ND-500
Micro Program Guide

ND-05.012.01

NORSK DATA A.S

ND-500
Micro Program Guide

ND-05.012.01

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
01/83	Version 01

ND-05.012.01
ND-500 Micro Program Guide
January 1983

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 document is intended to give a short introduction to ND-500 micro programming and will state rules for the use of some of the commands available in the ND-500 mnemonic symbols.

The reader

The document is addressed to people writing micro program routines for the ND-500 and to people working with ND-500 hardware.

Prerequisite knowledge

Some knowledge about the ND-500 architecture and detailed knowledge about the ND-500 hardware is required to use the manual. This can be found in the manual:

ND-500 Reference manual ND - 05.009

TABLE OF CONTENTS

Section Page

Section Page

1 INTRODUCTION

Micro instructions in the ND-500 control the communication between different parts of the central processing unit (CPU).

Arithmetic logic unit (ALU).
External arithmetic. Floating point or BCD arithmetic.
Prefetch processor.
Cache and memory system.
Input and output system.
Trap system.
Sequencer.

A macro instruction will need a number of micro program instructions depending on the complexity of the instruction to be performed. The prefetch processor will, for the same macro instruction, execute a number of cycles depending on the number of operands involved in the operation.

The micro program will use data fetched by the prefetch processor from a register or from the cache and memory system in ALU operations or output from operations carried out by the external arithmetic. This involves synchronization with the prefetch processor, the cache and memory system and the external arithmetic.

The micro program may be divided into four parts.

1 Entry point part.
2 Part for macro instructions requiring more than one micro instruction.
3 The ND-100 / ND-500 communication.
4 Trap handling.

2 ND-500 REGISTERS

Macro instructions requiring more than one micro instruction usually require scratch registers for saving operands or results. Some of the scratch registers are allocated for special use in the micro program and should not be changed by user micro program routines.

Registers allocated for constants used in micro program:

AM#0 : AL#0	Double floating register = D1 register.
AM#1 : AL#1	Double floating register = D2 register.
AM#2 : AL#2	Double floating register = D3 register.
AM#3 : AL#3	Double floating register = D4 register.
AM#4 : AL#4	Floating constant -1.0.
AM#5 : AL#5	Floating constant 0.0.
AM#6 : AL#6	Floating constant 1.0.
AM#37: AL#37	Integer constant -1.

Registers allocated for for special use in the micro program:

AM#7	Top of stack register.
AL#7	Trap handler register.
AM#10	Process number.
AL#10	Communication / process status.
AM#12	Address of the status word in ND-100
	block involved in current / last message.
AL#12	
AM#13	
AL#13	
AM#14	CAD. Current executing alternativ domain.
AL#14	
AM#15	CED. Current executing domain.
AL#15	
AM#16	
AL#16	
AM#17	
AL#17	PS. Process segment number and process control register.

The scratch registers allocated for free use are:

AM#11 : AL#11	Free to use. Save in context (process description) block when changing process.
AM#20 to AM#36	Floating scratch most.
AL#20 to AL#36	Floating scratch least.

3 FORMAT OF THE MICRO WORD

The ND-500 micro word, being 144 bits wide, is divided into groups, each group controlling special parts or functions in the ND-500 CPU. The value of each function is given a mnemonic symbol. The mnemonic symbols may be combined to perform functions. Symbols using the same field should not be used together. This will be allowed by the ND-500 micro assembler, as long as the mnemonics do not try to set the same bits in the field.

The mnemonic symbols may be handled by the ND-500 micro assembler. See appendix for further description of the assembler.

Micro word bits	
143	Control function.
142	Control store parity.
141 - 137	SP clock inhibit.
136 - 135	ALU-function select (true).
134 - 125	Carry select.
124 - 116	A-operand select.
115 - 106	B-operand select.
105 - 101	Destination select.
100 - 99	OR-logic enable.
98 - 93	Data type control.
92	Memory control.
91	Condition save.
90	Conditional sequence enable.
89	Conditional ALU enable.
88 - 84	Conditional MEM enable.
83 - 76	Test condition select.
75	Sequence instructions (true).
74	Set condition select.
73 - 66	Delayed sequence.
65 - 64	Alternative (false) sequence instruction.
63 - 60	Status control.
59 - 58	Prefetch control.
57 - 55	Timing control.
54	Wait for select.
53	Status save.
52	Loop couter decrement.
51 - 43	Index counter adjust.
42	Address arithmetic control.
41 - 40	External arithmetic activate.
41 - 40	External arithmetic ident.
39 - 38	Carry select alternative ALU-function.
37	Data bus control.
36 - 32	External result enable.
36 - 32	External arithmetic function select.
31	Alternative (false) ALU-function.
30	Effective address 1 clock inhibit.
29 - 16	Effective address 2 clock inhibit.
31 - 0	Absolute micro program address.
15 - 0	Long argument. 32 bits wide.
	Short argument. Sign extended to 32 bits.

4 OR-LOGIC FUNCTION

The prefetch processor will fetch instructions and operands, and will generate addresses for operands to be written.

The prefetch processor supplies the OR-logic with information about source and destination select and data type control decoded from the macro instruction being executed. For source select the prefetch will select operands either on the A-operand or the B-operand, depending on the operand definitions for the macro instruction. Operands are defined as a register, a general operand or a constant.

If a register is an operand, it is routed to the A-operand.

In the case of a general operand or a constant as operand, routing is to the B-operand.

These rules must be followed when using the OR-logic for operand read.

The OR-logic commands and their use are:

ORA	Select register as A-operand.
ORB	Select general operand as B-operand.
ORT	Select data type for operation.
ORD	Select register destination (if any).

The OR-logic will dominate source and destination select and data-type control specified in a micro instruction. Specifying ORT and TYP,BY will cause data type to be decoded from the macro instruction being executed and TYP,BY not affecting the operation.

5 ARITHMETIC FUNCTIONS

The ALU and the external arithmetic are used for performing arithmetic functions. As input for an operation, A-operand, B-operand, data type control and destination may be selected from separate fields or they may be selected by the OR-logic.

The true ALU-functions and the external arithmetic are controlled from separate fields so you can run ALU-functions and external arithmetic functions in the same micro instruction.

5.1 ALU-FUNCTIONS

The arithmetic logic unit (ALU) may perform integer arithmetic and logic functions.

The ALU-functions are specified by the symbols ALU,<func> for true ALU-function select. The ALU-functions may also be specified as a false ALU-function by the F,<func> commands. This is only activated by the C,ALU command.

The ALU-functions are:

Arithmetic operations :

```
       ALU,A+1      : A-operand plus one.
       ALU,A+A      : A-operand plus A-operand.
       ALU,A+A+1    : A-operand plus A-operand plus one.
       ALU,A+B      : A-operand plus B-operand.
       ALU,A+B+1    : A-operand plus B-operand plus one.
       ALU,A+B+C    : A-operand plus B-operand plus carry.
       ALU,A-1      : A-operand minus one.
       ALU,A-B      : A-operand minus B-operand.
       ALU,A-B-1    : A-operand minus B-operand minus one.
       ALU,A-B-1+C  : A-operand minus B-operand minus one plus carry.
Logic operations:
       ALU,ADIR     : A-operand direct.
       ALU,ADIRC    : A-operand complemented.
       ALU,BDIR     : B-operand direct.
       ALU,BDIRC    : B-operand complemented.
       ALU,FZRO     : Force zero from ALU output.
       ALU,FONE     : Force ones from ALU output.
       ALU,AND      : A-operand AND B-operand.
       ALU,ANDCA    : A-operand complemented AND B-operand.
       ALU,ANDCB    : A-operand AND B-operand complemented.
       ALU,NAND     : A-operand NAND B-operand.
       ALU,OR       : A-operand OR B-operand.
       ALU,ORCA     : A-operand complemented OR B-operand.
       ALU,ORCB     : A-operand OR B-operand complemented.
       ALU,NOR      : A-operand NOR B-operand.
       ALU,XOR      : A-operand XOR B-operand.
       ALU,XNOR     : A-operand XNOR B-operand.
```

5.2 FLOATING ARITHMETIC

The floating point arithmetic may perform shift operations, floating and integer conversion, floating point arithmetic and integer multiply. The floating point arithmetic is activated by the EX,<func> commands.

For shift and conversion commands only the B-operand is used. The exception to this is the EX,DTOFR command which needs operand supplied on the A-operand with B-operand equal to zero.

For shift and multiply, shift count may be specified either to be taken from the shift count register or from the short argument field.

Shift count > 0 : Shift left. Shift count < 0 : Shift right (not for rotational shift).

Any integer register, floating double register, floating most register or memory data may be selected as operands for the floating point arithmetic.

> Note that floating least register is invalid as the operand select.

Floating double and floating most registers may be directly selected as destination. If integer registers or memory is destination, the data has to be read through the ALU. The result is always returned on the XOPA-bus (floating-bus), enabled by the XRES command to the ALU B-operand.

> Note that the commands B,XRESM and B,XRESL contain the XRES command.

Some of the operations give opportunity to save the result either in the SA or the SP registers located at the floating point arithmetic.

For floating operations TYP,F and SINGLE or DOUBLE must be specified. For integer operations the OR-logic (ORT) as well as the integer data type control commands may be used.

The operations performed by the floating point arithmetic lasts for more than 200 nsec. (one micro cycle) and synchronization is done by the W,EXT command.

The commands for activating floating point arithmetic are:

```
EX,SHA      : Shift arithmetic.
EX,SHL      : Shift logic.
EX,SHR      : Shift rotational.

EX,CTF      : Convert integer to floating.
EX,UCTF     : Unsigned convert integer to floating.
EX,DTOFR    : Double to single convert with rounding.
EX,INT      : Integer part (in float) truncated.
EX,INTR     : Integer part (in float) rounded.

EX,CTI      : Convert floating to integer truncated.
EX,CTIR     : Convert floating to integer with rounding.

EX,SUM      : Floating add      :  A + B → CPU.
EX,ASUM     : Floating add      : SA + B → CPU.
EX,ASUMA    : Floating add      : SA + B → SA.

EX,DIFF     : Floating subtract :  A - B → CPU.
EX,COMPARE  : Floating compare  :  A - B.

EX,MUL      : Multiply          :  A * B → CPU.
EX,MULA     : Multiply and save :  A * B → SA.
EX,UMUL     : Unsigned multiply :  A * B → CPU.
EX,APMULA   : Multiply and save : SA *SP → SA.

EX,ARMULA   : Divide step 1     :  A * 1/B' → SA.
EX,BRMULP   : Divide step 2     :  B * 1/B' → SP.
EX,APIMULA  : Divide step 3     : SA * SP'  → SA.
EX,PPIMULA  : Divide step 4     : SP * SP'  → SP.
EX,APIMUL   : Last divide step  : SA * SP'  → CPU.

EX,TORMULA  : Divide            :  A * 1/B' → CPU.
```

To do a proper single floating point divide, the divide steps 1, 2 and the last one are to be used. For a double floating point devide, all the divide steps are to be used. The double floating point includes adding one to the least significant part of the result of the division. The carry from the least significant part is then added to the most significant part of the division.

An integer divide is done by converting the operands to double floating point numbers and then do a double floating point divide.

To avoid divide by 0, the test condition COND,MDZ, is selected when doing the first divide step. Result of this test condition may be used in the following micro instruction together with the second divide step to abort the divide in case of x/0.0.

5.3 BCD ARITHMETIC

The BCD arithmetic may perform the decimal functions ADD, SUB, MPY, SHIFT, COMPARE, PACK, UNPACK, BINC and BCDC. The BCD arithmetic is micro programmed and is controlled by the CPU micro program. The different operations are carried out by the BCD arithmetic.

The CPU micro program will, for a BCD operation, control the function and handle the descriptors of BCD numbers involved in an operation. The result of the operation is returned on the XOPA-bus, - on the most significant part of the data bus.

Data input to the BCD arithmetic may be selected from any floating most register, from any integer register or from data memory.

Synchronization is done by the W,EXT command and is required between each activate of the BCD arithmetic. Data type BCD must be specified together with each activate.

The commands for activating the BCD arithmetic are:

BCD,ADD	: BCD add without rounding.
BCD,ADDR	: BCD add with rounding.
BCD,BCDC	: BCD to binary convert.
BCD,BINC	: Binary to BCD convert.
BCD,COMP	: BCD compare.
BCD,DATA	: Data to or from BCD arithmetic.
BCD,DESCA	: BCD descriptor.
BCD,MPY	: BCD multiply without rounding.
BCD,MPYR	: BCD multiply with rounding.
BCD,PACK	: Pack BCD to ASCII without rounding.
BCD,PACKR	: Pack BCD to ASCII with rounding.
BCD,SHIFT	: BCD shift without rounding.
BCD,SHIFTR	: BCD shift with rounding.
BCD,SUB	: BCD subtract without rounding.
BCD,SUBR	: BCD subtract with rounding.
BCD,UPACK	: Unpack BCD to ASCII without rounding.
BCD,UPACKR	: Unpack BCD to ASCII with rounding.

6 DESTINATION CONTROL

Different commands and facilities may be used for destination select.

The result of an arithmetic operation, either ALU or external arithmetic, may be routed to the desired destination by the D,<dest> commands if a register is the destination, or with memory write if memory is the destination.

Data from memory are always 'latched' on the external arithmetic B-bus. This may also be controlled by the micro program. Data in at the end of a cycle (read in current or a preceding cycle) may be 'latched' on the external arithmetic B-bus.

Data input at the end of a cycle may also be 'latched' in the DP1 register.

If data in is to be used unchanged in a write operation, the data in register may be routed to the data out register.

Move data in to <dest>:

MV,DTOXBM	Data in routed to external B-bus most.
MV,DTOXBL	Data in routed to external B-bus least.
MV,DTODP	Data in routed to DP1.
MV,DINTOD	Data in routed to data out.

The XD-bus has several registers connected which may be moved by specfying XDMOV. Source as A,XD,<source> and destination as D,<dest> must be specified. This leaves the ALU B-operand free for other purposes. The loop counter, located on the XD-bus is a very useful register for loop control. Test condition for loop counter equal to zero may be selected. Also note the command LCDECR for decrementing the loop counter in paralell with ALU,xxx operations or floating point operations.

