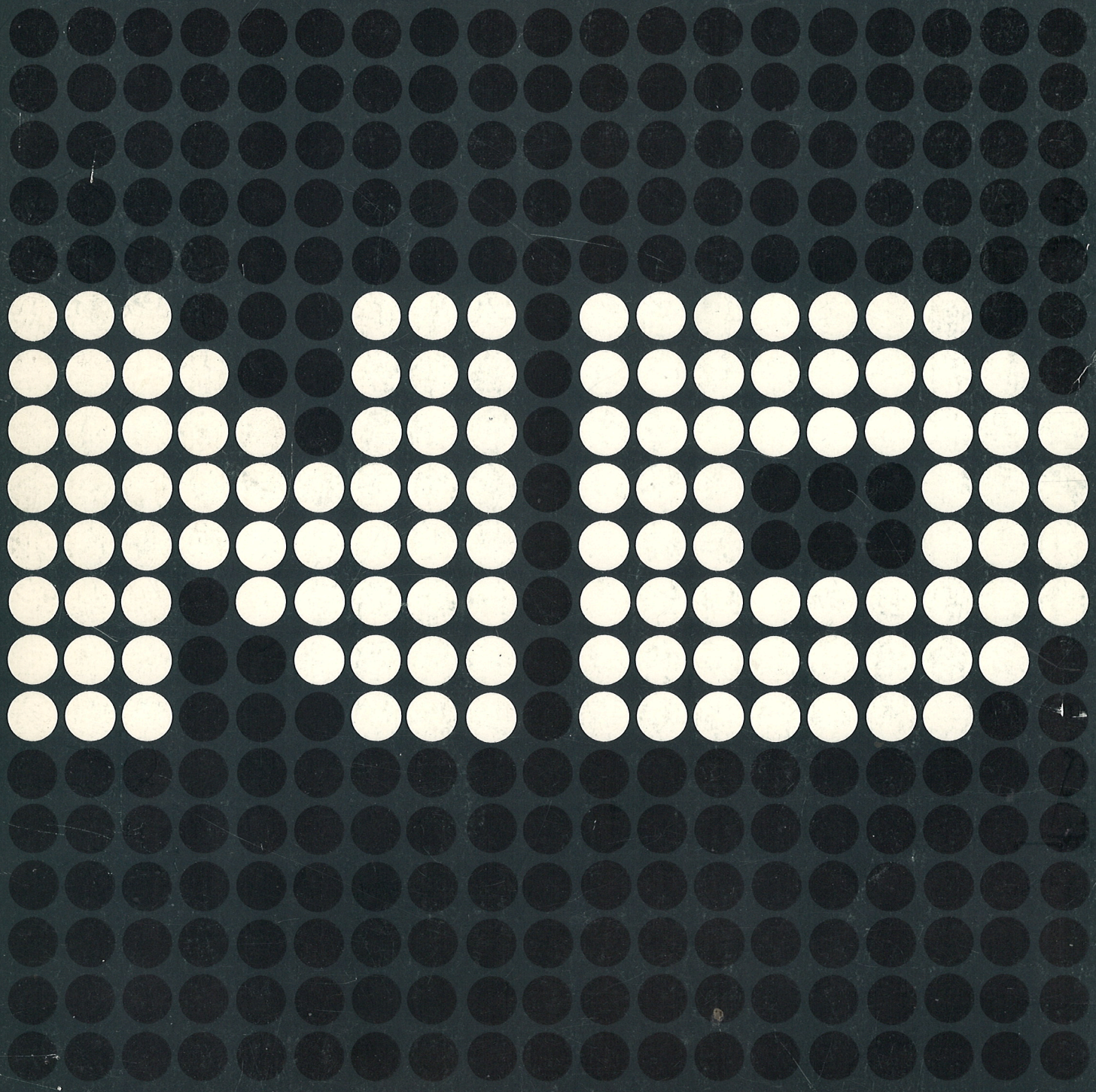
NORD - 50
REFERENCE MANUAL

NORSK DATA A.S



NORD - 50
REFERENCE MANUAL

REVISION RECORD

Revision	Notes
02/76	Original Printing

NORD-50 Reference Manual
Publication No. ND-05.003.01

A/S NORSK DATA-ELEKTRONIKK
Lørenveien 57, Oslo 5 - Tlf.: 21 73 71

TABLE OF CONTENTS

1 GENERAL DEFINITIONS

1.1 Word

The word is the basic storage unit, both in memory and in the central processing unit, CPU. The wordlength of NORD-50 is 32 bits.

Each word in NORD-50 has a unique name which is the name of a register or an address to a word in memory. The content of a word is denoted: (name of word).

1.2 Instruction

Contents of a word interpreted as an instruction to the central processor, CPU.

The instructions in NORD-50 are always one word long. The instructions are divided into three groups as follows:

a) memory reference instructions, Section 2

b) inter-register instructions, Section 3.

c) argument instructions, Section 4.

1.3 Integer

Contents of a word interpreted as a binary number.

Negative integers are represented in two's complement. Arithmetic is performed in two's complement.

One's complement is obtained by setting each bit in the word to its opposite value. Two's complement is obtained by adding one to the one's complement of the word.

1.4 Bits and Bit fields

Each bit in a word or floating word has an unique index as follows.

The least significant bit is bit 0. The most significant bit is bit 31. In double precision floating words bit 63 is bit 31 in the first of the two words making the floating words. Bit 0 in the double precision floating word is bit 0 in the second of the two words.

The content of bit i in a word is denoted: (name of word $_i$). A number of contiguous bits in a word is called a bit field. The bit field ranging from bit i to bit j is denoted: name of word $_{i-j}$, The content of a bit field is denoted: (name of word $_{i-j}$).

1.5 Floating Words

A single precision floating word consists of one 32 bit word. A double precision floating word consists of two contiguous 32 bit words in memory, logically connected to give one 64 bit word. The first word of the floating word is called the left portion word, the second word is called the right portion word. Thus, bit field 0-31 is in the right portion word and bit field 32-63 is in the left portion word of the double precision floating word. The address of a double precision floating word is the address of its left portion word. Cfr. Section 1.9.4, Floating Register.

Relation between words and double precision floating words in memory:

bit_{31}	$word_i$	bit_0	bit_{31}	$word_{i+1}$	bit_0
bit_{63}	double precision floating $word_i$				bit_0

In the description of the instructions, double precision floating point $word_i$ will be denoted: $word_i$, $word_{i+1}$.

1.6 Floating Point Number

A floating point number is given by its mantissa, m, and exponent, n, as follows:

$$number = m \cdot 2^n$$

A floating point number is stored in a floating word, cfr. Section 1.5, Floating Words.

The representation of m and n in the floating words are as follows:

Single precision

bit 0-21+1 bit	mantissa, m
bit 22-30	exponent , n
bit 31	sign of mantissa , m is negative if bit 31=1

Double precision

bit 0-53+1 bit	mantissa , m
bit 54-62	exponent , n
bit 63	sign of mantissa , m is negative if bit 63=1

The range of the exponent is from -377_8 to $+377_8$ making the range of the floating point number approximately from $-10^{67.8}$ to $+10^{67.8}$. The exponent is represented in a 9 bit field but is biased by 400_8; i.e. 400_8 is added to the exponent to make it a positive number.

Note: The one extra bit in the mantissa is the most significant, and is set to one if not all bits in the exponent is zero. It is removed in the result.

The length of the mantissa is 23 bits in single precision, which corresponds to about 7 decimal digits.

The length of the mantissa in double precision is 55 bits, which corresponds to about 16.5 decimal digits.

The floating point numbers have to be normalized before CPU can operate properly on them. There is no method for the CPU to determine whether a number is normalized or not. All results from the floating arithmetic are normalized.

1.7 Integer Arithmetic

Integer operands are always one word. Negative numbers are represented in two's complement.

In inter register arithmetic the add and subtract functions may affect, or be affected by a carry bit, CB, if the proper subcode is specified.

A multiply instruction will affect an overflow register, OR, which will contain the 32 most significant bits of the product. A divide instruction will affect a remainder register, RR, which will contain a 32 bits remainder after the division.

