I N D E X

Page

Page

GENERAL INFORMATION

The M 38 System is an 8 bit microprocessor family made up by the following devices:

1 - The CPU - M 380 -

The unit is working on 8 bits, and 48 general purpose registers are available inside.

Also I/O facilities are included (8 bit Input, 4 bit output).

2 - The ROM - M 382 -

This device, intended as program memory, is a 1 K by 8 bits ROM.
Two I/O ports, 8 bit each, are available on every ROM device.

3 - The RAM - M 383 -

This device, intended as data memory, is a 128 words by 8 bits RAM, and an I/O port by 8 bits is also available.

4 - The ROM/RAM - M 381 -

This is a composite device with 768 words of ROM and 18 words of RAM.

Two I/O ports by 8 bits are included.

./...

Due to its peculiar architecture the M 38 system lends itself to very compact hardware configurations.
The minimum system configuration is a 2 chip system, composed of:

1 - CPU

2 - ROM or ROM/RAM

This system has already all the basic elements which are needed to solve problems, namely:

- CPU
- 48 Registers
- 1 K ROM
- 24 I/O bits

The maximum clock frequency of the system is 800 KHZ, while the basic machine cycle is 5 μ sec.
The execution time per instruction is ranging from 5 μ sec. to 20 μ sec. maximum.

The family is implemented by P-Channel technology and it requires 0, -5 V, - 17 V as power supplies.
The typical power dissipation is 500 mw/chip.

./..

SYSTEM ARCHITECTURE

Let's look now little more in detail at the hardware configu ration of a system.
On Figure 1 all the typical elements and devices of a small system are represented:

1 - a CPU chip
2 - a ROM chip
3 - a ROM/RAM chip
4 - a RAM chip
5 - a Clock generator
6 - a Syncronization network
7 - a Set of internal buses (data bus, address bus, control lines, clock and sync. line)

The clock generator is a very simple circuit and it could be done by 2 COS-MOS inverters and either an R.C. network or a crystal quarz depending on the time accuracy required.
The syncronization network is also a simple circuit made up by two D flip-flop and one inverter. Inputs on this network are the clock and the start switch signals.
The Output is the sync. signal changing from Ø to 1 together with the clock leading edge that follows the start switch commutation (see fig. 2).
From that moment on the internal timing on every chip is enabled with the same time relationship, and the system is in operation.
On figure 3, the circuit details for the clock generator and the syncronization network are indicated.

We have discussed so far the extra circuits needed for the timing, and we may note here that only two lines (clock, sync.) are required by the system for timing purposes.

./..

If we now take a look to the other interconnecting lines we can see:

1 - A standard 8 bit bidirectional data bus.

2 - A set of 4 control lines coming out of the CPU, which are carrying information to the system elements (ROM, RAM, I/O) about internal operations to be performed (mainly the data transfer direction).

3 - A 6 bit address bus. The address range of this bus is from 1 to 64.
Of course, the bus is not to address program or data memories, but to enable some logic blocks by which the system is made up.

The last point is peculiar to the M 38 system, and we better spend some more words about that feature.
As we have stated before, the system is divided in elementary blocks, (they will be called, from now on as modules) of the following types:

1 - A block of 256 words of ROM.

2 - A block of 128 words of RAM.

3 - One I/O port.

4 - A block of 18 words of RAM (the small RAM included into the ROM/RAM device).

During the design stage of the system, the system has to be divided in modules according to the previous definitions and to every module it must be assigned one of the 64 available codes.
Any time that the CPU wants to enable a module on the system to perform some operation (receive or transfer data, push or pop operation on a stack, etc.), it has to send on the address bus the corresponding module address and this will act as an enable on the module itself.
So the address bus is used to address or enable modules(maximum number of modules into a system is 64).

./..

The hardware of the system has been implemented in such a way that there are two different situations depending on on the addressed module type:

1 - RAM or I/O modules are enabled one per time: this means that when the system is addressing one of these modules, it has to generate one of the 64 codes of the available address range, and so the 6 bis address must be fully specified.

2 - ROM modules are enabled 8 per times. This is due to the fact that when a ROM module is addressed, the hardware will take into account only the three most significant bits on the address bus. The selection between the 8 modules, and inside the module itself, is done by the logic into the enabled ROM. A block of 8 ROM modules corresponds to a block of 2 K ROM.

The modular organization so far described, has the main advantage to need a limited number of lines for addressing purposes (only 6).
And this leads to another advantage the saving of chip area. Infact only 6 buffers are needed for the address bus.

By these two hardware features it has been possible to include into the hardware of the system two very important facilities:

1 - Distributed I/O Ports. Every device has one or two I/O Port (8 bit each) available inside.

2 - RAM Storage (48 registers by 8 bit) into the CPU.

The module concept either from the system design point of view or from the hardware point of view will be explained in more details in the next two paragraphs.

./..

MODULE :
256×8 ROM
128×8 RAM
1 I/O PORT (8 BIT)
18×8 RAM

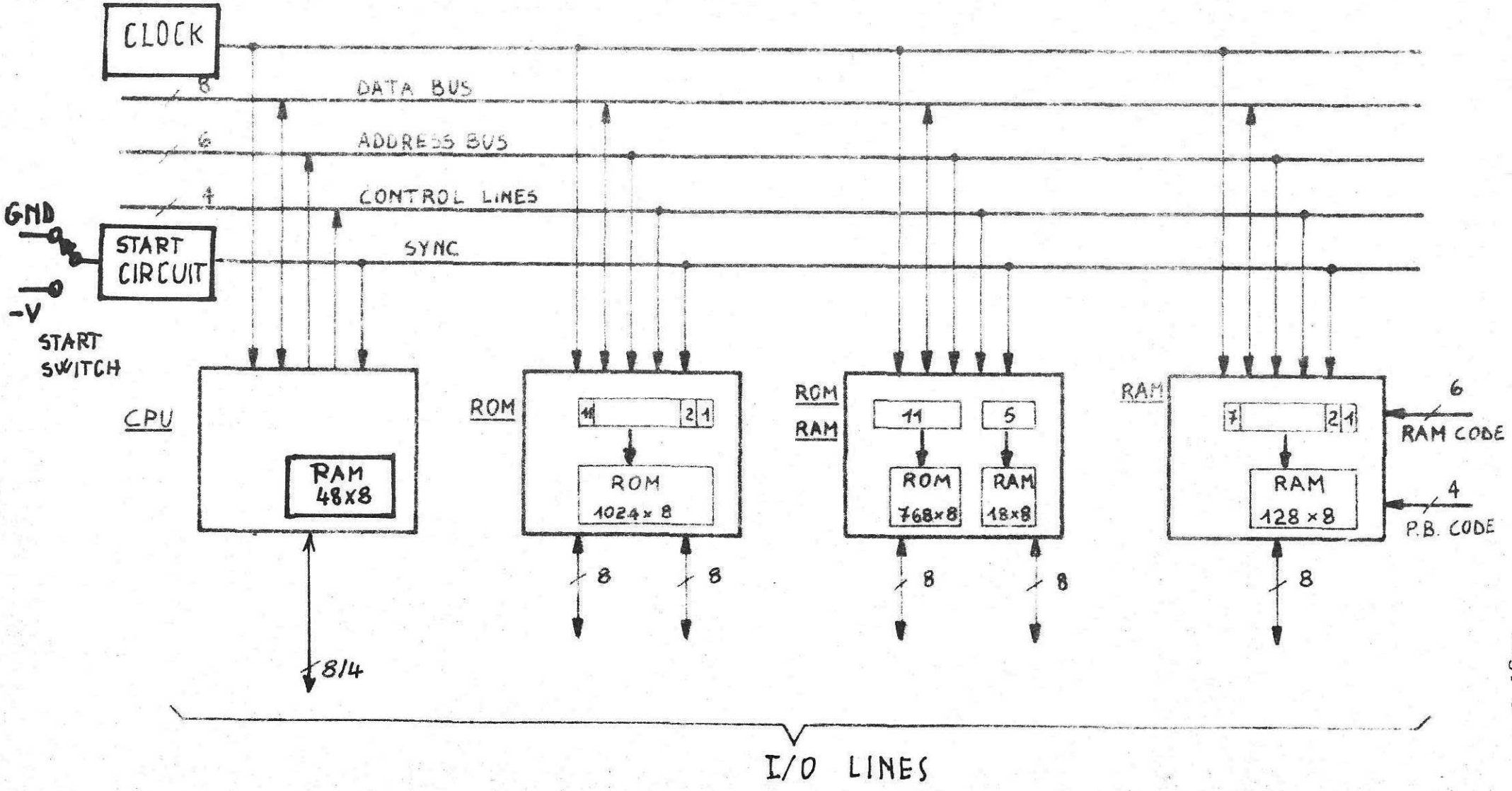


Fig. 1 - TYPICAL SYSTEM

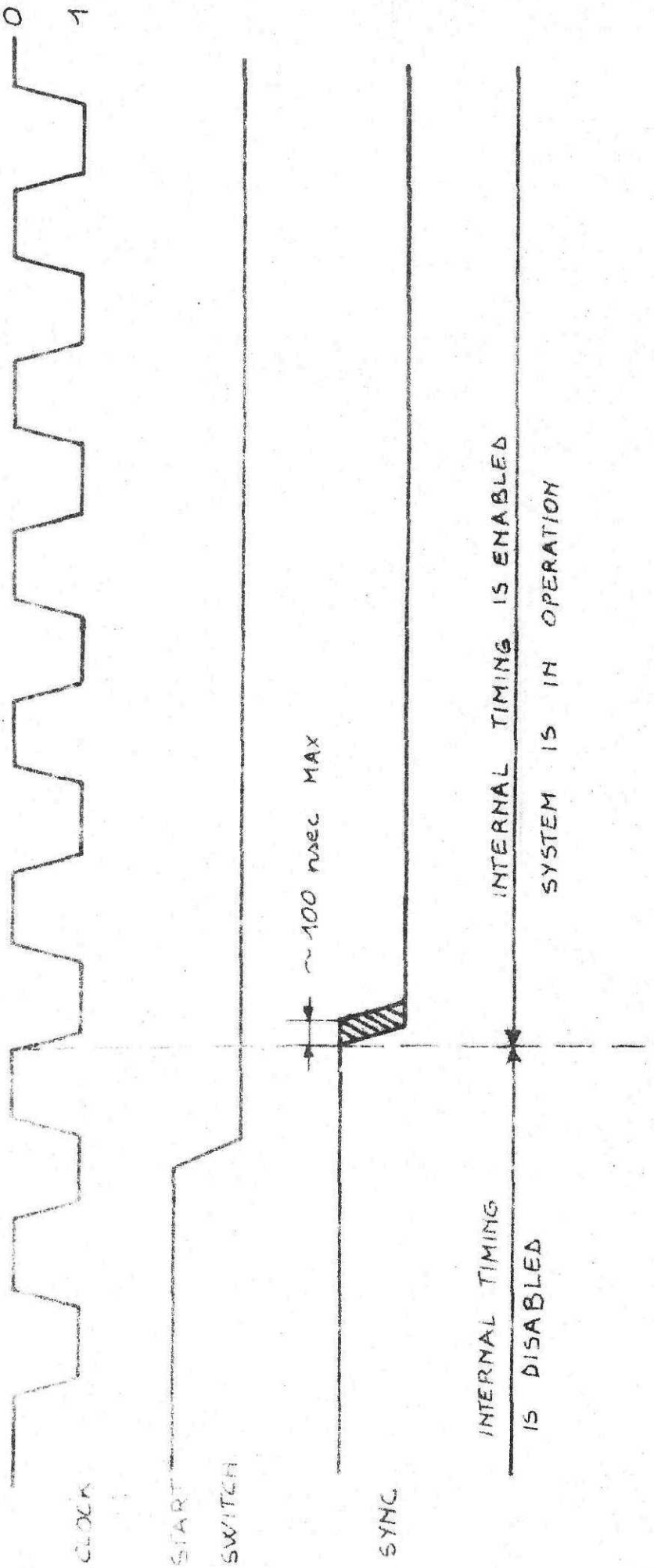
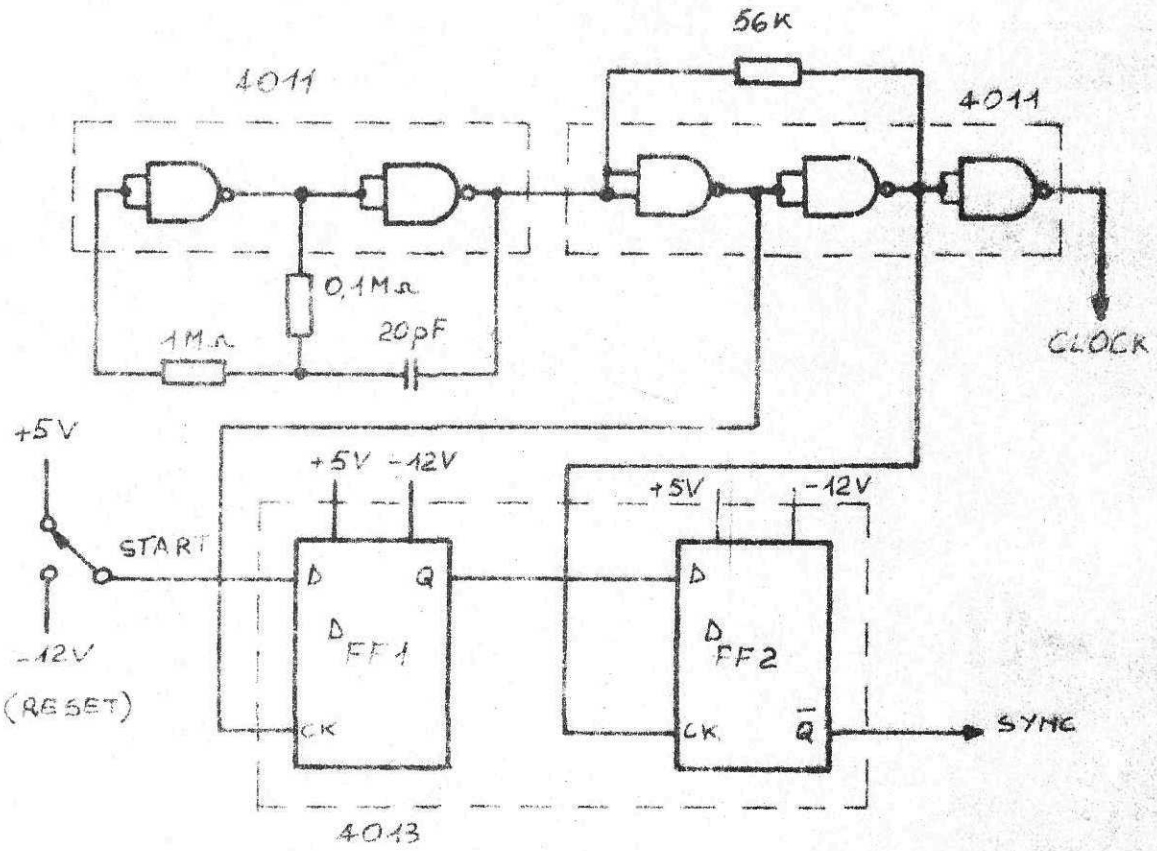


Fig. 2 - TIME RELATIONSHIP BETWEEN CLOCK, START SWITCH, SYNC SIGNALS



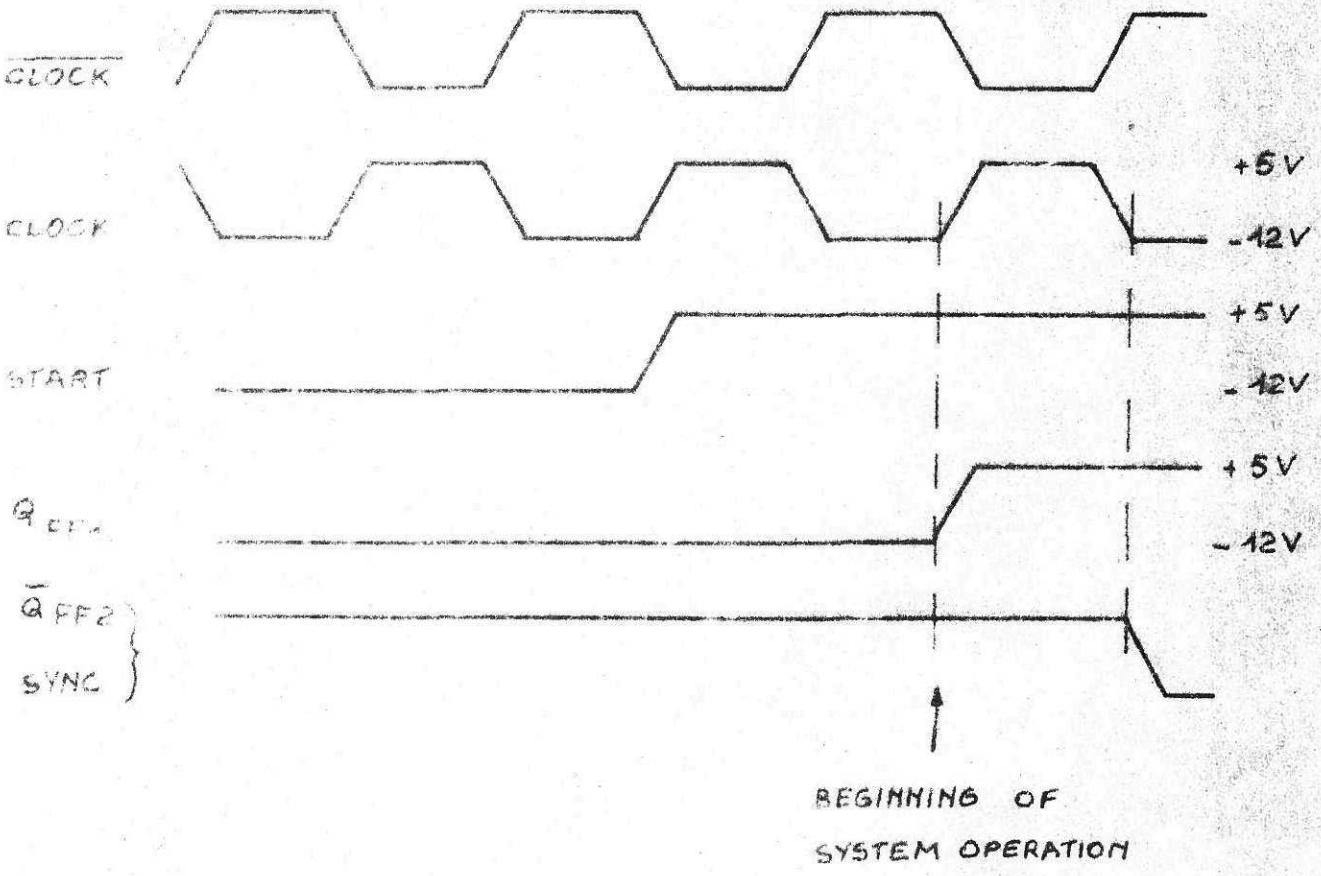


FIG 3 - CLOCK AND START CIRCUIT

MODULE CODE ASSIGNEMENT

It is already clear from the previous discussion that the system is divided in elementary blocks or modules, and that to enable a specific module to perform some operation the CPU has to send on the address bus the code of the module itself.
To have a better understanding of these module codes let's look how the user has to proceed when designing a system:

1 - The first step of the design stage is to draw a rought configuration of the hardware needed to solve the problem.
Of course, this configuration may be different from the final one, but, due to the modularity of the system, the successive refinements can be easily done without requiring the full ride sign of the system.
So at the end of this preliminary design, the evaluated hardware configuration could be, for example:

- CPU
- 3 K of ROM
- 1 RAM
- 2 I/O ports

2 - The next step will be the module code assignement to every component.
For the previous configuration this could be a possible assignement:

- 2 K ROM	000	000
	000	001
	000	010
	000	011
	000	100
	000	101
	000	110
	000	111
- 1 K ROM	001	000
	001	001
	001	010
	001	011

./..

These are unusable codes	001	100
	001	101
	001	110
	001	111
- RAM	010	000
- I/O N.1	111	000
- I/O N.2	111	001

Let's make some comments about this code table.

1 - The ROM's have been divided in blocks of 2 K. It may happen that the last block is less than 2 K.
At every 2 K block, it has been assigned a set of contigous codes which can be identified only by the 3 most significant bits.
On our example the first 2 K ROM is identified by the codes ØØØ XXX (X is for "don't care").
The second 2 K ROM is an incomplete block (only 1 K is needed): the codes assigned to this block are ØØ1 XXX, but only the codes:

001	000
001	001
001	010
001	011

are meaningfull for the system.
The other four codes are not used, and are unusable for either I/O ports or RAM's, because the hardware of the system is addressing the 8 ROM modules per time, so either the corresponding modules are on the system or are left out without using them for any other purposes.

./..

2 - The choice of ØØØ XXX for the first 2 K of ROM is not casual.
As it will be discussed in more details later, the hardware of the system is done in such a way to force the module address Ø immediately after the START command.
So the program which is to be executed first, must be on a ROM having module code Ø.
This ROM module must always be included on the system.

3 - The next available module code on the example, is 010 000 and it has been assigned the RAM block.

4 - The codes assigned to the I/O ports 1 and 2 are:

111 000
111 001

The codes for the I/O ports could have been for examples:

010 001
010 010

which are immediately following the RAM code.
The reason to prefer the first two codes, is that the instruction set can address directly I/O ports with the code included on the range:

111 000 to 111 111

while I/O ports with module codes out of that range can be addressed only indirectly. This will be more clear after the discussion about the instruction set.
Please note that the code 111 111 has been assigned by the hardware into the CPU, to the I/O port of the CPU itself.

./..

5 - The I/O instructions which allow a direct addressing of the I/O ports will refer to these I/O ports by the codes from Ø to 7. The CPU will generate by hardware the proper code as indicated by the following table:

Software Code	Hardware Code
Ø	111 ØØØ
1	111 ØØ1
2	111 Ø1Ø
3	111 Ø11
4	111 1ØØ
5	111 1Ø1
6	111 11Ø
7	111 111

For the indirect address, there is no difference inbetween the software and hardware I/O code.

When the system of the example will be in operation, the CPU will generate the following binary codes on the address bus:

A - ØØØ on the most significant bits, to address the first 2 K of ROM.

B - ØØ1 on the most significant bits, to address the third K of ROM.

C - ØØ1 ØØØ to address the RAM.

D - 111 ØØØ to address the I/O port N. 1.

E - 111 ØØ1 to address the I/O port N. 2.

./..

Let's now summarize the more important points about the module code assignement:

1 - Divide the ROM in block of 2 K by 2 K. The last block may be incomplete.

2 - Assign to every block a set of 8 codes which must be identified by the 3 most significant bits of the address bus.

3 - Assign to every RAM block one full code.

4 - Assign to every I/O port one full code on the range 111ØØØ to 11111Ø.

5 - The code for the I/O port of the CPU is assigned by hardware and is 111 111. Then this code cannot be used for any other module.

6 - In case the I/O ports are more then 8, a full code (out of the 111 000 ÷ 111 111 range) must be assigned to some of them. These I/O ports can be addressed only indirectly.

And now a small limitation on the module code assignement will be discussed.
The limitation refers to the module code for the I/O port on the RAM devices (M 383).
This code can be programmed by four strap lines only on the least significant bits: the bits 6 and 5 of the module code are assumed to be 1 by the hardware inside the chip.
This means that the possible code range is from

11Ø ØØØ to 111 11Ø

in binary or from 48 to 62 in decimal representation.

./..

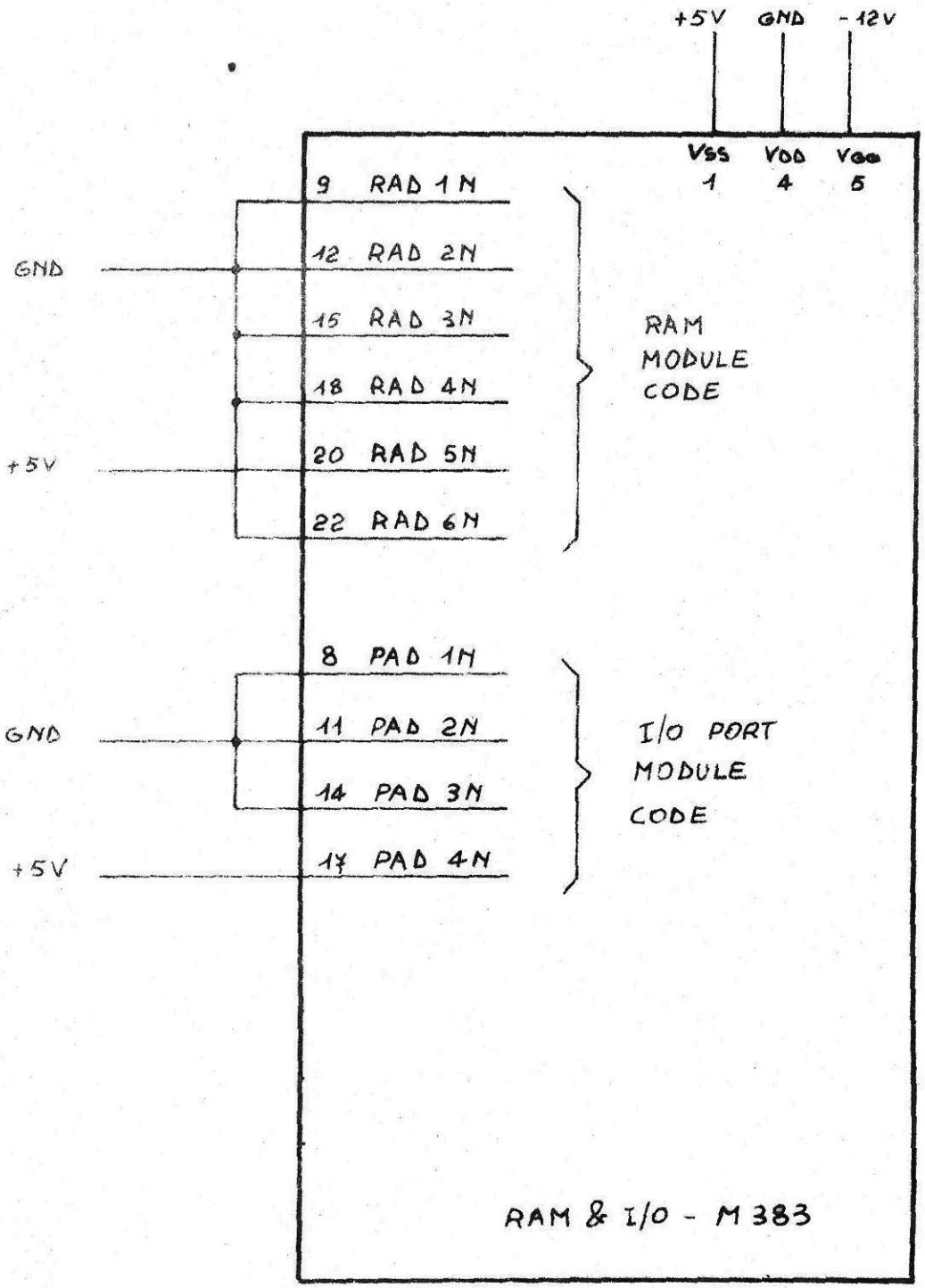
MODULE CODE IMPLEMENTATION

The implementation of the assigned module codes into the hardware of the system is done in two different ways depending on the device type (ROM or RAM) the code refers to.

A - ROM device (M 382 - M 381) -
On this case the module codes for the ROM or the I/O ports will be masked into the device at the same time when the program is masked.
Information about the proper code for every device must be supplied to SGS-ATES together with the ROM truth table, according to the format specified on the electrical specifications.

B - RAM device (M 383) -
The module codes for the RAM block or the I/O port included into this device must be handled directly by the user by means of the strap lines provided on the device.
For the RAM block, 6 lines are available, while for the I/O port 4 lines only are available.
The last limitation has been already discussed on the previous paragraph.
Please note that a logic "1" into the code means a connection of the corresponding pin to the most positive voltage, while a logic "Ø" means a connection to the most negative voltage.
Figure 4 is an example for the hardware connections needed for code 010 000 assigned to the RAM and for code 111 000 assigned to the I/O port of a M 383 device.
Please note again that the code connections for the I/O port must be done only for the 4 least significant bits.

./..



STRAP LINE CONNECTION FOR

1 - RAM MODULE CODE = 010 000

2 - I/O PORT MODULE CODE = 111 000

THESE TWO BITS ARE HARDWIRED INTO THE DEVICE

Fig. 4

HOW ADDRESS BUS AND MODULE CODE WORK

The operation of the address bus and the module code is shown on figure 5.

The ROM device has a logic inside where the masked module code is compared against the three most significant bits of the address bus.
When the two information are the same, an enable signal becames true into the chip and a proper action will take place, according to the control lines commands.

For the I/O ports and the RAM devices, the comparison between the code (masked or wired throught strap lines) and the address bus is extended to all the 6 bits.
Again, when the two informations are the same, the enable signal becomes true and the required operation will take place on the enabled module.

./..

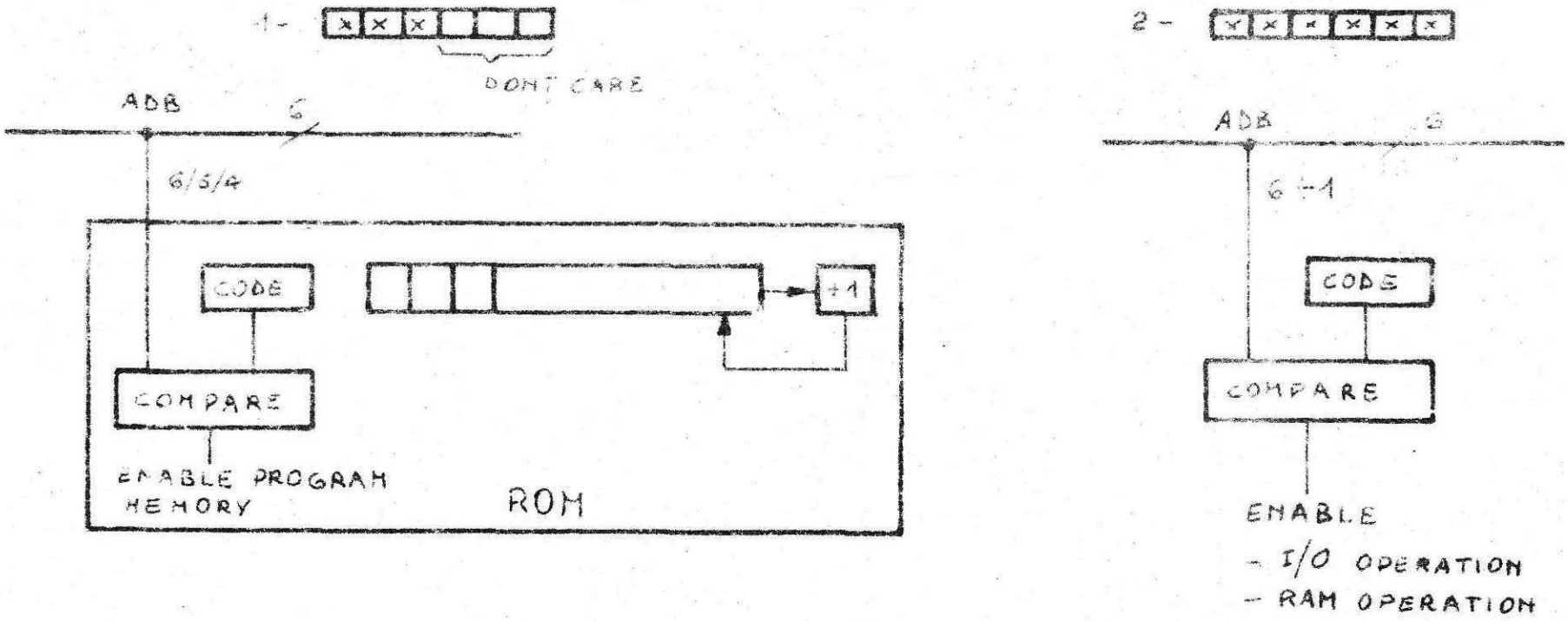


FIG 5 - HOW ADDRESS BUS AND MODULE CODE WORK

THE PROGRAM COUNTER

The program counter has been implemented in every ROM chip, and it is a register 11 bit long.
This allows a direct addressing on 2048 bytes, which are equivalent to 8 modules.
As previously discussed, these modules must satisfy the following conditions:

A - They have to be contiguous.

B - The set of the 8 modules must be identified by the 3 most significant bits of the address bus.

On figure 6 a 2 K ROM configuration is shown.
This requires two M 382 chips with the same module code masked into the ROM CODE area.
It follows that the "COMPARE" logic will generate a TRUE signal in both chips when the address bus signals on the three most significant bits are the same as the "ROM CODE".
The selection between the low range modules (address Ø to 1777_8) and the high range modules (address 2000_8 to 2777_8) is done by the most significant bit of the program counter.
On the "low range" chip this bit is inverted before going into the "ENABLE" gate, while on the "high range" chip there is a straight connection between this bit and the "ENABLE" gate.
This connection, either inverted or straight, is done by metal masking as for the "ROM CODE" and the ROM program.

During the program execution, only one of the two ROMS is enabled, and its content is sent on the date bus, but the program counters on both chips receive the same commands and have inside the same binary information.

./..

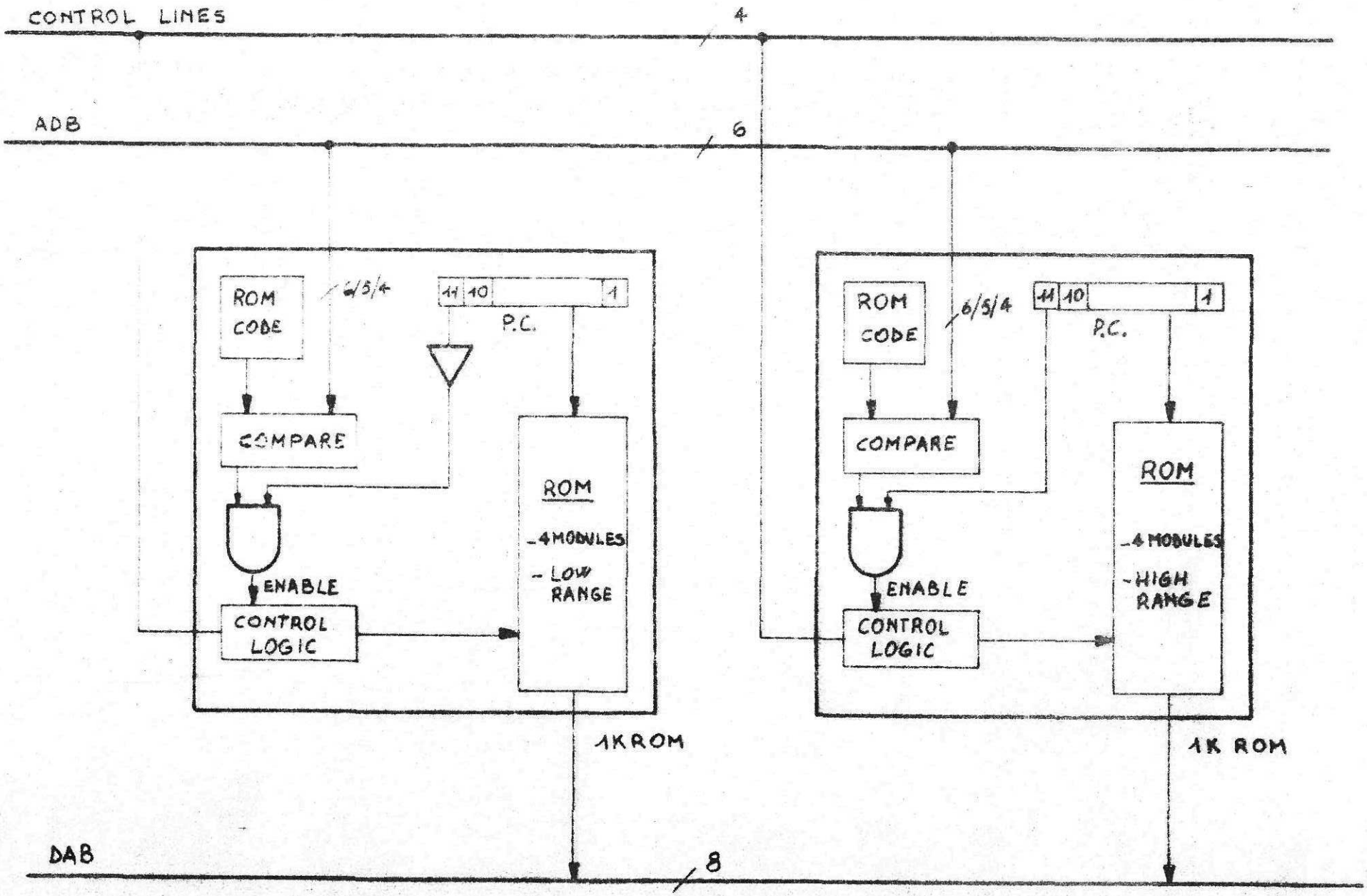

CONTROL LINES
4
ADB
6
ROM CODE
6/5/4
11 10 1
P.C.
COMPARE
ENABLE
CONTROL LOGIC
ROM
-4 MODULES
- LOW RANGE
1K ROM
ROM CODE
6/5/4
11 10 1
P.C.
COMPARE
ENABLE
CONTROL LOGIC
ROM
-4 MODULES
-HIGH RANGE
1K ROM
DAB
8

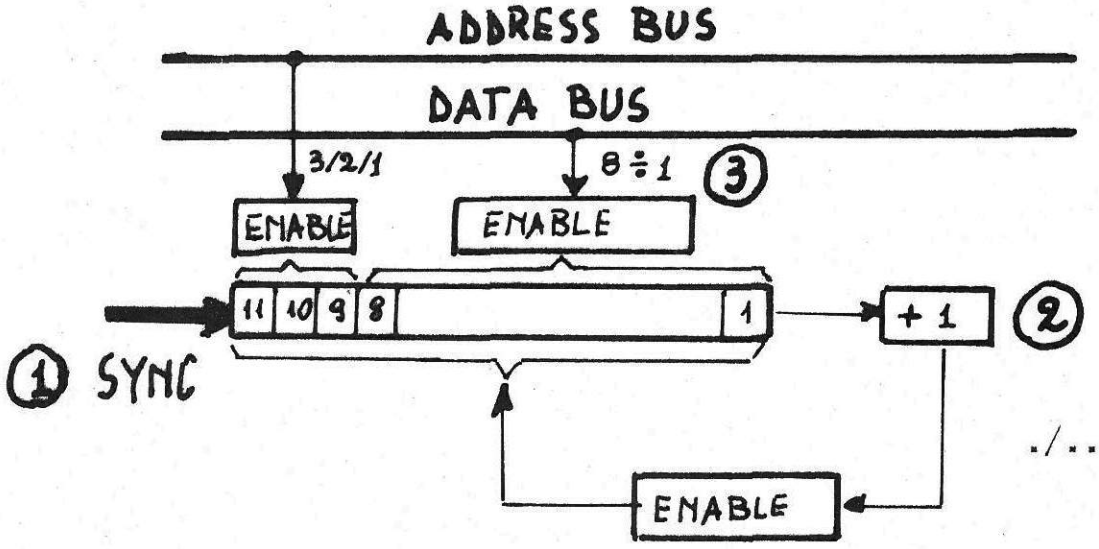
Fig. 6 2K ROM

The program counter content may be changed in three different ways (fig. 7):

1 - By the SYNC command.
As long as the SYNC line is kept to the most positive voltage the system timing is locked and the every program counter of the system is filled of Ø's.

2 - By the increment command.
After the fetch cycle of an instruction, the CPU is sending out an increment command throught the control lines.
This command will be effective only into the ROM chips whose "ROM CODE" is coincident with the three most significant bits of the address bus, and it will increment by one the program counter of those chips.

3 - By the jump command.
Some instructions (the jump instructions) during the execution need to change the program counter content.
This is accomplished by the CPU which will send the new program counter content on the three most significant bits of the address bus and on the date bus (see fig.7), together with the jump command on the control lines.
As for the increment, the jump command will be enabled only by the enabled ROM or ROMS.



./..

FIG 7

AN EXAMPLE: WHERE TO START.

The following paragraphs will describe the main steps throught which the user normally has to go in designing a microcomputer system.

Of course, this will be just a review without entering to much into a detailed discussion.

Let's suppose that the application to be worked out is a controller for a magnetic tape cassette.

The basic structure of the controller is shown on figure 8: on the left side of the controller there is the "MAIN SYSTEM", generating the commands or orders for the controller and sending or receiving data; on the right side there is the magnetic tape cassette operated by the controller.

The point to begin with is to define the specifications.
On our example the following characteristics are of main concern:

1 - Magnetic tape speed.

2 - Basic commands for the cassette (forward, reversed, fast, normal, read, write, start, stop).

3 - Theinterface signals necessary for the hardshake protocol.

4 - The orders or commands coming from the main system (read, write, tape mark, rewind, Back space, erase) .

5 - Other features (read after write technique, phase encode methode for writing, operation according to ECMA specifications)

./..

Once everything has been clearly stated, the hardware configu ration can be defined with a good approximation.
So, for the cassette tape controller we may end up with the schematic shown on figure 9.
This first approach could in some case require to write some parts of the program in order to check the feasibility of the system.
The critical points for the controller were:

1 - The CRC generation and control.

2 - The bit transfer rate.

By a first evaluation it was apparent that (for the magnetic tape speed chosen: 756 char/sec), the CRC could have been do ne by software, but only if the data were transferred from the controller into the cassette by bytes (8 bits per time). This, of course, requires some hardware logic for the parallel to serial and for the serial to parallel conversion of the data.

./..

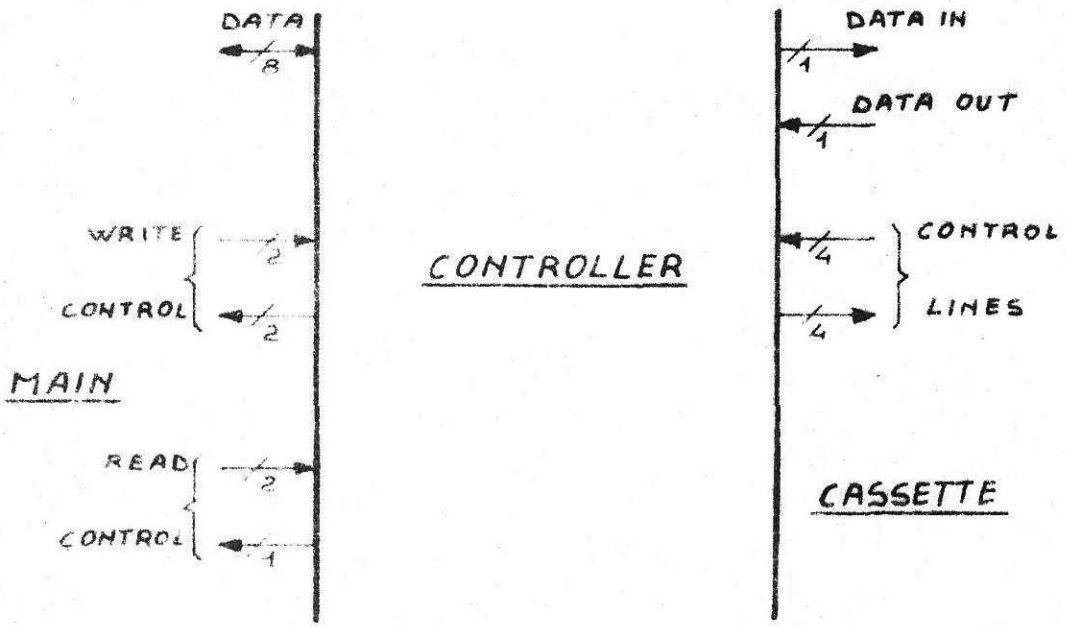


FIG 8 - BASIC STRUCTURE FOR MAGNETIC TAPE CASSETTE CONTROLLER

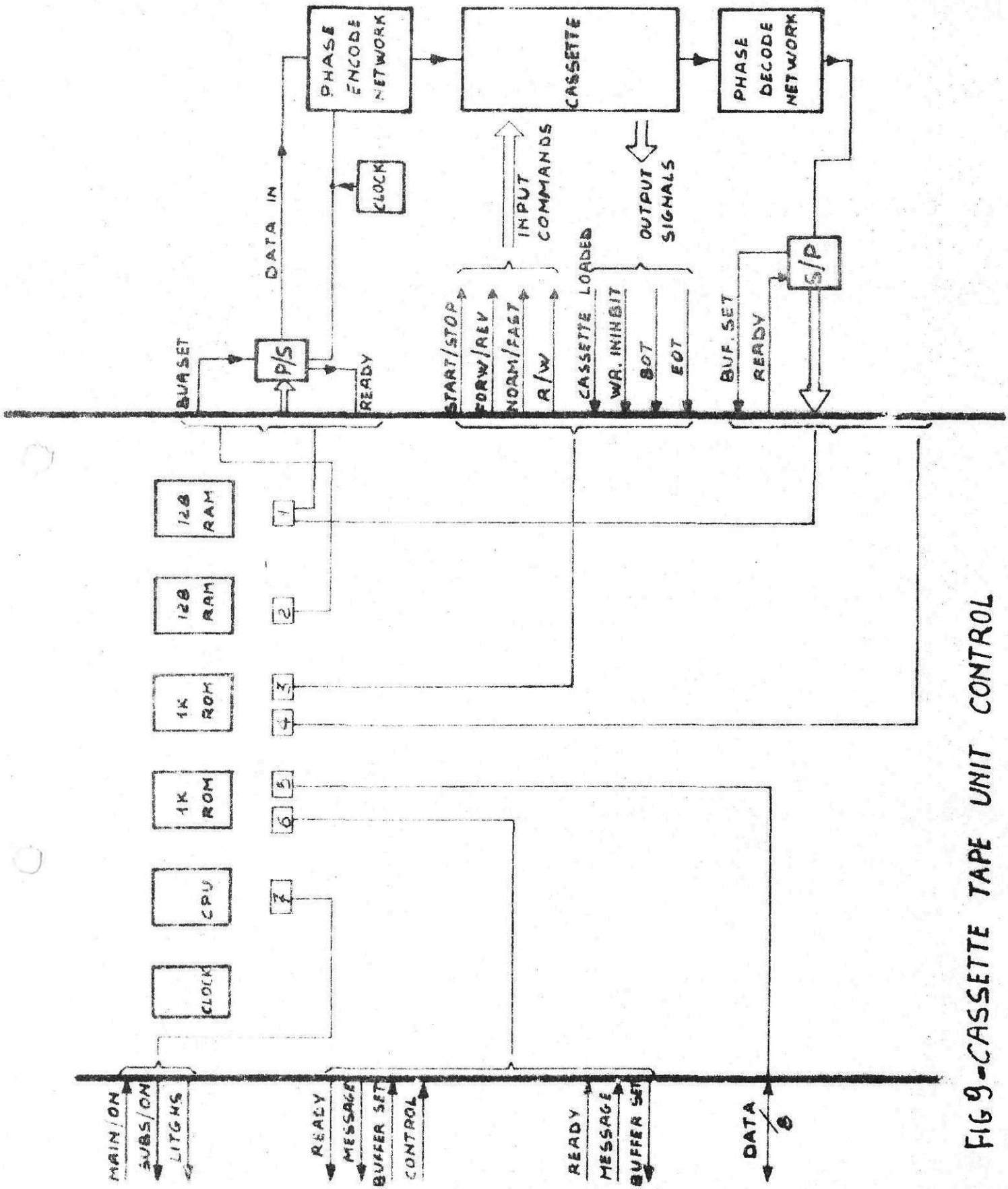


FIG 9 - CASSETTE TAPE UNIT CONTROL

8.0

AN EXAMPLE: MODULE CODE ASSIGNEMENT

We know now that the controller is made up by the following elements:

- CPU
- 2 K ROM
- 2 RAM (128 x 8)
- 7 I/O PORTS

and we have to assign to every element a proper code.
This could be the assignement:

COMPONENT	MODULE CODE	
	BINARY	DECIMAL
2 K ROM	ØØØ XXX	Ø to 7
1° RAM	ØØ1 ØØØ	8
2° RAM	ØØ1 ØØ1	9
1° I/O PORT	111 ØØ1	57
2° I/O PORT	111 Ø1Ø	58
3° I/O PORT	111 Ø11	59
4° I/O PORT	111 1ØØ	60
5° I/O PORT	111 1Ø1	61
6° I/O PORT	111 11Ø	62
7° I/O PORT	111 111	63

./..

