NORD 50
FUNCTIONAL DESCRIPTION

NORSK DATA A.S

NORD 50
FUNCTIONAL DESCRIPTION

REVISION RECORD

Revision	Notes
11/77	Original Printing

Publ. No. ND-05.007.01
November 1977

NORSK DATA A.S.

Lørenveien 57, Postboks 163 Økern, Oslo 5, Norway

MAIN CONTENTS

++++

++++

S E C T I O N I

INTRODUCTION

I INTRODUCTION

NORD-50 is a 32 bit special purpose computer designed to be a slave processor to the general purpose NORD-10/S computer.

The NORD-50 consists of 3 19" racks located in a cabinet identical to that of NORD-10/S.

Of the three racks named, the A, B and C rack, the NORD-50 CPU occupies the B-rack with 32 modules (148 x 156 mm).

The A and C racks contain the "external arithmetic" executing instructions not performed in the NORD-50 CPU.

NORD-50 may be connected to physically the same memory as NORD-10/S (located in the NORD-10/S cabinet) via one port in the multiport memory system (shared memory), or to a separate multiport memory system as a private memory.

The NORD-50 can execute the following instructions:

1. Data move instructions
2. Arithmetic/logic instructions
3. Data Manipulation instructions
4. Sequence or branch instructions

The NORD-50 does not have

- Input/Output instructions
- Interrupt
  and
- Memory Management instructions.

These functions are controlled by the NORD-10/S.

S E C T I O N I I

NORD-50 FLOATING POINT NUMBERS REPRESENTATION

II FLOATING POINT NUMBERS

A floating point number F may be represented as follows:

$$F = M \cdot 2^{E}$$

The number is represented by two groups of bits, the EXPONENT E and the MANTISSA M.

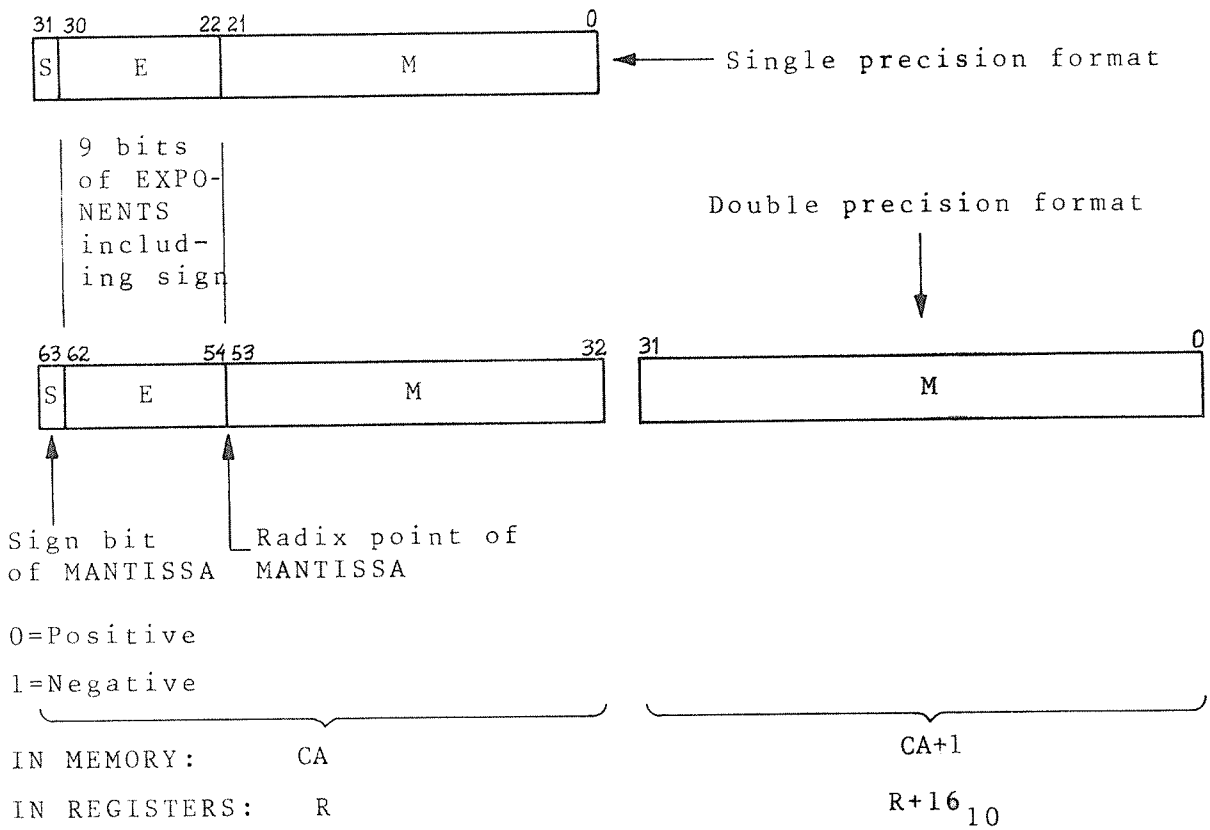
The floating point value is the product of its mantissa and base 2 raised to the power of its exponent.

NB! In IBM terminologi the exponent is called the characteristic and the mantissa is called the fraction part of the floating number.

FLOATING POINT FORMAT

NORD-50 operates with 2 floating point formats, one with 22 bits to represent the mantissa (single precision) and one with 54 bits to represent the mantissa (double precision).

The exponent in both formats consists of 9 bits.



R+(R+16) = Floating Double-Precision Registers (FDR)

CA means Calculated Address.

MANTISSA

The sign of the mantissa is contained in bit 31 for single- and bit 63 for double-precision numbers.

The binary radix point is assumed to be at the left of the highest order bit in the mantissa, so the magnitude of the mantissa is always less than 1.

The most significant bit in the mantissa is always set to a 1 and the floating number is then said to be normalized.

The value of a normalized mantissa is then between $\frac{1}{2}$ and 1.

Because the most significant bit in the mantissa is always a 1, it needs not be represented directly and the bit can be omitted to give a better resolution of the mantissa.

A floating number in memory or in floating registers does not contain this "1" bit, but the "1" bit is inserted in the External-Arithmetic before floating operations, and is removed when the floating result is returned to the CPU or memory.

EXPONENT

The 9-bits EXPONENT field allows exponent values between -255_{10} (-377_8) and $+255_{10}$ (377_8). In NORD-50 256_{10} or 400_8 is added to the exponent, leaving an offset or biased exponent.

For example, an exponent of -32_{10} would be represented by $256_{10} - 32_{10} = 224_{10}$. An exponent of $+100_{10}$ would then be $256_{10} + 100_{10} = 356_{10}$.

Because of the bias, the correspondence between actual values and coded representation is as follows:

REPRESENTATION

BINARY:		OCTAL:	ACTUAL VALUE DECIMAL:
Bit:30	22 Single prec.		
62	54 Double prec.		
EXPONENT OVERFLOW			
111 111 111		777	+255
111 111 110		776	+254
	RANGE	402	+ 2
100 000 001	OF	401	+ 1
	USABLE		
100 000 000	EXPONENTS	400	0
011 111 111		377	- 1
011 111 110		376	- 2
000 000 001		1	-255
000 000 000		0	-256
EXPONENT UNDERFLOW			

If the actual value of the exponent is equal to -256_{10} (-400_8), meaning a total floating number of less than 2^{-256} (since the mantissa is between $\frac{1}{2}$ and 1) the floating number will be assumed to be 0, regardless of the sign or mantissa bits.

As the table indicates: When the exponent is decreased towards its minimum representable value, the binary number in the exponent approaches zero. Zero has 0 in all bits, both in the exponent and the mantissa.

An exponent overflow occurs if the exponent exceeds +255, and an exponent underflow occurs if the exponent is less than -255.

EXAMPLES OF FLOATING NUMBER REPRESENTATION:
(single precision)

BIT: 31	30	29 28 27	26 25 24	23 22	21 20	19 18 17	0		
0	1	111	111	111	11	111	MAX: $1.15792 : 10^{77}$		
0	1	000	001	111	11	111	$0.999 \cdot 2^{7}$	127.999	
0	1	000	001	110	00	000	$0.5 \cdot 2^{7}$	64	
0	1	000	001	101	11	111	$0.999 : 2^{6}$	63.999	
0	1	000	001	100	00	000	$0.5 \cdot 2^{6}$	32	
0	1	000	001	011	11	111	$0.999 \cdot 2^{5}$	31.999	
0	1	000	001	010	00	000	$05 \cdot 2^{5}$	16	
0	1	000	001	001	11	111	$0.999 \cdot 2^{4}$	15.999	
0	1	000	001	000	00	000	$0.5 \cdot 2^{4}$	8	
0	1	000	000	111	11	111	$0.999 \cdot 2^{3}$	7.999	
0	1	000	000	110	00	000	$0.5 \cdot 2^{3}$	4	
0	1	000	000	101	11	111	$0.999 \cdot 2^{2}$	3.999	
0	1	000	000	100	00	000	$0.5 \cdot 2^{2}$	2	
0	1	000	000	011	11	111	$0.999 \cdot 2^{1}$	1.999	
0	1	000	000	010	00	000	$0.5 \cdot 2^{1}$	1	
0	1	000	000	001	11	111	$0.99 \cdot 2^{0}$	0.999	
0	1	000	000	000	00	000	$0.5 \cdot 2^{0}$	0.5	
0	0	111	111	111	11	111	$0.999 \cdot 2^{-1}$	0.4999	
0	0	111	111	110	00	000	$0.5 \cdot 2^{-1}$	0.25	
0	0	111	111	101	11	111	$0.999 \cdot 2^{-2}$	0.2499	
0	0	111	111	100	00	000	$0.5 \cdot 2^{-2}$	0.125	
0	0	111	111	011	11	111	$0.999 \cdot 2^{-3}$	0.1249	
0	0	111	111	010	00	000	$0.5 \cdot 2^{-3}$	0.0625	
0	0	111	111	001	11	111	$0.999 \cdot 2^{-4}$	0.06249	
0	0	111	111	000	00	000	$0.5 \cdot 2^{-4}$	0.03125	
0	0	000	000	010	00	000	MIN: $8.63617 \cdot 10^{-78}$		
0	0	000	000	000	00	000	0		
1	0	000	000	010	00	000	MIN: $-8.63617 \cdot 10^{-78}$		
1	1	111	111	111	11	111	MAX: $-1.15792 \cdot 10^{77}$		

Exponent

Mantissa

SIGN BIT

BIAS BIT

Hidden "1" bit in mantissa

S E C T O N I I I

N O R D 5 0 C P U

DETAILED CONTENTS
++++++++++++++++

III.1 NORD-50 CPU - INTRODUCTION

The NORD-50 CPU contains the main registers and arithmetic, the communication registers NORD-10/NORD-50, memory address and data lines and line driver/receivers for external arithmetic.

These functions are organized on three different circuit boards, each handling four bits:

Address Arithmetic	1501
Register	1502
Arithmetic Buffer	1503

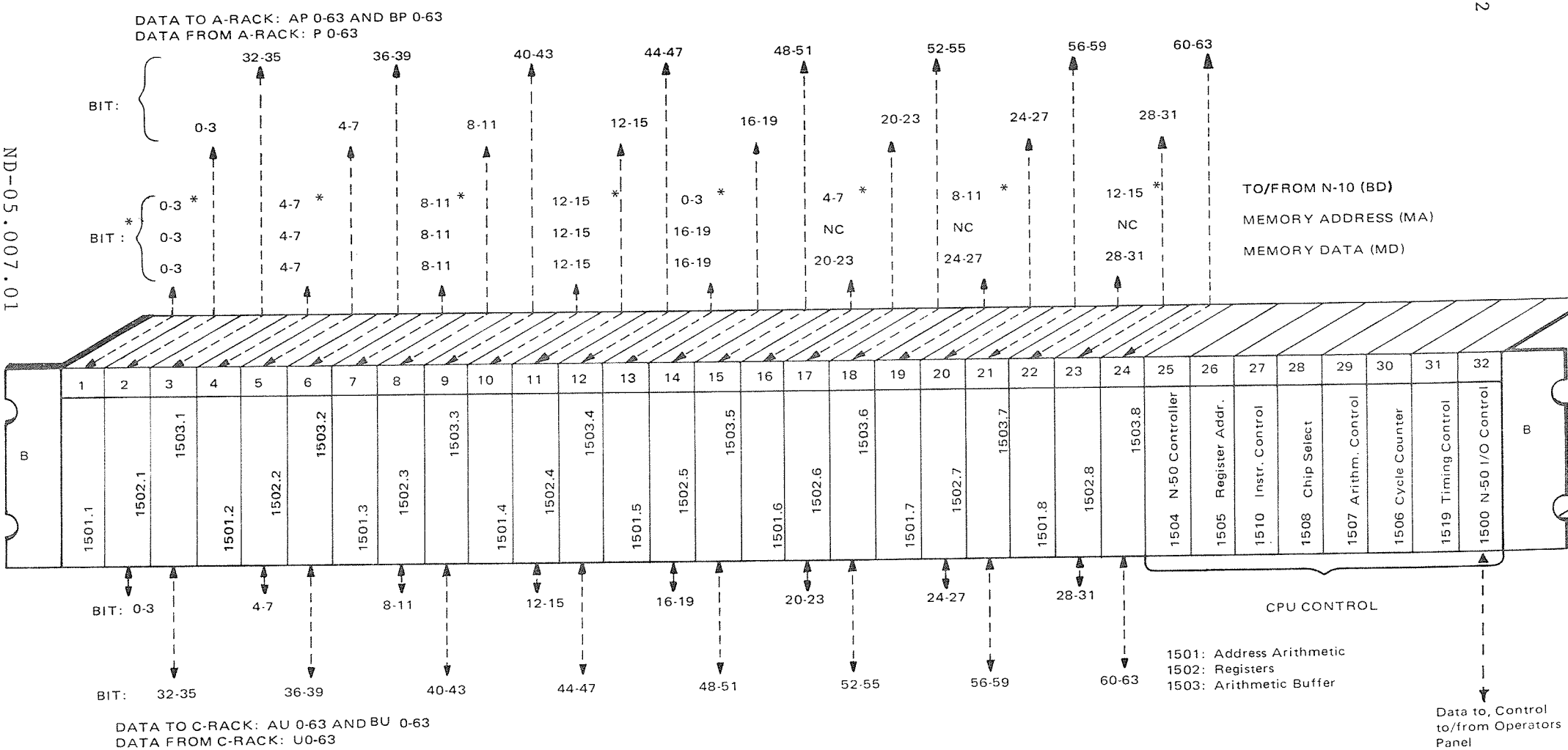
The 32 bit CPU thus uses eight of each board, making a total of 24 boards.

The timing and control section of the CPU uses eight different boards:

NORD-50 I/O Control	1500
NORD-50 Controller	1504
Register Address	1505
Cycle Counter	1506
Arithmetic Control	1507
Chip Select	1508
Instruction Control	1510
Timing Control	1519

Figure III.1.1 illustrates the data flow between NORD-50 CPU (B-rack) and the external units.

NORD-50 CPU (B-Rack) Data Flow To/From External Units



* *These bits are connected to C28-32 to give a 1-1 connection to N-10 and Memory.*

Figure III.1.1

III.2 NORD-10/NORD-50 CONNECTIONS

NORD-10 looks upon NORD-50 as an I/O device connected to the NORD-10 I/O system.

There are two NORD-50 interface modules in NORD-10, the 1071 card for data and control and the 1532.II card for address.

The communication between NORD-10 and NORD-50 in addition to common memory takes place on two differential lines, one for data (16 bits) + control (4 bits) and one for I/O address (6 bits).

At start-up time the start address and stop conditions are transferred to NORD-50. When the NORD-50 stops, a Status Register indicates the stop reason. The Status Register is available on the communication lines.

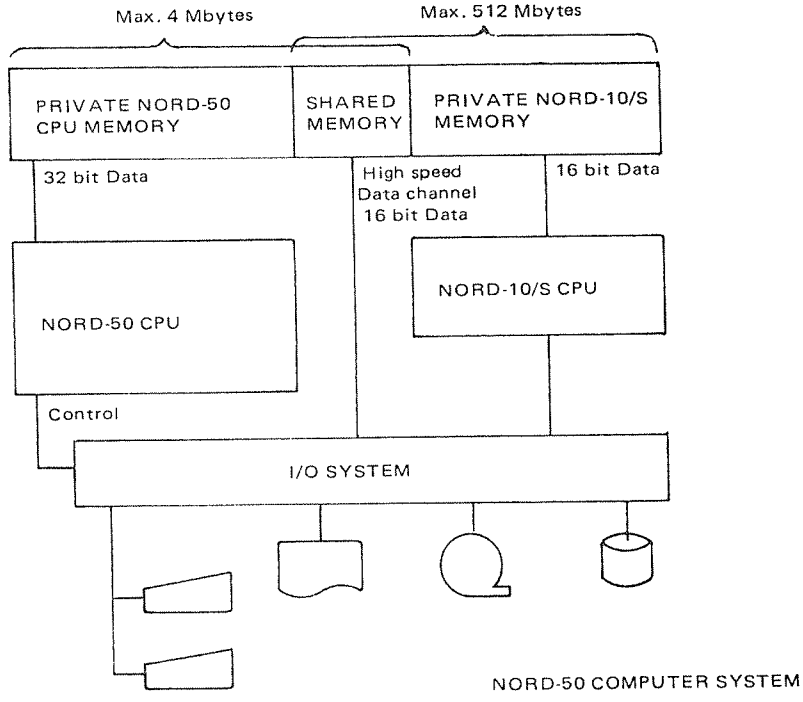
If the interrupt system in the NORD-10 is turned on and the NORD-50 interface is enabled for interrupts, the NORD-10 may execute in parallell with the NORD-50.

Only when an interrupt occures will the NORD-10 be engaged to identify the interrupt source.

In NORD-50, three modules are taking care of the communication with NORD-10, the 1532.II, 1531 and the 1532.I module located in the C-rack.

III.3 NORD-50 MEMORY CONNECTIONS

NORD-50 may be connected to physically the same memory as NORD-10 via one port in the multiport memory system (shared memory) or to a separate multiport memory system as a private memory.



NORD-50 COMPUTER SYSTEM

As an option, up to 8 high speed static memory modules may be installed in the free positions in the C-rack as private NORD-50 memory (Maximum 32K x 32 bits).

NORD-50 supplies differential address lines of 20 bits + 2 control signals to the port via a 1 to 1 cable.

For carrying the 32 bit data word, two cables with differential lines are used, one for bits 0-15 plus 2 parity bits, and one for bits 16-32 plus 2 parity bits.

In memory (the multiport memory system), NORD-50 occupies one port out of four. The remaining three are used by NORD-10, DMA mass-storage devices, and the third may be used by a second NORD-50.

The address area each port can see is set up by lower and upper limit switches on the address module (1083) in the multiport system. In this way, NORD-10 and NORD-50 can have locations of private or shared memory.

The priority for NORD-10 or NORD-50 requests are fixed and determined by the physical position of the data receiver/driver module and the address receiver module in the multiport rack.

The 32 bit memory word is divided between two 18 bit memory banks with identical memory addresses. The bank in the upper multiport rack takes care of bits 16-31, while the bank in the lower rack takes care of bits 0-15.

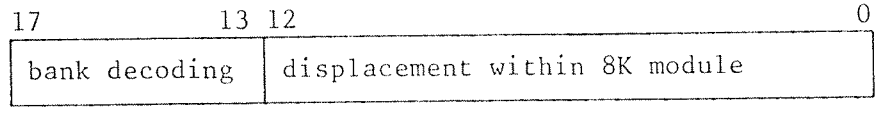
NORD-50 receives two sets of data ready/address ready signals from the memory modules in the two banks. These signals are supplied via the data module (1081) to NORD-50 where the signals are latched, waiting for the last one to appear.

Interleave

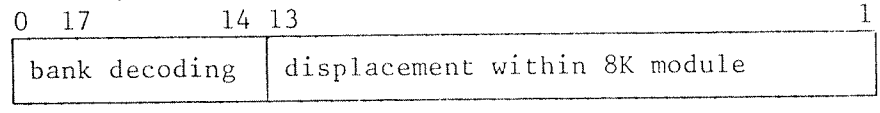
For NORD-10 to access a NORD-50 word in shared memory as two consecutive locations, the NORD-10 address has a two-way interleave.

To achieve a two-way interleave, the address bits in the address cable from NORD-10 to the multiport are shifted one position to the right:

Normal address:



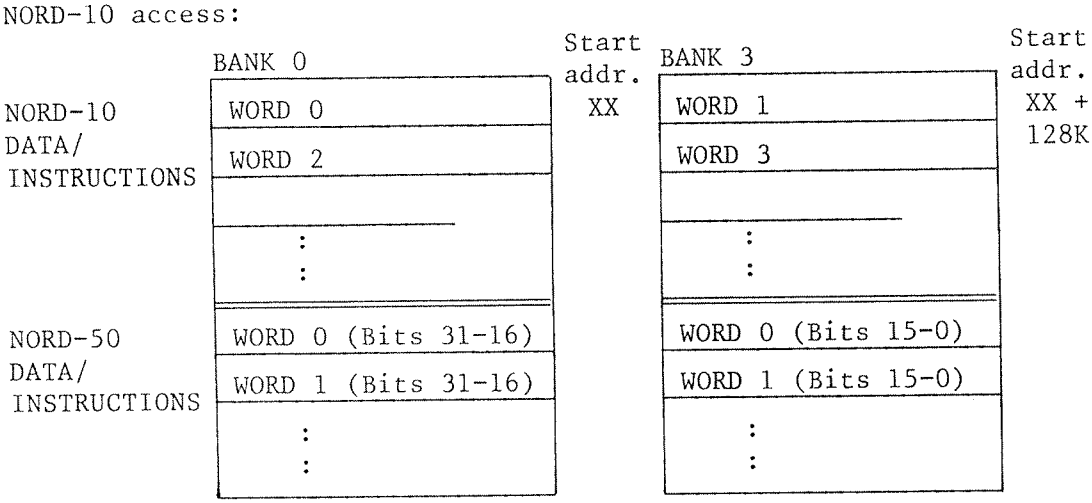
Interleaved (shifted) address:



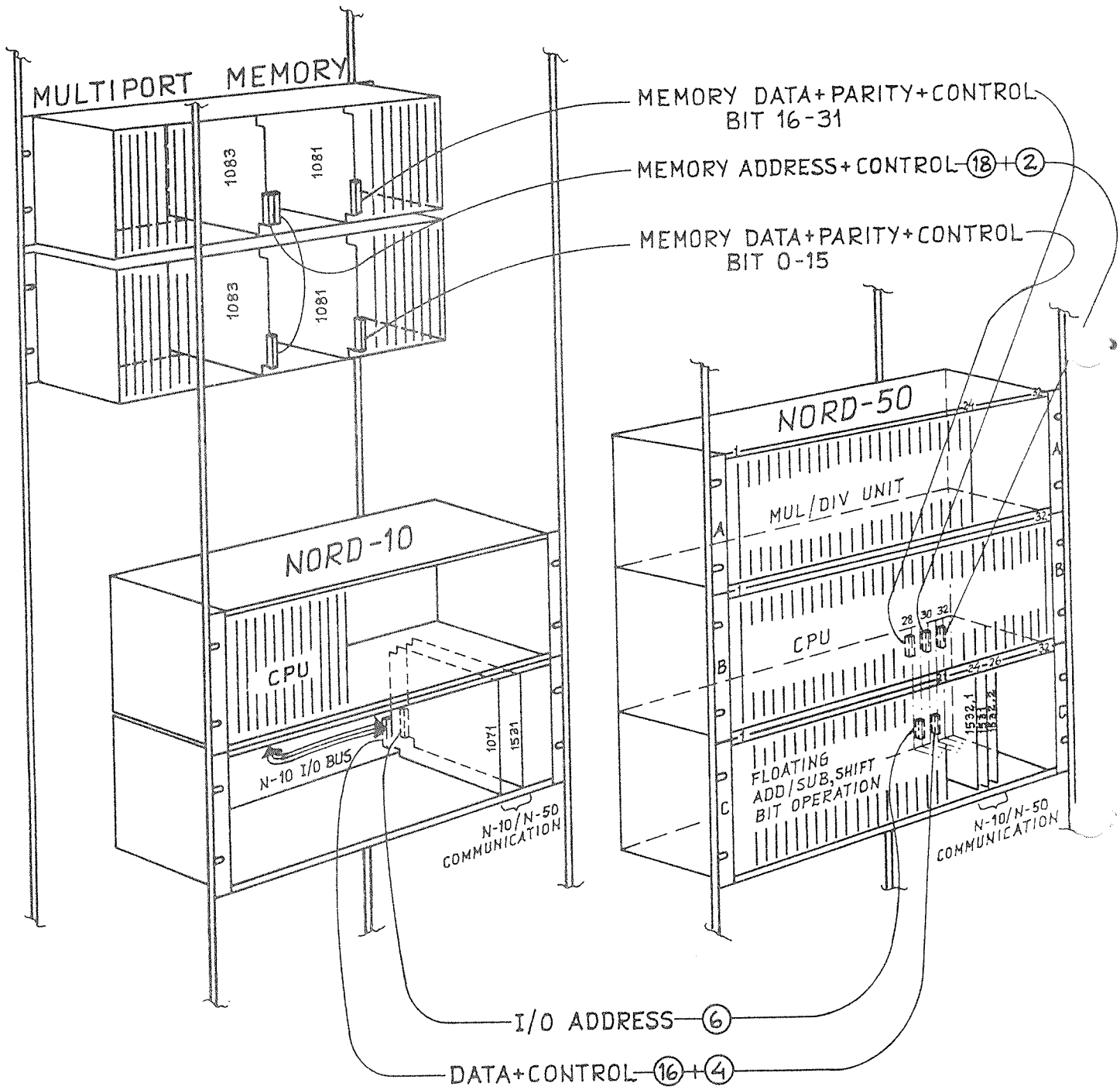
Address bit 0 being the most significant address bit, will now determine the bank selection (The difference in start address for the two banks will be 128K).

For NORD-10, two banks will alternately be accessed if consecutive addresses are issued. For NORD-50, each bank is accessed in parallel.

NORD-10 access:



An even NORD-10 address (address bit 0 = 0) will access NORD-50 bits 31-16, while an odd NORD-10 address will access NORD-50 bits 15-0.



NORD-10 - NORD-50
CONFIGURATION

ND-05.007.01

Figure III.3.1

III.4 NORD-10/NORD-50 COMMUNICATION

In NORD-10, the Write Interface Control Register (WIC) is used for starting the NORD-50 (bit 2), and the Interface Status (IS) is used for checking the state of NORD-50 (bit 3). Both registers are located on the 1071 card. Refer to figure III.4.2.

The IOX instruction IOX WIC = IOX 33 writes the contents of the A-register into the WIC register, and the IOX RIS = IOX 32 reads the IS-register to the A-register. IOX address bit 0 determines the transfer direction.

The IOX device addresses used are valid for the first NORD-50 in any NORD-10/NORD-50 system.

In NORD-50, the following 16 bit registers are located on the 1531 card:

1. The NORD-50 Modus Register, Written by IOX MOD50, IOX 31
2. The NORD-50 Status Register, Read by IOX RSTN50, IOX 30

The Modus Register is loaded as part of the NORD-50 start-up procedure, and keeps the stop conditions for the NORD-50. In addition to the stop conditions, the EXAMINE/DEPOSIT and SIMULATE MEMORY modes are set here.

The Status Register is read after the NORD-50 has stopped to find the STOP reason.

In addition to these two registers, the following 32 bit readable or writable registers are situated on the 1501 card (Address Arithmetic) in the NORD-50 CPU.

READABLE:

PC: The NORD-50 Program Counter points to the next instruction.

TA: Test Address Register, keeps the last memory reference address.

TD: Test Data Register keeps the last Data to/from memory.

SA: Simulate Address Register (only read in SIMULATE MEMORY mode).

WRITABLE:

SA: The Start Address Register First Instruction Fetch.

BP: Break Point Register 1 (Lower address limit).

BQ: Break Point Register 2 (Upper address limit).

SD: Simulate Data Register (only written into during SIMULATE MEMORY mode).

All these registers are read/written by means of two IOX instructions, each transferring 16 bit in parallel. If device address bit 1 in the IOX instruction is set to zero, the 16 least significant bits (bits 15-0) are transferred. (Transfer direction = bit 0). If bit 1 equals one, the 16 most significant bits (bits 31-16) are transferred.

More information is given in the NORD-10/NORD-50 Communication System Manual.

III.4.1 NORD-10/NORD-50 COMMUNICATION DETAILS:

As an attempt to explain the NORD-10/NORD-50 communication in som details, the signals on figure III.4.1 are listed below.

Modules in NORD-10:

1532.II

BA: NORD-10 address bus carrying the 12 lower bits of the instruction register during an IOX instruction. These bits, also referred to as the device address bits, indetifies the I/O interface register to be read or written into during the IOX instruction execution.

M: IOX address bits on differencial lines.

1071 (Also refer to figure III.4.2)

BD: NORD-10 I/O DATA bus (16 bits). Equals the A-register during an IOX output instruction (IOX address bit 0=1). Equals the contens of the IOX addressed register at the end of an IOX input instruction (BA0 = 0).

D: Data bus on differencial lines.

DEVC: COMPLETION signal from NORD-50 (1519) indicating that NORD-50 have just stopped. Resets the RUN flip-flop and generates an interrupt to level 12 if the interrupt enable flip-flop is set. Resetting the RUN flip-flop will set bit 3 in the Interface Status Register, and at the same time a light diode is turned off, indicating that NORD-50 has stopped.

DEVS: START signal to NORD-50.
Active when bit 2 is set in the Interface Control Word Register. Sets the RUN flip-flop and a light diode is lit, indicating that NORD-50 is running.

STOP: STOP signal to NORD-50 (external-stop).
Active when bit 4 is set in the Interface Control Word Register or Master Clear in NORD-10 is pushed.

STROBE: (STR) Active when device address 60-77 in IOX instruction is specified.
Start decoding NORD-10 I/O address bits IOA0,IOA1 and IOA5 to

1. Either read or write communication register in NORD-50 (IOA0)

2. To read or write the 16 least or most significant bits (IOA1)

The STROBE signal may either do as indicated, or start decoding IOA0-IOA5 to

1. Read NORD-50 Status Register (IOX 30)

2. Write NORD-50 Modus Register (IOX 31)

Modules in NORD-50:

1532.II Position C26

The control signals DEVS and STR are converted to TTL level and supplied to the 1519 modules to START or STOP NORD-50.

The control signal STR together with the NORD-10 I/O device address are converted to TTL level and supplied to the 1504 module.

1532.I Position C24

Converts the tri-state NORD-50 data bus BD to differencial lines and vice versa.

The IOA0 signal controls the data flow direction of the BD bus.

1531 Position C25

This module contains the Modus Register (set by IOX 31) and the NORD-50 Status Register (read by IOX 30).

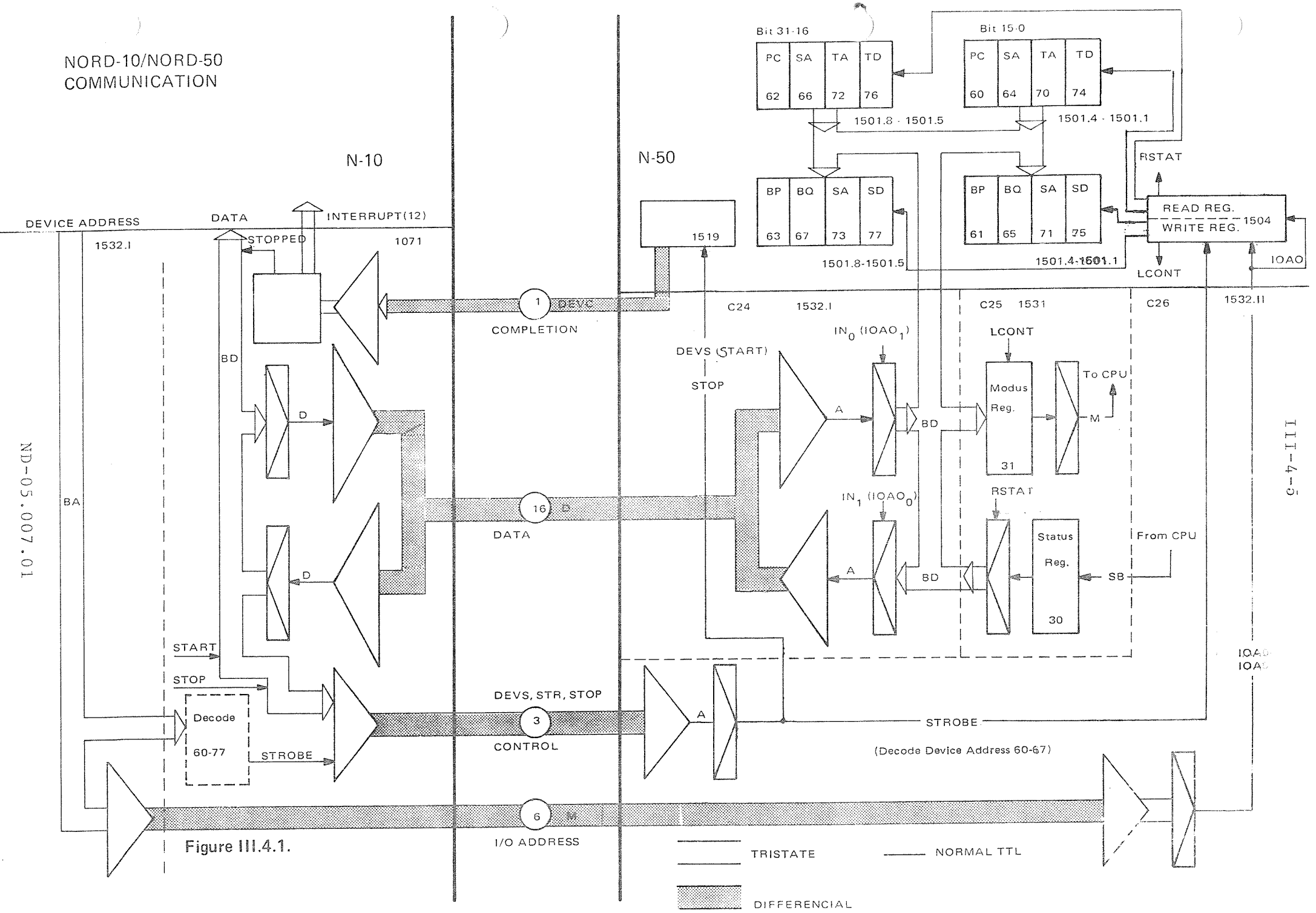


Figure III.4.1.

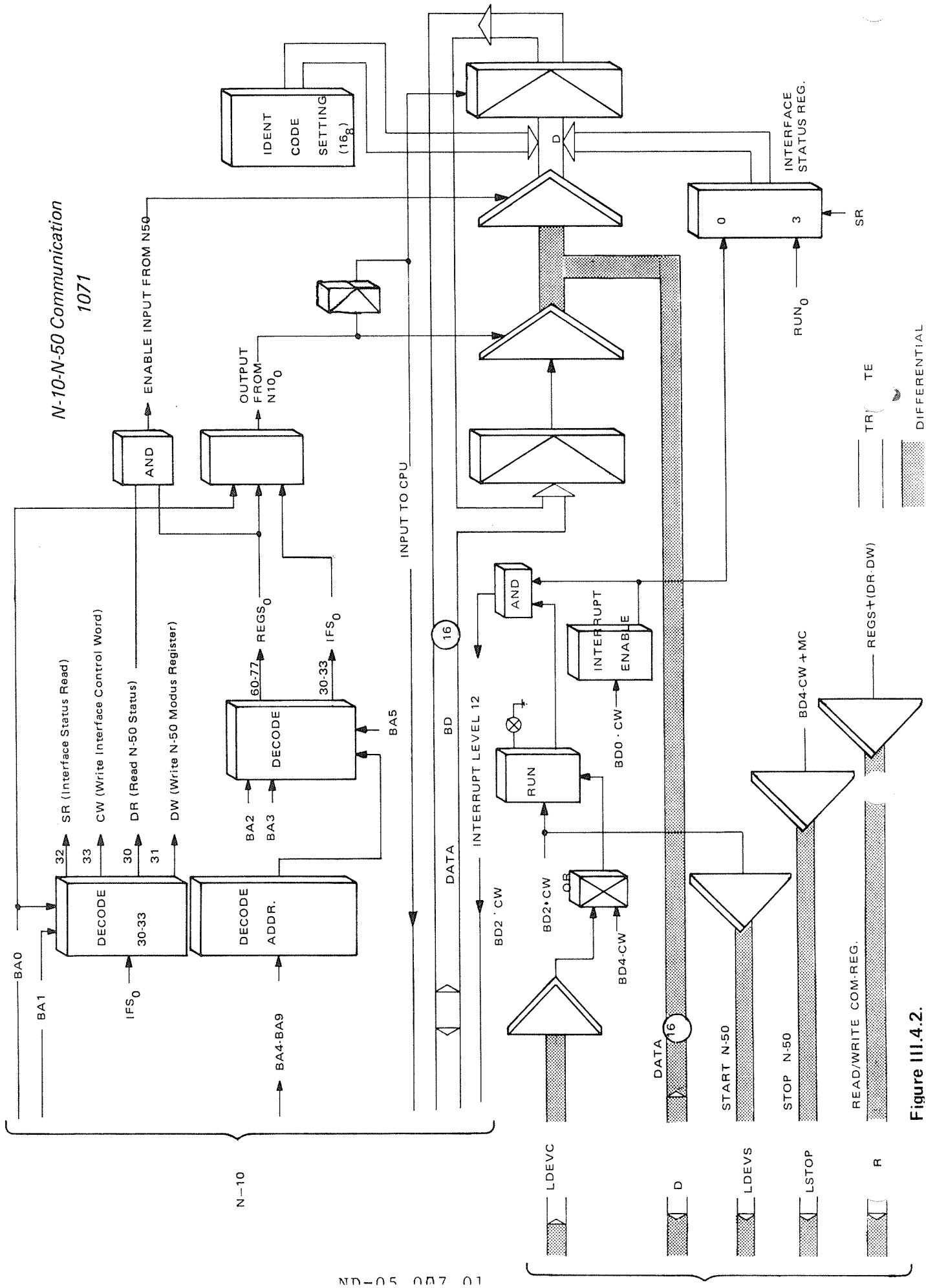


Figure III.4.2.

III.4.2 NORD-50 MEMORY EXAMINE/DEPOSIT

Refer to figure III.4.3.

When NORD-50 is in stop mode, the memory cells may be examined or written into via the A-register in the NORD-10.

The write feature is useful when executing programs in NORD-50 private memory. This is a part of the memory locations where NORD-10 has no direct access, either because of system or memory address restrictions.

DEPOSIT:

1. Set the DEPOSIT bit (bit 13) in NORD-50 Modus-register
   A = 20 000; IOX MOD50

2. Load the new data into the SD (Simulate Data) register
   A = Bit 0-15 of DATA; IOX WRSD;
   A = Bit 16-31 of DATA; IOX WLSD

3. Load the target address number into the SA (Start Address) register
   A = Bit 0-15 of ADR; IOX WRSA;
   A = Bit 16-31 of ADR; IOX WLSA

4. Activate NORD-50 to execute the write operation
   SAA 4, IOX WIC

EXAMINE:

1. Set the EXAMINE bit (bit 12) in NORD-50 Modus-register
   A = 10 000; IOX MOD50

2. Load the address number into SA as 3. in DEPOSIT

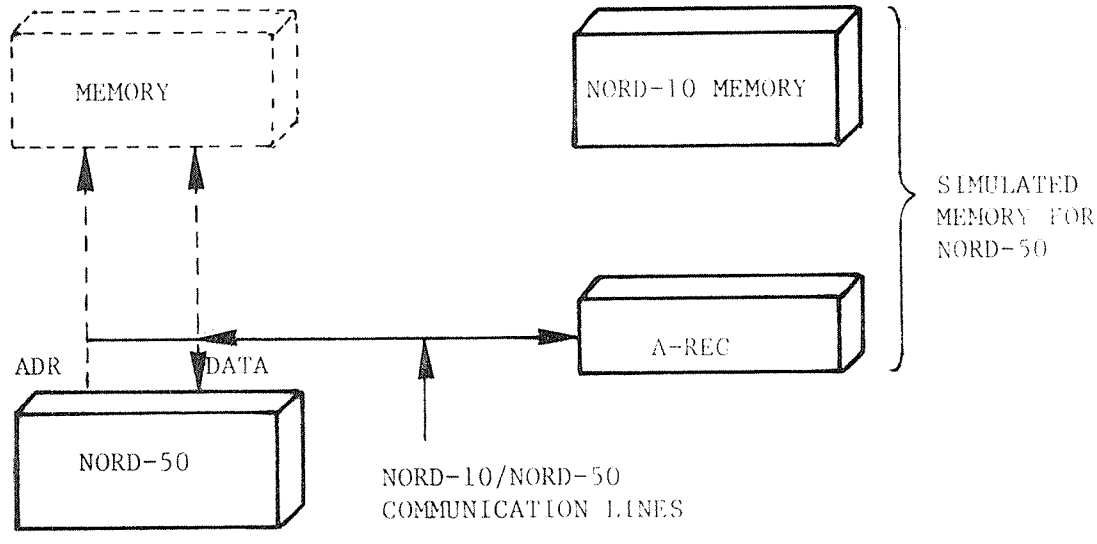
3. Activate NORD-50 to execute the read operation
   SAA 4, IOX WIC

4. Transfer data word from TD to A-register
   IOX RRTD % Bit 0-15 to A;
   IOX RLTD % Bit 16-31 to A

To DEPOSIT/EXAMINE the next memory location, the SA register must be loaded (step 3) with an address number incremented by one.

