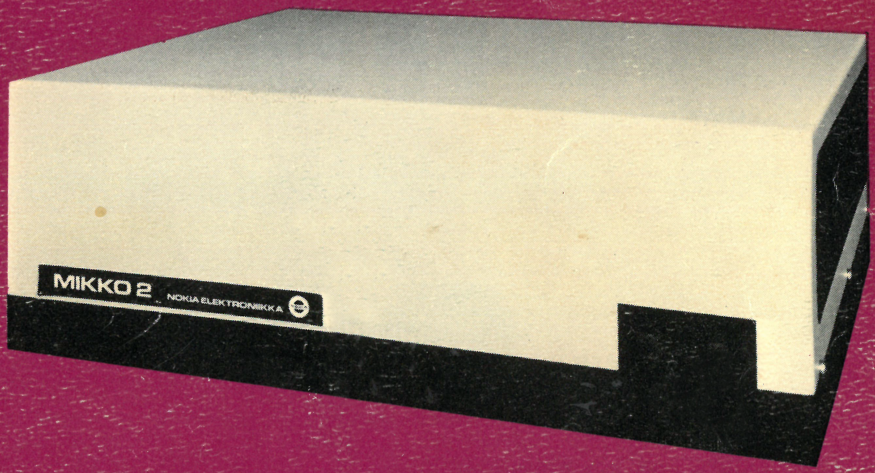
MIKKO 2

PROCESSOR HANDBOOK



NOKIA

MIKKO 2

PROCESSOR HANDBOOK

Ref.: Ø8.5Ø.Ø7/EO, Vers. B

OY NOKIA AB
ELECTRONICS

00101 HELSINKI 10
Finland
P.O. Box 780
Telephone 90 - 59 131
Telex 12-2062 eleno sf
Telegrams ELECTRONOKIA

First edition March 1976

(PR - AMI - tå)

Printed by MARTINPAINO

CONTENTS

1. INTRODUCTION

This guide is intended to briefly introduce what a user would need to know about the machine's construction and instruction set.

More detailed information and the operation of peripherals have been excluded. The intention is to provide a guide for system and application programmers. Only a few examples are given. An education in electronic data processing is a prerequisite for anyone to understand MIKKO 2 solely on the information presented in this book. The book is, for example, useful as a framework for a programming course.

The second section lists the main features of the CPU and peripheral interfaces. The microprocessor within the CPU and its comprehensive interrupt handling are covered in depth. The third section describes the machine's microprogrammed basic instruction set.

2. THE MINICOMPUTER MIKKO 2

The MIKKO 2 is a minicomputer which uses a word length of 16 bits and contains a microprogrammed memory. It is designed for the flexible use of interface units and a medium execution speed. It is equipped with a versatile interrupt handling system, so its applications range from process control to real time data processing.

2.1 The CPU

The CPU of MIKKO 2 consists of two basic parts: the microprogram processor and the arithmetic unit. These are situated on one printed card and they form one combined unit. In order to achieve a maximal handling speed the basic parts work parallel serving as one synchronous unit.

The asynchronous bus concept makes it possible to connect units with different cycle speeds into one entity functioning with one cycle speed. At the same time it guarantees expandability which is not limited by the lengths of the data paths.

2.1.1 THE MICROPROGRAM PROCESSOR

The centre of the MIKKO 2 is the processor built around the microprogram memory. This processor controls clocks and drives the functions of all the other units. The microprogram processor contains the following:

1. The Microprogram Memory: a fast semiconductor memory (access time 80 ns) the size of which in the basic version is 768 x 24 bits. It contains the microprogram procedures needed by the machine instruction set and for interrupt handling.

2. Microprogram Logic: a logic unit containing two address registers, an instruction register, a status register, a microprogram register, an address multiplexer and their control units.

3. Interrupt Control Logic: a unit which controls the microprogram logic handling interrupt request of different levels.

4. Test Logic: a logic which tests the internal status of the CPU.

5. Control Logic: contains a combination logic and a register, which controls the control bus and the arithmetic unit.

The basic instruction set (see chapter 3) is microprogrammed as a subroutine for the CPU, which can be divided according to its functions into the following groups:

- instruction search routine
- address counting routine
- routines belonging to a specific instruction
- interruption routines

Operation codes of machine instructions and interrupt requests given to the microprocessor for processing are subroutine calls. While processing subroutines the microprocessor produces data which controls the other units. A part of its function is to generate new tasks for itself so that functioning continues uninterruptedly. Figure 1 shows the interaction between the microprocessor and the parts of the system surrounding it.

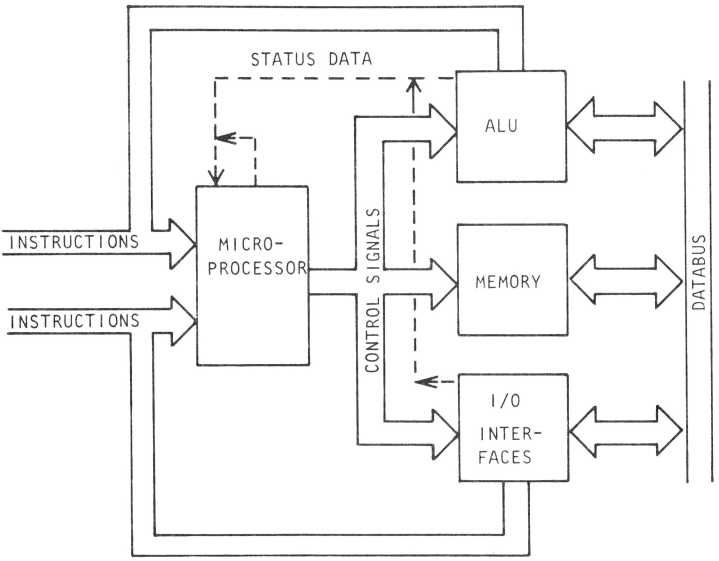


Figure 1

2.1.2 THE ARITHMETIC UNIT

The arithmetic unit contains a 16 bit binary parallel arithmetic unit, 8 working registers, the data multiplexers and the interface logic of the data multiplexers and the interface logic of the databus.

The arithmetic unit is a fast adder which functions on the "Carry Look ahead" principle and performs the following basic functions:

A + B	addition
A - B	subtraction
A · B	and-function
A ⊕ B	exclusive or -function

The working registers are multiport registers and are interconnected like the arithmetic unit, by an internal ALU bus and by data multiplexers. The arithmetic has been designed for easy byte handling.

