DLNK(226) (CP1H CPU Units)

The CPU BUS UNIT I/O REFRESH instruction (DLNK(226)) can be used to refresh memory allocated to CJ-series CPU Bus Units in the CIO and DM Areas, as well as data link data and other data specific to the CPU Bus Units. The unit number of the CPU Bus Unit is specified when DLNK(226) is executed to refresh all of the following data at the same time.

- Words allocated to the Unit in CIO Area
- Words allocated to the Unit in DM Area
- Special refreshing for the Unit (e.g., data links for Controller Link Units or remote I/O for DeviceNet Units)

1-1-11 Program Capacity

The maximum program capacities of the CP-series CPU Units for all user programs (i.e., the total capacity of all tasks) are given in the following table. All capacities are given as the maximum number of steps. The capacities must not be exceeded, and writing the program will be disabled if an attempt is made to exceed the capacity.

Each instruction is from 1 to 7 steps long. Refer to *SECTION 4 Instruction Execution Times and Number of Steps* for the specific number of steps in each instruction. (The length of each instruction will increase by 1 step if a double-length operand is used.)

Series	CPU Unit type	Model	Max. program capacity
CP Series CP1H CPU Units	XA	CP1H-XA40D□-□	20K steps
	X	CP1H-X40D□-□	
	Y	CP1H-Y20DT-D	
CP Series CP1L CPU Units	M	CP1L-M40D□-□	10K steps
		CP1L-M30D□-□	
	L	CP1L-L20D□-□	5K steps
		CP1L-L14D□-□	

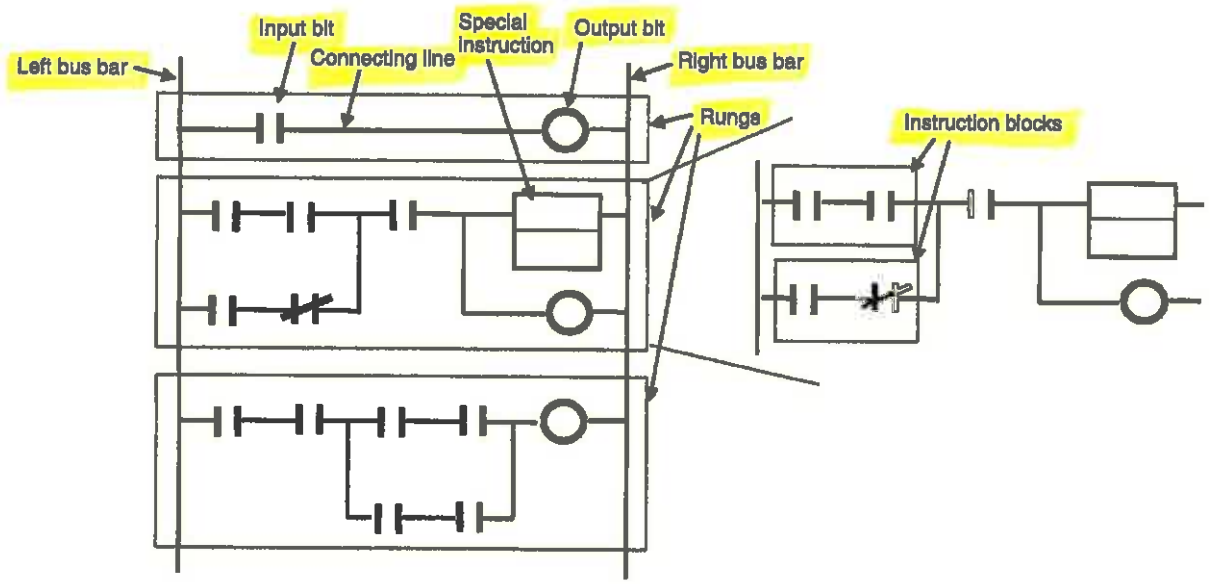
Note Memory capacity for CP-series PLCs is measured in steps, whereas memory capacity for previous OMRON PLCs, such as the C200HX/HG/HE and CV-series PLCs, was measured in words. Refer to the information at the end of *SECTION 4 Instruction Execution Times and Number of Steps* for guidelines on converting program capacities from previous OMRON PLCs.

1-1-12 Basic Ladder Programming Concepts

Instructions are executed in the order listed in memory (mnemonic order). The basic programming concepts as well as the execution order must be correct.

General Structure of the Ladder Diagram

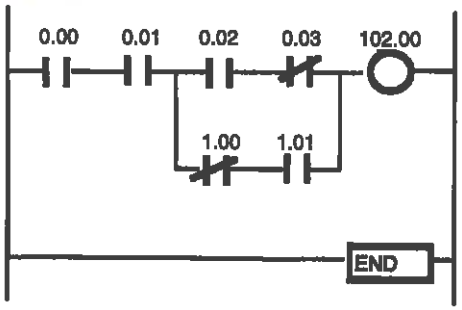
A ladder diagram consists of left and right bus bars, connecting lines, input bits, output bits, and special instructions. A program consists of one or more program runs. A program rung is a unit that can be partitioned when the bus is split horizontally. In mnemonic form, a rung is all instructions from a LD/LD NOT instruction to the output instruction just before the next LD/LD NOT instructions. A program rung consists of instruction blocks that begin with an LD/LD NOT instruction indicating a logical start.



Mnemonics

A mnemonic program is a series of ladder diagram instructions given in their mnemonic form. It has program addresses, and one program address is equivalent to one instruction.

Example



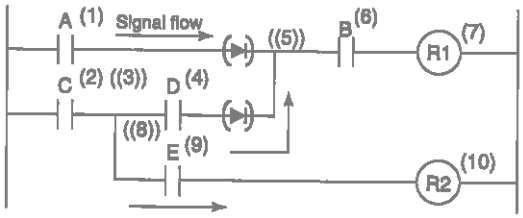
Program Address	Instruction (Mnemonic)	Operand
0	LD	0.00
1	AND	0.01
2	LD	0.02
3	AND NOT	0.03
4	LD NOT	1.00
5	AND	1.01
6	OR LD	
7	AND LD	
8	OUT	102.00
9	END	

Basic Ladder Program Concepts

1,2,3... 1. When ladder diagrams are executed by PLCs, the signal flow (power flow) is always from left to right. Programming that requires power flow from right to left cannot be used. Thus, flow is different from when circuits are made up of hard-wired control relays. For example, when the circuit "a" is implemented in a PLC program, power flows as though the diodes in brackets

were inserted and coil R2 cannot be driven with contact D included. The actual order of execution is indicated on the right with mnemonics. To achieve operation without these imaginary diodes, the circuit must be re-written. Also, circuit "b" power flow cannot be programmed directly and must be rewritten.

