공장자동화에 있어서 프로그램식 제어기의 역할

(2)

3. PC 프로그래밍

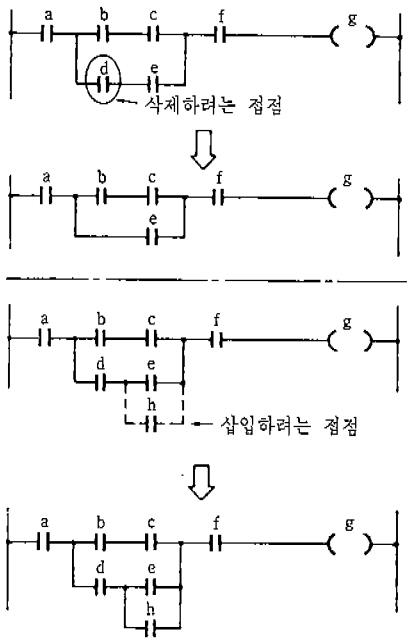
PC(Programmable Controller)는 프로그래밍을 간단히 할 수 있는 컨트롤러이며 프로그래밍 방법의 良否가 PC의 가치를 결정할 정도로 PC를 선택하는데 중요한 포인트가 되고 있다.

PC의 프로그래밍을 위한 필요한 조건에는 첫째, 단순한 프로그램 언어만이 문제가 아니고 프로그램 입출력 기기에서 프로그램의 수정 기능이나 프로그램의 보존성 그리고 각 분야의 사람들이 PC 프로그램을 관계하기 때문에 누구나 간단하게 이해되는 프로그램 언어가 되어야 한다.

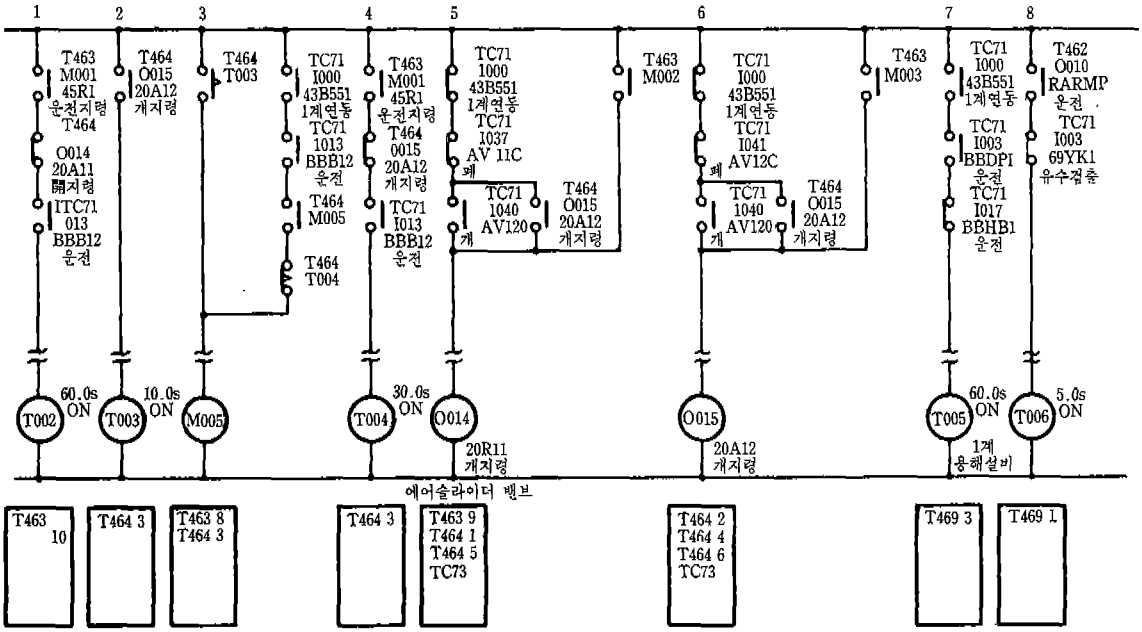
둘째, 특별한 컴퓨터 지식이나 숙련을 필요로 하지 않으며 이해하는데 시간이 걸려서는 안된다.

또한 다른 사람이 프로그램만 보더라도 어떠한 제어를 하고 있는가를 쉽게 판별할 수 있는 객관성이 필요하다.

셋째, 프로그램 입출력기기에서 프로그램 작성과 변경이 쉬워야 한다. 예를 들면 래더 다이어그램방식(Ladder Diagram Type)인 경우에 엘리먼트(접점이나 코일) 단위의 직렬삽입, 병렬삽입, 삭제 또는 블록단위의 삽입과 삭제를 자유롭고 쉽게 할 수 있어야 한다(그림14 참조).



<그림14> 프로그램의 변경



<그림15> 전개접속도의 출력예

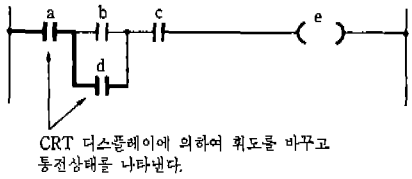
넷째, 프로그래밍상의 제약이 적어야 한다. 예를 들면 한 개의 코일에 대한 접점수가 극히 제한되거나 병렬회로를 취하는 방법에 제약이 있으면 내부 코일이 증가하고 결과적으로는 객관성이 없는 프로그램이 된다. 제약은 많거나 적거나 반드시 있으므로 주의할 필요가 있다.

다섯째, PC의 제어에 있어서 프로그래밍의 변경은 부수적인 것이며 변경시 사용자로서는 생산성의 관점에서 가능한한 설비의 운전을 정지시키는 일없이 제어동작을 변경할 수 있어야 한다.

이를 위해서는 약간의 변경으로 온라인 동작중에도 프로그램을 변경할 수 있어야 하나 대폭적인 변경이면 PC를 온라인 동작중에 변경하는 것은 좋은 일이 아니다.

그리고 프로그램을 수정하는 동안 PC를 정지시키는 것은 생산을 저하시키므로 프로그램 입출력 기기에 본체와 동일한 메모리를 갖게 하거나 부가시켜야 한다.

또한 PC 본체와 독립적으로 프로그램을 수정하고 메모리의 내용을 PC 본체의 메모리에 전송하는 짧은 시간동안만 PC를 정지하면 된다.



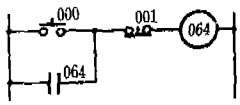
<그림16> 모니터링

여섯째, 입력한 프로그램을 래더프린터 등으로 입력했을 때와 같게 하거나 또는 그 이상의 표현법으로 출력할 필요가 있다.

