IJC-01 PLC
Instruction Sets

User's Manual

MIRLE Automation Corporation

Version 1.3(11/16/2016)

Table of Contents

FOREWORD

IJC series PLCs bring the high performance and application flexibility. Ten contacts and abundant function blocks (also referred to as instructions) are provided for application control programs using the IJC series PLC. In this manual, the usage for contacts and function blocks is described together with application examples.

Contact elements include:

(1) ┤ ├ (A normally open contact, usually referred to as: “A contact”)
(2) ┤/├ (A normally closed contact, usually referred to as: “B contact”)
(3) **-( )-** (A normal coil)
(4) **-(S)-** (A set coil)
(5) **-(R)-** (A reset coil)
(6) **-(↑)-** (A positive transitional coil)
(7) **-(↓)-** (A negative transitional coil)
(8) **-(M)-** (A holding coil during power loss)
(9) **-(SM)-** (A holding set coil during power loss)
(10) **-(RM)-** (A holding reset coil during power loss)

Function blocks instructions include:

(1) Timers and counters:
Timers: **T1.0, T0.1, T0.01,**
Counters: **UCTR, DCTR.**

(2) Mathematical blocks:
Adders: **ADD, ADDB, ADDL, ADBL, FADD,**
Subtracts**: SUB, SUBB, SUBL, SBBL,**
Multipliers: **MUL, MULB, MULM, MUBM, MULL, MLBL,**
Dividers: **DIV, DIVB, DIVM, DVBM, DIVL, DVBL,**
Square root: **ISQR.**

(3) Register, Table, Array instructions:
Move: **R->T, T->R, BLKM, PACK,**
Rotate/Shift: **T_RS, BROT, ODSR,**
Modify: **MBIT,**
Compare**: T_CM, CMPR,**
Logic: **AND, OR, XOR, COMP**
Stack: **PUSH, POP,**
Sense: **SENS,**

Encoder, Decoder: **ENCO, DECO,**
Convert: **B->C, C->B, EI->F, EF->I.**

(4) Flow control instructions:
Main program: **EOP, SKIP, MCS, MSE, JMP, EOJ, MSCJ, MSEJ**
Subroutine: JSR, SBR, RET, **CALL, LSL, RTS, CALLJ, LSLJ, RTSJ**
Loop: **FOR, NEXT,**
Pointer: **INIP, INCP, DECP, PADD, PSUB.**

(5) System related instructions:

(6) Others:
MOVE

Users are advised to become familiar with the binary operation (which can be found in any Digital Design Textbook) and the characteristics for each contact element and function block before designing a control application program. Please also be advised that the data and illustrations in this manual are not binding. We reserve the right to modify our products in line with our policy of continuous product improvement. Information in this manual is subject to change without notice and should not be treated as a commitment by MIRLE Automation Corp. MIRLE assumes no responsibility for any errors that may appear in this manual.

CHAPTER 1: INTRODUCTION

The basic concept required to use this manual and the elements (contacts , function blocks, and instructions) in IJC PLC is briefly described in this Chapter. In Section 1, the terminology and numerical representation are described. The constituents of a function block are described in Section 2 and the convention used to represent the function blocks is described in Section 3.

SECTION 1: Terminology and Numerical Representations:

BIT:

The basic unit is the binary system. The value of a bit is either 0 or 1. The abbreviation for bit is B, such as B0, B1 … etc.

NIBBLE:

A nibble is composed of four bits such as B3~B0. It can be used to represent decimal values ranging from 0 to 9, or hexadecimal values ranging from 0~F. The abbreviation for nibble is NB, such as NB0, NB1 … etc.

BYTE:

A byte is composed of eight bits (B7~B0) or two contiguous nibbles (NB1~NB0). It can be used to represent hexadecimal values ranging from 00~FF. The abbreviation for byte is BY, such as BY0, BY1 … etc.

WORD:

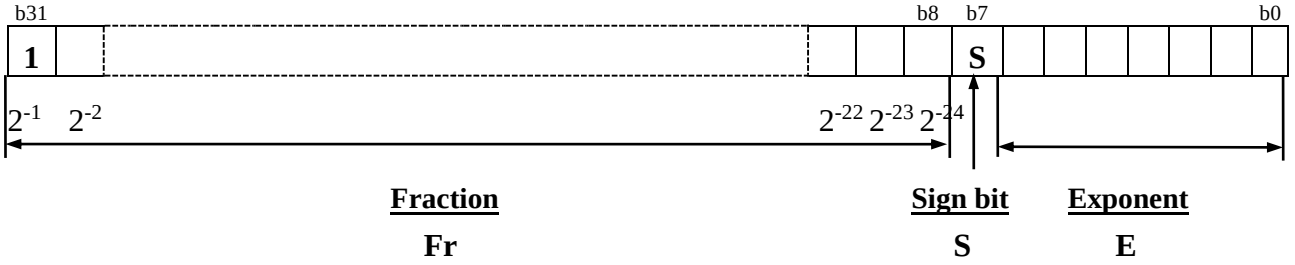
A word is composed of sixteen bits. It can be used to represent hexadecimal values ranging from 0000~FFFF or 0~65535 in the decimal system. The abbreviation for word is W, such as W0, W1 … etc. Since IJC PLC is based on 16-bit microcomputer architecture, a word occupies one register in the computer memory.

LONG WORD:

A long word is composed of two continuous words or 32 bits. It can be used to represent hexadecimal values ranging from 00000000~FFFFFFFF, floating point numbers through special convention, or decimal format ranging from 0~99999999. The abbreviation for long word is LW, such as LW0, LW1 … etc. A long word occupies two continuous registers in the computer memory. The first register contains the most significant 16 bits (usually referred to as HIGH WORD), the second register contains the least significant 16 bits (usually referred to as LOW WORD). A long word is referenced by the address occupied by the High Word.

Floating Point Representation using a Long Word:

A long word (32 bits) can be used to represent a floating point number. The bit assignment is shown in the following figure:



Formula: $\mathbf{I = (-1)^{S} \times 2^{(E-64)} \times Fr}$

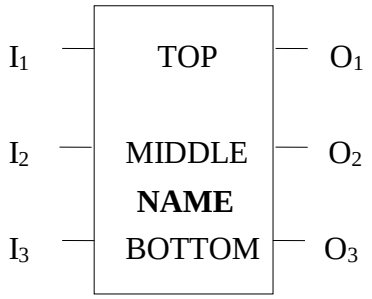
For example, assuming that the content of register 40130 is C000h and register 40131 is 0042h; then for an operation using floating point referencing register 40130 (40130 and 40131 actually), the value used is:

40130	40131
1100 0000 0000 0000	0000 0000 0100 0010

$\mathbf{I = (-1)^{0} \times 2^{(66-64)} \times (2^{-1}+2^{-2}) = 3}$

SECTION 2: Constituents of a Function Block

In IJC series PLC, a function block is composed of four parts: Function Name, Input Control, Operand and Function Output as shown in the following figure:



Where:
1. I_1, I_2, I_3 are Input controls
2. O_1, O_2, O_3 are Function outputs.
3. TOP, MIDDLE, BOTTOM stand for Top node, middle node and bottom node. These three nodes are operands.
4. NAME is the name of the function block.

Function Name:

The function name is an abbreviation or acronym of the operation performed by the function block. Two to four characters are used to represent the function. A complete list of the function block names may be found in the FOREWORD of this manual.

Input Control:

There must be one input control for each function block. This input control (usually referred to as I_1) is used to determine whether to execute this function block or not. For some function blocks, there are two additional input controls (I_2 and I_3). They are used to determine the execution mode of the function block.

Function Output:

There must be a function output control for each function block. This output (usually referred to as O_1) is used to drive a coil or used as an input control for the next function block. For some function blocks, there are two additional output controls (O_2 and O_3), they are also used to represent the results of the execution.

Operands:

Operands, as the name implies, are the objects of operations. An operand whose content is not altered by the operation is called a SOURCE. An operand that is used to store the result of the operation is called a DESTINATION. Operands can be Input contact, Output coil or register in memory. For IJC PLC, the designations of operands are listed in the following table:

Table 1.1: Operands

Initial	NAME	DESCRIPTION
0	Output Coil (Discrete output)	Use Output coil as an operand. Since 1 word = 16 bits, thus the number assignment of the operand must be a multiple of 16 plus 1. For example: 00001, 00017, 00033.
1	Input contact (Discrete input)	Use Input contact as an operand. The number assignment of the operand must be a multiple of 16 plus 1. For example: 10001, 10017, 10033.
3	Input register	Use Input register as an operand. For example: 30001, 30003.
4	Holding register	Use Holding register as an operand. For example: 40001, 40003.
C	Constant	For some function blocks, a constant can be defined as an operand: and during control program execution, the value of the constant is readily available rather then fetching from register memory. For example: #00001, #0020h. The former is a decimal constant, and the latter is a hexadecimal constant.
P	Pointer	For some function blocks, a pointer can be defined as an operand, and this pointer can be used for indirect addressing pointing to 0-, 1-, 3-, 4-type variable. For example: P0001
L	Label	For paired instructions (such as FOR and NEXT), their operands are label, and the label for each instruction must be the same in order for program to be executed correctly. For example: L0001.

Currently, there are three models of IJC controllers: MJ9000, MX1 series. The memory size and the CPU capability are different between models to meet different control requirements. Therefore, the numbers of spaces available for operands are also different. The available ranges for operands for each model are listed in the following table.

Table 1.2: Available operand ranges for IJC series controller

OPERAND	**IJC01**
0	00001~07168
1	10001~11024
3	30001~35873
4	40001~49999
L	L1~L32
C	0~65535
P	P0~P15

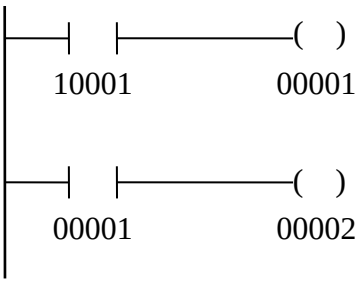
CHAPTER 2: CONTACTS

Contact elements are the most fundamental elements in Ladder Programs. Familiarization with their characteristics and usage is highly recommended.

(1) ┤├ Normally Open Contact:

This type of contact is usually referred to as "**A Contact**". When a contact is energized, the said "A contact" becomes conductive; and vice versa.

【EXAMPLE】



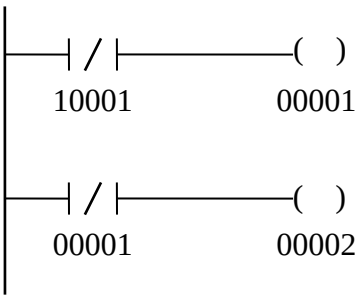
【Meaning】

When input contact 10001 is 'ON', coil 00001 is energized, and "A contact" 00001 becomes conductive, thus, coil 00002 is energized.

(2) ┤/├ Normally Closed Contact:

This type of contact is usually referred to as "**B Contact"**. When a contact is not energized, the said "B contact" becomes conductive; and vice versa.

【EXAMPLE】



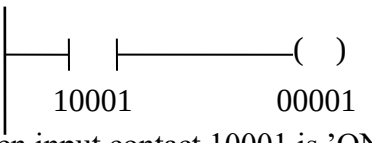
【Meaning】

When input contact 10001 is 'OFF', coil 00001 is energized, and "B contact" 00001 becomes non-conductive, thus, coil 00002 is not energized.

(3) -( )- Output Coil:

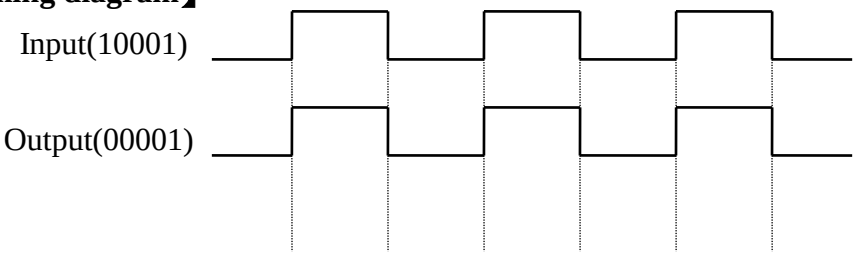
This output coil reflects the state of the elements connected to it. If the element is in the 'ON' state, then this coil is said to be energized; and vice versa.

【EXAMPLE】



【Meaning】 When input contact 10001 is 'ON', then output coil 00001 is 'ON'; When input contact 10001 is 'OFF', then output coil 00001 is 'OFF'

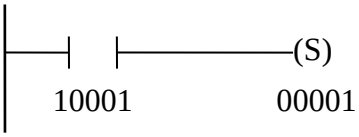
【Timing diagram】



(4) -(S)- Set Coil :

When the element connected to this coil is 'ON', then this set coil is set to 'ON' and remains in that "ON' state until the "RESET coil" with the same reference number is energized.

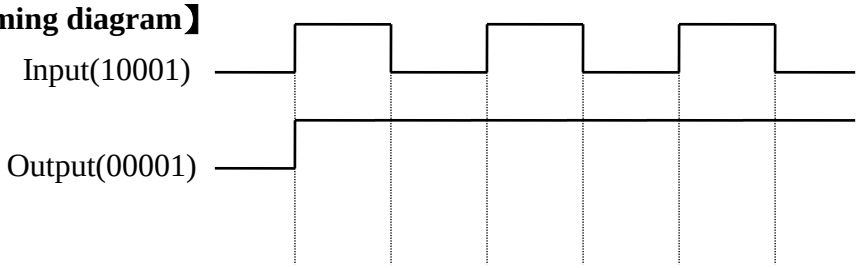
【EXAMPLE】



【Meaning】

When contact 10001 is 'ON', the set coil 00001 is 'ON' and remains 'ON' no matter how contact 10001 is changed.

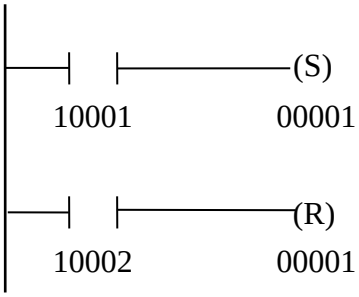
【Timing diagram】



(5) -(R)- Reset coil:

When the element connected to this coil is 'ON', then this set coil is set to 'OFF' and remains in that "OFF' state until the "SET coil" with the same reference number is energized.

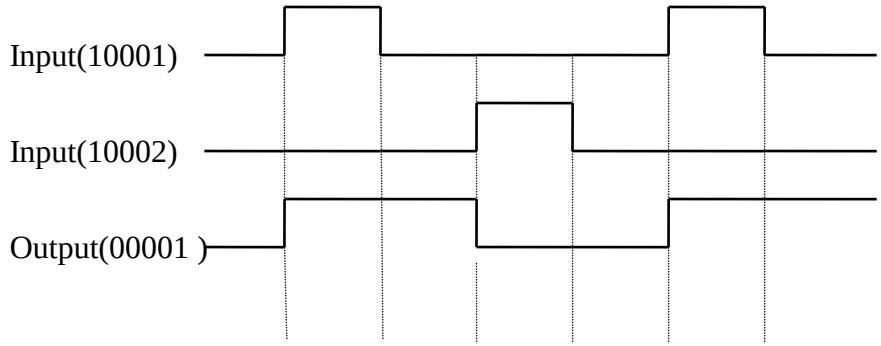
【EXAMPLE】



【Meaning】

When input contact 10001 is 'ON', output coil 00001is set to 'ON' and remains in that state. Until input contact 10001 is 'OFF' and input contact 10002 is 'ON', then output coil 00001is set to 'OFF' and remains 'OFF'.

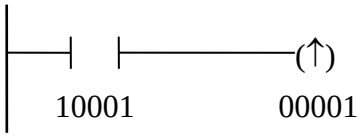
【Timing diagram】



(6) -(↑)- Positive Transitional Pulse Output Coil:

When the element connected to this output has an 'OFF'⇒'ON' transition, a pulse('OFF'⇒'ON') is generated for this output.

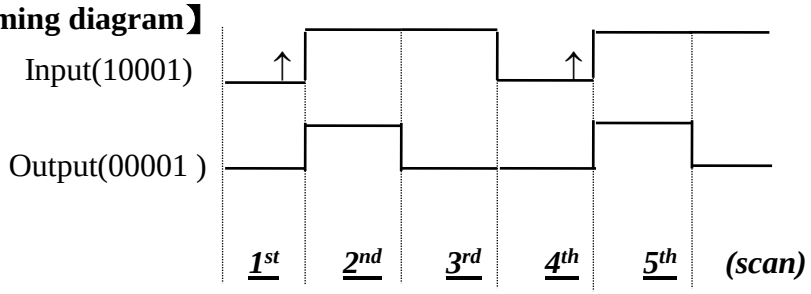
【EXAMPLE】



【Meaning】

When input contact 1000 receives a transition 'OFF'⇒'ON', then a pulse 'OFF'⇒'ON' is generated for output coil 00001. The width of the pulse is 1 scan time.

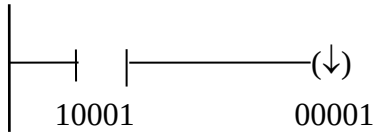
【Timing diagram】



(7) -(↓)- Negative Transitional Pulse Output Coil:

When the element connected to this output has an 'ON'⇒'OFF' transition, a pulse('OFF⇒'ON') is generated for this output.

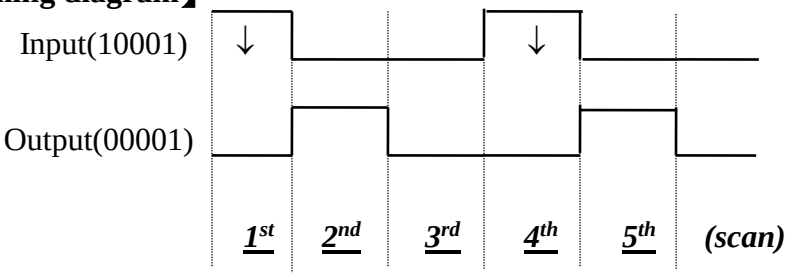
【EXAMPLE】



【Meaning】

When input contact 10001 receives a transition 'ON'⇒'OFF', then a pulse 'OFF'⇒'ON' is generated for output coil 00001. The width of the pulse is 1 scan time.

【**Timing diagram**】



(8) -(M)- Holding Coil during power loss:

This output coil reflects the state of the elements connected to it. If the element is in the 'ON' state, then this coil is said to be energized; and vice versa. The last state of the coil is maintained after system power is shut down and turned on again.

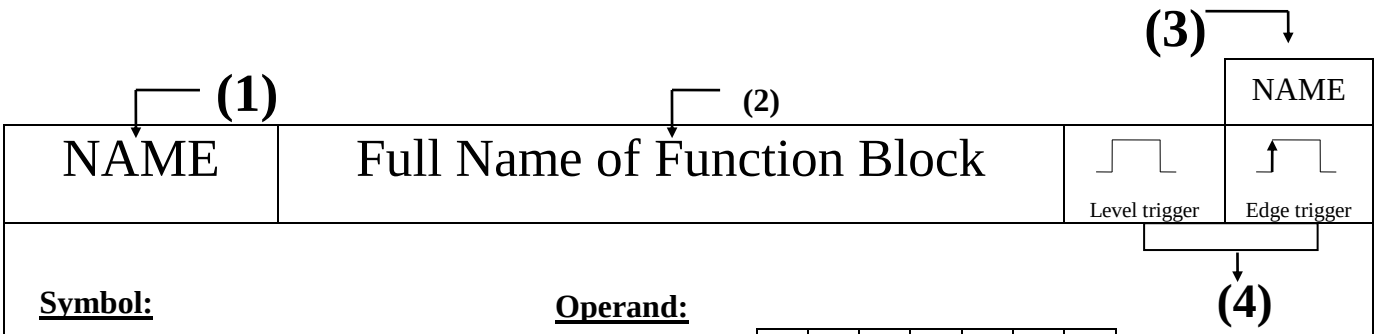
(9) -(SM)- Holding Set Coil during power loss:

When the element connected to this coil is 'ON', then this coil is set to 'ON' and remains in that "ON' state until the "RESET coil" with the same reference number is energized. The last state of the coil is maintained after system power is shut down and turned on again.

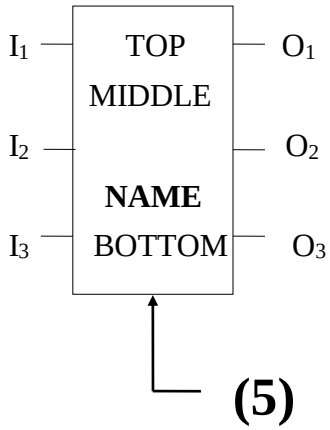
(10) -(RM)- Holding Reset Coil during power loss:

When the element connected to this coil is 'ON', then this coil is set to 'OFF' and remains in that "OFF' state until the "SET coil" with the same reference number is energized. The last state of the coil is maintained after system power is shut down and turned on again.

CHAPTER 3: FUNCTION BLOCKS



Symbol:



Operand:

	0	1	3	4	C	P	L
TOP NODE	○	○	○	○	①	○	
MIDDLE NODE							
BOTTOM NODE	○	○	○	○		○	

①0~65535 (6)

Description : (7)

Node description:

TOP： (8)

MIDDLE：

BOTTOM：

Input Control:

I_1：

I_2： (9)

I_3：

Function Output:

O_1：

O_2： (10)

O_3：

The template for the description of a function block is divided into ten areas (א~ב). The meaning for each area is described as follows:

(1) NAME:

NAME is an abbreviation or acronym for the operation performed by the function block. Two to four characters are used to represent the function. When displaying the ladder program on screen, the name of the function block is also displayed.

(2) Full Name of Function Block:

The operation of the function block is given briefly in this area.

(3) NAME:

This area is provided for easy reference to function blocks.

(4)Trigger mode:

The entry here is used to indicate the trigger mode of the function block. For "Level trigger" mode, when I1 is HIGH, then the function block is executed. For "Edge trigger" mode, when there is an OFF to ON transition, then the function block is executed. For edge-trigger function blocks, a "^" mark is prefixed to the name of the function block in the PP programming environment.

(5) Symbol:

The symbol of the function block as used in this manual is displayed in the ladder diagram.

(6) Operands

Operands available for the function block have a circle "○" marked in the table.

(7) Function blocks description:

A brief description of the major function of the function block together with its input control, function output and result of the execution is given in this area.

(8) Node description:

The usage of each node, whether it is a Source or a Destination, is given in this area.

(9) Input Control:

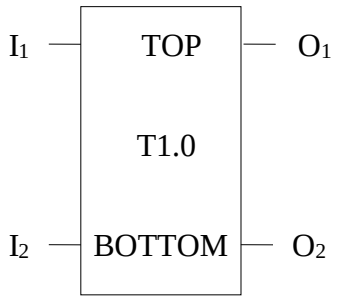
The condition (I_1) required for the function block to be executed is described here. The execution mode (I_2 and/or I_3) is also described here.

(10) Function Output:

The results of the execution (O_1, O_2, O_3) are given in this area.

T1.0	**1.0 SECOND TIMER**		

SYMBOL:



OPERANDS:

	0	1	3	4	C	P	L
TOP			○	○	①		
BOTTOM				○			

①0~65535

Description:

Timer increments by one at intervals of one second. When the accumulated time (stored in the BOTTOM node) reaches the timer preset (stored in TOP node), the timer stops. Input control can be used to start, stop and reset the timer. The timer status (whether the elapsed time has reached the preset time) can be detected by examining the function output.

Node description:

TOP: Preset value for timer.

BOTTOM: Accumulated value since timer started.

Input Control:

I_1：Execution control. When $I_1 = 1$, timer starts; $I_1 = 0$, timer stops.

I_2：Reset control, when $I_2 = 1$, the accumulated value is cleared to zero.

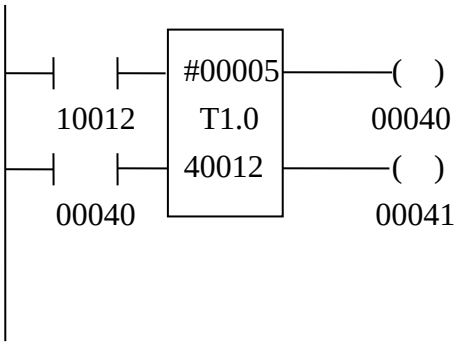
Function Output:

O_1 =1, if accumulated value ≥ preset value.

=0, if accumulated value ＜ preset value.

O_2: Complement of O_1

【EXAMPLE】



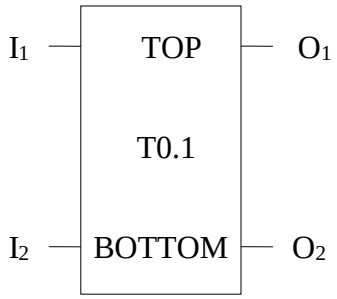
【DESCRIPTION】

This example shows a five-second timer. The decomposition of actions is:

1. 40012 is 0, then 00040 = ’OFF’ and 00041 = ’ON’ at the beginning.
2. When input control 10012 is ‘ON’, register 40012 increases by one for every one second.
3. When the content of register 40012 = 5 (as defined in the top node), the function output: 00040 = ‘ON’, 00041 = ‘OFF’.
4. Since 00040 = ‘ON’, I_2 changes to ’ON’, and clears register 40012 to ’0’.
5. Since 40012 = 0, then 00040 = ’OFF’, 00041 = ‘ON’, register 40012 continues incrementing, and the execution continues from STEP 3.

T0.1	**0.1 SECOND TIMER**		

SYMBOL:



OPERANDS:

	0	1	3	4	C	P	L
TOP			○	○	①		
BOTTOM				○			

①0~65535

Description:

Timer increments by one at intervals of 0.1 second. When the accumulated time (stored in the BOTTOM node) reaches the timer preset (stored in TOP node), the timer stops. Input control can be used to start, stop and reset the timer. The timer status (whether the elapsed time has reached the preset time) can be detected by examining the function output.

Node description:

TOP: Preset value for timer.

BOTTOM: Accumulated value since timer started.

Input Control:

I_1： Execution control. When $I_1 = 1$, timer starts; $I_1 = 0$, timer stops.

I_2：Reset control, when $I_2 = 1$, the accumulated value is cleared to zero.

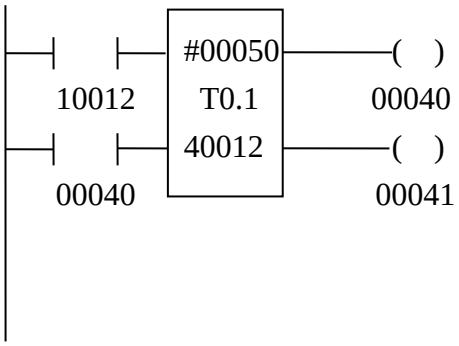
Function Output:

O_1 =1, if accumulated value ≥ preset value.

=0, if accumulated value ＜ preset value.

O_2: Complement of O_1

【EXAMPLE】



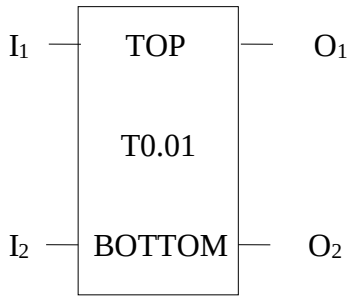
【DESCRIPTION】

This example shows a five-second timer. The decomposition of actions are:

1. 40012 is 0, then 00040=’OFF’ and 00041=’ON’ at the beginning.
2. When input control 10012 is ‘ON’, register 40012 increases by one for every one second.
3. When the content of register 40012 = 50 (as defined in the top node), the function output: 00040 = ‘ON’, 00041 = ‘OFF’.
4. Since 00040 = ‘ON’, I_2 changes to ’ON’, and clears register 40012 to ’0’.
5. Since 40012 = 0, then 00040 = ’OFF’, 00041 = ‘ON’, register 40012 continues incrementing, and the execution continues from STEP 3.

T0.01	**0.01 SECOND TIMER**		

SYMBOL:



OPERANDS:

	0	1	3	4	C	P	L
TOP			○	○	①		
BOTTOM				○			

①0~65535

Description:

Timer increments by one at intervals of 0.01 second. When the accumulated time (stored in the BOTTOM node) reaches the timer preset (stored in TOP node), the timer stops. Input control can be used to start, stop and reset the timer. The timer status (whether the elapsed time has reached the preset time) can be detected by examining the function output.

Node description:

TOP: Preset value for timer.

BOTTOM: Accumulated value since timer started.

Input Control:

I_1： Execution control. When $I_1 = 1$, timer starts; $I_1 = 0$, timer stops.

I_2：Reset control, when $I_2 = 1$, the accumulated value is cleared to zero.

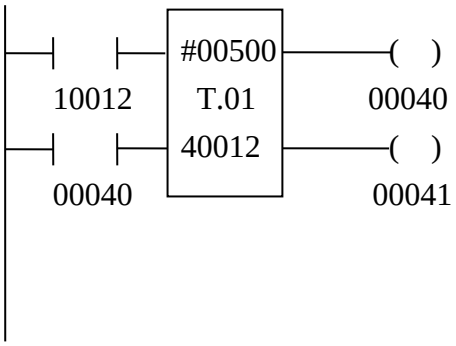
Function Output:

O_1 =1, if accumulated value ≥ preset value.

=0, if accumulated value ＜ preset value.

O_2: Complement of O_1

【EXAMPLE】



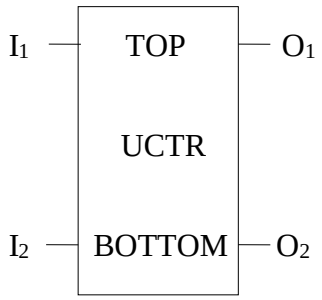
【DESCRIPTION】

This example shows a five-second timer. The decomposition of actions are:

1. 40012 is 0, then 00040 = ’OFF’ and 00041 = ’ON’ at the beginning.
2. When input control 10012 is ‘ON’, register 40012 increases by one for every 0.01 second.
3. When the content of register 40012 = 500 (as defined in the top node), the function output: 00040 = ‘ON’, 00041 = ‘OFF’.
4. Since 00040 = ‘ON’, I_2 changes to ’ON’, and clears register 40012 to ’0’.
5. Since 40012 = 0, then 00040 = ’OFF’, register 40012 continues incrementing, and the execution continues from STEP 3.

UCTR	**UP COUNTER**		

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP			○	○	①		
BOTTOM				○			

①0~65535

Description:

This counter counts the pulses presented at I_1 from 0 to a preset value. Input control can be used to start, stop and reset the counter. The counter status (whether the accumulated value has reached the preset value) can be detected by examining the function output.

Node description:

TOP: Preset value for counter.

BOTTOM: Accumulated value since counter started.

Input Control:

I_1：Counter control. When I_1 receives an 'OFF'⇒'ON' transition, The counter is increased by 1.

I_2：Reset control. When I_2 = 1, the accumulated value is cleared to zero.

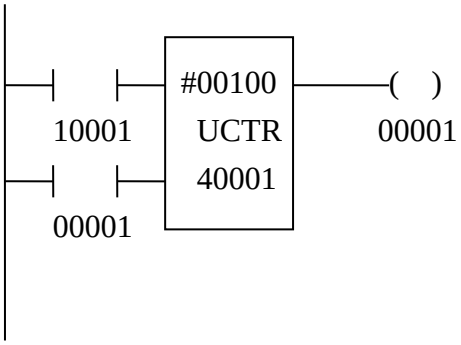
Function Output:

O_1 =1, if accumulated value ≥ preset value.

=0, if accumulated value < preset value.

O_2: Complement of O_1

【EXAMPLE】

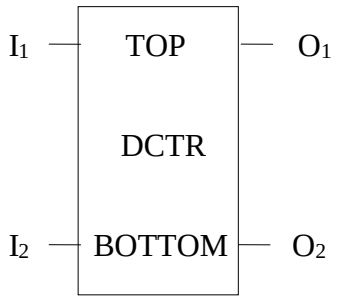


【DESCRIPTION】

When contact 10001 receives an OFF to ON transition, the accumulated value of the counter (40001) is incremented by 1. When the accumulated value reaches 100, coil 00001 is energized. When normal-open contact 00001 opens, the counter is reset.

DCTR	**DOWN COUNTER**		

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP			○	○	①		
BOTTOM				○			

①0~65535

Description:

This counter counts the pulses presented at I_1 from a preset value to 0. Input control can be used to start, stop and reset the counter. The counter status (whether the accumulated value has reached 0) can be detected by examining the function output.

Node description:

TOP: Preset value for counter.