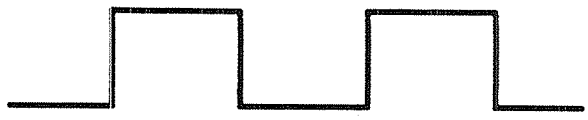
Example:

```
XDMOV A,XD,SARG D,LC 4;  %.. 4 → LC
```

7 WAIT FOR COMMANDS

Some commands are used for waiting, ie., stretch a micro cycle to synchronize with either floating point arithmetic, cache and memory system or input/output system. The wait state is only active if a request has been generated.

The different wait states are explained below.



	Command:	Function.	Wait state.
1	W,XD	Wait for master clear. Full stop. →	b
2	W,IO	Wait for input / output. Wait at end of cycle. →	c
3	W,PMEM	Wait for previous memory request. Wait at start of cycle for memory read from a preceding cycle. →	a
4	W,MEM	Wait for memory request. As for W,PMEM and →	a
		wait at end of cycle for memory request generated in current cycle. →	c
5	W,EXT	Wait for external arithmetic. Wait at start of cycle if result is taken through the ALU. →	a
		Wait at end of cycle if result is taken direct to the register block. →	c
6	W,IOEM	Wait for combination 2,4 and 5. →	a/c
7	W,IOEPM	Wait for combination 2,3 and 5. →	a/c

8 MICRO PROGRAM SEQUENCE

The ND-500 micro instructions may use different commands for sequencing the micro program. The commands will either cause the next micro instruction in a sequence to be executed, or some kind of a jump. A stack holding maximum four different adresses, with the top word as input to the micro program word counter (m.p.c) is also available.

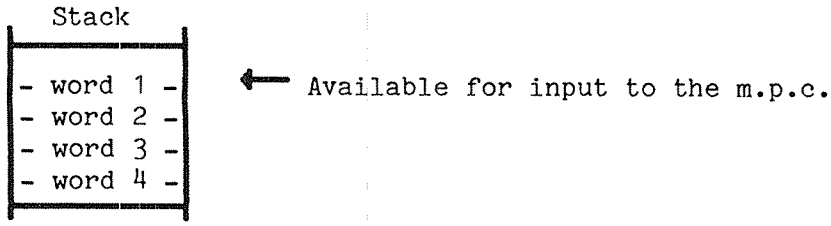


Figure: 8I

The sequence is controlled by a sequence command and a stack command. The ND-500 micro sequence control mnemonics are defined with both sequence commands and stack commands except for NEXTNS, F,NEXTNS, JMPNS and F,JMPNS. These require a stack command for desired function on stack.

The sequence control is also available as false sequence control written as F,<seq>. False stack control is written as F,<contr>. The false sequence control is only activated by the C,SEQ command.

8.1 STACK COMMANDS

Stack commands and functions are:

HOLD : Leave stack unchanged.
LOAD : Word 1 is changed to current micro address + 1.
The rest of the stack is unchanged.
PUSH : Word 4 is lost.
Word 3 → word 4.
Word 2 → word 3.
Word 1 → word 2.
Current address + 1 → word 1.
POP : Word 1 may be used as return address. Word 1 ← word 2.
Word 2 ← word 3.
Word 3 ← word 4.
Word 4 ← word 4.

8.2 SEQUENCE COMMANDS

The sequence control functions are with stack commands:

Stack command: None.

NEXTNS	Take next micro instruction.
JMPNS	Jump to micro address <addr.>.

Stack command: HOLD

NEXT	Take next micro instruction.
JMP	Jump to micro address <addr.>.
JMPMAP	Jump to map address, ie., start of next macro instr.
JMPCAR	Jump to computed address taken from CAR.
JMPREL	Jump relative to m.p.c. displacement-1 in CAR.
JMPSTK	Jump to stack address (word 1).
NOPOPRET	Jump to stack address (word 1).
JMPWA	Jump to write address (WA).
REP	Repeat current micro instruction.
HBRET	Return to address HB (hardware branch register).

Stack command: PUSH.

JSR	Jump to subroutine address <addr.>.
JSRMAP	Jump to subroutine map address.
JSRCAR	Jump to subroutine address taken from CAR.
JSRREL	Jump to subroutine relativ to m.p.c.(see JMPREL).
JSRSTK	Jump to subroutine stack address (word 1).
JSRWA	Jump to subroutine write address.

Stack command: POP

POPRET	Jump to stack address (word 1).

Example of use of stack:

```
a)    NEXTNS PUSH;
b)    NEXT;
c)    ALU,<func> SET COND,<cond> NEXT;
d)    C,SEQ F,NEXTNS F,LOAD %.. Change return address.
      IFT   NEXT;           %.. Hold stack unchanged.
e)    ALU,<func> SET COND,<cond> NEXT;
f)    C,SEQ F,NOPOPRET      %.. Return to stack address b or e.
      IFT   NEXT;           %.. Leave the loop.
```

8.3 MICRO CYCLE TIME

Commands for control of micro cycle time:

SLOW1	Stretch both high and low state of micro cycle. Must be used when using XD destinations. Not to be used in connection with PRF,EOP.
SLOW2	Stretch start (high state) of micro cycle. Must be used when using XD sources. Used in combination with JMPMAP and PRF,EOP.

9 CONDITIONAL OPERATIONS

The test conditions listed below may be used for three different purposes in the ND-500 micro program. Some conditions may select test result either from main status or from the micro-status. These conditions are written as (M)ZRO. This means that the condition COND,ZRO will take test result from the Z-bit in main status.The condition COND,MZRO takes test result from the micro-status. To activate the selected condition the command SET must be used.

9.1 ND-500 TEST CONDITIONS

Possible test conditons and their relations

	COND,condition	true : false
Arithmetic operations:		
Equal	(M)ZRO	true
Unequal	(M)ZRO	false
Signed:		
Greater	(M)SORZ	false
Greater or equal	(M)SGN	false
Less	(M)SGN	true
Less or equal	(M)SORZ	true
True less or greater or equal:		
Less	MSEXO	true
Greater or equal	MSEXO	false
Magnitude:		
Greater	(M)CNZ	true
Greater or equal	(M)CRY	true
Less	(M)CRY	false
Less or equal	(M)CNZ	false
Overflow	(M)OVFL	true : false
Parity (from ALU-output):		
Odd parity	PARITY	true
Even parity	PARITY	false
External arithmetic:		
Avoid X / 0	MDZ	true : error false : ok
Floating over or underflow	MFUFO	true : false
Floating sign	MFS	true : false
Floating overflow	MFO	true : false
Floating underflow	MFU	true : false
BCD overflow	MBO	true : false
Process status:		
Loop counter = 0	LCZ	true : false
Flag	K	true : false
Data source / destination	DATOP	true : false
Constant source / destination	CONOP	true : false
Part done, ie., restart	PDONE	true : false
Trap	TRAP	true : false
Next instruction is enter code	ENTER	true : false
BCD invalid operation	MIVO	true : false

Instruction channel ready	ICRDY	true : false
Saved condition 1	SAVC1	true : false
Saved condition 2	SAVC2	true : false

9.2 CONDITIONAL SEQUENCE

Select true or false sequence of micro instruction.

Result of a conditon set in a preceding micro instruction with result from previous micro instruction affecting selected condition, determines true or false sequence when C,SEQ is used. When delayed sequence (C,SEQD) is used, the test condition may be set in current or a preceding micro instruction.

Example:

```
a)    ALU,A-B A,<ao> B,<bo> TYP,<tt> SET COND,MZRO NEXT;
b)    C,SEQ   NEXT F,JMP 5000,0;
c)    ALU,A+B A,<ao> B,<bo> TYP,<tt> NEXT;
```

Result from ALU-operation in micro instruction a gives either true or false micro zero. Result = 0 gives true sequence, ie., next → c from micro cycle b. Result >< 0 gives false sequence, ie., jump to micro address 5000 from micro instruction b.

The example above is equivalent to:

```
a)    ALU,A-B A,<ao> B,<bo> TYP,<tt> NEXT;
b)    C,SEQD SET COND,MZRO  NEXT F,JMP 5000,0;
c)    ALU,A+B A,<ao> B,<bo> TYP,<tt> NEXT;
```

9.3 CONDITIONAL ALU-OPERATION

Select true or false ALU operation in a micro instruction.

Result of a condition set in current or a preceding micro instruction with result from previous micro instruction affecting the selected condition, determines true or false ALU-operation.

Example:

```
a)    ALU,A-B A,<ao> B,<bo> TYP,<tt> SET COND,MSGN NEXT;
b)    C,ALU   ALU,<func> F,<func>
      A,<ao>  B,<bo> TYP,<tt> D,<dest> NEXT;
```

Result from ALU-operation in micro instruction a gives either true or false micro sign.

Micro sign, ie., <ao> < <bo> gives true ALU-function in micro instruction b.

Not micro sign, ie., <ao> >= <bo> gives false ALU-function in micro instruction b.

The condition may for ALU-operation select be set in the same micro instruction as is used in a conditional ALU-operation.

The example above is equivalent to:

```
a)    ALU,A-B A,<ao> B,<bo> TYP,<tt> NEXT;
b)    C,ALU   ALU,<func> F,<func> SET COND,MSGN
      A,<ao>  B,<bo> TYP,<tt> D,<dest> NEXT;
```

9.4 CONDITIONAL MEMORY REFERENCE

Conditional memory reference. This request is only generated if the condition is true.

The result of a condition set in the current or a preceding micro instruction with the result from the previous micro instruction affecting the selected condition, gives memory request, read or write, only if the selected condition is true.

Example:

```
a)    ALU,ADIR  A,DATA TYP,BY SET COND,PARITY NEXT;
b)    C,MEM     MEM,WR1 W,MEM AA+AB AA,<ao> AB,<bo>
      ALU,XOR   A,DATA B,BM#7 NEXT;
```

The result from micro instruction a is used to change parity of a byte.

Parity ok, ie., condition false, no write request in micro instruction b.

Parity not ok, ie., condition true, write request in micro instruction b with parity bit changed.

9.5 CONDITION SAVE

By using CSAVE, any test condition may be saved for later use. The saved condition may be selected for test, true or false, in a later micro instruction by setting saved condition 1 or 2.

CSAVE saves the result of the condition set in current or a preceding micro instruction with result from previous micro instruction affecting the selected condition.

Example:

```
a)    ALU,<func>  A,<ao> B,<bo> TYP,<tt> SET COND,<cond> NEXT;
b)    CSAVE NEXT;
c)    == == == == == == NEXT;
      == == == == == == NEXT;

n)    SET COND,SAVC1 NEXT;
m)    C,SEQ  JMP F,<seq>
      C,MEM  MEM,<read/write> AA+AB AA,<ao> AB,<bo>
      C,ALU  ALU,<func> F,<func>;
```

The result from micro instruction a is saved in micro instruction b. Saved condition 1 is selected in micro instruction n and may be used for test in the following or a later micro cycle. This is equal to the following example.

```
a)    ALU,<func>  A,<ao> B,<bo> TYP,<tt> NEXT;
b)    CSAVE SET COND,<cond> NEXT;
c)    == == == == == == NEXT;
      == == == == == == NEXT;

n)    SET COND,SAVC1 NEXT;
m)    C,SEQ  JMP F,<seq>
      C,MEM  MEM,<read/write> AA+AB AA,<ao> AB,<bo>
      C,ALU  ALU,<func> F,<func>;
```

10 CONTROL OF STATUS BITS

Status bits may be controlled directly by using either the ALU,OR or the ALU,ANDCB function and bit mask for setting or resetting of status bits. For reading of the status, A,XD,S2 is used for status bits 63 - 32 and A,XD,S1 is used for status bits 31 - 0. For writing status, D,S2 is used for writing status bits 63 - 32, D,S1 is used for writing status bits 29 - 25 and bits 16 - 0 and D,XST1 is used for writing status bits 31 - 30 and bits 24 - 17.

For operations affecting data status bits, commands are used for control of data status bits according to result of operation.

Data status bits save:

ST,SAVA	: Save status from ALU-operation.
ST,SAVC	: Save status from ALU-operation in compare.
ST,SAVF	: Save status from floating-operation.
ST,SAVB	: Save status from BCD-operation.

Flag (K) and descriptor range (DR) control:

K,ZRO	: K ← 0.
K,ONE	: K ← 1.
K,1IFZ	: K ← 1 if MZRO = 1. DR ← 0.

11 PREFETCH PROCESSOR COMMANDS

The prefetch commands are used to control the next operation of the prefetch processor. This operation will depend on the macro instruction executed and is partly controlled by the prefetch micro code for the macro instruction. This implies that the prefetch commands used have to complement the prefetch micro code.

The prefetch commands are:

PRF,ISAMP	: Interupt sample.
PRF,PCONT	: Next step in prefetch (PCONT).
PRF,EOP	: End of operation. ISAMP and PCONT.
PRF,CEOPT	: Enable prefetch branch if condition is true.
PRF,CEOPF	: Enable prefetch branch if condition is false.
PRF,FADC	: Start fetch of a general operand.
PRF,WFIN	: Wait for previous prefetch operation to be finished.
PRF,CLEAR	: Clear prefetch.
PRF,START	: Start prefetch.
PRF,ACONT	: Not used.
PRF,FOPR	: Not used.
PRF,FARG	: Not used.
PRF,FOPC	: Not used.

12 ADDRESS ARITHMETIC

The address arithmetic is controlled by the prefetch processor but may also be controlled by the micro program. Since ND-500 is byte addressed the micro program must handle the address arithmetic according to the data-type in question using the address arithmetic.

The commands for control of address arithmetic are:

AA+AB	Address A-operand plus address B-operand.
PASSAA	Address A-operand direct through.
PASSAB	Address B-operand direct through.

Input to the address arithmetic:

Address A-operand:

AA,DP1	DP1 register as input.
AA,DP2	DP2 register as input.
AA,EA1	EA1 register as input.
AA,EA2	EA2 register as input.

Address B-operand:

IX0	B-operand is index register 1.
IX1	B-operand is index register 2.
IX2	B-operand is index register 3.
IX3	B-operand is index register 4.
AB,B	B-operand is B (Base register).
AB,R	B-operand is R (Record register).
AB,L	B-operand is L (Link register).
AB,PC	B-operand is P (Program counter).
AB,DPARG	B-operand is sign extended argument.
AB,ORAB	B-operand is index register selected from the macro instruction with data type scaling determined by the prefetch processor.

Address B-operand index scaling for data type		: Data type :
AB,1/8IX	Index register scaled by 1/8	: Bit.
AB,IX	Index register scaled by 1	: Byte.
AB,2IX	Index register scaled by 2	: Half word.
AB,4IX	Index register scaled by 4	: Word,sing.float.
AB,8IX	Index register scaled by 8	: Doubl.float.

