Test Micro Program Descriptions for ND-500

Test Micro Program Descriptions for ND-500

NOTICE

The information in this document is subject to change without notice. Norsk Data A.S assumes no responsibility for any errors that may appear in this document. Norsk Data A.S assumes no responsibility for the use or reliability of its software on equipment that is not furnished or supported by Norsk Data A.S.

PRINTING RECORD

Printing	Notes
05/81	Version 01
09/81	Version 02

Test Micro Program Descriptions for ND-500
Publ.No.ND-30.013.02

NORSK DATA A.S
P.O. Box 4, Lindeberg gård
Oslo 10, Norway

Manuals can be updated in two ways, new versions and revisions. New versions consist of a complete new manual which replaces the old manual. New versions 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 Bulletin and can be ordered as described below.

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

These forms, together with all types of inquiry and requests for documentation should be sent to the local ND office or (in Norway) to:

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

PREFACE

THE PRODUCT

This manual describes the ND-500 micro test programs:

COMTE	ND-100/ND-500 communication test program
SLICE	ND-500 slice test program
MEMIC	ND-500 cache and memory test program
PREEF	ND-500 prefetch processor test program
ARITH	ND-500 external arithmetic test program
NOMAN	ND-500 no-memory-management test program
GMOFF	ND-500 memory-management-off test program
GMENT	ND-500 memory-management-on test program
TRAPT	ND-500 trap system test program
EXTRA	Extra ND-500 test program

THE READER

This manual is mainly written for the ND-500 production and service staff.

PREREQUISITE KNOWLEDGE

NORD I/O SYSTEMS.

These programs may be run by persons without detailed knowledge of ND-500. If errors are reported, however, a good knowledge of ND-500 hardware is necessary in order to locate and repair the errors.

THE MANUAL

This manual describes how to load and start the test programs. Some useful information about ND-500 hardware (registers etc.) is also included. Each test program describes:
What the separate tests do,
Which break characters to use,
What the U-register contains.

RELATED MANUALS

ND-500 Reference Manual ND.05.009

ACKNOWLEDGEMENT

The author is grateful for the willingness of the hardware chaps to supply the information necessary to the writing of this manual.

TABLE OF CONTENTS

Section Page

Section Page

Section Page

Section Page

Section Page

Section Page

Section Page

1. Introduction.

The micro test programs test the ND-500 by executing IOX-instructions. They do not use the interrupt system.

The micro test programs work in two ways:
They test the communication interfaces (3022, 5015) by executing IOX-instructions from the ND-100 (ND-500 is passive). In this way, the first testing of the ND-100/ND-500 communication is done.
They test ND-500 by loading small micro programs into the control store, starting them, and checking the results. These micro programs are always loaded from the specified minimum control store address. Utility micro programs, like dump programs, initiation programs, etc., are loaded into the uppermost part of the control store. The micro programs are relocatable.

The programs consist of sets of subroutines, each testing a small part of ND-500. There are two different kinds of subroutines:
Verification routines that run tests, check results, and report errors, if any.
Test routines, intended to run repeatedly and to be used together with oscilloscopes, logic probes, etc., to locate errors. As input to these routines there is a simulated OPR, with the possibility to flip (change from 0 to 1 to 0 to 1 ...) any of the bits in it.

Each program may run in one-by-one-mode (one routine runs over and over again), or in all-mode where all routines are run in sequence, one after another.

All the programs may also be run in sequence (one after another). This is intended for week-end runs, for instance. The programs may be started friday nights, and the results collected monday morning.

The error messages are assumed to be self-explanatory.

A list of some abbreviations that occur in the text.

2. A list of some abbreviations that occur in the text.

Some of the abbreviations are micro code mnemonics and should be looked for in the mnemonics list.

ALT. Alternative addressing mode.

BOU. Data bus on the 3022 card.

CDB. Internal bus for the ND-500 CPU cards.

DBU. A bus between the 3022 card and the 5015 card.

DOUB. Data bus on the 3022 card.

DUEN. A decoded value in the TAGIN register on the 5015 card. It enables the least significant part of the DATA-OUT register (IODOUT).

DUT. A bus on the 5015 card.

MM. Memory Management system.

MOST. Bit 7 in the TAGOUT register on the 5015 card. Will select the most or the least significant part of some 32 bit registers.

OPR. Operator register. Will be displayed on the ND-100 front panel if the user types U/2F cr on terminal 1 when this is in monitor mode. It is used by most routines to display test information.

TE. Trap enable register.

TSB. Translation Speedup Buffer. Belongs to the memory management system.

WA. Write Address register on the 5015 card. It holds a control store address.

A list of some abbreviations that occur in the text.

3. A short list of registers, IOX instructions etc.

The interface between the ND-100 and the ND-500 consists of 2 interface cards; the 3022 card on the ND-100, and the 5015 card on the ND-500. These cards contain several registers, which are listed below.

3.1. The CONTROL word register on 3022.

Bit:	Meaning:
0	Enable interrupt from ND-500
1	Not used
2	Activate ND-500 operation (and lock the communication)
3	Test mode
4	ND-500 programmed clear
5	Disable TAG-IN decoding when locked
6	DMA error
7	Command chaining
8-14	ND-500 operation
15	Not used

3.2. The STATUS register on 3022.

Bit:	Meaning:
0	Interrupt enabled
1	Not used
2	ND-500 busy
3	ND-500 finished
4	Error
5	Interface locked
6	DMA error
7	ND-500 power fault (set by micro program). The stop bit is set
8	ND-500 power is/has been off
9	ND-500 micro clock has stopped
10-14	ND-500 stop reason
15	CONTROL register bit 15

3.3. The memory address register (MAR) on 3022.

This is a 24 bit register, pointing to the ND-100 memory. It is used in DMA transfers. It must be loaded from the 16-bit A-register in two operations. The most significant part is loaded first. It must also be read in two operations. The least significant part will be read first. When it is read, the upper half of the leftmost 16 bits of MAR (bits 24-31, not used) will be equal to the upper half of the rightmost 16 bits (bits 8-15).

3.4. The DATA register on 3022.

This is a 16 bit register. It acts as an intermediary between the ND-500 and the ND-100 memory in DMA transfers from ND-500 to ND-100. In DMA transfers from ND-100 to ND-500, the DATAX register is used as the intermediary register, but the DATA register is set, nonetheless.

3.5. The DATAX register on 3022.

This 16-bit register connects the bus DOUB with the bus BOU. It is also used in DMA transfers from ND-100 to ND-500. Do not confuse it with the DATA register.

3.6. The DATA-IN register on 5015.

This 32-bit register is either used as a whole, or as DATA-IN-1 (the lower 16 bits), and DATA-IN-2 (the uppermost 16 bits). When the other registers on the 5015 cards are loaded from ND-100, data goes via the DATA-IN register to the CDB bus. In DMA read (ND-100 memory read by ND-500), data will go to the DATA-IN register. The MOST bit selects the most or least significant part.

3.7. The DATA-OUT register on 5015.

This 32-bit register is either used as a whole, or as DATA-OUT-1 (the lower 16 bits), and DATA-OUT-2 (the uppermost 16 bits). When the other registers on 5015 are read from ND-100, data goes via DATA-OUT to ND-100. In DMA write (ND-500 to ND-100), data must be placed in DATA-OUT before the write. The MOST bit selects the most or least significant part.

3.8. The BREAK register on 5015.

This 16-bit register is used when the control store is loaded. Data to be loaded must be in the BREAK register. The BREAK register is connected to the least significant part of the CDB bus.

3.9. The write address register (WA) on 5015.

The 16-bit WA register is used to hold the control store address when loading and reading the control store. The WA register is connected to the least significant part of the CDB bus.

3.10. The lower and upper limit registers (LL, UL) on 3022.

These are 16-bit registers, and represent bits 8-23 of a DMA address. They are compared with bits 8-23 of the MAR register to ensure that ND-500 keeps within limits. For instance, if LL contains 1, and UL contains 3, the legal area for DMA transfers is 0400, 0401, ... , 01376, and 01377.

3.11. The control register (CSCNT) on 5015.

Bit:	Name:	Meaning:
0	CSLOAD	Control store load
1	CSREAD	Control store read
2-5	WE0,WE1,WE2,WE3	Control store group (0-8)
6	BRKEN	BREAK enable
7	STADREN	Start address enable
8	TSTPTY	Test control-store-parity-checking (ND-500 passive)
9	TSTTGU	Returns TAG-OUT instead of TAG-IN
10	CSPTY	Control store parity
11	AFIN	Prefetch addr. calc. not finished
12	PFIN	Prefetch instruction not finished
13	BALUM	Memory reference not finished
14-15		Not used

Bits 10-15 may only be read. They give micro program stop conditions.

3.12. The TAG-IN register on 5015 (I/O from ND-100).

The tag registers are additional control registers used to control the communication.
Bits 0-3 in the TAG-IN register on 5015 give 16 code values. Bit 4 is not used, and bit 5 (octal 040) is used to return TAG-IN bits (0-4). The codes are:

Bit:	Name:	Meaning:
0		Not used
1	DICLK1	Clock DATA-IN-1 register
2	DICLK2	Clock DATA-IN-2 register
3	DUCLK	Clock DATA-OUT register (both)
4	WACLK	Clock write-addr register
5	BRKCLK	Clock BREAK register
6	TGUCLK	Clock TAG-OUT register
7	CNTCLK	Clock CSCNT register
8	DIEN	Enable DATA-IN register to bus (CDB)
9	DUEN	Enable DATA-OUT register (least sign.)
10	WAR	Read write-addr register
11	BRKR	Read BREAK register
12	CNTR	Read CSCNT register
13	RESBRK	Reset break
14	DUNL	Unlock
15	EDUTEN	Enable data line driver (from ND-500)

3.13. The TAG-OUT register on 5015 (data from ND-500).

Bits 0-2 in the TAG-OUT register on 5015 give 8 code values.
Bit 3 means ND-100 if it is 0, and not ND-100 if it is 1.
Bits 4-6 are not used.
Bit 7 is the MOST bit. It enables the most significant part of the DATA-OUT register, and determines which part of the register to use when micro programmed. MOST also controls least/most significant part of the DATA-IN register. The codes are (for MOST=1, add 0200):

Bit: Meaning:

0 Read memory address register
1 Write memory address register
2 Read STATUS register
3 Write STATUS register
4 Read CONTROL register
5 Reset activate
6 Read DATA register (and ND-100 memory)
7 Write DATA register (and then into ND-100 memory)

3.14. IOX instructions.

The ND-500 communication can be locked or unlocked, in test mode or not in test mode. These states are set by IOX LCON (load CONTROL register). IOX instructions have different meanings, depending on the state. In the following list, the three columns display the MAC mnemonics of physical device numbers, the octal device numbers themselves, and their meaning.

Locked and not in test mode:

RSTA 062 Read STATUS register
MCLR 066 ND-500 Master Clear
TERM 067 Terminate
RTAG 070 Read TAG-IN
WTAG 071 Write TAG-OUT
WDAT 073 Write DATAX (NB not the DATA register)
SLOC 074 Set locked
CLKD 075 Clock DATA
UNLC 076 Release locked (unlock)
RETG 077 Return tag

Locked and in test mode:

RSTA 062 Read STATUS register
RCON 064 Read CONTROL register

A short list of registers, IOX instructions etc.

Unlocked and not in test mode:

```
RMAR 060  Read memory address register
LMAR 061  Load memory address register
RSTA 062  Read STATUS register
LCON 065  Load CONTROL register
MCLR 066  ND-500 Master Clear
TERM 067  Terminate
RTAG 070  Read TAG-IN
WTAG 071  Write TAG-OUT
WDAT 073  Write DATAX (NB not the DATA register)
SLOC 074  Set locked
UNLC 076  Release locked (unlock)
RETG 077  Return tag
```

Unlocked and in test mode:

```
RMAR 060  Read memory address register (do it twice)
LMAR 061  Load memory address register (do it twice)
RSTA 062  Read STATUS register
LSTA 063  Load STATUS register
RCON 064  Read CONTROL register
LCON 065  Load CONTROL register
MCLR 066  Read DATA register
TERM 067  Load DATA register
RTAG 070  Read upper limit register
WTAG 071  Load upper limit register
RLOW 072  Read lower limit register
WDAT 073  Load lower limit register
```

ND-100 bits 0-15 go to limit register bits 8-23.

3.15. Some widely used communication subroutines.

The routines that follow below are written in MAC (assembly) code.

3.15.1. Master clear, set stop bit, reset tag bits.

```
IOX   UNLC     % unlock
SAA   040
IOX   LCON
SAA   2
IOX   RETG     % set stop bit
IOX   MCLR
SAA   0
IOX   WTAG     % write TAG-OUT on 3022
SAA   044
IOX   LCON     % activate
IOX   UNLC
SAA   040
IOX   LCON     % reset activate
EXIT
```

3.15.2. Write tag from the A register.

```
IOX   WTAG     % write TAG-out on 3022
SAA   044
IOX   LCON     % activate
IOX   UNLC
SAA   040
IOX   LCON     % reset activate
EXIT
```

3.15.3. Write data to 5015 from the A register.

The following routine uses the most/least significant part of the DATA-IN register, depending on the value of n (DATA-IN-1 is the least significant part):

```
IOX   WDAT     % A register to DATAX
SAA   n        % n=1: clock DATA-IN-1.  n=2: clock DATA-IN-2
IOX   WTAG
SAA   044
IOX   LCON     % activate
IOX   UNLC
SAA   040
IOX   LCON     % reset activate
SAA   010      % enable DATA-IN to the CDB bus on 5015
IOX   WTAG
SAA   044
IOX   LCON     % activate
IOX   UNLC
SAA   040
IOX   LCON     % reset activate
EXIT
```

3.15.4. Read data from 5015 to the A register.

The following routine has 3 entry points. The first does not enable the DATA-OUT register (DUEN). The third does not clock the CDB bus to the DATA-OUT register.

```
ENTR1=*

SAA   3
IOX   WTAG     % clock CDB to DATA-OUT
SAA   044
IOX   LCON     % activate
IOX   UNLC
SAA   040
IOX   LCON     % reset activate
JMP   COMON
```

A short list of registers, IOX instructions etc.

```
ENTR2=*

SAA   3
IOX   WTAG     % clock CDB to DATA-OUT
SAA   044
IOX   LCON     % activate
IOX   UNLC
SAA   040
IOX   LCON     % reset activate

ENTR3=*

SAA   011
IOX   WTAG     % enable DATA-OUT
SAA   044
IOX   LCON     % activate
IOX   UNLC
SAA   040
IOX   LCON     % reset activate

COMON=*

SAA   017
IOX   WTAG     % enable data line driver (DUT to DBU)
SAA   044
IOX   LCON     % activate
IOX   CLKD     % clock DATA on 3022
IOX   UNLC
SAA   050
IOX   LCON     % set test mode
SAA   0
IOX   MCLR     % read DATA (test mode)
STA   SAVE
SAA   040
IOX   LCON
SAA   0
IOX   WTAG     % reset tag bits
SAA   044
IOX   LCON
IOX   UNLC
SAA   040
IOX   LCON     % reset activate
LDA   SAVE
EXIT
```

3.16. Subroutines to write and read the control store.

The control store address is supposed to be in the WA register. The part number is a number in the range 0-010. A control store word consists of 9 16-bit words, and the part number points to one of these 9 words. Part number 010 (8) points to the most significant part. Data to be written must be in the BREAK register. Data that is read will appear in DATA-OUT-1. The WA register is set by the sequence

```
LDA  ADDR; JPL  WRDAT; SAA  4; JPL WRTAG
```

3.16.1. Write a 16-bit word into the control store.

The A register contains the 16 bit data word. The T register contains a control word that is 1, 5, 011, 015, ... , 041 depending on the part number (0-010).

```
STA   SAVE
COPY  SL DA
STA   LINK
LDA   SAVE
JPL   WRDAT     % data to the CDB bus on 5015
SAA   5
JPL   WRTAG     % clock the BREAK register
COPY  ST DA
JPL   WRDAT     % control word to the CDB bus
SAA   7
JPL   WRTAG     % clock the CSCNT register
LDA   SAVE
JMP   I LINK
```

3.16.2. Read a 16-bit word from the control store.

The A register contains a control word that is 2, 6, 012, 016, ..., 042 depending on the part number (0-010).

```
STA   SAVE
COPY  SL DA
STA   LINK
LDA   SAVE
JPL   WRDAT     % control word to the CDB bus
SAA   7
JPL   WRTAG     % clock the CSCNT register
JPL   ENTR3     % read data, already in DATA-OUT
JMP   I LINK
```

3.17. Other registers used by the test programs.

3.17.1. The prefetch status register (PSTAT, 32-bit, read only).

Bits:	Name:	Meaning:
0-10	EP	Operation code. Bit 10 is 0: short operation code. Bits 8-9 are then both zero. Bits 0-7 contain 252 different operation codes, complemented, and not 256. The codes 111111xx, where x is 1 or 0, do not exist for short codes. When the six most significant bits are one, it means long operation code. Bit 10 is 1: long operation code. Bits 0-9 contain 1024 different operation codes, complemented. A long operation code consists of 16 bits. The six most significant bits are 1, and, since EP is 11 bits long, 5 of them are discarded.
11-14	PCD	Program counter displacement. Gives the length (complemented) of the current instruction. 017 means 1 byte, 016 2 bytes, and so on.
15-16	VLB	Valid bytes. 3 means 4 bytes left in the instruction buffer, 2 means 3 bytes left, and so on.
17-19	OPTYP	Operand type. From 0 to 5: word, float, halfword, byte, bit, and double float.
20	REGOP	Register operand. 1 if the address code (first byte of operand specifier) was 0320-0323, otherwise 0.
21	CONOP	Constant operand. 1 for constant operands as, for instance, in argument instructions, otherwise 0.

22 DESC Descriptor addressing.

0 if legal, otherwise 1.

23 WR Write operation.

1 if write operation, otherwise 0.

24 Not used.

25 PFIRST First operand.

1 for the first operand, otherwise 0. Becomes 0 as soon as the first operand has been fetched. For a sequence of LDR instructions, for instance, it will be 1 all the time.

26-27 DX Descriptor register.

Used in descriptor adressing to give the number of the register to use. 3 means R1, 2 means R2, 1 means R3, and 0 means R4.

28-29 SXSEL Source register select.

Gives the number of the source register, when there is one. 3 means R1, and so on.

30-31 DXSEL Destination register select.

Gives the number of the destination register, when there is one. 3 means R1, and so on.

3.17.2. The (trap) status register S1.

This is a 32 bit register. Only bits 9-31 can give a trap. If one of bits 9-29 is to give a trap, the corresponding bit must be set in the trap enable (TE) register.

Bit:	Meaning:
0	Not used
1	Privileged instruction allowed
2	Part done
3	Instruction reference
4	Process switch disable
5	Zero
6	Carry
7	Sign
8	Flag
9	Overflow
10	Not used
11	Invalid operation
12	Divide by zero
13	Floating underflow
14	Floating overflow
15	BCD overflow
16	Illegal operand value
17	Single instruction trap
18	Branch trap
19	Call trap
20	Breakpoint instruction trap
21	Address trap fetch
22	Address trap read
23	Address trap write
24	Address zero access
25	Descriptor range
26	Illegal index
27	Stack overflow
28	Stack underflow
29	Programmed trap
30	Disable process switch timeout
31	Disable process switch error

If bits are going to be set in S1 by software, two mnemonics can be used. D,XST1 must be used to set the bits 17-19, 21-24, or 30-31. D,S1 must be used to set the other bits.

3.17.3. The (trap) status register S2.

This is a 12 bit register.

Bit:	Meaning:
0	Index scaling error
1	Illegal instruction code
2	Illegal operand specifier
3	Instruction sequence error
4	Not used
5	Activate from ND-100
6	Terminate from ND-100
7	Not used
8	Instruction failure (PV, MOR, CPE, MME, MSE, PGE)
9	Data failure
10	Power fail
11	Processor fault

3.17.4. The memory and cache registers.

The cache length is always 4K. The width may be 32, 64, or 128 bits. This corresponds to (byte) address ranges of 0-037777, 0-077777, and 0-177777. If one cache module is present, the width is 32 bits. If 2, the width is 64 bits, and if 4 modules are present, the width is 128 bits.

The whole cache may be used (partitions 0-3). Two partitions may be used, 0-1, 1-2, or 2-3. Only one partition may be used, 0, 1, 2, or 3. The use of the cache is controlled by the data and instruction memory control registers. There are also status registers to display the status of the instruction and data cache.

3.17.4.1. Data memory status registers (DSTS0, DSTS1, DSTS2).

DSTS0

Bits:	Meaning:
0-1	Partition number
2-3	Number of partitions (0-3 means 1-4)
4	TSB-fault
5	Memory parity error
6	Cache parity error + illegal use of cache
7	Blocked. If this bit is 1, then bits 8-15 in DSTS0 and bits 12-15 in DSTS1 will be blocked (they will not change).
8	Cache parity error, cache module 0.
9	" " " " " 1.
10	" " " " " 2.
11	" " " " " 3.
12	Memory " " " " 0.
13	" " " " " 1.
14	" " " " " 2.
15	" " " " " 3.

DSTS1

Bits:	Meaning:
0	Memory parity error, byte 0 (bits 7- 0).
1	" " " " 1 (" 15- 8).
2	" " " " 2 (" 23-16).
3	" " " " 3 (" 31-24).
4	Cache " " " 0 (" 7- 0).
5	" " " " 1 (" 15- 8).
6	" " " " 2 (" 23-16).
7	" " " " 3 (" 31-24).
8-9	Cache module number (0-3).
10	Memory timeout.
11	Illegal partition setting.
12	Cache control parity error, byte 0.
13	" " " " " 1.
14	" " " " " 2.
15	Cache clear is active.

DSTS2

Bits:	Meaning:
0-7	Memory channel 0-7. If bit 10 in DSTS1 is 1, then some of the bits 0-7 will also be 1.

3.17.4.2. Data memory control registers (DCON0, DCON1).

DCON0

Bits:	Meaning:
0-1	Select (the first) partition number
2-3	Number of partitions (0-3 means 1-4)
4	Cache disable (must be zero)

DCON1

Bits:	Meaning:
0-1	Select cache module no. for bits 0-7, DSTS1.
2	HIC (hit in cache).
3	Clear block.
4	TSB trap enable.
5	Memory parity error trap enable.
6	Cache parity error trap enable.
7	Memory out of range trap enable.

3.17.4.3. Instruction memory status registers (ISTS0, ISTS1, ISTS2).

These registers have the same format as the data memory status registers.

3.17.4.4. Instruction memory control registers (ICON0, ICON1).

These registers have the same format as the data memory control registers.

3.17.5. Memory modus register (MMOD).

Bit:	Meaning:
0	Alternative address area (default).
1	Alternative address area selected by ALTMOD.
2	Lock until write (not used yet)
3	Data do not use cache
4	Instruction do not use cache
5	Instruction memory reference from micro code

3.17.6. Limit registers (HL,LL).

These higher and lower limit registers contain 32 bit logical addresses. They are constantly compared to logical program and data addresses, and may give a trap condition if the proper address traps are enabled.

To get an address trap, the proper bit in TE must be set to 1. In addition, if the address of a memory reference (fetch, read or write) is called ADDR, trap depends on the value of D,SETLIM:

D,SETLIM is 0: LL<ADDR.AND.ADDR<=HL is true gives trap
D,SETLIM is 1: LL<ADDR.OR.ADDR<=HL is true gives trap

When the ND-500 has memory management, and bits 31-27 in ADDR are all zero, the segment register will be taken as the uppermost 5 bits of ADDR in the comparison with LL and HL.

3.17.7. Memory management substitute registers.

ND-500 may be without memory management. Then, there will be some additional registers:

DZPA and IZPA: Data and instruction memory zero point adjust registers. They are 14-bit registers and contain page numbers. A page has 2K bytes. These registers point to the physical page in the memory where the first page of the program itself is loaded.

DUPL and IUPL: Data and instruction memory upper page limit register. They are similar to DZPA and IZPA, and point to the program's last physical page in the memory.

DCINHLL, DCINHLU, ICINHLL, and ICINHLU: Data and instruction memory cache inhibit limit registers, lower and upper. They are similar to DZPA and IZPA, and inhibit write into the cache memory when the actual program's physical page number is in the range lower to upper (LL <= pageno <= LU).

DRADDRL, DRADDRM, IRADDRL, IRADDRM. Data and instruction memory least and most significant real (physical) address registers. DRADDRL and IRADDRL contain 16 bits, and DRADDRM and IRADDRM contain 8 bits. A real address is a 24-bit byte address (a real address has actually 25 bits, but the most significant bit is removed). The page number in DZPA/IZPA multiplied by 04000 is added to a program's logical data and instruction addresses, and the result goes to the real address registers. If errors occur, the real address registers are locked (that is, new real addresses will not be loaded into them before the clear-block bit in DCON1/ICON1 is set).

3.17.8. Memory management registers.

There are two sets of these registers, one for the data memory and one for the instruction memory.

A real address is a logical address translated by the memory management system. The translated address is then shifted one position to the right, thereby discarding bit 0. The real address is therefore a halfword address.

3.17.8.1. Scratch files (ISCRF,DSCRF).

These are two sets of 16 16-bit registers. Such a register is addressed by loading ISCFA or DSCFA with a number in the range 0-15. After each access, ISCFA or DSCFA is incremented by 1, modulo 16.

3.17.8.2. Status registers (IMSTS,DMSTS).

Bit:	Name:	Meaning:
0	PALT	0: ALT mode. Locked by TSB-fault.
1	SMM0	0: SEGEQ (same segm). Locked by TSB-fault. The segment register and bits 31-27 of the logical address are equal.
2	SMM1	0: SEGZ (zero segm). Locked by TSB fault. Bits 31-27 of the logical address are zero.
3	PUS	1: Real-addressed page is used.
4	WIP	1: Real-addressed page is written into.
5	USED	0: Used. Dynamic USED-status of the hashed part of TSB. Only valid if bit 13=0.
6	TSBF	1: TSB-fault (PON=0: 0. PON=1: 1 if bit 5=1 or not match).
7	NEWS	0: New segment (1 when DMSTS). Bits 31-27 of the logical address are not all zero, and they are not equal to bits 4-0 of the segment register.
8	MMTR	1: MM-trap (locked. Inclusive or of bits 6, compl(7), 9, 10,23).
9	ALTPV	1: ALT protect violation.
10	WRPV	1: Write protect violation.
11	PON	1: Paging on.
12	TSBC	1: TSB clear is active (not completed).
13	FAS2A	1: Match not found in sequential TSB, if TSB fault. Sequential TSB is accessed only if TSBF = 1 and if FAS2 = 1 (in IPROCC/DPROCC) and if USED = 0 (in actual hashed TSB entry)
14	SPARE	Not defined.
15	SPARE	Not defined.
16	SP0	1: Parity error 0 (PROC0-2, DOM0-4).
17	SP1	1: Parity error 1 (DOM5-7, SEG0-4, AD19-26).
18	SP2	1: Parity error 2 (AD11-18).

19	SP3	1: Parity error 3 (BSG0-15). Page number + two dummy bits.
20	SP4	1: Parity error 4 (the three permit bits). See ICSEG/DCSEG, bits 5-7.
21	SPARE	0
22	SPARE	0
23	BUFFP	1: OR-ed parity error (0 if PON=0 or not used).
24	TEQ0	0: Match on PROC and DOM bit 0-4.
25	TEQ1	0: Match on SEGM (or bits 27-31) and DOM bit 5-7.
26	TEQ2	0: Match on log. addr. bits 11-18.
27	TEQ3	0: Match on log. addr. bits 19-26.
28	USED	0: used. Static USED-status of the hashed part of TSB.
29	SPARE	Not defined.
30	SPARE	Not defined.
31	SPARE	Not defined.

Locked bits are unlocked when the memory management is turned off, or when the TSB is written into.

3.17.8.3. Logical address (ILADDR,DLADDR).

These two 32-bit registers hold the instruction and data logical addresses.

3.17.8.4. WIP/PGU broadside (IWIPGU,DWIPGU).

A broadside is a 16-bit extract from a 16K bit buffer. There are two such buffers, one for WIP (written in page) and one for PGU (page used). The 16 bits represent one group of 16 pages. Each group is addressed by means of the 10 most significant bits of the real address. Bit 0 represents the page with the lowest page number of the 16, bit 15 represent the page with the highest page number. To read WIP or PGU, bit 9 in IMCNTR or DMCNTR has to be set. Then bit 7 in IPROCC or DPROCC selects either WIP or PGU. If 1, WIP is selected, and if 0, PGU. Default for this bit is 0.

3.17.8.5. Real address (IRADDR,DRADDR).

These two 24-bit registers hold the instruction and data real addresses. A real address is a logical address translated by the memory management system, and then divided by 2. The result is a halfword address.

3.17.8.6. Control registers (IMCNTR,DMCNTR).

Bit:	Meaning:
4	Clear ITSB or DTSB.
9	Start to read IWIPGU or DWIPGU.

3.17.8.7. Scratch file address (ISCFA,DSCFA).

Two 4-bit registers, each pointing to one of the 32 scratch file registers (16 in each set).

3.17.8.8. Process control registers (IPROCC,DPROCC).

Bit:	Name:	Meaning:
0	PROC0	Bit 0 of process number.
1	PROC1	Bit 1 of process number.
2	PROC2	Bit 2 of process number.
3	PON	Paging on.
4	TSBD	Disable TSB. 1: writing into TSB, 0: reading.
5		Not used.
6	FAS2	Enable use of sequential TSB (STSB).
7	SWIP	Select WIP-part of IWIPGU/DWIPGU (default 0).

3.17.8.9. Domain registers (IDOMR,DDOMR).

Two 8-bit registers, containing the main domain number (0-255). In the ND-500 Reference Manual, DOMR is called CED (Current Executing Domain).

3.17.8.10. Alternative domain registers (IADOM,DADOM).

Two 8-bit registers, containing the alternative domain number (0-255). In the ND-500 Reference Manual, ADOM is called CAD (Current Alternative Domain).

3.17.8.11. Current segment registers (ICSEG,DCSEG).

Two 8-bit registers, containing the current segment number in bits 0-4, and the protect status in bits 5-7.

Bit:	Meaning:
5	0: Shared segment status. 1: Not shared
6	0: Parameter access permitted. 1: Not permitted
7	0: Write permitted. 1: Not permitted

In the ND-500 Reference Manual, CSEG is called CES (Current Executing Segment).

3.17.8.12. Alternative segment registers (IASEG,DASEG).

Similar to current segment registers, but containing alternative segment number and status. In the ND-500 Reference Manual, ASEG is called CAS (Current Alternative Segment).

3.17.8.13. Translate speed-up buffer page (ITSB,DTSB).

Two buffers, each contains 768 entries. Each entry contains:

3 process number bits
8 domain number bits
8 segment number bits (3 of these are protect status bits)
16 logical address bits (logical address bits 26-11)
14 page number bits (the page part of the real address)
2 dummy bits

One buffer is for data and one for instruction memory. Each buffer is divided into two sections. The lower section, with 512 entries, is addressed by a hashing algorithm, and the upper, with 256 entries, is addressed sequentially.

The hashing algorithm computes a 9-bit index by EXCLUSIVE OR-ing three numbers A, B, C. In the following, if AD31-27 are all zero, SEG4-0 comes from the segment register, bits 4-0. If AD31-27 are not all zero, SEG4-0 comes from AD31-27 (the five most significant bits of the logical address).

A:	0	0	0	DOM0	DOM1	DOM2	DOM3	DOM4	DOM5
B:	1	SEG0	SEG1	SEG4	SEG3	SEG2	PROC2	PROC1	PROC0
C:	AD19	AD18	AD17	AD16	AD15	AD14	AD13	AD12	AD11

3.17.8.14. Sequential TSB address register (ISTSB,DSTSB).

Two 8-bit registers. Top of sequential buffer. 0 means that the sequential buffer is empty, 0377 means that it is full (255 entries). ISTSB/DSTSB must be set and updated by software (micro program).

3.17.8.15. Index for hashed or sequential TSB (IHXA,DHXA).

The lower 8 bits of the 9-bit index may be read and checked. There is one index for instruction memory, and one for data. Either the computed index for the hashed part of TSB is read, or ISTSB/DSTSB. This depends upon the value of bit 13 (FAS2A) of the status register (IMSTS/DMSTS). If this bit is 1, ISTSB/DSTSB is read. If it is 0, the computed index for the hashed part is read.

The most significant bit of IHXA/DHXA can not be read. It is assumed to be the opposite of AD19, when FAS2A=0.

A short list of registers, IOX instructions etc.

4. How to use the programs.

The programs may be run in three different ways: without SINTRAN (stand-alone), one by one with SINTRAN, or all programs in sequence, with SINTRAN.

4.1. Stand-alone.

To run the programs without SINTRAN, a diskette with the programs in BPUN format (as written on the diskette by MAC with the MAC command)BPUN) must be used. Insert the diskette in the floppy disk drive, unit 0. Press MASTER CLEAR and type 1560& on terminal 1. The floppy monitor will be read from the diskette and respond by printing an asterisk. Type LOAD COMTE, LOAD SLICE, LOAD MEMIC, etc., etc., followed by carriage return, and the selected program will start.

4.2. SINTRAN.

4.2.1. Loading.

Before the programs can be run under SINTRAN, they must be loaded as RT-programs by the RT-loader. A diskette with the programs in BPUN format is needed. Insert the diskette in the floppy disk drive. The loading procedure is the same for each program. In the following, NAME is used as the name of the RT-program, d=directory, u=user, and nnn is the segment number:

```
@ENT-DIR,,F-D-1,0          Response: @
@RT-LOADER                 Response: heading, asterisk.
*SET-P-T,2                 Response: asterisk.
*CL-SEG,nnn                Necessary to remove previous versions.
Y                          Response: asterisk.
*NEW-SEG,nnn,2,DM,,,       Response: segment number message.
*READ-BIN,(d:u)NAME,,      Response: asterisk.
*SET-L-A,nnn,177777        Response: asterisk.
*DEC-P,NAME,,              Response: asterisk.
*END                       Response: asterisk.
*SET-P-T,2                 Response: asterisk.
*CH-RT-D,NAME,10,nnn,,0,   Response: asterisk.
*EX                        Response: @
@REL-DIR d
```

and the loading of the program is complete.

4.2.2. Running one program at a time.

To start one of the programs, log in as user SYSTEM or RT on terminal 1. Type the commands

```
@RT,NAME
@LOG
```

and the program will start on terminal 1, when it is free to use.

It is also possible to log in as user RT on any terminal and start the background program STMIC (short for STart MICro test program). Before this can be done, the file STMIC:BPUN must have been loaded and dumped by the commands
@PLACE,STMIC and
@DUMP,STMIC,0,0
by the user RT. STMIC will ask for which RT program to start, and then it will start this RT program directly on the terminal of the user.
STMIC uses the file N500-TEST-PARAM:DATA.

4.2.3. Running all programs in sequence.

It is possible to run all the programs in sequence, one after another. This requires the loading and starting of another RT-program called HAREM. This program is able to start any of the micro test programs. When HAREM is started, it will ask for all parameters needed, put them on a file, and start (for instance) COMTE. When COMTE has completed a run, it will give control back to HAREM, which will then start SLICE, and so on. When the last one is done, it will start the first one again. The programs will take their parameters from the file which has the name:-

N500-TEST-PARAM:DATA

If the file does not exist, HAREM will create it as a continuous file with one page and save the parameters on it. This means that user RT must have some free pages.

4.3. Break characters.

The program execution may be interrupted by break characters, which act as commands. When a character is typed on the terminal, the program will stop normal execution and respond with the appropriate action, perhaps after a slight delay. After the proper action for the break character is carried out, the program will continue automatically, or ask for another break character. The break characters are letters only, with one exception:
If escape is typed, the program will halt and wait for the next break character. If any character other than a break character is typed, an error message will be printed, and the program continues.
Escape may also be typed as a response to questions from the programs. Another break character must then be typed. This may be useful when the user does not want to answer the question, but instead wants to restart the program, etc.
The break characters are:

C Octal control store dump. Will ask for first and last address of the control store area to be dumped. Each control store location is dumped as an address and 9 16-bit numbers. After this, the program will be in one-by-one mode.

D Display current test subroutine (symbolic). Lists the symbolic ND-500 micro code if the test is micro programmed. If not, pseudo assembly code is listed.

E End of program.

F Flip (switch) simulated OPR bit. Will ask for bit number of the bit to flip. After this, the specified bit in the simulated OPR will be alternately 1 or 0.

G ND-500 register dump. A micro program will be started. It will read and send the contents of most of the ND-500 registers to ND-100 for dumping. After this, the program will be in one-by-one mode.

I Initialise the program. The program is restarted at its very beginning.

L LMP. The value of the user register is dumped. Useful in the case where ND-100 is without a front panel. This is a 16-bit register.

