6. 래더(LD) 프로그래밍 심화

6.1 타이머(Timer) 제어 (TON, TOF, TONR)

(PLC에서 시간 기반 제어를 구현하는 핵심 기능)

✅ 개요

타이머(Timer)는 PLC 프로그램에서 **시간 지연, 유지, 반복 동작**을 구현할 때 사용하는 대표적인 기능이다.
IEC 61131-3 표준에 따라 다음과 같은 3가지 대표적인 타이머 타입이 존재한다:

- **TON (ON-Delay Timer)**: 지연 후 ON
- **TOF (OFF-Delay Timer)**: 지연 후 OFF
- **TONR (Retentive ON-Delay Timer)**: 누적 지연 ON (전원 꺼져도 누적 시간 유지)

✅ 1. TON (ON-Delay Timer)

개념

입력이 들어오면 일정 시간이 지난 후에 **출력이 ON**되는 타이머

시퀀스 흐름

1. X0 ON → 타이머 동작 시작
2. 시간(T)에 도달하면 Q 출력 ON

📘 래더 예제

```
[ X0 ]──────────( TON T0, PT=5s )──────────
                → Q → [ Y0 ]
```

⏳ ST 코드 예제

```
TON_1(IN := X0, PT := T#5s);
Y0 := TON_1.Q;
```

✅ 2. TOF (OFF-Delay Timer)

개념

입력이 꺼진 후 일정 시간이 지나야 **출력이 OFF**됨

시퀀스 흐름

1. X0 ON → Q 즉시 ON
2. X0 OFF → 시간(T) 후 Q OFF

📘 래더 예제

```
[ X0 ]──────────( TOF T1, PT=3s )──────────
                → Q → [ Y1 ]
```

⏳ ST 코드 예제

```
TOF_1(IN := X0, PT := T#3s);
Y1 := TOF_1.Q;
```

✅ 3. TONR (Retentive ON Timer)

개념

입력이 ON된 시간을 **누적**하여, 총 시간이 도달하면 출력 ON
입력이 꺼져도 **누적 시간은 유지됨** (전원 OFF 예외)

리셋 조건이 별도로 존재 (e.g., R 트리거)

📘 래더 예제

```
[ X0 ]──────────( TONR T2, PT=10s )──────────
[ R  ]──────────( RESET T2 )                  → Q → [ Y2 ]
```

⏳ ST 코드 예제

```
TONR_1(IN := X0, PT := T#10s);
IF R THEN
    TONR_1(IN := FALSE);
    TONR_1.ET := T#0s;
END_IF;

Y2 := TONR_1.Q;
```

✅ 4. 비교 요약

항목	TON (지연 ON)	TOF (지연 OFF)	TONR (누적 지연 ON)
입력 ON	지연 후 출력 ON	즉시 출력 ON	누적 시작
입력 OFF	출력 OFF	지연 후 출력 OFF	누적 유지
리셋 기능	필요 없음	필요 없음	필요함 (`R`)

항목	TON (지연 ON)	TOF (지연 OFF)	TONR (누적 지연 ON)
누적 동작	없음	없음	있음
용도	일정 지연 후 작동	종료 후 지연	일정 누적 후 작동

✅ 5. 산업 예시

적용 예	사용 타이머	설명
냉방기 5초 후 작동	TON	전원 ON → 5초 지연 후 냉방 시작
경고등 3초 후 OFF	TOF	스위치 OFF 후에도 3초간 유지
경고 누적 10초 초과 시 알람	TONR	여러 조건이 반복되면 누적 시간 기준 경보 발생

✅ 6. 주의사항

- **TONR은 리셋 회로 반드시 필요**
- 타이머는 내부 메모리 자원을 사용하므로 **타이머 번호 중복 주의**
- 모든 타이머의 동작 단위는 보통 **1ms 또는 100ms 단위** (PLC 설정에 따라 다름)
- **실시간성이 중요한 회로에서는 타이머 정확도 오차 감안 필요**