AN EXAMPLE: PROGRAM DESIGN AND DEBUGGING

At this point the program design may start.
The specifications and the hadrware must be studied very carefully in order to avoid errors into the program.
The first thing to remember when working on a program is to get the program fully designed by a proper structure before coding it.
The top-down approach together with some structured programming tecniques may help very much in having a correct program.
Once the program has been designed and coded, it must be debugged.
It is known and accepted by everybody that it is impossible to test thoroughly a program.
Let's see which is the role the testing can play.
The first step to do is to get the object code from the simbolic code: on this stage only the lexical errors are detected.
They are the far less important and easy to recognize and remove.

Starting from a correct object code the following point should be satisfied:

1 - Every statement should be executed at least once.

2 - Every condition should be tested in every way.

3 - Every elementary conponent of the input data has to be read.

4 - Every elementary component of the output data has to be produced.

The debugging tools are:

- The assemblar
- The software simulator
- The hardware simulator

They will be treated in a special section.

./..

At the end of the debugging the correct program is available in different forms (listing, punched cards, punched paper tape etc...).
The listing for our example will look like the one on the table n. 1

TABLE 1

ADDRESS		CODE		LABEL	MNEMO	COMMENTS
DEC	OCT	OCT	EX			
Ø	Ø	ØØ4	Ø4	INIT	LAL 1Ø	INITIALIZE
1	1	Ø12	ØA			
2	2	Ø32	ØA		SAX	X REGISTER
=	=					
=	=					
=	=					
595	1131	Ø41	21		INP 1	INPUT FROM I/O 1
596	1132	—	—		— —	
=	=					
=	=					
2Ø46	3776	—	—		— —	
2Ø47	3777	—	—		— —	

./..

AN EXAMPLE: THE SYSTEM IN OPERATION

Let's now suppose to have built a prototype.
A part of it will look as the one represented on figures 10,11, where the fetch phase and the execute phase for the instruction INP 1 (input data from PORT n. 57) are shown.
The INP 1 instruction is located at the address 1131_8 , so the first 1 K of the program must be enabled, during the fetch phase.

Let's look carefully to the fetch phase (Fig. 10):

1 - The address bus has the following information ØØØ XXX.
This means that the "COMPARE" signal is true in both devices ROM 1 and ROM 2.

2 - The program counter content in both chips is 1131_8 or in binary

Ø1 ØØ1 Ø11 ØØ1

Due to the Ø into the MSB of the program counter the first chip (ROM 1) is enabled and the second (ROM 2) is disabled.

3 - The other 1Ø bits of the program counter are addressing the location 1131_8 where the code 41_8 is stored.

4 - This code is prought throught the data bus into the instruction register.

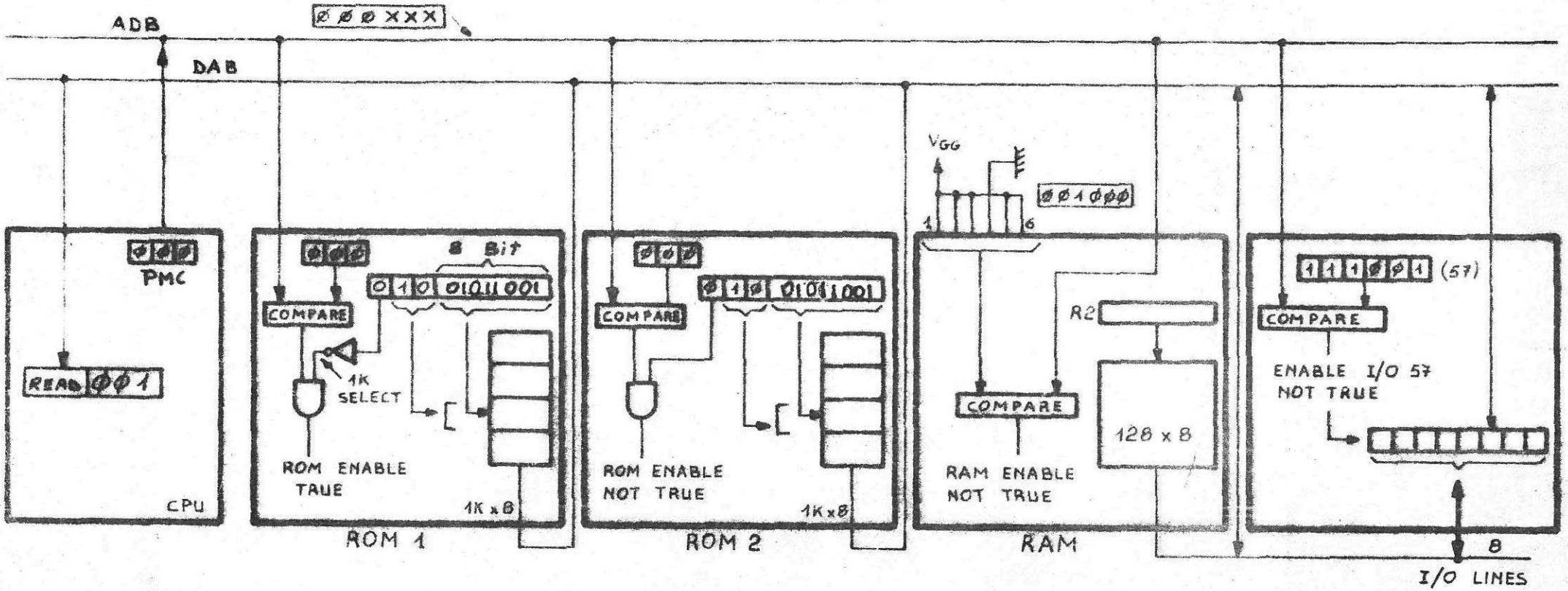
Now the execute phase will begin, and we will examine it (Fig. 11):

1 - The I/O port code is computed by the CPU starting from the software code (ØØ1 on our example) and adding to it the binary number 111 ØØØ.
The result is sent on the address bus.

./..

2 - The "COMPARE" signal of the I/O port 57 becomes true and the transfer between the I/O port and the data bus will take place.
The direction of the transfer is due to the control lines status which is a result of the decoded input instruction.

./..



SYSTEM CONFIGURATION

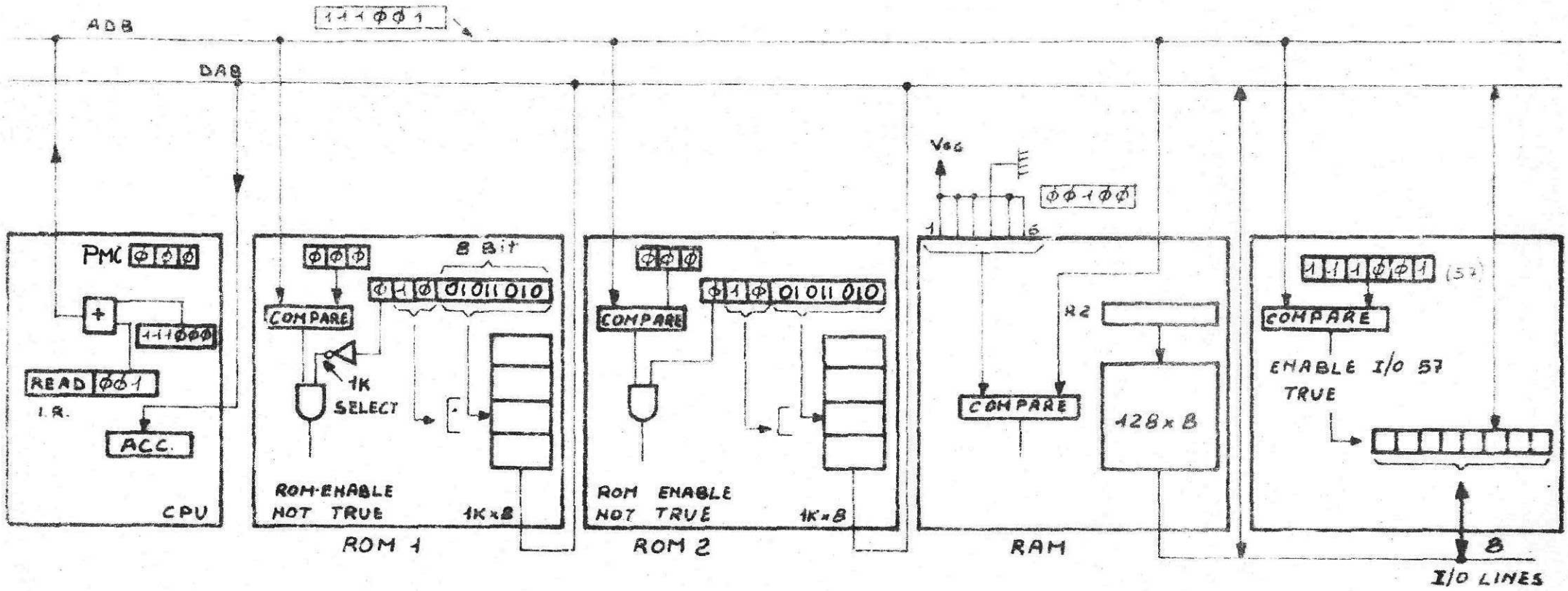
2K ROM - MODULE CODE ØØØXXX ADDRESS { Ø ÷ 1777_8 ($Ø \div 1Ø23_{1Ø}$) 1° CHIP; $2ØØØ_8 \div 3777_8$ ($1Ø24 \div 2Ø47_{1Ø}$) 2° CHIP

1 RAM - MODULE CODE ØØ1ØØØ $8_{1Ø}$

1 I/O PORT - MODULE CODE 111ØØ1 $57_{1Ø}$

- The ADB is pointing on 2K ROM
- The addressed word is on first K.
- Module address is 10 = 2
- Address is 01011001

FIG 10 - FETCH PHASE FOR 'JMP 1'



SYSTEM CONFIGURATION

2K ROM _MODULE CODE ØØØXXX

1 RAM _MODULE CODE ØØ1ØØØ 8_{10}

1 I/O PORT _MODULE CODE 111ØØ1 57_{10}

ADDRESS { Ø ÷ 1777_8 1° CHIP ; $2ØØØ_8 \div 3777_8$ 2° CHIP

- The ADB is pointing on I/O PORT N.57
- The module address is 57
- The program address has been incremented

FIG 11 - EXECUTE PHASE FOR 'INP 1'

BASIC TIMING

A basic instruction cycle requires 4 clock pulses, named respectively F1, F2, F3, F4.
This timing is generated internally to every chip by using the clock signal.
The sincronization of the timing signals in every device is done by means of the SYNC pulse, as described before.

The tecnique used throughout the system to transfer binary information is outlined briefly, before the discussion about the timing phases.
Any transfer of information (data or instruction codes) whichever is the direction of the transfer itself, is performed in two different steps:

A - First the binary code is stored on the data bus.

B - Then it is brought into the memory cells of the device enabled to accept the information.

So, for example, during the fetch phase, the instruction code addressed by the program counter belonging to the enabled module, is set on the data bus.
In a subsequent time, this code is copied from the data bus into the CPU instruction register.
As another example, if the instruction execution requires to store the accumulator into an enabled RAM, the accumulator is first stored on the data bus and at a later time the data bus is copied into the RAM.

Let's see now the basic operations performed by the system during the four phases:

./..

1 - Phase F1.
During that time, the data bus is transferred into the enabled Flip-Flops.
As it is shown by the schematics of figure 12, the voltage level stored on the C_n capacitors during F4, is transferred to the Flip-Flop inputs if the enable signal has been simoultaneously generated by the logic.

2 - Phase F2.
During F2 the data bus is prechanged to the Ø logic level (to the most positive voltage).
At the same time the control signals are generated by the CPU.
They are valid approximately 300 nsec after the leading edge of F2: their status is a function of the instruction code read into the instruction register during F1 or, in case of a multiple cycle instruction is a function of the current cycle.
These control signals are needed through out the system to enable the operation or operations to be performed during the current machine cycle.

3 - Phase F3 and F4 are the phases during which the code to be transferred is written on the data bus.
This code may come or from some flip-flops or from the program memory.
The transfer timing is different depending if the information is coming from the CPU or from any other part of the system.
On the first case (CPU to DAB), the transfer will begin by F3, while on all other cases (I/O to DAB, RAM to DAB, ROM to DAB) the transfer will begin by F4.

./..

The schematic on the upper part of figure 12 shows the hardware related to one line of the data bus.

A - The one bit structure of the CPU is a typical interface structure between DAB and CPU: the output buffer may have at the input a data from the arithmetic logic unit or from a temporary register, while the input memory bit may be an instruction register bit or a temporary register bit or an accumulator bit.

B - The interface structure between DAB and ROM or RAM devices is also shown.
On this case the output buffer may have as an input either a data to be transferred on the data bus, or a data from the program memory (i.e. ROM) or from the RAM, or from an I/O Port.
The input memory bit may be an I/O Port bit or a program counter bit, or a RAM bit.

Let's suppose that a transfer from a RAM into the CPU has to be done.
Then, the following sequence of operations will take plase:

- During F3 and F4 the Flip-Flops of the system are refreshed.

- During F4 an enable is generated into the addressed RAM and the data from that RAM are stored on data bus and on C_n capacitors, through the RAM output buffer.

- During F1 there will be an enable only for the input Flip-Flops of the CPU.
The voltage levels which are on the C_n capacitors of the CPU are transferred into the Flip-Flops: their previous content is then lost.

./..

- During F2 the data bus is prechanged to the Ø logic level.

If the transfer to be performed is from the CPU toward an I/O Port, the system will go through a similar sequence:

- During F3 and F4 the Flip-Flops of the system are refreshed.

- At the same time (F3 + F4) the enable signal becomes true on the output buffer of the CPU and the data connected to that buffer (out signals) are transferred on the DAB.

- During F4 the same binary information is brought to the C_n capacitors.

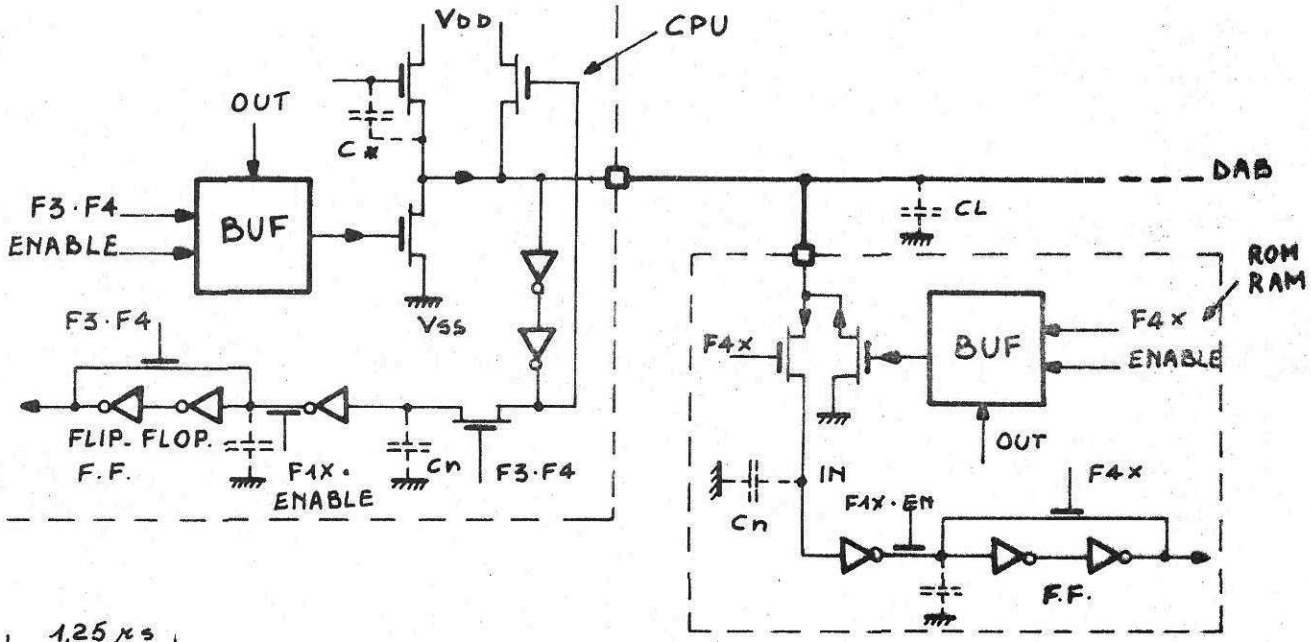
- During F1 there will be an enable only for the input flip-flops of the addressed I/O Port, and the voltage level which are on their C_n capacitors are transferred into the flip-flips of that I/O Port.

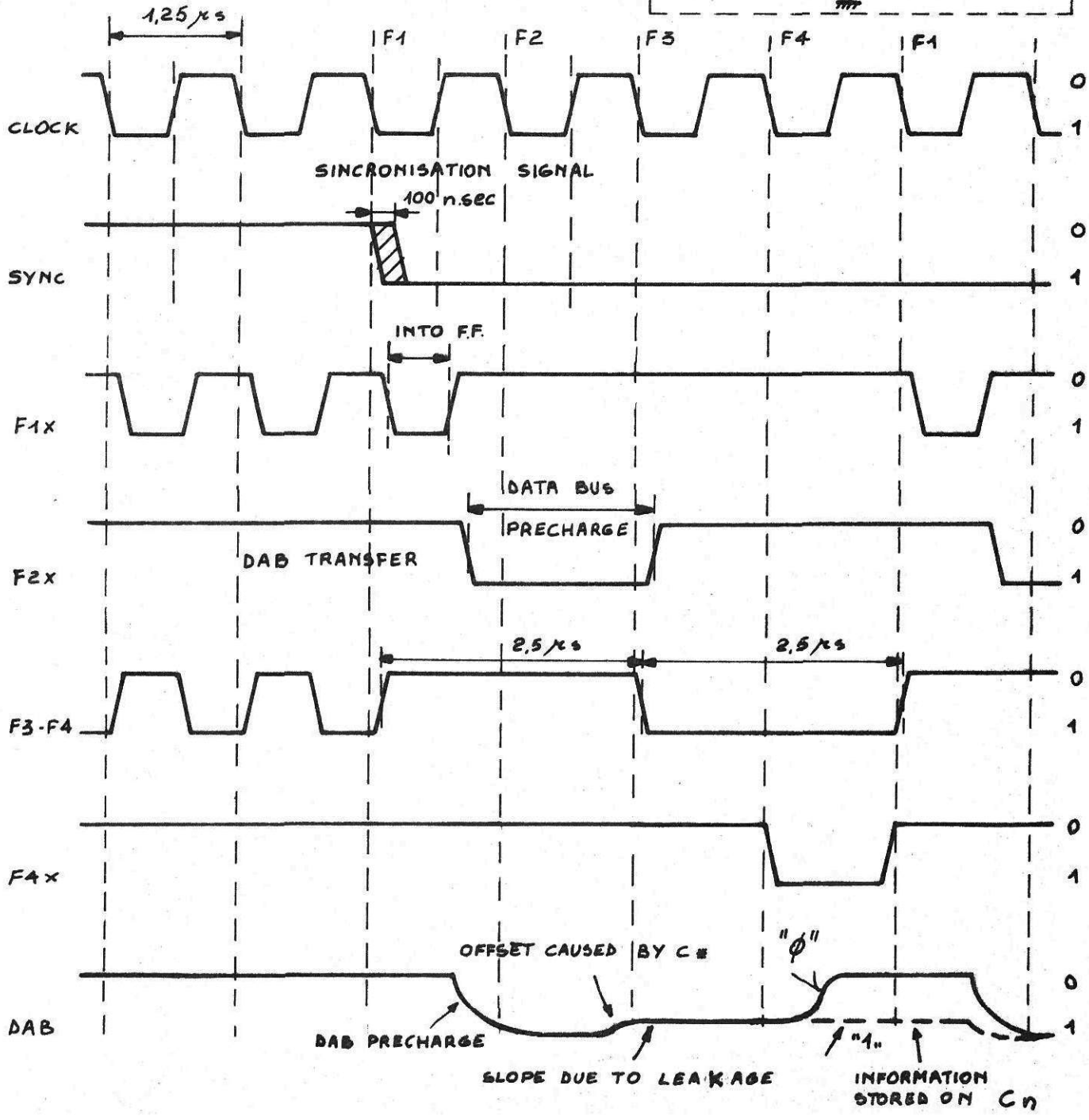
- During F2 the data bus is prechanged to the Ø logic level.

./..

SGS - ATES COMPONENTI ELETTRONICI S. p. A.

FIG 12 - BASIC TIMING





ONE CYCLE INSTRUCTIONS

Many instructions of the M 38 system can be executed in one basic machine cycle.
The timing for a one cycle instruction is shown in figure 13.

Let's assume the following system initial conditions at the beginning of the X cycle:

1 - The PMC register is connected to the ADB (three most significant) and it has the same code as the module code of the program memory shown of figure.
This means that the system is executing the program inside of that module.

2 - The program counter content of the enabled program memory is N, after the phase F2.
This means that the INSTR A is addressed.

The initial conditions of any other register (for example I.R., ACC.....) are not important for our discussion.

These are the successive operations performed by the system:

1 - X CYCLE:
- During F2 the DAB is preghanged. The control lines are set to Ø: this is interpreted by the ROM as an enable to send out the addressed word.
- During F4 the fetch of the INSTR A is done: the INSTR A code is brought on the DAB.

2 - (X+1) CYCLE:
- During F1 the program counter is updated to N + 1: the command for that increment is brought into the ROM by the control lines.
At the same time the "INSTR A" code is transferred from the DAB into the instruction register.

- During F2 the execution of the "INSTR A" begins, the DAB is prechanged and the control lines are set to the proper value for the next command.
  On this example they stay to Ø and this again is interpreted by the ROM as an enable to send out data on the DAB.
- During F4 while "INSTR A" is still in execution the fetch for "INSTR B" is performed.

3 - (X+2) CYCLE- The system proceeds on the same way as described before.

By analising the time diagram shown on figure 13, it is important to make the following remarks:

1- For a one cycle instruction, the system overlaps the execution of one instruction (example "EXECUTE INSTR A") by the fetch of the next instruction ("INSTR B" read on DAB).

2- This is possible due to the fact that the execution of "INSTR A" does not need to access neither the address bus nor the Data bus.
In other words, the one cycle instructions do not require to access data out of the CPU.

3- Due to the overlapping of fetch for the next instruction and the execution of the current instruction, the final result is that only one machine cycle is needed for this kind of instructions.

4- The control lines are carrying throughout the system some commands which an performed at a different time during the machine cycle.

./..

For example, the status ØØØØ on these lines has two different effects: ROM output enable (during F4) and Program counter increment (during next F1).

./..

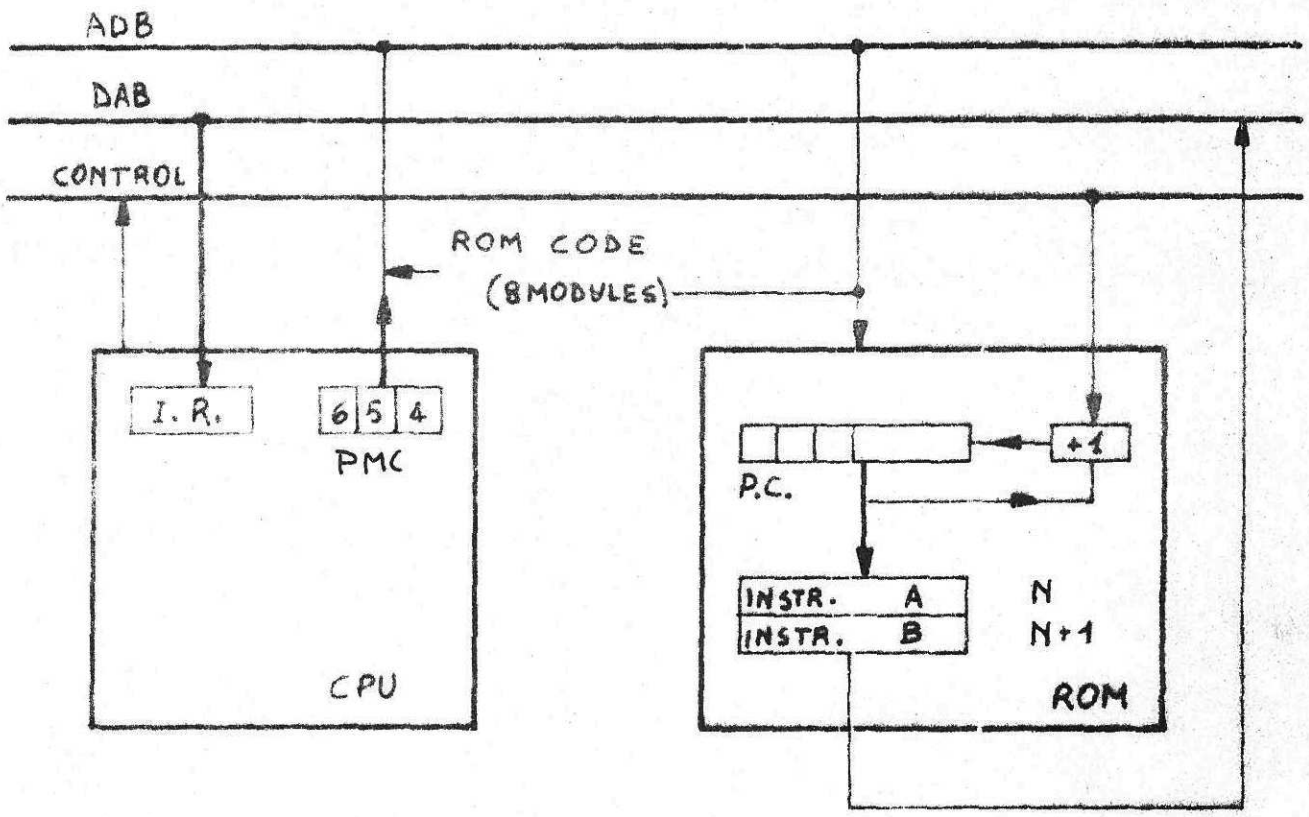


Fig 13A - ONE CYCLE INSTRUCTION

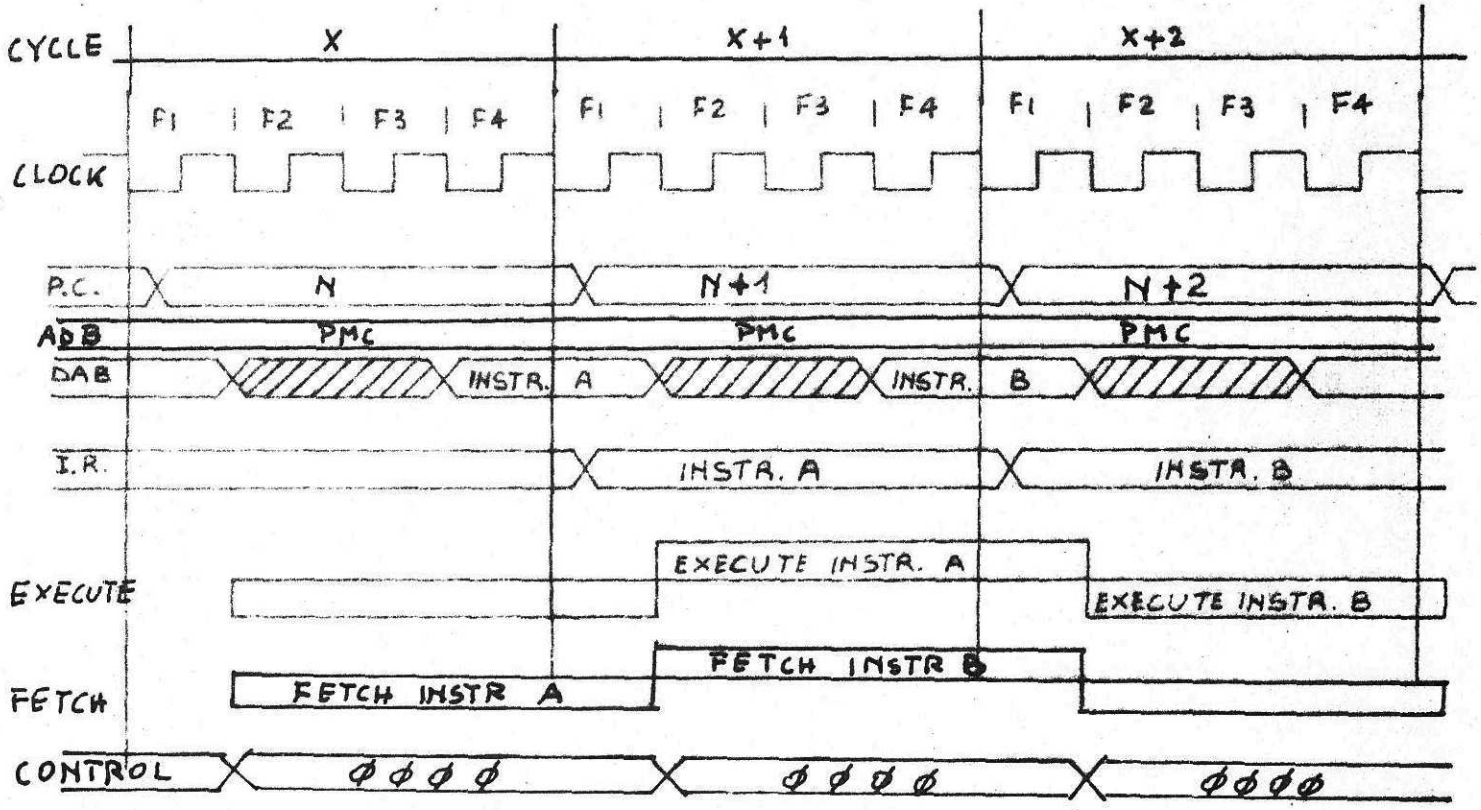


FIG. 13 B - TIMING FOR A ONE CYCLE INSTRUCTION

MULTIPLE CYCLE INSTRUCTIONS

Instruction which require to handle data not included on the CPU, are performed in more than one machine cycle.
An example of such a kind of instructions is represented on figure 14, where the "INP" instruction is illustrated.
The instruction INP C performes a data input into the accumulator from the I/O Port whose module code is C.
At the end of the execution, the status of the 8 lines connected to the I/O Port "C" is copied into the accumulator.

By starting with the same initial conditions defined for the one cycle instruction, let's analyse step by step how this transfer is done:

1 - X CYCLE:
- DAB is precharged during F2.
- The INP C code is brought on the DAB at F4.
  This is caused by the control line status (ØØØØ) and by the address bus (connected to the PMC register): in this case the control lines are responsible of the data transfer toward the DAB while the address bus is responsible to get the data from the program memory.

2 - (X+1) CYCLE:- At F1 the INP C code is copied from the DAB into the I.R.
  At the same time the program counter is incremented: this increment is due to the control lines status (ØØØØ) and to the fact that these lines are effective on the program memory logic (the address bus is connected to the PMC register during this time).

./..

- During F2 the execution of the instruction INP C begins by connecting on the address bus the I/O Port module code C. The control lines are set the proper value by the instruction: on this case they will stay at ØØØØ.
  This control line status will be interpreted later on (at F4) by the enabled device (the I/O port C) as a command to send its data on the DAB.
  Always during F2, the DAB precharge is performed.

- During F4, the I/O Port C lines are copied on the Data bus.

3 - (X+2) CYCLE:- At F1 the execution continues by transferring the DAB into the temporary register of the CPU and by connecting the temporary register to the ALU.
The program counter is not incremented now, inspite of the fact that the control lines are at ØØØØ status, because the enabled module is an I/O Port and not the program memory.

- During F2 the address bus comes back to the standard connection to the PMC register.
  The control lines are set again to ØØØØ.
  By these two conditions the ROM-DAB interconnection is again enabled and it will of course become effective during F4.
  As usual the DAB precharge is now performed.

- During F4 the new instruction code appears on the DAB while the data read from the I/O Port is going through the ALU.

./..

4 - (X+3) CYCLE:- During F1 the execution is completed by storing into the accumulator the data coming out of the ALU.
The program counter is incremented and the new instruction code is accepted into the Instruction Register.

Let's make now few remarks:

1 - Also on this case there is a part of the execute phase which is overlapped by the fetch phase of the next instruction.
The result is that such a type of instruction needs as a whole only two machine cycle.

2 - The control lines signals generated by the "INP C" instruction are the ones of the (X+1) and (X+2) machine cycles.
During the (X+1) cycle, they are referred to the I/O Port C while during the next cycle they are referred to the program memory.
The reference is set by the address bus.

./..

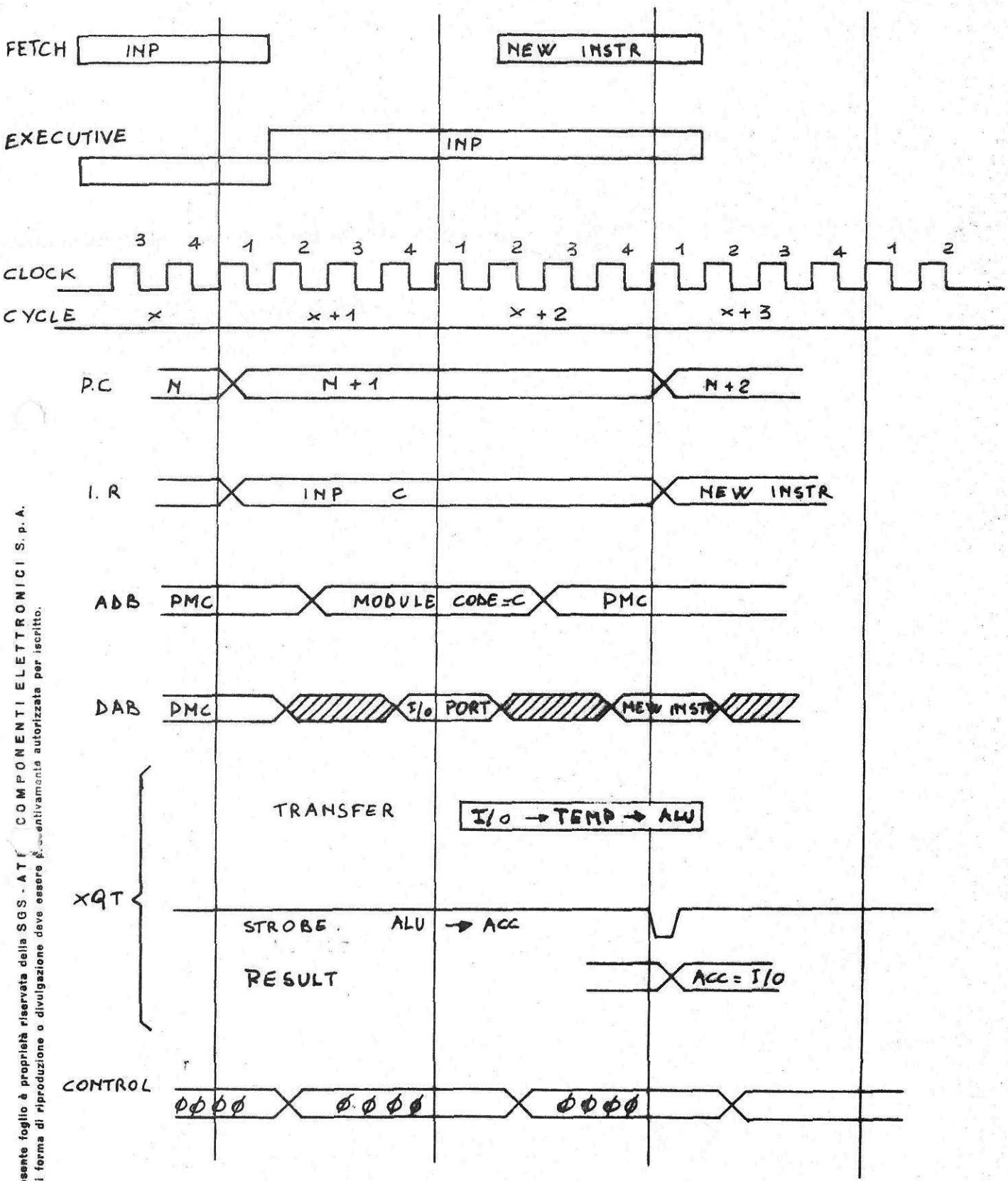


FIG. 14A – TIMING FOR 'INP' INSTRUCTION

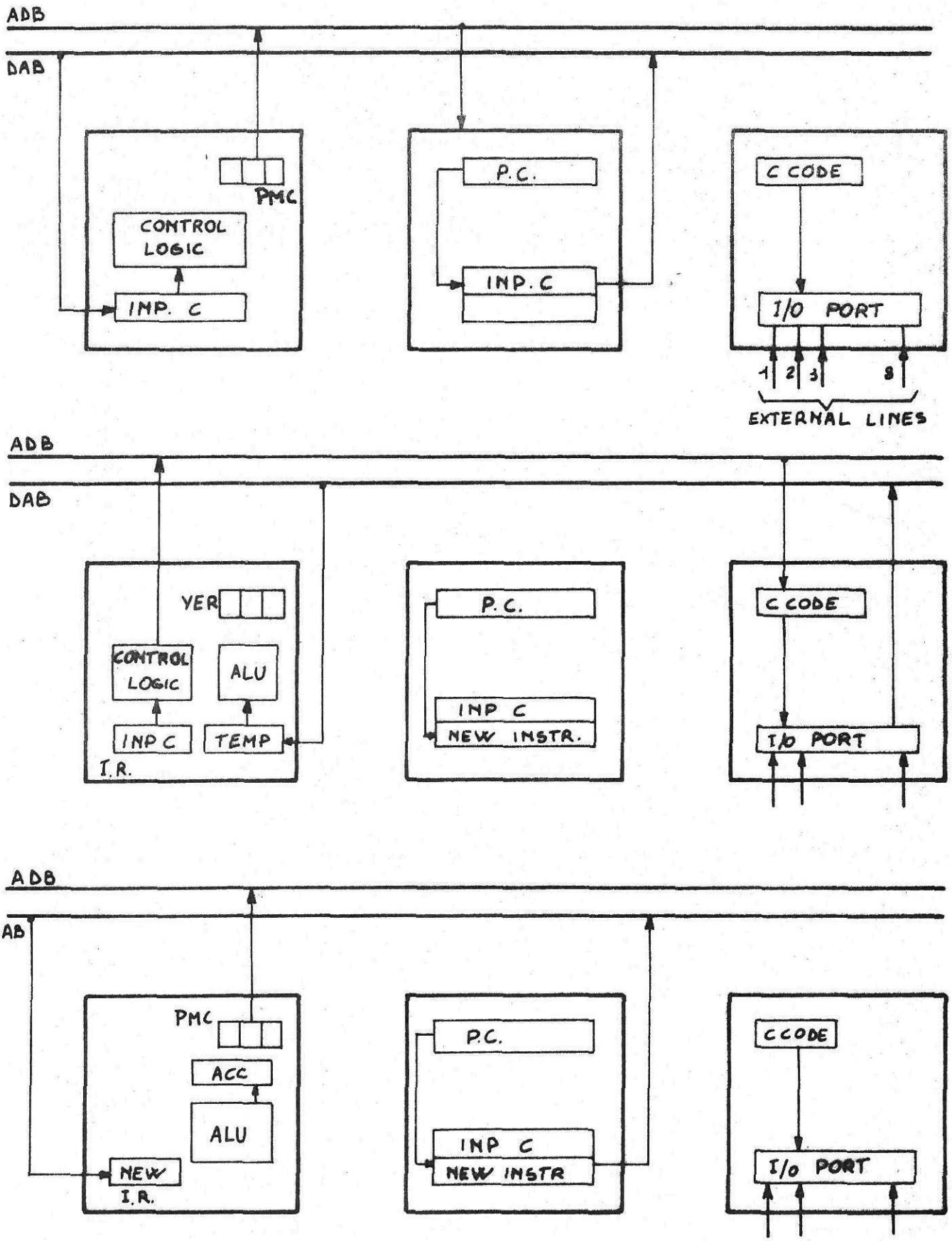


FIG. 14B 'INP' INSTRUCTION

OVERLAPPING OF THE FETCH AND EXECUTE PHASE

As it has been shown by the last two paragraphs, the microprocessor overlaps the execution of the current instruction with the fetch of the next instruction whenever possible.
Both the examples which have been discussed, have the overlap feature, but not all the available instructions are allowed to overlap.
Figure 15 shows the full picture of the instruction set.
The instructions having the same timing are grouped together.
Let's now summarize here the main characteristics of the system timing:

1 - Any instruction execution goes through two basic phases:
- Fetch phase. The instruction is read from the program memory into the instruction registers.
- Execute phase. On this phase the internal operations by the ALU or the needed external operation on data and address bus are performed.

The fetch phase requires one machine cycle (5 μ sec).
The execute phase requires one or more machine cycles (max. 3).

2 - When the last cycle of the execute phase needs only data inside of the CPU, the data and address bus are available for the fetch phase of the next instruction.
This overlapping makes the effective execution time shorten then the nominal (fetch + execute) time.

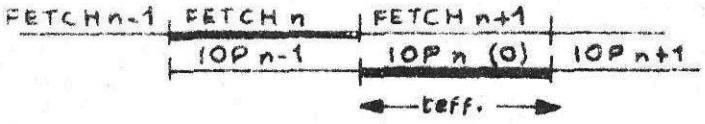
./..

3 - The diagrams on figure 15 show the instruction set grouped by instruction types, and the possible overlappings of the current instruction with the previous and the following instruction.
The nomenclature used on the diagram is as follows:

- Fetch: A machine cycle during which the instruction code is read from the program memory into the instruction register.
- IOP : (Internal operation): A machine cycle during which an instruction is executed by using only the ALU and the registers inside of the CPU. There is no access to the data and the address bus.
- EOP : (External operation): A machine cycle during which an instruction is executed by using the data and address bus.
- t_{eff} : The effective time to fetch and execute an instruction.
- n, n-1, n+1: These are subscripts to indicate the current instruction, the previous and the next instruction.
- The numbers between parenthesis indicate the cycle counter state.

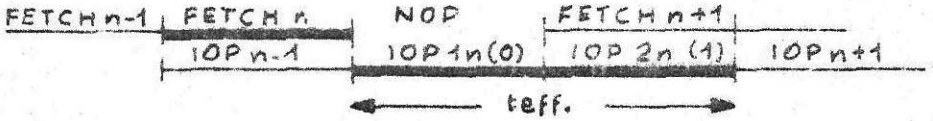
./..

TIMING



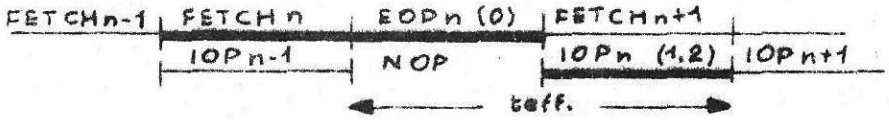
INSTRUCTIONS

LAS, LSS, LTS
LAR, SAR, ADR, ANR, EOR, DER
LAV, LAW, LAX, LAY.
SAV, SAW, SAX, SAY, SAT, SST
ALS, ARS, ALF, ARF



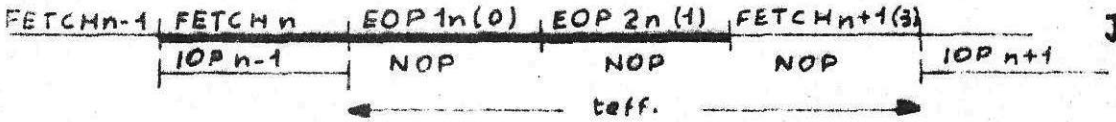
DAR

LAL, ANL, EOL, ORL, ADL, CML



INP

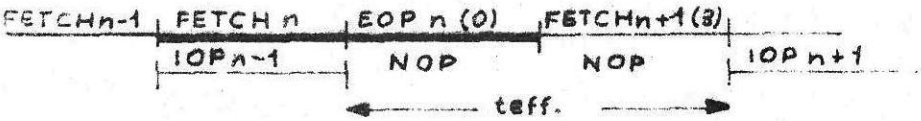
JUMP: JAZ, JAN, JAP, JCZ, JCN, JSD



JMP, JSB

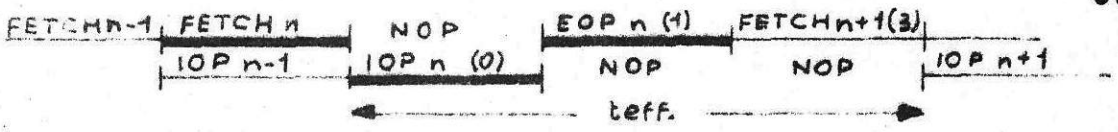
NO JUMP: JAZ, JAN, JAP, JCZ, JCN, JSD

RET



SIX, SZX, SZY, SQX, SQY

OUT



LIX, LIY

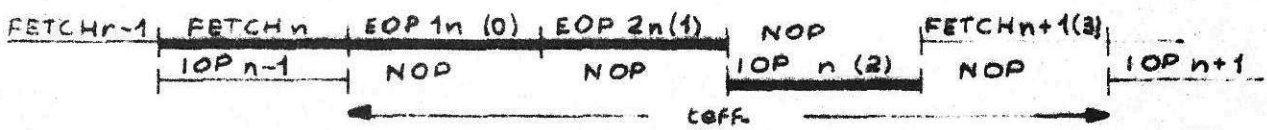


Fig. 15 - INSTRUCTION OVERLAPPING

CONTROL LINES

The control lines needed by the M 38 system are four.
Their simbolic names and the main usage are described here:

1 - DBDI: - Data bus direction.
A "Ø" logic on this line indicates a transfer from the Data Bus toward the CPU.
Vice-versa, the logic "1" indicates a transfer from the CPU to the Data Bus.

2 - ADSL: - Address/Data select.
A "Ø" logic defines the data bus content as a data.
The data bus content will then be stored into the enabled memory or I/O Port.

A "1" logic on this line defines the data bus content as an address.
On this case the data bus content will be stored into the address register (Q or Z) of the enabled module.

3 - WEQZ: - Write enable on Q or Z register.
This line defines the address register which has to receive the data bus content.
As it will be described later on by the block diagrams of the system devices, two are the address registers used by the program memory or the data memory devices: they are named Q and Z register.
A "Ø" logic on the WEQZ line enables the Q register to receive the address.
Vice-versa, a "1" logic on that line enables the Z register to receive the address.

./..

4 - PPOP: - Push/Pop operation.
On the program memory device, an hardware stack of four registers is provided.
They are the Q, RA, RB, Z registers, and their content may be pushed down to the immediately following register or popped up from one register to the preceeding one.
The push-pop operation is performed over the full stack of four registers and is controlled by the PPOP line.
A "Ø" logic on this line is push/pop disable, while a "1" logic is interpreted as enable for the push/pop operation.
The selection between push and pop is done by the DBDI line (1 logic for push, Ø logic for pop).

The following table is the summary for the possible control line configurations.
On the table it is also indicated for every line condition:

1 - The function type performed by the code.

2 - The external operations carried out by the system and caused by the code.
These external operations are associated with the proper phase.

The following symbols are used on the table:

I.R. = Instruction register -

Temp = Temporary register -

DAB = Data bus -

DEV = Device either ROM or RAM or I/O -

The brackets have the meaning: "The content of".

./..

PPOP	WEQZ	DBDI	ADSL	EXTERNAL OPERATION	PHASE	FUNCTION
Ø - NO PUSH/POP 1 - PUSH/POP	Ø - WRITE Q 1 - WRITE Z	Ø - DEV CPU 1 - CPU DEV	Ø - DATA 1 - ADDRESS			
Ø	Ø	Ø	Ø	1- (ROM)by Q DAB 2- DAB I.R, TEMP 3- INCR Q	F 4 F 1 F 1	READ PROGRAM
Ø	1	Ø	Ø	1- (DEV) by Z DAB 2- DAB TEMP	F 4 F 1	READ DATA FROM MEMORY (ROM OR RAM) OR I/O
Ø	Ø	Ø	1	1- (TEMP) DAB 2- NO OPERATION	F 4 F 1	NOP
Ø	Ø	1	Ø	1- CPU DAB 2- DAB DEVICE	F 4 F 1	WRITE DATA INTO MEMORY (RAM) OR I/O
Ø	Ø	1	1	1- CPU DAB 2- DAB Q	F 4 F 1	WRITE ADDRESS INTO Q
Ø	1	1	1	1- CPU DAB 2- DAB Z	F 4 F 1	WRITE ADDRESS INTO Z
1	Ø	1	1	1- (TEMP) DAB 2- DAB Q	F 4 F 1	PUSH OPERATION
1	Ø	Ø	1			POP OPERATION

Mod. 3050B241 - 5000 - Oct-Sipi S. p. A.

I/O STRUCTURE

Every I/O port of the M 38 system is composed of 8 logic blocks, whose logic circuit and interconnection with the CPU is shown on figure 16.