Arithmetical overflow or a positive product where OR is non-zero (OR is all ones for negative products) may, if specified, cause a monitor call.

1.8 Floating Point Arithmetic

The results of the floating add and subtract functions are rounded. As rounding indicators are used the two most significant of the bits which are shifted out to keep a 23 bit or a 55 bit mantissa. More about this rounding in Appendix A.

In floating multiply, bit 0 in the final result is normally set to one Bit 0 is not set to one if the shifted-out bits are all zero.

In floating divide, bit 0 in the final result is normally set to one. Bit 0 is not set to one if the remainder is zero.

1.9 Register Structure of NORD-50

Registers are storage cells in the central processing unit, CPU. The register used by the programmers in NORD-50 is one word long (32 bits) or one double precision word long (64 bits)

1.9.1 P-Register

The program counter, P contains the address of the instruction being read from memory for execution by the CPU. P is one word long. All instructions affect P. The instructions increment P by one, except the stop, jump and skip instructions.

1.9.2 Instruction Register

The instruction register, IR, contains the instruction being executed by the CPU. IR is one word long.

1.9.3 General Registers

The NORD-50 CPU has 64 general single word registers. 32 of these may be used as single precision floating registers, or all 64 may be used to form 32 double precision floating word registers.

The general registers are normally used by the programs as storage cells for operands and results.

Some of the registers may be used as base registers, index registers and modification registers. Cfr. Section 1.9.4 - Floating Registers, Section 1.9.5 - Base Register, Section 1.9.6 - Index Register, and Section 1.9.7 - Modification Register. See also Figures 1.1 and 1.2.

The 64 general registers are denoted GR_0, $GR_1, \ldots, GR_{63}$ in the descriptions.

Note: GR_0 always contains zero. This implies that $(FR_0) = 0$, $(BR0) = 0$, $(XR_0) = 0$ and that MR_0 cannot be used as modification register to GR_0.

1.9.4 Floating Registers

There are 32 floating point registers with 32 or 64 bit wordlength. For double precision where wordlength is 64 bits, one floating register consists of a pair of general registers. The exponent part will always be in the same general register regardless of word length.

The 32 floating registers FR_0 - FR_{15} and FR_{32} - FR_{47} are organized from the general registers as follows:

	Single precision	Double precision
$0 \leqslant i < 16$	FRi = GRi	FRi = GRi, GRi+16
$32 \leqslant i < 48$	FRi = GRi	FRi = GRi, GRi+16

Floating registers are denoted F in single precision, and FD in double precision.

A – FIXED POINT

0 GR1, x 1, B1 GR2, x 2, B2 GR15, X15, B15	GR16, M0 GR17, M1 GR18, M2 GR31, M15	GR32 GR33 GR34 GR47	GR48 GR49 GR50 GR63

B – 32 BIT FLOATING POINT (SINGLE PRECISION)

0 FR1 FR2 FR15	TRANSPARENT	FR32 FR33 FR34 FR47	TRANSPARENT

C – 64 BIT FLOATING POINT (DOUBLE PRECISION)

0 FDR1 FDR2 FDR15	FDR32 FDR33 FDR34 FDR47

Figure 1.1: *NORD-50 – Register Block*

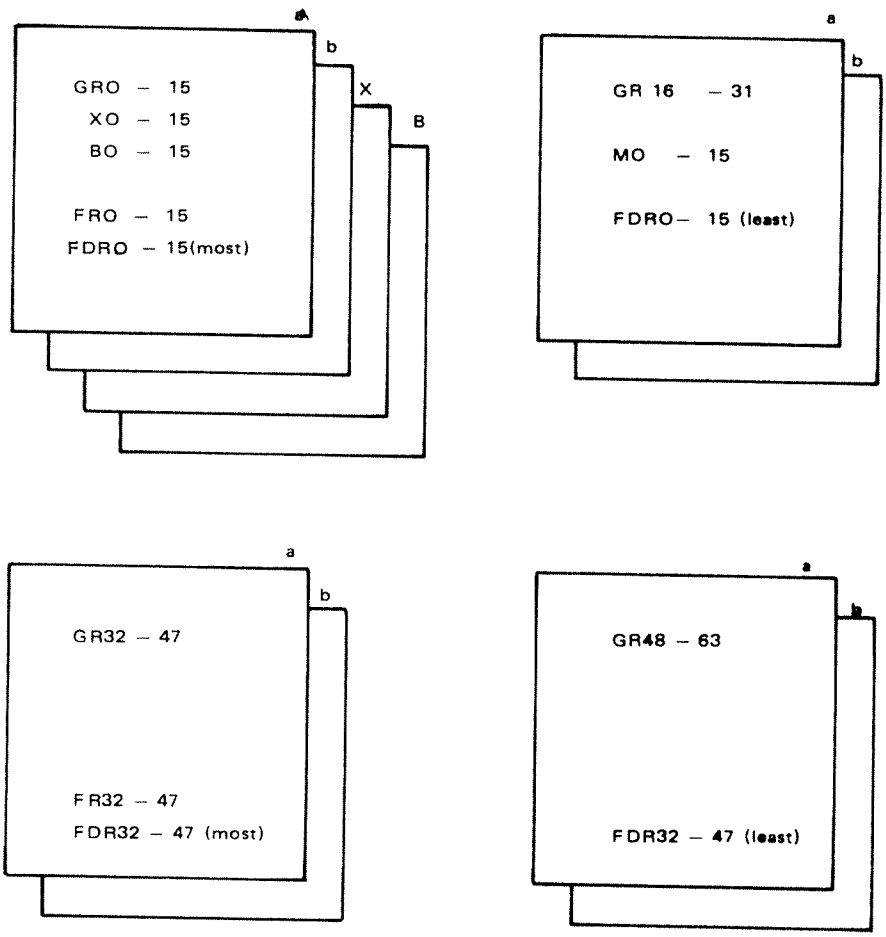


Figure 1.2: *NORD-50 – Register Block Lay-out*

1.9.5 Base Registers

The 16 first general registers may be used as base register, BR.

$BR_i = GR_i \qquad 0 \leq i < 16$

Cfr. Section 2.3 - Memory Addressing - for further description.

1.9.6 Index Registers

The 16 first general registers may be used as index registers, XR.

$XR_i = GR_i \qquad 0 \leq i < 16$

Cfr. Section 2.3 - Memory Addressing - for further description.

If the register has been used as a destination register of a floating point arithmetic operation or double shift, the result cannot be used as a base or index address modification.

1.9.7 Modification Registers

Associated with each of the first 16 general registers there is one modification register, MR. The modification registers are organized from the general registers as follows:

$MR_i = GR_{i+16} \qquad 0 \leq i < 16$

The modification registers are used to hold an increment to base or index registers in certain jump instructions. Cfr. Section 2 - Memory Reference Instruction.

1.9.8 Auxiliary Register

The overflow register, OR, is used to hold the upper 32 bit part of the product of a multiplication.

The remainder register, RR, is used to hold the remainder after a division. OR and RR can be read and written by RIO-instructions.

1.10 Syntax of Instruction Descriptions

1.10.1 := Assignment Operator

Set data element to the left of := equal to the data element at the right side of :=.

1.10.2 +, -, *, / Arithmetic Operators

The mode of arithmetic operation is shown by using integer word indicators for integer arithmetic and floating word indicators for floating point arithmetic.

1.10.3 $>\geq = \neq <\leq$ Relation Operators

The relation operators are used to show arithmetical relations. A relation has the value true or false.

1.10.4 Logical Operators

The logical operations are done by using two operands, each one word long, giving a one word result. However, the logical operations are by its nature single bit operations. They are done on pairs of operand bits, which are made by taking the bits with the same bit address from the two operand words. The result is stored in the same bit position in the result word.

Following is a description of the logical operations as single bit operations.

Logical OR , $\vee$

A	0	0	1	1
B	0	1	0	1
A$\vee$B	0	1	1	1

Logical exclusive OR , $\forall$

A	0	0	1	1
B	0	1	0	1
A$\forall$B	0	1	1	0

Logical AND , $\wedge$

A	0	0	1	1
B	0	1	0	1
A$\wedge$B	0	0	0	1

Logical Complement , — (one's complement)

A	0	0	1	1
$\overline{A}$	1	1	0	0

The logical complement is a one operand operation. Each bit is set to its opposite value. The one's complement of a word is not its corresponding negative value. Negative integer numbers are represented in two's complement.