The eight working registers of the machine are identical but the microprogrammed instruction set gives each of them a specific use:

- A, B — the working registers for arithmetic and logic calculation
- I, K — index registers for address calculation
- PR — the program counter
- LR — the address calculation register
- APU — a scratchpad register
- DMR — a register for microprogrammed memory channel functions

2.2 THE MAIN MEMORY

The standard memory of the machine is a semiconductor memory (MOS) the complete size of which is 64K x 8 bits. The memory consists of independent 16K x 8 bit modules of which there may be 4. The memory is so organized that the CPU can transfer data into the memory both in words and in bytes (2 bytes = 1 word). The basic component of the memory is a 1 bit dynamic random access memory RAM.

The execution times of the memory are the following:

read cycle time	500 ms
write cycle time	700 ns
read access time approx.	250 ns

The speeds are the same for byte or word transfer. The memory is protected from power-outs by a standby power supply unit connected to the main one.

2.3 THE PERIPHERAL INTERFACES

The following peripheral interfaces are available:

1. The synchronous line controller SLC-M2

   SLC-M2 is a high speed line controller for communication via a synchronous data channel. The unit makes MIKKO 2 suitable for terminal control and data concentration applications.

   SLC-M2 is designed to be available in applications involving different line procedures, i.e.

   - procedures based on control characters (BSC, ECMA, etc)
   - HDLC
   - SDLC
   - DDCMP

SLC-M2 is based on a microprocessor and incorporates following features:

- transmission (direct memory access) speed up to 1 M Baud,
- double buffered receive and transmission with automatic buffer switching,
- full- or half-duplex operation,
- data set control,
- auto answering capability,
- control character recognition and sequence control,
- automatic address recognition capability,
- automatic BCC (Block Check Character) generation and error detection,
- selectable BCC generation,
- transparent mode with automatic insertion and stripping of DLE characters,
- test loops through SLC-M2 and modem

2. The universal line interface ULC-M2

The ULC-M2 is a universal line interface for full- or half-duplex communication between a variety of serial communication devices and the MIKKO 2 computer.

With the ULC-M2 interface a MIKKO 2 computer can communicate with a local terminal, a remote terminal via data sets and private line or switched telephone facilities or with another local or remote MIKKO 2. The data transfer rate is in asynchronous mode from 110 bps up to 9600 bps and up to 56.000 bps in synchronous mode. Several ULC-M2's can be handled by MIKKO 2, each interface consisting of 8 channels.

For each channel in the ULC-M2 there is a status and configuration register. The receiver and transmitter data buffer registers are shared by the eight channels through an internal multiplex method.

3. Cassette Tape Recorder Interface

An interface which controls two DC-300 type digital cassette tape recorders and uses a microprogrammed memory channel. The specifications are according to ANSI proposals:

- bit transfer rate 24.000 bit/s or 48.000 bit/s
- phase modulation
- serial recording
- automatic CRC-check
- block length is specifiable in the program
- automatic load-point search
- automatic writing of the interrecord gaps

4. The Teletype Console Interface

A normal half-duplex current loop interface for the teletype ASR 33 of 110 bauds.

5. The Card Reader Interface (M-600L)

A card reader the speed of which is 600 cards/min.

6. The Alphanumeric Keyboard

7. The Function Keyboard and the Semiconductor Display Interface

2.4 INTERRUPTIONS

The central processor recognizes 4 different types of interrupts. They are in order of priority as follows:

- power fail
- console interrupts
- microprogrammed memory channel interrupts
- device interrupts

2.4.1 INTERRUPT TYPES AND THEIR PRIORITIES

Power Fail

If the mains power fails, a power fail interrupt is generated. When this happens the registers are automatically stored in a stack and at the same time the main memory is protected by switching on the reserve power pack. When the mains power returns the registers are restored.

Console Interrupts

The storing of words into memory, examination of the contents of the memory and the registers and the starting and interruption of programs may be performed using the console.

Memory Channel Interrupts

The microprogrammed memory channel interfaces generate interrupts by which the CPU transfers data between the memory and the interface. It also updates the address and character counters used when transferring data without a program interrupt.