M ND-500 memory dump. The first and last ND-500 memory address of the area to dump will be requested. A micro program will be started. It will read from the ND-500 memory and send to ND-100 for dumping. After this, the program will be in one-by-one mode.

N Start the next test subroutine.

O OPR. The value of simulated OPR will be printed out.The program will not continue before a new value, or CR (carriage return) is given. CR means that the old value will remain unchanged. The simulated OPR is a 32 bits register. Most of the routines in

O OPR. The value of simulated OPR will be printed out.The program will not continue before a new value, or CR (carriage return) is given. CR means that the old value will remain unchanged. The simulated OPR is a 32 bits register. Most of the routines in COMTE, though, will use only the lower part of OPR.

P Start the previous test subroutine.

R Repeat the current test subroutine. After this, the program will be in one-by-one-mode.

S Stop. Returns to the last question in the initiation. After this question has been answered, a memory initiation routine will set some memory registers and clear the cache. This is very useful whenever the power has been turned off.

T Trace micro program addresses. Will trace an already loaded micro program, and print out the addresses of the micro program as it is executed. It asks for start address. A sequence of micro addresses, with no jumps, will be dumped as two numbers, the first and the last of the sequence. It is also possible to specify a break address, and to have printed the last 0100 micro addresses before the break. After this, the program will be in one-by-one mode.

U User micro program. The user may enter his own micro program, or modify an already loaded one, and then start the program. Before the program is started, the ND-500 DATA-IN register is loaded with simulated OPR. If ND-500 unlocks, the micro address is printed, and the ND-500 DATA-OUT register is read by ND-100 and dumped in three different formats. The micro program is then continued. After this, the program will be in one-by-one mode. The user micro program is described in a separate chapter.

W Write error messages. Used to turn on error messages in one-by-one-mode.

X Exclude error messages. Used to turn off error messages in one-by-one-mode.

esc Wait for the next break character.

4.4. The A (all) and O (one-by-one) mode.

A program can be run in A mode or O mode. A means that the routines in the program are all run, one after another. When all the routines have been executed, the process is repeated. O means that only one routine is run, repeatedly. This continues until a break character is typed, for instance N or P. When the programs are run under HAREM control, the mode is always A. When the mode is A mode, and a routine has printed the maximum number of error messages, the next routine is entered immediately.

4.5. The user register.

The user register may be displayed on the ND-100 front panel. The different subroutines use this register for the display of test values, addresses, etc. This may be helpful when the tests are run, because one is able to follow the progress of each test. The contents of the user register is described for each test.

4.6. Recommended executing sequence.

It is important that the programs are run in a proper sequence. The first run should be COMTE, and maximum control store address should be specified as zero. The communication between ND-100 and ND-500 will then be tested only from ND-100. After this, SLICE should be run, to check that the sequencing works. When this is OK, COMTE should be run with the maximum control store address different from zero, to test the communication from the ND-500 side also. After this, SLICE should be run, and MEMIC, if ND-500 has memory, and then the remaining programs. At present, the maximum control store address is usually specified as 010000.

4.7. Stop on full page.

It is possible to make the program halt when a full page has been written on the terminal. A full page has 22 lines. To continue, the user must type one character. The program will then continue. The character will be ignored by the program and cannot be used as input. If the user wishes to type a break character when the program has written a full page, he should first type the escape character, and then the break character.

5. The user micro program.

5.1. Description of the commands.

When the user types the break character U, a special program is entered. It makes it possible for the user to create, modify, and start his own micro program. This special program has its own set of commands:

C	Control store dump
D	Disassembly of micro instructions
G	Get back saved micro program (see P)
H	Help (list the commands).
K	Continual master clear
L	Look-at control store or memory
M	Mnemonics list
P	Put octal micro program into the save area
R	Run
S	Symbolic assembly
esc	Exit. Waits for break character

When the program is ready for a command, it prints U: .
One of the command letters should then be typed by the user. If a letter is typed before the program is ready for a command, it will be taken as a break character. In this way, the OPR register may be changed while the user micro program is executed. Below follows a description of the commands:

C This is the usual control store dump.

D The contents of the control store will be disassembled into symbolic micro instructions. The first and last control store addresses will be requested. It is possible to output the disassembled micro instructions to a file, if the test program runs under SINTRAN. The file name will be requested.

G Gets back a saved octal micro program. There is a memory area in ND-100 with room for 0200 (128) micro instructions. If a micro program has been saved (Put) there, it will be loaded back into the proper addresses in the control store.

H Help. Will print a list of commands.

K Continual master clear (please excuse the K, as C was already used!). This routine will first call the master clear routine, and then remove the stop bit. Then, it will give continual master clear with a frequency determined by the OPR register. OPR=0 will give the highest frequency, and OPR=077777 will give the lowest. Continual master clear is very useful for debugging, as it starts the micro program in address 0 over and over again.

L Look-at control store (C), instruction memory (I), or data memory (D). If the next character typed is C, this can be used to inspect and maybe change the octal contents of the control store. The routine will ask for control store address, and then print out the contents of the control store, one part at a time. After an octal number has been printed, the user can change it by typing another number, followed by CR (carriage return). If no change is wanted, CR must be typed. This proceeds until the user types a point (full stop) instead of a new number or CR. The program will then ask for the next command.

If the next character typed is I or D, the instruction or data memory may be inspected and/or changed. The routine will first initiate some memory registers and clear the cache. After this, the procedure is the same as for look-at control store.

M This command will produce the list of the micro code mnemonics, together with their values. There are over 720 such mnemonics.

P Puts an octal micro program into the save area, which is an area in the ND-100 memory with room for 128 micro instructions. The routine will request the first and last control store addresses. It is very useful to save an octal micro program while the ND-500 is being debugged. If the user wants to put a printed circuits board on an extension board, for instance, the power must be turned off, and his micro program will disappear. Therefore, it should first be saved by typing P.

R Run a micro program. The start address will be requested. Before the micro program is started, the OPR register is loaded into IODIN (the DATA-IN register), and then the micro program runs until it is stopped by the user, or until ND-500 unlocks. When ND-500 unlocks, ND-100 will print out the micro address of the next instruction and the value of IODOUT (the DATA-OUT register) in three different formats, word, halfword, and byte. After this, ND-100 will load the value of OPR into IODIN and continue the micro program. The user can stop all this by typing the break character U. Control will then be returned to the user program. Remember that the OPR register may be changed at any time during the run of a micro program, but IODIN will not be loaded with this value unless ND-500 unlocks.

S Symbolic assembly. The control store address will be requested. The user types mnemonic codes, separated by space, and the program will translate them to octal. When a semicolon is typed, the octal micro instruction will be loaded into the control store. Long arguments and jump addresses must be typed as two 16 bit octal numbers. For instance, a long argument may be typed as 6,6 The jump address 5 must be typed as 5,0. A short argument, for instance 3, can be typed as 0,3 or only as 3. After the last semicolon, a point (full stop) will end the assembly. An incorrect mnemonic will be ignored, or it may be erased (before the separating space is typed) by typing ctrl W. If the same mnemonic is typed twice, the whole micro instruction is deleted (set to zero).
It is possible to assemble symbolic files from the disk, if the test program runs under SINTRAN. The file name will be requested. In this case, comments (%) are skipped during the assembly. If the file is not ended by a full stop (.), end-of-file will simulate it.

5.2. Some useful micro instructions.

```
ALU,ADIR A,XD,IODIN SLOW1 D,IODOUT NEXT; % This loop can be used to
UNLOCK NEXT;                             % check the bus from IODIN
W,XD NEXT;                               % through ALU to IODOUT by
JMP BACKTHREE;                           % changing simulated OPR.

ALU,ADIR A,XD,SARG SLOW1 2 D,TAG NEXT;   % read STATUS to IODIN-1
W,IO NEXT;

ALU,ADIR A,XD,SARG SLOW1 3 D,TAG NEXT;   % write IODOUT-1 to STATUS
W,IO NEXT;
```

In the next example, the MOST bit (0200) selects IODIN-2.
The two parts of MAR will be read alternately, and are not dependent on the MOST bit.

```
ALU,ADIR A,XD,SARG SLOW1 0 D,TAG NEXT;   % least sign MAR to IODIN-1
W,IO NEXT;
ALU,ADIR A,XD,SARG SLOW1 0200 D,TAG NEXT;% most sign. MAR TO IODIN-2
W,IO NEXT;
ALU,ADIR A,XD,IODIN SLOW1 D,IODOUT NEXT;
ALU,ADIR A,XD,SARG SLOW1 0201 D,TAG NEXT;% IODOUT-2 to MAR
W,IO NEXT;
ALU,ADIR A,XD,SARG SLOW1 1 D,TAG NEXT;   % IODOUT-1 to MAR
W,IO NEXT;

ALU,ADIR A,XD,SARG SLOW1 6 D,TAG NEXT;   % read 16 bit from ND-100
W,IO NEXT;                               % MAR was already set
ALU,ADIR A,XD,IODIN SLOW1 D,IODOUT NEXT; % 16-bit word to IODOUT
ALU,ADIR A,XD,SARG SLOW1 7 D,TAG NEXT;   % write to next ND-100 word
W,IO NEXT;                               % MAR was autom. incr.
```

In the examples above, with tag codes 0 to 7, it is the least significant part of IODOUT that is used. If the most significant part is wanted, 0200 must be added (0200 to 0207).

```
ALU,A-B A,X#0 B,X#0 SET COND,MZRO NEXT;  % condition is now true
C,SEQ JMP PREVIOUS F,NEXT;
UNLOCK NEXT;
W,XD NEXT;                               % stop
JMP BEGIN;                               % continue here after stop

ALU,ADIR A,XD,SARG SLOW1 0100 D,LC NEXT; % loop counter:=0100
LCDECR NEXT;                             % decrement loop counter
ALU,ADIR A,XD,LC SLOW1 D,IODOUT NEXT;
UNLOCK NEXT;
W,XD NEXT;
JMP BACKFOUR;                            % jump to new decrement
```

In the following example, the contents of X#0 is written to the memory. X#1 holds the memory address.

```
ALU,ADIR A,XD,SARG SLOW1 0 D,MMOD NEXT;              % data memory
ALU,ADIR A,X#0 PASSAB AB,IX IX1 MEM,WR4 W,MEM NEXT;

ALU,ADIR A,XD,SARG SLOW1 040 D,MMOD NEXT;            % instr mem.
PASSAB AB,IX IX1 AD,ILC NEXT;
ALU,ADIRC A,X#0 SLOW1 D,IDAT MEM,WR4 W,MEM PASSAB NEXT;
```

Next, the contents of the memory word whose address is in X#1, is read to X#0.

```
ALU,ADIR A,XD,SARG SLOW1 0 D,MMOD NEXT;              % data memory
PASSAB AB,IX IX1 MEM,RD4 W,MEM NEXT;
ALU,ADIR A,DATA D,X#0 NEXT;

ALU,ADIR A,XD,SARG SLOW1 040 D,MMOD NEXT;            % instr mem.
PASSAB AB,IX IX1 AD,ILC NEXT;
A,XD,IDAT XDMOV D,CONST MEM,RD4 W,MEM PASSAB NEXT;
ALU,ADIRC A,XD,CONST SLOW2 D,X#0 NEXT;
```

Next, the B-operand is converted from byte to halfword:

```
ALU,BDIR B,X#0 TYP,BY BYTH SLOW1 D,IODOUT NEXT;
```

Please remember that a lot of small micro programs may be studied by typing D when the test programs are running!

6. COMTE - the ND-100/ND-500 communication test program.

6.1. General information.

COMTE tests the communication between ND-100 and ND-500. The first 22 routines are not verification routines. The first 39 routines are not micro programmed. They just use IOX instructions from the ND-100, the ND-500 being passive. If the user wants to know what the different routines do, the break character D will dump the symbolic micro code of the test, or pseudo assembly code. The break character C may then be used to inspect the octal micro program. It is always loaded from the minimal control store address.

6.2. How to load and start the program.

Stand-alone: MASTER CLEAR. 1560&. *LOAD COMTE.

SINTRAN: @RT COMTE and possibly @LOG, or @STMIC.

COMTE first asks whether the user wants to run on another terminal. If the answer is Y, COMTE will request a decimal logical device number (SINTRAN), or an octal physical device number (stand-alone).

If the program runs under SINTRAN, it will reserve the ND-500 and possibly some memory (COMTE needs no ND-500 memory). COMTE will ask all the necessary questions.

COMTE asks for maximum control store address. Answer 0, and the micro programmed routines will not be used. Answer 02000, 010000, etc, depending on the hardware configuration, and tests both with ND-500 passive and ND-500 active (micro programmed) will be run.

COMTE asks for minimum control store address. The answer should be less than maximum control store address. If this address is different from zero, the micro programs will be relocated when they are loaded.

COMTE asks for stop on full page. If you want the program to pause when 22 lines have been printed on the terminal, you should type Y, otherwise N. stop on full page is useful when you are using a screen terminal.

COMTE asks for TEST, VERIFY, OR BOTH ?
If the answer is T, only routines 1 to 22 are run.
If the answer is V, only routines 23 to 49 are run.
If the answer is B, all routines are run.

COMTE asks if you want information. If the answer is Y, a list, describing the two modes and the break characters, is printed.

COMTE asks if the tests are to be run all in succession, or one by one. If the answer is the letter O, COMTE will ask for the routine number, and if error messages are wanted. Then the routine with the given number will be run repeatedly.
If the answer is A, all routines will be run in succession in a ring.

The execution of the program may be interrupted and changed at any time by the typing of break characters.

6.3. The test routines.

6.3.1. TST01 Continual master clear.

Will give master clear (IOX MCLR) continually. The frequency depends on simulated OPR. This routine may be used to start a micro program in address 0 again and again, if the stop bit is off.
The user register is not used.

6.3.2. TST02 Set and reset stop bit.

Will switch (flip-flop) the stop bit (IOX RETG) continually (from bit 1 in the A-register).
The user register is not used.

6.3.3. TST03 Set and reset activate.

Will switch (flip-flop) the activate bit (IOX LCON) continually (from bit 2 in the A-register).
The user register is not used.

6.3.4. TST04 Set and reset reverse tag bus bit.

Will switch (flip-flop) the reverse tag bus bit (IOX RETG) continually (from bit 0 in the A-register).
The user register is not used.

6.3.5. TST05 Test 3022 control register (bits 3-2 always 10).

Will load the control register with the contents of simulated OPR, and then read it back and display it in the user register. Bit 3 is always set to 1, and bit 2 is always set to 0.

6.3.6. TST06 Test 3022 status register.

Will load the status register with the contents of simulated OPR, and then read it back and display it in the user register.

6.3.7. TST07 Test 3022 memory address register.

Will load MAR with the contents of simulated OPR (the lower 24 bits of the 32 available), and then read it back and display it in the user register. If the lowermost 16 bits of the read back value are all zero, the uppermost 8 bits will be displayed. If not, the lowermost 16 bits are displayed.

6.3.8. TST08 Test 3022 data register.

Will load the data register with the contents of simulated OPR, and then read it back and display it in the user register.

6.3.9. TST09 Test 3022 lower limit register.

Will load the lower (DMA) limit register with the contents of simulated OPR, and then read it back and display it in the user register.

6.3.10. TST10 Test 3022 upper limit register.

Will load the upper (DMA) limit register with the contents of simulated OPR, and then read it back and display it in the user register.

6.3.11. TST11 Test tag dataway.

Simulated OPR is used as tag code (write-tag). The tag code is read back (IOX RTAG) and displayed in the user register.

6.3.12. TST12 Test DATA-IN to DATA-OUT.

Data is taken from simulated OPR, written to 5015 via the DATA-IN register, and then read from the DATA-OUT register. Only the least significant part is tested. The result is displayed in the user register.

6.3.13. TST13 Test DATA-IN to WA-reg to DATA-OUT.

Data is taken from simulated OPR, written to 5015 via the DATA-IN register, and the WA (write-address) register is clocked. Then the WA register is enabled, and data is read from the DATA-OUT register. The result is displayed in the user register.

6.3.14. TST14 Test DATA-IN to BREAK-reg to DATA-OUT.

Data is taken from simulated OPR, written to 5015 via the DATA-IN register, and the BREAK register is clocked. Then the BREAK register is enabled, and data is read from the DATA-OUT register. The result is displayed in the user register.

6.3.15. TST15 Test DATA-IN to CSCNT-reg to DATA-OUT.

Data is taken from simulated OPR, written to 5015 via the DATA-IN register, and the CSCNT register is clocked. Then the CSCNT register is enabled, and data is read from the DATA-OUT register. The result is displayed in the user register.

6.3.16. TST16 Test TAG-OUT on 5015.

Data is taken from simulated OPR, written to 5015 via the DATA-IN register, and the TAG-OUT register is clocked. Then bit 9 (TSTTGU, 01000) is written to 5015 via the DATA-IN register, and the CSCNT register is clocked. IOX RETG (return tag) with the A-register equal to 3, and IOX RTAG (read tag) are executed. The result is displayed in the user register.

6.3.17. TST17 Test DATA-IN to DATA-OUT, most significant 16 bits.

Data is taken from simulated OPR and written to 5015 via the most significant part of the DATA-IN register. Then the MOST bit (0200) is written to 5015 via the least significant part of the DATA-IN register, and the TAG-OUT register is clocked. Data is read back from the DATA-OUT register without using the DUEN bit. The result, the most significant part of the DATA-OUT register, is displayed in the user register.

6.3.18. TST18 Test control signals for load control store.

The BREAK register is filled with a 16 bit micro code word (at present, it is 0). Then simulated OPR is put into the CSCNT register.
The user register is not used by this test.

6.3.19. TST19 Test control signals for read control store.

Simulated OPR is put into the CSCNT register. Then the DATA-OUT register is read and displayed in the user register.

6.3.20. TST20 Test write-and-read one 16-bit word in the control store.

The user has to specify a control store address and a part number in the range 0-8. After that, 9 16-bit words, all zero, are written into the specified location in the control store, and the specified address is put into the WA register. Then a loop is started, writing simulated OPR into the control store location with the specified part number in the specified address, reading it back, and displaying it in the user register.

6.3.21. TST21 Test control signals for start (from stop mode).

The stop bit is set, simulated OPR is put into the CSCNT register, and the stop bit is reset.
The user register is not used by this test.

6.3.22. TST22 Test break (by setting WA=BREAK).

The stop bit is set, simulated OPR is put into the WA register and the BREAK register, and the break enable bit (0100) is put into the CSCNT register.
The user register is not used by this test.

6.4. The verification routines.

6.4.1. TST23 Verify 3022 DATA register.

The DATA register is loaded and read by the sequence
SAA 010; IOX LCON; LDA DATA; IOX TERM; SAA 0; IOX MCLR
The result is checked, and error messages are printed where applicable.
The user register displays the read-back value. It will be continually incremented.

6.4.2. TST24 Verify 24 bits memory address register.

The MAR register is loaded twice by IOX LMAR and read twice by IOX RMAR. The most significant part is loaded first, and the first part that is read back, is assumed to be the least significant part. The result is checked, and error messages are printed where applicable.
The user register displays the least significant part of the read-back value. It will be continually incremented.

6.4.3. TST25 Verify 3022 CONTROL register (Bit 2=0. Bit 4=1 clears bit 6).

The CONTROL register is loaded and read by IOX instructions. The result is checked, and error messages are printed where applicable. Bit 2 (the activate bit) is always 0 in this test.
The user register displays the read-back value. It will be continually incremented.

6.4.4. TST26 Verify STATUS register (not bits 0, 5, 011 and 017).

The STATUS register is loaded and read by IOX instructions. The result is checked, and error messages are printed where applicable. Some of the bits are not part of the test, as they cannot be set or reset by IOX LSTA.
The user register displays the read-back value. It will be continually incremented.

6.4.5. TST27 Verify 3022 (DMA) lower limit register.

The LL register is loaded and read by IOX instructions. The result is checked, and error messages are printed where applicable.
The user register displays the read-back value. It will be continually incremented.

6.4.6. TST28 Verify 3022 (DMA) upper limit register.

The UL register is loaded and read by IOX instructions. The result is checked, and error messages are printed where applicable.
The user register displays the read-back value. It will be continually incremented.

6.4.7. TST29 Verify tag dataway.

This routine will write tag, read it back, and check the result. Error messages will be printed where applicable.
The user register will display the value that was read back (it will be continually incremented).

6.4.8. TST30 Verify dataway, least significant 16 bits.

The least significant 16 bits of the data path from ND-100 to DATA-IN to DATA-OUT to ND-100 will be verified. Error messages will be printed where applicable.
The user register will display the value that was read back. It will be continually incremented.

6.4.9. TST31 Verify WA register.

Data is sent from ND-100 to DATA-IN to WA to DATA-OUT to ND-100, and then checked for correctness. Error messages are printed where applicable.
The user register will display the read-back value. It will be continually incremented.

6.4.10. TST32 Verify BREAK register.

Data is sent from ND-100 to DATA-IN to the BREAK register to DATA-OUT to ND-100, and the checked for correctness. Error messages are printed where applicable.
The user user register will display the read-back value. It will be continually incremented.

6.4.11. TST33 Verify CSCNT register.

Data is sent from ND-100 to DATA-IN to the CSCNT register to DATA-OUT to ND-100, and then checked for correctness. Error messages are printed where applicable.
The user register will display the read-back value. It will be continually incremented up to 02000.

6.4.12. TST34 Verify TAG-OUT.

Data is sent from ND-100 to TAG-OUT. Then bit 9 (TSTTGU, 01000) is put into the CSCNT register, and TAG-OUT is read back and checked. Error messages are printed where applicable.
The user register will display the read-back value. It will be continually incremented.

6.4.13. TST35 Verify data least significant 16 bits controlled by MOST bit.

Zero is put into the TAG-OUT register, to ensure that the MOST bit is off. After this, data is sent from ND-100 to DATA-IN to DATA-OUT to ND-100, without using the DUEN bit. The result is checked, and error messages printed where applicable.
The user register displays the read-back value. It will be continually incremented.

6.4.14. TST36 Verify data most significant 16 bits controlled by MOST bit.

The MOST bit (0200) is put into TAG-OUT. Then data from ND-100 is written into the most significant part of the DATA-IN register. The DATA-OUT register is read back to ND-100 without using the DUEN bit, and the result is checked. Error messages are printed where applicable.
The user register will display the read-back value. It will be continually incremented.

6.4.15. TST37 Verify control store.

This routine will check the specified part of the control store. First, it will do an address in address test. Then it will write a test pattern once and read it 4 times, and check for errors. Error messages are printed where applicable. There are 4 different patterns. Every time the pattern is written into a control store location (144 bits), it is rotate shifted 1 left. This means that within one 144-location block, no two locations are alike. Every time a 144-bit word is read from the control store, the CSPTY bit in the 5015 CSCNT register is checked.
The user register will display the control store address, both for write and read. For write, bit 15 (017) will be set to 1.

6.4.16. TST38 Verify load of WA, BREAK, CONTROL, TAG-OUT, and read back a lot of times.

The TAG-OUT, WA, BREAK and CONTROL registers on 5015, are loaded with 017, 036000, 140300, and 01460, respectively, and then they are read back many times. The results are checked, and error messages are printed where applicable.
The user register displays a counter.

6.4.17. TST39 Verify DATA-IN to DATA-OUT, 32 bits.

The value of a counter is sent from ND-100 to the least significant part of DATA-IN. The complement of this counter is sent to the most significant part of DATA-IN. The least significant part of DATA-OUT is read, and then the most significant part. The results are checked, and error messages are printed where applicable.
The user register displays the least significant part of the counter.

6.5. The micro programmed routines.

6.5.1. TST40 Verify write-STATUS on 3022 from ND-500 (not bits 017, 011-7, 5, 0).

This routine loads a small micro program into the control store and starts it. The micro program loads the STATUS register on 3022, and then unlocks ND-500. ND-100 then reads the STATUS register and checks it. Error messages are printed where applicable. If ND-500 does not unlock, a timeout message is printed.
The user register displays the STATUS register, as it is read back by ND-100. It will rotate shift left a single 1, or a single 0. Some of the bits in the STATUS register are not part of the test, as they are not controlled by a simple STATUS load from ND-500.

6.5.2. TST41 Verify write MAR on 3022 from ND-500.

A small micro program is loaded and started. It will load the memory address register from ND-500 and then unlock. ND-100 reads and checks the register, and prints error messages where applicable.
The user register displays the least significant part of the memory address register as it is read by ND-100. It will rotate shift left a single 1, or a single 0.

6.5.3. TST42 Verify store-in-memory (write-DATA) from ND-500.

ND-100 loads the memory address register. When the program is run under SINTRAN, it is loaded with 0; 017. When it is run stand-alone, it is loaded with 0; addr, where addr is an address somewhere in the program area. Then a micro program is loaded and started. This will do a DMA transfer (write) of one word, and then unlock. ND-100 will fetch the result from the DATA register and check it. If OK, ND-100 will read the memory address register and check that it has been incremented by 1. If OK, ND-100 will fetch the result from the memory and check it. Where applicable, error messages will be printed.
The user register will display the DATA register as it is read by ND-100. It will rotate shift left a single 1 or a single 0.

6.5.4. TST43 Verify read-and-write-STATUS on 3022 from ND-500 (not bits 017, 011-7, 5, 0).

A micro program is loaded and started. ND-500 will read the STATUS register, complement it, write it back, and then unlock. ND-100 will check that the STATUS register has been complemented, and print error messages where applicable.
The user register will display the value of the STATUS register as it is read by ND-100. It will rotate shift a single 1 or a single 0.

6.5.5. TST44 Verify read-and-write-MAR on 3022 from ND-500.

A micro program is loaded and started. ND-500 will read the memory address register, complement it, write it back, and unlock. ND-100 will check that the register has been complemented. Where applicable, error messages are printed.
The user register will display the least significant part of the memory address register as it is read by ND-500. It will rotate shift left a single 1,or a single 0.

6.5.6. TST45 Verify read-CONTROL-and-write-MAR on 3022 from ND-500 (not bits 6-3, always bit 2).

A micro program is loaded and started. ND-500 will read the control word from 3022 and write it back to the least significant part of MAR, and unlock. ND-100 will check the result. Error messages are printed where applicable.
The user register will display the value of the least significant part of MAR as it is read from ND-100. It will rotate shift a single 1 or a single 0. Bits 3-5 have to be 0 for this test, and bit 2 has to be 1.

6.5.7. TST46 Verify read-and-write-DATA on 3022 from ND-500.

ND-100 first loads the memory address register. Then a micro program is loaded and started. ND-500 will read a 16-bit word from the ND-100 memory (via the DATAX register), decrement the memory address register by 1, complement the 16-bit word and write it back (via the DATA register), and then unlock. ND-100 will check that the memory address register is incremented by 1, and that the data word is complemented.
The user register will display the data word as it is read by ND-100. It will rotate shift a single 1 or a single 0.

6.5.8. TST47 Verify DATA-IN to DATA-OUT, 32 bits, and then DATA-OUT-2 to DATA-OUT-1.

The most significant part of DATA-IN is loaded with the value of a counter. A micro program is loaded and started. ND-500 will copy DATA-IN to DATA-OUT, move the most significant part of DATA-OUT to the least significant part (thereby destroying the most significant part), and then unlock. ND-100 will check the results and print error messages where applicable.
The user register displays the counter as it is loaded into DATA-IN.

6.5.9. TST48 Verify lower and upper (DMA) limit registers during DMA transfer.

The LL register is loaded with 0 and the UL register is loaded with 177777. After this, the MAR register is loaded with all values from 0,0 to 0377,177000 with an increment of 0400. The program checks that the status register, bit 6, is always 0 (legal DMA). Then the limit registers are loaded with 177777 and 0 (no DMA is legal). The MAR register is loaded with all values from 0,0 to 0377,177400. The program checks that the status register, bit 6, is always 1 (illegal DMA). After this, the LL and UL registers are loaded with 0 and 1. This means that DMA transfers into the memory area from 0 to 0377 are legal. A micro program is then started. It will do three DMA transfers (read) from address 0376, 0377, and 0400. Before each DMA transfer, the micro program will read S2 (ND-500 status register 2). ND-100 will check that the two first have bit 6=0 (legal), and that the last has bit 6=1 (illegal). The user register will display the uppermost 16 bits of the MAR register as it is loaded.

6.5.10. TST49 Micro programmed moving control store test.

This is a fairly big micro program. It consists of a little less than 400 micro instructions. When it is loaded and started, it will write into and read from the control store three different patterns. They are address+partnumber in address, complement of address+partnumber in address, and a fixed pattern with a mixture of ones and zeros. When the control store has been written and read with all three patterns, ND-500 unlocks, and ND-100 loads the micro program from the next higher control store address and starts it again. If ND-500 detects errors, it will unlock, and ND-100 will print error messages.
The user register will display the load address of the micro program.

COMTE - the ND-100/ND-500 communication test program.

7. SLICE - the ND-500 slice test program.

7.1. General information.

SLICE tests the ND-500 slice and sequencing: single step, sequencing through single stepping, bit mask, ALU functions, registers, etc. Only the first routine is a test routine, and the rest are verification routines. All routines test with the help of micro programs. All micro programs are loaded from the minimum control store address specified by the user in the initiation of the program. They are relocatable. After SLICE has been started, its execution may be modified and altered by typing break characters (one-letter commands).

7.2. How to load and start the program.

Stand-alone: MASTER CLEAR. 1560&. *LOAD SLICE.

SINTRAN: @RT SLICE and possibly @LOG, or @STMIC.

SLICE asks for some parameters. These questions are similar to the questions asked by COMTE, with one exception. Maximum control store address should not be specified as zero. See the chapter on COMTE.

7.3. The test routines.

7.3.1. TST01 Test single step.

This is not a verification routine. It tests single step by putting the first control store address into the WA and BREAK registers. Then it puts the break enable and start address enable bits (6-7, 0300) into the CSCNT register. It activates ND-500 and removes the stop bit. After that, the program unlocks ND-500, sets the stop bit, resets the break and start address enable bits, and resets break.
The user register is not used by this routine.

7.4. The verification routines.

7.4.1. TST02 Verify sequencing (single step a lot of NEXT).

This routine starts by filling the specified part of the control store with the micro instruction NEXT; Then the micro program is started in the first control store address, and the micro program is single stepped through every micro instruction. This is done 8 times. ND-100 reads the CSA bus (control store address bus) and checks that the address has been incremented by 1 for each single step. Error messages are printed where applicable.
The user register will display the control store address. When the micro program is loaded, the user register will count with an increment of 0100. When the micro program is executed, it will count with an increment of 1.

7.4.2. TST03 Verify sequencing (single step a lot of JMP *-1).

The control store is filled with the micro instruction JMP *-1; (jump to the previous location). In the first location, a jump to the last micro instruction in the program is inserted. When this is done, the micro program is started in its first address. ND-100 single steps through the entire micro program 8 times. It reads the CSA bus (control store address bus) and checks that it has been decremented by 1. Error messages are printed where applicable.
The user register counts with an increment of 0100 when the micro program is loaded, else it counts with an increment of -1.

7.4.3. TST04 Verify sequencing (single step a lot of JSR SUB).

The control store is filled with the micro instruction JSR SUBR1; except for the first 4 and the last 3 locations. In the first 4 locations are loaded two subroutines jumping to other subroutines. In the last 3 are loaded two subroutines. The first one jumps to the next, and the second only returns. In this way, ND-100 will, when it is single stepping through the micro program, check the 4 level subroutine jump facility. ND-100 checks this by reading the CSA bus (control store address bus) for every single step. Error messages are printed where applicable.
The user register counts with an increment of 0100 when the micro program is loaded. When the micro program is executed (4 times), it is a little more complicated. It is started in location n+4, if the first control store address is n, and the sequence of addresses will be

n, n+2, last-2, last, last-1, n+3, n+1, n+5
n, n+2, last-2, last, last-1, n+3, n+1, n+6
n, n+2, last-2, last, last-1, n+3, n+1, n+7
and so on.

7.4.4. TST05 Verify sequencing (single step a lot of ALU,ADIR and JMPCAR and W,XD).

This routine verifies the computed address register. It fills the control store with the three micro instructions ALU,ADIR; JMPCAR; W,XD; over and over again, from address n+1 (if the first control store address is n). The ALU,ADIR instruction loads the computed address register with an address (different for each ALU,ADIR: n+4, n+7, n+012, n+015, and so on). The program is started in location n+1 (if the first control store address is n), and the CSA bus should go like this: n+1 n+2 n+4 n+5 n+7 n+010 ... ND-100 reads the CSA bus (control store address bus) and checks it. Error messages are printed where applicable.
The user register counts with an increment of 077 when the micro program is loaded. When the micro program is executed, the user register displays the CSA bus (n+1 n+2 n+4 n+5 n+7 n+010 n+012 etc.).

7.4.5. TST06 Verify sequencing (single step a lot of ALU,ADIR and JMPREL and NEXT).

This routine verifies jumps relative to the micro address. The displacement is in the computed address register. The control store is loaded over and over again with the micro instructions ALU,ADIR; JMPREL; NEXT; The ALU,ADIR instruction loads the computed address register with a displacement of -5. The program is started in one of the last ALU,ADIR instructions. SLICE will step through the micro program, read the CSA bus, and check it. Error messages are printed where applicable.
The user register will count with an increment of 077 when the micro program is loaded. When it is executed, it will go like this: 011 012 013 6 7 010 3 4 5 ...

7.4.6. TST07 Verify bit mask bits as A-opr and B-opr.

This routine verifies the bit mask by letting a micro program load the X#0-X#3 registers with 4 different bit masks, and then report back to ND-100 (by doing DMA transfers). ND-100 checks that the bits are correct, loads another piece of micro program with other bit masks, and starts ND-500 once more. Error messages are printed where applicable. The bit mask is generated both as A-operand and B operand.
The user register will count to 0100, because the routine is run 64 times (each run testing 32 A-operands and 32 B-operands).

7.4.7. TST08 Verify A,BMR and B,BMR (bit mask register).

The 5-bit Bit Mask Register is verified by letting a micro program load it with a bit number, and then load two of the X#0-X#3 register with numbers containing only one 1-bit, decoded from the BMR. This is done with the BMR both as A-operand and B-operand. ND-500 then reports back to ND-100 (by DMA transfers), and ND-100 checks the results. Error messages are printed where applicable. ND-100 then modifies the micro program with other bit numbers, and starts ND-500 once more.
The user register displays the number of runs by counting to 040, since the BMR register can hold 32 different values (it has 5 bits).

7.4.8. TST09 Verify all logical ALU functions.

All the 16 logical ALU functions are verified. Those with only one operand are checked first. All four data types, word, float, halfword, and byte, are used. There are 66 operands: 0, -1, single 1-bit (32 different operands), single 0-bit (32 different operands). A micro program is loaded that executes the logical function and reports back to ND-100 by doing DMA transfers. ND-100 checks the results and prints error messages where applicable. Then another micro program is loaded, and the process is repeated. If the program is run in one-by-one mode, all functions, or only one, can be tested. When only one function is selected, the routine will run this function until N or P is typed, thus making it possible to extensively check a specific function.
The user register displays the octal value of the logical function in bits 2-6, with bit 6 always equal to 1, and the data type in bits 0-1.

7.4.9. TST10 Verify all arithmetical ALU functions.

The 16 arithmetical ALU functions are verified in numerical order. Those with no or only one operand will execute faster (0, 3, 014, 017). The four data types word, float, halfword, and byte, are all used. The four carry types forced zero, forced one, carry from status, and carry from micro status, are all used. There are 66 operands: 0, -1, single 1-bit (32 different operands), single 0-bit (32 different operands). ND-100 loads a micro program and starts it. ND-500 then executes an arithmetical ALU function, with a given data type, carry type, and zero, one, or two operands. The result is sent back to ND-100 by DMA transfers, and ND-500 unlocks. ND-100 then checks the results and prints error messages where applicable. The micro program is modified, loaded and started, and the whole process is repeated. If the program is run in one-by-one mode, all functions, or only one, can be tested. When only one function is selected, the routine will run this function until N or P is typed, thus making it possible to do an extensive check of a specific function.
The user register displays octal arithmetical ALU function code in bits 4-8 (bit 8 is zero), data type in bits 2-3, and carry type in bits 0-1.

7.4.10. TST11 Verify ND-500 registers (X#0-X#3, AM#0-AL#0-AM#1-AL#1, etc.).

The ND-500 registers are tested, 4 32-bits registers at a time, by loading patterns into them. A micro program loads the registers, sends their contents to ND-100 by DMA transfers, and unlocks. ND-100 checks the results and prints error messages where applicable. Then the micro program is modified by inserting the next shifted version of the pattern, and the program is reloaded and restarted. When the pattern has been completely used, new register codes are inserted into the micro program, and the process is repeated. Four 128-bit patterns are used, and each of them is rotate shifted left for each run. If the program is run in one-by-one mode, all registers, or only one register block, can be tested. When only one register block is selected, the routine will check this specific register block over and over again, until N or P is typed.
The user register displays the number of the register block under test, in bits 9-13, pattern number in bits 7-8, and shifted version number in bits 0-6. There are 024 register blocks.

