현장 기술자를 위한

소형 PC에 의한 시퀀스 제어

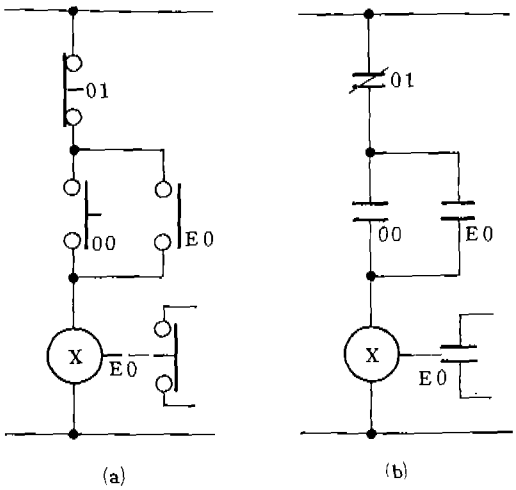
현·장 기·술 9

출판홍보과

제8장 래더 다이어그램으로부터의 프로그래밍

래더 다이어그램이란 릴레이 시퀀스 회로에 사용하는 특수한 심볼을 사용하여 시퀀스 회로를 표현한 그림으로서, 기본적으로는 이미 기술한 유접점 회로와 다르지 않은 것이다. 다만, 접점기호를 사다리(영어로 lader : 래더)가 이어진 것 같이 그리기 때문에 붙여진 이름이다.

이 래더 다이어그램으로부터의 프로그램에 관해서는 제5장까지 기술한 로직 다이어그램으로부터의 프로그램 작성의 기본적 사고방식으로 충분히 설명할 수 있는 것이다.



〈그림 8.1〉 래더 다이어그램

그러나 전자 릴레이 시퀀스에 익숙해진 사람들은 이 전자 릴레이 시퀀스 회로와 동일한 표현형식을 취하는 래더 다이어그램으로부터 곧바로 PC의 프로그램을 만들려고 하는 요구가 있다. 그래서 이에 대응하기 위해 래더 다이어그램에서 직접 코딩하는 방법에 대해서 기술하기로 한다.

확실히 이 래더 다이어그램에서 직접 프로그램을 만드는 이 방식은 로직 심볼도로 변환하는 번거로움이 없다든가, 종래의 릴레이 시퀀스 제어회로에 익숙해져 있는 사람은 취급하기 쉽다는 등의 몇가지 장점은 있다.

그러나 제3장에서 기술한 바와 같이 프로그램을 만들어나가는 데 있어서 몇가지 제약조건이나 프로그램 언어 사용방법의 불명료성은 남는다. 그것은 이론의 흐름이나 구조를 표현하는 것에는 이 래더 다이어그램이 반드시 적합한 것은 아니기 때문이다. 이것을 염두에 두고 다음 설명을 들어야 할 필요가 있다.

다음에 래더 다이어그램의 표현 형식은 다음과 같이 한다. 그림 8.1(a)는 전자 릴레이 접점 기호를 사용하여 그린 회로이다. 이것을 래더 다이어

〈표 8.1〉 프로그램 리스트

스 텝 No.	명 령		비 고
	OP	ADD	
00	LD	00	
01	OR	E0	
02	AND	01	
03	OUT	E0	

그램으로 표현한 것이 그림 (b)이다. 접점기호가 다를 뿐이고 양 회로의 표현상의 본질적인 상이점은 없다. 그 때문에 본래 래더 다이어그램이란 그림 8.1(b)를 말하는 것이지만 이후 본고에서는 전자 릴레이 접점기호를 사용하여 그린 그림 8.1(a) 타입의 그림을 원칙으로 사용하기로 한다.

8.1 프로그래밍의 특징

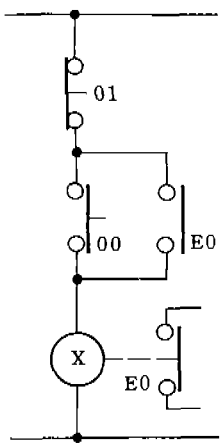
(1) 출력 OUT의 데이터를 접점 데이터로서 사용

그림 8.2에서 출력은 E0번지에 나가도록 번지가 할당되고 있지만 이 E0번지는 유지용 접점의 번지에도 사용되고 있다. 이 예와 같이 출력 X와 연동하는 접점을 유지용 접점으로 사용하는 것은 전자 릴레이 회로의 배선에서는 흔히 행하여지는 것으로서 문제가 되지 않는 것이다. 그러나 PC에 있어 래더 다이어그램에서 프로그램을 만드는 경우에는 주의하지 않으면 안되는 것이다.