BOTTOM: Accumulated value since counter started.

Input Control:

I_1：Counter control. When I_1 receives an 'OFF'⇒'ON' transition, the counter is decreased by 1.

I_2：Reset control. When I_2 = 1, the accumulated value is set to preset value.

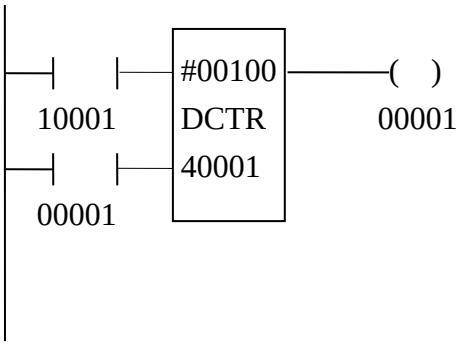
Function Output:

O_1 =1, if accumulated value=0.

=0, if accumulated value > 0.

O_2: Complement of O_1

【EXAMPLE】

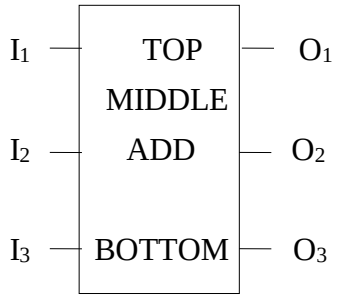


【DESCRIPTION】

When contact 10001 receives an OFF to ON transition, the accumulated value of the counter (40001) is decreased by 1. When the accumulated value reaches 0, coil 00001 is energized. When normal-open contact 00001 opens, the counter is reset and the value in counter (40001) is set to 100.

ADD	**FOUR DIGIT DECIMAL ADDER**	⎍	⎍↑

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP			○	○	①	○	
MIDDLE			○	○	①	○	
BOTTOM				○		○	

①0~9999

word＋word→word (Decimal)

Description:

The decimal values stored in the top and middle nodes are added and the sum is stored in the bottom node. Sum = (top + middle + I_3) MOD 10000.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Function output (O_3) may be used to determine whether or not an overflow has occurred.

Node Description:

TOP: Summand must be < 10000.

MIDDLE: Addend must be < 10000.

BOTTOM:1.(top + middle + I_3) MOD 10000

2. If error (ref. to O_2) occurred, the content of the bottom node remains unchanged.

Input Control:

I_1: When ⎍ (⎍↑) is presented, the function block is executed.

I_2 : error in

I_3 : carry in

Function Output:

$O_1 = I_1$

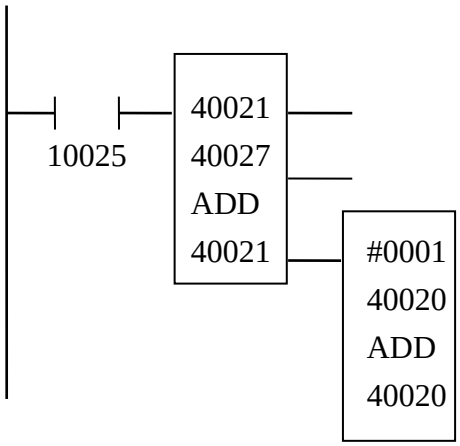
O_2 = error output (O_2 is '1' if I_2 is '1' or the value of either top node or middle node is over 9999)

O_3 ： overflow/carry

= 1, Sum ＞9999

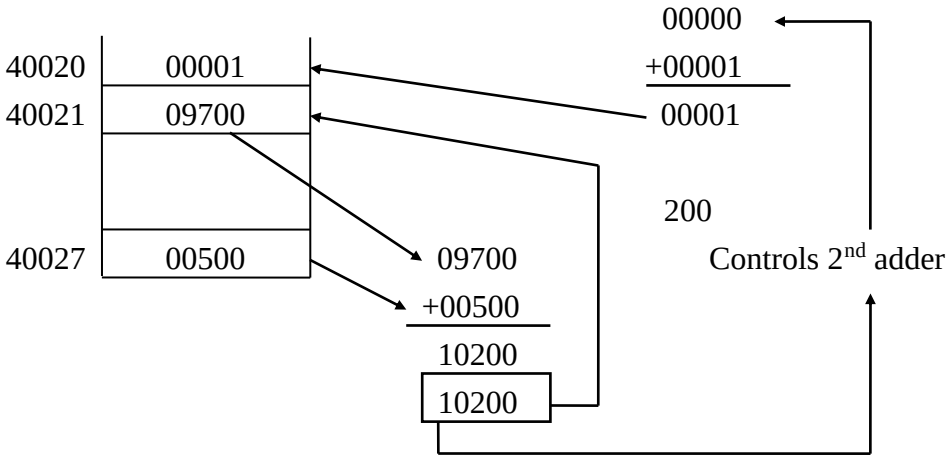
= 0, Sum ≦9999

【EXAMPLE】



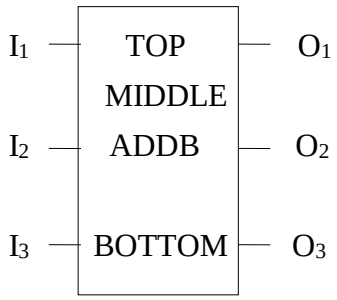
【DESCRIPTION】

When the contact 10025 has an “ON”(or ‘OFF’ to ‘ON’), the content of register 40021 is added to the content of register 40027 and the sum is stored back to register (40021). Since the sum is larger than 9999, therefore, the second adder is energized.



ADDB	**FOUR DIGIT HEXADECIMAL ADDER**	⎍	⎍↑

SYMBOL:



OPERANDS

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP			○	○	①	○	
MIDDLE			○	○	①	○	
BOTTOM				○		○	

①0~65535

word＋word→word (hexadecimal)

Description:

The values (hexadecimal) stored in the top and middle nodes are added and the sum is stored in the bottom node. Sum = (top + middle) MOD 65536.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Function output (O_3) may be used to determine whether or not an overflow has occurred.

Node Description:

TOP: Summand, must be ＜65535.

MIDDLE: Addend, must be ＜65535.

BOTTOM: Sum < 65535. The carry, if any, is ignored.

Input Control:

I_1: When ⎍ (⎍↑) is presented, the function block is executed.

Function Output:

$O_1 = I_1$

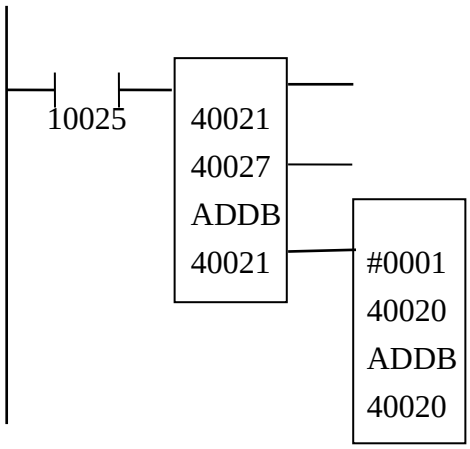
$O_2 = 0$

O_3：overflow

= 1, Sum ＞65535

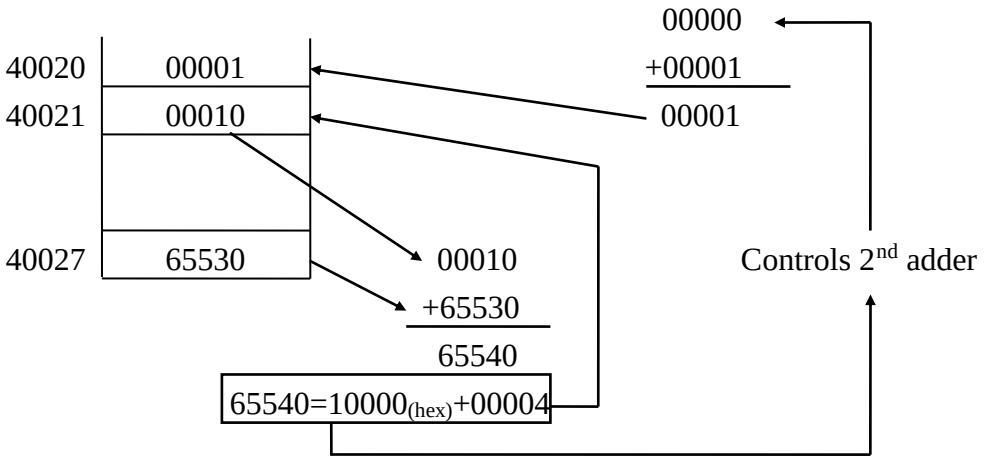
= 0, Sum ≦65535

【EXAMPLE】



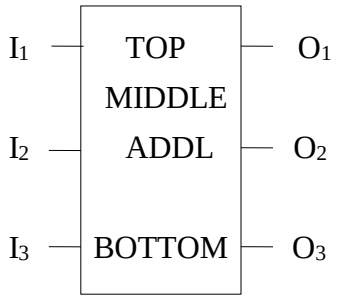
【DESCRIPTION】

When contact 10025 is energized, the content of register 40021 is added to the content of register 40027 and the sum is stored back to register 40021. Since the sum is larger than 65535, therefore, the second adder is energized.



ADDL	**EIGHT DIGIT DECIMAL ADDER**	⎍	⎍↑

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP				○		○	
MIDDLE				○	①	○	
BOTTOM				○		○	

① 0~9999

Lword ＋ Lword → Lword (Decimal)

Description:

The values (long word, decimal) stored in the top and middle nodes are added and the sum is stored in the bottom node. Sum = MOD (top + middle + I_3) MOD 100000000.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Function output (O_3) may be used to determine whether or not an overflow has occurred.

Node Description:

TOP: Summand, must be ＜100000000.

MIDDLE: Addend, must be ＜100000000.

BOTTOM :1.(top + middle + I_3) MOD 10000000.

2.If error (refer to O_2) occurred, the content of the bottom node remains unchanged.

Input Control:

I_1: When ⎍ (⎍↑) is presented, the function block is executed.

I_2: error in

I_3: carry in

Function Output:

$O_1 = I_1$

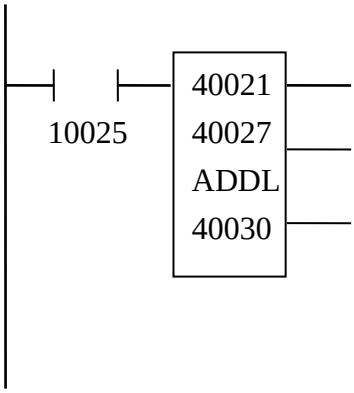
O_2 = error output (O_2 is '1' if I_2 is '1' or the value of either top node or middle node is over 99999999.)

O_3：overflow/carry

= 1, Sum ＞99999999

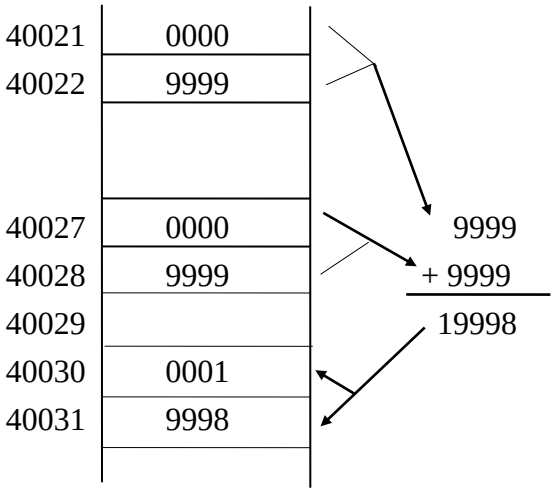
= 0, Sum ≦99999999

【EXAMPLE】



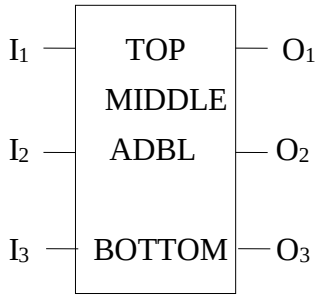
【DESCRIPTION】

When contact 10025 is “ON’ (or ‘OFF’ to ‘ON’), the content of registers 40027/40028 is added to the content of registers 40021/40022. The sum is stored in registers 40030/40031. Since the sum is less then 99999999, thus, O_1 ： ON, $O_2 = O_3$ = OFF.



ADBL	**EIGHT DIGIT HEXADECIMAL ADDER**	⎍	⎍↑

SYMBOL:



OPERANDS

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP				○		○	
MIDDLE				○	①	○	
BOTTOM				○		○	

①0~65535

Lword＋Lword→ Lword (Hexadecimal)

Description:

The values (long word, hexadecimal) stored in the top and middle nodes are added and the sum is stored in the bottom node. Sum = MOD (top+middle) FFFFFFFF.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Function output (O_3) may be used to determine whether or not an overflow has occurred.

Node Description:

TOP: Summand, must be ＜100000000_{hex}.

MIDDLE: Addend, must be ＜100000000_{hex}.

BOTTOM: Sum < 100000000_{hex}. The carry, if any, is ignored.

Input Control:

I_1: When ⎍ (⎍↑) is presented, the function block is executed.

Function Output:

$O_1 = I_1$

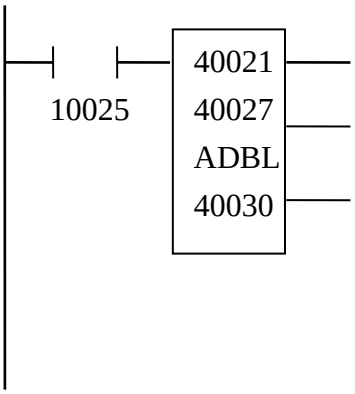
$O_2 = 0$

O_3：overflow

=1, Sum＞4294967295 (=100000000_{hex})

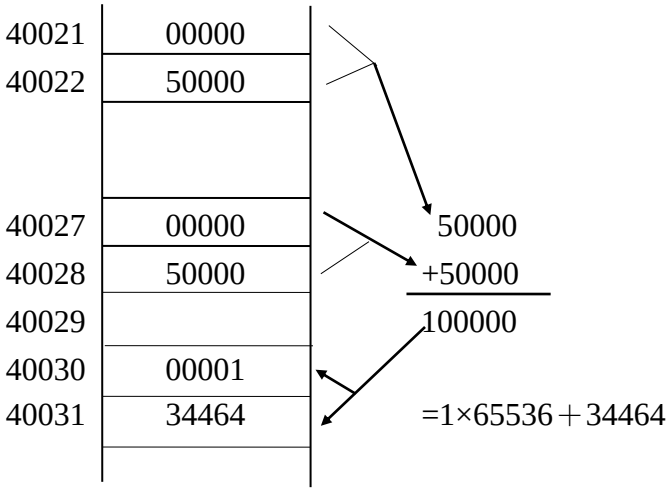
=0, Sum≦4294967295 (=100000000_{hex})

【EXAMPLE】



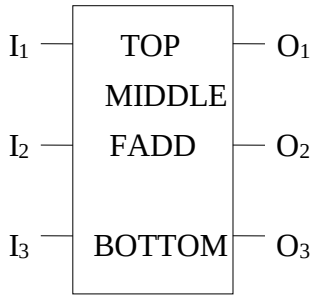
【DESCRIPTION】

When contact 10025 is “ON’ (or ‘OFF’ to ‘ON’), the content of registers 40027 & 40028 is added to the content of registers 40021/40022. The sum is stored in the registers 40030 & 40031.



FADD	FLOATING POINT ADDER	⎍	↑⎍

SYMBOL:



OPERANDS:

	0	1	3	4	C	P	L
TOP				○		○	
MIDDLE				○	←	○	
BOTTOM				○		○	

←0~65535

float ＋ float → float

Description:

The values (floating point) stored in the top and middle nodes are added and the sum is stored in the bottom node.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Node Description:

TOP: Summand.

MIDDLE: Addend.

BOTTOM: Sum.

Input Control:

I_1: When ⎍ (↑⎍) is presented, the function block is executed.

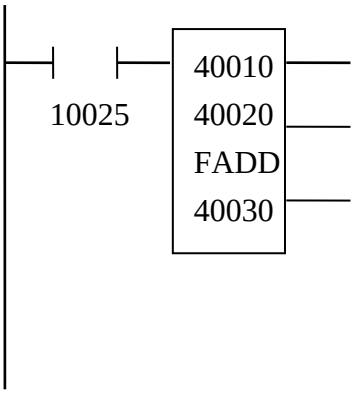
Function Output:

$O_1 = I_1$

$O_2 = 0$

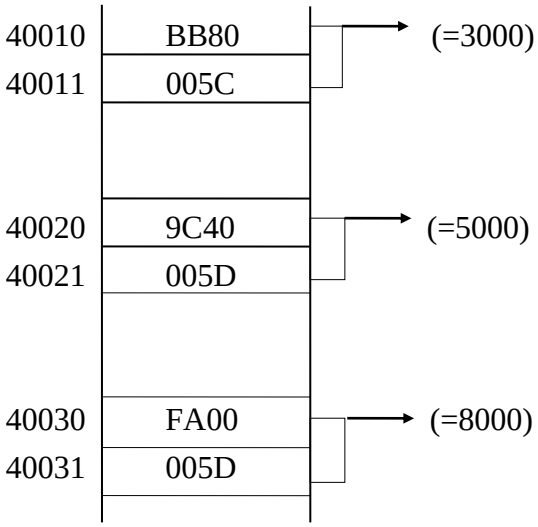
$O_3 = 0$

【EXAMPLE】



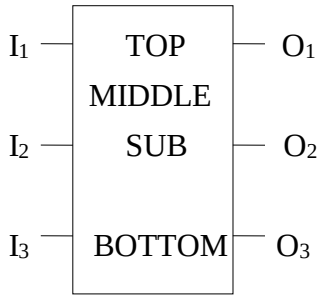
【DESCRIPTION】

When contact 10025 is “ON’ (or ‘OFF’ to ‘ON’), the content of registers 40010/40011 is added to the content of registers 40020/40021; the sum is stored in registers 40030/40031; and O_1：ON, $O_2 = O_3$ = OFF.



SUB	FOUR DIGIT DECIMAL SUBTRACTOR	⎍	↑⎍

SYMBOL:



OPERANDS:

	0	1	3	4	C	P	L
TOP			○	○	①	○	
MIDDLE			○	○	①	○	
BOTTOM				○		○	

①0~9999

word－word→word (Decimal)

Description:

The value stored in the middle node is subtracted from the top node, and the difference is stored in the bottom node.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Function output may be used to determine the relationship between minuend and subtrahend (＞, **=,** ＜).

Node Description:

TOP: Minuend, must be ＜10000.

MIDDLE: Subtrahend, must be ＜10000.

BOTTOM: Difference.

Input Control:

I_1: When ⎍ (↑⎍) is presented, the function block is executed.

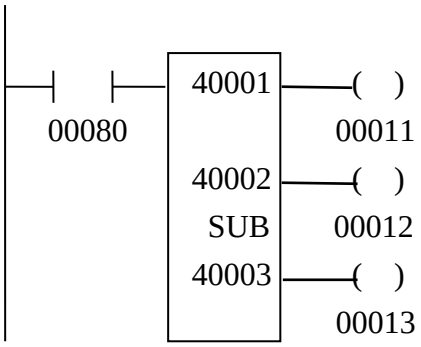
Function Output:

O_1 = 1, if difference > 0 (Top node > Middle node).

O_2 = 1, if difference = 0 (Top node = Middle node).

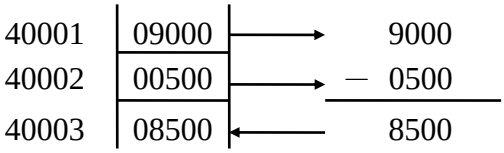
O_3 = 1, if difference < 0 (Top node < Middle node).

【EXAMPLE】



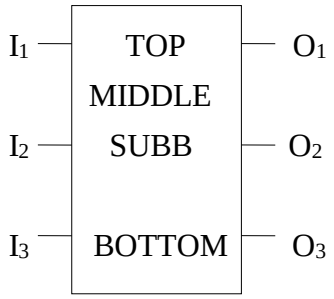
【DESCRIPTION】

Assume that register (40001)=9000(10), and (40002)=500(10). when contact 00080 is 'ON' (or 'OFF' to 'ON'), the subtraction: **(40003)=(40001)－(40002)** is performed. Since the minuend is larger than the subtrahend, thus coil 00011 is 'ON', 00012 is 'OFF' and 00013 is 'OFF'.



SUBB	**FOUR DIGIT HEXADECIMAL SUBTRACTOR**	⎍	⎍↑

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP			○	○	①	○	
MIDDLE			○	○	①	○	
BOTTOM				○		○	

①0~65535

word－word→word (Binary)

Description:

The value stored in the middle node is subtracted from the top node, and the difference is stored in the bottom node.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Function output may be used to determine the relationship between minuend and subtrahend (＞,=,＜).

Node Description:

TOP: Minuend

MIDDLE: Subtrahend

BOTTOM: Difference.

Input Control:

I_1: When ⎍ (⎍↑) is presented, the function block is executed.

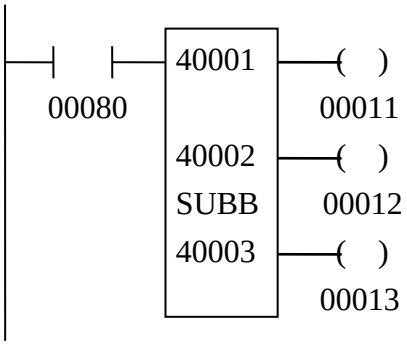
Function Output:

O_1 = 1, if difference > 0.

O_2 = 1, if difference = 0.

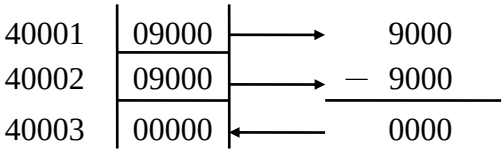
O_3 = 1, if difference < 0.

【EXAMPLE】



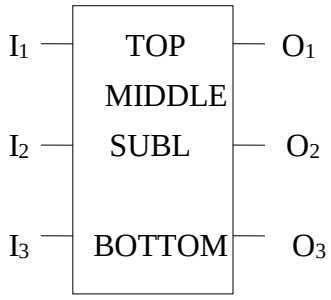
【DESCRIPTION】

Assume that register (40001)=9000(10), and (40002)=9000(10). when contact 00080 is ‘ON’ (or ‘OFF’ to ‘ON’), the subtraction: **(40003)=(40001)－(40002)** is performed. Since the minuend is equal to the subtrahend, thus coil 00012 is ‘ON’.



SUBL	**EIGHT DIGIT DECIMAL SUBTRACTOR**	⎍	⎍↑

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP				○	←	○	
MIDDLE				○	←	○	
BOTTOM				○		○	

←0~65535

Lword－Lword→Lword (Decimal)

Description:

The value stored in the middle node is subtracted from the top node, and the difference is stored in the bottom node.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Function output may be used to determine the relationship between minuend and subtrahend (＞, =, ＜).

Node Description:

TOP: Minuend, must be ＜=99999999.

MIDDLE: Subtrahend, must be ＜=99999999.

BOTTOM: Difference.

|

Input Control:

I_1: When ⎍ (⎍↑) is presented, the function block is executed.

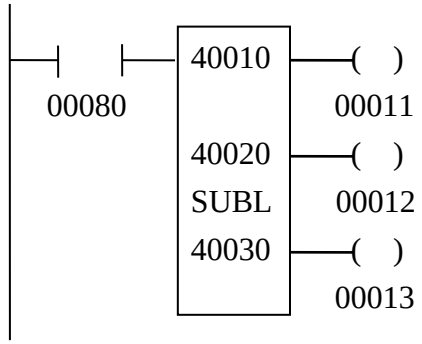
Function Output:

O_1 = 1, if difference > 0.

O_2 = 1, if difference = 0.

O_3 = 1, if difference < 0.

【EXAMPLE】



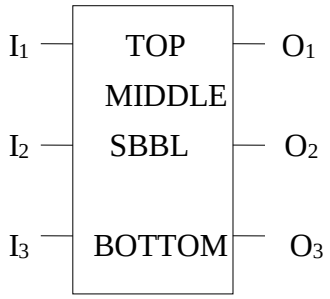
【DESCRIPTION】

Assume that long word (40010)=$9999_{(10)}$ and long word(40020)=$9999_{(10.)}$. when contact 00080 is 'ON' (or 'OFF' to 'ON'), the operation: long word**(40030)=long word(40010)－long word(40020)** is performed. Since the minuend is equal to the subtrahend, thus coil 00012 is 'ON'.

40010	0000
40011	9999
40020	0000
40021	9999
40030	0000
40031	0000

SBBL	**EIGHT DIGIT HEXADECIMAL SUBTRACTOR**	⎍	⎍↑

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP				○	←	○	
MIDDLE				○	←	○	
BOTTOM				○		○	

←0~65535

Lword－Lword→Lword(Binary)

Description:

The value stored in the middle node is subtracted from the top node, and the difference is stored in the bottom node.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Function output may be used to determine the relationship between minuend and subtrahend (＞, =, ＜).

Node Description:

TOP: Minuend.

MIDDLE: Subtrahend.

BOTTOM: Difference .

Input Control:

I_1: When ⎍ (⎍↑) is presented, the function block is executed.

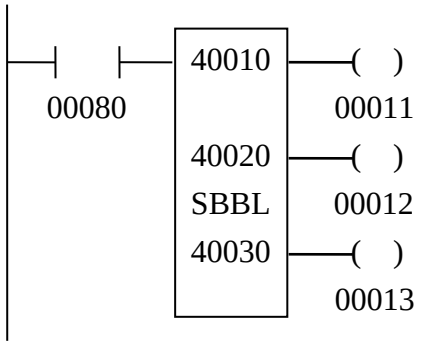
Function Output:

O_1 = 1, if difference > 0.

O_2 = 1, if difference = 0.

O_3 = 1, if difference < 0.

【EXAMPLE】



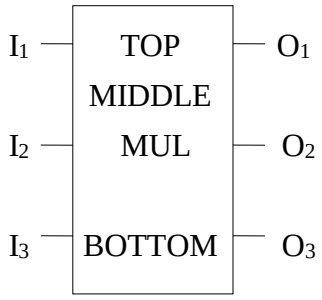
【DESCRIPTION】

Assume that long word (40010) = $65536_{(10)}$ and long word (40020) = $65536_{(10)}$. when contact 00080 is 'ON(or 'OFF' to 'ON')', the operation: long word**(40030)=long word(40010)－long word(40020)** is performed. Since the minuend is equal to the subtrahend, thus coil 00012 is 'ON'.

40010	0001
40011	0000
40020	0001
40021	0000
40030	0000
40031	0000

MUL	**FOUR DIGIT DECIMAL MULTIPLIER**	⎍	⎍↑

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP			○	○		○	
MIDDLE			○	○	①	○	
BOTTOM				○		○	

①0~9999

word × word→Lword (Decimal)

Description:

The value in the top node is multiplied by the value in the middle node, and the product is stored in the bottom node (long word).

Input control (I_1) is used to determine whether this function block is to be executed or not.

Function output (O_3) may be used to determine whether or not an overflow has occurred.

Node Description:

TOP: Multiplicand, must be <=9999.

MIDDLE: Multiplier, must be <=9999.

BOTTOM: 1. Product, Long word.

2. If error (refer to O_2) occurred, the content of the bottom node remains unchanged.

Input Control:

I_1: When ⎍ (⎍↑) is presented, the function block is executed.

I_2: error in

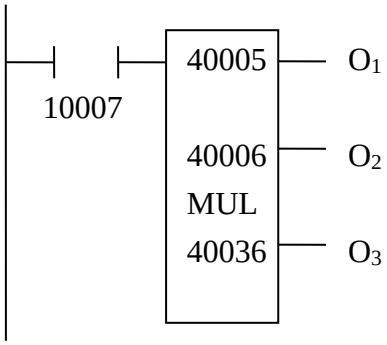
Function Output:

$O_1 = I_1$

$O_2 = 1$ (If the value of either top node or middle node is greater than 9999).

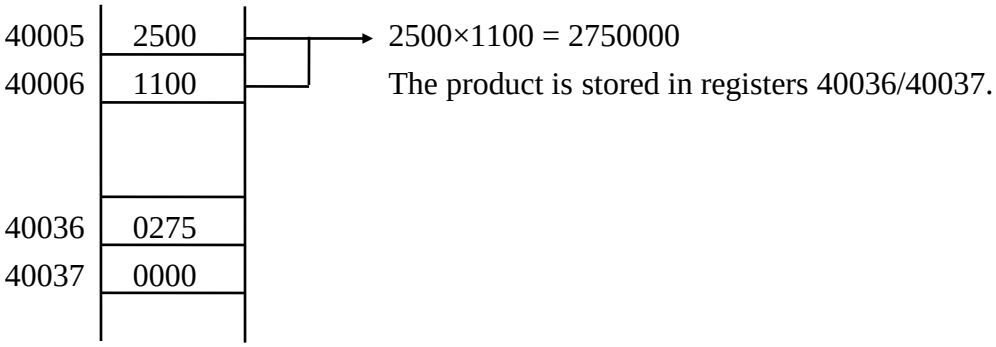
O_3：0

【EXAMPLE】



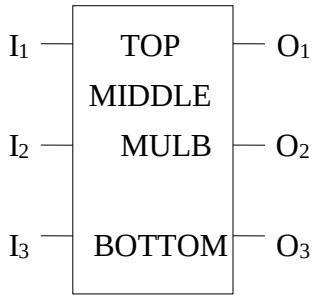
【DESCRIPTION】

Let register (40005)=2500 and (40006)=1100. When contact 10007 is ‘ON’ (or ‘OFF’ to ‘ON’), the operation:
long word (40036)=(40005)×(40006) is performed.



MULB	**FOUR DIGIT HEXADECIMAL MULTIPLIER**	⎍	↑⎍

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP			○	○		○	
MIDDLE			○	○	①	○	
BOTTOM				○		○	

①0~65535

word ×word→Lword (Binary)

Description:

The value in the top node is multiplied by the value in the middle node, and the product is stored in the bottom node (long word).

Input control (I_1) is used to determine whether this function block is to be executed or not.

Node Description:

TOP: Multiplicand.

MIDDLE: Multiplier.

BOTTOM: Product, Long word.

Input Control:

I_1: When ⎍ (↑⎍) is presented, the function block is executed.

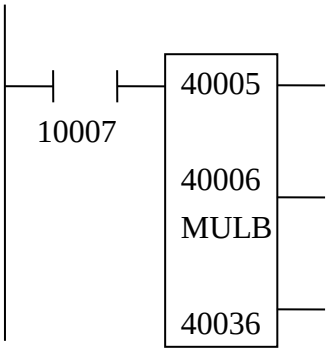
Function Output:

$O_1 = I_1$

$O_2 = 0$

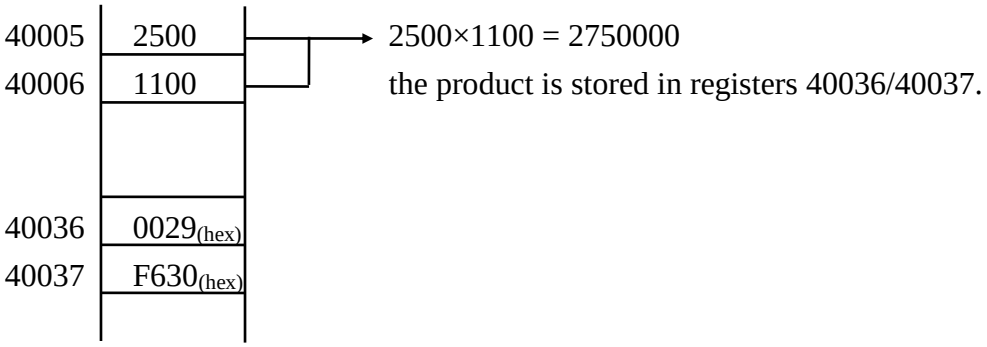
$O_3 = 0$

【EXAMPLE】



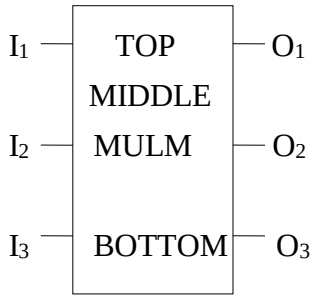
【DESCRIPTION】

Let register (40005)=2500 and (40006)=1100. When contact 10007 is ‘ON’ (or ‘OFF’ to ‘ON’), the operation:
long word (40036)=(40005)×(40006) is performed.