The following operations may be done:

1 - Transfer from the Data bus to the peripheral line.
When the transfer is performed, the voltage level is inverted and then stored into the output flip-flop during F1.
Of course the "OUT ENABLE" must be true.
The output to the peripheral line is done by means of an open drain MOS: this stage inverts again the voltage level.
Please note that the power supply used for the system is +5, Ø, -12 V for TTL, DTL compability, as it will be discussed on the next paragraph.
It is also worthy to note that while the voltage level at the point A reaches its value during F1, the transient at the point B, depends on the RC time constant connected to the peripheral line (see figure 17).
Another thing to remember is that a logic inversion is done when the accumulator is transferred on the data bus.
So the data value along the transfer path for an output instruction (ACC to I/O port) is as shown on the following table:

ACC	DAB		OUTPUT MOS	I/O LINE	
LOGIC	VOLT	LOGIC		VOLT	LOGIC
1	+ 5	Ø	ON	+ 5	Ø
Ø	GND	1	OFF	- 12	1

./..

2 - Transfer from the peripheral line to the data bus.
This transfer is done during F4 when the "INPUT ENABLE" is true.
Note that, in order to have the signal at the point B properly set by the peripheral line, the open drain MOS device must be off.
The data value along the transfer path for an input instruction (I/O to ACC), is shown on the following table:

I/O LINE		BUFFER	DAB		ACC
VOLT	LOGIC		VOLT	LOGIC	
+ 5	Ø	ON	+ 5	Ø	1
- 12	1	OFF	- 12	1	Ø

The timing for an INPUT instruction has been also described in figure 14.

3 - As it is shown by the shematics, the SYNC signal holds the output device off as long as it is to Ø logic.

./..

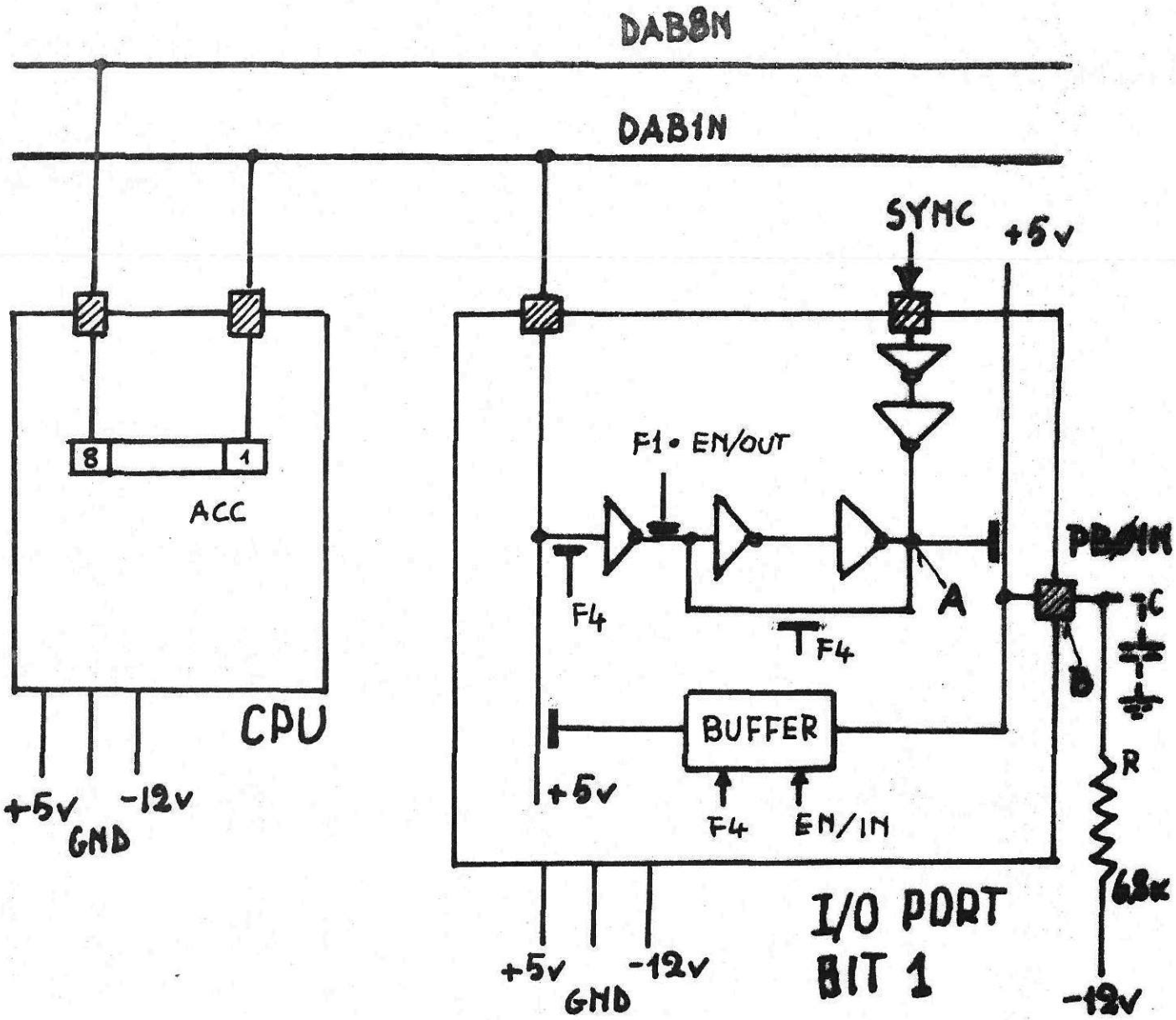


FIG 16 - I/O PORT : 1 BIT STRUCTURE

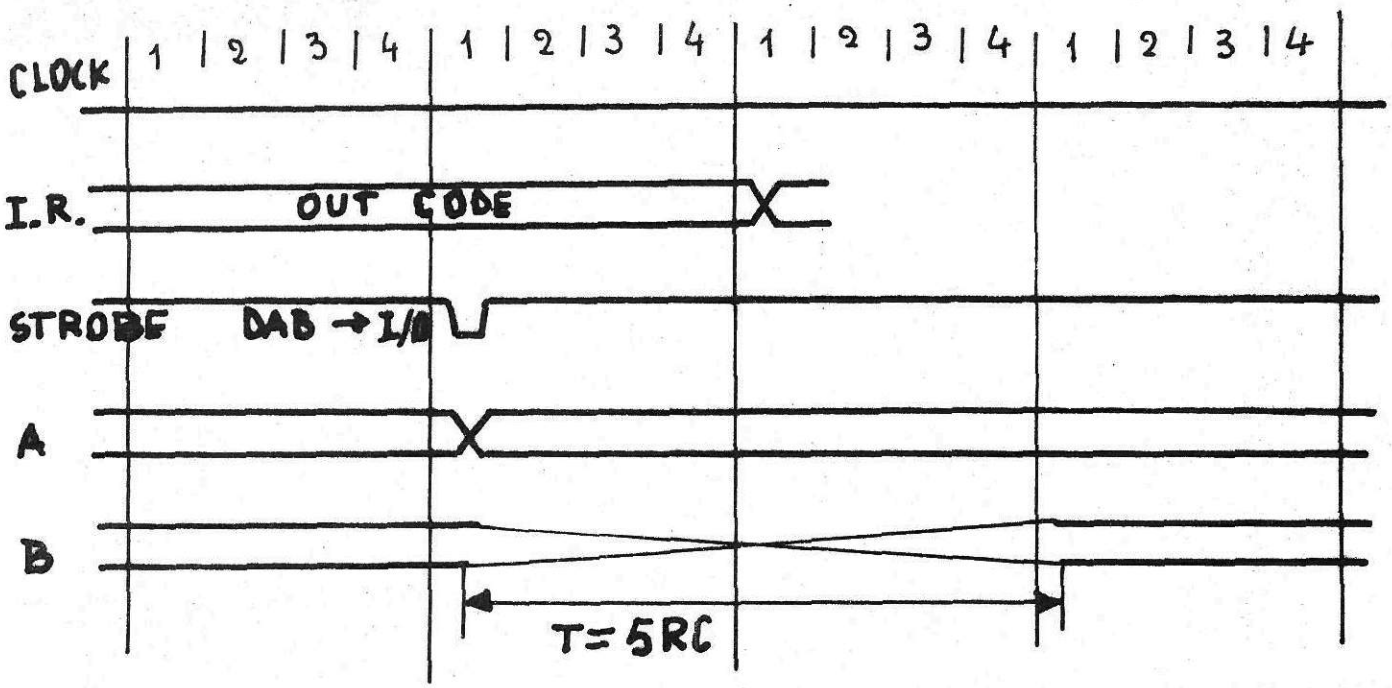


FIG 17 - DATA OUTPUT

I/O CONFIGURATION

The M 38 system may be connected to external devices only through the I/O Ports.
The electrical specifications for these points are indicated on figure 18, where the power supply is supposed to be Ø, - 17 V.

When the M 38 system has to be interfaced with TTL / DTL devices, the common configuration for the power supply will be: + 5V, ØV, - 12V.
This in order to have the Ø, + 5V available for the TTL, DTL logic.
The TTL / DTL and M 38 interconnections are shown on figure 19.
A pull-up resistor on the range of 4K is recommended to assure adequate positive driving levels when the TTL or DTL device is connected to the I/O input.

When the TTL / DTL device has to be connected at the output, a 6,8 K resistor to - 12V is required to garantee the compatibility between MOS and DTL, TTL on worst case conditions.

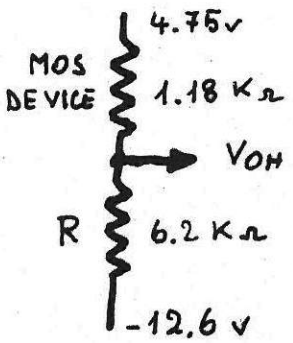
Let's analyse the two possible cases:

- The MOS device is off
- The MOS device is on

A - When the MOS device is off, there is a logic Ø at the DTL or TTL input.
On the worst case, the driving circuit must garantee 1,6 mA at the TTL input, with Ø,4 V as input voltage.
By assuming + 5% on - 12V and + 10% on R, the current will be:

$$I = \frac{11,6 + 0,4}{7,5} = 1,6 \text{ mA}$$

B - When the MOS device is on, there is a logic 1 at the DTL or TTL input.
On the worst case the MOS device must guarantee 1,9 V.
By assuming - 5% on 5V, + 5% on - 12V and - 10% on R, and assuming also the MOS R_{on} = 1,18 KΩ the equivalent circuit on this case is:



./..

The total current flowing on the two resistors is:

$$I = \frac{4,75 + 12,6}{1,18 + 6,2} = \frac{17,35}{7,38} = 2,35 \text{ mA}$$

The voltage drop across the MOS device is:

$$V_{mos} = 1,18 \times 2,35 = 2,77$$

and the $V_{oh} = V_{al} - V_{mos} = 4,75 - 2,77 = 1,98$ V.

Figures from 20 to 22 show the suggested output configuration to drive a magnet, a nixie tube or a LED lamp.
For magnet or nixie driver, a transistor has been used as interface, due to the high voltage swing required by these two output devices.
For LED devices the interface may be either a DTL / TTL or a transistor, or a COSMOS.

Figure 23 refers to a keyboard matrix 8x8 connected to two I/O PORTS:

- I/O 1 is used only as an output port -
- I/O 2 is used as an input port. -

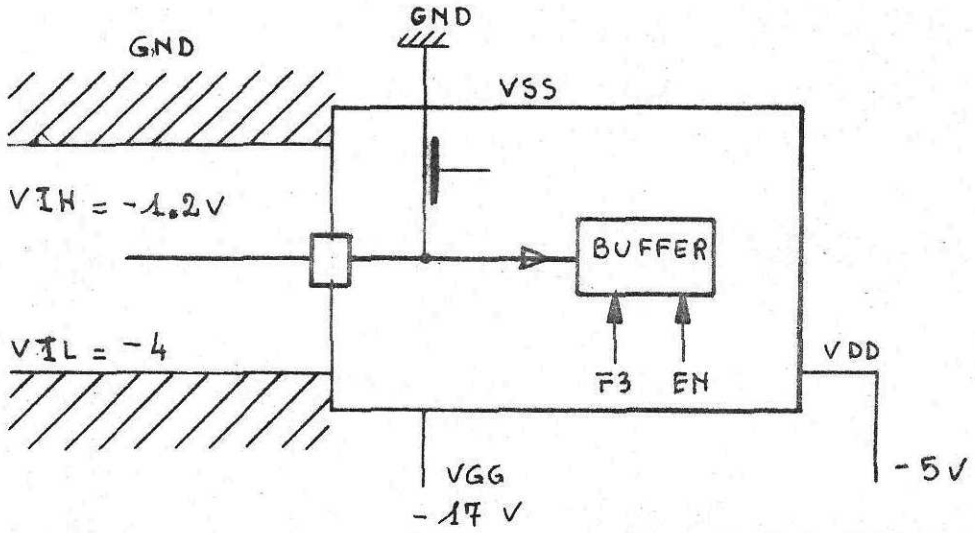
The keyboard scanning is performed by sending out from I/O 1 an enable on one row only (only one of the 8 bits is to 1 logic) and by reading the result from I/O 2.
On figure 23 B it has been represented the hardware for only one key.
For a wright operation, M 1 must be always off.
To test the key position M 2 is set ON.
The Vout will be:

A - Vout = - 12 V if the key is not pressed -

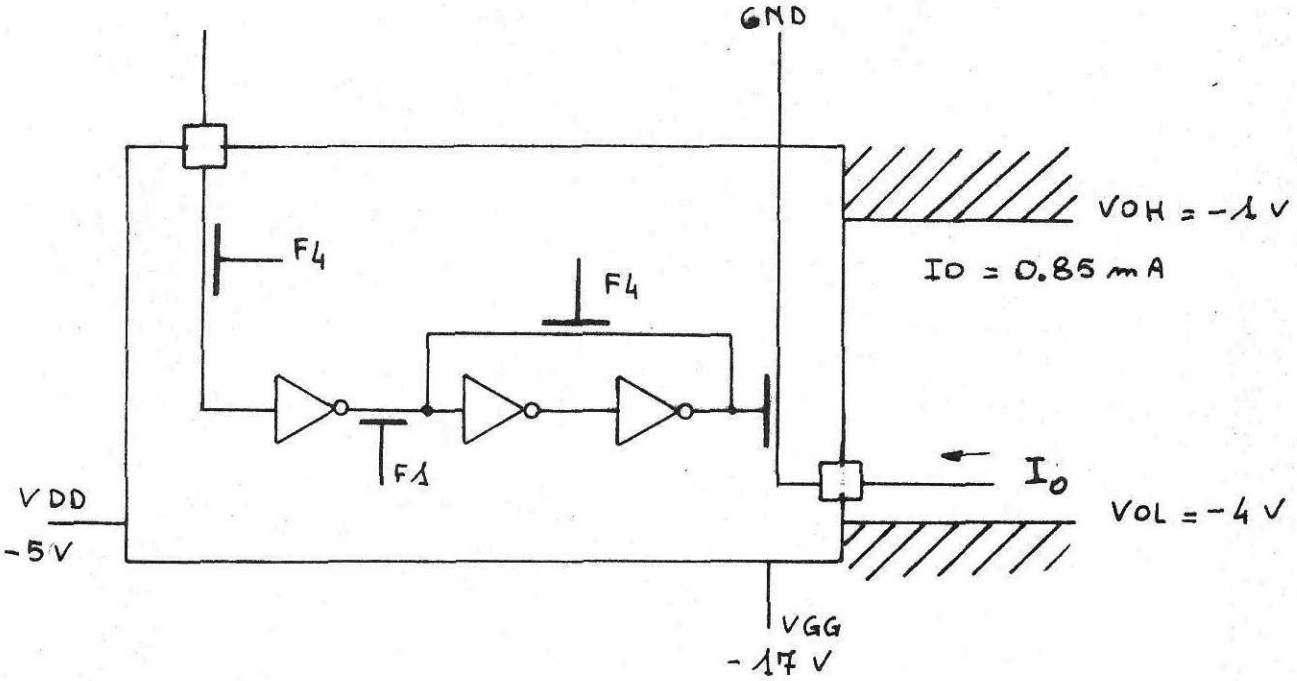
B - Vout = + 2,5 V is the key is pressed.

./..

M-38 SPECIFICATIONS



INPUT SPECIFICATIONS FIG 18A



OUTPUT SPECIFICATIONS FIG 18B

TTL/DTL — M-38

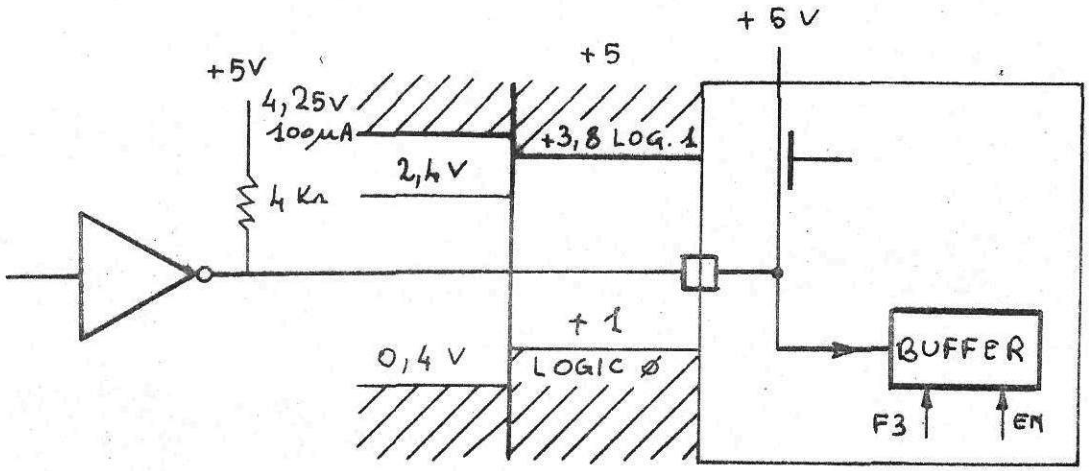


FIG 19 A

M38 — TTL/DTL

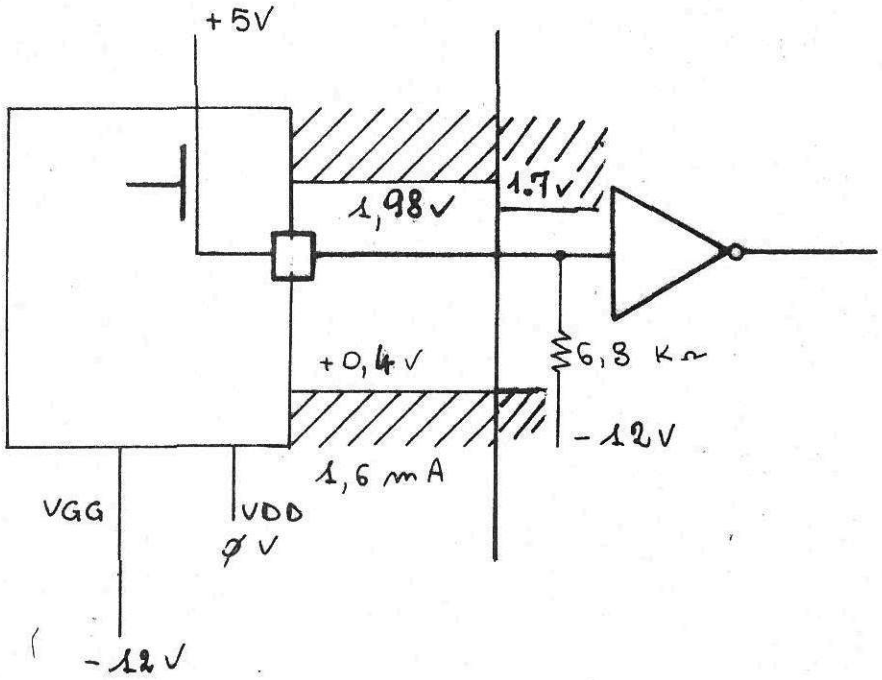


FIG 19 B

I/O CONFIGURATIONS

- 1 -

MAGNET DRIVER

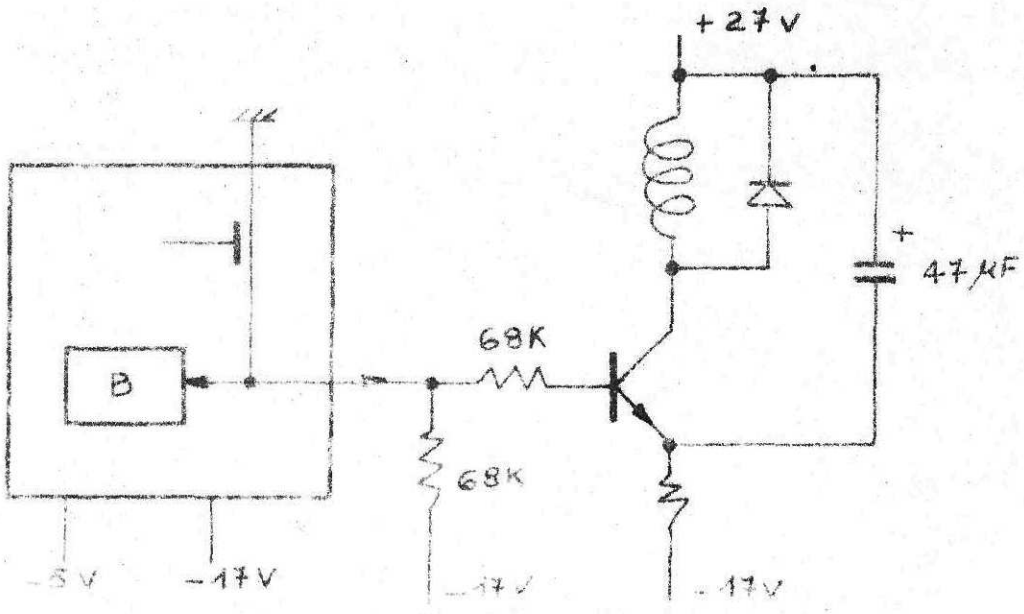


FIG 20

NIXIE DRIVER

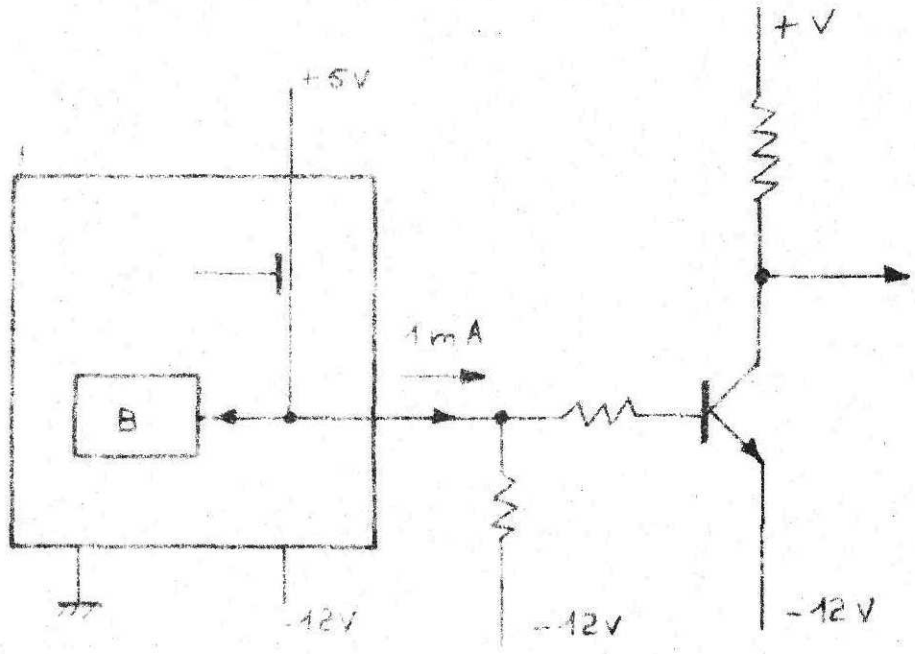


FIG 21

I/O CONFIGURATIONS

- 2 -

LED DRIVER

1_ DTL - TTL

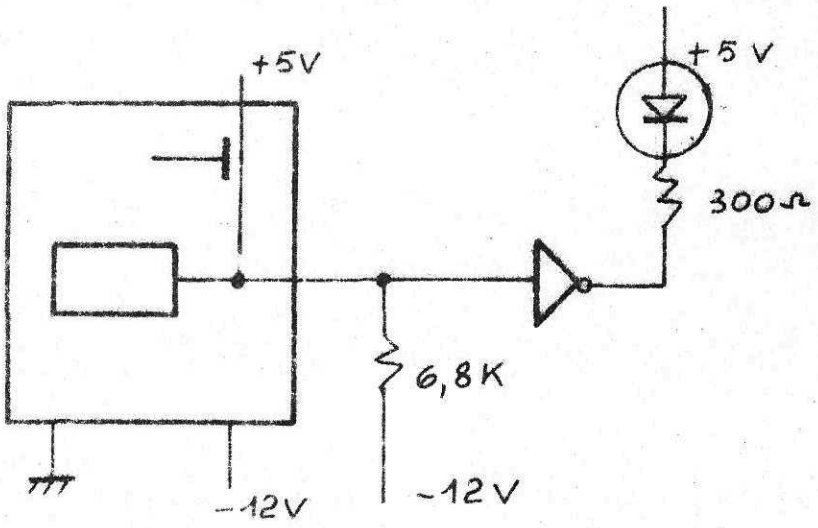


FIG 22 A

2_ TRANSISTORS

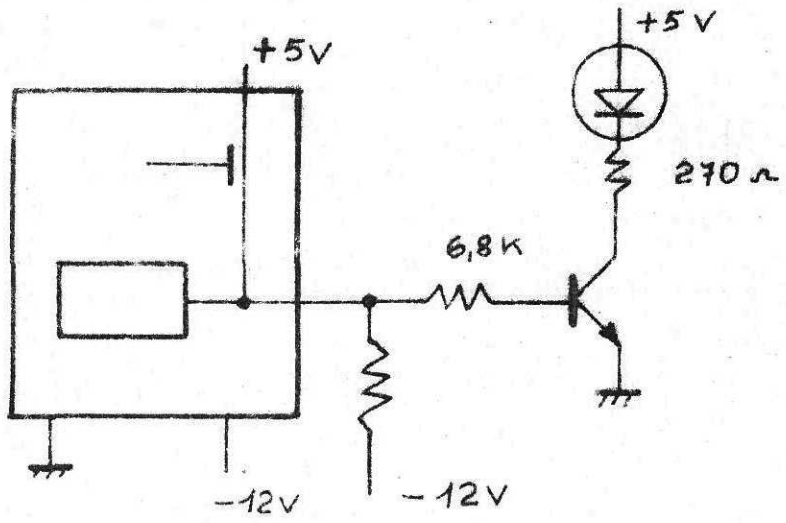


FIG 22 B

I/O CONFIGURATIONS

-3-

LED DRIVER

3 - COSMOS

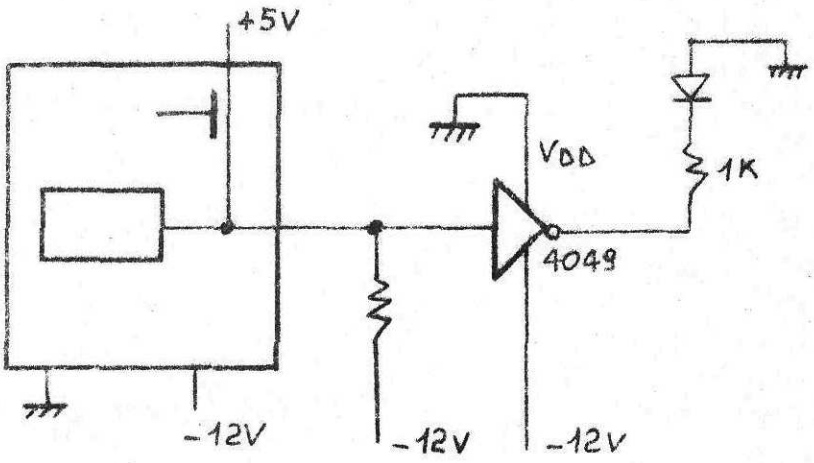
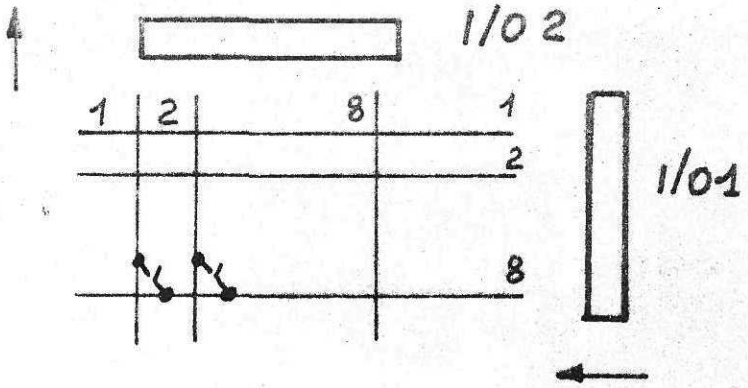


FIG 22 C

I/O PORT INTERCONNECTION

FIG 23A-

KEYBOARD MATRIX CONFIGURATION



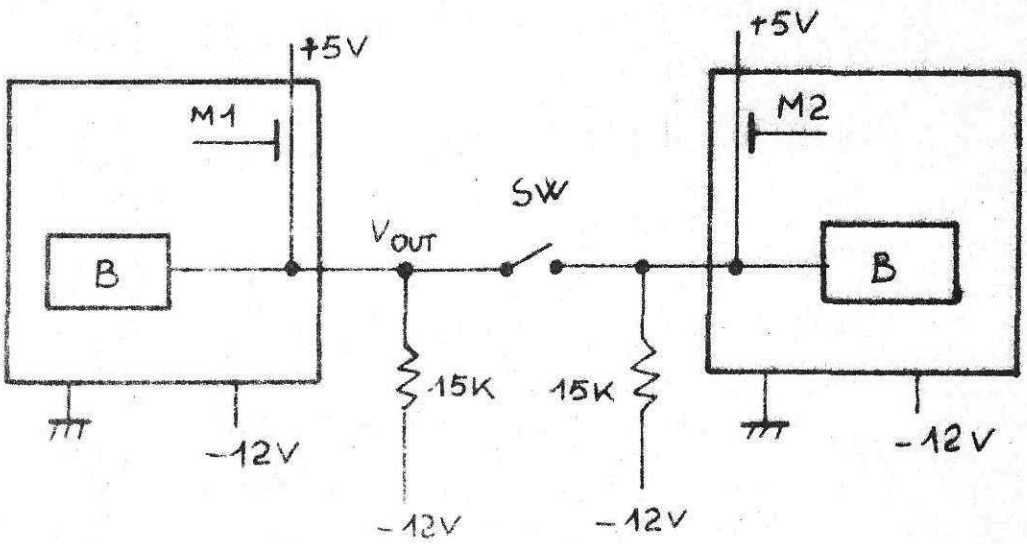


FIG 23B- ONE KEY INTERCONNECTION

STARTING OF A PROGRAM

As it has been described at the beginning by figure 3, a START switch is provided for ON/OFF operation of the system.
When the START switch is on the OFF position, the SYNC signal is at Ø logic, and this has the following consequences:

1 - The internal timing is locked -

2 - The PMC register which is the one on which the current program module code is stored, is held to Ø.
This means that the program memory enabled after the SYNC is moved to 1 is the ROM whose module code is Ø

3 - All the program counters are held to Ø.
This means that the program execution will begin from address Ø when the SYNC will be removed from the Ø state.

4 - All the I/O port flip-flops are set to Ø.
This means that the output open drain devices are off.

When the START is removed from the reset position, the SYNC signal will go to 1, and the system begins executing the program from location Ø of the module Ø.

./..

SYSTEM CAPACITY

The system capacity is limited by the number of module codes available.
The module code range is from Ø to 64 (decimal) and their assignement to the logic blocks has been already discussed extensively.

We may show here some examples in order to give to the user the feeling of what is possible to achieve by the M 38 system.

Example 1

1) Requirements for the Application:

10 K of program memory -
1 K by 8 of data memory -

2) The module codes needed to satisfy these requirements are:

10 K ROM : Ø to 39 -
8 RAMS (128x8) : 4Ø to 47 -

3) The available module codes are 24 from 48 to 63 which may be assigned to I/O ports, or may be used for later extension of the system.

Example 2

1) Requirements for the Application:

12 K of program memory -
10 I/O Ports -

2) The module codes needed for this applicazion are:

12 K ROM : Ø to 47
10 I/O Ports: 54 to 63

3) The available module codes are 6 from 48 to 53. They may be assigned, if needed, to 6 RAMS (128x8), or may be used for later extension of the system.

./..

BLOCK DIAGRAMS

The following paragraphs will describe the block diagram of the individual devices of the family.

At the end, the connection diagram and the pin definition for every device is given.

./..

BLOCK DIAGRAM - M 380 - (Fig. 24)

The M 380 is an 8 bit parallel-control processing unit built on a single MOS chip using P channel Silicon gate processing. It includes:

- An Accumulator
- A 48 byte RAM
- An Arithmetic and Logic Unit
- A Control Unit
- One 8 bit I/O Port (8in, 4out)

The Memory has a size of 48 x 8 bit words.
Every word can be addressed indirectly through the Address Registers S and T.
The T Register is used as a Page address, a Page is defined as a block of 8 consecutive words starting from Ø (Ø to 7, 8 to 16,...).
The S Register is used for Word address within a Page.
The S Register is also an Index Register which can be used to automatically scan a page.
The words Ø to 15 can be addressed directly by the instruction register.
Two words, X and Y (the last two on the second page), are used for to access the ROM, RAM or I/O ports directly.
The Arithmetic and Logic Unit performs the following basic operations:

- Add in binary or BCD notation
- Increment or decrement
- And, Or, Exclusive-or logic
- One bit or four bit shift
- Load and Store

./..

The operations are performed between the Accumulator and any one of the 48 Memory Registers, or a temporary Register used for I/O transfer or immediate operations.
The result is stored in the Accumulator or in another register in the microprocessor.
The control unit consists of the:

- Instruction Register
- The Control ROM
- The Cycle Counter
- The Program Module Code Register PMC
- Control Flip-Flops

The control ROM decodes the Instruction Register and generates the control signals for the Arithmetic Logic Unit and for the addressed module.

More complex instructions are performed step by step under the control of a cycle counter (the max number of cycles is 4).
The module address, generated by the CPU and coming either from the PMC register, the X or Y registers, or the instruction register, is sent to the address bus by means of a multiplexer which receives commands from the control logic.

The four Control Flip-Flops which detect:

(i) carry, on the 8th bit
(ii) accumulator contents equal zero
(iii) accumulator positive
(iiii) register S contents equal to 7.

are set by the current instruction execution and may be used as a condition on a program branch.
The I/O Channel of the M 380 includes a 4 bit output buffer where the information from the most significant bits of the Accumulator can be stored: they will stay there until the next output operation.
The data input (8 bit) is performed through the Temporary Register and the information is stored into the Accumulator.

./..

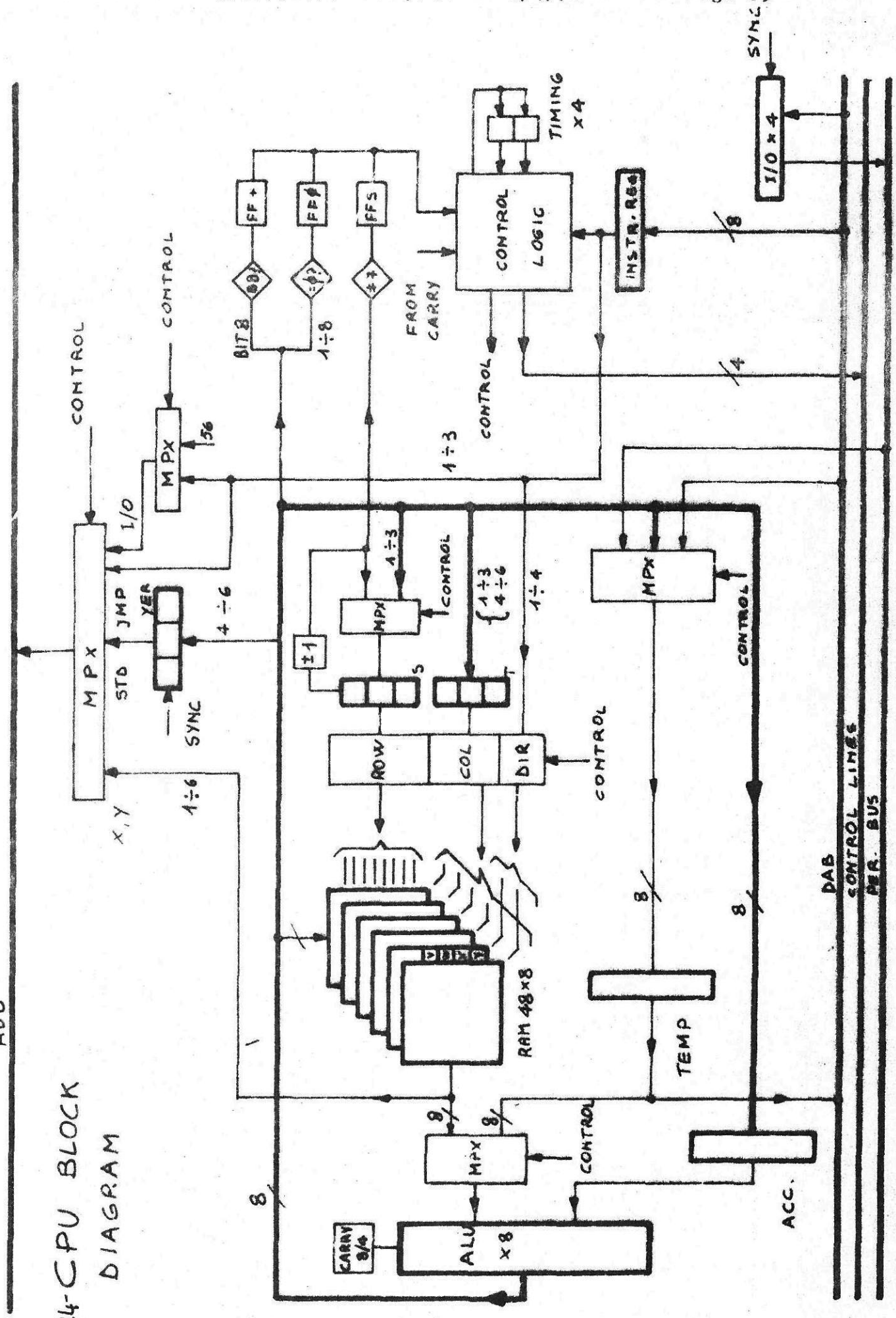


FIG 24 - CPU BLOCK DIAGRAM

BLOCK DIAGRAM - M 382 - (Fig. 25)

The M 382 is a 8192 MOS Read Only Memory organized as 1024 words by 8 bits.
The ROM is designed for the M 38 microcomputer system and can be directly connected to the M 38 system buses without any additional logic.
The unit includes:

- 1 K byte ROM.
- The 11 bit registers Q, RA, RB, Z. These are organized as a Stack and the information stored in them can be moved up and down by control commands.
- The ROM address is stored in the Q register and the content can be incremented on command.
  In some cases the Z register can be used to address the ROM: the selection of Q or Z is done by the control logic and is a function of the control line status.
  The Q and Z register content can be changed be storing ADB bit 3/2/1 into Q bits 11/10/9 or Z bits 11/10/9 and DAB into Q or Z (bits 8 to 1): this is done by commands decoded from the control lines.
- 2 I/O ports of 8 bits.
  Through each I/O port it is possible to transfer information onto the data bus from 8 independent lines connected to peripherals or to store the data bus information in 8 flip-flops, connected to the same 8 lines.
- The module codes assigned to the unit during the system design and the logic to compare this code with the information on the address bus.
  The module codes are built onto the chip by the custom mask/interconnect process.

./..

Possible operations controlled by the logic included on the chip are:

- Read the memory content addressed by Q or Z.
- Increment Q.
- Push/pop operation on the register stack (Q,RA,RB, Z).
- Store ADB and DAB on Q or Z.
- Read the information from one of two sets of eight independent lines onto the data bus via I/O ports.
- Store the data bus information in one of two I/O ports.

These operations are defined at a predetermined time by the control line status and become effective only in the M 382 whose masked module code is coincident with the code appearing at the proper time on the address bus (ADB).

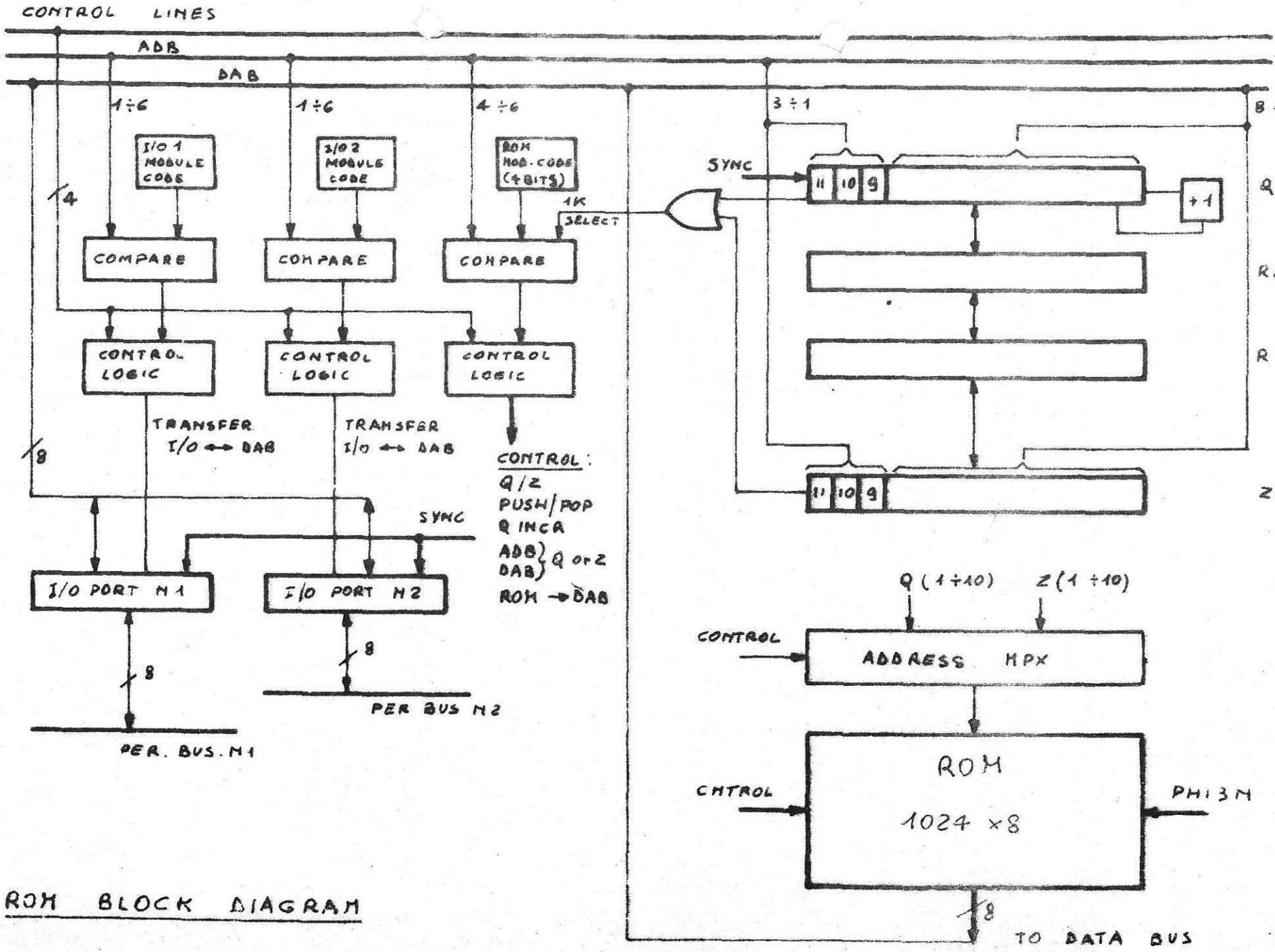


FIG 25 - ROM BLOCK DIAGRAM

BLOCK DIAGRAM - M 381 - (Fig. 26)

The M 381 is a ROM/RAM including:

- 768 words of 8 bits of Read Only Memory.
- 18 words of 8 bits of Random Access Memory.
- The 11 bit registers Q, RA, RB, Z. These are organized as a Stack and the information stored in them can be moved up and down by control commands.

  In some cases the Z register is used to address the ROM: the selection of Q or Z is done by the control logic and is a function of the control lines status.
  The Q and Z register contents can be changed by storing ADB bits 3/2/1 into Q bits 11/10/9 or Z (bits 11/10/9) and DAB into Q or Z (bit 8 to 1): this is done by commands decoded from the control lines.
- Two I/O ports of 8 bits.
  Through each I/O port it is possible to transfer information into the data bus from 8 lines of a peripheral or to store the data bus information into 8 flip-flops whose output is connected to the same 8 lines.
- The masked module code assigned to the unit during system design and the logic to compare this code with the informations on the address bus.

Possible operations controlled by the logic included on the chip are:

- Read the memory content addressed by Q or Z.
- Write the data bus into the RAM addressed by Z.
- Push/pop operations on the register stack (Q,RA,RB,Z).
- Store ADB and DAB on Q or Z.
- Read the information from one of two sets of 8 independent lines onto the data bus.
- Store the data bus information into one of two I/O ports.

These operations are defined at a predermined time by the control line status and become effective only in the M 381 whose masked module code is coincident with the code appearing at the same time on the address bus (ADB).
The memory address of the ROM and RAM are not the same and different commands are used for the two memories.

./..

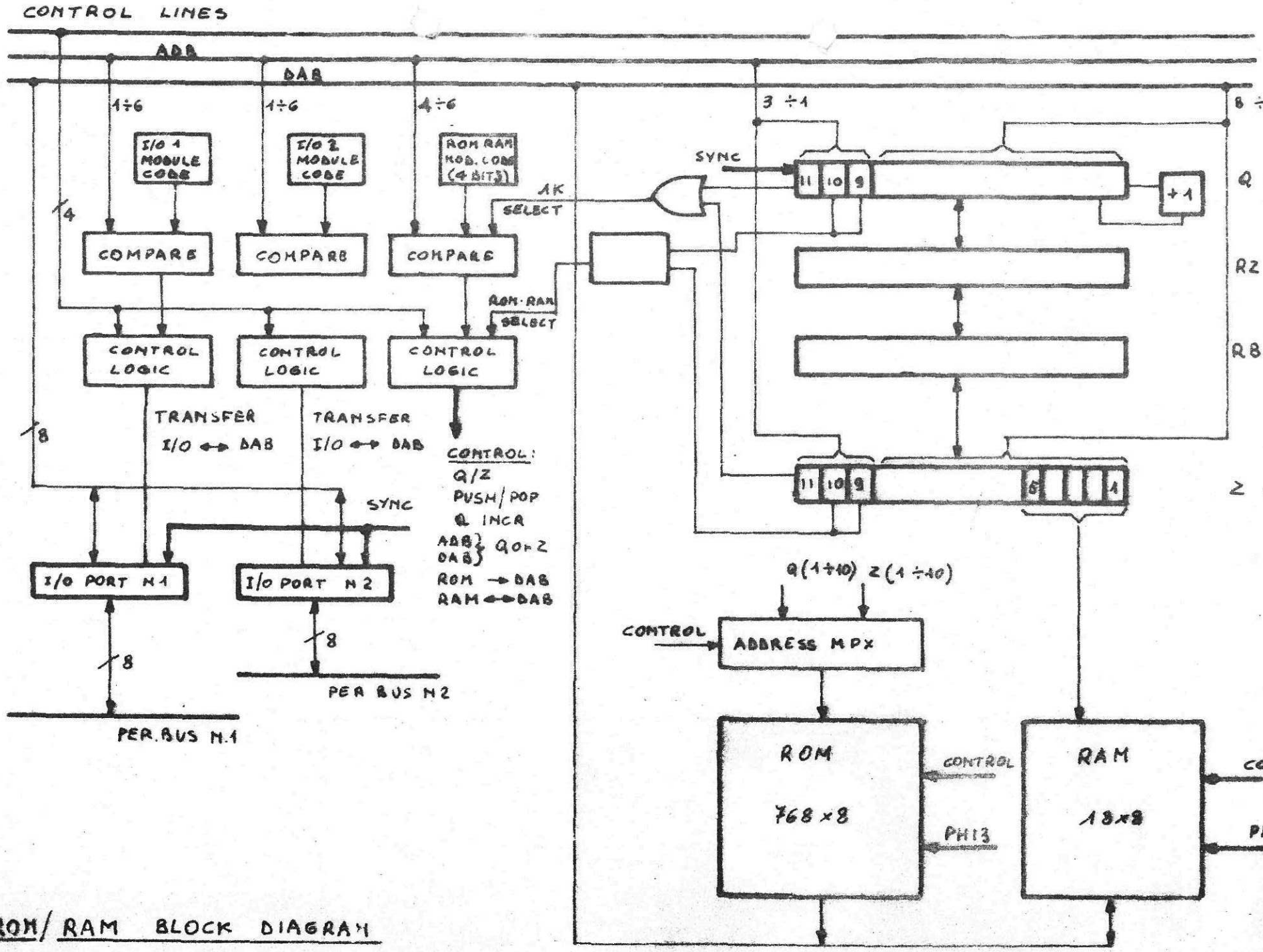


FIG 26-ROM/RAM BLOCK DIAGRAM

BLOCK DIAGRAM - M 383 - (Fig. 27)

The M 383 is an 128 words by 8 bits MOS Random Access Memory. This RAM is specifically designed for the microcomputer system M 38 and it can be directly connected on the M 38 buses without any additional logic.
The unit includes:

- 128 words of 8 bit RAM.
- The logic to compare the module codes selected on the strap lines (assigned to the unit during system design) and the information on the address bus.
- A 7 bit register (the Z register) on which the RAM address is stored.
  The Z register content can be modified by writing into it from the data bus.
- One I/O port of 8 bits.
  It is possible to transfer information from 8 independent lines connected to a peripheral onto the data bus or to store the data bus in 8 flip-flops connected to the same above lines.