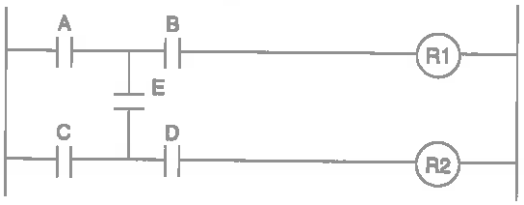
Circuit "a"



Order of execution (mnemonic)

(1) LD A	(6) AND B
(2) LD C	(7) OUT R1
(3) OUT TR0	(8) LD TR0
(4) AND D	(9) AND E
(5) OR LD	(10) OUT R2

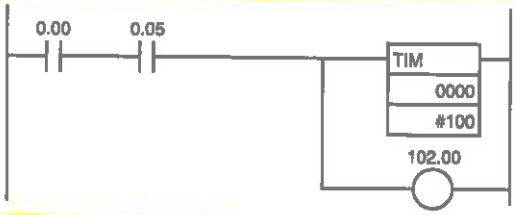
Circuit " b"



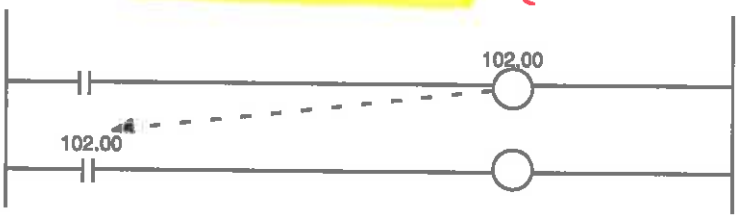
In circuit "a," coil R2 cannot be driven with contact D included.

In circuit "b," contact E included cannot be written in a ladder diagram. The program must be rewritten.

2. There is no limit to the number of I/O bits, work bits, timers, and other input bits that can be used. Rungs, however, should be kept as clear and simple as possible even if it means using more input bits to make them easier to understand and maintain.
3. There is no limit to the number of input bits that can be connected in series or in parallel in series or parallel rungs.
4. Two or more output bits can be connected in parallel.



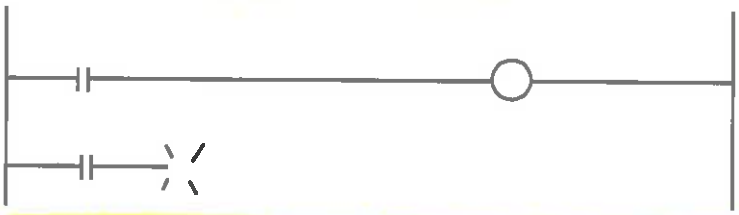
5. Output bits can also be used as input bits. (internal relay)



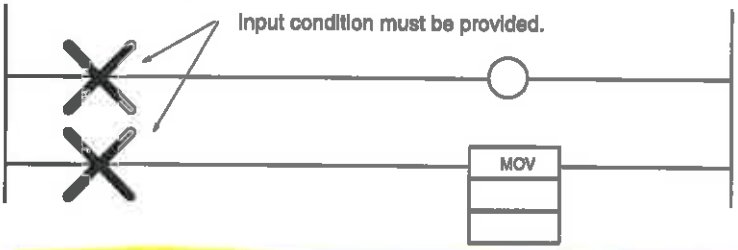
Restrictions

1,2,3...

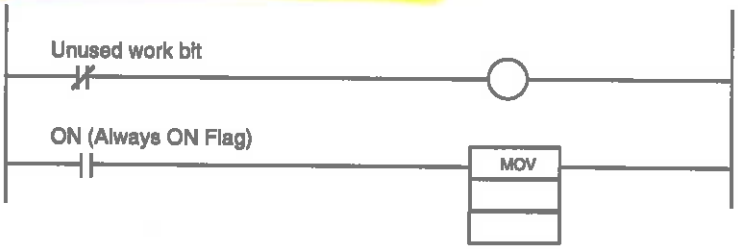
1. A ladder program must be closed so that signals (power flow) will flow from the left bus bar to the right bus bar. A rung error will occur if the program is not closed (but the program can be executed).



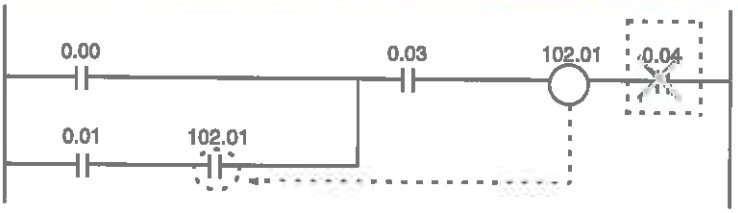
2. Output bits, timers, counters and other output instructions cannot be connected directly to the left bus bar. If one is connected directly to the left bus bar, a rung error will occur during the programming check by the CX-Programmer. (The program can be executed, but the OUT and MOV(021) will not be executed.)



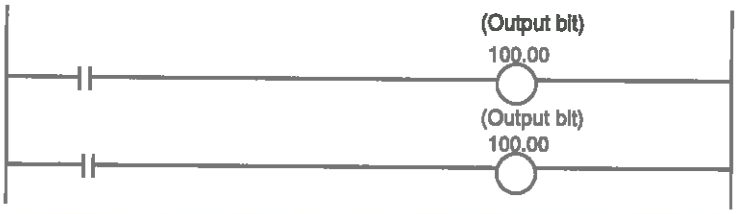
Insert an unused N.C. work bit or the ON Condition Flag (Always ON Flag) if the input must be kept ON at all times.



3. An input bit must always be inserted before and never after an output instruction like an output bit. If it is inserted after an output instruction, then a location error will occur during the CX-Programmer program check.

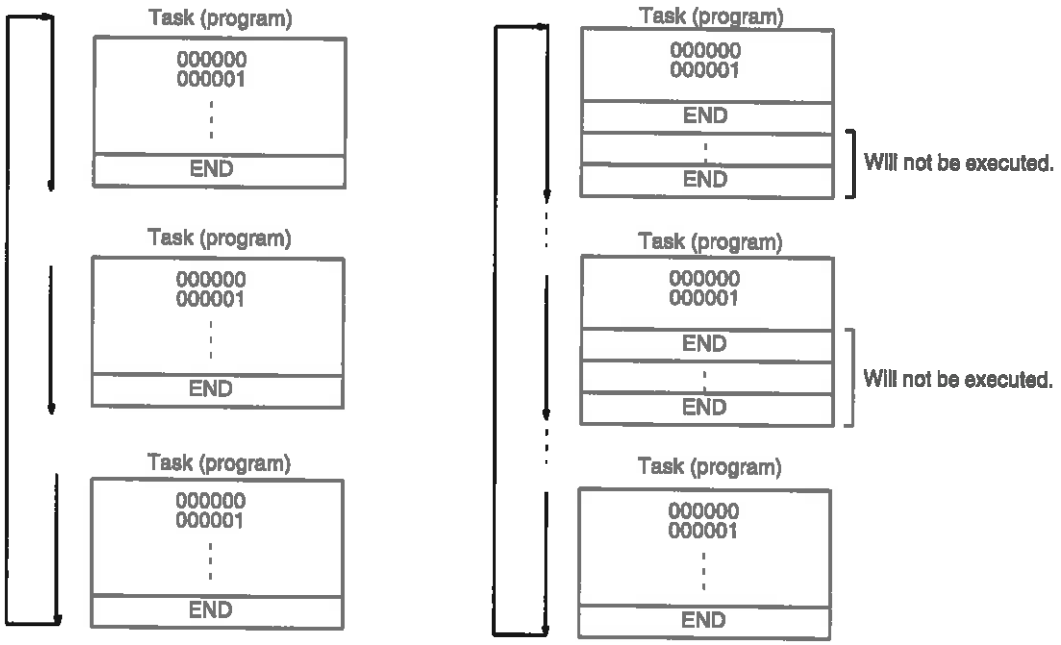


4. **The same output bit cannot be programmed in an output instruction more than once. Instructions in a ladder program are executed in order from the top rung in a single cycle, so the result of output instruction in the lower rungs will be ultimately reflected in the output bit and the results of any previous instructions controlling the same bit will be overwritten and not output.**



5. **An input bit cannot be used in an OUTPUT instruction (OUT).**

6. **An END(001) instruction must be inserted at the end of the program in each task.**
   - If a program without an END(001) instruction starts running, a program error indicating No End Instruction will occur, the ERR/ALM LED on the front of the CPU Unit will light, and the program will not be executed.
   - If a program has more than one END(001) instruction, then the program will only run until the first END(001) instruction.
   - Debugging programs will run much smoother if an END(001) instruction is inserted at various break points between sequence rungs and the END(001) instruction in the middle is deleted after the program is checked.