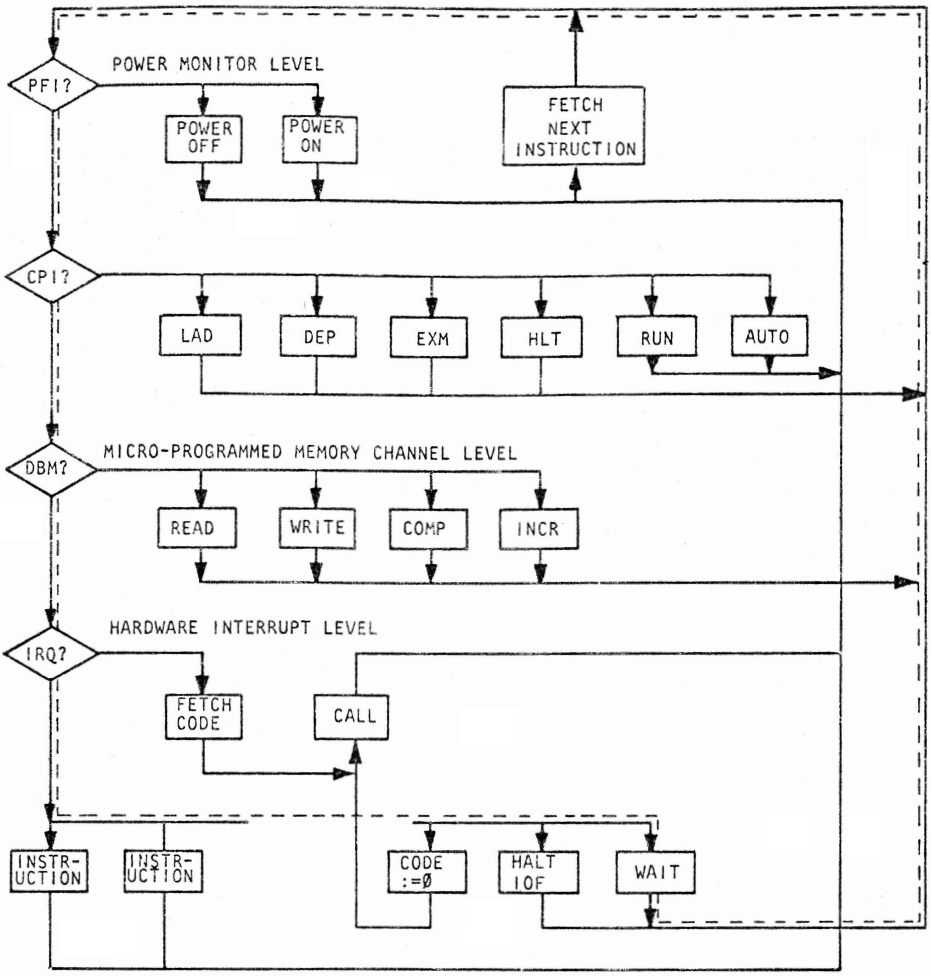

POWER MONITOR LEVEL
PFI?
POWER OFF
POWER ON
FETCH NEXT INSTRUCTION
CPI?
LAD
DEP
EXM
HLT
RUN
AUTO
DBM?
MICRO-PROGRAMMED MEMORY CHANNEL LEVEL
READ
WRITE
COMP
INCR
IRQ?
HARDWARE INTERRUPT LEVEL
FETCH CODE
CALL
INSTR-UCTION
INSTR-UCTION
CODE :=Ø
HALT IOF
WAIT

Figure 2

Device Interrupts

The microprograms serving the device interrupts are able to identify 256 different interrupt requests. The identification is done through the interrupt number generated by the interrupting unit. When an interrupt occurs the registers are stacked, after which the interrupt number is transferred into the working register A. The return from an interrupt is performed by using the instruction RFI (see instruction set) which restores the registers.

The priority levels and the branching to these levels are described in Figure 2, which represents the functioning of the microprogram processor.

If the microprogram processor is idle, it performs the loop marked by a broken line in the figure, which it can leave only through an interrupt. This interrupt can be among others the console RUN interrupt, the result of which on the next round is a branch to the console interrupt level. From there on it follows the RUN routine, which is followed by the microprocessor ending in a loop where it fetches the following machine instruction by itself. If during this round no interrupt request has been received, we come to machine instruction level where branching occurs according to the operation code of the instruction. It is to be noted from the figure that once in the interrupt inhibit state, only device interrupt requests are prevented from arriving at the microprogram processor. If such requests do arrive, the microprogram processor is ready to handle them.

2.4.2 STACKING AND OTHER FUNCTIONS CAUSED BY INTERRUPTS

Some words in the main memory have been dedicated to the handling of interrupt functions:

- Ø stack address
- 2 address of the interrupt handler
- 4 address of the power fail interrupt handler
- 6 address of the line controller device table
- 8 address of the cassette controller device

There are also some words used by the microprograms (character counters, pointers) in the line and cassette tables. The rest of the memory can be used freely.

Upon the arrival of an interrupt necessitating the stacking of the registers (power control device interrupts and software interrupts, i.e. switching from one module to another, see instruction set, CALL) the following procedures are followed:

1. The registers A, B, I, K, the carry bit and the PR are taken to memory locations which are located just after the location given by the stack pointer (stack pointer + 2, stack pointer + 4, etc.).

2. The stack pointer is increased by 12, making it point at the surface element.

3. If a device interrupt occurs, the interrupt code is loaded into register A.

4. The interrupt controller is set to the interrupt-inhibit state. Thus no further interrupts will arrive.

5. The contents of memory word 4 are loaded into the PR-register if a power control interrupt occurs, otherwise the contents of the memory word 2. Execution then continues from this address.

The return from an interrupt is achieved by using the machine instruction RFI (except power control interrupt), which results in the following:

1. Values from the location indicated by the stack pointer are loaded into registers PR, CARRY, K, I, B and A.

2. The interrupt inhibit state is removed.

3. The execution of instructions continues from the location indicated by the PR.

Example: The state of the memory being (in hexadecimal):

Address	Contents	
Ø	572A	stack pointer
2	0404	address of the interrupt handler
:	:	
404	17Ø3	beginning of the interrupt handler
4Ø6	Ø502	
:	:	
1232	2Ø5Ø	machine instruction last executed
1234	4Ø6Ø	machine instruction to be executed
:	:	
572A	2244	surface element of stack
572C	3536	
:	:	

Let the contents of the registers be the following:

A = 5792, B = 2249, I = 5797, K = 0000, CARRY = 1 and PR naturally 1234. If now a device interrupt having the code 81 arrives, the situation is the following upon arrival at the interrupt handler:

0	5736		
2	Ø4Ø4		
:			
4Ø4	17Ø3		
4Ø6	Ø502		
:			
1232	2Ø5Ø		
1234	4Ø6Ø		
:			
572A	2244		
572C	5792	A	register values stored into the stack
572E	2249	B	
573Ø	5797	I	
5732	0000	K	
5734	0001	C	
5736	1234	PR	
5738	:		
:	:		

The contents of the registers are now:

A = 0081, PR = ØØØ4, the rest remaining as they were. When returning from the corresponding interrupt for example at the end of the interrupt handler the old values are loaded from the stack to the registers:

P: = 1234, CARRY: = 1, K: = 0000, I: = 5797, B: = 2249 and A: = 5792. The stack pointer (memory location Ø) is updated correspondingly to be 572A. Thus the value of register P becomes 1234 and the execution of the interrupted program goes on from there. Both stacking and unstacking are performed during one "cycle" of the microprocessor. Thus they are not interrupted by an interrupt request of higher priority.

2.4.3 THE FUNCTION OF THE CONSOLE

The console is a peripheral connected to the CPU with which one may examine the contents of the memory and the registers, write words into the memory and start and interrupt programs. The visible part of the console consists of one signal light, two displays of 4 decades, a choice of display and the input control keyboard (see Figure 3).

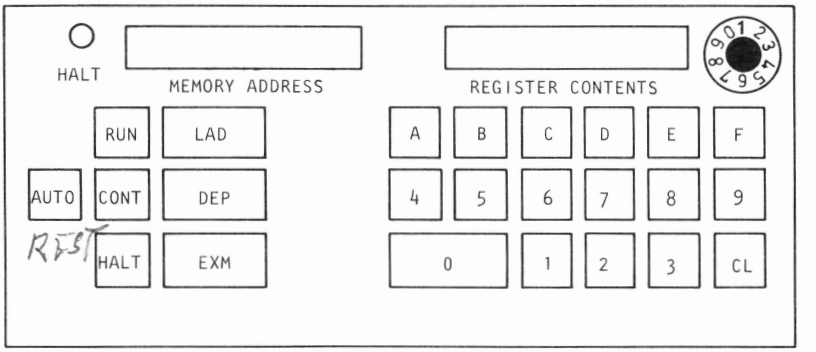


Figure 3

Displays

The HALT display shows whether the microprogram processor is executing a HALT loop or handling an interrupt (see microprogram processor).

