ND-110 Instruction Set

ND-06.029.1 EN

ND-110 Instruction Set

ND-06.029.1 EN

Preface

The product

The ND-110 Instruction Set Manual describes the data, address and instruction format of the ND-110 CPU (Central Processing Unit) - product number ND 110110.

The reader

This manual is intended for all personnel who require information about the ND-110 assembly language.

Assumed background

The reader is assumed to have a general knowledge of programming techniques and computers.

The manual

This manual is a reference guide to the low-level programming language of the ND-110 CPU. Each chapter can be read individually and outlines the different aspects of the low-level programming as follows:

- Chapter 1. Instruction and data format.
- Chapter 2. Memory addressing.
- Chapter 3. Alphabetic index of the instruction mnemonics described and detailed description of the instructions.

The Appendices give a glossary of terms, PLANC listings of the new SINTRAN instructions, an alphabetic list of the instruction mnemonics with their octal codes and the TRR and TRA instructions for internal registers.

Related manuals

The following manuals may be useful:

ND-110 Functional Description (ND-06.026) - a detailed description of the hardware and software features, in particular micro-instructions, program levels and ND-110 enhancements.

ND-100 Reference Manual (ND-06.014) - a general outline of the ND-100 computer.

MAC Interactive Assembly and Debugging System User's Guide (ND-60.096) - information on the ND-100 instruction set and assembler disassembler operation.

Table of contents

Table of appendices

List of figures

List of tables

CHAPTER 1 INSTRUCTION AND DATA FORMAT

CHAPTER 1 INSTRUCTION AND DATA FORMAT

The ND-110 has a 16-bit word format. The bits are numbered 0 to 15, where bit 15 is the most significant and bit 0 the least significant.

octal format

The ND-110 16-bit word is represented by a 6 digit octal code. The use of octal is related to the architecture of the ND-100 family, so instructions and registers are quoted as:

SWAP octal instruction code 144000_8
STS octal status register code 000001_8

binary format

Often when analysing what happens to a register or what certain parts of an instruction do, it is easier to look at the word in its binary format, where the value of a bit as 1 or 0 is an important feature, for example, in the status register, bit 7 is the carry flag (*C*).

Data and instruction types

The ND-110 instruction set handles the following data and instruction types:

- bit
- byte (8 bits)
- word (16 bits)
- double word (32 bits)
- floating point words (32 and/or 48 bit) †

† depends upon CPU version (see page 72)

bit

Bit instructions specify operations on any bit in any of the general (A,B,D,L,P,STS,T,X) registers.

byte
(8 bits)

Bytes (occasionally described in other manuals as half words) are used for byte operations. If two bytes are packed into a word for byte addressing, the even byte address points to the most significant half of the word.

Numeric range: 0 to 255_{10}

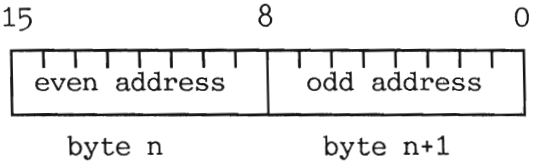


Figure 1. Byte addressing

word
(16-bits)

The ND-110 uses 16-bit addresses and data words. Data words can represent negative numbers, by using 2's complement notation.

Numeric range: -32768_{10} to 32767_{10} (signed)

or 0_{10} to 65535_{10} (unsigned)

double word
(32 bits)

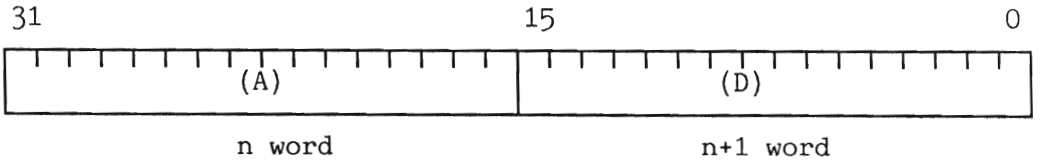


Figure 2. Double word structure

A double word is a 32-bit number occupying two consecutive memory locations (n and n+1). A double word is always referred to by the address of its most significant part; the most significant word being transferred to the A register when used and the least significant to the D register.

Numeric range: -2147483648_{10} to 2147483647_{10}

floating point words (32-bits)

The 32-bit floating word has the following format:

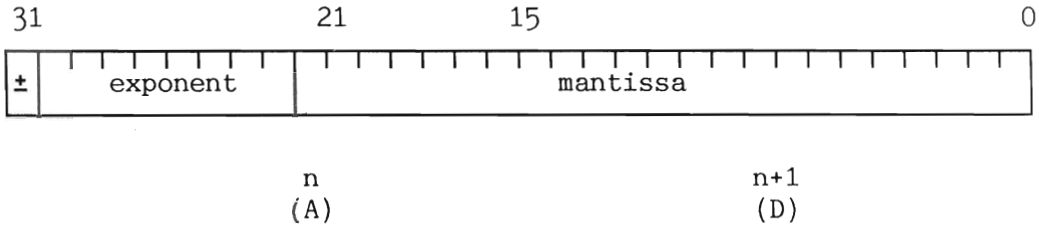


Figure 3. 32-bit floating point word structure

The 32-bit word occupies two consecutive 16-bit memory locations, such that address n provides the sign, exponent and 6 most significant bits of the mantissa while address n+1 provides the lower 16 bits of the mantissa. The two words are operated on in the floating bit accumulator (A and D registers).

The exponent consists of 9 bits: the most significant bit is the complement of the sign and the remaining 8 bits the exponent value (-256 to 256).

The mantissa is normalised to lie from 0.5 to approximately 1; the decimal point is one place to the left of the mantissa. The exponent is biassed with 2^8.

mantissa: $0.5 \equiv m < 1$

range: $10^{-76} < x < 10^{76}$

accuracy: 23 bits (approximately 7 decimal places)

floating zero: 0 in all 32 bits

Examples:

Table 1. Examples of 32-bit floating point numbers

integer	octal	
	A word	D word
0	000000_8	000000_8
+1	040100_8	000000_8
-1	140100_8	000000_8
+3	040240_8	000000_8

48-bit floating point

The 48-bit floating point word has the following format:

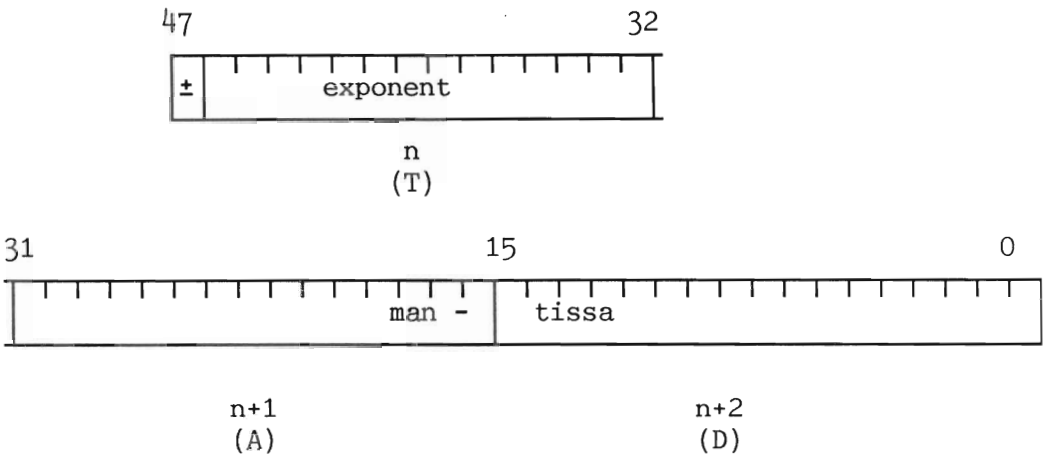


Figure 4. 48-bit floating point word structure

Here the floating point data word occupies three consecutive locations in memory. Address n holds the single bit sign and the 15-bit exponent value, address n+1 the most significant part of the mantissa and address n+2 the least significant. For operations, the three words become the A,D and T registers respectively and are defined as the floating point accumulator.

Range: $10^{-4920} < x < 10^{4920}$

Accuracy: 32 bits (approximately 10 decimal digits)

Floating Zero:0 in all bits

Examples:

Table 2. Examples of 48-bit floating point numbers

integer	octal T word	octal A word	octal D word
0	000000_8	000000_8	000000_8
+1	040001_8	100000_8	000000_8
-1	140001_8	100000_8	000000_8
+3	040240_8	100000_8	000000_8

Decimal notation

BCD - Binary Coded Decimal

Decimal digits are represented in binary-coded decimal (BCD), sometimes known as packed decimal.

Four bits are used to represent a decimal digit:

Table 3. BCD notation

binary notation msb			lsb	decimal equivalent
0	0	0	0	1
0	0	0	1	2
0	0	1	0	3
0	0	1	1	4
0	1	0	0	5
0	1	0	1	6
0	1	1	0	7
0	1	1	1	8
1	0	0	0	9
1	0	0	1	10
1	0	1	0	+
1	0	1	1	-
1	1	0	0	+ †
1	1	0	1	- †
1	1	1	0	+
1	1	1	1	(+)

(+) represents unsigned, it is treated as a plus.

† The ND-110 instruction set uses only the codes 1100 for plus and 1101 for minus.

The maximum length of an operand is 31 decimal digits plus a sign nibble (4 bits), this occupies eight consecutive memory locations (eight 16-bit words).

ASCII coded decimal

ASCII-coded decimal notation uses eight bits to represent a decimal digit.

The format of an ASCII code decimal is :

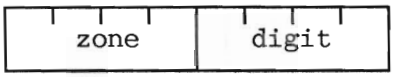


Figure 5. ASCII byte structure

Table 4. ASCII notation

ASCII Code msb lsb	Decimal Equivalent
0 0 1 1 0 0 0 0	0
0 0 1 1 0 0 0 1	1
0 0 1 1 0 0 1 0	2
0 0 1 1 0 0 1 1	3
0 0 1 1 0 1 0 0	4
0 0 1 1 0 1 0 1	5
0 0 1 1 0 1 1 0	6
0 0 1 1 0 1 1 1	7
0 0 1 1 1 0 0 0	8
0 0 1 1 1 0 0 1	9

Bit 7 (msb) is the *parity bit* and is always zero in ASCII code.

sign representation: The ASCII notation for sign is as follows:

+	00101011	53_8
-	00101101	55_8

There are four ways of representing the sign in a decimal operand:

separate trailing The byte following the last significant digit contains the sign.

separate leading The byte preceding the ASCII digit code contains the sign.

embedded trailing The byte representing the least significant decimal digit also contains the sign.

embedded leading The byte representing the most significant digit also contains the sign.

embedded sign coding The embedded codes are represented by ASCII notation as follows:

Table 5. ASCII embedded notation

decimal operand	positive sign ASCII value		negative sign ASCII value	
	octal	binary	octal	binary
0	173	01111011	175	01111101
1	101	01000001	112	01001010
2	102	01000010	113	01001011
3	103	01000011	114	01001100
4	104	01000100	115	01001101
5	105	01000101	116	01001110
6	106	01000110	117	01001111
7	107	01000111	120	01010000
8	110	01001000	121	01010001
9	111	01001001	122	01010010

CHAPTER 2 MEMORY ADDRESSING

The ND-110 accesses memory as 16-bit words. There are four different types of memory access.

1. **Instruction fetch.** The word being fetched will be interpreted as an instruction.
2. **Operand read.** The word being fetched will be used as data.
3. **Operand write.** The word being written is data.
4. **Indirect address fetch.** The word being fetched will be treated as an address for the current operation.

The ND-110 uses relative addressing. This means that the address is specified relative to the contents of the program counter (P register), or relative to the contents of the B and/or X registers.

The following pages detail the various addressing modes available on the ND-110 (including byte addressing and direct physical memory addressing). Each addressing mode is given its own page and headed by its title and bit format. These pages are preceded by a general description of the instruction format and the terminology used.

2.1 Address structure

A large group of memory reference instructions share the same format:

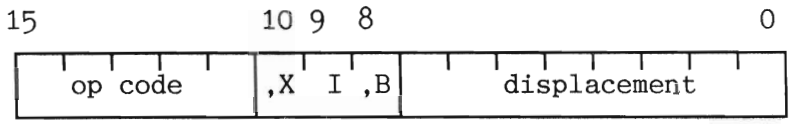


Figure 6. Memory reference instruction format

Bits 8 to 10 define the addressing mode and bits 0 to 7 the displacement. Together these two fields define the memory address.

The 8-bit displacement field is a 2's complement signed number (giving a displacement range of +127 to -128).

The five most significant bits, the op code, define the type of operation executed.

The eight possible combinations of ",X", "I" and ",B" give the following address modes:

- P relative addressing
- B relative addressing
- P indirect addressing
- B indirect addressing
- X relative addressing
- B indexed addressing
- P indirect indexed addressing
- B indirect indexed addressing

Byte addressing

This is a special type of address mode used to manipulate character strings within memory. It is described after the relative addressing modes.

Physical memory addressing

This address mode is used to address a memory location within the physical memory without using the memory management system (for memory addresses > 2000000_8). Its description follows byte addressing.

Execution times

When indirect addressing is used, the execution time of a memory reference instruction increases. One extra microcycle is needed if the indirect address is found in cache; if not, the extra time is the length of a memory access.

When B relative indexed addressing (,X,B) is used the instruction execution time is increased by one microcycle. However, this does **NOT** apply to B indirect indexed addressing (,X I,B).

Memory management

Addressing modes are described in this manual in reference to their 16-bit virtual address, this is normally translated to a 24-bit physical address by the memory management system (extended mode). Older programs may use a 19-bit physical address (normal mode).

When memory management is ON, the translation of a 16-bit address to a 24-bit physical address is done with the help of the normal page table (PT) or alternate page table (APT). The rule is: P relative addressing uses PT and B relative or indexed (X) addressing use APT addressing modes.

Indirect(I) addressing results in two memory accesses. One for the indirect address and the second for the instruction operand itself. The memory management system regards these two accesses as separate operations and chooses PT or APT modes, according to the above rule, for each memory access.

2.2 Addressing modes

Addressing mode notation

The following symbols are used in the description of the ND-110 addressing modes:

,X	address relative to X register (post-indexed)
I	indirect address
,B	address relative to B register (pre-indexed)
d	displacement (bits 0-7 of instruction) as a 2's complement value
( )	contents of
ea	effective address
n	arbitrary address of a word in memory
K	memory-block base-address pointer
*	current value of the program counter
←	points to
>	loaded into
PT	normal page table
APT	alternate page table

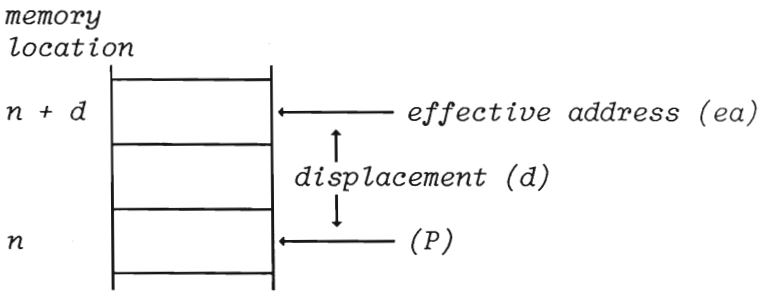
Note: The **effective address** is the term given to the memory location which is finally accessed after all address modification (pre- and post- indexing) has taken place.

Addressing modes

P relative addressing
,X=0
I=0
,B=0

Effective address: ea = (P) + displacement

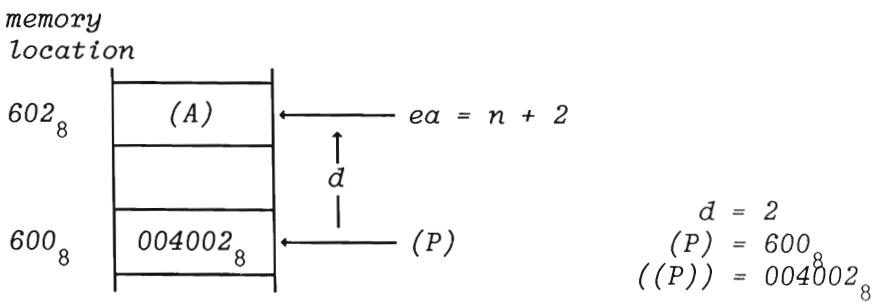
Description: The effective memory address is calculated by adding the value of the displacement to the contents of the P register (program counter). If memory management is being used, the normal page table (PT) will be used.



Note: d may have any value in the range -128 to 127.

Example: STA *2 (instruction code 004002_8)

Store contents of A register in the memory location two words ahead of this instruction.

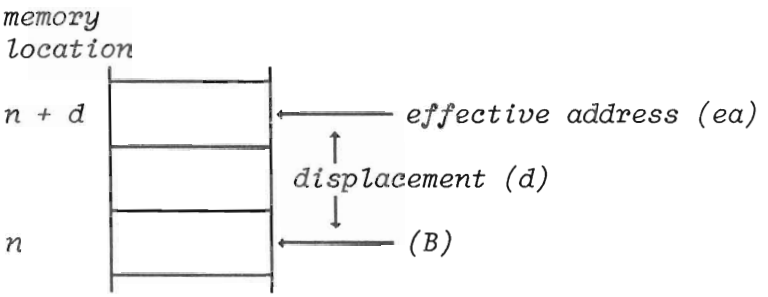


B relative addressing
,X=0
 I=0
,B=1

Effective address: ea = (B) + displacement

Description: The effective address is calculated by adding the value of the displacement vector to the contents of the B register.

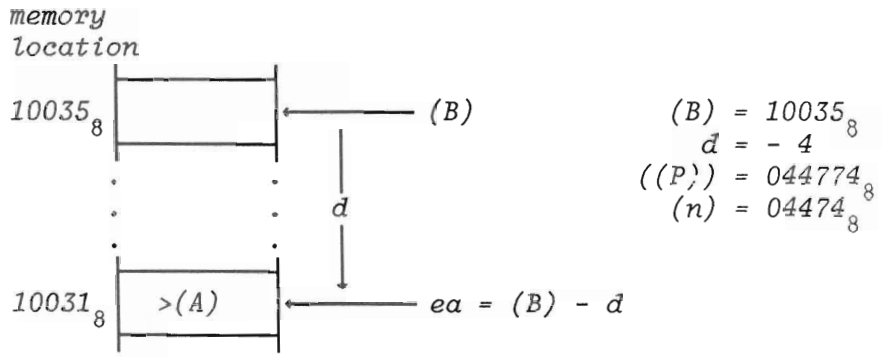
If memory management is ON, the alternate page table (APT) converts the effective address to a physical address.



Note: d may have any value in the range -128 to 127.

Example: LDA -4,B (instruction code 044774_8)

Load the contents of a memory location into the A register.The effective address location is the contents of the B register minus the displacement value (= 4).

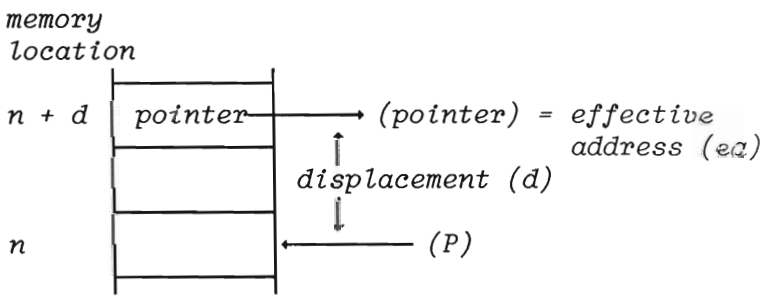


P indirect addressing
,X=0
I=1
,B=0

Effective address: ea = ((P) + displacement)

Description: The contents of the P register (program counter) are added to the value of the displacement to find the indirect address (pointer). If memory management is ON, the standard page table (PT) converts the indirect address to a physical address.

