79/15

EISCAT
TECHNICAL
NOTES

A DESCRIPTION OF THE ASSEMBLY LANGUAGE FOR THE EISCAT DIGITAL CORRELATOR

by

Bård Willy Tørustad
EISCAT Norway
N-9027 Ramfjordbotn
Norway

KIRUNA
Sweden

EISCAT Technical Note 79/15

A DESCRIPTION OF THE ASSEMBLY LANGUAGE FOR THE EISCAT DIGITAL CORRELATOR

by

Bård Willy Törustad
EISCAT Norway
N-9027 Ramfjordbotn
Norway

EISCAT Scientific Association
S-981 01 Kiruna, Sweden
August 1979

A DESCRIPTION OF THE ASSEMBLY LANGUAGE FOR THE EISCAT DIGITAL CORRELATOR

BY

BÅRD WILLY TØRUSTAD
EISCAT NORWAY
N-9027 RAMFJORDBOTN

NORWAY

ABSTRACT

AS THE WRITING OF CORRELATOR PROGRAMS WENT ALONG, ONE EARLY SAW THE NEED OF AN EASIER WAY OF PROGRAMMING THAN THE PRESENT ONE. IN THAT SCHEME ONE DEALS ALL THE TIME WITH OCTAL NUMBERS IN A RATHER TEDIOUS WAY.
TO MAKE IT EASIER TO PROGRAM, A LANGUAGE WAS DEFINED BY ANDY PEPPERDINE EARLY 1979. HIS CONCEPT HAS ONLY BEEN SLIGHTLY MODIFIED LATER.
THIS MANUAL DESCRIBES THIS LANGUAGE. TO BE ABLE TO USE THE MANUAL AND THE LANGUAGE, ONE SHOULD HAVE KNOWLEDGE ABOUT THE BASIC PRINCIPLES OF THE CORRELATOR. (REF 1)

IN THE FUTURE ONE HOPES TO RAISE THE LEVEL OF THE LANGUAGE BY IMPLEMENTING SUBROUTINE STRUCTURES, LOOPS ETC.

ANY COMMENTS, CORRECTIONS AND SUGGESTIONS SHOULD BE ROUTED TO EISCAT-RAMFJORDMOEN (ATT.: BÅRD TØRUSTAD)

TABLE OF CONTENTS

1. INTRODUCTION
2. GENERAL
3. PROGRAMMING THE MICRO-INSTRUCTION

   3.0 GENERAL INSTRUCTION FORMAT
   3.1 KEY-WORD = PRO (PROGRAM INSTRUCTIONS)
   3.2 KEY-WORD = APB (ADDRESS PROCESSOR BUFFER MEM.)
   APM (ADDRESS PROCESSOR RESULT MEM.)
   3.3 KEY-WORD = ARI (ARITHMETICAL INSTRUCTIONS)
   3.4 KEY-WORD = ACC (ACCUMULATION INSTRUCTIONS)
   3.5 KEY-WORD = OUT (INSTRUCTIONS FOR DATA TRANSFER TO THE COMPUTER)
   3.6 KEY-WORD = I/O (INSTRUCTIONS FOR DATA- AND ADDR.-EXCHANGING IN A MULTI CORRELATOR SYSTEM)

4. AUXILIARY KEYWORDS

   4.0 NXT (NEXT)
   4.1 IDL (IDLE)
   4.2 END
   4.3 LOC (LOCATION)

5. LABEL FACILITIES

   5.0 LAB (DEFINING LABELS)
   5.1 GTO (USING LABELS FOR JUMPS)
   5.2 EXAMPLES OF USE

6. DEFINING THE DATAFIELD
7. CONDITIONAL ASSEMBLING
8. A COMPLETE PROGRAM

   8.0 GENERAL PROGRAM DEFINITION
   8.1 THE DUMMY STATEMENT DEFINITION

9. USING CORPREP

   9.0 INVOKING CORPREP AND TALKING TO IT
   9.1 AN EXAMPLE

APPENDIX A: REFERENCES
APPENDIX B: A TABLE OF ALL THE AVAILABLE TESTS FOR MICRO-PROGRAM BRANCHING

1. INTRODUCTION

THIS REPORT GIVES A DESCRIPTION OF THE CORRELATOR-LANGUAGE CORPREP, AND HOW TO USE IT.
CORPREP IS A PREPASS ASSEMBLER, THAT IN THE PRESENT STATE SIMPLY, ON THE BASES OF A SET OF DEFINED KEYWORDS, SUB-CODES AND OP. CODES, GENERATES A CORRELATOR PROGRAM IMAGE. THIS MEAN THAT THERE IS NO CHECKING AS TO WHETHER THE PROGRAM IS DOING CRAZY THINGS OR NOT. HOWEVER, ERROR MESSAGES WILL BE ISSUED WHEN ILLEGAL CODING IN THE SOURCE TEXT IS DETECTED.
(THE SENSIBILITY OF A PROGRAM CAN TO SOME EXTENT BE TESTED USING THE CORRSIM PROGRAM. (SEE REF 2))
ALSO THE ENTIRE DATAFIELD OF THE CORRELATOR CAN BE DEFINED.

BEFORE ANY DECODING IS DONE, THE PROGRAM IMAGE IS SET TO A DEFINED DUMMY STATE. (SEE 8.1) THUS ONLY ACTIVE INSTRUCTINS NEED BE DEFINED. (AN ACTIVE INSTRUCTION IS AN INSTRUCTION THAT IS DOING SENSIBLE THINGS FOR THE PROGRAM)
NO DUMMIES ARE NECESSARY TO DEFINE.

THE OUTPUT (OBJECT-CODE) FROM CORPREP IS COMPATIBLE WITH THE EXISTING SOFTWARE, HENCE ALL THE DIFFERENT PARTS OF THE PROGRAM CORRSIM (REF 2), CAN BE APPLIED TO THE OUTPUT.

