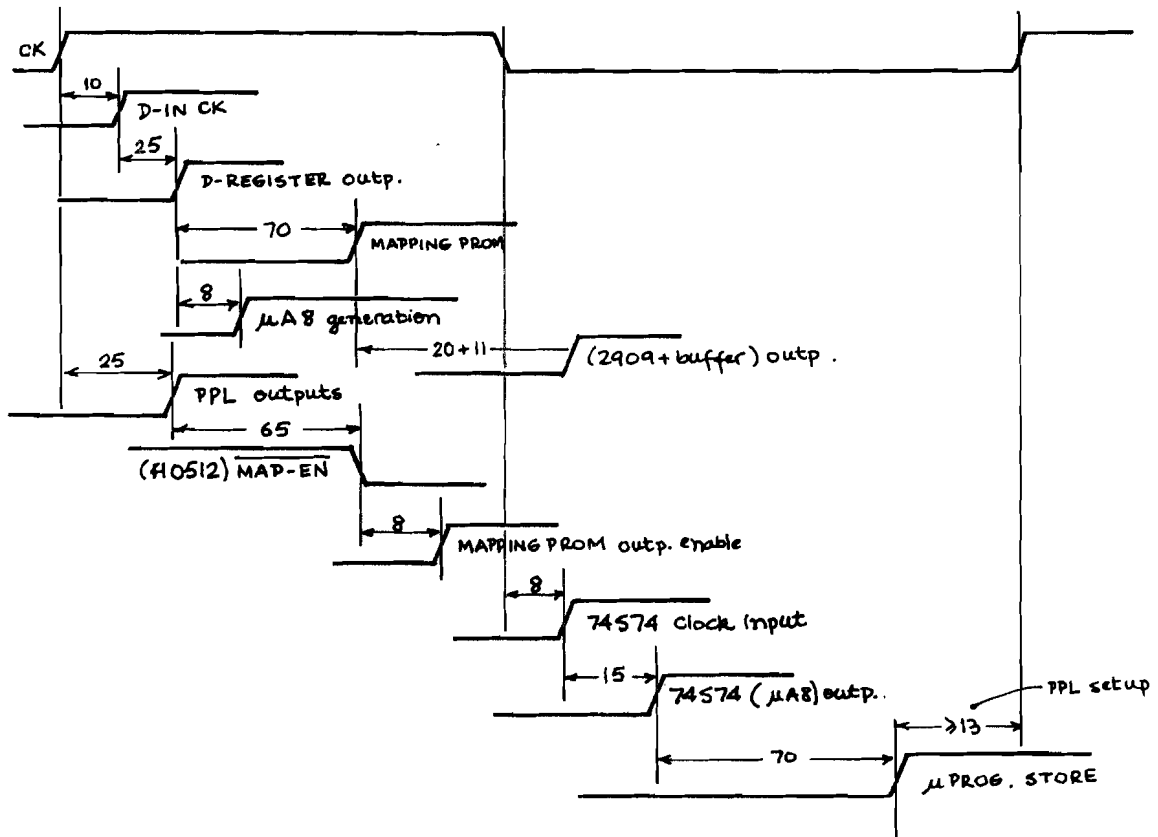
- conditional branch if event counter = 0, supposing it was decremented in the previous microinstruction.
 $t_{CY} > 282$ ns



- jump to address given by Mapping PROM

$$t_{HI} > 98 \text{ ns}$$

$$t_{LOW} > 106 \text{ ns}$$

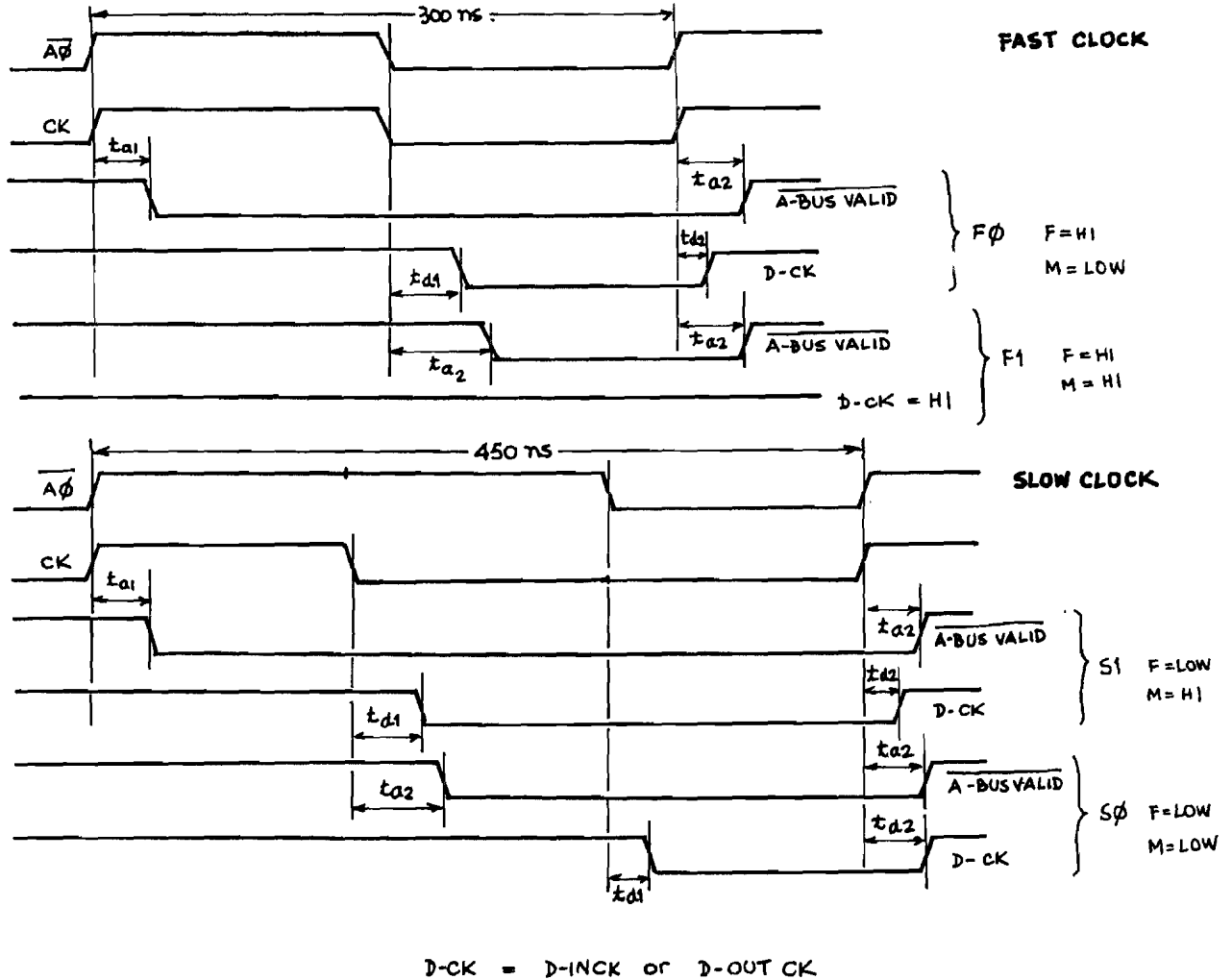


Thus, to satisfy all possible cases, t_{HI} was set as 150 ns., while t_{LOW} may be either 150 ns. (fast clock cycle), or 300 ns. (slow clock cycle), when there is a conditional branch on CO. Consequently, t_{CY} is either 300 ns. or 450 ns.

- Read/write Operations

Fast and slow clock cycles also find use in read/write operations, where the addressed unit is known to satisfy certain speed requirements. They provide timing for the C-BUS signals: A-BUS VALID and D-OUT CK (in case of a write) or D-IN CK (if it is a read). The positive-going edge of D-OUT CK and D-IN CK is used to clock the data into the receiving register. So, these signals should rise before A-BUS VALID goes HI, to guarantee the data on D-BUS is still valid.

According to the value assumed by the F and M bits in the microinstruction, the following patterns are generated:



$$t_{a1} = (\text{Pipeline} + \text{FPLA}) \text{ propag. delays} < 25 + 50 = 75 \text{ ns}$$

$$t_{d1} = t_{a2} = \text{FPLA propag. delay} < 50 \text{ ns}$$

$$t_{d2} = \text{74S00 propag. delay} < 8 \text{ ns}$$

Taking in consideration the worst cases, we have calculated the requirements concerning the access time t_{ACCESS} (from address to output onto the \overline{D} -BUS) and the write cycle pulse width t_{WP} . From the t_{ACCESS} we must deduct 15 ns for the data set-up time of a 74S174 register (taken as the standard receiving register on the bus lines). These requirements are to be met by the addressed units in case the CPU uses one of the patterns above to read or write it.

F0 is used when the 3002 internal register MAR already contains the desired address:

$$t_{\text{ACCESS}} < 225 - 15 = 210 \text{ ns} \quad t_{\text{WP}} < 100 \text{ ns}$$

S0 applies for write operations when either register MAR or AC are going to be loaded with the desired value during the present microinstruction. In read, of course, it can only be register MAR that is being now loaded:

$$t_{\text{ACCESS}} < 250 - 15 = 235 \text{ ns} \quad t_{\text{WP}} < 100 \text{ ns}$$

F1 followed by S1 gives a long A-BUS VALID signal. Register MAR, and also AC if it is a write, should be loaded before the S1 pattern microinstruction.

$$t_{\text{ACCESS}} < 550 - 15 = 535 \text{ ns} \quad t_{\text{WP}} < 250 \text{ ns}$$

To possibilitate quicker read/write operations, i.e., the ones using F0 and S0 patterns, the fast section of the Memory, the LCI, the SMI and the Console must stay within the limits:

$$t_{\text{ACCESS}} < 210 \text{ ns} \quad t_{\text{WP}} < 100 \text{ ns}$$

The slower section of the Memory is supposed to be read/written by a sequence of two microinstructions, a F1 pattern followed by a S1. Therefore, it has:

$$t_{\text{ACCESS}} < 535 \text{ ns} \quad t_{\text{WP}} < 250 \text{ ns}$$

For peripherals with unknown timing, a wait for ACK loop may be used. The peripheral, when ready, pulls LOW the ACK line, notifying the CPU.

It is recommended that different sources do not write on the D-BUS in two consecutive clock cycles, because one may enable its output before the previous source has had time to disable. For some instants, two enabled outputs may be short-circuited.

Some examples of read/write operations, where only the microinstruction fields involved in the operation are shown. The '3002 task' means that FUNCTION, MASK and CI fields must have a proper value so that the desired task is performed by the 3002 array.

Ex. a : PHY-to-LOG conversion table (in Network Map) read operation. Register MAR was not yet loaded with the desired PHY address.

Ex. b : LOG-to-PHY conversion table (Network Map) write operation.

Register MAR was beforehand loaded with LOG address.

Ex. c : LCI write operation. The 4 LSB of MAR = 0 already, but register AC not yet loaded with the command to be sent.

Ex. d : Slow Memory write. Position 'Ad' loaded with data 'Data'.

Ex. e : Slow Memory read. AC loaded with contents of position 'Ad'

F	M	MAP	L/P	P/M	<u>D-REG</u>	<u>A-ENABLE</u>	<u>D-ENABLE</u>	3002 task	Ex.
0	0	1	0	0	0	0	1	MAR ← PHY	a
1	x	0	0	x	1	1	1	AC ← D-Register	
1	0	1	1	0	1	0	0	AC ← Data	b
1	0	1	0	1	1	0	0	AC ← command	c
1	1	0	0	0	1	0	0	MAR ← Ad	d
0	1	0	0	0	1	0	0	AC ← Data	
1	1	0	0	0	0	0	1	MAR ← Ad	
0	1	0	0	0	0	0	1	-	e
1	x	0	0	x	1	1	1	AC ← D-Register	

. ICU polling

To poll the ICU, a two-step sequence applies. First, the Status Word is sent to ICU in a S0 pattern write operation (F0 would not give time enough for the Intel 3214). In the next clock cycle, the ICU writes a LOW on the INTREQ line of the C-BUS if it decides that an interruption should be requested. This line is valid during only one clock cycle, so that it must be tested in the second microinstruction. The requesting interrupt level may be read in a F0 pattern operation at any time after the polling.

APPENDIX d : Instruction Codes

All instruction, LCI messages, Console commands are decoded by the Mapping PROM, implying different codes must be allocated to them.

Due to the way in which we have implemented the division of the Microprogram Store in two independent sections, the device messages, the LCI messages, Console commands and the two special instructions, all of which have their microprograms in the upper half of the Microprogram Store, are supposed to have the 2 MSb of their codes both HI.

The codes proposed are complemented to the way as they appear on the D-BUS lines, due to the fact that those lines are in inverted logic. So, logical '1' represents a HI, but on the D-BUS a LOW will be written.

	1 st word	2 nd word
. Special instructions		
LCI MESS	1111 0101 xxxx	
CONS COMM	1100 1100 xxxx	
. Device messages		
'Connect <u>a</u> to <u>b</u> '	1110 1110 xxxx	LOG-b xx PHY-a
'Disconnect <u>a</u> and <u>b</u> '	1110 1101 xxxx	LOG-b xx PHY-a
'Inoperative'	1110 1011 xxxx	xxxxxx xx PHY-a
'Operative'	1110 0111 xxxx	xxxxxx xx PHY-a
'LOG = <u>a</u> '	1110 0001 xxxx	LOG-a xx PHY-a
. LCI messages		
'No messages'	1111 1111 xxxx	
'Test OK'	1111 0110 xxxx	
'Test failed'	1111 0000 xxxx	
'LC <u>a</u> crazy'	1111 1010 xxxx	xxxxxx xx PHY-a

. Console commands

'Load Memory'	1100 0000 xxxx	
'Dump Memory'	1100 0110 xxxx	
'Dump Registers'	1100 1010 xxxx	
'Go to n'	1100 1111 xxxx	
'New Priority Mask'	1101 0001 xxxx	
'New Status Word'	1101 0111 xxxx	
'Device <u>a</u> off'	1101 1011 xxxx	xxxxxx xx LOG-a
'Device <u>a</u> on'	1101 1101 xxxx	xxxxxx xx LOG-a
'No commands'	1111 1111 xxxx	

LOG-a = logical address of device a

PHY-a = physical address of device a

x = doesn't matter

The Console command codes are proposed above, but many of them need some more words of information, and these following words are left open for further definition.

The codes above are arranged in order to keep a Hamming distance of at least 2 between any two codes, to guarantee further safety.

Note that 'No commands' and 'No messages' share the same code, since both lead to the microroutine that fetches the next instruction. Like the invalid codes, they behave as a NOP instruction.

The standard instructions occupy the lower part of the Microprogram Store and must not have the 2 MSb of their code both HI. In register addressing mode, the 4 LSb of the code select the register:

Register	4 LSb
R1	1110
R2	1101
R3	1100
R4	1011
R5	1010

R6	1001
R7	1000
R8	0111
R9	0110
ACC (T)	0101

In all other addressing modes, the 4 LSb are all HI. Immediate addressing requires the operand to be in the second word. In absolute addressing, the MSb of the second word indicates indirection is it is HI, and an 11-bit address occupies the rest of the word.

The table on the page after the next shows the codes assigned to the standard instructions. In this table,

rrrr assumes the appropriate value for the register R
r'r'r'r' " " " " " " " R'
ccc " " " " to select a condition
pppp is the address of the peripheral.
LLL is the new LOCK value

Now, the different messages and commands originated in the CPU may repeat the codes above, since they are not decoded by the Mapping PROM.

. MC messages to device

'You are operative'	00000 01 00000
'You are inoperative'	11111 01 00000
'Xpoint defective'	00000 11 00000
'You are crazy'	11111 01 10000
'Device <u>b</u> inoperative'	00000 01 10000
'Device <u>b</u> busy'	11111 11 10000
'Connections disabled'	00000 11 10000
'Your logical address?'	11111 11 00000

. CPU commands to LCI

'Break Xpoint'	11111 00 00000	PHY-a xx PHY-b
'Scan'	00000 00 10000	
'Test'	00000 00 11111	PHY-a xx PHY-b
'Stop'	11111 00 10000	
'New Priority Mask'	11111 10 10000	

'Send Message to device'	00000 10 10000	
'Connection to device <u>a</u> '	00000 10 01111	xxxxx xx PHY-a
. MC messages to Console		
'Initializing'	00000 01 01111	
'Reinitializing'	00000 10 01111	
'Disconnection failed'	00000 11 01111	
'Connection failed'	11111 00 01111	
'Waiting Connection'	11111 10 01111	
'Send Status Word'	00000 00 11111	
'LCI crazy'	00000 01 11111	
'Check LCI'	00000 10 11111	
'Check SM'	00000 11 11111	
'Error message'	leeee 11 11111	

eeee = Error Number

. CPU command to SMI

The only command is the order to accionate a crosspoint (a,b). So, no command code is needed, just the addresses are sent in a write operation:

Make Xpoint PHY-a xx PHY-b

The codes chosen for these messages and commands produced by the CPU are intended to be easily generated within a micro-instruction, by using the MASK field to form these bit patterns.

STANDARD INSTRUCTION CODES

Mnemonics	Addr.mode	Instruct. code
LOAD	R	1011 r'r'r'r' rrrr
	A	1011 0011 1111
	I	1011 0000 1111
STORE	A	1010 1111 1111
INP, n	ACC	1001 pppp 1111
	A	1000 pppp 1111
OUTP, n	ACC	0101 pppp 1111
	A	0100 pppp 1111
XCH	R	1010 1001 rrrr
	A	1010 1000 1111
AND	R	0111 1111 rrrr
	A	0111 1101 1111
	I	0111 1100 1111
IOR	R	0111 1011 rrrr
	A	0111 1001 1111
	I	0111 1000 1111
XOR	R	0111 0011 rrrr
	A	0111 0001 1111
	I	0111 0000 1111
ANDM	R	0111 1110 rrrr
IORM	R	0111 1010 rrrr
XORM	R	0111 0010 rrrr
ADD	R	0110 1111 rrrr
	A	0110 1110 1111
	I	0110 1101 1111
ISZ	R	0110 0111 rrrr
DSZ	R	0110 0011 rrrr
CLEAR	R	1010 0111 rrrr
SET	R	1010 0101 rrrr
CMPL	R	1010 0011 rrrr
RTR, n	R	0001 0nnn rrrr
JSBC, c	A	0011 0ccc 1111
JMPC, c	A	0011 1ccc 1111
RETC, c	R	0010 0ccc 1111

INDX	R	0000 1111 rrrr
LOCK, n	-	0001 1LLL 1111
HALT	-	0000 0000 1111

APPENDIX e : FPLA

A Signetics 82S100 16x48x8 FPLA concentrated most of the logics, keeping down the chip count.

Fifteen inputs were occupied by signals coming from:

- . Clock : CK, $\overline{A0}$
- . Pipeline : $\overline{D-ENABLE}$, $\overline{A-ENABLE}$, $\overline{D-REG}$, F, M, P/M, MAP, NEXT ADDRESS 1
- . Lock/Key : $\overline{CC-OR}$, IDENT, LOCK (indicated as L0, L1, L2)
 $\overline{CC-OR}$ goes LOW when CCSEL \neq 0.
 IDENT goes HI when NEXT ADDRESS = C₁₆ or E₁₆.

The FPLA generates seven outputs:

- . \overline{ICUW}

It goes LOW in a ICU write operation, during the HI period of CK.

$$\overline{ICUW} = \underbrace{\overline{IDENT} \cdot \overline{NEXT ADDRESS 1}}_{\text{NEXT ADDRESS} = C_{16}} \cdot \overline{CC-OR} \cdot \overline{D-ENABLE} \cdot CK$$

- . \overline{ICUR}

LOW in a ICU read operation, during the whole clock cycle.

$$\overline{ICUR} = \overline{IDENT} \cdot \overline{NEXT ADDRESS 1} \cdot \overline{CC-OR} \cdot \overline{D-REG}$$

- . I/O VALID

LOW if there is an attempt to read/write conflicting with the present value of LOCK.

$$\overline{I/O VALID} = \overline{APROVED} \cdot \overline{A-ENABLE}$$

The internal auxiliary function $\overline{APROVED}$ goes HI when the address and operation intended do not fit the lock. But $\overline{I/O VALID}$ only goes HI if the operation is really attempted, i.e., if $\overline{A-ENABLE}$ is HI.

$$\overline{APROVED} = \overline{L0.L1.L2} + \overline{MAP.L2.L0} + \overline{P/M.MAP.L1} + \overline{P/M.MAP.L2} + \overline{P/M.D-REG.L1.L0} + \overline{MAP.D-REG.L0}$$

- . L-LOAD

At its positive-going edge, the Lock Register loads the CCSEL field as the new LOCK.

$$L-LOAD = \underbrace{\overline{IDENT} \cdot \overline{NEXT ADDRESS 1}}_{\text{NEXT ADDRESS} = E_{16}} \cdot \overline{CC-OR} \cdot \overline{CK}$$

. A-BUS VALID

LOW only in a valid read/write operation. Its shape depends on the chosen pattern (dictated by F, M bits).

$$\overline{\text{A-BUS VALID}} = \overline{\text{I/O}} + \overline{\text{APROVED}} + \text{SHAPE}$$

The internal auxiliary function $\overline{\text{I/O}}$ is LOW when either a read or a write operation takes place.

$$\overline{\text{I/O}} = \overline{\text{A-ENABLE}} + \text{D-ENABLE} \cdot \text{D-REG} + \overline{\text{D-ENABLE}} \cdot \overline{\text{D-REG}}$$

SHAPE forces the desired waveform.

$$\text{SHAPE} = \text{CK} \cdot \text{A0} + \overline{\text{F}} \cdot \overline{\text{M}} \cdot \text{CK} + \overline{\text{A0}} \cdot \text{F} \cdot \text{M} + \overline{\text{CK}} \cdot \overline{\text{A0}} \cdot \text{F}$$

. D-OUT CONTR

D-OUT CK goes LOW only in a valid write operation. Depending on F, M, it follows the desired pattern. Since its positive-going edge clocks the addressed unit to store the contents of the D-BUS, it must go HI soon after the end of the clock cycle, because the data may suffer modification in the next cycle. Due to the long propagation delay in the FPLA, the logical product that produces this positive edge is performed by a 74S00, while the rest of the logic needed for the D-OUT CK is handled by the FPLA, resulting in D-OUT CONTR.

$$\text{D-OUT CK} = \overline{\text{APROVED}} + \underbrace{\overline{\text{I/O}} + \overline{\text{D-ENABLE}}}_{\text{LOW in write}} + \text{AUX}$$

The auxiliary function AUX gives the shape to the signal, according to the pattern.

$$\text{AUX} = \overline{\text{M}} \cdot \overline{\text{A0}} + \text{F} \cdot \text{M} + \text{CK}$$

$$\begin{aligned} \text{D-OUT CK} &= (\overline{\text{APROVED}} + \overline{\text{A-ENABLE}} + \overline{\text{D-ENABLE}} + \text{D-REG} + \\ &\quad \overline{\text{M}} \cdot \overline{\text{A0}} + \text{F} \cdot \text{M}) + \text{CK} \\ &= \text{D-OUT CONTR} + \text{CK} \end{aligned}$$

. D-IN CONTR

D-IN CK goes LOW only in a valid read operation, or in a D-Register loading with the contents of internal working register AC, or in a ICU read. For similar reasons as above, the critical product occurs in a 74S00 and the rest of the logic results in D-IN CONTR.

$$\begin{aligned} \text{D-IN CK} &= (\overline{\text{APROVED}} + \overline{\text{I/O}} + \overline{\text{D-REG}} + \text{AUX}) \cdot (\overline{\text{SWAP}} + \text{AUX}) \cdot \\ &\quad \cdot (\overline{\text{ICUR}} + \text{AUX}) \end{aligned}$$

where $\overline{\text{SWAP}} = \text{A-ENABLE} + \overline{\text{D-ENABLE}} + \overline{\text{D-REG}}$

Fumbling with the formula, we get:

$$\begin{aligned} \text{D-IN CK} &= \overline{\text{APROVED}} + (\overline{\text{A-ENABLE.D-ENABLE}} + \\ &+ \text{A-ENABLE.D-ENABLE}).(\overline{\text{IDENT}} + \text{NEXT ADDR.1} + \overline{\text{CC-OR}}) + \\ &+ \overline{\text{D-REG}} + \overline{\text{M.A0}} + \text{F.M} + \text{CK} \\ \text{D-IN CK} &= \text{D-IN CONTR} + \text{CK} \end{aligned}$$

The contents of the FPLA are shown in the next page. Due to corrections introduced, it is not optimally utilized, but it is logically correct.

