
RCSL No: 42-i1540
Edition: November, 1980
Author: Knud Erik Hansen

Title:

RC3503 Reference Manual

Keywords:

RC3503 General Purpose Controller, Reference Manual.

Abstract:

This paper describes the RC3503.

(62 printed pages)

Copyright © 1980, A/S Regnecentralen af 1979
RC Computer A/S
Printed by A/S Regnecentralen af 1979, Copenhagen

Users of this manual are cautioned that the specifications contained herein are subject to change by RC at any time without prior notice. RC is not responsible for typographical or arithmetic errors which may appear in this manual and shall not be responsible for any damages caused by reliance on any of the materials presented.

<u>TABLE OF CONTENTS</u>	<u>PAGE</u>
1. INTRODUCTION	1
2. DESIGN OBJECTIVES	3
2.1 Word Length	3
2.2 Storage Addressing	3
2.3 Register Structure	4
2.4 Instruction Format	4
2.5 Input/Output Structure	5
3. COMPUTER ORGANIZATION	7
3.1 Data Formats	7
3.2 Number Representation	7
3.3 Working Registers	8
3.4 Interruption System	9
3.5 Input/Output	12
3.6 Basic Instruction Formats	16
4. ARITHMETIC AND LOGICAL INSTRUCTIONS	17
4.1 Instruction Format 1	17
4.2 Address and Operand Calculation	18
4.3 Instruction List	20
5. BRANCH AND CONTINUE INSTRUCTIONS	25
5.1 Instruction Format 2	25
5.2 Address and Operand Calculation	26
5.3 Instruction List	26
6. SHIFT-ROTATE INSTRUCTIONS	32
6.1 Format 3 for Shift-Rotate Instructions	32
6.2 Fetch Algorithm for Format 3	32
6.3 Instruction List	33
7. INTERLEVEL INSTRUCTIONS	35
7.1 Format 3 for Interlevel Instructions	35
7.2 Instruction List	36

<u>TABLE OF CONTENTS (continued)</u>	<u>PAGE</u>
8. INPUT/OUTPUT INSTRUCTIONS	37
8.1 Format 3 for Input/Output Instructions	37
8.2 Instruction List	38
9. INSTRUCTION SET	41
9.1 Arithmetic and Logical Instructions	41
9.2 Branch and Continue Instructions	42
9.3 Shift-Rotate Instructions	43
9.4 Interlevel Instructions	43
9.5 Input/Output Instructions	44
10. SWITCHES AND INDICATORS	45
10.1 Processor Front Panel	45
10.1.1 Switches	45
10.1.1.1 Bus Switches	46
10.1.1.2 Debug Mode Switch	46
10.1.2 Indicators	47
10.1.3 Jack	49
10.2 The Debug Console	49
10.2.1 Activation	49
10.2.2 Display Commands	50
10.2.3 Control Commands	51
10.2.4 Command Parameters	52
10.3 Installation	52
11. INSTRUCTION EXECUTING TIMES	53
11.1 Address Calculation Times (AT)	53
11.2 Execution Times (ET)	54

1. INTRODUCTION

1.

The RC3503 is a small, general purpose 16-bit controller with an outstanding interruption and input/output system. This makes the computer especially suited for applications such as:

- Data and media conversion systems
- Communication terminal concentration systems
- Front-end computer for larger computer systems
- Process supervision and control systems
- Controller for a number of medium to low-speed peripherals

The main features of the RC3503 are summarized below:

Storage:

The used memory is dynamical, MOS with cycle times of 580 ns. The word length is 16 bits + 2 parity bits.

32 K words RAM memory is the max. address space.

Storage Addressing:

8-bit bytes and 16-bit words are directly addressable.

Arithmetic:

The arithmetic unit processes 16-bit binary integers in parallel using twos complement notation.

Interruption Levels:

32 interruption levels are included in the basic version. Can be expanded to ~~124~~ levels.
110

Working Registers:

The programmer has access to a set of 8 registers per interruption level. One of the registers in each set contains a program counter and a carry indicator, whereas the remaining 7 are general registers of which 3 are used both as accumulators and index registers, and 4 as spare registers.

Instruction Set:

The computer has a repertoire of 50 standard instructions of which 34 refer to storage. All storage reference instructions can also be specified to operate with their effective address as operand, i.e. immediate addressing. An instruction consists of 1, 2, or 3 consecutive words.

Input/Output:

A number of input/output instructions governs the data transfer between central processor and peripherals. Data transfer between peripherals and CPU is performed in serial mode, but the data are treated as parallel data, depending on the characteristics of the external device in question. Peripherals having high transfer rates communicate in blocks of words or bytes directly with main storage. All input/output operations are directed by interrupt signals. RC3503 is equipped with 32 serial I/O channels. The number of I/O channels can optional be expanded to ~~124~~ ¹¹⁰ channels.

Debug Facilities:

A TTY compatible device can be connected directly to RC3503, and communicates with the micromachine in RC3503 to simulate a standard operator panel.

Real Time Clock:

RC3503 is equipped with an onboard Real Time Clock, the frequency of which is controlled by the debugger. Default the Real Time Clock frequency is set to 20 ms and connected to the interrupt level 1.

2. DESIGN OBJECTIVES

2.

The primary design objectives of the RC3503 computer were to construct a low cost computer with a flexible input/output system and with a powerful instruction set for ease of programming. Furthermore, peripheral devices should be connected to the computer with a minimum of interfacing. Important considerations in meeting the objectives are explained below.

2.1 Word Length

2.1

The bits for input/output devices ranges from 1 bit for telecommunication up to 12 bits for card readers and analog-to-digital converters. The basic arithmetic operands should therefore not be less than 12 bits, preferably a few bits longer; on the contrary, a very much longer format serves no purpose since the computer is not intended for administrative data processing.

Because the majority of peripheral equipment transfers 8 bits in parallel at a time, the word length must also be a multiple of 8 bits in order to optimize the utilization of storage. Cost considerations dictated that the smallest integer which satisfies aforesaid conditions should be used, and consequently a word length of 16 bits was chosen.

2.2 Storage Addressing

2.2

Data are handled most efficiently if they can be addressed directly, and because of the widespread use of 8-bit bytes, a byte was adopted as the smallest addressable unit. From an addressing point of view, the instructions are divided into two classes according to whether the storage address refers to bytes or words.

2.3 Register Structure

2.3

Internal registers of importance to the programmer are three 16-bit general registers, one 15-bit program counter, and a carry indication. The advantage of more than one general register is that it reduces the number of storage references required in a given program. The general registers of the RC3503 computer act both as accumulators and index registers, for which reason the full instruction set becomes available for address modifications. The collection of the above-mentioned programable registers is referred to as a set of working registers.

The computer has not only one set of working registers, but 32 sets, namely one for each interruption level. Hence, we have eliminated the storing of current status (program counter, carry, and accumulators) and loading of new status whenever program switching is demanded by the interruption system. The response time for answering an interrupt is thus reduced to a minimum, and the input/output system is designed to take full advantage of this fact.

2.4 Instruction Format

2.4

In principle, an instruction consists of an operation part and an address part. The operation part specifies an operand code, a result register (if more than one exists), and an address mode (indexing, immediate, indirect, etc.). The address part is either a full address or an abbreviated address used in conjunction with either a fixed page scheme or as a displacement relative to an index register or the program counter.

For a fixed 16-bit instruction format we have that if the operand portion expands at the expense of the address, the number of instructions, result registers, and address modes taken together increase, but the usefulness of these instructions is severely affected due to the limited address range. Reversely, a large address part reduces the instruction repertoire and its mode features so that only a fraction of the entire instruction set may

fully utilize the improved address range. As a consequence of this, it was decided to employ a variable instruction format, where an instruction may occupy 1, 2, or 3 words.

2.5 Input/Output Structure

2.5

In the design particular emphasis was put on the construction of an efficient and versatile input/output system. This is important because a major part of small computer systems is used for real-time control where the I/O capability has a profound influence on the total effectiveness of the system.

A normal approach is to use two types of I/O data channels: (1) a low speed channel to which character-oriented devices such as typewriters, paper tape readers, etc. are connected; and (2) a high-speed channel to which block-oriented devices such as magnetic tapes and display units are connected.

Character transfer, via the low-speed channel, is under program control; and the program must either repeatedly test the channel status to see when the device becomes available or the device must respond with an interrupt when it is ready for transmission of next character. The first approach implies that the central processor can perform no other operations until all characters are transmitted, which in many applications are unacceptable. The efficiency of the second scheme is directly related to the time it takes to switch from one program to another.

The high-speed channel transfers blocks of data at high rates directly between storage and I/O devices, independent of program control once the transfer has been initiated. The transfer is supervised by additional hardware, normally: (1) a counter for current storage address; (2) a register for last storage address; (3) a comparator to detect end of block; (4) facilities to pack characters into word format and vice versa; and (5) logic to control the operation of storage during I/O transfers. The hardware mentioned in (1) to (4) is placed in the device controller for which reason the cost of the device is significantly increased.

The logic mentioned in (5) is incorporated in the central processor, but because of its complexity it is normally offered as an option at an additional cost.

Looking at the low-speed channel, it is obvious that the most advantageous I/O system is one where I/O operations are initiated by interrupts and program switching time is zero. With regard to the high-speed channel, it is preferable to keep the block-mode concept, but desirable to minimize cost. The RC3503 solution to this problem is to have one data channel to which both low- and high-speed peripheral equipment may be connected. I/O operations are under program control and an interrupt signal is generated whenever a character is ready for transmission. Each interrupt signal is wired to its own set of working registers, and a fixed priority order exists among the register sets. Switching from one program to another is then reduced to merely move a pointer from the current register set to the new one; and we have in fact an ideal low-speed channel. Block-oriented devices make use of the general registers and the inherent arithmetic capabilities of the central processor, so no extra hardware is needed in the device controllers. When no I/O operation is in progress, the working registers are free for normal program execution. The time required to transmit one character in or out of storage is two storage cycles, one to execute an I/O instruction and one to the actual transfer. The maximum transfer rate is thus halved in relation to a sophisticated high-speed channel, but this difference is seldom of practical interest for small computers.

The available I/O instructions for data transfer are:

1. Transfer one word between working register W1 and external device.
3. Transfer a block of words or bytes directly between storage and external device.

3. COMPUTER ORGANIZATION

3.

3.1 Data Formats

3.1

Two data formats are recognized by the RC3503, an 8-bit byte and a 16-bit word. A word consists of two bytes of which the left-most byte must have an even storage address.

