4. Programming language XPLC

4.1 Function range

XPLC actual a programming software for BWO compact controls CNC 902/903/904/905.
The software actual on all Windows (ME, XP, NT) and Linux PC's executably. The connection is made by an Ethernet Interface (TCP/IP).

XPLC possesses the following function range:

- PLC Programming in LogiCode and AWL
- Symbolic addressing and allocation of symbol terms
- Function editor
- Symbol editor
- Standard function blocks (LogiCode)
- Creation of user specific functions
- On-line debuggers
- On-line test and diagnostic functions
- Programming of the EEPROM memory

4.2 Syntax

The syntax of the programming language actual a subset of the C language.

Used items are:

- Symbols
- Symbolic addresses
- Types
- Mathematical and logical operators
- Separators
- Comments
- C commands

4.3 Symbols (symbolic terms)

On symbol or symbol term actual a consequence of letters and digits, which may contain no separators and start with a letter must.

for example:
i
test
alfa
f00123
Ablink
Simul

123foo does not permit!

4.4 Symbolic addresses

are fixed addresses for

Inputs: e1.1.1
Outputs: a1.1.1
Flags: m1.1

Free symbol terms can be assigned to the symbolic addresses, which are used in the program like symbolic addresses.

	symbolic addresses		symbolic terms	
Inputs:	e1.1.1	—>	ELT-ON	comment
Outputs:	a1.1.1	—>	Ablink	comment
Flags:	m200.3	—>	Simul	comment

4.5 Type definition

4.5.1 Simple (pre-defined) types

bit }	Using only the LSB bit
bool }	
char }	1 byte and sign (limiting values : -128 to +127)
byte }	
int }	
short }	2 byte and sign (limiting values: -32768 to +32767)
word }	
long }	4 byte and sign (limiting values: -7FFFFFFF to +80000000)
fixed }	4 byte (limiting values: -2000000 to +2000000) Type for fixed point arithmetic

Each not defined symbol is a bit and/or a byte

4.5.2 Combined types (in the listing types)

May only of simple types consist.
Can be created you as required.

counter Up / downward counters

bit out
bit trig
long value

toggle Toggle
bit out
bit trig

timer Ozillator, in-/ turn-off delay

bit out
bit enabled
byte typ
bit state
long counter
long max

parameter CNC parameter
byte fill
long nummer
long mantisse
long status
word exponent
byte cmd

4.5.3 Further one combined types

Uses for BWO standard modules.

System-Calls

syscall_129 Timer

```
byte  number
long  timer_ad
```

syscall_130 CAN Bus

```
byte  number
byte  cmd
byte  node
byte  bus_status
byte  adress
byte  present
byte  status
word  error
byte  error_register
byte   custom_error_0
byte   custom_error_1
byte   custom_error_2
byte   custom_error_3
byte   custom_error_4
```

syscall_131 Analog in/output

```
byte  number
byte  read
byte  channel
word  value
```

4.6 Mathematical and logical operators

+	Addition			
-	Subtraction			
*	Multiplication	-->	or contents of …	with pointer operations
/	Division			
%	Remainder of the division			
&	Bit by bit AND	-->	or address of …	with pointer operations
|	Bit by bit OR			
^	Bit by bit XOR			
	Negation			
&&	Logical AND			
||	Logical OR			

==	directly
!=	unequally
>	more largely
|	smaller
> =	more largely directly
< =	smaller directly
=	Allocation

Trennzeichen
alle Operatoren und folgende Zeichen
;
,
(
)
{
}

comments

/ /	Line comment
/ *	
	Comment block
*/	

4.7 Instruction in LogiCode

Simple instruction become also ; terminated.
Assembled instruction are summarized by { } brackets in a block.
Within the parentheses individual instruction can be, in each case also ; are terminated.