1.10.6 Non-conditional Instructions

A non-conditional instruction is described by one or more assignment statements. If there are more than one assignment statement, they will be connected by the word AND. The assignment statement is executed in the sequence they are shown.

Note: In all instructions P is incremented by one unless otherwise specified.

1.10.7 Conditional Instructions

The conditional instructions are described as follows:

IF relation is true

THEN non-conditional instructions
Instruction finished

ELSE non-conditional instructions
Instruction finished

If there are statements before the conditional phase, the word AND is used to connect the "IF ... " to the rest of the description.

1.11 Instruction Execution Times (μs)

Break conditions set:	None	Store	Any
LDR, ADD	1.56	1.56	1.76
STR	1.62	1.77	1.84
LDD	2.40	2.40	2.70
STD	2.37	2.64	2.70
FAD	1.84	1.84	1.89
FADD	2.51	2.51	2.69
RAF, RAFD, SHIFT, BIT	0.84	0.84	0.84
FMU, FDV	4.76	4.76	4.82
RMF, RDF	3.74	3.74	3.74
RMFD	7.18	7.18	7.18
FDVD	8.97	8.97	9.08
RAD, ADDA	0.75	0.75	0.80
MPY	5.77	5.77	5.83
DIV	2.52-5.79	2.52-5.79	5.58-5.83
ADM	2.30	2.45	2.64
JFM÷	0.75	0.75	0.80
JFM+	1.40	1.40	1.60
EXC (IM)	1.30	1.30	1.50
EXC	2.12	2.12	2.42

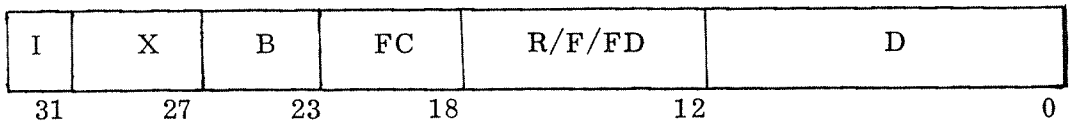
2 MEMORY REFERENCE INSTRUCTION

2.1 Introduction

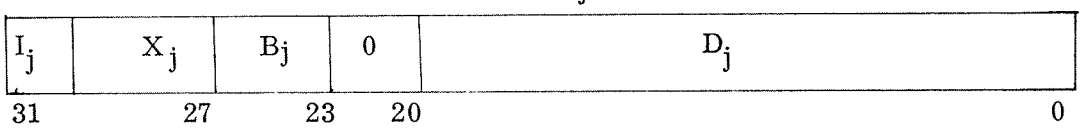
The memory reference instructions have in common that the execution of an instruction involves calculation of a memory address. In some of the memory reference instructions the memory address is used as operand, i.e. jumps and remote execution.

2.2 Instruction Word

Instruction Word, IW



Indirect Address Word of level j, IAW_j



IW : Instruction Word

I : Indirect addressing flag

X : Index register designator

B : Base register designator

FC : Function Code

R/F/FD General register, floating register or double precision floating register designator.

D : Displacement $0 \leq D < 4096$

IAW_j : Indirect Address Word at level j

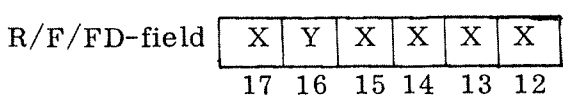
I_j : Indirect addressing flag in IAW_j $I_0 = I$

X_j : Index register designator in IAW_j

B_j : Base register designator in IAW_j

D_j : Displacement in IAW_j, $0 \leq D_j < 1\ 048\ 576$

Note: If floating operation is specified, the R/F/FD field is used to select one of 32 floating registers. Floating registers are denoted Fi for single precision, and FDi for double precision, $0 \leq i < 16$ or $32 \leq i < 48$. The register number is given by the X-bits in the field. The Y-bit is used to determine whether it is a single or double precision operation. Y = 0 gives single precision, Y = 1 gives double precision.



2.3 Memory Addressing

All memory reference instructions calculate a memory address. The result of this calculation is called the effective address, Ea.

The memory addressing may be direct or indirect. When the addressing is direct, Ea is calculated without memory reference (except for the read of the instruction). When indirect addressing, the CPU has to reference the memory for operands to the calculation of Ea. We say that each memory reference, which is done to calculate Ea in one instruction, requires one level of indirect addressing. The maximum number of levels possible in the NORD-50 is 16 in addition to the read of the instruction.

The algorithm for calculating Ea is as follows: (symbols defined in 2.2)

$$Ea_0 = (R_B) + (R_X) + D$$

if $I = 0$, then $Ea = Ea_0$

$$Ea_1 = (R_{B_1}) + (R_{X_1}) + D_1, \quad IAW_1 = (Ea_0)$$

$$Ea_j = (R_{B_j}) + (R_{X_j}) + D_j, \quad IAW_j = (Ea_{j-1})$$

$$Ea = EA_j, \quad I_j = 0, \quad 0 < j \leq 16, \quad I_i = 1, \quad 0 \leq i < j$$

Note: Each level of indirect addressing adds one "read instruction time" to the execution time.

A store operation to the location immediately following the storing instruction will have a special effect. The storing will be executed correctly, but the next instruction will be the old content of the location.
Example: STR REG, *+1.

Indirect addressing is not allowed for the instruction EXC.

2.4 Instruction List

FC

0 Refer to inter register and argument instructions.

1 RTJ Return Jump

(R) : = (P) + 1 AND

(P) : = Ea

2 EXC Remote Execute (Two instructions)

R = 0, (IR) : = (Ea)

R = 1, (IR) : = Ea

3 MIN Memory Increment

(Ea) : = (Ea) + (R) + 1 AND

IF (Ea) : = 0 THEN (P) : = (P) + 2

ELSE (P) : = (P) + 1

4 CRG Skip if register is greater or equal memory word

IF (R) $\geq$ (Ea) THEN (P) : = (P) + 2

ELSE (P) : = (P) + 1

5 CRL Skip if register is less than memory word

IF (R) $<$ (Ea) THEN (P) : = (P) + 2

ELSE (P) : = (P) + 1

6 CRE Skip if register and memory word is equal

IF (R) = (Ea) THEN (P) : = (P) + 2

ELSE (P) : = (P) + 1

7 CRD Skip if register not equal memory word

IF (R) $\neq$ (Ea) THEN (P) : = (P) + 2

ELSE (P) : $\neq$ (P) + 1

FC		
8	JRP	Jump if register is positive IF (R) $\geq$ 0 THEN (P) : = Ea ELSE (P) : = (P) + 1
9	JRN	Jump if register is negative IF (R) $<$ 0 THEN (P) : = Ea ELSE (P) : = (P) + 1
10	JRZ	Jump if register is zero IF (R) = 0 THEN (P) : = Ea ELSE (P) : = (P) + 1
11	JRF	Jump if register is non-zero IF (R) $\neq$ 0 THEN (P) : = Ea ELSE (P) : = (P) + 1
12	JPM	Modify register by its modification register. If register is positive, then jump (R) : = (R) + (M_R) AND IF (R) $\geq$ 0 THEN (P) : = Ea ELSE (P) : = (P) + 1
13	JNM	Modify register by its modification register. If register is negative, then jump. (R) : (R) + (M_R) AND IF (R) $<$ 0 THEN (P) : = Ea ELSE (P) : = (P) + 1
14	JZM	Modify register by its modification register. If register is zero, then jump. (R) : = (R) + (M_R) AND IF (R) = 0 THEN (P) = Ea ELSE (P) = (P) + 1

FC		
15	JFM	Modify register by its modification register. If contents of register is non-zero, then jump. (R) : = (R) + (M_R) IF (R) $\neq$ 0 THEN (P) : = Ea ELSE (P) : = (P) + 1
16	ADD	Add content of memory word to register (R) : = (R) + (Ea)
17	SUB	Subtract content of memory word from register (R) : = (R) - (Ea)
18	AND	Make "logical AND" between memory word and register. Result in register. (R) : = (R) $\wedge$ (Ea)
19	LDR	Load content of memory word to register (R) : = (Ea)
20	ADM	Add contents of register to memory word (Ea) : = (Ea) + (R)
21		Instruction set aside for future extensions.
22	XMR	Exchange contents of registers and memory word. (R) : = (Ea) AND (Ea) : = (R)