MULM	**FOUR DIGIT DECIMAL MULTIPLIER**	⎍	↑⎍

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP			○	○		○	
MIDDLE			○	○	①	○	
BOTTOM				○		○	

①0~9999

word × word→word (Decimal)

Description:

The value in the top node is multiplied by the value in the middle node, and the product is stored in the bottom node (long word).

Input control (I_1) is used to determine whether this function block is to be executed or not.

Function output (O_3) may be used to determine whether or not an overflow has occurred .

Node Description:

TOP: Multiplicand (<= 9999).

MIDDLE: Multiplier(<=9999).

BOTTOM: 1. Product.

2. If error (refer to O_2) occurred, the content of the bottom node remains unchanged.

Input Control:

I_1: When ⎍ (↑⎍) is presented, the function block is executed.

I_2: error in.

Function Output:

$O_1 = I_1$

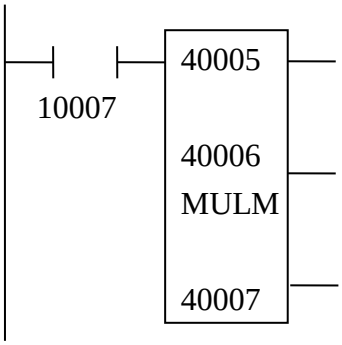
O_2 =1 (If the value of either top node or middle node is greater than 9999, or I_2 = 1)

O_3：Overflow

=1,Product≧10000

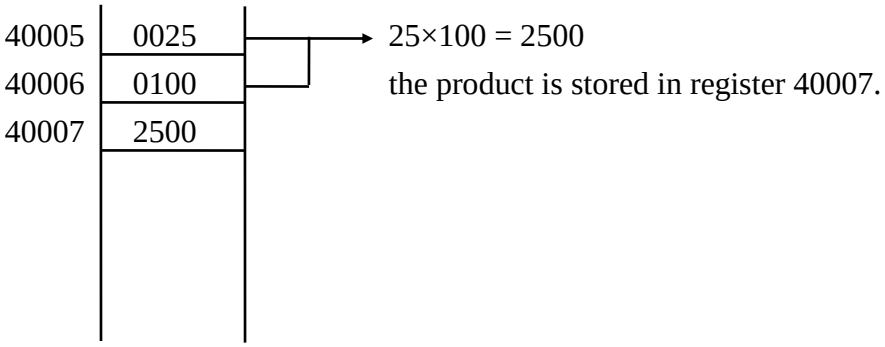
=0,Product＜10000

【EXAMPLE】



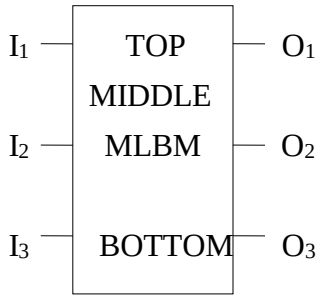
【DESCRIPTION】

Let register (40005)=25 and (40006)=100. When contact 10007 is ‘ON’ (or ‘OFF’ to ‘ON’), the operation:
(40036)=(40005)×(40006) is performed.



MLBM	**FOUR DIGIT HEXADECIMAL MULTIPLIER**	⎍	↑⎍

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP			○	○		○	
MIDDLE			○	○	①	○	
BOTTOM				○		○	

①0~65535

word × word→word (Binary)

Description:

The value in the top node is multiplied by the value in the middle node, and the product is stored in the bottom node (long word).

Input control (I_1) is used to determine whether this function block is to be executed or not.

Function output (O_3) may be used to determine whether or not an overflow has occurred.

Node Description:

TOP: Multiplicand.

MIDDLE: Multiplier.

BOTTOM: Product.

Input Control:

I_1: When ⎍ (↑⎍) is presented, the function block is executed.

Function Output:

$O_1 = I_1$

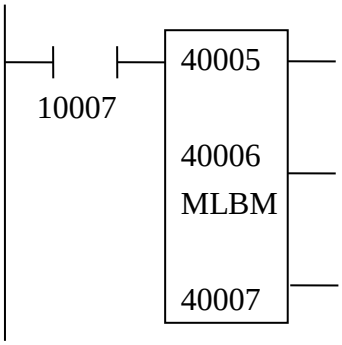
$O_2 = 0$

O_3：Overflow

=1,Product≧65536

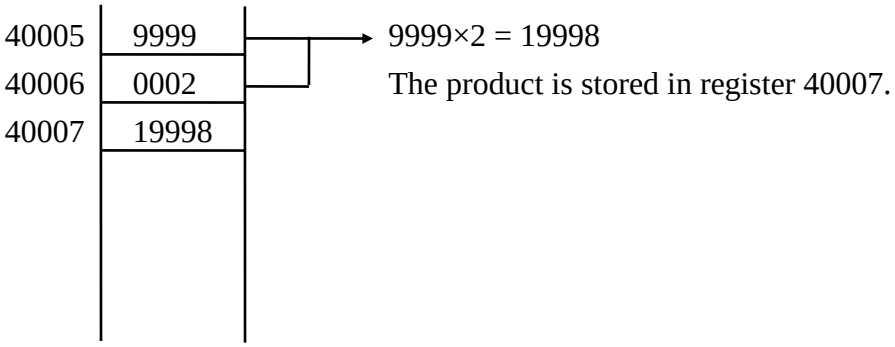
=0,Product＜65536

【EXAMPLE】



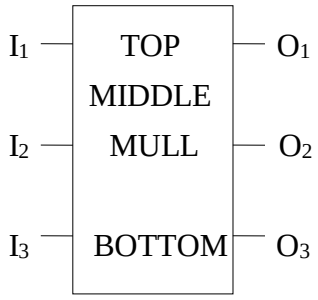
【DESCRIPTION】

Let register (40005)=9999 and (40006)=2. When contact 10007 is ‘ON’ (or ‘OFF’ to ‘ON’), the operation:
(40036)=(40005)×(40006) is performed.



MULL	EIGHT DIGIT DECIMAL MULTIPLIER	⎍	⎍↑

SYMBOL:



OPERANDS:

	0	1	3	4	C	P	L
TOP				○		○	
MIDDLE				○	①	○	
BOTTOM				○		○	

①0~65535

Lword × Lword→Lword (Decimal)

Description:

The value in the top node is multiplied by the value in the middle node, and the product is stored in the bottom node. All operands are long words.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Function output (O_3) may be used to determine whether or not an overflow has occurred.

Node Description:

TOP: Multiplicand, must be <= 99999999.

MIDDLE: Multiplier, must be <= 99999999.

BOTTOM: 1. Product.

2. If error (refer to O_2) occurred, the content of the bottom node remains unchanged.

Input Control:

I_1: When ⎍ (⎍↑) is presented, the function block is executed.

I_2: error in

Function Output:

$O_1 = I_1$

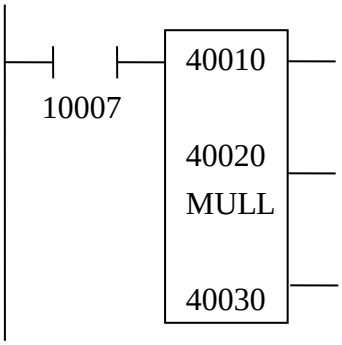
$O_2 = 1$ (If the value of either top node or middle node is greater than 99999999, or $I_2 = 1$).

O_3：Overflow

=1, Product≧100000000

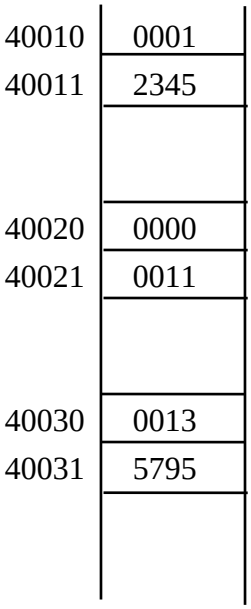
=0, Product＜100000000

【EXAMPLE】



【DESCRIPTION】

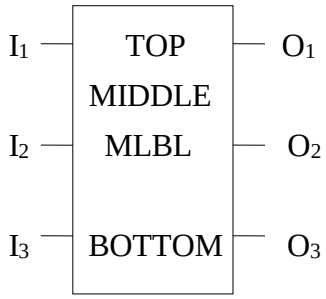
Let register (40010/40011)=12345 and (40020/40021)=11. When contact 10007 is ‘ON’ (or ‘OFF’ to ‘ON’), the operation: long word(40030)=long word(40010)×long word(40020) is performed.



DECIMAL

MLBL	**EIGHT DIGIT HEXADECIMAL MULTIPLIER**	⎍	⎍↑

SYMBIL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP				○		○	
MIDDLE				○	①	○	
BOTTOM				○		○	

①0~65535

Lword × Lword→Lword (Binary)

Description:

The value in the top node is multiplied by the value in the middle node, and the product is stored in the bottom node. All operands are long words.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Function output (O_3) may be used to determine whether or not an overflow has occurred .

Node Description:

TOP: Multiplicand.

MIDDLE: Multiplier.

BOTTOM: Product.

Input Control:

I_1: When ⎍ (⎍↑) is presented, the function block is executed.

Function Output:

$O_1 = I_1$

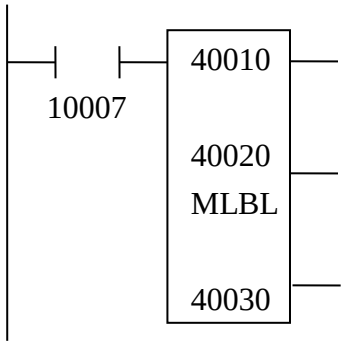
$O_2 = 0$

O_3：Overflow

=1, Product＞= 4294967296 (=100000000_{hex})

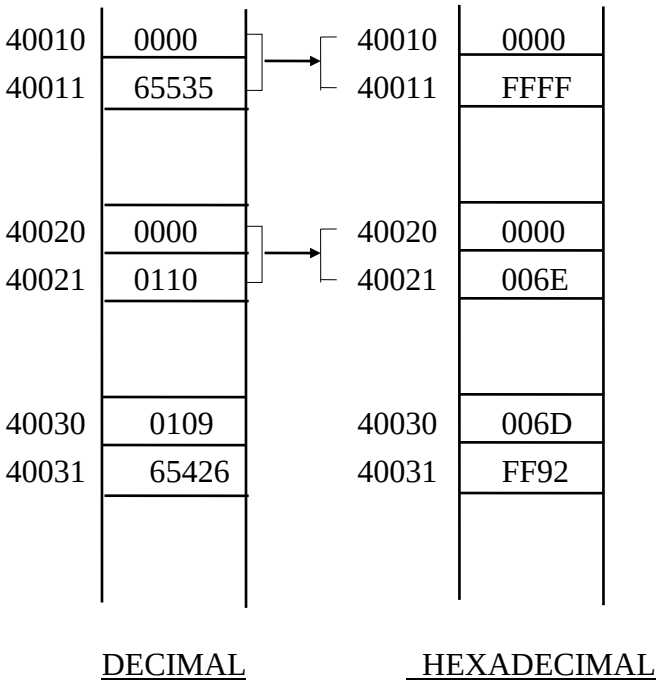
=0, Product＜ 4294967296 (=100000000_{hex})

【EXAMPLE】



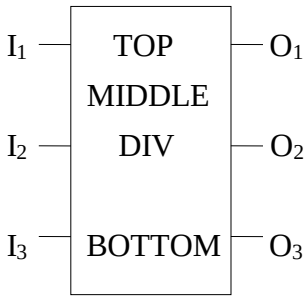
【DESCRIPTION】

Let register (40010/40011)=65535 and (40020/40021)=11. When contact 10007 is ‘ON’ (or ‘OFF’ to ‘ON’), the operation: long word(40030)=long word(40010)×long word(40020) is performed.



DIV	**FOUR DIGIT DECIMAL DIVIDER(1)**	⎍	⎍↑

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP			○	○	①	○	
MIDDLE			○	○	①	○	
BOTTOM				○		○	

①0~9999

Lword÷word→word (Decimal)

Description:

The value stored in the top node is divided by the value in the middle node, and the result is stored in the bottom node.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Function outputs can be used to determine whether the function block has been executed, divisor is zero and overflow.

Node Description:

TOP: Constant dividend, must be ＜= 9999; **else the LONG WORD value is used.**

MIDDLE: Divisor, must be＜= 9999

BOTTOM: 1.Result of Division. The quotient is stored in the first word. **Depending on the input control, the remainder or the first four digits after decimal point of quotient are stored in the second word.**

2.If error occurred, the content of the bottom node remains unchanged.

Input Control:

I_1: When ⎍ (⎍↑) is presented, the function block is executed.

I_2 = 0, the second word of the bottom node is used to store the remainder.

= 1, the second word of the bottom node is used to store the first four digits after the decimal point.

I_3：error in

Function Output:

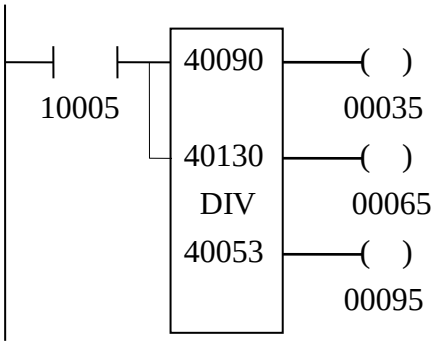
$O_1 = I_1$

O_2 = 1, if overflow, i.e. quotient＞9999

O_3 (error output)= 1 (1.If the value of either top node or middle node is greater than 9999 or

2.If divisor = 0)

【**EXAMPLE**】



(DESCRIPTION)

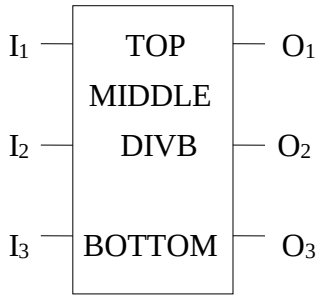
Let **long word** (40090)=9999 and (40130) =10. When contact 10005 is energized, I_1 and I_2 = 'ON'. The quotient (=999) is stored in register 40053. Since I_2 = 'ON', thus the first four digits (=9000) are stored in register 40054.

40053	0999	Integer portion of the quotient
40054	9000	First four digits of the fractional portion of the quotient
40090	0000	Dividend
40091	9999	
40130	0010	Divisor

9999÷10 = 999.9000

DIVB	**EIGHT DIGIT HEXADECIMAL DIVIDER(1)**	⎍	⎍↑

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP			○	○	①	○	
MIDDLE			○	○	①	○	
BOTTOM				○		○	

①0~65535

Lword÷word→word (Binary)

Description:

The value stored in the top node is divided by the value in the middle node, and the result is stored in the bottom node.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Function outputs can be used to determine whether the function block has been executed, divisor is zero and overflow.

Node Description:

TOP: Dividend, (**long word**)

MIDDLE: Divisor

BOTTOM: 1. Result of Division. The quotient is stored in the first word. **Depending on the input control, the remainder or the first four digits after decimal point of quotient are stored in the second word.**

2. If error occurred, the content of the bottom node remains unchanged.

Input Control:

I_1: When ⎍ (⎍↑) is presented, the function block is executed.

I_2 =0, the second word of the bottom node is used to store the remainder.

=1, the second word of the bottom node is used to store the first four digits after the decimal point.

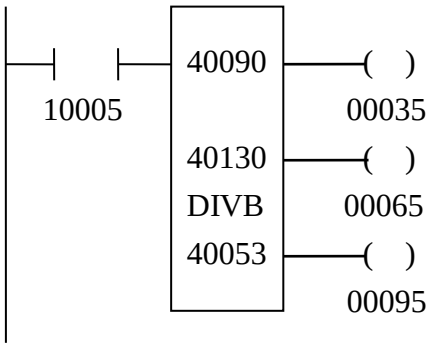
Function Output:

$O_1 = I_1$

O_2 = 1, if overflow, i.e. quotient＞65535

O_3 (error output)= 1, if divisor = 0.

【EXAMPLE】



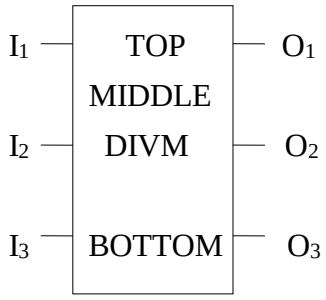
【DESCRIPTION】

Let **long word** (40090)=65535 and (40130) =12. When contact 10005 is energized, I_1 = 'ON', and the quotient (=5461) is stored in register 40053. Since I_2 = 'OFF', the remainder (=0003) is stored in register 40054.

40053	5461	Quotient	
40054	0003	Remainder	65535÷12 = 5461, remainder 3
40090	0000	Dividend	
40091	65535		
40130	0012	Divisor	

DIVM	**FOUR DIGIT DECIMAL DIVIDER(2)**	⎍	⎍↑

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP			○	○	①	○	
MIDDLE			○	○	①	○	
BOTTOM				○		○	

①0~9999

word÷word→word (Decimal)

Description:

The value stored in the top node is divided by the value in the middle node, and the result is stored in the bottom node.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Function outputs can be used to determine whether the function block has been executed, divisor is zero and overflow.

Node Description:

TOP: Dividend, must be <= 9999.

MIDDLE: Divisor, must be <= 9999.

BOTTOM: 1. Result of Division. The quotient is stored in the first word. **Depending on the input control, the remainder or the first four digits after decimal point of quotient are stored in the second word.**

2. If error occurred, the content of the bottom node remains unchanged.

Input Control:

I_1: When ⎍ (⎍↑) is presented, the function block is executed.

I_2 =0, the second word of the bottom node is used to store the remainder.

=1, the second word of the bottom node is used to store the first four digits after the decimal point.

I_3 = error in

Function Output:

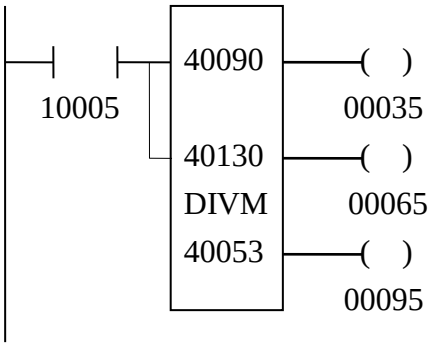
$O_1 = I_1$

$O_2 = 0$

O_3 (error output) = 1 (1. If divisor = 0 or

2. If the value of either top node or middle node is greater than 9999.)

【EXAMPLE】



【DESCRIPTION】

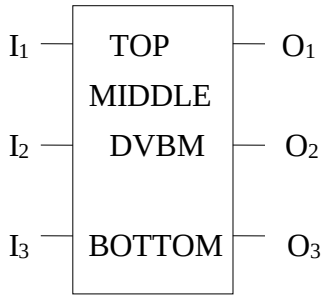
Let **long word** (40090)=9999 and (40130) =10. When contact 10005 is energized, I_1 and I_2 = 'ON'. The quotient (=999) is stored in register 40053. Since I_2 = 'ON', thus the first four digits (=9000) are stored in register 40054.

9999÷10 = 999.9000

40053	0999	Quotient
40054	9000	First four digits of the fractional portion of the quotient
40090	9999	Dividend
40130	0010	Divisor

DVBM	**FOUR DIGIT HEXADECIMAL DIVIDER**	⎍	⎍↑

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP			○	○	①	○	
MIDDLE			○	○	①	○	
BOTTOM				○		○	

①0~65535

word÷word→word (Binary)

Description:

The value stored in the top node is divided by the value in the middle node, and the result is stored in the bottom node.

Input control ($I1_{)}$ is used to determine whether this function block is to be executed or not.

Function outputs can be used to determine whether the function block has been executed, divisor is zero and overflow.

Node Description:

TOP: Dividend.

MIDDLE: Divisor.

BOTTOM: 1.Result of Division. The quotient is stored in the first word. **Depending on the input control, the remainder or the first four digits after decimal point of quotient are stored in the second word.**

2.If error occurred, the content of the bottom node remain unchanged.

Input Control:

I_1: When ⎍ (⎍↑) is presented, the function block is executed.

I_2 =0, the second word of the bottom node is used to store the remainder.

=1, the second word of the bottom node is used to store the first four digits after the decimal point.

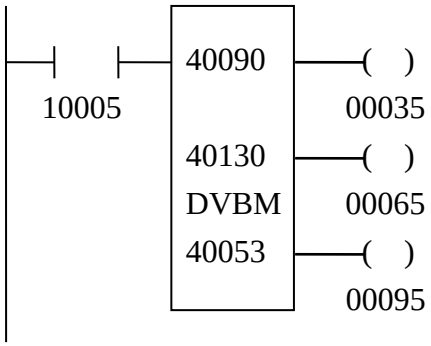
Function Output:

$O_1 = I_1$

$O_2 = 0$

O_3 (error output) = 1, if divisor = 0.

【EXAMPLE】



【DESCRIPTION】

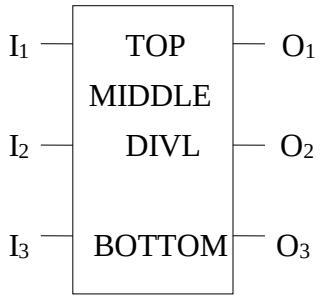
Let **long word** (40090)=65535 and (40130) =12. When contact 10005 is energized, I_1 = 'ON', and the quotient (=5461) is stored in register 40053. Since I_2 = 'OFF', the remainder (=0003) is stored in register 40054.

40053	5461	Quotient
40054	0003	Remainder
40090	65535	Dividend
40130	0012	Divisor

65535÷12 = 5461, remainder 3

DIVL	**EIGHT DIGIT DECIMAL DIVIDER(2)**	⎍	⎍↑

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP				○	←	○	
MIDDLE				○	←	○	
BOTTOM				○		○	

←0~65535

Lword÷Lword→Lword (Decimal)

Description:

The value stored in the top node is divided by the value in the middle node, and the result is stored in the bottom node. Input control (I_1) is used to determine whether this function block is to be executed or not. Function outputs can be used to determine whether the function block has been executed, divisor is zero and overflow.

Node Description:

TOP: Dividend, must be <= 99999999.

MIDDLE: Divisor, must be <= 99999999.

BOTTOM: 1. Result of Division. The quotient is stored in the first and second words. Depending on the input control, the remainder or the first eight digits after decimal point of quotient are stored in the third and fourth words.

2. If error occurred, the content of the bottom node remains unchanged.

Input Control:

I_1: When ⎍ (⎍↑) is presented, the function block is executed.

I_2 =0, the third and the fourth words of the bottom node is used to store the remainder.

=1, the third and the fourth words of the bottom node is used to store the first eight digits after the decimal point.

I_3：error in

Function Output:

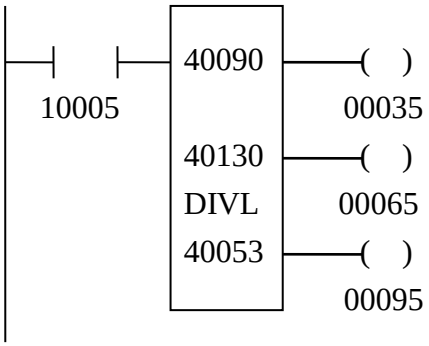
$O_1 = I_1$

$O_2 = 0$

O_3 (error output)= 1 (1. If divisor = 0 or

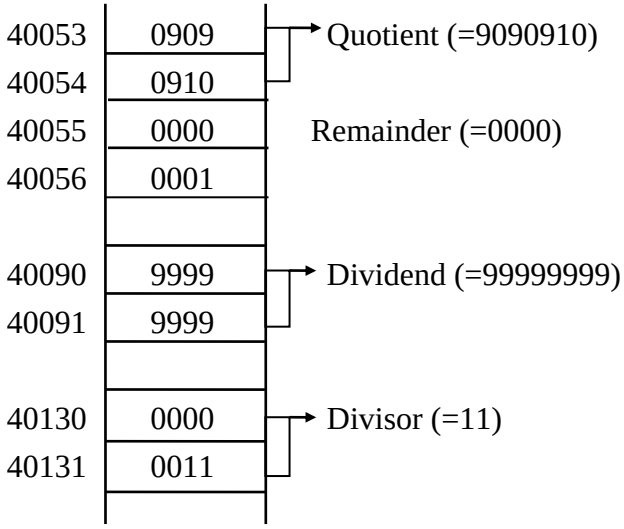
2. If the value of either top node or middle node is greater than 99999999)

【EXAMPLE】



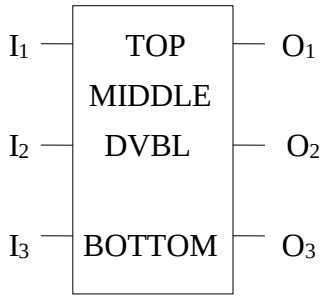
【DESCRIPTION】

Let **long word** (40090)=99999999 and **long word**(40130) =11. When contact 10005 is energized, I_1 = 'ON', and the quotient (=9090910) is stored in the **long word** 40053. Since I_2 = 'OFF', the remainder (=0001) is stored in register 40055 and 40056.



DVBL	**EIGHT DIGIT HEXADECIMAL DIVIDER**	⎍	⎍↑

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP				○	①	○	
MIDDLE				○	①	○	
BOTTOM				○		○	

①0~65535

Lword÷Lword→Lword

Description:

The value stored in the top node is divided by the value in the middle node, and the result is stored in the bottom node.

Input control ($I1_{)}$ is used to determine whether this function block is to be executed or not.

Function outputs can be used to determine whether the function block has been executed, divisor is zero and overflow.

Node Description:

TOP: Dividend.

MIDDLE: Divisor.

BOTTOM: Result of Division. The quotient is stored in the first and second words. Depending on the input control, the remainder or the eight digits after decimal point of quotient are stored in the third and fourth words.

Input Control:

I_1: When ⎍ (⎍↑) is presented, the function block is executed.

I_2 =0, the third and the fourth words of the bottom node is used to store the remainder.

=1, the third and the fourth words of the bottom node is used to store the first eight digits after the decimal point.

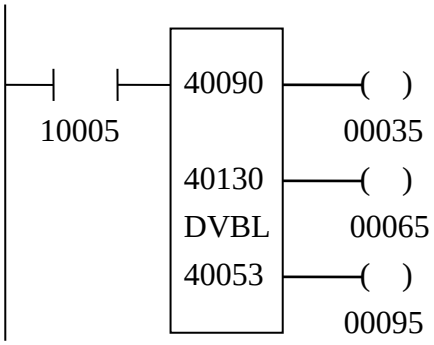
Function Output:

$O_1 = I_1$

$O_2 = 0$

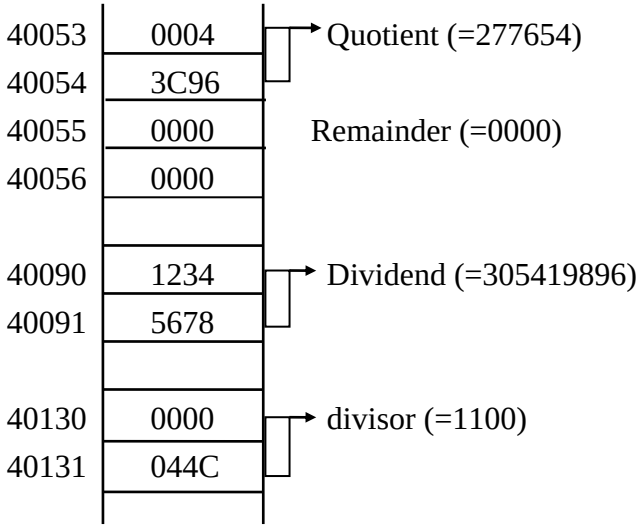
O_3 = 1, if divisor = 0.

【EXAMPLE】



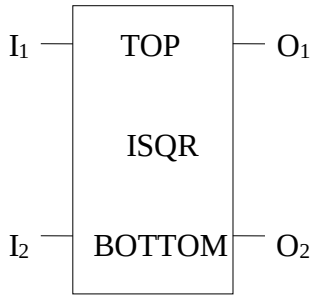
【DESCRIPTION】

Let **long word** (40090)=305419896 and **long word**(40130) =1100. When contact 10005 is energized, I_1 = 'ON', and the quotient (=277654) is stored in **long word** 40053. Since I_2 = 'OFF', the remainder (=0000) is stored in register 40055 and 40056.



ISQR	**SQUARE ROOT OF AN INTEGER**	⎍	⎍↑

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP			○	○		○	
BOTTOM				○		○	

Description:

The square root of the value stored in the top node is found and stored in the bottom node. The result of the square root operation is truncated to integer. Input control ($I1_{)}$ is used to determine whether this function block is to be executed or not. Function outputs can be used to determine whether the function block has been executed.

Node Description:

TOP: An integer whose square root is desired.

BOTTOM: Square root.

Input Control:

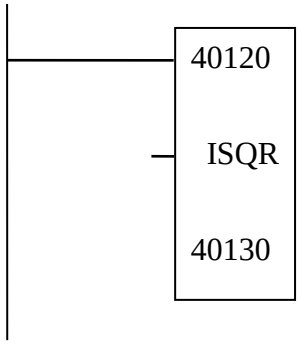
I_1: When ⎍ (⎍↑) is presented, the function block is executed.

Function Output:

$O_1=I_1$

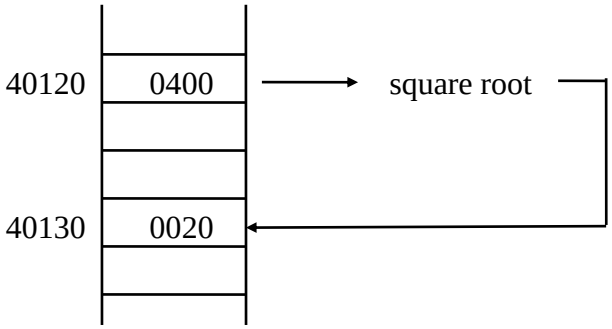
$O_2=0$

【EXAMPLE】



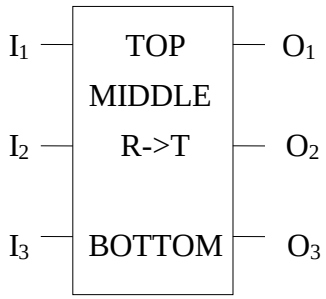
【DESCRIPTION】

Let (40120) =400. When this rung is scanned, the square roots of the values stored in the top node are stored in the bottom node.



R->T	**MOVE FROM REGISTER TO TABLE**	⎍	⎍↑

SYMBOL:



OPERANDS:

	0	1	3	4	C	P	L
TOP	○	○	○	○		○	
MIDDLE	○			○		○	
BOTTOM					①		

①1~255

Description:

The content of the top node is filled onto the table defined in the following address by the middle node. Table length is defined in the bottom node.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Input control (I_2) is used to define the action mode of the INDEX.

Input control (I_3) is used to clear the INDEX.

Function outputs can be used to determine whether the function block has been executed and whether the INDEX exceeded the table length.

Node Description:

TOP: Source register.

MIDDLE: Reference register. First word defined as INDEX into the target table. If the value of the INDEX is equal to zero, then the INDEX is pointing to the first entry in the target table. The target table starts with the second word.

BOTTOM: Table Length. If the INDEX value is greater than or equal to this number, table movement is prohibited disregarding the state of I_1.

Input Control:

I_1: When ⎍ (⎍↑) is presented, the function block is executed.

I_2: INDEX control.

=0, INDEX is incremented by one after each execution.

=1, INDEX remains unchanged.

I_3: Reset INDEX.

=1, clear INDEX to 0.

Function Output:

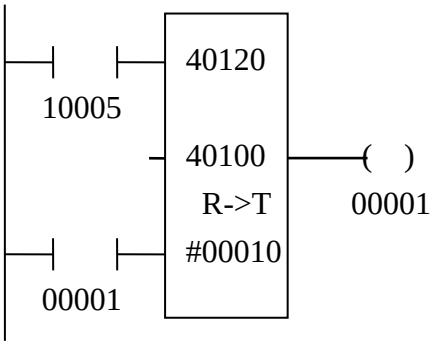
$O_1 = I_1$

O_2: INDEX indicator.

=1, INDEX ≧table length, the INDEX is pointing to an address beyond table limit.

