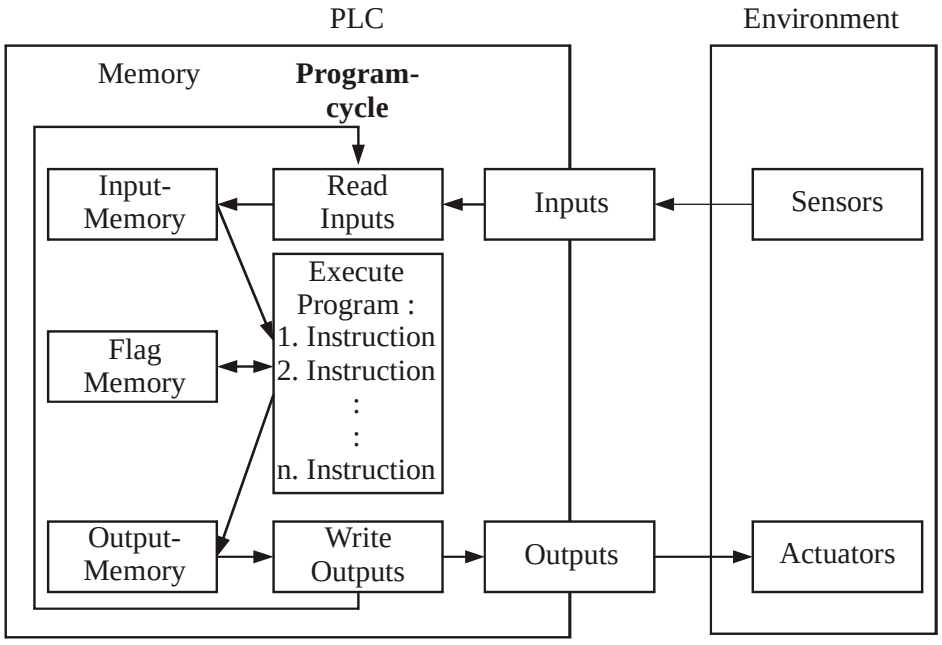
Programmable Logic Controller (PLC)
Speicherprogrammierbare Steuerung (SPS)

Literature

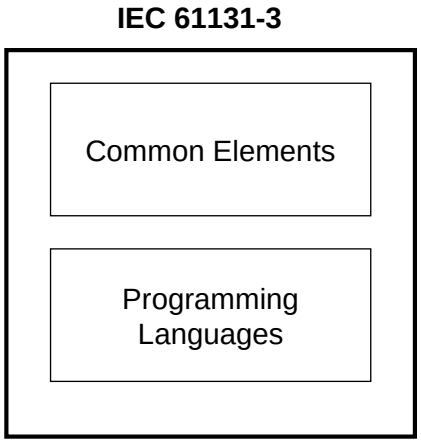
R. W. Lewis:

“Programming industrial control systems using IEC 1131-3“

PLC Hardware



Software Architecture following IEC 61131-3



- **Common Elements**
  - Configurations
  - Resources
  - Tasks
  - Programs
  - Functions
  - Function Blocks
  - Data Types
  - Variable declarations
  - Initial values
- **Programming Languages**
  - Ladder Diagram (LD)
  - Function Block Diagram (FBD)
  - Instruction List (IL)
  - Structured Text (ST)
  - Sequential Function Chart (SFC)

Code Structure

Configuration
- Defines the software for a complete PLC
- Contains one or more resources
- Is specific to a particular type of PLC product and the arrangement of the PLC hardware
- Considers
  - Processing resources (type of micro processor)
  - Memory addresses (I/O)
  - System capabilities (max. number of tasks and execution rates)

Resource
- Corresponds to a processing facility
- Contains variable declarations, tasks and program declarations

Task
- Executes periodically or in response to a particular boolean variable state
- Runs a program or a function block

Program
- Largest form of program organization
- Contains variable declarations and a body which contains software describing the program's behavior
- Can call programs, function blocks and functions
- Can only be declared within resources

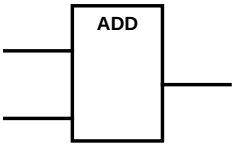
Functions and Function Blocks

Functions

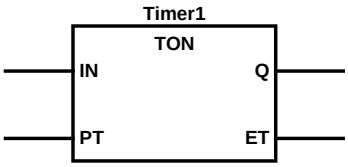
- Calculates a result depending on inputs
- Reusable
- Multiple Input
- Single Output
- Can **not** store values within internal variables
- No declaration
- Examples:
  - ADD
  - MAX
  - AND
  - COS
  - GT (greater than)