FC		
23	STR	Store register in memory word (Ea) : = (R)
24	MPY	Multiply contents of register and memory word and set result in register. (R) : = (R) * (Ea) Note: Upper 32 bit part of 64 bit product is in OR.
25	DIV	Divide content of register with content of word. Quotient is in register. Remainder is in the remainder register. (R) : = (R) / (Ea) Note: Remainder is in RR.
26	LDD	Load double precision register with content of double precision word. (FD) : (Ea, Ea + 1)
27	STD	Store content of double precision register in double precision word (Ea, Ea + 1) : = (FD)
28	FAD	Add content of floating word to content of floating register (F) : = (F) + (Ea,) R/F/FD-field X 0 X X X X
29	FADD	Add content of double precision floating word to content of double precision floating registers (FD) : = (FD) + (Ea, Ea + 1) R/F/ FD-field X 1 X X X X
30	FSB	Subtract content of floating word from content of floating register (F) : = (F) - (Ea R/F/FD-field X 0 X X X X

FC		
31	FSBD	Subtract content of double precision floating word from content of double precision floating register. (FD) : = (FD) - (Ea, Ea + 1) R/F/FD-field [X|1|X|X|X|X]
32	FMU	Multiply content of floating word by content of floating register. Result in floating register. (F) : = (F) * (Ea) R/F/FD-field [X|0|X|X|X|X]
33	FMUD	Multiply content of double precision floating word by content of double precision floating register. Result in double precision floating register. (FD) : = (FD) * (Ea, Ea + 1) R/F/FD-field [X|1|X|X|X|X]
34	FDV	Divide content of floating register by content of floating word. Result in floating register. (F) : = (F) / (Ea) R/F/FD-field [X|0|X|X|X|X]
35	FDVD	Divide content of double precision floating register by content of double precision floating word. Result in double precision floating register. (FD) : = (FD) / (Ea, Ea + 1) R/F/FD-field [X|1|X|X|X|X]

3 INTER-REGISTER INSTRUCTIONS

3.1 Introduction

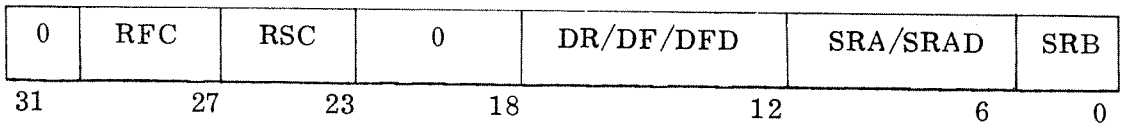
The inter-register instructions normally affect the general registers, or floating registers organized from the general registers. In their general form, the instructions specify two operand registers, called source register A and source register B, and one result register, called the destination register.

The usage of source registers and destination registers is completely general. It is possible to specify any registers as source registers and destination registers.

Some of the inter-register instructions have special forms which will be described under the specific instruction.

3.2 Instruction Layout

Instruction Word, IW



IW	Instruction Word
RFC	Inter-Register Function Code
RSC	Inter-Register Sub-function Code
SRA	Source Register A designator
SRAD	Double precision Source Register A designator
SRB	Source Register B designator
DR	Destination Register designator
DF	Destination Floating Register designator
DFD	Destination Double Precision Floating Register designator

The following abbreviations will also be used:

BN	Bit Number
SC	Shift Count

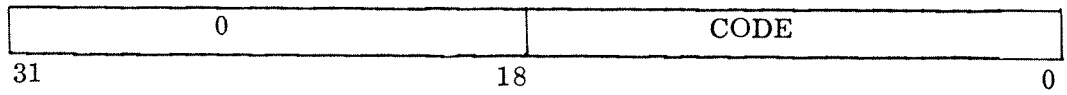
Note: For all instructions except FIX, FIXD, FIR and FIRD, the destination (DF/DFD) determines whether it is a single or double precision operation, i.e. whether SRA and, if used, SRB are single or double precision.
For the FIX, FIXD, FIR and FIRD instructions SRA/SRAD determines whether it is single or double precision.

3.3 Inter-register Instruction Descriptions

3.3.1 STOP (Monitor Call)

Stop NORD-50 CPU and call monitor program in NORD-10.

IW



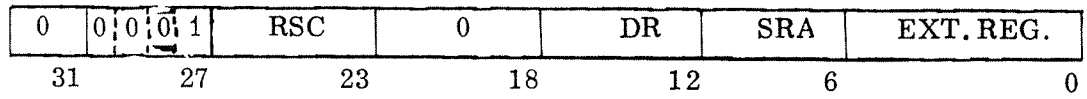
$(IW)_{18-31} = 0$

$(IW)_{0-17}$ = code used to specify monitor function

(P) : = (P)

3.3.2 RIO : Register Input/Output

Transfer content of a specified external register to a specified register, or opposite.



RSC = 0 Read

RSC = 1 Write

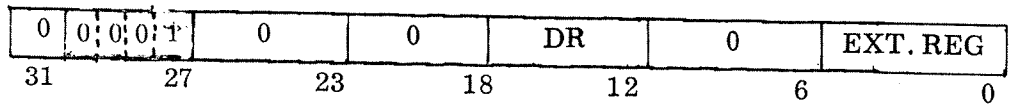
EXT. REG: External Register

2 : Overflow, OR-register

3 : Remainder, RR-register

3.3.3 RIN : Register Input

Transfer content of specified external register to a specified register.



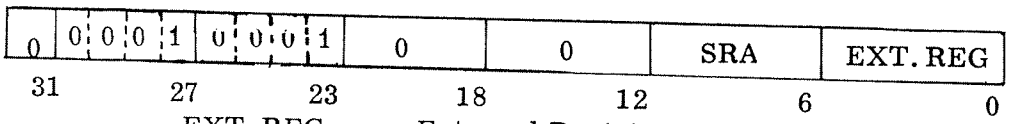
EXT. REG: External Register

2: Overflow, OR-register

3: Remainder, RR-register

3.3.4 ROUT: Register Output

Transfer content of specified register to specified external register.



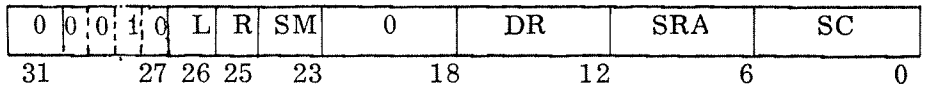
EXT. REG: External Register

2: Overflow, OR-register

3: Remainder, RR-register

3.3.5 SHR : Shift Register

Copy content of SR to DR and shift DR.



RSC	divided in three fields, L, R, and SM
L=1	Shift left
R=1	Shift right
SM	Shift mode
SM=0	Rotate register
SM=1	Rotate register
SM=2	Arithmetical shift. For left shift same as logical. For right shift bit 31 is copied to each bit shifted in (sign extension).
SM=3	Logical shift. The bits which are shifted out of the word are lost and zeroes are put in the other end.

$0 \leq SC \leq 31$

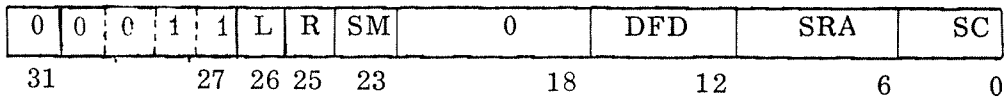
(DR) : = (SR) AND

(DR) : = (DR) * 2^{SC}

If both L = 1 and R = 1, then the right will be performed first and then the left shift.

3.3.6 SHD : Shift Double Precision Register

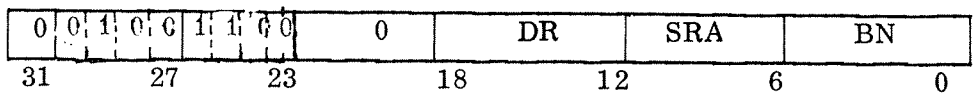
Copy SRA to DFD and shift DFD.