Allocation: =

```
LED1 = REG_FR;
```

if: Conditioned versions fulfills

```
if (HAND_K1 & REOK_K1 & TA6)  condition
{
M03_K1 = 1;                   fulfills
M04_K1 = 0;
}
```

if else: Conditioned version fulfilled / does not fulfill

```
if (Ref1)                     condition
{
Refz1 = 5;                    fulfills
}
else
        {
        REPOP1 = RPFMA1;      does not fulfill
        REPOM1 = RPFPA1;
        REFZ1 = 1;
        }
```

while: Loop with test at the start

```
i=0;
while (i < 5)                 test at the start
{
        i+1;
}
```

4.7 Instruction in LogiCode (continued)

do while: Loop with test at the end

```
i=0;
do {
        i+1;
   } while ( i > 5 );          Test at the end
```

switch, case, break, default

instruction for multiple branches (state machine)

Example:

```
switch (SK1)                    query
{
        case 0                  branch 1
                Sk1 = 1;
                break;          end

        case 1                  branch 2
                SK1 = 2;
                break;

        case 2
                SK1 = 3;
                break;

        case 3:
                SK1 = 4;
                break;

        case 4:
                SK1 = 5;
                break;

        case 5:
                SK1 = 0;
                break;

        default;

                break;
```

4.8 Instruction in LogiCode (C-Code)

Example of a function in LogiCode (C-CODE)

// **fillmemory**

```
                        // interface definition
byte  *memory;          // memory actual on pointers on on byte
long  count;            // number of cells which can be filled
byte  pattern;          // samples
                        // end of the interface

{ // beginning the C code statements for the function

      long i;           // counting variable

      i=0;
      while (i < count)
            {
            *mamory = pattern;        Contents pattern are written the position,
                                      on the memory show
             memory = memory+1;       pointers increase
            i = i+1;                  loop +1
            }

} // end the C code statements
```

Function call

fillmemory(&start, number, samples);

Sample application

fillmemory(&ME50,30,0x00);

Start:	ME50	flag
Number:	30	constant
Sample:	0x00	

4.8 Instruction in LogiCode (C-Code) (continued)

Example of a function in statement list

```
*
*  move
*
*  Interface definition
*
interface 0        * start of the interface        amount of local variable
dst                * target
src                * source
count              * number of items
end                * end of the interface
*
*  Beginning of the statements for the function
*
       fwd
       lrs         src
       lrs         dst
       lrs         count
       emove
*
*  End of the statements for the function
*
```

Function call

move(Quelle, target, Anzahl);

Sample application

move(&M100.1, M150.1, 32);

Source:	M100.1	flag
Target:	M150.1	flag
Number	32	constant

4.9 Standard functions blocks

COUNTER UP **c_CTU**

Call

c_CTU(&Counter_x, Enable, Reset, Max)

Example

M8_TEST = c_CTU(&Z1, TA9, TA11, 20);

Counter_x:	Z1	Type Counter 1
Enable:	TA9	Flag PLC Key 09
RESET:	TA11	Flag PLC Key 11
Max one:	20	constant

M8_TEST is settinged, if count value achieves max (= 20) .

COUNTER DOWN **c_CTD**

Call

c_CTD(&Counter_x, Enable, RESET, Max)

Example

M9_TEST = c_CTD(&Z2, TA10, TA11, 10);

Counter_x:	Z2	Type Counter 2
Enable:	T10	Flag PLC Key 09
Reset:	T11	Flag PLC Key 11
Max one:	10	constant

M9_TEST is settinged, if count value achieves 0 .

4.9 Standard functions blocks (continued)

TRIGGER **c_TRIG**

Call

c_TRIG(&Trig_x, In)

Example

```
if (c_TRIG(&TRIG1, TA8))
{
M10_TES = 1;
}
```

Trig_x:	TRIG1	flag trigger	1
In:	TA8	flag PLC Key 8	

Flag M10_TES is settinged with positive edge by TA8 .

RS-FLIP-FLOP **c_RS**

Call

c_RS(&RS_x, set, Reset)

Example

ATSTR1 = c_RS(&RS1, TA12, TA13);

RS_x:	RS1	flag RS flip flop 1
Set:	TA12	flag PLC key 12
RESET:	TA13	flag PLC key 13

Output ATSTR1 is settinged with indicator TA12, reset with TA13 .