Possible operations controlled by the logic included on the chips are:

- Read or write at the memory location addressed by Z.
- Store data bus in Z register.
- Read information from one set of 8 external lines onto data bus.
- Store the data bus information into one 8 bit I/O port.

These operations are defined at a predetermined time by the control lines status and become effective into the RAM unit whose module code cabled on the strap lines is coincident with the code present at the same time on the address bus (ADB).
The memory output is enabled to store information on data bus during F3 (STATX pin to Ø).
However, it is possible, by using the STATX pin connected to -14V, to enable a static transfer on data bus RAM output: this is necessary when the RAM is connected to a static data bus.

./..

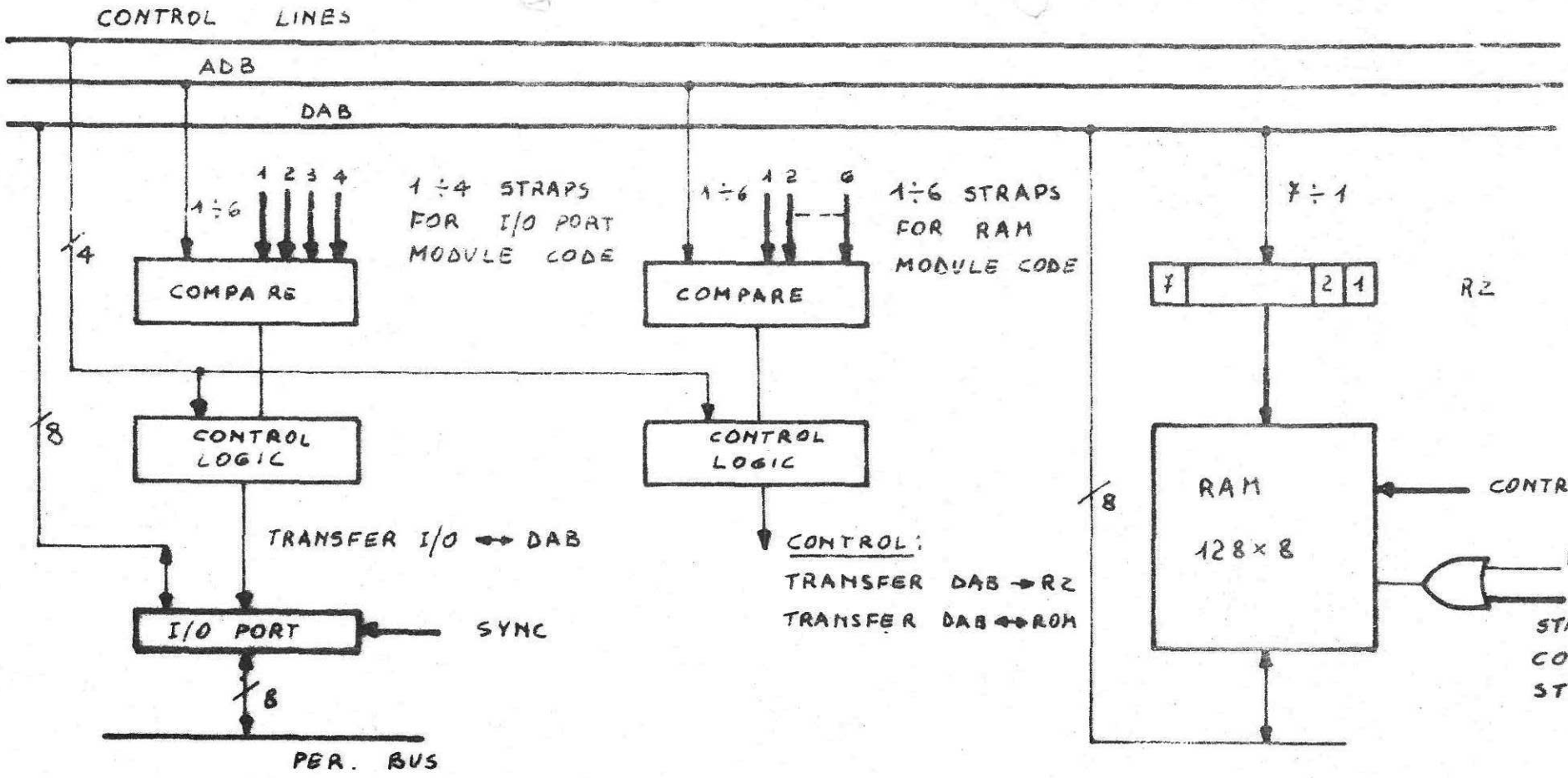
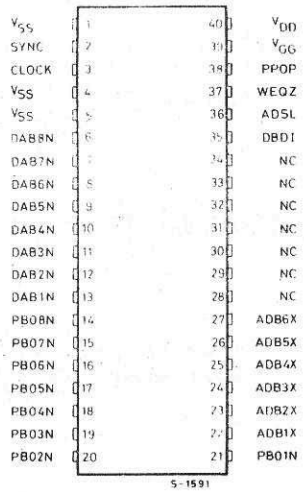


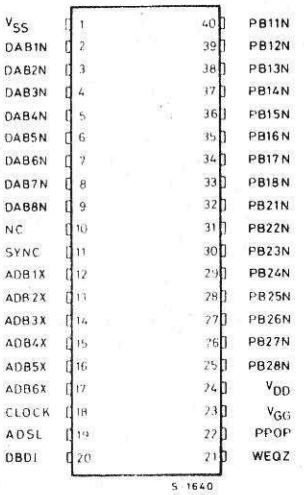
Fig 27 - RAM BLOCK DIAGRAM

CONNECTION DIAGRAM and PIN DEFINITION - 8 BIT MICROPROCESSOR - M 380



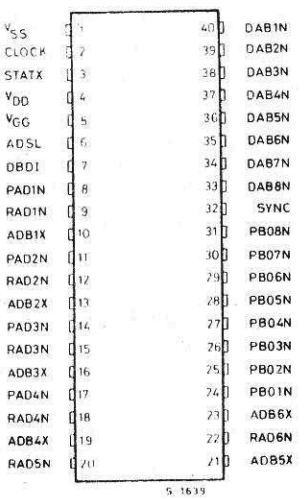
SYNC		Initial reset/synchronization (input)
CLOCK		System clock (input)
DAB 8N . . . . . DAB 1N		8 bit bidirectional data-bus
PBO 8N . . . . . PBO 5N		4 bit input-output peripheral-bus
PBO 4N . . . . . PBO 1N		4 bit input peripheral bus
ADB 6X . . . . . ADB 1X		6 bit address bus (outputs)
DBDI	Data bus direction	4 control lines (outputs)
ADSL	Address/data select	
WEQZ	Write enable a or Z	
PPOP	Push/pop operation	
X		Data true if logic level is 1
N		Data true if logic level is 0

CONNECTION DIAGRAM and PIN DEFINITION - 1K BYTE ROM - M 382
768 BYTE ROM/18 BYTE RAM - M 381



DAB 8N . . . . . DAB 1N		8 bit bidirectional Data Bus
SYNC.		Initial reset/synchronization (input)
ADB 1X . . . . . ADB 6X		6 bit address bus (inputs)
CLOCK		System clock (input)
ABDI	Data bus direction	4 control lines (outputs)
ADSL	Address/data select	
WEQZ	Write enable a or Z	
PPOP	Push/pop operation	
PB 28N . . . . . PB 21N		8 bit bidirectional peripheral bus
PB 18N . . . . . PB 11N		8 bit bidirectional peripheral bus
X		Data true if logic level is 1
N		Data true if logic level is 0

CONNECTION DIAGRAM and PIN DEFINITION - 128 BYTE RAM - M 383



CLOCK		System clock (input)
STATX		Used to select static/dynamic operation of the device
DBDI	Data bus direction	2 control lines (inputs)
ADSL	Address/data select	
ADB 1X . . . . . ADB 6X		6 bit address bus (inputs)
RAD 1N . . . . . RAD 6N		6 strap lines for the module code of the RAM (inputs)
PAD 1N . . . . . PAD 4N		4 strap lines for the 4 LSB of the module code of the peripheral channel (inputs) The 2 MSB are set at logic level 1 inside the chip.
PBO 1N . . . . . PBO 8N		8 bit bidirectional peripheral bus
SYNC		Initial reset/synchronization (input)
DAB 8N . . . . . DAB 1N		8 bit bidirectional data bus
X		Data true if logic level is 1
N		Data true if logic level is 0

INSTRUCTION SET

Altogether 48 instructions are available to the user program. These can be grouped as follows:

a) Instructions referring to data included in the instruction itself (immediate reference instructions).
The data may be of 3,4 or 8 bits.
These instructions carry out operations between the accumulator and the data, sending the result to the accumulator.
The possible operations involve loading, arithmetic, logic (AND, OR, EX-OR) and comparison.

b) Jump instructions.
Unconditional and conditional jump instructions (on the carry, on the conditions ACC = Ø and ACC ≠ Ø, positive and negative accumulator content, register S ≠ 7) and call to subroutine instructions (maximum nesting level = 3) are available.

c) Register reference instructions.
Logic and arithmetic instructions between accumulator and registers (Ø to 47) can be performed.
The registers Ø to 11 can be addressed directly.
All the registers can be addressed indirectly through S and T registers.
Operations to be performed between the addressed register and the accumulator are:

- Logical
- Arithmetic
- Load and Store

Registers 12 to 15 and S, T are special registers: they can be addressed directly only for load and store operations.

./..

d) Shift instructions on the accumulator (possible shifts: 1 or 4 bits, towards the right or left).

e) Input/Output instructions for transferring information from external lines to the accumulator or vice-versa.

f) Instructions for indirectly reading from or writing into the RAMs and the I/O ports and for writing into the Q and Z registers.

The system's basic cycle is 5 usec.
For most of the instructions mentioned above only one machine cycle is required.
To carry out the most complex instructions 2, 3 or 4 cycles may be necessary.

The instruction set is summarized on the table N. 2, where the mnemonic code, the description of the instruction itself and the binary code are indicated.
Every group of instruction is then described in detail on the paragraphs following the Table N. 2.

For every instruction group the common characteristics are described (format, number of machine cycles, internal operations on the CPU, signals available on the buses) and then every instruction is discussed by defining the operations performed and by giving some typical examples.
A list of the symbols which are used throughout these paragraphs is given at the beginning.

./..

TABLE 2

TABLE 2

INSTRUCTION SET

	Mnemo	Description		Binary code	Cycle number	Bytes	Note n°
SHORT IMMEDIATE	LAS I	Load accumulator with I (decimal numbers 0 to 15)	I (binary form) → ACC (1 to 4) 0 → ACC (5 to 8)	1111 XXXX	1	1	
	LSS I	Load S register with I (decimal numbers 0 to 7)	I (binary form) → S	00101 XXX	1	1	
	LTS I	Load T register with I (decimal numbers 0 to 7)	I (binary form) → T	00111 XXX	1	1	
LONG IMMEDIATE	LAL I	Load accumulator with I (decimal numbers 0 to 255)	I (binary form) → ACC (1 to 8)	0000 0100 XXXX XXXX	2	2	
	ANL I	AND accumulator with I (decimal numbers 0 to 255)	I (binary form) ∧ ACC (1 to 8) → ACC	0000 0101 XXXX XXXX	2	2	
	EOL I	XOR accumulator with I (decimal numbers 0 to 255)	I (binary form) ⊕ ACC (1 to 8) → ACC	0000 1100 XXXX XXXX	2	2	
	ORL I	OR accumulator with I (decimal numbers 0 to 255)	I (binary form) ∨ ACC (1 to 8) → ACC	0000 1101 XXXX XXXX	2	2	
	ADL I	ADD accumulator with I CARRY is lost	I (binary form) + C (1 to 8) → ACC	0000 1110 XXXX XXXX	2	2	
	CML I	Compare accumulator with 2' complement of I	ACC < (Ī + 1), carry F F = 0 ACC ≥ (Ī + 1), carry F F = 1 ACC ≠ (Ī + 1), zero F F = 0 ACC = (Ī + 1), zero F F = 1	0000 1111 XXXX XXXX	2	2	
JUMP	JMP LABEL	Unconditional jump to Address specified by LABEL	LABEL (1 to 11) → PC	01000 XXX XXXX XXXX	3	2	1
	JAZ LABEL	Jump to Address specified by LABEL if ACC = 0	if ZERO F F = 1 LABEL (1 to 11) → PC	01001 XXX XXXX XXXX	2,3	2	1
	JAN LABEL	Jump to Address specified by LABEL if ACC ≠ 0	if ZERO F F = 0 LABEL (1 to 11) → PC	01010 XXX XXXX XXXX	2,3	2	1
	JAP LABEL	Jump to Address specified by LABEL if ACC is positive	if ACC (bit 8) = 0 LABEL (1 to 11) → PC	01011 XXX XXXX XXXX	2,3	2	1
	JSD LABEL	Jump to Address specified by LABEL if Register S ≠ 7	if S ≠ 7 LABEL (1 to 11) → PC	01100 XXX XXXX XXXX	2,3	2	1
	JCN LABEL	Jump to Address specified by LABEL if CARRY F F = 1	if CARRY = 1 LABEL (1 to 11) → PC	01101 XXX XXXX XXXX	2,3	2	1
	JCZ LABEL	Jump to Address specified by LABEL if CARRY F F = 0	if CARRY = 0 LABEL (1 to 11) → PC	01110 XXX XXXX XXXX	2,3	2	1
	JSB LABEL	Jump to subroutine	Push operation PC + 1 → PC PC → RA → RB → RZ LABEL (1 to 11) → PC	01111 XXX XXXX XXXX	3	2	1
	RET	Return from subroutine	Pop operation: RZ → RB → RA → PC	0000 0000	2	1	1
I/O	INP N	Load input from I/O Addressed by N into ACC	Input N → ACC	00100 XXX	2	1	5
	OUT N	Store ACC into I/O port Addressed by N	ACC → Output N	00110 XXX	3	1	5
REGISTER REFERENCE	LAR R	Load ACC with the content of Register R (decimal number)	REG (1 to 8) → ACC	1000 XXXX	1	1	2
	SAR R	Store ACC in Register R (decimal number)	ACC (1 to 8) → REG	1001 XXXX	1	1	2
	ADR R	Add ACC and Register R, CARRY F F is not affected	ACC (1 to 8) + REG (1 to 8) → ACC	1010 XXXX	1	1	2
	ANR R	AND ACC with Register R	ACC (1 to 8) ∧ REG (1 to 8) → ACC	1011 XXXX	1	1	2
	EOR R	XOR ACC with Register R	ACC (1 to 8) ⊕ REG (1 to 8) → ACC	1100 XXXX	1	1	2
	DER R	Decrement Register R	REG (1 to 8) -1 → REG	1101 XXXX	1	1	2
	DAR R	Decimal addition of ACC, Register R and CARRY FF. CARRY is affected by overflow from most significant BCD byte	ACC (1 to 8) + REG (1 to 8) + CARRY → REG BCD overflow → CARRY	1110 XXXX	2	1	2-3
SPECIAL REGISTER	LAV	Load V Register into ACC	V (1 to 8) → ACC	0000 1000	1	1	4
	LAW	Load W Register into ACC	W (1 to 8) → ACC	0000 1001	1	1	4
	LAX	Load X Register into ACC	X (1 to 8) → ACC	0000 1010	1	1	4
	LAY	Load Y Register into ACC	Y (1 to 8) → ACC	0000 1011	1	1	4
	SAV	Store ACC into V Register	ACC (1 to 8) → V	0001 1000	1	1	4
	SAW	Store ACC into W Register	ACC (1 to 8) → W	0001 1001	1	1	4
	SAX	Store ACC into X Register	ACC (1 to 8) → X	0001 1010	1	1	4
	SAY	Store ACC into Y Register Store ACC (4, 5, 6) into PMC Register	ACC (1 to 8) → Y ACC (4 to 6) → PMC	0001 1011	1	1	4
	SAT	Store ACC (1, 2, 3) into T Register	ACC (1 to 3) → T	0000 0001	1	1	
	SST	Store ACC (1, 2, 3) into S Register Store ACC (4, 5, 6) into T Register	ACC (1 to 3) → S ACC (4 to 6) → T	0000 0011	1	1	

TABLE 2

INSTRUCTION SET

	Mnemo	Description		Binary code	Cycle number	Bytes	Notes n°
SHIFT	ALS	ACC left shift 1 bit CARRY FF set to 1	ACC (1 to 7) → ACC (2 to 8) 0 → ACC (1) 1 → CARRY	0001 1100	1	1	
	ARS	ACC right shift 1 bit CARRY FF set to 0	ACC (2 to 8) → ACC (1 to 7) 0 → ACC (8) 0 → CARRY	0001 1101	1	1	
	ALF	ACC left shift 4 bit	ACC (1 to 4) → ACC (5 to 8) 0 → ACC (1 to 4)	0001 1110	1	1	
	ARF	ACC right shift 4 bit	ACC (5 to 8) → ACC (1 to 4) 0 → ACC (5 to 8)	0001 1111	1	1	
MODULE REFERENCE	SIX	Store ACC into RAM or I-O module Addressed by content of CPU X (1 to 6) Register The RAM location is identified by the content of Z Register	X (1 to 6) → ADB If module (X) is I/O: ACC → I/O (X) If module (X) is RAM ACC → RAM WORD (Z)	0000 0010	3	1	
	LIX	Load ACC with content of RAM, I/O or ROM module Addressed by content of CPU X (1 to 6) Register. The RAM/ROM location is identified by the content of Z Register.	X (1 to 6) → ADB If module (X) is I/O: I/O (X) → ACC If module is memory: memory word (Z) → ACC	0000 0110	4	1	
	LIY	Same as LIX but the module Address is set by Y Register	Y (1 to 6) → ADB If module (Y) is I/O I/O (Y) → ACC If module (Y) is memory: memory word (Z) → ACC	0000 0111	4	1	
	SQX	Store ACC into Q Register of the ROM module addressed by content of CPU X (1 to 6) Register; ACC is stored in Q (1 to 8) Register; X Register (1 to 3) is stored in Q (9 to 11) Register	X (1 to 6) → ADB ACC (1 to 8) → Q (1 to 8) X (1 to 3) → Q (9 to 11)	0001 0110	3	1	
	SQY	Same as SQX but the module Address is set by Y Register	Y (1 to 6) → ADB ACC (1 to 8) → Q (1 to 8) X (1 to 3) → Q (9 to 11)	0001 0111	3	1	
	SZX	Store ACC into Z Register of RAM/ROM module Addressed by content of CPU X (1 to 6) Register If the Addressed module is ROM the ACC (1 to 8) is stored into Z (1 to 8) and X (1 to 3) Register is stored in Z (9 to 11) Register If the Addressed module is RAM the ACC (1 to 7) is stored in Z (1 to 7) Register	X (1 to 6) → ADB If module is ROM: ACC (1 to 8) → Z (1 to 8) X (1 to 3) → Z (9 to 11) If module is RAM ACC (1 to 7) → Z (1 to 7)	0001 0010	3	1	
	SZY	Same as SZX but the module Address is set by Y Register	Y (1 to 6) → ADB If module is ROM ACC (1 to 8) → Z (1 to 8) Y (1 to 3) → Z (9 to 11) If module is RAM: ACC (1 to 7) → Z (1 to 7)	0001 0011	3	1	

NOTES: 1 - LABEL = Jump Address in the range 0 to 2047. The direct addressing field is 2 K ROM (equivalent to 8 ROM modules or 2 ROM chips) enabled by Address Bus bits 6, 5, 4.

2 - Register Address R In the range 0 to 11 is a direct Address
12 is Addressing indirectly through S T
13 is Addressing indirectly through S T with indexing (S → T → S)
14 is Addressing indirectly through S T with indexing (S + 1 → S)
15 code not allowed

3 - The DAR instruction is implemented in 2 CPU cycles:
a) 8 bit binary addition
b) Decimal correction, adding 10 to the first 4 bits if there is carry coming from the fourth bit and to the second 4 bits, if there is carry coming from the eigth bit generated during the first cycle. At the end of the instruction the eigth bit carry is stored in the carry flip-flop.

4 - The registers V, W, X, Y are in the CPU RAM, they are located in the positions 12, 13, 14, 15 respectively. The resisters X, Y are also used to implement the instructions: SQX, SQY, SZX, SZY, SIX, LIX, LIY.

5 - N address must be specified in the range 0 to 7. Actual I/O port module address is computed by CPU adding 56 (decimal) to the specified address.

SYMBOL LIST

The following are the symbols used throughout the instruction set description.

(R)	- Means 'the content of R'. - The symbol between brackets refers always to a register.
((R))	- Means 'the content of the register or memory location whose address is on R'.
(R) n ÷ m	- Means 'bits from n to m of R register content'.
A	- Accumulator -
I.R.	- Instruction register.
Temp	- Temporary register.
Q	- Q register:the program counter.
Z	- Z register:the data address register.
S	- S register:CPU-RAM row address.
T	- T register: CPU-RAM page address.
V	- Auxiliary register.
W	- Auxiliary register.
X	- Register for module address.
Y	- Register for the current program module address.
PMC	- Program module code register.
RA	- Register RA of the hardware stack.
RB	- Register RB of the hardware stack.

./..

DB	- DBDI: Data bus direction control line.
AD	- ADSL: Address/data select control line.
WE	- WEQZ: control line for write enable on Q or Z.
PP	- PPOP: push/pop operation control line.
I/O	- I/O register of peripheral lines.
ZF	- Zero flip-flop.
SF	- Sign flip-flop.
CR4	- Binary carry from bit 4 to 5.
CRY	- Binary carry from bit 8.
DD	- Decimal or exadecimal number.
BBBB	- 4 bit binary number.
^	- Logic and.
V	- Logic or.
⊕	- Exclusion or.
R	- A CPU register.
IMM	- The Immediate.
N	- The logic code for I/O.

./..

INSTRUCTIONS BY SHORT IMMEDIATE

All these instructions are one byte instructions.
The immediate may be 3 bit or 4 bit long.
The format may be:

either

OP. CODE				

(4 bit immediate)

or

OP. CODE			

(3 bit immediate)

The instruction is executed in one machine cycle.
The operations performed and the signals available on the data and address bus and on the control lines during the executions are described by the following table:

CYCLE	CONTROL LINES	ADDRESS BUS		DATA BUS	PROGRAM COUNTER	INTERNAL OPERATION
	OB/AS/WE/PP	6/5/4	3/2/1			
Ø	ØØØØ	PMC	I.R.(1/3)	((Q)) → I.R. and TEMP	(Q)+1 → Q	IMMEDIATE INTO REGISTER

The execution of this instruction is overlapped with the fetch of the next instruction.
The internal operation transfers the immediate, which has been brought into the temporary register, into the addressed register (either the accumulator or the S register or the T register).

./..

1 - LOAD ACCUMULATOR BY SHORT IMMEDIATE

MNEMO

LAS IMM

Operation	operand

Operation:-Load accumulator (bit Ø to 3) by the specified immediate.

-Accumulator (bit 4 to 7) are set to Ø.

Operand :-The immediate may be expressed by decimal or exadecimal notation.
The range of the constans is: from Ø to 15 (decimal).

OBJECT

FØ to FF (Exadecimal) 1111 BBBB (Binary)

EXAMPLE

mnemo : LAS 5
object : F5 1111 Ø1Ø1

- Let ACC = 1011 0110
- At the end ACC = ØØØØ Ø1Ø1

./..

2 - LOAD S REGISTER BY SHORT IMMEDIATE

MNEMO

LSS IMM

Operation	Operand

Operation: Load S register by the specified immediate.

Operand : The immediate may be expressed by a decimal or exadecimal notation.
The range of the immediate is: from Ø to 7.

OBJECT

28 to 2F	ØØ1Ø 1BBB
(Exadecimal)	(Binary)

EXAMPLE

mnemo : LSS 6
object : 2E ØØ1Ø 111Ø

- Let S register = Ø1Ø
- At the end: S register = 11Ø

./..

3 - LOAD T REGISTER BY SHORT IMMEDIATE

MNEMO

LTS IMM

Operation	Operand

Operation: Load T register by the specified immediate.

Operand : The immediate may be expressed by a decimal or exadecimal notation.
The range of the immediate is: from Ø to 7.

OBJECT

38 to 3F	ØØ11 1BBB
(Exadecimal)	(Binary)

EXAMPLE

mnemo : LTS 3
object : 3B ØØ11 1Ø11

- Let T register = ØØØ
- At the end T register = Ø11

./..

INSTRUCTIONS BY LONG IMMEDIATE

All these instructions are a two bytes instructions.
The format is the following:

OP. CODE	1° BYTE
IMMEDIATE	2° BYTE

The instruction is executed in two machine cycles.

The operations performed and the signals available on the data and address bus and on the control lines at the execution are described on the following table:

CYCLE	CONTROL LINES	ADDRESS BUS 6/5/4	ADDRESS BUS 3/2/1	DATA BUS	PROGRAM COUNTER	INTERNAL OPERATION
0	0000	PMC	I.R.(1/3)	((Q)) → TEMP	(Q)+1 → Q	(A) → A
1	0000	PMC	I.R.(1/3)	((Q)) → I.R. and TEMP	(Q)+1 → Q	DEPENDS ON INSTR.

The first cycle is used to bring into the temporary register the immediate.
The second cycle executes the required internal operation and also fetches the next instruction.

./..

1 - LOAD ACCUMULATOR BY LONG IMMEDIATE

MNEMO

LAL IMM

Operation	Operand

Operation: Load Acc. by next byte.

Operand : Immediate (by decimal (DDD) or exadecimal notation (/DD)). The range of the immediate is: Ø to 255 decimal.

OBJECT

Ø4	ØØØØ ØlØØ	(1° Byte)
DD	BBBB BBBB	(2° Byte)
(Exadecimal)	(Binary)	

AT THE EXECUTION the following actions will take place:

CYCLE	DATA BUS	INTERNAL OPERATION	PROGRAM COUNTER
Ø	((Q)) → TEMP	(A) → A	(Q) +1 → Q
1	((Q)) → I.R. and TEMP	(TEMP) → A	(Q) +1 → Q

EXAMPLE

mnemo : LAL 1Ø (Decimal)
object : Ø4 ØØØØ Ø1ØØ
ØA ØØØØ 1Ø1Ø

- LET ACC = 1ØØ1 Ø111 at the beginning.
- At the end ACC = ØØØØ 1Ø1Ø.

./..

2 - AND LONG IMMEDIATE BY ACC

MNEMO

ANL IMM

Operation	Immediate

Operation:-Logical and of ACC. by next byte.
-The result into ACC.

Immediate: By decimal (DDD) or exadecimal notation (/DD). The range of the immediate is: Ø to 255 decimal.

OBJECT

Ø5	ØØØØ ØlØl	(1° Byte)
DD	BBBB BBBB	(2° Byte)
(Exadecimal)	(Binary)	

AT THE EXECUTION the following actions will take place:

CYCLE	DATA BUS	INTERNAL OPERATION	PROGRAM COUNTER
Ø	((Q)) → TEMP	(A) → A	(Q) +1 → Q
1	((Q)) → I.R. and TEMP	(A)∧(TEMP) → A	(Q) +1 → Q

EXAMPLE

mnemo :LAL /24 (Exadecimal)
object :Ø5 ØØØØ ØlØl
24 ØØlØ ØlØØ

- LET ACC = 1Ø1Ø ØØ11
- The Immediate is = ØØlØ ØlØØ
- At the end ACC = ØØlØ ØØØØ

./..

3 - EXCLUSIVE OR LONG IMMEDIATE BY ACC

MNEMO

EOL	IMM
Operation	Immediate

Operation: - Exclusive or of Acc. by next byte.
- Result into Acc.

Immediate: By decimal (DDD) or exadecimal (/DD) notation. The range of the immediate is: Ø to 255 decimal.

OBJECT

ØC	ØØØØ 11ØØ	(1° Byte)
DD	BBBB BBBB	(2° Byte)
(Exadecimal)	(Binary)	

AT THE EXECUTION the following actions will take place:

CYCLE	DATA BUS	INTERNAL OPERATION	PROGRAM COUNTER
Ø	((Q)) → TEMP	(A) → A	(Q) +1 → Q
1	((Q)) → I.R. and TEMP	(A) ⊕ (TEMP) → A	(Q) +1 → Q

EXAMPLE

mnemo : EOL /35 (Exadecimal)
object : ØC ØØØØ 11ØØ
35 ØØ11 Ø1Ø1

- LET ACC = 1Ø1Ø ØØ11
- The Immediate is = ØØ11 Ø1Ø1
- At the end ACC = 1ØØ1 Ø11Ø

./..

4 - OR LONG IMMEDIATE BY ACC

MNEMO

ORL	IMM
Operation	Immediate

Operation: - Logical or of ACC. by next byte.
- Result into ACC.

Immediate: By decimal (DDD) or exadecimal (/DD) notation. The range of the immediate is: Ø to 255 decimal.

OBJECT

ØD	ØØØØ 11Ø1	(1° Byte)
DD	BBBB BBBB	(2° Byte)
(Exadecimal)	(Binary)	

AT THE EXECUTION the following actions will take place:

CYCLE	DATA BUS	INTERNAL OPERATION	PROGRAM COUNTER
Ø	((Q)) → TEMP	(A) → A	(Q) +1 → Q
1	((Q)) → I.R. and TEMP	(A) ∨ (TEMP) → A	(Q) +1 → Q

EXAMPLE

mnemo : ORL /4A (Exadecimal)
object : ØD ØØØØ 11Ø1
4A Ø1ØØ 1Ø1Ø

- LET ACC = 1Ø1Ø ØØ11
- The Immediate is = Ø1ØØ 1Ø1Ø
- At the end ACC = 111Ø 1Ø11

./..

5 - ADD LONG IMMEDIATE AND ACC

MNEMO

ADL	IMM
Operation	Immediate

Operation: - Binary addition of Acc and next byte.
- The result into Acc.
- The carry is not affected by this instruction.

Immediate: By decimal (DDD) or exadecimal (/DD) notation.
The range of the immediate is: Ø to 255 decimal.

OBJECT

ØE	ØØØØ 111Ø	(1° Byte)
DD	BBBB BBBB	(2° Byte)
(Exadecimal)	(Binary)	

AT THE EXECUTION the following actions will take place:

CYCLE	DATA BUS	INTERNAL OPERATION	PROGRAM COUNTER
Ø	((Q)) → TEMP	(A) → A	(Q) +1 → Q
1	((Q)) → I.R. and TEMP	(A)+(TEMP) → A	(Q) +1 → Q

EXAMPLE

mnemo : ADL 161 (Decimal)
Object : ØE ØØØØ 111Ø
A1 1Ø1Ø ØØØ1

- LET ACC : 1Ø1Ø ØØ11
- The Immediate is : 1Ø1Ø ØØØ1
- The carry is set to : Ø
- At the end: ACC : Ø1ØØ Ø1ØØ
CARRY FF : Ø

./..

6 - COMPARE LONG IMMEDIATE AND ACC

MNEMO

CML	IMM
Operation	Immediate

Operation: - Binary addition of acc. and next byte.
- The acc. is not affected.
- Carry and zero FF are set as follows:

1 - Carry = Ø if ACC < ($\overline{\text{IMM}}$ +1)
2 - Carry = 1 if ACC ≥ ($\overline{\text{IMM}}$ +1)
3 - Zero = Ø if ACC ≠ ($\overline{\text{IMM}}$ +1)
4 - Zero = 1 if ACC = ($\overline{\text{IMM}}$ +1)

Immediate: By decimal (DDD) or exadecimal (/DD) notation. The range of the immediate is: Ø to 255 decimal.

PLEASE NOTE:

1 - The CML instruction refers to the 2' complement of the Immediate.

2 - The carry FF will keep the status until another CML or DAR (decimal addition) or ALS or ARS (shift) instruction is performed.

3 -The zero FF may be tested by the next instruction, but its status changes at the end of the next instruction.

OBJECT

Ø F	ØØØØ 1111	(1° Byte)
D D	BBBB BBBB	(2° Byte)
(Exadecimal)	(Binary)	

./..

AT THE EXECUTION the following actions will take place:

CYCLE	DATA BUS	INTERNAL OPERATION	PROGRAM COUNTER
Ø	((Q)) → TEMP	(A) → A	(Q) +1 → Q
1	((Q)) → I.R. and TEMP	(A)+(TEMP) → CRY (A) → A	(Q) +1 → Q

EXAMPLE

Mnemo : CML 16 (Decimal)

Object : ØF ØØØØ 1111
: 1Ø ØØØ1 ØØØØ

- The immediate is : ØØØ1 ØØØØ
- The 2' complement of the immediate is : 1111 ØØØØ

A - Let ACC = : ØØØØ Ø1ØØ
ACC < ($\overline{TMM}$ + 1)
- After the execution - Carry = : Ø
- Zero = : Ø

B - Let ACC = : 1111 ØØ11
ACC > ($\overline{TMM}$ + 1)
- After the execution - Carry = : 1
- Zero = : Ø

C - Let ACC = : 1111 ØØØØ
ACC = ($\overline{TMM}$ + 1)
- After the execution - Carry = : 1
- Zero = : 1

./..

AUXILIARY REGISTER INSTRUCTIONS

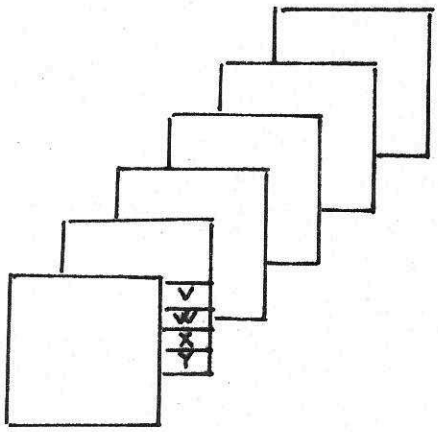
All these instructions are one byte instructions, with the following format:

OP	CODE

The instructions of this group are for:

- Loading the accumulator by the auxiliary registers.
- Storing the accumulator into the auxiliary registers.
- Storing the accumulator into T register or S/T register.

The auxiliary registers referred by the instructions are the registers located into the CPU RAM at the positions 12, 13, 14, 15.



Their symbolic name is V, W, X, Y respectively.
The registers V, W have no special functions and are for general purpose operations.
The X register is for general purpose operation and to set the module address (bit 1 to 6) referenced by these instructions:

SQX, SZQ, SIX, LIX -

./..

The Y register is directly connected (bits 4,5,6) to the PMC register and it defines the active program memory.
It is also used to set the module address (Y reg. bit 1 to 6) referenced by these instructions:

SQY, SZY, LIY -

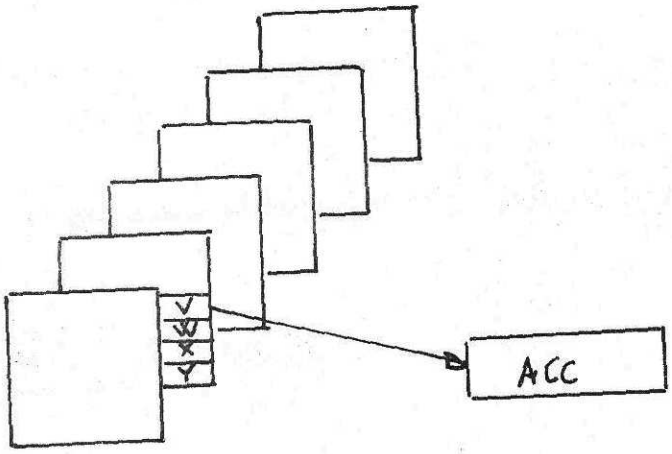
The instructions are executed in one machine cycle.
The operations performed, and the available signals on the buses and control lines during the execution are described by the following table:

CYCLE	CONTROL LINES DB/AS/WE/PP	ADDRESS BUS 6/5/4	3/2/1	DATA BUS	PROGRAM COUNTER	INTERNAL OPERATION
Ø	ØØØØ	PMC	I.R.(1/3)	((Q)) → I.R. and TEMP	(Q) +1 → Q	TRANSFER BETWEEN THE SPECIFIED REGISTER

The execution of these instructions is overlapped with the fetch of the next instruction.
The Internal operation is a transfer between the registers specified by the operation code.

./..

1 - LOAD ACCUMULATOR BY V REGISTER CONTENT



MNEMO
LAV

OBJECT	
Ø 8	ØØØØ 1ØØØ

EXAMPLE

- LET VREG = ØØ1Ø Ø11Ø
  ACC = 1ØØ1 1111
- AT THE END VREG = ØØ1Ø Ø11Ø
  ACC = ØØ1Ø Ø11Ø

./..

2 - LOAD ACCUMULATOR BY W REGISTER CONTENT

MNEMO	LAW		
OBJECT	Ø9	ØØØØ	1ØØ1

3 - LOAD ACCUMULATOR BY X REGISTER CONTENT

MNEMO	LAX		
OBJECT	ØA	ØØØØ	**1Ø1Ø**

4 - LOAD ACCUMULATOR BY Y REGISTER CONTENT

MNEMO	LAY		
OBJECT	ØB	ØØØØ	1Ø11

./..

1 - STORE ACCUMULATOR INTO V REGISTER

MNEMO
SAV

OBJECT	
18	ØØØ1 1ØØØ

EXAMPLE - LET V REG = ØØ1Ø Ø11Ø

- AT THE END V REG= 1ØØ1 1111
ACC = 1ØØ1 1111

./..

2 - STORE ACCUMULATOR INTO W REGISTER

MNEMO	SAW		
OBJECT	19	ØØØ1	1ØØ1

3 - STORE ACCUMULATOR INTO X REGISTER

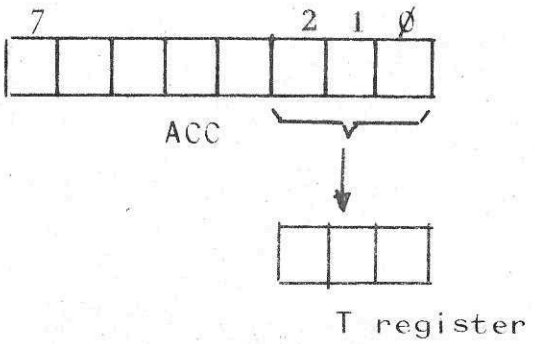
MNEMO	SAX		
OBJECT	1A	ØØØ1	1Ø1Ø

4 - STORE ACCUMULATOR INTO Y REGISTER

MNEMO	SAY		
OBJECT	1B	ØØØ1	1Ø11

./..

1 - STORE ACCUMULATOR INTO T REGISTER



MNEMO

SAT

Operation: The three least significant bits of the Accumulator are copied into the T register.
The Accumulator is unchanged.

OBJECT

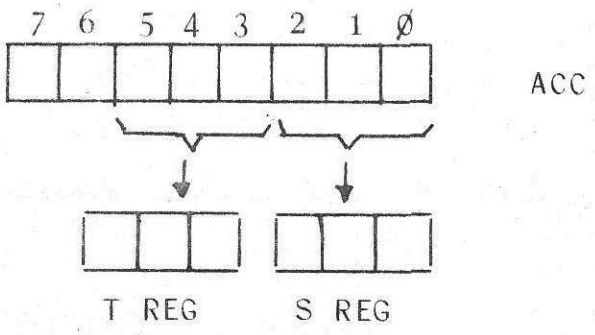
Ø1	ØØØØ ØØØ1

EXAMPLE

- LET T REG = 1Ø1
  ACC = 1Ø 111 ØØ1
- AT THE END
  T REG = ØØ1
  ACC = 1Ø 111 ØØ1

./..

2 - STORE ACCUMULATOR INTO S.T REGISTERS



MNEMO

SST

Operation:
- Bits Ø, 1, 2 of the Accumulator are copied into the S register.
- Bits 2, 3, 4 of the Accumulator are copied into the T register.
- The Accumulator is unchanged.

OBJECT

Ø3 ØØØØ ØØ11

EXAMPLE
- LET
  S REG = 111
  T REG = Ø11
  ACC = Ø1 Ø1Ø ØØØ
- AT THE END
  S REG = ØØØ
  T REG = Ø1Ø
  ACC = Ø1 Ø1Ø ØØØ

./..

SHIFT INSTRUCTIONS

The instructions of this group are one byte instructions with the following format:

OP	CODE

By these instructions the shift operation can be performed on the accumulator content.
Two of these instructions (ALS, ARS) are used to set or reset the carry flip-flops.
The instructions are executed in one machine cycle.
The operation performed, and the available signals on the buses and control lines during the execution, are described by the following table:

CYCLE	CONTROL LINES DR/AS/WE/PP	ADDRESS BUS 6/5/4	ADDRESS BUS 3/2/1	DATA BUS	PROGRAM COUNTER	INTERNAL OPERATION
Ø	####	PMC	I.R.(1/3)	((Q)) → I.R. and Temp	(Q)+1 → Q	SHIFT ON THE ACC.CONTENT

The execution of this instruction is overlapped with the fetch of the next instruction.

./..

1 - ACC SHIFT LEFT ONE BIT

MNEMO

ALS

Operation: - Acc is shifted left one bit
- LSB of Acc = Ø
- MSB of Acc is lost
- Carry FF is set to 1

OBJECT

1C	ØØØ1	11ØØ

EXAMPLES

1 - LET ACC = Ø1 11Ø 111, CARRY = 1
AFTER ALS EXECUTION:
ACC = 11 1Ø1 11Ø, CARRY = 1

2 - LET ACC = ØØ Ø11 111, CARRY = Ø
AFTER ALS EXECUTION:
ACC = ØØ 111 11Ø, CARRY = 1

./..

2 - ACC SHIFT RIGHT ONE BIT

MNEMO

ARS

Operation: - Acc is shifted right one bit.
- MSB of Acc = Ø
- LSB of Acc is lost
- Carry FF is set to Ø

OBJECT

1D	ØØØ1	11Ø1

EXAMPLES

1 - LET ACC = Ø1 11Ø 111, CARRY = 1

AFTER ARS EXECUTION:

ACC = ØØ 111 Ø11, CARRY = Ø

2 - LET ACC = ØØ ØØ1 111, CARRY = Ø

AFTER ARS EXECUTION:

ACC = ØØ ØØØ 111, CARRY = Ø

./..

3 - ACC SHIFT LEFT 4 BITS

MNEMO

ALF

Operation: - Acc is shifted left 4 bits
- 4 LSB's of Acc = Ø
- 4 MSB's of Acc are lost
- Carry is not affected

OBJECT

1E ØØØ1 111Ø

EXAMPLE

LET ACC = 11Ø1 Ø11Ø

AFTER ALF EXECUTION:

ACC = Ø11Ø ØØØØ - CARRY AS IT WAS BEFORE

4 - ACC SHIFT RIGHT 4 BITS

MNEMO

ARF

Operation: - Acc is shifted right 4 bits
- 4 MSB's of Acc = Ø
- 4 LSB's of Acc are lost
- Carry is not affected

OBJECT

1F ØØØ1 1111

EXAMPLE

LET ACC = 11Ø1 Ø11Ø

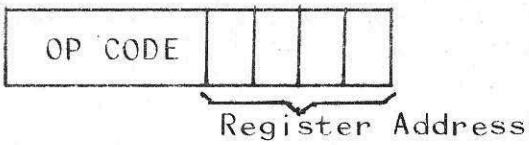
AFTER ARF EXECUTION:

ACC = ØØØØ 11Ø1 - CARRY AS IT WAS BEFORE

./..

REGISTER REFERENCE INSTRUCTIONS

The instructions of this group are one byte instruction with the following format:



All these instructions (but the DAR instruction) require only one machine cycle for the execution.
The DAR instruction is executed in two machine cycles: and it will be discussed in detail by itself.
For all the other instructions, the available signals on the buses and control lines during the execution are described by the following table:

CYCLE	CONTROL LINES DB/AS/WE/PP	ADDRESS BUS 6/5/4	3/2/1	DATA BUS	PROGRAM COUNTER	INTERNAL OPERATION
0	0000	PMC	I.R.(1/3)	((Q)) → I.R. and TEMP	(Q)+1 → Q	DEPENDS ON THE INSTRUCTIONS

The execution of these instructions is overlapped with the fetch of the next instruction.
The internal operation may be:

- A load / store between Acc and a CPU register.
- An arithmetic or logic operation between Acc and a CPU register, with result into Acc.
- A decrement operation directly performed on a register.

./..

All these instruction refer to a CPU memory register.
The CPU memory is made up of 48 registers divided in pages (a block of 8 consecutive registers) and rows (a row within a page is an 8 bit register).
These registers (see fig. 28) may be addressed:

- Directly - Only some of them (specifically registers Ø to 11) may be addressed directly.
- Indirectly - By means of the (S,T) registers. All the 48 registers may be addressed on this way.

The indirect addressing of the CPU registers is done by the page address through the T register and by the row address through the S register.
The last one may also be indexed by plus or minus 1.

The register address is specified by a 4 bit logic code placed into the 4 least significant bits of the instruction code.
The correlation between the logic address code, the addressed register and the indexing operations is indicated on the following table:

Register Logic Code		Address Type	Register Addressed	Indexing
Binary	Decimal			
0000	0	Direct	0	No
0001	1		1	
0010	2		2	
0011	3		3	
0100	4		4	
0101	5		5	
0110	6		6	
0111	7		7	
1000	8		8	
1001	9		9	
1010	10		10	
1011	11		11	
1100	12	Indirect	BY (S,T)	No
1101	13	Indirect	BY (S,T)	S-1 → S
1110	14	Indirect	BY (S,T)	S+1 → S
1111	15	Not Allowed		

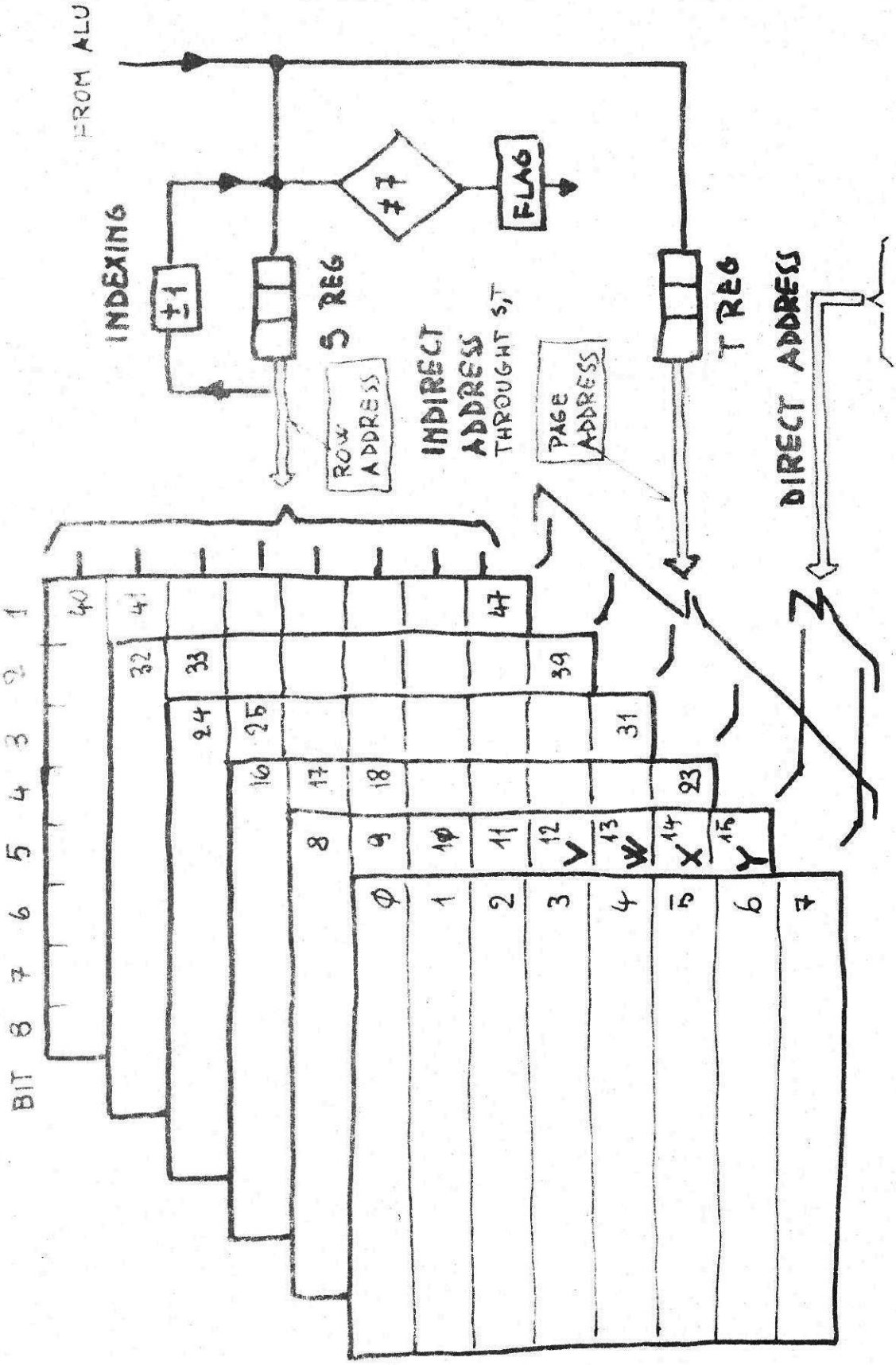


FIG 28 - CPU MEMORY

1 - LOAD ACC BY REGISTER

MNEMO

LAR R

Operation: - The register content addressed by the logic code is stored into the Acc.