Description as for SHR, Section 3.3.5, except $0 \leq SC \leq 63$.

3.3.7 BST : Bit Set

Copy SRA to DR and set specified bit in DR to one.

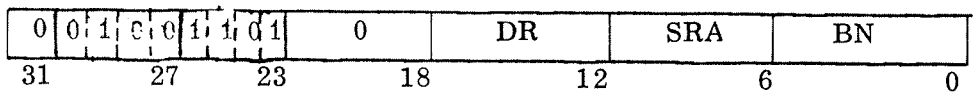


(DR) : = (SRA) AND

(DR_{BN}) : = 1

3.3.8 BCM : Bit Complement

Copy SRA to DR and complement specified bit in DR.

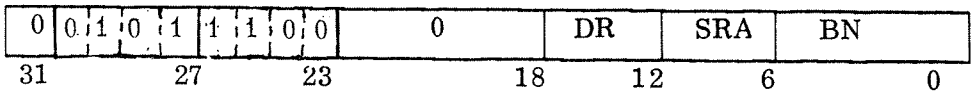


(DR) : = (SRA) AND

(DR_{BN}) : = $\overline{(SRA_{BN})}$

3.3.9 BCL : Bit Clear

Copy SRA to DR and set specified bit in DR to zero.

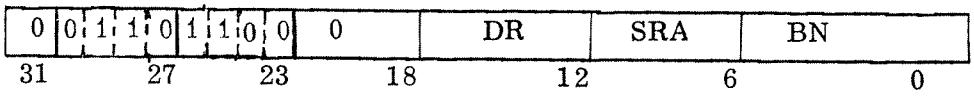


(DR) : = (SRA) AND

(DR_{BN}) : = 0 ELSE

3.3.10 BSZ : Bit Skip on Zero

Copy SRA to DR and skip if specified bit in DR is zero.



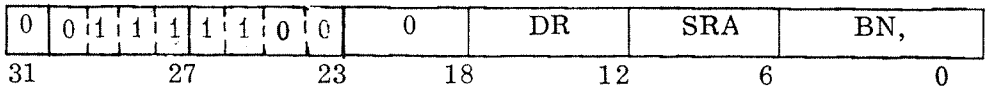
(DR) : = (SRA) AND

IF (DR_{BN}) = 0 THEN (P) : = (P) + 2

ELSE (P) : = (P) + 1

3.3.11 BSO : Bit Skip on One

Copy SRA to DR and skip if specified bit in DR is one.



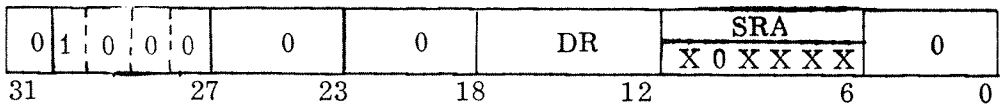
(DR) : = (SRA) AND

IF (DR_{BN}) = 1 THEN (P) : = (P) + 2

ELSE (P) : = (P) + 1

3.3.12 FIX : Convert Floating to Integer

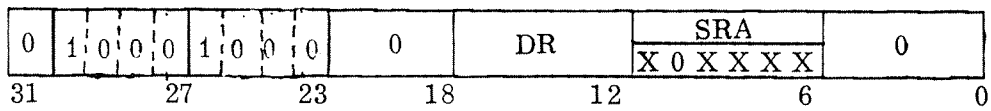
Convert floating point number in SRA to truncated integer and place result in DR.



(DR) : = Truncated integer (SRA)

3.3.13 FIR : Convert Floating to Rounded Integer

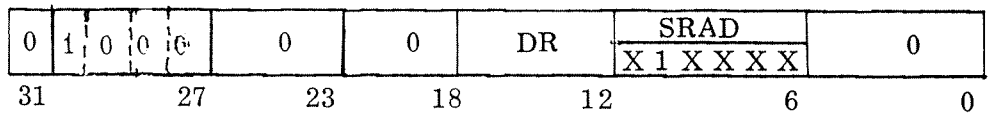
Convert floating point number in SRA to the nearest integer and place result in DR.



(DR) : = truncated integer (SRA) + 0.5

3.3.14 FIXD : Convert Double Precision Floating to Integer

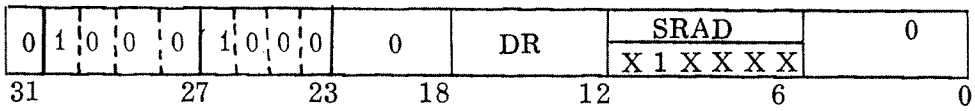
Convert double precision floating point number in SRAD to truncated integer and place result in DR.



(DR) : = truncated integer (SRAD)

3.3.15 FIRD : Convert Double Precision Floating to Rounded Integer

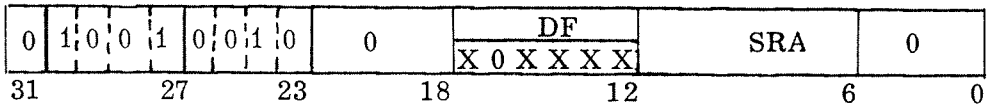
Convert double precision floating point number in SRAD to nearest integer and place result in DR



(DR) : = truncated integer (SRAD) + 0.5

3.3.16 FLO : Convert Integer to Floating

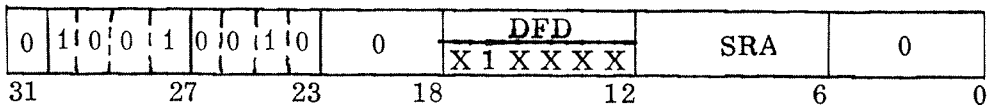
Convert the integer number in SRA to a floating point number in DF. The result is not rounded.



(DF) : = float (SRA)

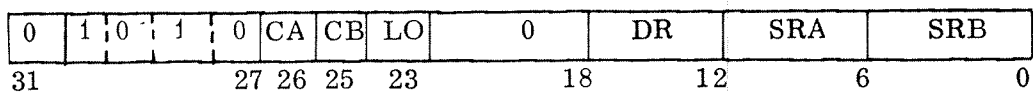
3.3.17 FLOD : Convert Integer to Double Precision Floating

Convert the integer number in SRA to a double precision floating point number in DFD. There is no need for rounding due to the word length.



(DFD) : = float (SRA)

3.3.18 LRO : Logical Register Operation



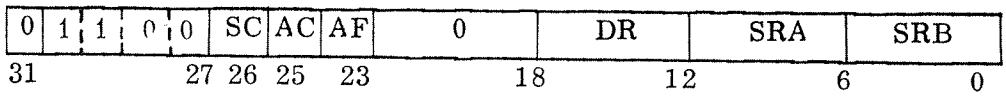
The RSC is divided in 3 fields, CA, CB and LO, If CA is set, the operand in SRA is complemented before the logical operation is performed. CB has the same effect on SRB.

The complementation of the operands does not affect the original contents of SRA and SRB.

RSC =	CA	CB	LO	
	0	0	0	(DR) : = 0
	0	0	1	(DR) : = (SRA) $\wedge$ (SRB)
	0	0	2	(DR) : = (SRA) $\not\vee$ (SRB)
	0	0	3	(DR) : = (SRA) $\vee$ (SRB)
	0	1	0	(DR) : = 0
	0	1	1	(DR) : = (SRA) $\wedge$ $\overline{\text{(SRB)}}$
	0	1	2	(DR) : = (SRA) $\not\vee$ $\overline{\text{(SRB)}}$
	0	1	3	(DR) : = (SRA) $\vee$ $\overline{\text{(SRB)}}$
	1	0	0	(DR) : = 0
	1	0	1	(DR) : = $\overline{\text{(SRA)}}$ $\wedge$ (SRB)
	1	0	2	(DR) : = $\overline{\text{(SRA)}}$ $\not\vee$ (SRB)
	1	0	3	(DR) : = $\overline{\text{(SRA)}}$ $\vee$ (SRB)
	1	1	0	(DR) : = 0
	1	1	1	(DR) : = $\overline{\text{(SRA)}}$ $\wedge$ $\overline{\text{(SRB)}}$
	1	1	2	(DR) : = $\overline{\text{(SRA)}}$ $\not\vee$ $\overline{\text{(SRB)}}$
	1	1	3	(DR) : = $\overline{\text{(SRA)}}$ $\vee$ $\overline{\text{(SRB)}}$

3.3.19 The inter-register instruction with RFC = 11 is saved for future usage.

3.3.20 IRO : Inter-Register Arithmetic



AF : Arithmetical function

AF = 0 (DR) : = (SRA) + (SRB)

1 (DR) : = (SRA) - (SRB)

2 (DR) : = (SRA) * (SRB)

3 (DR) : = (SRA) / (SRB)

The inter-register add and subtract may affect or be affected by a one bit register called the Carry Bit, CB.