INPUTS

OUTPUTS

NEXT ADDR. 1
 CC-OR
 IDENT
 L0
 L1
 L2
 P/M
 MAP
 A-ENABLE
 D-ENABLE
 D-REG
 M
 F
 A0
 CK

ICW
 ICUR
 I/O VALID
 L-LOAD
 A-BUS VALID
 D-OUT CONTR
 D-IN CONTR

NO.	PRODUCT TERM*															
	INPUT VARIABLE															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				L	L	L										
1				L		H										
2																
3																
4																
5				L						H						
6												L				
7												H	H			
8			L													
9	H															
10		H														
11			L													
12	H															
13		H														
14																
15																
16																
17																
18																
19																
20															L	H
21																
22																
23																
24	H	L	H													L
25																
26																
27																
28																
29																
30																
31	L	L	H													
32	L	L	H													H
33																
34																
35																
36																
37																
38																
39																
40																
41																
42																
43																
44																
45																
46																
47																

NO.	ACTIVE LEVEL								
	OUTPUT FUNCTION								
	7	6	5	4	3	2	1	0	
0								A	A
1								A	A
2								A	A
3								A	A
4									
5								A	A
6								A	A
7								A	A
8									
9									
10									
11									
12									
13									
14									
15									
16								A	A
17									
18								A	A
19								A	A
20								A	A
21								A	A
22								A	A
23								A	A
24									
25									
26									
27									
28									
29									
30									
31									
32									
33									
34									
35									
36									
37									
38									
39									
40									
41									
42									
43									
44									
45									
46									
47									

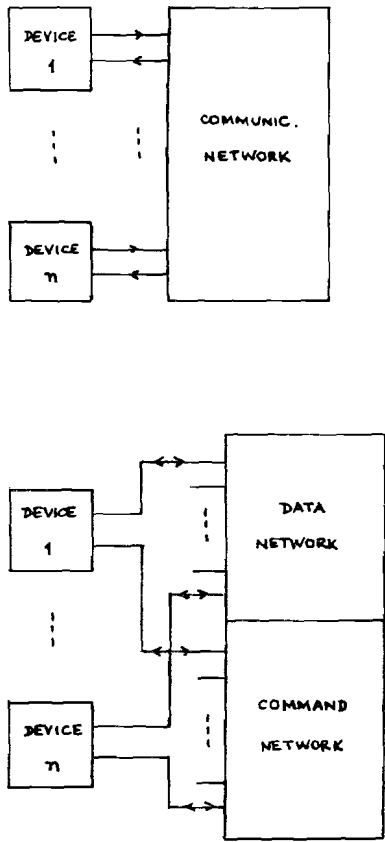


FIG.1 - COMMUNICATION NETWORK

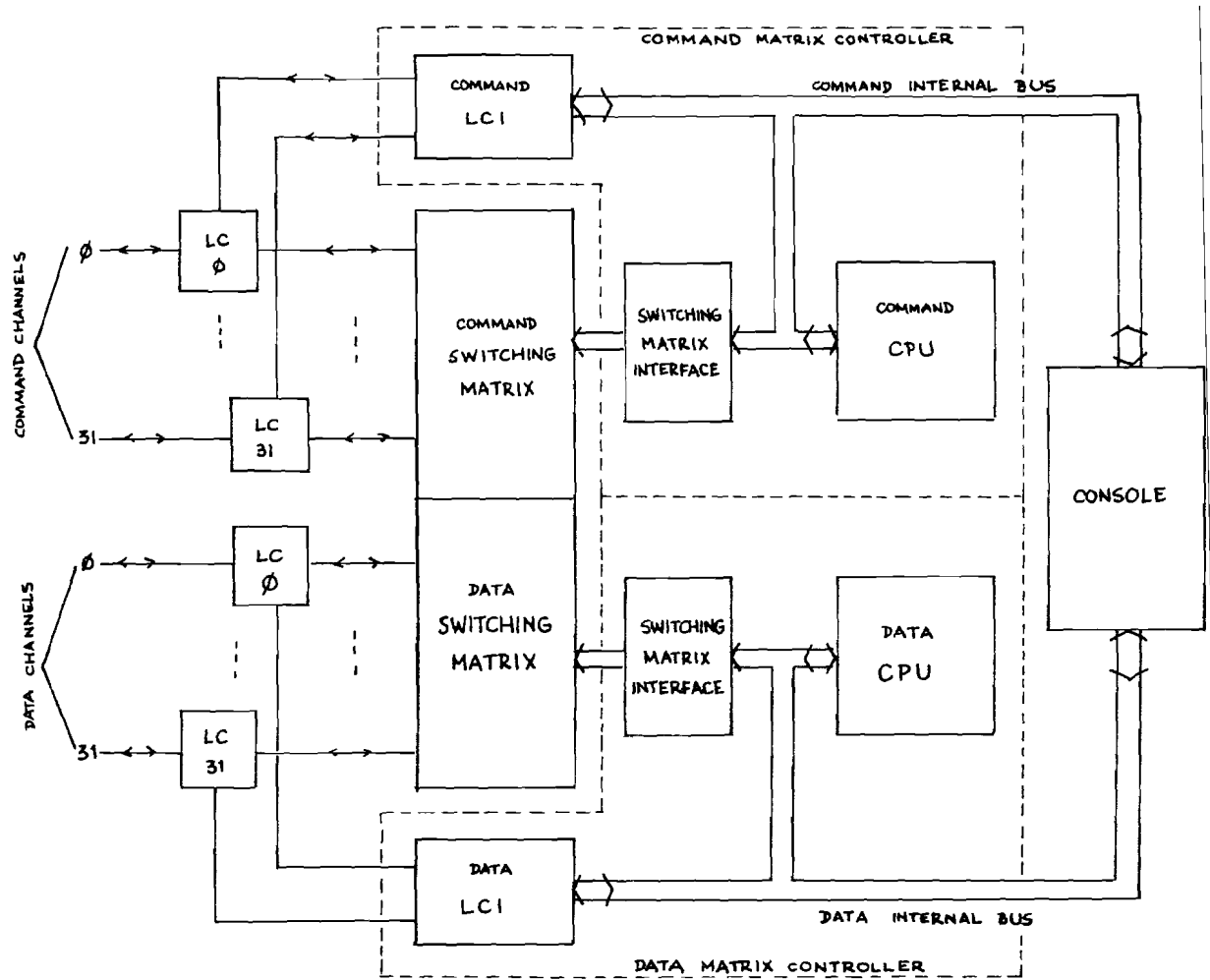


FIG.2 - MATRIX CONTROLLER (BLOCK DIAGRAM)

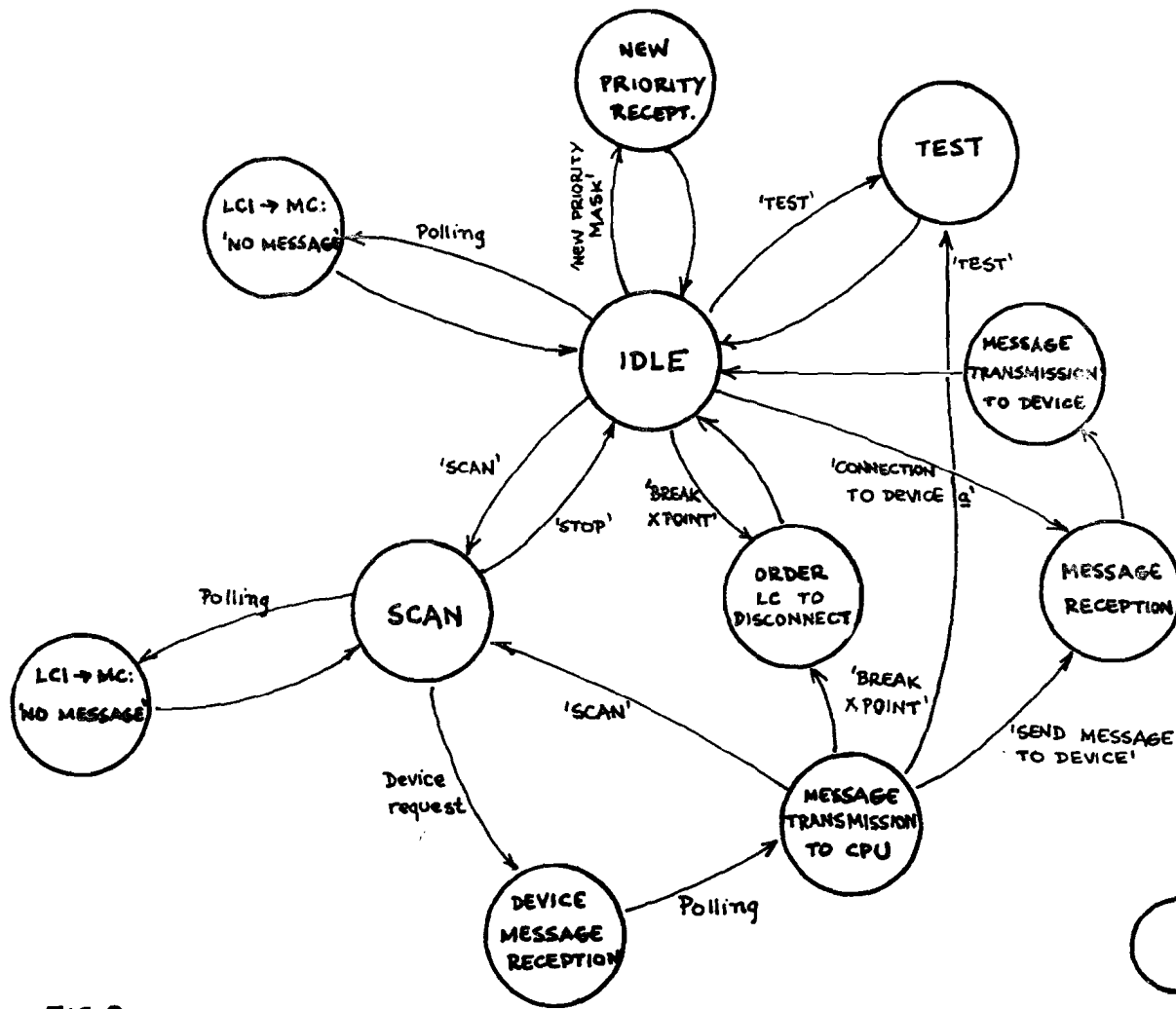


FIG. 3- LCI STATE DIAGRAM

- 'STOP' COMMAND ALWAYS FORCES LCI TO GO 'IDLE'. IT WAS OMITTED IN THE DIAGRAM FOR SIMPLICITY'S SAKE.
- TRANSITIONS WITHOUT NAME OCCUR AUTOMATICALLY AFTER TASK IS COMPLETED
- COMMANDS NOT ALLOWED FOR THE LCI STATE SHOULD CAUSE A LCI INTERRUPT REQUEST.

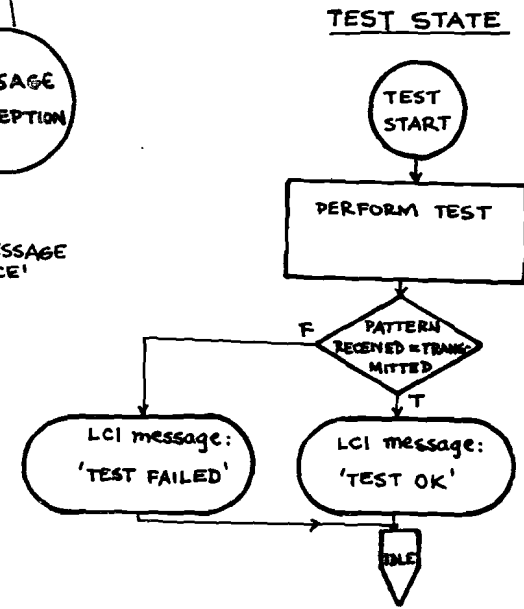
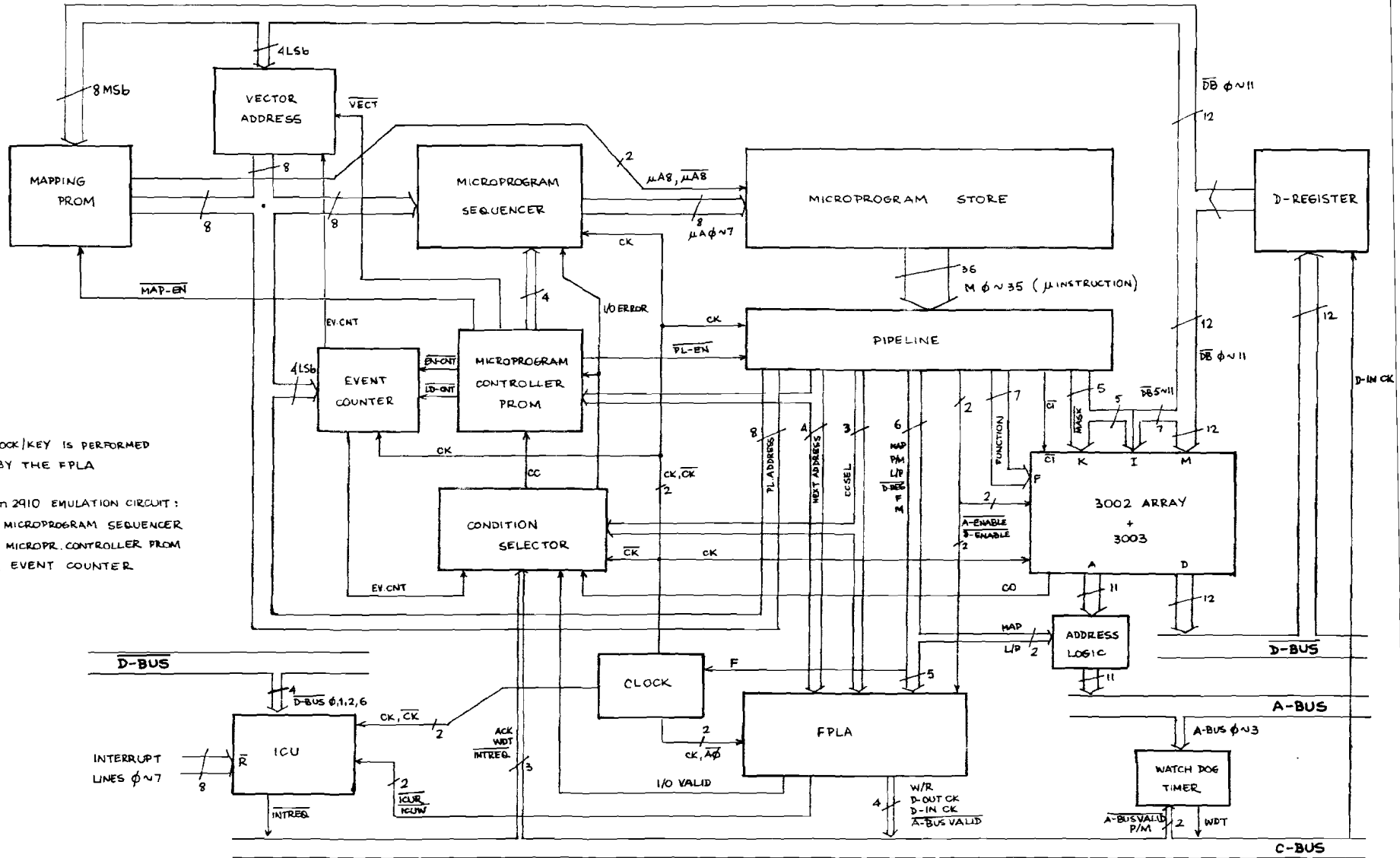


FIG. 4 - MATRIX CONTROLLER

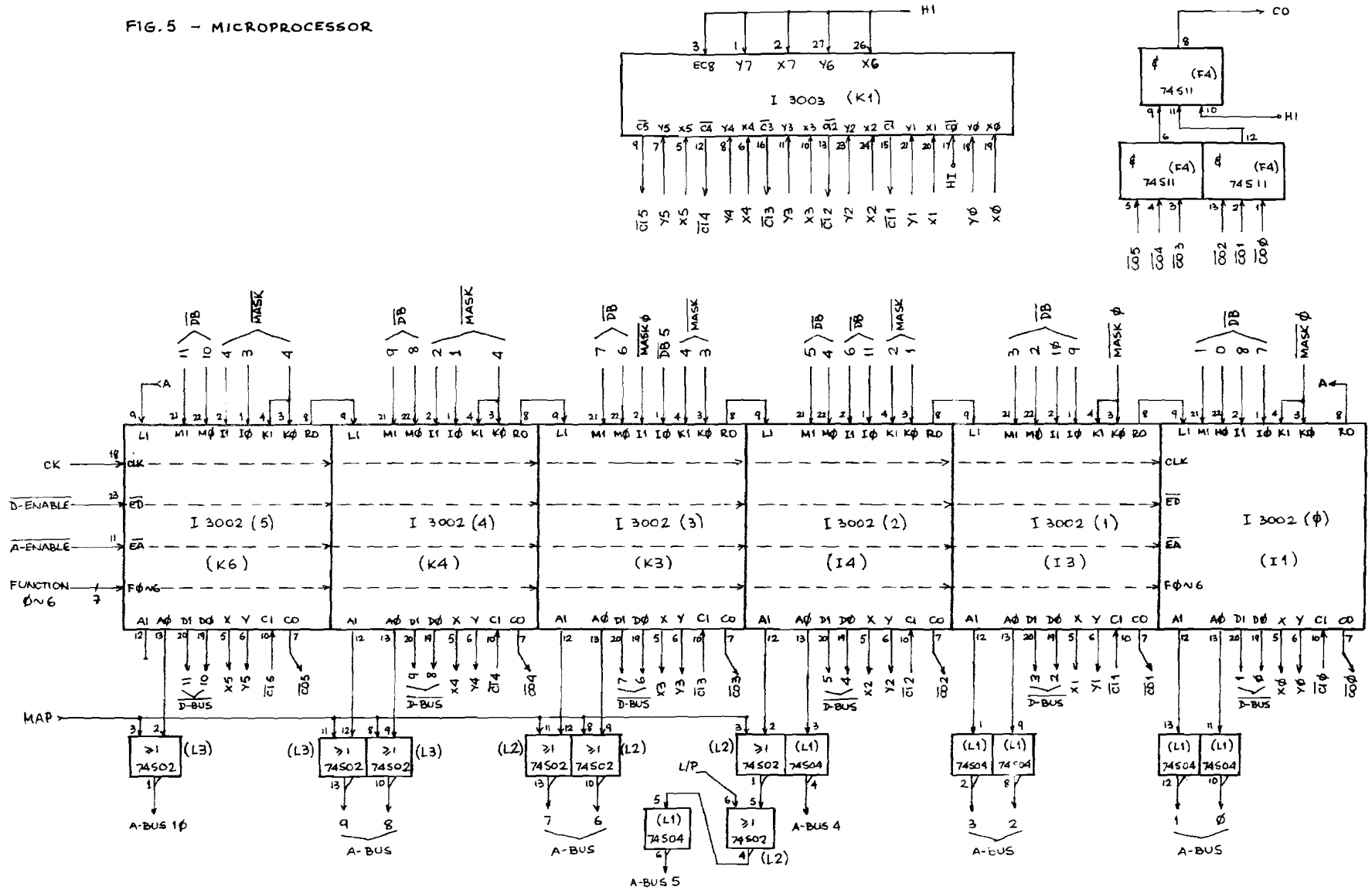


71

- LOCK/KEY IS PERFORMED BY THE FPLA
- AM 2910 EMULATION CIRCUIT:
 - MICROPROGRAM SEQUENCER
 - MICROPR. CONTROLLER PROM
 - EVENT COUNTER