✅ 정리

- TON: ON 지연 → **입력 후 일정 시간 지나야 ON**
- TOF: OFF 지연 → **입력 OFF 후 일정 시간 지나야 OFF**
- TONR: 누적 지연 → **누적 ON 시간 조건 충족 시 작동**

6.2 카운터(Counter) 제어 (CTU, CTD, RES)

(CTU, CTD, RES – 누적/감산/리셋을 통한 이벤트 수 제어)

✅ 개요

카운터(Counter)는 PLC에서 **특정 조건이 발생한 횟수를 셈**으로써
정해진 수 이상일 때만 동작하거나, 특정 수에 도달하면 동작을 멈추는 논리를 설계할 수 있게 한다.

> IEC 61131-3 표준에서 카운터는 다음 세 가지로 구성된다:

카운터 종류	의미	설명
CTU	Count Up	상승 카운터 (증가)
CTD	Count Down	하강 카운터 (감소)

카운터 종류	의미	설명
RES	Reset	카운터 초기화

✅ 1. CTU (Count Up)

📘 기능

- Rising Edge(입력 상승엣지)마다 카운트 증가
- 누적 값이 `PV(설정값)`에 도달하면 **Q 출력이 ON**

🧩 예시: 버튼 X0을 5번 누르면 램프 Y0 ON

```
[ X0 ] ──────( CTU C0, PV=5 )──────
                  Q ────[ Y0 ]
```

✍️ ST 코드

```
CTU_1(CU := X0, PV := 5);
Y0 := CTU_1.Q;
```

✅ 2. CTD (Count Down)

📘 기능

- Rising Edge마다 카운트 **감소**
- 누적 값이 0에 도달하면 **Q 출력이 ON**

🧩 예시: 재고 수량 3개 → 버튼 누를 때마다 차감 → 0이 되면 알림

```
[ X1 ] ──────( CTD C1, PV=3 )──────
                  Q ────[ Y1 ]
```

✍️ ST 코드

```
CTD_1(CD := X1, PV := 3);
Y1 := CTD_1.Q;
```

✅ 3. RES (Reset)

📘 기능

- CTU나 CTD의 **카운터 값을 0으로 초기화**
- 리셋 신호(R)가 ON되면 카운트 즉시 초기화

🧩 예시: 리셋 버튼 X2로 CTU 카운터 초기화

```
[ X2 ] ————————( RES C0 )
```

✍️ ST 코드

```
IF X2 THEN
    CTU_1(CU := FALSE);     // OFF edge
    CTU_1.CV := 0;
END_IF;
```

✅ 4. CTU + RES 통합 예시

```
[ X0 ] ————————( CTU C0, PV=3 )
[ X2 ] ————————( RES C0 )
```

- 버튼 X0을 3번 누르면 Y0 ON
- 리셋 버튼 X2 누르면 다시 0부터 시작

✅ 5. CTU & CTD 함께 사용 (Up/Down)

📘 예: 재고 증가/감소 시

```
[ 입고 X3 ] ——( CTU C2, PV=10 )
[ 출고 X4 ] ——( CTD C2, PV=10 )
```

- C2 공통 카운터 사용
- 입고 시 +1, 출고 시 -1 → 재고 추적 가능
- CV(Current Value)로 현재 수량 관리

✅ 6. 비교 요약

항목	CTU	CTD	RES
기능	카운트 증가	카운트 감소	카운터 초기화
조건	상승엣지	상승엣지	Rising edge

항목	CTU	CTD	RES
사용 예	버튼 클릭 수	출고 수량	카운트 리셋
출력 조건	CV ≥ PV	CV = 0	항상 즉시

✅ 7. 실전 활용 예

분야	설명
작업 횟수 카운트	버튼 누름, 조립 횟수 측정
생산품 수량 제어	일정 수량 달성 후 램프 점등
재고 관리	입출고에 따라 수량 가감
에러 발생 횟수	에러 횟수 누적 후 정지
고객 수 계산기	출입문 센서 기반 Up/Down