The MEMORY ADDRESS display shows the contents of the PR during a HALT loop; otherwise it shows the contents of the LR.

The REGISTER CONTENTS display shows the contents of the element given by the selector.

Selector Display

The element to be displayed by the REGISTER CONTENTS can be chosen by the display selector in the following way:

Ø contents of register A
1 contents of register B
2 contents of register I
3 contents of register K
4 contents of register PR
5 contents or register LR (not in use during the HALT loop)
6 contents of register APU
7 contents of the memory word given by the PR during a Halt loop, otherwise contents of register DMR
8 locks the console, i.e. inhibition of interrupt requests from the console

The Input Keyboard

Data is entered into a buffer of 4 hexadecimal numbers using the keyboard. The keys represent one hexadecimal number each, with the exception of the CL key which clears the buffer.

The Control Keys

The use of a control key generates an interrupt request at console level to the microprogram processor. The functions of these keys are:

LAD — Load Address; the contents of the input buffer are loaded into the PR and the microprogram processor goes into a HALT loop.

DEP — Deposit; the contents of the input buffer become the contents of the memory word given by the PR and the machine remains in a HALT loop.

EXM — Examine Next; the contents of the PR are increased by two, the machine remains in a HALT loop.

RUN — execution is started from the address in the PR.

CONT — Continue; execution of the machine instruction pointed to the PR, after which the machine remains in a HALT loop (the contents of the PR are incremented appropriately).

HALT — The microprogram processor goes into a HALT loop and the contents of the PR are decreased by two.

REST — Restart, which causes the microprogram processor to move a fixed autostart memory from the console to the beginning of the main memory. After this the microprogram processor starts the execution of the relocated program from address Ø on.

If the keys CL and REST are used simultaneously the main memory is cleared.

2.5 THE USE OF THE PERIPHERALS

There is no special instruction in the instructions set for the use of the peripherals. The communication between the device interfaces and the program is achieved through a so-called device page. The device page is separate from the main memory and it can be used by both the device interfaces and the software by using so called device page sharing (see instruction set). The device page contains 256 bytes used for status and I/O data for the devices. In addition to this there are 256 one bit device flags and device masks for the control of the devices. The device flags indicate, depending on the device, the existence of interrupt requests or the status of the devices. An arriving interrupt is acknowledged by for example clearing the flag, depending on the device in question. By setting a mask the sending of a corresponding interrupt request to the central unit is inhibited. Correspondingly the clearing of the mask permits the sending of interrupt requests.

The peripheral interfaces can be divided into three classes depending on their manner of operation:

- interfaces using cycle stealing memory channel transfer (DMA)
- interfaces using microprogrammed memory channel transfer
- interfaces using character transfer

The use of DMA devices:

The devices using microprogrammed memory channel transfer function in the following manner when transferring data:

1. The user sets values showing to or from which memory location the data is transferred into certain registers on the device page.

2. The user transfers his instruction, for example read, write, etc. into a certain position in the device page.

3. From here the transfer takes place directly from memory/into memory so that the interface takes the internal device bus-line for its own use and controls the whole transfer between memory and interface. The transfer may take place in bytes or in words according to

the interface requirements. Every device which can reserve the bus-line has a certain priority. The device which has the higher priority can take bus control from that which has the lower priority, but only after the other has finished its transfer sequence.

4. When the function has been completed the interface sends a device interrupt (assuming that the mask in question has been cleared).

The devices using microprogrammed memory channel transfer function in the following manner when transferring data:

1. The user sets values showing from which buffer/to which buffer the data is transferred into certain device tables also used by the microprogram. He also gives the amount of data to be transferred etc.

2. The user transfers his instruction, for example read, write, etc. into a certain position in the device page.

3. From here the transfer takes place directly from memory/into memory so that after each transfer of a character the interface sends a microprogrammed interrupt at memory channel level. When this interrupt is sent the micro-software updates the necessary counters and pointers and checks if the function wanted has been completed.

4. When the function has been completed or if an execution error has occured the interface sends a device interrupt (assuming that the mask in question has been cleared).

5. The user acknowledges the interrupt, reads the cause of the interrupt from the interrupt code of the status data of the device and executes the procedures necessary for continuing processing. If serving the interrupt takes so much time that the machine cannot be held in the interrupt inhibit state for the whole time, the user should set the corresponding interrupt mask for the duration of the interrupt condition.

Character oriented device operation is very much less complicated. The input to the machine is executed in the following way:

1. When sending a character, the device sends an interrupt request (if this has not been inhibited by an interrupt inhibit state or by a mask). Now the character can be read from a certain location in the device page.

2. The user transfers the character from the device page into a location that he has chosen and acknowledges the interrupt. If no further characters are wanted from the device the corresponding interrupt mask is set. Otherwise the next character is awaited.

The output to the device is as follows:

1. The user transfers the character to be output into a certain location in the device page.

2. After receiving the character, the device sends an interrupt to signal its readiness to receive the next character.

In general the devices also produce a signal of unsuccessful transfers and other disturbances by interrupts. The necessary procedures must in these cases be executed by the user.

3. MIKKO 2 INSTRUCTION SET B

The B instruction set has been realized using microprogramming so it is easily adaptable to the user's needs. The instruction set is byte oriented.

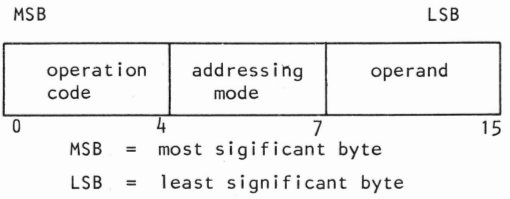
3.1 MEMORY REFERENCE INSTRUCTIONS

Memory reference instructions perform functions between the registers and memory words or bytes. The addressable memory field depends on the memory size in use; the largest possible memory stretches from Ø-$FFFF_{16}$.

As already mentioned in the previous chapter, memory addresses Ø...9 are dedicated to special uses.