7.4.11. TST12 Verify scratch registers as A-block and B-block.

This routine is run 4 times, by a micro program loaded and started by ND-100. All the scratch registers are loaded with patterns, the first two times by ADIR instructions, the next two by ADIRC instructions. In the first and third run, the scratch registers are copied to X#0-X#3 by A-operands, and in the second and fourth run, by B-operands. The X#0-X#3 registers are sent to ND-100 by DMA transfers, and ND-100 checks the results. Error messages are printed where applicable.
The user register displays the run number (0 to 3).

7.4.12. TST13 Verify sequencing (conditional jumps).

This routine checks conditional jumps. It loads a micro program and single steps through it. For every single step, the CSA bus, containing the control store address, is read and compared to an expected value. In case that the address is not at all like the expected value, an error message containing the two numbers will be printed. Otherwise, the error messages will contain information about status or micro status, the condition that failed, etcetera. The user register will contain the control store address, and bits 10-15 will be continually incremented.

7.4.13. TST14 Verify loop counter decrement (LCDECR).

The control store is filled with the micro instruction LCDECR; IODIN is loaded with a start value. The micro program is started. The start value is copied to the loop counter, and all the micro instructions are executed, thereby decrementing the loop counter. The result is copied to IODOUT, and ND-500 unlocks. ND-100 reads and checks this value, and prints error messages where applicable. Then another start value is loaded into IODIN, and the process is repeated.
The user register will first display control store load address, and then the different start values that are loaded into IODIN.

8. MEMIC - the ND-500 cache and memory test program.

8.1. General information.

MEMIC tests the ND-500 memory and cache. There are two channels, one for data, and one for instructions. The same memory may be used both as instruction and data memory. The first two routines are not verification routines. All the routines are micro programmed. They are loaded and started by ND-100. When the program has been started, its execution may be modified and altered by typing break characters (one-letter commands).

8.2. How to load and start the program.

Stand-alone: MASTER CLEAR. 1560&. *LOAD MEMIC.

SINTRAN: @RT MEMIC and possibly @LOG, or @STMIC.

MEMIC first asks whether the user wants to run on another terminal. If the answer is Y, MEMIC will request a decimal logical device number (SINTRAN), or an octal physical device number (stand-alone).

If the program runs under SINTRAN, it will reserve the ND-500 and possibly some memory, if the user wants it. MEMIC will ask all the necessary questions.

Which memory addresses to use when MEMIC is run, depends on the circumstances. MEMIC may be run under SINTRAN, or stand-alone. It may check only memory, only cache, or both cache and memory. It may run with memory management on, or off, or without memory management at all. It can run in a system where the ND-100 and the ND-500 both have their own memory, or where the ND-100 at least has memory which the ND-500 cannot reach. It can also run in a system where the ND-100 and the ND-500 share all the memory (user, beware!). The memory configuration of a system may be checked by the stand-alone test program MPEST. It will print out two memory maps, one for the ND-100 and one for the ND-500. If there is shared memory, the ND-100 value of address 0 of the ND-500 address space will also be printed out. Below follow some advice on how to use MEMIC (memory management is shortened to MM):

Stand-alone and no MM.
If the ND-100 has local memory: specify any addresses.
If the ND-100 and the ND-500 share the memory: specify addresses from 0400000 and up. This is the start address of (ND-100) bank 1, and the micro programs started by ND-500 will not overwrite MEMIC itself.

Stand-alone and MM off.
The same as for no MM above.

Stand-alone and MM on.
If the ND-100 has local memory: specify any addresses. Be aware that an initiation micro program fills the TSB (translation speed-up buffer). If too much memory is specified, the memory addresses specified by the user will be changed by the program, because the TSB is filled up. For instance, if memory addresses from 0 to 06000000 (6 million) are specified, the program will change the last address to 05773774.
If the ND-100 and the ND-500 share the memory: specify addresses from 0400000 and up. Remember that the TSB has room for not more than 01377 pages!

SINTRAN and no MM.
Specify addresses from 0 and up. The size of the specified memory area should correspond to the number of pages reserved (each page holds 04000 bytes). If this area is too big, MEMIC will change the last address. The IZPA and DZPA registers will be set pointing to the first reserved memory page, and IUPL and DUPL will point to the last.

SINTRAN and MM off.
The addresses specified should correspond to the pages reserved, with the first address pointing to the address of the first page, and the last address pointing to the last page (multiply the page numbers by 04000). If any of the specified addresses point outside the memory area reserved, MEMIC will change them.

SINTRAN and MM on.
Specify addresses from 0 and up. The size of the specified memory area should correspond to the number of pages reserved. If this area is too big, MEMIC will change the last address. Address 0 will point to the first reserved memory page, and the TSB will be initiated for the reserved pages. If too much memory is specified (the TSB is filled up), the last address will be changed by MEMIC.

MEMIC may also run without memory. If the maximum memory addresses are specified as 0, only the cache will be used. In that case, memory need not be reserved when MEMIC runs under SINTRAN. When the machine has memory management, this should be off when running without memory. If not, a lot of misleading error messages will appear.

MEMIC asks for maximum control store address. The answer must be different from zero, i.e., 02000, 010000, or anything else suitable.

MEMIC then asks for the first control store address. The answer must be less than the maximum address. If it is nonzero, the micro programs will be relocated when they are loaded. The micro programs are always loaded from the first control store address.

MEMIC asks for stop on full page. If you want the program to pause when 22 lines have been written on the terminal, you should type Y, otherwise N. Stop on full page is useful when you are using a screen terminal.

MEMIC asks for the last and first instruction memory addresses. Memory addresses occupy 25 bits. The first address must be less than the last. MEMIC will set the two last bits in both addresses to zero, to get addresses at word boundary. The two addresses might be given as for instance 0 and 0100000. If the last address is given as 0, the instruction memory will not be tested (The cache memory will be tested. The HIC bit will be set to 1, meaning Hit In Cache. All memory references will be forced to go to the cache, and therefore not use the memory at all.)

MEMIC then asks for the last and first data memory address. The answers follow the rules for the instruction memory questions.

MEMIC then asks for the number of instruction and data cache modules. The answer to both these questions is a single digit: 0, 1, 2, or 4. If the answer is 1 or 2 or 4, MEMIC will print the corresponding last cache address and ask if the user wants to change this last address. If the answer is Y, MEMIC asks for the new last address. This can be useful if only part of the cache memory is wanted (for instance if the user wants to avoid using a specific address bit). If the answer is 0, only the memory (and not the cache memory) will be tested. The program will set the don't-use-cache bit in the memory modus register.

MEMIC then asks whether the program is to be run with memory management or not. If memory management exists, MEMIC asks if it is to be on or off. After this question, an initiation micro program is run. It will clear the cache, set some registers, and, if the MM is on, fill the TSB (translation speed-up buffer). If necessary, MEMIC will change the memory addresses specified by the user.

MEMIC asks: TEST, VERIFY, OR BOTH? If the answer is T, only the first two routines will be run. If the answer is V, all routines except the first two will be run. If the answer is B, all routines are run.

MEMIC then asks whether information is wanted. If the answer is Y, a list, describing the two modes and the break characters, is printed.

MEMIC then asks whether all routines are to run in succession, or one by one. If the answer is the letter O, MEMIC will ask if error messages are wanted, and for the number of the test. The test routine with the specified number will then be run repeatedly.
If the answer was A, all routines will be run in succession, one after the other.
After the answer to this question, the memory initiation micro program is always run.

At last, MEMIC asks whether the tests 4 to 7 should check memory and cache parity or not. If the answer is Y, the memory status registers will be read and checked for every memory read. A side effect of this is that the IR register will not be used when parity checking, because ICON1/DCON1 is loaded for every read.

The execution of the program may be interrupted and changed at any time by the typing of break characters.

8.3. The test routines.

8.3.1. TST01 Test EA (OPR(32-bits)-to-X#0-to-EA).

This is not a verification routine. It will ask for data or instruction memory. The answer is D or I. Then MEMIC will load a micro program and start it. ND-500 will then copy simulated OPR to the Effective Address register. OPR may be changed by the break character O, or by F (flip (switch) a bit in OPR).
The user register is not used by this test.

8.3.2. TST02 Test data/instr. memory (OPR(32-bits)-to-memory).

This is not a verification routine .It will ask for data or instruction memory. The answer is D or I. Then it will request a memory address, load a micro program and start it. ND-500 will then write simulated OPR into the memory, in the specified address. The contents of the memory may be changed by the break characters O or F.
The user register is not used by this test.

8.4. The verification routines.

8.4.1. TST03 Verify address arithmetic.

A micro program with ten different address arithmetic tests is loaded. Then the first test is started. It uses 200 different patterns, and also the complement of the patterns. Errors are reported back to ND-100 by DMA transfers. When one test is completed, the next is started, until all the ten tests are done. The following address arithmetic modes are tested:

PASSAA AA,DP1
PASSAA AA,DP2
PASSAA AA,EA1
PASSAA AA,EA2
PASSAB AB,IX
PASSAB AB,2IX
PASSAB AB,4IX
PASSAB AB,8IX
PASSAB AB,1/8IX
PASSAB AB,B
PASSAB AB,R
PASSAB AB,PC
PASSAB AB,DPARG
AA+AB AA,EA1 AB,B
AA+AB AA,EA2 AB,R
A,SP
A,P
The user register displays test number. It counts from 0 to 011.

8.4.2. TST04 Verify data memory (address in address).

This routine verifies the data memory, if any. ND-500 will store word address in word address in the specified memory area, and then read it back. If errors are detected, or the reading is complete, ND-500 unlocks. ND-100 checks the results and prints error messages where applicable. Memory reading and writing are done in two ways, sequentially and randomly. Sequentially means that the memory write and read starts at a minimum address, is incremented by 4, and ends at a maximum address. Randomly means that the addresses occur in this sequence:
minimum, maximum, minimum+4, maximum-4, minimum+010, ...
There are four runs:

0: Write sequentially, read sequentially
1: Write sequentially, read randomly
2: Write randomly, read sequentially
3: Write randomly, read randomly

In addition, when writing randomly, the micro program does 4 writes by consecutive micro instructions in order to stress the memory.
When reading from the memory (and/or the cache), the registers DSTS0, DSTS1, and DSTS2, are read and checked, if parity check is requested.
The user register displays run number.

8.4.3. TST05 Verify data memory (compl. of address in address).

This routine is similar to the previous one. The difference is that the complement of word address is stored.

8.4.4. TST06 Verify instruction memory (address in address).

This routine is similar to TST04 above. The difference is that it verifies the instruction memory, and checks the registers ISTS0, ISTS1, ISTS2.

8.4.5. TST07 Verify instruction memory (compl. of address in address).

This routine is similar to TST05 above. The difference is that it verifies the instruction memory.

8.4.6. TST08 Verify memory (write once and read five times).

This routine verifies both data and instruction memory. ND-500 will write into the memory 17 32-bit words over and over again. When the specified memory area is filled, ND-500 will read it 5 times. Errors will be reported to ND-100 by DMA transfers. When 5 reads are completed, the pattern will be rotate shifted one left, and the process is repeated. Altogether, the writing takes place 01040 times.
The user register displays the number of reads, 05240 per channel (if both channels are verified, the count will go to 012500).

8.4.7. TST09 Verify data read and write with 1, 2, 3, and 4 bytes.

This routine verifies a memory area of the same size as the cache (never more than 0200000 bytes). The contents of this area will always be halfword address in halfword during this test. This means that the first halfword will contain 0, the next will contain 2, the next 4, and so on. ND-500 first writes this area full, 4 bytes at a time, and reads it back, 4 bytes at a time. Next, ND-500 reads back 1 byte at a time, 2 bytes at a time, and 3 bytes at a time. Errors are reported to ND-100. After this, ND-500 writes halfword address in halfword address three times more, but now it writes 1 byte, 2 bytes, and 3 bytes at a time. Each time the whole area is written, it is read back 4 bytes at a time and checked, and errors are reported to ND-100. ND-100 waits for unlock, and prints error messages where applicable.
The user register is not used in this test.

8.4.8. TST10 Verify data cache directory.

The specified memory area is first written with word address in word. After this, the memory contents is read back in such a way that the cache directory bits are checked. An example will clarify this: If the first memory address is specified as 0, and the number of bytes in the cache is 0100000 (two cache modules), then the memory read addresses will be in this sequence:

0, 0100000, 0, 0200000, 0, 0300000, 0, ... ,
4, 0100004, 4, 0200004, 4, 0300004, 4, ... ,

and so on.
The user register is not used in this test.

8.4.9. TST11 Verify instruction cache directory.

This test is similar to the one above.

9. PREEF - the ND-500 prefetch processor test program.

9.1. General information.

PREEF tests the prefetch processor. It loads instructions into the memory, and loads and starts small micro programs. These micro programs simulate macro program execution. They read and check the prefetch status register and the P register.

9.2. How to load and start the program.

Stand-alone: MASTER CLEAR. 1560&. *LOAD PREEF.

SINTRAN: @RT PREEF and possibly @LOG, or @STMIC.

PREEF asks for some parameters. These questions are similar to the questions asked by MEMIC. See the chapter on MEMIC.

9.3. The test routines.

9.3.1. TST01 Test prefetch clear and prefetch start.

The P register is loaded with zero. Then the micro program does prefetch clear and prefetch start. The micro program is restarted continually by master clear from ND-100.

9.3.2. TST02 Test prefetch clear, start, and continue.

The start address will be asked for. Then the micro program is loaded and started. It will load the P register with the specified start address, and then do prefetch clear, prefetch start, and prefetch continue. The micro program will be continually restarted by master clear from ND-100.
The user micro program (break character U) with the command L may be used to insert different macro instructions into the instruction memory prior to running this routine.

9.4. The verification routines.

9.4.1. TST03 Verify execution of instructions with no operands.

The memory is filled with instructions without operands, for instance NOOP. Then a micro program is loaded and started. It loads the P register, and does PRF,CLEAR and PRF,START and one PRF,PCONT for each macro instruction. It will read the prefetch status register and the P register after each PRF,PCONT. Errors are reported to the ND-100. Only one error is allowed for each macro instruction.
The user register will display the instruction under test.

9.4.2. TST04 Verify execution of instructions without operands, with register number.

The memory is filled with instructions, for instance W1 CLR. Then a micro program is loaded and started. It will load the P register, do PRF,CLEAR and PRF,START and one PRF,PCONT for each macro instruction loaded into the memory. It will read and check the prefetch status register and the P register after each PRF,PCONT. Errors are reported to the ND-100. Only one error is allowed for each macro instruction.
The user register will display the instruction under test.

9.4.3. TST05 Verify execution of instructions with one operand.

The memory is filled with instructions, for instance 'H3=:B.377:B. Then a small micro program is loaded and run. It will load the P register and do PRF,CLEAR and PRF,START and one PRF,PCONT for each macro instruction loaded into the memory. It will read and check the prefetch status register and the P register after each PRF,PCONT. Also, for some macro instructions, it will read and check a constant in A,DATA and an address (B+displacement) in B,EA1. Errors are reported to the ND-100. Only one error is allowed for each macro instruction.
The user register will display the instruction under test.

9.4.4. TST06 Verify execution of GO:B.

The memory is filled with GO *+6; GO *-2 over and over again. Then a small micro program is loaded. It will start the macro program in address 2, check the results, and report errors where applicable. The most important check is that the P register is correct (2, 010, 6, 014, 012, 020, ...). Errors are reported to the ND-100. Only one error is allowed.
The user register is not used during this test.

PREEF - the ND-500 prefetch processor test program.

9.4.5. TST07 Not yet implemented.

9.4.6. TST08 Verify execution of JMPMAP.

The control store is filled with
ALU,ADIR A,XD,SARG SLOW2 D,X#0 <addr> JMP LABEL,
where <addr> is the micro instruction's own address, and LABEL is the entry point of the verifying part of the micro program. A ND-500 macro instruction is fetched from a table, stored in the instruction memory and a small micro program starts it. The JMPMAP instruction in the micro program will jump to an entry point (a control store address), and this address will be loaded into X#0. X#0 is then verified against a table. This table contains approx. 1850 macro instructions. After all the legal macro instructions have been checked, it is checked that all the illegal macro instructions jump to a common entry point.
If this test is run in one-by-one mode, with maximum control store address equal to 01000, and the break character I is typed, the user can prepare a loop to use with the oscilloscope. Use maximum control address 010000, start the user micro program, and assemble JMP 0762,0; into address 0 and 01000. Start the user micro program in 0762. OPR may then be set with different instruction codes, and followed on the oscilloscope.
The user register will first contain a micro program load address. It will be incremented by 0100 as long as the micro program is loaded. When JMPMAP is checked, the user register contains a macro instruction table index. Afterwards, the user register contains an illegal instruction table index.

PREEF - the ND-500 prefetch processor test program.

10. ARITH - The ND-500 external arithmetic test program.

10.1. General information.

ARITH tests the external arithmetic. It does this by loading small micro programs that shift logical, arithmetical, and rotational. They also do floating add, subtract, multiply, and divide. The results are checked, and errors reported.

10.2. How to load and start the program.

Stand-alone: MASTER CLEAR. 1560&. *LOAD ARITH.

SINTRAN: @RT ARITH and possibly @LOG, or @STMIC.

ARITH asks for some parameters. These questions are similar to the questions asked by SLICE. See the chapter on SLICE.

10.3. The test routines.

10.3.1. TST01 Test logical shift.

The routine will ask for two 32-bit numbers and a shift count. Then the first number is shifted logically. The number of positions to shift is specified by the shift count. Negative shift count is shift right, positive is shift left. Then the second number is shifted in the same way, and the whole process is repeated until a break character is typed.
If the jump instructions in the micro program at addresses 5 and 012 are changed to NEXT; by the user micro program, the results will be output to the terminal.

10.3.2. TST02 Test arithmetical shift.

This routine is similar to the previous one, but the shift is arithmetical (the sign bit is copied in right shifts).

10.3.3. TST03 Test rotational shift.

This routine is similar to the previous two, but the shift is rotational, and the shift count should only be positive.

10.3.4. TST04 Test single floating sum (A+B).

The program will ask for two numbers. Each is given either as a floating number or as two 16-bit octal numbers. Then a small micro program will add the two numbers repeatedly, first A+B, then B+A, and so on.
If the jump instructions at addresses 5 and 012 are replaced with NEXT; by the user micro program, the results will be output to the terminal.

10.3.5. TST05 Test single floating diff (A-B).

This routine is similar to the previous one, except that it computes the difference, and not the sum.

10.3.6. TST06 Test single floating mult (A*B).

This routine is similar to the previous two, except that it computes the product of the two numbers.

10.3.7. TST07 Test single floating div (A/B).

This routine will compute A/B, but not B/A. Jump instructions at addresses 4, 010, 015, 021, and 026 may be replaced with NEXT; by the user micro program. The intermediate and final results will then be output to the terminal.

10.3.8. TST08 Test convert to floating.

This routine will convert two 32-bit integers to floating. If the micro instructions at addresses 4, 010, 015, and 021 are changed to NEXT; by the user micro program, the results will be output to the terminal.

10.3.9. TST09 Test convert to integer.

This routine will convert two floating numbers to integers. If the micro instructions at the addresses 5 and 012 are changed to NEXT; by the user micro program, the results will be output to the terminal.

10.3.10. TST10 Test single integer div (A/B).

This routine will convert two integers to floating, divide them, and convert the result back to integer. Jump instructions at addresses 4, 010, 015, 021, 026, 032, 037, 043, 050, 054, 061, 065, 072, 076 and 0103 may be changed to NEXT; by the user micro program in order to produce results on the terminal.

10.4. The verification routines.

10.4.1. TST11 Verify shift logical with shift count as argument.

Shift logical is tested as word shift, halfword shift, and byte shift. Shift count is in the range 037 to -037, 017 to -017, or 7 to -7. For each type and shift count, a set of patterns is shifted, and the results are checked.
The user register displays shift count in bits 11-7, and a pattern number in bits 6-0.

10.4.2. TST12 Verify shift logical with shift count from shift count register.

This routine is similar to the previous one, except that it takes the shift count from the shift count register.

10.4.3. TST13 Verify shift arithmetical with shift count as argument.

This routine is similar to TST11 above. The only difference is that it checks arithmetical shift.

10.4.4. TST14 Verify shift arithmetical with shift count from shift count register.

This routine is similar to TST12 above. The only difference is that it checks arithmetical shift.

10.4.5. TST15 Verify shift rotational with shift count as argument.

This routine is similar to TST11 above, except that it checks rotational shift, and the shift count is in the range 037 to 0, 017 to 0, or 7 to 0.

10.4.6. TST16 Verify shift rotational with shift count from shift count register.

This routine is similar to TST12 above, except that it checks rotational shift. The shift count is only positive.

10.4.7. TST17 Verify double floating sum.

Pairs of double floating numbers are added together, and the resulting sums are checked. The first part of this test consists of 63 numbers with only one bit in the mantissa (apart from the hidden bit). This number is added to itself, and the result should have the same mantissa, and an exponent incremented by one. The second part consists of a set of different patterns with given results to check against.
The user register will first display a bit number in the range 0 to 077, and then a pattern number.

10.4.8. TST18 Verify double floating diff.

This routine is similar to the previous one, except that it checks double floating difference.

10.4.9. TST19 Verify double floating mult.

Pairs of double floating numbers are multiplied together, and the results are checked.
The user register displays a pattern number.

10.4.10. TST20 Verify double floating div.

This routine is similar to the previous one, except that it checks double floating division. The divisions will be 1/(2**n-1), where n is in the range 1 to 55 (** means exponentiation).

11. NOMAN - the ND-500 no-memory-management test program.

11.1. General information.

NOMAN tests the no-memory-management. Some machines are not equipped with memory management. NOMAN is intended for use on these machines.

A word of warning should be given here. NOMAN must not be run under SINTRAN on systems where the memory is shared and the ND-100 has no local memory (that is, address 0 for both the ND-100 and the ND-500 is the same memory word). If this were attempted, SINTRAN will be destroyed immediately.

11.2. How to load and start the program.

Stand-alone: MASTER CLEAR. 1560&. *LOAD NOMAN.

SINTRAN: @RT NOMAN and possibly @LOG, or @SIMIC.

NOMAN asks for some parameters. These questions are similar to the questions asked by MEMIC. See the chapter on MEMIC.

11.3. The test routines.

11.3.1. TST01 Test no-memory-management registers

This routine loads and reads the registers DCINHLL, DCINHLU, DZPA, DUPL, or ICINHLL, ICINHLU,IZPA, IUPL. They are loaded with the value of OPR. This value may be changed at any time by using the break character O.

11.4. The verification routines.

11.4.1. TST02 Verify DCINHLL-DCINHLU-DZPA-DUPL.

These four 14-bit registers are loaded with all possible values. After each load, the values are read back and checked.
The user register displays the value loaded.

11.4.2. TST03 Verify ICINHLL-ICINHLU-IZPA-IUPL.

This routine is similar to the previous one, except that it checks the instruction registers.

11.4.3. TST04 Verify DRADDRM and DRADDRL.

This routine checks the data real address registers. DRADDRM is an 8-bit register, and DRADDRL is a 16-bit register. DRADDRM is the most significant part of the real address, and DRADDRL is the least significant part. The real address is a 24-bit physical (not logical) memory byte address. The most significant bit is missing. Real address is the sum of the logical address and 04000*DZPA. The routine writes in the memory with different logical addresses and different contents of DZPA. Then it reads the real address and checks it.
The user register displays the value of DZPA.

11.4.4. TST05 Verify IRADDRM and IRADDRL.

This routine is similar to the previous one, except that it tests the corresponding instruction registers.

11.4.5. TST06 Verify cache inhibit for data memory.

This routine checks the data cache inhibit limit registers DCINHLL and DCINHLU. First, both registers are set to zero, meaning no inhibit. Then 4096 words are written into the available cache in such a way that not two words belong to the same page (the first address is 0, and the address is incremented by 040004, 0100010, or 0200020, depending on the number of cache modules installed). The contents of the written words are address in address. After this, DCINHLL and DCINHLU are loaded with test values, and 4096 words are written into the cache, containing complement of address in address. After this, all the 4096 words are read back and checked. The area with page numbers greater than DCINHLL and smaller than DCINHLU should contain address in address. If not, cache inhibit does not work. The rest of the area should contain complement of address in address. If not, cache inhibit is active outside the range specified.
All this is repeated 4 times, in order to check all 14 bits of DCINHLL/DCINHLU (the 4 initial addresses are 0, 040000000, 0100000000, 0140000000).
The user register displays the value 0 to 3 in bits 7-6, and a table index for test values ranging from 0 to 074 in bits 5-0.

11.4.6. TST07 Verify cache inhibit for instruction memory.

This routine is similar to the previous one. The only difference is that it checks the corresponding functions in the instruction channel.

11.4.7. TST08 Verify that read and write only affects the DZPA-DUPL area.

This routine tests the DZPA and DUPL registers. First, DZPA is loaded with 0, and DUPL with 037777. Then address in address is written into the available memory area. After this, DZPA and DUPL are loaded with test values, and complement of real address is written into the memory. Then DZPA and DUPL are reloaded with 0 and 037777, cache is cleared, and the contents of the memory is read and checked. The memory area with page numbers lower than DZPA , or with page numbers greater than DUPL, should contain address in address. If not, DZPA and/or DUPL did not work properly. The rest of the memory should contain complement of address in address. If not, DZPA and/or DUPL failed.
Please observe that this test uses cache clear. If cache clear does not work, this test will fail. TST10 will check cache clear.
If run with a memory size of 010000000 (ten million octal) bytes, this test will take approx. 2 hours 10 minutes.
The user register is not used by this test.

11.4.8. TST09 Verify that read and write only affects the IZPA-IUPL area.

This routine is similar to the previous one, except that it checks the corresponding functions in the instruction channel.

11.4.9. TST10 Verify data cache clear.

This routine checks that data cache clear works. The cache has 4 partitions, and any combination of these 4 may be cleared. First, the program selects an area of available data memory twice the size of the available cache. This memory area is thought to be divided in 8 parts, parts 0-7. DZPA is then made to point to the first page in part 0, and DUPL is made to point to the last page in part 7. Then the memory area is filled with address in address, written from the highest address to the lowest. After this, DZPA is made to point to the first page in part 4. Cache is not cleared. Part 0 to 3 of the memory is read, and it is checked that it really contains addresses for part 0 to 3. Then cache is cleared, with an argument in the range 0 to 017. This means that any combination of cache partitions may be cleared. Then parts 0 to 3 are read once more. The contents should be addresses from part 4 or 0, 5 or 1, 6 or 2, 7 or 3, depending on if the cache is cleared or not. If cache is cleared, the contents should come from part 4 to 7.
The user register displays the cache clear argument (0-017).

11.4.10. TST11 Verify instruction cache clear.

This routine is similar to the previous one, except that it checks instruction cache clear.

NOMAN - the ND-500 no-memory-management test program.

12. GMOFF - the ND-500 memory-management-off test program.

12.1. General information.

GMOFF tests the memory management, with memory management turned off. Some machines are not equipped with memory management, but the majority are. GMOFF is intended for use on those that are so equipped.

12.2. How to load and start the program.

Stand-alone: MASTER CLEAR. 1560&. *LOAD GMOFF.

SINTRAN: @RT GMOFF and possibly @LOG, or @STMIC.

GMOFF asks for some parameters. These questions are similar to the questions asked by MEMIC. See the chapter on MEMIC.

12.3. The test routines.

12.3.1. TST01 Not yet implemented.

12.3.2. TST02 Dump routine for IWIPGU/DWIPGU.

Both for instructions and data, there are two buffers, written-in-page (WIP) and page-used (PGU). Each of these buffers (4 altogether) contains 16384 bits. Each bit represents a page in the memory. These bits may be dumped on the terminal by this routine. 8 16-bit words (broadsides) are printed on each line. The bit for the smallest page number is in bit 0 of the leftmost word, and the bit for the greatest page number is in bit 15 of the rightmost word. The broadside number of the first 16 bits on a line is printed first on the line.

12.4. The verification routines.

12.4.1. TST03 Verify data scratch file (DSCFA and DSCRF).

The 16 16-bit words in the scratch file are written, each separately, and then read back and checked. After this, all 16 words are written using the automatic incrementation of DSCFA, and then read back and checked.
The user register will display a scratch file word number in the range 0-017.

12.4.2. TST04 Verify instr. scratch file (ISCFA and ISCRF).

This routine is similar to the previous one. It checks the instruction scratch file.

12.4.3. TST05 Verify DLADDR and DRADDR.

The data memory is referenced (read 4 bytes). After this, the logical and real address registers are read and checked (they should be equal, except that the real address is shifted 1 to the right).
The user register displays the logical address.

12.4.4. TST06 Verify ILADDR and IRADDR.

This routine is similar to the previous one. It checks the instruction logical and real address registers.

12.4.5. TST07 Verify WIP-buffer for data memory (DWIPGU).

The WIP buffer for data memory is filled with a data pattern. After this, the pattern is read back and checked. Several different patterns are used.
The user register will contain 0 during this test.

12.4.6. TST08 Verify PGU-buffer for data memory (DWIPGU).

This routine is similar to the previous one. It checks the PGU buffer for data memory.

12.4.7. TST09 Verify WIP-buffer for instr. memory (IWIPGU).

This routine is similar to the previous one. It checks the WIP buffer for instruction memory.

12.4.8. TST10 Verify PGU-buffer for instr. memory (IWIPGU).

This routine is similar to the previous one. It checks the PGU buffer for instruction memory.

12.4.9. TST11 Verify WIP and PGU for data memory write and read (HIC=1).

The data memory is written into and read from, over the whole address range. The HIC bit is set to 1 in this test, to ensure that Sintran will not be destroyed (only the cache is used). As a result of this, patterns of bits should be 1 in WIP and PGU. These buffers are then read and checked.

The user register is not used in this test.

12.4.10. TST12 Verify WIP and PGU for instr. memory (HIC=1).

This routine is similar to the previous one. It checks the instruction memory.

GMOFF - the ND-500 memory-management-off test program.

13. GMENT - the ND-500 memory-management-on test program.

13.1. General information.

GMENT tests the memory management, with memory management turned on. Some machines are not equipped with memory management, but the majority are. GMENT is intended for use on those that are so equipped.

13.2. How to load and start the program.

Stand-alone: MASTER CLEAR. 1560&. *LOAD GMENT.

SINTRAN: @RT GMENT and possibly @LOG, or @STMIC.

GMENT asks for some parameters. These questions are similar to the questions asked by MEMIC. See the chapter on MEMIC.

13.3. The test routines.

13.3.1. TST01 Test hashed TSB.

This is a routine for accessing the memory with memory management on. It is both for instruction and data memory. TST01 asks for all relevant parameters. Based on these parameters, the hashed index is computed in the same way as the hardware will do, and printed on the terminal. If the user wants another index, the parameters will be asked for once more. After this, a micro program is loaded and started. It will run in a loop until stopped. It will do the following:

Enable the TSB for write.
Clear the TSB and wait until the clear is finished.
Enable the TSB for read.
Load the process, segment, and domain registers.
Load the memory modus register.
Do a clear-block (bit 3 in DCON1/ICON1).
Read from the memory (address in X#0).
Read the status and put it in X#1 (should give TSB fault).
Enable the TSB for write.
Insert a page number into the TSB.
Enable the TSB for read.
Do a clear block.
Read from the memory.
Read the status and put it in X#2 (should not give TSB fault).
Read the real address and put it in X#3.
Read the hardware computed hashed index and put it in AL#1.
Repeat the whole sequence.

13.3.2. TST02 Dump routine for IWIPGU/DWIPGU.

This is the same dump routine as TST02 under GMOFF.

13.3.3. TST03 Test sequential TSB (hash index = 0).

This routine is somewhat similar to TST01, but it will test the sequential part of the TSB, instead of only the hashed part. The routine inserts one entry in the hashed part of the TSB, with hash index equal to zero. After this, the memory is accessed three more times, with hash index equal to zero, but with other parameters. The next three entries, therefore, will go into the sequential part of the TSB. The routine will do the following:

Enable the TSB for write.
Clear the TSB and wait until the clear is finished.
Enable the TSB for read.
Load the process, segment, and domain registers, all with 0.
Load the memory modus register and do a clear block.
Load the X#3 register with 010,0.
Read from the memory, address in X#3.
Read the status to IODOUT (should give TSB fault).
Enable the TSB for write.
Insert a page number into the TSB.

After this, the first entry in the hashed part of the TSB holds a page number. Then the memory is accessed three times more, and the sequential part of the TSB will be used. Only the first of the three follows below:

Load ISTSB/DSTSB with 0. This means that the sequential part of the TSB is empty, as ISTSB/DSTSB always points to the first free entry. (The next two times, load with 1 and 2.)
Enable the TSB for read.
Activate FAS2. This makes the use of the sequential TSB possible.
Load the process register with 1, and the segment and domain registers with 0. (The next time, load the process register with 2, and the last time load with 4.)
Load the memory modus register and do a clear block.
Read from address 0.
Read the status to IODOUT (should give TSB fault).
Enable the TSB for write. Insert a page number (0100, 0101, 0102) in the (sequential) TSB.
Read the sequential TSB pointer (IHXA/DHXA) to AL#1 (or AL#2 or AL#3).
Repeat two more times, then do all of it again.

13.4. The verification routines.

13.4.1. TST04 Verify the hash addressed TSB for data memory (DTSB).

This routine will fill the 512 entries of the TSB, and check the memory management status and the real address. The values of the process, segment, and domain registers are taken from a table, together with the logical address. The routine executes the following steps:

1. Clear the TSB. Read from the memory with the values from the 512 entries of the table, and insert page numbers 0 to 0777. Check the memory management status (TSB-fault should occur).
2. Read from the memory once more with the values from the table. Check the memory management status (TSB-fault should not occur), the real address, and the hardware computed hashed TSB index (remember, the most significant of the 9 bits is not checked).
3. As step 1, but some of the values from the tables are ones-complemented, and the page numbers to be inserted go from 037777 to 037000. In this way, all the bits in the TSB will get both the values 0 and 1.
4. As step 2, with some values ones-complemented.
5. As step 1, but with some other values ones-complemented.
6. As step 2, but with some other values ones-complemented.

The user register will display the following values:
the value 0 to 5 in bits 14-12, and an index in bits 11-0.

13.4.2. TST05 Verify the hash addressed TSB for instr. memory (ITSB).

This routine is similar to the one above, except that it checks the instruction part.

13.4.3. TST06 Verify the sequential addressed TSB for data memory (DTSB).

This routine is somewhat similar to TST04, but it checks the sequential part of the TSB. It has the same 6 steps. It will read from the memory 257 times per step. All hashed indexes will be 0, because of the 257 different values taken from a table. This table is made in such a way that all the entries give the same hashed index, namely 0. The first access goes to the hashed part of the TSB, and the next 256 go to the sequential part of the TSB, since they have the same hashed index, and since FAS2 is enabled. The status, real address and sequential TSB pointer are checked.

13.4.4. TST07 Verify the sequential addressed TSB for instr. memory (ITSB).

This routine is similar to the one above, except that it checks the instruction part.

13.4.5. TST08 Verify data cache clear.

This routine checks that data cache clear works. The cache has 4 partitions, and any combination of these 4 may be cleared. First, the program selects an area of available data memory twice the size of the available cache. This memory area is thought to be divided in 8 parts, parts 0-7. Then the TSB is initiated for memory parts 0-7 with domain number 0, and for memory parts 4-7 with domain number 077 (63). In the first initiation, the first logical address is made to point to part 0. In the second initiation, the same first logical address is made to point to part 4. In this way, the same logical address will be translated to two different real addresses, dependent upon the value of the domain register. Now the memory area is filled with address in address, written from the highest address to the lowest, and domain number 0 is used. After this, the domain number is set to 077. Cache is not cleared. Part 0 to 3 of the memory is read, and it is checked that it really contains addresses for part 0 to 3. Then cache is cleared, with an argument in the range 0 to 017. This means that any combination of cache partitions may be cleared. Then parts 0 to 3 are read once more. The contents should be addresses from part 4 or 0, 5 or 1, 6 or 2, 7 or 3, depending on whether the cache is cleared or not. If cache is cleared, the contents should come from part 4 to 7.
The user register displays the cache clear argument (0-017).

13.4.6. TST09 Verify instr. cache clear.

This routine is similar to the previous one, except that it checks instruction cache clear.

14. TRAPT - the ND-500 trap system test program.

14.1. General information.

TRAPT tests the trap system, using the registers S1, S2, and TE.

14.2. How to load and start the program.

Stand-alone: MASTER CLEAR. 1560&. *LOAD TRAPT.

SINTRAN: @RT TRAPT and possibly @LOG, or @STMIC.