Function Blocks

- Calculates a result depending on inputs and local variables
- Reusable
- Multiple Input
- Multiple Output
- Can store values within internal variables
- Declaration needed
- Examples:
  - TON (On delay timer)
  - TOF (Off delay timer)
  - TP (Pulse timer)
  - CTU (Counter up)
  - SR (Set/Reset)
  - R_TRIG (Rising trigger)



(Represented as FBD)



Generic Data Types

ANY

In parentheses: number of Bits

- ANY_BIT
  - BOOL (1), BYTE (8), WORD (16), DWORD (32), LWORD (64)
- ANY_NUM
  - ANY_INT
    - SINT (8), INT (16), DINT (32), LINT (64)
    - USINT (8), UINT (16), UDINT (32), ULINT (64)
  - ANY_REAL
    - REAL (32)
    - LREAL (64)
- TIME (*)
- ANY_DATE
  - DATE_AND_TIME (*)
  - DATE, TIME_OF_DAY (*)
- ANY_STRING
  - STRING (*)
  - WSTRING (*)

Abbreviations:
S...short (•0,5)
D...double (•2)
L...long (•4)
U...Unsigned

(*) required memory depends on the system

Variable Declaration

- Variables are declared at the beginning of each POU
- POU…Program organization unit: program, function block, function
- Syntax of Structured Text
- Internal variables

```
VAR
   AVE_SPEED: REAL;
   Inhibit  : INT  := 2;
END_VAR
```

- Input variables

```
VAR_INPUT
   SetPoint : REAL;
   MAX_Count: USINT := 150;
END_VAR
```

- Output variables

```
VAR_OUTPUT
   Message  : STRING(10);
   Status   : BOOL;
END_VAR
```

- Input/Output variables: interface to other POU's

```
VAR_IN_OUT
   counter  : INT;
END_VAR
```

- Global variables

```
VAR_GLOBAL
   Job_Num  : INT;
END_VAR
```

- Temporary variables

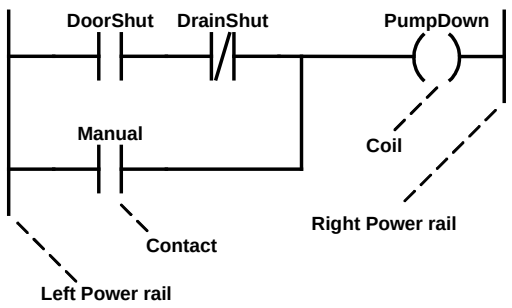
```
VAR_TEMP
   Result   : REAL;
END_VAR
```

- Directly represented variables:
  Adressing memory locations with %
  - First letter code: I…Input, Q…Output, M…Internal memory
  - Second letter code: X…Bit, B…Byte, W…Word, D…Double word, L…Long word

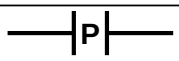
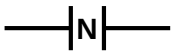
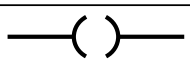
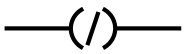
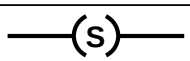
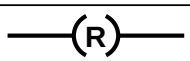
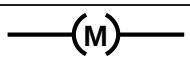
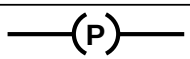
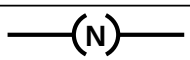
  Examples: `%IX2.6`, `%QW122`

Ladder Diagram (LD) – *Kontaktplan (KOP)*

- Historical background: Contactor circuits
- Application: Bit operations
- Structure: Networks
- Representation: Graphical
- Benefits:
  - Detection of signal edges
  - Clear expression for AND and OR operators
  - Easy to understand
- Disadvantages:
  - Bad support for operations except bit operations
  - Complex projects become confused