4.9 Standard functions blocks (continued)

T-FLIP-FLOP **c_TOGGLE**

Call

c_TOGGLE(&Toggle_x, In)

Example

M2_TEST = c_TOGGLE(&TOGG5, TA2);

Toggle_x:	TOGG5	Type Toggle
In:	TA2	Flag PLC Key 2

Flag M2_TEST is settinged or reset alternately by flag TA2

OSZILLATOR **c_OSZ**

Call

c_OSZ(&Timer_x, Enable, Zeit)

Example

ABLINK = c_OSZ(&TIMER1,!PWON, 1000)

Timer_x:	TIMER1	type timer
Enable:	PWON	flag power on
Time (in ms):	1000	constant (1000 ms)

4.9 Standard functions blocks (continued)

SWITCH-OFF DELAY **c_TOFF**

Call

c_TOFF(&Timer_x, Enable, Zeit)

Example

MFRMDR = c_TOFF(&TIMER8, FRMOTUI, 1000);

Timer_x:	TIMER8	type timer 8
Enable:	FRMOTUI	input
Time (in ms):	1000	constant (1000 ms)

If milling motor stop: (input FRMTUI = L) flag MFRMDR delays reset

SWITCH-ON DELAY **c_TON**

Call

c_TON(&Timer_x, Enable, Zeit) time in ms

Example

```
if (c_TON(&TIMER7, M41_K1, 2000))
{
M41_K1 = 0;
}
```

Timer_x:	TIMER7	type timer 7
Enable:	M41_K1	flag
Time (in ms):	2000	constant

4.9 Standard functions blocks (continued)

READING PARAMETER BY CNC **LPARC**

Call

LPARC(Ziel, register, channel, p-number, Enable)

Example

LPARC(&P410E, 1, 410, m11_TES)

or

```
if (c_TRIG(&TRIG4, E1.1.6))
{
LPARC(&P410E, 1, 410, e1.1.6);
}
```

Target register:	P410E	long
Channel:	1	constant
Parameter number:	410	constant
Enable:	E1.1.6	input

4.9 Standard functions blocks (continued)

WRITE PARAMETERS CNC **SPARC**

Call

SPARC(Kanal, p-number, mantissa, exponent, status, command, Enable)

Example

```
if (c_TRIG(&TRIG2, E1.1.5))
{
SPARC(1,405,P405WE,0,0,0,E1.1.5);
}
```

or

SPARC(1,406,P406WE,0,0,0,c_TRIG(&TRIG3,E1.1.5));

Channel:	1	constant
Parameter number:	406	constant
Parameter mantissa:	P406WE	long
Parameter exponent	0	constant
Parameter status:	0	constant
Parameter command:	0	constant
Enable:	E1.1.5	input

4.9 Standard functions blocks (continued)

ANALOGUE VALUE INPUT **analog_in**

(2 channel analogue input clip)

Call

Input value = analog_in (node no, clip no, channel no)

Input value register (in mV):	-10000 - + 10000
Node no:	1 - 5
Analog input clip no:	1, 2
Channel no (clip):	1, 2

Example

```
ADW1 = analog_in(2,1,1);
```

or

```
if(ADWEN1)
{
ADW1 = analog_in(2,1,1);
}
```

Input value register (in mV):	ADW1	long
Node no:	2	
Analog input clip no:	1	
Channel no (clip):	1	

4.9 Standard functions blocks (continued)

ANALOGUE VALUE OUTPUT **analog_out**

(2 channel analogue output clip)

Call

Analog_out(node no, clip no, channel no, output voltage)

Node no:	1 - 5
Analog output clip no:	1, 2
Channel no (clip):	1, 2
Output voltage in mV:	-10000 - +10000

Example

```
analog_out(2,1,1,6000);
```

or

```
if(SPIRE1)
{
analog_out(1,1,1,q1656W);
}
else
{
        analog_out(1,1,1,0);
}
```

Node no:	1	constant
Analog output clip NR:	1	constant
Channel NR (clip):	1	constant
Output voltage in mV:	q1656W/0 long/Konstante	

4.9 Standard functions blocks (continued)

MOVE FLAG AREA **MOVE**

Call

move(Source, target, number)

