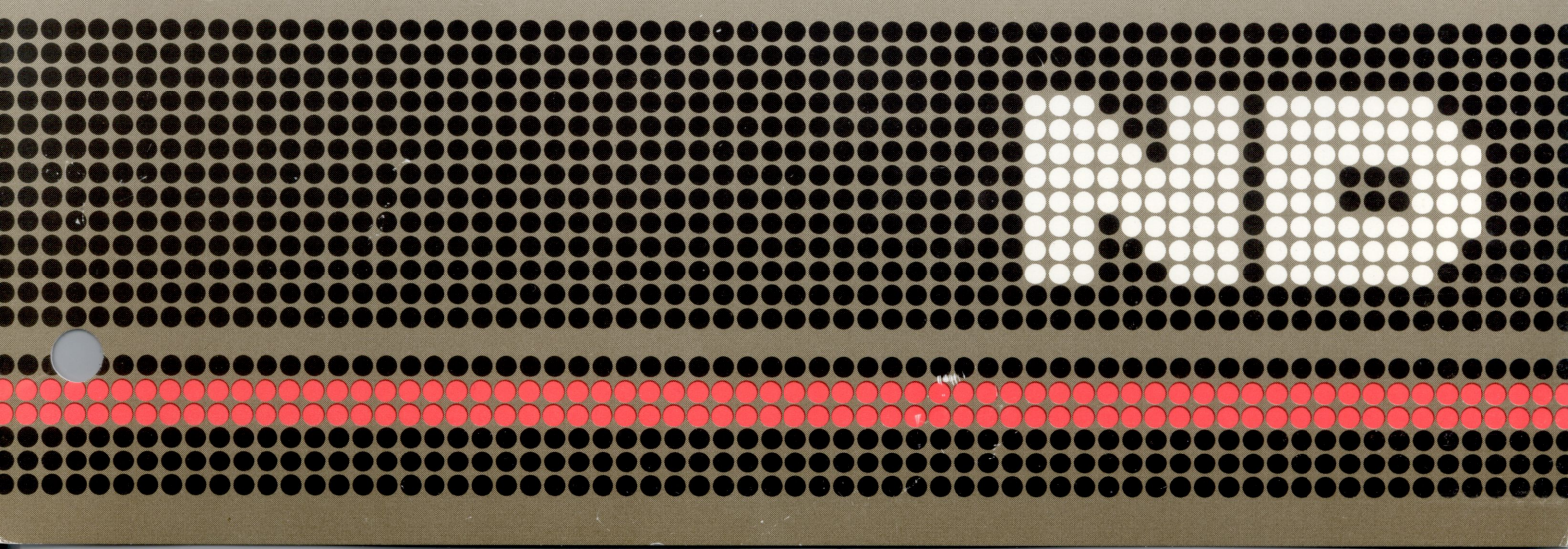
ND–5000 Microprogram Guide

ND-05.022.1 EN



UPDATING

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

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

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

PRINTING RECORD	
PRINTING	NOTES
06/87	Version 1

ND-05.022.1
ND-5000 Microprogram Guide

RING BINDER OR PLASTIC COVER

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

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

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

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

I would like to order

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

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

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

Name: .....

Company: .....

Address: .....

Preface

The manual

This manual gives an introduction to the ND-5000 microprogramming and states rules for use of some of the commands available in the ND-5000 mnemonic symbols.

The manual describes how the different ND-5000 CPU hardware units should be controlled when they are involved in execution of the ND-5000 microprogram. Rules for controlling these hardware units are stated when possible.

For a detailed register hardware description, see the manual ND-5000 Hardware Description (ND-05.020).

The reader

The manual is made for people writing microprogram routines for the ND-5000, and for people working with the ND-5000 hardware.

Prerequisite knowledge

Some knowledge about the ND-500 architecture and detailed knowlegde about the ND-5000 hardware is required to use the manual.

Related manuals

- SAMSON Design Information ND-05.021
- ND-5000 Hardware Description ND-05.020
- ND-500 Reference manual ND-05.009

Table of contents

Table of appendices

CHAPTER 1 INTRODUCTION

In the ND-5000, the microprogram controls the communication between different parts of the central processing unit (CPU). These parts are:

- Microinstruction controller (MIC)
- Instruction cache (ICA)
- Instruction address controller (IAC)
- Instruction decode unit (IDU)
- Instruction memory management (IMM)
- Data cache (DC)
- Data address controller (DAC)
- Data memory management (DMM)
- Input and output system
- Trap system (TRP)
- Arithmetic logic unit (ALU)
- Additional arithmetic processor (AAP)

The ND-5000 microprogram consists of microinstructions. The microinstruction width is 128 bits, and the bits are explained on page 3. The complete microinstruction format is shown in Appendix B, page 65.

An ND-500 macroinstruction needs a number of microinstructions, depending on the complexity of the macroinstruction. The CPU microprogram controls the fetch of operands connected to a macro-instruction.

The microprogram uses data, which is fetched by the CPU microprogram, from registers or from the cache and memory system. This data is used in ALU operations or operations carried out on the AAP. This involves synchronization with the AAP. Synchronization with the cache and the memory system is done automatically.

The microprogram is divided into three parts.

1. Instructions
2. Communication between the I/O processor (ND-110) and the ND-5000
3. Trap handling

Microinstructions in the ND-5000 CPU are pipelined. This means that the CPU is placing microinstructions to be executed in a pipeline. The pipeline consists of four levels of micro-instructions to be executed one after the other. The levels are:

- I-level (instruction level)
- M-level (data level)
- A-level (ALU level)
- F-level (result level)

When the CPU microprogram asks for a new instruction and a new operand, the necessary action is taken on the I-level of the pipeline. Instruction fetch, operand fetch and instruction decoding are controlled by the IDU and IAC.

On the M-level of the pipeline, the MIC is active. It is dealing with sequencing of the microprogram and generating addresses to the scratch register file. The DAC is completing the operand address, and the data cache or the register file is accessed.

On the A-level of the pipeline, the ALU is active, performing operations on data selected from one of these sources:

- an operand decoded from a macroinstruction
- the working register file (WRF)
- the scratch register file (SRF)
- registers elsewhere in the CPU.

On the F-level, the result from operations done on the A-level is routed to the selected destination (if any). This destination can be memory, or registers in the CPU.

CHAPTER 2 FORMAT OF THE MICROWORD

The ND-5000 microword is 128 bits wide. It is divided into several groups, each group controlling special parts or functions in the ND-5000 CPU. The value of each function is given a mnemonic symbol. The mnemonic symbols can be combined to represent more complex functions. Symbols using the same field should not be used together. This is allowed by the ND-5000 microassembler, as long as the mnemomic symbol do not try to set the same bits in the field. See also Appendix B, page 65.

Microword bits	Control function
127 - 122	ALU function and carry select (true)
121 - 116	ALU function and carry select (false)
115	Execute unconditional
114	Enable conditional ALU operation
113 - 111	Q-register control
110 - 103	Additional arithmetic processor control
102 - 101	Timing control
100 - 98	Data-type control
97	Or control (ORCON) enable
96 - 89	A-operand select
88 - 84	B-operand select
83 - 76	Destination select
75 - 72	Status bits control
71	Index counter increment
70	Loop counter decrement
69	Enable conditional sequence
68 - 65	Sequence and stack control (true)
64 - 61	Sequence and stack control (false)
60	Invert sequence condition
59	Save test condition
58 - 53	Select test object
52 - 51	Alternative branch control
50 - 48	Instruction cache write control
47 - 44	Fetch control
43	Stop
42	AAP synchronization
40	Address arithmetic control, OCA/Micro
39 - 38	Effective address save control
37	Memory request controlled by address code
35	Address arithmetic activate (ADACT)
41, 34 - 32	Data memory control
31 - 16	Absolute microprogram address
15 - 13	Address A-operand select
12 - 9	Address B-operand select
31 - 0	Long argument
15 - 0	Short argument with sign extension
7 - 0	Mini argument with sign extension
5 - 0	Or logic control (ORCON)

CHAPTER 3 THE ND-5000 REGISTERS

Macroinstructions requiring more than one microinstruction usually require scratch registers for temporary storing of operands or results. The registers available in the ND-5000 CPU may be divided into several groups. From the microprogrammer's point of view, the registers present in the CPU, may be divided into three groups:

1. Context registers, connected to the running process
2. Scratch registers
3. Registers allocated for special use

However, this may be different from the hardware point of view. Registers within one of the groups defined in table 1, may reside in different hardware modules. A detailed description of the registers with respect to hardware modules is found in the manual ND-5000 Hardware Description (ND-05.020.1).

3.1 CONTEXT REGISTERS

Context registers should only be changed by microcode when control is decoded from an assembly instruction. An exception is the context scratch registers.

Register	Abbreviation	Resides in
Program counter	P	IAC gate array
Link registerer	L	IAC gate array
Base register	B	DAC gate array
Record register	R	DAC gate array
Index registers	I1 to I4	WRF gate array
Floating most registers	A1 to A4	WRF gata array
Floating least registers	E1 to E4	WRF gate array
Status register	S1 + S2	Different gate arrays
Process register	PS	DMM + IMM gate arrays
Current executing domain reg.	CED	DMM + IMM gate arrays
Current alternative domain reg.	CAD	DMM + IMM gate arrays
Microprogram scratch register	SC1 + SC2	WRF gate array
Registers for each domain:		
Top of stack register	TOS	
Lower limit register	LL	
Upper limit register	HL	
Trap handler address register	THA	Domain info. table
Own trap enable register	OTE1 + OTE2	Different gate array
Mother trap enable register	MTE1 + MTE2	Different gate array
Child trap enable register	CTE1 + CTE2	Domain info. table
Trap modification mask	TEMM1 + TEMM2	Domain info. table

Table 1. Context Registers for a Process in the ND-5000

As table 1 shows, the context registers are found in different gate arrays. The base (B) and the record (R) registers reside in the DAC gate array. The floating registers (D1 to D4), index registers (I1 to I4) and context scratch registers (SC1 and SC2) reside in the WRF gate array. The program counter (P) and the link register (L) reside in the IAC gate array.

The trap enable register will have parts residing in different gate arrays according to traps detected by the different gate arrays. Hardware trap enable register is either MTE when inside trap handler or MTE OR'ed with OTE when outside trap handler. The process (PS), current executing (CED) and alternative (CAD) registers will reside in both the DMM and IMM gate array. The trap handler (THA), the child trap enable (CTE1 and CTE2) and the trap enable modification mask (TEMM1 and TEMM2) registers will reside only in the Domain Information Table.

3.2 Scratch Registers

A large number of scratch registers are present in the ND-5000 CPU. The most important scratch register is the Q-register located at the ALU output. Special hardware is dedicated to control multiply and divide operations on Q-register in parallel with ALU operations.

The WRF has three-address structure, which means that it is accessed easily through two read ports (A and B) and one write port. All 'read before write' problems are solved in hardware.

The SRF is accessed through a single port (bus), and the microprogram must know of the pipeline peculiarities.

Some scratch registers are located in the working register file (WRF) close to the ALU. It may be used from the microprogram for saving of temporary results. As an extension of these scratch register, the scratch register file (SRF) is implemented, not located as near to the ALU as the WRF.

The scratch registers are called SC1 to SC13.

The scratch register file (SRF) may also be used as scratch registers, when required. The scratch register file consists of 4K of 32-bit wide registers.

3.3 SPECIAL ALLOCATED REGISTERS

In the SRF, from address 0 to 15, SRF0 to SRF15, and from SRF address 2000B to 7777B, registers are allocated for special use and should be used as read only.

Constants used in the mathematical functions are allocated from address 4000B.

Some of these registers allocated for special use contain information without any copy elsewhere in the CPU. These registers are dynamically updated according to requirements of the running process and should only be changed by the system related microprogram routines.

CHAPTER 4 OPERAND CONTROL

Control of operands, read and write, together with operand select, data type control and updating the cache system, is done by the ND-5000 CPU microprogram. The operand control may be divided into fetch control and OR-logic control and involves communication with several units in the ND-5000 CPU.

4.1 FETCH CONTROL

The different commands for control of fetch operations must be used according to the operand definitions for an assembly instruction. For multi-operand instructions, with the second or later operand defined as a direct operand, fetch of the direct operand must indicate the size of the operand. Special fetch commands are not required for the first operand in case of direct operands.

Fetch commands:

G,OOPS Fetch next instruction and first operand specifier.

G,OPS Fetch next operand specifier.

G,OPSTRD Fetch second operand specifier for string operations.

G,DIR1 Fetch a one-byte direct operand.

G,DIR2 Fetch a two-byte direct operand.

G,DIR4 Fetch a four-byte direct operand.

G,COOPS Fetch enter instruction and operand specifier after a CALL or CALLG instruction.

G,OOPS,T Fetch next instruction and operand specifier for preferred branch route and break if test condition is true.

G,OOPS,F Fetch next instruction and operand specifier for preferred branch route and break if test condition is false.

G,TOOPS Fetch next instruction to check for Call, Entm, Entt and Jumpg.

The fetch commands fetching the next assembly instruction, will cause the microprogram to start execution in the map address of the fetched instruction.

After fetch of general operands, a command for updating the operand cache must be used. The IAC has to be told how to process operands requested by a fetch command. These commands are the TBC,<xx>, ABR,<yy>, OR,<select> commands. TBC means 'to be cached as next address'. ABR means 'alternative branch or return address'. OR means 'OR-logic control', i.e. process information from assembly instruction to produce microcode operand addresses.

These commands are required in the microinstruction following a fetch cycle. TBC and ABR will control sequencing of instructions on assembly level.

Note!

If for some reason the read is missing after a fetch, the read may be activated in a later cycle.

The OR-logic to be used is specified one cycle before the read.

G,OOPS,T and true test condition : ABR

G,OOPS,T and false test condition : TBC

G,OOPS,F and true test condition : TBC

G,OOPS,F and false test condition : ABR

G,OOPS,T and G,OOPS,F are used in the control instruction on assembly level such as IF = GO <displacement> etc.

The TBC field is only active when the operand cache has to be filled. The ABR field is used to calculate the next instruction address when the next field of the instruction cache should not be used.

TBC commands:

TBC,PREL Cache write P relative jump address. Branch target address relative to P (P+displacement). Used for branch instructions with displacement specified as second or later operand.