이것은 도면 관리상 항상 PC에 입력되어 있는 프로그램과 동일한 도면이 필요하기 때문이며 최근에는 디바이스명(예를 들면 검출기의 번호와 용도명 등)의 데이터를 별도로 입력해서 프로그램과 디바이스명이 함께 합성되어 출력하거나 프로그램의 편집을 자유로이 할 수 있는 것도 있다 (그림15 참조). 이외에 온라인 동작중에 모니터링

을 할 수 있어야 한다(그림16 참조).

일곱째, PC 본체에 대한 불의의 사고나 프로그램 입출력 기기의 오동작으로 인하여 PC 본체의 메모리 내용이 파괴되는 경우를 고려하고 보수관계상 프로그램을 PC의 메모리 이외에 보존시킬 필요가 있다.



(a) PC용 시퀀스

어드레스	명 령	데이터
000	LD	000
001	OR	064
002	AND NOT	001
003	OUT	064

(b) 코딩시트

<그림18> 코딩의 예

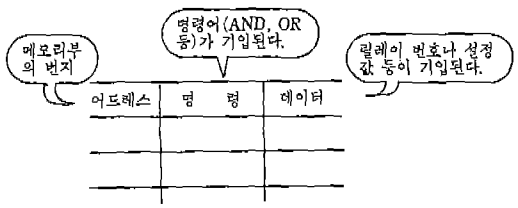
3·1 코 딩

릴레이 시퀀스가 전자계전기의 On-Off(勵磁, 消磁)에 의하여 제어를 하고 있는데 반하여 PC는 CPU 내부에서 회로를 논리적으로 제어할 수 있도록 메모리부(프로그램 기억부)에 제어방법을 기억시켜 두어야 한다.

제어 내용을 메모리부에 기억시키기 위하여는 프로그래밍 콘솔에서 키를 조작하여 프로그램을 입력하여야 한다.

그리고 PC용 시퀀스회로에 따라 메모리부에 어떤 어드레스번호로 각 프로그램을 기억시킬지를 결정하는 것을 코딩(Coding)이라고 하며 이때 사용하는 용지를 코딩시트라고 한다.

코딩은 그림17과 같이 어드레스, 명령어 데이터를 기입하게 되어 있으며 어드레스는 메모리의 수에 의하여 어느 위치에 어떠한 명령을 저장하는가라는 메모리상의 번호를 나타내는데 이것을 스텝번호(Step Number)라고도 한다. 프로그램할 때는 코딩시트에 기입하여 하는데 이는 변경, 추가, 삭제 때에 매우 유용하다. 그림18은 PC용 시퀀스회로와 코딩 예를 나타낸다.



<그림17> 코딩시트

3·2 명령어

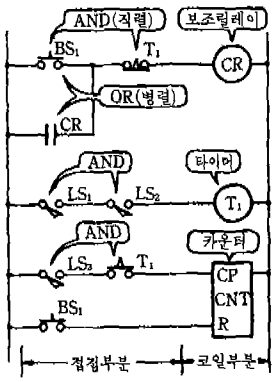
그림17의 코딩시트에 명령란이 있으며 여기에 명령어가 기입되는데 여기에 어떤 것이 있는가를 설명한다.

명령어란 PC가 갖고 있는 독자적인 것이며 정확하게 기억시키기 위해서는 각각의 취급설명서를 읽어야 한다(표3 참조). 표3에 있는 No.1의 로드명령은 LD로 표시되어 논리적인 스타트를 나타내는데 메이커가 다르면 LOD, SG 등으로 되기 때문에 주의하여야 한다.

또한 어느 것으로 하든지간에 좌측 제어모선에서 직접 인출되는 명령의 의미가 된다.

3·3 기본회로

일반적으로 제어는 기본회로의 조합으로 성립되어 있다. 즉 AND, OR, NOT, TIM(타이머),



<그림19> 기본회로

<표 3 > 대표적인 명령어

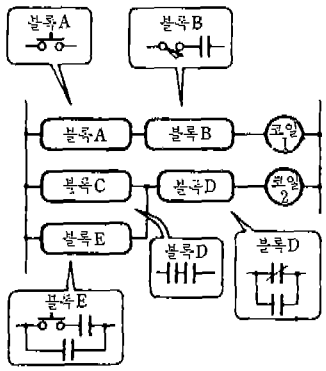
No.	명 령	기 호	기 능	데 이 터
1	로드	LD	논리스타트를 나타낸다.	입출력 릴레이 000~063
2	로드 NOT	LD NOT	논리부정스타트를 나타낸다.	보조릴레이 064~103
3	AND	AND	논리곱 조건으로 접속되어 있는 것을 나타낸다.	특수보조릴레이 104~111
4	AND NOT	AND NOT	논리곱 부정조건으로 접속되어 있는 것을 나타낸다.	타이머 TIM 0~7
5	OR	OR	논리곱 조건으로 접속되어 있는 것을 나타낸다.	카운터 CNT 0~7
6	OR NOT	OR NOT	논리합 부정조건으로 접속되어 있는 것을 나타낸다.	킵릴레이 KR 0~7
7	AND 로드	AND LD	앞의 조건과 논리곱을 한다.	고속카운터 출력 HDM 00~31
8	AND 로드	OR LD	앞의 조건과 논리합을 한다.	가역카운터 출력 RDM 00~31
9	출력	OUT	논리연산처리한 결과를 출력릴레이, 내부 보조릴레이, 킵릴레이, 시프트 레지스터의 비트로 출력	입출력 릴레이 012~063 보조릴레이 064~103 특수보조릴레이 104
10	타이머	TIM	온딜레이 타이머의 동작을 나타낸다.	타이머 TIM 0~7
11	카운터	CNT	감산카운터의 동작을 나타낸다.	카운터 CNT 0~7
12	킵릴레이	KR	킵릴레이의 동작을 나타낸다.	킵릴레이 KR 0~7
13	고속카운터	HDM	고속가산카운터의 동작을 나타낸다.	–
14	가역카운터	RDM	가역카운터의 동작을 나타낸다.	–
15	인터록	IL	본 명령전의 결과에 따라 IL~IL, END 명령 사이의 코일이 리세트 되거나 되지 않거나 한다.	–
16	인터록엔드	IL END	IL 명령을 해제한다.	–
17	엔드	END	프로그램이 끝난 것을 나타낸다.	–

CNT(카운터), KR(킵 릴레이)를 말하며 그림19와 같이 접점부분은 병렬 또는 직렬접속으로 되어 있다.