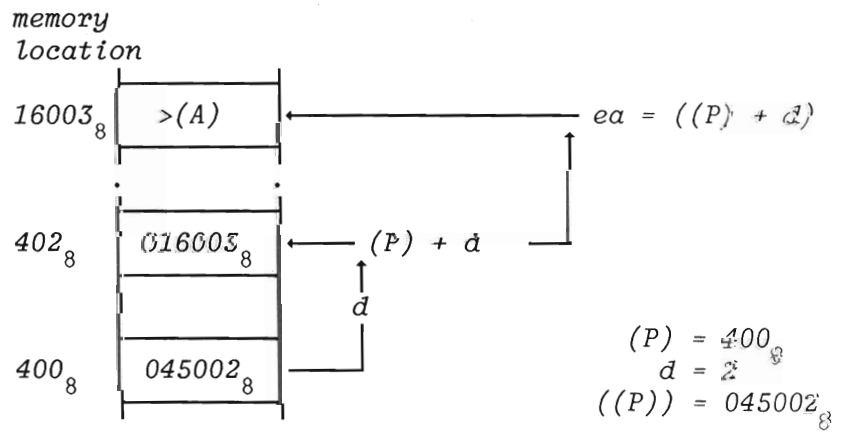
The 16-bit word pointed to by the indirect address is the effective address for the operation. If memory management is ON, the alternate page table (APT) converts the effective address to a physical address.



Note: d may have any value in the range -128 to 127.

Example: LDA I *2 (instruction 045002_8)

Load the contents of the effective address into the A register. The effective address is the contents of the memory location two words (d = 2) ahead of the current instruction.



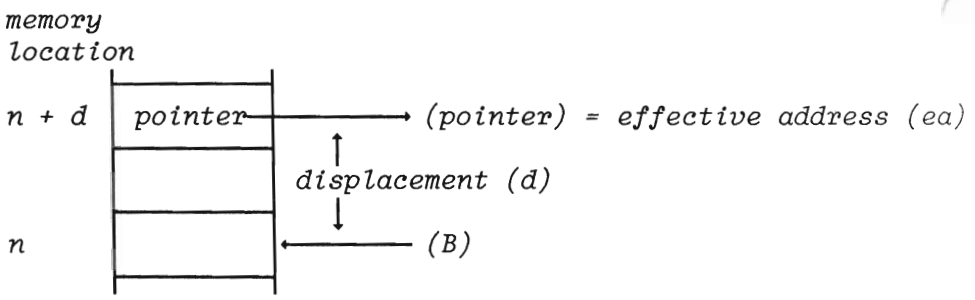
B indirect addressing
,X=0
 I=1
,B=1

Effective address: ea = ((B) + displacement)

Description: The contents of the B register are added to the displacement value. The resulting 16-bit value is the indirect address.

The 16-bit word fetched from this location is the effective address for the operation. If memory management is ON, the alternate page table (APT) will be used to convert both the indirect and effective addresses to physical addresses.

NOTE: Indirect addressing adds one extra memory access to the execution time of the instruction.

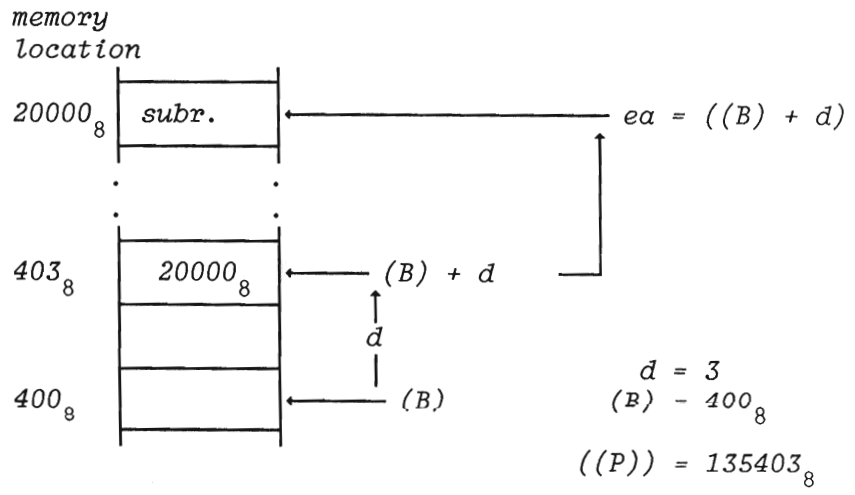


Note: d may have any value in the range -128 to 127.

Example: JPL I 3,B (octal code for instruction 135403_8)

The contents of the B register plus the value of the displacement point to the memory location which contains the effective address.

The instruction saves the contents of the P register (program counter) in the L register and loads the P register with the effective address. This results in the next instruction (marked subr. in the diagram below) being fetched from the effective address.

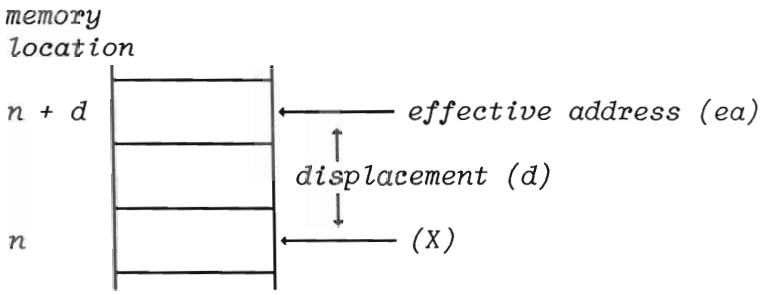


X relative addressing
,X=1
I=0
,B=0

Effective address: ea = (X) + displacement

Description: The effective address is calculated by adding the value of the displacement to the contents of the X register.

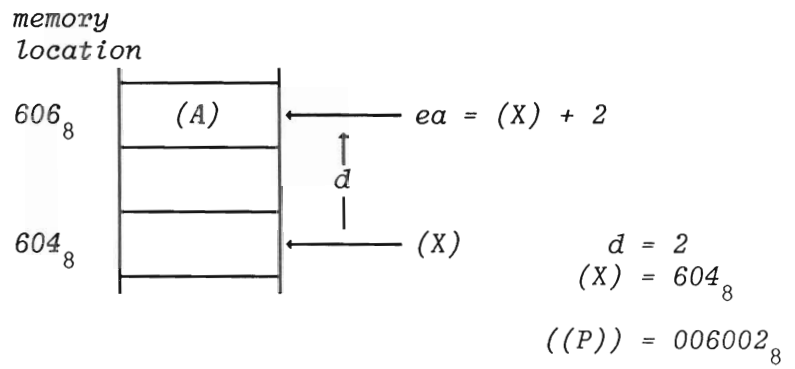
If memory management is being used, the alternate page table (APT) is used to convert the effective address to a physical address.



Note: d may have any value in the range -128 to 127.

Example: STA 2,X (instruction code 006002_8)

Store contents of X register in the memory location two words ahead of this instruction.



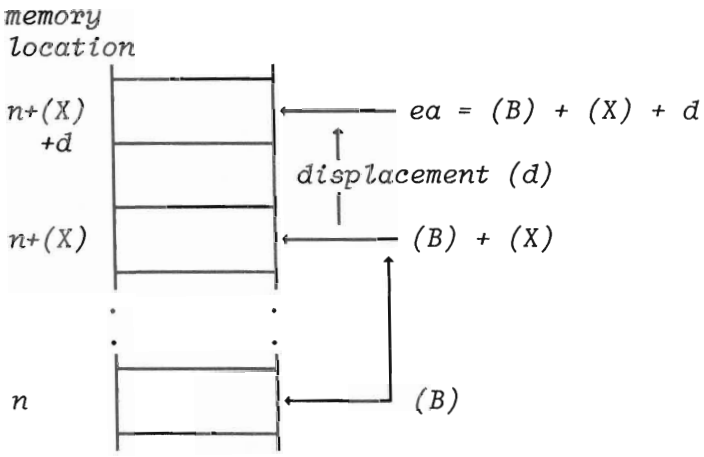
B indexed addressing
,X=1
 I=0
,B=1

Effective address: ea = (B) + (X) + displacement

Description: The effective address is calculated by adding the contents of the B register to the contents of the X register, and then adding the result to the value of the displacement.

If memory management is being used, the alternate page table (APT) will be used to convert the effective address to a physical addresses.

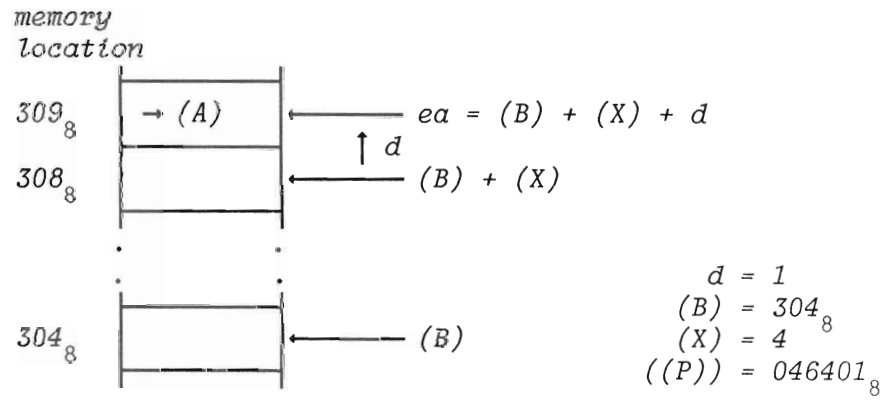
NOTE: This addressing mode adds one extra microcycle to the execution time of the instruction.



Note: d may have any value in the range -128 to 127.

Example: LDA 1,B ,X (instruction code 046401_8)

Load the contents of the memory location into the A register.The effective address is the contents of the B and X registers added together plus the displacement(= 1).



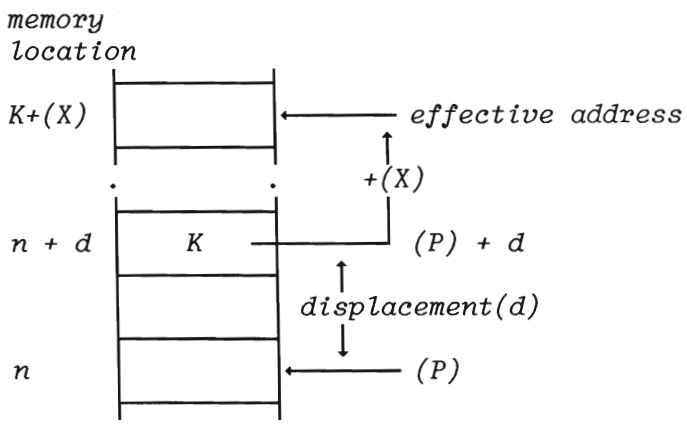
P indirect indexed addressing
,X=1
I=1
,B=0

Effective address: ea = ((P) + displacement) + (X)

Description: The displacement value is added to the contents of the P register to determine an indirect address. The 16-bit word at this location is added to the contents of X (index) register to find the effective address. The indirect address can be used as a base pointer to a block of memory with (X) the index.

If memory management is being used, the alternate page table (APT) will be used to convert the effective address to a physical addresses.

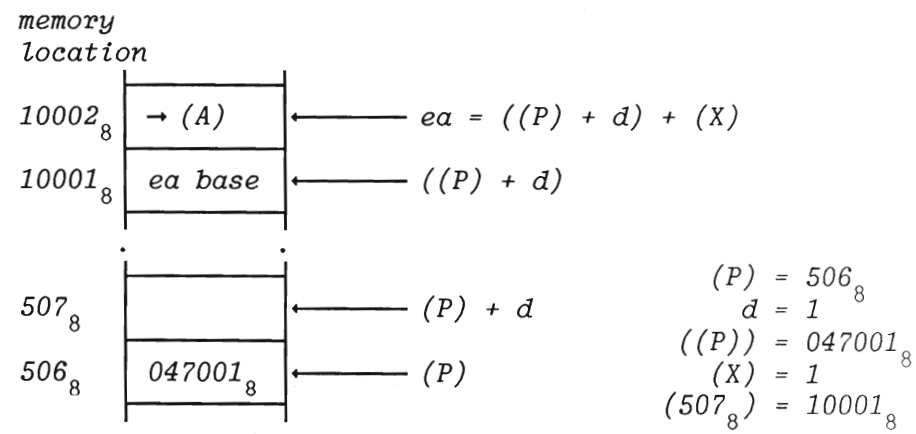
NOTE: Indirect addressing adds one extra memory access to the execution time of the instruction.



Note: d may have any value in the range -128 to 127.

Example: LDA ,X I *1 (instruction code 047001_8)

The contents of the P register (program counter) are added to the value of the displacement (= 2) and the value fetched is used as the effective address. The contents of the effective address are loaded into the A register.



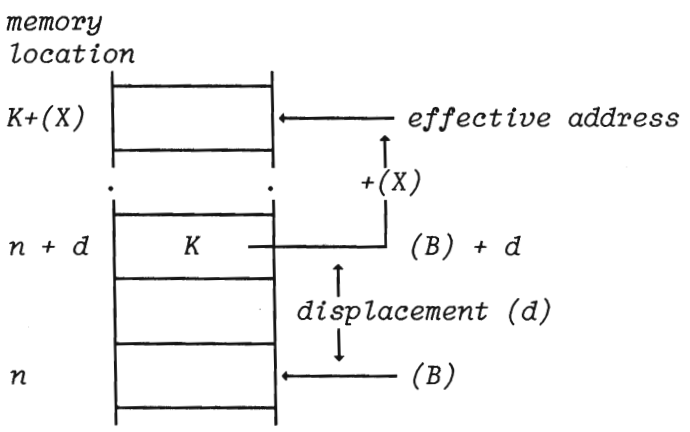
B indirect indexed addressing
,X=1
 I=1
,B=1

Effective address: ea = ((B) + displacement) + (X)

Description: The displacement value is added to the contents of the B register to determine an indirect address. The 16-bit word at this location is added to the contents of X (index) register to find the effective address. The indirect address can be used as a base pointer to a block of memory with (X) the index.

If memory management is being used, the alternate page table (APT) will be used to convert the effective address to a physical address.

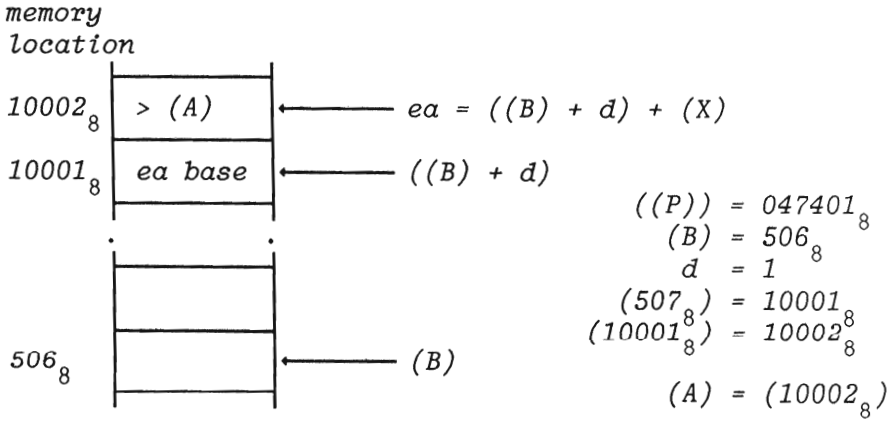
NOTE: Indirect addressing adds one extra memory access to the execution time of the instruction.



Note: d may have any value in the range -128 to 127.

Example: LDA ,X I ,B *1 (instruction code 047401_8)

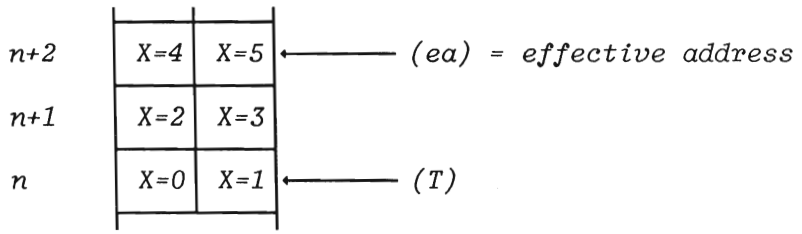
Load the contents of the memory location into the A register.The memory location pointed to by the contents of the B register (program counter) plus a displacement of two gives an intermediate memory location containing the base effective address of a block of data each word located by the index (X register) contents.



Effective address: ea = (T) + (X)÷2

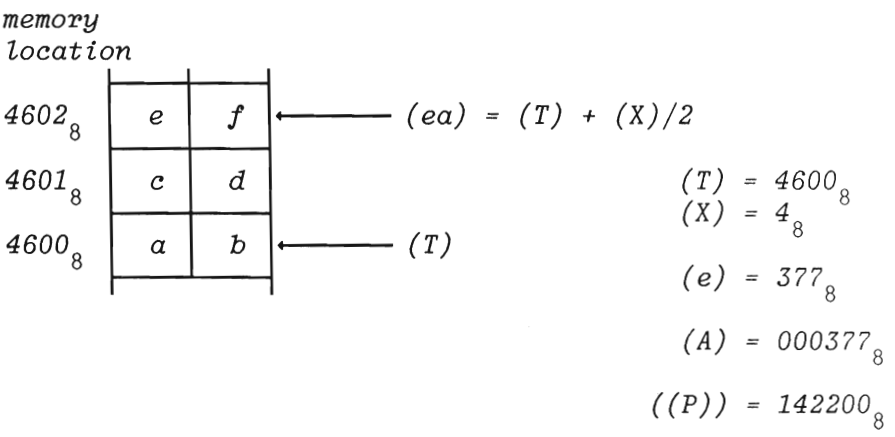
Description: Byte instructions use bytes within memory, these are addressed by the T and X registers.

The T register contents point to the start of a character string in memory and the contents of the X register point to a byte within the string.



Example: LBYT (instruction code 142200_8)

Load the byte addressed by the contents of the T and X register into the lower byte of the A register; set the higher byte to zero.



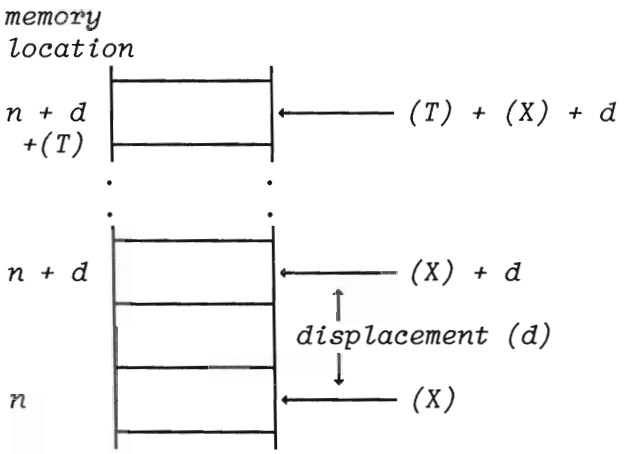
Physical memory addressing

Effective address: ea = (T) + (X) + displacement

Description: There are seven privileged instructions (see pages 133-136) which read/write to any physical memory location whether the memory management system is enabled or not (paging on/off). However, they will affect the page tables if the address is within page-table range.

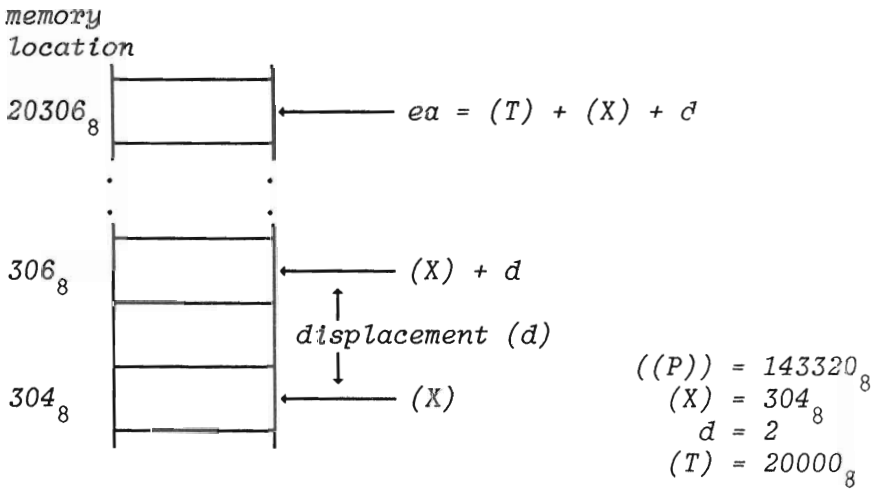
The effective address is calculated by adding a 3-bit displacement value to the T and X register contents.