✅ 정리

- **CTU**: 상승엣지 → 누적 증가 → 설정값 도달 시 ON
- **CTD**: 감소 → 0 도달 시 ON
- **RES**: 초기화용 → CTU/CTD와 함께 사용
- **CV / PV** 값 주의해서 설계하고, 리셋 조건을 명확히 줄 것

6.3 시퀀스 제어 (단속운전, 순차제어)

(Step-by-Step 방식으로 진행되는 논리 흐름 제어)

✅ 개요

시퀀스 제어(Sequence Control)란,
정해진 순서에 따라 조건을 만족해야 다음 단계로 진행하는 방식의 제어다.
예를 들어, 기계를 단계적으로 시동하거나, 자동화 공정을 **1단계 → 2단계 → 3단계...** 식으로 **순차 수행**할 때 사용된다.

시퀀스 제어는 크게 두 가지로 나뉜다:

유형	설명
단속운전	버튼이 눌릴 때마다 한 단계씩 이동
순차제어	특정 조건(센서, 타이머 등)에 따라 단계 이동

✅ 1. 시퀀스 제어 핵심 개념

- 각 **스텝(Step)**은 하나의 상태
- 상태 전이는 **조건(입력, 타이머, 센서 등)**에 의해 발생
- **각 단계별 출력** 정의
- 마지막 단계 후 초기화 또는 반복 가능

✅ 2. 단속운전 예제

버튼을 누를 때마다 1 → 2 → 3단계로 진행 (Y0 → Y1 → Y2)

📘 래더 예제 구조

```
[ X0 ]──┐                    ( M0 ) ──────( Y0 )
        ├─( CTU C0, PV=3 )→  ( M1 ) ──────( Y1 )
[ RST ]─┘                    ( M2 ) ──────( Y2 )
```

- `C0.CV` 값에 따라 M0, M1, M2 출력
- M0: CV=0, M1: CV=1, M2: CV=2
- 리셋 버튼으로 C0 초기화

✍️ ST 예제

```
CTU_Seq(CU := X0, PV := 3);

M0 := CTU_Seq.CV = 0;
M1 := CTU_Seq.CV = 1;
M2 := CTU_Seq.CV = 2;

Y0 := M0;
Y1 := M1;
Y2 := M2;

IF RST THEN
    CTU_Seq(CU := FALSE);
    CTU_Seq.CV := 0;
END_IF;
```

✅ 3. 순차제어 예제

시작 → 작업1 → 작업2 → 작업3 → 종료

📘 예:

- X0: 시작 버튼
- X1: 작업1 완료 센서
- X2: 작업2 완료 센서
- Y0: 작업1 작동 출력
- Y1: 작업2 작동 출력
- Y2: 작업3 작동 출력

```
[ X0 ]──( M0 )
[ M0 ][ X1 ]──( M1 )
[ M1 ][ X2 ]──( M2 )

[ M0 ]──────( Y0 )   ← 작업1
[ M1 ]──────( Y1 )   ← 작업2
[ M2 ]──────( Y2 )   ← 작업3
```

✍️ ST 예제

```
IF X0 THEN M0 := TRUE; END_IF;
IF M0 AND X1 THEN M1 := TRUE; END_IF;
IF M1 AND X2 THEN M2 := TRUE; END_IF;

Y0 := M0;
Y1 := M1;
Y2 := M2;
```

✅ 4. 타이머 기반 순차제어 예

단계	출력	대기 시간
1단계	Y0	3초
2단계	Y1	5초
3단계	Y2	유지

래더 구조 (TON 사용)

```
[ X0 ]──( TON T0, PT=3s )→ Q → ( M0 )
[ M0 ]─( TON T1, PT=5s )→ Q → ( M1 )

[ M0 ]──( Y0 )
[ M1 ]──( Y1 )
[ M1 ]──( Y2 )
```