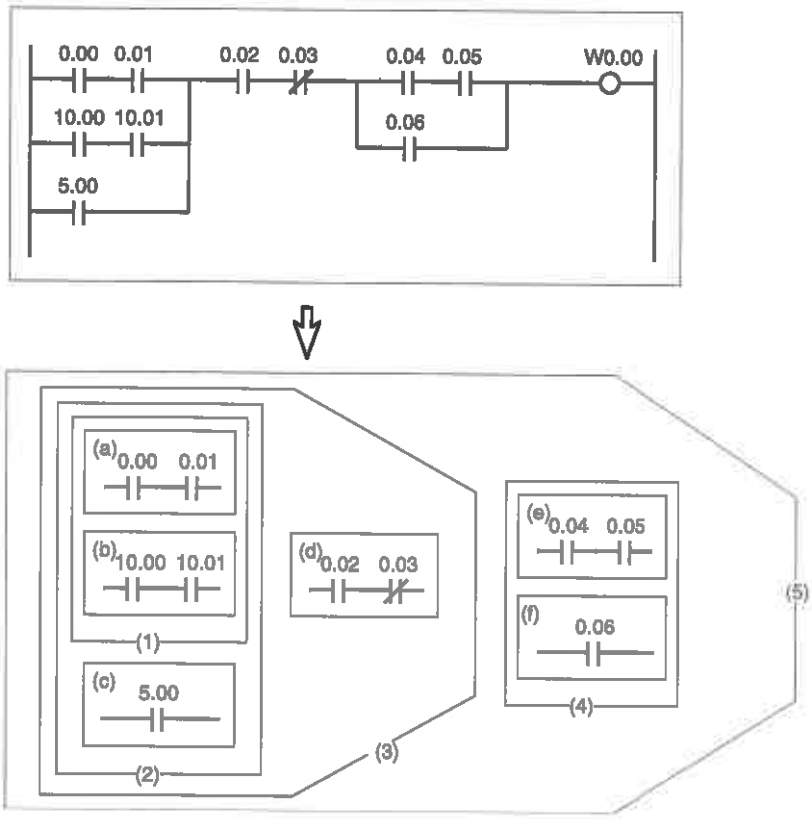


1-1-13 Inputting Mnemonics

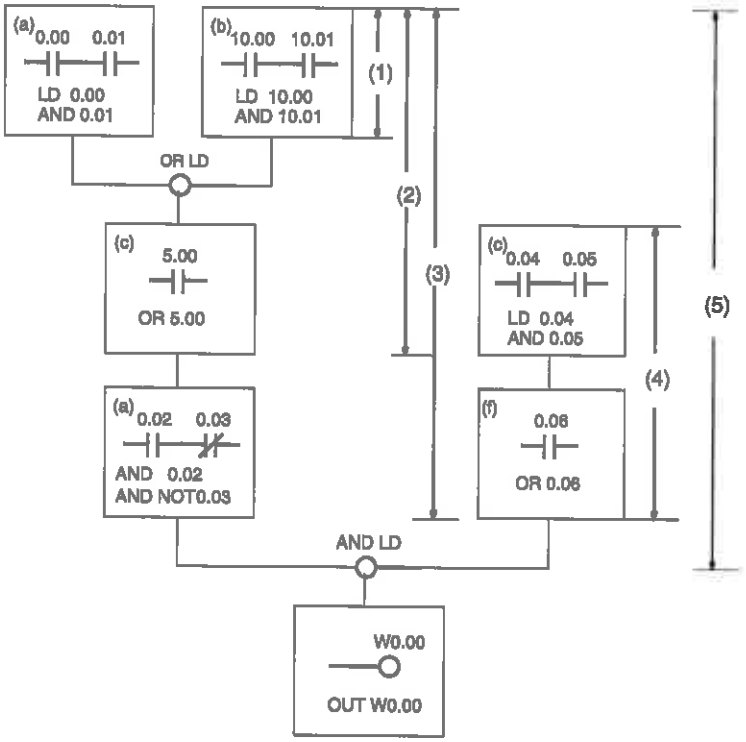
A logical start is accomplished using an LD/LD NOT instruction. The area from the logical start until the instruction just before the next LD/LD NOT instruction is considered a single instruction block.

Create a single rung consisting of two instruction blocks using an AND LD instruction to AND the blocks or by using an OR LD instruction to OR the blocks. The following example shows a complex rung that will be used to explain the procedure for inputting mnemonics (rung summary and order).

1,2,3... 1. First separate the rung into small blocks (a) to (f).



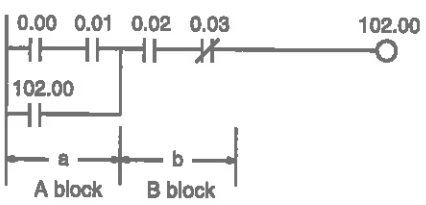
2. Program the blocks from top to bottom and then from left to right.



	Address	Instruction	Operand
(a)	200	LD	0.00
	201	AND	0.01
(b)	202	LD	10.00
	203	AND	10.01
	204	OR LD	---
(c)	205	OR	5.00
(d)	206	AND	0.02
	207	AND NOT	0.03
(e)	208	LD	0.04
	209	AND	0.05
(f)	210	OR	0.06
	211	AND LD	---
	212	OUT	W0.00

1-1-14 Program Examples

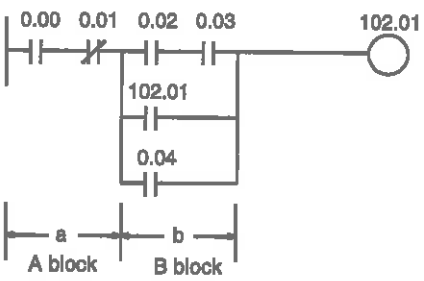
Parallel/Series Rungs



Instruction	Operands	
LD	0.00	a
AND	0.01	
OR	102.00	
AND	0.02	b
AND NOT	0.03	
OUT	102.00	

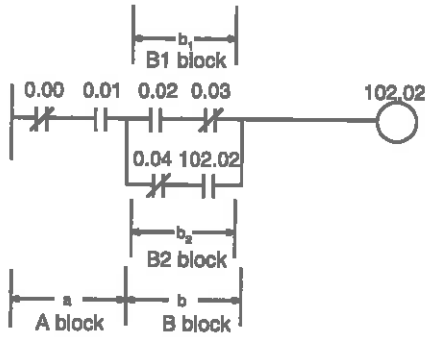
Program the parallel instruction in the A block and then the B block.