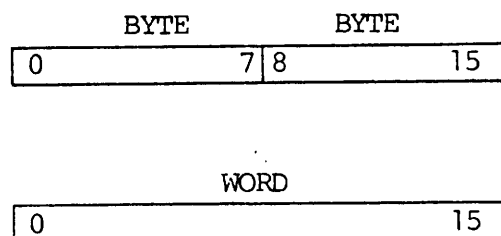


Figure 3.1: Data formats.

Storage addresses are always expressed as byte addresses. The byte locations are numbered consecutively starting with zero.

When an instruction refers to a word in storage, bit 15 in the effective address is ignored. Thus it is irrelevant whether the effective address refers to the left or right byte of a word.

3.2 Number Representation

3.2

The arithmetic unit of the computer assumes that numerical data are signed integers represented in twos complement form. In this notation the left-most bit represents the sign. For positive numbers the sign bit is zero, for negative numbers one.

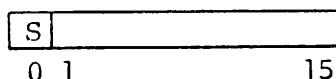


Figure 3.2: Integer word.

The negative of a number is obtained by complementing all bits, including the sign bit, in the number and then adding a 1 to the

right-most bit. A signed integer represented by 16 bits is confined to the range

$$-32,768 \leq \text{integer word} \leq 32,767.$$

Other data formats such as various forms of packed data words, multiple-length fixed-point, and floating-point representations must be defined by appropriate software routines. The programming of these routines is greatly simplified by a CARRY indication. If a carry occurs out of bit 0 in an arithmetic instruction, CARRY is set to 1, otherwise CARRY is reset to 0. A set of instructions makes it possible explicitly to set and reset CARRY just as there is a possibility for branching on the current value of CARRY.

3.3 Working Registers

3.3

Figure 3.3. depicts the working registers, W0 to W3, associated to each interruption level.

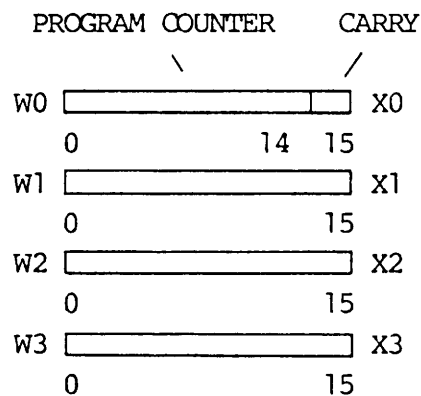


Figure 3.3: A set of working registers.

Registers W1 to W3 are used by the programmer both as accumulators and index registers, in the latter case they are referred to as X1 to X3. Register W0 occupies, in some respect, an exceptional position, because bits 0 to 14 specify the program counter, PC, and bit 15 equals the value of CARRY. Fifteen bits are just enough to locate any instruction in storage, since instructions are always an integer number of full words.

Although there is a distinction in interpretation of W0 and the other registers, they are all alike with respect to instruction execution. By this we mean that any instruction is executed with an operand in, say W2, also is executed if the operand were placed in W0. For example, a LOAD instruction may load any of the W registers with a word from storage, but of course, loading of W0 has the special effect of executing a program jump.

Inter register load and store instructions are also provided which makes it possible to transfer information between W registers belonging to the same or different interruption levels.

3.4 Interruption System

3.4

A program interrupt feature permits an automatic switching from the current program sequence to another sequence in immediate response to specific external or internal events. The efficiency of an interruption system is directly related to the time it takes the central processor to perform this switching. Therefore, in the RC3503, we have associated a set of working registers to each interruption level so switching is reduced to connect a new set of working registers to the control and arithmetic unit; confer figure 3.4.

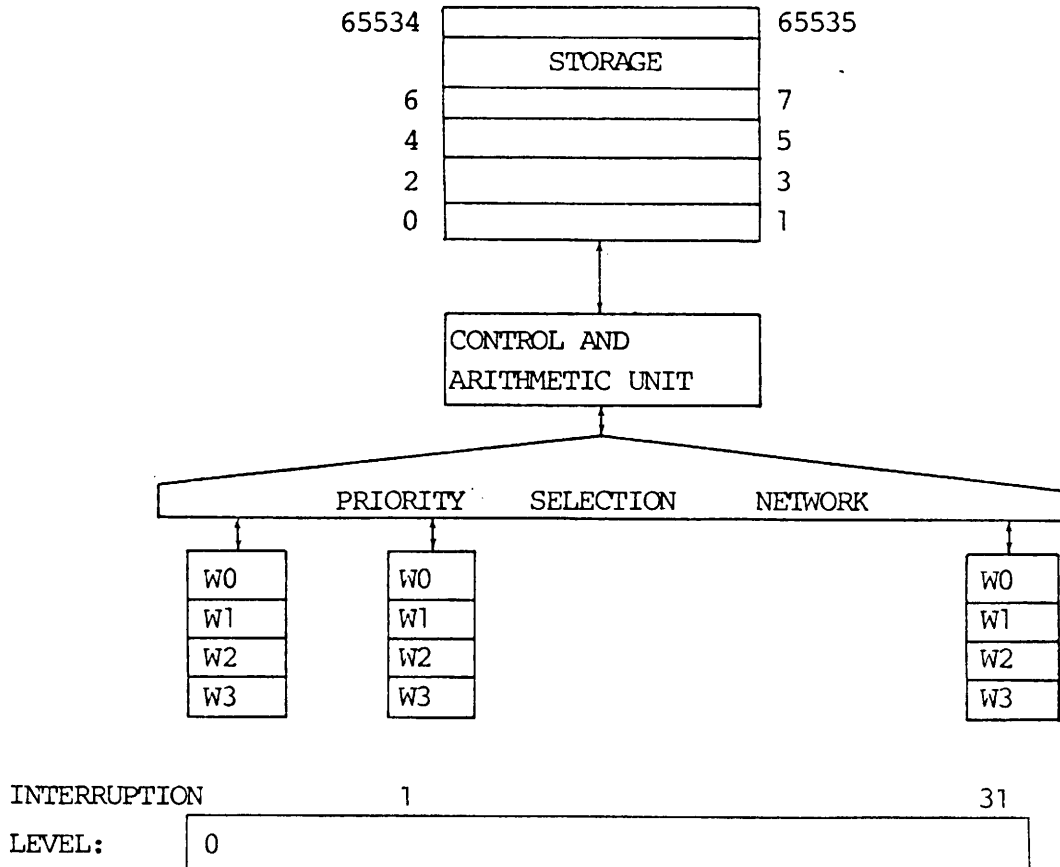


Figure 3.4: RC3503 block diagram.

The priority selection network contains a 32-bit interrupt requester buffer, IR, of which bits 1 to 31 can collect up to 31 incoming interrupt signals. A one in a bit position corresponds to an interrupt request, and bit N corresponds to the Nth interruption level. Bit 0 in register IR occupies an exceptional position, because the value of this bit remains one equivalent to a permanent interrupt request.

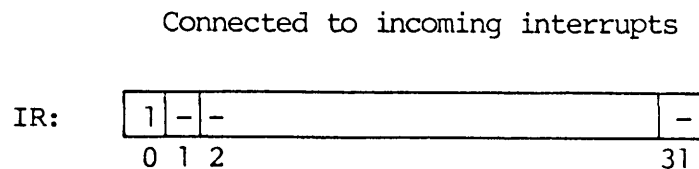


Figure 3.5: Interrupt request register.

Simultaneous interrupt signals are served in order of priority and the signal which is wired to the highest bit number in IR has highest priority. Input/Output devices with a high transfer rate should therefore have their interrupt lines wired to the right-most bits of register IR. If all incoming interrupt signals are honoured and no new requests arrive, interruption level 0 is automatically selected.

The interruption system is normally enabled, i.e. an interrupt request having a priority higher than the current one, causes a new set of working registers to be activated. In some cases, however, it is desirable to disable the interruption system for shorter intervals. In disable mode, other interrupts are indeed collected, but they cannot change the interruption level, i.e. the current program proceeds. The two modes are under program control, and the flip-flop ENABLE is 1 for enable mode, 0 for disable. Register IR can be set under program control and we may summarize the supervision of the interruption system as follows:

1. A bit in IR is set to 1 by a request from an I/O device
2. A bit in IR is set to 1 by a request from the program by the instruction type

INTERLEVEL INSTRUCTIONS

3. The bit in IR, corresponding to the current interruption level, is cleared by two instruction types

LOAD REGISTER CLEAR INTERRUPT AND ENABLE I/O INSTRUCTIONS

4. ENABLE is set to 1 by three instruction types

LOAD REGISTER CLEAR INTERRUPT AND ENABLE INTERLEVEL INSTRUCTIONS I/O INSTRUCTIONS

5. ENABLE is cleared by the instruction type

INTERLEVEL INSTRUCTIONS

Any peripheral is connected to the CPU by means of a 4 pairs cable, which is transformer coupled at both ends secure a very high noise immunity.

The transmission of data is performed in serial mode whether the device connected is of parallel or of serial nature. The data word transferred consists of a 4 bits header and 0 to 16 information bits.

Header

Data part

"1" X ₀ X ₁ "1"	0 to 16 information bits
---------------------------------------	--------------------------

synchronizing bits

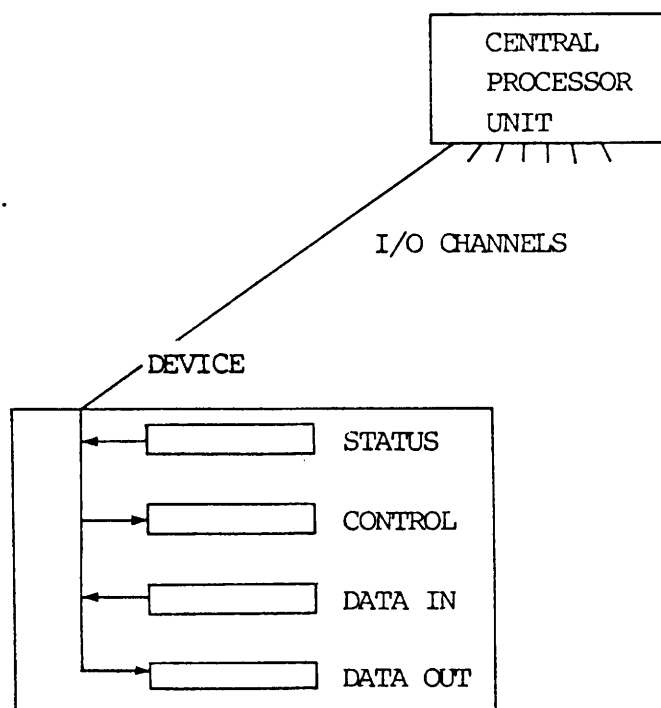
The header contains the following information:

TRANSMISSION FROM CPU TO PERIPHERAL

"OUTPUT	1 0 0 1	Read data
Header"	1 0 1 1	Read status
	1 1 0 1	Write data
	1 1 1 1	write control

TRANSMISSION TO CPU FROM PERIPHERAL

"INPUT	1 0 0 1	16 bits (word)
HEADER"	1 0 1 1	EOI (end of information)
	1 1 0 1	8 bits (byte)
	1 1 1 1	not used



In principle each device contains 4 16-bit registers. The registers are 2 data registers (one for each direction of flow), one control register and one status register. However, for actual devices one or more of the registers may be omitted, some of the registers may be combined to one, or the register length may be shortened from 16 bits right down to 1 bit. The central processor initiates an I/O instruction by selecting the I/O cable leading to the device addresses whereafter the data are transferred.