III.4.3 SIMULATED MEMORY MODE

When bits 14 and 15 are set in the NORD-50 Modus-register, the NORD-10 acts as a memory for NORD-50 by means of the communication registers and data communication lines.



The NORD-50 memory can be disconnected and NORD-10 will supply the instructions and operand data and receive data on store instructions.

In SIMULATE MEMORY mode, the communication registers are used as follows:

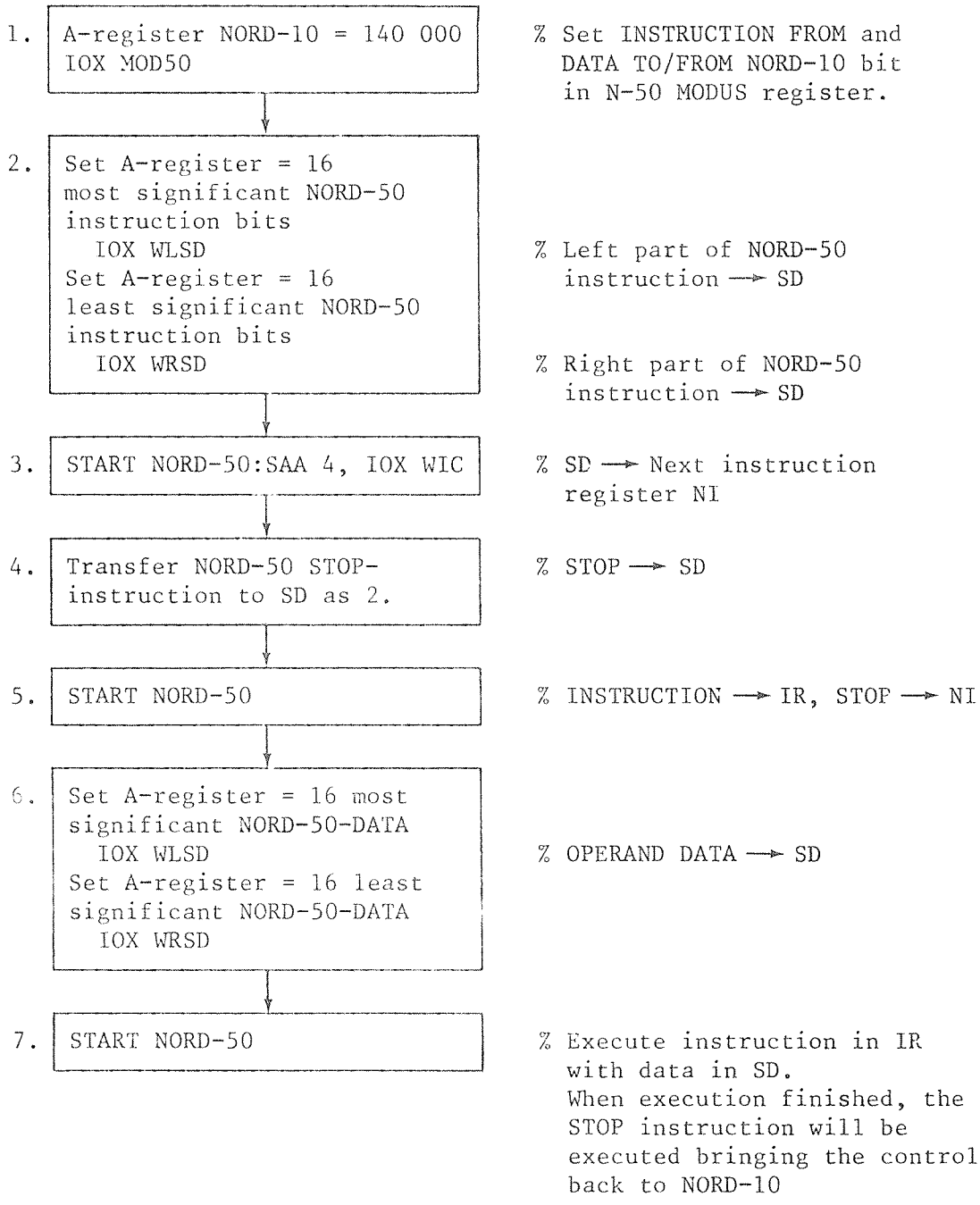
TA: Used to transmit memory addresses to NORD-10

SD: Used to transmit data to NORD-50 (from simulated memory)

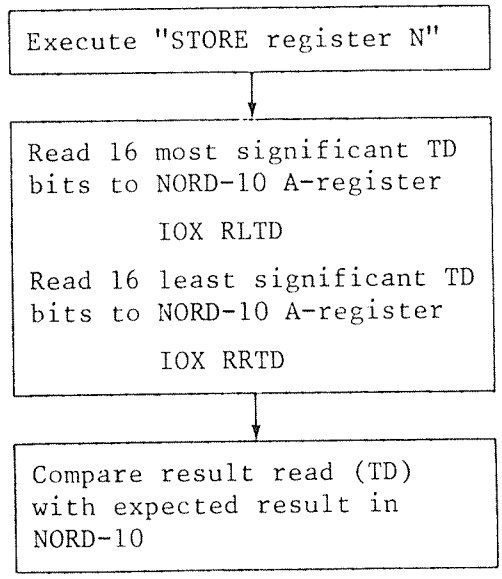
TD: Used to transmit data from NORD-50

FLOW-CHART SHOWING A MEMORY REFERENCE READ INSTRUCTION EXECUTED IN SIMULATE MEMORY MODE:

Refer to figure III.4.3.



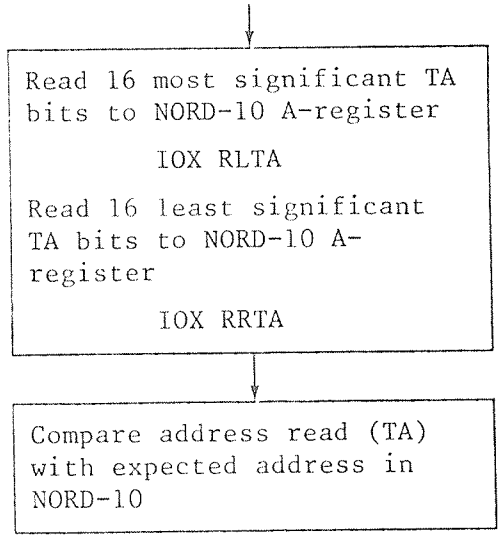
If instruction being executed does not write result to memory, but to register N, a "STORE" instruction must be executed to write result to SIMULATED MEMORY (the SUM bus is written into the TD-register).



If the instruction being executed was a sequence or branch-instruction, the data is of no interest.

In these cases, the output of the address arithmetic containing the branch address is latched in the TA-register.

This makes it possible to test all JUMP, CONDITIONAL JUMP or SKIP instructions.



N-10 - N-50 MNEMONIC DEFINITION

Mnemonic	Octal value	Function
IOX	164000	
WRBP	61	A ⟶ BP(0,15)
WLBP	63	A ⟶ BP(16,31)
WRBQ	65	A ⟶ BQ(0,15)
WLBQ	67	A ⟶ BQ(16,31)
WRSA	71	A ⟶ SA(0,15)
WLSA	73	A ⟶ SA(16,31)
WRSD	75	A ⟶ SD(0,15)
WLSD	77	A ⟶ SD(16,31)
MOD50	31	A ⟶ MODUS(0,15)

NORD-50 MODUS-REG.

Bit	Meaning
0	Stop on overflow
1	Stop on underflow
2	Stop on parity error
3	
4	Stop if BP ≤ Any Reference ≺ BQ
5	Stop if BP ≤ Program Counter ≺ BQ
6	Stop if BP ≤ Data Reference ≺ BQ
7	Stop if BP ≤ Data Store Address ≺ BQ
8	Invert limits on 4-7 (X ≺ BP or X ≥ BQ)
9	Master Clear
10	
11	
12	Read Memory Cell (EXAMINE)
13	Write Memory Cell (DEPOSIT)
14	Data to/from NORD-10
15	Instructions from NORD-10

NORD-50/NORD-10

RRPC	60	PC(0,15) ⟶ A
RLPC	62	PC(16,31) ⟶ A
RRSA	64	SA(0,15) ⟶ A
RLSA	66	SA(16,31) ⟶ A
RRTA	70	TA(0,15) ⟶ A
RLTA	72	TA(16,31) ⟶ A
RRTD	74	TD(0,15) ⟶ A
RLTD	76	TD(16,31) ⟶ A
RSTN50	30	STN5(0,15) ⟶ A

NORD-50 STATUS REGISTER

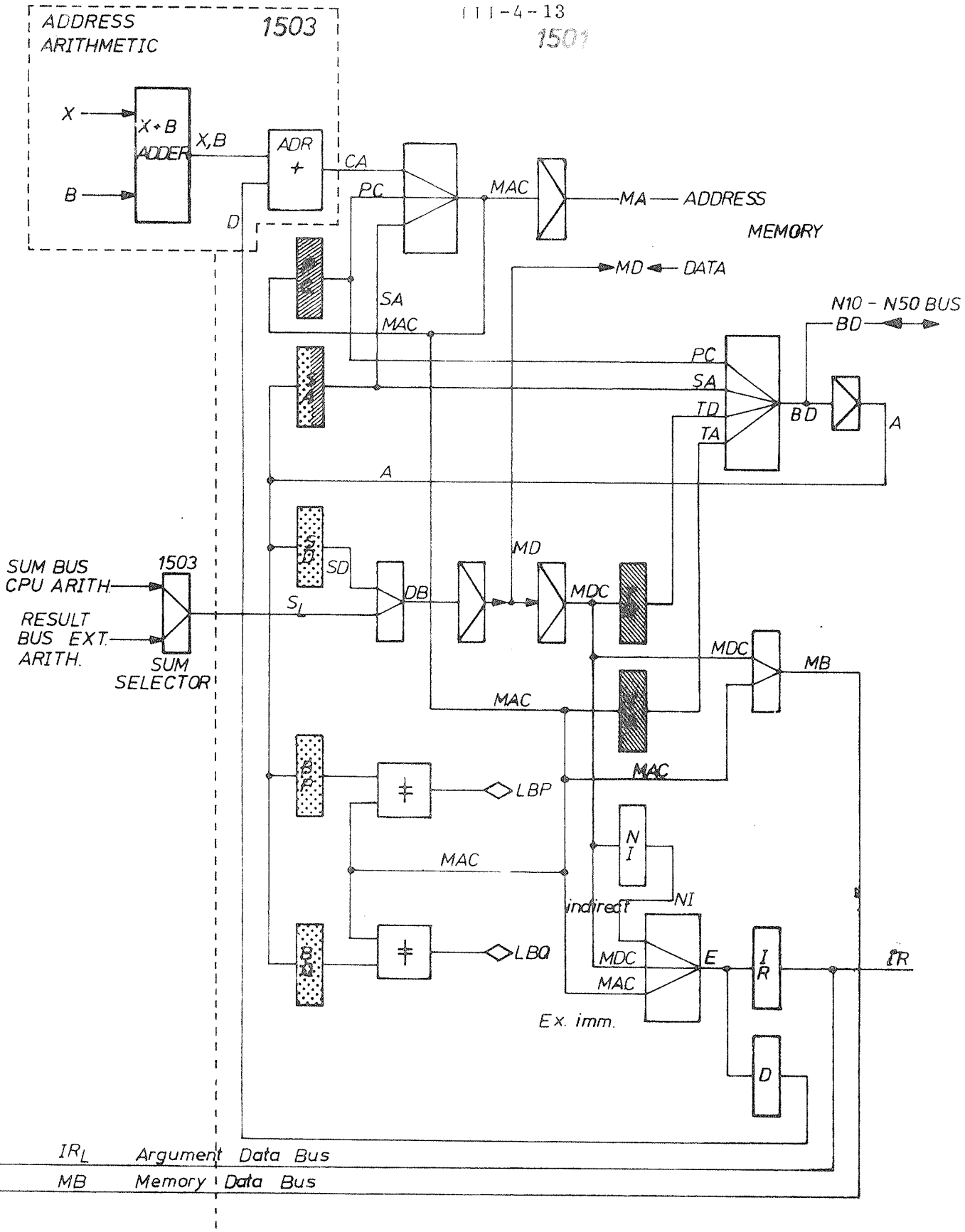
Status Register Bit No.	Meaning
0	Program STOP
1	Address Violation
2	Instruction hang-up
3	Overflow
4	Underflow
5	Memory Request
6	Memory WRITE/READ
7	Parity error
8	Memory hang-up

COMMANDS TO NORD-50 (WRITE INTERFACE CONTROL)

WIC50	33	A0=1: Enable interrupt A2=1: Start NORD-50 A4=1: Stop NORD-50

INTERFACE STATUS

RIS50	32	A0=1: Interrupt is enabled A3=1: NORD-50 is stopped



N-50 COMMUNICATION REGISTERS

Figure III.4.3

ND-05.007.01

Tegn. 8.8.77 PK/BW

III.5 NORD-50 REGISTERS

The registers in NORD-50 are built with type SN7489 64 bit READ/WRITE MEMORY integrated circuits.

Each IC consists of 16 registers or memory cells of 4 bits each.

It takes approximately 200 ns from selecting the register until the contents of the selected register is latched in the operand latch.

After enabling the register address for approcimately 200 ns, the enabling can be taken away, and a new register selected.

To reduce access time for 2-operand instructions, the register block is duplicated, with identical content.

This identical copy of the register block is called the REGISTER BLOCK B.

When writing information back to the register block, the same information is written both into the Register Block A and Register Block B.

The 16 index and base registers are duplicated two times more, still with identical contents. One set is used for reading X and one for B during address calculations.

During WRITE, the address for the X and the B register is the same as for the register blocks A and B, therefore during WRITE, identical data is written into register blocks A and B, index register RX and base register RB.

For address calculation, the address of the RX and the RB register is taken from the X and B field of the memory reference instruction.

There are 32 floating point registers with a 32 or 64 bit word length.

For 64 bit (normal) precision where word length is 64 bits, one floating register consists of a pair of general registers.

Floating registers are denoted FR in 32 bit precision, and FDR in 64 bit precision.

The NORD-50 arithmetic unit handles both 32 and 64 bits as complete parallel operations in hardware.

NB! Register 0 always contains ZERO. This implies that FR0 = 0, BR0 = 0, XR0 = 0.

In hardware, this is implemented by disabling the operand selector (giving zero output) each time register zero is read.

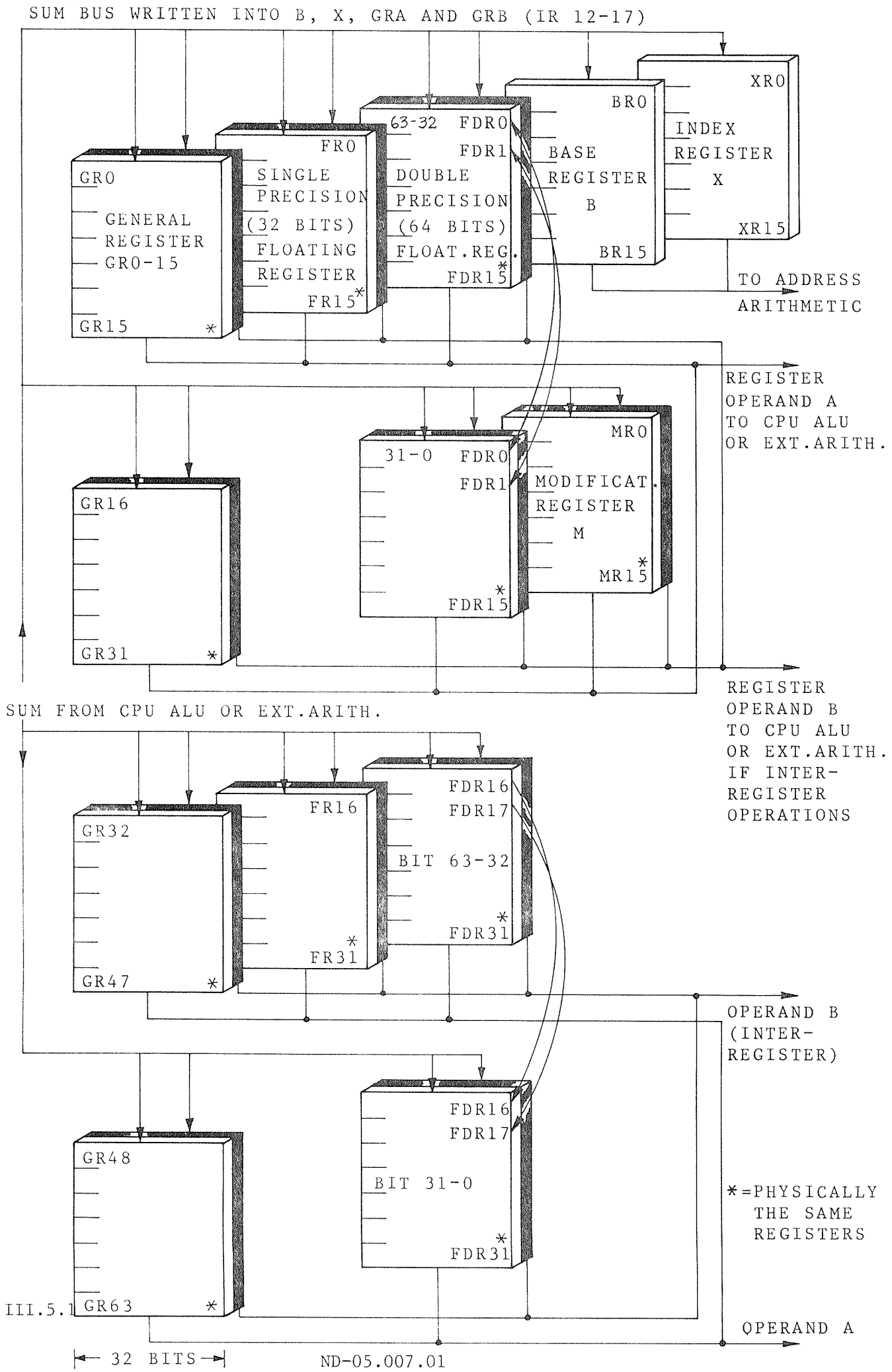


Figure III.5.1

III.6 NORD-50 FIXED POINT ARITHMETIC

The main component in the NORD-50 arithmetic is the ARITHMETIC/ LOGIC UNIT SN74181.

This integrated circuit works with two operands, the A and B-operand, 4 bits on each IC. 8 ea. 74181 IC's is then needed to handle operands of 32 bits or 1 NORD-50 word. The ALU is located on the REGISTER board 1502, one on each board. (There is a second arithmetic unit for floating point additions).

The state of the mode control input (FS4 pin 8) (refer to figure III.6.1) determines whether to do an ARITHMETIC operation (ADD, SUB) on the A and B-operand, or to do a LOGICAL operation on the operands.

The four function select inputs FS0-3 select one of 16 different functions to perform.

The instruction repertoire of the NORD-50 is built according to these operations, and the Function Code bits (FC) in the instruction are selected to give a straight forward decoding to generate the Function Select bits FS0-3.

When executing instructions that cannot be done in the ALU, for instance SHIFT, BIT OPERATIONS, FLOATING ADD/SUBTRACT, MULTIPLY and DIVIDE, this is decoded from the Function Code field in the instruction, and a EXT (External) signal is generated and the operands are sent to the EXTERNAL ARITHMETIC which executes the instructions and presents the result back to CPU. The ALU is disabled during this operation, presenting only zero on the output data lines.

The carry input is only in use during an ARITHMETIC operation and the carry input to the first ALU may be forced to a "1" and propagated through the ALU's.

To speed up the carry propagation, the carry is fed to a carry-look ahead circuit 74182 and the carry generated is then fed to the next ALU.

The output of the ALU:

1. The OUTPUT DATA or SUM (SS)
2. A ZERO signal indicating that all the output data is equal to zero
3. Carry to the next ALU

For both operand input to the ALU, there is a selector and a latch.

The latch latches the operands before the arithmetic/logic function is done.

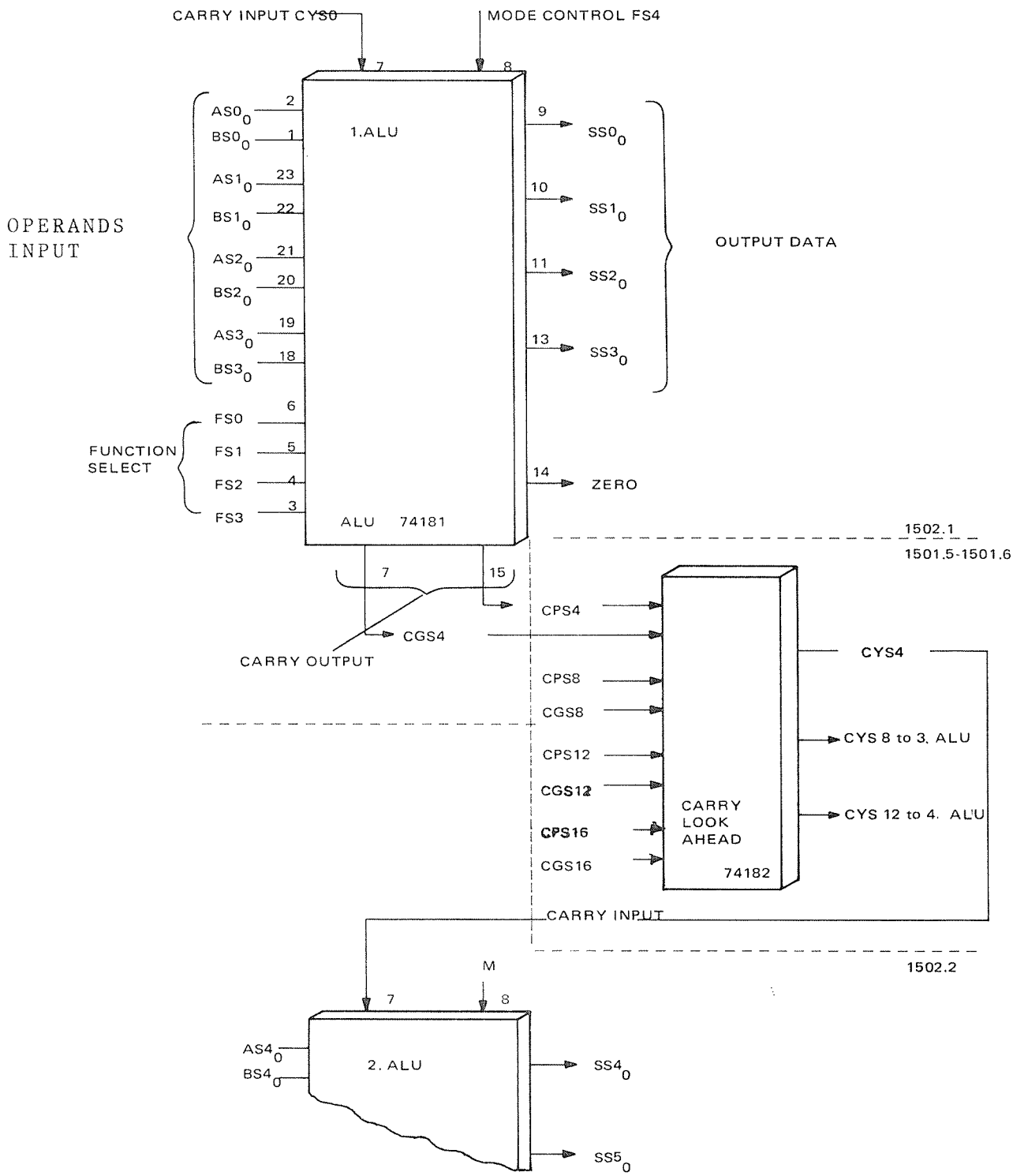
As operand A, one of the following units is selected:

1. Data from the Register Block A
2. The Overflow Register
3. The Remainder Register

and as operand B, one of the following lines is selected:

1. Data from the Register Block B
2. Data from memory
3. Operand from the instruction itself (argument data)

N50 ALU



CARRY LOOK AHEAD CIRCUITS ON THE 1502 CARD USAGE:

CARD	CARRY LOOK AHEAD FOR:
1501.1, 1501.2	Address arithmetic X, B + D adder
1501.3, 1501.4	Address arithmetic X + B adder
1501.7	
1501.5, 1501.6	CPU ALU

Figure III.6.1.

III.7 MEMORY REFERENCE INSTRUCTIONS

The memory reference instructions have in common that the execution of an instruction involves calculation of a memory address, either to read out the operand or to write the operand or result back. The memory addressing may be direct or indirect.

Memory reference instructions contain a single bit direct addressing indicator I = 31, two 4 bit register addresses (X and B), a 5 bit function code, a 6 bit register address for the other operand, and a 12 bit displacement.

If I equals 0, the effective memory operand address or calculated address (CA) will be the sum of the 32 bit contents of the X and B registers and the 12 bit displacement.

The 30 memory reference instructions are two-address instructions affecting one directly or indirectly addressed memory location and one register.

The result may either be stored in the same register or the same location.

During a jump or conditional jump instruction, the next instruction is fetched from the calculated address.

The operand read from memory (refer to Figure III.7.1) goes via the MB lines through the B operand selector and is latched in the B operand latch.

The register operand is selected through the A operand selector and latched in the A operand latch.

One exception from this is the MODIFY instructions JPM, JNM, JZM and JFM where the MODIFY REGISTERS in register block B are selected and latched as B operand.

The result is written either to the selected register or calculated address via the SUM bus.

In the following, all the memory reference instructions are lsited with information of the hardware execution of the instruction in the CPU ALU.

In the hardware diagrams the memory reference instructions are devided into groups given by IR bits 20,21, and 22. (Group-numbers are listed).

MEMORY REFERENCE INSTRUCTION ALU OPERATION:

Instruction		GROUP	Operand A	Operand B	Function Select FS 0	1	2	3	Mode Contr. FS4	Forced Carry C	Means	Dest.	Comments
													DATA TRANSFER INSTRUCTIONS:
LDR	Load Register	IG4		MB	0	1	0	1	1	0	BDIR	R	B-operand (MB) direct through ALU
LDD	Load Double Register or			MB	0	1	0	1	1	0	BDIR	R	First cycle transfer (CA) to R
	Floating Register (FR)	IG6		MB	0	1	0	1	1	0	BDIR	R+16	Second cycle transfer (CA+1) to R+16
STR	Store Register	IG5	R		1	1	1	1	1	0	ADIR	CA	Store register in calculated address (CA)
STD	Store Double Register or		R		1	1	1	1	1	0	ADIR	CA	First cycle transfer R to CA
	Floating Register	IG5	R+16		1	1	1	1	1	0	ADIR	CA+1	Second cycle transfer R+16 to CA+1
XMR	Exchange (CA) and (R)	IG5											First cycle: (R) → A latch, (CA) → B latch
			R	MB	1	1	1	1	1	0	ADIR	CA	Second cycle: Write B latch → R
			R	MB	0	1	0	1	1	0	BDIR	R	Third cycle: Write A latch → CA
													ARITHMETIC INSTRUCTIONS:
ADD	Add (CA) to R	IG4	R	MB	1	0	0	1	0	0	A+B	R	
SUB	Subtract (CA) from R	IG4	R	MB	0	1	1	0	0	1	A-B-1	R	With forced carry, gives A-B as function
ADM	Add to memory	IG5											First cycle: (R) → A latch, (CA) → B latch
			R	MB	1	0	0	1	0	0	A+B	CA	Second cycle: Execute ADD
			R	MB	1	0	0	1	0	0	A+B	CA	Third cycle: Write result to memory
MIN	Memory increment	IG0											First cycle: (R) → A latch, (CA) → B latch
			R	MB	1	0	0	1	0	1	A+B	CA	Second cycle: With forced carry, gives A+B+1
			R	MB	1	0	0	1	0	1	A+B	CA	Third cycle: Write result to memory
													Enter cycle 4 if skip
													LOGIC INSTRUCTIONS:
AND	And (CA) and R	IG4	R	MB	0	1	1	1	1	0	A·B	R	

MB = Operand from memory

Instruction group

MEMORY REFERENCE INSTRUCTIONS ALU OPERATIONS:

Instruction		GROUP	Operand A	Operand B	Function Select FS 0	1	2	3	Contr. FS4	Forced Carry	Means	Dest.	Comments		
													CONDITIONAL SEQUENCE AND SEQUENCE INSTRUCTIONS:		
CRG	Skip if (R) ⩾ (CA)	IG1	R	MB	0	1	1	0	0	1	A-B	None	The sign bit (S31) output of ALU checked if SKIP eff.		If SKIP eff., enter cycle 4
CRL	Skip if (R) < (CA)		R	MB	0	1	1	0	0	1	A-B	No write pulse gen.			
CRE	Skip if (R) = (CA)		R	MB	0	1	1	0	0	1	A-B		The ZERO output of ALU checked if SKIP eff.		
CRD	Skip if (R) ≠ (CA)		R	MB	0	1	1	0	0	1	A-B				
JRP	Jump if (R) ⩾ 0	IG2	R		1	1	1	1	1	0	ADIR	None	The SIGN bit (S31) checked if JUMP eff.		If JUMP eff., enter cycle 4
JRN	Jump if (R) < 0		R		1	1	1	1	1	0	ADIR	None			
JRZ	Jump if (R) = 0		R		1	1	1	1	1	0	ADIR	None	The ZERO checked if JUMP eff.		
JRF	Jump if (R) ≠ 0		R		1	1	1	1	1	0	ADIR	None			
JPM	Add Modify Register to R and JUMP if (R) ⩾ 0	IG3	R	R+16	1	0	0	1	0	0	A+B	R	Add R and MR in ALU	Check S31 if JUMP eff.	If JUMP eff., enter cycle 4
JNM	Add Modify Register to R and JUMP if (R) < 0		R	R+16	1	0	0	1	0	0	A+B	R			
JZM	Add Modify Register to R and JUMP if (R) = 0		R	R+16	1	0	0	1	0	0	A+B	R		Check ZERO if JUMP eff.	
JFM	Add Modify Register to R and JUMP if (R) ≠ 0		R	R+16	1	0	0	1	0	0	A+B	R			
RTJ	Return Jump PC+1→ R	IG0	Zero	PC+1*	1	0	0	1	0	0	A+B	R	Program Counter (PC)+1 written back in specified register		

* Enabled on MB-bus

MEMORY REFERENCE INSTRUCTION N O T USING THE CPU ALU:

Instruction		GROUP	Function Select FS 0	1	2	3	Mode Contr. FS4	Forced Carry	Means	Comments
EXC	Remote Execute	IG0								
	R = 0		0	0	1	1	1	0	ZERO output	Execute content of CA as Instruction
	R = 1		0	0	1	1	1	0	ZERO output	Execute the CA as Instruction
MPY	Integer multiply	IG6	0	0	1	1	1	0	ZERO output	Operand latched in CPU and
DIV	Integer divide		0	0	1	1	1	0	ZERO output	instruction executed in
FAD	Floating add	↑	0	0	1	1	1	0	ZERO output	EXTERNAL arithmetic
FADD	Floating add double precision		0	0	1	1	1	0	ZERO output	
FSB	Floating subtract		0	0	1	1	1	0	ZERO output	
FSBD	Floating subtrac double	IG7	0	0	1	1	1	0	ZERO output	
FMU	Floating multiply		0	0	1	1	1	0	ZERO output	
FMU	Floating multiply double		0	0	1	1	1	0	ZERO output	
FDV	Floating divide		0	0	1	1	1	0	ZERO output	
FDVD	Floating divide double	↓	0	0	1	1	1	0	ZERO output	

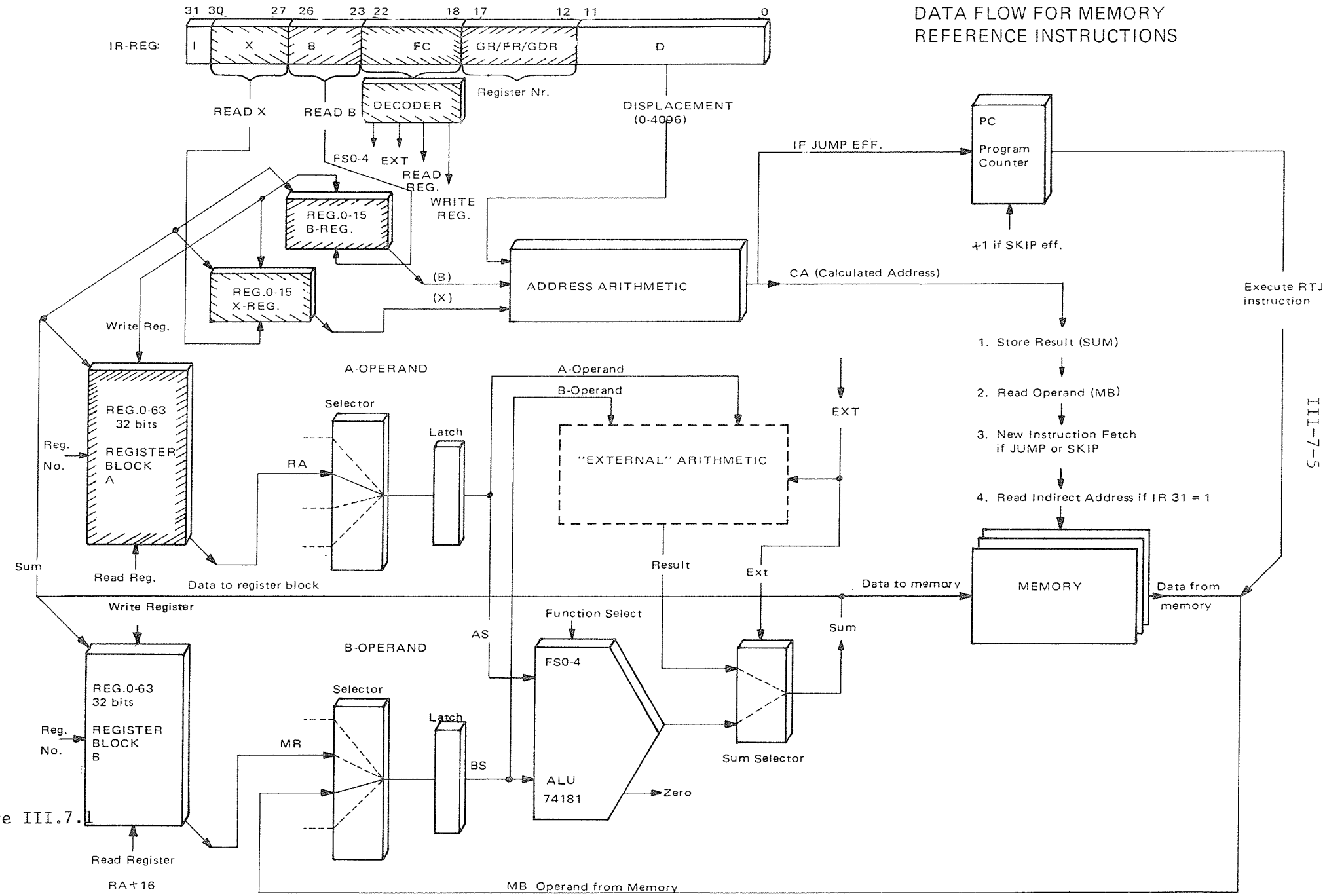


Figure III.7.1

III.7.1 INDIRECT ADDRESSING

Indirect Addressing takes place when bit 31 in the Instruction Register is equal to 1.

This means that the address now in the instruction word does not point to the data or operand, but rather points to a "pointer to" the data. Thus, the instruction word address is said to be ONE LEVEL away from the data.

It is also possible for the instruction word address to be more than one level away from the data.

If a word of memory is a pointer to a pointer, rather than a pointer to the data, then a bit of the memory word is set to indicate this (bit 31).

In addition to the IR-register, there is also a register with identical contents except during indirect addressing, the displacement register D. This register is the input to the address arithmetic in use during a memory reference instruction with the bits used as follows:

Bits 31-20: Not added in the X, B and D adder

Bits 19-12: Added only during second indirect address calculation

Bits 11-0: Always added in a memory reference instruction address calculation

If bit 31 - 0 in the IR-register, only bits 11-0 in the D-register are added in the X, B and D adder during the first indirect address calculation.

If bit 31 in the now calculated address is a 1, a new level of indirect addressing is inserted.

Up to 16 levels of indirect addressing is possible. More levels will cause INSTRUCTION HANG-UP interrupt.

Bit 31 equals one in the calculated address means that the contents of this cell is used to calculate the next indirect address.

Bit 31 equals zero indicates the last level of indirect addressing and the contents of this cell is the operand address.

In both cases, the 20 lower bits are clocked into the D-register.

Bits 31-23 are clocked into the IR-register while bits 22-12 in IR are kept unchanged. This is done to avoid destroying the function code and the register involved in the instruction.

Indirect addressing inserts an extra indirect cycle (C1) and each level of indirect addressing adds one "read instruction time" to the execution time.

INDIRECT ADDRESSING

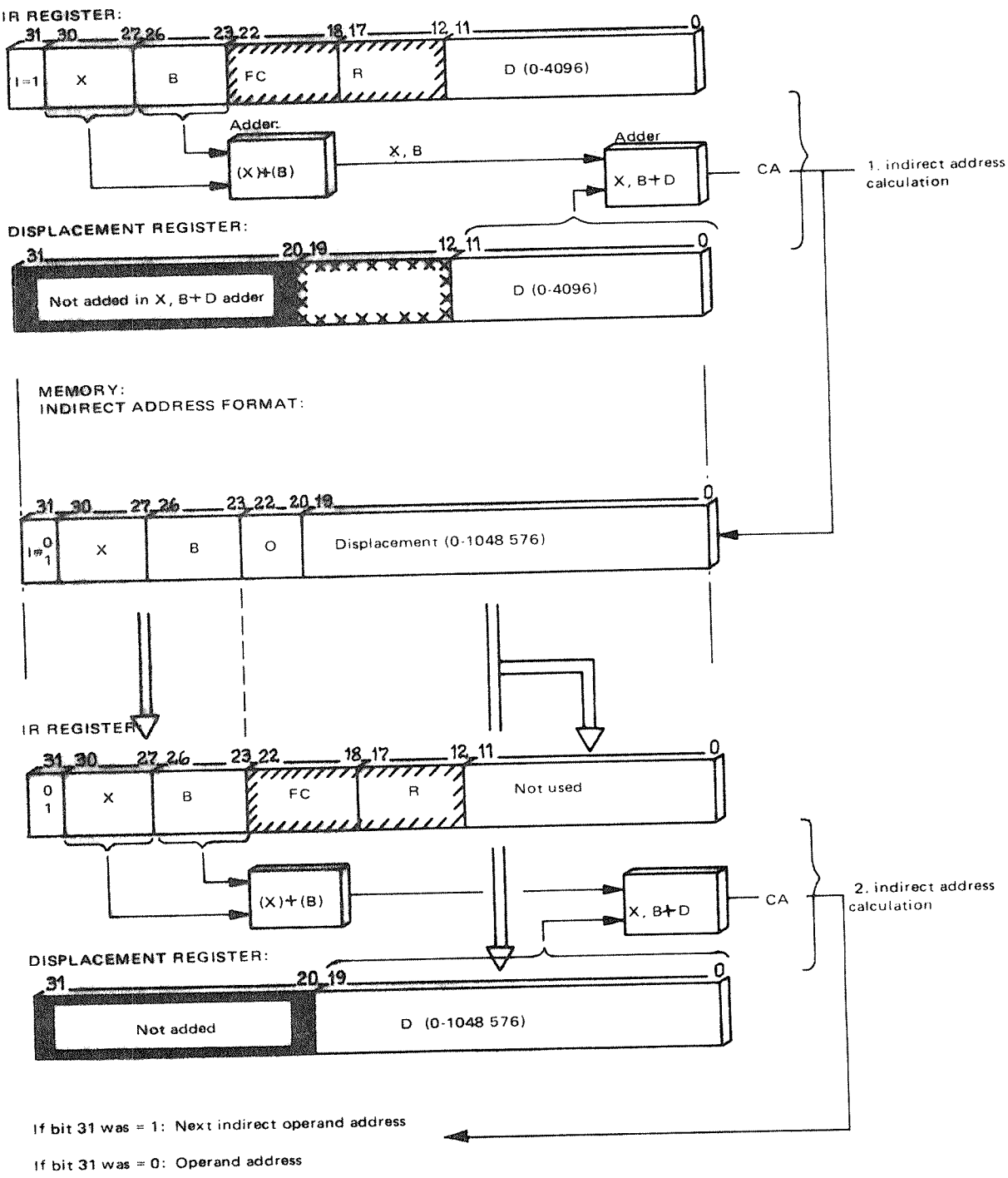


Figure III.7.2.