O_3=0

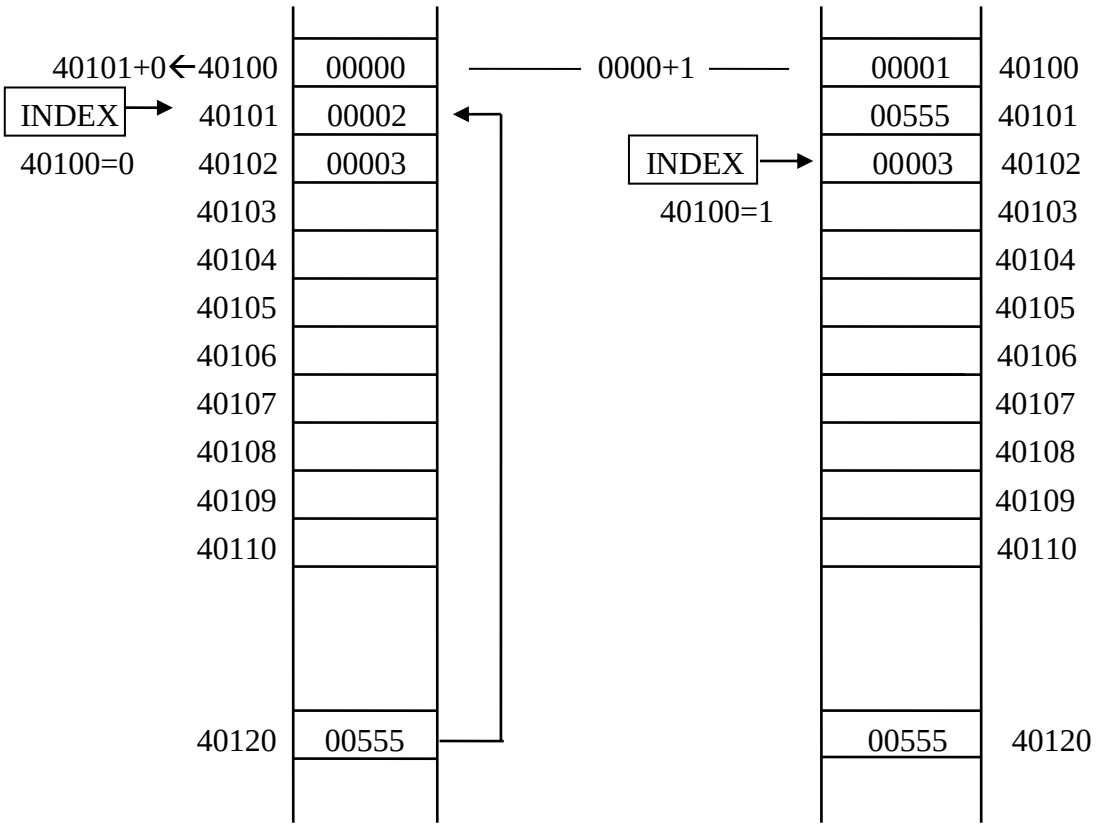
【EXAMPLE】



【DESCRIPTION】

When contact 10005 is energized, the content of input register 40120 is copied to table registers (40101~40110), one register per scan. And the action of INDEX (40100) increases by one after each scan.

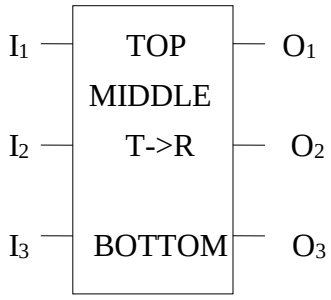
When INDEX (in 40100) reaches preset value of BOTTOM node (#00010), then coil 00001 is energized and the content of register 40100 is cleared. The movement continues until contact 10005 is OFF.



BEFORE **AFTER**

T->R	**MOVE FROM TABLE TO REGISTER**	⎍	⎍↑

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP	○	○	○	○		○	
MIDDLE	○			○		○	
BOTTOM					①		

①1~255

Description:

The content of the top node is moved to the following address defined by the middle node. Table length is defined in the bottom node.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Input control (I_2) is used to define the action mode of the INDEX.

Input control (I_3) is used to clear the INDEX.

Function outputs can be used to determine whether the function block has been executed and whether the INDEX exceeded the table length.

Node Description:

TOP: Source table.

MIDDLE: Source INDEX is defined at the first word. If the value of the INDEX is equal to zero, then the INDEX is pointing to the first entry in the source table. The target register is in the second word.

BOTTOM: Table Length. If the INDEX value is greater than or equal to this number, table movement is prohibited disregarding the state of I_1.

Input Control:

I_1: When ⎍ (⎍↑) is presented, the function block is executed.

I_2: INDEX control.

=0, INDEX is incremented by one after each execution.

=1, INDEX remains unchanged.

I_3: Reset INDEX.

=1, clear INDEX to 0.

Function Output:

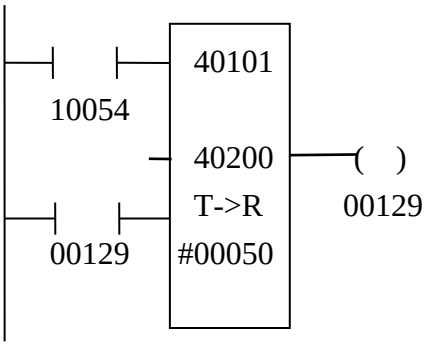
$O_1 = I_1$

O_2: INDEX indicator.

=1, INDEX ≧table length, the INDEX is pointing to an address beyond table limit.

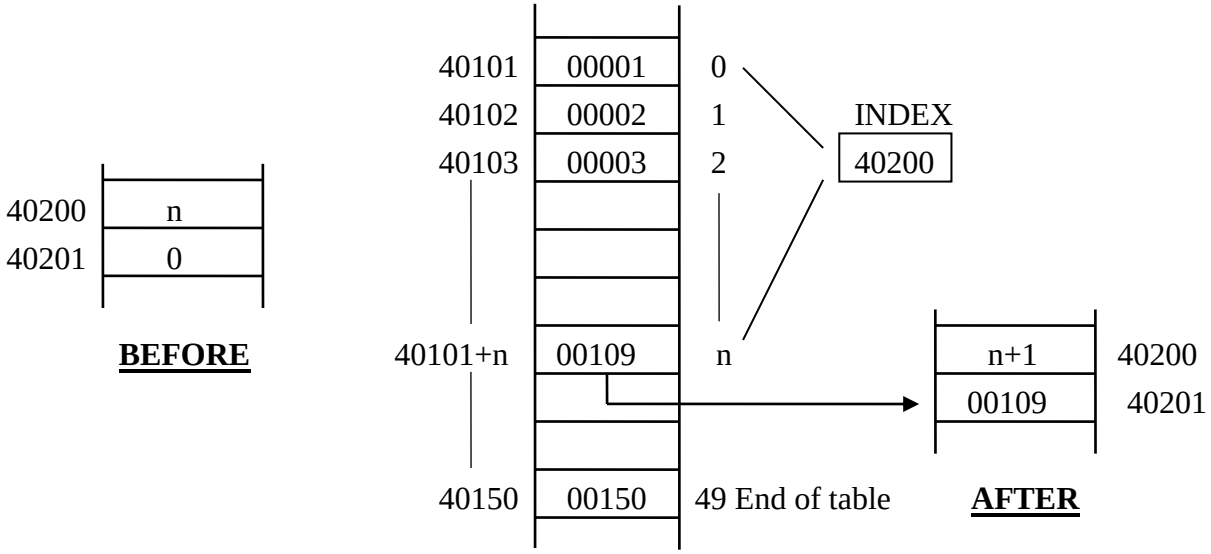
O_3=0

【EXAMPLE】



【DESCRIPTION】

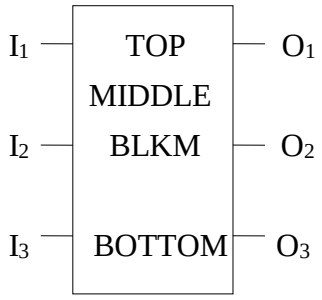
When contact 10054 is energized, source data pointed to by INDEX (40101) is moved to 40201 (the next address defined by the middle node). For every scan of the PLC controller, data movement occurs once until the INDEX reaches the end of table (#00050). Then Coil 00129 is energized and the INDEX is cleared. In this manner, data movement can be repeated. The following is the state after the nth scan since INDEX reset to 0.



Table

BLKM	MEMORY BLOCK MOVE	⎍	⎍↑

SYMBOL:



OPERANDS:

	0	1	3	4	C	P	L
TOP	○	○	○	○		○	
MIDDLE	○			○		○	
BOTTOM					①		

①1~255

Description:

Memory contents of the table defined in the top node are copied to the table defined in the middle node in one scan. Table length is defined in the bottom node.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Function outputs can be used to determine whether the function block has been executed.

Node Description:

TOP: Starting address of the source memory block.

MIDDLE: Starting address of the target memory block.

BOTTOM: Block length.

Input Control:

I_1: When ⎍ (⎍↑) is presented, the function block is executed.

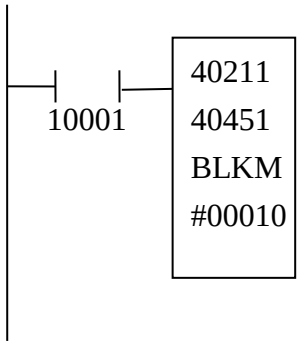
Function Output:

$O_1 = I_1$

$O_2 = 0$

$O_3 = 0$

【EXAMPLE】



【DESCRIPTION】

When 10001 receives a transition from ‘OFF’ to ‘ON’, then the entries of the first tables as defined in the top node (40211) are moved to the second table defined in the middle node(40451).

40211	01111
40212	02222
40213	03333
40214	04444
40215	05555
40216	06666
40217	07777
40218	08888
40219	09999
40220	00000

BEFORE

40451	00000
40452	00000
40453	00000
40454	00000
40455	00000
40456	00000
40457	00000
40458	00000
40459	00000
40460	00000
40461	00000

BEFORE

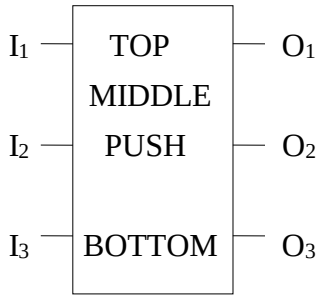
⇨

40451	01111
40452	02222
40453	03333
40454	04444
40455	05555
40456	06666
40457	07777
40458	08888
40459	09999
40460	00000

AFTER

PUSH	**PUSH FROM REGISTER TO STACK**	⎍	⎍↑

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP	○	○	○	○		○	
MIDDLE				○		○	
BOTTOM					①		

①1~255

Description:

This function pushes the content of a register (TOP NODE) to a predefined stack (MIDDLE NODE). Stack size is defined in the BOTTOM NODE.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Input control (I_2) is used to define the stack mode.

Function outputs can be used to determine whether the function block has been executed, and whether the stack is full.

Node Description:

TOP: Data to be pushed into stack.

MIDDLE: INDEX is defined in the first word. The starting of the stack is defined in the second word of the middle node. If the INDEX is equal to zero, then the INDEX is pointing to the top of the stack.

BOTTOM: Length of stack.

Input Control:

I_1: When ⎍ (⎍↑) is presented, the function block is executed. INDEX is incremented by 1.

I_2 =0, data is pushed into the stack at designated address. (LIFO). Used as STACK.

=1, data is put into the bottom of the stack , the original content at the top of the stack is moved to Next word. (FIFO). Used as QUEUE.

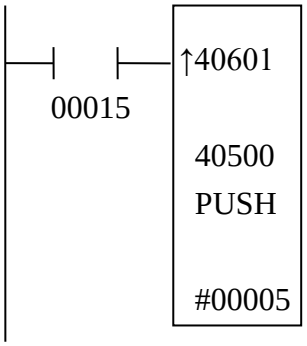
Function Output:

$O_1 = I_1$

O_2: =1, if stack is full, i.e. INDEX = stack length.

O_3=0

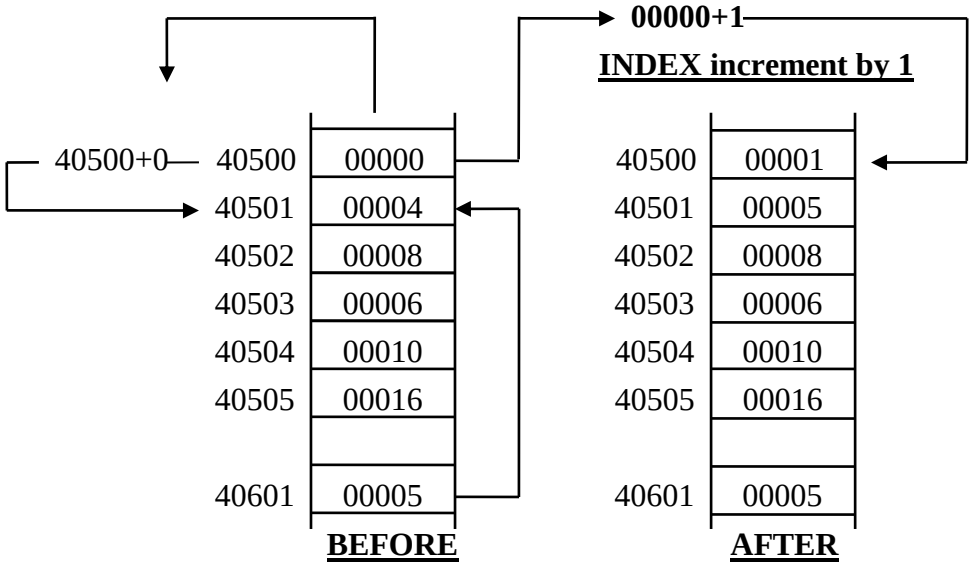
【EXAMPLE 1】



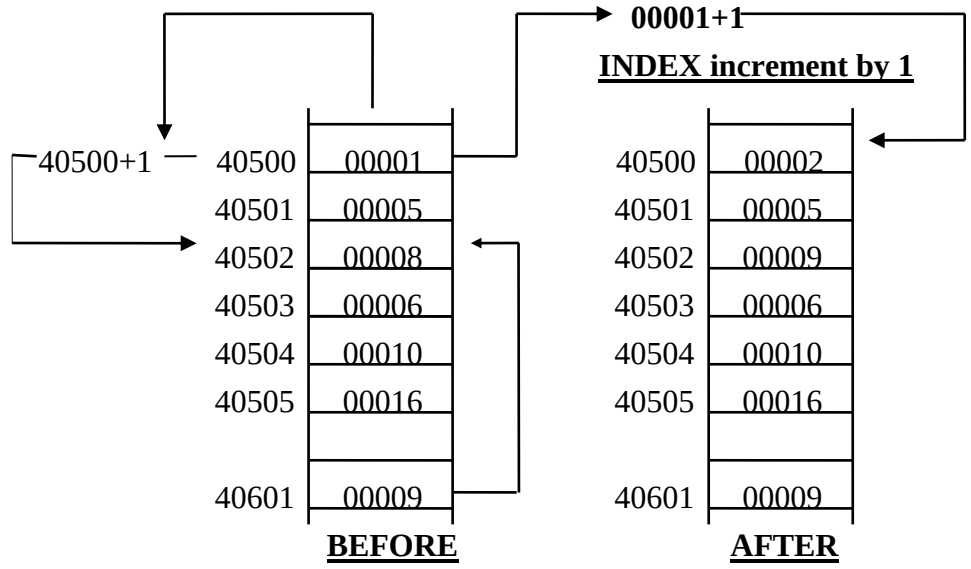
【DESCRIPTION】

Since I_2 = 'OFF', thus the operation mode is LIFO (Last In First Out).

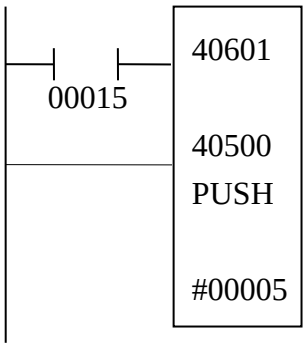
When I_1 receives an "OFF→ON" transition, the PUSH function is executed as follows:



For the next "OFF→ON" transition on I_1:

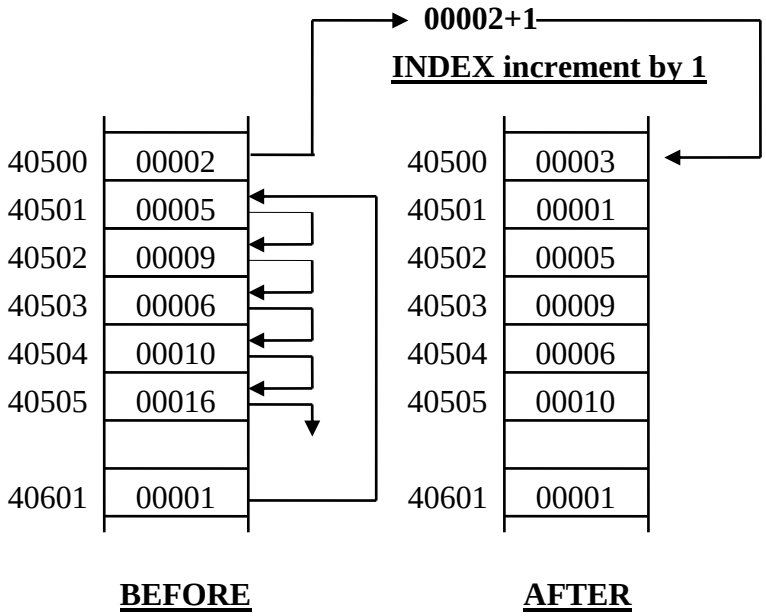


【EXAMPLE 2】

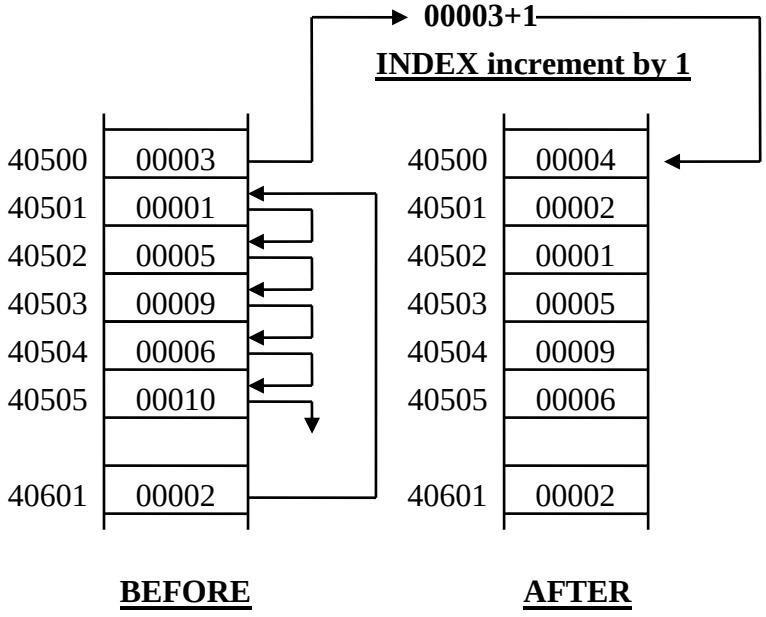


【**DESCRIPTION**】Since I_2 = 'ON', thus the operation mode is FIFO (First in first out).

When I_1 receives an "OFF→ON" transition, the PUSH function is executed as follows:

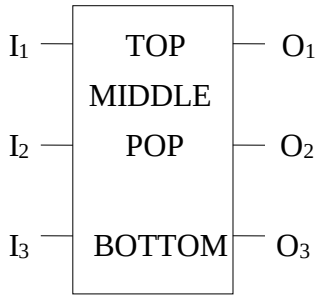


For the next "OFF→ON" transition on I_1:



POP	**POP FROM STACK TO REGISTER**	⎍	↑⎍

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP				○		○	
MIDDLE	○	○	○	○		○	
BOTTOM					①		

①1~255

Description:

This function moves the content of a stack defined in the top node to the register defined in the middle node.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Function outputs can be used to determine whether the function block has been executed, and whether the stack is empty.

Node Description:

TOP: INDEX is defined in the first word. The starting of the stack is defined in the second word of the middle node. If the INDEX is equal to zero, then the INDEX is pointing to the top of the stack.

MIDDLE: Data retrieved from stack.

BOTTOM: stack length.

Input Control:

I_1: When ⎍ (↑⎍) is presented, the function block is executed. INDEX is decreased by 1 first. Then the data is retrieved according to the INDEX.

Function Output:

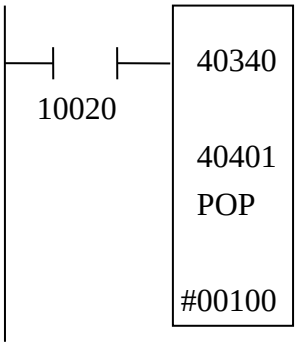
$O_1 = I_1$

$O_2 = 0$

O_3 = Stack status.

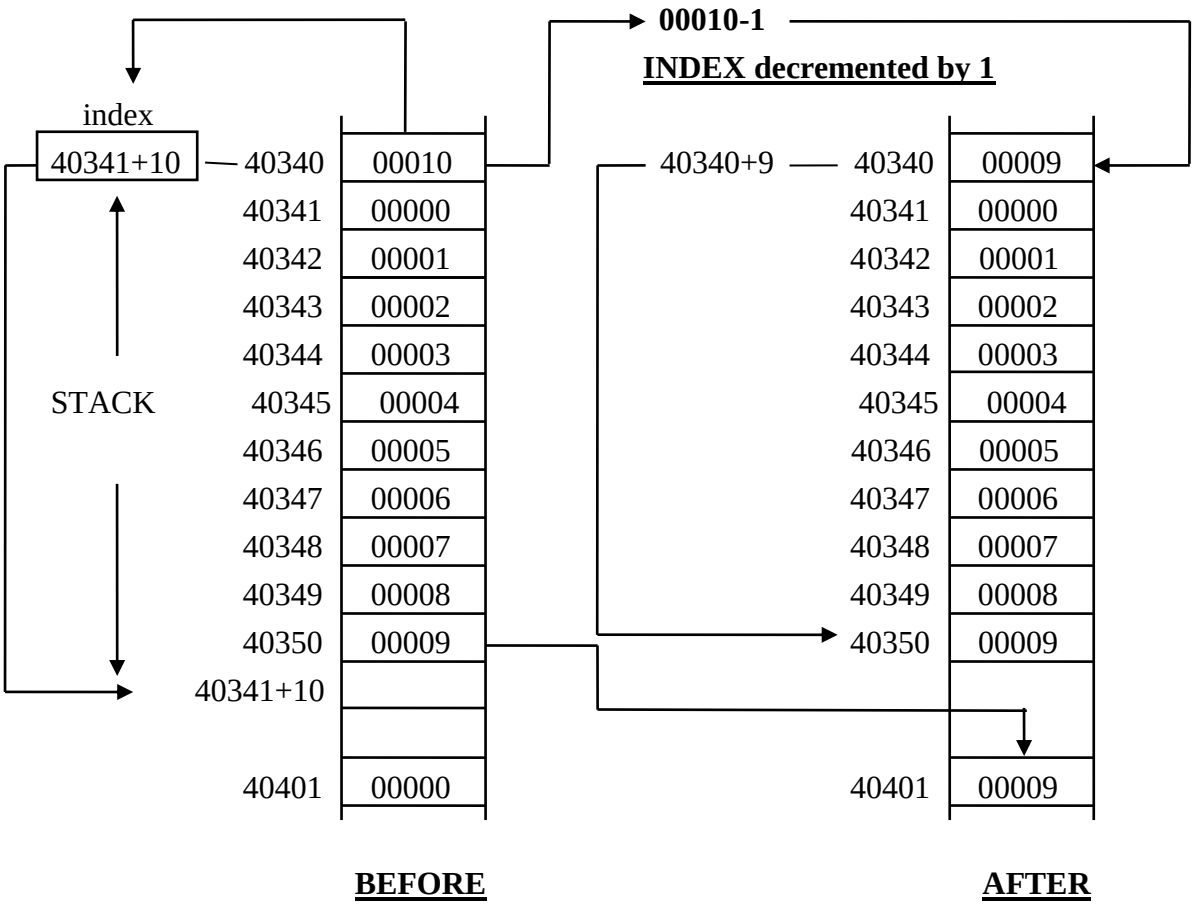
=1, if stack is empty, i.e. INDEX = 0.

【EXAMPLE】



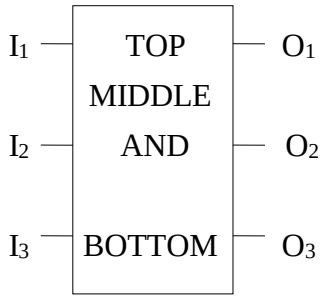
【DESCRIPTION】

When contact 10020=’ON’, the INDEX (40340) is decremented by 1; then the value pointed by the INDEX is retrieved and stored in the location pointed by the middle node. Through repeated conducting of contact 10020, the values in the stack are popped successively.



AND	AND OPERATION FOR ARRAYS	⎍	↑⎍

SYMBOL:



OPERANDS:

	0	1	3	4	C	P	L
TOP	○	○	○	○	①	○	
MIDDLE	○			○		○	
BOTTOM					↑		

①0~65536

↑1~255

Description:

The contents of top and middle nodes are ANDed, and the result is stored in the middle node.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Function outputs can be used to determine whether the function block has been executed.

Remark: When the content of the top node is a constant, the constant and the contents of middle node are ANDed and the result is stored in the middle node.

Node Description:

TOP: Source Array 1, or constant.

MIDDLE: Source Array 2, Resultant Array (after processing).

BOTTOM: Length of Array.

Input Control:

I_1: When ⎍ (↑⎍) is presented, the function block is executed.

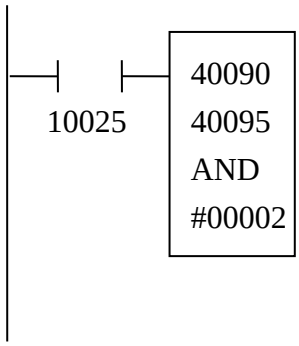
Function Output:

$O_1 = I_1$

$O_2 = 0$

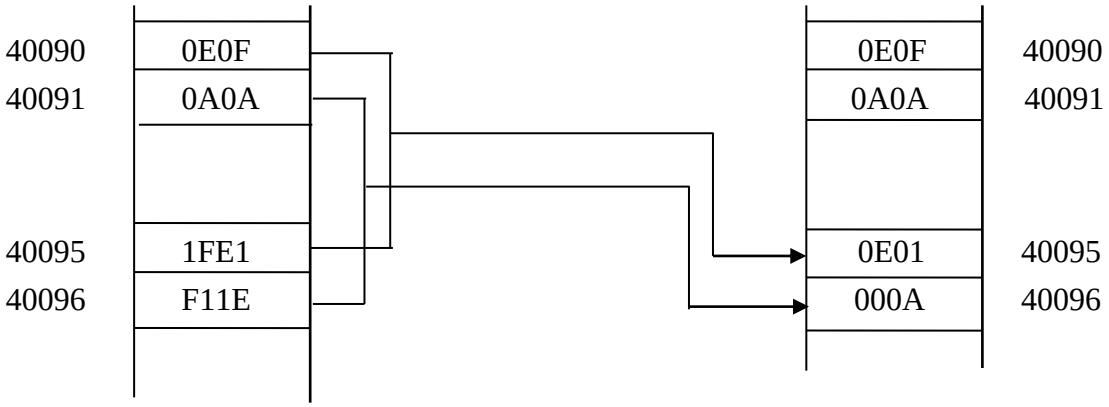
$O_3 = 0$

【EXAMPLE 1】

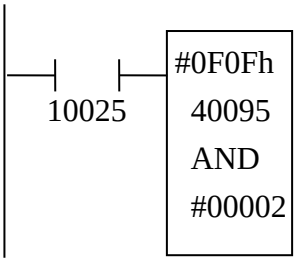


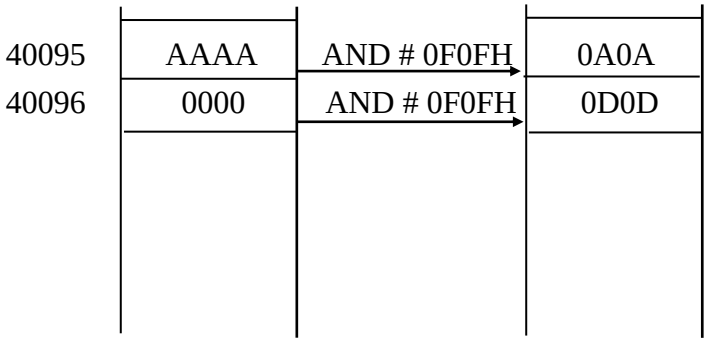
【DESCRIPTION】

When contact 10025 is energized, the contents of registers 40090 and 40095 are ANDed, and the result is returned to register 40095.



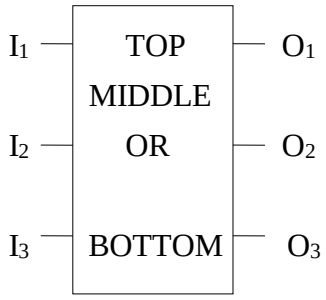
【EXAMPLE 2】





OR	**OR OPERATION FOR ARRAYS**	⎍	↑⎍

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP	○	○	○	○	①	○	
MIDDLE	○			○		○	
BOTTOM					↑		

①0~65535

↑1~255

Description:

The contents of top and middle nodes are ORed, and the result is stored in the middle node.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Function outputs can be used to determine whether the function block has been executed.

Remark: When the content of the top node is a constant, the constant and the contents of middle node are ORed and the result is stored in the middle node.

Node Description:

TOP: Source Array 1, or constant.

MIDDLE: Source Array 2, Resultant Array (after processing).

BOTTOM: Length of Array.

Input Control:

I_1: When ⎍ (↑⎍) is presented, the function block is executed.

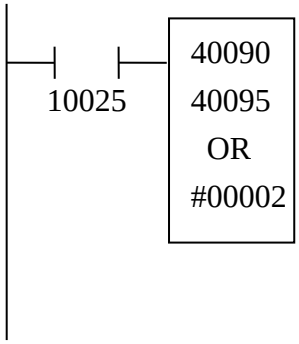
Function Output:

$O_1 = I_1$

$O_2 = 0$

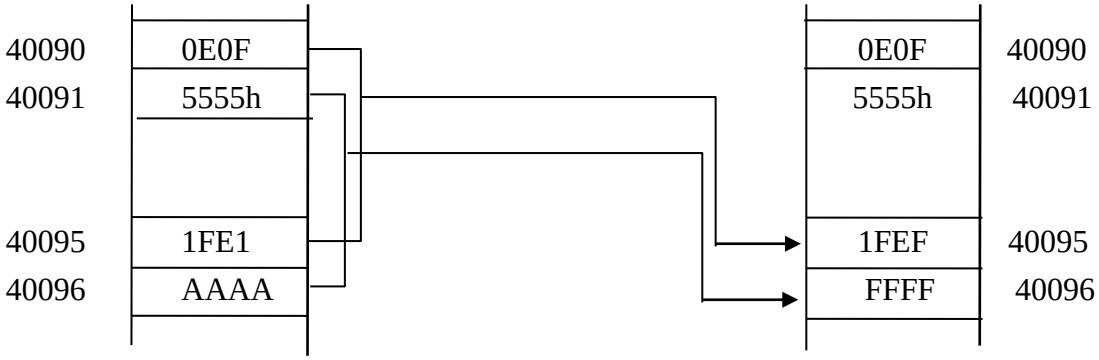
$O_3 = 0$

【EXAMPLE 1】

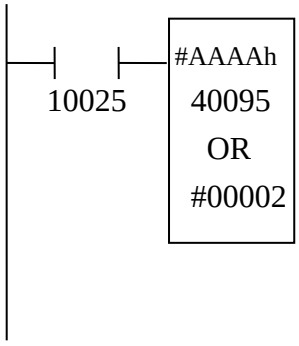


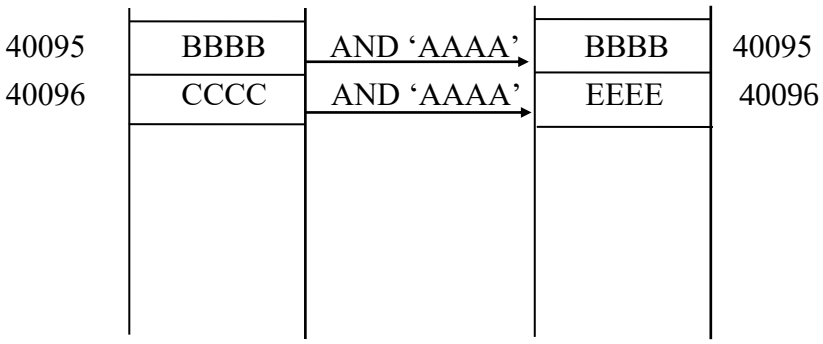
【DESCRIPTION】

When contact 10025 is energized, the contents of registers 40090 and 40095 are ORed, and the result is returned to register 40095.