- The register does not change.

Operand : - A CPU register R specified by the register logic code.
The code range is from Ø to 14 (decimal).

OBJECT

Binary: 1ØØØ BBBB

Operation	Reg. Address

2 - STORE ACC INTO REGISTER

MNEMO

SAR R

Operation: - The Acc is stored into the register addressed by the logic code.

- The Acc does not change.

Operand : - A CPU register R specified by the register logic code.
The code range is from Ø to 14 (decimal).

OBJECT

Binary: 1ØØ1 BBBB

Operation	Reg. Address

./..

3 - ADD ACC AND REGISTER

MNEMO

ADR R

Operation: - Binary addition between Acc and register addressed by the logic code.
- The result is stored into the Acc.
- The register does not change.
- The carry is not affected.

Operand : - A CPU register R specified by the register logic code.
The code range is from Ø to 14 (decimal).

OBJECT

Binary: 1Ø1Ø BBBB

Operation	Reg. Address

4 - LOGIC AND BETWEEN ACC AND REGISTER

MNEMO

ANR R

Operation: - Logic AND between acc and register addressed by the logic code.
- The result is stored into the Acc.
- The register does not change.
- The carry is not affected.

Operand : - A CPU register R specified by the register logic code.
The code range is from Ø to 14 (decimal).

OBJECT

Binary 1Ø11 BBBB

Operation	Reg. Address

./..

5 - EXCLUSIVE OR BETWEEN ACC AND REGISTER

MNEMO

EOR R

Operation: - Exclusive or between Acc and register addressed by the logic code.
- The result is stored into the Acc.
- The register does not change.
- The carry is not affected.

Operand : - A CPU register R specified by the register logic code.
The code range is from Ø to 14 (decimal).

OBJECT

Binary: 11ØØ BBBB

Operation	Reg. Address

6 - DECREMENT REGISTER

MNEMO

DER R

Operation: - Decrement by 1 the register addressed by the logic code.
- The result is stored on the addressed register.
- Acc and carry do not change.

Operand : - A CPU register R specified by the register logic code.
The code range is from Ø to 14 (decimal).

OBJECT

binary: 11Ø1 BBBB

Operation	Reg. Address

./..

EXAMPLES

Let's have the following data on page 1 and page 2 of the CPU memory:

Reg.Addr.	Reg.Content	Page	Row
8	ØØ ØØØ ØØ1	1	Ø
9	ØØ ØØØ Ø1Ø		1
10	ØØ ØØØ Ø11		2
11	ØØ ØØØ 1ØØ		3
12	ØØ ØØØ 1Ø1		4
13	ØØ ØØØ 11Ø		5
14	ØØ ØØØ 111		6
15	ØØ ØØ1 ØØØ		7
16	1Ø ØØØ ØØ1	2	Ø
17	1Ø ØØØ Ø1Ø		1
18	1Ø ØØØ Ø11		2
19	1Ø ØØØ 1ØØ		3
20	1Ø ØØØ 1Ø1		4
21	1Ø ØØØ 11Ø		5
22	1Ø ØØØ 111		6
23	1Ø ØØ1 ØØØ		7

Every example is supposed to begin with the following data into T, S registers and into the Accumulator:

T register = Ø1Ø (T points to page 2)

S register = Ø11 (S points to page 3)

Accumulator= 11 ØØ1 ØØ1

./..

EXAMPLE 1: LAR 1Ø

After the execution the Acc = ØØ ØØØ Ø11

EXAMPLE 2: LAR 12

After the execution the Acc = 1Ø ØØØ 1ØØ
and the S register = Ø11

EXAMPLE 3: LAR 13

After the execution the Acc = 1Ø ØØØ 1ØØ
and the S register = Ø1Ø

If the S, T registers are not changed, the next LAR 13 will produce the following results:

Acc = 1Ø ØØØ Ø11
S register = ØØ1

EXAMPLE 4: ADR 14

The instruction performs the binary addition between:

Acc = 11 ØØ1 ØØ1
and the addressed reg. = 1Ø ØØØ 1ØØ

The result will be:

Acc = Ø1 ØØ1 1Ø1
S register = 1ØØ
Carry flip-flop does not change.

EXAMPLE 5: ANR 11

The instruction performs the logic AND
between Acc = 11 ØØ1 ØØ1
and the addressed register = ØØ ØØØ 1ØØ

The result will be Acc = ØØ ØØØ ØØØ

./..

EXAMPLE 6: EOR 12

The instruction performs the exclusive OR

between Acc	=	11	ØØ1	ØØ1
and the addressed register	=	1Ø	ØØØ	1ØØ
The result will be Acc	=	Ø1	ØØ1	1Ø1
and the S register	=	Ø11		

EXAMPLE 7: DER 9

After the exccution the Acc is still the same while the register 9 has been decremented

Register 9	=	ØØ	ØØØ	ØØ1

./..

7 - DECIMAL ADDITION BETWEEN ACC AND REGISTER

MNEMO

DAR R

Operation: - Addition of 2 BCD bytes.
- The addition is implemented in 2 cycles:

1° Cycle - 8 bit binary addition Acc + Reg.+ Carry.
- The result is stored into the addressed register.
The carry flip-flops are properly set by the operation.
- The Accumulator does not change.

2° Cycle - Decimal correction (Addition by 10 dec = 1Ø1Ø) on one or both the 2 BCD bytes, depending on the 4 bit or 8 bit carry generated during the first cycle.
- The correction is applied if carry = Ø.
- At the end of the instruction the carry from the MSB is stored into the carry FF for next DAR operation.
- The result is stored into the addressed register, and the 8 bit carry is stored into the carry flip-flops.
- The Acc is not affected.

OBJECT

Binary:

111Ø BBBB

Operation	Reg. Address

./..

The operations performed by the DAR instruction and the available signals on the buses and control lines are described on the following table:

CYCLE	CONTROL LINES DB/AS/WE/PP	ADDRESS BUS 6/5/4	3/2/1	DATA BUS	PROGRAM COUNTER	INTERNAL OPERATION
0	0100	PMC	I.R.(1/3	1111 1111*	-------	(R)+(A)+CRY → (R), CR4, CRY (A) → A
1	0000	PMC	I.R.(1/3	((Q)) → I.R. & TEMP.	(Q)+1 → Q	IF CR4 = 0 $(R)_{1 \div 4} + 10 \rightarrow (R)_{1 \div 4}$ IF CRY = 0 $(R)_{5 \div 8} + 10 \rightarrow (R)_{5 \div 8}$

For a correct operation of the DAR instruction one of the two BCD bytes must be prepared in advance by adding to it the exadecimal constant 66.

By assuming that the byte to be prepared is on the Accumulator, the basic instructions needed for the BCD addition will be:

1 - ADL /66 Acc added by 66 exadecimal

2 - DAR R BCD addition between ACC, R, CRY

./..

EXAMPLE N.1

Let's suppose to have the following BCD numbers into the Acc and the Register R:

ACC	= 19	=	ØØØ1 1ØØ1
R	= 74	=	Ø111 Ø1ØØ

Let also CARRY= Ø

Let's now analize which are the operations performed on R and ACC by the basic program:

```
ADL   /66
DAR   R
```

1 - ADL /66 execution

ACC is added to 66 (Exadecimal)

		ØØØ1	1ØØ1 +
		Ø11Ø	Ø11Ø
ACC	=	Ø111	1111

2 - DAR R execution

1° Cycle: ACC + R + CARRY → R

CARRY	=		Ø
ACC	=	Ø111	1111
R	=	Ø111	Ø1ØØ
R	=	1111	ØØ11

During this cycle it was:

CARRY (4 bit)	=	1
CARRY (8 bit)	=	Ø

2° Cycle: The correction by 1Ø is applied only to the second byte, because CARRY (8 bit) = CRY = Ø.
So the operation performed on this cycle will be:

R	=	1111	ØØ11
CORR.FACTOR	=	1Ø1Ø	ØØØØ
R	=	1ØØ1	ØØ11 = 93

The CARRY (8 bit) = CRY will stay to Ø: it is not affected by the 2° cycle correction.

./..

EXAMPLE N. 2

The following BCD numbers are stored into the ACC and R:

ACC	= 16	=	ØØØ1 Ø11Ø
R	= 7Ø	=	Ø111 ØØØØ

and CRY = 1

Let's again analize the basic program execution:

```
ADL   /66
DAR   R
```

1 - ADL /66 execution

ACC	=	ØØØ1 Ø11Ø
66 (exadec.)	=	Ø11Ø Ø11Ø
ACC	=	Ø111 11ØØ

2 - DAR R execution

1° Cycle:

CRY	=	1
ACC	=	Ø111 11ØØ
R	=	Ø111 ØØØØ
R	=	111Ø 11Ø1

During this cycle it was:

CARRY (4bit) =	Ø
CARRY (8bit) =	Ø

2° Cycle: The correction by 1Ø is applied to byte 1 and 2.

R	=	111Ø 11Ø1
CORR.	=	1Ø1Ø 1Ø1Ø
R	=	1ØØØ Ø111

Please note that during the second cycle the carry from bit 4 is inhibited and so also the carry from bit 8.
As a result: CRY = CARRY (8 bit) = Ø.
This means that CRY will keep the information stored into it during the first cycle.

./..

EXAMPLE N. 3

As a last example, the following BCD numbers are to be added:

ACC	=	ØØ11	Ø11Ø	= 36
R	=	Ø111	Ø1ØØ	= 74

1 - ADL /66 execution

ACC	=	ØØ11	Ø11Ø
66 (Exad)	=	Ø11Ø	Ø11Ø
ACC	=	1ØØ1	11ØØ

2 - DAR R

1° Cycle:

ACC	=	1ØØ1	11ØØ
R	=	Ø111	Ø1ØØ
R	=	ØØØ1	ØØØØ
CR4	=	1	
CRY	=	1	

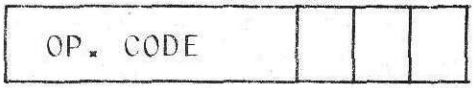
2° Cycle: no correction is applied.

R	=	ØØØ1	ØØØØ	
CORR	=	ØØØØ	ØØØØ	
R	=	ØØØ1	ØØØØ	= 1Ø
CRY	=	1		

./..

INPUT - OUTPUT INSTRUCTIONS

These instructions are one byte instruction with the following format:



I/O PORT ADDRESS

The I/O Port address is a logic code and it must be within the following range: Ø ÷ 7.
The CPU will change this logic code into the I/O Port module code at the execution time, by adding to it the quantity 56 (dec) = 111 ØØØ.
Then the correspondence between the logic code and the module code is indicated on the following table:

Logic Code	Module Code	
	Dec.	Bin.
ØØØ	56	111 ØØØ
ØØ1	57	111 ØØ1
Ø1Ø	58	111 Ø1Ø
Ø11	59	111 Ø11
1ØØ	6Ø	111 1ØØ
1Ø1	61	111 1Ø1
11Ø	62	111 11Ø
111	63	111 111

Please note that the logic code 7, equivalent to the module code 63 (dec) is assigned by hardware to the I/O Port of the CPU.
Remember also the hardware implicazions for an I/O transfer:they have been discussed under the paragraph "I/O STRUCTURES".

The input instruction is executed in two machine cycles, while the Output instruction requires three machine cycles.

The signals on the buses and control lines are described on the following paragraphs.

./..

1 - INPUT DATA FROM PERIPHERAL LINES INTO ACC

MNEMO

INP N

Operation: - Input into Acc.
- The I/O Port logic code is changed into the I/O Port module code by adding 56 (dec) = 111 ØØØ.
- The peripheral line data of the enabled I/O Port is stored into Acc.

Operand : - I/O Port logic code (Ø ÷ 7).

OBJECT

Binary: ØØ1ØØ BBB

Operation	I/O address

The operations performed and the signals on the buses and control lines are described by the following table:

CYCLE	CONTROL LINES DB/AS/WE/PP	ADDRESS BUS 6/5/4	3/2/1	DATA BUS	PROGRAM COUNTER	INTERNAL OPERATION
Ø	####	111	I.R. (1/3)	PER.LINES → → TEMP	------	(A) → A
2	####	PMC	I.R. (1/3)	((Q)) → I.R. & TEMP	(Q)+1 → Q	(TEMP) → A

./..

EXAMPLE - The system is composed of:

- CPU
- ROM Module code Ø
- I/O 1 Module code 58
- I/O 2 Module code 6Ø

LET:-	ACC	=	1Ø	11Ø 111
	PER.LINES 1	=	ØØ	1ØØ Ø1Ø
	PER.LINES 2	=	11	ØØØ ØØ1

A - If the instruction code is ØØ1ØØ Ø1Ø the I/O Port 58 is enabled and after execution ACC = ØØ 1ØØ Ø1Ø.

B - If the instruction code is: ØØ1ØØ 1ØØ the I/O Port 6Ø is enabled and after execution ACC = 11 ØØØ ØØ1.

./..

2 - OUTPUT DATA TO PER. LINES FROM ACC

MNEMO

OUT N

Operation:-Output from Acc.
- The I/O Port logic code is changed into the I/O Port module code by adding 56 (dec.) to it.
- The Acc is stored into the enabled I/O Port flip-flops, and the information is going to the per. lines via an open drain MOS.

Operand :- I/O Port logic code (Ø ÷ 7).

OBJECT

Binary: ØØ11Ø RRR

Operation	I/O Address

Note: The I/O Port of the CPU has only 4 flip-flops into which Acc (bits 5÷8) are transferred.

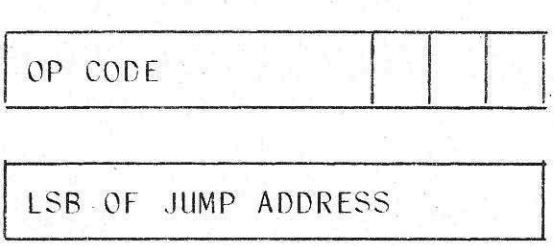
The operations performed and the signals on the buses and control lines are described by the following table:

CYCLE	CONTROL LINES DB/AS/WE/PP	ADDRESS BUS 6/5/4	ADDRESS BUS 3/2/1	DATA BUS	PROGRAM COUNTER	INTERNAL OPERATION
Ø	Ø1ØØ	111	I.R.(3/1)	TEMP	------	(A) → TEMP
1	1ØØØ	111	I.R.(3/1)	TEMP → → PER.LINES	------	(A) → A
3	ØØØØ	PMC	I.R.(3/1)	((Q)) → I.R. & TEMP	(Q)+1 → Q	(A) → A

./..

JUMP INSTRUCTIONS

The jump instructions are two byte instructions with the following format:



The "RET" instruction included on this group is a one byte instruction.

The jump address may be specified either by a number (decimal or exadecimal) or by a label.
The number of bits for the jump address are 11.
It follows that the address range for a jump instruction is from Ø to 2Ø47 (decimal).
In other words, the direct addressing field is 2K of ROM (equivalent to 8 ROM modules).
The jump instruction is effective in the 8 ROM modules which are enabled by the address bus bits 6/5/4.
As it has been indicated by the format, the jump address is divided in two parts:

- The three most significant bits are stored into the first byte.
- The other 8 bits are stored into the next byte.

When the instruction is executed, while the first byte is into the Instruction Register, the second byte is read into the temporary register.
Then the complete 11 bit address is transferred into the enabled program counter via the ADB (bit 3/2/1) and the data bus (see also fig. 7, and the paragraph "Program Counter").

./..

The jump instructions may be divided in three different categories:

1 - Unconditional jumps
2 - Conditional jumps
3 - Jump to subroutine and return

The jump conditions and their use is the object of the following discussion.
The CPU has four flop-flops which can be tested:

1 - The carry flip-flop from the most significant bit of the ALU.
This flip-flop is set to Ø or 1 by the results of the ADD operation performed either by the CML or by the DAR instruction.
Two other instructions are available to set or reset the flip-flop:
- The ALS instruction sets the flip-flop to 1.
- The ARS instruction sets the flip-flop to Ø.

2 - The zero flip-flop.
This flip-flop is properly set to Ø or 1 by the result of the DER or DAR instructions.
For every other instruction (but LSS and LTS), the flip-flop is set or reset by the Accumulator content.
The two instructions LSS, LTS bring the zero flip-flop into the "Non Zero" position.
The conditional jump instructions are referred to the Accumulator or Register content (jump if ACC = Ø, jump if ACC ≠ Ø).

./..

3 - The index register flip-flop.
This flip-flop is set to Ø or 1 by the S register content.
The conditional jump instruction by which the S register can be tested, refers to the "different then 7" content of the S register.

4 - The sign flip-flop.
This flip-flop is set to Ø or 1 by the most significant bit of the result, when either a DER or DAR instructions are executed.
For every other instruction (but LSS and LST), the flip-flop is set or reset by the most significant bit of the Accumulator.
The two instructions LSS and LTS bring the signal flip-flop into the "positive sign" condition.
The conditional jump instructions always refer to the positive or negative sign either of the Accumulator or the Register.

Remember that:

- Positive sign means the most significant bit of accumulator or register equal to Ø.
- Negative sign means the most significant bit of Accumulator or register equal to 1.

The jump instructions are executed:

A - In two machine cycles if the condition is not true.
The operations performed and the available signals on the buses during the execution are described by the following table:

./..

CYCLE	CONTROL LINES DB/AS/WE/PP	ADDRESS BUS 6/5/4	3/2/1	DATA BUS	PROGRAM COUNTER	INTERNAL OPERATIONS
Ø	ØØØØ	PMC	I.R.(3÷1)	((Q)) → TEMP	(Q)+1 → Q	(A) → A
3	ØØØØ	PMC	I.R.(3÷1)	((Q)) → I.R. & TEMP	(Q)+1 → Q	(A) → A

B - In three machine cycles if the jump condition is true.
On this case, the operations and signals are indicated on the following table:

CYCLE	CONTROL LINES DB/AS/WE/PP	ADDRESS BUS 6/5/4	3/2/1	DATA BUS	PROGRAM COUNTER	INTERNAL OPERATION
Ø	ØØØØ	PMC	I.R.(3÷1)	((Q)) → TEMP	(Q)+1 → Q	(A) → A
1	11ØØ	PMC	I.R.(3÷1) → → Q (9÷11)	(TEMP) → Q(1÷3)	.....	(A) → A
3	ØØØØ	PMC	I.R.(3÷1)	((Q)) → I.R. & TEMP	(Q)+1 → Q	(A) → A

C - The "jump to subroutine" is a three machine cycle instruction with tha same characteristics described under the comma B.
The only changes are the control lines signals during the second cycle (they are 11Ø1).
This is causing the push operation on the Q/RA/RB/Z stack.

D - The "return" instruction is a two machine cycle instruction.
The characteristics are the same as the one described at comma A with the following change: during the first cycle the control line signals are Ø1Ø1.

./..

1 - JUMP UNCONDITIONAL

MNEMO

JMP LABEL

Operation: After the jump instruction the program will continue from the jump address.

Operand : Jump address specified by a symbolic name.

OBJECT

ØlØØØ B_1 B_2 B_3

BBBB $BBBB_{11}$

The jump code is : ØlØØØ
The jump address is : $B_1, B_2, B_3, B_4, B_5, B_6, B_7, B_8, B_9, B_{10}, B_{11}$.

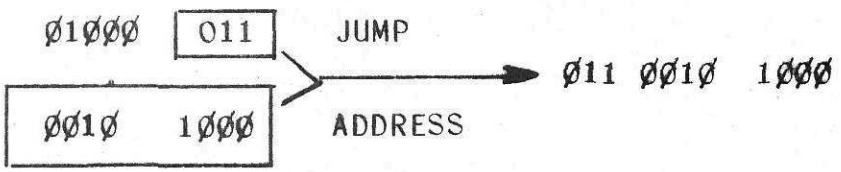
EXAMPLE

ADDRESS (Ex)	CODE (Ex)	MNEMONIC	
ØØ9	═══	═══	
ØØA	43	JMP	ERR
ØØB	28		
	═══	═══	
	═══	═══	
	═══	═══	
328	FØ	ERR LAS	Ø

When the program counter has the value:

PC = ØØA = ØØØ ØØØØ 1Ø1Ø

the CPU will execute the jump instructions.



After execution:

PC = Øll ØØ1Ø 1ØØØ

./..

2 - JUMP IF ACC = Ø

MNEMO

JAZ LABEL

Operation: When the instruction is executed, the program will continue from the jump address if ACC = Ø, otherwise the program will go ahead to the next instruction.

Operand : Jump address specified by a symbolic name.

OBJECT

ØlØØl BBB

BBBB BBBB

3 - JUMP IF ACC ≠ Ø

MNEMO

JAN LABEL

Operation: When the instruction is executed, the program will continue from the jump address if ACC ≠ Ø, otherwise the program will execute the next instruction.

Operand : Jump address specified by a symbolic name.

OBJECT

ØlØlØ BBB

BBBB BBBB

./..

EXAMPLES: Conditional jump on register result.

Let's have the following program:

ADDRESS (EX)	CODE (EX)	MNEMONIC	
═══	═══	═══	
ØØ2	═══	═══	
ØØ3	DA	DER	1Ø
ØØ4	48	JAZ	LABX
ØØ5	12	═══	
═══	═══	═══	
═══	═══	═══	
Ø12	═══	LABX ═══	

A - At PC = ØØ3 Let's assume to be:

REG 1Ø = ØØ ØØØ ØØ1
ACC = ØØ 111 111

This will be the program execution:

1 - DER 1Ø execution: REG 1Ø = ØØ ØØØ ØØØ
ACC = ØØ 111 111

2 - Program counter is incremented: PC = ØØ4

3 - JAZ LABX execution:
The conditional jump comes after an operation on register, then the jump is performed by looking at the register content, which is now equal to Ø.
Then: PC = Ø12

./..

B - At PC = ØØ3 Let's assume to be:

REG 1Ø	=	ØØ	ØØØ ØØØ
ACC	=	ØØ	ØØØ ØØØ

This will be the program execution:

1 - DER 1Ø execution: REG 1Ø = 11 111 111
ACC = ØØ ØØØ ØØØ

2 - Program counter is incremented:
PC = ØØ4

3 - JAZ LABX execution:
it will not affect the normal increment of the program counter.
Then: PC = ØØ5

EXAMPLES - Conditional jump on Accumulator.

Let's have the following program:

ADDRESS (EX)	MNEMONIC		
ØØ2		DER	1Ø
ØØ3		LAR	9
ØØ4		JAZ	LABX
═══		═══	
═══		═══	
═══		═══	
Ø12	LABX	═══	

At PC = ØØ2 let's assume to have the following situation:

REG 9	=	ØØ ØØØ	ØØ1
REG 1Ø	=	ØØ ØØØ	ØØ1
ACC	=	ØØ 111	111

This will be the program execution:

1 - DER 1Ø execution:

REG 9	=	ØØ ØØØ	ØØ1
REG 1Ø	=	ØØ ØØØ	ØØØ
ACC	=	ØØ 111	111

2 - Program counter is incremented:

PC	=	ØØ3

3 - LAR 9 execution:

REG 9	=	ØØ ØØØ	ØØ1
REG 1Ø	=	ØØ ØØØ	ØØØ
ACC	=	ØØ ØØØ	ØØ1

4 - Program counter is incremented:

PC	=	ØØ4

5 - JAZ LABX

The Accumulator is not zero, then the program will continue with the next instruction.

The program counter is incremented:

PC = ØØ5

Other esamples of conditional jump affected by the Accumulator status are:

```
A -          DER    1Ø
             LAR     9
             JMP    LABY
             ===
             =====
             ===
             ===
             LAR     8
LABY         JAZ    LABX
             ===
             ===
             ===
LABX         ===
             ===

B -          DER    1Ø
             SIX
             JAZ    LABX
             ===
             ===
             ===
LABX         ===
             ===
```

4 - JUMP IF ACC POSITIVE

MNEMO

JAP LABEL

Operation: Jump if bit 8 of Acc = Ø , otherwise the program will go ahead with the next instruction.

Operand : Jump address.

OBJECT

Ø1Ø11 BBB

BBBB BBBB

EXAMPLE:

ADDRESS	MNEMONIC	
ØØ3	DER	1Ø
ØØ4	JAP	LABX
═══	═══	
═══	═══	
═══	═══	
Ø12	LABX ═══	

A - Status at PC = ØØ3 : REG 1Ø = 1

- DER Execution : REG 1Ø = Ø
- PC = ØØ4
- JAP LABX execution will change PC to
- PC = Ø12

B - Status at PC = ØØ3 : REG 1Ø = 11 111 111

- DER Execution : REG 1Ø = 11 111 11Ø
- PC = ØØ4
- JAP LABX execution will not affect the normal PC increment
- PC = ØØ6

./..

5 - JUMP IF S REG ≠ 7

MNEMO

JSD LABEL

Operation: Jump to Label if S register ≠ 7.

Operand : Jump address.

OBJECT

Ø11ØØ BBB

BBBB BBBB

EXAMPLE:

ADDRESS	MNEMONIC		
ØØ3	LAR	*-	(see Note)
ØØ4	JSD	LABX	
	═══		
	═══		
	═══		
312	LABX ═══		

Note: Load ACC indirect by (S,T) then decrement S.

A - Status at PC = ØØ3 S REG = 7 = 111

- LAR *- execution 1 - REG (S,T) ⟶ ACC
  2 - S REG = 11Ø
- PC = ØØ4
- JSD LABX execution : PC = 312

B - Status At PC = ØØ3 S REG = Ø

- LAR *- execution 1 - REG (S,T) ⟶ ACC
  2 - S REG = 111
- PC = ØØ4
- JSD LABX execution : PC = ØØ6

./..

6 - JUMP IF CARRY ≠ Ø

MNEMO

JCN LABEL

OBJECT

Ø11Ø1 BBB
BBBB BBBB

EXAMPLE:

ADDRESS	MNEMONIC		
ØØ3		DER	1Ø
ØØ4		JCN	LABX
		═══	
		═══	
		═══	
Ø12	LABX	═══	

Status at PC = ØØ3:

1 - CARRY = Ø From a previous CML or ALR or DAR.
2 - REG 1Ø = Ø

- DER execution : CARRY= Ø not affected.
  REG 1Ø= 11 111 111
- PC = ØØ4
- JCN LABX execution : PC = ØØ6

./..

7 - JUMP IF CARRY = Ø

MNEMO

JCZ LABEL

OBJECT

Ø111Ø BBB
BBBB BBBB

EXAMPLE

ADDRESS (EX)	CODE (EX)	MNEMONIC		
ØØ3	ØF		CML	/CØ
ØØ4	CØ			
ØØ5	58		JAP	LABX
ØØ6	3Ø			
ØØ7	73		JCZ	LABY
ØØ8	62			
═══	═══		═══	
═══	═══		═══	
Ø3Ø	═══	LABX	═══	
═══	═══		═══	
═══	═══		═══	
═══	═══		═══	
362	═══	LABY	═══	

Status at PC = ØØ3 ACC = 11 111 111

- CML /CØ execution : IMM = 1ØØ ØØØØ Added to ACC
  as a result : CARRY = 1
  Sign FF set to negative
- PC = ØØ5 / ØØ6
- JAP LABX execution : ACC neg then no Jump
- PC = ØØ7 / ØØ8
- JCZ LABY execution : CARRY = 1 from CML exec then Jump
- PC = 362

./..

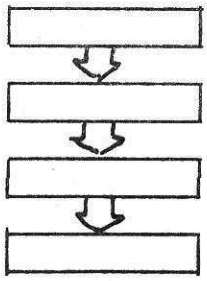
8 - JUMP TO SUBROUTINE

MNEMO

JSB LABEL

Operation: -PC is incremented.
-A push operation is performed on a 4 level stack.

Operand : Subroutine address.



OBJECT

01111 BBB

BBBB BBBB

NOTE

A jump to subroutine in a ROM chip not implemented on the system is automatically generating a RETURN (code ØØ ØØØ ØØØ) as a next instruction.

This is due to the fact that during the fetch of the next instruction, the code read into the instruction register is Ø (data bus precharged).

./..

EXAMPLE

- First K ROM program - Module code Ø -

ADDRESS (OCT)	MNEMONIC
Ø	═══
	═══
	═══
	JSB SQRT
	═══
	═══
1777	═══

- Second K ROM program - Module code Ø -

ADDRESS (OCT)	MNEMONIC
2ØØØ	═══
═══	═══
═══	SQRT ═══
═══	═══
3777	

If the second K ROM has not been put into the system, after the JSB SQRT execution, the next code to be executed is ØØ ØØØ ØØØ which means that the program returns to the instruction next to JSB SQRT.

9 - RETURN FROM SUBROUTINE

MNEMO
RET

Operation: A pop operation is performed on a 4 level stack.

OBJECT
ØØØØ ØØØØ

./..

MODULE REFERENCE INSTRUCTIONS

These instructions are one byte instructions with the following format:

OP. CODE

The use of these instructions is:

A - Indirect READ operation of either a constant (from ROM) or a variable data (from RAM or I/O Port).
The instructions belonging to this class are: LIX, LIY.

B - Indirect WRITE operation of a variable data into either a RAM or an I/O Port.
The instruction for that is the SIX instruction.

C - To WRITE the address of a constant or a variable data into the data address register (the Z register).
The instructions for this are: SZX, SZY.

D - To change the program counter content. (the Q register).
The change may be done either on the enabled program memory module or on any other program memory module.
The first case is equivalent to an unconditional jump execution and it is usefull as a program branch in a look-up table technique.
The second case is a program counter preset before jumping from the enabled 2K program memory into another 2K program memory (module range change).
This will be explained in details by the example N. 1 and 2 on the next section.
The instruction for these operations are: SQX and SQY.

The instructions of this group use are indirect module address.
The module address to be used by the some instructions (LIX, SIX, SZX, SQX) must be prepared in advance into the X register.
The other instructions (LIY, SZY, SQY) use the current module address.

The number of cycles and the available signals on the buses and control lines are described instruction by instruction.

./..

1 - STORE ACC INDIRECT BY REGISTER X

MNEMO

SIX

- The CPU is enabling the module having the code = X content.
- The ACC is stored into the module.
  - A - If the module is a RAM the ACC is stored into the word addressed by Z.
  - B - If the module is an I/O Port the ACC is stored into the I/O Port flip-flops.

OBJECT

Ø2 ØØØØ ØØ1Ø

OPERATIONS AND SIGNALS

This instruction is executed in three machine cycles. The operations performed and the signals available on the buses and control lines are described on the following table:

CYCLE	CONTROL LINES DB/AS/WE/PP	ADDRESS BUS 6/5/4	3/2/1	DATA BUS	PROGRAM COUNTER	INTERNAL OPERATION
Ø	Ø1ØØ	PMC	I.R.(3÷1)	(TEMP)	-----	(A) → TEMP
1	1###	X(6÷4)	X (3÷1)	(*)	-----	(A) → A
3	####	PMC	I.R.(3÷1)	((Q)) → I.R and TEMP	(Q)+1 → Q	(A) → A

./..

*During this cycle, the data bus content, coming from the accumulator, is transferred into the enabled module:

1 - If the address bus enables a RAM module, the data bus is copied into the RAM word addressed by the Z register.

2 - If the address bus enables an I/O Port, the data bus is copied into the I/O Port flip-flops.

EXAMPLE 1 (SEE FIG. 29) THE SYSTEM IS COMPOSED OF:

- CPU
- ROM Module code Ø
- RAM Module code ØØ1 ØØØ
- I/O PORT Module code 56 (DEC)

Let

X REG	= ØØ ØØ1	ØØØ
Z REG (RAM)	= ØØØ	ØØ1Ø
RAM WORD (Z)	= 1Ø1Ø	ØØØ1
I/O 56 (DEC)	= ØØØ1	11ØØ
ACC	= 1Ø11	Ø11Ø

After SIX execution only the RAM Word (Z) is changed to

1Ø11 Ø11Ø

./..

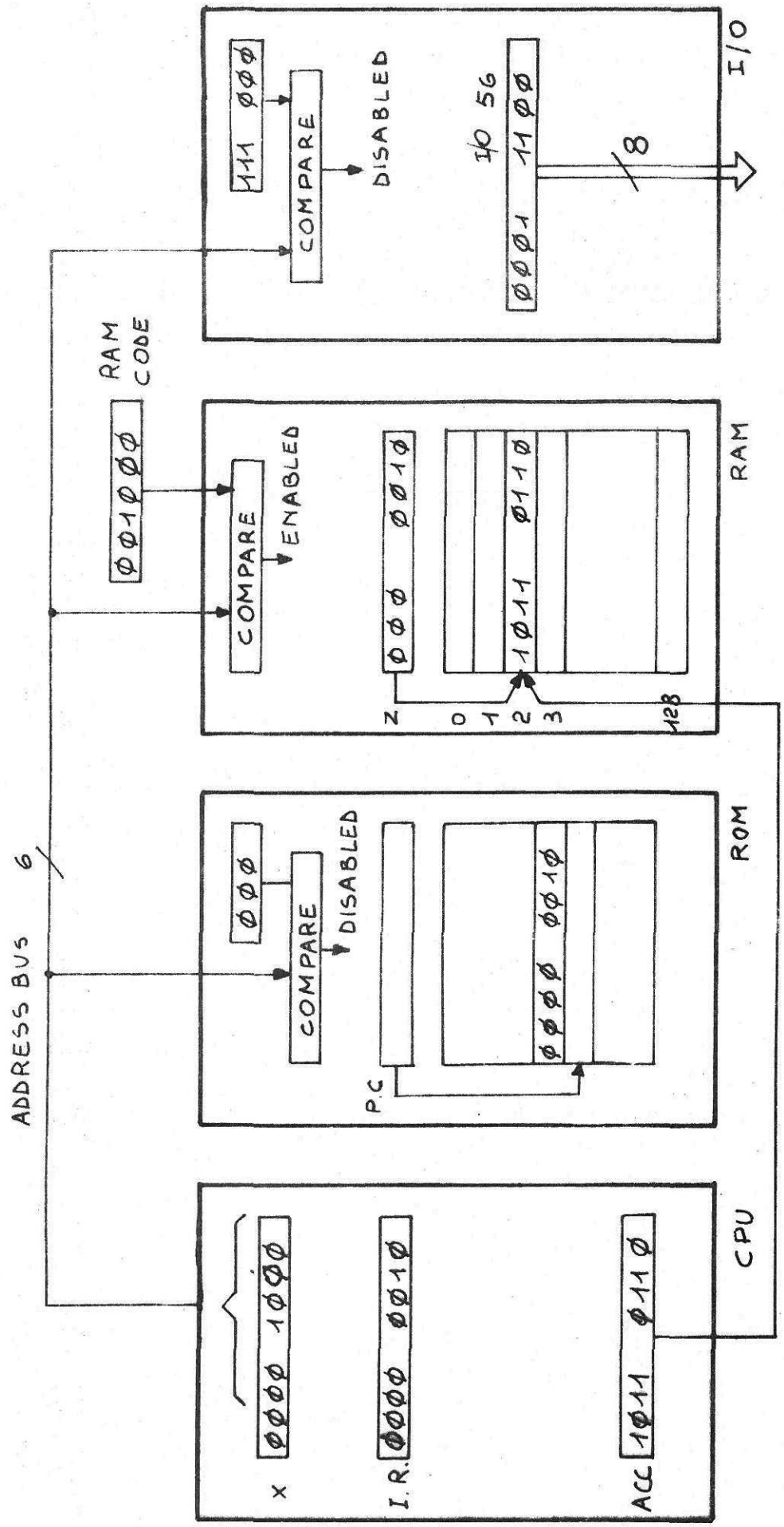


FIG 29- SIX EXECUTION - RAM

EXAMPLE 2 (SEE FIG. 30) THE SYSTEM IS COMPOSED OF

- CPU
- ROM Module code Ø
- RAM Module code ØØ1 ØØØ
- I/O PORT Module code 56 (DEC)

Let
X REG = 56 (DEC)
Z REG. (RAM) = ØØØ ØØ1Ø
RAM WORD (Z) = 1Ø1Ø ØØØ1
I/O 56 (DEC) = ØØØ1 11ØØ
ACC = 1Ø11 Ø11Ø

After SIX execution only the I/O Port is changed to 1Ø11 Ø11Ø

./..

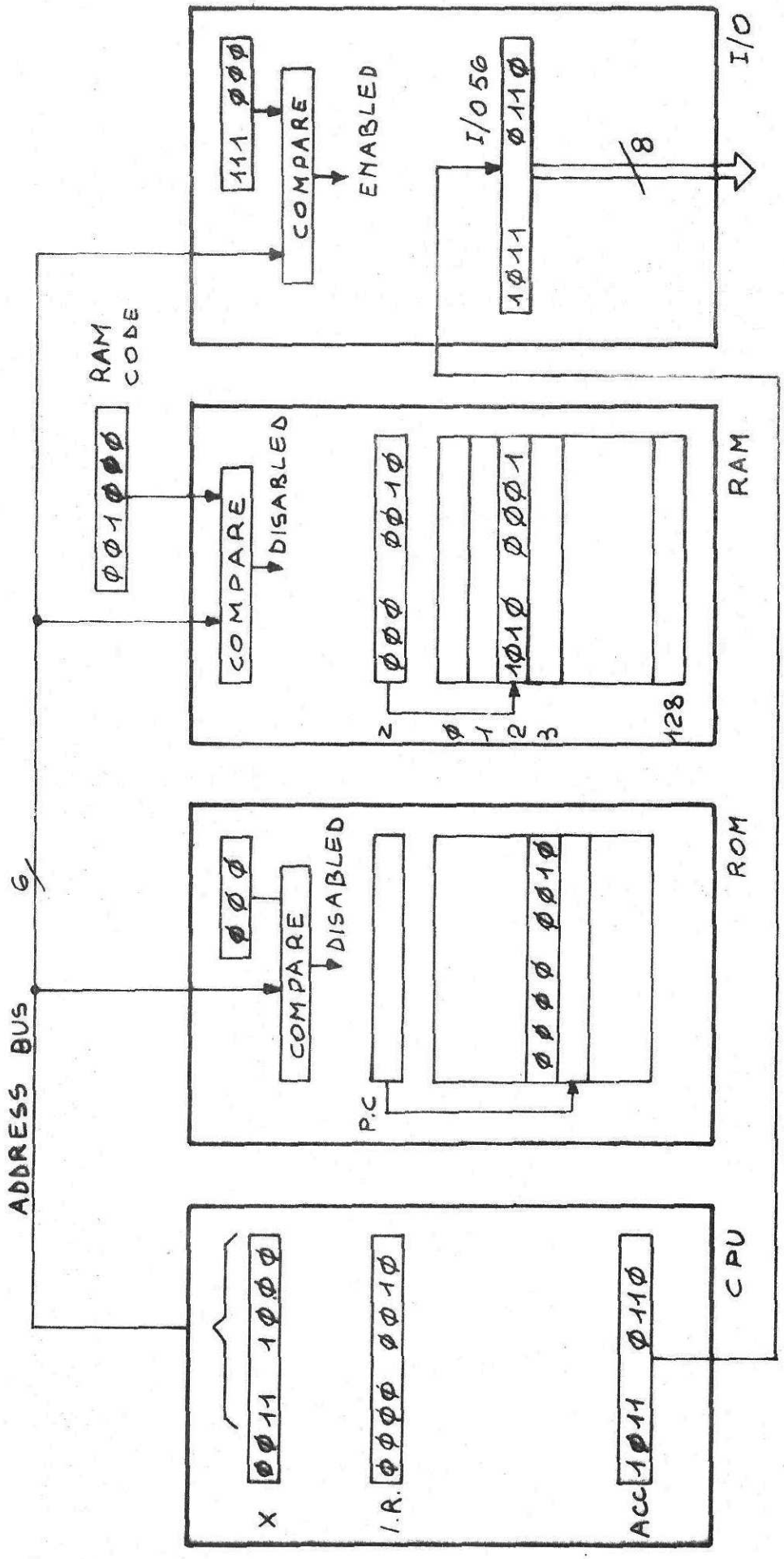


Fig 30 - SIX EXECUTION - I/O

2 - LOAD ACC INDIRECT BY X REGISTER

MNEMO

LIX

- The CPU enables the module having the code = X register.
- The data coming from the enabled module is stored into the accumulator.

The enabled module may be:

A - RAM module or ROM module.
The word addressed by Z is transferred into the ACC.

B - I/O PORT.
The data coming from the peripheral lines is transferred into the ACC.

OBJECT

Ø6 ØØØØ Ø11Ø

./..

OPERATIONS AND SIGNALS

This instruction is executed in 4 machine cycles, which are described by the following table:

CYCLE	CONTROL LINES DB/AS/WE/PP	ADDRESS BUS 6/5/4	ADDRESS BUS 3/2/1	DATA BUS	PROGRAM COUNTER	INTERNAL OPERATION
0	0010	X(6÷4)	X(3÷1)	(*)	-----	(A) → A
1	0010	X(6÷4)	X(3÷1)	(*)	-----	(A) → A
2	0100	PMC	I.R.(3÷1)	((Q)) → I.R. and TEMP	-----	(TEMP) → A
3	0000	PMC	I.R.(3÷1)	((Q)) → I.R. and TEMP	(Q)+1 Q	(A) → A

(*) During these cycles, the data coming from the enabled module is copied on the data bus.
At the end of the second cycle, the data bus is transferred into the temporary register.

The enabled module may be:

1 - RAM or ROM device.
On this case the word addressed by the Z Register is copied on the data bus.
Please note that a RAM module is addressed by X (1 ÷ 6), while a ROM module address requires only X (4 ÷ 6).

2 - I/O PORT.
On this case the peripheral lines data are copied into the data bus.
The module address is X (6 ÷ 1).

./..

EXAMPLE (FIG. 31) -

The system is composed of:

- CPU
- ROM Module code Ø
- RAM Module code ØØ1 ØØØ
- I/O PORT Module code 56 (DEC)

LET:
- Z REG (ROM) = ØØØØØØØØØ11
- ROM WORD (Z) = ØØØ1 111Ø
- Z REG (RAM) = ØØØØ1ØØ
- RAM WORD (Z) = 11Ø1 11ØØ
- I/O 56 (DEC) = OFF (ALL Ø's)
- PER BUS = 1111 Ø1ØØ
- ACC = Ø1Ø1 1Ø11

A - IF X REG = Ø

After the LIX execution the ROM WORD addressed by Z is transferred into the ACC:
- ACC = ØØØ1 111Ø

B - IF X REG = 1Ø (EX)

After the LIX execution the RAM WORD addressed by Z is transferred into the ACC:
- ACC = 11Ø1 11ØØ

C - IF X REG = 56 (DEC)

After the LIX execution the peripheral bus data is transferred into the ACC:
- ACC = 1111 Ø1ØØ

./..

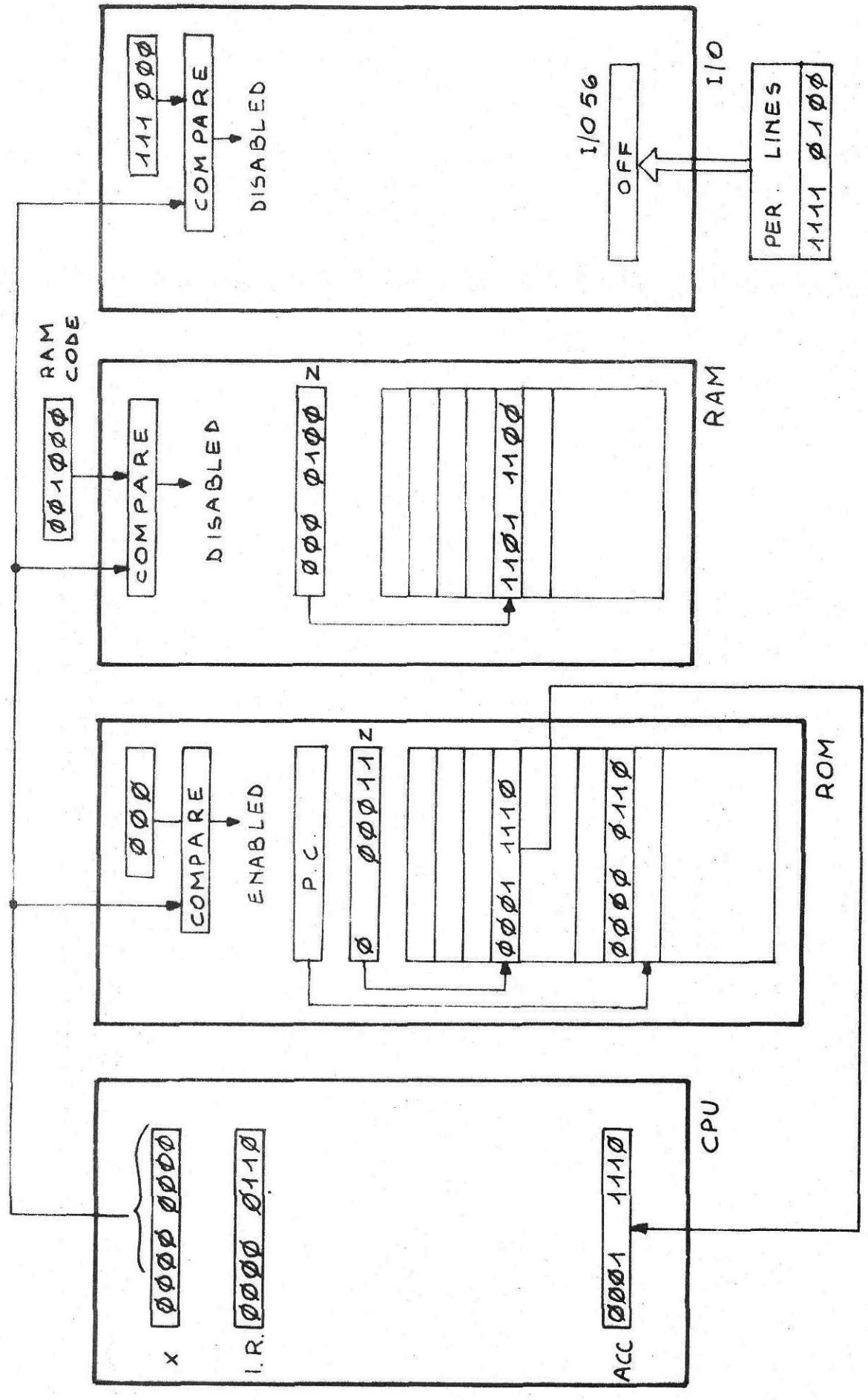


FIG 31 - LIX EXECUTION - ROM

3 - LOAD ACC. INDIRECT BY Y REGISTER

MNEMO

LIY

OBJECT

Ø7 ØØØØ Ø111

OPERATION: Same as "LIX" but the CPU is enabling a module having the code equal to Y register.
This means that the module on which the read operation has to be performed, is the ROM device whose program is in execution.

*/**

4 - STORE ACC INDIRECT INTO Q BY X REGISTER

MNEMO

SQX
- The CPU enables the module having the code = X REG. (bits 6/5/4).
- The Q REG. of this module is changed by transferring into it
  1 - The ACC (8-1) on Q (8-1)
  2 - The X REG (bits 3/2/1) on Q (11/10/9).
- The enabled module must be a ROM.

OBJECT

16 ØØØ1 Ø11Ø

NOTE : 1 - if X ≠ Y this instruction does not change the program counter in operation

2 - if X = Y this instruction changes the program counter in operation.