FIG. 5 - MICROPROCESSOR

72



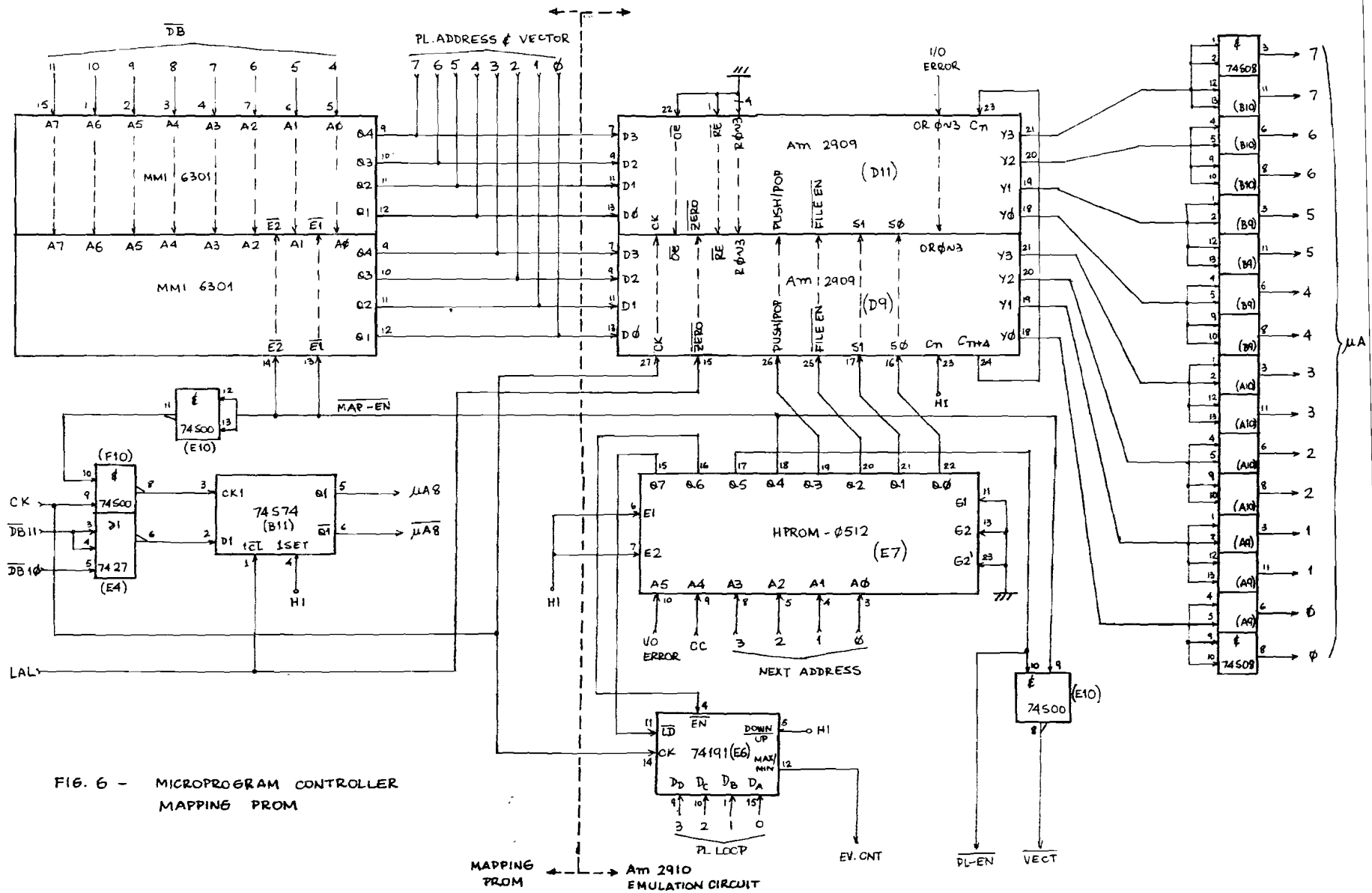
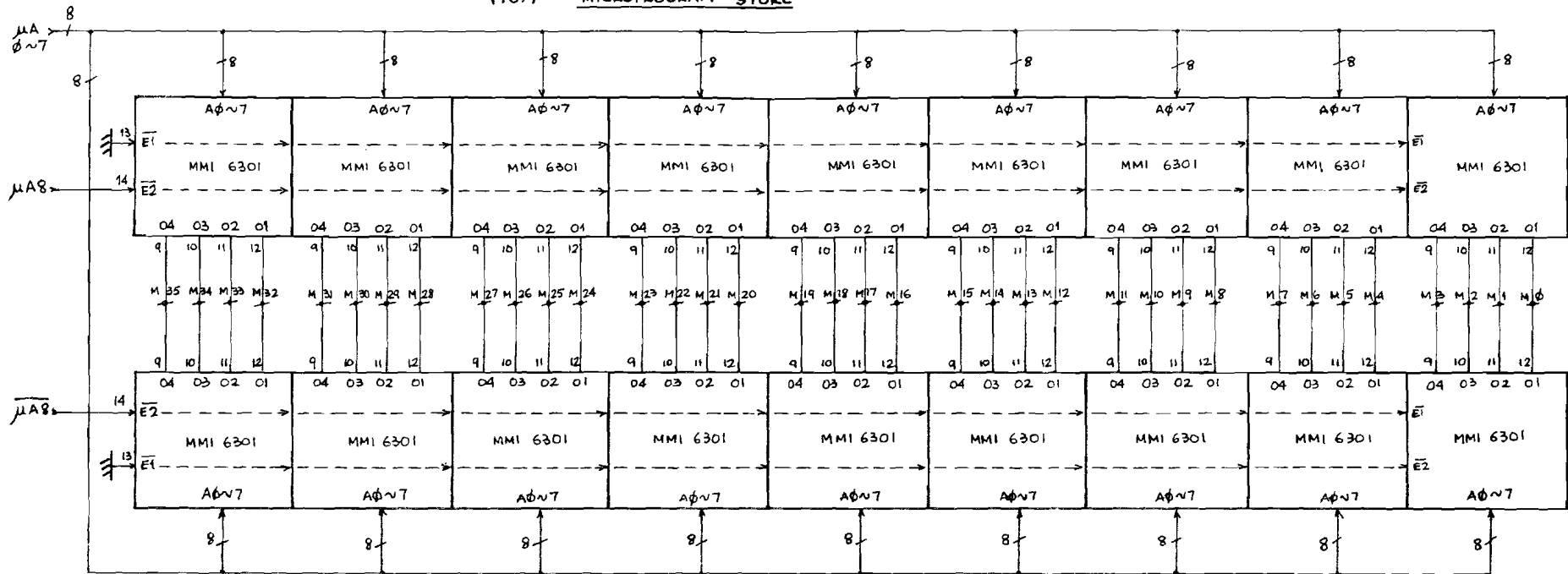


FIG. 6 - MICROPROGRAM CONTROLLER MAPPING PROM

FIG. 7 - MICROPROGRAM STORE



74

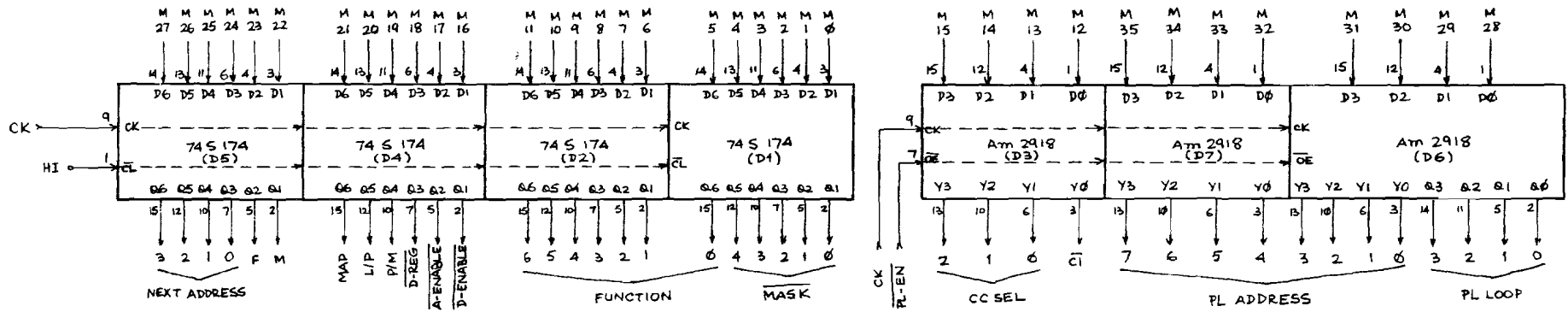


FIG. 8 - PIPELINE

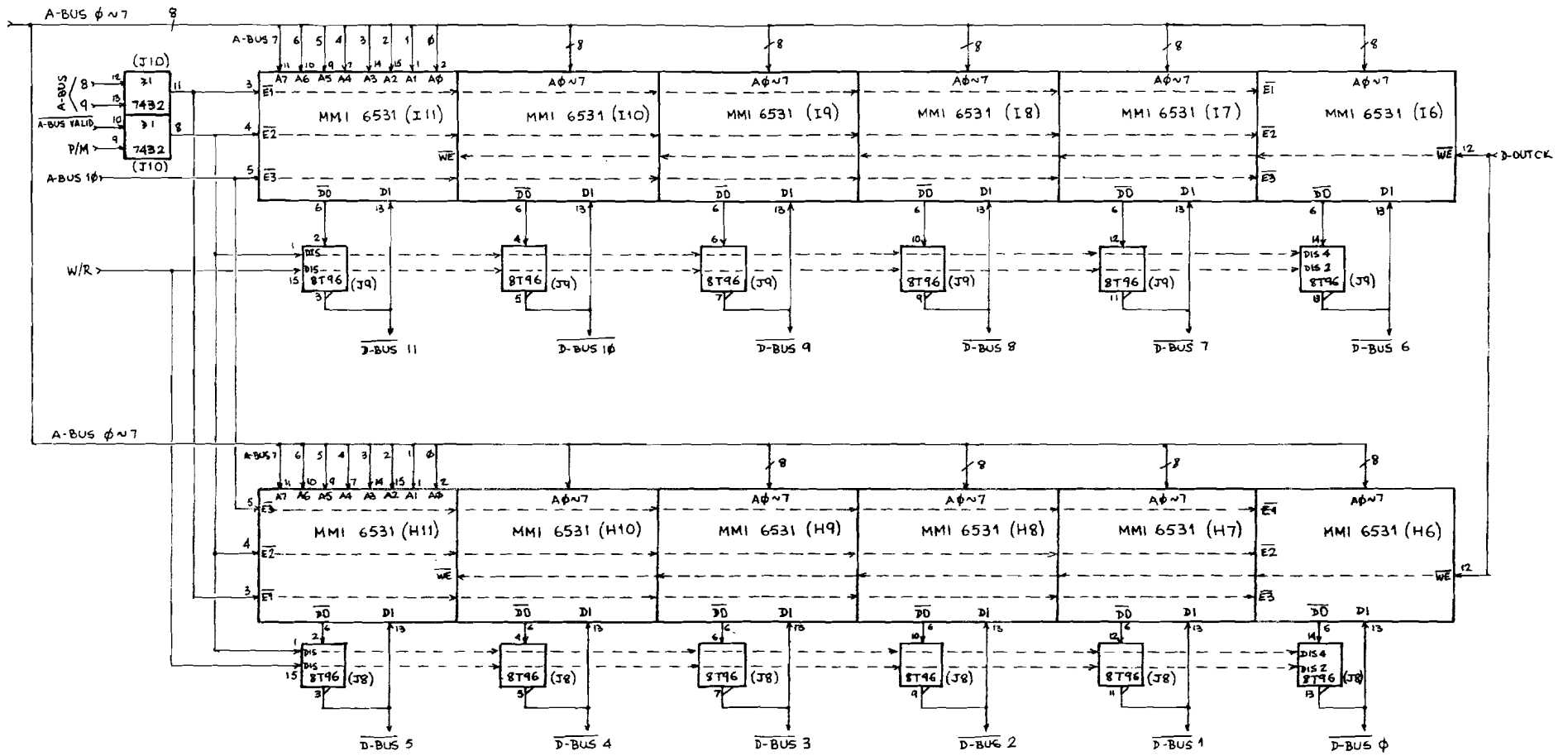


FIG. 9 - MEMORY (ONLY THE FAST SECTION)

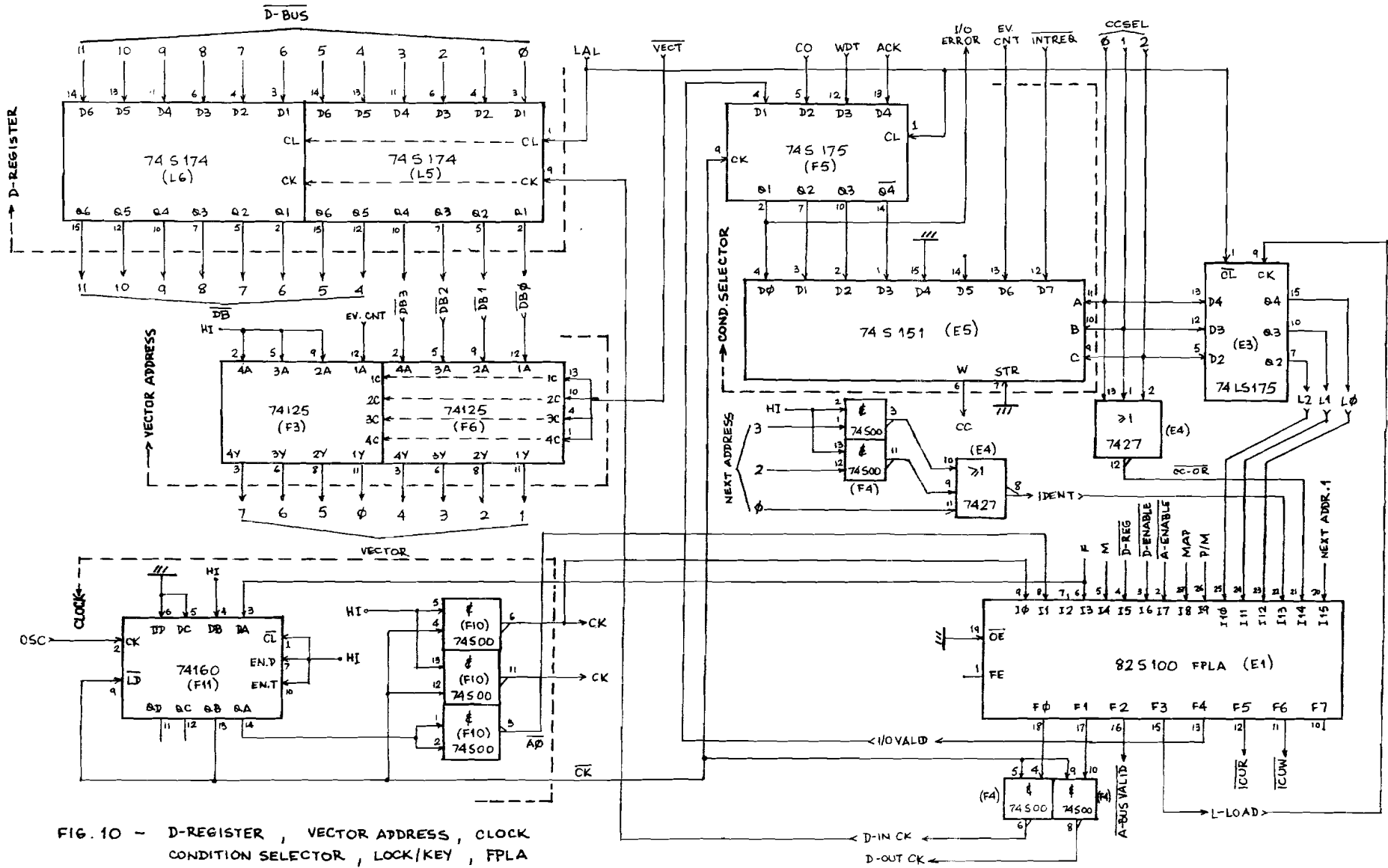
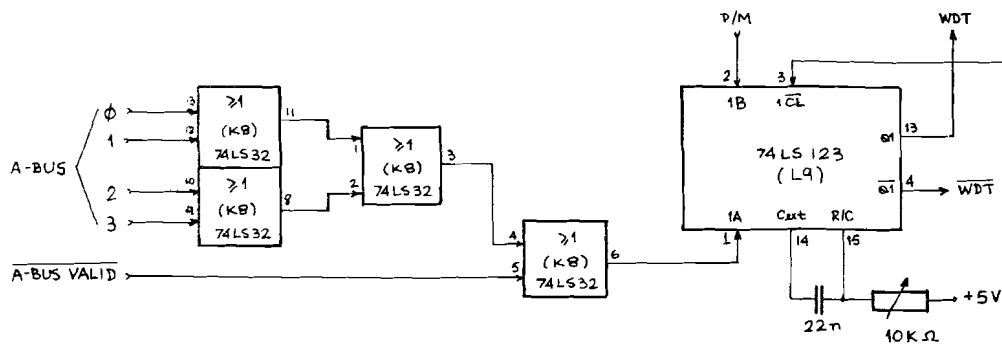
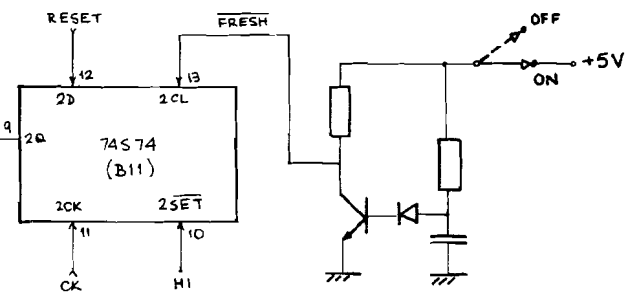


FIG. 10 - D-REGISTER , VECTOR ADDRESS , CLOCK
CONDITION SELECTOR , LOCK/KEY , FPLA

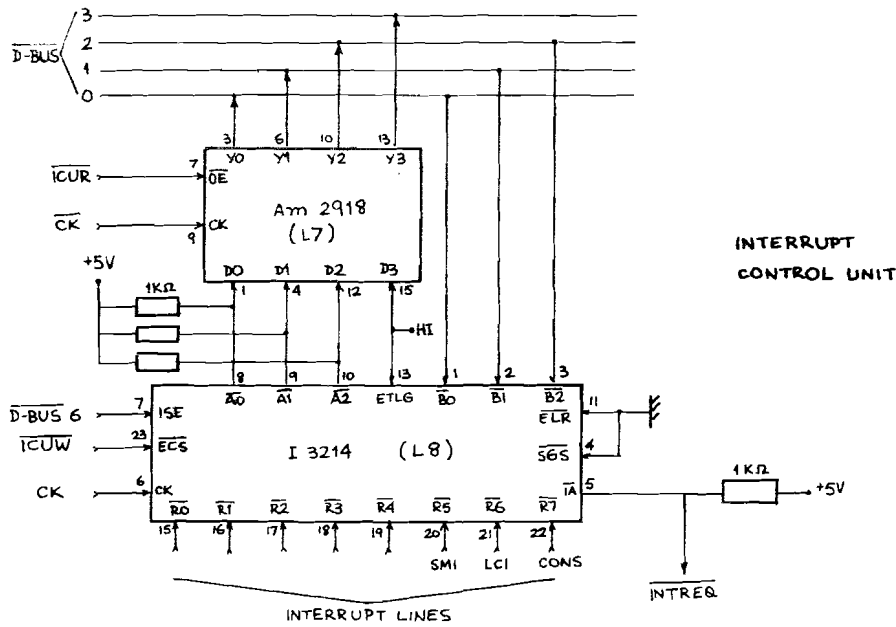


WATCH DOG TIMER



POWER-ON RESET CIRCUIT

77



INTERRUPT CONTROL UNIT

FIG. 11 - WDT , ICU , POWER-ON RESET CIRCUIT

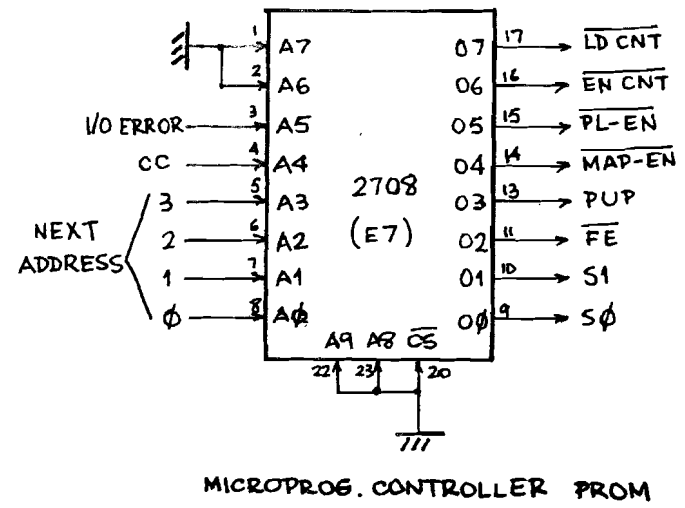
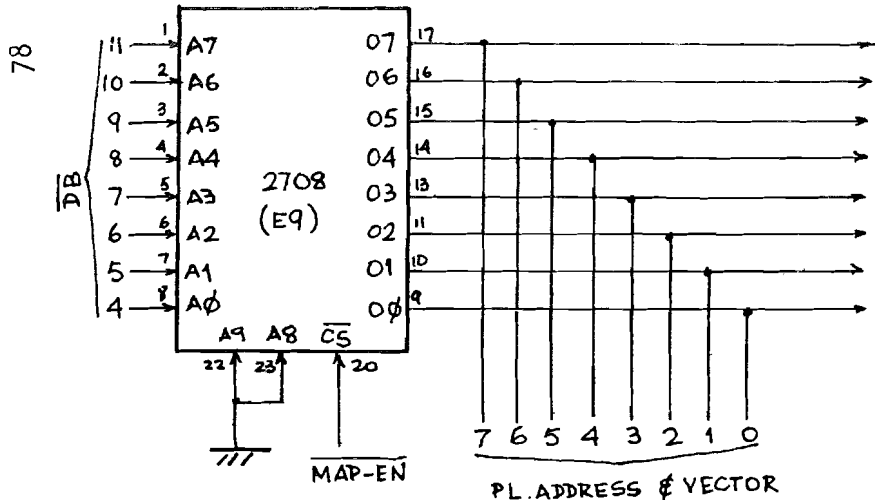
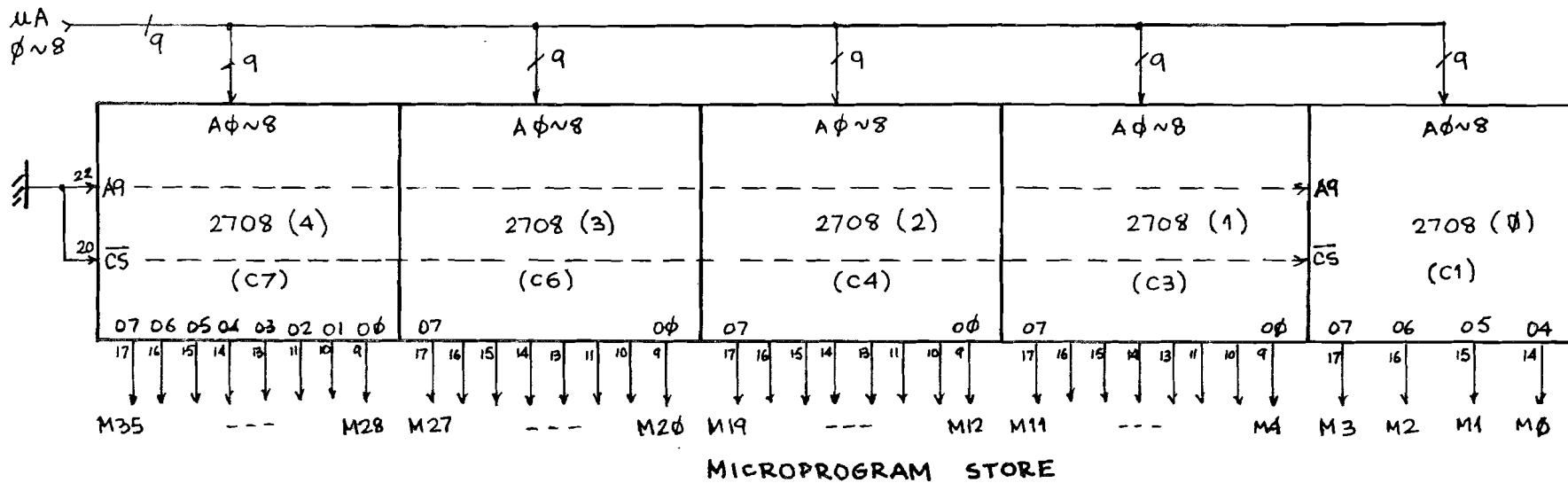


