Call kommando

CALL (Statement)

Function

Call a program and execute it.

Syntax

CALL <programname> [(<argument>[,<argument>...])]

Description

This statement calls the program specified by <programname> and transfers control to that program.

An END statement in a called program is functionally equivalent to a RETURN statement in a subroutine so that execution of the END statement returns control to the caller program.

<argument> is passed to the program specified by <programname> either by "call by value" or "call by reference." The former uses a constant or expression; the latter passes a variable. For details, refer to 8.16 "Calling with a Value and with Reference".

Call by value

In calling by value, a constant, arithmetic expression or string expression is passed as a value.

Even a variable, if enclosed in parentheses ( ), is regarded as an expression and can be passed as a value. In the case of numbered variables, integer, floating-point, double-precision, and string variables can be passed if enclosed in parentheses ( ).

Example: Calling PROGRAM SUB1(AA#)

- To pass a variable as a value CALL SUB1((D1))
- To pass a constant CALL SUB1(10#)
- To pass an arithmetic expression CALL SUB1(D1+D2)

Example

```
DEFDBL ld1, ld2
CALL SUB1((ld1))          'Pass a variable as a value
CALL SUB1(10#)            'Pass a constant
CALL SUB1(ld1 + ld2)      'Pass an arithmetic expression
```

DO...LOOP

Gentag løkke:

11.3 Repeat

DO...LOOP (Statement)

Function

Repeat a block of statements while a condition is True or until a condition becomes True.

Syntax

DO [{WHILE|UNTIL}[<conditional expression>]]
:
LOOP
Or
DO
:
LOOP [{WHILE|UNTIL}[<conditional expression>]]

Description

DO WHILE and DO UNTIL are pretest loops.

LOOP WHILE and LOOP UNTIL are posttest loops.

A WHILE statement executes repeatedly while a condition is true (not 0), an UNTIL statement, until a condition becomes true.

In the expression specified by <conditional expression>, if the right-hand side is omitted, the system evaluates whether or not the value is true (not 0).

If <conditional expression> is omitted, the condition will be assumed as true (not 0). As a result, the WHILE statement falls into an infinite loop. An UNTIL never executes in a pretest loop and it executes only once in a posttest loop.

Although there is also a WHILE...WEND statement that is functionally equivalent to the DO WHILE...LOOP statement, using only DO WHILE...LOOP is recommended.

Although there is also a REPEAT...UNTIL statement that is functionally equivalent to the DO...LOOP UNTIL statement, using only DO...LOOP UNTIL is recommended.

WHILE and UNTIL can be omitted, but the loop becomes infinite.

Related Terms

EXIT DO, WHILE...WEND, REPEAT...UNTIL, FOR...NEXT

FOR...NEXT

Gentag løkke:

FOR...NEXT (Statement)

Function

Repeatedly execute a block of statements in a FOR...NEXT loop.

Syntax

FOR <variablename> = <initial value> TO <final value> [STEP <increment>]

:

NEXT [<variablename>]

Description

This statement repeatedly executes a block of statements in a FOR...NEXT loop according to the condition specified in the FOR line.

<initial value> and <final value> specify the initial and final values of the variable specified by <variablename>, respectively.

<increment> specifies the increment from the initial to the final values. If STEP is omitted, the increment is regarded as 1.

In the following cases, the FOR...NEXT is not executed and control is transferred to the position immediately following the NEXT.

- <increment> is positive and <initial value> is greater than <final value>
- <increment> is negative and <initial value> is smaller than <final value>

Note that <initial value> is assigned to <variablename>.

You can nest FOR...NEXT statements in one FOR...NEXT, which is referred to as a nested construction. In such a case, each FOR...NEXT must have different variables.

Additionally, a nested FOR...NEXT must be concluded within the upper-level FOR...NEXT.

ATTENTION

FOR and NEXT must be a pair.

Jumping into or out of a FOR...NEXT loop with GOTO does not assure the program operation. If the increment is set to 0, the loop becomes infinite.

Related Terms

DO...LOOP, EXIT FOR

GOTO kommando

Ubetinget hop

GOTO (Statement)

Function

Unconditionally branch a program.