병렬접속은 OR이며 직렬 접속은 AND로서 a 접점은 그대로이고 b접점은 NOT를 붙인다. 코일 부분을 보면 CR은 내부 보조릴레이, T는 타이머, CNT는 카운터이며 접점부분에 의한 AND, OR, NOT 등으로 작성된 신호가 코일부분에 접속되어 있다. 이와 같이 기본회로의 조합으로 전체가 구성되어 있다.

다음에는 그림20과 같이 복잡한 제어회로를 프로그램할 때 필요한 블록에 대하여 설명한다.

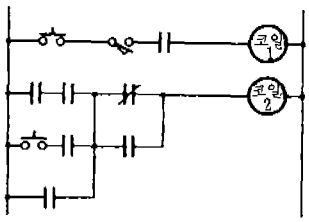
PC용 시퀀스에 한정되지 않고 좌측 제어모선부터 출발하여 우측 제어모선으로 접속된 코일까지



<그림20> 제어블록

를 1回路라고 하며 이것은 그림20과 같이 각 블록으로 나누어져 각종 접점이 접속되어 있다.

코일1은 블록의 AND 접속으로 되어 있으며



<그림21> 시퀀스회로

코일2는 블록C와 E가 병렬로 되어 있다. 그 뒤에 블록D가 직렬로 되어 있고 블록 사이는 OR과 AND로 구성되어 있다.

각 블록의 내용은 접점으로 구성되어 있으나 접점이 한 개뿐인 경우도 있고 여러 개인 것도 있으며 또한 이것이 직렬(AND) 또는 병렬로 되는 것도 있으나 그 내용까지는 명시되어 있지 않다.

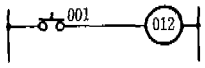
그림20의 회로는 5개의 블록으로 나누어져 있으며 전체를 종합해 보면 그림21과 같이 되는데 그림20과 같은 블록으로 나누어지는 능력이 중요하다.

다음에 구체적인 명령어에 대하여 설명한다.

(1) 로드(LD)/아웃(OUT) 명령

LD 명령은 각 블록의 시작(논리스타트)을 나타내고 OUT 명령은 코일(출력 릴레이, 내부 보조릴레이, 증설 입출력 릴레이)을 나타낸다.

그림22는 입력단자 "001"에 a접점이 접속된 버튼스위치가 폐회로가 되었을 때에 출력 릴레이 "002"를 동작시키려는 프로그램이며 그림23은 입력단자 "001"에 a접점이 접속된 버튼스위치가

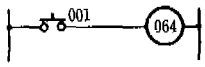


(a) 시퀀스

어드레스	명 령	데이터
0 0	LD	0 0 1
0 1	OUT	0 1 2

(b) 코딩

<그림22> OUT 명령 (1)

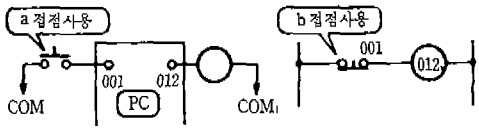


(a) 시퀀스

어드레스	명 령	데이터
0 0	LD	0 0 1
0 1	OUT	0 6 4

(b) 코딩

<그림23> OUT 명령 (2)



(a) I/O접속 (b) 시퀀스

어드레스	명 령	데이터
0 0	LD NOT	0 0 1
0 1	OUT	0 1 2

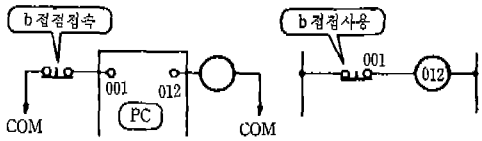
(c) 코딩

<그림24> LD 명령 (1)

폐회로가 되었을 때에 내부 보조릴레이 "064"를 여자시키지 않는다는 프로그램이다.

이상의 2가지 예와 같이 코일은 출력 릴레이든 아니든간에 OUT 명령을 사용하며 그림24의 예는 입력단자 "001"에 a접점이 접속된 버튼스위치로서 회로도상에서 b접점이 사용되고 있을 때는 LD·NOT이라고 코딩한다.

버튼스위치가 b접점에 접속되어 있고 회로도

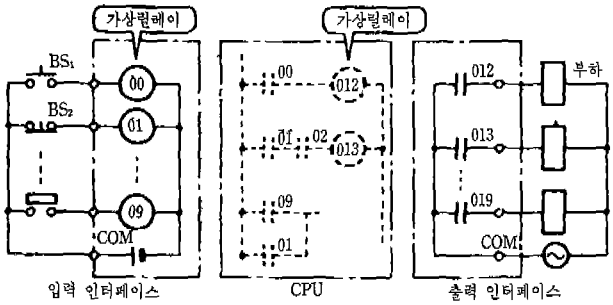


(a) I/O접속 (b) 시퀀스

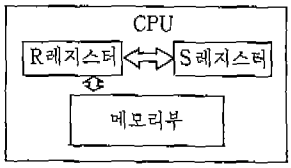
어드레스	명 령	데이터
0 0	LD	0 0 1
0 1	OUT	0 1 2

(c) 코딩

<그림25> LD 명령 (2)



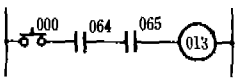
<그림26> 假想 릴레이



<그림29> 레지스터

상에도 b접점이 사용되고 있을 때에는 LD라고 코딩하여야 하는데 이것은 그림26과 같이 입출력 릴레이와 함께 가상 릴레이를 상정하고 있기 때문이며, 그림25와 그림26을 합쳐서 생각하면 b접점이 접속되어 있는 입력단자는 신호가 들어와 있으므로 가상 릴레이는 ON이 된다.

시퀀스회로에서 b접점이 사용되고 있는 개소는 ON으로 되어 있어야 하기 때문에 입력상태를 CPU에 도입하기 위해서는 LD 명령이며 LD NOT 명령은 아니다.

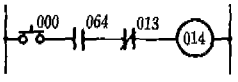


(a) 시퀀스

어드레스	명 령	데이터
0 0	LD	0 0 0
0 1	AND	0 6 4
0 2	AND	0 6 5
0 3	OUT	0 1 3

(b) 코딩

<그림27> AND 명령



(a) 시퀀스

어드레스	명 령	데이터
0 0	LD	0 0 0
0 1	AND	0 6 4
0 2	AND NOT	0 1 3
0 3	OUT	0 1 4

(b) 코딩

<그림28> AND·NOT 명령

한편 좌측 제어모선에서 출발하여 우측모선의 코일까지가 1회로이며 이러한 회로는 LD 명령으로 시작하고 OUT 명령으로 끝난다.