✅ 5. 시퀀스 제어 상태 다이어그램 예

```
[START] → [STEP1: Y0] → [STEP2: Y1] → [STEP3: Y2] → [RESET or HOLD]
              ↑             ↑             ↑
            조건1         조건2         조건3
```

✅ 6. 시퀀스 제어 설계 팁

팁	설명
단계별 내부 메모리(M0~)	각 단계 플래그로 사용
조건 명확화	센서 입력/시간 등 전이 조건 확정
리셋/재시작 논리 포함	예외 상황 대비
SFC 방식 추천	복잡한 순서 제어 시 상태기반으로 변환 설계 권장

✅ 7. 활용 사례

분야	설명
자동 조립기	단계별 삽입, 고정, 검사, 배출
세정장비	세정 → 헹굼 → 건조 순
포장기	투입 → 감지 → 밀봉 → 배출
공장 자동화 공정	정해진 순서대로 장비 제어

✅ 정리

- 시퀀스 제어는 **정해진 순서에 따라 논리 흐름을 이동**시키는 기법
- 단속운전은 버튼 등 **외부 트리거 기반**, 순차제어는 **자동 진행 조건 기반**
- 복잡한 시퀀스는 SFC 또는 스텝 메모리를 통해 **상태 기반 FSM처럼 관리**하는 방식이 효과적

6.4 반복 루프, 조건 분기

(PLC에서 반복 실행과 조건 분기 로직을 구현하는 핵심 기법)

✅ 개요

일반적인 프로그래밍 언어에는 `for`, `while`, `if`, `switch` 같은 **반복 및 조건 분기 구조**가 존재한다.
PLC 언어는 구조가 단순하지만, 동일한 기능을 **래더 다이어그램, ST, FBD 등으로 간접적으로 구현**할 수 있다.

PLC의 반복 및 조건 분기는 다음과 같이 정리된다:

개념	PLC 구현 방식
조건 분기	접점 조합 (`AND`, `OR`, `NOT`), 비교 명령, `IF` 문
반복 루프	카운터/타이머 기반 반복, 단계 시퀀스

✅ 1. 조건 분기 (IF / CASE)

📌 LD 방식

예제: X0가 ON이고, X1도 ON이면 Y0 출력

```
[ X0 ]──[ X1 ]─────( Y0 )
```

> 조건: `X0 AND X1`일 때 `Y0 ON`

📌 ST 방식

```
IF X0 AND X1 THEN
    Y0 := TRUE;
ELSE
    Y0 := FALSE;
END_IF;
```

📌 다중 조건 분기 (CASE)

```
CASE Mode OF
    0: Y0 := TRUE;
    1: Y1 := TRUE;
    2: Y2 := TRUE;
ELSE
    Y3 := TRUE;
END_CASE;
```

✅ 2. 반복 루프 (FOR / WHILE에 해당하는 구조)

PLC에는 **직접적인 루프 구조가 없음**
→ 대신 **카운터(CTU), 타이머(TON), 단계 전이(Mx 변수)**를 활용하여 구현

📌 예: 버튼 누를 때마다 10번 반복 동작

LD 구조

```
[ X0 ]──( CTU C0, PV=10 )────
C0.CV < 10 ──────( Y0 )
```

→ `X0` 누를 때마다 1씩 증가, `C0.CV`가 10 미만일 동안 `Y0` ON

ST 구조 (의사 코드 느낌)

```
CTU_1(CU := X0, PV := 10);

IF CTU_1.CV < 10 THEN
    Y0 := TRUE;
ELSE
    Y0 := FALSE;
END_IF;
```

📌 타이머 기반 반복 루프

예: 3초마다 모터를 5회 반복 ON/OFF

```
( TON T0, PT=3s ) → T0.Q → ( CTU C1, PV=5 )

C1.CV < 5 → ( Y_MOTOR )
```