Series/Parallel Rungs



Instruction	Operands	
LD	0.00	a
AND NOT	0.01	
LD	0.02	b
AND	0.03	
OR	102.01	
OR	0.04	
AND LD	---	
OUT	102.01	

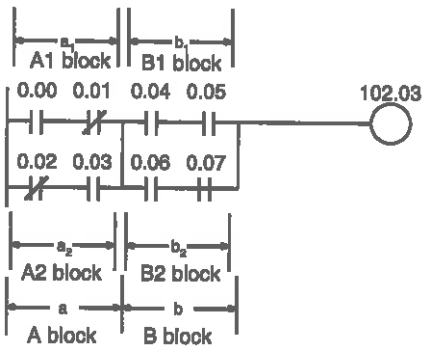
- Separate the rung into A and B blocks, and program each individually.
- Connect A and B blocks with an AND LD.
- Program A block.



Instruction	Operands	
LD NOT	0.00	a
AND	0.01	
LD	0.02	b_1
AND NOT	0.03	
LD NOT	0.04	
AND	102.02	b_2
OR LD	---	$b_1 + b_2$
AND LD	---	$a \cdot b$
OUT	102.02	

- Program B_1 block and then program B_2 block.
- Connect B_1 and B_2 blocks with an OR LD and then A and B blocks with an AND LD.

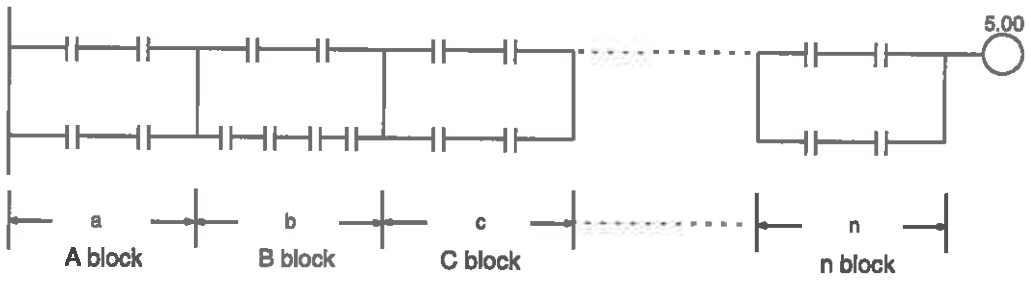
Example of Series Connection in a Series Rung



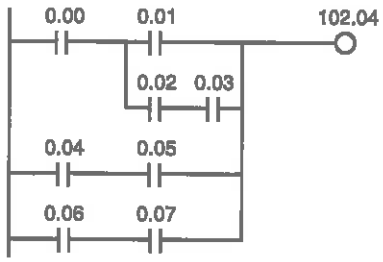
Instruction	Operands	
LD	0.00	a_1
AND NOT	0.01	
LD NOT	0.02	a_2
AND	0.03	
OR LD	---	$a_1 + a_2$
LD	0.04	b_1
AND	0.05	
LD	0.06	b_2
AND	0.07	
OR LD	---	$b_1 + b_2$
AND LD	---	$a \cdot b$
OUT	102.03	

- Program A_1 block, program A_2 block, and then connect A_1 and A_2 blocks with an OR LD.
- Program B_1 and B_2 the same way.
- Connect A block and B block with an AND LD.

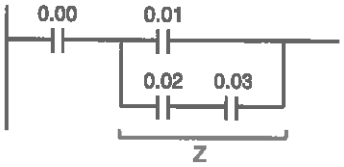
• Repeat for as many A to n blocks as are present.



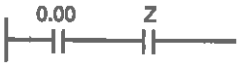
Complex Rungs



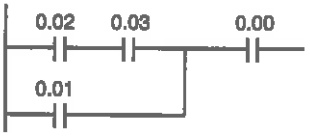
Instruction	Operand	
LD	0.00	
LD	0.01	
LD	0.02	Z
AND	0.03	
OR LD	---	
AND LD	---	
LD	0.04	
AND	0.05	
OR LD	---	
LD	0.06	
AND	0.07	
OR LD	---	
OUT	102.04	

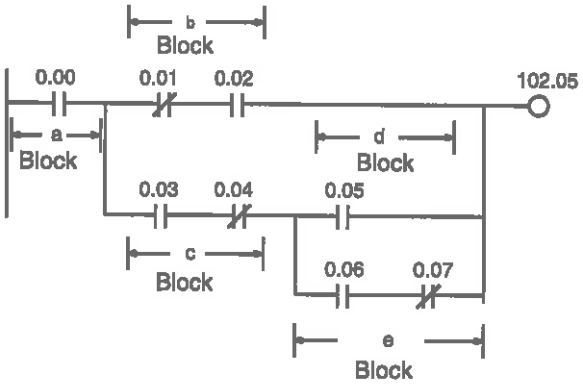


The diagram above is based on the diagram below.



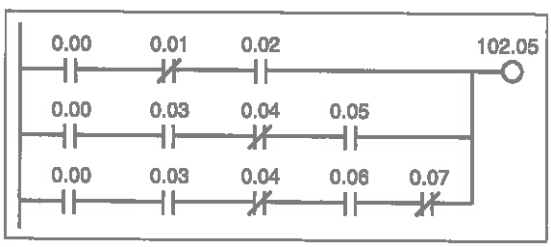
A simpler program can be written by rewriting this as shown below.

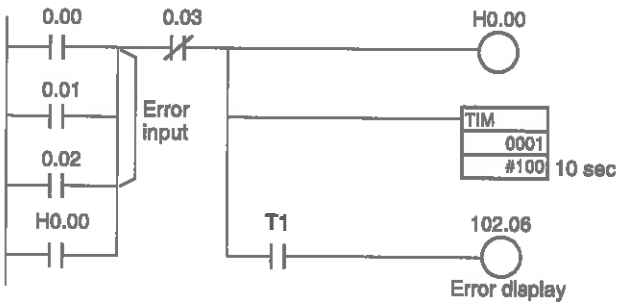




Instruction	Operand	
LD	0.00	a
LD NOT	0.01	b
AND	0.02	b
LD	0.03	c
AND NOT	0.04	c
LD	0.05	d
LD	0.06	d
AND NOT	0.07	e
OR LD	---	e
AND LD	---	d + e
OR LD	---	(d + e) · c
AND LD	---	(d + e) · c + b
OUT	102.05	((d + e) · c + b) · a

The above rung can be rewritten as follows:





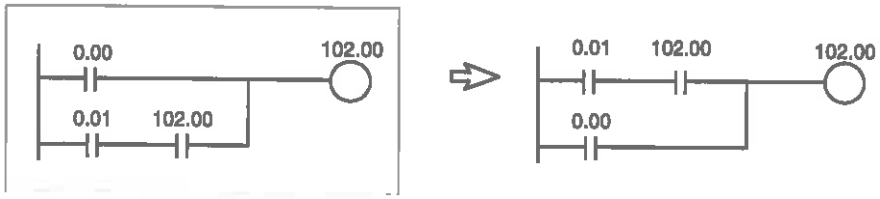
Instruction	Operand
LD	0.00
OR	0.01
OR	0.02
OR	H0.00
AND NOT	0.03
OUT	H0.00
TIM	0001
	#100
AND	T1
OUT	102.06

If a holding bit is in use, the ON/OFF status would be held in memory even if the power is turned OFF, and the error signal would still be in effect when power is turned back ON.