회로를 프로그램할 때는 LD 명령 OUT 명령이 원칙이 되며 이외에 LD 명령으로 시작해서 타이머 코일, 카운터 코일, 킵 릴레이 코일로 끝나는 것도 원칙적으로 포함된다.

(2) 앤드(AND) 명령

데이터에서 지정된 릴레이 번호의 내용(ON 또는 OFF)과 R레지스터의 내용(ON 또는 OFF)에 대한 논리곱을 취하고 그 결과를 새로이 R레지스터에 저장하는 명령이다.

이러한 예를 그림27, 28에 나타내었으며 그림27은 3개의 접점이 직렬회로이기 때문에 모든 것을 AND 명령으로 코딩하는데 처음의 "000"은 논리 스타트이므로 LD 명령으로 시작한다.

그림28은 직렬접점 안에 b접점이 포함된 예인데 b접점인 경우에는 AND·NOT이라고 프로그램한다. 그런데 CPU에는 그림29와 같이 R레지스터, S레지스터, 메모리라고 하는 3개의 프로그램이나 연산결과를 저장하는 장소가 있다.

R레지스터의 작용을 그림29를 기본으로 하여 표4로서 설명한다. 처음의 LD 명령으로 R레지스터에는 입력단자 "000"의 내용(ON 또는 OFF)이 저장된다.

다음의 AND 명령으로 내부 보조릴레이 "064"

<표 4 > AND 명령

어드레스	명 령	데이터	의 미	R레지스터
0 0	LD	0 0 0	입력단자 "000"의 내용을 R레지스터에 저장	000
0 1	AND	0 6 4	R레지스터와 "064"의 논리곱을 취하고 그 결과를 R레지스터에 저장	000 064
0 2	AND	0 6 5	R레지스터와 "065"의 논리곱을 취하고 그 결과를 R레지스터에 저장	000 064 065
0 3	OUT	0 1 3	R레지스터의 내용을 출력단자 "013"에 출력하지 않는다.	000 064 065

의 내용이 메모리부에서 호출되어 R레지스터와의 논리곱이 취해지고 그 연산결과를 다시 R레지스터에 저장한다.

"000"과 "064"의 2개가 모두 ON이면 R레지스터에는 "ON"이라는 연산결과가 저장되며 어느 한쪽이 OFF이면 AND 명령에는 "OF"라는 연산결과가 저장된다.

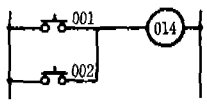
어드레스 "02"의 AND와 065명령이 출력되면 CPU는 다시 내부 보조릴레이 "065"의 내용을 메모리부에서 호출하여 처음의 AND 명령(어드레스 "01")뒤의 연산결과가 저장되어 있는 R레지스터와의 곱을 취하고 그 연산결과를 R레지스터에 저장한다.

그리고 "000", "064", "065"의 세 가지 내용이 모두 ON이면 R레지스터의 내용은 "ON"으로 되며 AND 명령일 때만 S레지스터를 사용하지 않는다.

최후의 OUT 명령은 R레지스터의 내용을 출력단자 "013"에 출력하라는 의미이며 OUT 명령이 출력되어도 R레지스터의 내용은 바뀌지 않는다.

(3) 오어(OR) 명령

데이터로 지정된 릴레이 번호의 내용(ON 또는 OFF)과 R레지스터 내용의 논리합(OR)을 취하고 그 결과를 새로이 R레지스터에 저장하는 명령이다. 그림30은 2개의 접점이 병렬회로이기 때문에 OR 명령으로 되며 그림31은 병렬회로 안에

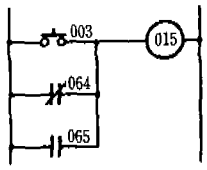


(a) 시퀀스

어드레스	명 령	데이터
0 0	LD	0 0 1
0 1	OR	0 0 2
0 2	OUT	0 1 4

(b) 코딩

<그림30> OR 명령 (1)



(a) 시퀀스

어드레스	명 령	데이터
0 0	LD	0 0 3
0 1	OR NOT	0 6 4
0 2	OR	0 6 5
0 3	OUT	0 1 5

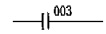
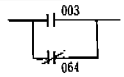
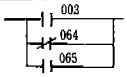
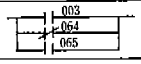
(b) 코딩

<그림31> OR 명령 (2)

b접점 1개를 포함하고 있어 OR NOT이라고 프로그램한다.

OR 명령일 때는 R레지스터의 작용을 그림31과 표5를 참고로 하여 설명한다. 먼저 어드레스 "00"의 LD 명령으로 입력단자 "003"의 내용(ON 또는 OFF)이 레지스터에 저장되는데 이때까지 저장되어 있던 R레지스터의 내용은 없어진다. 또한 어드레스 "01"의 OR · NOT 명령에서는 내부 보조릴레이 "064"의 내용이 메모리부에서 호출되고 우선 그 내용을 부정(NOT)한 다음에 레지스터의 내용(003의 내용)과의 논리합(OR)이 취해져 새로운 결과가 R레지스터에 저장된다.

<표 5 > OR 명령

어드레스	명 령	데이터	의 미	R 레지스터
0 0	LD	0 0 3	입력단자 "003"의 내용을 R레지스터에 저장	
0 1	OR NOT	0 6 4	R레지스터의 내용과 "064"의 부정내용에 대한 논리합을 취하고 그 결과를 R레지스터에 저장	
0 2	OR	0 6 5	R레지스터의 내용과 "065"의 내용에 대한 논리합을 취하고 그 결과를 R레지스터에 저장	
0 3	OUT	0 1 5	R레지스터의 내용을 출력단자 "015"에 출력하지 않는다.	

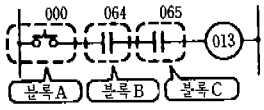
한편 "003"이 ON이든가 또는 "064"가 OFF(반전하여 ON)이든가 하면 R레지스터에는 "ON"이 저장된다. 그리고 어드레스 "02"의 OR 명령으로 내부 보조릴레이 "065"의 내용이 메모리부에서 호출되어 새로운 R레지스터의 내용과 논리합(OR)을 취하고 연산결과를 R레지스터에 저장한다.

(4) 앤드 로드(AND LD) 명령

그림27가 그림28의 회로는 별도의 코딩을 할 수 있으며 그림27을 예로 들면 그림32와 같이 3개의 블록으로 나눌 수 있다.