TRAPT asks for some parameters. These questions are similar to the questions asked by MEMIC. See the chapter on MEMIC.

14.3. The test routines.

14.3.1. TST01 Not yet implemented.

14.3.2. TST02 Not yet implemented.

14.4. The verification routines.

14.4.1. TST03 Verify S1 traps enabled by the TE register.

This routine will check all S1 traps that may be enabled by the TE (trap enable) register. First, S1 and S2 are set to zero, and one of the bits 9 to 29 (011 to 035) is set to 1 in TE (the trap is enabled). The trap system is turned on. Then the routine checks that no traps occur. After that, the routine will set (by software) one of the bits 9 to 29 in the S1 register to 1, and check that the trap occurs (whenever there is a trap, the micro program continues in micro address 0627). When this routine is run in one-by-one mode, it will ask for the number of the bit to test. If the user answers with a negative number, a list of all the bits will be printed.
The user register displays the number of the bit under test.

14.4.2. TST04 Verify S1 traps not enabled by the TE register.

This test is similar to the previous one, except that TE register is set to zero. Only two bits are tested, bits 30 and 31 (036 and 037).

14.4.3. TST05 Verify S2 traps.

This routine is similar to the previous two. The difference is that it checks the S2 traps. The TE register is set to zero. Bits 0 to 11 (0 to 013) are checked.

14.4.4. TST06 Verify S1 bit 5 and 6 (zero and carry, integer arithmetic).

This routine checks that bits 5 and 6 in S1 are set properly by hardware. Two equal numbers are subtracted, and the micro program checks that only the two bits 5 and 6 become 1 in S1. This is repeated about 2 million times. Before the test starts, the registers TE, S1 and S2 are all set to zero.
The user register is not used in this test.

14.4.5. TST07 Verify S1 bit 5 (zero, floating arithmetic).

This routine is similar to the previous one. It subtracts two equal floating numbers from each other and checks that S1 contains 1 only in bit 5.

14.4.6. TST08 Verify S1 bit 6 and 011 (carry and overflow).

This routine is similar to the previous two. A number is subtracted from an integer that has only one bit, the sign bit. The micro program checks that only bits 6 and 9 in S1 are 1.

14.4.7. TST09 Verify S1 bit 7 (sign, integer arithmetic).

One number is subtracted from another, the first being 1 greater than the second. The result should be -1. It is checked that S1 contains 1 in bit 7 only.

14.4.8. TST10 Verify S1 bit 7 (sign, floating arithmetic).

Two floating numbers are subtracted from each other. The first is one less than the second, giving the result -1.0. The micro program checks that bit 7 in S1 becomes 1.

14.4.9. TST11 Verify S1 bit 010 and 6 and 5 (flag and carry and zero).

Two numbers are subtracted from each other, giving zero as a result. Then K,1IFZ is applied, setting the flag to 1. This is then checked.

14.4.10. TST12 Verify overflow trap (S1 bit 011).

S1 and S2 are set to zero, and bit 9 is set in TE. The trap system is turned on. Then the number which has a 1 next to the sign bit is added into itself. This should give overflow trap. The micro program checks that the trap has occurred. This is done about half a million times. When the micro program stops, it leaves the last S1 in the IODOUT register, where the ND-100 fetches it, and checks that bit 9 is 1. If there is no trap, S1 is also left in IODOUT, but bit 0 is set to one to indicate a missing trap.
The user register is not used by this test.

14.4.11. TST13 Verify floating underflow trap (S1 bit 015).

This routine is similar to the previous one. Bit 13 in TE is set to 1. A very small floating number (0100,1) is multiplied by itself. The micro program checks that the trap takes place. When the micro program stops, after about half a million trys, ND-100 reads IODOUT and examines bit 0. If 1, there was a trap missing. If 0, ND-100 checks that bit 13 is 1.

14.4.12. TST14 Verify floating overflow trap (S1 bit 016).

A great floating number (077777,177777) is multiplied by itself, after bit 14 in TE has been set to 1. IODOUT is checked by the ND-100 as in the previous routine.

14.4.13. TST15 Verify single instruction trap (S1 bit 021).

Four NOOP instructions are loaded into the memory, and bit 17 in TE is set to 1. Then the program in the memory (the 4 NOOP's) is started. When the first instruction is finished, the micro program checks that trap has taken place. Error checking is done by the micro program and by the ND-100 as in the previous routines.

14.4.14. TST16 Verify branch trap (S1 bit 022).

Two instructions (JMPG ABSOLUTE) are loaded into the memory. Bit 18 is set to 1 in TE. The program is started. When the first instruction is completed, the micro program checks that the trap has taken place. S1 is checked by the ND-100.

14.4.15. TST17 Verify address trap fetch (S1 bit 025).

Four NOOP's are loaded into the memory. TE bit 21 is set to 1. The lower limit register (LL) is loaded with the address of the first NOOP. The higher limit register (HL) is loaded with the address+4. D,SETLIM is set to 1 (In this case, it could as well have been set to 0. See HL and LL.), and the program is started. When the first instruction has been completed, the micro program checks that the trap has occurred.

14.4.16. TST18 Verify address trap read for data memory (S1 bit 026).

For no memory management:

Bit 22 in TE is set to 1. The LL register is loaded with the address+020. The HL register is loaded with the address. D,SETLIM is set to 1. Then data is read from the address+040, and the micro program checks that the trap has occurred.

For memory management:

The logical address ADDR is set to a start value of 020000000 (twenty million).
The segment register will take all values from 0 to 037.
D,SETLIM will be either 0 or 1.
LL and HL will take the values ADDR-4, ADDR, ADDR+4.
For each set of values, data is read from the memory, and the micro program checks whether trap occurs or not, according to the rules laid down for the LL and HL registers (the limit registers). When the uppermost 5 bits of ADDR are all zero, bits 4-0 of the segment register are used in their place in the comparison with LL and HL.
After each check, ADDR is increased by 020000000 (twenty million). When ADDR becomes zero, the test is finished.

14.4.17. TST19 Verify address trap write for data memory (S1 bit 027).

Bit 23 in TE is set to 1. The LL register is loaded with the address. The HL register is loaded with the address+040. D,SETLIM is set to 0. Then data is written to address+020, and the micro program checks that the trap has occurred.

14.4.18. TST20 Verify address zero access trap for instruction fetch (S1 bit 030).

Bit 24 in TE is set to 1. Four NOOP instructions are loaded into address zero in the memory and started. When the first has been finished, the micro program checks that the trap has occurred.

TRAPT - the ND-500 trap system test program.

14.4.19. TST21 Verify address zero access trap for data memory (S1 bit 030).

Bit 24 in TE is set to 1. Data is read from address zero. The micro program checks that the trap has occurred.

14.4.20. TST22 Verify disable process switch timeout (S1 bit 036).

The TE register is set to zero. The loop counter (LC) is set to 0400 (256). Then bit 4 in S1 (disable process switch) is set to 1. The timeout should take place after 256 micro cycles. The loop counter is now decremented every second micro cycle. If it becomes zero, no trap took place, and the error is reported to the ND-100. If a trap occurred, the micro program checks that the value of the loop counter is in the range 0174 to 0203. If not, the error is reported to the ND-100.

14.4.21. TST23 Verify disable process switch error (S1 bit 037).

The TE register is set to zero. Bit 4 in S1 (disable process switch) is set to 1, and then bit 9 in S2 is set to 1 (DFAIL). This should give a trap. ND-100 checks that S1 (read from IODOUT) has a 1 in bit 31 (037).

14.4.22. TST24 Verify index scaling error trap (S2 bit 0).

The instructions W1:=B.0(R3) and NOOP are loaded into the memory. X#2 is loaded with a number with only one bit equal to 1, namely bit 30 (036). Then the program is started. When the first instruction is finished, the micro program checks if the trap has occurred.
When the micro program stops, it puts S2 in IODOUT for ND-100 to read. S2 has only 12 bits. The rest are set to 1 if there has not been any trap, and to zero if trap took place. This is the same for all routines that checks S2 traps.

14.4.23. TST25 Verify illegal instruction code trap (S2 bit 1).

Illegal instructions are loaded into the memory (octal value 001). Then the program is started, and when the first illegal instruction has been executed, the micro program checks for a trap.

14.4.24. TST26 Verify illegal operand specifier trap (S2 bit 2).

The memory is loaded with two instructions (W1=:0). Then the program is started, and when the first instruction is finished, the micro program checks for a trap.

14.4.25. TST27 Verify activate-from-ND-100 trap (S2 bit 5).

The micro program does not reset activate (tag code 5) as all the other routines initially do. Trap should therefore occur. This is checked by the micro program, and by the ND-100, as usual, checking S2.

14.4.26. TST28 Verify terminate-from-ND-100 trap (S2 bit 6).

The micro program starts by setting the loop counter to 0400 and by counting it down to zero. This is done in order to let the ND-100 have time to execute IOX TERM. After the delay, the micro program checks for a trap.

14.4.27. TST29 Verify IFAIL trap (S2 bit 010).

The micro program starts by doing a cache disable (bit 4 in ICON0). Then it enables all kinds of error traps in ICON1. A program is started in the memory. This should set some error bits in ISTS1, and these bits should give a trap.

14.4.28. TST30 Verify DFAIL trap (S2 bit 011).

Bit 4 is set to 1 in DCON0 (cache disable). Error traps are enabled in DCON1. Then data is read from the memory. This should set error bits in DSTS1, and these bits should give a trap.

14.4.29. TST31 Verify processor fault trap (S2 bit 013).

In the micro program, address 014, a jump to address 020015 is made. The micro program will continue in address 015, but bit 13 in the address should give processor fault trap.

If ND-500 has 020000 control store locations, processor fault will not occur. Instead, the program checks that control store parity error stops the micro program.

15. EXTRA - the ND-500 extra test program.

15.1. General information.

EXTRA will do some additional tests that there were no room for in the other test programs. The tests in FXTRA are mainly prefetch processor tests.

15.2. How to load and start the program.

Stand-alone: MASTER CLEAR. 1560&. *LOAD EXTRA.

SINTRAN: @RT EXTRA and possibly @LOG, or @STMIC.

EXTRA asks for some parameters. These questions are similar to the questions asked by MEMIC. See the chapter on MEMIC.

15.3. The test routines.

15.3.1. TST01 Not yet implemented.

15.4. The verification routines.

15.4.1. TST02 Verify index counters.

There are four 8-bit index counters. First, they are cleared by D,ICCLR, and the micro program reads all four and checks that they really are zero. Then they are incremented, one by one, by IADJ. After each increment, the four counters are read and checked.
The user register is not used by this test.

15.4.2. TST03 Verify conversion (BYTH BYTW HWIW).

The number 04000000 (four million) is put in X#0. Then X#0 is converted in three different ways, and the results are put into X#1, X#2, and X#3. The results are then checked. After each check, X#0 is decremented by 1, and the test continues until X#0 is zero. The user register is not used by this test.

15.4.3. TST04 Verify conditional ALU.

Conditional ALU is checked by first setting COND,MZRO to true, and then using the ALU operations ALU,FZRO and F,FONE. The result is then checked. Next, the condition is set to false, and the same ALU operations are used and checked once more. The test is repeated 04000000 (four million) times.
The user register is not used in this test.

15.4.4. TST05 Verify prefetch addressing modes.

28 of the 30 addressing modes described in the ND-500 Reference Manual are tested (see Survey of addressing modes). The two exceptions are DESC and ALT. The test is done in the following way:
The micro program starts by loading 30 macro instructions into the instruction memory. The data memory is written in such a way that every word contains its own address+020. Then execution is started at the first instruction memory address. After the execution of every macro instruction, the micro program checks EA1 and EA2, the data register, P, the result in one of the index registers, and the prefetch status register. If necessary, the constant register is also checked. If an error message occurs, the test is stopped at once (if multiple errors, only the first is reported). When the last macro instruction has been executed, a GO instruction brings the macro program back to the beginning again. This loop is executed 040000 times.
The user register is not used by this test.

15.4.5. TST06 Verify W1:=DESC(B.0110:B)(R4).

The instruction memory is loaded with four macro instructions, and the data memory is loaded with the descriptor (array length and array address). Then execution is started. The micro program will check array length in the data register, logical index in ORB, descriptor address in EA1 and EA2, the P register, the prefetch status register, the address of the current array element (in EA1 and EA2), and the logical index after incrementation. If an error message occurs, the test is stopped at once. The array that the descriptor points to takes all of the specified data memory. Therefore, the bigger the data memory is, the longer the test will last.

EXTRA - the ND-500 extra test program.

15.4.6. TST07 Verify INIT, CALL, ENTS, RET.

The instruction memory is loaded with the four macro instructions INIT, CALL, ENTS and RET. Then execution is started at the first instruction memory address. After all four macro instructions have been executed, the micro program checks the results.

For INIT:

The bottom-of-stack address in the displacement register.
The main-stack-demand in ORB.
The total-stack-demand in ORB.
The P register.

For CALL:

The subroutine address in the displacement register.
The number of arguments in ORB.
The argument address in EA1.
The P register.
The conditions CONOP, DATOP and ENTER.

For ENTS:

The stack-demand in ORB.

For RET:

The P register.

The test is repeated 65535 times.
The user register is not used by this test.

EXTRA - the ND-500 extra test program.

Appendix A ND-500 micro mnemonics in alphabetical and numerical order

ND-500 micro mnemonics in alphabetical and numerical order

ND-500 micro mnemonics in alphabetical and numerical order

```
1  'A,-1       ;000000;100000;000000;000000;000000;000000; % FLOATING -1.0
2  'A,A#0      ;000000;000000;000000;000000;000000;000000; % DOUBLE FLOAT REG 0
3  'A,A#1      ;000000;020000;000000;000000;000000;000000; % DOUBLE FLOAT REG 1
4  'A,A#2      ;000000;040000;000000;000000;000000;000000; % DOUBLE FLOAT REG 2
5  'A,A#3      ;000000;060000;000000;000000;000000;000000; % DOUBLE FLOAT REG 3
6  'A,AL#0     ;000004;000000;000000;000000;000000;000000; % A-REG 0 LEAST SIGN.
7  'A,AL#1     ;000004;020000;000000;000000;000000;000000; % A-REG 1 LEAST SIGN.
8  'A,AL#10    ;000005;000000;000000;000000;000000;000000; % A-REG 10 LEAST SIGN.
9  'A,AL#11    ;000005;020000;000000;000000;000000;000000; % A-REG 11 LEAST SIGN.
10 'A,AL#12    ;000005;040000;000000;000000;000000;000000; % A-REG 12 LEAST SIGN.
11 'A,AL#13    ;000005;060000;000000;000000;000000;000000; % A-REG 13 LEAST SIGN.
12 'A,AL#14    ;000005;100000;000000;000000;000000;000000; % A-REG 14 LEAST SIGN.
13 'A,AL#15    ;000005;120000;000000;000000;000000;000000; % A-REG 15 LEAST SIGN.
14 'A,AL#16    ;000005;140000;000000;000000;000000;000000; % A-REG 16 LEAST SIGN.
15 'A,AL#17    ;000005;160000;000000;000000;000000;000000; % A-REG 17 LEAST SIGN.
16 'A,AL#2     ;000004;040000;000000;000000;000000;000000; % A-REG 2 LEAST SIGN.
17 'A,AL#20    ;000006;000000;000000;000000;000000;000000; % A-REG 20 LEAST SIGN.
18 'A,AL#21    ;000006;020000;000000;000000;000000;000000; % A-REG 21 LEAST SIGN.
19 'A,AL#22    ;000006;040000;000000;000000;000000;000000; % A-REG 22 LEAST SIGN.
20 'A,AL#23    ;000006;060000;000000;000000;000000;000000; % A-REG 23 LEAST SIGN.
21 'A,AL#24    ;000006;100000;000000;000000;000000;000000; % A-REG 24 LEAST SIGN.
22 'A,AL#25    ;000006;120000;000000;000000;000000;000000; % A-REG 25 LEAST SIGN.
23 'A,AL#26    ;000006;140000;000000;000000;000000;000000; % A-REG 26 LEAST SIGN.
24 'A,AL#27    ;000006;160000;000000;000000;000000;000000; % A-REG 27 LEAST SIGN.
25 'A,AL#3     ;000004;060000;000000;000000;000000;000000; % A-REG 3 LEAST SIGN.
26 'A,AL#30    ;000007;000000;000000;000000;000000;000000; % A-REG 30 LEAST SIGN.
27 'A,AL#31    ;000007;020000;000000;000000;000000;000000; % A-REG 31 LEAST SIGN.
28 'A,AL#32    ;000007;040000;000000;000000;000000;000000; % A-REG 32 LEAST SIGN.
29 'A,AL#33    ;000007;060000;000000;000000;000000;000000; % A-REG 33 LEAST SIGN.
30 'A,AL#34    ;000007;100000;000000;000000;000000;000000; % A-REG 34 LEAST SIGN.
31 'A,AL#35    ;000007;120000;000000;000000;000000;000000; % A-REG 35 LEAST SIGN.
32 'A,AL#36    ;000007;140000;000000;000000;000000;000000; % A-REG 36 LEAST SIGN.
33 'A,AL#37    ;000007;160000;000000;000000;000000;000000; % A-REG 37 LEAST SIGN.
34 'A,AL#4     ;000004;100000;000000;000000;000000;000000; % A-REG 4 LEAST SIGN.
35 'A,AL#5     ;000004;120000;000000;000000;000000;000000; % A-REG 5 LEAST SIGN.
36 'A,AL#6     ;000004;140000;000000;000000;000000;000000; % A-REG 6 LEAST SIGN.
37 'A,AL#7     ;000004;160000;000000;000000;000000;000000; % A-REG 7 LEAST SIGN.
38 'A,AM#0     ;000000;000000;000000;000000;000000;000000; % A-REG 0 MOST SIGN.
39 'A,AM#1     ;000000;020000;000000;000000;000000;000000; % A-REG 1 MOST SIGN.
40 'A,AM#10    ;000001;000000;000000;000000;000000;000000; % A-REG 10 MOST SIGN.
41 'A,AM#11    ;000001;020000;000000;000000;000000;000000; % A-REG 11 MOST SIGN.
42 'A,AM#12    ;000001;040000;000000;000000;000000;000000; % A-REG 12 MOST SIGN.
43 'A,AM#13    ;000001;060000;000000;000000;000000;000000; % A-REG 13 MOST SIGN.
44 'A,AM#14    ;000001;100000;000000;000000;000000;000000; % A-REG 14 MOST SIGN.
45 'A,AM#15    ;000001;120000;000000;000000;000000;000000; % A-REG 15 MOST SIGN.
46 'A,AM#16    ;000001;140000;000000;000000;000000;000000; % A-REG 16 MOST SIGN.
47 'A,AM#17    ;000001;160000;000000;000000;000000;000000; % A-REG 17 MOST SIGN.
48 'A,AM#2     ;000000;040000;000000;000000;000000;000000; % A-REG 2 MOST SIGN.
49 'A,AM#20    ;000002;000000;000000;000000;000000;000000; % A-REG 20 MOST SIGN.
50 'A,AM#21    ;000002;020000;000000;000000;000000;000000; % A-REG 21 MOST SIGN.
51 'A,AM#22    ;000002;040000;000000;000000;000000;000000; % A-REG 22 MOST SIGN.
52 'A,AM#23    ;000002;060000;000000;000000;000000;000000; % A-REG 23 MOST SIGN.
53 'A,AM#24    ;000002;100000;000000;000000;000000;000000; % A-REG 24 MOST SIGN.
54 'A,AM#25    ;000002;120000;000000;000000;000000;000000; % A-REG 25 MOST SIGN.
55 'A,AM#26    ;000002;140000;000000;000000;000000;000000; % A-REG 26 MOST SIGN.
56 'A,AM#27    ;000002;160000;000000;000000;000000;000000; % A-REG 27 MOST SIGN.
57 'A,AM#3     ;000000;060000;000000;000000;000000;000000; % A-REG 3 MOST SIGN.
58 'A,AM#30    ;000003;000000;000000;000000;000000;000000; % A-REG 30 MOST SIGN.
59 'A,AM#31    ;000003;020000;000000;000000;000000;000000; % A-REG 31 MOST SIGN.
60 'A,AM#32    ;000003;040000;000000;000000;000000;000000; % A-REG 32 MOST SIGN.
```

ND-500 micro mnemonics in alphabetical and numerical order

```
'A,AM#33 ;000003;060000;000000;000000;000000;000000; % A-REG 33 MOST SIGN.
'A,AM#34 ;000003;100000;000000;000000;000000;000000; % A-REG 34 MOST SIGN.
'A,AM#35 ;000003;120000;000000;000000;000000;000000; % A-REG 35 MOST SIGN.
'A,AM#36 ;000003;140000;000000;000000;000000;000000; % A-REG 36 MOST SIGN.
'A,AM#37 ;000003;160000;000000;000000;000000;000000; % A-REG 37 MOST SIGN.
'A,AM#4  ;000000;100000;000000;000000;000000;000000; % A-REG 4 MOST SIGN.
'A,AM#5  ;000000;120000;000000;000000;000000;000000; % A-REG 5 MOST SIGN.
'A,AM#6  ;000000;140000;000000;000000;000000;000000; % A-REG 6 MOST SIGN.
'A,AM#7  ;000000;160000;000000;000000;000000;000000; % A-REG 7 MOST SIGN.
'A,B     ;000160;040000;000000;000000;000000;000000; % B REGISTER
'A,BM#0  ;000040;000000;000000;000000;000000;000000; % BIT MASK BIT 0
'A,BM#1  ;000040;020000;000000;000000;000000;000000; % BIT MASK BIT 1
'A,BM#10 ;000041;000000;000000;000000;000000;000000; % BIT MASK BIT 10
'A,BM#11 ;000041;020000;000000;000000;000000;000000; % BIT MASK BIT 11
'A,BM#12 ;000041;040000;000000;000000;000000;000000; % BIT MASK BIT 12
'A,BM#13 ;000041;060000;000000;000000;000000;000000; % BIT MASK BIT 13
'A,BM#14 ;000041;100000;000000;000000;000000;000000; % BIT MASK BIT 14
'A,BM#15 ;000041;120000;000000;000000;000000;000000; % BIT MASK BIT 15
'A,BM#16 ;000041;140000;000000;000000;000000;000000; % BIT MASK BIT 16
'A,BM#17 ;000041;160000;000000;000000;000000;000000; % BIT MASK BIT 17
'A,BM#2  ;000040;040000;000000;000000;000000;000000; % BIT MASK BIT 2
'A,BM#20 ;000042;000000;000000;000000;000000;000000; % BIT MASK BIT 20
'A,BM#21 ;000042;020000;000000;000000;000000;000000; % BIT MASK BIT 21
'A,BM#22 ;000042;040000;000000;000000;000000;000000; % BIT MASK BIT 22
'A,BM#23 ;000042;060000;000000;000000;000000;000000; % BIT MASK BIT 23
'A,BM#24 ;000042;100000;000000;000000;000000;000000; % BIT MASK BIT 24
'A,BM#25 ;000042;120000;000000;000000;000000;000000; % BIT MASK BIT 25
'A,BM#26 ;000042;140000;000000;000000;000000;000000; % BIT MASK BIT 26
'A,BM#27 ;000042;160000;000000;000000;000000;000000; % BIT MASK BIT 27
'A,BM#3  ;000040;060000;000000;000000;000000;000000; % BIT MASK BIT 3
'A,BM#30 ;000043;000000;000000;000000;000000;000000; % BIT MASK BIT 30
'A,BM#31 ;000043;020000;000000;000000;000000;000000; % BIT MASK BIT 31
'A,BM#32 ;000043;040000;000000;000000;000000;000000; % BIT MASK BIT 32
'A,BM#33 ;000043;060000;000000;000000;000000;000000; % BIT MASK BIT 33
'A,BM#34 ;000043;100000;000000;000000;000000;000000; % BIT MASK BIT 34
'A,BM#35 ;000043;120000;000000;000000;000000;000000; % BIT MASK BIT 35
'A,BM#36 ;000043;140000;000000;000000;000000;000000; % BIT MASK BIT 36
'A,BM#37 ;000043;160000;000000;000000;000000;000000; % BIT MASK BIT 37
'A,BM#4  ;000040;100000;000000;000000;000000;000000; % BIT MASK BIT 4
'A,BM#5  ;000040;120000;000000;000000;000000;000000; % BIT MASK BIT 5
'A,BM#6  ;000040;140000;000000;000000;000000;000000; % BIT MASK BIT 6
'A,BM#7  ;000040;160000;000000;000000;000000;000000; % BIT MASK BIT 7
'A,BMR   ;000044;000000;000000;000000;000000;000000; % DECODED BMR BIT
'A,DATA  ;000020;000000;000000;000000;000000;000000; % MEMORY DATA
'A,L     ;000160;000000;000000;000000;000000;000000; % L REGISTER
'A,ONE   ;000000;140000;000000;000000;000000;000000; % FLOATING 1.0
'A,P     ;000160;020000;000000;000000;000000;000000; % P REGISTER
'A,R     ;000160;060000;000000;000000;000000;000000; % R REGISTER
'A,SP    ;000160;120000;000000;000000;000000;000000; % SAVED P
'A,THA   ;000004;160000;000000;000000;000000;000000; % TRAP HANDLER REGISTER
'A,TOS   ;000000;160000;000000;000000;000000;000000; % TOP OF STACK
'A,X#0   ;000140;000000;000000;000000;000000;000000; % X REG 0
'A,X#1   ;000140;020000;000000;000000;000000;000000; % X REG 1
'A,X#2   ;000140;040000;000000;000000;000000;000000; % X REG 2
'A,X#3   ;000140;060000;000000;000000;000000;000000; % X REG 3
'A,XD,BMR ;000072;100000;000000;000000;000000;000000; % BIT MASK REGISTER
'A,XD,CAR ;000072;040000;000000;000000;000000;000000; % COMPUTED ADDRESS REGISTER
'A,XD,CONST ;000076;020000;000000;000000;000000;000000; % INSTRUCTION CONSTANT
'A,XD,CSBRK ;000074;040000;000000;000000;000000;000000; % CONTROL STORE BREAK REG
'A,XD,CSCNT ;000074;060000;000000;000000;000000;000000; % CONTROL STORE CONTROL REG
```

ND-500 micro mnemonics in alphabetical and numerical order

```
121  'A_XD_CSWA      ';000074;020000;000000;000000;000000;000000;000000; % CONTR. ST. WRITE ADDR REG  .
122  'A_XD_DCINHLL   ';000065;020000;000000;000000;000000;000000;000000; % DATA CACHE INHIBIT LOWER L..
123  'A_XD_DCINHLU   ';000065;000000;000000;000000;000000;000000;000000; % DATA CACHE INHIBIT UPPER L..
124  'A_XD_DHXA      ';000065;120000;000000;000000;000000;000000;000000; % DMM HASH OR STSB INDEX  .
125  'A_XD_DISP      ';000076;060000;000000;000000;000000;000000;000000; % DISPLACEMENT REG  .
126  'A_XD_DLADDR    ';000065;040000;000000;000000;000000;000000;000000; % DMM LOGICAL ADDRESS  .
127  'A_XD_DMSTS     ';000065;020000;000000;000000;000000;000000;000000; % DMM STATUS  .
128  'A_XD_DRADDR    ';000065;100000;000000;000000;000000;000000;000000; % DMM REAL ADDRESS (PHYSICAL).
129  'A_XD_DRADDRL   ';000065;120000;000000;000000;000000;000000;000000; % DATA REAL ADDR LEAST SIGNIF.
130  'A_XD_DRADDRM   ';000065;100000;000000;000000;000000;000000;000000; % DATA REAL ADDR MOST SIGNIF..
131  'A_XD_DSCRF     ';000065;000000;000000;000000;000000;000000;000000; % DMM SCRATCH F., 16 16-BIT RG
132  'A_XD_DSTSU     ';000064;060000;000000;000000;000000;000000;000000; % DATA MEM. STATUS REG. 0  .
133  'A_XD_DSTS1     ';000064;040000;000000;000000;000000;000000;000000; % DATA MEM. STATUS REG. 1  .
134  'A_XD_DSTS2     ';000064;020000;000000;000000;000000;000000;000000; % DATA MEM. STATUS REG. 2  .
135  'A_XD_DUPL      ';000065;140000;000000;000000;000000;000000;000000; % DATA UPPER PAGE LIMIT  .
136  'A_XD_DWIPGU    ';000065;060000;000000;000000;000000;000000;000000; % DMM WIP/PGU BROADSIDE, 16 PG
137  'A_XD_DZPA      ';000065;160000;000000;000000;000000;000000;000000; % DATA ZERO POINT ADJUST REG..
138  'A_XD_HL        ';000066;020000;000000;000000;000000;000000;000000; % HIGHER LIMIT REGISTER  .
139  'A_XD_ICINHLL   ';000063;020000;000000;000000;000000;000000;000000; % INSTR CACHE INHIBIT LOWER L.
140  'A_XD_ICINHLU   ';000063;000000;000000;000000;000000;000000;000000; % INSTR CACHE INHIBIT UPPER L.
141  'A_XD_IDAT      ';000076;000000;000000;000000;000000;000000;000000; % INSTRUCTION MEMORY DATA  .
142  'A_XD_IHXA      ';000063;120000;000000;000000;000000;000000;000000; % IMM HASH OR SISB INDEX  .
143  'A_XD_ILADDR    ';000063;040000;000000;000000;000000;000000;000000; % IMM LOGICAL ADDRESS  .
144  'A_XD_INSTS     ';000063;020000;000000;000000;000000;000000;000000; % IMM STATUS  .
145  'A_XD_INDXC#0   ';000073;000000;000000;000000;000000;000000;000000; % INDEX COUNTER 0  .
146  'A_XD_INDXC#1   ';000073;020000;000000;000000;000000;000000;000000; % INDEX COUNTER 1  .
147  'A_XD_INDXC#2   ';000073;040000;000000;000000;000000;000000;000000; % INDEX COUNTER 2  .
148  'A_XD_INDXC#3   ';000073;060000;000000;000000;000000;000000;000000; % INDEX COUNTER 3  .
149  'A_XD_IODIN     ';000074;000000;000000;000000;000000;000000;000000; % DATA-IN REG  .
150  'A_XD_IODOUT    ';000074;120000;000000;000000;000000;000000;000000; % DATA-OUT REG  .
151  'A_XD_IRADDR    ';000063;100000;000000;000000;000000;000000;000000; % IMM REAL ADDRESS (PHYSICAL).
152  'A_XD_IRADDRL   ';000063;120000;000000;000000;000000;000000;000000; % INSTR REAL ADDR LEAST SIGN..
153  'A_XD_IRADDRM   ';000063;100000;000000;000000;000000;000000;000000; % INSTR REAL ADDR MOST SIGN..
154  'A_XD_ISCRF     ';000062;000000;000000;000000;000000;000000;000000; % IMM SCRATCH F., 16 16-BIT RG
155  'A_XD_ISTSU     ';000062;060000;000000;000000;000000;000000;000000; % INSTR MEM. STATUS REG. 0  .
156  'A_XD_ISTS1     ';000062;040000;000000;000000;000000;000000;000000; % INSTR MEM. STATUS REG. 1  .
157  'A_XD_ISTS2     ';000062;020000;000000;000000;000000;000000;000000; % INSTR MEM. STATUS REG. 2  .
158  'A_XD_IUPL      ';000063;140000;000000;000000;000000;000000;000000; % INSTR UPPER PAGE LIMIT  .
159  'A_XD_IWIPGU    ';000063;060000;000000;000000;000000;000000;000000; % IMM WIP/PGU BROADSIDE, 16 PG
160  'A_XD_IZPA      ';000063;160000;000000;000000;000000;000000;000000; % INSTR ZERO POINT ADJUST REG.
161  'A_XD_LARG      ';000072;020000;000000;000000;000000;000000;000000; % LONG ARGUMENT (32 BITS)  .
162  'A_XD_LC        ';000076;040000;000000;000000;000000;000000;000000; % LOOP COUNTER  .
163  'A_XD_LL        ';000066;000000;000000;000000;000000;000000;000000; % LOWER LIMIT REG  .
164  'A_XD_MISTAT    ';000070;160000;000000;000000;000000;000000;000000; % MICRO STATUS REGISTER  .
165  'A_XD_MMOD      ';000070;100000;000000;000000;000000;000000;000000; % MEMORY MODUS REGISTER  .
166  'A_XD_NBY       ';000070;120000;000000;000000;000000;000000;000000; % NO OF BYTES IN LAST MEM REF.
167  'A_XD_PSTAT     ';000072;120000;000000;000000;000000;000000;000000; % PREFETCH STATUS  .
168  'A_XD_S1        ';000070;000000;000000;000000;000000;000000;000000; % STATUS REG ONE  .
169  'A_XD_S2        ';000070;020000;000000;000000;000000;000000;000000; % STATUS REG TWO  .
170  'A_XD_SARG      ';000072;000000;000000;000000;000000;000000;000000; % SHORT ARGUMENT (16 BITS)  .
171  'A_XD_SHC       ';000072;060000;000000;000000;000000;000000;000000; % SHIFT COUNT REGISTER  .
172  'A_XD_TE        ';000070;040000;000000;000000;000000;000000;000000; % LOCAL TRAP ENABLE REG  .
173  'A_XD_TRAPINF   ';000070;140000;000000;000000;000000;000000;000000; % TRAP INFORMATION  .
174  'A_ZRO          ';000000;120000;000000;000000;000000;000000;000000; % FLOATING 0.0  .
175  'AA+AB          ';000000;000000;000000;000000;000014;000000;000000; % ADDR AOPR + ADDR BOPR  .
176  'AA_DP1         ';000000;000000;000000;000000;000000;000000;000000; % ADDR AOPR IS DP1 REG  .
177  'AA_DP2         ';000000;000000;000000;000000;010000;000000;000000; % ADDR AOPR IS DP2 REG  .
178  'AA_EA1         ';000000;000000;000000;000000;004000;000000;000000; % ADDR AOPR IS EFF. ADDR 1 REG
179  'AA_EA2         ';000000;000000;000000;000000;014000;000000;000000; % ADDR AOPR IS EFF. ADDR 2 REG
180  'AB_1/8IX       ';000000;000000;000000;000000;020000;000000;000000; % ADDR BOPR IS XREG SCALED 1/8
```