표 8.1의 프로그램 리스트에서

01 스텝 OR E0

는 출력 OUT의 데이터인 E0번지의 내용을 사용



〈그림 8.2〉
출력 OUT의 데이터를 접점 데이터로서 사용하는 예

〈표 8.2〉 명령 실행에 의한 메모리의 상태

명령실행에 의해 메모리에 명령의 연산결과가 격납되는 명령군	명령실행에 의해 메모리에 명령의 데이터가 격납되는 명령군
LD, LD · NOT OR, OR · NOT AND, AND · NOT INV, INV · NOT	STO, STO · NOT TIM, TIM · NOT OUT, OUT · NOT

〈표 8.3〉 프로그램 리스트

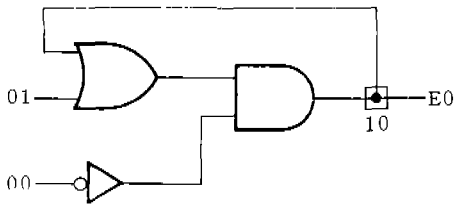
스 텝 No.	명 령		비 고
	OP	ADD	
00	LD	00	
01	OR	10	
02	AND	01	
03	STO	10	
04	OUT	E0	

하고 있다. 즉, 출력 데이터의 분기가 행하여지고 있는 것이므로 이 E0번지의 데이터는 본래 어딘가에서 STO 명령을 사용하여 메모리에 격납해 두지 않으면 안되는 것이다. 그런데도 표 6.1의 프로그램으로 바르게 동작한다.

그 이유는 OUT 명령은 STO 명령의 기능도 함께 갖도록 만들어져 있어 이 OUT 명령을 실행함으로써 동시에 데이터가 메모리에 격납되게 되어 있기 때문이다. 이에 의해 데이터 분기가 있어도 STO 명령을 사용할 필요가 없어진다.

명령 실행에 의한 메모리의 상태는 명령의 연산결과가 격납되는 것과 명령 데이터가 격납되는 것의 두 가지로 분류된다. 참고로 표 8.2에 여기서 사용하는 PC에 대해서 그것들을 분류한 것을 나타낸다.

로직 다이어그램으로부터의 프로그램 작성에서는 데이터의 분기 개소에는 STO 명령을 사용하여 데이터를 일단 메모리에 격납해 둘 필요가 있었



〈그림 8.3〉 STO의 사용예

〈표 8.4〉 프로그램 리스트

스 텝 No.	명 령 OP	명 령 ADD	비 고
00	LD	00	
01	AND · NOT	E1	
02	OUT	E0	
03	LD	01	
04	AND · NOT	E0	
05	OUT	E1	

다. 이 방식으로 앞의 회로와 동일 동작을 하는 프로그램을 만들면 표 8.3의 프로그램이 된다. 그림 8.3의 로직 다이어그램을 참고로 하여 이 표 8.3의 프로그램을 검토하면 알 수 있듯이 래더 다이어그램으로부터의 프로그램은 OUT 명령이 STO 기능을 함께 가지고 있기 때문에 STO가 불필요해진 모양을 알 수 있다.

(2) AND · NOT 등의 복합명령 사용

그림 8.4의 회로에서 접점 X_2, X_1이 b접점에서 각각 접점 A, B에 연결되어 있다. 이것을 윗 쪽의 접점측에서 프로그래하는 경우에 대해서 생각해 본다. 표 8.4의 프로그램 리스트를 보기로 하자. 우선

01 스텝 LD 00

에서 접점 A(00번지)의 내용을 판독해 넣는다.