Output of the address arithmetic is latched in the EA1 and EA2 registers. This may be avoided by either the EA1INH or the EA2INH commands. The address arithmetic activate will cause address latch to be sent to the cache and the memory system when it is required for read or write operations. This may be avoided by the NADL command. EA1INH and EA2INH has no effect on the address latch signal.

The micro programmer may hold base addresses in either DP1,DP2,EA1 or the EA2 registers to generate addresses relative to these. The DP1 and DP2 registers are coupled as a stack and destination DP (D,DP or MV,DTODP) will cause DP1 → DP2 with new contents in DP1. The command AB,DPARG will also cause DP1 → DP2 with new contents in DP1. When the prefetch processor is active, it will change the DP-registers.

Base addresses should therefore not be placed in either DP1 or DP2 when the commands mentioned above are used.

The prefetch uses the EA2 register for address calculate . The EA2INH, EA1 enable or EA2 enable must not be used in the same micro instruction as the prefetch is activated for operand fetch.

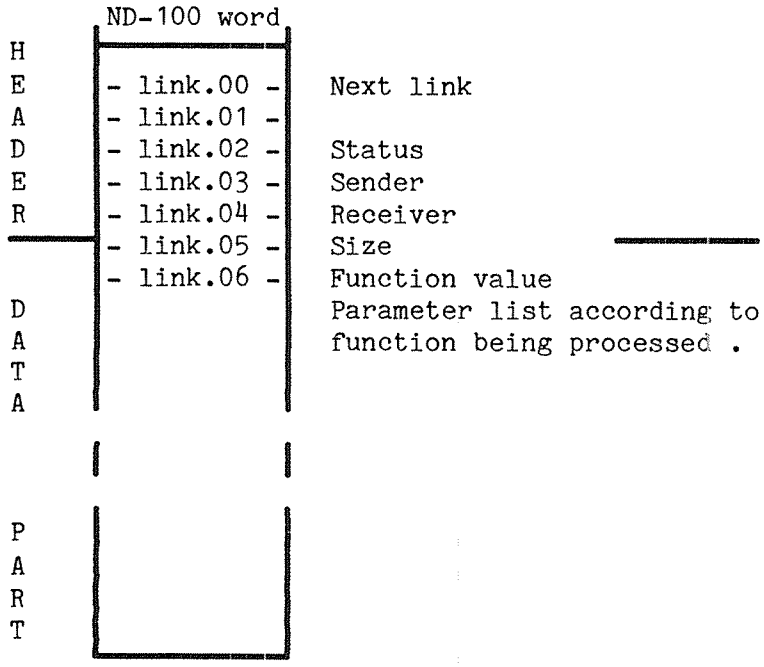
13 ND-100 ND-500 COMMUNICATION

At initialization of the ND-500, the micro program is loaded into the control store and started. After going through some initialization, the micro program enters the IDLE-loop. The initialization of the ND-500 implies clearing of the prefetch, setting of the floating and integer constant registers, resetting of trap enable registers and resetting the status register. The call/enter flag is also initiated. Then the micro program enters the communication part and initiates the communication before the IDLE loop is entered. Nothing but an activate or a terminate from the ND-100 can cause the micro program to leave the IDLE loop.

The communication between the ND-100 and ND-500 is built on a message block, residing in RESIDENT of SINTRAN III. Before an activate is given, the message block is initiated. The way it is initiated depends on the operation to be carried out. The activate gives the control to the ND-500. The ND-500 examines the message block and a micro program routine is entered according to the function specified in the block. The ND-500 will then use the block to return messages back to the ND-100.

Each block contains a header and a data part. The header consists of six words in ND-100 describing the message. The data part consists of a function value and a number of parameters depending on operation to be carried out.

13.1 THE MESSAGE BLOCK



Next link: The two first words of the block hold the start address of the next block. If start address of the next block is equal to -1, this means end of link.

Status of the block: Status gives information about the message currently being processed.

0	:	Block free.
1	:	Message to ND-500.
2	:	Message in process. Set by micro program at start of handling the message.
3	:	Answer to ND-100. Set when the micro program is finished handling the message.
4	:	Error return from ND-500 .

Sender: Address of RT description for sender.

Receiver: Receiver is the ND-500 process number to receive the block.

Size: Size is the the size of the data part of the message block.

The data part: Each message between the ND-100 and the ND-500 contains a data part. The first word of the data part defines the function to be performed. The different functions require different numbers of parameters to be involved in the data part of the link.

Function value and their related functions used are:

value		function
1	:	Read micro program version.
6	:	Physical data memory examine.
7	:	Physical data memory deposit.
10	:	Logical data memory read.
11	:	Logical data memory write.
12	:	Set cache mode.
13	:	Physical data memory read.
14	:	Physical data memory write.
16	:	Register examine.
17	:	Register deposit.
20	:	Register read.
21	:	Register write.
23	:	Start
23	:	Monitor call.
23	:	Trap.
24	:	Restart after monitor call.
25	:	Restart after trap.
30	:	Physical segment read.
31	:	Physical segment write.
34	:	Logicl instruction memory read.
35	:	Logicl instruction memory write.
42	:	Programmed trap.
44	:	Histogram read.

13.2 READ MICRO PROGRAM VERSION

Data part	Value	Function
- link.06 -	1	Read micro program version.
- link.07 -	←	Micro program version returned.

13.3 PHYSICAL DATA MEMORY EXAMINE

Data part	Value	Function
- link.06 -	6	Physical data memory examine.
- link.07 -	→	Physical ND-500 address.
- link.10 -		
- link.11 -	←	Returned data from ND-500.
- link.12 -		

13.4 PHYSICAL DATA MEMORY DEPOSIT

Data part	Value	Function
- link.06 -	7	Physical data memory for deposit.
- link.07 -	→	Physical ND-500 address.
- link.10 -		
- link.11 -	→	Data to deposit.
- link.12 -		

13.5 LOGICAL DATA MEMORY READ

Data part	Value	Function
- link.06 -	10	Logical data memory read.
- link.07 -	→	Logical ND-500 address.
- link.10 -		
- link.11 -	→	Physical ND-100 address.
- link.12 -		
- link.13 -	→	Number of bytes.

13.6 LOGICAL DATA MEMORY WRITE

Data part	Value	Function
- link.06 -	11	Logical data memory write.
- link.07 -	→	Logical ND-500 address.
- link.10 -		
- link.11 -	→	Physical ND-100 address.
- link.12 -		
- link.13 -	→	Number of bytes.

13.7 SET CACHE MODE

Data part	Value	Function
- link.06 -	12	Set cache mode.
- link.07 -	→	Cache control.

Cache control:

Instruction and data cache control are controlled by the 16 bit word transmitted to the ND-500.

Bits 0- 3 : Instruction cache partition clear.
Bits 4- 7 : Instruction cache partition set.
Bits 10-13 : Data cache partition clear.
Bits 14-17 : Data cache partition set.

13.8 PHYSICAL DATA MEMORY READ

Data part	Value	Function
- link.06 -	13	Physical data memory read.
- link.07 -	→	Physical ND-500 address.
- link.10 -		
- link.11 -	→	Physical ND-100 address.
- link.12 -		
- link.13 -	→	Number of bytes.

13.9 PHYSICAL DATA MEMORY WRITE

Data part	Value	Function
- link.06 -	14	Physical data memory write.
- link.07 -	→	Physical ND-500 address.
- link.10 -		
- link.11 -	→	Physical ND-100 address.
- link.12 -		
- link.13 -	→	Number of bytes.

13.10 REGISTER EXAMINE

Data part	Value	Function
- link.06 -	16	Register examine.
- link.07 -	→	Register number.
- link.10 -	←	Returned data from ND-500.
- link.11 -		

13.11 REGISTER DEPOSIT

Data part	Value	Function
- link.06 -	17	Register deposit.
- link.07 -	→	Register number.
- link.10 -	→	Value for register deposit.
- link.11 -		

13.12 REGISTER READ

Data part	Value	Function
- link.06 -	20	Register read.
- link.07 -	→	First register number to write.
- link.10 -	→	Number of registers.
- link.11 -	→	Physical ND-100 address for
- link.12 -		registers returned from ND-500.

13.13 REGISTER WRITE

Data part	Value	Function
- link.06 -	21	Register write.
- link.07 -	→	First register number to write.
- link.10 -	→	Number of registers.
- link.11 -	→	Physical ND-100 address for
- link.12 -		registers to write to ND-500.

13.14 START

Data part	Value	Function
- link.06 -	23	Start.
- link.07 -		
- link.10 -		

The start function is only returned as a monitor call or a trap. The function value is unchanged while the rest of the data part of the block indicates whether it is a monitor call or a trap.

13.15 MONITOR CALL

Data part	Value	Function
- link.06 -	23	Monitor call or trap.
- link.07 -	←	ND-500 P register returned.
- link.10 -		
- link.11 -	1	Monitor call.
- link.12 -	←	Number of parameters.
- link.13 -	←	Monitor call number.
| |		
- link.40 -	←	Address of first parameter.
- link.41 -		
- link.42 -	←	Address of second parameter.
- link.43 -		
| |		Space for 16 parameter addresses.
- link.100-	←	Value of first parameter
- link.101-		
- link.102-	←	Value of second parameter.
- link.103-		
		Space for 16 parameter addresses.

13.16 TRAP

Data part	Value	Function
- link.06 -	23	Monitor call or trap.
- link.07 -		
- link.10 -		
- link.11 -	2	Trap.
- link.12 -	←	Trapping P register.
- link.13 -		
- link.14 -	←	Restart P.
- link.15 -		
- link.16 -	←	Trap number.
- link.17 -	←	Varies depending on trap number.
- link.20 -		
- link.21 -		
- link.22 -		

13.17 RESTART AFTER MONITOR CALL

Data part	Value	Function
- link.06 -	24	Restart after monitor call.
- link.07 -	→	Cache control.
- link.10 -		
- link.11 -	→	0/1 → K (flag).
- link.12 -	→	Write back mask.
- link.13 -	→	Function value → I1.
- link.14 -		
| |		
- link.40 -	→	Address of first parameter.
- link.41 -		
- link.42 -	→	Address of second parameter.
- link.43 -		
| |		Space for 16 parameter addresses.
- link.100-	→	Value of first parameter.
- link.101-		
- link.102-	→	Value of second parameter.
- link.103-		
		Space for 16 parameter values.

13.18 RESTART AFTER TRAP

Data part	Value	Function
- link.06 -	25	Restart after trap.
- link.07 -		
- link.10 -		

13.19 PHYSICAL SEGMENT READ

Data part	Value	Function
- link.06 -	30	Physical segment read.
- link.07 -	→	Physical ND-500 address on segment.
- link.10 -		
- link.11 -	→	Physical ND-100 address.
- link.12 -		
- link.13 -	→	Number of bytes.
- link.14 -	→	Physical segment number.

13.20 PHYSICAL SEGMENT WRITE

Data part	Value	Function
- link.06 -	31	Physical segment write.
- link.07 -	→	Physical ND-500 address on segment.
- link.10 -		
- link.11 -	→	Physical ND-100 address.
- link.12 -		
- link.13 -	→	Number of bytes.
- link.14 -	→	Physical segment number.

13.21 LOGICAL INSTRUCTION MEMORY READ

Data part	Value	Function
- link.06 -	34	Logical instruction memory read.
- link.07 -	→	Logical ND-500 address.
- link.10 -		
- link.11 -	→	Physical ND-100 address.
- link.12 -		
- link.13 -	→	Number of bytes.

13.22 LOGICAL INSTRUCTION MEMORY WRITE

Data part	Value	Function
- link.06 -	35	Logical instruction memory write.
- link.07 -	→	Logical ND-500 address.
- link.10 -		
- link.11 -	→	Physical ND-100 address.
- link.12 -		
- link.13 -	→	Number of bytes.

13.23 PROGRAMMED TRAP

Data part	Value	Function
- link.06 -	42	Programmed trap.

13.24 HISTOGRAM READ

Data part	Value	Function
- link.06 -	44	Histogram read.
- link.07 -	←	Returned P register from ND-500.
- link.10 -		
- link.11 -	←	Current process.

14 MICRO INSTRUCTION

The ND-500 micro instruction is a combination of the ND-500 mnemonic symbols, constants or defined symbols separated with a space. The micro instruction is terminated by ';' and may occupy several lines of symbols. Each line is limited to 80 characters. Characters on a line after '%' are taken as comments.

14.1 MNEMONIC SYMBOLS

The mnemonic symbols are direct functions or operator select except the symbols listed below. These require an octal number connected to the mnemonic symbol by '#' for desired operator select.

Mnemonic symbol	Number range	Function
A,AM / A,AL / A,A	0-37	Floating A operand select.
B,AM / B,AL / B,A	0-37	Floating B operand select.
D,AM / D,AL / D,AD	0-37	Floating destination select.
A,X / B,X / D,X	0-3	Index register select.
A,BM / B,BM	0-37	Bit mask generate.

14.2 CONSTANTS

Constants used in the micro program must be octal integers. The constants are either used in the short argument field or the long argument field, in the micro program address field or as micro program address modifier.

14.3 SHORT ARGUMENT

Short argument is specified by one 16 bit integer. The value of the constant is placed in the short argument field during assembly (control store bits 15-0). During execution in the ND-500 the short argument is sign extended to 32 bits by A,XD,SARG and AB,DPARG.

14.4 LONG ARGUMENT

Long argument is specified by two 16 bits integers separated by ','. The value of the constant is placed in the long argument field during assembly (control store bits 31-0).

14.5 MICRO PROGRAM ADDRESS

Micro program address may be selected either by reference to a label or by specifying a long argument where the most significant part is taken as a micro program address (control store bits 28-16). Reference to a label will cause the value of the label to be placed in the micro program address field.

14.6 MICRO PROGRAM ADDRESS MODIFIER

The micro program address may be modified by a 12 bit integer terminated by '/' located as the first element of a micro instruction. Current micro program address is set equal to integer specified.

> Note that integers occuring as the first element of a micro instruction are taken as micro program modifiers.

14.7 DEFINED SYMBOLS

Labels are defined by alpha numeric characters terminated by ':' . The label must be located as the first symbol of a micro instruction. The value of the label is the current control store address. The first 12 characters are significant. Reference to a label will cause the value of the label to be placed in the micro program address field (control store bits 28-16).