FIG. 12 - EPROM'S USED IN THE PROTOTYPE

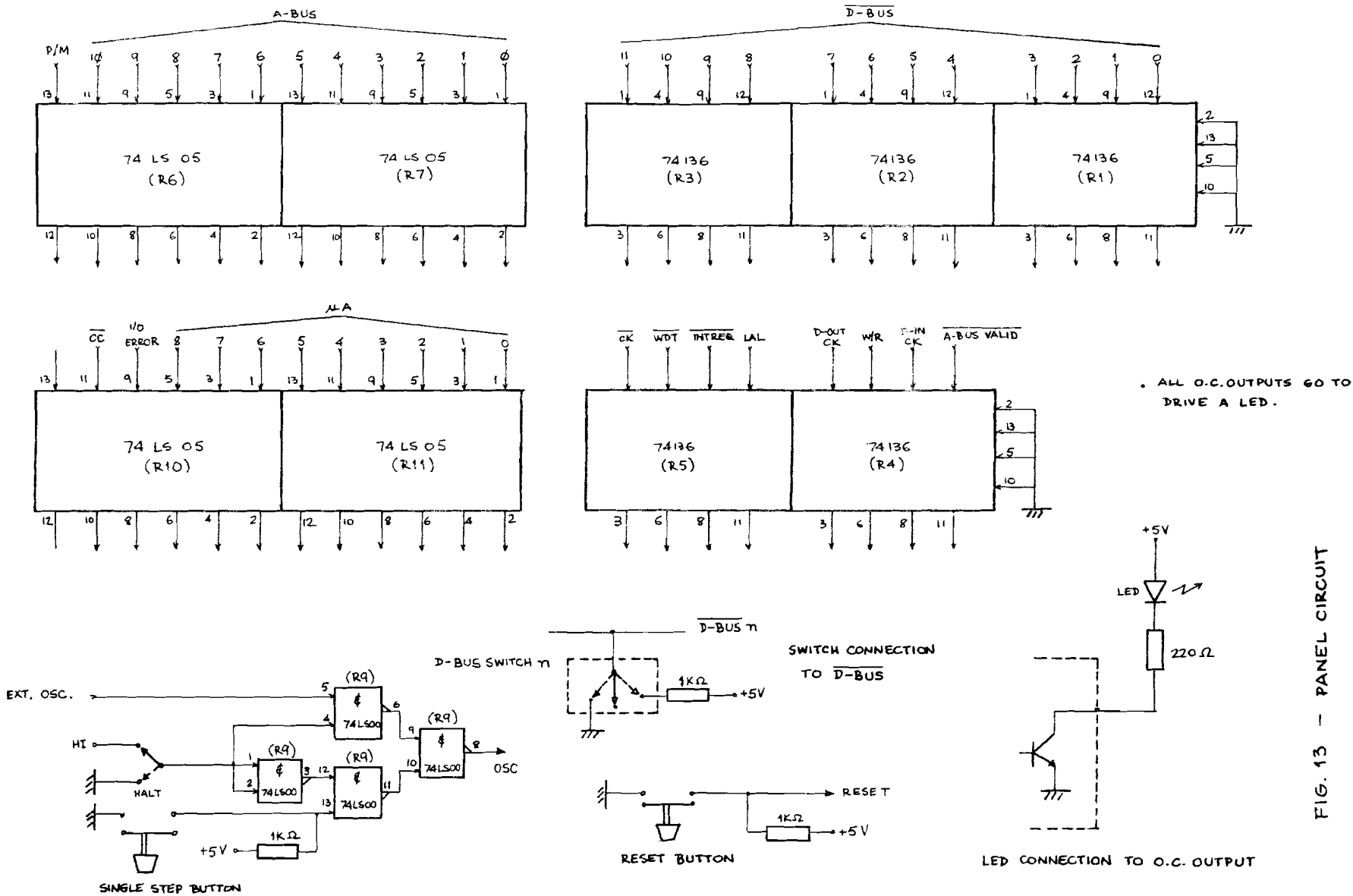


FIG. 13 - PANEL CIRCUIT

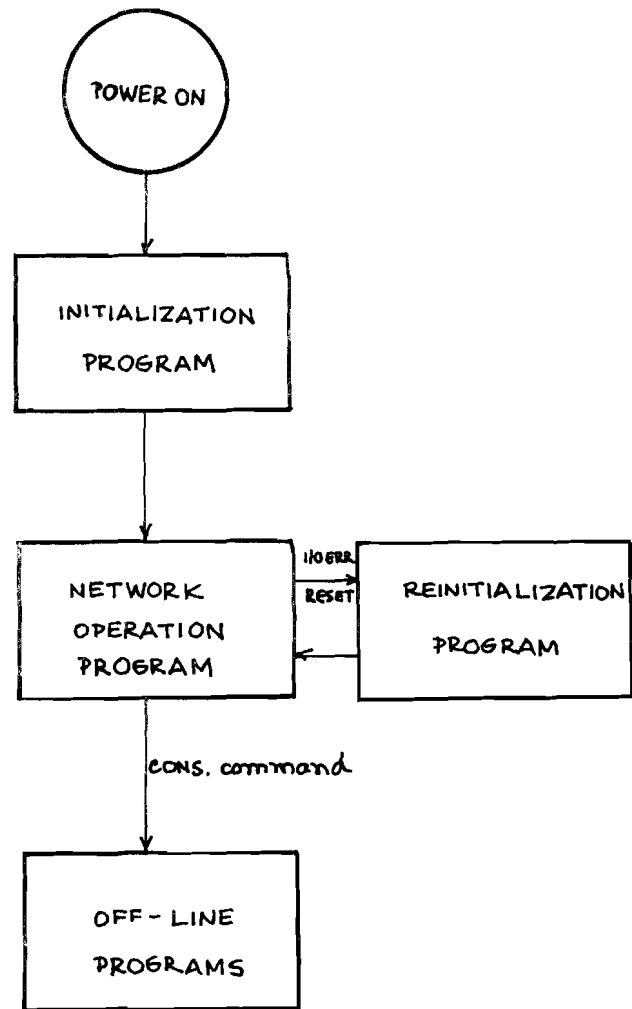


FIG. 14 - SYSTEM

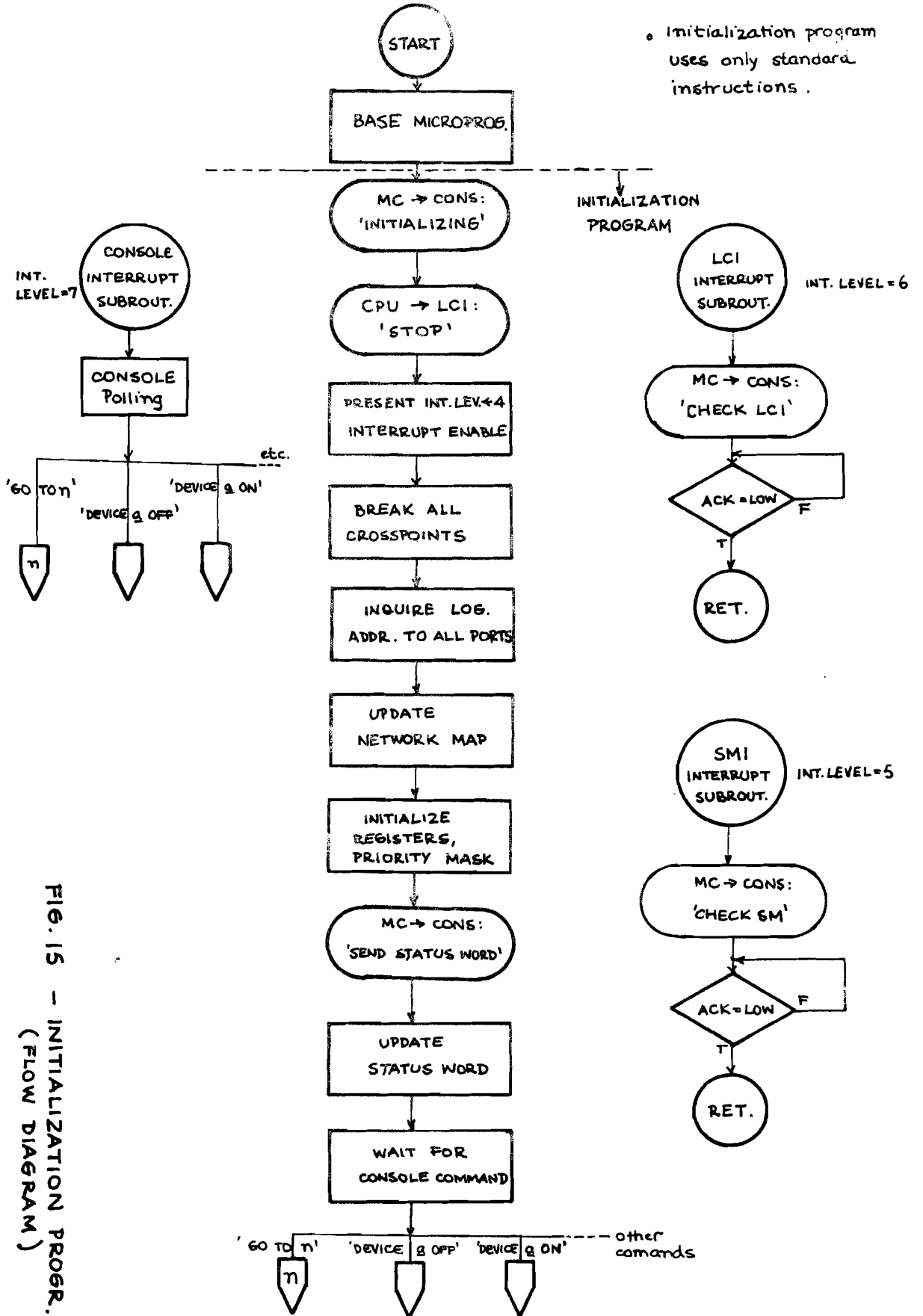
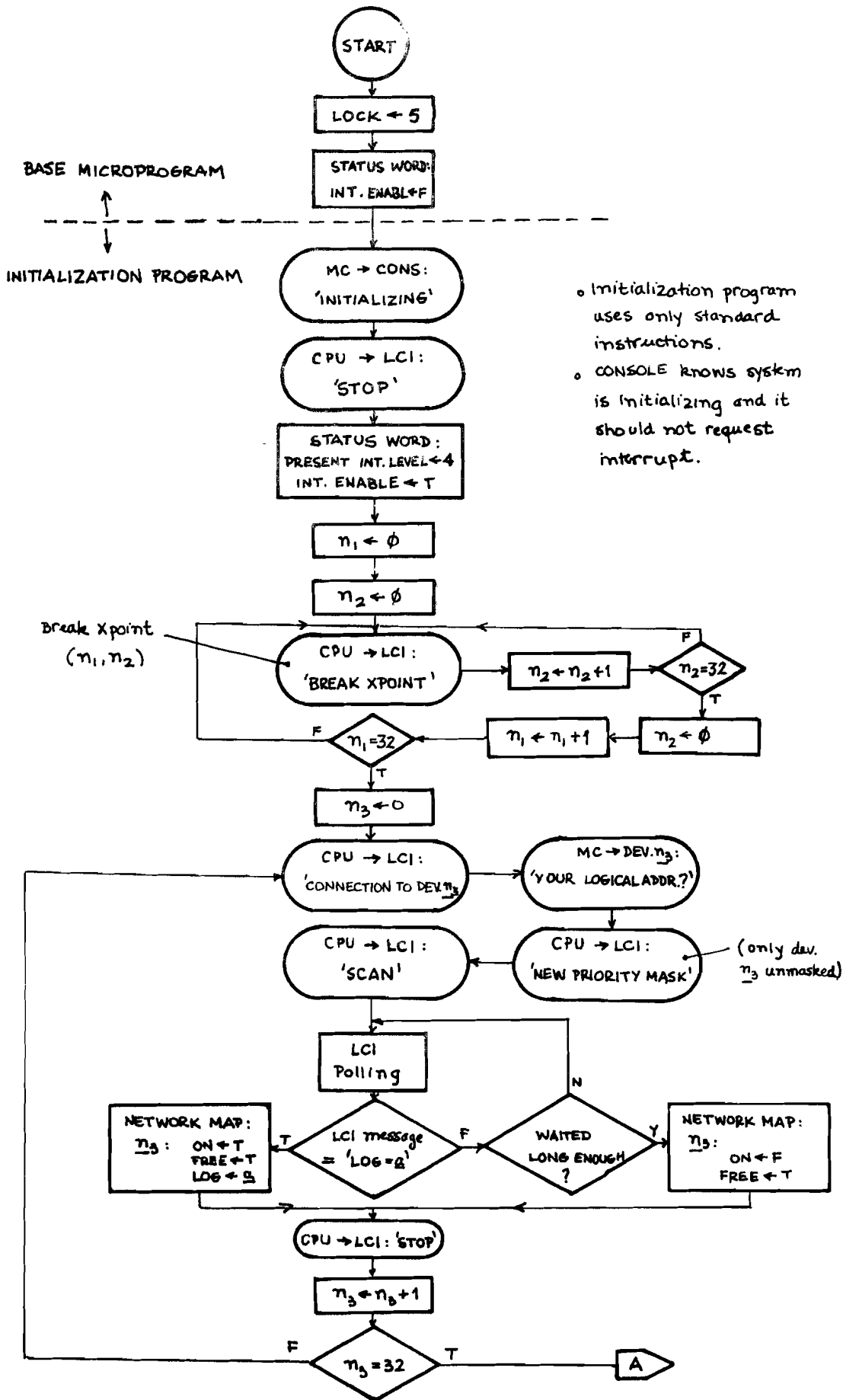


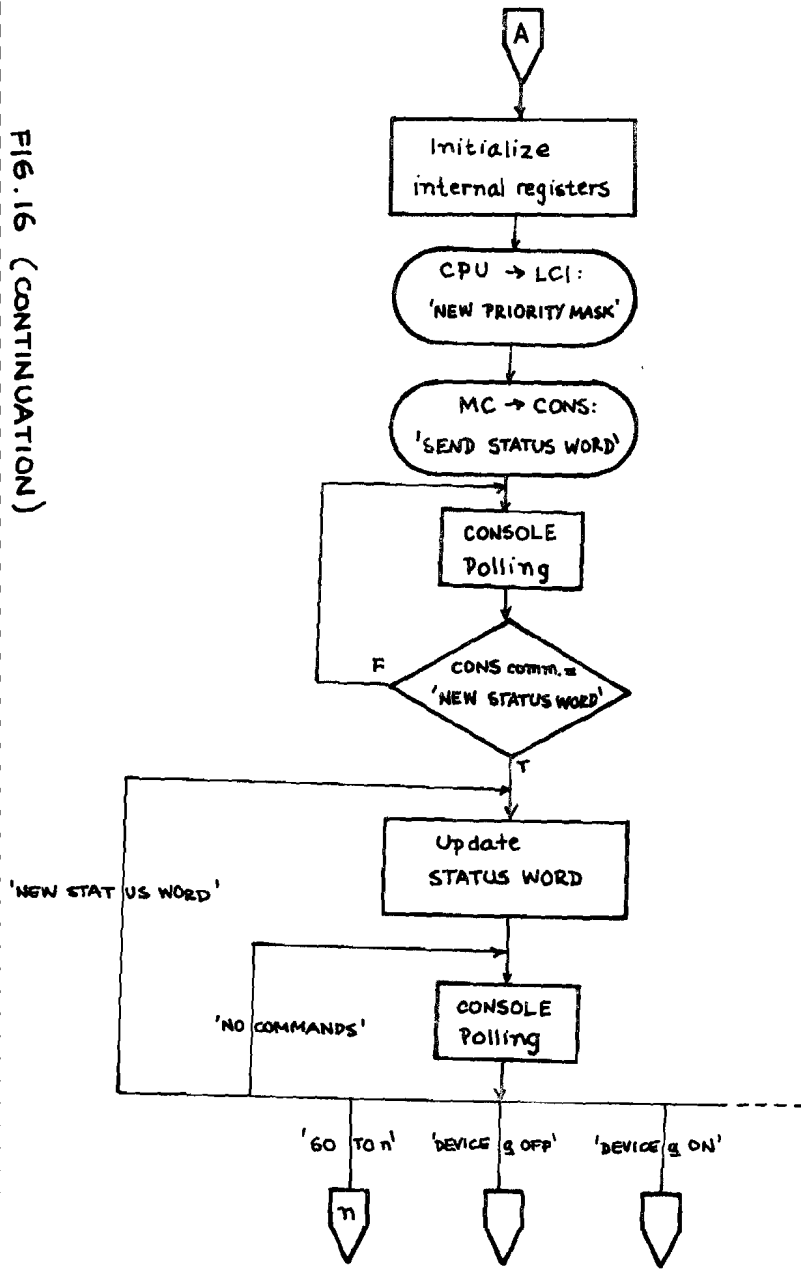
FIG. 15 - INITIALIZATION PROGR.
(FLOW DIAGRAM)

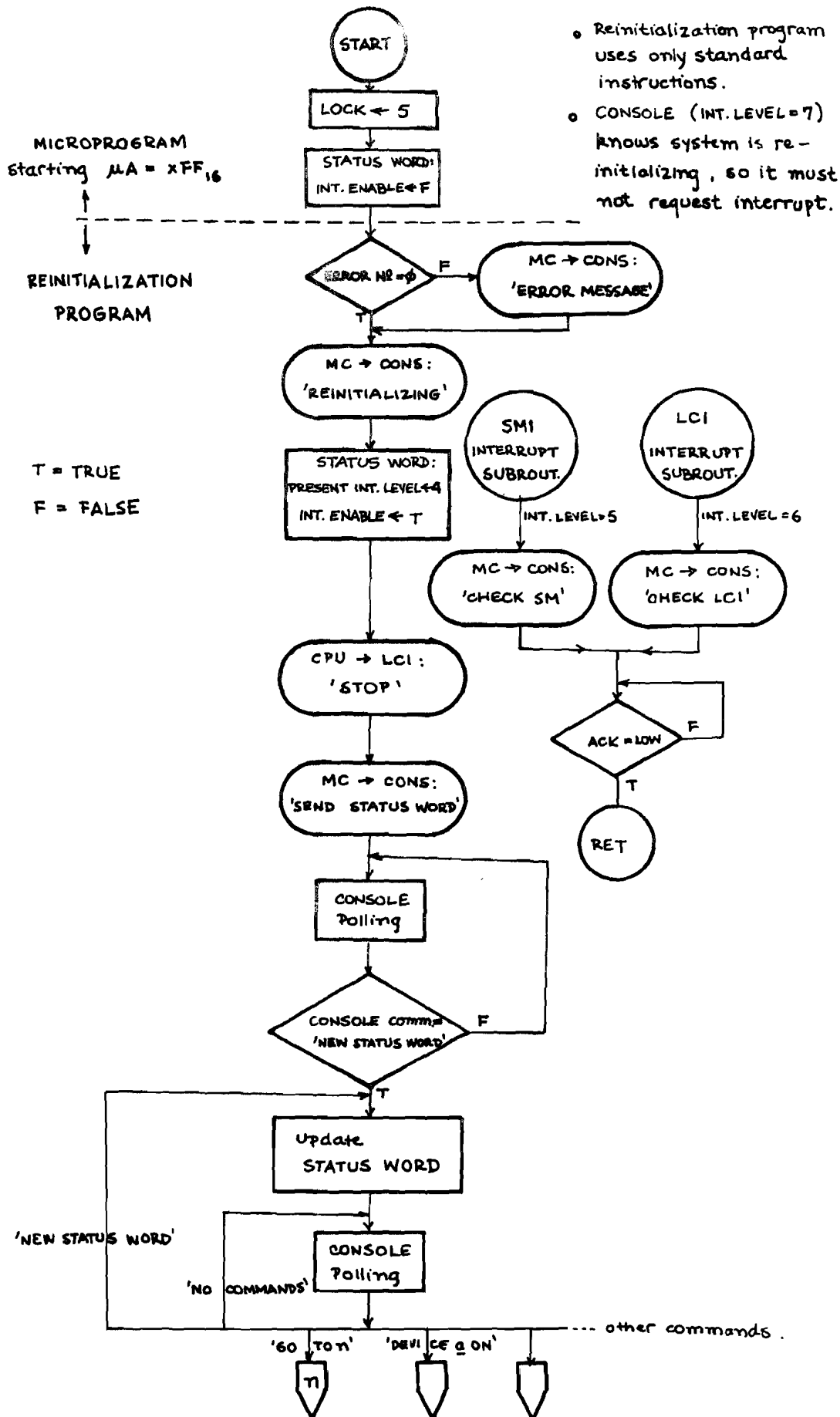


- Initialization program uses only standard instructions.
- CONSOLE knows system is initializing and it should not request interrupt.

FIG. 16 - INITIALIZATION PROGR. (DETAILED)

FIG. 16 (CONTINUATION)





- Reinitialization program uses only standard instructions.
- CONSOLE (INT. LEVEL = 7) knows system is re-initializing, so it must not request interrupt.