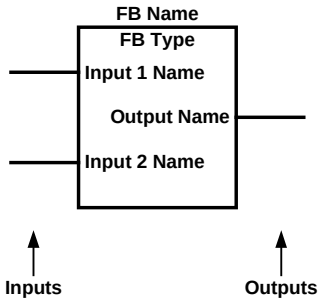


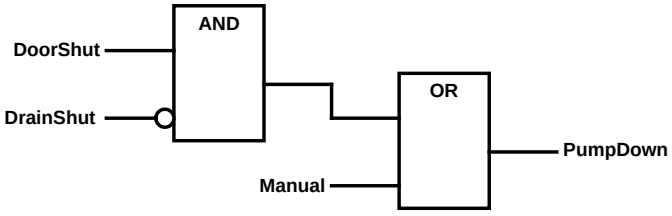
Ladder Diagram Features

Graphical feature	Graphic
Normally open contact	
Normally closed contact	
Positive transition-sensing contact	
Negative transition-sensing contact	
Coil	
Negated coil	
SET coil	
RESET coil	
Retentive memory coil (variable is stored retained in memory)	
SET retentive memory coil	
RESET retentive memory coil	
Positive transition-sensing coil	
Negative transition-sensing coil	

Function Block Diagram (FBD) – *Funktionsbausteinsprache (FBS)*

- Historical background: Electronic circuits
- Application: Logical connections
- Structure: Networks
- Representation: Graphical
- Benefits:
  - Good support for arithmetic operations
  - Easy use of functions and function blocks
  - Easy to understand
- Disadvantages:
  - Needs much space for documentation





Instruction List (IL) – *Anweisungsliste (AWL)*

- Historical background: Machine assembler language
- Application: Difficult problems with branches
- Structure: Rows
- Representation: Textual
- Benefits:
  - Easier to implement for PLC designer
  - Flexible
  - Suitable for performance optimized execution
- Disadvantages:
  - Hard to follow the program flow (especially when using jumps)

```
Label1: LD      DoorShut        (* If the door is shut *)
        ANDN    DrainShut       (* and the Drain switch is NOT shut *)
        OR      Manual          (* or the manual switch is pressed *)
        ST      PumpDown        (* then turn on the pump *)
```

Label **Operator** **Operand** **Comment**

Instruction List Operators (1)

Operator	Modifiers	Operand	Comments
LD	N	ANY	Load operand into result register
ST	N	ANY	Store result register into operand
S		BOOL	Set operand true
R		BOOL	Reset operand false
AND	N,(	ANY	Boolean AND
&	N,(	ANY	Boolean AND (equivalent to AND)
OR	N,(	ANY	Boolean OR
XOR	N,(	ANY	Boolean exclusive OR
NOT		ANY	Logical negation (one's complement)
ADD	(	ANY	Addition
SUB	(	ANY	Subtraction
MUL	(	ANY	Multiplication
DIV	(	ANY	Division
MOD	(	ANY	Modulo-division

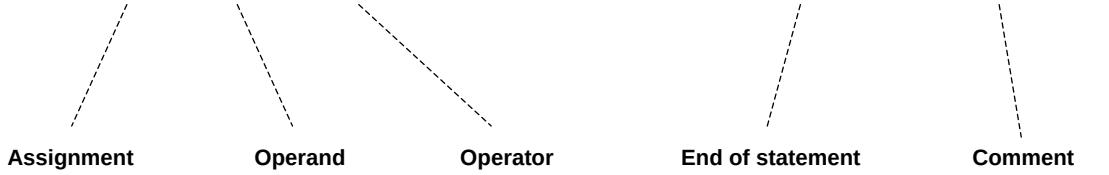
Instruction List Operators (2)

Operator	Modifiers	Operand	Comments
GT	(	ANY	Comparison greater than
GE	(	ANY	Comparison greater than, equal
EQ	(	ANY	Comparison equal
NE	(	ANY	Comparison not equal
LE	(	ANY	Comparison less than, equal
LT	(	ANY	Comparison less than
JMP	C,N	LABEL	Jump to label
CAL	C,N	NAME	Call function block
RET	C,N		Return from function or function block
)			Execute the last deferred operator

Structured Text (ST) – *Strukturierter Text (ST)*

- Historical background: High level programming languages (similar to Pascal)
- Application: Complex arithmetic calculations, loops
- Structure: Semicolon separated statements
- Representation: Textual
- Benefits:
  - Clear representation of loops
  - Compact
- Disadvantages:
  - No “free” jumps
  - No “SET” and “RESET”

```
PumpDown := DoorShut AND NOT DrainShut OR Manual;    (* Comment *)
```