3.1.1 INSTRUCTION FORMAT

The length of a memory reference instruction is 2 bytes (16 bits). The bits Ø...4 of the instruction contain the operation code, 5...7 the addressing mode and the rest (8...15) contain the operand.



MSB = most sigificant byte

LSB = least significant byte

3.1.2 ADDRESSING MODES

When studying the memory addressing instructions it is to be noted that a MIKKO 2's page is dynamic. (A page is the area which can be referred without using index registers or indirect addresses.)

The address given in the lower byte of the machine instruction is relative to the value of the address counter. The addressable area is thus 128 bytes either side of the instruction being executed.

This feature frees the programmer from the difficulties caused by fixed pages. On the other hand the programmer must remember the interpretation of the addresses when using, for example, combinations of indexed addressing and symbolic references.

The addressing modes of the memory reference instructions (codes ØØ...BF) are as follows:

Ø Direct Relative Addressing
The effective address is found by adding the effective address (d) to the PR register's contents.

E.A. = PR + d (where $-128_{10} \leq d \leq 127_{10}$)

The only exception is with the JMP instruction when:

E.A. = PR + d when the instruction code is Ø3
E.A. = PR - d when the instruction code is Ø7 (where $0 \leq d \leq 255_{10}$)

1 Address indexed from I
The effective address is found by adding the instruction address to the contents of the I register.

E.A. = I + d (where $-128_{10} \leq d \leq 127_{10}$)

2 Address indexed from K
The effective address is found by adding the instruction address to the contents of the K register.

E.A. = K + d (where $-128_{10} \leq d \leq 127_{10}$)

3 The addressing of devices

The last byte of the instruction gives the address of the device question. This is the only way of addressing a device.

E.A. = d (where $0 \leq d \leq 255_{10}$)

4 Indirect address

The effective address is found in the memory address which is found by adding the instruction address to the contents of the PR register.

E.A. = [PR + d] (where $-128_{10} \leq d \leq 127_{10}$)

5 Indirect address from I

The effective address is found by adding the I register contents to the contents of the memory address found by adding PR to the instruction address (d).

E.A. = [PR + d] + I (where $-128_{10} \leq d \leq 127_{10}$)

An exception occurs with a JMP instruction (Ø5).

E.A. = [PR + d + 2I]

6 Indirect address from K

The effective address is found by adding together the contents of the K register and the contents of memory address PR + d

E.A. = [PR + d] + K (where $-128_{10} \leq d \leq 127_{10}$)

An exception occurs on JMP instruction Ø6, where:

E.A. = [PR + d + 2K]

7 Immediate addressing

The instruction's last byte is interpreted as an operand.

E.A. = PR + 1

3.1.3 LOCATION OF THE OPERAND OR INSTRUCTION IN MEMORY

If the instruction or operand is one word long (2 bytes), the effective address gives the location of the highest byte. The low byte goes into next address above the effective address. When the effective address is calculated, the value of the Program Counter PR gives the location of the highest byte.

3.1.4. ADDRESS CALCULATION REGISTERS

The address calculation and fetch routines (microprograms) use the LR and APU registers, the effective address being always loaded into the LR register. It is possible to use the control panel to check the last memory reference instruction's effective address by examining the LR register.

After writing to memory, the bytes from the memory loading instruction are sought from the effective memory addresses' contents and loaded into the APU register. In the memory storage command the bytes are written into the effective address. Because memory read and write is byte-oriented, after the execution of the instruction the LR register contains the address of the last byte.

3.1.5 OPERAND REGISTERS

Registers A, B, I, K and APU are also used in arithmetic and logic operations. The operand located in the memory in byte form, is shifted in this form into the APU register. The arithmetic operation proper is executed as a micro-operation between the working register and APU register.

3.1.6 INSTRUCTIONS

The memory reference instruction's first five bits comprise the instruction while the last three give the mode of addressing. In the following section the operational code is marked with an x if the fifth bit is zero and y if one.

For example instruction INCB, direct addressing from register I, is coded 11; instruction code DECB, indirect addressing from register K, is 1E. For some memory reference codes all addressing modes do not work. This would happen because these codes are microprogrammed to another instruction.

Øx JMP JUMP

The effective address is loaded into PR. Note: the addressing modes for this instruction have special exceptions (ØØ is an invalid instruction code).

Øy JSR JUMP TO SUBROUTINE

The PR register's contents are loaded into register K and the stack. PR is loaded with the effective address. The stack pointer is raised by two.
Note: ØB PUSA and ØF RFS*

1x INCB INCREMENT BYTE

The byte stored in the effective address is increased by one. The registers remain unchanged.
Note: 13 PUSB and 17 SPZA*

1y DECB DECREMENT BYTE & SKIP 2 IF POSITIVE

The byte stored in the effective address is decreased by one. The registers stay unchanged. If the result is not negative, the program skips two bytes.
Note: 1B PUSI AND 1F SPZB*

2x LWA LOAD WORD TO A

The word stored in the effective address is loaded into accumulator A, whose previous contents are destroyed.
Note: 23 PUSK*

2y LWB LOAD WORD TO B

The word stored in the effective address is loaded into accumulator B, whose previous contents are destroyed.
Note: 2B POPA*

*See register reference instructions.

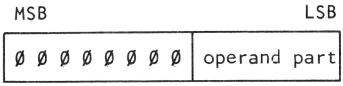
3x LWI LOAD WORD TO I

The word stored in the effective address is loaded into index register I, whose previous contents are destroyed.
Note: 33 POPB*

3y LWK LOAD WORD TO K

The word stored in the effective address is loaded into index register K, whose previous contents are destroyed.
Note 3B POPI*

LWA, LWB, LWI or LWK together with the immediate operand gives a word whose high byte is zero, while the low byte takes the value of the next byte of operational code



The immediate operand

4x LBA LOAD BYTE TO A

The byte stored in the effective address is loaded in as the low byte of accumulator A. The high byte is set to zero.
Note: 47 NOP*

4y LBB LOAD BYTE TO B

The byte stored in the effective address is loaded as in the low byte of accumulator B. The high byte is set to zero.
Note: 4F POPK*

5x SWA STORE WORD FROM A

The contents of accumulator A are written into the effective address. The acculator's contents remain unchanged.
Note: 53 ADAB and 57 ADBA*

*See register reference instructions.

5y SWB STORE WORD FROM B

The word in accumulator B is written into the effective address. The accumulator's contents remain unchanged.
Note: 5B and 5F are invalid instruction codes.