2. GENERAL

NOTATIONS USED THRUOUT THE TEXT:

<> ANYTHING ENCLOSED WITHIN THESE BRACKETS SHOULD BE TREATED AS A SINGLE ENTITY, WHICH MAY TAKE VALUES OR STRUCTURES DEFINED ELSEWHERE.

::= MEANS 'IS DEFINED AS'

// ANYTHING ENCLOSED WITHIN 2 SLASHES IS A DEFINITION.

() IN SYNTAX DEFINITIONS; ANYTING ENCLOSED IN PARANTHESIS MAY BE REPEATED ANY NUMBER OF TIMES ON THAT LINE.

EXAMPLE:

A DIGIT COULD BE DEFINED LIKE THIS:

<DIGIT>::=/1/2/3/4/5/6/7/8/9/0/

I.E. A DIGIT IS A '1' OR A '2' OR A '3' ETC.

ANYTHING NOT APPEARING IN SPECIAL BRACKETS, <>,
IS TO BE TAKEN TO MEAN EXACTLY WHAT IT IS.

IN THE CORPREP LANGUAGE EACH LINE IS CONSIDERED A COMMAND. EVERY LINE (I.E. EVERY COMMAND) HAS A 3-CHARACTER IDENTIFIER DENOTED KEYWORD.
THE 3 LEFTMOST CHARACTERS OF A LINE MUST ALWAYS BE A KEYWORD. (AN EXCEPTION IS MADE FOR COMMENTS AND CONDITIONAL ASSEMBLING (SEE 7.))

THE FOLLOWING KEYWORDS ARE DEFINED

<KEY-WORD>::=/NXT/IDL/END/LOC/SUB/LAB/GTO/PRO/APB/APM/
/ACC/ARI/OUT/I/O/

EACH KEYWORD IS EXPLAINED IN SEPERATE SECTIONS LATER.

LEGAL SEPERATORS ARE

<SEP>::=/<SPACE>/,/=/:/-/<SEP>/

COMMENTS CAN BE PUT IN ANYWHERE AFTER A PER-CENT SIGN (%).

3. PROGRAMMING THE MICRO-INSTRUCTION.

3.0 GENERAL INSTRUCTION FORMAT

A GENERAL INSTRUCTION WOULD LOOK LIKE THIS:

<STATEMENT>::=
<KEY-WORD>(<SEP><SUBCODE><SEP><OP. CODE>)....

PERMITTED KEYWORDS CONCERNING MICRO-INSTRUCTION ARE:

<KEY-WORD>::=/PRO/APB/APM/ARI/ACC/OUT/I/O/

(NOTE THAT I/O IS ONE KEYWORD)
THESE KEYWORDS REFER TO SUBFIELDS IN THE MICRO-INSTRUCTION WORD.
TO DO A COMPLETE PROGRAMMING OF A MICRO-INSTRUCTION WORD 7 LINES OF CODE ARE NEEDED.
NOTE HOWEVER, THAT ONLY ACTIVE PROGRAM FIELDS NEED BE DEFINED IN THIS CONCEPT.
EACH KEYWORD HAS A SET OF SUBCODES THAT ARE PARTICULAR TO IT. THESE WILL BE DEFINED IN THE SECTIONS 3.1 THRU 3.6, WITH ONE SECTION FOR EACH KEYWORD.
IN ITS TURN EACH SUBCODE HAS A SET OF OP. CODES THAT ARE PARTICULAR TO IT. THESE ARE DEFINED AT THE SAME TIME AS THE SUBCODES.

3.1 KEY-WORD = PRO (PROGRAM INSTRUCTION)

SYNTAX:

PRO(<SEP><SUBCODE><SEP><OPCODE>)....

SUBCODE	MEANING
/CC/	CONDITION-CODE

FOR THIS PARTICULAR SUBCODE THERE ARE 5 DIFFERENT WAYS TO DEFINE THE OP. CODE.:

THE DIFFERENT WAYS ARE LISTED BELOW:
(NOTE THAT ALL THE CASES BELOW MUST BE PRECEDED BY PRO<SEP> THUS GETTING PRO-CC=....)

CASE 1: CC=<OCTAL CODE FOR THE TEST>
THE OCTAL CODES ARE FOUND IN REF 1.

CASE 2: CC=(USE-A)
THIS MEANS USE A UNCONDITIONALLY.

CASE 3: CC=(IF <TEST> THEN B ELSE A)
IF TEST IS TRUE USE B ELSE USE A
THE POSSIBLE TESTS ARE:

<TEST>::=/LC1=0/
/LC2=0/
/LC1=0 OR LC2=0/
/LC3=0/
/LC1=0 OR LC3=0/
/LC2=0 OR LC3=0/
/LC1=0 OR LC2=0 OR LC3=0/

/LC2#0/
/LC1=0 OR LC2#0/
/LC2#0 OR LC3=0/
/LC1=0 OR LC2#0 OR LC3=0/

/LC3#0/
/LC1=0 OR LC3#0/
/LC2=0 OR LC3#0/
/LC1=0 OR LC2=0 OR LC3#0/
/LC2=0 OR LC3#0/

/LC1=0 OR LC2#0 OR LC3#0/

```
CASE 4: CC=(IF <TEST1> THEN B ELSEIF <TEST2> THEN A
            OTHERWISE CONT)

        (IF TEST1 IS TRUE USE B ELSEIF TEST2 IS TRUE
        USE A OTHERWISE TAKE NEXT INSTRUCTION.)

        <TEST1> AND <TEST2> MUST BE ONE OF THE PAIRS
        LISTED BELOW:

        <TEST1>::=                          <TEST2>::=

        /LC1=0/                              /LC2=0/
        /LC3=0/                              /LC2=0/
        /LC1=0 OR LC3=0/                     /LC2=0/

        /LC1=0/                              /LC2#0/
        /LC3=0/                              /LC2#0/
        /LC1=0 OR LC3=0/                     /LC2#0/

        /LC1#0/                              /LC2=0/
        /LC3#0/                              /LC2=0/
        /LC1#0 OR LC3#0/                     /LC2=0/

        /LC1#0/                              /LC2#0/
        /LC3#0/                              /LC2#0/
        /LC1#0 OR LC3#0/                     /LC2#0/

CASE 5: CC=(IF <TEST> THEN B OTHERWISE CONT)

        (IF TEST IS TRUE USE B OTHERWISE CONTINUE
         TO NEXT INSTRUCTION)

        <TEST>::=/LC1=0 OR LC3=0/
                 /LC1#0 OR LC3#0/

EXAMPLES OF USE :

CASE 1: PRO-CC=71;....

        LOOKING UNDER CHAPTER FOR PROGRAM INSTRUCTIONS IN REF 1,
        ONE OBSERVES THAT CONDITION-CODE 71 MEANS
        IF LC1=0 THEN B ELSE A

CASE 2: PRO-CC=(USE-A);....

CASE 3: PRO-CC=(IF LC2=0 OR LC3=0 THEN B ELSE A);....

CASE 4: PRO-CC=(IF LC3#0 THEN B ELSEIF LC2=0 THEN A
                OTHERWISE CONT);....

CASE 5: PRO-CC=(IF LC1=0 OR LC3=0 THEN B OTHERWISE CONT);....
```

SUBCODE	MEANING	OP. CODE	MEANING
/A/		/COND/	PC=PC+1;POP STACK
		/RETD/	PC=RETURN ADDRESS;POP STACK
		/GTOD/	PC=ADDRESS;POP STACK
		/SARD/	PC=SAR;POP STACK
		/CON/	PC=PC+1
		/RET/	PC=RETURN ADDRESS
		/GTO/	PC=ADDRESS
		/SAR/	PC=SAR
		/CONS/	PC=PC+1;PUSH STACK
		/RETS/	PC=RETURN ADDRESS;PUSH STACK
		/GTOS/	PC=ADDRESS;PUSH STACK
		/SARS/	PC=SAR;PUSH STACK
/B/		SAME OP. CODES AS FOR A.	
/LC1/ -	LOOP COUNTER 1		
		/NOOP/	NO OPERATION
		/DEC/	LC1=LC1-1
		/LCR1/	LC1=LCR1
		/LC1A/	LC1=LCR1A
		/CID2/	IF LC1=0 THEN (LC1=LCR1;LC2=LC2-1) ELSE (LC1=LC1-1)
		/CT3A/	IF (LC1=0 AND LC3=0) THEN (LC1=LCR1A) ELSE (LC1=LC1-1)
		/C1/	IF LC1=0 THEN (LC1=LCR1) ELSE (LC1=LC1-1)
		/CA/	IF LC1=0 THEN (LC1=LCR1A) ELSE (LC1=LC1-1)
/LC2/ -	LOOP COUNTER 2		
		/NOOP/	NO OPERATION
		/DEC/	LC2=LC2-1
		/LCR2/	LC2=LCR2
/LC3/ -	LOOP COUNTER 3		
		/NOOP/	NO OPERATION
		/DEC/	LC3=LC3-1
		/LCR3/	LC3=LCR3
		/CR3/	IF LC3=0 THEN (LC3=LCR3) ELSE (LC3=LC3-1)

/LC1A/ - TEMPORARY REGISTER FOR LC1
(ALSO CALLED LCR1A)

/NOOP/	NO OPERATION
/LC1/	LCR1A=LC1

/RELD/ - RELOAD

/NO/	NO OPERATION
/YES/	RELOAD REGISTER WITH VALUE FROM APB (ADDRESS PROCESSOR FOR BUFFER MEMORY)

/RADR/ - REGISTER ADDRESS WHEN RELOADING REGISTER
(I.E. WHEN RELD = YES)

/SAR/	START ADDRESS REGISTER
/BAR/	BASE ADDRESS REGISTER (APB)
/LCR1/	LOOP COUNTER 1
/LCR2/	LOOP COUNTER 2
/LCR3/	LOOP COUNTER 3

(LC1, LC2 AND LC3 MEAN LOOP-COUNTER 1, 2 AND 3 RESPECTIVELY)

3.2 KEY-WORD = APB (ADDRESS PROCESSOR BUFFER MEM.)
KEY-WORD = APM (ADDRESS PROCESSOR RESULT MEM.)

SYNTAX:

APB(<SEP><SUBCODE><SEP><OPCODE>)....
APM(<SEP><SUBCODE><SEP><OPCODE>)....