ND-500 micro mnemonics in alphabetical and numerical order

```
181  'AB,2IX     ';000000;000000;000000;000000;040000;000000; % ADDR BOPR IS XREG SC. BY 2 .
182  'AB,4IX     ';000000;000000;000000;000000;100000;000000; % ADDR BOPR IS XREG SC. BY 4 .
183  'AB,8IX     ';000000;000000;000000;000000;140000;000000; % ADDR BOPR IS XREG SC. BY 8 .
184  'AB,B       ';000000;000000;000000;000001;040000;000000; % ADDR BOPR IS B REGISTER . .
185  'AB,DPARG   ';000000;000000;000000;000001;000000;000000; % ADDR BOPR IS DP . . . . .
186  'AB,IX      ';000000;000000;000000;000001;100000;000000; % ADDR BOPR IS INDEX REG . .
187  'AB,ORAB    ';000000;000000;000000;000001;060000;000000; % ADDR BOPR IS OR-CONTROLLED .
188  'AB,PC      ';000000;000000;000000;000001;140000;000000; % ADDR BOPR IS P REGISTER . .
189  'AB,R       ';000000;000000;000000;000001;100000;000000; % ADDR BOPR IS R REGISTER . .
190  'AD,ILC     ';000066;140000;000000;000000;000000;000000; % DEST IS INSTR LOOK AHEAD A.C
191  'AD,NPC     ';000066;040000;000000;000000;000000;000000; % DEST IS NEXT PROGRAM COUNTER
192  'AD,PC      ';000070;060000;000000;000000;000000;000000; % DEST IS PROGRAM COUNTER . .
193  'ALTMOD     ';000000;000002;000000;000000;000000;000000; % SELECT BIT 1 IN MEMORY MODUS
194  'ALU,A+1    ';017200;000000;000000;000000;000000;000000; % INCREMENT AOPR . . . . . .
195  'ALU,A+A    ';014000;000000;000000;000000;000000;000000; % AOPR + AOPR . . . . . .
196  'ALU,A+A+1  ';014200;000000;000000;000000;000000;000000; % AOPR + AOPR + 1 . . . . .
197  'ALU,A+B    ';011000;000000;000000;000000;000000;000000; % AOPR + BOPR . . . . . .
198  'ALU,A+B+1  ';011200;000000;000000;000000;000000;000000; % AOPR + BOPR + 1 . . . . .
199  'ALU,A+B+C  ';011400;000000;000000;000000;000000;000000; % AOPR + BOPR + CARRY . . . .
200  'ALU,A-1    ';000000;000000;000000;000000;000000;000000; % DECREMENT AOPR . . . . . .
201  'ALU,A-B    ';006200;000000;000000;000000;000000;000000; % AOPR - BOPR . . . . . .
202  'ALU,A-B-1  ';006000;000000;000000;000000;000000;000000; % AOPR - BOPR - 1 . . . . .
203  'ALU,A-B-1+C';006400;000000;000000;000000;000000;000000; % AOPR - BOPR - 1 + CARRY . .
204  'ALU,ADIR   ';037000;000000;000000;000000;000000;000000; % AOPR DIRECT THROUGH ALU . .
205  'ALU,ADIRC  ';020000;000000;000000;000000;000000;000000; % AOPR COMPLEMENTED THR. ALU .
206  'ALU,AND    ';036000;000000;000000;000000;000000;000000; % AOPR AND BOPR . . . . . .
207  'ALU,ANDCA  ';030000;000000;000000;000000;000000;000000; % COMPL(AOPR) AND BOPR . . .
208  'ALU,ANDCB  ';035000;000000;000000;000000;000000;000000; % AOPR AND COMPL(BOPR) . . .
209  'ALU,BDIR   ';032000;000000;000000;000000;000000;000000; % BOPR DIRECT THROUGH ALU . .
210  'ALU,BDIRC  ';025000;000000;000000;000000;000000;000000; % BOPR COMPLEMENTED THR. ALU .
211  'ALU,FONE   ';023000;000000;000000;000000;000000;000000; % FORCED ONE . . . . . .
212  'ALU,FZRO   ';034000;000000;000000;000000;000000;000000; % FORCED ZRO . . . . . .
213  'ALU,NAND   ';021000;000000;000000;000000;000000;000000; % AOPR NAND BOPR . . . . .
214  'ALU,NOR    ';024000;000000;000000;000000;000000;000000; % AOPR NOR BOPR . . . . . .
215  'ALU,OR     ';033000;000000;000000;000000;000000;000000; % AOPR OR BOPR . . . . . .
216  'ALU,ORCA   ';022000;000000;000000;000000;000000;000000; % COMPL(AOPR) OR BOPR . . .
217  'ALU,ORCB   ';027000;000000;000000;000000;000000;000000; % AOPR OR COMPL(BOPR) . . .
218  'ALU,XNOR   ';031000;000000;000000;000000;000000;000000; % AOPR EXNOR BOPR . . . . .
219  'ALU,XOR    ';026000;000000;000000;000000;000000;000000; % AOPR EXOR BOPR . . . . . .
220  'B,-1       ';000000;010100;000000;000000;000000;000000; % FLOATING -1.0 . . . . . .
221  'B,A#0      ';000000;010000;000000;000000;000000;000000; % DOUBLE FLOAT REG 0 . . . .
222  'B,A#1      ';000000;010020;000000;000000;000000;000000; % DOUBLE FLOAT REG 1 . . . .
223  'B,A#2      ';000000;010040;000000;000000;000000;000000; % DOUBLE FLOAT REG 2 . . . .
224  'B,A#3      ';000000;010060;000000;000000;000000;000000; % DOUBLE FLOAT REG 3 . . . .
225  'B,AL#0     ';000000;011000;000000;000000;000000;000000; % A-REG 0 LEAST SIGN. . . . .
226  'B,AL#1     ';000000;011020;000000;000000;000000;000000; % A-REG 1 LEAST SIGN. . . . .
227  'B,AL#10    ';000000;011200;000000;000000;000000;000000; % A-REG 10 LEAST SIGN. . . . .
228  'B,AL#11    ';000000;011220;000000;000000;000000;000000; % A-REG 11 LEAST SIGN. . . . .
229  'B,AL#12    ';000000;011240;000000;000000;000000;000000; % A-REG 12 LEAST SIGN. . . . .
230  'B,AL#13    ';000000;011260;000000;000000;000000;000000; % A-REG 13 LEAST SIGN. . . . .
231  'B,AL#14    ';000000;011300;000000;000000;000000;000000; % A-REG 14 LEAST SIGN. . . . .
232  'B,AL#15    ';000000;011320;000000;000000;000000;000000; % A-REG 15 LEAST SIGN. . . . .
233  'B,AL#16    ';000000;011340;000000;000000;000000;000000; % A-REG 16 LEAST SIGN. . . . .
234  'B,AL#17    ';000000;011360;000000;000000;000000;000000; % A-REG 17 LEAST SIGN. . . . .
235  'E,AL#2     ';000000;011040;000000;000000;000000;000000; % A-REG 2 LEAST SIGN. . . . .
236  'B,AL#20    ';000000;011400;000000;000000;000000;000000; % A-REG 20 LEAST SIGN. . . . .
237  'B,AL#21    ';000000;011420;000000;000000;000000;000000; % A-REG 21 LEAST SIGN. . . . .
238  'B,AL#22    ';000000;011440;000000;000000;000000;000000; % A-REG 22 LEAST SIGN. . . . .
239  'B,AL#23    ';000000;011460;000000;000000;000000;000000; % A-REG 23 LEAST SIGN. . . . .
240  'B,AL#24    ';000000;011500;000000;000000;000000;000000; % A-REG 24 LEAST SIGN. . . . .
```

ND-500 micro mnemonics in alphabetical and numerical order

```
241  'B,AL#25  ;000000;011520;000000;000000;000000;000000;000000;000000; % A-REG 25 LEAST SIGN.
242  'F,AL#26  ;000000;011540;000000;000000;000000;000000;000000;000000; % A-REG 26 LEAST SIGN.
243  'B,AL#27  ;000000;011560;000000;000000;000000;000000;000000;000000; % A-REG 27 LEAST SIGN.
244  'B,AL#3   ;000000;011060;000000;000000;000000;000000;000000;000000; % A-REG 3 LEAST SIGN.
245  'F,AL#30  ;000000;011600;000000;000000;000000;000000;000000;000000; % A-REG 30 LEAST SIGN.
246  'B,AL#31  ;000000;011620;000000;000000;000000;000000;000000;000000; % A-REG 31 LEAST SIGN.
247  'B,AL#32  ;000000;011640;000000;000000;000000;000000;000000;000000; % A-REG 32 LEAST SIGN.
248  'F,AL#33  ;000000;011660;000000;000000;000000;000000;000000;000000; % A-REG 33 LEAST SIGN.
249  'B,AL#34  ;000000;011700;000000;000000;000000;000000;000000;000000; % A-REG 34 LEAST SIGN.
250  'B,AL#35  ;000000;011720;000000;000000;000000;000000;000000;000000; % A-REG 35 LEAST SIGN.
251  'F,AL#36  ;000000;011740;000000;000000;000000;000000;000000;000000; % A-REG 36 LEAST SIGN.
252  'B,AL#37  ;000000;011760;000000;000000;000000;000000;000000;000000; % A-REG 37 LEAST SIGN.
253  'B,AL#4   ;000000;011100;000000;000000;000000;000000;000000;000000; % A-REG 4 LEAST SIGN.
254  'F,AL#5   ;000000;011120;000000;000000;000000;000000;000000;000000; % A-REG 5 LEAST SIGN.
255  'B,AL#6   ;000000;011140;000000;000000;000000;000000;000000;000000; % A-REG 6 LEAST SIGN.
256  'B,AL#7   ;000000;011160;000000;000000;000000;000000;000000;000000; % A-REG 7 LEAST SIGN.
257  'F,AM#0   ;000000;010000;000000;000000;000000;000000;000000;000000; % A-REG 0 MOST SIGN.
258  'B,AM#1   ;000000;010020;000000;000000;000000;000000;000000;000000; % A-REG 1 MOST SIGN.
259  'B,AM#10  ;000000;010200;000000;000000;000000;000000;000000;000000; % A-REG 10 MOST SIGN.
260  'F,AM#11  ;000000;010220;000000;000000;000000;000000;000000;000000; % A-REG 11 MOST SIGN.
261  'B,AM#12  ;000000;010240;000000;000000;000000;000000;000000;000000; % A-REG 12 MOST SIGN.
262  'B,AM#13  ;000000;010260;000000;000000;000000;000000;000000;000000; % A-REG 13 MOST SIGN.
263  'F,AM#14  ;000000;010300;000000;000000;000000;000000;000000;000000; % A-REG 14 MOST SIGN.
264  'B,AM#15  ;000000;010320;000000;000000;000000;000000;000000;000000; % A-REG 15 MOST SIGN.
265  'B,AM#16  ;000000;010340;000000;000000;000000;000000;000000;000000; % A-REG 16 MOST SIGN.
266  'B,AM#17  ;000000;010360;000000;000000;000000;000000;000000;000000; % A-REG 17 MOST SIGN.
267  'B,AM#2   ;000000;010040;000000;000000;000000;000000;000000;000000; % A-REG 2 MOST SIGN.
268  'B,AM#20  ;000000;010400;000000;000000;000000;000000;000000;000000; % A-REG 20 MOST SIGN.
269  'F,AM#21  ;000000;010420;000000;000000;000000;000000;000000;000000; % A-REG 21 MOST SIGN.
270  'B,AM#22  ;000000;010440;000000;000000;000000;000000;000000;000000; % A-REG 22 MOST SIGN.
271  'B,AM#23  ;000000;010460;000000;000000;000000;000000;000000;000000; % A-REG 23 MOST SIGN.
272  'F,AM#24  ;000000;010500;000000;000000;000000;000000;000000;000000; % A-REG 24 MOST SIGN.
273  'B,AM#25  ;000000;010520;000000;000000;000000;000000;000000;000000; % A-REG 25 MOST SIGN.
274  'B,AM#26  ;000000;010540;000000;000000;000000;000000;000000;000000; % A-REG 26 MOST SIGN.
275  'F,AM#27  ;000000;010560;000000;000000;000000;000000;000000;000000; % A-REG 27 MOST SIGN.
276  'B,AM#3   ;000000;010060;000000;000000;000000;000000;000000;000000; % A-REG 3 MOST SIGN.
277  'B,AM#30  ;000000;010600;000000;000000;000000;000000;000000;000000; % A-REG 30 MOST SIGN.
278  'F,AM#31  ;000000;010620;000000;000000;000000;000000;000000;000000; % A-REG 31 MOST SIGN.
279  'B,AM#32  ;000000;010640;000000;000000;000000;000000;000000;000000; % A-REG 32 MOST SIGN.
280  'B,AM#33  ;000000;010660;000000;000000;000000;000000;000000;000000; % A-REG 33 MOST SIGN.
281  'F,AM#34  ;000000;010700;000000;000000;000000;000000;000000;000000; % A-REG 34 MOST SIGN.
282  'B,AM#35  ;000000;010720;000000;000000;000000;000000;000000;000000; % A-REG 35 MOST SIGN.
283  'B,AM#36  ;000000;010740;000000;000000;000000;000000;000000;000000; % A-REG 36 MOST SIGN.
284  'F,AM#37  ;000000;010760;000000;000000;000000;000000;000000;000000; % A-REG 37 MOST SIGN.
285  'B,AM#4   ;000000;010100;000000;000000;000000;000000;000000;000000; % A-REG 4 MOST SIGN.
286  'F,AM#5   ;000000;010120;000000;000000;000000;000000;000000;000000; % A-REG 5 MOST SIGN.
287  'B,AM#6   ;000000;010140;000000;000000;000000;000000;000000;000000; % A-REG 6 MOST SIGN.
288  'B,AM#7   ;000000;010160;000000;000000;000000;000000;000000;000000; % A-REG 7 MOST SIGN.
289  'B,BM#0   ;000000;004000;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 0
290  'B,BM#1   ;000000;004020;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 1
291  'B,BM#10  ;000000;004200;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 10
292  'B,BM#11  ;000000;004220;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 11
293  'F,BM#12  ;000000;004240;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 12
294  'B,BM#13  ;000000;004260;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 13
295  'B,BM#14  ;000000;004300;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 14
296  'R,BM#15  ;000000;004320;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 15
297  'B,BM#16  ;000000;004340;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 16
298  'B,BM#17  ;000000;004360;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 17
299  'F,BM#2   ;000000;004040;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 2
300  'B,BM#20  ;000000;004400;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 20
```

ND-500 micro mnemonics in alphabetical and numerical order

```
'E,BM#21  ;000000;004420;000000;000000;000000;000000;000000; % BIT MASK BIT 21
'B,BM#22  ;000000;004440;000000;000000;000000;000000;000000; % BIT MASK BIT 22
'E,BM#23  ;000000;004460;000000;000000;000000;000000;000000; % BIT MASK BIT 23
'E,BM#24  ;000000;004500;000000;000000;000000;000000;000000; % BIT MASK BIT 24
'B,BM#25  ;000000;004520;000000;000000;000000;000000;000000; % BIT MASK BIT 25
'E,BM#26  ;000000;004540;000000;000000;000000;000000;000000; % BIT MASK BIT 26
'B,BM#27  ;000000;004560;000000;000000;000000;000000;000000; % BIT MASK BIT 27
'B,BM#3   ;000000;004060;000000;000000;000000;000000;000000; % BIT MASK BIT 3
'B,BM#30  ;000000;004600;000000;000000;000000;000000;000000; % BIT MASK BIT 30
'E,BM#31  ;000000;004620;000000;000000;000000;000000;000000; % BIT MASK BIT 31
'B,BM#32  ;000000;004640;000000;000000;000000;000000;000000; % BIT MASK BIT 32
'E,BM#33  ;000000;004660;000000;000000;000000;000000;000000; % BIT MASK BIT 33
'E,BM#34  ;000000;004700;000000;000000;000000;000000;000000; % BIT MASK BIT 34
'B,BM#35  ;000000;004720;000000;000000;000000;000000;000000; % BIT MASK BIT 35
'E,BM#36  ;000000;004740;000000;000000;000000;000000;000000; % BIT MASK BIT 36
'B,BM#37  ;000000;004760;000000;000000;000000;000000;000000; % BIT MASK BIT 37
'B,BM#4   ;000000;004100;000000;000000;000000;000000;000000; % BIT MASK BIT 4
'B,BM#5   ;000000;004120;000000;000000;000000;000000;000000; % BIT MASK BIT 5
'E,BM#6   ;000000;004140;000000;000000;000000;000000;000000; % BIT MASK BIT 6
'B,BM#7   ;000000;004160;000000;000000;000000;000000;000000; % BIT MASK BIT 7
'E,BMR    ;000000;005000;000000;000000;000000;000000;000000; % DECODED BMR BIT
'E,DATA   ;000000;002000;000000;000000;000000;000000;000000; % MEMORY DATA
'E,DATA2  ;000000;002020;000000;000000;000000;000000;000000; % EXTRA MEMORY DATA REGISTER
'B,EA1    ;000000;006000;000000;000000;000000;000000;000000; % EFFECTIVE ADDR REG 1
'E,EA2    ;000000;006020;000000;000000;000000;000000;000000; % EFFECTIVE ADDR REG 2
'E,ONE    ;000000;010140;000000;000000;000000;000000;000000; % FLOATING 1.0
'B,THA    ;000000;011160;000000;000000;000000;000000;000000; % TRAP HANDLER EG
'E,TOS    ;000000;010160;000000;000000;000000;000000;000000; % TOP OF STACK
'B,X#0    ;000000;012000;000000;000000;000000;000000;000000; % INDEX REG 0
'B,X#1    ;000000;012020;000000;000000;000000;000000;000000; % INDEX REG 1
'E,X#2    ;000000;012040;000000;000000;000000;000000;000000; % INDEX REG 2
'B,X#3    ;000000;012060;000000;000000;000000;000000;000000; % INDEX REG 3
'B,XRESL  ;000000;016100;000000;000000;000000;000040;000000; % EXT. RESULT LEAST SIGN.
'B,XRESM  ;000000;016000;000000;000000;000000;000040;000000; % EXT. RESULT MOST SIGN.
'B,ZRO    ;000000;010120;000000;000000;000000;000040;000000; % FLOATING 0.0
'ACD      ;000000;000000;000000;000000;001000;000000;000000; % BCD TYPE
'BYTH     ;000000;000100;000000;000000;000000;000000;000000; % BYTE TO HALFW. SIGN EXT.
'BYTW     ;000000;000200;000000;000000;000000;000000;000000; % BYTE TO WORD SIGN EXTENSION
'C,ALU    ;000000;000000;002000;000000;000000;000000;000000; % CONDITIONAL ALU
'C,MEM    ;000000;000000;001000;000000;000000;000000;000000; % CONDITIONAL MEMORY REFERENCE
'C,SEQ    ;000000;000000;004000;000000;000000;000000;000000; % CONDITIONAL SEQUENCE
'C,SEQD   ;000000;000000;004000;002000;004000;000000;000000; % DELAYED COND. SEQUENCE
'COND,CNZ  ;000000;000000;000220;000000;000000;000000;000000; % CARRY AND NOT ZERO
'COND,CONOP;000000;000000;000260;000000;000000;000000;000000; % CONSTANT OPERAND
'COND,CRY  ;000000;000000;000060;000000;000000;000000;000000; % CARRY
'COND,DATOP;000000;000000;000160;000000;000000;000000;000000; % DATA OPERAND
'COND,ENTER;000000;000000;000020;000000;000000;000000;000000; % NEXT MACRO INSTR IS ENTER
'COND,ICRDY;000000;000000;000640;000000;000000;000000;000000; % INSTRUCTION MEMORY READY
'COND,K    ;000000;000000;000140;000000;000000;000000;000000; % TEST COND. IS K (FLAG)
'COND,LCZ  ;000000;000000;000760;000000;000000;000000;000000; % LOOP COUNTER ZERO
'COND,MBO  ;000000;000000;000600;000000;000000;000000;000000; % MICRO BCD OVERFLOW
'COND,MBS  ;000000;000000;000620;000000;000000;000000;000000; % MICRO BCD SIGN
'COND,MCNZ ;000000;000000;000460;000000;000000;000000;000000; % MICRO CARRY AND NOT M. ZERO.
'COND,MCRY ;000000;000000;000320;000000;000000;000000;000000; % MICRO CARRY
'COND,MDZ  ;000000;000000;000660;000000;000000;000000;000000; % DIVIDE BY ZRO
'COND,MFO  ;000000;000000;000540;000000;000000;000000;000000; % MICRO FLOAT OVERFLOW
'COND,MFS  ;000000;000000;000520;000000;000000;000000;000000; % MICRO FLOATING SIGN
'COND,MFU  ;000000;000000;000560;000000;000000;000000;000000; % MICRO FLOAT UNDERFLOW
'COND,MFUFO;000000;000000;000240;000000;000000;000000;000000; % MICRO FLOAT UNDER/OVERFLOW
'COND,MOVFL;000000;000000;000360;000000;000000;000000;000000; % MICRO OVERFLOW
```

ND-500 micro mnemonics in alphabetical and numerical order

```
'COND_MSEXO  ';000000;000000;000400;000000;000000;000000; % M. SIGN EXOR M. OVERFLOW
'COND_MSGN   ';000000;000000;000340;000000;000000;000000; % MICRO SIGN
'COND_MSORZ  ';000000;000000;000440;000000;000000;000000; % M. SIGN OR M. ZERO
'COND_MZRO   ';000000;000000;000300;000000;000000;000000; % MICRO ZERO
'COND_OVFL   ';000000;000000;000126;000000;000000;000000; % OVERFLOW
'COND_PARITY ';000000;000000;000700;000000;000000;000000; % PARITY OF ALU OUTPUT
'COND_PDONE  ';000000;000000;000500;000000;000000;000000; % PART DONE
'COND_SAVC1  ';000000;000000;000720;000000;000000;000000; % SAVED CONDITION 1
'COND_SAVC2  ';000000;000000;000740;000000;000000;000000; % SAVED CONDITION 2
'COND_SGN    ';000000;000000;000100;000000;000000;000000; % SIGN
'COND_SORZ   ';000000;000000;000200;000000;000000;000000; % SIGN OR ZERO
'COND_TRAP   ';000000;000000;000420;000000;000000;000000; % TEST CONDITION IS TRAP
'COND_ZRO    ';000000;000000;000040;000000;000000;000000; % ZERO
'CRY_MISTAT  ';000800;000000;000000;000000;000000;000000; % CARRY FROM MICRO STATUS
'CRY_ONE     ';002000;000000;000000;000000;000000;000000; % CARRY IS FORCED ONE
'CRY_STAT    ';004000;000000;000000;000000;000000;000000; % CARRY FROM STATUS
'CRY_ZRO     ';000000;000000;000000;000000;000000;000000; % CARRY IS FORCED ZERO
'CSAVE       ';000000;000000;010000;000000;000000;000000; % CONDITION SAVE (PUSH)
'D_AD#0      ';000000;000012;000000;000000;000000;000000; % DEST IS A-REG 0 DOUBLE
'D_AD#1      ';000000;000012;002000;000000;000000;000000; % DEST IS A-REG 1 DOUBLE
'D_AD#10     ';000000;000012;020000;000000;000000;000000; % DEST IS A-REG 10 DOUBLE
'D_AD#11     ';000000;000012;022000;000000;000000;000000; % DEST IS A-REG 11 DOUBLE
'D_AD#12     ';000000;000012;024000;000000;000000;000000; % DEST IS A-REG 12 DOUBLE
'D_AD#13     ';000000;000012;026000;000000;000000;000000; % DEST IS A-REG 13 DOUBLE
'D_AD#14     ';000000;000012;030000;000000;000000;000000; % DEST IS A-REG 14 DOUBLE
'D_AD#15     ';000000;000012;032000;000000;000000;000000; % DEST IS A-REG 15 DOUBLE
'D_AD#16     ';000000;000012;034000;000000;000000;000000; % DEST IS A-REG 16 DOUBLE
'D_AD#17     ';000000;000012;036000;000000;000000;000000; % DEST IS A-REG 17 DOUBLE
'D_AD#2      ';000000;000012;004000;000000;000000;000000; % DEST IS A-REG 2 DOUBLE
'D_AD#20     ';000000;000012;040000;000000;000000;000000; % DEST IS A-REG 20 DOUBLE
'D_AD#21     ';000000;000012;042000;000000;000000;000000; % DEST IS A-REG 21 DOUBLE
'D_AD#22     ';000000;000012;044000;000000;000000;000000; % DEST IS A-REG 22 DOUBLE
'D_AD#23     ';000000;000012;046000;000000;000000;000000; % DEST IS A-REG 23 DOUBLE
'D_AD#24     ';000000;000012;050000;000000;000000;000000; % DEST IS A-REG 24 DOUBLE
'D_AD#25     ';000000;000012;052000;000000;000000;000000; % DEST IS A-REG 25 DOUBLE
'D_AD#26     ';000000;000012;054000;000000;000000;000000; % DEST IS A-REG 26 DOUBLE
'D_AD#27     ';000000;000012;056000;000000;000000;000000; % DEST IS A-REG 27 DOUBLE
'D_AD#3      ';000000;000012;006000;000000;000000;000000; % DEST IS A-REG 3 DOUBLE
'D_AD#30     ';000000;000012;060000;000000;000000;000000; % DEST IS A-REG 30 DOUBLE
'D_AD#31     ';000000;000012;062000;000000;000000;000000; % DEST IS A-REG 31 DOUBLE
'D_AD#32     ';000000;000012;064000;000000;000000;000000; % DEST IS A-REG 32 DOUBLE
'D_AD#33     ';000000;000012;066000;000000;000000;000000; % DEST IS A-REG 33 DOUBLE
'D_AD#34     ';000000;000012;070000;000000;000000;000000; % DEST IS A-REG 34 DOUBLE
'D_AD#35     ';000000;000012;072000;000000;000000;000000; % DEST IS A-REG 35 DOUBLE
'D_AD#36     ';000000;000012;074000;000000;000000;000000; % DEST IS A-REG 36 DOUBLE
'D_AD#37     ';000000;000012;076000;000000;000000;000000; % DEST IS A-REG 37 DOUBLE
'D_AD#4      ';000000;000012;010000;000000;000000;000000; % DEST IS A-REG 4 DOUBLE
'D_AD#5      ';000000;000012;012000;000000;000000;000000; % DEST IS A-REG 5 DOUBLE
'D_AD#6      ';000000;000012;014000;000000;000000;000000; % DEST IS A-REG 6 DOUBLE
'D_AD#7      ';000000;000012;016000;000000;000000;000000; % DEST IS A-REG 7 DOUBLE
'D_AL#0      ';000000;000010;000000;000000;000000;000000; % A0 LEAST
'D_AL#1      ';000000;000010;002000;000000;000000;000000; % A1 LEAST
'D_AL#10     ';000000;000010;020000;000000;000000;000000; % A10 LEAST
'D_AL#11     ';000000;000010;022000;000000;000000;000000; % A11 LEAST
'D_AL#12     ';000000;000010;024000;000000;000000;000000; % A12 LEAST
'D_AL#13     ';000000;000010;026000;000000;000000;000000; % A13 LEAST
'D_AL#14     ';000000;000010;030000;000000;000000;000000; % A14 LEAST
'D_AL#15     ';000000;000010;032000;000000;000000;000000; % A15 LEAST
'D_AL#16     ';000000;000010;034000;000000;000000;000000; % A16 LEAST
'D_AL#17     ';000000;000010;036000;000000;000000;000000; % A17 LEAST
```

ND-500 micro mnemonics in alphabetical and numerical order

```
421 'D_AL#2
422 'D_AL#20
423 'D_AL#21
424 'D_AL#22
425 'D_AL#23
426 'D_AL#24
427 'D_AL#25
428 'D_AL#26
429 'D_AL#27
430 'D_AL#3
431 'D_AL#30
432 'D_AL#31
433 'D_AL#32
434 'D_AL#33
435 'D_AL#34
436 'D_AL#35
437 'D_AL#36
438 'D_AL#37
439 'D_AL#4
440 'D_AL#5
441 'D_AL#6
442 'D_AL#7
443 'D_AM#0
444 'D_AM#1
445 'D_AM#10
446 'D_AM#11
447 'D_AM#12
448 'D_AM#13
449 'D_AM#14
450 'D_AM#15
451 'D_AM#16
452 'D_AM#17
453 'D_AM#2
454 'D_AM#20
455 'D_AM#21
456 'D_AM#22
457 'D_AM#23
458 'D_AM#24
459 'D_AM#25
460 'D_AM#26
461 'D_AM#27
462 'D_AM#3
463 'D_AM#30
464 'D_AM#31
465 'D_AM#32
466 'D_AM#33
467 'D_AM#34
468 'D_AM#35
469 'D_AM#36
470 'D_AM#37
471 'D_AM#4
472 'D_AM#5
473 'D_AM#6
474 'D_AM#7
475 'D_B
476 'D_BMR
477 'D_CAR
478 'D_CONST
479 'D_CSBRK
480 'D_CSCNT

; 000000; 000010; 004000; 000000; 000000; 000000; 000000; % A2 LEAST
; 000000; 000010; 040000; 000000; 000000; 000000; 000000; % A20 LEAST
; 000000; 000010; 042000; 000000; 000000; 000000; 000000; % A21 LEAST
; 000000; 000010; 044000; 000000; 000000; 000000; 000000; % A22 LEAST
; 000000; 000010; 046000; 000000; 000000; 000000; 000000; % A23 LEAST
; 000000; 000010; 050000; 000000; 000000; 000000; 000000; % A24 LEAST
; 000000; 000010; 052000; 000000; 000000; 000000; 000000; % A25 LEAST
; 000000; 000010; 054000; 000000; 000000; 000000; 000000; % A26 LEAST
; 000000; 000010; 056000; 000000; 000000; 000000; 000000; % A27 LEAST
; 000000; 000010; 006000; 000000; 000000; 000000; 000000; % A3 LEAST
; 000000; 000010; 060000; 000000; 000000; 000000; 000000; % A30 LEAST
; 000000; 000010; 062000; 000000; 000000; 000000; 000000; % A31 LEAST
; 000000; 000010; 064000; 000000; 000000; 000000; 000000; % A32 LEAST
; 000000; 000010; 066000; 000000; 000000; 000000; 000000; % A33 LEAST
; 000000; 000010; 070000; 000000; 000000; 000000; 000000; % A34 LEAST
; 000000; 000010; 072000; 000000; 000000; 000000; 000000; % A35 LEAST
; 000000; 000010; 074000; 000000; 000000; 000000; 000000; % A36 LEAST
; 000000; 000010; 076000; 000000; 000000; 000000; 000000; % A37 LEAST
; 000000; 000010; 010000; 000000; 000000; 000000; 000000; % A4 LEAST
; 000000; 000010; 012000; 000000; 000000; 000000; 000000; % A5 LEAST
; 000000; 000010; 014000; 000000; 000000; 000000; 000000; % A6 LEAST
; 000000; 000010; 016000; 000000; 000000; 000000; 000000; % A7 LEAST
; 000000; 000006; 000000; 000000; 000000; 000000; 000000; % A0 MOST
; 000000; 000006; 002000; 000000; 000000; 000000; 000000; % A1 MOST
; 000000; 000006; 020000; 000000; 000000; 000000; 000000; % A10 MOST
; 000000; 000006; 022000; 000000; 000000; 000000; 000000; % A11 MOST
; 000000; 000006; 024000; 000000; 000000; 000000; 000000; % A12 MOST
; 000000; 000006; 026000; 000000; 000000; 000000; 000000; % A13 MOST
; 000000; 000006; 030000; 000000; 000000; 000000; 000000; % A14 MOST
; 000000; 000006; 032000; 000000; 000000; 000000; 000000; % A15 MOST
; 000000; 000006; 034000; 000000; 000000; 000000; 000000; % A16 MOST
; 000000; 000006; 036000; 000000; 000000; 000000; 000000; % A17 MOST
; 000000; 000006; 004000; 000000; 000000; 000000; 000000; % A2 MOST
; 000000; 000006; 040000; 000000; 000000; 000000; 000000; % A20 MOST
; 000000; 000006; 042000; 000000; 000000; 000000; 000000; % A21 MOST
; 000000; 000006; 044000; 000000; 000000; 000000; 000000; % A22 MOST
; 000000; 000006; 046000; 000000; 000000; 000000; 000000; % A23 MOST
; 000000; 000006; 050000; 000000; 000000; 000000; 000000; % A24 MOST
; 000000; 000006; 052000; 000000; 000000; 000000; 000000; % A25 MOST
; 000000; 000006; 054000; 000000; 000000; 000000; 000000; % A26 MOST
; 000000; 000006; 056000; 000000; 000000; 000000; 000000; % A27 MOST
; 000000; 000006; 006000; 000000; 000000; 000000; 000000; % A3 MOST
; 000000; 000006; 060000; 000000; 000000; 000000; 000000; % A30 MOST
; 000000; 000006; 062000; 000000; 000000; 000000; 000000; % A31 MOST
; 000000; 000006; 064000; 000000; 000000; 000000; 000000; % A32 MOST
; 000000; 000006; 066000; 000000; 000000; 000000; 000000; % A33 MOST
; 000000; 000006; 070000; 000000; 000000; 000000; 000000; % A34 MOST
; 000000; 000006; 072000; 000000; 000000; 000000; 000000; % A35 MOST
; 000000; 000006; 074000; 000000; 000000; 000000; 000000; % A36 MOST
; 000000; 000006; 076000; 000000; 000000; 000000; 000000; % A37 MOST
; 000000; 000006; 010000; 000000; 000000; 000000; 000000; % A4 MOST
; 000000; 000006; 012000; 000000; 000000; 000000; 000000; % A5 MOST
; 000000; 000006; 014000; 000000; 000000; 000000; 000000; % A6 MOST
; 000000; 000006; 016000; 000000; 000000; 000000; 000000; % A7 MOST
; 000000; 000004; 000000; 000000; 000000; 000000; 000000; % B REGISTER
; 000000; 000015; 050000; 000000; 000000; 000000; 000000; % BIT MASK REGISTER
; 000000; 000015; 042000; 000000; 000000; 000000; 000000; % COMPUTED ADDRESS REGISTER
; 000000; 000015; 144000; 000000; 000000; 000000; 000000; % CONSTANT REGISTER
; 000000; 000015; 104000; 000000; 000000; 000000; 000000; % CONTROL STORE BREAK REGISTER
; 000000; 000015; 110000; 000000; 000000; 000000; 000000; % CONTROL STORE CONTROL REG.
```

ND-500 micro mnemonics in alphabetical and numerical order

```
481  'D,CSWA     ;000000;000015;102000;000000;000000;000000;000000; % CONT. ST. WRITE ADDRESS REG.
482  'D,DADDM    ;000000;000014;124000;000000;000000;000000;000000; % DMM ALT. DOMAIN REGISTER  -
483  'D,DASEG    ;000000;000014;130000;000000;000000;000000;000000; % DMM ALT. SEGM.           -
484  'D,DATAIN   ;000000;000015;020000;000000;000000;000000;000000; % DATA-IN HOLD REG. ON SLICE -
485  'D,DATAIN2  ;000000;000015;022000;000000;000000;000000;000000; % EXTRA DATA-IN HOLD REG ON SL
486  'D,DCINHLL  ;000000;000014;102000;000000;000000;000000;000000; % DATA CACHE INHIBIT LOWER LIM
487  'D,DCINHLU  ;000000;000014;100000;000000;000000;000000;000000; % DATA CACHE INHIBIT UPPER LIM
488  'D,DCLCACH  ;000000;000014;112000;000000;000000;000000;000000; % DATA CLEAR CACHE           -
489  'D,DCON0    ;000000;000014;110000;000000;000000;000000;000000; % DATA MEM. CONTROL REG. 0    -
490  'D,DCON1    ;000000;000014;106000;000000;000000;000000;000000; % DATA MEM. CONTROL REG. 1    -
491  'D,DCSEG    ;000000;000014;126000;000000;000000;000000;000000; % DMM CURR. SEGMENT           -
492  'D,DDOMR    ;000000;000014;122000;000000;000000;000000;000000; % DMM DOMAIN REG              -
493  'D,DMCNTR   ;000000;000014;100000;000000;000000;000000;000000; % DMM CONTROL                 -
494  'D,DP       ;000000;000015;114000;000000;000000;000000;000000; % DISPLACEMENT REG           -
495  'D,DPROCC   ;000000;000014;120000;000000;000000;000000;000000; % DMM PROCESS CONTR. REG      -
496  'D,DSCFA    ;000000;000014;116000;000000;000000;000000;000000; % DMM SCR. FILE REGISTER ADDR.
497  'D,DSCRF    ;000000;000014;114000;000000;000000;000000;000000; % DMM SCR. FILE (16 16-BIT RG)
498  'D,DSTSB    ;000000;000014;136000;000000;000000;000000;000000; % DMM SEQ. TRANSL. SPEED-UP BF
499  'D,DTSB     ;000000;000014;132000;000000;000000;000000;000000; % DMM TRANSL. SPEED-UP BUF PAG
500  'D,DUPL     ;000000;000014;114000;000000;000000;000000;000000; % DATA UPPER PAGE LIMIT       -
501  'D,DWIPGU   ;000000;000014;134000;000000;000000;000000;000000; % DMM WIP/PGU BROADSIDE (16 P)
502  'D,DZPA     ;000000;000014;116000;000000;000000;000000;000000; % DATA ZERO POINT ADJUST REG..
503  'D,HL       ;000000;000014;142000;000000;000000;000000;000000; % HIGHER LIMIT REG           -
504  'D,IADDM    ;000000;000014;064000;000000;000000;000000;000000; % IMM ALT. DOMAIN REGISTER    -
505  'D,IASEG    ;000000;000014;070000;000000;000000;000000;000000; % IMM ALT. SEGM.              -
506  'D,ICCLR    ;000000;000015;044000;000000;000000;000000;000000; % INDEX COUNTER CLEAR         -
507  'D,ICINHLL  ;000000;000014;042000;000000;000000;000000;000000; % INSTR CACHE INHIBIT LOWER L.
508  'D,ICINHLU  ;000000;000014;040000;000000;000000;000000;000000; % INSTR CACHE INHIBIT UPPER L.
509  'D,ICLCACH  ;000000;000014;052000;000000;000000;000000;000000; % INSTR CLEAR CACHE            -
510  'D,ICON0    ;000000;000014;050000;000000;000000;000000;000000; % INSTR MEM. CONTROL REG. 0    -
511  'D,ICON1    ;000000;000014;046000;000000;000000;000000;000000; % INSTR MEM. CONTROL REG. 1    -
512  'D,ICSEG    ;000000;000014;066000;000000;000000;000000;000000; % IMM CURR. SEGMENT           -
513  'D,IDAT     ;000000;000015;142000;000000;000000;000000;000000; % INSTRUCTION MEMORY DATA     -
514  'D,IDOMR    ;000000;000014;062000;000000;000000;000000;000000; % IMM DOMAIN REG              -
515  'D,IMCNTR   ;000000;000014;040000;000000;000000;000000;000000; % IMM CONTROL                 -
516  'D,IODOUT   ;000000;000015;100000;000000;000000;000000;000000; % DATA-OUT                    -
517  'D,IPROCC   ;000000;000014;060000;000000;000000;000000;000000; % IMM PROCESS CONTR. REG      -
518  'D,ISCFA    ;000000;000014;056000;000000;000000;000000;000000; % IMM SCR. FILE REGISTER ADDR.
519  'D,ISCRF    ;000000;000014;054000;000000;000000;000000;000000; % IMM SCR. FILE (16 16-BIT KG)
520  'D,ISTSB    ;000000;000014;076000;000000;000000;000000;000000; % IMM SEQ. TRANSL. SPEED-UP BF
521  'D,ITSB     ;000000;000014;072000;000000;000000;000000;000000; % IMM TRANSL. SPEED-UP BUF PAG
522  'D,IUPL     ;000000;000014;054000;000000;000000;000000;000000; % INSTR. UPPER PAGE LIMIT     -
523  'D,IWIPGU   ;000000;000014;074000;000000;000000;000000;000000; % IMM WIP/PGU BROADSIDE (16 P)
524  'D,IZPA     ;000000;000014;056000;000000;000000;000000;000000; % INSTR. ZERO POINT ADJUST REG
525  'D,L        ;000000;000004;002000;000000;000000;000000;000000; % L REGISTER                  -
526  'D,LC       ;000000;000015;140000;000000;000000;000000;000000; % LOOP COUNTER                -
527  'D,LL       ;000000;000014;140000;000000;000000;000000;000000; % LOWER LIMIT REG            -
528  'D,MMOD     ;000000;000015;012000;000000;000000;000000;000000; % MEMORY MODUS REGISTER       -
529  'D,R        ;000000;000004;004000;000000;000000;000000;000000; % R REGISTER                  -
530  'D,S1       ;000000;000015;000000;000000;000000;000000;000000; % STATUS REG 1               -
531  'D,S2       ;000000;000015;002000;000000;000000;000000;000000; % STATUS REG 2               -
532  'D,SETLIM   ;000000;000015;010000;000000;000000;000000;000000; % SET LIMIT CONDS FOR HL-LL   -
533  'D,SHC      ;000000;000015;040000;000000;000000;000000;000000; % SHIFT COUNT REGISTER        -
534  'D,TAG      ;000000;000015;106000;000000;000000;000000;000000; % TAG REGISTER               -
535  'D,TE       ;000000;000015;004000;000000;000000;000000;000000; % LOCAL TRAP ENABLE REG      -
536  'D,THA      ;000000;000010;016000;000000;000000;000000;000000; % TRAP HANDLER REG           -
537  'D,TOS      ;000000;000006;016000;000000;000000;000000;000000; % TOP OF STACK               -
538  'D,TRAPCLR  ;000000;000015;016000;000000;000000;000000;000000; % TRAP SYSTEM CLEAR          -
539  'D,TRAPONOFF;000000;000015;006000;000000;000000;000000;000000; % TRAP SYSTEM ON/OFF          -
540  'D,TSBIND   ;000000;000015;014000;000000;000000;000000;000000; % SET AND RESET TSB INDICATOR.
```