6x SBA STORE BYTE FROM A

The low byte in accumulator A is written into the effective address. The accumulator's contents remain unchanged.
Note: 67 BISA*

6y SBB STORE BYTE FROM B

The low byte in accumulator B is written into the effective address. The accumulator's contents remain unchanged.
Note: 6F BISB*

7x ADDA ADD WORD TO A

The word in the effective address is added into the contents of accumulator A. An immediate operand is taken as positive ($0\text{-}255_{10}$). Overflow sets the Carry-flip-flop.
Note: 73 BICA*

7y ADDB ADD WORD TO B

The word in the effective address is added into the contents of accumulator B. An immediate operand is taken as positive ($0\text{-}255_{10}$). Overflow sets the Carry-flip-flop. Note 7B BICB*

8x SUBA SUBTRACT WORD FROM A

The word in the effective address is subtracted from the contents of accumulator A. An immediate operand is regarded as positive. Overflow sets the Carry-flip-flop.
Note: 83 CLRM*

8y SUBB SUBTRACT WORD FROM B

The word in the effective address is subtracted from the contents of accumulator B. An immediate operand is regarded as positive. Overflow sets the Carry-flip-flop.
Note: 8B invalid instruction code.

*See register reference instructions.

9x ANDA AND A TO MEMORY WORD

An "and" function between the word in the effective address and the contents of accumulator A. The immediate operand's highest byte is set to FF.

Note: 93 SETM*

9y ANDB AND B TO MEMORY WORD

An "and" function between the word in the effective address and accumulator B. An immediate operand is treated the same as before.

Note: 9B invalid instruction code.

Ax XORA EXCLUSIVE OR A & MEMORY WORD

An "exclusive or" function between the word in the effective address and accumulator A. The immediate operand's high byte is set to zero.

Note: A3, A8, A9, AA, AB, AC, AD, AE and AF are invalid instructions codes.

Bx CMPI COMPARE TO I, SKIP 2 IF I GREATER, 4 IF EQUAL

Index register I and the effective address contents are compared and the result controls a skip as follows:

I < M	No skip
I > M	Skip 2 bytes
I = M	Skip 4 bytes

The register contents are regarded as a 16 bit positive number. The contents of memory and register remain unchanged. An immediate operand is regarded as positive ($0\text{-}255_{10}$).

Note: B3 invalid instruction code.

By CMPK COMPARE TO K, SKIP 2 IF K GREATER, 4 IF EQUAL

The same as before, but with index register K.

Note BB invalid instruction code.

*See register reference instructions.

3.2 REGISTER REFERENCE INSTRUCTIONS

3.2.1 INSTRUCTION LENGTH

The register reference instruction may be one or two bytes long. In the following explanation, the letter after the instruction code gives the length: a for a byte and b for a word.

3.2.2 INSTRUCTIONS

ØBa PUSA — PUSH A TO STACK
Store register A onto systemstack.

ØFa RFS — RETURN FROM SUBROUTINE
The return address is loaded from the stack into PR. The stack pointer drops two.

13a PUSB — PUSH B TO STACK
Store register B onto systemstack.

17a SPZA — SKIP ON ZERO A
A jump of two bytes if A is zero.

1Ba PUSI — PUSH I TO STACK
Store register I onto systemstack.

1Fa SPZB — SKIP ON ZERO B
A jump of two bytes if B is zero.

23a PUSK — PUSH K TO STACK
Store register K onto systemstack.

2Ba POPA — POP A FROM STACK
Load register A from systemstack.

33a POPB — POP B FROM STACK
Load register B from systemstack.

3Ba POPI — POP I FROM STACK
Load register I from stack.

47a — NO OPERATION
The contents of register PR are increased by one.

4Fa POPK — POP K FROM STACK
Load register K from stack.

53a ADAB — ADD A TO B
Add contents of registers A and B together and load the result into B.

57a ADBA — ADD B TO A
Add contents of registers A and B together and load the result into A.

67b BICA — BIT CLEAR A
Clear one bit in A. The bit index is defined in the following byte.

6Fb BICB — BIT CLEAR B
Clear one bit in B. The bit index is defined in the following byte.

73b BISA — BIT SET A
Set one bit in A. The bit index is defined in the following byte.

7Bb BISB — BIT SET B
Set one bit in B. The bit index is defined in the following byte.

83 CLRM — CLEAR MASK
Enable interrupts from device specified in register A.

93 SETM — SET MASK
Inhibit interrupts from device specified in register A.

C0b ADDI — ADD IMMEDIATE OPERAND TO I

The immediate operand, understood as positive ($0\text{-}255_{10}$), is added into index register I.

C1b ADDK — ADD IMMEDIATE OPERAND TO K

Same as before, but for index register K.

C2b SUBI — SUBTRACT IMMEDIATE OPERAND FROM I

The immediate operand, again understood as positive ($0\text{-}255_{10}$), is subtracted from the contents of index register I.

C3b SUBK — SUBTRACK IMMEDIATE OPERAND FROM K

Same as before, but for register K.

C4a CTOA — LOAD CARRY TO A

Load A with a zero or a one depending on the state of the carry. The accumulator's original contents are destroyed.

C5a CTOB — LOAD CARRY TO B

Load B with a zero or a one depending on the state of the carry. The accumulator's original contents are destroyed.

C6...CA — See Macroinstructions.

CBb TSTF — TESTFLAG AND SKIP 2 IF ON

This tests the peripheral device flag, found at the given address. If the flag has been set, the program skips two bytes.

CCb CLRF — CLEAR FLAG

Resets the addressed FLAG flip-flop to zero

CDb STF — SET FLAG

Sets the addressed FLAG flip-flop to one.

Note: CE and CF are invalid instruction codes.

D0a CLRC — CLEAR CARRY
Resets the carry to zero.

D1a SETC — SET CARRY
Sets the carry to one.

D2a ATOI — LOAD A TO I
Loads the contents of accumulator A into register I.

D3a ATOK — LOAD A TO K
Loads the contents of accumulator A into register K.

D4a CLRA — CLEAR A
Resets accumulator A to zero.

D5a CLRB — CLEAR B
Resets accumulator B to zero.

D6a BTOI — LOAD B TO I
Loads the contents of accumulator B into register I.

D7a BTOK — LOAD B TO K
Loads the contents of accumulator B into register K.

D8a ITOA — LOAD I TO A
Loads the contents of register I into accumulator A.