Rungs Requiring Caution or Rewriting

OR and OL LD Instructions

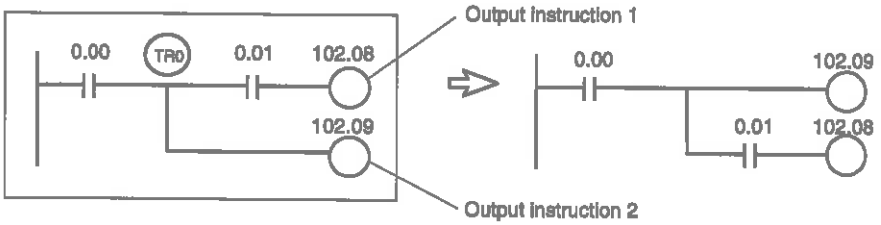
With an OR or OR NOT instruction, an OR is taken with the results of the ladder logic from the LD or LD NOT instruction to the OR or OR NOT instruction, so the rungs can be rewritten so that the OR LD instruction is not required.



Example: An OR LD instruction will be needed if the rungs are programmed as shown without modification. A few steps can be eliminated by rewriting the rungs as shown.

Output Instruction Branches

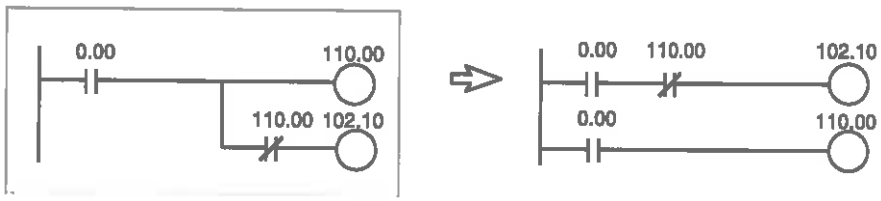
A TR bit will be needed if there is a branch before an AND or AND NOT instruction. The TR bit will not be needed if the branch comes at a point that is connected directly to output instructions and the AND or AND NOT instruction or the output instructions can be continued as is.



Example: A temporary storage bit TR0 output instruction and load (LD) instruction are needed at a branch point if the rungs are programmed without modification. A few steps can be eliminated by rewriting the rungs.

Mnemonic Execution Order

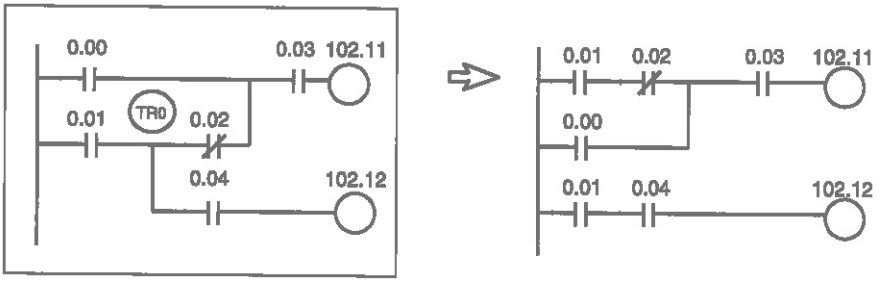
PLCs execute ladder programs in the order the mnemonics are entered so instructions may not operate as expected, depending on the way rungs are written. Always consider mnemonic execution order when writing ladder diagrams.



Example: CIO 102.10 in the above diagram cannot be output. By rewriting the rung, as shown above, CIO 102.10 can be turned ON for one cycle.

Rungs Requiring Rewriting

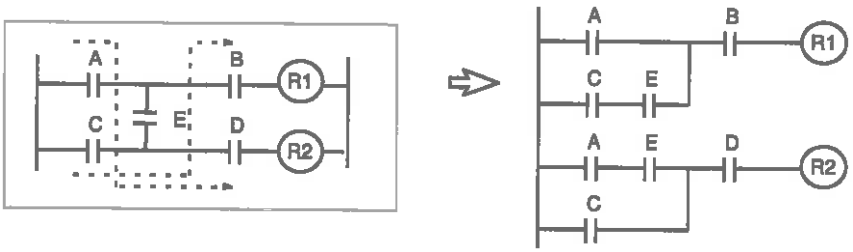
PLCs execute instructions in the order the mnemonics are entered so the signal flow (power flow) is from left to right in the ladder diagram. Power flows from right to left cannot be programmed.



Example: The program can be written as shown in the diagram at the left where TR0 receives the branch. The same value is obtained, however, by the rungs at the right, which are easier to understand. It is recommended, therefore, that the rungs at the left be rewritten to the rungs at the right.

Rewrite the rungs on the left below. They cannot be executed.

The arrows show signal flow (power flow) when the rungs consist of control relays.



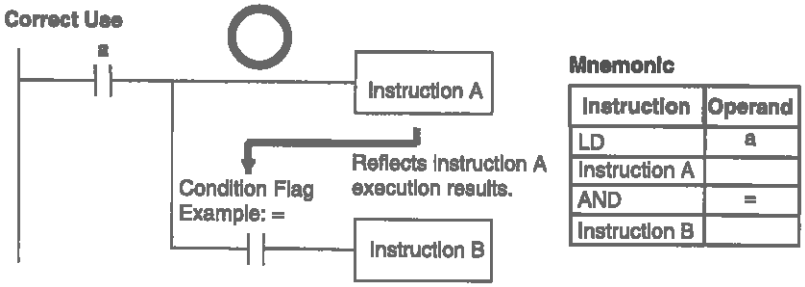
1-2 Precautions

1-2-1 Condition Flags

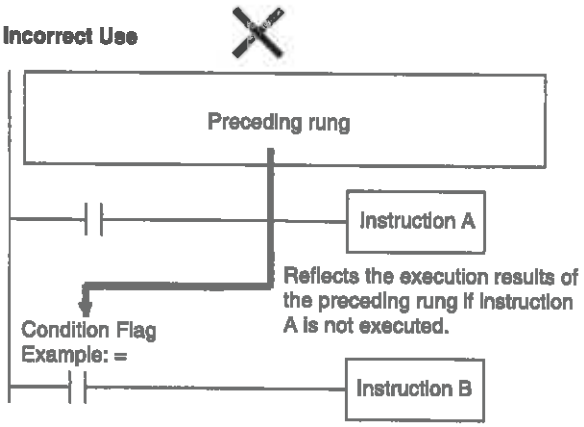
Using Condition Flags

Conditions flags are shared by all instructions, and will change during a cycle depending on results of executing individual instructions. Therefore, be sure to use Condition Flags on a branched output with the same execution condition immediately after an instruction to reflect the results of instruction execution. Never connect a Condition Flag directly to the bus bar because this will cause it to reflect execution results for other instructions.

Example: Using Instruction A Execution Results



The same execution condition (a) is used for instructions A and B to execute instruction B based on the execution results of instruction A. In this case, instruction B will be executed according to the Condition Flag only if instruction A is executed.