SUBCODE	MEANING	OP. CODE	MEANING
/SRC/	ALU-SOURCE	/AQ/	R=RS(A) - S=Q
		/AB/	R=RS(A) - S=RS(B)
		/ZQ/	R=ZERO - S=Q
		/ZB/	R=ZERO - S=RS(B)
		/ZA/	R=ZERO - S=RS(A)
		/IA/	R=DATAI - S=RS(A)
		/IQ/	R=DATAI - S=Q
		/IZ/	R=DATAI - S=ZERO
/FUNC/	ALU-FUNCTION	/R+S/	F=R+S
		/S-R/	F=S-R
		/R-S/	F=R-S
		/RORS/	F=R.OR.S
		/RNDS/	F=R.AND.S
		/NRS/	F=.NOT.(R).AND.S
		/RXS/	F=R.EXOR.S
		/RXNS/	F=R.EXNOR.S
/DEST/	ALU-DEST.	/QF/	ADDR.=F - Q=F
		/F/	ADDR.=F
		/BFOA/	ADDR.=RS(A)- RS(B)=F
		/BF/	ADDR.=F - RS(B)=F
		/B/Q//	ADDR.=F - RS(B)=F/2;Q=Q/2
		/B//	ADDR.=F - RS(B)=F/2
		/B2Q2/	ADDR.=F - RS(B)=2F;Q=2Q
		/B2/	ADDR.=F - RS(B)=2F
/A/	A-ADDR.	<OCT. NR.> (0-17)	REGISTER STACK ADDRESS (ONLY FOR READING)
/B/	B-ADDR	<OCT. NR.> (0-17)	REGISTER STACK ADDRESS (BOTH READING AND WRITING)
/SEL/	SELECT	/NO/	B-ADDRESS IS USED TO ADDRESS THE STACK WHENEVER B-ADDRESS IS REFERENCED. (I.E. RS(B))
		/YES/	SELECTS LC1 TO ADDRESS THE STACK WHENVER B-ADDRESS IS REFERENCED. I.E. REGISTER STACK ADDRESS := LC1. (I.E. RS(LC1) INSTEAD OF RS(B))

THE SELECT OPTION APPLIES ONLY TO THE APB PROCESSOR.

3.3 KEY-WORD = ARI (ARITHMETICAL INSTRUCTIONS)

SYNTAX:

ARI(<SEP><SUBCODE><SEP><OP. CODE>)....

SUBCODE	MEANING
/M1A/	SOURCE FOR MULTIPLIER 1 REGISTER A
/M2A/	SOURCE FOR MULTIPLIER 2 REGISTER A
/M3A/	SOURCE FOR MULTIPLIER 3 REGISTER A
/M4A/	SOURCE FOR MULTIPLIER 4 REGISTER A
/M1B/	SOURCE FOR MULTIPLIER 1 REGISTER B
/M2B/	SOURCE FOR MULTIPLIER 2 REGISTER B
/M3B/	SOURCE FOR MULTIPLIER 3 REGISTER B
/M4B/	SOURCE FOR MULTIPLIER 4 REGISTER B

SUBCODE	OP. CODE	MEANING
/M1A/	/XINT/	REG-A=X-INTERNAL
	/YINT/	REG-A=Y-INTERNAL
	/XEXT/	REG-A=X-EXTERNAL
	/YEXT/	REG-A=Y-EXTERNAL
	/ONE/	REG-A=1 (ONE)

SAME OP. CODES AND MEANINGS APPLY TO REGISTER A FOR THE OTHER MULTIPLIERS AS WELL.

SUBCODE	OP. CODE	MEANING
/M1B/	/XINT/	REG-B=X-INTERNAL
	/YINT/	REG-B=Y-INTERNAL
	/XEXT/	REG-B=X-EXTERNAL
	/YEXT/	REG-B=Y-EXTERNAL

SAME OP. CODES AND MEANINGS APPLY TO REGISTER B OF THE OTHER MULTIPLIERS AS WELL.

SUBCODE	MEANING
/S1/S2/S3/S4/	STROBE NEW CONTENT INTO MULTIPLIERS 1,2,3 AND 4.

SUBCODE	MEANING	OP. CODE	MEANING
/S1/	STROBE	/NOOP/	NO OPERATION
		/A/	NEW OPERAND INTO A-REGISTER
		/B/	NEW OPERAND INTO B-REGISTER
		/AB/	NEW OPERANDS INTO BOTH A- AND B-REGISTERS.

SAME APPLY TO /S2/, /S3/ AND /S4/ AS WELL.

SUBCODE	MEANING
/M12/	APPLIES TO ALU12 (I.E. THE ALU CONNECTED TO MULTIPLIER 1 AND 2).
/M34/	APPLIES TO ALU34 (I.E. THE ALU CONNECTED TO MULTIPLIER 3 AND 4).

(THE OP. CODE IS HOW THE OUTPUT FROM THE ALUS IS FORMED)

SUBCODE	MEANING	OP. CODE	MEANING
/M12/	ALU 12	/M1/	OUTPUT = MULT1
		/M2/	OUTPUT = MULT2
		/DIFF/	OUTPUT = MULT1 - MULT2
		/SUM/	OUTPUT = MULT1 + MULT2
		/MIN1/	OUTPUT = -1
/M34/	ALU 34	/M3/	OUTPUT = MULT3
		/M4/	OUTPUT = MULT4
		/DIFF/	OUTPUT = MULT3 - MULT4
		/SUM/	OUTPUT = MULT3 + MULT4
		/MIN1/	OUTPUT = -1

3.4 KEY-WORD = ACC (ACCUMULATION INSTRUCTIONS)

SYNTAX:

ACC(<SEP><SUBCODE><SEP><OP. CODE>)....

SUBCODE	MEANING
/SIO/	STROBE IN- AND OUT-REGISTERS
/WRIT/	CONCERNS THE WRITING TO RESULT MEMORY
/READ/	CONCERNS READING FROM RESULT MEMORY
/SET1/	CONTROLS FLIP-FLOP 1
/CLR1/	--- " ---
/SET2/	CONTROLS FLIP-FLOP 2
/CLR2/	--- " ---

SUBCODE	OP. CODE	MEANING OF OP. CODE
/SIO/	/NO/	NO OPERATION
	/YES/	STROBE IN- AND OUT-REGISTERS
/WRIT/	/NO/	NO OPERATION
	/YES/	WRITE OUT-REGISTER TO RESULT MEMORY
/READ/	/NO/	IN-REGISTER := ACCUMULATOR (I.E. INTERNAL ACCUMULATION)
	/YES/	READ RESULT MEMORY TO IN-REGISTER OR SET IN-REGISTER TO 0 (ZERO). (THE CHOICE IS DEPENDANT ON THE FLIP-FLOPS. CONFER FF1 AND FF2)
/SET1/	/NO/	NO OPERATION
	/YES/	SET FLIP-FLOP 1 TO 1 (FF1 := 1)
/CLR1/	/NO/	NO OPERATION
	/YES/	CLEAR FLIP-FLOP 1 (FF1 := 0)
/SET2/	/NO/	NO OPERATION
	/YES/	SET FLIP-FLOP 2 TO 1 (FF2 := 1)
/CLR2/	/NO/	NO OPERATION
	/YES/	CLEAR FLIP-FLOP 2 (FF2 := 0)

THE FLIP-FLOPS WILL MAINTAIN THEIR PRESENT STATE UNTIL A SET OR CLR IS EXECUTED.