The displacement is added to the X register first. If this results in a carry, the carry is dropped and NOT added to the T register. Hence, the T register always determines which 64 K memory area to address.



Note: d may have any value in the range 0 to 7.

Example: LDATX (instruction code 143320_8)



CHAPTER 3 THE INSTRUCTION SET

The range of instructions that can be executed by the ND-110 is **the instruction set**. It includes operations on data, varying from bits to triple words; BCD, floating point, arithmetical and logical operations and system-control functions.

This chapter gives a brief explanation of instruction execution and timing, followed by a detailed description of the instruction set. Addressing modes are described in the previous chapter.

Instructions are listed in alphabetical groups according to the type of operation. Each instruction mnemonic is highlighted on the relevant page edge to help you scan through the chapter for a specific instruction. Each page has a general heading to the type of operation.

The instruction set is preceded by an alphabetical index of mnemonics and a key to the notation used.

3.1 Instructions

3.1.1 Using privileged instructions

The instruction set can be subdivided into two instruction types:

- privileged
- user

The privileged instructions are used by the operating system and RT programs only.

Privileged instructions execute all I/O transfers, control memory management and interrupt systems, and enable inter-program level communication.

A user executes the instruction-set subset which excludes privileged instructions; the instruction MON providing the only source of user-operating system communication.

3.1.2 How an instruction is executed

Each instruction in the ND-110 has a corresponding microprogram sequence (a set of micro-instructions) in the microprogram control store. The instruction code is decoded in RMIC gate array to find which microprogram is to be run. Instructions are loaded into cache memory improving execution time of repetitive instructions, as they can be fetched from the local cache each time instead of memory.

The next instruction is fetched during the last microcycle of the current instruction. This differs from the procedure followed in the ND-100 where the next instruction was prefetched during the current instruction. The ND-110 does not lose speed by fetching instructions in the last microcycle, as they are normally fetched from cache memory where they have been partly decoded.

The program counter (P register) points to the instruction address, if the instruction is fetched from memory and the memory management system (MMS) is on, the 16-bit virtual address will be converted to a 24-bit physical address and a 16-bit instruction fetched from the address. If MMS is off, the 16-bit program counter is the address of the instruction.

Detailed information on how the microprogram is decoded is described in the ND-110 Functional Description (ND-06.026).

3.1.3 How to change the microprogram

The microprogram control store is writable and can be set dynamically using two instructions TRR CS and TRA CS (new to the ND-110 instruction set).

3.1.4 Instruction timing

The shortest instructions are executed from cache and use 1 microcycle (compared to the 4 cycles needed in the ND-100).

1 microcycle = 1 internal CPU cycle
= 6+ nanocycles (6 * 26ns) for ND-110
= 4+ nanocycles (4 * 26ns) for ND-110/CX

3.1.5 Alphabetic index of the instruction set

* denotes privileged instruction

3.1.6 Instruction set notation

The following symbols are used in the description of the ND-110 instruction set:

( )	contents of
⟵	becomes e.g. a ⟵ b is a becomes b
¯	1's complement/invert e.g. $\bar{A}$ invert the bits in register A
ea	effective address (see previous chapter)
$_8$	base 8 - octal
$_{10}$	base 10 - decimal
*	assembler mnemonic for the P register
PT	normal page table
APT	alternative page table
•	unused bit - value insignificant

3.2 The instructions

These instructions specify operations between source (sr) and destination (dr) registers.

Instruction format:

<register operation> <sub-instruction (s)> <sr> <dr>

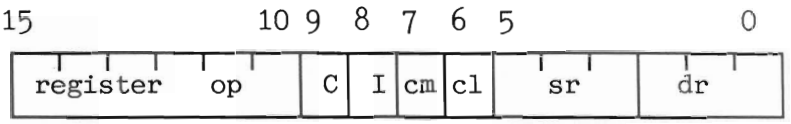
OR

<register operation> <sr> <dr>

OR

<register operation> <dr>

Instruction structure:



cm is CM1
cl is CLD

Source and destination specification: (bits 0-5)

The source and destination register contents provide the operands for these instructions. The result of the instruction is loaded into the the dr register; the sr register contents are unchanged.

	as source:		as destination:	
register	mnemonic	code	mnemonic	code
D	SD	10_8	DD	1_8
P	SP	20_8	DP	2_8
B	SB	30_8	DB	3_8
L	SL	40_8	DL	4_8
A	SA	50_8	DA	5_8
T	ST	60_8	DT	6_8
X	SX	70_8	DX	7_8

Note:

1. If sr is not specified, sr is assumed to be 0.

2. If dr = 0, a no-operation normally occurs. (EXIT,EXR,RDIV,MIX3 are exceptions to this and RADD instructions clear the carry flag, *C* only.)

3. If the P register is specified as either sr or dr, the value of the next instruction is used as an operand.

Sub-instruction specification (bits 6-10)

The following sub-instructions are selected by setting the relevant bit(s):

CLD
bit 6

CLD=1
Zero is used instead of the destination register as an operand (dr register contents are unchanged).

CM1
bit 7

CM1=1
The 1's complement of the source register is used as an operand (sr register contents are unchanged).

ADC

C=0 I=1 Add previous carry to destination register.

AD1

C=1 I=0 Add 1 to destination register.

ADC and AD1 are only valid for RADD instructionss and those combined mnemonics which replace certain RADD *<sub-instruction (s)>*, that is, COPY, RDCR, RINC and RSUB.

ADC and AD1 cause a no-operation when used together in the same instruction.

CM2

This compound sub-instruction is equivalent to CM1 AD1

Comment:
Sub-instructions can make your code less clear, some register operations have combined mnemonics to help.

Flags affected: The carry (*C*) and overflow (*O* and *Q*) are only affected by RADD instructions (and the combined mnemonics which replace certain RADD instructions).

Octal coding: The instructions are quoted with their base octal code, that is the code for the operation without any sub-instruction, sr or dr codes, these should be added accordingly.

Register Instructions
Description

The following instructions are register operations divided into valid format groups:

Format *<register op> <sub-instruction(s)> <sr> <dr> :*
OR *<register op> <sr> <dr> :*
OR *<register op> <dr> :*

Description: register copy

dr ⟵ sr

Format: COPY *<sub-instruction(s)> <sr> <dr>*

Octal code: 146100_8

Optional sub-instructions: COPY is a compound mnemonic for RADD CLD.

The following sub-instructions are allowed:

CM1, CM2, ADC, AD1

Note:
Using ADC and AD1 in the same instruction generates a no-operation.

RCLR — This compound mnemonic represents the COPY instruction: COPY 0 and is used with *<dr>* to clear a specific register (base octal code 146100_8).

EXIT — This compound mnemonic represents the specific COPY instruction: COPY SL DP and causes the return from a subroutine by copying the stored return address into the program counter (P register). It has a unique instruction code 146142_8.

Flags affected: See RADD instruction.

Examples:

1. COPY SA DD (146151_8)

   Copy the contents of register contents A into the D register.

2. COPY CM2 SA DD (146655_8)

   2's complement the A register.
   (Equivalent to RADD CM1 ADC SA DA.)

Register Instructions
EXIT

Description: return from subroutine

EXIT is a compound mnemonic for COPY SL DP (or RADD CLD SL DP).

Format: EXIT

Octal code: 146142_8

Flags affected: See RADD instruction.

Example: EXIT (146142_8) [(L)=108_8 (P)=190_8]

Leave subroutine at location 190_8 and return to location 108_8.

Description: register execute

Format: EXR <*sr*>

Registers affected: (IR) and those registers changed by the instruction.

Octal code: 140600_8

Comments: The contents of the sr register are executed as the next instruction.

If sr contains a memory reference instruction, the address is given as part of the instruction.

Flag affected: EXR <*sr*> cannot be used to fetch an sr register containing another EXR <*sr*> instruction.
If you attempt this the error flag (*Z*) is set.

Examples:

1. EXR SA (140650_8) [(A) = 014177_8 ≡ STX *177_8]

   Execute the instruction held in the A register. Store the X register contents in the memory location pointed to by the program counter plus 177_8.

2. EXR SB (140630_8) [(B) = 134020_8 ≡ JPL 20_8]

   Execute the instruction held in the B register. The instruction is a jump to a subroutine at memory location ((P) + 20_8). The return address (the address of the instruction after EXR; returned to once the subroutine has been completed is held in the L register.

Register Instructions
MIX3

Description: multiply index by 3

(X) ⟵ [(A)-1] x 3

Format: MIX3

Registers affected: (X)

Octal code: 143200_8

Example: MIX3

Take the contents of the A register as an operand and subtract one. Multiply the result by three and place it in the X register.

Description: register add

dr ⟵ dr + sr

Format: RADD *<sub-instruction(s)> <sr> <dr>*

Octal code: 146000_8

Optional sub-instructions:

CLD CLD=1 Zero is used instead of the destination register as an operand (dr register contents are unchanged).

CM1 CM1=1 The 1's complement of the source register is used as an operand (sr register contents are unchanged).

ADC This mnemonic represents C=0 I=1 :
Add previous carry to destination register.

AD1 This mnemonic represents C=1 I=0 :
Add 1 to destination register.

CM2 This compound sub-instruction is equivalent to CM1 ADC.

Flags affected: RADD instructions affect the carry (*C*) and overflow (*O* and *Q*) flags as follows:

C = 1
If a carry occurs from the signed bit positions of the adder.

O = 1 ; *Q* = 1
If an overflow occurs, that is if the signs of the two operands are equal and the sign of the result is different.

O = 1 ; *Q* = 0
If overflow does not exist, the dynamic overflow flag (*O*) is reset while the static overflow flag (*Q*) is left unchanged.

Instruction combinations:

	RADD <*sr*> <*dr*>	dr ⟵ dr + sr
COPY	RADD CLD <*sr*> <*dr*>	dr ⟵ sr ≡ COPY
	RADD CM1 <*sr*> <*dr*>	dr ⟵ dr + $\overline{sr}$
	RADD CM1 CLD <*sr*> <*dr*>	dr ⟵ $\overline{sr}$
	RADD AD1 <*sr*> <*dr*>	dr ⟵ dr + sr + 1
	RADD CLD AD1 <*sr*> <*dr*>	dr ⟵ sr + 1
RSUB	RADD CM1 AD1 <*sr*> <*dr*>	dr ⟵ dr - sr ≡ RSUB
	RADD CM1 CLD AD1 <*sr*> <*dr*>	dr ⟵ - sr
	RADD ADC <*sr*> <*dr*>	dr ⟵ dr + sr + c
	RADD CLD ADC <*sr*> <*dr*>	dr ⟵ sr + c
	RADD CM1 ADC <*sr*> <*dr*>	dr ⟵ dr + $\overline{sr}$ + c
	RADD CM1 CLD ADC <*sr*> <*dr*>	dr ⟵ $\overline{sr}$ + c

Note:

RINC — RADD AD1 <*dr*> is equivalent to RINC <*dr*>, increment dr register by one.

RDCR — RADD CM1 <*dr*> is equivalent to RDCR <*dr*>, decrement dr register by one.

RCLR — RADD CLD 0 (COPY 0) is equivalent to RCLR <*dr*>, register clear.

EXIT — RADD CM1 SL DP (COPY SL DP) is equivalent to EXIT, return from subroutine.

Examples:

1. RADD SA DX (146057_8)

   Add the contents of the A and X registers together and place the result in the X register.

2. RADD CLD SX DB (146173_8) ≡ COPY SX DB

   Use zero as the destination operand, add the X register contents and leave the result in B. THAT IS copy the contents of the X register into A.

3. RADD CM1 CLD AD1 SX DB (146773_8) ≡ COPY CLD AD1 or RSUB CM1

   Copy the negative value of the X register contents into B.

Description: AND register

dr ⟵ dr AND sr

Format: RAND *<sub-instruction(s)>* *<sr>* *<dr>*

Octal code: 144400_8

Optional sub-instructions:

CLD=1 Zero is used instead of the destination register as an operand (dr register contents are unchanged).

CM1=1 The 1's complement of the source register is used as an operand (sr register contents are unchanged).

Instruction combinations:

RAND *<sr>* *<dr>*	dr ⟵ dr AND sr
RAND CLD *<sr>* *<dr>*	dr ⟵ 0
RAND CM1 *<sr>* *<dr>*	dr ⟵ dr AND $\overline{\text{sr}}$
RAND CM1 CLD *<sr>* *<dr>*	dr ⟵ 0

Examples:

1. RAND SL DX (144447_8)

   AND the contents of the L and X registers. Store the result in the X register.

2. RAND CM1 ST DB (144663_8)

   AND the contents of the T and B registers, taking the 1's complement of the sr as the source operand.

RCLR

Description: register clear

dr ⟵ 0

RCLR is a compound mnemonic for COPY 0 (or RADD CLD 0).

Format: RCLR <*dr*>

Octal code: 146100_8

Flags affected: See RADD instruction.

Example: RCLR DP (146102_8)

Clear the P register.

Description: register decrement

dr ⟵ dr - 1

RDCR is a compound mnemonic for RADD CM1.

Format: RDCR *<dr>*

Octal code: 146200_8

Flags affected: See RADD instruction.

Example: RDCR DB (146203_8)

Decrement the contents of the B register by one.

Register Instructions
RDIV

Description: register divide

(AD) / sr

(A) ⟵ quotient
(D) ⟵ remainder

Format: RDIV *<sr>*

Registers affected: (A), (D)

Flags affected: *Z, C, O, Q*

Octal code: 141600_8

Comments: The 32-bit signed integer held in the double accumulator AD is divided by the contents of sr.

If division causes overflow, the error flag (*Z*) is set.

The numbers are fixed point integers with the fixed point after the rightmost position.

Example:

Before division:		After division:		
AD	*<sr>*	A	D	*Z*
22_{10}	4_{10}	5_{10}	2_{10}	0
-22_{10}	4_{10}	-5_{10}	-2_{10}	0
378452_{10}	-16_{10}	-23653_{10}	4_{10}	0
32767_{10}	1_{10}	32767_{10}	0_{10}	0
32768_{10}	1_{10}	-	-	1
65535_{10}	2_{10}	32762_{10}	1_{10}	0

Description: XOR register

dr ⟵ dr XOR sr

Format: REXO *<sub-instruction(s)> <sr> <dr>*

Octal code: 145000_8

Optional sub-instructions:

CLD=1 Zero is used instead of the destination register as an operand (dr register contents are unchanged).

CM1=1 The 1's complement of the source register is used as an operand (sr register contents are unchanged).

Instruction combinations:

REXO *<sr> <dr>*	dr ⟵ dr XOR sr
REXO CLD *<sr> <dr>*	dr ⟵ sr
REXO CM1 *<sr> <dr>*	dr ⟵ dr XOR $\overline{sr}$
REXO CM1 CLD *<sr> <dr>*	dr ⟵ $\overline{sr}$

Example: REXO ST DB (145063_8)

Exclusive OR the contents of the B and T register, leaving the result in B.

Description: register increment

dr ⟵ dr + 1

RINC is a compound mnemonic for RADD AD1.

Format: RINC *<dr>*

Octal code: 146400_8

Flags affected: See RADD instruction.

Example: RINC DA (146405_8)

Increment the contents of the A register by one.

Description: register multiply

(AD) ⟵ sr x dr

Format: RMPY *<sr>* *<dr>*

Registers affected: (A), (D)

Flags affected: *C*, *O*, *Q*

Octal code: 141200_8

Comments: The sr and dr registers hold the two operands to be multiplied together. The result is a 32-bit signed integer held in the A and D register (the A register contains the 16 most significant bits).

Example: RMPY SA DX (141257_8)

Multiply the contents of the A and X registers together, leaving the result in the A and D registers.

Description: OR register

dr ⟵ dr OR sr

Format: RORA *<sub-instruction(s)> <sr> <dr>*

Octal code: 145400_8

Optional sub-instructions:

CLD=1 Zero is used instead of the destination register as an operand (dr register contents are unchanged).

CM1=1 The 1's complement of the source register is used as an operand (sr register contents are unchanged).

Instruction combinations:

RORA *<sr> <dr>*	dr ⟵ dr OR sr
RORA CLD *<sr> <dr>*	dr ⟵ sr
RORA CM1 *<sr> <dr>*	dr ⟵ dr OR $\overline{sr}$
RORA CM1 CLD *<sr> <dr>*	dr ⟵ $\overline{sr}$

Examples: RORA ST DB (145463_8)

OR the contents of the B and T registers leaving the result in B.

Description: register subtract

dr ⟵ dr - sr

RSUB is a compound mnemonic for RADD AD1 CM1 (or RADD CM2 using the AD1 CM2 compound mnemonic).

Format: RSUB *<sub-instruction> <sr> <dr>*

Octal code: 146600_8

Optional sub-instructions: The following sub-instruction is allowed:

CLD

Note:
Sometimes the RADD form will be more readable.

Flags affected: See RADD instruction.

Example: RSUB ST DB (146663_8)

Subtract the contents of the T register from the contents of the B register leaving the result in B.

Register Instructions
SWAP

Description: register swap

dr ⟷ sr

Format: SWAP *<sub-instruction(s)> <sr> <dr>*

Octal code: 144000_8

Optional sub-instructions:

CLD=1 Zero is used instead of the destination register as an operand (dr register contents are unchanged).

CM1=1 The 1's complement of the source register is used as an operand (sr register contents are unchanged).

Instruction combinations:

SWAP *<sr> <dr>*	dr ⟷ sr
SWAP CLD *<sr> <dr>*	dr ⟵ sr; sr ⟵ 0
SWAP CM1 *<sr> <dr>*	dr ⟵ $\overline{sr}$; sr ⟵ dr
SWAP CM1 CLD *<sr> <dr>*	dr ⟵ $\overline{sr}$; sr ⟵ 0

Examples:

1. SWAP SA DD (144051_8)

   Exchange A and D register contents.

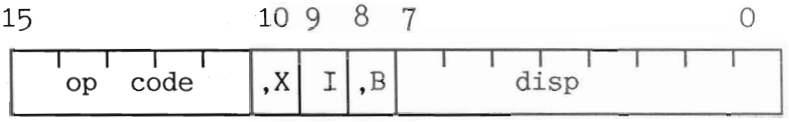
2. SWAP CLD SA DX (144157_8)

   Use zero as the destination operand.
   Exchange the A and X register contents.
   A register contents are zero, X register contents are the previous contents of A.

These instructions specify memory transfers.

Instruction format: *<opcode> <address mode> <disp>*

Instruction structure:



op code: The opcode determines what type of operation occurs.

,X I ,B: These three bits give the addressing mode for the instruction as follows:

,X	I	,B	Effective Address	Address Relative to:	Page ref:
0	0	0	(P) + disp	P	19
0	0	1	(B) + disp	B	20
0	1	0	((P) + disp)	P indirect	21
0	1	1	((B) + disp)	B indirect	22
1	0	0	(X) + disp	X indexed	23
1	0	1	(B) + disp + (X)	B indexed	24
1	1	0	((P) + disp) + (X)	P indirect indexed	25
1	1	1	((B) + disp) + (X)	B indirect indexed	26

disp: displacement

disp This 8-bit signed field gives the memory address displacement. (2's complement notation giving a displacement range of -128 to 127 memory locations.)

LDA

Description: Load A register

(A) ⟵ (ea)

Load the contents of the memory location pointed to by the effective address into the A register.

Format: LDA *<address mode> <disp>*

Octal code: 044000_8

LDD

Description: Load double word

(A) ⟵ (ea)
(D) ⟵ (ea) + 1

Load the contents of the memory location pointed to by the effective address into the A register and the contents of the memory location pointed to by the effective address plus one into the D register.

Format: LDD *<address mode> <disp>*

Octal code: 024000_8

LDF

Description: Load floating point accumulator (32- and 48-bit)

(T) ⟵ (ea)
(A) ⟵ (ea) + 1
(D) ⟵ (ea) + 2

Load the contents of the memory location pointed to by the effective address into the T register, the contents of the effective address plus one into the A register and the contents of the effective address plus two into the D register.

Format: LDF *<address mode> <disp>*

Octal code: 034000_8

LDT

Description: Load T register

(T) ⟵ (ea)

Load the contents of the memory location pointed to by the effective address into the T register.