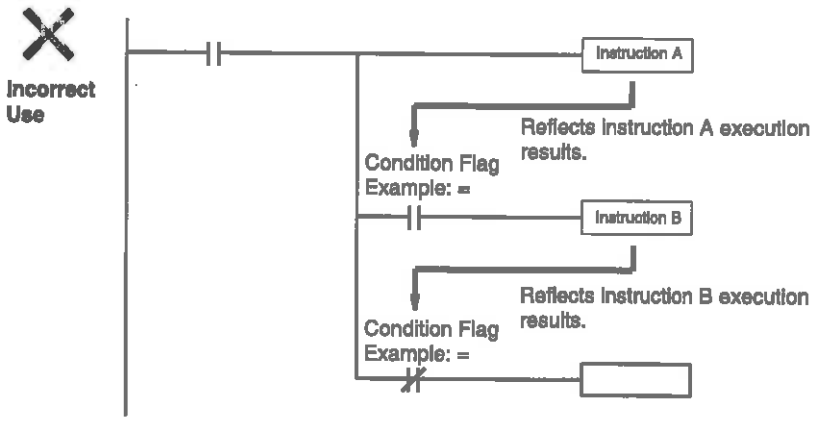


If the Condition Flag is connected directly to the left bus bar, instruction B will be executed based on the execution results of a previous rung if instruction A is not executed.

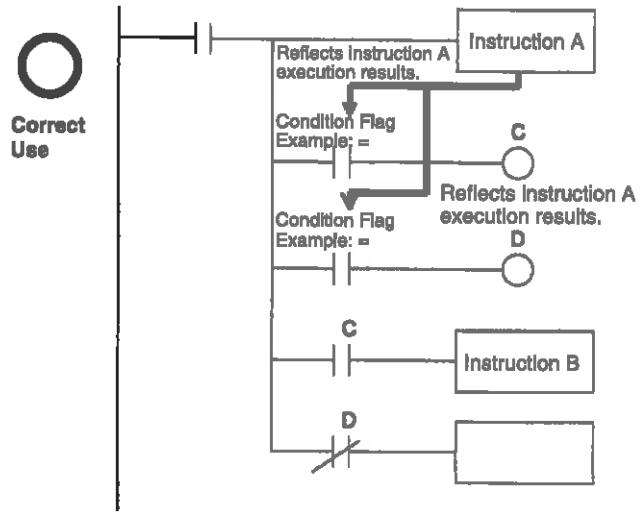
Note Condition Flags are used by all instruction within a single program (task) but they are cleared when the task switches. Therefore execution results in the preceding task will not be reflected later tasks. Since conditions flags are shared by all instructions, make absolutely sure that they do not interfere with each other within a single ladder-diagram program. The following is an example.

Using Execution Results in N.C. and N.C. Inputs

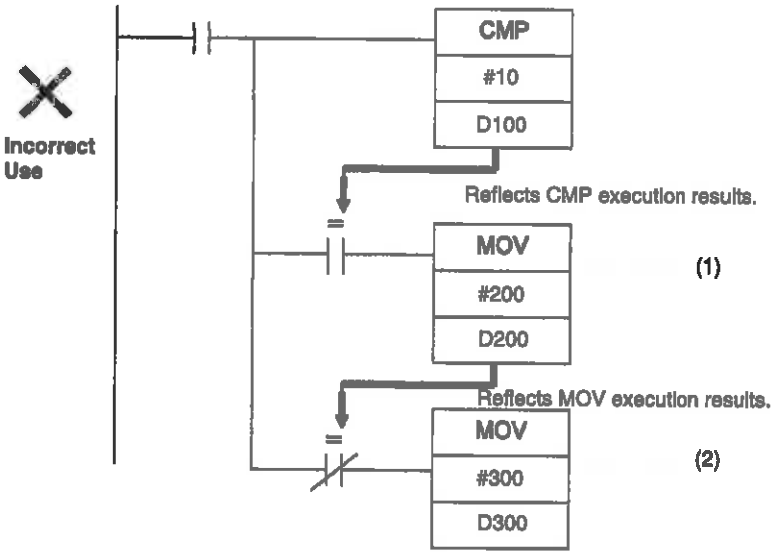
The Condition Flags will pick up instruction B execution results as shown in the example below even though the N.C. and N.O. input bits are executed from the same output branch.



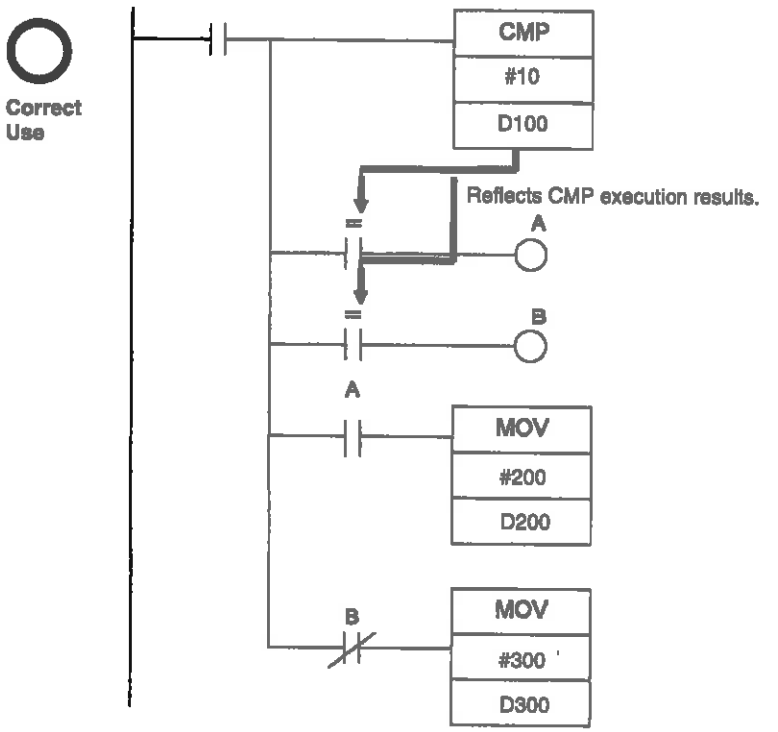
Make sure each of the results is picked up once by an OUTPUT instruction to ensure that execution results for instruction B will be not be picked up.



Example: The following example will move #200 to D200 if D100 contains #10 and move #300 to D300 if D100 does not contain #10.



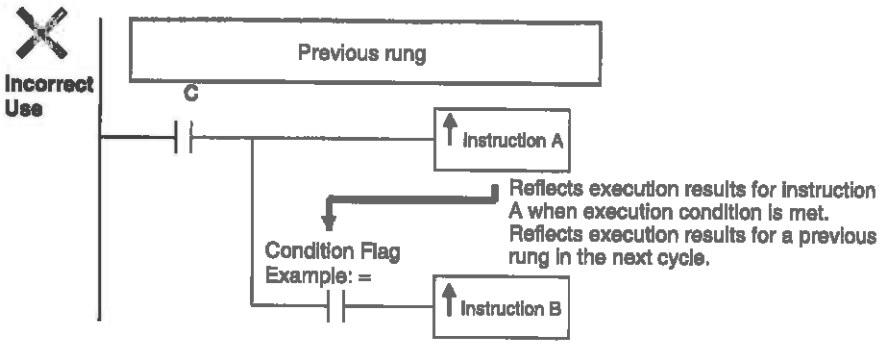
The Equals Flag will turn ON if D100 in the rung above contains #10. #200 will be moved to D200 for instruction (1), but then the Equals Flag will be turned OFF because the #200 source data is not 0000 hex. The MOV instruction at (2) will then be executed and #300 will be moved to D300. A rung will therefore have to be inserted as shown below to prevent execution results for the first MOVE instruction from being picked up.