FIG. 17 - REINITIALIZATION PROGRAM

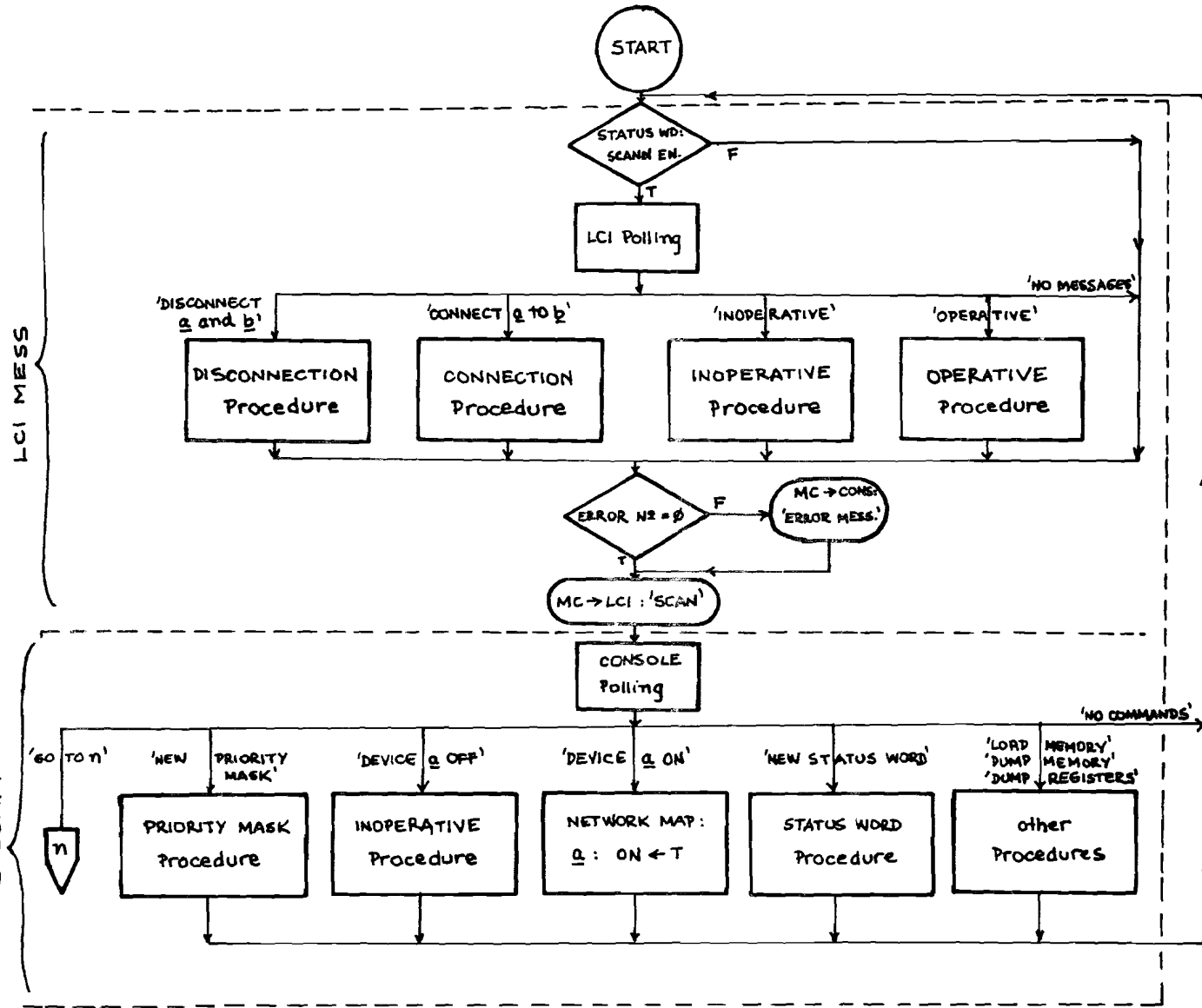
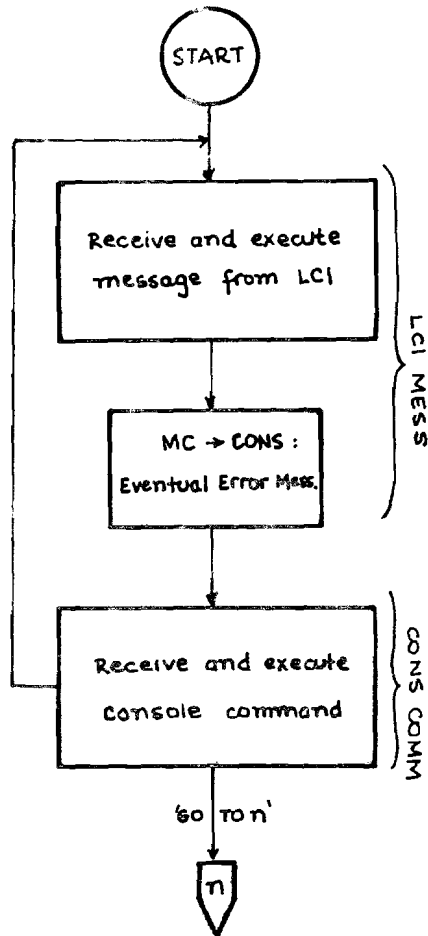
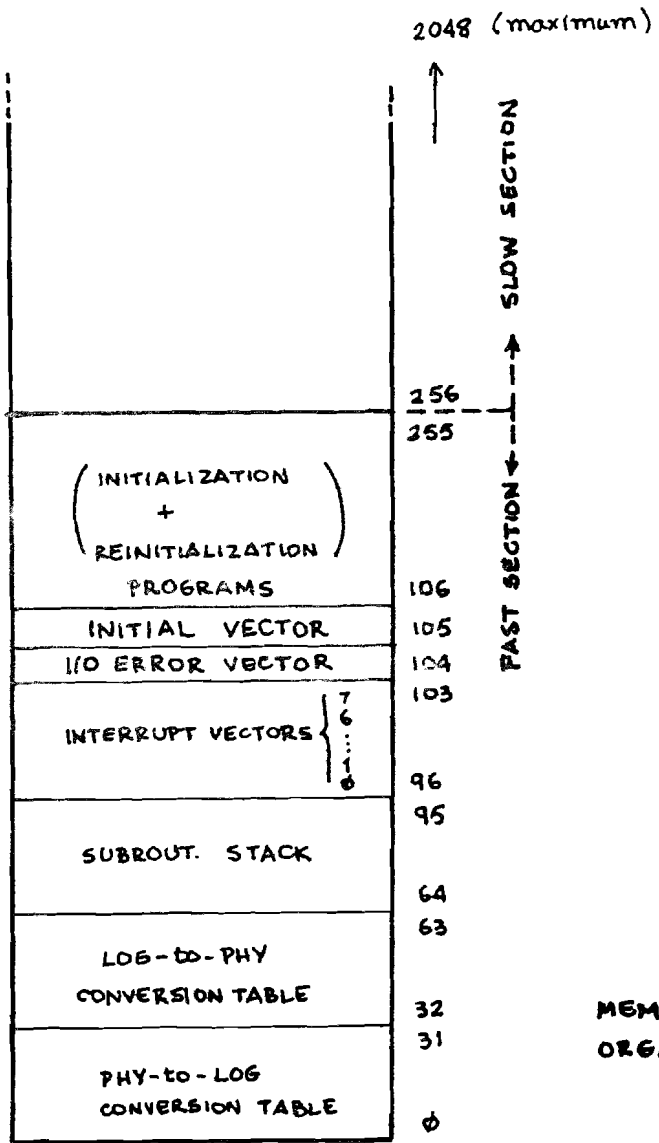
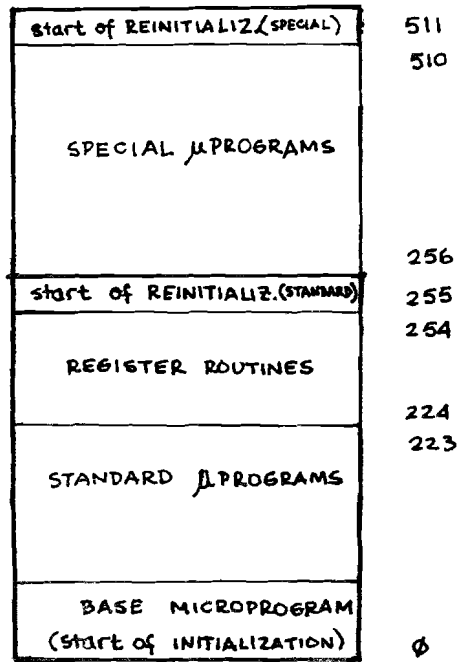


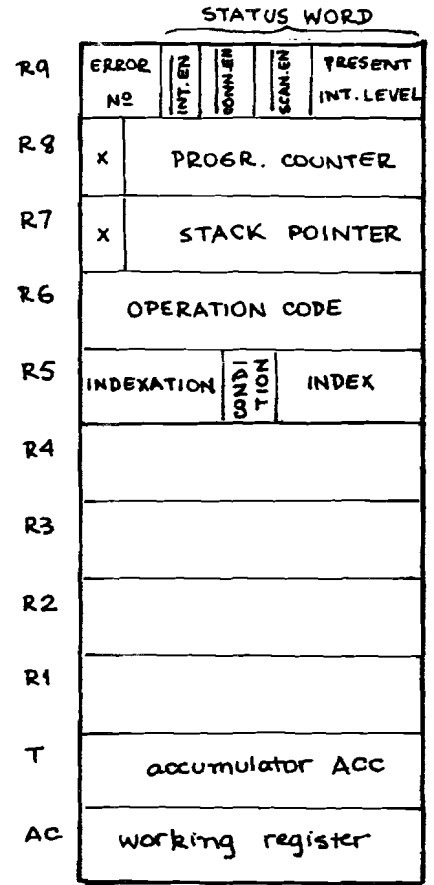
FIG.18 - NETWORK OPERATION PROGRAM



MEMORY ORGANIZATION



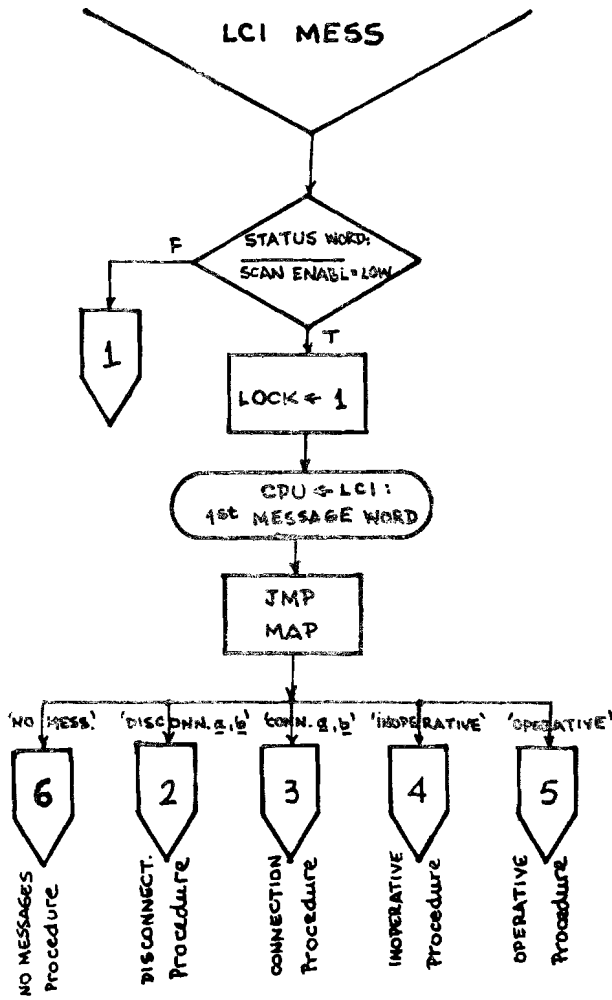
MICROPROGRAM STORE ORGANIZATION




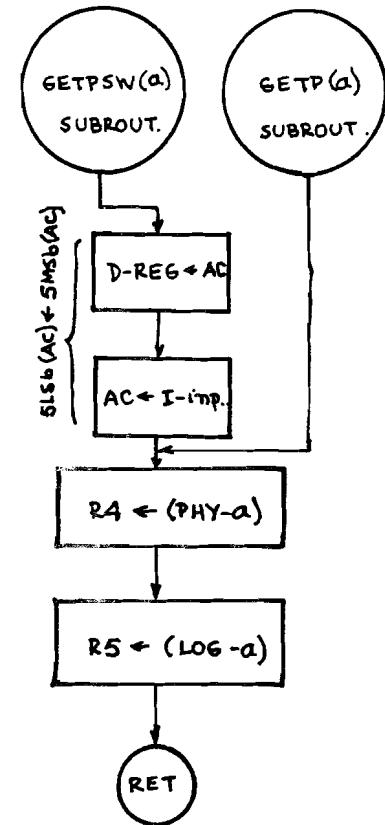
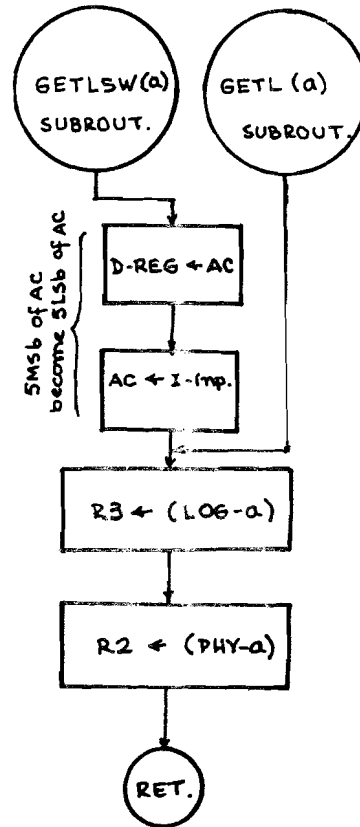
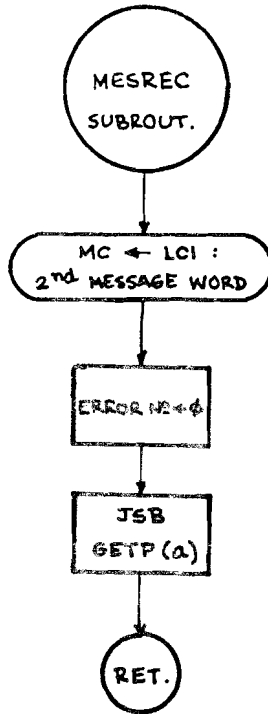
INTERNAL REGISTERS

FIG. 19

FIG. 20 - LCI MESS



For all flow diagrams,  ≡ GO TO "NEXT INSTRUCTION" μROUTINE (for special instruction)



GETLSW(a) / GETPSW(a) : AC should come with LOG/PHY address of desired device in the 5MSb
 GETL(a) / GETP(a) : LOG/PHY address of desired device in the 5LSb of AC.

• LCI MESS Instruction

- IF SCAN. ENABLE = HI (SCAN. ENABLE IS THE BIT 4 OF THE STATUS WORD)
- THEN
 - LOCK ← 1 (NETWORK MAP, SMI, LCI may be read and written)
 - 1st MESSAGE WORD FROM LCI IS READ AND A JUMP TO THE APPROPRIATE PROCEDURE OCCURS
- AC IS ZEROED

• MESREC Subroutine

- READS FROM LCI THE 2nd MESSAGE WORD
- STATUS WORD : ERROR NUMBER ← ϕ
- REGISTERS :
 - R4 ← (PHY-a)
 - R5 ← (LOG-a)
- AC = ϕ IS ASSUMED AT THE START OF THE SUBROUTINE

88

• GETL(a), GETLSW(a) Subroutines

- REGISTER AC SHOULD CONTAIN LOG-a
 - IN THE 5 LSB (for GETL(a) subroutine)
 - IN THE 5 MSB (" GETLSW(a) ")
- REGISTERS :
 - R2 ← (PHY-a)
 - R3 ← (LOG-a)

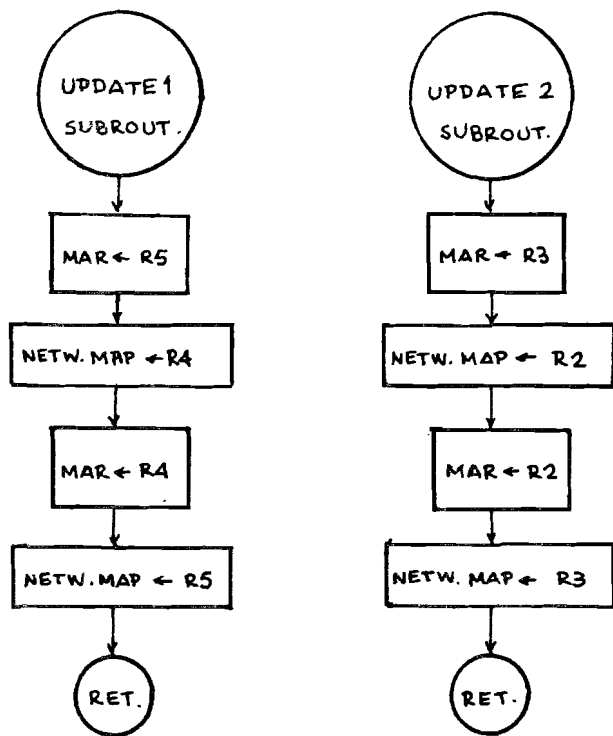
• GETP(a), GETPSW(a) Subroutines

- REGISTER AC SHOULD CONTAIN PHY-a
 - IN THE 5 LSB (for GETP(a) subroutine)
 - IN THE 5 MSB (" GETPSW(a) ")
- REGISTERS :
 - R4 ← (PHY-a)
 - R5 ← (LOG-a)

FIG. 20 a - LCI MESS

μA	PL. ADDRESS	NEXT ADDRESS	F	M	MAP	L/P	P/M	D-REG	A-EN	D-EN	CC. SEL	CI	FUNCTION	MASK	COMMENTS
LCI MESS Instr. →	-	CONTINUE	1	-	0	0	-	1	1	1	0	1	0 1 (R9)	11111	AC ← R9
	NEXT INST. FET	COND JMP PL	0	-	0	0	-	1	1	1	1	1	5 1 1 1 (AC)	11101	AC, CO ← SCAN.ENABLE
	-	CONTINUE	0	0	1	0	1	0	0	1	1	1	1 1 (AC)	11111	LOCK ← 1 ; MAR ← 0 (LCI addr); D-REG ← 1 st Mess.word
MESREC SUB →	0	JMP MAP	1	-	0	0	-	1	1	1	0	1	5 1 1 1 (AC)	11111	Go to proper procedure; AC ← 0
	-	CONTINUE	0	0	1	0	1	0	0	1	0	1	1 1 (AC)	11111	MAR ← AC(=0) (LCI addr.) D-REG ← 2 nd Mess.word
	-	CONTINUE	1	-	0	0	-	1	1	1	0	0	1 1 1 (AC)	00000	AC ← 2 nd Mess. word
	GETP (a)	COND JSB PL	1	-	0	0	-	1	1	1	4	0	2 1 (R6)	00000	R6 ← 2 nd Mess. word
	-	COND RTN	1	-	0	0	-	1	1	1	4	1	5 1 (R9)	10000	ERROR N ^o ← 0
	-	CONTINUE-	1	0	0	0	-	0	1	0	0	0	2 1 (AC)	00000	D-REG ← AC
	-	CONTINUE	1	-	0	0	-	1	1	1	0	0	2 1 1 1 (AC)	00000	AC is swapped (5LSb - 5MSb)
	-	CONTINUE	0	0	1	1	0	0	0	1	0	1	1 1 (AC)	11111	MAR ← AC; D-REG ← (LOG-a)
	-	CONTINUE	1	-	0	0	-	1	1	1	0	1	1 1 1 1 (AC)	11111	AC, MAR ← (LOG-a)
	-	CONTINUE	1	0	1	0	0	0	0	1	0	0	2 1 (R3)	00000	R3 ← (LOG-a); D-REG ← (PHY-a)
GETL (a) Subr. →	-	CONTINUE	1	-	0	0	-	1	1	1	0	1	1 1 1 1 (AC)	11111	AC ← (PHY-a)
	-	COND RTN	1	-	0	0	-	1	1	1	4	0	2 1 (R2)	00000	R2 ← (PHY-a)

FIG. 20b - LCI MESS μ PROGRAMS



UPDATE 1 / UPDATE 2 suppose (LOG-a) is in register R5/R3
 (PHY-a) " " " R4/R2
 and stores (PHY-a) and (LOG-a) into NETWORK MAP

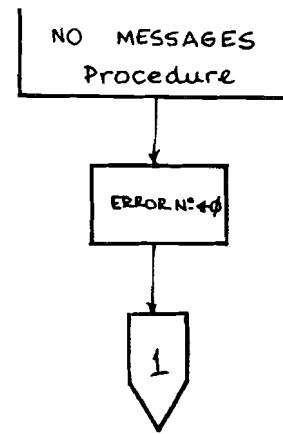


FIG. 21 - LCI MESS

- UPDATE 1 , UPDATE 2 Subroutines

- IT IS ASSUMED THAT : { (LOG-a) IS IN REGISTER R5 AND (PHY-a) IN R4 FOR SUBROUT. UPDATE1
 " " " " R3 " " " R2 " " UPDATE2

- NETWORK MAP IS UPDATED :

- (LOG-a) and (PHY-a) are updated with the new data contained in R5 and R4 (UPDATE1)
 " " " " " " " " " " R3 and R3 (UPDATE 2)

- NO MESSAGE Procedure

- STATUS WORD : ERROR NUMBER ← ϕ

FIG. 21a - LCI MESS

Below the dashed line, is the routine that serves eventual waiting devices that may now talk with the devices just disconnected.

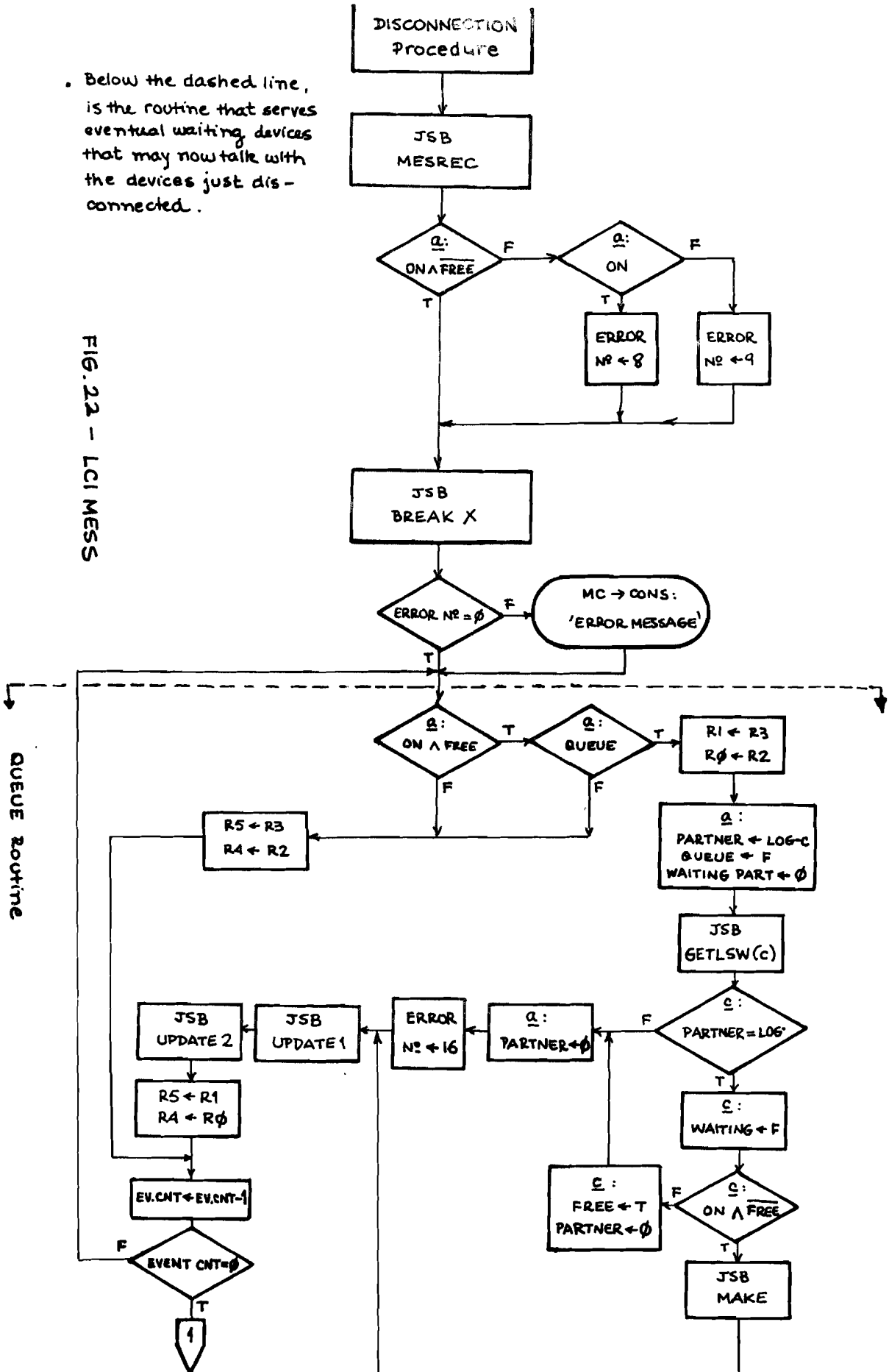


FIG. 22 - LCI MESS

DISCONNECTION Procedure :

- DISCONNECTION OF THE CROSSPOINT IS ORDERED , EXCEPT IF : \underline{b} - ON A $\overline{\text{FREE}}$ \wedge $\overline{(\text{PARTNER} = \text{LOG-a})}$
- WAITING QUEUE FOR \underline{a} IS SERVED IF : $\left\{ \begin{array}{l} \underline{a} - \text{ON A QUEUE} \\ \underline{c} - \text{ON A } \overline{\text{FREE}} \wedge (\text{PARTNER} = \text{LOG-a}) \end{array} \right.$ and
- WAITING QUEUE FOR \underline{b} IS SERVED IF : $\left\{ \begin{array}{l} \underline{b} - \text{ON A QUEUE} \\ \underline{c'} - \text{ON A } \overline{\text{FREE}} \wedge (\text{PARTNER} = \text{LOG-b}) \end{array} \right.$ and
- NETWORK MAP UPDATED (see QUEUE routine effects)
- REGISTERS MODIFIED (" " " ")
- MC \rightarrow # \underline{a} : 'DEVICE \underline{b} BUSY' if \underline{b} - ON A $\overline{\text{FREE}}$ \wedge $\overline{(\text{PARTNER} = \text{LOG-a})}$
- MC \rightarrow CONS : $\left\{ \begin{array}{l} \text{'DISCONNECTION FAILED'} \\ \text{'LCI CRAZY'} \end{array} \right.$ if LCI answers 'TEST FAILED'
if LCI answers neither 'TEST FAILED' nor 'TEST OK'

93

QUEUE Routine :

- AS input to the routine the EVENT counter should contain either 1 or 2 .
- CONNECTION IS ATTEMPTED IF $\left\{ \begin{array}{l} \underline{a} - \text{ON A QUEUE} \wedge \overline{\text{FREE}} \\ \underline{b} - \text{ON A QUEUE} \wedge \overline{\text{FREE}} \end{array} \right.$ and $\left\{ \begin{array}{l} \underline{c} - \text{ON A } \overline{\text{FREE}} \wedge (\text{PARTNER} = \text{LOG-a}) \\ \underline{c'} - \text{ON A } \overline{\text{FREE}} \wedge (\text{PARTNER} = \text{LOG-b}) \end{array} \right.$ and

FIG. 22 a - LCI MESS

- NETWORK MAP IS UPDATED IN THE FOLLOWING CASES :

• IF EVENT COUNTER = 1, 2 AT THE START OF THE ROUTINE and a - ON A FREE \wedge QUEUE ,
 THEN $\left[\begin{array}{l} (\text{PHY}-a), (\text{LOG}-a), (\text{PHY}-c), (\text{LOG}-c) \text{ ARE UPDATED} \\ \text{AND A CONNECTION IS ESTABLISHED BETWEEN } \underline{a} \text{ AND } \underline{c} . \end{array} \right.$

• IF EVENT COUNTER = 2 AT THE START OF ROUTINE and b - ON A FREE \wedge QUEUE ,
 THEN ALSO $\left[\begin{array}{l} (\text{PHY}-b), (\text{LOG}-b), (\text{PHY}-c'), (\text{LOG}-c') \text{ ARE UPDATED} \\ \text{AND A CONNECTION IS MADE BETWEEN } \underline{b} \text{ AND } \underline{c}' \end{array} \right.$

obs : c' IS THE DEVICE WAITING TO TALK WITH DEVICE b .