III.7.2 INTER-REGISTER INSTRUCTIONS

The inter-register instructions have in common that both operands are taken from the registers, called source register A and source register B and the result is written back into a third register, the destination register.

The 23 inter-register instructions are three-address instructions containing three 6-bit register addresses: two source registers holding operands A and B respectively, and a destination register.

As indicated in figure III.7.3, the register block A is selected and latched in the A operand latch, while register block B is selected and latched in the B operand latch.

As A operand the Overflow Register OR and Remainder Register RR may also be selected.

In the following, all the inter-register instructions are listed with information on the hardware execution of the instruction in the CPU ALU.

INTER-REGISTER INSTRUCTIONS ALU OPERATION:

Instruction	Function Select FS 0	1	2	3	Mode Contr. FS4	Forced Carry CYSO	Means	Comments
								DATA TRANSFER INSTRUCTIONS:
RIN RR or OR register → Register R	1	1	1	1	0	1	ADIR	A-operand (RR or OR) direct through ALU
ROUT R → RR or OR	1	1	1	1	0	1	ADIR	A-operand R direct through ALU
								ARITHMETIC INSTRUCTIONS:
RAD Register add	1	0	0	1	0	0/1	A+B	The forced carry depends on IR25 , See Reference Manual 3-9
RSB Register subtract	0	1	1	0	0	0/1	A-B-1	With forced carry, gives A-B as function
								LOGIC INSTRUCTIONS:
RND Register AND	0	1	1	1	1	0	$A \cdot B$	
RNDA Register AND, complement SRA	0	0	0	1	1	0	$\overline{A} \cdot B$	
RNDB Register AND, complement SRB	1	0	1	1	1	0	$A \cdot \overline{B}$	
RXO Register exclusive OR	1	0	0	1	1	0	$A \not\vee B$	
RXOA Register exclusive OR, compl. SRA	0	1	1	0	1	0	$\overline{A} \not\vee B$	
RXOB Register exclusive OR, compl. SRB	0	1	1	0	1	0	$A \not\vee \overline{B}$	
ROR Register OR	1	1	0	1	1	0	A + B	
RORA Register OR, complement SRA	0	1	0	0	1	0	$\overline{A}$ + B	
RORB Register OR, complement SRB	1	1	1	0	1	0	A + $\overline{B}$	
SZR Set destination register zeros	0	0	1	1	1	0	Zero	The ALU outputs ZERO, written into destination register

INTER REGISTER INSTRUCTIONS ALU OPERATION:

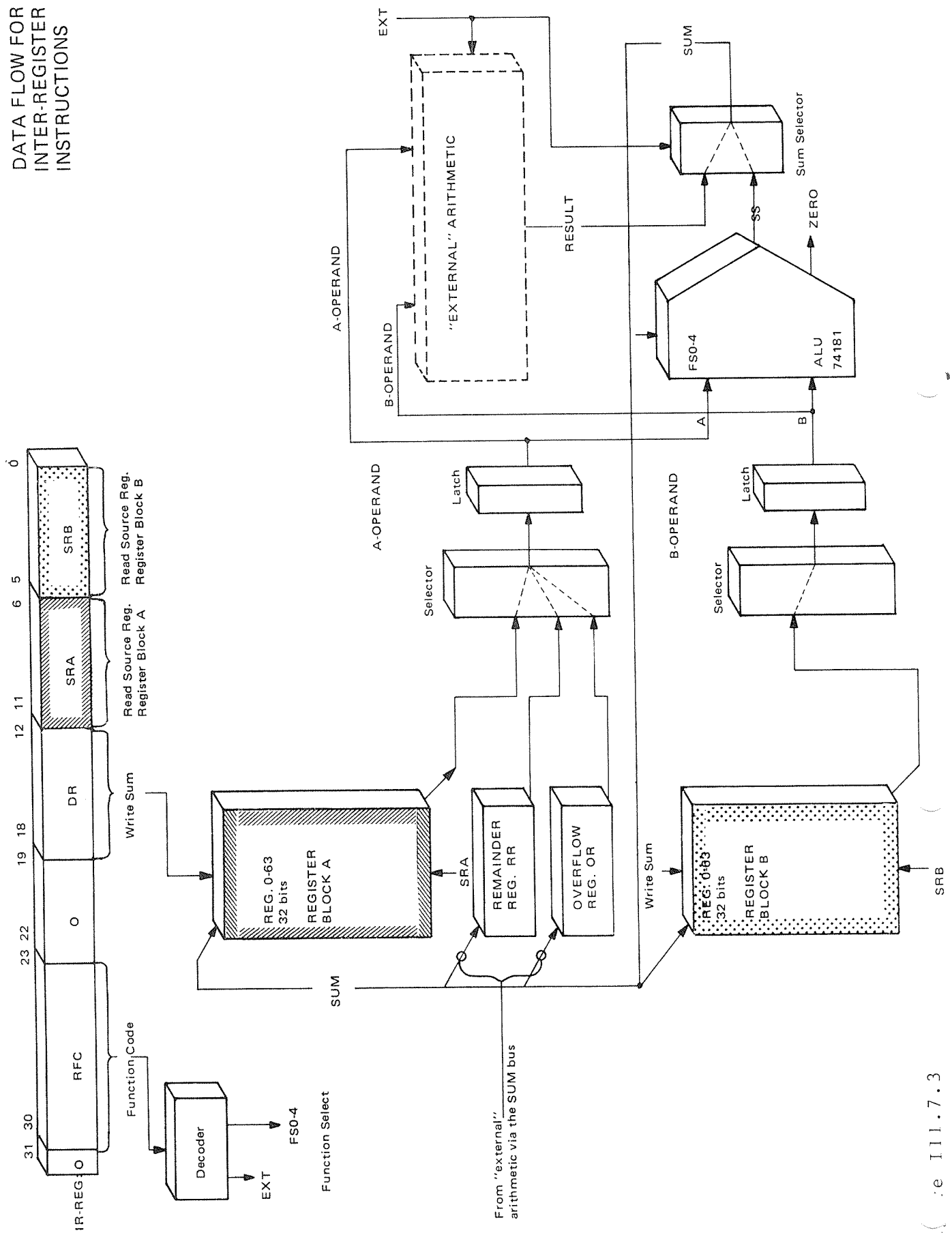
Instruction			Function Select FS 0	1	2	3	Mode Contr. FS4	Forced Carry CYSO	Means	Comments	
										CONDITIONAL SEQUENCE INSTRUCTIONS:	
ASG	Add registers, skip if result	≥0	1	0	0	1	0	0	A+B	The SIGN BIT (S31) output of ALU checked if skip effective	If SKIP effective enter cycle 4
ASL	Add registers, skip if result	<0	1	0	0	1	0	0	A+B		
ASE	Add registers, skip if result	=0	1	0	0	1	0	0	A+B	The ZERO output of ALU checked if skip effective	
ASU	Add registers, skip if result	≠0	1	0	0	1	0	0	A+B		
SGR	Subtract registers, skip if result	≥0	0	1	1	0	0	1	A-B	The SIGN BIT (S31) output of ALU checked if skip effective	
SLE	Subtract registers, skip if result	<0	0	1	1	0	0	1	A-B		
SEQ	Subtract registers, skip if result	=0	0	1	1	0	0	1	A-B	The ZERO output of ALU checked if skip effective	
SUE	Subtract register, skip if result	≠0	0	1	1	0	0	1	A-B		

INTER REGISTER INSTRUCTIONS N O T USING THE CPU ALU:

Instruction		Function Select FS 0 1 2 3	Mode Contr. FS4	Means	Comments
					DATA MANIPULATION INSTRUCTIONS:
SLR, SLRD	Register, Double Reg. LEFT shift	0 0 1 1	1	Zero output of CPU ALU	Operand read out from the Register block A to External arithmetic. Instruction executed and result written back to Destination Register (Register Block A and B)
SRR, SRRD	Register,Double Reg. RIGHT shift	0 0 1 1	1		
SLA, SLAD	LEFT ARITHMETIC shift	0 0 1 1	1		
SRA, SRAD	RIGHT ARITHMETIC shift	0 0 1 1	1		
SLL, SLLD	LEFT LOGICAL shift	0 0 1 1	1		
SRL, SRLD	RIGHT LOGICAL shift	0 0 1 1	1		
BST	Bit set	0 0 1 1	1		
BCM	Bit complement	0 0 1 1	1		
BCL	Bit clear	0 0 1 1	1		
FIX, FIXD	Floating → Integer	0 0 1 1	1		
FIR, FIRD	Floating → Rounded Integer	0 0 1 1	1		
FLO, FLOD	Integer → Floating	0 0 1 1	1		

INTER REGISTER INSTRUCTIONS N O T USING THE CPU ALU:

Instruction			Function Select FS 0	1	2	3	Mode Contr. FS4	Means	Comments
									ARITHMETIC INSTRUCTIONS:
RMU		Register multiply	0	0	1	1	1	Zero output of CPU ALU	
RDV		Register divide	0	0	1	1	1		Instructions executed in external arithmetic. Result written back in CPU Destination-Register
RAF,	RAFD	Floating register (FR) add,double precision register FRD add	0	0	1	1	1		
RSF,	RSFD	FR, FRD subtract	0	0	1	1	1		
RMD,	RMFD	FR, FRD multiply	0	0	1	1	1		
RDF,	RDFD	FR, FRD divide	0	0	1	1	1		
									CONDITIONAL SEQUENCE INSTRUCTIONS:
SGF,	SGD	FR, FRD Register subtract, skip if result ⩾0	0	0	1	1	1		Floating add/subtract done in external arithmetic. Result ⩾0, <0 checked against sign bit. Result =0, ≠0 checked against floating zero.
SLF,	SLD	skip if result <0	0	0	1	1	1		
SEF,	SED	skip if result =0	0	0	1	1	1		
SUF,	SUF	skip if result ≠0	0	0	1	1	1		
ASGF,	ASFD	FR FRD Register add, skip if result ⩾0	0	0	1	1	1		If SKIP effective; SKIP signal sent to CPU and CPU enter cycle 4
ASLF,	ALFD	skip if result <0	0	0	1	1	1		
ASEF,	AEFD	skip if result =0	0	0	1	1	1		
ASUF,	AUFD	skip if result ≠0	0	0	1	1	1		
BSZ		Bit skip on ZERO	0	0	1	1	1		Bit checked in external arithmetic. If SKIP effective; SKIP signal sent to CPU and CPU enter cycle 4
BSO		Bit skip on ONE	0	0	1	1	1		


DATA FLOW FOR
INTER-REGISTER
INSTRUCTIONS
EXT
SUM
"EXTERNAL" ARITHMETIC
A-OPERAND
B-OPERAND
RESULT
Sum Selector
SS
ZERO
FS0-4
ALU
74181
A
B
Latch
Selector
A-OPERAND
B-OPERAND
Read Source Reg.
Register Block B
Read Source Reg.
Register Block A
SRB
SRA
DR
O
RFC
IR-REG
Write Sum
REG. 0-63
32 bits
REGISTER
BLOCK A
SRA
REMAINDER
REG. RR
OVERFLOW
REG. OR
Write Sum
REG. 0-63
32 bits
REGISTER
BLOCK B
SRB
SUM
Function Code
Decoder
FS0-4
EXT
Function Select
From "external"
arithmetic via the SUM bus

Fi(.e III.7.3

III.7.3 ARGUMENT INSTRUCTIONS

The Argument Instructions have in common that one of the operands (operand B) is taken from the instruction itself (Bits 0-15) and the other operand is taken from the Register Block A, specified by bits 23-28 in the Instruction Register.

The result or SUM is written back to the same register given by bits 23-28 in the IR-register in both the Register Block A and Register Block B (X and B if register address 0-15).

The 16 most significant bits in the B-operand are always set to zero.

In the following, all the argument instructions are listed with information on the hardware execution of the instruction in the CPU ALU.

ARGUMENT INSTRUCTION ALU OPERATION:

Instruction		Function Select FS 0	1	2	3	Mode Contr. FS4	Forced Carry	Means	Comments
									ARITHMETIC INSTRUCTIONS:
ADDA	Add Argument to Register	1	0	0	1	0	0	A+B	
ADCA	Add Complement of Argument to Register	0	1	1	0	0	1	A-B-1	With forced carry: Argument Subtracted from Reg
									LOGIC INSTRUCTIONS:
ANDA	AND Arg. and Register	0	1	1	1	1	0	A·B	
ORA	OR Arg. and Register	1	1	0	1	1	0	A+B	
XORA	Exclusive OR Arg and Register	1	0	0	1	1	0	A∀B	
									DATA MANIPULATION INSTRUCTIONS:
SETA	Set Arg. to Register	1	0	0	1	0	0	A+B	The A-operand is set to ZERO (Reg. 0 selected)
SECA	Set Complement of Arg to Register	0	1	1	0	0	1	A-B-1	The A-operand is set to ZERO (Reg. 0 selected) With forced carry: -B (-ARG) to selected Reg.
									CONDITIONAL SEQUENCE INSTRUCTIONS:
DDP	SKIP if Reg ⩾Arg	0	1	1	0	0	1	A-B-1	The SIGN Bit (S31) output of ALU checked if SKIP eff. If SKIP enter cycle 4
DDN	SKIP if Reg <Arg	0	1	1	0	0	1	A-B-1	
DDZ	SKIP if Reg =Arg	0	1	1	0	0	1	A-B-1	The ZERO output of ALU checked if SKIP eff. If SKIP enter cycle 4
DDF	SKIP if Reg ≠Arg	0	1	1	0	0	1	A-B-1	
DSP	SKIP if Reg ⩾-Arg	1	0	0	1	0	0	A+B	The SIGN Bit (S31) output of ALU checked if SKIP eff If SKIP eff enter cycle 4
DSN	SKIP if Reg <-Arg	1	0	0	1	0	0	A+B	
DSZ	SKIP if Reg =-Arg	1	0	0	1	0	0	A+B	The ZERO output of ALU checked if SKIP eff If SKIP eff enter cycle 4
DSF	SKIP if Reg ≠-Arg	1	0	0	1	0	0	A+B	

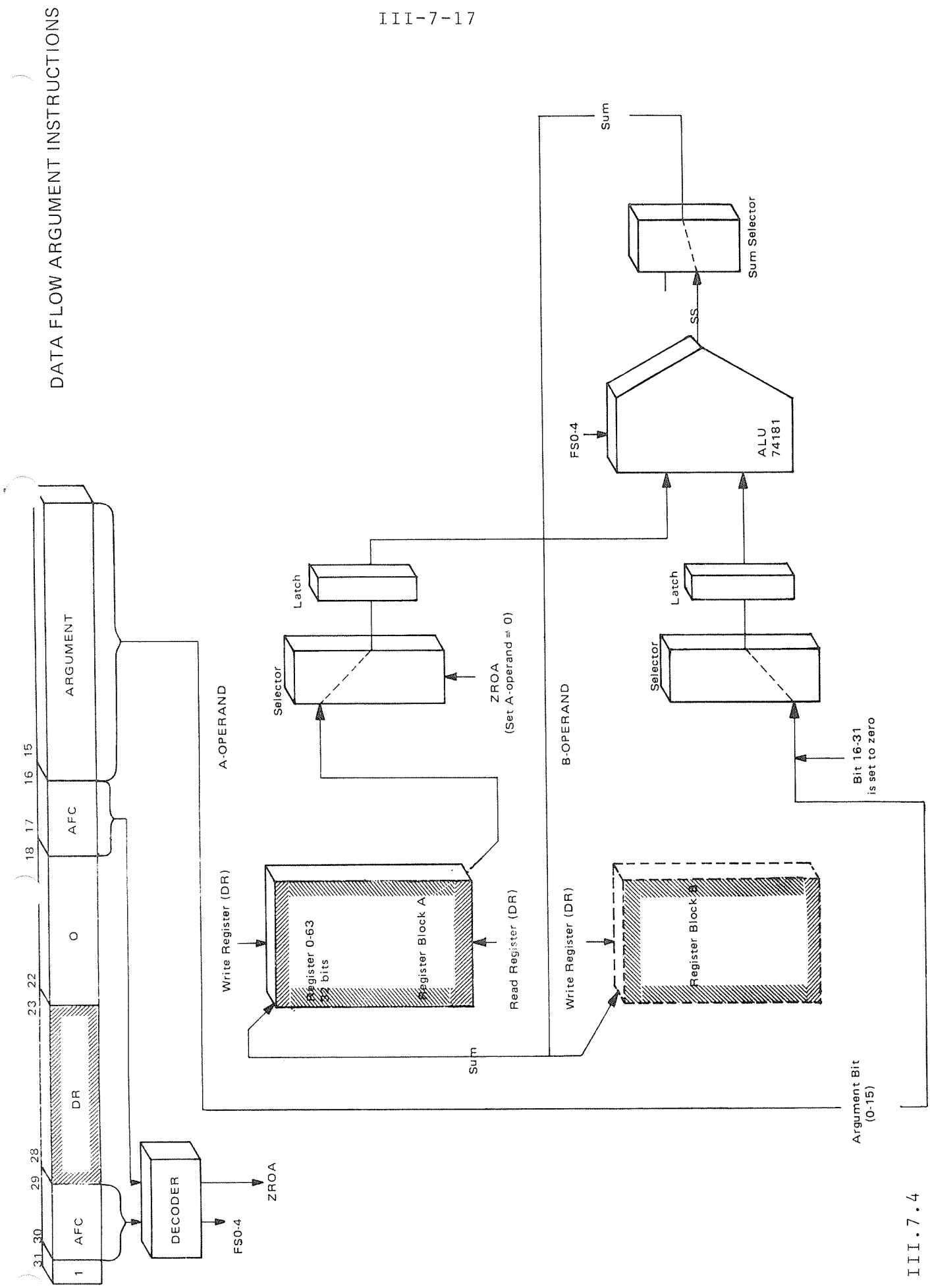


Figure III.7.4

III.8 INSTRUCTION EXECUTE SEQUENCE

In parallel with executing the instruction in the Instruction Register IR, the next instruction in hte program (PC+1) is requested and written into the NEXT INSTRUCTION register NI.

By implementing this instruction pre-fetch feature, the waiting time for memory reply is utilized to execute the instruction in the instruction register. When the instruction execution is finished, the next instruction register is transferred to the IR-register. This is the way inter-register and argument instructions are executed.

When executing a memory reference instruction, the CPU will be busy calculating the memory operand address (CA) for the instruction in IR, while the next instruction is fetched and clocked into the NI register. When this memory sequence is finished, the memory read or write reference in calculated address can take place.

SPECIAL CASES

The normal execute sequence is to transfer the NI-register to the IR-register and start execution. But the instruction in the NI-register may not be the one to be executed next. This is the situation in the cases mentioned below:

1. At START-UP time
2. Executing a sequence instruction
   Transferring program execution from the current location to some other location in memory
3. Executing a remote execute (EXC) instruction

1. When NORD-50 stops, the NI-register will be equal to the instruction following the last executed instruction.

   At START-UP time, a new instruction must be fetched, pointed to by the Start Address (SA). This instruction is clocked into the NI-register and further to the IR-register.

   A special cycle (DC4) takes care of reading the contents of the program counter = SA to the NI-register and next cycle DC0 will transfer NI to IR and the execution can start.

2. During a jump, conditional jump or skip, where the jump or skip condition is fulfilled, the NI-instruction is not the one to be executed next.

   IR = JUMP or CONDITIONAL JUMP

   The program counter is parallel loaded with the calculated address from the address arithmetic. The same DC4 cycle is entered and the PC-address is fetched and clocked into the NI-register.

   IR = SKIP

   The program counter is incremented by one pointing two instructions ahead of the instruction in IR. The same DC4 is entered fetching the new instruction.

3. The input to the IR-register consists of a 3-line to 1-line selector. The first line is the NI-register and the two remaining lines are in use while executing a remote execute instruction.

   IR = Remote Execute (EXR)

   If the register field equals zero, this means: execute contents of calculated address as instruction.

   The instruction in the calculated address is fetched and placed directly in the IR-register via the memory data line MDC.

   If the register field equals one, this means: execute the calculated address as instruction. The calculated address is fed directly to the IR-register via the memory address line MAC.

N - 50 MEMORY SEQUENCE

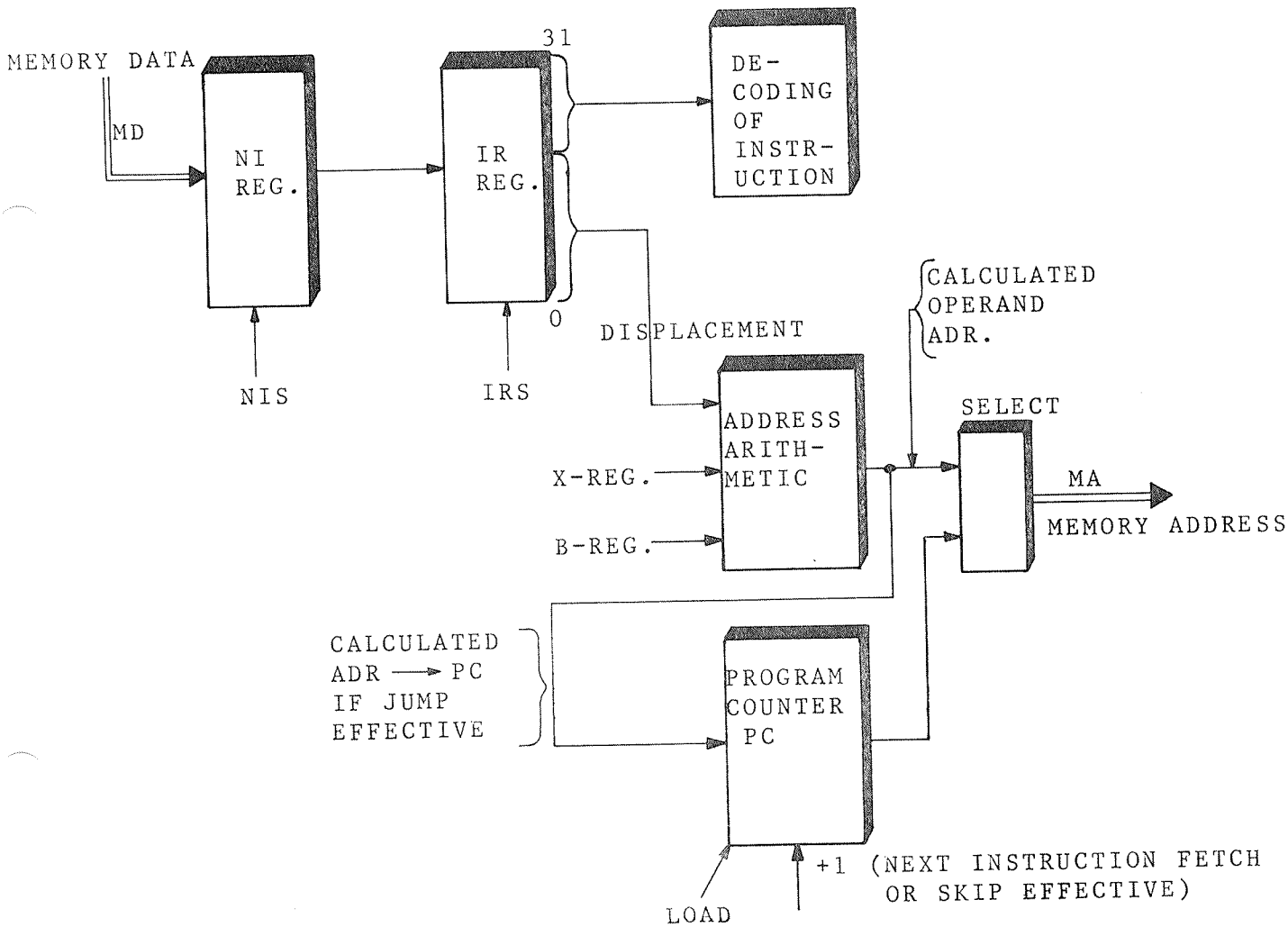


Figure III.8.1

N - 50 MEMORY SEQUENCE

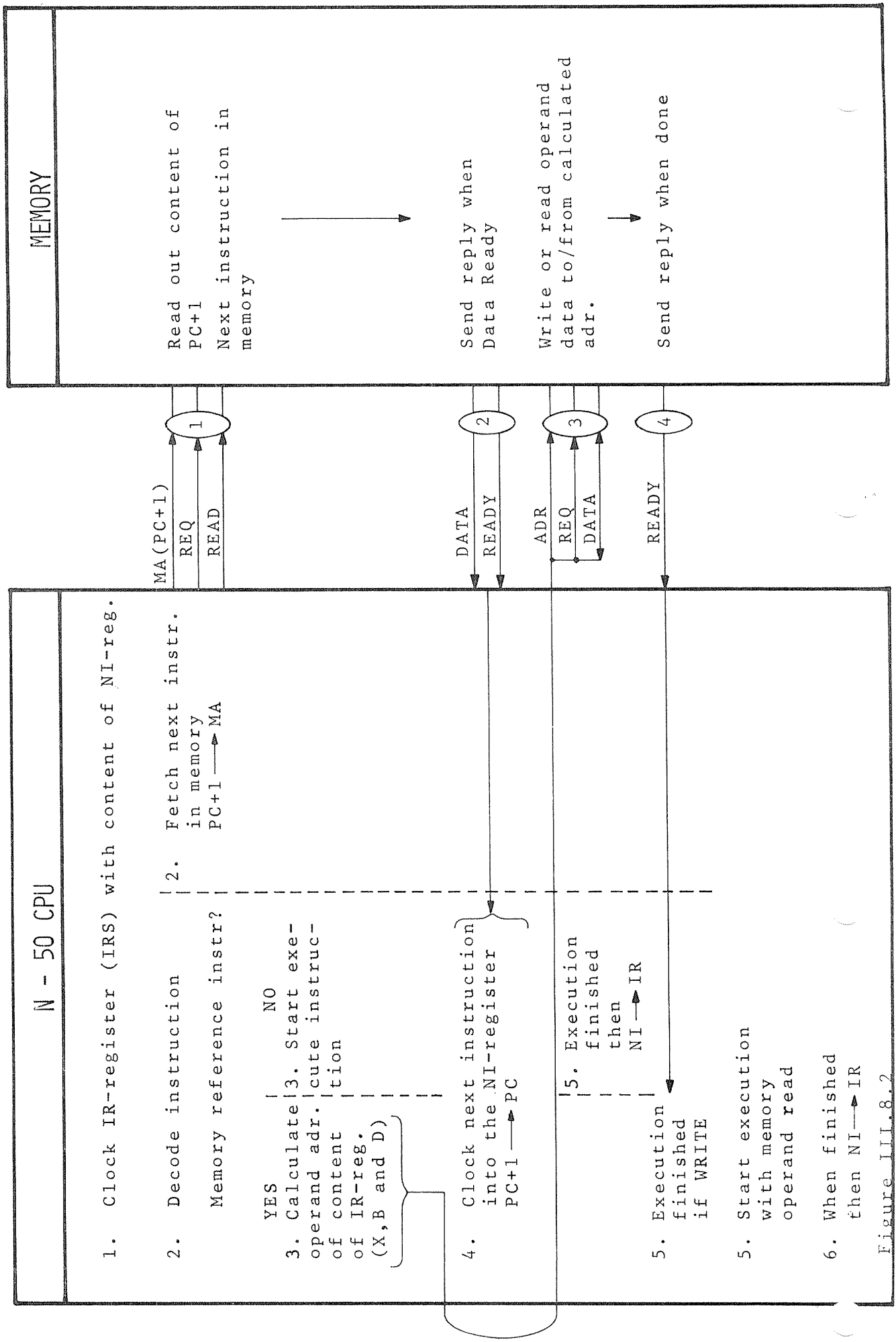


Figure III.8.2

III.9 NORD-50 MAIN STATES

The NORD-50 main states are determined by three flip-flops located on the CYCLE COUNTER card 1506; DCC0, DCC1 and DCC2, also called the cycle counter flip-flops.

The state of the flip-flops is decoded on the 1519 card and gives the implemented cycles listed below:

DCC0	DCC1	DCC2	CYCLE
0	0	0	DC0
1	0	0	DC1
0	1	0	DC2
1	1	0	DC3
0	0	1	DC4
1	1	1	DC7

The basic machine cycles in NORD-50 are:

DC0: First cycle of instruction execution.
Memory reference instruction:
Execution/operand fetch cycle.

Inter-register or argument instructions:
execute cycle

DC1: Indirect address cycle

DC2: Instruction execute if memory reference instruction

DC3: Instruction execute for instructions with two data references in memory

DC4: Instruction readout to next instruction register (NI) in start sequence, in skip instruction effective or jump instruction effective.

DC7: Stop Cycle

Data may be transferred to/from NORD-50 communication registers via the NORD-10 I/O system.

Each time the clock pulse DCCS arrive the output of the NEXT CYCLE DECISION LOGIC will be clocked into the DCC0-2 flip-flops. This implies that clocking the same information once more will extend the cycle for one clock period.

During multi-level indirect, for instance, the lines containing 001 (the input to DCC2 = 0, input to DCC1 = 0 and input to DCC0 = 1), may be clocked up to 15 times.

The cycle will be terminated and a new cycle entered (or the same cycle extended) at WP time, equal to the time new information in written into the CPU register block either from CPU ALU or from external arithmetic.

Cycle DC7, the STOP cycle may be entered in two ways:

A) Presenting a ground level on the MASTER CLEAR (MCL) line to the clear input of the DCC0-2 flip-flops.

   The MCP signal generated on the 1519 card is a programmed MASTER CLEAR pulse which occurs when bit 9 in the modus register is set. Used to put NORD-50 into a well-defined state.

B) Forcing the NC7 signal (Next Cycle DC7) to a ground level, the NAND gates output will be all ones and at the next arriving DCCS pulse, DC7 will be entered.

The ND7 signal will force the cycle flip-flops to DC7 at the first arriving DCCS clock pulse in the following cases:

1. Protect Violation
   The requested address was protected by the BP and BQ address limits.

   Hardware:
   ABPBQ signal generated on the 1504 card. Status bit 1 set to a one.

2. External stop from NORD-10
   Executing an IOX NOR50 instruction and A = 10 in NORD-10, the NORD-50 will enter cycle DC7.

3. STOP instruction executed in NORD-50

4. Parity error of modus bit 2 is set; detected on the 1504 card.

5. Overflow detected in external arithmetic and modus bit 0 set (OFL = SB3)
6. Underflow detected in external arithmetic and modus bit 1 set (UFL = SB4)

} Sensed at RYX time on 1504

7. Instruction hang-up
   More than 15 levels of indirect addressing or execute of EXC instruction (counted on the 1510 card).

Input to the NEXT CYCLE DECISION LOGIC is information about current cycle DC0-DC7 and IR function code bits informing what cycle to go through executing the instruction.

An active JEFF signal (jump or skip effective) will force the NEXT CYCLE DECISION LOGIC to present 100 on the input lines to the DCC0-2 flip-flops, entering cycle DC4 for instruction readout to NI.

The JUMP OR SKIP CONDITION FULFILLED logic has this input in addition to the IR function code bits:

ZERO (CPU ALU output equal to zero)
S31 (CPU ALU sign-bit output)
FRSBIT, SELBIT } From external arithmetic (C-rack)

These signals are checked against the IR bits if the jump or skip conditions are fulfilled.

NORD-50 MAIN STATE LOGIC

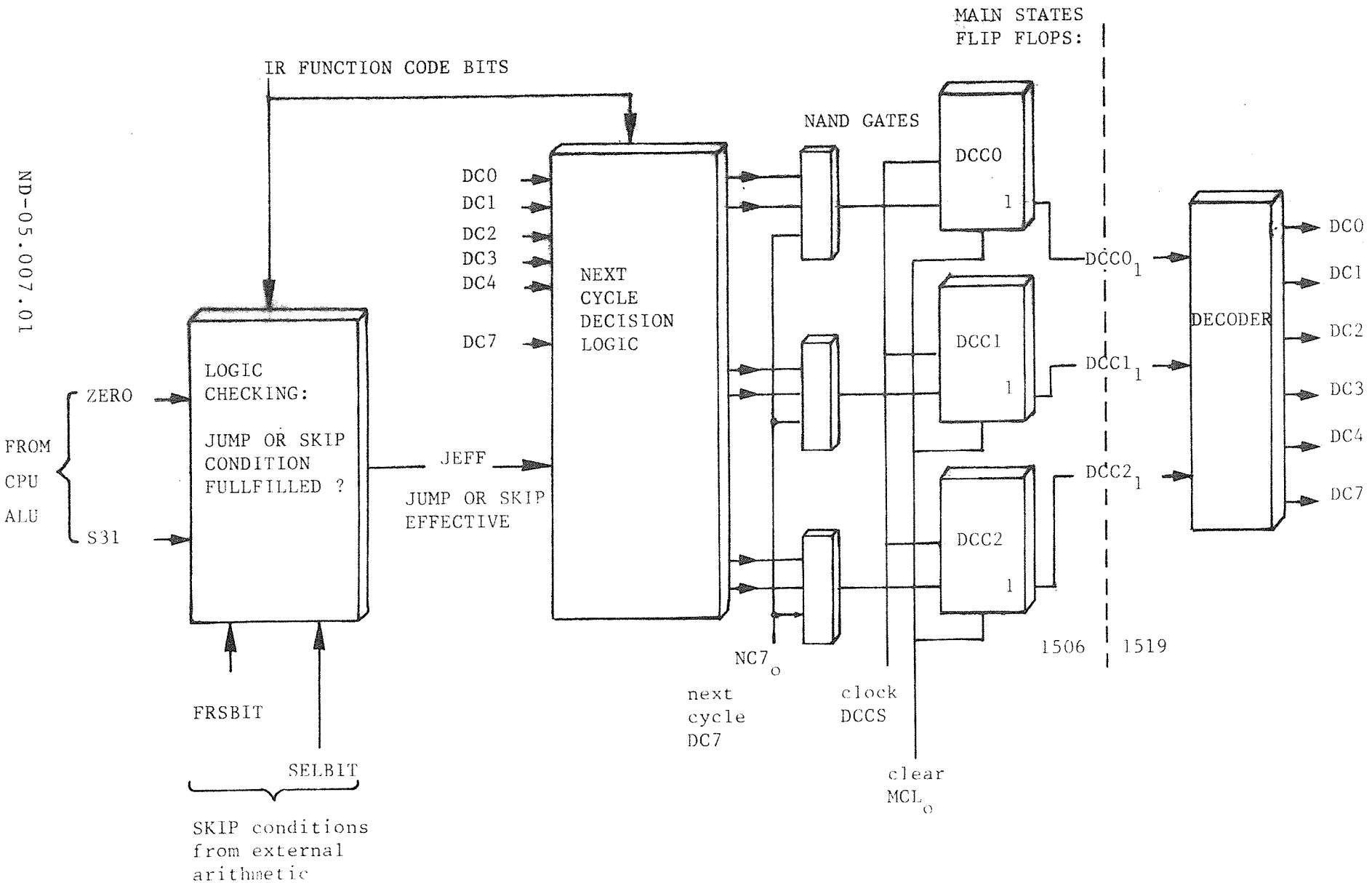
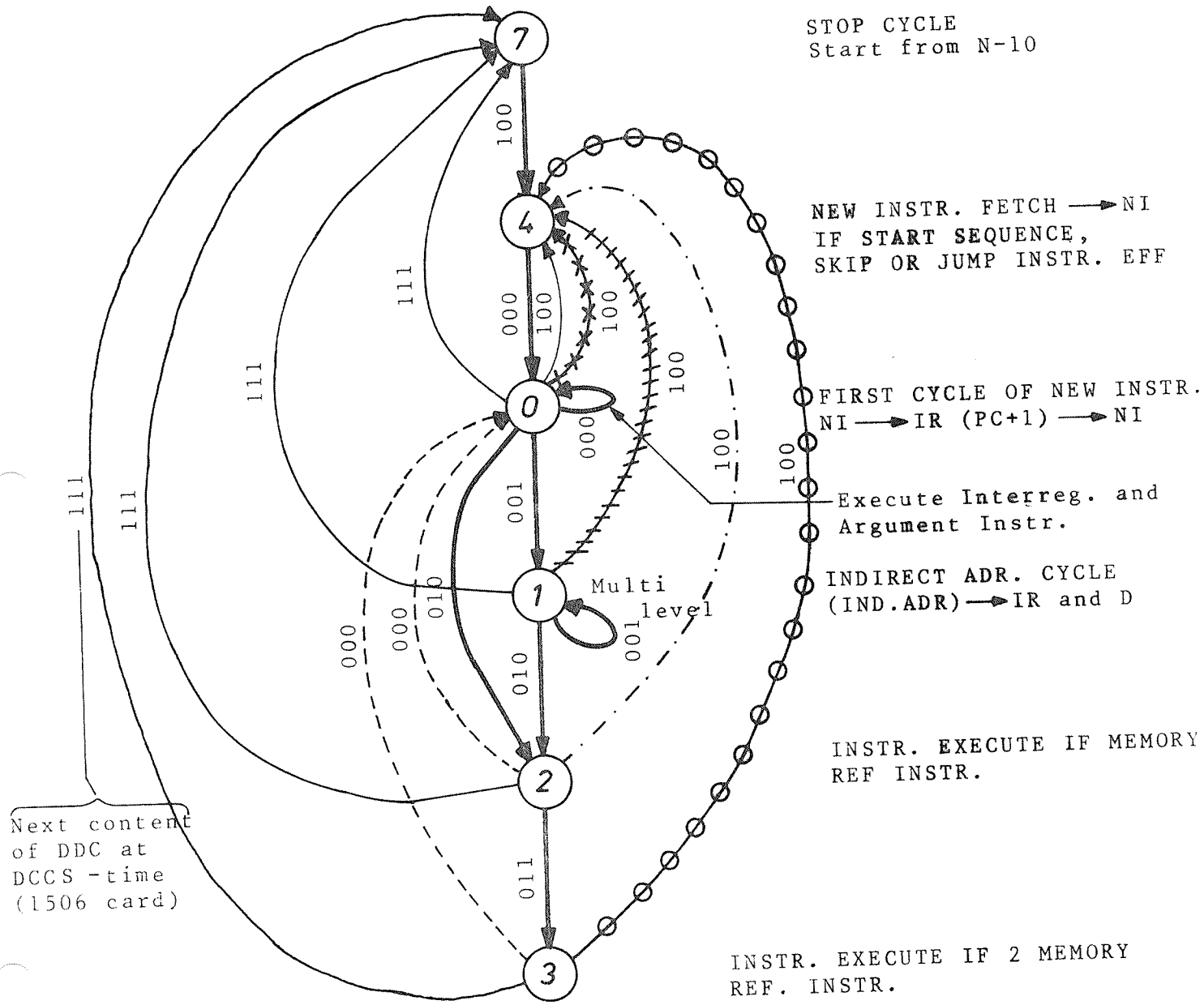


Figure III.9.1



CONTINUE TO NEXT CYCLE

JUMP EFF

INDIRECT JUMP EFF

SKIP EFF

} To fetch new-instruction

RESULT OF MIN = 0

NEXT INST

Way back to execute
NEXT INSTRUCTION

STOP

REGISTER SKIP EFF

Figure III.9.2 MAIN STATE DIAGRAM

III.9.1 CYCLES INVOLVED EXECUTING A MEMORY REFERENCE INSTRUCTION

Underlined: Instruction fetch

Normal type: Instruction execution

DC7: STOP CYCLE

N10/N50 communication is active to load start address to SA, break addresses to BP and BQ and break conditions to modus register M.

Activate N50 by SAA4; IOX NOR50.

SA to program counter PC via the memory address bus MAC. Request instruction pointed to by PC.

A simulated Data Ready signal (DRS) is generated to change cycle to DC4.