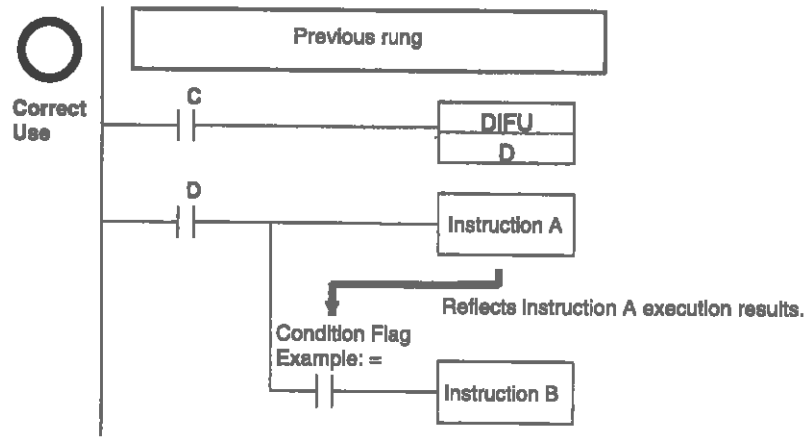
Using Execution Results from Differentiated Instructions

With differentiated instructions, execution results for instructions are reflected in Condition Flags only when execution condition is met, and results for a previous rung (rather than execution results for the differentiated instruction) will be reflected in Condition Flags in the next cycle. You must therefore be aware of what Condition Flags will do in the next cycle if execution results for differentiated instructions to be used.

In the following for example, instructions A and B will execute only if execution condition C is met, but the following problem will occur when instruction B picks up execution results from instruction A. If execution condition C remains ON in the next cycle after instruction A was executed, then instruction B will unexpectedly execute (by the execution condition) when the Condition Flag goes from OFF to ON because of results reflected from a previous rung.



In this case then, instructions A and B are not differentiated instructions, the DIFU (of DIFD) instruction is used instead as shown below and instructions A and B are both upwardly (or downwardly) differentiated and executed for one cycle only.



Note The CP1H CPU Units support instructions to save and load the Condition Flag status (CCS(282) and CCL(283)). These can be used to access the status of the Condition Flags at other locations in a task or in a different task.

Main Conditions Turning ON Condition Flags

Error Flag

The ER Flag will turn ON under special conditions, such as when operand data for an instruction is incorrect. The instruction will not be executed when the ER Flag turns ON.

When the ER Flag is ON, the status of other Condition Flags, such as the <, >, OF, and UF Flags, will not change and status of the = and N Flags will vary from instruction to instruction.

Refer to the descriptions of individual instructions in the *CP-series CP1H CPU Unit Programming Manual (W451)* for the conditions that will cause the ER Flag to turn ON. Caution is required because some instructions will turn OFF the ER Flag regardless of conditions.

Note The PLC Setup Settings for when an instruction error occurs determines whether operation will stop when the ER Flag turns ON. In the default setting, operation will continue when the ER Flag turns ON. If Stop Operation is specified when the ER Flag turns ON and operation stops (treated as a program error), the program address at the point where operation stopped will be stored at in A298 to A299. At the same time, A295.08 will turn ON.

Equals Flag

The Equals Flag is a temporary flag for all instructions except when comparison results are equal (=). It is set automatically by the system, and it will change. The Equals Flag can be turned OFF (ON) by an instruction after a previous instruction has turned it ON (OFF). The Equals Flag will turn ON, for example, when MOV or another move instruction moves 0000 hex as source data and will be OFF at all other times. Even if an instruction turns the Equals Flag ON, the move instruction will execute immediately and the Equals Flag will turn ON or OFF depending on whether the source data for the move instruction is 0000 hex or not.

Carry Flag

The CY Flag is used in shift instructions, addition and subtraction instructions with carry input, addition and subtraction instruction borrows and carries, as well as with Special I/O Unit instructions, PID instructions, and FPD instructions. Note the following precautions.

Note

(1) The CY Flag can remain ON (OFF) because of execution results for a certain instruction and then be used in other instruction (an addition and subtraction instruction with carry or a shift instruction). Be sure to clear the Carry Flag when necessary.

(2) The CY Flag can be turned ON (OFF) by the execution results for a certain instruction and be turned OFF (ON) by another instruction. Be sure the proper results are reflected in the Carry Flag when using it.

Less Than and Greater Than Flags

The < and > Flags are used in comparison instruction, as well as in the LMT, BAND, ZONE, PID and other instructions.
The < or > Flag can be turned OFF (ON) by another instruction even if it is turned ON (OFF) by execution results for a certain instruction.

Negative Flag

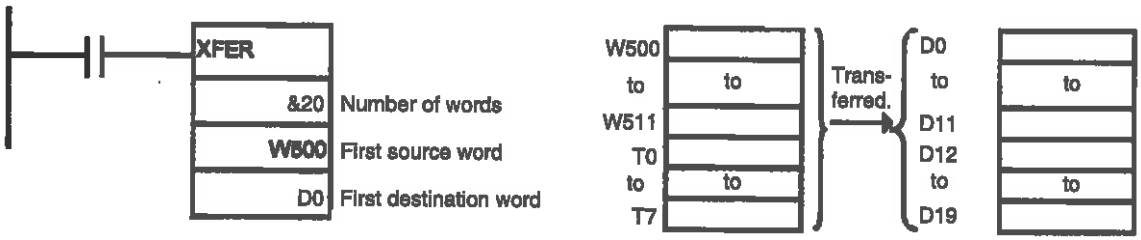
The N Flag is turned OFF when the leftmost bit of the instruction execution results word is "1" for certain instructions and it is turned OFF unconditionally for other instruction.

Specifying Operands for Multiple Words

With the CP-series PLCs, an instruction will be executed as written even if an operand requiring multiple words is specified so that all of the words for the operand are not in the same area. In this case, words will be taken in order of the PLC memory addresses. The Error Flag will **not** turn ON.

As an example, consider the results of executing a block transfer with XFER(070) if 20 words are specified for transfer beginning with W500. Here, the Work Area, which ends at W511, will be exceeded, but the instruction will be executed without turning ON the Error Flag. In the PLC memory addresses, the present values for timers are held in memory after the Work Area, and thus for the following instruction, W500 to W511 will be transferred to D0 to D11 and the present values for T0 to T7 will be transferred to D12 to D19.

Note For specific PLC memory addresses in CP1H CPU Units, refer to *Appendix E: Memory Map* in the *CP Series CP1H CPU Units Operation Manual (W450)*. For specific PLC memory addresses in CP1L CPU Units, refer to *Appendix E: Memory Map* in the *CP Series CP1L CPU Units Operation Manual (W462)*.



1-2-2 Special Program Sections

CP-series programs have special program sections that will control instruction conditions. The following special program sections are available.