다음에는 이 결과와 X_2(E1번지)의 부정과의 AND를 취하는데, 이것을

02 스텝 AND · NOT E1

〈표 8.5〉 잘못된 프로그램

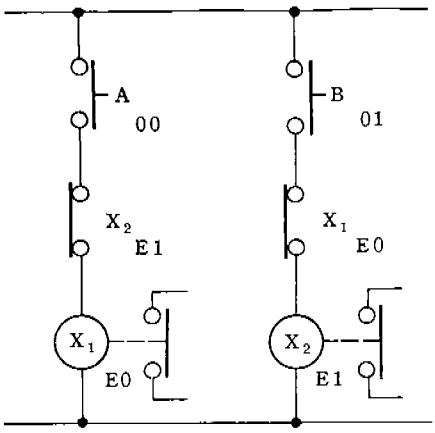
스 텝 No.	명 령 OP	명 령 ADD	비 고
00	LD	00	
01	AND	E1	
02	INV	00	
03	OUT	E0	
04	LD	01	
05	AND	E0	
06	INV	00	
07	OUT	E1	

으로 한다. 이 AND · NOT의 기능은 E1번지의 부정(NOT)과 위의 결과와의 AND를 취하라는 것이다. 이 2개 기능을 하나의 명령에 갖게 하여 AND · NOT라고 기록하므로 이것을 복합명령이라고 부르기로 한다. 이 복합명령을 사용하지 않고 표 8.5와 같이 프로그램하면 잘못이 된다.

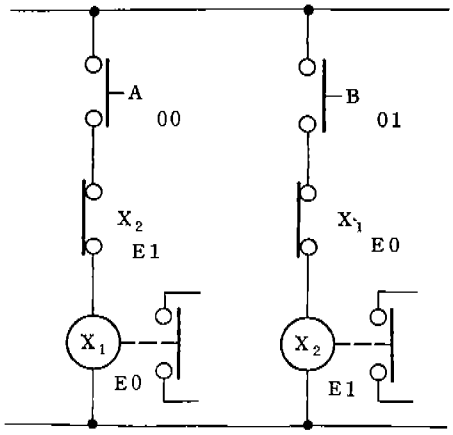
이 종류의 (연산명령) · NOT의 형을 한 복합명령은 여기서 기술한 AND · NOT 외에 모든 연산명령에 대해서 있다. 사용방법은 이 AND · NOT와 동일하므로 대표적으로 이 AND · NOT의 예를 나타냈다.

(3) LD · NOT, OUT · NOT의 사용

복합명령에는 (연산명령) · NOT의 형만이 아니고 (판독, 출력명령) · NOT의 형도 있다. 그림 8.5는 앞의 그림 8.4와 동일하지만 이 회로를 X_2(E1번지) 접점측보다 먼저 코딩하는 경우에 대해서



〈그림 8.4〉 AND · NOT의 사용예



〈그림 8.5〉 LD · NOT의 사용예

〈표 8.6〉 프로그램 리스트

스 텝 No.	명 령 OP	명 령 ADD	비 고
00	LD · NOT	E1	
01	AND	00	
02	OUT	E0	
03	LD · NOT	E0	
04	AND	01	
05	OUT	E1	

생각한다. 표 8.6의 프로그램 리스트가 이 경우의 것이다. 우선

01 스텝 LD · NOT E1

으로 X_2의 부정을 판독한다. 그리고 다음에 이 결과와 A(00번지)와의 AND를 취하므로

02 스텝 AND 00

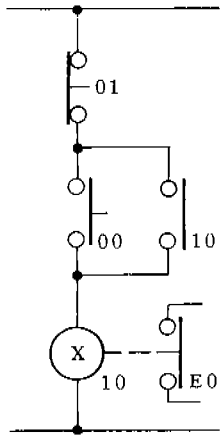
로 계속한다. LD · NOT를 사용하지 않고 표 8.7과 같이 하여도 동일 내용이 된다.

OUT · NOT의 사용방법은 LD · NOT의 사용방법과 동일하다.

(4) STO 사용방법

이 STO 명령은 로직 다이어그램으로부터의 프로그래밍에서는 전자계산기의 메모리 사용방법과 동일하게 사용할 수 있는 것을 기술하였다. 래더 다이어그램으로부터의 프로그램도 본질적인 것은 다름이 없다. 다만 래더 다이어그램의 특징을 알고 사용할 필요가 있다.