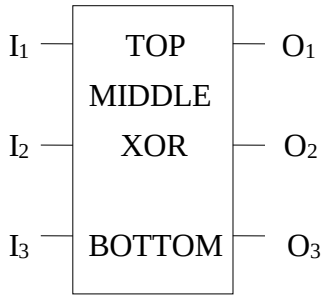
【EXAMPLE 2】





XOR	XOR OPERATION FOR ARRAYS	⎍	↑⎍

SYMBOL:



OPERANDS:

	0	1	3	4	C	P	L
TOP	○	○	○	○	①	○	
MIDDLE	○			○		○	
BOTTOM					↑		

① 0~65535

↑ 1~255

Description:

The contents of top and middle nodes are XORed, and the result is stored in the middle node.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Function outputs can be used to determine whether the function block has been executed.

Remark: When the content of the top node is a constant, the constant and the contents of middle node are XORed and the result is stored in the middle node.

Node Description:

TOP: Source Array 1 or constant.

MIDDLE: Source Array 2, Resultant Array (after processing).

BOTTOM: Length of Array.

Input Control:

I_1: When ⎍ (↑⎍) is presented, the function block is executed.

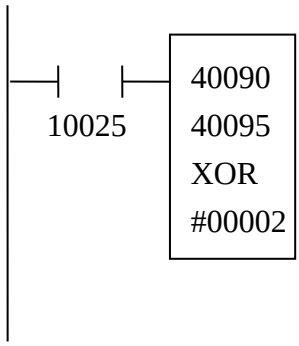
Function Output:

$O_1 = I_1$

$O_2 = 0$

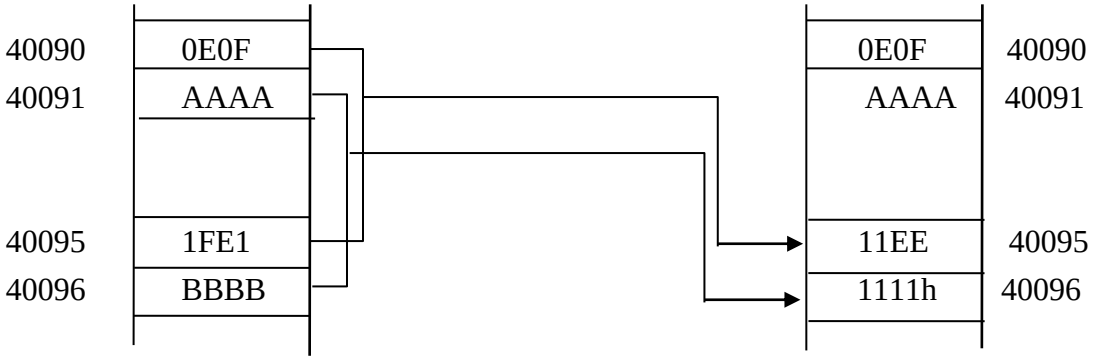
$O_3 = 0$

【EXAMPLE 1】

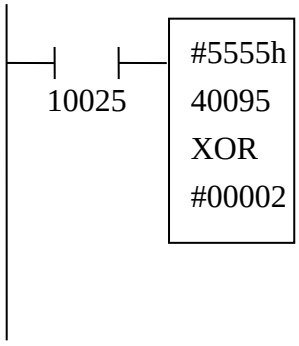


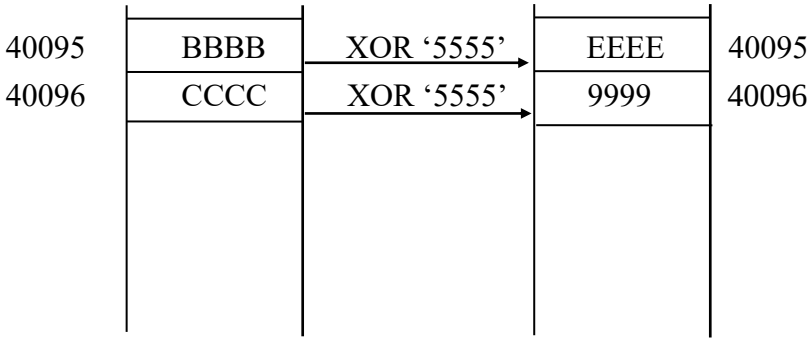
【DESCRIPTION】

When contact 10025 is energized, the contents of registers 40090 and 40095 are XORed, and the result is returned to register 40095.



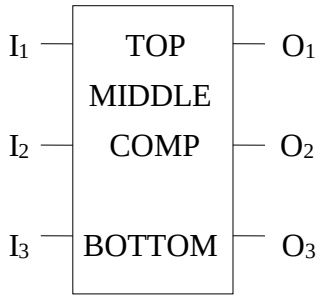
【EXAMPLE 2】





COMP	1'S COMPLEMENT FOR ARRAYS	⎍	↑⎍

SYMBOL:



OPERANDS:

	0	1	3	4	C	P	L
TOP	○	○	○	○		○	
MIDDLE	○			○		○	
BOTTOM					①		

①1~255

Description:

1's complement is obtained for the content of the top node, and the result is stored in the middle node.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Function outputs can be used to determine whether the function block has been executed.

Node Description:

TOP: Source Array.

MIDDLE: Resultant Array.

BOTTOM: Length of Array.

Input Control:

I_1: When ⎍ (↑⎍) is presented, the function block is executed.

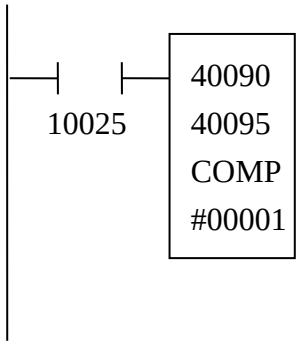
Function Output:

$O_1 = I_1$

$O_2 = 0$

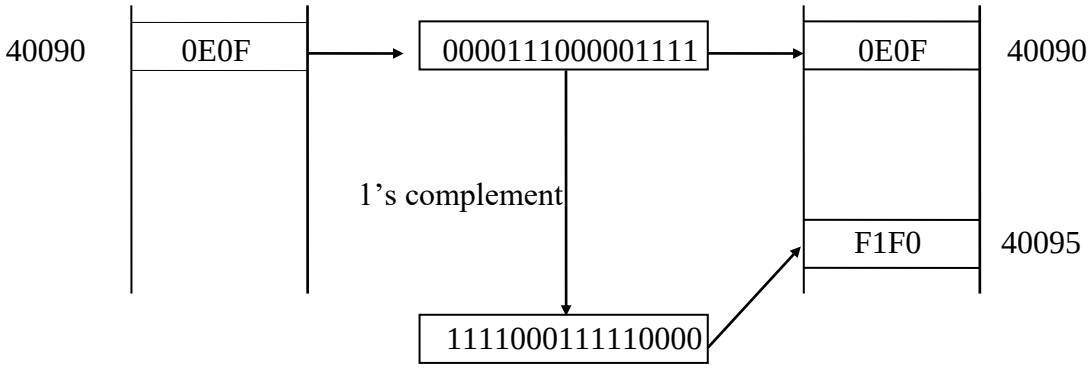
$O_3 = 0$

【EXAMPLE】



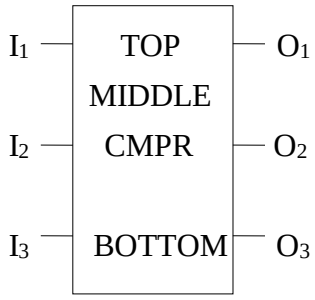
【DESCRIPTION】

When contact 10025 is energized, 1's complement is obtained for the content of register 40090, and the result is returned to register 40095.



CMPR	**BIT COMPARISON BETWEEN TWO MATRIX**	⎍	⎍↑

SYMBOL:



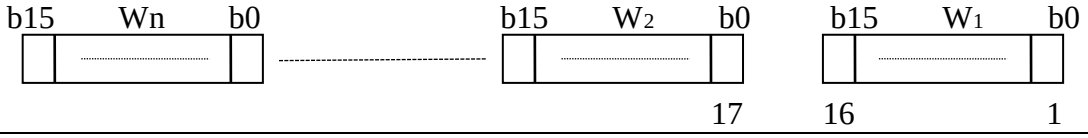
OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP	○	○	○	○		○	
MIDDLE				○		○	
BOTTOM					①		

①1~255

Description:

This function compares the matrix pointed by the top and middle nodes. If a difference is found between the corresponding matrix locations, then the index of that element is stored in the middle node. Input control (I_1) is used to determine whether this function block is to be executed or not. Input control (I_2) is used to indicate the position where comparison is started. Function outputs can be used to determine whether the function block has been executed and whether those tables are different or not.



Node Description:

TOP: matrix 1.

MIDDLE: Index and matrix 2. INDEX is stored in the first word. Matrix 2 is stored starting from the second word. If the value of the INDEX is zero after searching, it represents that the contents of the two matrixes are identical.

BOTTOM: Length of matrix (word).

Input Control:

I_1: When ⎍ (⎍↑) is presented, the function block is executed. When a difference is found, the INDEX points to the position where the difference is found.

I_2: Start position of the comparison.

=0, start from the position pointed to by the INDEX.

=1, start from the first position

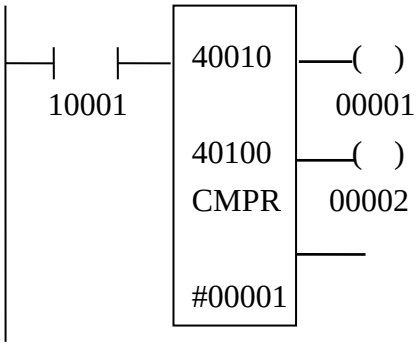
Function Output:

$O_1 = I_1$

O_2=1, if a difference is found.

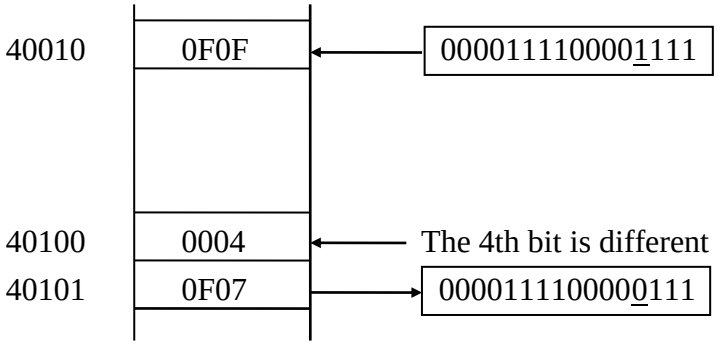
O_3=0

【EXAMPLE】



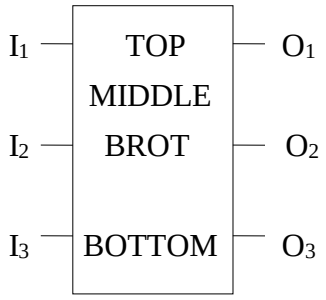
【DESCRIPTION】

When contact 10001 is energized, the matrix starting from 40010 is compared against the matrix starting from 40100. Since the fourth bit is different, then the index of that location is stored in the middle node and coil 00002 is energized.



BROT — BIT ROTATE/SHIFT FOR MATRIX

SYMBOL:



OPERANDS:

	0	1	3	4	C	P	L
TOP	○	○	○	○		○	
MIDDLE	○			○		○	
BOTTOM					①		

①1~255

Description:

Using the bit as a unit, this function performs array rotate or shift. The result is stored in the middle node.
The matrix is defined in the top node.
Table length is defined in the bottom node.
Input control (I_1) is used to determine whether this function block is to be executed or not.
Input control (I_2) is used to define the direction.
Input control (I_3) is used to define the mode.
Function outputs can be used to determine whether the function block has been executed.

Node Description:

TOP: Source matrix.

MIDDLE: Target matrix.

BOTTOM: Length of matrix (word).

Input Control:

I_1: Execution control.

When ⎍ (⎍↑) is presented, rotate/shift operation is performed one bit per scan.

I_2: Direction

=0, Left.

=1, Right.

I_3: Mode.

=0, Shift.

=1, Rotate.

Function Output:

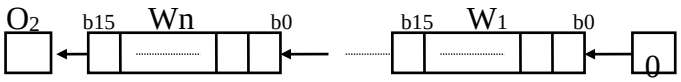
$O_1 = I_1$

O_2: Bit shifted into this position.

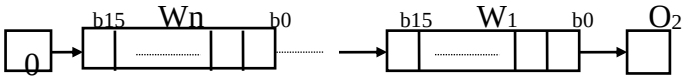
O_3=0

SHIFT MODE

LEFT

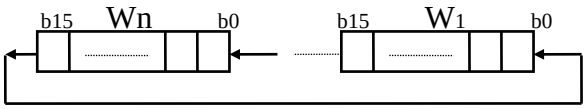


RIGHT

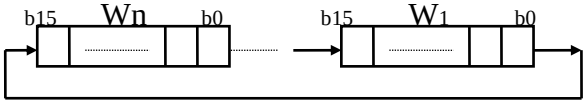


ROTATE MODE

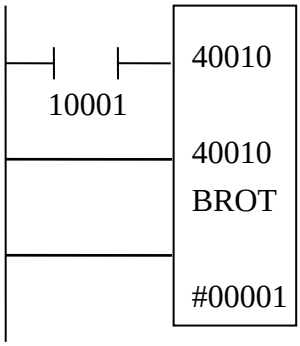
LEFT



RIGHT

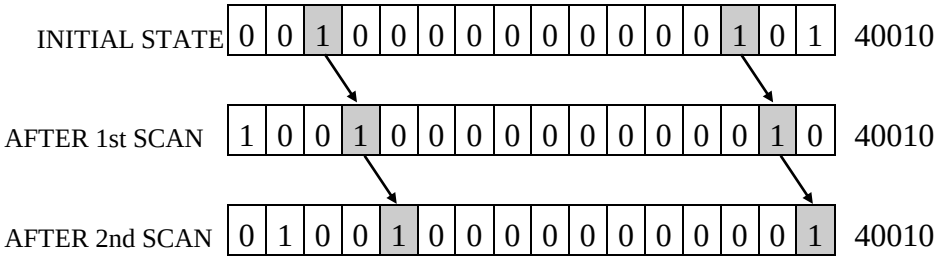


【EXAMPLE】



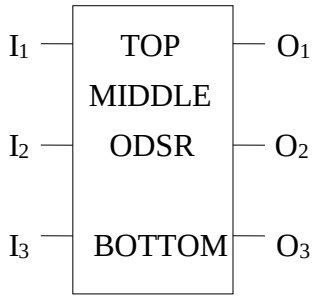
【DESCRIPTION】

When contact 10001 receives a transition from ‘OFF’ to ‘ON’ and I_2=I_3=1, then a right rotate operation is performed.



ODSR	**NIBBLE ROTATE/SHIFT FOR MATRIX**	⎍	⎍↑

SYMBOL:



OPERANDS:

	0	1	3	4	C	P	L
TOP	○	○	○	○		○	
MIDDLE	○			○		○	
BOTTOM					①		

①1~255

Description:

Using the nibble as a unit, this function performs array rotate or shift. The result is stored in the middle node. The matrix is defined in the top node. Table length is defined in the bottom node.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Input control (I_2) is used to define the direction.

Input control (I_3) is used to define the mode.

Function outputs can be used to determine whether the function block has been executed.

Node Description:

TOP: Source matrix

MIDDLE: Target matrix

BOTTOM: Length of matrix (word)

Input Control:

I_1: Execution control.

When ⎍ (⎍↑) is presented,

rotate/shift operation is performed one nibble per scan.

I_2: Direction

=0, Left

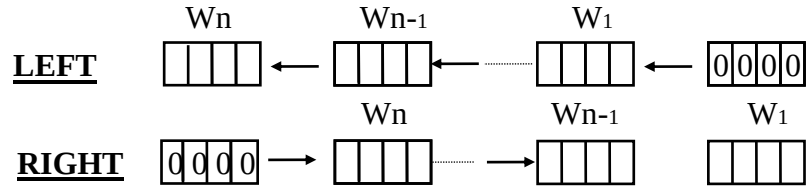
=1, Right

I_3: Mode.

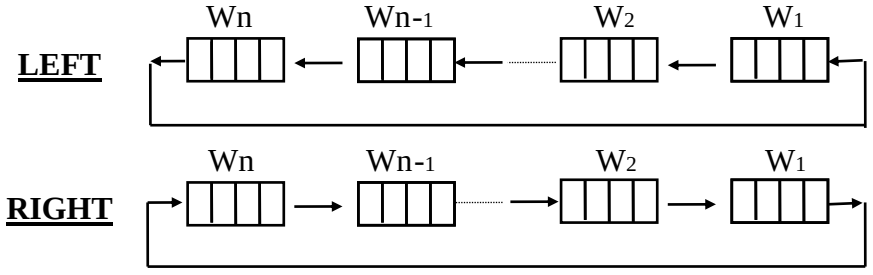
=0, Shift.

=1, Rotate.

SHIFT MODE



ROTATE MODE



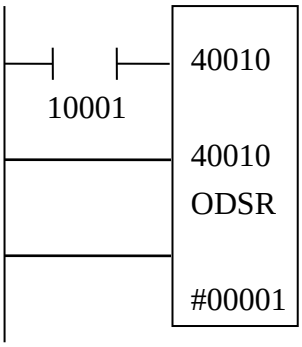
Function Output:

$O_1 = I_1$

$O_2 = 0$

$O_3 = 0$

【EXAMPLE】



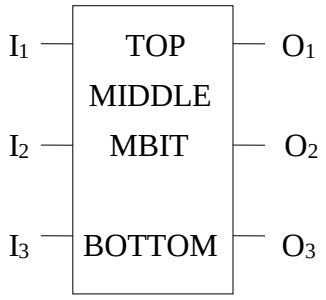
【DESCRIPTION】

When contact 10001 receives a transition from ‘OFF’ to ‘ON’ and I_2=I_3=1, then a right rotate operation is performed.

Initial State	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	40010
After 1st Scan	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	40010
After 2nd Scan	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	40010

MBIT	**MODIFY BIT MATRIX**	⎍	⎍↑

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP			○	○	①	○	
MIDDLE	○			○		○	
BOTTOM					②		

①1~65535

②1~255

Description:

This function is used to SET or CLEAR a certain bit in a matrix. Bit location is defined in the top node. Array to be modified is defined in the middle node. Array length (WORD) is defined in the bottom node.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Input control (I_2) is used to define the action (SET or Clear).

Input control (I_3) is used to define the behavior of the INDEX.

Function outputs can be used to determine whether the function block has been executed, and the status of the INDEX.

Node Description:

TOP: INDEX (pointing to the bit to be modified). INDEX=1 ➔ The first bit.

MIDDLE: Source matrix.

BOTTOM: Matrix length (word).

Input Control:

I_1: When ⎍ (⎍↑) is presented, the function block is executed.

I_2: Action.

=0, bit clear

=1, bit set

I_3: INDEX control. If I_3=1 and TOP = 4xxxx, then the INDEX is incremented by 1 after execution.

Function Output:

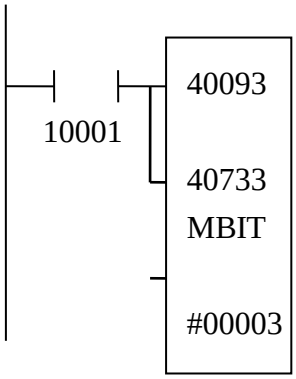
$O_1 = I_1$

$O_2 = I_2$

O_3: Status of the INDEX.

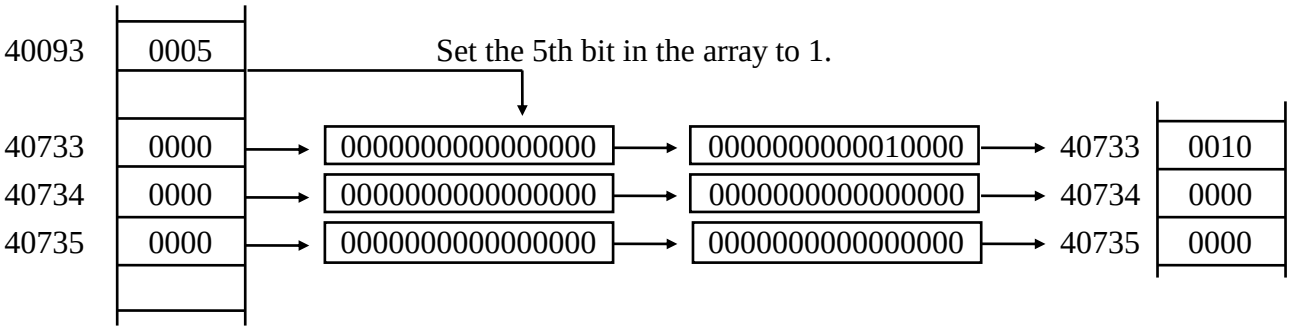
=1, if INDEX is larger than the value of the BOTTOM node times 16.

【EXAMPLE】



【DESCRIPTION】

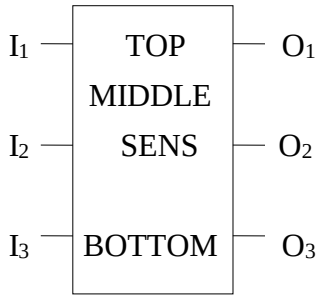
00005 is stored in the top node (40093). When contact 10001 is energized, and I_2=1, then the 5th bit of the matrix starting from 40733 to 40735 is set to 1.



BEFORE **AFTER**

SENS	**SENSING OF A BIT IN MATRIX**	⎍	↑⎍

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP			○	○	①	○	
MIDDLE	○			○		○	
BOTTOM					②		

①1~255

②1~65535

Description:

This function is used to sense a certain bit in a matrix. Bit location is defined in the top node. Matrix to be modified is defined in the middle node. Array length is defined in the bottom node.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Input control (I_2) is used to define the behavior of the INDEX.

Input control (I_3) is used to reset the INDEX.

Function outputs can be used to determine whether the function block has been executed, and the status of the INDEX.

Node Description:

TOP: INDEX (pointing to the bit to be checked). INDEX=1 ➔ The first bit.

MIDDLE: Source matrix.

BOTTOM: Matrix length (word).

Input Control:

I_1: When ⎍ (↑⎍) is presented, the function block is executed.

I_2: INDEX control. If I_3=1 and the top node is 4XXXX, then the INDEX is incremented by 1 after execution.

I_3: INDEX control.

=1, Reset INDEX.

Function Output:

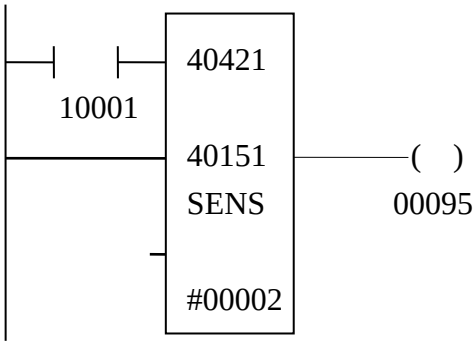
$O_1 = I_1$

O_2 = The state of the bit sensed.

O_3: Status of the INDEX.

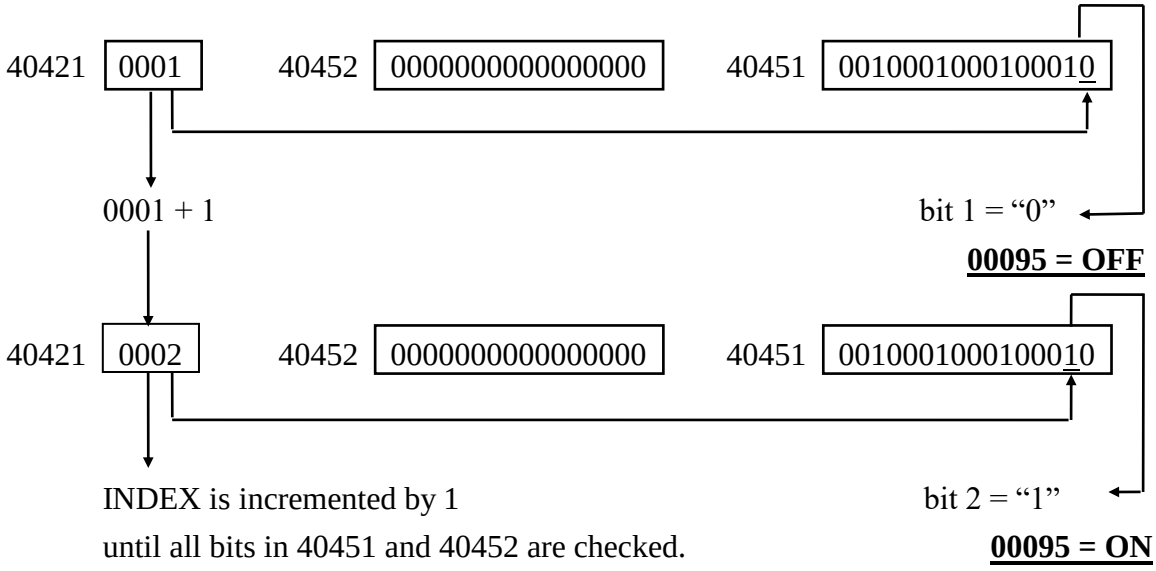
=1, if INDEX is equal to zero or larger than the value of the BOTTOM node times 16.

【EXAMPLE】



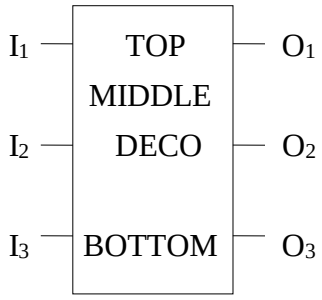
【DESCRIPTION】

When contact 10001 is energized and I_2=1, the state of coil 00095 is set to that of the bit checked. Since (40421)=0001, The 1st bit is checked. And since the bottom node is #00002, thus the registers 40151～40152 are checked.



DECO	**DECODER (4->16)**	⎍	⎍↑

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP	○	○	○	○		○	
MIDDLE	○			○		○	
BOTTOM					①		

①0~3

Description:

This function is a 4 bit to 16 bit decoder. The top node contains 4 sets of 4-bit data. The set of data to be decoded is defined in the bottom node.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Function outputs can be used to determine whether the function block has been executed.

Node Description:

TOP: Input to decoder, only four bits (nibble) are used.

MIDDLE: Decoder output.

BOTTOM: Determine which nibble in the TOP node is to be decoded.

Input Control:

I_1: When ⎍ (⎍↑) is presented, the function block is executed.

Function Output:

$O_1 = I_1$

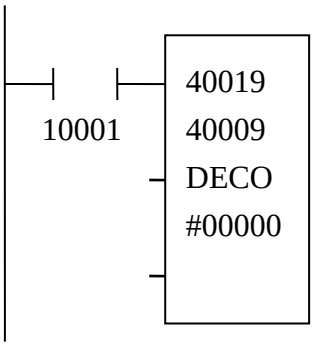
$O_2 = 0$

$O_3 = 0$

TRUTH TABLE

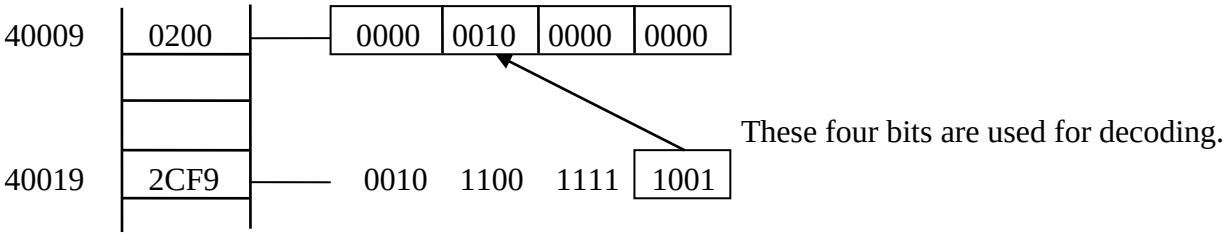
INPUTS				OUTPUT WORD														
3	**2**	**1**	**0**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

【EXAMPLE】



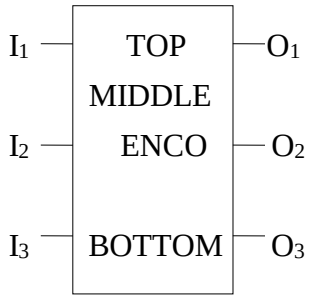
【DESCRIPTION】

Let register 40019 = 2CF9h = 0010 1100 1111 1001B, and #00000 is defined in the bottom node. #00000 indicates that the first set of 4-bit data is to be used as the decoder function input. The first 4-bit set in this example is 1001B, which is equal to 9. Therefore, the 10th bit (0 means the 1st bit and 15 means the 16th bit) in the middle node (40009) will be set after contact 10001 is energized.



ENCO	**ENCODER (16->4)**	⎍	↑⎍

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP	○	○	○	○		○	
MIDDLE	○			○		○	
BOTTOM					①		

①0~3

Description:

This function is a 16-bit to 4-bit encoder. The top node contains the data to be encoded. The bottom node indicates the 4-bit set to be used to store the encoded result, and the encoded data is stored in the middle node.

NOTE: If more than one bit is set in the top node, then the bit which is closer to the most significant bit will be used for encoding.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Function outputs can be used to determine whether the function block has been executed.

Node Description:

TOP: Input to encoder.

MIDDLE: Encoder result.

BOTTOM: Nibble (0~3) where the encoder result is stored.

Input Control:

I_1: When ⎍ (↑⎍) is presented, the function block is executed.

Function Output:

$O_1 = I_1$

O_2 = indicator

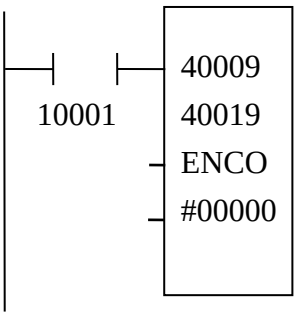
=1, if the valued stored in the TOP node is zero.

O_3=0

TRUTH TABLE

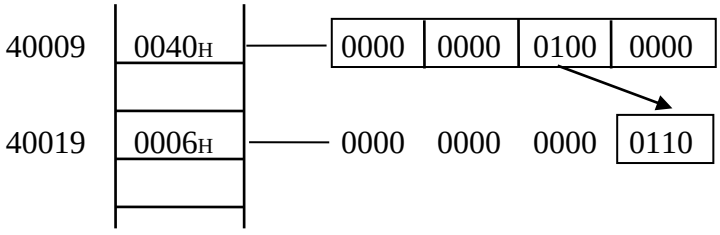
INPUT WORD																OUTPUTS			
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**3**	**2**	**1**	**0**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	1	0	1
0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	1	1	0
0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	1	1	1
0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	1	0	0	1
0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	1	0	1	0
0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1
0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1
0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

【EXAMPLE】



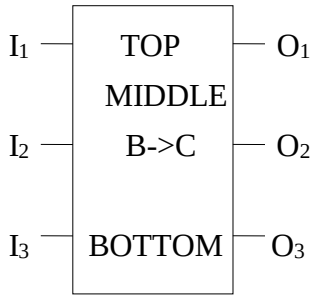
【DESCRIPTION】

Let 40009 = 0040h = 0000 0000 0100 0000h, and #00000 is given in the bottom node. Since MSB is the 16th bit and LSB is the first bit in a 16-bit register, thus, the 7th bit is encoded to 6; and 6 is equal to 0110B. This 0110B 4-bit set is moved to the 1st 4-bit set of register 40019 as defined in the bottom node (#00000)



B->C	**BINARY TO BCD CONVERTION**	⎍	⎍↑

SYMBOL:



OPERANDS:

	0	1	3	4	C	P	L
TOP	○	○	○	○		○	
MIDDLE	○			○		○	
BOTTOM					①		

①1~2

Description:

This function performs binary to binary-coded-decimal conversion. The data to be converted is defined in the top node, and the converted data is stored in the middle node. The bottom node defines the conversion type (word or long word).

Input control (I_1) is used to determine whether this function block is to be executed or not.

Function outputs can be used to determine whether the function block has been executed and whether the result is correct or not.

Node Description:

TOP: data set (binary) to be converted, must be＜=9999(decimal).

MIDDLE: Conversion result.

BOTTOM:1.Word conversion.

2.Long word conversion.

Input Control:

I_1: When ⎍ (⎍↑) is presented, the function block is executed.