14.8 THE ASSEMBLER

The assembler works on mass storage files and may handle 25 input files and 5 output files. Output files required by the assembler are marked by '*'. In addition, the user running the assembler also requires the mnemonic symbol file (N500-MNE-SYMBOLS:SYMB) and the mnemonic value file (N500-MNE-VALUES:DATA). The input and output files are :SYMB type, except the object file which is :DATA type.

The output files with content are:
* Undefined symbols list-file contains all undefined symbols.
* Error list-file contains errors detected during assembly.
* Object file contains input to control store. List-file contains symbolic list of the micro program with control store address. Unsorted label list-file contains all labels defined with coresponding micro program address. Octal list-file contains octal listing of the object file.

Example of running the ND-500 micro assembler:

```
@N500-8K-ASSEM
  ND-500 MICRO-CODE ASSEMBLER 1.7  1981:07:01

  INPUT SEQUENCE TERMINATED TYPING <CR>
GIVE FILENAME OF ENTRY NO. 1  : N500-MICRO-01-00:SYMB
GIVE FILENAME OF ENTRY NO. 2  : N500-MICRO-02-00:SYMB
GIVE FILENAME OF ENTRY NO. 3  :

UNDEFINED SYMBOLS LIST-FILE   : N500-MICRO-UDEFV:SYMB
ERROR LIST-FILE               : N500-MICRO-ERROR:SYMB
```

```
OBJECT FILE                    : N500-MICRO-OBJEC:DATA
LIST-FILE                      : N500-MICRO-SLIST:SYMB
UNSORTED LABEL LIST-FILE       : N500-MICRO-USORT:SYMB
OCTAL LIST-FILE                : N500-MICRO-OCTAL:SYMB

LENGTH OF MICROPROGRAM IN KILOWORDS (EACH 144 BITS): 8

   100 WORDS ASSEMBLED
   200 WORDS ASSEMBLED

   100 ITEMS IN UDFV TABLE RECOGNIZED
     0 DIAGNOSTICS HAS BEEN DETECTED

ALL PROGRAM FUNCTIONS TERMINATED

@
```

14.9 ERROR MESSAGES FROM THE MICRO ASSEMBLER

The error messages from the ND-500 micro assembler give the micro program address where an error is detected, ERROR AT CLC <octal number>, and type of error. Below, the different error messages are listed along with a short explanation of each. At end of the assembly, the number of errors detected is written on both the error file and the terminal.

ERROR AT CLC XXXXXXB
CURRENT LOCATION COUNTER IS AT UPPER LIMIT
Moving outside address space. This means that the upper control store address is reached for this size control store.

ERROR AT CLC XXXXXXB
BLOCK NUMBER TOO LARGE:
Modified micro program address is the outside address space for this size control store.

ERROR AT CLC 000000B
YEAR-MONTH-DAY IDENTIFIER POSITION
(WORD 0, BITS 0-15) ALREADY OCCUPIED BY CODE
Contents of micro program address 0 is used as an identifier assigning year (bits 15-9), mounth (bits 8-5) and date (bits 4-0) of assembling. The space is already occupied.

ERROR AT CLC XXXXXXB
ILLEGAL CHARACTER IN ROUTINE "TRANSFORM"
Not an octal number at source file.

ERROR AT CLC XXXXXXB
TRANSFORM OVERFLOW
Overflow in convert to octal. Too large octal number at source file.

ERROR AT CLC XXXXXXB
ILLEGAL FORMAT ON CLC MODIFIER
Illegal format when modifying the micro program address.

ERROR AT CLC XXXXXXB
CLC MODIFIER ERROR
Error in modifying the micro program address.

ERROR AT CLC XXXXXXB
TOO MANY MNEMONICS BETWEEN SEMICOLONS
The input buffer containing source code for assembling is full.

ERROR AT CLC XXXXXXB
TOO LONG MNEMONIC
More than 20 characters in a mnemonic symbol.

ERROR AT CLC XXXXXXB
ATTEMPT TO WRITE ON FORMER ENTRY
Attempt to write into a previously used micro program address.

ERROR AT CLC XXXXXXB
OR-ING REJECTED DUE TO OVERLAPPING OF MNE-VALUES
Error occured because same bits should be set for combination of mnemonic symbols or arguments. The rest of the micro instruction is not assembled.

FATAL ERROR!!!! OVERFLOW IN DFV ARRAY (DFVPACK)
No more space for defined symbols.

ERROR AT CLC XXXXXXB
ILLEGAL FORMAT ON DFV
Error in area containing defined symbols.

ERROR AT CLC XXXXXXB
MNEMONIC USED AS LABEL:
Labels equal to mnemonic symbols not allowed.

ERROR AT CLC XXXXXXB
ALREADY DEFINED:
Label already defined.

15 ND-500 MNEMONIC SYMBOLS

```
A,-1          A-OPERAND IS FLOATING -1.0 CONSTANT
A,A           A-OPERAND IS DOUBLE FLOATING REGISTER
A,AL          A-OPERAND IS LEAST FLOATING REGISTER
A,AM          A-OPERAND IS MOST FLOATING REGISTER
A,B           A-OPERAND IS B REGISTER
A,BM          A-OPERAND IS DECODED BIT MASK BIT
A,BMR         A-OPERAND IS DECODED BIT MASK FROM BIT MASK REGISTER
A,DATA        A-OPERAND IS MEMORY DATA (DATA IN REGISTER)
A,L           A-OPERAND IS L REGISTER
A,ONE         A-OPERAND IS FLOATING 1.0 CONSTANT
A,P           A-OPERAND IS P REGISTER
A,R           A-OPERAND IS R REGISTER
A,SP          A-OPERAND IS SAVED P
A,THA         A-OPERAND IS TRAP HANDLER REGISTER
A,TOS         A-OPERAND IS TOP OF STACK REGISTER
A,X           A-OPERAND IS INDEX REGISTER
A,XD,BMR      A-OPERAND IS BIT MASK REGISTER (NOT DECODED)
A,XD,CAR      A-OPERAND IS COMPUTED ADDRESS REGISTER
A,XD,CONST    A-OPERAND IS INSTRUCTION CONSTANT REGISTER
A,XD,CSBRK    A-OPERAND IS CONTROL STORE BREAK REGISTER
A,XD,CSCNT    A-OPERAND IS CONTROL STORE CONTROL REGISTER
A,XD,CSWA     A-OPERAND IS CONTROL STORE WRITE ADDRESS REGISTER
A,XD,DCINHLL  A-OPERAND IS DATA CACHE INHIBIT LOWER LIMIT
A,XD,DCINHLU  A-OPERAND IS DATA CACHE INHIBIT UPPER LIMIT
A,XD,DHXA     A-OPERAND IS DATA TSB ADDRESS
A,XD,DISP     A-OPERAND IS DISP REGISTER
A,XD,DLADDR   A-OPERAND IS MM DATA LOGICAL ADDRESS
A,XD,DMSTS    A-OPERAND IS MM DATA STATUS
A,XD,DRADDR   A-OPERAND IS MM DATA REAL ADDRESS
A,XD,DRADDRL  A-OPERAND IS DATA REAL ADDRESS LEAST SIGNIFICANT
A,XD,DRADDRM  A-OPERAND IS DATA REAL ADDRESS MOST SIGNIFICANT
A,XD,DSCRF    A-OPERAND IS MM DATA SCRATCH FILE
A,XD,DSTS0    A-OPERAND IS DATA MEMORY STATUS REG 0
A,XD,DSTS1    A-OPERAND IS DATA MEMORY STATUS REG 1
A,XD,DSTS2    A-OPERAND IS DATA MEMORY STATUS REG 2
A,XD,DUPL     A-OPERAND IS DATA UPPER PAGE LIMIT
A,XD,DWIPGU   A-OPERAND IS MM DATA WIP/PGU BROAD
A,XD,DZPA     A-OPERAND IS DATA ZERO POINT ADJUST REGISTER
A,XD,HL       A-OPERAND IS HIGHER LIMIT REGISTER
A,XD,ICINHLL  A-OPERAND IS INSTRUCTION CACHE INHIBIT LOWER LIMIT
A,XD,ICINHLU  A-OPERAND IS INSTRUCTION CACHE INHIBIT UPPER LIMIT
A,XD,IDAT     A-OPERAND IS INSTRUCTION MEMORY DATA
A,XD,IHXA     A-OPERAND IS INSTRUCTION TSB ADDRESS
A,XD,ILADDR   A-OPERAND IS MM INSTRUCTION LOGICAL ADDRESS
A,XD,IMSTS    A-OPERAND IS MM INSTRUCTION STATUS
A,XD,INDXC    A-OPERAND IS INDEX COUNTER NO.#
A,XD,IODIN    A-OPERAND IS IO DATA IN REGISTER
A,XD,IODOUT   A-OPERAND IS IO DATA OUT REGISTER
A,XD,IRADDR   A-OPERAND IS MM INSTRUCTION REAL ADDRESS
A,XD,IRADDRL  A-OPERAND IS INSTRUCTION REAL ADDRESS LEAST SIGN.
A,XD,IRADDRM  A-OPERAND IS INSTRUCTION REAL ADDRESS MOST SIGN.
```

A,XD,ISCRF	A-OPERAND IS MM INSTRUCTION SCRATCH FILE
A,XD,ISTS0	A-OPERAND IS INSTRUCTION MEMEMORY STATUS REG. 0
A,XD,ISTS1	A-OPERAND IS INSTRUCTION MEMEMORY STATUS REG. 1
A,XD,ISTS2	A-OPERAND IS INSTRUCTION MEMEMORY STATUS REG. 2
A,XD,IUPL	A-OPERAND IS INSTRUCTION UPPER PAGE LIMIT
A,XD,IWIPGU	A-OPERAND IS MM INSTRUCTION WIP/PGU BROAD
A,XD,IZPA	A-OPERAND IS INSTRUCTION ZERO POINT ADJUST REGISTER
A,XD,LARG	A-OPERAND IS LONG ARGUMENT (32 BITS)
A,XD,LC	A-OPERAND IS LOOP COUNTER
A,XD,LL	A-OPERAND IS LOWER LIMIT REGISTER
A,XD,MISTAT	A-OPERAND IS MICRO STATUS REGISTER
A,XD,MMOD	A-OPERAND IS MEMORY MODUS REGISTER
A,XD,NBY	NO OF BYTES IN LAST MEMORY REFERANCE
A,XD,PSTAT	A-OPERAND IS PREFETCH STATUS REGISTER
A,XD,S1	A-OPERAND IS STATUS REGISTER ONE
A,XD,S2	A-OPERAND IS STATUS REGISTER TWO
A,XD,SARG	A-OPERAND IS SHORT ARGUMENT (16 BITS)
A,XD,SHC	A-OPERAND IS SHIFT COUNT REGISTER
A,XD,TE1	A-OPERAND IS LOCAL TRAP ENABLE REGISTER
A,XD,TRAPCSA	A-OPERAND IS SAVED CSA WHEN TRAP
A,XD,TRAPINF	A-OPERAND IS TRAP INFORMATION
A,ZRO	A-OPERAND IS FLOATING 0.0 CONSTANT
AA+AB	ADDRESS A-OPERAND PLUS ADDRESS B-OPERAND
AA,DP1	ADDRESS A-OPERAND IS DP1 REGISTER
AA,DP2	ADDRESS A-OPERAND IS DP2 REGISTER
AA,EA1	ADDRESS A-OPERAND IS EFFECTIVE ADDRESS 1 REGISTER
AA,EA2	ADDRESS A-OPERAND IS EFFECTIVE ADDRESS 2 REGISTER
AB,1/8IX	ADDRESS B-OPERAND IS INDEX REGISTER SCALED BY 1/8
AB,2IX	ADDRESS B-OPERAND IS INDEX REGISTER SCALED BY 2
AB,4IX	ADDRESS B-OPERAND IS INDEX REGISTER SCALED BY 4
AB,8IX	ADDRESS B-OPERAND IS INDEX REGISTER SCALED BY 8
AB,B	ADDRESS B-OPERAND IS B REGISTER
AB,DPARG	ADDRESS B-OPERAND IS ARGUMENT (SIGN EXTENDED)
AB,IX	ADDRESS B-OPERAND IS INDEX REGISTER SCALED BY 1
AB,ORAB	ADDRESS B-OPERAND IS OR-LOGIC CONTROLLED
AB,PC	ADDRESS B-OPERAND IS P REGISTER
AB,R	ADDRESS B-OPERAND IS R REGISTER
AD,ILC	DESTINATION IS INSTRUCTION LOOK AHEAD ADDRESS COUNTER
AD,NPC	DESTINATION IS NEXT PROGRAM COUNTER
AD,PC	DESTINATION IS PROGRAM COUNTER
ALTMOD	SELECT "APT/PT" ACCORDING TO FIRST OPERAND
ALU,A+1	A-OPERAND INCREMENT
ALU,A+A	A-OPERAND PLUS A-OPERAND
ALU,A+A+1	A-OPERAND PLUS A-OPERAND PLUS ONE
ALU,A+B	A-OPERAND PLUS B-OPERAND
ALU,A+B+1	A-OPERAND PLUS B-OPERAND PLUS ONE
ALU,A+B+C	A-OPERAND PLUS B-OPERAND PLUS CARRY
ALU,A-1	A-OPERAND DECREMENT
ALU,A-B	A-OPERAND MINUS B-OPERAND
ALU,A-B-1	A-OPERAND MINUS B-OPERAND MINUS 1
ALU,A-B-1+C	A-OPERAND MINUS B-OPERAND MINUS 1 PLUS C
ALU,ADIR	A-OPERAND DIRECT THROUGH ALU
ALU,ADIRC	A-OPERAND COMPLEMENTED THROUGH ALU
ALU,AND	AND-FUNCTION OF A-OPERAND AND B-OPERAND
ALU,ANDCA	AND-FUNCTION OF A-OPERAND COMPLEMENTED AND B-OPERAND