Communication may be initiated either by program or by interrupt from a device. An interrupt is detected on the data line when a "1" bit is sent to the CPU, and it has not asked for data.

The data flow from the computer to the four device registers and vice versa is illustrated by 4 general I/O commands. The commands describe the possible patterns of execution for the I/O channel and the device controllers, whereas specific details concerning how data are stored in the computer is postponed to Chapter 8, where the actual I/O instructions are defined.

Read Status:

The read status command is a request to the device to transfer the contents of status to working register W1. The number of status bits and their interpretation is device dependent, but the following list indicates their nature.

Busy/Available, the busy mode signifies that the previous operation has not yet been completed, - available mode the opposite.

Disconnected, i.e. power to the device is switched off.

Intervention, i.e. the operator has interfered manually with the device, e.g. he has switched the typewriter to operate in off-line mode.

Timer, i.e. the device or the operator has not completed an operation within a predetermined time.

Write Control:

The write control causes an immediate transfer of a control word from working register W1 to the control register, and the computer proceeds to the next instruction. When the control operation is completed, an interrupt is normally generated.

The individual bits in the control register specify commands, which are executed by the device controller. Examples are:

- Generate an interrupt request
- Start or stop peripheral equipment
- Check or generate parity
- Operations such as for magnetic tapes rewind, erase, upspace, file, etc.

Read Data:

The read data command causes an immediate transfer of information from data buffer to working register W1 or directly into storage. The computer is released as soon as this transfer is completed. When the next character is ready in the device, it is normally indicated by a new interrupt signal.

Write Data:

The write data command causes an immediate transfer from working register W1 or directly from storage to the data buffer in the selected device. The computer is released as soon as this transfer is completed. When the device is ready to receive the next character, it is normally indicated by a new interrupt. Transfer of information is done in analogy with read data.

There are three basic formats and the instructions are grouped according to formats as follows:

Format 1: Arithmetic and Logical instructions

Format 2: Branch and Continue instructions

Format 3: Shift-Rotate, Interlevel, and Input/Output instructions

The length of instruction Format 1 is one or two words, depending on whether the 7-bit displacement is sufficient or not for specifying a word in storage. Format 2 has a length of two or three words also determined by the range of the displacement. Format 3 consists of one word or two words. The chapters to follow define the RC3503 instruction set. Chapter 9 gives a complete instruction list with corresponding mnemonics.

4. ARITHMETIC AND LOGICAL INSTRUCTIONS

4.

4.1 Instruction Format 1

4.1

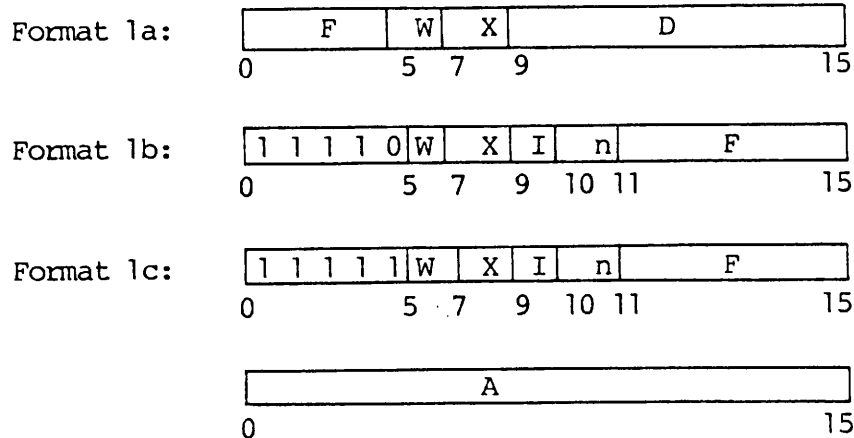


Figure 4.1: Instruction format 1.

Format 1 is shown in Figure 4.1 and it is characterized by having a function or operand code, F, where the left-most bit of the F field equals 0. The remaining four bits specify therefore 16 basic instructions, having the numeric codes from 0 to 15.

Format 1a may be regarded as the basic format and the two others as expansions with the sole purpose of extending the addressing facilities. Format 1b is derived from the basic format by removing the original F field to the five right-most positions of the word, and in the thus released positions the numeric code 30 is inserted. Code 31 is used in a similar way to specify format 1c. The length of formats 1a and 1b is one word, but two words are used in case 1c.

The 2-bit W field specifies one of the four working registers as the result register. Bits 7 to 8 constitute the X field and this field selects the current index register, if any. Working registers W1, W2, and W3 act as index registers when the X field has the values 1, 2, and 3, respectively, whereas no indexing is specified when the value is 0. The latter implies that W0 and with that the program counter cannot be used for address modifications; in other words relative addressing is not available.

The field D and A supply further information for calculating the effective address. D specifies a displacement from -64 to 63 bytes within a contiguous storage area, and A can be used to directly address any byte in a page of maximum capacity of max 32 Kwords.

Immediate addressing is specified if the 1-bit I field equals 1. If immediate addressing is employed, the operand is equal to the calculated effective address and not to the contents of the location to which the effective address points, as is the case when I = 0.

The n bit (i.e. next) controls a count feature that automatically increases the contents of the working register which is selected by the X field. Register W0 is incremented if the X field equals 0. Incrementation takes place for n = 1, and the increment is 2 for instructions which refer to words and 1 for byte references. The instruction list to follow defines the amount of incrementation for each instruction.

4.2 Address and Operand Calculation

4.2

From the programmer's point of view storage addresses are always expressed as byte addresses. The byte locations are numbered consecutively from zero to maximum. The effective address, EA, is calculated on basis of either a displacement, D, or a full address, A, and probably an index register, X. Each of these components are treated as signed integers, but when EA refers to storage, ST, it is interpreted as an unsigned integer of 16 bits. An example, the effective address -1 refers to storage location 65535.

Figure 4.2 shows the evaluation of the effective address and from this it is obvious that Format 1c reduces to Format 1b when A equals zero. As previously mentioned, the operand equals for I = 1 the effective address and for I = 0 the contents of the location specified by EA. Since 1a does not have an I bit, the programmer

cannot specify immediate addressing as he wishes. By definition we have immediate addressing for X field equal to zero, otherwise not.

The program Counter, W0(0:14), is automatically incremented by the central processor to point at the next word whenever an instruction or part of it is fetched from storage.

The calculation of the effective address and the operand is condensed into the following Fetch algorithm:

FORMAT 1a:

X field	EA		Operand	
			I=0	I=1
00	D	$-64 \leq D \leq 63$	-	D
01	X1+D		ST(X1+D)	-
10	X2+D		ST(X2+D)	-
11	X3+D		ST(X3+D)	-

FORMATS 1b and 1c:

X field	EA		Operand	
			I=0	I=1
00	A	$-32,768 \leq A \leq 32,767$	ST(A)	A
01	X1+A		ST(X1+A)	X1+A
10	X2+A		ST(X2+A)	X2+A
11	X3+A		ST(X3+A)	X3+A

Figure 4.2: Address and operand calculation.

Fetch:

FR:= ST(W0); comment Fetch instruction, 1st word;

W0:= W0 + 2; comment Increase Program Counter;

if FR(0:5) < > 30 and FR(0:5) < > 31 then

begin comment Extend the displacement to a 16-bit signed integer;

EA:= 9extFR(9)conFR(9:15)

end;

```

if FR(0:5) = 30 then EA:= 0;
if FR(0:5) = 31 then
    begin comment Fetch instruction, 2nd. word;
        EA:=ST(W0); W0:= W0 + 2
    end;
if FR(7,8) < > 0 then EA:= EA + W(FR(7,8)); comment indexing;
Operand:= if I = 1 then EA else ST(EA);
go to Instruction Execution;

```

4.3 Instruction List

4.3

This section defines arithmetic and logical instructions. The CARRY bit is explicitly mentioned if it is altered by an instruction whose result register is different from W0.

LOAD REGISTER

Load the W register with the storage word addressed or for immediate addressing the effective address of the instruction. The contents of storage remain unchanged. If count is specified, the contents of X are increased by 2.

```

W:= Operand;
if n = 1 then X:= X + 2;

```

LOAD COMPLEMENT

Load the W register with the twos complement of the storage word addressed or for immediate addressing the effective address of the instruction. The contents of storage remain unchanged. Complementation of the maximum negative number yields the maximum negative number, since it has no positive counterpart. CARRY is set in accordance with the carry out of position zero. If count is specified, the contents of X are increased by 2.

```

W:= - Operand;
CARRY:= carry;
if n = 1 then X:= X + 2;

```

LOAD INDIRECT

Load the W register with the storage word addressed by the operand. The contents of storage remain unchanged. If count is specified, the contents of X are increased by 2.

```
W:= ST(Operand);
if n = 1 then X:= X + 2;
```

STORE REGISTER

Store the W register in the storage word addressed. For immediate addressing and no indexing the W register is stored in the location specified by the effective address. For immediate addressing and indexing the W register is stored in the X register. The W register remains unchanged. If count is specified, the contents of X are increased by 2.

```
if l = 0 then ST(EA):= W;
if l = 1 and X field = 0 then ST(EA):= W;
if l = 1 and X field < > 0 then X:= W;
if n = 1 then X:= X + 2;
```

STORE INDIRECT

Store the W register in the storage word addressed by the operand. The register remains unchanged. If count is specified, the contents of X are increased by 2.

```
ST(Operand):= W;
if n = 1 then X:= X + 2;
```

EXCHANGE STORE AND REGISTER

Store the W register in the storage addressed and the previous contents of the storage word is loaded into the register. For immediate addressing and no indexing the W register is stored in the location specified by the effective address and the effective address is loaded into the W register. For immediate addressing and indexing the contents of W and X are exchanged. If count is specified, the contents of X are increased by 2.

```

if l = 0 then begin A:= ST(EA); ST(EA):= W; W:= A end;
if l = 1 and X field = 0 then begin ST(EA):= W; W:= EA end;
if l = 1 and X field < > 0 then begin A:= X; X:= W; W:= A end;
if n = 1 then X:= X + 2;