FF2 = 0 ==> FF1 IS CONTROLLING THE READ STATEMENT.
FF2 = 1 ==> FF1 CONTROL OVER READ STATEMENT IS INHIBITED.

ASSUME FF2 = 0 :

FF1 = 0 ==> SET IN-REGISTER TO 0 (ZERO)(IN-REG. := 0)
FF1 = 1 ==> READ RESULT MEMORY TO IN-REGISTER.
(IN-REGISTER := RES.MEM.(APM ADDRESS))

FF2 = 1 ==> ALWAYS READ RESULT MEMORY TO IN-REGISTER.
(IN-REGISTER := RES.MEM.(APM(ADDRESS))
INDEPENDANTLY OF FF1

3.5 KEY-WORD = OUT (INSTRUCTIONS FOR DATA TRANSFER TO THE COMPUTER)

SYNTAX:

OUT(<SEP><SUBCODE><SEP><OP. CODE>)....

DEALS WITH THE DATA-TRANSFER FROM THE RESULT-MEMORY OF THE CORRELATOR TO THE COMPUTER VIA CAMAC AND DMA.

SUBCODE	MEANING	OP. CODE	MEANING
/XFER/	TRANSFER	/NO/	NO OPERATION
		/YES/	ENABLE CORRELATOR FOR DATA X-FER
/INHC/	INHIBIT	/NO/	KEEP INTERNAL CLOCK
		/YES/	INHIBIT INTERNAL CLOCK (DATA-RECEIVED FROM CAMAC ADVANCES CLOCK)
/RDY/	DATA-READY	/NO/	NO OPERATION
		/YES/	'DATA READY' IS SENT TO CAMAC
/XCOD/	TRANSFER CODE	/STAT/	STATUS-WORD
		/CTRL/	CONTROL-WORD
		/CH1L/	CHANNEL 1 LEAST SIGNIFICANT PART
		/CH1M/	CHANNEL 1 MOST SIGNIFICANT PART
		/CH2L/	CHANNEL 2 LEAST SIGNIFICANT PART
		/CH2M/	CHANNEL 2 MOST SIGNIFICANT PART
		/TST1/	TEST WORD 1
		/TST2/	TEST WORD 2
/SRC/	DATA SOURCE	/SLV1/	SLAVE 1
		/SLV2/	SLAVE 2
		/SLV3/	SLAVE 3
		/MSTR/	MASTER MODULE

3.6 KEY-WORD = I/O (INSTRUCTIONS FOR DATA- AND ADDR.-EXCHANGING IN A MULTI CORRELATOR SYSTEM)

SYNTAX:

I/O(<SEP><SUBCODE><SEP><OP. CODE>)....

DEALS WITH I/O COMMUNICATION IN MULTI-CORRELATOR SYSTEMS.

SUBCODE	MEANING	OP. CODE	MEANING
/SETF/		/NO/	NO OPERATION
		/YES/	SET FLAG (SIGNAL AT EXTERNAL CONNECTOR)
/CLRF/		/NO/	NO OPERATION
		/YES/	CLEAR FLAG (REMOVE SIGNAL FROM EXTERNAL CONNECTOR)
/SBUF/	SELECT BUFFER ADDRESS	/NO/	BUFFER ADDRESS FROM INTERNAL HARDWARE
		/YES/	BUFFER ADDRESS FROM EXTERNAL ADDRESS BUS (EAB)
/STRI/		/NO/	NO OPERATION
		/YES/	STROBE I-REGISTER
/EDB/		/NO/	DISABLE EXTERNAL DATA BUS
		/YES/	ENABLE EXTERNAL DATA BUS
/EAB/		/NO/	NO OPERATION
		/YES/	INTERNAL ADDRESS ACTIVE ON EXTERNAL ADDRESS BUS.

4. AUXILIARY KEYWORDS

4.0 NXT (NEXT)

THIS KEYWORD IS USED TO STEP UP THE LOCATION COUNTER WITH 1. THUS IF ONE IS CURRENTLY PROGRAMMING LOCATION N, APPLYING NXT CAUSES PROGRAMMING OF LOCATION N+1 TO START. THE FOLLOWING STATEMENTS UP TO NEXT NXT, APPLIES TO N+1.

E.G.:

```
....
....          STATEMENTS IN LOCATION N
....
NXT
....          STATEMENTS IN LOCATION N+1
....
NXT
....          STATEMENTS IN LOCATION N+2
....
```

4.1 IDL (IDLE)

THIS IS A DO NOTHING KEYWORD, PRIMARLY INTENDED FOR USE IN LOCATION 0, I.E. THE IDLE LOOP. ITS EFFECT IS THAT IT MAINTAINS THE DUMMY INSTRUCTIONS IN THE PARTICULAR LOCATION WHERE IT IS APPLIED.

E.G.:

```
LOC=0
IDL              %IDLE LOOP IN LOCATION 0
NXT
....
```

4.2 END

A PROGRAM MUST ALWAYS BE TERMINATED BY END. I.E. THE VERY LAST KEYWORD ENCOUNTERED IN A PROGRAM MUST BE END.