TBC,SUBR Cache write subroutine address. Next address to be used is subroutine address of CALL or CALLG instruction.

TBC,NPCREL Cache write NPC relative jump address. Branch target address relative to NPC (NPC + displacement). Used for branch instructions with displacement specified as first operand.

TBC,NEXT Cache write next address. Next address is current address + length of current instruction part.

ABR commands:

ABR,NEXT Alternative branch address is current address (in NPC) + length of current instruction part. The result is put into the IAC scratch register (A,IAC,S).

ABR,NEXTL Alternative branch address is current address (in NPC) + length of current instruction part. The result is put into the L register.

ABR,NPCREL Alternative address is branch target address relative to NPC (NPC + displacement -> IAC scratch register). NPC must be valid. NPC points to beginning of an instruction until and including a fetch operation (G,<fetch>).

4.2 OR-LOGIC CONTROL

Data is enabled into the ALU on either the A or B operands. On the A operand, data is enabled to the ALU by the mnemonic symbols ORA together with source select for the operand in the ORCON field (MIR bits 5 to 0).

On the B operand to the ALU, only operands decoded from the instruction may be selected.

On the A operand to the ALU, operands may be decoded either from the instruction code or from a general operand specifier. For the A operand, the ORCON field then has to select the source for an operand according to operand definitions for the assembly instruction being executed.

The mnemonic symbol ORA,IN, will supply the OR-logic with information that the A operand is to be decoded from the instruction code. The mnemonic symbol ORA,OP will cause the A operand to be decoded from the current operand specifier.

After fetch (G,OOPS or G,OPS, etc.), the ORCON field is not used, else the ORCON field is used in order to select integer register, most significant part of floating register, or least significant part of floating register.

For destination select, the mnemonic symbol ORD is used in the microinstruction, routing data back to a destination. In addition, the ORCON field is to be used in the same manner as for controlling the A operand.

In addition, the ORCON field may select OR-control for next microinstruction. The mnemonic symbol OR,N tells that the OR-logic is to be used in the following microinstruction. The mnemonic symbol OR,NE tells that the OR-logic should be used in the following microinstruction, and that the extension part of a

double floating operand is selected.

If the OR-logic is controlled by the Operand Cache (OCA), we have that after fetch, OR-logic is done in the same cycle as reading/writing the operand.

When OR-logic is used for accessing an already decoded operand, the OR-logic is done in the microinstruction executed prior to the cycle using the OR-logic. If the sequence is different from JMP, the microinstruction pointed to by the jump field will give OR-logic control for the microinstruction to use the OR-logic.

The OR-logic for data type control may also be used to select operations on either byte (8 bits), half-word (16 bits), word and single floating point (32 bits) etc. This is done by using the TYP,OR in the data type control field.

When accessing operands with data type different from the data type for the assembly instruction, the data type must be explicitly controlled. When fetching such operands, selection of index register is done according to the data type control field. Data-type control must then correspond to the data type of the operand in order to have correct scaling in case of post indexing (address code = 340B).

An example is the instruction BYn SFILL <=dest/w/by/I2=>,<m/r/W>. The last operand has a data type different from the data type of the instruction.

In addition to fetch and operand select, the microcode also has to activate the DAC module to generate addresses for operands requested. This is done by the mnemonic symbol ADACT. This is to be used after fetch of a general operand is started and must be done before the read cycle for the operand is completed. For direct operands this is not required.

Example: Instruction: By1 + B.24B

```
        G,OOPS          <end of previous instruction>


<ADD>   ALU,A+B ORA  ORB  TYP,OR        % use ORed data type
        ST,SAVA ORA,OP                  % A ored from operand
                                        % B ored from instr.
        ADACT            READ           % Activate read A
                ORD,IN   ORD            % Dest. from instruct.
                                        % write in ored dest.
        G,OOPS  HOLD;                   % end of By1 + B.24B
```

Example: Instruction: H ADD2 B.24,R.0

```
        G,OOPS          <end of previous instruction>

<ADD2>  ALU,A   ORA       TYP,OR  D,SC5 % use ored data type
                ORA,OP                  % A ored 1. operand
        ADACT             READ          % activate read B.24B
        EA1SAVE                         % save address  B.24B
        G,OPS                           % Fetch next operand
        NEXT*;

        ALU,A+B ORA B,SC5 TYP,OR  D,SC5  ST,SAVA
                ORA,OP                  % A ored 2. operand
        ADACT             READ          % activate read R.0
                ORD,OP1                 % Dest. is 1. operand
        AB,EA1DIR                       % give address latch
                OR,N                    % and OR-logic in next
        NEXT*;

        ALU,A   A,SC5     TYP,OR
                WRITE                   % write to 1. operand
                ORD                     % in case of register
        G,OOPS;
```

CHAPTER 5 ARITHMETIC FUNCTIONS

The arithmetic logic unit (ALU) and the additional arithmetic processor (AAP) are used for arithmetic operations.

The ALU is a gate array especially designed for the ND-5000 CPU.

In the first version of the ND-5000 CPU the AAP consists of the ND-570 floating point unit cards.

In connection with an arithmetic operation, A-operand, B-operand, data type control and destination may be selected from separate fields in the microprogram word.

True and false ALU operations and the AAP are controlled from separate fields, giving the opportunity to run both ALU-operations and AAP operations in the same microinstruction.

5.1 ALU FUNCTIONS

The ALU may perform integer arithmetic and logic operations as described on the next page.

The ALU operations are specified by the symbols ALU,<func> for true ALU-operation select. The ALU operations may also be specified as a false ALU operation by the symbols ALUF,<func>. The false ALU-operation commands enable the conditional ALU operation automatically.

Arithmetic ALU functions without Q-register control:

ALU,A	A-operand + 0
ALU,A+1	A-operand + 1
ALU,A-1	A-operand - 1
ALU,A+B	A-operand plus B-operand
ALU,A+B+1	A-operand plus B-operand + 1
ALU,A-B	A-operand minus B-operand
ALU,A-B-1	A-operand minus B-operand - 1
ALU,A-B-1+C	A-operand minus B-operand - 1 + status carry
ALU,B-A	B-operand minus A-operand
ALU,B-A-1	B-operand minus A-operand - 1

Arithmetic ALU functions with Q-register control:

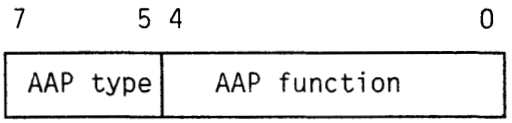
ALU,A,/2	A-operand + 0 FBUS = ALU.output/2. FBUS.bit.31 = carry
ALU,A+B,/2	A-operand plus B-operand FBUS = ALU.output/2. FBUS.bit.31 = carry
ALU,A+B,*2	A-operand plus B-operand FBUS = ALU.output*2. FBUS.bit.0 = 0
ALU,A-B,*2	A-operand minus B-operand FBUS = ALU.output*2. FBUS.bit.0 = 0
ALU,A-B-1,*2	A-operand minus B-operand - 1 FBUS = ALU.output*2. FBUS.bit.0 = 0

Logic ALU functions:

ALU,FZRO	Force zero from ALU output
ALU,ADIRC	A-operand complemented
ALU,AND	A-operand AND B-operand
ALU,ANDCA	A-operand complemented AND B-operand
ALU,ANDCB	A-operand AND B-operand complemented
ALU,OR	A-operand OR B-operand
ALU,XOR	A-operand XOR B-operand

5.2 The Additional Arithmetic Processor (AAP)

The additional arithmetic processor (AAP) is given a separate field (AAPC) in the microword for control:



The AAP type used in the first version of the ND-5000 is called the AAP1. The AAP1 consists of the ND-570 floating point unit cards. These are interfaced to the ND-5000 by the 5456 (or 5466) AAP interface card. It is controlled by the AAP control field AAPC(0-7) with AAPC(5-7)=001, and by the type field in the ND-5000 microword.

The AAP function bits, AAPC(0-4), give space for 32 different operations. The AAP1 can perform shift operations, floating point and integer conversion, floating point arithmetic and integer multiply. The following functions are available:

AAPC 765 43210	Function mnemonic	Allowed TYP,<type>	Function definition
001 00000			reserved
001 00001	AAP1,CTF	BY HW W	Convert to floating
001 00010	CTDF	BY HW W	Convert to double floating
001 00011	UCTF	W	Unsigned convert to floating
001 00100	UCTDF	W	Unsigned convert to double floating
001 00101	CTBYR	F DF	Convert to byte rounded
001 00110	CTHWR	F DF	Convert to halfword rounded
001 00111	CTWR	F DF	Convert to word rounded
001 01000	CTBY	F DF	Convert to byte
001 01001	CTHW	F DF	Convert to halfword
001 01010	CTW	F DF	Convert to word
001 01011	INTR	F DF	Integer part rounded
001 01100	INT	F DF	Integer part
001 01101	SHA	BY HW W	Shift arithmetic
001 01110	SHL	BY HW W	Shift logical
001 01111	SHR	BY HW W	Shift rotational
001 10000	DTOFR	DF	Convert double to single rounded
001 10001	A + B	F DF	Add
001 10010	B - A	F DF	Subtract
001 10011	B / A	F DF	Divide
001 10100			Unused
001 10101	COMP	F DF	Compare (B - A)
001 10110			Unused
001 10111	DIVP	F DF	Partial divide (B / A)
001 11000	A * B	BY HW W F DF	Multiply
001 11001	UMUL	W	Unsigned multiply with overflow
001 11010	MUL4	W	Multiply with overflow
001 11011	RRF		Read AAP register file (32 bits)
001 11100	WRF		Write AAP register file (32 bits)
001 11101			Unused
001 11110	CLEAR		Reset AAP (interface)
001 11111			Unused

Table 2. AAP1 Functions

5.3 INPUT/OUTPUT TO/FROM THE AAP1

Input operands are specified in the A- and B-operand fields of the microinstruction.

For the two-operand instructions, the function mnemonic usually shows which operand is which.

Example: AAP1,B/A and AAP1,B-A.

The shift instructions require the shift count on B-operand, and the operand to be shifted on A-operand.

Shift count > 0 implies shift left. Shift count < 0 implies shift right. Shift right is not implemented in rotational shift.

In the one-operand instructions, the operand is given as A-operand, except for DTOFR which needs the operand as B-operand, and zero on A-operand.

64-bit operands (double floating) must be given in two following cycles. In the first cycle, the function code and type specification are given together with the most significant part of the operand(s) in the A (and B) fields. In the next cycle, the least significant part of the operand(s) is specified in the A (and B) fields. Function code must not be given again in this second cycle. Type is not necessary.

In the convert instructions, the type field specifies the type of the source operand, while the result type is implicit in the function code.

To get the result back from the AAP1, the AAPSYNC must be set from the microcode. This cannot be done in the instruction that starts the AAP1, nor in the second instruction where eventual least parts are given (double floating). In the following instructions, AAPSYNC may be given anywhere. If the AAP1 has then completed its operation, the result is immediately handed over. If not, the ND-5000 will wait for the completion. The result is sent to the operand specified in the D (destination) field of the microword, in the same instruction as the AAPSYNC is given.

If this instruction also contains an ALU operation, the presence of the AAPSYNC will ensure that it is the AAP1 result, and not the ALU result, which goes to the specified destination.

However, the ALU may generate a result for status testing. Also the ALU result might be sent to the Q register by the Q,F specification, since this is not possible for the AAP1 result, which must use the proper destination field.

Similar to input operands, a 64-bit result is sent back in two cycles, first the most significant part, then the least. In MUL4 and UMUL the overflow part is sent first, and then the result.

The AAPSYNC must be specified in both cycles.

5.4 STATUS SETTING

Status save is done in the instruction with the AAPSYNC, where the result is read back. For F or DF, type ST,SAVF is suitable.

There is no status code for integer types that takes sign and zero from the AAP1, and also sends the AAP1 overflow to the integer overflow status bit. Therefore Such results are therefore sent through the ALU afterwards, with the ST,SAVM (save mixed status) code.

5.5 LIMITATIONS

In byte and halfword instructions, the type specification in the calling instruction effects the input operand(s), by using only 8 (BY) or 16 (HW) bits of it. This type specification also controls the result status, e.g. if type = BY and bit 7 of the result = 1, S is set. It does not control the actual result itself, this contains the full word-length result from the AAP1. Nor does the type specification in the instruction where the result is read back have any effect on truncating the result down to type.

Accordingly, when a BY/HW result is sent back to an operand of the macroinstruction, it must first pass through the ALU, with a proper type specification, to cut it down. Internally in the microcode, in a sequence of AAP1 calls, this is not necessary, since the input values are truncated.

The instructions CTBY/R and CTHW/R do not set the IOVFL according to type, only word integer overflow is detected. Since they also do not cut the result, as mentioned above, they are of no use, so the CTW is used instead, with special tests for overflow.

The divide function does not set the status bit DZ (divide with zero). Thus both the cases 0/X and X/0 should be tested and treated separately when using the divide function.

CHAPTER 6 SOURCE AND DESTINATION CONTROL

This chapter describes how to select source and destinations from the ND-5000 microprogram. This is divided into four different sections, because of the hardware architecture of the registers connected to the CPU:

1. The Q-register
2. The working register file
3. The scratch register file
4. Memory

6.1 THE Q-REGISTER

The bottom level of the registers connected to the CPU is the Q-register. The Q-register is closely connected to the ALU wich have special hardware to control the register. The control of the Q-register is done from a separate field, not affecting the operand select for the ALU or the AAP. This is done in order to implement microcoded divide and multiply operations with reasonable speed without any AAP.

- The Q-register may be loaded independantly of other F-bus operations.
- The Q-register may be shifted left or right independantly of other destinations.
- During left shift the serial input is specically controlled to allow divide function to be carried out easily.
- When both the Q-register and the F-BUS are shifted in the same direction, they are controlled so that a 64-bit shift is performed.

Control functions of the Q-register	
Q,F	Q-register loaded from ALU output.
Q,Q*DIV	Q-register = Q-register*2. Q.bit.0 = DIVR.
Q,Q*LOG	Q-register = Q-register*2. Q.bit.0 = 0.
Q,Q*ROT	Q-register = Q-register*2. Q.bit.0 = Q.sign.bit.
Q,Q/ARI	Q-register = Q-register/2. Q.sign.bit = Q.sign.bit.
Q,Q/LOG	Q-register = Q-register/2. Q.sign.bit = 0.
Q,Q/ROT	Q-register = Q-register/2. Q.sign.bit = Q.bit.0.