```

LOAD BYTE

Load the right half of the W register with the storage byte addressed and clear the left half of W. For immediate addressing, the right half of W is loaded by either the left-most or right-most 8 bits of the effective address of the instruction depending on whether the effective address is even or odd. The contents of X are increased by 1.

```

W(0:7):= 0;
W(8:15):= if EA(15) = 0 then Operand(0:7) else Operand(8:15);
if n = 1 then X:= X + 1;

```

STORE BYTE

Store the right half of the W register in the storage byte addressed. The register remains unchanged. Immediate addressing has the same effect. If count is specified, the contents of X are increased by 1.

```

if EA(15) = 0 then ST(EA)(0:7):= W(8:15)
                else ST(EA)(8:15):= W(8:15);
if n = 1 then X:= X + 1;

```

ADD INTEGER WORD

Add to the W register the storage word addressed or for immediate addressing the effective address of the instruction. The contents of storage remain unchanged. CARRY is set in accordance with the carry out of position zero. If count is specified, the contents of X are increased by 2.

```

W:= W + Operand;
CARRY:= carry;
if n = 1 then X:= X + 2;

```

SUBTRACT INTEGER WORD

Subtract from the W register the storage word addressed or for immediate addressing the effective address of the instruction. The contents of storage remain unchanged. CARRY is set in accordance with the carry out of position zero. If count is specified, the contents of X are increased by 2.

```
W:= W - Operand;
CARRY:= carry;
if n = 1 then X:= X + 2;
```

LOGICAL AND

Form the logical AND operation of the W register and the storage word addressed or for immediate addressing the effective address of the instruction. The result is placed in the W register. The contents of storage remain unchanged. If count is specified, the contents of X are increased by 2.

```
W:= W and Operand;
if n = 1 then X:= X + 2;
```

LOGICAL OR

Form the logical OR operation of the W register and the storage word addressed or for immediate addressing the effective address of the instruction. The result is placed in the W register. The contents of storage remain unchanged. If count is specified, the contents of X are increased by 2.

```
W:= W or Operand;
if n = 1 then X:= X + 2;
```

LOGICAL EXCLUSIVE OR

Form the logical EXCLUSIVE OR operation of the W register and the storage word addressed or for immediate addressing the effective address of the instruction. The result is placed in the W register. The contents of storage remain unchanged. If count is specified, the contents of X are increased by 2.

```
W:= W exclusive or Operand;
if n = 1 then X:= X + 2;
```

LOAD REGISTER CLEAR INTERRUPT AND ENABLE

Load the W register with the operand and clear the current interrupt bit. The interrupt system is enabled. The contents of storage remain unchanged. If count is specified, the contents of X are increased by 2.

```
W:= Operand;
IR(current interrupt):=.0;
ENABLE:= 1;
if n = 1 then X:= X + 2;
```

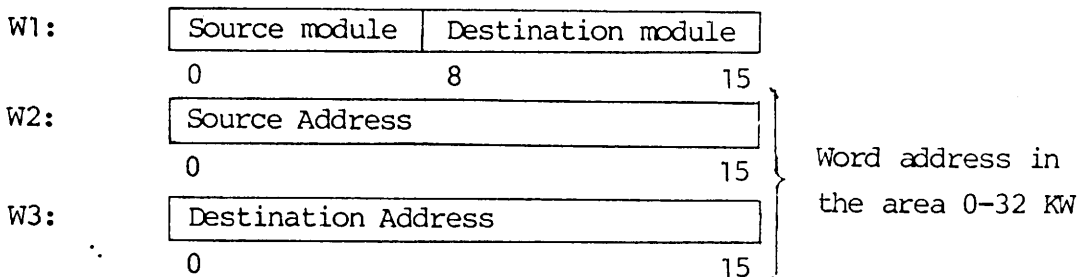
READ SWITCHES

Load the W register with the contents of the Front Panel data switches. If count is specified, the contents of X are increased by 2.

```
W:= switches;
if n = 1 then X:= X + 2;
```

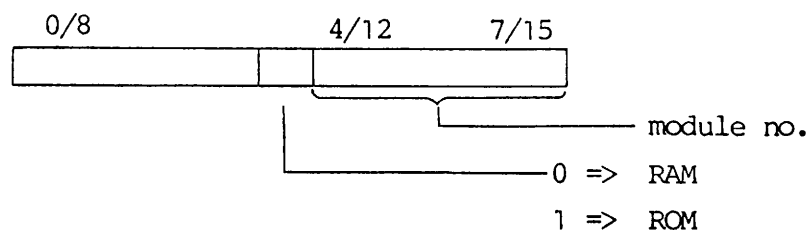
MOVE

Before execution of this instruction W1, W2, and W3 had to be defined:



This instruction moves a word from source place in memory to the destination place. The two addresses is not needed to be in the same 64 K bytes memory module. The addressed W and X register is not destroyed. B.+ 5 - 15 in the instruction are dummy.

The source/destination module in the W1 register has the following layout:



5. BRANCH AND CONTINUE INSTRUCTIONS

5.

5.1 Instruction Format 2

5.1

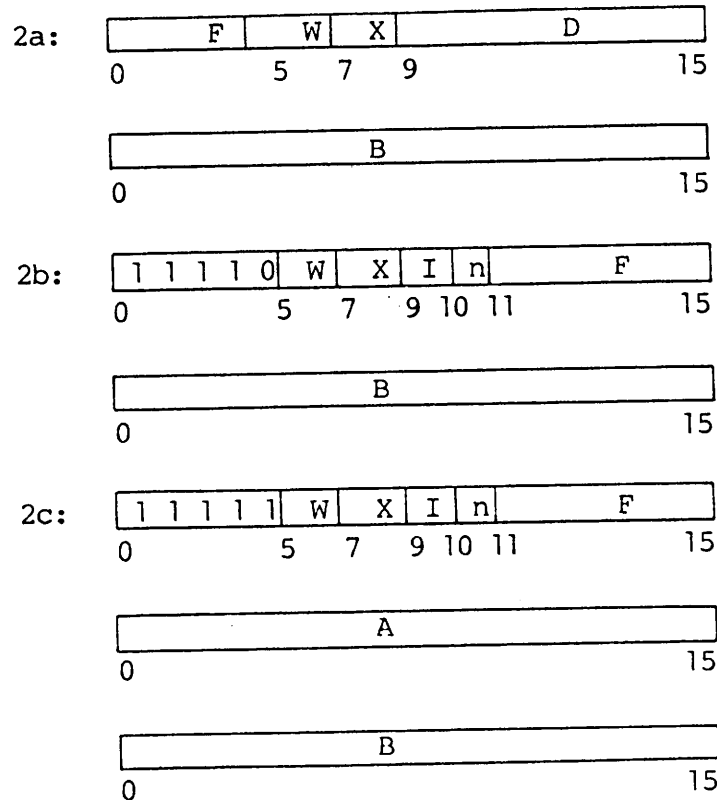


Figure 5.1: Instruction format 2.

Format 2 is shown in figure 5.1 and it is characterized by having a function or operand code, *F*, where the left-most two bits of the *F* field equal 10. The remaining three bits specify eight branch instructions, i.e. instructions which break the normal instruction sequence if a branch condition is satisfied. The three variations of Format 2 are identical to the corresponding Formats 1a, 1b, and 1c apart from two exceptions. First, the new format is extended with one word, namely the *B* field; and second, the *n* bit is interpreted differently.

The 16-bit *B* field contains a branch address to which the program jumps if the specified condition is true. Fields *X*, *D*, *A*, and *I*

have no influence on the branch address, but are employed in defining the effective address and the operand exactly as for Format 1.

The n bit (i.e. not), if one, extends the branch instruction with their logical counterparts, called the continue instructions. The difference between the two instruction groups is obvious from the following two algorithms:

Branch Instructions (n = 0):

if condition = true then branch else continue;

Continue Instructions (n = 1):

if condition = true then continue else branch;

A continue instruction, for example,

continue if p > q

is, of course, equivalent to the branch instruction

branch if p <= q,

but we have preferred not to call them branch instructions since they do not exist in Format 2a.

5.2 Address and Operand Calculation

5.2

The effective address is calculated exactly as defined in the Fetch algorithm in Chapter 4.2; the same apply to the operand.

5.3 Instruction List

5.3

This section defines Format 2 instructions. The CARRY bit is unaltered if no branch occurs, otherwise it is explicitly set to 0 or 1 depending on whether the branch address is even or odd. Branch instructions are defined for all Format 2 variations, whereas continue instructions exist in Format 2b and 2c only.

BRANCH IF GREATER

Compare the W register and the operand as signed integers. The operand equals either the storage word addressed or for immediate addressing the effective address of the instruction. If the register is greater than the operand, then branch to location B, otherwise execute the next instruction in sequence.

W0:= if W > Operand then B else W0 + 2;

BRANCH IF EQUAL

Compare the W register and the operand as signed integers. The operand equals either the storage word addressed or for immediate addressing the effective address of the instruction. If the register equals the operand, then branch to location B, otherwise execute the next instruction in sequence.

W0:= if W = Operand then B else W0 + 2;

BRANCH IF LESS

Compare the W register and the operand as signed integers. The operand equals either the storage word addressed or for immediate addressing the effective address of the instruction. If the register is less than the operand, then branch to location B, otherwise execute the next instruction in sequence.

W0:= if W < Operand then B else W0 + 2;

BRANCH IF GREATER THAN BYTE

Compare the W register and the byte operand extended at the extreme left with zeroes. The byte operand equals the storage byte addressed, except for immediate addressing where the byte operand is the left- or right-most 8 bits of the effective address depending on whether the effective address is even or odd. If the register is greater than the positive operand, then branch to location B, otherwise execute the next instruction in sequence.

A(0:7):= 0;

A(8:15)= if EA(15) = 0 then Operand(0:7) else Operand(8:15);

W0:= if W > A then B else W0 + 2;

BRANCH IF EQUAL TO BYTE

Compare the W register and the byte operand extended at the extreme left with zeroes. The byte operand equals the storage byte addresses; for immediate addressing the byte operand is the left- or right-most 8 bits of the effective address of the instruction depending on whether the effective address is even or odd. If the register equals the extended operand, then branch to location B, otherwise execute the next instruction in sequence.

A(0:7):= 0;

A(8:15):= if EA(15) = 0 then Operand(0:7) else Operand(8:15)

W0:= if W = A then B else W0 + 2;

BRANCH IF LESS THAN BYTE

Compare the W register and the byte operand extended at the extreme left with zeroes. The byte operand equals the storage byte addressed; for immediate addressing the byte operand is the left- or right-most 8 bits of the effective address of the instruction depending on whether the effective address is even or odd. If the register is less than the extended operand, then branch to location B, otherwise execute the next instruction in sequence.