Example

move(&C1_ER0, s130 + 10.6);

Further functional modules uses for BWO standard modules

TIMER LOG ON TO THE SYSTEM **c_register_timer**

Call

c_register_timer

ALL TIMERS LOG ON AUTOMATICALLY **c_setup_timer**

Call

c_setup_timer

ALL TIMERS LOG ON AUTOMATICALLY **syscall**

Call

syscall(number, area)

4.10 Assembler (XASM) instruction

In the following all usable assembler (XASM) instruction (statement list) are listed.
The following notation use finds:

const = Numbers or time constant (decimal: 534, in hexadecimals: $$A436, octal: ‘ 346
paddr = Program address (absolute: $$F9B2, symbolically: START, relative: * +39)
maddr = Storage address (absolute: $610, symbolically: M98.1)

General ones of instruction

NO OPERATION **NOP**
Dummy instruction. NOP actual always in a program line available (substitute symbols).

WORD SIZE **SIZE [4,2]**
Word width for certain instruction of 4 to 2 byte switch and in reverse.

COUNT ADDRESS FORWARD, COUNT ADDRESS BACKWARD **FWD, BWD**
After the instruction FWD or BWD is advanced with certain storage instructions the address in ascending or descending order.

4.10 Assembler (XASM) instruction (continued)

Instruction for boolsche algebra

IS load operand 2 **L maddr, LN maddr, L(maddr, LN(maddr**
Drawer contents from operand 2 to IS register. Use with negation N and brackets (.

IS AND operand 2 **U maddr,UN maddr,UN(maddr**
Logical AND linkage IS register with contents of operand 2. Use with negation N and brackets (. The result of the linkage is stored in the IS register.

IS OR operand 2 **O maddr, ON maddr, ON(maddr**
Logical OR linkage IS register with contents of operand 2. Use with negation N and brackets (. The result of the linkage is stored in the IS register.

LOAD operand 2 IS **= maddr, = N maddr**
Allocation of contents of the IS register to operand 2. Use with negation N.

SET operand 2 **S maddr, SN maddr**
Settinging contents of operand 2 if the IS register a 1 contains. With use with negation N: Setting contents of operand 2 if the IS register 0 contains maddr may be only flag or original address.

RESET operand 2 **R maddr, RN maddr**
Resetting contents of operand 2 if the IS register a 1 contains. With use with negation N: Reset contents of operand 2 if the IS register 0 contains maddr may be only flag or original address.

4.10 Assembler (XASM) instruction (continued)

Instruction for memory influence

LOAD MEMORY HIGH, LOAD MEMORY LOW **LDH maddr, LDL maddr**
Loading of a memory byte with 1 or with 0. maddr may be only flag or original address.

LOAD operand 2 X, LOAD operand 2 Y **LX maddr,LY maddr**
Allocation of contents of the x-register or y-register after operand 2.
Cross reference: Y LOAD

LOAD operand 2 XY **LXY maddr**
Allocation of the total of contents from X and y-register to operand 2.
Cross reference: LX,LY,X LAD,Y LAD

LOAD [X] IS **ISRX**
Loading of the memory byte, whose address is located in the x-register, with contents of the IS register. The address in the x-register may be only flag or original address. The value in the x-register is incremented thereafter.
Cross reference: XISR

FILL MEMORY [X] **FILL const n**
Loading of n memory byte with const, starting with the storage address, which is located in the x-register. The storage address in the x-register may be only flag or original address.
Cross reference: EFILL,FWD,BWD,X LAD

FILL MEMORY [TOS] **EFILL**
Loading of n memory byte with a constant, which is in the TOS register. The address of the first memory byte is just like the number of n of the memory bytes which can be filled in the TOS register:

n	=	TOS
maddr	=	TOS-1
const	=	TOS-2

Frame around three registers decremented maddr may only flag or original address be.
Cross reference: FILL,FWD,BWD,LCS,LCHS

4.10 Assembler (XASM) instruction (continued)

Instruction for memory influence

MOVE MEMORY [X,Y] **MOVE const**

Copying contents of const memory bytes. Source address and destination address are in the index registers:

Source address = Y-register
Destination address = X-register.