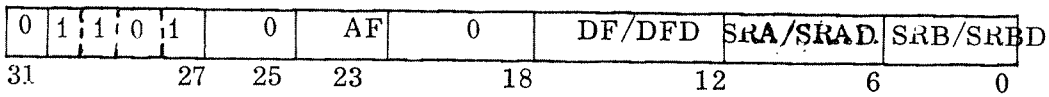
If SC = 1, the content of CB will be set to its proper value after the arithmetical operation. If AC = 1 the content of CB will be added to the result of the arithmetical function.

The carry bit may be used to simulate multiple precision arithmetic. Using SC = 1 will have the same effect as extending the operands with one bit containing zero. The result register will be extended by one bit containing one or zero, according to the arithmetical condition.

Integer multiply and divide will affect the auxiliary registers. OR will contain the upper 32 bit part of the 64 bit product. RR will contain a 32 bit remainder after an integer divide.

In all integer arithmetic, overflow conditions may occur. If so specified, an overflow condition may cause a monitor call. Overflow occurs when the result of an arithmetical operation is in size greater than $\pm (2^{23} - 1)$.

3.3.21 FRO : Floating Point Arithmetic



AF : Arithmetical function

AF = 0 (DF/DFD) : = (SRA) + (SRB)

1 (DF/DFD) : = (SRA) - (SRB)

2 (DF/DFD) : = (SRA) * (SRB)

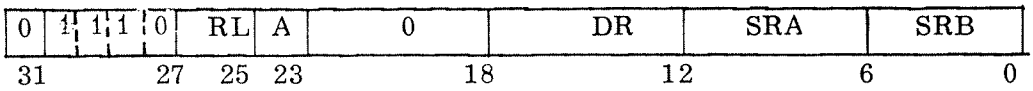
3 (DF/DFD) : = (SRA) / (SRB)

Cfr. Section 1.8, Floating Point Arithmetic.

SINGLE PRECISION : (DF), (SRA) and (SRB) : = (X 0 X X X X)

DOUBLE PRECISION : (DF), (SRA) and (SRB) : = (X 1 X X X X)

3.3.22 IRS : Integer Register Skip



The inter-register skip is a general arithmetical instruction followed by a relation between the result and zero.

AF : Arithmetical function

AF = 0 Add

1 Subtract

RL : Relation between result and zero

RL = 0 result ≥ 0

1 result < 0

2 result $= 0$

3 result $\neq 0$

(DR) : = (SRA) arithmetical operation (SRB) AND

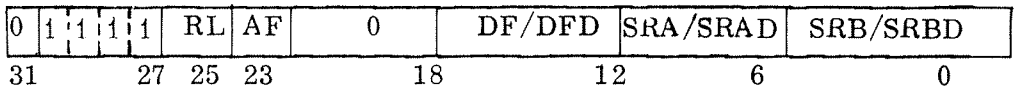
IF relation THEN (P) : = (P) + 2

ELSE (P) : = (P) + 1

Normally, DR will be affected, but if general register 0 is specified as DR, no destination register is affected. However, the relation on the result will still be effective.

When DR = 0 and AF = 1, the IRS operation is a traditional skip.

3.3.23 FRS: Floating Register Skip



The description of FRS is the same as for IRS, Section 3.3.22, except that the arithmetic is performed on floating registers in floating point mode.

Single precision : (DF), (SRA) and (SRB) : = (X 0 X X X X)
Double precision: (DF), (SRA) and (SRB) : = (X 1 X X X X)

Floating register skip instructions causing an underflow condition gives zero as result, and the next instruction is selected according to this. Additionally, the underflow flag is set.

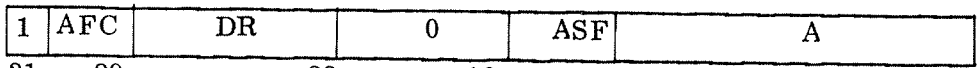
4 ARGUMENT INSTRUCTION

4.1 Introduction

Typical for the argument instruction is that one of the operands is contained in the instruction itself; it is called the argument, A. The other operand is contained in one of the general registers.

The argument is a 16 bit positive number. Before the specified operation takes place, the argument is extended to a 32 bit number with the one 16 upper bits all equal to zero.

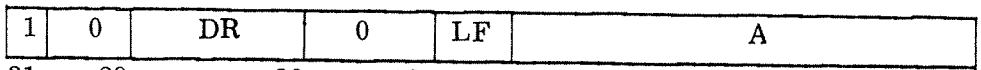
4.2 Argument Instruction Layout



A	Argument, 16 bits
AFC	Argument instruction function code
DR	Destination register
ASF	Argument instruction sub-function code

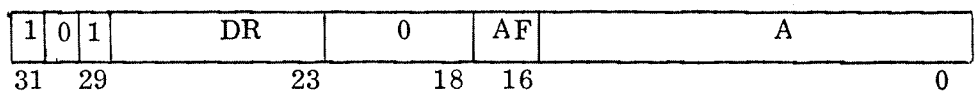
4.3 Instruction Description

4.3.1 DLR : Direct Logical Operation



LF	Logical Function
LF = 0	(DR) : = 0
1	(DR) : = (DR) $\forall$ A
2	(DR) : = (DR) $\wedge$ A
3	(DR) : = (DR) $\vee$ A

4.3.2 DAR : Direct Arithmetic Function



AF Arithmetical Function

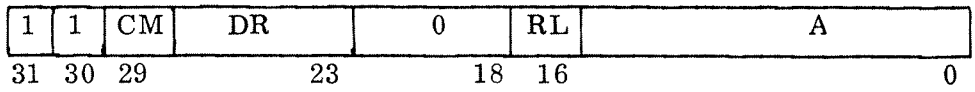
AF = 0 (DR) : = A

1 (DR) : = -A (2's complement

2 (DR) : = + A

3 (DR) : = (DR) - A

4.3.3 DSK : Direct Skip



CM Complement flag (2's complement)

RL Relation designator

RF = 0	CM = 0	IF (DR) ≥ A THEN (P) : = (P) + 2 ELSE (P) : = (P) + 1	DDP
1	0	IF (R) < A THEN (P) : = (P) + 2 ELSE (P) : = (P) + 1	DDN
2	0	IF (DR) = A THEN (P) : = (P) + 2 ELSE (P) : = (P) + 1	DDZ
3	0	IF (DR) ≠ A THEN (P) : = (P) + 2 ELSE (P) : = (P) + 1	DDF
0	1	IF (DR) ≥ -AR THEN (P) : = (P)+2 ELSE (P) : = (P) + 1	DSP
1	1	IF (DR) < -AR THEN (P) : = (P) +2 ELSE (P) : = (P) + 1	DSN
2	1	IF (DR) = -AR THEN (P) : = (P) + 2 ELSE (P) : = (P) + 1	DSZ
3	1	IF (DR) ≠ -AR THEN (P) : = (P) + 2 ELSE (P) : = (P) + 1	DSF