A(0:7):= 0;

A(8:15):= if EA(15) = 0 then Operand(0:7) else Operand(8:15);

W0:= if W < A then B else W0 + 2;

BRANCH IF ALL SELECTED BITS ARE ZERO

Use the operand as a mask to test selected bits in the W register. The operand equals either the storage word addressed or for immediate addressing the effective address of the instruction. If all the selected bits are zero, then branch to location B, otherwise execute the next instruction in sequence.

A:= W and Operand;

W0:= if A = 0 then B else W0 + 2;

BRANCH IF SELECTED BITS HAVE ODD PARITY

Use the operand as a mask to test selected bits in the W register. The operand equals either the storage word addressed or for immediate addressing the effective address of the instruction. If all the selected bits consists of an odd number of ones, then branch to location B, otherwise execute the next instruction in sequence.

A:= W and Operand;

W0:= if odd A then B else W0 + 2;

CONTINUE IF GREATER

Compare the W register and the operand as signed integers. The operand equals either the storage word addressed or for immediate addressing the effective address of the instruction. If the register is greater than the operand, then execute the next instruction in sequence, otherwise branch to location B.

W0:= if W > Operand then W0 + 2 else B;

CONTINUE IF EQUAL

Compare the W register and the operand as signed integers. The operand equals either the storage word addressed or for immediate addressing the effective address of the instruction. If the register equals the operand, then execute the next instruction in sequence, otherwise branch to location B.

W0:= if W = Operand then W0 + 2 else B;

CONTINUE IF LESS

Compare the W register and the operand as signed integers. The operand equals either the storage word addressed or for immediate addressing the effective address of the instruction. If the register is less than the operand, then execute the next instruction in sequence, otherwise branch to location B.

W0:= if W < Operand then W0 + 2 else B;

CONTINUE IF GREATER THAN BYTE

Compare the W register and the byte operand extended at the extreme left with zeroes. The byte operand equals the storage byte addressed; for immediate addressing the byte operand is the left- or right-most 8 bits of the effective address of the instruction depending on whether the effective address is even or odd. If the register is greater than the extended byte operand, then execute the next instruction in sequence, otherwise branch to location B.

A(0:7):= 0;

A(8:15):= if EA(15) = 0 then Operand(0:7) else Operand(8:15);

W0:= if W > A then W0 + 2 else B;

CONTINUE IF EQUAL TO BYTE

Compare the W register and the byte operand extended at the extreme left with zeroes. The byte operand equals the storage byte addresses; for immediate addressing the byte operand is the left- or right-most 8 bits of the effective address of the instruction depending on whether the effective address is even or odd. If the register equals the extended operand, then execute the next instruction in sequence, otherwise branch to location B.

A(0:7):= 0;

A(8:15):= if EA(15) = 0 then Operand(0:7) else Operand(8:15);

W0:= if W = A then W0 + 2 else B;

CONTINUE IF LESS THAN BYTE

Compare the W register and the byte operand extended at the extreme left with zeroes. The byte operand equals the storage byte addressed; for immediate addressing the byte operand is the left- or right-most 8 bits of the effective address of the instruction depending on whether the effective address is even or odd. If the register is less than the extended operand, then execute the next instruction in sequence, otherwise branch to location B.

A(0:7):= 0;

A(8:15):= if EA(15) = 0 then Operand(0:7) else Operand(8:15);

W0:= if W < A then W0 + 2 else B;

CONTINUE IF ALL SELECTED BITS ARE ZERO

Use the operand as a mask to test selected bits in the W register. The operand equals either the storage word addressed or for immediate addressing the effective address of the instruction. If all the selected bits are zero, then execute the next instruction in sequence, otherwise branch to location B.

A:= W and Operand;

W0:= if A = 0 then W0 + 2 else B;

CONTINUE IF SELECTED BITS HAVE ODD PARITY

Use the operand as a mask to test selected bits in the W register. The operand equals either the storage word addressed or for immediate addressing the effective address of the instruction. If all the selected bits consists of an odd number of ones, then execute the next instruction in sequence, otherwise branch to location B.

A:= W and Operand;

W0:= if odd A then W0 + 2 else B;

6. SHIFT-ROTATE INSTRUCTIONS

6.

6.1 Format 3 for Shift-Rotate Instructions

6.1

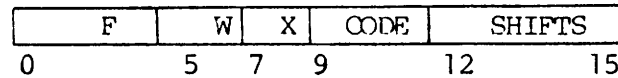


Figure 6.1: Format 3 for shift-rotate instructions.

The shift-rotate group has a format as illustrated in Figure 6.1 with a function code equal to 28. SHIFTS is a positive integer in the interval from 0 to 15 which defines the number of shifts if $x = 0$. If the $X \neq 0$ then the shift value is taken from the register specified by the $x =$ field (x_1, x_2, x_3) bit 12:15. The 3-bit CODE field specifies the type of instruction to be executed as follows

CODE	INSTRUCTION
000	Rotate Right
001	Rotate Left
010	Logical Shift Right
011	Logical Shift Left
100	No carry rotate right
101	No carry rotate left
110	Arithmetic Shift Right
111	Undefined

6.2 Fetch Algorithm for Format 3

6.2

The Fetch algorithm for format 3 instructions becomes very simple since this format consists of only one word.

Fetch:

```
FR:= ST(W0); comment Fetch instruction;
W0:= W0 + 2; comment Increase Program Counter;
go to Instruction Execution;
```

This section defines shift-rotate instructions. The CARRY bit is explicitly mentioned if it is altered by an instruction whose result register is different from W0. (The term "con" used in the following list means "concatenate").

ROTATE RIGHT

Rotate right the contents of CARRY and the W register the number of places specified by SHIFTS. CARRY is rotated into bit 0 of W, bit 15 into CARRY.

```
for SHIFTS:= SHIFTS - 1 step - 1 until 0 do
    CARRYconW(0:15):= W(15)conCARRYconW(0:14);
```

ROTATE LEFT

Rotate left the contents of CARRY and the W register the number of places specified by SHIFTS. CARRY is rotated into bit 15 of W, bit 0 into CARRY.

```
for SHIFTS:= SHIFTS - 1 step - 1 until 0 do
    CARRYconW(0:15):= W(0:15)conCARRY;
```

LOGICAL SHIFT RIGHT

Shift right, with zero insertion in bit 0, the contents of the W register the number of places specified by SHIFTS.

```
for SHIFTS:= SHIFTS - 1 step - 1 until 0 do
    begin CARRY:= W(15); W(0:15):= 0conW(0:14) end;
```

LOGICAL SHIFT LEFT

Shift left, with zero insertion in bit 15, the contents of the W register the number of places specified by SHIFTS.

```
for SHIFTS:= SHIFTS - 1 step - 1 until 0 do
    begin CARRY:= W(0); W(0:15):= W(1:15)con0 end;
```

ARITHMETIC SHIFT RIGHT

Shift right, with sign extension in bit 0, the contents of the W register the number of places specified by SHIFTS.

```
for SHIFTS:= SHIFTS - 1 step - 1 until 0 do
  begin CARRY:= W(15); W(0:15):= W(0)conW(0:14) end;
```

NO CARRY ROTATE RIGHT

Rotate right the contents of W(0:15) the number of places specified by SHIFTS. CARRY is not changed. Bit 15 is rotated into bit 0.

```
for SHIFTS:= SHIFTS - 1 step - 1 until 0 do
  W(0:15):= W(15) con W(0:14);
```

NO CARRY ROTATE LEFT

Rotate left the contents of W(0:15) the number of places specified by SHIFTS. CARRY is not changed. Bit 0 is rotated into bit 15.

```
for SHIFTS:= SHIFTS - 1 step - 1 until 0 do
  W(0:15):= W(1:15) con W(0);
```

7. INTERLEVEL INSTRUCTIONS

7.

7.1 Format 3 for Interlevel Instructions

7.1

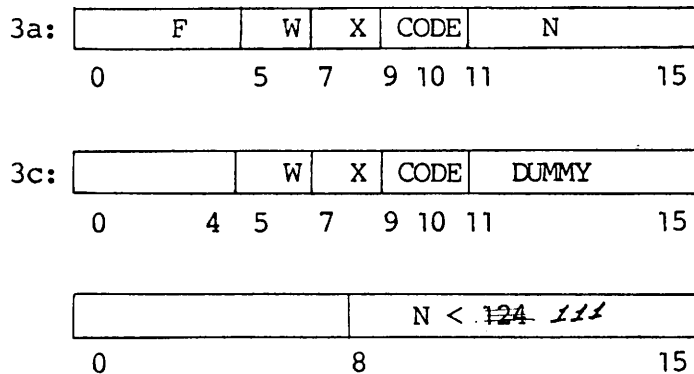


Figure 7.1: Format 3 for interlevel instructions.

The purpose of interlevel instructions is to transfer information between two working registers belonging to the same or to different interruption levels. They have a format as depicted in Figure 7.1. W selects a working register in the current interruption level and X a register belonging to the interruption level specified by the N field. The interlevel instructions may, in addition to their normal execution, also control interruption system. A letter D, E, or S is then used as a suffix to the basic mnemonic code. The effects of the suffixes and the corresponding D and E fields are as follows:

Suffix	Code	Effect on Interruption System
E	01	Enable the interruption system for all interrupts
D	10	Disable the interruption system for all interrupts
S	11	Set the interrupt level specified by the N field

INTERLEVEL LOAD

Load the W register with the working register specified by X from interruption level N. The contents of X remain unchanged. Control the status of the interruption system as follows:

```
W(current level):= X(Nth level);  
if CODE = D then ENABLE:= 0;  
if CODE = E then ENABLE:= 1;  
if CODE = S then INTERRUPT(N):= 1;
```

INTERLEVEL STORE

Store the W register into the working register specified by X from interrupt level N. The contents of W remain unchanged. Control the status of the interruption system as follows:

```
X(Nth level):= W(current level);  
if CODE = D then ENABLE:= 0;  
if CODE = E then ENABLE:= 1;  
if CODE = S then INTERRUPT(N):= 1;
```

8. INPUT/OUTPUT INSTRUCTIONS

8.

8.1 Format 3 for Input/Output Instructions

8.1

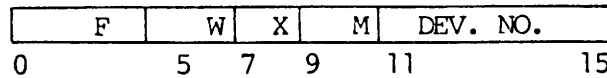


Figure 8.1: Format 3 for input/output instructions.

Figure 8.1 shows the input/output instruction format and it is characterized by a function code where the left-most four bits equal 1100. Hence, only two basic input/output instructions exist. One of them, the read instruction, governs the data transfer from a peripheral device to the computer, whereas the other, the write instructions, controls the data path in opposite direction.