6.2 The Working Register File

The next level of the register consists of the working register file. This working register file consists of 24 32-bit registers. This working register file contains:

- registers connected to the running process
- four index registers
- four floating registers, most and least significant part
- some scratch registers

Two of the scratch registers (SC1 and SC2) will be included in the context block, where key values to survive a process change may be held. This is useful in the string instruction and instructions of type array processing functions.

- A-operand and B-operand may be selected independently from the WRF as input to the ALU or the AAP.
- The F-bus may be written into the working register file independantly of selection of A-operand and B-operand select for the ALU or the AAP.
- Only one register block destination may be selected in the same microinstruction. Two working register file sources may be selected in the same microinstruction, independant of destination select.
- The working register file is addressed either directly by addresses present in the A-operand or B-operand select field or by the OR-logic with addresses derived from the assembly instruction being executed.
- A working register file register may be both read and written in the same microinstruction.

Registers in the working register file	
• Index registers	(X1 to X4)
• Floating most registers	(A1 to A4)
• Floating least registers	(E1 to E4)
• Context scratch registers	(SC1 to SC2)
• Scratch registers	(SC1 to SC14)

6.3 The Scratch Register File

The next level of registers connected to the CPU is the scratch register file (SRF). It contains 4k of 32-bit registers. The scratch register file is farther from the ALU and thus not as flexible as the working register file. There are some restrictions when using the scratch register file:

- Registers in the scratch register file may only be addressed directly by microprogram. The scratch register file is either addressed from the A-operand field with register addresses in the range 0 to 15, or by using the RFA1 and RFA2 register as addresses pointing to a register within the scratch register file. (When using the address registers as address in the scratch register file, the register pointed to by the address register is accessed.) In parallel, the address register may either be held at the same value or decremented when accessing a register.

  When reading from the SRF through an address register, this must be set two cycles before.

  The selected register from the scratch register file is enabled on the A-operand to the ALU.

  Example:

```
ALU,A  A,BM05     D,RFA1 NEXT*; % 40 to address reg 1
ALU,A  A,MIC,RFA1 D,NONE NEXT*; % prev. address reg 1
                  D,NONE NEXT;  % 'MIC' acc. impossible
ALU,A  A,RF1D     D,SC3  NEXT*; % Read from SRF.40
```

  When writing to the SRF through an address register, the address may be set in the previous cycle.

Example:

```
ALU,A  A,BM05     D,RFA1 NEXT*; % 40 to address reg 1
ALU,A  A,BM06     D,RFA1 NEXT*; % 100 to address reg 1
ALU,A  A,SC5      D,RF1D NEXT*; % Write in SRF.100
ALU,A  A,SC6      D,RF1  NEXT*; % Write in SRF. 77
```

- Only one scratch register file register may be read in one microinstruction.
- When the scratch register file is selected as destination, the working register file cannot be destination in the same microinstruction. A scratch register file register written in one microinstruction, cannot be read in the two following microinstructions.

6.4 MEMORY

Memory may also be selected as source or destination with address generation controlled either by the assembly instruction executed, or by microcoded control of the address arithmetic with microcoded memory request. Accessing memory requires address latch of an operand in the microcycle executed prior to read or write. Synchronization with the memory system is automatically done at both read and write.

Example of microprogrammed read/write:

```
D,NONE
AA,EA1   10   AB,MARG      % Address of source
NEXT*;

ALU,A     A,DATA   TYP,BY   D,SC3   READ
AA,EA2   374    AB,MARG    % Address of destination
NEXT*;

ALU,A     A,SC3   TYP,B4   D,NONE   WRITE
NEXT*;
```

CHAPTER 7 MICROPROGRAM SEQUENCE

The ND-5000 microinstructions may use different commands for sequence control of the microprogram. The microprogram sequence control consists of a stack command and a sequence command. This implies that both sequence commands and stack commands are required in a microinstruction.

7.1 STACK COMMANDS

The microprogram stack may hold a maximum of four different addresses. The top word of this stack may be selected as input to the microprogram address counter (m.p.c) by the sequence command RETURN.

An important restriction in the microprogram sequencer is that the sequencer stack is not stable before the microinstruction following a load of sequencer stack is executed. This implies that the sequencer stack cannot be used as address input immediately after being loaded. Hence, a one cycle micro-instruction subroutine is not possible. A one cycle subroutine will also lose time in execution speed.

Stack commands and functions are:

HOLD Leave stack unchanged.

LOAD Word 1 is changed to current microaddress + 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.

The stack commands are also available as false stack commands, written as F,<stack> for operation on the sequencer stack.

7.2 SEQUENCE COMMANDS

The sequence commands will either cause the next microinstruction in a sequence to be executed, or will cause some kind of a jump.

The microprogram sequencer works on a three level pipeline and generates addresses to the control store, telling where to find the next microinstruction. Addresses are generated at the first level of the pipeline, the I-level. The sequencer picks the jump address from the I-level and guesses that a jump is in progress.

If any sequence command different from jump is used, the microprogram sequencer has to run one extra sequencer cycle to generate the correct microprogram address. If the guess was true, the address is present and the jump is carried out in one cycle.

In connection with conditional sequence, the true path is selected as a preliminary route by the microprogram sequence. Hence the true sequence command should be a jump instruction. To make it always possible to place a jump in the true sequence field, a control store bit may be used to invert test condition. This is done by the INVSEQ command.

In connection with the mapping to the start of the next assembly instruction, both the true and false sequence field must contain a jump command. The true sequence field must follow the mapping while the false sequence field may be used to stop execution e.g. in connection with reporting an error detected at the end of an instruction.

When accessing operands, these may be prefixed by an address code causing mapping to special microprogram routines to handle the address code prefix. Because of the microcode pipeline, the microprogram sequencer is using the jump address to find the way back to the trapped microinstruction. This means that a read/write/laddr cycle of a general operand always has to use JMP *+1 as sequencer command.

An easy way to generate jump addresses and jump as sequencer command, is to leave a special mnemonic symbol for the microprogrammer, understood by the microcode assembler. Whenever the jump field in a microprogram address is free, i.e. no long argument used, a '*' may be added to the NEXT command. This will cause current microprogram address + 1 to be inserted in the jump address of current microprogram address and the NEXT command is substituted with JMP.

Example of microprogram sequence control:

```
a)  NEXT*  PUSH;              %.. Push b) as stack address
b)  NEXT*  HOLD;              %.. Stack unchanged
c)  ALU,<func>  NEXT* HOLD;
d)  COND,<cond>
    C,SEQ F,NEXT F,LOAD       %.. Change return address
    IFT   NEXT*   HOLD;       %.. Hold stack unchanged
e)  ALU,<func>  NEXT* HOLD;
f)  COND,<cond>
    C,SEQ F,RETURN F,HOLD     %.. Return to stack address b) or e)
    IFT   NEXT*   POP;        %.. leave the sequence,remove b) or e)
```

The sequence control functions are:

- NEXT Take next microinstruction.
- JMP Jump to microprogram address <addr.field>.
- JMPREL Jump relative to <addr.field>
- RETURN Return (from subroutine) to stack address

The sequence commands are also available as false sequence commands, written as F,<seq> for sequencing the microprogram. Enable of conditional sequencing is done by the false sequence and false stack commands.

7.3 Restrictions in Sequencing

7.3.1 Sequence Instructions Rules

1. The instruction in the TRUE field is the main instruction, while the instruction in the FALSE field is the alternative instruction. According to this, the TRUE sequence is used in unconditional sequence and also as a preliminary route, during conditional sequence.

2. JUMP instructions use one clock cycle, while other instructions use two clock cycles. According to this, you should use JUMP instead of NEXT instructions. During conditional sequence it is an advantage to put a JUMP in the TRUE field if possible.

3. When doing conditional sequence, it is possible to force a specific preliminary route by matching the TRUE and FALSE field with the inverted-sequence bit.

4. A test at the end of a microroutine executing a macro-instruction must follow these rules:

   i. The error action must be held in the FALSE field. This can be arranged by means of the inverted-sequence bit.

   ii. The TRUE field must contain a JUMP.

5. Since the sequence control and the stack control are separated, the RETURN instruction does not POP the stack. It only uses the top of stack as next the address to the control store.

6. Instructions generating a hardware branch shall have a jump address pointing to the immediate following instruction.

7.3.2 Stack Instructions Rules

The stack control is separated from the sequence control. During conditional sequence the stack control is pipelined. Therefore you must be very careful when doing stack operations in connection with a conditional sequence and EXUC (execute

unconditional).

1. Unconditional stack control is done directly from the M-level, while conditional control is pipelined one stage down. This will cause collision on the stack if a conditional instruction is immediately followed by an unconditional instruction containing a stack instruction. In this case, the stack control from the last instruction will be lost.

2. The first instruction of a subroutine must not be a conditional stack operation different from HOLD. This also applies to one-instruction subroutines.

3. A one-instruction subroutine should not be called conditionally (JUMP & PUSH).

7.3.3 Accessing the MIC as A-operand/Destination

Read from MIC/SRF is done directly, while writing is pipelined two levels. After a write, there must be two dummy cycles before the same data can be read back. Reading in the first cycle gives old data. Reading in the second cycle results in a collision on the X-bus.

7.3.4 Extra (Sneak) Instructions and the EXUC

Bit number 115 in the microword is called EXUC (execute unconditional). When the sequence instructions NEXT, RETURN or JMPREL are executed, an extra (sneak) cycle is entered into the pipeline on the I-level prior to the 'real' instruction. This extra cycle is stopped on the I-level unless the EXUC facility is used. If the extra cycle is going to be executed, the EXUC bit in the previous microinstruction must be set TRUE. Both stack and sequence instructions in the extra cycle are then ignored.

7.3.5 Conditional Sequence and the EXUC

The construction of the pipeline system makes it necessary to run two microinstructions after a conditional sequence has entered the pipeline, until the condition is valid. These two instructions are called EXCYC1 and EXCYC2. They enter the I-level, and are normally stopped there, but by using EXUC, they can be carried out as ordinary instructions. The rules for using EXUC in this case are:

1. If EXUC is TRUE in the conditional sequence instruction, the EXCYC1 is executed at all pipeline levels.

2. If EXUC is TRUE in the conditional sequence instruction and EXUC is TRUE in the EXCYC1, the EXCYC2 is executed at all pipeline levels.

3. Stack control is influenced by the EXUC. This means that if conditional break does not occur, the stack is controlled as specified. If a conditional break occurs, the stack is not changed.

EXUC works with the microinstruction controller through the clock enable signals.

Note!

If possible, avoid a combination of conditional sequence, stack control and EXUC. This is an example on how you should not combine them:

```
C,SEQ    EXUC    PUSH
                 (POP)
                 (LOAD)
```

CHAPTER 8 CONDITIONAL OPERATIONS

The test conditions listed below may be used for three different purposes in the ND-5000 microprogram. The test conditions selected may have either true or false as state. Some conditions may select test results from either the main status or from the micro-status.

These test conditions are written in the form (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. The selected test condition is automatically activated. The test condition must be selected whenever conditional sequence, conditional ALU operation or conditional save is used.

8.1 ND-5000 Test Conditions

	COND,<condition>	STATE <true/false>
ARITHMETIC OPERATIONS		
Equal to	(M)ZRO	true
Unequal to	(M)ZRO	false
Signed:		
Greater than	(M)SORZ	false
Greater than or equal to	(M)SGN	false
Less than	(M)SGN	true
Less than or equal to	(M)SORZ	true
True less than or greater than or equal to:		
Less than	MSEXO	true
Greater than or equal to	MSEXO	false
Magnitude:		
Greater than	(M)CNZ	true
Greater than or equal to	(M)CRY	true
Less than	(M)CRY	false
Less than or equal to	(M)CNZ	false
Overflow	(M)OVFL	true : false
Parity (from ALU-output):		
Odd parity	PARITY	true
Even parity	PARITY	false
ADDITIONAL ARITHMETIC OPERATIONS		
Check for x/0	MDZ	true : error
Floating sign	MFS	true : false
Floating overflow	MFO	true : false
Floating underflow	MFU	true : false
BCD overflow	MBO	true : false
Invalid operation	MIVO	true : false
PROCESS STATUS		
Loop counter = 0	LCZ	true : false
Bit 0 of the Q-register	Q0	true : false
Flag	K	true : false
Enter instruction ENTF e.t.c	ENTER	true : false
Enter module instruction	ENTM	true : false
Enter trap instruction	ENTT	true : false
Test on jump general	JUMPG	true : false
Data source / destination	DATOP	true : false
Constant source / destination	CONOP	true : false
Part done i.e. restart	PDONE	true : false
Trap	TRAP	true : false
ALU.cond Q0, seq.cond. LCZ	AQSLZ	true : false
Saved condition 1	SAVC1	true : false
Saved condition 2	SAVC2	true : false

8.2 CONDITIONAL SEQUENCE

Conditional sequence is used to select true or false sequence and stack operations in a microinstruction.

Result of a test condition selected in current microinstruction with result from previous microinstruction affecting selected condition, determines true or false sequence and stack control in the microinstruction using the C,SEQ command (C,SEQ implicit in F,<seq> and F,<stack>).

Because of the pipelining of the ND-5000 microprogram, prefetch of microinstructions is active also during conditional operations. The true path is always selected for fetching new microprogram addresses. Thus the true path should contain a jump command causing the microprogram to run the normal path of execution to obtain optimum speed. If this is not possible, the INVSEQ command is used to invert test condition for sequence control. This command has no effect on conditional ALU operations.

The microinstructions in the pipeline will not be executed when the microcode pipeline is broken. However, a microprogrammer seeking optimum efficiency, may wish to execute these microinstructions anyway. In order to prevent duplication of the code, the microcode may be executed unconditionally. This is done by the EXUC command. With exception of the sequence commands, the microinstruction pointed to by the jump field will be executed if EXUC is present.