Syntax

{GOTO|GO TO}<labelname>

Description

This statement unconditionally transfers control to a label specified by <labelname> and continues execution there.

GO TO can be used instead of GOTO.

Related Terms

GOSUB

Notes

Example

```
DIM li1 As Integer
IF li1 = 0 THEN          'If li1 is 0
  STOP                   'Stop program execution
ELSEIF li1 = 1 THEN      'If li1 is 1
  GOTO *samp1            'Jump to the label *samp1
  GO TO *samp2           'Jump to the label *samp2
ELSEIF li1 = 2 THEN      'If li1 is 2
  GOSUB *samp3           'Call the subroutine beginning with the label name *samp3
ELSE                     'If li1 is any other value
  RETURN                 'Return to the caller program
END IF                   'Declare the end of the IF statement
```

IF…END IF

Betinget hop:

IF...END IF (Statement)

Function

Conditionally execute specified statement blocks depending upon the evaluation of a conditional expression.

Syntax

IF <conditional expression> THEN

:

[ELSEIF <conditional expression> THEN]

:

[ELSE]

:

END IF

Description

This statement controls the execution of statement blocks depending upon the evaluation of <conditional expression>.

If <conditional expression> of the IF statement is true (not 0), the statement block following IF and preceding ELSEIF is executed; if false (0), <conditional expression> of the ELSEIF statement is evaluated. In this manner, subsequent statement blocks are also evaluated and executed depending upon the result.

Related Terms

IF...THEN...ELSE

Example

```
DIM li1 As Integer
IF li1 = 0 THEN          'If li1 is 0
   STOP                  'Stop program execution
ELSEIF li1 = 1 THEN      'If li1 is 1
  GOTO *samp1            'Jump to the label *samp1
  GO TO *samp2           'Jump to the label *samp2
ELSEIF li1 = 2 THEN      'If li1 is 2
  GOSUB *samp3           'Call the subroutine beginning with the label name *samp3
ELSE                     'If li1 is any other value
  RETURN                 'Return to the caller program
END IF                   'Declare the end of the IF statement
```

IF…THEN…ELSE

Betinget hop

IF...THEN...ELSE (Statement)

Function

Conditionally execute specified statement depending upon the evaluation of a conditional expression.

Syntax

IF <conditional expression> THEN {<statement>|<labelname>}

[ELSE {<statement>|<labelname>}]

Description

This statement controls the execution of specified <statement>s depending upon the evaluation of <conditional expression>.

If <conditional expression> is true (not 0), <statement> immediately following THEN is executed. If it is false (0), <statement> immediately following ELSE is executed.

Related Terms

IF...END IF

Example

```
IF i1 = 0 THEN STOP ELSE GOSUB *samp1
                         'If i1 is 0, stop program execution. If i1 is any other value,
                         'call the subroutine beginning with the label name *samp1
i1 = i1 + 1              'Add
END                      'Declare the end of program
*samp1:                  'Define the subroutine label
i0 =0                    'Assign 0 to i0
RETURN                   'Return to the caller program
```

Label kommando

Mærke:

8.3 Label

A label can be used to indicate a branch destination and the position of a statement in a program.
The following rules apply when using a label.

- A label name starts with an asterisk (*) .
- The second letter of a label name must be an arbitrary alphabet letter.
- Any combination of alphabet letters and numerals can be used for the third letter and the following letters in a label name.
- The last letter of a label name must be a colon (:) .
- A reserved word cannot be used as a label name.
- A label name to be referred to must be placed at the head of a line.
- An error occurs if the same label name to be referred to is duplicated.
- The range in which a label can be referred to is only in the program where the label is present.

```
PROGRAM WITH_LABEL
    IF IO138=1  THEN*ACTION:  ELSE  *NOACTION:
    *ACTION:        SPEED 100
                    MOVE P, P1
                    DELAY 200
                    MOVE P, P2
    *NOACTION:      DELAY 200
    END IF
END
```