Bits 11 to 15 select the device that is to respond to the instruction. if bit (11:15) = 0 then the device number is set equal to the value of current interrupt level. The M field consist of two bits, which modify the execution of the basic instruction. The remaining two fields, W and X, pick out two registers for compare or address purpose, as it appears from the instruction definitions to follow.

The operation of the device controllers is already explained in Chapter 3.5, where we introduced four general I/O commands. This explanation is also valid for the actual I/O instructions. But, in the previous discussion, we did not define how incoming and outgoing data are interpreted by the computer nor how data may influence the instruction sequence. These questions are answered in the instruction list to follow.

Tables 8.1 and 8.2 show the relations between the M field in the instruction format and the actual I/O instructions. The second column connects an I/O instruction to its equivalent general I/O command; thereby the action taken by the device controllers for various I/O instructions is determined.

	GENERAL	I/O INSTRUCTIONS
M	I/O COMMANDS	
00	Read Data	Read Word and Compare
01	Read Status	Read Status and Compare
10	Read Data	Read Block of bytes
11	Read Data	Read Block of Words

Table 8.1 Modifications of Basic Read Instruction

	GENERAL	I/O INSTRUCTIONS
M	I/O COMMANDS	
00	Write Data	Write Word and Compare
01	Write Control	Write Control and Compare
10	Write Data	Write Block of Bytes
11	Write Data	Write Block of Words

Table 8.2 Modifications of Basic Write Instruction

8.2 Instruction List

8.2

READ STATUS AND COMPARE

Transfer the contents of the STATUS buffer in the selected device to working register W1. If the contents of register W and X differ, and End of Information is zero, then clear the current interrupt bit and repeat the read instruction; otherwise execute the next instruction in sequence. In any event, the interruption system is enabled.

```

A:= X;
W1:= STATUS;
if W < > A and EOI = 0 then begin IR(current level):=0;
    W0:= W0 - 2 end;
ENABLE:= 1;

```

READ WORD AND COMPARE

Transfer a word from the DATA IN buffer in the selected device to working register W1. If the contents of registers W and X differ, and End of Information is zero, then clear the current interrupt bit and repeat the read instruction; otherwise execute the next instruction in sequence. In any event, the interruption system is enabled.

```

A:= X;
W1:= DATA IN;
if W < > A and EOI = 0 then begin IR(current level):= 0;
    W0:= W0 - 2 end;
ENABLE:= 1;

```

READ BLOCK OF BYTES

Transfer a byte from the DATA IN buffer in the selected device to a byte location in storage specified by register X. If X, after being incremented by 1, differs from W, and End of Information is zero, then clear the current interrupt and repeat the read instruction; otherwise execute the next instruction in sequence. In any event, the interruption system is enabled.

```

if X(15) = 0 then ST(X)(0:7):= DATA IN(8:15)
    else ST(X)(8:15):= DATA IN(8:15);
X:= X + 1;
if W < > X and EOI = 0 then begin IR(current level):=0;
    W0:= W0 - 2 end;
ENABLE:= 1;

```

READ BLOCK OF WORDS

Transfer a word from the DATA IN buffer in the selected device to a word location in storage specified by register X. If X, after being incremented by 2, differs from W, and End of Information is zero, then clear the current interrupt bit and repeat the read instruction; otherwise execute the next instruction in sequence. In any event, the interruption system is enabled.

```

ST(X):= DATA IN;
X:= X + 2;
if W < > X and EOI = 0 then begin IR(current level):= 0;
    W0:= W0 - 2 end;
ENABLE:= 1;

```

WRITE CONTROL AND CLEAR

Transfer the contents of register W1 to the CONTROL buffer in the selected device. If the X field of the instruction equals zero, then clear the current interrupt bit and execute the next instruction in sequence; otherwise just execute the next instruction in sequence. In any event, the interruption system is enabled.

```

CONTROL:= W1;
if X field = 0 then begin IR(current level):= 0; end;
ENABLE:= 1;

```

WRITE WORD AND COMPARE

Transfer the contents of register W1 to the DATA OUT buffer in the selected device. If the contents of register W and X differ, then repeat the write instruction, otherwise execute the next instruction in sequence. In any event, the interruption system is enabled.

```

DATA OUT:= W;
if W < > X then begin IR(current level):= 0; W0:= W0 - 2 end;
ENABLE:= 1;

```

WRITE BLOCK OF BYTES

Transfer a byte from the byte location in storage specified by X to the DATA OUT buffer in the selected device. If X, after being incremented by 1, differs from W, then clear the current interrupt bit and repeat the write instruction; otherwise execute the next instruction in sequence. In any event, the interruption system is enabled.

```

DATA OUT(8:15):= if X(15) = 0 then ST(X)(0:7) else ST(X)(8:15);
X:= X + 1;
if W < > X then begin IR(current level):= 0; W0:= W0 - 2 end;
ENABLE:= 1;

```

9. INSTRUCTION SET

9.

9.1 Arithmetic and Logical Instructions

9.1

Format 1a:

F	W	X	D
---	---	---	---

0
5 7 9
15

Format 1b:

1	1	1	1	0	W	X	I	n	F
---	---	---	---	---	---	---	---	---	---

0
5 7 9 10 11
15

Format 1c:

1	1	1	1	1	W	X	I	n	F
---	---	---	---	---	---	---	---	---	---

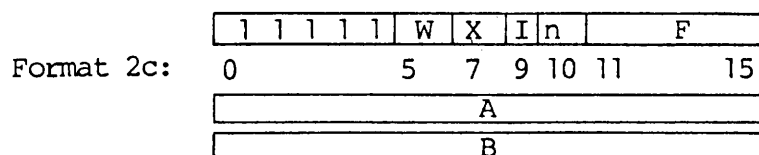
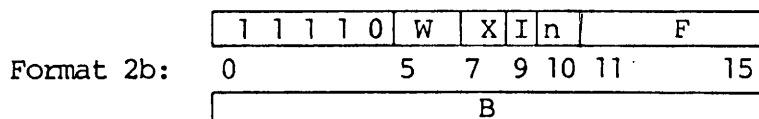
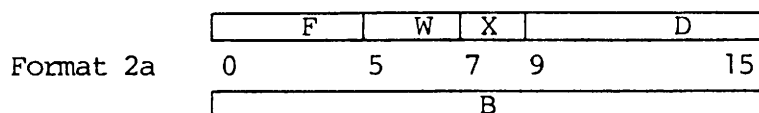
0
5 7 9 10 11
15

A

		F0	F1	F2	F3	F4
ldr	Load Register	0	0	0	0	0
ldc	Load Complement	0	0	0	0	1
ldi	Load Indirect	0	0	0	1	0
str	Store Register	0	0	0	1	1
sti	Store Indirect	0	0	1	0	0
xsr	Exchange Store and Register	0	0	1	0	1
ldb	Load Byte	0	0	1	1	0
stb	Store byte	0	0	1	1	1
add	Add Integer Word	0	1	0	0	0
sub	Subtract Integer Word	0	1	0	0	1
and	Logical AND	0	1	0	1	0
lor	Logical OR	0	1	0	1	1
xor	Logical Exclusive OR	0	1	1	0	0
lce	Load Register Clear Interrupt + Enable	0	1	1	0	1
mov	Move Word between Memory Modules	0	1	1	1	0
rsw	Read Switches	0	1	1	1	1

9.2 Branch and Continue Instructions

9.2



		F0	F1	F2	F3	F4
bgw	Branch if Greater	1	0	0	0	0
bew	Branch if Equal	1	0	0	0	1
blw	Branch if Less	1	0	0	1	0
bgb	Branch if Greater than Byte	1	0	0	1	1
beb	Branch if Equal to Byte	1	0	1	0	0
blb	Branch if Less than Byte	1	0	1	0	1
bsz	Branch if All Selected Bits are Zero	1	0	1	1	0
bop	Branch if Selected Bits have Odd Parity	1	0	1	1	1

Continue instructions as mentioned above, with the suffix ".n" added to the mnemonic code.

9.3 Shift-Rotate Instructions

9.3

Format 3::

F	W	X	CODE	SHIFTS
0	5	7	9	12 15

ror	Rotate Right	0	0	0	1	1	1	0	0
rol	Rotate Left	0	0	1	1	1	1	0	0
lsr	Logical Shift Right	0	1	0	1	1	1	0	0
lsl	Logical Shift Left	0	1	1	1	1	1	0	0
asr	Arithmetic Shift Right	1	1	0	1	1	1	0	0
nrr	No carry rotate right	1	0	0	1	1	1	0	0
nrl	No carry rotate left	1	0	1	1	1	1	0	0
	Undefined	1	1	1	1	1	1	0	0

9.4 Interlevel Instructions

9.4

Format 3a:

F	W	X	CODE	N
0	5	7	9	11 15

Format 3c:

F	W	X	CODE	DUMMY
	5	7	9	11

	N < 124 ///
8	15

ild	Interlevel Load	1	1	0	1	0
ist	Interlevel Store	1	1	0	1	1
	Not used	1	1	1	0	1
	Format b	1	1	1	1	0
	Format c	1	1	1	1	1

9.5 Input/Output Instructions

9.5

Format 3:

F	W	X	M	Dev. No
0	5	7	9	11

 15

					F0	F1	F2	F3	F4
rsc	Read Status and Compare	0	1		1	1	0	0	0
wcc	Write Control and Clear	0	1	-	1	1	0	0	1
rcw	Read Word and Compare	0	0		1	1	0	0	0
rbb	Read Block of Bytes	1	0		1	1	0	0	0
rbw	Read Block of Words	1	1		1	1	0	0	0
wwc	Write Word and Compare	0	0		1	1	0	0	1
wbb	Write Block of Bytes	1	0		1	1	0	0	1
wbw	Write Block of Words	1	1		1	1	0	0	1

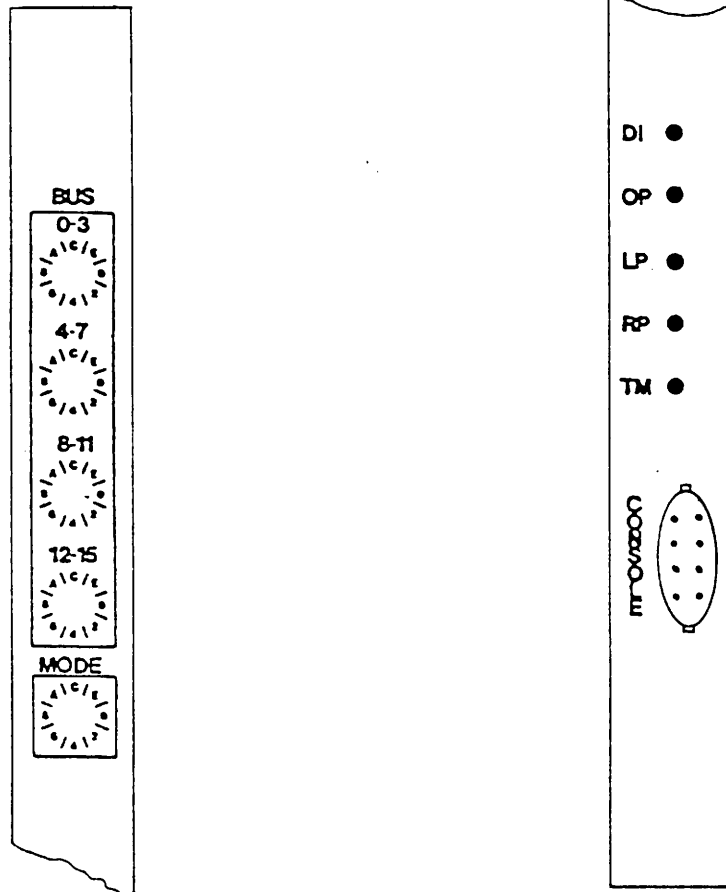
10. SWITCHES AND INDICATORS

10.