ND-500 micro mnemonics in alphabetical and numerical order

```
'D_X#0      ;000000;000002;000000;000000;000000;000000;000000;000000; % INDEX 0 REG
'D_X#1      ;000000;000002;002000;000000;000000;000010;000000;000000; % INDEX 1 REG
'D_X#2      ;000000;000002;004000;000000;000000;000000;000000;000000; % INDEX 2 REG
'D_X#3      ;000000;000002;006000;000000;000000;000000;000000;000000; % INDEX 3 REG
'D_XSTI     ;000000;000015;024000;000000;000000;000000;000000;000000; % S1, BITS 17-19,21-24, 30-31
'DOUBLE     ;000000;000000;000000;000000;000000;000400;000000;000000; % DOUBLE FLOATING
'EA1INH     ;000000;000000;000000;000000;000000;040000;000000;000000; % EFF. ADDR. 1 REG CLOCK INH..
'EA2INH     ;000000;000000;000000;000000;000000;100000;000000;000000; % EFF. ADDR. 2 REG CLOCK INH..
'EX_APIMUL  ;000000;000000;000000;000000;000000;002037;000000;000000; % LAST FL. DIV. STEP SA*SP(IN)
'EX_APIMULA ;000000;000000;000000;000000;000000;002036;000000;000000; % THIRD FL DIV STEP SA*SP(INV)
'EX_APMULA  ;000000;000000;000000;000000;000000;002034;000000;000000; % SA*SP TO SA
'EX_ARMULA  ;000000;000000;000000;000000;000000;002026;000000;000000; % FRST FL DIV STEP A*1/B TO SA
'EX_ASUM    ;000000;000000;000000;000000;000000;002022;000000;000000; % FL ADD SA+B TO CPU
'EX_ASUMA   ;000000;000000;000000;000000;000000;002023;000000;000000; % FL ADD SA+B TO SA
'EX_BRMULP  ;000000;000000;000000;000000;000000;002033;000000;000000; % SEC FL DIV STEP B*1/B TO SP.
'EX_COMPARE ;000000;000000;000000;000000;000000;002025;000000;000000; % COMPARE A AND B
'EX_CTF     ;000000;000000;000000;000000;000000;002000;000000;000000; % CONVERT TO FLOATING
'EX_CTIR    ;000000;000000;000000;000000;000000;002005;000000;000000; % CONVERT TO INTEGER TRUNCATED
'EX_DIFF    ;000000;000000;000000;000000;000000;002004;000000;000000; % CNVRT TO INTEGER WITH ROUND.
'EX_DTOFR   ;000000;000000;000000;000000;000000;002024;000000;000000; % FL SUBT. A-B TO CPU
'EX_INT     ;000000;000000;000000;000000;000000;002017;000000;000000; % CNVRT DBL FL TO SGL, ROUNDED
'EX_INTR    ;000000;000000;000000;000000;000000;002007;000000;000000; % INTEGER PART TRUNCATED
'EX_MUL     ;000000;000000;000000;000000;000000;002006;000000;000000; % INTEGER PART ROUNDED
'EX_MULA    ;000000;000000;000000;000000;000000;002030;000000;000000; % MULTIPLY A*B TO CPU
'EX_PPIMULP ;000000;000000;000000;000000;000000;002032;000000;000000; % MULT AND SAVE A*B TO SA
'EX_SHA     ;000000;000000;000000;000000;000000;002035;000000;000000; % 4. FL DIV STEP SP*SP(INV)
'EX_SHL     ;000000;000000;000000;000000;000000;002010;000000;000000; % SHIFT ARITHMETIC
'EX_SHR     ;000000;000000;000000;000000;000000;002012;000000;000000; % SHIFT LOGICAL
'EX_SUM     ;000000;000000;000000;000000;000000;002013;000000;000000; % SHIFT ROTATIONAL
'EX_TORMULA ;000000;000000;000000;000000;000000;002021;000000;000000; % FL ADD A+B TO CPU
'EX_UCTF    ;000000;000000;000000;000000;000000;002027;000000;000000; % A * 1/A TO SA
'EX_UMUL    ;000000;000000;000000;000000;000000;002001;000000;000000; % UNSIGNED CNVRT TO FLOATING
'F_A+1      ;000000;000000;000000;000000;000000;002031;000000;000000; % UNSIGN MULT A*B TO CPU
'F_A+A      ;000000;000000;000000;000000;000000;000417;000000;000000; % INCREMENT AOPR IF FALSE
'F_A+A+1    ;000000;000000;000000;000000;000000;000014;000000;000000; % AOPR + AOPR IF FALSE
'F_A+B      ;000000;000000;000000;000000;000000;000414;000000;000000; % AOPR + AOPR +1 IF FALSE
'F_A+B+1    ;000000;000000;000000;000000;000000;000011;000000;000000; % AOPR + BOPR IF FALSE
'F_A+B+C    ;000000;000000;000000;000000;000000;000411;000000;000000; % AOPR + BOPR + 1 IF FALSE
'F_A-1      ;000000;000000;000000;000000;000000;001011;000000;000000; % AOPR + BOPR + CARRY IF FALSE
'F_A-B      ;000000;000000;000000;000000;000000;000406;000000;000000; % DECREMENT AOPR IF FALSE
'F_A-B-1    ;000000;000000;000000;000000;000000;000406;000000;000000; % AOPR - BOPR IF FALSE
'F_A-B-1+C  ;000000;000000;000000;000000;000000;000006;000000;000000; % AOPR - BOPR - 1 IF FALSE
'F_ADIR     ;000000;000000;000000;000000;000000;001006;000000;000000; % AOPR - BOPR - 1 + CRY IF FLS
'F_ADIRC    ;000000;000000;000000;000000;000000;000037;000000;000000; % AOPR DIRECT IF FALSE
'F_AND      ;000000;000000;000000;000000;000000;000020;000000;000000; % AOPR COMPLEMENTED IF FALSE
'F_ANDCA    ;000000;000000;000000;000000;000000;000036;000000;000000; % AOPR AND BOPR IF FALSE
'F_ANDCB    ;000000;000000;000000;000000;000000;000030;000000;000000; % COMPL(AOPR) AND BOPR IF FLS.
'F_EDIR     ;000000;000000;000000;000000;000000;000035;000000;000000; % AOPR AND COMPL(BOPR) IF FLS.
'F_EDIRC    ;000000;000000;000000;000000;000000;000032;000000;000000; % BOPR DIRECT IF FALSE
'F_FONE     ;000000;000000;000000;000000;000000;000025;000000;000000; % BOPR COMPLEMENTED IF FALSE
'F_FZRO     ;000000;000000;000000;000000;000000;000023;000000;000000; % FORCED ONE IF FALSE
'F_JMP      ;000000;000000;000000;000000;000000;000034;000000;000000; % FORCED ZERO IF FALSE
'F_JMPCAR   ;000000;000000;000000;001434;000000;000000;000000;000000; % JUMP ABSOLUTE IF FALSE
'F_JMPMAP   ;000000;000000;000000;001234;000000;000000;000000;000000; % JUMP CAR IF FALSE
'F_JMPNS    ;000000;000000;000000;001034;000000;000000;000000;000000; % JUMP MAP IF FALSE
'F_JMPREL   ;000000;000000;000000;001404;000000;000000;000000;000000; % JUMP NOT PUSH STACK IF FALSE
'F_JMPSTK   ;000000;000000;000000;001670;000000;000000;000000;000000; % JUMP RELATIVE IF FALSE
'F_JMPWA    ;000000;000000;000000;001734;000000;000000;000000;000000; % JUMP STACK IF FALSE
'F_JSR      ;000000;000000;000000;000634;000000;000000;000000;000000; % JUMP WA IF FALSE
            ;000000;000000;000000;001434;000000;000000;000000;000000; % JUMP TO SUBROUTINE IF FALSE.
```

ND-500 micro mnemonics in alphabetical and numerical order

```
601  'F_JSRCAR  ' ;000000;000000;000000;001204;000000;000000;000000; % JUMP TO SUBR. IN CAR IF FALS
602  'F_JSRMAP  ' ;000000;000000;000000;001004;000000;000000;000000; % JUMP TO SUBR. IN MAP IF FALS
603  'F_JSREL   ' ;000000;000000;000000;001640;000000;000000;000000; % JUMP TO SUBR. REL IF FALSE .
604  'F_JSRSTK  ' ;000000;000000;000000;001704;000000;000000;000000; % JUMP TO SUBR. IN STACK IF FL
605  'F_JSRWA   ' ;000000;000000;000000;000604;000000;000000;000000; % JUMP TO SUBR. IN WA IF FALSE
606  'F_LOAD    ' ;000000;000000;000000;000310;000000;000000;000000; % LOAD STACK IF FALSE .
607  'F_NAND    ' ;000000;000000;000000;000000;000000;000021;000000; % AOPR NAND BOPR IF FALSE .
608  'F_NEXI    ' ;000000;000000;000000;001674;000000;000000;000000; % NEXT MICRO INSTR. IF FALSE .
609  'F_NEXINS  ' ;000000;000000;000000;001644;000000;000000;000000; % NEXT WITH SEP ST CTRL IF FLS
610  'F_NOPOPRET' ;000000;000000;000000;001734;000000;000000;000000; % RETURN WITHOUT POP IF FALSE.
611  'F_NOR     ' ;000000;000000;000000;000000;000024;000000;000000; % AOPR NOR BOPR IF FALSE .
612  'F_OR      ' ;000000;000000;000000;000000;000033;000000;000000; % AOPR OR BOPR IF FALSE .
613  'F_ORCA    ' ;000000;000000;000000;000000;000022;000000;000000; % C(AOPR) OR BOPR IF FALSE .
614  'F_ORCB    ' ;000000;000000;000000;000000;000027;000000;000000; % AOPR OR C(BOPR) IF FALSE .
615  'F_POP     ' ;000000;000000;000000;000020;000000;000000;000000; % POP STACK IF FALSE .
616  'F_POPRET  ' ;000000;000000;000000;001724;000000;000000;000000; % RETURN FROM SUBR. IF FALSE .
617  'F_PUSH    ' ;000000;000000;000000;000000;000000;000000;000000; % PUSH STACK IF FALSE .
618  'F_REP     ' ;000000;000000;000000;001774;000000;000000;000000; % REPEAT CURR. INSTR. IF FALSE
619  'F_XNOR    ' ;000000;000000;000000;000000;000026;000000;000000; % AOPR EXNOR BOPR IF FALSE .
620  'F_XOR     ' ;000000;000000;000000;000000;000031;000000;000000; % AOPR OR BOPR IF FALSE .
621  'FAST      ' ;000000;000000;000000;000000;002000;000000;000000; % FAST CYCLE .
622  'HBRET     ' ;000000;000000;000004;070000;000000;000000;000000; % RETURN TO HARDW. BRANCH REG.
623  'HWTW      ' ;000300;000000;000000;000000;000000;000000;000000; % HALFW. TO WORD SIGN EXT.
624  'IADJ      ' ;000000;000000;000000;000000;000020;000000;000000; % INDEX ADJUST (INCR IND C.)
625  'IFMEM     ' ;000000;000000;000000;000000;000000;000000;000000; % IF MEMORY, THEN ...
626  'IFT       ' ;000000;000000;000000;000000;000000;000000;000000; % IF CONDITION TRUE, THEN...
627  'IX0       ' ;000000;000000;000000;000000;000000;000000;000000; % INDEX REGISTER 0
628  'IX1       ' ;000000;000020;000000;000000;000000;000000;000000; % INDEX REGISTER 1
629  'IX2       ' ;000000;000040;000000;000000;000000;000000;000000; % INDEX REGISTER 2
630  'IX3       ' ;000000;000060;000000;000000;000000;000000;000000; % INDEX REGISTER 3
631  'JMP       ' ;000000;000000;000014;070000;000000;000000;000000; % JUMP ABSOLUTE
632  'JMPCAR    ' ;000000;000000;000012;070000;000000;000000;000000; % JUMP TO COMPUTED ADDR. REG .
633  'JMPMAP    ' ;000000;000000;000010;070000;000000;000000;000000; % JUMP TO MAP ADDRESS .
634  'JMPNS     ' ;000000;000000;000014;010000;000000;000000;000000; % JUMP WITH SEP STACK CONTROL.
635  'JMPREL    ' ;000000;000000;000016;160000;000000;000000;000000; % JUMP REL TO COMP ADDR REG .
636  'JMPSTK    ' ;000000;000000;000017;070000;000000;000000;000000; % JUMP TO STACK ADDRESS .
637  'JMPWA     ' ;000000;000000;000006;070000;000000;000000;000000; % JUMP TO ADDRESS IN WA .
638  'JSR       ' ;000000;000000;000014;010000;000000;000000;000000; % JUMP TO SUBROUTINE ABSOLUTE.
639  'JSRCAR    ' ;000000;000000;000012;010000;000000;000000;000000; % JUMP TO SUBR IN COMP ADDR R.
640  'JSRMAP    ' ;000000;000000;000010;010000;000000;000000;000000; % JUMP TO SUBR IN MAP ADDRESS.
641  'JSRREL    ' ;000000;000000;000016;100000;000000;000000;000000; % JUMP TO SUBR REL TO COMP ADR
642  'JSRSTK    ' ;000000;000000;000017;010000;000000;000000;000000; % JUMP TO SUBR IN STACK .
643  'JSRWA     ' ;000000;000000;000006;010000;000000;000000;000000; % JUMP TO SUBR IN WA .
644  'K_1IFZ    ' ;000000;000000;000000;000003;000000;000000;000000; % SET K IF ZERO ALU .
645  'K_ONE     ' ;000000;000000;000000;000001;000000;000000;000000; % SET K .
646  'K_ZRO     ' ;000000;000000;000000;000002;000000;000000;000000; % CLEAR K .
647  'LCDECR    ' ;000000;000000;000010;000040;000000;000000;000000; % LOOP COUNTER DECREMENT .
648  'LOAD      ' ;000000;000000;000000;020000;000000;000000;000000; % LOAD STACK .
649  'MEM_MV1   ' ;000000;000005;000000;000000;000000;000000;000000; % MOVE 1 WORD IN DATA MEMORY .
650  'MEM_MV2   ' ;000000;000005;020000;000000;000000;000000;000000; % MOVE 2 WORDS IN DATA MEMORY.
651  'MEM_MV3   ' ;000000;000005;040000;000000;000000;000000;000000; % MOVE 3 WORDS IN DATA MEMORY.
652  'MEM_MV4   ' ;000000;000005;060000;000000;000000;000000;000000; % MOVE 4 WORDS IN DATA MEMORY.
653  'MEM_RD    ' ;000000;000104;000000;000000;000000;000000;000000; % READ MEMORY WITH ORED TYPE .
654  'MEM_RD1   ' ;000000;000004;000000;000000;000000;000000;000000; % READ 1 BYTE FROM DATA MEMORY
655  'MEM_RD2   ' ;000000;000004;020000;000000;000000;000000;000000; % READ 2 BYTES FROM DATAMEMORY
656  'MEM_RD3   ' ;000000;000004;040000;000000;000000;000000;000000; % READ 3 BYTES FROM DATAMEMORY
657  'MEM_RD4   ' ;000000;000004;060000;000000;000000;000000;000000; % READ 4 BYTES FROM DATAMEMORY
658  'MEM_WR    ' ;000000;000104;100000;000000;000000;000000;000000; % WRITE MEMORY WITH ORED TYPE.
659  'MEM_WR1   ' ;000000;000004;100000;000000;000000;000000;000000; % WRITE 1 BYTE TO DATA MEMORY.
660  'MEM_WR2   ' ;000000;000004;120000;000000;000000;000000;000000; % WRITE 2 BYTES TO DATA MEMORY
```

ND-500 micro mnemonics in alphabetical and numerical order

```
661  'MEM,WR3      ;000000;000000;000004;140000;000000;000000;000000; % WRITE 3 BYTES TO DATA MEMORY
662  'MEM,WR4      ;000000;000000;000004;160000;000000;000000;000000; % WRITE 4 BYTES TO DATA MEMORY
663  'MV,DINTOD    ;000000;000000;000000;000000;000000;000300;000000; % DATA-IN TO D
664  'MV,DTODP     ;000000;000000;000000;000000;000000;000200;000000; % ADDR SAVE FR MEM TO DP1-DP2.
665  'MV,DTOXBL    ;000000;000000;000000;000000;000000;000100;000000; % MEM DBUS TO FL BBUS LEAST SG
666  'MV,DTOXBM    ;000000;000000;000000;000000;000000;000000;000000; % MEM DBUS TO FL BBUS MOST SIG
667  'NADL         ;000000;000000;000000;001000;000000;000000;000000; % NOT ADDR LATCH.
668  'NEXT         ;000000;000000;000000;170000;000000;000000;000000; % NEXT MICRO INSTRUCTION
669  'NEXTNS       ;000000;000000;000016;110000;000000;000000;000000; % NEXT WITH SEP STACK CONTROL.
670  'NOPOPREF     ;000000;000000;000017;070000;000000;000000;000000; % RETURN WITHOUT POP
671  'NPCSEL       ;000000;000000;000000;010000;000000;000000;000000; % SELECT NPC TO L REGISTER
672  'NSEXCONV     ;000000;000200;000000;000000;000000;000000;000000; % NO SIGN EXT. WITH B,XRESL
673  'ORA          ;000000;001000;000000;000000;000000;000000;000000; % OR AOPR ACC. TO INSTRUCTION.
674  'ORB          ;000000;000400;000000;000000;000000;000000;000000; % OR BOPR ACC. TO INSTRUCTION.
675  'ORD          ;000000;002000;000000;000000;000000;000000;000000; % OR DEST ACC. TO INSTRUCTION.
676  'ORT          ;000000;000040;000000;000000;000000;000000;000000; % OR DATATYPE ACC. TO INSTR.
677  'PASSAA       ;000000;000000;000000;000010;000000;000000;000000; % PASS ADDR AOPR THR ADDR ARIT
678  'PASSAB       ;000000;000000;000000;000016;000000;000000;000000; % PASS ADDR BOPR THR ADDR ARIT
679  'POP          ;000000;000000;000000;040000;000000;000000;000000; % POP STACK
680  'POPRET       ;000000;000000;000017;050000;000000;000000;000000; % RETURN FROM SUBROUTINE
681  'PRF,ACONT    ;000000;000000;000000;100000;000000;000000;000000; % ADDR ARITHMETIC CONTINUE
682  'PRF,CEOPF    ;000000;000000;000000;120000;000000;000000;000000; % END-OF-OP IF FALSE
683  'PRF,CEOPT    ;000000;000000;000000;020000;000000;000000;000000; % END-OF-OP IF TRUE
684  'PRF,CLEAR    ;000000;000000;000000;110000;000000;000000;000000; % PREFETCH CLEAR
685  'PRF,EOP      ;000000;000000;000000;010000;000000;000000;000000; % END-OF-OPERATION
686  'PRF,FADC     ;000000;000000;000000;040000;000000;000000;000000; % FETCH ADDR CODE
687  'PRF,FARG     ;000000;000000;000000;050000;000000;000000;000000; % FETCH ARGUMENT
688  'PRF,FOPC     ;000000;000000;000000;030000;000000;000000;000000; % FETCH OPCODE
689  'PRF,FOPR     ;000000;000000;000000;150000;000000;000000;000000; % FETCH OPERAND
690  'PRF,ISAMP    ;000000;000000;000000;170000;000000;000000;000000; % INTERRUPT SAMPLE
691  'PRF,PCONT    ;000000;000000;000000;070000;000000;000000;000000; % PREFETCH CONTINUE
692  'PRF,START    ;000000;000000;000000;060000;000000;000000;000000; % START
693  'PRF,WFIN     ;000000;000000;000000;160000;000000;000000;000000; % WAIT FOR PREFEICH FINISHED
694  'PUSH         ;000000;000000;000000;000000;000000;000000;000000; % PUSH (SEQUENCER) STACK
695  'REP          ;000000;000000;000017;170000;000000;000000;000000; % REPEAT CURRENT MICRO INSTR.
696  'SCRB         ;000100;000000;000000;000000;000000;000000;000000; % USE BLOCK B AS A-OPERAND
697  'SET          ;000000;000000;000000;004000;000000;000000;000000; % SET CONDITION SELECT
698  'SHARG        ;000000;000000;000000;000000;000000;000000;000000; % SHIFT COUNTER ARGUMENT
699  'SHCFR        ;000010;000000;000000;000000;000000;000000;000000; % SHIFT COUNT FROM SH COUNT RG
700  'SINGLE       ;000000;000000;000000;000000;000000;000000;000000; % SINGLE FLOATING
701  'SLOW1        ;000000;000000;000000;000000;004000;000000;000000; % SLOW CYCLE
702  'SLOW2        ;000000;000000;000000;000000;006000;000000;000000; % FIRST HALF CYCLE IS SLOW
703  'SPAREBIT     ;040000;000000;000000;000000;000000;000000;000000; % SPARE BIT
704  'ST,SAVA      ;000000;000000;000000;000100;000000;000000;000000; % SAVE ALU STATUS
705  'ST,SAVAC     ;000000;000000;000000;000001;000100;000000;000000; % SAVE COMPARE STATUS
706  'ST,SAVB      ;000000;000000;000000;000003;000100;000000;000000; % SAVE BCD STATUS
707  'ST,SAVF      ;000000;000000;000000;000002;000100;000000;000000; % SAVE FLOATING STATUS
708  'TYP,BY       ;000000;000000;000030;000000;000000;000000;000000; % DATATYPE IS BYTE
709  'TYP,F        ;000000;000010;000000;000000;000000;000000;000000; % DATATYPE IS FLOATING
710  'TYP,HW       ;000000;000020;000000;000000;000000;000000;000000; % DATATYPE IS HALFWORD
711  'TYP,W        ;000000;000000;000000;000000;000000;000000;000000; % DATATYPE IS WORD
712  'UNLOCK       ;000015;132000;000000;000000;000000;000000;000000; % UNLOCK INTERFACE TO N100
713  'W,EXT        ;000000;000000;000000;000000;000600;000000;000000; % WAIT FOR EXT. ARITHM. UNIT
714  'W,IO         ;000000;000000;000000;000000;000400;000000;000000; % WAIT FOR INPUT/OUTPUT
715  'W,IOEM       ;000000;000000;000000;000000;001600;000000;000000; % WAIT FOR IO/EXT/MEM
716  'W,IOEPM      ;000000;000000;000000;000000;001200;000000;000000; % WAIT FOR IO/EXT/PMEM
717  'W,MEM        ;000000;000000;000000;000000;001400;000000;000000; % WAIT FOR MEMORY
718  'W,PMEM       ;000000;000000;000000;000000;001000;000000;000000; % WAIT FOR PREVIOUS MEM CYCLE.
719  'W,XD         ;000000;000000;000000;000000;002000;000000;000000; % WAIT FOR XD-BUS
720  'XDMOV        ;000000;000002;000000;000000;000000;000000;000000; % XD-BUS MOVE (NO ALU TRANSF).
```

ND-500 micro mnemonics in alphabetical and numerical order

```
721  'XRES    ';000000;000000;000000;000000;000000;000000;000040;000000;000000; % ENABLE EXT RESULT INTO CPU .
722
```

ND-500 micro mnemonics in alphabetical and numerical order

```
'SPAREBIT      ;040000;000000;000000;000000;000000;000000; % SPARE BIT
'ALU,ADIR      ;037000;000000;000000;000000;000000;000000; % AOPR DIRECT THROUGH ALU
'ALU,AND       ;036000;000000;000000;000000;000000;000000; % AOPR AND BOPR
'ALU,ANDCM     ;035000;000000;000000;000000;000000;000000; % AOPR AND COMPL(BOPR)
'ALU,FZRO      ;034000;000000;000000;000000;000000;000000; % FORCED ZRO
'ALU,OR        ;033000;000000;000000;000000;000000;000000; % AOPR OR BOPR
'ALU,BDIR      ;032000;000000;000000;000000;000000;000000; % BOPR DIRECT THROUGH ALU
'ALU,XOR       ;031000;000000;000000;000000;000000;000000; % AOPR EXOR BOPR
'ALU,ANDCA     ;030000;000000;000000;000000;000000;000000; % COMPL(AOPR) AND BOPR
'ALU,ORCM      ;027000;000000;000000;000000;000000;000000; % AOPR OR COMPL(BOPR)
'ALU,XNOR      ;026000;000000;000000;000000;000000;000000; % AOPR EXNOR BOPR
'ALU,BDIRC     ;025000;000000;000000;000000;000000;000000; % BOPR COMPLEMENTED THR. ALU
'ALU,NOR       ;024000;000000;000000;000000;000000;000000; % AOPR NOR BOPR
'ALU,FONE      ;023000;000000;000000;000000;000000;000000; % FORCED ONE
'ALU,ORCA      ;022000;000000;000000;000000;000000;000000; % COMPL(AOPR) OR BOPR
'ALU,NAND      ;021000;000000;000000;000000;000000;000000; % AOPR NAND BOPR
'ALU,ADIRC     ;020000;000000;000000;000000;000000;000000; % AOPR COMPLEMENTED THR. ALU
'ALU,A+1       ;017200;000000;000000;000000;000000;000000; % INCREMENT AOPR
'ALU,A+A+1     ;014200;000000;000000;000000;000000;000000; % AOPR + AOPR + 1
'ALU,A+A       ;014000;000000;000000;000000;000000;000000; % AOPR + AOPR
'ALU,A+B+C     ;011400;000000;000000;000000;000000;000000; % AOPR + BOPR + CARRY
'ALU,A+B+1     ;011200;000000;000000;000000;000000;000000; % AOPR + BOPR + 1
'ALU,A+B       ;011000;000000;000000;000000;000000;000000; % AOPR + BOPR
'ALU,A-B-1+C   ;006400;000000;000000;000000;000000;000000; % AOPR - BOPR - 1 + CARRY
'ALU,A-B       ;006200;000000;000000;000000;000000;000000; % AOPR - BOPR
'ALU,A-B-1     ;006000;000000;000000;000000;000000;000000; % AOPR - BOPR - 1
'CRY,MISTAT    ;000600;000000;000000;000000;000000;000000; % CARRY FROM MICRO STATUS
'CRY,STAT      ;000400;000000;000000;000000;000000;000000; % CARRY FROM STATUS
'CRY,ONE       ;000200;000000;000000;000000;000000;000000; % CARRY IS FORCED ONE
'A,SP          ;000160;120000;000000;000000;000000;000000; % SAVED P
'A,R           ;000160;060000;000000;000000;000000;000000; % R REGISTER
'A,B           ;000160;040000;000000;000000;000000;000000; % B REGISTER
'A,P           ;000160;020000;000000;000000;000000;000000; % P REGISTER
'A,L           ;000160;000000;000000;000000;000000;000000; % L REGISTER
'A,X#3         ;000140;060000;000000;000000;000000;000000; % X REG 3
'A,X#2         ;000140;040000;000000;000000;000000;000000; % X REG 2
'A,X#1         ;000140;020000;000000;000000;000000;000000; % X REG 1
'A,X#0         ;000140;000000;000000;000000;000000;000000; % X REG 0
'SCRB          ;000100;000000;000000;000000;000000;000000; % USE BLOCK B AS A-OPERAND
'A,XD,DISP     ;000076;060000;000000;000000;000000;000000; % DISPLACEMENT REG
'A,XD,LC       ;000076;040000;000000;000000;000000;000000; % LOOP COUNTER
'A,XD,CONST    ;000076;020000;000000;000000;000000;000000; % INSTRUCTION CONSTANT
'A,XD,IDAT     ;000076;000000;000000;000000;000000;000000; % INSTRUCTION MEMORY DATA
'A,XD,IODOUT   ;000074;120000;000000;000000;000000;000000; % DATA-OUT REG
'A,XD,CSCNT    ;000074;060000;000000;000000;000000;000000; % CONTROL STORE CONTROL REG
'A,XD,CSBRK    ;000074;040000;000000;000000;000000;000000; % CONTROL STORE BREAK REG
'A,XD,CSWA     ;000074;020000;000000;000000;000000;000000; % CONTR. ST. WRITE ADDR REG
'A,XD,IODIN    ;000074;000000;000000;000000;000000;000000; % DATA-IN REG
'A,XD,INDXC#3  ;000073;060000;000000;000000;000000;000000; % INDEX COUNTER 3
'A,XD,INDXC#2  ;000073;040000;000000;000000;000000;000000; % INDEX COUNTER 2
'A,XD,INDXC#1  ;000073;020000;000000;000000;000000;000000; % INDEX COUNTER 1
'A,XD,INDXC#0  ;000073;000000;000000;000000;000000;000000; % INDEX COUNTER 0
'A,XD,PSTAT    ;000072;120000;000000;000000;000000;000000; % PREFETCH STATUS
'A,XD,BMR      ;000072;100000;000000;000000;000000;000000; % BIT MASK REGISTER
'A,XD,SHC      ;000072;060000;000000;000000;000000;000000; % SHIFT COUNT REGISTER
'A,XD,CAR      ;000072;040000;000000;000000;000000;000000; % COMPUTED ADDRESS REGISTER
'A,XD,LARG     ;000072;020000;000000;000000;000000;000000; % LONG ARGUMENT (32 BITS)
'A,XD,SARG     ;000072;000000;000000;000000;000000;000000; % SHORT ARGUMENT (16 BITS)
'A,XD,MISTAT   ;000070;160000;000000;000000;000000;000000; % MICRO STATUS REGISTER
'A,XD,TRAPINF  ;000070;140000;000000;000000;000000;000000; % TRAP INFORMATION
```

ND-500 micro mnemonics in alphabetical and numerical order

```
'A_XD_NBY     ';000070;120000;000000;000000;000000;000000;000000; % NO OF BYTES IN LAST MEM REF.
'A_XD_MMOD    ';000070;100000;000000;000000;000000;000000;000000; % MEMORY MODUS REGISTER      .
'AD_PC        ';000070;060000;000000;000000;000000;000000;000000; % DEST IS PROGRAM COUNTER    .
'A_XD_TE      ';000070;040000;000000;000000;000000;000000;000000; % LOCAL TRAP ENABLE REG       .
'A_XD_S2      ';000070;020000;000000;000000;000000;000000;000000; % STATUS REG TWO              .
'A_XD_S1      ';000070;000000;000000;000000;000000;000000;000000; % STATUS REG ONE              .
'AD_ILC       ';000066;140000;000000;000000;000000;000000;000000; % DEST IS INSTR LOOK AHEAD A.C
'AD_NPC       ';000066;040000;000000;000000;000000;000000;000000; % DEST IS NEXT PROGRAM COUNTER
'A_XD_HL      ';000066;020000;000000;000000;000000;000000;000000; % HIGHER LIMIT REGISTER       .
'A_XD_LL      ';000066;000000;000000;000000;000000;000000;000000; % LOWER LIMIT REG             .
'A_XD_DZPA    ';000065;160000;000000;000000;000000;000000;000000; % DATA ZERO POINT ADJUST REG..
'A_XD_DUPL    ';000065;140000;000000;000000;000000;000000;000000; % DATA UPPER PAGE LIMIT       .
'A_XD_DHXA    ';000065;120000;000000;000000;000000;000000;000000; % DMM HASH OR STSB INDEX      .
'A_XD_DRADDRL ';000065;120000;000000;000000;000000;000000;000000; % DATA REAL ADDR LEAST SIGNIF.
'A_XD_DRADDR  ';000065;100000;000000;000000;000000;000000;000000; % DMM REAL ADDRESS (PHYSICAL).
'A_XD_DRADDRM ';000065;100000;000000;000000;000000;000000;000000; % DATA REAL ADDR MOST SIGNIF..
'A_XD_DWIPGU  ';000065;060000;000000;000000;000000;000000;000000; % DMM WIP/PGU BROADSIDE, 16 PG
'A_XD_DLADDR  ';000065;040000;000000;000000;000000;000000;000000; % DMM LOGICAL ADDRESS         .
'A_XD_DCINHLL ';000065;020000;000000;000000;000000;000000;000000; % DATA CACHE INHIBIT LOWER L..
'A_XD_DMSTS   ';000065;020000;000000;000000;000000;000000;000000; % DMM STATUS                  .
'A_XD_DCINHLU ';000065;000000;000000;000000;000000;000000;000000; % DATA CACHE INHIBIT UPPER L..
'A_XD_DSCRF   ';000065;000000;000000;000000;000000;000000;000000; % DMM SCRATCH F.,16 16-BIT RG
'A_XD_DSTSU   ';000064;060000;000000;000000;000000;000000;000000; % DATA MEM. STATUS REG. 0      .
'A_XD_DSTS1   ';000064;040000;000000;000000;000000;000000;000000; % DATA MEM. STATUS REG. 1      .
'A_XD_DSTS2   ';000064;020000;000000;000000;000000;000000;000000; % DATA MEM. STATUS REG. 2      .
'A_XD_IZPA    ';000063;160000;000000;000000;000000;000000;000000; % INSTR ZERO POINT ADJUST REG.
'A_XD_IUPL    ';000063;140000;000000;000000;000000;000000;000000; % INSTR UPPER PAGE LIMIT      .
'A_XD_IHXA    ';000063;120000;000000;000000;000000;000000;000000; % IMM HASH OR STSB INDEX      .
'A_XD_IRADDRL ';000063;120000;000000;000000;000000;000000;000000; % INSTR REAL ADDR LEAST SIGN..
'A_XD_IRADDR  ';000063;100000;000000;000000;000000;000000;000000; % IMM REAL ADDRESS (PHYSICAL).
'A_XD_IRADDRM ';000063;100000;000000;000000;000000;000000;000000; % INSTR REAL ADDR MOST SIGN..
'A_XD_IWIPGU  ';000063;060000;000000;000000;000000;000000;000000; % IMM WIP/PGU BROADSIDE, 16 PG
'A_XD_ILADDR  ';000063;040000;000000;000000;000000;000000;000000; % IMM LOGICAL ADDRESS         .
'A_XD_ICINHLL ';000063;020000;000000;000000;000000;000000;000000; % INSTR CACHE INHIBIT LOWER L.
'A_XD_IMSTS   ';000063;020000;000000;000000;000000;000000;000000; % IMM STATUS                  .
'A_XD_ICINHLU ';000063;000000;000000;000000;000000;000000;000000; % INSTR CACHE INHIBIT UPPER L.
'A_XD_ISCRF   ';000063;000000;000000;000000;000000;000000;000000; % IMM SCRATCH F.,16 16-BIT RG
'A_XD_ISTSU   ';000062;060000;000000;000000;000000;000000;000000; % INSTR MEM. STATUS REG. 0     .
'A_XD_ISTS1   ';000062;040000;000000;000000;000000;000000;000000; % INSTR MEM. STATUS REG. 1     .
'A_XD_ISTS2   ';000062;020000;000000;000000;000000;000000;000000; % INSTR MEM. STATUS REG. 2     .
'A_BMR        ';000044;000000;000000;000000;000000;000000;000000; % DECODED BMR BIT              .
'A_BM#37      ';000043;160000;000000;000000;000000;000000;000000; % BIT MASK BIT 37              .
'A_BM#36      ';000043;140000;000000;000000;000000;000000;000000; % BIT MASK BIT 36              .
'A_BM#35      ';000043;120000;000000;000000;000000;000000;000000; % BIT MASK BIT 35              .
'A_BM#34      ';000043;100000;000000;000000;000000;000000;000000; % BIT MASK BIT 34              .
'A_BM#33      ';000043;060000;000000;000000;000000;000000;000000; % BIT MASK BIT 33              .
'A_BM#32      ';000043;040000;000000;000000;000000;000000;000000; % BIT MASK BIT 32              .
'A_BM#31      ';000043;020000;000000;000000;000000;000000;000000; % BIT MASK BIT 31              .
'A_BM#30      ';000043;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 30              .
'A_BM#27      ';000042;160000;000000;000000;000000;000000;000000; % BIT MASK BIT 27              .
'A_BM#26      ';000042;140000;000000;000000;000000;000000;000000; % BIT MASK BIT 26              .
'A_BM#25      ';000042;120000;000000;000000;000000;000000;000000; % BIT MASK BIT 25              .
'A_BM#24      ';000042;100000;000000;000000;000000;000000;000000; % BIT MASK BIT 24              .
'A_BM#23      ';000042;060000;000000;000000;000000;000000;000000; % BIT MASK BIT 23              .
'A_BM#22      ';000042;040000;000000;000000;000000;000000;000000; % BIT MASK BIT 22              .
'A_BM#21      ';000042;020000;000000;000000;000000;000000;000000; % BIT MASK BIT 21              .
'A_BM#20      ';000042;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 20              .
'A_BM#17      ';000041;160000;000000;000000;000000;000000;000000; % BIT MASK BIT 17              .
'A_BM#16      ';000041;140000;000000;000000;000000;000000;000000; % BIT MASK BIT 16              .
'A_BM#15      ';000041;120000;000000;000000;000000;000000;000000; % BIT MASK BIT 15              .
```