Format: LDT *<address mode> <disp>*

Octal code: 050000_8

LDX

Description: Load X register

(X) ⟵ (ea)

Load the contents of the memory location pointed to by the effective address into the X register.

Format: LDX *<address mode> <disp>*

Octal code: 054000_8

MIN

Description: Increment memory and skip if zero

(ea) ⟵ (ea) + 1
(P) ⟵ (P) + 2 IF new (ea) = 0

The contents of the memory location pointed to by the effective address are incremented by one. If the new memory location when incremented becomes zero, the next instruction is skipped.

Format: MIN *<address mode> <disp>*

Octal code: 040000_8

STA

Description: Store A register

(ea) ⟵ (A)

Store the contents of the A register in the memory location pointed to by the effective address.

Format: STA *<address mode> <disp>*

Octal code: 004000_8

STD

Description: Store double word

(ea) ⟵ (A)
(ea) + 1 ⟵ (D)

Store the contents of the A register in the memory location pointed to by the effective address; store the contents of the D register in the memory location pointed to by the effective address plus one.

Format: STD *<address mode> <disp>*

Octal code: 020000_8

STF

Description: Store floating point accumulator (32- and 42-bit)

(ea) ⟵ (T)
(ea) + 1 ⟵ (A)
(ea) + 2 ⟵ (D)

Store the contents of the floating accumulator (T, A, and D registers) in the memory locations pointed to by the effective address.

Format: STF *<address mode> <disp>*

Octal code: 030000_8

STT

Description: Store T register

(ea) ⟵ (T)

Store the contents of the T register in the memory location pointed to by the effective address.

Format: STT *<address mode> <disp>*

Octal code: 010000_8

STX

Description: Store X register

(ea) ⟵ (X)

Store the contents of the X register in the memory location pointed to by the effective address.

Format: STX *<address mode> <disp>*

Octal code: 014000_8

Memory Transfer
Store Instructions

STZ

Description: Store zero

(ea) ⟵ 000000_8

Store zero in the contents of the memory location pointed to by the effective address.

Format: STZ *<address mode> <disp>*

Octal code: 000000_8

ADD

Description: Add to A register

(A) ⟵ (A) + (ea)

Add the contents of the memory location pointed to by the effective address to the A register, leaving the result in A.

Format: ADD *<address mode> <disp>*

Octal code: 060000_8

Flags affected:

C = 1
If a carry occurs from the signed bit positions of the adder.

O = 1 ; *Q* = 1
If an overflow occurs, that is if the signs of the two operands are equal and the sign of the result is different.

O = 1 ; *Q* = 0
If overflow does not exist, the dynamic overflow flag (*O*) is reset 0 while the static overflow flag (*Q*) is left unchanged.

MPY

Description: Multipy integer

(A) ⟵ (A) x (ea)

Multiply the contents of the memory location pointed to by the effective address with the contents of the A register, leaving the result in A.

Format: MPY *<address mode> <disp>*

Octal code: 120000_8

Flags affected:

O = 1 ; *Q* = 1
If an overflow occurs, that is if the result has an absolute value greater than 32767_{10}.

SUB

Description: Subtract from A register

(A) ⟵ (A) - (ea)

Subtract the contents of the memory location pointed to by the effective address from the A register contents, leaving the result in A.

Format: SUB *<address mode> <disp>*

Octal code: 064000_8

Flags affected:

C = 1
If a carry occurs from the signed bit positions of the adder.

O = 1 ; *Q* = 1
If an overflow occurs, that is if the signs of the two operands are equal and the sign of the result is different.

O = 1 ; *Q* = 0
If overflow does not exist, the dynamic overflow flag (O) is reset to 0 while the static overflow flag (Q) is left unchanged.

AND

Description: AND memory contents with A register

(A) ⟵ (A) AND (ea)

AND the contents of the memory location pointed to by the effective address with the A register contents, leaving the result in A.

Format: AND *<address mode> <disp>*

Octal code: 070000_8

ORA

Description: OR memory contents with A register

(A) ⟵ (A) OR (ea)

OR the contents of the memory location pointed to by the effective address with the A register contents, leaving the result in A.

Format: ORA *<address mode> <disp>*

Octal code: 074000_8

For 48-bit floating point see note on page 72.

LDF

This loads the floating point accumulator (see LOAD Instructions).

FAD

Description: Add to floating point accumulator

(A) ⟵ (ea) + (T)
(D) ⟵ (ea + 1) + (A)

The contents of two sequential memory locations, pointed to by the effective address, are added to the contents of the floating point accumulator (T and A registers). The result is held in the accumulator.

Format: FAD *<address mode> <disp>*

Octal code: 100000_8

FDV

Description: Divide floating point accumulator

The contents of the floating point accumulator (A and D registers) are divided by the contents of two sequential memory locations, pointed to by the effective address. The result is held in the accumulator.

Flag affected: Division by zero sets the error flag (*Z*). This can be detected by the BSKP instruction (see Bit Instructions).

Format: FDV *<address mode> <disp>*

Octal code: 114000_8

For 48-bit floating point see note on page 72.

FMU

Description: Multiply floating point accumulator

The contents of the floating point accumulator (A and D registers) are multiplied by the contents of two sequential memory locations, pointed to by the effective address. The result is held in the accumulator.

Format: FMU *<address mode> <disp>*

Octal code: 110000_8

FSB

Description: Subtract from floating point accumulator

The contents of two sequential memory locations, pointed to by the effective address, are subtracted from the contents of the floating point accumulator (A and D registers). The result is held in the accumulator.

Format: FSB *<address mode> <disp>*

Octal code: 104000_8

STF

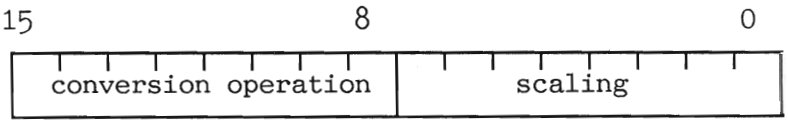
This stores the floating point accumulator (see STORE Instructions).

Floating Point Conversion Instructions
Description

These instructions convert to/from a single precision fixed point number from/to a floating point number.

Instruction format: *<conversion operation> <scaling>*

Instruction structure:



conversion operation: There are two conversion instructions:

NLZ
DNZ

scaling: A scaling factor is given to the conversion of -128 to 127 (approximately 10^{-39} to 10^{39}).

For 48-bit floating point see note on page 72.

DNZ

Description:

Denormalize

The number in the floating point accumulator (A and D registers) is converted to its single precision fixed point equivalent in the A register using the scaling factor given.

When converting to an integer, a scaling factor of -16_{10} should always be used and will give a fixed point number with the same value as the integer part of the floating point number. Other scaling factors will have the same result but the overflow test will be affected.

The D register will be cleared after this instruction.

If the conversion causes underflow, the A and D registers will be set to zero. If overflow occurs (the resulting integer has an absolute value greater than 32767), the error flag (Z) is set to one.

Format:

DNZ *<scaling>*

Octal code:

152000_8

48-bit:

48-bit CPUs allow different scaling factors to be used for DNZ operations. However, the overflow test is only failproof for a scaling factor of 16_{10}.

NLZ

Description: Normalize

The number in the A register is converted to its floating point equivalent in the floating point accumulator (A and D registers), using the scaling factor given.

For integers, a scaling factor of $+16_{10}$ will give a floating point number with the same value as the integer.

The larger the scaling factor, the larger the floating point number.

The D register will be cleared when using single precision fixed point numbers.

Format: NLZ *<scaling>*

Octal code: 151400_8

Comments on 48-bit floating point CPU features:

For the ND-110, 48-bit floating point CPU option, a further register (T) and memory location (ea + 2) are used. In this case, the T register is linked to location ea, A to ea + 1 and D to ea + 2.

How to test for a 32-bit or 48-bit floating point CPU:

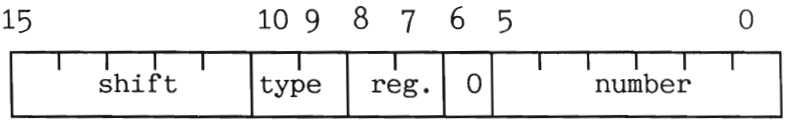
```
SAT 0
SAA 1
NLZ 20₈
```

This tests whether T is changed, if so, the CPU is 48-bit; otherwise it is 32-bit.

These instructions specify register shifts

Instruction format: *<shift register> <type> <mode>*

Instruction structure:



bit 6 is always zero

shift and reg. fields: (bits 15-11, 8 and 7)

Shift operations are allowed on three working registers:

register	mnemonic	octal code
A and D	SAD	154600_8
A	SHA	154400_8
D	SHD	154200_8
T	SHT	154000_8

type: (bits 10 and 9)

Four types of shift can be specified:

bits 10	bits 9	octal code	mnemonic	description:
0	0	000000_8	---	Arithmetic shift
0	1	001000_8	ROT	Rotational shift
1	0	002000_8	ZIN	Zero end input
1	1	003000_8	LIN	Link end input

number field (bits 0-5):

This 6-bit signed field specifies the number of shifts and the shift direction.

bits 0-4 = number of shifts

bit 5 = 1 then shift right (max.32 times)
bit 5 = 0 then shift left (max.31 times)

SHR

This a feature of the assembler. This mnemonic can be used to specify shift right, so that instead of calculating the 2's complement for the number of right shifts required, SHR can be used.

Example:
The instruction which shifts the A register contents three places to the right can be written as:

SHA 75_8

SHA 100_8 - 3_8

SHA SHR 3_8

M flag

Every shift instruction places the last bit discarded in the multi-shift flag (*M*). *M* can be used as an input for the next shift instruction.

M is bit 8 of the STS Register.

SAD

Description:	Shift A and D registers connected Bit 0 of the A register is connected to bit 15 of the D register allowing 32-bit numbers to be shifted.
Format:	SAD *<type> <number>*
Octal code:	154600_8
Flag affected:	*M*

SHA

Description:	Shift A register
Format:	SHA *<type> <number>*
Octal code:	154400_8
Flag affected:	*M*

SHD

Description:	Shift D register
Format:	SHD *<type> <number>*
Octal code:	154200_8
Flag affected:	*M*

SHT

Description:	Shift T register
Format:	SHT *<type> <number>*
Octal code:	154000_8
Flag affected:	*M*

Jump Instructions
Description

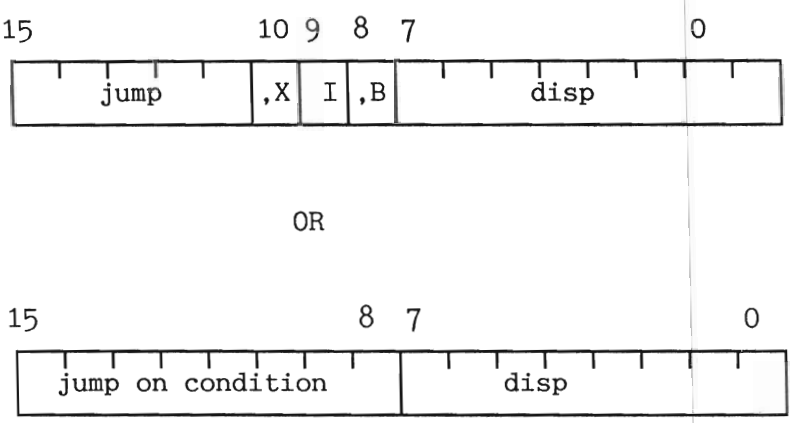
Jump instructions redirect program execution.

Instruction format: *<jump> <address mode> <disp>*

OR

<jump on condition> <disp>

Instruction structure:



jump instructions: These are:

JMP
JPL

and have the following address modes:

,X	I	,B	Effective Address	Address Relative to:	Page ref:
0	0	0	(P) + disp	P	19
0	0	1	(B) + disp	B	20
0	1	0	((P) + disp)	P indirect	21
0	1	1	((B) + disp)	B indirect	22
1	0	0	(X) + disp	X indexed	23
1	0	1	(B) + disp + (X)	B indexed	24
1	1	0	((P) + disp) + (X)	P indirect indexed	25
1	1	1	((B) + disp) + (X)	B indirect indexed	26

disp: displacement

disp These eight bits determine the memory address displacement.
Seven bits give the displacement and the most significant eighth bit the sign (that is, a range of -128 to 127 memory locations).

jump on condition

These jump instructions give conditions for jumping sections of program. The jump is always relative to the program counter (P register).

disp

The eight displacement (disp) bits give a signed range of -128 to 127 locations to be jumped if the condition is true.

Jump Instructions
JMP and JPL

JMP

Description: Jump (unconditional)

(P) ⟵ (ea)

The address of the next instruction is the effective address of the JUMP instruction.

Format: JMP *<address mode> <disp>*

Octal code: 124000_8

JPL

Description: Jump to subroutine (jump link)

(L) ⟵ (P) + 1
(P) ⟵ (ea)

The address of the next instruction is the effective address of the JPL instruction. The value of the program counter contents plus one (the return address) is saved in the L register before the jump takes place.

Format: JPL *<address mode> <disp>*

Octal code: 134000_8

FOR ALL JUMP ON CONDITION TRUE INSTRUCTIONS:

Disp range: -128 to 127 locations

General description: (P) ⟵ (ea)

If condition true, jump to the address of the program counter plus the value of disp.

If condition false, continue program execution at (P) + 1.

JAF

Description: Jump if (A) ≠ 0 (jump if A filled)

Format: JAF *<disp>*

Octal code: 131400_8

JAN

Description: Jump if (A) < 0 (jump if A negative)

If A (bit 15) = 1

Format: JAN *<disp>*

Octal code: 130400_8

JAP

Description: Jump if (A) ≥ 0 (jump if A positive or zero)

If A (bit 15) = 0

Format: JAP *<disp>*

Octal code: 130000_8

JAZ

Description: Jump if (A) = 0 (jump if A zero)

Format: JAZ *<disp>*

Octal code: 131000_8

Jump Instructions
Jump on condition true

JNC

Description: Count and jump if (X) < 0 (jump if negative and count)

(X) ⟵ (X) + 1

THEN

Jump if X (bit 15) = 1

Format: JNC *<disp>*

Octal code: 132400_8

JPC

Description: Count and jump if (X) ≥ 0 (jump if positive and count)

(X) ⟵ (X) + 1

THEN

Jump if X (bit 15) = 0

Format: JPC *<disp>*

Octal code: 132000_8

JXN

Description: Jump if (X) < 0 (jump if X negative)

If X (bit 15) = 1

Format: JXN *<disp>*

Octal code: 133400_8

JXZ

Description: Jump if (X) = 0 (jump if X zero)

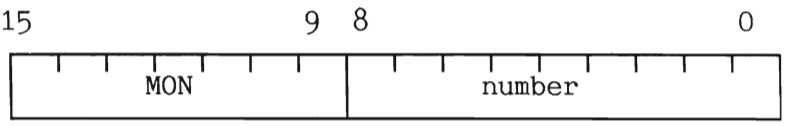
Format: JXZ *<disp>*

Octal code: 133000_8

This instruction is used for monitor calls and causes an internal interrupt to program level 14.

Instruction format: MON *<number>*

Instruction structure:



MON — Monitor instruction mnemonic.

Octal code: 153000_8

number — This unsigned field allows 256 monitor calls.

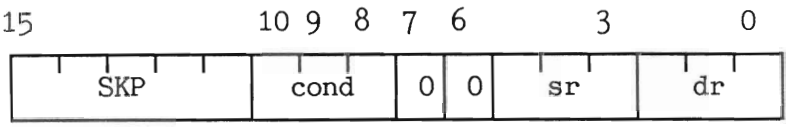
The field is loaded into the T register on level 14. The higher byte of the T register is sign extended.

Skip Instruction
SKP

The next instruction is skipped if a specified condition is true.

Instruction format: SKP *<dr> <cond> <sr>*

Instruction structure:



bits 6 and 7 are always zero

SKP — Skip instruction mnemonic.

Basic octal code: 140000_8

sr and dr specification: (bits 0 - 5)

register	as source: mnemonic	as source: code	as destination: mnemonic	as destination: code
D	SD	10_8	DD	1_8
P	SP	20_8	DP	2_8
B	SB	30_8	DB	3_8
L	SL	40_8	DL	4_8
A	SA	50_8	DA	5_8
T	ST	60_8	DT	6_8
X	SX	70_8	DX	7_8

Note:
If sr not specified, sr is taken to be the value 0.
If dr=0, a no-operation occurs.
If sr or dr are specified as the P register, the value used is that of the next instruction.

Condition codes: (bits 8-10)

SKP can be qualified by eight different mnemonics which the flags set by the following expression:

(dr) - (sr)

Four flags are affected by this calculation:

S sign

Z zero result (error)

C carry

O overflow

The condition codes are as follows:

bits 10 9 8	mnemonic	description	flag(s) condition if true
0 0 0	EQL	Equal	$Z = 1$
0 0 1	GEQ	Greater or equal to †	$S = 0$
0 1 0	GRE	Greater or equal to * †	$\overline{S + O} = 0$
0 1 1	MGRE	Magnitude greater or equal to *	$C = 1$
1 0 0	UEQ	Unequal	$Z = 0$
1 0 1	LSS	Less than †	$S = 1$
1 1 0	LST	Less than * †	$\overline{S + O} = 1$
1 1 1	MLST	Magnitude less than *	$C = 0$

* denotes overflow is taken care of
† denotes contents of sr and dr are treated as signed numbers

Note:
By swapping the sr and dr fields, these relationships can be tested:

> Greater than
≤ Less than or equal

Skip Instruction
SKP

Examples:

1. SKP DD EQL SL
   Skip next instruction if the D register contents equal that of the A register.

2. SKP DB LSS SA
   Skip the next instruction if the contents of the A register are less than the B register contents.

   OR

   Skip the next instruction if the contents of the B register are greater than the A register contents.

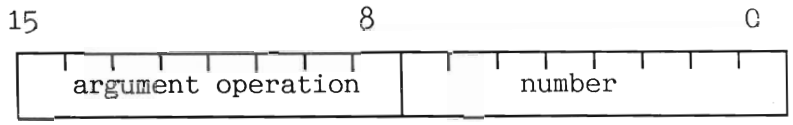
3. SKP DL UEQ
   Skip the next instruction if the contents of the L register do not equal zero.

4. SKP LSS SD
   Skip the next instruction if the D register is less than zero.

These instructions operate on registers.

Instruction format: *<argument operation> <number>*

Instruction structure:



argument operation: There are eight argument instructions:

mnemonic	octal code	description
AAA	AAA	172400_8	add argument to A
AAB	AAB	172000_8	add argument to B
AAT	AAT	173000_8	add argument to T
AAX	AAX	173400_8	add argument to X
SAA	SAA	170400_8	set argument to A
SAB	SAB	170000_8	set argument to B
SAT	SAT	171000_8	set argument to T
SAX	SAX	171400_8	set argument to X

Sign extension: 8-bit argument numbers are extended to 16-bits using sign extension.

The 8-bit argument becomes the least significant byte; the higher byte is extended with ones or zeros. Positive arguments have the higher byte extended with zeros; negative numbers are extended with ones with the argument in 2's complement form.

Examples:

1. SAT 13_8

   Set the T register equal to 13_8. Bits 8-15 are zero due to sign extension.

2. SAB -26_8

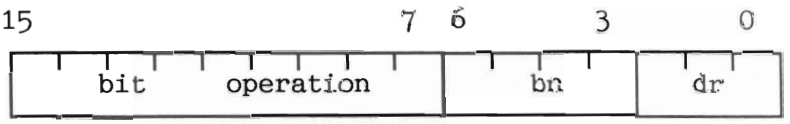
   The contents of the B register are set to 177752_8, bits 8-15 have been extended with ones as the argument is negative; bits 0-7 have the argument in its 2's complement form.

3. AAA 3_8

   Add 3 to the contents of the A register. The contents of bits 8-15 depend on the previous contents of the A register. The carry and overflow flags may also be affected.

These instructions manipulate single bits within the working and STS registers.