• a (b) - ON A FREE \wedge QUEUE and c (c') - ON A $\overline{\text{FREE}}$ \wedge (PARTNER = LOG - a(b))

THEN a (b) BECOME : QUEUE \leftarrow F ; FREE \leftarrow F ; PARTNER \leftarrow LOG - c(c') ; WAITING PART. \leftarrow ϕ

• a (b) - ON A FREE \wedge QUEUE and c (c') - ON A $\overline{\text{FREE}}$ \wedge (PARTNER = LOG - a(b))

THEN a (b) ARE UPDATED WITH : QUEUE \leftarrow F , FREE \leftarrow T , PARTNER \leftarrow ϕ ; WAITING PART \leftarrow ϕ

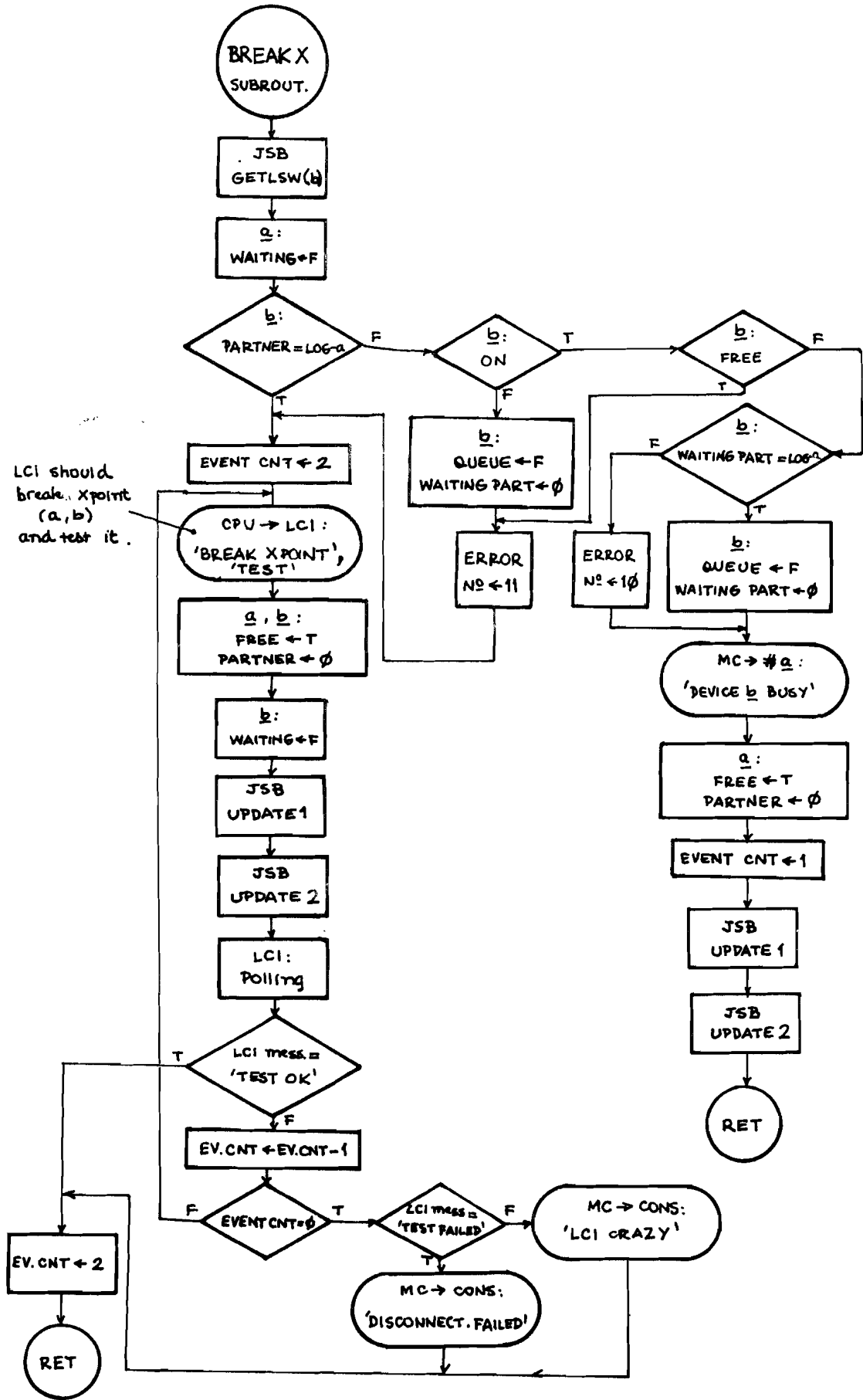
• a (b) - ON A FREE \wedge QUEUE and c (c') - (PARTNER = LOG - a(b))

IF $\left[\begin{array}{l} \underline{c} (\underline{c}') - \text{ON A } \overline{\text{FREE}} , \underline{c} (\underline{c}') \text{ UPDATED WITH : WAITING } \leftarrow \text{F ; FREE } \leftarrow \text{F ; PARTNER } \leftarrow \text{LOG-a(b)} \\ \underline{c} (\underline{c}') - \text{ON A } \overline{\overline{\text{FREE}}} , \underline{c} (\underline{c}') \text{ UPDATED WITH : WAITING } \leftarrow \text{F ; FREE } \leftarrow \text{T ; PARTNER } \leftarrow \phi ; \end{array} \right.$

• IN ALL OTHER CASES, NO MODIFICATION OCCURS IN THE NETWORK MAP.

FIG. 22 b - LCI MESS

FIG. 23 - LCI MESS



BREAK X Subroutine

- AT THE START OF THE SUBROUTINE, THE INTERNAL REGISTERS ARE SUPPOSED TO CONTAIN:

$$\begin{aligned} AC &\leftarrow (PHY-a) \\ R4 &\leftarrow (PHY-a) \\ R5 &\leftarrow (LOG-a) \end{aligned}$$

- NETWORK MAP IS UPDATED AS FOLLOWS:

o \underline{a} IS ALWAYS UPDATED WITH: $WAITING \leftarrow F$; $FREE \leftarrow T$; $PARTNER \leftarrow \phi$

o IF $\underline{b} - (PARTNER = LOG-a)$, \underline{b} UPDATED WITH: $FREE \leftarrow T$; $WAITING \leftarrow F$; $PARTNER \leftarrow \phi$

o IF $\underline{b} - \overline{(PARTNER = LOG-a)}$ and

$$\left[\begin{array}{l} \underline{b} - ON \wedge \overline{FREE} \wedge (WAITING \text{ PART} = LOG-a), \underline{b} \text{ IS UPDATED WITH: } QUEUE \leftarrow F; WAITING \text{ PART} \leftarrow \phi \\ \underline{b} - ON \wedge \overline{FREE} \wedge \overline{(WAITING \text{ PART} = LOG-a)}, \underline{b} \text{ SUFFERS NO MODIFICATION.} \\ \underline{b} - ON \wedge FREE, \underline{b} \text{ UPDATED WITH: } FREE \leftarrow T; WAITING \leftarrow F; PARTNER \leftarrow \phi \\ \underline{b} - ON, \underline{b} \text{ UPDATED WITH: } \left[\begin{array}{l} QUEUE \leftarrow F; FREE \leftarrow T; WAITING \leftarrow F; \\ PARTNER \leftarrow \phi; WAITING \text{ PART} \leftarrow \phi \end{array} \right. \end{array} \right.$$

- EVENT COUNTER AT THE END IS LOADED WITH $\left[\begin{array}{ll} 1 & \text{IF } \underline{b} - ON \wedge \overline{FREE} \\ 2 & \text{IF } \underline{b} - \overline{(ON \wedge \overline{FREE})} \end{array} \right.$

- MC \rightarrow CONS: $\left[\begin{array}{l} \text{'DISCONNECTION FAILED'} \text{ IF LCI REPORTS TEST ON X POINT WAS 'TEST FAILED'} \\ \text{'LCI CRAZY'} \text{ IF LCI ANSWERS WITH NEITHER 'TEST FAILED' NOR 'TEST OK'} \end{array} \right.$

- DISCONNECTION OF X POINT (a,b) IS ORDERED EXCEPT IF $\underline{b} - ON \wedge \overline{FREE} \wedge \overline{(PARTNER = LOG-a)}$

- QUEUE IS UNDONE IF $\underline{b} - ON \wedge \overline{FREE} \wedge (WAITING \text{ PART} = LOG-a) \wedge \overline{(PARTNER = LOG-a)}$

- MC \rightarrow # \underline{a} : 'DEVICE \underline{b} BUSY' IF $\underline{b} - ON \wedge \overline{FREE} \wedge \overline{(PARTNER = LOG-a)}$

FIG. 23 a - LCI MESS

The procedure makes many tests in order to detect uncoherences. It could be simplified, sacrificing safety.

FIG. 24 - LCI MESS

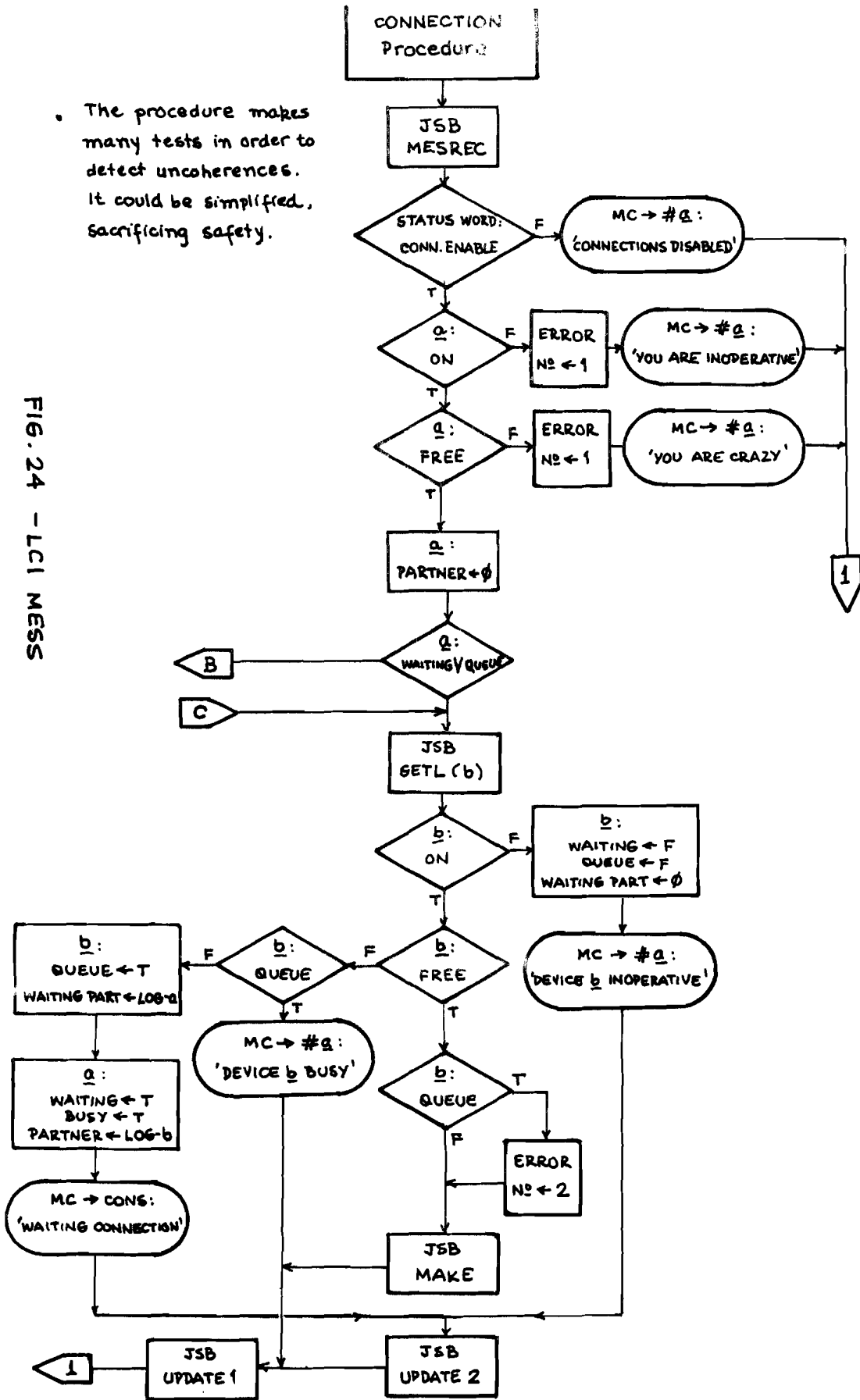
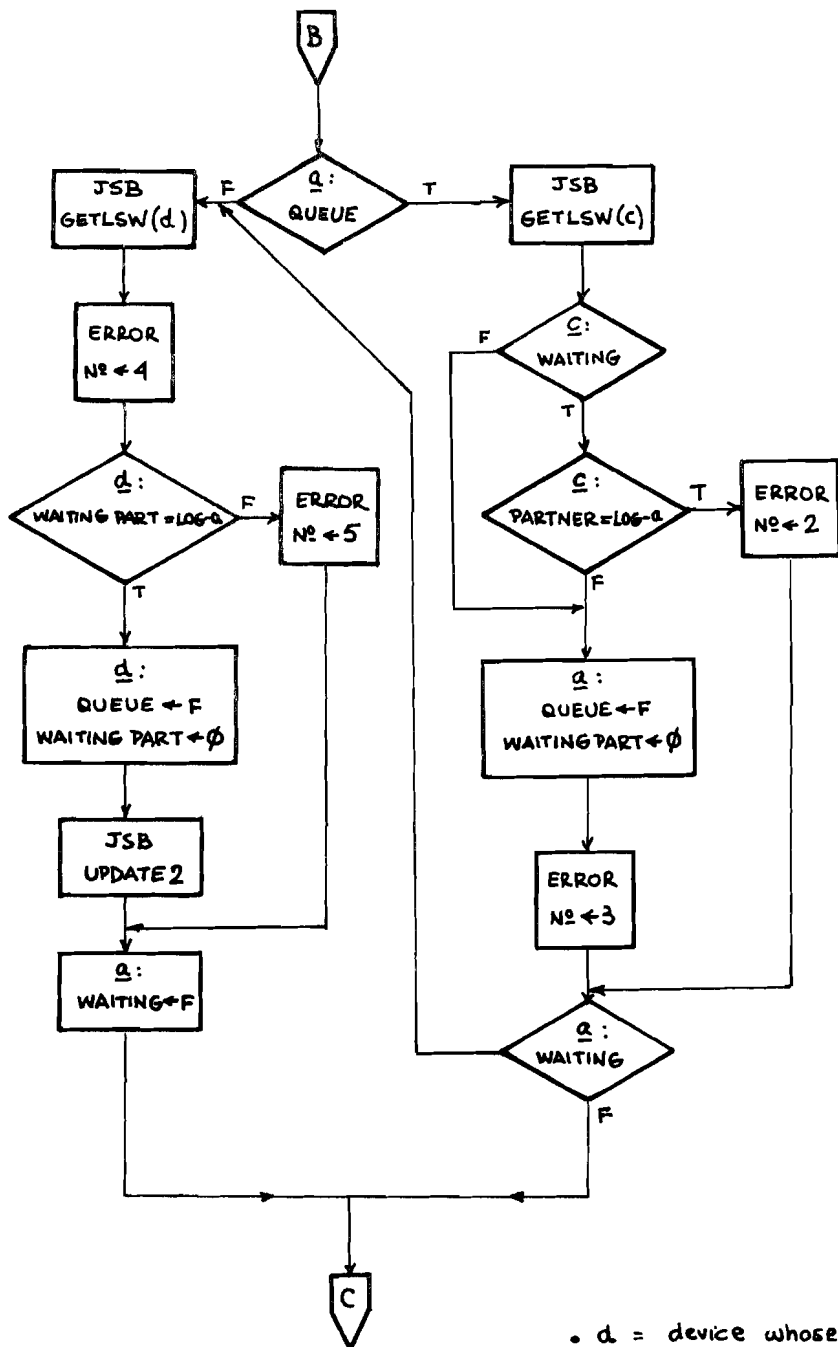


FIG. 24a - LCI MESS



• a = device whose LOG address is in the PARTNER field of (PHY-a)

• CONNECTION Procedure

- NETWORK MAP UPDATED IN THE CASES BELOW:

• IF \underline{a} - ON \wedge FREE and

$\left[\begin{array}{l} \underline{b} - \overline{\text{ON}} \\ \underline{b} - \text{ON} \wedge \overline{\text{FREE}} \wedge \overline{\text{QUEUE}} \\ \underline{b} - \text{ON} \wedge \overline{\text{FREE}} \wedge \text{QUEUE} \\ \underline{b} - \text{ON} \wedge \text{FREE} \end{array} \right.$	$\left[\begin{array}{l} \underline{a} \text{ IS UPDATED WITH : } \text{ON} \leftarrow \text{T ; FREE} \leftarrow \text{T ; WAITING} \leftarrow \text{F ; PARTNER} \leftarrow \phi \\ \underline{b} \text{ " " " : } \text{WAITING} \leftarrow \text{F ; QUEUE} \leftarrow \text{F ; WAITING PART} \leftarrow \phi \end{array} \right.$
	$\left[\begin{array}{l} \underline{a} \text{ GETS : } \text{FREE} \leftarrow \text{F ; WAITING} \leftarrow \text{T ; PARTNER} \leftarrow \text{LOG-b} \\ \underline{b} \text{ GETS : } \text{FREE} \leftarrow \text{F ; QUEUE} \leftarrow \text{T ; WAITING PART} \leftarrow \text{LOG-a} \end{array} \right.$
	$\left[\begin{array}{l} \underline{a} \text{ GETS : } \text{FREE} \leftarrow \text{T ; WAITING} \leftarrow \text{F ; PARTNER} \leftarrow \phi \\ \underline{b} \text{ SUFFERS NO MODIFICATION} \end{array} \right.$
	$\left[\begin{array}{l} \underline{a} \text{ GETS : } \text{FREE} \leftarrow \text{F ; WAITING} \leftarrow \text{F ; PARTNER} \leftarrow \text{LOG-b} \\ \underline{b} \text{ " : } \text{FREE} \leftarrow \text{F ; WAITING} \leftarrow \text{F ; PARTNER} \leftarrow \text{LOG-a} \end{array} \right.$

• IF \underline{a} - WAITING and \underline{d} - (WAITING PART = LOG-a),

THEN \underline{d} GETS : QUEUE \leftarrow F ; WAITING PART \leftarrow ϕ

obs. : \underline{d} is the device whose LOG address is in the PARTNER field of (PHY-a) at the start of the procedure.