래더 다이어그램은 그림 8.6과 같이 X코일에는



〈그림 8.6〉 데이터 분기의 예

〈표 8.7〉 표 8.6과 동일 내용 프로그램

스 텝 No.	명 령 OP	명 령 ADD	비 고
00	LD	E1	
01	INV	00	
02	AND	00	
03	OUT	E0	
04	LD	E0	
05	INV	00	
06	AND	01	
07	OUT	E1	

〈표 8.8〉 프로그램 리스트

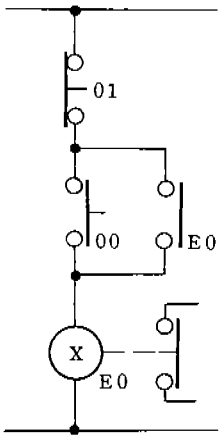
스 텝 No.	명 령 OP	명 령 ADD	비 고
00	LD	00	
01	OR	10	
02	ANDT	01	
03	STO	10	
04	OUT	E0	

접점 01, 00, 10이 어떻게 접속되어 있는가를 나타내고 있을 뿐이고 X코일(10번지)과 접점 10번지가 연동하고 있는 것을 기호나 할당 번지에서 판단하지 않으면 안되는 것이다.

즉, 로직 다이어그램과 같이 신호(데이터)가 어떠한 논리 기호를 통해서 흐르고 어디로 나가는가 하는 데이터가 흐르는 모양이나 논리의 연결이 반드시 확실하지는 않은 것이다.

이 때문에 STO 명령을 왜 이 개소에서 사용하지 않으면 안되는가, 래더 다이어그램 상으로 확실하지 않은 경우가 약간 생긴다. 그래서 앞에서든 AND · NOT 등의 복합명령을 사용하여 STO 명령을 가급적 사용하지 않도록 하거나 또는 STO 명령을 사용하여야 할 곳에 STO 기능를 갖게 한 다른 명령을 주어 그것으로 대처하고 있는 경우가 있다.

그러나 본고에서는 STO 명령을 적극적으로 사용하여 래더 다이어그램에서 프로그램을 작성하는 방식을 기술하기로 한다. 그 이유는 첫째로 STO를 사용함으로써 프로그램의 구조를 명확히 하여 프로그램을 만들어 나가는 순서와 프로그램 구성을 말끔하게 하기 때문에, 두번째로 이 STO의 기본적 사용방식을 마스터하면 STO를 사용하지 않



〈그림 8.7〉 연산결과 대기의 예

는 복합명령에 의한 프로그램 작성법도 이해될 것으로 생각되기 때문이다.

STO 설정 개소의 기본은 ① 데이터의 분기(보조접점) 개소, ② 연산결과 대기 개소의 두가지이다. 이하, 이것에 대해서 설명한다.

① 데이터 분기(보조접점)

그림 8.6에서 릴레이 코일 Ⓧ는 출력접점(E0)을 동작시킴과 동시에 보조접점 10도 동시에 동작시키는 것으로서 앞에서 든 표 8.1의 프로그램 리스트와 같이 릴레이 코일의 데이터(E0번지의 내용)를 출력접점 및 보조접점으로 사용하는 방법으로 프로그램을 만들 수도 있었다.

여기서는 Ⓧ코일의 상태를 즉시 출력하는 것은 아니고 복잡한 회로의 다음 단계의 접점을 동작시키기 위한 보조접점에 사용하는 경우를 상정하고 있다. 이와 같은 때는 Ⓧ코일을 일시 기억(10번지)으로 설정하고 STO를 사용하여 그 데이터(내용)를 일단 메모리에 기억시켜 두고 필요할 때 그 번지를 지정하여 데이터를 취하도록 한다.

구체적으로 표 8.8의 프로그램 리스트를 보기로 하자.

03 스텝 STO 10

으로 101번지에 Ⓧ코일의 상태가 스토어되고 있으므로

01 스텝 OR 10

으로 사용할 수 있고 또 다음 단계의 회로 접점 데이터로서도 사용할 수 있는 것이다. 그러나 이

〈표 8.9〉 프로그램 리스트

스 텝 No.	명 령 OP	명 령 ADD	비 고
00	LD	01	
01	STO	10	
02	LD	00	
03	OR	E0	
04	AND	10	
05	OUT	E0	

〈표 8.10〉 스택 이용의 프로그램

스 텝 No.	명 령 OP	명 령 ADD	비 고
00	LD	01	스택
01	LD	00	
02	OR	E0	
03	AND	FF	주의
04	OUT	E0	

예에서는

04 스텝 OUT E0

와 Ⓧ코일의 상태를 출력하여 프로그램을 완결시키고 출력상태에서 프로그램의 동작을 확인할 수 있게 되어 있다.