Destination address may be only flag or original address.
Cross reference: FWD,BWD

MOVE MEMORY [TOS] **EMOVE**

Copying contents of n memory byte. n, source address and destination address are located in the TOS register:

n = TOS
Destination address = TOS-1
Source address = TOS-2

Frame is decremented around three registers. Destination address may be only flag or original address.
Cross reference: LCS,LCHS,FWD,BWD

4.10 Assembler (XASM) instruction (continued)

Instruction for register influence

LOAD X REGISTER, LOAD Y REGISTER **X LAD const, Y LAD const**
Loading of the X or Y register with the value const.
Cross reference: LSX,LSY

LOAD STACK TO X, LOAD STACK TO Y **LSX, LSY**
Loading of the X or Y register with the value, which is in the Low byte of the TOS register. Frame is decremented over on registers.
Cross reference: LCS,XLAD,YLAD

EXCHANGE X Y **EXY**
Exchange contents of x-register and y-register.
Cross reference: LCS,XLAD,YLAD,LSX,LSY

LOAD INDIRECT IS [X] **XISR**
Loading of the IS register with contents of the memory byte, of its address in the x-register contain actual. The value in the x-register is decremented thereafter!
Cross reference: ISRX,XLAD

LOAD CONSTANT TO STACK **LCS const**
Loading of the lower 16-Bit of the TOS register (Low Word) with const. The High Word remains uninfluenced. Frame is incremented over on registers.

LOAD CONSTANT TO HIGH STACK **LCHS const**
Loading of the upper 16-Bit of the TOS register (High Word) with const The Low Word and frame remain uninfluenced.

LOAD X TO STACK, LOAD Y TO STACK **LXS,LYS**
Loading of contents of the x-register or y-register in the TOS register. The High Word remains uninfluenced. Frame is incremented over on registers.

4.10 Assembler (XASM) instruction (continued)

Instruction for register influence

LOAD REGISTER TO STACK **LRS maddr**

Loading contents of maddr to maddr+3 (4 byte) or of maddr and maddr+1 (2 byte) in the TOS register. Whether 2 or 4 byte to be loaded depends on the respective adjustment over the instruction SIZE. Frame is incremented around two or on registers.
Cross reference: SIZE,LSR,BLRS

LOAD STACK TO REGISTER **LSR maddr**

Loading of the memory bytes maddr to maddr+3 (4 byte) or maddr and maddr+1 (2 bytes) with contents of the TOS register. Whether 2 or 4 byte to be loaded depends on the respective adjustment over the instruction SIZE. Frame around two or on registers decremented maddr may be only flag or original address.
Cross reference: SIZE,LRS,BLSR

BYTE LOAD REGISTER TO STACK **BLRS maddr**

Loading TOS registers with contents of maddr (1 byte). Frame is incremented over on registers.
Cross reference: SIZE,BLSR,LRS,LSEA

BYTE LOAD STACK TO REGISTER **BLSR maddr**

Loading of the memory byte maddr with contents of the TOS register. Frame over on registers decremented maddr may be only flag or original address.
Cross reference: SIZE,BLRS,LSR,LEAS

LOAD EA TO STACK **LEAS**

Loading TOS registers with n of the initially or output bit. Only the niederwertigste bit (bit 0, LSB) is used and stored by contents of the in and original addresses in the TOS register bit-oriented. n and source address are located in the TOS register:

n	=	TOS
Address Eingang1/Ausgang1	=	TOS-1

n<=32. frame is incremented over on registers.
Cross reference: FWD,BWD,LRS,BLRS

4.10 Assembler (XASM) instruction (continued)

Instruction for register influence

LOAD STACK TO EA **LSEA**

Copy from n bits from the TOS register to n original addresses. Each bit in the TOS register is put on an original address. n and destination address are located in the TOS register:

n	=	TOS
Destination address	=	TOS-1
Bit 1	=	TOS-2

n<=32. Frame is decremented over on registers.
Cross reference: LCS,LCHS,FWD,BWD,LSR,BLSR

DUPLICATE TOS **DUP**

Copy from TOS-1 to TOS. Frame over on registers is incremented.
Cross reference: POP

SWAP TOS **SWAP**

Interchange of TOS-1 and TOS. Frame remains unchanged.