Example:

```
a)  ALU,A-B A,<ao> B,<bo> TYP,<tt> NEXT* HOLD;
b)  C,SEQ F,HOLD F,NEXT INVSEQ COND,MZRO
    JMP  HOLD m;
c)  ALU,A+B A,<a1> B,<b1> TYP,<t3> D,<d1> NEXT* HOLD;
d)  ALU,XOR A,<a2> B,<b2> TYP,<t3> D,<d2> NEXT* HOLD;
e)  ALU,< nothing special> HOLD JMP o;
m)  ALU,A+B A,<a1> B,<b1> TYP,<t3> D,<d1> NEXT* HOLD;
n)  ALU,XOR A,<a2> B,<b2> TYP,<t3> D,<d2> NEXT* HOLD;
o)  NEXT* LOAD;
```

The result from the ALU operation in microinstruction a) gives either true or false micro zero.

Result = 0 gives true inverted = false sequence i.e. next -> c) from microcycle b). The microcode pipeline is broken, and this costs time.

Result >< 0 gives false inverted sequence i.e. jump to microaddress m) from microinstruction b). The microcode pipeline is not broken.

If microinstruction c) is identical to m) and d) is identical to

n), the microcode could be executed unconditional. EXUC should then be inserted in b) and m) and the c) and d) microinstructions should be removed.

8.3 Conditional ALU Operation

Conditional ALU operation is used to select either true or false ALU operation in a microinstruction.

The result of a test condition selected in current microinstruction with the result from the previous micro-instruction affecting the selected condition, determines true or false ALU operation.

Example:

```
a)  ALU,A-B A,<ao> B,<bo> TYP,<tt>  NEXT* HOLD;
b)  ALU,<func> ALUF,<func>   COND,MSGN
    A,<ao>  B,<bo> TYP,<tt> D,<dest> NEXT* HOLD;
```

The result from the ALU operation in microinstruction a) gives either true or false micro sign.

Micro sign i.e. <ao> < <bo> gives true ALU-function in micro-instruction b).

Not micro sign i.e. <ao> >= <bo> gives false ALU-function in microinstruction b).

8.4 Condition Save (CSAVE)

By using the CSAVE command, any test condition may be saved for later use. Any of the two last saved conditions, true or false, may be selected for test in a later microinstruction by selecting saved condition 1 or 2.

The saved test conditions function as a one bit wide stack, where a new test object may be continuously pushed, loosing the bottom of the stack.

CSAVE saves the result of the condition set in the current micro-instruction with the result from the previous microinstruction affecting the selected condition.

Example:

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

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

The result from the operation in microinstruction a) and selected test object in microinstruction b) is saved in microinstruction b).

Saved condition 1 (COND,SAVC1) is selected in microinstruction m) and is used for test in this microinstruction.

CHAPTER 9 CONTROL OF STATUS BITS

The status register may be controlled by writing the result of an ALU operation into the status register, or by one of the commands saving the status information from an operation, either from the ALU or from the AAP.

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

Status bits control:

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
ST,SAVM	Save mixed status from integer multiply
ST,ACCM	Save and accumulate mixed status
ST,ACCA	Save and accumulate status from ALU operation
ST,ACCF	Save and accumulate status from AAP operation
ST,LOAD	Load F-bus to the status register

Save of mixed status, ST,SAVM and ST,ACCM, results in overflow taken from the AAP, and zero and sign taken from the ALU.

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

K,ZRO	: K <- 0.
K,ONE	: K <- 1.
K,1IFZ	: K <- 1 if MZRO is true. DR <- 0.

The status register S1, is an OR'ed status read from the MIC, the IDU and the ALU gate arrays.

- Bits in S1 residing in the MIC: 37 to 32, 24, 20, 4 to 0.
- Bits in S1 residing in the IDU: 30 to 25, 23 to 21.
- Bits in S1 residing in the ALU: 31, 17 to 5.

A detailed description of the status bits is given in the manual ND-5000 Hardware Description (ND-05.020).

CHAPTER 10 ADDRESS ARITHMETIC

The only operation possible in the address arithmetic is an add operation on the address A-operand and the B-operand. Both A and B operands may be zero, thus allowing only one source of address. In addition addresses saved in the EA0 and EA1 registers may be enabled from the output of the address arithmetic.

The address arithmetic may either be controlled by the operand cache (OCA) or from the microcode. When controlled by the OCA, the address arithmetic is activated by the microprogram when a general operand is requested. This is usualy done together with read of a general operand, but may in some cases be done before the read cycle for the operand is entered.

Since the ND-5000 is byte addressed, the microprogram must handle the address arithmetic according to the data type in question when using the address arithmetic.

INPUT TO THE ADDRESS ARITHMETIC:

Address A-operand:

AA,0	Zero selected as input
AA,DATA	Data in register as input
AA,DISP	Displacement register as input
AA,EA0	EA0 register as input
AA,EA1	EA1 register as input
AA,EA2	EA2 register as input
AA,EA3	EA3 register as input
AA,MARG	8 bits argument, sign extended to 32 bits

Address B-operand:

AB,0	Zero as input
AB,B	Base register as input
AB,IX1	Index register 1
AB,IX2	Index register 2
AB,IX3	Index register 3
AB,IX4	Index register 4
AB,MARG	8 bits argument with sign extension
AB,R	Record register as input

Direct output address possibility:

AB,ADR	Previous address
AB,ADR+4	Previous address + 4
AB,EA1DIR	Use EA1

In order to make access easier when indexing data elements with type different from byte, index registers used on the B-operand of the address arithmetic input may be scaled according to data type of the instruction. The index registers may also be scaled

independant of data type.

Address B-operand input with scaling from instruction data type:

AB,X1ORS Desc(address)(I1), Index register 1 scaled according to data type of instruction.

AB,X2ORS Desc(address)(I2), Index register 2 scaled according to data type of instruction.

AB,X3ORS Desc(address)(I3), Index register 3 scaled according to data type of instruction.

AB,X4ORS Desc(address)(I4), Index register 4 scaled according to data type of instruction.

Address B-operand index scaling for data type. Data type:

IX*1	Index register scaled by 1	Byte
IX*2	Index register scaled by 2	Half word
IX*4	Index register scaled by 4	Word,sing.float
IX*8	Index register scaled by 8	Doubl.float
IX/8	Index register scaled by 1/8	Bit
IX/16	Index register scaled by 16	80-bit floating

Output of the address arithmetic is always latched in the EA0 register. In addition, the microprogram may control address arithmetic output to be saved in either EA1, EA2 or EA3. This is done by the EA<nr>SAVE commands to save result from the address arithmetic to the specified effective address register. 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.

The microprogrammer may hold base addresses in either EA1, EA2 or EA3 registers to generate addresses relative to these. When a fetch operation is started, only the EA0 register is changed, unless an EA<nr>SAVE command is used in the same microinstruction.

When the DAC is busy with the calculation of an OCA controlled memory request, and the microprogram wants to perform a new memory request in the next microinstruction, new address may be generated from microcode, but only OCA controlled, while OCA controls the DAC. Only a limited number of address arithmetic activate commands may then be used. Only AB,ADR, AB,ADR+4 and the AB,EA1DIR may be used and will cause the address to be presented by the DAC in the next cycle.

CHAPTER 11 THE ND-5000 MICROASSEMBLER

11.1 MICROINSTRUCTION

The ND-5000 microinstruction is a combination of the ND-5000 mnemonic symbols, constants or defined symbols separated with space. The microinstruction is terminated by ';' and may occupy several lines of symbols. Characters of a line after '%' are taken as comments to the microinstruction.

11.2 MNEMONIC SYMBOLS

The mnemonic symbols are direct functions or operator selectors. One special case, related to the sequencing of the microprogram, is treated within the microcode assembler. In the case that the microprogram jump address is free, the source code may use the mnemonic symbol NEXT immediately followed by '*' (NEXT*). This will cause the microcode assembler to generate a JMP *+1 inserted as sequence control in the microinstruction. If the large argument field is used for values different from 0 this will cause the error message:

ORING REJECTED DUE TO OVERLAPPING MNEMONICS

11.3 CONSTANTS

Constants used in the microprogram must be octal integers (or optional: hexadecimal digits). The constants are either used in the mini argument field, the short argument field, the long argument field, the microprogram address field, or as microprogram address modifier

Mini argument is specified by one 8-bit integer. The value of the constant is placed in the mini argument field during assembly (control store bits 8-0). During execution in the ND-5000 the mini argument is sign extended to 32 bits by A,MARG, AA,MARG and AB,MARG.

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-5000 the short argument is sign extended to 32 bits by A,SARG.

Long argument is specified by two 16-bit integers separated by ',' or by one 32-bit integer. The value of the constant is placed in the long argument field during assembly (control store bits 31-0).

Microprogram address may be selected either by referencing a label or by specifying a long argument where the most significant part is taken as microprogram address (control store bits 31-16). Referencing a label will cause the value of the label to be placed in the microprogram address field.

The microprogram address may be modified by a 16-bit integer terminated by '/' located as the first element of a micro-instruction. Current microprogram address is set equal to integer specified.

11.4 Defined Symbols

Labels are defined by up to 16 alphanumeric characters terminated by ':'. The label must be located as the first symbol of a micro-instruction. Value of the label is current control store address. The 16 first characters are significant. Reference to a label will cause the value of the label to be placed in the microprogram address field (control store bits 31-16).

11.5 The Assembler

The assembler works on mass storage files and may handle 40 input files and may give 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 (SAM-MNE-SYMBOLS:SYMB) and the mnemonic value file (SAM-MNE-VALUES:DATA). The input and output files are of type :SYMB except the object file which is of type :DATA.

The output files with content are:

- * Required: Undefined symbols list-file contains all undefined symbols.
- * Required: Error list-file contains errors detected during assembly.
- * Required: Object file contains input to control store.

- List-file contains symbolic list of the microprogram with control store address.
- Unsorted label list-file contains all labels defined with corresponding microprogram address.

The assembler may also be used for converting the object file to an octal list file of the microprogram.

The assembler also has a built in mnemonic editor in order to edit mnemonic symbols, values and comments as well as listing the mnemonic symbol table. File name of the mnemonic comments is assumed to be: SAM-MNE-COMMENT:SYMB.

The assembler has a "help" command which provides information about possible commands.

Example of running the ND-5000 microassembler:

```
@SAM-MICRO-ASSEM
  The ND-5000 micro-code assembler 2.0  September 1983

Microassembler * ASSEMBLE-MICRO-PROGRAM
Give filename of entry no. 1   : SAM-MICRO-01-00:SYMB
Give filename of entry no. 2   : SAM-MICRO-02-00:SYMB
Give filename of entry no. 3   :

Undefined symbols list-file    : SAM-MICRO-UDEFV:SYMB
Error list-file                : SAM-MICRO-ERROR:SYMB
Object file                    : SAM-MICRO-OBJEC:DATA
List-file                      : SAM-MICRO-SLIST:SYMB
Unsorted label list-file       : SAM-MICRO-USORT:SYMB

Length of microprogram in kilowords (each 128 bits): 8

    100 Words assembled
    200 Words assembled

    100 Items in udfv table recognized
      0 Diagnostics have been detected

All program functions terminated
microassembler * EXIT

@
```

11.5.1 Error messages from the microassembler

The error messages from the ND-5000 microassembler give the microprogram address where an error is detected, ERROR AT CLC ‹octal number›, followed by additional error information. The different error messages are listed below together with a short explanation. At the 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 upper control store address is reached for this size of control store.

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

ERROR AT CLC XXXXXXB
ILLEGAL CHARACTER IN ROUTINE "TRANSFORM"
Octal number is not at source file.

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

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

ERROR AT CLC XXXXXXB
CLC MODIFIER ERROR
Error in modifying the microprogram address.

ERROR AT CLC XXXXXXB
TOO MANY MNEMONICS BETWEEN SEMICOLONS
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
Try to write into a previously used microprogram 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. Rest of the microinstruction 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. May be caused by defined symbols with more than 16 characters.

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

```
ERROR AT CLC  XXXXXXB
ALREADY DEFINED:
Label already defined.
```

CHAPTER 12 USER INSTRUCTIONS FOR MICROPROGRAM EXTENSIONS

Some instruction codes in the ND-5000 are available for user written microprogram. This means that an instruction code has an entry in the ND-5000 microprogram, but is not used. 'Not used' means that the instructions generate an illegal instruction code. These instruction codes may be used for special microprogramming to implement new functions.

Three instruction codes are used for controlling the built in timer, and four instruction codes are used to control the built in hardware trace module. These are marked as used in the table below.

The instructions available may be divided into three different groups, depending on prefetch and operand decoding. These groups are divided into subgroups, one group for each data type. A general description of the different types of instructions is also given. The instructions are listed with instruction code, default data type for the operand and the entry point in the microprogram.

The space available for user written microprogram, depends on the microprogram 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 microprogram is empty or contains only a jump to microprogram address 200. Space available for user written microcode will be defined on the program description sheet for the different microprogram versions.

For a detailed description of the space available for user written microprogram, the program description sheet for the product should be considered.

12.1 CLASSIFICATION

Classification of the ND-5000 user instructions is done depending on 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

12.2 INSTRUCTION GROUP 1

The following user instructions are available in group 1.

Instruction code	Instruction type	Microprogram entry	
236	W EXT	1637	Read Mic.Adr. trace
237	W EXT	1640	Read D/I.Adr. trace
177004	W EXT	1641	Read status of tracer
177005	W EXT	1642	Load control tracer
177006	W EXT	1643	
177007	W EXT	1644	
177036	W EXT	1645	
177037	W EXT	1646	Timer interrupt.
177436	W EXT	1647	Timer clear.
177437	W EXT	1650	Timer read.

12.3 INSTRUCTION GROUP 2

The prefetch processor is for group 2, fetching one memory operand. This is for data type byte, halfword, word, and single floating point. For the data type double floating point, address generation of the extension part of the double floating point operand is required in order to read the least significant part of the operand.

The following user instructions are available in group 2.