Instruction structure:



bit operation: (bits 7-15)

Bits 11-15 are always set to one for a bit operation. Bits 7-10 determine the type of operation as follows:

	type bits 10 9 8 7	mnemonic	description	code
			set bit to:	
BSET	0 0 0 0	BSET ZRO	bit ← 0	174000_8
	0 0 0 1	BSET ONE	bit ← 1	174200_8
	0 0 1 0	BSET BCM	bit ← $\overline{bit}$	174400_8
	0 0 1 1	BSET BAC	bit ← K	174600_8
			skip next instruction if:	
BSKP	0 1 0 0	BSKP ZRO	bit ← 0	175000_8
	0 1 0 1	BSKP ONE	bit ← 1	175200_8
	0 1 1 0	BSKP BCM	bit ← $\overline{bit}$	175400_8
	0 1 1 1	BSKP BAC	bit ← K	175600_8
			for one bit accumulator:	
BSTC	1 0 0 0	BSTC	bit ← K, $\overline{K}$ ← 1	176000_8
BSTA	1 0 0 1	BSTA	bit ← K, K ← 0	176200_8
BLDC	1 0 1 0	BLDC	K ← $\overline{bit}$	176400_8
BLDA	1 0 1 1	BLDA	K ← bit	176600_8
BANC	1 1 0 0	BANC	K ← ($\overline{bit}$ AND K)	177000_8
BAND	1 1 0 1	BAND	K ← (bit AND K)	177200_8
BORC	1 1 1 0	BORC	K ← ($\overline{bit}$ OR K)	177400_8
BORA	1 1 1 1	BORA	K ← (bit OR K)	177600_8

K is the 1 bit accumulator (bit 3 of STS register)

Sub-instructions:

Only the BSET and BSKP instructions have the following qualifying sub-instructions:

ZRO
ONE
BCM
BAC

bn:
(bits 3-6)

The address of the bit to be manipulated is given by these four bits.

Remember that each bit is given its OCTAL address.

dr:
(bits 0-2)

The following registers allow bit operations and are specified as follows:

register	mnemonic	code
STS	†	0_8
D	DD	1_8
P	DP	2_8
B	DB	3_8
L	DL	4_8
A	DA	5_8
T	DT	6_8
X	DX	7_8

† For STS no mnemonic is required as it is implied by the following table of compound mnemonics:

STS register

There are only eight bits which can be operated on in the STS register. They have special mnemonics and unique octal code values which combine the bn and dr fields.

compound mnemonic	STS bit	description	octal code
SSPTM	0	page table flag	00_8
SSTG	1	floating point rounding flag	10_8
SSK	2	1 bit accumulator (*K*)	20_8
SSZ	3	error flag (*Z*)	30_8
SSQ	4	dynamic overflow flag (*Q*)	40_8
SSO	5	static overflow flag (*O*)	50_8
SSC	6	carry flag (*C*)	60_8
SSM	7	multi-shift link flag (*M*)	70_8

Examples:

1. BSKP ONE SSC

   Skip the next instruction if the carry flag is set.

2. BSET ZRO SSO

   Reset the static overflow flag.

3. BORC 60_8 DX

   Complement bit 6 in the X register, then OR the bit with *K*, leaving the result in*K*.

These instructions address single bytes within the memory map.

Byte addressing:

A special addressing mode is used for these instructions, using the T and X registers (see page 27).

The contents of T point to the beginning of a character string and the contents of X to a byte within the string.

LBYT

Description: Load byte

Load the byte addressed by the contents of the T and X register into the lower byte of the A register. The higher byte of the A register is cleared.

Format: LBYT

Octal code: 142200_8

SBYT

Description: Store byte

Load the byte contained in bits 0 to 7 of the A register into one half of the memory location addressed by the T and X registers, the second half of this location is not changed.

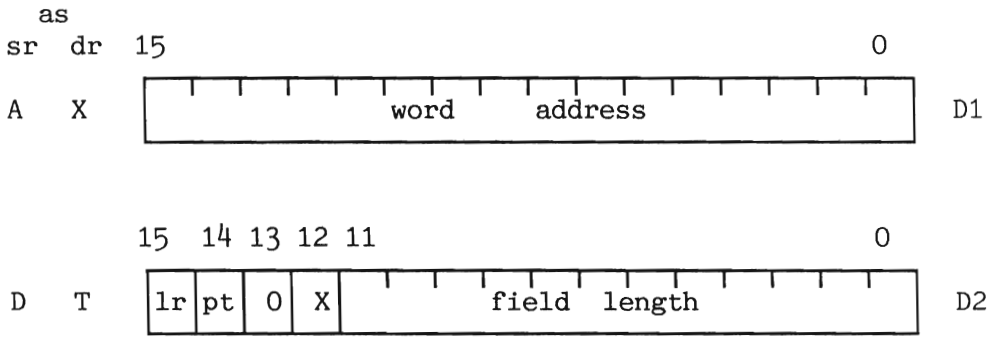
Format: SBYT

Octal code: 142600_8

These instructions use byte operands.

Byte operands:

Byte operands occupy fields within the memory. Operands are specified by two 16-bit words, known as descriptors, giving the start address and the field length.



Note:
bit 13 should always be zero
bit 12 can be any value

D1

The first part of the descriptor, D1, gives the start address of the operand.

D2

The second word has the following features:

lr - determines whether the operand starts in the left or right byte of the memory location addressed by D1.

lr = 0 left byte start
lr = 1 right byte start

pt - gives the page table mode:

pt = 0 normal
pt = 1 alternative

field length - the number of operand bytes (a maximum of 4095 bytes).

sr and dr

The A and D registers hold the source operand desciptor and the X and T registers hold the destination operand.

BFILL

Description: Byte fill

Only the destination is used as an operand in this instruction (it is placed in the X and T registers). The lower byte of the A register is then filled with the destination field.

After execution, bit 15 of the T register points to the end of the field (after the last byte position) and the field length equals zero.

Format: BFILL

Octal code: 140130_8

MOVB

Description: Move byte

This instruction moves a block of bytes from the memory location addressed by the source operand to that of the memory location addressed by the destination operand.

After execution, bit 15 of the D and T registers point to the end of the field that has been moved. The field length of the D register (source) equals zero and the T register (destination) field length is equal to the number of bytes moved.

Format: MOVB

Octal code: 140131_8

MOVBF

Description:

Move bytes forward

This instruction moves a block of bytes from the memory location addressed by the source operand to that of the memory location addressed by the destination operand.

After execution, bit 15 of the D and T registers point to the end of the field that has been moved. The field length of the D

register (source) equals zero and the T register (destination) field length is equal to the number of bytes moved.

Format:

MOVBF

Octal code:

140132_8

This instruction moves a block of words from one area of memory to another.

The type of transfer is given by the opcode field denoted ΔΔ. The base octal code is $1431\Delta\Delta_8$:

$\Delta\Delta_8$	move from:	move to:	
00_8	PT	PT	
01_8	PT	APT	
02_8	PT	phy.memory	*
03_8	APT	PT	
04_8	APT	APT	
05_8	APT	phy.memory	*
06_8	phy.memory	PT	*
07_8	phy.memory	APT	*
10_8	phy.memory	phy.memory	*

PT normal page table
APT alternative page table
phy. physical
* privileged instruction

The following registers control transfer:

A and D - the source address
X and T - the destination address
L - the number of words to be moved (2048 words maximum)

A and/or X only are used for physical memory-block moves and are incremented when the D and/or T registers overflow.

If the L register contains a value greater than 2048 ($L = 4000_8$) no words are moved and A,D,T and X are unchanged.

After transfer, the registers contain:
A,D,T,X - the addresses after the last moved word
L - zero

Special cases:

If the memory management system is off, bank 0 of physical memory is addressed (Bit PTM of the STS register is zero) and the following transfer fields become equivalent:

ΔΔ = 00 = 01 = 03 = 04
ΔΔ = 02 = 05
ΔΔ = 06 = 07

MOVEW can be interrupted. L, A, D, X, T and P registers are then changed to restart execution.

Format:

MOVEW

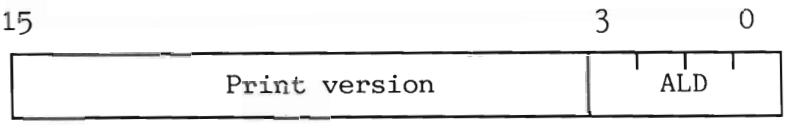
Octal code:

1431ΔΔ

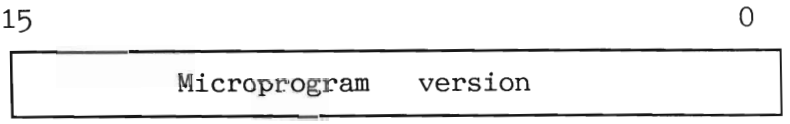
This instruction is used to read the version of ND-110 CPU installed.

Three registers are loaded simultaneously with information in the following format:

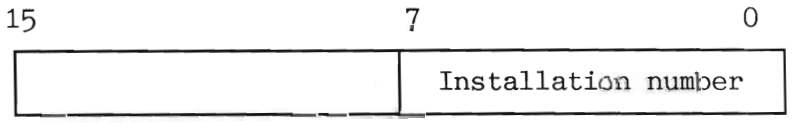
A register



T register



D register



The installation number of the CPU is 16 bytes long.

The VERSN instruction can only load one byte of the installation number into the D register each time it is executed. The A register is used to address the sixteen bytes.

To read a byte of the installation number the A register must be loaded with an installation byte address before the VERSN instruction is executed. The address of the byte to be read is given by the value of bits 8-11 in the A register (equivalent to bytes 0-16). To read the complete installation number both the A register bit field must be incremented and the VERSN instruction executed sixteen times.

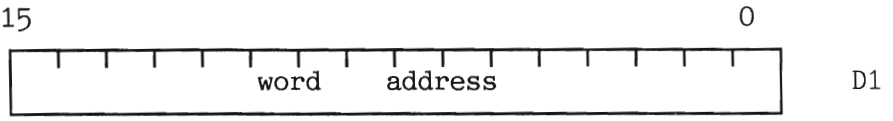
Instruction format: VERSN

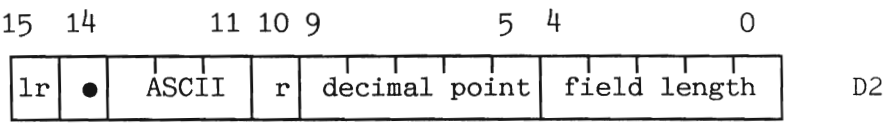
Octal code: 140133_8

These instructions use decimal operands residing in main memory only.

Instruction format: *<decimal instruction>*

Instruction structure:





Descriptors:
(D1 and D2)

Two 16-bit words (D1 and D2) specify the operands used in decimal instructions:

D1:

The first descriptor, D1, gives the **word address** of the decimal operand in memory.

D2:

D2 describes the following operand features:

lr
(bit 15)

lr = 0
The operand starts in the left byte of a memory word. (In the least significant 8 bits.)

lr = 1
The operand starts in the right byte of a memory word. (In the most significant 8 bits.)

ASCII (bits 11 - 13)

These three bits give the sign representation used for ASCII format (see Decimal Notation Section).

bits 13	12	11	sign representation
0	0	0	embedded trailing
0	0	1	separate trailing
0	1	0	embedded leading
0	1	1	separate leading
1	0	0	unsigned

Bit 13 also represents an unsigned number in BCD representation.

r (bit 10)

rounding bit
If rounding is selected, one is added to the shifted operand when the least significant digits are lost during shift and the last digit shifted out of the field is ≥5.

r = 1, rounding on
r = 0, rounding off

decimal point (bits 5 - 9)

These bits give the position of the decimal point. The range is 32 places (from 0 to 31), positive or negative. Zero is the decimal place to the right of the least significant digit.

The number MUST be less than the operand field length.

field length (bits 0 - 4)

These bits give the operand field length in nibbles (4-bit values) or bytes (8-bit values). BCD numbers are represented by 4 bits (1 nibble) so the field length will be in nibbles; an ASCII coded digit is represented by a byte and the field length will be in bytes.

Operands start at any byte address in memory.

The maximum field length is 32 nibbles/bytes.

Decimal operands:

Decimal operands occupy a maximum of eight 16-bit memory locations. Each operand consists of BCD coded numbers.

Decimal operands must be right adjusted so that the least significant digit and sign are in the last byte of the operand field.

Before any instruction is executed the operands are read into the register file. The result of the instruction is written into memory.

All decimal instructions use two operands. The descriptors of each operand are held in separate registers:

First operand descriptor: A and D registers.
Second operand descriptor: X and T registers.

Decimal overflow:

Decimal overflow is caused by

EITHER a carry from the most significant digit position in the result

OR an oversized result, the second operand was larger than the first causing the significant digits of the result to be lost.

Note:
the field size alone does not indicate possible overflow.

Most decimal instructions are followed by an instruction or jump to a routine which takes care of overflow errors, known as **an error return.** A decimal instruction executed without error generation skips the error return and program execution continues at the second instruction after it.

Description: Add two decimal operands

(op1) ⟵ (op1) + (op2)

Add the second operand to the first operand, leaving the result in the first operand's location.

If the first operand field is too short to contain all the significant digits of the result then **decimal overflow** occurs.

If bit 13 of D2 in the first operand is set, the sign of the result will be 17_8 (BCD unsigned).

Any empty operand, that is with a field length of zero, is treated as a positive zero.

Format:

```
ADDD
```

Octal code: 140120_8

Instruction sequence:

```
ADDD
error handling instruction
next instruction after ADDD or after error
handling routine
```

Note:
Operands should be normalized before this instruction is executed using the SHDE instruction.

Example:

```
ADDD
JMP *12
STX 20
```

The ADDD instruction causes the program counter to skip the next instruction UNLESS an error has been generated. In this case, the instruction immediately after ADDD will handle the error in some way (in this example a jump is executed on error to ((P) + 12_8).

(* is the assembler mnemonic for the P register)

COMD

Description:

Compare two decimal operands

(A) ⟵ (op1) compared to (op2)

(A) = 0 if (op1) = (op2)
(A) = 1 if (op1) > (op2)
(A) = -1 if (op1) < (op2)

Compare the first operand with the second operand, leaving the result in the A register.

If the two operands are unequal in field length, the shorter operand is extended with zeros to allow comparison. The operands are unaffected by the instruction.

The positions of the decimal points are not taken into account when the two operands are compared, so the two operands should be normalized using the SHDE instruction first.

Any empty operand, that is with a field length of zero, is treated as a positive zero. An unsigned number is treated as positive. Positive and negative zeros are equal.

Format:

```
COMD
```

Octal code:

140122_8

Instruction sequence:

```
COMD
error handling instruction
next instruction after COMD or after error
handling routine
```

Example:

```
COMD
JMP *30₈
AAA 20₈
```

The COMD instruction causes the program counter to skip the next instruction UNLESS an error has been generated. In this case, the instruction immediately after COMD will handle the error in some way (in this example a jump is executed on error to ((P) + 30_8).

(* is the assembler mnemonic for the P register)

Description:

Convert to BCD

(op2) ⟵ (op1) in BCD format

Convert the first operand from its ASCII format to BCD format, placing the result in the second operand location.

Conversion process:

PACK carries out the following steps:

1. Checks the sign and digits of the operand (op1) are encoded as ASCII digits. (Reporting illegal codes as error code 2.)

2. Takes the 4 least significant bits of each ASCII digit as the equivalent BCD digit.

3. Converts the ASCII sign of the operand to BCD:

ASCII	BCD	Sign
53_8	14_8	+
55_8	15_8	-

Note: If bit 13 of the descriptor D2 is set, then the code 17_8 (BCD unsigned) is used.

4. Extends the second operand field (op2) with zeros if the result is too small to fill the field.
Reports overflow has occurred (error code 3) if the second operand field length is too short to contain the significant digits of the result (the remaining digits are ignored).

Format:

PACK

Octal code:

140124_8

Error code:

An error code is placed in bits 0 to 4 of the D register if an illegal code conversion is attempted or if overflow occurs. The first detected error will be reported.

Error code:	2	illegal code†
	3	overflow

† bit 15 of both the A and D registers point to the byte containing the illegal code.

SHDE

Description:

Decimal shift

(op2) ⟵ (op1) shifted

This instruction is used to normalize operands for decimal operations.

The shift count determines whether the operand is shifted to the left or right:

(op2 - op1) positive then shift op1 to right
(op2 - op1) negative then shift op1 to left

If significant digits are lost by carrying out a left shift, an error is generated, directing the program counter to the instruction after the SHDE (the error return). If no errors occur this instruction is skipped.

The digits of the first operand are shifted and the result is placed in the second operand's memory location.

The number of places shifted is given by the difference in decimal position of the two operands. This normalizes the first operand (op1) to the second (op2) for decimal operations such as ADDD.

The sign of the normalized operand (op1) is as follows:

BCD	Sign
14_8	+
15_8	-

An unsigned operand is converted to a plus unless bit 13 of the descriptor D2 is set, when the BCD equivalent of unsigned (17_8) is used.
The sign and digits of the first operand are checked before execution and any illegal digit codes reported.

If bit 10 of descriptor D2 (op2) is set the result is rounded, that is a 1 is added to the operand if the last digit shifted out of the field is ≥ 5.

Format: SHDE

Octal code: 140126_8

Instruction sequence:
SHDE
error handling instruction
next instruction after SHDE or after error handling routine

Example:

```
SHDE
JMP *10₈
SAD 20₈
```

The SHDE instruction causes the program counter to skip the next instruction UNLESS an error has been generated, when the instruction immediately following the SHDE will handle the error in some way (in this example a jump is executed on error to (P) + 10_8)).

(* is the assembler mnemonic for the P register)

SUBD

Description: Subtract two decimal operands

(op1) ⟵ (op1) - (op2)

Subtract the second operand from the first operand, leaving the result in the location of the first operand.

If the first operand field is too short to contain all the significant digits of the result then **decimal overflow** occurs.

If bit 13 of D2 in the first operand is set, the sign of the result will be 17_8 (BCD unsigned).

Any empty operand, that is with a field length of zero, is treated as a positive zero.

A zero difference can have either a negative or positive sign.

Format: SUBD

Octal code: 140121_8

Instruction sequence: SUBD *error handling instruction*
next instruction after SUBD or after error handling routine

Example:

```
SUBD
JPL *30(8)
ADD *15(8)
```

The SUBD instruction causes the program counter to skip the next instruction UNLESS an error has been generated, when the instruction immediately following the SUBD will handle the error in some way (in this example, a jump is executed on error to a subroutine at (P) + 15_8)).

(* is the assembler mnemonic for the P register)

UPACK

Description: Convert to ASCII

(op2) ⟵ (op1) in ASCII format

Convert the first operand from its BCD format to ASCII format, placing the result in the second operand location.

Conversion process: The instruction carries out the following steps:

1. Checks the sign and digits of the operand (op1) are encoded as BCD digits. (Illegal codes generate an error code 2.)

2. Takes each BCD digit as the lower nibble of an equivalent ASCII digit. Sets the upper nibble of the ASCII byte (the zone) to 0011_2.

3. Converts the BCD sign of the operand to ASCII:

BCD	ASCII	Sign
0	53_8	+
1	55_8	-

Note: If bit 13 of the descriptor D2 is set then the code 17_8 (BCD unsigned) is used.

4. Extends the second operand field (op2) with ASCII zeros (60_8) if the result is too small to fill the field.
Reports overflow has occurred (error code 3), if the second operand field length is too short to contain the significant digits of the result (the remaining digits are ignored).

Format: UPACK

Octal code: 140125_8

Error code: An error code is placed in bits 0 to 4 of the D register if an illegal code conversion is attempted or if overflow occurs. The first detected error will be reported.

Error code:		
	2	illegal code†
	3	overflow

† bit 15 of both the A and D registers point to the byte containing the illegal code.

STACK INSTRUCTIONS
Description

These instructions handle stack operations improving the execution time of high-level language-based programs.

The B register will always point to a "stack-frame" containing:

stack frame content mnemonic	pointed to by B=	description
LINK	-200_8	next instruction address †
PREVB	-177_8	previous stack frame address
STP	-176_8	next stack frame address
SMAX	-175_8	top of stack address
-	-174_8	reserved for system use
ERRCODE	-173_8	(A) after an ELEAV instruction

† In the case of a LEAVE instruction

The stack-handling instructions are page-fault tolerant in the ND-110.