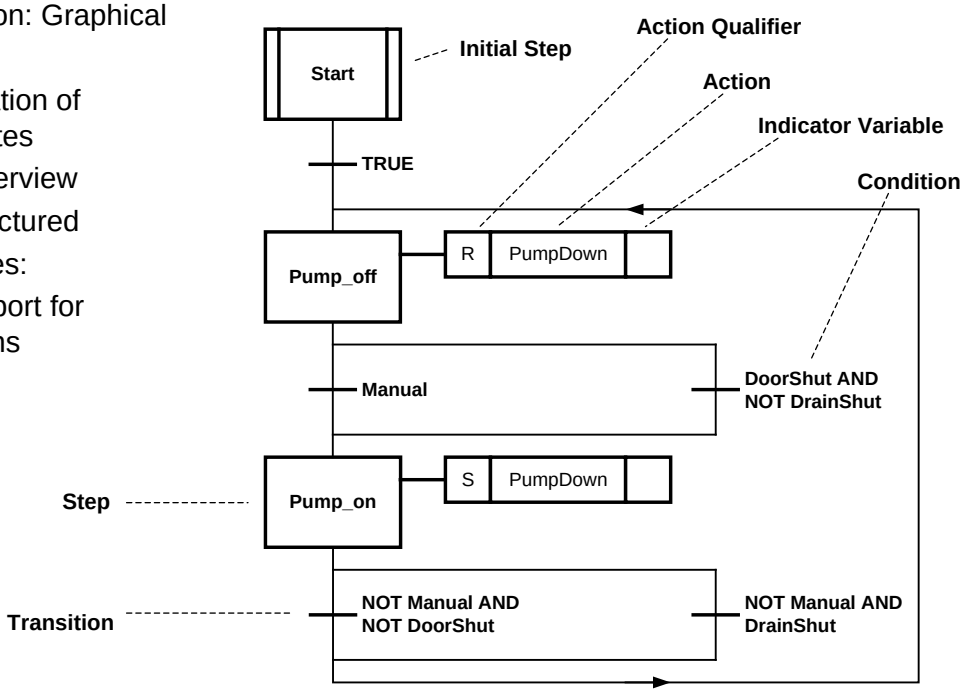


Structured Text Operators

Operator	Description	Precedence
(…)	Parenthesis expression	Highest
Function(…)	Parameter list of a function, function evaluation	
**	Exponentiation, i. e. raising to a power	
-	Negation	
NOT	Boolean complement, i. e. value with opposite sign	
*	Multiplication	
/	Division	
MOD	Modulus operation	
+	Addition	
-	Subtraction	
<, >, <=, >=	Comparison operators	
=	Equality	
<>	Inequality	
AND, &	Boolean AND	
XOR	Boolean exclusive OR	
OR	Boolean OR	Lowest

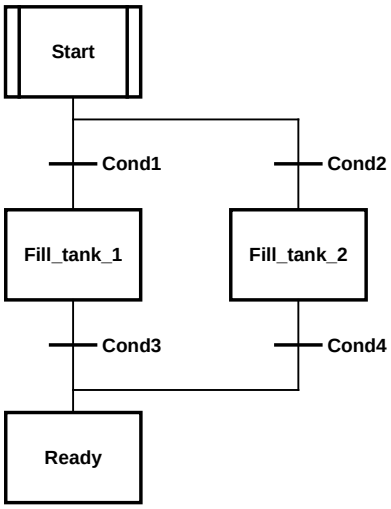
Sequential Function Chart (SFC) – *Ablaufsprache (AS)*

- Historical background: Automata, Petri nets, Grafcet
- Application: Sequential control, structuring of processes, not useful for low level
- Structure: Steps and Transitions
- Representation: Graphical
- Benefits:
  - Visualization of main states
  - Clear overview
  - Well structured
- Disadvantages:
  - Bad support for operations