DC4: INSTRUCTION TO NEXT INSTRUCTION REGISTER (NI) CYCLE

As Address Ready (AR) from memory appear:
PC + 1 $\longrightarrow$ PC. Request instruction now pointed to by PC (next instruction).

At Data Ready (DR) time: Data now ready equals instruction requested in DC7. Clock instruction to NI register (NIS1).

Change cycle to DC0.

DC0: EXECUTE AND MEMORY OPERAND READ/WRITE CYCLE

NI $\longrightarrow$ Instruction Register IR (IRS)

Calculate operand address CA.

Memory write instruction:	x	Memory read instruction
	x	or Indirect address-
	x	ing:
Execute instruction in	x	
IR. Enable result to	x	
memory data bus (MDC).	x	

At AR time:
PC + 1 $\longrightarrow$ PC (PC is now 2 ahead of executing instruction).

Request calculated address CA.

At DR time:
Clock next instruction requested in DC4 ⟶ NI.

Enter cycle DC2

x Indirect ? enter DC1
x otherwise DC2.
x

DC1: INDIRECT ADDRESS CYCLE

At DR time:

Clock data = indirect address into IR and displacement register D.

IR31 = 1

x
x
YES (Multilevel indirect) x NO
x

Request calculated address.

Extend DC1

x Enter cycle DC2.
x DC2 * if two data
x references in memory.
x DC2 if one data
x reference in memory.

DC2: INSTRUCTION EXECUTE CYCLE
(ONE DATA REFERENCE)

At DR time:
Request instruction now pointed to by PC.

Memory Write Instruction x Memory Read Instruction:
x
Instruction finished x Latch operand on
x data bus into B operand
x latch.
x
x Execute instruction in
x CPU ALU or external
x arithmetic.
x

At delayed DR time:

x Write result into
Enter cycle DC0 x destination register
x if result from CPU -
x ALU, enter DC0.

x If SKIP or JUMP effective
x enter DC4 to fetch new
x instruction to NI.

x Instruction executed
x in external arithmetic:
x At External Data Ready
x (RYX) time:
x
x Write result to destin-
x ation register, enter
x cycle DC0.

DC2*: INSTRUCTION EXECUTE CYCLE
(TWO DATA REFERENCES)

At DR time:
Request CA address if: MIN, XMR or ADM.
Request CA + 1 address if floating double (FD) instruction.

Memory Write:

x Memory Read:
x
x latch operand read in
x B latch if MIN, XMR or
x ADM.

First operand written if FD

x
x
x Latch first operand
x (Bit 63-32) if FD
x instruction.

At delayed DR time:

x Write first operand to
x FD register if LDD.

Enter cycle DC3.

DC3: INSTRUCTION EXECUTION SECOND CYCLE

At DR time:
Request instruction now pointed to by PC.

Memory Write:

x Memory Read:
x

MIN, XMR and ADM, operand written

x
x
x

Second FD operand written.

x Latch second operand
x (Bit 31-0) if FD
x instruction.

At delayed DR time:

x Write second operand
x to FD register if LDD
x

Enter cycle DC0.

If MIN skip effective enter cycle DC4 to fetch new instruction to NI.

x Enter DC0 at external
x ready time if
x instruction executed
x in external arithmetic

III.9.2 CYCLES INVOLVED EXECUTION AN INTER-REGISTER OR ARGUMENT INSTRUCTION

Underlined: Instruction Fetch

Normal types: Instruction execution

DC7: STOP CYCLE

NORD-10/NORD-50 communication is active to load start address to SA, break addresses to BP and BQ and break conditions to MODUS register M. Active NORD-50 by SAA 4; IOX NOR50.

SA to Program Counter PC via the memory address bus MAC. Request instruction pointed to by PC.

A simulated Data Ready Signal (DRS) is generated to change cycle to DC4.

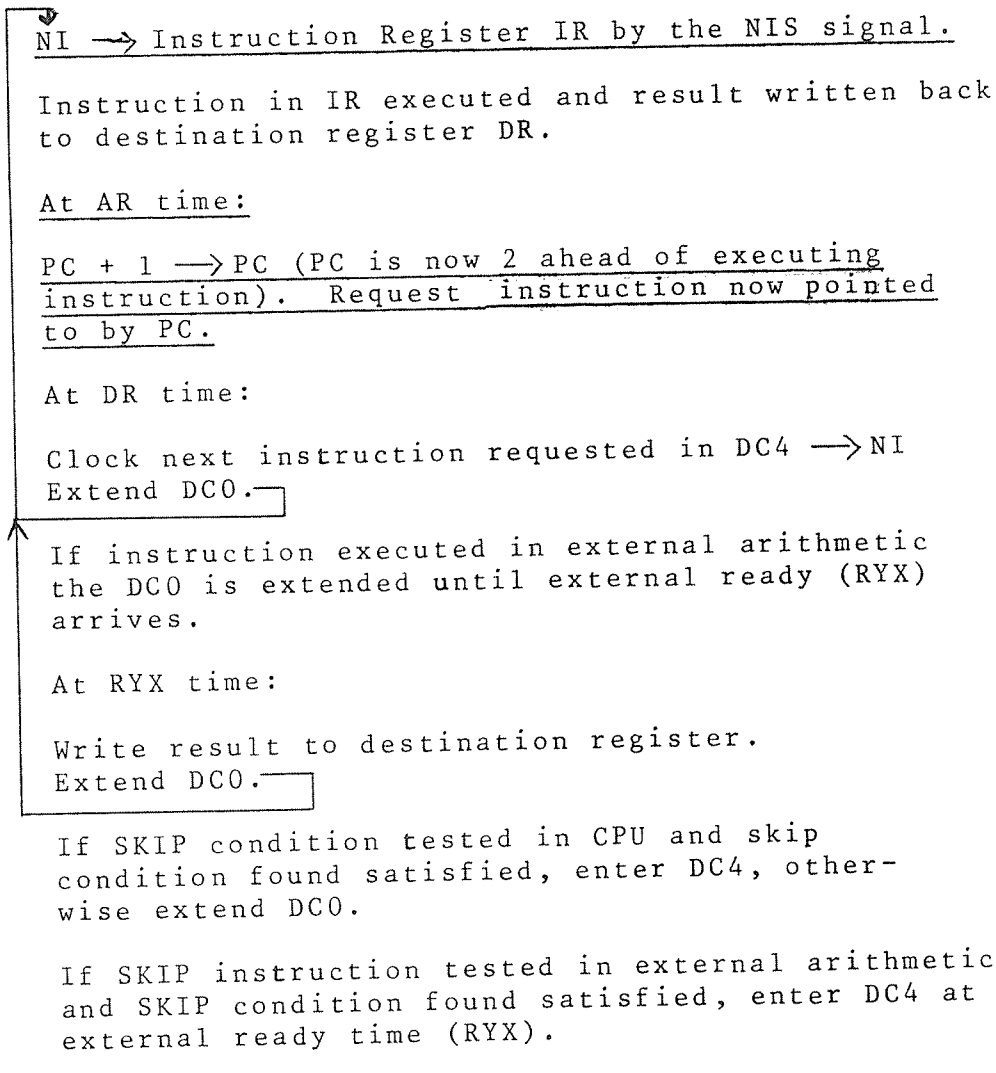
DC4: INSTRUCTION TO NEXT INSTRUCTION ON REGISTER (NI) CYCLE

As Address Ready (AR) from memory appear:
PC + 1 ⟶ PC. Request instruction now pointed to by PC (next instruction).

At Data Ready (DR) time:
Data now ready equal instruction requested in DC7.
Clock instruction to NI register (NIS1).

Change cycle to DC0.

DC0: INSTRUCTION EXECUTE CYCLE



III-10 TIMING SEQUENCE

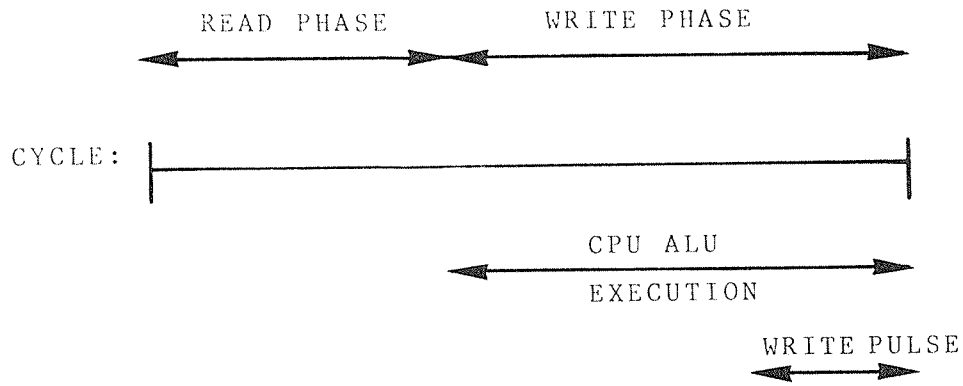
Each basic cycle in NORD-50 where instructions are executed has two phases, the read and the write phase. Decoded on the 1519 card and named DCR0 - DCR3 (Read phase cycle DC0 - 3) and DCW0 - DCW3 (Write phase cycle DC0 - 3).

During the READ phase of cycle DC0 the operands are read from the register in the register block given in the source address field and clocked into the operand latches.

During the READ phase of cycle DC2 the operand is read from memory and clocked into the operand latches.

When the operands have been latched, the phase can be changed to write. During an interregister or argument instruction execution in cycle DC0, the register address is changed to destination address for both register blocks (At SP time on 1519). At the register WRITE pulse the output of the ALU is written into both register blocks.

For a memory reference instruction the write phase of DC2 will be used to write the result to the register if the register was the destination or back to memory if memory was the destination.

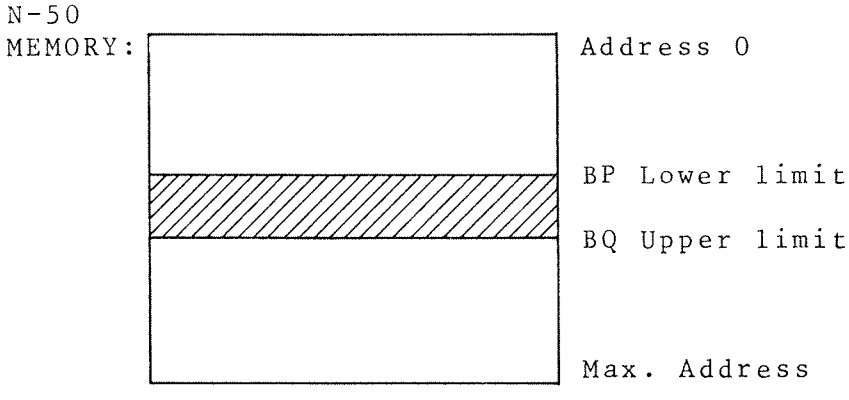


III.1.1 MEMORY PROTECT SYSTEM

The heart of the memory protect system in NORD-50 is the address limit registers BP and BQ.

BP = Break Point Register No. 1 (20 bits)
Lower address limit, and

BQ = Break Point Register No. 2 (20 bits)
Upper address limit



The BP and BQ register can be used to protect either the memory area bounded by the addresses within the two registers, or the memory area outside these bounds against any of the following:

- All accesses
- Instruction fetches
- Data read and write
- Data write only

The break conditions are given in the following bits in the modus register:

MODUS BIT NO.:	
4	Stop if BP $\leq$ Any Reference $<$ BQ
5	Stop if BP $\leq$ Program Counter $<$ BQ
6	Stop if BP $\leq$ Data Reference $<$ BQ
7	Stop if BP $\leq$ Data Store Reference $<$ BQ
8	Invert limit on bit 4-7: Stop if $0 \leq x < BP$ or $BQ \leq x <$ Max. address

The address lines MAC to memory have two sources (the third is used for start-up) selected through on the 1501 card.

1	Program Counter	%Selected when instruction fetch
2	The Address Arithmetic	%Selected when reading or writing memory operand address

The memory address lines MAC are continously compared against the BP and BQ registers on the 1501 card. The result of this comparison is the signals

LBP 20 = Referred address less than BP and

LBQ 20 = Referred address less than BQ

These signals are fed to the 1504 card.

On the 1504 card these signals are checked against the stop-conditions given in modus bits 4-7.

If modus bit 8 is set, the LBP 20 and LBQ 20 signal is inverted before checking.

The result of this comparison (the signal ABPBQ) is then used to decide whether a request can be sent to memory or not.

If the address points to the protected area, the request will be blocked. NORD-50 will be stopped, entering cycle DC7 by the NC7 signal, and as a response to that a completion signal is sent to the NORD-10.

Status bit 1 will be set, and the TA register will hold the address that caused the stop.

To compare the memory address against BP and BQ takes time, approximate 90 nsec.

The request will be delayed for 100 ns to be sure that the address is legal. This is done by the IPROT signal on the 1519 card, which will extend the cycle where the request is initiated for 100 ns.

Modus bit 4

M4 = 1 : A
(All references) : All memory references will be checked.
All cycles will be delayed.

M5 = 1
(Instruction fetch) : All cycles where the program counter is selected to the MAC bus is delayed.

M6 = 1
(Data read and write): All memory reference instructions execute cycles where the address arithmetic is selected to the MAC bus is delayed.

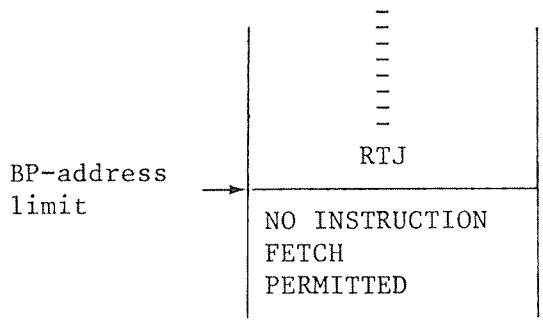
M7 = 1
(Data write) : As for M6 but only memory reference instructions execute cycles where memory is destination is delayed.

III.11.1 PROTECT ADDRESS SETTING

Because of the NORD-50 instruction prefetching, the following is to be considered:

Assume that the break condition for INSTRUCTION FETCH is set in the MODUS register. The lower break address register BP is set, and in the location before the BP-address an instruction is to be executed.

Because of the prefetch, the execution of this instruction will generate an address violation interrupt.



The rule is thus:
The last location before an address limit must not contain an instruction to be executed.

This is the case if the protect-mode is

1. All references
2. Instruction fetch

III.12 CPU AND MEMORY OPERANDS TO EXTERNAL ARITHMETIC

The operands to the external arithmetic are selected through two selectors in CPU, each selecting 32 bits (2 x 4 bits on each 1502 card).

The operands are latched in the operand latches and driven by a tri-state driver to the external arithmetic.

The operands may either be 32 bits (for instance Floating Add Single Precision) where only one operand selector/latch/driver is enabled, or 64 bits where both the operand selector/latch/driver operates with 64 bits in parallel.

Memory Reference Instructions

A memory reference instruction using the external arithmetic (FAD, FSB, FMU ...) will read one operand (the B-operand) from the calculated address on the MB-lines from memory and select it through the darkened part of the B-operand selector.

Refer to figure III.12.1.

The A-operand selector will have its input from the register block A, and only a 2-input selector is used.

A floating double precision operand is stored in memory in two consecutive locations.

During a double precision floating memory reference instruction, the 32 most significant bits are read from the calculated address and latched in the operand latch (bits 32-63) by the STRB2 pulse.

An extra cycle is entered to read the least significant operand bits (bits 0-31) from the CA+1 location, select them through the operand selector and latch them in the operand latch by the STRB1 pulse.

The 64 operand bits can then be presented in parallel.

Inter-Register Operation

The A-operand is, as for memory reference instructions, taken from the Register Block A.

The B-operand is taken from the Register Block B.

The B_L lines out of registers 16-31 and 48-63 will always carry the bits from 0-31, while the B_M lines out of registers 0-15 and 32-47 will contain either bits 32-63 in the operand for double register or floating double register operations or bits 0-31 if none of these operations.

The registers 16-31 and 48-63 are not in use for single precision floating.

During double register (shift double) or double floating register instructions 64 bits are selected through the operand selector in parallel.

CPU AND MEMORY-OPERANDS TO EXTERNAL-ARITHMETIC
(B-operand shown)

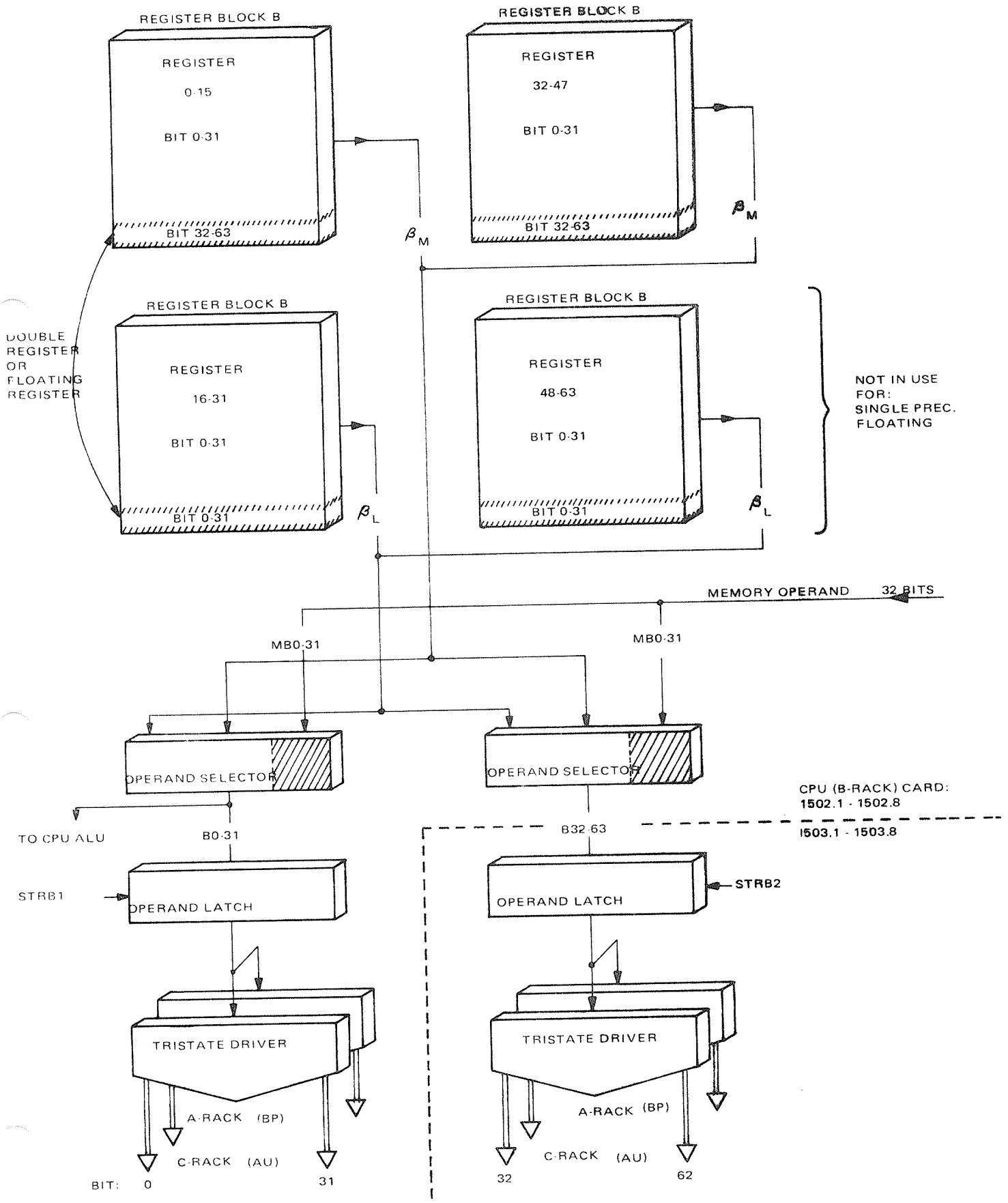
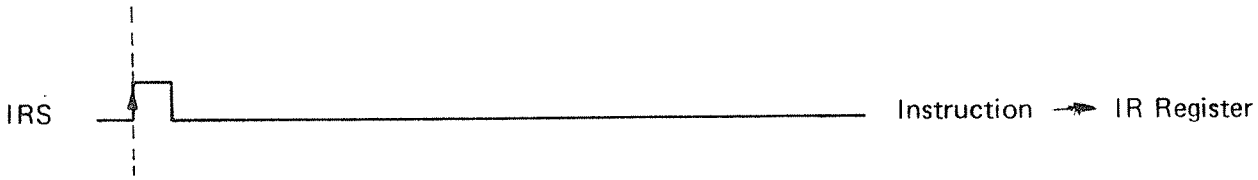


Figure III.12.1

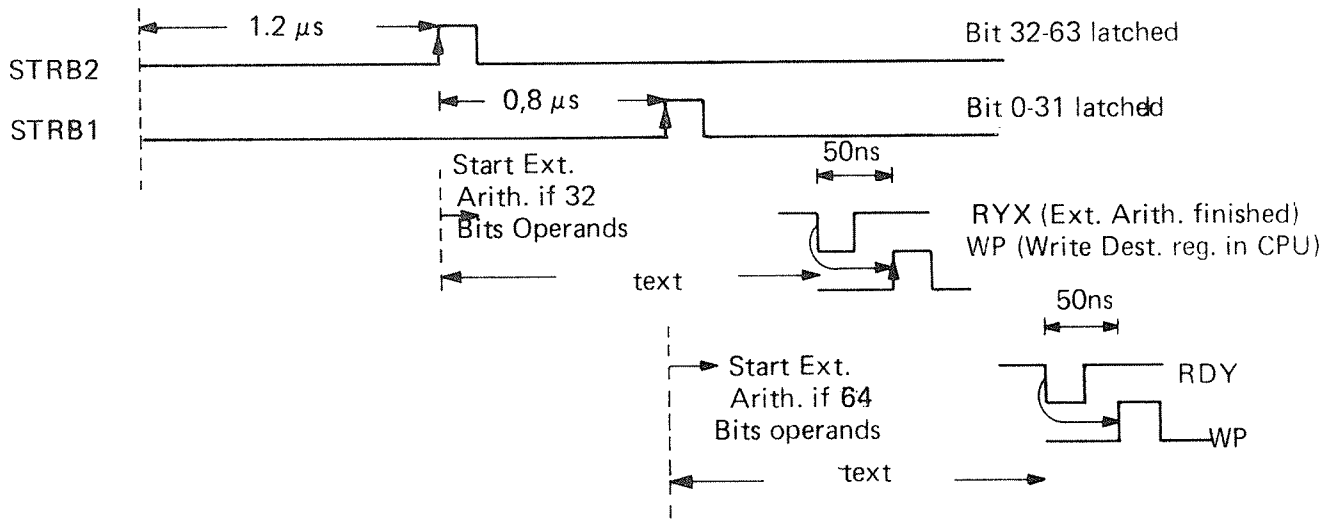
OPERAND SELECT AND EXTERNAL ARITHMETIC TIMING:

Memory Reference Instructions:

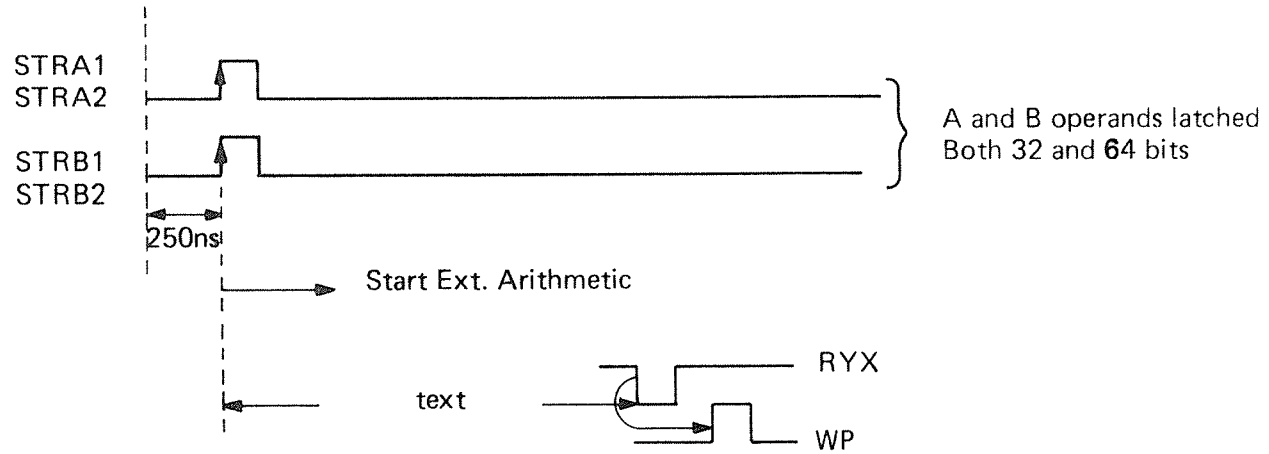


A-Operand

B-Operand



Inter-Register Instructions:



Text will depend on type of instruction.

III.13 RESULT FROM EXTERNAL ARITHMETIC BACK TO CPU

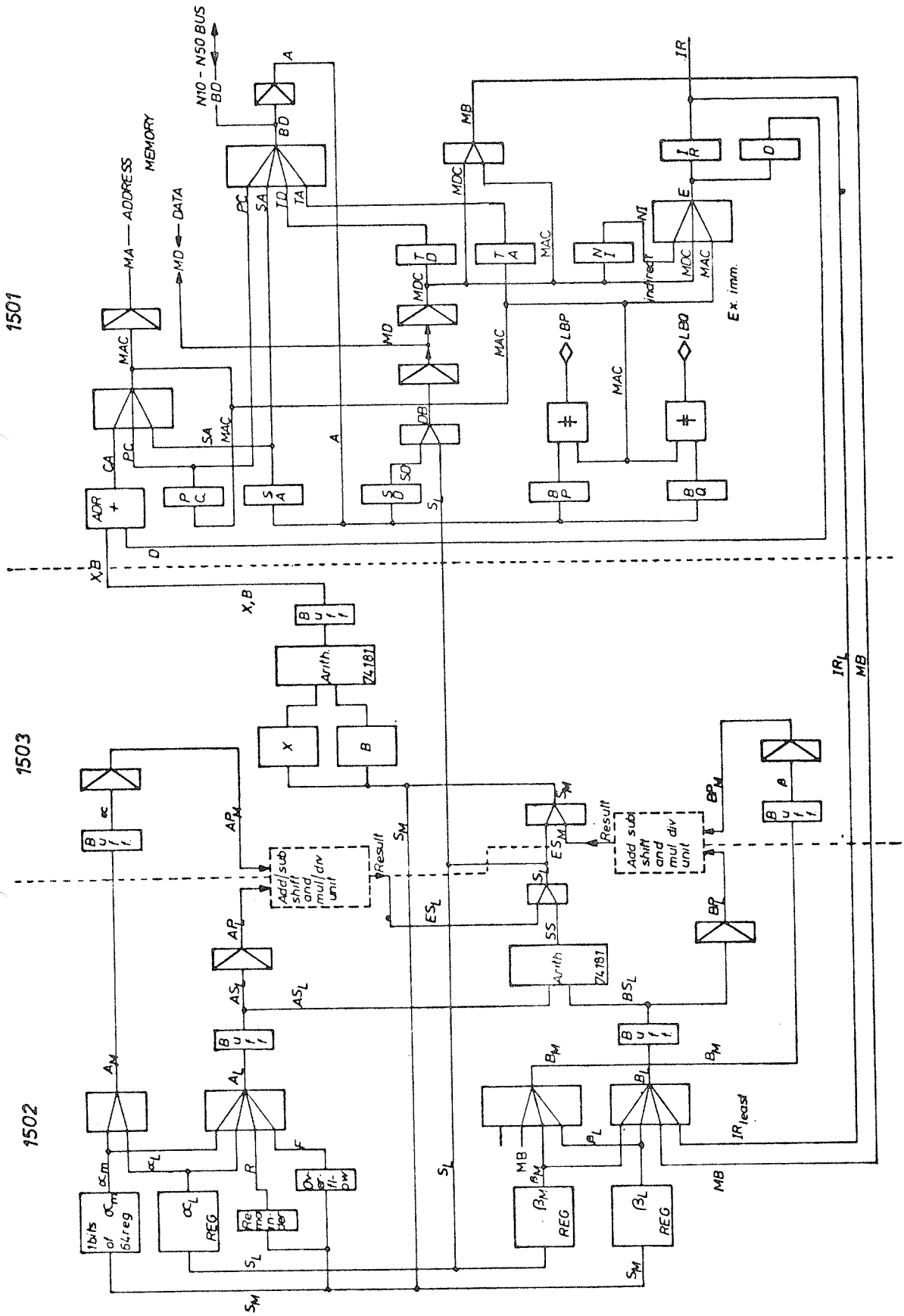
All results achieved in the external arithmetic are written back into the Register Block A and B in parallel. The register number is specified in the destination field in the instruction.

When the ready signal (RYX) from the external arithmetic arrives, the result is ready on the data lines and the ready pulse triggers the register write pulse (WR) in CPU on the 1519 card.

When the external arithmetic returns a double precision, 64 bit word as a result, all the 64 bits are written back simultaneously. Bits 32-63 via the most significant SUM bus SM and bits 0-31 via the least significant SUM bus SL.

When executing integer multiply (MPY) or integer divide (DIV) in external arithemtic, the result on the most significant SUM bus will be the overflow (more than 32 bits result) for multiply and remainder for divide.

When the RYX signal arrives, the CPU is pre-determined to write the most significant SUM bus into the OVERFLOW register if FMU and into the REMAINDER register if DIV. This is done in parallel with writing the least significant SUM bus (the result) into the destination register.



N-50 CPU DATAFLOW
6B14

Ajourf. 22.6.77 PK/L
Tegn 11.7.75 PK/B

SECTION IV

EXTERNAL ARITHMETIC C-RACK

DETAILED CONTENTS

IV.1 THE C-RACK

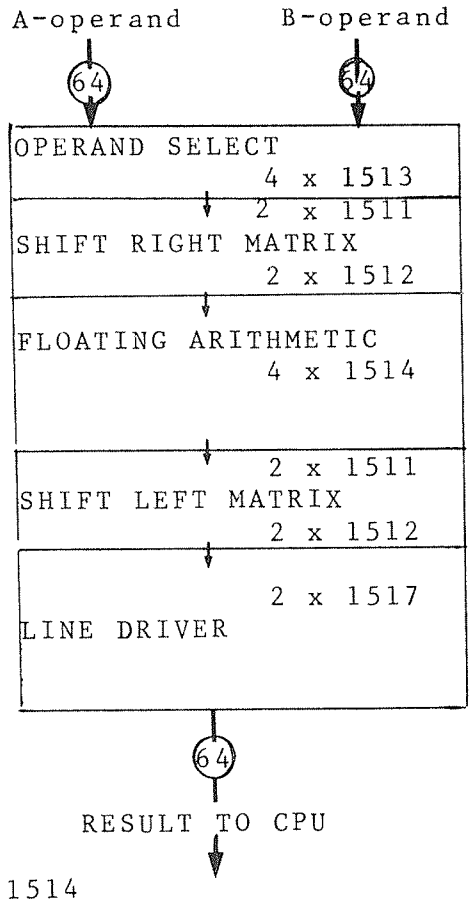
In the C-rack, as part of the external arithmetic, the following instructions are executed:

Mnemonic:	Memory Ref. Instr.	Inter. Register Instr.	Means:	No.of operand bits:
FAD	X		Floating Add F+(Ea)	32
FADD	X		Floating DP Add FD+ (Ea, Ea+1)	64
RAF		X	Floating Reg. Add	32
RAFD		X	Floating DP Reg.Add	64
FSB	X		Floating Subtract F - Ea	32
FSBD	X		Floating DP Subtract FD - (Ea, Ea + 1)	64
FIX		X	Convert floating to Integer	32/32
FIXD		X	Convert DP floating to Integer	64/32
FIR		X	Convert floating to Rounded Integer	32/32
FIRD		X	Convert DP floating to Rounded Integer	64/32
FLO		X	Convert Integer to Floating	32/32
FLOD		X	Convert Integer to DP Floating	32/64
		X	Shift a 32 or 64 bit operand up to 63 places to the right or left with arithmetic, logical or rotational shift	32 or 64
BST		X	Bit set	32
BCL		X	Bit clear	32
BCM		X	Bit complement	32
BSZ		X	Bit skip on zero	32
BSO		X	Bit skip on one	32

As the table indicates, four of the instructions are memory reference instructions (one operand from memory and one from a register) and the rest are inter-register instructions (both operands, if two, are taken from the register block).

DATA FLOW

The C-rack consists of this main card set:



OPERAND SELECT 4 x 1513

Four 1513 cards select the operand to the internal bus.

SHIFT RIGHT MATRIX

Two 1511 cards shift the operand 0, 8, 16, 24, 32, 40, 48 or 56 places. The shift can be rotational or arithmetic.

Two 1512 cards shift the operand 0, 1, 2, 3, 4, 5, 6 or 7 places.

FLOATING ARITHMETIC 4 x 1514

The floating mantissas are added or subtracted on these cards. (Exponent arithmetic on card 1515.)

Shift operands go directly through. For bit instructions, the specified bit should now be in bit 0 and is manipulated and checked on card 1516.

SHIFT LEFT MATRIX

As right shift matrix but shifted left.

LINE DRIVER

Drives result on tri-state lines to CPU.

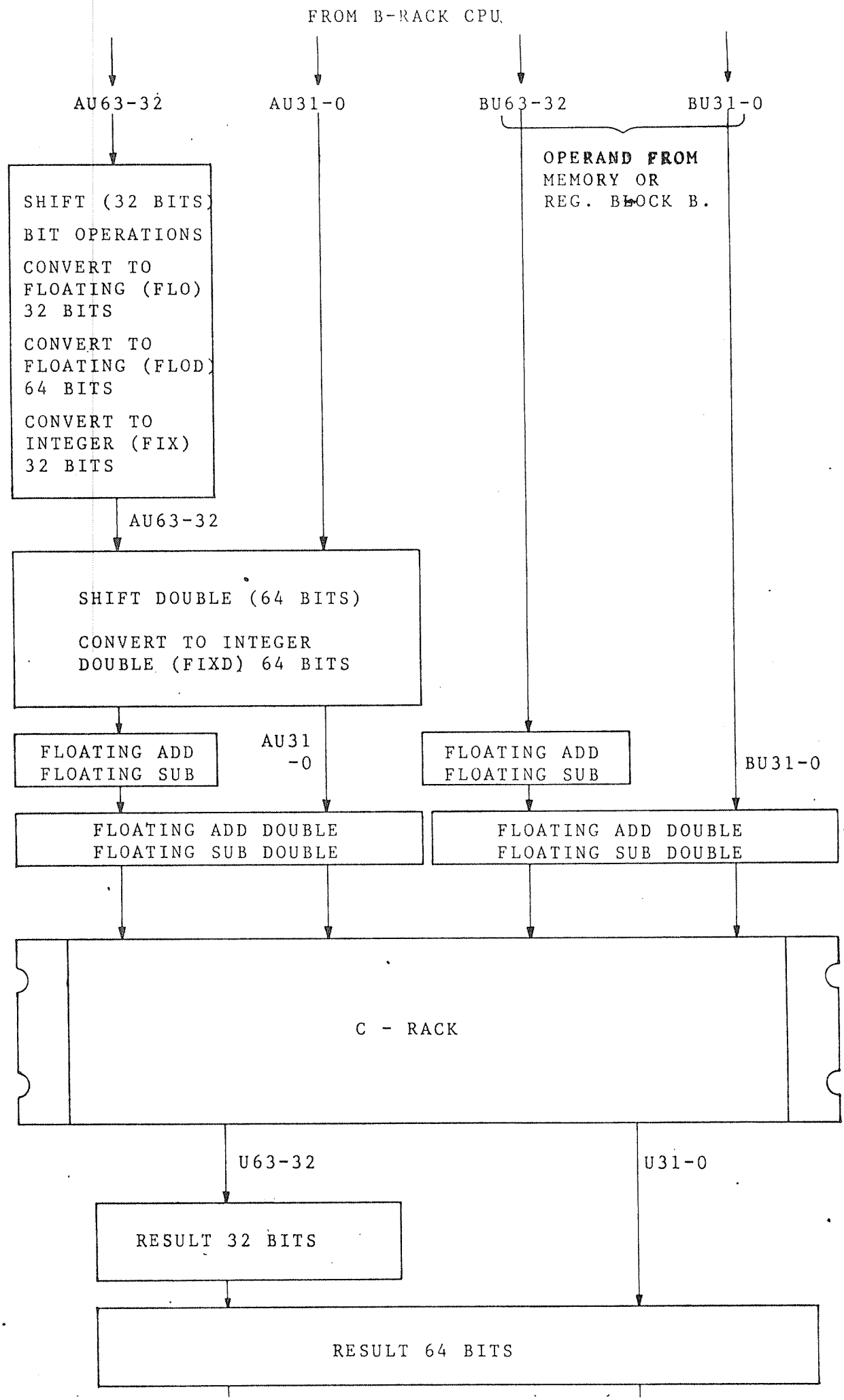
TIME USED:

All instructions executed in the C-rack take the same amount of time. The time from the operands are presented until the result is on the result bus to the CPU is approximately 400 ns.

The operands passing through the C-rack are never latched, so the time used is given by adding the integrated circuit delays together.

The OR of all start signals to the C-rack triggers a one shot on the FLOATING CONTROL 1516 card. This one-shot is set to approximately 400 ns equal the time the operands need passing through the logic and the result is ready on the data lines to CPU.

C - RACK DATA FLOW



CONTROL SIGNAL TO THE C-RACK:

In addition to the operand data given in Figure IV.1.1, the following control signals are decoded in the CPU and sent to the C-rack.

	Gen. on cards:	Means:	
SFAD	1508	Start Floating Add	operands latched and ready in CPU
SFSB	1508	Start Floating Subtract	
SOPR	1508	Start Bit-, Shift- or Convert instructions	
DPA	1505	Double precision floating point	

The following IR signals are buffered in the CPU (1505) and sent to the C-rack.

IRX 0 - 5
IRX 23 - 30

SIGNALS BACK TO THE CPU

	Gen.:	Means
SELBIT	1516	Specified bit used by the CPU in Bit-Skip instructions
FRSBIT	1516	Floating register compare bit used by CPU to determine SKIP condition
OF2	1516	Floating or Integer (FIX) Overflow
UF2	1516	Floating Underflow (Result equals 0)
RYP	1516	Result ready on data lines to CPU.

IV.2 SHIFT MATRIX

In the NORD-50 all shift operations are performed on two cards: 1511 and 1512. The same cards are used both for right and left shifts.

The 1511 card shifts the operand 0, 8, 16, 24, 32, 40, 48 or 56 places, while the 1512 card shifts the operand 0, 1, 2, 3, 4, 5, 6 or 7 places.

The output of the 1511 card is connected to the input lines on the 1512 card. This makes it possible to shift the operand any combination of shifts from 0-63.

For shifting a 64 bit operand, 2 of each of these cards are needed.

The cards are built up of 8-1 line sectors (74151), where the 3 select lines, drawn on the top of the IC, determine which operand bit to be selected to the output, i.e., the number of places to shift the whole operand.

In this way, the time involved in shifting is independent of the number of shifts. Shifting an operand 1 place takes the same time as shifting an operand 63 places. The time used is equal to the IC delay.

On the 1511 card, the IR bits 3-5 give the number of places to shift the operand, while on the 1512 card, IR bits 0-2 give the number to shift the operand.

On both cards, the output from the selectors can be disabled (not shown on the drawing), getting only zero from the shift matrix.