ND-500 micro mnemonics in alphabetical and numerical order

```
'A_BM#14  ;000041;100000;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 14
'A_BM#13  ;000041;060000;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 13
'A_BM#12  ;000041;040000;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 12
'A_BM#11  ;000041;020000;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 11
'A_BM#10  ;000041;010000;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 10
'A_BM#7   ;000040;160000;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 7
'A_BM#6   ;000040;140000;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 6
'A_BM#5   ;000040;120000;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 5
'A_BM#4   ;000040;100000;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 4
'A_BM#3   ;000040;060000;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 3
'A_BM#2   ;000040;040000;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 2
'A_BM#1   ;000040;020000;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 1
'A_BM#0   ;000040;010000;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 0
'A_DATA   ;000020;000000;000000;000000;000000;000000;000000;000000; % MEMORY DATA
'SHCFR    ;000010;000000;000000;000000;000000;000000;000000;000000; % SHIFT COUNT FROM SH COUNT RG
'A_AL#37  ;000007;160000;000000;000000;000000;000000;000000;000000; % A-REG 37 LEAST SIGN.
'A_AL#36  ;000007;140000;000000;000000;000000;000000;000000;000000; % A-REG 36 LEAST SIGN.
'A_AL#35  ;000007;120000;000000;000000;000000;000000;000000;000000; % A-REG 35 LEAST SIGN.
'A_AL#34  ;000007;100000;000000;000000;000000;000000;000000;000000; % A-REG 34 LEAST SIGN.
'A_AL#33  ;000007;060000;000000;000000;000000;000000;000000;000000; % A-REG 33 LEAST SIGN.
'A_AL#32  ;000007;040000;000000;000000;000000;000000;000000;000000; % A-REG 32 LEAST SIGN.
'A_AL#31  ;000007;020000;000000;000000;000000;000000;000000;000000; % A-REG 31 LEAST SIGN.
'A_AL#30  ;000007;000000;000000;000000;000000;000000;000000;000000; % A-REG 30 LEAST SIGN.
'A_AL#27  ;000006;160000;000000;000000;000000;000000;000000;000000; % A-REG 27 LEAST SIGN.
'A_AL#26  ;000006;140000;000000;000000;000000;000000;000000;000000; % A-REG 26 LEAST SIGN.
'A_AL#25  ;000006;120000;000000;000000;000000;000000;000000;000000; % A-REG 25 LEAST SIGN.
'A_AL#24  ;000006;100000;000000;000000;000000;000000;000000;000000; % A-REG 24 LEAST SIGN.
'A_AL#23  ;000006;060000;000000;000000;000000;000000;000000;000000; % A-REG 23 LEAST SIGN.
'A_AL#22  ;000006;040000;000000;000000;000000;000000;000000;000000; % A-REG 22 LEAST SIGN.
'A_AL#21  ;000006;020000;000000;000000;000000;000000;000000;000000; % A-REG 21 LEAST SIGN.
'A_AL#20  ;000006;000000;000000;000000;000000;000000;000000;000000; % A-REG 20 LEAST SIGN.
'A_AL#17  ;000005;160000;000000;000000;000000;000000;000000;000000; % A-REG 17 LEAST SIGN.
'A_AL#16  ;000005;140000;000000;000000;000000;000000;000000;000000; % A-REG 16 LEAST SIGN.
'A_AL#15  ;000005;120000;000000;000000;000000;000000;000000;000000; % A-REG 15 LEAST SIGN.
'A_AL#14  ;000005;100000;000000;000000;000000;000000;000000;000000; % A-REG 14 LEAST SIGN.
'A_AL#13  ;000005;060000;000000;000000;000000;000000;000000;000000; % A-REG 13 LEAST SIGN.
'A_AL#12  ;000005;040000;000000;000000;000000;000000;000000;000000; % A-REG 12 LEAST SIGN.
'A_AL#11  ;000005;020000;000000;000000;000000;000000;000000;000000; % A-REG 11 LEAST SIGN.
'A_AL#10  ;000005;000000;000000;000000;000000;000000;000000;000000; % A-REG 10 LEAST SIGN.
'A_AL#7   ;000004;160000;000000;000000;000000;000000;000000;000000; % A-REG 7 LEAST SIGN.
'A_THA    ;000004;140000;000000;000000;000000;000000;000000;000000; % TRAP HANDLER REGISTER
'A_AL#6   ;000004;120000;000000;000000;000000;000000;000000;000000; % A-REG 6 LEAST SIGN.
'A_AL#5   ;000004;100000;000000;000000;000000;000000;000000;000000; % A-REG 5 LEAST SIGN.
'A_AL#4   ;000004;060000;000000;000000;000000;000000;000000;000000; % A-REG 4 LEAST SIGN.
'A_AL#3   ;000004;040000;000000;000000;000000;000000;000000;000000; % A-REG 3 LEAST SIGN.
'A_AL#2   ;000004;020000;000000;000000;000000;000000;000000;000000; % A-REG 2 LEAST SIGN.
'A_AL#1   ;000004;000000;000000;000000;000000;000000;000000;000000; % A-REG 1 LEAST SIGN.
'A_AL#0   ;000003;160000;000000;000000;000000;000000;000000;000000; % A-REG 0 LEAST SIGN.
'A_AM#37  ;000003;140000;000000;000000;000000;000000;000000;000000; % A-REG 37 MOST SIGN.
'A_AM#36  ;000003;120000;000000;000000;000000;000000;000000;000000; % A-REG 36 MOST SIGN.
'A_AM#35  ;000003;100000;000000;000000;000000;000000;000000;000000; % A-REG 35 MOST SIGN.
'A_AM#34  ;000003;060000;000000;000000;000000;000000;000000;000000; % A-REG 34 MOST SIGN.
'A_AM#33  ;000003;040000;000000;000000;000000;000000;000000;000000; % A-REG 33 MOST SIGN.
'A_AM#32  ;000003;020000;000000;000000;000000;000000;000000;000000; % A-REG 32 MOST SIGN.
'A_AM#31  ;000003;000000;000000;000000;000000;000000;000000;000000; % A-REG 31 MOST SIGN.
'A_AM#30  ;000002;160000;000000;000000;000000;000000;000000;000000; % A-REG 30 MOST SIGN.
'A_AM#27  ;000002;140000;000000;000000;000000;000000;000000;000000; % A-REG 27 MOST SIGN.
'A_AM#26  ;000002;120000;000000;000000;000000;000000;000000;000000; % A-REG 26 MOST SIGN.
'A_AM#25  ;000002;100000;000000;000000;000000;000000;000000;000000; % A-REG 25 MOST SIGN.
'A_AM#24  ;000002;060000;000000;000000;000000;000000;000000;000000; % A-REG 24 MOST SIGN.
```

ND-500 micro mnemonics in alphabetical and numerical order

```
'A,AM#23' ;000002;060000;000000;000000;000000;000000;000000; % A-REG 23 MOST SIGN.
'A,AM#22' ;000002;040000;000000;000000;000000;000000;000000; % A-REG 22 MOST SIGN.
'A,AM#21' ;000002;020000;000000;000000;000000;000000;000000; % A-REG 21 MOST SIGN.
'A,AM#20' ;000002;000000;000000;000000;000000;000000;000000; % A-REG 20 MOST SIGN.
'A,AM#17' ;000001;160000;000000;000000;000000;000000;000000; % A-REG 17 MOST SIGN.
'A,AM#16' ;000001;140000;000000;000000;000000;000000;000000; % A-REG 16 MOST SIGN.
'A,AM#15' ;000001;120000;000000;000000;000000;000000;000000; % A-REG 15 MOST SIGN.
'A,AM#14' ;000001;100000;000000;000000;000000;000000;000000; % A-REG 14 MOST SIGN.
'A,AM#13' ;000001;060000;000000;000000;000000;000000;000000; % A-REG 13 MOST SIGN.
'A,AM#12' ;000001;040000;000000;000000;000000;000000;000000; % A-REG 12 MOST SIGN.
'A,AM#11' ;000001;020000;000000;000000;000000;000000;000000; % A-REG 11 MOST SIGN.
'A,AM#10' ;000001;000000;000000;000000;000000;000000;000000; % A-REG 10 MOST SIGN.
'A,AM#7'  ;000000;160000;000000;000000;000000;000000;000000; % A-REG 7 MOST SIGN.
'A,TOS'   ;000000;140000;000000;000000;000000;000000;000000; % TOP OF STACK
'A,AM#6'  ;000000;140000;000000;000000;000000;000000;000000; % A-REG 6 MOST SIGN.
'A,ONE'   ;000000;140000;000000;000000;000000;000000;000000; % FLOATING 1.0
'A,AM#5'  ;000000;120000;000000;000000;000000;000000;000000; % A-REG 5 MOST SIGN.
'A,ZRO'   ;000000;120000;000000;000000;000000;000000;000000; % FLOATING 0.0
'A,-1'    ;000000;100000;000000;000000;000000;000000;000000; % FLOATING -1.0
'A,AM#4'  ;000000;100000;000000;000000;000000;000000;000000; % A-REG 4 MOST SIGN.
'A,AM#3'  ;000000;060000;000000;000000;000000;000000;000000; % DOUBLE FLOAT REG 3
'A,AM#3'  ;000000;060000;000000;000000;000000;000000;000000; % A-REG 3 MOST SIGN.
'A,AM#2'  ;000000;040000;000000;000000;000000;000000;000000; % DOUBLE FLOAT REG 2
'A,AM#2'  ;000000;040000;000000;000000;000000;000000;000000; % A-REG 2 MOST SIGN.
'A,AM#1'  ;000000;020000;000000;000000;000000;000000;000000; % DOUBLE FLOAT REG 1
'A,AM#1'  ;000000;020000;000000;000000;000000;000000;000000; % A-REG 1 MOST SIGN.
'E,XRESL' ;000000;016100;000000;000000;000040;000000;000000; % EXT. RESULT LEAST SIGN.
'B,XRESM' ;000000;016000;000000;000000;000040;000000;000000; % EXT. RESULT MOST SIGN.
'E,X#3'   ;000000;012060;000000;000000;000000;000000;000000; % INDEX REG 3
'E,X#2'   ;000000;012040;000000;000000;000000;000000;000000; % INDEX REG 2
'B,X#1'   ;000000;012020;000000;000000;000000;000000;000000; % INDEX REG 1
'B,X#0'   ;000000;012000;000000;000000;000000;000000;000000; % INDEX REG 0
'F,AL#37' ;000000;011760;000000;000000;000000;000000;000000; % A-REG 37 LEAST SIGN.
'B,AL#36' ;000000;011740;000000;000000;000000;000000;000000; % A-REG 36 LEAST SIGN.
'B,AL#35' ;000000;011720;000000;000000;000000;000000;000000; % A-REG 35 LEAST SIGN.
'E,AL#34' ;000000;011700;000000;000000;000000;000000;000000; % A-REG 34 LEAST SIGN.
'B,AL#33' ;000000;011660;000000;000000;000000;000000;000000; % A-REG 33 LEAST SIGN.
'B,AL#32' ;000000;011640;000000;000000;000000;000000;000000; % A-REG 32 LEAST SIGN.
'B,AL#31' ;000000;011620;000000;000000;000000;000000;000000; % A-REG 31 LEAST SIGN.
'B,AL#30' ;000000;011600;000000;000000;000000;000000;000000; % A-REG 30 LEAST SIGN.
'B,AL#27' ;000000;011560;000000;000000;000000;000000;000000; % A-REG 27 LEAST SIGN.
'E,AL#26' ;000000;011540;000000;000000;000000;000000;000000; % A-REG 26 LEAST SIGN.
'B,AL#25' ;000000;011520;000000;000000;000000;000000;000000; % A-REG 25 LEAST SIGN.
'F,AL#24' ;000000;011500;000000;000000;000000;000000;000000; % A-REG 24 LEAST SIGN.
'B,AL#23' ;000000;011460;000000;000000;000000;000000;000000; % A-REG 23 LEAST SIGN.
'B,AL#22' ;000000;011440;000000;000000;000000;000000;000000; % A-REG 22 LEAST SIGN.
'B,AL#21' ;000000;011420;000000;000000;000000;000000;000000; % A-REG 21 LEAST SIGN.
'B,AL#20' ;000000;011400;000000;000000;000000;000000;000000; % A-REG 20 LEAST SIGN.
'B,AL#17' ;000000;011360;000000;000000;000000;000000;000000; % A-REG 17 LEAST SIGN.
'B,AL#16' ;000000;011340;000000;000000;000000;000000;000000; % A-REG 16 LEAST SIGN.
'B,AL#15' ;000000;011320;000000;000000;000000;000000;000000; % A-REG 15 LEAST SIGN.
'E,AL#14' ;000000;011300;000000;000000;000000;000000;000000; % A-REG 14 LEAST SIGN.
'B,AL#13' ;000000;011260;000000;000000;000000;000000;000000; % A-REG 13 LEAST SIGN.
'B,AL#12' ;000000;011240;000000;000000;000000;000000;000000; % A-REG 12 LEAST SIGN.
'B,AL#11' ;000000;011220;000000;000000;000000;000000;000000; % A-REG 11 LEAST SIGN.
'B,AL#10' ;000000;011200;000000;000000;000000;000000;000000; % A-REG 10 LEAST SIGN.
'B,AL#7'  ;000000;011160;000000;000000;000000;000000;000000; % A-REG 7 LEAST SIGN.
'B,THA'   ;000000;011160;000000;000000;000000;000000;000000; % TRAP HANDLER EG
'B,AL#6'  ;000000;011140;000000;000000;000000;000000;000000; % A-REG 6 LEAST SIGN.
'E,AL#5'  ;000000;011120;000000;000000;000000;000000;000000; % A-REG 5 LEAST SIGN.
```

ND-500 micro mnemonics in alphabetical and numerical order

```
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
```

```
'B,AL#4
'B,AL#3
'B,AL#2
'B,AL#1
'B,AL#0
'B,AM#37
'B,AM#36
'B,AM#35
'B,AM#34
'B,AM#33
'B,AM#32
'B,AM#31
'B,AM#30
'B,AM#27
'B,AM#26
'B,AM#25
'B,AM#24
'B,AM#23
'B,AM#22
'B,AM#21
'B,AM#20
'B,AM#17
'B,AM#16
'B,AM#15
'B,AM#14
'B,AM#13
'B,AM#12
'B,AM#11
'B,AM#10
'B,AM#7
'B,TOS
'B,AM#6
'B,ONE
'B,AM#5
'B,ZRO
'B,-1
'B,AM#4
'B,AM#3
'B,AM#3
'B,AM#2
'B,AM#2
'B,AM#1
'B,AM#1
'B,AM#0
'B,AM#0
'B,EA2
'B,EA1
'B,BMR
'B,BM#37
'B,BM#36
'B,BM#35
'B,BM#34
'B,BM#33
'B,BM#32
'B,BM#31
'B,BM#30
'B,BM#27
'B,BM#26
'B,BM#25
'B,BM#24
```

```
;000000;011100;000000;000000;000000;000000; % A-REG 4 LEAST SIGN.
;000000;011060;000000;000000;000000;000000; % A-REG 3 LEAST SIGN.
;000000;011040;000000;000000;000000;000000; % A-REG 2 LEAST SIGN.
;000000;011020;000000;000000;000000;000000; % A-REG 1 LEAST SIGN.
;000000;011000;000000;000000;000000;000000; % A-REG 0 LEAST SIGN.
;000000;010760;000000;000000;000000;000000; % A-REG 37 MOST SIGN.
;000000;010740;000000;000000;000000;000000; % A-REG 36 MOST SIGN.
;000000;010720;000000;000000;000000;000000; % A-REG 35 MOST SIGN.
;000000;010700;000000;000000;000000;000000; % A-REG 34 MOST SIGN.
;000000;010660;000000;000000;000000;000000; % A-REG 33 MOST SIGN.
;000000;010640;000000;000000;000000;000000; % A-REG 32 MOST SIGN.
;000000;010620;000000;000000;000000;000000; % A-REG 31 MOST SIGN.
;000000;010600;000000;000000;000000;000000; % A-REG 30 MOST SIGN.
;000000;010560;000000;000000;000000;000000; % A-REG 27 MOST SIGN.
;000000;010540;000000;000000;000000;000000; % A-REG 26 MOST SIGN.
;000000;010520;000000;000000;000000;000000; % A-REG 25 MOST SIGN.
;000000;010500;000000;000000;000000;000000; % A-REG 24 MOST SIGN.
;000000;010460;000000;000000;000000;000000; % A-REG 23 MOST SIGN.
;000000;010440;000000;000000;000000;000000; % A-REG 22 MOST SIGN.
;000000;010420;000000;000000;000000;000000; % A-REG 21 MOST SIGN.
;000000;010400;000000;000000;000000;000000; % A-REG 20 MOST SIGN.
;000000;010360;000000;000000;000000;000000; % A-REG 17 MOST SIGN.
;000000;010340;000000;000000;000000;000000; % A-REG 16 MOST SIGN.
;000000;010320;000000;000000;000000;000000; % A-REG 15 MOST SIGN.
;000000;010300;000000;000000;000000;000000; % A-REG 14 MOST SIGN.
;000000;010260;000000;000000;000000;000000; % A-REG 13 MOST SIGN.
;000000;010240;000000;000000;000000;000000; % A-REG 12 MOST SIGN.
;000000;010220;000000;000000;000000;000000; % A-REG 11 MOST SIGN.
;000000;010200;000000;000000;000000;000000; % A-REG 10 MOST SIGN.
;000000;010160;000000;000000;000000;000000; % A-REG 7 MOST SIGN.
;000000;010140;000000;000000;000000;000000; % TOP OF STACK
;000000;010140;000000;000000;000000;000000; % A-REG 6 MOST SIGN.
;000000;010140;000000;000000;000000;000000; % FLOATING 1.0
;000000;010120;000000;000000;000000;000000; % A-REG 5 MOST SIGN.
;000000;010100;000000;000000;000000;000000; % FLOATING 0.0
;000000;010100;000000;000000;000000;000000; % FLOATING -1.0
;000000;010100;000000;000000;000000;000000; % A-REG 4 MOST SIGN.
;000000;010060;000000;000000;000000;000000; % DOUBLE FLOAT REG 3
;000000;010060;000000;000000;000000;000000; % A-REG 3 MOST SIGN.
;000000;010040;000000;000000;000000;000000; % DOUBLE FLOAT REG 2
;000000;010040;000000;000000;000000;000000; % A-REG 2 MOST SIGN.
;000000;010020;000000;000000;000000;000000; % DOUBLE FLOAT REG 1
;000000;010020;000000;000000;000000;000000; % A-REG 1 MOST SIGN.
;000000;010000;000000;000000;000000;000000; % DOUBLE FLOAT REG 0
;000000;010000;000000;000000;000000;000000; % A-REG 0 MOST SIGN.
;000000;006020;000000;000000;000000;000000; % EFFECTIVE ADDR REG 2
;000000;006000;000000;000000;000000;000000; % EFFECTIVE ADDR REG 1
;000000;005000;000000;000000;000000;000000; % DECODED BMR BIT
;000000;004760;000000;000000;000000;000000; % BIT MASK BIT 37
;000000;004740;000000;000000;000000;000000; % BIT MASK BIT 36
;000000;004720;000000;000000;000000;000000; % BIT MASK BIT 35
;000000;004700;000000;000000;000000;000000; % BIT MASK BIT 34
;000000;004660;000000;000000;000000;000000; % BIT MASK BIT 33
;000000;004640;000000;000000;000000;000000; % BIT MASK BIT 32
;000000;004620;000000;000000;000000;000000; % BIT MASK BIT 31
;000000;004600;000000;000000;000000;000000; % BIT MASK BIT 30
;000000;004560;000000;000000;000000;000000; % BIT MASK BIT 27
;000000;004540;000000;000000;000000;000000; % BIT MASK BIT 26
;000000;004520;000000;000000;000000;000000; % BIT MASK BIT 25
;000000;004500;000000;000000;000000;000000; % BIT MASK BIT 24
```

ND-500 micro mnemonics in alphabetical and numerical order

```
1023  'B_BM#23  ;000000;004460;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 23
1024  'B_BM#22  ;000000;004440;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 22
1025  'B_BM#21  ;000000;004420;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 21
1026  'B_BM#20  ;000000;004400;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 20
1027  'B_BM#17  ;000000;004360;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 17
1028  'B_BM#16  ;000000;004340;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 16
1029  'B_BM#15  ;000000;004320;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 15
1030  'B_BM#14  ;000000;004300;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 14
1031  'B_BM#13  ;000000;004260;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 13
1032  'B_BM#12  ;000000;004240;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 12
1033  'B_BM#11  ;000000;004220;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 11
1034  'B_BM#10  ;000000;004200;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 10
1035  'B_BM#7   ;000000;004160;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 7
1036  'B_BM#6   ;000000;004140;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 6
1037  'B_BM#5   ;000000;004120;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 5
1038  'B_BM#4   ;000000;004100;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 4
1039  'B_BM#3   ;000000;004060;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 3
1040  'B_BM#2   ;000000;004040;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 2
1041  'B_BM#1   ;000000;004020;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 1
1042  'B_BM#0   ;000000;004000;000000;000000;000000;000000;000000;000000; % BIT MASK BIT 0
1043  'B_DATA2  ;000000;002020;000000;000000;000000;000000;000000;000000; % EXTRA MEMORY DATA REGISTER
1044  'B_DATA   ;000000;002000;000000;000000;000000;000000;000000;000000; % MEMORY DATA
1045  'HWTW     ;000000;000300;000000;000000;000000;000000;000000;000000; % HALFW. TO WORD SIGN EXT.
1046  'BYTW     ;000000;000200;000000;000000;000000;000000;000000;000000; % BYTE TO WORD SIGN EXTENSION.
1047  'NSEXCONV ;000000;000200;000000;000000;000000;000000;000000;000000; % NO SIGN EXT. WITH B,XRESL
1048  'BYTH     ;000000;000100;000000;000000;000000;000000;000000;000000; % BYTE TO HALFW. SIGN EXT.
1049  'IX3      ;000000;000060;000000;000000;000000;000000;000000;000000; % INDEX REGISTER 3
1050  'IX2      ;000000;000040;000000;000000;000000;000000;000000;000000; % INDEX REGISTER 2
1051  'IX1      ;000000;000020;000000;000000;000000;000000;000000;000000; % INDEX REGISTER 1
1052  'D_CONST  ;000000;000015;144000;000000;000000;000000;000000;000000; % CONSTANT REGISTER
1053  'D_IDAT   ;000000;000015;142000;000000;000000;000000;000000;000000; % INSTRUCTION MEMORY DATA
1054  'D_LC     ;000000;000015;140000;000000;000000;000000;000000;000000; % LOOP COUNTER
1055  'UNLOCK   ;000000;000015;132000;000000;000000;000000;000000;000000; % UNLOCK INTERFACE TO N100
1056  'D_DP     ;000000;000015;114000;000000;000000;000000;000000;000000; % DISPLACEMENT REG
1057  'D_CSCNT  ;000000;000015;110000;000000;000000;000000;000000;000000; % CONTROL STORE CONTROL REG.
1058  'D_TAG    ;000000;000015;106000;000000;000000;000000;000000;000000; % TAG REGISTER
1059  'D_CSBRK  ;000000;000015;104000;000000;000000;000000;000000;000000; % CONTROL STORE BREAK REGISTER
1060  'D_CSWA   ;000000;000015;102000;000000;000000;000000;000000;000000; % CONT. ST. WRITE ADDRESS REG.
1061  'D_IODOUT ;000000;000015;100000;000000;000000;000000;000000;000000; % DATA-OUT
1062  'D_BMR    ;000000;000015;054000;000000;000000;000000;000000;000000; % BIT MASK REGISTER
1063  'D_ICCLR  ;000000;000015;044000;000000;000000;000000;000000;000000; % INDEX COUNTER CLEAR
1064  'D_CAR    ;000000;000015;042000;000000;000000;000000;000000;000000; % COMPUTED ADDRESS REGISTER
1065  'D_SHC    ;000000;000015;040000;000000;000000;000000;000000;000000; % SHIFT COUNT REGISTER
1066  'D_XSTI   ;000000;000015;024000;000000;000000;000000;000000;000000; % S1, BITS 17-19,21-24,30-31
1067  'D_DATAIN2;000000;000015;022000;000000;000000;000000;000000;000000; % EXTRA DATA-IN HOLD REG ON SL
1068  'D_DATAIN ;000000;000015;020000;000000;000000;000000;000000;000000; % DATA-IN HOLD REG. ON SLICE
1069  'D_TRAPCLR;000000;000015;016000;000000;000000;000000;000000;000000; % TRAP SYSTEM CLEAR
1070  'D_ISPIND ;000000;000015;014000;000000;000000;000000;000000;000000; % SET AND RESET ISB INDICATOR.
1071  'D_MMOD   ;000000;000015;012000;000000;000000;000000;000000;000000; % MEMORY MODUS REGISTER
1072  'D_SETLIM ;000000;000015;010000;000000;000000;000000;000000;000000; % SET LIMIT CONDS FOR HL-LL
1073  'D_TRAPONOFF;000000;000015;006000;000000;000000;000000;000000;000000; % TRAP SYSTEM ON/OFF
1074  'D_TE     ;000000;000015;004000;000000;000000;000000;000000;000000; % LOCAL TRAP ENABLE REG
1075  'D_S2     ;000000;000015;002000;000000;000000;000000;000000;000000; % STATUS REG 2
1076  'D_S1     ;000000;000015;000000;000000;000000;000000;000000;000000; % STATUS REG 1
1077  'D_HL     ;000000;000014;142000;000000;000000;000000;000000;000000; % HIGHER LIMIT REG
1078  'D_LL     ;000000;000014;140000;000000;000000;000000;000000;000000; % LOWER LIMIT REG
1079  'D_DSTSB  ;000000;000014;136000;000000;000000;000000;000000;000000; % DMM SEQ. TRANSL. SPEED-UP BF
1080  'D_DWIPGU ;000000;000014;134000;000000;000000;000000;000000;000000; % DMM WIP/PGU BROADSIDE (16 P)
1081  'D_DTSB   ;000000;000014;132000;000000;000000;000000;000000;000000; % DMM TRANSL. SPEED-UP BUF PAG
1082  'D_DASEG  ;000000;000014;130000;000000;000000;000000;000000;000000; % DMM ALT. SEGM.
```

ND-500 micro mnemonics in alphabetical and numerical order

```
1083  'D,DCSEG  ';000000;000014;126000;000000;000000;000000;000000; % DMM CURR. SEGMENT
1084  'D,DADDM  ';000000;000014;124000;000000;000000;000000;000000; % DMM ALT. DOMAIN REGISTER
1085  'D,DDOMR  ';000000;000014;122000;000000;000000;000000;000000; % DMM DOMAIN REG
1086  'D,DPROCC ';000000;000014;120000;000000;000000;000000;000000; % DMM PROCESS CONTR. REG
1087  'D,DSCFA  ';000000;000014;116000;000000;000000;000000;000000; % DMM SCR. FILE REGISTER ADDR.
1088  'D,DZPA   ';000000;000014;116000;000000;000000;000000;000000; % DATA ZERO POINT ADJUST REG.
1089  'D,DSCRF  ';000000;000014;114000;000000;000000;000000;000000; % DMM SCR. FILE (16 16-BIT RG)
1090  'D,DUPL   ';000000;000014;114000;000000;000000;000000;000000; % DATA UPPER PAGE LIMIT
1091  'D,DCLCACH';000000;000014;112000;000000;000000;000000;000000; % DATA CLEAR CACHE
1092  'D,DCON0  ';000000;000014;110000;000000;000000;000000;000000; % DATA MEM. CONTROL REG. 0
1093  'D,DCON1  ';000000;000014;106000;000000;000000;000000;000000; % DATA MEM. CONTROL REG. 1
1094  'D,DCINHLL';000000;000014;102000;000000;000000;000000;000000; % DATA CACHE INHIBIT LOWER LIM
1095  'D,DCINHLU';000000;000014;100000;000000;000000;000000;000000; % DATA CACHE INHIBIT UPPER LIM
1096  'D,DMCNTR ';000000;000014;076000;000000;000000;000000;000000; % DMM CONTROL
1097  'D,ISISB  ';000000;000014;074000;000000;000000;000000;000000; % IMM SEQ. TRANSL. SPEED-UP BF
1098  'D,IWIPGU ';000000;000014;074000;000000;000000;000000;000000; % IMM WIP/PGU BROADSIDE (16 P)
1099  'D,IITSB  ';000000;000014;072000;000000;000000;000000;000000; % IMM TRANSL. SPEED-UP BUF PAG
1100  'D,IASEG  ';000000;000014;070000;000000;000000;000000;000000; % IMM ALT. SEGM.
1101  'D,ICSEG  ';000000;000014;066000;000000;000000;000000;000000; % IMM CURR. SEGMENT
1102  'D,IADDM  ';000000;000014;064000;000000;000000;000000;000000; % IMM ALT. DOMAIN REGISTER
1103  'D,IDOMR  ';000000;000014;062000;000000;000000;000000;000000; % IMM DOMAIN REG
1104  'D,IPROCC ';000000;000014;060000;000000;000000;000000;000000; % IMM PROCESS CONTR. REG
1105  'D,ISCFA  ';000000;000014;056000;000000;000000;000000;000000; % IMM SCR. FILE REGISTER ADDR.
1106  'D,IZPA   ';000000;000014;056000;000000;000000;000000;000000; % INSTR. ZERO POINT ADJUST REG
1107  'D,ISCRF  ';000000;000014;054000;000000;000000;000000;000000; % IMM SCR. FILE (16 16-BIT RG)
1108  'D,IUPL   ';000000;000014;054000;000000;000000;000000;000000; % INSTR. UPPER PAGE LIMIT
1109  'D,ICLCACH';000000;000014;052000;000000;000000;000000;000000; % INSTR CLEAR CACHE
1110  'D,ICON0  ';000000;000014;050000;000000;000000;000000;000000; % INSTR MEM. CONTROL REG. 0
1111  'D,ICON1  ';000000;000014;046000;000000;000000;000000;000000; % INSTR MEM. CONTROL REG. 1
1112  'D,ICINHLL';000000;000014;042000;000000;000000;000000;000000; % INSTR CACHE INHIBIT LOWER L.
1113  'D,ICINHLU';000000;000014;040000;000000;000000;000000;000000; % INSTR CACHE INHIBIT UPPER L.
1114  'D,IMCNTR ';000000;000014;040000;000000;000000;000000;000000; % IMM CONTROL
1115  'D,AD#37  ';000000;000012;076000;000000;000000;000000;000000; % DEST IS A-REG 37 DOUBLE
1116  'D,AD#36  ';000000;000012;074000;000000;000000;000000;000000; % DEST IS A-REG 36 DOUBLE
1117  'D,AD#35  ';000000;000012;072000;000000;000000;000000;000000; % DEST IS A-REG 35 DOUBLE
1118  'D,AD#34  ';000000;000012;070000;000000;000000;000000;000000; % DEST IS A-REG 34 DOUBLE
1119  'D,AD#33  ';000000;000012;066000;000000;000000;000000;000000; % DEST IS A-REG 33 DOUBLE
1120  'D,AD#32  ';000000;000012;064000;000000;000000;000000;000000; % DEST IS A-REG 32 DOUBLE
1121  'D,AD#31  ';000000;000012;062000;000000;000000;000000;000000; % DEST IS A-REG 31 DOUBLE
1122  'D,AD#30  ';000000;000012;060000;000000;000000;000000;000000; % DEST IS A-REG 30 DOUBLE
1123  'D,AD#27  ';000000;000012;056000;000000;000000;000000;000000; % DEST IS A-REG 27 DOUBLE
1124  'D,AD#26  ';000000;000012;054000;000000;000000;000000;000000; % DEST IS A-REG 26 DOUBLE
1125  'D,AD#25  ';000000;000012;052000;000000;000000;000000;000000; % DEST IS A-REG 25 DOUBLE
1126  'D,AD#24  ';000000;000012;050000;000000;000000;000000;000000; % DEST IS A-REG 24 DOUBLE
1127  'D,AD#23  ';000000;000012;046000;000000;000000;000000;000000; % DEST IS A-REG 23 DOUBLE
1128  'D,AD#22  ';000000;000012;044000;000000;000000;000000;000000; % DEST IS A-REG 22 DOUBLE
1129  'D,AD#21  ';000000;000012;042000;000000;000000;000000;000000; % DEST IS A-REG 21 DOUBLE
1130  'D,AD#20  ';000000;000012;040000;000000;000000;000000;000000; % DEST IS A-REG 20 DOUBLE
1131  'D,AD#17  ';000000;000012;036000;000000;000000;000000;000000; % DEST IS A-REG 17 DOUBLE
1132  'D,AD#16  ';000000;000012;034000;000000;000000;000000;000000; % DEST IS A-REG 16 DOUBLE
1133  'D,AD#15  ';000000;000012;032000;000000;000000;000000;000000; % DEST IS A-REG 15 DOUBLE
1134  'D,AD#14  ';000000;000012;030000;000000;000000;000000;000000; % DEST IS A-REG 14 DOUBLE
1135  'D,AD#13  ';000000;000012;026000;000000;000000;000000;000000; % DEST IS A-REG 13 DOUBLE
1136  'D,AD#12  ';000000;000012;024000;000000;000000;000000;000000; % DEST IS A-REG 12 DOUBLE
1137  'D,AD#11  ';000000;000012;022000;000000;000000;000000;000000; % DEST IS A-REG 11 DOUBLE
1138  'D,AD#10  ';000000;000012;020000;000000;000000;000000;000000; % DEST IS A-REG 10 DOUBLE
1139  'D,AD#7   ';000000;000012;016000;000000;000000;000000;000000; % DEST IS A-REG 7 DOUBLE
1140  'D,AD#6   ';000000;000012;014000;000000;000000;000000;000000; % DEST IS A-REG 6 DOUBLE
1141  'D,AD#5   ';000000;000012;012000;000000;000000;000000;000000; % DEST IS A-REG 5 DOUBLE
1142  'D,AD#4   ';000000;000012;010000;000000;000000;000000;000000; % DEST IS A-REG 4 DOUBLE
```