Function Output:

$O_1 = I_1$

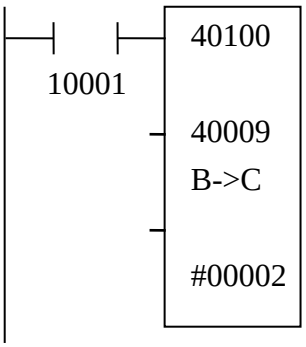
O_2= indicator

= 1, if the valued stored in the TOP node is ＞9999(decimal) when the value of bottom node is ‘1’.

= 1, if the valued stored in the TOP node is ＞99999999(decimal) when the value of bottom node is ‘2’.

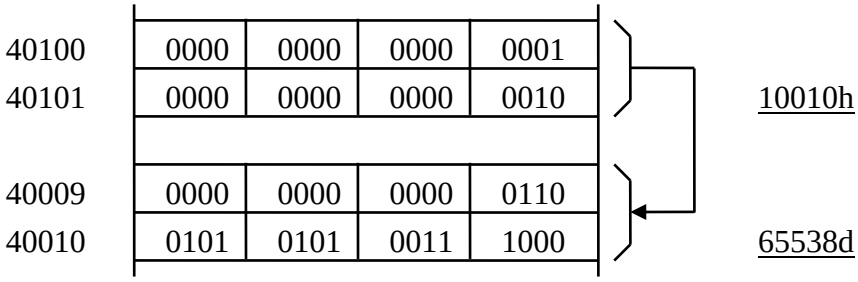
$O_3 = 0$

【EXAMPLE】



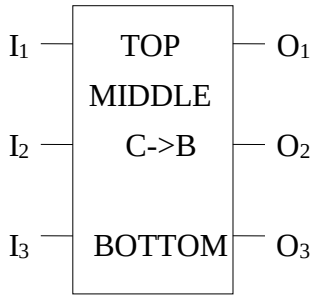
【DESCRIPTION】

Let register (40100)= 0001h, and register (40101)= 0002h. When contact 10001 is energized, since #00002 is defined as a long word conversion, then the top node long word (40100) 10010h=65538d are converted and stored in registers 40009 and 40010.



C->B	**BCD TO BINARY CONVERTION**		

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP	○	○	○	○		○	
MIDDLE	○			○		○	
BOTTOM					①		

①1~2

Description:

This function performs binary-coded-decimal to binary conversion. The data to be converted is defined in the top node, and the converted data is stored in the middle node. The bottom node defines the conversion type (word or long word).

Input control (I_1) is used to determine whether this function block is to be executed or not.

Function outputs can be used to determine whether the function block has been executed and whether the result is correct or not.

Node Description:

TOP: data set (BCD) to be converted.

MIDDLE: Conversion result.

BOTTOM:1. Word

2. Long word

Input Control:

I_1: When ⎍ (⍐) is presented, the function block is executed.

Function Output:

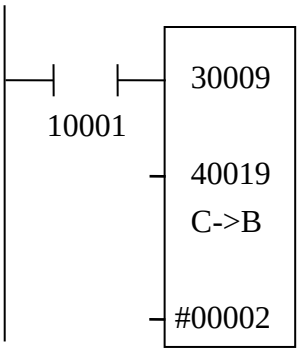
$O_1 = I_1$

O_2 = indicator

= 1, if the valued stored in the TOP node is not in BCD format.

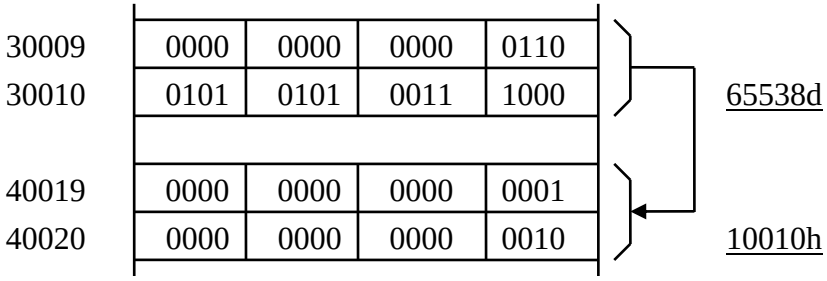
O_3=0

【EXAMPLE】



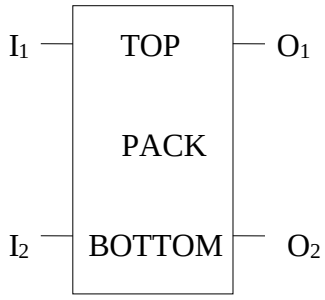
【DESCRIPTION】

Science this is a long word conversion (bottom node is #00002). When contact 100d is energized, the top node long word (30009) = 65538d is converted to 10010h and stored in middle node (40019), (40020).
Let register (30009) = 8888d = 22B8h, and register (30010) = 7777d = 1E61h.
When contact 10001 is energized, since #00002 is defined to the bottom node, then the converted BCD codes are stored in registers 40019 and 40020.



PACK	**WORD PACK/UNPACK**		

SYMBOL:



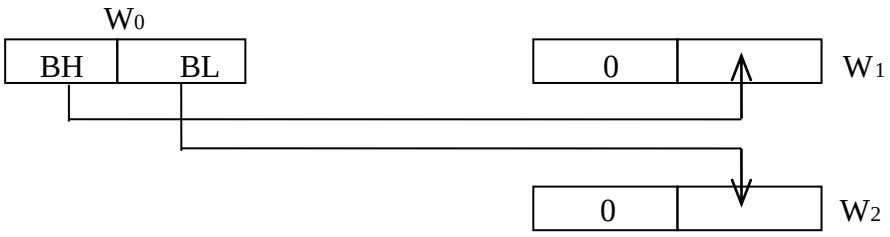
OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP	○	○	○	○		○	
BOTTOM	○			○		○	

Description:

Depending on I_2, this function splits the contents of the top node into two bytes, and stores them in the middle node; or, takes two LOW BYTEs from the top node, concatenate to form a new 16-bit word and stores it in the middle node.

Input control (I_1) is used to determine whether this function block is to be executed or not.



Node Description:

TOP: Data to be processed.

BOTTOM: Process result.

Input Control:

I_1: When ⎍ (⎍↑) is presented, the function block is executed.

I_2: Pack/Unpack

=0, Unpack (splits the source data into two words and stores them in the bottom node).

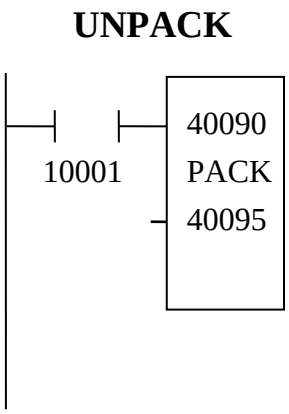
=1, Pack (concatenate the lower bytes of two words and stores the word in the bottom node).

Function Output:

$O_1 = I_1$

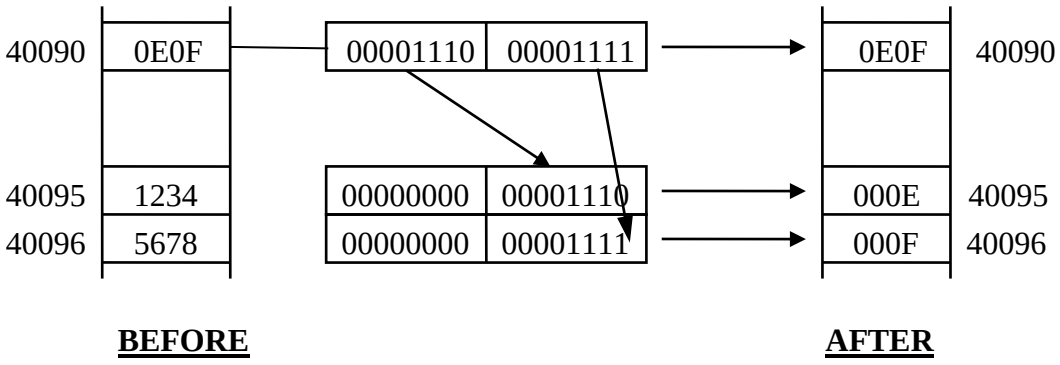
$O_2 = 0$

【EXAMPLE 1】



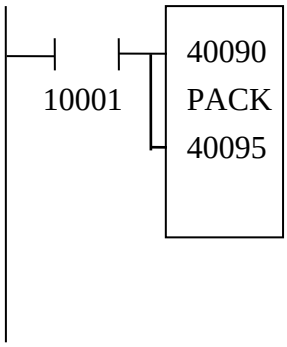
【DESCRIPTION】

When contact 10001 is energized, the content of the top node (40090) is split into two bytes which are stored in the middle node (40095 and 40096)



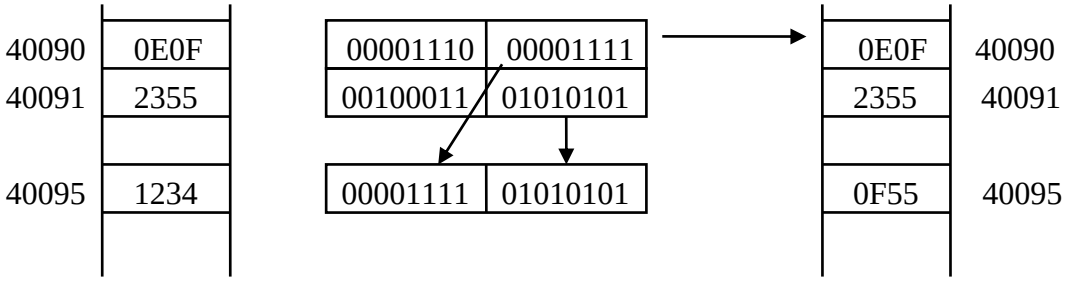
【EXAMPLE 2】

PACK



【DESCRIPTION】

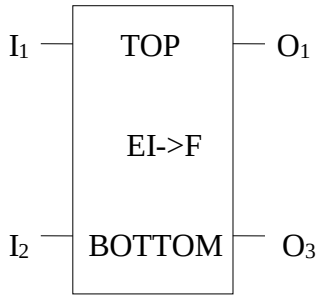
When contact 10001 is energized, two LOW BYTEs taken from the top node (40090 and 40091), are concatenated to form a new 16-bit word which is stored in the middle node(40095).



BEFORE **AFTER**

EI->F	**INTEGER TO FLOATING POINT CONVERSION**	⎍	⎍↑

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP			○	○		○	
BOTTOM				○		○	

Description:

This function converts an integer stored in the top node to a floating point number and stores in the registers defined in the middle node.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Function outputs can be used to determine whether the function block has been executed.

Node Description:

TOP: Data to be converted, integer (16 bits).

BOTTOM: Conversion results (32 bits).

Input Control:

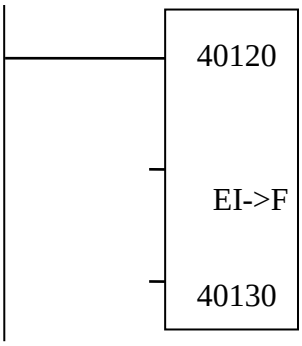
I_1: When ⎍ (⎍↑) is presented, the function block is executed.

Function Output:

$O_1 = I_1$

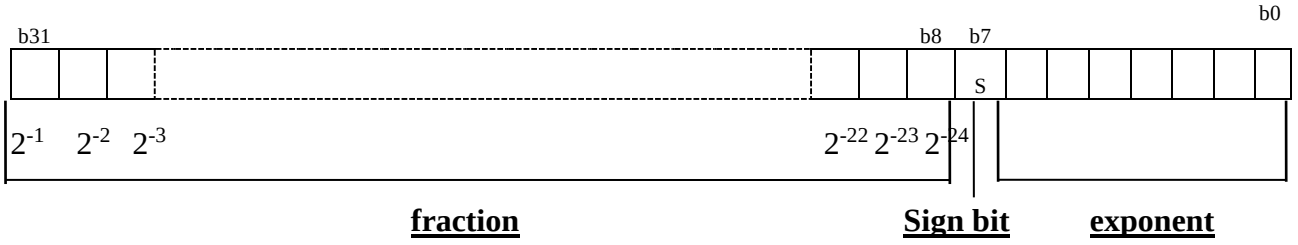
$O_2 = 0$

【EXAMPLE】



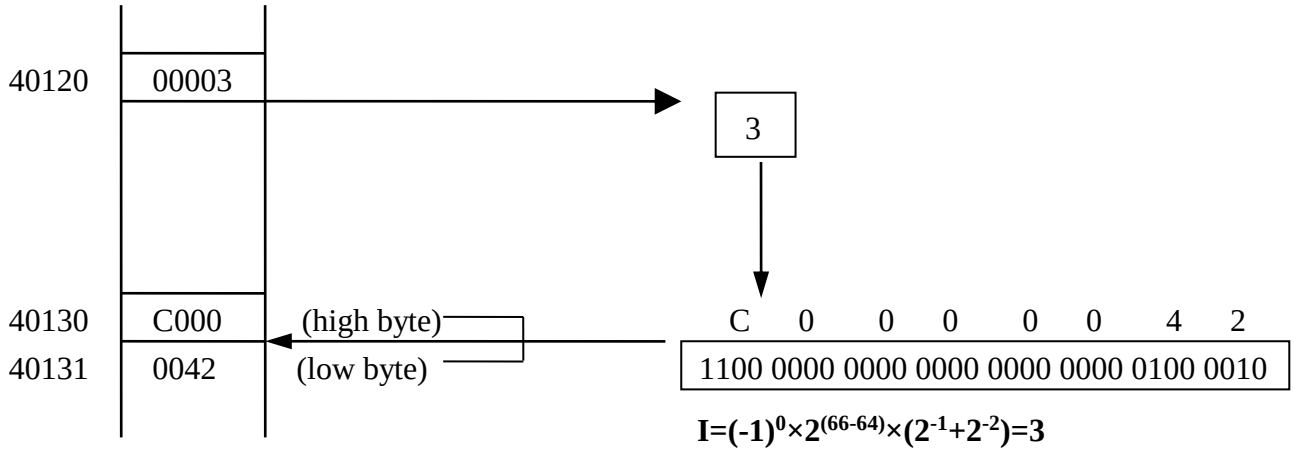
【DESCRIPTION】

Converts the integer stored in the top node (40120) to a floating number and stores in the middle node (40130 & 40131). A floating point number is represented by two words: bit0~bit6 represent the exponent, bit7is the sign bit (0: positive, 1:negative), and bit8~bit31 represent the fraction.



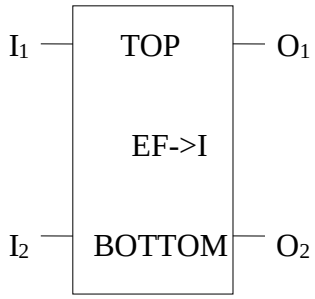
Formula:

$I = (-1)^S \times 2^{(E-64)} \times Fr$ **S= sign bit, E=exponent**



EF->I	**FLOATING POINT TO INTEGER CONVERSION**	⎍	⎍↑

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP				○		○	
BOTTOM				○		○	

Description:

This function converts a floating point number stored in the top node to an integer and stores in the registers defined in the middle node.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Function outputs can be used to determine whether the function block has been executed.

Node Description:

TOP: Data to be converted, **floating point number** (32 bits).

BOTTOM: Conversion results. **Integer** (16 bits).

Input Control:

I_1: When ⎍ (⎍↑) is presented, the function block is executed.

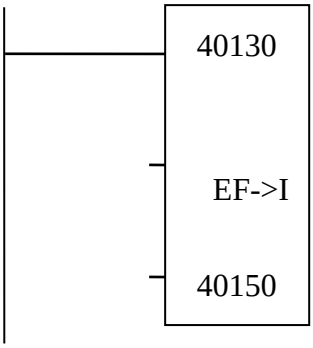
Function Output:

$O_1 = I_1$

O_2: Error output

=1, (overflow or < 0)

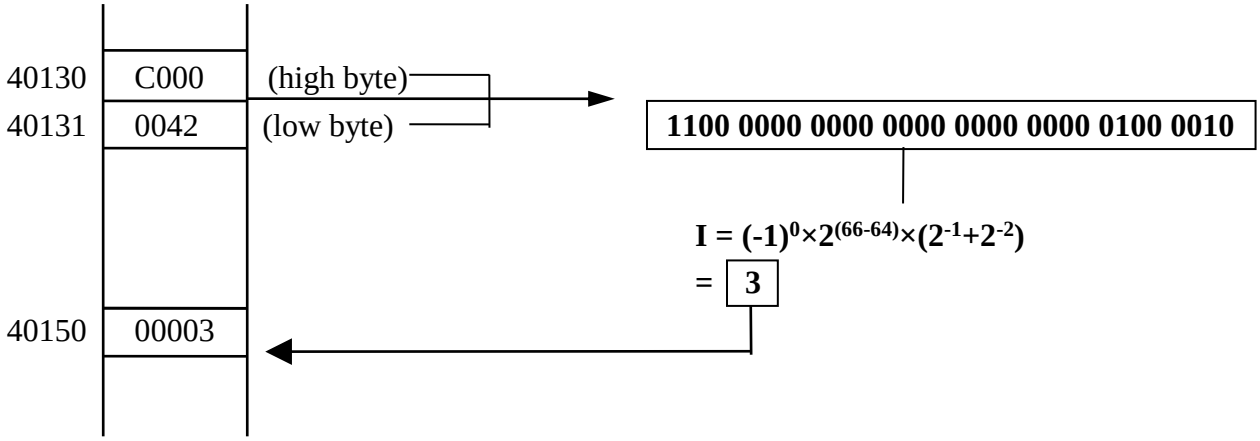
【EXAMPLE】



【DESCRIPTION】

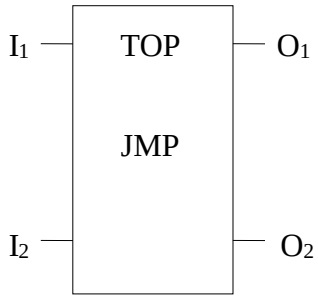
For a floating point number C000, 0042 stored in registers 40130 and 40131 respectively, the conversion returns 0003 stored in 40150.

$$I=(-1)^0\times2^{(66-64)}\times(2^{-1}+2^{-2})=3$$



JMP	**JUMP**	⎍	⎍↑

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP							①

①L1~L150

Description:

This instruction is used to instruct the program to JUMP to the other portion of the program with matched label number and EOJ instruction.

Input control (I_1) is used to determine whether this instruction is to be executed or not.

Function outputs can be used to determine whether the instruction has been executed.

Node Description:

TOP: Label where JUMP is intended.

Input Control:

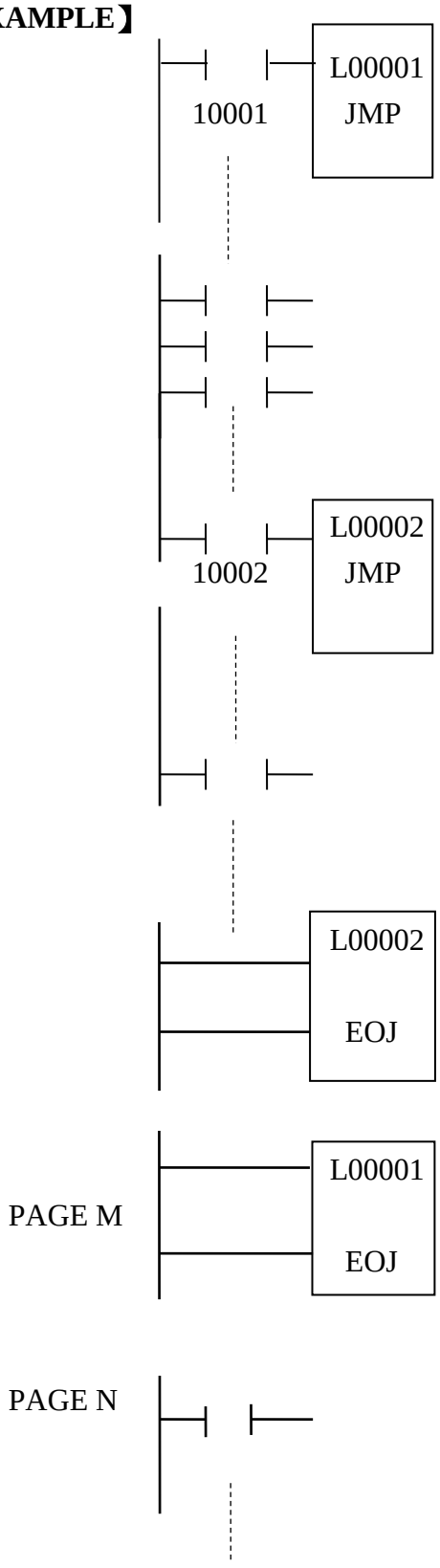
I_1: When ⎍ (⎍↑) is presented, the instruction is executed.

Function Output:

$O_1 = I_1$

$O_2 = 0$

【EXAMPLE】



【DESCRIPTION】

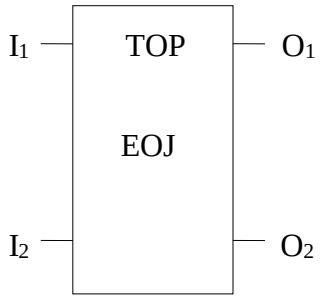
When contact 10001 is energized, the program between JMP L00001and EOJ L00001 is skipped. The execution continues from PAGE N.

If contact 10001 is not energized, then no JUMP action is performed.

If contact 10001 is not energized, but contact 10002 is energized, then the program between JMP L00002 and EOJ L00002 is skipped. The execution continues from PAGE M.

EOJ	END OF JUMP		

SYMBOL:



OPERANDS:

	0	1	3	4	C	P	L
TOP							①

①L1~L150

Description:

This instruction is used with JMP instruction. The label numbers must be matched. Only one JMP-EOJ pair is allowed in a ladder page. The label number must not be repeated. The program between JMP and EOJ is skipped if the Input Control condition is met.

Node Description:

TOP: Label indicating the end of JUMP.

Input Control:

I_1: don't care.

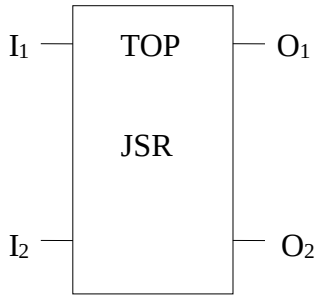
Function Output:

$O_1 = I_1$

$O_2 = 0$

JSR	**JUMP to SUBROUTINE**	⎍	⎍↑

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP							①

①L1~L32

Description:

This instruction is used to call the subroutine whose label is the same as the one defined in the Top node. Subroutine calls may be nested, but only 16 levels are allowed.

Programming requirements are: ①SBR and RET are paired. ②SBR instruction is behind the JSR instruction, and ③RET instruction is behind the SBR instruction.

Input control (I_1) is used to determine whether this instruction is to be executed or not.

Function outputs can be used to determine whether the instruction has been executed.

Node Description:

TOP: Label of the subroutine to be called.

Input Control:

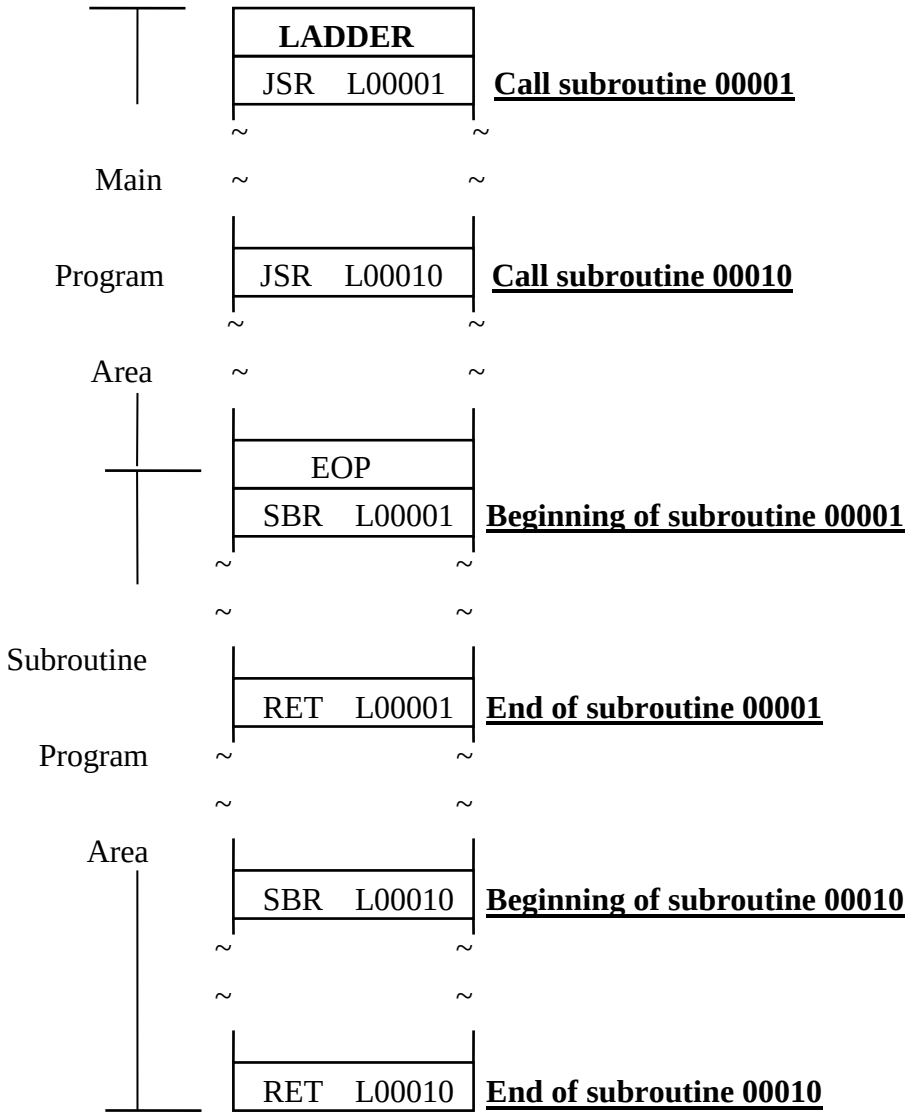
I_1: When ⎍ (⎍↑) is presented, the instruction is executed.

Function Output:

$O_1 = I_1$

$O_2 = 0$

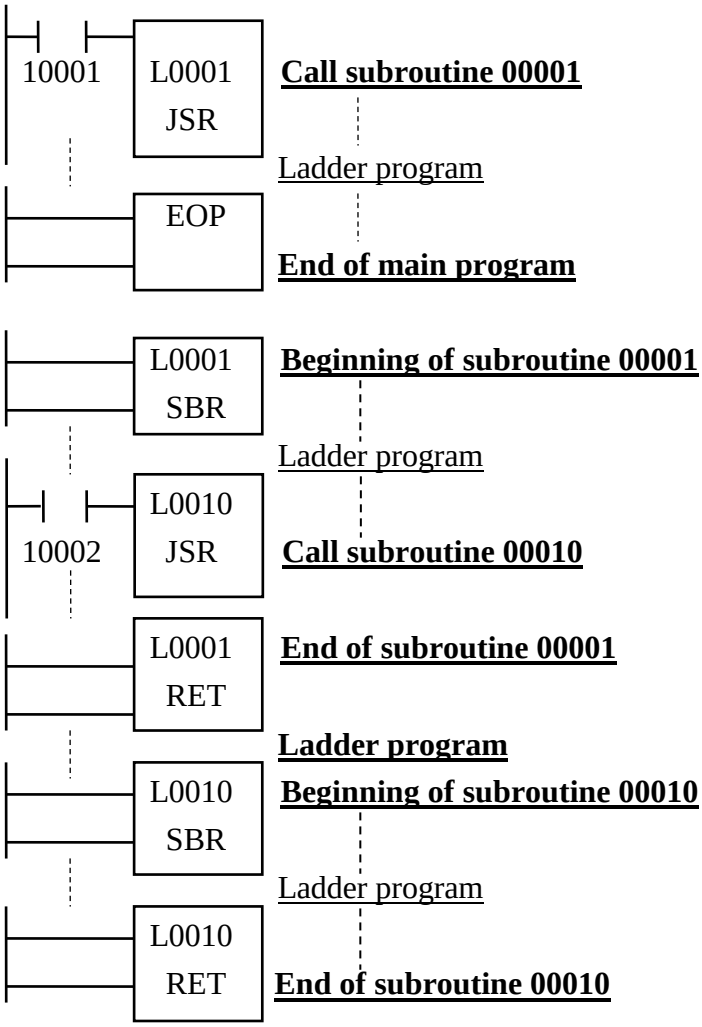
【EXAMPLE】



【DESCRIPTION】

The main program area and the subroutine area are separated by the EOP instruction.
If the EOP instruction does not exist, then the first SBR instruction is used as a program delimiter.

【EXAMPLE】

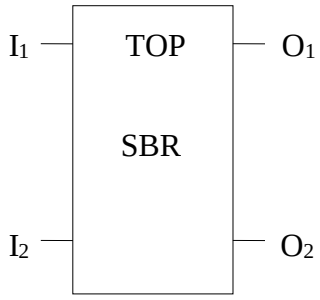


【DESCRIPTION】

When contact 10001 is energized, subroutine L00001 is executed. The program control is returned to the main program when RET L00001 is encountered. The execution of the main program is terminated when the EOP instruction is encountered.
During the execution of subroutine L00001, if contact 10002 is energized, then subroutine L00010 is executed until RET L00010 is encountered. When RET L00001 is encountered, the program control returns to the main program.
If contact 10001 is not energized, then neither subroutine L00001 nor L00010 is executed.

SBR	SUBROUTINE		

SYMBOL:



OPERANDS:

	0	1	3	4	C	P	L
TOP							①

①L1~L32

Description:

This instruction is used to define the beginning of a subroutine. A matched label RET instruction is required to define the end of the subroutine. When the subroutine is called, the program control is transferred from the main program to the next page of the program where the SBR is defined.

Node Description:

TOP: Label of the subroutine defined.

Input Control:

I_1: Don't care.

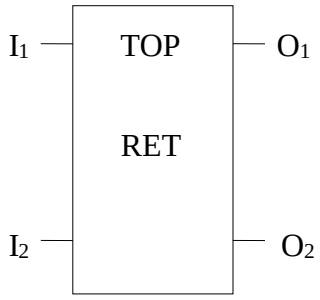
Function Output:

O_1= 0

O_2= 0

RET	**RETURN FROM SUBROUTINE**		

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP							①

①L1~L32

Description:

This instruction is used to define the end of a subroutine. The label number is defined in the top node and must be the same as the calling SBR label number.

Node Description:

TOP: Label of the subroutine.

Input Control:

I_1: Don't care

Function Output:

O_1= 0

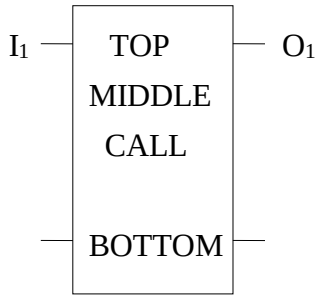
O_2= 0

CALL SUBROUTINE CALL

CALL

CALL	SUBROUTINE CALL		

SYMBOL



OPERANDS

	0	1	3	4	C	P	L
TOP							①
MIDDLE				○		○	
BOTTOM				○		○	

①L1~L32

Instruction Description：

This application instruction is used to call the subroutine defined by the LBL instruction with the same value of top node; the subroutine can be allowed to call subroutine, but only up to 16 layers of nested calls are allowed. In addition, the subroutine's initial setting of values by the parameters' transfer and the function of results return can also be reached,

NOTE: When they are mixed with the JSR instruction to use, the total nested calls of JSR instruction and CALL instruction shall not exceed 16.

In the aspect of control action, the external signal can control the execution of instruction.

In the aspect of function block output, it can be indicated if the instruction is executed.

Note1: A ladder page can only have this unique instruction; that is, no other instructions can be put after this instruction in this page, and it must be placed in the upper left corner of the ladder page during editing.

Note2: When this instruction is used, all of the values of registers and coil state of the subroutine which has been executed will be maintained, jumping out the subroutine, the system does not automatically change or clear its value.

Note3: CALL must be used in conjunction with LBL and RTS.

Node Description：

Top node：The label of subroutine.

Middle node：Input the starting address of the parameters; the first word represents the number of parameters to pass (up to 9, at least 0), and the second WORD begins to place the transferred parameter values.