Program Example Using Label

Motion Control

Robot Control Statements					
Motion Control	APPROACH	Execute the absolute movement designated in the tool coordinate system.	○	○	12-1
	DEPART	Executes the relative motion in the tool coordinate system.	○	○	12-4
	DRAW	Executes the relative movement designated in the work coordinate system.	⊙	⊙	12-7
	DRIVE	Executes the relative motion of each axis.	⊙	⊙	12-9
	DRIVEA	Executes the absolute motion of each axis.	⊙	⊙	12-11
	GOHOME	Moves to the position (home position) defined by the HOME statement.	⊙	⊙	12-13
	MOVE	Moves the robot flange to the specified coordinates. If specified with an EX option (relative motion of extended-joints) or EXA option (absolute motion of extended-joints), the MOVE can move both the robot flange and the extended-joints synchronously. (i.e. Synchronized start and stop from/at the specified positions is possible.).	○	○	12-14
	ROTATE	Executes a rotation movement around the designated axis.	○	○	12-19
	ROTATEH	Executes rotary motion by taking an approach vector as an axis.	⊙	⊙	12-22
	CURJNT	Obtains the current angle of the robot using type J.	○	○	12-24
	CURPOS	Obtains the current position in the tool coordinate system using type P.	○	○	12-25
	CURTRN	Obtains the current position in the tool coordinate system using type T.	⊙	⊙	12-26
	CUREXJ	Gets the current angle of an extended-joint into a floating-point variable.	V1.5	V1.5	12-27

Move kommando

MOVE (Statement)

Function

Moves the robot flange to the specified coordinates.

If specified with an EX option (relative motion of extended-joints) or EXA option (absolute motion of extended-joints), the MOVE can move both the robot flange and the extended-joints synchronously. (i.e. Synchronized start and stop from/at the specified positions is possible.).

Syntax

- Interpolation method: Except "Free curve" [Version 2.3 or later]
  MOVE <Interpolation method>,[@<Path start displacement>]<Pose>[,[@<Path start displacement>]<Pose>...][,<Motion option>][,NEXT]
- Interpolation method: "Free curve" [Version 2.3 or later]
  For a free curve MOVE S, [@<Pass start displacement>] <Trajectory number> [<EX or EXA option>] [, <Motion option>] [, NEXT]

Description

This statement moves the robot from the current position to the designated coordinate <Pose>.

For <Pose>, the position type (P type), joint type (J type) or homogeneous transformation type (T type) can be used.

For the position type, a variable, a pose array by numbered variables, a constant or the current position (*, CURPOS) can be used.

For the joint type, a variable, a pose array by numbered variables, or the current angle (CURJNT) can be used.

For the homogeneous transformation type, a variable or a pose array by numbered variables can be used.

Expression of the pose array: P[3 TO 6] From P[3] to P[6].

P, L, C or S can be selected for the For <Interpolation method>.

Interpolation method	Meaning
P (or PTP)	The robot moves from the current position to the designated coordinates using PTP control.
L	The robot moves from the current position to the designated coordinates using CP control.
C	The robot moves to a purpose pose performing arc interpolation via a relay point pose from the current position. The robot performs an interpolation motion from the figure of the current position to that of the purpose pose (the figure of the relay point pose is ignored). In arc interpolation, a relay point pose and a purpose pose must be designated. (A pose array cannot be used for C.) Even if pass start displacement is designated to a relay point pose, the motion does not change. Use MOVE C, P1, @P P2 to designate the pass taken when the arc interpolation motion is finished.
S (Free curve) [Version 2.3 or later]	Moves from the current position to the final viapoint through the registered viapoints by SETSPLINEPOINT. The path becomes a smooth curve and the tool moves at a constant speed on the path, except upon acceleration/deceleration. The pose passes each viapoint by pass movement.

Operatør panel

Chapter 8 Creating TP Easy Operation Panel Screen

8.1 How to Use Sample Programs for Creating Operation Panel Screen

Sample programs that help you create TP panels are stored in the Samples folder in the WINCAPSIII install disk. To import them, choose Project | Add Existing File... and select a desired sample program.