② 연산처리 대기

그림 8.7에서 접점 01부터 최초에 프로그램해 나가는 경우를 생각한다. 표 8.9의

00 스텝 LD 01

에서 01번지의 내용을 읽어 넣고 다음의 병렬회로(00와 E0번지의)를 프로그램하려고 하면 지금 읽어 넣은 LD 01의 처리가 곤란해진다(일반적으로는 최초 병렬회로측으로부터 프로그램을 만들어 나가므로 이와 같이 되지는 않는다. 이 예는 연산처리 대기의 간단한 예로서 적당한 것이기 때문에 들었다).

그래서 지금까지의 처리 결과를 다음 병렬회로의 연산처리 대기를 위해 일단 일시기억 10번지에 격납해 둔다. 표 8.9의

01 스텝 STO 10

이 그것이다. 그리고 02, 03 스텝에서 병렬회로의 처리를 하고 04 스텝에서 이 결과와 전에 10번지에 격납해 둔 결과의 연산 처리를 한다.

04 스텝 AND 10

[참고] STO 명령을 사용하지 않는 프로그램 작성방법

STO 명령을 사용하지 않는 프로그램 작성방법의 예를 표 8.10에 들었다. 이 예는 스택 레지스터라고 하는 메모리를 이용한 것으로서 스택(stak)에 연산처리 결과를 순서적으로 넣어 두고 뒤부터 또 순서적으로 인출하여 연산처리를 시키게 한 것이다.

이 예에서는 00 스텝에서 01번지의 내용을 판독하고 또 계속해서 01 스텝에서 00번지를 판독하게 되어 있다. 이와 같이 다음의 판독(LD 명령)이 행하여진 시점에서 LD에서 시작하여 다음의 LD전까지의 연산처리 결과가 스택 레지스터에 격납된다. 표 8.9의 01 스텝 STO 10과 비교해 보자.

그리고 03 스텝에서 01과 02 스텝의 연산처리 결과와 스택 레지스터 내용과의 AND를 취하고 있다.

03 스텝 AND FF

이 AND FF라는 것은 스택 레지스터와의 AND를 취한다는 명령으로, FF가 FF번지라는 것은 아니다. PC에 따라서는 이 명령을 복합명령으로 AND · STK(앤드 · 스택)나 AND · LD(앤드 · 로드) 등으로 하고 있는 경우도 있지만 동일 내용의 명령이다.

이 방법의 특징은 스택 레지스터에 하나의 완결된 블록 내용을 순서적으로 넣기만 하면 되고 메모리 번지를 할당할 필요가 없다는 것이다. 그리고 인출할 때는 후입 선출방식(後入先出方式)이라고 해서 최후에 스택 레지스터에 넣어진 내용부터 순서적으로, 즉 넣었을 때와는 반대 순서로 인출해서 연산시키는 방법이 채택된다.

그러므로 스택 레지스터에 넣은 데이터의 순서를 의식하여 프로그램한 순서를 반대로 틀리지 않도록 하지 않으면 안된다. 그러면 처음부터 일시 기억한 데이터는 그 번지까지 도면상에 표시하고 STO 명령을 사용하여 격납하는 방법(표 8.9의 프로그램 작성방법)이 후에 사용할 때 명확하고 사용하기 쉬운 것이 된다.

본고에서는 이 때문에 적극적으로 STO 명령을 사용하도록 한 것이다. STO의 기본적 사용방법을 알면 스택 레지스터를 이용한 프로그램 작성방법은 자연히 이해되게 될 것으로 생각한다. 다만 스택 레지스터의 구조가 후입 선출방식으로 되어 있는 것은 알아 두어야 한다.

(5) 타이머 취급

동작 지연의 타이머 회로를 그림 8.8에 들었다. 타이머 회로의 취급은 그림의 타이머 코일 개소에 번지 할당(이 경우 30번지)과 동작시간의 설정(이 경우 2초)을 하는 요령으로 프로그램을 만든다. 이것이 표 8.11의 프로그램 리스트의

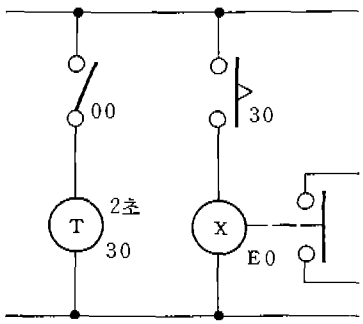
01 스텝 TIM 40

이다. 타이머 번지 및 타이머 시간의 설정에 대해서는 로직 다이어그램의 TIMER의 항을 참조하기 바란다.