• MC \rightarrow # \underline{a} :

$\left[\begin{array}{l} \text{'CONNECTIONS DISABLED'} \\ \text{'YOU ARE INOPERATIVE'} \\ \text{'YOU ARE CRAZY'} \\ \text{'DEVICE } \underline{b} \text{ INOPERATIVE'} \\ \text{'DEVICE } \underline{b} \text{ BUSY'} \\ \text{'X POINT DEFECTIVE'} \end{array} \right.$	<p>IF STATUS WORD HAS BIT 5 = HI</p> <p>IF \underline{a} - $\overline{\text{ON}}$</p> <p>IF \underline{a} - ON \wedge $\overline{\text{FREE}}$</p> <p>IF \underline{a} - ON \wedge FREE <u>and</u> \underline{b} - $\overline{\text{ON}}$</p> <p>IF \underline{a} - ON \wedge FREE <u>and</u> \underline{b} - ON \wedge $\overline{\text{FREE}} \wedge$ QUEUE</p> <p>IF $\left[\begin{array}{l} \underline{a} - \text{ON} \wedge \text{FREE} \text{ and} \\ \underline{b} - \text{ON} \wedge \text{FREE} \text{ and} \end{array} \right.$</p> <p style="padding-left: 40px;">LCI REPORTS THAT THE 'TEST FAILED'</p>
--	---

- CONNECTION OF XPOINT (a,b) IS ORDERED IF \underline{a} - ON \wedge FREE and \underline{b} - ON \wedge FREE

FIG. 24b - LCI MESS

- WAITING QUEUE IS DONE IF \underline{a} - ON A FREE and \underline{b} - ON A FREE A QUEUE
- MC → CONS :

'CONNECTION FAILED'	IF LCI REPORTS THAT THE 'TEST FAILED'
'LCI CRAZY'	" " " " " " " " " " NEITHER 'TEST FAILED' NOR 'TEST OK'

• MAKE Subroutine

- NETWORK MAP IS UPDATED :

\underline{a}	GETS	:	FREE ← F	;	PARTNER ← LOG - b
\underline{b}	"	:	FREE ← F	;	PARTNER ← LOG - a ; WAITING ← F
- EVENT COUNTER COMES OUT WITH $\phi, 1$ OR 2
- CONNECTION OF XPOINT (a, b) IS ALWAYS ORDERED
- MC → # \underline{a} : 'XPOINT DEFECTIVE' IF LCI REPORTS THAT THE 'TEST FAILED'
- MC → CONS :

['CONNECTION FAILED'	"	"	"	"	"	"	"
	'LCI CRAZY'	"	"	"	"	"	"	NEITHER 'TEST FAILED' NOR 'TEST OK'

100

FIG. 24c - LCI MESS

101

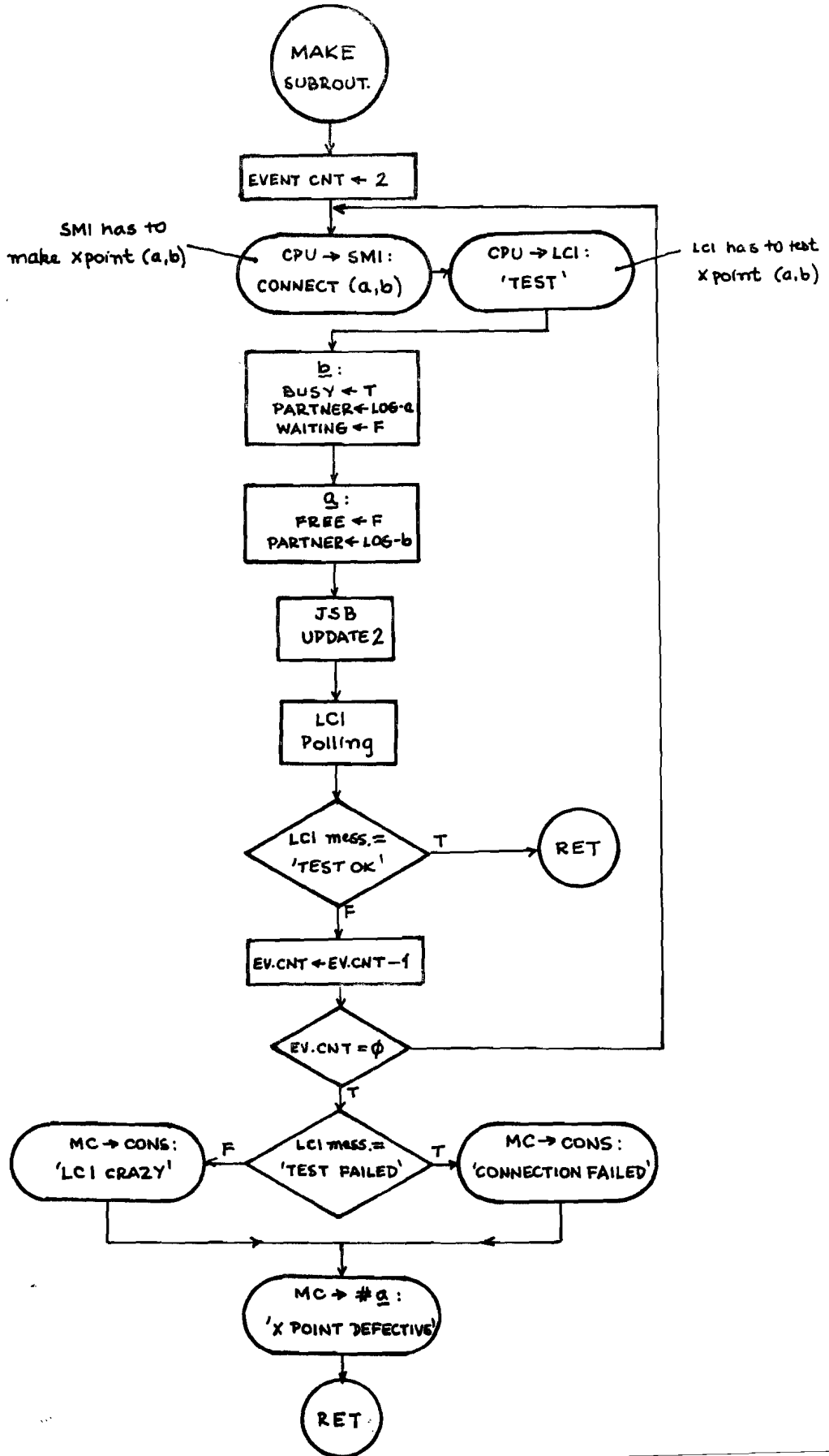


FIG. 25 - LCI MESS

FIG. 26 - LCI MESS

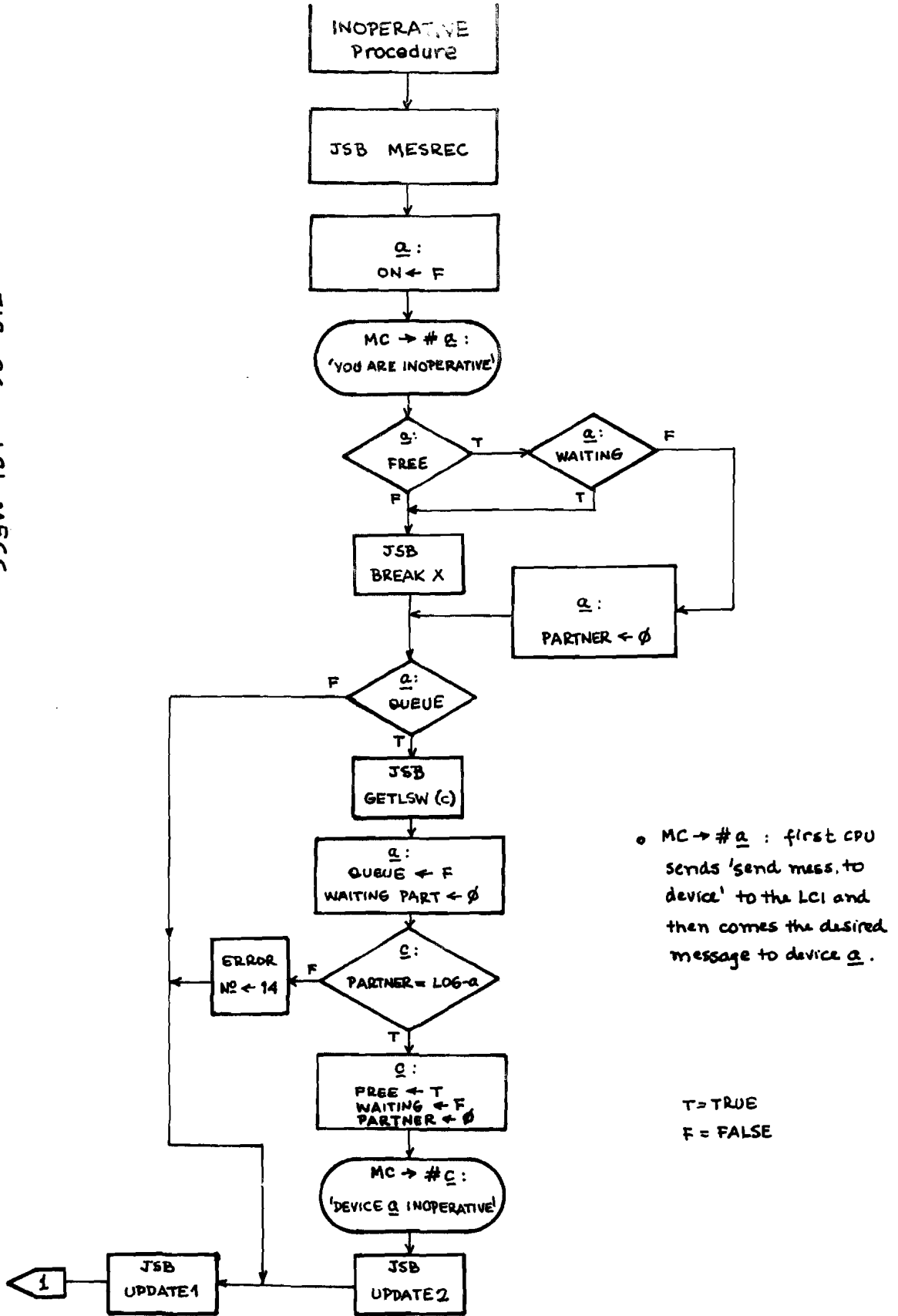
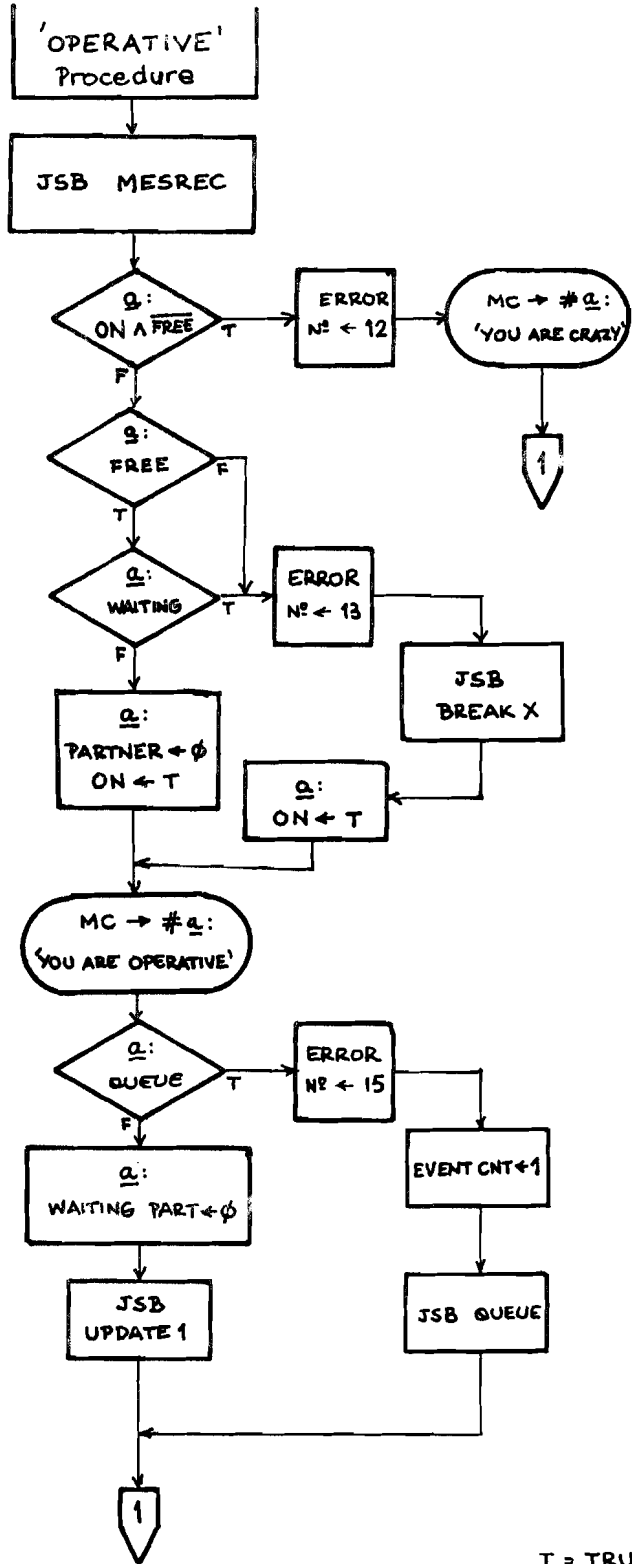


FIG. 27 - LCI MESS



T = TRUE
F = FALSE

• MC → # : means first a 'send message to device' message to LCI, followed by the desired message to the device .

OPERATIVE Procedure

- NETWORK MAP UPDATED IN THE FOLLOWING CASES :

- IF \underline{a} - ON \wedge $\overline{\text{FREE}}$, NOTHING IS MODIFIED .
- IF \underline{a} - $(\overline{\text{ON}} \vee \text{FREE}) \wedge \text{QUEUE}$ and \underline{c} - ON \wedge $\overline{\text{FREE}} \wedge (\text{PARTNER} = \text{LOG}-\underline{a})$
 THEN \underline{a} GETS : $\left[\begin{array}{l} \text{ON} \leftarrow \text{T} ; \text{FREE} \leftarrow \text{F} ; \text{WAITING} \leftarrow \text{F} ; \text{QUEUE} \leftarrow \text{F} ; \\ \text{PARTNER} \leftarrow \text{LOG}-\underline{c} ; \text{WAITING PART} \leftarrow \emptyset \end{array} \right.$
- IF \underline{a} - $\overline{\text{ON}} \vee \text{FREE}$ (EXCEPT THE CASE ABOVE DESCRIBED) ,
 THEN \underline{a} GETS : $\left[\begin{array}{l} \text{ON} \leftarrow \text{T} ; \text{FREE} \leftarrow \text{T} ; \text{WAITING} \leftarrow \text{F} ; \text{QUEUE} \leftarrow \text{F} ; \\ \text{PARTNER} \leftarrow \emptyset ; \text{WAITING PART} \leftarrow \emptyset \end{array} \right.$
- IF \underline{a} - $(\text{FREE} \wedge \text{WAITING}) \vee (\overline{\text{FREE}} \wedge \overline{\text{ON}})$ and
 $\left[\begin{array}{l} \underline{b} - (\text{PARTNER} = \text{LOG}-\underline{a}) , \text{ THEN } \underline{b} \text{ GETS : } \text{FREE} \leftarrow \text{T} ; \text{WAITING} \leftarrow \text{F} ; \text{PARTNER} \leftarrow \emptyset \\ \underline{b} - (\overline{\text{PARTNER}} = \text{LOG}-\underline{a}) \wedge \left[\begin{array}{l} \text{ON} \wedge \overline{\text{FREE}} \wedge (\text{WAITING PART} = \text{LOG}-\underline{a}) , \underline{b} \text{ GETS : } \text{QUEUE} \leftarrow \text{F} ; \text{WAITING PART} \leftarrow \emptyset \\ \text{ON} \wedge \overline{\text{FREE}} \wedge (\overline{\text{WAITING PART}} = \text{LOG}-\underline{a}) , \underline{b} \text{ IS NOT MODIFIED} \\ \text{ON} \wedge \text{FREE} , \underline{b} \text{ GETS : } \text{FREE} \leftarrow \text{T} ; \text{WAITING} \leftarrow \text{F} ; \text{PARTNER} \leftarrow \emptyset \\ \overline{\text{ON}} , \underline{b} \text{ GETS : } \left[\begin{array}{l} \text{FREE} \leftarrow \text{T} ; \text{QUEUE} \leftarrow \text{F} ; \text{WAITING} \leftarrow \text{F} ; \\ \text{PARTNER} \leftarrow \emptyset ; \text{WAITING PART} \leftarrow \emptyset \end{array} \right. \end{array} \right.$
- IF \underline{a} - $(\overline{\text{ON}} \vee \text{FREE}) \wedge \text{QUEUE}$ and
 $\left[\begin{array}{l} \underline{c} - (\text{PARTNER} = \text{LOG}-\underline{a}) \wedge \left[\begin{array}{l} \text{ON} \wedge \overline{\text{FREE}} , \underline{c} \text{ GETS : } \text{FREE} \leftarrow \text{F} ; \text{WAITING} \leftarrow \text{F} ; \text{PARTNER} \leftarrow \text{LOG}-\underline{a} \\ \overline{\text{ON}} \vee \text{FREE} , \underline{c} \text{ GETS : } \text{FREE} \leftarrow \text{T} ; \text{WAITING} \leftarrow \text{F} ; \text{PARTNER} \leftarrow \emptyset \end{array} \right. \\ \underline{c} - (\overline{\text{PARTNER}} = \text{LOG}-\underline{a}) , \underline{c} \text{ IS NOT MODIFIED} \end{array} \right.$

FIG. 27 a - LCI MESS

- CONNECTION OF X POINT (a,c) IS ORDERED IF

$$\underline{a} - (\overline{ON} \vee \overline{FREE}) \wedge \text{QUEUE} \quad \text{and} \quad \underline{c} - \overline{ON} \wedge \overline{FREE} \wedge (\text{PARTNER} = \text{LOG-a})$$

- DISCONNECTION OF X POINT (a,b) IS ORDERED IF

$$\underline{a} - (\overline{FREE} \wedge \overline{WAITING}) \vee (\overline{ON} \wedge \overline{FREE}) \quad \text{and} \quad \underline{b} - \overline{ON} \wedge \overline{FREE} \wedge (\overline{\text{PARTNER} = \text{LOG-a}})$$

- WAITING QUEUE IS UNDONE IF

$$\underline{a} - (\overline{FREE} \wedge \overline{WAITING}) \vee (\overline{ON} \wedge \overline{FREE}) \quad \text{and} \quad \underline{b} - \overline{ON} \wedge \overline{FREE} \wedge (\overline{\text{PARTNER} = \text{LOG-a}}) \wedge (\overline{\text{WAIT. PART} = \text{LOG-a}})$$

- MC \rightarrow # a : $\left[\begin{array}{ll} \text{'YOU ARE CRAZY'} & \text{IF } \underline{a} - \overline{ON} \wedge \overline{FREE} \\ \text{'YOU ARE OPERATIVE'} & \text{IF } \underline{a} - \overline{ON} \vee \overline{FREE} \end{array} \right.$

obs. : \underline{b} is the device whose LOG address appears in the PARTNER field of (PHY-a)
 \underline{c} " " " " " " " " " " " " " " WAITING PART field of (LOG-a)

FIG. 27b - LCI MESS

• This routine must be in the upper half of the μ Program Store

• P.C. = Program Counter

• SP = Stack Pointer

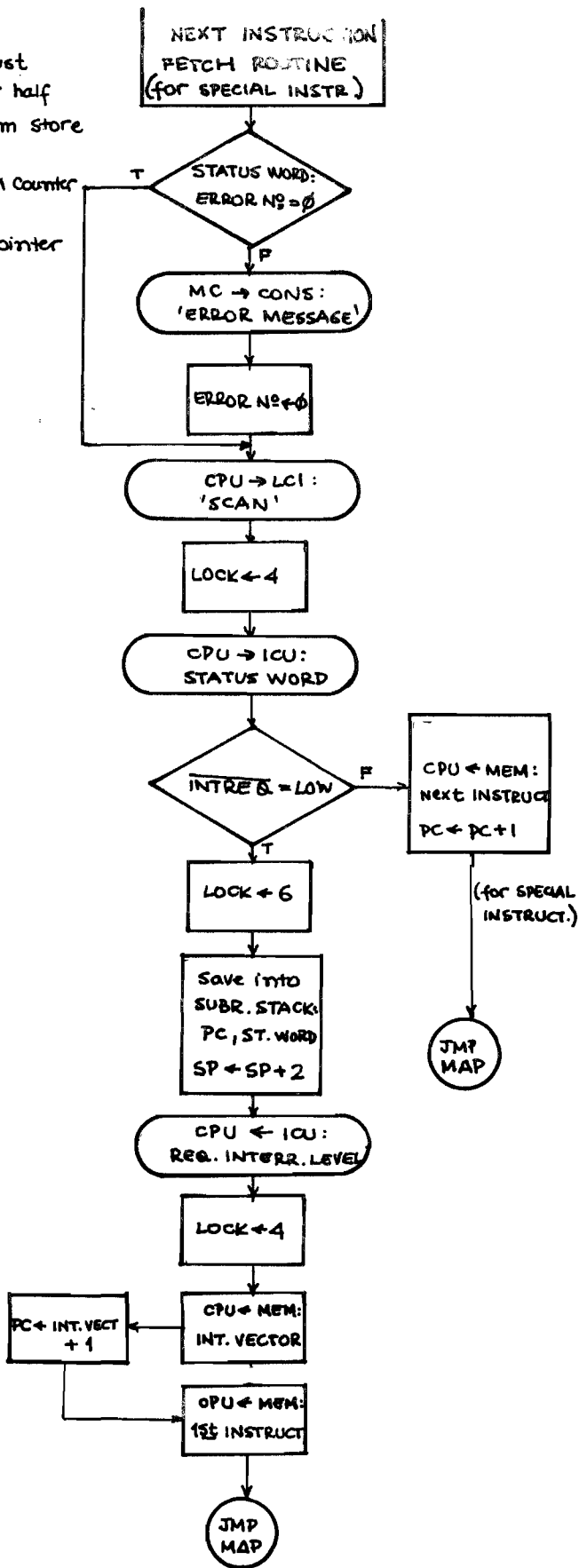


FIG. 28 - NEXT INSTRUCTION FETCH μ ROUTINE
(FOR SPECIAL INSTRUCTIONS)

• NEXT INSTRUCTION FETCH Routine (for special instructions)

- IT IS ASSUMED THAT THE STACK POINTER POINTS AT THE FIRST FREE PLACE IN THE SUBROUTINE STACK.
- IF $\text{INTREQ} \wedge \text{INTERRUPT ENABLE}$ (obs.: INT. ENABLE IS THE BIT 6 IN THE STATUS WORD)
 - THEN

PROGRAM COUNTER (R8)	\leftarrow INTERRUPT VECTOR (n) + 1				
STACK POINTER (R7)	\leftarrow STACK POINTER + 2				
SUBROUTINE STACK :	<table style="border-left: 1px solid black; border-right: 1px solid black; border-collapse: collapse;"> <tr> <td style="padding: 5px;">POSITION (SP + 64)</td> <td style="padding: 5px;">\leftarrow old STATUS WORD</td> </tr> <tr> <td style="padding: 5px;">" (SP + 65)</td> <td style="padding: 5px;">\leftarrow old PROGRAM COUNTER</td> </tr> </table>	POSITION (SP + 64)	\leftarrow old STATUS WORD	" (SP + 65)	\leftarrow old PROGRAM COUNTER
POSITION (SP + 64)	\leftarrow old STATUS WORD				
" (SP + 65)	\leftarrow old PROGRAM COUNTER				
STATUS WORD :	<table style="border-left: 1px solid black; border-right: 1px solid black; border-collapse: collapse;"> <tr> <td style="padding: 5px;">INTERRUPT LEVEL</td> <td style="padding: 5px;">$\leftarrow n$</td> </tr> <tr> <td style="padding: 5px;">ERROR NUMBER</td> <td style="padding: 5px;">$\leftarrow \phi$</td> </tr> </table>	INTERRUPT LEVEL	$\leftarrow n$	ERROR NUMBER	$\leftarrow \phi$
INTERRUPT LEVEL	$\leftarrow n$				
ERROR NUMBER	$\leftarrow \phi$				
 - obs.: n is the new interrupt requesting level
- IF $\text{INTREQ} \wedge \text{INTERRUPT ENABLE}$
 - THEN