- `T0`가 만료될 때마다 `C1` 증가
- `C1.CV`가 5보다 작으면 `Y_MOTOR` ON
- 도달하면 자동 정지

✅ 3. 조건 루프 종료 조건 설정

반복 중간에 조건이 맞지 않으면 중지하도록 **조건 분기 포함**

```
IF X0 = TRUE THEN
    CTU_1(CU := TRUE, PV := 10);
END_IF;

IF Sensor_Error THEN
    CTU_1.CV := 0;
    Y0 := FALSE;
END_IF;
```

✅ 4. 반복 루프 + 조건 분기 종합 예제

공장 라인 자동화:

- 시작 버튼 누르면 5회 반복
- 센서 이상이 감지되면 즉시 루프 중단

```
IF Start_Button THEN
    CTU_Loop(CU := TRUE, PV := 5);
END_IF;

IF CTU_Loop.CV < 5 AND NOT Sensor_Fail THEN
    Y_Motor := TRUE;
ELSE
    Y_Motor := FALSE;
END_IF;
```

✅ 5. 반복 루프를 상태 전이로 처리 (FSM 방식)

```
Step 0 → [시작 조건] → Step 1
Step 1 → [조건 만족] → Step 2
...
Step N → 초기화 or 완료
```

각 Step을 M0, M1, M2 등의 메모리 비트로 나타냄 → 상태 기반 반복 구조 구현

✅ 정리

- **조건 분기**는 `IF`, `CASE`, 접점 논리로 구현
- **반복 루프**는 타이머나 카운터로 반복 조건 제어
- 복잡한 반복은 **상태 기반 FSM 구조**로 설계하면 안정성 향상
- 조건 분기와 루프는 함께 사용하는 것이 일반적 (조건 → 반복 → 조건 → 반복...)

6.5 함수(Function), 서브루틴 사용

(PLC에서 코드의 재사용성과 구조화를 위한 핵심 기능)

✅ 개요

함수(Function)와 **서브루틴(Subroutine, 또는 Function Block)**은
PLC 프로그램을 **모듈화, 재사용, 유지보수 용이하게** 만들어주는 구조적 설계 요소다.

> IEC 61131-3에서 제공하는 주요 구조:

- **Function (함수)**: 입력 → 처리 → 단일 출력 (순수 함수 개념)
- **Function Block (FB)**: 상태 보유, 다중 입력/출력, 내부 변수 포함 (OOP 객체 느낌)

- **Program Organization Unit (POU)**의 일종

✅ 1. Function (함수)

📌 개념

- 입력을 받아 **처리를 수행하고 단일 출력을 반환**
- **상태를 가지지 않음** (Stateless)
- 동일 입력 → 동일 출력 보장 (순수 함수적)

📌 예제: 두 숫자 중 큰 값 반환

정의

```
FUNCTION MAX_OF_TWO : INT
VAR_INPUT
    A : INT;
    B : INT;
END_VAR

IF A > B THEN
    MAX_OF_TWO := A;
ELSE
    MAX_OF_TWO := B;
END_IF;
```

사용

```
Result := MAX_OF_TWO(10, 20);  // Result = 20
```

✅ 2. Function Block (FB, 함수 블록)

📌 개념

- 입력/출력 + 내부 상태 변수 포함 가능
- 호출 시 **인스턴스(instance)** 생성 → 독립적으로 상태 유지 가능
- 재사용 가능한 모듈 구성에 적합
- **TON, CTU, PID** 등의 내장 기능도 FB

📌 예제: ON 지연 타이머

```
FUNCTION_BLOCK ON_DELAY
VAR_INPUT
    IN  : BOOL;
    PT  : TIME;
END_VAR
VAR_OUTPUT
```

```
    Q   : BOOL;
END_VAR
VAR
    StartTime : TIME;
    Active    : BOOL;
END_VAR

IF IN AND NOT Active THEN
    StartTime := TIME(); // 현재 시간 기록
    Active := TRUE;
END_IF;

IF Active THEN
    IF TIME() - StartTime >= PT THEN
        Q := TRUE;
    END_IF;
END_IF;

IF NOT IN THEN
    Q := FALSE;
    Active := FALSE;
END_IF;
```