블록의 시작은 LD 명령으로 시작하며 블록의 끝에는 OUT 명령이 없다. 블록 사이의 논리곱은 AND · LD 명령을 사용하고 블록 사이를 직렬접속한다.

한마디로 AND · LD 명령은 블록과 블록을 직



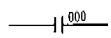
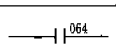
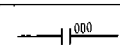
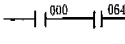
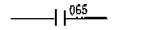
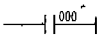
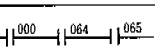
(a) 시퀀스

어드레스	명 령	데이터
0 0	LD	0 0 0
0 1	OR	0 6 4
0 2	AND · LD	—
0 3	LD	0 6 5
0 4	AND · LD	—
0 5	OUT	0 1 3

(b) 코딩

<그림32> AND · LD 명령 (I)

<표 6 > AND · LD 명령 (I)

어드레스	명 령	데이터	의 미	R 레지스터	S 레지스터
0 0	LD	0 0 0	입력단자 "000"의 내용을 R레지스터에 저장		
0 1	LD	0 6 4	처음에 R레지스터의 내용을 S레지스터에 전송하고 새로이 "064"를 R레지스터에 저장		
0 2	AND·LD	—	R레지스터의 내용과 S레지스터에 있는 내용의 AND를 취하고 R레지스터에 저장		
0 3	LD	0 6 5	처음에 R레지스터의 내용을 S레지스터에 전송하고 새로이 "065"를 R레지스터에 저장		
0 4	AND·LD	—	R레지스터의 내용과 S레지스터에 있는 내용의 AND를 취하고 R레지스터에 저장		
0 5	OUT	0 1 3	R레지스터의 내용을 출력단자 "013"에 출력		

렬로 종합할 때 사용하는 명령이다. 다음에 그림 32와 표 6으로서 S레지스터의 작용을 설명한다.

좌측 제어모선부터 LD 명령으로 시작하여 우측 제어모선의 OUT 명령으로 끝나는 것이 1회로에 대한 프로그래밍의 원칙이다.

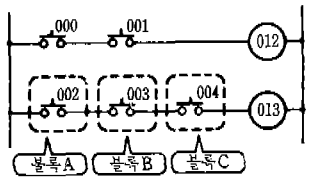
그러나 블록 사이의 논리곱(직렬접속)에서는 처음의 LD 명령 다음에 OUT 명령이 출력되기 전에 다시 LD 명령이 출력되어 이것이 블록 사이를 연산하는 특징이 된다.

어드레스 "00"의 LD 명령으로 블록A의 연산이 시작되며 다음에는 어드레스 "01"의 LD 명령으로 블록B의 연산이 시작된다.

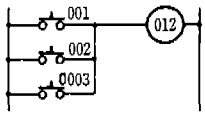
두번째의 LD 명령으로 블록A의 연산 결과가 S레지스터에 전송되며 또한 어드레스 "02"의 AND·LD 명령으로 S레지스터에 전송되어 있던 블록A의 연산결과가 재차 호출되고 블록B와의 연산이 이루어진다.

어드레스 "03"에서는 다시 LD 명령이 출력되어 있으므로 R레지스터의 내용이 다시 S레지스터로 전송되어 블록C의 연산이 시작되며 어드레스 "04"의 AND·LD 명령으로 다시 S레지스터의 내용이 호출된다. 그 후 R레지스터 사이에서 연산이 이루어지게 된다.

그림33과 표 7은 블록 사이의 논리곱과 단순한 접점 사이의 논리곱에 대한 차이를 나타낸다.



<그림33> AND·LD 명령 (2)



(a) 시퀀스

어드레스	명 령	데이터
0 0	LD	0 0 1
0 1	OR	0 0 2
0 2	OR	0 0 3
0 3	OUT	0 1 2

(b) 코딩

<그림34> OR 명령

<표 7> AND·LD 명령 (2)

어드레스	명 령	데이터	R레지스터	S레지스터
0 0	LD	0 0 0	000	–
0 1	AND	0 0 1	000 001	–
0 2	OUT	0 1 2	000 001	–
0 3	LD	0 0 2	002	–
0 4	LD	0 0 3	003	002
0 5	AND·LD	–	002 003	–
0 6	LD	0 0 4	004	002 003
0 7	AND·LD	–	002 003 004	–
0 8	OUT	0 1 3	002 003 004	–

블록의 시작이 LD 명령이며 끝나는 것이 OUT 명령이기 때문에 R레지스터만을 사용한다.

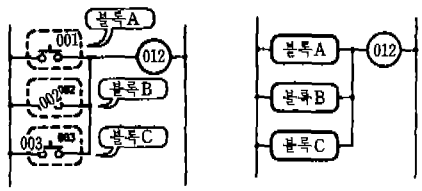
OUT 명령 다음의 LD 명령은 다만 R레지스터의 내용이 바뀐다.

A블록을 S레지스터에 전송한 후 B블록을 연산동작

블록A와 블록B의 AND

블록A-B의 결과를 S레지스터에 전송후 블록C를 연산동작

블록A-B-C의 AND



(a) 시퀀스 (b) 블록도

어드레스	명 령	데이터
0 0	LD	0 0 1
0 1	LD	0 0 2
0 2	OR · LD	–
0 3	LD	0 0 3
0 4	OR · LD	–
0 5	OUT	0 1 2

(c) 코딩

<그림35> OR · LD 명령

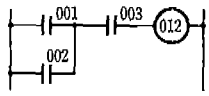
한편 OR LD 명령(그림34)은 그림35와 같이 3개의 블록으로 나눌 수 있으며 블록 사이의 논리합(OR · LD)은 블록과 블록을 병렬로 접속한다.

그림35에 대한 R레지스터와 S레지스터의 작용은 표8과 같이 설명된다.

(5) 직 · 병렬회로와 블록화

그림36에 직 · 병렬회로의 예를 나타내며 AND 명령이나 OR 명령은 그때까지의 연산결과에 대한 AND이며 OR이라는 것이다.

어드레스 "00"의 LD 명령으로 연산이 시작되고 다음의 OR 명령으로 "001"의 OR를 취한 결과를 R레지스터에 저장한다. 다시 어드레스 "02"의 AND 명령으로 R레지스터와 "003"의 AND와의 연산이 이루어지는 순서가 된다.