8.2 Creating TP Easy Operation Panel Screen

Describes the sample programs which are provided in WINCAPSIII.

Select_Screen (Sample program) [Version 1.7 or later]

Function

A use example of general-purpose operation screen switching processing

Description

This program controls screen switching in the general-purpose operation function.

This program monitors the I/O numbers of the screen switching buttons defined on each operation panel screen. When I/O is turned on, the corresponding operation panel screen is displayed.

This program is a sample; so, change it according to the actual processing.

Macro Definition

<button.h> is necessary.

Related Terms

Build_Lamp_Screen, Build_Man_Screen, Build_Plan_Screen

Reset commando

Deaktiver:

RESET (Statement)

Function

Sets an I/O port to OFF.

Syntax

RESET <I/O variable>

Description

Sets the port designated by <I/O variable> to OFF.

Related Terms

SET, DEFIO

Example

```
TAKEARM                  'Obtains robot control priority.
IOBLOK ON                'Moves to the coordinates (902.7,0,415.3,180,50,180,1)
                         '(PTP control)
MOVE P, (902.7,0,415.3,180,50,180,1)
                         'Moves (PTP control) to the coordinates
                         '(902.7,0,415.3,180,50,180,1).
SET IO[240]              'Sets the port 240 BIT type to ON.
SET IO[241],40           'Sets the port 241 BIT type to ON for 40 ms.
SET IO[SOL1]             'Sets the port designated by I/O variable SOL1 to ON.
SET IO[104 TO 110]       'Sets the port 104 to 110 BIT type to ON.
IF IO[242] THEN
  RESET IO[240]          'Sets the port 240 type to OFF.
  RESET IO[SOL1]         'Sets the port designated by I/O variable SOL1 to OFF.
  RESET IO[104 TO 110]   'Sets the port 104 to 110 BIT type to OFF.
ENDIF
IOBLOCK OFF
```

Set commando

Aktiver:

SET (Statement)

Function

Sets an I/O port to ON.

Syntax

SET <I/O variable>[,<Output time>]

Description

This statement sets the designated port in <I/O variable> to ON.

If <Output time> is designated a pulse is output. (The output time unit is ms.)

If <Output time> is designated the system does not proceed to the next instruction until this time elapses. The specified output time value is the minimum output time while the actual output time will change according to task priority.

Related Terms

RESET, DEFIO

Notes

- If output time is designated, it may be extended due to factors such as the presence of another program during movement, pendant operation, or communication with external devices.
- When output time is designated, it may possibly shift by 16.7ms since the standard clock for controller processing is ±16.7 ms.
- Note the following two points when using the time designation SET.
  - If you RESET the same port with another task while the port is ON due to the time designation SET, the port is set to OFF from the time of RESET (this means the time designation SET is valid).
  - If you keep resetting the same port and SET with another task while the port is ON due to the time designation SET, the designated port is set to OFF after the designated time elapses (time designation SET is valid).
- When output time is used, note that even during temporary stoppage the output time will elapse after an instantaneous stop during execution of the instruction and restart of the system.
- Specifying any number other than the ones for user outputs, hand outputs, and internal I/O ports causes the ERROR21FB (Reserved output area writing error). Be careful especially when using a mathematical expression for specifying a port number.
- Cannot be used for the I/O variable declared as a SINGLE type. When executed, error 777F (real number conversion failure) will occur.

Set kommando fortsat:

Example

```
TAKEARM                    'Obtains robot control priority.
IOBLOK ON                  'Concurrently executes an I/O instruction with the next
                           'motion instruction.
MOVE P, (902.7,0,415.3,180,50,180,1)
                           'Moves (PTP control) to the coordinates
                           '(902.7,0,415.3,180,50,180,1).
SET IO[240]                'Sets the port 240 BIT type to ON.
SET IO[241],40             'Sets the port 241 BIT type to ON for 40 ms.
SET IO[SOL1]               'Sets the port designated by I/O variable SOL1 to ON.
SET IO[104 TO 110]         'Sets the port 104 to 110 BIT type to ON.
IF IO[242] THEN
  RESET IO[240]            'Sets the port 240 type to OFF.
  RESET IO[SOL1]           'Sets the port designated by I/O variable SOL1 to OFF.
  RESET IO[104 TO 110]     'Sets the port 104 to 110 BIT type to OFF.
ENDIF
IOBLOCK OFF

TAKEARM                    'Obtains the robot control priority.
SET IO[I1 * 5]             'Set the port number obtained by I1 multiplied
                           'by 5 to ON.
SET IO[27-24]              'Set the port number obtained by subtracting 24
                           'from 27 (=3) to ON.
```