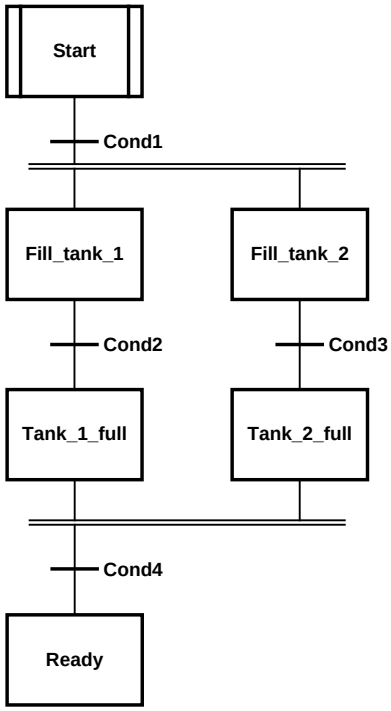


Sequential Function Chart Branching

Alternative branching
- Only one leg is executed



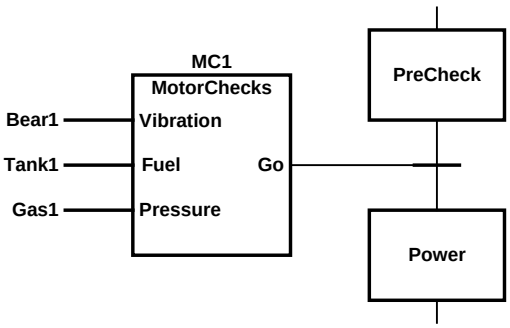
Simultaneous branching
- All parallel legs are executed



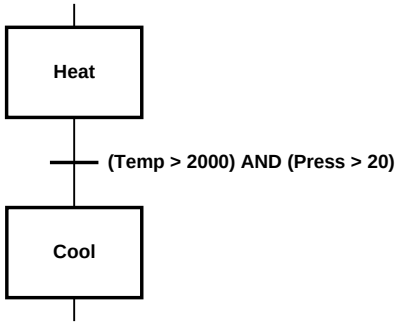
Transition Conditions for SFC

- Conditions are assigned to the Transitions by Structured Text, Function Block Diagram or Ladder Diagram
- Result of an expression must be a boolean value

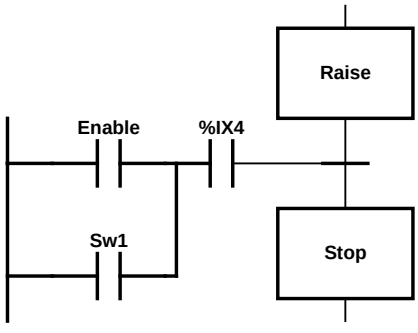
Function Block Diagram



Structured Text



Ladder Diagram

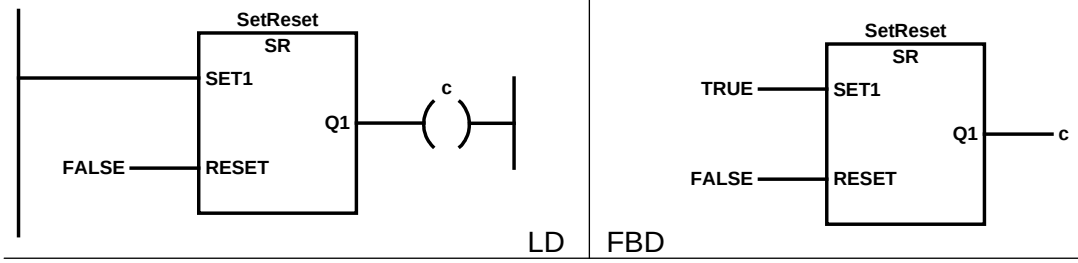


SFC Action Qualifiers

Qualifier	Description
None	Non-stored, default, same as ‘N’.
N	Non-stored, executes while associated step is active.
R	Resets a stored action.
S	Sets an action active, i. e. stored.
L *Time*	Time limited action, terminates after a given period.
D *Time*	Time delayed action, starts after a given period.
P	A pulse action that only executes once when a step is activated, and once when the step is deactivated.
P1	A pulse action that only executes once when a step is activated.
P0	A pulse action that only executes once when a step is deactivated.
SD *Time*	Stored and time delayed. The action is set active after a given period, even if the associated step is deactivated before the delay period.
DS *Time*	Action is time delayed and stored. If the associated step is deactivated before the delay period, the action is not stored.
SL *Time*	Stored and time limited. The action is started and executes for a given period.