ALU,ANDCB	AND-FUNCTION OF A-OPERAND AND B-OPERAND COMPLEMENTED
ALU,BDIR	B-OPERAND DIRECT THROUGH ALU
ALU,BDIRC	B-OPERAND COMPLEMENTED THROUGH ALU
ALU,FONE	FORCED ONE ALU OUTPUT
ALU,FZRO	FORCED ZERO ALU OUTPUT
ALU,NAND	NAND-FUNCTION OF A-OPERAND AND B-OPERAND
ALU,NOR	NOR-FUNCTION OF A-OPERAND AND B-OPERAND
ALU,OR	OR-FUNCTION OF A-OPERAND AND B-OPERAND
ALU,ORCA	OR-FUNCTION OF A-OPERAND COMPLEMENTED AND B-OPERAND
ALU,ORCB	OR-FUNTION OF A-OPERAND AND B-OPERAND COMPLEMENTED
ALU,XNOR	EXCLUSIVE NOR-FUNCTION OF A-OPERAND AND B-OPERAND
ALU,XOR	EXCLUSIVE OR-FUNCTION OF A-OPERAND AND B-OPERAND
B,-1	B-OPERAND IS FLOATING -1.0 CONSTANT
B,A	B-OPERAND IS DOUBLE FLOATING REGISTER
B,AL	B-OPERAND IS LEAST FLOATING REGISTER
B,AM	B-OPERAND IS MOST FLOATING REGISTER
B,BM	B-OPERAND IS DECODED BIT MASK BIT
B,BMR	B-OPERAND IS DECODED BIT MASK FROM BIT MASK REGISTER
B,DATA	B-OPERAND IS MEMORY DATA
B,DATA2	B-OPERAND IS EXTRA MEMORY DATA REGISTER
B,EA1	B-OPERAND IS EA1 (EFFECTIVE ADDRESS 1.ZERO POLARITY)
B,EA2	B-OPERAND IS EA2 (EFFECTIVE ADDRESS 1.ZERO POLARITY)
B,ONE	B-OPERAND IS FLOATING 1.0 CONSTANT
B,THA	B-OPERAND IS TRAP HANDLER REGISTER
B,TOS	B-OPERAND IS TOP OF STACK REGISTER
B,X	B-OPERAND IS INDEX REGISTER
B,XRESL	B-OPERAND IS EXTERNAL (FLOATING) RESULT LEAST
B,XRESM	B-OPERAND IS EXTERNAL (FLOATING) RESULT MOST
B,ZRO	B-OPERAND IS FLOATING 0.0 CONSTANT
BCD	BCD TYPE
BCD,ADD	BCD ADD WITHOUT ROUNDING
BCD,ADDR	BCD ADD WITH ROUNDING
BCD,BCDC	BCD TO BINARY CONVERT
BCD,BINC	BINARY TO BCD CONVERT
BCD,COMP	BCD COMPARE
BCD,DATA	DATA IN OR DATA OUT FROM BCD ARITHMETIC
BCD,DESCA	BCD DESCRIPTOR
BCD,MPY	BCD MULTIPLY WITHOUT ROUNDING
BCD,MPYR	BCD MULTIPLY WITH ROUNDING
BCD,PACK	PACK ASCII TO BCD WITHOUT ROUNDING
BCD,PACKR	PACK ASCII TO BCD WITH ROUNDING
BCD,SHIFT	BCD SHIFT WITHOUT ROUNDING
BCD,SHIFTR	BCD SHIFT WITH ROUNDING
BCD,SUB	BCD SUBTRACT WITHOUT ROUNDING
BCD,SUBR	BCD SUBTRACT WITH ROUNDING
BCD,UPACK	UNPACK BCD TO ASCII WITHOUT ROUNDING
BCD,UPACKR	UNPACK BCD TO ASCII WITH ROUNDING
BYTH	BYTE TO HALF-WORD SIGN EXTENSION
BYTW	BYTE TO WORD SIGN EXTENSION
C,ALU	CONDITIONAL ALU ENABLE
C,MEM	CONDITIONAL MEMORY REFERENCE ENABLE
C,SEQ	CONDITIONAL SEQUENCE ENABLE
C,SEQD	DELAYED CONDITIONAL SEQUENCE ENABLE
COND,CNZ	TEST COND IS CARRY NOT ZERO
COND,CONOP	TEST COND IS CONOP

```
COND,CRY        TEST COND IS CARRY
COND,DATOP      TEST COND IS DATOP
COND,ENTER      TEST COND IS NEXT INSTRUCTION 'ENTER'
COND,ICDRY      TEST COND IS INSTRUCTION MEMORY READY
COND,K          TEST COND IS K (FLAG)
COND,LCZ        TEST COND IS LOOP COUNTER ZERO
COND,MBO        TEST COND IS MICRO BCD OVERFLOW
COND,MCNZ       TEST COND IS MICRO CARRY NOT ZERO
COND,MCRY       TEST COND IS MICRO CARRY
COND,MDZ        TEST COND IS DIVIDE BY 0
COND,MFO        TEST COND IS MICRO FLOATING OVERFLOW
COND,MFS        TEST COND IS MICRO FLOATING SIGN
COND,MFU        TEST COND IS MICRO FLOAT UNDERFLOW
COND,MFUFO      TEST COND IS FLOATING OVER- OR UNDER-FLOW
COND,MIVO       TEST COND IS MICRO BCD INVALID OPERATION
COND,MOVFL      TEST COND IS MICRO OVERFLOW
COND,MSEXO      TEST COND IS MICRO SIGN EXOR OVERFLOW
COND,MSGN       TEST COND IS MICRO SIGN
COND,MSORZ      TEST COND IS MICRO SIGN OR ZERO
COND,MZRO       TEST COND IS MICRO ZERO
COND,OVFL       TEST COND IS OVERFLOW
COND,PARITY     TEST CONDITION IS PARITY OF ALU OUTPUT
COND,PDONE      TEST COND IS PART DONE
COND,SAVC1      TEST COND IS SAVED CONDITION ONE
COND,SAVC2      TEST COND IS SAVED CONDITION TWO
COND,SGN        TEST COND IS SIGN
COND,SORZ       TEST COND IS SIGN OR ZERO
COND,TRAP       TEST COND IS TRAP
COND,ZRO        TEST COND IS ZERO
CSAVE           CONDITION SAVE (PUSH CONDITION RESULT)
D,AD            DEST IS FLOATING DOUBLE REGISTER
D,AL            DEST IS FLOATING LEAST REGISTER
D,AM            DEST IS FLOATING MOST REGISTER
D,ATRCLR        RESET ADDR. TRAP-OCCURED FLIP-FLOP
D,B             DEST IS B REGISTER
D,BMR           DEST IS BIT MASK REGISTER
D,CAR           DEST IS COMPUTED ADDRESS REGISTER
D,CONST         DEST IS CONSTANT REGISTER
D,CSBRK         DEST IS CONTROL STORE BREAK REGISTER
D,CSCNT         DEST IS CONTROL STORE CONTROL REGISTER
D,CSWA          DEST IS CONTROL STORE WRITE ADDRESS REGISTER
D,DADOM         DEST IS MM DATA ALT DOMR
D,DASEG         DEST IS MM DATA ALT. SEGM.
D,DATAIN        DEST IS DATA-IN HOLD REGISTER ON SLICE
D,DATAIN2       DEST IS EXTRA DATA-IN REGISTER ON SLICE
D,DCINHLL       DEST IS DATA CACHE INHIBIT LOWER LIMIT
D,DCINHLU       DEST IS DATA CACHE INHIBIT UPPER LIMIT
D,DCLCACH       DEST IS DATA CLEAR CACHE
D,DCON0         DEST IS DATA CONTROL REGISTER 0
D,DCON1         DEST IS DATA MEMORY CONTROL REGISTER 1
D,DCSEG         DEST IS MM DATA CURRENT SEGMENT
D,DDOMR         DEST IS MM DATA DOMAIN REGISTER
D,DMCNTR        DEST IS MM DATA CONTROL
D,DP            DEST IS DISPLACEMENT REGISTER
D,DPROCC        DEST IS MM DATA PROC. CONTROL REGISTER
```

D,DSCFA	DEST IS MM DATA SCRF ADDRESS
D,DSCRF	DEST IS MM DATA SCRATCH FILE
D,DSTSB	DEST IS MM DATA SEQUENTIAL TSB
D,DTSB	DEST IS MM DATA TSB-PAGE
D,DUPL	DEST IS DATA UPPER PAGE LIMIT
D,DWIPGU	DEST IS MM DATA WIP/PGU BROAD
D,DZPA	DEST IS DATA ZERO POINT ADJUST REGISTER
D,HL	DEST IS HIGHER LIMIT REGISTER
D,IADOM	DEST IS MM INSTRUCTION ALT. DOMR.
D,IASEG	DEST IS MM INSTRUCTION ALT. SEGM.
D,ICCLR	INDEX COUNTER CLEAR
D,ICINHLL	DEST IS INSTRUCTION CACHE INHIBIT LOWER LIMIT
D,ICINHLU	DEST IS INSTRUCTION CACHE INHIBIT UPPER LIMIT
D,ICLCACH	DEST IS INSTRUCTION CLEAR CACHE
D,ICON0	DEST IS INSTRUCTION CONTROL REGISTER 0
D,ICON1	DEST IS INSTRUCTION MEMORY CONTROL REGISTER 1
D,ICSEG	DEST IS MM INSTRUCTION CURRENT SEGMENT
D,IDAT	DEST IS INSTRUCTION MEMORY DATA
D,IDOMR	DEST IS MM INSTRUCTION DOMAIN REGISTER
D,IMCNTR	DEST IS MM INSTRUCTION CONTROL
D,IODOUT	DEST IS IO DATA OUT REGISTER
D,IPROCC	DEST IS MM INSTRUCTION PROC. CONTROL REGISTER
D,ISCFA	DEST IS MM INSTRUCTION SCRF ADDRESS
D,ISCRF	DEST IS MM INSTRUCTION SCRATCH FILE
D,ISTSB	DEST IS MM INSTRUCTION SEQUENTIAL TSB
D,ITSB	DEST IS MM INSTRUCTION TSB PAGE
D,IUPL	DEST IS INSTRUCTION UPPER PAGE LIMIT
D,IWIPGU	DEST IS MM INSTRUCTION WIP/PGU BROAD
D,IZPA	DEST IS INSTRUCTION ZERO POINT ADJUST
D,L	DEST IS LINK REGISTER
D,LC	DEST IS LOOP COUNTER
D,LL	DEST IS LOWER LIMIT REGISTER
D,MCC	DEST IS MICRO CYCLE COUNTER
D,MMOD	DEST IS MEMORY MODUS REGISTER
D,R	DEST IS RECORD REGISTER
D,S1	DEST IS STATUS REGISTER 1
D,S2	DEST IS STATUS REGISTER 2
D,SETLIM	SET LIMIT BONDS FOR HL-LL
D,SHC	DEST IS SHIFT COUNT REGISTER
D,TAG	DEST IS TAG REGISTER
D,TE1	DEST IS LOCAL TRAP ENABLE REGISTER
D,THA	DEST IS TRAP HANDLER REG
D,TOS	DEST IS TOP OF STACK
D,TRACECLR	TRAP SYSTEM CLEAR TRACE BITS
D,TRAPCLR	TRAP SYSTEM CLEAR
D,TRAPONOFF	TRAP SYSTEM ON/OFF
D,TSBIND	SET AND RESET TSB INDIC
D,X	DEST IS INDEX REGISTER
D,XST1	DEST IS HARDWARE CONTROLLED BITS IN STATUS
DOUBLE	DOUBLE FLOATING OPERATION
EA1INH	EFFECTIVE ADDRESS REGISTER 1 CLOCK INHIBIT
EA2INH	EFFECTIVE ADDRESS REGISTER 2 CLOCK INHIBIT
EX,APIMUL	LAST FLOATING DIVIDE STEP SA*SP(INVERTED) TO CPU
EX,APIMULA	THIRD FLOATING DIVIDE STEP SA*SP(INVERTED) TO SA
EX,APMULA	SA*SP TO SA

EX,ARMULA	FIRST FLOATING DIVIDE STEP A*1/B' TO SA
EX,ASUM	FLOATING ADD SA+B TO CPU
EX,ASUMA	FLOATING ADD SA+B TO SA
EX,BRMULP	SECOND FLOATING DIVIDE STEP B*1/B' TO SP
EX,COMPARE	COMPARE A AND B
EX,CTF	CONVERT INTEGER TO FLOATING
EX,CTI	CONVERT FLOATING TO INTEGER TRUNCATED
EX,CTIR	CONVERT FLOATING TO INTEGER WITH ROUNDING
EX,DIFF	FLOATING SUBTRACT A-B TO CPU
EX,DTOFR	CONVERT DOUBLE TO SINGLE FLOATING WITH ROUNDING
EX,INT	INTEGER PART IN FLOATING FORMAT TRUNCATED
EX,INTR	INTEGER PART IN FLOATING FORMAT ROUNDED
EX,MUL	MULTIPLY A*B TO CPU
EX,MULA	MULTIPLY AND SAVE A*B TO SA
EX,PPIMULP	FOURTH FLOATING DIVIDE STEP SP*SP(INVERTED) TO SP
EX,SHA	SHIFT ARITHMETIC
EX,SHL	SHIFT LOGICAL
EX,SHR	SHIFT ROTATIONAL
EX,SUM	FLOATING ADD A+B TO CPU
EX,TORMULA	A * 1/B ⟹ SA
EX,UCTF	UNSIGNED CONVERT INTEGER TO FLOATING
EX,UMUL	UNSIGNED MULTIPLY A*B TO CPU
F,A+1	A-OPERAND INCREMENT
F,A+A	A-OPERAND PLUS A-OPERAND
F,A+A+1	A-OPERAND PLUS A-OPERAND PLUS 1
F,A+B	A-OPERAND PLUS B-OPERAND
F,A+B+1	A-OPERAND PLUS B-OPERAND PLUS 1
F,A+B+C	A-OPERAND PLUS B-OPERAND PLUS C
F,A-1	A-OPERAND DECREMENT
F,A-B	A-OPERAND MINUS B-OPERAND
F,A-B-1	A-OPERAND MINUS B-OPERAND MINUS 1
F,A-B-1+C	A-OPERAND PLUS B-OPERAND MINUS 1 PLUS C
F,ADIR	A-OPERAND DIRECT THROUGH ALU
F,ADIRC	A-OPERAND COMPLEMENTED THROUGH ALU
F,AND	AND-FUNC OF A- AND B-OPERAND
F,ANDCA	AND-FUNC OF A-OPERAND COMPLEMENTED AND B-OPERAND
F,ANDCB	AND-FUNC OF A-OPERAND AND B-OPERAND COMPLEMENTED
F,BDIR	B-OPERAND DIRECT THROUGH ALU
F,BDIRC	B-OPERAND COMPLEMENTED THROUGH ALU
F,FONE	FORCED ONE ALU OUTPUT
F,FZRO	FORCED ZERO ALU OUTPUT
F,HBRET	RETURN TO HARDWARE BRANCH REGISTER
F,JMP	JUMP ABSOLUTE
F,JMPCAR	JUMP TO ADDRESS TAKEN FROM CAR
F,JMPMAP	JUMP TO MAP ADDRESS. I.E., START OF NEXT INSTRUCRION
F,JMPNS	JUMP ASOLUTE. STACK CONTROL NOT INCLUDED
F,JMPREL	JUMP RELATIVE TO M.P.C. DISPLACEMENT-1 IN CAR
F,JMPSTK	JUMP STACK TO STACK ADDRESS
F,JMPWA	JUMP TO WRITE ADDRESS (WA)
F,JSR	JUMP TO SUBROUTINE ABSOLUTE
F,JSRCAR	JUMP TO SUBROUTINE IN CAR
F,JSRMAP	JUMP TO SUBROUTINE IN MAP ADDRESS
F,JSRREL	JUMP TO SUBROUTINE RELATIVE. DISPLACEMENT-1 IN CAR
F,JSRSTK	JUMP TO SUBROUTINE IN STACK
F,JSRWA	JUMP TO SUBROUTINE IN WRITE ADDRESS (WA)