DECREMENT SP **POP**

TOS select. Frame is decremented over on registers.
Cross reference: DUP

LOAD EEPROM-CHECKSUM **CKS**

Loading of the EEPROM Check total in the TOS register. Frame is incremented over on registers.

LOAD STACK POINTER **LSP**

Loading of frame in the TOS register. Frame is incremented over on registers.

4.10 Assembler (XASM) instruction (continued)

Arithmetic instruction

INCREMENT MEMORY **INC maddr**
Increment the 32-Bit of memory word, which is in the memory in the addresses maddr to maddr+3 (4 byte).
Cross reference: DEC

DECREMENT MEMORY **DEC maddr**
Decrement the 32-Bit of memory word, which is in the memory in the addresses maddr to maddr+3 (4 byte)
Cross reference: INC

ADD TOS WITH TOS-1 **ADD**
32-Bit addition of the value in the TOS register with the value in TOS-1. The result is stored in the TOS register, SP are decremented over on registers.
Cross reference: LCS,LCHS

SUBTRACT TOS-1 WITH TOS **SUB**
32-Bit subtraction of the value in the TOS register of the value in TOS-1. The result is stored in the TOS register, SP are decremented over on registers.
Cross reference: LCS,LCHS

MULTIPLY TOS WITH TOS-1 **MUL**
32-Bit multiplication of the value in the TOS register with the value in TOS-1. The result is stored in the TOS register, SP are decremented over on registers.
Cross reference: LCS,LCHS

DIVIDE TOS-1 WITH TOS **DIV**
32-Bit division of the value in TOS-1 by the value in the TOS register. The result is stored in the TOS register, SP are decremented over on registers.
Cross reference: LCS,LCHS,REM

4.10 Assembler (XASM) instruction (continued)

Arithmetic instruction

MODULO DIVISION TOS-1 WITH TOS **REM**

32-Bit modulo division of the value in TOS-1 by the value in the TOS register. The result is stored in the TOS register, SP are decremented over on registers.
Cross reference: LCS,LCHS,DIV

NEGATION TOS **NEG**

Reversal of sign at the value, which is located in the TOS register. SP remains unchanged.
Cross reference: LCS,LCHS

ADD CONSTANT TO X, ADD CONSTANT TO Y **XIADD const, YIADD const**

16-Bit addition of the value in the x-register or y-register with the value const. The result is stored in x-x-bzw. the y-register.
Cross reference: XLAD,YLAD,ADD

4.10 Assembler (XASM) instruction (continued)

Logical 32-Bit of instruction

LOGIC AND TOS-1 WITH TOS **AND**
Logical AND linkage bit by bit of the bit pattern in the TOS register with the bit pattern in TOS-1. The result is stored in the TOS register, SP are decremented over on registers.
Cross reference: LCS,LCHS

LOGIC OR TOS-1 WITH TOS **OR**
Logical OR linkage bit by bit of the bit pattern in the TOS register with the bit pattern in TOS-1. The result is stored in the TOS register, SP are decremented over on registers.
Cross reference: LCS,LCHS

LOGIC NOT TOS **NOT**
Logical NOT linkage bit by bit of the bit pattern in the TOS register with the bit pattern in TOS-1. The result is stored in the TOS register, SP are decremented over on registers.
Cross reference: LCS,LCHS

LOGIC EXCLUSIVE OR TOS-1 WITH TOS **XOR**
Logical exclusive OR linkage bit by bit of the bit pattern in the TOS register with the bit pattern in TOS-1. The result is stored in the TOS register, SP are decremented over on registers.
Cross reference: LCS,LCHS

LOGIC SHIFT TOS LEFT **SFTL**
Shift bit by bit of the bit pattern in TOS -1 around in the TOS register to the left. The result is stored in the TOS register, SP are decremented over on registers.
Cross reference: LCS,LCHS,SFTAR