TIME$

TIME$ (System Variable)

Function

Obtains the current time.

Syntax

TIME$

Description

This statement stores the current time in the following format: "hh:mm:ss" (Hour: minute: second).
Time is displayed using the 24 hour system.

Related Terms

DATE$

Example

```
DIM ls1 As String
ls1 = TIME$              'Assigns the current time to ls1.
```

Wait kommando

19.1.2 WAIT

Suspend program execution according to a given conditional expression.

Syntax `WAIT_<conditional expression> [,<timeout>] [,<storage variable>]]`

Description The `WAIT` statement suspends program execution until `<conditional expression>` is satisfied.

If `WAIT` is not executed within the period specified by `<timeout>`, a timeout occurs and control passes to the next command. Using the timeout avoids an infinite stop. `<timeout>` is expressed in ms.

Specifying `<storage variable>` assigns TRUE (1) or FALSE (0) to the variable specified by `<storage variable>` when control passes out of the `WAIT` by the satisfied `<conditional expression>` or by timeout, respectively.

Example

```
DEFINT li1
WAIT li1 = 1            'Wait until expression li1 = 1 is satisfied.
WAIT IO[10] = ON        'Wait until IO10 is turned ON.
WAIT IO[5] = 0, 2000    'Wait until IO5 is turned OFF. If IO5 is not turned ON
                        'within 2 seconds, pass control to the next statement.
WAIT I3 = 5, 1000, I4   'Wait until expression I3 = 5 is satisfied.
                        'If the expression is satisfied, set I4 to 1.
                        'If the expression is not satisfied within one second,
                        'set I4 to 0 and pass control to the next statement.
```

Wait kommando

WAIT (Statement)

Function

Stops program processing based on a condition.

Syntax

WAIT <Conditional expression> [,<Timeout time> [,<Storage variable>]]

Description

This statement stops program processing until <Conditional expression> is satisfied.

If <Timeout time> is set, control stops the execution of a WAIT statement after the designated time elapses and proceeds to the next command. Infinite stoppage can be avoided by using this.

<Timeout time> is expressed in ms.

The reevaluation interval for monitoring <Conditional expression> or <Timeout time> depends on the priority of the task.

In Version 1.8 or later, when <Storage variable> is set, the WAIT command will assign TRUE (1) or FALSE (0) to the designated variable if control passes out of the WAIT statement by the satisfied <Conditional expression> or by timeout, respectively.

Related Terms

DELAY

Notes

When using timeout time, if an instantaneous stop during execution of an instruction is executed and the system is restarted, the designated time will elapse even during suspension.

Example

```
DEFINT li1, li2, li3, li4, li5
WAIT li1 = 1              'Waits until li1 = 1 is satisfied.
WAIT li2 = 0, 2000        'Waits until li2 = 0 is satisfied.
                          'Even if it is not satisfied
                          'after 2 seconds, the system
                          'proceeds to the next statement.
WAIT li3 = li4, li5       'Waits until li3 = li4 is satisfied.
                          'Even if it is not
                          'satisfied after the time of li5,
                          'the system proceeds to the next statement.
WAIT IO[10] = ON          'Waits until the 10th IO comes ON.
```

[Version 1.8 or later]

```
WAIT li3 = li4, li5, li6 'Wait until li3 = li4. If the conditional expression
                         'is not satisfied within li5 period, pass control to
                         'the next statement and assign FALSE to li6.
                         'If satisfied within li5 period, pass control to the
                         'next statement and assign TRUE to li6.
```