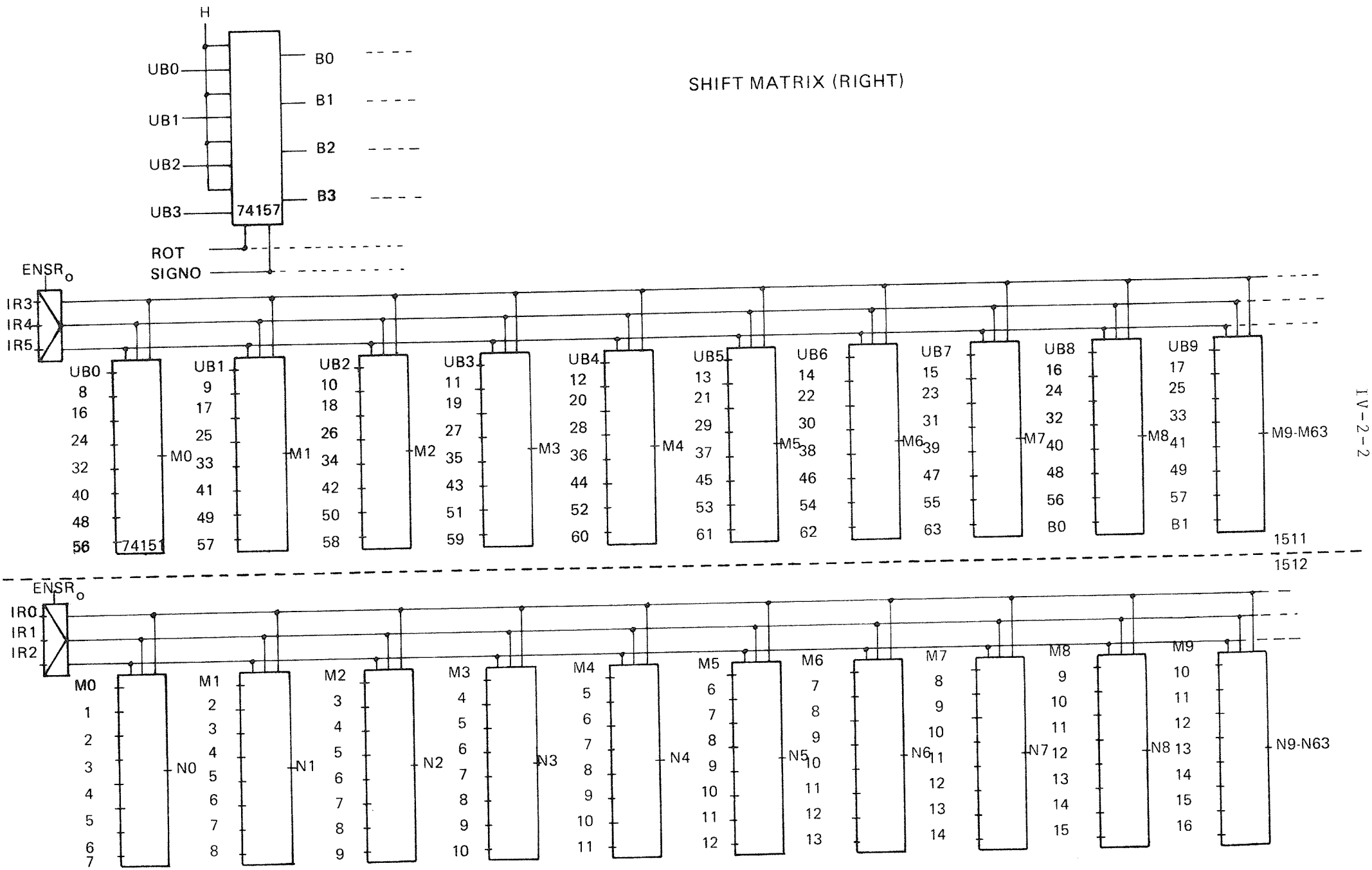
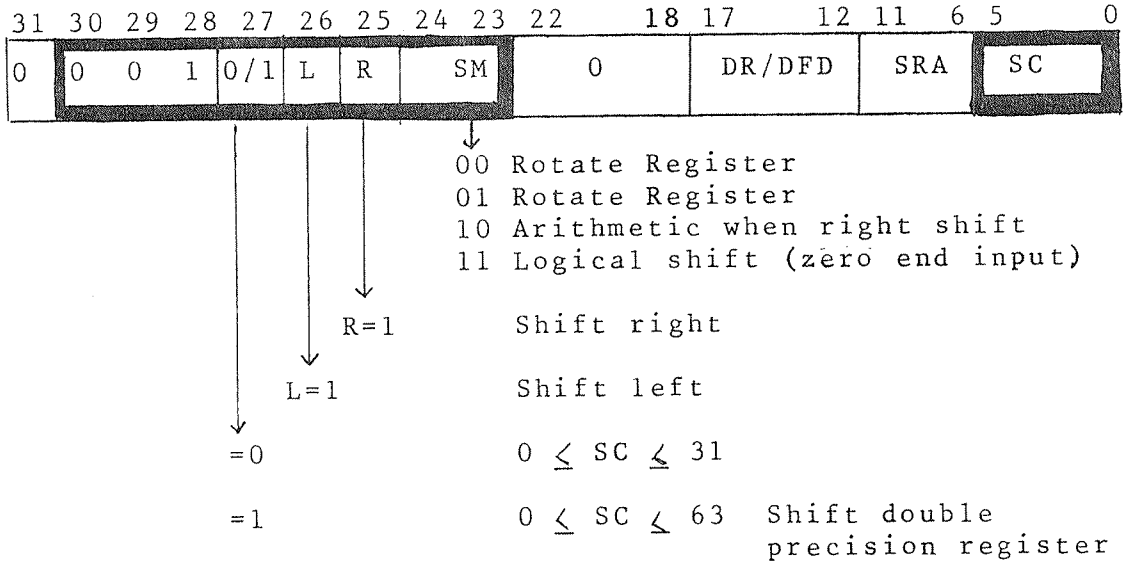

SHIFT MATRIX (RIGHT)
H
UB0
UB1
UB2
UB3
74157
B0
B1
B2
B3
ROT
SIGNO
ENSR$_o$
IR3
IR4
IR5
UB0
UB1
UB2
UB3
UB4
UB5
UB6
UB7
UB8
UB9
M0
M1
M2
M3
M4
M5
M6
M7
M8
M9-M63
74151
1511
1512
IR0
IR1
IR2
N0
N1
N2
N3
N4
N5
N6
N7
N8
N9-N63

Figure IV.2.1

IV.3 SHIFT INSTRUCTIONS

Shift register instruction format in IR register:



= IR signals sent to C-rack.

During right shift (R=1) the IR bits 5-0 are enabled to the SHIFT RIGHT MATRIX (1511, 1512) to give the shift count.

During left shift (L=1) the IR bits 5-0 are enabled to the SHIFT LEFT MATRIX to give the shift count.

As indicated on the C-rack data flow, the shift operands go as the A operand and during 32 bits shift the operand is presented on the most significant AU lines.

On the SELECTOR cards 1513, the 32 least significant bits are set to zero. So during any shift 64 bits are always shifted.

During arithmetic shift (SM=2) right AU 63, the sign bit is extended during the shifting. (sign extension SIGNO)

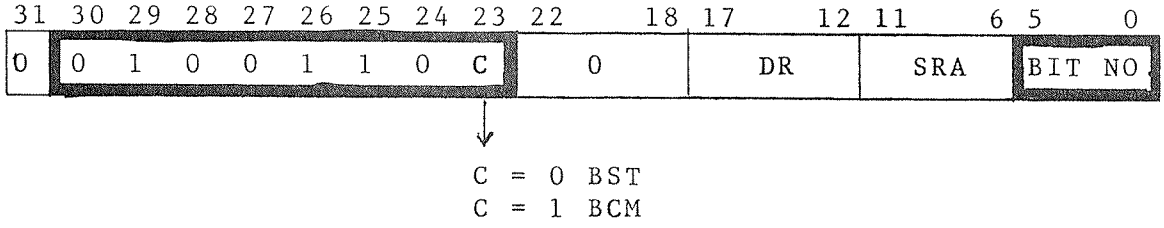
In left shift, zeros are fed into vacated bit positions.

In logical shift, the bits which are shifted out of the word are lost and zeros are put in the other end.

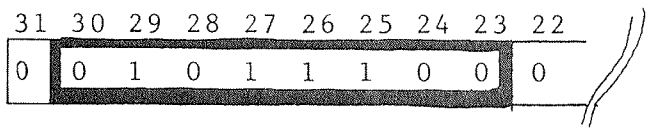
IV.4 BIT INSTRUCTIONS

Bit instructions format in IR register:

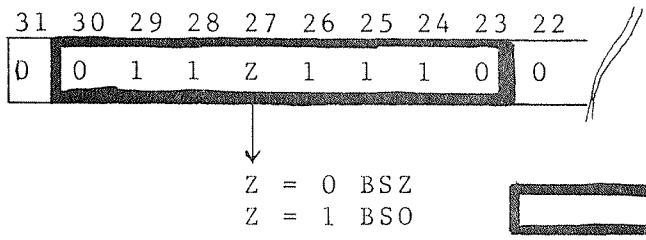
BIT SET (BST), BIT COMPLEMENT (BCM)



BIT CLEAR (BCL)



BIT SKIP ON ZERO (BSZ), BIT SKIP ON ONE (BSO)



During Bit operations, both the SHIFT RIGHT and the SHIFT LEFT matrix are enabled.

The IR bit 5-0 is used to shift the specified bit number to bit position 0 in the shift right matrix (rotational shift).

When the bit is in position 0, the bit is either SET, COMPLEMENTED or CLEARED. This is done on the 1516 card. In the shift left matrix, the whole operand word is shifted back leaving the specified bit in its original position.

BIT SKIP instructions are executed in the same way, but this time, the specified bit is selected and gated back to the CPU (1516) where it is checked against the skip condition (1506). If the SKIP condition is satisfied, the CPU skips the next instruction.

IV.5 CONVERTING INTEGER TO FLOATING (FLO)

CONVERTING INTEGER TO DOUBLE PRECISION FLOATING (FLOD)

INSTRUCTIONS

Hardware Execution:	On Card:
Select the operand AU63-32 to the internal bus UB63-32. Set UB31-0 equal to zero.	1513
Shift UB63-32, 32 places right. Sign extension (ARITHMETIC shift) if AU63 was equal to one (negative integer).	1511, 1512
Invert (2's complement) if AU63 was equal to one, otherwise not. The priority encoder looks for the most significant one bit. This information, equal to the number of left shifts counted from the most significant mantissa position (SLZ0-5), is fed to the 1515 card and subtracted from a ''normalized'' exponent. (An exponent bias 400_8 and the maximum number of shifts ($55_{10} = 67_8$) added.) The exponent looks like this:	1514.1,1514.2
100 110 111 bias 67	1515
The output of this subtraction is a ready biased exponent (UD54-62).	
The number of left shifts (SL0-5) is fed to the SHIFT LEFT matrix where the integer is shifted to be a normalized mantissa. The exponent part is masked away.	1511, 1512
The mantissa bit 54 is removed.	1512.4
The exponent UD62-54 is fed from the 1515 card, and the shifted mantissa is fed from the shift left matrix to the DATA BUFFER card, driving the result (U) to the CPU.	1517
The sign bit U63, equal to AU63, is driven from the 1516 card to the CPU.	1516

IV.6 CONVERT FLOATING, DOUBLE PRECISION FLOATING TO INTEGER (FIX, FIXD)

CONVERT FLOATING, DOUBLE PRECISION FLOATING TO ROUNDED INTEGER (FIR, FIRD)

Hardware Execution:	On cards:
The exponent part of the floating operand AU62-54 is fed directly to the 1515 card. The mantissa part is selected to the internal bus (BU) to the SHIFT RIGHT matrix.	1513
The exponent part is subtracted from a "normalized" exponent, giving the shift count to the mantissa to form an integer (RSH0-4). The magnitude of the exponent is checked. Integer overflow (PIOFL) is generated if exponent $> 32_{10}$.	1515
Integer overflow (SIOFL) is detected in case of FIR, FIRD when the result before rounding is the greatest possible integer and the rounding adds a one.	1516
The mantissa is shifted in the SHIFT RIGHT matrix according to the value of the exponent (RSH0-4).	1511,1512
In case of PIOFL, the SHIFT RIGHT matrix is disabled (NENBL) giving the greatest possible integer as output.	1512
The integer is inverted (2's complemented) if AU63 was equal to one.	1514
Direct through the SHIFT LEFT matrix, no shift.	1511,1512
Integer driven to CPU.	1517

IV.7 FLOATING ADD. SUBTRACT

The process of adding/subtracting two normalized floating point numbers can be devided into four steps:

1. Selecting the greatest exponent as the main or result exponent.

2. Shifting the mantissa of the operand with the smallest exponent.

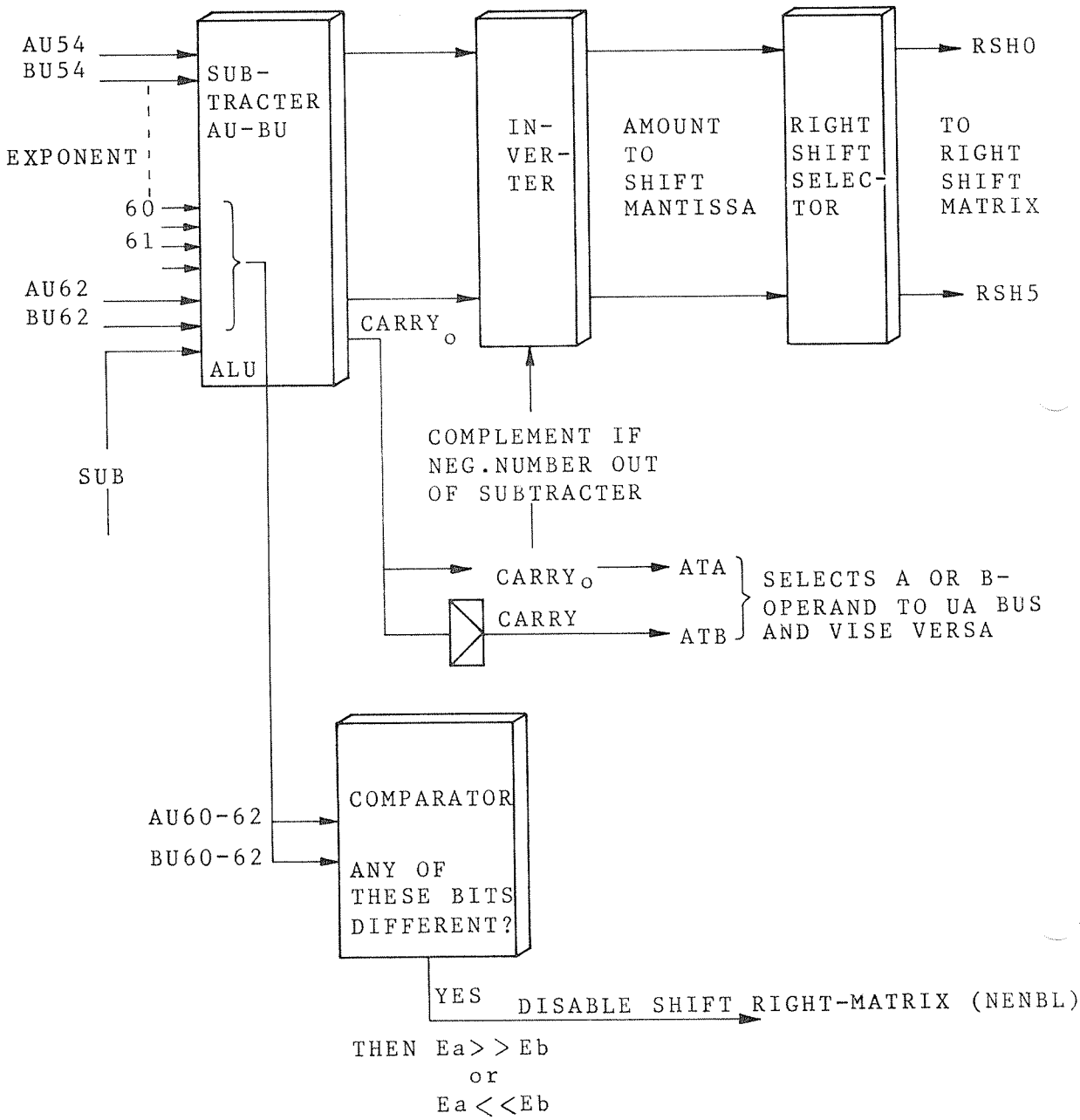
3. Add/subtract the main and the shifted mantissa.

4. Normalize the result mantissa.

The A-operand exponent AU62-55 is subtracted from the B-operand exponent BU62-55 (Refere to figure IV.7.1).
The carry signal of this subtraction is used to select the greatest exponent as the result exponent via the UA BUS, and select the mantissa of the smallest exponent as input to the shift-right matrix via the UB-BUS.
The output of the exponent-subtrator gives the shift count to shift the mantissa on the UB-BUS right.

If BU>AU the subtractor output will be negative and the absence of the carry is then used to invert the output to generate a positive shift-count.

The three most significant bits in the exponent are compared to see if one of the exponents is much greater (>>) than the other. If that is the case, the greatest operand is taken as the answer and the output of the shift-right matrix is disabled. (Zero is added to the final result).



EXPONENT MAGNITUDE COMPARATION CIRCUITS

(ON 1515 CARD)

Figure IV.7.1

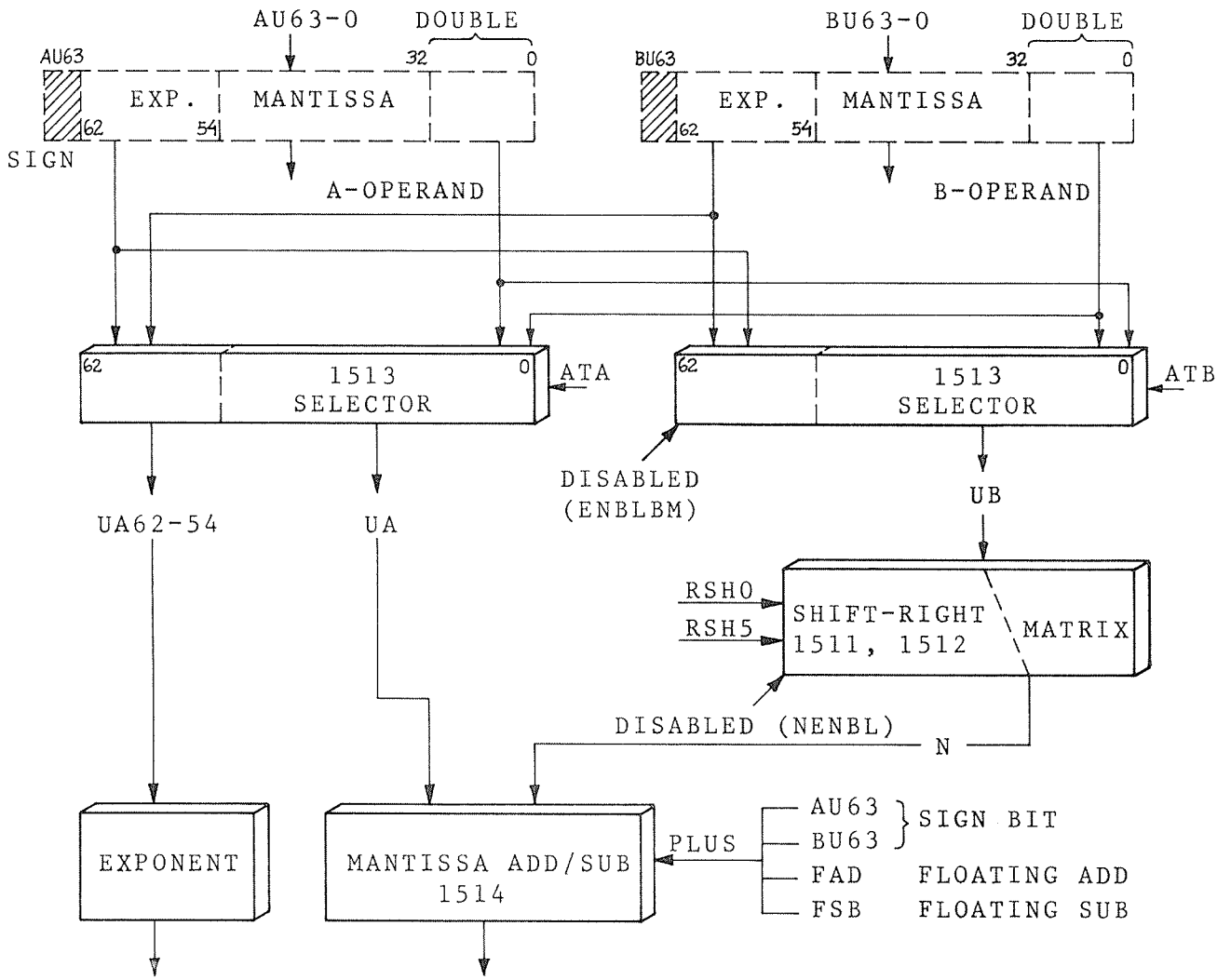


Figure IV.7.2

The output of the subtracter/inverter gives the number of shifts the mantissa of the smallest operand must be shifted to the right (RSH0-RSH5) to make the exponents equal.

The operand with the highest exponent is selected as the main exponent via the UA BUS.

MANTISSA ADD/SUB

Whether the two mantissas, the main and the shifted one, are to be subtracted or added depends on the sign of the two operands and the operation. (Floating ADD or Floating SUB).

The mantissa add/sub is performed with an ALU (74181) on the 1514 card.

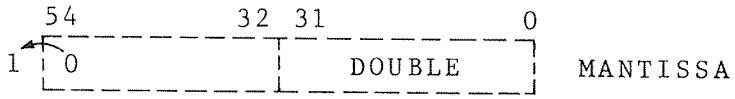
The shifted mantissa N is added/subtracted to/ from the main mantissa UA according to this table:

AU63	BU63	SFAD FLOATING ADD	SFSB FLOATING SUB
0	0	PLUS	MINUS
0	1	MINUS	PLUS
1	0	MINUS	PLUS
1	1	PLUS	MINUS

MANTISSA OVERFLOW

Mantissa overflow can occur in the cases involving a PLUS as given by the previous table. (MINUS resulting in overflow is a special case explained later). The mantissas of the two operands are added together. The carries are propagated from the low order end to the high order end.

After addition:



The resultant mantissa >1): the mantissa is not normalized. $1 <$ MANTISSA $< \frac{1}{2}$ = normalized mantissa.

NB! The most significant bit of the mantissa, UA54, (the bit is always 1 when normalized), is only used internally in the floating arithmetic. In memory or in the register block, bit 54 is the least significant bit in the exponent. Bit 54, the most significant in the mantissa is inserted on the 1513.4 card.

The least significant bit in the exponent is internally called (EUA54).

The most significant bit in the mantissa is masked away before presenting the result to CPU by simply disconnecting this bit. (The output bit 54 of the mantissa shift left matrix is not connected to the driver circuits).

In the case of mantissa overflow, the mantissa is shifted one location to the right (63 locations to the left with rotational shift) and the exponent is incremented by one, and we are finished.

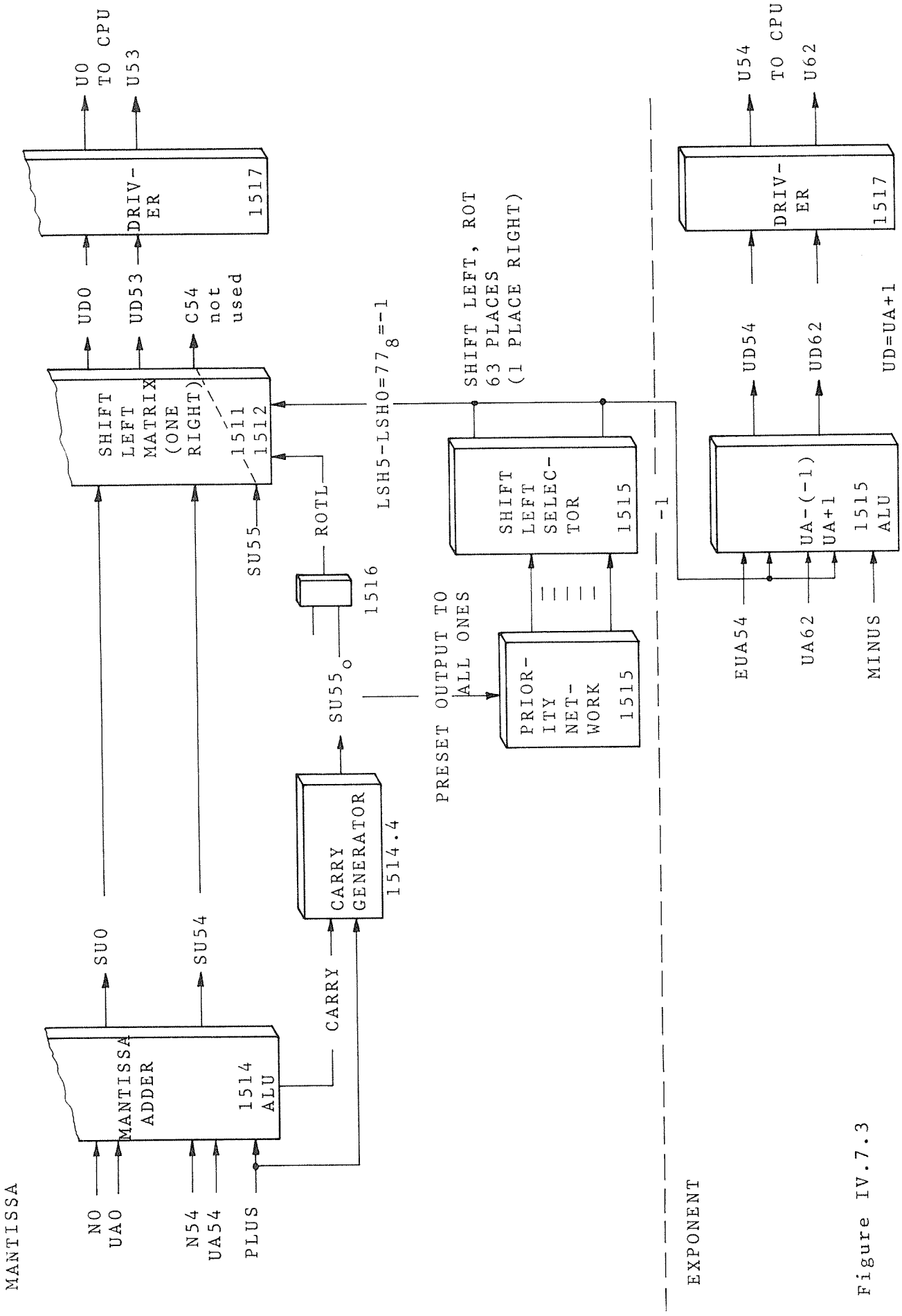

U0
TO CPU
U53
DRIV-
ER
1517
U54
TO CPU
U62
DRIV-
ER
1517
UD0
UD53
C54
not
used
UD54
UD62
UD=UA+1
SHIFT
LEFT
MATRIX
(ONE
RIGHT)
1511
1512
LSH5-LSH0=77₈=-1
SHIFT LEFT, ROT
63 PLACES
(1 PLACE RIGHT)
SHIFT
LEFT
SELEC-
TOR
1515
UA-(-1)
UA+1
1515
ALU
-1
SU55
ROTL
1516
SU55₀
PRESET OUTPUT TO
ALL ONES
PRIOR-
ITY
NET-
WORK
1515
EUA54
UA62
MINUS
CARRY
GENERATOR
1514.4
SU0
SU54
CARRY
MANTISSA
ADDER
1514
ALU
N0
UA0
N54
UA54
PLUS
MANTISSA
EXPONENT

Figure IV.7.3

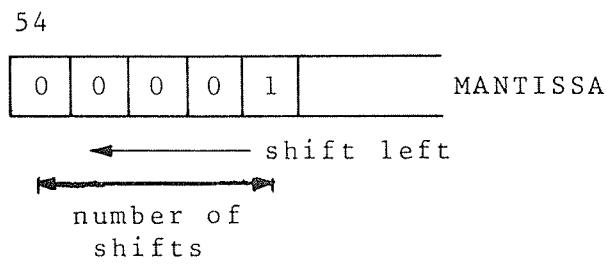
NORMALIZATION

If the most significant bit in the mantissa (BIT 54) is not a 1 after the subtraction (and there is no overflow), we must normalize the mantissa, that is get a 1 in the most significant bit of the mantissa.

We now shift the mantissa to the left until we get a 1 bit in this location.

For each left shift required the exponent is reduced by 1.

We are now finished and can present the result to the CPU.



NB! Normalization may only be necessary in the MINUS cases from previous table.

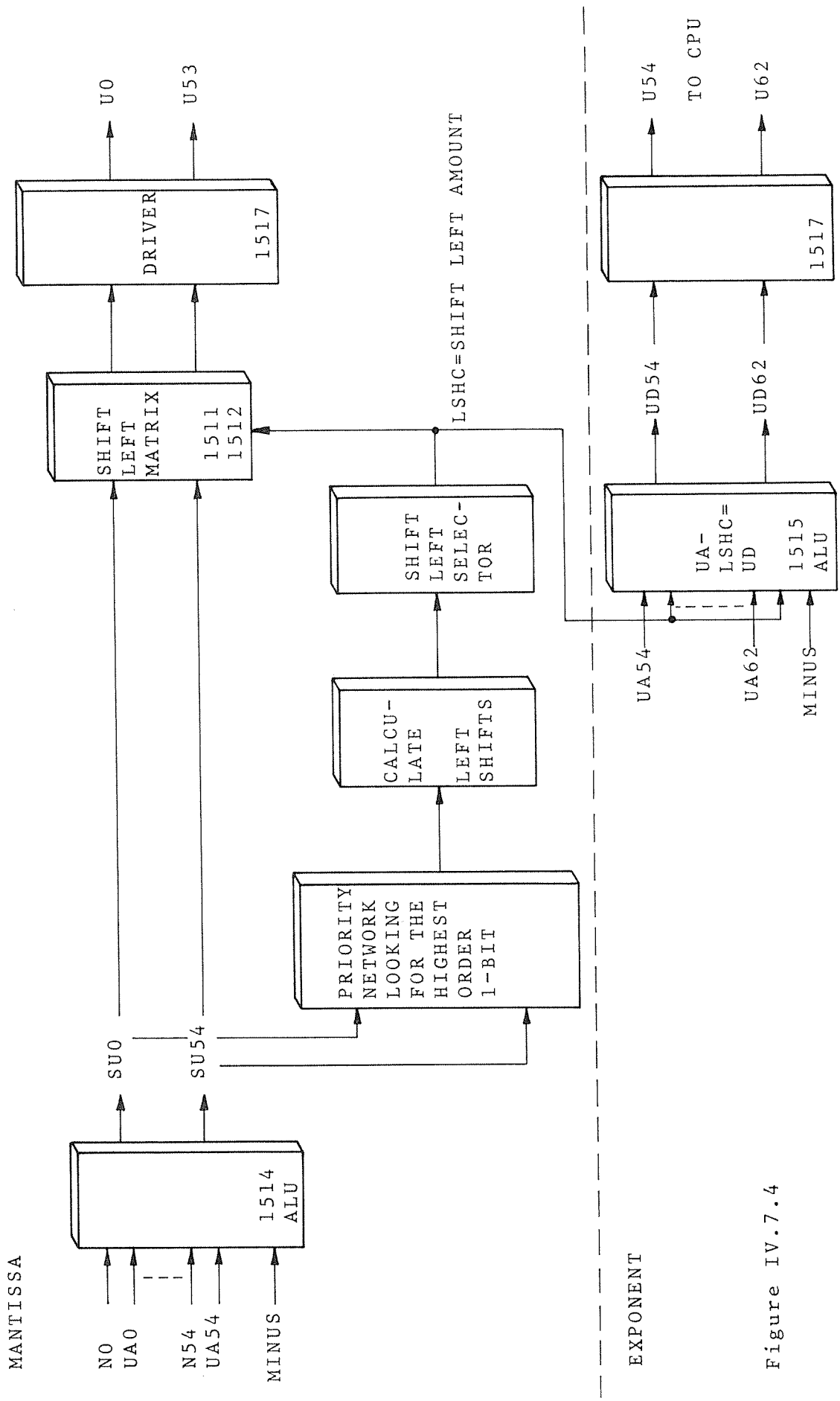

MANTISSA
EXPONENT
TO CPU
1514 ALU
1517 DRIVER
SHIFT LEFT MATRIX 1511 1512
1517
SHIFT LEFT SELEC-TOR
CALCU-LATE LEFT SHIFTS
PRIORITY NETWORK LOOKING FOR THE HIGHEST ORDER 1-BIT
UA-LSHC= UD 1515 ALU
N0 UA0
N54 UA54
MINUS
SU0
SU54
U0
U53
U54
U62
UD54
UD62
UA54
UA62
MINUS
LSHC=SHIFT LEFT AMOUNT

Figure IV.7.4

SPECIAL CASE

The exponents are equal. We are doing a MINUS operation in the mantissa ALU. The mantissa ALU does not generate a carry (C54).

This means that the result is negative): we have subtracted the greater mantissa from the smaller mantissa. Meaning that we have choosen the wrong mantissa to the A- and B-bus.

The mantissa ALU output is then inverted (2'complemented) (INV G 1516) in the second ALU on the 1514 card before it enters the shift matrix, and the output buffer.

ROUNDING (IN THE FLOATING ARITHMETIC UNIT)

If we are going to round a number, let us say the ith bit, then we examine the bit directly to the right, the (i-1)th bit.

The rules for performing this rounding are that if the (i-1) bit is 0, we leave the ith position unchanged, and we drop the other bits. If it is 1, we add 1 to the ith position and drop the other right-hand bits

The question about rounding arises when shifting the mantissa to the right, to equalize the exponents.

Two of the shifted out-bits are taken care of (EN1, EN2) (-1, -2).

One is enough for the addition and convert instruction, and two are sufficient for subtraction.

If the mantissa is already normalized after addition, the EN1 bit is added to the mantissa (1514.1).

If mantissa overflow occurs, the mantissa is shifted one location to the right.

Before the shifting, the least significant mantissa bit (the one to be shifted out) is incremented by one (ADD00 G 1516) giving a rounded answer if the least significant mantissa bit was equal 1.

In single precision (32 bits) operation, the 1-bit (ADD 32, G 1516) is added to bit 32 out of the mantissa adder.

For those with a bright mind more about rounding can be found in the NORD-50 REFERENCE MANUAL.

FLOATING - POINT ADDITION AND SUBTRACTION

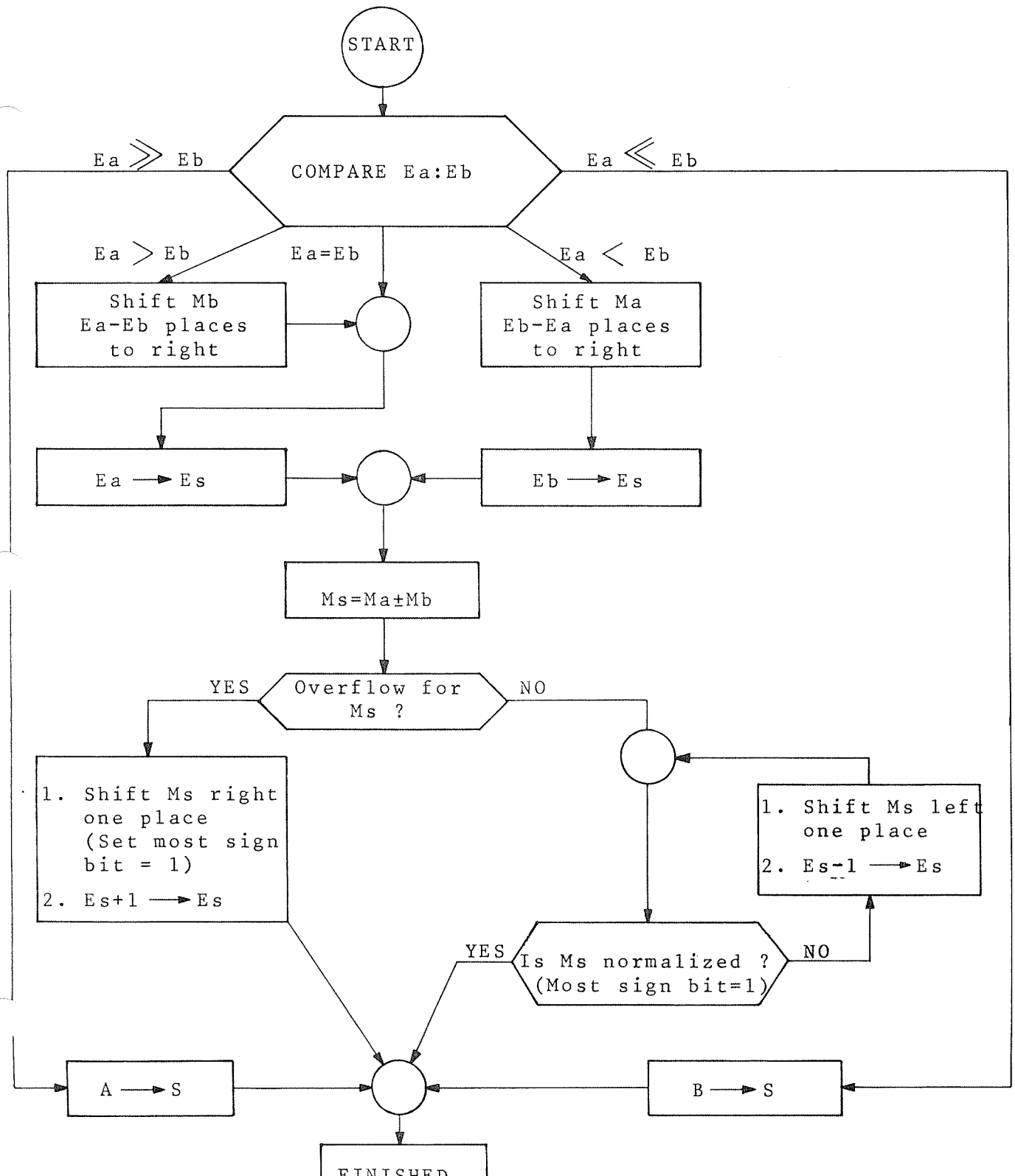
Given 2 floating-point numbers:

$A = Ma \cdot 2^{Ea}$ % Placed in CPU reg-block A

$B = Mb \cdot 2^{Eb}$ % Placed in CPU reg-block B or in memory.

We will now calculate the sum (difference)

$S = A \pm B$



IV.8 FLOATING REGISTER SKIP INSTRUCTIONS

ASGF(D): Add floating registers and skip if result ≥ 0 *

ASLD(D): Add floating registers and skip if result < 0 *

ASEF(D): Add floating registers and skip if result $= 0$

ASUD(D): Add floating registers and skip if result $\neq 0$

SGF(D): Subtract floating registers and skip if result ≥ 0 *

SLF(D): Subtract floating registers and skip if result < 0 *

SEF(D): Subtract floating registers and skip if result $= 0$

SUF(D): Subtract floating registers and skip if result $\neq 0$

(D): double precision floating register

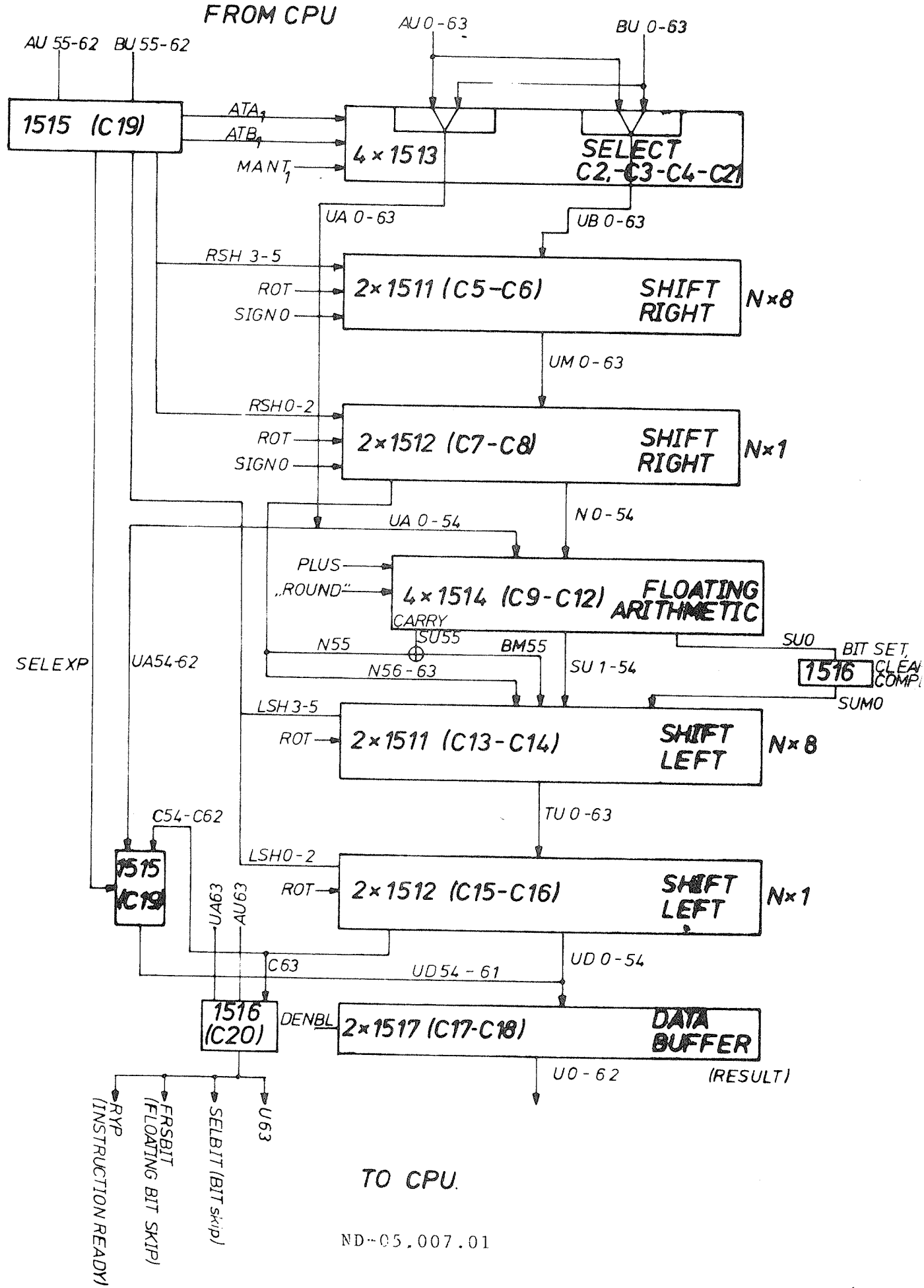
The instructions are executed as ordinary floating add/subtract instructions with the result written back to the CPU register block.