사용 (인스턴스화)

```
VAR
    Delay1 : ON_DELAY;
END_VAR

Delay1(IN := X0, PT := T#3s);
Y0 := Delay1.Q;
```

✅ 3. Subroutine (서브루틴) - 특정 PLC 제조사 용어

- **서브루틴**이라는 용어는 특정 PLC 브랜드(Siemens, Mitsubishi 등)에서
  프로그램 내 특정 로직을 따로 분리해 호출하는 기능으로 사용됨
- 일반적으로 **FC (Function)** 또는 **FB (Function Block)**으로 구현
- **CALL**, **RET**, **JMP** 명령으로 호출 가능

✅ 4. 구조 요약 비교

구분	Function	Function Block (FB)
상태 유지	❌ 없음 (Stateless)	✅ 있음
출력 개수	보통 1개	다중 출력 가능
재사용성	높음	매우 높음

구분	Function	Function Block (FB)
내부 변수	❌ 없음	✅ 있음
예시	수학 계산 함수	타이머, PID, 시퀀서 등

✅ 5. 활용 예시

이름	형태	설명
`ADD_INT`	Function	두 정수 덧셈
`TON`	FB	ON 딜레이 타이머
`MOTOR_SEQ`	FB	모터 순차 제어 로직
`ALARM_CHECK`	Function	에러 상태 판별

✅ 6. 사용자 정의 함수 활용 팁

- **중복되는 논리**를 함수로 빼면 가독성과 유지보수 용이
- 복잡한 제어는 FB로 묶어서 상태와 출력까지 통합 관리
- 변수 명은 **직관적이고 표준화된 규칙 사용** 권장

✅ 7. 실제 산업 활용

분야	활용
자동화 라인	순차 제어를 FB로 모듈화
빌딩 자동화	PID 제어 FB 사용
제조 설비	에러 판별, 데이터 처리 함수화
물류 시스템	트래킹 시퀀스 모듈화

✅ 정리

- **Function**: 간단한 계산, 판별용. Stateless.
- **Function Block**: 복잡한 상태 제어, 다중 입출력 로직.
- FB는 인스턴스를 생성해 **동시 독립 실행 가능**
- 반복되는 로직, 공용 로직은 반드시 함수화하여 유지보수성과 확장성 확보

6.6 데이터 블록(DB), 태그 기반 변수

(PLC에서의 데이터 저장 구조와 변수 관리 방식)

✅ 개요

PLC에서 프로그램이 사용하는 모든 데이터는 메모리에 저장되며,
이를 관리하고 구조화하기 위해 **데이터 블록(Data Block, DB)** 또는 **태그 기반 변수(Tag-based Variable)**를 사용한다.

> 제조사에 따라 용어 차이가 있지만 목적은 동일:

- **Siemens**: DB (Data Block)
- **Allen-Bradley (Rockwell)**: Tag-based variable
- **Mitsubishi**: 레지스터 또는 태그 주소
- **IEC 61131-3 표준**: 변수 선언 방식 제공

✅ 1. 데이터 블록 (DB)

📌 정의

Data Block(DB)은 PLC에서 사용되는 **데이터의 저장소**로,
하나의 프로그램 또는 Function Block에서 사용하는 변수들을 **묶어서 저장**할 수 있는 공간이다.

📌 종류

유형	설명
인스턴스 DB (Instance DB)	특정 Function Block의 개별 인스턴스에 연결됨
공유 DB (Global DB)	전역 데이터 저장 용도, 여러 프로그램에서 접근 가능
배열/구조형 DB	구조체, 배열 단위로 데이터를 묶어 저장

📘 Siemens 예: DB1 생성

```
// DB1: Motor 상태를 저장하는 DB
DB1:
    MotorStatus : BOOL;
    MotorSpeed  : INT;
    ErrorCode   : WORD;
```

- DB1.MotorStatus 등으로 접근
- 프로그램 간 공유 가능

✅ 2. 태그 기반 변수 (Tag-based Variable)