F,LOAD	LOAD STACK
F,NAND	NAND-FUNC OF A- AND B-OPERAND
F,NEXT	NEXT MICROINSTRUCTION
F,NEXTNS	NEXT. STACK CONTROL NOT INCLUDED
F,NOPOPRET	RETURN WITH HOLD CONTROL TO STACK
F,NOR	NOR FUNC OF A- AND B-OPERAND
F,OR	OR FUNC OF A- AND B-OPERAND
F,ORCA	OR FUNC OF A-OPERAND COMPLEMENTED AND B-OPERAND
F,ORCB	OR FUNC OF A-OPERAND AND B-OPERAND COMPLEMENTED
F,POP	POP STACK
F,POPRET	RETURN FROM SUBROUTINE
F,PUSH	PUSH STACK
F,REP	REPEAT CURRENT MICROINSTRUCTION
F,XNOR	EXCLUSIVE NOR FUNC OF A- AND B-OPERAND
F,XOR	EXCLUSIVE OR FUNC OF A- AND B-OPERAND
FAST	FAST CYCLE
HBRET	RETURN TO HARDWARE BRANCH REGISTER
HWTW	HALFWORD TO WORD SIGN EXTENSION
IADJ	INDEX ADJUST (INCREMENT INDEX COUNTER)
IFMEM	IF MEMORY THEN....
IFT	IF CONDITION TRUE THEN....
IX0	ADDRESS B-OPERAND IS INDEX REGISTER 1
IX1	ADDRESS B-OPERAND IS INDEX REGISTER 2
IX2	ADDRESS B-OPERAND IS INDEX REGISTER 3
IX3	ADDRESS B-OPERAND IS INDEX REGISTER 4
JMP	JUMP ABSOLUTE
JMPCAR	JUMP TO ADDRESS TAKEN FROM CAR
JMPMAP	JUMP TO MAP ADDRESS. I.E., START OF NEXT INSTRUCTION
JMPNS	JUMP ABSOLUTE. STACK CONTROL NOT INCLUDED
JMPREL	JUMP RELATIVE TO M.P.C. DISPLACEMENT IN CAR
JMPSTK	JUMP TO STACK ADDRESS
JMPWA	JUMP TO WRITE ADDRESS (WA)
JSR	JUMP TO SUBROUTINE ABSOLUTE
JSRCAR	JUMP TO SUBROUTINE IN CAR
JSRMAP	JUMP TO SUBROUTINE IN MAP ADDRESS
JSRREL	JUMP TO SUBROUTINE RELATIVE. DISPLACEMET - 1 IN CAR
JSRSTK	JUMP TO SUBROUTINE IN STACK
JSRWA	JUMP TO SUBROUTINE IN WRITE ADDRESS (WA)
K,1IFZ	SET K = 1 IF ZERO ALU OUTPUT. SET DR = 0
K,ONE	SET K = 1
K,ZRO	SET K = 0
LCDECR	LOOP COUNTER DECREMENT
LOAD	LOAD STACK
MEM,MV1	MOVE ONE WORD IN DATA MEMORY
MEM,MV2	MOVE TWO WORDS IN DATA MEMORY
MEM,MV3	MOVE THREE WORDS IN DATA MEMORY
MEM,MV4	MOVE FOUR WORDS IN DATA MEMORY
MEM,RD	READ MEMORY WITH OR'ED TYPE
MEM,RD1	READ ONE BYTE FROM DATA MEMORY
MEM,RD2	READ TWO BYTES FROM DATA MEMORY
MEM,RD3	READ THREE BYTES FROM DATA MEMORY
MEM,RD4	READ FOUR BYTES FROM DATA MEMORY
MEM,WR	WRITE MEMORY WITH OR'ED TYPE
MEM,WR1	WRITE ONE BYTE TO DATA MEMORY
MEM,WR2	WRITE TWO BYTES TO DATA MEMORY

MEM,WR3	WRITE THREE BYTES TO DATA MEMORY
MEM,WR4	WRITE FOUR BYTES TO DATA MEMORY
MV,DINTOD	DATA-IN TO DATA OUT
MV,DTODP	DATA-IN TO DP
MV,DTOXBL	DATA IN TO FLOATING B-BUS LEAST SIGNICANT HALF
MV,DTOXBM	DATA IN TO FLOATING B-BUS MOST SIGNICANT HALF
NADL	NOT ADDRESS LATCH. USE ADDR ARITH WITHOUT MEM REF
NEXT	NEXT MICRO INSTRUCTION
NEXTNS	NEXT. STACK CONTROL NOT INCLUDED
NOPOPRET	RETURN WITH HOLD
NPCSEL	SELECT NPC TO L REGISTER
NSEXCONV	NOT SIGN EXTENSION WITH B,XRESL
ORA	OR A-OPERAND ACCORDING TO INSTRUCTION
ORB	OR B-OPERAND ACCORDING TO INSTRUCTION
ORD	OR DEST ACCORDING TO INSTRUCTION
ORT	OR DATATYPE ACCORDING TO INSTRUCTION
PASSAA	PASS ADDRESS A-OPERAND THRUOGH ADDRESS ARITH
PASSAB	PASS ADDRESS B-OPERAND THROUGH ADDRESS ARITH
POP	POP STACK
POPRET	RETURN FROM SUBROUTINE
PRF,ACONT	CC ADDRESS ARITH CONTINUE
PRF,CEOPF	END-OF-OPERATION IF FALSE
PRF,CEOPT	END-OF-OPERATION IF TRUE
PRF,CLEAR	PREFETCH CLEAR
PRF,EOP	END-OF-OPERATION.
PRF,FADC	FETCH GENERAL OPERAND
PRF,FARG	CC FETCH ARGUMENT
PRF,FOPC	CC FETCH OPCODE
PRF,FOPR	CC FETCH OPERAND
PRF,ISAMP	INTERRUPT SAMPLE
PRF,PCONT	PREFETCH CONTINUE
PRF,START	PREFETCH START
PRF,WFIN	WAIT FOR PREFETCH FINISHED (PREVIOUS)
PUSH	PUSH STACK (THE SEQUENCER STACK)
REP	REPEAT CURRENT MICROINSTRUCTION
SCRB	SELECT SCRATCH B-BLOCK AS A-OPERAND
SET	SET CONDITION SELECT
SHARG	SHIFT COUNTER FROM SHORT ARGUMENT FIELD
SHCFR	SHIFT COUNT FROM SHIFT COUNT REGISTER
SINGLE	SINGLE FLOATING OPERATION
SLOW1	SLOW CYCLE 1 (STRETCH HIGH AND LOW STATE)
SLOW2	SLOW CYCLE 2 (STRETCH LOW STATE)
SPAREBIT	SP(OLD PROGRAM COUNTER) CLOCK INHIBIT
ST,SAVA	SAVE STATUS FROM ALU OPERATION
ST,SAVAC	SAVE STATUS FROM ALU OPERATION IN COMPARE
ST,SAVB	SAVE STATUS FROM BCD OPERATION
ST,SAVF	SAVE STATUS FROM FLOATING OPERATION
TYP,BY	DATA TYPE IS BYTE
TYP,F	DATA TYPE IS FLOATING
TYP,HW	DATA TYPE IS HALFWORD
TYP,W	DATA TYPE IS WORD
UNLOCK	UNLOCK INTERFACE TO ND-100
W,EXT	WAIT FOR EXTERNAL ARITHMETIC
W,IO	WAIT FOR IO (INPUT-OUTPUT)
W,IOEM	WAIT FOR IO/EXT/MEM

W,IOEPM	WAIT FOR IO/EXT/PMEM
W,MEM	WAIT FOR MEMORY
W,PMEM	WAIT FOR PREVIOUS MEMORY CYCLE
W,XD	WAIT FOR MASTER CLEAR
XDMOV	XD-BUS MOVE (NO ALU TRANSFER)
XRES	ENABLE EXTERNAL RESULT.

16 ND-500 USER INSTRUCTIONS

Some instructions in the ND-500 are available for user written micro code. This means that an instruction code has an entry in the ND-500 micro program but is not used. 'Not used' means that the instruction generates illegal instruction code. These instructions may be used for special micro code to implement new functions. Some of the instruction codes are used in the micro program version containing the ND-500 Area Processing Instruction set. The instructions available are listed below. The instructions available may be divided into three different groups, depending on prefetch and operand decoding. A general description of the different types of instructions is also given. The instructions are listed with instruction code, default data data type for the operands and the entry point in the micro program.

The space available for user written micro code, depends on the micro program version. New contents may be placed in the upper part of the writable control store. A general rule is that the area free for user written micro code be empty or contain only a jump to micro program address 453. For future micro program versions, the area for user written micro code, as shown below, may be reduced without any notice. The space available for user written micro code is for the different micro program versions:

ND-500 standard micro program	version 105xx : 13000B to 17777B
ND-500 CX micro program	version 103xx : 13000B to 17777B
ND-500 AX option	version 104xx : 16000B to 17777B
ND-500 CX, AX option	version 106xx : 16000B to 17777B

16.1 CLASSIFICATION

Classification of the ND-500 user instructions is done depending on prefetch of operands and operand decoding.

Instruction group 1 : No operand is fetched.

Instruction group 2 : A memory operand is fetched.

Instruction group 3 : A general operand is fetched.

16.1.1 INSTRUCTION GROUP 1

The prefetch processor is for group 1 doing nothing, ie., no fetch of operands is done. The prefetch command PRF,EOP is required in the last micro instruction.

The following user instructions are available in group 1.

Instruction code	Instruction type	Micro program entry
236	W EXT	534
237	W EXT	535
177004	W EXT	536
177005	W EXT	537
177006	W EXT	540
177007	W EXT	541
177036	W EXT	542
177037	W EXT	543
177436	W EXT	544
177437	W EXT	545

16.1.2 INSTRUCTION GROUP 2

The prefetch processor is for group 2, fetching one memory operand. The only prefetch command to be used, is 'PRF,EOP' in the last micro-instruction of the instruction. This is for data type byte, halfword, word, and single floating point. For the data type double floating point, the prefetch command 'PRF,PCONT' is required after the most significant part of the double floating point operand is read, to switch to the least significant part of the operand. A memory request is required to get the least significant part of the operand. The command 'PRF,EOP' in the last microinstruction is also required.

The following user instructions are available in group 2.

Instruction code	Instruction type	Micro program entry
177460	By EXT <operand/r/BY>	752
177461	By EXT <operand/r/BY>	753
177462	By EXT <operand/r/BY>	754
177463	By EXT <operand/r/BY>	755
177464	By EXT <operand/r/BY>	756
177465	By EXT <operand/r/BY>	757
177466	By EXT <operand/r/BY>	760
177467	By EXT <operand/r/BY>	761

Instruction code	Instruction type	Micro program entry
177470	H EXT <operand/r/H>	762
177471	H EXT <operand/r/H>	763
177472	H EXT <operand/r/H>	764
177473	H EXT <operand/r/H>	765
177474	H EXT <operand/r/H>	766
177475	H EXT <operand/r/H>	767
177476	H EXT <operand/r/H>	770
177477	H EXT <operand/r/H>	771

Instruction code	Instruction type	Micro program entry
177477	W EXT <operand/r/W>	771
177500	W EXT <operand/r/W>	772
177501	W EXT <operand/r/W>	773
177502	W EXT <operand/r/W>	774
177503	W EXT <operand/r/W>	775
177504	W EXT <operand/r/W>	776
177505	W EXT <operand/r/W>	777
177506	W EXT <operand/r/W>	1000
177507	W EXT <operand/r/W>	1001

Instruction code	Instruction type	Micro program entry
177510	F EXT <operand/r/F>	1002
177511	F EXT <operand/r/F>	1003
177512	F EXT <operand/r/F>	1004
177513	F EXT <operand/r/F>	1005
177514	F EXT <operand/r/F>	1006
177515	F EXT <operand/r/F>	1007
177516	F EXT <operand/r/F>	1010
177517	F EXT <operand/r/F>	1011

Instruction code	Instruction type	Micro program entry
177520	D EXT <operand/r/D>	1012
177521	D EXT <operand/r/D>	1013
177522	D EXT <operand/r/D>	1014
177523	D EXT <operand/r/D>	1015
177524	D EXT <operand/r/D>	1016
177525	D EXT <operand/r/D>	1017
177526	D EXT <operand/r/D>	1020
177527	D EXT <operand/r/D>	1021

16.1.3 INSTRUCTION GROUP 3

The prefetch processor is for group 3, fetching one general operand, either a constant from program area, a register or a memory operand. The only prefetch command to be used, is 'PRF,EOP' in the last micro-instruction for the instruction. This is for data type byte, halfword, word, and single floating point. For the data type double floating point, the prefetch command 'PRF,PCONT' is required after the most significant part of the double floating point operand is read to switch to the least significant part of the operand. A memory request is required to get the least significant part of the operand when the operand is located in data memory. The command 'PRF,EOP' in the last microinstruction is also required.

The following user instructions are available in group 3.