PROGRAM COUNTER	\leftarrow PROGRAM COUNTER + 1
STACK POINTER	UNCHANGED
SUBROUTINE STACK	"
STATUS WORD :	ERROR NUMBER $\leftarrow \phi$
- LOCK $\leftarrow 4$ (ONLY MEMORY READ ALLOWED)
- MC \rightarrow CONS : 'ERROR MESSAGE' (with ERROR NUMBER given by ERROR NUMBER field in the STATUS WORD), IF ERROR NUMBER $\neq \phi$ at the start of the routine.
- CPU \rightarrow LCI : 'SCAN'

FIG. 28 a - NEXT INSTRUCTION FETCH ROUTINE

μA	PL. ADDRESS	NEXT ADDRESS	F	M	MAP L/P	D/M	D-REG	A-EN	D-EN	CC SEL	CI	FUNCTION	MASK	COMMENTS
	-	CONTINUE	1	-	0 0	-	1	1	1	0	1	0 1 (R9)	11111	AC \leftarrow R9 (STATUS WORD)
	ERROR	COND JSB PL	0	-	0 0	-	1	1	1	1	1	5111 (AC)	01111	AC, CO \leftarrow ERROR N ^o
	-	CONTINUE	0	0	1 0	1	1	0	0	0	1	1 1 (AC)	11101	MAR \leftarrow 0 (LCI addr); LCI \leftarrow 'SCAN'
	-	CONTINUE	1	-	0 0	-	1	1	1	4	1	0 1 (R9)	11111	AC \leftarrow R9 ; LOCK \leftarrow 4
	F ₁₆	LD CNTR&CONTIN	0	-	0 0	1	1	1	0	1	0	2 11 (AC)	00000	ICU \leftarrow INT.ENABLE+INT.LEVEL
	PROCHAIN	COND JMP PL	1	1	0 0	0	1	0	0	7	1	1 1 (R8)	11111	MAR \leftarrow R8
	-	CONTINUE	0	1	0 0	0	0	0	1	0	0	0 1 (R8)	11111	R8 \leftarrow R8 + 1; D-REG \leftarrow next instr.
	-	JMP MAP	1	-	0 0	-	1	1	1	0	1	5111 (AC)	11111	AC \leftarrow 0; Go to next instr. μ prog
	-	CONTINUE	1	-	0 0	-	1	1	1	6	0	1 1 (R7)	11111	MAR \leftarrow R7 (SP.); R7 \leftarrow R7 + 1
	-	CONTINUE	0	0	0 0	0	1	0	0	0	1	0 1 (R9)	11111	AC, Subr.Stack \leftarrow R9 (STAT.WD)
	F ₁₆	LD CNTR&CONTINUE	1	0	0 0	1	0	1	1	1	0	1 1 (R7)	11111	MAR \leftarrow R7 ; R7 \leftarrow R7 + 1 ; D-REG \leftarrow new Interr. Level
	-	CONTINUE	0	0	0 0	0	1	0	0	0	1	0 1 (R8)	11111	AC, Subr.Stack \leftarrow R8 (P.C.)
	-	CONTINUE	1	-	0 0	-	1	1	1	4	1	5 1 (R9)	00001	LOCK \leftarrow 4; INTERR.LEVEL \leftarrow 0
	-	CONTINUE	1	-	0 0	-	1	1	1	0	1	5 11 (AC)	11110	AC \leftarrow new Interr. Level
	-	CONTINUE	1	-	0 0	-	1	1	1	0	1	3 1 (R9)	11110	INTERR.LEVEL \leftarrow new Interr.Level
	-	CONTINUE	0	0	0 0	0	0	0	1	0	1	1 1 (AC)	10011	MAR \leftarrow INT.LEVEL + 96 ; D-REG \leftarrow Interr. Vector
	-	CONTINUE	1	1	0 0	0	0	0	1	0	0	1 11 (AC)	11111	MAR \leftarrow Interr.Vector ; AC \leftarrow Interr.Vector + 1
	-	CONTINUE	0	1	0 0	0	0	0	1	0	0	2 1 (R8)	00000	R8 \leftarrow AC ; D-REG \leftarrow 1 st Instr.

FIG. 28b - NEXT INSTR. FETCH μ ROUTINE

μA	PL. ADDRESS	NEXT ADDRESS	F	M	MAP	L/P	P/M	D-REG	A-EN	D-EN	CC. SEL	CI	FUNCTION	MASK	COMMENTS
ERROR \rightarrow	-	JMP MAP	1	-	0 0	-		1	1	1	0	1	5111 (AC)	11111	AC \leftarrow 0; Go to 1 st inst. μ prog.
	-	CONTINUE	1	-	0 0	-		1	1	1	0	1	1 1 (AC)	10000	MAR \leftarrow F ₁₆ (CONS addr.); AC \leftarrow 'ERROR MESSAGE'
	-	CONTINUE	1	0	0 0	1		1	0	0	0	1	5 1 (R9)	10000	ERROR N ^o \leftarrow 0; CONS \leftarrow 'ERROR MESSAGE'
	-	COND RTN	1	-	0 0	-		1	1	1	4	1	5111 (AC)	11111	AC \leftarrow 0 ; Return

FIG. 28 C - NEXT INSTR. FETCH μ ROUTINE (continuation)

END OF MICROPROGRAM OF A STANDARD INSTRUCTION :

IF IT NEEDS STORE RESULT
BACK INTO R_n

EV. COUNTER $\leftarrow \phi$

JMP
VECTOR

IF IT DOES NOT HAVE
TO STORE IN R_n

I

(obs.: I means a branch to
STAND. INSTR. FETCH routine)

REGISTER ROUTINE
(EV. COUNTER $\neq \phi$)

$AC \leftarrow R_n$

JMP
MAP

REGISTER ROUTINE
(EV. COUNTER = ϕ)

$R_n \leftarrow AC$

I

- $R_n = R_1, R_2, \dots, R_9, ACC(T)$
- If the 4 LSB in the op. code of the standard instruction are all HI, $R_n = AC$, so that nothing happens, just a branch.

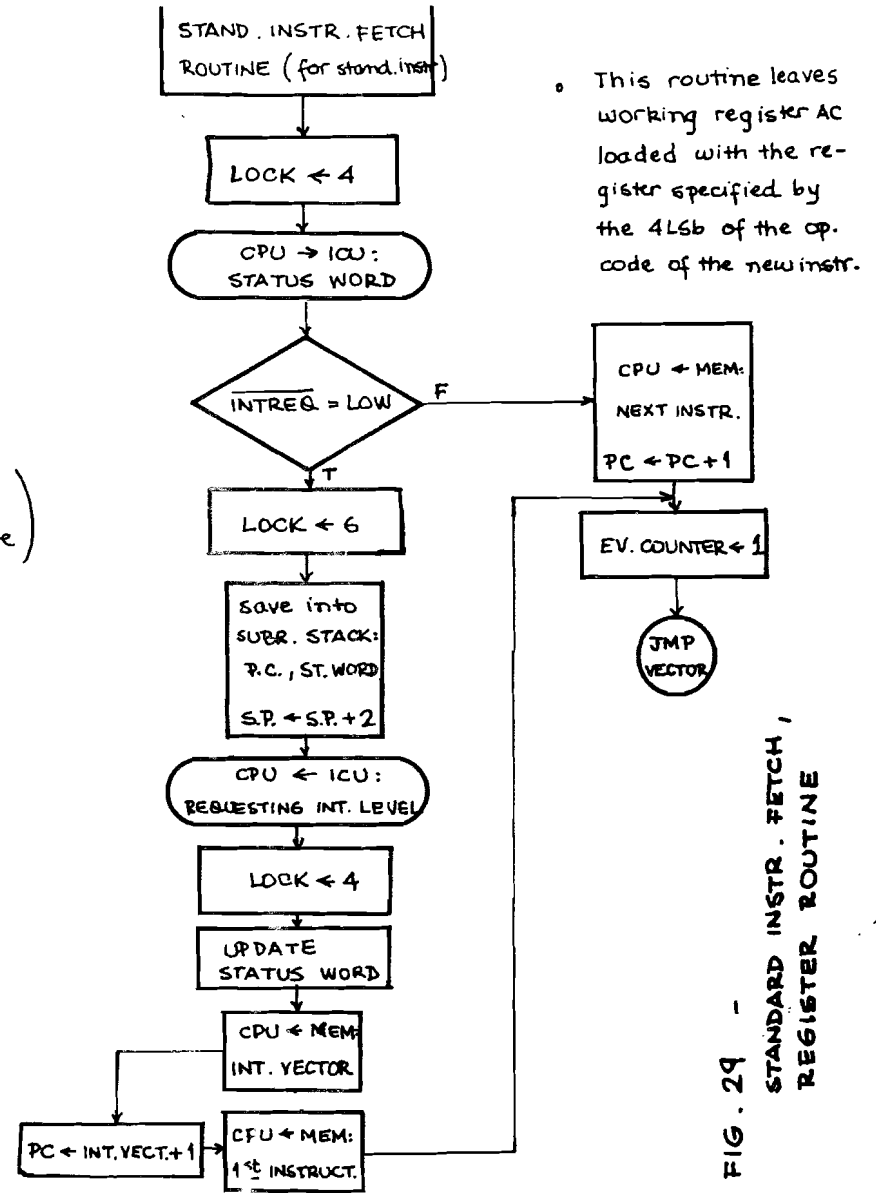


FIG. 29 -
STANDARD INSTR. FETCH,
REGISTER ROUTINE

μA	PL ADDRESS	NEXT ADDRESS	F	M	MAP	L/P	P/M	D-REG	A-EN	D-EN	CC SEL	CI	FUNCTION	MASK	COMMENTS
0E0	-	JMP MAP	1	-	0	0	-	1	1	1	0	1	5111 (AC)	00000	registers unchanged
0E1	STAND. INST. FET	COND JMP PL	1	-	0	0	-	1	1	1	4 (TRUE)	1	5111 (AC)	00000	" "
0E2	-	JMP MAP	1	-	0	0	-	1	1	1	0	1	0 1 (R1)	11111	AC ← R1
0E3	STAND. INST. FET	COND JMP PL	1	-	0	0	-	1	1	1	4	0	2 1 (R1)	00000	R1 ← AC
0E4	-	JMP MAP	1	-	0	0	-	1	1	1	0	1	0 1 (R2)	11111	AC ← R2
0E5	STAND. INST. FET	COND JMP PL	1	-	0	0	-	1	1	1	4	0	2 1 (R2)	00000	R2 ← AC
0E6	-	JMP MAP	1	-	0	0	-	1	1	1	0	1	0 1 (R3)	11111	AC ← R3
0E7	STAND. INST. FET	COND JMP PL	1	-	0	0	-	1	1	1	4	0	2 1 (R3)	00000	R3 ← AC
.															.
.															.
.															.
0F2	-	JMP MAP	1	-	0	0	-	1	1	1	0	1	0 1 (R9)	11111	AC ← R9
0F3	STAND. INST. FET	COND JUMP PL	1	-	0	0	-	1	1	1	4	0	2 1 (R9)	00000	R9 ← AC
0F4	-	JMP MAP	1	-	0	0	-	1	1	1	0	1	0 1 (T)	11111	AC ← ACC(T)
0F5	STAND. INST. FET	COND JMP PL	1	-	0	0	-	1	1	1	4	0	2 1 (T)	00000	ACC(T) ← AC

FIG. 29 a - REGISTER ROUTINES

- +5V : RAILS
- GND : METALIZED TRACKS
- 4114R CONTAINS 1KΩ RESISTORS TO +5V.

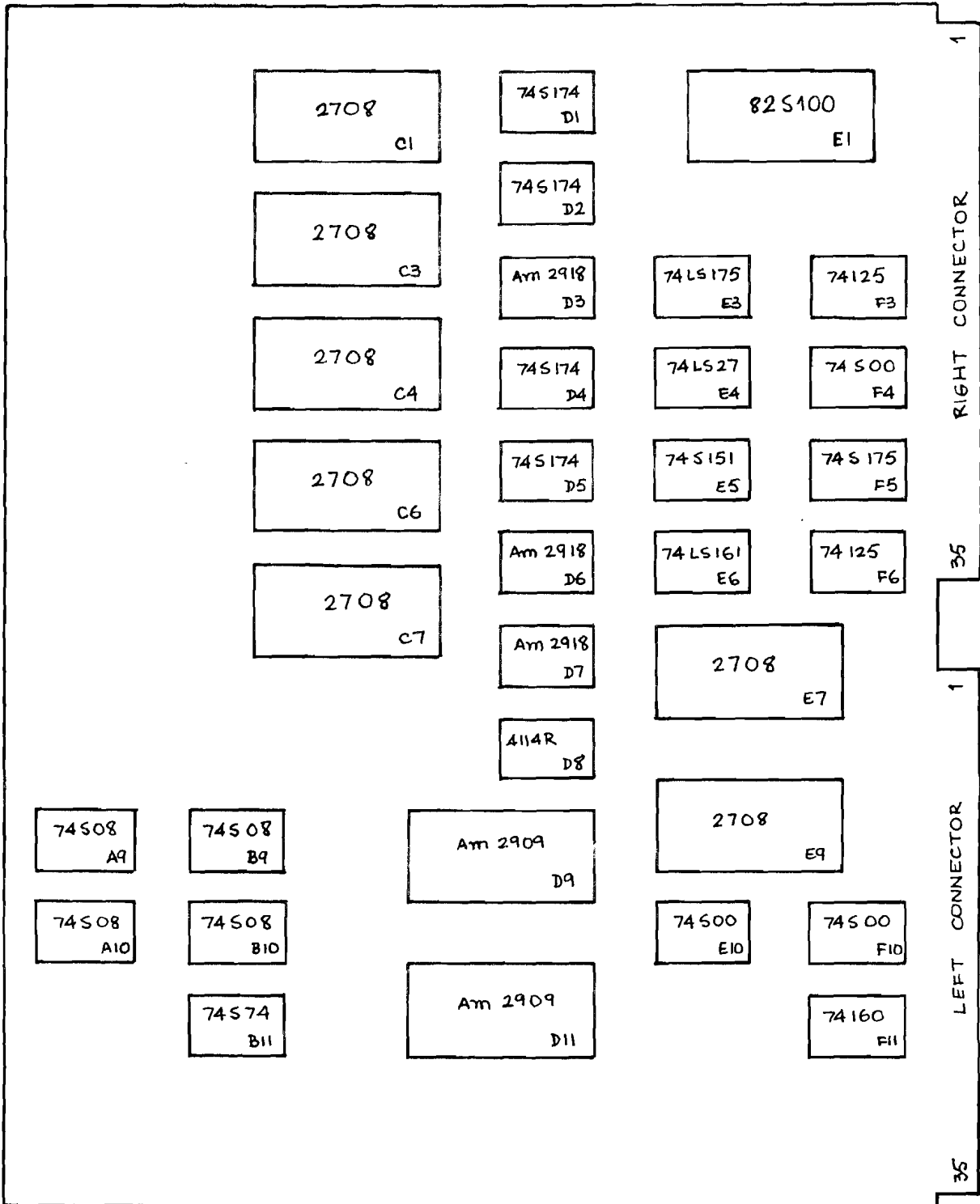


FIG. 30 - CIRCUIT BOARD I : COMPONENTS' SIDE

- +5V : RAILS
- GND : METALLIZED TRACKS
- 4114R CONTAINS 1K Ω RESISTORS TO +5V

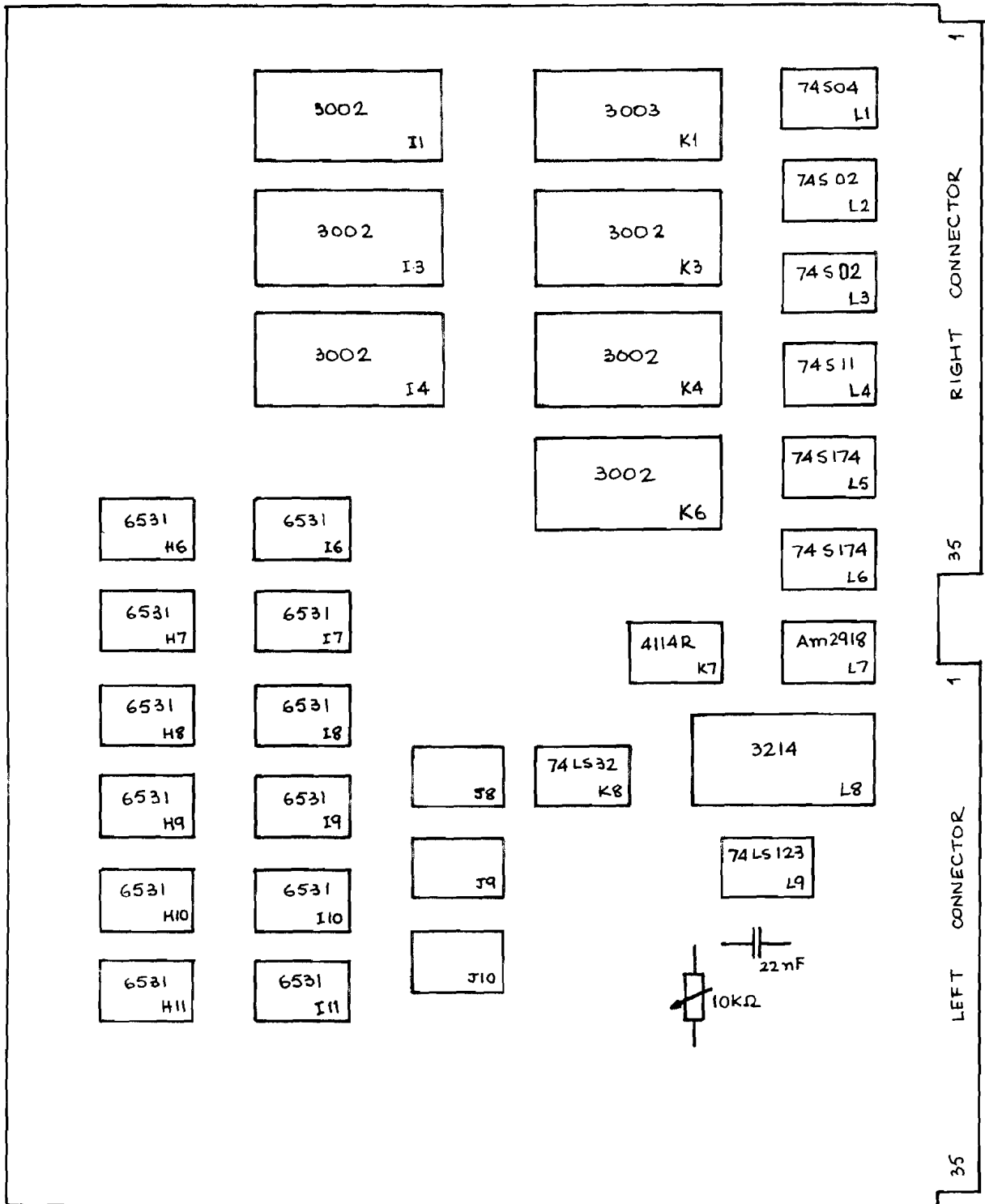


FIG. 31 - CIRCUIT BOARD II : COMPONENTS' SIDE

- +5V : WHITE RAILS
- GND : BARE RAILS
- 4114R CONTAINS 220Ω CONNECTED BETWEEN LED AND OPEN COLLECTOR OUTPUT .

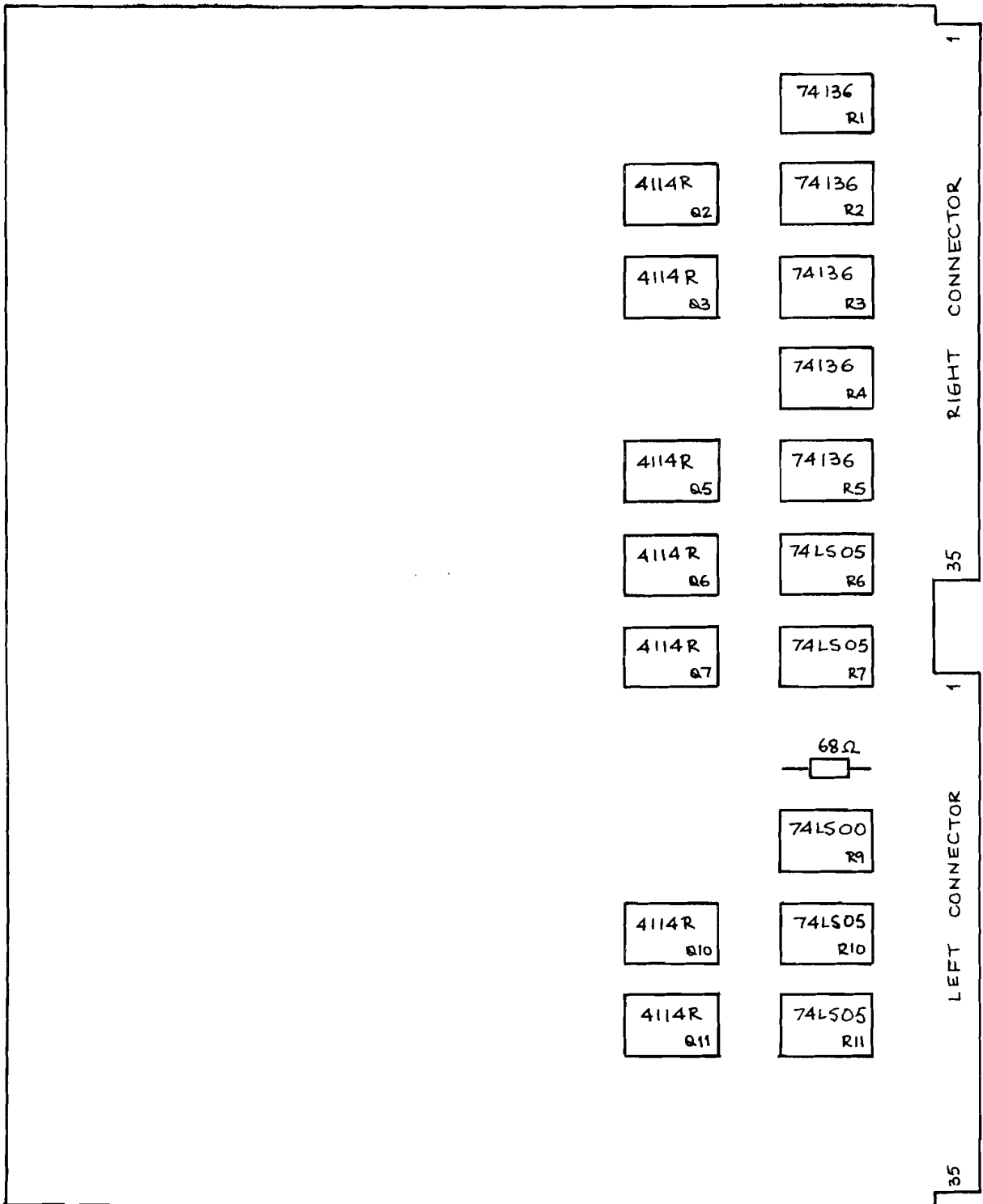


FIG. 32 - CIRCUIT BOARD III : COMPONENTS' SIDE