Instruction code	Instruction type	Microprogram entry
177460	By EXT <operand/r/BY>	1721
177461	By EXT <operand/r/BY>	1723
177462	By EXT <operand/r/BY>	1725
177463	By EXT <operand/r/BY>	1727
177464	By EXT <operand/r/BY>	1731
177465	By EXT <operand/r/BY>	1733
177466	By EXT <operand/r/BY>	1735
177467	By EXT <operand/r/BY>	1737

Instruction code	Instruction type	Microprogram entry
177470	H EXT <operand/r/H>	1741
177471	H EXT <operand/r/H>	1743
177472	H EXT <operand/r/H>	1745
177473	H EXT <operand/r/H>	1747
177474	H EXT <operand/r/H>	1751
177475	H EXT <operand/r/H>	1753
177476	H EXT <operand/r/H>	1755
177477	H EXT <operand/r/H>	1757

Instruction code	Instruction type	Microprogram entry	
177500	W EXT <operand/r/W>	1761	Rphs
177501	W EXT <operand/r/W>	1763	Wphs
177502	W EXT <operand/r/W>	1765	CAD :=
177503	W EXT <operand/r/W>	1767	
177504	W EXT <operand/r/W>	1771	
177505	W EXT <operand/r/W>	1773	used in AX
177506	W EXT <operand/r/W>	1775	used in AX
177507	W EXT <operand/r/W>	1777	used in AX

Instruction code	Instruction type	Microprogram entry	
177510	F EXT <operand/r/F>	2001	
177511	F EXT <operand/r/F>	2003	
177512	F EXT <operand/r/F>	2005	
177513	F EXT <operand/r/F>	2007	
177514	F EXT <operand/r/F>	2011	
177515	F EXT <operand/r/F>	2013	used in AX
177516	F EXT <operand/r/F>	2015	used in AX
177517	F EXT <operand/r/F>	2017	used in AX

Instruction code	Instruction type	Microprogram entry
177520	D EXT <operand/r/D>	2021
177521	D EXT <operand/r/D>	2023
177522	D EXT <operand/r/D>	2025
177523	D EXT <operand/r/D>	2027
177524	D EXT <operand/r/D>	2031
177525	D EXT <operand/r/D>	2033
177526	D EXT <operand/r/D>	2035
177527	D EXT <operand/r/D>	2037

12.4 INSTRUCTION GROUP 3

For instructions in group 3, a general operand, either a constant from program area, a register or a memory operand is fetched. In addition, OR-logic selection of register is possible.

The following user instructions are available in group 3.

Instruction code	Instruction type	Microprogram entry
177300 - 177303	Byn EXT <operand/r/BY>	1651
177304 - 177307	Byn EXT <operand/r/BY>	1653
177310 - 177313	Byn EXT <operand/r/BY>	1655
177314 - 177317	Byn EXT <operand/r/BY>	1657

Instruction code	Instruction type	Microprogram entry
177320 - 177323	Hn EXT <operand/r/H>	1661
177324 - 177327	Hn EXT <operand/r/H>	1663
177330 - 177333	Hn EXT <operand/r/H>	1665
177334 - 177337	Hn EXT <operand/r/H>	1667

Instruction code	Instruction type	Microprogram entry
177340 - 177343	Wn EXT <operand/r/W>	1671
177344 - 177347	Wn EXT <operand/r/W>	1673
177350 - 177353	Wn EXT <operand/r/W>	1675
177354 - 177357	Wn EXT <operand/r/W>	1677

Instruction code	Instruction type	Microprogram entry
177360 - 177363	Fn EXT <operand/r/F>	1701
177364 - 177367	Fn EXT <operand/r/F>	1703
177370 - 177373	Fn EXT <operand/r/F>	1705
177374 - 177377	Fn EXT <operand/r/F>	1707

Instruction code	Instruction type	Microprogram entry
177440 - 177443	Dn EXT <operand/r/D>	1711
177444 - 177447	Dn EXT <operand/r/D>	1713
177450 - 177453	Dn EXT <operand/r/D>	1715
177454 - 177457	Dn EXT <operand/r/D>	1717

APPENDIX A ALPHABETIC LIST OF MNEMONIC SYMBOLS

1	ALU,FZRO	FORCE ZERO ALU OUTPUT
2	ALU,ADIRC	ALU OUTPUT COMPLEMENTED
3	ALU,AND	LOGICAL AND OF A AND B
4	ALU,ANDCB	LOGICAL AND OF A AND B COMPLEMENTED
5	ALU,A	A OPERAND DIRECT THROUGH THE ALU
6	ALU,A+1	ADD 1 TO A OPERAND
7	ALU,XOR	LOGICAL EXCLUSIVE OR OF A AND B
8	ALU,ANDCA	LOGICAL AND OF A COMPLEMENTED AND B
9	ALU,OR	LOGICAL OR OF A AND B
10	ALU,A-1	DECREMENT A OPERAND
11	ALU,A,/2	FBUS = ALU.OUTPUT/2; FBUS(31) = CARRY
12	ALU,A-B	A MINUS B OPERAND
13	ALU,A-B-1	A MINUS B OPERAND MINUS 1
14	ALU,A-B-1+C	A MINUS B OPERAND MINUS 1 ADDED CARRY
15	ALU,A-B,*2	FBUS = ALU.OUTPUT*2; FBUS(00) = 0
16	ALU,A-B-1,*2	FBUS = ALU.OUTPUT*2; FBUS(00) = 0
17	ALU,A+B,/2	FBUS = ALU.OUTPUT/2; FBUS(31) = CARRY
18	ALU,A+B	A OPERAND ADDED B OPERAND
19	ALU,A+B+1	A OPERAND ADDED B OPERAND ADDED 1
20	ALU,B-A	B OPERAND MINUS A OPERAND
21	ALU,B-A-1	B OPERAND MINUS A OPERAND MINUS 1
22	ALU,A+B,*2	FBUS = ALU.OUTPUT*2; FBUS(00) = 0
23	CRY,ONE	ONE AS CARRY
24	CRY,C	C FROM STATUS AS CARRY
25	CRY,MC	MICRO CARRY AS CARRY
26	ALUF,FZRO	FORCE ZERO ALU OUTPUT
27	ALUF,ADIRC	ALU OUTPUT COMPLEMENTED
28	ALUF,AND	LOGICAL AND OF A AND B
29	ALUF,ANDCB	LOGICAL AND OF A AND B COMPLEMENTED
30	ALUF,A	A OPERAND DIRECT THROUGH THE ALU
31	ALUF,A+1	ADD 1 TO A OPERAND
32	ALUF,XOR	LOGICAL EXCLUSIVE OR OF A AND B
33	ALUF,ANDCA	LOGICAL AND OF A COMPLEMENTED AND B
34	ALUF,OR	LOGICAL OR OF A AND B
35	ALUF,A-1	DECREMENT A OPERAND
36	ALUF,A,/2	FBUS = ALU.OUTPUT/2; FBUS(31) = CARRY
37	ALUF,A-B	A MINUS B OPERAND
38	ALUF,A-B-1	A MINUS B OPERAND MINUS 1
39	ALUF,A-B-1+C	A MINUS B OPERAND MINUS 1 ADDED CARRY
40	ALUF,A-B,*2	FBUS = ALU.OUTPUT*2; FBUS(00) = 0
41	ALUF,A-B-1,*2	FBUS = ALU.OUTPUT*2; FBUS(00) = 0
42	ALUF,A+B,/2	FBUS = ALU.OUTPUT/2; FBUS(31) = CARRY
43	ALUF,A+B	A OPERAND ADDED B OPERAND
44	ALUF,A+B+1	A OPERAND ADDED B OPERAND ADDED 1
45	ALUF,B-A	B OPERAND MINUS A OPERAND
46	ALUF,B-A-1	B OPERAND MINUS A OPERAND MINUS 1
47	ALUF,B-A-1+C	B OPERAND MINUS A OPERAND MINUS 1 ADDED CARRY
48	ALUF,A+B,*2	FBUS = ALU.OUTPUT*2; FBUS(00) = 0
49	CRYF,ONE	ONE AS CARRY
50	CRYF,C	C FROM STATUS AS CARRY
51	CRYF,MC	MICRO CARRY AS CARRY