4.3 LOC (LOCATION)

GENERAL SYNTAX:

LOC=<OCTAL NUMBER>

<OCTAL NUMBER>=0,1,2,3.....77 (OCTAL)

BY MEANS OF LOC THE USER CAN FORCE PROGRAMMING TO START AT ANY LOCATION. FOR EXAMPLE IF ONE WANTS TO PROGRAM LOCATION 36 THE FOLLOWING WILL DO:

```
LOC=36          %START PROGRAMMING AT LOCATION 36 (OCTAL)
```

LOC MUST ALWAYS APPEAR FIRST IN A GROUP OF STATEMENTS FOR A PARTICULAR LOCATION.
BY CHANGING THE LOC STATEMENT, A SUBROUTINE CAN EASILY BE RELOCATED TO ANYWHERE IN THE PROGRAM MEMORY.

5. LABEL FACILITIES - KEYWORDS LAB/GTO

5.0 LAB (DEFINING LABELS)

GENERAL SYNTAX:

```
LAB=<LABEL NAME>
SUB=<LABEL NAME>
```

/SUB/ AND /LAB/ ARE EQUIVALENT.

<LABEL NAME> CAN BE ANY COMBINATION OF NUMBERS (1-9) AND LETTERS (A-Z), MAXIMUM 4 CHARACTERS LONG.
BY MEANS OF THESE STATEMENTS ANY LOCATION CAN BE GIVEN A 'NAME':

```
LOC=40
LAB=LAB1        (OR EQUIVALENTLY: SUB=LAB1)
....
```

THIS SEQUENCE WILL MAKE IT POSSIBLE TO REFER TO LOCATION 40 THRU LABEL LAB1 LATER ON.

5.1 GTO (GOTO)

GENERAL SYNTAX:

```
GTO=<LABEL NAME>
```

(FOR <LABEL NAME> SEE 5.0.)

IF /GTO/,/GTOD/ OR /GTOS/ IS SPECIFIED IN THE PROGRAM FIELD (SEE 3.1) THE 'GOTO ADDRESS' MAY BE DEFINED BY MEANS OF GTO. SO IF YOU HAVE:

```
....
PRO-CC=(USE-A);A=GTO;
GTO LAB1
....
```

THIS STATEMENT, IF ENCOUNTERED, WILL CAUSE A JUMP TO A LOCATION GIVEN THE NAME 'LAB1' BY MEANS OF LAB (SEE 5.0)

5.2 USING LABELS

AN EXAMPLE OF HOW TO USE LABELS:

```
....
LOC=20
PRO-CC=(IF LC1=0 THEN B ELSE A);A=GTOD;B=CON;
GTO SUBR
NXT
....
NXT
....
....
....
LOC=60
LAB=SUBR
....
....
```

WHEN EXECUTING LOCATION 20, AND LC1=0 NOT EQUAL TO ZERO, A WILL BE CHOSEN. CONSEQUENTLY A JUMP TO LABEL 'SUBR' IN LOCATION 60 WILL RESULT.

6. DEFINING THE DATAFIELD

GENERAL SYNTAX:

REG(<SEP><REGISTER NAME><SEP><VALUE>)....

<REGISTER NAME>::=/B<N>/M<N>/SAR/STAT/BAR/I/LCR1/LCR2/LCR3/

/B<N>/ - REGISTER STACK, APB PROCESSOR.
/M<N>/ - REGISTER STACK, APM PROCESSOR.
/SAR/ - START ADDRESS REGISTER.
/STAT/ - STATUS REGISTER.
/BAR/ - BASE ADDRESS REGISTER OF APB.
/I/ - THE DATAI REGISTER OF APB.
/LCR1/ - LOOP-COUNTER REGISTER 1
/LCR2/ - LOOP-COUNTER REGISTER 2
/LCR3/ - LOOP-COUNTER REGISTER 3

FOR /B<N>/ AND /M<N>/ N=0,1,...,17 (OCTAL)
I.E. ONE GETS B0,B1,B2,....B17
AND M0,M1,M2,....M17

FOR B<N> 0 <= VALUE <= 177777 (OCTAL)
FOR M<N> 0 <= VALUE <= 7777 (OCTAL)

EXAMPLES:

```
REG-STAT=3700;  %STATUS REGISTER GETS THE VALUE 3700 (OCT)

REG-B17=0;B0=10;M0=5;M17=1777;
```

THE APB REGISTER-STACK REGISTER 17 GETS 0, REGISTER 0 GETS 10 (OCTAL).
THE APM REGISTER-STACK REGISTER 0 GETS 5 WHEREAS REGISTER 17 GETS 1777 (OCTAL)

7. CONDITIONAL ASSEMBLING

CORPREP ALLOWS FOR SO CALLED CONDITIONAL ASSEMBLING, I.E. CERTAIN SECTIONS OF A PROGRAM CAN BE MARKED, IN SUCH A WAY THAT THE USER CAN DECIDE WHETHER HE/SHE WANTS TO ASSEMBLE THESE SECTIONS WHEN USING CORPREP.
THE USER CAN VIRTUALLY TELL CORPREP WHICH SECTION IS TO BE IGNORED AND WHICH IS TO BE ASSEMLED.
EXAMPLE:

```
....
NXT
&C
....
....                          %SECTION 1
....
&
&T
....
....                          %SECTION 2
....
&
....
```

(THE '&' TELLS CORPREP THAT THE SECTION UP TO THE NEXT '&' IS SUBJECT TO CONDITIONAL ASSEMBLING)
(AS TO HOW TO SPECIFY THE CONDITIONS SEE 8.)
THE CHARACTER SUCCEDING THE '&' IS CONSIDERED THE CONDITION.
IN THE FIRST SECTION 'C' IS THE CONDITION, AND IF 'C' IS SPECIFIED TO CORPREP, THE CODE UP TO THE NEXT '&' WILL BE ASSEMBLED.
IF BOTH 'C' AND 'T' ARE WRITTEN IN, BOTH SECTIONS WILL BE ASSEMBLED, ETC.
IF NO CONDITION IS SPECIFIED, THE SEQUENCES &C-& AND &T-& ARE IGNORED.