SUMMARY OF INSTRUCTIONS

MEMORY REFERENCE INSTRUCTIONS

+++
+

MNEMONIC	ACTION	SECTION 2.4
RTJ	Return jump	1
EXC	Remote execute	2
MIN	Memory increment	3
CRG	Skip if (R) $\geq$ (Ea)	4
CRL	Skip if (R) $<$ (Ea)	5
CRE	Skip if (R) = (Ea)	6
CRD	Skip if (R) $\neq$ (Ea)	7
JRP	Jump if (R) ≥ 0	8
JRN	Jump if (R) < 0	9
JRZ	Jump if (R) $= 0$	10
JRF	Jump if (R) $\neq 0$	11
JPM	Modify (R) and jump if (R) ≥ 0	12
JNM	Modify (R) and jump if (R) < 0	13
JZM	Modify (R) and jump if (R) $= 0$	14
JFM	Modify (R) and jump if (R) $\neq 0$	15
ADD	Add (Ea) to (R)	16
SUB	Subtract (Ea) from (R)	17
AND	Logical AND between (Ea) and (R)	18
LDR	Load (R) with (Ea)	19
ADM	Add (R) to (Ea)	20
XMR	Exchange (Ea) and (R)	22
STR	Store (R) in (Ea)	23
MPY	Multiply (R) by (Ea)	24
DIV	Divide (R) by (Ea)	25
LDD	Load (FD) with (Ea, Ea + 1)	26
STD	Store (FD) in (Ea, Ea + 1)	27
FAD	Add (Ea) to (F)	28
FADD	Add (Ea, Ea + 1) to (FD)	29

MEMORY REFERENCE INSTRUCTIONS (contd)

MNEMONIC	ACTION	SECTION 2.4
FSB	Subtract (Ea) from (F)	30
FSBD	Subtract (Ea, Ea + 1) from (FD)	31
FMU	Multiply (F) by (Ea)	32
FMUD	Multiply (FD) by (Ea, Ea + 1)	33
FDV	Divide (F) by (Ea)	34
FDVD	Divide (FD) by Ea, Ea + 1)	35

INTER REGISTER OPERATIONS

1 SHIFT INSTRUCTIONS

MNEMONIC	ACTION
SLR	Left rotational shift
SRR	Right rotational shift
SLA	Left arithmetical shift
SRA	Right arithmetical shift
SLL	Left logical shift
SRL	Right logical shift
SLRD	Left rotational double register shift
SRRD	Right rotational double register shift
SLAD	Left arithmetical double reg. shift
SRAD	Right arithmetical double reg. shift
SLLD	Left logical double reg. shift
SRLD	Right logical double reg. shift

2 MISCELLANEOUS OPERATIONS

BST	Bit Set
BCM	Bit Complement
BCL	Bit clear
BSZ	Bit skip on zero
BSO	Bit skip on one
FIX	Convert floating to integer
FIR	Convert floating to rounded integer
FIXD	Convert double precision floating to integer
FIRD	Convert double precision floating to rounded integer
FLO	Convert integer to floating
FLOD	Convert integer to double precision floating

MNEMONIC	ACTION
3	ARITHMETIC OPERATIONS
RAD	Register add
RSB	Register subtract
RMU	Register multiply
RDV	Register divide
RAF	Floating register add
RSF	Floating register subtract
RMF	Floating register multiply
RDF	Floating register divide
RAFD	Double precision floating register add
RSFD	Double precision floating reg. subtract
RMFD	Double precision floating reg. multiply
RDFD	Double precision floating reg. divide
4	TEST AND SKIP
SGR	Subtract regs. and skip if result $\geqslant 0$
ASG	Add regs. and skip if result $\geqslant 0$
SLE	Subtract regs. and skip if result < 0
ASL	Add registers and skip if result < 0
SEQ	Subtract regs. and skip if result $= 0$
ASE	Add registers and skip if result $= 0$
SUE	Subtract regs. and skip if result $\neq 0$
ASU	Add regs. and skip if result $\neq 0$
SGF	Subtract floating regs. and skip if result $\geqslant 0$
ASGF	Add floating registers and skip if result $\geqslant 0$
SLF	Subtract floating regs. and skip if result < 0
ASLF	Add floating registers and skip if result < 0
SEF	Subtract floating regs. and skip if result $= 0$
ASEF	Add floating regs. and skip if result $= 0$

MNEMONIC	ACTION	SECTION
4	TEST AND SKIP (contd)	
SUF	Subtract floating regs. and skip if result $\neq 0$	
ASUF	Add floating regs. and skip if result $\neq 0$	
SGD	Subtract double precision and skip if result $\geqslant 0$	
AGFD	Add double precision and skip if result $\geqslant 0$	
SLD	Subtract double precision and skip if result < 0	
ALFD	Add double precision and skip if result < 0	
SED	Subtract double precision and skip if result $= 0$	
AEFD	Add souble precision and skip fi result $= 0$	
SUD	Subtract double precision and skip if result $\neq 0$	
AUFD	Add double precision and skip if result $\neq 0$	
5	LOGICAL OPERATIONS	
RND	Register AND	
RNDA	Register AND, use complement of (SRA)	
RNDB	Register AND, use complement of (SRB)	
RXO	Register exclusive OR	
RXOA	Register exclusive OR, use complement of (SRA)	
RXOB	Register exclusive OR, use complement of (SRB)	
ROR	Register OR	
RORA	Register OR, use complement of (SRA)	
RORB	Register OR, use complement of (SRB)	
SZR	Set all zeroes.	

6 ARGUMENT INSTRUCTIONS

MNEMONIC	ACTION	SECTION
XORA	Exclusive or	
ANDA	And	
ORA	Or	
SETA	Set register	
SECA	Set register to complement	
ADDA	Add	
ADCA	Add complement	
DDP	Skip if (DR) $\geqslant$ ARG	
DDN	Skip if (DR) $<$ ARG	
DDZ	Skip if (DR) $=$ ARG	
DDF	Skip if (DR) $\neq$ ARG	
DSP	Skip if (DR) $\geqslant$ -ARG	
DSN	Skip if (DR) $<$ -ARG	
DSZ	Skip if (DR) $=$ -ARG	
DSF	Skip if (DR) $\neq$ -ARG	

0	0 0 0 0	0 0 0 0	0 0 0 0 0	MONITOR CODE			STOP (monitor call)
0	0 0 0 1	0 0 0 W	0 0 0 0 0	DR	SRA	EXT. REG. NO.	RIO
0	0 0 1 0	L R SM	0 0 0 0 0	DR	SRA	SHIFT COUNT	SHR
0	0 0 1 1	L R SM	0 0 0 0 0	DFD	SRAD	SHIFT COUNT	SHD
0	0 1 0 0	1 1 0 C	0 0 0 0 0	DR	SRA	BITNO	BST, BCM
0	0 1 0 1	1 1 0 0	0 0 0 0 0	DR	SRA	BITNO	BCL
0	0 1 1 0	1 1 0 0	0 0 0 0 0	DR	SRA	BITNO	BSZ
0	0 1 1 1	1 1 0 0	0 0 0 0 0	DR	SRA	BITNO	BSO
0	1 0 0 0	R 0 0 0	0 0 0 0 0	DR	SRA/SRAD	0 0 0 0 0 0	FIX, RIR FIXD, FIRD
0	1 0 0 1	0 0 1 0	0 0 0 0 0	DF/DFD	SRA	0 0 0 0 0 0	FLO, FLOD
0	1 0 1 0	LO	0 0 0 0 0	DR	SRA	SRB	LRO
0	1 0 1 1		0 0 0 0 0				Not used
0	1 1 0 0	AF	0 0 0 0 0	DR	SRA	SRB	IRO
0	1 1 0 1	0 0 AF	0 0 0 0 0	DF/DFD	SRA/SRAD	SRB/SRBD	FRO
0	1 1 1 0	RL AF	0 0 0 0 0	DR	SRA	SRB	IRS
0	1 1 1 1	RL AF	0 0 0 0 0	DF/DFD	SRA/SRAD	SRB/SRBD	FRS
I	X	B	0 0 0 0 1	R	D		RTJ
I	X	B	0 0 0 1 0	R	D		EXC
I	X	B	0 0 0 1 1	R	D		MIN
I	X	B	0 0 1 0 0	R	D		CRG
I	X	B	0 0 1 0 1	R	D		CRL
I	X	B	0 0 1 1 0	R	D		CRE
I	X	B	0 0 1 1 1	R	D		CRD
I	X	B	0 1 0 0 0	R	D		JRP
I	X	B	0 1 0 0 1	R	D		JRN
I	X	B	0 1 0 1 0	R	D		JRZ
I	X	B	0 1 0 1 1	R	D		JRF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1		X				B			0	1	1	0	0		R								D								JPM
I		X				B			0	1	1	0	1		R								D								JNM
I		X				B			0	1	1	1	0		R								D								JZM
I		X				B			0	1	1	1	1		R								D								JFM
I		X				B			1	0	0	0	0		R								D								ADD
I		X				B			1	0	0	0	1		R								D								SUB
I		X				B			1	0	0	1	0		R								D								AND
I		X				B			1	0	0	1	1		R								D								LDR
I		X				B			1	0	1	0	0		R								D								ADM
I		X				B			1	0	1	0	1		R								D								Not used
I		X				B			1	0	1	1	0		R								D								XMR
I		X				B			1	0	1	1	1		R								D								STR
I		X				B			1	1	0	0	0		R								D								MPY
I		X				B			1	1	0	0	1		R								D								DIV
I		X				B			1	1	0	1	0		FD								D								LDD
I		X				B			1	1	0	1	1		FD								D								STD
I		X				B			1	1	1	0	0		F/FD								D								FAD/FADD
I		X				B			1	1	1	0	1		F/FD								D								FSB/FSBD
I		X				B			1	1	1	1	0		F/FD								D								FMU/FMUD
I		X				B			1	1	1	1	1		F/FD								D								FDV/FDVD
I_j		X_j				B_j			0	0	0												D_j								Ind. addr. word
I	0	0			DR				0	0	0	0	0	LF									A								DLR
1	0	1			DR				0	0	0	0	0	AF									A								DAR
1	1	CM			DR				0	0	0	0	0	RL									A								DSK