52	EXUC	EXECUTE UNCONDITIONAL
53	Q,F	Q <- ALU OUTPUT.
54	Q,Q*DIV	Q <- Q*2; Q(00) <- DIVR
55	Q,Q*LOG	Q <- Q*2; Q(00) <- 0
56	Q,Q/ARI	Q <- Q/2; Q(SIGN.BIT) <- Q(SIGN.BIT)
57	Q,Q/LOG	Q <- Q/2; Q(SIGN.BIT) <- 0
58	Q,Q/ROT	Q <- Q/2; Q(SIGN.BIT) <- Q(00)
59	Q,Q*ROT	Q <- Q*2; Q(00) <- Q(SIGN.BIT)
60	EXPISO	ISOLATE FLOATING EXPONENT; FBUS(8-0) <- F(30-22)
61	IXADJ	INDEX COUNTER INCREMENT
62	TYP,W	DATA TYPE IS WORD
63	TYP,F	DATA TYPE IS SINGLE FLOATING
64	TYP,HW	DATA TYPE IS HALF WORD
65	TYP,BY	DATA TYPE IS BYTE
66	TYP,BI	DATA TYPE IS BIT
67	TYP,DF	DATA TYPE IS DOUBLE FLOATING (64-BITS REAL)
68	TYP,DD	DATA TYPE IS 128 BITS FLOATING POINT
69	TYP,OR	DATA TYPE CONTROLED BY THE ICA
70	A,BM00	A-BUS IS BIT MASK 0
71	A,BM01	A-BUS IS BIT MASK 1
72	A,BM02	A-BUS IS BIT MASK 2
73	A,BM03	A-BUS IS BIT MASK 3
74	A,BM04	A-BUS IS BIT MASK 4
75	A,BM05	A-BUS IS BIT MASK 5
76	A,BM06	A-BUS IS BIT MASK 6
77	A,BM07	A-BUS IS BIT MASK 7
78	A,BM10	A-BUS IS BIT MASK 10
79	A,BM11	A-BUS IS BIT MASK 11
80	A,BM12	A-BUS IS BIT MASK 12
81	A,BM13	A-BUS IS BIT MASK 13
82	A,BM14	A-BUS IS BIT MASK 14
83	A,BM15	A-BUS IS BIT MASK 15
84	A,BM16	A-BUS IS BIT MASK 16
85	A,BM17	A-BUS IS BIT MASK 17
86	A,BM20	A-BUS IS BIT MASK 20
87	A,BM21	A-BUS IS BIT MASK 21
88	A,BM22	A-BUS IS BIT MASK 22
89	A,BM23	A-BUS IS BIT MASK 23
90	A,BM24	A-BUS IS BIT MASK 24
91	A,BM25	A-BUS IS BIT MASK 25
92	A,BM26	A-BUS IS BIT MASK 26
93	A,BM27	A-BUS IS BIT MASK 27
94	A,BM30	A-BUS IS BIT MASK 30
95	A,BM31	A-BUS IS BIT MASK 31
96	A,BM32	A-BUS IS BIT MASK 32
97	A,BM33	A-BUS IS BIT MASK 33
98	A,BM34	A-BUS IS BIT MASK 34
99	A,BM35	A-BUS IS BIT MASK 35
100	A,BM36	A-BUS IS BIT MASK 36
101	A,BM37	A-BUS IS BIT MASK 37
102	A,X1	A-BUS IS INDEX REGISTER X1
103	A,X2	A-BUS IS INDEX REGISTER X2
104	A,X3	A-BUS IS INDEX REGISTER X3
105	A,X4	A-BUS IS INDEX REGISTER X4
106	A,A1	A-BUS IS FLOATING MOST REGISTER A1
107	A,A2	A-BUS IS FLOATING MOST REGISTER A2
108	A,A3	A-BUS IS FLOATING MOST REGISTER A3

```
109  A,A4            A-BUS IS FLOATING MOST REGISTER A4
110  A,SC1           A-BUS IS SCRATCH REGISTER 1
111  A,SC2           A-BUS IS SCRATCH REGISTER 2
112  A,SC3           A-BUS IS SCRATCH REGISTER 3
113  A,SC4           A-BUS IS SCRATCH REGISTER 4
114  A,E1            A-BUS IS FLOATING LEAST REGISTER E1
115  A,E2            A-BUS IS FLOATING LEAST REGISTER E2
116  A,E3            A-BUS IS FLOATING LEAST REGISTER E3
117  A,E4            A-BUS IS FLOATING LEAST REGISTER E4
118  A,SC5           A-BUS IS SCRATCH REGISTER 5
119  A,SC6           A-BUS IS SCRATCH REGISTER 6
120  A,SC7           A-BUS IS SCRATCH REGISTER 7
121  A,SC10          A-BUS IS SCRATCH REGISTER 10
122  A,SC11          A-BUS IS SCRATCH REGISTER 11
123  A,SC12          A-BUS IS SCRATCH REGISTER 12
124  A,SC13          A-BUS IS SCRATCH REGISTER 13
125  A,SC14          A-BUS IS SCRATCH REGISTER 14
126  A,DATA          A-BUS IS DATA INPUT REGISTER
127  A,BMLC          A-BUS IS BIT MASK FROM LOOP COUNTER
128  A,AAPRES        A-BUS IS AAP RESULT
129  A,Q             A-BUS IS Q-REGISTER
130  A,ALU,STS       A-BUS IS ALU STATUS BITS
131  A,ALU,TE        A-BUS IS ALU TRAP ENABLE BITS
132  A,PXBM          A-BUS IS POST-INDEX BIT-MASK
133  A,IMM,PSTP      A-BUS IS IMM PSTP REGISTER
134  A,DMM,PSTP      A-BUS IS DMM PSTP REGISTER
135  A,IMM,PUWP      A-BUS IS IMM PUWP REGISTER
136  A,DMM,PUWP      A-BUS IS DMM PUWP REGISTER
137  A,IMM,LA        A-BUS IS IMM LA REGISTER
138  A,DMM,LA        A-BUS IS DMM LA REGISTER
139  A,IMM,WR        A-BUS IS IMM WR REGISTER
140  A,DMM,WR        A-BUS IS DMM WR REGISTER
141  A,IMM,CAP       A-BUS IS IMM CAPABILITY
142  A,DMM,CAP       A-BUS IS DMM CAPABILITY
143  A,IMM,PS        A-BUS IS IMM PS REGISTER
144  A,DMM,PS        A-BUS IS DMM PS REGISTER
145  A,IMM,PHS       A-BUS IS IMM PHS REGISTER
146  A,DMM,PHS       A-BUS IS DMM PHS REGISTER
147  A,IMM,DOM       A-BUS IS IMM DOM REGISTER
148  A,DMM,DOM       A-BUS IS DMM DOM REGISTER
149  A,IMM,MEM       A-BUS IS INSTRUCTION MEMORY
150  A,DMM,MEM       A-BUS IS DATA MEMORY
151  A,IMM,PHYS      A-BUS IS INSTRUCTION PHYSICAL ADDR.
152  A,DMM,PHYS      A-BUS IS DATA PHYSICAL ADDRESS
153  A,IMM,STS       A-BUS IS IMM STS REGISTER
154  A,DMM,STS       A-BUS IS DMM STS REGISTER
155  A,IMM,ADOM      A-BUS IS IMM ADOM REGISTER
156  A,DMM,ADOM      A-BUS IS DMM ADOM REGISTER
157  A,SPEC,MOD      A-BUS IS MODUS-REGISTER
158  A,SPEC,AOB      A-BUS IS AOB-REGISTER
159  A,SPEC,IAR      A-BUS IS IAR-REGISTER
160  A,SPEC,OC,DP    A-BUS IS DPA-PART OF OC
161  A,SPEC,OC,AD    A-BUS IS NADDR-PART OF OC
162  A,SPEC,OC,CO    A-BUS IS CONTROL-PART OF OC
163  A,SPEC,AC       A-BUS IS address-CACHE
164  A,SPEC,IC       A-BUS IS INSTRUCTION-CACHE
165  A,SPEC,OLAH2    A-BUS IS OLAH2-REGISTER
```

```
166  A,SPEC,AFLAG     A-BUS IS ACCP-FLAG-REGISTER
167  A,SPEC,AOBASR    A-BUS IS COMM.-REGISTER
168  A,SPEC,IRL       A-BUS IS INSTR-READ-LATCH
169  A,SPEC,DACR      A-BUS IS DAC-REGISTER
170  A,SPEC,ACH       A-BUS IS AC-HOLD-REGISTER
171  A,SPEC,DLAH      A-BUS IS DLA-HOLD-REGISTER
172  A,SPEC,LA        A-BUS IS LA-LATCH
173  A,SPEC,FLA       A-BUS IS FORWARD-LA-LATCH
174  A,SPEC,DPSDOM    A-BUS IS DATA PS/DOM
175  A,SPEC,IPSDOM    A-BUS IS INSTRUCTION PS/DOM
176  A,SPEC,IDIR      A-BUS IS INSTR-CACHE-DIR
177  A,SPEC,DCALA     A-BUS IS DATA-CACHE LA
178  A,SPEC,CSTRC     A-BUS IS
179  A,SPEC,DCADAT    A-BUS IS DATA-CACHE DATA
180  A,SPEC,STRACE    A-BUS IS STRACE
181  A,SPEC,ITRACE    A-BUS IS ITRACE
182  A,SPEC,ATRACE    A-BUS IS ATRACE
183  A,SPEC,DTRACE    A-BUS IS DTRACE
184  A,SPEC,CTRACE    A-BUS IS CTRACE
185  A,MIC,MISTS      A-BUS IS MIC STATUS REGISTER
186  A,MIC,VECT       A-BUS IS MIC VECTOR REGISTER
187  A,MIC,RFA1       A-BUS IS RF-ADDRESS REGISTER 1
188  A,MIC,RFA2       A-BUS IS RF-ADDRESS REGISTER 2
189  A,MIC,STS        A-BUS IS MIC STATUS BITS
190  A,MIC,TE         A-BUS IS MIC TRAP ENABLE BITS
191  A,MIC,CURR       A-BUS IS MIC CURR REGISTER
192  A,MIC,CNT32      A-BUS IS MIC 32-BIT COUNTER
193  A,RF1            A-BUS IS REG.FILE POINTED TO BY RF1 REGISTER
194  A,RF2            A-BUS IS REG.FILE POINTED TO BY RF2 REGISTER
195  A,RF1D           A-BUS IS REG.FILE POINTED TO BY RF1,RF1 DECREMENT
196  A,RF2D           A-BUS IS REG.FILE POINTED TO BY RF2,RF2 DECREMENT
197  A,SRF0           A-BUS IS SRF-WORD 0
198  A,SRF1           A-BUS IS SRF-WORD 1
199  A,SRF2           A-BUS IS SRF-WORD 2
200  A,SRF3           A-BUS IS SRF-WORD 3
201  A,SRF4           A-BUS IS SRF-WORD 4
202  A,SRF5           A-BUS IS SRF-WORD 5
203  A,SRF6           A-BUS IS SRF-WORD 6
204  A,SRF7           A-BUS IS SRF-WORD 7
205  A,SRF00          A-BUS IS SRF-WORD 0
206  A,SRF01          A-BUS IS SRF-WORD 1
207  A,SRF02          A-BUS IS SRF-WORD 2
208  A,SRF03          A-BUS IS SRF-WORD 3
209  A,SRF04          A-BUS IS SRF-WORD 4
210  A,SRF05          A-BUS IS SRF-WORD 5
211  A,SRF06          A-BUS IS SRF-WORD 6
212  A,SRF07          A-BUS IS SRF-WORD 7
213  A,SRF10          A-BUS IS SRF-WORD 10
214  A,SRF11          A-BUS IS SRF-WORD 11
215  A,SRF12          A-BUS IS SRF-WORD 12
216  A,SRF13          A-BUS IS SRF-WORD 13
217  A,SRF14          A-BUS IS SRF-WORD 14
218  A,SRF15          A-BUS IS SRF-WORD 15
219  A,SRF16          A-BUS IS SRF-WORD 16
220  A,SRF17          A-BUS IS SRF-WORD 17
221  A,IDU,TE         A-BUS IS MIC TRAP ENABLE REGISTER
222  A,IDU,HL         A-BUS IS IDU HL REGISTER
```

```
223  A,IDU,LL        A-BUS IS IDU LL REGISTER
224  A,IDU,LIMC      A-BUS IS IDU LIMIT CONTROL REGISTER
225  A,IDU,B2        A-BUS IS IDU BUFFER-2
226  A,IDU,STS       A-BUS IS IDU STATUS REGISTER
227  A,IDU,DPA       A-BUS IS DPA-BUS-REGISTER
228  A,IAC,ILAR      A-BUS IS IAC LA-REGISTER
229  A,IAC,S         A-BUS IS IAC SCRATCH REGISTER
230  A,IAC,Y         A-BUS IS IAC Y REGISTER
231  A,IAC,SP        A-BUS IS IAC SP REGISTER
232  A,IAC,L         A-BUS IS IAC L (LINK) REGISTER
233  A,IAC,P         A-BUS IS IAC P REGISTER
234  A,IAC,NPC       A-BUS IS IAC NPC REGISTER
235  A,DAC,DLAR      A-BUS IS DAC LA-REGISTER
236  A,DAC,EA0       A-BUS IS DAC EA0 REGISTER
237  A,DAC,EA1       A-BUS IS DAC EA1 REGISTER
238  A,DAC,EA2       A-BUS IS DAC EA2 REGISTER
239  A,DAC,EA3       A-BUS IS DAC EA3 REGISTER
240  A,MARG          A-BUS IS MINI ARGUMENT
241  A,DAC,B         A-BUS IS DAC B REGISTER
242  A,DAC,R         A-BUS IS DAC R REGISTER
243  A,SARG          A-BUS IS SHORT ARGUMENT
244  A,LARG          A-BUS IS LONG ARGUMENT
245  B,X1            B-BUS IS INDEX REGISTER X1
246  B,X2            B-BUS IS INDEX REGISTER X2
247  B,X3            B-BUS IS INDEX REGISTER X3
248  B,X4            B-BUS IS INDEX REGISTER X4
249  B,A1            B-BUS IS FLOATING MOST REGISTER A1
250  B,A2            B-BUS IS FLOATING MOST REGISTER A2
251  B,A3            B-BUS IS FLOATING MOST REGISTER A3
252  B,A4            B-BUS IS FLOATING MOST REGISTER A4
253  B,SC1           B-BUS IS FLOATING SCRATCH REGISTER SC1
254  B,SC2           B-BUS IS FLOATING SCRATCH REGISTER SC2
255  B,SC3           B-BUS IS FLOATING SCRATCH REGISTER SC3
256  B,SC4           B-BUS IS FLOATING SCRATCH REGISTER SC4
257  B,E1            B-BUS IS FLOATING LEAST REGISTER E1
258  B,E2            B-BUS IS FLOATING LEAST REGISTER E2
259  B,E3            B-BUS IS FLOATING LEAST REGISTER E3
260  B,E4            B-BUS IS FLOATING LEAST REGISTER E4
261  B,SC5           B-BUS IS SCRATCH REGISTER SC5
262  B,SC6           B-BUS IS SCRATCH REGISTER SC6
263  B,SC7           B-BUS IS SCRATCH REGISTER SC7
264  B,SC10          B-BUS IS SCRATCH REGISTER SC10
265  B,SC11          B-BUS IS SCRATCH REGISTER SC11
266  B,SC12          B-BUS IS SCRATCH REGISTER SC12
267  B,SC13          B-BUS IS SCRATCH REGISTER SC13
268  B,SC14          B-BUS IS SCRATCH REGISTER SC14
269  B,LC            B-BUS IS LOOP COUNTER (LC)
270  B,Q             B-BUS IS Q-REGISTER
271  B,BCD           B-BUS IS BCD CORRECTION (Q/4 OR Q/8)
272  B,IXC           B-BUS IS INDEX-COUNTERS
273  D,X1            DESTINATION IS INDEX REGISTER X1
274  D,X2            DESTINATION IS INDEX REGISTER X2
275  D,X3            DESTINATION IS INDEX REGISTER X3
276  D,X4            DESTINATION IS INDEX REGISTER X4
277  D,A1            DESTINATION IS FLOATING MOST REGISTER A1
278  D,A2            DESTINATION IS FLOATING MOST REGISTER A2
279  D,A3            DESTINATION IS FLOATING MOST REGISTER A3
```

```
280  D,A4             DESTINATION IS FLOATING MOST REGISTER A4
281  D,SC1            DESTINATION IS SCRATCH REGISTER SC1
282  D,SC2            DESTINATION IS SCRATCH REGISTER SC2
283  D,SC3            DESTINATION IS SCRATCH REGISTER SC3
284  D,SC4            DESTINATION IS SCRATCH REGISTER SC4
285  D,E1             DESTINATION IS FLOATING LEAST REGISTER E1
286  D,E2             DESTINATION IS FLOATING LEAST REGISTER E2
287  D,E3             DESTINATION IS FLOATING LEAST REGISTER E3
288  D,E4             DESTINATION IS FLOATING LEAST REGISTER E4
289  D,SC5            DESTINATION IS SCRATCH REGISTER SC5
290  D,SC6            DESTINATION IS SCRATCH REGISTER SC6
291  D,SC7            DESTINATION IS SCRATCH REGISTER SC7
292  D,SC10           DESTINATION IS SCRATCH REGISTER SC10
293  D,SC11           DESTINATION IS SCRATCH REGISTER SC11
294  D,SC12           DESTINATION IS SCRATCH REGISTER SC12
295  D,SC13           DESTINATION IS SCRATCH REGISTER SC13
296  D,SC14           DESTINATION IS SCRATCH REGISTER SC14
297  D,NONE           NO DESTINATION
298  D,IXC            DESTINATION IS INDEX-COUNTERS CLEAR
299  D,LC             DESTINATION IS LOOP COUNTER (LC)
300  D,SPEC,MOD       WRITE MODUS REGISTER
301  D,SPEC,AIB       WRITE ACCP-INPUT-BUFFER
302  D,SPEC,DCADAT    WRITE DATA-CACHE DATA
303  D,SPEC,OC,DP     WRITE DPA-PART OF OC
304  D,SPEC,OC,AD     WRITE NADDR-PART OF OC
305  D,SPEC,OC,CO     WRITE CONTROL-PART OF OC
306  D,SPEC,AC        WRITE ADDRESS-CACHE
307  D,SPEC,IC        WRITE INSTRUCTION-CACHE
308  D,SPEC,MIB       WRITE MIB-REGISTER
309  D,SPEC,TRPARM    TRAP-ARM
310  D,SPEC,TRPCLR    TRAP-CLEAR
311  D,SPEC,CC        WRITE CONTROL-WORD-CACHE
312  D,SPEC,LA        WRITE LA-REGISTER
313  D,SPEC,FLA       WRITE FORWARD-LA-REGISTER
314  D,SPEC,CLDCA     CLEAR DATA-CACHE
315  D,SPEC,CLICA     CLEAR INSTRUCTION-CACHE
316  D,SPEC,CTRACE    WRITE CTRACE
317  D,DMM,PSTP       DESTINATION IS IMM PSTP REGISTER
318  D,IMM,PSTP       DESTINATION IS DMM PSTP REGISTER
319  D,MM,PSTP        DESTINATION IS IMM AND DMM PSTP REGISTER
320  D,DMM,PUWP       DESTINATION IS IMM PUWP REGISTER
321  D,IMM,PUWP       DESTINATION IS DMM PUWP REGISTER
322  D,MM,PUWP        DESTINATION IS IMM AND DMM PUWP REGISTER
323  D,DMM,LA         DESTINATION IS IMM LA REGISTER
324  D,IMM,LA         DESTINATION IS DMM LA REGISTER
325  D,MM,LA          DESTINATION IS IMM AND DMM LA REGISTER
326  D,DMM,WR         DESTINATION IS IMM WR REGISTER
327  D,IMM,WR         DESTINATION IS DMM WR REGISTER
328  D,MM,WR          DESTINATION IS IMM AND DMM WR REGISTER
329  D,DMM,CAP        DESTINATION IS IMM CAPABILITY REGISTER
330  D,IMM,CAP        DESTINATION IS DMM CAPABILITY REGISTER
331  D,MM,CAP         DESTINATION IS IMM AND DMM CAP REGISTERS
332  D,DMM,PS         DESTINATION IS IMM PS REGISTER
333  D,IMM,PS         DESTINATION IS DMM PS REGISTER
334  D,MM,PS          DESTINATION IS IMM AND DMM PS REGISTER
335  D,DMM,PHS        DESTINATION IS IMM PHS REGISTER
336  D,IMM,PHS        DESTINATION IS DMM PHS REGISTER
```

```
337  D,MM,PHS        DESTINATION IS IMM AND DMM PHS REGISTER
338  D,DMM,DOM       DESTINATION IS IMM DOM REGISTER
339  D,IMM,DOM       DESTINATION IS DMM DOM REGISTER
340  D,MM,DOM        DESTINATION IS IMM AND DMM DOM REGISTER
341  D,DMM,MEM       WRITE MEMORY DMM
342  D,IMM,MEM       WRITE MEMORY IMM
343  D,DMM,WTSB      DESTINATION IS IMM TSB
344  D,IMM,WTSB      DESTINATION IS DMM TSB
345  D,MM,WTSB       DESTINATION IS IMM AND DMM TSB
346  D,DMM,CTSB      IMM TSB CLEAR
347  D,IMM,CTSB      DMM TSB CLEAR
348  D,MM,CTSB       IMM AND DMM TSB CLEAR
349  D,DMM,CTRP      TRAP CLEAR AND UNLOCK THE IMM
350  D,IMM,CTRP      TRAP CLEAR AND UNLOCK THE DMM
351  D,MM,CTRP       TRAP CLEAR AND UNLOCK THE IMM AND DMM
352  D,DMM,DIRTY     DESTINATION IS DMM DIRTY-DOM-PS REGISTER
353  D,IMM,DIRTY     DESTINATION IS IMM DIRTY-DOM-PS REGISTER
354  D,MM,DIRTY      DESTINATION IS  MM DIRTY-DOM-PS REGISTER
355  D,DMM,ADOM      DESTINATION IS IMM ADOM REGISTER
356  D,IMM,ADOM      DESTINATION IS DMM ADOM REGISTER
357  D,MM,ADOM       DESTINATION IS IMM AND DMM ADOM REGISTER
358  D,MIC,MISTS     DESTINATION IS MIC STATUS REGISTER
359  D,MIC,VECT      DESTINATION IS MIC VECTOR REGISTER
360  D,RFA1          DEST. IS RF1 ADDR. REG.
361  D,RFA2          DEST. IS RF2 ADDR. REG.
362  D,MIC,STS       DESTINATION IS MIC STS-BITS
363  D,MIC,TE        DESTINATION IS MIC TRAP ENABLE BITS
364  D,MIC,BRK       DESTINATION IS MIC BREAKPOINT-REGISTER
365  D,MIC,CNT32     DESTINATION IS MIC 32-BIT COUNTER
366  D,MIC,RESTU     CLEAR STACK UNDERFLOW
367  D,RF1           DESTINATION IS REG.FILE POINTED TO BY RF1 REGISTER
368  D,RF2           DESTINATION IS REG.FILE POINTED TO BY RF2 REGISTER
369  D,RF1D          DESTINATION IS REG.FILE POINTED TO BY RF1,RF1 DECR.
370  D,RF2D          DESTINATION IS REG.FILE POINTED TO BY RF2,RF2 DECR.
371  D,SRF0          DESTINATION IS SRF-WORD 0
372  D,SRF1          DESTINATION IS SRF-WORD 1
373  D,SRF2          DESTINATION IS SRF-WORD 2
374  D,SRF3          DESTINATION IS SRF-WORD 3
375  D,SRF4          DESTINATION IS SRF-WORD 4
376  D,SRF5          DESTINATION IS SRF-WORD 5
377  D,SRF6          DESTINATION IS SRF-WORD 6
378  D,SRF7          DESTINATION IS SRF-WORD 7
379  D,SRF00         DESTINATION IS SRF-WORD 0
380  D,SRF01         DESTINATION IS SRF-WORD 1
381  D,SRF02         DESTINATION IS SRF-WORD 2
382  D,SRF03         DESTINATION IS SRF-WORD 3
383  D,SRF04         DESTINATION IS SRF-WORD 4
384  D,SRF05         DESTINATION IS SRF-WORD 5
385  D,SRF06         DESTINATION IS SRF-WORD 6
386  D,SRF07         DESTINATION IS SRF-WORD 7
387  D,SRF10         DESTINATION IS SRF-WORD 10
388  D,SRF11         DESTINATION IS SRF-WORD 11
389  D,SRF12         DESTINATION IS SRF-WORD 12
390  D,SRF13         DESTINATION IS SRF-WORD 13
391  D,SRF14         DESTINATION IS SRF-WORD 14
392  D,SRF15         DESTINATION IS SRF-WORD 15
393  D,SRF16         DESTINATION IS SRF-WORD 16
```

```
394  D,SRF17          DESTINATION IS SRF-WORD 17
395  D,IDU,TE         DESTINATION IS MIC TRAP ENABLE REGISTER
396  D,IDU,HL         DESTINATION IS IDU HL REGISTER
397  D,IDU,LL         DESTINATION IS IDU LL REGISTER
398  D,IDU,LIMC       DESTINATION IS IDU LINIT CONTROL REGISTER
399  D,IDU,CSIT       CONDITIONA SETTTING OF SINGLE INSTRUCTION-TRAP
400  D,IDU,STS        DESTINATION IS IDU STATUS REGISTER
401  D,IDU,AREG       DESTINATION IS IDU ADDRESS REGISTER
402  D,IDU,IBUF       DESTINATION IS IDU IBUF-REGISTER
403  D,IAC,NPC        DESTINATION IS IAC NPC REGISTER
404  D,IAC,P          DESTINATION IS IAC P REGISTER
405  D,IAC,L          DESTINATION IS IAC L (LINK) REGISTER
406  D,IAC,SUML       SUM IS TRANSFERRED TO IAC Y REGISTER
407  D,IAC,DPA        DESTINATION IS IAC-DPA-REGISTER
408  D,IAC,CLKNPC     LA -> NPC
409  D,IAC,CLKP       NPC -> P
410  D,IAC,CLKSP      P -> SP
411  D,DAC,R          DESTINATION IS DAC R (RECORD) REGISTER
412  D,DAC,B          DESTINATION IS DAC B (BASE) REGISTER
413  D,DAC,SUMB       SUM IS TRANSFERRED TO DAC B REGISTER
414  D,DAC,DPA        DESTINATION IS DAC DPA-REGISTER
415  K,ONE            SET K (FLAG)   1 TO K
416  K,ZRO            CLEAR K (FLAG) 0 TO K
417  K,1IFZ           SET K TO 1 IF ALU OPERATION IS 0
418  ST,SAVA          SAVE STATUS FROM ALU OPERATION
419  ST,SAVC          SAVE STATUS FROM ALU IN COMPARE
420  ST,SAVF          SAVE STATUS FROM FLOATING OPERATION
421  ST,SAVB          SAVE STATUS FROM BCD OPERATION
422  ST,LOAD          LOAD ALU STATUS.
423  ST,SAVM          SAVE MIXED STATUS FOR INTEGER MULTIPLY
424  ST,ACCA          SAVE AND ACCUMULATE ALU STATUS
425  ST,ACCM          SAVE AND ACCUMULATE MIXED STATUS
426  ST,ACCF          SAVE AND ACCUMULATE AAP STATUS
427  TE,ALU,LOAD      LOAD ALU TRAP ENABLE BITS
428  LCDECR           DECREMENT THE LOOP COUNTER
429  ADACT            ADDRESS ARITHMETIC ACTIVATE
430  EA1SAVE          SAVE ADDRESS IN EA1 AND EA0
431  EA2SAVE          SAVE ADDRESS IN EA2 AND EA0
432  EA3SAVE          SAVE ADDRESS IN EA3 AND EA0
433  C,MEMOT          MEMORY REQUEST IF DATA-OPERAND
434  CSAVE            PUSH TEST CONDITION TO STACK(2)
435  C,SEQ            ENABLE CONDITIONAL SEQUENCE
436  COND,MSEXO       EXOR OF S AND O FROM ALU RESULT
437  COND,MSORZ       OR OF S AND Z FROM ALU OPERATION
438  COND,SORZ        OR OF S AND Z FROM STATUS (S1)
439  COND,MCNZ        AND OF C AND NOT Z FROM ALU OPERATION
440  COND,CNZ         AND OF C AND NOT Z FROM STATUS (S1)
441  COND,MZRO        Z FROM ALU OPERATION
442  COND,MCRY        C FROM ALU OPERATION
443  COND,MSGN        S FROM ALU OPERATION
444  COND,MOVFL       O FROM ALU OPERATION
445  COND,ZRO         Z FROM S1
446  COND,CRY         C FROM S1
447  COND,SGN         S FROM S1
448  COND,K           K FROM S1
449  COND,OVFL        O FROM S1
450  COND,PARITY      PARITY OF LEAST SIGNIFICANT BYTE OF F-BUS
```

```
451  COND,QO          Q-REGISTER BIT 0.
452  COND,SAVC1       TOP BIT OF SAVED CONDITION STACK
453  COND,SAVC2       BOTTOM BIT OF SAVED CONDITION STACK
454  COND,LCZ         LOOP COUNTER ZERO RESULT
455  COND,ENTER       CHECK FOR ENT- INSTRUCTIONS
456  COND,DATOP       CHECK FOR DATA AS OPERAND
457  COND,CONOP       CHECK FOR CONSTANT AS OPERAND
458  COND,PDONE       PART DONE FROM STATUS (S1)
459  COND,MFS         S FROM FLOATING AAP
460  COND,MFO         O FROM FLOATING AAP
461  COND,MFU         U FROM FLOATING AAP
462  COND,MDZ         DIVIDE BY 0 FROM FLOATING AAP
463  COND,MIVO        INVALID OPERATION FROM BCD AAP
464  COND,MBO         O FROM BCD AAP
465  COND,RF1OCT      ZERO IN RF-ADDRESS 1 BITS 0-2
466  COND,RF2OCT      ZERO IN RF-ADDRESS 2 BITS 0-2
467  COND,GOOPS       GET-TYPE IS G,OOPS
468  COND,AQSLZ       QO FOR ALU, LCZ FOR SEQ.
469  COND,IRALT       FIRST-OPERAND IS ALT-ADDRESSED
470  COND,CALL        MACROINSTR. IS CALL
471  COND,ENTM        MACROINSTR. IS ENTM
472  COND,ENTT        MACROINSTR. IS ENTT
473  COND,JUMPG       MACROINSTR. IS JUMPG
474  JMP              JUMP TO ADDRESS
475  JMPREL           JUMP TO VECTOR ADDRESS
476  RETURN           RETURN TO SEQUENCER ADDRESS
477  NEXT             NEXT MICRO INSTRUCTION
478  HOLD             HOLD SEQUENCER STACK
479  POP              POP SEQUENCER STACK
480  LOAD             LOAD SEQUENCER STACK
481  PUSH             PUSH SEQUENCER STACK
482  F,JMP            FALSE JUMP TO ADDRESS
483  F,JMPREL         FALSE VECTOR JUMP TO ADDRESS
484  F,RETURN         FALSE RETURN TO TOP OF STACK
485  F,NEXT           FALSE NEXT MICRO INSTRUCTION
486  F,HOLD           FALSE HOLD SEQUENCER STACK
487  F,POP            FALSE POP SEQUENCER STACK
488  F,LOAD           FALSE LOAD SEQUENCER STACK
489  F,PUSH           FALSE PUSH SEQUENCER STACK
490  INVSEQ           INVERT TEST CONDITION FOR SEQUENCE
491  AA,0             ADDRESS A OPERAND IS ZERO
492  AA,MARG          ADDRESS A OPERAND IS MINIARGUMENT
493  AA,DISP          ADDRESS A OPERAND IS DISPLACEMENT
494  AA,DATA          ADDRESS A OPERAND IS DATA REGISTER
495  AA,EA0           ADDRESS A OPERAND IS EA0 REGISTER
496  AA,EA1           ADDRESS A OPERAND IS EA1 REGISTER
497  AA,EA2           ADDRESS A OPERAND IS EA2 REGISTER
498  AA,EA3           ADDRESS A OPERAND IS EA3 REGISTER
499  AB,0             ADDRESS B OPERAND IS ZERO
500  AB,MARG          ADDRESS B OPERAND IS MINIARGUMENT
501  AB,B             ADDRESS B OPERAND IS BASE (B) REGISTER
502  AB,R             ADDRESS B OPERAND IS RECORD (R) REGISTER
503  AB,IX1           ADDRESS B OPERAND IS INDEX REGISTER X1
504  AB,IX2           ADDRESS B OPERAND IS INDEX REGISTER X2
505  AB,IX3           ADDRESS B OPERAND IS INDEX REGISTER X3
506  AB,IX4           ADDRESS B OPERAND IS INDEX REGISTER X4
507  AB,CMBRET        RETURN FROM CMISS U-CODE
```