Instruction code	Instruction type	Micro program entry
177300 - 177303	Byn EXT <operand/r/BY>	546
177304 - 177307	Byn EXT <operand/r/BY>	547
177310 - 177313	Byn EXT <operand/r/BY>	550
177314 - 177317	Byn EXT <operand/r/BY>	731

Instruction code	Instruction type	Micro program entry
177320 - 177323	Hn EXT <operand/r/H>	732
177324 - 177327	Hn EXT <operand/r/H>	733
177330 - 177333	Hn EXT <operand/r/H>	734
177334 - 177337	Hn EXT <operand/r/H>	735

Instruction code	Instruction type	Micro program entry
177340 - 177343	Wn EXT <operand/r/W>	736
177344 - 177347	Wn EXT <operand/r/W>	737
177350 - 177353	Wn EXT <operand/r/W>	740
177354 - 177357	Wn EXT <operand/r/W>	741

Instruction code	Instruction type	Micro program entry
177360 - 177363	Fn EXT <operand/r/F>	742
177364 - 177367	Fn EXT <operand/r/F>	743
177370 - 177373	Fn EXT <operand/r/F>	744
177374 - 177377	Fn EXT <operand/r/F>	745

Instruction code	Instruction type	Micro program entry
177440 - 177443	Dn EXT <operand/r/D>	746
177444 - 177447	Dn EXT <operand/r/D>	747
177450 - 177453	Dn EXT <operand/r/D>	750
177454 - 177457	Dn EXT <operand/r/D>	751

16.2 PROBLEM APPROACH

The following section will show how user instructions may be defined. It is difficult to start writing micro code for user instructions before the problem is completely isolated. One approach to the problem may be first to implement the inner part of the function in micro code and place the loop control in assembly. This may give some better performance for the function. But later, when the inner part of the function is working, the loop control is included in the user instruction with even better performance obtained. This is because the user instructions may handle the loop control much more efficiently than possible from assembly. In micro code, the loop control may be done paralell to the operations being done. The indexing may also be done faster than possible in high level language.

The approach to the problem, depends on the user instruction to be implemented. The user defined instruction may have only one or more operands involved in the operation. In the case of only one operand, the problem is quite different from a function involving several operands. In any case, an interface or library routine is required to link user instructions to high level language. By using subroutines, the operands required for the user instruction may be organized so that access to the operands is made easy. The call instruction requires a parameter list. When entering the library routine, the addresses of the operands are placed on the data stack for the routine. This gives a rather easy access to the operands to be used by the user instructions.

We are to implement a user written instruction to add elements of two areas and leave the result in a third area. The FORTRAN code that we want to implement as a user instruction may look as follows:

C The index increments are previously defined.

```
        IA = 1
        IB = 1
        IC = 1
        DO 100 I=1,NN
           VC(IC) = VB(IB) + VA(IA)
           IA = IA + INCA
           IB = IB + INCB
           IC = IC + INCC
100     CONTINUE
```

The micro program then needs access to the addresses of the three areas. We know that the first index is 1 and we need access to the index increments and the element count (NN). This is done by placing the user instruction in a subroutine. This routine may be called from a FORTRAN program, for example. We then define the parameters to be used in the call statement and design the routine to be used as a library.

The call statement is defined as follows:

```
CALL ADD(VA,INCA,VB,INCB,VC,INCC,NN)
```

and the FROTRAN equialence of the routine will be as follows:

```
        SUBROUTINE ADD(VA,INCA,VB,INCB,VC,INCC,NN)
        DIMENSION VA(1),VB(1),VC(1)
        IA = 1
        IB = 1
        IC = 1
        DO 100 I=1,NN
           VC(IC) = VB(IB) + VA(IA)
           IA = IA + INCA
           IB = IB + INCB
           IC = IC + INCC
100     CONTINUE
        RETURN
        END
```

The library routine to be designed, then needs a data stack for the addresses of the parameters in the call. The data stack is used by the user instruction to find the addresses of input and output areas and get the values of index increments and number of element to be added.

To build the library routine, we use the ND-500 assembly language. Choose a user instruction code and write the routine. We may name the instruction we are using, whatever we want, and we use a macro definition to define the name. The instruction chosen is the first single floating point instruction in GROUP 2. The instruction has one memory operand fetched by the prefetch processor. Instruction code chosen is 177510B and has entry in the micro program at address 1002B.

```
        $MACRO ADDM(OPERAND)
"LABEL" H PROG 177510B;GENOP "OPERAND"
        $ENDMACRO
        MODULE ADDINTERFACE             % NAME OF MODULE.
        EXPORT ADD                      % MAKE ADD BE GLOBAL.
        LIB ADD                         % MAKE LIBRARY.
        ROUTINE ADD                     % NAME OF ROUTINE.
        $PACK                           % PACK THE ROUTINE.
DSTACK: STACK FIXED                     % FIX STACK HEADER.
VA:     W  BLOCK 1                      % ADDRESS OF VA.
INCA:   W  BLOCK 1                      % ADDRESS OF INCA.
VB:     W  BLOCK 1                      % ADDRESS OF VB.
INCB:   W  BLOCK 1                      % ADDRESS OF INCB.
VC:     W  BLOCK 1                      % ADDRESS OF VC.
INCC:   W  BLOCK 1                      % ADDRESS OF INCC.
NN:     W  BLOCK 1                      % POINTER TO ELEMENT COUNT.
ENDSTACK                                % END OF DATA STACK.
ADD:    ENTF DSTACK                     % ENTER THE CALL PARAMETERS.
        W MOVE IND(B.INCA),B.INCA       % MOVE INCA TO B.INCA.
        W MOVE IND(B.INCB),B.INCB       % MOVE INCB TO B.INCB.
        W MOVE IND(B.INCC),B.INCC       % MOVE INCC TO B.INCC.
        W MOVE IND(B.NN),B.NN           % MOVE ELEMENT COUNT TO B.NN .
        ADDM(B.VA)                      % USER INSTRUCTION.
        RET                             % RETURN TO CALLER.
        ENDROUTINE                      % END OF ROUTINE.
        ENDMODULE                       % END OF MODULE.
```

This example will cause the stack header to be initiated when entering the routine. The seven locations following the stack header, contain

the information to be used by the user instruction. Index increments and element count are moved onto the data stack and are to be used by their values. This is the library routine we are going to use. The micro code must then be written according to the layout of the data stack to be used and the function to be done.

Then the micro code for the user instruction is to be designed.

The micro program to be written then must take care of indexing the three areas (read source data, write result) and control the loop so that the desired number of elements are added. At the start of the instruction, the address of area VA is fetched by the prefetch and the six next parameters are to be read by the micro program from the data stack.

An instruction with an element count like this may cause an execution time of several milli seconds and must be interruptable. Trap may also interrupt the instruction. This requires that the instruction must be started from its very beginning again, ie., started from the entry in the micro program, and continue the operation at the point it was interrupted. This implies that the indexing and loop control must be done so that the operation continues and ends in the same way as no trap occured. Some day one is writing a program that is using one of the input areas as result area. Even then this instruction should behave as the FORTRAN routine.

To make an instruction interruptable, the ND-500 micro mnemonic symbol PRF,ISAMP is used in the loop. The micro instruction containing PRF,ISAMP is executed while the next micro instruction is trapped and not completed.

Note that in case of page fault, the instruction containing mnemonic symbol causing the micro program to wait for a memory request, is not completed, but trapped immediately.

16.3 MICRO PROGRAM EXAMPLE

The micro program for this routine may then be written as follows.

> Note that this micro code is not the only sollution to the problem. Many other sollutions may be found. The main thing is that the micro code is working properly. Later the problem with optimization may be faced and the real hard problems may arise.

```
% ....   ENTRY IN MICRO PROGRAM.
1002/
MADD1:
        ALU,ADIR A,DATA TYP,W D,AM#20 W,PMEM  % ADDR. VA.
        AA+AB    AA,EA1 AB,DPARG 4 MEM,RD4    % READ ADDR. INCA.
        JMP      MADD2;

% ....   SECOND ENTRY IN UPPER PART OF WRITABLE CONTROL-STORE.
16010/
MADD2:
        ALU,ADIR A,DATA TYP,W D,AL#20 W,PMEM  % INCA.
        AA+AB    AA,EA1 AB,DPARG 4 MEM,RD4    % READ ADDR. VB.
        NEXT;

        ALU,ADIR A,DATA TYP,W D,AM#21 W,PMEM  % ADDR. VB.
        AA+AB    AA,EA1 AB,DPARG 4 MEM,RD4    % READ INCB.
        NEXT;

        ALU,ADIR A,DATA TYP,W D,AL#21 W,PMEM  % INCB.
        AA+AB    AA,EA1 AB,DPARG 4 MEM,RD4    % READ ADDR. VC
        NEXT;

        ALU,ADIR A,DATA TYP,W D,AM#22 W,PMEM  % ADDR. VC.
        AA+AB    AA,EA1 AB,DPARG 4 MEM,RD4    % READ INCC.
        SET      COND,PDONE                   % TEST RESTART .
        NEXT;

        ALU,ADIR A,DATA TYP,W D,AL#22 W,PMEM  % INCC.
        AA+AB    AA,EA1 AB,DPARG 4 MEM,RD4    % READ ELEMENT COUNT.
        JMPNS    LOAD   MADD3;

% ....   RETURNS TO THIS POINT AT RESTART ENTRY.
%       ADJUST COUNTER FOR ADDITIONS DONE (AM$11) AND INDEX
%       REGISTERS.NOTE THE LOCATION INCREMENT OF INDEX REG.
%       COUNTER FOR ADDITIONS DONE AND LOCATION OF PRF,ISAMP.
MADDR:
        ALU,A-1 A,AM#11 TYP,W D,AM#11
        NEXT;                                 % DECR. COUNTER.

        ALU,A-B A,X#0 B,AL#20 TYP,W D,X#0
        NEXT;                                 % ADJUST IX0.

        ALU,A-B A,X#1 B,AL#21 TYP,W D,X#1
        NEXT;                                 % ADJUST IX1.
```

```
          ALU,A-B A,X#2 B,AL#22 TYP,W D,X#2
          POPRET;                                % ADJUST IX2.

% ....    END OF INITIATING AT RESTART.
% ....    RETURNS TO THIS POINT AT FIRSET ENTRY.
%         INITIATING REQUIRED ACCORDING TO CODE LAYOUT.
MADDF:
          ALU,A-B  A,ZRO B,AL#20 TYP,W D,X#0
          NEXT;                                  % IX0 = 0 - INCA.

          ALU,ADIR A,ZRO TYP,W D,X#1
          NEXT;                                  % IX1 = 0.

          ALU,A-B  A,ZRO B,AL#22 TYP,W D,X#2
          NEXT;                                  % IX2 = 0 - INCC.

          ALU,OR   A,XD,S1 B,BM#2 TYP,W D,S1 SLOW1
          NEXT;                                  % SET PART DONE IN S1.

          ALU,FZRO D,AM#11 POPRET;               % CLEAR AM#11.

% ....    END OF INITIATING FIRST ENTRY.

% ....    CHECK FIRST ENTRY OR RESTARTED.
MADD3:
          ALU,ADIR A,DATA TYP,W D,AM#30 W,PMEM
          C,SEQ    F,JSR  MADDF                  % FIRST ENTRY.
          IFT      JSRSTK;                       % RESTARTED.

          ALU,A-B  A,AM#30 B,AM#11 TYP,W D,LC SLOW1
          SET      COND,MSORZ                    % CHECK IF END.
          NEXT;

          ALU,ADIR A,AM#20 TYP,W D,DP SLOW1      % ADDR. VA TO DP1.
          SET      COND,LCZ                      % TEST CON. IN LOOP.
          C,SEQ    F,NEXT
          IFT      JMP      MADDEND;             % END.

          ALU,ADIR A,AM#21 TYP,W D,DP SLOW1      % ADDR. VB TO DP1.
          PASSAA   AA,DP1                        % ADDR. VA TO EA1.
          NEXT;

          ALU,ADIR A,AM#22 TYP,W D,DP SLOW1      % ADDR. VC TO DP1.
          PASSAA   AA,DP1 EA1INH                 % ADDR. VB TO EA2.
          NEXTNS   LOAD;                         % STACK RETURN ADDR.

          ALU,A+1  A,AM#11 TYP,W D,AM#11         % COUNT N.
          AA+AB    AA,EA2 AB,4IX IX1 EA1INH EA2INH
          MEM,RD4                                % READ VB(IB).
          LCDECR                                 % COUNT DOWN.
          NEXT;

          ALU,A+B  A,X#0  B,AL#20 TYP,W D,X#0    % IA = IA + INCA.
          NEXT;

          ALU,A+B  A,X#1  B,AL#21 TYP,W D,X#1    % IB = IB + INCB.
          NEXT;
```

```
        ALU,A+B  A,X#2  B,AL#22 TYP,W D,X#2    % IC = IC + INCC.
        PRF,ISAMP                              % SAMPLE INTERRUPTS.
        NEXT;

        ALU,ADIR A,DATA TYP,W D,AM#20 W,PMEM   % WAIT, SAVE VB(IB).
        AA+AB    AA,EA1 AB,4IX IX0 EA1INH EA2INH
        MEM,RD4                                % READ VA(IA).
        NEXT;

        EX,SUM    A,AM#20 B,DATA TYP,F SINGLE % VB(IB) + VA(IA).
        W,PMEM                                 % WAIT FOR VA(IA).
        NEXT;

        ALU,BDIR  B,XRESM TYP,F SINGLE         % READ FLOAT.RESULT.
        AA+AB     AA,DP1 AB,4IX IX2 EA1INH EA2INH
        MEM,WR4                                % WRITE TO VC(IC)
        W,IOEM                                 % WAIT FLOAT. & MEM.
        C,SEQ     F,JMPSTK                     % NOT END.
        IFT       NEXT;                        % END.

MADDEND:
        ALU,ANDCB A,XD,S1 B,BM#2 TYP,W D,S1 SLOW1
        NEXT;                                  % RESET PART DONE.

        ALU,A+B   A,X#0 B,AL#20 TYP,W D,X#0
        NEXT;

        ALU,A+B   A,X#2 B,AL#22 TYP,W D,X#2
        NEXT;

        JMPMAP    SLOW2 PRF,EOP;
```

16.4 INSTALLING USER INSTRUCTIONS

The micro code for user written instructions may be loaded into the writable control store after the system micro code is loaded. Then the entries of each user written instruction are to be modified according to the first micro instruction. During the test phase, the entry in the micro program may contain only a jump to the second part of the instruction. Referring to the example we have been working with, the following is a guide on how to include user written micro code to the writable control store. The entry in the upper part of the writable control store is address 16010B. The user instruction code to be used is 177510B, ie., the entry in the micro program at address 1002.

To load the writable control store, enter the system as user SYSTEM.

@ND-500-MONITOR

N500:CC set unavailable for other users and load control store.