A MAXIMUM OF 5 DIFFERENT CONDITIONS CAN BE DEFINED.
THE '&' MUST ALWAYS APPEAR AS THE FIRST CHARACTER ON A LINE.

8. A COMPLETE PROGRAM

8.0 GENERAL PROGRAM DEFINITION

PROGRAMMING SHOULD START BY DEFINING THE IDLE LOOP IN LOCATION 0. THE USER CAN SET UP AN IDLE LOOP OF HIS OWN, BUT A WAY THAT NEED NO 'THINKING' IS:

```
LOC=0
LAB=ZERO          %DEFINE A LABEL FOR CONVENIENCE
IDL               %IDLE LOOP IN LOC. 0
NXT               %START PROGRAMMING OF NEXT LOCATION
....              %START ACTUAL PROGRAM
```

IN EACH LOCATION SEVERAL LINES OF CODE MIGHT BE NECESSARY TO DEFINE WHAT ONE WANTS TO DO.
E.G.:

```
.......
LOC=24
LAB=LAB2
PRO-CC=(USE-A);A=GTOD;
GTO LAB1
APB-SRC=IZ;FUNC=R+S;DEST=QF;
APM-SRC=IZ;FUNC=RXNS;DEST=BF;B=1;
ARI-M1A=XINT;S1=A;M12=M1;
ACC-SET1=YES                          %FF1 := 1
OUT-XFER=NO                           %NO TRANSFER OF DATA
I/O-SETF=YES                          %SIGNAL AT EXT. CONNECTOR
%
%END OF THIS STATEMENT - START NEXT
%
NXT
.....
```

NOTE THAT FOR ONE LOCATION ANY NUMBER OF INSTRUCTIONS STARTING WITH KEYWORDS PRO,APB,APM,ARI,ACC,OUT AND I/O CAN BE SPECIFIED

THE LAST KEY-WORD OF A PROGRAM MUST ALWAYS BE END.
NOTE AGAIN THAT ONLY ACTIVE PROGRAM FIELDS NEED BE DEFINED. AN ACTIVE PROGRAM FIELD IS A PROGRAM FIELD THAT DOES SOMETHING MEANINGFUL. (SO FOR EXAMPEL LOOKING AT THE PROGRAM ABOVE, ONE CAN SEE THAT THE INSTRUCTION 'OUT-XFER=NO' ACTUALLY IS SUPERFLUOUS.)

8.1 THE DUMMY STATEMENT DEFINITION

```
LOC=0
PRO-CC=(USE-A);A=GTO;B=GTO;LC1=NOOP;LC2=NOOP;LC3=NOOP;
PRO-LC1A=NOOP;RELD=NO;RADR=SAR;ADDR=00;
%
APB-SRC=ZQ;FUNC=RNDS;DEST=NOOP;A=0;B=0;SEL=NO;
APM-SRC=ZQ;FUNC=RNDS;DEST=NOOP;A=0;B=0;
%
ARI-M1A=ONE;M2A=ONE;M3A=ONE;M4A=ONE;
ARI-M1B=YEXT;M2B=YEXT;M3B=YEXT;M4B=YEXT;S1=NOOP;S2=NOOP
ARI-S3=NOOP;S4=NOOP;M12=MIN1;M34=MIN1
%
ACC-SIO=NO;WRIT=NO;SET1=NO;READ=NO;CLR1=NO;SET2=NO;CLR2=NO
%
OUT-XFER=NO;INHC=NO;RDY=NO;XCOD=STAT;SRC=MSTR
%
I/O-SETF=NO;CLRF=NO;SBUF=NO;STRI=NO;EDB=NO;EAB=NO
%
END
```

9. USING CORPREP

9.0 INVOKING CORPREP AND TALKING TO IT

THE CORPREP IS ON THE NORD-10 INVOKED BY ISSUING THE FOLLOWING COMMAND TO SINTRAN:
ECORPREP

CAUSING THE RESPONSE

```
******   E I S C A T  CORRELATOR PREPASS  ******
SOURCE-FILE:<USER SOURCE FILE>
OBJECT-FILE:<USER OBJECT FILE>
LIST DEVICE:<DEVICE NAME>
CONDITIONALS:<CONDITION CHARACTERS>
```

<USER SOURCE FILE> CAN BE ANY FILE WHERE THE USER HAS PUT HIS CORRELATOR PROGRAM.

<USER OBJECT FILE> CAN BE ANY FILE WHERE FROM THE USER WANTS TO LOAD THE CORRELATOR. (IF NO OBJECT-CODE IS WANTED, JUST TYPE RETURN.) THE FILE IS AUTOMATICALLY CREATED IF IT DOES NOT EXIST.

<DEVICE-NAME> COULD BE EITHER /L-P/ FOR LINE-PRINTER OR /TERM/ FOR TERMINAL.

<CONDITION CHARACTERS> COULD BE ANY CHARACTERS DEFINING THE CONDITION CODE IN THE USER PROGRAM. (MAXIMUM 5 CHARACTERS CAN BE DEFINED) ALSO SEE 7.

UPON FINISH THE FOLLOWING IS DISPLAYED:

```
NO ERROR DETECTED
OR
<NN> ERROR(S) DETECTED
```

NOTE THAT NO OBJECT CODE IS WRITTEN ANYWHERE IF ERRORS ARE DETECTED.