(a) 시퀀스

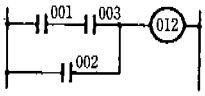
어드레스	명 령	데이터
0 0	LD	0 0 1
0 1	OR	0 0 2
0 2	AND	0 0 3
0 3	OUT	0 1 2

(b) 코딩

<그림36> 직병렬회로 (I)

<표 8 > OR · LD 명령

어드레스	명 령	데이터	의 미	R 레지스터	S 레지스터
0 0	LD	0 0 1	입력단자 "001"의 내용을 R레지스터에 저장. 블록A의 연산 개시	001	–
0 1	LD	0 0 2	R레지스터의 내용을 S레지스터에 전송후 R레지스터에 "002"를 저장. 블록B의 연산 개시	002	001
0 2	OR·LD	–	R레지스터의 내용과 S레지스터에 있는 내용의 OR을 취하고 R레지스터에 저장. 블록A와 블록B의 블록간 논리합(OR)	001 002	–
0 3	LD	0 0 3	R레지스터의 내용을 S레지스터에 전송하고 R레지스터에 "003"을 저장. 블록C의 연산 개시	003	001 002
0 4	OR·LD	–	R레지스터의 내용과 S레지스터에 있는 내용의 OR을 취하고 R레지스터에 저장. 블록A, 블록B, 블록C의 논리합	001 002 003	–
0 5	OUT	0 1 2	R레지스터의 내용을 출력단자 "012"에 출력	001 002 003	–

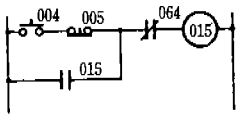


(a) 시퀀스

어드레스	명 령	데이터
0 0	LD	0 0 1
0 1	AND	0 0 3
0 2	OR	0 0 2
0 3	OUT	0 1 2

(b) 코딩

<그림37> 직병렬회로 (2)

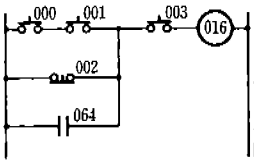


(a) 시퀀스

어드레스	명 령	데이터
0 0	LD	0 0 4
0 1	AND NOT	0 0 5
0 2	OR	0 1 5
0 3	AND NOT	0 6 4
0 4	OUT	0 1 5

(b) 코딩

<그림38> 직병렬회로 (3)



(a) 시퀀스

어드레스	명 령	데이터
0 0	LD	0 0 0
0 1	AND	0 0 1
0 2	OR · NOT	0 0 2
0 3	OR	0 6 4
0 4	AND	0 0 3
0 5	OUT	0 1 6

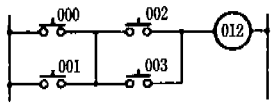
(b) 코딩

<그림39> 직병렬회로 (4)

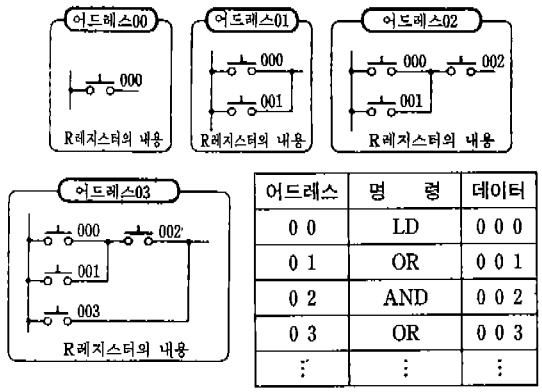
그러나 그림40의 회로를 프로그램할 때 그림41처럼 하면 잘못이다. 연산결과(R레지스터의 내용)에 다음의 AND나 OR의 명령으로 연산처리가 되지 않을 때는 이미 설명한 대로 블록분배를 하여야 한다.

예를 들면 그림42와 같이 블록분배를 하여 코딩하여야 한다. R레지스터와 S레지스터의 작용을 표 9에 나타내었다.

한편 그림43의 회로를 그림44와 같이 프로그램



<그림40> 직병렬회로 (5)

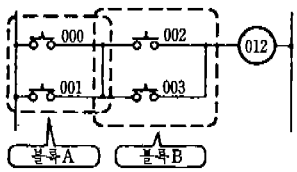


어드레스	명 령	데이터
0 0	LD	0 0 0
0 1	OR	0 0 1
0 2	AND	0 0 2
0 3	OR	0 0 3
⋮	⋮	⋮

<그림41> 코딩에러

그런데 그림37의 (a)와 같이 LD "001" 다음에 AND "003"이라고 코딩하면 "001"과 "003"이 AND한 연산결과가 R레지스터에 저장된다.

다음의 OR "002"에서 AND의 결과와 "002"의 OR이라는 연산이 이루어지는데 그림38과, 그림39는 코딩 예를 나타낸다. 두 회로는 모두 R레지스터의 결과와 다음 명령의 연산이라는 형태로 논리연산이 진행되므로 AND와 OR만으로 코딩이 된다.



(a) 시퀀스

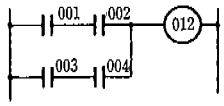
어드레스	명 령	데이터
0 0	LD	0 0 0
0 1	OR	0 0 1
0 2	LD	0 0 2
0 3	OR	0 0 3
0 4	AND · LD	–
0 5	OUT	0 1 2

(b) 코딩

<그림42> 블록화

<표 9 > AND · LD 명령

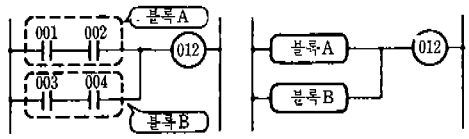
어드레스	명 령	데이터	의 미	R 레지스터	S 레지스터
0 0	LD	0 0 0	블록A의 연산 시작, R레지스터에 "000"의 내용을 저장	000	–
0 1	OR	0 0 1	블록A 연산중, R레지스터의 내용과 "001"의 OR을 취하고 R레지스터에 저장	000 001	–
0 2	LD	0 0 2	R레지스터의 내용을 S레지스터에 전송한 후 "002"의 내용을 R레지스터에 저장. 블록B의 연산 개시	002	000 001
0 3	OR	0 0 3	블록B의 연산중, R레지스터의 내용과 "003"의 OR을 취하고 R레지스터에 저장	002 003	000 001
0 4	AND·LD	–	R레지스터와 S레지스터에 있는 내용의 AND를 취하고 R레지스터에 저장. 블록A와 블록B 사이의 AND	000 002 001 003	–
0 5	OUT	0 1 2	R레지스터의 내용을 출력	000 002 001 003	–



<그림43> 직병렬회로 (6)

어드레스	명 령	데이터
0 0	LD	0 0 1
0 1	AND	0 0 2
0 2	OR	0 0 3
0 3	AND	0 0 4
⋮	⋮	⋮

<그림44> 코딩에러



(a) 시퀀스 (b) 블록화

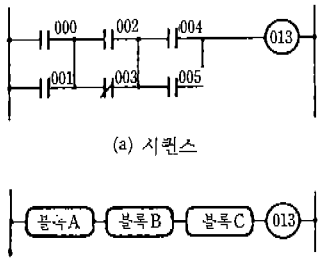
<그림45> 직병렬회로

하여도 에러가 되므로 그림45와 같이 블록을 분배할 필요가 있으며 표10에 각 레지스터의 작용을 나타낸다.