📌 개념

태그는 주소 기반(`M0.0`, `Q0.0`) 대신, **의미 있는 이름으로 정의된 변수**
→ 코드 가독성 향상, 유지보수 용이

📌 예시

```
StartButton : BOOL := %I0.0;      // 입력
MotorOutput : BOOL := %Q0.0;      // 출력
MotorSpeed  : INT  := %MW10;      // 워드 메모리
```

→ 이후 `StartButton`, `MotorOutput` 같은 이름만으로 제어 가능

✅ 3. 변수 종류와 스코프

변수 유형	설명	사용 예
전역(Global)	모든 프로그램에서 공유	생산량, 에러코드
지역(Local)	특정 프로그램/FB 내에서만 사용	내부 상태 저장
입력(Input)	외부 입력 신호	버튼, 센서
출력(Output)	외부 장치 제어	모터, 밸브
메모리(Memory)	중간 연산값 저장	M0.0, MW10 등

✅ 4. 데이터 타입

타입	설명	예
`BOOL`	참/거짓	스위치 상태
`INT`	정수형 (-32,768 ~ 32,767)	속도, 카운트
`DINT`	더 큰 정수형	누적 생산량
`REAL`	실수	온도, 거리
`TIME`	시간	타이머 설정값
`STRUCT`	구조체	복합 데이터 관리
`ARRAY`	배열	리스트 처리

✅ 5. 구조체 (STRUCT)와 배열 (ARRAY)

📌 구조체

```
TYPE MotorInfo :
STRUCT
    Status : BOOL;
    Speed  : INT;
    Temp   : REAL;
END_STRUCT
END_TYPE;
```

→ `MotorInfo1.Speed`, `MotorInfo2.Status`처럼 개별 제어

📌 배열

```
VAR
    SensorValues : ARRAY[1..5] OF INT;
```

→ 반복 처리, 그룹 데이터 처리에 적합

✅ 6. 변수 선언 방식 (ST 기준)

```
VAR_GLOBAL
    SystemReady : BOOL;
    TotalCount  : DINT;
END_VAR

VAR
    LocalTemp   : REAL;
    Timer1      : TON;
END_VAR
```

✅ 7. 데이터 블록과 FB 연결 (Siemens 기준)

```
FB1: 온도 제어 로직
→ DB1: FB1에 대한 인스턴스 데이터 저장소
```

- FB1 호출 시 DB1이 연결되어 해당 FB 내부 상태를 기억

✅ 8. 실제 예시: 모터 제어용 DB 구성

변수명	타입	설명
`Motor.Status`	`BOOL`	모터 동작 여부
`Motor.Speed`	`INT`	현재 속도
`Motor.Temp`	`REAL`	온도
`Motor.AlarmCode`	`WORD`	에러 코드

→ 이를 DB나 태그로 선언하여 유지보수성 향상

✅ 9. 설계 시 팁

팁	설명
태그명은 **기능 중심으로 명확하게**	`X0` → `StartButton`, `M1.0` → `MotorEnable`
구조체로 **관련 데이터 그룹화**	`MotorInfo`, `SensorGroup`, `LineStatus`
배열 사용 시 **범위 주의**	인덱스 초과 방지
DB는 **논리적 기능별 분리**	`DB_Alarm`, `DB_Production`, `DB_Config` 등

✅ 정리

- **DB (Data Block)**: PLC 내 데이터 저장소, 인스턴스별 상태 기억 가능
- **태그 변수**: 가독성 높은 의미 기반 변수 → 주소 기반보다 유지보수 우수
- **구조체, 배열, 타입 선언**으로 복잡한 데이터를 체계적으로 관리
- **POU + DB 설계 전략**으로 고급 모듈화 가능

6.7 Shift Register 제어, 패턴 출력

(연속된 이력 저장과 시퀀스 제어를 위한 핵심 기법)

✅ 개요

Shift Register(시프트 레지스터)는 입력 신호를 순차적으로 이동시키며 저장하는 구조로,
PLC에서는 주로 **이력 저장, 이동 제어, 패턴 출력, 순차 조명, 컨베이어 추적** 등에 활용된다.