When executing the instructions marked * the SIGN BIT (C 63) is selected and gated back to the CPU via the FRSBIT line.

When executing the other instructions floating zero (FZERO) is selected and gated back to CPU via the FRSBIT line.

The FRSBIT signal is tested against the IR function code bits in the CPU (1506), whether the SKIP condition is fulfilled or not.

C - RACK OVERVIEW



SECTION V

EXTERNAL ARITHMETIC A-RACK

DETAILED CONTENTS
++++++++++++++

V.1 THE A-RACK

In the A-rack, the following instructions are executed:

MPY	Integer Multiply		32 bit operands
DIV	Integer Divide		32 bit operands
FMU	Floating Multiply	single precision	32 bit operands
FDV	Floating Divide	single precision	32 bit operands
FMUD	Floating Multiply	double precision	64 bit operands
FDVD	Floating Divide	double precision	64 bit operands

The instructions may either be memory reference instructions (one operand from memory and one from the register) or inter-register instructions (both operands from the register block).

The operands presented on the data lines from the CPU are latched in the A-rack before the arithmetic milling is started with repeated additions for mulitply and repeated subtractions for divide.

One card (out of 24) takes care of the floating exponent add/subtract while the mantissa is "milled" in the same unit as for integer.

An 8-1 line selector - the OUTPUT SELECTOR - selects the final result of the operation as input to a tri-state driver driving the result back to the CPU.

The result is selected through the SUM-selector and written back in the destination register via the SUM-bus.

Control Signals to the A-Rack

In addition to operand data given in figure V.1.1 the following control signals are decoded in the CPU out of the function code bits in the IR-register and sent to the A-rack:

	Gen. on card:	Means:	
SMPY	1508	Start Integer Multiply	Operands latched in CPU and exterm. arithm. milling can start
SDIV	1508	Start Integer Divide	
SFMU	1508	Start Floating Multiply	
SFDIV	1508	Start Floating Divide	
DPM	1505	Double precision floating point	

Signals back to CPU

	Gen.:	Means:
RYM	1526	Data Ready. Final result on tri-state lines to CPU
OF1	1526	Integer multiply overflow. Integer and floating divide by zero.
UF1	1526	Floating underflow.

Time used:

The table below indicates the time used by the A-rack to execute the instructions (text). The time is from the different START pulses to the RYM pulse.

Instruction:	text in µs:	
FMU	2.5	MEMORY REFERENCE OR INTER-REGISTER INSTRUCTIONS
FMUD	6.2	
FDV	2.5	
FDVD	6.2	
MPY *	4.0	
DIV *	1.0-4.0	

*Time used is data dependent.

In the following a brief description of binary multiply, divide and floating point multiply/divide will be given with some information on how this is implemented in NORD-50.

A - RACK (MUL - DIV UNIT) DATA FLOW.

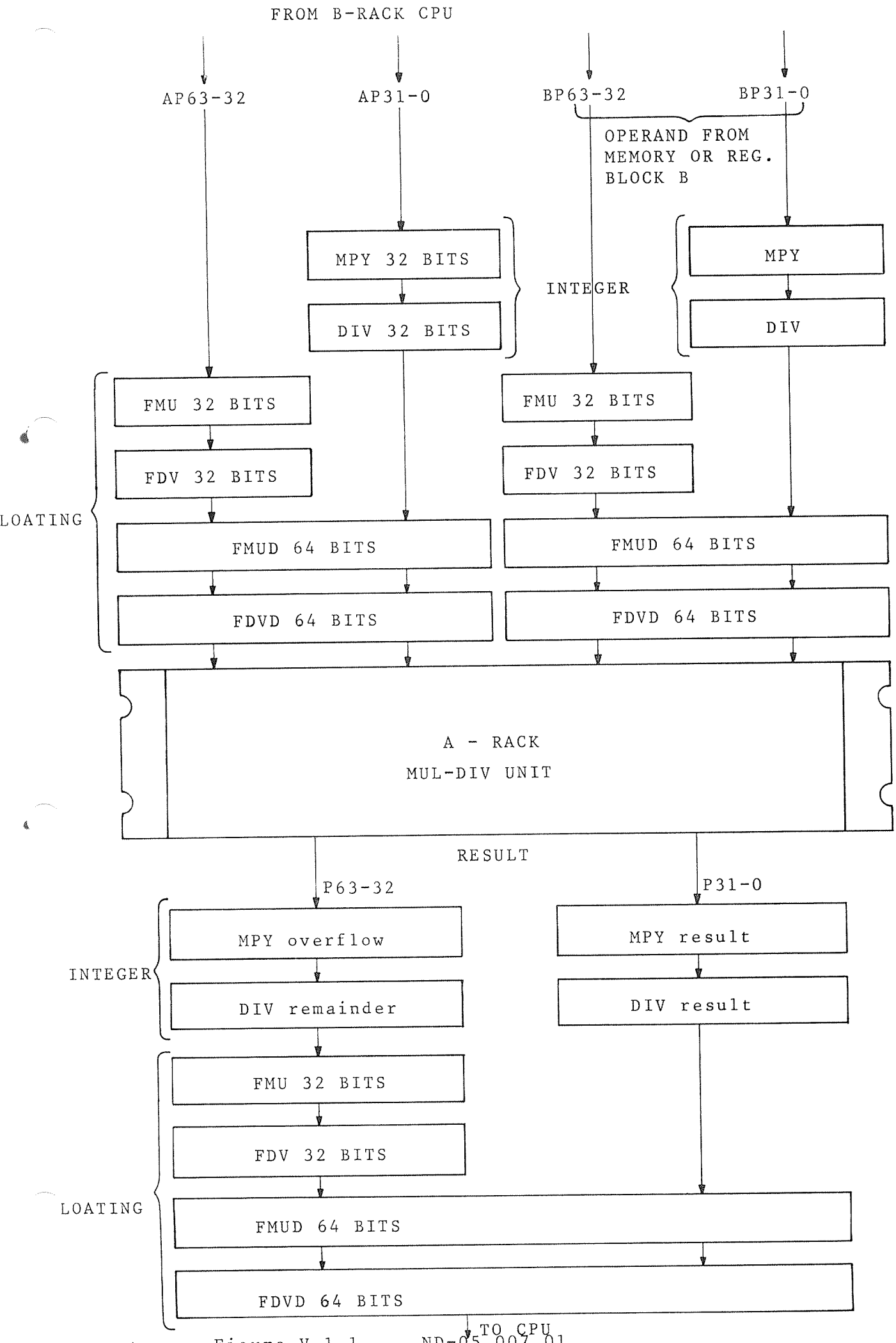


Figure V-1-1 ND-05.007.01

V.2 INTEGER MULTIPLY

In any number system multiplication consists of adding a number to itself as many times as specified by the multiplier. In actual practice binary multiplication reduces to copying the multiplicand whenever the multiplier digit is 1, and not copying it (or writing zeroes) whenever the multiplier digit is 0. As in decimal multiplication you must, of course, also shift one place to the left after obtaining each partial product and in the end add up all the partial products to obtain the answer.

The following example illustrates the simple procedure:

```
multiplicand (13)  multiplier (5)
      1 1 0 1    x     1 0 1
                       1 1 0 1   first partial product
                     0 0 0 0     second partial product
                   1 1 0 1       third partial product
                   -------------
product =  1 0 0 0 0 0 1 (65)
           =============
```

We could, of course, have omitted the second partial product of zeroes by simply shifting over to the left one additional place. Thus, binary multiplication can be reduced to the process of copying the multiplicand whenever the multiplier digit is 1, then shifting over one place to the left (multiply with 2) if the next multiplier digit is a 1, or shifting over one additional place for each 0 in the multiplier.

The partial products are then added to obtain the answer.

The method used in most computers is to add the partial products together as soon as they appear.

An example using this method follows:

multiplicand (15) multiplier (13)

```
      1 1 1 1   x   1 1 0 1
                    1 1 1 1   product 1
                  1 1 1 1     product 2
              1 0 0 1 0 1 1   sum
              1 1 1 1         product 3
            ---------------
            1 1 0 0 0 0 1 1   final sum
```

Note that the answer has twice as many bits as either number. If both numbers have the same number of bits , the product will always have twice as many digits, or be of double length.

INTEGER MULTIPLICATION

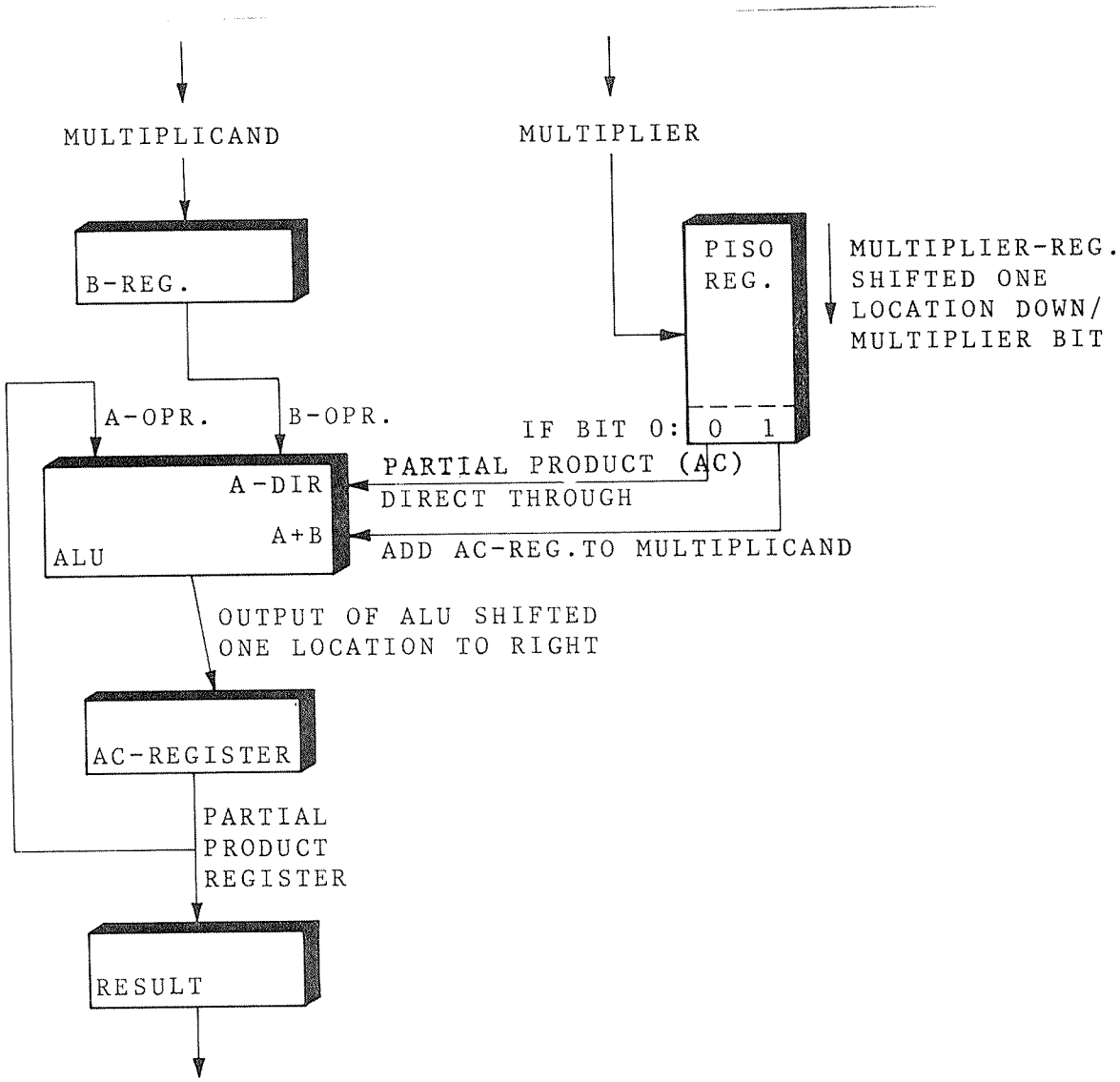


Figure V.2.1

Improvements in the Shifting Multiplier

In order to reduce the speed in the multiplication process a method of inspecting the multiplier bits in pairs is used in the NORD-50.

An example:

multiplicand multiplier

1 x 11 10 01 00

00 The multiplier is shifted 2 locations to the left. NO ADDITION

01 Shifted 2 locations MULTIPLICAND ADDED TO THE PARTIAL PRODUCT

10 Shifted 2 locations TWICE THE MULTIPLICAND ADDED TO THE PARTIAL PRODUCT

(Twice the multiplicand is the multiplicand shifted once to the left

11 Shifted 2 locations THREE TIMES THE MULTIPLICAND ADDED TO THE PARTIAL PRODUCT

sum : 11 10 01 00

If the MULTIPLIER is latched in the B-register and PARTIAL PRODUCT-REGISTER is called AC, we get these four combinations of the two least significant bits in the MULTIPLIER (PISO) register.

MULTIPLIER
BIT 1, 0

0 0	: 1/4(AC) → AC	% AC shifted 2 locations to the right NO ADD
0 1	: 1/4 [(AC)+(B)] → AC	
1 0	: 1/4 [(AC)+2(B)] → AC	
1 1	: 1/4 [(AC)+3(B)] → AC	

3(B): Three times the multiplicand is the output of an adder with one time the multiplicand as one input, and as the other input, twice the multiplicand (the multiplicand shifted once to the left).

N-50 INTEGER MULTIPLY

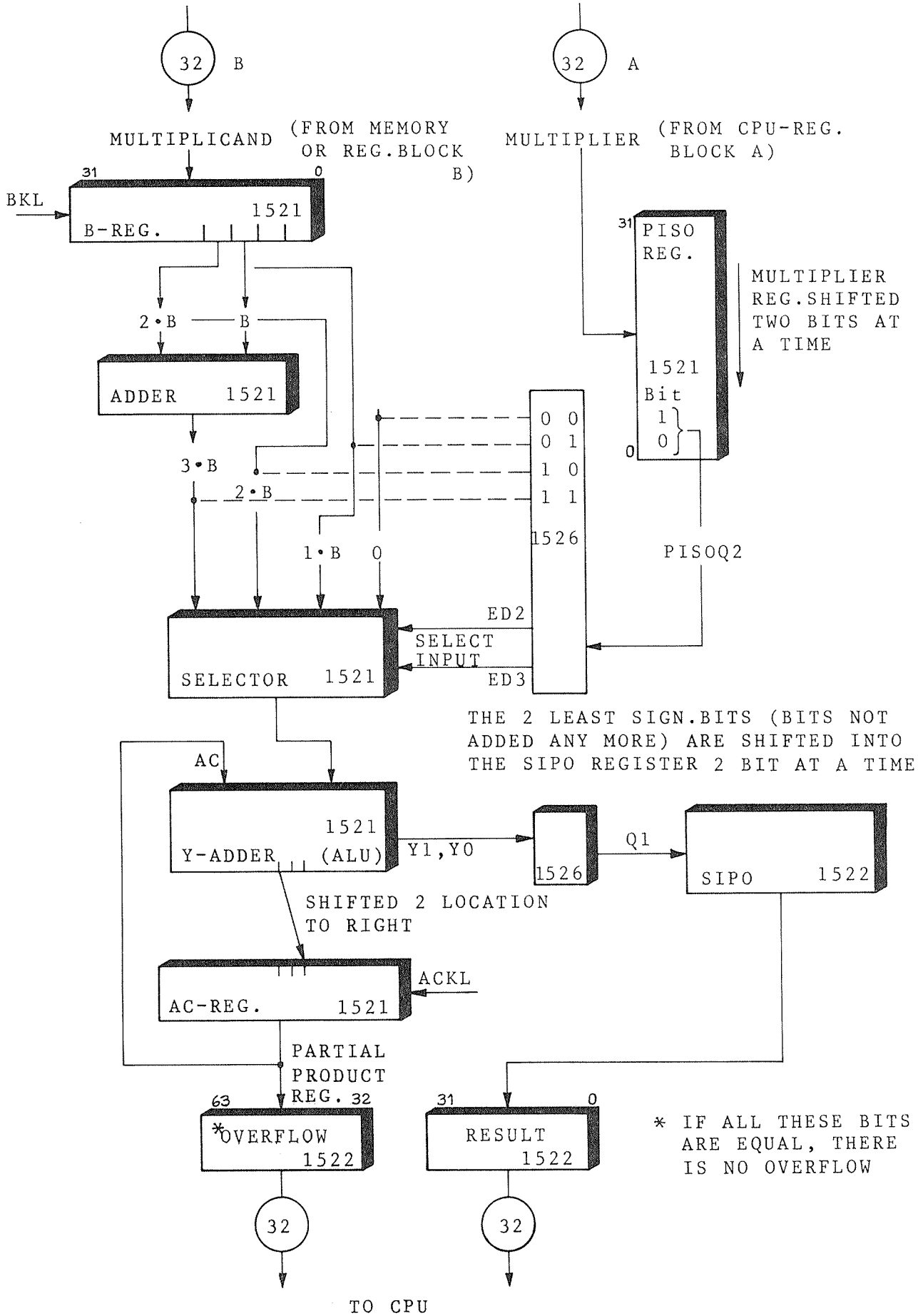


Figure V.2.2

NORD-50 INTEGER MULTIPLY
(32 bits x 32 bits = 64 bits)

Cycle:

C0 B-operand (Multiplicand) BP→F→M→IB→SB (via 1523, 1511, 1512, 1523) without change clocked into the RB-register.

IB0→SB23(1523.1)→RB23(1521.7)

IB31→SB54(1523.4)→RB54(1521.7)
→RB55(1521.7) Sign.extension.

A-operand(Multiplier) AP31-2 clocked into the PISO-register on 1521.
AP0,1 direct to 1526 for generating control signals ED2,3 for first operation in next cycle C1.

0→AC by clocking Y0.

C1 Loop, 32 clock pulses (16 iterations, each handling 2 bits).
The A-operand is shifted out of the SIPO (PISOQ2) two bits at a time and choose RB*0, RB*1, RB*2 or RB*3 (ED2,3) to be added to the AC-register. The output of the Y-adder is shifted 2 bits to the right and clocked into the AC-register.

The 2 least significant bits of the Y-adder Y0, Y1 (1521.1) are latched on the 1526 card (AMN1, AMN2) and shifted into the SIPO register (1522.1), two bits at a time (Q1).
The SIPO is an extension to the AC-register.

C2 Correction:

If A-operand negative (A31=1):

Then the R•B was added one time too much, and to get the sign correct, we must subtract the R•B.

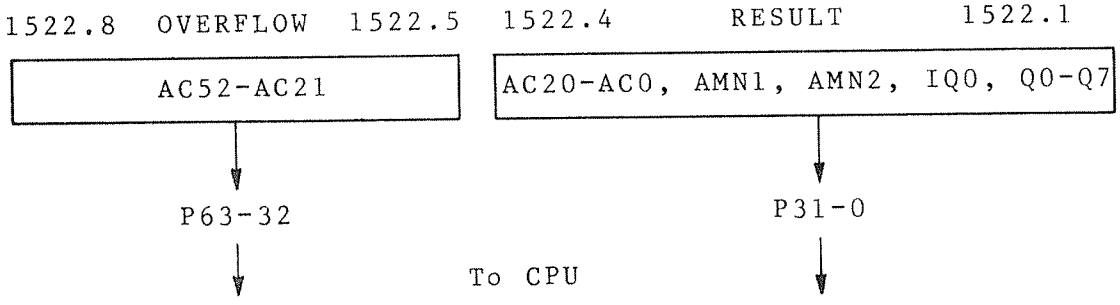
1/4(AC-1•RB)→AC ED2=ED3=0
% YADR=SUB

C2
cont. If A-operand positive:

1/4 AC→AC

NB! Only Y-adder in use.

C3 Present result on P-lines to CPU.



MPY OVERFLOW:

Multiplying 2 integer 32 bits come up with a double precision number, 64 bits.

An overflow condition exists unless all the 32 most significant bits are equal, and equal to the most significant product bit.

The AC52-21 bits are checked for all ones or all zeroes on the OUTPUT SELECTOR cards 1522,5-1522.8. If AC20 and AC52-21 are not equal an overflow signal (OFL) G 1524 is sent to CPU.

In Test System:

Overflow-register is shown as LEAST-RESULT:

V.3 BINARY DIVISION, INTEGER

Division in any system is the inverse of multiplication.

It is the process of determining how many times one number (the divisor) can be subtracted from another number (the dividend), while still leaving a positive remainder.

The number of times this can be done is the result, or the quotient.

For example, dividing binary 110111 (55) by 101 (5). Answer 1011 (11) remainder 0.

```
 DIVIDEND  DIVISOR   QUOTIENT
 110111  : 101     = 1011
-101
  11
   0
  111
 -101
   101
  -101
     0   REMAINDER.
```

In binary division we are either able to subtract the divisor (in which case the quotient digit is 1), or we are not able to subtract the divisor (giving a quotient digit of 0).

A way of implementing this in hardware would be to latch the divisor in a register (B) and the dividend in another register, the accumulator or the AC-register. Refer to figure V.3.1

After latching the divisor and the dividend, we now subtract the B-register from the AC-register.

If the subtraction is succesful (B>AC), that is it leaves a positive remainder, a carry is generated by the subtractor.

The carry is fed to a quotient-generator, generating a 1 bit that is fed to a serial in-parallell out register (SIPO), the quotient-register.

The carry signal also selects the input to the AC-register:

If the dividend is greater than the divisor, the AC-B value is fed into the AC-register shifted one location to the left.

If, however, the larger number (B) is subtracted from the smaller (AC), no carry is generated (the divisor is too large).

A zero is generated in the quotient-generator and fed into the SIPO shifted one location to the left. The AC-register is fed back to itself by the carry, and shifted to the left in preparation for another trial.

At the end the AC-register holds the remainder and the result (the quotient) is in the SIPO-register.

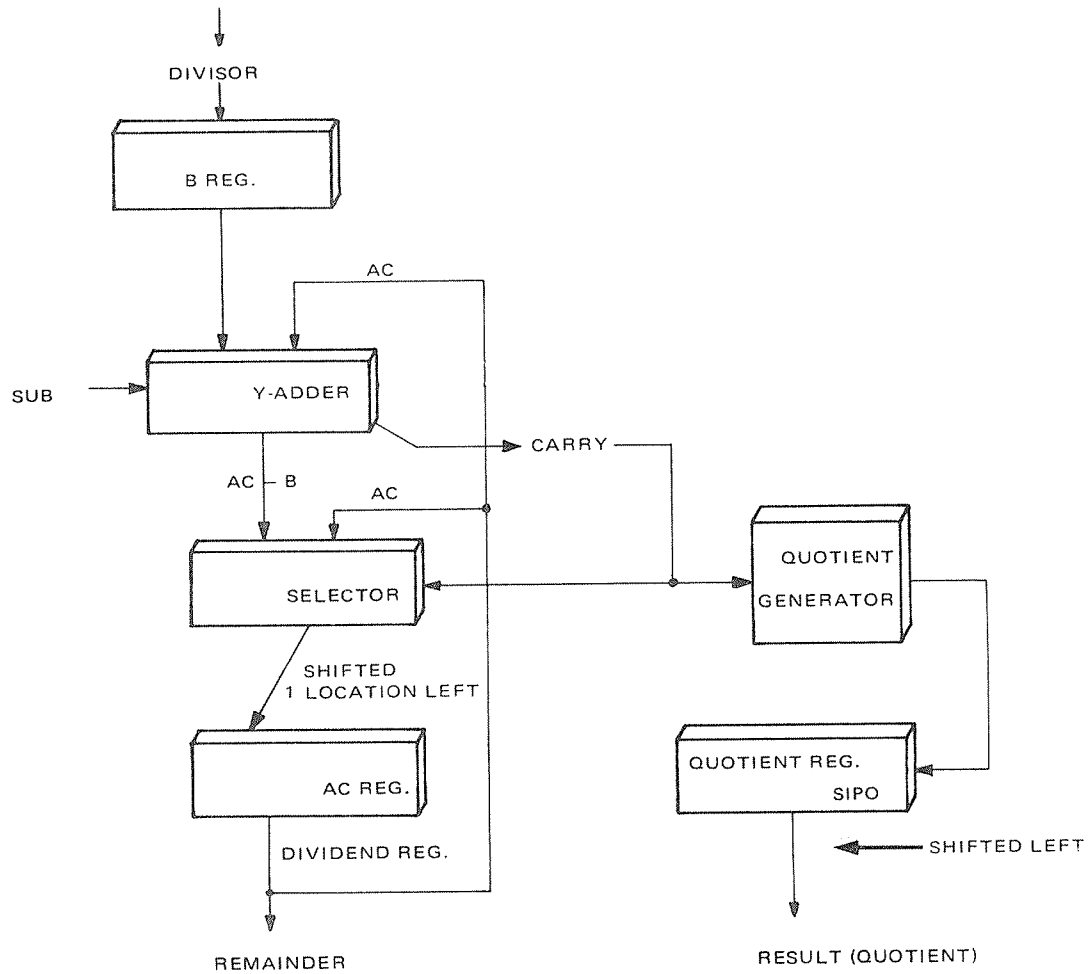


Figure V.3.1 *Division Hardware*

DIVISION IN NORD-50.

INITIALIZING THE DIVISION.

THE DIVISOR:

To reduce the complexity of determing what multiple of the divisor can be subtracted from the dividend, the divisor is aligned to a normalized-position.

The most significant "1" bit in the B-operand (the divisor) is shifted left to the most significant B-register position (RB54).

In hardware this is done by using a priority encoder to look for the most significant "1" bit.

The required shift-length for the divisor is found by subtracting the output of the priority encoder "1" bit position from the most significant bit-position. The shift-length (SH4-0) is latched for later use.

The divisor is shifted left in the shift-left matrix cards 1511 and 1512, and clocked into the divisor-register (RB). The B-register is stable during the rest of the operation.

Before the subtractions take place, both operands are made positive. The B-operand (the divisor) is inverted by the 1523 card (if it is negative). This takes place before the shifting.

THE DIVIDEND:

We have to shift the dividend one place right if the divisor was shifted an odd number of places left, because as we shall see later we are handling the bits in pairs. In hardware we do this by selecting the A_{n+1} bit as input to the dividend register (AC).

The dividend is inverted if it is negative. This is done in the X-adder by subtracting the A-operand (dividend) from the initial zero content of the AC-register.

If it is positive the dividend is added to the zero content of the AC-register (directly through the adder). The output of the X-adder is shifted 2 places left and clocked into the AC-register.

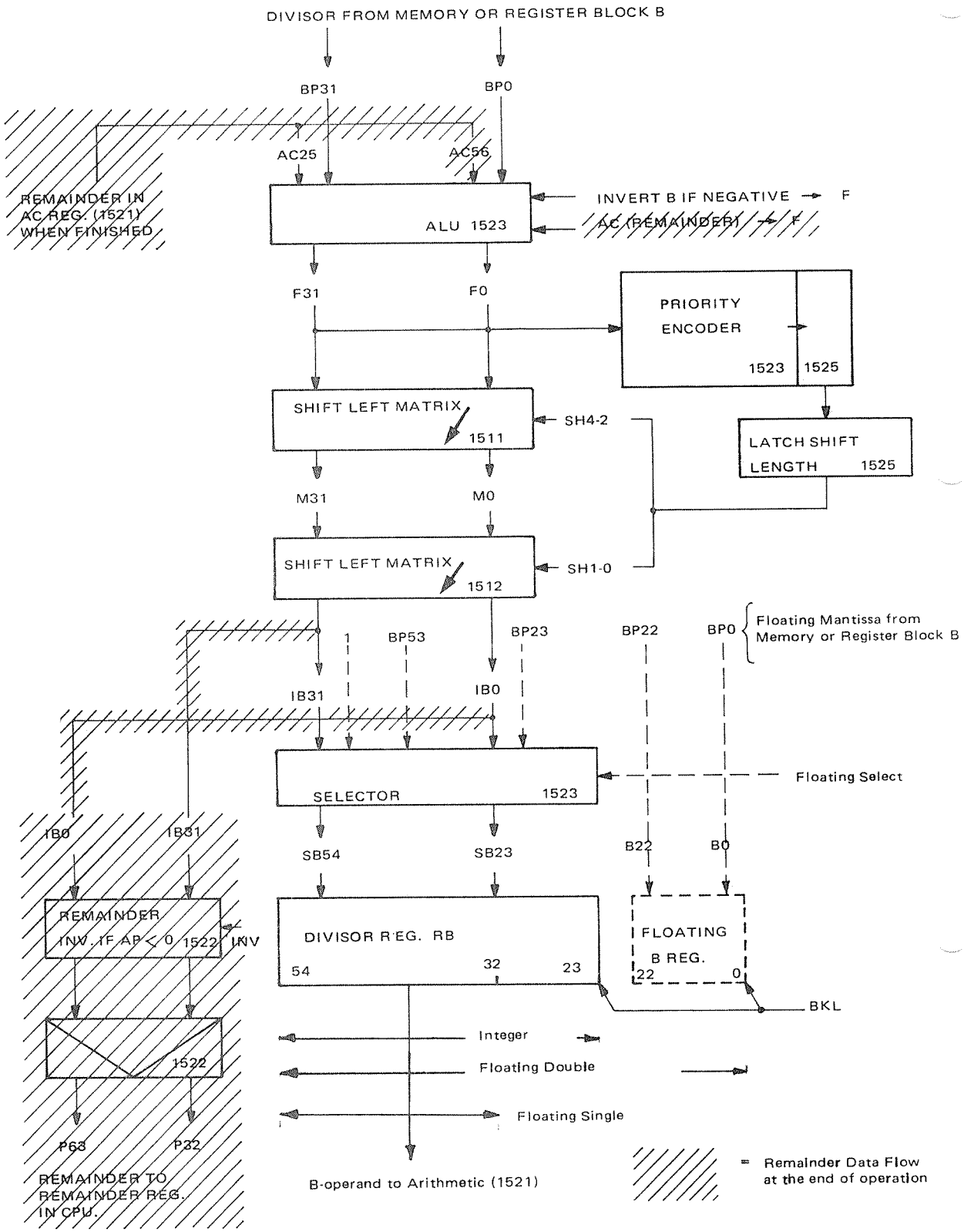


Figure V.3.2 *Divisor (B-Operand) and Remainder Data Flow*

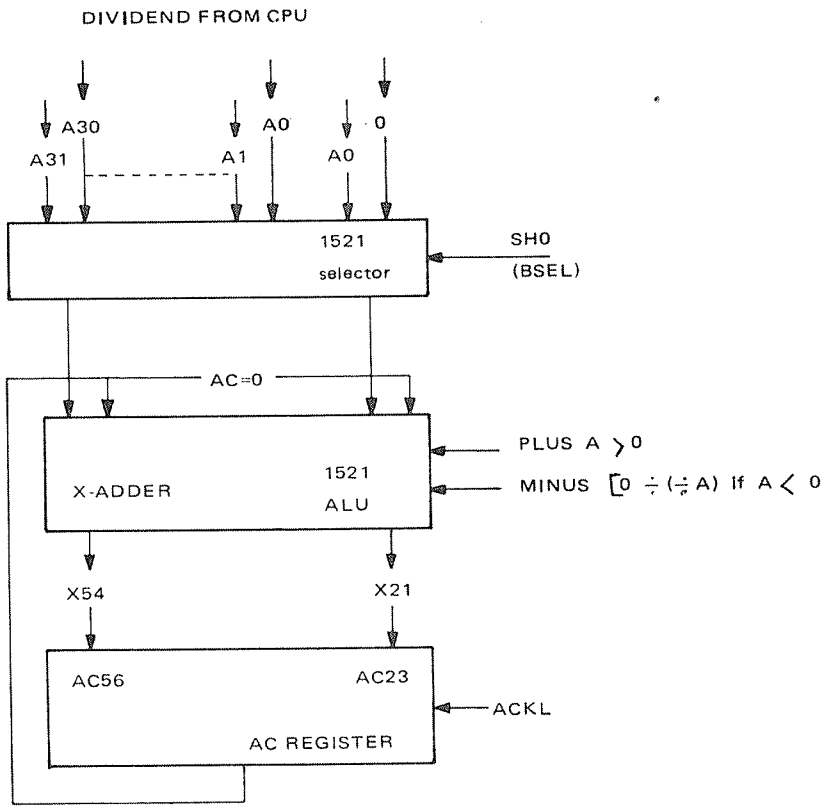


Figure V.3.3 *A-Operand (Dividend) Initial Data Flow*

NUMBER OF SUBTRACTIONS

The number of subtractions depends on the number of shifts of the divisor. The bits are handled in pairs, and the number of subtractions required for a division with two 32 bit operands is:

16 - (position of the most sign "1" bit) /2

or

16 - (SH4-0) /2

This means that dividing goes faster with a large divisor than with a small one.

If the divisor is a small number, 1 for example then the maximum time is required to do the division. In this case, the "1" bit has to be shifted 31 places left to the most significant bit position This is the equivalent of doing a division using pencil and paper:

When dividing by 1, the quotient will be equal to the divident and there will be no remainder,

when dividing by a large number, the quotient will be small and generally will leave a remainder.

As we have mentioned before, there must be an even number of bit-positions between the decimal-points of the divisor and the dividend.

DIVISION LOOP

When doing decimal division with pencil and paper, we first make an estimate or an intelligent guess at what multiple of the divisor can be subtracted from the dividend still leaving a positive remainder.

If our quess turns out to be wrong, we reduce the multiple by one and try again. The final multiple, with a value from 0-9, will then give us the quotient.

The NORD-50 MUL/DIV unit carries out division in the manner described above. Refer to figure

To increase the speed of this operation, bits are examined in pairs. This means that the dividend can be subtracted either 3, 2, 1 or 0 times from the divisor.

If the subtraction is successful, the quotient will be either 3, 2, 1 or 0.

If the subtraction is not successful, we reduce the multiple of the divisor by one, and reduce the estimated quotient by one.

We estimate the quotient by looking at the 2 most significant B-register bits (B54, B53) and the 3 most significant AC-register bits AC 56-54.

The estimated quotient bits ED2 and ED3 (G 1526) are used to select various multiples ofthe divisor register as input to the Y-adder and the X-adder. The X-adder input is always one multiple less than the Y-adder input.

In the X and Y-adder these inputs are subtracted from the AC-register (the partial remainder-register).

If the subtraction in the Y-adder is successful (YC 57 = 1,) the output of the Y-adder is selected as input to the AC-register shifted 2 places left.

If the subtraction is not a success, (YC 57 = 0) the estimate was wrong and the X-adder output is selected to the AC-register.

The quotient is generated on the 1526 card and if the estimate is ok, we use the estimate bits ED 2 and ED 3 as the quotient.

If the subtraction in the Y-adder is not successful, we reduce the estimated quotient by one.

The quotient bits are fed into the SIPO-register 2 bits at a time.

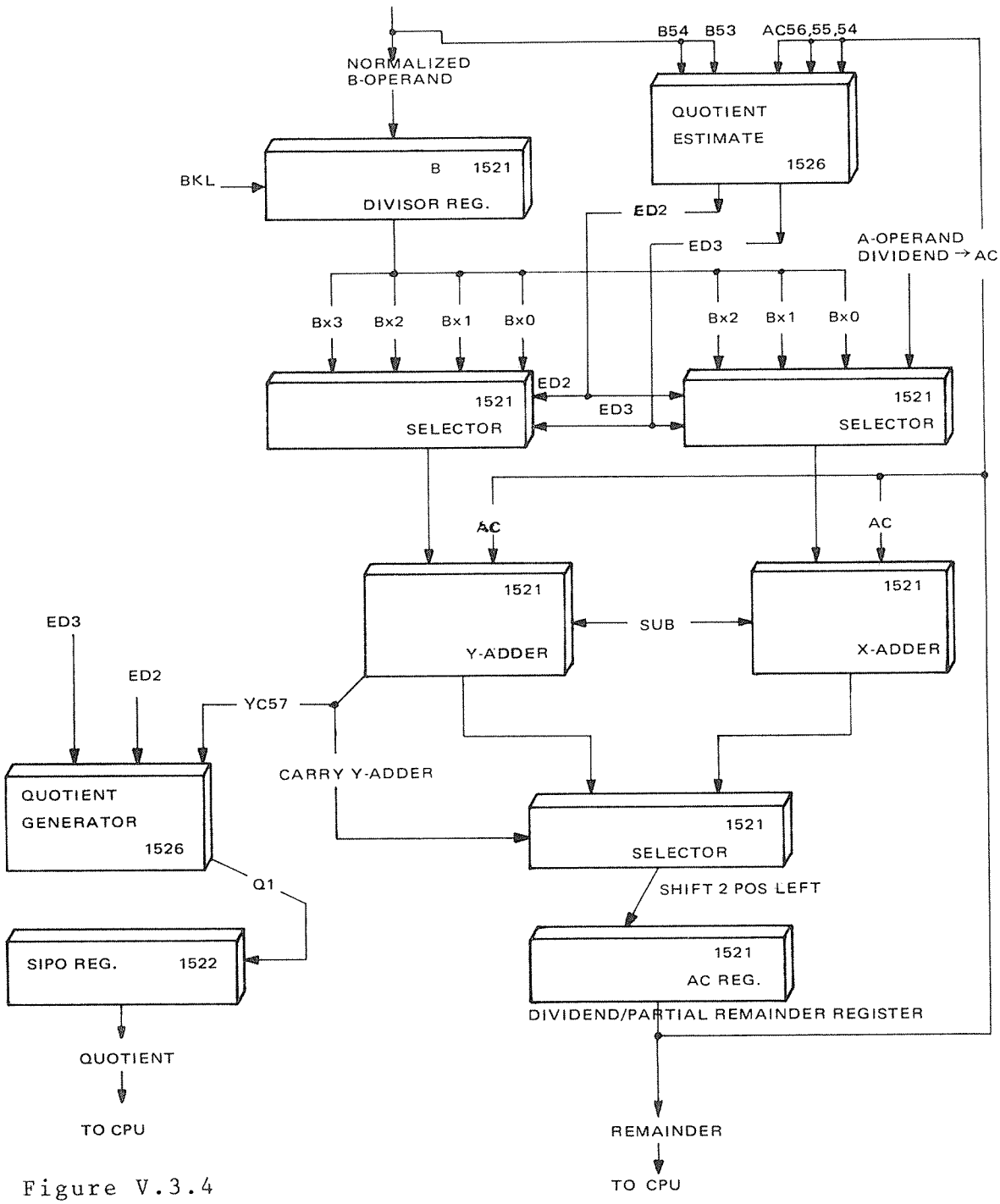


Figure V.3.4
Quotient and Remainder Generation

DIVISION

PARTIAL REMAINDER, QUOTIENT TABLE.

DIVISOR MAGNITUDE: SB54	SB53	PARTIAL REMAINDER MAGNITUDE (3 MOST SIGN. AC BITS)							
		56,55,54 0 0 0	56,55,54 0 0 1	56,55,54 0 1 0	56,55,54 0 1 1	56,55,54 1 0 0	56,55,54 1 0 1	56,55,54 1 1 0	56,55,54 1 1 1
1	0								
Estimate: Output Y-adder, Output X-adder *		ED2 = 0, ED3 = 0 AC - RB AC		ED2=1, ED3=0 AC - 2·RB AC - RB	ED2 = 0, ED3 = 1 AC - 3 · RB AC - 2 · RB				
		Quotient first in to SIPO	Quotient second in to SIPO						
Quotients if YC57 = 0		0	0	0	1				
Quotients if YC57 = 1		0	1	1	0				
1	1								
Estimate: Ouput Y-adder, Ouput X-adder *		ED2 = 0, ED3 = 0 AC - RB AC		ED2 = 1, ED3 = 0 AC - 2 · RB AC - RB	ED2 = 0, ED3 = 1 AC - 3 · RB AC - 2 · RB				
Quotients if YC57 = 0		0	0	0	1				
Quotients if YC57 = 1		0	1	1	0				

* If Y-adder carry (YC57) equals one (successfull subtraction) then Y → AC shifted 2 places left.
If Y-adder carry (YC57) equals zero (unsuccessfull subtraction) then X → AC shifted 2 places left.