APPENDIX A

ROUNDING RULES

for floating addition, floating subtraction and rounded convert to integer in NORD-50.

The philosophy with the rounding is to get a result that is as correct as possible from the information contained in the operands.

Instructions that result in floating add or subtract, either memory reference instructions, inter register instructions or floating register skip tests, are divided into two groups, one that results in a physical addition and one that results in a physical subtraction. The convert instruction will be treated separately. Note that negative integers are in 2's complement.

ADDITION

The result of a physical addition is fairly easy to round. If the most significant bit that is shifted out from the mantissa is a one, one is added in the least significant bit in the mantissa.

To explain this more clearly, I will give some examples with a floating number containing a 4 bit mantissa.

A	B	C	D	E	F	G

The mantissa bits of a normalized number are B, C, D and E, and if it is not zero, B will be a 1. In NORD-50 however, this bit is masked away in the operands and result, but is used by the floating arithmetic. Zero is represented by all exponent bits equal 0. In this way, one bit is „gained", but zero-tests are slightly more complex; and there is no way of telling that a number is not normalized.

If two floating operands are added in NORD-50, it is done in the following way.

1) The exponent part of the numbers is compared, and the mantissa of the operand with the least exponent is shifted the same number of places to the right as the exponent difference indicates. Let operand M have exponent 5 and operand N have exponent 2, then the difference is 3. The example after shifting, before addition will then be:

		M_B	M_C	M_D	M_E			
PLUS		0	0	0	N_B	N_C	N_D	
SUM	P_A	P_B	P_C	P_D	P_E	P_F	P_G	

Note that only two of the shifted out bits are taken care of. One is enough for addition and convert instruction, and two is sufficient for subtraction.

2) The operands are added and the not rounded sum P is made. In the partial sum P, the most significant 1 in the mantissa will be either P_B or P_A. If it is P_A, the mantissa must be shifted one place to the right, and a 1 is added in position P_E before shifting to generate a carry into P_D and thus get a rounded answer if P_E is a 1. If P_B is the most significant 1, N_C (P_F) is added to P_E to make a rounded answer.

3) The next step is to add 1 in the P_E position if either P_A or N_C is 1.

P_A	P_B	P_C	P_D	P_E
S_A	S_B	S_C	S_D	S_E

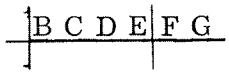
4) The mantissa for the sum S is then selected as S_B, S_C, S_D and S_E if $S_A = 0$, or as S_A, S_B, S_C and S_D if $S_A = 1$. If $S_A = 1$, one is added to the exponent.

SUBTRACTION

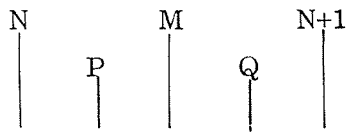
First of all, let us see how many of the shifted out bits we have to take care of to maintain the amount of information that the mantissa can hold, or take care of the information contained in the operands.

If the exponents for the operands are the same, or the difference between them is 1, none or one bit is shifted out, and the only thing we have to do is to shift that bit in again or round the result due to that bit if no shift is necessary after the subtraction. If the difference between the exponents is 2 or more, none or one left shift can occur to normalize the mantissa after the subtraction. If no leftshift is necessary, the last shifted out bit determines how to round. If one leftshift is necessary to necessary to normalize the result, the second last shifted out bit determines how to round when the subtraction includes the two last shifted out bits, and the result in the last shifted out bit position is shifted in again.

Let us look at this more closely. We have the example with mantissa bits B, C, D and E.



and the solution in bit E corresponds to the distance between N and N+1 in the next figure.



Now, if the least operand is shfited a maximum of one place to the right before subtraction, we cannot give the answer with a better solution than corresponding to the difference between N and M, where M is the mid-point between N and N+1, because the operands do not give more information.

In the other case where the least operand is shifted 2 or more places to the right, we know that the maximum number of leftshift, after subtraction, is one, which gives us the maximum solution corresponding to the distance between N and M, and if the mantissa is not shifted after subtraction, the maximum solution corresponds to the distance between N and N+1.

As the arithmetic uses the two last shifted out bits, we can look at a figure

	N	P	M	Q	N+1
Last shifted out bit		1		0	
2 last shifted out bit	1	0	1	0	
Answer, mantissa bits before rounding &normalization)	N	N	N	N+1	
Answer, the 2 extra bits	01	10	11	00	

First of all, let us look at the point M, If all the shifted out bits but the last are zero, the subtraction should give us a result equal to M. But now, if no leftshifts are done after the subtraction, we have no means to give the answer M, but have to give either N or N+1 as the result. The distance from N to M equals the distance from M to N+1 so the amount of information lost is just the same whichever we choose. As already mentioned, only the two last shifted out bits are taken care of. If now, one or more of the less significant shifted out bits is 1, which means that the exact result should lie in the region between P and M, N world be a more correct answer than N +1. Therefore, if we get a result that seems to be M, but this has to be rounded, the value N is chosen.

To go further on, the same philosophy is used for the case where one left-shift takes place after subtraction, Results in the region N and including P are rounded to N, from P, and including Q gives M as the result, and the rest from Q to N+1 gives N+1 as the result.

CONVERT TO INTEGER

If a convert to integer instruction is specified with rounding, will the result be rounded awat from 0 when the last shifted out bit is 1. 3.5 gives 4 and -3.5 gives -4.

FLOATING OVERFLOW

Floating overflow may be the result of a physical addition. The overflow flag is set, and the result will be the greatest positive or negative number dependent on the sign of the operands. All bits except the sign bit, are forced to one.

FLOATING UNDERFLOW

Floating underflow may be the result if a small number is subtracted from another small number. The result can be so small that it is impossible to represent it by a NORD-50 floating word. In this case, the underflow flag is set, and the result is set to zero.

INTEGER OVERFLOW

Integer overflow from the floating arithmetic may be the result of a convert to integer instruction. The overflow flag is set, and the result will be the greatest positive or negative integer.

A/S NORSK DATA-ELEKTRONIKK
Lørenveien 57, Oslo 5 - Tlf. 21 73 71

COMMENT AND EVALUATION SHEET

NORD-50 Reference Manual
February 1976

Publ. No. ND-05.003.01

In order for this manual to develop to the point where it best suits your needs, we must have your comments, corrections, suggestions for additions, etc. Please write down your comments on this pre-addressed form and post it. Please be specific wherever possible.

FROM: _____

_____

_____

– we make bits for the future

NORSK DATA A.S BOX 4 LINDEBERG GÅRD OSLO 10 NORWAY PHONE: 39 16 01 TELEX: 18661