./..

OPERATIONS AND SIGNALS

This instruction is executed in 3 machine cycles which are described by the following table:

CYCLE	CONTROL LINES DB/AS/WE/PP	ADDRESS BUS 6/5/4	3/2/1	DATA BUS	PROGRAM COUNTER	INTERNAL OPERATION
0	0100	PMC	I.R.(3÷1)	(TEMP)	-----	(A) → TEMP
1	1100	X (6÷4)	X (3÷1)	(TEMP) (*)	-----	(A) → A
3	0000	PMC	I.R.(3÷1)	((Q)) → I.R. and TEMP	(Q)÷1 → Q	(A) → A

(*) During this cycle, the data coming from the Accumulator (through the temporary register) is stored into the program counter Q (bit 8 ÷1), belonging to the ROM module enabled by X (6 ÷ 4).

The same program counter Q (bits 11 ÷ 9) are modified by bits X (3 ÷ 1) coming through the address bus.

./..

EXAMPLE (FIG. 32)

The system is composed of:

- CPU
- ROM 1 Module code ØØØ XXX
- ROM 2 Module code ØØ1 XXX
- I/O PORT Module code 56 (DEC)

Let X REG = ØØØØ1ØØØ

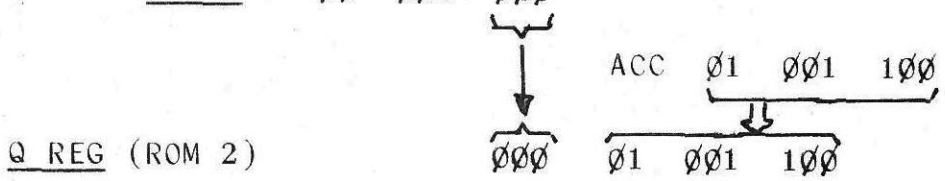
Y REG = Ø

Q REG (ROM 1) = ØØ 1ØØ 111 Ø1Ø

Q REG (ROM 2) = 1Ø 1ØØ Ø11 111

ACC = Ø1 ØØ1 1ØØ

After the SQX execution, the Q register (ROM 2) is modified as follows:



*/**

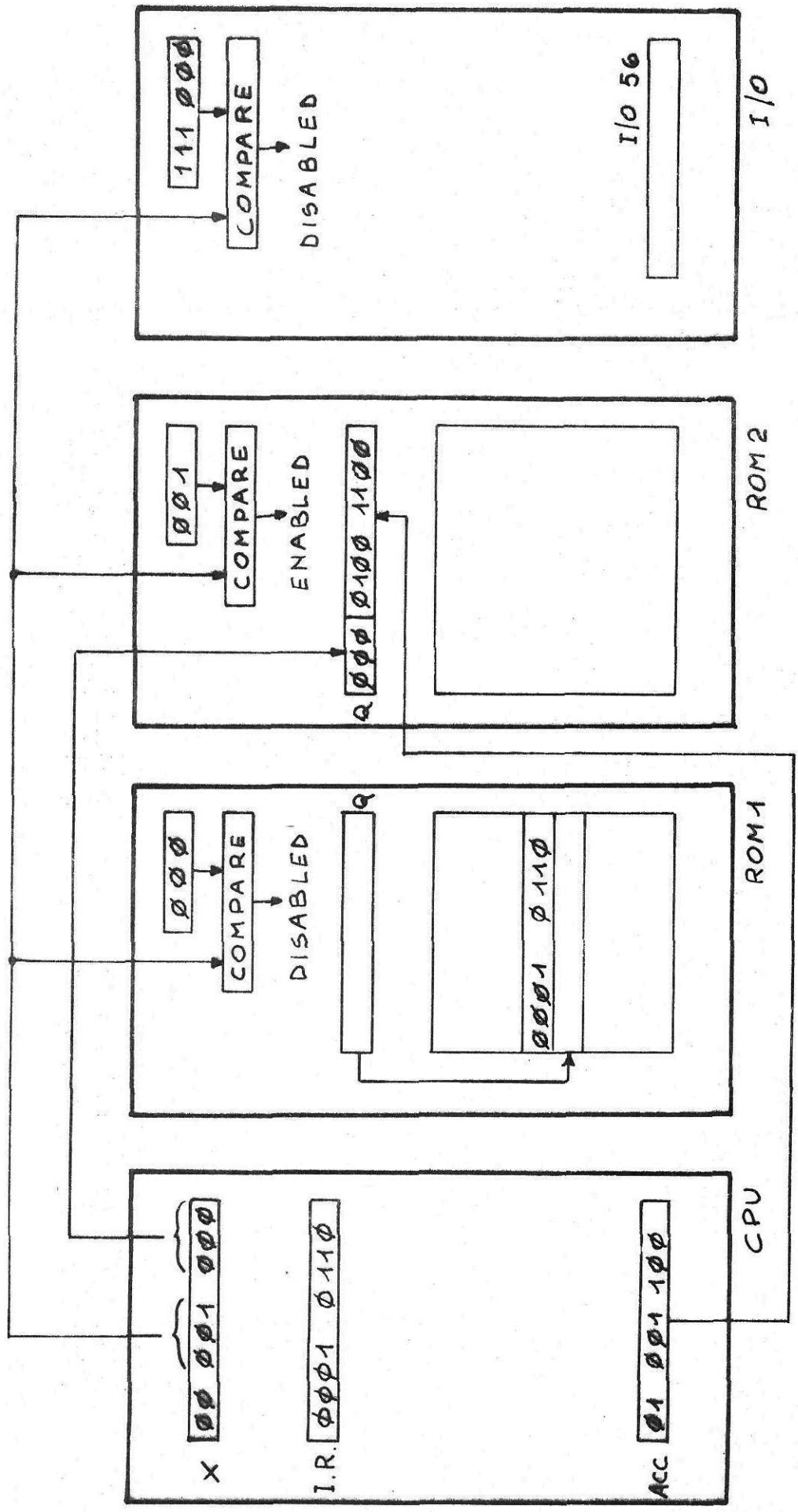


FIG 32 - SQX EXECUTION

5 - STORE ACC. INDIRECT INTO Q BY Y REGISTER

MNEMO

SQY

OBJECT

17 ØØØ1 Ø111

Operation: Same as "SQX" but the CPU enables a module having the code = Y REG.

At the end of execution, the current program counter is changed.

./..

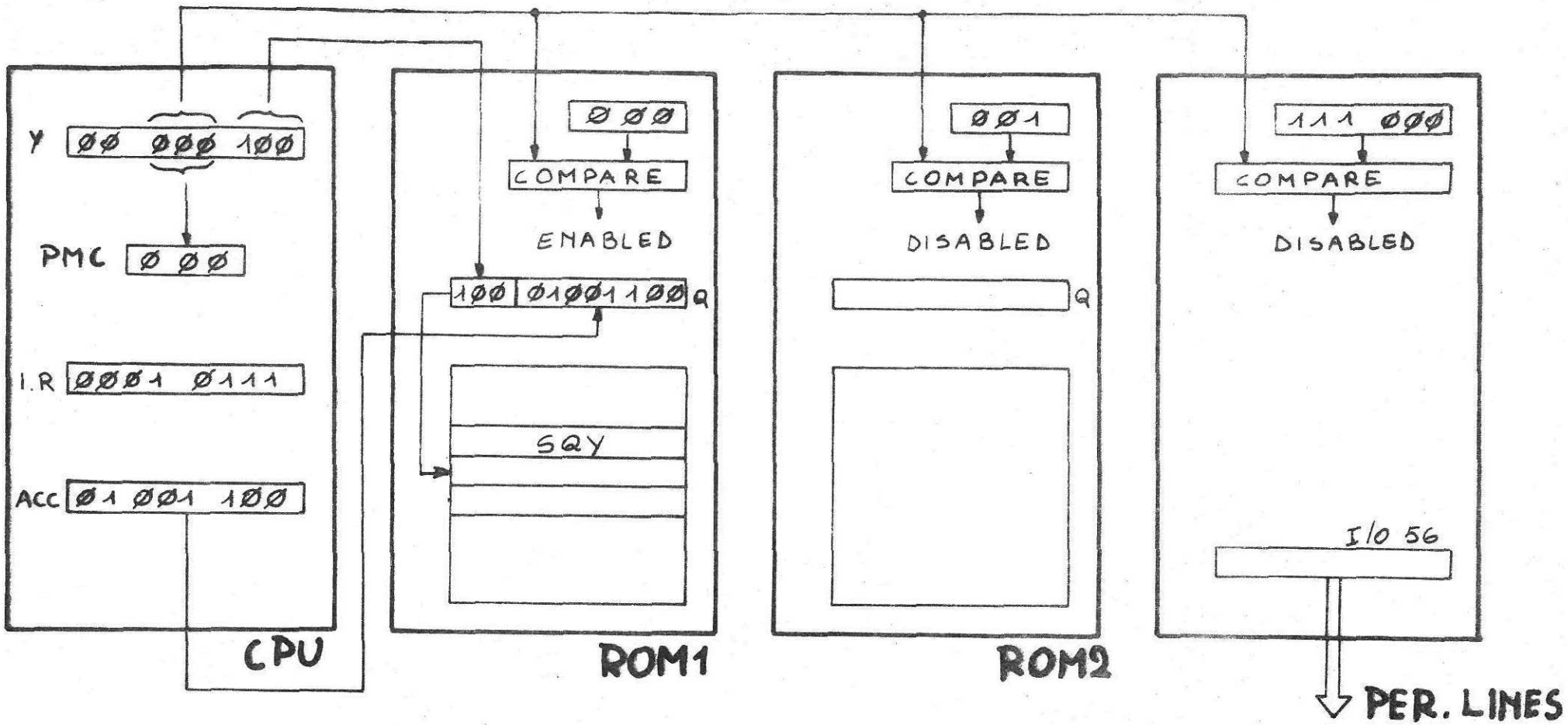


FIG 33 - SQY EXECUTION

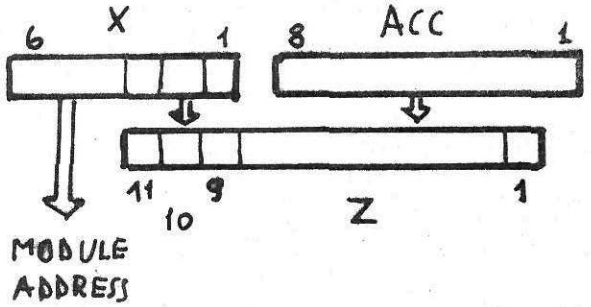
6 - STORE ACC INDIRECT INTO Z BY X REGISTER

MNEMO

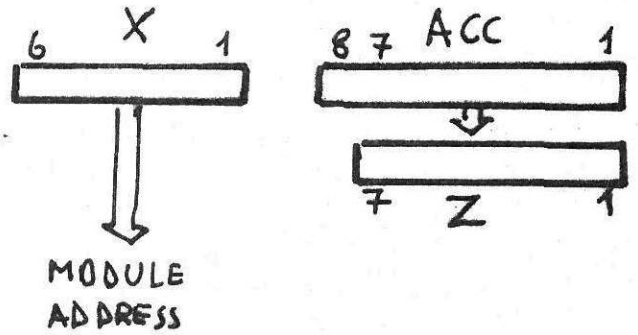
SZX - The CPU is enabling the module having the code = X Register (bits 6 ÷ 1).

- The module may be:

1 - ROM
The X Register (bits 3/2/1) is stored into Z Register (bits 11/10/9).
The ACC is stored into Z Register (bits 8 ÷ 1)



2 - RAM
The ACC (bits 7 ÷ 1) is stored into Z Register (bits 7 ÷ 1).



OBJECT

12 ØØØ1 ØØ1Ø

./..

OPERATIONS AND SIGNALS

This instruction is executed in 3 machine cycles, which are described by the following table:

CYCLE	CONTROL LINES DB/AS/WE/PP	ADDRESS BUS 6/5/4	3/2/1	DATA BUS	PROGRAM COUNTER	INTERNAL OPERATION
0	0100	PMC	I.R.(3÷1)	(TEMP)	-----	(A) → TEMP
1	1110	X (6÷4)	X (3÷1)	(*)	-----	(A) → A
3	0000	PMC	I.R.(3÷1)	((Q)) → I.R. and TEMP	(Q) +1 → Q	(A) → A

(*) During this cycle, the Z register of the module enabled by the address bus is changed:

A - If the addressed module is a ROM, the data coming from the Accumulator (through the temporary register) are copied into the Z register (bit 8 ÷ 1).
The X Register (bit 3 ÷ 1) are copied into the same Z Register (bit 11 ÷ 9).

B - If the addressed module is a RAM, the data coming from the Accumulator (bit 7 ÷ 1) are copied into the Z register.

./..

EXAMPLE - THE SYSTEM IS COMPOSED OF (Fig.34)

- CPU
- ROM Module code Ø
- RAM Module code ØØ1ØØØ
- I/O Module code 56 (DEC)

LET: Z REG (ROM) = Ø1 ØØØ 1Ø1 Ø11
Z REG (RAM) = 1 ØØ1 ØØØ
ACC = 1Ø 11Ø ØØ1

A - IF X REG = ØØ ØØØ 1Ø1 after SZX execution
Z REG (ROM) = 1Ø1 1Ø11 ØØØ1

B - IF X REG = 11 ØØ1 ØØØ after SZX execution
Z REG (RAM) = Ø11 ØØØ1

./..

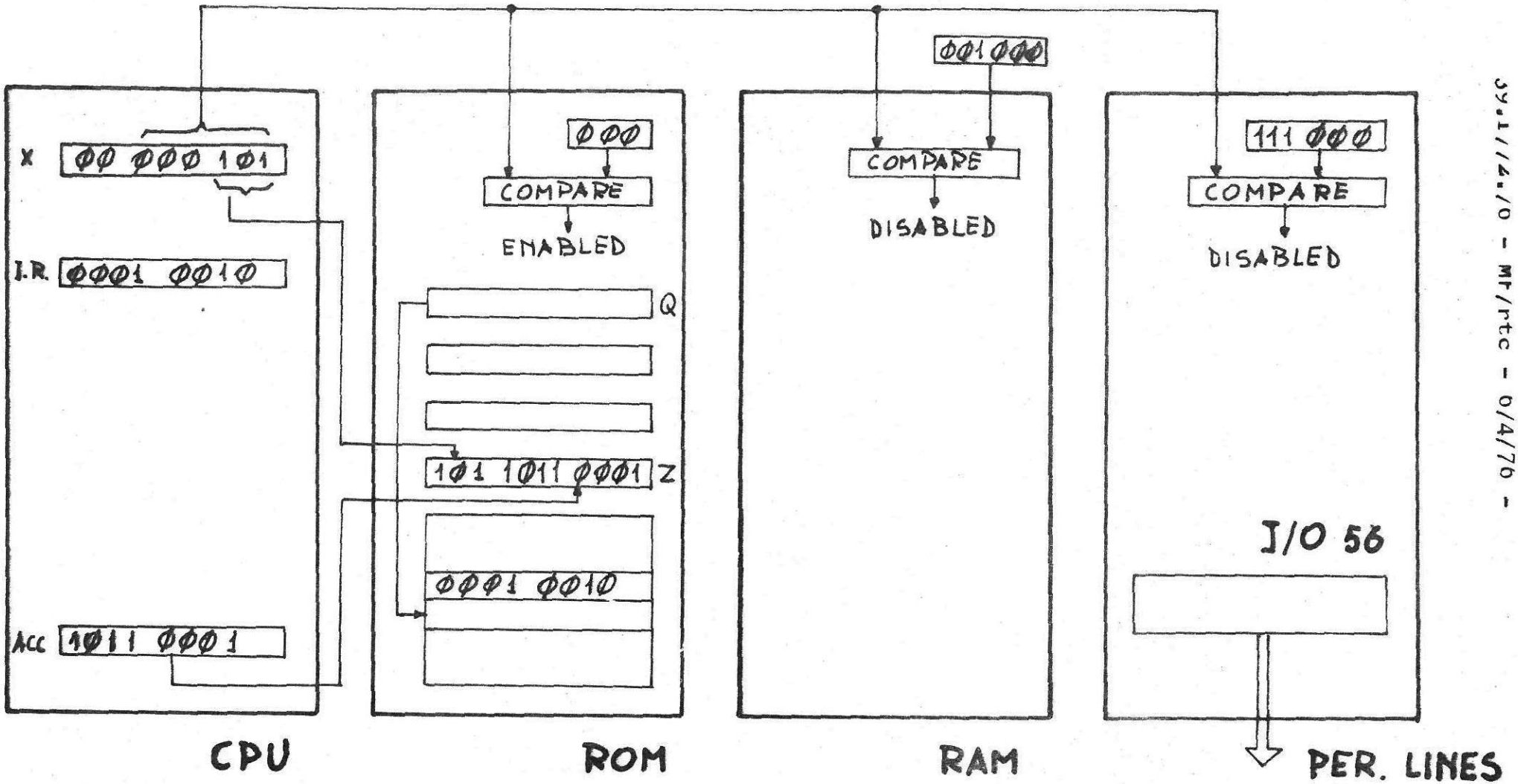


FIG 34- SZX EXECUTION

7 - STORE ACC INDIRECT INTO Z BY Y REGISTER

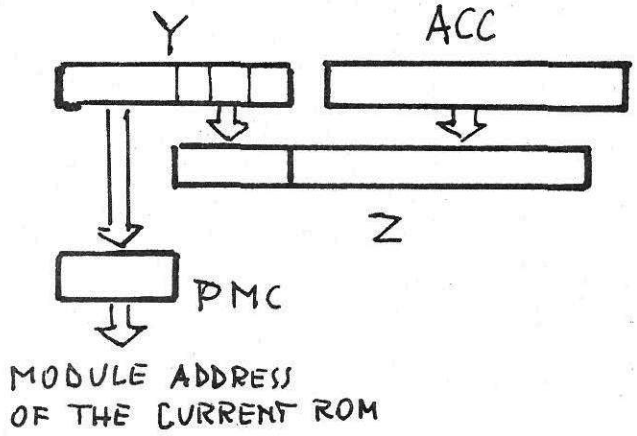
MNEMO

SZY - The CPU is enabling the module having the code = Y Register (bits 6 ÷ 1).
The module is the current program memory (Y = PMC).

- The Z Register is modified in the following way:

1 - Z Register (bits 11/10/9) = Y REG (bits 3/2/1)

2 - Z Register (bits 8 ÷ 1) = ACC.



OBJECT

13 ØØØ1 ØØ11

Operations and Signals: Similar to the ones of the SZX instruction.

./..

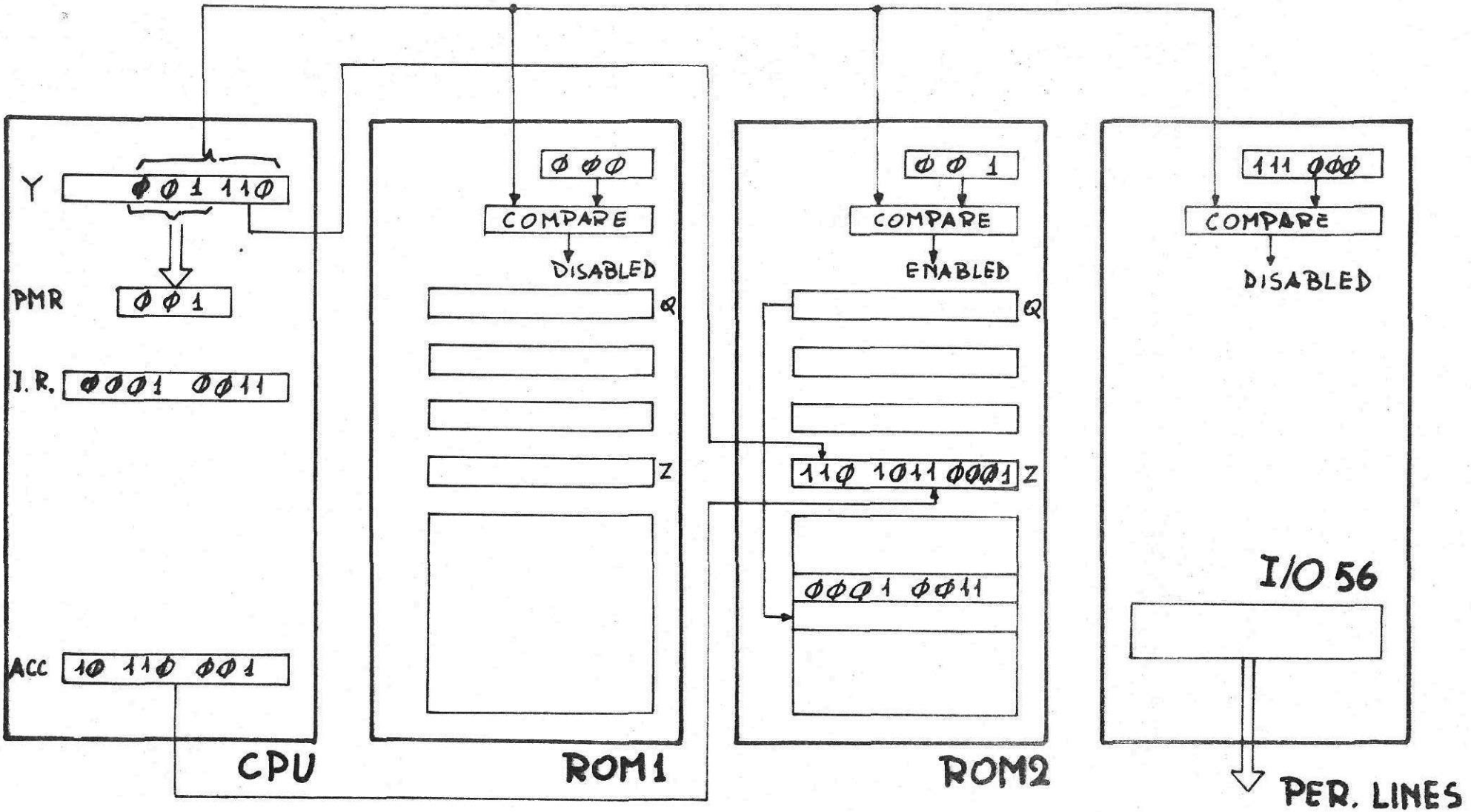


FIG 35 - SZY EXECUTION

SOME SIMPLE EXAMPLES

The following paragraphs of this manual are devoted to show some simple programs.
The examples which are discussed are:

1 - Program to jump from a block of 2K of ROM into another block of 2K ROM.

2 - Keyboard matrix scanning.

3 - Data input from a teletype.

4 - Decimal addition and substraction.

5 - Data transfer between the CPU and an external COS-MOS memory.

JUMP FROM 2K ROM INTO ANOTHER 2K ROM

It is apparent from the jump instruction format that the jump address may be specified by 11 bits.
This means that the address range for a jump instruction is 2048 (decimal).
For every program which is greater than 2K other ROM devices are needed. When a 2K program is in execution, to go and execute another 2K program, the following steps are to be performed:

1 - Prepare the module code of the next block into the three most significant bits of the X register.

2 - Prepare the start address of the new block into the accumulator and into the three least significant bits of the X register.

3 - Store the start address into the new program counter.

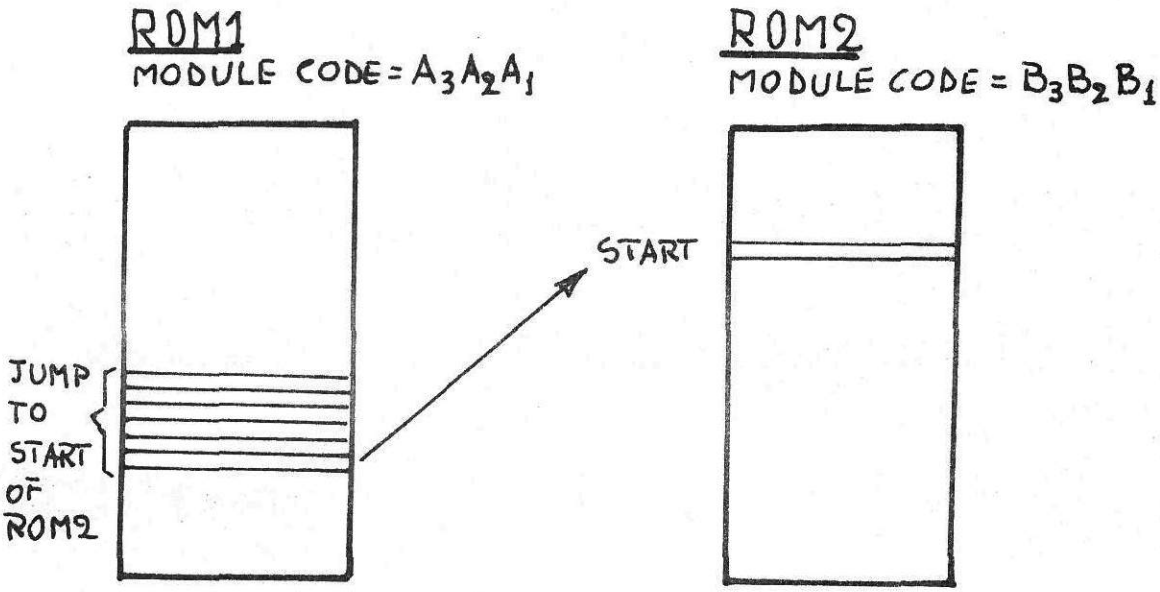
4 - Enable the new program memory module.

This is the program implementing the 2K jump :

```
      ---
      ---
      LAL   NEWM     LOAD NEW MODULE CODE INTO ACC
      SAX            AND STORE IT INTO X REG
      LAL   STADR    LOAD NEW START ADDRESS INTO ACC
      SQX            AND STORE IT INTO NEW Q REG
      LAX            LOAD NEW MODULE CODE INTO ACC
      SAY            AND STORE IT INTO Y AND PMC REG
      ---
      ---

NEWM  EQU / NEW CODE
STADR EQU / ADDRESS
```

The following figure 36 describes the problem and the symbols which have been used into the program:



$A_3A_2A_1$: 3 bit module code for ROM 1

$B_3B_2B_1$: 3 bit module code for ROM 2

START : Start address when the jump from ROM1 to ROM2 is executed. Let's assume that its binary representation is:

$$START = D_{11}\ D_{10}\ D_9\ D_8\ D_7\ \ldots\ldots\ D_1$$

NEW CODE : Is an exadecimal number having the following binary form:

$$B_3\ B_2\ B_1\ D_{11}\ D_{10}\ D_9$$

ADDRESS : Is an exadecimal number having the following binary form:

$$D_8\ D_7\ \ldots\ldots\ D_1$$

Fig. 36 - 2K JUMP

KEYBOARD MATRIX SCANNING (Fig.37)

A keyboard matrix 8x8 is connected from one side to the output I/O PORT having the module code 58 (decimal) and from the other side to the input I/O PORT the module code of which is 63.
The keyboard has to be scanned in order to recognize which key is pressed.

The problem has been simplied with the following assumptions:

1 - The key bouncing has not been taken into account.

2 - The first key detected by the scanning program is executed and therefore there is no control about two or more keys depressed at the same time.

The scanning process is performed in this way:

1 - A column mask is generated by the microcomputer. The mask is so done to enable one column per time. If the column to be enabled is the fifth one, the mask applied to the matrix will be:

ØØØ1 ØØØØ

2 - For every column mask the data coming out of the rows are read and analyzed.

3 - On the register R1Ø are stored the result of the scanning process. Every time the data coming out of the rows is zero, the register R1Ø is incremented by $1Ø_8$.
In case the row data is different than zero, the program looks for the first row which is not zero, starting from the least significant position (the bottom one).
For every row = Ø the register R1Ø is incremented.
When the condition row = 1 is found the scanning is ended and the register R1Ø content corresponds at that moment to the code of the key depressed.
On Fig.37 the situation for the key having the code 45_8 is shown.

4 - The program computes now the start address of the service routine for that key.
This is performed by the following steps:

A - The R1Ø content is added to the initial address N of the look up table.
Into this table for every key it is stored the start address of the corresponding service routine.

B - The result ($R_{1Ø}$ + N) is then copied into the program counter itself.
From that moment on the service routine for the key depressed is in execution.

The flow chart of the program is shown on Fig.38 and the coding is indicated on table N°3.

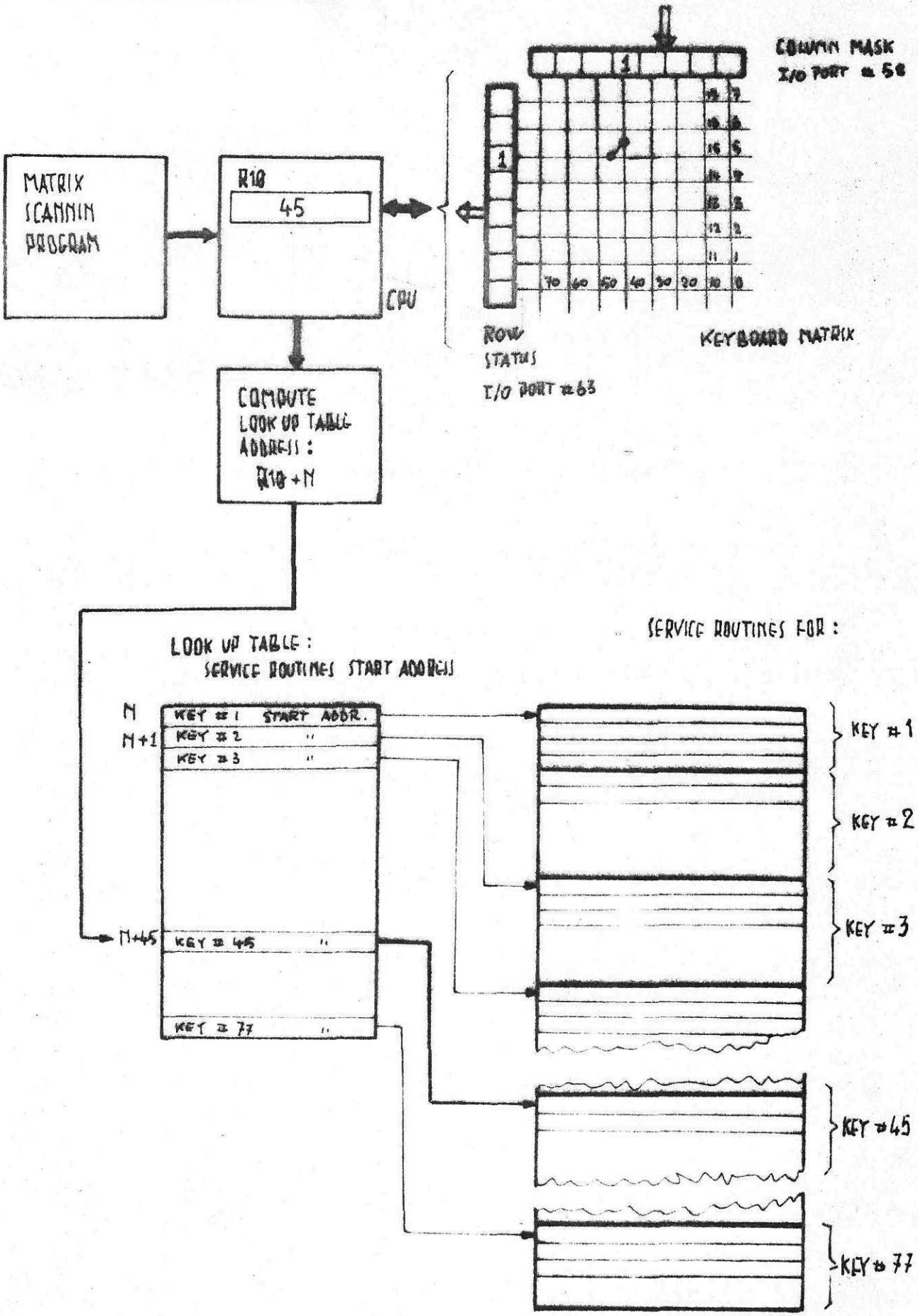


FIG 37 - KEYBOARD MATRIX AND LOOK-UP TABLE

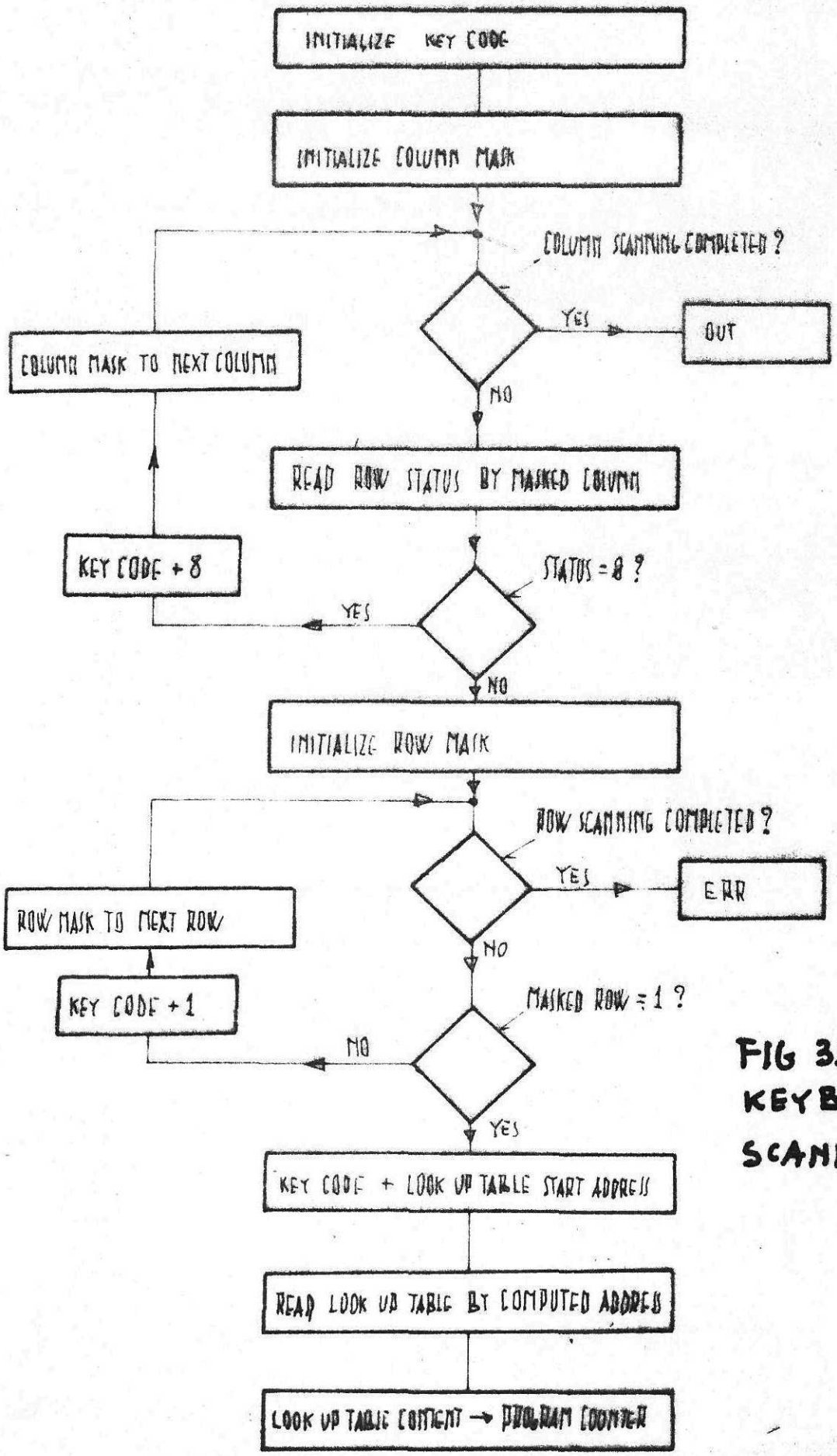


FIG 38 - KEYBOARD MATRIX SCANNING

TABLE 3 - 8 BY 8 KEYBOARD MATRIX SCANNING

```
            LAS     ZERO
            SAR     R1Ø         INITIALIZE KEY CODE
            LAL     ONE         INITIALIZE COLUMN MASK

AGAIN       SAV                 COLUMN MASK IN V REG
            LAV
            JAZ     OUT         SCANNING COMPLETED : NO KEY DOWN
            OUT     KBIN        COLUMN MASK TO KB MATRIX
            INP     KBOUT       READ MATRIX ROWS
            JAZ     NEXT        NO KEY DOWN FOR THIS COLUMN
            SAR     R11         KEY DOWN DETECTED. RESULT TO R11
            LAL     ONE         INITIALIZE ROW MASK

INLOOP      SAW                 SAVE ROW MASK TO W
            LAW
            JAZ     ERR         NO KEY DOWN : ERROR
            ANR     R11         ROW MASKING
            JAN     LOOK        FOUND KEY DOWN : TO LOOK UP COMPUTATION
            LAS     ONE         THIS KEY UP, THEN
            ADR     R1Ø         INCREMENT KEY CODE
            SAR     R1Ø         AND SAVE IT
            LAW                 GET ROW MASK AND
            ALS                 UPDATE IT THEN
            JMP     INLOOP      GO TO EXAMINE NEXT ROW

NEXT        LAS     TEN         NO KEY DOWN ON THIS COLUMN:
            ADR     R1Ø         INCREMENT BY 8 THE KEY CODE
            SAR     R1Ø         AND SAVE IT TO R1Ø
            LAV                 GET COLUMN MASK AND
            ALS                 UPDATE IT THEN
            JMP     AGAIN       GO TO EXAMINE NEXT COLUMN

LOOK        LAR     R1Ø         GET THE COMPUTED KEY CODE AND
            ADL     N           ADD THE LOOK UP TABLE START ADDRESS
            SZY                 DEPOSIT IT INTO RZ(Y) REGISTER THEN
            LIY                 READ THE LOOK UP TABLE CONTENT AND
            SQY                 CHANGE THE PROGRAM COUNTER BY THIS CONTENT

OUT         ---     ---
            ---     ---
            ---     ---

ERR         ---     ---
            ---     ---
            ---     ---
```

TABLE 3 - 8 BY 8 KEYBOARD MATRIX SCANNING (2)

LITERAL DEFINITIONS

```
ZERO    EQU   Ø
ONE     EQU   1
R1Ø     EQU   1Ø
R11     EQU   11
KBIN    EQU   58
KBOUT   EQU   63
TEN     EQU   8
N       EQU   (LOOK UP TABLE START ADDRESS)
```

INPUT FROM TELETYPE

The example which is analyzed is described by the Fig. 39 and 40. At a certain point of the main program there is a requirement to introduce a string of data into abuffer.
The buffer is 128 words long and the conditions for ending the input from the teletype and continue the main program are either one of these two:

1 - The number of characters fed into the buffer has exceeded 128.

2 - The last character introduced by the teletype is an 'ESCAPE' (equivalent to the octal code 33).

To input a character from the teletype the hardware needed is the following:

1 - An UART device. The basic input operation of this device is described on Fig.41.
Serial data are received by the device and presented at the output in parallel form.
The 'DATA READY' signal may be reset by the 'data ready reset' before presenting the serial data of the input.
After a serial data has been received and translated in parallel form the 'DATA READY' is set again to 1.

2 - An 8 bit input port.
This I/O port is directly connected to the 8 bit parallel output of the UART device.
The microcomputer will read the character from this port when the character is valid.
The I/O port for this operation is the one with the module code 57.

3 - A 1 bit input for 'BUFFER READY'.
This bit is connected to the 'DATA READY' signal of the UART device and is used by the microcomputer to know when the output character is valid.

4 - A 1 bit output to reset the 'data ready' signal of the UART at the beginning of any character readout cycle.

The read operation is performed by means of the subroutine 'TI' whose flow chart is shown on Fig.39B.
The operation goes through the following steps:

1 - A 'START' pulse is generated by outputting a '1' followed by a 'Ø' on bit 8 of the I/O port 56.
This pulse resets the 'DATA READY' signal of the UART.

2 - After that, the 'DATA READY' signal is continuously read and tested up to the moment when it is found equal to 1.
This means that a character from the TTY has been received by the UART and its code is now valid to the I/O port 57.

3 - The character code is read into the Accumulator and a return to the main program is executed.

The operations executed by the main program are:

1 - Prepare the new buffer address where to store the new character.
The characters are saved into the buffer starting from the Ø address and, after a character is copied into the buffer, the current address is incremented by adding 1 (ADL ONE) to it.
The last available address is:

Ø1 111 111

So, when the address 1Ø ØØØ ØØØ is detected, the character introduction is stopped and the program will continue from location ERR.

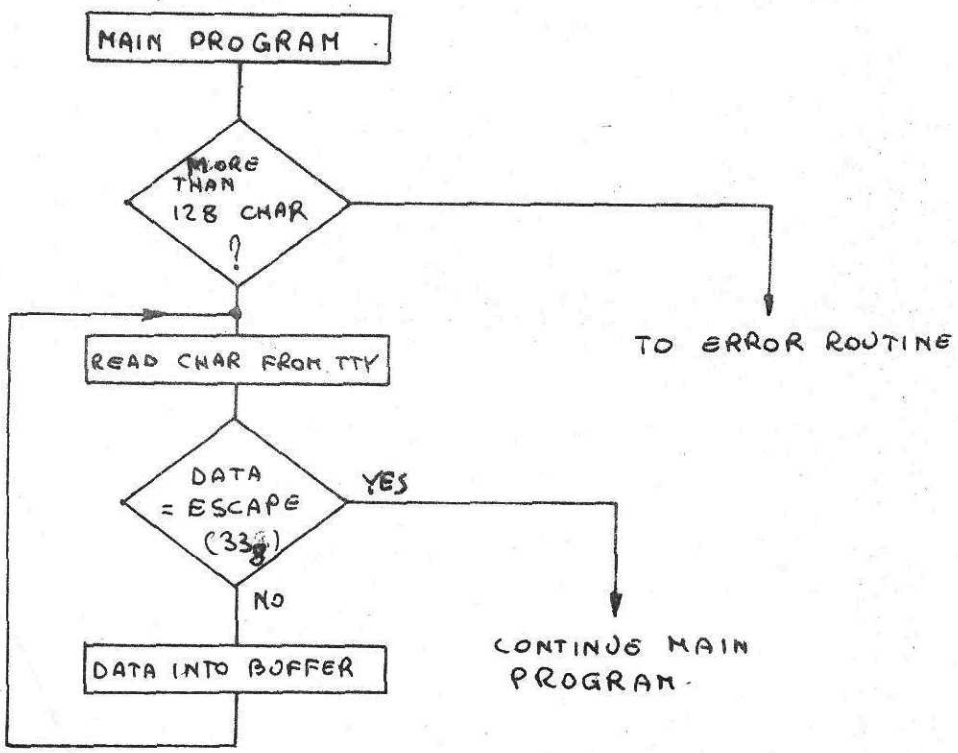
2 - Check if the number of characters,which have been introduced,is greater than 128.
If the answer is positive, the data input is stopped.

3 - If the answer is negative, a new character is accepted from the teletype by a 'read' operation as described before.

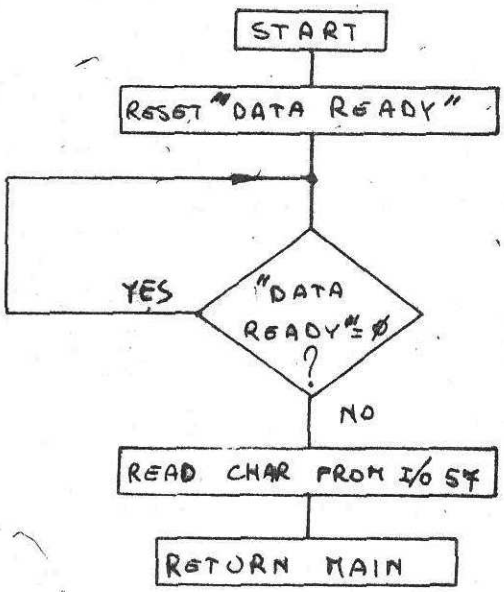
4 - The character is saved into the buffer area and then is compared against the 'ESCAPE' code.

5 - If the character is equal to the ESCAPE code, the input from teletype is terminated and the main program will continue from the 'OUT' location, otherwise the program comes back to the point 1.

The program listing is shown in table 4 and 5.



DATA FROM TTY INTO A BUFFER - FIG 39 A



SUBROUTINE TO READ CHAR FROM TTY.

FIG 39 B

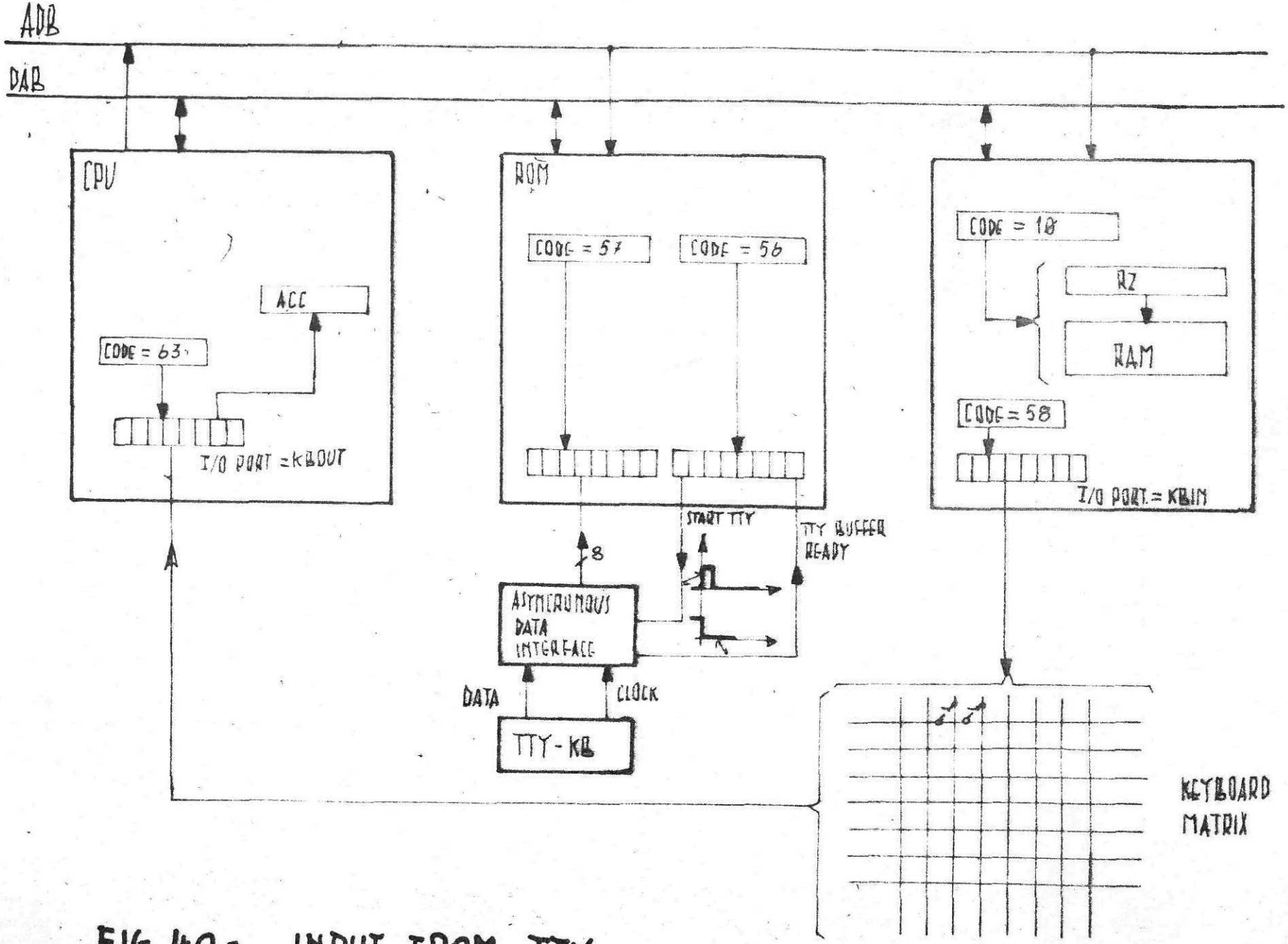
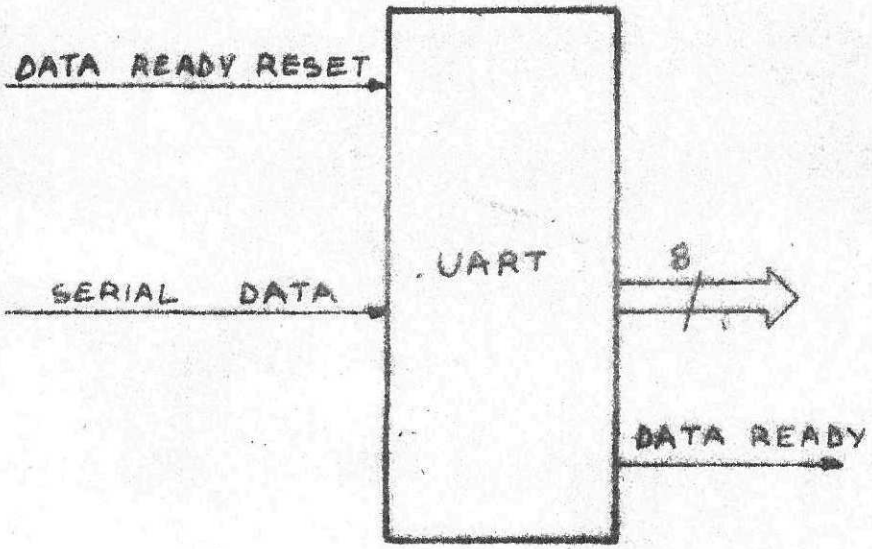


FIG 40 - INPUT FROM TTY

Mod. 3030824I - 20000 - Ced-Sipi S. p. A.