ELEAV

Description: Error leave stack

$(B = 200_8) \longleftarrow (B = 200_8) - 1$
$(P) \longleftarrow (B = 200_8)$ {LINK}
$(B) \longleftarrow (B = 177_8)$ {PREVB}
$(A) \longleftarrow$ ERRCODE
$(B = 173_8) \longleftarrow (A)$ {ERRCODE}

If an error occurs leave the stack.

This instruction saves the previous stack pointer in LINK and restores the B register to its previous value (PREVB) before leaving the stack. The stack is left by loading the P register (program counter) with the return address (LINK). The A register is loaded with an error code which is saved in the ERRCODE stack entry (pointed to by $B = 173_8$).

Format: ELEAV

Octal code: 140137_8

ENTR

Description: Enter stack

$(B = 177_8) \longleftarrow (B)$ {save current pointer in PREVB}

$(B = 175_8) \longleftarrow (B = 175_8)$ {SMAX}

$(B = 200_8) \longleftarrow (L) + 1$ {save return address in LINK}

$(B) \longleftarrow (B = 176_8) + 200_8$ {new pointer}

$(B = 176_8) \longleftarrow$ stack demand $+ (B)$

This instruction saves the current stack pointer (B), the return address (LINK) and previous stack pointer (PREVB). It transfers the top of stack address (SMAX) and establishes the new stack demand and pointer.

Stack overflow causes an error return, that is the program continues at the address following the stack demand value. In all other cases, the program skips this address to find the return address from the stack.

Format:

ENTR
<stack demand-value in words>

<error return address>
<return address>

Octal code: 140135_8

INIT

Description: Initialize stack

LINK ⟵ L + 1 {stack start}

PREVB ⟵ (B) {save current pointer}

SMAX ⟵ stack start address
+ maximum stack size

(B) ⟵ (B = 200_8) + 200_8 {establish new pointer}

STP ⟵ stack demand + (B)

Load the addresses pointed to by B with the stack frame addresses.

Note:
Stack overflow and flag error causes an error return, that is the program continues at the address following the stack demand value. In all other cases, the program skips this address to find the return address from the stack.
(Flag bit 0 ≠ STS register bit 0 is a flag error.)

Format:

INIT
number of words allocated to stack
address of stack start
maximum stack size
flag
address left empty

error return address
return address

Octal code: 140134_8

Description: Leave stack

$$(P) \longleftarrow (B = 200_8) \qquad \{LINK\}$$
$$(B) \longleftarrow (B = 177_8) \qquad \{PREVB\}$$

This saves the previous stack pointer in LINK. The B register is restored to its previous value (PREVB) and the stack left by loading the P register (program counter) with the return address (LINK).

Format: LEAVE

Octal code: 140136_8

RDUS

Description: Read a word without using cache

(T) points to the virtual memory word to be accessed
(A) ⟵ memory word addressed by T

The address given by the T register is a logical memory address. It is normally translated into a physical address using page tables (if the memory management system is on). The contents of the location addressed by T are loaded into the A register.
The old content of the memory address is always read from the memory and never from cache.

The execution time of this instruction includes two read bus cycles (semaphore cycles - see ND-110 Functional Description Manual ND.06.027)

Format: RDUS

Octal code: 140127_8

TSET

Description: Test and set

(T) points to the virtual memory word to be accessed
(A) ⟵ memory word addressed by T

The address given by the T register is a logical memory address. It is normally translated into a physical address using page tables (if memory management is on). The contents of the location addressed by T are simultaneously loaded into the A register as the location is written to with all 1s. The memory system is dedicated to this task and no other memory access is allowed during the operation. This can be used for processor synchronization.
The old content of the memory address is always read from the memory and never from cache. The all 1s' data word is never written to cache.

Format: TSET

Octal code: 140123_8

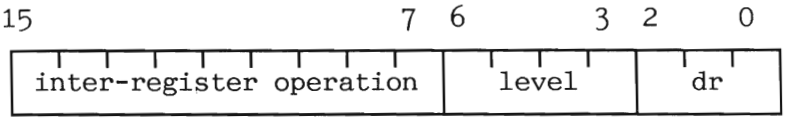
These instructions are PRIVILEGED and only available to:

programs running in system mode (rings 2-3)
programs running without memory protection

These instructions access registers outside the current program level. (There is a register set for each of the 16 program levels.)

Instruction format: *<inter-register operation> <level$_8$ x 10$_8$> <dr>*

Instruction structure:



inter-register operation: (bits 15-7)

There are two inter-register instructions:
IRR
IRW
They read and write to/from a register outside the current program level.

level: (bits 3-6)

These bits give the program level of the block (0-15).The level is written in its OCTAL format and multiplied by 10_8 to set the correct bits in the octal code.

<level$_8$ x 10$_8$>

dr: (bits 0-2)

The following registers allow bit operations and are specified as follows:

register	mnemonic	code
STS	-	0_8
D	DD	1_8
P	DP	2_8
B	DB	3_8
L	DL	4_8
A	DA	5_8
T	DT	6_8
X	DX	7_8

Inter-level Register Instructions

IRR

Description: Inter-register read

Read into the A register of the current level the contents of a register in the program level given by the instruction.

(This instruction can also be used to read registers within the current program level into the A register.)

If the status register is read (STS), the A register is loaded with the lower byte (bits 0-7) of STS only, the higher byte is cleared.

Format: IRR <$level_8$ x 10_8> <*dr*>

Octal code: 153600_8

Example: IRR 160_8 DP (153762_8)

Copy the program counter on program level 14 into the A register of the current program level.

IRW

Description: Inter-register write

Write the contents of the A register on the current level into the A register of the program level given by the instruction.

Note:
This instruction results in a no-operation if the A register of the current program level is used.

If the status register (STS) is the destination, only the lower byte (bits 0-7)is written to with bits 0-7 of the A register.

Format: IRW <$level_8$ x 10_8> <*dr*>

Octal code: 153400_8

Example: IRW 100_8 DB (153503_8)

Copy the A register on the current program level into the B register on program level 8.

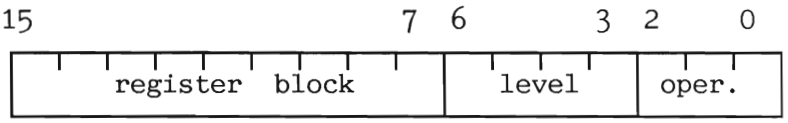
These instructions are PRIVILEGED and only available to:

programs running in system mode (rings 2-3)
programs running without memory protection

These instructions access the program level register blocks.

Instruction format: *<register block operation> <level$_8$ x 10$_8$>*

Instruction structure:



register block oper. (bits 0-2 and 7-15)

There are two register block instructions:
LRB
SRB

The register block is always loaded or stored in the following register sequence:

P	(program counter)
X	
T	
A	
D	
L	
STS	(status register)
B	

Note:
Only the lower byte (bits 0-7) of the STS register are loaded or stored; the higher byte is zero.

The X register contents point to the base address of the register block in memory.

level: (bits 3-6)

These bits give the program level of the block (0-15). The level is written in its OCTAL format and multiplied by 10_8 to set the correct bits in the octal code.

<level$_8$ x 10$_8$>

Register Block Instructions

LRB

Description: Load register block

Load the contents of a memory block pointed to by the X register into the register block of the program level given in the instruction.

If the instruction specifies the current program level, the P register (program counter) is not loaded from memory and is unchanged.

Format: LRB <$level_8$ x 10_8>

Octal code: 152600_8

Example: LRB 160_8 (152760_8)

Load the memory block pointed to by the X register into the register on program level 14.

```
P (level 14) ⟵ (ea)
X (level 14) ⟵ (ea + 1)
      ↓          ↓       ↓
B (level 14) ⟵ (ea + 7)
```

SRB

Description: Store register block

Load the register block of the program level given in the instruction into the memory block pointed to by the X register.

If the instruction specifies the current program level the P register points to the instruction following SRB.

Format: SRB <$level_8$ x 10_8>

Octal code: 152402_8

Example: LRB 100_8 (152702_8)

Store the register block of program level 8 in the memory block pointed to by X.

```
    (ea) ⟵ P (level 8)
(ea + 1) ⟵ X (level 8)
      ↓        ↓       ↓
(ea + 7) ⟵ B (level 8)
```

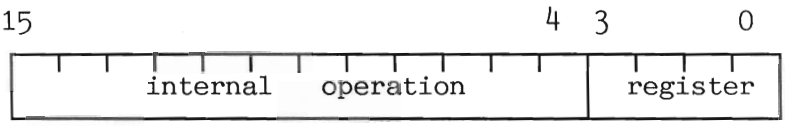
These instructions are PRIVILEGED and only available to:

programs running in system mode (rings 2-3)
programs running without memory protection

These instructions access internal CPU registers which cannot be reached by normal register operations.

Instruction format: *<internal operation> <register>*

Instruction structure:



internal operation: (bits 15-4)

There are four internal instructions:

MCL
MST
TRA
TRR

They operate on internal registers or specific areas; a table of the internal registers affected by these instructions can be found after the description of the instructions (see page 120).

register (bits 3-0): These bits give the internal register address.

MCL

Description: Masked clear

The A register is used as a mask to clear bits within the selected internal register. Setting a bit in the A register clears the corresponding bit in the internal register.

See table for internal registers that allow MCL.

Format: MCL *<register>*

Octal code: 150200_8

Example: MCL STS (150201_8) [(A) = 000100_8]

Clear the carry flag (bit 6) in the status (STS) register.

MST

Description: Masked set

The A register is used as a mask to set bits within the selected internal register. Setting a bit set in the A register sets the corresponding bit in the internal register.

See table for internal registers that allow MST.

Format: MST *<register>*

Octal code: 150300_8

Example: MST PIE (150307_8) [(A) = 000140_8]

Set bits 5 and 6 in the priority interrupt enable register (PIE).

TRA

Description: Transfer to A register

The internal register given in the instruction is copied into the A register.

See table for internal registers that allow TRA.

Format: TRA *<register>*

Octal code: 150000_8

Example: TRA (150012_8)

Copy the contents of the automatic load descriptor into the A register.

TRR

Description: Transfer A to internal register

The internal register given in the instruction is loaded with the contents of the A register.

See table for internal registers that allow TRR.

Format: TRR *<register>*

Octal code: 150100_8

Example: TRR (150306_8)

Transfer the A register contents into the priority interrupt detect register.

Register Transfer Instructions

Internal Register	Name and Description	Octal Code	TRA	TRR	MCL	MST
PANS	Panel Status	0_8	•			
PANC	Panel Control	0_8		•		
STS	Status	1_8	•	•	•	•
OPR	Operator Panel Switch	2_8	•			
LMP	Operator Lamp	2_8		•		
PGS	Paging Status	3_8	•			
PCR	Paging Control	3_8		•		
PVL	Previous Program Level	4_8	•			
IIC	Internal Interrupt Control	5_8	•			
IIE	Internal Interrupt Enable	5_8		•		
PID	Priority Interrupt Detect	6_8	•	•	•	•
PIE	Priority Interrupt Enable	7_8	•	•	•	•
CSR	Cache Status	10_8	•			
CCL	Cache Clear	10_8		•		
ACTL	Active Level	11_8	•			
LCIL	Lower Cache Inhibit Limit	11_8		•		
ALD	Automatic Load Descriptor	12_8	•			
UCIL	Upper Cache Inhibit Limit	12_8		•		
PES	Parity Error Status	13_8	•			
CILP†	Cache Inhibit Page	13_8		•		
PGC	Paging Control	14_8	•			
PEA	Parity Error Address	15_8	•			
ECCR	Error Correction Control	15_8		•		
CS†	Control Store	17_8	•	•		

† These registers are not available on the ND-100.

This instruction is PRIVILEGED and only available to:

programs running in system mode (rings 2-3)
programs running without memory protection

The instruction controls all transfers between the ND-110 and any external devices.

Description:

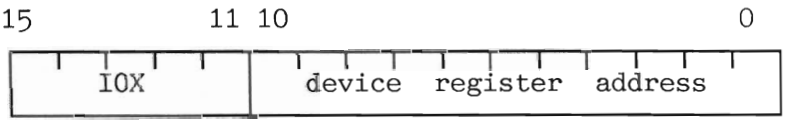
Exchange information between I/O system and A register.

IOX can be used to address a maximum of 2048 device registers for external devices connected to the ND-110 CPU. Data, control and status between device and CPU can be exchanged.

Instruction format:

IOX *<device register address>*

Instruction structure:



IOX:

This field is the fixed code of the instruction IOX (11101_2)

device register address: (bits 0-10)

These 11 bits limit the number of external devices that can be addressed by the CPU.

Bit 0 gives the direction of transfer:

If 0 input (from device to CPU)
If 1 output (from CPU to device)

Register addresses can hold data, command or status information for a device.

An external device may require more than one register address, for example a magnetic tape unit may need several register addresses; these should be given successive device-register addresses (remembering to use odd addresses for input and even addresses for output).

Note:
The number of external devices that can be controlled by the CPU depends on the configuration of the devices.

Octal code:

164000_8

Examples:

1.To give an instruction to a device:

LDA *<device command code>*
IOX

The lsb of IOX will be 1. The A register contents are output to the device addressed within the IOX opcode.

2.To check the status of a device:IOX

The lsb of IOX will be 0. The status code of the device addressed by the IOX opcode will be loaded into the A register.

3.Transfer data:IOX

Data from the device addressed by the IOX instruction, is read into the A register if the lsb of IOX is 0. If the lsb of IOX is 1, the A register contents are output to the device.

This instruction is PRIVILEGED and only available to:

programs running in system mode (rings 2-3)
programs running without memory protection

The instruction controls transfers between the ND-110 and external devices.

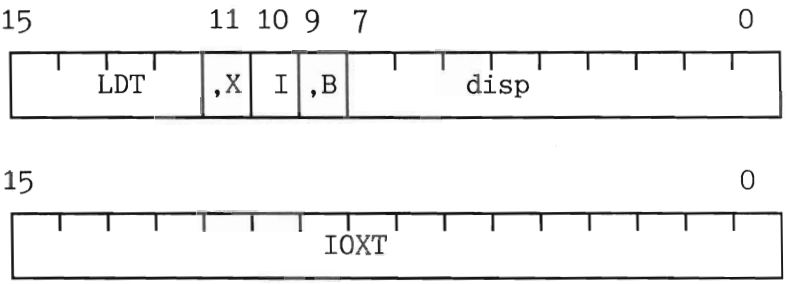
Description:

IOXT can be used to address a maximum of 65536 device-register addresses for external devices connected to the ND-110 CPU.

Instruction format:

LDT *<address mode>* *<disp>*
IOXT

Instruction structure:



LDT instruction:

The IOXT instruction uses the T register contents as the device register address. The 16-bit T register gives a limit of 65536 register addressses.

This address MUST be loaded into the T register before IOXT is executed, hence LDT is used. (See Memory Transfer Instructions,LOAD for explanation of LDT.)

Note: The number of external devices that can be controlled by the CPU depends on the configuration of the devices.

IOXT instruction:

IOXT is used as a single mnemonic.

Octal code:

150415_8

Examples:

See ND-110 Functional Description (ND-06.026) for the standard ND assignment of device register addresses.

INTERRUPT CONTROL INSTRUCTIONS
Description

These instructions are PRIVILEGED and only available to:

programs running in system mode (rings 2-3)
programs running without memory protection

These instructions control the CPU interrupt system.

Instruction format: *<interrupt control operation>*

General description: The ND-110 has a priority interrupt system with 16 program levels. Each program level has its own set of working registers (A, B, D, L, P, STS, T, X). The program levels have increasing priority, that is program level 15 has the highest priority and program level 0 the lowest.

The 16 levels are subdivided as follows:

level	used for:	controlled by:
15	very fast user interrupts	program/ext.device
14	internal hardware status interrupts	program/ext.device
13-10	vectored interrupts*	program/ext.device
9-0	system and user programs	program

* 2048 possible sources

Program level selection and control is via two 16-bit registers:

PID Priority Interrupt Detect
PIE Priority Interrupt Enable

PID is affected by program and external interrupts; PIE is controlled by program only. They can only be changed or monitored by the privileged instructions: TRA, TRR, MST and MCL (see pages 118-120).

Note:
When the power is turned on, the power-up sequence resets PIE and PID so that program level 0 is selected.

Interrupt programming is via three registers:

IIC Internal Interrupt Code
IIE Internal Interrupt Enable
PVL Previous Level (of hardware interrupt source)

The following instructions control interrupts:

IDENT

Description:

Identify vectored interrupt

Identify and service the input/output device causing the interrupt.

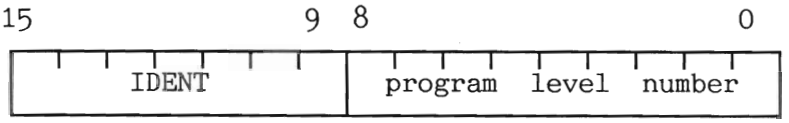
Bits 0-8 of the A register are loaded with the identification code of the device causing the interrupt (bits 9-15 are zeros). If IDENT is executed without an interrupting device to service, the A register is unchanged.

Note:
If several devices on the same program level have simultaneous interrupts, the device plugged into the ND-110 card frame nearest to the CPU card has highest priority and is serviced first. The PID register bit corresponding to this interrupt line will remain set until all the interrupting devices are serviced.

Format:

IDENT <program level number>

Structure:



program level number:

Vectored interrupts are only allowed on program levels 10 to 13. The following program level number mnemonics are used:

level	mnemonic	code
10	PL10	04_8
11	PL11	11_8
12	PL12	22_8
13	PL13	43_8

Octal code:

143600_8

Interrupt Control Instructions

IOF

Description: Interrupt System OFF

Disables the interrupt system.

On IOF the ND-110 continues operation at the same program level.

Format: IOF

Operator indication ION display is reset.

PVL status: The PVL register is unchanged.

Octal code: 150401_8

ION

Description: Interrupt System ON

Enables the interrupt system.

On ION the ND-110 resumes operation in the program level with highest priority.

Format: ION

Operator indication ION display is lit.

PVL status The PVL register will change to the new program level.

Octal code: 150402_8

PIOF

Description: Memory management and interrupt system OFF

Disables both the memory management and interrupt systems. This combines the functions of the IOF and POF instructions.

BEFORE USE: Check conditions of the IOF instruction.

Format: PIOF

Octal code: 150405_8

PION

Description: Memory management and interrupt system ON

Enable both the memory management and interrupt systems. This combines the functions of the ION and PON instructions.

BEFORE USE: Check conditions of ION and PON instructions.

Format: PION

Octal code: 150412_8

WAIT

Description: Wait

This operates as follows:

IF the interrupt system is OFF ...

The ND-110 stops with the program counter (P register) pointing to the instruction after the wait and the front panel RUN indicator is turned off. (To restart the system, type ! on the console terminal.)

IF the interrupt system is ON ...

The ND-110 exits from the current program level (resetting the corresponding PID bit) and enters the program level with the highest priority, normally a program level lower than the one which executes the WAIT instruction. If there are no interrupt requests awaiting service then program level 0 is entered.

Note:
A WAIT on program level 0 is ignored.

WAIT followed by a number less than 400_8 can be used to detect which location caused the program stop.

Format: WAIT

Octal code: 151000_8

These instructions are PRIVILEGED and only available to:

programs running in system mode (rings 2-3)
programs running without memory protection

These instructions control the CPU memory management system.

PIOF

Description: Memory management and interrupt system OFF, see Interrupt Control Instructions.

PION

Description: Memory management and interrupt system ON, see Interrupt control instructions

POF

Description: Memory management OFF

Disable memory management system. The next instruction will be taken from a physical address given by the address following the POF instruction.

Note:
The CPU will be in an unrestricted mode without any hardware protection features - all instructions are legal and all memory accessible.

Format: POF

Octal code: 150404_8

PON

Description: Memory management ON

Enable memory management system. The next instruction after PON will then use the page-index table specified by PCR.

BEFORE USE: Ensure that the:
Interrupt system is enabled,
internal hardware interrupts are enabled,
page tables and PCR registers are initialized.