LOGIC SHIFT TOS RIGHT **SFTR**
Shift bit by bit of the bit pattern in TOS -1 around in the TOS register to the right. The result is stored in the TOS register, SP are decremented over on registers.
Cross reference: LCS,LCHS,SFTAR

LOGIC SHIFT TOS ARITHMETIC RIGHT **SFTAR**
Arithmetic shift bit by bit of the bit pattern in TOS -1 around in the TOS register to the right. The TOS register contains a signed value. The result is stored in the TOS register, SP are decremented over on registers.
Cross reference: LCS,LCHS,SFTL,SFTR

4.10 Assembler (XASM) instruction (continued)

Jump instructions

JUMP IMMEDIATE **JMP,SP paddr**
Absolute jump to paddr (absolute, relative or symbolic address).
Cross reference: JMPT,JMPF,SPB,SH,SL

JUMP IF TOS TRUE **JMPT paddr**
Conditioned jump to paddr (absolute, relative or symbolic address), if the lowest bit (Bit0, LSB) of the TOS register 1 actual
Cross reference: JMP,JMPF,SPB,SH,SL

JUMP IF TOS FALSE **JMPF paddr**
Conditioned jump to paddr (absolute, relative or symbolic address), if the lowest bit (Bit0, LSB) of the TOS register 0 actual
Cross reference: JMPT,JMP,SPB,SH,SL

SKIP IF MEMORY HIGH **SH** n **maddr**
Conditioned jump for n of instruction, if contents of the memory byte maddr 1 (High) actual.
Cross reference: JMPT,JMPF,JMP,SPB

SKIP IF MEMORY LOW **SL** n **maddr**
Conditioned jump for n of instruction, if contents of the memory byte maddr 0 (Low) actual.
Cross reference: JMPT,JMPF,JMP,SPB

CONDITIONED JUMP IS **SPB paddr, SPBN paddr**
Conditioned jump to paddr (absolute, relative or symbolic address), if in IS register the value 1 or 0 is.
Cross reference: JMPT,JMPF,JMP,SH,SL

DECODER [TOS] **DCD paddr**
Indirect calculated jump. The destination address results off contents of the TOS register and paddr (absolute or symbolic address). Frame is decremented over on registers.

4.10 Assembler (XASM) instruction (continued)

Jump instructions

DECREMENT X AND JUMP IF ZERO **XDCR** n

Conditioned jump for n of instruction. Contents of the x-register are first decremented. Afterwards tested whether the x-register the value zero contains. Actual this the case n the following instruction are skipped.

DECREMENT Y AND JUMP IF ZERO **YDCR** n

Conditioned jump for n of instruction. Contents of the y-register are first decremented. Afterwards tested whether the y-register the value zero contains. Actual this the case n the following instruction are skipped.

4.10 Assembler (XASM) instruction (continued)

Comparing instruction

TEST IF TOS-1 EQUAL TO TOS **TEQ**

Comparison of TOS register contents TOS and TOS-1. The result is stored into TOS:
TOS = TOS-1: TOS=1
TOS <> TOS-1: TOS=0
SP over on register is decremented.

TEST IF TOS-1 GREATER THAN TOS **TGT**

Comparison of TOS register contents TOS and TOS-1. The result is stored into TOS:
TOS-1 > TOS: TOS=1
TOS-1 <= TOS: TOS=0
SP over on register is decremented.

TEST IF TOS-1 LESS THAN TOS **TLT**

Comparison of TOS register contents TOS and TOS-1. The result is stored into TOS:
TOS-1 < TOS: TOS=1
TOS-1 >= TOS: TOS=0
SP over on register is decremented.

4.10 Assembler (XASM) instruction (continued)

BCD conversion instructions

CONVERT INTEGER TO BCD TOS **IBCD**
Changes contents of the TOS register from the Integer format to BCD format. SP remains unchanged.
Cross reference: BCDI

CONVERT BCD TO INTEGER TOS **BCDI**
Changes contents of the TOS register from the BCD format to Integer format. SP remains unchanged.
Cross reference: IBCD