NOTE: If the number setting of transfer parameters is greater than 9, it will automatically be treated as 9; if the number is 0, it means do not pass parameters.

Bottom node：Return parameters starting address. The return parameters are placed by the specified address (Up to 9, at least 0).

Input Control Description:

I_1: Activation control

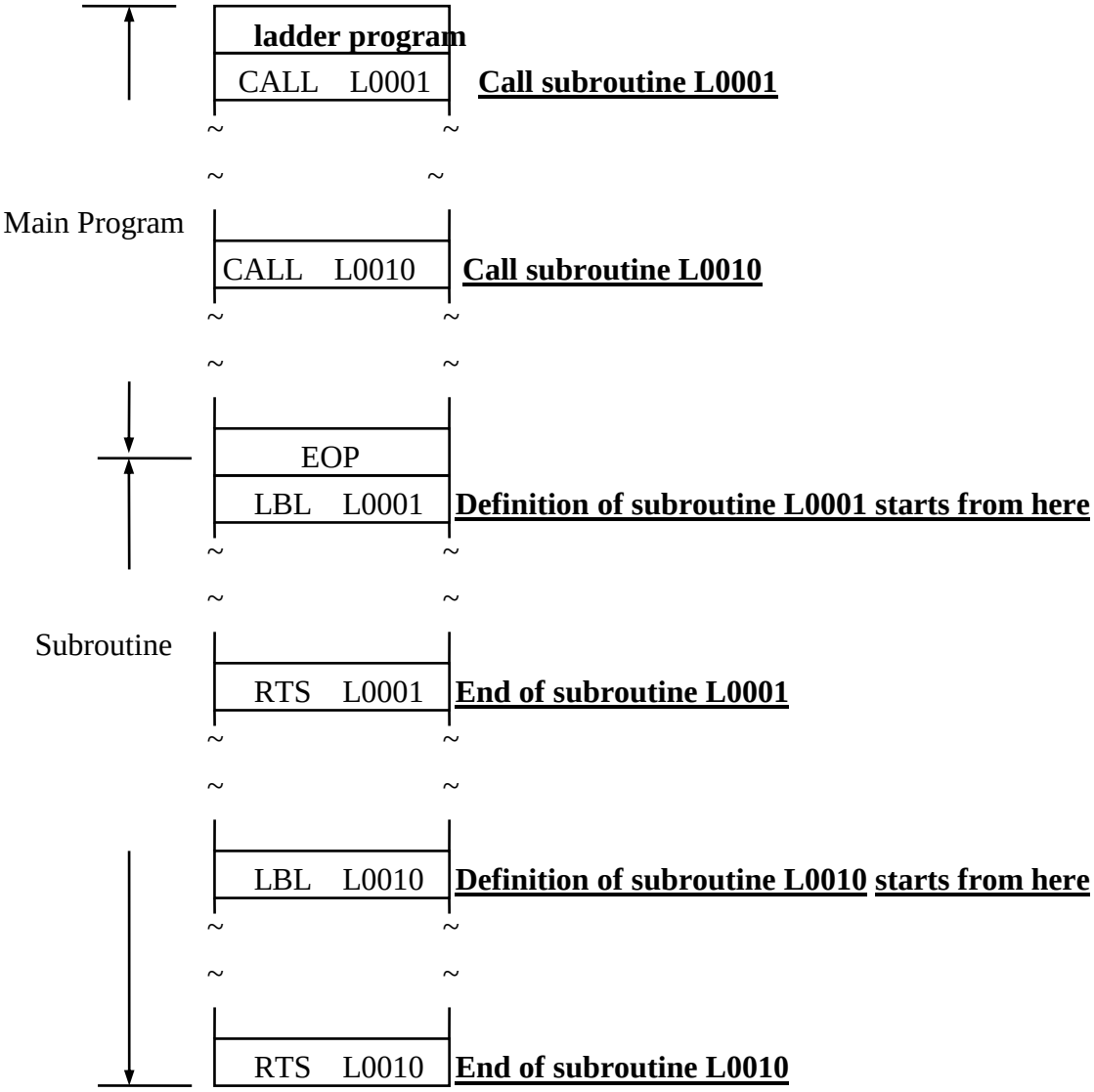
When the input is activated ( , ), if the following conditions are met, then the subroutine call is executed, the same values of top node ①LBL instruction and RTS instruction both exists, ②LBL instruction bits is after the CALL instruction, ③RTS instruction bits is after the LBL instruction.

Function Output Description:

$O_1 = I_1$

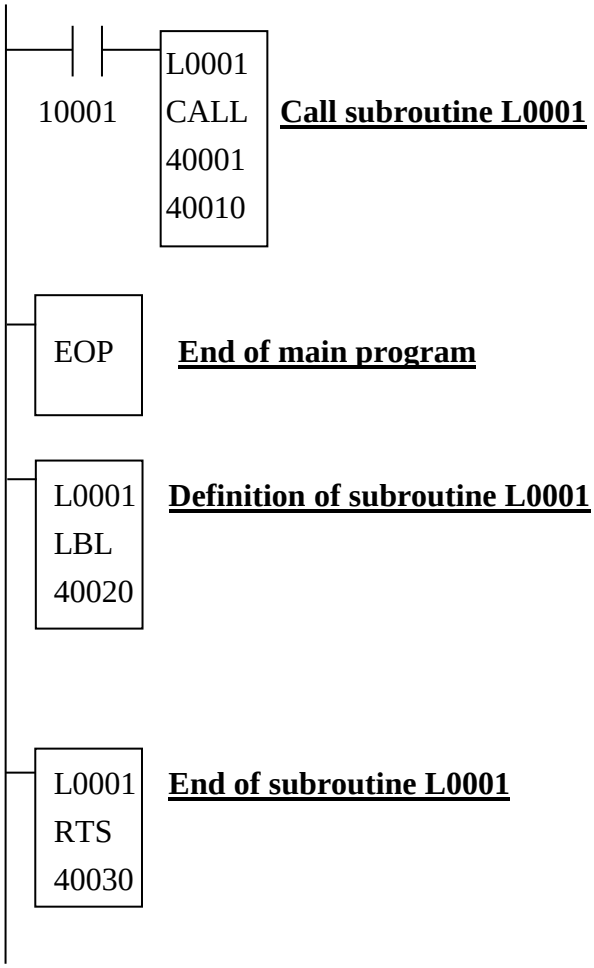
$O_2 = 0$

【**EXAMPLE**】



【**Description**】The program memory area is divided into main-program memory area and subroutine memory area. These two areas can be separated through typing the “EOP” instruction by users. If there is no EOP instruction in the program, the first instruction of “SBR” or “LBL” or “LBLJ” is taken as the starting address of memory area in subroutine.

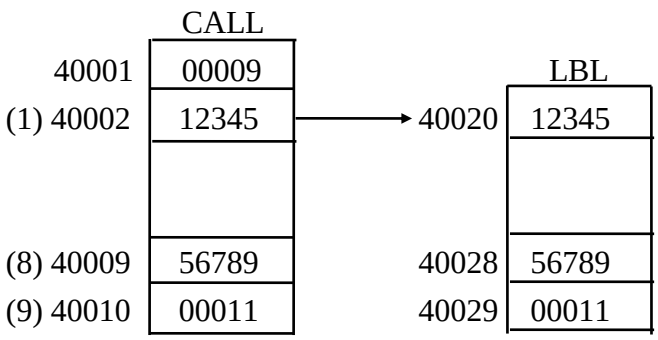
【EXAMPLE】



【**Description**】When the input contact 10001 is energized, subroutine L0001 will be called and executed. Since the middle node 40001 of CALL instruction is set as 9, the contents of 40002~40102 will be copied to 40020~40029 set by LBL for use of subroutine input. When the subroutine is executed to RTS L0001 instruction, it is returned to the main program. Since the bottom node of 40030 RTS instruction is set as 5, the contents of 40031~40035 will be copied to bottom node 40010~40014 of CALL instruction as the parameters of subroutine, and the program which has not been yet executed is continuously executed. When the "EOP" instruction is executed, the execution of main program ends.

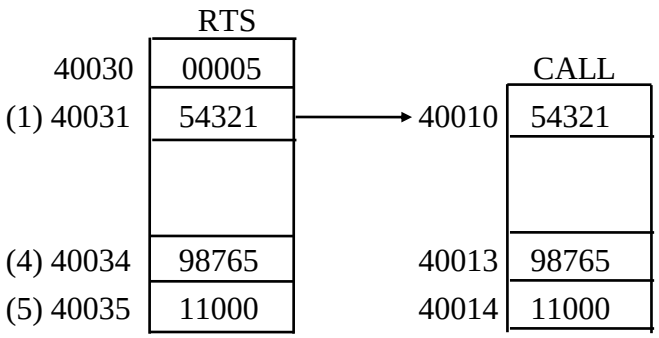
【**process**】

➔When the main program is executed to CALL L0001, the subroutine is called, jump to LBL L0001, and the contents of 40002～40010 are copied to 40020～40029 as the input parameters of subroutine.



➔The problem-solving starts in subroutine, and 40020～40029 can be used to perform the operation.

➔When the program is executed to RTS L0001, the treatment of subroutine is finished and returns to the main program, and 40031～40035 are copied to 40010～40014 as the return parameters of subroutine.



➔The main program is continuous to be executed, and 40010～40014 can be used to perform operation.

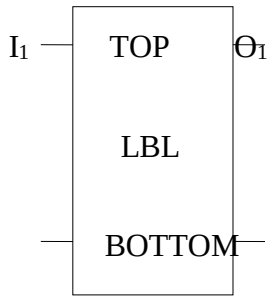
➔When the main program is executed to EOP, the execution of main program is finished, and the problem-solving process is completed.

LBL LABEL

LBL

LBL	LABLE		

SYMBOL



OPERANDS

	0	1	3	4	C	P	L
TOP							①
BOTTOM				○		○	

①L1~L32

Instruction Description:

This application instruction is used to define the beginning of subroutine in collocation with the RTS instruction of the same top node.

When the defined subroutine is called and executed, the program is executed from the next page which the LBL instruction is located, in other words, the LBL instruction is not executed under normal circumstances.

When the main program did not use the EOP instruction to do the end, and when LBL instruction is executed, the scanned program this time complete the execution.

Note1: A ladder page can only have this unique instruction; that is, no other instructions can be put after this instruction in this page, and it must be placed in the upper left corner of the ladder page during editing.

Note2: When the instruction is used, all of the values used by the registers which are executed by the subroutine as well as the coil state will be maintained; jumping out of this subroutine, the system does not automatically change or clear its value.

Note3: LBL and RTS must be used in pairs.

Node Description:

Top node: The label which the subroutine is defined (label)

Bottom node: input parameters starting address, the passed parameters start to place from the specified address. (Up to 9, at least 0).

Input control Description:

I_1: It's not related to the program execution

Function Output Description:

O_1= 0

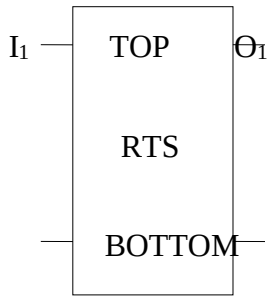
O_2= 0

RTS SUBROUTINE RETURN

RTS

RTS	RETURN FROM SUBROUTINE		

SYMBOL



OPERANDS

	0	1	3	4	C	P	L
TOP							①
BOTTOM				○		○	

①L1~L32

Instruction Description:

This application instruction is used to indicate the end of subroutine in collocation with the LBL instruction of the same top node value.

In the aspect of function block output, the execution status of the instructions can be indicated.

Note1: A ladder page can only have this unique instruction; that is, no other instructions can be put after this instruction in this page, and it must be placed in the upper left corner of the ladder page during editing.

Note2: When this instruction is used, all of the values of registers and coil state of the subroutine which has been executed will be maintained, jumping out the subroutine, the system does not automatically change or clear its value.

Note3: LBLJ MUST BE PAIRED WITH RTSJ DURING USE.

Node Description:

Top node: the label of subroutine

Bottom node: Return parameters starting address. The first WORD indicate the number of parameters to be transferred (Up to 9, at least 0); the second WORD start placing the return value of the parameters.

NOTE: If the number setting of transfer parameters is larger than 9, it will automatically be treated as 9; if the number is 0, it means do not pass parameters.

Input Control Description:

I_1: It's not related to the program execution

Function Output Description:

O_1= 0

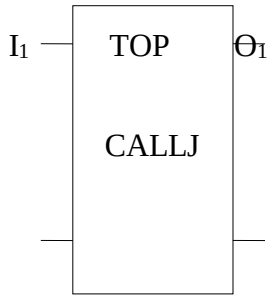
O_2= 0

CALLJ CALL SUBROUTINE

CALLJ

CALLJ	Call SUBROUTINE		

SYMBOL



OPERANDS

	0	1	3	4	C	P	L
TOP							①

①L1~L32

Instruction Description:

This application instruction is used to call subroutine defined by LBLJ instructions that are the same as the top node. In the aspect of control action, the external signal can control the execution of instruction. In the aspect of function block output, it can indicate if the instruction executed.

Note1: A ladder page can only have this unique instruction; that is, no other instructions can be put after this instruction in this page, and it must be placed in the upper left corner of the ladder page during editing.

Note2: When the CALLJ instruction is closed, the same label of top node between LBLJ and RTSJ output coil-**( )**-/upper-edge pulse wave output coil-**(↑)**-/lower-edge pulse wave output coil-**(↓)**-in the program will be 'OFF' by CPU, and all of Registers values and setting output coil used in this section program-**(S)**-state and reset output coil-**(R)**-all maintain the state before closing CALLJ, the system will not automatically change or clear its value.

Note3: The subroutines are allowed to call subroutine, but only up to 16 layers of nested call are allowed, and when the subroutine instructions are mixed to use, the nested call sum of JSR instruction, CALL instruction and CALLJ instruction should not exceed 16.

Note4: Attention! If there is the case of calling subroutine in subroutine; when the subroutine in the outermost layer is closed, then the middle subroutine won't be executed, and all of the registers and output coils in the middle will be maintained the state before the subroutine is closed.

Note5: CALLJ must be used with LBLJ and RTSJ.

Node Description:

Top node: The label of subroutine

Input Control Description:

I_1: Activation control

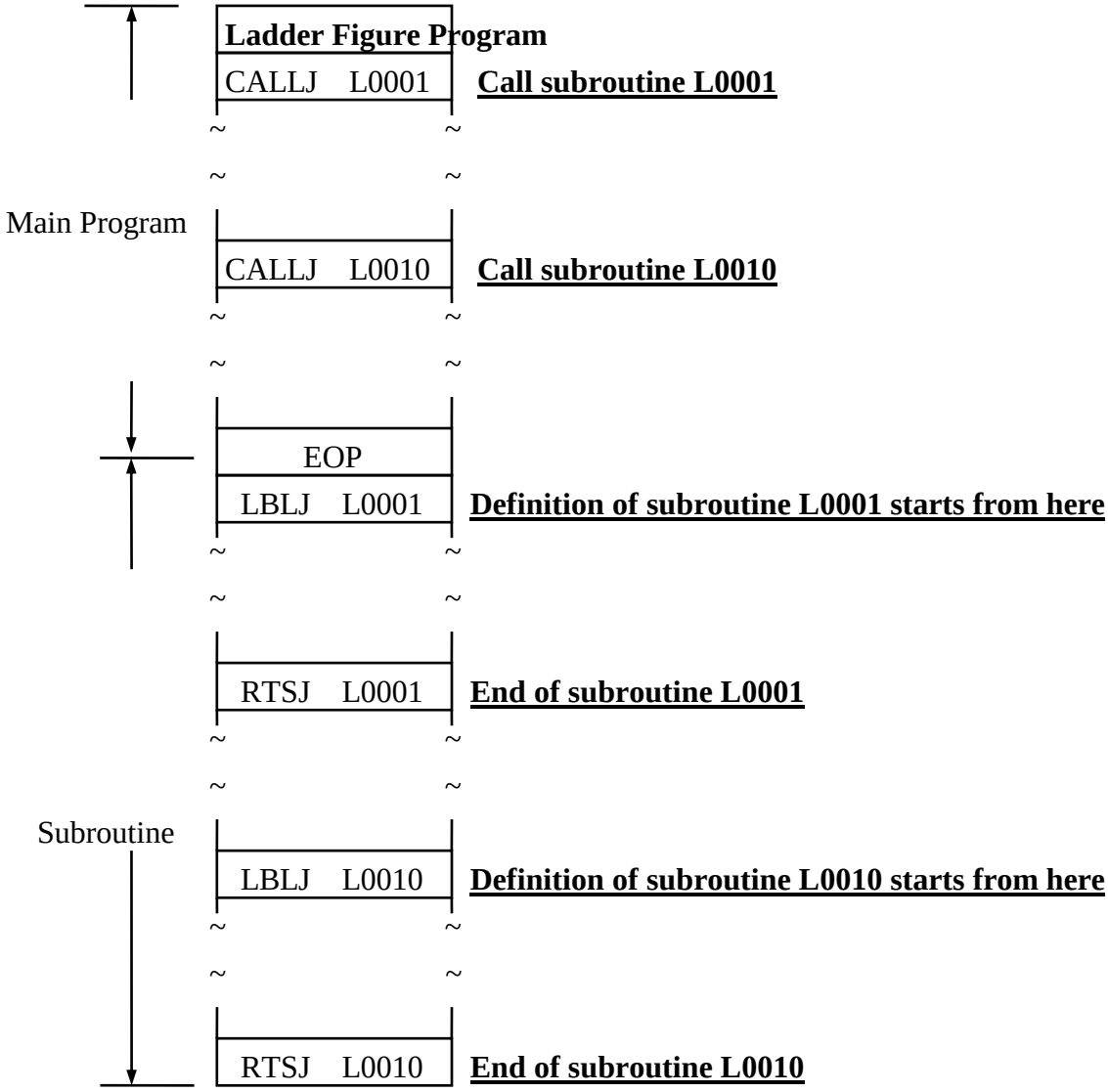
When the input is activated( , ), if the following conditions are met, then the subroutine call is executed, top node values the same ①LBLJ instruction and RTSJ instruction both exist, ②LBLJ instruction bit is behind the CALLJ instruction, ③RTSJ instruction bit is behind the LBLJ instruction.

Function Output Description:

$O_1 = I_1$

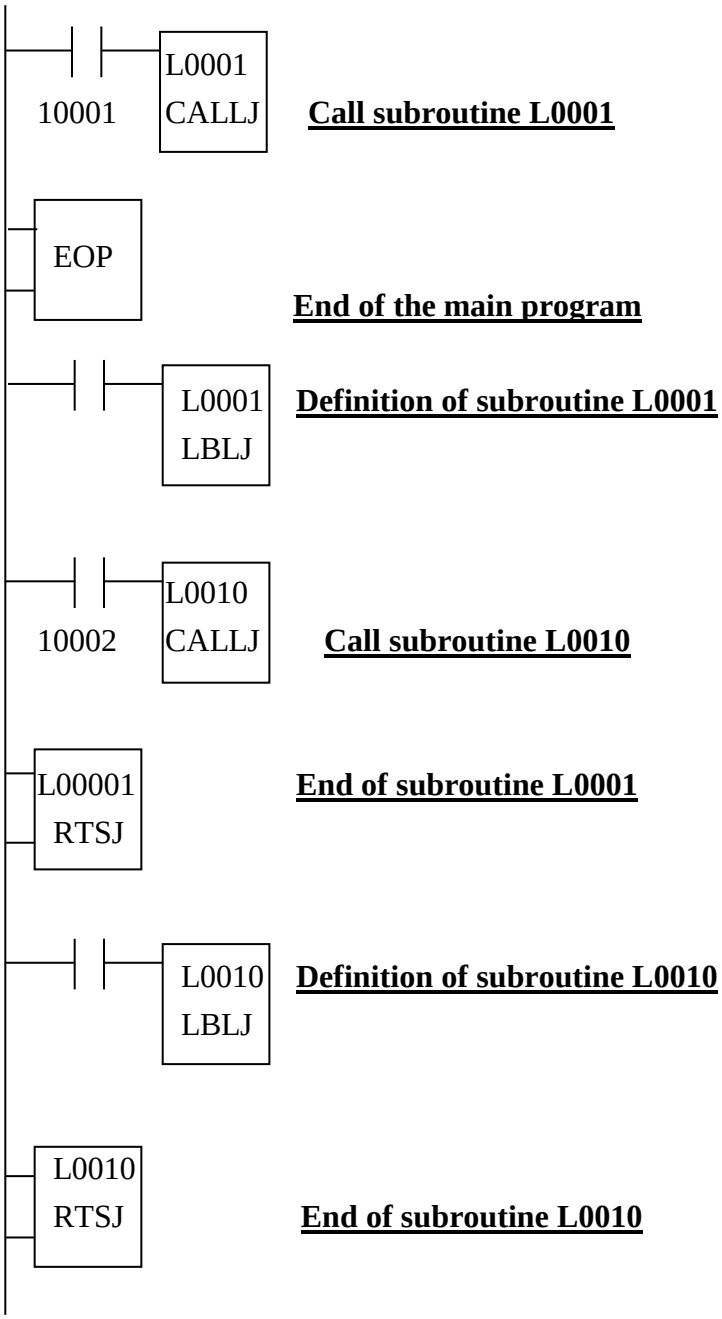
$O_2 = 0$

【**EXAMPLE**】



【**Description**】 The program memory is divided into the main-program area and subroutine area. The “EOP” instruction can be typed for users to separate these two areas. If there is no EOP instruction in the program, then the first “LBLJ” instruction is taken as the starting address of subroutine area.

【**EXAMPLE**】



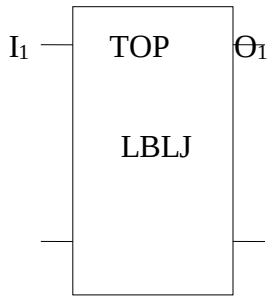
【**Description**】When the input contact 10001 is energized, the subroutine L0001 will be called and executed. When it is executed to RTSJ L0001 instruction, return to the main program, and the program which has not been executed yet is continuously executed. When the "EOP" instruction is executed, the execution of main program ends.

When subroutine L0001 is executed, and the input contact 10002 is energized, then the subroutine L0010 is called and executed. When the RTSJ L0010 instruction is executed, then return to subroutine L0001 and the program is continuously executed, till the RTSJ L0001 instruction is executed, return to the main program and continue the execution; if the input contact 10001 is not energized, the subroutine L0001 and L0010 both are not executed.

LBLJ DEFINITION OF SUBROUTINE

LBLJ	DEFINITION OF SUBROUTINE		

SYMBOL



OPERANDS

	0	1	3	4	C	P	L
TOP							①

①L1~L32

Instruction Description:

This application instruction is used to define the beginning of the subroutine in collocation with RTSJ instruction with the same value of the top node. When the defined subroutine is called and executed, the program is executed from the next page which the LBLJ instruction is located, that is, the LBLJ instruction won't be executed under normal circumstances.

When the EOP instruction is not used to be the end for the main program, and the LBLJ instruction is executed, then the program execution of scan this time ends here.

Note1: A ladder page can only have this unique instruction; that is, no other instructions can be put after this instruction in this page, and it must be placed in the upper left corner of the ladder page during editing.

Note2: LBLJ must be paired with RTSJ during use.

Node Description:

Top node: The label of the defined subroutine.

Input control Description:

I_1: It's not related to the program execution

Function Output Description:

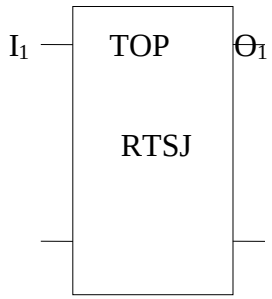
O_1= 0

O_2= 0

ENDING OF RTSJ SUBROUTINE & RETURN

RTSJ	ENDING OF SUBROUTINE & RETURNS		

SYMBOL



OPERANDS

	0	1	3	4	C	P	L
TOP							①

①L1~L32

Instruction Description:

This application instruction is used to represent the end of subroutine in collocation with the LBLJ instruction of the same top node. In the aspect of function block output, it can be indicated the execution condition of the instruction.

Note1: A ladder page can only have this unique instruction; that is, no other instructions can be put after this instruction in this page, and it must be placed in the upper left corner of the ladder page during editing.

Note2: LBLJ must be paired with RTSJ during use.

Node Description:

Top node: The label of subroutine

Input Control Description:

I_1: It's not related to the program execution

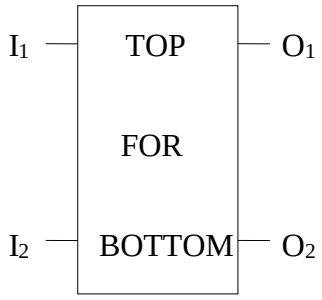
Function Output Description:

O_1= 0

O_2= 0

FOR	LOOP		

SYMBOL:



OPERANDS:

	0	1	3	4	C	P	L
TOP							②
BOTTOM				○	①	○	

① 1~255

② L1~L64

Description:

The program segment between the FOR and NEXT instructions with the same label number (defined in the TOP node) are repeated for a number of times (as defined in the BOTTOM node).
Loops may be nested. Maximum of 8 levels are allowed.

Node Description:

TOP: Label of the loop.
BOTTOM: Number of repetitions.

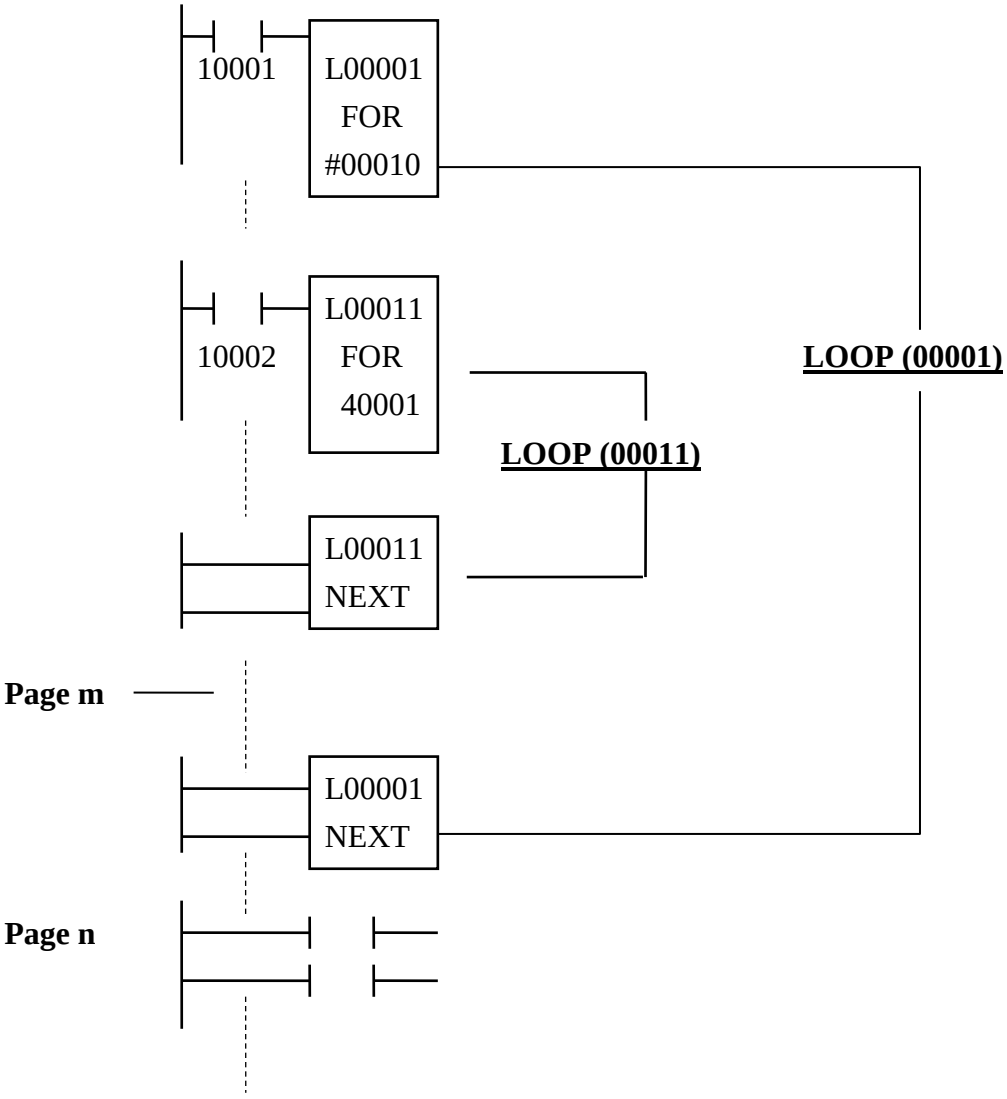
Input Control:

I_1: When ⎍ (↑) is presented, a matched label NEXT instruction is found whose position is behind the FOR instruction. Then, this instruction is executed.

Function Output:

O_1= 0
O_2= 0

【EXAMPLE】

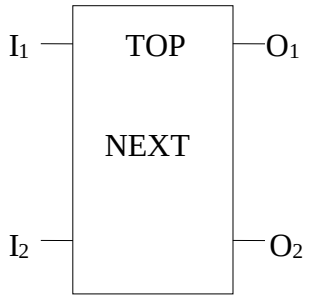


【DESCRIPTION】

When contact 10001is energized, the loop (L00001) is executed 10 times, then the program resumes from page n. If contact 10001 is not energized, the loop (L00001) is skipped. If both contacts 10001 and 10002 are energized, then loop L0011 is executed N times (as defined in the bottom node 40001), and loop L00001 is executed 10 times. If contact 10001 is energized while contact 10002 is not, then the loop (L00001) is executed for 10 times, while loop L0011 is skipped.

NEXT	END of LOOP		

SYMBOL:



OPERANDS:

	0	1	3	4	C	P	L
TOP							①

①L1~L64

Description:

This instruction is used to define the end of a loop with the same label number.

Node Description:

TOP: Label of the loop.

Input Control:

I_1: no action.

Function Output:

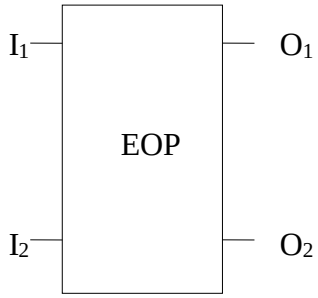
$O_1 = I_1$

$O_2 = 0$

CHAPTER 4: FLOW CONTROL INSTRUCTIONS

EOP	**END of MAIN PROGRAM**		

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**

Description:

This instruction is used to define the end of a program. All the programming behind this instruction is ignored. The program scan terminates when this instruction is encountered.

Node Description:

Input Control:

I_1: Don't care

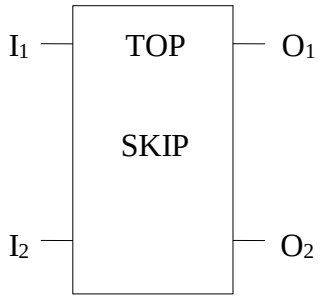
Function Output:

O_1=0

O_2=0

SKIP	**SKIP**		

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP			○	○	○	○	

Description:

This instruction is used to control the sequence of the program execution.

Input control (I_1) is used to determine whether this instruction is to be executed or not.

Users are recommended to have only a SKIP instruction in a ladder page.

Node Description:

TOP: number of program pages to be skipped. If this value is equal to 0, then the program scan is terminated.

Input Control:

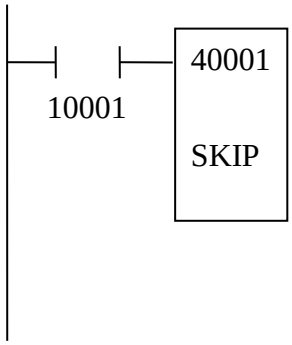
I_1: When ⎍ (⎍↑) is presented, the instruction is executed.

Function Output:

O_1= 0

O_2= 0

【EXAMPLE】

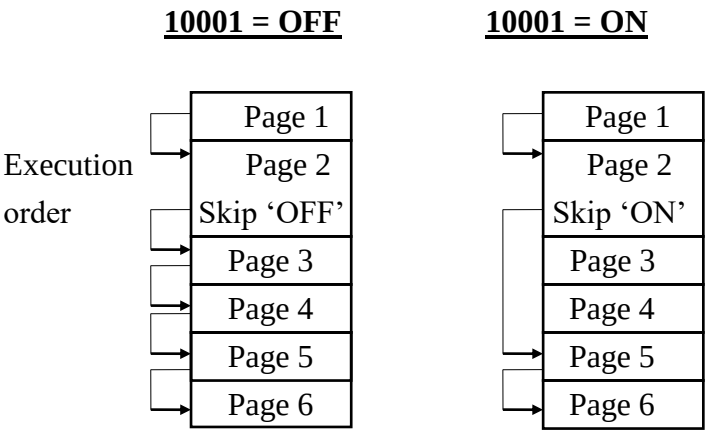


【DESCRIPTION】

When contact 10001 is energized, then the skip instruction is executed.