블록 사이의 논리곱(AND · LD)이나 논리합(OR · LD)을 취할 때에 S레지스터를 사용하는데 S레지스터의 깊이에 제한이 없으므로 그림46의 예로서 설명한다.

코딩은 표11과 같이 되고 어드레스 "02"에서 블록A가 S레지스터로 전송되고 어드레스 "04"에서 블록B가 다시 S레지스터에 전송되어 있으므로 S레지스터의 상태는 그림47과 같이 된다.

OUT 명령이 출력되기 전에 차례로 LD 명령



(a) 시퀀스

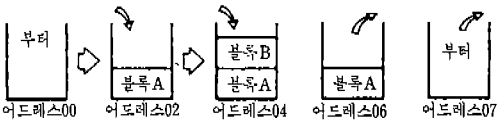
(b) 블록화

<그림46> 직병렬회로 (7)

<표10> OR·LD 명령

어드레스	명 령	데이터	의 미	R레지스터	S레지스터
00	LD	001	블록A의 연산 시작, R레지스터에 "001"의 내용을 저장	001	–
01	AND	002	블록A 연산중, R레지스터의 내용과 "002"의 AND를 취하고 R레지스터에 저장	001 002	–
02	LD	003	R레지스터의 내용을 S레지스터에 전송한 후 "003"의 내용을 R레지스터에 저장. 블록B의 연산 개시	003	001 002
03	AND	003	블록B의 연산중, R레지스터의 내용과 "004"의 AND를 취하고 R레지스터에 저장	003 004	001 002
04	OR·LD	–	R레지스터와 S레지스터에 있는 내용의 OR을 취하고 R레지스터에 저장. 블록A와 블록B 사이의 OR	001 002 / 003 004	–
05	OUT	012	R레지스터의 내용을 출력	001 002 / 003 004	–

이 출력되면 이때까지의 연산결과는 차례로 S레지스터에 전송된다. 그런데 깊이에는 제한이 없고 S레지스터에 다시 호출될 때는 전송된 순서의 반대가 된다.



<그림47> S레지스터

<표11> 레지스터 사이의 전송

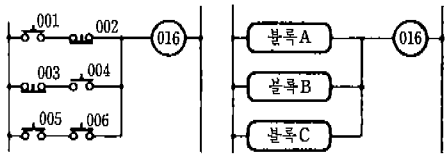
어드레스	명 령	데이터
00	LD	000
01	OR	001
02	LD	002
03	OR NOT	003
04	LD	004
05	OR	005
06	AND·LD	–
07	AND·LD	–
08	OUT	013

블록A
블록A를 S레지스터에 전송
블록B
블록B를 S레지스터에 전송
블록C
블록C와 블록B의 AND
블록C·블록B와 블록A의 AND

그림48의 회로에서 블록 사이의 논리값이나 논리합을 취하는 두 가지 방법을 설명한다.

첫번째 방법은 블록A의 연산결과를 S레지스터에 전송해 두고 블록B와의 연산을 하며 이 결과를 S레지스터에 전송한 후 블록C의 연산을 R레지스터로 하고 S레지스터와 OR을 취하는 방법이다.

두번째 방법은 우선 블록A의 결과를 S레지스터에 전송하고 블록B도 이어서 S레지스터에 전송한다. 그리고 블록C를 R레지스터로 연산한 다음 블록B와의 논리합을 취하여 그 결과를 레지스터로 전송한 블록A와의 논리합을 취하는 방법이다(표12 참조). 끝으로 그림49 이하의 그림들은 직·병렬회로와 코딩한 예를 나타낸다.



(a) 시퀀스 (b) 블록화
<그림48> 직병렬회로 (8)

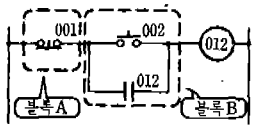
<표12> 블록간의 논리합

(a) 코딩 (1)

어드레스	명 령	데이터	
0 0	LD	0 0 1	블록A
0 1	AND NOT	0 0 2	블록A를 S레지스터로
0 2	LD NOT	0 0 3	블록B
0 3	AND	0 0 4	블록B와 블록A의 OR
0 4	OR·LD	–	블록A와 블록B의 OR를 S레지스터로
0 5	LD	0 0 5	블록C
0 6	AND	0 0 6	
0 7	OR·LD	–	블록A와 블록B의 OR과 블록C의 OR
0 8	OUT	0 1 6	

(b) 코딩 (2)

어드레스	명 령	데이터	
0 0	LD	0 0 1	블록A
0 1	AND NOT	0 0 2	블록A를 S레지스터로
0 2	LD NOT	0 0 3	블록B
0 3	AND	0 0 4	블록B를 S레지스터로
0 4	LD	0 0 5	블록C
0 5	AND	0 0 6	블록C와 블록B의 OR
0 6	OR·LD	–	
0 7	OR·LD	–	블록C와 블록B의 OR과 블록A의 OR
0 8	OUT	0 1 6	

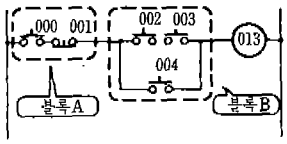


(a) 시퀀스

어드레스	명 령	데 이 터	비 고
0 0	LD NOT	0 0 1	블 록 A
0 1	LD	0 0 2	블 록 B
0 2	OR	0 1 2	
0 3	AND LD	–	A & B
0 4	OUT	0 1 2	–

(b) 코딩

<그림49> 직병렬회로 (9)

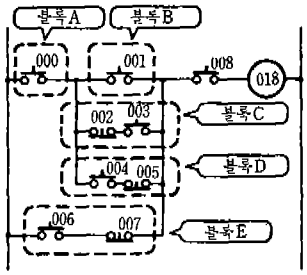


(a) 시퀀스

어드레스	명 령	데 이 터	비 고
0 0	LD	0 0 0	블 록 A
0 1	AND NOT	0 0 1	
0 2	LD	0 0 2	블 록 B
0 3	AND	0 0 3	
0 4	OR	0 0 4	
0 5	AND LD	–	A & B
0 6	OUT	0 1 3	–