10.1 Processor Front Panel

10.1

The front panel of the processor board contains five switches, five indicators, and a jack.



10.1.1 Switches

10.1.1

All switches are rotary switches with 16 positions, indicated by the hexadecimal numbers 0 to F. The switches are set by means of a screwdriver.

10.1.1.1 Bus Switches

10.1.1.

The four switches marked BUS are used to supply the processor with data. There is a switch for bits 0 to 3, 4 to 7, 8 to 11, and 12 to 15.

10.1.1.2 Debug Mode Switch

10.1.1.2

The switch marked MODE is used to select the baud rate for the debug console (see section 10.1.3) and to control the execution of the built-in test programs as follows:

<u>Setting</u>	<u>Baud Rate</u>	<u>Test Program Execution</u>	
0 , 8	300 bps	run test	loop
1 , 9	1200 bps	run test	loop
2 , A	300 bps	skip test	
3 , B	1200 bps	skip test	
4 , C	300 bps	run test	no loop
5 , D	1200 bps	run test	no loop
6 , E	300 bps	skip test	
7 , F	1200 bps	skip test	

Switch state 8-F same as 0-7, but with blocked terminal mode ("BELL" will not switch to Debug).

Explanation

run test	The programs are executed whenever the autoloader button is pressed.
skip test	The programs are not executed.
loop	The programs are executed in an endless loop.
no loop	The programs are executed once.

The test programs are described in the next subsection.

DI Disabled Interrupt

This lamp, when lit, indicates that the processor is running in the disabled interrupt mode.

OP Operation

This lamp, when lit, indicates that the processor is running normally; when it is extinguished, the processor is stopped.

LP Left Parity Error

This lamp, when lit, indicates that a parity error has been detected during a memory read in the left byte.

This lamp can be extinguished only by a power restart or autoloading.

RP Right Parity Error

This lamp, when lit, indicates that a parity error has been detected during a memory read in the right byte.

This lamp can be extinguished only by the power restart or autoloading.

TM Test Mode

This lamp, when lit, indicates that the processor is executing the built-in test programs.

The OP, LP, RP, and TM lamps indicate the number of the test program

TM is the least significant bit in the test program number.

If an error is detected by a test program, one of the following messages is displayed on the debug console:

1 8085 Communication Test

Message: err 1 <dummy> <dummy>
 y<5d> gives 6 bytes transmitted data.
 y<7l> gives 6 bytes received data.

3 Communication Test

Microprogram request to the debug microprocessor.
 Message: none (indicators only).

5 Working Register Address Test

Message: err 5 <address> <data wanted>
 w<address> gives data read.

7 Working Register Data Test

Message: err 7 <address> <data wanted>

9 Memory Address Test

Message: err 9 <offset address> <data wanted>
 y<40> gives memory module number.
 m<module.offset> gives data read.

B Memory Data Test

Pattern 5555.
 Message: err b <offset address> <data wanted>
 y<40> gives memory module number.
 m<module.offset> gives data read.

D Memory Data Test

Pattern AAAA.
 Message: err d <offset address> <data wanted>
 y<40> gives memory module number.
 m<module.offset> gives data read.

10.1.3 Jack

10.1.3

The jack, marked CONSOLE, is a 9-pin CANNON jack for connection of a debug console (a Teletype compatible device, compatible with RC822).

The console must have the following characteristics:

- 8 data bits per character
- No parity bit
- 1 stop bit
- 300 or 1200 bps.

The electrical/logical characteristics of the signals in the jack must conform to Recommendation V.24.

The signals are as follows:

<u>Pin No.</u>	<u>Signal Name</u>
1	+5V
2	+12V
3	received data
4	-12V
5	0V
6	transmitted data
7	0V
8	(not used)
9	0V

10.2 The Debug Console

10.2

10.2.1 Activation

10.2.1

The debug console can be activated at any time (if the mode switch is in the range 0-7) by pressing the BELL key (CTRL G), without stopping the instruction execution in the PU.

Display commands cause eight words of data to be displayed. These commands are executed in two situations:

1. When a display command is entered. Here one can modify the displayed data by entering new data in its place. Pressing the space bar will move the cursor one position to the right, without changing data.

A display command is terminated by pressing one of the following keys:

- | | |
|-----|---|
| CR | Carriage Return terminates the current display command and causes the debug console to await the next command. |
| + | Plus terminates the current display command and executes a display command for the succeeding eight words (M,Y) or the eight registers on the succeeding level (W). |
| - | Minus terminates the current display command and executes a display command for the preceding eight words or the eight registers on the preceding level. |
| ESC | Terminates current command without changing anything. (<ESC> is printed on the console). |

2. When a control command is terminated, the last executed display command is repeated, but no modification of the displayed data is allowed. Here the debug console awaits the next command.

The following display commands are available: (in capital letters):

- | | |
|----------|---|
| M <gadr> | <u>Modify memory</u>
Displays the contents of the 8 memory locations starting at <gadr>. |
|----------|---|

W <reg>	<u>Modify working registers</u> Displays the contents of the 8 working registers starting at <reg>.
L <level>	<u>Modify working registers</u> Displays the level number and contents of the 8 working registers belonging to <level>.
Y <yadr>	<u>Modify control microprocessor RAM</u> Displays the contents of the 8 control microprocessor RAM locations starting at <yadr>.

10.2.3 Control Commands

10.2.3

The following control commands are available (in capital letters):

R	<u>Run</u> Causes the PU to start instruction execution.
S	<u>Instruction stop</u> The PU will execute one instruction, stop, and re-activate the debug console.
S <steps>	<u>Multi-instruction step</u> The PU will execute <steps> instructions, stop, and reactivate the debug console.
C	<u>Instruction step on current level</u> Like S, but on the current level if possible.
C <steps>	<u>Multi-instruction step on current level</u> Like S <steps>, but on the current level if possible.

10.2.4 Command Parameters

10.2.4

All numbers entered and displayed are hexadecimal.

At any time the entering of an empty command (i.e. CR) will cause the previous command to be repeated.

A global address <gadr> is entered in one of the two following formats:

```
<base> : <disp>
      : <disp>
```

where

<base> is the leftmost 16 bits of the 32-bit address.
Always 0000 (Hex) for RC3503.

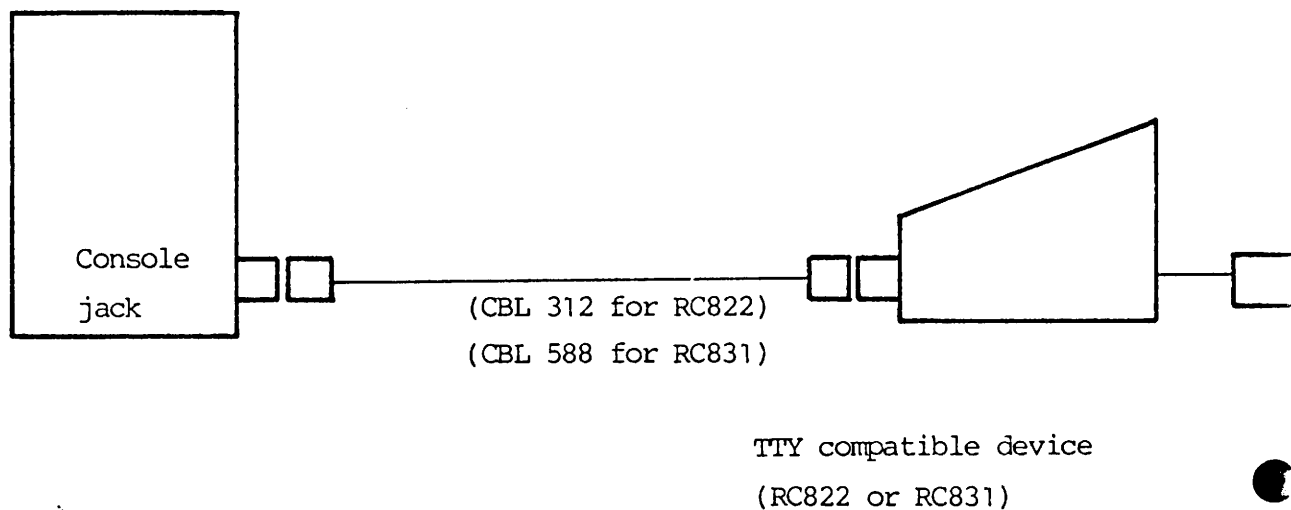
<disp> is the displacement within the selected memory module, i.e. the rightmost 16 bits of an address.

If the second format (: <disp>) is employed, the last entered base is echoed and used.

<STEP> must always consist of 4 HEX digits.

10.3 Installation

10.3



11. INSTRUCTION EXECUTING TIMES

11.

In the following sections the following mnemonics mean:

NC (normal cycle) = 217 nS

MR (memory read cycle) = ~~434 nS~~ 651 nS

MW (memory write cycle) = 651 nS

11.1 Address Calculation Times (AT)

11.1

This scheme describes the execution times (AT) for the address calculation dependent of the format and the index register selected, and if immediate addressing is selected.