Format:	PON
Octal code:	150410_8

REX

Description:	Reset extended address mode Set the paging system to the 19-bit address mode instead of the 24-bit address mode. (Creating a physical address space of up to 512K words.)
Flag affected:	*SEXI* (bit 13 of the STS register) cleared
Format:	REX
Octal code:	150407_8

SEX

Description:	Set extended address mode Set the paging system to the 24-bit address mode instead of the 19-bit address mode. (Creating a physical address space of up to 16M words.)
Flag affected:	*SEXI* (bit 13 of the STS register) set.
Format:	SEX
Octal code:	150406_8

Physical Memory Control Instructions
DEPOSIT and EXAM

These instructions are PRIVILEGED and only available to:

programs running in system mode (rings 2-3)
programs running without memory protection

These instructions monitor physical memory location contents.

DEPO

Description: Deposit

Store the contents of the T register in the physical memory location pointed to by the A and D register contents.

Format: DEPO

Octal code: 150417_8

EXAM

Description: Examine

Load the contents of the physical memory location, pointed to by the A and D register contents, into the T register.

Format: EXAM

Octal code: 150416_8

This instruction is PRIVILEGED and only available to:

programs running in system mode (rings 2-3)
programs running without memory protection

LWCS is a no-operation in the ND-110.

The ND-110 is software compatible but not microcode compatible and writing to the writable control store has no meaning in the ND-110. A no-operation is executed so that programs written for the ND-100 and NORD-10 can continue.

Octal code = 143500_8

Unused areas of the microprogram can be read or written to using the TRR CS or TRA CS instruction (see page 119).

Further information on the LWCS instruction for the ND-100 can be found in the ND-100 Reference Manual (ND-06.014).

OPCOM Mode Instruction
OPCOM

This instruction is PRIVILEGED and only available to:

programs running in system mode (rings 2-3)
programs running without memory protection

Description:

Operator Commmunication

This instruction allows the programmer to use a terminal in direct communication with the CPU board.

When the CPU is running, MOPC can be used to read input from the console.

This the software equivalent to pressing the OPCOM button on the control panel of the ND-110.

Format:

OPCOM

Octal code:

150400_8

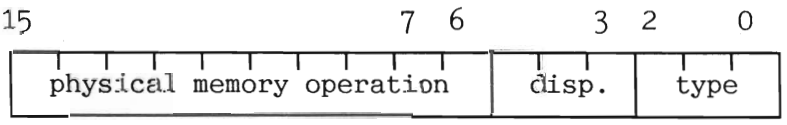
These instructions are PRIVILEGED and only available to:

programs running in system mode (rings 2-3)
programs running without memory protection

These instructions read/write from/to physical memory locations independent of whether paging is on or off. If the address is within the page-table range then the page tables are affected.

Instruction format: *<physical instruction mnemonic> <disp>*

Instruction structure:



physical memory operation type: (bits 15-6 and 2-0)

There are seven physical memory read/write instructions, specified by a base octal code of 143300_8 (bits 15-6) and type field (bits 2-0).

disp.

The contents of the T and X register give the effective address of the physical memory location (see page 28). A 3-bit displacement can be added to the X register within the instruction code. This is denoted by Δ in the following codes:

instruction mnemonic	octal code
LDATX	$1433\Delta0_8$
LDXTX	$1433\Delta1_8$
LDDTX	$1433\Delta2_8$
LDBTX	$1433\Delta3_8$ †
STATX	$1433\Delta4_8$
STZTX	$1433\Delta5_8$
STDTX	$1433\Delta6_8$

† If you use programs written for ND-100 computers with the microprogram version numbers 015xx A-J (48 bit) or 026xx A-F (32 bit), LDBTX would have been followed by a word containing 177777_8. This is not necessary for later ND-100 versions nor for the ND-110. Running these earlier programs may change the status of the K bit in the ND-110 and later ND-100s.

SINTRAN III Memory Transfer Instructions

LDATX

Description: Load A register

(A) ⟵ (ea)

Load the contents of the physical memory location pointed to by the effective address into the A register.

Format: LDATX *<disp>*

Octal code: $1433\Delta0_8$

LDBTX

Description: Load B register

(B) ⟵ 177000_8 OR (2(ea))

Load the contents of the physical memory location pointed to by the twice the effective address contents into the B register, then OR the value with 177000_8.

See description for usage.

Format: LDBTX *<disp>*

Octal code: $1433\Delta3_8$

LDDTX

Description: Load double word

(A) ⟵ (ea)
(D) ⟵ (ea + 1)

Load the contents of the physical memory location pointed to by the effective address into the A register and the contents of the effective address plus one into the D register.

Format: LDDTX *<disp>*

Octal code: $1433\Delta2_8$

LDXTX

Description: Load X register

(X) ⟵ (ea)

Load the contents of the physical memory location pointed to by the effective address into the X register.

Format: LDXTX *<disp>*

Octal code: $1433\Delta1_8$

STATX

Description: Store A register

(ea) ⟵ (A)

Store the contents of the A register in the memory location given by the effective address.

Format: STATX *<disp>*

Octal code: $1433\Delta3_8$

STDTX

Description: Store double word

(ea) ⟵ (A)
(ea) + 1 ⟵ (D)

Store the double word held in the A and D registers in the memory locations given by the effective address and the effective address plus one.

Format: STDTX *<disp>*

Octal code: $1433\Delta6_8$

STZTX

Description: Store zero

(ea) ⟵ 000000_8

Store zero in the memory location given by the effective address.

Format: STZTX *<disp>*

Octal code: $1433\Delta5_8$

These instructions are PRIVILEGED and only available to:

programs running in system mode (rings 2-3)
programs running without memory protection

These instructions monitor the contents of physical memory.

CHREENTPAGES

Description: Change page tables.

The X register is used to address the current (R1) and previous (Rp) scratch registers.

If the R1 is zero, the reentrant page has nothing to change so the loop is left, otherwise the contents of the memory location pointed to by the R1 + 2 are loaded into T.

T then contains the protect table entry, if the page has not been written to (WIP bit 12 is zero) T and R1 are loaded with Rp. R1 (now containing Rp) is tested again for zero. If the page has been written to, the T register is loaded with the contents of the second scratch register (R2), pointed to by R1, and R2 becomes the address of Rp. X is loaded with R1 as the new pointer to the reentrant pages and Rp is loaded into the D register pointed to by A.

Format: CHREENTPAGES

Octal code: 140303_8

CLEPT

Description: Clear page tables.

This instruction can replace the following instructions:

```
CLEPT:  JXZ * 10(8)
        LDBTX 10(8)
        LDA ,B
        JAZ *3(8)
        STATX 20(8)
        STZ ,B
        LDXTX 00(8)
        JMP *-7(8)
```

Each time the loop is executed (until X becomes zero) the physical memory location addressed by X is loaded into the B register.

The B register contents provide the address of a page table entry, which is loaded into the A register.

If the page table entry is zero (unused) the loop is restarted.

If the page table entry is not zero (used) it is stored in a physical location addressed by X (8 locations away from its original entry) and the original page table entry cleared by placing zero in the location addressed by the B register.

The physical location addressed by X is then loaded into the X register itself and the loop restarted.

* is the mnemonic for P relative addressing.

Format: CLEPT

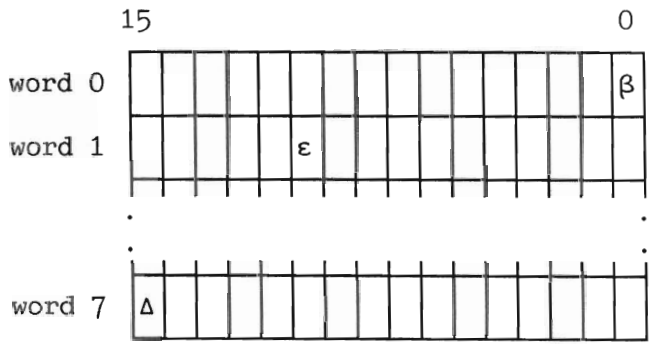
Octal code: 140301_8

CLEPU

Description: Clear page tables and collect PGU information.

This instruction collects information on the PGU (page used) bit of a page table entry whilst executing CLEPT.

The instruction places PGU information in an eight word table called **the page map bank.** Each bit in the bank represents the status of a page's PGU bit as follows:



Δ denotes page 177_8 PGU bit
ε denotes page 32_8 PGU bit
β denotes page 0_8 PGU bit

The L register contains the address of the map entry.

Format: CLEPU

Octal code: 140304_8

CLNREENT

Description: Clear non reentrant pages.

The contents of the memory address at A + 2 are read to find the page table to be cleared along with the SINTRAN RT bitmap (addressed by the X and T registers). The page table entries corresponding to those bits set in the RT bitmap are then cleared.

Format: CLNREENT

Octal code: 140302_8

CLPT

Description: Clear segment from the page tables. (See Appendix B for a software description.)

Format: CLPT

Octal code: 140505_8

CNREK

Description: Clear non reentrant pages (SINTRAN K only). (See Appendix B for a software description.)

Format: CNREK

Octal code: 140504_8

ENPT

Description: Enter segment in page tables. (See Appendix B for a software description.)

Format: ENPT

Octal code: 140506_8

INSPL

Description: Insert page in page list. (See Appendix B for a software description.)

Format: INSPL

Octal code: 140502_8

LACB

Description: Load the A register from the core map-table bank (CMBNK).

(A) ⟵ (ea)

ea = (B) + Δ = CMBNK entry

Δ 3-bit displacement added to B included in the instruction opcode

Format: LACB

Octal code: $1407\Delta2_8$

LASB

Description: Load the A register with the contents of the segment-table bank (STBNK).

(A) ⟵ (ea)

ea = (B) + Δ = STBNK entry

Δ 3-bit displacement added to B included in the instruction opcode.

Format: LASB

Octal code: $1407\Delta0_8$

LBIT

Description: Load single bit accumulator(K) with logical memory bit.

(X) points to the start of a bit array
(A) points to the bit within the array

Format: LBIT

Octal code: 140510_8

LBITP

Description: Load single bit accumulator(K) with physical memory bit.

(T) points to the bank number containing the bit array
(X) points to the start of a bit array
(A) points to the bit within the array

Format: LBITP

Octal code: 140511_8

LBYTP

Description: Load the A register with a byte from physical memory.

(D) points to the bank number containing the byte array
(T) points to the start of a byte array
(X) points to the actual byte within the array

Format: LBYTP

Octal code: 140514_8

LXCB

Description: Load the X register from the core table bank (CMBNK).

(X) ⟵ (ea)

ea = (B) + Δ = CMBNK entry

Δ 3-bit displacement added to B included in the instruction opcode.

Format: LXCB

Octal code: $1407\Delta5_8$

LXSB

Description: Load the X register from the segment table bank (STBNK).

(X) ⟵ (ea)

ea = (B) + Δ = STBNK entry

Δ 3-bit displacement added to B included in the instruction opcode.

Format: LXSB

Octal code: $1407\Delta4_8$

RDUSP

Description: Read a physical memory word without using cache.

(T) points to the physical memory bank to be accessed
(X) points to the address within the bank
(A) is loaded with the memory word

The old content of the memory address is always read from the memory and never from cache.

Note:The execution time of this instruction includes two read-bus cycles (The CPU uses semaphore cycles - see ND-110 Functional Description Manual ND.06.027)

Format: RDUSP

Octal code: 140517_8

REMPL

Description: Remove page from page list. (See Appendix B for a software description.)

Format: REMPL

Octal code: 140503_8

REPT

Description: Enter reentrant segment in page tables. (See Appendix B for a software description.)

Format: REPT

Octal code: 140507_8

RGLOB

Description: Examine global pointers.

(T) ⟵ bank number of segment table (STBNK)
(A) ⟵ start address within bank (STSRT)
(D) ⟵ bank number of core map table (CMBNK)

Format: RGLOB

Octal code: 140501_8

SACB

Description: Store the A register in the core map table bank (CMBNK).

(ea) ⟵ (A)

ea = (B) + Δ = CMBNK entry

Δ 3-bit displacement added to B included in the instruction opcode.

Format: SACB

Octal code: $1407\Delta3_8$

SASB

Description: Store the A register contents in the segment table bank (STBNK).

(ea) ⟵ (A)

ea = (B) + Δ = STBNK entry

Δ 3-bit displacement added to B included in the instruction opcode.

Format: SASB

Octal code: $1407\Delta1_8$

SBIT

Description: Store the single bit accumulator (K) in a logical memory bit.

(X) points to the start of a bit array
(A) points to the bit within the array

Format: SBIT

Octal code: 140512_8

SBITP

Description: Store the single bit accumulator (K) in a physical memory bit.

(T) points to the bank number containing the bit array
(X) points to the start of a bit array
(A) points to the bit within the array

Format: SBITP

Octal code: 140513_8

SBYTP

Description: Store a byte in physical memory.

(D) points to the bank number containing the byte array
(T) points to the start of a byte array
(X) points to the actual byte within the array

Format: SBYTP

Octal code: 140515_8

SETPT

Description:

Set page tables.

This instruction can replace the following instructions:

```
SETPT:  JXZ * 7(8)
        LDDTX 20(8)
        BSET ZRO 130(8) DA
        LDBTX 10(8)
        STD ,B
        LDXTX 00(8)
        JMP *-6(8)
```

Each time the loop is executed (until X becomes zero) two consecutive physical memory locations addressed by X are loaded into the A and D registers.

The word in A is the protect field of the page table, bit 11 (the PGU bit) is cleared to set the page table. The double word (in A and D) is then stored in two consecutive locations pointed to by the contents of the B register, the page table address.

* is the mnemonic for P relative addressing.

Format:

SETPT

Octal code:

140300_8

SZCB

Description:

Store zero in the core map-table bank (CMBNK).

(ea) ⟵ 0

ea = (B) + Δ = CMBNK entry

Δ 3-bit displacement added to B included in the instruction opcode.

Format:

SZCB

Octal code:

$1407\Delta7_8$

SZSB

Description: Store zero in the segment-table bank (STBNK).

(ea) ⟵ 0

ea = (B) + Δ = STBNK entry

Δ 3-bit displacement added to B included in the instruction opcode.

Format: SZSB

Octal code: $1407\Delta6_8$

TSETP

Description: Test and set physical memory word.

(T) points to the physical memory bank to be accessed
(X) points to the address within the bank
(A) is loaded with the word

The contents of the location addressed by T and X are simultaneously loaded into the A register as the location is written to with all 1s. No other memory access is allowed during this operation.
The old content of the memory address is always read from the memory and never from cache. The all 1s' data word is never written to cache.
This instruction can be used for processor synchronization.

Format: TSETP

Octal code: 140516_8

WGLOB

Description: Initialize global pointers.

(T) = bank number of segment table (STBNK)
(A) = start address within bank (STSRT)*
(D) = bank number of core map table (CMBNK)

* must be divisible by 8

Format: WGLOB

Octal code: 140500_8

APPENDIX A GLOSSARY

APPENDIX A GLOSSARY

1s and 2s complement	Binary methods of representing signed numbers.
accumulator	The part of the computer which carries out arithmetical functions.
BCD	Binary Coded Decimal notation also known as packed decimal.
cache memory	Short term memory used to hold instructions and/ data allowing faster execution of repetitive operations.
commercial extended instructions	Instructions which are privileged or for BCD arithmetic...now standard ND-110 instructions.
effective address	The address calculated from the contents of various registers and/or a displacement.
floating point	A method of representing and calculating in binary with a number and an exponent.
general registers	A, B, D, L, P, T, X and STS registers.
lsb	The least significant bit of a number.
mantissa	The most significant bits following the binary point.
microcycle	Internal CPU cycle period.
microprogram	The sequence of micro-instructions executed to perform an instruction.
msb	The most significant bit of a number.
ND-100 family	The family of 16-bit general purpose computers from Norsk Data consisting of the following machines: ND-100 ND-100/CE ND-100/CX ND-100 Compact ND-100 Satellite ND-110

nanocycle	= 26ns
no-operation	An executed instruction which has no effect.
octal	Base 8 representation of digits.
paging	The method used for translating a 16-bit address into a 24-bit address.
physical memory	The memory available to the computer. Paging may be required to address the entire physical memory.
PLANC	Programming Language ND Computers. A high-level systems programming language.
triple word	A 48-bit word.
virtual address	The 16-bit address which can be used to address a larger address range of 24-bits, providing page tables are implemented.
working registers	A, B, D, L, P, T and X registers.

APPENDIX B PLANC LISTINGS OF THE NEW SINTRAN INSTRUCTIONS

APPENDIX B PLANC LISTINGS OF THE NEW SINTRAN INSTRUCTIONS

The following privileged instructions are described in their high-level language (PLANC) form:

CLPT

Clear segment from page tables.

```
WHILE X<>0 DO
  B := ( ((cmbnk,X).3) V 176000 ) * 2
  IF     A<0 THEN
         0 =: B.0
  ELSEIF A>0 THEN
         R3 := B.0
         IF R3<>0 THEN
            R3 =: (cmbnk,X).2
         ENDIF
  ELSE
         R3 := B.0
         IF R3<>0 THEN
            R3 =: (cmbnk,X).2
            0 =: B.0
         ENDIF
  ENDIF
  X := (cmbnk,X).0
  IF interrupt_pending THEN
         P := P-1
         EXIT
  ENDIF
ENDDO
EXIT
```

CNREK

Clear non re-entrant pages.

```
Q := (stbnk,A).2
 IF A=0 THEN
    EXIT
 ENDIF
 R1 := ( (QΛ1700) * 2) + 174000
 DO FOR R2=X TO X+10
    R4 := (T,R2).0
    IF R2 = X+10 THEN
       EXIT
    ENDIF
    DO  FOR lc=0 TO 17
        IF bit(lc,R4) = 1 THEN
           0 =: (R1).0
        ENDIF
        R1 := R1 + 2
    ENDDO
ENDDO
```

ENPT

Enter segment in page tables.

```
WHILE X<>0 DO
      A  := ( (cmbnk,X).2 ) Λ 173777
      R3 := X/4
      B  := ( ( (cmbnk,X).3 ) V 176000 ) * 2
      A  =: B.0
      R3 =: B.1
      X  := (cmbnk,X).0
      IF interrupt_pending THEN
            P := P-1
            EXIT
      ENDIF
ENDDO
EXIT
```

INSPL

Insert page in page list.

```
R1 := (stbnk,B).7
X  =: (stbnk,B).7
R1 =: (cmbnk,X).0
 IF R1<>0 THEN
    Q := (cmbnk,R1).1
    X =: (cmbnk,R1).1
 ELSE
    Q := ( (B - stsrt) / 2 ) + 3
 ENDIF
Q := (cmbnk,X).1
T =: (cmbnk,X).3
```

REMPL

Remove page from page list.

```
R1 := (cmbnk,X).0
R2 := (cmbnk,X).1
 IF R2Λ3 = 0 THEN
    R1 =: (cmbnk,R2).0
 ELSE
    Q := (R2 * 2) + strst
    R1 =: (stbnk,Q).7
 ENDIF
 IF R1<>0 THEN
    R2 =: (cmbnk,R1).1
 ENDIF
0 =: (cmbnk,X).0
0 =: (cmbnk,X).1
```

REPT

```
Enter re-entrant segment in page tables.

WHILE X<>0 DO
      A  := ( (cmbnk,X).2 ) ∧ 073777
      R3 := X/4
      B  := ( ( (cmbnk,X).3 ) V 176000 ) * 2
      A  =: B.0
      R3 =: B.1
      X  := (cmbnk,X).0
      IF interrupt_pending THEN
            P := P-1
            EXIT
      ENDIF
ENDDO
EXIT
```