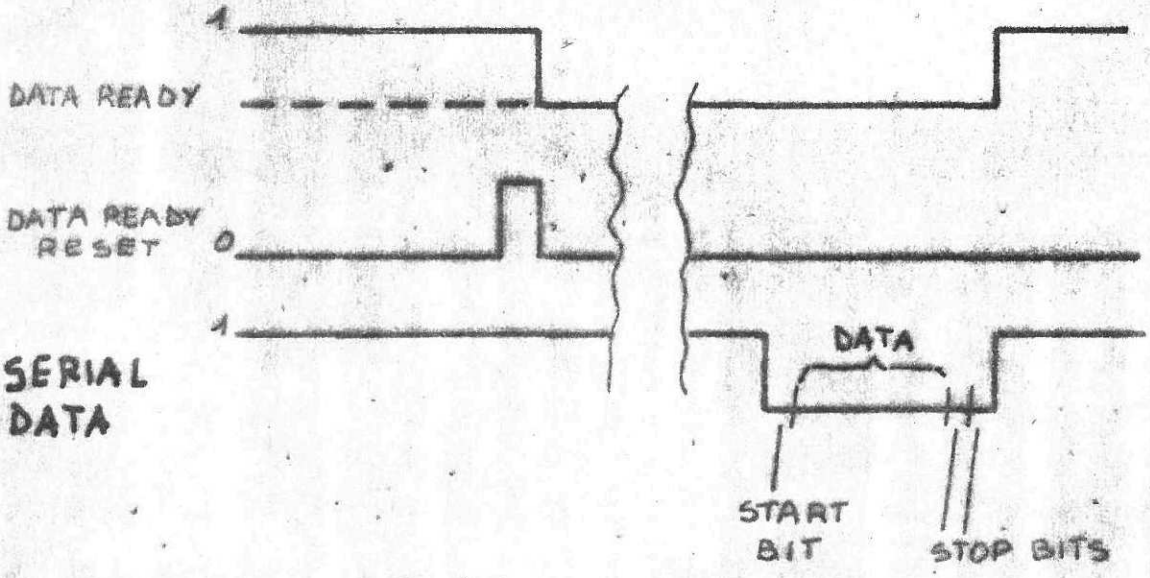


FIG 41 - UART OPERATION FOR DATA INPUT

TABLE 4 - INPUT FROM TTY

A - MAIN PROGRAM

```
        LAL   RØ    }
        SAX         }   RAM CODE → X

        LAL   MAX       INITIALIZE START ADDRESS
BACK    ADL   ONE       INCREMENT RAM ADDRESS
        SAV             SAVE RAM ADDRESS → V

        JAP   AHEAD     CHECK IF NUMBER OF CHARS. > 128
        JMP   ERR       MORE THAN 128 WORDS
AHEAD   SZX             STORE NEW RAM ADDRESS → R2 (RØ)
        JSB   TI        TTY INPUT
        SIX             SAVE ACC  → RAM
        EOL   MAX       COMPLEMENT ACC
        ADL   ESC       COMPARE 'ESC' = 33₈
        JAZ   OUT       END OF INPUT IF CHAR = ESC
        LAV             RAM ADDRESS → ACC
        JMP   BACK      LOOP AGAIN

OUT     ____    }
        ____    }   MAIN CONTINUE
        ____    }

ERR     ____    }
        ____    }   ERROR ROUTINE
        ____    }
```

LITERAL DEFINITION

```
RØ      EQU     1Ø
MAX     EQU     377
ONE     EQU     1
ESC     EQU     34
```

TABLE 5 - INPUT FROM TTY

B - TTY INPUT SUBROUTINE

```
TI      LAL     BY
        OUT     ZERO        OUTPUT START → 1
        LAL     ZERO
        OUT     ZERO        OUTPUT START → 0, BØ→Ø

LOOP    INP     ZERO        READ I/O PORT Ø
        ANL     B1          MASK BIT Ø
        JAZ     LOOP        JUMP BACK IF BUFFER NOT READY
        INP     B1          READ I/O PORT 1 (TTY)
        RET                 RETURN
```

LITERAL DEFINITION

```
ZERO    EQU     Ø
BY      EQU     2ØØ
B1      EQU     1
```

DECIMAL ADDITION AND SUBTRACTION

The example refers to a decimal addition or substration of 2 numbers, each one with 14 digits.
The 2 numbers are represented in a BCD notation and are stored into the CPU RAM page Ø and 2.
The result of the operation is stored into page 2.
The format is shown on Fig.42, while the program is listed in Table 6.
Let's examine step by step the 'decimal addition' program:

1 -		ARS	The 'right shift' instructions is used to set the carry to Ø.
2 -		LSS 6	Load S register by the immediate = 6. This instruction prepares the row pointer to the row when the least significant digits are stored.
3 -	ALOOP	LTS Ø	Load T register by the immediate = Ø. This instruction prepares the page pointer to the page where the first number is stored.
4 -		LAR 12	Load into the accumulator the register indirectly addressed by S, T. The first time this instructions is executed the accumulator will be loaded by the first and second digit of the number located on page Ø. The successive times the instruction is executed, the accessed row on the same page will be different due to the indexing performed on the S register by the instruction at position 7.
5 -		ADL /66	The Accumulator is added by the exadecimal number 66 and the result is stored into the Accumulator. This prepares the 2 digits on the accumulator for the next DAR instruction.

6 -	LTS	2	Now the page pointer is moved to page 2 by loading the immediate 2 into the T register.
7 -	DAR	13	The accumulator is added to the 2 digit number addressed by S, T. By performing this operation the carry is taken into account and is also changed by the result itself. The first time this instruction is executed the accumulator is added to the first and second digit on page 2. The successive times the accessed row on the same page 2 will be different due to the indexing on the S register performed by this instruction. The decimalcorrection, as explained on the instruction set, is applied to the result, which then will be saved into the register addressed by S and T. At the end of this instruction the S register is decremented by 1 in order to point to the next two digits to be added.
8 -	JSD	ALOOP	A jump to location 'ALOOP' is done if the S register content is not 7. The condition S = 7 is reached when the full page has been scanned. In this case, the program will continue by the next location.

The decimal subtraction is performed in a similar way.
Please note the following things:

1) The carry flip-flop must be set to 1 at the beginning.
2) The subtractor must be on the page referenced as the first.
3) The subtractor is complemented but not corrected by 66 exadecimal.

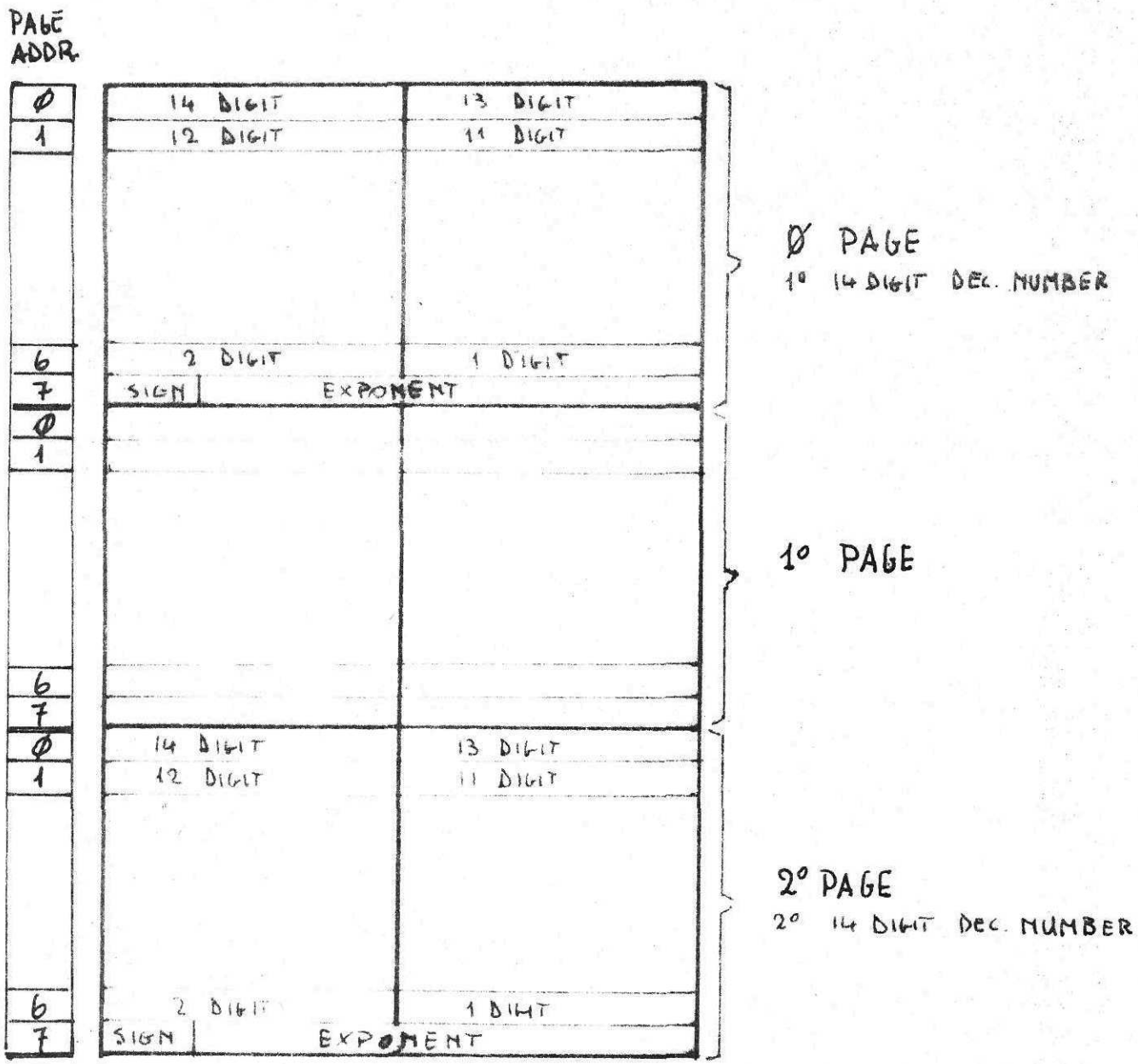


FIG 42 - DATA STRUCTURE FOR TWO 14 DIGIT DEC. NUMBERS LOCATED ON PAGE Ø AND 2

TABLE 6A - DECIMAL ADDITION

```
        ARS          Ø → CARRY
        LSS  6       6 → S
ALOOP   LTS  Ø       Ø → T : pointer to first field
        LAR  12      Register address by (S,T) → ACC
        ADL  /66     ACC + (66)16 → ACC
        LTS  2       2 → T : pointer to second field
        DAR  13      (ACC) + ((S,T)) + CARRY → (S,T)
                     STORE   CARRY
                     (S)-1 → S
        SSP  ALOOP   Jump to aloop if S ≠ 7
```

TABLE 6B - DECIMAL SUBSTRACTION

```
        ALS          1 → CARRY
        LSS  6       6 → S
        LTS  Ø       Ø → T : pointer to first field
        LAR  12      ((S,T)) → ACC
        EOL  255     Complement ACC
        LTS  2       2 → T : pointer to second field
        DAR  13      (ACC) + ((S,T)) + CARRY → (S,T)
                     STORE   CARRY
                     Decrement S
        SSP  SLOOP   Jump to sloop if S ≠ 7
```

===

===

DATA TRANSFER FROM CPU INTO COS-MOS MEMORY

AND VICEVERSA

The example describes an application where some data have to be saved into an external COS-MOS memory backed-up by a battery during power down periods.
In this way, the data stored into the COS-MOS memory are not lost in case of power failure.
The hardware of the system is shown in Fig.43A:

1 - Two COS-MOS memories 256 words by 4 bits are used in order to have a memory of 256 by 8 bits.
2 - The address lines AØ to A7 of the two devices are connected in parallel and to the I/O port 56.
3 - Data inputs and outputs are wired-or on every device and connected to the I/O port 57.
4 - Four bits of the I/O port 58 are for control functions on the memories (Read/Write, Device Enable, Output Disable).
The other four bits of the same I/O port are as spare, and on the program are always set to zero.
5 - The COS-MOS devices are connected to the +5V power supply through a diode.
At the power supply pin the battery back-up circuit is also connected. By this circuit a 3 V battery is continuously kept under charge. In case of +5V power failure the data retention on the memories is guaranteed by the battery back-up circuit which is able to supply 2,3 V to the devices.
6 - Every I/O bit of the M38 microcomputer is connected to the COS-MOS device by the circuit shown in Fig.43B.

The data transfer programs are described on Tables 7A and 7B, and by the flow-charts of Fig.44.
For a better understanding of the program please note the following things:

1 - Table 7A and Fig.44A describe the write operation into the COS-MOS memory, while Table 7B and Fig.44B describe the 'read' operation from the COS-MOS memory.

2 - In both cases the block which is transfered in or out is an 8 word block, which corresponds to a RAM page of the CPU.

The page is scanned from the last row (row 7) up to the first row on either read or write operation.
Similarly, the COS-MOS memory is scanned always from the bottom up.

3 - The control signals for the COS-MOS memory are:

- RW (Read/Write). It has to be Ø for WRITE operation.
- $\overline{CE1}$, CE2 (chip enable 1 and 2). To enable the devices they must be:

  $\overline{CE1} = Ø$

  $CE2 = 1$

  If the state for these two signals is different, the device is disabled.
  In particular, when CE2 = Ø, the device is disabled and a minimum stand-by current is drawn by the device itself.
- OD (output disable). It must be set to 1 for an input (or write) operation.

They are normally set into the quiescent state when no read or write operation is performed.
The quiescent or 'NO OPERATION' state is as follows:

Bit Ø = $\overline{RW}$ = 1
Bit 1 = $\overline{CE1}$ = 1
Bit 2 = CE2 = Ø
Bit 3 = OD = Ø

To set this state on memories an immediate = 3 must be sent out from the I/O port 58.
The read state and the write state are identified by the following configuration:

A - READ : Bit Ø = $\overline{RW}$ = 1
Bit 1 = $\overline{CE1}$ = Ø
Bit 2 = CE2 = 1
Bit 3 = OD = Ø

This means that the immediate to be transfered into the I/O port 58 is 5.

8 - WRITE : Bit Ø = $\overline{RW}$ = Ø
Bit 1 = $\overline{CE1}$ = Ø
Bit 2 = CE2 = 1
Bit 3 = OD = 1

This means that the immediate to be transfer ed into the I/O port 58 is 12 (decimal).

- The register 8 of the CPU is reserved to hold the current address of the in/out data of the COS-MOS memory.
This address has to be prepared in advance before the beginning of the transfer program.
During the trasfer, the address is incremented every time a data is transfered.

TABLE 7A - WRITE OPERATION PROGRAM

```
         ---
         ---
         ---
         LSS   7    Indirect row address set to 7
         LTS   2    Indirect page address set to 2

LOOPW    LAR   8    Get the external memory address from page 8
         OUT   Ø    And output it on I/O port 56
         LAR  13    Get the data, decrement row address
         OUT   1    And output it on I/O port 57
         LAL  12    Prepare the control signals for 'WRITE'
         OUT   2    And output to I/O 58: WRITE executed now
         LAL   3    Prepare the control signals 'NO OPERATION'
         OUT   2    And output to I/O 58 to disable memory
         DER   8    Decrement external memory address
         LAR   8    And check if the address is valid:
         JAP   NEXTW - Address within 128: valid
         JMP   ERR   - Address more than 128: not valid

NEXTW    JSD   LOOPW  Check if page is fully scanned

         ---
         ---
         ---

ERR      ---
         ---
         ---
```

TABLE 7B - READ OPERATION PROGRAM

```
         ---
         ---
         ---
         LSS   7       Indirect row address set to 7
         LPS   2       Indirect page address set to 2

LOOPR    LAR   8       Get the external memory address from reg. 8
         OUT   Ø       And output it on I/O port 56
         LAL   5       Prepare the control signals for 'READ'
         OUT   2       And output it on I/O port 58
         INP   1       READ executed
         SAR   13      Store data, decrement row address
         LAL   3       Prepare the control signal 'NO OPERATION'
         OUT   2       And output to I/O port to disable memory
         DER   8       Decrement the external memory address
         LAR   8       And check if the address is valid.
         JAP   NEXTR       - Address within 128: valid
         JMP   ERR         - Adddress more than 128: not valid
NEXTR    JSD   LOOPR   Check if page is fully scanned.

         ---
         ---
         ---

ERR      ---
         ---
         ---
```

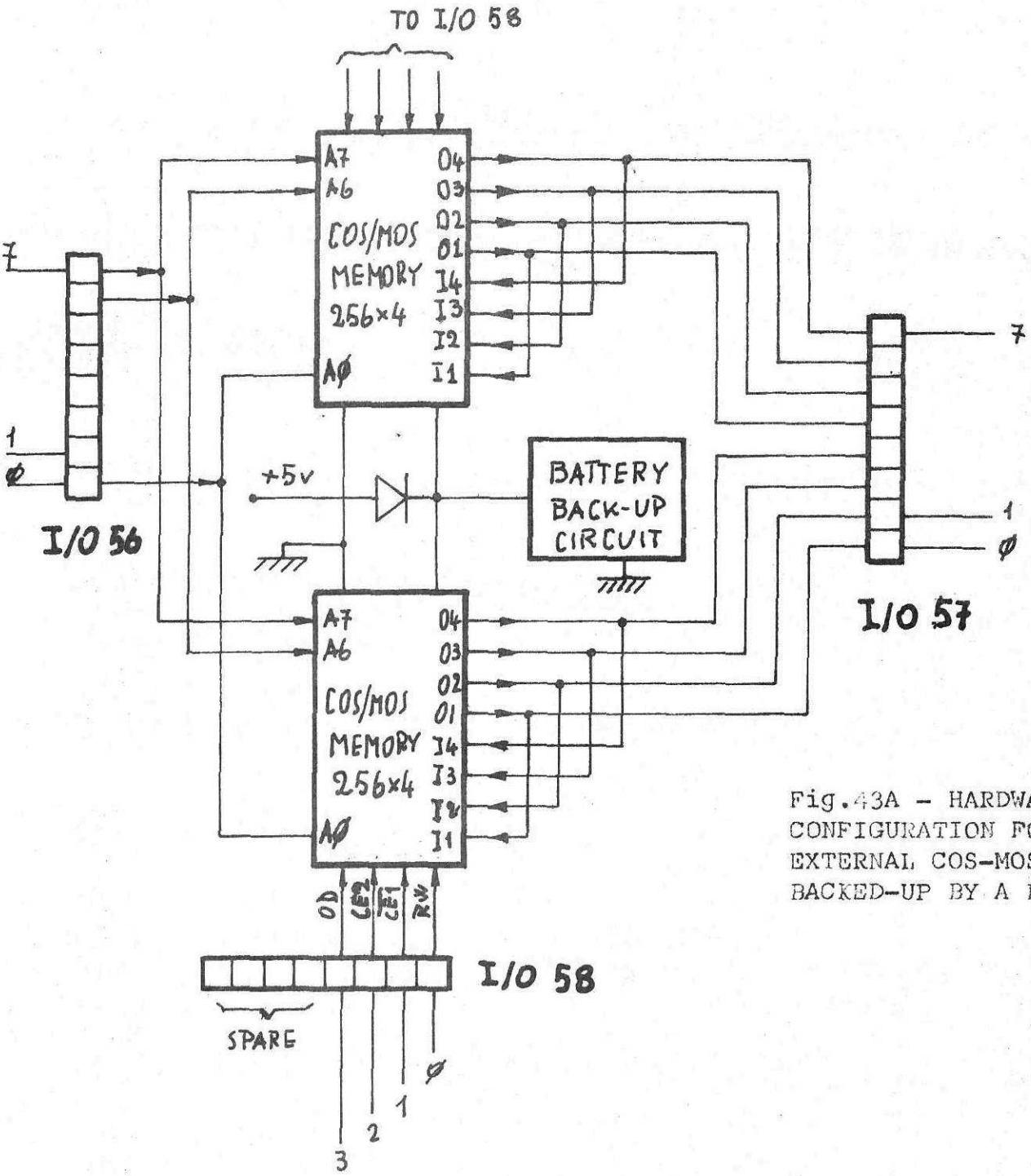


Fig.43A - HARDWARE CONFIGURATION FOR AN EXTERNAL COS-MOS MEM. BACKED-UP BY A BATTERY

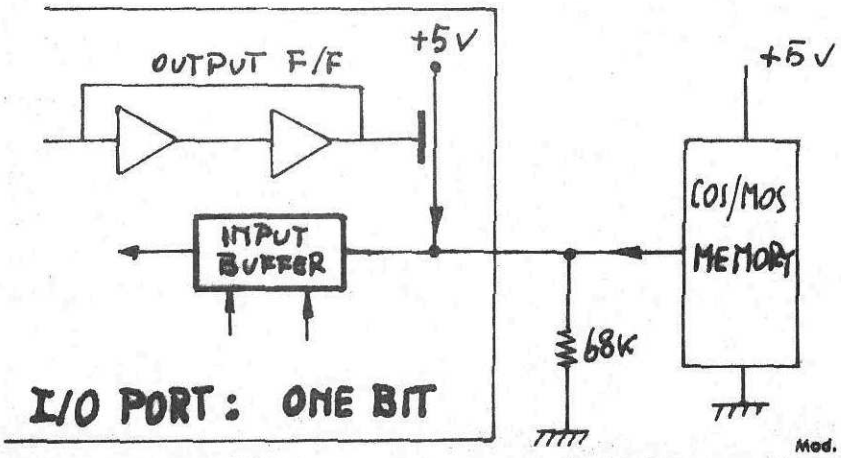


Fig.43B - TYPICAL I/O BIT INTERCONNECTION

MAIN PROGRAM

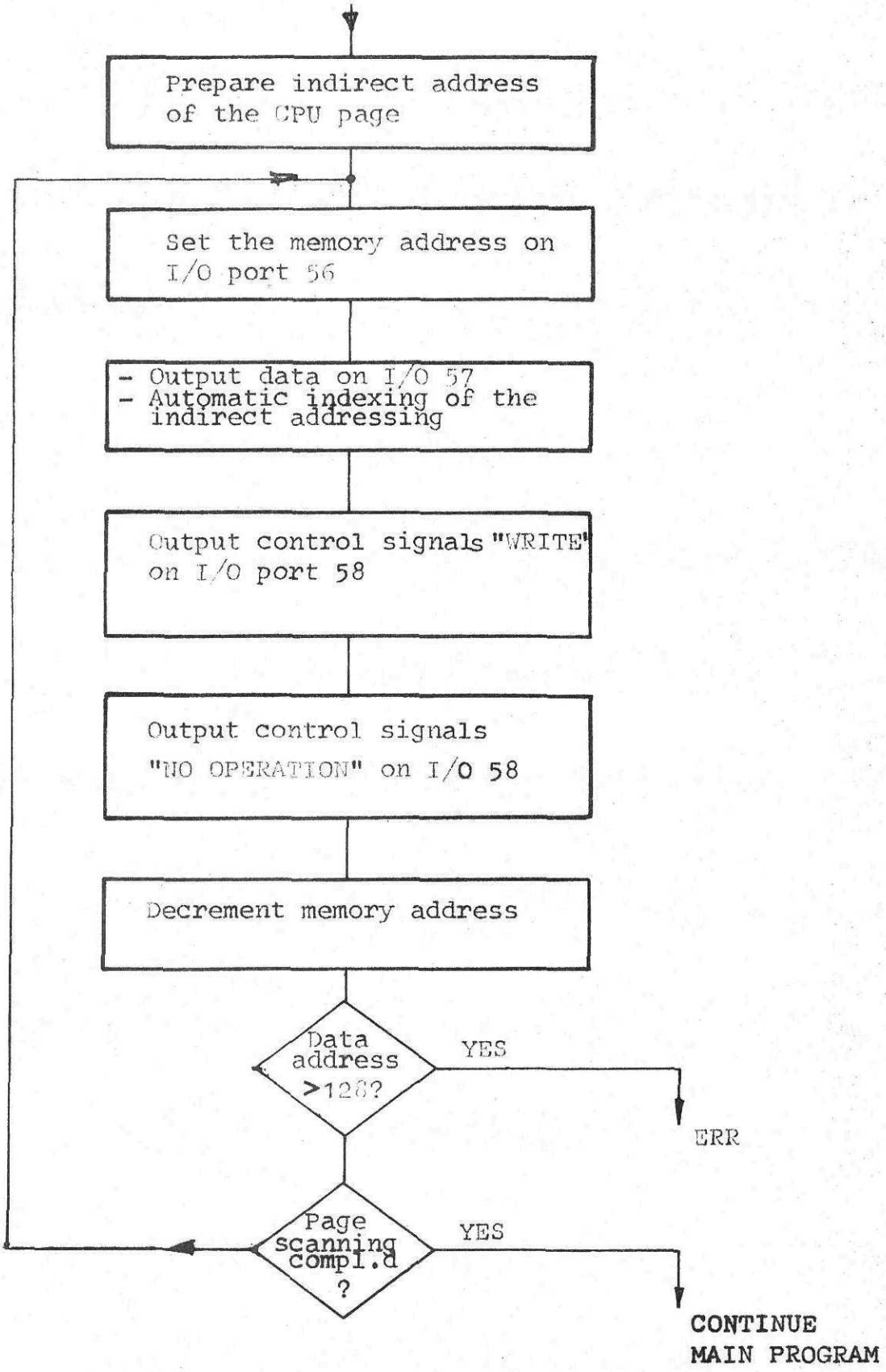


Fig.44A - FLOW-CHART: WRITE ON COS-MOS MEMORY

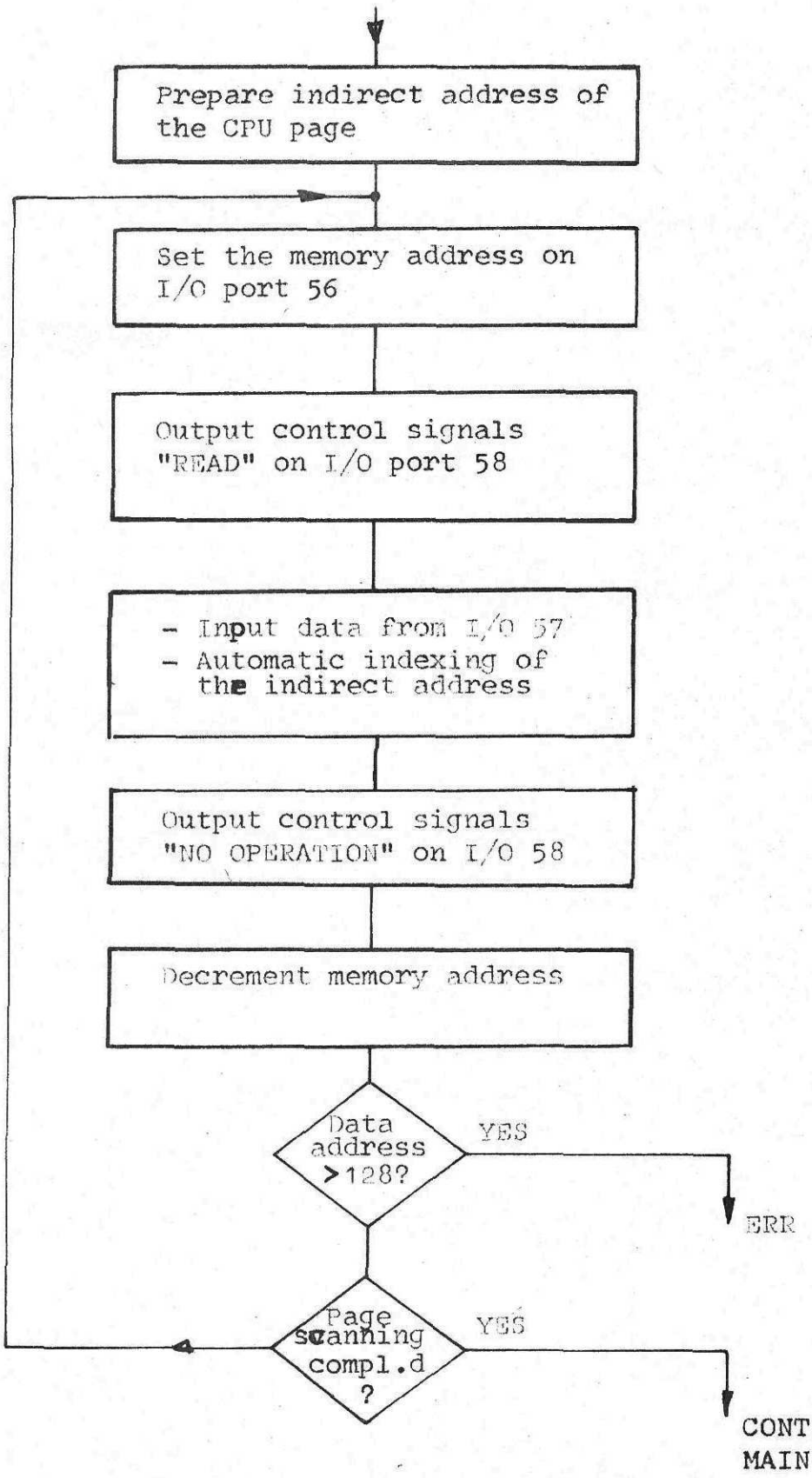


Fig.44B - FLOW-CHART: READ FROM COS-MOS MEMORY

M38 SOFTWARE SUPPORT

CROSS ASSEMBLER SOFTWARE PACKAGE

8.0 - General Information

The M38 cross assembler is a computer program written in ANSI standard FORTRAN IV language.
This program provides a translation into a machine language of M38 programs written by using simbolic codes as specified in the paragraph (8.6).

Input data can be fed into a computer either by cards or paper tape or directly by a teletype keyboard.
Maximum number of lines per program to be assembled are 16.000
Symbolic addresses through labels can be used: the maximum number of labels is 1000.

Numeric terms may be decimal or exadecimal. In the second case the term must be preceeded by a /.

The output to be produced by the assembler is controlled by a special command.
The output may be:

1 - A listing containing symbolic codes, machine codes and error flags.

2 - A paper tape or a magnetic tape or a set of cards containing machine codes on special format to be used either by the hardware simulator or by the PROM programming or as an input for the ROM masking production.

3 - A file to be used by the software simulator.

8.1 - Statement characteristics

The fields of the symbolic statement appear in the following order:

- Label
- Operation code
- Operand
- Comments

One or more spaces separate the fields of the statement.
Some of those fields are optional or are not required by some types of instructions.
The characters that may appear in the label, op code, operand field, are:

Alphanumeric
* asterisk
+ (plus)
- (minus)
/ (slash)
(space)

Any character may appear in the comment field.

8.2 - Label field

The label field identifies the statement and may be used as a reference point by other statements in the program.
The fields starts in position one.
A statement with a space in position one is considered unlabeled.
A label must be symbolic. It may have one to five alphanumeric characters.
The first character must be alphabetic.

8.3 - Operation code field

The operation code defines an operation to be performed by the computer or the assembler.
The machine operation codes must be specified by the mnemonic codes which are described in the paragraph 8.6.
The pseudo-operation codes which are used by the assembler are the following:

1 - EQU

The format is:

Label EQU v

During the assembly phase any reference to the label is assigned the value v.
The operand v may be a decimal or exadecimal number or a name defined by another EQU.
EQU may appear in any place on the program.

2 - ORG

The format is

- ORG v

It is used to define the absolute address of a program or the absolute address of subsequent sections of the program.
A zero address is assigned the first instruction of a program not having any ORG statement at the beginning.
The operand v may be any decimal or exadecimal number or a name defined by an EQU.

3 - DC

The format is:

- DC v

This pseudo-instruction stores a constant on the object program at the current address.
The constant has the value v specified in the operand field and it may be any decimal or exadecimal number or a name defined by EQU.

4 - SPACE

The format is:

- SPACE -

It is used to skip one line on the source program listing.

5 - EJECT

The format is :

- EJECT -

It is used to skip to the top of the page of the source program listing.

6 - END

The format is:

- END -

It terminates the source program input.

8.4 - Operand field

The meaning and the format of the operand field are related to the operation code in the source statement.
Some instructions do not require any operand: in this case the operand field is treated as a comment field.

The operand may be:

1 - A data to be loaded into the registers.
The operand may be any decimal or exadecimal number or a name defined by EQU.

2 - An address for a conditional or unconditional jump.

The operand may be:

- A name of a label specified on the program.
The address will be that of the instruction labeled by the same name.
- A name ± a decimal number.
- An asterisk ± a number.
The asterisk indicates the current address.

3 - An address of a CPU register.
The operand may be:

- A decimal number from 0 to 11. In this case it will be addressed one of the first 12 registers on the CPU.
- An asterisk.
In this case a CPU register will be addressed through (S, T) registers.
The 8 register will not be changed.

- An asterisk followed by + or -.
  In this case a CPU register will be addressed through (S, T) registers and the S register will be incremented or decremented.

8.5 - Comments field

The comment field may be used by the user for documentation purposes.
The notes on the comment field will be listed with the source language coding on the assembler printout.
The comments field terminates on position 72.
A full line of comments may be included by using an asterisk as a first character.

Mnemonic code	Operand	Internal code	Function
LAR	RR	8Ø ÷ 8E	(RR) → A
SAR	RR	9Ø ÷ 9E	(A) → RR
ADR	RR	AØ ÷ AE	(A) + (RR) → A (binary)
ANR	RR	BØ ÷ BE	(A) ∧ (RR) → A
EOR	RR	CØ ÷ CE	(A) ⊕ (RR) → A
DER	RR	DØ ÷ DE	(RR) - 1 → RR
DAR	RR	EØ ÷ EE	(A) + (RR) + (C) → RR, C (decimal)
LAS	I1	FØ ÷ FF	I1 1 ... 4 → A
LSS	I1	28 ÷ 2F	I1 1 ... 3 → S
LTS	I1	38 ÷ 3F	I1 1 ... 3 → T
INP	I1	2Ø ÷ 27	(Port number 56 + I1) → A
OUT	I1	3Ø ÷ 37	(A) → Port number 56 + I1
LAL	I2	Ø4 (ØØ ÷ FF)	I2 → A
ANL	I2	Ø5 (ØØ ÷ FF)	I2 ∧ (A) → A
EOL	I2	ØC (ØØ ÷ FF)	I2 ⊕ (A) → A
ORL	I2	ØD (ØØ ÷ FF)	I2 + (A) → A
ADL	I2	ØE (ØØ ÷ FF)	I2 + (A) → A (binary)
CML	I2	ØF (ØØ ÷ FF)	I2 + (A) → C, Z
JMP	LAB	4Ø ÷ 47 (ØØ ÷ FF)	-
JAZ	LAB	48 ÷ 4F (ØØ ÷ FF)	(A) = Ø
JAN	LAB	5Ø ÷ 57 (ØØ ÷ FF)	(A) ≠ Ø
JAP	LAB	58 ÷ 5F (ØØ ÷ FF)	$(A)_8$ = Ø
JSD	LAB	6Ø ÷ 67 (ØØ ÷ FF)	(S) ≠ 7
JCN	LAB	68 ÷ 6F (ØØ ÷ FF)	(C) = 1

I1 1...3	Q9...11
I2	Q1... 8

Mnemonic code	Operand	Internal code	Function
JCZ	LAB	7Ø ÷ 77 (ØØ ÷ FF)	(C) = Ø
JSB	LAB	78 ÷ 7F (ØØ ÷ FF)	Q + 1 → Q RB → RZ, RA → RB, Q → RA I1 1 ... 3 → Q 9 ... 11 I2 → Q1 ... 8
RET		ØØ	RA → Q, RB → RA, RZ → RB (Y4...6)
ALS		1C	(A) 1÷7 → A 2÷8, Ø → A1, 1 → C
ARS		1D	(A) 2÷8 → A 1÷7, Ø → A8, 0 → C
ALF		1E	(A) 1÷4 → A 5÷8, Ø → A 1÷4
ARF		1F	(A) 5÷8 → A 1÷4, Ø → A 5÷8
SAT		Ø1	(A) 1÷3 → T
SST		Ø3	(A) 1÷6 → S, T
LAV		Ø8	(V) → A
LAW		Ø9	(W) → A
LAX		ØA	(X) → A
LAY		ØB	(Y) → A
SAV		18	(A) → V
SAW		19	(A) → W
SAX		1A	(A) → X
SAY		1B	(A) → Y
SIX		Ø2	RAM: A → (Z1...7) (X1...6) I/O: A → I/O (X1...6)
LIX		Ø6	RAM: ((Z1...7))(X/Y1...6) → A
LIY		Ø7	ROM: ((Z1...11)) (X/Y4...6) → A I/O: I/O (X/Y1...6) → A
SZX		12	RAM: A → Z1...7 (X/Y1...6)
SZY		13	ROM: A → Z/Q1...8 (X/Y4...6)
SQX		16	X/Y1...3 Z/Q9...11
SQY		17	(X/Y4...6)

8.6 - MNEMONIC CODES (CONTINUE)

ITEMS DESCRIPTION

A	Accumulator
A_8	Accumulator bit 8
C	Carry
I1	Least significant bits of instruction
I2	Second byte of instruction
Q	Q register of ROM
RA	A register of ROM
RB	B register of ROM
RZ	Z register of ROM
RR	Ram register of RSE
S	Row address register
T	Page address register
V, W, X, Y	RAM register V,W,X,Y
Z	Zero flag

8.7 - ERROR FLAGS

This is the list of the error flags which are generated by the assembler, when an error is detected during the assembly phase:

L	Illegal character on a label
O	Illegal character on an operator
D	Duplicated label
W	Error by using the 'ORG' pseudo-instruction
X	Error by using the 'EQU' pseudo-instruction
Y	Error on register reference instruction
B, C, S	Missing or duplicated label
H	Error on a constant
F	Error on long immediate instructions or on I/O instructions

They will appear at the beginning of the line where the error has been detected.

8.8 - EXAMPLES

In this section the following examples of the M38 Assembler are shown:

1 - Input format for the Assembler
2 - AssemblerListing
3 - Assembler Symbol Table
4 - Listing with Error Flags

The program which the example refers to is the one that has been discussed in the Section 'Some simple examples' under the paragraph: 'Data input from teletype'.

And now just a few comments about these examples:

1 - Input format for the Assembler:

It shows the data input as it should be presented to the Assembler.
The program may be punched on a deck of cards or on a paper tape or may be input directly through the teletype.

2 - Assembler listing and symbol table

It shows the Assembler printout which is one of the results of the Assembler process.

The pieces of information appearing on the Assembler listing are the following:

A - The first column is a progressive decimal number to identify any program line.

B - The second column (titled 'DEC') is the program address expressed by a decimal notation.

C - The third column (titled 'OCTL') is again the program address expressed by octal notation. Please note that the pseudo-instructions do not generate any program location.

D - On the two columns titled 'OCT' and 'EX' the machine code corresponding to the symbolic instruction code has been printed out.

E - The following part of the listing is showing the input symbolic program.

Please note the effect on the listing of the pseudo-instruction 'SPACE', 'EJECT'.

On the symbol table every label of the program is printed out in alphabetic order together with the corresponding address expressed in exadecimal notation.

3 - Listing with error flags

The example shows part of the input symbolic program where some errors are included and the listing produced by the Assembler. Please note the flags printed out after the first column.

INPUT FORMAT

FOR THE ASSEMBLER

```
*
*          INPUT FROM TELETYPE
*
RAMCO  EQU 10          RAM CODE
ONES   EQU 255         ALL ONES
ESC    EQU 27
PORT5  EQU 5           DATA IN PORT
PORT6  EQU 6           CONTROL PORT
*
*
       LAL RAMCO       RAM CODE
       SAX             SET RAM CODE
       LAL ONES        RAM POINTER
       SPACE
BACK
       ADL 1           INCREMENT
*                      RAM POINTER
       SAV             SAVE
*                      RAM POINTER
       JAP AHEAD       JUMP TO READ
*                      CHARACTER FROM TTY
       JMP ERR         JUMP TO ERROR
*                      MORE THEN 128 CHARACTERS
       SPACE
AHEAD
       SZX             SET Z-RAM-REGISTER
       JSB TI          JUMP TO SUB.TI
       SIX             STORE DATA INTO RAM
       EOL ONES        COMPL. ACC.
       ADL 1
       ADL ESC         TEST FOR ESCAPE
       JAZ OUT         END OF DATA
       LAV             RESUMES RAM POINTER
       JMP BACK
       EJECT
*
*          SUBROUTINE INPUT FROM TELETYPE
*          THE INPUT CHARACTER IS
*          IN THE ACCUMULATOR
*
       SPACE
TI
       LAL 128
       OUT PORT6
*                      RESET TTY
       LAS 0
       OUT PORT6
LOOP
       INP PORT6       WAIT FOR READY
       ANL 1           TEST IF READY
       JAZ LOOP        NO READY WAIT
       INP PORT5       READ DATA CHAR.
       RET
       EJECT

*          CONTINUE
       SPACE
*
OUT    LAS 0
*          CONTINUE
ERR
       LAS 0
       END
```

TABLE A2

ASSEMBLER

M38 ASSEMBLER **LISTING** PAGE

```
                    DEC OCTL    OCT EX
*     1                                    -*
*     2                                    -*          INPUT FROM TELETYPE
*     3                                    -*
*     4                         012 0A     -RAMCO  EQU 10          RAM CODE
*     5                         377 FF     -ONES   EQU 255         ALL ONES
*     6                         033 1B     -ESC    EQU 27
*     7                         005 05     -PORT5  EQU 5           DATA IN PORT
*     8                         006 06     -PORT6  EQU 6           CONTROL PORT
*     9                                    -*
*     10                                   -*
*     11          0 0000        004 04     -       LAL RAMCO       RAM CODE
*                 1 0001        012 0A
*     12          2 0002        032 1A     -       SAX             SET RAM CODE
*     13          3 0003        004 04     -       LAL ONES        RAM POINTER
*                 4 0004        377 FF
*     14
*     15          5 0005                   -BACK
*     16          5 0005        016 0E     -       ADL 1           INCREMENT
                  6 0006        001 01
*     17                                   -*                      RAM POINTER
*     18          7 0007        030 18     -       SAV             SAVE
*     19                                   -*                      RAM POINTER
*     20          8 0008        130 58     -       JAP AHEAD       JUMP TO READ
*                 9 0009        014 0C
*     21                                   -*                      CHARACTER FROM TTY
*     22         10 000A        100 40     -       JMP ERR         JUMP TO ERROR
*                11 000B        050 28
*     23                                   -*                      MORE THEN 128 CHARACTER
*     24
*     25         12 000C                   -AHEAD
*     26         12 000C        022 12     -       SZX             SET Z-RAM-REGISTER
*     27         13 000D        170 78     -       JSB TI          JUMP TO SUB.TI
*                14 000E        033 1B
*     28         15 000F        002 02     -       SIX             STORE DATA INTO RAM
*     29         16 0010        014 0C     -       EOL ONES        COMPL. ACC.
*                17 0011        377 FF
*     30         18 0012        016 0E     -       ADL 1
*                19 0013        001 01
*     31         20 0014        016 0E     -       ADL ESC         TEST FOR ESCAPE
*                21 0015        033 1B
*     32         22 0016        110 48     -       JAZ OUT         END OF DATA
*                23 0017        047 27
*     33         24 0018        010 08     -       LAV             RESUMES RAM POINTER
*     34         25 0019        100 40     -       JMP BACK
*                26 001A        005 05
```

TABLE A3 - ASSEMBLER LISTING (Continued)

M38 ASSEMBLER PAGE

```
                DEC OCTL    OCT EX
*    36                                 -*
*    37                                 -*          SUBROUTINE INPUT FROM TELETYPE
*    38                                 -*          THE INPUT CHARACTER IS
*    39                                 -*          IN THE ACCUMULATOR
*    40                                 -*
*    41
*    42          27 001B                -TI
*    43          27 001B    004 04      -       LAL 128
*                28 001C    200 80
*    44          29 001D    066 36      -       OUT PORT6
*    45                                 -*                          RESET TTY
*    46          30 001E    360 F0      -       LAS 0
*    47          31 001F    066 36      -       OUT PORT6
*    48          32 0020                -LOOP
*    49          32 0020    046 26      -       INP PORT6           WAIT FOR READY
*    50          33 0021    005 05      -       ANL 1               TEST IF READY
*                34 0022    001 01
*    51          35 0023    110 48      -       JAZ LOOP            NO READY WAIT
*                36 0024    040 20
*    52          37 0025    045 25      -       INP PORT5           READ DATA CHAR.
*    53          38 0026    000 00      -       RET
```

TABLE A4 - ASSEMBLER LISTING (Continued)

```
M38 ASSEMBLER                                                    PAGE
                    DEC OCTL    OCT EX
*    55                                    -*          CONTINUE
*    56
*    57                                    -*
*    58             39 0027     360 F0     -OUT    LAS 0
*    59                                    -*          CONTINUE
*    60             40 0028                -ERR
*    61             40 0028     360 F0     -       LAS 0
*    62                                    -       END


* * * TOTAL ERRORS NUMBER =       0
```

TABLE A5

ASSEMBLER

SYMBOL TABLE

* * * LABELS

-	1-	AHEAD	000C
-	2-	BACK	0005
-	3-	ERR	0028
-	4-	LOOP	0020
-	5-	OUT	0027
-	6-	TI	001B

* * * END LABELS

BRKPT PUNCH$

FREE DUMMY2.