508	AB,ADR	EA0 IF RECYCLE NOT NECESSARY
509	AB,EA1DIR	EA1 IF RECYCLE NOT NECESSARY
510	AB,ADR+4	PREVIOUS ADDRESS +4 IF RECYCLE NOT NECESSARY
511	AB,X1ORS	DESC(X)(I1),I1 SCALED ACCORDING TO INSTRUCTION
512	AB,X2ORS	DESC(X)(I2),I2 SCALED ACCORDING TO INSTRUCTION
513	AB,X3ORS	DESC(X)(I3),I3 SCALED ACCORDING TO INSTRUCTION
514	AB,X4ORS	DESC(X)(I4),I4 SCALED ACCORDING TO INSTRUCTION
515	LADDR	PERFORM A LADDR REQUEST
516	WR,POF	PERFORM A PHYSICAL WRITE WITH MMS
517	CCDD	CLEAR CACHE AND DUMP DIRTY
518	WR,PHYS	WRITE PHYSICAL SEGMENT
519	WR,DOM	WRITE DATA MEMORY IN NORMAL DOMAIN
520	WR,ADOM	WRITE DATA MEMORY IN ALTERNATIVE DOMAIN
521	WRITE	WRITE DATA MEMORY
522	QVACC	FORCE QVACC (USE WITH A,IAC, AND LOADLA)
523	RD,POF	PERFORM A PHYSICAL READ WITH MMS
524	RD,PX	READ DATA MEMORY, WRITE PERMIT REQUIRED
525	RD,PHYS	READ PHYSICAL SEGMENT
526	RD,DOM	READ DATA MEMORY IN NORMAL DOMAIN
527	RD,ADOM	READ DATA MEMORY IN ALTERNATIVE DOMAIN
528	READ	READ DATA MEMORY
529	CLEAR	CLEAR IAC
530	ISAMP	INTERRUPT SAMPLE
531	G,OOPS	GET NEXT INSTRUCTION AND OPERAND SPECIFIER
532	G,OOPS,T	GET NEXT INSTRUCTION AND OPERAND SPECIFIER IF TRUE
533	G,OOPS,F	GET NEXT INSTRUCTION AND OPERAND SPECIFIER IF FALSE
534	G,COOPS	GET NEXT INSTRUCTION AND OPERAND AFTER CALL
535	G,DIR1	GET IMMEDIATE OPERAND 1 BYTE LONG
536	G,DIR2	GET IMMEDIATE OPERAND 2 BYTES LONG
537	G,OPS	GET SECOND OR LATER OPERAND SPECIFIER
538	G,DIR4	GET IMMEDIATE OPERAND 4 BYTES LONG
539	G,OPSTRD	GET SECOND OPERAND SPECIFIER FOR STRING INSTR
540	G,TOOPS	GET NEXT INSTRUCTION CODE, FOR TESTING ONLY (NO MAPPIN
541	LOADLA	SET START ADDRESS FROM IB TO LA
542	TBC,NEXT	CACHE WRITE NEXT INSTRUCTION STREAM ADDRESS
543	TBC,SUBR	CACHE WRITE SUBROUTINE ADDRESS
544	TBC,L	CACHE WRITE LINK REGISTER
545	TBC,NPCREL	CACHE WRITE NPC RELATIVE JUMP ADDRESS
546	TBC,PREL	CACHE WRITE P RELATIVE JUMP ADDRESS
547	TBC,INCRILAR	ILAR + 4 -> ILAR
548	TBC,NOOP	NO TBC-OPERATION
549	ABR,NEXT	CALCULATE NEXT INSTRUCTION STREAM ADDRESS
550	ABR,NPCREL	CALCULATE JUMP TARGET ADDRESS
551	ABR,NEXTL	CALCULATE NEXT ADDRESS TO LINK REGISTER
552	ORA	USE OR LOGIC-CONTROLLED A-OPERAND
553	ORA,IN	OR A OPERAND IN CURRENT FROM INSTRUCTION
554	ORA,OP	OR A OPERAND IN CURRENT FROM CURRENT OPERAND SPECIFIER
555	ORA,ALTEN	OR A OPERAND (IN NEXT) FROM STRING SOURCE OPERAND
556	ORB	USE OR LOGIC-CONTROLLED B-OPERAND
557	ORD	USE OR LOGIC-CONTROLLED DESTINATION
558	ORD,IN	OR DESTINATION IN CURRENT FROM INSTRUCTION
559	ORD,OP	OR DESTINATION IN CURRENT FROM OPERAND SPECIFIER
560	ORD,OP1	OR DESTINATION (IN NEXT) FROM FIRST OPERAND SPECIFIER
561	ORD,ALTEN	OR DESTINATION (IN NEXT) FROM STRING DEST. OPERAND
562	OR,N	OR-CONTROL IS FOR NEXT CYCLE
563	OR,NE	ENABLE EXTENSION REGISTER IN NEXT MICRO CYCLE
564	IFT	IF TRUE THEN ...