ND-500 micro mnemonics in alphabetical and numerical order

```
1143  'D_AD#3     ;000000;000012;006000;000000;000000;000000;000000; % DEST IS A-REG 3 DOUBLE
1144  'D_AD#2     ;000000;000012;004000;000000;000000;000000;000000; % DEST IS A-REG 2 DOUBLE
1145  'D_AD#1     ;000000;000012;002000;000000;000000;000000;000000; % DEST IS A-REG 1 DOUBLE
1146  'D_AD#0     ;000000;000012;000000;000000;000000;000000;000000; % DEST IS A-REG 0 DOUBLE
1147  'D_AL#37    ;000000;000010;076000;000000;000000;000000;000000; % A37 LEAST
1148  'D_AL#36    ;000000;000010;074000;000000;000000;000000;000000; % A36 LEAST
1149  'D_AL#35    ;000000;000010;072000;000000;000000;000000;000000; % A35 LEAST
1150  'D_AL#34    ;000000;000010;070000;000000;000000;000000;000000; % A34 LEAST
1151  'D_AL#33    ;000000;000010;066000;000000;000000;000000;000000; % A33 LEAST
1152  'D_AL#32    ;000000;000010;064000;000000;000000;000000;000000; % A32 LEAST
1153  'D_AL#31    ;000000;000010;062000;000000;000000;000000;000000; % A31 LEAST
1154  'D_AL#30    ;000000;000010;060000;000000;000000;000000;000000; % A30 LEAST
1155  'D_AL#27    ;000000;000010;056000;000000;000000;000000;000000; % A27 LEAST
1156  'D_AL#26    ;000000;000010;054000;000000;000000;000000;000000; % A26 LEAST
1157  'D_AL#25    ;000000;000010;052000;000000;000000;000000;000000; % A25 LEAST
1158  'D_AL#24    ;000000;000010;050000;000000;000000;000000;000000; % A24 LEAST
1159  'D_AL#23    ;000000;000010;046000;000000;000000;000000;000000; % A23 LEAST
1160  'D_AL#22    ;000000;000010;044000;000000;000000;000000;000000; % A22 LEAST
1161  'D_AL#21    ;000000;000010;042000;000000;000000;000000;000000; % A21 LEAST
1162  'D_AL#20    ;000000;000010;040000;000000;000000;000000;000000; % A20 LEAST
1163  'D_AL#17    ;000000;000010;036000;000000;000000;000000;000000; % A17 LEAST
1164  'D_AL#16    ;000000;000010;034000;000000;000000;000000;000000; % A16 LEAST
1165  'D_AL#15    ;000000;000010;032000;000000;000000;000000;000000; % A15 LEAST
1166  'D_AL#14    ;000000;000010;030000;000000;000000;000000;000000; % A14 LEAST
1167  'D_AL#13    ;000000;000010;026000;000000;000000;000000;000000; % A13 LEAST
1168  'D_AL#12    ;000000;000010;024000;000000;000000;000000;000000; % A12 LEAST
1169  'D_AL#11    ;000000;000010;022000;000000;000000;000000;000000; % A11 LEAST
1170  'D_AL#10    ;000000;000010;020000;000000;000000;000000;000000; % A10 LEAST
1171  'D_AL#7     ;000000;000010;016000;000000;000000;000000;000000; % A7 LEAST
1172  'D_THA      ;000000;000010;016000;000000;000000;000000;000000; % TRAP HANDLER REG
1173  'D_AL#6     ;000000;000010;014000;000000;000000;000000;000000; % A6 LEAST
1174  'D_AL#5     ;000000;000010;012000;000000;000000;000000;000000; % A5 LEAST
1175  'D_AL#4     ;000000;000010;010000;000000;000000;000000;000000; % A4 LEAST
1176  'D_AL#3     ;000000;000010;006000;000000;000000;000000;000000; % A3 LEAST
1177  'D_AL#2     ;000000;000010;004000;000000;000000;000000;000000; % A2 LEAST
1178  'D_AL#1     ;000000;000010;002000;000000;000000;000000;000000; % A1 LEAST
1179  'D_AL#0     ;000000;000010;000000;000000;000000;000000;000000; % A0 LEAST
1180  'D_AM#37    ;000000;000006;076000;000000;000000;000000;000000; % A37 MOST
1181  'D_AM#36    ;000000;000006;074000;000000;000000;000000;000000; % A36 MOST
1182  'D_AM#35    ;000000;000006;072000;000000;000000;000000;000000; % A35 MOST
1183  'D_AM#34    ;000000;000006;070000;000000;000000;000000;000000; % A34 MOST
1184  'D_AM#33    ;000000;000006;066000;000000;000000;000000;000000; % A33 MOST
1185  'D_AM#32    ;000000;000006;064000;000000;000000;000000;000000; % A32 MOST
1186  'D_AM#31    ;000000;000006;062000;000000;000000;000000;000000; % A31 MOST
1187  'D_AM#30    ;000000;000006;060000;000000;000000;000000;000000; % A30 MOST
1188  'D_AM#27    ;000000;000006;056000;000000;000000;000000;000000; % A27 MOST
1189  'D_AM#26    ;000000;000006;054000;000000;000000;000000;000000; % A26 MOST
1190  'D_AM#25    ;000000;000006;052000;000000;000000;000000;000000; % A25 MOST
1191  'D_AM#24    ;000000;000006;050000;000000;000000;000000;000000; % A24 MOST
1192  'D_AM#23    ;000000;000006;046000;000000;000000;000000;000000; % A23 MOST
1193  'D_AM#22    ;000000;000006;044000;000000;000000;000000;000000; % A22 MOST
1194  'D_AM#21    ;000000;000006;042000;000000;000000;000000;000000; % A21 MOST
1195  'D_AM#20    ;000000;000006;040000;000000;000000;000000;000000; % A20 MOST
1196  'D_AM#17    ;000000;000006;036000;000000;000000;000000;000000; % A17 MOST
1197  'D_AM#16    ;000000;000006;034000;000000;000000;000000;000000; % A16 MOST
1198  'D_AM#15    ;000000;000006;032000;000000;000000;000000;000000; % A15 MOST
1199  'D_AM#14    ;000000;000006;030000;000000;000000;000000;000000; % A14 MOST
1200  'D_AM#13    ;000000;000006;026000;000000;000000;000000;000000; % A13 MOST
1201  'D_AM#12    ;000000;000006;024000;000000;000000;000000;000000; % A12 MOST
1202  'D_AM#11    ;000000;000006;022000;000000;000000;000000;000000; % A11 MOST
```

ND-500 micro mnemonics in alphabetical and numerical order

```
1203  'D_A##10  ; 000000; 020000; 000000; 000000; 000000; 000000; % A10 MOST
1204  'D_A##7   ; 000000; 016000; 000000; 000000; 000000; 000000; % A7 MOST
1205  'D_TOS    ; 000000; 016000; 000000; 000000; 000000; 000000; % TOP OF STACK
1206  'D_A##6   ; 000000; 014000; 000000; 000000; 000000; 000000; % A6 MOST
1207  'D_A##5   ; 000000; 012000; 000000; 000000; 000000; 000000; % A5 MOST
1208  'D_A##4   ; 000000; 010000; 000000; 000000; 000000; 000000; % A4 MOST
1209  'D_A##3   ; 000000; 006000; 000000; 000000; 000000; 000000; % A3 MOST
1210  'D_A##2   ; 000000; 004000; 000000; 000000; 000000; 000000; % A2 MOST
1211  'D_A##1   ; 000000; 002000; 000000; 000000; 000000; 000000; % A1 MOST
1212  'D_A##0   ; 000000; 000000; 000000; 000000; 000000; 000000; % A0 MOST
1213  'D_B      ; 000000; 006000; 000000; 000000; 000000; 000000; % B REGISTER
1214  'D_R      ; 000000; 004000; 000000; 000000; 000000; 000000; % R REGISTER
1215  'D_L      ; 000000; 002000; 000000; 000000; 000000; 000000; % L REGISTER
1216  'D_X#3    ; 000000; 006000; 000000; 000000; 000000; 000000; % INDEX 3 REG
1217  'D_X#2    ; 000000; 004000; 000000; 000000; 000000; 000000; % INDEX 2 REG
1218  'D_X#1    ; 000000; 002000; 000000; 000000; 000000; 000000; % INDEX 1 REG
1219  'D_X#0    ; 000000; 000000; 000000; 000000; 000000; 000000; % INDEX 0 REG
1220  'XDBOV    ; 000000; 000002; 000000; 000000; 000000; 000000; % XD-BUS MOVE (NO ALU TRANSF).
1221  'NPCSEL   ; 000000; 001000; 000000; 000000; 000000; 000000; % SELECT NPC TO L REGISTER
1222  'ORA      ; 000000; 001000; 000000; 000000; 000000; 000000; % OR AOPR ACC. TO INSTRUCTION.
1223  'ORB      ; 000000; 000400; 000000; 000000; 000000; 000000; % OR BOPR ACC. TO INSTRUCTION.
1224  'ORD      ; 000000; 000200; 000000; 000000; 000000; 000000; % OR DEST ACC. TO INSTRUCTION.
1225  'MEM_WR   ; 000000; 000104; 100000; 000000; 000000; 000000; % WRITE MEMORY WITH ORED TYPE.
1226  'MEM_RD   ; 000000; 000104; 000000; 000000; 000000; 000000; % READ MEMORY WITH ORED TYPE.
1227  'ORT      ; 000000; 000040; 000000; 000000; 000000; 000000; % OR DATATYPE ACC. TO INSTR.
1228  'TYP_BY   ; 000000; 000030; 000000; 000000; 000000; 000000; % DATATYPE IS BYTE
1229  'TYP_HW   ; 000000; 000020; 000000; 000000; 000000; 000000; % DATATYPE IS HALFWORD
1230  'TYP_F    ; 000000; 000010; 000000; 000000; 000000; 000000; % DATATYPE IS FLOATING
1231  'MEM_MV4  ; 000000; 000005; 060000; 000000; 000000; 000000; % MOVE 4 WORDS IN DATA MEMORY.
1232  'MEM_MV3  ; 000000; 000005; 040000; 000000; 000000; 000000; % MOVE 3 WORDS IN DATA MEMORY.
1233  'MEM_MV2  ; 000000; 000005; 020000; 000000; 000000; 000000; % MOVE 2 WORDS IN DATA MEMORY.
1234  'MEM_MV1  ; 000000; 000005; 000000; 000000; 000000; 000000; % MOVE 1 WORD IN DATA MEMORY.
1235  'MEM_WR4  ; 000000; 000004; 160000; 000000; 000000; 000000; % WRITE 4 BYTES TO DATA MEMORY
1236  'MEM_WR3  ; 000000; 000004; 140000; 000000; 000000; 000000; % WRITE 3 BYTES TO DATA MEMORY
1237  'MEM_WR2  ; 000000; 000004; 120000; 000000; 000000; 000000; % WRITE 2 BYTES TO DATA MEMORY
1238  'MEM_WR1  ; 000000; 000004; 100000; 000000; 000000; 000000; % WRITE 1 BYTE TO DATA MEMORY.
1239  'MEM_RD4  ; 000000; 000004; 060000; 000000; 000000; 000000; % READ 4 BYTES FROM DATAMEMORY
1240  'MEM_RD3  ; 000000; 000004; 040000; 000000; 000000; 000000; % READ 3 BYTES FROM DATAMEMORY
1241  'MEM_RD2  ; 000000; 000004; 020000; 000000; 000000; 000000; % READ 2 BYTES FROM DATAMEMORY
1242  'MEM_RD1  ; 000000; 000004; 000000; 000000; 000000; 000000; % READ 1 BYTE FROM DATA MEMORY
1243  'ALTMOD   ; 000000; 000002; 000000; 000000; 000000; 000000; % SELECT BIT 1 IN MEMORY MODUS
1244  'CSAVE    ; 000000; 000000; 010000; 000000; 000000; 000000; % CONDITION SAVE (PUSH)
1245  'C_SEQD   ; 000000; 000000; 004000; 002000; 004000; 000000; % DELAYED COND. SEQUENCE
1246  'C_SEQ    ; 000000; 000000; 004000; 002000; 000000; 000000; % CONDITIONAL SEQUENCE
1247  'C_ALU    ; 000000; 000000; 002000; 000000; 000000; 000000; % CONDITIONAL ALU
1248  'C_MEM    ; 000000; 000000; 001000; 000000; 000000; 000000; % CONDITIONAL MEMORY REFERENCE
1249  'NADL     ; 000000; 000000; 001000; 000000; 000000; 000000; % NOT ADDR LATCH.
1250  'COND_LCZ ; 000000; 000000; 000760; 000000; 000000; 000000; % LOOP COUNTER ZERO
1251  'COND_SAVC2 ; 000000; 000000; 000740; 000000; 000000; 000000; % SAVED CONDITION 2
1252  'COND_SAVC1 ; 000000; 000000; 000720; 000000; 000000; 000000; % SAVED CONDITION 1
1253  'COND_PARITY ; 000000; 000000; 000700; 000000; 000000; 000000; % PARITY OF ALU OUTPUT
1254  'COND_MDZ ; 000000; 000000; 000660; 000000; 000000; 000000; % DIVIDE BY ZRO
1255  'COND_ICRDY ; 000000; 000000; 000640; 000000; 000000; 000000; % INSTRUCTION MEMORY READY
1256  'COND_MBS ; 000000; 000000; 000620; 000000; 000000; 000000; % MICRO BCD SIGN
1257  'COND_MBO ; 000000; 000000; 000600; 000000; 000000; 000000; % MICRO BCD OVERFLOW
1258  'COND_MFU ; 000000; 000000; 000560; 000000; 000000; 000000; % MICRO FLOAT UNDERFLOW
1259  'COND_MFO ; 000000; 000000; 000540; 000000; 000000; 000000; % MICRO FLOAT OVERFLOW
1260  'COND_MFS ; 000000; 000000; 000520; 000000; 000000; 000000; % MICRO FLOATING SIGN
1261  'COND_PDONE ; 000000; 000000; 000500; 000000; 000000; 000000; % PART DONE
1262  'COND_MCNZ ; 000000; 000000; 000460; 000000; 000000; 000000; % MICRO CARRY AND NOT M. ZERO.
```

ND-500 micro mnemonics in alphabetical and numerical order

```
1263 'COND_MSBNZ  ' ;000000;000000;000000;000440;000000;000000;000000; % M. SIGN OR M. ZERO
1264 'COND_TRAP   ' ;000000;000000;000000;000420;000000;000000;000000; % TEST CONDITION IS TRAP
1265 'COND_MSEXO  ' ;000000;000000;000000;000400;000000;000000;000000; % M. SIGN EXOR M. OVERFLOW
1266 'COND_MOVFL  ' ;000000;000000;000000;000360;000000;000000;000000; % MICRO OVERFLOW
1267 'COND_MSGN   ' ;000000;000000;000000;000340;000000;000000;000000; % MICRO SIGN
1268 'COND_MCRY   ' ;000000;000000;000000;000320;000000;000000;000000; % MICRO CARRY
1269 'COND_MZRO   ' ;000000;000000;000000;000300;000000;000000;000000; % MICRO ZERO
1270 'COND_CONOP  ' ;000000;000000;000000;000260;000000;000000;000000; % CONSTANT OPERAND
1271 'COND_MFUFO  ' ;000000;000000;000000;000240;000000;000000;000000; % MICRO FLOAT UNDER/OVERFLOW
1272 'COND_CNZ    ' ;000000;000000;000000;000220;000000;000000;000000; % CARRY AND NOT ZERO
1273 'COND_SORZ   ' ;000000;000000;000000;000200;000000;000000;000000; % SIGN OR ZERO
1274 'COND_DATOP  ' ;000000;000000;000000;000160;000000;000000;000000; % DATA OPERAND
1275 'COND_K      ' ;000000;000000;000000;000140;000000;000000;000000; % TEST COND. IS K (FLAG)
1276 'COND_OVFL   ' ;000000;000000;000000;000120;000000;000000;000000; % OVERFLOW
1277 'COND_SGN    ' ;000000;000000;000000;000100;000000;000000;000000; % SIGN
1278 'COND_CRY    ' ;000000;000000;000000;000060;000000;000000;000000; % CARRY
1279 'COND_ZRO    ' ;000000;000000;000000;000040;000000;000000;000000; % ZERO
1280 'COND_ENTER  ' ;000000;000000;000000;000020;000000;000000;000000; % NEXT MACRO INSTR IS ENTER
1281 'REP        ' ;000000;000000;000017;170000;000000;000000;000000; % REPEAT CURRENT MICRO INSTR.
1282 'JMPSTK     ' ;000000;000000;000017;070000;000000;000000;000000; % JUMP TO STACK ADDRESS
1283 'NOPOPRET   ' ;000000;000000;000017;070000;000000;000000;000000; % RETURN WITHOUT POP
1284 'POPRET     ' ;000000;000000;000017;050000;000000;000000;000000; % RETURN FROM SUBROUTINE
1285 'JSRSTK     ' ;000000;000000;000017;010000;000000;000000;000000; % JUMP TO SUBR IN STACK
1286 'NEXT       ' ;000000;000000;000016;170000;000000;000000;000000; % NEXT MICRO INSTRUCTION
1287 'JMPREL     ' ;000000;000000;000016;160000;000000;000000;000000; % JUMP REL TO COMP ADDR REG
1288 'NEXTNS     ' ;000000;000000;000016;110000;000000;000000;000000; % NEXT WITH SEP STACK CONTROL.
1289 'JSRREL     ' ;000000;000000;000016;100000;000000;000000;000000; % JUMP TO SUBR REL TO COMP ADR
1290 'JMP        ' ;000000;000000;000014;070000;000000;000000;000000; % JUMP ABSOLUTE
1291 'JMPNS      ' ;000000;000000;000014;010000;000000;000000;000000; % JUMP WITH SEP STACK CONTROL.
1292 'JSR        ' ;000000;000000;000014;010000;000000;000000;000000; % JUMP TO SUBROUTINE ABSOLUTE.
1293 'JMPCAR     ' ;000000;000000;000012;070000;000000;000000;000000; % JUMP TO COMPUTED ADDR. REG
1294 'JSRCAR     ' ;000000;000000;000012;010000;000000;000000;000000; % JUMP TO SUBR IN COMP ADDR R.
1295 'JMPMAP     ' ;000000;000000;000010;070000;000000;000000;000000; % JUMP TO MAP ADDRESS
1296 'JSRMAP     ' ;000000;000000;000010;010000;000000;000000;000000; % JUMP TO SUBR IN MAP ADDRESS.
1297 'JMPWA      ' ;000000;000000;000006;070000;000000;000000;000000; % JUMP TO ADDRESS IN WA
1298 'JSRWA      ' ;000000;000000;000006;010000;000000;000000;000000; % JUMP TO SUBR IN WA
1299 'HBRET      ' ;000000;000000;000004;070000;000000;000000;000000; % RETURN TO HARDW. BRANCH REG.
1300 'POP        ' ;000000;000000;000004;040000;000000;000000;000000; % POP STACK
1301 'LOAD       ' ;000000;000000;000002;020000;000000;000000;000000; % LOAD STACK
1302 'SET        ' ;000000;000000;000001;004000;000000;000000;000000; % SET CONDITION SELECT
1303 'F_REP      ' ;000000;000000;000000;001774;000000;000000;000000; % REPEAT CURR. INSTR. IF FALSE
1304 'F_JMPSTK   ' ;000000;000000;000000;001734;000000;000000;000000; % JUMP STACK IF FALSE
1305 'F_NOPOPRET ' ;000000;000000;000000;001734;000000;000000;000000; % RETURN WITHOUT POP IF FALSE.
1306 'F_POPRET   ' ;000000;000000;000000;001724;000000;000000;000000; % RETURN FROM SUBR. IF FALSE.
1307 'F_JSRSTK   ' ;000000;000000;000000;001704;000000;000000;000000; % JUMP TO SUBR. IN STACK IF FL
1308 'F_NEXT     ' ;000000;000000;000000;001674;000000;000000;000000; % NEXT MICRO INSTR. IF FALSE.
1309 'F_JMPREL   ' ;000000;000000;000000;001670;000000;000000;000000; % JUMP RELATIVE IF FALSE
1310 'F_NEXTNS   ' ;000000;000000;000000;001644;000000;000000;000000; % NEXT WITH SEP ST CTRL IF FLS
1311 'F_JSRREL   ' ;000000;000000;000000;001640;000000;000000;000000; % JUMP TO SUBR. REL IF FALSE.
1312 'F_JMP      ' ;000000;000000;000000;001434;000000;000000;000000; % JUMP ABSOLUTE IF FALSE
1313 'F_JMPNS    ' ;000000;000000;000000;001404;000000;000000;000000; % JUMP NOT PUSH STACK IF FALSE
1314 'F_JSR      ' ;000000;000000;000000;001404;000000;000000;000000; % JUMP TO SUBROUTINE IF FALSE.
1315 'F_JMPCAR   ' ;000000;000000;000000;001234;000000;000000;000000; % JUMP CAR IF FALSE
1316 'F_JSRCAR   ' ;000000;000000;000000;001204;000000;000000;000000; % JUMP TO SUBR. IN CAR IF FALS
1317 'F_JMPMAP   ' ;000000;000000;000000;001034;000000;000000;000000; % JUMP MAP IF FALSE
1318 'F_JSRMAP   ' ;000000;000000;000000;001004;000000;000000;000000; % JUMP TO SUBR. IN MAP IF FALS
1319 'F_JMPWA    ' ;000000;000000;000000;000634;000000;000000;000000; % JUMP WA IF FALSE
1320 'F_JSRWA    ' ;000000;000000;000000;000604;000000;000000;000000; % JUMP TO SUBR. IN WA IF FALSE
1321 'F_POP      ' ;000000;000000;000000;000020;000000;000000;000000; % POP STACK IF FALSE
1322 'F_LOAD     ' ;000000;000000;000000;000010;000000;000000;000000; % LOAD STACK IF FALSE
```

ND-500 micro mnemonics in alphabetical and numerical order

```
1323  'ST,SAVB      ;000000;000000;000000;000003;000100;000000;000000; % SAVE BCD STATUS
1324  'K,1IFZ       ;000000;000000;000000;000003;000000;000000;000000; % SET K IF ZERO ALU
1325  'ST,SAVF      ;000000;000000;000000;000002;000100;000000;000000; % SAVE FLOATING STATUS
1326  'K,ZRO        ;000000;000000;000000;000002;000000;000000;000000; % CLEAR K
1327  'ST,SAVAC     ;000000;000000;000000;000001;000100;000000;000000; % SAVE COMPARE STATUS
1328  'K,ONE        ;000000;000000;000000;000001;000000;000000;000000; % SET K
1329  'PRF,ISAMP    ;000000;000000;000000;000000;170000;000000;000000; % INTERRUPT SAMPLE
1330  'PRF,WFIN     ;000000;000000;000000;000000;160000;000000;000000; % WAIT FOR PREFETCH FINISHED
1331  'PRF,FOPR     ;000000;000000;000000;000000;150000;000000;000000; % FETCH OPERAND
1332  'PRF,CEOPF    ;000000;000000;000000;000000;120000;000000;000000; % END-OF-OP IF FALSE
1333  'PRF,CLEAR    ;000000;000000;000000;000000;110000;000000;000000; % PREFETCH CLEAR
1334  'PRF,ACONT    ;000000;000000;000000;000000;100000;000000;000000; % ADDR ARITHMETIC CONTINUE
1335  'PRF,PCONT    ;000000;000000;000000;000000;070000;000000;000000; % PREFETCH CONTINUE
1336  'PRF,START    ;000000;000000;000000;000000;060000;000000;000000; % START
1337  'PRF,FARG     ;000000;000000;000000;000000;050000;000000;000000; % FETCH ARGUMENT
1338  'PRF,FADC     ;000000;000000;000000;000000;040000;000000;000000; % FETCH ADDR CODE
1339  'PRF,FOPC     ;000000;000000;000000;000000;030000;000000;000000; % FETCH OPCODE
1340  'PRF,CEOPT    ;000000;000000;000000;000000;020000;000000;000000; % END-OF-OP IF TRUE
1341  'PRF,EOP      ;000000;000000;000000;000000;010000;000000;000000; % END-OF-OPERATION
1342  'SLOW2        ;000000;000000;000000;000000;006000;000000;000000; % FIRST HALF CYCLE IS SLOW
1343  'SLOW1        ;000000;000000;000000;000000;004000;000000;000000; % SLOW CYCLE
1344  'FAST         ;000000;000000;000000;000000;002000;000000;000000; % FAST CYCLE
1345  'W,IOEM       ;000000;000000;000000;000000;001600;000000;000000; % WAIT FOR IO/EXT/MEM
1346  'W,MEM        ;000000;000000;000000;000000;001400;000000;000000; % WAIT FOR MEMORY
1347  'W,IOEPM      ;000000;000000;000000;000000;001200;000000;000000; % WAIT FOR IO/EXT/PMEM
1348  'W,PMEM       ;000000;000000;000000;000000;001000;000000;000000; % WAIT FOR PREVIOUS MEM CYCLE
1349  'W,EXT        ;000000;000000;000000;000000;000600;000000;000000; % WAIT FOR EXT. ARITHM. UNIT
1350  'W,IO         ;000000;000000;000000;000000;000400;000000;000000; % WAIT FOR INPUT/OUTPUT
1351  'W,XD         ;000000;000000;000000;000000;000200;000000;000000; % WAIT FOR XD-BUS
1352  'ST,SAVA      ;000000;000000;000000;000000;000100;000000;000000; % SAVE ALU STATUS
1353  'LCDECR       ;000000;000000;000000;000000;000040;000000;000000; % LOOP COUNTER DECREMENT
1354  'IADJ         ;000000;000000;000000;000000;000020;000000;000000; % INDEX ADJUST (INCR IND C.)
1355  'PASSAB       ;000000;000000;000000;000000;000016;000000;000000; % PASS ADDR BOPR THR ADDR ARIT
1356  'AA+AB        ;000000;000000;000000;000000;000014;000000;000000; % ADDR AOPR + ADDR BOPR
1357  'PASSAA       ;000000;000000;000000;000000;000010;000000;000000; % PASS ADDR AOPR THR ADDR ARIT
1358  'AB,PC        ;000000;000000;000000;000001;140000;000000;000000; % ADDR BOPR IS P REGISTER
1359  'AB,R         ;000000;000000;000000;000001;100000;000000;000000; % ADDR BOPR IS R REGISTER
1360  'AB,B         ;000000;000000;000000;000001;040000;000000;000000; % ADDR BOPR IS B REGISTER
1361  'AB,DPARG     ;000000;000000;000000;000001;000000;000000;000000; % ADDR BOPR IS DP
1362  'AB,8IX       ;000000;000000;000000;000001;140000;000000;000000; % ADDR BOPR IS XREG SC. BY 8
1363  'AB,4IX       ;000000;000000;000000;000001;100000;000000;000000; % ADDR BOPR IS XREG SC. BY 4
1364  'AB,ORAB      ;000000;000000;000000;000001;060000;000000;000000; % ADDR BOPR IS OR-CONTROLLED
1365  'AB,2IX       ;000000;000000;000000;000001;040000;000000;000000; % ADDR BOPR IS XREG SC. BY 2
1366  'AB,1/8IX     ;000000;000000;000000;000001;020000;000000;000000; % ADDR BOPR IS XREG SCALED 1/8
1367  'AA,EA2       ;000000;000000;000000;000001;014000;000000;000000; % ADDR AOPR IS EFF. ADDR 2 REG
1368  'AA,DP2       ;000000;000000;000000;000001;010000;000000;000000; % ADDR AOPR IS DP2 REG
1369  'AA,EA1       ;000000;000000;000000;000001;004000;000000;000000; % ADDR AOPR IS EFF. ADDR 1 REG
1370  'EX,APIMUL    ;000000;000000;000000;000000;002037;000000;000000; % LAST FL. DIV. STEP SA*SP(IN)
1371  'EX,APIMULA   ;000000;000000;000000;000000;002036;000000;000000; % THIRD FL. DIV STEP SA*SP(INV)
1372  'EX,PPIMULP   ;000000;000000;000000;000000;002035;000000;000000; % 4. FL DIV STEP SP*SP(INV)
1373  'EX,APMULA    ;000000;000000;000000;000000;002034;000000;000000; % SA*SP TO SA
1374  'EX,PRMULP    ;000000;000000;000000;000000;002033;000000;000000; % SEC FL DIV STEP B*1/B TO SP
1375  'EX,MULA      ;000000;000000;000000;000000;002032;000000;000000; % MULT AND SAVE A*B TO SA
1376  'EX,UMUL      ;000000;000000;000000;000000;002031;000000;000000; % UNSIGN MULT A*B TO CPU
1377  'EX,MUL       ;000000;000000;000000;000000;002030;000000;000000; % MULTIPLY A*B TO CPU
1378  'EX,TORMULA   ;000000;000000;000000;000000;002027;000000;000000; % A * 1/A TO SA
1379  'EX,ARMULA    ;000000;000000;000000;000000;002026;000000;000000; % FRST FL DIV STEP A*1/B TO SA
1380  'EX,COMPARE   ;000000;000000;000000;000000;002025;000000;000000; % COMPARE A AND B
1381  'EX,DIFF      ;000000;000000;000000;000000;002024;000000;000000; % FL SUBT. A-B TO CPU
1382  'EX,ASUMA     ;000000;000000;000000;000000;002023;000000;000000; % FL ADD SA+B TO SA
```

ND-500 micro mnemonics in alphabetical and numerical order

```
1383  'EX,ASUM      ;000000;000000;000000;002022;000000;000000; % FL ADD SA+B TO CPU
1384  'EX,SUM       ;000000;000000;000000;002021;000000;000000; % FL ADD A+B TO CPU
1385  'EX,DTOFR     ;000000;000000;000000;002017;000000;000000; % CNVRT DBL FL TO SGL, ROUNDED
1386  'EX,SHR       ;000000;000000;000000;002013;000000;000000; % SHIFT ROTATIONAL
1387  'EX,SHL       ;000000;000000;000000;002012;000000;000000; % SHIFT LOGICAL
1388  'EX,SHA       ;000000;000000;000000;002010;000000;000000; % SHIFT ARITHMETIC
1389  'EX,INT       ;000000;000000;000000;002007;000000;000000; % INTEGER PART TRUNCATED
1390  'EX,INTR      ;000000;000000;000000;002006;000000;000000; % INTEGER PART ROUNDED
1391  'EX,CTI       ;000000;000000;000000;002005;000000;000000; % CONVERT TO INTEGER TRUNCATED
1392  'EX,CIIR      ;000000;000000;000000;002004;000000;000000; % CNVRT TO INTEGER WITH ROUND.
1393  'EX,UCTF      ;000000;000000;000000;002001;000000;000000; % UNSIGNED CNVRT TO FLOATING
1394  'EX,CTF       ;000000;000000;000000;002000;000000;000000; % CONVERT TO FLOATING
1395  'F,A+B+C      ;000000;000000;000000;001011;000000;000000; % AOPR + BOPR + CARRY IF FALSE
1396  'F,A-B-1+C    ;000000;000000;000000;001006;000000;000000; % AOPR - BOPR - 1 + CRY IF FLS
1397  'BCD          ;000000;000000;000000;001000;000000;000000; % BCD TYPE
1398  'F,A+1        ;000000;000000;000000;000417;000000;000000; % INCREMENT AOPR IF FALSE
1399  'F,A+A+1      ;000000;000000;000000;000414;000000;000000; % AOPR + AOPR +1 IF FALSE
1400  'F,A+B+1      ;000000;000000;000000;000411;000000;000000; % AOPR + BOPR + 1 IF FALSE
1401  'F,A-B        ;000000;000000;000000;000406;000000;000000; % AOPR - BOPR IF FALSE
1402  'DOUBLE       ;000000;000000;000000;000400;000000;000000; % DOUBLE FLOATING
1403  'MV,DINTOD    ;000000;000000;000000;000300;000000;000000; % DATA-IN TO D
1404  'MV,DTODP     ;000000;000000;000000;002200;000000;000000; % ADDR SAVE FR MEM TO DP1-DP2.
1405  'MV,DTOXBL    ;000000;000000;000000;000100;000000;000000; % MEM DBUS TO FL BBUS LEAST SG
1406  'XRES         ;000000;000000;000000;000040;000000;000000; % ENABLE EXT RESULT INTO CPU
1407  'F,ADIR       ;000000;000000;000000;000037;000000;000000; % AOPR DIRECT IF FALSE
1408  'F,AND        ;000000;000000;000000;000036;000000;000000; % AOPR AND BOPR IF FALSE
1409  'F,ANDCB      ;000000;000000;000000;000035;000000;000000; % AOPR AND COMPL(BOPR) IF FLS.
1410  'F,ZRO        ;000000;000000;000000;000034;000000;000000; % FORCED ZERO IF FALSE
1411  'F,OR         ;000000;000000;000000;000033;000000;000000; % AOPR OR BOPR IF FALSE
1412  'F,BDIR       ;000000;000000;000000;000032;000000;000000; % BOPR DIRECT IF FALSE
1413  'F,XOR        ;000000;000000;000000;000031;000000;000000; % AOPR OR BOPR IF FALSE
1414  'F,ANDCA      ;000000;000000;000000;000030;000000;000000; % COMPL(AOPR) AND BOPR IF FLS.
1415  'F,ORCB       ;000000;000000;000000;000027;000000;000000; % AOPR OR C(BOPR) IF FALSE
1416  'F,XNOR       ;000000;000000;000000;000026;000000;000000; % AOPR EXNOR BOPR IF FALSE
1417  'F,BDIRC      ;000000;000000;000000;000025;000000;000000; % BOPR COMPLEMENTED IF FALSE
1418  'F,NOR        ;000000;000000;000000;000024;000000;000000; % AOPR NOR BOPR IF FALSE
1419  'F,ONE        ;000000;000000;000000;000023;000000;000000; % FORCED ONE IF FALSE
1420  'F,ORCA       ;000000;000000;000000;000022;000000;000000; % C(AOPR) OR BOPR IF FALSE
1421  'F,NAND       ;000000;000000;000000;000021;000000;000000; % AOPR NAND BOPR IF FALSE
1422  'F,ADIRC      ;000000;000000;000000;000020;000000;000000; % AOPR COMPLEMENTED IF FALSE
1423  'F,A+A        ;000000;000000;000000;000014;000000;000000; % AOPR + AOPR IF FALSE
1424  'F,A+B        ;000000;000000;000000;000011;000000;000000; % AOPR + BOPR IF FALSE
1425  'F,A-B-1      ;000000;000000;000000;000006;000000;000000; % AOPR - BOPR - 1 IF FALSE
1426  'EA2INH       ;000000;000000;000000;100000;000000;000000; % EFF. ADDR. 2 REG CLOCK INH..
1427  'EA1INH       ;000000;000000;000000;040000;000000;000000; % EFF. ADDR. 1 REG CLOCK INH..
1428  'A,AMU        ;000000;000000;000000;000000;000000;000000; % DOUBLE FLOAT REG 0
1429  'A,AMMU       ;000000;000000;000000;000000;000000;000000; % A-REG 0 MOST SIGN.
1430  'AA,DP1       ;000000;000000;000000;000000;000000;000000; % ADDR AOPR IS DP1 REG
1431  'AB,IX        ;000000;000000;000000;000000;000000;000000; % ADDR BOPR IS INDEX REG
1432  'ALU,A-1      ;000000;000000;000000;000000;000000;000000; % DECREMENT AOPR
1433  'CRY,ZRO      ;000000;000000;000000;000000;000000;000000; % CARRY IS FORCED ZERO
1434  'F,A-1        ;000000;000000;000000;000000;000000;000000; % DECREMENT AOPR IF FALSE
1435  'F,PUSH       ;000000;000000;000000;000000;000000;000000; % PUSH STACK IF FALSE
1436  'IFMEM        ;000000;000000;000000;000000;000000;000000; % IF MEMORY, THEN ...
1437  'IFT          ;000000;000000;000000;000000;000000;000000; % IF CONDITION TRUE, THEN...
1438  'IXU          ;000000;000000;000000;000000;000000;000000; % INDEX REGISTER 0
1439  'MV,DTOXBM    ;000000;000000;000000;000000;000000;000000; % MEM DBUS TO FL BBUS MOST SIG
1440  'PUSH         ;000000;000000;000000;000000;000000;000000; % PUSH (SEQUENCER) STACK
1441  'SHARG        ;000000;000000;000000;000000;000000;000000; % SHIFT COUNTER ARGUMENT
1442  'SINGLE       ;000000;000000;000000;000000;000000;000000; % SINGLE FLOATING
```

ND-500 micro mnemonics in alphabetical and numerical order

```
1443 'TYP,W      ';000000;000000;000000;000000;000000;000000;000000;000000;000000; % DATATYPE IS WORD
1444
```

ND-500 micro mnemonics in alphabetical and numerical order

1445
1446

ND-500 micro mnemonics in alphabetical and numerical order

Index

Systems that put people first