9.1 AN EXAMPLE

```
ECC INVOKE CORPREP

E(CORR)CORP
******   E I S C A T  CORRELATOR PREPASS-VI  ******
SOURCE FILE:(TORU)CORR-PROG
OBJECT FILE:(TORU)CORR-OBJ
LIST DEVICE:TERM
CONDITIONALS:L
LAB=ZERO
LOC=0
IDL                              %IDLE LOOP
NXT                              %START PROG. NEXT LOCATION

##### MEMORY LOCATION: 01

PRO-CC=(USE-A);A=CON;LC1=LCR1;
NXT                              %START PROGRAMMING LOC. 2

##### MEMORY LOCATION: 02

&L                               %THE CODE UP TO FIRST '&'
%                                %WILL BE ASSEMBLED
%                                 IF CONDITIONAL L IS SPEC.
%                                 OTHERWISE IT IS IGNORED.
LAB=LOOP
&
PRO-CC=(USE-A);A=CON;LC1=DEC;
NXT

##### MEMORY LOCATION: 03

PRO-CC=(IF LC1=0 THEN B ELSE A);
PRO-B=CON;A=GTO;
GTO=LOOP
NXT

##### MEMORY LOCATION: 04

PRO-CC=(USE-A);A=GTO;
GTO ZERO
REG-LCR1=40;                     %DEFINE LOOP-COUNTER 1
END
SUBMIT A TITLE FOR YOUR PROGRAM (UP TO EXCL. MARK)
THE TITLE FACILITY IS FOR CORRSIM (REF 2)

NO ERROR DETECTED
*** END OF PREPASS ***
```

APPENDIX A: REFERENCES

REF 1:H.J. ALKER - "INSTRUCTION MANUAL FOR EISCAT DIGITAL CORRELATOR"
(EISCAT BLUE BOOK 79/10)

REF 2:H.J. ALKER - "PROGRAM CORRSIM: SYSTEM FOR PROGRAM DEVELOPMENT AND SOFTWARE SIMULATION OF EISCAT DIGITAL CORRELATOR"
(EISCAT BLUE BOOK 79/9)

APPENDIX B: A TABLE OF ALL THE AVAILABLE TESTS FOR MICRO-PROGRAM BRANCHING

```
PRO-CC=(USE-A);

PRO-CC=(IF LC1=0 THEN B ELSE A);

PRO-CC=(IF LC2=0 THEN B ELSE A);

PRO-CC=(IF LC1=0 OR LC2=0 THEN B ELSE A);

PRO-CC=(IF LC3=0 THEN B ELSE A);

PRO-CC=(IF LC1=0 OR LC3=0 THEN B ELSE A);

PRO-CC=(IF LC2=0 OR LC3=0 THEN B ELSE A);

PRO-CC=(IF LC1=0 OR LC2=0 OR LC3=0 THEN B ELSE A);

PRO-CC=(IF LC2#0 THEN B ELSE A);

PRO-CC=(IF LC1=0 OR LC2#0 THEN B ELSE A);

PRO-CC=(IF LC2#0 OR LC3=0 THEN B ELSE A);

PRO-CC=(IF LC1=0 OR LC2#0 OR LC3=0 THEN B ELSE A);

PRO-CC=(IF LC3#0  THEN B ELSE A);

PRO-CC=(IF LC1=0 OR LC3#0 THEN B ELSE A);

PRO-CC=(IF LC2=0 OR LC3#0 THEN B ELSE A);

PRO-CC=(IF LC1=0 OR LC2=0 OR LC3#0 THEN B ELSE A);

PRO-CC=(IF LC2#0 OR LC3#0 THEN B ELSE A);

PRO-CC=(IF LC1=0 OR LC2#0 OR LC3#0 THEN B ELSE A);

PRO-CC=(IF LC1=0  THEN B ELSEIF  LC2=0  THEN A OTHERWISE CONT);

PRO-CC=(IF LC3=0  THEN B ELSEIF  LC2=0  THEN A OTHERWISE CONT);
```

```
PRO-CC=(IF LC1=0 OR LC3=0 THEN B ELSEIF  LC2=0 THEN  A OTHERWISE CONT);
PRO-CC=(IF LC1=0  THEN B ELSEIF  LC2#0  THEN A OTHERWISE CONT);
PRO-CC=(IF LC3=0  THEN B ELSEIF  LC2#0  THEN A OTHERWISE CONT);
PRO-CC=(IF LC1=0 OR LC3=0 THEN B ELSEIF  LC2#0  THEN A OTHERWISE CONT);
PRO-CC=(IF LC1#0  THEN B ELSEIF  LC2=0  THEN A OTHERWISE CONT);
PRO-CC=(IF LC3#0  THEN B ELSEIF  LC2=0  THEN A OTHERWISE CONT);
PRO-CC=(IF LC1#0 OR LC3#0 THEN B ELSEIF  LC2=0  THEN A OTHERWISE CONT);
PRO-CC=(IF LC1#0  THEN B ELSEIF  LC2#0  THEN A OTHERWISE CONT);
PRO-CC=(IF LC3#0  THEN B ELSEIF  LC2#0  THEN A OTHERWISE CONT);
PRO-CC=(IF LC1#0  OR LC3#0 THEN  B ELSEIF LC2#0 THEN A OTHERWISE CONT);
PRO-CC=(IF LC1=0 OR LC3=0 THEN B OTHERWISE CONT);
PRO-CC=(IF LC1#0 OR LC3#0 THEN B OTHERWISE CONT);
```

EISCAT SCIENTIFIC ASSOCIATION
S-981 01 KIRUNA 1, SWEDEN
TELEPHONE 0980/187 40
TELEX 8754 GEOFYSK S