TABLE A6

```
*
*          INPUT FROM TELETYPE
*
RAMCO EQU 10          RAM CODE
ONES  EQU 255         ALL ONES
ESC   EQU 27
PORT5 EQU 5           DATA IN PORT
PORT6 EQU 6           CONTROL PORT
*
*
BACK
      LAL RAMCO       RAM CODE
      SAX             SET RAM CODE
      LAL ONES        RAM POINTER
      SPACE
BACK                                    ERROR:
      ADL 1           INCREMENT           LABEL .. DUPLICATED
*                     RAM POINTER
      SAV             SAVE
*                     RAM POINTER
      JAP AHEAD       JUMP TO READ
*                     CHARACTER FROM TTY
      JMP ERR         JUMP TO ERROR
*                     MORE THEN 128 CHARACTERS
      SPACE
   AHEAD                                ERROR: A SPACE BEFORE
      SZX             SET Z-RAM-REGISTER      LABEL ..
      JSB TI          JUMP TO SUB.TI
      SIX             STORE DATA INTO RAM
      EOL ONES        COMPL. ACC.
      ADL 1
      ADL ESC         TEST FOR ESCAPE
      JAZ OUT         END OF DATA
      LAV             RESUMES RAM POINTER
      JMP BACKK                         ERROR: UNKNOWN LABEL
      EJECT
*
*          SUBROUTINE INPUT FROM TELETYPE
*          THE INPUT CHARACTER IS
*          IN THE ACCUMULATOR
*
      SPACE
TI                                      ERROR: IMMEDIATE OUT
      LAL 328                                  OF RANGE
      OUT PORT6
*                     RESET TTY
      LAS 0
      OUT PORT6
LOOP
      INP PORT6       WAIT FOR READY
      ANL 1           TEST IF READY
      JAZ LOOP        NO READY WAIT
      INP PORT5       READ DATA CHAR.
      REV                               ERROR: UNKNOWN OP. CODE
```

TABLE A7

ERROR FLAGS

LISTING

M38 ASSEMBLER PAGE

```
                    DEC OCTL    OCT EX

*     1                                 -*
*     2                                 -*          INPUT FROM TELETYPE
*     3                                 -*
*     4                       012 0A    -RAMCO  EQU 10          RAM CODE
*     5                       377 FF    -ONES   EQU 255         ALL ONES
*     6                       033 1B    -ESC    EQU 27
*     7                       005 05    -PORT5  EQU 5           DATA IN PORT
*     8                       006 06    -PORT6  EQU 6           CONTROL PORT
*     9                                 -*
*    10                                 -*
*    11     D       0 0000              -BACK
*    12             0 0000    004 04    -       LAL RAMCO       RAM CODE
*                   1 0001    012 0A
*    13             2 0002    032 1A    -       SAX             SET RAM CODE
*    14             3 0003    004 04    -       LAL ONES        RAM POINTER
*                   4 0004    377 FF
*    15
     16     D       5 0005              -BACK
     17             5 0005    016 0E    -       ADL 1           INCREMENT
*                   6 0006    001 01
*    18                                 -*                      RAM POINTER
*    19             7 0007    030 18    -       SAV             SAVE
*    20                                 -*                      RAM POINTER
*    21     S       8 0008    130 58    -       JAP AHEAD       JUMP TO READ
*                   9 0009    000 00
*    22                                 -*                      CHARACTER FROM TTY
*    23            10 000A    100 40    -       JMP ERR         JUMP TO ERROR
*                  11 000B    047 27
*    24                                 -*                      MORE THEN 128 CHARACTE
*    25
*    26     0                           -   AHEAD
*    27            12 000C    022 12    -       SZX             SET Z-RAM-REGISTER
*    28            13 000D    170 78    -       JSB TI          JUMP TO SUB.TI
*                  14 000E    033 1B
*    29            15 000F    002 02    -       SIX             STORE DATA INTO RAM
*    30            16 0010    014 0C    -       EOL ONES        COMPL. ACC.
                   17 0011    377 FF
*    31            18 0012    016 0E    -       ADL 1
*                  19 0013    001 01
*    32            20 0014    016 0E    -       ADL ESC         TEST FOR ESCAPE
*                  21 0015    033 1B
*    33            22 0016    110 48    -       JAZ OUT         END OF DATA
*                  23 0017    046 26
*    34            24 0018    010 08    -       LAV             RESUMES RAM POINTER
*    35     S      25 0019    100 40    -       JMP BACKK
*                  26 001A    000 00
```

ERROR FLAGS

LISTING

M38 ASSEMBLER PAGE

```
                    DEC OCTL   OCT EX
*   37                                  -*
*   38                                  -*      SUBROUTINE INPUT FROM TELETYPE
*   39                                  -*      THE INPUT CHARACTER IS
*   40                                  -*      IN THE ACCUMULATOR
*   41                                  -*
*   42
*   43              27 001B             -TI
*   44    F         27 001B   004 04    -       LAL 328
*                   28 001C   000 00
*   45              29 001D   066 36    -       OUT PORT6
*   46                                  -*                      RESET TTY
*   47              30 001E   360 F0    -       LAS 0
*   48              31 001F   066 36    -       OUT PORT6
*   49              32 0020             -LOOP
*   50              32 0020   046 26    -       INP PORT6       WAIT FOR READY
*   51              33 0021   005 05    -       ANL 1           TEST IF READY
*                   34 0022   001 01
*   52              35 0023   110 48    -       JAZ LOOP        NO READY WAIT
*                   36 0024   040 20
*   53              37 0025   045 25    -       INP PORT5       READ DATA CHAR.
*   54    0                             -       REV
```

SIMULATION SOFTWARE PACKAGE

1.0 - General information

The M38 simulator is a computer program written in ANSI standard FORTRAN IV language, which enables the user to test by a software simulation any application program of an M38 system.

The simulator process goes through the following phases:

1 - Hardware definition of the system
2 - Program loading
3 - Input of the executive commands
4 - Simulation

The required information may be loaded into the computer either by a card reader or by a time sharing terminal.

In the second case, the simulation is an interactive process and gives to the user a high degree of flexibility.

2.0 - System definition (see table S1, S6, S10, S12)

The system definition is started by the command

$SYSTEM

and is ended by a "$" on the first column.
In this phase the following information may be entered:

1 - PSE Module code

Every 2K ROM or every fraction of 2K ROM has to be specified by this command.
On the first field the first module code of the corresponding 2K ROM has to be indicated.
One line is needed for every ROM block.

2 - DSE Module code

Every block of 128 RAM words included in the system must be specified by this command.
On the first field the module code assigned to the RAM block has to be indicated.
One line is needed for every RAM block.

3 - I/O Module code

Every I/O port of the system must be specified by this command, indicating on the first field the module code assigned to the I/O port. One line is needed for every I/O port.

3.0 - Program loading - Executive commands - Simulation (see table S1, S6, S10, S12)

A - The load command will transfer during this phase the machine code produced by the cross-assembler into the proper area of the simulator.

B - The executive commands may be entered during the next phase.

The executive commands allow:

- To display the memory and register status
- To modify the simulation system (memory, registers, initial value set up...)
- To inform the simulator about program break points and tracing conditions

They are described in the next paragraph.

C - The GO command will then activate the simulation program.

The simulation will proceed until a breakpoint condition is met.

D - New executive commands may be entered and the simulation may be started again.

Please note that the order by which the executive commands are entered is not important.

3.1 - Examples of the simulation package

The tables from S1 to S16 show some simulation examples where the most important executive commands are illustrated.
The program which the examples refer to is the same that has been used to demonstrate the assembler package. The example has been discussed in the section 'Some simple examples' under the paragraph 'Data input from teletype' (page 174).

4.0 - Executive commands

The executive commands, their format and meaning are summarized briefly below.
The key word of every executive command must be preceeded by a "$" and it may be followed by some fields separated by a comma on which the information required by the command must be specified.

1 - LOAD type, module, source

The LOAD command reads the machine code of the source file into the simulated memory whose module code is specified in the second field.
The first field is used to specify the memory type (ROM or RAM).
Looking at the table S1 the 'LOAD' command has the following format:

```
$LOAD  ROM,Ø,5
 ADD,P DUMMY2
```

The meaning of this command is:

A - Load the source file into the ROM whose first module code is Ø.

B - The source file is indicated in a way that is peculiar for the EXEC 8 Univac system, but the source file call may be fitted to any computer or miniocomputer with its own input/output configuration.
The execution of the 'LOAD' command will access the 'ROM CODE' indicated in table S3 and transfer it into the proper ROM area.

2 - GO module, address

The GO command activates the execution of the loaded program.
The start address is indicated by the first field (ROM module code) and the second (address inside the module).
In case that a breakpoint was previously encountered, the execution continues and the module address may not be specified.

3 - BREAK instruction number

The BREAK command automatically stops the simulation after the instruction number specified in the first field.
In table S1 a break after 1000 instruction is indicated but the end of the simulation process is not printed out.
The same break condition is used in the example of table S10 and in table S11 is shown the result: after the break condition is reached, the next execution command is executed.

4 - TRACE type, module, number, number, mode

This command enables the tracing operation when the specified conditions are met.
Tracing conditions are indicated in field 1, 2,3, 4.
The first field will have the following information:

- ABS : Absolute tracing
  Tracing is under control of CPU cycle counter.
  An example of 'Absolute' tracing is shown in tables S1 to S5.
- PC : Program counter tracing
  Tracing is under control of the program counter.
  An example of 'Program Counter' tracing is shown in tables S6 to S8.
- INSTR : Instruction tracing
  Tracing is under control of the instruction code to be executed.

The second field will indicate the module code to whom the tracing is referred to.
The third and fourth field will indicate the start and the stop limits for the tracing.
The fifth field will specify the data to be printed by the following format:

A - CPU
The data of the CPU registers (including S,T register, the PMC register and the I/O flip-flops) are printed out. (See table S6,S7).
The CPU-RAM is printed out page by page, every page being a line.

B - RAM code
The data of the RAM specified by the module code indicated on the first field is printed out.

C - ROM code
The data of 2K ROM specified by the module code indicated in the first field are printed out.

D - I/O code
Idem for the I/O port.

E - NORMAL
By this way a short form of tracing is requested (see table S1 to S5). The result of this command is shown in table S4 and S5. For every instruction executed the following data will be printed out on one line:

- Program number of the executed instruction
- Program counter
- The PMC register
- The Instruction register
- The cumulative time required for the program execution
- The accumulator and the carry from the 8th bit
- The data-bus and address-bus

In addition to that, every time an I/O operation is performed, it is also printed out:

- The arrows ' ' will indicate an output
- The arrows ' ' will indicate an input

5 - STOP condition, module, address

The stop command stops the simulation when the condition specified by the first field is met in the memory location indicated by the second field and third field.
The conditions are:

- ABS : Memory location addressed
- CHANGE : Memory content changed

6 - DISPLAY area, area, area,.....

This command displays memory lacation, CPU registers, I/O ports, as indicated by the first field.
The memory location to be displayed have to be indicated by one or more of the following mnemonics (see table S1, S6, S10, S12):

- ROM code
- RAM code
- I/O Code
- CPU

separated by a comma.

7 - CHANGE type, module, address start, address end, value

This command will set to the specified value a memory block, as indicated by the field one to four:

- The first field is used to specify the type of memory on which the changes are to be done (CPU, RAM code, ROM code, I/O code).
- The second field is used to specify the module code of the memory.
- The third and fourth field are for the start and end address of the locations which are to be changed.

8 - PRINT
The PRINT command controls the output listing.
The command $PRINT Ø disables any diagnostic message and any message related to I/O operations.
Only the TRACE and DISPLAY commands are satistied (see table S1Ø and S11).
To enable the diagnostic and I/O messages, the $PRINT 1 must be entered.

9 - NAME
This command allows the user to specify 6 alphanumeric characters to be used on punched cards or on the listing as a title

10 - PUNCH type, module, output file, start address, end address.
This command will allows the user to punch the memory content whose address is specified by the 1, 2, 4, 5 field.
The format is the one required for ROM programming.
The third field is available to specify the output device (magnetic type, card punch or a file).

The command BREAK, TRACE, STOP may be deleted by using / before the command word.

TABLE S1 **SIMULATION : ABSOLUTE TRACING**

```
 ELT,IDL  TPF$.CONTST
ELT006-RLIB67-10  01/21-16:19:29
CYCLE (00)
000001      000   $PRINT 1
000002      000   $SYSTEM
000003      000   PSE 00
000004      000   DSE 10
000005      000   I/O 60
000006      000   I/O 61
000007      000   I/O 62
000008      000   $
000009      000   $BREAK 1000
000010      000   $LOAD ROM,0,5
000011      000    ADD,P  DUMMY2.
000012      000   $TRACE ABS,0,0,250,NORMAL
000013      000   $GO 0,0
000014      000     0
000015      000     0
000016      000     0
000017      000     1
000018      000     8
000019      000     0
000020      000     0
000021      000     0
000022      000     0
000023      000     0
000024      000     0
000025      000     1
000026      000    16
000027      000     0
000028      000     0
000029      000     0
000030      000     1
000031      000    24
000032      000     0
000033      000     0
000034      000     1
000035      000    32
000036      000     0
000037      000     1
000038      000    40
000039      000     0
000040      000     1
000041      000     9
000042      000     0
000043      000     1
000044      000    27
000045      000   $DISPLAY  CPU,RAM 10
000046      000   $EOF
```

SYSTEM DEFINITION

EXECUTIVE COMMANDS

ROM CODE FILE

DATA

```
 XQT  MOLINA*CP3F.ABS
```

TABLE S2

```
* SGS-ATES * DIGITAL SYSTEMS ENGINEERING *
* CP-3F MICROPROCESSOR-SYSTEM * SIMULATION PROGRAM *


ADD,P  TPF$.CONTST                                  SIMULATED SYSTEM
CPU I/O PORT CODE ASSIGNED IS  :  63
MODULE CODE ASSIGNED TO PSE IS :   0
MODULE CODE ASSIGNED TO PSE IS :   1
MODULE CODE ASSIGNED TO PSE IS :   2
MODULE CODE ASSIGNED TO PSE IS :   3
MODULE CODE ASSIGNED TO PSE IS :   4
MODULE CODE ASSIGNED TO PSE IS :   5
MODULE CODE ASSIGNED TO PSE IS :   6
MODULE CODE ASSIGNED TO PSE IS :   7

MODULE CODE ASSIGNED TO DSE IS :  10
MODULE CODE ASSIGNED TO I/O IS :  60
MODULE CODE ASSIGNED TO I/O IS :  61
MODULE CODE ASSIGNED TO I/O IS :  62
$BREAK 1000

SIMULATION WILL STOP AFTER :    1000   INSTRUCTIONS EXECUTED


$LOAD ROM,0,5

            LOADED PROGRAM

   MODULE      ADDRESS     INSTRUCTION
```

TABLE S3

ADD,P DUMMY2.

0	0	004
0	1	012
0	2	032
0	3	004
0	4	377
0	5	016
0	6	001
0	7	030
0	8	130
0	9	014
0	10	100
0	11	050
0	12	022
0	13	170
0	14	033
0	15	002
0	16	014
0	17	377
0	18	016
0	19	001
0	20	016
0	21	033
0	22	110
0	23	047
0	24	010
0	25	100
0	26	005
0	27	004
0	28	200
0	29	066
0	30	360
0	31	066
0	32	046
0	33	005
0	34	001
0	35	110
0	36	040
0	37	045
0	38	000
0	39	360
0	40	360

ROM CODE

41 WORDS LOADED O.K.

$TRACE ABS,0,0,250,NORMAL

TRACE OK.

$TRACE PC,0,16,20,CPU

TRACE OK.

$GO 0,0

TABLE S4 **ABS TRACING**

EXCUTD INSTR.	PC	YER	I.R.	TIME	A	C	DAB	ADB
1	1	000	004	5	000	0	004	000
2	3	000	032	15	012	0	032	000
3	4	000	004	20	012	0	004	000
4	6	000	016	30	377	0	016	000
5	8	000	030	40	000	0	030	000
6	9	000	130	45	000	0	130	000
7	13	000	022	65	000	0	022	000
8	14	000	170	80	000	0	170	000
9	28	000	004	100	000	0	004	000
10	30	000	066	110	200	0	066	000
		>>>>> PORT# 62		DATA OUT: 128				
11	31	000	360	125	200	0	360	000
12	32	000	066	130	000	0	066	000
		>>>>> PORT# 62		DATA OUT: 0				
13	33	000	046	145	000	0	046	000
		<<<<< PORT# 62						
		<<<<< DATA: 0						
14	34	000	005	155	000	0	005	000
15	36	000	110	165	000	0	110	000
16	33	000	046	185	000	0	046	000
		<<<<< PORT# 62						
		<<<<< DATA: 0						
17	34	000	005	195	000	0	005	000
18	36	000	110	205	000	0	110	000
19	33	000	046	225	000	0	046	000
		<<<<< PORT# 62						
		<<<<< DATA: 0						
20	34	000	005	235	000	0	005	000
21	36	000	110	245	000	0	110	000
22	33	000	046	265	000	0	046	000
		<<<<< PORT# 62						
		<<<<< DATA: 1						
23	34	000	005	275	001	0	005	000
24	36	000	110	285	001	0	110	000
25	38	000	045	295	001	0	045	000
		<<<<< PORT# 61						
		<<<<< DATA: 8						

TABLE S5 **ABS TRACING**

EXCUTD INSTR.	PC	YER	I.R.	TIME	A	C	DAB	ADB
26	39	000	000	305	010	0	000	000
27	16	000	002	315	010	0	002	000
28	17	000	014	330	010	0	014	000
29	19	000	016	340	367	0	016	000
30	21	000	016	350	370	0	016	000
31	23	000	110	360	023	0	110	000
32	25	000	010	370	023	0	010	000
33	26	000	100	375	000	0	100	000
34	6	000	016	395	000	0	016	000
35	8	000	030	405	001	0	030	000
36	9	000	130	410	001	0	130	000
37	13	000	022	430	001	0	022	000
38	14	000	170	445	001	0	170	000
39	28	000	004	465	001	0	004	000
40	30	000	066	475	200	0	066	000
		>>>>> PORT# 62		DATA OUT: 128				
41	31	000	360	490	200	0	360	000
42	32	000	066	495	000	0	066	000
		>>>>> PORT# 62		DATA OUT: 0				
43	33	000	046	510	000	0	046	000
		<<<<< PORT# 62						
		<<<<< DATA: 0						
44	34	000	005	520	000	0	005	000
45	36	000	110	530	000	0	110	000
46	33	000	046	550	000	0	046	000
		<<<<< PORT# 62						
		<<<<< DATA: 0						
47	34	000	005	560	000	0	005	000
48	36	000	110	570	000	0	110	000
49	33	000	046	590	000	0	046	000
		<<<<< PORT# 62						
		<<<<< DATA: 0						
50	34	000	005	600	000	0	005	000

EXCUTD INSTR.	PC	YER	I.R.	TIME	A	C	DAB	ADB
51	36	000	110	610	000	0	110	000
52	33	000	046	630	000	0	046	000
		<<<<< PORT# 62						

TABLE S6 **SIMULATION: PROGRAM COUNTER TRACING**

```
 ELT,IDL  TPF$.CONTST
ELT006-RLI967-10  01/21-17:49:49
CYCLE (00)
000001      000   $PRINT 1
000002      000   $SYSTEM
000003      000   PSE 00
000004      000   DSE 10
000005      000   I/O 60
000006      000   I/O 61
000007      000   I/O 62
000008      000   $
000009      000   $BREAK 1000
000010      000   $LOAD ROM,0,5
000011      000    ADD,P  DUMMY2.
000012      000   $TRACE  PC,0,16,20,CPU
000013      000   $GO 0,0
000014      000     0
000015      000     0
000016      000     0
000017      000     1
000018      000     8
000019      000     0
000020      000     0
000021      000     0
000022      000     0
000023      000     0
000024      000     0
000025      000     1
000026      000    16
000027      000     0
000028      000     0
000029      000     0
000030      000     1
000031      000    24
000032      000     0
000033      000     0
000034      000     1
000035      000    32
000036      000     0
000037      000     1
000038      000     40
000039      000     0
000040      000     1
000041      000     9
000042      000     0
000043      000     1
000044      000    27
000045      000   $DISPLAY  CPU,RAM 10
000046      000   $EOF
```

```
 XQT  MOLINA*CP3F.ABS
```

Note: This printout will be followed by the same printout shown on table S2, S3, omitted here for sake of simplicity.
After that, the printout continues by table S7.

./..

TABLE S7

```
>>>>> PORT# 62    DATA OUT: 128

>>>>> PORT# 62    DATA OUT:   0

<<<<< PORT# 62
<<<<< DATA:   0

<<<<< PORT# 62
<<<<< DATA:   0

<<<<< PORT# 62
<<<<< DATA:   0

<<<<< PORT# 62
<<<<< DATA:   1

<<<<< PORT# 61
<<<<< DATA:   8
```

PC TRACING START

```
                CPU-RAM
  0   1   2   3   4   5   6   7        A   S   T  YER  C  DAB ADB I/O
000 000 000 000 000 000 000 000      010 000 000 000   0  002 000 000
000 000 000 000 000 000 012 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000

 PC    RA    RB    RZ    INSTR. N.    I.R.       TIME  (USEC.)
 16     0     0     0          27     002                315

                CPU-RAM
  0   1   2   3   4   5   6   7        A   S   T  YER  C  DAB ADB I/O
000 000 000 000 000 000 000 000      010 000 000 000   0  014 000 000
000 000 000 000 000 000 012 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000

 PC    RA    RB    RZ    INSTR. N.    I.R.       TIME  (USEC.)
 17     0     0     0          28     014                330

                CPU-RAM
  0   1   2   3   4   5   6   7        A   S   T  YER  C  DAB ADB I/O
```

TABLE S8

```
000 000 000 000 000 000 000 000     367 000 000 000  0  016 000 000
000 000 000 000 000 000 012 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000

 PC    RA    RB    RZ    INSTR. N.    I.R.       TIME  (USEC.)
 19     0     0     0          29     016                340
```

TR. END

```
                    >>>>> PORT# 62    DATA OUT: 128

                    >>>>> PORT# 62    DATA OUT:   0

                    <<<<< PORT# 62
                    <<<<< DATA:   0

                    <<<<< PORT# 62
                    <<<<< DATA:   0

                    <<<<< PORT# 62
                    <<<<< DATA:   0

                    <<<<< PORT# 62
                    <<<<< DATA:   0

                    <<<<< PORT# 62
                    <<<<< DATA:   0

                    <<<<< PORT# 62
                    <<<<< DATA:   0

                    <<<<< PORT# 62
                    <<<<< DATA:   1

                    <<<<< PORT# 61
                    <<<<< DATA:  16
```

PC TRACING START

```
                CPU-RAM
  0   1   2   3   4   5   6   7      A   S   T  YER  C  DAB ADB I/O
000 000 000 000 000 000 000 000    020 000 000 000  0  002 000 000
000 000 000 000 001 000 012 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
```

TABLE S9

```
 PC    RA    RB    RZ    INSTR. N.    I.R.       TIME  (USEC.)
 16     0     0     0          66     002             800

                  CPU-RAM
  0   1   2   3   4   5   6   7        A   S   T  YER  C  DAB ADB I/O
000 000 000 000 000 000 000 000      020 000 000 000  0  014 000 000
000 000 000 000 001 000 012 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000

 PC    RA    RB    RZ    INSTR. N.    I.R.       TIME  (USEC.)
 17     0     0     0          67     014             815

                  CPU-RAM
  0   1   2   3   4   5   6   7        A   S   T  YER  C  DAB ADB I/O
000 000 000 000 000 000 000 000      357 000 000 000  0  016 000 000
000 000 000 000 001 000 012 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000

 PC    RA    RB    RZ    INSTR. N.    I.R.       TIME  (USEC.)
 19     0     0     0          68     016             825
```

TR. END

```
>>>>> PORT# 62    DATA OUT: 128

>>>>> PORT# 62    DATA OUT:   0

<<<<< PORT# 62
<<<<< DATA:   0

<<<<< PORT# 62
<<<<< DATA:   0

<<<<< PORT# 62
<<<<< DATA:   0

<<<<< PORT# 62
<<<<< DATA:   1

<<<<< PORT# 61
<<<<< DATA:  24
```

C.I. N.39.1772.76 - MF/rtc 6/4/76 -

SIMULATION:
PRINT Ø
NO TRACING
DIPLAY

TABLE S10

```
 ELT,IDL  TPF$.CONTST
ELT006-RLIB67-10  01/21-14:58:28
CYCLE (00)
000001      000   $PRINT 0
000002      000   $SYSTEM
000003      000   PSE 00
000004      000   DSE 10
000005      000   I/O 60
000006      000   I/O 61
000007      000   I/O 62
000008      000   $
000009      000   $BREAK 1000
000010      000   $LOAD ROM,0,5
000011      000    ADD,P  DUMMY2.
000012      000   $GO 0,0
000013      000     0
000014      000     0
000015      000     0
000016      000     1
000017      000     8
000018      000     0
000019      000     0
000020      000     0
000021      000     0
000022      000     0
000023      000     0
000024      000     1
000025      000    16
000026      000     0
000027      000     0
000028      000     0
000029      000     1
000030      000    24
000031      000     0
000032      000     0
000033      000     1
000034      000    32
000035      000     0
000036      000     1
000037      000    40
000038      000     0
000039      000     1
000040      000     9
000041      000     0
000042      000     1
000043      000    27
000044      000   $DISPLAY  CPU,RAM 10
000045      000   $EOF
```

```
XQT  MOLINA*CP3F.ABS
```

TABLE S11 (simulation: Print Ø - No Tracing - Display -comtinue)

```
* SGS-ATES * DIGITAL SYSTEMS ENGINEERING *
* CP-3E MICROPROCESSOR-SYSTEM * SIMULATION PROGRAM *



ADD,P  TPES.CONTST


ADD,P  DUMMY2.

  41 WORDS LOADED O.K.

    GO O.K.                          EVERY DIAGNOSTIC AND I/O
                                     PRINT-OUT IS INHIBITED.
                                     DISPLAY IS EXECUTED

*** FETCH NOT POSSIBLE ***

*** ILLEGAL GUARD MODE /SIMULATION ROUTINE


                CPU-RAM
  0   1   2   3   4   5   6   7        A   S   T  YER  C  DAB ADB I/O
000 000 000 000 000 000 000 000      000 000 000 000  0  000 000 000
000 000 000 000 006 000 012 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000

 PC    RA    RB    RZ    INSTR. N.    I.R.       TIME  (USEC.)
 42     0     0     0         202     000             2420


          DSE  10 MODULE DUMP
   0   010 020 030 040 050 011 033 000 000 000 000 000 000 000 000 000
  16   000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
  32   000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
  48   000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
  64   000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
  80   000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
  96   000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
 112   000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000


          RZ :   6


DISPLAY OK.
```

ABSOLUTE TRACING
BREAK AFTER 27 INSTR.
DISPLAY CPU, RAM

TABLE S12

```
 ELT,IDL  TPF$.CONTST
ELT006-RLIB67-10  01/21-18:07:11
CYCLE (00)
000001      000   $PRINT 1
000002      000   $SYSTEM
000003      000   PSE 00
000004      000   DSE 10
000005      000   I/O 60
000006      000   I/O 61
000007      000   I/O 62
000008      000   $
000009      000   $BREAK 27
000010      000   $LOAD ROM,0,5
000011      000    ADD,P  DUMMY2.
000012      000   $TRACE ABS,0,0,250,NORMAL
000013      000   $GO 0,0
000014      000     0
000015      000     0
000016      000     0
000017      000     1
000018      000     8
000019      000   $DISPLAY CPU,RAM 10
000020      000   $BREAK 10
000021      000   $GO
000022      000   $DISPLAY CPU,RAM 10
000023      000   $BREAK 1000
000024      000   $GO
000025      000     0
000026      000     0
000027      000     0
000028      000     0
000029      000     0
000030      000     0
000031      000     1
000032      000    16
000033      000     0
000034      000     0
000035      000     0
000036      000     1
000037      000    24
000038      000     0
000039      000     0
000040      000     1
000041      000    32
000042      000     0
000043      000     1
000044      000     40
000045      000     0
000046      000     1
000047      000     9
000048      000     0
000049      000     1
000050      000    27
000051      000   $DISPLAY  CPU,RAM 10
000052      000   $EOF
```

TABLE S13 (Sim. Abs - Break - Display - continue)

```
* SGS-ATES * DIGITAL SYSTEMS ENGINEERING *
* CP-3F MICROPROCESSOR-SYSTEM * SIMULATION PROGRAM *



ADD,P  TPES.CONTST

CPU I/O PORT CODE ASSIGNED IS  :  63

MODULE CODE ASSIGNED TO PSF IS :   0

MODULE CODE ASSIGNED TO PSE IS :   1

MODULE CODE ASSIGNED TO PSE IS :   2

MODULE CODE ASSIGNED TO PSE IS :   3

MODULE CODE ASSIGNED TO PSE IS :   4

MODULE CODE ASSIGNED TO PSE IS :   5

MODULE CODE ASSIGNED TO PSE IS :   6

MODULE CODE ASSIGNED TO PSE IS :   7

MODULE CODE ASSIGNED TO OSE IS :  10

MODULE CODE ASSIGNED TO I/O IS :  60

MODULE CODE ASSIGNED TO I/O IS :  61

MODULE CODE ASSIGNED TO I/O IS :  62

$BREAK 27

SIMULATION WILL STOP AFTER :      27   INSTRUCTIONS EXECUTED


$LOAD ROM,0,5

          LOADED PROGRAM

   MODULE     ADDRESS     INSTRUCTION
```

TABLE S14 (Sim. Abs - Break - Display - continue)

```
ADD,P   DUMMY2.
        0               0               004
        0               1               012
        0               2               032
        0               3               004
        0               4               377
        0               5               016
        0               6               001
        0               7               030
        0               8               130
        0               9               014
        0              10               100
        0              11               050
        0              12               022
        0              13               170
        0              14               033
        0              15               002
        0              16               014
        0              17               377
        0              18               016
        0              19               001
        0              20               016
        0              21               033
        0              22               110
        0              23               047
        0              24               010
        0              25               100
        0              26               005
        0              27               004
        0              28               200
        0              29               066
        0              30               360
        0              31               066
        0              32               046
        0              33               005
        0              34               001
        0              35               110
        0              36               040
        0              37               045
        0              38               000
        0              39               360
        0              40               360

  41 WORDS LOADED O.K.

$TRACE ABS,0,0,250,NORMAL

  TRACE OK.

$GO 0,0

 START ADDRESS ON MODULE   0 ,AT LOCATION      0

    GO O.K.
```

TABLE S 15 (Sim. Abs - Break - Display - continue)

EXCUTD INSTR.	PC	YER	I.R.	TIME	A	C	DAB	ADB
1	1	000	004	5	000	0	004	000
2	3	000	032	15	012	0	032	000
3	4	000	004	20	012	0	004	000
4	6	000	016	30	377	0	016	000
5	8	000	030	40	000	0	030	000
6	9	000	130	45	000	0	130	000
7	13	000	022	65	000	0	022	000
8	14	000	170	80	000	0	170	000
9	28	000	004	100	000	0	004	000
10	30	000	066	110	200	0	066	000
		>>>>> PORT# 62		DATA OUT: 128				
11	31	000	360	125	200	0	360	000
12	32	000	066	130	000	0	066	000
		>>>>> PORT# 62		DATA OUT: 0				
13	33	000	046	145	000	0	046	000
		<<<<< PORT# 62						
		<<<<< DATA: 0						
14	34	000	005	155	000	0	005	000
15	36	000	110	165	000	0	110	000
16	33	000	046	185	000	0	046	000
		<<<<< PORT# 62						
		<<<<< DATA: 0						
17	34	000	005	195	000	0	005	000
18	36	000	110	205	000	0	110	000
19	33	000	046	225	000	0	046	000
		<<<<< PORT# 62						
		<<<<< DATA: 0						
20	34	000	005	235	000	0	005	000
21	36	000	110	245	000	0	110	000
22	33	000	046	265	000	0	046	000
		<<<<< PORT# 62						
		<<<<< DATA: 1						
23	34	000	005	275	001	0	005	000
24	36	000	110	285	001	0	110	000
25	38	000	045	295	001	0	045	000
		<<<<< PORT# 61						
		<<<<< DATA: 8						

TABLE S16 (Sim. Abs - Break - Display - continue)

```
EXECUTE INSTR.    PC    YER    I.R.        TIME    A    C   DAB ADB

      26          39    000    000          305   010   0   000 000
      27          16    000    002          315   010   0   002 000
                                                    NOW BREAK IN EXECUTION
SIMULATION BREAK AFTER :        27  EXECUTED INSTRUCTIONS

$DISPLAY CPU,RAM 10

                CPU-RAM
  0   1   2   3   4   5   6   7       A   S   T   YER   C   DAB ADB I/O
000 000 000 000 000 000 000 000     010 000 000 000   0   002 000 000
000 000 000 000 000 000 012 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000

 PC    RA    RB    RZ    INSTR. N.    I.R.       TIME  (USEC.)
 16     0     0     0          27     002             315

          DSE  10 MODULE DUMP
   0   000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
  16   000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
  32   000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
  48   000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
  64   000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
  80   000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
  96   000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
 112   000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000

        RZ :   0

  DISPLAY OK.

$BREAK 10
                                                    NEW EXEC. COMMAND
                                                    ENTERED
SIMULATION WILL STOP AFTER :      10   INSTRUCTIONS EXECUTED


$GO

 START ADDRESS ON MODULE  0 ,AT LOCATION    16

    GO O.K.
```

ELECTRICAL SPECIFICATIONS

ABSOLUTE MAXIMUM RATINGS

V_{DD}	Supply voltage (referred to V_{SS})	-20 to + 0.5	V
V_I	Input voltage	V_{DD} to + 0.5	V
I_O	Output current (at any pin)	3	mA
T_{stg}	Storage temperature	-65 to 150	°C
T_{op}	Operating temperature	0 to 70	°C

RECOMMENDED OPERATING CONDITIONS

V_{GG}	Supply voltage	- 17 ± 5%	V
V_{DD}	Supply voltage	- 5 ± 5%	V
V_I	Input voltage	0 to V_{DD}	V
V_{dk}	Clock generator ampliture	17 ± 10%	V
f_i	Input frequency	800	KHz
T_{op}	Operating temperature	0 to 70	°C

* Typical data power consumption per chip 500 mW

STATIC ELECTRICAL CHARACTERISTICS for M380-M381-M382-M383 (Tamb = 25 °C, V_{ss} = 0V, over recommended operating conditions)
Logic "1" is defined as the most negative level.
Logic "0" is defined as the most positive level.

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNIT
I_{DD} Average supply current from V_{DD}	for M 380		20	48	mA
	M 381		20	48	mA
	M 382		25	55	mA
	M 383		0	0	mA
I_{GG} Average supply current from V_{GG}	for M 380		30	55	mA
	M 381		27	50	mA
	M 382		36	50	mA
	M 383		20	32	mA

./..

STATIC ELECTRICAL CHARACTERISTICS (continued)

ELECTRICAL SPECIFICATIONS (cont.)

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
I_L Input voltage	Data bus	V_I = - 5 V			1	uA
	all other pins				10	uA
V_{OL} Low level output voltage	Data bus	C_L = 275 pF				
	Peripheral channel	See test circuit			-4	V
	Address and control bus	C_L=200 pF for M 380 only				
V_{OH} High level output voltage	Data bus	C_L = 275 pF				
	Peripheral channel	I_{OH} = 0.85 mA	-1			V
	Addres and control bus	C_L=200 pF for M 381 only				
V_{IL} Low level input voltage	Clock		V_{GG}		-14,3	V
	Reset		V_{GG}		-14,3	V
	Data bus				- 4	V
	Peripheral channel				- 4	V
	Address bus and Control	for M 381 - M 382 M 383 only			- 4	V
V_{IH} High level input voltage	Clock		-0,5			V
	Reset		- 1			V
	Data bus		-1,2			V
	Peripheral channel		-1,2			V
	Address bus and control	for M 381 - M 382 M 383 only	- 1			V
I_{OH} High level output current	Peripheral bus	V_{OHP} --1 V	0,85			mA

SGS-ATES COMPONENTI ELETTRONICI S.p.A.

ELECTRICAL SPECIFICATIONS (Continue)

DYNAMIC ELECTRICAL CHARACTERISTICS: (V_{GG} = -17V ± 5%, V_{SS} = 0V, V_{DD} = -5V ± 5%; Tamb = 0 to 70 °C unless otherwise specified)

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNITS
T Clock period		t_r, t_f 130 ns	1.25		3	/us
t_r, t_f Clock rise and fall times		between -5 to -14.3 V			130	ns
t_w clock pulse width	Low Level	measured at -14.3 V	450			ns
	High Level	measured at - 0.5 V	450			ns
t_{hold} Hold time	SYNC (high to low with reference to clock (high to low)	measured at - 1 V			200	ns
	*Address and control lines with reference to clock (high to low)	for M 381 - 382- 383			0.3	/us
t_p Output delay tima	** Address bus	C_L = 200 pF for M 380			0.3	/us
	** Data bus	C_L = 275 pF for 380 for M381-382-383			0.6 1.2	/us
	** Peripheral channel	C_L = 400 pF for M 380 C_L = 575pF for M381-382-383			1 1	/us
t_{setup} Input setup time	** Data bus	for M 380 for M 381-382-383			1.1 0.6	/us
	** Peripheral channel *	for M 380 for M381-382-383			0.8 0.8	/us

CAPACITANCE

C_I Input capacitance	f = 1 MHz	5	10	pF
C In/Out capacitance on data bus and peripheral channels	f = 1 MHz	7	15	pF

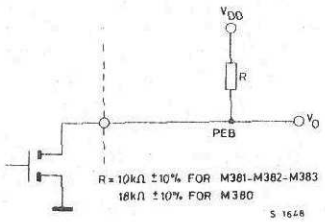
* Measured from F3 negative clock edge for M 381 -382 - 383 only -

** Measured from F4 negative clock edge -

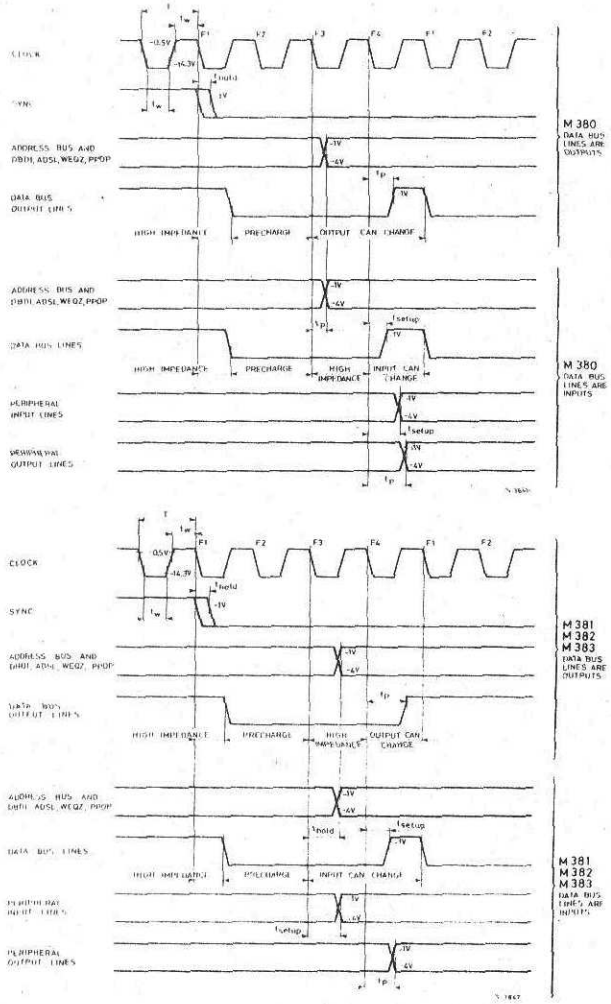
./..

ELECTRICAL SPECIFICATIONS (continue)

TEST CIRCUIT (for peripheral channels)



- Timing diagrams



CUSTOM ROM PROGRAMMING INFORMATION

ROM programming information for M382 and M383 devices should be sent in the form of computer punched cards. AN 80 column Hollerith card, preferably interpreted, punched by an IBM 026 or 029 keypunch should be submitted.
The deck of cards is composed of:

1 - A title card

2 - A 'control ROM' card for the module code identification

3 - The data cards
The following general format is applicable to the data field:

- A data field must start with the most significant bit and end with the least significant bit.

- A data field should consist of "0" and "1" (binary).

The required format is detailed in the following tables and Fig.43A, B,C.

M382 PROGRAMMING FORMAT

A - FIRST CARD (TITLE CARD)

It must have the following format (see Fig.43A):

Column	Data
1÷4	Punch "M382"
4÷10	Blank
11÷16	6 alphanumeric characters that should identify the contents. Must be repeated identically on all data cards.
47÷80	Blank

- SECOND CARD (CONTROL ROM)

It must have the following format (see Fig.43B):

Column	Data
1÷4	Punch "M382"
4÷10	Blank
11÷14	Punch "CRTL"
21÷26	6 alphanumeric characters that should identify the contents.
27÷30	Blank
31	Ø to address ROM-low range (Ø÷1Ø23 address) 1 to address ROM-high range (1Ø24÷2Ø47 address)
31÷40	Blank
41÷42	Specify the module code assigned to the ROM (decimal number Ø÷63)
43÷50	Blank
51÷52	Specify the module code assigned to the I/O Port n° 1
53÷60	Blank
61÷62	Specify the module code assigned to the I/O Port n° 2
63÷80	Blank

B - DATA CARDS (1024 CARDS)

They must have the following format (see Fig.43C):

Column	Data
1÷4	Punch "M382"
4÷10	Blank
11÷14	Punch "Main"
15÷20	Blank
21÷26	6 alphanumeric characters: must be the same that have been punched on title card in column 11÷16
27÷30	Blank
31÷40	Ten binary bits (Ø or 1) that represent the binary address of the ROM. They must be ordered from 0000000000 to 1111111111 (decimal Ø to 1Ø23)
41-5Ø	Blank
51÷58	Output word 51 out 1 58 out 8
59÷72	Blank
73÷76	Decimal count of cards and address (from Ø to 1Ø23)
77÷80	Blank

M381 PROGRAMMING FORMAT

A - FIRST CARD (TITLE CARD) AND DATA CARD

Same as "M382" except column 1÷4 that must have "M381" instead of "M382" on both title and data cards. In addition, it should be noted that the maximum address is 1011111111 (Decimal 767).

B - CONTROL ROM

Same as "M382" except column 1÷4 that must have "M381" instead of "M382".

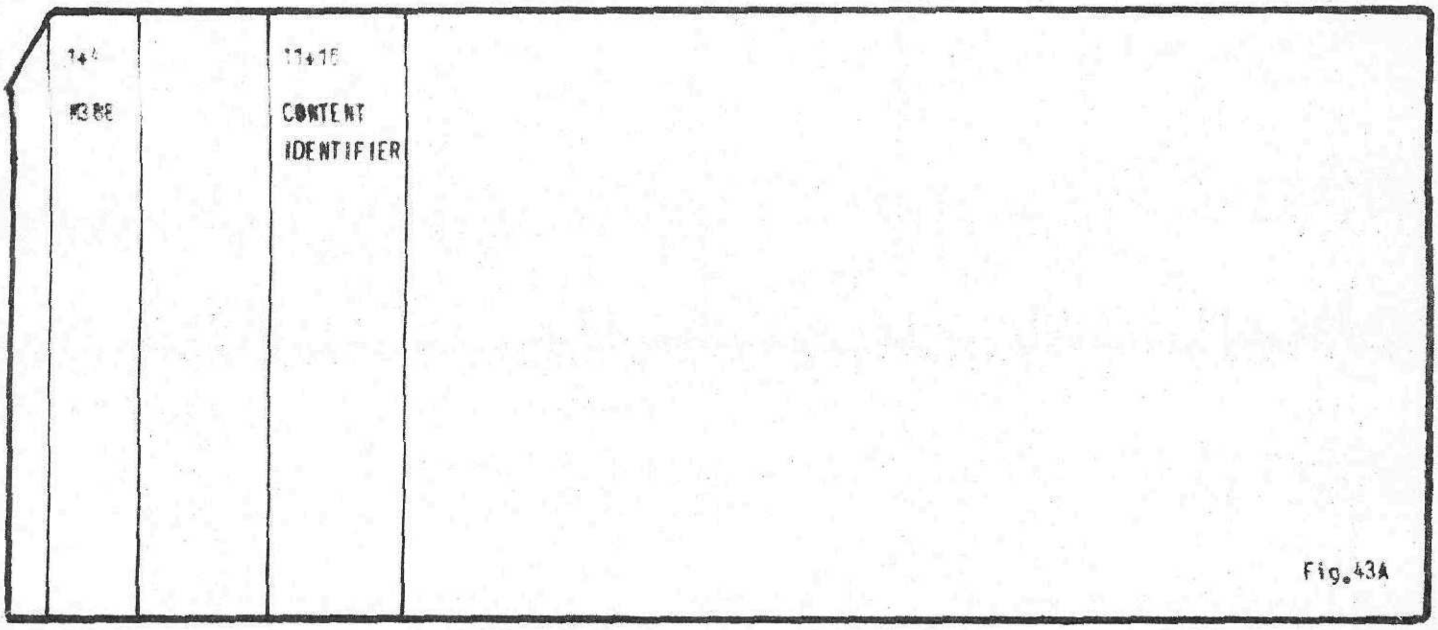


Fig.43A

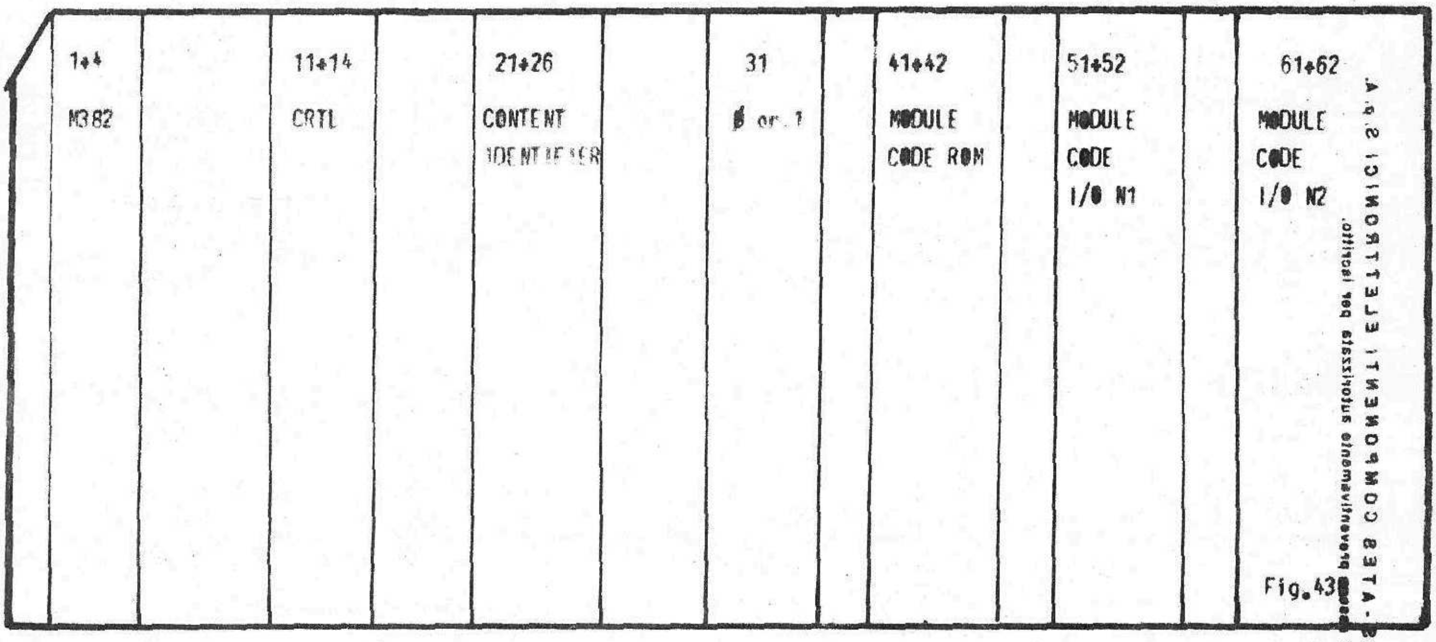


Fig.43B

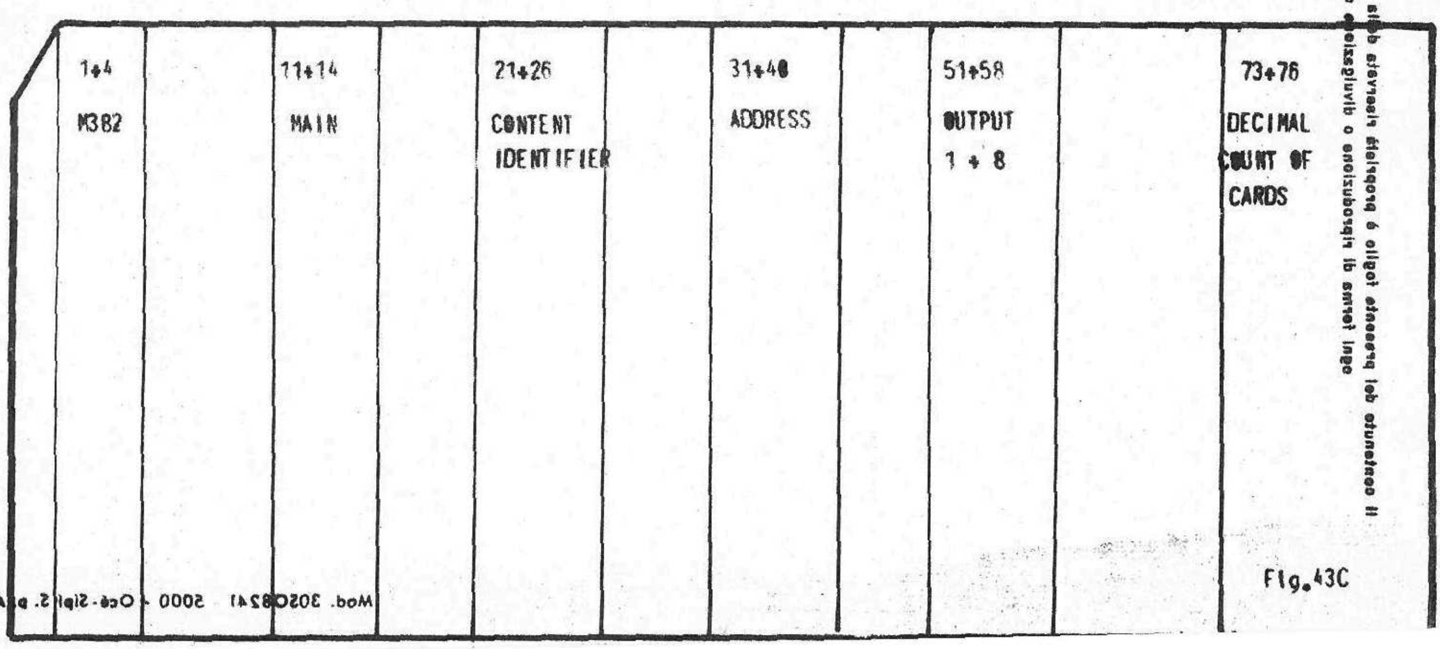


Fig.43C