SHIFTING THE REMAINDER BACK

Because the divisor is shifted left at the start of the operation, the remainder on completion of the operation will be placed too far to the left in the AC-register.

The remainder must now be shifted to the right the number of places the divisor was shifted to the left.

Since there is no right shift register in the MUL/DIV unit we must shift the remainder left. Refer to figure V.3.2 to see how this is done.

The remainder in the AC-register is reversed and placed in the shift-left register and shifted. It is taken out and reversed again and placed in the output selector for presentation to the CPU.

SIGNS

In integer divison we operate with positive operands and we have to remember the sign, and correct the sign at the end of the operation.

If dividend and divisor (A and B operand) have different signs, we have to invert the quotient.

In hardware this is done as follows:

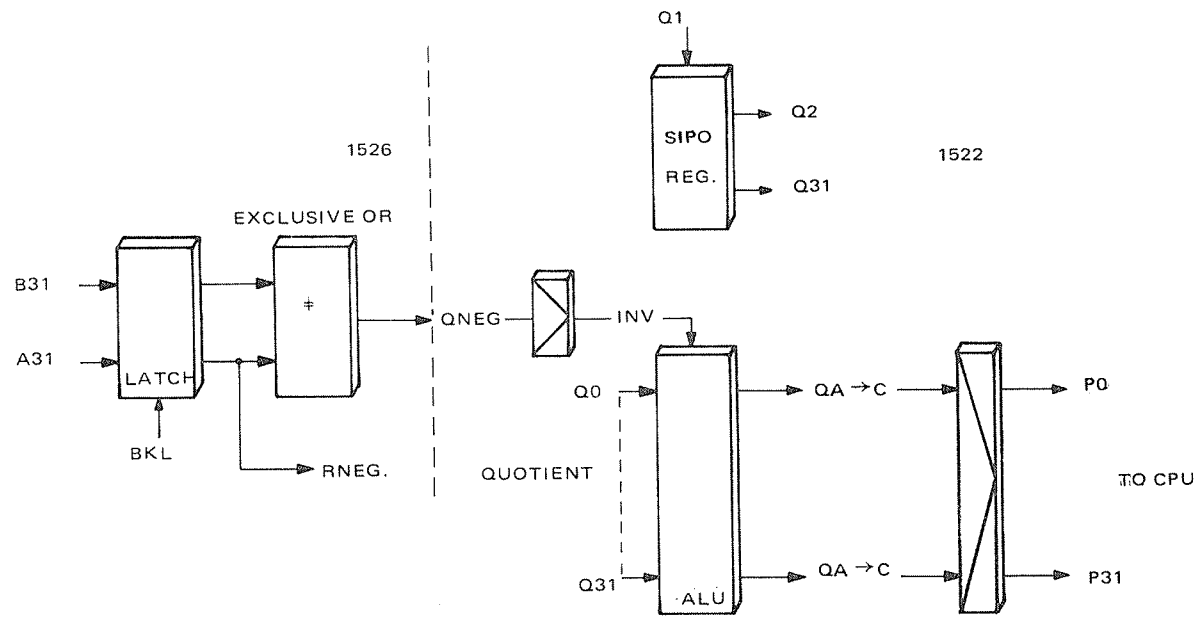


Figure V.3.5

If the A-operand (the dividend) was negative, then the remainder (that is in fact what is left of the dividend) should also be negative.

The IB-signals from the shift-register are inverted on the output selector cards 1522.

DIVIDE INTEGER NORD-50

C0 - B-operand (the divisor) BP is, if negative, inverted on the B-input 1523 card → F → M normalizing shift (shifting the most significant "1" bit left into the most significant position in the B-register RB 54). The shiftlength (SH4-0) is saved so that the remainder can be shifted back in C3. After shifting → SB, and clocked into the B-register (1521).

A-operand (the dividend) AP, is shifted one place if SHO = 1 (odd number of shifts for the divisor) in the dividend selector (1521).

If the A-operand is negative it is inverted in the X-adder 1521 and clocked into the AC-register as X•4.

NB! AC is now < 4•B-register.

C1 - Loop; the number of times the loop is performed is determined by the shift-length for the divisor (B-operand) in C0.
For each loop two alternative multiples of RB (divisor) are subtracted in the X and Y-adder. (The alternative in the X-adder is one multiple less then the Y-adder.

If Y is positive Y x 4 → AC,

otherwise X x 4 → AC.

Two bits of the quotient are entered into the SIPO-register

C2 - The Remainder in AC is shifted back into its correct position (shiftlength = SH4-0) AC → F → M shifted → IB, and inverted (1522) if AP<0.

The Quotient in the SIPO-register 1522 is inverted if AP • BP<0.

C3 - Present result → C → P → CPU

NB! In the TEST-SYSTEM, the remainder register is shown in LEAST RESULT.

V.4 FLOATING POINT MULTIPLY/DIVIDE

When multiplying two normalized floating numbers

$$(Ma \cdot 2^{Ea} \quad \text{and} \quad Mb \cdot 2^{Eb})$$

we multiply the mantissas and add the exponents

$$P = Ma \cdot Mb \cdot 2^{Ea + Eb}$$

when dividing two normalized floating numbers we divide the mantissas and subtracts the exponents.

$$Q = Ma/Mb \cdot 2^{Ea-Eb}$$

We notice that the exponents are added/subtracted and that they need not to be aligned as in floating addition.

As the simplified block diagram figure V.4.1 and V.4.2 indicates, the floating mul/div operations can be divided into two parts. One part for adding/subtracting the exponents and one for multiplying/dividing the mantissas.

FLOATING MULTIPLY
SIMPLIFIED BLOCK DIAGRAM

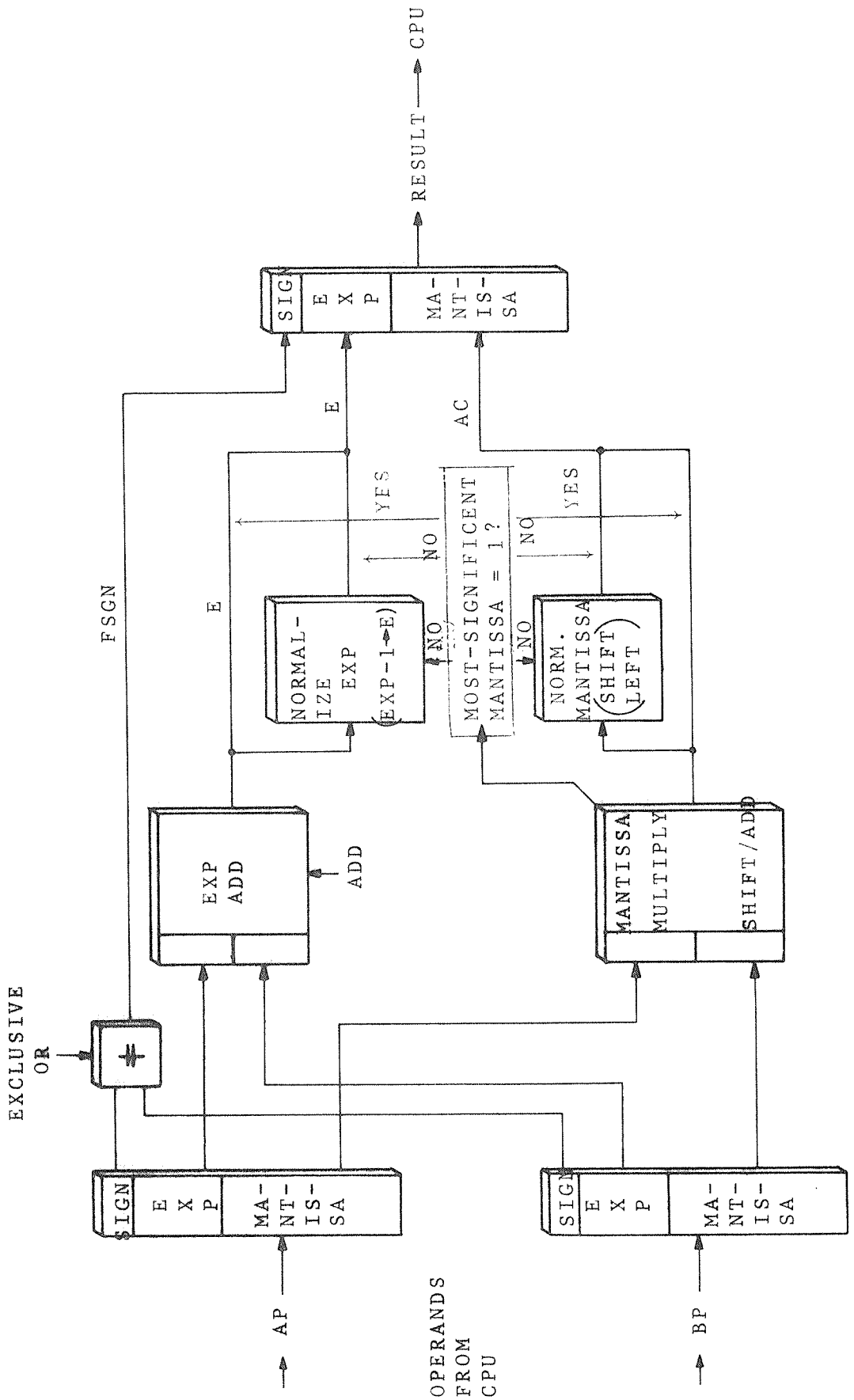
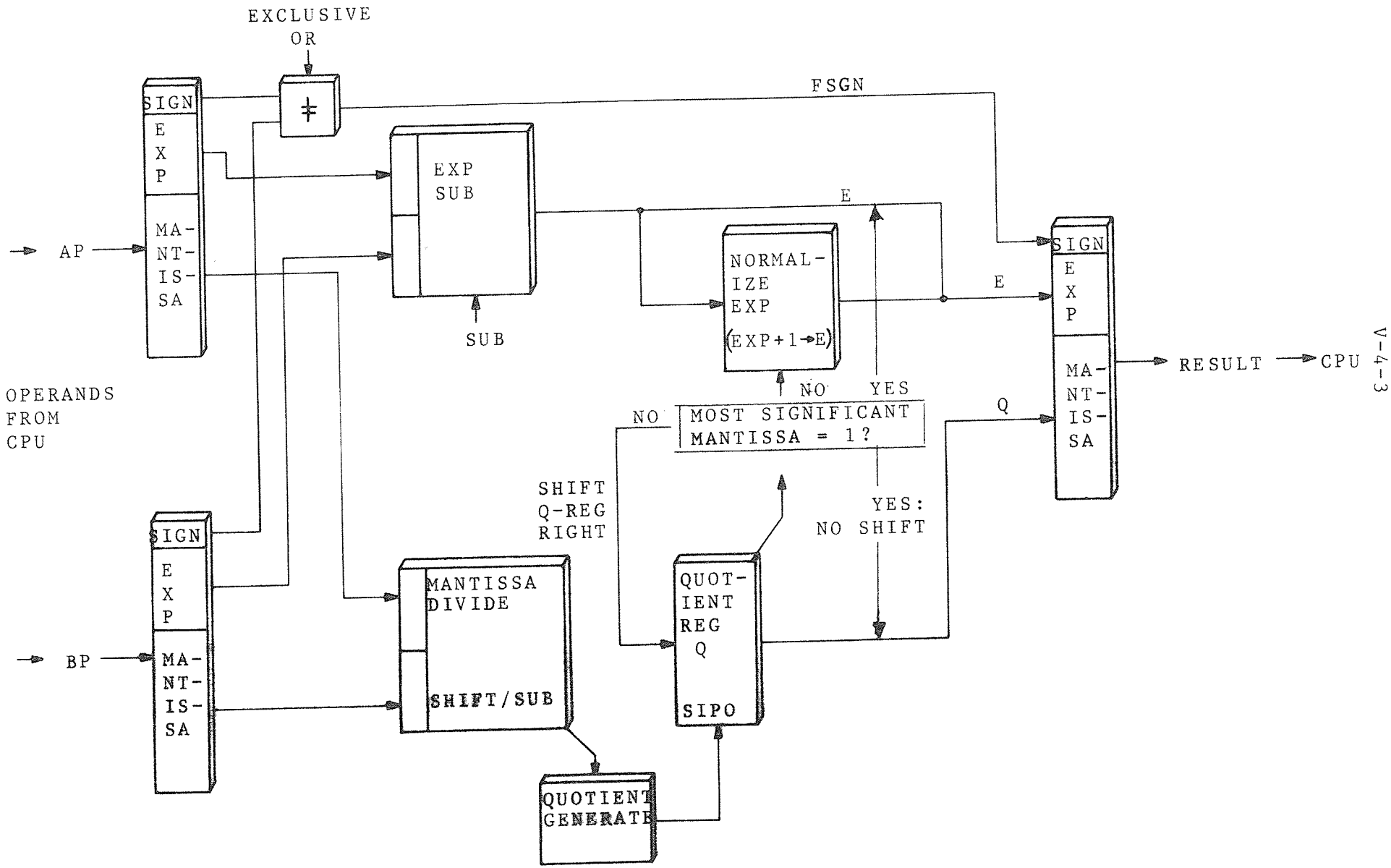


Figure V.4.1

FLOATING DIVIDE

SIMPLIFIED BOLCK DIAGRAM



Tegn 13.4.77.PK/bw.

EXPONENT ARITHMETIC

The exponent treatment for multiply and divide is taken care of on the Exponent Arithmetic card 1524. See block diagram figure V.4.3

Both operands are entered into an operand register by the BKL-signal. The exponents are examined to see if they are zero. If all bits are zero, it is a zero number.

If either of the exponents is zero in floating multiply, the result will be zero. Zero is selected as result on the output selector 1522 (S0=0, S1=1, S2=1 via CZRO 1524)

If the B-operand is zero in floating divide, this means division by zero, and an overflow signal is sent to the CPU.

Bit 62, the most significant bit of the exponent, the bias or offset bit is inverted to take away the bias and get a standard number.

Bit 63, the sign bits are exclusive-ored to get the sign of the floating point result.

In the Exponent Add/subtract ALU the exponents are added if floating multiply and subtracted if divide.

MULTIPLICATION

A normalized mantissa is between 1/2 and 1 and therefore always contains a 1 in the most significant mantissa bit.

If the result of the mantissa multiplication gives a number between 1/4 and 1/2, leaving a 0 in the most significant mantissa bit (AC53 = 0), the mantissa have to be shifted to the left one location (multiply by 2) and the exponent reduced by one.

In hardware a CORR signal is generated selecting an exponent reduced by one to the output selector to CPU.

DIVISION

If the result of the mantissa division gives a qoutient between 1/2 and 2 (Q53 = 0), the mantissa has to be shifted one place down (to the right) and 1 added to the exponent. In hardware the CORR signal selects an exponent incremented by one to the output selector to CPU.

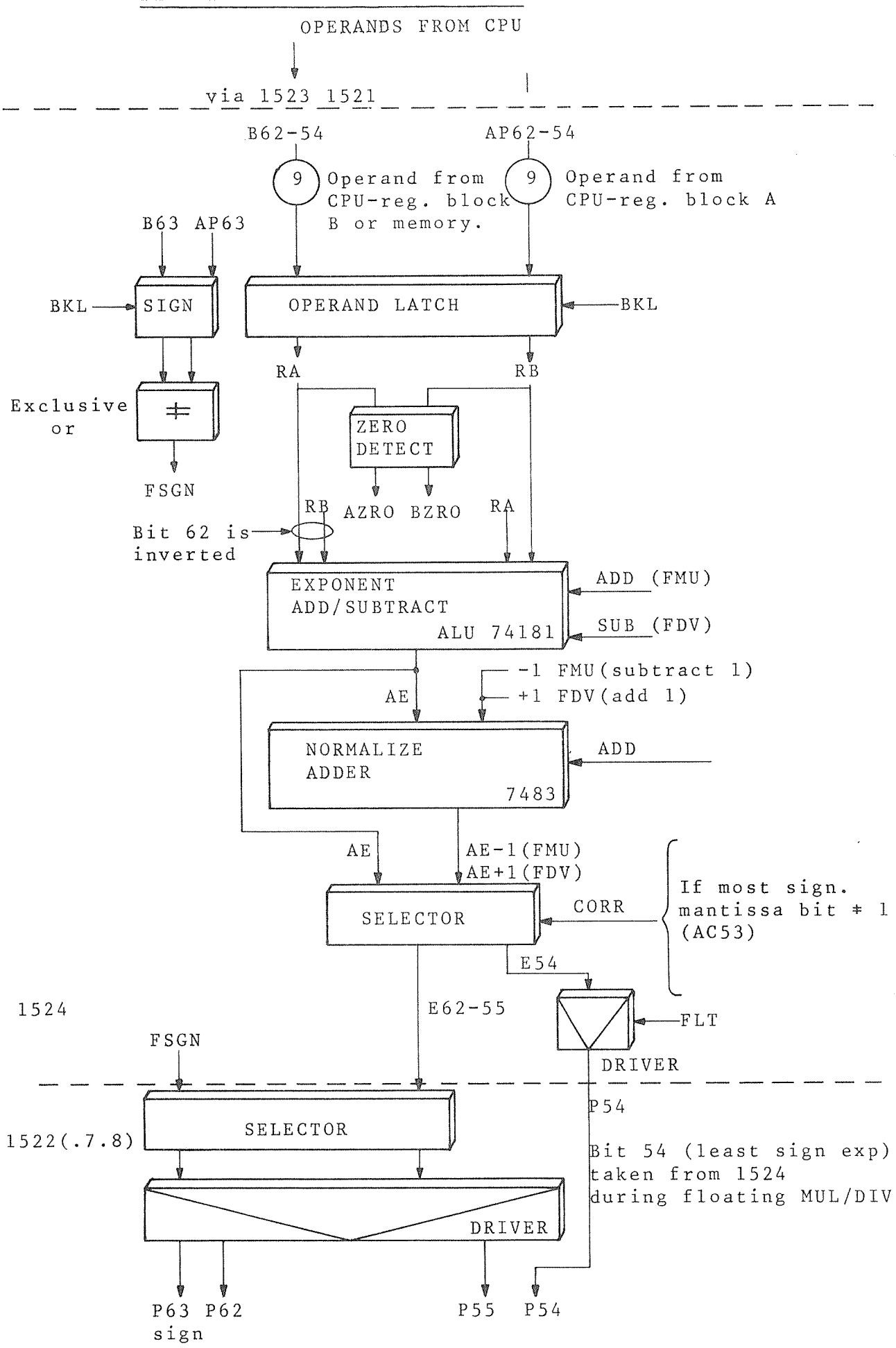


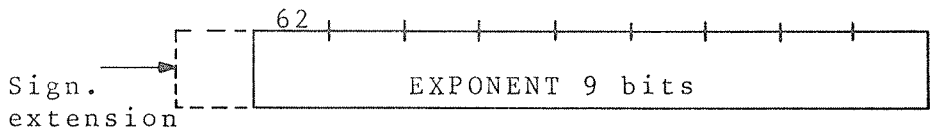
Figure V.4.3 Result Exponent to CPU.

ND-05.007.01

30/3/77/PK/bw.

EXPONENT OVERFLOW/UNDERFLOW

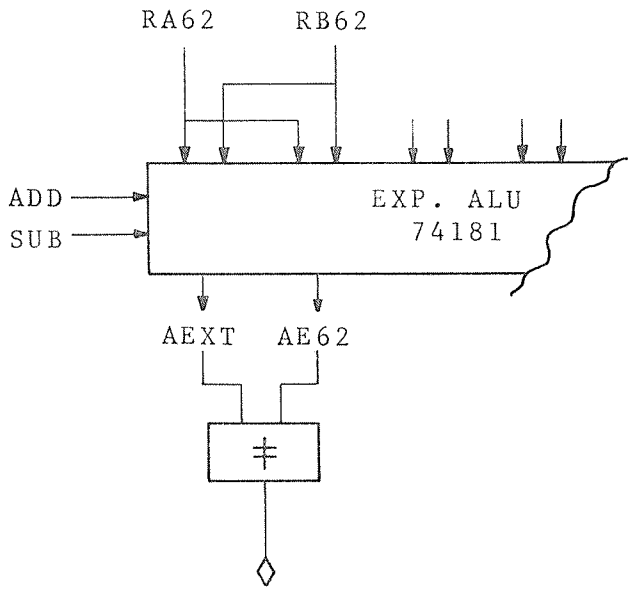
On the EXPONENT ARITHMETIC card 1524, bit 62 (the most significant bit of the exponent) for both operands is sign extended.



This gives us two 10 bit sign-extended exponents which are led into the exponent adder/subtracter ALU. (Adding for MPY, subtracting for DIV)

The two most significant bits out of the ALU (AE62 and AEXT) are checked.

These bits should be equal, otherwise we can not represent the exponent with 9 bits and it is overflow or underflow.



EQUAL?: NO OVERFLOW, UNDERFLOW.

OVERFLOW

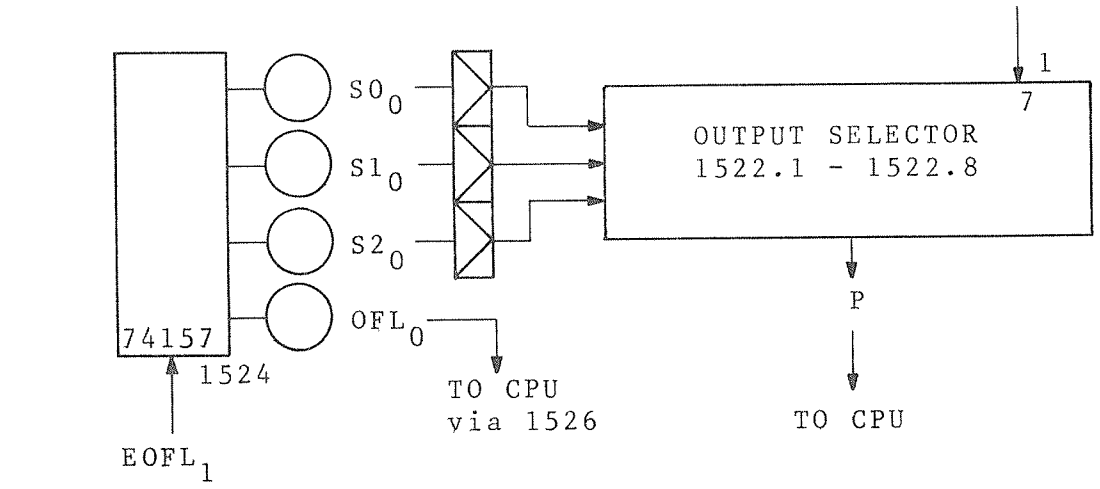
Exponent overflow indicates that the sum or difference of the exponents exceeds the capacity of the machine. A carry is propagated into the sign position of the floating-word.

The overflow flag is set (OFL,G 1524 to CPU via 1526) and the result will be the greatest positive or negative number representable dependent on the sign of the operands.

All bits except the sign bit are forced to one.

The output-selector signals (S0 - S2) are forced to all ones, (7), selecting a 1 in all bits to CPU, except the sign bit.

Divide by zero will also be generate OFL.

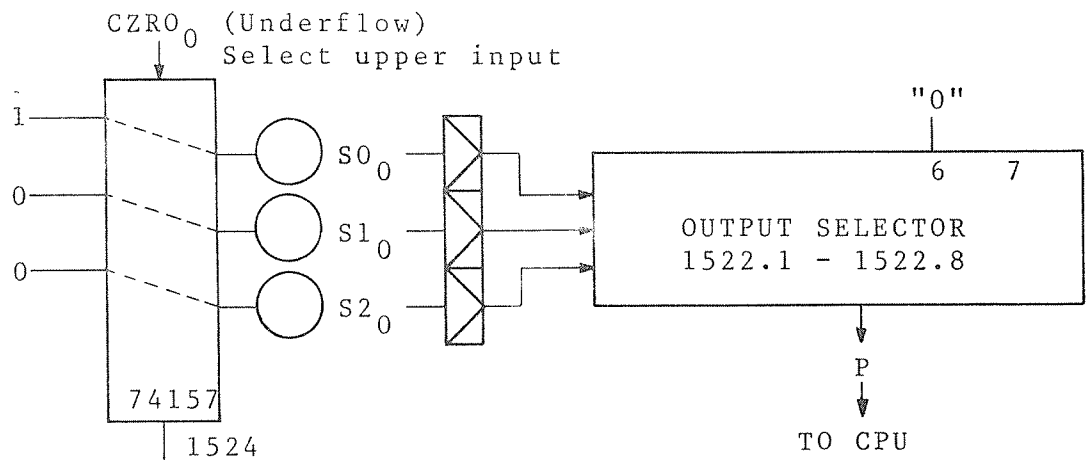


UNDERFLOW

Exponent underflow may be the result if a small positiv number is added to a small negative 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 (UFL,G 1524) and the result is set to zero.

The output-selector signals (S0-S2) are set to 6, selecting all zeroes through the output-selector 1522 to CPU.



MANTISSA MULTIPLICATION

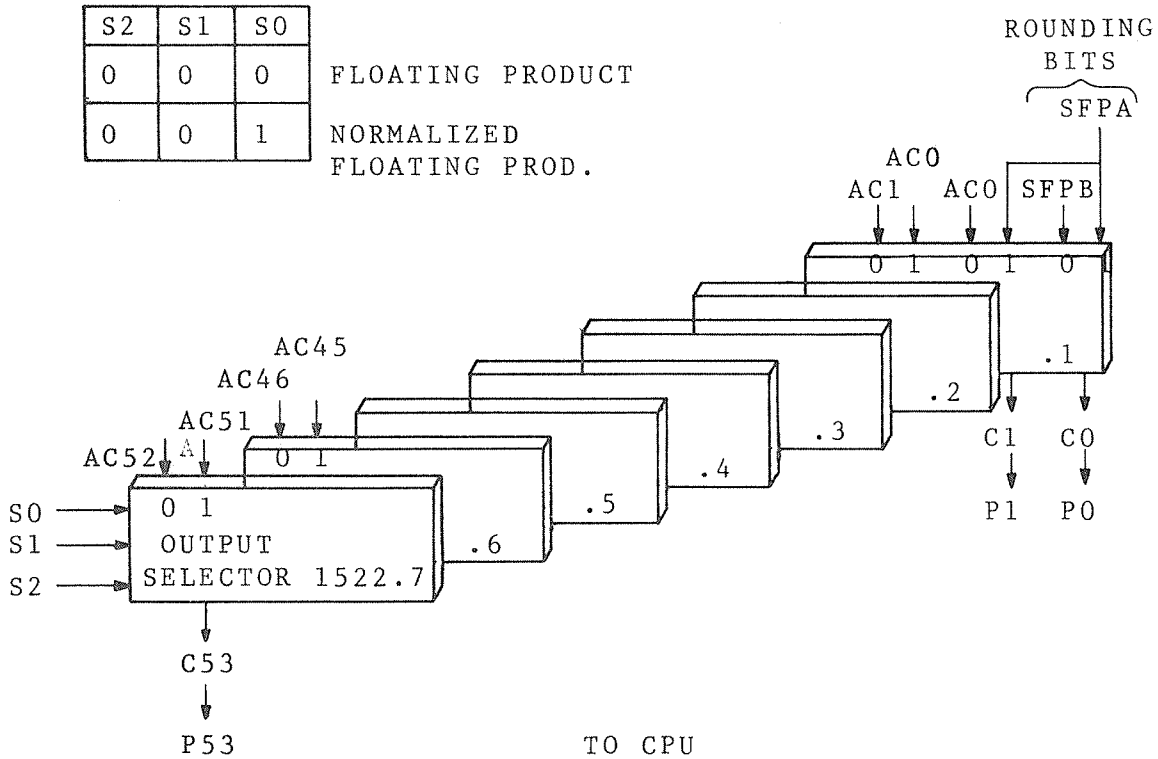
Multiplying the mantissas uses the same circuits as for Integer Multiply.

56 or 24 bits are multiplied, depending on whether double or single precision is required.

It is only the 54 or 22 most significant product bits that are taken care of. The least significant bits, the bits first shifted out of the Y-adder (Y0,1 or Y32,33) are connected to the SIPO as for Integer but not used.

The AC register keeps the final mantissa after multiplication. If the most significant mantissa bit (AC53) after multiplication is not a 1, the mantissa must be normalized; that is shifted one location to the left. This shifting is done on the Output-Selector card 1522 by simply selecting the AC register shifted one location left as input.

S2	S1	S0	
0	0	0	FLOATING PRODUCT
0	0	1	NORMALIZED FLOATING PROD.



FLOATING POINT MULTIPLY (FMU) NORD-50

Cycle:

C0 - B-operand entered into RB-register (1521) via 1523, 1 → RB54 (1523.4)
0 → RB55 (BSGN 1521.7)
A-operand entered into PISO on 1521 AP0,1 if double precision or AP32,33 if single → control of ED2,3 on 1526.

0 → AC (Y ≠ 0 → AC)

Exponent part AP54-63 and BP54-63 → 1524.

C1 - 56 in double or 24 in single pres. clock-pulse on CL, 28 or 12 iterations in AC. Same as for integer multiply.

C2 - The mantissa-product which remains in the AC register is normalized if necessary, using the selector on 1522 for the mantissa and selects the reduced exponent from 1524.

C3 - Present result on P-lines to CPU.

MANTISSA DIVISION

Dividing the mantissas uses the same circuits as for Integer Divide.

The B-operand (the divisor) is latched in the RB-register.

Aligning the divisor is much more simple than in integer divide, where we looked for the most significant 1 bit and shifted it to the most significant RB-position (RB54).

In floating the divisor is already normalized, a 1 bit in the most significant mantissa.

The 1 bit in the mantissa is not present in the CPU or memory, but is inserted on the selector on the B-INPUT card 1523, when a floating operand is selected. (refer to figure)

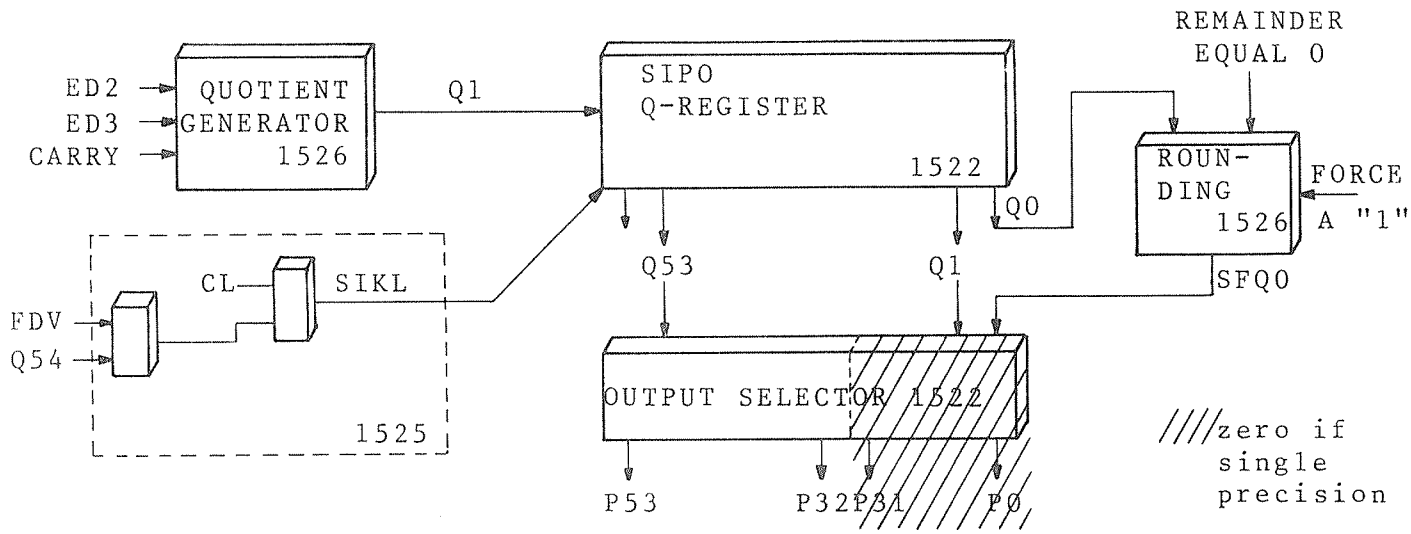
56 or 24 mantissa bits are divided depending on whether double or single precision is required.

In floating divide it is only the generated qoutient that is presented back to the CPU. The remainder (whats left in the AC-register) is not sent back to the CPU.

The Q or the quotient register (located on the OUTPUT SELECTOR 1522) keeps the final mantissa after division.

If the most significant mantissa bit (Q54) after division is not a 1, the mantissa must be normalized; that is shifted one location to the right.

This is done by generating a shift pulse (SIKL G 1525) and shift the Q-register before presenting the floating quotient to the CPU.



The least significant mantissa bit is forced to a 1 in the rounding circuits on the 1526 card. In one case this is not done and that is when the result is exactly represented; when the remainder is zero.

The most significant mantissa bit (Q54) is after normalization always a 1, and is not sent to the CPU. This bit is disabled on the EXPONENT ARITHMETIC card 1524, and instead the least significant exponent bit is presented on the P54 line to CPU.

FLOATING POINT DIVIDE (FDV) N-50

C0 - A and B-operand exponents (bit 54-63) are latched on the 1524 card.

The B-operand BP mantissa → SB (1523) is clocked into the RB-register (1521) 1 → RB54 (the hidden "1" bit in the most significant mantissa is inserted) (1523.4)
0 → RB55 (1521.7)

The AC-register is cleared at the leading edge at C0.

The A-operand AP mantissa is clocked into the AC-register via the X-selector and X-ALU.
AP53 → X52 → AC54
1 → AC55 (the hidden "1" bit in most sign. mantissa) (1521.7)
0 → AC56 (1524)

C1 - The exponents are subtracted on the 1524 card.

56 clock pulses if double precision or 24 clock pulses if single precision gives 28 or 12 subtractions in the X and Y adder.

Each time 2 alternative multiples of the RB-register are subtracted and clocked into the AC-register
if $Y \geq 0$ (YC57 = 1) : $4 \cdot Y \rightarrow AC$
if $Y < 0$ (YC57 = 0) : $4 \cdot X \rightarrow AC$

Two bits of the quotient ED3,2 or ED3,2-1 are entered into the SIPO register (Q register)

C2 - Normalized quotient if Q53 (Most sign. quotient) is = 0.

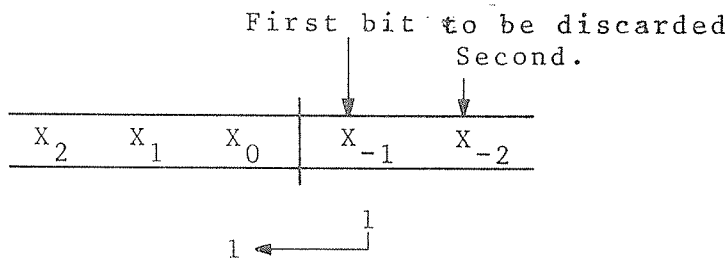
The quotient in the SIPO, Q register is shifted one location right, and an exponent reduced by one is selected on the 1524 card.

C3 - Floating exponent E55 → 62 → C → P (1522.8)
E54 → B → FC54 → P54 (1524)

Floating quotient in SIPO
Q → C → P (1522.1 - 1522.7)

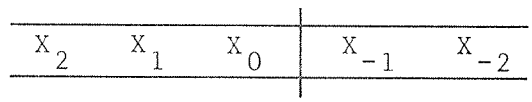
ROUNDING IN MUL/DIV UNIT.

Perfect rounding: IN N-50 ADD/SUB-UNIT.



\+ carry propagation and normalization if nessecary

Simplified rounding: IN N-50 MUL/DIV UNIT.



We always force the last representable bit (X_0) to a 1.

If there was a 0 in X_0, we add a 1 (We add one in 50% of the cases).

If X_0 was a 1, we do nothing (We add zero in 50% of the cases).

If we take into consideration the value of bit X_{-1} we get this percentage:

25% of cases: We add 1 when we should not (one to much)
$X_0 = 0$ $X_{-1} = 0$

25% of cases: } OK $X_0 = 1$ $X_{-1} = 0$

25% of cases: } OK $X_0 = 0$ $X_{-1} = 1$

25% of cases: We did not get rounding when we should (one to little) $X_0 = 1$ $X_{-1} = 1$

NOT FORCING A ONE

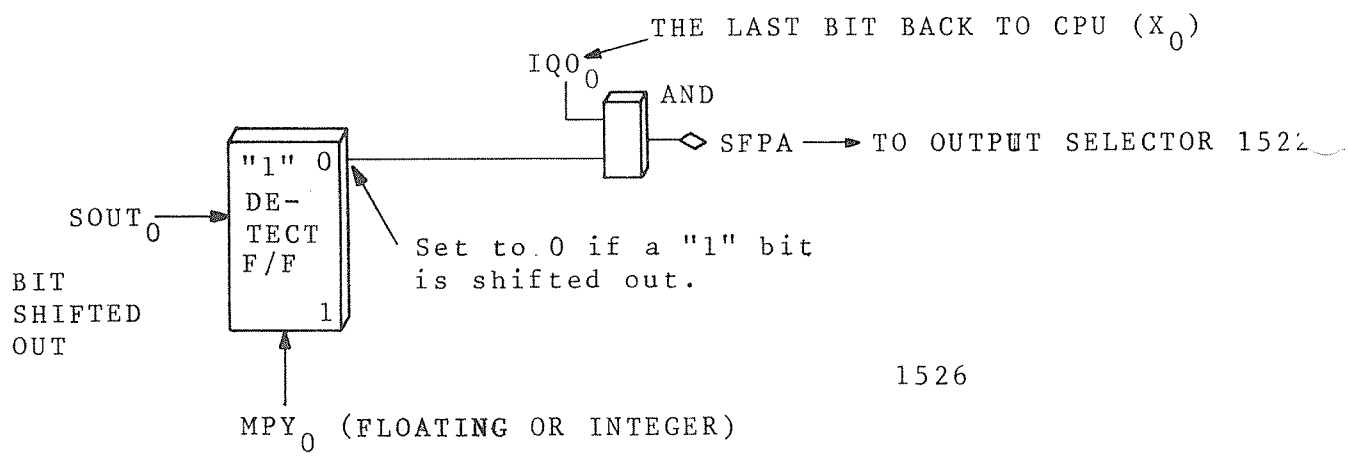
A one is not forced into the last bit if the result is exactly represented by the significant bits.

2 x 2 = 4 and not 4.00001

In multiply we look at the bits shifted out, and if no "1" bit is shifted out (all bits zero), we do not force a 1-bit.

If at any time during a division the remainder becomes zero, (Detected in the X-adder 1521) we do not force a 1-bit.

BIT-FORCING CIRCUITS FOR MULTIPLY



NB! FOR SINGLE-PRECISION THE "1" BIT IS FORCED INTO BIT 32 IN THE OUTPUT SELECTOR.

V.5 MUL-DIV TIMING

All the operations have the same basic timing. There are four separate cycles to run through.

C0 : The time it takes for the operands to become stable.

To clock the B-operand into the B-register and the A-operand into AC (dividend in division) and SIPO (during multiplication).

C1 : The arithmetic operation takes place.

"Milling" in AC (accumulator),

Repeated additions for multiply,

Repeated subtractions for divide.

C2 : Correction cycle.