APPENDIX C ALPHABETIC LIST OF INSTRUCTION MNEMONICS AND THEIR OCTAL CODES

APPENDIX C ALPHABETIC LIST OF INSTRUCTION MNEMONICS AND THEIR OCTAL CODES

			Page
AAA	add argument to A	172400	85
AAB	add argument to B	172000	85
AAT	add argument to T	173000	85
AAX	add argument to X	173400	85
ADD	add to A	060000	65
ADDD	add decimal	140120	101
AND	logical AND to A	070000	67
BANC	AND with bit complement	177000	87
BAND	AND to K	177200	87
BFILL	byte fill	140130	93
BLDA	load K	176600	87
BLDC	load bit complement to K	176400	87
BORA	OR to K	177600	87
BORC	OR with bit complement	177400	87
BSET	bit set	174000	87
BSKP	skip next location if cc	175000	87
BSTA	store and clear K	176200	87
BSTC	store complement and set K	176000	87
CHREENTPAGES	change non reentrant pages	140303	137
CLEPT	clear page tables	140301	137
CLEPU	clear page tables,collect PGU info	140304	138
CLNREENT	clear non reentrant	140302	139
CLPT	clear segment from page tables	140505	139
CNREK	clear non-reentrant pages	140504	139
COMD	compare decimal	140122	102
COPY	register transfer	146100	43
DEPO	memory deposit	150417	130
DNZ	convert FA number to A	152000	71
ELEAV	error leave stack	140137	109
ENPT	enter segment into page tables	140506	139
ENTR	enter stack	140135	109
EXAM	memory examine	150416	130
EXIT	return from subroutine	146142	44
EXR	execute register	140600	45
FAD	add to floating accumulator	100000	68
FDV	divide floating accumulator	114000	68
FMU	multiply floating accumulator	110000	69
FSB	subtract from floating accumulator	104000	69
IDENT	identify interrupt	143600	125
INIT	initialize stack	140134	110
INSPL	insert page in page list	140502	140
IOF	turn off interrupting system	150401	126
ION	turn on interrupting system	150402	126
IOX	input/output	164000	121
IOXT	input/output	150415	123
IRR	inter-register read	153600	114
IRW	inter-register write	153400	114
JAF	jump if A not 0	131400	79
JAN	jump if A -ve	130400	79

			Page
JAP	jump if A +ve or 0	130000	79
JAZ	jump if A 0	131000	79
JMP	jump	124000	78
JNC	increment X;jump if -ve	132400	80
JPC	increment X;jump if +ve	132000	80
JPL	jump to subroutine	134000	78
JXN	jump if X -ve	133400	80
JXZ	jump if X 0	133000	80
LACB	load A with core map table bank	1407Δ2	140
LASB	load A in segment table bank	1407Δ0	140
LBIT	load K flip-flop with logical memory bit	140510	140
LBITP	load K flip-flop with physical memory bit	140511	141
LBYT	load byte	142200	91
LBYTP	load byte from physical memory	140514	141
LDA	load A	044000	60
LDATX	load A with physical memory contents	143300	134
LDBTX	load B with physical memory contents	143303	134
LDD	load double word	024000	60
LDDTX	load D with physical memory contents	143302	134
LDF	load floating accumulator	034000	60
LDT	load T	050000	61
LDX	load X	054000	61
LDXTX	load X with physical memory contents	143301	135
LEAVE	leave stack	140136	111
LRB	load register block	152600	116
LWCS	load writeable control store	143500	131
LXCB	load X with core map table bank	1407Δ5	141
LXSB	load X with segment table bank	1407Δ4	142
MCL	masked clear of register	150200	118
MIN	memory increment;skip if 0	040000	62
MIX3	multiply index by 3	143200	46
MON	monitor call	153000	81
MOVB	move bytes	140131	93
MOVBF	move bytes forward	140132	93
MOVEW	move word block(range 00 to 80 = xx)	1431xx	95
MPY	multiply integer	120000	65
MST	masked set of register	150300	118
NLZ	convert A number to floating in FA	151400	71
OPCOM	set to OPCOM mode	150400	132
ORA	inclusive OR A	074000	67
PACK	convert to packed decimal	140124	103
PIOF	turn paging and interrupt off	150405	126
PION	turn paging and interrupt on	150412	127
POF	turn memory management off	150404	128
PON	turn memory management on	150410	128
RADD	register add	146000	47
RAND	register AND	144400	49
RCLR	register clear	146100	50
RDCR	register decrement	146200	51
RDIV	register div	141600	52
RDUS	read do not use cache	140127	112
RDUSP	read a word without using cache	140517	142
REMPL	remove page from page list	140503	142

			Page
REPT	enter reentrant segment in page tables	140507	143
REX	reset extended address mode	150407	129
REXO	register exclusive OR	145000	53
RGLOB	examine STBNK,STSRT;CMBNK	140501	143
RINC	register increment	146400	54
RMPY	register multiply	141200	55
RORA	register inclusive OR	145400	56
RSUB	register subtract	146600	57
SAA	set argument to A	170400	85
SAB	set argument to B	170000	85
SACB	store A in core map table bank	1407Δ3	143
SAD	shift A and D registers	154600	75
SASB	store A in segment table bank	1407Δ1	143
SAT	set argument to T	171000	38
SAX	set argument to X	171400	85
SBIT	store K flip-flop in logical memory bit	140512	144
SBITP	store K flip-flop in physical memory bit	140513	144
SBYT	store byte	142600	91
SBYTP	store byte in physical memory	140515	144
SETPT	set page tables	140300	145
SEX	set extended address mode	150406	129
SHA	shift A register	154400	75
SHD	shift D register	154200	75
SHDE	decimal shift	140126	104
SHT	shift T register	154000	38
SKP	skip next location on cc	140000	82
SRB	store register block	152402	116
STA	store A	004000	62
STATX	store in A physical memory contents	143304	135
STD	store double word	020000	62
STDTX	store in D physical memory contents	143306	135
STF	store floating accumulator	030000	18
STT	store T	010000	63
STX	store X	014000	18
STZ	store 0	000000	64
STZTX	store in Z physical memory contents	143305	136
SUB	subtract from A	064000	66
SUBD	subtract decimal	140121	105
SWAP	register exchange	144000	58
SZCB	store 0 in core map table bank	1407Δ7	145
SZSB	store 0 in segment table bank	1407Δ6	146
TRA	transfer internal register to A	150000	119
TRR	transfer internal register from B	150100	119
TSET	test and set	140123	112
TSETP	physical test-and-set request	140516	146
UPACK	convert to unpacked decimal	140125	106
VERSN	cpu version	140133	97
WAIT	give up priority	151000	127
WGLOB	initialize global pointers	140500	146

APPENDIX D THE TRR AND TRA INSTRUCTIONS FOR INTERNAL REGISTERS

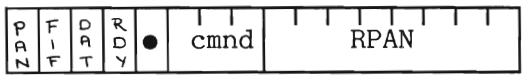
APPENDIX D THE TRR AND TRA INSTRUCTIONS FOR INTERNAL REGISTERS

The A register contents after a TRR and/or TRA instruction(s) are listed below.

TRA reads the contents of the internal register selected into the A register. The following diagrams for TRR <internal register> illustrate the format of the A register contents after the instruction.

TRR writes the contents of the A register into the internal register selected. The following diagrams for TRA <internal register> show the format the A register should take before the instruction is executed and • denotes an insignificant bit.

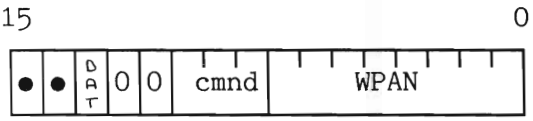
PANS Panel Status Register 0_8



TRA PANS

15	PAN	panel is installed (this is zero if no panel is installed)
14	FIF	FIFO buffer ready for data
13	DAT	last processed command requested data
12	RDY	last command has been completed (this bit is cleared by TRA PANS)
10-8	cmnd	the last command processed
7-0	RPAN	the data requested by the last processed command (if no data was requested, the field contains bits 0-7 of PANC)

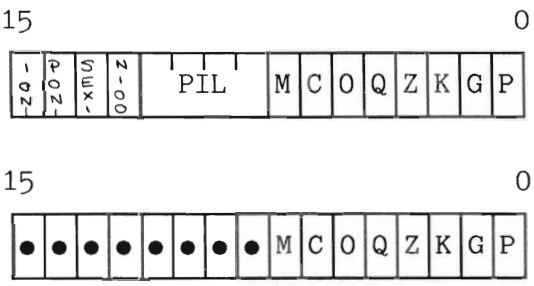
PANC Panel Control Register 0_8



TRR PANC

13	DAT	the command requests data from the panel (data is placed in bits 0-7 of PANS)
10-8	cmnd	panel processor command
7-0	RPAN	data to the panel processor

STS	Status	1_8



15	IONI	interrupt system on flag
14	PONI	memory management on flag (normal mode : 19-bit addresses)
13	SEXI	memory management is in extended mode (24-bit addresses used NOT 19-bit)
12	N100	ND-100 flag (indicates an ND-100 family CPU)
11-8	PIL	current program level
7	P	paging table mode (enables alternate page table mode)
6	G	rounding flag (for floating point operations)
5	K	1-bit accumulator (used for bit operations)
4	Z	error flag
3	Q	dynamic overflow flag
2	O	static overflow flag
1	C	carry flag
0	M	multishift link flag (1-bit extension for the A, D or T register)

OPR Operator Panel Switch 2_8

TRA OPR

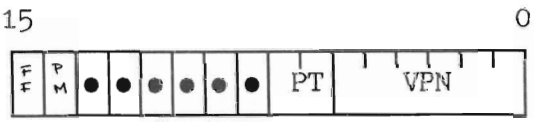
This register is a simulated panel switch register. Data is written into the register by OPCOM operations; TRA OPR can be used to read the the register contents.

LMP Operator Lamp 2_8

TRR LMP

PGS	Paging Status	3_8

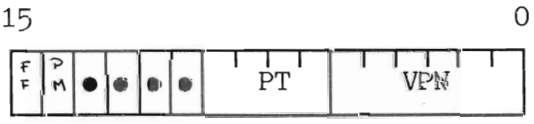
four page table mode:



TRA PGS

15	FF	fetch fault
14	PM	permit violation
7-6	PT	page table number (when violation occured)
5-0	VPN	virtual page number

sixteen page table mode:

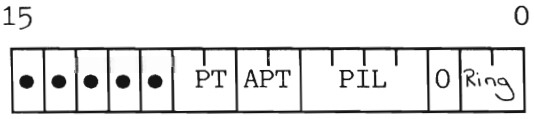


TRA PGS

15	FF	fetch fault
14	PM	permit violation
9-6	PT	page table number (when violation occured)
5-0	VPN	virtual page number

PCR Paging Control 3_8

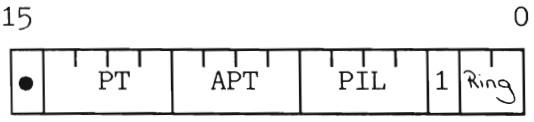
Four page table mode:



TRR PCR

10-9	PT	normal page table
7-8	APT	alternative page table
3-6	PIL	current program level
1-0	Ring	ring protection level (0-2)

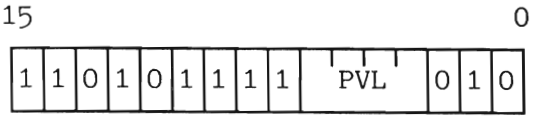
Sixteen page table mode:



TRR PCR

14-11	PT	normal page table
10-7	APT	alternative page table
3-6	PIL	current program level
1-0	Ring	ring protection level (0-2)

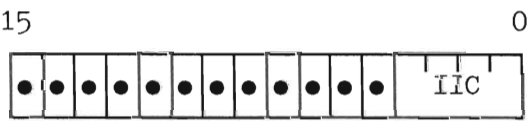
PVL Previous Program Level 4_8



TRA PVL

3-6	PVL	previous program level (0-15)

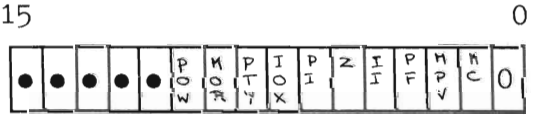
IIC Internal Interrupt Control 5_8



TRA IIC

3-0 IIC code denoting the source of an internal interrupt (see IIE)

IIE Internal Interrupt Enable 5_8



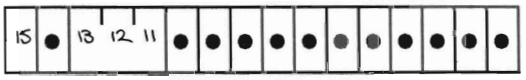
TRR IIE

The bits enable the following internal interrupts:

			IIC code
10	POW	power failure	12_8
9	MOR	memory out of range (or addressing non-existent memory)	11_8
8	PTY	memory parity error	10_8
7	IOX	IOX error (no answer from an external device)	7_8
6	PI	privileged instruction	6_8
5	Z	error flag	5_8
4	II	illegal instruction (instruction not implemented)	4_8
3	PF	page fault (page not in memory)	3_8
2	MPV	memory protect violation (page number is found in the PSR)	2_8
1	MC	monitor call	1_8

PID Priority Interrupt Detect 6_8

15 0

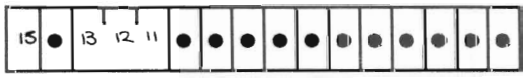


TRA PID

An external interrupt on program levels 15, 13-11 will set the corresponding bit in this register.

PIE Priority Interrupt Enable 7_8

15 0

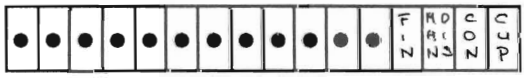


TRR PIE

This register enables external interrupts on program levels 15, 13-11.

CSR Cache Status 10_8

15 0



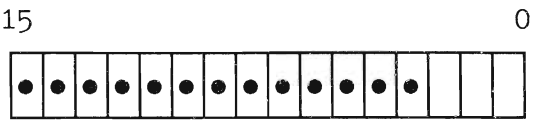
TRA CSR

4	FIN	cache clear finished
3	MAN DIS	cache disabled manually
2	CON	cache on
1	CUP	cache updated on current memory request

CCL Cache Clear 10_8

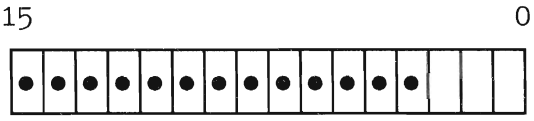
This register has no data. Executing a TRR CCL will exchange the two cache-used bit-maps, so one bit-map can be cleared. (see ND-110 Functional Description ND-06.026.1)

ACTL Active Level 11_8



TRA ACTL

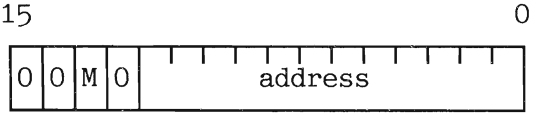
LCIL Lower Cache Inhibit Limit 11_8



TRR LCIL

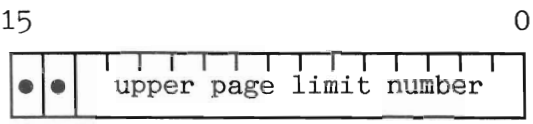
The TRR LCIL sets bits in the cache bit-map (equivalent in function to the setting the lower limit register of the ND-100).

ALD Automatic Load Descriptor 12_8



TRA ALD

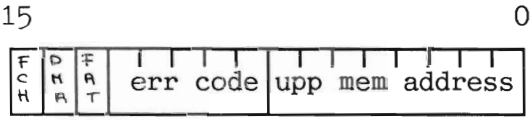
UCIL Upper Cache Inhibit Limit 12_8



TRR UCIL

The TRR UCIL sets bits in the cache bit-map (equivalent in function to the setting the upper limit register of the ND-100).

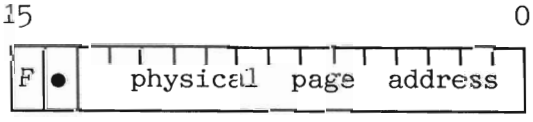
PES Parity Error Status 13_8



TRA PES

15	FCH	error during an instruction fetch
14	DMA	error during DMA reference
13	FAT	fatal error (multiple-bit error)
12-8		error code
7-0		8 msb of the last memory address on the ND-100 bus

CILP Cache Inhibit Page 13_8

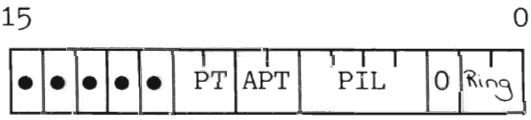


TRR CILP

15	F	page format (1 = normal; 0 = inhibit)
13-0		physical page address of affected page

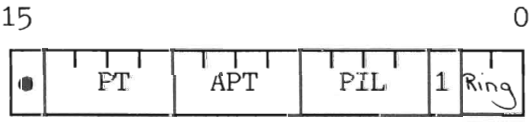
PGC	Paging Control	14_8

Four page table mode:



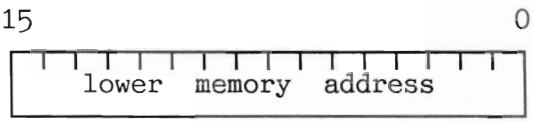
10-9	PT	normal page table
7-8	APT	alternative page table
3-6	PIL	current program level
1-0	Ring	ring protection level (0-2)

Sixteen page table mode:



14-11	PT	normal page table
10-7	APT	alternative page table
3-6	PIL	current program level
1-0	Ring	ring protection level (0-2)

PEA Parity Error Address 15_8



TRA PEA

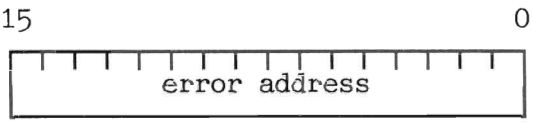
15-0 16 lsb of the physical memory address on ND-100 bus
(at the time of the memory access that caused an interrupt)

ECCR Error Correction Control 15_8

TRR ECCR

4	6TS	simulate memory error in bit 6
3	DIS	disable ECC system and parity interrupt
2	ANY	enable parity interrupt on all errors (reset for only multiple-bit errors)
1	15T	simulate memory error in bit 15
0	0TS	simulate memory error in bit 0

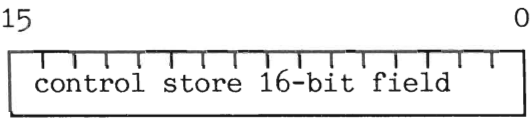
PEA	Parity Error Address	15_8



TRA PEA

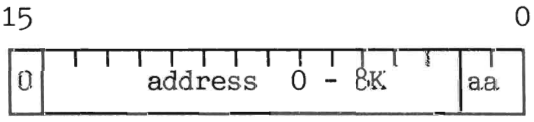
This register contains the 16 lsbs of the address causing a parity error. Reading this register unlocks both PEA and PES.

CS	Control Store	17_8



TRA CS and TRR CS

The control store is 8 K by 64 bits. The X register must be loaded with the control store address before either a TRA CS or TRR CS instruction. The X register should have the following format:



where aa selects one of four 16-bit fields from the addressed 64-bit control store word.

UPDATING

Manuals can be updated in two ways, new versions and revisions. New versions consist of a completely new manual which replaces the old one, and incorporate all revisions since the previous version. Revisions consist of one or more single pages to be merged into the manual by the user, each revised page being listed on the new printing record sent out with the revision. The old printing record should be replaced by the new one.

New versions and revisions are announced in the ND Customer Support Information and can be ordered from the address below.

The reader's comments form at the back of this manual can be used both to report errors in the manual and give an evaluation of the manual. Both detailed and general comments are welcome.

PRINTING RECORD	
PRINTING	NOTES
03/87	Version 1 EN

ND-110 Instruction Set
Publ.No. ND-06.029.1 EN

RING BINDER OR PLASTIC COVER

The manual can be placed in a ring binder for greater protection and convenience of use. Ring binders may be ordered at a price of NKr. 45.- per binder.

The manual may also be placed in a plastic cover. This cover is more suitable for manuals of less than 100 pages than for larger manuals.

Please send your order, as well as all types of inquiries and requests for documentation to the local ND office, or (in Norway) to:

Graphic Center
Norsk Data A.S
P.O.Box 25 BOGERUD
N-0621 OSLO 6 - Norway

I would like to order

..... Ring Binders, B5, at NOK 35,- per binder

..... Ring Binders, A4, at NOK 45.- per binder

..... Plastic Covers, A4, at NOK 10.- per cover

Name: .....

Company: .....

Address: .....

********** 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: ND-110 Instruction Set

Manual number: ND-06.029.1 EN

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 ? _____

_____

NOTE!

This form is primarily for documentation errors. Software and system errors should be reported on Customer System Reports.

Send to:

Norsk Data A.S
Documentation Department
P.O. Box 25, Bogerud
0621 Oslo 6, 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 25, Bogerud
0621 Oslo6, Norway