D9a ITOB — LOAD I TO B
Loads the contents of register I into accumulator B.

DAa CLRI — CLEAR I
Resets register I to zero.

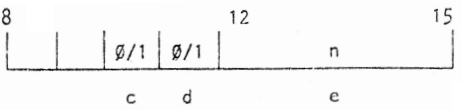
DBa CLRK — CLEAR K
Resets register K to zero.

DCa KTOA — LOAD K TO A
Load the contents of register K into accumulator A.

DDa KTOB — LOAD K TO B
Load the contents of register K into accumulator B.

DEb TSTA TEST A

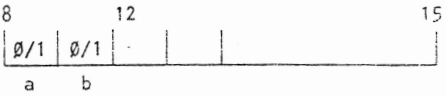
Examines the contents of carry and/or accumulator A and skips two bytes if the test is positive. The test itself is specified by the next byte.



c. Bit test no/yes
d. Bit false/true test
e. Index of bit to be tested

DEb TSTC TEST CARRY

Examines the contents of carry bit and skips two bytes if the test is positive. The test itself is specified by the next byte.



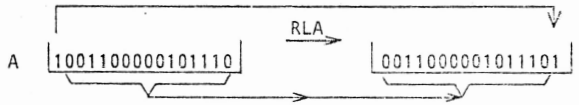
a. Carry test no/yes
b. Carry false/true test

DFb TSTB TEST B

As above but for the B accumulator.

EØa RLA ROTATE A ONCE TO LEFT

A is rotated one bit to the left.



E1a RLB ROTATE B ONCE TO LEFT

B is rotated one bit to the left.

E2a RL3A ROTATE A THREE TIMES TO LEFT

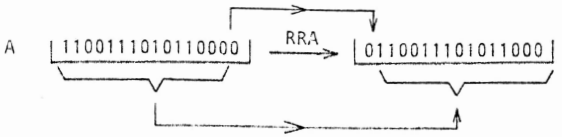
A is rotated three times to left.

E3a RL3B ROTATE B THREE TIMES TO LEFT

B is rotated three times to left.

E4a RRA ROTATE A ONCE TO RIGHT

A is rotated once to the right.



E5a RRB ROTATE B ONCE TO RIGHT

B is rotated once to the right.

E6a RR3A ROTATE A THREE TIMES TO RIGHT

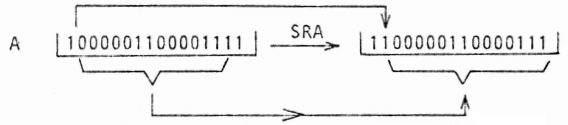
A is rotated three places right.

E7a RR3B ROTATE B THREE TIMES TO RIGHT

B is rotated three places right.

E8a SRA SHIFT A ONCE TO RIGHT

A is arithmetically shifted once right.



E9a SRB SHIFT B ONCE TO RIGHT

B is arithmetically shifted once right.

EAa SR3A SHIFT A THREE TIMES TO RIGHT

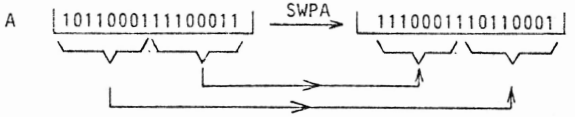
A is arithmetically shifted three places right.

EBa SR3B SHIFT B THREE TIMES TO RIGHT

B is arithmetically shifted three places right.

ECa SWPA SWAP BYTES IN A

The positions of low and high bytes in accumulator A are switched.



EDa SWPB SWAP BYTES IN B

The positions of low and high bytes in accumulator B are switched.

EEa CMPA COMPARE A TO B, SKIP 2 IF A GREATER, 4 IF EQUAL

A and B are compared and the result influences the skip as follows:

A < B	No skip
A > B	Two bytes are skipped
A = B	Four bytes are skipped

Numbers are regarded as containing 16 bits in the 2's complement system.

Note: EF invalid instruction code.

FØa COMA ONE'S COMPLEMENT OF A

The contents of A are inverted.

F1a COMB ONE'S COMPLEMENT OF B

The contents of B are inverted.

F2a NEGA TWO'S COMPLEMENT OF A

A is negated and overflow sets the CARRY.

F3a NEGB TWO'S COMPLEMENT OF B

B is negated and overflow sets the CARRY.

Note: F4, F5, F6, F7, F8, F9, FA and FB are invalid instruction codes.

FCa ION INTERRUPT ON

The processor is opened to interrupt requests.

FDa IOF INTERRUPT OFF

Interrupt requests are ignored.

FEa HALT — HALT, INTERRUPT OFF

The processor stops and ignores interrupts.

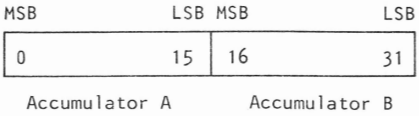
FFa WAIT — HALT, INTERRUPT ON

The processor stops and awaits interrupts.

3.3 MACRO INSTRUCTIONS

C6A MULT — MULTIPLY A AND B

Immediately before the instruction becomes effective the multiplier is in accumulator A and the multiplicand in accumulator B. The result is a 32 bit double word, which is generated on execution of the instruction. It is located in the accumulators as follows:



Multiplier and multiplicand are always taken as positive numbers not longer than 16 bits.

C7A DIV — DIVIDE A WITH B

Immediately before instruction execution the dividend is located in the accumulator A, while the divisor is in accumulator B. The result is 16 bits split over two locations. The number before the point goes into accumulator A while the fractional part goes into accumulator B. The mantissa on the remainder goes into the I register and may be useful if further accuracy is needed. In this case the remainder is moved into the accumulator A and another division occurs, giving a continuation of the previous result of more binary places.

Because the division also divides the divisor's leading zeroes, the accuracy of the result may in the worst case be only one significant ditit (when the dividend is one). Divisor and dividend are understood to be positive integers of length not greater than 15 bits.

C8a MOVE MOVE

The move instruction relocates a block of memory from an area whose first address is in the K register to an area whose first address is in the I register. The length of the block is found in the accumulator A. After execution K and I contain the sending and receiving addresses one after the last address in the areas affected and A is zeroed.

C9a CALL JUMP TO INTERRUPT ROUTINE

The software interrupt procedure. The instruction shifts the contents of A, B, I, K, CARRY and PR into the interrupt stack. The program jumps to the address given by address number 2 and continues with the word stored there. The machine is switched into its IOF state.