N500:SET-ND-500-UNAVAILABLE

N500:CC load the user written micro program

N500:LOAD-CONTROL-STORE CONTROL-STORE,16010,200

N500:CC modify the entry of the user instruction

N500:LOOK-AT-CONTROL-STORE 1002

MODIFICATIONS TO BE SAVED ON (SYSTEM)CONTROL-STORE:DATA ? <n>

1002: JMP 000453 000000 <type EDIT and <CR> to modify>.

Insert new contents of address 1002. EDIT is terminated by <CR> The new contents of address 1002 are rewritten by the ND-500 monitor. Check if the whole micro instruction is included. Be careful. EXIT to leave LOOK-AT-CONTROL-STORE.

N500:CC start the ND-500.

N500:MASTER-CLEAR

N500:MICRO-START 0

N500:LOAD-SWAPPER SWAPPER

N500:START-SWAPPER

N500:GIVE <xxxx>

Exit from the ND-500 monitor will cause the ND-500 available for other users. However, the ND-500 should be unavailable as long as debugging the new micro code.

The user written micro code may also be included in the control-store file by reading the contents from the data file to be included and write the contents into the control-store file. This should be done when new instructions are to be included in the ND-500.

Note the following before trying to modify the writable control store:
- The ND-500 monitor has an ND-500 micro code assembler and disassembler which make it easy to modify the ND-500 writable control store.
- Mnemonic symbols with corresponding value equal to zero, should not be used when editing the writable control store.
- Symbols with overlapping values or symbols not recognized will also cause an error message from the micro code assembler. This will cause the whole edit to be skipped and the old contents to be retained.

To verify the contents of a micro instruction, W for word display, G for group display may be used together with the mnemonic list. S will turn back to symbolic display.

16.5 DEBUGGING

A useful command in the ND-500 monitor, while debugging new instructions installed in the ND-500, is the command:

LOOK-AT-HARDWARE <register>

The legal terms for <register> are:

- INTERFACE
- A,XD
- MMS
- any register name found in the ND-500 mnemonic symbol table.

Contents of the registers are displayed together with the name of the register.

In addition, no register may be given (two carriage returns), causing the ND-500 monitor to dump contents of the context registers together with scratch registers, loop counter, effective address registers etc.

> Note that the effective address registers are given in 1's complement.

When INTERFACE is used, status of the ND-500 is reported, along with any stop reason and micro program address for the stop.

When A,XD is used, any register connected to the XD-BUS and containing the A,XD instruction is dumped.

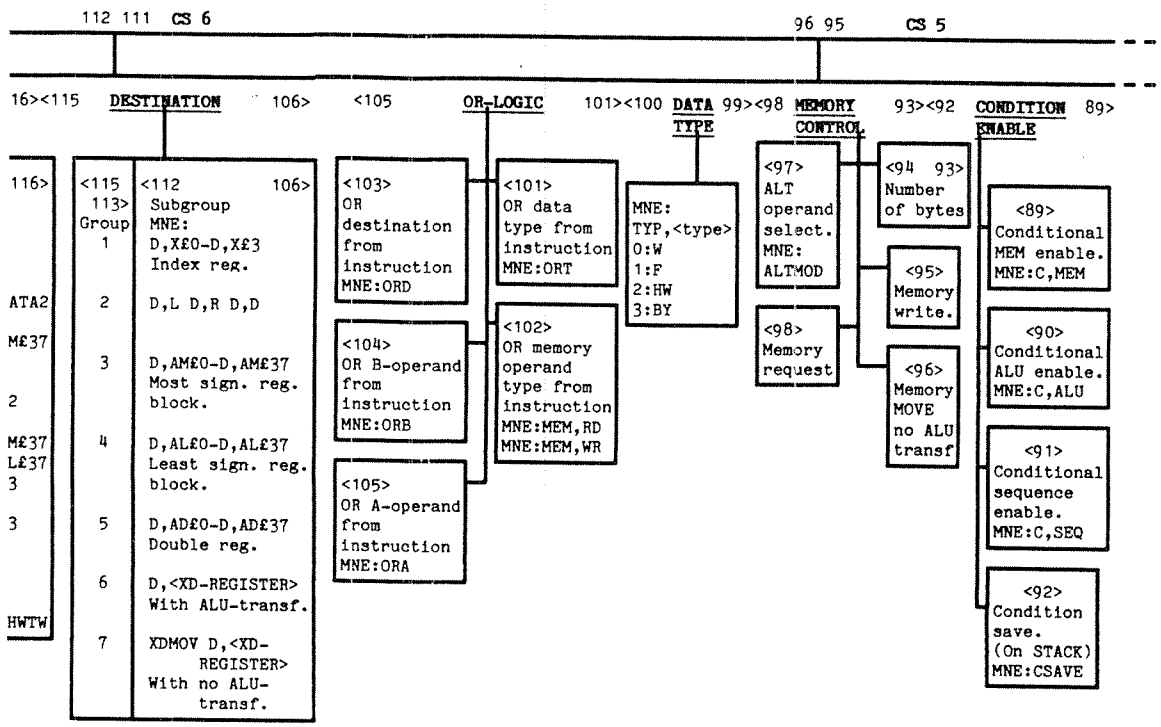
> Note that only the rightmost character of the register name are displayed.

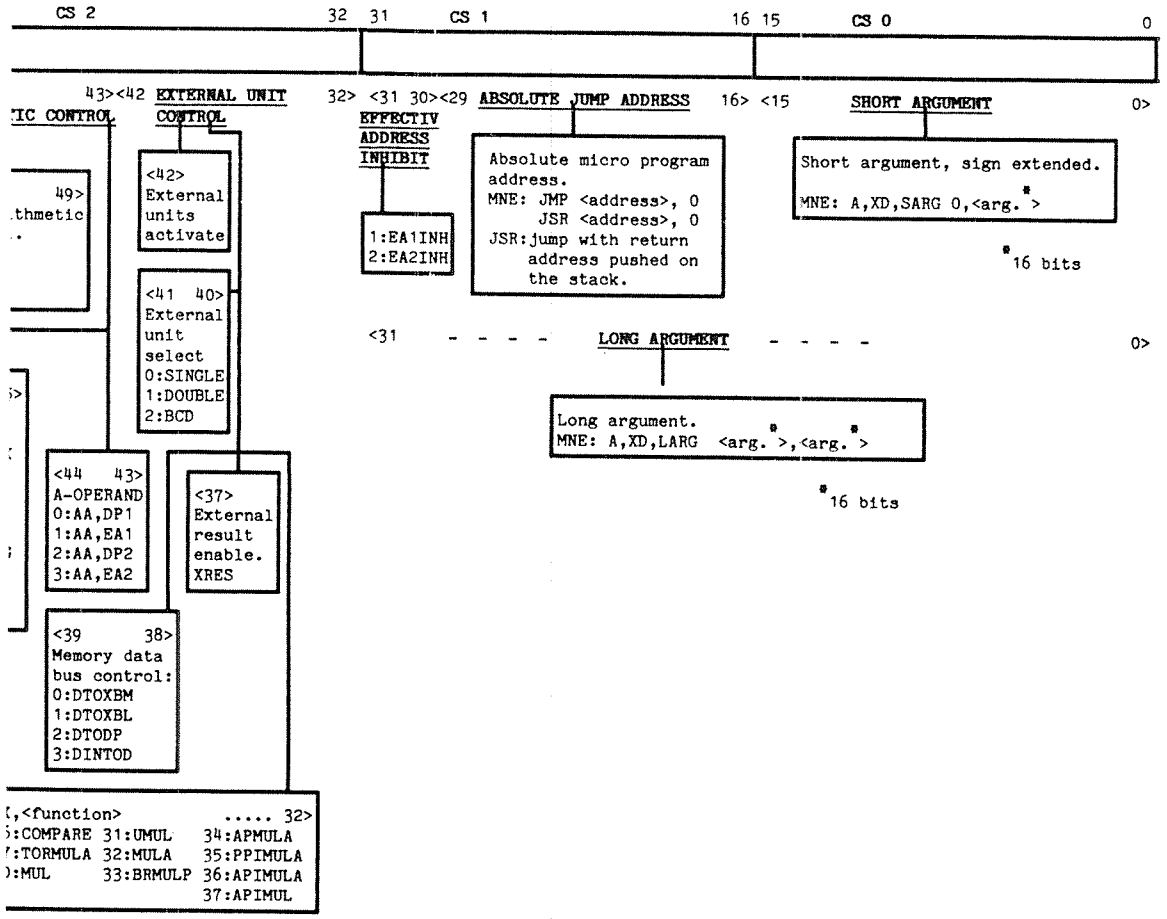
When MMS is used, the memory management scratch file for data and instruction is dumped.

While debugging micro code, the writable control store may be modified. The mnemonic symbol W,XD may be placed in the writable control store to stop the micro program and check 'key values' for the micro code. Contents of registers may be displayed by the command LOOK-AT-HARDWARE <register>.

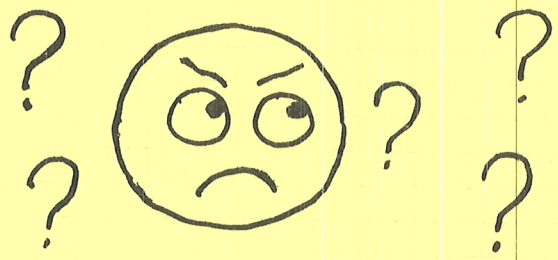
While debugging user written instructions, the ND-500 should be set unavailable for other users. This is because any stop in the micro program will cause the other processes currently running in the ND-500 to stop. No one would be happy about this. As long as an user written instruction does not run even without temporary stops, the ND-500 should be kept unavailable for other users while debugging the micro code.

Note that when using the command
LOOK-AT-HARDWARE or LOOK-AT-CONTROL-STORE the
ND-500 is stopped and has to be restarted again.
Restart of the ND-500 means MASTER-CLEAR,
MICRO-START 0, LOAD-SWAPPER <file>,
START-SWAPPER and GIVE <number of pages>.


112 111 CS 6
96 95 CS 5
16><115 DESTINATION 106>
<105 OR-LOGIC 101><100 DATA TYPE 99><98 MEMORY CONTROL 93><92 CONDITION ENABLE 89>
116>
<115 113> Group
<112 106> Subgroup MNE:
1 D,X£0-D,X£3 Index reg.
2 D,L D,R D,D
3 D,AM£0-D,AM£37 Most sign. reg. block.
4 D,AL£0-D,AL£37 Least sign. reg. block.
5 D,AD£0-D,AD£37 Double reg.
6 D,<XD-REGISTER> With ALU-transf.
7 XDMOV D,<XD-REGISTER> With no ALU-transf.
ATA2
M£37
2
M£37
L£37
3
3
HWTW
<103> OR destination from instruction MNE:ORD
<104> OR B-operand from instruction MNE:ORB
<105> OR A-operand from instruction MNE:ORA
<101> OR data type from instruction MNE:ORT
<102> OR memory operand type from instruction MNE:MEM,RD MNE:MEM,WR
MNE: TYP,<type> 0:W 1:F 2:HW 3:BY
<97> ALT operand select. MNE: ALTMOD
<98> Memory request
<94 93> Number of bytes
<95> Memory write.
<96> Memory MOVE no ALU transf
<89> Conditional MEM enable. MNE:C,MEM
<90> Conditional ALU enable. MNE:C,ALU
<91> Conditional sequence enable. MNE:C,SEQ
<92> Condition save. (On STACK) MNE:CSAVE


CS 2
32 31 CS 1
16 15 CS 0
0
43><42 EXTERNAL UNIT CONTROL
32> <31 30><29 ABSOLUTE JUMP ADDRESS
16> <15 SHORT ARGUMENT 0>
EFFECTIV ADDRESS INHIBIT
1:EA1INH
2:EA2INH
Absolute micro program address.
MNE: JMP <address>, 0
JSR <address>, 0
JSR:jump with return address pushed on the stack.
Short argument, sign extended.
MNE: A,XD,SARG 0,<arg. >
*16 bits
<31 LONG ARGUMENT 0>
Long argument.
MNE: A,XD,LARG <arg. >,<arg. >
*16 bits
<42> External units activate
<41 40> External unit select 0:SINGLE 1:DOUBLE 2:BCD
<44 43> A-OPERAND 0:AA,DP1 1:AA,EA1 2:AA,DP2 3:AA,EA2
<37> External result enable. XRES
<39 38> Memory data bus control: 0:DTOXBM 1:DTOXBL 2:DTODP 3:DINTOD
,<function> ..... 32>
):COMPARE 31:UMUL 34:APMULA
':TORMULA 32:MULA 35:PPIMULA
):MUL 33:BRMULP 36:APIMULA
37:APIMUL

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

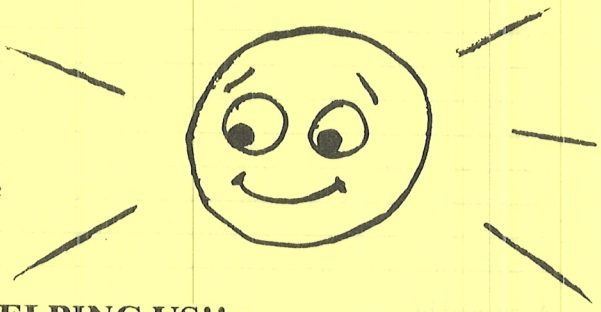


Are you frustrated because of unclear information in this manual? Do you have trouble finding things? Why don't you join the Reader's Club and send us a note? You will receive a membership card - and an answer to your comments.

Please let us know if you

* find errors
* cannot understand information
* cannot find information
* find needless information

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



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

Manual name: ND-500 Micro Program Guide Manual number: ND-05.012.01

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

_____

_____

_____

_____

_____

_____

_____

_____

_____

Do you have suggestions for improving this manual? _____

_____

_____

_____

_____

_____

_____

_____

Your name: _____ Date: _____

Company: _____ Position: _____

Address: _____

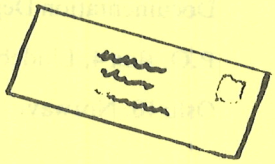
_____

What are you using this manual for? _____

_____

_____

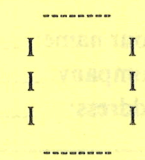
Send to: Norsk Data A.S.
Documentation Department
P.O. Box 4, Lindeberg Gård
Oslo 10, Norway



Norsk Data's answer will be found on reverse side

Answer from Norsk Data _____

Answered by _____ Date _____



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

– we make bits for the future

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