Program section	Instructions	Instruction condition	Status
Subroutine	SBS, SBN and RET instructions	Subroutine program is executed.	The subroutine program section between SBN and RET instructions is executed.
IL - ILC section	IL and ILC instructions	Section is interlocked	The output bits are turned OFF and timers are reset. Other instructions will not be executed and previous status will be maintained.
Step Ladder section	STEP S instructions and STEP instructions		
FOR-NEXT loop	FOR instructions and NEXT instructions	Break in progress.	Looping
JMP0 - JME0 section	JMP0 instructions and JME0 instructions		Jump
Block program section	BPRG instructions and BEND instructions	Block program is executing.	The block program listed in mnemonics between the BPRG and BEND instructions is executed.

Instruction Combinations

The following table shows which of the special instructions can be used inside other program sections.

	Subroutine	IL - ILC section	Step ladder section	FOR - NEXT loop	JMP0 - JME0 section	Block program section
Subroutine	Not possible.	Not possible.	Not possible.	Not possible.	Not possible.	Not possible.
IL - ILC	OK	Not possible.	Not possible.	OK	OK	Not possible.
Step ladder section	Not possible.	OK	Not possible.	Not possible.	OK	Not possible.
FOR - NEXT loop	OK	OK	Not possible.	OK	OK	Not possible.
JMP0 - JME0	OK	OK	Not possible.	Not possible.	Not possible.	Not possible.
Block program section	OK	OK	OK	Not possible.	OK	Not possible.

Note Instructions that specify program areas cannot be used for programs in other tasks. Refer to *2-2-2 Task Instruction Limitations* for details.

Subroutines

Place all the subroutines together just before the END(001) instruction in all programs but after programming other than subroutines. (Therefore, a subroutine cannot be placed in a step ladder, block program, FOR - NEXT, or JMP0 - JME0 section.) If a program other than a subroutine program is placed after a subroutine program (SBN to RET), that program will not be executed.

Instructions Not Available in Subroutines

The following instructions cannot be placed in a subroutine.

Function	Mnemonic	Instruction
Process Step Control	STEP(008)	Define step ladder section
	SNXT(009)	Step through the step ladder

Note **Block Program Sections**
A subroutine can include a block program section. If, however, the block program is in WAIT status when execution returns from the subroutine to the main program, the block program section will remain in WAIT status the next time it is called.

Instructions Not Available in Step Ladder Program Sections

Function	Mnemonic	Instruction
Sequence Control	FOR(512), NEXT(513), and BREAK(514)	FOR, NEXT, and BREAK LOOP
	END(001)	END
	IL(002) and ILC(003)	INTERLOCK and INTERLOCK CLEAR
	JMP(004) and JME(005)	JUMP and JUMP END
	CJP(510) and CJPN(511)	CONDITIONAL JUMP and CONDITIONAL JUMP NOT
	JMP0(515) and JME0(516)	MULTIPLE JUMP and MULTIPLE JUMP END
Subroutines	SBN(092) and RET(093)	SUBROUTINE ENTRY and SUBROUTINE RETURN

Function	Mnemonic	Instruction
Block Programs	IF(802) (NOT), ELSE(803), and IEND(804)	Branching Instructions
	BPRG(096) and BEND(801)	BLOCK PROGRAM BEGIN/END
	EXIT(806) (NOT)	CONDITIONAL BLOCK EXIT (NOT)
	LOOP(809) and LEND(810) (NOT)	Loop control
	WAIT(805) (NOT)	ONE CYCLE WAIT (NOT)
	TIMW(813) and TIMWX(816)	TIMER WAIT
	TMHW(815) and TMHWX(817)	HIGH-SPEED TIMER WAIT
	CNTW(814) and CNTWX(818)	COUNTER WAIT
	BPPS(811) and BPRS(812)	BLOCK PROGRAM PAUSE and RESTART

Note

(1) A step ladder program section can be used in an interlock section (between IL and ILC). The step ladder section will be completely reset when the interlock is ON.

(2) A step ladder program section can be used between MULTIPLE JUMP (JMP0) and MULTIPLE JUMP END (JME0).

Instructions Not Supported in Block Program Sections

The following instructions cannot be placed in block program sections.

Classification by Function	Mnemonic	Instruction
Sequence Control	FOR(512), NEXT(513), and BREAK(514)	FOR, NEXT, and BREAK LOOP
	END(001)	END
	IL(002) and ILC(003)	INTERLOCK and INTERLOCK CLEAR
	JMP0(515) and JME0(516)	MULTIPLE JUMP and MULTIPLE JUMP END
Sequence Input	UP(521)	CONDITION ON
	DOWN(522)	CONDITION OFF
Sequence Output	DIFU	DIFFERENTIATE UP
	DIFD	DIFFERENTIATE DOWN
	KEEP	KEEP
	OUT	OUTPUT
	OUT NOT	OUTPUT NOT
Timer/Counter	TIM and TIMX(550)	TIMER
	TIMH(015) and TIMHX(551)	HIGH-SPEED TIMER
	TMHH(540) and TMHHX(552)	ONE-MS TIMER
	TTIM(087) and TTIMX(555)	ACCUMULATIVE TIMER
	TIML(542) and TIMLX(553)	LONG TIMER
	MTIM(543) and MTIMX(554)	MULTI-OUTPUT TIMER
	CNT and CNTX(546)	COUNTER
	CNTR(012) and CNTRX(548)	REVERSIBLE COUNTER

Classification by Function	Mnemonic	Instruction
Subroutines	SBN(092) and RET(093)	SUBROUTINE ENTRY and SUBROUTINE RETURN
Data Shift	SFT	SHIFT
Ladder Step Control	STEP(008) and SNXT(009)	STEP DEFINE and STEP START
Data Control	PID	PID CONTROL
Block Program	BPRG(096)	BLOCK PROGRAM BEGIN
Damage Diagnosis	FPD(269)	FAILURE POINT DETECTION
Instructions with a differentiation option	@XXX	Instruction with upward differentiation
	%XXX	Instruction with downward differentiation

Note

(1) Block programs can be used in a step ladder program section.

(2) A block program can be used in an interlock section (between IL and ILC). The block program section will not be executed when the interlock is ON.

(3) A block program section can be used between MULTIPLE JUMP (JMP0) and MULTIPLE JUMP END (JME0).

(4) A JUMP instruction (JMP) and CONDITIONAL JUMP instruction (CJP/CJPN) can be used in a block program section. JUMP (JMP) and JUMP END (JME) instructions, as well as CONDITIONAL JUMP (CJP/CJPN) and JUMP END (JME) instructions cannot be used in the block program section unless they are used in pairs. The program will not execute properly unless these instructions are paired.

1-3 Checking Programs

CP-series programs can be checked at the following stages.

- Input check during CX-Programmer input and other operations
- Program check by CX-Programmer
- Instruction check during execution
- Fatal error check (program errors) during execution

1-3-1 CX-Programmer

The program will be automatically checked by the CX-Programmer at the following times.

Timing	Checked contents
When inputting ladder diagrams	Instruction inputs, operand inputs, programming patterns
When loading files	All operands for all instructions and all programming patterns
When downloading files	Models supported by the CP Series and all operands for all instructions
During online editing	Capacity, etc.

The results of checking are output to the text tab of the Output Window. Also, the left bus bar of illegal program sections will be displayed in red in ladder view.