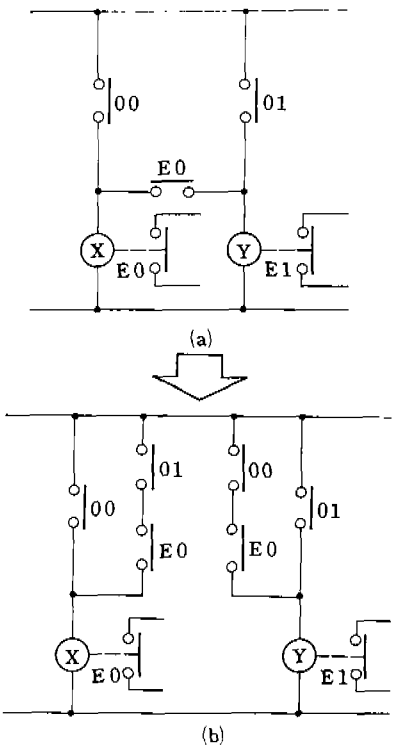
다음에 타이머 접점(한시동작의 a접점)을 타이머 코일에 할당한 번지와 동일 번지(30번지)에서 판독한다. 이것이 다음이다.

02 스텝 LD 30

이 예는 타이머 접점이 하나의 회로 구성이므로



〈그림 8.8〉 타이머 회로



〈그림 8.9〉 프로그램하기 쉬운 회로에의 변환예

LD 30으로 되어 있지만 다른 접점회로 구성으로 되어 있으면 LD명령이 아니고 AND라든가 OR 등의 연산명령이 사용되는 것은 당연하다.

〈표 8.11〉 프로그램 리스트

스 텝 No.	명 령		비 고
	OP	ADD	
00	LD	00	
01	TIM	40	2초
02	LD	30	
03	OUT	E0	

〈표 8.12〉 프로그램 리스트

스 텝 No.	명 령		비 고
	OP	ADD	
00	LD	01	
01	AMD	E0	
02	OR	00	
03	OUT	E0	
04	LD	00	
05	AND	E0	
06	OR	01	
07	OUT	E1	

(6) 프로그램하기 쉬운 회로로 변환

그림 8.9(a)와 같은 회로는 일단 그림 8.9(b)의 ⓧ코일, Ⓨ코일 각각을 중심으로 한 회로로 바꾸어 그리고 나서 프로그램하면 알기 쉽다. 그리고 표 8.12에 이 회로의 프로그램 예를 나타냈다.

☞ 다음호에 계속 ☜

일본어투 생활용어 순화

순화대상용어	순화한 용어	순화대상용어	순화한 용어	순화대상용어	순화한 용어
도라무(drum)	드럼(통)	도라이바(driver)	드라이버	도란스(transformer)	변압기
도랏쿠(truck)	화물차, 트럭	도리(取)	① 독차지, ② 줄떼기	도리우치(鳥打)	납작모자, 캡
도비라(扉)	속표지	도쿠리(徳利)	① 긴목셔츠, ② 조막병	도합(都合)	모두, 합계
돈부리	덮밥	돗쏭(凸版)	볼록판	따불(double)	곱, 갑절
뗑깡(癪癇)	생떼	라이방(Ray Ban)	보안경, 색안경	라지에타(radiator)	방열기
레루(rail)	레일	레자(leather)	인조 가죽	레자(leisure)	여가(활동), 레저
레지(register)	(다방)종업원	렌가(煉瓦)	벽돌	레떼루(letter)	상표, 레테르
로라(roller)	땅다지개, 롤러	로쿠부(六分)	엿푼	로타리(rotary)	둥근거리, 로터리
료마에(兩前)	겹여밈(웃), 겹자락	루베(立方米)	입방미터, m^3	리야카(rear car)	손수레
마도(窓)	① 창, ② 인쇄	마메인(豆印)	잔도장	마메콩(豆--)	콩