예를 들어, 센서가 감지한 물체의 위치를 트래킹하거나, LED를 순차적으로 켜는 로직에 사용된다.

✅ 1. Shift Register의 작동 원리

```
[신호 입력] → [M0] → [M1] → [M2] → ... → [Mn]
```

- 새로운 입력이 들어오면, 기존 값들이 오른쪽으로 밀리고 마지막 값은 사라짐
- **FIFO(First-In First-Out)** 형태의 비트 이동 구조

✅ 2. 일반적인 응용 사례

용도	설명
컨베이어 물체 감지	센서 감지 후 위치 추적
순차 LED 출력	LED를 한 칸씩 밀면서 점등
이력 기반 제어	N번 전 상태를 참조한 제어
타겟 도달 시 작동	N단계 뒤의 작업(분사, 로봇 작동 등)

✅ 3. 래더 예제 (비트 이동)

예: 센서 X0가 감지되면 M0~M4로 Shift

```
[X0] ──(SHL M0, 5비트) ──> [M0][M1][M2][M3][M4]
```

- `SHL`: Shift Left
- 감지된 입력은 M0 → M1 → M2 ...로 1주기마다 이동

✅ 4. ST 언어 예제

```
VAR
    SR : ARRAY[0..4] OF BOOL;
    INP : BOOL;
END_VAR

IF RisingEdge(INP) THEN
    SR[4] := SR[3];
    SR[3] := SR[2];
    SR[2] := SR[1];
    SR[1] := SR[0];
    SR[0] := TRUE;     // 현재 입력을 SR[0]에 저장
END_IF;
```

✅ 5. 패턴 출력 (LED 이동)

📌 예제: 4개의 LED가 순차적으로 1초 간격으로 점등

```
패턴: 0001 → 0010 → 0100 → 1000 → 반복
pascal코드 복사VAR
    LED : BYTE := 1;
    T : TON;
END_VAR

T(IN := TRUE, PT := T#1s);

IF T.Q THEN
    LED := SHL(LED, 1);         // 왼쪽으로 시프트
    IF LED > 8 THEN
        LED := 1;               // 패턴 반복
    END_IF;
    T(IN := FALSE);             // 타이머 리셋
END_IF;

Y0 := (LED AND 1) <> 0;
Y1 := (LED AND 2) <> 0;
Y2 := (LED AND 4) <> 0;
Y3 := (LED AND 8) <> 0;
```

✅ 6. 실제 활용 예: 컨베이어와 에어분사

- X0: 센서 (물체 감지)
- 5초 뒤 Y0: 에어분사

```
시간 지연 → Shift 5단계 (5초)
```

```
IF X0 RisingEdge THEN
    SR[0] := TRUE;
END_IF;

FOR i := 4 TO 1 BY -1 DO
    SR[i] := SR[i-1];
END_FOR;

SR[0] := FALSE;  // 새로운 입력이 없으면 초기화

Y0 := SR[4];  // 5단계 후 출력
```

✅ 7. 패턴 생성 팁

구조	활용
비트 배열	패턴 로직 표현
시프트 연산 (SHL, SHR)	이동 효과
MASK 연산 (AND)	특정 출력 추출
상태 기억용 메모리	각 단계 기록 및 조건 분기

✅ 8. 고급 활용

기능	예
타겟 추적	작업물 이동 위치 추적
비트 패턴 매칭	특정 조건일 때만 동작 (예: `01010`)
고속 시프트	32비트 이상 비트 연산 가능 (DINT 등 사용)

✅ 정리

- **Shift Register**는 상태 이력, 연속 동작, 추적 제어에 필수적
- 타이머, 비트 연산, 조건 분기와 함께 사용하면 매우 강력한 시퀀스 로직 구성 가능
- **순차 제어, 추적 자동화, 디지털 패턴 생성** 등에 핵심적으로 활용