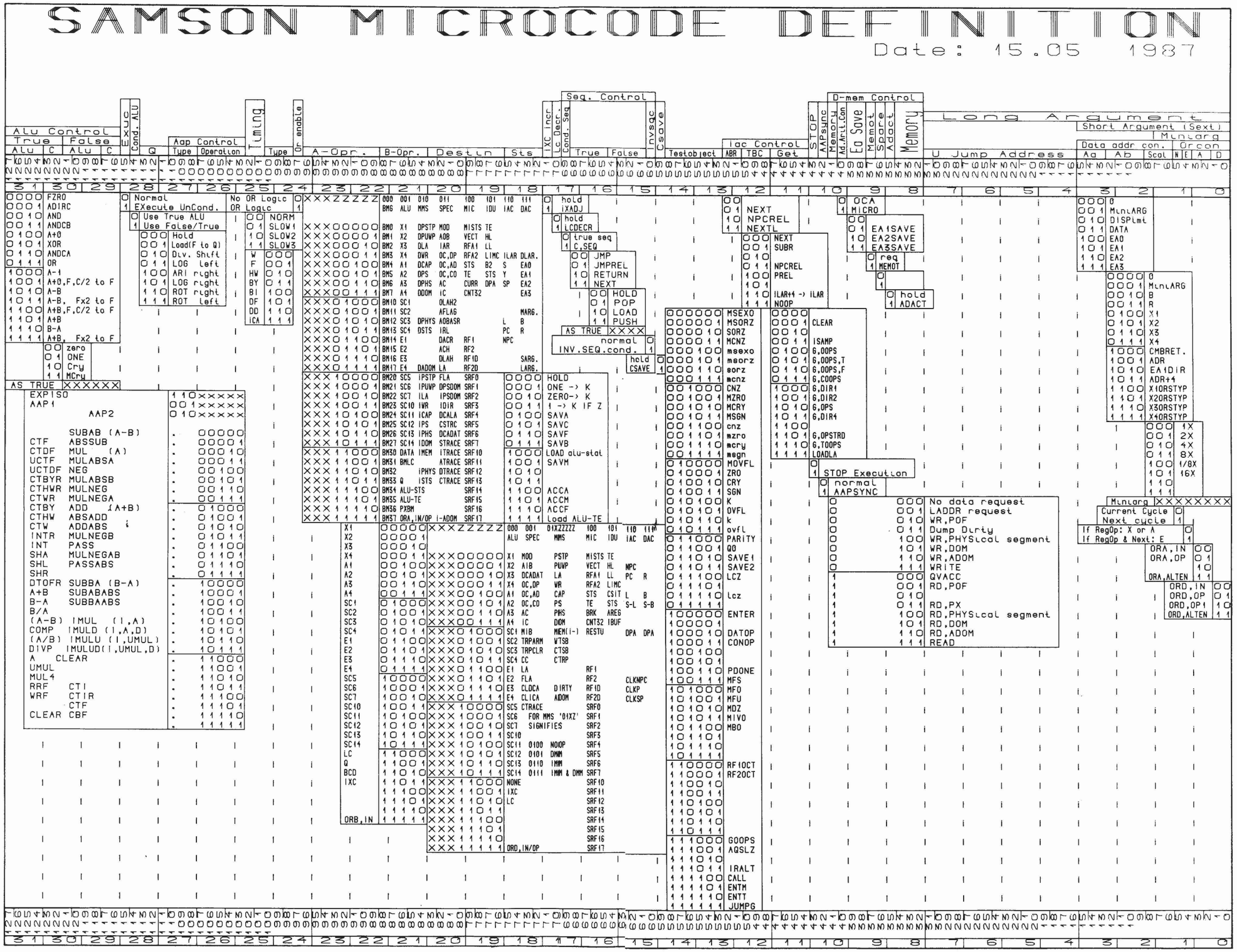
```
565  SLOW1          CYCLE TIME = 110 N.SEC.
566  SLOW2          CYCLE TIME = 160 N.SEC.
567  FSLOW1         FORCE SLOW1 ON A,SPEC,<-->
568  AAPSYNC        WAIT FOR AAP READY
569  AAPSYNCL       WAIT FOR AAP READY (USED FOR LEAST PART)
570  IX*1           SCALING = *1
571  IX*2           SCALING = *2
572  IX*4           SCALING = *4
573  IX*8           SCALING = *8
574  IX/8           SCALING = /8
575  IX*16          SCALING = *16
576  AAP1,CTF       AAP1: CONVERT TO FLOATING
577  AAP1,CTDF      AAP1: CONVERT TO FLOATING
578  AAP1,UCTF      AAP1: UNSIGN CONVERT TO FLOATING
579  AAP1,UCTDF     AAP1: UNSIGN CONVERT TO FLOATING
580  AAP1,CTBYR     AAP1: CONVERT TO INT. ROUNDED
581  AAP1,CTHWR     AAP1: CONVERT TO INT. ROUNDED
582  AAP1,CTWR      AAP1: CONVERT TO INT. ROUNDED
583  AAP1,CTBY      AAP1: CONVERT TO INT. ROUNDED
584  AAP1,CTHW      AAP1: CONVERT TO INT. ROUNDED
585  AAP1,CTW       AAP1: CONVERT TO INT. ROUNDED
586  AAP1,INTR      AAP1: INTEGER-PART ROUNDED
587  AAP1,INT       AAP1: INTEGER-PART TRUNCATED
588  AAP1,SHA       AAP1: SHIFT  ARITHMETICAL
589  AAP1,SHL       AAP1: SHIFT LOGICAL
590  AAP1,SHR       AAP1: SHIFT ROTATIONAL
591  AAP1,DTOFR     AAP1: CONVERT DOUBLE TO FLOATING ROUNDED
592  AAP1,A+B       AAP1: A+B
593  AAP1,B-A       AAP1: B-A
594  AAP1,B/A       AAP1: B/A
595  AAP1,A-B       AAP1: A-B
596  AAP1,COMP      AAP1: COMPARE (A-B)
597  AAP1,A/B       AAP1: A/B
598  AAP1,DIVP      AAP1: PARTIAL DIVIDE A/B
599  AAP1,A*B       AAP1: A*B
600  AAP1,UMUL      AAP1: UNSIGNED MULTIPLY.
601  AAP1,MUL4      AAP1: MULTIPLY WITH OVERFLOW
602  AAP1,RRF       AAP1: READ AAP REGISTERFILE
603  AAP1,WRF       AAP1: WRITE AAP REGISTERFILE
604  AAP1,CLEAR     AAP1: COPY A TO F
605  AAP2,SUBAB     AAP2: SUBTRACT A-B
606  AAP2,A-B       AAP2: SUBTRACT A-B
607  AAP2,ABSSUB    AAP2: MAGNITUDE OF DIFFERENCE
608  AAP2,MUL       AAP2: MULTIPLY
609  AAP2,A*B       AAP2: MULTIPLY
610  AAP2,MULABSA   AAP2: B TIMES MAGNITUDE OF A
611  AAP2,NEG       AAP2: NEGATE
612  AAP2,MULABSB   AAP2: A TIMES MAGNITUDE OF B
613  AAP2,MULNEG    AAP2: MULTIPLY AND NEGATE
614  AAP2,MULNEGA   AAP2: B TIMES NEGATIVE VALUE OF A
615  AAP2,ADD       AAP2: ADD
616  AAP2,A+B       AAP2: ADD
617  AAP2,ABSADD    AAP2: MAGNITUDE OF SUM
618  AAP2,ADDABS    AAP2: SUM OF MAGNITUDES
619  AAP2,MULNEGB   AAP2: A TIMES NEGATIVE VALUE OF B
620  AAP2,PASS      AAP2: IDENTITY
621  AAP2,MULNEGAB  AAP2: NEGATIVE VALUE OF A TIMES B
```

```
622  AAP2,PASSABS    AAP2: ABSOLUTE VALUE
623  AAP2,SUBBA      AAP2: SUBTRACT
624  AAP2,B-A        AAP2: SUBTRACT
625  AAP2,SUBABABS   AAP2: DIFFERENCE OF MAGNITUDES
626  AAP2,SUBBAABS   AAP2: DIFFERENCE OF MAGNITUDES
627  AAP2,IMUL       AAP2: INTEGER MUL, ONE RESULT
628  AAPI,A*B        AAP2: INTEGER MUL, ONE RESULT
629  AAP2,IMULD      AAP2: INTEGER MUL, TWO RESULTS
630  AAPI,A*B,D      AAP2: INTEGER MUL, TWO RESULTS
631  AAP2,IMULU      AAP2: INTEGER UMUL, ONE RESULT
632  AAPI,UMUL       AAP2: INTEGER UMUL, ONE RESULT
633  AAPI,UMUL,D     AAP2: INTEGER UMUL, TWO RESULTS
634  AAP2,CLEAR      AAP2: CLEAR ONGOING AAP2-SEQUENCE
635  AAP2,CTI        AAP2: CONVERT TO INTEGER
636  AAP2,CTIR       AAP2: CONVERT TO INTEGER ROUNDED
637  AAP2,CTF        AAP2: CONVERT TO  FLOATING
638  AAP2,CBF        AAP2: CONVERT TO OTHER FLOATING FORMAT
639  AAP2,EXPISO     AAP2: EXPONENT ISOLATE
640  #A,OP           READ + ORA + ADACT + TYP,OR + ORA,OP
641  #A,OPM          #A,OP + OR,NE + AB,ADR+4
642  #A,OPL          READ + ORA + C,MEMOT + TYP,OR
643  #A,WOP          READ + ORA + ADACT + TYP,OR + ORA,OP + EA1SAVE + ORD,0
644  #A,WOPM         #A,WOP + OR,NE + AB,ADR+4 + ORD,OP
645  #A,WOPL         READ + ORA + C,MEMOT + TYP,OR
646  STOP            STOP-MICROPROGRAM
```

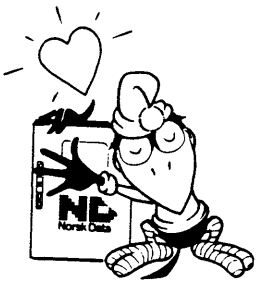
APPENDIX B THE MICROINSTRUCTION FORMAT

SAMSON MICROCODE DEFINITION

Date: 15.05 1987



Index



SEND US YOUR COMMENTS!

Are you frustrated because of unclear information in our manuals? Do you have trouble finding things?

Please let us know if you:
– find errors
– cannot understand information
– cannot find information
– find needless information.

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

Send to:
Norsk Data A.S
Documentation Department
P.O. Box 25 BOGERUD
N - 0621 OSLO 6 - Norway

NOTE!

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

Manual Name: _____ Manual number: _____

Which version of the product are you using? _____

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

_____

_____

_____

_____

_____

Do you have suggestions for improving this manual? _____

_____

_____

_____

_____

_____

_____

Your name: _____ Date: _____

Company: _____ Position: _____

Address: _____

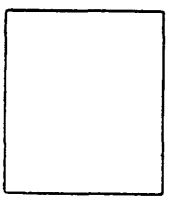
_____

What are you using this manual for? _____

_____

Answer from Norsk Data _____

Answered by _____ Date _____



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