~~These times must be added to the times listed below for the instructions to get the total time of an instruction.~~

~~Instruction execution time: $T = AT + ET$.~~

AT		FORMAT 1 & 2			FORMAT 3	
		a	b	c		
I = 0	X = X0	—	3NC+1MR	5NC+2MR		cycles
	X <> X0	4NC+1MR	3NC+1MR	6NC+2MR		
I = 1	X = X0	2NC	2NC	4NC+1MR		cycles
	X <> X0		2NC	5NC+1MR		cycles
FORMAT 3					0 6NC+1MR	cycles

~~Note 1: All times in the scheme are for Disable mode.~~

~~If an enable and fetch on same level add 1NC.~~

~~If an enable and fetch on another level add 2NC.~~

11.2 Execution Times (ET)

11.2

This scheme describes the times which must be added to the ADDRESS CALCULATION TIMES (AT) to get the exact execution time for an instruction.

FORMAT 1 & 2

ldr	2NC
ldc	4NC
ldi	5NC + 1MR 4NC + 1MR
str	$\begin{cases} I = 1 & \text{4NC, } X < > 0 \\ I = 0 & \text{8NC + 1MR} \end{cases}$
sti	7NC + 1MR
xsr	$\begin{cases} X < > X0 & \text{4NC} \\ X = X0 & \text{8NC + 1MR} \end{cases}$
ldb	6NC
stb	7NC + 1MR
add	4NC 3NC
sub	4NC 3NC
and	3NC
lor	3NC
xor	3NC
lce	9NC 5NC
rsw	9NC FORMAT B AND C 8NC FORMAT A
Move	19NC + 1MR + 1MR
MOB	BYTE BLOCK: 19NC + 2MR + (9NC + 1MR + 1MR) * NUMBER OF BYTES WORD BLOCK: 21NC + 2MR + (8NC + 1MR + 1MR) * NUMBER OF WORDS

If count is specified 2NC must be added.

bgw	6NC	For branch and continue
bew	3NC	instructions the same timing is
blw	6NC	valid, if the instruction, however,
bgb	12NC	branches, 5NC + 1MR must be added.
beb	9NC	
blb	12NC	
bsz	3NC	
bop	4NC	

FORMAT 3

ror	}	9NC	Add 1NC per shift
rol			
lsr			
lsl			
asr			

ild	8NC	9NC	FORMAT A	AND LEV < H80 } + 2 NC IF CODE = D, E AND LEV >= H80 } + 5 NC IF CODE = S
ist	8NC	12NC 14NC	FORMAT C FORMAT C	
		9 12 16	FORMAT A FORMAT C FORMAT C	+ 2 NC IF CODE = D, E + 5 NC IF CODE = S AND LEV >= H80 } + 2 NC IF CODE = D, E
rsc	27NC	NOTE		
wcc -	7, 6NC	23 NC	IF X = 0	
rwc	27NC	9 NC	IF X <> 0	
brr -	30NC	NOTE		
rbw -	30NC	NOTE		
wwc -	14NC			
wbb -	22NC			
wbw -	22NC			

Note: add $(2t + 4t + nt)$ uSec., where t is the bit length of the selected device and n is the number of bits transferred.

0,2 μ s

11.3 FETCH TIME (FT)

Enable and no levelshift	6NC + 1MR	= 1.750
Enable and levelshift	9NC + 1MR	
Disable and fetch on current level	6NC + 1MR	
Disable and fetch on another level	9NC + 1MR	

11.4 TOTAL EXECUTION TIME (T)

$$T = AT + ET + FT$$

11. INSTRUCTION EXECUTING TIMES

11.

In the following sections the following mnemonics mean:

NC (normal cycle) = 217 nS

MR (memory read cycle) = 434 nS

MW (memory write cycle) = 651 nS

11.1 Address Calculation Times (AT)

11.1

This scheme describes the execution times (AT) for the address calculation dependent of the format and the index register selected, and if immediate addressing is selected.

These times must be added to the times listed below for the instructions to get the total time of an instruction.

Instruction execution time: $T = AT + ET$.

AT		FORMAT 1 & 2			FORMAT 3	
		a	b	c		
I = 0	X = X0		4NC+1MR	8NC+2MR		cycles
	X <> X0	4NC+1MR	4NC+1MR	8NC+2MR		
I = 1	X = X0	2NC	2NC	5NC+1MR		cycles
	X <> X0		2NC	5NC+1MR		
FORMAT 3					6NC+1MR	cycles

Note 1: All times in the scheme are for Disable mode.

If an enable and fetch on same level add 1NC.

If an enable and fetch on another level add 2NC.

11.2 Execution Times (ET)

11.2

This scheme describes the times which must be added to the ADDRESS CALCULATION TIMES (AT) to get the exact execution time for an instruction.

FORMAT 1 & 2

ldr	2NC
ldc	4NC
ldi	5NC + 1MR
str	$\begin{cases} I = 1 & 4NC \\ I = 0 & 8NC + 1MW \end{cases}$
sti	7NC + 1MW
xsr	$\begin{cases} X \diamond X0 & 5NC \\ X = X0 & 8NC + 1MW \end{cases}$
ldb	6NC
stb	7NC + 1MW
add	4NC
sub	4NC
and	3NC
lor	3NC
xor	3NC
lce	9NC
rsw	9NC
Move	19NC + 1MR + 1MW

If count is specified 2NC must be added.

bgw	6NC	For branch and continue
bew	3NC	instructions the same timing is
blw	6NC	valid, if the instruction, however,
bgb	12NC	branches, 5NC + 1MR must be added.
beb	9NC	
blb	12NC	
bsz	3NC	
bop	4NC	

FORMAT 3

ror	}	9NC	Add 1NC per shift
rol			
lsr			
lsl			
asr			

ild	8NC
-----	-----

ist	8NC
-----	-----

rsc	27NC	NOTE
-----	------	------

wcc -	1,6NC
-------	-------

rwC	27NC	NOTE
-----	------	------

brr -	30NC	NOTE
-------	------	------

rbw -	30NC	NOTE
-------	------	------

wwc -	14NC
-------	------

wbb -	22NC
-------	------

wbw -	22NC
-------	------

Note: add $(2t + 4t + nt)$ uSec., where t is the bit length of the selected device and n is the number of bits transferred.

RETURN LETTER

Title: RC3503 Reference Manual

RCSL No.: 42-i1540

A/S Regnecentralen af 1979/RC Computer A/S maintains a continual effort to improve the quality and usefulness of its publications. To do this effectively we need user feedback, your critical evaluation of this manual.

Please comment on this manual's completeness, accuracy, organization, usability, and readability:

Do you find errors in this manual? If so, specify by page.

How can this manual be improved?

Other comments?

Name: _____ Title: _____

Company: _____

Address: _____

Date: _____

Thank you

..... Fold here

..... Do not tear - Fold here and staple

Affix
postage
here

 **REGNECENTRALEN**
af 1979

Information Department
Lautrupbjerg 1
DK-2750 Ballerup
Denmark

11. INSTRUCTION EXECUTING TIMES

11.1

In the following sections the following mnemonics mean:

NC (normal cycle) = 217 nS

MR (memory read cycle) = ~~434 nS~~ 651 nS

MW (memory write cycle) = 651 nS

11.1 Address Calculation Times (AT)

11.1

This scheme describes the execution times (AT) for the address calculation dependent of the format and the index register selected, and if immediate addressing is selected.

~~These times must be added to the times listed below for the instructions to get the total time of an instruction.~~

~~Instruction execution time: $T = AT + ET$.~~

AT		FORMAT 1 & 2			FORMAT 3	
		a	b	c		
I = 0	X = X0	—	3NC+1MR	5NC+2MR		cycles
	X <> X0	4NC+1MR	3NC+1MR	6NC+2MR		
I = 1	X = X0	2NC	2NC	4NC+1MR		cycles
	X <> X0		2NC	5NC+1MR		cycles
FORMAT 3					0 6NC+1MR	cycles

~~Note 1: All times in the scheme are for Disable mode.~~

~~If an enable and fetch on same level add 1NC.~~

~~If an enable and fetch on another level add 2NC.~~

11.2 Execution Times (ET)

11.2

This scheme describes the times which must be added to the ADDRESS CALCULATION TIMES (AT) to get the exact execution time for an instruction.

FORMAT 1 & 2

ldr	2NC
ldc	4NC
ldi	5NC + 1MR 4NC + 1MR
str	$\begin{cases} I = 1, X < > 0 & 4NC \\ I = 0 & \cancel{8NC + 1MW} \quad 7NC + 1MW \end{cases}$
sti	7NC + 1MW
xsr	$\begin{cases} X \diamond X0 & \cancel{5NC} \quad 4NC \\ X = X0 & \cancel{8NC + 1MW} \quad 7NC + 1MW \end{cases}$
ldb	6NC
stb	7NC + 1MW
add	4NC 3NC
sub	4NC 3NC
and	3NC
lor	3NC
xor	3NC
lce	9NC 5NC
rsw	9NC FORMAT B AND C 8NC FORMAT A
Move	19NC + 1MR + 1MW
MOB	BYTE BLOCK: $19NC + 2MR + (9NC + 1MR + 1MW) \times \text{NUMBER OF BYTES}$ WORD BLOCK: $21NC + 2MR + (8NC + 1MR + 1MW) \times \text{NUMBER OF WORDS}$

If count is specified 2NC must be added.

bgw	6NC	For branch and continue
bew	3NC	instructions the same timing is
blw	6NC	valid, if the instruction, however,
bgb	12NC	branches, 5NC + 1MR must be added.
beb	9NC	
blb	12NC	
bsz	3NC	
bop	4NC	

FORMAT 3

ror	}	9NC	Add 1NC per shift
rol			
lsr			
lsl			
asr			

ild	8NC	9NC	FORMAT A	} + 2 NC IF CODE = D, E + 5 NC IF CODE = S
		12NC	FORMAT C	
ist	8NC	14NC	FORMAT C	AND LEV < H80 } + 2 NC IF CODE = D, E AND LEV >= H80 + 5 NC IF CODE = S
		9	FORMAT A	} + 2 NC IF CODE = D, E + 5 NC IF CODE = S
		12	FORMAT C	
		16	FORMAT C	
rsc	27NC		NOTE	AND LEV >= H80 + 2 NC IF CODE = D, E
wcc -	1, 6NC	23NC	IF X = 0	
		9 NC	IF X <> 0	
rwc	27NC		NOTE	
brr -	30NC		NOTE	
rbw -	30NC		NOTE	
wwc -	14NC			
wbb -	22NC			
wbw -	22NC			

Note: add (2t + 4t + nt) uSec., where t is the bit length of the selected device and n is the number of bits transferred.

0,245

11.3 FETCH TIME (FT)

Enable and no levelshift	6NC + 1MR	= 1.953
Enable and levelshift	9NC + 1MR	
Disable and fetch on current level	6NC + 1MR	
Disable and fetch on another level	9NC + 1MR	

11.4 TOTAL EXECUTION TIME (T)

$$T = AT + ET + FT$$