(b) 코딩

<그림50> 직병렬회로 (10)

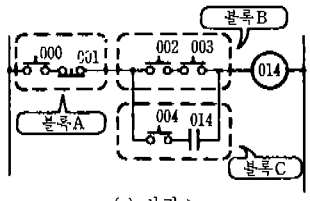


(a) 시퀀스

어드레스	명 령	데 이 터	비 고
0 0	LD	0 0 0	블 록 A
0 1	LD	0 0 1	블 록 B
0 2	LD NOT	0 0 2	블 록 C
0 3	AND	0 0 3	
0 4	LD	0 0 4	블 록 D
0 5	AND NOT	0 0 5	
0 6	OR LD	–	C+D
0 7	OR LD	–	B+C+D
0 8	AND LD	–	(B+C+D)·A
0 9	LD	0 0 6	블 록 E
1 0	AND NOT	0 0 7	
1 1	OR LD	–	(B+C+D)·A+E
1 2	AND	0 0 8	–
1 3	OUT	0 1 8	–

(b) 코딩

<그림55> 직병렬회로 (15)

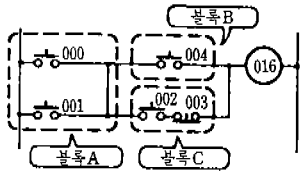


(a) 시퀀스

어드레스	명 령	데 이 터	비 고
0 0	LD	0 0 0	블 록 A
0 1	AND NOT	0 0 1	
0 2	LD	0 0 2	블 록 B
0 3	AND	0 0 3	
0 4	LD	0 0 4	블 록 C
0 5	AND	0 1 4	
0 6	OR LD	–	B+C
0 7	AND LD	–	(B+C)A
0 8	OUT	0 1 4	–

(b) 코딩

<그림51> 직병렬회로 (1)

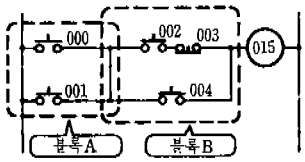


(a) 시퀀스

어드레스	명 령	데 이 터	비 고
0 0	LD	0 0 0	블 록 A
0 1	OR	0 0 1	
0 2	LD	0 0 4	블 록 B
0 3	LD	0 0 2	블 록 C
0 4	AND NOT	0 0 3	
0 5	OR LD	–	B+C
0 6	AND LD	–	(B+C)·A
0 7	OUT	0 1 6	–

(b) 코딩

<그림53> 직병렬회로 (3)

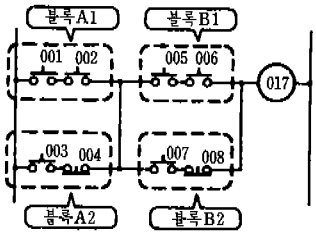


(a) 시퀀스

어드레스	명 령	데 이 터	비 고
0 0	LD	0 0 0	블 록 A
0 1	OR	0 0 1	
0 2	LD	0 0 2	블 록 B
0 3	AND NOT	0 0 3	
0 4	OR	0 0 4	
0 5	AND LD	–	A & B
0 6	OUT	0 1 5	–

(b) 코딩

<그림52> 직병렬회로 (2)

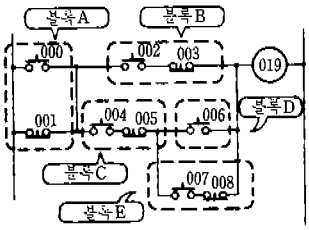


(a) 시퀀스

어드레스	명 령	데 이 터	비 고
0 0	LD	0 0 1	블 록 A1
0 1	AND	0 0 2	
0 2	LD	0 0 3	블 록 A2
0 3	AND NOT	0 0 4	
0 4	OR LD	–	A1+A2
0 5	LD	0 0 5	블 록 B1
0 6	AND	0 0 6	
0 7	LD	0 0 7	블 록 B2
0 8	AND NOT	0 0 8	
0 9	OR LD	–	B1+B2
1 0	AND LD	–	(A1+A2)·(B1+B2)
1 1	OUT	0 1 7	–

(b) 코딩

<그림54> 직병렬회로 (4)

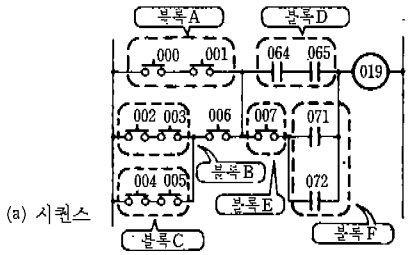


(a) 시퀀스

어드레스	명 령	데 이 터	비 고
0 0	LD	0 0 0	블 록 A
0 1	OR NOT	0 0 1	
0 2	LD	0 0 2	블 록 B
0 3	AND NOT	0 0 3	
0 4	LD	0 0 4	블 록 C
0 5	AND NOT	0 0 5	
0 6	LD	0 0 6	블 록 D
0 7	LD	0 0 7	블 록 E
0 8	AND NOT	0 0 8	
0 9	OR LD	—	D+E
1 0	AND LD	—	(D+E)·C
1 1	OR LD	—	(D+E)·C+B
1 2	AND LD	—	((D+E)·C+B)·A
1 3	OUT	0 1 9	—

(b) 코딩

<그림56> 직병렬회로 (16)



(a) 시퀀스

어드레스	명 령	데 이 터	비 고
0 0	LD	0 0 0	블 록 A
0 1	AND	0 0 1	
0 2	LD	0 0 2	블 록 B
0 3	AND	0 0 3	
0 4	LD	0 0 4	블 록 C
0 5	AND	0 0 5	
0 6	OR LD	—	(B+C)·006
0 7	AND	0 0 6	—
0 8	OR LD	—	(B+C)·006+A
0 9	LD	0 6 4	블 록 D
1 0	AND	0 6 5	
1 1	LD	0 0 7	블 록 E
1 2	LD	0 7 1	블 록 F
1 3	OR	0 7 2	
1 4	AND LD	—	E·F
1 5	OR LD	—	E·F+D
1 6	AND LD	—	(E·F+D)((B·C)·006+A)
1 7	OUT	0 1 9	—

(b) 코딩

<그림57> 직병렬회로 (17)

☞ 다음 호에 계속

겨울철 실내적정 난방온도 기준은
18~20°C
☞ 난방온도 1°C를 낮추면
약 1,200억원의 에너지가 절약됩니다.