The interrupt stack pointer is stored in address number Ø and its value after execution gives the location of the last register's contents (PR). The pointer value may only be even so the CARRY-bit gets a word of space. The accumulator A is reset to zero, while the remainder stay unchanged.

CAa RFI RETURN FROM INTERRUPT

The opposite of the previous instruction. The contents of registers A, B, I, K, CARRY and PR are replaced from the interrupt stack, whose pointer is appropriately relocated. The machine goes back into ION state and the program continues from the location in the PR register.

3.4 INSTRUCTION SET AND EXECUTION TIMES

The following passage gives the basic instruction set of the machine. The performance times of each instruction (in microseconds) are given for when the processor is working at the speed given in chapter 2 (read , write and registerphase). The addressing modes of the memory reference instructions and the symbols x and y after the instruction symbol have the same meaning as before. The three numbers after the memory reference instruction give the execution time of the instruction when addressed directly, indirectly and using an immediate operand. The use of indexing does not influence the times given.

MEMORY REFERENCE INSTRUCTIONS

			Direct	Indirect	Immediate
Øx	JMP	JUMP	3,1 µs	7,0 µs	-
Øy	JMS	JUMP TO SUBR	17,3 *	17,9	-
ØF	RFS	RETURN FROM SUBR			
1x	INCB	op + 1	7,2	7,9	-
1y	DECB	op - 1, SKIP IF POS.	8,2	8,9	-
2x	LWA	A ← op	6,7	8,1	2,8
2y	LWB	B ← op	6,7	8,1	2,8
3x	LWI	I ← op	6,7	8,1	2,8
3y	LWK	K ← op	6,7	8,1	2,8
4x	LBA	A [8 - 15] ← op	6,6	7,3	-
4y	LBB	B [8 - 15] ← op	6,6	7,3	-
5x	SWA	A → op	9,1	9,7	-
5y	SWB	B → op	9,1	9,7	-
6x	SBA	A [8 - 15] → op	7,2	7,9	-
6y	SBB	B [8 - 15] → op	7,2	7,9	-
7x	ADDA	A ← A + op	7,1	8,5	3,1
7y	ADDB	B ← B + op	7,1	8,5	3,1
8x	SUBA	A ← A - op	7,1	8,5	3,3
8y	SUBB	B ← B - op	7,1	8,5	3,3
9x	ANDA	A ← A & op	6,7	8,1	2,8
9y	ANDB	B ← B & op	6,7	8,1	2,8
Ax	XORA	A ← A ⊕ op	6,7	8,1	2,8

* total time given because of the logical context

			Direct	Indirect	Immediate
Bx	CMPI	COMPARE I and op, SKIP IF...**	11	13	7,5
By	CMPK	COMPARE K and op, SKIP IF...**	11	13	7,5

REGISTER REFERENCE AND MACRO INSTRUCTIONS

17	SPZA	SKIP ON ZERO A**	2,7
1F	SPZB	SKIP ON ZERO B**	2,7
47	NOP	NO OPERATION	1,3
CØb	ADDI	I ← I + Im.op	2,8
C1b	ADDK	K ← K + Im.op	2,8
C2b	SUBI	I ← I - Im.op	2,8
C3b	SUBK	K ← K - Im.op	2,8
C4	CTOA	A ← C	1,3
C5	CTOB	B ← C	1,3
C6	MULT	AB ← A * B	40 (approx.)
C7	DIV	AB ← A/B (note 1)***	40...80
C8	MOVE	MOVE (note A, I, K)	2,2 + 4,5/byte
C9	CALL	JUMP TO INTERRUPT RTN	
CA	RFI	RETURN FROM INTERRUPT RTN	60 (approx.)
CB	TSTF	FLAGTEST, SKIP IF ON	4,5
CCb	RSF	F (op) ← Ø	3,1
CDb	STF	F (op) ← 1	3,1
DØ	CLRC	CARRY ← Ø	1,3
D1	SETC	CARRY ← 1	1,3
D2	ATOI	I ← A	1,3
D3	ATOK	K ← A	1,3
D4	CLRA	A ← Ø	1,3
D5	CLRB	B ← Ø	1,3
D6	BTOI	I ← B	1,3
D7	BTOK	K ← B	1,3

** mean time given in test instructions; the performance time depends on number of skips caused by the result of the test

*** depending on the operands of the division

D8	ITOK	A ← I	1,3	
D9	ITOB	B ← I	1,3	
DA	CLRI	I ← Ø	1,3	
DB	CLRK	K ← Ø	1,3	
DC	KTOA	A ← K	1,3	
DD	KTOB	B ← K	1,3	
DEb	TSTA	TEST A, SKIP IF ON	3,3	
DFb	TSTB	TEST B, SKIP IF ON	3,3	
EØ	RLA	ROTATE A LEFT	1,3	
E1	RLB	ROTATE B LEFT	1,3	
E2	RL3A	ROTATE A THRICE LEFT	2,6	
E3	RL3B	ROTATE B THRICE LEFT	2,6	
E4	RRA	ROTATE A RIGHT	1,3	
E5	RRB	ROTATE B RIGHT	1,3	
E6	RR3A	ROTATE B THRICE RIGHT	2,6	
E7	RR3B	ROTATE B THRICE RIGHT	2,6	
E8	SRA	SHIFT A RIGHT	5,0	
E9	SRB	SHIFT B RIGHT	5,0	
EA	SR3A	SHIFT A THRICE RIGHT	6,9	
EB	SR3B	SHIFT B THRICE RIGHT	6,9	
EC	SWPA	SWAP BYTES OF A	1,3	
ED	SWPB	SWAP BYTES OF B	1,3	
EE	CMPA	COMPARE A TO B, SKIP	5,5	**
FØ	COMA	ONE'S COMPL. OF A	1,3	
F1	COMB	ONE'S COMPL. OF B	1,3	
F2	NEGA	TWO'S COMPL. OF A	1,8	
F3	NEGB	TWO'S COMPL. OF B	1,8	
FC	ION	INTERRUPT ON	1,5	
FD	IOF	INTERRUPT OFF	1,5	
FE	HALT	HALT, IOF	-	
FF	WAIT	HALT, ION	-	

OY NOKIA AB

00101 HELSINKI 10
Finland
P.O. Box 780
Telephone 90 - 59 131
Telex 12-2062 eleno sf
Telegrams ELECTRONOKIA

08.50.07/EO vers. B