Call Function Blocks and Programs

```
Declaration
VAR
   c           : BOOL;
   SetReset    : SR;
END_VAR
```



LD FBD

IL

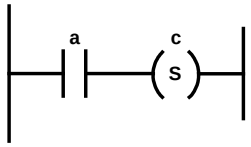
```
CAL SetReset(SET1:=TRUE, RESET:=FALSE)
LD  SetReset.Q1
ST  c
```

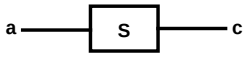
ST

```
SetReset(SET1:=TRUE, RESET:=FALSE);
c:=SetReset.Q1;
```

SET Operator

```
Declaration
VAR
    a    : BOOL := TRUE;
    c    : BOOL;
END_VAR
```





LD | FBD

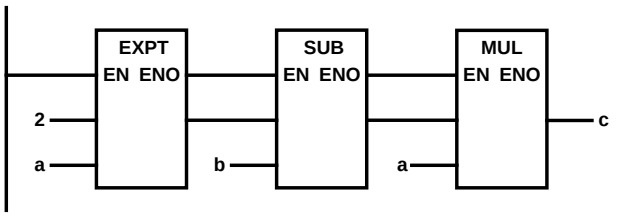
IL | ST

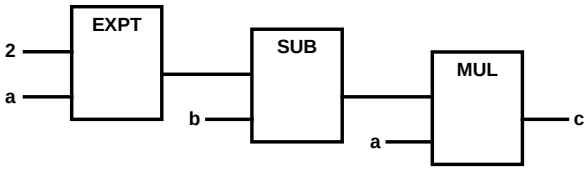
```
LD      a
S       c
```

```
IF a
THEN c:=TRUE;
END_IF;
```


Arithmetic Operators

```
Declaration
VAR
    a: INT := 4;
    b: INT := 22;
    c: REAL;
END_VAR
```





LD | FBD

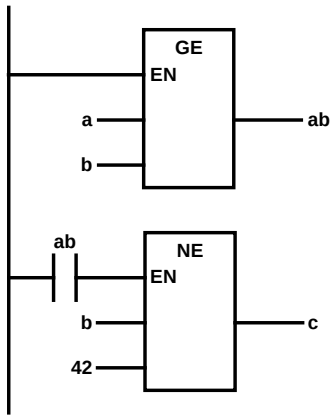
IL | ST

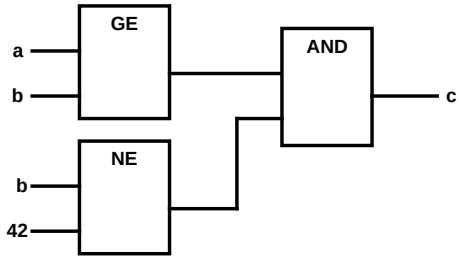
```
LD      2
EXPT    a
SUB     b
MUL     a
ST      c
```

```
c:=(2**a-b)*a;
```

Comparison Operators

```
Declaration
VAR
   a    : INT := 44;
   b    : INT := 42;
   ab   : INT; (*LD*)
   c    : BOOL;
END_VAR
```





LD | FBD

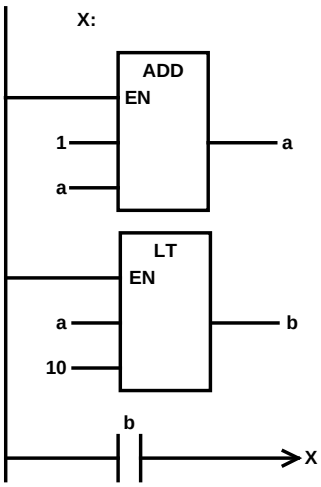
IL | ST

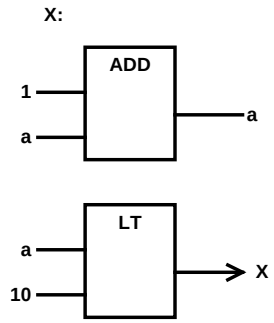
```
LD      a
GE      b
AND     (
LD      b
NE      42
)
ST      c
```

```
c:=(a>=b) AND (b<>42);
```


Jump Operator / Loop

```
Declaration
VAR
   a    : INT := 0;
   b    : INT;(*LD*)
END_VAR
```





LD | FBD

IL | ST

```
X:      LD      1
        ADD     a
        ST      a
        LT      10
        JMPC    X
```

```
REPEAT
  a:=a+1;
UNTIL
  a>=10
END_REPEAT
```