1. If the content of register 40001is #00002, then the next two pages are skipped.
2. If the content of register 40001is 0, then the program execution for this scan is terminated.

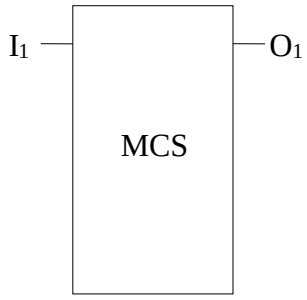
Let register (40001)=00002:



The **SKIP** instruction is at the bottom of Page 2.

MCS	**MASTER CONTROL SET**		

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**

Description:

This function block is used for controlling the program flow. There must be a matched label MSE (Master control end) function block for the ladder program to execute correctly. The power rail input of the ladder program segment between the MCS-MSE pair is determined by the I_1 of MCS. If I_1 is ON, the power rail input of the ladder program segment between the MCS-MSE pair is ON or vice versa.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Function outputs can be used to determine whether the function block has been executed.

Nesting MCS are not supported.

Node Description:

Input Control:

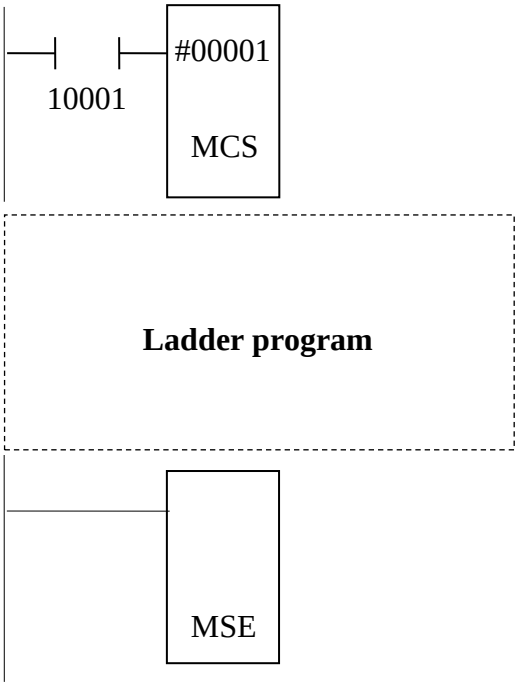
I_1: Power control

Function Output:

O_1: I_1

O_2=0

【EXAMPLE】

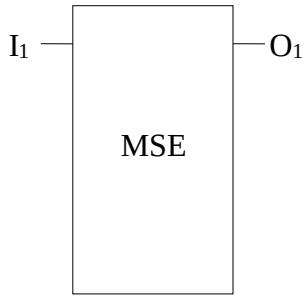


【DESCRIPTION】

When contact 10001 is energized, the power rail input of ladder program segment between the MCS-MSE function blocks is OFF. If the contact 10001 is not energized, then the ladder program segment is executed as usual.

MSE	**MASTER CONTROL END**		

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**

Description:

This function block is the matched ending instruction for MCS function block.

Node Description:

Input Control:

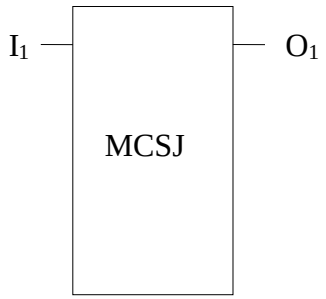
Function Output:

O_1=0

O_2=0

MCSJ	MASTER CONTROL SET (cleared after JUMP)		

SYMBOL



OPERANDS

	0	1	3	4	C	P	L
							①

①L1~L64

Instruction Description:

This application instruction is used to perform process control with the MCRJ instruction. Input control is used to control the activations of instruction. The output function is used to display execution state of the application instruction

Note1: The ladder page can only have this unique instruction; that is, no other instructions can be put after this instruction in this page, and it must be placed in the upper left corner of the ladder page during editing.

Note2: The execution of MCSJ instruction will make the program controlled between MCSJ and MCRJ.

Note3: When the MCSJ instruction is closed, the coils of output coil-( )-/upper edge pulse wave output coil-( )-/lower edge pulse wave output coil-( )- between MCSJ and MCRJ will be closed by CPU, but all registers values and output coil-**(S)**-state and reset output coil-**(R)**-state used in the section of the program will be maintained.

Note4: MCSJ and MSRJ must be used in pairs.

Note5: It may not be nested during use.

Node Description:

Top node: The label of the defined process control.

Input Control Description:

I_1 = 0, the power rail input of the ladder program segment between MCSJ and MCRJ are all in 'OFF' state

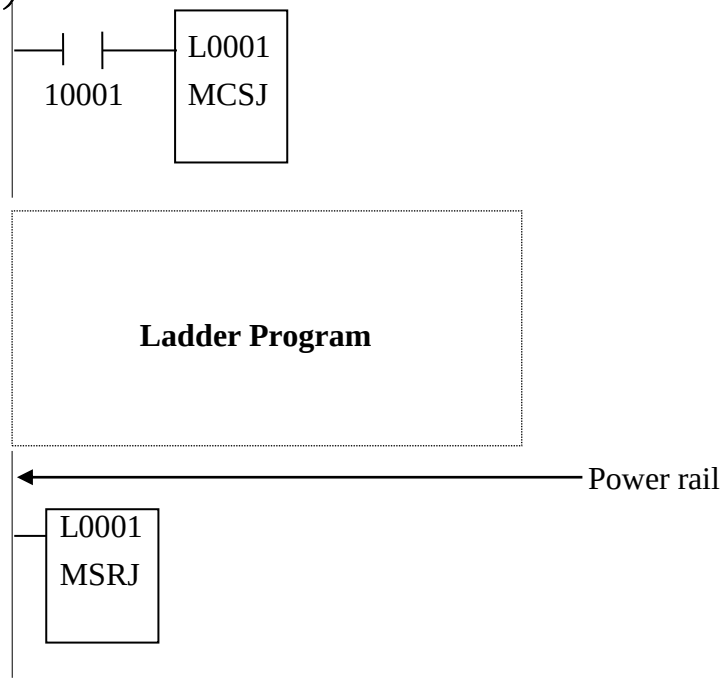
= 1, the power rail input of the ladder program segment between MCSJ and MCRJ are all in 'ON' state

Function Output Description:

$O_1 = I_1$

$O_2 = 0$

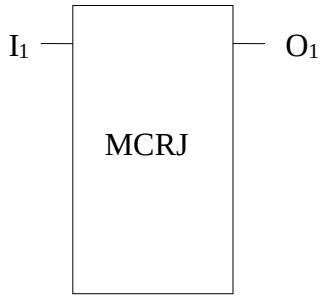
（**EXAMPLE**）



（**Description**） When the input contact 10001 is energized, the execution of Ladder program inside the dashed box in the above figure is influenced. If 10001 is not energized, then the power rail state after MCSJ instruction will be all ’OFF’; until the execution of the MSRJ instruction, the power rail will be restored to the 'ON' state.

MCRJ	MASTER CONTROL RESET (cleared after JUMP)		

SYMBOL



OPERANDS

	0	1	3	4	C	P	L
							①

①L1~L64

Instruction Description:

This application instruction is used to perform process control with the MCSJ instruction. Input control is used to control the activation of the instruction. The output function is used to display execution state of the application instruction

note1: The ladder page can only have this unique instruction; that is, no other instructions can be put after this instruction in this page, and it must be placed in the upper left corner of the ladder page during editing.

note2: MCSJ and MSRJ must be used in pairs.

Node Description:

Top node: The label of the defined process control.

Input Control Description:

I_1：It's not related to the program execution.

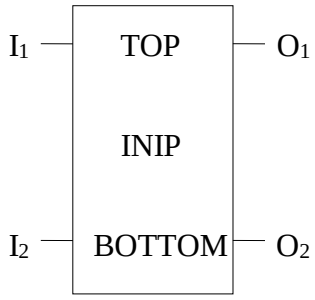
Function Output Description:

O_1= 0

O_2= 0

INIP	**INITIALIZATION OF POINTER**	⎍	⎍↑

SYMBOL:



OPERANDS:

	0	1	3	4	C	P	L
TOP	○	○	○	○			
BOTTOM						①	

①P0~P15

Description:

This function is used to define the content of a pointer. The constant in the bottom node is used to define which pointer is to be initialized, and the number in the top node is the initialization value.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Node Description:

TOP: Type of register and its number.

BOTTOM: Pointer to be defined.

Input Control:

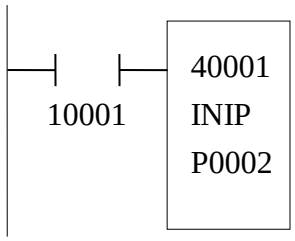
I_1: When ⎍ (⎍↑) is presented, the instruction is executed.

Function Output:

$O_1=I_1$

$O_2=0$

【EXAMPLE】

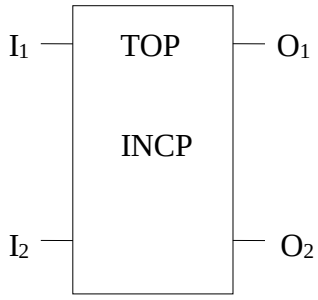


【DESCRIPTION】

When contact 10001 is energized, the relationship: (P0002) = 40001is defined.
This means the (P0002) pointer points to this 40001 register.

INCP	INCREMENT OF POINTER	⎍	↑⎍

SYMBOL:



OPERANDS:

	0	1	3	4	C	P	L
TOP						①	

①P0~P15

Description:

This function is used to increment the pointer by one. The constant in the top node defines which pointer is to be incremented.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Node Description:

TOP: Pointer to be incremented.

Input Control:

I_1: When ⎍ (↑⎍) is presented, the instruction is executed.

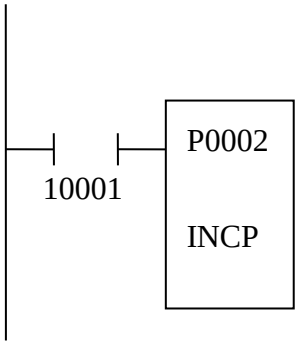
Function Output:

$O_1=I_1$

O_2=Error

=1, When this function is called, the reference number pointed by the pointer is already pointed to the last reference number of that reference type.

【EXAMPLE】

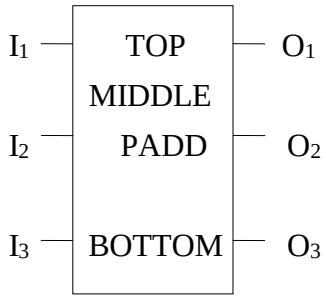


【DESCRIPTION】

Assume that pointer 2 contains 40001, when 10001=1. Then, pointer 2=40002, i.e. P0002=40002 after execution.

PADD	**ADDITION OF POINTER**		

SYMBOL:



OPERANDS:

	0	**1**	**3**	**4**	**C**	**P**	**L**
TOP						①	
						↑	
						①	

①P0~P15

↑0~9999

Description:

The content of the pointer is the top node and the constant in middle node are added and the sum is stored in the content of the pointer in the bottom node.

Node Description:

TOP: Pointer of top node

Middle: A constant

Bottom: Pointer of bottom node

Input Control:

I_1: When ⌐¬ (⌐↑¬) is presented, the function block is executed.

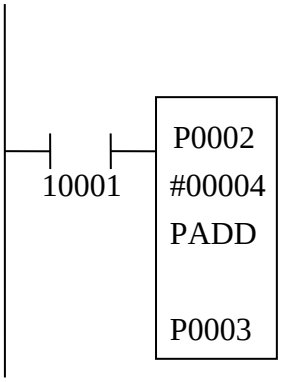
Function Output:

$O_1 = I_1$ (O_1 will be '0' when O_3 is '1')

$O_2 = 0$

$O_3 = 1$ (error output) = 1, if pointer is beyond the upper limited address.

【EXAMPLE】

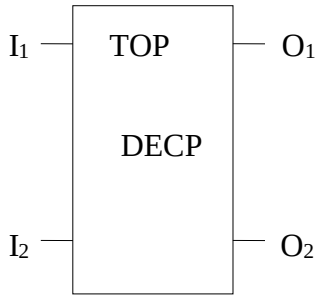


【DESCRIPTION】

When the contact 10001 is energized, the content of pointer P0002 is added 4 and the sum is stored to the content of pointer P0003.

DECP	**DECREMENT OF POINTER**		

SYMBOL:



OPERANDS:

	0	1	3	4	C	P	L
TOP						①	

①P0~P15

Description:

This function is used to decrement the pointer by one. The constant in the top node defines which pointer is to be decremented.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Node Description:

TOP: Pointer to be incremented.

Input Control:

I_1: When ⎍ (⎍↑) is presented, the instruction is executed.

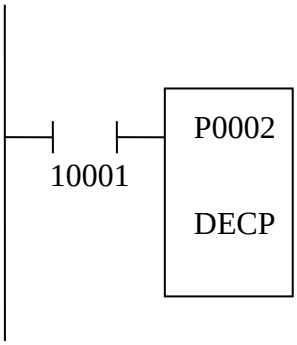
Function Output:

$O_1=I_1$

O_2= Error

= 1, when this function is called, the reference number pointed by the pointer is already pointed to the first reference number of that reference type.

【EXAMPLE】

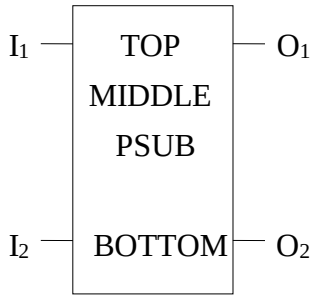


【DESCRIPTION】

Assume that pointer 2 contains 40011, when 10001=1. Then, pointer 2 = 40010, i.e. P0002 = 40010 after execution.

PSUB	SUBRATION OF POINTER	⎍	⎍↑

SYMBOL:



OPERANDS:

	0	1	3	4	C	P	L
TOP						①	
						↑	
BOTTOM						①	

①P0~P15

↑0~9999

Description:

The content of the pointer in the top node is subtracted by a constant in middle node and the result is stored in the content of the pointer in the bottom node.

Node Description:

TOP: Source pointer

MIDDLE: A constant

BOTTOM: Destination pointer

Input Control:

I_1: When ⎍ (⎍↑) is presented, the function block is executed.

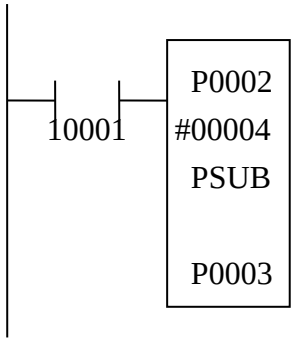
Function Output:

$O_1 = I_1$ (O_1 will be '0' when O_3 is '1')

$O_2 = 0$

O_3 (error output)=1, if pointer is lower than the low limited address.

【EXAMPLE】

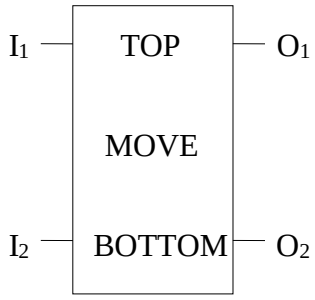


【DESCRIPTION】

When the contact of 10001 is energized, the content of pointer P0002 is subtracted 4 and the sum is stored to the content of pointer P0003.

MOVE	DATA MOVE	⎍	⎍↑

SYMBOL:



OPERANDS:

	0	1	3	4	C	P	L
TOP	○	○	○	○	①	○	
BOTTOM	○			○		○	

①0~65535

Description:

This function is used to define the content of a register (4xxxx) or discrete output (0XXXX).

Input control (I_1) is used to determine whether this function block is to be executed or not.

Node Description:

TOP: Referenced (or source) register or a constant.

BOTTOM: Register or (0XXXX) to be initialized (target).

Input Control:

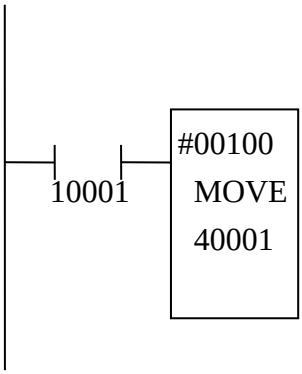
I_1: When ⎍ (⎍↑) is presented, the instruction is executed.

Function Output:

$O_1=I_1$

$O_2=0$

【EXAMPLE】

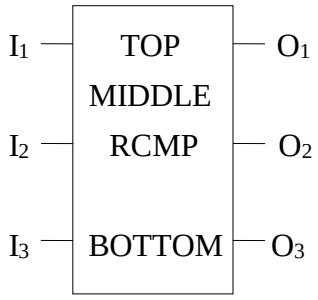


【DESCRIPTION】

When contact 10001 is energized, the constant #00100 is stored in register 40001, i.e. (40001) = 100.

RCMP	REGISTER COMPARE		

SYMBOL:



OPERANDS:

	0	1	3	4	C	P	L
TOP	○	○	○	○	①	○	
MIDDLE	○	○	○	○	①	○	
BOTTOM				○	↑	○	

①0~65535

↑1~2

Description:

This function is used to compare the data in the top node and the middle node.

Input control (I_1) is used to determine whether this function block is to be executed or not.

Outputs (O_1, O_2, O_3) are represented the comparing result (>, =, <) of top node and the middle node when this function block is executed.

Node Description:

TOP: Top node data.

MIDDLE: Middle node data.

BOTTOM: Length to be compared (1: Word, 2: Long word)

Input Control:

I_1: When ⎍ (⎍↑) is presented, the instruction is executed.

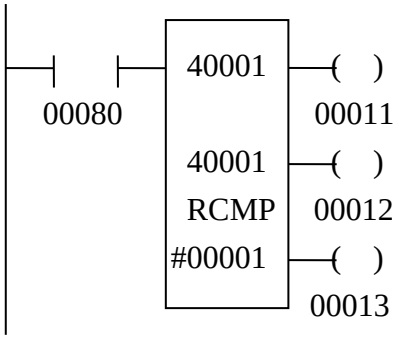
Function Output:

O_1 : comparing result (data of top node > middle node)

O_2 : comparing result (data of top node = middle node)

O_3 : comparing result (data of top node < middle node)

【EXAMPLE】



【DESCRIPTION】

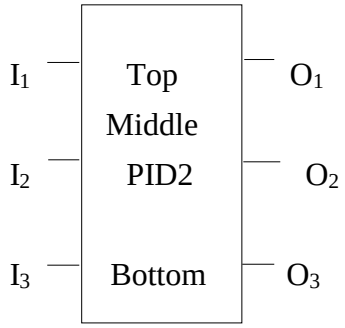
Assumed that the content of register (40001) = 9000(10) and the content of register (40002) = 500(10): When the contact of 00080 is energized, the coil of 00011 will be 'ON' because the content of (40001) > (40002).

CHAPTER 5: OTHERS

PID2

PID2	PID2 Control		

Symbol



Operands

	0	1	3	4	C	P	L
Top				○			
Middle				○			
Bottom					○		

Descriptions:

This function block calculate the difference between the present value and the set-point, and produce control signal to minimize the difference via PID2 calculation.

Nodes:

Top: PID2 function parameters. Please see the next page.

Middle: Working Area and Status area for PID2 function. Please see the following page.

Bottom: Cycle time for PID2 Function, unit: 1/10 sec.

INPUT:

I_1:Auto/Manual Mode

= 1, Output is controlled by PID2 function, = 0, Output is obtained from manual input. Error detecting is still enabled.

I_2: Bumpless transition during Manual to Auto mode switching.

= 1, Bumpless transition enabled, = 0, Bumpless transition disabled.

I_3 : Direct/Reverse Mode

= 1, Decrease Output as Error increases. = 0, Increase Output as Error increases.

OUTPUT:

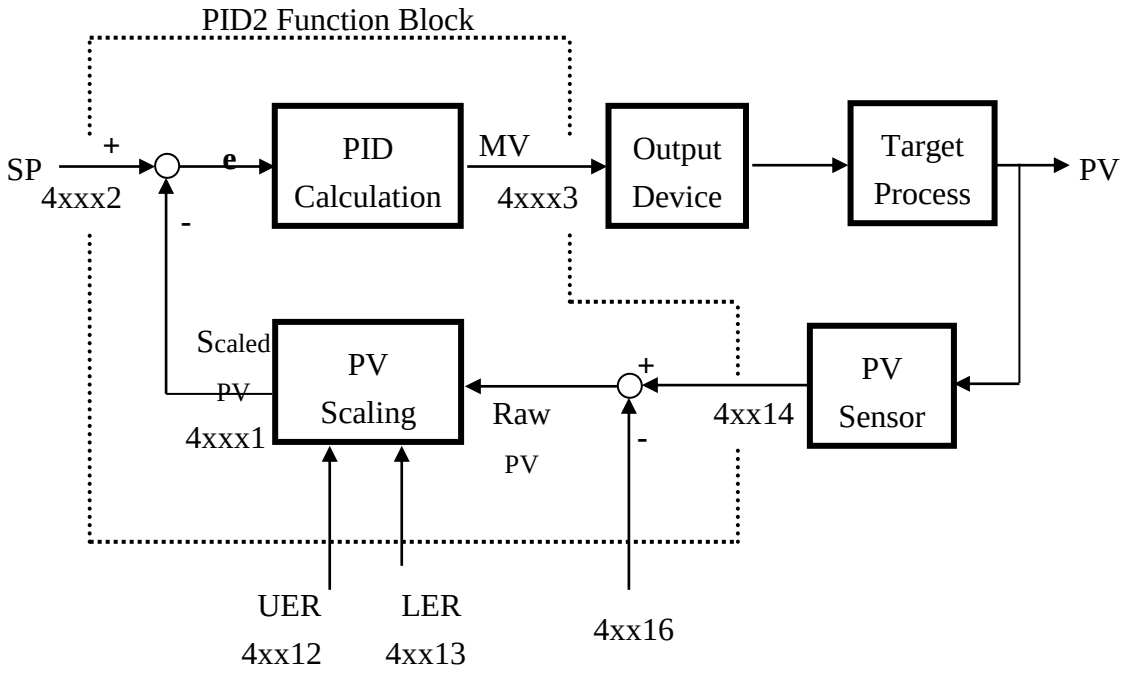
O_1: = 1, if there is any parameter error.

O_2: = 1, if the present value (scaled PV) is higher than the high alarm limit.

O_3: = 1, if the present value (scaled PV) is lower than the low alarm limit.

Description:

PID2 Control Loop:



PID2 formula:

$$MV(t) = \frac{100}{Pb}\left[e(t) + K_I\int e(t)dt + K_D\left(\frac{de(t)}{dt}\right)\right] + Bias$$

Where:

MV(t) = Control Output

Pb = Proportional Band

e(t)= Error (Difference between Scaled PV and SP)

K_I = Constant for Integration Term, or, reset time constant

K_D = Constant for Derivative Term, or, rate time constant

Bias = Correction Value, or offset to Output

TOP Node: Register: 4xxx1 ~ 4xx16

4xxx1: An internal register used to store the scaled PV in Engineering Unit.

$$\text{Scaled PV} = \frac{\text{Raw PV}}{\text{Sensor Range}} * (\text{UER} - \text{LER}) + \text{LER}$$

Where: Raw PV: Obtained from the difference of Register 4xx14 and 4xx16.

UER: Upper bound of Engineering measurement Range

(See also Register 4xx12)

LER: Lower bound of Engineering measurement Range

(See also Register 4xx13)

Sensor Range: **4096**. Assuming that an AD020 module is used to convert Raw PV signal (0~10V) to digital data (0~65535), then the Raw PV must be divided by **16** first to maintain consistency.

4xxx2: Set Point in Engineering Unit. (0~9999)

4xxx3: PID2 control output MV (0~4096). Please use a proper scaling factor to scale this control output and then send to the Output Device.

In Auto Mode (I_1 =1), the data in this register is the result of PID2 calculation. In Manual Mode (I_1 =0), filling this register by user is required.

4xxx4: High alarm limit in Engineering Unit (0~9999). This number should be greater than the Set Point.

4xxx5: Low alarm limit in Engineering Unit (0~9999). This number should be less than the Set Point.

4xxx6: Proportional Band (Pb:5~500). The term Proportional Band is also referred to as the "sensitivity". The reciprocal of Pb is "Gain". As seen from the PID2 formula, the "Gain" is the proportional factor between "Error" and output MV. For example: if Pb=5, then MV is amplified 20 times.

4xxx7: Constant for Integration Term, or, Reset time Constant (K_I: 0~9999). As seen from the PID2 formula, the K_I represents the contribution of the Integral. If $K_I = 0$, then this function block becomes a PD function block.

4xxx8: Constant for Derivative Term, or, Rate time Constant (K_D: 0~9999). As seen from the PID2 formula, the K_D represents the contribution of the Derivative. If $K_D = 0$, then this function block becomes a PI function block. If both $K_I = 0$ and $K_D = 0$, then this function block becomes a proportional control function block.

4xxx9: Bias, Correction Value, or offset to Output (0~4095).

4xx10: High integral wind-up limit, or, upper bound of output. Usually this value is set at 4095.

4xx11: Low integral wind-up limit, or, lower bound of output. Usually this value is set at 0000.

4xx12: Upper bound of Engineering Range (0001~9999). Specify the upper limit of the sensor output in Engineering Unit in this register. For example, a RTD10 module produce unsigned digital

data 1500 ~7500 for temperature 0℃~600℃, then specify 600 for this register. This number should be greater than the Set Point.

4xx13: Lower bound of Engineering Range (0000~9998). Specify the lower limit of the sensor output in Engineering Unit in this register. For example, a RTD10 module produce unsigned digital data 1500 ~7500 for temperature 0℃~600℃, then specify 0000 for this register. This number should be less than the Set Point.

4xx14: Raw PV. Move the data from the output PV sensor to this register. (See also Register 4xx16)

4xx15: Internal Register for storing the status of "Auto" or "Manual" mode. If the content of this register is 11(Hex), the PID2 function block is in Manual mode. If 55(Hex), "Auto" mode.

4xx16: Correction value for Row PV (0~4096). Specify a correction value in this register. This value is subtracted from the Raw PV (obtained from Register 4xx14), and the result is then used in the calculation of Register 4xxx1.

Middle Node: Register 4yyy1~4yyyy5

4yyy1: PID2 function Block Status.

Bit 1: =1, if there is any parameter error.
Bit 2: =1, if High Alarm limit is exceeded.
Bit 3: =1, if Low Alarm limit is exceeded.
Bit 4 ~ Bit 5: Reserved.
Bit 6: =1, if PID2 function Block is in "Auto" mode and computing.
Bit 7 ~ Bit 12: Reserved.
Bit 13: $=I_3$
Bit 14: $=I_2$
Bit 15: $=I_1$
Bit 16: Reserved.

4yyy2: Internal Register for PID2 Loop timer.

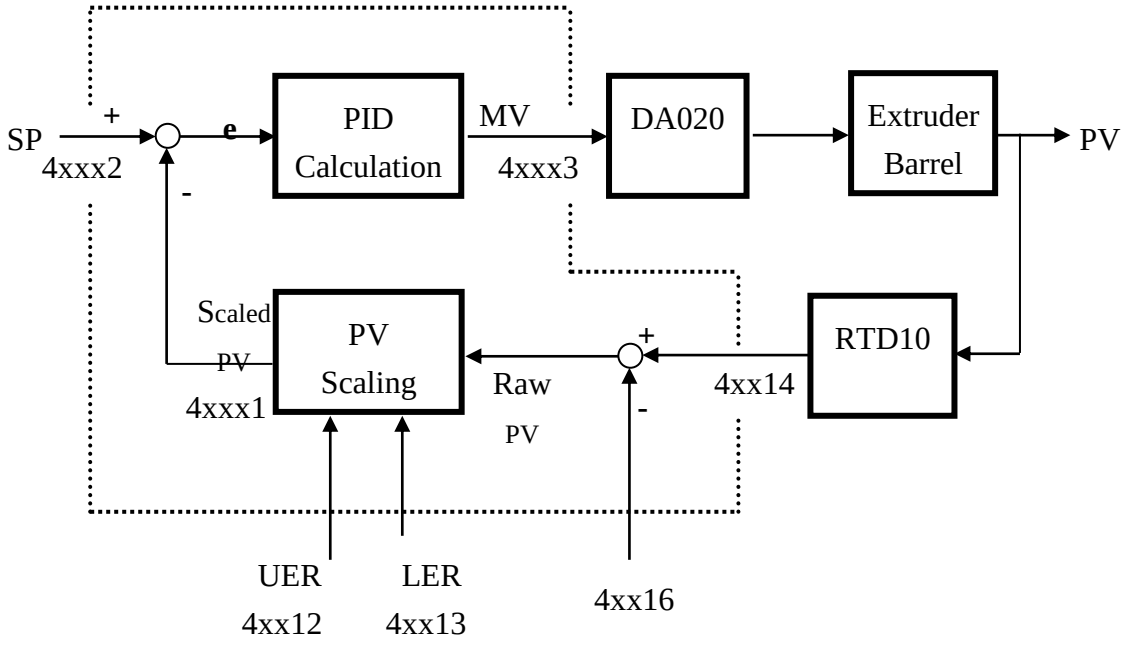
4yyy3: Internal Register for storing High order integral summation.

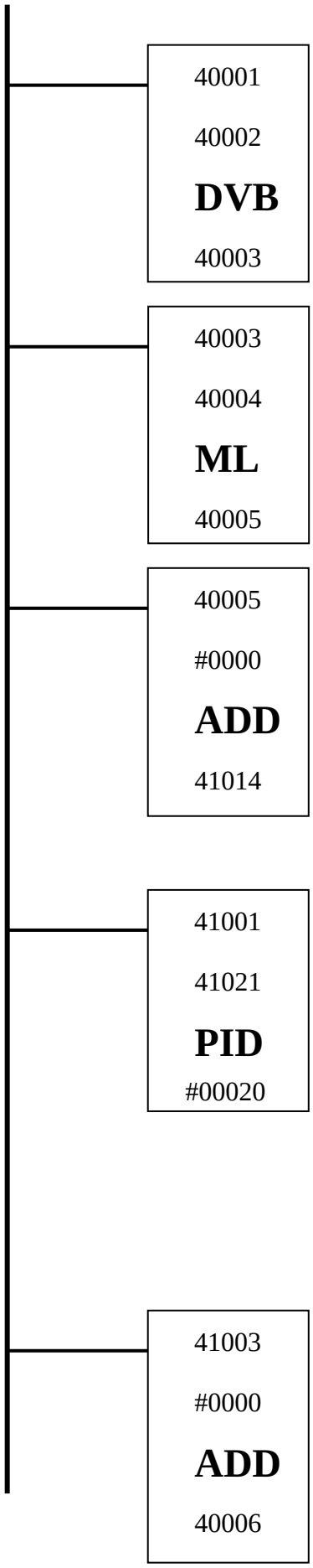
4yyy4: Internal Register for storing Low order integral summation.

4yyy5: Internal Register for storing Scaled PV used in the previous scan.

Bottom Node: Cycle time, unit: 1/10 sec. 00010 stands for one second.

Example:


SP
4xxx2
+
-
e
PID
Calculation
MV
4xxx3
DA020
Extruder
Barrel
PV
Scaled
PV
4xxx1
PV
Scaling
Raw
PV
+
-
4xx14
RTD10
UER
4xx12
LER
4xx13
4xx16



40001: Obtain from RTD module

40002: 1770(Hex)=6000 (=7500-1500) ,
Data range for 0~600℃

40003: Quotient

40004: 1000(Hex)=4096, Sensor Range

40005: Product, Raw PV

Move Raw PV to PID2's TOP node Register 41014.

41001

41002: 0220℃, processing temperature for HDPE

41003: Control Output (MV)

41004: 0250℃, High Alarm Limit

41005: 0200℃, Low Alarm Limit

41006: 0200, Proportional Band

41007: 0300, KI

41008: 0000, thus, PI control only.

41009: 0000, Bias

41010: 4095, upper bound of MV

41011: 0000, lower bound of MV

41012: 0600℃, UER

41013: 0000℃, LER

41014: Raw PV

41015: 0055, Auto mode

41016: 0000, Correction for Raw PV

Bottom Node: Cycle time, 2 sec.

Move PID2 output to DA020 module