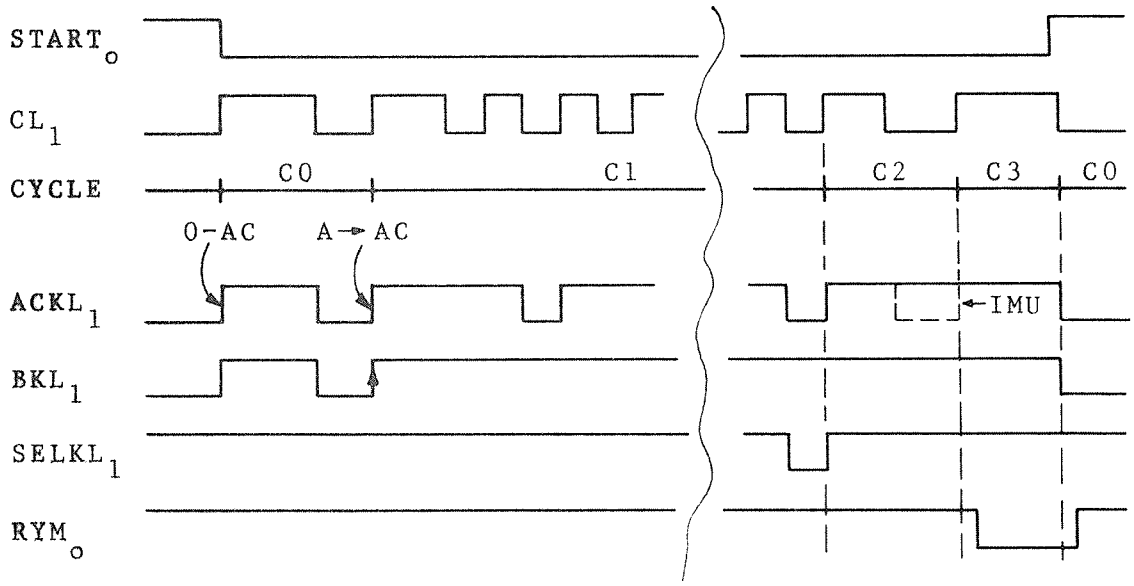
MPY : Subtraction to get the sign correct.

DIV : Remainder shifted back
Inverting quotients if necessary.

FMU,
FDV : Normalize: add, subtract exponent and shift mantissa.

C3 : Return result to CPU.

MUL-DIV TIMING



START : OR of all possible start signals.

SFMU Start Floating multiply
SFDV Start Floating divide
SMPY Start Integer multiply
SDIV Start Integer divide

The START signal starts the actual unit and terminates as response of a READY-signal.

CL : The clock-signal.

One clock-pulse during C0 and a number of clock-pulses during C1. Two clock-pulses for each iteration.

ACKL : In C0 the ACKL clears and clocks the A-operand into the AC-register.

In C1 the partial product or the partial remainder is clocked into the AC-register.

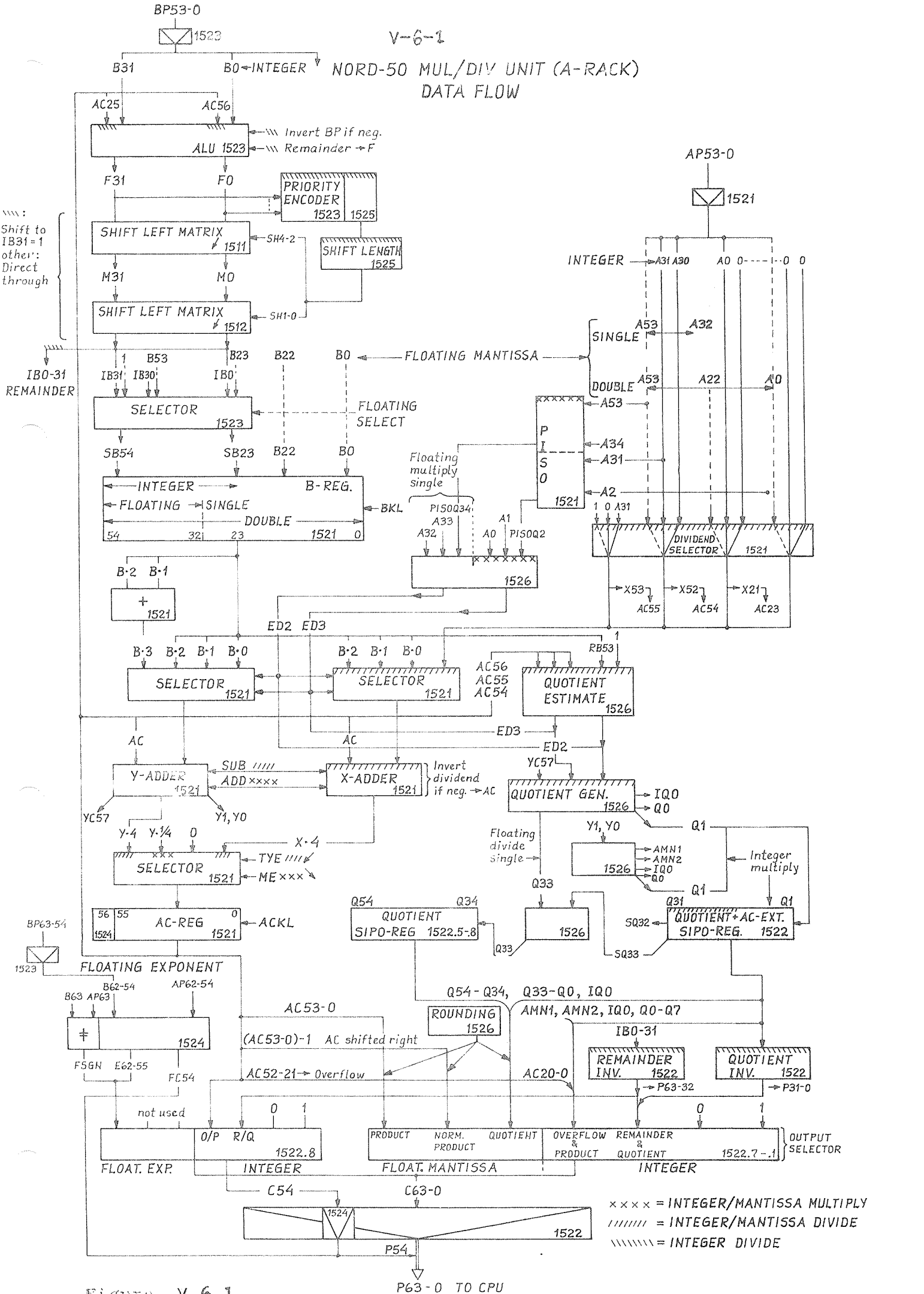
For IMU the subtracted result is clocked into the AC-register.

BKL : Clocks the B-operand.

SELKL : Clocks correction information on the last clock-pulse in C1.

RYM : Data ready on the CPU-bus.

NORD-50 MUL/DIV UNIT (A-RACK) DATA FLOW



APPENDIX A

APPENDIX A

NORD-50 OPERATORS PANEL

The idea of the operators panel is to give additional information about NORD-50.

The DISPLAY PC push-button (lightened when pushed) display the Program Counter in blocks of 4K in the ACTIVE ADDRESS lights. When pushing DISPLAY DATA REF the data reference address will be displayed as above.

Thus:

Address $0\text{-}7777_8$ will light bit 0
$10000\text{-}17777_8$ will light bit 1
$20000\text{-}37777_8$ will light bit 2, etc.

The indicator light in the middle field gives further information about any parity error (light in PARITY ERROR lamp to the left).

The four lamps to the right indicates in which part of the 32 bit memory word the parity error appeared.

BYTE 0 = Bits 0-7
BYTE 1 = Bits 8-15
BYTE 2 = Bits 16-23
BYTE 3 = Bits 24-31

The three next lamps indicate in what kind of reference the parity error occurred.

1. DATA REFERENCE
2. INDIRECT REFERENCE
3. INSTRUCTION READ

Light in the lower field buttons in the panel has the following meaning:

RUN:	NORD-50 is running
STOP:	NORD-50 stopped
EXTERNAL STOP:	NORD-50 was stopped from NORD-10
STATUS BREAK:	NORD-50 was stopped by an internal error condition
SIMULATE DATA & SIMULATE INSTR:	NORD-50 fetches data and instructions from NORD-10 in simulated memory mode
MEMORY DEPOSIT & MEMORY EXAMINE:	NORD-50 in STOP mode and NORD-50 memory locations are examined/deposited.

The operators panel is connected to the 1500 card in position B32.

APPENDIX B

APPENDIX B/I

THE NORD-50 TEST SYSTEM

The debugging and maintenance of the NORD-50 is based on the principles of having a computer test another computer.

The NORD-50 TEST SYSTEM is running in NORD-10, simulating the NORD-50 memory.

This gives us the possibility of testing the whole NORD-50 CPU regardless of the shape of the NORD-50 memory.

There is always the difficulty when running test programs that the memory has to be working before the CPU and the instruction execution can be tested.

Running the test in this way makes it possible to isolate an error almost regardless of how serious the error might be.

The error message is printed on a terminal with information about instruction failing, expected and achieved results (up to 16384 combinations of data are used to test one instruction).

Cards may easily be interchanged to observe if the error follows the card or not. The TEST SYSTEM will in fact never be destroyed by pulling cards in and out of NORD-50, since the program is located in NORD-10.

The error may be isolated by the use of an oscilloscope under the TEST SYSTEM, because the failing test is repeated with the bit combination failing until the next test or the next bit combination is wanted by the operator.

For every test the scope triggering signal and debugging hints are given in the manual.

The TEST SYSTEM responds to the following single letters commands:

D Display further information about the current error; a short description and name of the test program, followed by the code of the instruction.

C Continue; continues the test with the next input data.

B Back; restarts the current test with data taken from the beginning of the input data table.

N Next test.

P Previous test.

R Reset; restarts from the first program in the test table.

EXAMPLES OF NORD 50 TEST SYSTEM PRINT OUT:

```
        1000!
        NORD-50  TEST SYSTEM RUNNING

        TRANSFER TEST NO.   0
        INPUT  DATA 0 000 000 000 000 000   0 000 000 000 000 000
        OUTPUT DATA 0 001 000 000 000 000   0 001 000 000 000 000
        D
1       T0  SA COMM.

                  N
        TRANSFER TEST NO.   2
        INPUT  DATA 0 000 000 000 000 000   0 000 000 000 000 000
        OUTPUT DATA 0 001 000 000 000 000   0 001 000 000 000 000
        D
        T2 SD/TD
```

```
        TRANSFER TEST NO.   4
        INPUT  DATA 1 111 111 111 111 111   1 111 111 111 111 111
        OUTPUT DATA 1 111 111 111 110 000   1 111 111 111 111 111
2       D
        T7 LDR/STR
        INSTR. 0 000 000 001 001 100   0 101 000 000 000 000
```

```
        TRANSFER TEST NO.   5
        INPUT  DATA 1 111 111 111 111 111   1 111 111 111 111 111
        OUTPUT DATA 1 110 111 111 111 111   1 110 111 111 111 111
        D
        T7B LDR/STR
        INSTR. 0 000 000 001 001 100   0 011 111 111 111 111
        N
3       TRANSFER TEST NO.   5
        INPUT  DATA 1 111 111 111 111 111   1 111 111 111 111 111
        OUTPUT DATA 1 110 111 111 111 111   1 110 111 111 111 111
        C
        TRANSFER TEST NO.   5
        INPUT  DATA 1 111 111 111 111 111   1 111 111 111 111 110
        OUTPUT DATA 1 110 111 111 111 111   1 110 111 111 111 110
        C
```

```
        C
        TRANSFER TEST NO.  23
        INPUT  DATA 0 000 000 000 000 000   0 000 000 000 000 001
        OUTPUT DATA 0 000 000 000 000 000   0 000 000 100 000 001
        D
        T76 RTJ JUMP
        INSTR. 0 000 000 010 000 100   0 000 000 000 000 000
        C
4       TRANSFER TEST NO.  23
        INPUT  DATA 0 000 000 000 000 000   0 000 000 000 000 010
        OUTPUT DATA 0 000 000 000 000 000   0 000 000 100 000 010
        D
        T76 RTJ JUMP
        INSTR. 0 000 000 010 000 100   0 000 000 000 000 000
```

```
TRANSFER TEST NO.  25
INPUT  DATA 0 000 000 000 000 000   0 000 000 000 010 000
OUTPUT DATA 0 000 000 000 000 000   0 000 000 100 010 000
D
5       T16EL MEMORYC
TRANSFER TEST NO.  25
INPUT  DATA 0 000 000 000 000 000   0 000 000 000 100 000
OUTPUT DATA 0 000 000 000 000 000   0 000 000 100 100 000
```

```
R
TRANSFER TEST NO.  31
INPUT  DATA 0 000 000 000 000 000   0 000 000 000 000 000
6       OUTPUT DATA 0 000 000 000 000 000   0 000 000 000 000 001
D
T17 MEMORY ADRESS IN ADRESS
```

```
TWO ARGUMENT TEST NO.  53
OPERAND A 1 111 111 111 111 111   1 111 111 111 111 111
OPERAND B 0 000 000 000 000 000   0 000 000 000 000 000
RESULT    1 111 111 111 111 111   1 111 111 111 111 111
CORRECT   0 000 000 000 000 000   0 000 000 000 000 000
D
T40 SET ARG
7       INSTR. 1 010 100 100 000 000   0 000 000 000 000 000

C
TWO ARGUMENT TEST NO.  53
OPERAND A 1 111 111 111 111 111   1 111 111 111 111 111
OPERAND B 1 111 111 111 111 111   1 111 111 111 111 111
RESULT    1 111 111 111 111 111   1 111 111 111 111 111
CORRECT   0 000 000 000 000 000   1 111 111 111 111 111
```

```
R
TWO ARGUMENT TEST NO. 1 4
OPERAND A 0 000 000 000 000 000   0 000 000 000 000 000
OPERAND B 0 000 000 000 000 000   0 000 000 000 000 000
RESULT    0 000 000 000 000 000   0 000 000 000 000 010
CORRECT   0 000 000 000 000 000   0 000 000 000 000 000
D
8       T156 JZM JUMP
INSTR. 0 000 000 010 111 000   0 010 000 000 000 000

C
TWO ARGUMENT TEST NO. 1 4
OPERAND A 1 111 111 111 111 111   1 111 111 111 111 111
OPERAND B 0 000 000 000 000 000   0 000 000 000 000 001
RESULT    0 000 000 000 000 000   0 000 000 000 000 010
CORRECT   0 000 000 000 000 000   0 000 000 000 000 001
```

```
N
TWO ARGUMENT TEST NO. 131
OPERAND A 0 000 000 000 000 000    0 000 000 000 000 000
OPERAND B 1 111 111 111 111 111    1 111 111 111 111 111
RESULT    0 000 000 000 000 000    0 000 100 000 010 000
CORRECT   0 000 000 000 000 000    0 000 000 000 010 000
N
9       TWO ARGUMENT TEST NO. 131
OPERAND A 0 000 000 000 000 000    0 000 000 000 000 000
OPERAND B 1 111 111 111 111 111    1 111 111 111 111 111
RESULT    0 000 000 000 000 000    0 000 100 000 100 000
CORRECT   0 000 000 000 000 000    0 000 000 000 100 000
D
T100 BIT SET
INSTR. 0 010 011 000 000 011    1 111 000 000 000 101
```

```
TWO ARGUMENT TEST NO. 131
OPERAND A 1 111 111 111 111 111    1 111 111 111 111 111
OPERAND B 1 111 111 111 111 111    1 111 111 111 111 111
RESULT    1 111 111 111 111 111    1 111 111 111 111 110
CORRECT   1 111 111 111 111 111    1 111 111 111 111 111
10      D
T100 BIT SET
INSTR. 0 010 011 000 000 011    1 111 111 111 000 001
```

```
TWO ARGUMENT TEST NO. 127
OPERAND A 0 000 000 000 000 000    0 000 000 000 000 000
OPERAND B 0 000 000 000 000 000    0 000 000 000 000 000
11      RESULT    0 000 000 000 000 000    0 000 000 000 000 001
CORRECT   0 000 000 000 000 000    0 000 000 000 000 010
D
T144 BIT SKIP ZERO
INSTR. 0 011 011 000 000 000    0 000 000 001 000 111
```

```
FLOATING TEST NO 162
MOST OPERAND A 0 100 010 000 000 000    0 000 000 000 000 000
MOST OPERAND B 0 100 010 000 000 000    0 000 000 000 000 000
MOST  RESULT   1 111 111 111 111 111    1 111 000 000 001 111
MOST  CORRECT  0 100 010 001 000 000    0 000 000 000 000 000
12
LEAST OPERAND A0 000 000 000 000 000    0 000 000 000 000 001
LEAST OPERAND B0 000 000 000 000 000    0 000 000 000 000 000
LEAST  RESULT  0 000 000 000 000 000    0 000 000 000 000 001
LEAST  CORRECT 0 000 000 000 000 000    0 000 000 000 000 001
D
T170 RAFD
INSTR. 0 110 100 000 000 001    0 001 010 001 010 010
```

```
FLOATING TEST NO 161
MOST OPERAND A 1 111 111 111 111 111   1 111 111 111 111 111
MOST OPERAND B 1 111 111 111 111 111   1 111 111 111 111 111
MOST  RESULT   0 000 000 000 000 000   0 000 000 000 000 001
MOST  CORRECT  0 000 000 000 000 000   0 000 000 000 000 001
LEAST  RESULT  0 000 000 000 000 000   0 000 000 000 100 000
LEAST  CORRECT 0 000 000 000 000 000   0 000 000 000 000 000
C
FLOATING TEST NO 161
MOST OPERAND A 0 000 000 000 000 000   0 000 000 000 000 001
MOST OPERAND B 0 000 000 000 000 000   0 000 000 000 000 001
13 MOST  RESULT   0 000 000 000 000 000   0 000 000 000 000 001
MOST  CORRECT  0 000 000 000 000 000   0 000 000 000 000 001
LEAST  RESULT  0 000 000 000 000 000   0 000 000 000 100 000
LEAST  CORRECT 0 000 000 000 000 000   0 000 000 000 000 000
D
T167 MUL
INSTR. 0 000 000 001 100 000   0 001 000 000 000 000
C
FLOATING TEST NO 161
MOST OPERAND A 0 000 000 000 000 000   0 000 000 000 000 010
MOST OPERAND B 0 000 000 000 000 000   0 000 000 000 000 010
MOST  RESULT   0 000 000 000 000 000   0 000 000 000 000 100
MOST  CORRECT  0 000 000 000 000 000   0 000 000 000 000 100
LEAST  RESULT  0 000 000 000 000 000   0 000 000 000 100 000
LEAST  CORRECT 0 000 000 000 000 000   0 000 000 000 000 000
```

```
FLOATING TEST NO 160
MOST OPERAND A 0 000 000 000 000 000   0 000 000 000 000 000
MOST OPERAND B 0 000 000 000 000 000   0 000 000 000 000 000
MOST  RESULT   1 111 111 111 111 100   0 000 000 000 000 000
MOST  CORRECT  0 000 000 000 000 000   0 000 000 000 000 000
LEAST  RESULT  0 000 000 000 000 000   0 000 000 000 000 0
LEAST  CORRECT 0 000 000 000 000 000   0 000 000 000 000 0
D
T166 DIV
INSTR. 0 000 000 001 100 100   0 001 000 000 000 000
N
14 FLOATING TEST NO 165
MOST OPERAND A 0 111 011 011 000 111   0 001 110 000 000 000
MOST OPERAND B 0 110 010 010 000 000   0 000 000 000 000 000
MOST  RESULT   0 101 001 010 111 111   1 111 111 111 111 111
MOST  CORRECT  0 101 001 010 000 111   0 001 110 000 000 000

LEAST OPERAND A0 000 000 000 000 000   0 000 000 000 000 000
LEAST OPERAND B0 000 000 000 000 000   0 000 000 000 000 000
LEAST  RESULT  1 111 111 111 111 110   0 000 000 000 000 000
LEAST  CORRECT 0 000 000 000 000 000   0 000 000 000 000 000
D
T173 RDFD
INSTR. 0 110 100 110 000 001   0 011 010 001 010 010
```

```
        FLOATING TEST NO 164
        MOST OPERAND A 0 100 000 001 000 000   0 000 000 000 000 000
        MOST OPERAND B 0 100 111 111 000 111   0 001 110 001 110 001
        MOST  RESULT   0 100 101 111 000 111   0 001 110 001 110 001
        MOST  CORRECT  0 100 111 111 000 111   0 001 110 001 110 001

15      LEAST OPERAND AO 000 000 000 000 000   0 000 000 000 000 000
        LEAST OPERAND BO 000 000 000 000 000   0 000 000 000 000 000
        LEAST  RESULT  0 000 000 000 000 000   0 000 000 000 000 000
        LEAST  CORRECT 0 000 000 000 000 000   0 000 000 000 000 000
        D
        T172 RMFD
        INSTR. 0 110 100 100 000 001   0 011 010 001 010 010
```

```
        N
        FLOATING TEST NO 165
        MOST OPERAND A 0 110 000 000 000 000   0 000 000 000 000 000
        MOST OPERAND B 0 100 000 001 100 000   0 000 000 000 000 000
        MOST  RESULT   0 101 101 111 010 101   0 101 010 101 010 101
        MOST  CORRECT  0 101 111 111 010 101   0 101 010 101 010 101

16      LEAST OPERAND AO 000 000 000 000 000   0 000 000 000 000 000
        LEAST OPERAND BO 000 000 000 000 000   0 000 000 000 000 000
        LEAST  RESULT  0 101 010 101 010 101   0 101 010 101 010 101
        LEAST  CORRECT 0 101 010 101 010 101   0 101 010 101 010 101
        D
        T173 RDFD
        INSTR. 0 110 100 110 000 001   0 101 010 001 010 010
```

```
        C
        FLOATING TEST NO 165
        MOST OPERAND A 0 111 011 011 000 111   0 001 110 000 000 000
        MOST OPERAND B 0 110 010 010 000 000   0 000 000 000 000 000
17      MOST  RESULT   0 101 001 010 111 111   1 111 111 111 111 110
        MOST  CORRECT  0 101 001 010 000 111   0 001 110 000 000 000

        LEAST OPERAND AO 000 000 000 000 000   0 000 000 000 000 000
        LEAST OPERAND BO 000 000 000 000 000   0 000 000 000 000 000
        LEAST  RESULT  0 000 000 000 000 000   0 000 000 000 000 000
        LEAST  CORRECT 0 000 000 000 000 000   0 000 000 000 000 000
        D
        T173 RDFD
        INSTR. 0 110 100 110 000 001   0 011 010 001 010 010
```

ALL AVAILABLE TESTS COMPLETED

APPENDIX B/III

NORD 50 TEST PROGRAMS PRINT OUT:

```
@N-50
                                        % INSTRUCTION TEST
NORD-50 MONITOR  -  E
*LOAD ITEST
NEW RUN DATA? (0 OR 1):
NO FILE OPENED WITH THIS NUMBER

 *** I/O-ERROR 0000124 AT ADDRESS 0015221 FILE 0000007
NOT OPENED FOR SEQUENTIAL READ
 *** JOB ABORTED ***
- *** END ***  - AT: 000015  -
*OPEN TERM 7 W
*LOAD ITEST
NEW RUN DATA? (0 OR 1):    1
LOOP ON ERROR? (0 OR 1):    0
NO. OF BITS PER GROUP (<14):   13
EXPECTED TIME PER RUN IS      46970.461 SEC.
NO. OF RUNS :   100
TIME NEEDED AFTER DATA READY IS    4697047.000 SEC.
NEW RUN DATA? (0 OR 1):
```

```
- EXTERNAL STOP  - AT: 004330  -    % MUL/DIV FLOATING
*LOAD FTEST                          TEST
NEW RUN DATA? (TYPE 0 OR 1)   1
LOOP ON ERROR? (0 OR 1):    0
TOLERABLE MULTIPLY ERROR :    0
TOLERABLE DIVIDE ERROR (>0):   1
NO. OF BITS PER GROUP (<11):   10
EXPECTED TIME PER RUN IS      69333.359 SEC.
NO. OF RUNS :   100
TIME NEEDED AFTER DATA READY IS    6933335.000 SEC
NEW RUN DATA? (TYPE 0 OR 1)
```

```
- EXTERNAL STOP  - AT: 004330  -  % MUL/DIV DOUBLE-
*LOAD DFTEST                        FLOATING TEST
NEW RUN DATA? (TYPE 0 OR 1)   1
LOOP ON ERROR? (0 OR 1):    0
TOLERABLE MULTIPLY ERROR :    0
TOLERABLE DIVIDE ERROR (>0):   1
NO. OF BITS PER GROUP (<11):   10
EXPECTED TIME PER RUN IS      69333.359 SEC.
NO. OF RUNS :   100
TIME NEEDED AFTER DATA READY IS    6933335.000 SEC
NEW RUN DATA? (TYPE 0 OR 1)
```

```
*PL TMEM
*B-C 200000,,AP          % MEMORY TEST
*RUN
THIS IS YOUR N-50 MOS MEMORY TEST PROGRAM
THE PROGRAM OCCUPIES LOCATIONS: 0000000  TO  0001227
LOWER TEST ADDRESS(OCTAL): 1230
UPPER TEST ADDRESS(OCTAL): 177777
SPECIFY TESTS TO BE RUN BY OCTAL
NUMBERS TERM. BY CR.STOPS WHEN 0 IS TYPED
77 MEANS ALL TESTS
77

WANT TESTS TO LOOP? 0:NO  1:YES
1
ERRORS WILL BE REPORTED IN THE FOLLOWING FORMAT
TEST1,2,3&4: <FAILING ADDR.>,<FOUND DATA>,<EXP.DATA>,<TEST NO>
TEST5: <WALK PATT. ADDR.>,<SAME AS ABOVE>,<READ/WRITE>

TEST PATTERN  25252525252  IS FINISHED

TEST PATTERN  12525252525  IS FINISHED

TEST PATTERN  37777777777  IS FINISHED

TEST PATTERN  00000000000  IS FINISHED

- STOP 0  - AT: 000415  -
```

```
*PL BM2
*OPEN TERM 5 W
*RUN                     % BENCH-MARK ASEA
BENCHMARK 2
  0.83349          94          28
 -0.00000          94          30
  0.88696          94          28
  0.00001          94          30
  0.90586          96          28
  0.99673          94          30
  1.00355          96          28
  1.00066          94          30
  0.99674          94          28
  0.99242          94          28
  0.98764          94          30
  0.98247          96          28
  0.97663          94          30
  0.97021          96          28
  0.96303          94          30
  0.95532          96          28
  0.94699          94          30
  0.93818          94          28
  0.92892          94          28
  0.91946          94          30

TOTAL TIME(S/100)        2640

- *** END ***  - AT: 000015  -
*CPU
CPU-TIME USED IN SECONDS:        25.28
*M
                ND-05.007.01
```

```
% INSTRUCTION
EXECUTE TIMES

*PL SEK
*B-C ,,N
*RUN
LDR   1.57
STR   1.61
ADD   1.57
LDF   2.42
STF   2.37
FAD   1.83
FADD  2.53
RAF   0.82
RAFD  0.82
FMU   4.53
FMF   3.51
FMFD  6.63
FDV   4.53
FDVD  8.36
RDF   3.51
RAD   0.72
ADDA  0.72
MPY   5.45
DIV   2.51
ADM   2.33
SLR   0.81
SLRD  0.81
JFM-  0.72
EXC'  1.26
EXC"  2.15
JFM+  1.40
- *** END ***
```

APPENDIX C/I

NORD-10 INSTRUCTIONS		NORD-50 INSTRUCTIONS	
DATA TRANSFER INSTRUCTIONS			
MOVING DATA FROM ONE LOCATION TO ANOTHER IN THE COMPUTER SYSTEM			
Memory ⟷ Register Block			
Ea = D, (B), (X) and I		Ea = D + (B) + (X) and I	
LDA	Load A reg. with (Ea)	LDR	Load R with (Ea)
LDT	Load T reg. with (Ea)		
LDX	Load X reg. with (Ea)		
LDD	Load A and D reg. with (FEa)	LDD	Load FD with (FEa)
LDF	Load T, A and D reg. with (FEa) (Ea + 2)		
LRB	Load register block		
LBYT	Load byte		
STZ	Store zero in Ea	STR O,	Store zero in Ea
STA	Store A reg.	STR R,	Store (R) in Ea
STT	Store T reg.		
STX	Store X reg.		
STD	Store A and D reg.	STD	Store (FD) in (FEa)
STF	Store A, D and T reg.		
STB	Store Register block		
SBYT	Store byte		
Register ⟷ Register Transfer			
COPY	SR ⟶ DR		
SWAP	SR ⟷ DR		
TRA	EXT reg. ⟶ A	RIN	(Overflow or Remainder reg.) ⟶ R
TRR	A ⟶ EXT reg.	ROUT	(R) ⟶ Overflow or remainder reg.
IOX	Data to-from I/O device buffer reg.		
IDENT	Ident code ⟶ A reg.		
Register ⟷ Memory Exchange			
		XMR	Exchange (Ea) and (R)

FEa = Ea, Ea + 1
SR = Source Register
DR = Destination Register
Ea = Effective or calculated address

ARITHMETIC/LOGIC INSTRUCTIONS

MODIFY THE CONTENT OF A REGISTER OR MEMORY WITH AN ARITHMETIC OR LOGIC OPERATION

Memory ⟶ Register Block

ADD	Add (Ea) to (A)
SUB	Subtract (Ea) from (A)
AND	AND (Ea) to (A)
ORA	OR (Ea) to (A)
MPY	Multiply (Ea) by (A)
FAD	Floating Add (FEa + FR)
FSB	Floating Subtract
FMU	Floating Multiply
FDV	Floating Divide

ADD	Add (Ea) to (R)
SUB	Subtract (Ea) from (R)
AND	AND (Ea) to (R)
MPY	Multiply (Ea) by (R)
DIV	Divide (Ea) by (R)
FAD	Floating Add (Ea) to (F)
FSB	Floating Subtract
FMU	Floating Multiply
FDV	Floating Divide

} single pres.

FADD	Floating Add (FEa + FD)
FSBD	Floating Subtract
FMUD	Floating Multiply
FDVD	Floating Divide

} double pres.

Register Block ⟶ Memory

ADM	Add (R) to (Ea)

Register ⟷ Register Operation

RADD	Register Add
RSUB	Register Subtract
RMPY	Register Multiply
RDIV	Register Divide

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 prec. Floating Reg. Add
RSFD	Double prec. Floating Reg. Subtract
RMFD	Double prec. Floating Reg. Multiply
RDFD	Double prec. Floating Reg. Divide

FR = NORD-10 Floating Register: T, A and D.

ARITHMETIC/LOGIC INSTRUCTIONS

RAND	Register AND	RND	Register AND
		RNDA	Register AND, use complement of (SRA)
		RNDB	Register AND, use complement of (SRB)
REXO	Register exclusive OR	RXO	Register exclusive OR
		RXOA	Register exclusive OR, use complement of (SRA)
		RXOB	Register exclusive OR, use complement of (SRB)
RORA	Register OR	ROR	Register OR
		RORA	Register OR, use complement of (SRA)
		RORB	Register OR, use complement of (SRB)
		SZR	Set Register zero

Argument	*Instructions*		
AAA	Add Argument to A reg.	ADDA	Add Argument to (R)
AAX	Add Argument to X reg.		
AAT	Add Argument to T reg.		
AAB	Add Argument to B reg.		
		ADCA	Add complement of argument to (R)
		XORA	Exclusive OR
		ANDA	AND
		ORA	OR

DATA MANIPULATION INSTRUCTIONS

Shift Instructions

SHT	Shift T register	SHIFT (R); **Rot, Arithmetical, Right, Left**
SHD	Shift D register	SLR, SRR, SLA, SRA
SHA	**Shift A register**	
SAD	**Shift A and D register**	SHIFT DOUBLE REGISTER; **Rot, Arithmetical, Right, Left**
	Rot, Arithmetical, Zero Input, Right, Left	SLRD, SRRD, SLAD, SRAD

Bit Instructions

BSET	**One, Zero, BCM,** Bit no. **DR**	BSET	One Zero, BCM, Bit no. R

Argument Instructions

SAA	Set argument to A reg.	SETA	Set argument to R
SAX	Set argument to X reg.		
SAT	Set argument to T reg.		
SAB	Set argument to B reg.		
		SECA	Set complement of argument to R

Convert Floating Instructions

DNZ	Convert floating to integer	FIX	Convert floating to integer
		FIR	Convert floating to rounded integer
		FIXD	Convert double pres. floating to integer
		FIRD	Convert double pres. floating to rounded integer
NLZ	Convert integer to floating	FLO	Convert integer to floating
		FLOD	Convert integer to floating double

MIN Instructions: Increment memory cell, ship if zero.

MIN Ea	MIN Ea

SEQUENCING INSTRUCTIONS

INSTRUCTION TRANSFERRING PROGRAM EXECUTION FROM CURRENT LOCATION TO SOME OTHER LOCATION IN MEMORY

JPL	Return Jump P + 1 → L	RTJ R, Ea	Save P + 1 → R Jump to Ea
JMP	Non-Return Jump	RTJ O, Ea	No save Jump to Ea

Conditional Jump

JAP	Jump if (A) > 0	JRP	Jump if (R) ⩾ 0
JAN	Jump if (A) < 0	JRN	Jump if (R) < 0
JAZ	Jump if (A) = 0	JRZ	Jump if (R) = 0
JAF	Jump if (A) ≠ 0	JRF	Jump if (R) ≠ 0
JXN	Jump if (X) < 0		
JXZ	Jump if (X) = 0		

Add and Conditional Jump

JPC	Add 1 to X and jump if (X) ⩾ 0	JPM	Add (M) to R and jump if (R) ⩾ 0
JNC	Add 1 to X and jump if (X) < 0	JNM	Add (M) to R and jump if (R) < 0
		JZM	Add (M) to R and jump if (R) = 0
		JFM	Add (M) to R and jump if (R) ≠ 0

Skip Instructions (skip next instruction if specified condition is true)

SKIP IF: (SR) ⩾, <, =, ≠, (DR)	SKIP IF (R); ⩾, <, =, ≠, ARG (R); ⩾, <, =, ≠, −ARG
BSKP (BIT SKIP) if specified bit equals zero, one, DR	BSZ Bit skip on zero BSO Bit skip on one
	Add/Subtract Register and skip if result: ⩾ 0, < 0, = 0, ≠ 0.
	Add/Subtract Floating Register and skip if result: ⩾ 0, < 0, = 0, ≠ 0.
	Add/Subtract Double precision register and skip if result: ⩾ 0, < 0, = 0, ≠ 0.

Execute Instructions

EXR	(SR) Execute (source register) as an instruction	EXC 0, Ea EXC 1, R	Execute (Ea) as an instruction Execute Ea as an instruction Ea = (X) + (B) + D

SYSTEM CONTROL INSTRUCTIONS

ION	Interrupt system ON		
IOF	Interrupt system OFF		
PON	Paging system ON		
POF	Paging system OFF		
WAIT	STOP Give up priority if interrupt system is on	STOP	Stop N-50

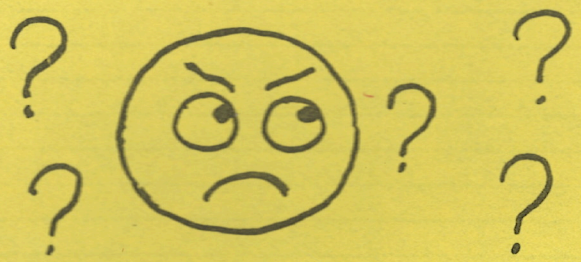
```
                                                    10 J=1
                                                    20 K=2
                                                    30 L=J*K
                                                    40 I=L+100
                                                    50 M=I-L*(K-1)
- NORD 50 FTN COMPILER -                                END
$DBLIST                                                 EOF
$COM 1,1,0


          177777     14                              MAIN
          177777      3                             ENTR.
          177777      2    00001040000    RTJ   04, 0000,    ,
               0      3                             8VBAS
               0      2    00000000000    STOP  00, 00, 00
               1      2    24300000000    SETA  06, 00000
               2      2    24040010000    SETA  01, 10000
               3      2    24100020000    SETA  02, 20000
               4      2    24140030000    SETA  03, 30000
     J=1
               5      2    24500000001    SETA  12, 00001
               6      2    00267120007    STR   12, 0007, 05,
     K=2
               7      2    24540000002    SETA  13, 00002
              10      2    00267130010    STR   13, 0010, 05,
     L=J*K
              11      2    14100121213    RMU   12, 12, 13
              12      2    00267120011    STR   12, 0011, 05,
     I=L+100
              13      2    24500400144    ADDA  12, 00144
              14      2    00267120012    STR   12, 0012, 05,
     M=I-L*(K-1)
              15      2    24540600001    ADCA  13, 00001
              16      2    00270130011    MPY   13, 0011, 05,
              17      2    14040121213    RSB   12, 12, 13
              20      2    00267120013    STR   12, 0013, 05,
     END
              21      3                             LEAV.
              21      2    00001000000    RTJ   00, 0000,    ,
              22      4                             8VBAS
              22      4                             8ABAS
              22      2    00000000000
              23      2    00000000000
              24      2    00000000000
              25      2    00000000000
              26      2    00000000000
              27      2    00000000000
              30      3                             8ABAS
              30      2    37777777777
              31      2    00000000000
              32      2    00000000000
              33      2    00000000000
              34      2    00000000000
              35      2    00000000000
              36      2    00000000000
              37      1          0
              37      6
    7*        EOF
E
USER BREAK AT   11216
```

FORTRAN PROGRAM NORD 10

```
@FTN

NORD FTN-1639C
$OBLIST
$PREMAP
$CLC
$COM FOLA,TERM,0
```

```
                                                    10 J=1
                                                    20 K=2
                                                    30 L=J*K
                                                    40 I=L+100
                                                    50 M=I-L*(K-1)
                                                       END
                                                       EOF
                                          *M TO(0)
                                          *W "FOLA"
                    1     17
                    1     32
                    1     14                  MAIN
                    1      1    135002    JPL I * 002
                    2      1         0    STZ * 000
                    3     20              8RTEN
                    4     16              LABEL 10
     4          10 J=1
                    4      1    170401    SAA 001
                    5      1      4615    STA ,B - 163
                    6     16              LABEL 20
     6          20 K=2
                    6      1    170402    SAA 002
                    7      1      4616    STA ,B - 162
                   10     16              LABEL 30
    10          30 L=J*K
                   10      1     44616    LDA ,B - 162
                   11      1    120615    MPY ,B - 163
                   12      1      4617    STA ,B - 161
                   13     16              LABEL 40
    13          40 I=L+100
                   13      1    170544    SAA 144
                   14      1     60617    ADD ,B - 161
                   15      1      4620    STA ,B - 160
                   16     16              LABEL 50
                   16      1     44616    LDA ,B - 162
                   17      1    172777    AAA - 001
    16          50 M=I-L*(K-1)
                   20      1    120617    MPY ,B - 161
                   21      1      4621    STA ,B - 157
                   22      1     44620    LDA ,B - 160
                   23      1     64621    SUB ,B - 157
                   24      1      4622    STA ,B - 156
    25             END
                   25      1    135001    JPL I * 001
                   26     20              8LEAV
                   27      7       227       2
                   27     11        23
                   52      7        52      31
                   52     21

---------- LOCAL  IDENTIFIERS ----------

   INTEGER VARIABLE        J        43
   INTEGER VARIABLE        K        44
   INTEGER VARIABLE        L        45
   INTEGER VARIABLE        I        46
   INTEGER VARIABLE        M        50
            TOTAL LOCAL DATASPACE       6 (OCTAL)

---------- COMMON IDENTIFIERS ----------
   NONE

---------- EXTERNAL REFERENCES ----------
   NONE
                              1     23
     1              EOF          ND-05.007.01

0 STATEMENTS COMPILED
```

* * * * * * * * * * SEND US YOUR COMMENTS!!! * * * * * * * * * *



Are you frustrated because of unclear information in this manual? Do you have trouble finding things? Why don't you join the Reader's Club and send us a note? You will receive a membership card - and an answer to your comments.

Please let us know if you

- find errors
- cannot understand information
- cannot find information
- find needless information

Do you think we could improve the manual by rearranging the contents? You could also tell us if you like the manual!!

* * * * * * * * * HELP YOURSELF BY HELPING US!! * * * * * * * * *

Manual name: NORD - 50 Functional Description Manual number: ND-05.007.01

What problems do you have? (use extra pages if needed) _____

Do you have suggestions for improving this manual? _____

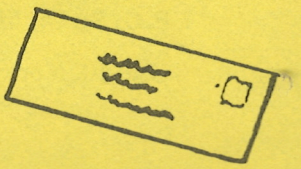
Your name: _____ _____ _____ Date: _____

Company: _____ _____ _____ Position: _____

Address: _____

What are you using this manual for? _____

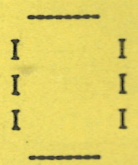
Send to: Norsk Data A.S.
Documentation Department
P.O. Box 4, Lindeberg Gård
Oslo 10, Norway



Norsk Data's answer will be found on reverse side

Answer from Norsk Data _____

Answered by _____ _____ _____ _____ Date _____



Norsk Data A.S.
Documentation Department
P.O. Box 4, Lindeberg Gård
Oslo 10, Norway

– we make bits for the future

NORSK DATA A.S BOX 4 LINDEBERG GARD OSLO 10 NORWAY PHONE: 30 90 30 TELEX: 18661